

1 Capstone Project

1.1 Notebook 2: Exploratory Data Analysis, Data Preprocessing, Modeling

- Tia Plagata
- <https://github.com/tiaplagata/capstone-project> (<https://github.com/tiaplagata/capstone-project>)
- tiaplagata@gmail.com (<mailto:tiaplagata@gmail.com>).

```
In [102]: 1 # Import Statements
2
3 import pandas as pd
4 import numpy as np
5
6 import matplotlib.pyplot as plt
7 import seaborn as sns
8 import plotly.express as px
9
10 import string
11 import regex as re
12 import spacy
13
14 from nltk.corpus import stopwords
15 # nltk.download('stopwords')
16 # nltk.download('punkt')
17 from nltk import word_tokenize
18 from nltk import FreqDist
19
20 import warnings
21 warnings.filterwarnings('ignore')
22
23 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
24 from sklearn.model_selection import train_test_split
25 from sklearn.naive_bayes import MultinomialNB, GaussianNB
26 from sklearn.ensemble import RandomForestClassifier
27 from sklearn.metrics import recall_score, accuracy_score, f1_score, confusion_matrix
28 from sklearn.utils import class_weight
29 from sklearn.pipeline import Pipeline
30 from sklearn.base import TransformerMixin
31 from sklearn import set_config
32
33 from PIL import Image
34 from wordcloud import WordCloud
35 from textwrap import wrap
36
37 import pickle
```

executed in 13ms, finished 09:37:49 2021-01-19

In [103]:

```
1 # Read in the DataFrame I created in the Data Collection notebook
2 df = pd.read_csv('/Users/tiaplagata/Documents/Flatiron/capstone-project')
3 df.head()
```

executed in 41ms, finished 09:37:49 2021-01-19

Out[103]:

	Attraction	City
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom
1	The Jack The Ripper Walking Tour in London	London, United Kingdom
2	Ghost Bus Tour of London	London, United Kingdom
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom
4	The Blood and Tears Walk: Serial Killers and L...	London, United Kingdom

▼ 1.2 Explore/clean the data

- Decide whether or not to get rid of duplicates
- Check out the class imbalance
- *No need to worry about null values because I scraped this dataset myself*

In [104]:

```
1 df.info()
```

executed in 8ms, finished 09:37:49 2021-01-19

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 28379 entries, 0 to 3693
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  --  
 0   Attraction   28379 non-null   object 
 1   City         28379 non-null   object 
dtypes: object(2)
memory usage: 665.1+ KB
```

In [105]:

```
1 df.describe()
```

executed in 23ms, finished 09:37:49 2021-01-19

Out[105]:

	Attraction	City
count	28379	28379
unique	27466	12
top	Desert Safari Dubai	Bali, Indonesia
freq	15	5000

```
In [106]: 1 df.shape
```

executed in 3ms, finished 09:37:49 2021-01-19

Out[106]: (28379, 2)

```
In [107]: 1 #No null values because I scraped everything myself. Just to double-check
2 df.isna().sum()
```

executed in 5ms, finished 09:37:49 2021-01-19

Out[107]: Attraction 0
City 0
dtype: int64

▼ 1.2.1 Duplicates

```
In [108]: 1 df.duplicated().sum()
```

executed in 14ms, finished 09:37:49 2021-01-19

Out[108]: 846

In [109]:

```

1 # Look at duplicates in one city
2 df[(df.duplicated()==True) & (df['City']=='London, United Kingdom')]

```

executed in 19ms, finished 09:37:49 2021-01-19

Out[109]:

	Attraction	City
185	Windsor Castle, Stonehenge, and Oxford Day Tri...	London, United Kingdom
241	Jack the Ripper Walking Tour in London	London, United Kingdom
259	British Museum Guided Tour	London, United Kingdom
618	Private Transfer from Heathrow Airport to London	London, United Kingdom
704	Oxford City Full-Day Private Tour from London	London, United Kingdom
705	Bath and Stonehenge Full-Day Private Tour from...	London, United Kingdom
706	Full-Day Private Guided Tour of Cambridge	London, United Kingdom
716	Full-Day Private Tour of Brighton	London, United Kingdom
900	PRIVATE Jack the Ripper Ghost Walking Tour in ...	London, United Kingdom
990	London to Southampton Cruise Terminals Private...	London, United Kingdom
1074	Liverpool the Beatles Legend Fab Four and Manc...	London, United Kingdom
1365	Private Full-Day Tour of Shakespeare's Stratfo...	London, United Kingdom
1367	Bournemouth and Durdle Door Jurassic Full Day ...	London, United Kingdom
1369	Full-Day Private Tour to the Historic Naval Ci...	London, United Kingdom
1468	Full-Day Private Fun Cultural Guided Tour of L...	London, United Kingdom
1469	London Royal's Full Day Tour	London, United Kingdom
1472	Changing of the Guard Half-Day Private Walking...	London, United Kingdom
1473	3-Hour Guided Tour of Science Museum in London	London, United Kingdom
1493	London's City Lights by Night Private Tour	London, United Kingdom
1494	Theme Parks of London Chessington Full-Day Pri...	London, United Kingdom
1502	J.R.R. Tolkien's Oxford and Stonehenge Private...	London, United Kingdom
1504	Private Layover Tour from London City Airport	London, United Kingdom
1516	London Full-Day Private Shore Excursion from S...	London, United Kingdom
1517	2-Day Private Wales Tour to Cardiff and Aberfa...	London, United Kingdom
1522	London Full Day Private Tour by Walking and Pu...	London, United Kingdom
1536	Oxford City and Cotswolds Private Tour	London, United Kingdom
1537	Salisbury Magna Carta Stonehenge and Bath Priv...	London, United Kingdom
1542	The Golden Triangle Tour London-Oxford-Cambr...	London, United Kingdom
1606	London Skyline Tour	London, United Kingdom
1612	Wimbledon Tennis and Museum Tour	London, United Kingdom
1613	London Shopping Experience Tour	London, United Kingdom
1622	Freestyle Football Workshop in England	London, United Kingdom

	Attraction	City
1675	The Crown Netflix TV London Half Day Private Tour	London, United Kingdom
1679	007 James Bond's London Private Half Day Tour	London, United Kingdom
1732	4 Hour Tour Harry Potter Locations In London (...)	London, United Kingdom
1812	Private Chauffeured Minivan at Your Disposal i...	London, United Kingdom
1846	Canterbury Cathedral and Leeds Castle Private ...	London, United Kingdom
1881	Windsor Castle Heathrow Airport Private Layover	London, United Kingdom
1882	Young Victoria's London: Windsor Castle & Kens...	London, United Kingdom
1920	9Hr Tour London Eye, Westminster Abbey and St ...	London, United Kingdom
1930	Essential London Full-Day Private Tour by Publ...	London, United Kingdom
1952	Heathrow Airport Transfer	London, United Kingdom
1961	Sherlock Holmes Walking Tour in London	London, United Kingdom
1974	Royal London Walking Tour	London, United Kingdom
2074	1066 Battle of Hastings, Birling Gap and Seven...	London, United Kingdom
2120	Full Day Traditional Private London Tour by Wa...	London, United Kingdom
2140	Zoom online tour of London	London, United Kingdom
2230	London Underground 2-Hour Tube Tour	London, United Kingdom
2282	London to Southampton Cruise Terminals Private...	London, United Kingdom
2285	Departure Private Transfers from London City t...	London, United Kingdom
2291	4 Hour Tour Tower of London and St Pauls Cathe...	London, United Kingdom
2304	Warner Bros' Making of Harry Potter Studio Tour	London, United Kingdom
2342	Arrival Private Transfers from London Railway ...	London, United Kingdom
2344	Beautiful Cornwall Two Days Private Tour	London, United Kingdom
2350	Jack the Ripper Mystery Walks	London, United Kingdom
2562	4 Hour Tour London Highlights with Private To...	London, United Kingdom
2618	The London Landmarks	London, United Kingdom
2735	Afternoon tea bus tour in London	London, United Kingdom
2772	Full Day London Pick & Mix Customized Tour	London, United Kingdom
2773	A Day at the Museum - Natural History Museum L...	London, United Kingdom

In [110]:

```
1 df[df['Attraction']=='Oxford City Full-Day Private Tour from London']
```

executed in 6ms, finished 09:37:49 2021-01-19

Out[110]:

	Attraction	City
431	Oxford City Full-Day Private Tour from London	London, United Kingdom
704	Oxford City Full-Day Private Tour from London	London, United Kingdom

In [111]:

```
1 # What about the top attraction?
2 df[df['Attraction']=='Desert Safari Dubai']
```

executed in 6ms, finished 09:37:49 2021-01-19

Out[111]:

	Attraction	City
414	Desert Safari Dubai	Dubai, United Arab Emirates
478	Desert Safari Dubai	Dubai, United Arab Emirates
811	Desert Safari Dubai	Dubai, United Arab Emirates
974	Desert Safari Dubai	Dubai, United Arab Emirates
998	Desert Safari Dubai	Dubai, United Arab Emirates
1001	Desert Safari Dubai	Dubai, United Arab Emirates
1689	Desert Safari Dubai	Dubai, United Arab Emirates
1718	Desert Safari Dubai	Dubai, United Arab Emirates
1722	Desert Safari Dubai	Dubai, United Arab Emirates
1944	Desert Safari Dubai	Dubai, United Arab Emirates
2301	Desert Safari Dubai	Dubai, United Arab Emirates
2514	Desert Safari Dubai	Dubai, United Arab Emirates
2854	Desert Safari Dubai	Dubai, United Arab Emirates
3101	Desert Safari Dubai	Dubai, United Arab Emirates
3426	Desert Safari Dubai	Dubai, United Arab Emirates

Clearly, I need to remove duplicates here, because there are some exact duplicates for certain cities.

In [112]:

```
1 df = df.drop_duplicates()
```

executed in 14ms, finished 09:37:49 2021-01-19

In [113]:

```
1 # df.to_csv('/Users/tiaplagata/Documents/Flatiron/capstone-project/Data')
```

executed in 2ms, finished 09:37:49 2021-01-19

▼ 1.2.2 Class Imbalance

In [114]:

```
1 display(df.City.unique())
2 print('Total Unique Cities:', len(df.City.unique()))
```

executed in 5ms, finished 09:37:49 2021-01-19

```
array(['London, United Kingdom', 'Paris, France', 'Crete, Greece',
       'Bali, Indonesia', 'Rome, Italy', 'Phuket, Thailand',
       'Sicily, Italy', 'Majorca, Balearic Islands', 'Barcelona, Spain',
       'Istanbul, Turkey', 'Goa, India', 'Dubai, United Arab Emirates'],
      dtype=object)
```

Total Unique Cities: 12

In [115]:

```
1 df.City.value_counts(normalize=True)
```

executed in 6ms, finished 09:37:49 2021-01-19

Out[115]:

Bali, Indonesia	0.177823
Rome, Italy	0.174990
Dubai, United Arab Emirates	0.123597
London, United Kingdom	0.099626
Paris, France	0.096938
Istanbul, Turkey	0.081248
Sicily, Italy	0.071986
Barcelona, Spain	0.066502
Phuket, Thailand	0.041260
Crete, Greece	0.037010
Majorca, Balearic Islands	0.016489
Goa, India	0.012530

Name: City, dtype: float64

In [116]:

```
1 cities = df.groupby('City').count()
```

executed in 6ms, finished 09:37:49 2021-01-19

In [117]:

```
1 cities.reset_index(inplace=True)
```

executed in 2ms, finished 09:37:49 2021-01-19

In [118]:

```
1 sorted_cities = cities.sort_values(by='Attraction', ascending=False)
2 sorted_cities
```

executed in 5ms, finished 09:37:49 2021-01-19

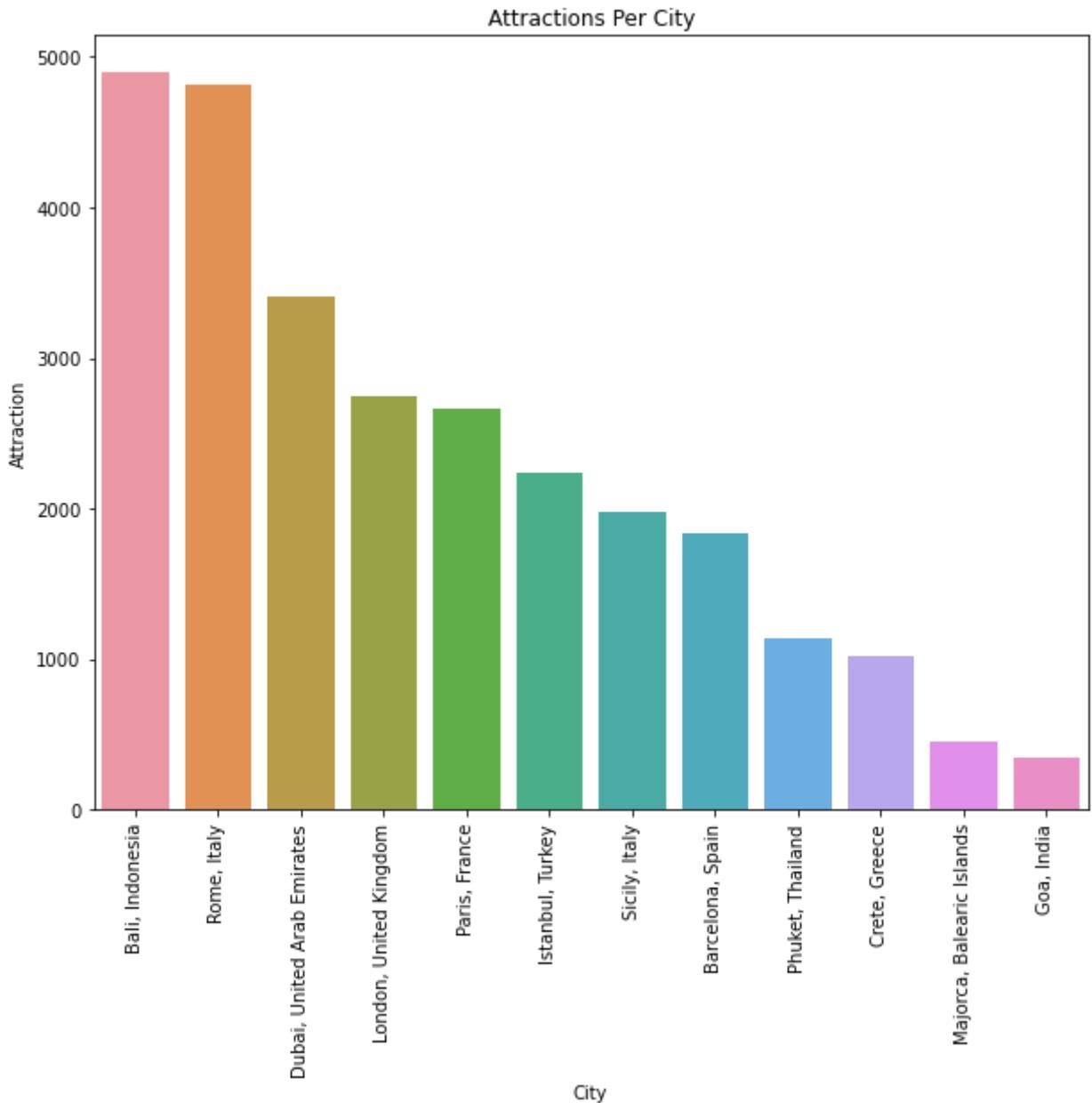
Out[118]:

	City	Attraction
0	Bali, Indonesia	4896
10	Rome, Italy	4818
3	Dubai, United Arab Emirates	3403
6	London, United Kingdom	2743
8	Paris, France	2669
5	Istanbul, Turkey	2237
11	Sicily, Italy	1982
1	Barcelona, Spain	1831
9	Phuket, Thailand	1136
2	Crete, Greece	1019
7	Majorca, Balearic Islands	454
4	Goa, India	345

In [119]:

```
1 # Plot the class imbalance
2 plt.figure(figsize=(10,8))
3 sns.barplot(x='City', y='Attraction', data=sorted_cities)
4 plt.title('Attractions Per City')
5 plt.xticks(rotation=90)
6 plt.show()
```

executed in 123ms, finished 09:37:49 2021-01-19



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

▼ 1.3 Text Cleaning & Preprocessing & More Exploration

- Remove punctuation and numbers
- Lowercase everything
- Remove stopwords
- Create a document term matrix grouped by city
 - count vectorization
 - tf-idf vectorization
 - bi-grams
- Visualize most frequent words
 - word clouds
 - bar plot/histogram

```
In [120]: 1 # Create a list of stopwords
           2 stopwords_list = stopwords.words('english')
           3 stopwords_list += list(string.punctuation)
```

executed in 4ms, finished 09:37:49 2021-01-19

```
In [121]: 1 # Preview the list
           2 stopwords_list[:10]
```

executed in 3ms, finished 09:37:49 2021-01-19

```
Out[121]: ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "yo
u're"]
```

In [122]:

```

1 # Lowercase all words in each corpus
2 df['cleaned'] = df['Attraction'].apply(lambda x: x.lower())
3 df.head()

```

executed in 14ms, finished 09:37:49 2021-01-19

Out[122]:

	Attraction	City	cleaned
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom	sea life london aquarium admission ticket
1	The Jack The Ripper Walking Tour in London	London, United Kingdom	the jack the ripper walking tour in london
2	Ghost Bus Tour of London	London, United Kingdom	ghost bus tour of london
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom	big bus london hop-on hop-off tour and river c...
4	The Blood and Tears Walk: Serial Killers and L...	London, United Kingdom	the blood and tears walk: serial killers and l...

In [123]:

```

1 # Remove commas, hyphens, colons, and other punctuation
2 df['cleaned'] = df['cleaned'].apply(lambda x: re.sub('[%s]' % re.escape(
3 df.head()

```

executed in 267ms, finished 09:37:49 2021-01-19

Out[123]:

	Attraction	City	cleaned
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom	sea life london aquarium admission ticket
1	The Jack The Ripper Walking Tour in London	London, United Kingdom	the jack the ripper walking tour in london
2	Ghost Bus Tour of London	London, United Kingdom	ghost bus tour of london
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom	big bus london hopon hopoff tour and river cru...
4	The Blood and Tears Walk: Serial Killers and L...	London, United Kingdom	the blood and tears walk serial killers and lo...

In [124]:

```

1 # Use regex to get rid of numbers
2 df['cleaned'] = df['cleaned'].apply(lambda x: re.sub('\w*\d\w*', '', x))
3 df.head(10)

```

executed in 319ms, finished 09:37:49 2021-01-19

Out[124]:

	Attraction	City	cleaned
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom	sea life london aquarium admission ticket
1	The Jack The Ripper Walking Tour in London	London, United Kingdom	the jack the ripper walking tour in london
2	Ghost Bus Tour of London	London, United Kingdom	ghost bus tour of london
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom	big bus london hopon hopoff tour and river cru...
4	The Blood and Tears Walk: Serial Killers and L...	London, United Kingdom	the blood and tears walk serial killers and lo...
5	London Ghost and Infamous Murders Walking Tour	London, United Kingdom	london ghost and infamous murders walking tour
6	Stonehenge, Windsor Castle, and Bath from London	London, United Kingdom	stonehenge windsor castle and bath from london
7	Warner Bros. Studio: The Making of Harry Potte...	London, United Kingdom	warner bros studio the making of harry potter ...
8	Ghosts, Ghouls & Gallows: London Virtual Tour	London, United Kingdom	ghosts ghouls gallows london virtual tour
9	High-Speed Thames River RIB Cruise in London	London, United Kingdom	highspeed thames river rib cruise in london

In [125]:

```
1 # !python -m spacy download en
```

executed in 1ms, finished 09:37:49 2021-01-19

In [126]:

```

1 # Lemmatize the text using spacy
2 nlp = spacy.load('en')
3
4 df['lemmatized'] = df['cleaned'].apply(lambda x: ' '.join(
5                                     [token.lemma_ for token in list(nlp
6 df.head(10)

```

executed in 1m 43.7s, finished 09:39:33 2021-01-19

Out[126]:

	Attraction	City	cleaned	lemmatized
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom	sea life london aquarium admission ticket	sea life london aquarium admission ticket
1	The Jack The Ripper Walking Tour in London	London, United Kingdom	the jack the ripper walking tour in london	jack ripper walking tour london
2	Ghost Bus Tour of London	London, United Kingdom	ghost bus tour of london	ghost bus tour london
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom	big bus london hopon hopoff tour and river cru...	big bus london hopon hopoff tour river cruise ...
4	The Blood and Tears Walk: Serial Killers and L...	London, United Kingdom	the blood and tears walk serial killers and lo...	blood tear walk serial killer london horror
	London Ghost and Infamous	London,	london ghost and infamous	london ghost infamous

In [127]:

```

1 # Group the corpora by city and join them
2 df_to_group = df[['City', 'lemmatized']]
3 df_grouped = df_to_group.groupby(by='City').agg(lambda x: ' '.join(x))
4 df_grouped

```

executed in 27ms, finished 09:39:33 2021-01-19

Out[127]:

City	lemmatized
Bali, Indonesia	hotel hotelbali private transfer daytime bali ...
Barcelona, Spain	interactive spanish cooking experience barcelo...
Crete, Greece	minoans world museum cinema crete wine ol...
Dubai, United Arab Emirates	premium red dune camel safari bbq al khayma ...
Goa, India	fontainhas heritage walk sunset cruise paradis...
Istanbul, Turkey	bosphorus sunset cruise luxury yacht istanbu...
London, United Kingdom	sea life london aquarium admission ticket jack...
Majorca, Balearic Islands	cave genova admission palma de mallorca shore ...
Paris, France	bateaux parisiens seine river gourmet dinner ...
Phuket, Thailand	phi phi maiton khai islands speedboat phi ph...
Rome, Italy	fast skiptheline vatican sistine chapel st pet...
Sicily, Italy	etna taormina fullday tour catania palermo str...



1.3.1 Look at different vectorization strategies

- Try different vectorization strategies and visualize them with word clouds
 - count vectorization
 - tf-idf vectorization
 - bi-grams

In [128]:

```

1 # Create a document term matrix using count vectorization
2 # Using count vectorization (most simple way to vectorize)
3 cv = CountVectorizer(analyzer='word', stop_words=stopwords_list)
4 data = cv.fit_transform(df_grouped['lemmatized'])
5 df_dtm = pd.DataFrame(data.toarray(), columns=cv.get_feature_names())
6 df_dtm.index = df_grouped.index
7 df_dtm

```

executed in 135ms, finished 09:39:33 2021-01-19

Out[128]:

	aal	abandon	abant	abba	abbate	abbey	abbeyprivate	abbeyst	aberfan	abian	...
City											
Bali, Indonesia	0	2	0	0	0	0	0	0	0	0	1 ...
Barcelona, Spain	1	0	0	0	0	0	0	0	0	0	0 ...
Crete, Greece	0	0	0	0	0	0	0	0	0	0	0 ...
Dubai, United Arab Emirates	0	1	0	0	0	0	0	0	0	0	0 ...
Goa, India	0	0	0	0	0	0	0	0	0	0	0 ...
Istanbul, Turkey	0	0	1	0	0	0	0	0	0	0	0 ...
London, United Kingdom	0	0	0	1	0	61	1	2	1	0	0 ...
Majorca, Balearic Islands	0	0	0	0	1	0	0	0	0	0	0 ...
Paris, France	0	0	0	0	0	5	0	0	0	0	0 ...
Phuket, Thailand	0	0	0	0	0	0	0	0	0	0	0 ...
Rome, Italy	0	0	0	0	0	3	0	0	0	0	0 ...
Sicily, Italy	0	0	0	0	0	0	0	0	0	0	0 ...

12 rows × 8676 columns

In [129]:

```

1 # Create a document term matrix using TF-IDF vectorization
2 # Might be good for classifying cities
3 tfidf = TfidfVectorizer(analyzer='word', stop_words=stopwords_list)
4 data2 = tfidf.fit_transform(df_grouped['lemmatized'])
5 df_dtm2 = pd.DataFrame(data2.toarray(), columns=tfidf.get_feature_names)
6 df_dtm2.index = df_grouped.index
7 df_dtm2

```

executed in 124ms, finished 09:39:33 2021-01-19

Out[129]:

City	aal	abandon	abant	abba	abbate	abbey	abbeyprivate	abbeyst	aberfan
Bali, Indonesia	0.000000	0.000699	0.000000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Barcelona, Spain	0.000737	0.000000	0.000000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Crete, Greece	0.000000	0.000000	0.000000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Dubai, United Arab Emirates	0.000000	0.000310	0.000000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Goa, India	0.000000	0.000000	0.000000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Istanbul, Turkey	0.000000	0.000000	0.00059	0.0000	0.000000	0.000000	0.0000	0.000	0.0000

Word Clouds with Count Vectorization

In [130]:

```

1 def generate_wordcloud(data, title):
2     cloud = WordCloud(width=400, height=330, max_words=150, colormap='t'
3     plt.figure(figsize=(10,8))
4     plt.imshow(cloud, interpolation='bilinear')
5     plt.axis('off')
6     plt.title('\n'.join(wrap(title,60)), fontsize=13)
7     plt.show()

```

executed in 2ms, finished 09:39:33 2021-01-19

```
In [131]: 1 # Transposing document term matrix
2 df_dtm = df_dtm.transpose()
3
4 # Plotting word cloud for each city
5 for index, city in enumerate(df_dtm.columns):
6     generate_wordcloud(df_dtm[city].sort_values(ascending=False), city)
```

executed in 2.92s, finished 09:39:36 2021-01-19

Bali, Indonesia



```
In [132]: 1 # Look at top words with count vectorizer (in total, not per city)
2 sum_words = data.sum(axis=0)
3 words_freq = [(word, sum_words[0, idx]) for word, idx in cv.vocabulary_
4 words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
5 words freq
```

executed in 34ms. finished 09:39:36 2021-01-19

```
Out[132]: [('tour', 13522),  
           ('private', 8730),  
           ('transfer', 3525),  
           ('day', 3518),  
           ('airport', 2940),  
           ('rome', 2822),  
           ('bali', 2359),  
           ('dubai', 2331),  
           ('city', 2131),  
           ('london', 2078),  
           ('paris', 1872),  
           ('guide', 1604),  
           ('istanbul', 1471),  
           ('barcelona', 1244),  
           ('trip', 1153),  
           ('safari', 1056),  
           ('dinner', 992),  
           ('walk', 990),  
           ('desert', 967),  
           ('beach', 951)]
```

There seems to be a lot of repetition with words like 'tour', 'private', 'transfer' between cities

because all of these cities will have airport transfers, and different tours. However, these words will not help in modeling since they are not unique to the cities.

Word Clouds with TF-IDF Vectorization

```
In [133]: 1 # Transposing document term matrix
2 df_dtm2 = df_dtm2.transpose()
3
4 # Plotting word cloud for each city
5 for index, city in enumerate(df_dtm2.columns):
6     generate_wordcloud(df_dtm2[city].sort_values(ascending=False), city)
```

executed in 2.93s, finished 09:39:39 2021-01-19



In [134]:

```

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

```

executed in 33ms, finished 09:39:39 2021-01-19

Out[134]:

```

[( 'tour', 3.1397055370950477),
 ('private', 2.0840633636356287),
 ('transfer', 0.9685547196506898),
 ('goa', 0.9226286509898938),
 ('barcelona', 0.9166731088394917),
 ('london', 0.8932185752670296),
 ('paris', 0.8836205705514917),
 ('istanbul', 0.8682353908231614),
 ('airport', 0.857539740811609),
 ('dubai', 0.8422028010450787),
 ('day', 0.8310517134188751),
 ('phuket', 0.8048581266769077),
 ('mallorca', 0.7500827249467662),
 ('bali', 0.7372449047082821),
 ('rome', 0.6645617714771512),
 ('palma', 0.5056361226203647),
 ('crete', 0.47454872097902234),
 ('palermo', 0.469178133177727),
 ('city', 0.46673456115636947),
 ...]

```

In contrast, there is not as much overlap with these words as in the count vectorization because tf-idf vectorization is finding more words that are unique to the cities. This tells us that tf-idf vectorization is probably a better vectorization technique to use while modeling in order to best predict the cities.



Word Clouds with Bi-Grams

In [135]:

```

1 cv2 = CountVectorizer(analyzer='word', stop_words=stopwords_list, ngram
2 data3 = cv2.fit_transform(df_grouped['lemmatized'])
3 df_dtm3 = pd.DataFrame(data3.toarray(), columns=cv2.get_feature_names())
4 df_dtm3.index = df_grouped.index
5 df_dtm3

```

executed in 293ms, finished 09:39:40 2021-01-19

Out[135]:

	aal	abandon	abandon	abandon	abant	abba	abbate	abbey	abbey	a
City	deep	ghost	hotel	village	yedigoller	sup	arrival	avebury	banquet	buckin
Bali, Indonesia	0	1	1	0	0	0	0	0	0	0
Barcelona, Spain	1	0	0	0	0	0	0	0	0	0
Crete, Greece	0	0	0	0	0	0	0	0	0	0
Dubai, United Arab Emirates	0	0	0	1	0	0	0	0	0	0
Goa, India	0	0	0	0	0	0	0	0	0	0
Istanbul,	0	0	0	0	1	0	0	0	0	0

In [136]:

```

1 # Transposing document term matrix
2 df_dtm3 = df_dtm3.transpose()
3
4 # Plotting word cloud for each city
5 for index, city in enumerate(df_dtm3.columns):
6     generate_wordcloud(df_dtm3[city].sort_values(ascending=False), city)

```

executed in 3.36s, finished 09:39:43 2021-01-19

In [137]:

```
1 # Look at top bi-grams (in total, not per city)
2 sum_words = data3.sum(axis=0)
3 words_freq = [(word, sum_words[0, idx]) for word, idx in cv2.vocabulary]
4 words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
5 words_freq
```

executed in 72ms, finished 09:39:43 2021-01-19

Out[137]:

```
[('private tour', 2228),
 ('private transfer', 1569),
 ('day tour', 824),
 ('city tour', 819),
 ('tour private', 813),
 ('desert safari', 765),
 ('walk tour', 694),
 ('guide tour', 628),
 ('day trip', 589),
 ('small group', 573),
 ('tour rome', 564),
 ('skip line', 513),
 ('airport transfer', 472),
 ('half day', 465),
 ('abu dhabi', 426),
 ('tour london', 423),
 ('tour istanbul', 392),
 ('private day', 386),
 ('tour bali', 350),
 ...]
```

The bi-grams were able to pick out important terms, such as 'windsor castle' for London, and 'cooking class' for Sicily. However, words like 'tour' are creating some noise in most of these cities.



1.4 Removing Noise from the Data

Since there are still lots of words in the word clouds like 'private', 'airport' and 'transfer', I want to try to take those attractions for airport transfers out because they are causing noise in the data.

In [138]:

1 df.head()

executed in 5ms, finished 09:39:43 2021-01-19

Out[138]:

	Attraction	City	cleaned	lemmatized
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom	sea life london aquarium admission ticket	sea life london aquarium admission ticket
1	The Jack The Ripper Walking Tour in London	London, United Kingdom	the jack the ripper walking tour in london	jack ripper walking tour london
2	Ghost Bus Tour of London	London, United Kingdom	ghost bus tour of london	ghost bus tour london
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom	big bus london hopon hopoff tour and river cru...	big bus london hopon hopoff tour river cruise ...
4	The Blood and Tears Walk: Serial Killers and L...	London, United Kingdom	the blood and tears walk serial killers and lo...	blood tear walk serial killer london horror

In [139]:

```

1 # Preview what I want to drop
2 df.loc[df['Attraction'].str.contains('airport')]

```

executed in 19ms, finished 09:39:43 2021-01-19

Out[139]:

	Attraction	City	cleaned	lemmatized
329	Private transfer from Heathrow airport to Sout...	London, United Kingdom	private transfer from heathrow airport to sout...	private transfer heathrow airport southampton ...
639	Private transfer from city airport to central ...	London, United Kingdom	private transfer from city airport to central ...	private transfer city airport central london
640	private transfer from central london to city a...	London, United Kingdom	private transfer from central london to city a...	private transfer central london city airport
1549	Private airport transfers in London	London, United Kingdom	private airport transfers in london	private airport transfer london
1830	London airport transfer from Heathrow Airport ...	London, United Kingdom	london airport transfer from heathrow airport ...	london airport transfer heathrow airport lhr l...
...
1407	Private 4-hour tour of Dubai from hotel, airpo...	Dubai, United Arab Emirates	private tour of dubai from hotel airport or c...	private tour dubai hotel airport cruise loca...
2398	Dubai airport terminal 1,2 or 3 to Ras Al Khaimah	Dubai, United Arab Emirates	dubai airport terminal or to ras al khaimah	dubai airport terminal ras al khaimah
2407	Dubai airport terminal 1,2 or 3 to Ajman	Dubai, United Arab Emirates	dubai airport terminal or to ajman	dubai airport terminal ajman
2408	Dubai airport terminal 1,2 or 3 to Sharjah city	Dubai, United Arab Emirates	dubai airport terminal or to sharjah city	dubai airport terminal sharjah city
3168	Dubai city tour Stop Over pick up from airport...	Dubai, United Arab Emirates	dubai city tour stop over pick up from airport...	dubai city tour stop pick airport morning tour

314 rows × 4 columns

In [140]:

```

1 # Get rid of the airport transfer 'attractions'
2 df2 = df.drop(df.loc[df['Attraction'].str.contains('airport')].index)

```

executed in 18ms, finished 09:39:43 2021-01-19

In [141]:

```

1 df2 = df.drop(df2.loc[df2['Attraction'].str.contains('transfer')].index)

```

executed in 18ms, finished 09:39:43 2021-01-19

In [142]:

```

1 # Just in case, add these words to the stopwords list
2 stopwords_list += ['airport', 'transfer', 'private']

```

executed in 2ms, finished 09:39:43 2021-01-19

In [143]:

```

1 print(df.shape)
2 print(df2.shape)

```

executed in 3ms, finished 09:39:43 2021-01-19

(27533, 4)
(25315, 4)

Create some functions to make the preprocessing steps easier

In [144]:

```

1 def preprocess_df(df, column, preview=True, lemmatize=True):
2     """
3         Input df with raw text attractions.
4         Return df with preprocessed text.
5         If preview=True, returns a preview of the new df.
6         """
7
8     df[column] = df['Attraction'].apply(lambda x: x.lower())
9     df[column] = df[column].apply(lambda x: re.sub('[%s]' % re.escape(s
10     df[column] = df[column].apply(lambda x: re.sub('\w*\d\w*', '', x))
11
12     if lemmatize:
13         df[column] = df[column].apply(lambda x: ' '.join(
14                                         [token.lemma_ for token in list
15     if preview:
16         display(df.head(10))
17
18     return df

```

executed in 5ms, finished 09:39:43 2021-01-19

In [145]:

```

1 def group_text_per_city(df, column):
2     """
3         Groups the preprocessed text per city.
4         """
5     df_to_group = df[['City', column]]
6     df_grouped = df_to_group.groupby(by='City').agg(lambda x: ' '.join(x
7     return df_grouped

```

executed in 2ms, finished 09:39:43 2021-01-19

In [146]:

```

1 def create_doc_term_matrix(df, column, count_vec=True, ngram_range=(1, 1
2     """
3         Creates a document term matrix.
4         Defaults to count vectorizer with optional n-gram param.
5         If count_vec==False, uses a TF-IDF vectorizer.
6         """
7     df_grouped = group_text_per_city(df, column)
8
9     if count_vec:
10         vec = CountVectorizer(analyzer='word', stop_words=stopwords_lis
11     else:
12         vec = TfidfVectorizer(analyzer='word', stop_words=stopwords_lis
13
14     data = vec.fit_transform(df_grouped[column])
15     df_dtm = pd.DataFrame(data.toarray(), columns=vec.get_feature_names
16     df_dtm.index = df_grouped.index
17     return df_dtm.transpose()

```

executed in 3ms, finished 09:39:43 2021-01-19

In [147]:

```

1 preprocessed_df = preprocess_df(df2, 'lemmatized')
2 dtm_cv = create_doc_term_matrix(preprocessed_df, 'lemmatized', count_vec=False)
3
4 for index, city in enumerate(dtm_cv.columns):
5     generate_wordcloud(dtm_cv[city].sort_values(ascending=False), city)

```

executed in 1m 36.9s, finished 09:41:20 2021-01-19

	Attraction	City	cleaned	lemmatized
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom	sea life london aquarium admission ticket	sea life london aquarium admission ticket
1	The Jack The Ripper Walking Tour in London	London, United Kingdom	the jack the ripper walking tour in london	jack ripper walking tour london
2	Ghost Bus Tour of London	London, United Kingdom	ghost bus tour of london	ghost bus tour london
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom	big bus london hopon hopoff tour and river cru...	big bus london hopon hopoff tour river cruise ...
5	London Ghost and Infamous Murders Walking Tour	London, United Kingdom	london ghost and infamous murders walking tour	london ghost infamous murder walk tour
	Stonehenge Windsor Castle	London,	stonehenge windsor castle	stonehenge windsor castle

In [148]:

```

1 dtm_tfidf = create_doc_term_matrix(df2, 'lemmatized', count_vec=False)
2
3 for index, city in enumerate(dtm_tfidf.columns):
4     generate_wordcloud(dtm_tfidf[city].sort_values(ascending=False), ci)

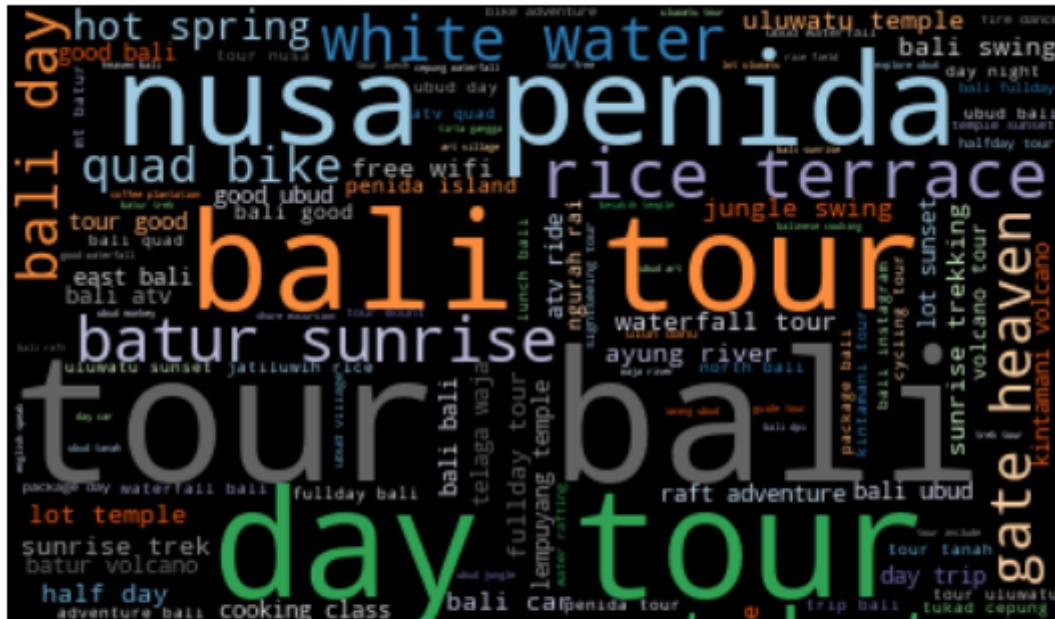
```

executed in 2.90s, finished 09:41:23 2021-01-19

```
In [149]: 1 dtm_bigram = create_doc_term_matrix(df2, 'lemmatized',
2                                         count_vec=True, ngram_range=(2,2))
3
4 for index, city in enumerate(dtm_bigram.columns):
5     generate_wordcloud(dtm_bigram[city].sort_values(ascending=False), c
```

executed in 3.32s, finished 09:41:26 2021-01-19

Bali, Indonesia



Much better! I removed some of the noise terms out of the top words.

1.4.1 Make Nicer Word Clouds

```
In [150]: 1 def generate_better_wordcloud(data, title, mask=None):
2     cloud = WordCloud(scale=3, max_words=150, colormap='tab20c', mask=m
3                         background_color='white').generate_from_frequencies(data)
4     plt.figure(figsize=(10,8))
5     plt.imshow(cloud, interpolation='bilinear')
6     plt.axis('off')
7     plt.title('\n'.join(wrap(title,60)), fontsize=13)
8     plt.show()
```

executed in 3ms. finished 09:41:26 2021-01-19

```
In [151]: 1 mask = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron/capsto  
executed in 12ms, finished 09:41:26 2021-01-19
```

executed in 1.71s, finished 09:41:28 2021-01-19



```
In [153]: 1 mask_london = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron  
executed in 5ms, finished 09:41:28 2021-01-19
```

executed in 5ms finished 09:41:28 2021-01-19

```
In [154]: 1 generate_better_wordcloud(dtm_tfidf['London, United Kingdom'].sort_value  
2                                'London, United Kingdom', mask=mask_london)
```

executed in 569ms, finished 09:41:29 2021-01-19



executed in 949ms, finished 09:41:29 2021-01-19

Paris, France



executed in 5.94s, finished 09:41:35 2021-01-19



executed in 1.33s, finished 09:41:37 2021-01-19

Sicily, Italy



executed in 1.10s, finished 09:41:38 2021-01-19

Barcelona, Spain



```
In [159]: 1 mask_dubai = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron/  
2 generate_better_wordcloud(dtm_tfidf['Dubai, United Arab Emirates'].sort  
3                                         'Dubai, United Arab Emirates', mask=mask_duba
```

executed in 1.39s, finished 09:41:39 2021-01-19

Dubai, United Arab Emirates



executed in 1.55s, finished 09:41:41 2021-01-19



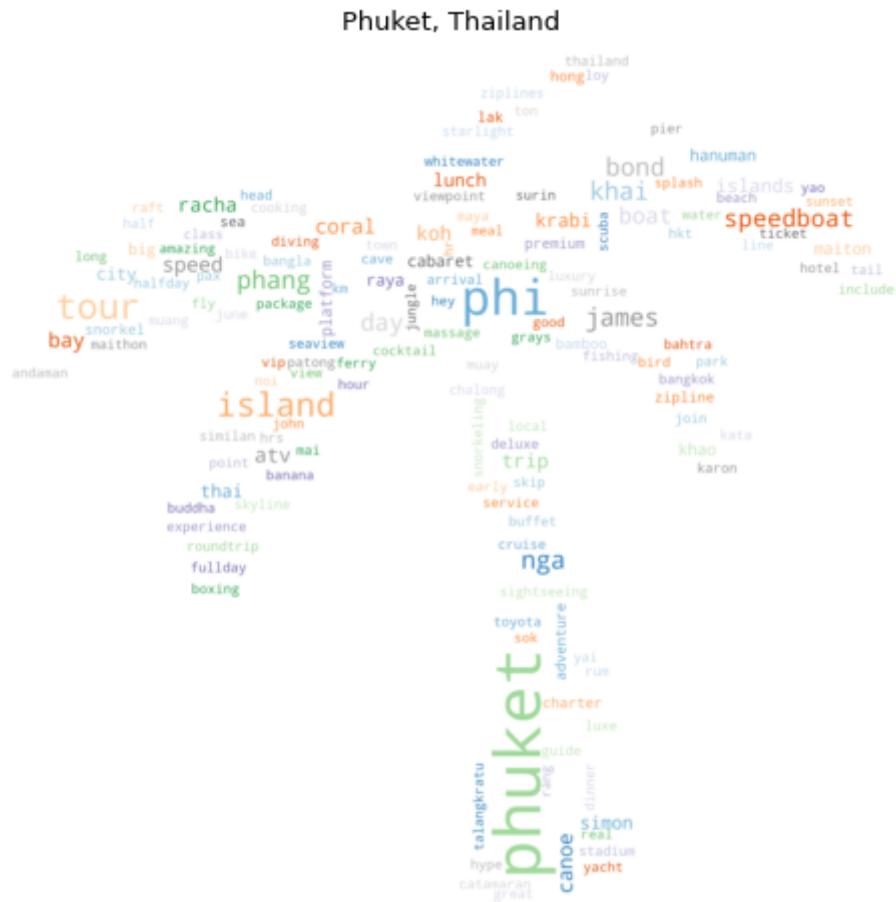
```
In [161]: 1 mask_istanbul = np.array(Image.open('/Users/tiaplagata/Documents/Flatir
 2 generate_better_wordcloud(dtm_tfidf['Istanbul, Turkey'].sort_values(asc
 3                                         'Istanbul, Turkey', mask=mask_istanbul)
```

executed in 991ms, finished 09:41:42 2021-01-19

Istanbul, Turkey



executed in 1.53s, finished 09:41:43 2021-01-19



```
In [163]: 1 generate_better_wordcloud(dtm_tfidf[['Majorca, Balearic Islands']].sort_v  
2                                         'Majorca, Balearic Islands', mask=mask)
```

executed in 1.51s, finished 09:41:45 2021-01-19



executed in 1.65s, finished 09:41:46 2021-01-19



▼ 1.4.2 Most Frequent Words Visualizations

- Find the most frequent words per city and visualize them

In [165]:

```

1 # Group the corpora by city and join them
2 df_to_group = preprocessed_df[['City', 'lemmatized']]
3 df_grouped = df_to_group.groupby(by='City').agg(lambda x: ' '.join(x))
4 df_grouped

```

executed in 21ms, finished 09:41:47 2021-01-19

Out[165]:

lemmatized

City	
Bali, Indonesia	hotel hotelbali private transfer daytime bali ...
Barcelona, Spain	interactive spanish cooking experience barcelo...
Crete, Greece	minoans world museum cinema crete wine ol...
Dubai, United Arab Emirates	premium red dune camel safari bbq al khayma ...
Goa, India	fontainhas heritage walk sunset cruise paradis...
Istanbul, Turkey	bosphorus sunset cruise luxury yacht istanbu...
London, United Kingdom	sea life london aquarium admission ticket jack...
Majorca, Balearic Islands	cave genova admission palma de mallorca shore ...
Paris, France	bateaux parisiens seine river gourmet dinner ...
Phuket, Thailand	phi phi maiton khai islands speedboat phi ph...
Rome, Italy	fast skiptheline vatican sistine chapel st pet...
Sicily, Italy	etna taormina fullday tour catania palermo str...

```
In [166]: 1 bali_text = df_grouped.loc['Bali, Indonesia', 'lemmatized']
2 fd = FreqDist(word_tokenize(bali_text))
3 fd.most_common(20)
```

executed in 116ms, finished 09:41:47 2021-01-19

```
Out[166]: [('bali', 2206),
 ('tour', 2187),
 ('private', 1025),
 ('ubud', 869),
 ('day', 669),
 ('temple', 513),
 ('waterfall', 402),
 ('raft', 308),
 ('batur', 292),
 ('transfer', 275),
 ('good', 275),
 ('water', 262),
 ('airport', 253),
 ('adventure', 252),
 ('nusa', 252),
 ('lot', 251),
 ('tanah', 242),
 ('fullday', 231),
 ('package', 226),
 ('swing', 224)]
```

```
In [167]: 1 city_freqs = {}
2 for city in df_grouped.index:
3     city_text = df_grouped.loc[city, 'lemmatized']
4     fd = FreqDist(word_tokenize(city_text))
5     city_freqs[city] = fd.most_common(20)
6 city_freqs_df = pd.DataFrame(city_freqs)
7 city_freqs_df.head()
```

executed in 634ms, finished 09:41:47 2021-01-19

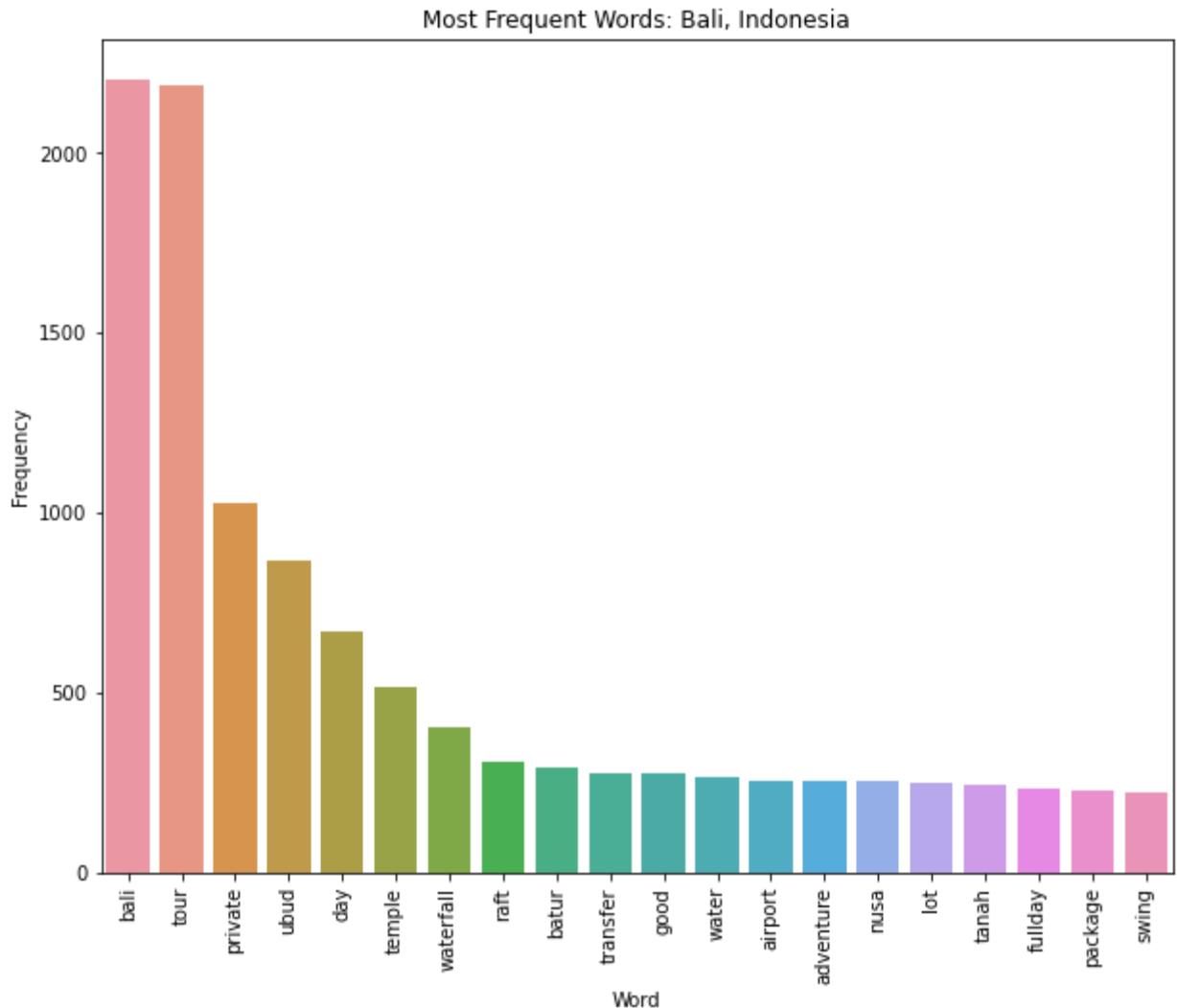
Out[167]:

	Bali, Indonesia	Barcelona, Spain	Crete, Greece	Dubai, United Arab Emirates	Goa, India	Istanbul, Turkey	London, United Kingdom	Majorca, Balearic Islands	Paris, France	Phu Thail
0	(bali, 2206)	(barcelona, 1105)	(private, 340)	(dubai, 2144)	(goa, 220)	(istanbul, 1338)	(london, 1862)	(mallorca, 209)	(paris, 1654)	(phu
1	(tour, 2187)	(tour, 861)	(tour, 272)	(tour, 1321)	(tour, 150)	(tour, 1243)	(tour, 1462)	(tour, 144)	(tour, 1152)	(t
2	(private, 1025)	(private, 616)	(transfer, 258)	(desert, 900)	(private, 45)	(day, 656)	(private, 1201)	(palma, 133)	(private, 949)	(
3	(ubud, 869)	(transfer, 166)	(airport, 251)	(safari, 898)	(walk, 41)	(private, 578)	(airport, 439)	(private, 97)	(airport, 293)	(isla
4	(day, 669)	(airport, 152)	(crete, 216)	(private, 708)	(day, 39)	(airport, 299)	(transfer, 404)	(de, 90)	(transfer, 280)	(

In [168]:

```
1 # Make one graph of most frequent words
2 xaxis = [x[0] for x in city_freqs_df['Bali, Indonesia']]
3 yaxis = [x[1] for x in city_freqs_df['Bali, Indonesia']]
4
5 plt.figure(figsize=(10,8))
6 sns.barplot(xaxis, yaxis)
7 plt.title('Most Frequent Words: Bali, Indonesia')
8 plt.xticks(rotation=90)
9 plt.xlabel('Word')
10 plt.ylabel('Frequency')
11 plt.show()
```

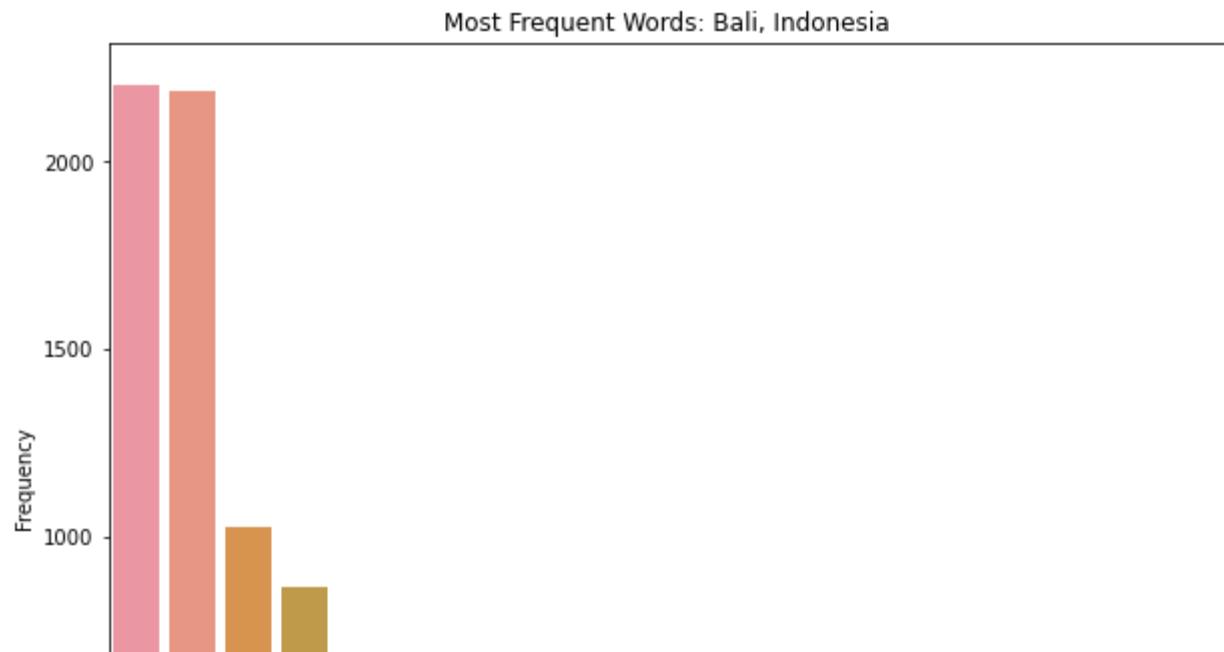
executed in 156ms, finished 09:41:47 2021-01-19



In [169]:

```
1 # Make graphs for each city
2 for city in city_freqs_df.columns:
3     xaxis = [x[0] for x in city_freqs_df[city]]
4     yaxis = [x[1] for x in city_freqs_df[city]]
5
6     plt.figure(figsize=(10,8))
7     sns.barplot(xaxis, yaxis)
8     plt.title(f'Most Frequent Words: {city}')
9     plt.xticks(rotation=90)
10    plt.xlabel('Word')
11    plt.ylabel('Frequency')
12    plt.show()
```

executed in 1.75s, finished 09:41:49 2021-01-19



1.5 Modeling



1.5.1 Baseline Naive Bayes Model

```
In [170]: 1 # Re-import the data to get a fresh start
2 data = pd.read_csv('/Users/tiaplagata/Documents/Flatiron/capstone-project/tourism.csv')
3 data.head()
executed in 34ms, finished 09:41:49 2021-01-19
```

Out[170]:

	Attraction	City
0	SEA LIFE London Aquarium Admission Ticket	London, United Kingdom
1	The Jack The Ripper Walking Tour in London	London, United Kingdom
2	Ghost Bus Tour of London	London, United Kingdom
3	Big Bus London Hop-On Hop-Off Tour and River C...	London, United Kingdom
4	The Blood and Tears Walk: Serial Killers and L...	London, United Kingdom

```
In [171]: 1 # Perform train/test split before cleaning/preprocessing
2 X = data['Attraction']
3 y = data['City']
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2,
5 X_train.shape, X_test.shape)
executed in 9ms, finished 09:41:49 2021-01-19
```

Out[171]: ((22703,), (5676,))

```
In [172]: 1 # Since this is a series, I will need to make it a DF for my preprocessing
2 X_train
executed in 4ms, finished 09:41:49 2021-01-19
```

```
Out[172]: 2092      The Colosseum and The Ancient City of Rome
814          Paintball in Canggu/Bali
1920     9Hr Tour London Eye, Westminster Abbey and St ...
348          Dubai 3h Sea escape: Swim! Tan! Sightsee!
3122     Private Bali Half Day Car Charter - Uluwatu Su...
...
1428          Barcelona City + La Roca Village Tour
14          Etna, Wine and Alcantara Tour - Small Groups f...
3345          Fujiearah East Coast Tour
4210     Early Morning Vatican Museums, Sistine Chapel, ...
246          Rafa Nadal Museum Mallorca Half Day Tour
Name: Attraction, Length: 22703, dtype: object
```

In [173]:

```

1 X_train_preprocessed = preprocess_df(pd.DataFrame(X_train, columns=['Attraction'])
2 X_test_preprocessed = preprocess_df(pd.DataFrame(X_test, columns=['Attraction'])

```

executed in 1m 48.2s, finished 09:43:37 2021-01-19

	Attraction	lemmatized
2092	The Colosseum and The Ancient City of Rome	colosseum ancient city rome
814	Paintball in Canggu/Bali	paintball canggubali
1920	9Hr Tour London Eye, Westminster Abbey and St Paul's Cathedral	tour london eye westminster abbey st pauls c...
348	Dubai 3h Sea escape: Swim! Tan! Sightsee!	dubai sea escape swim tan sightsee
3122	Private Bali Half Day Car Charter - Uluwatu Sunsets	private bali half day car charter uluwatu su...
1313	Rooftop Pasta Making Class and Food Market Tour in Rome	rooftop pasta make class food market tour rome
952	Colosseum, Forum and Baroque Squares	colosseum forum baroque square
972	Gothic Quarter's deepest secrets & Sangria	gothic quarter deep secret sangria
1622	Private Half-Day Montserrat Tour in Afternoon	private halfday montserrat tour afternoon pi...
1036	Changing of the Guard Half-Day Private Walking Tour	change guard halfday private walk london tour

In [174]:

```

1 stopwords_list = stopwords.words('english')
2 stopwords_list += list(string.punctuation)
3 stopwords_list += ['airport', 'transfer', 'private']

```

executed in 2ms, finished 09:43:37 2021-01-19

In [175]:

```

1 # Vectorize the text data to be suitable for modeling
2 vectorizer = TfidfVectorizer(analyzer='word', stop_words=stopwords_list)
3 X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['Attraction'])
4 X_test_tfidf = vectorizer.transform(X_test_preprocessed['Attraction'])

```

executed in 197ms, finished 09:43:38 2021-01-19

In [176]:

```

1 nb = MultinomialNB()
2 nb.fit(X_train_tfidf.todense(), y_train)

```

executed in 1.20s, finished 09:43:39 2021-01-19

Out[176]: MultinomialNB()

In [177]:

```

1 def plot_conf_matrix(y_true, y_pred):
2
3     """
4         Plots a confusion matrix and displays classification report.
5     """
6
7     cm = confusion_matrix(y_true, y_pred, normalize='true')
8     plt.figure(figsize=(15, 15))
9     sns.heatmap(cm, annot=True, cmap='Blues', fmt='0.2g', annot_kws={"s": 10, "dx": 0, "dy": 0}, xticklabels=nb.classes_, yticklabels=nb.classes_, square=True)
10    plt.xlabel('Predictions')
11    plt.ylabel('Actuals')
12    plt.show()
13

```

executed in 6ms, finished 09:43:39 2021-01-19

In [178]:

```

1 def evaluate_model(model, X_train, X_test):
2     y_preds_train = model.predict(X_train.todense())
3     y_preds_test = model.predict(X_test.todense())
4
5     print('Training Accuracy:', accuracy_score(y_train, y_preds_train))
6     print('Testing Accuracy:', accuracy_score(y_test, y_preds_test))
7     print('\n-----\n')
8     print('Training F1:', f1_score(y_train, y_preds_train, average='weighted'))
9     print('Testing F1:', f1_score(y_test, y_preds_test, average='weighted'))
10    print('\n-----\n')
11    print('Train Confusion Matrix\n')
12    plot_conf_matrix(y_train, y_preds_train)
13    print('Test Confusion Matrix\n')
14    plot_conf_matrix(y_test, y_preds_test)
15    print('\n-----\n')
16    print(classification_report(y_test, y_preds_test))

```

executed in 3ms, finished 09:43:39 2021-01-19

In [179]:

1 nb.classes_

executed in 3ms, finished 09:43:39 2021-01-19

Out[179]: array(['Bali, Indonesia', 'Barcelona, Spain', 'Crete, Greece',
 'Dubai, United Arab Emirates', 'Goa, India', 'Istanbul, Turkey',
 'London, United Kingdom', 'Majorca, Balearic Islands',
 'Paris, France', 'Phuket, Thailand', 'Rome, Italy',
 'Sicily, Italy'], dtype='|<U27')

In [180]:

```
1 evaluate_model(nb, X_train_tfidf, X_test_tfidf)
```

executed in 2.38s, finished 09:43:41 2021-01-19

Training Accuracy: 0.9226093467823636

Testing Accuracy: 0.8921775898520085

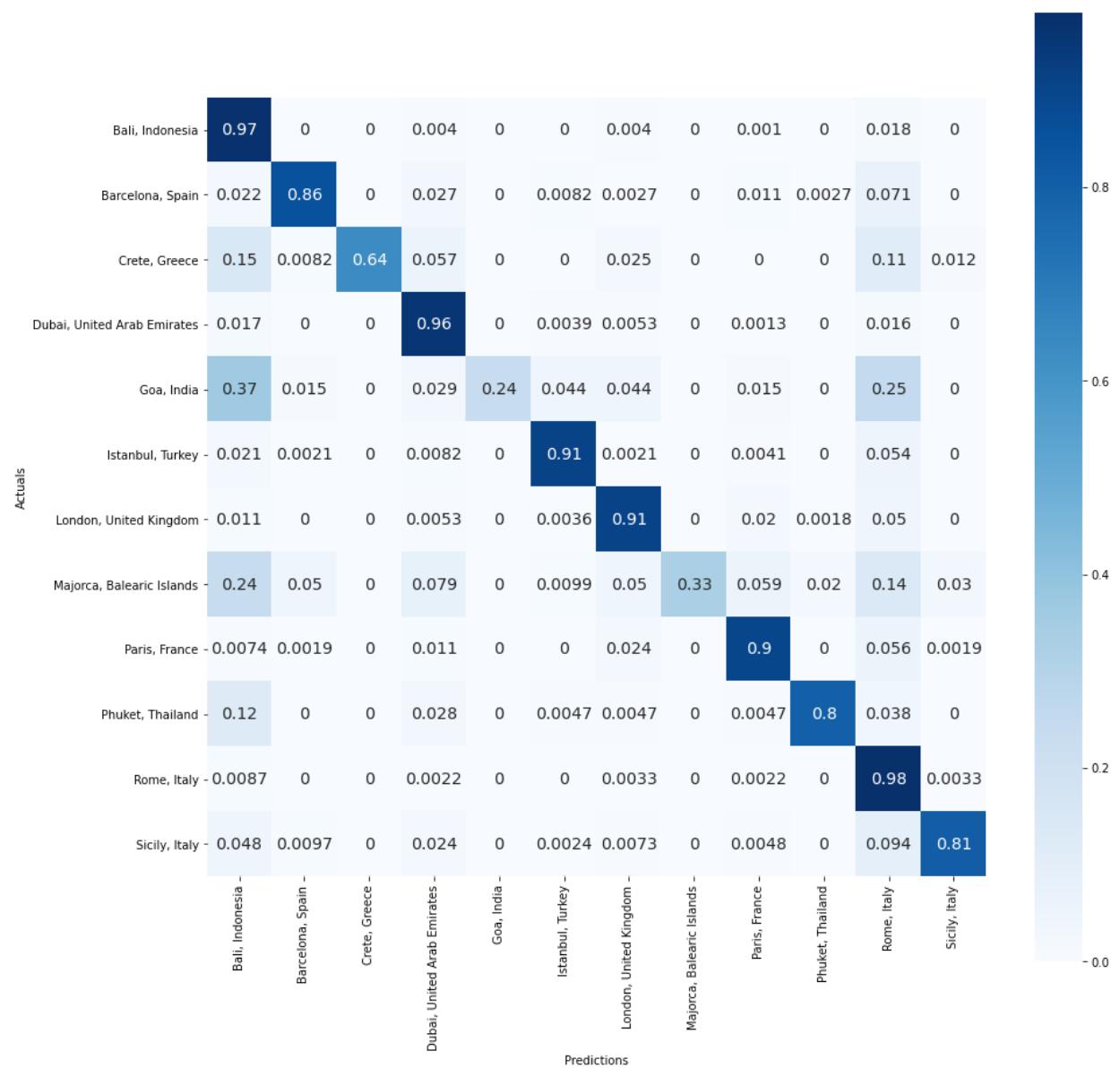
Training F1: 0.9181553338941022

Testing F1: 0.8867599504204542

Train Confusion Matrix



Test Confusion Matrix



	precision	recall	f1-score	support
Bali, Indonesia	0.84	0.97	0.90	1001
Barcelona, Spain	0.96	0.86	0.90	368
Crete, Greece	1.00	0.64	0.78	244
Dubai, United Arab Emirates	0.91	0.96	0.93	760
Goa, India	1.00	0.24	0.38	68
Istanbul, Turkey	0.97	0.91	0.94	485
London, United Kingdom	0.92	0.91	0.92	563
Majorca, Balearic Islands	1.00	0.33	0.49	101
Paris, France	0.94	0.90	0.92	538
Phuket, Thailand	0.98	0.80	0.88	212
Rome, Italy	0.79	0.98	0.87	923
Sicily, Italy	0.97	0.81	0.88	413
accuracy			0.89	5676
macro avg	0.94	0.77	0.82	5676
weighted avg	0.90	0.89	0.89	5676

Surprisingly, this model performs pretty well. However, the 3 classes with the lowest accuracy and F1 scores are Goa, Majorca, and Crete. These are also the 3 classes with the least attractions, meaning that class imbalance is definitely affecting this model. I can fix this issue using class weights in the next iteration.

▼ 1.5.2 Naive Bayes Iteration 2

- Using class weights to improve class imbalance.

In [181]:

```

1 # Compute class weights
2 class_weights = class_weight.compute_class_weight('balanced',
3                                                 np.unique(y_train),
4                                                 y_train)
5 weights_dict = dict(zip(np.unique(y_train), class_weights))
6 weights_dict

```

executed in 25ms, finished 09:43:41 2021-01-19

Out[181]:

```
{'Bali, Indonesia': 0.4730974410269234,
 'Barcelona, Spain': 1.272304416050213,
 'Crete, Greece': 2.42864783910997,
 'Dubai, United Arab Emirates': 0.6448250397636901,
 'Goa, India': 6.732799525504152,
 'Istanbul, Turkey': 1.038373582144164,
 'London, United Kingdom': 0.8446056547619047,
 'Majorca, Balearic Islands': 5.359537299338999,
 'Paris, France': 0.8795521462885479,
 'Phuket, Thailand': 1.9129592180653858,
 'Rome, Italy': 0.4640462758564304,
 'Sicily, Italy': 1.1891368112298344}
```

In [182]:

```

1 # Use class weights dictionary to calculate sample weight (needed for M
2 sample_weights = class_weight.compute_sample_weight(weights_dict, y_tr

```

executed in 14ms, finished 09:43:41 2021-01-19

In [183]:

```

1 nb = MultinomialNB()
2 nb.fit(X_train_tfidf.todense(),
3         y_train,
4         sample_weight=sample_weights)

```

executed in 810ms, finished 09:43:42 2021-01-19

Out[183]:

```
MultinomialNB()
```

In [184]:

```
1 evaluate_model(nb, X_train_tfidf, X_test_tfidf)
```

executed in 2.27s, finished 09:43:44 2021-01-19

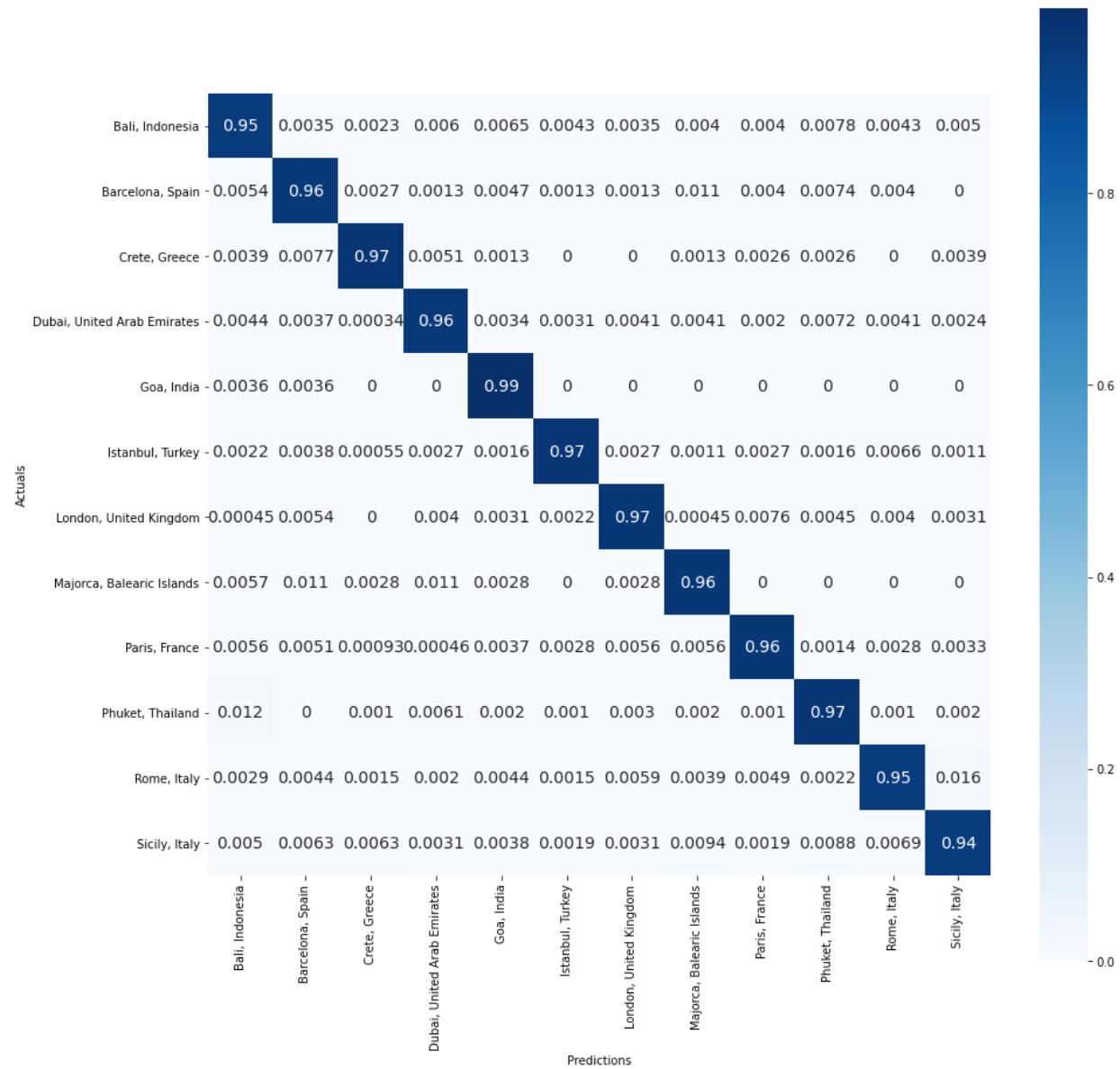
Training Accuracy: 0.9581112628287011

Testing Accuracy: 0.9277660324171952

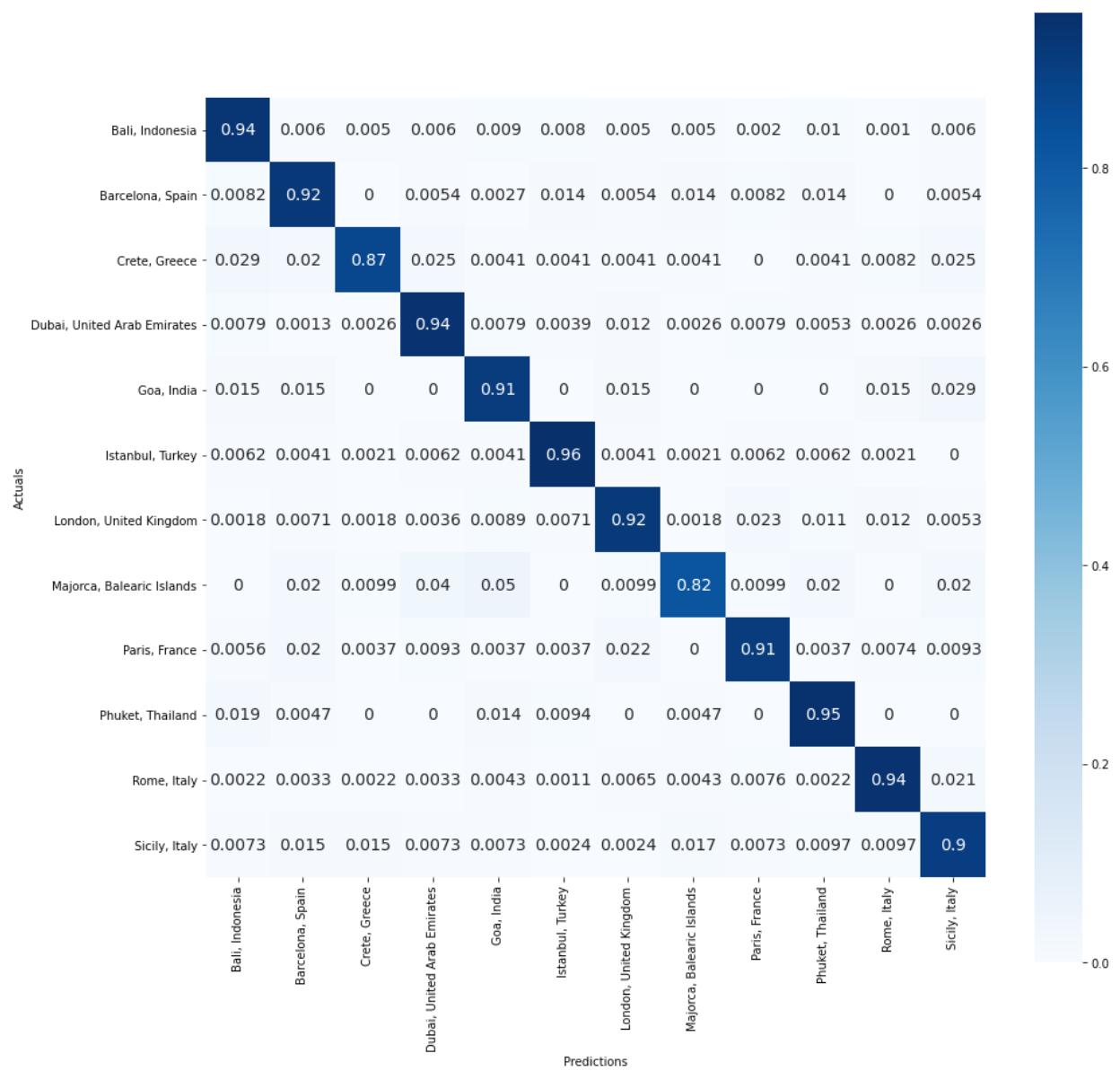
Training F1: 0.9585693303384949

Testing F1: 0.9287478733306096

Train Confusion Matrix



Test Confusion Matrix



	precision	recall	f1-score	support
Bali, Indonesia	0.97	0.94	0.95	1001
Barcelona, Spain	0.89	0.92	0.91	368
Crete, Greece	0.91	0.87	0.89	244
Dubai, United Arab Emirates	0.95	0.94	0.95	760
Goa, India	0.60	0.91	0.73	68
Istanbul, Turkey	0.95	0.96	0.95	485
London, United Kingdom	0.93	0.92	0.92	563
Majorca, Balearic Islands	0.75	0.82	0.79	101
Paris, France	0.93	0.91	0.92	538
Phuket, Thailand	0.84	0.95	0.89	212
Rome, Italy	0.98	0.94	0.96	923
Sicily, Italy	0.89	0.90	0.89	413
accuracy			0.93	5676
macro avg	0.88	0.92	0.90	5676
weighted avg	0.93	0.93	0.93	5676

This model did really well! Although, in many of these cities' attractions text, the name of the city is included. This may become an issue in the future because when we introduce new text to this model, it may not include the city name.

▼ **1.5.3 Iteration 3: What happens if I take the city names out?**

```
In [185]: 1 new_stopwords = stopwords_list + ['bali', 'barcelona', 'crete', 'dubai'  
2                                         'istanbul', 'london', 'majorca', 'phu  
3                                         'paris', 'rome', 'sicily', 'mallorca']
```

executed in 3ms, finished 09:43:44 2021-01-19

```
In [186]: 1 vectorizer = TfidfVectorizer(analyzer='word',  
2                                     stop_words=new_stopwords,  
3                                     decode_error='ignore')  
4 X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['lemmatized'])  
5 X_test_tfidf = vectorizer.transform(X_test_preprocessed['lemmatized'])
```

executed in 176ms, finished 09:43:45 2021-01-19

```
In [187]: 1 nb = MultinomialNB()  
2 nb.fit(X_train_tfidf.todense(),  
3         y_train,  
4         sample_weight=sample_weights)
```

executed in 794ms, finished 09:43:45 2021-01-19

Out[187]: MultinomialNB()

```
In [188]: 1 evaluate_model(nb, X_train_tfidf, X_test_tfidf)
```

executed in 2.32s, finished 09:43:52 2021-01-19

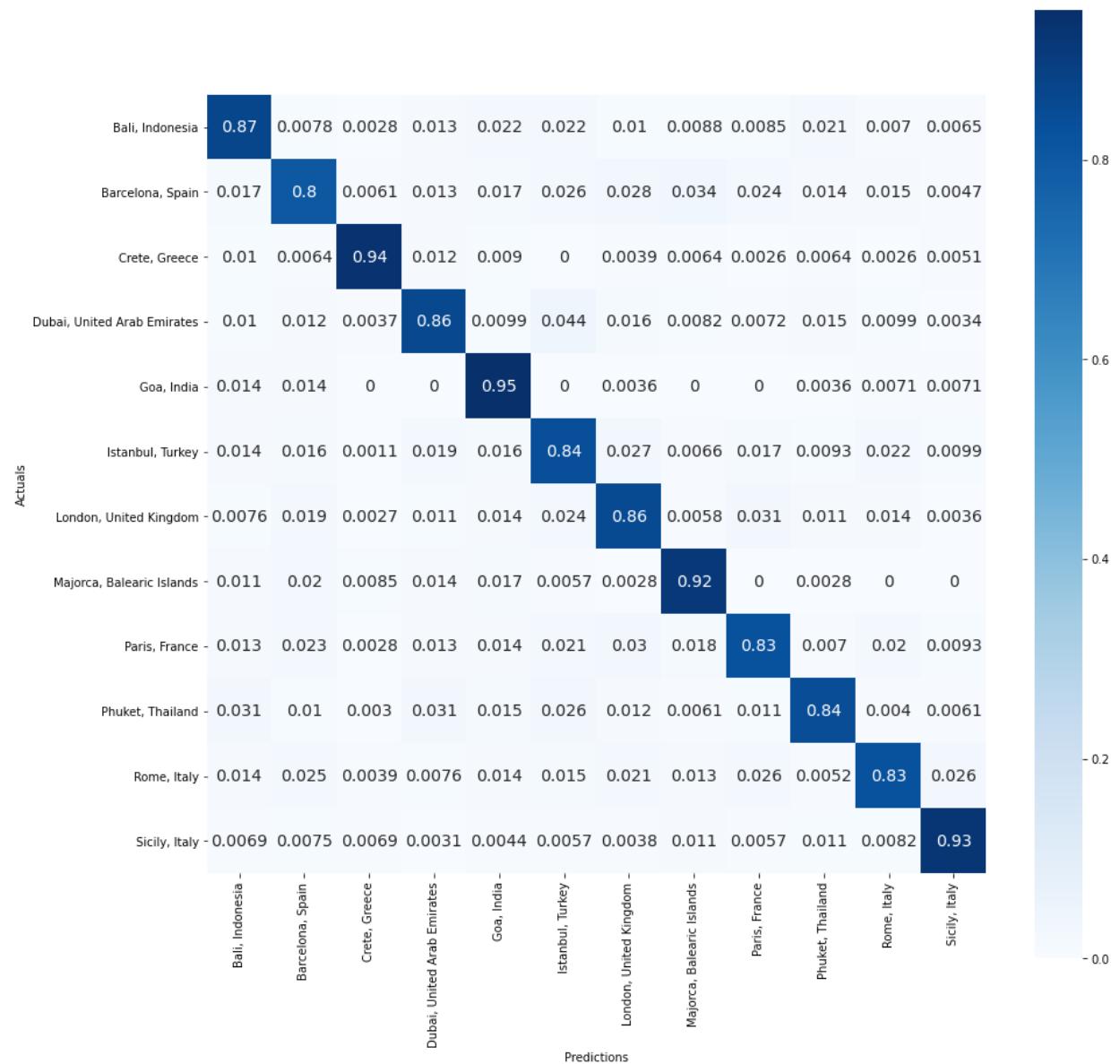
Training Accuracy: 0.8565387834206933

Testing Accuracy: 0.8097251585623678

Training F1: 0.8598573049581161

Testing F1: 0.8145056259393049

Train Confusion Matrix



Test Confusion Matrix



	precision	recall	f1-score	support
Bali, Indonesia	0.91	0.84	0.88	1001
Barcelona, Spain	0.68	0.71	0.69	368
Crete, Greece	0.87	0.82	0.84	244
Dubai, United Arab Emirates	0.89	0.86	0.87	760
Goa, India	0.33	0.75	0.46	68
Istanbul, Turkey	0.76	0.83	0.79	485
London, United Kingdom	0.81	0.77	0.79	563
Majorca, Balearic Islands	0.51	0.75	0.61	101
Paris, France	0.78	0.75	0.77	538
Phuket, Thailand	0.67	0.77	0.71	212
Rome, Italy	0.91	0.81	0.85	923
Sicily, Italy	0.82	0.88	0.85	413
accuracy			0.81	5676
macro avg		0.74	0.80	0.76
weighted avg		0.82	0.81	0.81

In [189]:

```
1 # Pickle the Naive Bayes Model
2 with open('/Users/tiaplagata/Documents/Flatiron/capstone-project/nb_mod'
3           'wb') as f:
4     pickle.dump(nb, f, pickle.HIGHEST_PROTOCOL)
```

executed in 4ms, finished 09:44:12 2021-01-19

Much better, because these are more realistic accuracy scores and F1 scores for when we introduce new text to the model.

▼ **1.5.4 Iteration 4: Try using Count Vectorization**

In [190]:

```
1 # Continuing each new iteration without city names
2 cv = CountVectorizer(analyzer='word',
3                      stop_words=new_stopwords,
4                      decode_error='ignore')
5 X_train_cv = cv.fit_transform(X_train_preprocessed['lemmatized'])
6 X_test_cv = cv.transform(X_test_preprocessed['lemmatized'])
7 nb_cv = MultinomialNB()
8 nb_cv.fit(X_train_cv.todense(),
9            y_train,
10           sample_weight=sample_weights)
11 evaluate_model(nb_cv, X_train_cv, X_test_cv)
```

executed in 5.34s, finished 09:44:21 2021-01-19

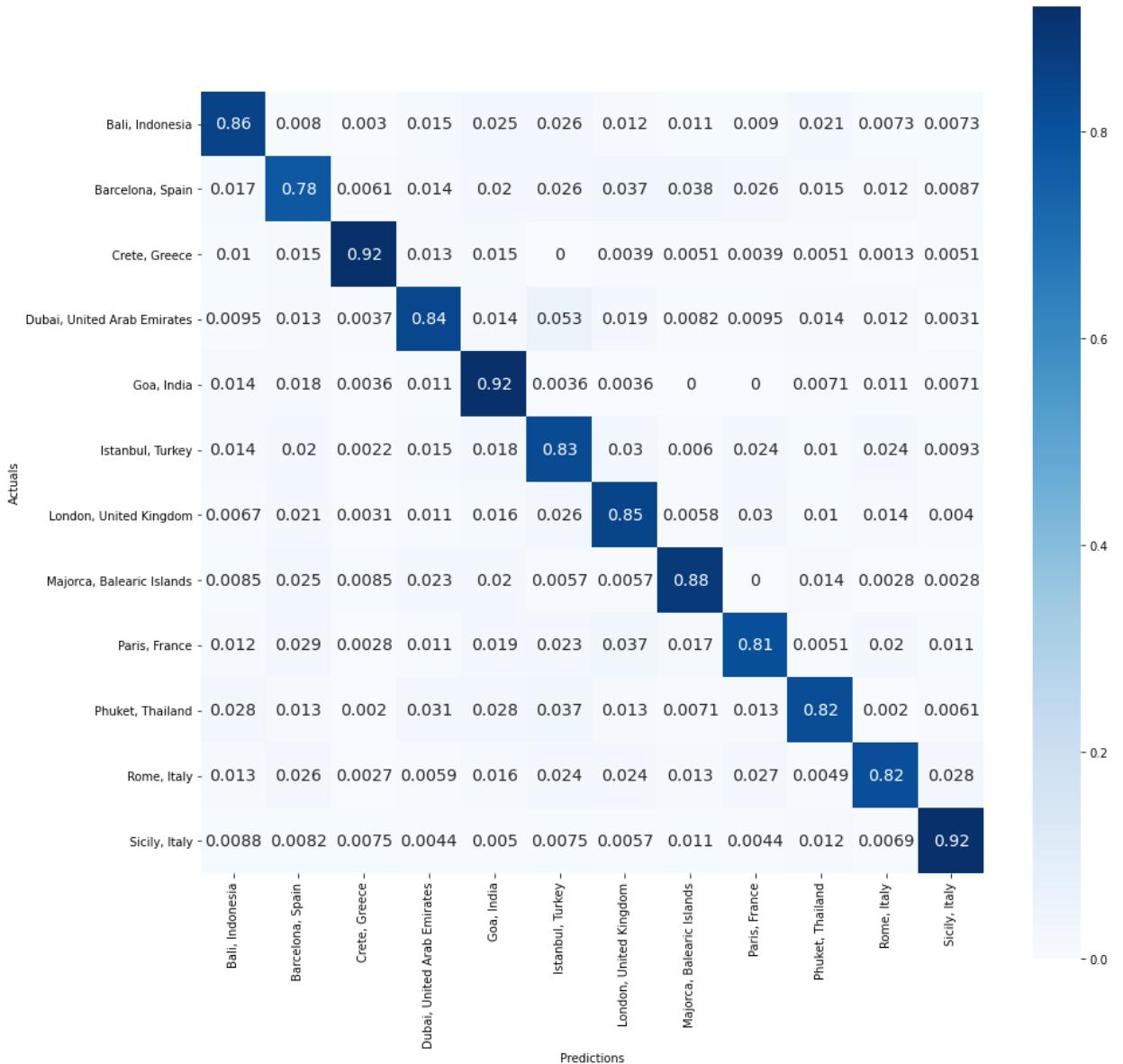
Training Accuracy: 0.8411663656785446

Testing Accuracy: 0.7986257928118393

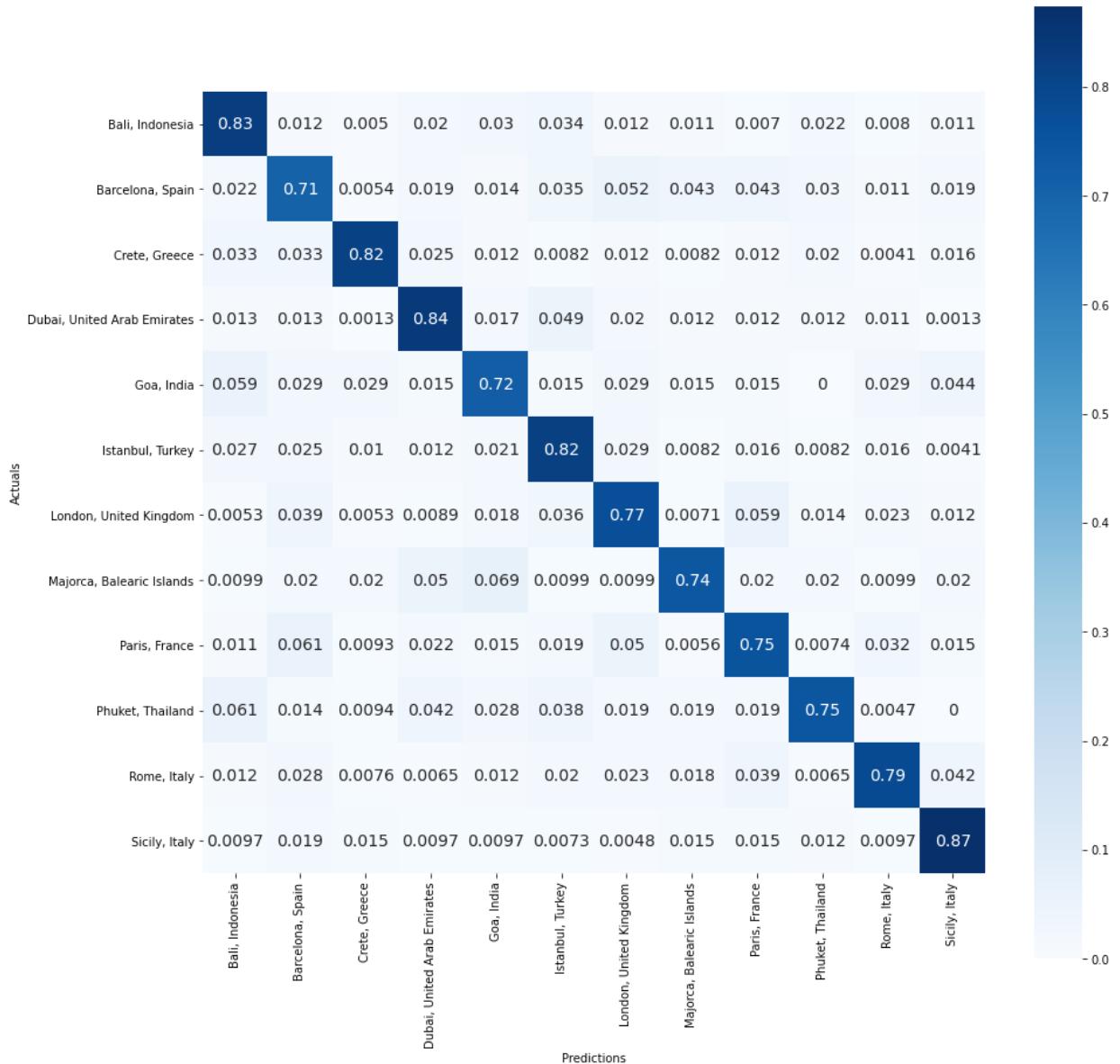
Training F1: 0.8457208352328576

Testing F1: 0.804145844014088

Train Confusion Matrix



Test Confusion Matrix



	precision	recall	f1-score	support
Bali, Indonesia	0.91	0.83	0.87	1001
Barcelona, Spain	0.65	0.71	0.68	368
Crete, Greece	0.83	0.82	0.82	244
Dubai, United Arab Emirates	0.89	0.84	0.86	760
Goa, India	0.31	0.72	0.44	68
Istanbul, Turkey	0.73	0.82	0.77	485
London, United Kingdom	0.78	0.77	0.78	563
Majorca, Balearic Islands	0.49	0.74	0.59	101
Paris, France	0.76	0.75	0.76	538
Phuket, Thailand	0.68	0.75	0.71	212

Rome, Italy	0.92	0.79	0.85	923
Sicily, Italy	0.81	0.87	0.84	413
accuracy			0.80	5676
macro avg	0.73	0.78	0.75	5676
weighted avg	0.82	0.80	0.80	5676

With count vectorization, the scores are very similar, but still a tiny bit lower than with TF-IDF vectorization, therefore I will keep the TF-IDF vectorization strategy.

▼ **1.5.5 Iteration 5: Try using Bi-Grams**

In [191]:

```
1 bigram = CountVectorizer(analyzer='word',
2                         stop_words=new_stopwords,
3                         decode_error='ignore',
4                         ngram_range=(2,2))
5 X_train_bg = bigram.fit_transform(X_train_preprocessed['lemmatized'])
6 X_test_bg = bigram.transform(X_test_preprocessed['lemmatized'])
7 nb_bg = MultinomialNB()
8 nb_bg.fit(X_train_bg.todense(),
9             y_train,
10            sample_weight=sample_weights)
11 evaluate_model(nb_bg, X_train_bg, X_test_bg)
```

executed in 24.5s, finished 09:46:12 2021-01-19

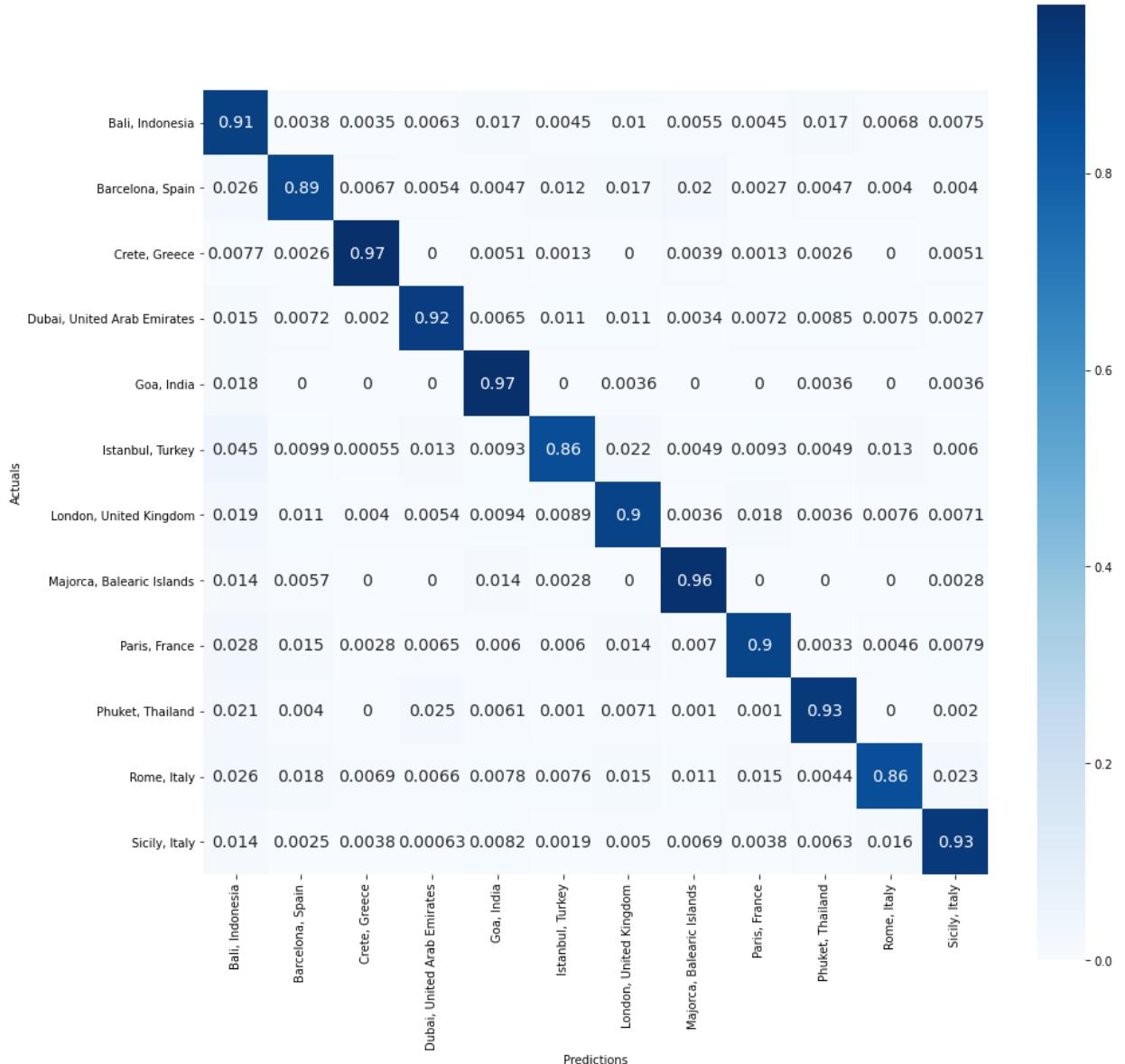
Training Accuracy: 0.9017310487600758

Testing Accuracy: 0.7330866807610994

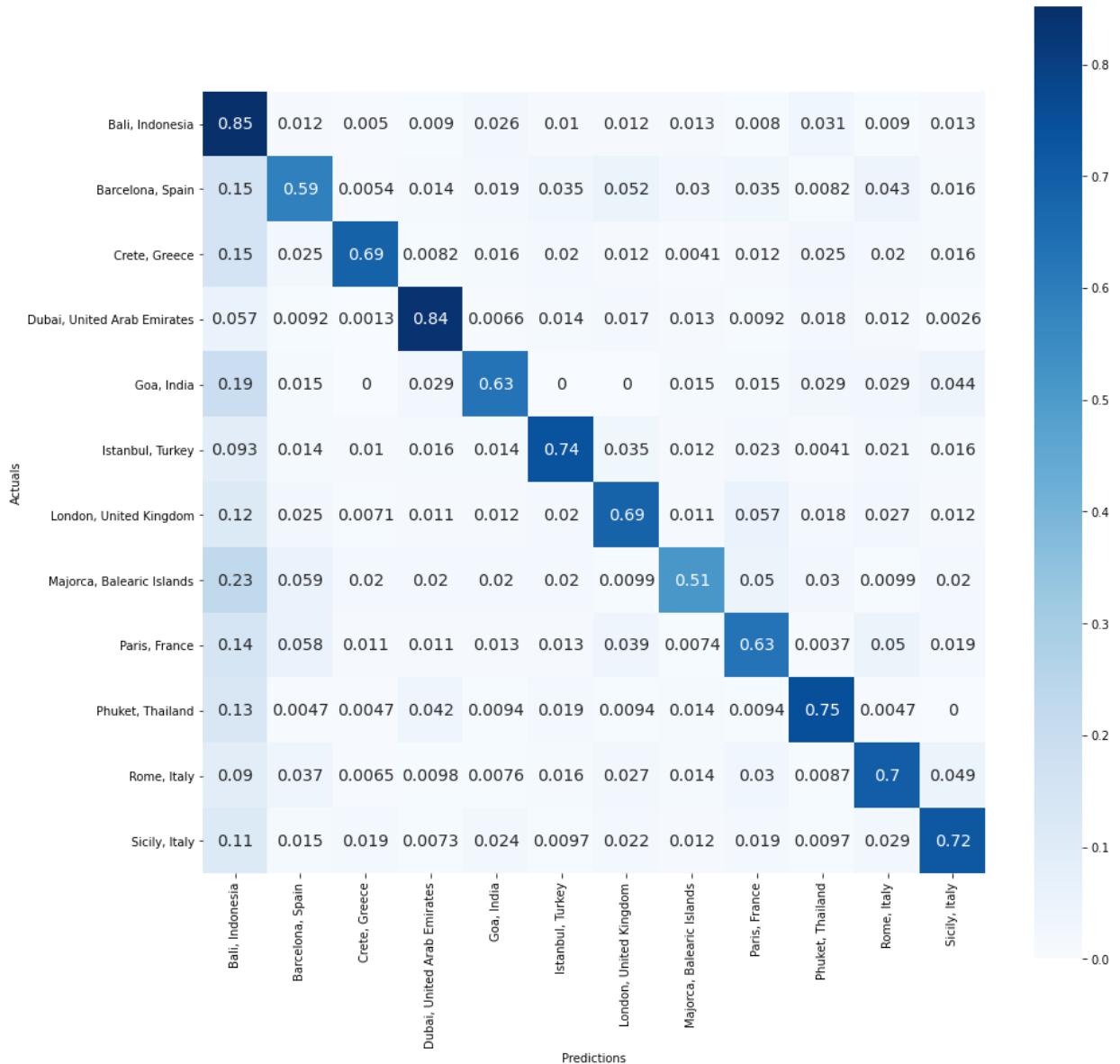
Training F1: 0.9028980844683272

Testing F1: 0.7365795331299736

Train Confusion Matrix



Test Confusion Matrix



	precision	recall	f1-score	support
Bali, Indonesia	0.62	0.85	0.72	1001
Barcelona, Spain	0.63	0.59	0.61	368

Crete, Greece	0.81	0.69	0.74	244
Dubai, United Arab Emirates	0.91	0.84	0.87	760
Goa, India	0.34	0.63	0.44	68
Istanbul, Turkey	0.81	0.74	0.78	485
London, United Kingdom	0.76	0.69	0.72	563
Majorca, Balearic Islands	0.42	0.51	0.46	101
Paris, France	0.74	0.63	0.68	538
Phuket, Thailand	0.65	0.75	0.70	212
Rome, Italy	0.86	0.70	0.77	923
Sicily, Italy	0.75	0.72	0.73	413
accuracy			0.73	5676
macro avg		0.69	0.70	5676
weighted avg		0.75	0.73	5676

The bi-grams did well for the training accuracy, but not so great for the testing accuracy. Thus, this model is very overfit, and TF-IDF vectorization is the best vectorization strategy for this dataset.

▼ 1.5.6 Iteration 6: Try using a Random Forest Model

- The benefit of this is the ability to see feature importances and get more insight into how the model is working with the text data

In [192]:

```

1 vectorizer = TfidfVectorizer(analyzer='word',
                                stop_words=new_stopwords,
                                decode_error='ignore')
2
3
4 X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['lemmatized'])
5 X_test_tfidf = vectorizer.transform(X_test_preprocessed['lemmatized'])

```

executed in 212ms, finished 09:46:56 2021-01-19

In [193]:

```
1 rf = RandomForestClassifier(class_weight=weights_dict)
2 rf.fit(X_train_tfidf.todense(), y_train)
```

executed in 2m 7s, finished 09:49:04 2021-01-19

```
Out[193]: RandomForestClassifier(class_weight={'Bali, Indonesia': 0.473097441026923
4,
                                              'Barcelona, Spain': 1.27230441605021
3,
                                              'Crete, Greece': 2.42864783910997,
                                              'Dubai, United Arab Emirates': 0.644
8250397636901,
                                              'Goa, India': 6.732799525504152,
                                              'Istanbul, Turkey': 1.03837358214416
4,
                                              'London, United Kingdom': 0.84460565
47619047,
                                              'Majorca, Balearic Islands': 5.35953
7299338999,
                                              'Paris, France': 0.8795521462885479,
                                              'Phuket, Thailand': 1.91295921806538
58,
                                              'Rome, Italy': 0.4640462758564304,
                                              'Sicily, Italy': 1.189136811229834
4})
```

In [194]:

```
1 evaluate_model(rf, X_train_tfidf, X_test_tfidf)
```

executed in 6.59s, finished 09:49:49 2021-01-19

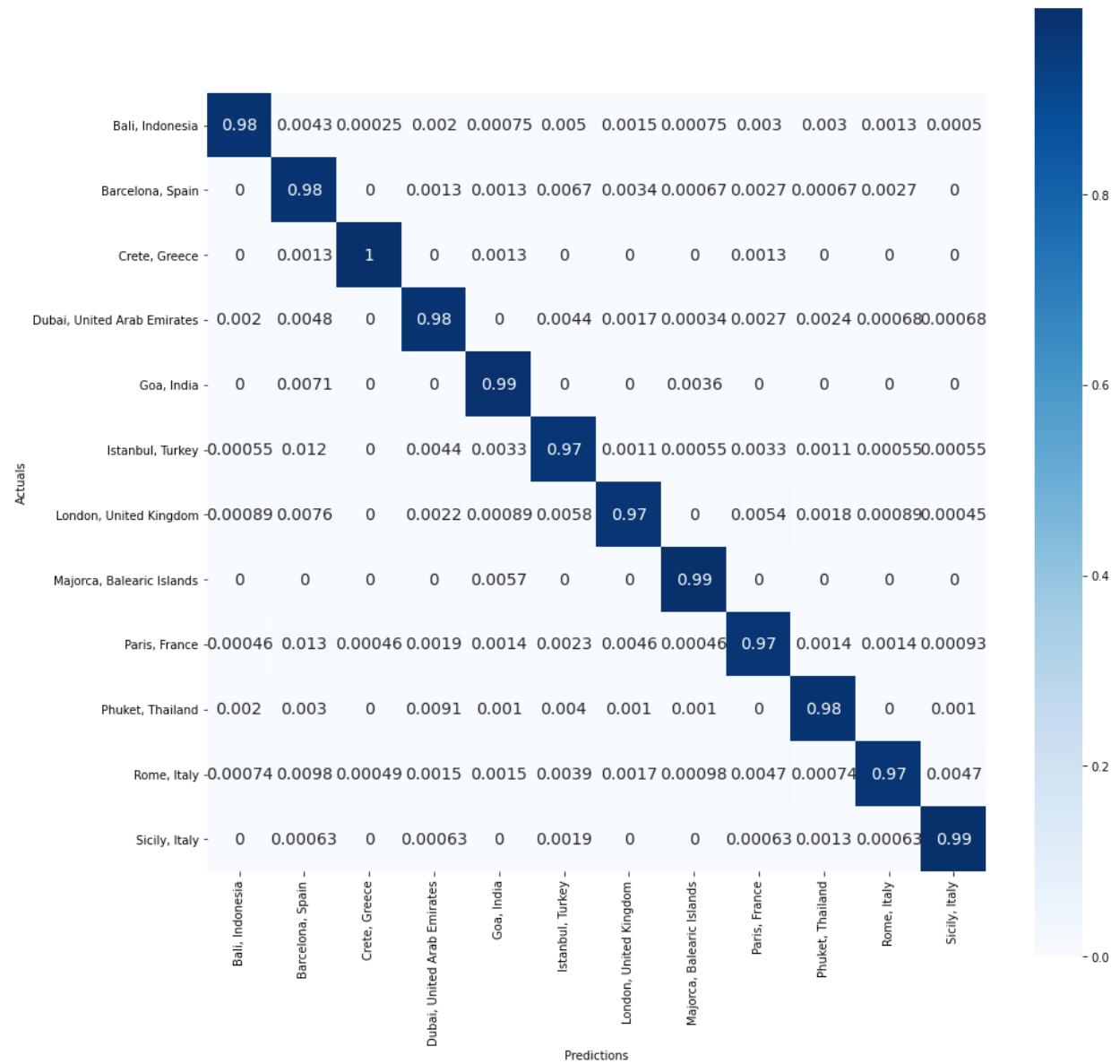
Training Accuracy: 0.9775800554992732

Testing Accuracy: 0.8053206483439042

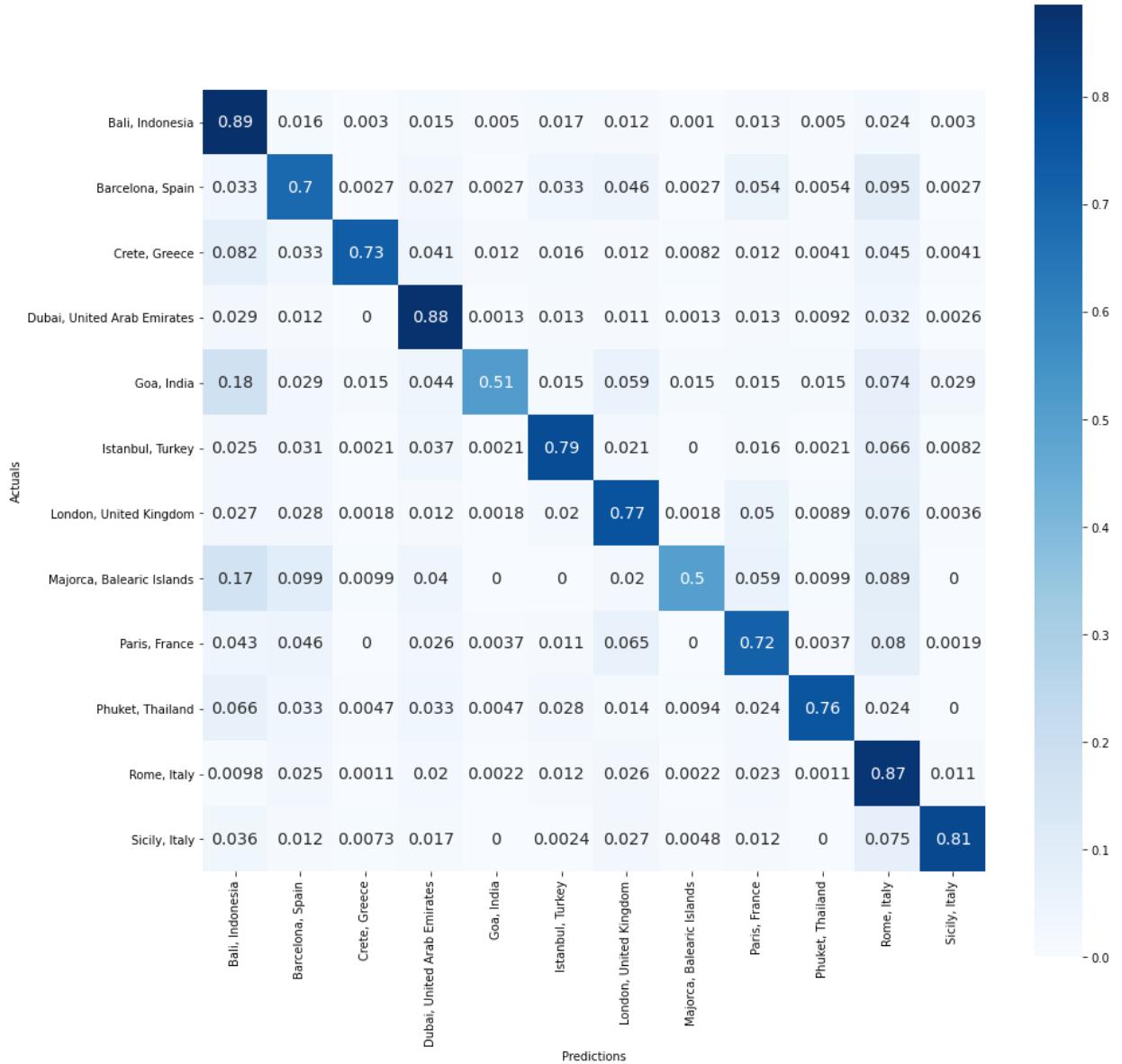
Training F1: 0.977715787875329

Testing F1: 0.8044417400712232

Train Confusion Matrix



Test Confusion Matrix



	precision	recall	f1-score	support
Bali, Indonesia	0.84	0.89	0.86	1001
Barcelona, Spain	0.65	0.70	0.67	368
Crete, Greece	0.93	0.73	0.82	244
Dubai, United Arab Emirates	0.85	0.88	0.87	760
Goa, India	0.67	0.51	0.58	68
Istanbul, Turkey	0.83	0.79	0.81	485
London, United Kingdom	0.77	0.77	0.77	563
Majorca, Balearic Islands	0.80	0.50	0.62	101
Paris, France	0.76	0.72	0.74	538
Phuket, Thailand	0.86	0.76	0.81	212
Rome, Italy	0.75	0.87	0.81	923
Sicily, Italy	0.93	0.81	0.86	413

accuracy		0.81	5676
macro avg	0.80	0.74	0.77
weighted avg	0.81	0.81	0.80

```
In [196]: 1 #Get feature importances
2 feat_imps = pd.Series(rf.feature_importances_,
3                         index=vectorizer.get_feature_names())
4 feat_imps[:11]
```

executed in 33ms, finished 09:50:01 2021-01-19

```
Out[196]: aal           6.367848e-06
abandon        1.781917e-05
abant          8.880404e-07
abbate         2.335015e-05
abbey          5.006061e-04
abbeyprivate   6.557968e-07
abbeyst        3.831503e-07
aberfan        5.211206e-06
abian          4.439020e-07
aboard         6.447375e-05
abra            3.188394e-05
dtype: float64
```

```
In [197]: 1 top_20_feats = feat_imps.sort_values(ascending=False).head(20)
2 top_20_feats
```

executed in 20ms, finished 09:50:11 2021-01-19

```
Out[197]: tour           0.018110
palma          0.015596
palermo        0.014395
chania         0.011152
ubud            0.010588
etna            0.010374
heraklion      0.010056
desert          0.009883
taormina       0.008365
day              0.008106
catania         0.008046
colosseum       0.007226
city             0.007188
safari           0.007093
de               0.006871
turkey          0.006273
pmi              0.006111
cappadocia      0.006092
vatican          0.006062
phi              0.005958
dtype: float64
```

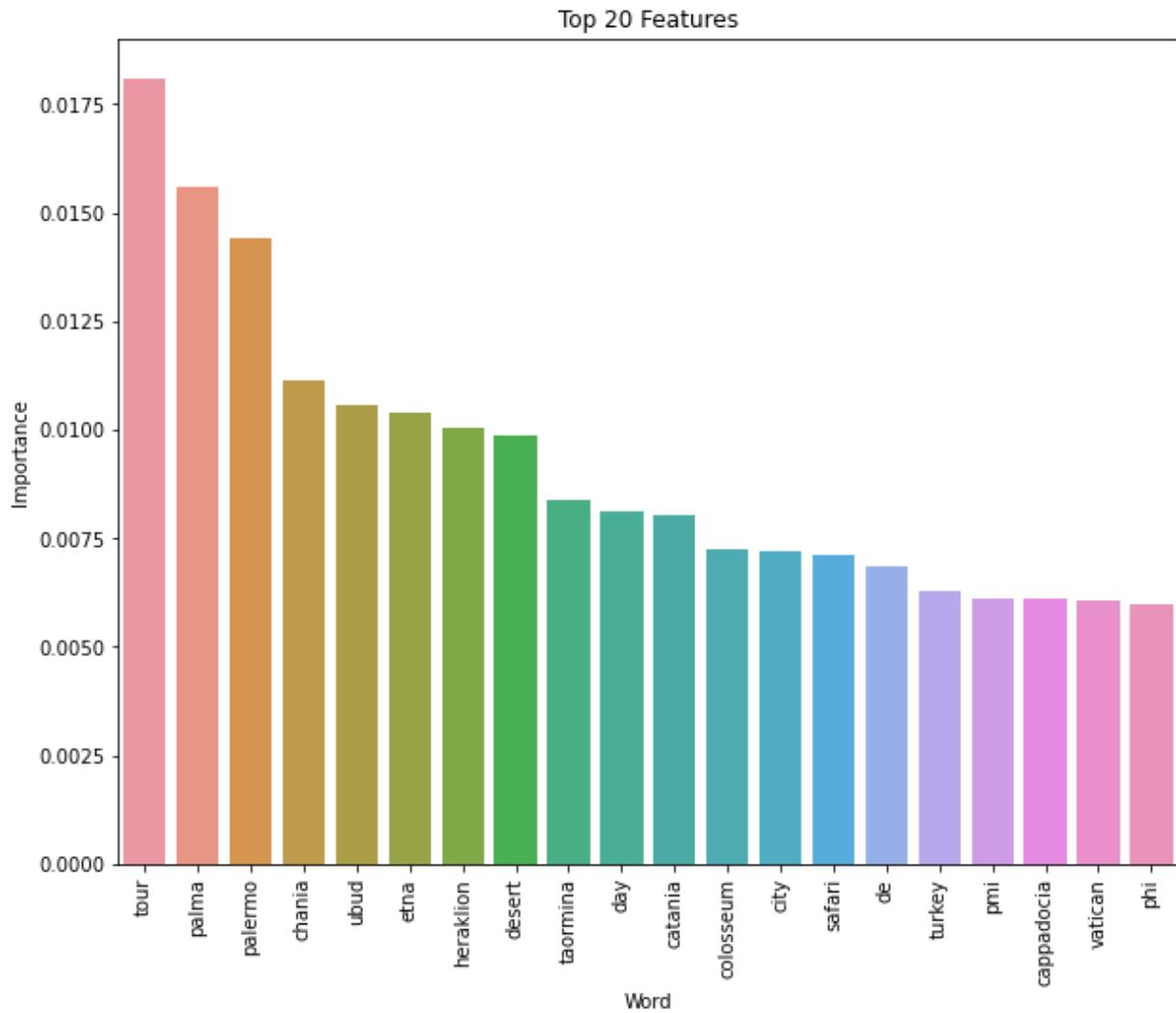
In [198]:

```

1 plt.figure(figsize=(10,8))
2 sns.barplot(x=top_20_feats.index, y=top_20_feats)
3 plt.title('Top 20 Features')
4 plt.xticks(rotation=90)
5 plt.ylabel('Importance')
6 plt.xlabel('Word')
7 plt.show()

```

executed in 173ms, finished 09:50:19 2021-01-19



This model is also overfit, even though it still performs very well with the test set.

Interestingly, the feature importances show a lot of city-specific words, such as 'etna'-- the name of a volcano in Sicily. 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.

This model's performance is very good, but random forests have 2 major flaws that will affect this model for its specific use-case:

1. They are more computationally expensive than Naive Bayes models (AKA they take longer to train and predict)
2. They use a greedy algorithm, meaning they often favor the bigger class (in this case it would predict Bali much more often than any of the other beach destinations)

For these reasons I still think iteration 3 is the best model so far.

▼ **1.5.7 Try out iteration 3 without lemmatization**

- One last thing I would like to try is using the cleaned text data without lemmatizing it. I created the preprocessing function to give me this option.

In [199]:

```

1 X_train_cleaned = preprocess_df(pd.DataFrame(X_train, columns=['Attraction'
2                                     'cleaned'], lemmatize=False))
3 X_test_cleaned = preprocess_df(pd.DataFrame(X_test, columns=['Attraction'
4                                     'cleaned'], lemmatize=False))

```

executed in 623ms, finished 09:56:07 2021-01-19

	Attraction	cleaned
2092	The Colosseum and The Ancient City of Rome	the colosseum and the ancient city of rome
814	Paintball in Canggu/Bali	paintball in canggubali
1920	9Hr Tour London Eye, Westminster Abbey and St ...	tour london eye westminster abbey and st paul...
348	Dubai 3h Sea escape: Swim! Tan! Sightsee!	dubai sea escape swim tan sightsee
3122	Private Bali Half Day Car Charter - Uluwatu Su...	private bali half day car charter uluwatu sun...
1313	Rooftop Pasta Making Class and Food Market Tou...	rooftop pasta making class and food market tou...
952	Colosseum, Forum and Baroque Squares	colosseum forum and baroque squares
972	Gothic Quarter's deepest secrets & Sangria	gothic quarters deepest secrets sangria
1622	Private Half-Day Montserrat Tour in Afternoon ...	private halfday montserrat tour in afternoon ...
1036	Changing of the Guard Half-Day Private Walking...	changing of the guard halfday private walking ...

	Attraction	cleaned
302	Phuket City Tour Fullday	phuket city tour fullday
66	Phuket: Guided Fast Track Phuket Airport	phuket guided fast track phuket airport
208	Guided Montserrat Monastery Day Tour with Hot ...	guided montserrat monastery day tour with hot ...
3846	Private Pizza & Tiramisu Class at a Cesarina's...	private pizza tiramisu class at a cesarinash...
111	Sierra Tramuntana: Mountain Tops and Cosy Vill...	sierra tramuntana mountain tops and cosy villages
1521	Bali Ubud Paon Cooking Class	bali ubud paon cooking class
277	Driver's license free boat rental	drivers license free boat rental
230	East bali tour	east bali tour
455	Private Tour: Istanbul Sightseeing Including Mu...	private tour istanbul sightseeing including mu...
1564	Catania Private Walking Tour	catania private walking tour

In [200]:

```

1 vectorizer = TfidfVectorizer(analyzer='word',
2                               stop_words=new_stopwords,
3                               decode_error='ignore')
4 X_train_tfidf = vectorizer.fit_transform(X_train_cleaned['cleaned'])
5 X_test_tfidf = vectorizer.transform(X_test_cleaned['cleaned'])

```

executed in 382ms, finished 09:56:11 2021-01-19

In [201]:

```
1 nb_cleaned = MultinomialNB()
2 nb_cleaned.fit(X_train_tfidf.todense(),
3                  y_train,
4                  sample_weight=sample_weights)
```

executed in 1.32s, finished 09:56:17 2021-01-19

Out[201]: MultinomialNB()

```
In [202]: 1 evaluate_model(nb_cleaned, X_train_tfidf, X_test_tfidf)
```

executed in 2.47s, finished 09:56:20 2021-01-19

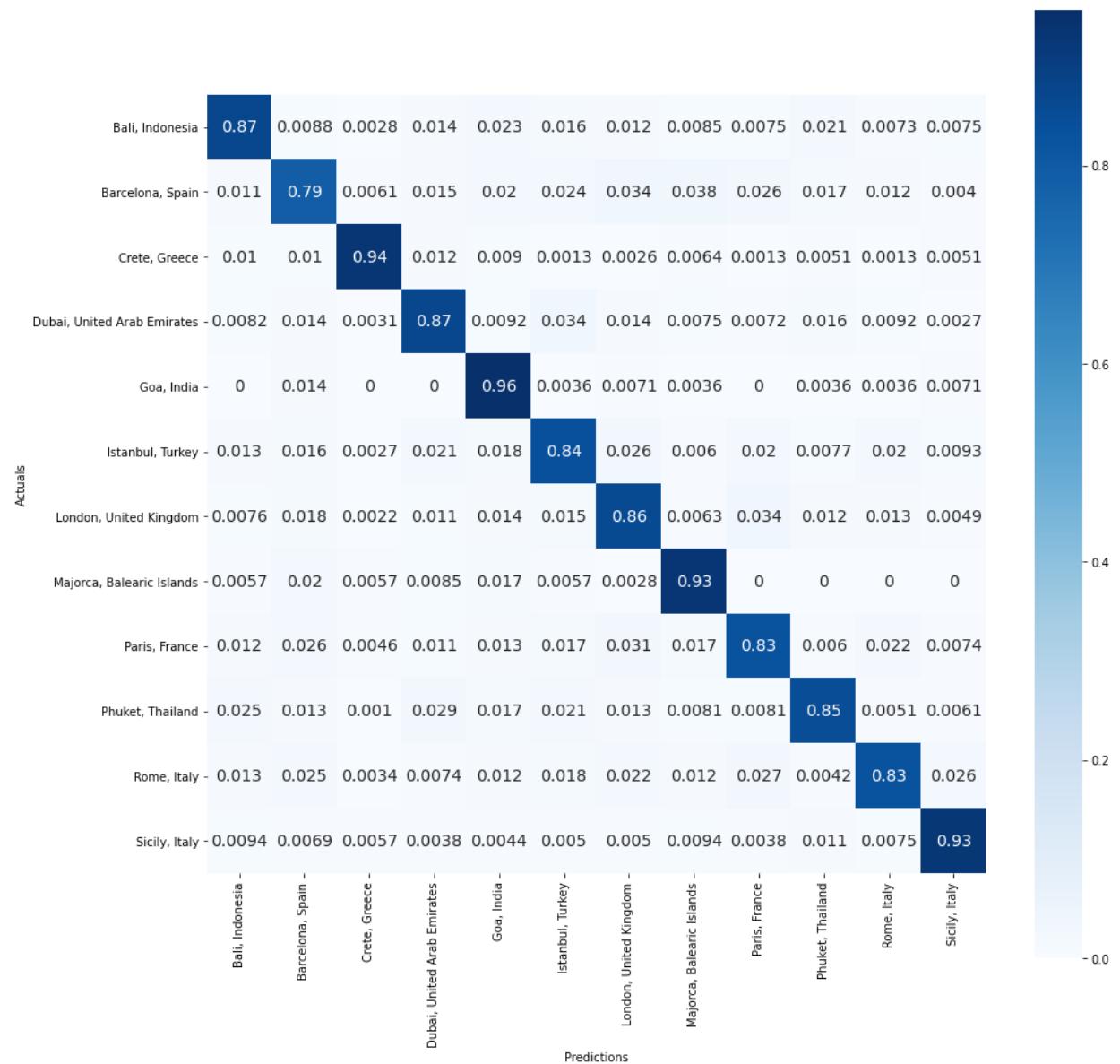
Training Accuracy: 0.8598863586310179

Testing Accuracy: 0.8146582100070472

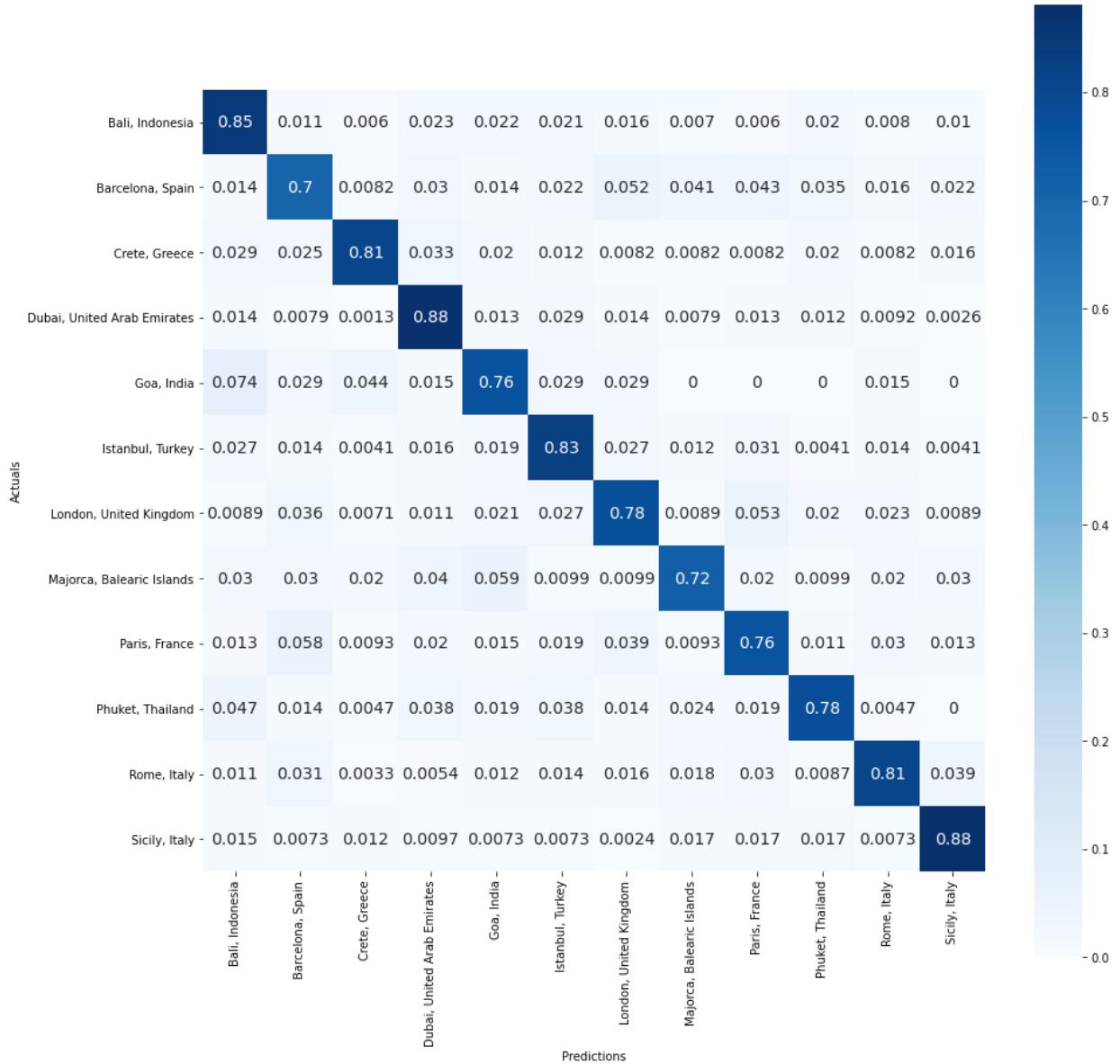
Training F1: 0.8631488745947244

Testing F1: 0.8190645588523411

Train Confusion Matrix



Test Confusion Matrix



	precision	recall	f1-score	support
Bali, Indonesia	0.91	0.85	0.88	1001
Barcelona, Spain	0.68	0.70	0.69	368
Crete, Greece	0.85	0.81	0.83	244
Dubai, United Arab Emirates	0.88	0.88	0.88	760
Goa, India	0.35	0.76	0.48	68
Istanbul, Turkey	0.79	0.83	0.81	485
London, United Kingdom	0.81	0.78	0.79	563
Majorca, Balearic Islands	0.49	0.72	0.59	101
Paris, France	0.77	0.76	0.77	538
Phuket, Thailand	0.67	0.78	0.72	212
Rome, Italy	0.92	0.81	0.86	923
Sicily, Italy	0.83	0.88	0.85	413
accuracy			0.81	5676
macro avg		0.75	0.80	0.76
				5676

Ultimately, this model performed VERY similar to the lemmatized version. There are a couple of small differences that made me choose this version as my final model:

- 1) The test accuracy and F1 scores are a tiny bit higher for this model compared to the lemmatized version. Even though it is only one percent higher overall, the breakdown under each city has increased some of the smaller classes, such as Goa and Phuket.
- 2) Lemmatization is more computationally expensive than omitting the lemmatization. It is a small difference, but should still be a consideration.

Therefore, my final model is iteration 3 (Naive Bayes) without lemmatizing the text.

In [203]:

```
1 # Pickle the best Naive Bayes Model
2 with open('/Users/tiaplagata/Documents/Flatiron/capstone-project/non_le
3         'wb') as f:
4     pickle.dump(nb_cleaned, f, pickle.HIGHEST_PROTOCOL)
```

executed in 9ms, finished 09:58:41 2021-01-19

1.6 Test out the model

- I will ultimately use this model to tell people where they should travel based on what they want to do while on vacation. Let's look at some of the sample predictions this model would give them, using iteration 3 as our final model.

In [204]:

```
1 def preprocess_text(text):
2     """
3     Input raw text.
4     Return preprocessed text.
5     """
6
7     preprocessed = text.lower()
8     preprocessed = re.sub('[%s]' % re.escape(string.punctuation), '', p
9     preprocessed = re.sub('\w*\d\w*', '', preprocessed)
10
11    return [preprocessed]
```

executed in 4ms, finished 10:00:44 2021-01-19

In [205]:

```
1 raw_text = 'I want to go to the beach, go hiking and snorkeling'
2 preprocessed_text = preprocess_text(raw_text)
3 preprocessed_text
```

executed in 11ms, finished 10:00:47 2021-01-19

Out[205]: ['i want to go to the beach go hiking and snorkeling']

In [206]:

```
1 nb_cleaned.predict(vectorizer.transform(preprocessed_text))
```

executed in 13ms, finished 10:00:49 2021-01-19

Out[206]: array(['Bali, Indonesia'], dtype='<U27')

```
In [207]: 1 preprocessed2 = preprocess_text('Go to historic museums')
2 print(preprocessed2)
3 nb_cleaned.predict(vectorizer.transform(preprocessed2))
executed in 14ms, finished 10:00:50 2021-01-19
['go to historic museums']
```

```
Out[207]: array(['Rome, Italy'], dtype='<U27')
```

```
In [208]: 1 preprocessed3 = preprocess_text('Wine tastings, long walks and dinners')
2 print(preprocessed3)
3 nb_cleaned.predict(vectorizer.transform(preprocessed3))
executed in 12ms, finished 10:00:51 2021-01-19
```

```
['wine tastings long walks and dinners']
```

```
Out[208]: array(['Crete, Greece'], dtype='<U27')
```

```
In [209]: 1 preprocessed4 = preprocess_text('Do yoga on the beach')
2 print(preprocessed4)
3 nb_cleaned.predict(vectorizer.transform(preprocessed4))
executed in 11ms, finished 10:00:55 2021-01-19
```

```
['do yoga on the beach']
```

```
Out[209]: array(['Goa, India'], dtype='<U27')
```

```
In [210]: 1 preprocessed5 = preprocess_text('Sunset cruises on a yacht with wine')
2 print(preprocessed5)
3 nb_cleaned.predict(vectorizer.transform(preprocessed5))
executed in 14ms, finished 10:00:55 2021-01-19
```

```
['sunset cruises on a yacht with wine']
```

```
Out[210]: array(['Crete, Greece'], dtype='<U27')
```

▼ **1.6.1 Make this process into a pipeline**

In [211]:

```

1 # Use OOP to get preprocessing steps into a pipeline
2 class PreprocessText(TransformerMixin):
3
4     def __init__(self):
5         self = self
6
7     def fit(self, X, y=None, **fit_params):
8         return self
9
10    def transform(self, X, **transform_params):
11        try:
12            X = pd.DataFrame(X, columns=['Attraction'])
13            X['cleaned'] = X['Attraction'].apply(lambda x: x.lower())
14            X['cleaned'] = X['cleaned'].apply(lambda x: re.sub('[%s]' %
15            X['cleaned'] = X['cleaned'].apply(lambda x: re.sub('\w*\d\w',
16
17            X = X['cleaned']
18        except:
19            pass
20        return X
21
22 class DenseTransformer():
23
24     def __init__(self):
25         self = self
26
27     def fit(self, X, y=None, **fit_params):
28         return self
29
30     def transform(self, X, y=None, **fit_params):
31         return X.todense()

```

executed in 19ms, finished 10:01:00 2021-01-19

In [212]:

```

1 # Test preprocessing class
2 prep = PreprocessText()
3 prep.transform(X_train)

```

executed in 519ms, finished 10:01:01 2021-01-19

Out[212]:

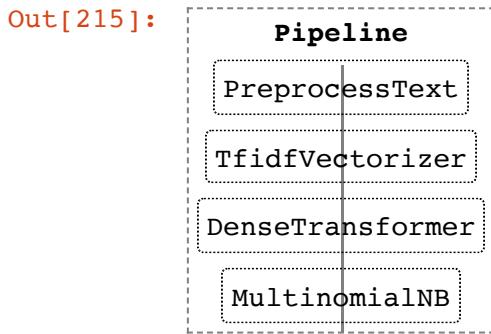
2092	the colosseum and the ancient city of rome
814	paintball in canggubali
1920	tour london eye westminster abbey and st paul...
348	dubai sea escape swim tan sightsee
3122	private bali half day car charter uluwatu sun...
	...
1428	barcelona city la roca village tour
14	etna wine and alcantara tour small groups fro...
3345	fujiearah east coast tour
4210	early morning vatican museums sistine chapel s...
246	rafa nadal museum mallorca half day tour

Name: cleaned, Length: 22703, dtype: object

```
In [213]: 1 pipe = Pipeline(steps=[  
2             ('TextPreprocessor', PreprocessText()),  
3             ('TfidfVectorizer', TfidfVectorizer(analyzer='word',  
4                                         stop_words=new_stop  
5                                         decode_error='ignore'))  
6             ('DenseTransformer', DenseTransformer()),  
7             ('NaiveBayes', MultinomialNB())])  
executed in 2ms, finished 10:01:01 2021-01-19
```

```
In [214]: 1 set_config(display='diagram')  
executed in 6ms, finished 10:01:02 2021-01-19
```

```
In [215]: 1 pipe.fit(X_train,  
2                  y_train,  
3                  **{'NaiveBayes__sample_weight': sample_weights})  
executed in 2.08s, finished 10:01:05 2021-01-19
```



```
In [216]: 1 pipe.score(X_test, y_test)  
executed in 355ms, finished 10:01:07 2021-01-19
```

Out[216]: 0.8146582100070472

```
In [217]: 1 pipe.predict(['I want to go snorkeling and tan on the beach'])  
executed in 36ms, finished 10:01:09 2021-01-19
```

Out[217]: array(['Bali, Indonesia'], dtype='<U27')

```
In [218]: 1 pipe.predict(['Go out for drinks'])  
executed in 15ms, finished 10:01:19 2021-01-19
```

Out[218]: array(['Barcelona, Spain'], dtype='<U27')

In [230]:

```
1 def evaluate_pipe(pipe, X_train, X_test):
2     y_preds_train = pipe.predict(X_train)
3     y_preds_test = pipe.predict(X_test)
4
5     print('Training Accuracy:', accuracy_score(y_train, y_preds_train))
6     print('Testing Accuracy:', accuracy_score(y_test, y_preds_test))
7     print('\n-----\n')
8     print('Training F1:', f1_score(y_train, y_preds_train, average='wei')
9     print('Testing F1:', f1_score(y_test, y_preds_test, average='weight'
10    print('\n-----\n')
11    print('Train Confusion Matrix\n')
12    plot_conf_matrix(y_train, y_preds_train)
13    print('Test Confusion Matrix\n')
14    plot_conf_matrix(y_test, y_preds_test)
15    print('\n-----\n')
16    print(classification_report(y_test, y_preds_test))
```

executed in 9ms, finished 10:18:59 2021-01-19

```
In [231]: 1 evaluate_pipe(pipe, X_train, X_test)
```

executed in 3.23s, finished 10:19:03 2021-01-19

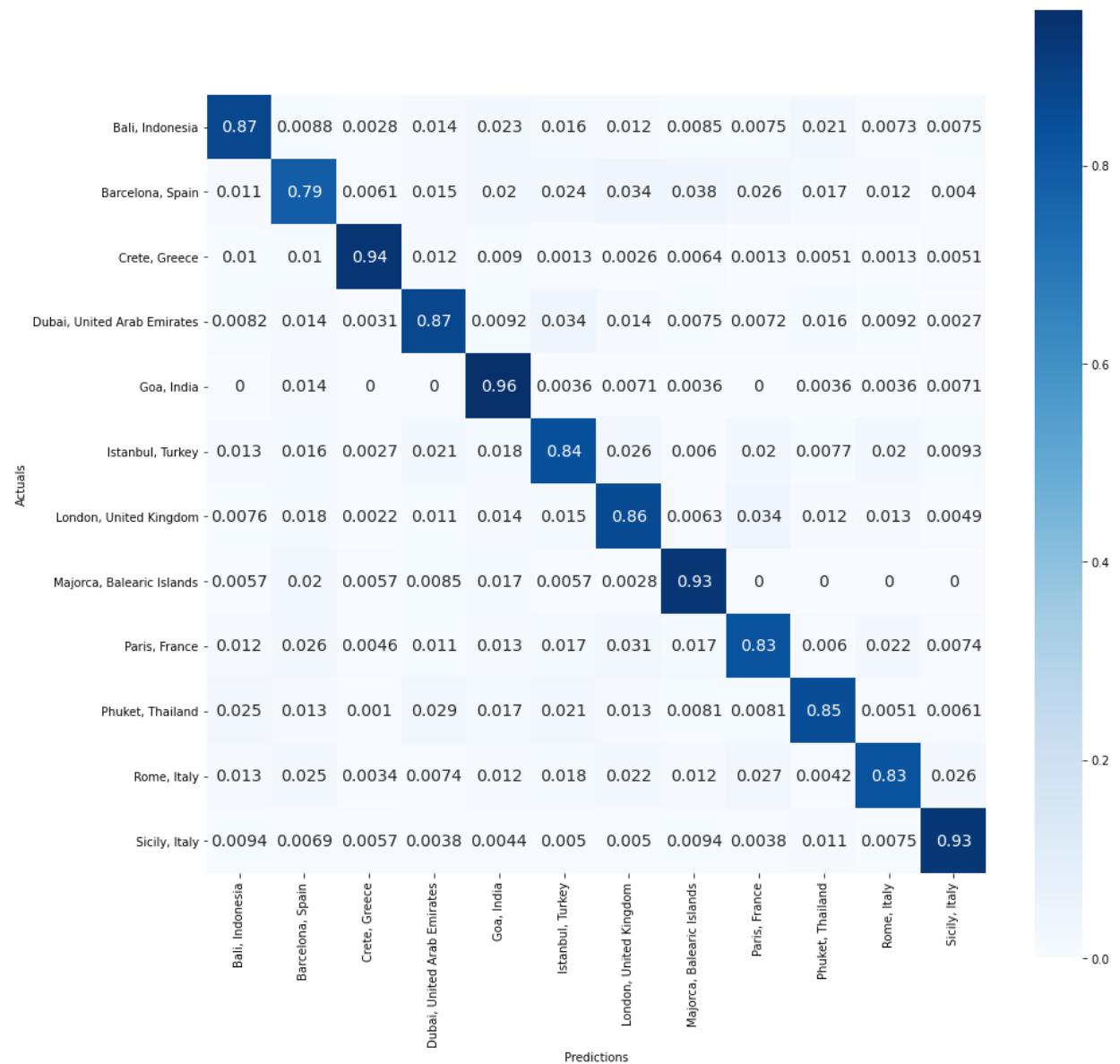
Training Accuracy: 0.8598863586310179

Testing Accuracy: 0.8146582100070472

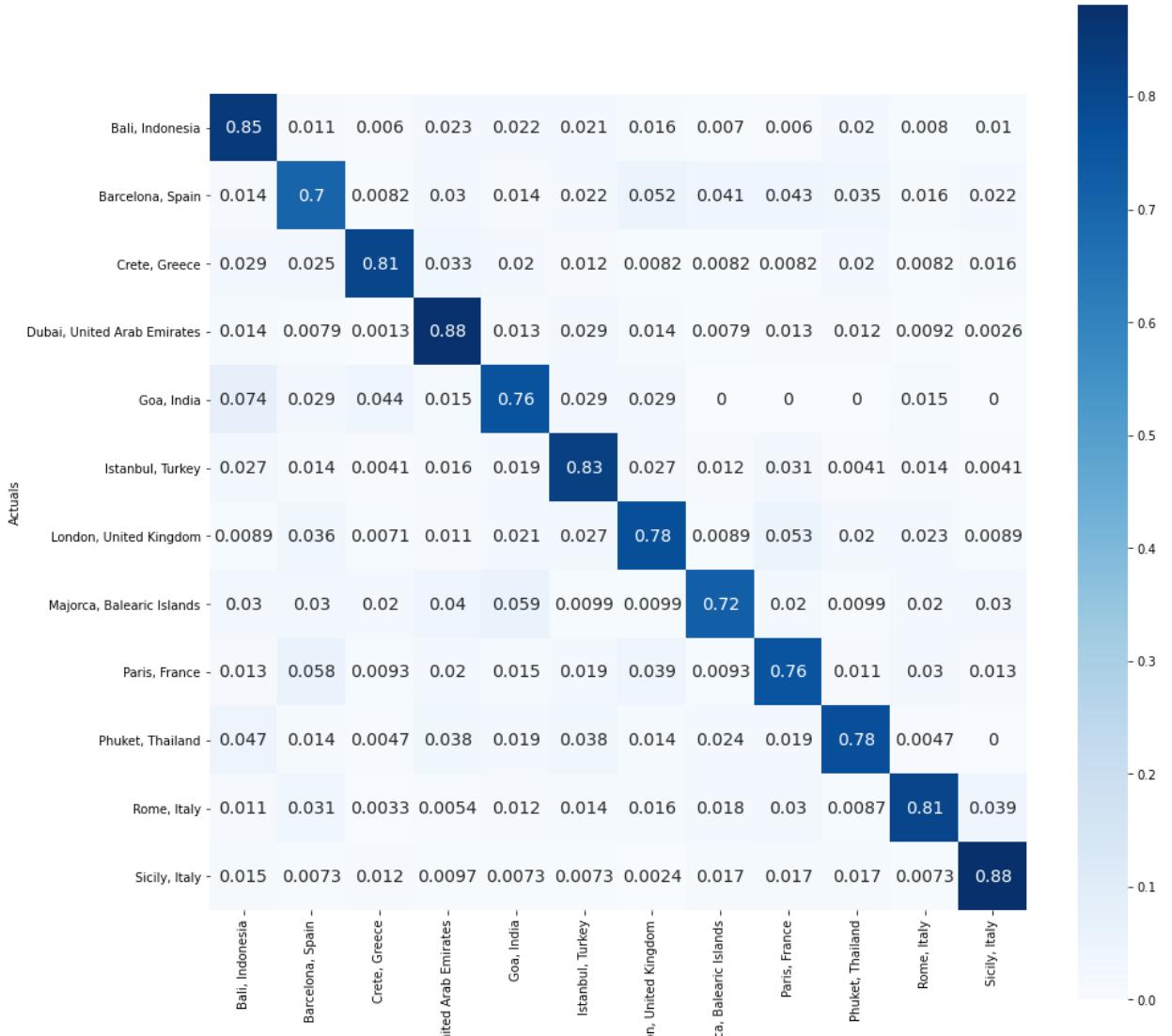
Training F1: 0.8631488745947244

Testing F1: 0.8190645588523411

Train Confusion Matrix



Test Confusion Matrix



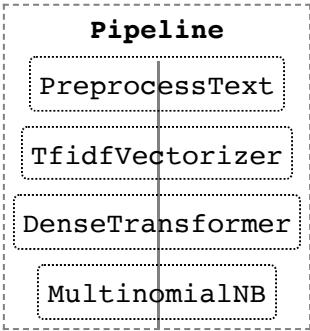
	precision	recall	f1-score	support
Bali, Indonesia	0.91	0.85	0.88	1001
Barcelona, Spain	0.68	0.70	0.69	368
Crete, Greece	0.85	0.81	0.83	244
Dubai, United Arab Emirates	0.88	0.88	0.88	760
Goa, India	0.35	0.76	0.48	68
Istanbul, Turkey	0.79	0.83	0.81	485
London, United Kingdom	0.81	0.78	0.79	563
Majorca, Balearic Islands	0.49	0.72	0.59	101
Paris, France	0.77	0.76	0.77	538
Phuket, Thailand	0.67	0.78	0.72	212
Rome, Italy	0.92	0.81	0.86	923
Sicily, Italy	0.83	0.88	0.85	413
accuracy				0.81
macro avg				0.76
weighted avg				0.82

```
In [221]: 1 # Pickle the Pipeline
2 with open('/Users/tiaplagata/Documents/Flatiron/capstone-project/final_
3           'wb') as f:
4     pickle.dump(pipe, f, pickle.HIGHEST_PROTOCOL)
executed in 16ms, finished 10:01:44 2021-01-19
```



```
In [222]: 1 # Load pipe from pickled file to test
2 with open('/Users/tiaplagata/Documents/Flatiron/capstone-project/final_
3           'rb') as f:
4     best_model_pipe = pickle.load(f)
5 best_model_pipe
executed in 31ms, finished 10:01:44 2021-01-19
```


Out[222]:



```
Pipeline
+-- PreprocessText
+-- TfidfVectorizer
+-- DenseTransformer
+-- MultinomialNB
```



```
In [223]: 1 best_model_pipe.predict(['I want to visit art galleries'])
executed in 16ms, finished 10:01:46 2021-01-19
```


Out[223]: array(['Paris, France'], dtype='<U27')

▼ 1.6.2 Get top 2 predictions from best model

- In the dash app, it would be good to give someone a second prediction just in case they have already been to the first place the model predicts.

```
In [224]: 1 probas = best_model_pipe.predict_proba(['I want to visit art galleries']
2 probas
executed in 17ms, finished 10:01:50 2021-01-19
```

```
Out[224]: array([[0.08062311, 0.10560993, 0.04660718, 0.03027719, 0.12294859,
0.03850689, 0.16403276, 0.06665658, 0.18981578, 0.01453859,
0.0769455 , 0.06343788]])
```

```
In [225]: 1 classes = best_model_pipe.classes_
2 classes
executed in 8ms, finished 10:01:51 2021-01-19
```

```
Out[225]: array(['Bali, Indonesia', 'Barcelona, Spain', 'Crete, Greece',
'Dubai, United Arab Emirates', 'Goa, India', 'Istanbul, Turkey',
'London, United Kingdom', 'Majorca, Balearic Islands',
'Paris, France', 'Phuket, Thailand', 'Rome, Italy',
'Sicily, Italy'], dtype='<U27')
```

```
In [226]: 1 # First Prediction
2 classes[probas.argmax()]
executed in 3ms, finished 10:01:52 2021-01-19
```

```
Out[226]: 'Paris, France'
```

```
In [227]: 1 # Second Prediction
2 classes[np.argsort(probas)[:, 10]][0]
executed in 9ms, finished 10:01:53 2021-01-19
```

```
Out[227]: 'London, United Kingdom'
```

▼ 1.7 Conclusion

Moving forward, the final model is iteration 3 without lemmatization. The final accuracy score and F1 score are both 81%.

Everything works and is ready to be put into a Dash app for its final use-case: predicting where people should travel based on the activities that they want to do on vacation!