1 Capstone Project

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

- Tausif Khan
- https://github.com/tausifKhan21
- tausifkit@gmail.com (mailto:tausifkit@gmail.com)

Table of Contents

2 Project Overview

The COVID-19 pandemic has severely affected the travel industry. International travel has been impacted, and in turn travel companies and travel websites have lost much of their engagement.

However, with the development of new vaccines for the virus, there is hope on the horizon for international travel and a time where life is somewhat back to normal. In order to increase engagement in the travel industry and increase excitement about travel opportunities, the Destination Dictionary was born!

The Destination Dictionary is a data product that allows future travelers to get a prediction for their perfect destination with the input of just a few words. Trained on over 28,000 unique text data points, the Destination Dictionary is able to predict a destination from 12 different popular cities with 81% accuracy based on text input of activities you want to do while on vacation.

2.0.1 Methodology & Data Used

This project utilized data from 12 top cities from TripAdvisor's list of Traveler's Choice destinations for Popular World Destinations 2020, which can be found via [this link (https://www.tripadvisor.com/TravelersChoice-Destinations)](https://www.tripadvisor.com/TravelersChoice-Destinations). The dataset was compiled by scraping the titles from Tripadvisor 'attractions' for each of the 12 cities. The final dataset included over 28,000 unique text values.

In [145]:

```
# Import Statements

import pandas as pd
import numpy as np

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

import string
import regex as re
import spacy

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, TfidfVecto
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, con
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 pickle
```

executed in 12ms, finished 08:59:29 2021-01-26

In [20]:

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

executed in 39ms, finished 08:37:49 2021-01-26

Out[20]:

	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

2.1 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 [21]:
```
df.info()
```
executed in 7ms, finished 08:37:49 2021-01-26

```
<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 [22]:
```
df.describe()
```
executed in 21ms, finished 08:37:49 2021-01-26

Out[22]:

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

In [23]:
```
df.shape
```
executed in 2ms, finished 08:37:49 2021-01-26

Out[23]: (28379, 2)

In [24]:
```
#No null values because I scraped everything myself. Just to double-che
df.isna().sum()
```
executed in 5ms, finished 08:37:49 2021-01-26

Out[24]:
```
Attraction    0
City          0
dtype: int64
```

2.1.1 Duplicates

In [25]:
```
df.duplicated().sum()
```
executed in 13ms, finished 08:37:49 2021-01-26

Out[25]: 846

In [26]:

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

executed in 20ms, finished 08:37:49 2021-01-26

Out[26]:

	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

	Attraction	City
1622	Freestyle Football Workshop in England	London, United Kingdom
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 [27]:
```
df[df['Attraction']=='Oxford City Full-Day Private Tour from London']
```
executed in 6ms, finished 08:37:49 2021-01-26

Out[27]:

	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 [28]:
```
# What about the top attraction
df[df['Attraction']=='Desert Sa ari Dubai']
```
executed in 6ms, finished 08:37:49 2021-01-26

Out[28]:

	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 [29]:
```
df = df.drop_duplicates()
```
executed in 13ms, finished 08:37:49 2021-01-26

In [30]:
```
# df.to_csv('/Users/tiaplagata/Documents/Flatiron/capstone-project/Data
```
executed in 2ms, finished 08:37:49 2021-01-26

2.1.2 Class Imbalance

In [31]:
```
display(df.City.unique())
print('Total Unique Cities:', len(df.City.unique()))
```
executed in 5ms, finished 08:37:49 2021-01-26

```
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 [32]:
```
df.City.value_counts(normalize=True)
```
executed in 6ms, finished 08:37:49 2021-01-26

Out[32]:
```
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 [33]:
```
cities = df.groupby('City').count()
```
executed in 7ms, finished 08:37:49 2021-01-26

In [34]:
```
cities.reset_index(inplace=True)
```
executed in 3ms, finished 08:37:49 2021-01-26

In [9]:

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

executed in 6ms, finished 08:37:49 2021-01-26

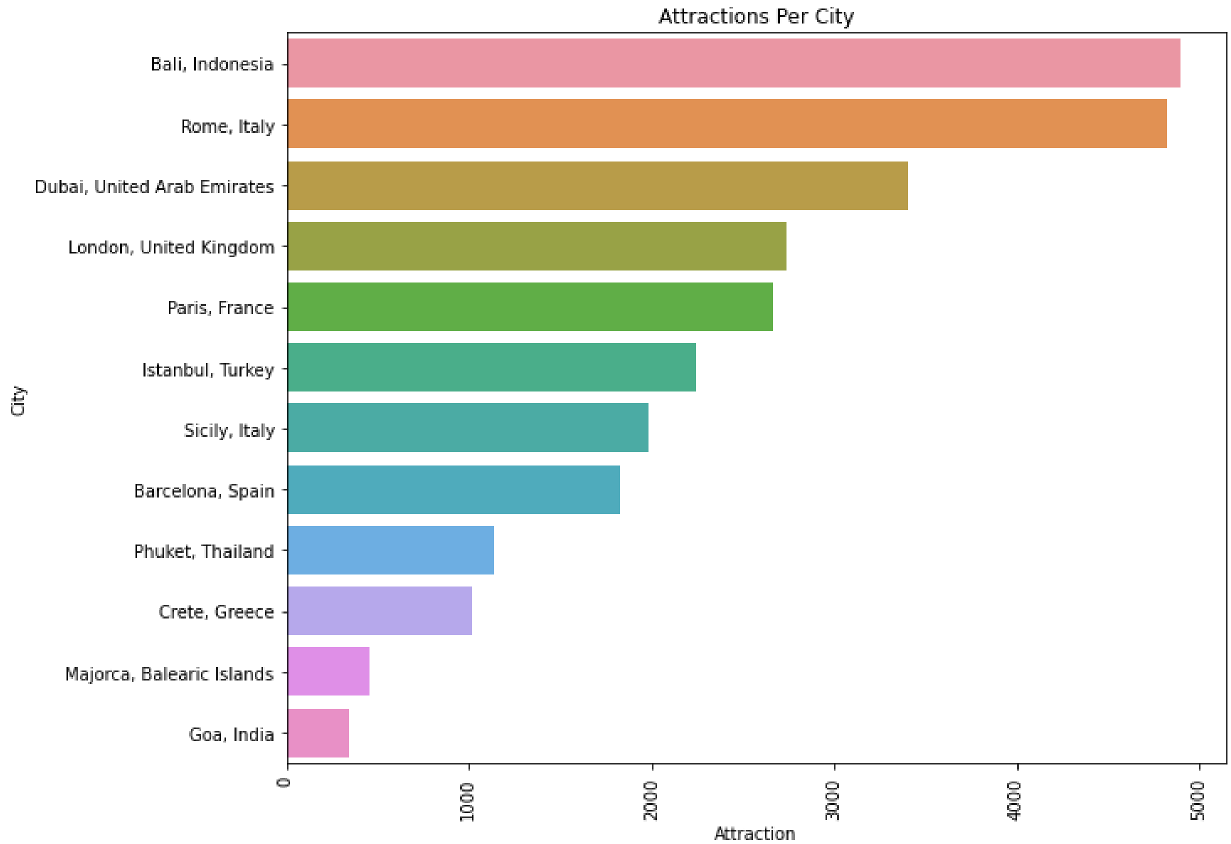
Out[35]:

	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 [10]:

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

executed in 122ms, finished 08:37:49 2021-01-26



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

2.2 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 [11]:
```
# Create a list of stopwords
stopwords_list = stopwords.words('english')
stopwords_list += list(string.punctuation)
```
executed in 5ms, finished 08:37:49 2021-01-26

In [38]:
```
# Preview the list
stopwords_list[:10]
```
executed in 2ms, finished 08:37:49 2021-01-26

Out[38]:
```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "yo
u're"]
```

In [39]:
```
# Lowercase all words in each corpus
df['cleaned'] = df['Attraction'].apply(lambda x: x.lower())
df.head()
```
executed in 15ms, finished 08:37:49 2021-01-26

Out[39]:

	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 [40]:
```
# Remove commas, hyphens, colons, and other punctuation
df['cleaned'] = df['cleaned'].apply(lambda x: re.sub('[%s]' % re.escape
df.head()
```
executed in 281ms, finished 08:37:50 2021-01-26

Out[40]:

	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 [41]:

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

executed in 328ms, finished 08:37:50 2021-01-26

Out[41]:

	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 [42]:

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

executed in 1ms, finished 08:37:50 2021-01-26

In [13]:

```
# Lemmatize the text using spacy
nlp = spacy.load('en')

df['lemmatized'] = df['cleaned'].apply(lambda x: ' '.join(
                                        [token.lemma_ for token in list(nlp
df.head(10)
```

executed in 1m 42.7s, finished 08:39:33 2021-01-26

1	The Jack The Ripper Walking Tour in 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
5	London Ghost and Infamous Murders Walking Tour	London, United Kingdom	london ghost and infamous murders walking tour	london ghost infamous murder walk tour
6	Stonehenge, Windsor Castle, and Bath from London	London, United Kingdom	stonehenge windsor castle and bath from london	stonehenge windsor castle bath london
7	Warner Bros. Studio: The	London, United	warner bros studio the	warner bros studio making

In [14]:

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

executed in 21ms, finished 08:39:33 2021-01-26

Out[44]:

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...

2.2.1 Look at different vectorization strategies

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

In [15]:

```
# 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())
df_dtm.index = df_grouped.index
df_dtm
```

executed in 124ms, finished 08:39:33 2021-01-26

Out[45]:

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

12 rows × 8676 columns

In [16]:

```
# Create a document term matrix using TF-IDF vectorization
# Might be good for classifying cities
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
df_dtm2.index = df_grouped.index
df_dtm2
```

executed in 127ms, finished 08:39:33 2021-01-26

Out[46]:

City	aal	abandon	abant	abba	abbate	abbey	abbeyprivate	abbeyst	aberfan
Bali, Indonesia	0.000000	0.000699	0.00000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Barcelona, Spain	0.000737	0.000000	0.00000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Crete, Greece	0.000000	0.000000	0.00000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Dubai, United Arab Emirates	0.000000	0.000310	0.00000	0.0000	0.000000	0.000000	0.0000	0.000	0.0000
Goa, India	0.000000	0.000000	0.00000	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 [47]:

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

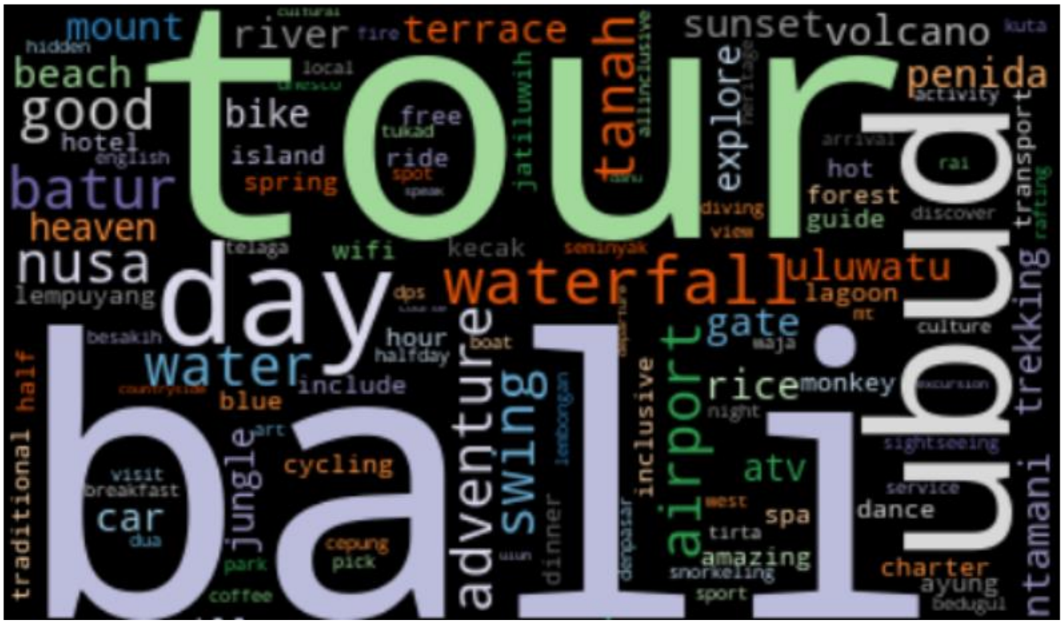
executed in 3ms, finished 08:39:33 2021-01-26

In [48]:

```
# Transposing document term matrix
df_dtm = df_dtm.transpose()

# Plotting word cloud for each city
for index, city in enumerate(df_dtm.columns):
    generate_wordcloud(df_dtm[city].sort_values(ascending=False), city)
```

executed in 2.77s, finished 08:39:36 2021-01-26

Bali, Indonesia



In [49]:

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

executed in 33ms, finished 08:39:36 2021-01-26

Out[49]:

```
[('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),
```

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.

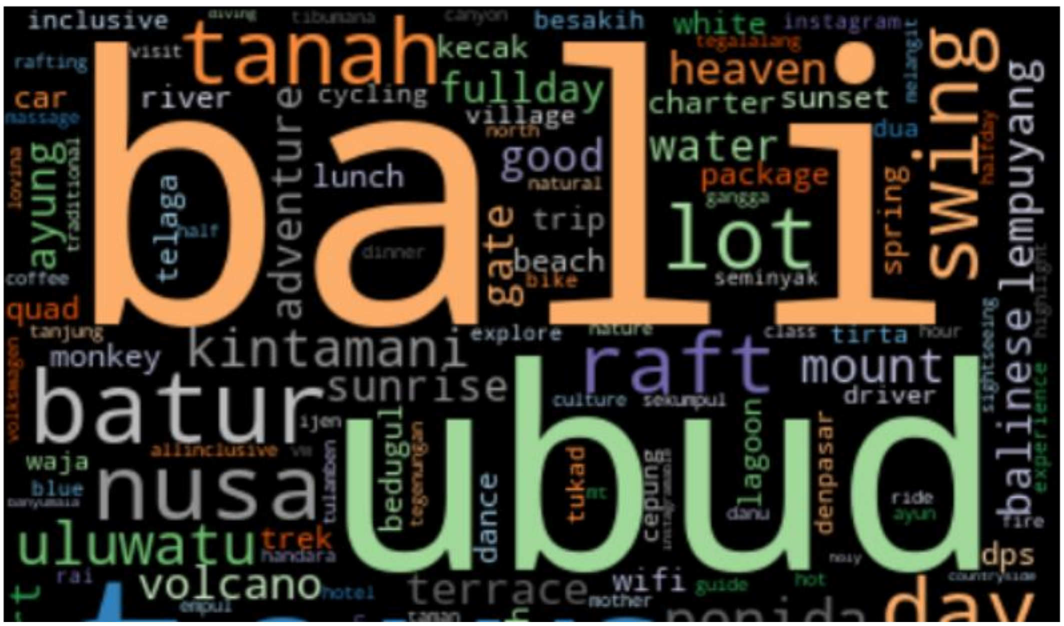
Word Clouds with TF-IDF Vectorization

In [50]:

```
# Transposing document term matrix
df_dtm2 = df_dtm2.transpose()

# Plotting word cloud for each city
for index, city in enumerate(df_dtm2.columns):
    generate_wordcloud(df_dtm2[city].sort_values(ascending=False), city
```

executed in 2.73s, finished 08:39:39 2021-01-26

Bali, Indonesia



In [19]:
```
# Look at top words with tf-idf vectorization (for total words, not per
sum_words = data2.sum(axis=0)
words_freq = [(word, sum_words[0, idx]) for word, idx in tfidf.vocabula
words_freq = sorted(words_freq, key=lambda x: x[1], reverse=True)
words_freq
```
executed in 34ms, finished 08:39:39 2021-01-26

Out[51]:
```
[('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 [20]:
```
cv2 = CountVectorizer(analyzer='word', stop_words=stopwords_list, ngram
data3 = cv2.fit_transform(df_grouped['lemmatized'])
df_dtm3 = pd.DataFrame(data3.toarray(), columns=cv2.get_feature_names()
df_dtm3.index = df_grouped.index
df_dtm3
```
executed in 259ms, finished 08:39:39 2021-01-26

Out[52]:

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

In [53]:
```
# Transposing document term matrix
df_dtm3 = df_dtm3.transpose()

# Plotting word cloud for each city
for index, city in enumerate(df_dtm3.columns):
    generate_wordcloud(df_dtm3[city].sort_values(ascending=False), city
```
executed in 3.02s, finished 08:39:42 2021-01-26

In [21]:

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

executed in 68ms, finished 08:39:42 2021-01-26

Out[54]:

```
[('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.

2.3 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 [22]:
```
df.head()
```
executed in 5ms, finished 08:39:42 2021-01-26

Out[55]:

	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 [23]:
```
# Preview what I want to drop
df.loc[df['Attraction'].str.contains('airport')]
```
executed in 17ms, finished 08:39:42 2021-01-26

Out[56]:

	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 [57]:
```
# Get rid of the airport transfer 'attractions'
df2 = df.drop(df.loc[df['Attraction'].str.contains('airport')].index)
```
executed in 17ms, finished 08:39:42 2021-01-26

In [58]:
```
df2 = df.drop(df2.loc[df2['Attraction'].str.contains('transfer')].index
```
executed in 18ms, finished 08:39:42 2021-01-26

In [59]:
```
# Just in case, add these words to the stopwords list
stopwords_list += ['airport', 'transfer', 'private']
```
executed in 1ms, finished 08:39:42 2021-01-26

In [60]:
```
print(df.shape)
print(df2.shape)
```
executed in 2ms, finished 08:39:42 2021-01-26

```
(27533, 4)
(25315, 4)
```

Create some functions to make the preprocessing steps easier

In [61]:

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

    df[column] = df['Attraction'].apply(lambda x: x.lower())
     df[column] = df[column].apply(lambda x: re.sub('[%s]' % re.escape(s
    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
    if preview:
        display(df.head(10))

    return df
```

executed in 4ms, finished 08:39:42 2021-01-26

In [62]:

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

executed in 1ms, finished 08:39:42 2021-01-26

In [63]:

```
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.
    """
    df_grouped = group_text_per_city(df, column)

    if count_vec:
        vec = CountVectorizer(analyzer='word', stop_words=stopwords_lis
    else:
        vec = TfidfVectorizer(analyzer='word', stop_words=stopwords_lis

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

executed in 4ms, finished 08:39:42 2021-01-26

In [64]:

```python
preprocessed_df = preprocess_df(df2, 'lemmatized')
dtm_cv = create_doc_term_matrix(preprocessed_df, 'lemmatized', count_ve

for index, city in enumerate(dtm_cv.columns):
    generate_wordcloud(dtm_cv[city].sort_values(ascending=False), city)
```

executed in 1m 38.5s, finished 08:41:21 2021-01-26

	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 [65]:

```python
dtm_tfidf = create_doc_term_matrix(df2, 'lemmatized', count_vec=False)

for index, city in enumerate(dtm_tfidf.columns):
    generate_wordcloud(dtm_tfidf[city].sort_values(ascending=False), ci
```

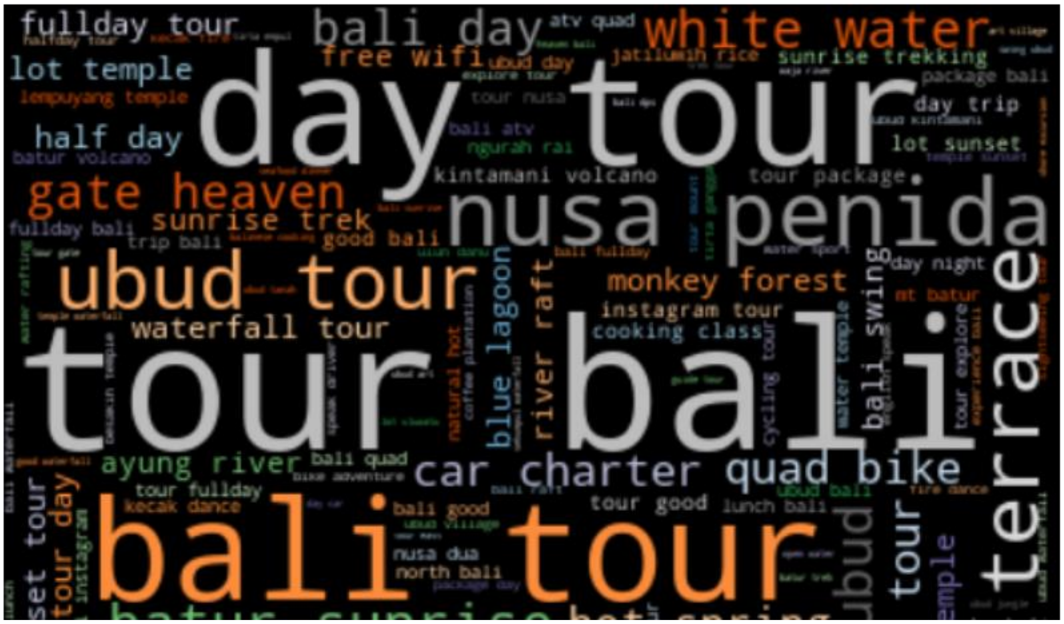
executed in 2.92s, finished 08:41:23 2021-01-26

In [66]:

```python
dtm_bigram = create_doc_term_matrix(df2, 'lemmatized',
                                    count_vec=True, ngram_range=(2,2))

for index, city in enumerate(dtm_bigram.columns):
    generate_wordcloud(dtm_bigram[city].sort_values(ascending=False), c
```

executed in 3.25s, finished 08:41:27 2021-01-26

Bali, Indonesia



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

2.3.1 Make Nicer Word Clouds

In [67]:

```python
def generate_better_wordcloud(data, title, mask=None):
    cloud = WordCloud(scale=3, max_words=150, colormap='tab20c', mask=m
                      background_color='white').generate_from_frequenci
    plt.figure(figsize=(10,8))
    plt.imshow(cloud, interpolation='bilinear')
    plt.axis('off')
    plt.title('\n'.join(wrap(title,60)), fontsize=13)
    plt.show()
```

executed in 2ms, finished 08:41:27 2021-01-26

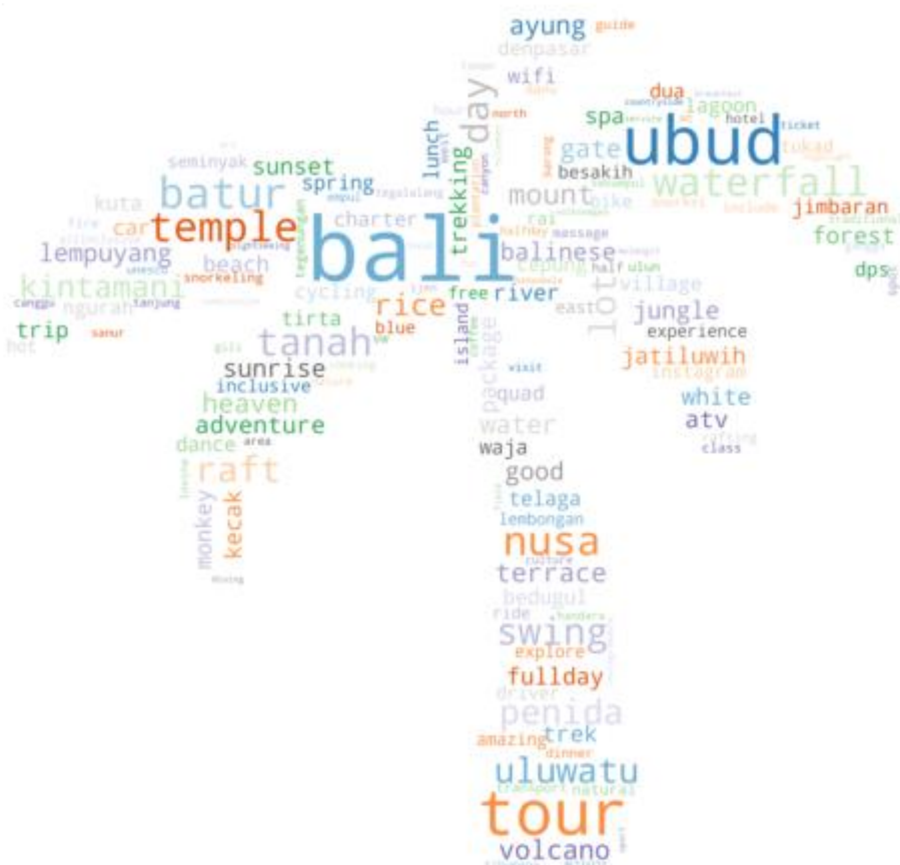
In [68]:

```python
mask = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron/capstc
```

executed in 16ms, finished 08:41:27 2021-01-26

In [69]:
```
generate_better_wordcloud(dtm_tfidf['Bali, Indonesia'].sort_values(asce
                          'Bali, Indonesia', mask=mask)
```
executed in 1.69s, finished 08:41:28 2021-01-26

Bali, Indonesia



In [70]:
```
mask_london = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron
```
executed in 7ms, finished 08:41:28 2021-01-26

In [28]:

```
generate_better_wordcloud(dtm_tfidf['London, United Kingdom'].sort_valu
                               'London, United Kingdom', mask=mask_london)
```

executed in 564ms, finished 08:41:29 2021-01-26

London, United Kingdom

In [29]:

```
mask_paris = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron/
generate_better_wordcloud(dtm_tfidf['Paris, France'].sort_values(ascend
                                'Paris, France', mask=mask_paris)
```

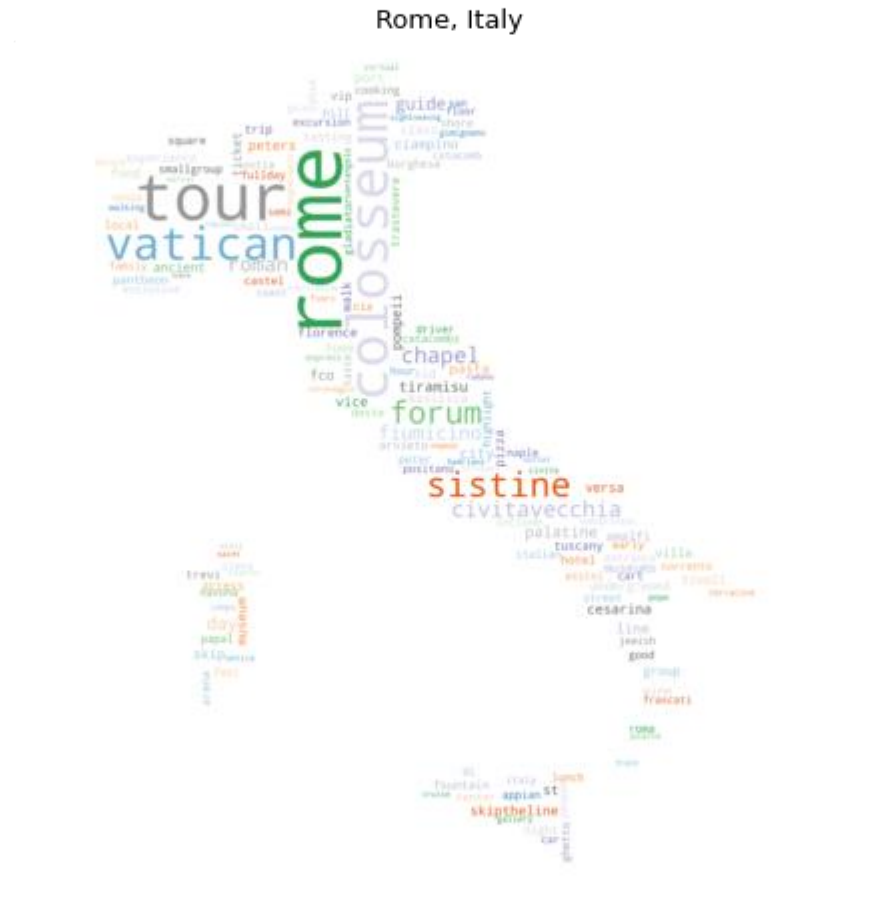
executed in 923ms, finished 08:41:30 2021-01-26

Paris, France

In [30]:

```
mask_rome = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron/c
generate_better_wordcloud(dtm_tfidf['Rome, Italy'].sort_values(ascendin
                          'Rome, Italy', mask=mask_rome)
```

executed in 5.99s, finished 08:41:36 2021-01-26



In [31]:

```
mask_sicily = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron
generate_better_wordcloud(dtm_tfidf['Sicily, Italy'].sort_values(ascend
                                  'Sicily, Italy', mask=mask_sicily)
```

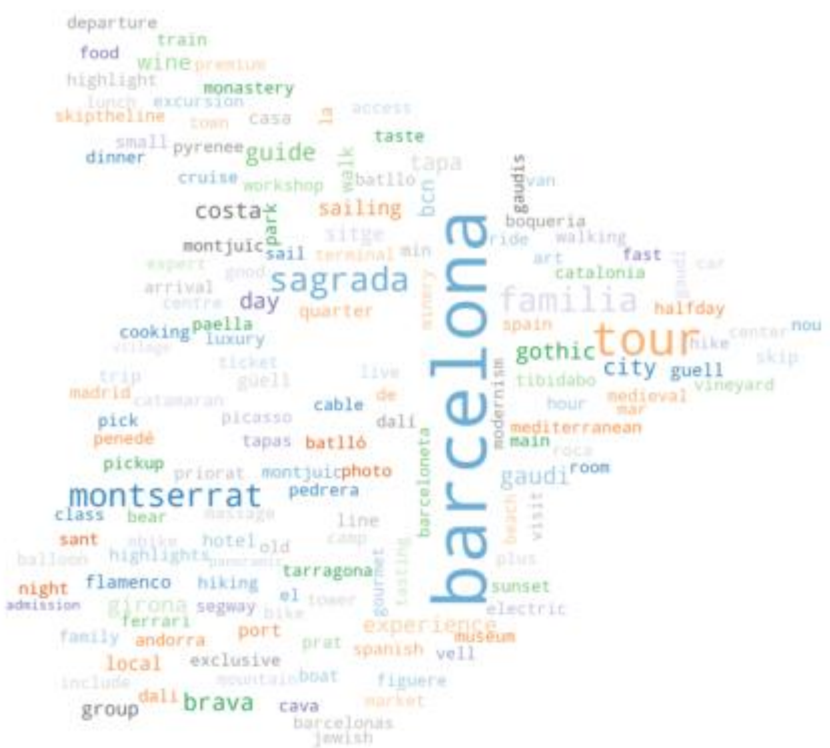
executed in 1.31s, finished 08:41:37 2021-01-26

Sicily, Italy

In [32]:
```
mask_barca = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron/
generate_better_wordcloud(dtm_tfidf['Barcelona, Spain'].sort_values(asc
                          'Barcelona, Spain', mask=mask_barca)
```

executed in 1.09s, finished 08:41:38 2021-01-26

Barcelona, Spain



In [33]:
```
mask_dubai = np.array(Image.open('/Users/tiaplagata/Documents/Flatiron/
generate_better_wordcloud(dtm_tfidf['Dubai, United Arab Emirates'].sort
                                 'Dubai, United Arab Emirates', mask=mask_duba
```
executed in 1.27s, finished 08:41:40 2021-01-26

In [34]:
```
generate_better_wordcloud(dtm_tfidf['Goa, India'].sort_values(ascending
                                   'Goa, India', mask=mask)
```

executed in 1.56s, finished 08:41:41 2021-01-26

Goa, India

In [35]:

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

executed in 984ms, finished 08:41:42 2021-01-26

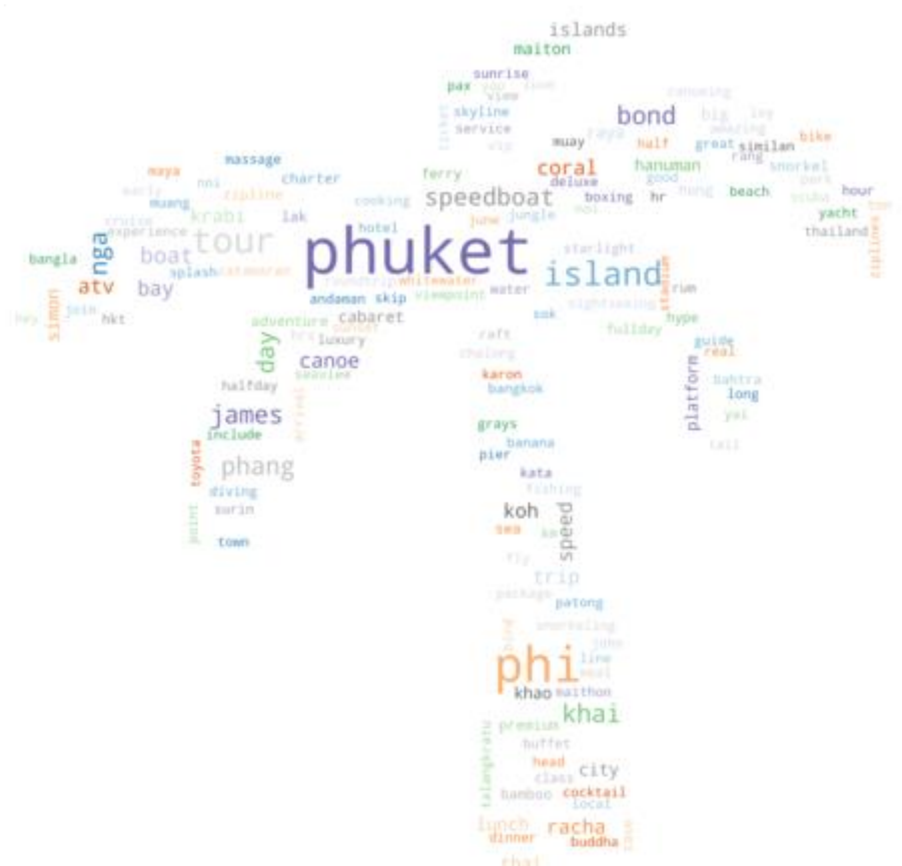
Istanbul, Turkey

In [36]:

```
generate_better_wordcloud(dtm_tfidf['Phuket, Thailand'].sort_values(asc
                                   'Phuket, Thailand', mask=mask)
```

executed in 1.50s, finished 08:41:44 2021-01-26

Phuket, Thailand



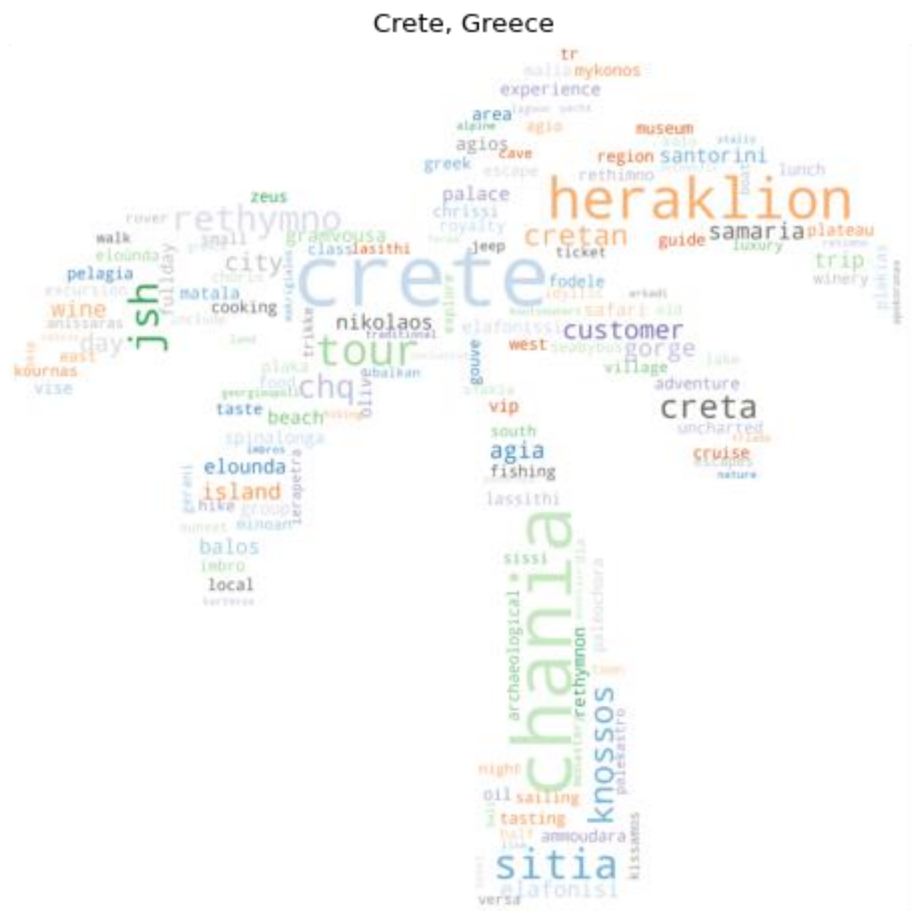
In [37]:

```
generate_better_wordcloud(dtm_tfidf['Majorca, Balearic Islands'].sort_v
                                   'Majorca, Balearic Islands', mask=mask)
```

executed in 1.51s, finished 08:41:45 2021-01-26

Majorca, Balearic Islands

In [38]:
```
generate_better_wordcloud(dtm_tfidf['Crete, Greece'].sort_values(ascenc
                                  'Crete, Greece', mask=mask)
```
executed in 1.62s, finished 08:41:47 2021-01-26



2.3.2 Most Frequent Words Visualizations

- Find the most frequent words per city and visualize them

In [39]:

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

executed in 16ms, finished 08:41:47 2021-01-26

Out[82]:

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...

In [40]:

```
bali_text = df_grouped.loc['Bali, Indonesia', 'lemmatized']
fd = FreqDist(word_tokenize(bali_text))
fd.most_common(20)
```

executed in 130ms, finished 08:41:47 2021-01-26

Out[83]:

```
[('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 [84]:

```
city_freqs = {}
for city in df_grouped.index:
    city_text = df_grouped.loc[city, 'lemmatized']
    fd = FreqDist(word_tokenize(city_text))
    city_freqs[city] = fd.most_common(20)
city_freqs_df = pd.DataFrame(city_freqs)
city_freqs_df.head()
```

executed in 708ms, finished 08:41:48 2021-01-26

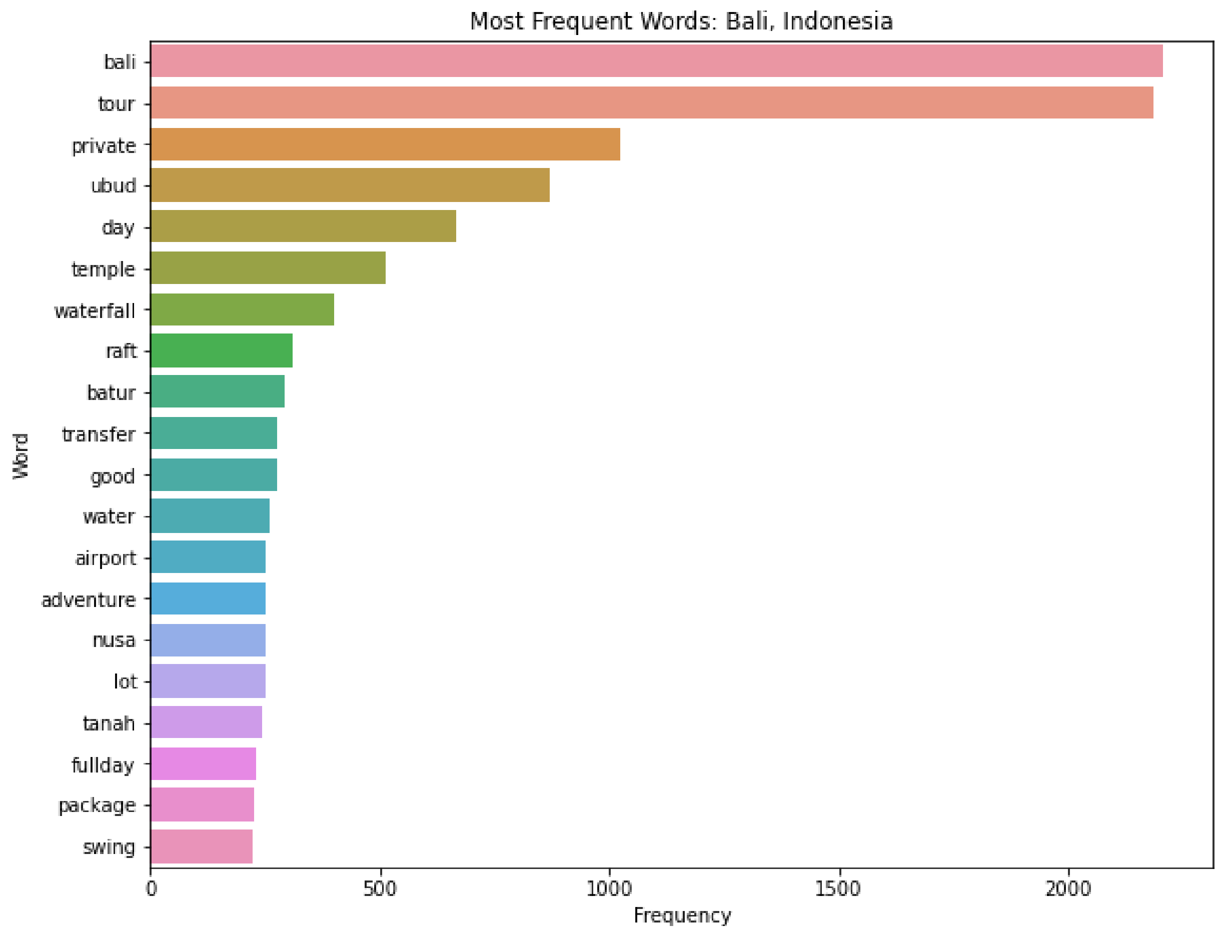
Out[84]:

	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 [41]:

```
# Make one graph of most frequent words
yaxis = [x[0] for x in city_freqs_df['Bali, Indonesia']]
xaxis = [x[1] for x in city_freqs_df['Bali, Indonesia']]

plt.figure(figsize=(10,8))
sns.barplot(xaxis, yaxis)
plt.title('Most Frequent Words: Bali, Indonesia')
plt.xlabel('Frequency')
plt.ylabel('Word')
plt.show()
```

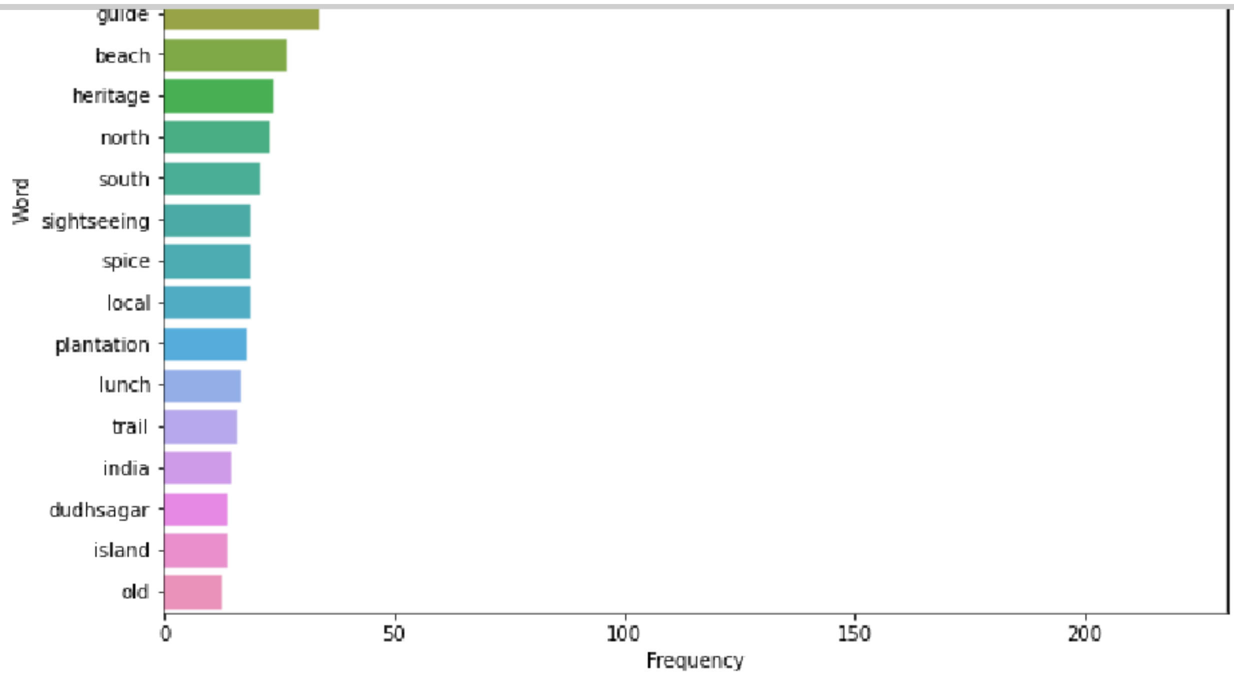
executed in 171ms, finished 08:42:03 2021-01-26



In [42]:

```python
# Make graphs for each city
for city in city_freqs_df.columns:
    yaxis = [x[0] for x in city_freqs_df[city]]
    xaxis = [x[1] for x in city_freqs_df[city]]

    plt.figure(figsize=(10,8))
    sns.barplot(xaxis, yaxis)
    plt.title(f'Most Frequent Words: {city}')
    plt.xlabel('Frequency')
    plt.ylabel('Word')
    plt.show()
```

executed in 1.82s, finished 08:42:15 2021-01-26



2.4 Modeling

2.4.1 Baseline Naive Bayes Model

In [88]:

```python
# Re-import the data to get a fresh start
data = pd.read_csv('/Users/tiaplagata/Documents/Flatiron/capstone-proje
data.head()
```

executed in 55ms, finished 08:42:41 2021-01-26

Out[88]:

	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 [89]:
```
# Perform train/test split before cleaning/preprocessing
X = data['Attraction']
y = data['City']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.2,
X_train.shape, X_test.shape
```
executed in 22ms, finished 08:42:42 2021-01-26

Out[89]:
```
((22703,), (5676,))
```

In [90]:
```
# Since this is a series, I will need to make it a DF for my preprocess
X_train
```
executed in 9ms, finished 08:42:43 2021-01-26

Out[90]:
```
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 [91]:
```
X_train_preprocessed = preprocess_df(pd.DataFrame(X_train, columns=['At
X_test_preprocessed = preprocess_df(pd.DataFrame(X_test, columns=['Attr
```
executed in 1m 46.2s, finished 08:44:30 2021-01-26

	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 ...	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 Su...	private bali half day car charter uluwatu su...
1313	Rooftop Pasta Making Class and Food Market Tou...	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...	change guard halfday private walk london tour

In [92]:
```
stopwords_list = stopwords.words('english')
stopwords_list += list(string.punctuation)
stopwords_list += ['airport', 'transfer', 'private']
```
executed in 3ms, finished 08:44:30 2021-01-26

In [93]:
```
# Vectorize the text data to be suitable for modeling
vectorizer = TfidfVectorizer(analyzer='word', stop_words=stopwords_list
X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['lemmatiz
X_test_tfidf = vectorizer.transform(X_test_preprocessed['lemmatized'])
```
executed in 181ms, finished 08:44:30 2021-01-26

In [94]:
```
def plot_conf_matrix(y_true, y_pred):

    """
    Plots a confusion matrix and displays classification report.
    """

    cm = confusion_matrix(y_true, y_pred, normalize='true')
    plt.figure(figsize=(15, 15))
    sns.heatmap(cm, annot=True, cmap='Blues', fmt='0.2g', annot_kws={"s
                xticklabels=nb.classes_, yticklabels=nb.classes_, squar
    plt.xlabel('Predictions')
    plt.ylabel('Actuals')
    plt.show()
```
executed in 3ms, finished 08:44:30 2021-01-26

In [95]:
```
def evaluate_model(model, X_train, X_test):
    y_preds_train = model.predict(X_train.todense())
    y_preds_test = model.predict(X_test.todense())

    print('Training Accuracy:', accuracy_score(y_train, y_preds_train))
    print('Testing Accuracy:', accuracy_score(y_test, y_preds_test))
    print('\n------------------------ \n')
    print('Training F1:', f1_score(y_train, y_preds_train, average='wei
    print('Testing F1:', f1_score(y_test, y_preds_test, average='weight
    print('\n------------------------ \n')
    print('Train Confusion Matrix\n')
    plot_conf_matrix(y_train, y_preds_train)
    print('Test Confusion Matrix\n')
    plot_conf_matrix(y_test, y_preds_test)
    print('\n-------------------------- \n')
    print(classification_report(y_test, y_preds_test))
```
executed in 3ms, finished 08:44:30 2021-01-26

In [96]:
```
nb = MultinomialNB()
nb.fit(X_train_tfidf.todense(), y_train)
```
executed in 1.28s, finished 08:44:31 2021-01-26

Out[96]:
```
MultinomialNB()
```

In [97]:
```
nb.classes_
```
executed in 5ms, finished 08:44:31 2021-01-26

Out[97]:
```
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 [98]:

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

executed in 2.28s, finished 08:44:33 2021-01-26

```
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.

2.4.2 Naive Bayes Iteration 2

- Using class weights to improve class imbalance.

In [99]:
```
# Compute class weights
class_weights = class_weight.compute_class_weight('balanced',
                                                  np.unique(y_train),
                                                  y_train)
weights_dict = dict(zip(np.unique(y_train), class_weights))
weights_dict
```
executed in 25ms, finished 08:44:33 2021-01-26

Out[99]:
```
{'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}
```

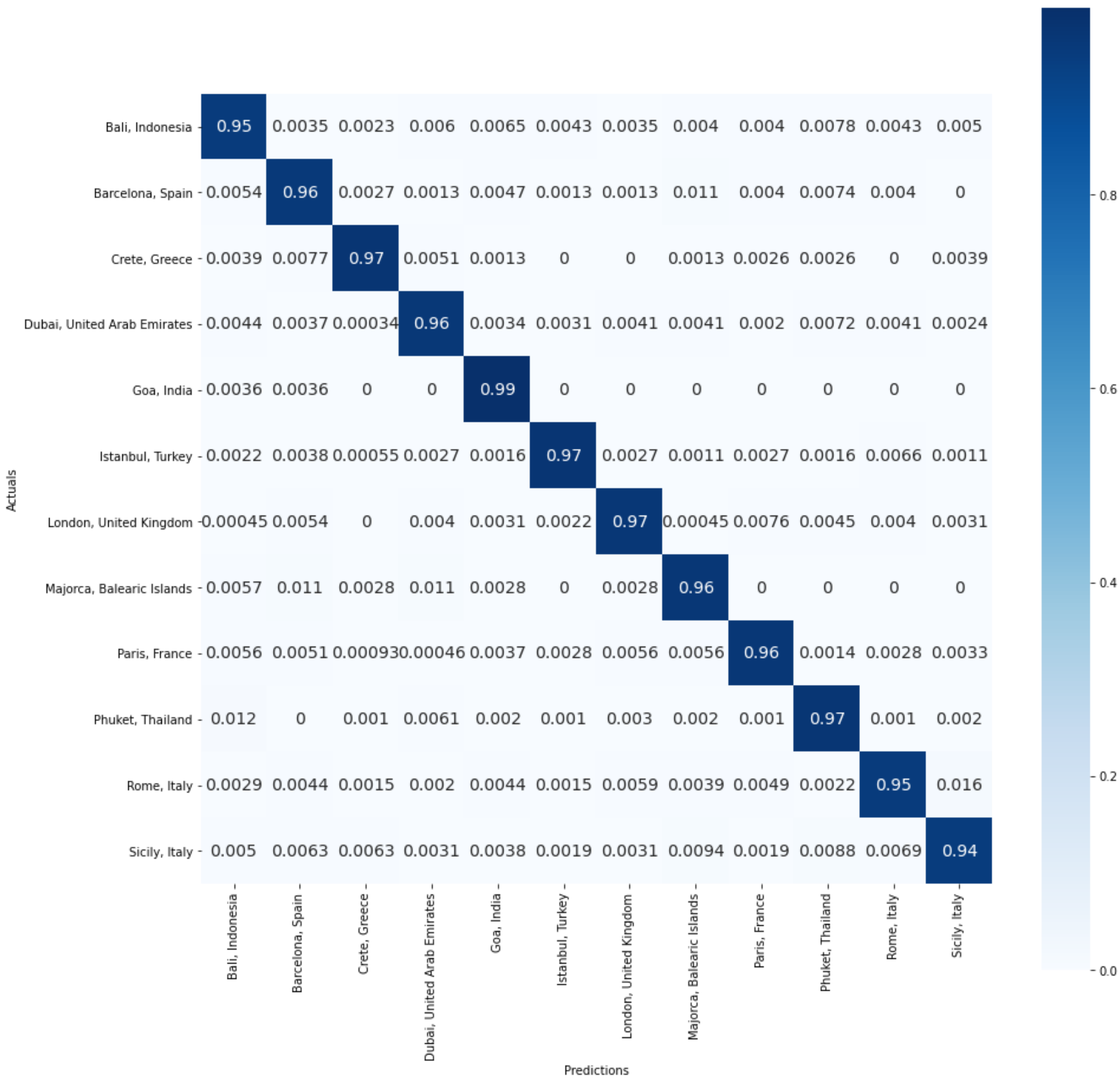
In [100]:
```
# Use class weights dictionary to calculate sample weight (needed for M
sample_weights = class_weight.compute_sample_weight(weights_dict, y_tra
```
executed in 13ms, finished 08:44:33 2021-01-26

In [101]:
```
nb = MultinomialNB()
nb.fit(X_train_tfidf.todense(),
       y_train,
       sample_weight=sample_weights)
```
executed in 772ms, finished 08:44:34 2021-01-26

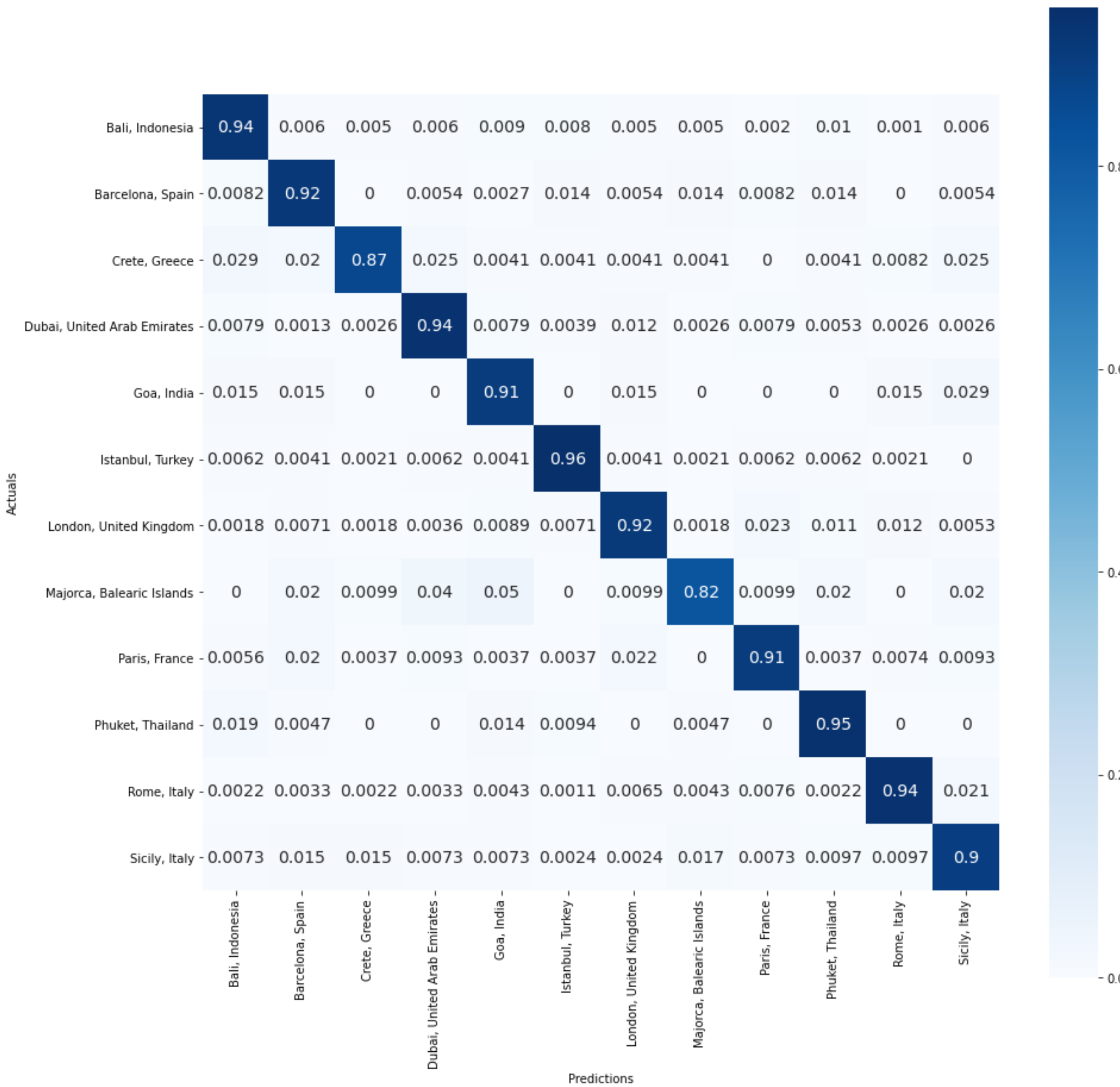
Out[101]:
```
MultinomialNB()
```

In [102]:
```
1 evaluate_model(nb, X_train_tfidf, X_test_tfidf)
```
executed in 2.34s, finished 08:44:37 2021-01-26

```
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.

2.4.3 Iteration 3: What happens if I take the city names out?

In [103]:

```
new_stopwords = stopwords_list + ['bali', 'barcelona', 'crete', 'dubai'
                                  'istanbul', 'london', 'majorca', 'phu
                                  'paris', 'rome', 'sicily', 'mallorca'
```

executed in 2ms, finished 08:44:37 2021-01-26

In [104]:

```
vectorizer = TfidfVectorizer(analyzer='word',
                             stop_words=new_stopwords,
                             decode_error='ignore')
X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['lemmatiz
X_test_tfidf = vectorizer.transform(X_test_preprocessed['lemmatized'])
```

executed in 280ms, finished 08:44:37 2021-01-26

In [105]:

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

executed in 786ms, finished 08:44:38 2021-01-26

Out[105]:

```
MultinomialNB()
```

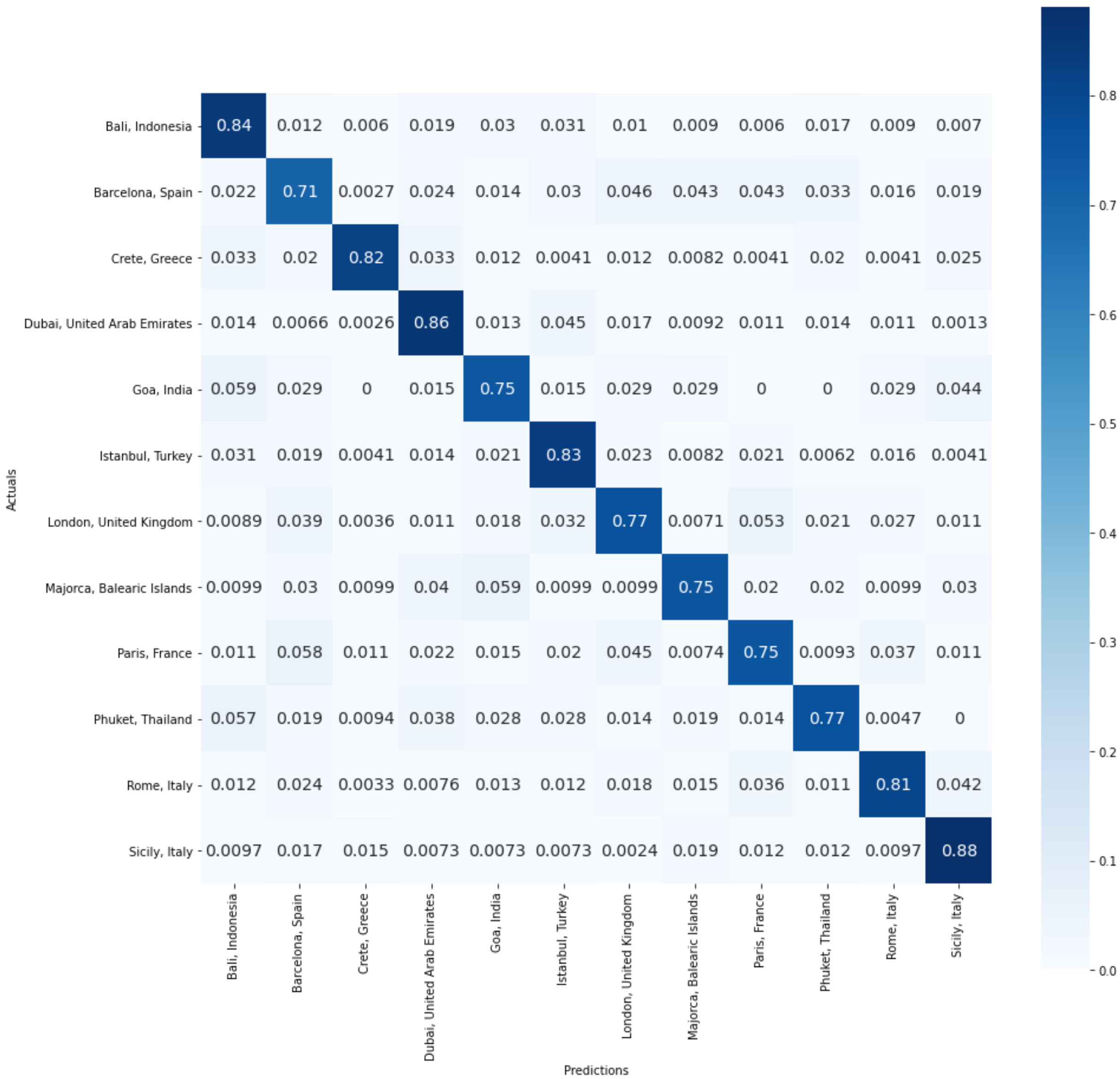
In [106]:

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

executed in 2.28s, finished 08:44:40 2021-01-26

```
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	5676
weighted avg	0.82	0.81	0.81	5676

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

In [107]:

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

executed in 5ms, finished 08:44:40 2021-01-26

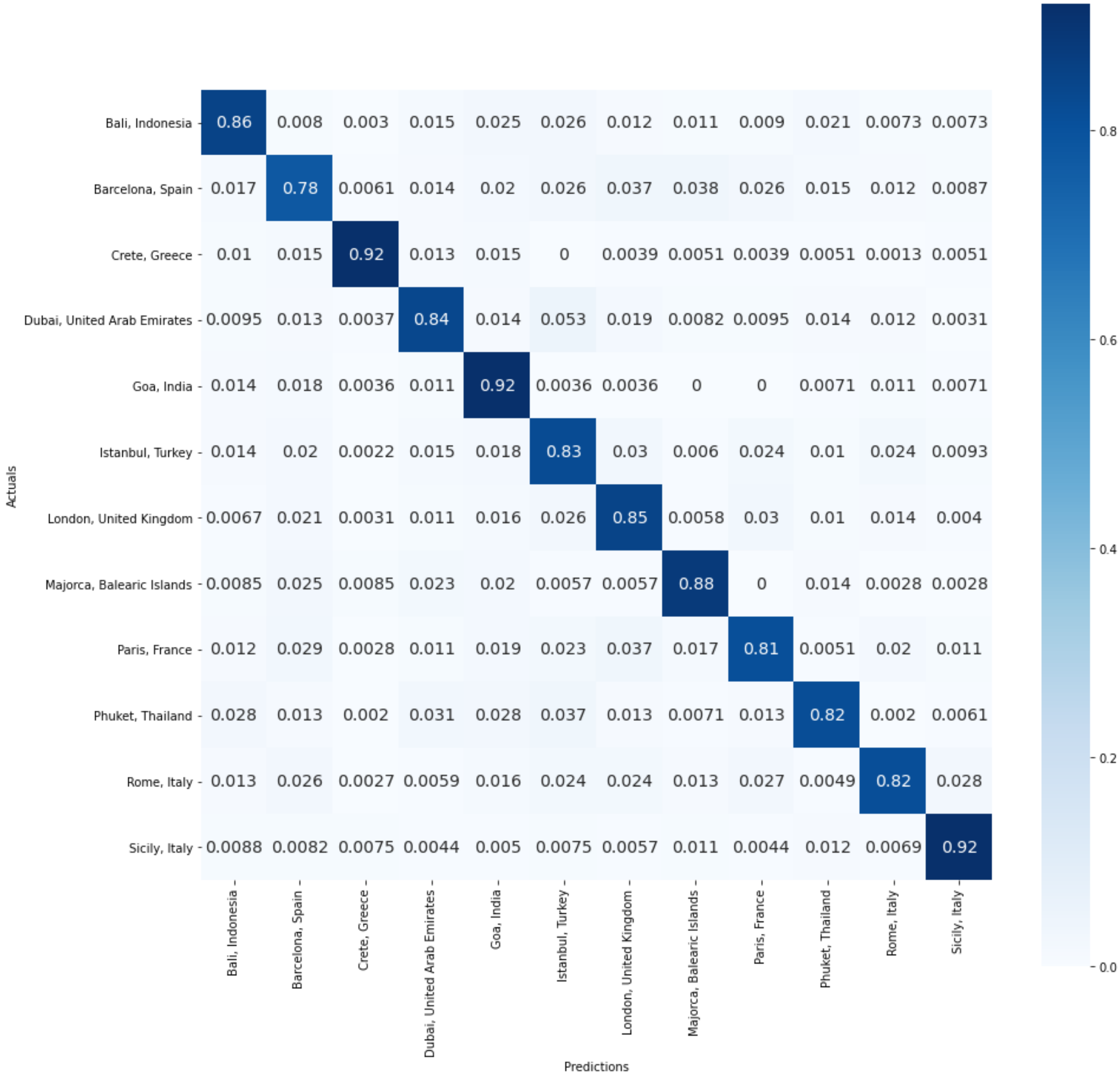
2.4.4 Iteration 4: Try using Count Vectorization

In [108]:

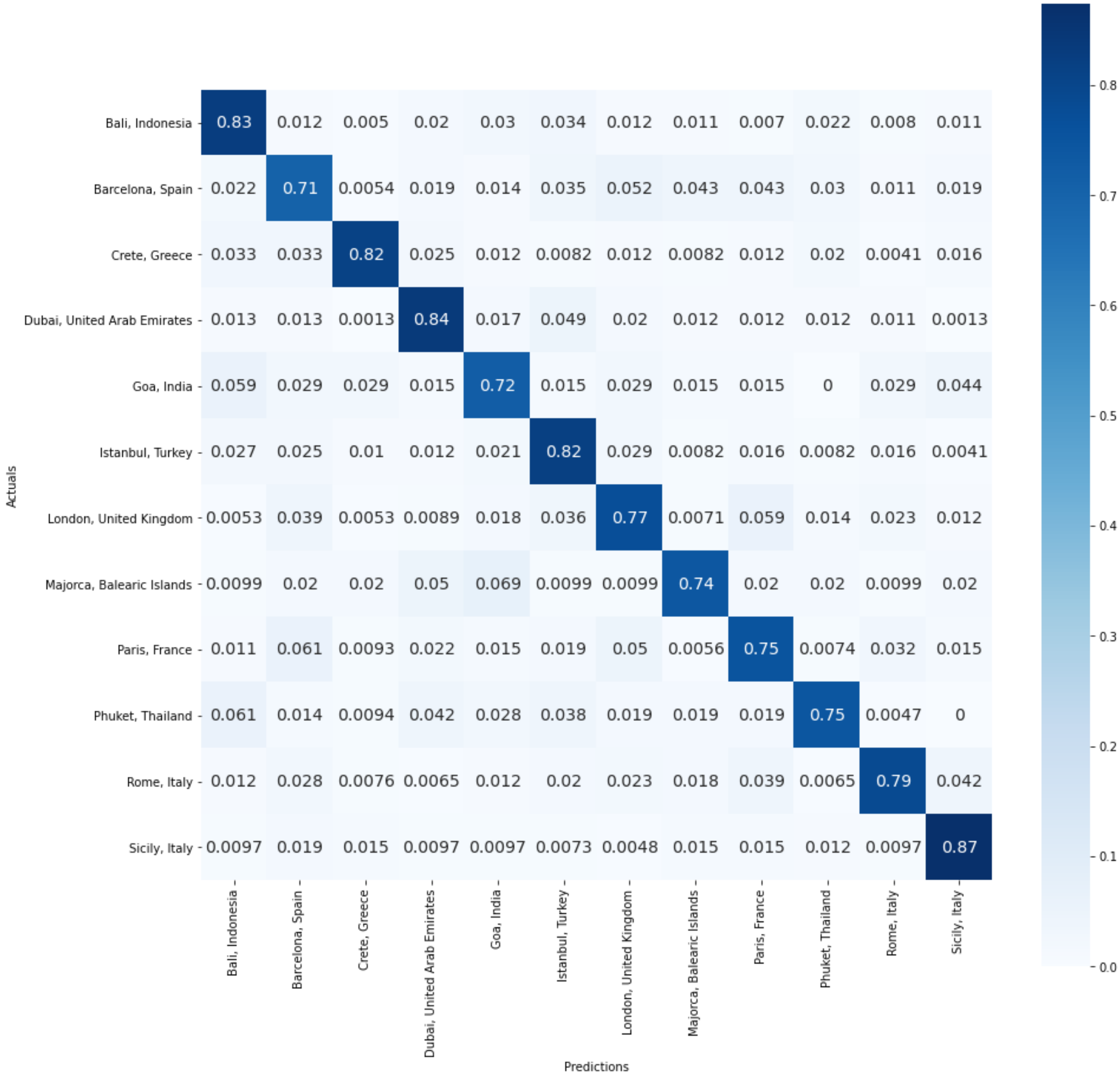
```
# Continuing each new iteration without city names
cv = CountVectorizer(analyzer='word',
                     stop_words=new_stopwords,
                     decode_error='ignore')
X_train_cv = cv.fit_transform(X_train_preprocessed['lemmatized'])
X_test_cv  =  cv.transform(X_test_preprocessed['lemmatized'])
nb_cv = MultinomialNB()
nb_cv.fit(X_train_cv.todense(),
          y_train,
          sample_weight=sample_weights)
evaluate_model(nb_cv, X_train_cv, X_test_cv)
```

executed in 5.00s, finished 08:44:45 2021-01-26

```
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.

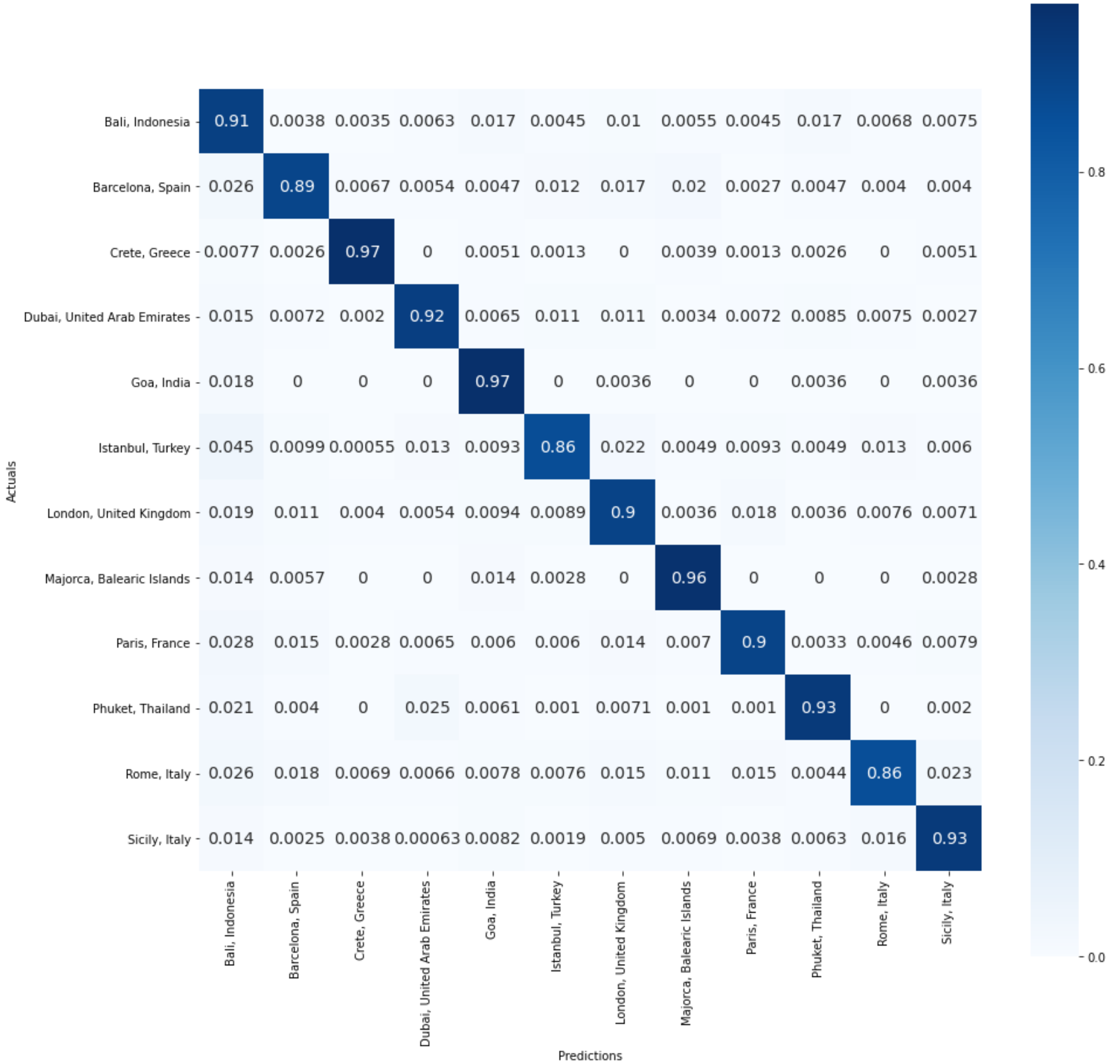
2.4.5 Iteration 5: Try using Bi-Grams

In [109]:

```
bigram = CountVectorizer(analyzer='word',
                         stop_words=new_stopwords,
                         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.todense(),
          y_train,
          sample_weight=sample_weights)
evaluate_model(nb_bg, X_train_bg, X_test_bg)
```

executed in 22.2s, finished 08:45:07 2021-01-26

```
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      0.69      5676
               weighted avg       0.75      0.73      0.74      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.

2.4.6 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 [110]:
```
vectorizer = TfidfVectorizer(analyzer='word',
                             stop_words=new_stopwords,
                             decode_error='ignore')
X_train_tfidf = vectorizer.fit_transform(X_train_preprocessed['lemmatiz
X_test_tfidf = vectorizer.transform(X_test_preprocessed['lemmatized'])
```
executed in 182ms, finished 08:45:07 2021-01-26

In [111]:
```
rf = RandomForestClassifier(class_weight=weights_dict)
rf.fit(X_train_tfidf.todense(), y_train)
```
executed in 2m 5s, finished 08:47:13 2021-01-26

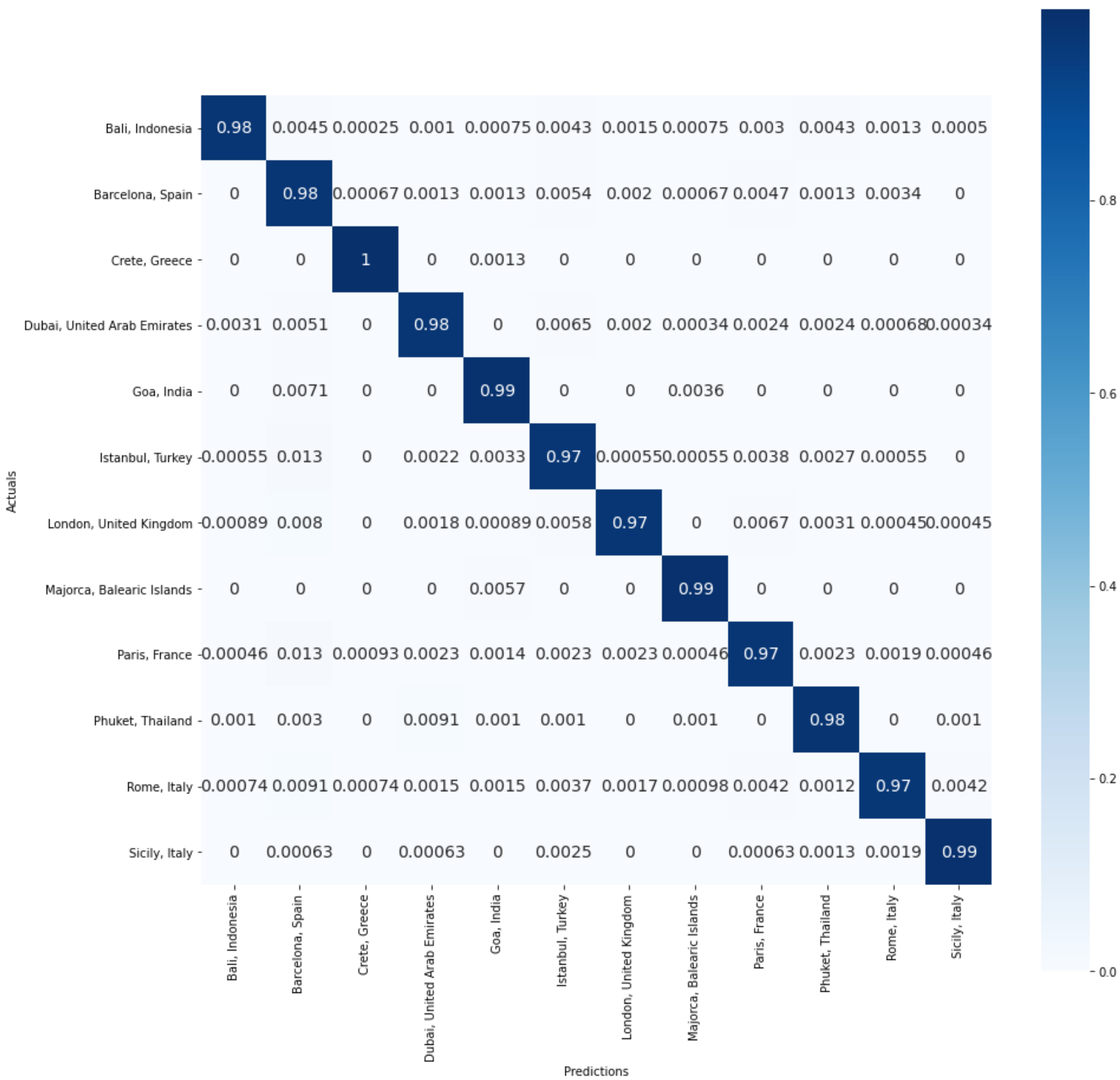
Out[111]:
```
RandomForestClassifier(class_weight={'Bali, Indonesia': 0.473097441026923
4,
                                     'Barcelona, Spain': 1.27230441605021
3,
                                     'Crete, Greece': 2.42864783910997,
                                     'Dubai, United Arab Emirates': 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 [112]:

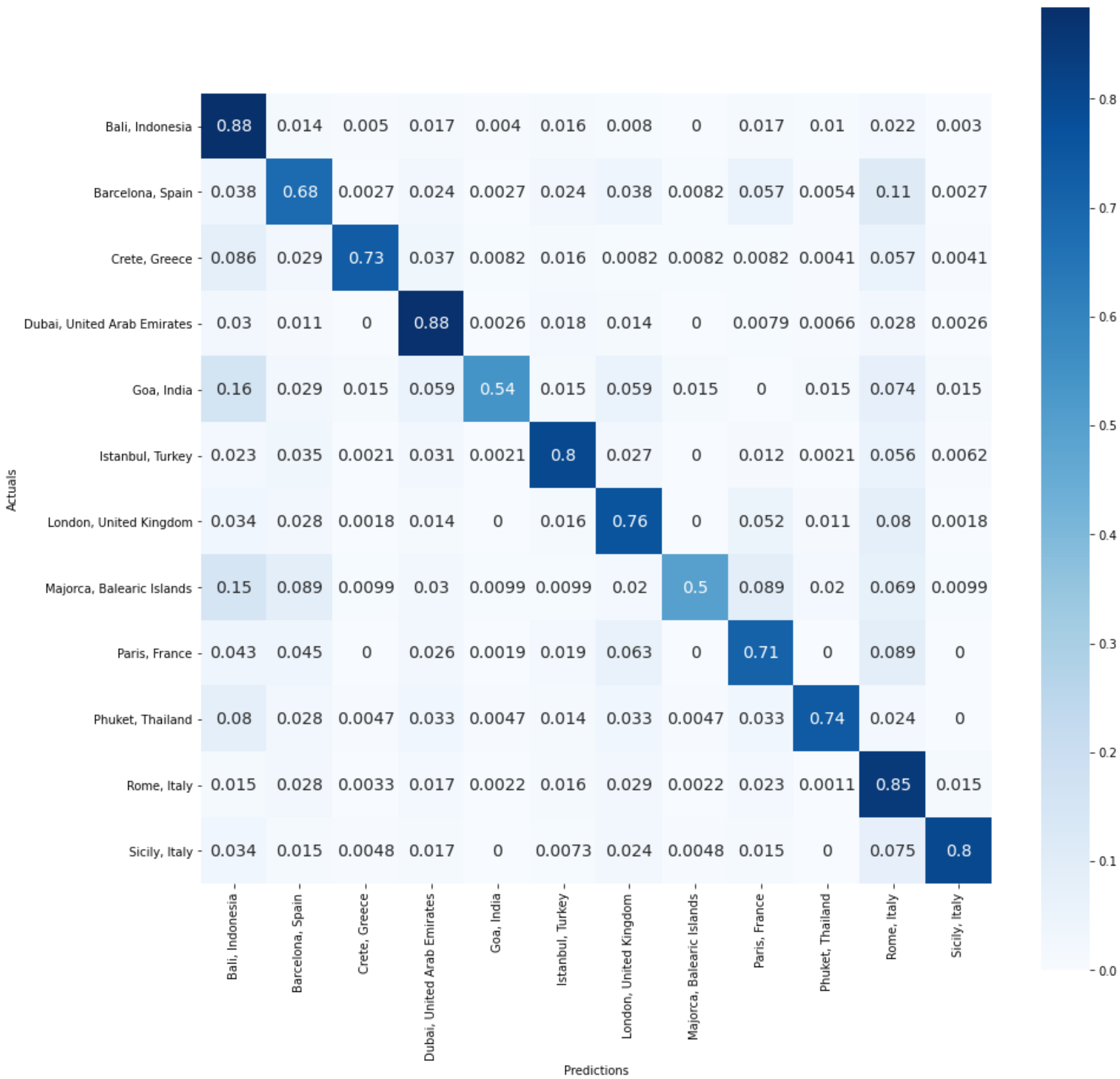
```
evaluate_model(rf, X_train_tfidf, X_test_tfidf)
```

executed in 5.69s, finished 08:47:18 2021-01-26

```
Training Accuracy: 0.9774479143725499
Testing Accuracy: 0.8007399577167019

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

Training F1: 0.9775901075153233
Testing F1: 0.7998604446251151

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

Train Confusion Matrix
```



Test Confusion Matrix



```
---------------
                              precision    recall  f1-score   support

            Bali, Indonesia       0.83      0.88      0.86      1001
           Barcelona, Spain       0.65      0.68      0.67       368
              Crete, Greece       0.92      0.73      0.82       244
Dubai, United Arab Emirates       0.86      0.88      0.87       760
                 Goa, India       0.71      0.54      0.62        68
           Istanbul, Turkey       0.82      0.80      0.81       485
     London, United Kingdom       0.76      0.76      0.76       563
  Majorca, Balearic Islands       0.82      0.50      0.62       101
              Paris, France       0.76      0.71      0.73       538
           Phuket, Thailand       0.84      0.74      0.79       212
                Rome, Italy       0.75      0.85      0.79       923
              Sicily, Italy       0.92      0.80      0.86       413

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

In [113]:

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

executed in 18ms, finished 08:47:18 2021-01-26

Out[113]:

```
aal             8.127660e-06
abandon         1.566779e-05
abant           6.703214e-06
abbate          4.990297e-05
abbey           6.088470e-04
abbeyprivate    1.116179e-06
abbeyst         1.747368e-06
aberfan         8.848928e-06
abian           1.849916e-07
aboard          6.325173e-05
abra            3.382000e-05
dtype: float64
```

In [114]:

```
top_20_feats = feat_imps.sort_values(ascending=False).head(20)
top_20_feats
```

executed in 4ms, finished 08:47:18 2021-01-26

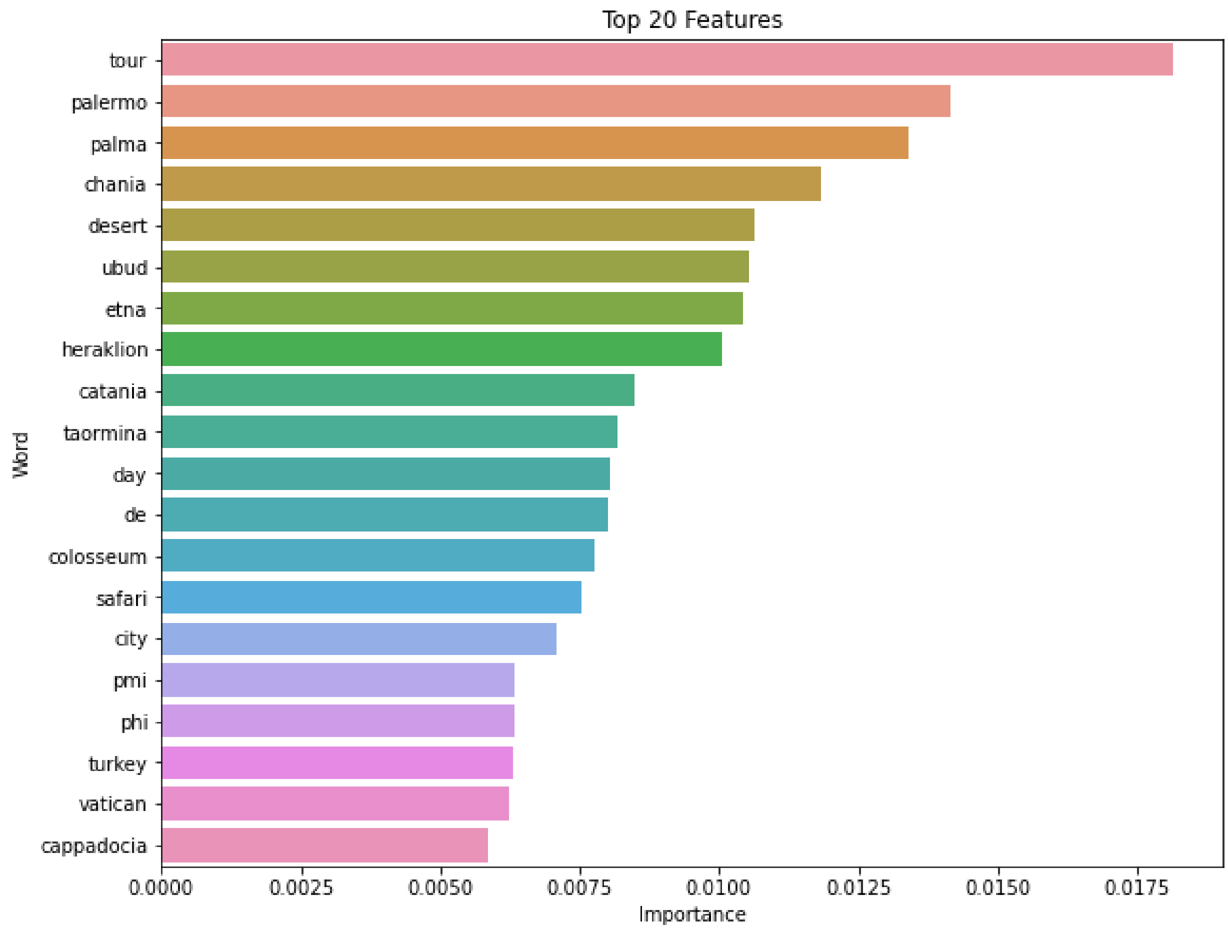
Out[114]:

```
tour          0.018132
palermo       0.014135
palma         0.013400
chania        0.011826
desert        0.010621
ubud          0.010546
etna          0.010433
heraklion     0.010064
catania       0.008469
taormina      0.008167
day           0.008053
de            0.008015
colosseum     0.007777
safari        0.007528
city          0.007086
pmi           0.006347
phi           0.006344
turkey        0.006303
vatican       0.006244
cappadocia    0.005848
dtype: float64
```

In [116]:

```
plt.figure(figsize=(10,8))
sns.barplot(x=top_20_feats, y=top_20_feats.index)
plt.title('Top 20 Features')
plt.ylabel('Word')
plt.xlabel('Importance')
plt.show()
```

executed in 172ms, finished 08:47:59 2021-01-26



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.

2.4.7 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 [117]:

```
X_train_cleaned = preprocess_df(pd.DataFrame(X_train, columns=['Attract
                                'cleaned', lemmatize=False)
X_test_cleaned = preprocess_df(pd.DataFrame(X_test, columns=['Attractic
                               'cleaned', lemmatize=False)
```

executed in 641ms, finished 08:48:09 2021-01-26

	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 cesarinas h...
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 M...	private tour istanbul sightseeing including mu...
1564	Catania Private Walking Tour	catania private walking tour

In [118]:

```
vectorizer = TfidfVectorizer(analyzer='word',
                             stop_words=new_stopwords,
                             decode_error='ignore')
X_train_tfidf = vectorizer.fit_transform(X_train_cleaned['cleaned'])
X_test_tfidf = vectorizer.transform(X_test_cleaned['cleaned'])
```

executed in 200ms, finished 08:48:09 2021-01-26

In [69]:
```
nb_cleaned = MultinomialNB()
nb_cleaned.fit(X_train_tfidf.todense(),
        y_train,
        sample_weight=sample_weights)
```
executed in 863ms, finished 08:48:11 2021-01-26

Out[119]:
```
MultinomialNB()
```

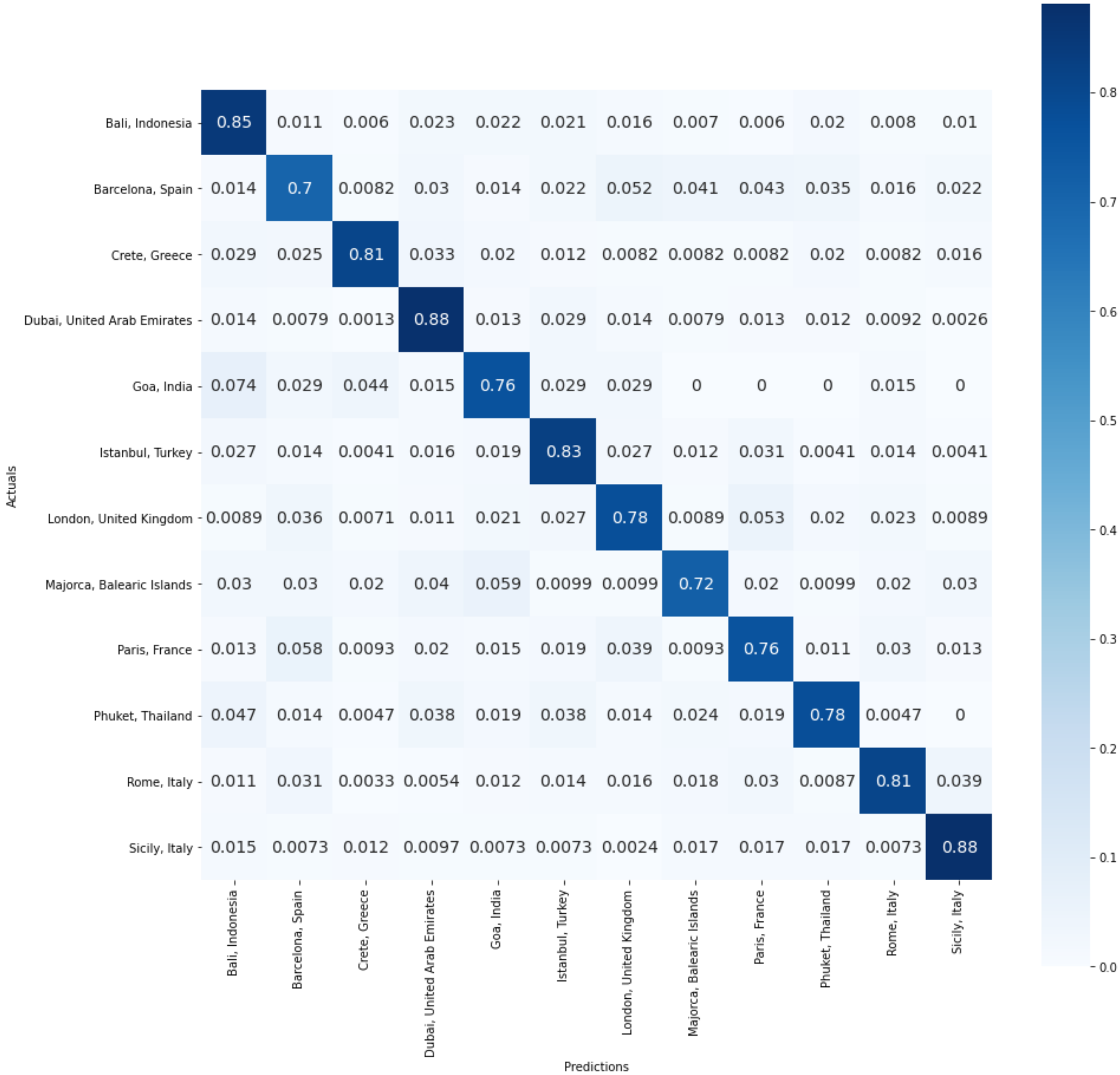
In [120]:

```
evaluate_model(nb_cleaned, X_train_tfidf, X_test_tfidf)
```

executed in 2.33s, finished 08:48:13 2021-01-26

```
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 [121]:
```
# Pickle the best Naive Bayes Model
with open('/Users/tiaplagata/Documents/Flatiron/capstone-project/non_le
          'wb') as f:
    pickle.dump(nb_cleaned, f, pickle.HIGHEST_PROTOCOL)
```
executed in 11ms, finished 08:48:15 2021-01-26

2.5 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 [122]:
```
def preprocess_text(text):
    """
    Input raw text.
    Return preprocessed text.
    """

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

    return [preprocessed]
```
executed in 8ms, finished 08:48:16 2021-01-26

In [123]:
```
raw_text = 'I want to go to the beach, go hiking and snorkeling'
preprocessed_text = preprocess_text(raw_text)
preprocessed_text
```
executed in 9ms, finished 08:48:17 2021-01-26

Out[123]:
```
['i want to go to the beach go hiking and snorkeling']
```

In [124]:
```
nb_cleaned.predict(vectorizer.transform(preprocessed_text))
```
executed in 14ms, finished 08:48:17 2021-01-26

Out[124]:
```
array(['Bali, Indonesia'], dtype='<U27')
```

In [125]:
```
preprocessed2 = preprocess_text('Go to historic museums')
print(preprocessed2)
nb_cleaned.predict(vectorizer.transform(preprocessed2))
```
executed in 13ms, finished 08:48:17 2021-01-26

```
['go to historic museums']
```

Out[125]:
```
array(['Rome, Italy'], dtype='<U27')
```

In [126]:
```
preprocessed3 = preprocess_text('Wine tastings, long walks and dinners'
print(preprocessed3)
nb_cleaned.predict(vectorizer.transform(preprocessed3))
```
executed in 14ms, finished 08:48:18 2021-01-26

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

Out[126]:
```
array(['Crete, Greece'], dtype='<U27')
```

In [127]:
```
preprocessed4 = preprocess_text('Do yoga on the beach')
print(preprocessed4)
nb_cleaned.predict(vectorizer.transform(preprocessed4))
```
executed in 12ms, finished 08:48:18 2021-01-26

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

Out[127]:
```
array(['Goa, India'], dtype='<U27')
```

In [128]:
```
preprocessed5 = preprocess_text('Sunset cruises on a yacht with wine')
print(preprocessed5)
nb_cleaned.predict(vectorizer.transform(preprocessed5))
```
executed in 15ms, finished 08:48:18 2021-01-26

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

Out[128]:
```
array(['Crete, Greece'], dtype='<U27')
```

2.5.1 Make this process into a pipeline

In [129]:

```
# Use OOP to get preprocessing steps into a pipeline
class PreprocessText(TransformerMixin):

    def__init__(self):
        self = self

    def fit(self, X, y=None, **fit_params):
        return self

    def transform(self, X, **transform_params):
        try:
            X = pd.DataFrame(X, columns=['Attraction'])
            X['cleaned'] = X['Attraction'].apply(lambda x: x.lower())
            X['cleaned'] = X['cleaned'].apply(lambda x: re.sub('[%s]' %
            X['cleaned'] = X['cleaned'].apply(lambda x: re.sub('\w*\d\w

            X = X['cleaned']
        except:
            pass
        return X

class DenseTransformer():

    def__init__(self):
        self = self

    def fit(self, X, y=None, **fit_params):
        return self

    def transform(self, X, y=None, **fit_params):
        return X.todense()
```

executed in 14ms, finished 08:48:19 2021-01-26

In [130]:

```
# Test preprocessing class
prep = PreprocessText()
prep.transform(X_train)
```

executed in 507ms, finished 08:48:21 2021-01-26

Out[130]:

```
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 [131]:

```
pipe = Pipeline(steps=[
                ('TextPreprocessor', PreprocessText()),
                ('TFIDFVectorizer', TfidfVectorizer(analyzer='word',
                                                    stop_words=new_stop
                                                    decode_error='ignor
                ('DenseTransformer', DenseTransformer()),
                ('NaiveBayes', MultinomialNB())])
```

executed in 9ms, finished 08:48:22 2021-01-26

In [132]:

```
set_config(display='diagram')
```

executed in 6ms, finished 08:48:23 2021-01-26

In [133]:

```
pipe.fit(X_train,
         y_train,
         **{'NaiveBayes__sample_weight': sample_weights})
```

executed in 1.54s, finished 08:48:25 2021-01-26

Out[133]:

Pipeline

PreprocessText

TfidfVectorizer

DenseTransformer

MultinomialNB

In [134]:

```
pipe.score(X_test, y_test)
```

executed in 360ms, finished 08:48:25 2021-01-26

Out[134]:

```
0.8146582100070472
```

In [135]:

```
pipe.predict(['I want to go snorkeling and tan on the beach'])
```

executed in 8ms, finished 08:48:25 2021-01-26

Out[135]:

```
array(['Bali, Indonesia'], dtype='<U27')
```

In [136]:

```
pipe.predict(['Go out for drinks'])
```

executed in 5ms, finished 08:48:25 2021-01-26

Out[136]:

```
array(['Barcelona, Spain'], dtype='<U27')
```

In [76]:

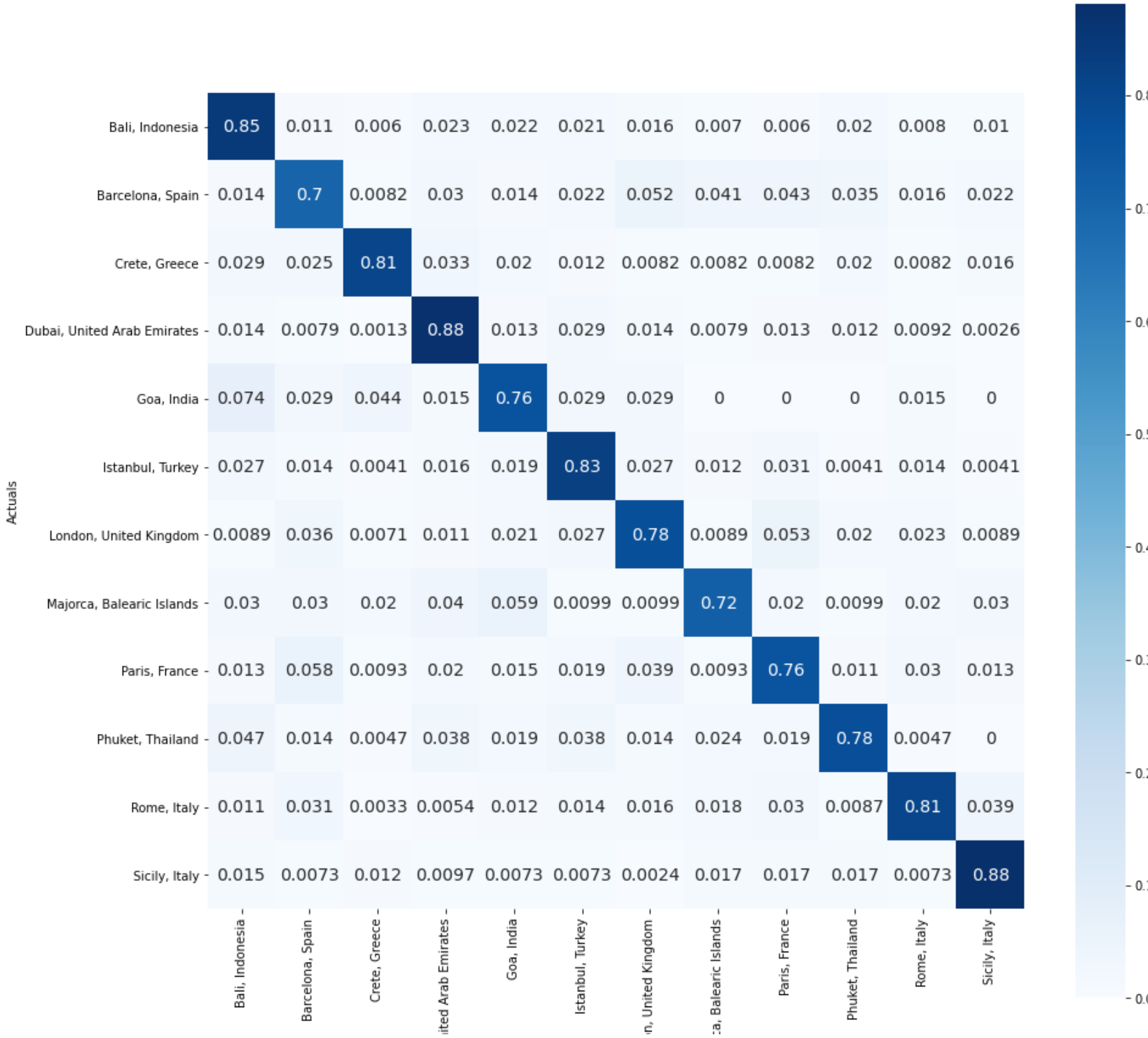
```
def evaluate_pipe(pipe, X_train, X_test):
    y_preds_train = pipe.predict(X_train)
    y_preds_test = pipe.predict(X_test)

    print('Training Accuracy:', accuracy_score(y_train, y_preds_train))
    print('Testing Accuracy:', accuracy_score(y_test, y_preds_test))
    print('\n------------------------\n')
    print('Training F1:', f1_score(y_train, y_preds_train, average='wei
    print('Testing F1:', f1_score(y_test, y_preds_test, average='weight
    print('\n------------------------\n')
    print('Train Confusion Matrix\n')
    plot_conf_matrix(y_train, y_preds_train)
    print('Test Confusion Matrix\n')
    plot_conf_matrix(y_test, y_preds_test)
    print('\n--------------------------\n')
    print(classification_report(y_test, y_preds_test))
```

executed in 8ms, finished 08:48:27 2021-01-26

In [138]:
```
evaluate_pipe(pipe, X_train, X_test)
```
executed in 3.15s, finished 08:48:32 2021-01-26

```
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
weighted avg	0.83	0.81	0.82	5676

In [139]:
```
# Pickle the Pipeline
with open('/Users/tiaplagata/Documents/Flatiron/capstone-project/final_
          'wb') as f:
    pickle.dump(pipe, f, pickle.HIGHEST_PROTOCOL)
```
executed in 13ms, finished 08:48:34 2021-01-26

In [140]:
```
# Load pipe from pickled file to test
with open('/Users/tiaplagata/Documents/Flatiron/capstone-project/final_
          'rb') as f:
    best_model_pipe = pickle.load(f)
best_model_pipe
```
executed in 32ms, finished 08:48:34 2021-01-26

Out[140]:
```
Pipeline
PreprocessText
TfidfVectorizer
DenseTransformer
MultinomialNB
```

In [141]:
```
best_model_pipe.predict(['I want to visit art galleries'])
```
executed in 13ms, finished 08:48:39 2021-01-26

Out[141]:
```
array(['Paris, France'], dtype='<U27')
```

2.5.2 Make Pipeline and Gridsearch for Random Forest

Now that I have a pipeline created, I will go back to iteration 6 with model tuning so see if I can get a better score with a less overfit model. I will use a gridsearch to tune the n_estimators and max_depth parameters, which commonly cause overfitting in Random Forest models.

In [170]:

```
rf_pipe = Pipeline(steps=[
                ('TextPreprocessor', PreprocessText()),
                ('TFIDFVectorizer', TfidfVectorizer(analyzer='word',
                                                    stop_words=new_stop
                                                    decode_error='ignor
                ('DenseTransformer', DenseTransformer()),
                ('RandomForest', RandomForestClassifier(class_weight=we
```

executed in 7ms, finished 09:45:50 2021-01-26

In [171]:

```
# Use a grid search to do some model tuning with the Random Forest (ite
param_grid = {'RandomForest__n_estimators': [100, 250, 500, 750],
              'RandomForest__max_depth': [5, 7, 9]}
```

executed in 7ms, finished 09:45:51 2021-01-26

In [172]:

```
rf_gridsearch = GridSearchCV(rf_pipe, param_grid=param_grid,
                             verbose=1, scoring='accuracy')
rf_gridsearch.fit(X_train, y_train)
```

executed in 26m 58s, finished 10:12:52 2021-01-26

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent w
orkers.
[Parallel(n_jobs=1)]: Done  60 out of  60 | elapsed: 26.1min finished
```

Out[172]:

```
GridSearchCV
PreprocessText
TfidfVectorizer
DenseTransformer
RandomForestClassifier
```

In [173]:

```
# See the params from the best model
rf_gridsearch.best_params_
```

executed in 3ms, finished 10:12:52 2021-01-26

Out[173]:

```
{'RandomForest__max_depth': 9, 'RandomForest__n_estimators': 500}
```

In [174]:

```
# Save best estimator in a variable and get accuracy score
best_rf = rf_gridsearch.best_estimator_
y_test_preds = best_rf.predict(X_test)
accuracy_score(y_test, y_test_preds)
```

executed in 653ms, finished 10:12:53 2021-01-26

Out[174]:

```
0.6464059196617337
```

In [175]:
```python
# Compare against training accuracy
y_train_preds = best_rf.predict(X_train)
accuracy_score(y_train, y_train_preds)
```
executed in 3.23s, finished 10:12:56 2021-01-26

Out[175]:
```
0.6529533541822666
```

Even with the model tuning, this model still is not performing as well as iteration 3. Although the model tuning helped to prevent overfitting, its accuracy score is not as good. Therefore, I will still conclude that the Naive Bayes iteration 3 without lemmatization is the best model.

2.5.3 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 [122]:
```python
probas = best_model_pipe.predict_proba(['I want to visit art galleries'
probas
```
executed in 5ms, finished 17:33:23 2021-01-21

Out[122]:
```
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 [123]:
```python
classes = best_model_pipe.classes_
classes
```
executed in 3ms, finished 17:33:23 2021-01-21

Out[123]:
```
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 [124]:
```python
# First Prediction
classes[probas.argmax()]
```
executed in 4ms, finished 17:33:23 2021-01-21

Out[124]:
```
'Paris, France'
```

In [125]:
```python
# Second Prediction
classes[np.argsort(probas)[:, 10]][0]
```
executed in 4ms, finished 17:33:23 2021-01-21

Out[125]:
```
'London, United Kingdom'
```

2.6 Conclusion

My final model is a Multinomial Naive Bayes classifier, which can predict a destination with 81% accuracy and an 82% F1 score (iteration 3 without lemmatization in this notebook). The text data put into this model is not lemmatized, but is lowercased with stopwords removed and city names removed.

2.6.1 Model Fit & Score

I used accuracy and F1 score to score this model. Since there are 12 classes, I want to model to be accurate, however, F1 score is also important to consider since there is some class imbalance in the dataset and to account for the model's false positives and false negatives.

The final model had the following training and testing accuracy and F1 scores:

- Testing Accuracy Score 0.81 | F1 Score 0.82
- Training Accuracy Score 0.86 | F1 Score 0.86

Looking at the above scores for both accuracy and F1, we can conclude that the model is a tiny bit overfit, but overall very accurate, especially considering that there are 12 classes.

I was surprised that the final/best-performing model was iteration 3 without lemmatization because I thought that lemmatizing the text would help the model's score.

2.6.2 Business Recommendations

- Integrate the Destination Dictionary technology into pages where Top Destination lists are published to drive engagement with future travelers and drive traffic to affiliate links

- Use the Destination Dictionary technology paired with a chatbot on travel websites to act as a virtual travel agent

- Offer paid sponsorship of the 'default' city-- ex. Tourism Board of Bali can pay be the first recommended city when you open the page

2.6.3 Next Steps -- Dash App

Everything works and is ready for the next step. This model will be put into The Destination Dictionary Dash app for its final use-case: predicting where people should travel based on the activities that they want to do on vacation!

You can see the GitHub Repo for the Dash App [here (https://github.com/tiaplagata/dash-travel-app)](https://github.com/tiaplagata/dash-travel-app)