ABP

January 11, 2022

En este notebook vamos a implementar una aplicación web donde el usuario pueda buscar las mejores ciudades para viajar en base una descripción y unas fotografía

1 Primeros pasos y carga de datos

Importamos PANDAS y cargamos nuestro propio dataset

[1]:
```
import pandas as pd
dataset = pd.read_csv("Datasets2.csv")
```

Vamos a renombrar los nombres de las columnas y a previsualizar como es nuestro DataFrame

[2]:
```
#Renombramos los nombres de las columnas y visualizamos nuestro dataset
dataset = dataset.rename(columns={'Unnamed: 0': 'City', 'Unnamed: 1' :␣
 ↪'Description', 'Unnamed: 2' : 'Country', 'Unnamed: 3' : 'Continent',␣
 ↪'Unnamed: 4' : 'LAT', 'Unnamed: 5' : 'LON'})
#Pasamos las columnas LAT y LON a números buenos en python (cambiamos las comas␣
 ↪',' por puntos '.')
for row in dataset.itertuples():
  #Sabiendo que cada row es una tupla y que el campo 5 y 6 es la LAT y la LON␣
 ↪respectivamente...
  lat = str(row[5])
  lon = str(row[6])
  lat = lat.replace(',','.')
  lon = lon.replace(',','.')

  dataset["LAT"].replace({row[5] : lat}, inplace=True)
  dataset["LON"].replace({row[6] : lon}, inplace=True)

dataset = dataset.drop([0])

dataset
```

[2]:

	City	...	LON
1	Paris	...	2.3488000
2	Berlin	...	13.4105300
3	Madrid	...	-3.7025600
4	Istanbul	...	28.9496600

```
5     Moscow  ...  37.6155600
..       ...  ...         ...
59      Giza  ...     31.2118
60    Ankara  ...     32.8500
61   Houston  ...    -95.3889
62   Atlanta  ...    -84.4224
63   Toronto  ...    -79.3733

[63 rows x 6 columns]
```

[3]:
```
listaPaises = list()

listaPaises = list(dataset["Country"])


#Quitamos los valores duplicados
result = []
for item in listaPaises:
    if item not in result:
        result.append(item)
result
```

[3]:
```
['France',
 'Germany',
 'Spain',
 'Turkey',
 'Russia',
 'Italy',
 'Brazil',
 'Mexico',
 'EEUU',
 'China',
 'Japan',
 'UK',
 'Czech Republic',
 'Belgium',
 'Romania',
 'Austria',
 'Greece',
 'Norway',
 'Egypt',
 'Netherlands',
 'Colombia',
 'Thailand',
 'Peru',
 'Argentina',
 'Angola']
```

Como podemos ver nuestro DataFrame tiene 4 parámetros:

1. ***City*** Nombre de la ciudad
2. ***Description*** Breve descripción de la ciudad mencionado sus características principales
3. ***Country*** Pais al que pertenece la ciudad
4. ***Continent*** Continente
5. ***LAT*** Latitud
6. ***LON*** Longitud

2 Procesado del texto

Vamos a realizar un tratado del texto de la descripción de cada ciudad, eliminando articulos, palabras irrelevantes y solo quedándonos con las raices de las palabras importantes.

Añadiremos una nueva columna a nuestro DataFrame llamada *processed_text_description* para ver el resultado de este procesamiento del campo descripción

[4]:
```
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

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

ps = PorterStemmer()

preprocessedText = []

for row in dataset.itertuples():


    text = word_tokenize(row[2]) ## indice de la columna que contiene el texto
    ## Remove stop words
    stops = set(stopwords.words("english"))
    text = [ps.stem(w) for w in text if not w in stops and w.isalnum()]
    text = " ".join(text)

    preprocessedText.append(text)

preprocessedData = dataset
preprocessedData['processed_text_description'] = preprocessedText

preprocessedData
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

[4]:
```
         City  ...                          processed_text_description
1       Paris  ...  pari capit popul citi franc estim popul resid ...
2      Berlin  ...  berlin world citi cultur polit media scienc It...
3      Madrid  ...  the madrid urban agglomer gdp european union i...
4    Istanbul  ...  istanbul largest citi turkey countri econom cu...
5      Moscow  ...  As northernmost coldest megac world histori da...
..        ...  ...                                                ...
59       Giza  ...  Is citi egypt cairo citi africa kinshasa lago ...
60     Ankara  ...  Is capit turkey locat central part anatolia ci...
61    Houston  ...  Is popul citi texa popul citi unit state popul...
62    Atlanta  ...  It serv cultur econom center atlanta metropoli...
63    Toronto  ...  Is capit citi canadian provinc ontario with re...

[63 rows x 7 columns]
```

Vemos como ahora nuestro dataset tiene una columna que es el texto procesado de la descripción. Se han eliminado las stopwords, se seleccionan solo las palabras relevantes y se saca su raiz.

3 Representación del texto como un vector: Bag of Words

Necesitamos representar el texto en un formato válido y la forma más utilziada es con un bag-of-words. En esta representación cada documento se transforma en un vector de tamaño igual al número de palabras representativas (términos) que lo constituyen. Cada elemento del vector indica el número de repeticiones de un término en el documento.

3.1 *##TF-IDF*

term Frecuency - inverse document frecuency. Es una forma de medir estadísticamente la importancia que tiene ciera palabra en un texto. Tiene en cuenta el número de veces que un término aparece en un único documento y el núemor de veces que aparece en todos los documentos de la colección

Podríamos resumir que el TF-IDF es una medida numérica que expresa cuán relevante es una palabra para un documento en una colección.

[5]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

bagOfWordsModel = TfidfVectorizer()
bagOfWordsModel.fit(preprocessedData['processed_text_description'])
textsBoW= bagOfWordsModel.
 ↪transform(preprocessedData['processed_text_description'])
print("Finished")
```

```
Finished
```

[6]:
```
print(type(textsBoW))
textsBoW.shape
```

```
<class 'scipy.sparse.csr.csr_matrix'>
```

[6]:
```
(63, 2058)
```

Vemos como nos dice que tenemos 33 filas (las 33 ciudades que hay en nuestro DataFrame) y 1262 columnas (701 palabras diferentes en todas las descripciones de las ciudades)

Vamos a mostrar está matriz (omitiendo los valores a 0).

Podemos ver en la siguiente representación:

- Número de la ciudad
- Número de la palabra
- Valor TF-IDF

(nžciudad, nžpalabra) valor TF-IDF

[7]:
```
#print(textsBoW)
```

Podemos ver todas las diferentes palabras que tenemos en nuestra *BagOfWords*

[8]:
```
bagOfWordsModel.get_feature_names_out()
```

[8]:
```
array(['10', '100', '1032', ..., 'zoo', 'ángel', 'dori'], dtype=object)
```

4 Cálculo de distancias entre vectores de frecuencias

El objetivo ahora es crear una matriz NxN (Número de ciudades) en donde el valor de la posición *matriz[i,j]* indique la distancia que existe entre la ciudad *i* y la ciudad *j*. Gracias a que ahora los textos están representados mediante vectores de frecuencias (*textsBoW*), se puede emplear para ello medidas de distancias standard entre vectores. En nuestro caso, vamos a emplear la distancia *coseno*

[9]:
```
from sklearn.metrics import pairwise_distances

distance_matrix= pairwise_distances(textsBoW,textsBoW ,metric='cosine')
```

[10]:
```
print(distance_matrix.shape)
print(type(distance_matrix))
pd.DataFrame(distance_matrix)
```

```
(63, 63)
<class 'numpy.ndarray'>
```

[10]:

	0	1	2	...	60	61	62
0	0.000000	0.974079	0.930827	...	0.921115	0.961461	0.915960
1	0.974079	0.000000	0.911556	...	0.952799	0.961741	0.943994
2	0.930827	0.911556	0.000000	...	0.951956	0.915403	0.917284
3	0.921908	0.961887	0.922449	...	0.942381	0.897336	0.891292
4	0.891645	0.943571	0.932833	...	0.948079	0.955208	0.942102
..	...	...	...	...	...	...	...

```
58  0.964453  0.977125  0.964068  ...  0.973141  0.976243  0.959212
59  0.955816  0.992906  0.977443  ...  0.947365  0.924269  0.944634
60  0.921115  0.952799  0.951956  ...  0.000000  0.924276  0.885217
61  0.961461  0.961741  0.915403  ...  0.924276  0.000000  0.928837
62  0.915960  0.943994  0.917284  ...  0.885217  0.928837  0.000000

[63 rows x 63 columns]
```

Vamos a crear un gráfico para ver está matriz un poco mejor y ver de un vistazo las cosas importantes

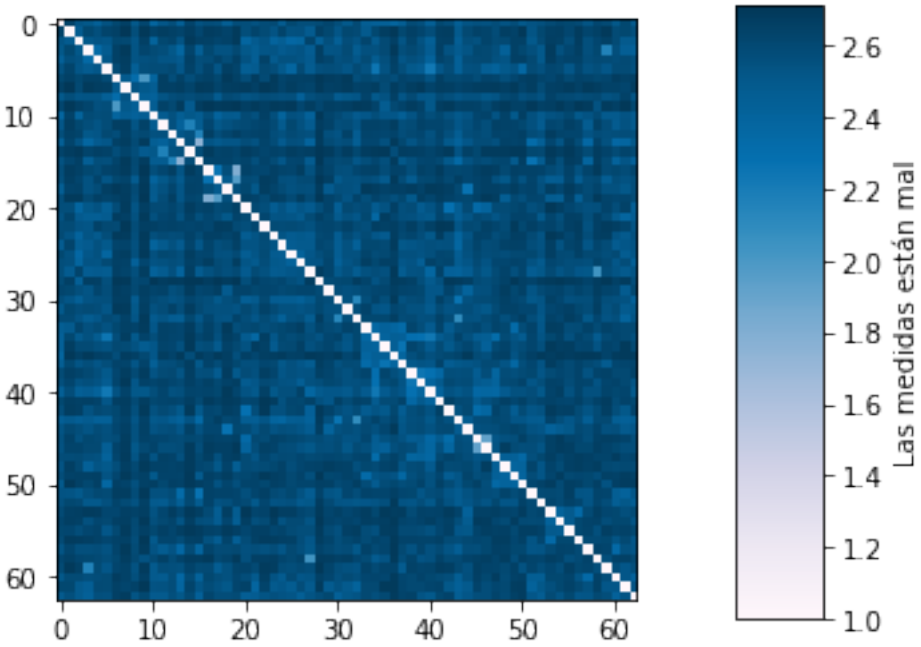
1. Cuanto más parecidas sean las ciudades, más claro el color
2. Cuanto más diferentes sean las ciudades, más oscuro el color

Con lo cual, vemos como la diagonal de la matriz es blanca, puesto que se compara la ciudad consigo misma.

[11]:
```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib import colors

XB = np.linspace(0,1,20)
YB = np.linspace(0,1,20)
X,Y = np.meshgrid(XB,YB)
Z = np.exp(distance_matrix)
im = plt.imshow(Z,interpolation='none',cmap='PuBu')

cax = plt.axes([0.85, 0.1, 0.075, 0.8])
plt.colorbar(im, cax=cax,label="Las medidas están mal")
plt.show()
```



5 Búsqueda de otras ciudades similares en base a la descripción

---

Supongamos que el usuario ha seleccionado la ciudad ***Hong Kong*** y vamos a buscar las ciudades más parecidas a esta.

[12]:
```
searchCity = "Hong Kong" #Ciudad base para las recomendaciones
indexOfCity = preprocessedData[preprocessedData['City']==searchCity].index.
 ↪values[0]
indexOfCity
```

[12]:
```
17
```

[13]:
```
#hay que restar uno al índice porque nuestro dataset no empieza en 0 si no en 1
ciudad = preprocessedData.iloc[indexOfCity-1]
ciudad
```

[13]:
```
City                                                          Hong Kong
Description                 Hong Kong , officially the Hong Kong Special A...
Country                                                           China
Continent                                                            AS
LAT                                                          22.3964280
LON                                                         114.1094970
processed_text_description    hong kong offici hong kong special administr r...
Name: 17, dtype: object
```

Calculamos las distancias comparando esa ciudad con las demas * 0ciudad muy similar * 1ciudad muy diferente

[14]:
```
distance_scores = list(enumerate(distance_matrix[indexOfCity-1]))
distance_scores
```

[14]:
```
[(0, 0.947667034987109),
 (1, 0.9864593018214056),
 (2, 0.9766593118281239),
 (3, 0.9601077051189635),
 (4, 0.9513032428580988),
 (5, 0.9721357108581354),
 (6, 0.9769962465491707),
 (7, 0.9881395086539354),
 (8, 0.9554911532576069),
 (9, 0.9985130262843562),
 (10, 0.9456957832450614),
 (11, 0.9638187186778284),
 (12, 0.9814120464615329),
 (13, 0.9291534313959126),
 (14, 0.9888511278964539),
```

```
(15, 0.967717639750795),
(16, 0.0),
(17, 0.8202857948945121),
(18, 0.9446571839396732),
(19, 0.5879309869473),
(20, 0.9749414449265978),
(21, 0.9628294283858932),
(22, 0.9624839955173763),
(23, 0.9776270719337808),
(24, 0.9690233577384336),
(25, 0.9664282937757258),
(26, 0.9849262799858077),
(27, 0.9372111300770123),
(28, 0.986813754974884),
(29, 0.9683429059745279),
(30, 0.9457657122109538),
(31, 0.9426749258886973),
(32, 0.9646151637807513),
(33, 0.9734713683921247),
(34, 0.9771189948790237),
(35, 0.9503056620172602),
(36, 0.9749131650299394),
(37, 0.9703573379511877),
(38, 0.9445956301127577),
(39, 0.9760380014545911),
(40, 0.9682436151143141),
(41, 0.9879699044012097),
(42, 0.9646550831319648),
(43, 0.9600514569205071),
(44, 0.9634484725398099),
(45, 0.9756316936347115),
(46, 0.9859143482795854),
(47, 0.9802971205642919),
(48, 0.957372960913624),
(49, 0.9797440827406757),
(50, 0.9749555380071211),
(51, 0.9747307193813092),
(52, 0.9701173896843269),
(53, 0.9550009342735407),
(54, 0.9404405887680177),
(55, 0.973538226015892),
(56, 0.9367713902279169),
(57, 0.9678704974932698),
(58, 0.9903349401154029),
(59, 0.9547005538596594),
(60, 0.9622119480270956),
(61, 0.9504400444542347),
```

```
 (62, 0.9650681504901736)]
```

Vamos a ordenar estas distancias para mostrar la ciudad más similar y solo cogemos las 10 primeras

[15]:
```
ordered_scores = sorted(distance_scores, key=lambda x: x[1])
print("Ciudades más parecidas:")
top_scores = ordered_scores[1:11]
top_scores
```

```
Ciudades más parecidas:
```

[15]:
```
[(19, 0.5879309869473),
 (17, 0.8202857948945121),
 (13, 0.9291534313959126),
 (56, 0.9367713902279169),
 (27, 0.9372111300770123),
 (54, 0.9404405887680177),
 (31, 0.9426749258886973),
 (38, 0.9445956301127577),
 (18, 0.9446571839396732),
 (10, 0.9456957832450614)]
```

Con lo cual, las listas de las **menos parecidas** estarán al final. Vamos a cogerlas!

[16]:
```
distint_scores = ordered_scores[-10:]
distint_scores
```

[16]:
```
[(12, 0.9814120464615329),
 (26, 0.9849262799858077),
 (46, 0.9859143482795854),
 (1, 0.9864593018214056),
 (28, 0.986813754974884),
 (41, 0.9879699044012097),
 (7, 0.9881395086539354),
 (14, 0.9888511278964539),
 (58, 0.9903349401154029),
 (9, 0.9985130262843562)]
```

[17]:
```
top_indexes = [i[0] for i in top_scores]
top_indexes
```

[17]:
```
[19, 17, 13, 56, 27, 54, 31, 38, 18, 10]
```

[18]:
```
distint_indexes = [i[0] for i in distint_scores]
distint_indexes
```

[18]:
```
[12, 26, 46, 1, 28, 41, 7, 14, 58, 9]
```

[19]:
```
preprocessedData['City'].iloc[top_indexes].tolist()
```

[19]:
```
['Macau',
 'Beijing',
```

```
 'New York',
 'Ganzhou',
 'Cairo',
 'Wuhan',
 'Bangkok',
 'Hamburg',
 'Tokyo',
 'Barcelona']
```

[20]:
```
preprocessedData['City'].iloc[distint_indexes]
```

[20]:
```
13       Mexico City
27              Oslo
47        Manchester
2             Berlin
29         Amsterdam
42          Dortmund
8               Vigo
15    Rio de Janeiro
59              Giza
10          Santiago
Name: City, dtype: object
```

5.1 Lista definitiva de las ciudades más parecidas

Dado que las tuplas son inmutables, vamos a pasar nuestra lista de tuplas *top_scores* a una lista de listas que contenta 3 elementos

- Índice de la ciudad en el DataFrame
- Distancia al elemento dado
- Nombre de la ciudad

Con lo cual nuestra lista quedaría de la siguiente forma:
[[indice1 , distancia1 , ciudad1],[indice2 , distancia2 , ciudad2],...]

[21]:
```
lista = list()
tupla = tuple()
listaSimilares = preprocessedData['City'].iloc[top_indexes].tolist()

for i in top_scores:
  sublista = list()
  for a in i:
    sublista.append(a)

  lista.append(sublista)

for i in lista:
  i.append(preprocessedData["City"].iloc[i[0]])
```

```
for i in lista:
  print(i)
```

```
[19, 0.5879309869473, 'Macau']
[17, 0.8202857948945121, 'Beijing']
[13, 0.9291534313959126, 'New York']
[56, 0.9367713902279169, 'Ganzhou']
[27, 0.9372111300770123, 'Cairo']
[54, 0.9404405887680177, 'Wuhan']
[31, 0.9426749258886973, 'Bangkok']
[38, 0.9445956301127577, 'Hamburg']
[18, 0.9446571839396732, 'Tokyo']
[10, 0.9456957832450614, 'Barcelona']
```

5.2 Lista definitiva de las ciudades más diferentes

Dado que las tuplas son inmutables, vamos a pasar nuestra lista de tuplas *distint_scores* a una lista de listas que contenta 3 elementos

- Índice de la ciudad en el DataFrame
- Distancia al elemento dado
- Nombre de la ciudad

Con lo cual nuestra lista quedaría de la siguiente forma:
[[indice1 , distancia1 , ciudad1],[indice2 , distancia2 , ciudad2],...]

[22]:
```
listaR = list()
listaDiferentes = preprocessedData['City'].iloc[distint_indexes].tolist()

for i in distint_scores:
  sublista = list()
  for a in i:
    sublista.append(a)

  listaR.append(sublista)

for i in listaR:
  i.append(preprocessedData["City"].iloc[i[0]])

for i in listaR:
  print(i)
```

```
[12, 0.9814120464615329, 'Mexico City']
[26, 0.9849262799858077, 'Oslo']
[46, 0.9859143482795854, 'Manchester']
[1, 0.9864593018214056, 'Berlin']
[28, 0.986813754974884, 'Amsterdam']
[41, 0.9879699044012097, 'Dortmund']
```

```
[7, 0.9881395086539354, 'Vigo']
[14, 0.9888511278964539, 'Rio de Janeiro']
[58, 0.9903349401154029, 'Giza']
[9, 0.9985130262843562, 'Santiago']
```

6 Creación de nuestra propia fórmula

Vamos a crear nuestra propia fórmula y así poder recomedar mejor las ciudades

Para empezar, si una ciudad está en el mismo pais que una dada, vamos a multiplicarla por 0.92. Vamos a ver como, si seleccionamos Hong Kong como antes, nos aparerá resultados un poco más similaes trás aplicar nuestra formula

Listado de ciudades sin aplicar la fórmula:

- Macau
- Beijing
- New York
- Ganzhou
- Cairo
- Wuhan
- Bangkok
- Hamburg
- Tokyo
- Barcelona

Vamos a ver el código para aplicar nuestra fórmula y el listado final para comparar resultados

[23]:
```
searchCity = "Hong Kong" #Ciudad base para las recomendaciones
indexOfCity = preprocessedData[preprocessedData['City']==searchCity].index.
 ↪values[0]
indexOfCity
```

[23]:
```
17
```

[24]:
```
#hay que restar uno al índice porque nuestro dataset no empieza en 0 si no en 1
ciudad = preprocessedData.iloc[indexOfCity-1]
ciudad
```

[24]:
```
City                                                        Hong Kong
Description                 Hong Kong , officially the Hong Kong Special A...
Country                                                         China
Continent                                                          AS
LAT                                                        22.3964280
LON                                                       114.1094970
processed_text_description  hong kong offici hong kong special administr r...
Name: 17, dtype: object
```

[25]:
```
distance_scores = list(enumerate(distance_matrix[indexOfCity-1]))
```

[26]:
```
pais = preprocessedData.iloc[indexOfCity-1]["Country"]
pais
```

[26]:
```
'China'
```

[27]:
```
l = list()

for i in distance_scores:
  t = tuple()
  indice = i[0]
  valor = i[1]
  pais_aux = preprocessedData.iloc[i[0]]["Country"]
  if pais == pais_aux:
    valor = valor*0.92
  t = (indice,valor)
  l.append(t)
#l
```

[28]:
```
ordered_scores = sorted(l, key=lambda x: x[1])
print("Ciudades más parecidas:")
top_scores = ordered_scores[1:11]
top_indexes = [i[0] for i in top_scores]
preprocessedData['City'].iloc[top_indexes].tolist()
```

```
Ciudades más parecidas:
```

[28]:
```
['Macau',
 'Beijing',
 'Ganzhou',
 'Wuhan',
 'Fuyang',
 'Heze',
 'New York',
 'Cairo',
 'Bangkok',
 'Hamburg']
```

Vemos como nuestra lista ahora está mucho mejor a la vista en la lista anterior

7 CRUD

Vamos a hacer una simulación de como se podría crear, ver, actualizar o eliminar ciudades en nuestra aplicación

7.1 *Create*

Crear un ciudad. Añadir una nueva ciudad a nuestro DataFrame

[29]:
```
city_name = "Cangas"
city_description = "Cangas is a very beautiful place to live on the coast of␣
 ↪Galicia with incredible beaches and very nice people"
country = "Spain"
continent = "EU"

preprocessedText = []

text = word_tokenize(city_description) ## le pasamos el user_input
## Remove stop words
stops = set(stopwords.words("english"))
text = [ps.stem(w) for w in text if not w in stops and w.isalnum()]
text = " ".join(text)

preprocessedText.append(text)

preprocessedText

user_row = {'City':city_name, 'Description': city_description, 'Country':␣
 ↪country, 'Continent':continent, 'processed_text_description':
 ↪preprocessedText[0]}
preprocessedData2 = preprocessedData.append(user_row,ignore_index=True)

preprocessedData2
```

[29]:

	City	...	processed_text_description
0	Paris	...	pari capit popul citi franc estim popul resid ...
1	Berlin	...	berlin world citi cultur polit media scienc It...
2	Madrid	...	the madrid urban agglomer gdp european union i...
3	Istanbul	...	istanbul largest citi turkey countri econom cu...
4	Moscow	...	As northernmost coldest megac world histori da...
..	...	...	...
59	Ankara	...	Is capit turkey locat central part anatolia ci...
60	Houston	...	Is popul citi texa popul citi unit state popul...
61	Atlanta	...	It serv cultur econom center atlanta metropoli...
62	Toronto	...	Is capit citi canadian provinc ontario with re...
63	Cangas	...	canga beauti place live coast galicia incred b...

[64 rows x 7 columns]

Una vez tenemos el nuevo DataFrame, hay que calcular de nuevo los TF-IDF
Calculando los nuevos TF_IDF

[30]:
```
bagOfWordsModel = TfidfVectorizer()
bagOfWordsModel.fit(preprocessedData2['processed_text_description'])
textsBoW= bagOfWordsModel.
 ↪transform(preprocessedData2['processed_text_description'])
textsBoW.shape
```

```
distance_matrix = pairwise_distances(textsBoW,textsBoW ,metric='cosine')
pd.DataFrame(distance_matrix)
```

[30]:

```
          0         1         2   ...        61        62        63
0   0.000000  0.973901  0.930801  ...  0.961019  0.915278  0.981868
1   0.973901  0.000000  0.911250  ...  0.961540  0.943804  1.000000
2   0.930801  0.911250  0.000000  ...  0.914975  0.916901  0.985436
3   0.921229  0.961668  0.922173  ...  0.896647  0.890523  1.000000
4   0.890813  0.943236  0.932353  ...  0.954804  0.941776  1.000000
..       ...       ...       ...  ...       ...       ...       ...
59  0.955203  0.992811  0.977241  ...  0.923935  0.944099  1.000000
60  0.920461  0.952614  0.951664  ...  0.923759  0.884579  1.000000
61  0.961019  0.961540  0.914975  ...  0.000000  0.928388  0.985452
62  0.915278  0.943804  0.916901  ...  0.928388  0.000000  1.000000
63  0.981868  1.000000  0.985436  ...  0.985452  1.000000  0.000000

[64 rows x 64 columns]
```

7.2 *Read*

Vamos a ver como buscar un Ciudad y leer su descripción y ver en qué país está y en que continente

[31]:

```
searchCity = "Cangas"
preprocessedData2[preprocessedData2['City']==searchCity]
#preprocessedData2[preprocessedData2['City']==searchCity]["City"]
#Vamos a conseguir poner esto en un formato mejor
```

[31]:

```
      City  ...                        processed_text_description
63  Cangas  ...  canga beauti place live coast galicia incred b...

[1 rows x 7 columns]
```

Vamos a mostrar esto en un mejor formato para mostrarlo en la interfaz. Para ello, comenzaremos haciendo una lista con el resultado de la serie que conseguimos

[32]:

```
lista = list(preprocessedData2[preprocessedData2['City']==searchCity].iloc[0])
lista
```

[32]:

```
['Cangas',
 'Cangas is a very beautiful place to live on the coast of Galicia with
incredible beaches and very nice people',
 'Spain',
 'EU',
 nan,
 nan,
 'canga beauti place live coast galicia incred beach nice peopl']
```

Una vez tenemos esto, es mucho más fácil acceder a los elementos de nuestra ciudad buscada

[33]:

```
print("City: " + lista[0] + "\n"+
      "Description: " + lista[1] + "\n" +
```

```
      "Country: " + lista[2] + "\n" +
      "Continent: " + lista[3])
```

```
City: Cangas
Description: Cangas is a very beautiful place to live on the coast of Galicia
with incredible beaches and very nice people
Country: Spain
Continent: EU
```

7.3 *Update*

Vamos a ver como actualizar una ciudad existente en el DataFrame

[34]:
```
city = "Cangas"
description = "Cangas is a very beautiful place to live on the coast of Galicia␣
 ↪with incredible beaches and very nice people"
country = "GALICIA"
continent = "EU"
```

Para actualizar una fila de nuestro DataFrame vamos a utilizar el método at[index, column] = value

[35]:
```
preprocessedData2[preprocessedData2['City']==searchCity].index
preprocessedData2.at[preprocessedData2[preprocessedData2['City']==searchCity].
 ↪index,"Country"] = country
preprocessedData2
```

[35]:

	City	...	processed_text_description
0	Paris	...	pari capit popul citi franc estim popul resid ...
1	Berlin	...	berlin world citi cultur polit media scienc It...
2	Madrid	...	the madrid urban agglomer gdp european union i...
3	Istanbul	...	istanbul largest citi turkey countri econom cu...
4	Moscow	...	As northernmost coldest megac world histori da...
..	...	...	...
59	Ankara	...	Is capit turkey locat central part anatolia ci...
60	Houston	...	Is popul citi texa popul citi unit state popul...
61	Atlanta	...	It serv cultur econom center atlanta metropoli...
62	Toronto	...	Is capit citi canadian provinc ontario with re...
63	Cangas	...	canga beauti place live coast galicia incred b...

[64 rows x 7 columns]

Tenemos que volver a actualizar las distancias por si se ha modificado el campo de descripción!

[36]:
```
bagOfWordsModel = TfidfVectorizer()
bagOfWordsModel.fit(preprocessedData2['processed_text_description'])
textsBoW= bagOfWordsModel.
 ↪transform(preprocessedData2['processed_text_description'])
textsBoW.shape

distance_matrix = pairwise_distances(textsBoW,textsBoW ,metric='cosine')
```

```
pd.DataFrame(distance_matrix)
```

[36]:

	0	1	2	...	61	62	63
0	0.000000	0.973901	0.930801	...	0.961019	0.915278	0.981868
1	0.973901	0.000000	0.911250	...	0.961540	0.943804	1.000000
2	0.930801	0.911250	0.000000	...	0.914975	0.916901	0.985436
3	0.921229	0.961668	0.922173	...	0.896647	0.890523	1.000000
4	0.890813	0.943236	0.932353	...	0.954804	0.941776	1.000000
..	...	...	...	...	...	...	...
59	0.955203	0.992811	0.977241	...	0.923935	0.944099	1.000000
60	0.920461	0.952614	0.951664	...	0.923759	0.884579	1.000000
61	0.961019	0.961540	0.914975	...	0.000000	0.928388	0.985452
62	0.915278	0.943804	0.916901	...	0.928388	0.000000	1.000000
63	0.981868	1.000000	0.985436	...	0.985452	1.000000	0.000000

[64 rows x 64 columns]

7.4 *Delete*

Vamos a ver como eliminar una ciudad existente en nuestro DataFrame

[37]:
```
city = "Cangas"
```

Para eliminar una ciudad en nuestro DataFrame, vamos a usar el método drop()

[38]:
```
preprocessedData2 = preprocessedData2.
 ↪drop(preprocessedData2[preprocessedData2['City']==city].index,axis=0)
preprocessedData2
```

[38]:

	City	...	processed_text_description
0	Paris	...	pari capit popul citi franc estim popul resid ...
1	Berlin	...	berlin world citi cultur polit media scienc It...
2	Madrid	...	the madrid urban agglomer gdp european union i...
3	Istanbul	...	istanbul largest citi turkey countri econom cu...
4	Moscow	...	As northernmost coldest megac world histori da...
..	...	...	...
58	Giza	...	Is citi egypt cairo citi africa kinshasa lago ...
59	Ankara	...	Is capit turkey locat central part anatolia ci...
60	Houston	...	Is popul citi texa popul citi unit state popul...
61	Atlanta	...	It serv cultur econom center atlanta metropoli...
62	Toronto	...	Is capit citi canadian provinc ontario with re...

[63 rows x 7 columns]

Podemos ver como nuestra ciudad "Cangas" ya no está en el DataFrame
Tenemos que volver a calcular las distancias!

[39]:
```
bagOfWordsModel = TfidfVectorizer()
bagOfWordsModel.fit(preprocessedData2['processed_text_description'])
textsBoW= bagOfWordsModel.
 ↪transform(preprocessedData2['processed_text_description'])
```

```
textsBoW.shape

distance_matrix = pairwise_distances(textsBoW,textsBoW ,metric='cosine')
pd.DataFrame(distance_matrix)
```

[39]:

	0	1	2	...	60	61	62
0	0.000000	0.974079	0.930827	...	0.921115	0.961461	0.915960
1	0.974079	0.000000	0.911556	...	0.952799	0.961741	0.943994
2	0.930827	0.911556	0.000000	...	0.951956	0.915403	0.917284
3	0.921908	0.961887	0.922449	...	0.942381	0.897336	0.891292
4	0.891645	0.943571	0.932833	...	0.948079	0.955208	0.942102
..	...	...	...	...	...	...	...
58	0.964453	0.977125	0.964068	...	0.973141	0.976243	0.959212
59	0.955816	0.992906	0.977443	...	0.947365	0.924269	0.944634
60	0.921115	0.952799	0.951956	...	0.000000	0.924276	0.885217
61	0.961461	0.961741	0.915403	...	0.924276	0.000000	0.928837
62	0.915960	0.943994	0.917284	...	0.885217	0.928837	0.000000

[63 rows x 63 columns]

8 DATABASE

En nuestra aplicación tendremos que guardar de alguna forma la información de las ciudades, usuarios, contraseñas... Para ello usaremos una base de datos con un modelo relación gracias a **sqlite3**.

8.1 *Usuarios*

Para empezar, tendremos una tabla llamada *user* donde guardaremos la informacion relacionada con los usuarios. Esta tabla tendrá los siguientes campos: * Username String * Password String * Privileges Boolean

Veamos un ejemplo de como sería esta tabla

```
usersDataFrame = pd.DataFrame({"Username": "marcos", "Password": "marcos",␣
↪"Privileges":True},index=[0])
usersDataFrame
```

[40]:

	Username	Password	Privileges
0	marcos	marcos	True

El siguiente código intenta mostrar como se realizará la gestión de la tabla en nuestra aplicación web

```
# DB Management
import sqlite3
conn = sqlite3.connect('data.db')
c = conn.cursor()
```

```
def create_usertable():
        c.execute('CREATE TABLE IF NOT EXISTS userstable(username TEXT,password␣
 ↪TEXT,privileges TEXT default FALSE, PRIMARY KEY (username))')

def add_userdata(username,password):
        if username == "admin":
                c.execute('INSERT INTO userstable(username,password,privileges)␣
 ↪VALUES (?,?,?)',(username,password,True))
        else:
                c.execute('INSERT INTO userstable(username,password) VALUES (?,?
 ↪)',(username,password))
        conn.commit()

def login_user(username,password):
        c.execute('SELECT * FROM userstable WHERE username =? AND password = ?
 ↪',(username,password))
        data = c.fetchall()
        return data

def view_all_users():
        c.execute('SELECT * FROM userstable')
        data = c.fetchall()
        return data
```

8.2 *Ciudades*

Las ciudades será lo realmente importante en nuestra aplicación. Tendremos que pasar del DataFrame cargado a una tabla relacional para poder operar con SQL. La tabla de las ciudades tendrá los siguientes campos:

- Name
- Description
- Country
- Continent
- Processed_text_description

Lo primero será comvertir nuestro DataFrame llamado *preprocessedData* a una tabla SQL. Para ello, usaremos el método .to_sql ya implementado en pandas para un DF.

[42]:
```
#from sqlalchemy import create_engine
# DB Management
#import sqlite3
#conn = sqlite3.connect('data.db')
#c = conn.cursor()

#engine = create_engine('sqlite://', echo=False)
#preprocessedData.to_sql('city',con=conn)
```

```
#conn.cursor().execute('INSERT INTO city(City,Description,Country,Continent,␣
 ↪Processed_text_description) VALUES (?,?,?,?,?)',("algo", "asidaisdia aidjais␣
 ↪dis el want to travel arround the world", "Spain", "EU","Null"))
#conn.cursor().execute("SELECT * FROM city").fetchall()
```

Una vez ya lo hemos conseguido y vemos todo el resultado de cada fila en tuplas con sus respectivos campos, vamos a ver los métodos que vamos a emplear en la aplicación web

8.3 *Comentarios*

Un usuario podrá dejar un comentario sobre una ciudad que ya haya visitado. Para ello, utilizaremos otra tabla en nuestra base de datos. En este caso se llamará *comment*. Esta tabla tendrá los siguientes campos:

- Username String
- City String
- Content String

Vamos a ver un ejemplo de cómo sería esta tabla:

[43]:
```
commentsDataFrame = pd.DataFrame({"Username": "marcos","City": "Paris",␣
 ↪"Content": "I did not like this city too much because it is very expensive␣
 ↪and there are few things to see"},index=[0])
commentsDataFrame
```

[43]:
```
  Username   City                                               Content
0   marcos  Paris  I did not like this city too much because it i...
```

Y la gestión en nuestra aplicación se realizará de la siguiente forma:

[44]:
```
# DB Management
import sqlite3
conn = sqlite3.connect('data.db')
c = conn.cursor()

def create_commenttable():
        c.execute('CREATE TABLE IF NOT EXISTS commenttable(username TEXT, city␣
 ↪TEXT, content TEXT, PRIMARY KEY (username,city))')

def add_commentdata(username, city, content):
        c.execute('INSERT INTO commenttable(username,city,content) VALUES (?,?,?
 ↪)',(username, city, content))
        conn.commit()

def view_all_comments():
        c.execute('SELECT * FROM commenttable')
        data = c.fetchall()
        return data

def view_all_comments_city(city):
        c.execute('SELECT * FROM commenttable WHERE city LIKE (?)',(city))
```

```
        data = c.fetchall()
        return data
```

9 COMENTARIOS

Nuestro sistema también podrá catalogar si un comentario es positivo, normal o negativo. Para ello, usaremos el csv *semeval-2017-train.csv*

[45]:
```
trainingData = pd.read_csv('semeval-2017-train.csv', delimiter='        ')
trainingData = trainingData.head(15000) #Eliminar la funcion head() si se␣
 ↪quiere usar todo el dataset. Para las pruebas usamos únicamente los 1000␣
 ↪primeros tweets
nltk.download('punkt')
nltk.download('stopwords')

ps = PorterStemmer()

preprocessedText = []

for row in trainingData.itertuples():


    text = word_tokenize(row[2]) ## indice de la columna que contiene el texto
    ## Remove stop words
    stops = set(stopwords.words("english"))
    text = [ps.stem(w) for w in text if not w in stops and w.isalnum()]
    text = " ".join(text)

    preprocessedText.append(text)

preprocessedData = trainingData
preprocessedData['processed_text'] = preprocessedText

preprocessedData
```

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

[45]:

	label	...	processed_text
0	1	...	one night like In vega I make dat nigga famou
1	1	...	walk chelsea time day rather love love london ...
2	0	...	and first play night aaron rodger int udfa CB ...
3	0	...	drove bike today 40 mile felt like jim carrey ...

```
4         -1  ...  look temp outsid get hotter sun goe feel like ...
...      ...  ...                                                ...
14995     -1  ...  cheringbieb MY birthday ON februari My friend ...
14996     -1  ...  step 1 put petraeu job ca talk much step 2 abs...
14997      0  ...        coolzjay I deca pictur tomorrow time pictur
14998      1  ...       craigoyok haha got ta give info tomorrow lol
14999      1  ...       fca tomorrow morn pac small group I hope see

[15000 rows x 3 columns]
```

Creación de la bolsa de palabras

[46]:
```
bagOfWordsModel = TfidfVectorizer()
bagOfWordsModel.fit(preprocessedData['processed_text'])
textsBoW= bagOfWordsModel.transform(preprocessedData['processed_text'])
```

Vamos a ver cuantas palabras tenemos

[47]:
```
textsBoW.shape
```

[47]:
```
(15000, 22678)
```

Entrenamiento de un algoritmo de clasificación (SVM)

[48]:
```
from sklearn import svm
svc = svm.SVC(kernel='linear') #Modelo de clasificación

X_train = textsBoW #Documentos
Y_train = trainingData['label'] #Etiquetas de los documentos
svc.fit(X_train, Y_train) #Entrenamiento
```

[48]:
```
SVC(kernel='linear')
```

Vamos a poner nosotros unos comentarios de prueba para ver como los puntua nuestro sistema

[49]:
```
testData  = pd.DataFrame({"label": 1, "text": "This city is incredible! There␣
 ↪are many beautiful monuments that you have to see. Worth!"},index=[0])
testData  = testData .append({"label": -1, "text": "I did not like this city at␣
 ↪all. Very boring"},ignore_index=True)
testData
```

[49]:
```
   label                                               text
0      1  This city is incredible! There are many beauti...
1     -1       I did not like this city at all. Very boring
```

[50]:
```
ps = PorterStemmer()

preprocessedText = []

for row in testData.itertuples():


    text = word_tokenize(row[2]) ## indice de la columna que contiene el texto
    ## Remove stop words
```

```
    stops = set(stopwords.words("english"))
    text = [ps.stem(w) for w in text if not w in stops and w.isalnum()]
    text = " ".join(text)

    preprocessedText.append(text)

preprocessedDataTest = testData
preprocessedDataTest['processed_text'] = preprocessedText

preprocessedDataTest
```

[50]:
```
   label  ...                                      processed_text
0      1  ...  thi citi incred there mani beauti monument see...
1     -1  ...                                I like citi veri bore

[2 rows x 3 columns]
```

Se tiene que emplear la misma representación de bolsa de palabras que se ha usado para entrenar!

[51]:
```
textsBoWTest= bagOfWordsModel.transform(preprocessedDataTest['processed_text'])
```

9.1 Clasificación de los documentos de test

[52]:
```
X_test = textsBoWTest #Documentos

predictions = svc.predict(X_test) #Se almacena en el array predictions las␣
 ↪predicciones del clasificador
predictions = list(predictions)
print(predictions)
```

```
[1, -1]
```

9.2 Evaluación de la predicción

[53]:
```
from sklearn.metrics import classification_report

Y_test = testData['label'] #Etiquetas reales de los documentos
Y_test
print(classification_report(Y_test, predictions))
```

```
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00         1
           1       1.00      1.00      1.00         1

    accuracy                           1.00         2
   macro avg       1.00      1.00      1.00         2
```

```
weighted avg       1.00      1.00      1.00         2
```

Podemos ver como acertó los dos! Pruebas hechas: * 1000 -> 50% [1 1] * 6000 -> 50% [1 0] * 15000 -> 100%. [1 -1]

9.3 Caso más real

Vamos a hacer un caso más real ahora donde un usuario introduce solamente un comentario

[54]:
```
User_text = "This city is incredible! There are many beautiful monuments that␣
 ↪you have to see. Worth!"

ps = PorterStemmer()

preprocessedText = []


text = word_tokenize(User_text) ## indice de la columna que contiene el texto
## Remove stop words
stops = set(stopwords.words("english"))
text = [ps.stem(w) for w in text if not w in stops and w.isalnum()]
text = " ".join(text)

preprocessedText.append(text)

#preprocessedDataTest = testData
#preprocessedDataTest['processed_text'] = preprocessedText

print(preprocessedText)

textsBoWTest= bagOfWordsModel.transform(preprocessedText)
X_test = textsBoWTest #Documentos

predictions = svc.predict(X_test) #Se almacena en el array predictions las␣
 ↪predicciones del clasificador
print(predictions)
```

```
['thi citi incred there mani beauti monument see worth']
[1]
```

Vemos como tenemos dos listas, una para el texto procesado y la segunda para la valoración que nos dice un 1, con lo cual, es que es un buen comentario.

Con lo cual, esto es justo lo que queremos que haga en nuestro sistema!

10 HOTELES

Vamos a añadir un nuevo dataset a nuestro proyecto para mostrar los mejores hoteles a una ciudad que quiera visitar el usuario

[55]:
```
import re
import pandas as pd
df_hoteles = pd.read_csv("hoteles.csv",encoding='unicode_escape')
df_hoteles
```

[55]:
```
                                    Hotel name continent_name  ... info.6
info.7
0                       Shangri-La Hotel, Beijing           Asia  ...    NaN
NaN
1                InterContinental Beijing Sanlitun           Asia  ...    NaN
NaN
2             Holiday Inn Express Beijing Yizhuang           Asia  ...    NaN
NaN
3      Shangri-La China World Summit Wing, Beijing           Asia  ...    NaN
NaN
4                              Kerry Hotel Beijing           Asia  ...    NaN
NaN
...                                            ...            ...  ...    ...
...
59702                         The Bright Resort  South Pacific  ...    NaN
NaN
59703                      Bogong View Motor Inn  South Pacific  ...    NaN
NaN
59704                      Autumn Abode Cottages  South Pacific  ...    NaN
NaN
59705                                The Odd Frog  South Pacific  ...    NaN
NaN
59706                     View Hill Holiday Units  South Pacific  ...    NaN
NaN

[59707 rows x 14 columns]
```

[56]:
```
df_hoteles["country_name"] == "China"
```

[56]:
```
0        False
1        False
2        False
3        False
4        False
         ...
59702    False
59703    False
59704    False
59705    False
59706    False
Name: country_name, Length: 59707, dtype: bool
```

10.1 Tratamiento del DataFrame de Hoteles

Como en las columnas *city_name* y *country_name* tienen espacios añadidos, tenemos que hacer un tratamiento para quitar dichos espacios. Hacemos esto porque si no, no concuerdan las búsquedas

Para hacer este tratamiento, vamos a utilizar la función *apply* implementada en la librería **pandas** y también *strip()* que es un método de String que elimina los espacios al principio y al final

[57]:
```
#for row in df_hoteles.itertuples():
df_hoteles["city_name"] = df_hoteles["city_name"].apply(lambda x : str(x).
 ↪strip())
df_hoteles["country_name"] = df_hoteles["country_name"].apply(lambda x : str(x).
 ↪strip())
```

[58]:
```
df_hoteles
```

[58]:
```
                                    Hotel name continent_name  ... info.6
info.7
0                       Shangri-La Hotel, Beijing           Asia  ...    NaN
NaN
1                InterContinental Beijing Sanlitun           Asia  ...    NaN
NaN
2             Holiday Inn Express Beijing Yizhuang           Asia  ...    NaN
NaN
3      Shangri-La China World Summit Wing, Beijing           Asia  ...    NaN
NaN
4                              Kerry Hotel Beijing           Asia  ...    NaN
NaN
...                                            ...            ...  ...    ...
...
59702                          The Bright Resort  South Pacific  ...    NaN
NaN
59703                       Bogong View Motor Inn  South Pacific  ...    NaN
NaN
59704                       Autumn Abode Cottages  South Pacific  ...    NaN
NaN
59705                                The Odd Frog  South Pacific  ...    NaN
NaN
59706                      View Hill Holiday Units  South Pacific  ...    NaN
NaN

[59707 rows x 14 columns]
```

Como veremos a continuación, ya puede concordar los elementos de búsqueda

[59]:
```
df_hoteles["country_name"] == "China"
```

[59]:
```
0         True
1         True
2         True
3         True
4         True
```

```
        ...
59702    False
59703    False
59704    False
59705    False
59706    False
Name: country_name, Length: 59707, dtype: bool
```

10.2 Ejemplo de uso

Vamos a ver un ejemplo de uso donde un usuario selecciona una ciudad, por ejemplo *"Paris"*, para mostrar información importante de dicha ciudad.

A mayores de toda la información, se le mostrará los mejores hoteles que están en nuestro datafame *df_hoteles*

[60]:
```
user_city = "Paris"
df_hoteles[df_hoteles["city_name"] == "Paris"]
```

[60]:

	Hotel name	...	info.7
16697	Hotel Malte - Astotel	...	NaN
16698	Hotel Ekta	...	pay at stay
16699	Hotel La Manufacture	...	NaN
16700	Hotel Eiffel Blomet	...	NaN
16701	Hotel Da Vinci & Spa	...	NaN
16702	Hotel 34B - Astotel	...	NaN
16703	B Montmartre Hotel	...	NaN
16704	Hotel Mademoiselle	...	NaN
16705	Secret de Paris - Hotel & Spa	...	NaN
16706	Hotel des Grands Hommes	...	NaN
16707	Hotel Terminus Lyon	...	NaN
16708	Hotel Darcet	...	NaN
16709	Pullman Paris Eiffel Tower Hotel	...	NaN
16710	Hotel Rose Bourbon	...	NaN
16711	Hotel CitizenM Paris Gare de Lyon	...	NaN
16712	Les Jardins de Mademoiselle	...	NaN
16713	Le Relais Saint Charles	...	NaN
16714	Hotel Apollon Montparnasse	...	NaN
16715	Hotel L'interlude	...	NaN
16716	Hotel Galileo	...	pay at stay
16717	Hyatt Regency Paris Etoile	...	NaN
16718	Hotel Maxim Folies	...	pay at stay
16719	Timhotel Paris Gare de l'Est	...	NaN
16720	Fred Hotel	...	NaN
16721	Le 55 Montparnasse Hotel	...	NaN
16722	Hotel Paris Vaugirard	...	NaN
16723	Timhotel Paris Berthier	...	NaN
16724	Hotel Orchidee	...	pay at stay
16725	Kyriad Paris 13 - Italie Gobelins	...	NaN

```
16726     Citadines Place d'Italie Paris  ...           NaN

[30 rows x 14 columns]
```

11 STREAMLIT

Vamos a lanzar Streamlit para mostrar estos resultados en la web. Todo lo mencionado anteriormente se implementa

11.1 Instalación de librerias

Se instala streamlit y pyngrok

[61]:
```
!pip install streamlit

!pip install pyngrok
```

```
Collecting streamlit
  Downloading streamlit-1.3.1-py2.py3-none-any.whl (9.2 MB)
     || 9.2 MB 7.3 MB/s
Collecting pydeck>=0.1.dev5
  Downloading pydeck-0.7.1-py2.py3-none-any.whl (4.3 MB)
     || 4.3 MB 58.9 MB/s
Requirement already satisfied: tornado>=5.0 in /usr/local/lib/python3.7
/dist-packages (from streamlit) (5.1.1)
Requirement already satisfied: click<8.0,>=7.0 in /usr/local/lib/python3.7/dist-
packages (from streamlit) (7.1.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-
packages (from streamlit) (21.3)
Requirement already satisfied: cachetools>=4.0 in /usr/local/lib/python3.7/dist-
packages (from streamlit) (4.2.4)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/dist-
packages (from streamlit) (7.1.2)
Requirement already satisfied: altair>=3.2.0 in /usr/local/lib/python3.7/dist-
packages (from streamlit) (4.1.0)
Collecting watchdog
  Downloading watchdog-2.1.6-py3-none-manylinux2014_x86_64.whl (76 kB)
     || 76 kB 5.6 MB/s
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-
packages (from streamlit) (1.19.5)
Requirement already satisfied: astor in /usr/local/lib/python3.7/dist-packages
(from streamlit) (0.8.1)
Collecting blinker
  Downloading blinker-1.4.tar.gz (111 kB)
     || 111 kB 58.0 MB/s
```

```
Collecting validators
  Downloading validators-0.18.2-py3-none-any.whl (19 kB)
Collecting base58
  Downloading base58-2.1.1-py3-none-any.whl (5.6 kB)
Collecting gitpython!=3.1.19
  Downloading GitPython-3.1.26-py3-none-any.whl (180 kB)
     || 180 kB 55.5 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from streamlit) (2.23.0)
Requirement already satisfied: tzlocal in /usr/local/lib/python3.7/dist-packages
(from streamlit) (1.5.1)
Requirement already satisfied: pandas>=0.21.0 in /usr/local/lib/python3.7/dist-
packages (from streamlit) (1.1.5)
Collecting pympler>=0.9
  Downloading Pympler-1.0.1-py3-none-any.whl (164 kB)
     || 164 kB 59.5 MB/s
Requirement already satisfied: pyarrow in /usr/local/lib/python3.7/dist-
packages (from streamlit) (3.0.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-
packages (from streamlit) (2.8.2)
Requirement already satisfied: protobuf!=3.11,>=3.6.0 in
/usr/local/lib/python3.7/dist-packages (from streamlit) (3.17.3)
Requirement already satisfied: toml in /usr/local/lib/python3.7/dist-packages
(from streamlit) (0.10.2)
Requirement already satisfied: attrs in /usr/local/lib/python3.7/dist-packages
(from streamlit) (21.2.0)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.7/dist-packages
(from altair>=3.2.0->streamlit) (2.11.3)
Requirement already satisfied: toolz in /usr/local/lib/python3.7/dist-packages
(from altair>=3.2.0->streamlit) (0.11.2)
Requirement already satisfied: jsonschema in /usr/local/lib/python3.7/dist-
packages (from altair>=3.2.0->streamlit) (2.6.0)
Requirement already satisfied: entrypoints in /usr/local/lib/python3.7/dist-
packages (from altair>=3.2.0->streamlit) (0.3)
Collecting gitdb<5,>=4.0.1
  Downloading gitdb-4.0.9-py3-none-any.whl (63 kB)
     || 63 kB 1.7 MB/s
Requirement already satisfied: typing-extensions>=3.7.4.3 in
/usr/local/lib/python3.7/dist-packages (from gitpython!=3.1.19->streamlit)
(3.10.0.2)
Collecting smmap<6,>=3.0.1
  Downloading smmap-5.0.0-py3-none-any.whl (24 kB)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.7/dist-
packages (from pandas>=0.21.0->streamlit) (2018.9)
Requirement already satisfied: six>=1.9 in /usr/local/lib/python3.7/dist-
packages (from protobuf!=3.11,>=3.6.0->streamlit) (1.15.0)
Requirement already satisfied: traitlets>=4.3.2 in /usr/local/lib/python3.7
/dist-packages (from pydeck>=0.1.dev5->streamlit) (5.1.1)
```

```
Requirement already satisfied: ipywidgets>=7.0.0 in /usr/local/lib/python3.7
/dist-packages (from pydeck>=0.1.dev5->streamlit) (7.6.5)
Collecting ipykernel>=5.1.2
  Downloading ipykernel-6.6.1-py3-none-any.whl (126 kB)
     || 126 kB 61.4 MB/s
Requirement already satisfied: debugpy<2.0,>=1.0.0 in
/usr/local/lib/python3.7/dist-packages (from
ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (1.0.0)
Collecting ipython>=7.23.1
  Downloading ipython-7.31.0-py3-none-any.whl (792 kB)
     || 792 kB 51.4 MB/s
Requirement already satisfied: jupyter-client<8.0 in
/usr/local/lib/python3.7/dist-packages (from
ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (5.3.5)
Requirement already satisfied: argcomplete>=1.12.3 in /usr/local/lib/python3.7
/dist-packages (from ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (2.0.0)
Requirement already satisfied: matplotlib-inline<0.2.0,>=0.1.0 in
/usr/local/lib/python3.7/dist-packages (from
ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (0.1.3)
Requirement already satisfied: importlib-metadata<5 in /usr/local/lib/python3.7
/dist-packages (from ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (4.8.2)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.7/dist-
packages (from ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (1.5.4)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-
metadata<5->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (3.6.0)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.7/dist-
packages (from ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(4.8.0)
Requirement already satisfied: decorator in /usr/local/lib/python3.7/dist-
packages (from ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(4.4.2)
Requirement already satisfied: pygments in /usr/local/lib/python3.7/dist-
packages (from ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(2.6.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.7/dist-
packages (from ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(0.2.0)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.7/dist-
packages (from ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(0.18.1)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.7
/dist-packages (from
ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (57.4.0)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.7/dist-
packages (from ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(0.7.5)
Collecting prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0
```

```
  Downloading prompt_toolkit-3.0.24-py3-none-any.whl (374 kB)
     || 374 kB 58.4 MB/s
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in
/usr/local/lib/python3.7/dist-packages (from
ipywidgets>=7.0.0->pydeck>=0.1.dev5->streamlit) (1.0.2)
Requirement already satisfied: widgetsnbextension~=3.5.0 in
/usr/local/lib/python3.7/dist-packages (from
ipywidgets>=7.0.0->pydeck>=0.1.dev5->streamlit) (3.5.2)
Requirement already satisfied: nbformat>=4.2.0 in /usr/local/lib/python3.7/dist-
packages (from ipywidgets>=7.0.0->pydeck>=0.1.dev5->streamlit) (5.1.3)
Requirement already satisfied: ipython-genutils~=0.2.0 in
/usr/local/lib/python3.7/dist-packages (from
ipywidgets>=7.0.0->pydeck>=0.1.dev5->streamlit) (0.2.0)
Requirement already satisfied: parso<0.9.0,>=0.8.0 in /usr/local/lib/python3.7
/dist-packages (from
jedi>=0.16->ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(0.8.3)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7
/dist-packages (from jinja2->altair>=3.2.0->streamlit) (2.0.1)
Requirement already satisfied: pyzmq>=13 in /usr/local/lib/python3.7/dist-
packages (from jupyter-
client<8.0->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (22.3.0)
Requirement already satisfied: jupyter-core>=4.6.0 in /usr/local/lib/python3.7
/dist-packages (from jupyter-
client<8.0->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit) (4.9.1)
Requirement already satisfied: ptyprocess>=0.5 in /usr/local/lib/python3.7/dist-
packages (from
pexpect>4.3->ipython>=7.23.1->ipykernel>=5.1.2->pydeck>=0.1.dev5->streamlit)
(0.7.0)
Requirement already satisfied: wcwidth in /usr/local/lib/python3.7/dist-packages
(from prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0->ipython>=7.23.1->ipykernel>=
5.1.2->pydeck>=0.1.dev5->streamlit) (0.2.5)
Requirement already satisfied: notebook>=4.4.1 in /usr/local/lib/python3.7/dist-
packages (from
widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->pydeck>=0.1.dev5->streamlit)
(5.3.1)
Requirement already satisfied: Send2Trash in /usr/local/lib/python3.7/dist-
packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->py
deck>=0.1.dev5->streamlit) (1.8.0)
Requirement already satisfied: terminado>=0.8.1 in /usr/local/lib/python3.7
/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0
.0->pydeck>=0.1.dev5->streamlit) (0.12.1)
Requirement already satisfied: nbconvert in /usr/local/lib/python3.7/dist-
packages (from notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->py
deck>=0.1.dev5->streamlit) (5.6.1)
Requirement already satisfied: bleach in /usr/local/lib/python3.7/dist-packages
(from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets>=7.0.0->
pydeck>=0.1.dev5->streamlit) (4.1.0)
```

```
Requirement already satisfied: mistune<2,>=0.8.1 in /usr/local/lib/python3.7
/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipyw
idgets>=7.0.0->pydeck>=0.1.dev5->streamlit) (0.8.4)
Requirement already satisfied: pandocfilters>=1.4.1 in /usr/local/lib/python3.7
/dist-packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipyw
idgets>=7.0.0->pydeck>=0.1.dev5->streamlit) (1.5.0)
Requirement already satisfied: defusedxml in /usr/local/lib/python3.7/dist-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets
>=7.0.0->pydeck>=0.1.dev5->streamlit) (0.7.1)
Requirement already satisfied: testpath in /usr/local/lib/python3.7/dist-
packages (from nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ipywidgets
>=7.0.0->pydeck>=0.1.dev5->streamlit) (0.5.0)
Requirement already satisfied: webencodings in /usr/local/lib/python3.7/dist-
packages (from bleach->nbconvert->notebook>=4.4.1->widgetsnbextension~=3.5.0->ip
ywidgets>=7.0.0->pydeck>=0.1.dev5->streamlit) (0.5.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging->streamlit) (3.0.6)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->streamlit) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7
/dist-packages (from requests->streamlit) (2021.10.8)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->streamlit) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7
/dist-packages (from requests->streamlit) (3.0.4)
Building wheels for collected packages: blinker
  Building wheel for blinker (setup.py) ... done
  Created wheel for blinker: filename=blinker-1.4-py3-none-any.whl size=13478
sha256=9c4cebcdfcb9bcf755ccbb166552bf8e2f35ff1b42c62c5fb0dffe95b4c7c39a
  Stored in directory: /root/.cache/pip/wheels/22/f5/18/df711b66eb25b21325c13275
7d4314db9ac5e8dabeaf196eab
Successfully built blinker
Installing collected packages: prompt-toolkit, ipython, ipykernel, smmap, gitdb,
watchdog, validators, pympler, pydeck, gitpython, blinker, base58, streamlit
  Attempting uninstall: prompt-toolkit
    Found existing installation: prompt-toolkit 1.0.18
    Uninstalling prompt-toolkit-1.0.18:
      Successfully uninstalled prompt-toolkit-1.0.18
  Attempting uninstall: ipython
    Found existing installation: ipython 5.5.0
    Uninstalling ipython-5.5.0:
      Successfully uninstalled ipython-5.5.0
  Attempting uninstall: ipykernel
    Found existing installation: ipykernel 4.10.1
    Uninstalling ipykernel-4.10.1:
      Successfully uninstalled ipykernel-4.10.1
```

```
ERROR: pip's dependency resolver does not currently take into account all
the packages that are installed. This behaviour is the source of the following
dependency conflicts.
jupyter-console 5.2.0 requires prompt-toolkit<2.0.0,>=1.0.0, but you have
prompt-toolkit 3.0.24 which is incompatible.
google-colab 1.0.0 requires ipykernel~=4.10, but you have ipykernel 6.6.1 which
is incompatible.
google-colab 1.0.0 requires ipython~=5.5.0, but you have ipython 7.31.0 which is
incompatible.
Successfully installed base58-2.1.1 blinker-1.4 gitdb-4.0.9 gitpython-3.1.26
ipykernel-6.6.1 ipython-7.31.0 prompt-toolkit-3.0.24 pydeck-0.7.1 pympler-1.0.1
smmap-5.0.0 streamlit-1.3.1 validators-0.18.2 watchdog-2.1.6
```

```
Collecting pyngrok
  Downloading pyngrok-5.1.0.tar.gz (745 kB)
     || 745 kB 7.4 MB/s
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-
packages (from pyngrok) (3.13)
Building wheels for collected packages: pyngrok
  Building wheel for pyngrok (setup.py) ... done
  Created wheel for pyngrok: filename=pyngrok-5.1.0-py3-none-any.whl size=19006
sha256=2fe3502adf20309f32d52364238de77bc83ce472e247478e6f7d5ed64eaee9d6
  Stored in directory: /root/.cache/pip/wheels/bf/e6/af/ccf6598ecefecd4410406937
1795cb9b3afbcd16987f6ccfb3
Successfully built pyngrok
Installing collected packages: pyngrok
Successfully installed pyngrok-5.1.0
```

Para ver la documentación sobre la API streamlit : https://docs.streamlit.io/library/api-reference

11.2 Código de la Web

A continuación se escribe todo el código de nuestra web al fichero TraveLearning.py.

[62]:
```
df_hoteles2 = pd.read_csv("hoteles.csv",encoding='unicode_escape')
```

[63]:
```
%%writefile TraveLearning.py
import streamlit as st
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer
import nltk
from sklearn import svm
nltk.download('punkt')
nltk.download('stopwords')
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import pairwise_distances
from PIL import Image
```

```
import numpy as np

#PAGE_CONFIG = {"page_title":"StColab.io","page_icon":":smiley:","layout":
 ↪"centered"}
#st.beta_set_page_config(**PAGE_CONFIG)

# DB Management
import sqlite3
conn = sqlite3.connect('data.db')
c = conn.cursor()

def create_usertable():
        c.execute('CREATE TABLE IF NOT EXISTS userstable(username TEXT,password␣
 ↪TEXT,privileges TEXT default FALSE, PRIMARY KEY (username))')

def add_userdata(username,password):
        if username == "admin":
                c.execute('INSERT INTO userstable(username,password,privileges)␣
 ↪VALUES (?,?,?)',(username,password,True))
        else:
                c.execute('INSERT INTO userstable(username,password) VALUES (?,?
 ↪)',(username,password))
        conn.commit()

def login_user(username,password):
        c.execute('SELECT * FROM userstable WHERE username =? AND password = ?
 ↪',(username,password))
        data = c.fetchall()
        return data

def view_all_users():
        c.execute('SELECT * FROM userstable')
        data = c.fetchall()
        return data

#--------------------------COMENTARIOS----------------------------#
def create_commenttable():
        c.execute('CREATE TABLE IF NOT EXISTS commenttable(username TEXT, city␣
 ↪TEXT, content TEXT, value INT DEFAULT 2, PRIMARY KEY␣
 ↪(username,city,content))')

def add_commentdata(username, city, content):
        c.execute('INSERT INTO commenttable(username,city,content) VALUES (?,?,?
 ↪)',(username, city, content))
        conn.commit()
```

```
def add_commentdata_value(username, city, content, value):
        c.execute('INSERT INTO commenttable(username,city,content,value) VALUES␣
 ↪(?,?,?,?)',(username, city, content,value))
        conn.commit()

def view_all_comments():
        c.execute('SELECT * FROM commenttable')
        data = c.fetchall()
        return data

def view_all_comments_value2():
        c.execute('SELECT * FROM commenttable WHERE value == 2')
        data = c.fetchall()
        return data

def view_all_comments_city(cityOption):
        c.execute('SELECT * FROM commenttable WHERE city==(?)',(cityOption,))
        data = c.fetchall()
        return data

def insert_value(city, content, value):
        c.execute("UPDATE commenttable SET value = ? WHERE city==? AND content␣
 ↪== ?", (value,city,content))
        data = c.fetchall()
        return data

def count_total_comments(city):
        c.execute("SELECT COUNT(*) FROM commenttable WHERE city == ?", (city,))
        data = c.fetchall()
        return data

def count_positive_comments(city):
        c.execute("SELECT COUNT(*) FROM commenttable WHERE city == ? AND␣
 ↪value==1", (city,))
        data = c.fetchall()
        return data


#c.execute("DROP TABLE commenttable")



#CREACION DEL DATAFRAME DE USUARIOS
usersDataFrame = pd.DataFrame({"Username": "admin", "Password": "admin",␣
 ↪"Privileges":True},index=[0])

#Importamos el dataset y hacemos lo de cambiar las columnas a nombres buenos
```

```
dataset = pd.read_csv("Datasets2.csv")
dataset = dataset.rename(columns={'Unnamed: 0': 'City', 'Unnamed: 1' :␣
 ↪'Description', 'Unnamed: 2' : 'Country', 'Unnamed: 3' : 'Continent',␣
 ↪'Unnamed: 4' : 'LAT', 'Unnamed: 5' : 'LON'})
dataset = dataset.drop([0])
#Pasamos las columnas LAT y LON a números buenos en python (cambiamos las comas␣
 ↪',' por puntos '.')
for row in dataset.itertuples():
  #Sabiendo que cada row es una tupla y que el campo 5 y 6 es la LAT y la LON␣
 ↪respectivamente...
  lat = str(row[5])
  lon = str(row[6])
  lat = lat.replace(',','.')
  lon = lon.replace(',','.')

  dataset["LAT"].replace({row[5] : lat}, inplace=True)
  dataset["LON"].replace({row[6] : lon}, inplace=True)

#Creamos la matriz de las distancias para luego pasarlo a las vistas -> es el␣
 ↪mismo código que está explicado arriba paso a paso
ps = PorterStemmer()
preprocessedText = []
for row in dataset.itertuples():
    text = word_tokenize(row[2]) ## indice de la columna que contiene el texto
    ## Remove stop words
    stops = set(stopwords.words("english"))
    text = [ps.stem(w) for w in text if not w in stops and w.isalnum()]
    text = " ".join(text)
    preprocessedText.append(text)
preprocessedData = dataset
preprocessedData['processed_text_description'] = preprocessedText

bagOfWordsModel = TfidfVectorizer()
bagOfWordsModel.fit(preprocessedData['processed_text_description'])
textsBoW= bagOfWordsModel.
 ↪transform(preprocessedData['processed_text_description'])
distance_matrix = pairwise_distances(textsBoW,textsBoW ,metric='cosine')


#Dataset de comentarios para el análisis de sentimientos!

df_hoteles = pd.read_csv("hoteles.csv",encoding='unicode_escape')

df_hoteles["city_name"] = df_hoteles["city_name"].apply(lambda x : str(x).
 ↪strip())
```

```
df_hoteles["country_name"] = df_hoteles["country_name"].apply(lambda x : str(x).
 ↪strip())

def main():
        st.markdown("<h1 style='text-align: center; color: #1ec2f3; font-size:
 ↪80px;'>TraveLearning</h1>", unsafe_allow_html=True)
        listaCiudades = dataset['City'].tolist()

        menu = ["Home","SingUp","Login","Catalogo de ciudades","Recomendar␣
 ↪Ciudades","Filtrar Busqueda", "Añadir comentario a una ciudad","Ver␣
 ↪comentarios de una ciudad","About","Valoraciones Comentarios"]
        choice = st.sidebar.selectbox('Menu',menu)

        if choice == 'Home':
                image = Image.open('Logo.PNG')
                st.image(image)
                st.markdown("<h4 style='text-align: center; color: #b5b7b8;
 ↪'>Viajar es la oportunidad de que cada día sea diferente, de conocer␣
 ↪personas, historias, lugares, la posibilidad de conocerte a ti mismo a␣
 ↪través de los otros.</h4>", unsafe_allow_html=True)
                create_commenttable()

        if choice == "SingUp":
                st.subheader("Crea una nueva cuenta!")
                new_user = st.text_input("Username")
                new_password = st.text_input("Password", type='password')

                if st.button("Singup"):
                        create_usertable()
                        add_userdata(new_user,new_password)
                        st.success("You have successfully create an valid␣
 ↪Account")
                        st.info("Go to Login Menu to login")


        if choice == 'Login':
                st.subheader("Login Section")

                username = st.sidebar.text_input("User Name")
                password = st.sidebar.text_input("Password", type='password')

                if st.sidebar.checkbox("Login"):
                        create_usertable()
                        result = login_user(username,password)
                        if result:
                                st.success("Logged in as {}".format(username))
```

```
                                task = st.selectbox("Task",["Añadir␣
↪destino","Ver destino","Editar destino","Eliminar destino", "Ver todos los␣
↪usuarios"])

                                if task == "Añadir destino":
                                        st.caption("Entrando en añadir ciudad")
                                        st.subheader("Escriba el nombre del␣
↪destino")
                                        city_user = st.text_input('Ciudad')
                                        st.subheader("Escriba la descripción␣
↪del destino")
                                        description_user = st.
↪text_area('Descripción')
                                        st.subheader("Escriba el país del␣
↪destino")
                                        country_user = st.text_input('País')
                                        st.subheader("Escriba el continente del␣
↪destino")
                                        continent_user = st.
↪text_input('Continente')


                                        ps = PorterStemmer()

                                        preprocessedText = []

                                        text = word_tokenize(description_user)␣
↪## le pasamos el user_input
                                        ## Remove stop words
                                        stops = set(stopwords.words("english"))
                                        text = [ps.stem(w) for w in text if not␣
↪w in stops and w.isalnum()]
                                        text = " ".join(text)

                                        preprocessedText.append(text)

                                        user_row = {'City':city_user,␣
↪'Description': description_user, 'Country': country_user, 'Continent':
↪continent_user, 'processed_text_description':preprocessedText[0]}
                                        preprocessedData2 = preprocessedData.
↪append(user_row,ignore_index=True)

                                        preprocessedData2

                                        bagOfWordsModel = TfidfVectorizer()
                                        bagOfWordsModel.
↪fit(preprocessedData2['processed_text_description'])
```

```
                                        textsBoW= bagOfWordsModel.
↪transform(preprocessedData2['processed_text_description'])
                                        textsBoW.shape

                                        distance_matrix =␣
↪pairwise_distances(textsBoW,textsBoW ,metric='cosine')
                                        pd.DataFrame(distance_matrix)

                                if task == "Ver destino" :
                                        option = st.selectbox('Seleccione una␣
↪ciudad para mostrar los detalles',dataset['City'])
                                        preprocessedText = []
                                        preprocessedData2 = preprocessedData
                                        preprocessedData2[preprocessedData2['City']==option]
                                        lista =␣
↪list(preprocessedData2[preprocessedData2['City']==option].iloc[0])
                                        print("City: " + lista[0] + "\n"+
                                        "Description: " + lista[1] + "\n" +
                                        "Country: " + lista[2] + "\n" +
                                        "Continent: " + lista[3])

                                if task == "Editar destino" :
                                        option = st.selectbox('Seleccione una␣
↪ciudad para editarla',dataset['City'])

                                        st.caption("Entrando en añadir ciudad")
                                        st.subheader("Escriba el nombre del␣
↪destino")
                                        city_user = st.text_input('Ciudad')
                                        st.subheader("Escriba la descripción␣
↪del destino")
                                        description_user = st.
↪text_area('Descripción')
                                        st.subheader("Escriba el país del␣
↪destino")
                                        country_user = st.text_input('País')
                                        st.subheader("Escriba el continente del␣
↪destino")
                                        continent_user = st.
↪text_input('Continente')
                                        ps = PorterStemmer()

                                        preprocessedText = []

                                        text = word_tokenize(description_user)␣
↪## le pasamos el user_input
```

```
                                        ## Remove stop words
                                        stops = set(stopwords.words("english"))
                                        text = [ps.stem(w) for w in text if not␣
↪w in stops and w.isalnum()]
                                        text = " ".join(text)

                                        preprocessedText.append(text)

                                        #user_row = {'City':city_user,␣
↪'Description': description_user, 'Country': country_user, 'Continent':
↪continent_user, 'processed_text_description':preprocessedText[0]}
                                        preprocessedData2 = preprocessedData
                                        preprocessedData2.
↪at[preprocessedData2[preprocessedData2['City']==option].index,"City"] =␣
↪city_user
                                        preprocessedData2.
↪at[preprocessedData2[preprocessedData2['City']==option].index,"Description"]␣
↪= description_user
                                        preprocessedData2.
↪at[preprocessedData2[preprocessedData2['City']==option].index,"Country"] =␣
↪country_user
                                        preprocessedData2.
↪at[preprocessedData2[preprocessedData2['City']==option].index,"Continent"] =␣
↪continent_user
                                        preprocessedData2

                                        bagOfWordsModel = TfidfVectorizer()
                                        bagOfWordsModel.
↪fit(preprocessedData2['processed_text_description'])
                                        textsBoW= bagOfWordsModel.
↪transform(preprocessedData2['processed_text_description'])
                                        textsBoW.shape

                                        distance_matrix =␣
↪pairwise_distances(textsBoW,textsBoW ,metric='cosine')
                                        pd.DataFrame(distance_matrix)

                                if task == "Eliminar destino" :
                                        option = st.selectbox('Selecciona un␣
↪destino a eliminar', dataset['City'])
                                        ps = PorterStemmer()

                                        preprocessedText = []

                                        preprocessedData2 = preprocessedData
```

```
                                        preprocessedData2 = preprocessedData2.
↪drop(preprocessedData2[preprocessedData2['City']==option].index,axis=0)
                                        preprocessedData2

                                        bagOfWordsModel = TfidfVectorizer()

                                        bagOfWordsModel.
↪fit(preprocessedData2['processed_text_description'])
                                        textsBoW= bagOfWordsModel.
↪transform(preprocessedData2['processed_text_description'])
                                        textsBoW.shape

                                        distance_matrix =␣
↪pairwise_distances(textsBoW,textsBoW ,metric='cosine')
                                        pd.DataFrame(distance_matrix)

                                        st.write("Has eliminado a", option,".")

                                if task == "Ver todos los usuarios":
                                        st.caption("Estos son todos los␣
↪usuarios")
                                        user_result = view_all_users()
                                        clean_db = pd.DataFrame(user_result,␣
↪columns=["Username", "Password","Privileges"])
                                        st.dataframe(clean_db)
                        else:
                                st.warning("Incorrect Password")

##########################################################################################
        if choice == 'Catalogo de ciudades':
                st.header("Bienvenido al cátalogo de ciudades!")
                st.write("Aquí encontrarás todas las ciudades que hay en␣
↪nuestro sistema y podrás ver información de ellas")
                for row in dataset.itertuples():
                        with st.expander(row[1]):
                                description = str(dataset[dataset['City'] ==␣
↪row[1]].iloc[0]["Description"])
                                country = str(dataset[dataset['City'] ==␣
↪row[1]].iloc[0]["Country"])
                                continent = str(dataset[dataset['City'] ==␣
↪row[1]].iloc[0]["Continent"])
                                lat = float(dataset[dataset['City'] == row[1]].
↪iloc[0]["LAT"])
                                lon = float(dataset[dataset['City'] == row[1]].
↪iloc[0]["LON"])
```

```
                                total_comments =␣
↪int(count_total_comments(str(row[1]))[0][0]) #Tenemos que haces esto último␣
↪de [0][0] porque nos devulve una lista con dentro una tupla
                                positive_comments =␣
↪count_positive_comments(str(row[1]))[0][0]


                                if total_comments == None:
                                        total_comments=0
                                        porcentaje=0
                                elif total_comments == 0:
                                        total_comments=0
                                        porcentaje=0

                                else:
                                        porcentaje = float((positive_comments/
↪total_comments)*100)

                                if positive_comments == None:
                                        positive_comments=0

                                st.write("Description")
                                st.caption(description)

                                st.write("Country")
                                st.caption(country)

                                st.write("Continent")
                                st.caption(continent)

                                st.write("Geolocation")
                                df  = pd.DataFrame({"lat": lat, "lon":␣
↪lon},index=[0])
                                st.map(df)

                                st.write("Total Comments")
                                st.caption(str(total_comments))

                                st.write("Positive Comments")
                                st.caption(str(positive_comments))

                                if porcentaje != 0:
                                        st.write("Good comments percentage")
                                        st.caption(str(porcentaje) + "%")

                                #Hoteles
                                st.write("Hotels")
```

```
                                #Ordenamos el resultado por mejor valor de␣
↪rating
                                df_h =␣
↪df_hoteles[df_hoteles["city_name"]==row[1]].sort_values("Rating",␣
↪ascending=False)

                                if(len(df_h)!=0):
                                        st.dataframe(df_h)
                                else:
                                        st.caption("We don't have information␣
↪about hotels in this city")



                commentsTable = view_all_comments()
                clean_db = pd.DataFrame(commentsTable, columns=["Username",␣
↪"City","Content", "Value"])
                st.dataframe(clean_db)

#########################################################################################
        if choice == 'Recomendar Ciudades':
                #Vamos a hacer que el usuario seleccione una ciudad, y se␣
↪muestre las ciudades más parecidas
                option = st.selectbox('Seleccione una ciudad para mostrar las␣
↪más similares',dataset['City'])

                #Mostrar las ciudades más parecidas para dicha ciudad = option
                st.write('Has seleccionado:', option)

                #Ahora vamos a buscar las ciudades más parecidas para esa opcion
                searchCity = option #Ciudad para las recomendaciones
                indexOfCity =␣
↪preprocessedData[preprocessedData['City']==searchCity].index.values[0]
                bagOfWordsModel = TfidfVectorizer()
                bagOfWordsModel.
↪fit(preprocessedData['processed_text_description'])
                textsBoW= bagOfWordsModel.
↪transform(preprocessedData['processed_text_description'])
                distance_matrix = pairwise_distances(textsBoW,textsBoW␣
↪,metric='cosine')
                distance_scores =␣
↪list(enumerate(distance_matrix[indexOfCity-1]))
                pais = preprocessedData.iloc[indexOfCity-1]["Country"]

                print("Ciudades más parecidas:")
                l = list()
```

```
                for i in distance_scores:
                        t = tuple()
                        indice = i[0]
                        valor = i[1]
                        pais_aux = preprocessedData.iloc[i[0]]["Country"]
                        if pais == pais_aux:
                                valor = valor*0.92
                        t = (indice,valor)
                        l.append(t)

                ordered_scores = sorted(l, key=lambda x: x[1])
                top_scores = ordered_scores[1:11] #indices de las ciudades más␣
↪parecidas
                distint_scores = ordered_scores[-10:] #indices de las ciudades␣
↪más diferentes
                top_indexes = [i[0] for i in top_scores]
                distint_indexes = [i[0] for i in distint_scores]

                #Contruimos la lista de las ciudades más similares
                lista = list()
                tupla = tuple()
                listaSimilares = preprocessedData['City'].iloc[top_indexes].
↪tolist()

                for i in top_scores:
                        sublista = list()
                        for a in i:
                                sublista.append(a)

                        lista.append(sublista)

                for i in lista:
                        i.append(preprocessedData["City"].iloc[i[0]])

                #Contruimos la lista de las ciudades más diferentes
                listaR = list()
                listaDiferentes = preprocessedData['City'].
↪iloc[distint_indexes].tolist()

                for i in distint_scores:
                        sublista = list()
                        for a in i:
                                sublista.append(a)

                        listaR.append(sublista)

                for i in listaR:
```

```
                        i.append(preprocessedData["City"].iloc[i[0]])


                listaCiudadesParecidas =lista
                listaCiudadesDistintas = listaR

                st.header('Lista de las ciudades mas parecidas')
                container = st.container()

#                if st.button('Say hello'):
#                        st.write('Why hello there')
#                else:
#                        st.write('Goodbye')


                for i in listaCiudadesParecidas:
                        #container.button(i[2] + " -> Distancia: " +str(i[1]))
                        with st.expander(i[2]):
                                description = str(dataset[dataset['City'] ==␣
 ↪i[2]].iloc[0]["Description"])
                                country = str(dataset[dataset['City'] == i[2]].
 ↪iloc[0]["Country"])
                                continent = str(dataset[dataset['City'] ==␣
 ↪i[2]].iloc[0]["Continent"])
                                lat = float(dataset[dataset['City'] == i[2]].
 ↪iloc[0]["LAT"])
                                lon = float(dataset[dataset['City'] == i[2]].
 ↪iloc[0]["LON"])


                                st.write("Description")
                                st.caption(description)

                                st.write("Country")
                                st.caption(country)

                                st.write("Continent")
                                st.caption(continent)

                                st.write("Geolocalización")
                                df  = pd.DataFrame({"lat": lat, "lon":␣
 ↪lon},index=[0])
                                st.map(df)

                                st.write("Hotels")
```

```
                                df_h =␣
↪df_hoteles[df_hoteles["city_name"]==i[2]].sort_values("Rating",␣
↪ascending=False)

                                if(len(df_h)!=0):
                                        st.dataframe(df_h)
                                else:
                                        st.caption("We don't have information␣
↪about hotels in this city")



                st.header('Lista de las ciudades mas diferentes')

                for i in listaCiudadesDistintas:
                        with st.expander(i[2]):
                                description = str(dataset[dataset['City'] ==␣
↪i[2]].iloc[0]["Description"])
                                country = str(dataset[dataset['City'] == i[2]].
↪iloc[0]["Country"])
                                continent = str(dataset[dataset['City'] ==␣
↪i[2]].iloc[0]["Continent"])
                                lat = float(dataset[dataset['City'] == i[2]].
↪iloc[0]["LAT"])
                                lon = float(dataset[dataset['City'] == i[2]].
↪iloc[0]["LON"])

                                st.write("Description")
                                st.caption(description)

                                st.write("Country")
                                st.caption(country)

                                st.write("Continent")
                                st.caption(continent)

                                st.write("Geolocalización")
                                df  = pd.DataFrame({"lat": lat, "lon":␣
↪lon},index=[0])
                                st.map(df)

                                st.write("Hotels")

                                df_h =␣
↪df_hoteles[df_hoteles["city_name"]==i[2]].sort_values("Rating",␣
↪ascending=False)
```

```
                                if(len(df_h)!=0):
                                        st.dataframe(df_h)
                                else:
                                        st.caption("We don't have information␣
↪about hotels in this city")

#############################################################################

        if choice == "Filtrar Busqueda":

                listaContinentesDuplicados = list()
                listaContinentesDuplicados = list(dataset["Continent"])
                #Quitamos los valores duplicados
                listaContinentes = list()
                for item in listaContinentesDuplicados:
                        if item not in listaContinentes:
                                listaContinentes.append(item)

                continent = st.sidebar.selectbox("Continente",listaContinentes)

                listaPaisesDuplicados = list()
                listaPaisesDuplicados = list(dataset[dataset["Continent"] ==␣
↪continent]["Country"])
                #Quitamos los valores duplicados
                listaPaises = list()
                for item in listaPaisesDuplicados:
                        if item not in listaPaises:
                                listaPaises.append(item)

                country = st.sidebar.selectbox("Country",listaPaises)
                cercania = st.sidebar.select_slider(
                                'Select your proximity to the city you want to␣
↪go',
                                options=['Muy cerca', 'Cerca', 'Sin más',␣
↪'Lejos', 'Muy lejos'])

                #Muy cerca -> mismo pais
                #Cerca -> mismo continente
                #Sin Más -> da igual, no pondera especial
                #Lejos -> Otro continente
                #Muy lejos -> Otro continente

                countPaises = list(dataset[dataset["Country"] ==␣
↪country]["City"])
                for i in range(len(countPaises)):
                        with st.expander(countPaises[i]):
```

```
                                description = str(dataset[dataset['City'] ==␣
↪countPaises[i]].iloc[0]["Description"])
                                country = str(dataset[dataset['City'] ==␣
↪countPaises[i]].iloc[0]["Country"])
                                continent = str(dataset[dataset['City'] ==␣
↪countPaises[i]].iloc[0]["Continent"])
                                lat = float(dataset[dataset['City'] ==␣
↪countPaises[i]].iloc[0]["LAT"])
                                lon = float(dataset[dataset['City'] ==␣
↪countPaises[i]].iloc[0]["LON"])

                                total_comments =␣
↪int(count_total_comments(str(countPaises[i]))[0][0]) #Tenemos que haces esto␣
↪último de [0][0] porque nos devulve una lista con dentro una tupla
                                positive_comments =␣
↪count_positive_comments(str(countPaises[i]))[0][0]


                                if total_comments == None:
                                        total_comments=0
                                        porcentaje=0
                                elif total_comments == 0:
                                        total_comments=0
                                        porcentaje=0

                                else:
                                        porcentaje = float((positive_comments/
↪total_comments)*100)

                                if positive_comments == None:
                                        positive_comments=0

                                st.write("Description")
                                st.caption(description)

                                st.write("Country")
                                st.caption(country)

                                st.write("Continent")
                                st.caption(continent)

                                st.write("Geolocalización")
                                df  = pd.DataFrame({"lat": lat, "lon":␣
↪lon},index=[0])
                                st.map(df)
```

```
                                st.write("Total Comments")
                                st.caption(str(total_comments))

                                st.write("Positive Comments")
                                st.caption(str(positive_comments))

                                if porcentaje != 0:
                                        st.write("Good comments percentage")
                                        st.caption(str(porcentaje) + "%")

                                st.write("Hotels")
                                df_h =␣
↪df_hoteles[df_hoteles["city_name"]==countPaises[i]].sort_values("Rating",␣
↪ascending=False)

                                if(len(df_h)!=0):
                                        st.dataframe(df_h)
                                else:
                                        st.caption("We don't have information␣
↪about hotels in this city")



##########################################################################################

        if choice == "Añadir comentario a una ciudad":

                username = st.text_input("Cual es tu nombre?")
                #create_commenttable()

                username = username.strip() #Para quitar los espacios en blanco

                if username == "":
                        st.warning("Tienes que poner tu nombre para continuar")
                else:
                        addOption = st.radio(
                                "What do you want to add, valoration or comment?
↪",
                                ('Comment', 'Valoration'))

                        st.subheader("Which city would you like to add a␣
↪comment/valoration to")
                        cityOption = st.selectbox("Seleccione una␣
↪ciudad",dataset['City'])

                        if addOption == "Comment":
```

```
                                user_input = st.text_area("Añade aquí tu␣
↪comentario:")

                                st.write("Tu comentario para **_", cityOption,␣
↪"_**es: *", user_input, "*")

                                if user_input != "":
                                        #create_commenttable()
                                        add_commentdata(username,cityOption,user_input)
                                        #User_text = "This city is incredible!␣
↪There are many beautiful monuments that you have to see. Worth!"
                                        user_input=""

                        if addOption == "Valoration":
                                st.caption("") #Para añadir más margen, única␣
↪solución :(
                                col1, col2 = st.columns(2)

                                st.caption("") #Para añadir más margen, única␣
↪solución :(

                                with col1:
                                        valorationOption = st.select_slider(
                                                        "Did you like the city?
↪",
                                                        options=["Not really",␣
↪"More or less", "Very much!"])
                                with col2:
                                        st.caption("")

                                if st.button('Guardar'):
                                        if valorationOption == "Not really":
                                                valorationOption = -1
                                        elif valorationOption == "More or less":
                                                valorationOption = 0
                                        else:
                                                valorationOption = 1

                                        add_commentdata_value(username,cityOption,"",valoratio
                                        st.success("Your comment has been␣
↪saved")


                commentsTable = view_all_comments()
                clean_db = pd.DataFrame(commentsTable, columns=["Username",␣
↪"City","Content", "Value"])
```

```
                st.dataframe(clean_db)

#########################################################################################
        if choice == "Ver comentarios de una ciudad":

                st.header("De que ciudad quieres ver los comentarios asociados?
↪")
                cityOption = st.selectbox("Seleccione una␣
↪ciudad",dataset['City'])

                commentsTable = view_all_comments_city(cityOption)
                st.write(len(commentsTable))

                if(len(commentsTable)==0):
                        st.error("No hay comentarios asociadas a dicha ciudad!")
                else:
                        clean_db = pd.DataFrame(commentsTable,␣
↪columns=["Username", "City","Content","Value"])
                        st.dataframe(clean_db)


        if choice == "About":
                st.caption("Este proyecto fue diseñado e implementado por: ...")


        if choice == "Valoraciones Comentarios":
                st.caption("Esta página sirve para que el sistema cargue el␣
↪dataset de comentarios y asigne una valoración a cada uno de si es bueno,␣
↪regular o malo")

                trainingData = pd.read_csv('semeval-2017-train.csv',␣
↪delimiter='        ')
                trainingData = trainingData.head(15000) #Eliminar la funcion␣
↪head() si se quiere usar todo el dataset. Para las pruebas usamos únicamente␣
↪los 1000 primeros tweets
                nltk.download('punkt')
                nltk.download('stopwords')

                ps = PorterStemmer()

                preprocessedText = []

                for row in trainingData.itertuples():

                        text = word_tokenize(row[2]) ## indice de la columna␣
↪que contiene el texto
```

```
                        ## Remove stop words
                        stops = set(stopwords.words("english"))
                        text = [ps.stem(w) for w in text if not w in stops and␣
↪w.isalnum()]
                        text = " ".join(text)

                        preprocessedText.append(text)

                preprocessedData2 = trainingData
                preprocessedData2['processed_text'] = preprocessedText

                bagOfWordsModel2 = TfidfVectorizer()
                bagOfWordsModel2.fit(preprocessedData2['processed_text'])
                textsBoW= bagOfWordsModel2.
↪transform(preprocessedData2['processed_text'])

                svc = svm.SVC(kernel='linear') #Modelo de clasificación

                X_train = textsBoW #Documentos
                Y_train = trainingData['label'] #Etiquetas de los documentos
                svc.fit(X_train, Y_train) #Entrenamiento

                ###############
                ps = PorterStemmer()

                preprocessedText = []

                commentsTable = view_all_comments_value2()
                clean_db = pd.DataFrame(commentsTable, columns=["Username",␣
↪"City","Content", "Value"])
                st.dataframe(clean_db)

                #user_text = "This city is incredible! There are many beautiful␣
↪monuments that you have to see. Worth!"

                for row in clean_db.itertuples():

                        text = word_tokenize(row[3]) ## indice de la columna␣
↪que contiene el texto
                        ## Remove stop words
                        stops = set(stopwords.words("english"))
                        text = [ps.stem(w) for w in text if not w in stops and␣
↪w.isalnum()]
                        text = " ".join(text)

                        preprocessedText.append(text)
```

```
                preprocessedData2 = clean_db
                preprocessedData2['processed_text'] = preprocessedText

                #st.write(preprocessedText)

                textsBoWTest= bagOfWordsModel2.
 ↪transform(preprocessedData2['processed_text'])
                X_test = textsBoWTest #Documentos

                predictions = svc.predict(X_test) #Se almacena en el array␣
 ↪predictions las predicciones del clasificador
                predictions = list(predictions)

                contador = 0

                for row in clean_db.itertuples():
                        city = row[2]
                        content = row[3]
                        value = int(predictions[contador])

                        insert_value(city, content, value)
                        contador = contador +1

                #add_commentdata("Prueba","Ourense","asdiasdiaiajsdiasdiaisdjiasdijasdijasjasi
                commentsTable = view_all_comments()
                clean_db = pd.DataFrame(commentsTable, columns=["Username",␣
 ↪"City","Content", "Value"])
                st.dataframe(clean_db)







if __name__ == '__main__':
        main()

def getUserDataFrame():
        return usersDataFrame
```

```
Writing TraveLearning.py
```

11.3 Lanzar Aplicación TraveLearning.py

Se ejecuta nuestra aplicación en segundo plano y se crea un NgrokTunnel en el puerto 8501

[69]:
```
!streamlit run TraveLearning.py &>/dev/null &
```

[65]:
```
from pyngrok import ngrok
```

[70]:
```
# Setup a tunnel to the streamlit port 8501
ngrok.set_auth_token("22EF8Enh10cKxuEhAZyKrPkOF0y_3NqBbv2dgw5skYRnQzWM")
public_url = ngrok.connect(8501)
public_url
```

[70]:
```
<NgrokTunnel: "http://e574-34-125-97-182.ngrok.io" -> "http://localhost:8501">
```

11.4 Deterner nuestra aplicación

Dejamos aquí el codigo por si se quisiera cerrar la aplicación

[67]:
```
ngrok.kill()
```

12 Exportación del código

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('ABP.ipynb')
```

```
File colab_pdf.py already there; not retrieving.

Mounted at /content/drive/

WARNING: apt does not have a stable CLI interface. Use with caution in scripts.


WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```