Apellidos y nombres: Ramos Ccari Jean Branco (190984)

LABORATORIO: Manipulación de Word embbeding

En este laboratorio, aplicará operaciones de álgebra lineal usando NumPy para encontrar relaciones entre palabras para lo cual se tiene como objetivo:

- Analizar el código
- Escribir conclusiones del Laboratorio de Manipulación de word embbeding

1. Manipulación de word embeddings

Los **word embeddings** son representaciones de palabras con vectores.

In [3]:
```
import pandas as pd
import numpy as np
import pickle

word_embeddings = pickle.load( open( "word_embeddings_subset.p", "rb" ) )
len(word_embeddings) #  243 words
```

Out[3]:
```
243
```

Ahora que el modelo está cargado, podemos echar un vistazo a las representaciones de palabras. Primero, tenga en cuenta que **word_embeddings** es un diccionario. Cada palabra es la clave de la entrada, y el valor es su correspondiente vector de presentación.

Recuerde que los corchetes permiten el acceso a cualquier entrada si existe la clave.

In [5]:
```
# Obtiene la representación vectorial de la palabra 'country'
countryVector = word_embeddings['Paris']
# Imprime el tipo del vector. Tenga en cuenta que es una matriz numpy
print(type(countryVector))
# Imprime los valores del vector.
print(len(countryVector))
```

```
<class 'numpy.ndarray'>
300
```

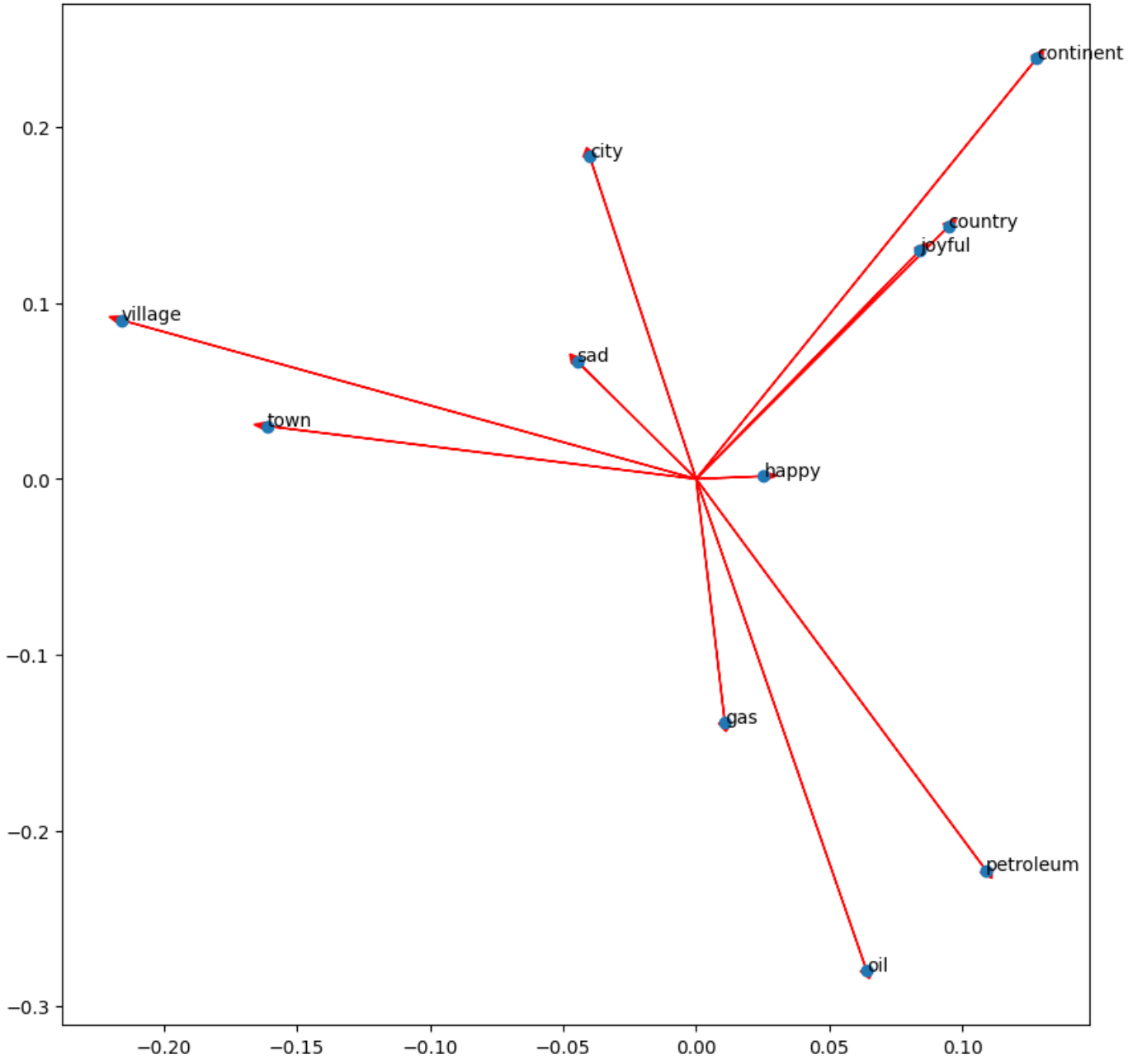
Es importante tener en cuenta que almacenamos cada vector como una arrayNumPy. Nos permite usar las operaciones de álgebra lineal en él.

Los vectores tienen un tamaño de 300, mientras que el tamaño del vocabulario de Google News es de alrededor de 3 millones de palabras.

In [6]:
```
# Función que obtiene el vector para una palabra dada:
def vec(w):
    return word_embeddings[w]
```

In [7]:

```
import matplotlib.pyplot as plt

words = ['oil', 'gas', 'happy', 'sad', 'city', 'town', 'village', 'country', 'continent'

# Convierte cada palabra a su representación vectorial
bag2d = np.array([vec(word) for word in words])

# Crea imagen de tamaño personalizado
fig, ax = plt.subplots(figsize = (10, 10))

col1 = 3 # Seleccione la columna para el eje x
col2 = 2 # Seleccione la columna para el eje y

# Imprime una flecha para cada palabra
for word in bag2d:
    ax.arrow(0, 0, word[col1], word[col2], head_width=0.005, head_length=0.005, fc='r',

# Plot un punto para cada palabra
ax.scatter(bag2d[:, col1], bag2d[:, col2]);

# Agrega la  etiqueta sobre cada punto en el diagrama de dispersión
for i in range(0, len(words)):
    ax.annotate(words[i], (bag2d[i, col1], bag2d[i, col2]))


plt.show()
```



Operando en word embeddings

Recuerde que comprender los datos es uno de los pasos más críticos en Data Science. Las word embeddingsson el resultado de procesos de aprendizaje automático y serán parte de la entrada para procesos posteriores. Este word embeddings debe validarse o al menos entenderse porque el rendimiento del modelo derivado dependerá en gran medida de su calidad.

Los word embeddings son arrays multidimensionales, generalmente con cientos de atributos que plantean un desafío para su interpretación.

En este Laboratorio, inspeccionaremos visualmente word embeddings de algunas palabras usando un par de atributos. Los atributos sin procesar no son la mejor opción para la creación de dichos gráficos, pero nos permitirán ilustrar la parte mecánica en Python.

En la siguiente celda, hacemos un gráfico para las word embeddings de algunas palabras. Incluso si trazar los puntos da una idea de las palabras, las representaciones de flechas también ayudan a visualizar la alineación del vector.

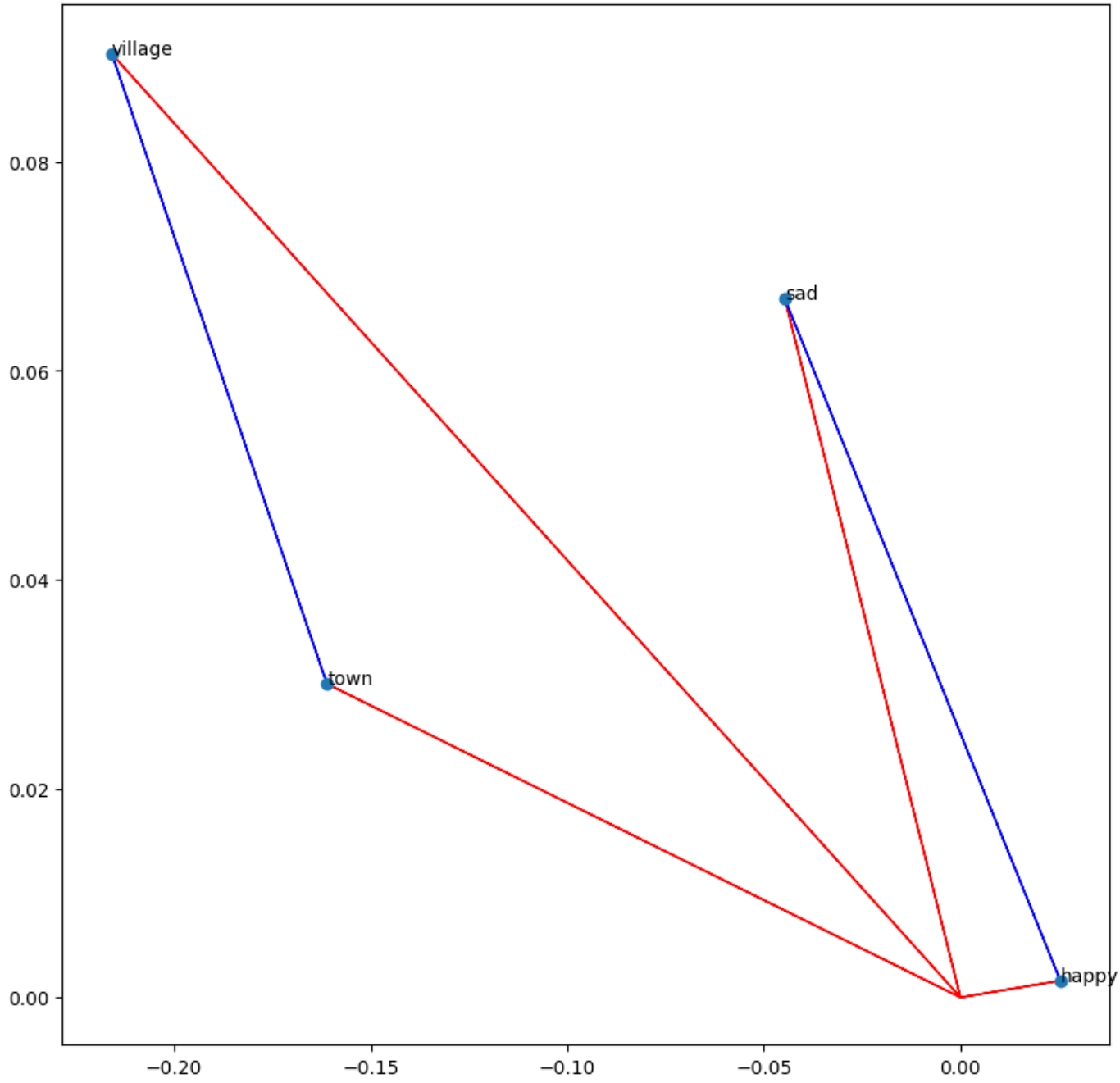
Tenga en cuenta que palabras similares como "village" y "town" o "petroleum", "oil" y "gas" tienden a apuntar en la misma dirección. Además, tenga en cuenta que 'sad' y 'happy' se parecen mucho; sin embargo, los vectores apuntan en direcciones opuestas.

En este cuadro, uno puede calcular los ángulos y las distancias entre las palabras. Algunas palabras están cerca en ambos tipos de métricas de distancia.

Distancia entre palabras

Ahora escribe las palabras 'sad', 'happy', 'town' y 'village'. En este mismo gráfico, muestra el vector de 'village' a 'town' y el vector de 'sad' a 'happy'. Usemos NumPy para estas operaciones de álgebra lineal.

In [50]:
```
words = ['sad', 'happy', 'town', 'village']

# Convierte cada palabra a su representación vectorial
bag2d = np.array([vec(word) for word in words])

fig, ax = plt.subplots(figsize = (10, 10)) # Create custom size image

col1 = 3 # Select the column for the x axe
col2 = 2 # Select the column for the y axe

# Imprime un flecha para cada palabra
for word in bag2d:
    ax.arrow(0, 0, word[col1], word[col2], head_width=0.0005, head_length=0.0005, fc='r'

# Imprime el vector diferencia entre village y town
village = vec('village')
town = vec('town')
diff = town - village
ax.arrow(village[col1], village[col2], diff[col1], diff[col2], fc='b', ec='b', width = 1

# Imprime el vector diferencia entre sad y happy
sad = vec('sad')
happy = vec('happy')
diff = happy - sad
ax.arrow(sad[col1], sad[col2], diff[col1], diff[col2], fc='b', ec='b', width = 1e-5)


ax.scatter(bag2d[:, col1], bag2d[:, col2]); # Plot a dot for each word

# Agregue la  etiqueta sobre cada punto en el diagrama
for i in range(0, len(words)):
    ax.annotate(words[i], (bag2d[i, col1], bag2d[i, col2]))


plt.show()
```



Algebra Lineal en word embeddings

En clases se vio las relaciones entre las palabras usando álgebra lineal en word embeddings. Veamos cómo hacerlo en Python con Numpy. Para empezar, obtenga la **norma** de una palabra en word embedding.

In [9]:
```
# Imprime la norma de la palabra town
print(np.linalg.norm(vec('town')))
# Imprime la norma de la palabra sad
print(np.linalg.norm(vec('sad')))
```

```
2.3858097
2.9004838
```

1. Predicción de capitales

Ahora, aplicando la diferencia y la suma de vectores, se puede crear una representación vectorial para una nueva palabra. Por ejemplo, podemos decir que el vector diferencia entre **France** y **Paris** representa el

concepto de **Capital**.

Uno puede moverse desde la ciudad de Madrid en la dirección del concepto de Capital, y obtener algo cercano al país correspondiente al cual Madrid es la Capital.

In [48]:
```
capital = vec('France') - vec('Paris')
country = vec('Madrid') + capital

print(country[0:5]) # Imprime los primeros 5 valores del vector
```

```
[-0.02905273 -0.2475586   0.53952026  0.20581055 -0.14862823]
```

Podemos observar que el vector **country** que esperábamos que fuera el mismo que el vector de **Spain** no lo es exactamente.

In [47]:
```
diff = country - vec('Spain')
print(diff[0:5])
```

```
[-0.15966797  0.13757324  0.32174683 -0.1965332  -0.09399414]
```

Entonces, tenemos que buscar las palabras más cercanas en el word embbeding que coincidan con el país candidato. Si el word embbeding funciona como se esperaba, la palabra más similar debe ser 'Spain'. Definamos una función que nos ayude a hacerlo. Almacenaremos nuestro word embedding como un DataFrame, que facilita las operaciones de búsqueda basadas en los vectores numéricos.

In [12]:
```
# Crea u dataframe a partir del diccionario embedding.
# Esto facilita las operaciones algebraicas.
keys = word_embeddings.keys()
data = []
for key in keys:
    data.append(word_embeddings[key])

embedding = pd.DataFrame(data=data, index=keys)

# Defina una función para encontrar la palabra más cercana a un vector:
def find_closest_word(v, k = 1):
    # Calcula el vector diferencia de cada palabra al vector de entrada
    diff = embedding.values - v
    # Obtenga la norma de cada vector de diferencia.
    # Significa la distancia euclidiana al cuadrado de cada palabra al vector de entrada
    delta = np.sum(diff * diff, axis=1)
    # Encuentre el índice de la distancia mínima en la matriz
    i = np.argmin(delta)
    # Devuelve el nombre de la fila para este elemento
    return embedding.iloc[i].name
```

In [36]:
```
# Imprime algunas filas de embedding como un Dataframe
embedding.head(60)
```

Out[36]:

	0	1	2	3	4	5	6	7	8	
country	-0.080078	0.133789	0.143555	0.094727	-0.047363	-0.023560	-0.008545	-0.186523	0.045898	-0.0
city	-0.010071	0.057373	0.183594	-0.040039	-0.029785	-0.079102	0.071777	0.013306	-0.143555	0.0
China	-0.073242	0.135742	0.108887	0.083008	-0.127930	-0.227539	0.151367	-0.045654	-0.065430	0.0
Iraq	0.191406	0.125000	-0.065430	0.060059	-0.285156	-0.102539	0.117188	-0.351562	-0.095215	0.2
oil	-0.139648	0.062256	-0.279297	0.063965	0.044434	-0.154297	-0.184570	-0.498047	0.047363	0.1
town	0.123535	0.159180	0.030029	-0.161133	0.015625	0.111816	0.039795	-0.196289	-0.039307	0.0

Canada	-0.136719	-0.154297	0.269531	0.273438	0.086914	-0.076172	-0.018677	0.006256	0.077637	-0.2
London	-0.267578	0.092773	-0.238281	0.115234	-0.006836	0.221680	-0.251953	-0.055420	0.020020	0.1
England	-0.198242	0.115234	0.062500	-0.058350	0.226562	0.045898	-0.062256	-0.202148	0.080566	0.0
Australia	0.048828	-0.194336	-0.041504	0.084473	-0.114258	-0.208008	-0.164062	-0.269531	0.079102	0.2
Japan	0.050781	0.250000	0.166992	0.084473	-0.265625	-0.077148	-0.039551	-0.298828	0.080566	0.0
Pakistan	-0.308594	0.029175	0.185547	0.177734	-0.343750	0.125977	-0.078613	-0.121094	-0.205078	-0.0
Iran	-0.126953	0.165039	0.213867	0.324219	0.044678	-0.036865	-0.155273	-0.167969	-0.228516	0.1
gas	-0.161133	0.105469	-0.138672	0.010803	0.017090	-0.041504	-0.145508	-0.341797	-0.028564	0.1
happy	-0.000519	0.160156	0.001610	0.025391	0.099121	-0.085938	0.324219	-0.021729	0.134766	0.1
Russia	-0.077148	-0.008606	0.122559	0.192383	-0.039062	0.052246	0.136719	-0.232422	-0.119141	0.1
Afghanistan	-0.057373	0.038330	0.026001	0.206055	-0.207031	0.036133	0.091309	-0.218750	-0.117188	-0.0
France	0.048584	0.078613	0.324219	0.034912	0.077148	0.035400	-0.125977	-0.386719	-0.131836	0.0
Germany	0.259766	0.140625	0.247070	0.009583	-0.250000	-0.082520	-0.099121	-0.353516	-0.148438	0.1
Georgia	0.125977	-0.247070	0.006409	0.273438	0.095703	0.078613	-0.169922	-0.139648	-0.213867	0.0
Baghdad	-0.046143	0.054932	-0.168945	0.158203	-0.142578	-0.094727	-0.163086	-0.527344	-0.149414	0.4
village	0.147461	0.205078	0.090332	-0.215820	0.027954	0.198242	0.101562	-0.058105	-0.029663	0.0
Spain	0.031494	-0.182617	0.163086	0.124512	-0.018555	-0.173828	-0.121094	-0.132812	-0.125000	0.2
Italy	0.057861	0.136719	0.169922	0.072754	-0.151367	-0.001472	-0.151367	-0.312500	-0.071289	0.2
Beijing	-0.132812	0.193359	0.136719	0.187500	-0.060791	-0.170898	-0.022461	0.083984	-0.175781	0.2
Jordan	0.253906	0.084961	-0.120117	-0.116699	0.053467	-0.105469	-0.029297	-0.322266	-0.012634	0.0
Paris	-0.018555	0.131836	-0.006317	0.198242	0.226562	0.099609	-0.291016	-0.283203	-0.152344	0.1
Ireland	0.202148	0.085449	0.019897	0.250000	0.058594	-0.164062	-0.113281	-0.123535	-0.002045	-0.0
Turkey	0.208008	-0.154297	-0.007050	0.067383	-0.152344	0.022827	-0.048828	-0.296875	-0.186523	0.1
Egypt	0.219727	0.208008	-0.136719	-0.062988	-0.036377	-0.040527	0.046875	-0.190430	-0.161133	0.0
Lebanon	0.131836	0.002686	-0.127930	-0.134766	-0.006744	-0.126953	0.007782	-0.263672	-0.182617	0.3
Taiwan	-0.034424	-0.068848	0.087402	0.094238	-0.215820	-0.201172	-0.025391	-0.046387	-0.092285	0.0
Tokyo	-0.109375	0.271484	-0.007874	0.146484	-0.201172	0.063965	-0.228516	-0.074219	0.021851	0.2
Nigeria	0.094238	-0.068359	-0.067383	0.045898	-0.156250	-0.150391	-0.175781	-0.386719	-0.271484	-0.0
Vietnam	0.214844	0.221680	0.007172	0.177734	-0.144531	-0.152344	0.345703	-0.251953	0.072266	0.1
Moscow	-0.133789	-0.023438	0.009399	0.433594	0.068848	0.149414	0.039307	0.021240	-0.215820	0.2
Greece	0.472656	0.007019	0.059082	0.090332	0.039551	-0.058594	0.060059	-0.257812	-0.241211	-0.0
Indonesia	0.251953	-0.244141	-0.075684	0.298828	-0.157227	-0.107422	0.032471	-0.312500	-0.095215	0.2
sad	0.189453	0.045898	0.066895	-0.044678	0.178711	0.019653	0.347656	-0.024658	0.406250	0.1
Syria	0.127930	0.105469	0.061523	0.005341	-0.104004	-0.019165	-0.147461	-0.124512	-0.157227	0.2
Thailand	0.109863	-0.117188	0.010010	0.238281	-0.193359	0.060303	0.059326	-0.402344	-0.139648	0.1
Libya	-0.065430	0.131836	0.096680	0.205078	-0.464844	0.017578	-0.200195	-0.308594	0.017090	-0.0
Zimbabwe	-0.055420	-0.004883	-0.000984	-0.038818	-0.237305	-0.010132	-0.017822	-0.316406	-0.289062	-0.6
Cuba	-0.125000	-0.035156	0.267578	0.219727	-0.324219	-0.056885	-0.291016	0.059814	-0.044189	-0.1

Ottawa	-0.225586	-0.056641	0.160156	0.562500	0.000252	-0.033936	-0.099121	0.086914	-0.088379	0.0
Tehran	-0.253906	0.109375	0.257812	0.500000	0.150391	-0.069824	-0.316406	-0.168945	-0.298828	0.3
Sudan	0.072754	0.416016	0.058594	-0.032959	-0.117676	-0.099121	-0.002731	-0.349609	-0.200195	-0.0
Kenya	0.089355	0.170898	0.025269	0.048096	-0.105957	-0.227539	-0.028931	-0.267578	-0.519531	-0.1
Philippines	0.116699	-0.132812	0.107422	0.146484	-0.079590	-0.154297	-0.041992	-0.166992	-0.047119	0.1
Sweden	0.099121	-0.069336	0.245117	0.044434	-0.197266	-0.071289	0.014465	-0.298828	0.061279	-0.0
Poland	0.166992	0.033203	0.279297	0.140625	-0.187500	-0.051025	0.123535	-0.102539	-0.067871	-0.0
Ukraine	-0.026367	-0.062012	0.069336	0.269531	0.031006	0.122070	0.111816	-0.170898	-0.209961	-0.0
Rome	0.235352	0.186523	-0.039062	0.314453	-0.010193	0.093750	-0.320312	-0.016357	-0.063477	0.2
Venezuela	-0.207031	-0.011780	0.138672	0.236328	-0.193359	-0.031006	-0.162109	-0.335938	-0.052002	-0.1
Switzerland	0.042236	0.061035	0.273438	0.133789	-0.152344	0.095215	-0.023926	-0.433594	0.062500	-0.0
Berlin	0.141602	0.251953	0.026245	0.000698	-0.098145	-0.154297	-0.206055	-0.148438	-0.164062	0.0
Bangladesh	-0.227539	0.087891	0.003860	0.122070	-0.137695	-0.034424	-0.071289	-0.236328	-0.298828	-0.0
Portugal	0.106934	-0.048096	0.148438	0.158203	-0.030884	-0.202148	-0.127930	-0.378906	-0.170898	-0.1
Ghana	0.085449	-0.027710	0.152344	0.034180	-0.132812	-0.367188	-0.100098	-0.304688	-0.220703	-0.3
Athens	0.265625	-0.062500	0.004486	0.103516	0.289062	-0.093262	-0.101074	0.076172	-0.279297	0.4

60 rows × 300 columns

Ahora busquemos el nombre que corresponde a nuestro país numérico:

In [14]:
```
find_closest_word(country)
```

Out[14]:
```
'Spain'
```

2. Predicción de otros países

In [40]:
```
find_closest_word(vec('Italy') - vec('Rome') + vec('Madrid'))
```

Out[40]:
```
'Spain'
```

In [23]:
```
print(find_closest_word(vec('Berlin') + capital))
print(find_closest_word(vec('Beijing') + capital))
```

```
Germany
China
```

Sin embargo, no siempre funciona.

In [38]:
```
print(find_closest_word(vec('Peru') + capital))
```

```
Peru
```

Representar una oración como un vector

Una oración completa se puede representar como un vector sumando todos los vectores de palabras que conforman la oración.

In [44]:
```
doc = "Spain petroleum city king"
vdoc = [vec(x) for x in doc.split(" ")]
doc2vec = np.sum(vdoc, axis = 0)
doc2vec
```

Out[44]:
```
array([ 2.87475586e-02,  1.03759766e-01,  1.32629395e-01,  3.33007812e-01,
       -2.61230469e-02, -5.95703125e-01, -1.25976562e-01, -1.01306152e+00,
       -2.18544006e-01,  6.60705566e-01, -2.58300781e-01, -2.09960938e-02,
       -7.71484375e-02, -3.07128906e-01, -5.94726562e-01,  2.00561523e-01,
       -1.04980469e-02, -1.10748291e-01,  4.82177734e-02,  6.38977051e-01,
        2.36083984e-01, -2.69775391e-01,  3.90625000e-02,  4.16503906e-01,
        2.83416748e-01, -7.25097656e-02, -3.12988281e-01,  1.05712891e-01,
        3.22265625e-02,  2.38403320e-01,  3.88183594e-01, -7.51953125e-02,
       -1.26281738e-01,  6.60644531e-01, -7.89794922e-01, -7.04345703e-02,
       -1.14379883e-01, -4.78515625e-02,  4.76318359e-01,  5.31127930e-01,
        8.10546875e-02, -1.17553711e-01,  1.02050781e+00,  5.59814453e-01,
       -1.17187500e-01,  1.21826172e-01, -5.51574707e-01,  1.44531250e-01,
       -7.66113281e-01,  5.36102295e-01, -2.80029297e-01,  3.85986328e-01,
       -2.39135742e-01, -2.86865234e-02, -5.10498047e-01,  2.59658813e-01,
       -7.52929688e-01,  4.32128906e-02, -7.17773438e-02, -1.26708984e-01,
        4.40673828e-02,  5.12939453e-01, -5.15808105e-01,  1.20117188e-01,
       -5.52978516e-02, -3.92089844e-01, -3.15917969e-01,  1.57226562e-01,
       -3.19702148e-01,  1.75170898e-01, -3.81835938e-01, -2.07031250e-01,
       -4.72717285e-02, -2.79296875e-01, -3.29040527e-01, -1.69067383e-01,
        1.61132812e-02,  1.71569824e-01,  5.73730469e-02, -2.44140625e-03,
        8.34960938e-02, -1.58203125e-01, -3.10119629e-01,  5.28564453e-02,
        8.60595703e-02,  5.12695312e-02, -7.22900391e-01,  4.97924805e-01,
       -5.85937500e-03,  4.49951172e-01,  3.82446289e-01, -2.80029297e-01,
       -3.28125000e-01, -6.27441406e-02, -4.81933594e-01,  1.93176270e-02,
       -1.69326782e-01, -4.28649902e-01,  5.39062500e-01, -1.28417969e-01,
       -8.83789062e-02,  5.13916016e-01,  9.13085938e-02, -1.60156250e-01,
        6.86035156e-02, -9.74121094e-02, -3.70712280e-01, -3.27270508e-01,
        1.77978516e-01, -4.65332031e-01,  1.70410156e-01,  9.08203125e-02,
        2.76857376e-01, -1.69677734e-01,  3.27728271e-01, -3.12500000e-02,
       -2.20809937e-01, -3.46679688e-01,  4.67407227e-01,  5.31860352e-01,
       -1.30615234e-01, -2.36816406e-02, -6.56250000e-01, -5.79589844e-01,
       -2.05810547e-01, -3.03222656e-01,  1.94259644e-01, -7.28515625e-01,
       -4.92522240e-01, -5.37109375e-01, -3.47656250e-01,  1.08642578e-01,
       -1.41601562e-01, -2.07031250e-01,  2.52441406e-01, -7.78808594e-02,
       -5.02441406e-01,  1.53808594e-02,  8.64257812e-02,  2.59765625e-01,
        6.64062500e-02, -7.12890625e-01, -1.45751953e-01,  7.56835938e-03,
        4.87792969e-01,  1.39160156e-01,  1.15722656e-01,  1.28662109e-01,
       -4.75585938e-01,  2.21191406e-01,  3.25317383e-01,  1.06323242e-01,
       -6.11083984e-01, -3.59619141e-01,  6.54296875e-02, -2.41699219e-01,
       -6.29882812e-02, -1.62109375e-01,  4.26269531e-01, -4.38354492e-01,
        1.93725586e-01,  4.89562988e-01,  5.31494141e-01, -7.29370117e-02,
        1.77246094e-01,  9.39941406e-02,  2.92236328e-01, -2.74047852e-01,
        2.63366699e-02,  4.36035156e-01, -3.76953125e-01,  3.10546875e-01,
        4.87304688e-01, -2.43041992e-01,  1.21612549e-02, -3.80371094e-01,
        3.80493164e-01, -6.22436523e-01, -3.98071289e-01,  1.24206543e-01,
       -8.20312500e-01, -2.72583008e-01, -6.21582031e-01, -4.87060547e-01,
        3.06671143e-01, -2.61230469e-01,  5.12451172e-01,  5.55694580e-01,
        5.66894531e-01,  7.33886719e-01, -1.75781250e-01,  4.13574219e-01,
       -2.54272461e-01,  1.32507324e-01, -4.78515625e-01,  4.63256836e-01,
       -6.21948242e-02, -1.80664062e-01, -5.46386719e-01, -6.31103516e-01,
       -1.47949219e-01, -3.15185547e-01, -7.12890625e-02, -7.67578125e-01,
        3.92272949e-01, -1.97753906e-01,  2.23144531e-01, -5.07324219e-01,
        8.39843750e-02, -4.98657227e-02,  1.01074219e-01,  2.07885742e-01,
       -2.77343750e-01,  1.03027344e-01, -1.38671875e-01,  2.87353516e-01,
       -4.81895447e-01, -1.66748047e-01, -1.47277832e-01,  3.61633301e-01,
        6.38504028e-02, -6.69189453e-01,  1.95312500e-03, -7.34375000e-01,
       -1.28158569e-01,  9.76562500e-04, -7.08007812e-02,  3.72558594e-01,
```

```
       8.31176758e-01,  5.94482422e-01,  5.37109375e-02, -3.00140381e-01,
      -4.53857422e-01,  1.11511230e-01, -1.32812500e-01,  1.25732422e-01,
       3.39843750e-01, -2.48352051e-01, -1.62353516e-02, -2.84667969e-01,
       4.70703125e-01, -4.48242188e-01,  8.50753784e-02,  2.69042969e-01,
       3.98254395e-03, -3.53759766e-01, -3.90625000e-02, -3.22753906e-01,
      -6.90917969e-02, -4.13818359e-02,  1.35314941e-01, -8.50396156e-02,
       1.28417969e-01,  6.15966797e-01,  3.55957031e-01, -6.05468750e-02,
      -2.25463867e-01, -2.62207031e-01, -2.72949219e-01, -5.16113281e-01,
       1.59179688e-01,  2.74902344e-01, -7.61718750e-02, -3.41796875e-03,
       4.37500000e-01,  2.98583984e-01, -4.40795898e-01, -3.43261719e-01,
       1.73583984e-01,  3.32092285e-01, -2.12646484e-01,  5.76171875e-01,
       2.06787109e-01, -7.91015625e-02,  5.79695702e-02, -1.01806641e-01,
      -7.06787109e-01, -3.40576172e-02, -4.11865234e-01,  9.82666016e-02,
      -1.70410156e-01, -4.18212891e-01,  8.39233398e-01, -1.15722656e-01,
       1.28173828e-01, -2.07763672e-01, -4.08203125e-01, -1.77612305e-01,
       1.01196289e-01,  4.24072266e-01, -5.26428223e-02, -5.58593750e-01,
       1.12304688e-02, -1.12060547e-01, -9.42382812e-02,  2.35595703e-02,
      -3.92578125e-01, -7.12890625e-02,  5.69824219e-01,  9.81445312e-02],
     dtype=float32)
```

In [19]:
```
find_closest_word(doc2vec)
```

Out[19]:
```
'petroleum'
```

ACTIVIDAD

A partir del desarrollo del laboratorio escriba 4 conclusiones del Laboratorio

- Los word embeddings son capaces de capturar el significado semántico de las palabras. A través de esta representación vectorial, las palabras con significados similares están agrupadas en áreas cercanas del espacio vectorial.
- Los word embeddings también tienen limitaciones, ya que no son capaces de capturar completamente el contexto y las connotaciones emocionales de las palabras, lo que puede ser un obstáculo en algunas aplicaciones de procesamiento del lenguaje natural.
- Creo que el uso de más datos puede mejorar la calidad de los word embeddings, pero es importante considerar cuidadosamente la cantidad, calidad y relevancia de los datos utilizados en el entrenamiento del modelo.
- Los word embeddings son muy útiles para una variedad de tareas de procesamiento del lenguaje natural, como la traducción automática, la clasificación de texto, la generación de texto y la identificación de relaciones semánticas o significado entre palabras.

In [ ]:
```
```