Sprint 4: S04 T01: Transformació Registre Log amb Regular expressions

Nivell 1

Exercici 1

Estandaritza, identifica i enumera cada un dels atributs / variables de l'estructura de l'arxiu "Web_access_log-akumenius.com" que trobaràs al repositori de GitHub "Data-sources".

In [4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
import requests
import json
import warnings

warnings.filterwarnings('ignore')
```

1.1 Lectura fichero .txt

In [5]:
```
data = pd.read_fwf(r"C:\Users\hecto\OneDrive\Documentos\IT Data Science\Sprint4\Spri
data
```

Out[5]:

	0
0	localhost 127.0.0.1 - - [23/Feb/2014:03:10:31 ...
1	localhost 127.0.0.1 - - [23/Feb/2014:03:10:31 ...
2	localhost 127.0.0.1 - - [23/Feb/2014:03:10:31 ...
3	localhost 127.0.0.1 - - [23/Feb/2014:03:10:31 ...
4	localhost 127.0.0.1 - - [23/Feb/2014:03:10:31 ...
...	...
261868	www.akumenius.com 5.255.253.53 - - [02/Mar/201...
261869	www.akumenius.com 74.86.158.107 - - [02/Mar/20...
261870	localhost 127.0.0.1 - - [02/Mar/2014:03:10:18 ...
261871	localhost 127.0.0.1 - - [02/Mar/2014:03:10:18 ...
261872	localhost 127.0.0.1 - - [02/Mar/2014:03:10:18 ...

261873 rows × 1 columns

1.2 Tipos de registros del Data Frame

a) localhost 127.0.0.1 - - [23/Feb/2014:03:10:31 +0100] "OPTIONS * HTTP/1.0" 200 - "-" "Apache (internal dummy connection)" VLOG=-

b) www.akumenius.com 66.249.76.216 - - [23/Feb/2014:03:10:31 +0100] "GET /hoteles-baratos/ofertas-hotel-Club-&-Hotel-Letoonia--en-Fethiye-8460b-destinos.html HTTP/1.1" 404 3100 "-" "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)" VLOG=-

c) www.akumenius.com 162.243.192.191 - - [23/Feb/2014:09:11:20 +0100] "GET /escapadas/ocio-497/6.html HTTP/1.1" 200 15432 "http://www.akumenius.com/escapadas/ocio-497/4.html" "Mozilla/5.0 (compatible; spbot/4.0.7; +http://OpenLinkProfiler.org/bot)" VLOG=-

1.3 Descripció de la información y asignación de campos

""" El Data Frame recoge la iformación almacenada por el servidor en el registro de acceso sobre todas las peticiones que procesa.Combina dos tipos de Formatos de información:

I. El Formato Común de Registro (Common Log Format : CLF) que es el formato estándar que pueden generar muchos servidores web diferentes y ser leidos por muchos de los programas que analizan registros y,

II. El Formato de Registro Combinado (Combined Log Format CLF): Es exactamente igual que Formato Común de Registro, pero añade dos campos adicionales.

Cada una de las partes de la entrada se explican a continuación: """ """ registro b) [hostTitle] = "www.akumenius.com", nombre de servidor.

[ip] = "66.249.76.216", es la dirección IP del cliente (host remoto) que hizo la petición al servidor. Las direcciones IP que se registran no son necesariamente las direcciones de los usuarios finales. Si existe un servidor proxy entre el usuario final y el servidor, la dirección que se registra es la del proxy.

[no asignado] = "-": Un "guión" siginifica que la información que debería ir en ese lugar no está disponible. Esta información es generalmente poco fiable y no debería ser usada nunca excepto con clientes que estén sometidos a controles muy estrictos en redes internas. Apache httpd ni siquiera intenta recoger esa información a menos que la directiva IdentityCheck tenga valor On.

[no asignado]= "-": Identificador de usuario de la persona que solicita el documento determinado por la autentificación HTTP. Normalmente ese mismo valor se pasa a los scripts CGI con la variable de entorno REMOTE_USER. Si el código de estado de la petición es 401, entonces el usuario no ha sido aún autentificado. Si el documento no está protegido por contraseña, se mostrará un guión "-" en esta entrada.

[dateTime]= "[23/Feb/2014:03:10:31 +0100] ": La hora a la que el servidor recibió la petición. El formato es: [día/mes/año:hora:minuto:segundo zona_horaria]

[request]= "GET": La línea de petición contiene mucha información de utilidad. Primero, el método usado por el cliente es GET.

[requested]=" /hoteles-baratos/oferta.../", el segundo parámetro de la línea de petición es el recurso al que el cliente ha hecho una petición.

[protocolo]= "HTTP/1.1", es el tercer elemento de la línea de peticion es el protocolo.

[code] [code2] = "404 3100" , es el código de estado que el servidor envía de vuelta al cliente. Esta información es muy valiosa, porque revela si la petición fue satisfecha con éxito por el servidor (los códigos que empiezan por 2), si se produjo una redirección (los códigos que

empiezan por 3), un error provocado por el cliente (los códigos que empiezan por 4), ó un error en el servidor (los códigos que empiezan por 5).

[no asignado]= "-", la última entrada indica el tamaño del objeto retornado por el cliente, no incluídas las cabeceras de respuesta. Si no se respondió con ningún contenido al cliente, este valor mostrará valor "-".

[code_body]= "-" [userAccess] = "Mozilla/5.0 (compatible; Googlebot/2.1; +http://www.google.com/bot.html)", es la información de identificación que el navegador del cliente incluye sobre sí mismo.

registro c)

[code_body]="http://www.akumenius.com/escapadas/ocio-497/4.html", muestra el servidor del que proviene el cliente. """

In [ ]:
```
#### 1.4 Extracción de la información
```

Para realizar la conversión del archivo he utilizado una Regular Expression que estaba definida en la libreria de regex previamente por otro usuario.

Se ha verificado que la expresión definida coincidía con un registro aleatorio de la lectura de "data" y renombrado alguno de los campos

https://regex101.com/r/HD415R/1

In [6]:
```
regex = r'^(?P<hostTitle>.*?[A-Za-z\.]*) (?P<ip>[0-9\.]*) - - \[(?P<dateTime>.*)\] "
```

In [7]:
```
dataRegEx = data[0].str.extract(regex, flags=re.MULTILINE)
dataRegEx
```

Out[7]:

	hostTitle	ip	dateTime	request	requested	protocol	cod
0	localhost	127.0.0.1	23/Feb/2014:03:10:31 +0100	OPTIONS	*	HTTP/1.0	20
1	localhost	127.0.0.1	23/Feb/2014:03:10:31 +0100	OPTIONS	*	HTTP/1.0	20
2	localhost	127.0.0.1	23/Feb/2014:03:10:31 +0100	OPTIONS	*	HTTP/1.0	20
3	localhost	127.0.0.1	23/Feb/2014:03:10:31 +0100	OPTIONS	*	HTTP/1.0	20
4	localhost	127.0.0.1	23/Feb/2014:03:10:31 +0100	OPTIONS	*	HTTP/1.0	20
...	...	...	...	...	...	...	.
261868	www.akumenius.com	5.255.253.53	02/Mar/2014:03:05:39 +0100	GET	/	HTTP/1.1	20

	hostTitle	ip	dateTime	request	requested	protocol	cod
261869	www.akumenius.com	74.86.158.107	02/Mar/2014:03:09:52 +0100	HEAD	/	HTTP/1.1	20
261870	localhost	127.0.0.1	02/Mar/2014:03:10:18 +0100	OPTIONS	*	HTTP/1.0	20
261871	localhost	127.0.0.1	02/Mar/2014:03:10:18 +0100	OPTIONS	*	HTTP/1.0	20
261872	localhost	127.0.0.1	02/Mar/2014:03:10:18 +0100	OPTIONS	*	HTTP/1.0	20

261873 rows × 10 columns

In [45]:
```
dataRegEx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 261873 entries, 0 to 261872
Data columns (total 10 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   hostTitle   171674 non-null  object
 1   ip          171674 non-null  object
 2   dateTime    171674 non-null  datetime64[ns, pytz.FixedOffset(60)]
 3   request     171674 non-null  object
 4   requested   171674 non-null  object
 5   protocol    171674 non-null  object
 6   code        171674 non-null  object
 7   code2       171674 non-null  object
 8   code_body   171674 non-null  object
 9   userAccess  171674 non-null  object
dtypes: datetime64[ns, pytz.FixedOffset(60)](1), object(9)
memory usage: 20.0+ MB
```

Nivell II

Exercici 2

Neteja, preprocesa, estructura i transforma (dataframe) les dades del registre d'Accés a la web.

Modificamos el Formato con datatime y pasamos el mes a número para facilitar los cálculos entre diferente fechas

In [9]:
```
from datetime import datetime
dataRegEx['dateTime'] = pd.to_datetime(dataRegEx['dateTime'], format="%d/%b/%Y:%H:%M
dataRegEx
```

Out[9]:

	hostTitle	ip	dateTime	request	requested	protocol	code	coc

	hostTitle	ip	dateTime	request	requested	protocol	code	coc
0	localhost	127.0.0.1	2014-02-23 03:10:31+01:00	OPTIONS	*	HTTP/1.0	200	
1	localhost	127.0.0.1	2014-02-23 03:10:31+01:00	OPTIONS	*	HTTP/1.0	200	
2	localhost	127.0.0.1	2014-02-23 03:10:31+01:00	OPTIONS	*	HTTP/1.0	200	
3	localhost	127.0.0.1	2014-02-23 03:10:31+01:00	OPTIONS	*	HTTP/1.0	200	
4	localhost	127.0.0.1	2014-02-23 03:10:31+01:00	OPTIONS	*	HTTP/1.0	200	
...	...	...	...	...	...	...	...	
261868	www.akumenius.com	5.255.253.53	2014-03-02 03:05:39+01:00	GET	/	HTTP/1.1	200	7!
261869	www.akumenius.com	74.86.158.107	2014-03-02 03:09:52+01:00	HEAD	/	HTTP/1.1	200	
261870	localhost	127.0.0.1	2014-03-02 03:10:18+01:00	OPTIONS	*	HTTP/1.0	200	
261871	localhost	127.0.0.1	2014-03-02 03:10:18+01:00	OPTIONS	*	HTTP/1.0	200	
261872	localhost	127.0.0.1	2014-03-02 03:10:18+01:00	OPTIONS	*	HTTP/1.0	200	

261873 rows × 10 columns

In [10]:
```
dataRegEx.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 261873 entries, 0 to 261872
Data columns (total 10 columns):
 #   Column      Non-Null Count   Dtype
---  ------      --------------   -----
 0   hostTitle   171674 non-null  object
 1   ip          171674 non-null  object
 2   dateTime    171674 non-null  datetime64[ns, pytz.FixedOffset(60)]
 3   request     171674 non-null  object
 4   requested   171674 non-null  object
 5   protocol    171674 non-null  object
 6   code        171674 non-null  object
 7   code2       171674 non-null  object
 8   code_body   171674 non-null  object
```

```
 9   userAccess  171674 non-null  object
dtypes: datetime64[ns, pytz.FixedOffset(60)](1), object(9)
memory usage: 20.0+ MB
```

In [11]:
```
print("Periodo temporal del Data Frame de: ", dataRegEx.dateTime.min(),"a ", dataReg
```

```
Periodo temporal del Data Frame de:  2014-02-23 03:10:31+01:00 a  2014-03-02 03:10:1
8+01:00
```

Exercici 3

3.1 Geolocalitza les IP's.

3.1.1 Agrupamos las ip por número de visitas

In [12]:
```
ipNumVisits = dataRegEx["ip"].value_counts().rename_axis("ip").reset_index(name = "N
ipNumVisits
```

Out[12]:

	ip	NumVisits
0	66.249.76.216	45500
1	127.0.0.1	13892
2	80.28.221.123	12259
3	217.125.71.222	4014
4	66.249.75.148	3426
...	...	...
2479	200.98.200.32	1
2480	77.7.126.108	1
2481	200.46.114.194	1
2482	87.221.5.240	1
2483	206.198.5.33	1

2484 rows × 2 columns

Observaciones:

a) Reducción del Data Framen eliminando los registros correspondientes al localhost que son 13.892, al no tener asignada geolocalización y evitar errores en el siguente paso.

b) Adicionalmene hemos usado una APi que tiene restringido el número de llamadas para conocer la localización en funcion de la latitud y longitud informada de una ip. Hemos aplicado un filtro para consultar sólo las ip con visitas supreriores a 200.

In [13]:
```
ipNumVisits = ipNumVisits[(ipNumVisits["ip"] != "127.0.0.1") & (ipNumVisits["NumVisi
ipNumVisits
```

Out[13]:

	ip	NumVisits
0	66.249.76.216	45500
2	80.28.221.123	12259
3	217.125.71.222	4014

	ip	NumVisits
4	66.249.75.148	3426
5	62.117.197.230	2460
...	...	...
69	77.228.79.234	215
70	84.76.76.26	214
71	62.117.179.122	213
72	217.125.108.49	205
73	188.119.219.26	204

73 rows × 2 columns

In [14]:
```
ipNumVisits.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73 entries, 0 to 73
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   ip         73 non-null     object
 1   NumVisits  73 non-null     int64 
dtypes: int64(1), object(1)
memory usage: 1.7+ KB
```

In [15]:
```
print("Número de visitas Totales", ipNumVisits.NumVisits.sum())
```

```
Número de visitas Totales 96905
```

Vemos que las ip con visitas superiores a 200 corresponden a 73 registros

In [42]:
```
dataIps = pd.DataFrame({"ip": ipNumVisits["ip"].unique()})
dataIps.head(10)
```

Out[42]:

	ip
0	66.249.76.216
1	80.28.221.123
2	217.125.71.222
3	66.249.75.148
4	62.117.197.230
5	162.243.192.191
6	176.31.255.177
7	198.143.133.154
8	81.39.110.171
9	80.58.250.94

In [17]:
```
```

```
dataIps.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73 entries, 0 to 72
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   ip      73 non-null     object
dtypes: object(1)
memory usage: 712.0+ bytes
```

Hemos verificado que esos 73 registros se corresponden con ip´s únicas

3.1.2 Seleccionamos la localización georáfica de las ip, a partir de las latitud y la longitud, mendiante una API de geolocalización que nos devuelve el país y la region.

In [18]:
```
countError=0
# Definimos los parametros de respuesta que queremos obtener
parametros = 'status,country,countryCode,region,regionName,city,zip,lat,lon,timezone
data = {"fields":parametros}
response_text_list = []
api_url = "http://ip-api.com/json/"
for ip in dataIps["ip"]:
    try:
        if __name__ == '__main__':
            #print("ip: ",ip)
            # Llamamos a la función ip_scraping y mostramos los resultados
            res = requests.get(api_url+ip, data=data)
            # Obtenemos y procesamos la respuesta JSON
            api_json_res = json.loads(res.content)
            response_text_list.append(api_json_res)
            #print("algo :", api_json_res)
    except TypeError:
        countError =countError+1
        pass

print("Registro con ip erróneo: ",countError)
```

```
Registro con ip erróneo:  0
```

In [19]:
```
geo_ip = pd.DataFrame.from_dict(response_text_list)
```

In [20]:
```
geo_ip.head(5)
```

Out[20]:

	status	country	countryCode	region	regionName	city	zip	lat	lon	
0	success	United States	US	CA	California	Mountain View	94043	37.4220	-122.08400	Amer
1	success	Spain	ES	MD	Madrid	Madrid	28760	40.5167	-3.66479	
2	success	Spain	ES	AN	Andalusia	Tocina	41340	37.6025	-5.73070	
3	success	United States	US	CO	Colorado	Aurora	80014	39.6663	-104.83430	

	status	country	countryCode	region	regionName	city	zip	lat	lon
4	success	Spain	ES	MD	Madrid	Madrid	28012	40.4163	-3.69340

In [21]:
```
geo_ip.rename(columns={'query': 'ip'}, inplace=True)
geo_ip.head(5)
```

Out[21]:

	status	country	countryCode	region	regionName	city	zip	lat	lon	
0	success	United States	US	CA	California	Mountain View	94043	37.4220	-122.08400	Amer
1	success	Spain	ES	MD	Madrid	Madrid	28760	40.5167	-3.66479	
2	success	Spain	ES	AN	Andalusia	Tocina	41340	37.6025	-5.73070	
3	success	United States	US	CO	Colorado	Aurora	80014	39.6663	-104.83430	
4	success	Spain	ES	MD	Madrid	Madrid	28012	40.4163	-3.69340	

Simplificamos el Data Frame para obtener las coordenadas que usaremos como base del mapa de geolocalizaciones

In [22]:
```
df_grafico =pd.DataFrame(geo_ip, columns=["ip", "country","regionName", "lat","lon"]
df_grafico.head(5)
```

Out[22]:

	ip	country	regionName	lat	lon
0	66.249.76.216	United States	California	37.4220	-122.08400
1	80.28.221.123	Spain	Madrid	40.5167	-3.66479
2	217.125.71.222	Spain	Andalusia	37.6025	-5.73070
3	66.249.75.148	United States	Colorado	39.6663	-104.83430
4	62.117.197.230	Spain	Madrid	40.4163	-3.69340

In [23]:
```
import geopandas as gpd
from shapely.geometry import Point, Polygon
```

In [24]:
```
# sistema de referència de coordenades
#crs = 'epsg:4269'
crs = "epsg:4326","epsg:4269"
```

In [25]:
```
geometry = [Point(xy) for xy in zip(df_grafico["lon"], df_grafico["lat"])]
```

In [26]:
```
df_geo = gpd.GeoDataFrame(df_grafico, geometry=geometry)
df_geo.head(5)
```

Out[26]:

	ip	country	regionName	lat	lon	geometry
0	66.249.76.216	United States	California	37.4220	-122.08400	POINT (-122.08400 37.42200)
1	80.28.221.123	Spain	Madrid	40.5167	-3.66479	POINT (-3.66479 40.51670)
2	217.125.71.222	Spain	Andalusia	37.6025	-5.73070	POINT (-5.73070 37.60250)
3	66.249.75.148	United States	Colorado	39.6663	-104.83430	POINT (-104.83430 39.66630)
4	62.117.197.230	Spain	Madrid	40.4163	-3.69340	POINT (-3.69340 40.41630)

In [27]:
```
df_geo_merger = df_geo[["ip", "country", "regionName","lat", "lon","geometry"]]
df_geo_NumVisits = pd.merge(ipNumVisits, df_geo_merger, on="ip")
df_geo_NumVisits.head(10)
```

Out[27]:

	ip	NumVisits	country	regionName	lat	lon	geometry
0	66.249.76.216	45500	United States	California	37.4220	-122.08400	POINT (-122.08400 37.42200)
1	80.28.221.123	12259	Spain	Madrid	40.5167	-3.66479	POINT (-3.66479 40.51670)
2	217.125.71.222	4014	Spain	Andalusia	37.6025	-5.73070	POINT (-5.73070 37.60250)
3	66.249.75.148	3426	United States	Colorado	39.6663	-104.83430	POINT (-104.83430 39.66630)
4	62.117.197.230	2460	Spain	Madrid	40.4163	-3.69340	POINT (-3.69340 40.41630)
5	162.243.192.191	2049	United States	New York	40.7597	-73.98100	POINT (-73.98100 40.75970)
6	176.31.255.177	1044	France	Hauts-de-France	50.6917	3.20157	POINT (3.20157 50.69170)
7	198.143.133.154	1038	United States	Illinois	41.8786	-87.63110	POINT (-87.63110 41.87860)
8	81.39.110.171	1006	Spain	Madrid	40.4163	-3.69340	POINT (-3.69340 40.41630)
9	80.58.250.94	928	Spain	Madrid	40.5167	-3.66479	POINT (-3.66479 40.51670)

In [28]:
```
sumaNumVisits_200 =df_geo_NumVisits.NumVisits.sum()
print("Total visitas de las ip´s con >200 :",sumaNumVisits_200)
```

```
Total visitas de las ip´s con >200 : 96905
```

In [29]:
```
df_geo_NumVisits["pct_NumVisits"] = (df_geo_NumVisits["NumVisits"]/sumaNumVisits_200
df_geo_NumVisits.head(10)
```

Out[29]:

	ip	NumVisits	country	regionName	lat	lon	geometry	pct_NumVisits
0	66.249.76.216	45500	United States	California	37.4220	-122.08400	POINT (-122.08400 37.42200)	46.953202
1	80.28.221.123	12259	Spain	Madrid	40.5167	-3.66479	POINT (-3.66479 40.51670)	12.650534
2	217.125.71.222	4014	Spain	Andalusia	37.6025	-5.73070	POINT (-5.73070 37.60250)	4.142201
3	66.249.75.148	3426	United States	Colorado	39.6663	-104.83430	POINT (-104.83430 39.66630)	3.535421
4	62.117.197.230	2460	Spain	Madrid	40.4163	-3.69340	POINT (-3.69340 40.41630)	2.538569
5	162.243.192.191	2049	United States	New York	40.7597	-73.98100	POINT (-73.98100 40.75970)	2.114442
6	176.31.255.177	1044	France	Hauts-de-France	50.6917	3.20157	POINT (3.20157 50.69170)	1.077344
7	198.143.133.154	1038	United States	Illinois	41.8786	-87.63110	POINT (-87.63110 41.87860)	1.071152
8	81.39.110.171	1006	Spain	Madrid	40.4163	-3.69340	POINT (-3.69340 40.41630)	1.038130
9	80.58.250.94	928	Spain	Madrid	40.5167	-3.66479	POINT (-3.66479 40.51670)	0.957639

In [30]:

```
print("Suma de porcentajes de Visitas en USA + Spain :",df_geo_NumVisits.pct_NumVisi
```

```
Suma de porcentajes de Visitas en USA + Spain : 59.6037356173572
```

3.1.3 Gráfico 1: Número de Visitas por País y Región (numVisitas>200)

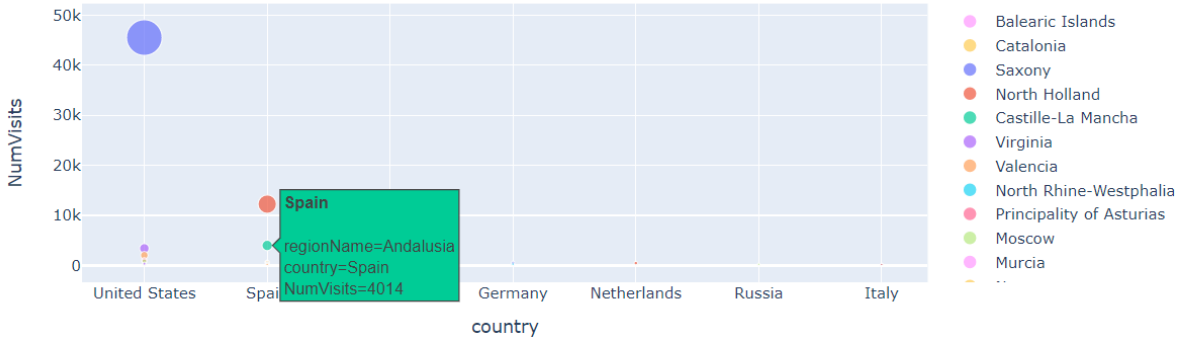
In [37]:

```
import plotly.express as px
fig= px.scatter(df_geo_NumVisits, x="country", y="NumVisits", animation_group="count
fig.show()
fig.write_html("Grafico1_Visitas_Country_Region.html")
```

Como se observa en el gráfico dinámico, las ip con mayores accesos o "visitas" se corresponden con United States y Spain, que concentran el 59,60% de las visitas de las ip >200

En cada eje, por cada country, se detallan las regionName con el volumen del número de Visitas, siendo la región con mayor número California con 45.500

En el gráfico estático siguiente se han resaltado los datos correspondientes a España, regionNames = Andalucía con número de visitas de 4.014.



3.1.4 Gráfico 2: Localización geográfica de las ip´s con "visitas">200

In [33]:
```
dfg_m =pd.DataFrame(df_grafico, columns=["lat","lon"]).copy()
dfg_m.head(5)
```

Out[33]:

	lat	lon
0	37.4220	-122.08400
1	40.5167	-3.66479
2	37.6025	-5.73070
3	39.6663	-104.83430
4	40.4163	-3.69340

In [34]:
```
# importar folio en Python
```

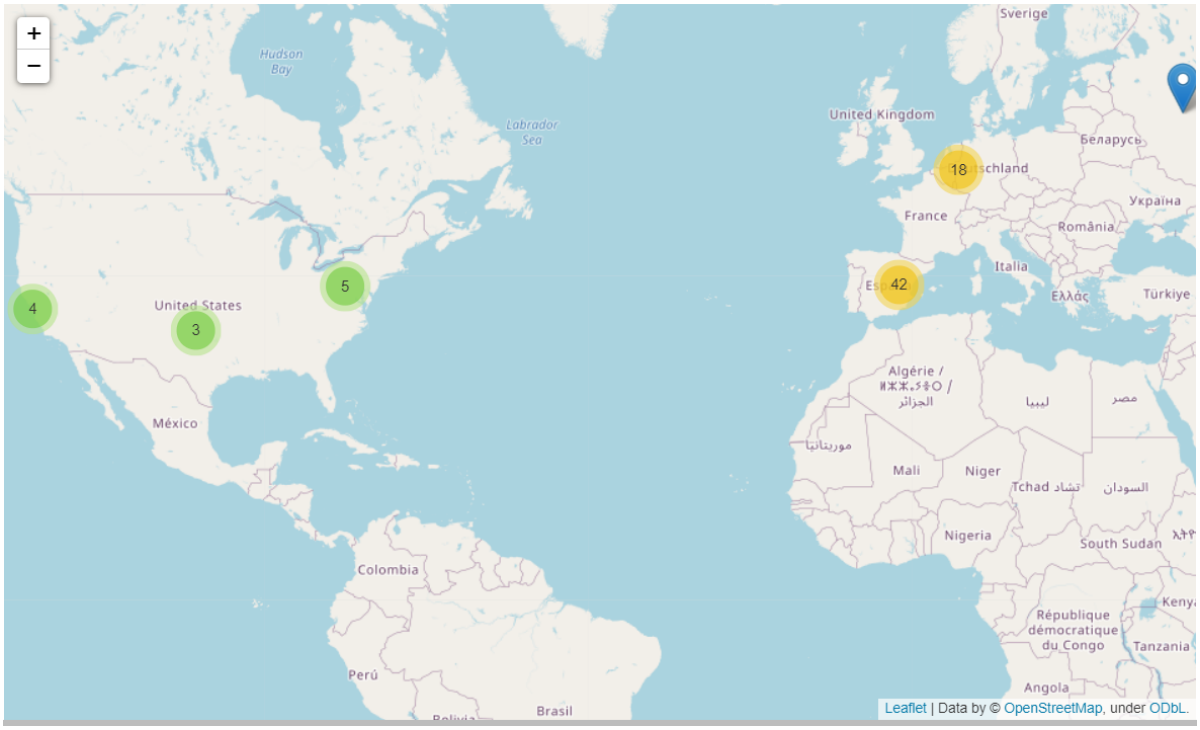
```
import folium
from folium.plugins import MarkerCluster

m = folium.Map(location=[40.4163,-3.69340],width="%100",height="%100")
location= dfg_m[["lat","lon"]]
#class folium.plugins.MarkerCluster(locations=None, popups=None, icons=None, name=No
MarkerCluster(location,control=True, Show=True).add_to(m)
m
```

Out[34]:

In [275…

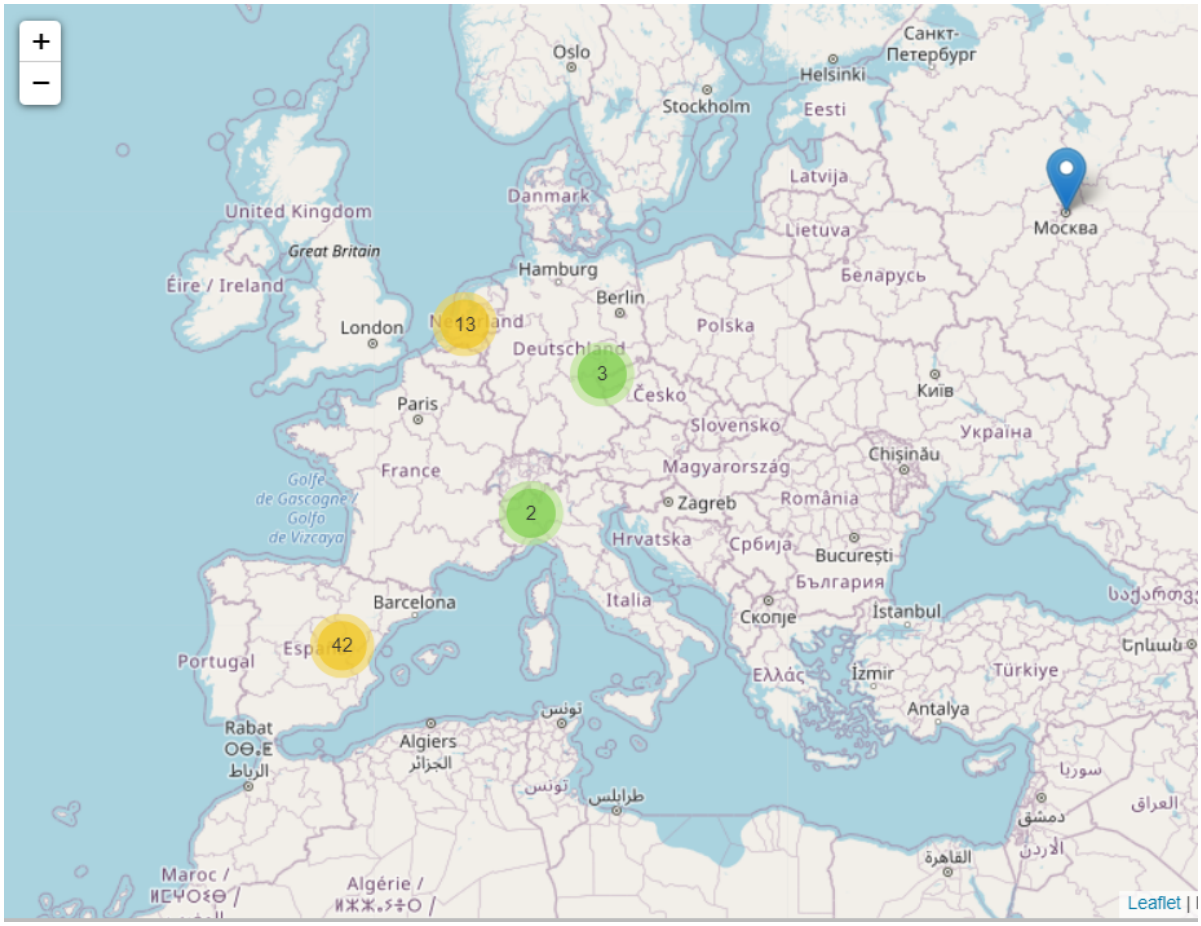
```
m.save("Grafico2_Geolocalización_Country_Region.html")
```

3.1.4.a) Direcciones ip geolocalizadas por continente



De las 73 ip con mayores visitas (>200), se han locacilzado 12 en USA, 42 en España, 18 en el resto de la Unión Europea y 1 en Rusia (Moscú)

3.1.4.b) Direcciones ip geolocalizada en Europa



En Europa la distribución geográfica de las ip's, se corresponden con: España 42, Paises Bajos 13, Alemania 3 e Italia 2.

3.1.4.c) Direcciones ip geolocalizada en España

De las direcciones ip geolocalizadas en España y con visitas superiores a 200, 15 se encuentran en el área de Madrid, 7 en Cataluña, 6 en Baleares, 5 Andalucía, 2 en Valencia, 2 en Castilla la Mancha, 2 en Asturias, una en Navarra y una Castellón.

Nivell 3

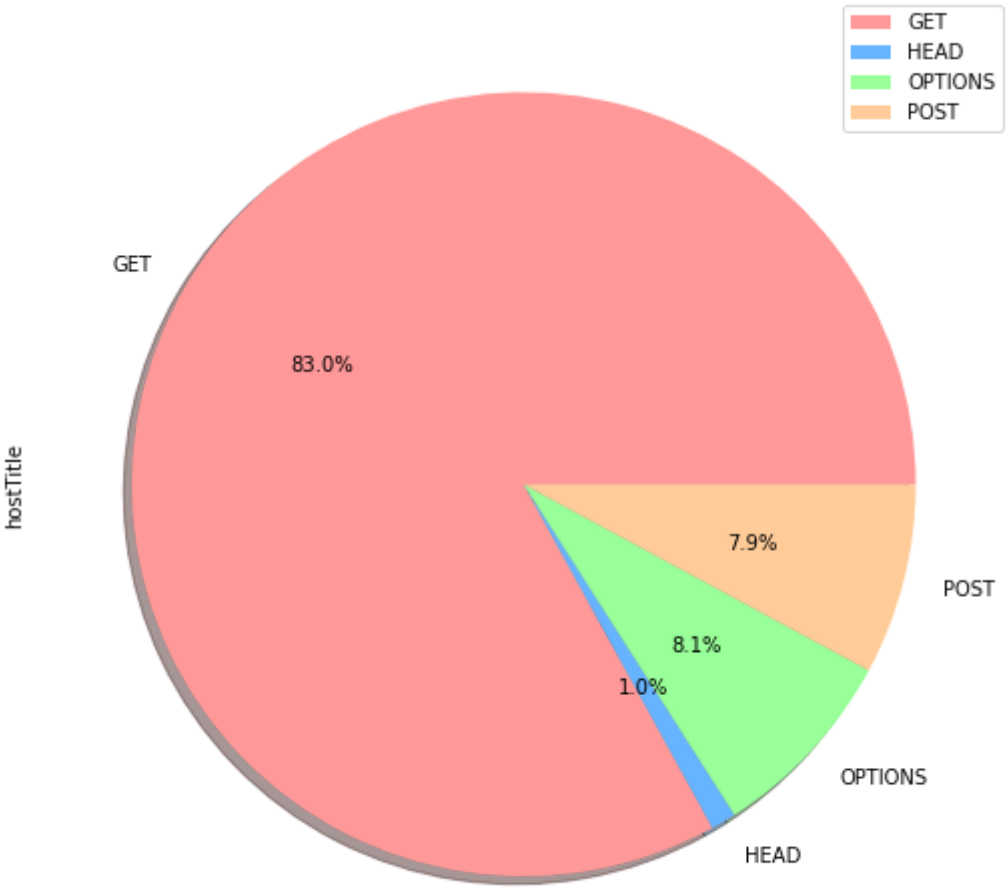
Exercici 3

Mostra'm la teva creativitat, Sorprèn-me fes un pas més enllà amb l'anàlisi anterior.

3.1 Gráfico 3: El request o método más común

In [50]:
```
import matplotlib.pyplot as plt
from matplotlib import cm
from matplotlib import colors
colors = ['#ff9999','#66b3ff','#99ff99','#ffcc99']
dataRegEx[['request', 'hostTitle']].groupby('request').count().plot(kind='pie', figs
```

Out[50]:
```
array([<AxesSubplot:ylabel='hostTitle'>], dtype=object)
```

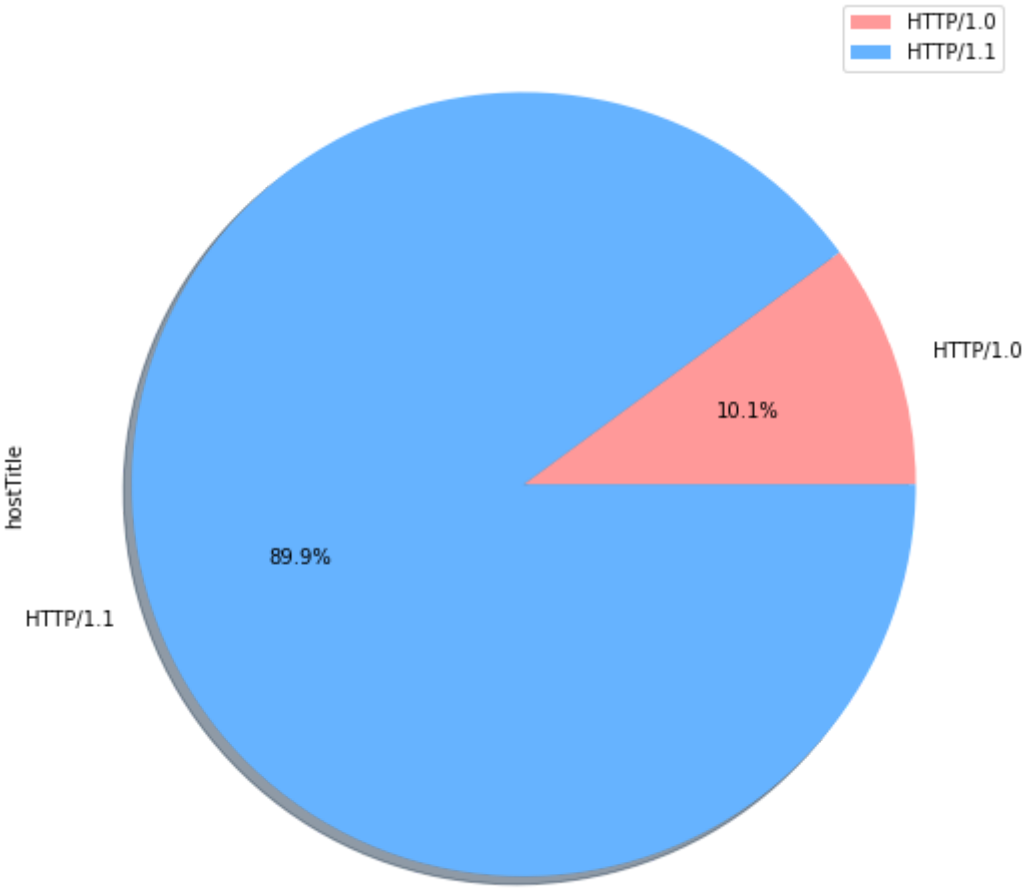


El método más común es el GET que supone el 83,01% de los datos informados en el Data Frame original

3.2 Gráfico 4: El protocolo más habitual

In [51]:
```
dataRegEx[['protocol', 'hostTitle']].groupby('protocol').count().plot(kind='pie', fi
```

Out[51]:
```
array([<AxesSubplot:ylabel='hostTitle'>], dtype=object)
```

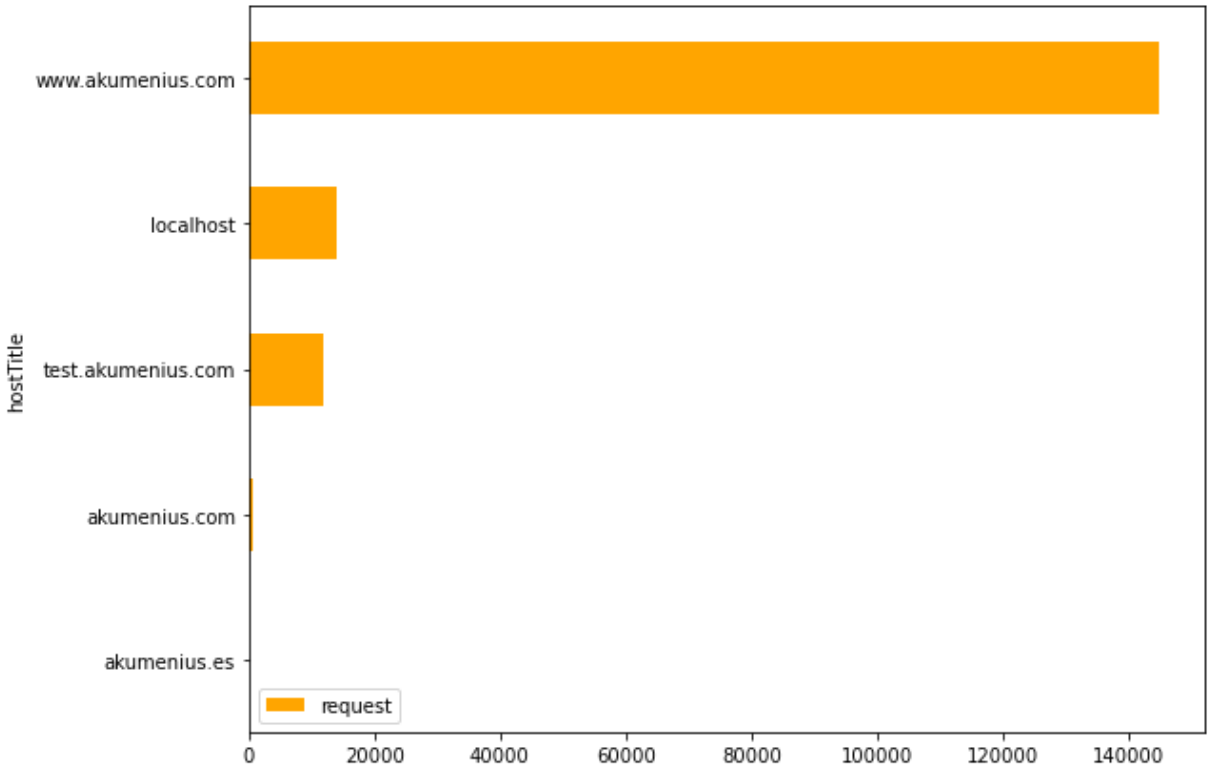


Los protocolos más usados e informados en el Data Frame son el HTTP/1.1 y 1.0, con un reparto del 80,9 y 10,1% respectivamente.

3.3 Gráfico 5: Domínio más demandado - Relación entrel el HostTitle y el campo request

In [46]:
```
dataRegEx[['hostTitle', 'request']].groupby('hostTitle').count().sort_values(by='req
```

Out[46]:
```
<AxesSubplot:ylabel='hostTitle'>
```



El dominio más demandado con gran diferencia respecto al resto es www.akumenius.com con mas de 14.000 request, seguido por el "localhost"

3.4 Elementos más solicitados - relación entre requested/request:

In [47]:
```
dataRequested = dataRegEx[['request', 'requested']]
dataRequested_ok = dataRequested[dataRequested["requested"] != "*"].groupby('request
dataRequested_ok.head(10)
```

Out[47]:

requested	request
/destinos-get	6818
/	3927
/hotel-list-data/	2093
/hotel-list	1442
/raton-search	1341
/hotels-consulted-update	1007
/icon.png	982
/includes/css/style.css	830
/includes/images/uploaded/logo.png	780
/newdesign/libraries/anythingSlider/images/1r.png	737

In [279…
```
dataRequested_ok.info()
```

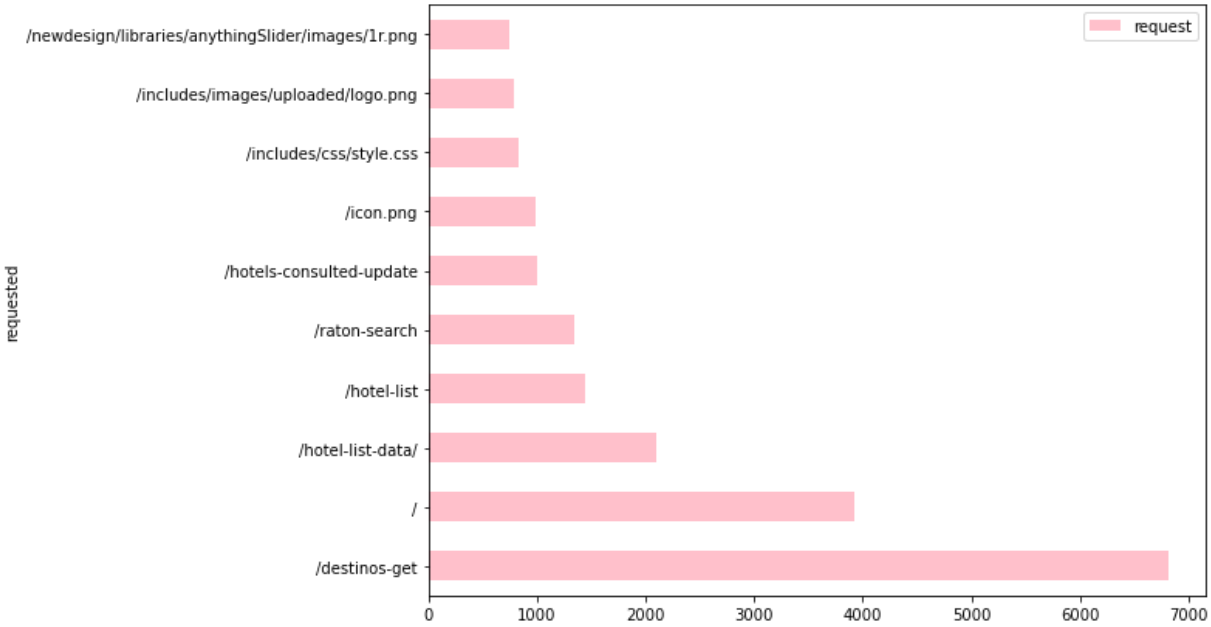
```
<class 'pandas.core.frame.DataFrame'>
Index: 63454 entries, /destinos-get to /wp/wp-login.php
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
```

```
---  ------   --------------  -----
 0   request  63454 non-null  int64
dtypes: int64(1)
memory usage: 991.5+ KB
```

3.4.a) Gráfico 6: Número de request por requested (los 10 primeros)

In [280…

```
dataRequested_ok.head(10).plot(kind='barh',color="pink" ,figsize=(9,7));
```



De los diez primeros requested, el que tiene mayores requerimientos es "/destinos-get" con 6.818, mientras que el segundo corresponde a un elemento no informdo, con 3.927 y el tercero corresponde a "/hotel-lst.data/", con 2.093

In [ ]: