S05_T01

October 30, 2021

1 S05 T01: Transformació Registre Log amb Regular expressions

1.0.1 Descripció

L'anàlisi de registres és una funció important per al control i l'alerta, el compliment de les polítiques de seguretat, l'auditoria i el compliment normatiu, la resposta a incidents de seguretat i fins i tot les investigacions forenses. En analitzar les dades de registre, les empreses poden identificar més fàcilment les possibles amenaces i altres problemes, trobar la causa arrel i iniciar una resposta ràpida per mitigar els riscos.

L'analista ha d'assegurar-se que els registres consisteixen en una gamma completa de missatges i s'interpreten segons el context. Els elements de registre han d'estandaritzar-se, utilitzant els mateixos termes o terminologia, per evitar confusions i proporcionar cohesió.

Com Científic de Dades se t'ha proporcionat accés als registres-Logs on queda registrada l'activitat de totes les visites a realitzades a la pàgina web de l'agència de viatges "akumenius.com".

1.0.2 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".

Atés que els Common Log Format son arxius de text estandaritzats, cada línia guardada en aquest format té la següent sintaxis:

```
- host
- IP address
- RFC identifyer
- user id: serà '-' en cas de dominis públics.
- datetime, entre brackets []
- request, entre doble cometes " "
- status response:
    * 2xx: resposta satisfactòria
    * 3xx: redirigit
    * 4xx: error client
    * 5xx: error servidor
- bytes
- URL d'on prové l'usuari, entre doble cometes " "
- browser, entre doble cometes " ".
- cookies
```

Un '-' en qualsevol part de la línia indica dades faltants o 'NaN'.

[3]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re

columns = ['domain', 'IP_address','RFC_id', 'user_id', 'datetime', 'request',␣
↪'status_response', 'bytes','from_URL', 'browser', 'cookies']

delimiter = r" (.+|-) (.+|-) (.+|-) \[(.*) \+0100\] (.+) (\d+|-) (\d+|-) \"(.
↪+)\" \"(.+)\" VLOG="

txt = pd.read_csv("/Users/deliagonzalezmata/Documents/IT_Academy/
↪Git_Data-Science/Data-sources/Web_access_log-akumenius.com.txt",
                  delimiter=delimiter,
                    engine="python",
                    names=columns,
                    na_values="-")

txt.sample(15)
```

[3]:

	domain	IP_address	RFC_id	user_id	\
179109	www.akumenius.com	79.145.213.52	NaN	NaN	
166486	www.akumenius.com	80.28.221.123	NaN	NaN	
39442	www.akumenius.com	144.76.95.232	NaN	NaN	
149178	www.akumenius.com	173.252.112.115	NaN	NaN	
220837	www.akumenius.com	81.203.188.244	NaN	NaN	
172716	localhost	127.0.0.1	NaN	NaN	
104401	test.akumenius.com	217.125.71.222	NaN	NaN	
88780	www.akumenius.com	87.219.136.85	NaN	NaN	
163556	www.akumenius.com	195.53.184.207	NaN	NaN	
233581	www.akumenius.com	157.55.32.89	NaN	NaN	
102796	www.akumenius.com	77.224.108.21	NaN	NaN	
91195	www.akumenius.com	66.249.76.216	NaN	NaN	
32768	www.akumenius.com	66.249.76.216	NaN	NaN	
3039	www.akumenius.com	5.10.83.52	NaN	NaN	
175326	www.akumenius.com	85.49.138.49	NaN	NaN	

	datetime	\
179109	26/Feb/2014:22:48:04	
166486	26/Feb/2014:15:33:22	
39442	23/Feb/2014:23:26:56	
149178	26/Feb/2014:02:31:28	
220837	27/Feb/2014:22:04:20	
172716	26/Feb/2014:18:56:44	

```
104401  25/Feb/2014:12:40:48
88780   24/Feb/2014:23:10:20
163556  26/Feb/2014:14:04:25
233581  28/Feb/2014:12:06:54
102796  25/Feb/2014:12:05:08
91195   25/Feb/2014:01:52:57
32768   23/Feb/2014:20:34:35
3039    23/Feb/2014:04:28:25
175326  26/Feb/2014:20:14:06

                                                  request  status_response  \
179109  "GET /libraries/jqueryui/js/jquery.ui.slider.m…              304
166486  "GET /modules/raton/views/themes/bcoos/images/…              304
39442   "GET /destinos-baratos/destinos-caracteristica…              200
149178  "GET /modules/raton/views/themes/bcoos/images/…              200
220837  "GET /libraries/anythingSlider/images/2a.png H…              200
172716                              "OPTIONS * HTTP/1.0"                200
104401  "GET /newdesign/libraries/anythingSlider/image…              304
88780   "GET /modules/raton/views/themes/bcoos/images/…              200
163556                       "POST /destinos-get HTTP/1.1"              200
233581  "GET /escapadas/escapada-bodega-borda-sabate-h…              200
102796  "GET /modules/raton/views/themes/bcoos/images/…              200
91195   "GET /destinos-caracteristicas/hoteles-baratos…              200
32768   "GET /destinos-caracteristicas/hoteles-baratos…              200
3039    "GET /hoteles-baratos/ofertas-hotel-Roulette-C…              404
175326  "GET /modules/raton/views/themes/bcoos/images/…              200

          bytes                                              from_URL  \
179109      NaN                    http://www.akumenius.com/hotel-list
166486      NaN                    http://www.akumenius.com/hotel-list
39442   11371.0                                                    NaN
149178    208.0  http://www.akumenius.com/destinosCaracteristic…
220837   2510.0                              http://www.akumenius.com/
172716      NaN                                                    NaN
104401      NaN                   http://test.akumenius.com/newdesign/
88780    3638.0  http://www.akumenius.com/destinos-baratos/hote…
163556    219.0                              http://www.akumenius.com/
233581   3100.0                                                    NaN
102796  12237.0  http://www.akumenius.com/booking/146677/b/Apol…
91195    7222.0                                                    NaN
32768   14624.0                                                    NaN
3039     3100.0                                                    NaN
175326   5150.0                              http://www.akumenius.com/

                                                  browser  cookies
179109  Mozilla/4.0 (compatible; MSIE 8.0; Windows NT …      NaN
166486  Mozilla/5.0 (X11; U; Linux i686; ca; rv:1.9.2…      NaN
```

```
39442   Mozilla/5.0 (compatible; MJ12bot/v1.4.4; http:…      NaN
149178  facebookexternalhit/1.1 (+http://www.facebook…      NaN
220837  Mozilla/5.0 (Windows NT 6.1; WOW64; rv:27.0) G…      NaN
172716                 Apache (internal dummy connection)      NaN
104401  Mozilla/5.0 (Windows NT 6.0; WOW64) AppleWebKi…      NaN
88780   Mozilla/5.0 (Windows NT 6.2; WOW64) AppleWebKi…      NaN
163556  Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKi…      NaN
233581  Mozilla/5.0 (compatible; bingbot/2.0; +http://…      NaN
102796  Mozilla/5.0 (compatible; MSIE 10.0; Windows NT…      NaN
91195   Mozilla/5.0 (compatible; Googlebot/2.1; +http:…      NaN
32768   Mozilla/5.0 (compatible; Googlebot/2.1; +http:…      NaN
3039    Mozilla/5.0 (compatible; AhrefsBot/5.0; +http:…      NaN
175326  Mozilla/5.0 (Windows NT 6.3; WOW64) AppleWebKi…      NaN
```

1.0.3 Exercici 2

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

[4]:
```
txt.dtypes
```

[4]:
```
domain              object
IP_address          object
RFC_id             float64
user_id             object
datetime            object
request             object
status_response      int64
bytes              float64
from_URL            object
browser             object
cookies            float64
dtype: object
```

[5]:
```
txt.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 261873 entries, 0 to 261872
Data columns (total 11 columns):
 #   Column           Non-Null Count   Dtype
---  ------           --------------   -----
 0   domain           261873 non-null  object
 1   IP_address       261873 non-null  object
 2   RFC_id           0 non-null       float64
 3   user_id          27 non-null      object
 4   datetime         261873 non-null  object
 5   request          261873 non-null  object
 6   status_response  261873 non-null  int64
 7   bytes            219538 non-null  float64
```

```
 8   from_URL         162326 non-null  object
 9   browser          261654 non-null  object
 10  cookies          0 non-null       float64
dtypes: float64(3), int64(1), object(7)
memory usage: 22.0+ MB
```

Veient que les columnes 'RFC_id' i 'cookies' només tenen valors nuls, les eliminem del dataset.

[6]:
```
txt = txt.drop('RFC_id', 1)
txt = txt.drop('cookies', 1)
```

```
/var/folders/f1/1k69t1011n32zcq6vt7pt73c0000gn/T/ipykernel_1090/1110556357.py:1:
FutureWarning: In a future version of pandas all arguments of DataFrame.drop
except for the argument 'labels' will be keyword-only
  txt = txt.drop('RFC_id', 1)
/var/folders/f1/1k69t1011n32zcq6vt7pt73c0000gn/T/ipykernel_1090/1110556357.py:2:
FutureWarning: In a future version of pandas all arguments of DataFrame.drop
except for the argument 'labels' will be keyword-only
  txt = txt.drop('cookies', 1)
```

[7]:
```
list(txt.columns)
```

[7]:
```
['domain',
 'IP_address',
 'user_id',
 'datetime',
 'request',
 'status_response',
 'bytes',
 'from_URL',
 'browser']
```

1.0.4 domain

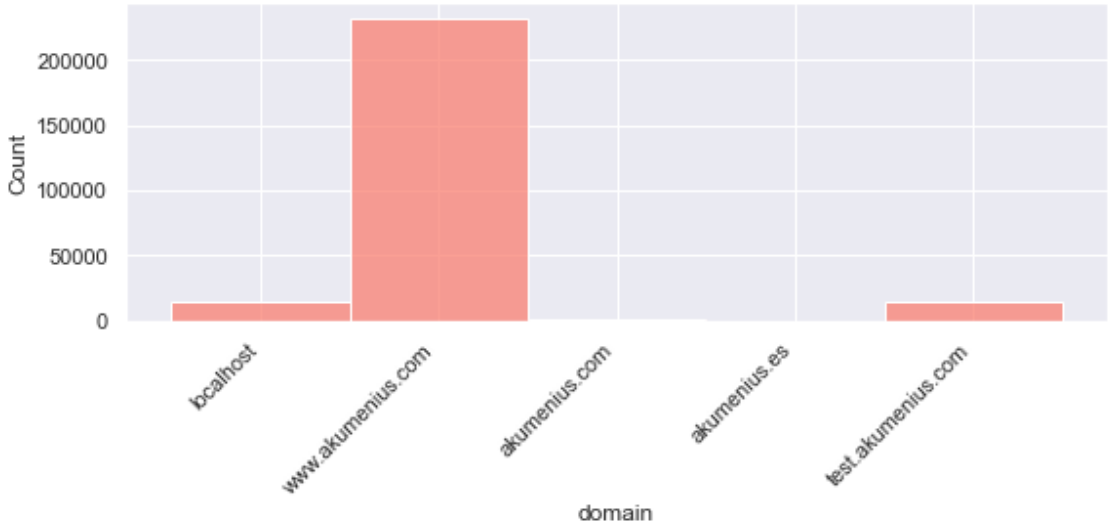
Tenim 5 dominis entre tots els usuaris. El més popular és www.akumenius.com, mentre que akumenius.com i akumenius.es no semblen ser gaire utilitzats:

[8]:
```
txt.domain.value_counts()
```

[8]:
```
www.akumenius.com     232300
test.akumenius.com     14610
localhost              14127
akumenius.com            742
akumenius.es              94
Name: domain, dtype: int64
```

[9]:
```
domain_counts = txt.domain.value_counts()

sns.set(rc = {'figure.figsize':(9,3)})
```

```
sns.set(style='darkgrid')
sns.histplot(data = txt, x = 'domain', color = 'salmon')

plt.xticks(rotation = 45, horizontalalignment = 'right')

plt.show()
```



1.0.5 IP_address

tenim 2.921 adreces IP diferents, que més endavant geolocalitzarem:

[10]:
```
txt.IP_address.value_counts()
```

[10]:
```
66.249.76.216      46382
80.28.221.123      14725
127.0.0.1          13892
217.125.71.222      5201
66.249.75.148       3558
                   ...
84.123.150.27          1
217.130.150.116        1
202.46.52.23           1
216.151.130.170        1
206.198.5.33           1
Name: IP_address, Length: 2921, dtype: int64
```

1.0.6 user_id

com hem vist abans amb la funció txt.info() només hi han 27 registres de user_id i la resta son valors nuls. Comprovem quins registres han quedat guardats:

[11]:
```
print(txt.user_id.unique())
```

```
[nan 'clarcat']
```

1.0.7 datetime:

la informació que se'ns dóna per defecte és del tipus:

'%d/%b/%Y:%H:%M:%S'

Abans hem vist que la columna anomenada datetime és del tipus 'object' dins la nostra BD. La transformem per a que tingui el tipus 'datetime' i poder treballar amb ella:

[12]:
```
txt['datetime'] = pd.to_datetime(txt['datetime'], format="%d/%b/%Y:%H:%M:%S")
txt.datetime.head()
```

[12]:
```
0   2014-02-23 03:10:31
1   2014-02-23 03:10:31
2   2014-02-23 03:10:31
3   2014-02-23 03:10:31
4   2014-02-23 03:10:31
Name: datetime, dtype: datetime64[ns]
```

Un cop transformada la columna 'datetime' en el tipus que volem, dividirem la informació en 2 columnes diferents: - una que contingui informació sobre la data (date) - altra que contingui informació sobre l'hora (time)

[13]:
```
txt['date'] = txt['datetime'].dt.date
txt['time'] = txt['datetime'].dt.time

txt[['date','time']].sample(10)
```

[13]:

	date	time
102599	2014-02-25	12:02:46
243233	2014-03-01	00:44:38
50075	2014-02-24	09:14:26
206084	2014-02-27	16:13:58
181908	2014-02-27	00:12:35
35611	2014-02-23	21:47:15
75083	2014-02-24	16:15:56
124947	2014-02-25	17:41:24
133495	2014-02-25	19:01:39
219690	2014-02-27	21:00:10

[14]:
```
txt[['date']].sort_index()
```

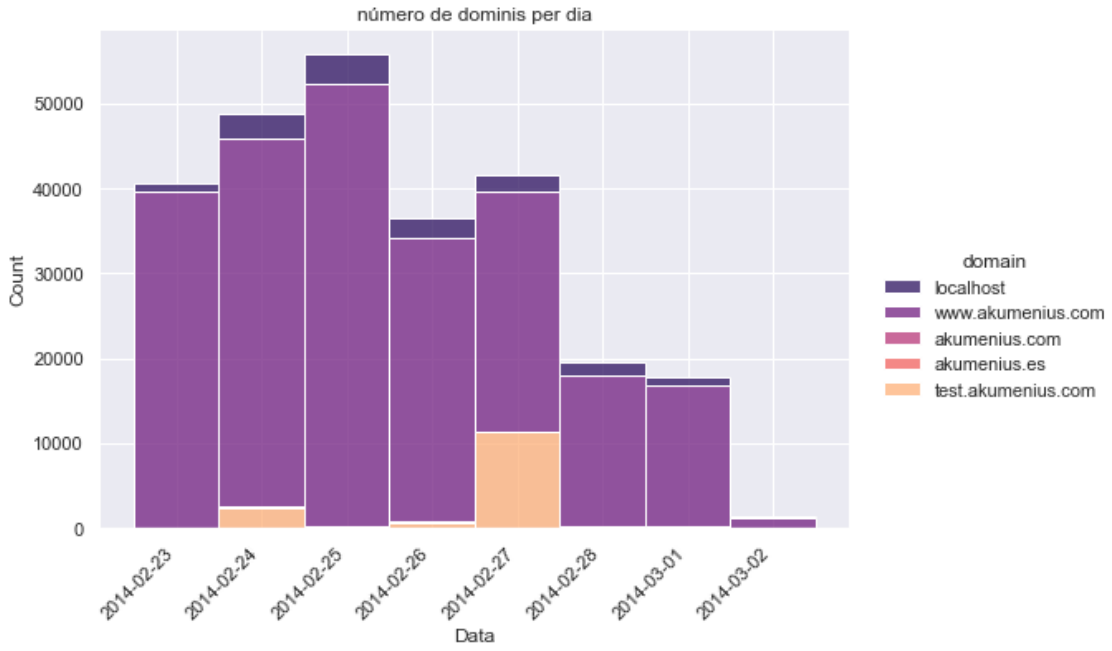
[14]:

	date
0	2014-02-23
1	2014-02-23
2	2014-02-23

```
3         2014-02-23
4         2014-02-23
…                …
261868    2014-03-02
261869    2014-03-02
261870    2014-03-02
261871    2014-03-02
261872    2014-03-02

[261873 rows x 1 columns]
```

Les dates entre les quals hem obtingut registres son la setmana del 23/02/2014 (diumenge) fins el 02/03/2014 (diumenge), ambdós inclosos.

[15]:
```
sns.displot(data=txt, x='date', kind='hist', hue='domain', multiple='stack',␣
↪palette='magma', aspect=1.5)

plt.title('número de dominis per dia')
plt.xlabel('Data')
plt.xticks(rotation = 45, horizontalalignment = 'right')

plt.show()
```



1.0.8 request:

les files de la columna "request" mostren el requeriment del client. Estan formades per 3 parts: - el mètode GET ('method') - el recurs sol•licitat ('resource') - el protocol HTTP ('http_protocol')

Crearem 3 noves col•lumnes que separin cada un dels elements:

[16]:
```
regex = re.compile(r"\"(?P<method>\w*) (?P<resource>[\/*].*?) (?
↪P<http_protocol>.*?)\"")

df_request = txt['request'].str.extract(regex)
df_request.sample(10)
```

[16]:
```
         method                                              resource  \
177341      GET              /includes/images/uploaded/ti_home.jpg
30483      POST                                        /destinos-get
188561     POST                                        /destinos-get
148861      GET  /modules/raton/views/themes/bcoos/images/logo_…
242983  OPTIONS                                                   *
246622      GET  /libraries/jqueryui/css/ui-lightness/images/ui…
186568      GET           //libraries/anythingSlider/images/3a.png
92892       GET  /destinosCaracteristicas/hoteles-baratos-en-Ge…
106911      GET  /destinosCaracteristicas/hoteles-baratos-en-Ca…
80812       GET                                        /robots.txt

       http_protocol
177341      HTTP/1.1
30483       HTTP/1.1
188561      HTTP/1.1
148861      HTTP/1.1
242983      HTTP/1.0
246622      HTTP/1.1
186568      HTTP/1.1
92892       HTTP/1.1
106911      HTTP/1.1
80812       HTTP/1.1
```

[17]:
```
df_request.method.value_counts()
```

[17]:
```
GET        229482
POST        16669
OPTIONS     13892
HEAD         1788
Name: method, dtype: int64
```

[18]:
```
df_request.http_protocol.value_counts()
```

[18]:
```
HTTP/1.1    241991
HTTP/1.0     19840
```

```
Name: http_protocol, dtype: int64
```

1.0.9 status_response

Els codis de resposta indiquen si s'ha completat satisfactoriament una sol•licitud o bé si hi ha hagut algun tipus d'error. Depenent de quin sigui el codi que s'imprimeix significa:

- 2xx: resposta satisfactòria
- 3xx: redirigit
- 4xx: error client
- 5xx: error servidor

Veiem les diferents respostes que hem obtingut mitjançant els registres proporcionats:

[19]:
```
txt.status_response.value_counts().sort_index()
```

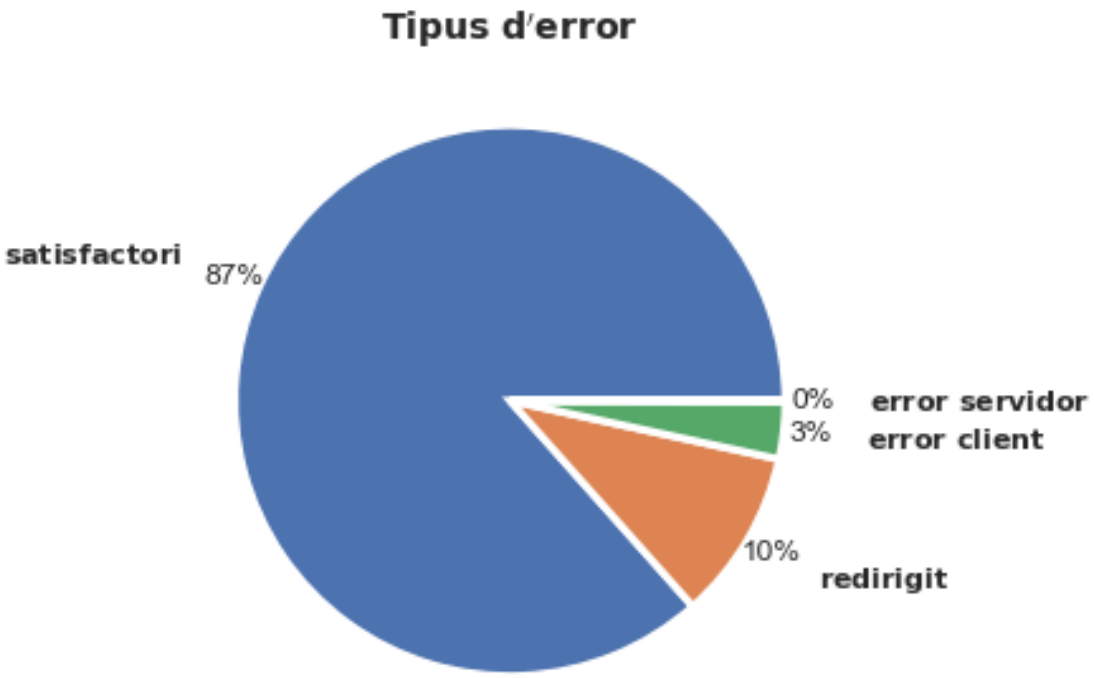
[19]:
```
200    226382
206       304
301       870
302       109
304     25269
400        26
401         5
403       194
404      8630
408        37
500         3
502        44
Name: status_response, dtype: int64
```

[20]:
```
error_2xx = (txt['status_response'].map(str)).str.extract(r'(^2)')
error_3xx = (txt['status_response'].map(str)).str.extract(r'(^3)')
error_4xx = (txt['status_response'].map(str)).str.extract(r'(^4)')
error_5xx = (txt['status_response'].map(str)).str.extract(r'(^5)')

print('codi tipus', error_2xx.value_counts())
print('codi tipus', error_3xx.value_counts())
print('codi tipus', error_4xx.value_counts())
print('codi tipus', error_5xx.value_counts())
```

```
codi tipus 2    226686
dtype: int64
codi tipus 3    26248
dtype: int64
codi tipus 4    8892
dtype: int64
codi tipus 5    47
dtype: int64
```

Així doncs, la gran majoria de sol·licituds han tingut una resposta satisfactòria. Veiem-ho en un gràfic:

[21]:
```
total = txt['status_response'].count()
pct2 = (((error_2xx.value_counts()).values)/total)*100
pct3 = (((error_3xx.value_counts()).values)/total)*100
pct4 = (((error_4xx.value_counts()).values)/total)*100
pct5 = (((error_5xx.value_counts()).values)/total)*100

plt.rcParams["figure.figsize"] = (20,5)

labels = ('$\mathbf{satisfactori}$', '$\mathbf{redirigit}$', '$\mathbf{error \␣
 ↪client}$', '$\mathbf{error \ servidor}$')
values = [float(pct2), float(pct3), float(pct4), float(pct5)]

plt.pie(values, labels = labels, labeldistance = 1.3,
        wedgeprops = {'linewidth' : 3, 'edgecolor' : 'white'},
       autopct='%1.0f%%', pctdistance=1.1)

plt.title('$\mathbf{Tipus \ d\'error}$', fontsize = 14)
plt.show()
```



1.0.10 bytes

[22]:
```
txt.bytes.describe().apply(lambda x: '%.5f' % x)
```

[22]:
```
count      219538.00000
mean        11827.71290
std        241363.02380
min             1.00000
25%          2510.00000
50%          6407.00000
75%          9674.00000
max      45564687.00000
Name: bytes, dtype: object
```

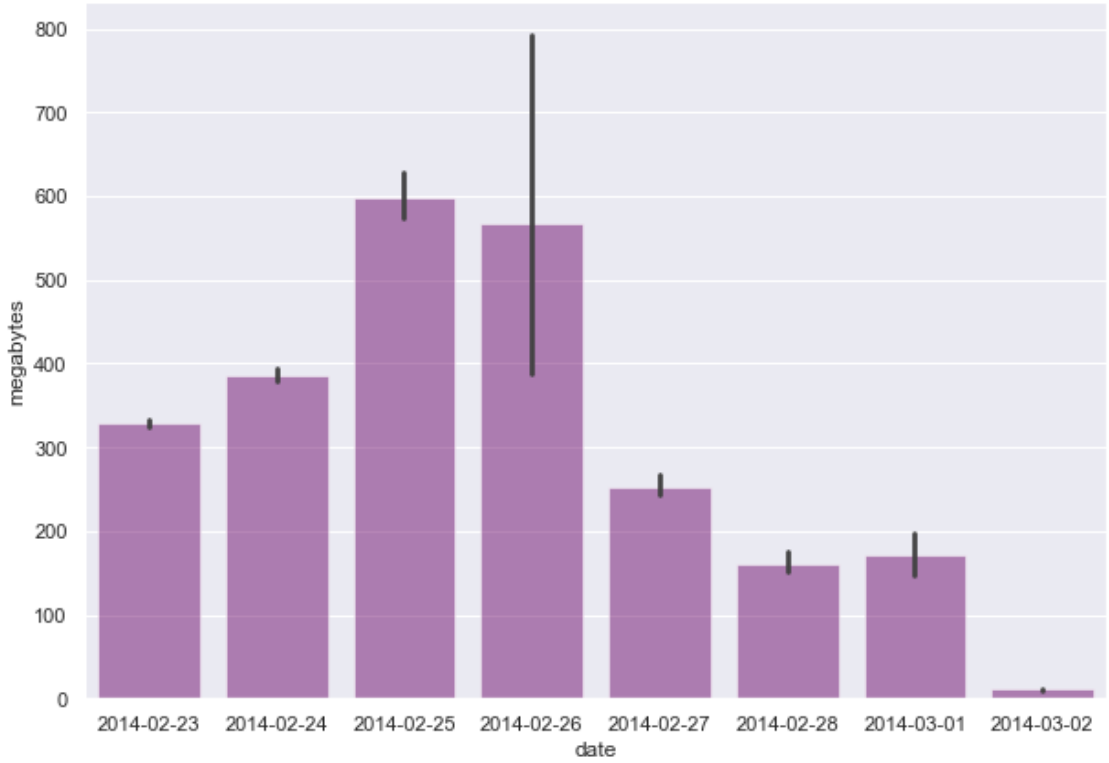
[23]:
```
txt.bytes.isna().sum()
```

[23]:
```
42335
```

[24]:
```
df_bytes = pd.DataFrame(columns = ['megabytes', 'date'])
df_bytes['megabytes'] = txt['bytes']/(2**10)/(2**10)

df_bytes['date'] = txt['date']

df_bytes.groupby('date').sum()
```

[24]:
```
             megabytes
date
2014-02-23  328.387009
2014-02-24  386.561088
2014-02-25  598.861448
2014-02-26  566.678618
2014-02-27  252.933409
2014-02-28  161.377036
2014-03-01  170.215617
2014-03-02   11.327437
```

[28]:
```
# En format gràfic:
plt.figure(figsize=(10, 7))
sns.barplot(x='date', y='megabytes', data = df_bytes,
            estimator=sum, color='purple', alpha = 0.5);
```



1.0.11 from_URL

les visites a la web d'akumenius han vingut referenciades des de 2.507 llocs webs diferents.

[29]:
```
txt.from_URL.unique().size
```

[29]: 2507

1.0.12 browser

[30]:
```
txt.browser.unique().size
```

[30]: 735

1.0.13 Exercici 3

Geolocalitza les IP's.

Com hem vist en l'exercici anterior, tenim 2.921 adreces IP diferents. Crearem una nova base de dades amb només les columnes relatives a l'adreça IP i quants cops han estat utilitzades

[31]:
```
df_IP = txt['IP_address'].value_counts().rename_axis('IP_address').
 ↪reset_index(name='quantitat')
df_IP
```

[31]:

	IP_address	quantitat
0	66.249.76.216	46382
1	80.28.221.123	14725
2	127.0.0.1	13892
3	217.125.71.222	5201
4	66.249.75.148	3558
…	…	…
2916	84.123.150.27	1
2917	217.130.150.116	1
2918	202.46.52.23	1
2919	216.151.130.170	1
2920	206.198.5.33	1

[2921 rows x 2 columns]

Atès que localhost sempre es tradueix en l'adreça IP 127.0.0.1, podem eliminar-les del nostre data set i definir la resta com a public IP.

[32]:
```
def localhost (ip):
    if ip == '127.0.0.1':
        return 'local host'
    else:
        return 'public IP'

df_IP['IP_type'] = df_IP['IP_address'].apply(localhost)
df_IP.head()
```

[32]:

	IP_address	quantitat	IP_type
0	66.249.76.216	46382	public IP
1	80.28.221.123	14725	public IP
2	127.0.0.1	13892	local host
3	217.125.71.222	5201	public IP
4	66.249.75.148	3558	public IP

[33]:
```
df_IP['IP_type'].value_counts()
```

[33]:
```
public IP     2920
local host       1
Name: IP_type, dtype: int64
```

com hem vist, només hi ha 1 sol registre de localhost, que eliminem del data set amb la següent funció i ens quedem només amb les IPs públiques:

[34]:
```
df_IP = df_IP[df_IP['IP_type'] == 'public IP']
df_IP.head()
```

[34]:

	IP_address	quantitat	IP_type
0	66.249.76.216	46382	public IP
1	80.28.221.123	14725	public IP

```
3   217.125.71.222       5201  public IP
4    66.249.75.148       3558  public IP
5  162.243.192.191       2927  public IP
```

[35]:
```
import requests

def check_for_None (text):
     if text is None:
        return ""
     else:
        return text

def get_geolocalization (ip):
    response = requests.get(("https://geolocation-db.com/json/
 ↪%s&position=true") % ip).json()
    latitude = check_for_None(response["latitude"])
    longitude = check_for_None(response["longitude"])
    city = check_for_None(response["city"])
    state = check_for_None(response["state"])
    country_name = check_for_None(response["country_name"])
    country_code = check_for_None(response["country_code"])
    return pd.Series([latitude, longitude, city, state, country_name,␣
 ↪country_code])

df_IP[['latitude', 'longitude', 'city', 'region', 'country_name',␣
 ↪'country_code']] = df_IP['IP_address'].apply(get_geolocalization)

df_IP.head()
```

```
/Users/deliagonzalezmata/opt/anaconda3/lib/python3.8/site-
packages/pandas/core/frame.py:3641: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  self[k1] = value[k2]
```

[35]:
```
        IP_address  quantitat    IP_type latitude longitude      city  \
0    66.249.76.216      46382  public IP   37.751   -97.822
1    80.28.221.123      14725  public IP  40.4165   -3.7026    Madrid
3   217.125.71.222       5201  public IP  37.3736   -6.0723  Bormujos
4    66.249.75.148       3558  public IP   37.751   -97.822
5  162.243.192.191       2927  public IP  40.7308  -73.9975  New York

     region   country_name country_code
0           United States           US
```

```
1    Madrid          Spain           ES
3   Seville          Spain           ES
4              United States           US
5  New York    United States           US
```

Per tal de construir un mapa del món necessitem 2 fitxers:

- un fitxer JSON que contingui dades geoespacials de les regions geogràfiques del món i les seves coordenades limítrofes: *world-countries.json*
- una matriu de dades amb 2 columnes: una pels valors clau i una altra amb el valor quantitatiu, que servirà per aplicar un color o un altre en el nostre mapa: *state-ip.csv*

Per enllaçar el nostre data set amb l'arxiu JSON de forma satisfactòria, el nom del país al data set ha de coincidir de manera exacta amb el nom del país de l'arxiu JSON. Per tant, analitzarem quins noms de país exactes conté l'arxiu JSON i modificarem els noms al data set en conseqüència.

[38]:
```
import json
world_countries = r'/Users/deliagonzalezmata/Downloads/world-countries.json'

# obrir l'arxiu json - el mètode json.load() retorna un diccionari
with open(world_countries) as world_file:
    world_json = json.load(world_file)

# amb un for loop pel diccionari, obtenim el nom dels països de l'arxiu json
denominations_json = []
for index in range(len(world_json['features'])):
    denominations_json.append(world_json['features'][index]['id'])

denominations_json[:20]
```

[38]:
```
['AFG',
 'AGO',
 'ALB',
 'ARE',
 'ARG',
 'ARM',
 'ATA',
 'ATF',
 'AUS',
 'AUT',
 'AZE',
 'BDI',
 'BEL',
 'BEN',
 'BFA',
 'BGD',
 'BGR',
 'BHS',
```

```
 'BIH',
 'BLR']
```

el codi de país que fa servir l'arxiu json és alpha-3, mentres que la nostra base de dades és alpha-2. Amb els codis (alpha-3) dels països que apareixen al json, modifiquem els codis del nostre data set per tal que siguin iguals

[44]:
```
df_country = pd.read_csv('/Users/deliagonzalezmata/Downloads/country-codes.
 ↪csv', sep=',', header = 0 )
df_country = df_country[['alpha-2', 'alpha-3']]
df_country.rename(columns={'alpha-2': 'country_code', 'alpha-3': 'country'},␣
 ↪inplace=True)
df_country.head()
```

[44]:

	country_code	country
0	AF	AFG
1	AX	ALA
2	AL	ALB
3	DZ	DZA
4	AS	ASM

[45]:
```
df_new = df_IP[['IP_address', 'country_code']]
df_new1 = df_new.groupby('country_code').count()
df_new1
```

[45]:

country_code	IP_address
	13
AD	3
AE	2
AR	23
AT	6
…	…
UA	18
US	682
UY	4
VE	5
VN	1

[65 rows x 1 columns]

[69]:
```
state_ip = pd.merge(df_new1, df_country, on='country_code', how='outer')
state_ip = state_ip.drop('country_code', 1)
state_ip.rename(columns={'country': 'country_code', 'IP_address': 'IP_count'},␣
 ↪inplace=True)
state_ip.IP_count.fillna(0, inplace=True)

state_ip.to_csv('/Users/deliagonzalezmata/Downloads/state-ip.csv')
```

```
state_ip
```

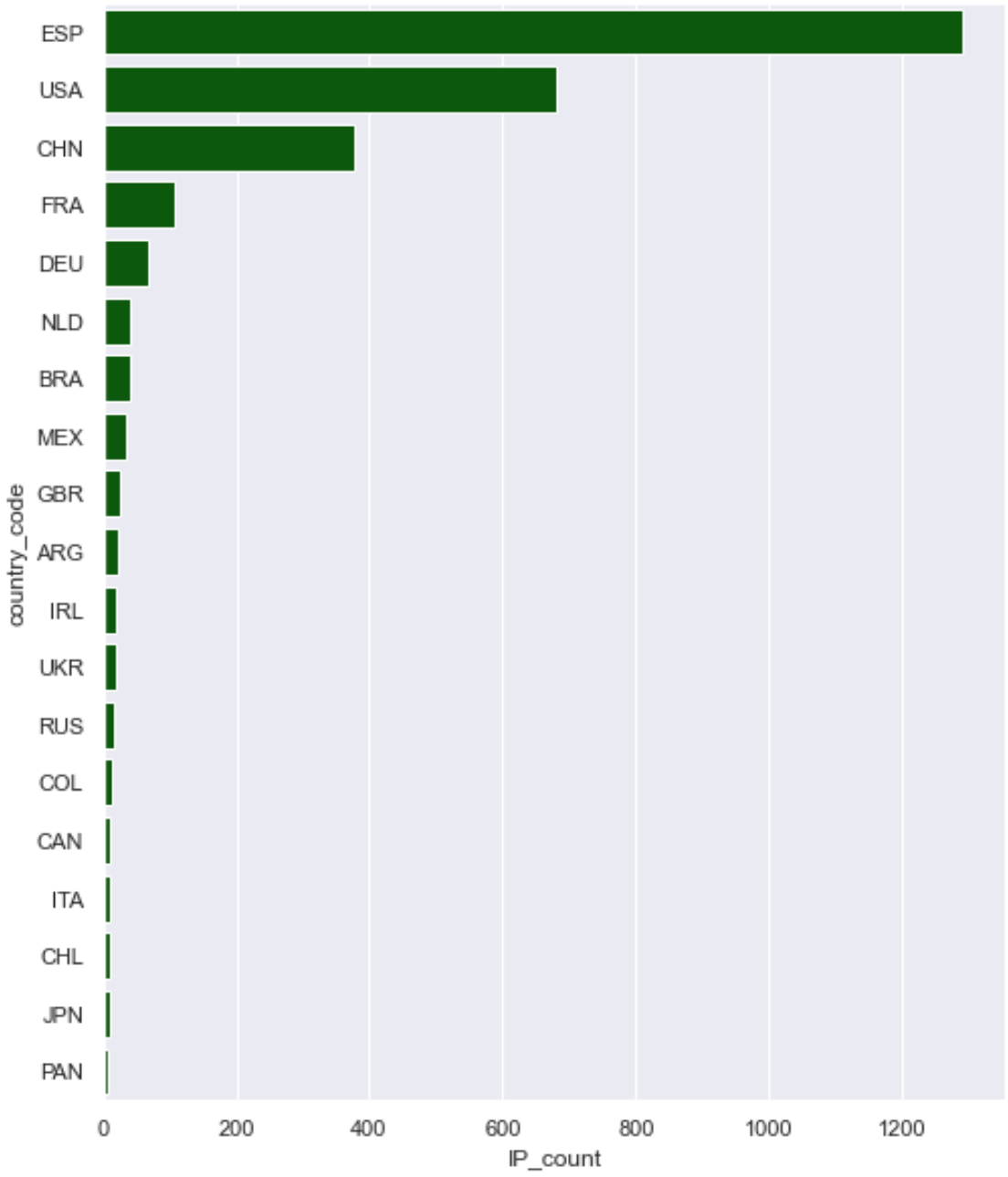
```
/var/folders/f1/1k69t1011n32zcq6vt7pt73c0000gn/T/ipykernel_1090/1127412655.py:2:
FutureWarning: In a future version of pandas all arguments of DataFrame.drop
except for the argument 'labels' will be keyword-only
  state_ip = state_ip.drop('country_code', 1)
```

[69]:

	IP_count	country_code
0	13.0	NaN
1	3.0	AND
2	2.0	ARE
3	23.0	ARG
4	6.0	AUT
..	…	…
246	0.0	WLF
247	0.0	ESH
248	0.0	YEM
249	0.0	ZMB
250	0.0	ZWE

[251 rows x 2 columns]

[58]:
```
sort = state_ip.sort_values("IP_count", ascending=False)[:20]

sns.set(rc = {'figure.figsize':(8,10)})
sns.barplot(data=sort, x= 'IP_count', y="country_code", color = 'darkgreen');
```



la majoria de les nostres IP es situen a Espanya, seguit d'Estats Units i la Xina.

[67]:
```
import folium

world_countries = f'/Users/deliagonzalezmata/Downloads/world-countries.json'
state_ip = state_ip #df amb columna 1 (key): country_code, col 2 (values):␣
 ↪IP_count
```

```
m = folium.Map(location=[40.4165, -3.7026], zoom_start=1.5)

choropleth = folium.Choropleth(
    geo_data=world_countries,
    name="choropleth",
    data=state_ip,
    columns=["country_code", "IP_count"],
    key_on="feature.id",#key in the json file that contains country code
    fill_color="YlGn",
    fill_opacity=0.7,
    line_opacity=0.2,
    legend_name="Quantitat de IPs",
    nan_fill_color="white",
    nan_fill_opacity=0.1).add_to(m)

# afegim etiquetes amb el nom de cada país
style_function = "font-size: 10px; font-weight: bold"
choropleth.geojson.add_child(
    folium.features.GeoJsonTooltip(['name'], style=style_function,␣
 ↪labels=False))

folium.LayerControl().add_to(m)


m
```

[67]: <folium.folium.Map at 0x7fdc1cf15c70>

[68]:
```
m.save('/Users/deliagonzalezmata/Downloads/choropleth-map-with-folium.html')
```