

## **PROYECTO EXPEDIA**



---

**Sofía Corral caballero**

**Javier Muñoz Fernández**

## ÍNDICE

1. Introducción
2. Importar nuestros datos
3. Mostrar columnas
4. Creación de nuevos DataFrame
5. Valores nulos
6. Borrar filas
7. Borrar columnas
8. Rellenar valores nulos
9. Añadir columnas nuevas
10. Correlación entre columnas
11. Gráficos
12. Api's
13. Web Scraping
14. Conclusiones

## 1. Introducción

El trabajo que hemos elegido es un proyecto que se encuentra en la plataforma de Kaggle. El objetivo de dicho proyecto es predecir qué grupo de hoteles va a reservar un determinado usuario.

Para ello, expedia hace visible una serie de datos donde nos muestran las diferentes características del viaje que ha reservado dicho usuario, características como el número de personas adultas que viajan o el país, al que pertenece dicho usuario entre otras.

Los datos que expedia nos proporcionan son una pequeña muestra de usuarios escogidos aleatoriamente. Dichos datos nos lo proporciona expedia en formato .csv.

A continuación, mostraremos una imagen en la que aparecerá la descripción que expedia pone para el concurso, junto con el link donde aparece el proyecto en la plataforma de Kaggle.

The screenshot shows the Kaggle competition page for 'Expedia Hotel Recommendations'. At the top, there's a logo for Expedia and the title 'Expedia Hotel Recommendations'. Below the title, it says 'Which hotel type will an Expedia customer book?' and '25,000 · 1,973 teams · 4 years ago'. A navigation bar at the top includes 'Overview' (which is underlined), 'Data', 'Notebooks', 'Discussion', 'Leaderboard', 'Rules', 'Team', 'My Submissions', and 'Late Submission' (which is highlighted in blue). The main content area has a section titled 'Overview' and another titled 'Description'. The 'Description' section contains text about planning a vacation and choosing between old standby hotels and new trendy ones. It also includes a row of seven icons representing different hotel types: a bed, a fork and knife, a calendar, a bathtub, a bell, a sun, and a magnifying glass. Below the icons, there's more text about Expedia's goal of providing personalized hotel recommendations and a note that the data is a random selection from Expedia.

Link: <https://www.kaggle.com/c/expedia-hotel-recommendations>

## 2. Importar nuestro datos

Una vez elegido el proyecto que queríamos llevar a cabo procedimos a descargarnos el dataset que expedia nos proporcionaba en formato .csv y a abrirlo en python.

Para poder extraer la información del dataset es necesario importar previamente una serie de librerías:

```
In [451]: import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
from pylab import *
```

Una vez importadas las librerías necesarias, procedemos a escribir el código necesario para que nos imprima nuestro dataset. En este caso, escribimos las cinco primeras filas de todo nuestro dataset:

```
In [452]: myData = pd.read_csv('./datos/train.csv')
myData.head()
```

	date_time	site_name	posa_continent	user_location_country	user_location_region	user_location_city	orig_destination_distance	user_id	is_mobile	is_pack
0	2014-08-11 07:46:59	2	3	66	348	48862	2234.2641	12	0	
1	2014-08-11 08:22:12	2	3	66	348	48862	2234.2641	12	0	
2	2014-08-11 08:24:33	2	3	66	348	48862	2234.2641	12	0	
3	2014-08-09 18:05:16	2	3	66	442	35390	913.1932	93	0	
4	2014-08-09 18:06:16	2	3	66	442	35390	913.6259	93	0	

5 rows × 24 columns

### 3. Mostrar columnas

Una vez cargado nuestro dataset, procedimos a entender el significado de los datos que nos proporcionaban. Lo más importante era entender el significado de cada fila y cada columna.

En el caso de las filas era bastante claro, cada fila pertenece a un usuario y a una visita determinada a la plataforma de expedia para reservar un viaje o simplemente para visitar la plataforma.

En el caso de las columnas es algo más complicado, ya que si imprimimos la información de nuestro dataset donde nos aparece el nombre de todas las columnas que tiene nuestro dataset, podemos observar como el nombre que aparece en alguna de las columnas no es muy característico.

```
In [6]: myData.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 243445 entries, 0 to 243444
Data columns (total 24 columns):
date_time           243445 non-null object
site_name            243445 non-null int64
posa_continent       243445 non-null int64
user_location_country 243445 non-null int64
user_location_region 243445 non-null int64
user_location_city   243445 non-null int64
orig_destination_distance 157252 non-null float64
user_id              243445 non-null int64
is_mobile             243445 non-null int64
is_package            243445 non-null int64
channel               243445 non-null int64
srch_ci                243218 non-null object
srch_co                243218 non-null object
srch_adults_cnt        243445 non-null int64
srch_children_cnt       243445 non-null int64
srch_rm_cnt              243444 non-null float64
srch_destination_id      243444 non-null float64
srch_destination_type_id 243444 non-null float64
is_booking              243444 non-null float64
cnt                     243444 non-null float64
hotel_continent          243444 non-null float64
hotel_country             243444 non-null float64
hotel_market              243444 non-null float64
hotel_cluster              243444 non-null float64
dtypes: float64(10), int64(11), object(3)
memory usage: 44.6+ MB
```

Por lo tanto, para poder usar cada columna era necesario entender su significado previamente. El significado de cada columna es el siguiente, significado que hemos encontrado en un apartado de la plataforma de kaggle y que hemos traducido:

- site\_name: Identificación del punto de venta de Expedia
- posa\_continent: ID del continente asociado
- user\_location\_country: ID del país donde está el cliente
- user\_location\_region: ID de la región donde está el cliente
- user\_location\_city: ID de la ciudad donde está el cliente
- orig\_destination\_distance: Distancia entre el cliente y el hotel en el momento de la búsqueda
- user\_id: ID del usuario
- is\_mobile: 1-cuando se conectó desde el móvil
- is\_package: 1-si la reserva se generó como parte de un paquete (con vuelo)
- channel: canal de marketing
- srch\_ci: cadena de fecha de registro
- srch\_co: cadena de fecha de pago
- srch\_adults\_cnt: número de adultos especificados en la habitación
- srch\_children\_cnt: número de niños (ocupación extra) especificados en la habitación
- srch\_rm\_cnt: número de habitaciones especificados en la búsqueda
- srch\_destination\_id: ID del destino donde se realizó la búsqueda del hotel
- srch\_destination\_type\_id: ID del tipo de destino
- hotel\_continent: continente del hotel
- hotel\_country: país del hotel
- hotel\_market: mercado del hotel
- is\_booking: 1- si es una reserva, 0- si es un click
- cnt: número de eventos similares en el contexto de la misma sesión de usuario
- hotel\_cluster: ID de un grupo de hoteles

#### 4. Creación de nuevos DataFrame

Para poder ver con mayor facilidad ciertas cosas o que a la hora de cargar la correlación, media... se hiciera con mayor facilidad, realizamos varios DataFrame donde solo aparecieran las columnas necesarias o que nos hiciera un determinado cálculo, los diferentes DataFrame que hemos creado son los siguientes:

- **Obtención del número de valores nulos/no nulos:**

Con este DataFrame hemos podido obtener la cantidad de números nulos/no nulos que hay con un determinado valor de una columna, así como su porcentaje y la suma total de los valores:

	user_location_country	nulos	ValoresNoNulos	ValoresTotales	%Nulos	%NoNulos
0		0	599	85	684	87.57
1		1	371	3382	3753	9.89
2		3	7017	0	7017	100.00
3		4	9	0	9	100.00
4		5	441	0	441	100.00
5		6	29	0	29	100.00
6		10	26	0	26	100.00
7		12	764	0	764	100.00
8		13	5	0	5	100.00
9		15	13	0	13	100.00
10		16	4	23	27	14.81
11		19	175	0	175	100.00
12		23	2189	0	2189	100.00
13		24	156	0	156	100.00

El código que hemos utilizado para realizar el DataFrame es el siguiente:

```
In [20]: #creamos dos funciones que nos cuenten los valores nulos y los no nulos
def count_nulos(series):
    return len([elem for elem in series if elem * 1 != elem]) # != para nulos, == para los no nulos
def count_NoNulos(series):
    return len([elem for elem in series if elem * 1 == elem])

#contamos los nulos que tiene cada valor de user_location_country en la columna orig_destination_distance,
#el valor user_location_country Ø tiene 4 valores nulos en la columna orig_destination_distance
dnull=myData[Mascara_1].groupby('user_location_country')[['orig_destination_distance']].apply(count_nulos).reset_index(name='nulos')
#contamos los no nulos que tiene cada valor de user_location_country en la columna orig_destination_distance,
#el valor user_location_country Ø tiene 4 valores no nulos en la columna orig_destination_distance
dNotNull=myData[Mascara_1].groupby('user_location_country')[['orig_destination_distance']].apply(count_NoNulos).reset_index(name='NoNulos')

#añadimos una columna para los ValoresNoNulos, los valores totales
dnull[['ValoresNoNulos']] = dnull[['nulos']] + dNotNull[['NoNulos']]

#añadimos otras dos columnas para el %NoNulos y el %nulos, y lo configuraremos para que se nos devuelvan solo dos decimales
resNull=100*dnull[['nulos']] / (dnull[['nulos']] + dNotNull[['NoNulos']])
resultNull=round(resNull,2)
dnull[['%nulos']] = resultNull
resNoNull=100*dNotNull[['NoNulos']] / (dnull[['nulos']] + dNotNull[['NoNulos']])
resultNoNull=round(resNoNull,2)
dnull[['%NoNulos']] = resultNoNull

#imprimimos el subdataset que acabamos de construir
dnull
```

#### - Obtención de la correlación:

Otro de los sitios en los que hemos usado un DataFrame es para calcular la correlación entre columnas, en este caso, para que nuestro programa cargara con mayor rapidez hemos realizado un DataFrame donde solo aparecen aquellas columnas que pensábamos que iban a tener correlación con la columna que queríamos llenar, es decir, de todas las columnas que tiene nuestro dataSet, hemos elegido únicamente aquellas que pensabamos que iban a poder tener una correlación elevada con la columna que queríamos comparar y así no tardaría tanto en cargar:

	user_location_country	user_location_region	user_location_city	orig_destination_distance	hotel_continent	hotel_country
3415	0	393	14187	NaN	6.0	17.0
3416	0	393	14187	NaN	6.0	17.0
3417	0	393	14187	NaN	6.0	17.0
3418	0	393	14187	NaN	3.0	104.0
3419	0	393	14187	NaN	6.0	17.0
3420	0	393	14187	NaN	6.0	17.0
3421	0	393	14187	NaN	6.0	17.0
3422	0	393	14187	NaN	3.0	182.0
3423	0	393	14187	NaN	3.0	182.0
3424	0	393	14187	NaN	3.0	182.0
3425	0	393	14187	NaN	3.0	182.0
3426	0	393	14187	NaN	3.0	182.0
3427	0	393	14187	NaN	3.0	182.0
3428	0	275	49723	NaN	3.0	182.0
3429	0	275	49723	NaN	3.0	182.0
3435	0	393	14187	NaN	3.0	171.0
3436	0	393	14187	NaN	3.0	171.0
3437	0	393	14187	NaN	3.0	171.0
3438	0	393	14187	NaN	3.0	104.0
3439	0	393	14187	NaN	3.0	171.0
3440	0	393	14187	NaN	3.0	171.0
3441	0	393	14187	NaN	3.0	171.0

El código que hemos usado es el siguiente, donde indicamos que columnas queremos que formen parte de nuestro nuevo DataFrame:

```
In [42]: df0 = pd.DataFrame({
    'user_location_country':myData[Mascara_0].iloc[:,3],
    'user_location_region':myData[Mascara_0].iloc[:,4],
    'user_location_city':myData[Mascara_0].iloc[:,5],
    'orig_destination_distance':myData[Mascara_0].iloc[:,6],
    'hotel_continent':myData[Mascara_0].iloc[:,18],
    'hotel_country':myData[Mascara_0].iloc[:,19],
})
df0
```

- Calcular el número de clientes que hay en cada país:

Otro DataFrame que hemos creado ha sido aquel que nos ordena de mayor a menor el número de clientes que hay de cada país, es decir, me calcula de cada país cuántos clientes hay y me lo ordena de mayor a menor, a parte de esto me calcula su porcentaje:

	ValoresDiferentes	SumaTodosValores	ArregloGrafico	%repeticiones
0	66	140891	1	57.87
1	205	21765	2	8.94
2	69	19661	3	8.08
3	3	7017	4	2.88
4	46	5739	5	2.36
5	1	3753	6	1.54
6	77	3644	7	1.50
7	133	3506	8	1.44
8	215	3250	9	1.34
9	119	2418	10	0.99
10	23	2189	11	0.90
11	70	1715	12	0.70
12	167	1622	13	0.67
13	194	1395	14	0.57
14	231	1131	15	0.46
15	80	1096	16	0.45
16	182	1062	17	0.44
17	195	893	18	0.37
18	179	859	19	0.35
19	154	829	20	0.34

El código que hemos usado es similar al que nos ayuda a calcular el número de valores nulos y no nulos que hay pero cambiando las columnas que queremos usar y alguna operación:

```
In [518]: #uento el numero de veces que se repite cada valor de ContentID
ValoresContentID=myData['user_location_country'].value_counts()
#como el resultado es una serie lo convierto en un dataframe, cambiando el nombre de las columnas de dicha serie
VID1 = ValoresContentID.to_frame().reset_index()
VID1['ValoresDiferentes']=VID1['index']
VID1['SumaTodosValores']=VID1['user_location_country']
VID1 = VID1.drop(['index'], axis=1)
VID1 = VID1.drop(['user_location_country'], axis=1)

VID1['ArregloGrafico'] = VID1.index+1
#añado una columna que me calcule el % de las repeticiones con respecto al total de valores, tomando ademas solo dos decimales
resRep=100*(VID1['SumaTodosValores'])/(VID1['SumaTodosValores'].sum())
resultRep=round(resRep,2)
VID1['%repeticiones']=resultRep
VID1.head(20)
```

Este código lo hemos usado para ordenar de mayor a menor el número de repeticiones de otras columnas:

- Columna 'is\_booking':

	ValoresDiferentes	SumaTodosValores	ArregloGrafico	%repeticiones
0	0.0	223012	1	91.61
1	1.0	20432	2	8.39

- Columna 'is\_mobile':

	ValoresDiferentes	SumaTodosValores	ArregloGrafico	%repeticiones
0	0	211134	1	86.73
1	1	32310	2	13.27

- Columna 'is\_package':

	ValoresDiferentes	SumaTodosValores	ArregloGrafico	%repeticiones
0	0	181359	1	74.5
1	1	62085	2	25.5

- Columna 'srch\_adults\_cnt':

	ValoresDiferentes	SumaTodosValores	ArregloGrafico	%repeticiones
0	2	160283	1	65.84
1	1	51194	2	21.03
2	4	13193	3	5.42
3	3	12507	4	5.14
4	6	2475	5	1.02
5	5	2091	6	0.86
6	0	693	7	0.28
7	8	606	8	0.25
8	7	314	9	0.13
9	9	88	10	0.04

- Columna 'srch\_children\_cnt':

	ValoresDiferentes	SumaTodosValores	ArregloGrafico	%repeticiones
0	0	192463	1	79.06
1	1	26876	2	11.04
2	2	19977	3	8.21
3	3	2999	4	1.23
4	4	946	5	0.39
5	5	73	6	0.03
6	6	64	7	0.03
7	7	27	8	0.01
8	9	10	9	0.00
9	8	9	10	0.00

## 5. Valores nulos

Una vez cargado nuestro dataset y una vez analizadas las columnas y filas que contenía, procedimos a descubrir si había valores nulos en algunas de nuestras columnas. Para ello utilizamos el siguiente código, el cual nos suma todos los valores nulos que hay en cada columna.

```
In [453]: myData.isnull().sum()
Out[453]: date_time          0
site_name           0
posa_continent      0
user_location_country 0
user_location_region 0
user_location_city    0
orig_destination_distance 86193
user_id              0
is_mobile            0
is_package           0
channel              0
srch_ci              227
srch_co              227
srch_adults_cnt      0
srch_children_cnt     0
srch_rm_cnt           1
srch_destination_id   1
srch_destination_type_id 1
is_booking            1
cnt                  1
hotel_continent       1
hotel_country         1
hotel_market          1
hotel_cluster          1
dtype: int64
```

Como podemos observar hay tres columnas en las que hay más de un valor nulo y nueve en las que únicamente hay un valor nulo, es decir, hay un total de 12 columnas en las que mínimo tienen un valor nulo.

## 6. Borrar Filas

Una vez descubiertos los valores nulos, decidimos descubrir a qué filas pertenecía el valor nulo de aquellas columnas que tenían un único valor nulo.

Imprimiendo los datos que tiene la columna ‘hotel\_cluster’, descubrimos que el valor nulo pertenecía a la fila 243440. Haciendo lo mismo para la columna ‘hotel\_market’, descubrimos que el valor nulo pertenecía a la misma fila (243440).

```
In [454]: myData.tail().hotel_cluster.dropna  
Out[454]: <bound method Series.dropna of 243440>      8.0  
          243441    20.0  
          243442    14.0  
          243443    35.0  
          243444    NaN  
Name: hotel_cluster, dtype: float64>
```

```
In [455]: myData.tail().hotel_market.dropna  
Out[455]: <bound method Series.dropna of 243440>      1447.0  
          243441    1447.0  
          243442    1447.0  
          243443    1447.0  
          243444    NaN  
Name: hotel_market, dtype: float64>
```

Por lo tanto, decidimos imprimir toda la fila para descubrir si el resto de columnas cuyo valor nulo era uno pertenecían a la misma fila:

```
In [456]: myData.iloc[243444,:]  
Out[456]: # ... srch_children_cnt srch_rm_cnt srch_destination_id srch_destination_type_id is_booking cnt hotel_continent hotel_country hotel_market hotel_cluster  
0 ... 0 NaN NaN NaN NaN NaN NaN NaN NaN NaN NaN
```

Tras descubrir que era cierto, decidimos borrar toda la fila, ya que la información que nos aportaba no era útil. Para ello utilizamos ‘drop’:

```
In [457]: myData = myData.drop([243444])
```

## 7. Borrar columnas

El siguiente paso que dimos para analizar los nulos e intentar quitarlos fue descubrir que hacíamos con las columnas ‘srch\_ci’ y ‘srch\_co’. Ambas columnas con 227 valores nulos.

El primer paso que hicimos fue descubrir que significaba cada columna y que información nos aportaba:

```
In [459]: myData.srch_ci.describe()
Out[459]: count      243217
unique      1063
top       2014-12-26
freq       1613
Name: srch_ci, dtype: object

In [460]: myData.srch_co.describe()
Out[460]: count      243217
unique      1063
top       2015-01-02
freq       1581
Name: srch_co, dtype: object
```

Con la información que nos aporta el método describe, podemos observar como el contenido de dicha columna son fechas, juntando esta información con el significado de ambas columnas:

- srch\_ci: cadena de fecha de registro.
- srch\_co: cadena de fecha de pago.

Podemos tomar la decisión de borrar ambas columnas, ya que la fecha del registro o del pago no nos serán útiles a la hora de tomar decisiones o de realizar cualquier gráfico.

Para borrar ambas columnas, utilizamos el mismo método que para borrar filas, es decir, el método ‘drop’, pero en este caso, en vez de decirle el número de fila que queremos borrar, le escribimos el nombre de las columnas que queremos borrar:

```
In [463]: myData = myData.drop(['srch_ci', 'srch_co'], axis=1)
```

## 8. Rellenar valores nulos

Una vez borradas las columnas ‘srch\_ci’ y ‘srch\_co’ con valores nulos y la fila 243440, actualizamos de nuevo el número de valores nulos que nos aparecen y a que columnas pertenecen:

```
In [464]: myData.isnull().sum()
Out[464]: date_time          0
site_name           0
posa_continent      0
user_location_country 0
user_location_region 0
user_location_city    0
orig_destination_distance 86192
user_id              0
is_mobile            0
is_package           0
channel              0
srch_adults_cnt      0
srch_children_cnt     0
srch_rm_cnt           0
srch_destination_id    0
srch_destination_type_id 0
is_booking            0
cnt                  0
hotel_continent       0
hotel_country          0
hotel_market            0
hotel_cluster           0
dtype: int64
```

Como podemos comprobar, tras los cambios, ya solo tenemos valores nulos en la columna ‘orig\_destination\_distance’. En este caso decidimos llenar los valores ya que pensamos que la columna era bastante importante y no podíamos borrarla. Y borrar todas las filas a las que pertenezca cada valor nulo hacía perder mucha información del resto de columnas que sí que tenían los valores rellenados. A la hora de llenar valores nulos hay muchas opciones, nosotros hemos usado tres diferentes que hacen que el valor que hemos rellenado tenga más o menos precisión:

## 8.1 Comparar con una columna y rellenar con la media

La primera opción que usamos para llenar los valores nulos fue la de compararlo con otra columna que tuviera relación con la columna que tenemos que llenar, nosotros elegimos la columna ‘user\_location\_country’.

Para ello imprimimos los valores únicos que tenía cada columna y de cada valor nulo fuimos calculando el número de valores nulos que tenía y el de valores llenados, calculando su valor máximo, el mínimo y la media.

```
In [16]: pd.unique(myData['user_location_country'].values.ravel())
Out[16]: array([ 66, 195, 69, 3, 55, 23, 46, 205, 194, 133, 182, 68, 215,
       103, 134, 1, 77, 202, 239, 80, 198, 85, 93, 28, 0, 70,
       119, 12, 148, 167, 75, 154, 231, 173, 141, 71, 24, 6, 190,
       209, 51, 49, 157, 162, 158, 5, 235, 63, 143, 54, 32, 62,
       29, 168, 52, 130, 214, 39, 50, 225, 27, 48, 179, 64, 221,
       91, 206, 82, 229, 191, 59, 219, 101, 19, 230, 142, 115, 166,
       208, 111, 217, 57, 47, 109, 26, 181, 218, 176, 174, 34, 233,
       105, 108, 92, 144, 192, 16, 114, 118, 104, 163, 74, 184, 150,
       178, 129, 125, 131, 117, 4, 58, 156, 30, 31, 65, 149, 126,
       67, 152, 228, 197, 123, 43, 83, 45, 13, 95, 15, 44, 139,
       10, 60, 146, 124, 76, 73, 137, 238, 99, 188, 224, 223, 155,
       72, 87, 122, 222, 196], dtype=int64)
```

```
In [17]: max(myData.user_location_country)
```

```
Out[17]: 239
```

```
In [18]: min(myData.user_location_country)
```

```
Out[18]: 0
```

	user_location_country	nulos	ValoresNoNulos	ValoresTotales	%Nulos	%NoNulos
0		0	599	85	684	87.57
1		1	371	3382	3753	9.89
2		3	7017	0	7017	100.00
3		4	9	0	9	100.00
4		5	441	0	441	100.00
5		6	29	0	29	100.00
6		10	26	0	26	100.00
7		12	764	0	764	100.00
8		13	5	0	5	100.00
9		15	13	0	13	100.00
10		16	4	23	27	14.81
11		19	175	0	175	100.00
12		23	2189	0	2189	100.00
13		24	156	0	156	100.00

```
In [21]: grouped_myData=myData[Mascara_1 ].groupby('user_location_country').agg({'orig_destination_distance':['min','max','mean']})
grouped_myData
```

	orig_destination_distance		
	min	max	mean
<b>user_location_country</b>			
0	0.6746	8594.8170	2805.382884
1	0.3227	10087.4340	1720.319671
3	NaN	NaN	NaN
4	NaN	NaN	NaN
5	NaN	NaN	NaN
6	NaN	NaN	NaN
10	NaN	NaN	NaN
12	NaN	NaN	NaN
13	NaN	NaN	NaN
15	NaN	NaN	NaN
16	0.1518	1623.7226	369.910548
19	NaN	NaN	NaN
23	NaN	NaN	NaN
24	NaN	NaN	NaN

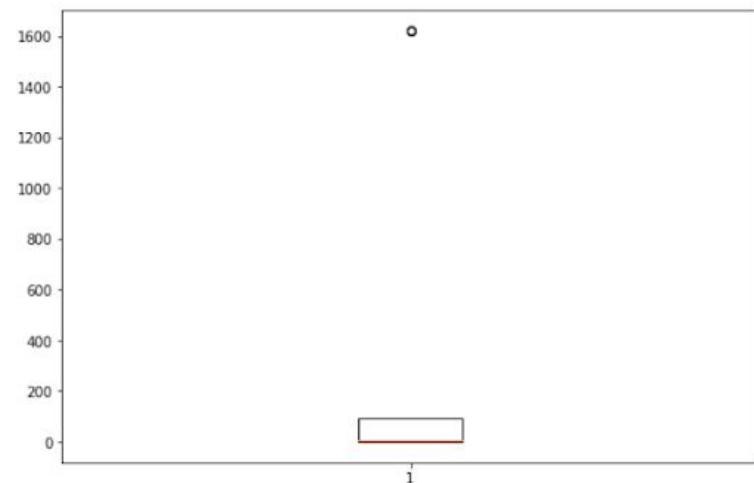
Con todos estos cálculos, descubrimos que no había mucha diferencia entre el valor mínimo y máximo cuando ‘user\_location\_country’ es 55. Así que decidimos rellenar el resto de valores nulos que quedaban con la media entre el valor mínimo y máximo:

	min	max	mean
<b>user_location_country</b>			
51	NaN	NaN	NaN
52	NaN	NaN	NaN
54	NaN	NaN	NaN
55	81.7849	83.6931	82.423800

```
In [26]: myData[Mascara_055]=myData[Mascara_055].fillna(value=82.423800)
myData[Mascara_055]
```

## 8.2 Comparar con una columna y llenar con los percentiles

Como el caso anterior sólo ocurría cuando ‘user\_location\_country’ es 55, decidimos buscar otra solución para llenar los valores más precisa, es decir, decidimos ver cómo se distribuían dichos valores. Para ello hicimos un diagrama de Caja-Bigote y calculamos los percentiles de cada valor diferente de la columna que tuviera valores no nulos, ya que si todos los valores que tiene un determinado valor de la columna, el cálculo de los percentiles, media... dará nulo.



```
In [30]: d16 = np.percentile(datos_1, 25)
d16
```

```
Out[30]: 1.54005
```

```
In [31]: e16 = np.percentile(datos_1, 50)
e16
```

```
Out[31]: 5.1583
```

```
In [32]: f16 = np.percentile(datos_1, 75)
f16
```

```
Out[32]: 90.40360000000001
```

Una vez visto cómo se distribuyen los valores decidimos llenar algunos con la media entre dos percentiles, ya que había una gran cantidad de valores con poca diferencia entre percentiles , es decir, en la imagen mostrada arriba, podemos observar como entre el percentil 25 y el 50 hay muy poca diferencia de valor y están situados los mismos valores que desde el percentil 50 hasta el percentil 75, cuyos valores están más dispersos.

Por lo tanto, decidimos llenar valores con la media entre los dos percentiles, en este caso, con la media entre el percentil 25 y el 50:

```
In [34]: mediaNueva = (d16+e16)/2  
mediaNueva  
Out[34]: 3.349175  
  
In [35]: myData[Mascara_016 ]= myData[Mascara_016 ].fillna(value=3.349175)  
myData[Mascara_016 ].tail()  
Out[35]:
```

### 8.3 Comparar con más de una columna y llenar con la media

En este caso, decidimos buscar una solución para llenar valores con mayor precisión. Para ello, decidimos compararlo con más de una columna, para ello elegimos las columnas que creímos que podían tener mayor relación y de ellas calculamos la correlación que tenía, una vez calculado todo, pudimos ver como la columna ‘hotel\_country’ tenía una correlación elevada con respecto a la columna ‘user\_location\_country’.

	user_location_country	user_location_region	user_location_city	orig_destination_distance	hotel_continent	hotel_country
user_location_country	Nan	Nan	Nan	Nan	Nan	Nan
user_location_region	Nan	1.000000	-0.072369	Nan	0.233834	0.396770
user_location_city	Nan	-0.072369	1.000000	Nan	0.106165	-0.084941
orig_destination_distance	Nan	Nan	Nan	1.000000	-0.533513	-0.602582
hotel_continent	Nan	0.233834	0.106165	-0.533513	1.000000	0.259268
hotel_country	Nan	0.396770	-0.084941	-0.602582	0.259268	1.000000

Cómo íbamos a usar una nueva columna era necesaria conocer la cantidad de valores únicos que tenía, para poder usar con mayor facilidad la columna.

```
In [45]: pd.unique(myData[Mascara_0 ]['hotel_country'].values.ravel())  
Out[45]: array([ 17., 104., 182., 171., 70., 50., 110., 168., 5., 68., 151.,  
48., 203., 36., 99., 204., 79., 135., 105., 107., 161., 132.])
```

Al descubrir que no había una gran cantidad de valores únicos, decidimos ir comparándolos uno a uno y calcular de cada uno su media y rellenar, ya que la correlación era bastante elevada, con la media.

Para ello primero hicimos una máscara donde aparecieran únicamente los valores que cumplieran con la condición, de dicha máscara calculamos la media y rellenamos con la solución que nos daba la media.

```
In [46]: Mascara_0 = myData.user_location_country == 0  
Mascara_50 = myData.hotel_country ==50  
myData[Mascara_0 & Mascara_50]  
  
Out[46]:  
date_time site_name posa_continent user_location_country user_location_region user_location_city orig_destination_distance user_id is_mobile  
3452 2013-12-11 19:58:34 22 2 0 393 14187 NaN 18798 0  
3453 2013-12-11 20:01:21 22 2 0 393 14187 NaN 18798 0  
3457 2014-02-03 13:30:19 22 2 0 393 14187 NaN 18798 0  
3462 2014-05-12 13:55:46 22 2 0 393 14187 NaN 18798 0  
3463 2014-05-12 13:57:21 22 2 0 393 14187 NaN 18798 0  
...  
  
In [47]: myData[Mascara_0 & Mascara_50]['orig_destination_distance'].mean()  
Out[47]: 7813.887669565221  
  
In [48]: myData[Mascara_0 & Mascara_50] = myData[Mascara_0 & Mascara_50].fillna(value=7813.887669565221)
```

Pero no todas las máscaras tenían valores rellenos y a la hora de calcular su media nos daba nulo, pese a que todos los valores que comprendía eran nulos. Para estos valores decidimos rellenarlos con la media general del resto de valores. Para ello calculamos la media general de todos los valores en los que la columna 'orig\_destination\_distance' fuera 0 sin tener en cuenta el valor de la columna 'hotel\_country' y rellenamos todos los valores que nos quedaban nulos porque no habíamos podido llenar anteriormente.

```
In [66]: myData[Mascara_0]['orig_destination_distance'].mean()  
Out[66]: 1997.0547941056013  
  
In [67]: myData[Mascara_0] = myData[Mascara_0].fillna(value=1997.0547941056013)
```

## 9. Añadir columnas nuevas

Una vez tenemos nuestro dataSet limpio y conocemos con claridad el significado de nuestras columnas, decidimos crear nuevas columnas para así poder sacar nuevas conclusiones y hacer más gráficas, las columnas que hemos creado han sido las siguientes:

- Columna date:

Esta columna nos muestra la fecha, es decir, el día, mes y año.

Para crear esta columna nos hemos basado en la columna 'date\_time', donde la información que nos mostraba dicha columna era el día, mes, año, hora, minuto y segundo.

Así pues en esta columna hemos querido almacenar la fecha exacta pero sin tener en cuenta las horas, minutos y segundos.

Hemos creado esta columna porque creímos que la información de la hora no era útil de cara a la realización de gráficas.

- Columna month-year:

Esta columna nos muestra el mes y año.

Para crear esta columna, al igual que la anterior nos hemos basado en la información que nos aportaba la columna 'date\_time'.

Hemos creado esta columna para poder ver la evolución de la empresa por meses y de una forma más detallada.

- Columna month:

Esta columna nos muestra el mes.

Para crear esta columna nos hemos basado en la columna 'date\_time'.

La creación de esta columna es bastante importante, ya que nos sirve para separar la fecha por trimestres o períodos y para comparar los meses de mayor o menor movimiento de la plataforma.

- Columna year:

Esta columna nos muestra el año, es decir 2013 o 2014, ya que nuestro dataSet únicamente tiene valores de esos dos años.

Para crear la columna al igual que todas las anteriores nos hemos basado en la columna que expedía nos ha proporcionado con nuestro dataSet, es decir, la columna 'date\_time'.

Hemos creado esta columna ya que así podemos comparar la evolución de la empresa en los diferentes años que nos muestran, es decir, en 2013 y 2014.

El código de todas las columnas anteriormente nombradas es el siguiente, dónde partiendo de los datos que expedia nos proporciona en la columna 'site\_name', borramos la parte que no queremos poner en cada una de las columnas, creando así columnas diferentes que podemos utilizar independientemente.

```
In [69]: myData['date'] = pd.to_datetime(myData['date_time'])
myData['date'] = myData['date'].dt.date
myData['month-year'] = pd.to_datetime(myData['date']).dt.to_period('M')
myData['month'] = pd.to_datetime(myData['date_time'])
myData['month'] = myData['month'].dt.month
myData['year'] = pd.to_datetime(myData['date']).dt.to_period('Y')
myData = myData.sort_values(by = 'month-year')
myData.head()
```

- Columna periodo:

En esta nueva columna hemos creado 4 valores correspondientes 1,2,3,4, cada valor corresponde a un determinado periodo. Por ejemplo el valor 1 corresponde con los meses de festividades navideñas, es decir con Diciembre y Enero. Más detalladamente:

- el valor 1 corresponde a la temporada de festividades navideñas es decir recoge la información únicamente de los meses de Enero y de Diciembre
- el valor 2 corresponde al periodo de tiempo en el que cae la semana santa( es decir las vacaciones de primavera) y por tanto la información de Marzo y de Abril.
- el valor 3 corresponde al periodo de vacaciones de verano es decir a los meses de Junio, Julio, Agosto y Septiembre.

- el valor 4 corresponde al resto de meses que, por decirlo de alguna manera, no tienen ningún tipo de festividad o vacaciones. Estos son los meses de Febrero, Mayo, Octubre y Noviembre

Para crear esta columna nos hemos basado en la información que nos aporta la nueva columna formada ‘month’, de esta manera hemos ido agrupando meses para formar periodos de tiempo “especiales”, que hemos citado anteriormente.

La creación de esta nueva columna es útil de cara a la creación de nuevos gráficos, a comparar los viajes y a conocer que grupo de hoteles que reserva cada cliente en cada determinada fecha.

El código que hemos utilizado es el siguiente, donde hemos tenido que enumerar todas las condiciones existentes por meses , a cada condición le hemos determinado un valor diferente y creado una nueva columna denominada ‘periodos’.

```
In [215]: condicionesPeriodo = [(myData.month == 1) & (myData.year == 2013),
                           (myData.month == 2) & (myData.year == 2013),
                           (myData.month == 3) & (myData.year == 2013),
                           (myData.month == 4) & (myData.year == 2013),
                           (myData.month == 5) & (myData.year == 2013),
                           (myData.month == 6) & (myData.year == 2013),
                           (myData.month == 7) & (myData.year == 2013),
                           (myData.month == 8) & (myData.year == 2013),
                           (myData.month == 9) & (myData.year == 2013),
                           (myData.month == 10) & (myData.year == 2013),
                           (myData.month == 11) & (myData.year == 2013),
                           (myData.month == 12) & (myData.year == 2013),
                           (myData.month == 1) & (myData.year == 2014),
                           (myData.month == 2) & (myData.year == 2014),
                           (myData.month == 3) & (myData.year == 2014),
                           (myData.month == 4) & (myData.year == 2014),
                           (myData.month == 5) & (myData.year == 2014),
                           (myData.month == 6) & (myData.year == 2014),
                           (myData.month == 7) & (myData.year == 2014),
                           (myData.month == 8) & (myData.year == 2014),
                           (myData.month == 9) & (myData.year == 2014),
                           (myData.month == 10) & (myData.year == 2014),
                           (myData.month == 11) & (myData.year == 2014),
                           (myData.month == 12) & (myData.year == 2014),
                           ]
eleccionesPeriodo = np.array((1,4,2,2,4,3,3,3,4,4,1,1,4,2,2,4,3,3,3,4,4,1), dtype="int8")
myData["periodo"] = np.select(condicionesPeriodo, eleccionesPeriodo, -1)
myData.head()
```

date	month-year	month	year	trimestre	Categoría	periodo
2013-01-14	2013-01	1	2013	1	9	1
2013-01-29	2013-01	1	2013	1	4	1
2013-01-29	2013-01	1	2013	1	4	1
2013-01-29	2013-01	1	2013	1	4	1
2013-01-29	2013-01	1	2013	1	4	1



- Columna categoría:

La última columna que hemos creado es la denominada ‘categoría’, esta columna tiene un total de 10 valores diferentes que van desde el número 1 hasta el 10, cada valor nos indica el número de adultos y niños que viajan:

- El número 1 corresponde a aquel viaje formado por 1 adulto sin importar el número de niños, excepto el formado por 1 adulto y 0 niños..
- El número 2 corresponde a aquel viaje formado por 0 niño y un número cualquiera de adultos..
- El número 3 corresponde a aquel viaje formado por 1 adulto y 0 niño.
- El número 4 corresponde a aquel viaje formado por 2 adultos y 0 niño.
- El número 5 corresponde a aquel viaje formado por 2 adultos y 1,2 o 3 niños.
- El número 6 corresponde a aquel viaje formado por 5,6,7,8 o 9 adultos y sin importar el número de niños.
- El número 7 corresponde a aquel viaje formado por 4,5,6,7,8,9 niño y sin tener en cuenta el número de adultos.
- El número 8 corresponde a aquel viaje formado por 1 adulto y 1,2 o 3 niños.
- El número 9 corresponde a aquel viaje formado por 3 adultos y 1,2 o 3 niños.

- El número 10 corresponde a aquel viaje formado por 4 adultos y 1,2 o 3 niños.

La creación de esta nueva tabla es bastante útil de cara a la finalidad que expedía nos pide, es decir, para determinar a qué grupo de hoteles va a pertenecer cada cliente, es muy importante conocer el número de personas que viajan y si son adultos o niños, con esta nueva columna creada va a ser más fácil conocer ese dato.

El código usado para la creación de la columna es bastante extenso, ya que era necesario indicarle todas las combinaciones existentes que hay de adultos y niños, es decir, había que escribirle todas las condiciones de arriba y a cada una de ellas determinarle un valor concreto. Una vez creada la condición e indicado la elección de valores que hemos tomado, procedemos a crear la propia columna, para ello es necesario ponerle un nombre, en este caso el nombre de la columna será ‘categoría’ y una vez creado el nombre simplemente hay que indicarle que las condiciones creadas anteriormente y la elección de valores es para esta columna.

```
In [141]: condiciones = [ (myData.srch_adults_cnt == 1) & (myData.srch_children_cnt == 0),
                      (myData.srch_adults_cnt == 2) & (myData.srch_children_cnt == 0),
                      (myData.srch_adults_cnt == 0),
                      (myData.srch_children_cnt == 0),
                      #(myData.srch_adults_cnt == 1) & (myData.srch_children_cnt == 0),
                      #(myData.srch_adults_cnt == 2) & (myData.srch_children_cnt == 0),
                      (myData.srch_adults_cnt == 2) & (myData.srch_children_cnt == 1),
                      (myData.srch_adults_cnt == 2) & (myData.srch_children_cnt == 2),
                      (myData.srch_adults_cnt == 2) & (myData.srch_children_cnt == 3),
                      (myData.srch_adults_cnt == 5),
                      (myData.srch_adults_cnt == 6),
                      (myData.srch_adults_cnt == 7),
                      (myData.srch_adults_cnt == 8),
                      (myData.srch_adults_cnt == 9),
                      (myData.srch_children_cnt == 4),
                      (myData.srch_children_cnt == 5),
                      (myData.srch_children_cnt == 6),
                      (myData.srch_children_cnt == 7),
                      (myData.srch_children_cnt == 8),
                      (myData.srch_children_cnt == 9),
                      (myData.srch_adults_cnt == 1) & (myData.srch_children_cnt == 1),
                      (myData.srch_adults_cnt == 1) & (myData.srch_children_cnt == 2),
                      (myData.srch_adults_cnt == 1) & (myData.srch_children_cnt == 3),
                      (myData.srch_adults_cnt == 3) & (myData.srch_children_cnt == 1),
                      (myData.srch_adults_cnt == 3) & (myData.srch_children_cnt == 2),
                      (myData.srch_adults_cnt == 3) & (myData.srch_children_cnt == 3),
                      (myData.srch_adults_cnt == 4) & (myData.srch_children_cnt == 1),
                      (myData.srch_adults_cnt == 4) & (myData.srch_children_cnt == 2),
                      (myData.srch_adults_cnt == 4) & (myData.srch_children_cnt == 3),]

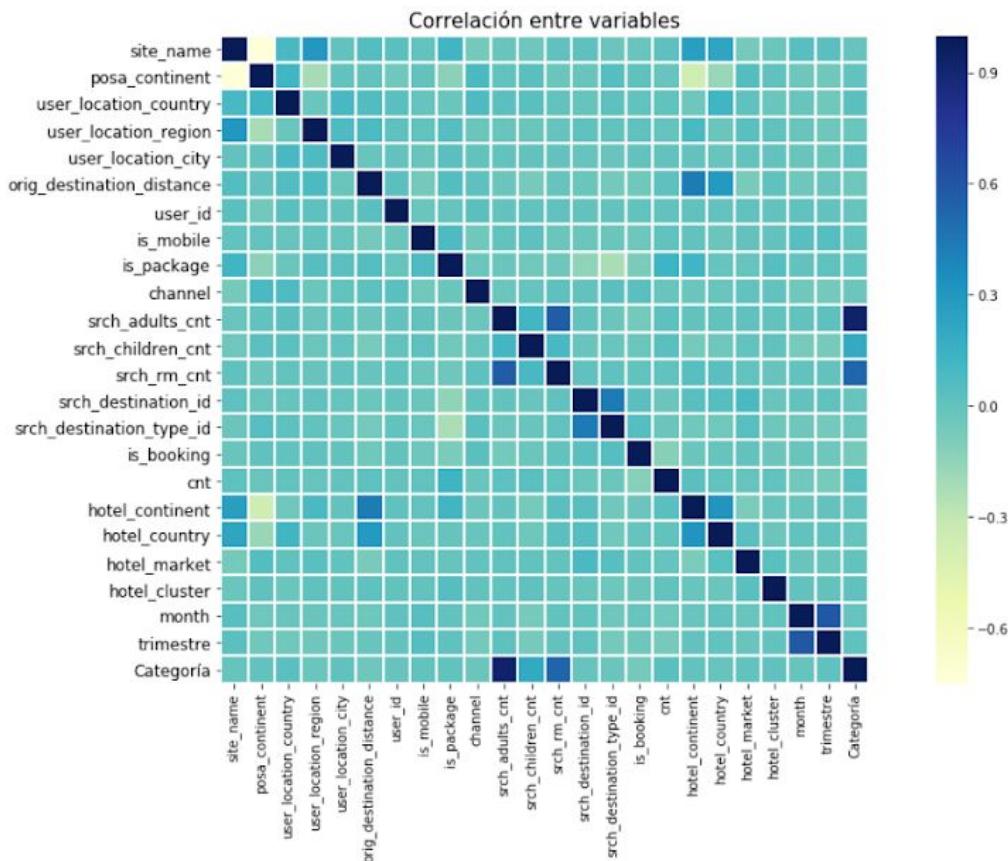
elecciones = np.array((3,4,1,2,5,5,5,6,6,6,6,7,7,7,7,8,8,8,9,9,9,10,10,10), dtype="int8")
myData["Categoría"] = np.select(condiciones, elecciones, 0)
myData.head()
```

date	month-year	month	year	trimestre	Categoria
2013-01-14	2013-01	1	2013	0	25
2013-01-29	2013-01	1	2013	0	15
2013-01-29	2013-01	1	2013	0	15
2013-01-29	2013-01	1	2013	0	15
2013-01-29	2013-01	1	2013	0	15

## 10. Correlación entre columnas

- Correlación del dataSet:

Con el cálculo de dicha correlación obtenemos la relación que hay entre todas las columnas que hay en nuestro dataSet y entre todas las filas, sin tener en cuenta ningún otro criterio. El gráfico obtenido es el siguiente:



Con este resultado podemos observar como entre las columnas 'site\_name' y 'posa\_continent' es donde se obtiene el valor más elevado de correlación.

- Correlación del DataFrame creado:

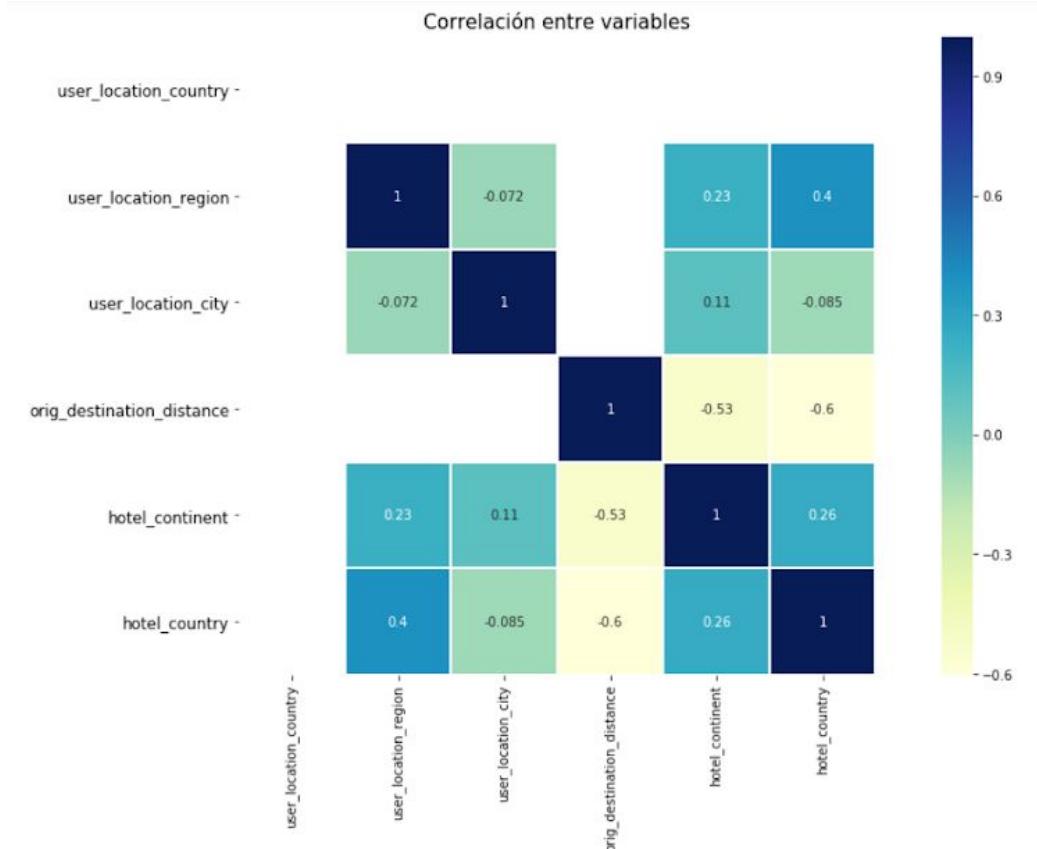
Con el cálculo de la correlación de este DataFrame, obteníamos, de una serie de columnas seleccionadas previamente cual de ellas tenía mayor relación con una columna determinada.

Este cálculo nos ayuda a la hora de llenar valores nulos, saber con qué columnas tenemos que comparar los resultados.

El DataFrame creado contiene las siguientes columnas:

Out[42]:	user_location_country	user_location_region	user_location_city	orig_destination_distance	hotel_continent	hotel_country
3415	0	393	14187	NaN	6.0	17.0
3416	0	393	14187	NaN	6.0	17.0
3417	0	393	14187	NaN	6.0	17.0
3418	0	393	14187	NaN	3.0	104.0
3419	0	393	14187	NaN	6.0	17.0
3420	0	393	14187	NaN	6.0	17.0
3421	0	393	14187	NaN	6.0	17.0
3422	0	393	14187	NaN	3.0	182.0
3423	0	393	14187	NaN	3.0	182.0
3424	0	393	14187	NaN	3.0	182.0
3425	0	393	14187	NaN	3.0	182.0
3426	0	393	14187	NaN	3.0	182.0

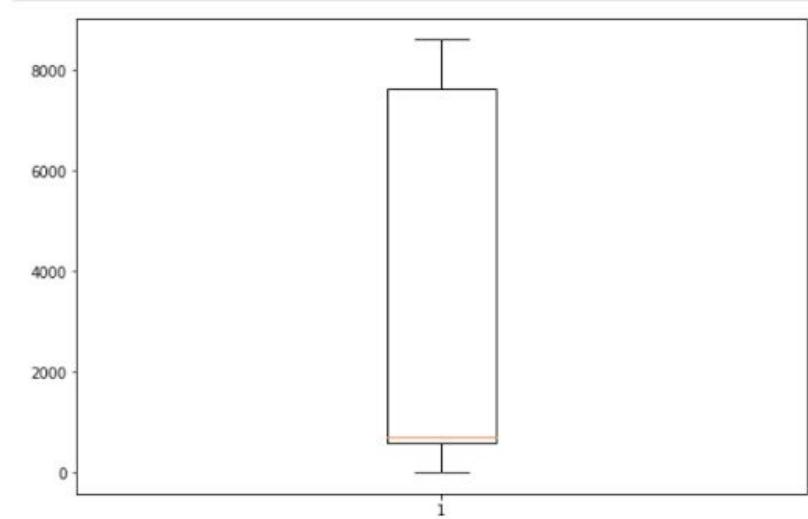
Por lo tanto, el resultado de la gráfica que indica la correlación es el siguiente:



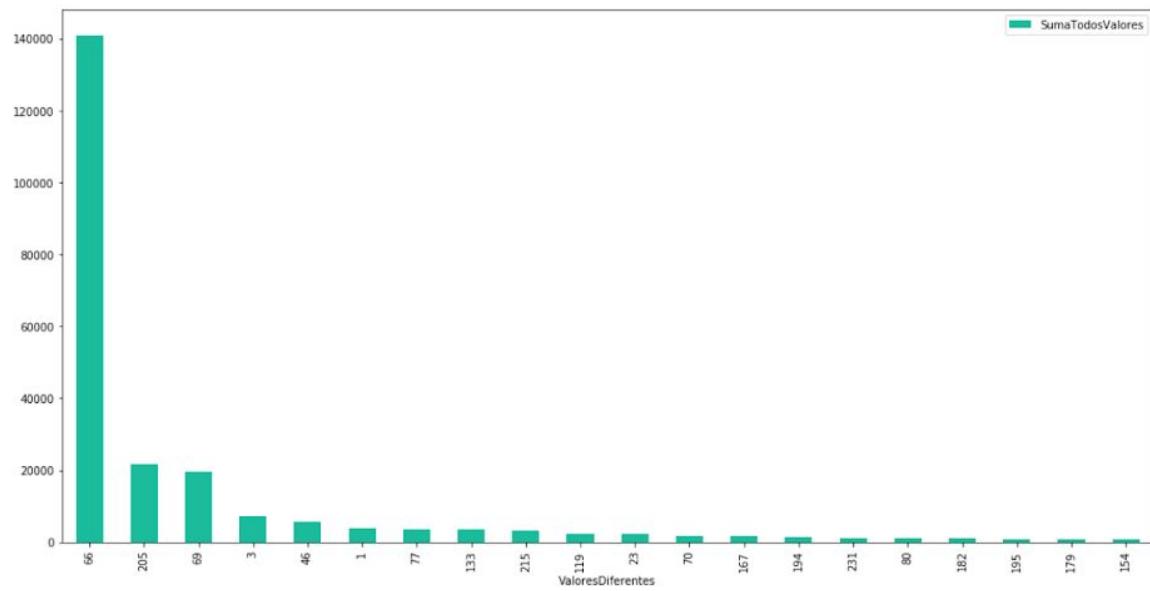
Como podemos observar, donde existe una mayor correlación es entre la columnas ‘hotel\_country’ y ‘orig\_destination\_distance’.

## 11. Gráficos

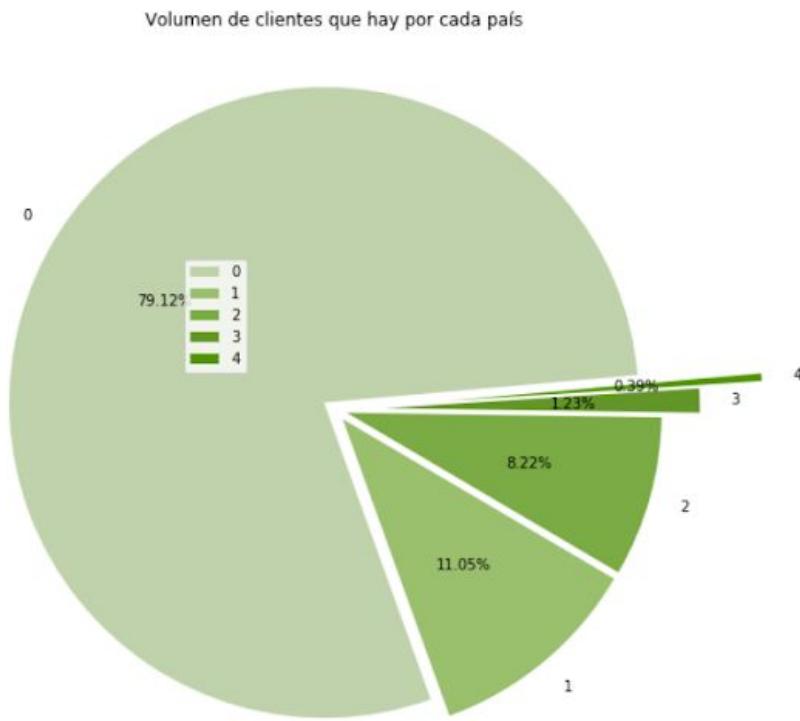
Una parte fundamental de nuestro trabajo era la creación de gráficos. Ya que el crear distintos tipos de gráficos nos han servido para poder llenar nulos de una manera lógica y viable, para poder hacer esto creamos gráficos de caja y bigotes que básicamente nos contaban como de dispersos estaban nuestros datos, es decir si nuestros datos se agrupaban entre un intervalo muy pequeño de valores o no. Así, podemos llenar los nulos con la media de ese intervalo.



Otros gráficos muy importantes en nuestro trabajo han sido los diagramas de barras y de líneas y los gráficos circulares, con estos gráficos hemos contado el número de veces que se repetían los diferentes valores de una columna y con ello hemos plasmado la información en gráficos para que en una primera vista podamos ver la distribución de los valores de una columna.

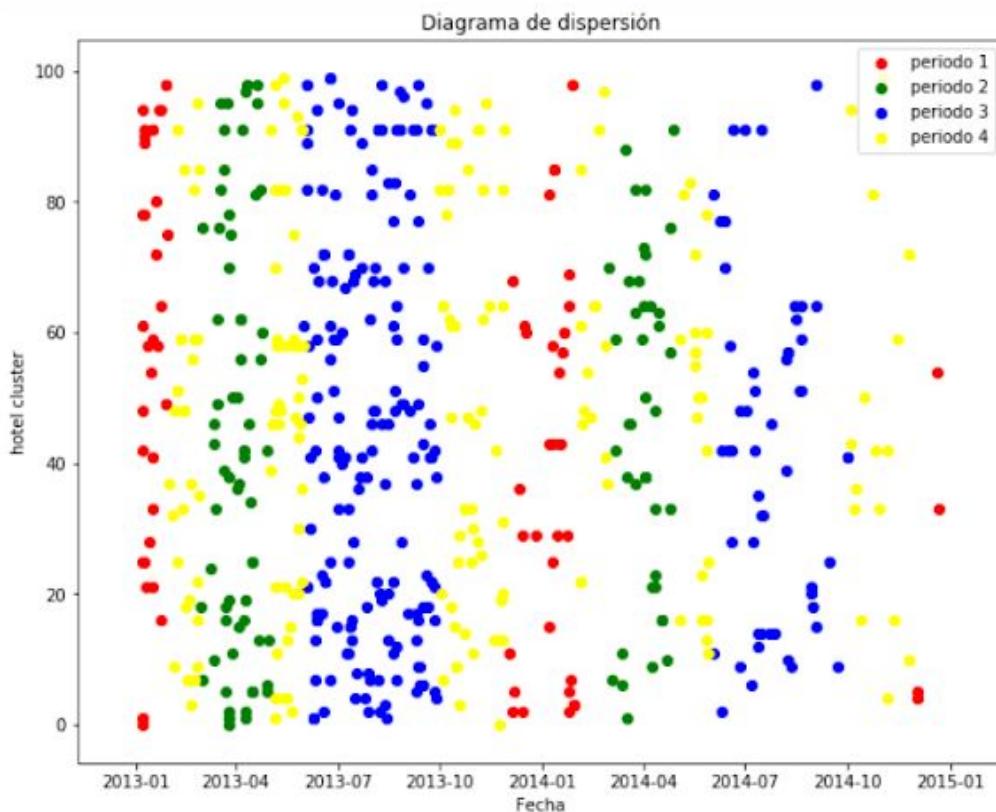


Por ejemplo, con este gráfico vemos a simple vista que el país que más reservas realiza es el país 66, de la misma manera vemos de un vistazo gracias al gráfico los países que más reservas realizan.



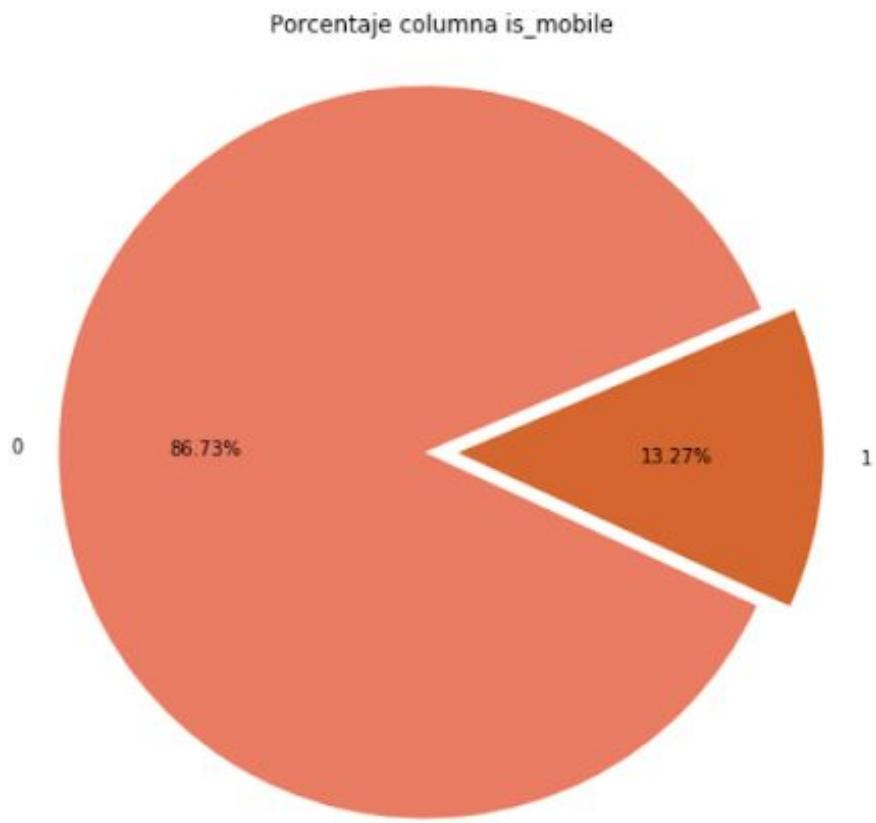
Del mismo modo los gráficos circulares también nos ayudan a ver la distribución de nuestros datos.

También intentamos darle una vuelta más a este tipo de gráficos y por ello creamos un gráfico de dispersión que mide hasta tres variables



Este gráfico nos da más información que los anteriores ya que nos dice cómo están distribuidos los datos pero además podemos distinguir la distribución de 4 distintos tipos de datos en un mismo gráfico, gracias a la inclusión de colores. Cada color corresponde a un determinado tipo de dato.

Lo más importante de la creación de los gráficos es que además de ayudarnos a predecir y rellenar valores nulos, también nos han aportado muchísima información de nuestros datos que no solo nos ayudan a sacar conclusiones de los datos, sino que además nos ayudan a fundamentar nuestras conclusiones y hacen que sea más fácil de entender lo que estamos explicando.



Por ejemplo con el gráfico circular de la columna 'is\_mobile', podemos sacar que es más interesante que nos centremos en los usuarios de ordenadores que en los de móviles, ya que como se ve en el gráfico la amplia mayoría de nuestros usuarios usan ordenador.

## 12. Api's

Para añadir un Api a nuestro trabajo, primero tuvimos que inscribirnos, en este caso, en Google que es la Api que vamos a usar.

Una vez inscritos rellenamos todo lo necesario para la obtención de una Api, en este caso, con geolocalización.

Tras tener la api y la contraseña que el propio Google nos proporciona, podíamos pasar a escribir en Python.

El primer paso que hicimos fue importar aquellas librerías que íbamos a utilizar:

```
In [589]: import urllib  
import urllib.request  
import json
```

En el siguiente paso, ya teníamos que usar la contraseña e indicar que era lo que queríamos obtener, en nuestro caso como nuestro trabajo es sobre expedía, decidimos buscar todos los hoteles que estaban a un cierto radio (100.000) de una determinada ubicación indicada por nosotros, en nuestro caso la ubicación seleccionada fue madrid (40.4378698,-3.8196207).

```
In [590]: googleGeocodeUrl = 'https://maps.googleapis.com/maps/api/js?key=AIzaSyAlCwmNlbmTcwmmri6Tv_lt74eBMs5Qpt4s&callback=initMap'  
termino = 'hotel'  
ubicacion= "&location=40.4378698,-3.8196207&radius100000"  
APIKEY = '&key = '+'AIzaSyAlCwmNlbmTcwmmri6Tv_lt74eBMs5Qpt4s'
```

El siguiente paso era crear el url, para ello unimos todo lo escrito en el código anterior, es decir el URL de la geolocalización de google, el término que queríamos buscar (en nuestro caso el término es 'hotel'), la ubicación y la APIKEY.

Por lo tanto, nuestro url queda de la siguiente manera:

```
In [591]: url = googleGeocodeUrl+termino+ubicacion+APIKEY  
print(url)  
  
https://maps.googleapis.com/maps/api/js?key=AIzaSyAlCwmNlbmTcwmmri6Tv_lt74eBMs5Qpt4s&callback=initMap&hotel&location=40.4378698,-3.8196207&radius100000&key = AIzaSyAlCwmNlbmTcwmmri6Tv_lt74eBMs5Qpt4s
```

Una vez obtenido el URL, si hacemos click en él, se abre una nueva página donde tenemos toda la información que buscábamos:

El URL es el siguiente:

[https://maps.googleapis.com/maps/api/js?key=AIzaSyAIcwmNlBmTcwMrI6Tv\\_lt74eBMq5Qpt4s&callback=initMapHotel&location=40.4378698,-3.8196207%radius100000&key = AIzaSyAIcwmNlBmTcwMrI6Tv\\_lt74eBMq5Qpt4s](https://maps.googleapis.com/maps/api/js?key=AIzaSyAIcwmNlBmTcwMrI6Tv_lt74eBMq5Qpt4s&callback=initMapHotel&location=40.4378698,-3.8196207%radius100000&key = AIzaSyAIcwmNlBmTcwMrI6Tv_lt74eBMq5Qpt4s)

Otro paso que hicimos fue pasar la información que nos daban mediante un URL a nuestro propio notebook, en nuestro caso, lo hicimos de la página oficial de Expedia. Para ello era necesario previamente la importación de una nueva librería:

```
In [592]: import requests
```

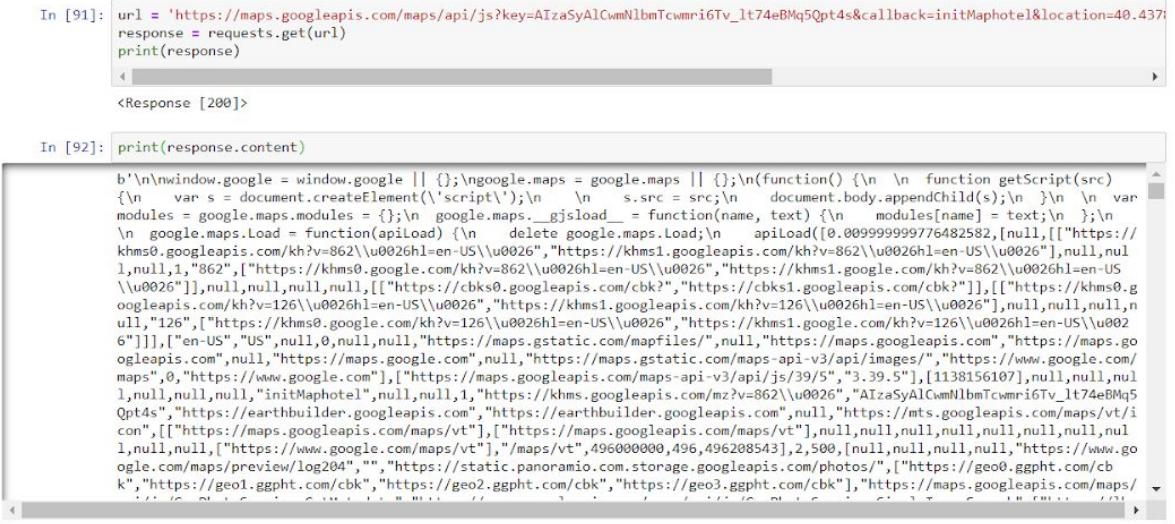
Con todas las librerías necesarias importadas, procedimos a, mediante el código URL que habíamos formado previamente pasar la información a nuestro notebook de Python.

El código es el siguiente:

```
In [91]: url = 'https://maps.googleapis.com/maps/api/js?key=AIzaSyAlCwmNlbmTcwri6Tv_lt74eBMs5Qpt4s&callback=initMap&hotel&location=40.437'
response = requests.get(url)
print(response)

<Response [200]>

In [92]: print(response.content)
```



## 13. Web Scraping

Nuestra idea de web Scraping fue obtener algún tipo de información dentro de la página de Expedia. Por ello pensamos en coger el Banner de la página, pero debido a su gran amplitud, obtuvimos una parte concreta del banner.

En primer lugar, una vez abierto un terminal e instalado scrappy creamos un nuevo proyecto que guardará la información sustraída con scrapy;

```
PS C:\Users\javie> scrapy startproject infoExpedia
New Scrapy project 'infoExpedia', using template directory 'c:\users\javie\anaconda3\lib\site-
project', created in:
C:\Users\javie\infoExpedia

You can start your first spider with:
cd infoExpedia
scrapy genspider example example.com
```

Ahora creamos un spider copiando la url de la web de donde sacaremos el scrapy:

```
PS C:\Users\javie\infoExpedia\infoExpedia> scrapy genspider SpiderExpedia https://www.expedia.es/?semcid=ES.B.GOOGLE.BT-c-E
Created spider 'SpiderExpedia' using template 'basic' in module:
infoExpedia.spiders.SpiderExpedia
```

```
Directorio: C:\Users\javie\infoExpedia\infoExpedia\spiders
```

Mode	LastWriteTime	Length	Name
----	-----	-----	-----
d-----	11/01/2020 13:12		__pycache__
-a---	11/01/2020 13:19	309	SpiderExpedia.py
-a---	12/12/2019 17:57	161	__init__.py

Lo siguiente es hacer ciertas modificaciones en el notebook del proyecto infoExpedia para mostrar una salida por pantalla:

```

# -*- coding: utf-8 -*-
import scrapy

class SpidercitasSpider(scrapy.Spider):
    name = 'SpiderCitas'
    allowed_domains = ['https://www.expedia.es/?semcid=ES.B.GOOGLE.BT-c-E']
    start_urls = ['https://www.expedia.es/?semcid=ES.B.GOOGLE.BT-c-E']

    def parse(self, response):
        page = response.url.split("/")[-2]
        self.log("realizando scrapping a la página %s" % page)

```

El siguiente paso es lanzar nuestro spider:

```

PS C:\Users\javiel\infoExpedia\infoExpedia\spiders> scrapy crawl SpiderExpedia
2020-01-11 13:36:28 [scrapy.utils.log] INFO: Scrapy 1.8.0 started (bot: infoExpedia)
2020-01-11 13:36:28 [scrapy.utils.log] INFO: Versions: lxml 4.3.4.0, libxml2 2.9.9, cssselect
21.0, Twisted 19.10.0, Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64
1.1.1c 28 May 2019), cryptography 2.7, Platform Windows-10-10.0.18362-SP0

```

```

PS C:\Users\javiel\infoExpedia\infoExpedia\spiders> scrapy shell https://www.expedia.es/?semcid=ES.B.GOOGLE.BT-c-E
2020-01-11 13:39:36 [scrapy.utils.log] INFO: Scrapy 1.8.0 started (bot: infoExpedia)
2020-01-11 13:39:36 [scrapy.utils.log] INFO: Versions: lxml 4.3.4.0, libxml2 2.9.9, cassele
21.0, Twisted 19.10.0, Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)], pyOpenSSL 19.0.0 (OpenSSL
1.1.1c 28 May 2019), cryptography 2.7, Platform Windows-10-10.0.18362-sp0
2020-01-11 13:39:36 [scrapy.crawler] INFO: Overridden settings: {'BOT_NAME': 'infoExpedia', 'DUPEFILTER_CLASS': 'scrapy.dupe
filters.BaseDupeFilter', 'LOGSTADS_INTERVAL': 0, 'NEWSPIDER_MODULE': 'infoExpedia.spiders', 'ROBOTSTXT_OBEY': True, 'SPIDE
R_MODULES': ['infoExpedia.spiders']}

```

(Con este paso podemos extraer el título de nuestra página web.)

```

In [2]: response.xpath('//title')
Out[2]: [<Selector xpath='//title' data='<title>¿Eres un robot?</title>'>]

In [3]: response.xpath('//title/text()').extract_first()
Out[3]: '¿Eres un robot?'

```

**El siguiente paso** fue coger la información del banner de la web para ello pusimos en la terminal la siguiente orden: `response.xpath(['@role="banner"]')`

Almacenándola en una variable así: `banner= response.xpath(['@role="banner"]')`.

Nótese que guardamos el banner extrayéndola de aquella parte encontrada en las herramientas para desarrolladores(inspeccionando la página).

```
er.html -->
<!-- GLOBAL_CONTROLS_HEADER -->
► <div class="skip-nav" id="skipNav">...</div>
► <div role="banner" class="site_9 es_ES b2p-yellow-header ">...</div>
  <div id="skip-nav-anchor" class="visuallyhidden">Inicio de la sección
  principal</div>
  <div data-gss-modal-version="2.2.11 static" style="display: none;"></div>
► <div id="gss-modal-main-container">...</div>
</div>
<div id="airattach-container" class="failed-to-load nocontent"></div>
► <meso-native-marquee fallbackImage="https://images.trvl-media.com/media/
  content/expus/graphics/launch/home/default_desktop_expedia.jpg" uci="NMO">...
```

En ese momento ya tenemos todo el banner almacenado y para enseñarlo por pantalla ejecutamos la siguiente orden:

`banner.xpath('[@role="banner"]').extract_first()`, el problema de esto fue que sacamos por pantalla absolutamente TODA la información del banner y ocupó TODA la terminal, por lo que no pudimos hacer capturas de pantalla de todo lo que hemos dicho anteriormente, salió algo así:

Asique viendo que había sacado TODA la información del banner el siguiente paso es acotar la información del banner para quedarse con algo concreto. Por ello seleccionamos la información que te dice si hay algo en tu cesta o no. Así pues, de nuestra variable Banner creada que almacenaba todo el banner seleccionamos eso:

```
In [8]: banner.xpath('./li[@class="shopping-cart-link hidden"]') extract first()
Out[8]: <li class="shopping-cart-link hidden" data-localized-content=\'{ "empty": "No hay nada en la cesta. Haz clic para acceder al contenido de la cesta.", "single": "Tienes un producto en tu cesta. Haz clic para acceder al contenido de la cesta.", "multiple": "Tienes {{numItems}} productos en tu cesta. Haz clic para acceder al contenido de la cesta.", "href": "/cart-data?rfrr=HP.SHOPPINGCART.HEADER.ENG"}\'}>\n<a id="header-cart-sm" data-tid="header-cart" target=
```

Y como se puede ver nos devuelve la información que queríamos.

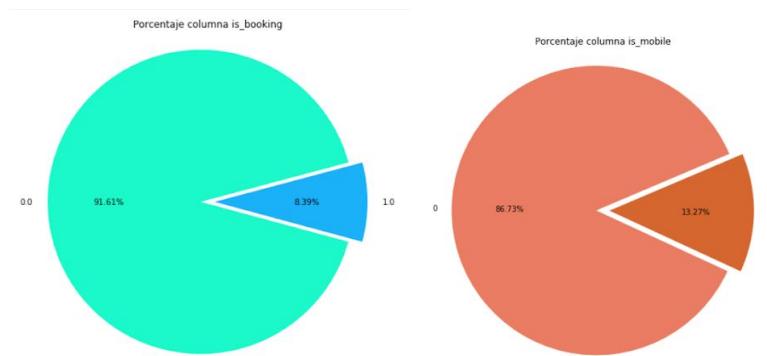
Nótese que la información sustraída la obtuvimos haciéndolo igual que con el banner; guardamos la información extrayéndola de aquella parte encontrada en las herramientas para desarrolladores (inspeccionando la página).

## 14. Conclusiones

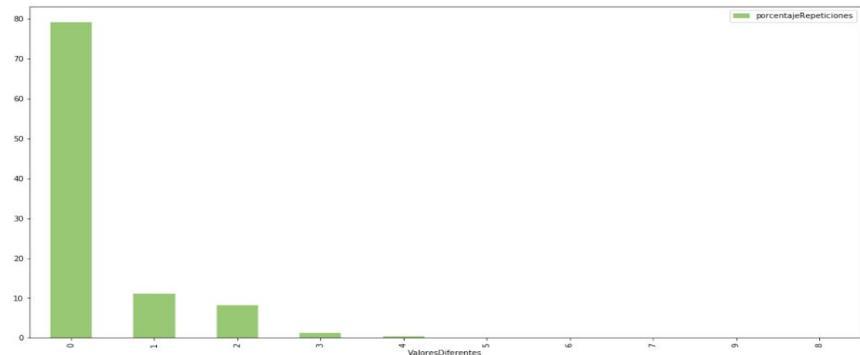
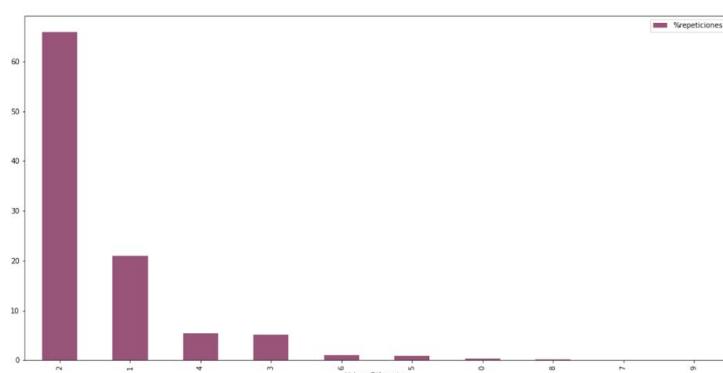
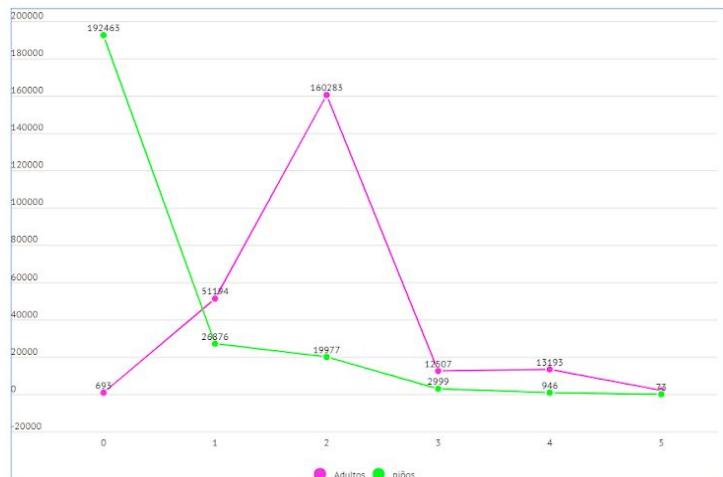
De conclusiones hemos distinguido dos tipos; por un lado las conclusiones extraídas de la representación, visualización y análisis gráfico del proyecto, y por otro lado están las conclusiones más generales del proyecto.

Conclusiones de la representación, visualización y análisis gráfico:

-En primer lugar haciendo un conteo de los datos de distintas columnas, vemos mediante esta gráfica(dcha) que la mayoría de clientes reservan desde dispositivos que no sean móviles (ordenadores), y así deducimos que es mucho más importante centrar nuestros anuncios a aplicaciones que se usan en su mayoría desde el ordenador. En la otra, sabiendo que la mayoría de los usuarios simplemente nos visitan y no reservan, intuimos que debemos conseguir que el cliente no se lo piense tanto y reserve a la primera.

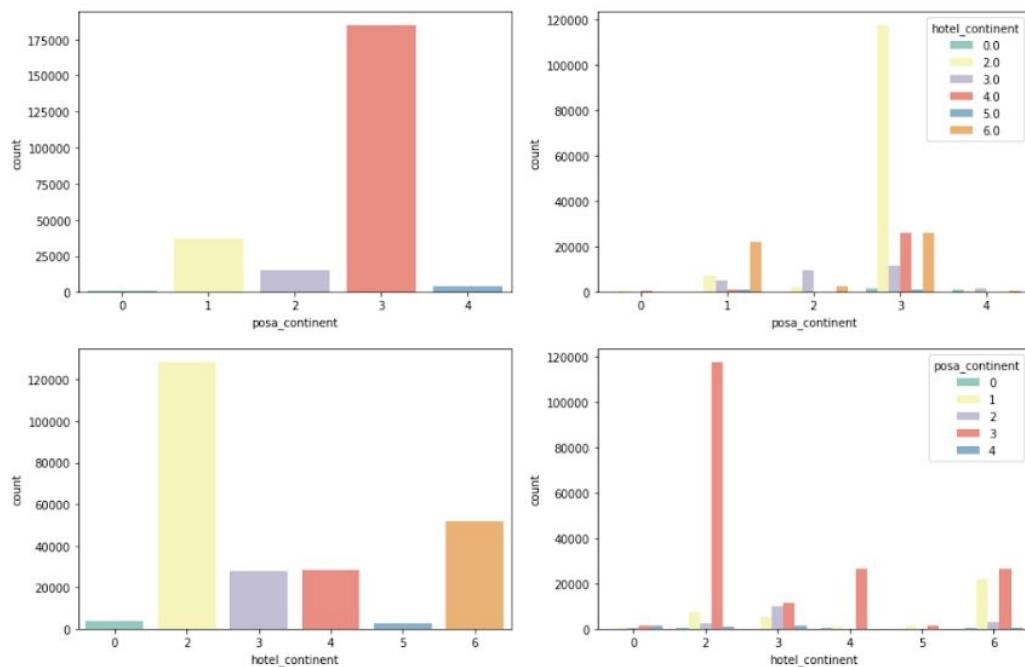


- Ahora con esta gráficas vemos que la mayoría de nuestras reservas las realiza o solamente un adulto o dos adultos y que la mayoría de nuestras reservas no contienen ningún niño. Y para visualizar esto de una sola vez construimos este diagrama de líneas en el que se ve como en la mayoría de las reservas hay 0 niños y dos adultos. Con todo esto definimos nuestro público objetivo, principalmente parejas sin niños y secundariamente adultos que viajan solos probablemente por trabajo.



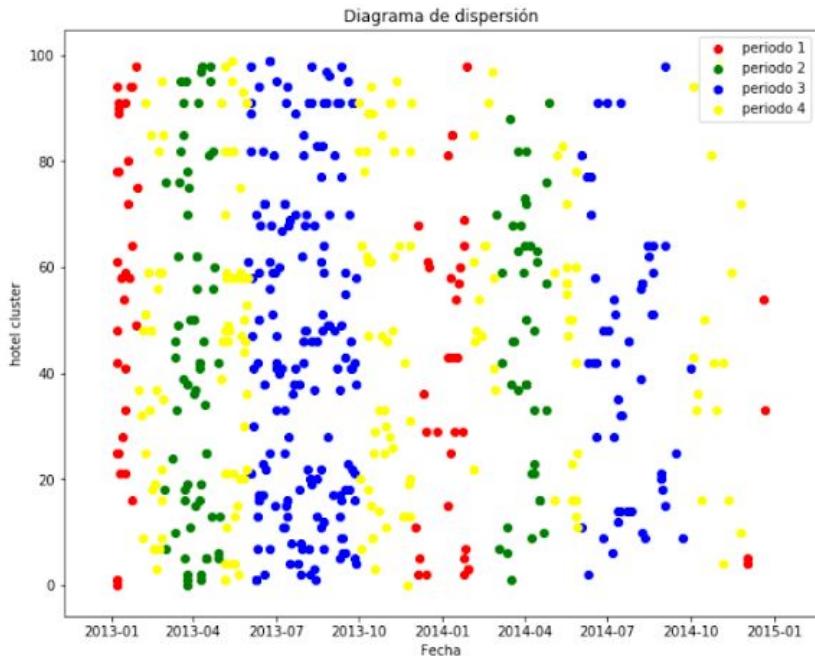
-También para representar más de una variable con una misma gráfica creamos diagramas de barras con los que en vez de medir solamente una columna, analizamos dos a la vez, y gracias a ello nos dimos cuenta de que la mayoría de las personas pertenecientes al continente 3 viajan en su mayoría al continente 2. Esto también nos ayuda a saber a qué continente pertenecen las personas que más viajan (en este caso pertenecen al continente número 3) y a saber cual es el continente al que más se viaja (en este caso el continente

número 2).



-De la misma manera que antes hacemos un recuento que nos indica qué grupo hotelero tiene más reservas, viéndolo con la gráfica de arriba. Y ahora con la gráfica de abajo vemos la dispersión de los grupos hoteleros pero basándonos en la nueva columna periodo que creamos antes, distinguiendo así los datos en función de cada uno de los períodos creados.





-Por último, como conclusión del proyecto, creemos que nuestra evolución sobre la obtención de información y el análisis de los datos ha crecido exponencialmente. Esto nos ayudará en el futuro. Ya que los conocimientos adquiridos en esta asignatura nos permiten sacar de un conjunto de datos nueva información.y además ahora somos capaces de extraer información útil y podemos representarla de una manera directa y concisa a través de distintos tipos de gráficos.

