Análisis de LTV, CAC y ROMI

Para este proyecto se estarán realizando las prácticas en el departamento de analítica de Showz, una empresa de venta de entradas de eventos. Nuestra primer tarea es ayudar a optimizar los gastos de marketing.

Para desarrollar este proyecto tenemos datos de la empresa Showz desde enero de 2017 hasta diciembre de 2018, un archivo con pedidos y otro con las estadísticas de gastos de marketing.

Se investigarán:

- cómo los clientes usan el servicio;
- cuándo empiezan a comprar;
- cuánto dinero aporta cada cliente a la compañía;
- cuándo los ingresos cubren el costo de adquisición de los clientes.

In [2]:
```
# # Importamos las liberias que requeriremos para el desarrollo del proyecto

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns
import matplotlib.cm as cm
```

In [3]:
```
visitas = pd.read_csv('visits_log_us.csv')
costos = pd.read_csv('costs_us.csv')
ordenes = pd.read_csv('orders_log_us.csv')
```

In [4]:
```
# Creamos una función para el análisis de los dataframes.

def analisis_exploratorio_df(data):
    print(data.head())
    print()
    print(data.info())
    print()
    print('Valores austentes:')
    print(data.isna().sum())
    print()
    print('Filas duplicadas: ', data.duplicated().sum())
```

In [5]:
```
analisis_exploratorio_df(visitas)
```

```
    Device              End Ts  Source Id            Start Ts  \
0    touch  2017-12-20 17:38:00          4  2017-12-20 17:20:00
1  desktop  2018-02-19 17:21:00          2  2018-02-19 16:53:00
2    touch  2017-07-01 01:54:00          5  2017-07-01 01:54:00
3  desktop  2018-05-20 11:23:00          9  2018-05-20 10:59:00
4  desktop  2017-12-27 14:06:00          3  2017-12-27 14:06:00

                   Uid
0  16879256277535980062
1    104060357244891740
2   7459035603376831527
3  16174680259334210214
4   9969694820036681168

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359400 entries, 0 to 359399
Data columns (total 5 columns):
 #   Column     Non-Null Count   Dtype 
---  ------     --------------   ----- 
 0   Device     359400 non-null  object
 1   End Ts     359400 non-null  object
 2   Source Id  359400 non-null  int64 
 3   Start Ts   359400 non-null  object
 4   Uid        359400 non-null  uint64
dtypes: int64(1), object(3), uint64(1)
memory usage: 13.7+ MB
None

Valores austentes:
Device       0
End Ts       0
Source Id    0
Start Ts     0
Uid          0
dtype: int64

Filas duplicadas:  0
```

In [6]:
```python
# Creamos una funsión para la limpieza de las columnas en los dataframes.

def limpiar_columnas(data):

    new_columns = []

    for columna in data.columns:
        columna_limpia = columna.lower()
        columna_limpia = columna_limpia.replace(' ', '_')
        new_columns.append(columna_limpia)

    data.columns = new_columns

    return data
```

In [7]:
```python
# Limpieza de data con la función creada.
```

```
visitas = limpiar_columnas(visitas)
```

In [8]:
```
# Modificación del formato de fecha a dos columnas.

visitas['start_ts'] = pd.to_datetime(visitas['start_ts'], format = '%Y-%m-%c
visitas['end_ts'] = pd.to_datetime(visitas['end_ts'], format = '%Y-%m-%d %H:
```

In [9]:
```
visitas.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359400 entries, 0 to 359399
Data columns (total 5 columns):
 #   Column     Non-Null Count   Dtype         
---  ------     --------------   -----         
 0   device     359400 non-null  object        
 1   end_ts     359400 non-null  datetime64[ns]
 2   source_id  359400 non-null  int64         
 3   start_ts   359400 non-null  datetime64[ns]
 4   uid        359400 non-null  uint64        
dtypes: datetime64[ns](2), int64(1), object(1), uint64(1)
memory usage: 13.7+ MB
```

In [10]:
```
# Visualización de cuantas visitas tengo por dispositivo.

visitas.value_counts('device')
```

Out[10]:
```
device
desktop    262567
touch       96833
Name: count, dtype: int64
```

In [11]:
```
# Conocer el número de visiatas de acuerdo a los canales de publidad contrat

visitas.value_counts('source_id')
```

Out[11]:
```
source_id
4     101794
3      85610
5      66905
2      47626
1      34121
9      13277
10     10025
7         36
6          6
Name: count, dtype: int64
```

Estas fuentes se refieren a las utms, para poder monitorear las diferentes campañas que se lanzan desde google apps, campañas de anuncios se pueden monitorear, traquear, a través de sus fuentes.

In [13]:
```
# Conocer cual fue el termino de nuestra primera y última visita.

print(visitas['start_ts'].min())
print(visitas['start_ts'].max())
```

```
print(visitas['end_ts'].min())
print(visitas['end_ts'].max())
```

```
2017-06-01 00:01:00
2018-05-31 23:59:00
2017-06-01 00:02:00
2018-06-01 01:26:00
```

In [14]:
```
visitas[visitas['end_ts'] > '2018-06-01']
```

Out[14]:

	device	end_ts	source_id	start_ts	uid
2330	desktop	2018-06-01 00:05:00	2	2018-05-31 23:30:00	5300693976971713685
43215	desktop	2018-06-01 00:13:00	4	2018-05-31 23:46:00	12296626599487328624
46667	desktop	2018-06-01 00:13:00	5	2018-05-31 23:48:00	1107134732939577311
154728	desktop	2018-06-01 00:09:00	2	2018-05-31 22:52:00	12484396335714047199
198329	desktop	2018-06-01 00:11:00	3	2018-05-31 23:59:00	3720373600909378583
216791	touch	2018-06-01 00:06:00	5	2018-05-31 23:45:00	9951896961656595558
263781	desktop	2018-06-01 00:25:00	1	2018-05-31 23:59:00	4906562732540547408
269213	touch	2018-06-01 00:12:00	10	2018-05-31 23:59:00	10723414689244282024
288563	touch	2018-06-01 00:01:00	4	2018-05-31 23:37:00	1094281763368510391
299620	desktop	2018-06-01 00:04:00	4	2018-05-31 23:59:00	83872787173869366
342205	touch	2018-06-01 01:26:00	1	2018-05-31 23:44:00	13670831770959132678

In [15]:
```
analisis_exploratorio_df(ordenes)
```

```
                 Buy Ts  Revenue                   Uid
0  2017-06-01 00:10:00    17.00  10329302124590727494
1  2017-06-01 00:25:00     0.55  11627257723692907447
2  2017-06-01 00:27:00     0.37  17903680561304213844
3  2017-06-01 00:29:00     0.55  16109239769442553005
4  2017-06-01 07:58:00     0.37  14200605875248379450

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50415 entries, 0 to 50414
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Buy Ts   50415 non-null  object
 1   Revenue  50415 non-null  float64
 2   Uid      50415 non-null  uint64
dtypes: float64(1), object(1), uint64(1)
memory usage: 1.2+ MB
None

Valores austentes:
Buy Ts     0
Revenue    0
Uid        0
dtype: int64

Filas duplicadas:  0
```

In [16]:
```
# Limpieza de las columnas del dataframe.

ordenes = limpiar_columnas(ordenes)
```

In [17]:
```
# Modificación del tipo de dato en columna.

ordenes['buy_ts'] = pd.to_datetime(ordenes['buy_ts'], format = '%Y-%m-%d %H:
```

In [18]:
```
ordenes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50415 entries, 0 to 50414
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   buy_ts   50415 non-null  datetime64[ns]
 1   revenue  50415 non-null  float64
 2   uid      50415 non-null  uint64
dtypes: datetime64[ns](1), float64(1), uint64(1)
memory usage: 1.2 MB
```

In [19]:
```
# ¿Cuál día se realizaron nuestra primera y última orden?

print(ordenes['buy_ts'].min())
print(ordenes['buy_ts'].max())
```

```
2017-06-01 00:10:00
2018-06-01 00:02:00
```

In [20]:
```
analisis_exploratorio_df(costos)
```

```
   source_id          dt  costs
0          1  2017-06-01  75.20
1          1  2017-06-02  62.25
2          1  2017-06-03  36.53
3          1  2017-06-04  55.00
4          1  2017-06-05  57.08

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2542 entries, 0 to 2541
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   source_id  2542 non-null   int64
 1   dt         2542 non-null   object
 2   costs      2542 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 59.7+ KB
None

Valores austentes:
source_id    0
dt           0
costs        0
dtype: int64

Filas duplicadas:  0
```

In [21]:
```
# Limpieza de columnas en data de costos.

costos = limpiar_columnas(costos)
```

In [22]:
```
# Cambio de tipo de data en una columna.

costos['dt'] = pd.to_datetime(costos['dt'])
```

In [23]:
```
costos.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2542 entries, 0 to 2541
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   source_id  2542 non-null   int64
 1   dt         2542 non-null   datetime64[ns]
 2   costs      2542 non-null   float64
dtypes: datetime64[ns](1), float64(1), int64(1)
memory usage: 59.7 KB
```

In [24]:
```
print(costos['dt'].min())
print(costos['dt'].max())
```

```
2017-06-01 00:00:00
2018-05-31 00:00:00
```

Visitas:

1. ¿Cuántas personas lo usan cada día, semana y mes?
2. ¿Cuántas sesiones hay por día? (Un usuario puede tener más de una sesión).
3. ¿Cuál es la duración de cada sesión?
4. ¿Con qué frecuencia los usuarios regresan?

In [26]:
```
visitas['sesion_mes'] = visitas['start_ts'].dt.month
visitas['sesion_semana'] = visitas['start_ts'].dt.isocalendar().week
visitas['sesion_fecha'] = visitas['start_ts'].dt.date
```

In [27]:
```
visitas.head()
```

Out[27]:

	device	end_ts	source_id	start_ts	uid	sesion_mes	sesio
0	touch	2017-12-20 17:38:00	4	2017-12-20 17:20:00	16879256277535980062	12	
1	desktop	2018-02-19 17:21:00	2	2018-02-19 16:53:00	104060357244891740	2	
2	touch	2017-07-01 01:54:00	5	2017-07-01 01:54:00	7459035603376831527	7	
3	desktop	2018-05-20 11:23:00	9	2018-05-20 10:59:00	16174680259334210214	5	
4	desktop	2017-12-27 14:06:00	3	2017-12-27 14:06:00	9969694820036681168	12	

In [28]:
```
visitas_agregadas_por_dia = visitas.groupby('sesion_fecha')['uid'].nunique()
```

In [29]:
```
visitas_agregadas_por_semana = visitas.groupby('sesion_semana')['uid'].nunic
```

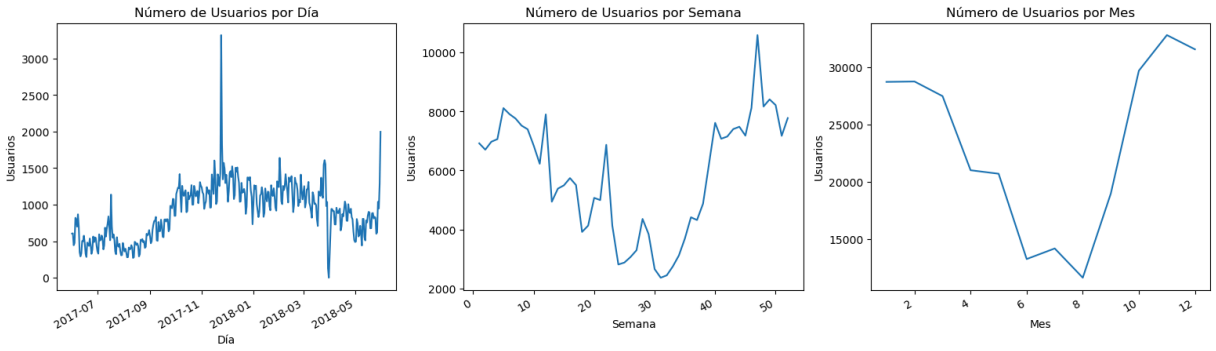
In [30]:
```
visitas_agregadas_por_mes = visitas.groupby('sesion_mes')['uid'].nunique().s
```

Usuarios por día, semana y mes.

In [32]:
```
print('Promedio de usuarios por día: ', round(visitas_agregadas_por_dia.mean
print('Promedio de usuarios por semana: ', round(visitas_agregadas_por_semar
print('Promedio de usuarios por mes: ', round(visitas_agregadas_por_mes.mean
```

```
Promedio de usuarios por día:  908
Promedio de usuarios por semana:  5825
Promedio de usuarios por mes:  23228
```

In [33]:
```
fig, ax = plt.subplots(1,3, figsize = (19, 5))

ax[0].plot(visitas_agregadas_por_dia)
ax[0].set(title = 'Número de Usuarios por Día', xlabel = 'Día', ylabel = 'Us
ax[1].plot(visitas_agregadas_por_semana)
ax[1].set(title = 'Número de Usuarios por Semana', xlabel = 'Semana', ylabel
ax[2].plot(visitas_agregadas_por_mes)
ax[2].set(title = 'Número de Usuarios por Mes', xlabel = 'Mes', ylabel = 'Us
fig.autofmt_xdate(rotation = 30)
plt.show()
```



Número de sesiones por día.

In [35]:
```
# ¿Cuál es el promedio de sesiones al día?

sesiones_por_dia = visitas.groupby('sesion_fecha')['uid'].count().mean().rou
print('El promedio de sesiones por día es de: ', sesiones_por_dia)
```

```
El promedio de sesiones por día es de:  987
```

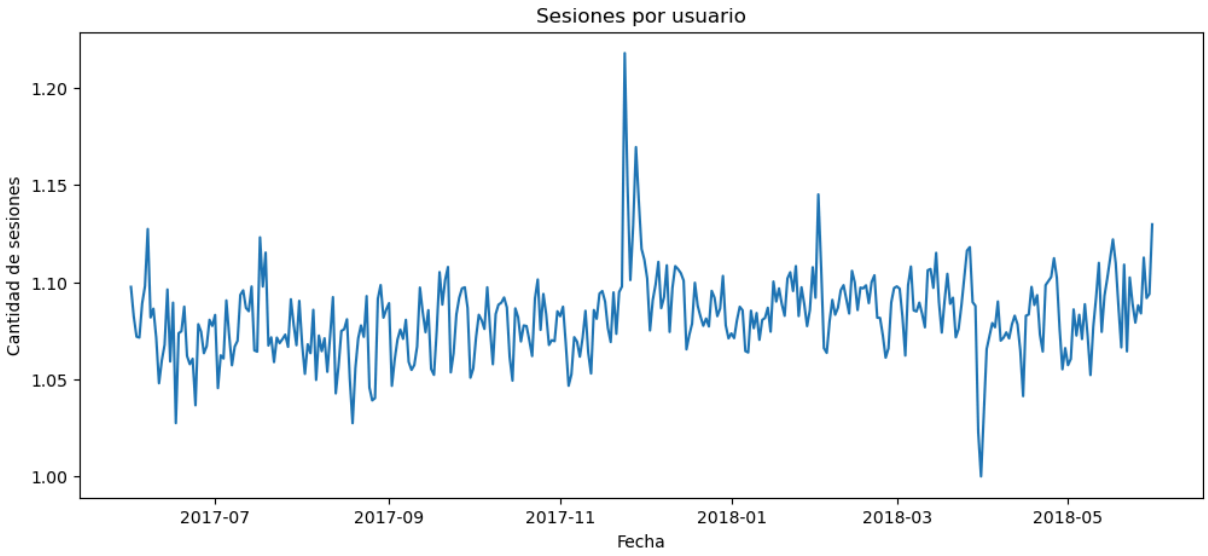
In [36]:
```
#Ahora vamos a ver la cantidad de sesiones que cada usuario hace por día.

sesiones_por_usuario = visitas.groupby('sesion_fecha').agg({'uid': ['count',
```

In [37]:
```
# Modificación de nombres de columnas.

sesiones_por_usuario.columns = ['n_sesiones', 'n_usuarios']
```

In [38]:
```
# Creación de una nueva columna sesión por usuario.

sesiones_por_usuario['sesion_por_usuario'] = sesiones_por_usuario['n_sesione
```

In [39]:
```
# Elaboración de un gráfico para una mejor visualización de las sesiones por

plt.figure(figsize = (12,5))
sesiones_por_usuario['sesion_por_usuario'].plot().set(
    title = 'Sesiones por usuario',
    xlabel = 'Fecha',
```

```
    ylabel = 'Cantidad de sesiones')
plt.show()
```



In [40]:
```
print('El promedio de sesiones por usuario es de:', round(sesiones_por_usuar
```

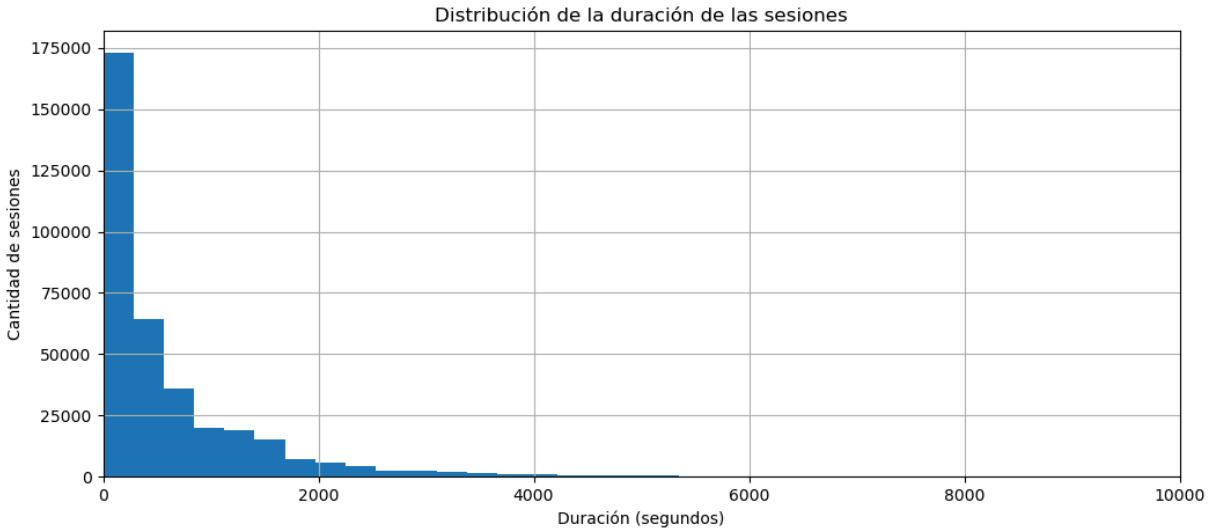
```
El promedio de sesiones por usuario es de: 1.08
```

In [41]:
```
# Creación de una nueva columna para la duración de la sesión.

visitas['duracion_sesion'] = (visitas['end_ts'] - visitas['start_ts']).dt.se
```

In [42]:
```
visitas['duracion_sesion'].describe()
```

Out[42]:
```
count    359400.000000
mean        643.506489
std        1016.334786
min           0.000000
25%         120.000000
50%         300.000000
75%         840.000000
max       84480.000000
Name: duracion_sesion, dtype: float64
```

In [43]:
```
# Elaboración de un histograma para visualizar mejor la distribución de la c

plt.figure(figsize = (12,5))
visitas['duracion_sesion'].hist(bins = 300).set(
    title = 'Distribución de la duración de las sesiones',
    xlabel = 'Duración (segundos)',
    ylabel = 'Cantidad de sesiones')
plt.xlim(0, 10000)
plt.show()
```



In [44]:
```
print('El promedio de la duración de una sesión es de: ', round(visitas['dur
print('La mediana de la duración de una sesión es de: ', round(visitas['dura
print('La moda de la duración de una sesión es de: ', round(visitas['duracio
```

```
El promedio de la duración de una sesión es de:  11
La mediana de la duración de una sesión es de:  5
La moda de la duración de una sesión es de:  1
```

In [45]:
```
# Se va a calcular el porcentaje que se va y que se queda de los usuarios, v

primeras_visitas = visitas.groupby('uid')['start_ts'].min().reset_index()
```

In [46]:
```
# Renombran las columnas.

primeras_visitas.columns = ['uid', 'primera_sesion_start_ts']
```

In [47]:
```
# Cambio de tipo de dato a dos columnas.

primeras_visitas['primera_sesion_fecha'] = primeras_visitas['primera_sesion_
primeras_visitas['primera_sesion_mes'] = primeras_visitas['primera_sesion_st
```

In [48]:
```
primeras_visitas.head()
```

Out[48]:

	uid	primera_sesion_start_ts	primera_sesion_fecha	primera_sesion_
0	11863502262781	2018-03-01 17:27:00	2018-03-01	2018
1	49537067089222	2018-02-06 15:55:00	2018-02-06	2018
2	297729379853735	2017-06-07 18:47:00	2017-06-07	2017
3	313578113262317	2017-09-18 22:49:00	2017-09-18	2017
4	325320750514679	2017-09-30 14:29:00	2017-09-30	2017

In [49]:
```
# Fusión de los dataframe primeras visitas y visitas por el uid.
```

```
visitas_full = pd.merge(primeras_visitas, visitas, on = 'uid')
```

In [50]:
```
# Cambio de datatype a dos columnas.

visitas_full['sesion_fecha'] = pd.to_datetime(visitas_full['sesion_fecha'])
visitas_full['primera_sesion_fecha'] = pd.to_datetime(visitas_full['primera_
```

In [51]:
```
visitas_full.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 359400 entries, 0 to 359399
Data columns (total 12 columns):
 #   Column                   Non-Null Count   Dtype         
---  ------                   --------------   -----         
 0   uid                      359400 non-null  uint64        
 1   primera_sesion_start_ts  359400 non-null  datetime64[ns]
 2   primera_sesion_fecha     359400 non-null  datetime64[ns]
 3   primera_sesion_mes       359400 non-null  period[M]     
 4   device                   359400 non-null  object        
 5   end_ts                   359400 non-null  datetime64[ns]
 6   source_id                359400 non-null  int64         
 7   start_ts                 359400 non-null  datetime64[ns]
 8   sesion_mes               359400 non-null  int32         
 9   sesion_semana            359400 non-null  UInt32        
 10  sesion_fecha             359400 non-null  datetime64[ns]
 11  duracion_sesion          359400 non-null  int32         
dtypes: UInt32(1), datetime64[ns](5), int32(2), int64(1), object(1), period
[M](1), uint64(1)
memory usage: 29.1+ MB
```

In [52]:
```
# Creación de columna que nos permida ver la edad en meses de las operacione

visitas_full['edad_meses'] = ((
    visitas_full['sesion_fecha'] - visitas_full['primera_sesion_fecha'])/(30
```

In [53]:
```
visitas_full.head()
```

Out[53]:

	uid	primera_sesion_start_ts	primera_sesion_fecha	primera_sesion_
0	11863502262781	2018-03-01 17:27:00	2018-03-01	2018
1	49537067089222	2018-02-06 15:55:00	2018-02-06	2018
2	297729379853735	2017-06-07 18:47:00	2017-06-07	2017
3	313578113262317	2017-09-18 22:49:00	2017-09-18	2017
4	313578113262317	2017-09-18 22:49:00	2017-09-18	2017

In [54]:
```
visitas_full['edad_meses'].value_counts()
```

Out[54]:
```
edad_meses
0     269867
1      20550
2      15094
3      11188
4      10194
5       7721
6       7452
7       5092
8       4424
9       3149
10      2292
11      1575
12       802
Name: count, dtype: int64
```

In [55]:
```
# Se ordenan valores por la columna: primera_sesion_fecha.

visitas_full.sort_values(by= 'primera_sesion_fecha')
```

Out[55]:

	uid	primera_sesion_start_ts	primera_sesion_fecha	prim
23146	1186757012633220834	2017-06-01 10:48:00	2017-06-01	
96924	4989871819839869689	2017-06-01 15:15:00	2017-06-01	
96923	4989871819839869689	2017-06-01 15:15:00	2017-06-01	
96922	4989871819839869689	2017-06-01 15:15:00	2017-06-01	
96921	4989871819839869689	2017-06-01 15:15:00	2017-06-01	
...	...	...	...	
338040	17319230204043134264	2018-05-31 22:40:00	2018-05-31	
310149	15845953194300008222	2018-05-31 16:00:00	2018-05-31	
220340	11264067749718153135	2018-05-31 23:22:00	2018-05-31	
218915	11217684785870472309	2018-05-31 09:27:00	2018-05-31	
2270	127310842069762690	2018-05-31 10:28:00	2018-05-31	

359400 rows × 13 columns

In [56]:
```
# Elaboración de tabla dinámica que nos permita crear las cohortes.

cohortes = visitas_full.pivot_table(
    index = 'primera_sesion_mes',
    columns = 'edad_meses',
    values = 'uid',
    aggfunc = 'nunique'
)
```

In [57]:
```
# Modificación de visualización de valores ausentes.
```

```
cohortes.fillna('')
```

Out[57]:

edad_meses	0	1	2	3	4	5	6	7	
primera_sesion_mes									
2017-06	13259.0	955.0	720.0	775.0	944.0	847.0	875.0	713.0	746
2017-07	13140.0	716.0	688.0	675.0	767.0	609.0	606.0	619.0	527
2017-08	10181.0	691.0	634.0	577.0	547.0	412.0	438.0	340.0	299
2017-09	16704.0	1239.0	1117.0	816.0	683.0	642.0	571.0	420.0	366
2017-10	25977.0	1858.0	1384.0	960.0	975.0	787.0	565.0	478.0	120
2017-11	27248.0	1849.0	1270.0	1016.0	900.0	639.0	520.0	107.0	
2017-12	25268.0	1257.0	1057.0	719.0	577.0	400.0	101.0		
2018-01	22624.0	1191.0	830.0	557.0	421.0	47.0			
2018-02	22197.0	1039.0	602.0	407.0	67.0				
2018-03	20589.0	835.0	533.0	81.0					
2018-04	15709.0	614.0	82.0						
2018-05	15273.0	100.0							

In [58]:
```
# Creación del nuevo dataframe para la información de la retención.

retencion = pd.DataFrame()
```

In [59]:
```
# Creación de un for para crear una nueva tabla concatenando la división de

for columna in cohortes.columns:
    retencion = pd.concat([retencion, cohortes[columna]/cohortes[0]], axis =
```

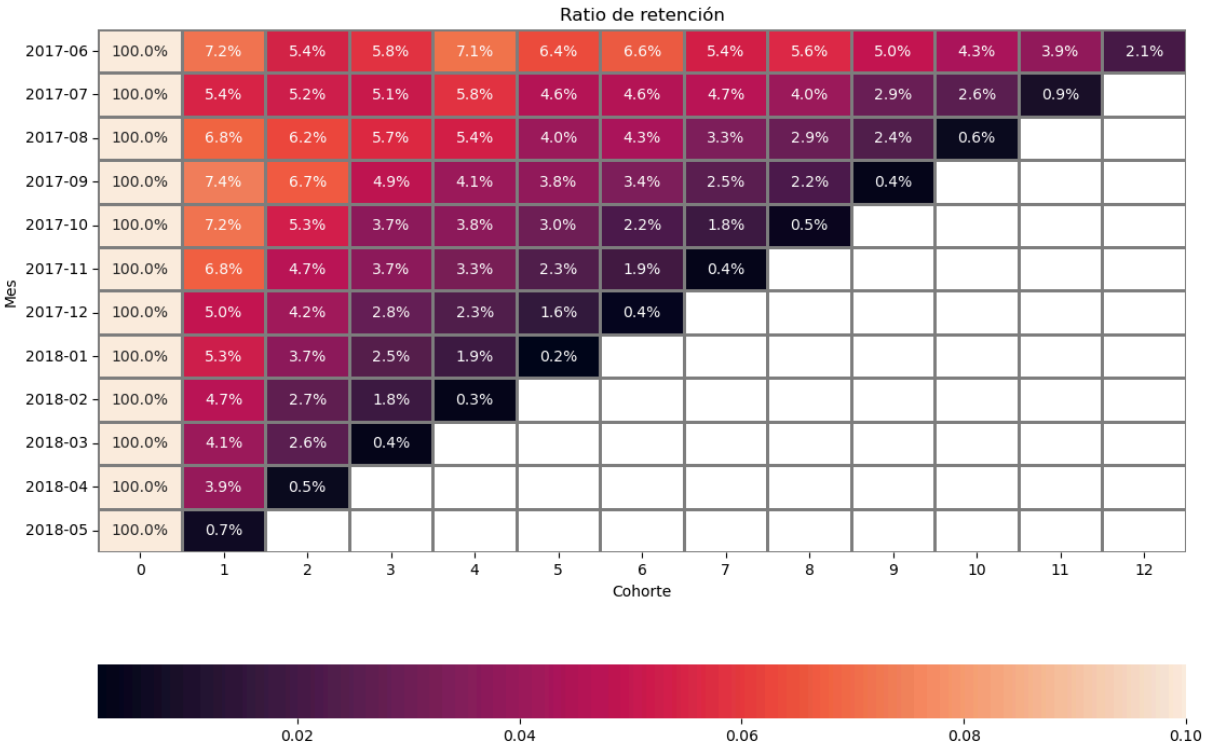
In [60]:
```
retencion.columns = cohortes.columns
```

In [61]:
```
retencion.index = [str(indice)[0:10] for indice in retencion.index]
```

¿Con qué frecuencia los usuarios regresan?

In [63]:
```
# Diseño de mapa de calor que nos permita visualizar el ratio de retención c

plt.figure(figsize = (13, 9))
sns.heatmap(retencion,
            annot = True,
            fmt = '.1%',
            linewidths = 1,
            linecolor = 'grey',
            vmax = 0.1,
            cbar_kws = {'orientation': 'horizontal'}).set(
```

```
            title = 'Ratio de retención',
            ylabel = 'Mes',
            xlabel = 'Cohorte')
plt.show()
```



Se tendría que realizar alguna campaña que nos permita conocer cuales son los comentarios de nuestros consumidores despues de acudir al primer evento porque en base al mapa de calor anterior se visualiza que el porcentaje de retención es muy bajo en todas las cohortes. De igual manera se podría realizar un análisis de mbudo para ver si hay algún problema al momento de estar haciendo las adquisiciones de sus entradas o ver si hay algún impedimento que no permita realizar la compra en la página.

In [65]:
```
# Promedio de retención para la primera cohorte.

print(retencion[1].mean())
```

```
0.05361100820922162
```

Ventas:

1. ¿Cuándo empieza la gente a comprar? (En el análisis de KPI, generalmente nos interesa saber el tiempo que transcurre entre el registro y la conversión, es decir, cuando el usuario se convierte en cliente. Por ejemplo, si el registro y la primera compra ocurren el mismo día, el usuario podría caer en la categoría Conversion 0d. Si la primera compra ocurre al día siguiente, será Conversion 1d. Puedes usar cualquier enfoque que te permita comparar las conversiones de diferentes cohortes para que puedas determinar qué cohorte o canal de marketing es más efectivo.)

2. ¿Cuántos pedidos hacen durante un período de tiempo dado?
3. ¿Cuál es el tamaño promedio de compra?
4. ¿Cuánto dinero traen? (LTV)

In [67]:
```
# Creación de una nueva tabla llamada compras.

compras = visitas_full.merge(ordenes, on = 'uid', how = 'outer')
```

In [68]:
```
# Elaboración de una nueva columna llamada buy day.

compras['buy_day'] = pd.to_datetime(compras['buy_ts'].dt.date)
```

In [69]:
```
# Agrupamiento por la columna 'uid' para traer la información de la primera

compras_conversion = compras.groupby('uid')[['primera_sesion_fecha', 'buy_da
```

In [70]:
```
# Se crea una nueva columna para obtener la información de la conversión en

compras_conversion['conversion_dias'] = (
    compras_conversion['buy_day'] - compras_conversion['primera_sesion_fecha
```

In [71]:
```
# Eliminación  de valores ausentes.

compras_conversion.dropna()
```

Out[71]:

	uid	primera_sesion_fecha	buy_day	conversion_dias
3	313578113262317	2017-09-18	2018-01-03	107.0
22	1575281904278712	2017-06-03	2017-06-03	0.0
35	2429014661409475	2017-10-11	2017-10-11	0.0
36	2464366381792757	2018-01-27	2018-01-28	1.0
38	2551852515556206	2017-11-24	2017-11-24	0.0
...	...	...	...	...
228142	18445147675727495770	2017-08-20	2017-11-24	96.0
228143	18445407535914413204	2017-09-22	2017-09-22	0.0
228148	18445601152732270159	2017-08-07	2018-03-26	231.0
228161	18446156210226471712	2017-11-07	2018-02-18	103.0
228162	18446167067214817906	2017-10-17	2017-10-17	0.0

36523 rows × 4 columns

In [72]:
```
# Agrupación por la columna 'converión_días para contar los 'uid' que caen e

conversion = compras_conversion.groupby('conversion_dias')['uid'].count().re
```

In [73]:
```
# Filtrado de información para obtener los visitantes que se convirteron en

conversion_0_dias = conversion.query('conversion_dias == 0.0').sum()
conversion_0_dias = conversion_0_dias['uid']
```

In [74]:
```
# Filtrado de información para obtener los visitantes que se convirteron en

conversion_7_dias = conversion.query('conversion_dias <= 7.0').sum()
conversion_7_dias = conversion_7_dias['uid']
```

In [75]:
```
# Filtrado de información para obtener los visitantes que se convirteron en

conversion_30_dias = conversion.query('conversion_dias <= 30.0').sum()
conversion_30_dias = conversion_30_dias['uid']
```

In [76]:
```
# Filtrado de información para obtener los visitantes que se convirteron en
```

```
conversion_anual = conversion.query('conversion_dias <= 365.0').sum()
conversion_anual = conversion_anual['uid']
conversion_anual_porcentaje = f'{conversion_anual/conversion_anual*100:.2f}%
```

In [77]:
```
# Se cambia el formato de la información.

porcentaje_conversion_0 = f'{(conversion_0_dias/conversion_anual)*100:.2f}%'
porcentaje_conversion_7 = f'{(conversion_7_dias/conversion_anual)*100:.2f}%'
porcentaje_conversion_30 = f'{(conversion_30_dias/conversion_anual)*100:.2f}
```

¿Cuándo empieza la gente a comprar?

In [79]:
```
print(f'Conversión 0 días: ',porcentaje_conversion_0)
print(f'Conversión 7 días: ',porcentaje_conversion_7)
print(f'Conversión 30 días: ',porcentaje_conversion_30)
print(f'Conversión anual: ',conversion_anual_porcentaje)
```

```
Conversión 0 días:  68.56%
Conversión 7 días:  80.32%
Conversión 30 días:  86.49%
Conversión anual:  100.00%
```

El 68.56% de nuestros visitadores se convirtieron en clientes el mismo día de su visita.
El 86.49% de nuestros visitadores ya se habían convertido en clientes al mes de su primer visita.
En el año que realizamos de análisis el 100% de nuestros visitantes se convirtieron en clientes.

In [81]:
```
# Se crea una copia del df ordenes.

ordenes_analisis = ordenes.copy()
```

In [82]:
```
# Creación de columnas para el día, semana y mes.

ordenes_analisis['buy_day'] = pd.to_datetime(ordenes_analisis['buy_ts'].dt.d
ordenes_analisis['buy_week'] = ordenes_analisis['buy_ts'].dt.isocalendar().w
ordenes_analisis['buy_mes'] = ordenes_analisis['buy_ts'].dt.month
```

In [83]:
```
# Se agrupan por día, semana y mes, para conocer el numero de uids únicos de

ordenes_diarias = ordenes_analisis.groupby('buy_day')['uid'].nunique().sort_
ordenes_semanales = ordenes_analisis.groupby('buy_week')['uid'].nunique().so
ordenes_mensuales = ordenes_analisis.groupby('buy_mes')['uid'].nunique().sor
```

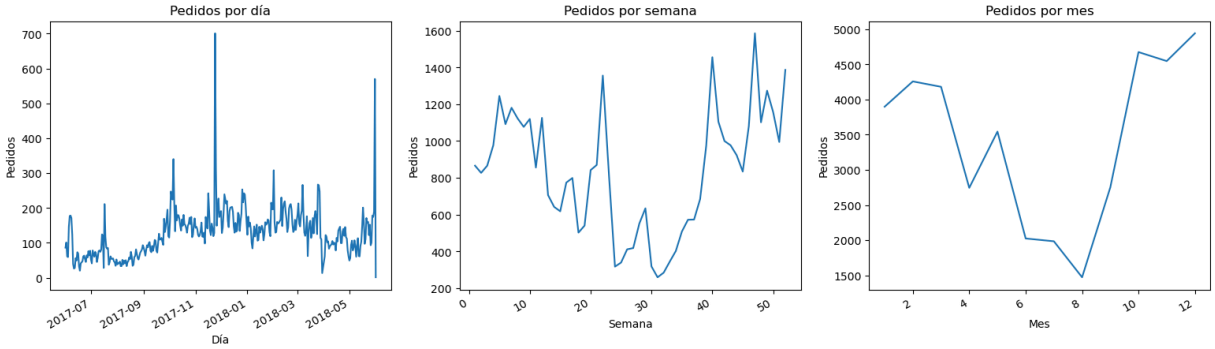
¿Cuántos pedidos hacen durante un período de tiempo dado?

Pedidos por día, mes y semana.

In [85]:
```
print(f'Promedio de pedidos por día:', round(ordenes_diarias.mean()))
print(f'Promedio de pedidos por semana:', round(ordenes_semanales.mean()))
print(f'Promedio de pedidos por mes:', round(ordenes_mensuales.mean()))
```

```
Promedio de pedidos por día: 128
Promedio de pedidos por semana: 833
Promedio de pedidos por mes: 3418
```

In [86]:
```
fig, ax = plt.subplots(1,3, figsize = (19, 5))

ax[0].plot(ordenes_diarias)
ax[0].set(title = 'Pedidos por día', xlabel = 'Día', ylabel = 'Pedidos')
ax[1].plot(ordenes_semanales)
ax[1].set(title = 'Pedidos por semana', xlabel = 'Semana', ylabel = 'Pedidos
ax[2].plot(ordenes_mensuales)
ax[2].set(title = 'Pedidos por mes', xlabel = 'Mes', ylabel = 'Pedidos')
fig.autofmt_xdate(rotation = 30)
plt.show()
```



En este grafico representamos el número de pedidos que se realizaron por día, semana y mes durante el año.

¿Cuál es el tamaño promedio de compra?

Facturación diaria, semanal y mensual.

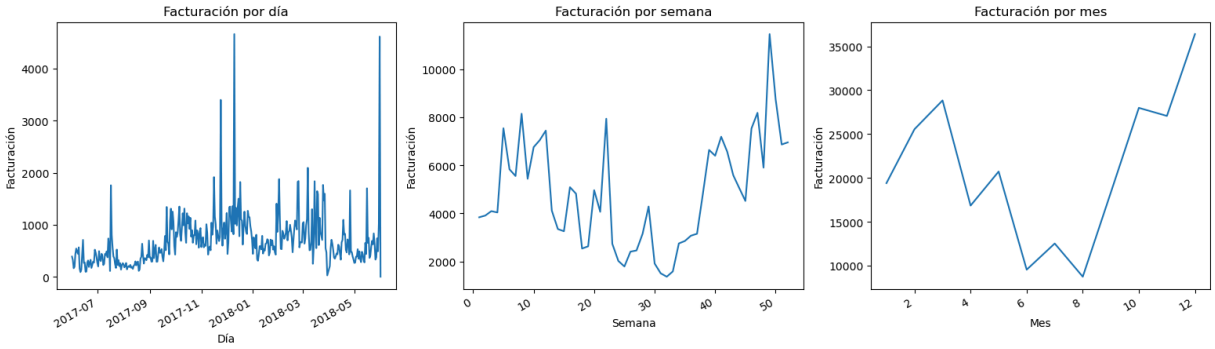
In [89]:
```
# Agrupación de pedidos por día, semana y mes, con el objativo de conocer cu

facturacion_diaria = ordenes_analisis.groupby('buy_day')['revenue'].sum().sc
facturacion_semanal = ordenes_analisis.groupby('buy_week')['revenue'].sum().
facturacion_mensual = ordenes_analisis.groupby('buy_mes')['revenue'].sum().s
```

In [90]:
```
print(f'Promedio de facturación por día: $',round(facturacion_diaria.mean()))
print(f'Promedio de facturación por semana: $',round(facturacion_semanal.mea
print(f'Promedio de facturación por mes: $',round(facturacion_mensual.mean()
```

```
Promedio de facturación por día: $ 692
Promedio de facturación por semana: $ 4847
Promedio de facturación por mes: $ 21005
```

In [91]:
```
fig, ax = plt.subplots(1,3, figsize = (19, 5))
```

```
ax[0].plot(facturacion_diaria)
ax[0].set(title = 'Facturación por día', xlabel = 'Día', ylabel = 'Facturaci
ax[1].plot(facturacion_semanal)
ax[1].set(title = 'Facturación por semana', xlabel = 'Semana', ylabel = 'Fac
ax[2].plot(facturacion_mensual)
ax[2].set(title = 'Facturación por mes', xlabel = 'Mes', ylabel = 'Facturaci
fig.autofmt_xdate(rotation = 30)
plt.show()
```



En estos gráficos se plasman el total de la facturación por día, por semana y por mes, los cuales llevan una tendencía muy parecida a los que se vieron en los pedidos.

In [93]:
```
ordenes_ltv = ordenes.copy()
```

In [94]:
```
# Se crea una columna para identificar el mes en el que se realizo la orden.

ordenes_ltv['orden_mes'] = ordenes['buy_ts'].dt.to_period('M')
```

In [95]:
```
ordenes_ltv['orden_mes'] = ordenes_ltv['orden_mes'].dt.start_time
```

In [96]:
```
# Agrupamiento por el uid, para seleccionar las ordenes de menor edad por me

primeras_ordenes = ordenes_ltv.groupby('uid').agg({'orden_mes': 'min'}).rese
primeras_ordenes.columns = ['uid', 'primera_orden_mes']
primeras_ordenes.head()
```

Out[96]:

	uid	primera_orden_mes
0	313578113262317	2018-01-01
1	1575281904278712	2017-06-01
2	2429014661409475	2017-10-01
3	2464366381792757	2018-01-01
4	2551852515556206	2017-11-01

In [97]:
```
# Conocer cuantos nuevos clientes se tuvieron por mes.

cohorts_sizes = primeras_ordenes.groupby('primera_orden_mes').agg({'uid': 'r
cohorts_sizes.columns = ['primera_orden_mes', 'n_buyers']
cohorts_sizes.head()
```

Out[97]:

	primera_orden_mes	n_buyers
0	2017-06-01	2023
1	2017-07-01	1923
2	2017-08-01	1370
3	2017-09-01	2581
4	2017-10-01	4340

In [98]:
```
# Fusión de ordenes ltv y primeras ordenes por el uid.

ordenes_primeras = pd.merge(ordenes_ltv, primeras_ordenes, on = 'uid')
ordenes_primeras.head()
```

Out[98]:

	buy_ts	revenue	uid	orden_mes	primera_orden_mes
0	2017-06-01 00:10:00	17.00	10329302124590727494	2017-06-01	2017-06-01
1	2017-06-01 00:25:00	0.55	11627257723692907447	2017-06-01	2017-06-01
2	2017-06-01 00:27:00	0.37	17903680561304213844	2017-06-01	2017-06-01
3	2017-06-01 00:29:00	0.55	16109239769442553005	2017-06-01	2017-06-01
4	2017-06-01 07:58:00	0.37	14200605875248379450	2017-06-01	2017-06-01

In [99]:
```
# Se agrupan por mes para conocer la suma de la facturación por mes de cada

cohortes_pedidos = ordenes_primeras.groupby(['primera_orden_mes', 'orden_mes
cohortes_pedidos.head()
```

Out[99]:

	primera_orden_mes	orden_mes	revenue
0	2017-06-01	2017-06-01	9557.49
1	2017-06-01	2017-07-01	981.82
2	2017-06-01	2017-08-01	885.34
3	2017-06-01	2017-09-01	1931.30
4	2017-06-01	2017-10-01	2068.58

In [100…
```
# Se genera una tabla mediante la fusión de cohorts sizes y cohortes pedidos

reporte = pd.merge(cohorts_sizes, cohortes_pedidos, on = 'primera_orden_mes'
reporte.head()
```

Out[100…

	primera_orden_mes	n_buyers	orden_mes	revenue
0	2017-06-01	2023	2017-06-01	9557.49
1	2017-06-01	2023	2017-07-01	981.82
2	2017-06-01	2023	2017-08-01	885.34
3	2017-06-01	2023	2017-09-01	1931.30
4	2017-06-01	2023	2017-10-01	2068.58

In [101…
```
reporte.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 79 entries, 0 to 78
Data columns (total 4 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   primera_orden_mes  79 non-null     datetime64[ns]
 1   n_buyers           79 non-null     int64         
 2   orden_mes          79 non-null     datetime64[ns]
 3   revenue            79 non-null     float64       
dtypes: datetime64[ns](2), float64(1), int64(1)
memory usage: 2.6 KB
```

In [102…
```
# Creación de una columna que nos permita saber cuantos meses transcurrieron

reporte['edad'] = ((reporte['orden_mes']-reporte['primera_orden_mes'])/(30*
```

In [103…
```
# Creación de la columna ltv.

reporte['ltv'] = reporte['revenue'] / reporte['n_buyers']
reporte.head()
```

Out[103…

	primera_orden_mes	n_buyers	orden_mes	revenue	edad	ltv
0	2017-06-01	2023	2017-06-01	9557.49	0	4.724414
1	2017-06-01	2023	2017-07-01	981.82	1	0.485329
2	2017-06-01	2023	2017-08-01	885.34	2	0.437637
3	2017-06-01	2023	2017-09-01	1931.30	3	0.954671
4	2017-06-01	2023	2017-10-01	2068.58	4	1.022531

In [104…
```
# Se cambia el tipo de fecha para ordenamiento de tabla.

reporte['primera_orden_mes'] = reporte['primera_orden_mes'].dt.to_period('M'
```

In [105…
```
# Creación de tabla dinámica para visualizar cuanto dinero aporto en promedi
ltv = reporte.pivot_table(
    index = 'primera_orden_mes',
    columns = 'edad',
    values = 'ltv',
```

```
    aggfunc = 'sum').round(2)

ltv.fillna('')
```

Out[105…

edad	0	1	2	3	4	5	6	7	8	9	10	
primera_orden_mes												
2017-06	4.72	0.49	0.44	0.95	1.02	0.74	0.95	0.58	0.55	0.61	0.57	0
2017-07	6.01	0.34	0.62	0.36	0.18	0.16	0.12	0.14	0.16	0.15	0.16	
2017-08	5.28	0.47	0.46	0.39	0.49	0.28	0.21	0.41	0.29	0.19		
2017-09	5.64	1.12	0.52	3.98	0.4	0.65	0.7	0.24	0.18			
2017-10	5.00	0.54	0.19	0.16	0.15	0.12	0.08	0.12				
2017-11	5.15	0.4	0.2	0.32	0.15	0.05	0.11					
2017-12	4.74	0.26	0.93	1.07	0.31	0.34						
2018-01	4.14	0.29	0.3	0.14	0.06							
2018-02	4.16	0.28	0.08	0.07								
2018-03	4.84	0.3	0.32									
2018-04	4.66	0.53										
2018-05	4.66											
2018-06	3.42											

In [106…

```
# Creamos una tabla dinámica que nos permita sumar acumulativamente las gana

ganancias_acumuladas = pd.DataFrame(columns=[f'{i}' for i in range(12)])

for index, row  in ltv.iterrows():
    acumulado = 0
    acumulados_fila = []
    for mes in range(12):
        acumulado += row[mes]
        acumulados_fila.append(acumulado)

    ganancias_acumuladas.loc[index] = acumulados_fila

ganancias_acumuladas.fillna('')
```

Out [106…

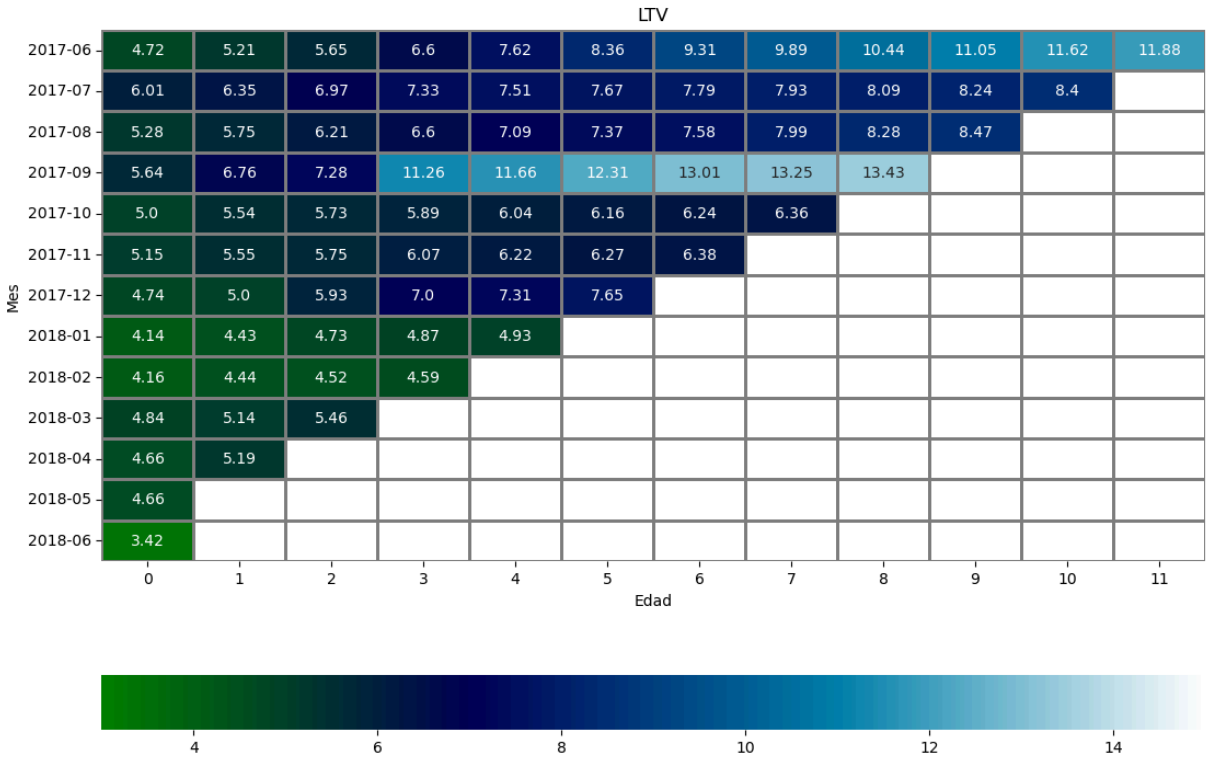
	0	1	2	3	4	5	6	7	8	9	10	11
2017-06	4.72	5.21	5.65	6.6	7.62	8.36	9.31	9.89	10.44	11.05	11.62	11.88
2017-07	6.01	6.35	6.97	7.33	7.51	7.67	7.79	7.93	8.09	8.24	8.4	
2017-08	5.28	5.75	6.21	6.6	7.09	7.37	7.58	7.99	8.28	8.47		
2017-09	5.64	6.76	7.28	11.26	11.66	12.31	13.01	13.25	13.43			
2017-10	5.00	5.54	5.73	5.89	6.04	6.16	6.24	6.36				
2017-11	5.15	5.55	5.75	6.07	6.22	6.27	6.38					
2017-12	4.74	5.0	5.93	7.0	7.31	7.65						
2018-01	4.14	4.43	4.73	4.87	4.93							
2018-02	4.16	4.44	4.52	4.59								
2018-03	4.84	5.14	5.46									
2018-04	4.66	5.19										
2018-05	4.66											
2018-06	3.42											

4. ¿Cuánto dinero traen? (LTV)

In [108…

```
# Creación de un mapa de calor que nos permita visualizar la información de

plt.figure(figsize = (13,9))
sns.heatmap(ganancias_acumuladas,
            annot = True,
            fmt = '.11',
            linewidths = 1,
            linecolor = 'grey',
            vmax = 15,
            vmin = 3,
            cmap = 'ocean',
            cbar_kws = {'orientation' : 'horizontal'}).set(
            title = 'LTV',
            xlabel = 'Edad',
```

```
            ylabel = 'Mes')

plt.show()
```



De las ganancias acumulativas que nos han dejado los clientes, en solo 2 cohortes se han despegado un poco del resto y han superado los `$` 10 dólares por cliente, en la primer cohorte del mes de junio y la cuarta cohorte del mes de septiembre. Teniendo en cuenta que el promedio de las ganancias iniciales de las cohortes han sido de aproximadamente `$` 5 dólares, solo en estas dos cohortes se han incrementado más del 100% las oatras cohortes se han quedado por debajo. Tendremos que ver la eficiencia de las campañas de marketing para poder saber el porque tan baja la retención, así como las utilidaes.

In [110…
```
costos_ = costos.copy()
```

In [111…
```
costos_['orden_mes'] = costos_['dt'].dt.to_period('M')
```

In [112…
```
costos_.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2542 entries, 0 to 2541
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   source_id  2542 non-null   int64
 1   dt         2542 non-null   datetime64[ns]
 2   costs      2542 non-null   float64
 3   orden_mes  2542 non-null   period[M]
dtypes: datetime64[ns](1), float64(1), int64(1), period[M](1)
memory usage: 79.6 KB
```

In [113…
```
# Se hace un dataframe para filtrar los costos por mes de cada compañía.

costos_source = costos_.groupby(['orden_mes','source_id'])['costs'].sum().re
```

In [114…
```
# Se genera una copía de la tabla de reporte.

reporte_ = reporte.copy()
```

In [115…
```
# Se agrega una columna para cambiar el tipo de fecha a periodo mensual.

reporte_['orden_mes'] = reporte_['orden_mes'].dt.to_period('M')
```

In [116…
```
costos_ltv = pd.merge(reporte_, costos_source, on = 'orden_mes')
```

In [117…
```
# Se crea una columna nueva entre los costos por mes de cada campaña entre e

costos_ltv['ltvxsource'] = costos_ltv['costs']/costos_ltv['n_buyers']
```

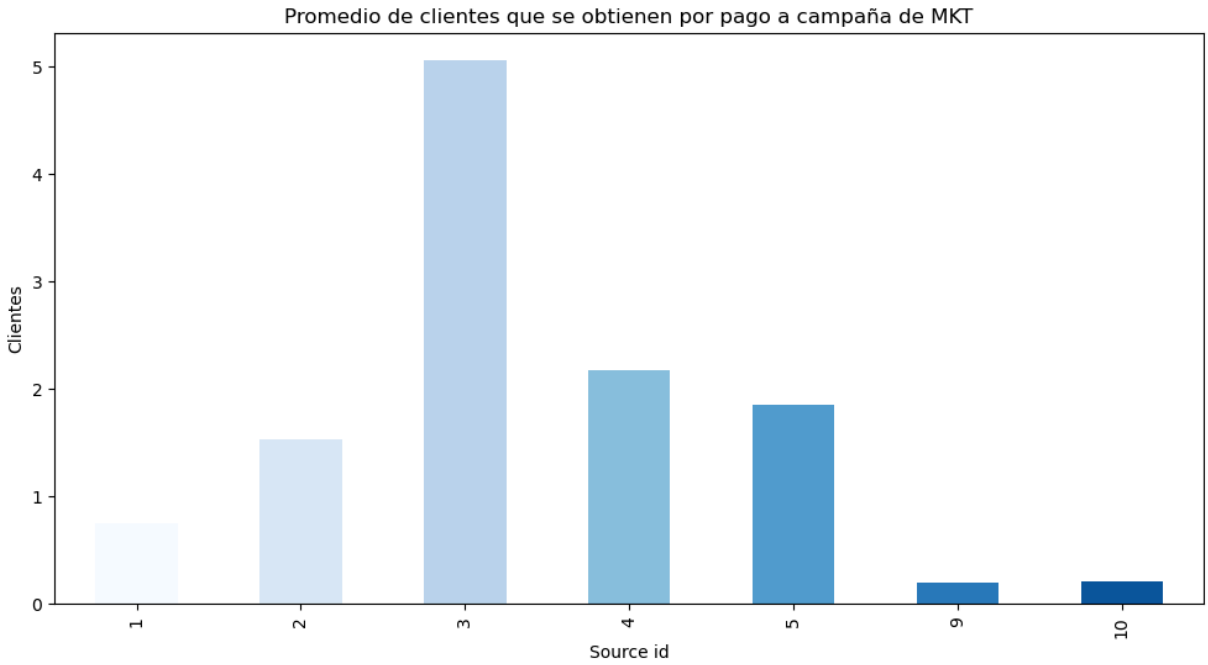
In [326…
```
promedio_ltv_source = costos_ltv.groupby('source_id')['ltvxsource'].mean()
```

In [328…
```
promedio_ltv_source
```

Out[328…
```
source_id
1     0.748166
2     1.529077
3     5.058681
4     2.175752
5     1.855744
9     0.197126
10    0.207584
Name: ltvxsource, dtype: float64
```

In [332…
```
colorprom = cm.Blues
colors = [colorprom(i / len(promedio_ltv_source)) for i in range(len(promedi

promedio_ltv_source.plot(
    kind = 'bar',
    figsize = (12,6),
    color = colors
)
plt.xlabel('Source id')
plt.ylabel('Clientes')
```

```
plt.title('Promedio de clientes que se obtienen por pago a campaña de MKT')
plt.show()
```



EL promedio del numero de compradores que se nos trajo cada vez que pagamos una de las publicación de marketing.
La compañía de mkt que más cliente nos trajo es la de la compañía 3, con un promedio de 5 clientes cada vez que realizamos un pago, seguido por la 4 y 5 que nos trajo en promedio 2 clientes, el detalle esta en que 5 de nuestras campañas no nos traen clientes.

Marketing:

1. ¿Cuánto dinero se gastó? (Total/por fuente de adquisición/a lo largo del tiempo)
2. ¿Cuál fue el costo de adquisición de clientes de cada una de las fuentes?
3. ¿Cuán rentables eran las inversiones? (ROMI)

In [265…
```
# Se crean nuevas columnas en la tabla de costos_ para el día, semana y mes.

costos_['dia'] = costos_['dt']
costos_['semana'] = costos_['dt'].dt.isocalendar().week
costos_['mes'] = costos_['dt'].dt.month
```

In [267…
```
# Se suman los costos totales, el costo por fuente anual y el promedio del c

costo_total = costos_['costs'].sum()
```

In [127…
```
costo_por_fuente_anual = costos_.groupby('source_id')['costs'].sum()
```

In [128…
```
costo_promedio_por_fuente = costos_.groupby('source_id')['costs'].mean()
```

¿Cuánto dinero se gastó? (Total/por fuente de adquisición/a lo largo del tiempo)

In [130…
```
print('El costo del pago total de publicidad fue de: $',costo_total)
print(f'El costo del pago total de publicidad por fuente fue de: \n',costo_p
print(f'El costo del pago promedio de publicidad por fuente fue de: \n',rour
```

```
El costo del pago total de publicidad fue de: $ 329131.62
El costo del pago total de publicidad por fuente fue de:
 source_id
1      20833.27
2      42806.04
3     141321.63
4      61073.60
5      51757.10
9       5517.49
10      5822.49
Name: costs, dtype: float64
El costo del pago promedio de publicidad por fuente fue de:
 source_id
1      57
2     118
3     389
4     168
5     142
9      15
10     16
Name: costs, dtype: int64
```

In [269…
```
# Se obtiene el total de clientes que se tuvieron durante el año.

total_clientes = cohorts_sizes['n_buyers'].sum()
print(total_clientes)
```

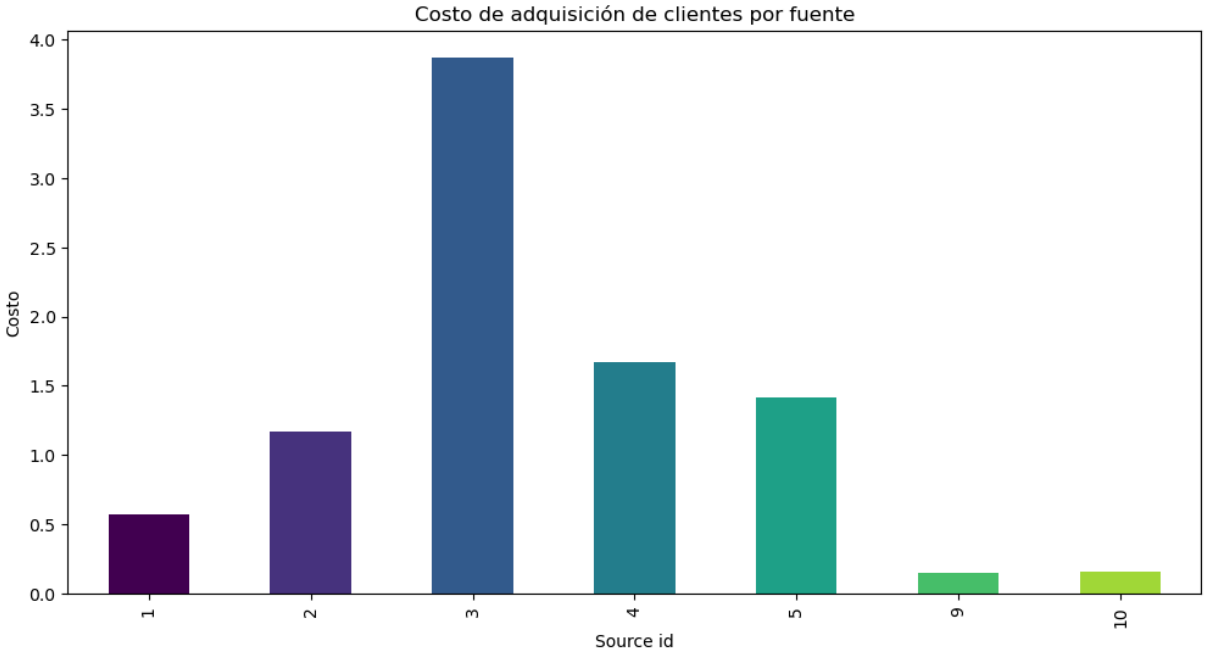
```
36523
```

In [320…
```
# Se obtiene el costo de obtención de clientes por cada compañía de mkt.

cac_source = (costo_por_fuente_anual / total_clientes)
```

In [322…
```
cac_source
```

Out[322…
```
source_id
1     0.570415
2     1.172030
3     3.869387
4     1.672196
5     1.417110
9     0.151069
10    0.159420
Name: costs, dtype: float64
```

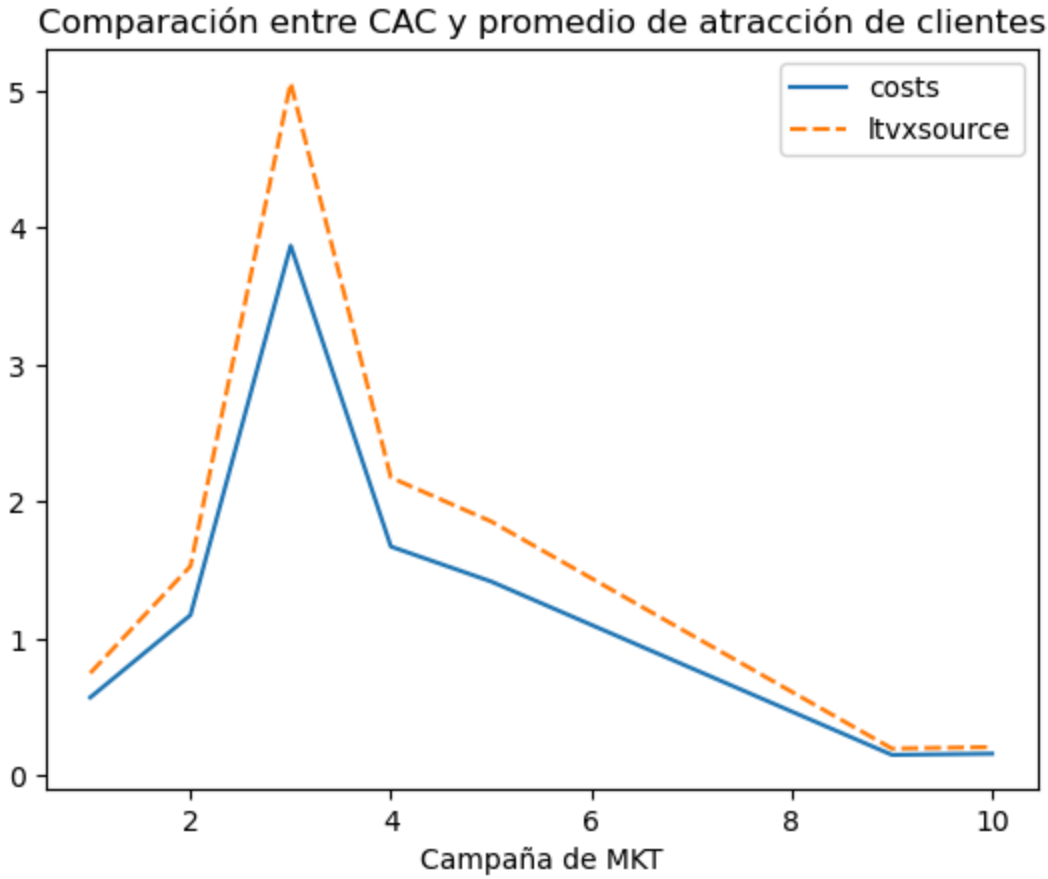
In [324…
```
# Se crea un gráfico para visualizar al CAC por cada compañia de MKT.

colormap = cm.viridis
colores = [colormap(i / len(cac_source)) for i in range(len(cac_source))]

cac_source.plot(
    kind = 'bar',
    figsize = (12,6),
    color = colores
)
plt.xlabel('Source id')
plt.ylabel('Costo')
plt.title('Costo de adquisición de clientes por fuente')
plt.show()
```



En esta tabla se representa el costo de adquisición por cliente de cada una de las compañías de MKT, la compañía que más nos cobra es la 3, nos cobra `$` 3.90 por cliente, seguida de las campañas 4 y 5 con un CAC de `$` 1.67 y `$` 1.41 por cliente, respectivamente.

In [344…
```
combine = pd.merge(cac_source, promedio_ltv_source, on = 'source_id')
```

In [360…
```
sns.lineplot(data = combine)
plt.title('Comparación entre CAC y promedio de atracción de clientes')
plt.xlabel('Campaña de MKT')
plt.show()
```



De hecho las tablas son muy parecidas, la del promedio de atracción de clientes por campaña publicitaria y las que más nos cuestan son las que más clientes nos atraen.

In [364…
```
# Se calcula el ingreso total.

ingreso_total = reporte_['revenue'].sum()
print(ingreso_total)
```

```
252057.2
```

In [366…
```
# Se calcula el beneficio entre las facturas totales menos el costo total de

beneficio = ingreso_total - costo_total
print(beneficio)
```

```
-77074.41999999998
```

Como se había visto en la tabla de atraccion de clientes que las diferentes campañas publicitarias no estaban funcionando, aqui se ve reflejado que estamos destinando mucho dinero en campañas publicitarias que no nos están generando los clientes que ocupamos para poder obtener un beneficio, ya que tenemos una perdida de $ 77,074 dólares.
¿Serán todas nuestras campañas publicitarias las que están fallando o si habrán algunas rescatables?

In [138…
```
def calcular_romi(ingreso_total, costo_por_fuente_anual, costo_total):
    beneficio_neto = ingreso_total - costo_por_fuente_anual
    romi = (beneficio_neto / costo_total) * 100
    return romi
```

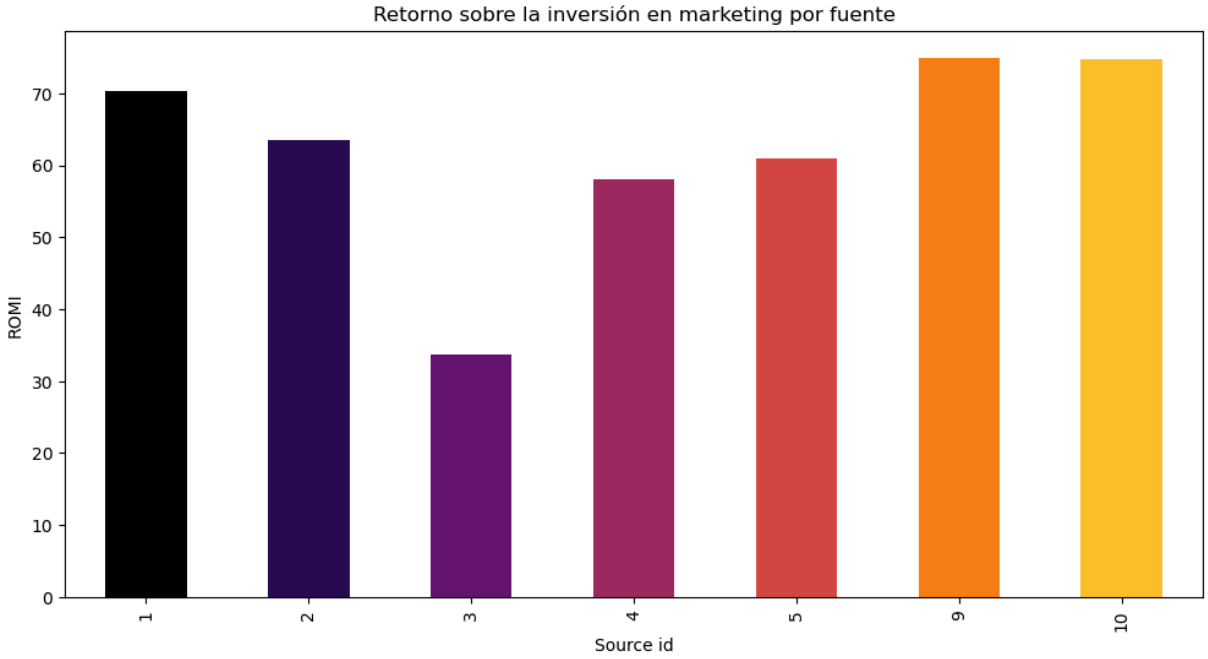
¿Cuán rentables eran las inversiones? (ROMI)

In [371…
```
# Se calcula el retorno de inversión por camapaña de MKT.

romi = round(calcular_romi(ingreso_total, costo_por_fuente_anual, costo_tota
```

In [140…
```
romi
```

Out[140…
```
source_id
1     70.25
2     63.58
3     33.64
4     58.03
5     60.86
9     74.91
10    74.81
Name: costs, dtype: float64
```

In [141…
```
colormap = cm.inferno
colores = [colormap(i / len(romi)) for i in range(len(romi))]

romi.plot(
    kind = 'bar',
    figsize = (12,6),
    color = colores
)
plt.xlabel('Source id')
plt.ylabel('ROMI')
plt.title('Retorno sobre la inversión en marketing por fuente')
plt.show()
```



Este gráfico nos muestra el ROMI de cada una de nuestras campañas publicitarias y nos indica que en ninguna de las campañas hemos recuperado la inversión que hemos realizado, irónicamente las campañas que menos clientes nos han traido muestran una mejor ROMI, pero esto es por el bajo costo que representan en nuestro gasto de publicidad.

Nuestra campaña publicitaría no. 3, que es la que nos cuesta más y que más clientes nos genera, solo nos esta dando un rendimiento del 33.64%, es decir que estamos muy por debajo de recuperar la inversión que hemos realizado en esta campaña. Tendríamos que estar obteniendo en promedio 15 clientes por medio de esta compañia cada vez que se realiza un pago publicitario para que por lo menos estuvieramos recuperando nuestra inversion.

Se tiene que considerar seriamente cambiar al equipo de MKT, para un análisis profundo de esos canales de publicidad para ver de que manera se pudieran mejorar nuestras ventas y la retención de nuestros clientes.