2c187956-0605-4e4a-bb64-2a284c9b4c4d-Copy1

November 6, 2024

1 SPRINT 9. Analisis de Showz

2 Índice

2.1 Introducción

El presente análisis tiene como objetivo principal optimizar la inversión en marketing de Showz, una empresa de venta de entradas para eventos, con el fin de maximizar el retorno de la inversión (ROI) y mejorar la eficiencia de las campañas publicitarias. A través de un análisis exhaustivo de los datos históricos de visitas, pedidos y gastos de marketing, se busca identificar las estrategias más efectivas para adquirir y retener clientes.

Para lograr este objetivo, se llevará a cabo un análisis detallado de los siguientes aspectos:

- Se estudiará cómo los usuarios interactúan con la plataforma de Showz, identificando patrones de navegación, duración de las sesiones y frecuencia de retorno.
- Se analizará el tiempo que transcurre desde la primera visita hasta la realización de una compra, así como los factores que influyen en la decisión de compra.
- Se evaluará el rendimiento de las diferentes fuentes de adquisición de clientes, calculando el costo de adquisición por cliente y el retorno sobre la inversión.
- Se identificarán grupos de clientes con características y comportamientos similares para adaptar las estrategias de marketing de manera más efectiva.

2.2 Analisis de datos

[1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import datetime
```

[2]:
```
visitas = pd.read_csv("/datasets/visits_log_us.csv")
pedidos = pd.read_csv("/datasets/orders_log_us.csv")
gastos_marketing = pd.read_csv("/datasets/costs_us.csv")
```

[3]:
```
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  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
```

[4]:
```
nuevo_visitas= []
for viejo_df in visitas.columns:
    lower = viejo_df.lower()
    space = lower.replace(' ','_')
    nuevo_visitas.append(space)
visitas.columns = nuevo_visitas

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  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
```

[5]:
```
print(visitas.duplicated().value_counts())
print(visitas.isnull().sum())
```

```
False    359400
dtype: int64
device       0
end_ts       0
source_id    0
start_ts     0
uid          0
dtype: int64
```

[6]:
```
visitas['start_ts'] = pd.to_datetime(visitas['start_ts'])
visitas['end_ts'] = pd.to_datetime(visitas['end_ts'])
```

[7]:
```
pedidos.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  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
```

[8]:
```
nuevo_pedidos = []
for viejo_df in pedidos.columns:
    lower = viejo_df.lower()
    space = lower.replace(' ','_')
    nuevo_pedidos.append(space)
pedidos.columns = nuevo_pedidos

pedidos.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  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
```

[9]:
```
print(pedidos.duplicated().value_counts())
print(pedidos.isna().sum())
```

```
False    50415
dtype: int64
buy_ts     0
revenue    0
uid        0
dtype: int64
```

[10]:
```
pedidos["buy_ts"] = pd.to_datetime(pedidos["buy_ts"])
```

[11]:
```
gastos_marketing.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   object
 2   costs      2542 non-null   float64
dtypes: float64(1), int64(1), object(1)
memory usage: 59.7+ KB
```

[12]:
```
print(gastos_marketing.duplicated().value_counts())
print(gastos_marketing.isna().sum())
```

```
False    2542
dtype: int64
source_id    0
dt           0
costs        0
dtype: int64
```

[13]:
```
gastos_marketing["dt"] = pd.to_datetime(gastos_marketing["dt"])
```

Se organizaron los datos, establaciendo el tipo de dato correcto para cada columna, verificando si existen duplicados o valores ausentes y se modificaron los nombres de columnas para optimizar el analisis

2.3 Visitas

2.3.1 Cuántas personas lo usan cada día, semana y mes

[14]:
```
visitas['dia'] = visitas['start_ts'].dt.date
visitas['semana'] = visitas['start_ts'].dt.isocalendar().week
visitas['mes_año'] = visitas['start_ts'].dt.to_period('M')
```

[15]:
```
# Usuarios únicos por día
usuarios_dia = visitas.groupby('dia')['uid'].nunique()
print(usuarios_dia.sort_values(ascending=False).head())

# Usuarios únicos por semana
usuarios_semana = visitas.groupby('semana')['uid'].nunique()
print(usuarios_semana.sort_values(ascending=False).head())

# Usuarios únicos por mes
usuarios_por_mes_año = visitas.groupby('mes_año')['uid'].nunique()
usuarios_por_mes_año = usuarios_por_mes_año.sort_index()
print(usuarios_por_mes_año.sort_values(ascending=False).head())
```

```
dia
2017-11-24    3319
2018-05-31    1997
2017-11-25    1817
2018-02-01    1640
2018-03-26    1609
Name: uid, dtype: int64
semana
47    10586
49     8407
50     8214
48     8166
46     8117
Name: uid, dtype: int64
mes_año
2017-11    32797
2017-12    31557
2017-10    29692
2018-02    28749
2018-01    28716
Freq: M, Name: uid, dtype: int64
```

Usuarios únicos por día: El día con mayor cantidad de usuarios únicos fue el 2017-11-24. Los siguientes días con mayor cantidad de usuarios se concentran en los meses de noviembre de 2017

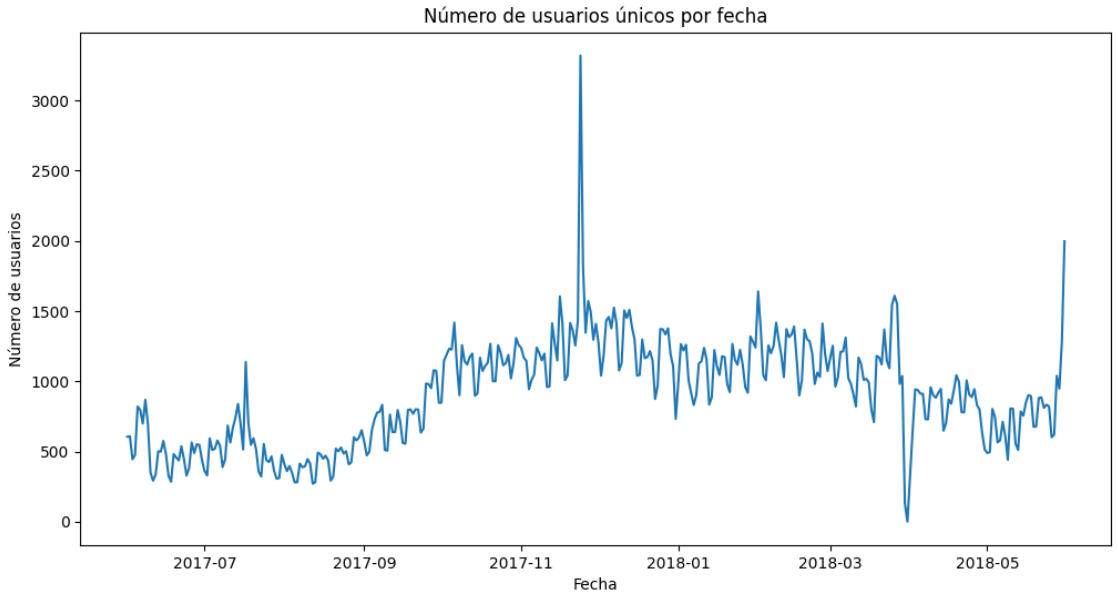
y febrero y marzo de 2018. Esto sugiere que pudo haber habido eventos o campañas específicas durante esos períodos que impulsaron la adquisición de nuevos usuarios.

Usuarios únicos por semana: Las semanas 47, 49 y 50 del año fueron las que registraron el mayor número de usuarios únicos. Esto podría indicar que hubo un período de alta actividad hacia finales de año y principios del año siguiente.

Usuarios únicos por mes: Los meses de noviembre y diciembre de 2017, así como febrero de 2018, fueron los que registraron el mayor número de usuarios únicos. Esto refuerza la idea de que hubo un período de alta actividad hacia finales de 2017 y principios de 2018.

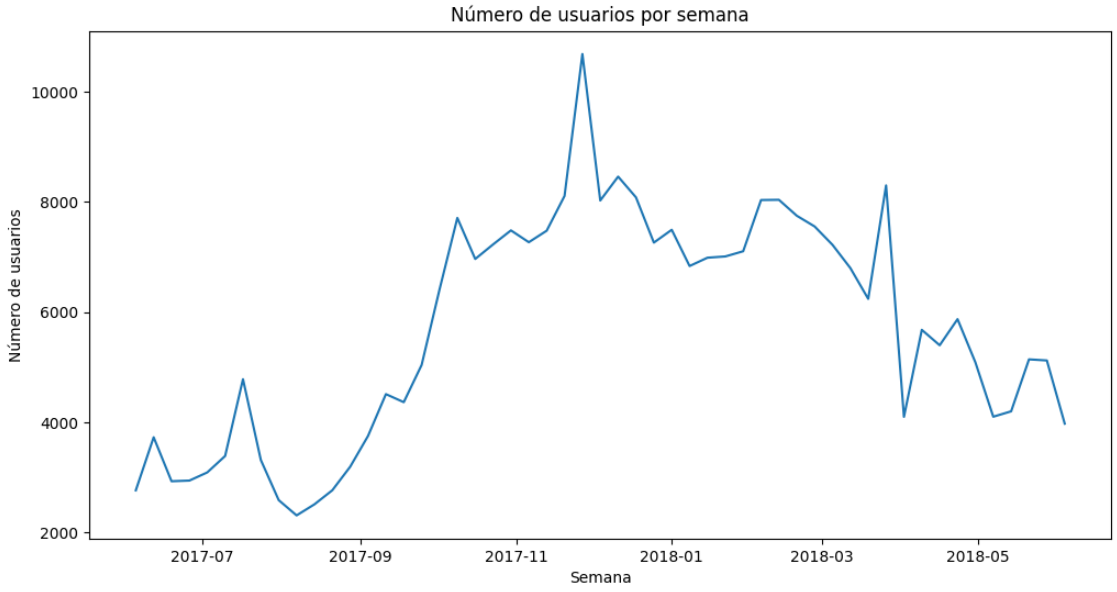
Graficos para las personas Showz usan cada día, semana y mes

[16]:
```
visitas = visitas.sort_values(by='start_ts')

# Usuarios por día
usuarios_por_fecha = visitas.groupby(visitas['start_ts'].dt.date)['uid'].
 ↪nunique()

# Gráfico de usuarios por fecha
plt.figure(figsize=(12, 6))
plt.plot(usuarios_por_fecha.index, usuarios_por_fecha.values)
plt.title('Número de usuarios únicos por fecha')
plt.xlabel('Fecha')
plt.ylabel('Número de usuarios')
plt.show()
```



Posee un crecimiento gradual a largo plazo, parece haber una tendencia positiva en el número de usuarios.

[17]:
```
# Usuarios por semana
usuarios_semana = visitas.resample('W-MON', on='start_ts')['uid'].nunique()

# Gráfico de usuarios por semana
plt.figure(figsize=(12, 6))
plt.plot(usuarios_semana.index, usuarios_semana.values)
plt.title('Número de usuarios por semana')
plt.xlabel('Semana')
plt.ylabel('Número de usuarios')
plt.show()
```



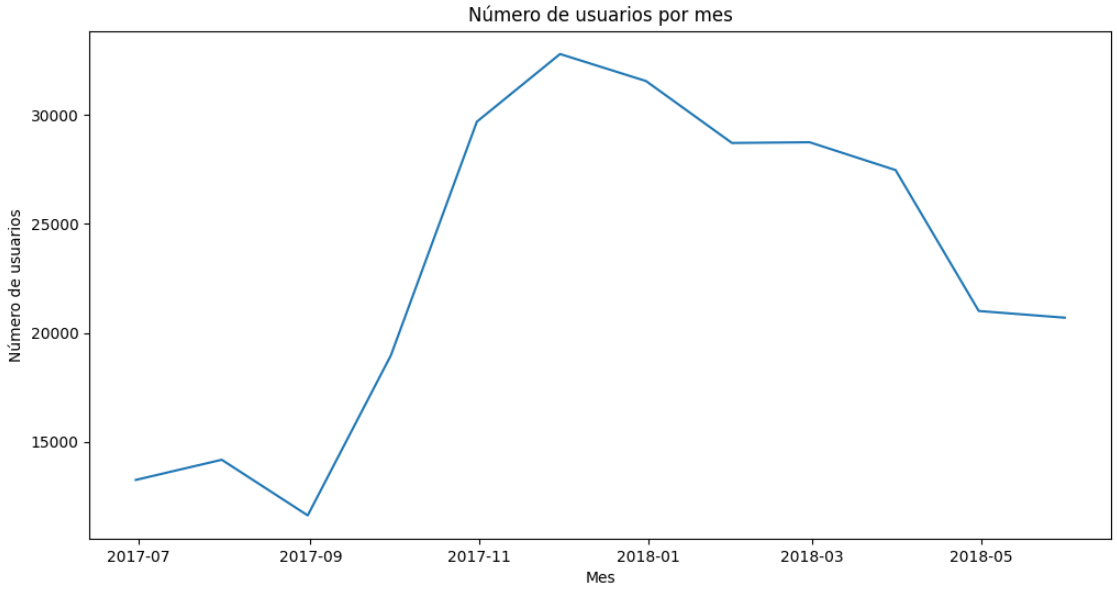
Con este grafico se pueden concluir pero a la vez hacerse varias preguntas:

*¿Qué eventos o campañas se llevaron a cabo durante los períodos de mayor actividad? Identificar estos eventos puede ayudar a entender qué estrategias son efectivas para atraer nuevos usuarios.

*¿Existen patrones estacionales? Analizar si hay variaciones significativas en el número de usuarios durante ciertas épocas del año que pueden coincidir con eventos especificos.

[18]:
```
# Usuarios por mes
usuarios_mes = visitas.resample('M', on='start_ts')['uid'].nunique()
usuarios_mes = usuarios_mes.sort_index()

# Gráfico de usuarios por mes
plt.figure(figsize=(12, 6))
plt.plot(usuarios_mes.index, usuarios_mes.values)
plt.title('Número de usuarios por mes')
plt.xlabel('Mes')
plt.ylabel('Número de usuarios')
```

```
plt.show()
```



Tiene una tendencia ascendente, claramente se observa un crecimiento sostenido en el número de usuarios a lo largo de los meses analizados.

Hay un crecimiento más pronunciado entre finales de 2017 y principios de 2018, lo que sugiere que se implementaron estrategias exitosas o que ocurrieron eventos que impulsaron el crecimiento durante ese período.Aunque no es tan evidente como en un gráfico semanal, podrían existir patrones estacionales si se analiza un período más largo.

Sin embargo, se observa una ligera desaceleración en el crecimiento hacia el final del período analizado. Esto podría deberse a diversos factores, como saturación del mercado, cambios en la competencia o ajustes en las estrategias de marketing.

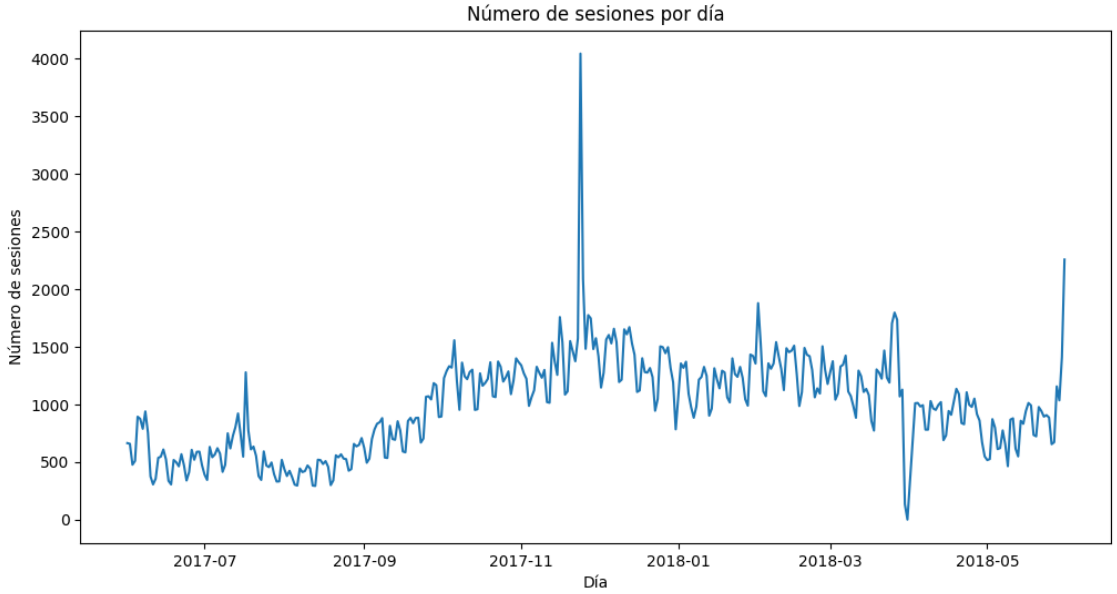
2.3.2 Sesiones por día

[19]:
```
sesiones_por_dia = visitas.groupby(visitas['start_ts'].dt.date)['uid'].count()
print(sesiones_por_dia)
```

```
start_ts
2017-06-01     664
2017-06-02     658
2017-06-03     477
2017-06-04     510
2017-06-05     893
              ...
2018-05-27     672
2018-05-28    1156
2018-05-29    1035
```

```
2018-05-30    1410
2018-05-31    2256
Name: uid, Length: 364, dtype: int64
```

[20]:
```
plt.figure(figsize=(12, 6))
plt.plot(sesiones_por_dia.index, sesiones_por_dia.values)
plt.title('Número de sesiones por día')
plt.xlabel('Día')
plt.ylabel('Número de sesiones')
plt.show()
```



Este grafico, al igual qeu el grafico de la cantidad de sesiones de usuarios unicos al dia reflejan un incremento a lo largo del tiempo.

2.3.3 Duración de cada sesión en minutos

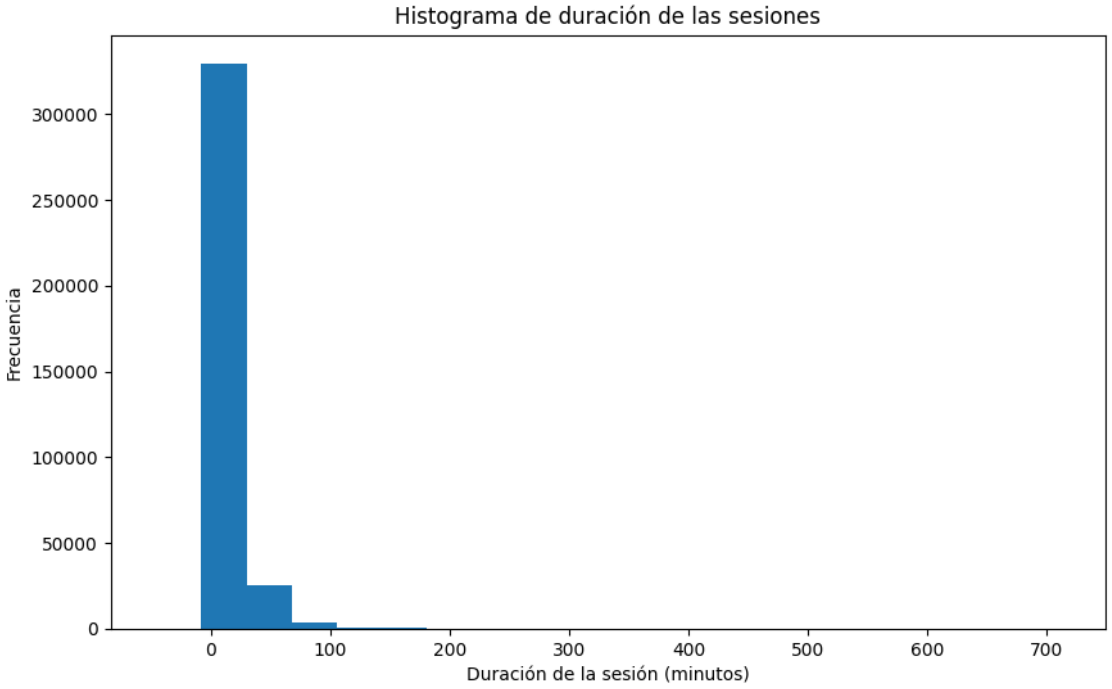
[21]:
```
visitas['sesion_duracion'] = (visitas['end_ts'] - visitas['start_ts']).dt.
 ↪total_seconds() / 60
visitas
```

[21]:

	device	end_ts	source_id	start_ts	\
308527	desktop	2017-06-01 00:02:00	5	2017-06-01 00:01:00	
260646	desktop	2017-06-01 00:02:00	3	2017-06-01 00:02:00	
245715	desktop	2017-06-01 00:16:00	3	2017-06-01 00:02:00	
235930	desktop	2017-06-01 00:04:00	3	2017-06-01 00:04:00	
11727	desktop	2017-06-01 00:11:00	1	2017-06-01 00:09:00	
…	…	…	…	…	
299620	desktop	2018-06-01 00:04:00	4	2018-05-31 23:59:00	

```
198329  desktop 2018-06-01 00:11:00          3 2018-05-31 23:59:00
269213    touch 2018-06-01 00:12:00         10 2018-05-31 23:59:00
294423  desktop 2018-05-31 23:59:00          2 2018-05-31 23:59:00
263781  desktop 2018-06-01 00:25:00          1 2018-05-31 23:59:00

                         uid         dia  semana  mes_año  sesion_duracion
308527  13890188992670018146  2017-06-01      22  2017-06              1.0
260646  16152015161748786004  2017-06-01      22  2017-06              0.0
245715  16706502037388497502  2017-06-01      22  2017-06             14.0
235930   8842918131297115663  2017-06-01      22  2017-06              0.0
11727   10329302124590727494  2017-06-01      22  2017-06              2.0
...                      ...         ...     ...      ...              ...
299620     83872787173869366  2018-05-31      22  2018-05              5.0
198329   3720373600909378583  2018-05-31      22  2018-05             12.0
269213  10723414689244282024  2018-05-31      22  2018-05             13.0
294423  10406407303624848652  2018-05-31      22  2018-05              0.0
263781   4906562732540547408  2018-05-31      22  2018-05             26.0

[359400 rows x 9 columns]
```

[22]:
```
plt.figure(figsize=(10, 6))
plt.hist(visitas['sesion_duracion'], bins=20)
plt.title('Histograma de duración de las sesiones')
plt.xlabel('Duración de la sesión (minutos)')
plt.ylabel('Frecuencia')
plt.show()
```



La gran mayoría de las sesiones tienen una duración muy corta, concentrándose en los primeros minutos. Esto sugiere que muchos usuarios realizan visitas rápidas al sitio web.A medida que aumenta la duración de la sesión, la frecuencia disminuye drásticamente. Hay muy pocas sesiones que superen los 100 minutos.

La forma del histograma indica una distribución sesgada hacia la derecha. Esto significa que hay una cola larga hacia los valores altos, lo que confirma la presencia de algunas sesiones muy largas, aunque sean pocas.

Sin embargo, si hay problemas de usabilidad en el sitio, los usuarios podrían abandonar las sesiones rápidamente al encontrar dificultades para navegar o encontrar lo que buscan.

2.3.4 Tasa de retorno

[23]:
```
semana = 7

visitas['ultima_visita'] = visitas.groupby('uid')['start_ts'].diff()

visitas['retorno'] = visitas['ultima_visita'] <= pd.Timedelta(days=semana)

retorno_usuarios = visitas['retorno'].sum()
total_usuarios = visitas['uid'].nunique()
tasa_retorno = (retorno_usuarios / total_usuarios) * 100

print("Tasa de retorno en", semana, "días:", tasa_retorno, "%")
```

```
Tasa de retorno en 7 días: 31.45694638623126 %
```

Un 31% de tasa de retorno en una semana sugiere que una parte significativa de los usuarios vuelve a la plataforma con regularidad.

Una tasa de retorno relativamente alta puede indicar un buen nivel de engagement por parte de los usuarios. Están encontrando valor en la plataforma y están dispuestos a volver.

Esta tasa también puede ser un indicador temprano de la capacidad de la plataforma para fidelizar a los usuarios.

2.4 Pedidos

2.4.1 Tiempo entre la primera visita y la primera compra

[24]:
```
# primera visita
primeras_visitas = visitas.groupby('uid')['start_ts'].min()

# primera compra
primeras_compras = pedidos.groupby('uid')['buy_ts'].min()
```

```
conversiones = pd.merge(primeras_visitas, primeras_compras, on='uid',␣
 ↪how='left')

# tiempo de conversión en días
conversiones['tiempo_conversion'] = (conversiones['buy_ts'] -␣
 ↪conversiones['start_ts']).dt.days

conversiones
```

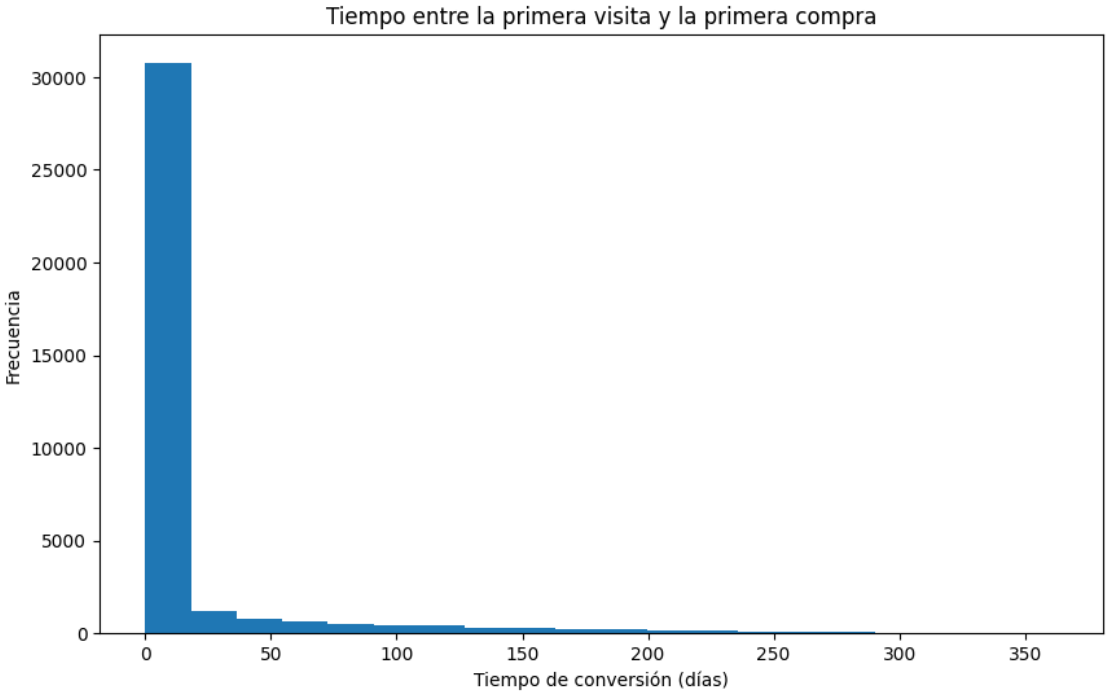
[24]:
```
                                start_ts              buy_ts  \
uid
11863502262781       2018-03-01 17:27:00                 NaT
49537067089222       2018-02-06 15:55:00                 NaT
297729379853735      2017-06-07 18:47:00                 NaT
313578113262317      2017-09-18 22:49:00 2018-01-03 21:51:00
325320750514679      2017-09-30 14:29:00                 NaT
…                                    …                   …
18446403737806311543 2017-11-30 03:36:00                 NaT
18446424184725333426 2017-12-06 20:32:00                 NaT
18446556406699109058 2018-01-01 16:29:00                 NaT
18446621818809592527 2017-12-27 13:27:00                 NaT
18446676030785672386 2017-10-04 16:01:00                 NaT

                     tiempo_conversion
uid
11863502262781                     NaN
49537067089222                     NaN
297729379853735                    NaN
313578113262317                  106.0
325320750514679                    NaN
…                                  …
18446403737806311543               NaN
18446424184725333426               NaN
18446556406699109058               NaN
18446621818809592527               NaN
18446676030785672386               NaN

[228169 rows x 3 columns]
```

Comentario de Revisor v2

Correcto, muy bien! Nota que algunos usuarios no tienen tiempo de conversión ya que nunca han comprado.

[25]:
```
# histograma del tiempo de conversión
plt.figure(figsize=(10, 6))
plt.hist(conversiones['tiempo_conversion'], bins=20)
plt.title('Tiempo entre la primera visita y la primera compra')
```

```
plt.xlabel('Tiempo de conversión (días)')
plt.ylabel('Frecuencia')
plt.show()
```



Dado que una gran parte de las conversiones ocurre en un período corto, es crucial optimizar la experiencia de compra inicial para facilitar la conversión. Esto incluye asegurarse de que el proceso de compra sea sencillo, rápido y seguro.

Implementar estrategias de retargeting para llegar a los usuarios que no han realizado una compra después de un período de tiempo determinado. Esto puede incluir campañas de email marketing, anuncios en redes sociales o mostrar anuncios personalizados en otros sitios web.

Como recomendacion se puede observar qué lleva a un usuario a demorar tanto su conversión, ver cuántos usuarios no convierten y por qué.

Nuevos usuarios por dia y sus fuentes

[26]:
```
pedidos['ordenes_dia'] = pedidos['buy_ts'].astype('datetime64[D]')


primera_orden = pedidos.groupby('uid').agg({'ordenes_dia': 'min'}).reset_index()
primera_orden.columns = ['uid','dia_primera_orden']
cohort_dia = primera_orden.groupby('dia_primera_orden').agg({'uid': 'nunique'}).
 ↪reset_index()
```

```
cohort_dia.columns = ['dia_primera_orden' , 'uid']


cohort_dia = cohort_dia.sort_values('uid', ascending=False)
print(cohort_dia.head())
```

```
    dia_primera_orden  uid
176        2017-11-24  597
362        2018-05-31  492
127        2017-10-06  298
245        2018-02-01  254
177        2017-11-25  225
```

[27]:
```
df_dia = pd.merge(primera_orden, visitas, on='uid', how='inner')

fuentes_por_cohorte_dia = df_dia.groupby(['dia_primera_orden', 'source_id']).
 ↪agg({'uid': 'nunique'}).reset_index()
fuentes_por_cohorte_dia.columns = ['dia_primera_compra', 'fuente', 'usuarios']

fuentes_por_cohorte_dia = fuentes_por_cohorte_dia.sort_values('usuarios',␣
 ↪ascending=False)

print(fuentes_por_cohorte_dia)
```

```
     dia_primera_compra  fuente  usuarios
1212         2017-11-24       3       265
1213         2017-11-24       4       239
1210         2017-11-24       1       221
2499         2018-05-31       2       218
1211         2017-11-24       2       203
…                    …     …         …
2299         2018-05-01      10         1
2293         2018-04-30      10         1
2292         2018-04-30       9         1
122          2017-06-18      10         1
2505         2018-06-01       4         1

[2506 rows x 3 columns]
```

Las fuentes con mayor número de “usuarios” en general suelen ser las más efectivas en términos de adquisición de nuevos clientes.Al analizar la evolución de cada fuente a lo largo del tiempo, se pueden identificar tendencias y cambios en su desempeño.

Si se observa una variación significativa en el número de usuarios nuevos a lo largo del tiempo, podría indicar la existencia de factores estacionales que influyen en el comportamiento de los usuarios (por ejemplo, campañas de marketing específicas, eventos estacionales).

Nuevos usuarios por semana y sus fuentes

[28]:
```
pedidos['ordenes_semana'] = pedidos['buy_ts'].astype('datetime64[W]')


primera_orden_semana = pedidos.groupby('uid').agg({'ordenes_semana': 'min'}).
 ↪reset_index()
primera_orden_semana.columns = ['uid','semana_primera_orden']
cohort_semana = primera_orden_semana.groupby('semana_primera_orden').agg({'uid':
 ↪ 'nunique'}).reset_index()


cohort_semana.columns = ['semana_primera_orden' , 'uid']

cohort_semana = cohort_semana.sort_values('uid', ascending=False)
print(cohort_semana.head())
```

```
   semana_primera_orden   uid
25           2017-11-23  1455
18           2017-10-05  1213
17           2017-09-28  1048
30           2017-12-28  1030
27           2017-12-07  1027
```

[29]:
```
df_semana = pd.merge(primera_orden_semana, visitas, on='uid', how='inner')

fuentes_por_cohorte_semana = df_semana.groupby(['semana_primera_orden',␣
 ↪'source_id']).agg({'uid': 'nunique'}).reset_index()
fuentes_por_cohorte_semana.columns = ['semana_primera_compra', 'fuente',␣
 ↪'usuarios']

fuentes_por_cohorte_semana = fuentes_por_cohorte_semana.sort_values('usuarios',␣
 ↪ascending=False)

print(fuentes_por_cohorte_semana)
```

```
    semana_primera_compra  fuente  usuarios
178            2017-11-23       3       640
179            2017-11-23       4       605
129            2017-10-05       3       480
207            2017-12-21       4       466
130            2017-10-05       4       460
..                    …     …       …
69             2017-08-03      10        10
62             2017-07-27      10        10
370            2018-05-31       9         9
76             2017-08-10      10         5
82             2017-08-17       7         1
```

```
[372 rows x 3 columns]
```

En este caso se puede ver como las fuentes 3 y 4 son las mas efectivas para atraer clientes. Si se ejecutan campañas de marketing específicas durante ciertas semanas, podemos evaluar su impacto en la adquisición de nuevos usuarios y comparar su efectividad y al observar las semanas con mayor y menor número de nuevos usuarios, podemos identificar períodos de alta y baja actividad y ajustar nuestras estrategias de marketing en consecuencia.

Nuevos usuarios por mes y sus fuentes

[30]:
```
pedidos['ordenes_mes'] = pedidos['buy_ts'].astype('datetime64[M]')


primera_orden_mes = pedidos.groupby('uid').agg({'ordenes_mes': 'min'}).
 ↪reset_index()
primera_orden_mes.columns = ['uid','mes_primera_orden']
cohort_mes = primera_orden_mes.groupby('mes_primera_orden').agg({'uid':␣
 ↪'nunique'}).reset_index()# escribe tu código aquí


cohort_mes.columns = ['mes_primera_orden' , 'uid']

cohort_mes = cohort_mes.sort_values('uid', ascending=False)
print(cohort_mes.head())
```

```
  mes_primera_orden   uid
6        2017-12-01  4383
4        2017-10-01  4340
5        2017-11-01  4081
8        2018-02-01  3651
9        2018-03-01  3533
```

[31]:
```
df_mes = pd.merge(primera_orden_mes, visitas, on='uid', how='inner')

fuentes_por_cohorte_mes = df_mes.groupby(['mes_primera_orden', 'source_id']).
 ↪agg({'uid': 'nunique'}).reset_index()
fuentes_por_cohorte_mes.columns = ['mes_primera_compra', 'fuente', 'usuarios']

fuentes_por_cohorte_mes = fuentes_por_cohorte_mes.sort_values('usuarios',␣
 ↪ascending=False)

print(fuentes_por_cohorte_mes)
```

```
   mes_primera_compra  fuente  usuarios
46         2017-12-01       4      1897
39         2017-11-01       4      1766
45         2017-12-01       3      1733
32         2017-10-01       4      1692
38         2017-11-01       3      1677
```

```
..                 …      …        …
76          2018-04-01      9       81
13          2017-07-01     10       71
21          2017-08-01     10       56
19          2017-08-01      7        1
85          2018-06-01      4        1

[86 rows x 3 columns]
```

Si se observan picos o caídas en el número de nuevos usuarios en ciertos meses, se pueden identificar patrones estacionales y ajustar las estrategias de marketing en consecuencia.

Nuestros datos revelan que las fuentes 3 y 4 son las que generan un mayor número de nuevos usuarios en Showz. Esto indica que estas fuentes están conectando con nuestra audiencia objetivo de manera más efectiva. Para maximizar nuestro crecimiento, proponemos aumentar la inversión en estas fuentes y explorar nuevas estrategias para potenciar las demás.

2.4.2 Cantidad de pedidos realizados y tamaño promedio de compra durante 2 trimestres diferentes

[32]:
```
mascarilla = (pedidos['buy_ts'] >= '2017-06-01') & (pedidos['buy_ts'] <=␣
 ↪'2017-08-31')
df_filtro = pedidos.loc[mascarilla]

# Contar los pedidos
num_pedidos = len(df_filtro)
print("Número total de pedidos entre octubre y diciembre de 2017:", num_pedidos)

promedio_compra = df_filtro['revenue'].mean()

print("El tamaño promedio de compra entre octubre y diciembre de 2017 es:",␣
 ↪promedio_compra)
```

```
Número total de pedidos entre octubre y diciembre de 2017: 6421
El tamaño promedio de compra entre octubre y diciembre de 2017 es:
4.696530135492914
```

[33]:
```
mascarilla = (pedidos['buy_ts'] >= '2018-03-01') & (pedidos['buy_ts'] <=␣
 ↪'2018-05-31')
df_filtro = pedidos.loc[mascarilla]

# Contar los pedidos
num_pedidos = len(df_filtro)
print("Número total de pedidos entre marzo y mayo de 2018:", num_pedidos)

promedio_compra = df_filtro['revenue'].mean()
```

```
print("El tamaño promedio de compra entre marzo y mayo de 2018 es:",␣
 ↪promedio_compra)
```

```
Número total de pedidos entre marzo y mayo de 2018: 12335
El tamaño promedio de compra entre marzo y mayo de 2018 es: 5.011326307255776
```

En cuanto al número de pedidos, se ha producido un aumento considerable, pasando de 6421 en 2017 a 12335 en 2018. Esto representa un incremento de casi el 92%, lo que indica una mayor demanda y un posible éxito en las estrategias de adquisición de clientes o en la fidelización de los existentes.

El tamaño promedio de compra también ha experimentado un crecimiento, aunque más moderado, pasando de 4.70 en 2017 a 5.01 en 2018. Este aumento del 6.6% sugiere que los clientes están gastando ligeramente más por pedido, lo que podría deberse a un aumento general en el valor percibido de los productos o servicios.

2.4.3 LTV

[34]:
```
primeras_compras = pedidos.groupby('uid')['buy_ts'].min().reset_index()
primeras_compras.rename(columns={'buy_ts': 'first_purchase_date'}, inplace=True)

# columna de mes de primera compra
primeras_compras['first_purchase_month'] =␣
 ↪primeras_compras['first_purchase_date'].dt.to_period('M').dt.to_timestamp()

# unir tabla de pedidos
pedidos = pd.merge(pedidos, primeras_compras, on='uid', how='left')

# columna de mes de la orden
pedidos['order_month'] = pedidos['buy_ts'].dt.to_period('M').dt.to_timestamp()

# antigüedad de la cohorte
pedidos['cohort_age'] = (pedidos['order_month'] -␣
 ↪pedidos['first_purchase_month']) / np.timedelta64(1, 'M')
pedidos['cohort_age'] = pedidos['cohort_age'].round().astype('int')

# agrupar por first_purchase_month y contar usuarios diferentes
cohort_sizes = pedidos.groupby('first_purchase_month')['uid'].nunique().
 ↪reset_index()
cohort_sizes.rename(columns={'uid': 'usuarios_cohorte'}, inplace=True)

# tabla ltv
ltv = pedidos.groupby(['first_purchase_month', 'cohort_age'])['revenue'].sum().
 ↪reset_index()

ltv = pd.merge(ltv, cohort_sizes, on='first_purchase_month', how='left')

ltv['ltv'] = ltv['revenue'] / ltv['usuarios_cohorte']
```

```
ltv_table_pivot = ltv.pivot(index='first_purchase_month', columns='cohort_age',␣
 ↪values='ltv')
ltv_table_pivot = ltv_table_pivot.cumsum(axis=1)

ltv_table_pivot
```

[34]:

cohort_age	0	1	2	3	4
first_purchase_month					
2017-06-01	4.724414	5.209743	5.647380	6.602051	7.624582
2017-07-01	6.010218	6.345429	6.968960	7.327936	7.504727
2017-08-01	5.276518	5.748511	6.206993	6.598270	7.092321
2017-09-01	5.644529	6.762115	7.283045	11.258838	11.659396
2017-10-01	5.003733	5.539495	5.730889	5.888035	6.039594
2017-11-01	5.154683	5.553916	5.753472	6.078424	6.226437
2017-12-01	4.738191	4.998565	5.923662	6.988937	7.301866
2018-01-01	4.135636	4.430394	4.734675	4.877453	4.940151
2018-02-01	4.156987	4.435262	4.513777	4.587921	NaN
2018-03-01	4.838803	5.139694	5.455253	NaN	NaN
2018-04-01	4.657597	5.189196	NaN	NaN	NaN
2018-05-01	4.660562	NaN	NaN	NaN	NaN
2018-06-01	3.420000	NaN	NaN	NaN	NaN

cohort_age	5	6	7	8	9
first_purchase_month					
2017-06-01	8.360084	9.310524	9.892116	10.445329	11.051117
2017-07-01	7.660775	7.780983	7.922803	8.084035	8.231180
2017-08-01	7.375861	7.586526	7.991533	8.283745	8.471723
2017-09-01	12.306463	13.008071	13.251220	13.435227	NaN
2017-10-01	6.159956	6.244772	6.360242	NaN	NaN
2017-11-01	6.280316	6.395244	NaN	NaN	NaN
2017-12-01	7.639913	NaN	NaN	NaN	NaN
2018-01-01	NaN	NaN	NaN	NaN	NaN
2018-02-01	NaN	NaN	NaN	NaN	NaN
2018-03-01	NaN	NaN	NaN	NaN	NaN
2018-04-01	NaN	NaN	NaN	NaN	NaN
2018-05-01	NaN	NaN	NaN	NaN	NaN
2018-06-01	NaN	NaN	NaN	NaN	NaN

cohort_age	10	11
first_purchase_month		
2017-06-01	11.622378	11.879234
2017-07-01	8.386854	NaN
2017-08-01	NaN	NaN
2017-09-01	NaN	NaN
2017-10-01	NaN	NaN
2017-11-01	NaN	NaN

```
2017-12-01                  NaN         NaN
2018-01-01                  NaN         NaN
2018-02-01                  NaN         NaN
2018-03-01                  NaN         NaN
2018-04-01                  NaN         NaN
2018-05-01                  NaN         NaN
2018-06-01                  NaN         NaN
```

[35]:
```
ltv_201706 = ltv_table_pivot.loc['2017-06-01'].sum()
ltv_201706
```

[35]:
```
102.36895205140873
```

[36]:
```
promedio_columna_5 = ltv_table_pivot.iloc[:, 5].mean()

print(promedio_columna_5)
```

```
7.969052630207335
```

[37]:
```
visitas_junio_2017 = visitas[visitas['start_ts'].dt.to_period('M') == '2017-06']

visitas_pedidos_junio_2017 = pd.merge(visitas_junio_2017, pedidos, on='uid',␣
 ↪how='left')

clientes_adquiridos_junio_2017 = visitas_pedidos_junio_2017.
 ↪groupby('source_id')['uid'].nunique()

gastos_marketing['month'] = gastos_marketing['dt'].dt.to_period('M')
gastos_junio_2017 = gastos_marketing[gastos_marketing['month'] == '2017-06']
gastos_por_fuente_junio_2017 = gastos_junio_2017.groupby('source_id')['costs'].
 ↪sum()

cac_junio_2017 = gastos_por_fuente_junio_2017 / clientes_adquiridos_junio_2017
print(cac_junio_2017)
```

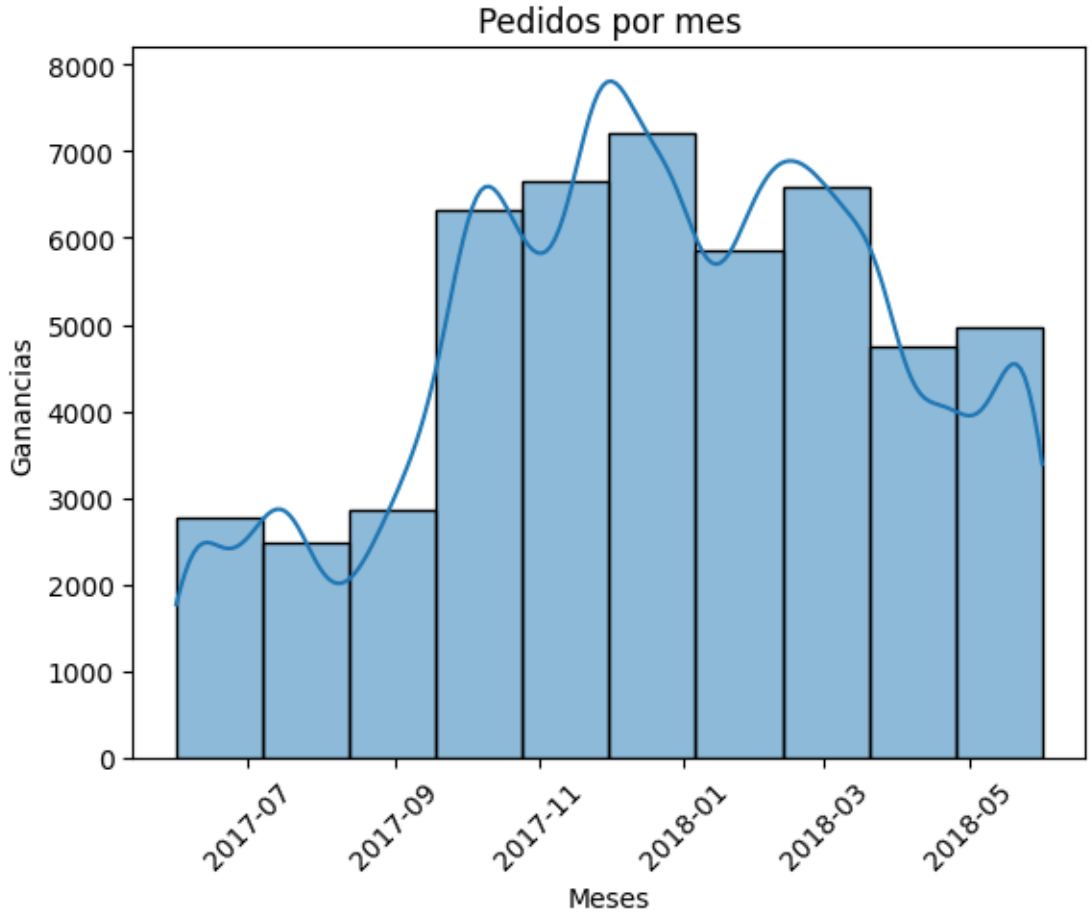
```
source_id
1     1.158035
2     1.584452
3     1.829543
4     0.966667
5     0.901178
7          NaN
9     0.378778
10    0.735878
dtype: float64
```

Las fuentes poseen un CAC, significativamente menor que el LTV, asi que se podria destinar un mayor presupuesto de inversion a los anuncios de marketing

Histograma de pedidos por mes

[38]:
```
plt.xticks(rotation=45)
plt.title('Pedidos por mes')
plt.xlabel('Meses')
plt.ylabel('Ganancias')
sns.histplot(pedidos['buy_ts'].astype("datetime64"), bins=10, kde=True)
```

[38]:
```
<AxesSubplot:title={'center':'Pedidos por mes'}, xlabel='Meses',
ylabel='Ganancias'>
```



Existe una tendencia general al alza en las ganancias a lo largo del período analizado, con un pico a mediados de diciembre. Esto sugiere un aumento en el número de pedidos o en el valor promedio de cada pedido.

La variación en las ganancias podría estar influenciada por factores estacionales, como festividades o eventos especiales que ocurren a finales de año.

La distribución de los pedidos no es uniforme a lo largo del tiempo, lo que indica que puede haber períodos de alta y baja demanda.

2.5 Marketing

2.5.1 Gastos

Total

[39]:
```
total_costos = gastos_marketing['costs'].sum()
print(f'Costo total: {total_costos}')
```

```
Costo total: 329131.62
```

El costo total de marketing para el periodo 2017-2018 asciende a 112,546.11 usd. Este valor representa el gasto total invertido en todas las fuentes de anuncios durante el periodo. Es un punto de referencia importante para evaluar la eficiencia de la inversión en marketing y comparar los gastos con otros periodos o presupuestos.

Por fuente de adquisición

[40]:
```
costos_por_fuente = gastos_marketing.groupby('source_id')['costs'].sum()
costos_por_fuente = costos_por_fuente.sort_values(ascending=False)
print(f'Costos por fuente: {costos_por_fuente}')
```

```
Costos por fuente: source_id
3     141321.63
4      61073.60
5      51757.10
2      42806.04
1      20833.27
10      5822.49
9       5517.49
Name: costs, dtype: float64
```

Al analizar los costos por fuente, podemos identificar que la fuente de anuncios “3” es la que ha recibido la mayor inversión, seguida de las fuentes “4” y “5”. Esto entonces coincide con lo analizado anteriormente, donde observamos que las fuentes 3 y 4 son las mas usadas o las que generan mayor numero de nuevos usuarios.

A lo largo del tiempo

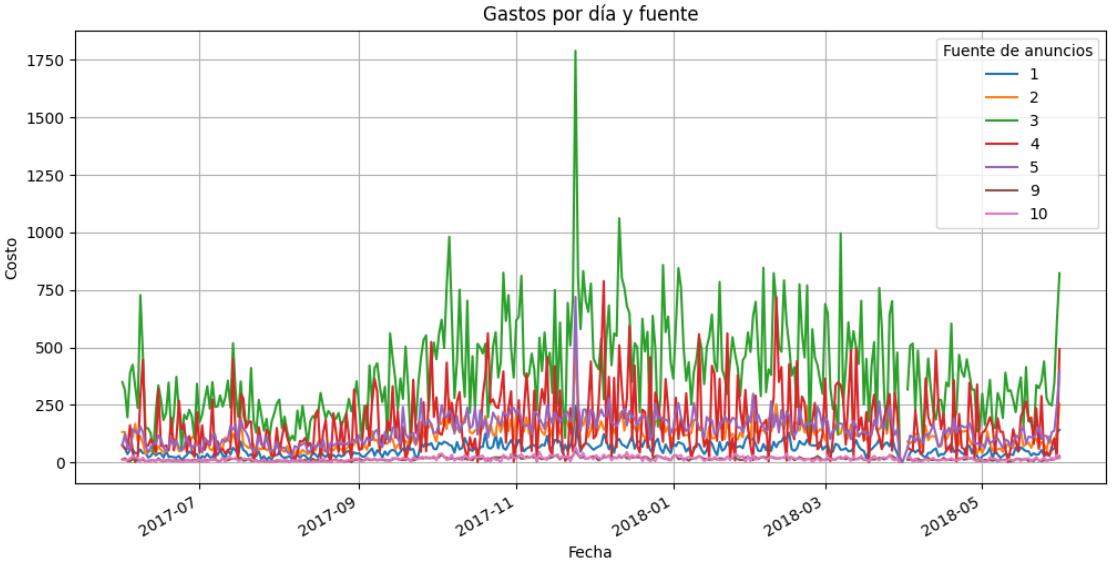
[41]:
```
costos_tiempo = gastos_marketing.groupby('dt')['costs'].sum()
costos_tiempo = costos_tiempo.sort_values(ascending=False)
print(f'Costos a lo largo del tiempo: {costos_tiempo}')
```

```
Costos a lo largo del tiempo: dt
2017-11-24    3458.86
2018-05-31    2153.70
2017-12-11    2121.13
2018-02-10    1981.39
2017-11-16    1811.02
               ...
2017-08-12     253.50
```

```
2017-08-07     252.77
2017-08-06     236.79
2018-03-30     106.35
2018-03-31       0.70
Name: costs, Length: 364, dtype: float64
```

La distribución de los gastos a lo largo del tiempo muestra una variabilidad considerable. Hay algunos días con gastos significativamente más altos que otros. Esto podría deberse a diferentes factores, como campañas de marketing específicas, promociones o eventos estacionales.

[42]:
```
gastos_por_dia_y_fuente = gastos_marketing.groupby(['dt',␣
↪'source_id'])['costs'].sum().unstack()

gastos_por_dia_y_fuente.plot(figsize=(12, 6))
plt.title('Gastos por día y fuente')
plt.xlabel('Fecha')
plt.ylabel('Costo')
plt.legend(title='Fuente de anuncios')
plt.grid(True)
plt.show()
```



Costo de adquisición de clientes de cada una de las fuentes

[43]:
```
primeras_compras = pedidos.groupby('uid')['buy_ts'].min().reset_index()
primeras_compras.rename(columns={'buy_ts': 'first_purchase_date'}, inplace=True)

primeras_visitas = visitas.sort_values(by='start_ts', ascending=True).
↪groupby('uid').first()
```

```
primeras_compras_visitas = pd.merge(primeras_visitas, primeras_compras,␣
 ↪on='uid', how='left')

primeras_compras_visitas.first_purchase_date = primeras_compras_visitas.
 ↪first_purchase_date.values.astype('datetime64[D]')

compradores_por_fuente_fecha = primeras_compras_visitas.groupby(['source_id',␣
 ↪'first_purchase_date'])['uid'].nunique().reset_index()
compradores_por_fuente_fecha.rename(columns={'uid': 'num_compradores',␣
 ↪'source_id': 'fuente'}, inplace=True)

compradores_por_fuente_fecha
```

[43]:

	fuente	first_purchase_date	num_compradores
0	1	2017-06-01	14
1	1	2017-06-02	7
2	1	2017-06-03	7
3	1	2017-06-04	3
4	1	2017-06-05	18
…	…	…	…
2430	10	2018-05-27	2
2431	10	2018-05-28	10
2432	10	2018-05-29	7
2433	10	2018-05-30	5
2434	10	2018-05-31	35

[2435 rows x 3 columns]

[44]:
```
# Costos por fecha y fuente
gastos_fuente_fecha = gastos_marketing.groupby(['source_id', 'dt'])['costs'].
 ↪sum().reset_index()
gastos_fuente_fecha.rename(columns={'costs': 'total_costs', 'source_id':
 ↪'fuente'}, inplace=True)

gastos_fuente_fecha
```

[44]:

	fuente	dt	total_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
…	…	…	…
2537	10	2018-05-27	9.92
2538	10	2018-05-28	21.26
2539	10	2018-05-29	11.32
2540	10	2018-05-30	33.15

```
2541      10 2018-05-31        17.60

[2542 rows x 3 columns]
```

[45]:
```
union = pd.merge(compradores_por_fuente_fecha, gastos_fuente_fecha, on =␣
 ↪'fuente', how= 'left')

# CAC diario
union['cac_diario'] = union['total_costs'] / union['num_compradores']

# Agrupar por fuente y calcular CAC promedio
cac_por_fuente = union.groupby('fuente')['cac_diario'].mean()

cac_por_fuente
```

[45]:
```
fuente
1     11.672781
2     20.020255
3     19.816979
4      9.253502
5     10.003057
7           NaN
9      7.230545
10     7.079685
Name: cac_diario, dtype: float64
```

El análisis del costo por adquisición (CAC) diario revela una variabilidad significativa entre las diferentes fuentes de adquisición. Las fuentes 3 y 5 presentan los CAC más elevados, lo que sugiere que estas estrategias de adquisición podrían ser menos rentables a largo plazo. Por el contrario, las fuentes 1 y 9 muestran un CAC más bajo, indicando una mayor eficiencia en la generación de nuevos clientes. Es importante destacar que la fuente 7 no presenta datos de CAC, lo que podría indicar que no hubo adquisiciones o gastos de marketing asociados a esta fuente durante el período analizado.

2.5.2 Rentabilidad de la inversión (ROMI)

[46]:
```
# Primera visita usuarios
primera_visita = (
    visitas[['uid','start_ts','source_id']]
    .sort_values(by='start_ts',ascending=True)
    .groupby('uid',as_index=False)
    .first())

# Mes de primera visita usuarios
primera_visita['first_visit_month'] = primera_visita['start_ts'].
 ↪astype('datetime64[M]')
```

```
# Primera orden usuarios
primer_pedido = (
    pedidos[['uid','buy_ts']]
    .sort_values(by='buy_ts',ascending=True)
    .groupby('uid',as_index=False)
    .first())

# Mes de primera orden usuarios
primer_pedido['first_order_month'] = primer_pedido['buy_ts'].
 ↪astype('datetime64[M]')

# Seleccionar columnas a usar de visitas
primera_visita = primera_visita[['uid','source_id','first_visit_month']]

# Unir data de primera orden a dataframe orders
orders = pd.merge(pedidos,primer_pedido[['first_order_month','uid']],on='uid')

# Obtener mes de compra
orders['order_month'] = orders['buy_ts'].astype('datetime64[M]')

# Seleccionar columnas a usar de ordenes
compradores = orders[['uid','first_order_month','revenue','order_month']]

# Unir data de ordenes con data de visitas
compradores = pd.merge(compradores,primera_visita, on='uid')
compradores
```

[46]:

	uid	first_order_month	revenue	order_month	source_id	first_visit_month
0	10329302124590727494	2017-06-01	17.00	2017-06-01	1	2017-06-01
1	11627257723692907447	2017-06-01	0.55	2017-06-01	2	2017-06-01
2	17903680561304213844	2017-06-01	0.37	2017-06-01	2	2017-06-01
3	16109239769442553005	2017-06-01	0.55	2017-06-01	2	2017-06-01
4	14200605875248379450	2017-06-01	0.37	2017-06-01	3	2017-06-01
…	…	…	…	…	…	…
50410	12296626599487328624	2018-05-01	4.64	2018-05-01	4	2018-05-01
50411	11369640365507475976	2018-05-01	5.80	2018-05-01	10	2018-05-01
50412	1786462140797698849	2018-05-01	0.30	2018-05-01	3	2018-05-01
50413	3993697860786194247	2018-05-01	3.67	2018-05-01	3	2018-05-01
50414	83872787173869366	2018-06-01	3.42	2018-06-01	4	2018-06-01

```
50410        2018-05-01
50411        2018-05-01
50412        2018-05-01
50413        2017-10-01
50414        2018-05-01

[50415 rows x 6 columns]
```

[47]:
```
# Generar columna de mes en los costos
gastos_marketing['month'] = gastos_marketing['dt'].astype('datetime64[M]')

# Agrupar gastos por fuente y mes
gastos_mensuales = gastos_marketing.groupby(['source_id',␣
 ↪gastos_marketing['dt'].dt.to_period('M')])['costs'].sum().reset_index()
gastos_mensuales
```

[47]:

	source_id	dt	costs
0	1	2017-06	1125.61
1	1	2017-07	1072.88
2	1	2017-08	951.81
3	1	2017-09	1502.01
4	1	2017-10	2315.75
..	…	…	…
79	10	2018-01	614.35
80	10	2018-02	480.88
81	10	2018-03	526.41
82	10	2018-04	388.25
83	10	2018-05	409.86

[84 rows x 3 columns]

[48]:
```
metricas = compradores.groupby(['source_id', compradores['first_order_month'].
 ↪dt.to_period('M')]).agg(
      tamaño_cohorte=('uid', 'nunique'),
      revenue_por_mes=('revenue', 'sum')).reset_index()
metricas
```

[48]:

	source_id	first_order_month	tamaño_cohorte	revenue_por_mes
0	1	2017-06	190	6392.39
1	1	2017-07	160	3342.52
2	1	2017-08	113	2110.91
3	1	2017-09	227	2364.55
4	1	2017-10	340	2850.08
..	…	…	…	…
81	10	2018-01	92	292.93
82	10	2018-02	123	319.02
83	10	2018-03	186	657.44

```
84          10           2018-04              107           261.93
85          10           2018-05              130           470.89

[86 rows x 4 columns]
```

[49]:
```
# Unir por source_id y mes
metricas_completas = metricas.merge(gastos_mensuales,
                                                    how='left',
                                                    left_on=['source_id',␣
 ↪'first_order_month'],
                                                    right_on=['source_id', 'dt'],
                                                    suffixes=('_pedidos',␣
 ↪'_marketing'))

  # Calcular LTV, CAC y ROMI
metricas_completas['ltv'] = metricas_completas['revenue_por_mes'] /␣
 ↪metricas_completas['tamaño_cohorte']
metricas_completas['cac'] = metricas_completas['costs'] /␣
 ↪metricas_completas['tamaño_cohorte']
metricas_completas['romi'] = metricas_completas['ltv'] /␣
 ↪metricas_completas['cac']
metricas_completas
```

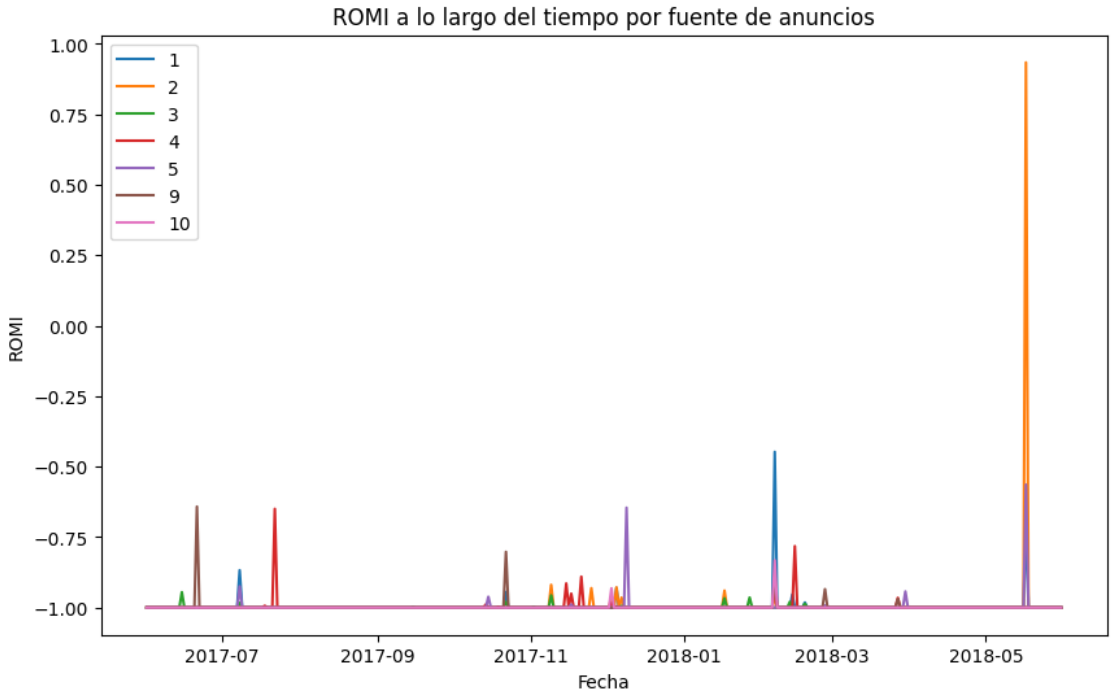
[49]:
```
    source_id first_order_month  tamaño_cohorte  revenue_por_mes       dt  \
0           1           2017-06             190          6392.39  2017-06
1           1           2017-07             160          3342.52  2017-07
2           1           2017-08             113          2110.91  2017-08
3           1           2017-09             227          2364.55  2017-09
4           1           2017-10             340          2850.08  2017-10
..        …                 …               …                …       …
81         10           2018-01              92           292.93  2018-01
82         10           2018-02             123           319.02  2018-02
83         10           2018-03             186           657.44  2018-03
84         10           2018-04             107           261.93  2018-04
85         10           2018-05             130           470.89  2018-05

      costs        ltv       cac      romi
0   1125.61  33.644158  5.924263  5.679045
1   1072.88  20.890750  6.705500  3.115465
2    951.81  18.680619  8.423097  2.217785
3   1502.01  10.416520  6.616784  1.574257
4   2315.75   8.382588  6.811029  1.230737
..      …         …         …         …
81   614.35   3.184022  6.677717  0.476813
82   480.88   2.593659  3.909593  0.663409
83   526.41   3.534624  2.830161  1.248912
84   388.25   2.447944  3.628505  0.674643
```

```
85    409.86    3.622231  3.152769  1.148905

[86 rows x 9 columns]
```

El análisis del ROMI por fuente y año revela una variabilidad significativa en la efectividad de las diferentes estrategias de adquisición. Si bien algunas fuentes mostraron un alto retorno de la inversión en 2017, se observó una tendencia general a la baja en el ROMI en 2018. Esto sugiere que podría ser necesario ajustar las estrategias de marketing para las fuentes con menor rendimiento.

2.5.3 ROMI a lo largo del tiempo por fuente de anuncios

[128]:
```
fecha_costos_por_fuente = gastos_marketing.groupby(['source_id',␣
 ↪'dt'])['costs'].sum().reset_index()

# ingresos por fuente y fecha
pedidos_visitas = pd.merge(pedidos,visitas, on="uid", how= "left")
fecha_ingresos_por_fuente = pedidos_visitas.groupby(['source_id',␣
 ↪'buy_ts'])['revenue'].sum().reset_index()
fecha_ingresos_por_fuente = fecha_ingresos_por_fuente.
 ↪rename(columns={'source_id': 'source_id', 'buy_ts': 'dt'})

# tablas de costos e ingresos
romi_data = pd.merge(fecha_costos_por_fuente, fecha_ingresos_por_fuente,␣
 ↪on=['source_id', 'dt'], how='left')
romi_data['revenue'] = romi_data['revenue'].fillna(0)

# ROMI
romi_data['romi'] = (romi_data['revenue'] - romi_data['costs']) /␣
 ↪romi_data['costs']

# ROMI por fuente y fecha
romi_over_time = romi_data.groupby(['source_id', 'dt'])['romi'].mean().
 ↪reset_index()

fig, ax = plt.subplots(figsize=(10, 6))

for source in romi_over_time['source_id'].unique():
  data = romi_over_time[romi_over_time['source_id'] == source]
  ax.plot(data['dt'], data['romi'], label=source)

ax.set_xlabel('Fecha')
ax.set_ylabel('ROMI')
ax.set_title('ROMI a lo largo del tiempo por fuente de anuncios')
ax.legend()
plt.show()
```



Las fuentes de anuncios no muestran un rendimiento estable a lo largo del tiempo. Esto podría deberse a diversos factores como cambios en las estrategias de marketing, fluctuaciones en la demanda o factores externos.

La mayoría de las fuentes presentan un ROMI negativo, lo que indica que, en general, las campañas de marketing no están siendo rentables.

Los resultados sugieren que es necesario realizar ajustes en las estrategias de marketing para mejorar el rendimiento de las campañas.

2.6 Conclusión general

El análisis exhaustivo de los datos de visitas, pedidos y gastos de marketing de Showz ha revelado patrones interesantes y oportunidades de mejora en las estrategias de adquisición de clientes.

Las diferentes fuentes de adquisición presentan un rendimiento variable en términos de adquisición de nuevos usuarios y costo por adquisición. Algunas fuentes, como las 3 y 4, han demostrado ser altamente efectivas en términos de volumen de usuarios, pero también presentan un costo por adquisición más elevado.

Se ha observado una tendencia creciente en el número de pedidos y un aumento en el tamaño promedio de compra, lo que indica un crecimiento general del negocio. Sin embargo, el retorno sobre la inversión (ROMI) ha mostrado una tendencia a la baja en algunos casos, lo que sugiere la necesidad de ajustar las estrategias de marketing.

Existen oportunidades significativas para optimizar los gastos de marketing al concentrar los esfuerzos en las fuentes más rentables, mejorar la segmentación de la audiencia y personalizar las campañas de marketing.

Recomendaciones:

- Aumentar la inversión en las fuentes que han demostrado un mayor retorno de la inversión (ROMI) y un costo por adquisición más bajo.
- Realizar pruebas A/B para identificar las creatividades, mensajes y canales más efectivos para cada fuente.
- Segmentar la audiencia en grupos más pequeños y personalizados para ofrecer mensajes más relevantes y aumentar las tasas de conversión.
- Analizar el valor de vida del cliente (LTV) para evaluar la rentabilidad a largo plazo de cada adquisición.
- Desarrollar programas de fidelización para aumentar la retención de clientes y fomentar las compras repetidas.
- Realizar un seguimiento continuo: Monitorear regularmente las métricas clave para evaluar el impacto de las estrategias de marketing y realizar ajustes según sea necesario.

En conclusión, al optimizar la asignación del presupuesto de marketing y enfocarse en las fuentes más rentables, Showz puede mejorar significativamente su retorno sobre la inversión y acelerar el crecimiento del negocio. Es fundamental adoptar un enfoque basado en datos para tomar decisiones informadas y adaptar las estrategias de marketing a las necesidades cambiantes del mercado.