4d62a4dc-371e-491d-a77a-c2ab228aa8a0

July 23, 2024

¡Hola!

Mi nombre es Tonatiuh Cruz. Me complace revisar tu proyecto hoy.

Al identificar cualquier error inicialmente, simplemente los destacaré. Te animo a localizar y abordar los problemas de forma independiente como parte de tu preparación para un rol como data-scientist. En un entorno profesional, tu líder de equipo seguiría un enfoque similar. Si encuentras la tarea desafiante, proporcionaré una pista más específica en la próxima iteración.

Encontrarás mis comentarios a continuación - **por favor no los muevas, modifiques o elimines**.

Puedes encontrar mis comentarios en cajas verdes, amarillas o rojas como esta:

Comentario del revisor

Éxito. Todo está hecho correctamente.

Comentario del revisor

Observaciones. Algunas recomendaciones.

Comentario del revisor

Necesita corrección. El bloque requiere algunas correcciones. El trabajo no puede ser aceptado con comentarios en rojo.

Puedes responderme utilizando esto:

Respuesta del estudiante.

1 ¿Cuál es la mejor tarifa?

Trabajas como analista para el operador de telecomunicaciones Megaline. La empresa ofrece a sus clientes dos tarifas de prepago, Surf y Ultimate. El departamento comercial quiere saber cuál de las tarifas genera más ingresos para poder ajustar el presupuesto de publicidad.

Vas a realizar un análisis preliminar de las tarifas basado en una selección de clientes relativamente pequeña. Tendrás los datos de 500 clientes de Megaline: quiénes son los clientes, de dónde son, qué tarifa usan, así como la cantidad de llamadas que hicieron y los mensajes de texto que enviaron en 2018. Tu trabajo es analizar el comportamiento de los clientes y determinar qué tarifa de prepago genera más ingresos.

El objetivo del proyecto es el análisis y presentación de datos de una muestra de clientes de dos planes de telefonía de la operadora Megaline para determinar el comportamiento de los clientes

y así determinar qué paquete aporta más ingresos a la empresa y en desde qué punto de vista se debería fortalecer su estrategia de publicidad.

Primero se deben cargar los datos y prepararlos: investigar valores ausentes y filas duplicadas, y manejarlas según amerite. Asimismo se van a enriquecer los datos de acuerdo a las instrucciones del proyecto, necesidades del análisis y requisitos tarifarios.

1.1 Inicialización

[1]:
```
# Cargar todas las librerías
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
from scipy import stats as st
import math as mt
import seaborn as sns
```

1.2 Cargar datos

[2]:
```
# Carga los archivos de datos en diferentes DataFrames
df_users=pd.read_csv('/datasets/megaline_users.csv')
df_calls=pd.read_csv('/datasets/megaline_calls.csv')
df_messages=pd.read_csv('/datasets/megaline_messages.csv')
df_internet=pd.read_csv('/datasets/megaline_internet.csv')
df_plans=pd.read_csv('/datasets/megaline_plans.csv')
```

1.3 Preparar los datos

Las tablas se examiman por separado y se preparan en función de los requerimientos del análisis.

1.4 Tarifas

[3]:
```
df_plans.info()# Imprime la información general/resumida sobre el DataFrame de␣
↪las tarifas
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 8 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   messages_included     2 non-null      int64
 1   mb_per_month_included  2 non-null      int64
 2   minutes_included      2 non-null      int64
 3   usd_monthly_pay       2 non-null      int64
 4   usd_per_gb            2 non-null      int64
 5   usd_per_message       2 non-null      float64
 6   usd_per_minute        2 non-null      float64
 7   plan_name             2 non-null      object
```

```
dtypes: float64(2), int64(5), object(1)
memory usage: 256.0+ bytes
```

[4]:
```
# Imprime una muestra de los datos para las tarifas
df_plans.head()
```

[4]:

	messages_included	mb_per_month_included	minutes_included	usd_monthly_pay	usd_per_gb	usd_per_message	usd_per_minute	plan_name
0	50	15360	500	20	10	0.03	0.03	surf
1	1000	30720	3000	70	7	0.01	0.01	ultimate

En esta tabla hay dos filas correspondientes a los dos planes prepago de Megaline: Surf y Ultimate. Las características y descripción de las tarifas concuerdan con lo esperado. También hay coherencia entre los valores mostrados en cada columna y el tipo de datos que registran. Para efectos del análisis de datos sería deseable que los valores de la columna mb_per_month se convirtieran a gigabytes, pues es con esa magnitud es con la que se va a trabajar.

1.5 Corregir datos

No se encontraron problemas en esta tabla.

[ ]:
```
```

1.6 Enriquecer los datos

Convertir los valores en megabytes a gigabytes y actualizar el nombre de la columna.

[5]:
```
df_plans['mb_per_month_included']=df_plans['mb_per_month_included']/1024
df_plans.rename(columns={'mb_per_month_included':'gb_per_month_included'},␣
 ↪inplace=True)
df_plans.head()
```

[5]:

	messages_included	gb_per_month_included	minutes_included	usd_monthly_pay	usd_per_gb	usd_per_message	usd_per_minute	plan_name
0	50	15.0	500	20	10	0.03	0.03	surf
1	1000	30.0	3000	70	7	0.01	0.01	ultimate

Comentario del revisor

Muy buen trabajo!! Es correcto considerar que 1024 megabytes son 1 gigabytes.

1.7 Usuarios/as

[6]:
```
df_users.info()# Imprime la información general/resumida sobre el DataFrame de␣
↪usuarios
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   user_id     500 non-null    int64
 1   first_name  500 non-null    object
 2   last_name   500 non-null    object
 3   age         500 non-null    int64
 4   city        500 non-null    object
 5   reg_date    500 non-null    object
 6   plan        500 non-null    object
 7   churn_date  34 non-null     object
dtypes: int64(2), object(6)
memory usage: 31.4+ KB
```

[7]:
```
df_users.sample(20)# Imprime una muestra de datos para usuarios
```

[7]:

	user_id	first_name	last_name	age	\
44	1044	Devora	Galloway	74	
345	1345	Pasquale	Caldwell	26	
372	1372	Patria	Kim	26	
459	1459	Santos	Head	40	
188	1188	Ethelene	Brock	31	
414	1414	Georgianne	Herring	30	
447	1447	Ramon	Hester	62	
19	1019	Shizue	Landry	34	
198	1198	Russ	Horne	69	
255	1255	Kennith	Rowland	22	
302	1302	Leonila	Morris	62	
105	1105	Micheal	Poole	57	
79	1079	Brian	Mccall	48	
493	1493	Cicely	Wynn	18	
338	1338	Janise	Bowman	21	
307	1307	Kristopher	Lang	28	
362	1362	Kenyetta	Mcknight	65	
101	1101	Sage	Conley	27	
240	1240	Drema	Lopez	61	
427	1427	Zofia	Brock	64	

	city	reg_date	plan	\
44	Albuquerque, NM MSA	2018-08-30	surf	

```
345            Los Angeles-Long Beach-Anaheim, CA MSA  2018-07-12      surf
372         New York-Newark-Jersey City, NY-NJ-PA MSA  2018-10-08      surf
459            San Francisco-Oakland-Berkeley, CA MSA  2018-04-27  ultimate
188                                  Richmond, VA MSA  2018-01-10  ultimate
414                            Urban Honolulu, HI MSA  2018-03-03      surf
447                 Orlando-Kissimmee-Sanford, FL MSA  2018-05-01      surf
19                                Jacksonville, FL MSA  2018-01-16      surf
198         New York-Newark-Jersey City, NY-NJ-PA MSA  2018-05-01      surf
255                             Oklahoma City, OK MSA  2018-08-01  ultimate
302                                Rochester, NY MSA  2018-01-21      surf
105                     Providence-Warwick, RI-MA MSA  2018-01-08      surf
79          New York-Newark-Jersey City, NY-NJ-PA MSA  2018-01-26      surf
493               Boston-Cambridge-Newton, MA-NH MSA  2018-03-06  ultimate
338       Minneapolis-St. Paul-Bloomington, MN-WI MSA  2018-08-09      surf
307               Boston-Cambridge-Newton, MA-NH MSA  2018-12-31      surf
362                   Denver-Aurora-Lakewood, CO MSA  2018-01-18      surf
101  Washington-Arlington-Alexandria, DC-VA-MD-WV MSA  2018-02-08      surf
240                               Baton Rouge, LA MSA  2018-03-18      surf
427  Washington-Arlington-Alexandria, DC-VA-MD-WV MSA  2018-01-26  ultimate

     churn_date
44          NaN
345         NaN
372         NaN
459         NaN
188         NaN
414  2018-09-01
447         NaN
19          NaN
198         NaN
255         NaN
302         NaN
105         NaN
79          NaN
493         NaN
338         NaN
307         NaN
362         NaN
101         NaN
240         NaN
427         NaN
```

En la tabla hay datos ausentes en la columna churn_date, que reflejan los usuarios que a la sazón del año 2018 todavía seguían siendo usuarios de Megaline. Por supuesto como la mayoría se usuarios continúan siendo clientes el número de valores ausentes es mayor a los no ausentes. Los valores de fechas de la columna reg_date podrían convertirse a objetos tipo datetime.

1.7.1 Corregir los datos

Realizar búsqueda y gestión de duplicados y asimismo corrección de valores ausentes.

[8]:
```
print(df_users.duplicated().sum(),'\n')
print(df_users['user_id'].duplicated().sum())
```

```
0

0
```

[9]:
```
df_users['churn_date'].fillna('non-churned', inplace=True)
```

1.7.2 Enriquecer los datos

Cambiar los registros de fechas a datos tipo datetime.

[10]:
```
df_users['reg_date'] = pd.to_datetime(df_users['reg_date'], format='%Y-%m-%d')
df_users.info()
df_users.sample(20)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
 #   Column      Non-Null Count  Dtype         
---  ------      --------------  -----         
 0   user_id     500 non-null    int64         
 1   first_name  500 non-null    object        
 2   last_name   500 non-null    object        
 3   age         500 non-null    int64         
 4   city        500 non-null    object        
 5   reg_date    500 non-null    datetime64[ns]
 6   plan        500 non-null    object        
 7   churn_date  500 non-null    object        
dtypes: datetime64[ns](1), int64(2), object(5)
memory usage: 31.4+ KB
```

[10]:

	user_id	first_name	last_name	age	\
66	1066	Ariel	Woods	61	
169	1169	Hai	Bean	67	
196	1196	Noel	Dawson	46	
337	1337	Lucius	Arnold	31	
472	1472	Maximo	Mendoza	51	
485	1485	Damion	Woodard	67	
426	1426	Lamont	Conner	44	
104	1104	Thurman	Stephens	20	
303	1303	Rosamaria	Reeves	67	
396	1396	Ardelia	Benton	65	
365	1365	Milford	Rush	19	

```
69      1069       Dino        Fry   31
424     1424   Kasandra      Keith   51
295     1295       Hung    Flowers   68
247     1247     Marion  Singleton   75
122     1122     Lashay      Reese   57
135     1135     Scotty      White   51
198     1198       Russ      Horne   69
228     1228       Jude       Hale   26
293     1293      Lanny      Nolan   23

                                                 city   reg_date      plan  \
66                Boston-Cambridge-Newton, MA-NH MSA 2018-03-08      surf
169         New York-Newark-Jersey City, NY-NJ-PA MSA 2018-12-06      surf
196            Los Angeles-Long Beach-Anaheim, CA MSA 2018-01-14  ultimate
337          Riverside-San Bernardino-Ontario, CA MSA 2018-03-11      surf
472            San Francisco-Oakland-Berkeley, CA MSA 2018-04-10      surf
485  Nashville-Davidson-Murfreesboro-Franklin, TN MSA 2018-08-21      surf
426            San Francisco-Oakland-Berkeley, CA MSA 2018-07-09  ultimate
104            Chicago-Naperville-Elgin, IL-IN-WI MSA 2018-12-23  ultimate
303                  Detroit-Warren-Dearborn, MI MSA 2018-10-25  ultimate
396                           Salt Lake City, UT MSA 2018-06-01      surf
365                    Grand Rapids-Kentwood, MI MSA 2018-02-09      surf
69           Houston-The Woodlands-Sugar Land, TX MSA 2018-09-17  ultimate
424    Virginia Beach-Norfolk-Newport News, VA-NC MSA 2018-07-24  ultimate
295                           Kansas City, MO-KS MSA 2018-08-11      surf
247            Los Angeles-Long Beach-Anaheim, CA MSA 2018-04-12  ultimate
122     Miami-Fort Lauderdale-West Palm Beach, FL MSA 2018-03-04      surf
135     Miami-Fort Lauderdale-West Palm Beach, FL MSA 2018-12-23  ultimate
198         New York-Newark-Jersey City, NY-NJ-PA MSA 2018-05-01      surf
228                  Detroit-Warren-Dearborn, MI MSA 2018-04-15      surf
293   Philadelphia-Camden-Wilmington, PA-NJ-DE-MD MSA 2018-08-13      surf

     churn_date
66   non-churned
169  non-churned
196  non-churned
337  non-churned
472  non-churned
485  non-churned
426  non-churned
104  non-churned
303  non-churned
396  non-churned
365  non-churned
69   non-churned
424  non-churned
295  non-churned
```

```
247  non-churned
122  non-churned
135  non-churned
198  non-churned
228  non-churned
293  non-churned
```

Comentario Revisor

Para hacer la transformación de reg_date a un datatime puedes hacer uso de la siguiente función:

`pd.to_datetime(user['reg_date'], format='%Y-%m-%d')`

te recomiendo agregar el argumento "format='%Y-%m-%d'" dentro de la función to_datetime(). De esta manera, puedes asegurarte siempre de que el formato de la fecha que deseas cambiar sea el que necesitas. Los mismo para las otras variables que son fechas de otras bases de datos.

Originalmente pensé en agregarlo así, pero como que noté que era el formato por defecto lo obvié. Corregido.

1.8 Llamadas

[11]:
```
df_calls.info()# Imprime la información general/resumida sobre el DataFrame de␣
↪las llamadas
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137735 entries, 0 to 137734
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   id         137735 non-null  object
 1   user_id    137735 non-null  int64
 2   call_date  137735 non-null  object
 3   duration   137735 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 4.2+ MB
```

[12]:
```
# Imprime una muestra de datos para las llamadas
df_calls.sample(20)
```

[12]:

	id	user_id	call_date	duration
122844	1434_141	1434	2018-12-06	16.63
102067	1362_831	1362	2018-12-13	12.85
117218	1410_50	1410	2018-12-07	0.54
44988	1163_47	1163	2018-11-01	0.67
75573	1268_77	1268	2018-12-17	14.89
130382	1468_165	1468	2018-09-30	23.36
114198	1400_548	1400	2018-11-18	0.00
59783	1215_168	1215	2018-10-07	10.98

```
119850  1417_550     1417  2018-10-21      3.70
17250   1066_358     1066  2018-05-01      9.97
30146   1113_449     1113  2018-11-21     29.50
120187   1418_71     1418  2018-12-08      7.70
107832  1382_699     1382  2018-09-03      0.00
54050   1195_220     1195  2018-11-30      0.00
81215   1292_167     1292  2018-08-29     11.96
44958    1163_17     1163  2018-12-05      9.55
69696   1249_667     1249  2018-07-31      5.38
28477   1109_104     1109  2018-07-28     14.90
137235  1498_163     1498  2018-11-13      0.00
71656   1255_266     1255  2018-11-05     10.19
```

[13]:
```
print(df_calls.duplicated().sum(),'\n')
```

```
0
```

En la tabla no hay datos ausentes ni duplicados. Sin embargo, los valores de fechas de la columna call_date podrían convertirse a objetos tipo datetime para cálculos ulteriores.

1.8.1 Corregir los datos

No se encontraron problemas de datos en esta tabla.

[ ]:
```

```

1.8.2 Enriquecer los datos

Cambiar los registros de llamadas a datos tipo datetime y crear una nueva columna donde conste el mes de la llamada. Adicionalmente se redondea la duración de llamadas individuales a valores discretos, pues Megaline redondea los segundos a minutos.

[14]:
```
df_calls['call_date'] = pd.to_datetime(df_calls['call_date'], format='%Y-%m-%d')
df_calls['call_month']=df_calls['call_date'].dt.month

df_calls.head()
```

[14]:
```
         id  user_id  call_date  duration  call_month
0   1000_93     1000 2018-12-27      8.52          12
1  1000_145     1000 2018-12-27     13.66          12
2  1000_247     1000 2018-12-27     14.48          12
3  1000_309     1000 2018-12-28      5.76          12
4  1000_380     1000 2018-12-30      4.22          12
```

Comentario Revisor

En todos los casos en los que cambiamos el tipo de variable a datetime, te recomiendo agregar el argumento "format='%Y-%m-%d' " dentro de la función to_datetime(). De esta manera, puedes

asegurarte siempre de que el formato de la fecha que deseas cambiar sea el que necesitas.

[15]:
```
df_calls['duration']= np.ceil(df_calls['duration'])
df_calls.head()
```

[15]:

	id	user_id	call_date	duration	call_month
0	1000_93	1000	2018-12-27	9.0	12
1	1000_145	1000	2018-12-27	14.0	12
2	1000_247	1000	2018-12-27	15.0	12
3	1000_309	1000	2018-12-28	6.0	12
4	1000_380	1000	2018-12-30	5.0	12

1.9 Mensajes

[16]:
```
# Imprime la información general/resumida sobre el DataFrame de los mensajes
df_messages.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76051 entries, 0 to 76050
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   id            76051 non-null  object
 1   user_id       76051 non-null  int64 
 2   message_date  76051 non-null  object
dtypes: int64(1), object(2)
memory usage: 1.7+ MB
```

[17]:
```
# Imprime una muestra de datos para los mensajes
df_messages.sample(20)
```

[17]:

	id	user_id	message_date
64458	1412_40	1412	2018-06-25
73141	1470_769	1470	2018-08-27
50985	1331_160	1331	2018-10-13
9642	1070_8	1070	2018-12-30
69352	1451_49	1451	2018-12-09
13797	1093_232	1093	2018-12-23
42100	1271_78	1271	2018-12-06
49537	1328_495	1328	2018-09-06
21775	1132_1257	1132	2018-10-02
10159	1076_364	1076	2018-10-10
72532	1470_160	1470	2018-10-12
53904	1341_180	1341	2018-12-26
29145	1178_215	1178	2018-12-10
12644	1082_208	1082	2018-07-16
5615	1053_113	1053	2018-08-12

```
68643   1444_126     1444   2018-12-23
9142    1066_317     1066   2018-08-31
4004    1039_102     1039   2018-08-22
815     1007_117     1007   2018-08-24
43082    1283_21     1283   2018-12-13
```

[18]:
```
df_messages.duplicated().sum()
```

[18]:
```
0
```

En la tabla no hay datos ausentes ni duplicados.

1.9.1 Corregir los datos

En la tabla no se ven problemas a corregir.

[ ]:
```

```

1.9.2 Enriquecer los datos

Cambiar los registros de mensajes a datos tipo datetime y crear una nueva columna donde se extraiga el mes de las fechas de los mensajes.

[19]:
```
df_messages['message_date'] = pd.to_datetime(df_messages['message_date'],␣
↪format='%Y-%m-%d')
df_messages['message_month']=df_messages['message_date'].dt.month
df_messages.head()
```

[19]:
```
          id  user_id message_date  message_month
0  1000_125     1000   2018-12-27             12
1  1000_160     1000   2018-12-31             12
2  1000_223     1000   2018-12-31             12
3  1000_251     1000   2018-12-27             12
4  1000_255     1000   2018-12-26             12
```

1.10 Internet

[20]:
```
# Imprime la información general/resumida sobre el DataFrame de internet
df_internet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104825 entries, 0 to 104824
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            104825 non-null  object
 1   user_id       104825 non-null  int64
 2   session_date  104825 non-null  object
```

```
 3   mb_used       104825 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 3.2+ MB
```

[21]:
```
# Imprime una muestra de datos para el tráfico de internet
df_internet.sample(20)
```

[21]:

	id	user_id	session_date	mb_used
25423	1117_356	1117	2018-05-31	543.37
96640	1454_96	1454	2018-10-05	512.19
35125	1158_170	1158	2018-12-28	514.75
79728	1373_7	1373	2018-11-18	1130.83
21541	1099_294	1099	2018-03-07	336.01
66124	1305_347	1305	2018-07-12	265.96
61460	1279_74	1279	2018-12-07	684.02
28212	1128_366	1128	2018-07-18	1084.89
68115	1317_69	1317	2018-11-21	542.55
8317	1043_584	1043	2018-10-10	921.05
3903	1022_314	1022	2018-11-16	39.25
27525	1127_221	1127	2018-10-25	363.94
87115	1403_463	1403	2018-09-16	571.26
79445	1370_169	1370	2018-12-04	452.28
53840	1243_142	1243	2018-11-13	331.11
63411	1292_115	1292	2018-08-03	511.61
36420	1164_103	1164	2018-12-18	779.63
60463	1274_1	1274	2018-12-19	602.70
31680	1144_416	1144	2018-11-11	756.94
86517	1402_46	1402	2018-09-30	438.51

[22]:
```
df_internet.duplicated().sum()
```

[22]:
```
0
```

En la tabla no hay datos ausentes ni duplicados.

1.10.1 Corregir los datos

En la tabla no se ven problemas obvios a corregir.

[ ]:
```

```

1.10.2 Enriquecer los datos

Convertir los valores que constan en megabytes a gigabytes y actualizar el nombre de la columna a gb_used. Los valores de consumo de datos se redondean hacia arriba porque sí se maneja Megaline.

[23]:
```
df_internet['mb_used']=df_internet['mb_used']/1024
df_internet.rename(columns={'mb_used':'gb_used'}, inplace=True)
```

```
df_internet['gb_used']=np.ceil(df_internet['gb_used'])
df_internet.head(10)
```

[23]:

	id	user_id	session_date	gb_used
0	1000_13	1000	2018-12-29	1.0
1	1000_204	1000	2018-12-31	0.0
2	1000_379	1000	2018-12-28	1.0
3	1000_413	1000	2018-12-26	1.0
4	1000_442	1000	2018-12-27	1.0
5	1001_0	1001	2018-08-24	1.0
6	1001_3	1001	2018-12-09	1.0
7	1001_4	1001	2018-11-04	1.0
8	1001_10	1001	2018-11-27	1.0
9	1001_15	1001	2018-12-13	1.0

[24]:
```
df_internet['session_date'] = pd.to_datetime(df_internet['session_date'],␣
↪format='%Y-%m-%d')
df_internet['session_month']=df_internet['session_date'].dt.month

df_internet.head(15)
```

[24]:

	id	user_id	session_date	gb_used	session_month
0	1000_13	1000	2018-12-29	1.0	12
1	1000_204	1000	2018-12-31	0.0	12
2	1000_379	1000	2018-12-28	1.0	12
3	1000_413	1000	2018-12-26	1.0	12
4	1000_442	1000	2018-12-27	1.0	12
5	1001_0	1001	2018-08-24	1.0	8
6	1001_3	1001	2018-12-09	1.0	12
7	1001_4	1001	2018-11-04	1.0	11
8	1001_10	1001	2018-11-27	1.0	11
9	1001_15	1001	2018-12-13	1.0	12
10	1001_16	1001	2018-10-28	1.0	10
11	1001_17	1001	2018-09-05	1.0	9
12	1001_24	1001	2018-09-05	1.0	9
13	1001_25	1001	2018-10-14	1.0	10
14	1001_26	1001	2018-09-17	0.0	9

1.11 Estudiar las condiciones de las tarifas

Surf 1. Pago mensual: 20 USD. 2. Prestaciones al mes: 500 minutos, 50 SMS y 15 GB de datos. 3. Si se exceden los límites del paquete:

1 minuto: 0.03 USD.

1 SMS: 0.03 USD.

1 GB de datos: 10 USD.

Ultimate 1. Pago mensual: 70 USD. 2. Prestaciones al mes: 3000 minutos, 1000 SMS y 30 GB de datos. 3. Si se exceden los límites del paquete:

1 minuto: 0.01 USD.

1 SMS: 0.01 USD.

1 GB de datos: 7 USD.

[25]:
```
# Imprime las condiciones de la tarifa y asegúrate de que te quedan claras
print(df_plans)
```

```
   messages_included  gb_per_month_included  minutes_included  \
0                 50                   15.0               500
1               1000                   30.0              3000

   usd_monthly_pay  usd_per_gb  usd_per_message  usd_per_minute plan_name
0               20          10             0.03            0.03      surf
1               70           7             0.01            0.01  ultimate
```

1.12 Agregar datos por usuario

[26]:
```
# Calcula el número de llamadas hechas por cada usuario al mes. Guarda el␣
 ↪resultado.
# Hacemos dos cálculos: una tabla dinámica para presentación del resultado y␣
 ↪una agrupación para cálculos posteriores.
calls_per_month=df_calls.pivot_table(index='user_id', columns='call_month',␣
 ↪values='id', aggfunc='count')
calls_per_month.fillna(0.0, inplace=True)
calls_per_month.head(10)
```

[26]:
```
call_month   1    2    3    4     5      6      7      8      9      10  \
user_id
1000        0.0  0.0  0.0  0.0   0.0    0.0    0.0    0.0    0.0    0.0
1001        0.0  0.0  0.0  0.0   0.0    0.0    0.0   27.0   49.0   65.0
1002        0.0  0.0  0.0  0.0   0.0    0.0    0.0    0.0    0.0   11.0
1003        0.0  0.0  0.0  0.0   0.0    0.0    0.0    0.0    0.0    0.0
1004        0.0  0.0  0.0  0.0  21.0   44.0   49.0   49.0   42.0   61.0
1005        0.0  0.0  0.0  0.0   0.0    0.0    0.0    0.0    0.0    0.0
1006        0.0  0.0  0.0  0.0   0.0    0.0    0.0    0.0    0.0    0.0
1007        0.0  0.0  0.0  0.0   0.0    0.0    0.0   70.0   63.0   80.0
1008        0.0  0.0  0.0  0.0   0.0    0.0    0.0    0.0    0.0   71.0
1009        0.0  0.0  0.0  0.0  71.0  110.0  124.0  109.0  116.0  114.0

call_month     11     12
user_id
1000          0.0   16.0
1001         64.0   56.0
1002         55.0   47.0
```

```
1003          0.0  149.0
1004         54.0   50.0
1005          0.0   59.0
1006          2.0    9.0
1007         80.0   87.0
1008         63.0   85.0
1009        105.0  107.0
```

Comentario del revisor

Muy buen trabajo!! la función de pivot_table() es muy recomendable para hacer los códigos más eficientes. Solamente te recomendaria que puedes agregar las siguientes varaibles a la función para que se vean más claros los resultados:

```
pivot_calls = calls.pivot_table(index=['user_id', 'month'],
                                values=['duration'],
                                aggfunc=['sum','count']).reset_index()
```

[27]:
```
# Calcula la cantidad de minutos usados por cada usuario al mes. Guarda el␣
 ↪resultado.
duration_calls=df_calls.pivot_table(index='user_id', columns='call_month',␣
 ↪values='duration', aggfunc='sum')
duration_calls.fillna(0.0, inplace=True)
duration_calls.head(20)
```

[27]:
```
call_month   1    2      3      4      5      6      7      8      9      10  \
user_id
1000        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1001        0.0  0.0    0.0    0.0    0.0    0.0    0.0  182.0  315.0  393.0
1002        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0   59.0
1003        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1004        0.0  0.0    0.0    0.0  193.0  275.0  381.0  354.0  301.0  365.0
1005        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1006        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1007        0.0  0.0    0.0    0.0    0.0    0.0    0.0  456.0  399.0  645.0
1008        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  476.0
1009        0.0  0.0    0.0    0.0  534.0  823.0  880.0  731.0  776.0  740.0
1010        0.0  0.0  429.0  656.0  532.0  553.0  698.0  637.0  601.0  711.0
1011        0.0  0.0    0.0    0.0    0.0  118.0  499.0  486.0  484.0  504.0
1012        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1013        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1014        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1015        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1016        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  173.0
1017        0.0  0.0    0.0    0.0    0.0    0.0    0.0   59.0  362.0  400.0
1018        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0  184.0
1019        0.0  0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
```

```
call_month     11      12
user_id
1000          0.0   124.0
1001        426.0   412.0
1002        386.0   384.0
1003          0.0  1104.0
1004        476.0   427.0
1005          0.0   496.0
1006         10.0    59.0
1007        524.0   617.0
1008        446.0   634.0
1009        714.0   756.0
1010        258.0     0.0
1011        505.0   311.0
1012         75.0    78.0
1013          0.0   219.0
1014        163.0  1114.0
1015          0.0    96.0
1016        649.0   837.0
1017        384.0   476.0
1018        636.0   476.0
1019         44.0   467.0
```

[28]:
```
# Calcula el número de mensajes enviados por cada usuario al mes. Guarda el␣
 ↪resultado.
mess_per_month=df_messages.pivot_table(index='user_id',␣
 ↪columns='message_month', values='id', aggfunc='count')
mess_per_month.fillna(0.0, inplace=True)
mess_per_month.sample(20)
```

[28]:
```
message_month   1     2     3     4     5      6      7      8      9      10  \
user_id
1167           0.0   0.0   0.0   0.0  16.0   49.0   59.0   31.0   44.0   52.0
1472           0.0   0.0   0.0  27.0  41.0   36.0   47.0   40.0   40.0   16.0
1356           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0   13.0   37.0
1315           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0    0.0    0.0
1096           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0    0.0    0.0
1470           0.0   0.0   0.0   0.0  51.0  133.0  147.0  141.0  130.0  145.0
1235           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0    0.0    0.0
1198           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0    7.0   11.0
1251           0.0   0.0  20.0  33.0  39.0   33.0   39.0   20.0   29.0   33.0
1255           0.0   0.0   0.0   0.0   0.0    0.0    0.0   54.0   47.0   57.0
1261           0.0  11.0  54.0  37.0  49.0   52.0   49.0   49.0   49.0   57.0
1351           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0    0.0    2.0
1060           0.0   0.0   0.0   0.0   0.0    0.0    0.0   38.0   66.0   74.0
1482           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0    0.0    2.0
1014           0.0   0.0   0.0   0.0   0.0    0.0    0.0    0.0    0.0    0.0
```

```
1036          0.0   0.0   0.0   0.0   0.0   19.0   63.0   77.0   83.0   75.0
1247          0.0   0.0   0.0   0.0   0.0    0.0   29.0  119.0  109.0  110.0
1254          0.0   0.0   0.0   0.0   0.0    0.0    0.0  106.0  112.0  106.0
1257          0.0   0.0   0.0   4.0  35.0   32.0   37.0   34.0   34.0   30.0
1199          0.0   0.0   0.0   0.0   0.0    0.0    0.0   80.0   71.0   72.0
```

```
message_month     11     12
user_id
1167            41.0   52.0
1472             0.0    0.0
1356            25.0   37.0
1315            21.0   35.0
1096             0.0   14.0
1470           156.0  153.0
1235            33.0   28.0
1198             6.0   12.0
1251            29.0   35.0
1255            49.0   60.0
1261            46.0   50.0
1351            18.0   19.0
1060            69.0   83.0
1482            87.0    5.0
1014             9.0   64.0
1036            76.0   45.0
1247           121.0  112.0
1254            91.0  110.0
1257            25.0   39.0
1199            61.0   67.0
```

[29]:
```
# Calcula el volumen del tráfico de Internet usado por cada usuario al mes.␣
 ↪Guarda el resultado.
session_per_month=df_internet.pivot_table(index='user_id',␣
 ↪columns='session_month', values='gb_used', aggfunc='sum')
session_per_month.fillna(0.0, inplace=True)
session_per_month=session_per_month.add_suffix('_int')
session_per_month.head(20)
```

[29]:
```
session_month  1_int  2_int  3_int  4_int  5_int  6_int  7_int  8_int  9_int  \
user_id
1000             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1001             0.0    0.0    0.0    0.0    0.0    0.0    0.0   21.0   41.0
1002             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1003             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1004             0.0    0.0    0.0    0.0   14.0   58.0   61.0   70.0   48.0
1005             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1006             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1007             0.0    0.0    0.0    0.0    0.0    0.0    0.0   45.0   52.0
```

```
1008             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1009             0.0    0.0    0.0    0.0   39.0   37.0   62.0   51.0   46.0
1010             0.0    0.0   29.0   49.0   46.0   42.0   29.0   47.0   45.0
1011             0.0    0.0    0.0    0.0    0.0   19.0   58.0   58.0   45.0
1012             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1013             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1014             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1015             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1016             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1017             0.0    0.0    0.0    0.0    0.0    0.0    0.0    6.0   60.0
1018             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0
1019             0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0    0.0

session_month  10_int  11_int  12_int
user_id
1000              0.0     0.0     4.0
1001             50.0    49.0    55.0
1002             15.0    45.0    37.0
1003              0.0     0.0    53.0
1004             42.0    56.0    50.0
1005              0.0     0.0    51.0
1006              0.0     7.0    62.0
1007             65.0    47.0    59.0
1008             48.0    56.0    40.0
1009             44.0    49.0    50.0
1010             50.0    19.0     0.0
1011             65.0    51.0    48.0
1012              0.0    32.0    22.0
1013              0.0     0.0    57.0
1014              0.0     2.0    17.0
1015              0.0     0.0    39.0
1016              8.0    52.0    53.0
1017             77.0    50.0    48.0
1018             22.0    38.0    45.0
1019              0.0    10.0    78.0
```

[Junta los datos agregados en un DataFrame para que haya un registro que represente lo que consumió un usuario único en un mes determinado.]

[30]:
```
# Fusiona los datos de llamadas, minutos, mensajes e Internet con base en␣
 ↪user_id y month
merge1=duration_calls.merge(mess_per_month, on='user_id', how='outer',␣
 ↪suffixes=['_call','_mess'])
merge1.head()
```

[30]:
```
         1_call  2_call  3_call  4_call  5_call  6_call  7_call  8_call  \
user_id
```

```
1000        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1001        0.0     0.0     0.0     0.0     0.0     0.0     0.0   182.0
1002        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1003        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1004        0.0     0.0     0.0     0.0   193.0   275.0   381.0   354.0

         9_call  10_call  …  3_mess  4_mess  5_mess  6_mess  7_mess  8_mess  \
user_id                   …
1000        0.0      0.0  …     0.0     0.0     0.0     0.0     0.0     0.0
1001      315.0    393.0  …     0.0     0.0     0.0     0.0     0.0    30.0
1002        0.0     59.0  …     0.0     0.0     0.0     0.0     0.0     0.0
1003        0.0      0.0  …     0.0     0.0     0.0     0.0     0.0     0.0
1004      301.0    365.0  …     0.0     0.0     7.0    18.0    26.0    25.0

         9_mess  10_mess  11_mess  12_mess
user_id
1000        0.0      0.0      0.0     11.0
1001       44.0     53.0     36.0     44.0
1002        0.0     15.0     32.0     41.0
1003        0.0      0.0      0.0     50.0
1004       21.0     24.0     25.0     31.0

[5 rows x 24 columns]
```

[31]:
```
merge2=merge1.merge(session_per_month, on='user_id', how='outer')
merge2.fillna(0.0, inplace=True)
merge2
```

[31]:
```
         1_call  2_call  3_call  4_call  5_call  6_call  7_call  8_call  \
user_id
1000        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1001        0.0     0.0     0.0     0.0     0.0     0.0     0.0   182.0
1002        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1003        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1004        0.0     0.0     0.0     0.0   193.0   275.0   381.0   354.0
…           …       …       …       …       …       …       …       …
1137        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1194        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1204        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1349        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0
1108        0.0     0.0     0.0     0.0     0.0     0.0     0.0     0.0

         9_call  10_call  …  3_int  4_int  5_int  6_int  7_int  8_int  \
user_id                   …
1000        0.0      0.0  …    0.0    0.0    0.0    0.0    0.0    0.0
1001      315.0    393.0  …    0.0    0.0    0.0    0.0    0.0   21.0
1002        0.0     59.0  …    0.0    0.0    0.0    0.0    0.0    0.0
```

```
1003        0.0      0.0  …    0.0    0.0    0.0    0.0    0.0    0.0
1004      301.0    365.0  …    0.0    0.0   14.0   58.0   61.0   70.0
…           …        …    …    …      …     …      …      …      …
1137        0.0      0.0  …    0.0    0.0    0.0    0.0    0.0    0.0
1194        0.0      0.0  …    0.0    0.0    0.0    0.0    0.0   11.0
1204        0.0      0.0  …    0.0    0.0    0.0    0.0    0.0    0.0
1349        0.0      0.0  …    0.0    0.0    0.0    0.0    0.0    0.0
1108        0.0      0.0  …    0.0    0.0    0.0    0.0    0.0    0.0

         9_int  10_int  11_int  12_int
user_id
1000       0.0     0.0     0.0     4.0
1001      41.0    50.0    49.0    55.0
1002       0.0    15.0    45.0    37.0
1003       0.0     0.0     0.0    53.0
1004      48.0    42.0    56.0    50.0
…          …       …       …       …
1137       0.0    15.0     8.0    15.0
1194      72.0    57.0    79.0    64.0
1204       0.0     0.0    39.0    78.0
1349       0.0    27.0    34.0    27.0
1108       0.0     0.0     0.0     2.0

[490 rows x 36 columns]
```

[32]:
```
# Añade la información de la tarifa
user_plan=df_users[['user_id','plan']]
merge3=merge2.merge(user_plan, on='user_id', how='outer')
merge3.fillna(0.0, inplace=True)
merge3.head(20)
```

[32]:

	user_id	1_call	2_call	3_call	4_call	5_call	6_call	7_call	8_call
0	1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	182.0
2	1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	1003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	1004	0.0	0.0	0.0	0.0	193.0	275.0	381.0	354.0
5	1005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6	1006	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
7	1007	0.0	0.0	0.0	0.0	0.0	0.0	0.0	456.0
8	1008	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
9	1009	0.0	0.0	0.0	0.0	534.0	823.0	880.0	731.0
10	1010	0.0	0.0	429.0	656.0	532.0	553.0	698.0	637.0
11	1011	0.0	0.0	0.0	0.0	0.0	118.0	499.0	486.0
12	1012	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
13	1013	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
14	1014	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

15	1015	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
16	1016	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17	1017	0.0	0.0	0.0	0.0	0.0	0.0	0.0	59.0
18	1018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19	1019	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

	9_call	…	4_int	5_int	6_int	7_int	8_int	9_int	10_int	11_int	\
0	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	315.0	…	0.0	0.0	0.0	0.0	21.0	41.0	50.0	49.0	
2	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	15.0	45.0	
3	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	301.0	…	0.0	14.0	58.0	61.0	70.0	48.0	42.0	56.0	
5	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
6	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0	
7	399.0	…	0.0	0.0	0.0	0.0	45.0	52.0	65.0	47.0	
8	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	48.0	56.0	
9	776.0	…	0.0	39.0	37.0	62.0	51.0	46.0	44.0	49.0	
10	601.0	…	49.0	46.0	42.0	29.0	47.0	45.0	50.0	19.0	
11	484.0	…	0.0	0.0	19.0	58.0	58.0	45.0	65.0	51.0	
12	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	32.0	
13	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
14	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	2.0	
15	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	8.0	52.0	
17	362.0	…	0.0	0.0	0.0	0.0	6.0	60.0	77.0	50.0	
18	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	22.0	38.0	
19	0.0	…	0.0	0.0	0.0	0.0	0.0	0.0	0.0	10.0	

	12_int	plan
0	4.0	ultimate
1	55.0	surf
2	37.0	surf
3	53.0	surf
4	50.0	surf
5	51.0	surf
6	62.0	ultimate
7	59.0	surf
8	40.0	ultimate
9	50.0	surf
10	0.0	surf
11	48.0	ultimate
12	22.0	surf
13	57.0	ultimate
14	17.0	surf
15	39.0	surf
16	53.0	surf
17	48.0	surf

```
18    45.0      surf
19    78.0      surf

[20 rows x 38 columns]
```

[33]:
```
#EB: Uso el parámetro 'left' para conservar el orden del 'user_id'.
merged_data=merge3.merge(df_plans, left_on='plan', right_on='plan_name',␣
 ↪how='left')
merged_data.drop('plan_name', axis='columns')
merged_data.head(20)
```

[33]:

	user_id	1_call	2_call	3_call	4_call	5_call	6_call	7_call	8_call	\
0	1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	182.0	
2	1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	1004	0.0	0.0	0.0	0.0	193.0	275.0	381.0	354.0	
5	1005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
6	1006	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	1007	0.0	0.0	0.0	0.0	0.0	0.0	0.0	456.0	
8	1008	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
9	1009	0.0	0.0	0.0	0.0	534.0	823.0	880.0	731.0	
10	1010	0.0	0.0	429.0	656.0	532.0	553.0	698.0	637.0	
11	1011	0.0	0.0	0.0	0.0	0.0	118.0	499.0	486.0	
12	1012	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13	1013	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
14	1014	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
15	1015	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16	1016	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17	1017	0.0	0.0	0.0	0.0	0.0	0.0	0.0	59.0	
18	1018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
19	1019	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

	9_call	…	12_int	plan	messages_included	gb_per_month_included	\
0	0.0	…	4.0	ultimate	1000	30.0	
1	315.0	…	55.0	surf	50	15.0	
2	0.0	…	37.0	surf	50	15.0	
3	0.0	…	53.0	surf	50	15.0	
4	301.0	…	50.0	surf	50	15.0	
5	0.0	…	51.0	surf	50	15.0	
6	0.0	…	62.0	ultimate	1000	30.0	
7	399.0	…	59.0	surf	50	15.0	
8	0.0	…	40.0	ultimate	1000	30.0	
9	776.0	…	50.0	surf	50	15.0	
10	601.0	…	0.0	surf	50	15.0	
11	484.0	…	48.0	ultimate	1000	30.0	
12	0.0	…	22.0	surf	50	15.0	

```
13     0.0  …    57.0  ultimate               1000                    30.0
14     0.0  …    17.0      surf                 50                    15.0
15     0.0  …    39.0      surf                 50                    15.0
16     0.0  …    53.0      surf                 50                    15.0
17   362.0  …    48.0      surf                 50                    15.0
18     0.0  …    45.0      surf                 50                    15.0
19     0.0  …    78.0      surf                 50                    15.0

    minutes_included  usd_monthly_pay  usd_per_gb  usd_per_message  \
0               3000               70           7             0.01
1                500               20          10             0.03
2                500               20          10             0.03
3                500               20          10             0.03
4                500               20          10             0.03
5                500               20          10             0.03
6               3000               70           7             0.01
7                500               20          10             0.03
8               3000               70           7             0.01
9                500               20          10             0.03
10               500               20          10             0.03
11              3000               70           7             0.01
12               500               20          10             0.03
13              3000               70           7             0.01
14               500               20          10             0.03
15               500               20          10             0.03
16               500               20          10             0.03
17               500               20          10             0.03
18               500               20          10             0.03
19               500               20          10             0.03

    usd_per_minute  plan_name
0             0.01   ultimate
1             0.03       surf
2             0.03       surf
3             0.03       surf
4             0.03       surf
5             0.03       surf
6             0.01   ultimate
7             0.03       surf
8             0.01   ultimate
9             0.03       surf
10            0.03       surf
11            0.01   ultimate
12            0.03       surf
13            0.01   ultimate
14            0.03       surf
15            0.03       surf
```

```
16            0.03       surf
17            0.03       surf
18            0.03       surf
19            0.03       surf

[20 rows x 46 columns]
```

Calculo los ingresos mensuales por usuario mediante un programa y un bucle.

[34]:
```python
merg_tarif=merged_data
# Calcula el ingreso mensual para cada usuario
def call_month (fila):
    plan=fila['plan']
    jan_c=fila[cm]
    jan_m=fila[ms]
    jan_i=fila[im]
    tarif=fila['usd_monthly_pay']
    callmin=fila['usd_per_minute']
    messex=fila['usd_per_message']
    gbex=fila['usd_per_gb']
    extra=0

    if plan=='surf':
        if jan_c >500:
            extra+=(jan_c-500)*callmin
        else:
            extra+=0
        if jan_m >50:
            extra+=(jan_m-50)*messex
        else:
            extra+=0
        if jan_i >15:
            extra+=(jan_i-15)*gbex
        else:
            extra+=0
        return tarif+extra

    else:
        if jan_c >3000:
            extra+=(jan_c-3000)*callmin
        else:
            extra+=0
        if jan_m >1000:
            extra+=(jan_m-1000)*messex
        else:
            extra+=0
        if jan_i >30:
```

```
            extra+=(jan_i-30)*gbex
        else:
            extra+=0
        return tarif+extra

months=[1,2,3,4,5,6,7,8,9,10,11,12]
for month in months:
    month=str(month)
    cm=month+'_call'
    ms=month+'_mess'
    im=month+'_int'
    tarf=month+'_tar'
    merg_tarif[tarf]=merg_tarif.apply(call_month,axis=1)
```

[65]:
```
# EB: Por si acaso verificamos dulicados en la tabla final de tarifas.
print(merg_tarif.duplicated().sum())
merg_tarif.head(50)
```

```
0
```

[65]:

	user_id	1_call	2_call	3_call	4_call	5_call	6_call	7_call	8_call	\
0	1000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
1	1001	0.0	0.0	0.0	0.0	0.0	0.0	0.0	182.0	
2	1002	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	1003	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	1004	0.0	0.0	0.0	0.0	193.0	275.0	381.0	354.0	
5	1005	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
6	1006	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7	1007	0.0	0.0	0.0	0.0	0.0	0.0	0.0	456.0	
8	1008	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
9	1009	0.0	0.0	0.0	0.0	534.0	823.0	880.0	731.0	
10	1010	0.0	0.0	429.0	656.0	532.0	553.0	698.0	637.0	
11	1011	0.0	0.0	0.0	0.0	0.0	118.0	499.0	486.0	
12	1012	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13	1013	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
14	1014	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
15	1015	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16	1016	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
17	1017	0.0	0.0	0.0	0.0	0.0	0.0	0.0	59.0	
18	1018	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
19	1019	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
20	1020	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
21	1021	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
22	1022	0.0	0.0	0.0	0.0	302.0	490.0	475.0	631.0	
23	1023	0.0	0.0	0.0	0.0	0.0	0.0	43.0	72.0	
24	1024	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

25	1026	0.0	0.0	0.0	0.0	0.0	0.0	163.0	194.0
26	1027	0.0	0.0	0.0	0.0	0.0	0.0	347.0	361.0
27	1028	0.0	0.0	39.0	53.0	67.0	49.0	53.0	76.0
28	1029	0.0	0.0	0.0	0.0	0.0	0.0	0.0	43.0
29	1030	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
30	1031	0.0	0.0	0.0	0.0	0.0	0.0	304.0	483.0
31	1032	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
32	1033	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
33	1034	0.0	0.0	0.0	0.0	0.0	0.0	0.0	7.0
34	1035	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
35	1036	0.0	0.0	0.0	0.0	0.0	161.0	537.0	512.0
36	1037	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
37	1038	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
38	1039	0.0	0.0	0.0	0.0	0.0	323.0	460.0	417.0
39	1040	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
40	1041	0.0	0.0	23.0	453.0	407.0	311.0	533.0	493.0
41	1042	114.0	254.0	214.0	279.0	489.0	372.0	374.0	358.0
42	1043	0.0	0.0	0.0	0.0	0.0	79.0	321.0	116.0
43	1044	0.0	0.0	0.0	0.0	0.0	0.0	0.0	13.0
44	1045	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
45	1046	0.0	0.0	0.0	0.0	0.0	350.0	509.0	507.0
46	1047	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
47	1048	0.0	0.0	0.0	0.0	0.0	0.0	0.0	72.0
48	1049	0.0	0.0	0.0	234.0	300.0	341.0	296.0	380.0
49	1050	0.0	0.0	135.0	353.0	418.0	471.0	430.0	358.0

	9_call	…	3_tar	4_tar	5_tar	6_tar	7_tar	8_tar	9_tar	\
0	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00	
1	315.0	…	20.0	20.00	20.00	20.00	20.00	80.00	280.00	
2	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
3	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
4	301.0	…	20.0	20.00	20.00	450.00	480.00	570.00	350.00	
5	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
6	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00	
7	399.0	…	20.0	20.00	20.00	20.00	20.00	320.03	390.00	
8	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00	
9	776.0	…	20.0	20.00	261.02	249.69	501.40	386.93	338.28	
10	601.0	…	160.0	364.68	330.96	291.59	165.94	344.11	323.03	
11	484.0	…	70.0	70.00	70.00	70.00	266.00	266.00	175.00	
12	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
13	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00	
14	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
15	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
16	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
17	362.0	…	20.0	20.00	20.00	20.00	20.00	20.00	470.00	
18	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	
19	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00	

20	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00
21	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00
22	510.0	…	20.0	20.00	20.00	520.00	200.00	503.93	320.30
23	0.0	…	20.0	20.00	20.00	20.00	60.00	110.00	20.00
24	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00
25	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00
26	325.0	…	20.0	20.00	20.00	20.00	220.00	340.00	300.00
27	37.0	…	315.0	371.00	329.00	357.00	385.00	392.00	266.00
28	763.0	…	20.0	20.00	20.00	20.00	20.00	20.00	257.89
29	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00
30	515.0	…	70.0	70.00	70.00	70.00	70.00	273.00	112.00
31	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00
32	167.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00
33	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00
34	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00
35	299.0	…	70.0	70.00	70.00	70.00	105.00	175.00	154.00
36	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00
37	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00
38	500.0	…	70.0	70.00	70.00	70.00	252.00	224.00	196.00
39	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00
40	298.0	…	70.0	273.00	224.00	224.00	280.00	343.00	273.00
41	252.0	…	20.0	50.00	60.00	20.00	40.00	30.00	140.00
42	200.0	…	70.0	70.00	70.00	70.00	273.00	280.00	343.00
43	346.0	…	20.0	20.00	20.00	20.00	20.00	20.00	350.00
44	0.0	…	20.0	20.00	20.00	20.00	20.00	20.00	20.00
45	522.0	…	20.0	20.00	20.00	200.00	440.30	540.21	470.66
46	0.0	…	70.0	70.00	70.00	70.00	70.00	70.00	70.00
47	260.0	…	20.0	20.00	20.00	20.00	20.00	20.00	80.00
48	356.0	…	20.0	20.00	100.00	150.00	100.00	110.00	150.00
49	371.0	…	70.0	168.00	147.00	119.00	119.00	189.00	196.00

	10_tar	11_tar	12_tar
0	70.00	70.00	70.00
1	370.09	360.00	420.00
2	20.00	320.00	240.00
3	20.00	20.00	418.12
4	290.00	430.00	370.00
5	20.00	20.00	380.00
6	70.00	70.00	294.00
7	524.62	340.72	463.51
8	196.00	252.00	140.00
9	317.20	366.42	377.68
10	376.33	60.00	20.00
11	315.00	217.00	196.00
12	20.00	190.00	90.00
13	70.00	70.00	259.00
14	20.00	20.00	58.84

```
15   20.00   20.00  260.54
16   20.00  394.47  410.11
17  640.00  370.00  350.00
18   90.00  254.08  320.00
19   20.00   20.00  652.34
20   20.00  190.00  420.00
21   20.00   20.00   20.00
22  402.22  480.00  472.55
23   20.00   20.00   20.00
24   20.00   20.00  151.92
25   70.00   70.00   70.00
26  220.00  290.00  230.00
27  476.00  448.00  406.00
28  305.76  204.14  353.96
29  112.00  168.00  154.00
30  189.00  161.00  112.00
31   70.00  119.00  182.00
32  329.00  329.00  245.00
33   20.00   20.00   20.00
34   20.00   20.00  274.29
35   70.00   70.00   70.00
36   70.00   70.00  119.00
37  140.00  420.00  469.00
38  133.00  301.00  231.00
39   20.00   20.00  240.00
40  224.00  378.00  175.00
41   20.00   50.00   20.00
42  252.00  287.00  357.00
43  270.00  310.00  160.00
44  120.00  562.01  363.69
45  502.28  550.00  294.32
46   70.00   70.00  196.00
47  180.00  120.00  140.00
48   70.00  100.00  110.00
49  273.00  126.00  175.00

[50 rows x 58 columns]
```

1.13 Estudia el comportamiento de usuario

Cálculo de estadísticas descriptivas que sean útiles y que muestren un panorama general.

1.13.1 Llamadas

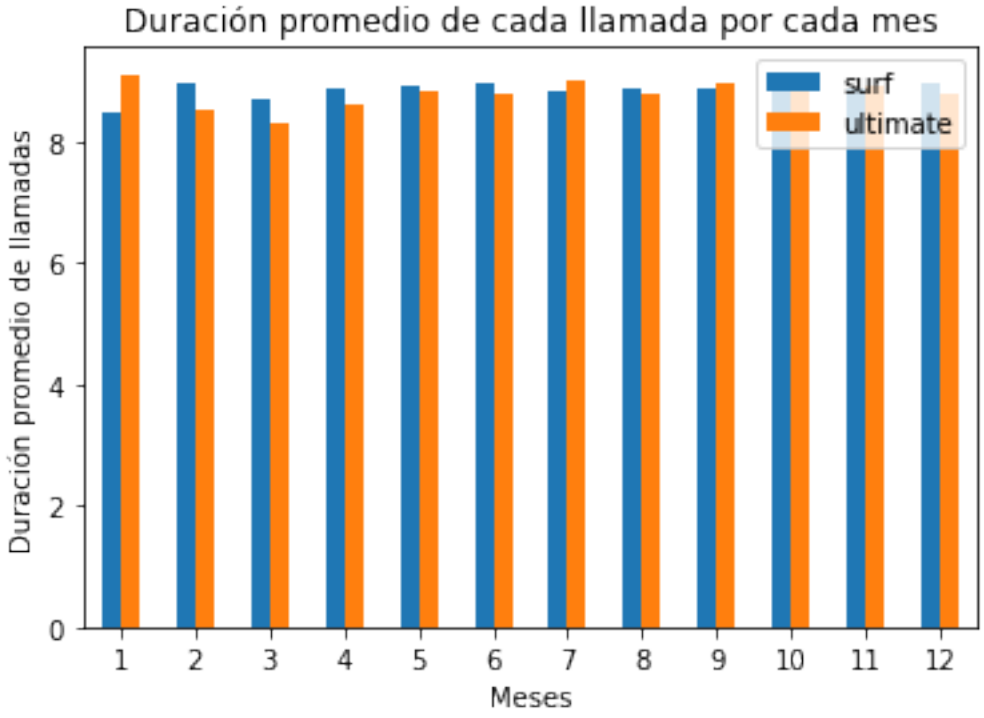
[36]:

```
# Compara la duración promedio de llamadas por cada plan y por cada mes. Traza un gráfico de barras para visualizarla.
# EB: Uno la tabla de llamadas con la de usuarios para extraer cada plan por separado.
call_min_list=df_calls.merge(df_users[['user_id','plan']],on='user_id',how='left')

# EB: Para evitar distorsiones, para el cálculo de la duración promedio de las llamadas
# usamos las entradas de llamada cuya duración haya sido mayor a 0.
call_min_surf1=call_min_list.query("plan=='surf' and duration > 0.0")
call_min_surf11=call_min_surf1.groupby(['call_month'])['duration'].mean()

grp_surf11_mean=call_min_surf11.reset_index()
grp_surf11_mean.rename(columns={'duration':'surf'}, inplace=True)
grp_surf11_mean.set_index("call_month", inplace = True)
print(grp_surf11_mean)

call_min_ultimate1=call_min_list.query("plan=='ultimate' and duration > 0.0")
call_min_ultimate11=call_min_ultimate1.groupby(['call_month'])['duration'].mean()

grp_ultimate11_mean=call_min_ultimate11.reset_index()
grp_ultimate11_mean.rename(columns={'duration':'ultimate'}, inplace=True)
grp_ultimate11_mean.set_index("call_month", inplace = True)
print(grp_ultimate11_mean)
```

```
                surf
call_month
1           8.458333
2           8.969799
3           8.714122
4           8.854912
5           8.933798
6           8.948877
7           8.817490
8           8.882086
9           8.859978
10          8.846174
11          8.857745
12          8.972786
            ultimate
call_month
1           9.105882
2           8.525680
3           8.316629
4           8.626829
```

```
5           8.837959
6           8.801111
7           9.018253
8           8.779930
9           8.941540
10          8.834689
11          8.912358
12          8.805284
```

[37]:
```
plotdata1=grp_surf11_mean.merge(grp_ultimate11_mean, on='call_month',␣
 ↪how='outer')
plotdata1.plot(kind='bar', title='Duración promedio de cada llamada por cada␣
 ↪mes', xlabel='Meses', ylabel='Duración promedio de llamadas', rot=0)

plt.show()
```



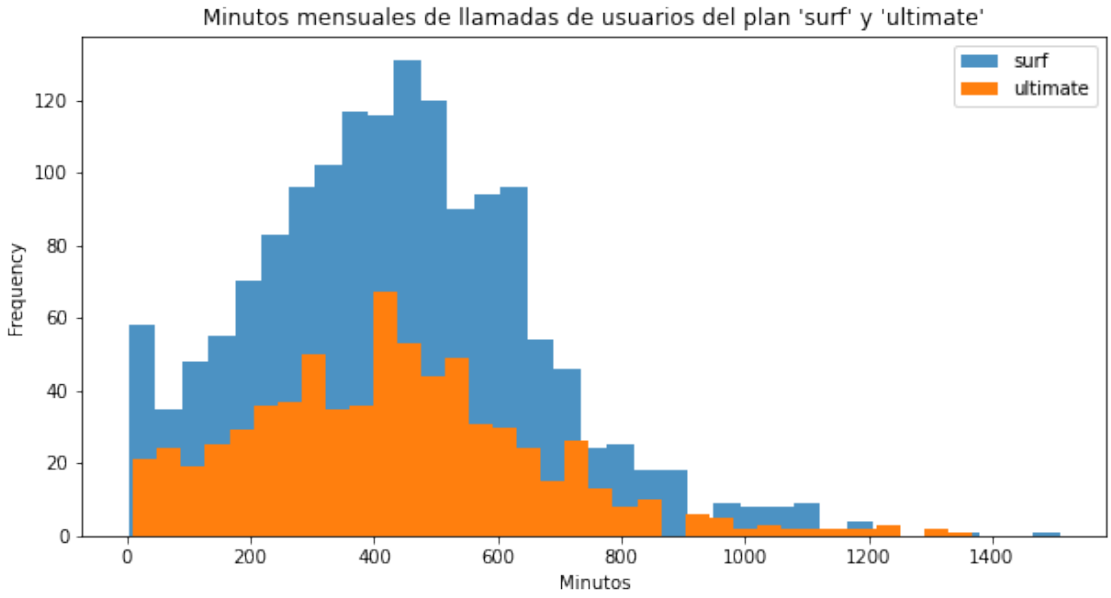
[38]:
```
# Compara el número de minutos mensuales que necesitan los usuarios de cada␣
 ↪plan. Traza un histograma.
# EB: Preparo los datos agrupando la suma de la duración de llamadas por␣
 ↪usuario y por mes.
call_mon_surf=call_min_surf1.groupby(['user_id','plan', 'call_month'])
call_mon_surf2=call_mon_surf['duration'].sum()
print(call_mon_surf2)
```

```
call_mon_ultimate=call_min_ultimate1.groupby(['user_id','plan', 'call_month'])
call_mon_ultimate2=call_mon_ultimate['duration'].sum()
print(call_mon_ultimate2)
```

```
user_id  plan  call_month
1001     surf  8             182.0
               9             315.0
               10            393.0
               11            426.0
               12            412.0
                             …
1498     surf  12            339.0
1499     surf  9             346.0
               10            385.0
               11            308.0
               12            496.0
Name: duration, Length: 1544, dtype: float64
user_id  plan      call_month
1000     ultimate  12            124.0
1006     ultimate  11             10.0
                   12             59.0
1008     ultimate  10            476.0
                   11            446.0
                                 …
1493     ultimate  9             529.0
                   10            450.0
                   11            500.0
                   12            473.0
1497     ultimate  12            300.0
Name: duration, Length: 712, dtype: float64
```

[39]:
```
call_mon_surf2.plot(kind='hist', title="Minutos mensuales de llamadas de␣
 ↪usuarios del plan 'surf' y 'ultimate'", figsize=(10,5), alpha=0.8, bins=35).
 ↪set_xlabel('Minutos')
call_mon_ultimate2.plot(kind='hist', figsize=(10,5), bins=35)
plt.legend(['surf', 'ultimate'])
```

[39]:
```
<matplotlib.legend.Legend at 0x7f8e038e6190>
```



[40]:
```
# Calcula la media y la varianza de la duración mensual de llamadas.
print("La media de la duración mensual de llamadas del plan 'surf' es:\n",␣
 ↪call_mon_surf2.mean(), "\n")
print("La varianza de la duración mensual de llamadas del plan 'surf' es:\n",␣
 ↪np.var(call_mon_surf2), "\n\n")

print("La media de la duración mensual de llamadas del plan 'ultimate' es:\n",␣
 ↪call_mon_ultimate2.mean(), "\n")
print("La varianza de la duración mensual de llamadas del plan 'ultimate' es:
 ↪\n", np.var(call_mon_ultimate2), "\n")
```

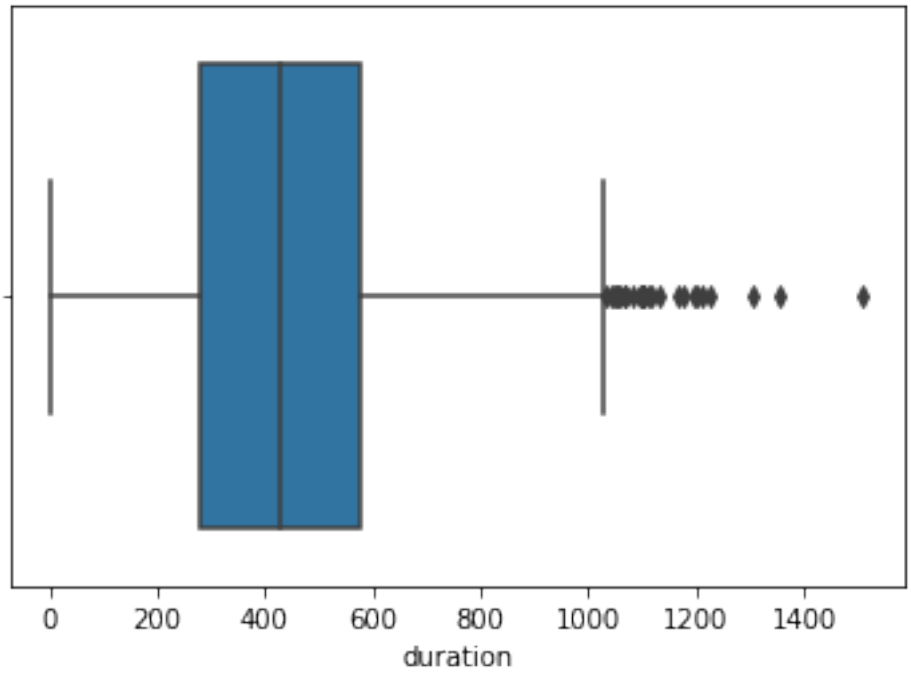
```
La media de la duración mensual de llamadas del plan 'surf' es:
 436.80246113989637

La varianza de la duración mensual de llamadas del plan 'surf' es:
 52447.570434357294


La media de la duración mensual de llamadas del plan 'ultimate' es:
 435.2865168539326

La varianza de la duración mensual de llamadas del plan 'ultimate' es:
 56307.88981820476
```

[41]:
```
# Traza un diagrama de caja para visualizar la distribución de la duración␣
 ↪mensual de llamadas
```

```
print("Diagrama de caja de la duración mensual de llamadas del plan 'surf'")
sns.boxplot(call_mon_surf2)
```

```
Diagrama de caja de la duración mensual de llamadas del plan 'surf'

/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

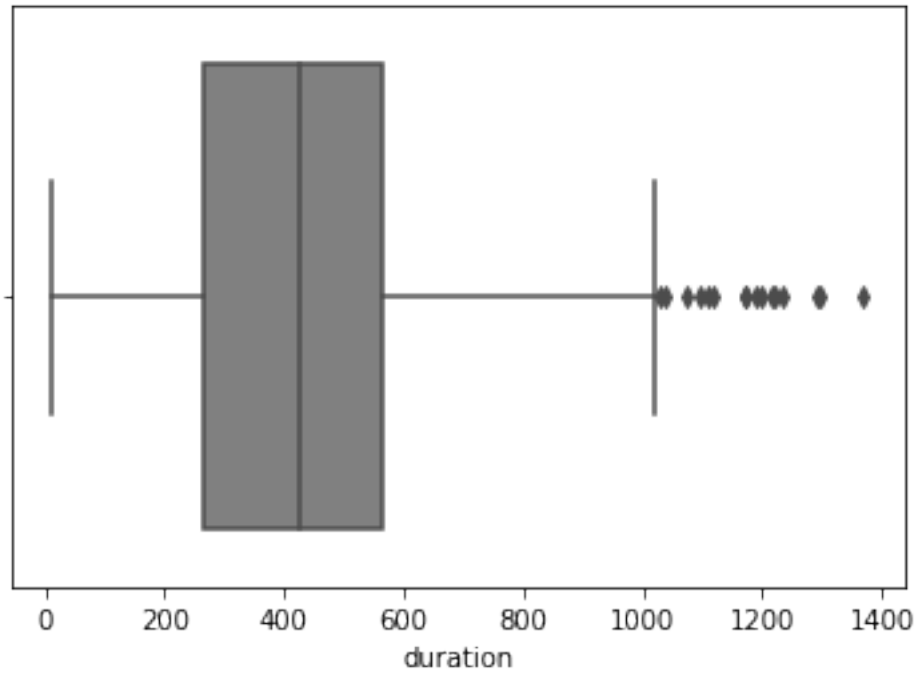
[41]: <AxesSubplot:xlabel='duration'>



[42]:
```
print("Diagrama de caja de la duración mensual de llamadas del plan 'ultimate'")
sns.boxplot(call_mon_ultimate2, color='0.5')
```

```
Diagrama de caja de la duración mensual de llamadas del plan 'ultimate'

/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

[42]: <AxesSubplot:xlabel='duration'>



Está claro que la mayoría de clientes usan el plan Surf. Esta diferencia de volumen es evidente en las gráficas de histogramas.

No se ven diferencias en la duración promedio de las llamadas entre los usuarios de ambos planes.

Asimismo tampoco se ven diferencias estadísticas importantes en los minutos que necesitan los usuarios paras las llamadas. Las medias y las varianzas son muy similares

Más de la mitad de usuarios del plan Surf se exceden considerablemente del límite de minutos mensuales de su plan; en casos atípicos, más del doble de los 500 minutos asignados en el plan. Por otro lado, no hay registros de usuarios del plan Ultimate excediéndose de su límite.

1.13.2 Mensajes

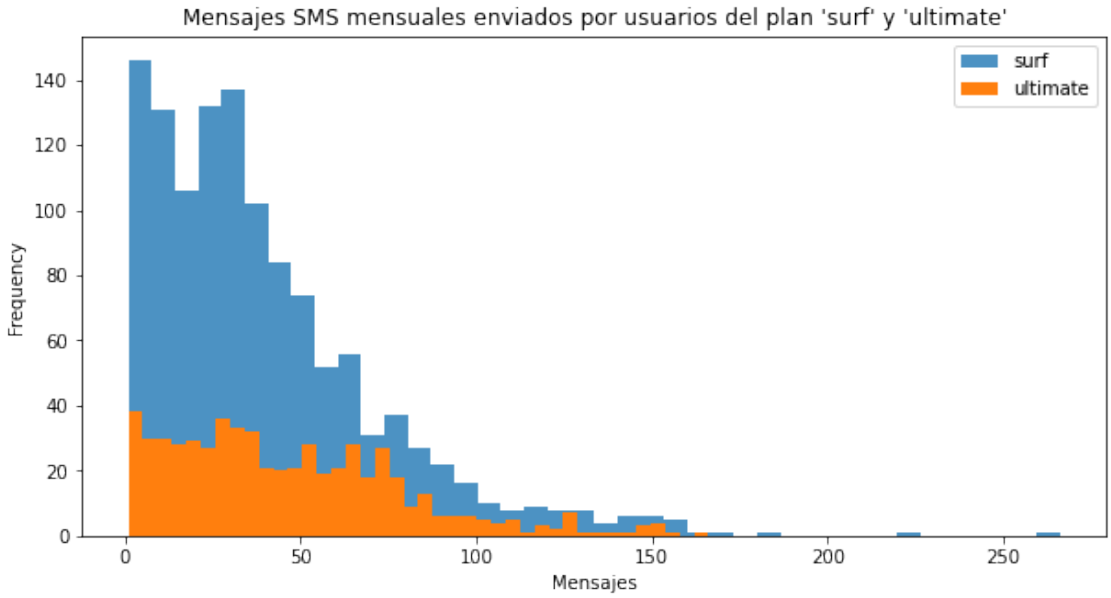
[43]:
```
# Comprara el número de mensajes que tienden a enviar cada mes los usuarios de␣
 ↪cada plan
mess_month_list=df_messages.merge(df_users[['user_id',␣
 ↪'plan']],on='user_id',how='left')
mess_month_surf1=mess_month_list.query("plan=='surf'")
mess_month_surf2=mess_month_surf1.groupby(['user_id','message_month'])['id'].
 ↪count()
print(mess_month_surf2,'\n\n')

mess_month_ultimate1=mess_month_list.query("plan=='ultimate'")
mess_month_ultimate2=mess_month_ultimate1.
 ↪groupby(['user_id','message_month'])['id'].count()
```

```
print(mess_month_ultimate2)
```

```
user_id  message_month
1001     8                30
         9                44
         10               53
         11               36
         12               44
                          ..
1496     8                 2
         9                21
         10               18
         11               13
         12               11
Name: id, Length: 1222, dtype: int64


user_id  message_month
1000     12                11
1006     11                15
         12               139
1008     10                21
         11                37
                          ...
1482     10                 2
         11                87
         12                 5
1487     12                66
1497     12                50
Name: id, Length: 584, dtype: int64
```

[44]:
```
# Comparación del número de mensajes mensuales que necesitan los usuarios de␣
 ↪cada plan.
mess_month_surf2.plot(kind='hist', title="Mensajes SMS mensuales enviados por␣
 ↪usuarios del plan 'surf' y 'ultimate'", figsize=(10,5), alpha=0.8, bins=40)
mess_month_ultimate2.plot(kind='hist', figsize=(10,5), bins=40).
 ↪set_xlabel('Mensajes')
plt.legend(['surf', 'ultimate'])
```

[44]:
```
<matplotlib.legend.Legend at 0x7f8e032aac10>
```



[45]:
```
# Calcula la media y la varianza de mensajes enviados mensualmente.
print("La media del número mensual de mensajes del plan 'surf' es:\n",␣
 ↪mess_month_surf2.mean(), "\n")
print("La varianza del número mensual de mensajes del plan 'surf' es:\n", np.
 ↪var(mess_month_surf2), "\n\n")

print("La media del número mensual de mensajes del plan 'ultimate' es:\n",␣
 ↪mess_month_ultimate2.mean(), "\n")
print("La varianza del número mensual de mensajes del plan 'ultimate' es:\n",␣
 ↪np.var(mess_month_ultimate2), "\n")
```

```
La media del número mensual de mensajes del plan 'surf' es:
 40.10965630114566

La varianza del número mensual de mensajes del plan 'surf' es:
 1090.4511506183685


La media del número mensual de mensajes del plan 'ultimate' es:
 46.29623287671233

La varianza del número mensual de mensajes del plan 'ultimate' es:
 1083.3249173156314
```

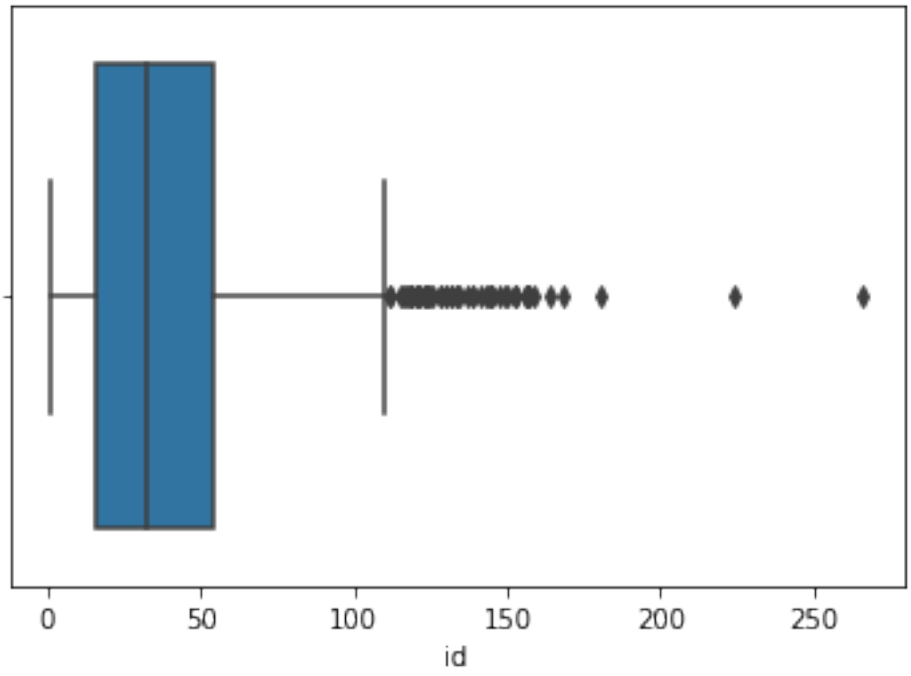
[46]:
```
print("Diagrama de caja del número mensual de mensajes de usuarios del plan␣
 ↪'surf'")
```

```
sns.boxplot(mess_month_surf2)
```

Diagrama de caja del número mensual de mensajes de usuarios del plan 'surf'

```
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

[46]: <AxesSubplot:xlabel='id'>

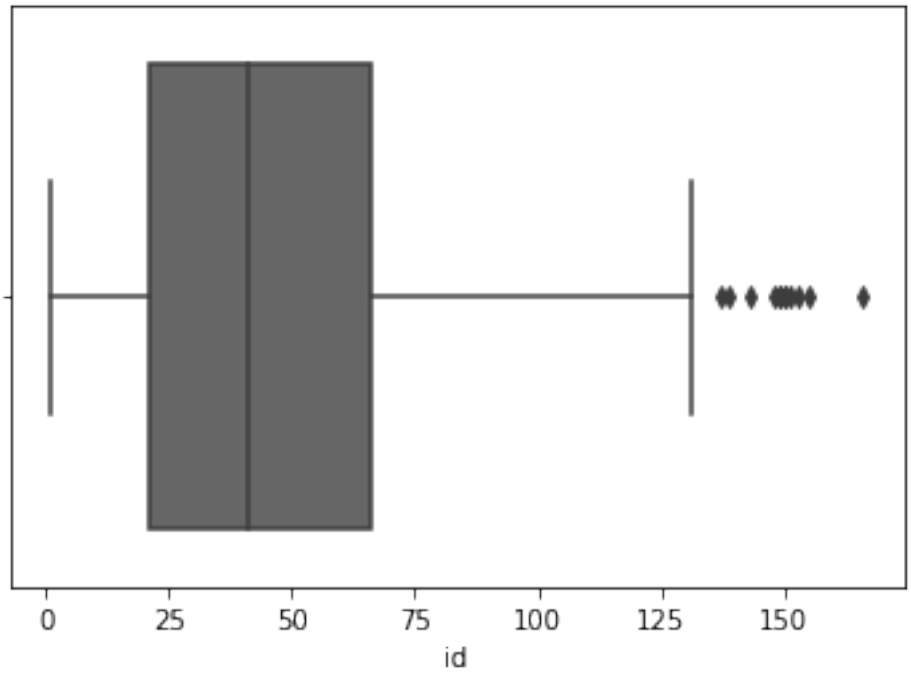


[47]:
```
print("Diagrama de caja del número mensual de mensajes de usuarios del plan␣
  ↪'ultimate'")
sns.boxplot(mess_month_ultimate2, color='0.4')
```

Diagrama de caja del número mensual de mensajes de usuarios del plan 'ultimate'

```
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

[47]: <AxesSubplot:xlabel='id'>



Los usuarios del plan Ultimate envían mensualmente un número de mensajes SMS en promedio 15% mayor a los del plan Surf.

Los histogramas del número de mensajes mensuales que han necesitado los usuarios muestran distribuciones asimétricas, aparentemente positivas, pero en donde las medias no son mayores a las medianas, pues las colas son relativamente livianas.

En el gráfico de bigotes se puede notar que más de un 25% de usuarios del plan Surf se exceden del límite de mensajes mensuales de su plan; en casos atípicos, hasta cuatro veces más de los 50 mensajes asignados en el plan. Por otro lado, no hay registros de usuarios del plan Ultimate excediéndose de su límite de 1000 mensajes al mes.

1.13.3 Internet

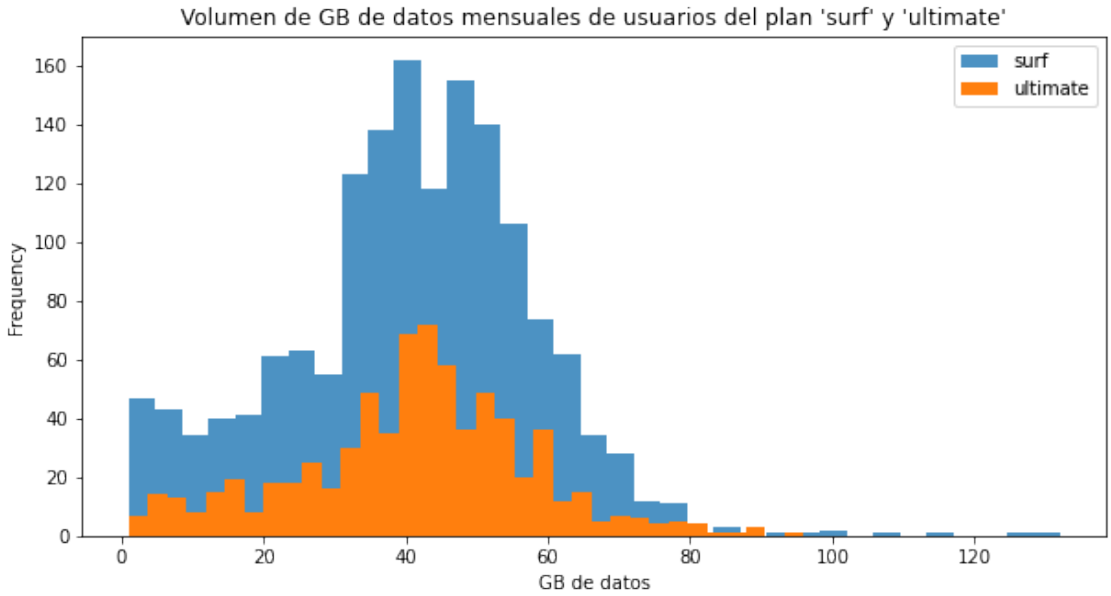
[48]:
```
# Compara la cantidad de tráfico de Internet consumido por usuarios por plan
int_month_list=df_internet.merge(df_users[['user_id',␣
 ↪'plan']],on='user_id',how='left')
int_month_surf1=int_month_list.query("plan=='surf'")
int_month_surf2=int_month_surf1.groupby(['user_id','session_month'])['gb_used'].
 ↪sum()
print(int_month_surf2,'\n\n')

int_month_ultimate1=int_month_list.query("plan=='ultimate'")
int_month_ultimate2=int_month_ultimate1.
 ↪groupby(['user_id','session_month'])['gb_used'].sum()
print(int_month_ultimate2)
```

```
user_id  session_month
1001     8                21.0
         9                41.0
         10               50.0
         11               49.0
         12               55.0
                          …
1498     12               56.0
1499     9                37.0
         10               52.0
         11               43.0
         12               58.0
Name: gb_used, Length: 1558, dtype: float64
```

```
user_id  session_month
1000     12                4.0
1006     11                7.0
         12               62.0
1008     10               48.0
         11               56.0
                          …
1493     9                40.0
         10               44.0
         11               41.0
         12               36.0
1497     12               28.0
Name: gb_used, Length: 719, dtype: float64
```

[49]:
```
# Comparación del volumen de datos mensuales que han usado los usuarios de cada␣
 ↪plan.
int_month_surf2.plot(kind='hist', title="Volumen de GB de datos mensuales de␣
 ↪usuarios del plan 'surf' y 'ultimate'", figsize=(10,5), alpha=0.8, bins=35)
int_month_ultimate2.plot(kind='hist', figsize=(10,5), bins=35).set_xlabel('GB␣
 ↪de datos')
plt.legend(['surf', 'ultimate'])
```

[49]:
```
<matplotlib.legend.Legend at 0x7f8e0385a490>
```



[50]:
```
# Calcula la media y la varianza del volumen de datos usados mensualmente.
print("La media de GB de datos usados por usuarios mensualmente del plan 'surf'␣
 ↪es:\n", int_month_surf2.mean(), "\n")
print("La varianza de GB de datos usados por usuarios del plan 'surf' es:\n",␣
 ↪np.var(int_month_surf2), "\n\n")

print("La media de GB de datos usados por usuarios mensualmente del plan␣
 ↪'ultimate' es:\n", int_month_ultimate2.mean(), "\n")
print("La varianza de GB de datos usados por usuarios del plan 'ultimate' es:
 ↪\n", np.var(int_month_ultimate2), "\n")
```

```
La media de GB de datos usados por usuarios mensualmente del plan 'surf' es:
 40.526957637997434

La varianza de GB de datos usados por usuarios del plan 'surf' es:
 305.5304029391554


La media de GB de datos usados por usuarios mensualmente del plan 'ultimate' es:
 41.13769123783032

La varianza de GB de datos usados por usuarios del plan 'ultimate' es:
 270.8475223469469
```

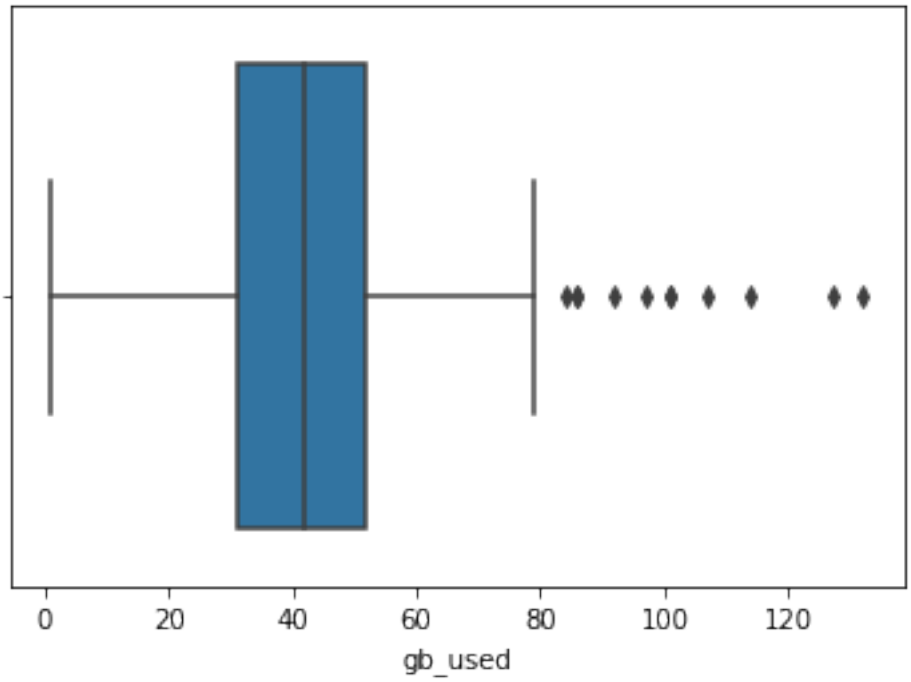
[51]:
```
print("Diagrama de caja del volumen mensual de GB de datos usados por usuarios␣
 ↪del plan 'surf'")
```

```
sns.boxplot(int_month_surf2)
```

Diagrama de caja del volumen mensual de GB de datos usados por usuarios del plan 'surf'

```
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

[51]: <AxesSubplot:xlabel='gb_used'>

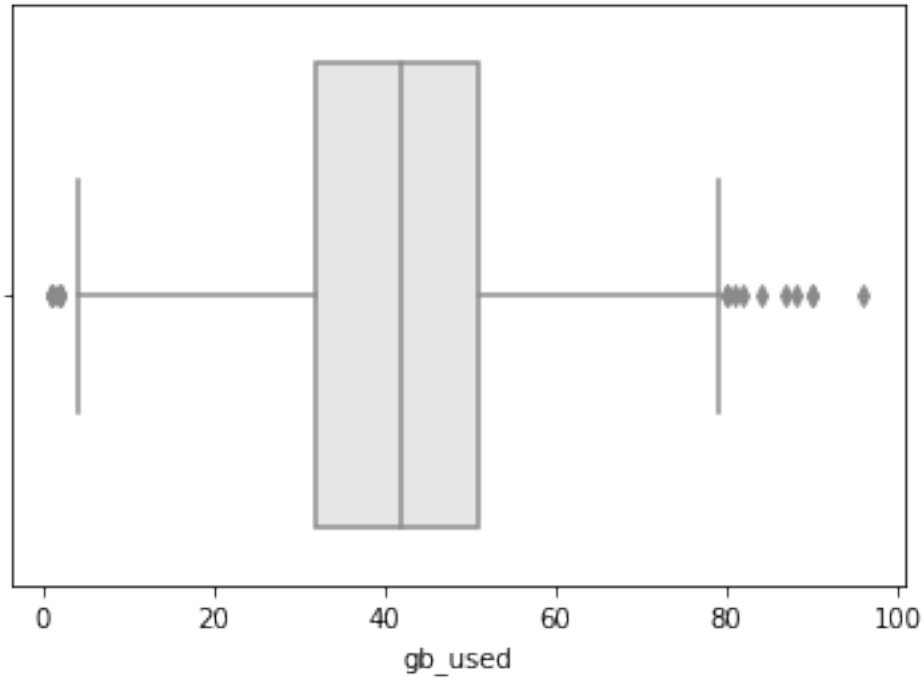


[52]:
```
print("Diagrama de caja del volumen mensual de GB de datos usados por usuarios␣
  ↪del plan 'ultimate'")
sns.boxplot(int_month_ultimate2, color='0.9')
```

Diagrama de caja del volumen mensual de GB de datos usados por usuarios del plan 'ultimate'

```
/opt/conda/lib/python3.9/site-packages/seaborn/_decorators.py:36: FutureWarning:
Pass the following variable as a keyword arg: x. From version 0.12, the only
valid positional argument will be `data`, and passing other arguments without an
explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

[52]: <AxesSubplot:xlabel='gb_used'>



Los usuarios del plan Surf y Ultimate usan en promedio un volumen similar de datos de Internet (alrededor de 40 GB).

Sin embargo, es notable que los usuarios de ambos planes se exceden dramáticamente del límite de datos ofrecido por sus respectivos planes. En el caso de usuarios del plan Surf, la gran mayoría, más de las tres cuartas partes, usan más de 15 GB al mes; asimismo la gran mayoría de usuarios del plan Ultimate se sobrepasan también de su límite de 30 GB. Hay usuarios del plan Surf que han usado más de 120 GB en un mes, mientras que los usuarios atípicos del plan Ultimate no se han excedido de 100 GB.

1.14 Ingreso

[53]:
```
# EB: Comparo los ingresos totales por cada plan y por cada mes. Trazo un␣
 ↪gráfico.
# EB: Preparamos los datos filtrando y desapilando las tablas del paso 2.

tarif_list=merg_tarif[['user_id','1_tar', '2_tar', '3_tar', '4_tar', '5_tar',␣
 ↪'6_tar', '7_tar', '8_tar', '9_tar', '10_tar', '11_tar', '12_tar']]
tarif_list.rename(columns={'1_tar': '1', '2_tar': '2','3_tar': '3', '4_tar':␣
 ↪'4','5_tar': '5', '6_tar': '6','7_tar': '7', '8_tar': '8','9_tar': '9',␣
 ↪'10_tar': '10','11_tar': '11', '12_tar': '12'}, inplace=True)
tarif_list.set_index(["user_id"], inplace = True, drop =True)
unp_tarif_list=tarif_list.unstack().reset_index(name='tarif')
```

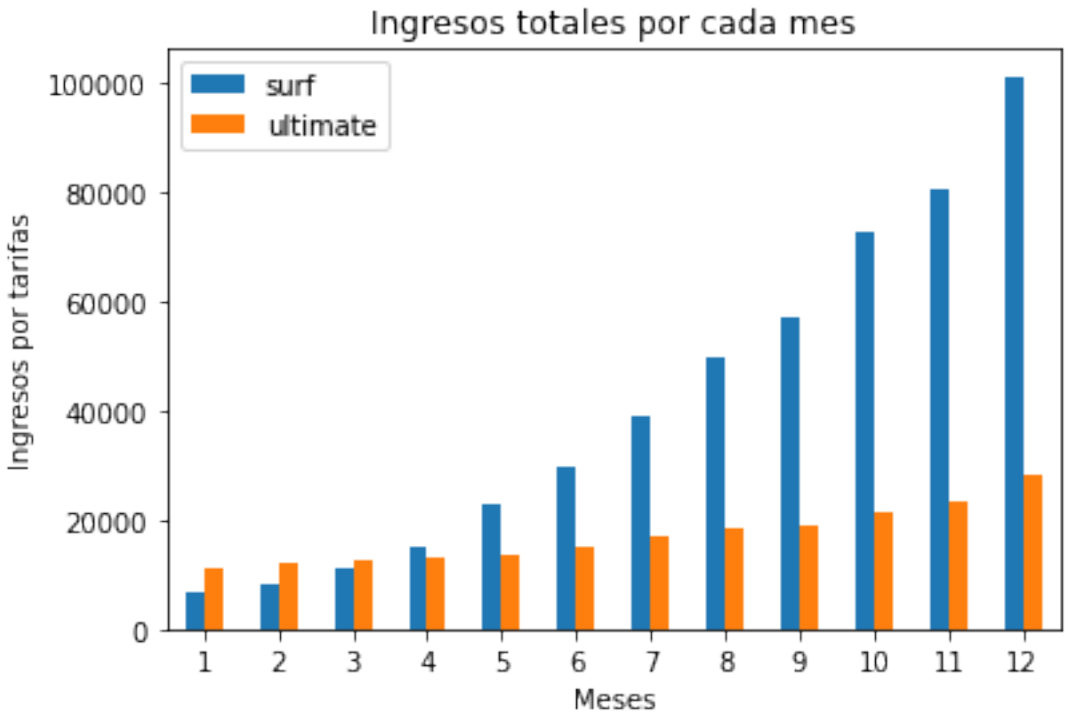
```
unp_tarif_list.rename(columns={'level_0': 'month'}, inplace=True)
all_tarif_list=unp_tarif_list.merge(df_users[['user_id','plan']], on='user_id',␣
 ↪how='left')

# EB: Necesito resetear el índice y ordenar los meses.
surf_tarif=all_tarif_list.query("plan=='surf'")
grp_surf_tarif=surf_tarif.groupby(['month'])['tarif'].sum()
grp_surf_tarif=grp_surf_tarif.iloc[grp_surf_tarif.index.astype(int).argsort()]
grp_surf_tarif=grp_surf_tarif.reset_index()
grp_surf_tarif.rename(columns={'tarif': 'surf'}, inplace=True)
print(grp_surf_tarif,'\n\n')

ultimate_tarif=all_tarif_list.query("plan=='ultimate'")
grp_ultimate_tarif=ultimate_tarif.groupby(['month'])['tarif'].sum()
grp_ultimate_tarif=grp_ultimate_tarif.iloc[grp_ultimate_tarif.index.astype(int).
 ↪argsort()]
grp_ultimate_tarif=grp_ultimate_tarif.reset_index()
grp_ultimate_tarif.rename(columns={'tarif': 'ultimate'}, inplace=True)
print(grp_ultimate_tarif,'\n\n')
```

```
   month       surf
0      1    6840.00
1      2    8289.57
2      3   11076.65
3      4   14875.53
4      5   22627.37
5      6   29709.80
6      7   38873.56
7      8   49786.85
8      9   57274.67
9     10   72844.83
10    11   80295.21
11    12  101177.14


   month  ultimate
0      1   11270.0
1      2   11928.0
2      3   12698.0
3      4   13188.0
4      5   13783.0
5      6   14819.0
6      7   16772.0
7      8   18697.0
8      9   18823.0
9     10   21259.0
10    11   23212.0
```

```
11    12   28364.0
```

```
/opt/conda/lib/python3.9/site-packages/pandas/core/frame.py:4441:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().rename(
```

[54]:
```
# EB: Gráfico de barras de los ingresos totales de cada mes.
plotdata2=grp_surf_tarif.merge(grp_ultimate_tarif, on='month')
plotdata2.plot(kind='bar', title= 'Ingresos totales por cada mes', x='month',␣
  ↪xlabel='Meses', ylabel='Ingresos por tarifas', rot=0)

plt.show()
```

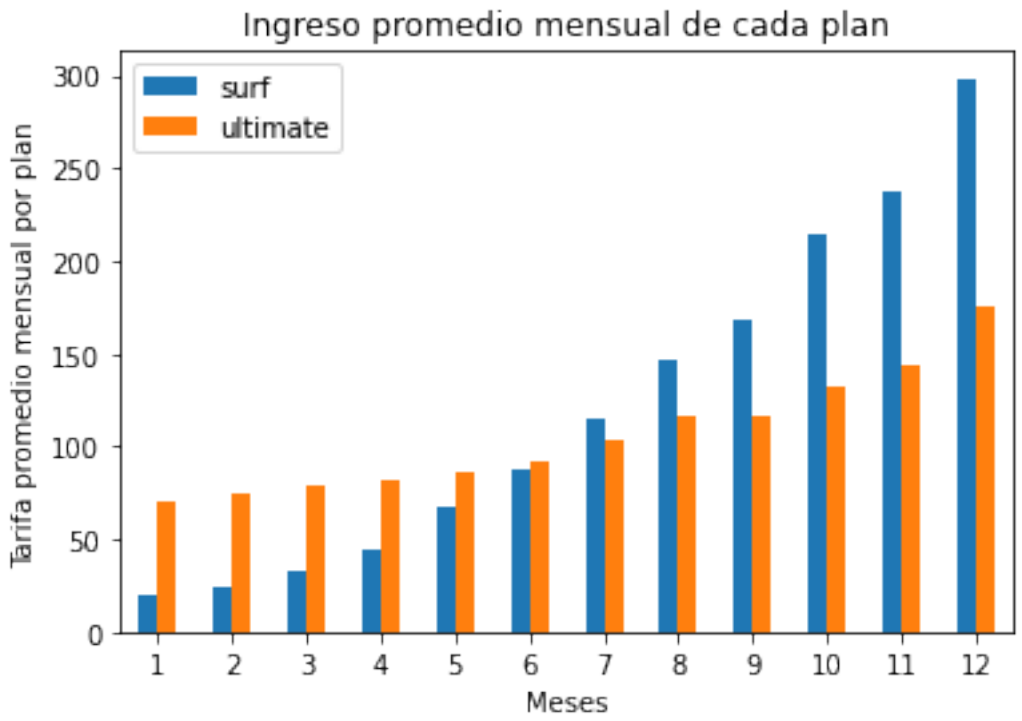


[55]:
```
# EB: Cálculo del ingreso promedio mensual de cada plan
grp_surf_tarif2=surf_tarif.groupby(['month'])['tarif'].mean()
grp_surf_tarif2=grp_surf_tarif2.iloc[grp_surf_tarif2.index.astype(int).
  ↪argsort()]
grp_surf_tarif2.name='surf'
```

```
grp_surf_tarif2=grp_surf_tarif2.reset_index()
print(grp_surf_tarif2,'\n\n')

grp_ultimate_tarif2=ultimate_tarif.groupby(['month'])['tarif'].mean()
grp_ultimate_tarif2=grp_ultimate_tarif2.iloc[grp_ultimate_tarif2.index.
 ↪astype(int).argsort()]
grp_ultimate_tarif2.name='ultimate'
grp_ultimate_tarif2=grp_ultimate_tarif2.reset_index()
print(grp_ultimate_tarif2,'\n\n')
```

```
   month        surf
0      1   20.176991
1      2   24.453009
2      3   32.674484
3      4   43.880619
4      5   66.747404
5      6   87.639528
6      7  114.671268
7      8  146.863864
8      9  168.951829
9     10  214.881504
10    11  236.859027
11    12  298.457640


   month    ultimate
0      1   70.000000
1      2   74.086957
2      3   78.869565
3      4   81.913043
4      5   85.608696
5      6   92.043478
6      7  104.173913
7      8  116.130435
8      9  116.913043
9     10  132.043478
10    11  144.173913
11    12  176.173913
```

[56]:
```
plotdata3=grp_surf_tarif2.merge(grp_ultimate_tarif2, on='month')
plotdata3.plot(kind='bar', title='Ingreso promedio mensual de cada plan',␣
 ↪x='month', legend=True, xlabel='Meses', ylabel='Tarifa promedio mensual por␣
 ↪plan', rot=0)

plt.show()
```



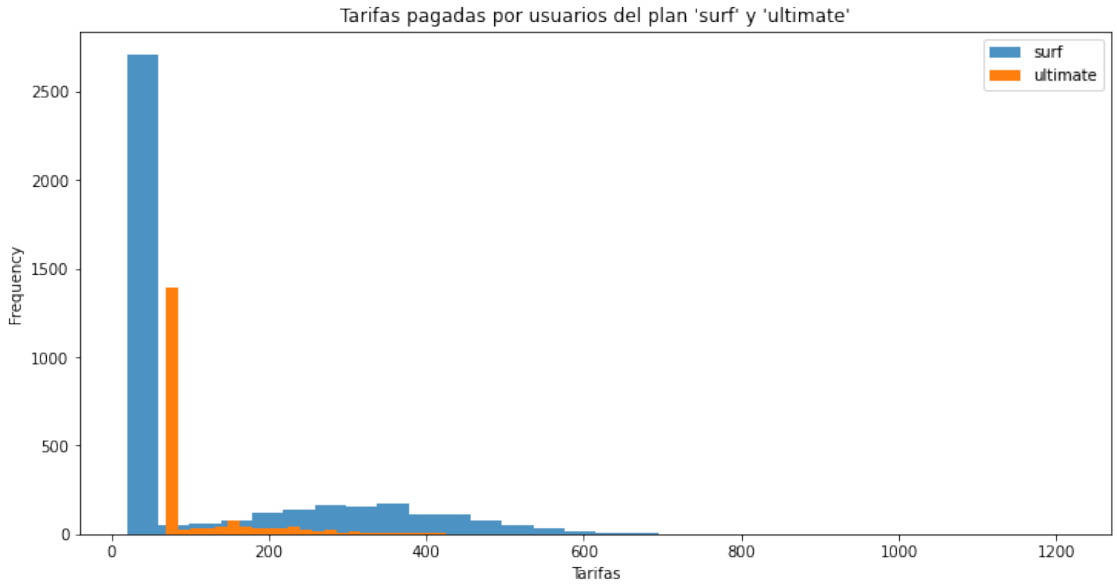
[62]:
```
grp_surf_tarif3=surf_tarif.groupby(['user_id'])['tarif'].mean()
grp_ultimate_tarif3=ultimate_tarif.groupby(['user_id'])['tarif'].mean()


print("La media mensual del promedio de tarifas pagadas por los usuarios del␣
 ↪plan 'surf' es:\n", grp_surf_tarif3.mean(), "\n")
print("La varianza mensual del promedio de tarifas pagadas por los usuarios del␣
 ↪plan 'surf' es:\n", np.var(grp_surf_tarif3), "\n\n")

print("La media mensual del promedio de tarifas pagadas por los usuarios del␣
 ↪plan 'ultimate' es:\n", grp_ultimate_tarif3.mean(), "\n")
print("La varianza mensual del promedio de tarifas pagadas por los usuarios del␣
 ↪plan 'ultimate' es:\n", np.var(grp_ultimate_tarif3), "\n")
```

```
La media mensual del promedio de tarifas pagadas por los usuarios del plan
'surf' es:
 121.3547640117994

La varianza mensual del promedio de tarifas pagadas por los usuarios del plan
'surf' es:
 7242.449264562468


La media mensual del promedio de tarifas pagadas por los usuarios del plan
```

```
'ultimate' es:
 106.01086956521739

La varianza mensual del promedio de tarifas pagadas por los usuarios del plan
'ultimate' es:
 1671.9760291955472
```

[58]:
```
surf_tarif['tarif'].plot(kind='hist', title="Tarifas pagadas por usuarios del␣
 ↪plan 'surf' y 'ultimate'", figsize=(12,6), alpha=0.8, bins=30)
ultimate_tarif['tarif'].plot(kind='hist', figsize=(12,6), bins=30).
 ↪set_xlabel('Tarifas')
plt.legend(['surf', 'ultimate'])
```

[58]:
```
<matplotlib.legend.Legend at 0x7f8e02d6aa60>
```



Los ingresos mensuales de ambos planes tienden a aumentar con el transcurso del año. Los ingresos totales del plan Surf superan a los Ultimate a partir del cuarto mes y de forma bastante marcada. Incluso promediando el ingreso mensual por usuario el promedio de tarifas pagadas de Surf supera al de Ultimate desde el mes séptimo. En total la media mensual pagada por los usarios de Surf es 20% mayor que la de Ultimate.

El usuario promedio del plan Surf está pagando más que el usuario promedio del plan Ultimate. Más aun, la gran mayoría de usuarios del plan Surf eventualmente llegan a pagar más de lo que pagarían con el otro plan.

Comentario revisor

Muy buena prática la de usar distintos tipos de gráficas identificar algunos hallazgos y llegar a conclusiones

1.15 Prueba las hipótesis estadísticas

Para probar la hipótesis de que la diferencia entre los ingresos promedio procedentes de los usuarios de los planes Surf y Ultimate es estadísticamente significativa uso la función ttest_ind de la librería scipy. En base a lo analizado previamente escojo un valor alfa de 0.01 para asegurar que no haya más de un 1% de probabilidades que las diferencias entre ambas poblaciones se puedan deber al azar. Asimismo se ha constatado en los análisis de datos que las varianzas entre las poblaciones son muy disímiles(mayor a una relación de 4:1), por tanto el parámetro equal_var se expresa como False.

La hipótesis nula es la que refuta la relación propuesta en la hipótesis de investigación.

Las hipótesis son las siguientes:

Hipótesis nula(H0): Los ingreso promedio procedentes de los usuarios de los planes de llamada Surf y Ultimate son iguales estadísticamente.

Hipótesis altenrativa(H1): Los ingreso promedio procedentes de los usuarios de los planes de llamada Surf y Ultimate son diferentes estadísticamente.

[59]:
```
# Prueba las hipótesis
surf_tarif_fil=surf_tarif['tarif']
ultimate_tarif_fil=ultimate_tarif['tarif']
alpha=0.01
results=st.ttest_ind(surf_tarif_fil, ultimate_tarif_fil, equal_var=False)
print('Valor p:', results.pvalue)

if(results.pvalue<alpha):
    print('Rechazamos la H0')
else:
    print('No podemos rechazar la H0')
```

```
Valor p: 3.9757718232846046e-07
Rechazamos la H0
```

Rechazamos la hipótesis nula y comprobamos que las medias de las tarifas de ambos planes son estadísticamente diferentes.

Para probar la hipótesis de que el ingreso promedio de los usuarios del área NY-NJ es diferente al de los usuarios de otras regiones uso la función ttest_ind de la librería scipy.

[60]:
```
# EB: Filtramos las filas por ciudad.
all_tarif_list_city=all_tarif_list.merge(df_users[['user_id','city']],␣
 ↪on='user_id', how='left')
tarif_NYNJ=all_tarif_list_city[all_tarif_list_city['city'].str.
 ↪contains("NY-NJ")]['tarif']
tarif_non_NYNJ=all_tarif_list_city[~all_tarif_list_city['city'].str.
 ↪contains("NY-NJ")]['tarif']
# EB: Comprobamos las varianzas
print(np.var(tarif_NYNJ))
print(np.var(tarif_non_NYNJ))
```

```
21331.597506863603
19366.593058269525
```

En base a lo analizado previamente escojo conservadoramente para esta muestra un valor alfa de 0.05 para asegurar que no haya más de un 5% de probabilidades que las diferencias entre ambas poblaciones se puedan deber al azar. Asimismo, como se ha constatado previamente que las varianzas entre las poblaciones no son muy disímiles, el parámetro equal_var se expresa como True.

Las hipótesis son las siguientes:

Hipótesis nula(H0): El ingreso promedio procedentes de los usuarios del área NY-NJ no es diferente estadísticamente al de los usuarios de otras regiones.

Hipótesis altenrativa(H1): El ingreso promedio procedentes de los usuarios del área NY-NJ es estadísticamente diferente al de los usuarios de otras regiones.

[61]:
```
# Prueba las hipótesis
alpha2=0.05
results2=st.ttest_ind(tarif_NYNJ, tarif_non_NYNJ, equal_var=True)
print('Valor p:', results2.pvalue)

if(results2.pvalue<alpha2):
    print('Rechazamos la H0')
else:
    print('No podemos rechazar la H0')
```

```
Valor p: 0.02768433038975269
Rechazamos la H0
```

Comentario revisor

Muy buen trabajo con las pruebas de hipótesis

Rechazamos la hipótesis nula. Según el análisis, para esta muestra solo hay un 2.7% de probabilidades de que la diferencia estadística entre el promedio de ingreso de los usuarios de NY-NJ y los de otras regiones solo sea una cuestión de azar.

1.16 Conclusión general

El preprocesamiento y arreglo de los datos para su presentación y análisis posterior ha sido lo más trabajoso del proyecto. Si bien hay varias formas de resolver los cálculos, no todas las formas de agrupar los datos han facilitado el análisis. En el paso 2 el agrupamiento de datos por tabla dinámica fue el más adecuado para los cálculos de las tarifas por mes. Para los cálculos del paso 3 de análisis de datos fue mucho más conveniente manejar las tablas desapiladas y usar datos agrupados.

El uso de una función en conjunto con un bucle facilitó el proceso de cáclulo de tarifas para el paso 2. Para el análisis de datos de tarifas se requirió desapilar y filtrar la última tablas dinámica fusionada, lo que requirió de varios pasos y consideraciones.

Con respecto a los planes de Megaline: los usuarios del paquete Surf son los que más aportan ingresos por tarifas. Los hábitos de consumo no difieren tanto entre los usuarios de ambos planes, sin embargo, dadas las prestaciones y condiciones del plan Surf, sus usuarios tienden a excederse

más en el uso de llamadas, mensajes y particularmente de datos, ergo pagando tarifas mayores a las establecidas por su pago mensual; incluso llegando a pagar mensualmente en promedio un 20% más que el usuario de Ultimate. Los usuarios de Ultimate, por otro lado, subutilizan el saldo de minutos de llamadas y de mensajes, pero no el de datos, que también exceden, pero con la ventaja de pagar menos por superar los límites del paquete.

Desde el punto de vista del operador Megaline, le conviene aumentar el presupuesto de publicidad para el plan Surf, haciendo especial énfasis en el consumo de datos de Internet. Por otro lado, ciertamente al usuario promedio de Surf, a la gran mayoría de ellos más bien, le convendría contratar el plan Ultimate. Eso considerando que basta que este usuario sobrepase 5 GB del paquete de datos pagar lo mismo que la tarifa base del paquete Ultimate, y los datos demuestran que la mayoría de usuarios se excede en tanto más que esa cantidad, por no mencionar los excesos en llamadas y mensajes.

Comentario revisor

En general creo que hiciste un muy buen trabajo con el proyecto, pudiste limpiar y trabajar las bases de datos de beuna manera, así como juntar la información. Además, considero que el análisis con las gráficas y con las pruebas de hipótesis es muy acertado. No obstante, recuerda que siempre podemos mejorar y te menciono algunos puntos que debes considerar:

- Realizar un análisis inicial de registros duplicados en todas las bases de datos
- verificar que redondeamos las variables antes de hacer la agrupación por usuario
- Cuando hacemos la agrupación de los datos, considerar agregar la variable de mes para que los análisis posteriores sean más sencillos e intuitivos de realizar

Gracias. Siempre hice un anñalisis de registros duplicados y desde el paso 1 se redondearon las variables de datos y llamadas con la función np.ceil.