Portafolio - Proyecto 2

April 20, 2025

1 Revenue Analysis of Prepaid Plans – Megaline Telecom

Victor Uriel Leyva
Analista de Datos Jr.

1.0.1 Objetivo del Proyecto:

Analizar el comportamiento de clientes de una empresa de telecomunicaciones ficticia (Megaline) con el fin de identificar cuál de sus dos planes prepago —Surf o Ultimate— genera mayores ingresos, para guiar decisiones estratégicas de publicidad.

1.0.2 Herramientas utilizadas:

- Python
- pandas
- matplotlib
- seaborn
- scipy.stats
- numpy
- math

1.0.3 Principales habilidades aplicadas:

- Limpieza y transformación de datos
- Análisis comparativo por grupo (plan y región)
- Pruebas de hipótesis estadísticas
- Visualización de patrones de consumo
- Interpretación de ingresos por cliente

1.1 Introducción

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.

1.2 Inicialización

[79]:
```
# Cargar todas las librerías
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from scipy import stats as st
import seaborn as sns
import math as mt
```

1.3 Cargar datos

[80]:
```
# Carga los archivos de datos en diferentes DataFrames
df_calls = pd.read_csv('megaline_calls.csv')
df_internet = pd.read_csv('megaline_internet.csv')
df_messages = pd.read_csv('megaline_messages.csv')
df_plans = pd.read_csv('megaline_plans.csv')
df_users = pd.read_csv('megaline_users.csv')
```

1.4 Preparar los datos

1.5 Tarifas

[81]:
```
# Imprime la información general/resumida sobre el DataFrame de las tarifas

df_plans.info()
```

```
<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
```

[82]:
```
# Imprime una muestra de los datos para las tarifas

print(df_plans.head())
```

```
   messages_included  mb_per_month_included  minutes_included  \
0                 50                  15360               500
1               1000                  30720              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
```

Al examinar la información general y la tabla de nuestro dataframe de tarifas, podemos afirmar que nuestros datos sobre los tipos de planes tienen un tipo de datos adecuado, no existen valores ausentes y los datos corresponden a la información anteriormente proporcionada.

1.6 Corregir datos

No considero que existan errores en los datos a corregir

1.7 Enriquecer los datos

Se creará una columna con la cantidad de datos incluidos al mes en Gygabytes.

[83]:
```
df_plans['gb_per_month_included'] = df_plans['mb_per_month_included'] / 1024
df_plans
```

[83]:
```
   messages_included  mb_per_month_included  minutes_included  \
0                 50                  15360               500
1               1000                  30720              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

   gb_per_month_included
0                   15.0
1                   30.0
```

1.8 Usuarios/as

[84]:
```
# Imprime la información general/resumida sobre el DataFrame de usuarios

df_users.info()
```

```
<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
```

[85]:
```
# Imprime una muestra de datos para usuarios

print(df_users.sample(10))
```

```
     user_id first_name last_name  age  \
285     1285     Joesph     Barry   28
243     1243     Isaias   Compton   71
191     1191       Zack    Waters   75
101     1101       Sage    Conley   27
185     1185      Loria   Freeman   47
160     1160     Steven    Morgan   44
223     1223   Patience    Wilson   37
304     1304   Dominque      Cole   69
50      1050       Jone      Owen   23
427     1427      Zofia     Brock   64

                                              city    reg_date      plan  \
285            Los Angeles-Long Beach-Anaheim, CA MSA  2018-08-03      surf
243                     Phoenix-Mesa-Chandler, AZ MSA  2018-05-11      surf
191                              St. Louis, MO-IL MSA  2018-01-21  ultimate
101  Washington-Arlington-Alexandria, DC-VA-MD-WV MSA  2018-02-08      surf
185            Louisville/Jefferson County, KY-IN MSA  2018-01-14  ultimate
160           Portland-Vancouver-Hillsboro, OR-WA MSA  2018-02-05      surf
223                     Phoenix-Mesa-Chandler, AZ MSA  2018-07-05      surf
304                         Birmingham-Hoover, AL MSA  2018-10-28  ultimate
50        Miami-Fort Lauderdale-West Palm Beach, FL MSA  2018-03-20  ultimate
427  Washington-Arlington-Alexandria, DC-VA-MD-WV MSA  2018-01-26  ultimate

     churn_date
285         NaN
243         NaN
191  2018-11-30
101         NaN
185         NaN
160         NaN
223         NaN
```

```
304         NaN
50   2018-10-07
427         NaN
```

Después de examinar la información general y obtener una muestra de diez usuarios de nuestro dataframe de usuarios, se llegó a la conclusión de que se pueden abordar los datos con otro tipo de datos. Por ejemplo para nuestras columnas de user_id, será más conveniente tratarlos como tipo de datos `object` y no tanto como `int64` ya que son valores que no se modificarán. También considero que nos será más útil si la columna de fecha de suscripción la manejamos como tipo de datos `datetime64`. Así como remplazar los valores ausentes en la columna `churn_date` por la cadena `'active'`, indicando que los usuarios seguían usando el servicio cuando se extrajeron las muestras.

1.8.1 Corregir los datos

Se remplazarán los valores ausentes de la columna `churn_date` por la leyenda "active". Y comprobaremos que ya no contamos con valores ausentes en nuestro dataframe.

[86]:
```
df_users['churn_date'] = df_users['churn_date'].fillna('active')
df_users.info()
```

```
<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  500 non-null    object
dtypes: int64(2), object(6)
memory usage: 31.4+ KB
```

1.8.2 Enriquecer los datos

Cambiaremos el tipo de datos de la columna `user_id` de int64 a object (string), ya que esta cadena numérica no se modificará. Así como la columna de fecha de suscripción, que se cambiara el tipo de datos de object a datetime64.

[87]:
```
df_users['user_id'] = df_users['user_id'].astype('object')
df_users['reg_date'] = pd.to_datetime(df_users['reg_date'])
df_users.info()
```

```
<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    object
 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(1), object(6)
memory usage: 31.4+ KB
```

1.9 Llamadas

[88]:
```
# Imprime la información general/resumida sobre el DataFrame de las llamadas

df_calls.info()
```

```
<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
```

[89]:
```
# Imprime una muestra de datos para las llamadas

print(df_calls.sample(10))
```

```
              id  user_id   call_date  duration
36551   1138_515     1138  2018-10-12     24.14
58355   1210_612     1210  2018-08-27     10.24
17410   1066_520     1066  2018-10-25      7.49
124363  1439_231     1439  2018-07-01     10.91
31901   1120_284     1120  2018-09-25      7.05
101331   1362_69     1362  2018-02-14      3.79
110453  1389_548     1389  2018-09-06     12.16
79599    1285_18     1285  2018-12-05     10.51
90684   1328_314     1328  2018-10-19      1.39
36610   1138_575     1138  2018-09-30      0.62
```

Para el dataframe de llamadas, al igual que con el dataframe de usuarios, vamos a mantener los valores de los ID de los clientes como una cadena para evitar que puedan modificarse. De ahí en

más nuestros datos se visualizan bien.

1.9.1 Corregir los datos

Vamos a convertir el tipo de datos `int64` a tipo `object` de la columna ID's de los usuarios. Así como la columna de fecha de llamada pasará a tener un tipo de datos `Datetime64` al aplicarle la función to_datetime().

[90]:
```
df_calls['user_id'] = df_calls['user_id'].astype('object')
df_calls['call_date'] = pd.to_datetime(df_calls['call_date'])
df_calls.info()
```

```
<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  object
 2   call_date  137735 non-null  datetime64[ns]
 3   duration   137735 non-null  float64
dtypes: datetime64[ns](1), float64(1), object(2)
memory usage: 4.2+ MB
```

1.9.2 Enriquecer los datos

Se creará una columna con los meses en los que se realizaron llamadas.

[91]:
```
df_calls['month'] = df_calls['call_date'].dt.month
print(df_calls)
```

```
              id user_id  call_date  duration  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
…            …      …          …        …      …
137730  1499_199    1499 2018-11-21      8.72     11
137731  1499_200    1499 2018-10-20     10.89     10
137732  1499_201    1499 2018-09-21      8.12      9
137733  1499_202    1499 2018-10-10      0.37     10
137734  1499_203    1499 2018-12-29     13.86     12

[137735 rows x 5 columns]
```

1.10 Mensajes

[92]:
```
# 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
```

[93]:
```
# Imprime una muestra de datos para los mensajes

print(df_messages.sample(10))
```

```
             id  user_id message_date
64269  1408_394     1408   2018-02-08
47701  1324_418     1324   2018-05-05
41231  1264_374     1264   2018-11-06
13214  1088_167     1088   2018-08-30
3421   1036_256     1036   2018-10-09
67843  1439_237     1439   2018-07-03
37700  1249_236     1249   2018-10-07
33336   1211_12     1211   2018-09-17
28287   1174_77     1174   2018-08-20
36736   1246_40     1246   2018-02-20
```

Los datos son correctos, no existen valores ausentes y las columnas coinciden con la información proporcionada. Solo vamos a cambiar el tipo de dato adecuado en la columna `user_id` y `message_date`.

1.10.1 Corregir los datos

Mismo caso que en el dataframe de llamadas, cambiaremos el tipo de datos en la columna id de usuario a una cadena; y la columna de fecha de SMS pasará del tipo `object` a `Datetime64`.

[94]:
```
df_messages['user_id'] = df_messages['user_id'].astype('object')
df_messages['message_date'] = pd.to_datetime(df_messages['message_date'])
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  object
 2   message_date  76051 non-null  datetime64[ns]
dtypes: datetime64[ns](1), object(2)
memory usage: 1.7+ MB
```

1.10.2 Enriquecer los datos

Se creará una columna con los meses en los que se realizaron mensajes.

[95]:
```
df_messages['month'] = df_messages['message_date'].dt.month
print(df_messages)
```

```
             id user_id message_date  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
…         …     …           …    …
76046  1497_526    1497   2018-12-24     12
76047  1497_536    1497   2018-12-24     12
76048  1497_547    1497   2018-12-31     12
76049  1497_558    1497   2018-12-24     12
76050  1497_613    1497   2018-12-23     12

[76051 rows x 4 columns]
```

1.11 Internet

[96]:
```
# 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
```

[97]:
```
# Imprime una muestra de datos para el tráfico de internet
```

```
print(df_internet.sample(10))
```

```
             id  user_id session_date  mb_used
54659   1247_74     1247   2018-10-01   848.89
77978  1363_159     1363   2018-09-08   618.96
83870   1391_78     1391   2018-12-17    75.66
22104  1100_308     1100   2018-10-30   206.21
52516  1236_383     1236   2018-10-12   494.36
5094   1028_673     1028   2018-11-21   336.97
85735  1399_347     1399   2018-11-02   430.47
54019   1245_52     1245   2018-10-28   378.10
97109   1458_42     1458   2018-08-24   245.62
23795   1110_53     1110   2018-08-19   561.81
```

Los datos son correctos, solamente cambiaremos el tipo de datos de las columnas de id de usuario y de fecha de la sesión web.

1.11.1 Corregir los datos

Haciendo uso de la funcion `astype()` cambiaremos el tipo de datos de la columna de id de clientes. Así como la columa de fecha de sesión web.

[98]:
```
df_internet['user_id'] = df_internet['user_id'].astype('object')
df_internet['session_date'] = pd.to_datetime(df_internet['session_date'])
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  object
 2   session_date  104825 non-null  datetime64[ns]
 3   mb_used       104825 non-null  float64
dtypes: datetime64[ns](1), float64(1), object(2)
memory usage: 3.2+ MB
```

1.11.2 Enriquecer los datos

Vamos a enriquecer nuestro dataframe de internet agregando una columna que muestre el volumen de datos gastados durante la sesión pero en Gigabytes, ya que es la unidad de medida que maneja la empresa Megaline. Así como una columna con los meses de la fecha de la sesión web.

[99]:
```
df_internet['gb_used'] = df_internet['mb_used'] / 1024
df_internet['month'] = df_internet['session_date'].dt.month
print(df_internet)
```

```
              id user_id session_date  mb_used   gb_used  month
0        1000_13    1000   2018-12-29    89.86  0.087754     12
```

```
1        1000_204    1000   2018-12-31     0.00  0.000000     12
2        1000_379    1000   2018-12-28   660.40  0.644922     12
3        1000_413    1000   2018-12-26   270.99  0.264639     12
4        1000_442    1000   2018-12-27   880.22  0.859590     12
…           …      …          …       …       …     …
104820   1499_215    1499   2018-10-20   218.06  0.212949     10
104821   1499_216    1499   2018-12-30   304.72  0.297578     12
104822   1499_217    1499   2018-09-22   292.75  0.285889      9
104823   1499_218    1499   2018-12-07     0.00  0.000000     12
104824   1499_219    1499   2018-12-24   758.31  0.740537     12

[104825 rows x 6 columns]
```

1.12 Estudiar las condiciones de las tarifas

[100]:
```
# Imprime las condiciones de la tarifa y asegúrate de que te quedan claras

df_plans
```

[100]:
```
   messages_included  mb_per_month_included  minutes_included  \
0                 50                  15360               500
1               1000                  30720              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

   gb_per_month_included
0                   15.0
1                   30.0
```

1.13 Agregar datos por usuario

[101]:
```
# Calcula el número de llamadas hechas por cada usuario al mes. Guarda el␣
↪resultado.

calls_per_month = df_calls.groupby(['user_id','month'])['id'].count()
calls_per_month = calls_per_month.reset_index()
calls_per_month.columns = ['user_id','month','number_of_calls']
print(calls_per_month)
```

```
      user_id  month  number_of_calls
0        1000     12               16
1        1001      8               27
2        1001      9               49
3        1001     10               65
4        1001     11               64
…         …      …                …
```

```
2253     1498     12                39
2254     1499      9                41
2255     1499     10                53
2256     1499     11                45
2257     1499     12                65

[2258 rows x 3 columns]
```

[102]:
```
# Calcula la cantidad de minutos usados por cada usuario al mes. Guarda el resultado.

minutes_per_month = df_calls.groupby(['user_id','month'])['duration'].sum().round().reset_index()
minutes_per_month.columns = ['user_id','month','minutes_used']
print(minutes_per_month)
```

```
      user_id  month  minutes_used
0        1000     12         117.0
1        1001      8         171.0
2        1001      9         298.0
3        1001     10         374.0
4        1001     11         405.0
…         …    …             …
2253     1498     12         325.0
2254     1499      9         330.0
2255     1499     10         363.0
2256     1499     11         289.0
2257     1499     12         468.0

[2258 rows x 3 columns]
```

[103]:
```
# Calcula el número de mensajes enviados por cada usuario al mes. Guarda el resultado.

messages_per_month = df_messages.groupby(['user_id','month'])['id'].count()
messages_per_month = messages_per_month.reset_index()
messages_per_month.columns = ['user_id','month','number_of_messages']
print(messages_per_month)
```

```
      user_id  month  number_of_messages
0        1000     12                  11
1        1001      8                  30
2        1001      9                  44
3        1001     10                  53
4        1001     11                  36
…         …    …                    …
1801     1496      9                  21
1802     1496     10                  18
```

```
1803     1496     11                  13
1804     1496     12                  11
1805     1497     12                  50

[1806 rows x 3 columns]
```

[104]:
```
# Calcula el volumen del tráfico de Internet usado por cada usuario al mes. Guarda el resultado.

internet_per_month = df_internet.groupby(['user_id','month'])['gb_used'].sum().round().reset_index()
internet_per_month.columns = ['user_id','month','internet_used']
print(internet_per_month)
```

```
      user_id  month  internet_used
0        1000     12            2.0
1        1001      8            7.0
2        1001      9           13.0
3        1001     10           22.0
4        1001     11           18.0
…         …    …              …
2272     1498     12           23.0
2273     1499      9           13.0
2274     1499     10           19.0
2275     1499     11           16.0
2276     1499     12           22.0

[2277 rows x 3 columns]
```

[105]:
```
# Fusiona los datos de llamadas, minutos, mensajes e Internet con base en user_id y month

consumption_per_user = calls_per_month.merge(minutes_per_month, on=['user_id','month'], how='outer')
consumption_per_user = consumption_per_user.merge(messages_per_month, on=['user_id','month'], how='outer')
consumption_per_user = consumption_per_user.merge(internet_per_month, on=['user_id','month'], how='outer')

# En las celdas donde tenemos valores ausentes vamos a remplazar los NaN por el valor 0
# Ya que la ausencia de valor nos indica que dicho usuario no utilizó llamadas, mensajes o internet
consumption_per_user = consumption_per_user.fillna(0)
display(consumption_per_user)
```

```
      user_id  month  number_of_calls  minutes_used  number_of_messages  \
0        1000     12             16.0         117.0                11.0
```

```
1        1001      8              27.0          171.0                30.0
2        1001      9              49.0          298.0                44.0
3        1001     10              65.0          374.0                53.0
4        1001     11              64.0          405.0                36.0
…         …      …               …             …                    …
2288     1349     12               0.0            0.0                61.0
2289     1361      5               0.0            0.0                 2.0
2290     1482     10               0.0            0.0                 2.0
2291     1108     12               0.0            0.0                 0.0
2292     1311      6               0.0            0.0                 0.0

      internet_used
0               2.0
1               7.0
2              13.0
3              22.0
4              18.0
…               …
2288           13.0
2289            1.0
2290            0.0
2291            0.0
2292            1.0

[2293 rows x 6 columns]
```

[106]:
```
# Añade la información de la tarifa

df_users_merged = df_users.rename(columns={'plan':'plan_name'})
df_users_merged = df_users_merged.merge(df_plans, on='plan_name', how='outer')
consumption_per_user = consumption_per_user.merge(df_users_merged, on=␣
 ↪'user_id', how='left')
display(consumption_per_user)
```

```
     user_id  month  number_of_calls  minutes_used  number_of_messages  \
0       1000     12             16.0         117.0                11.0
1       1001      8             27.0         171.0                30.0
2       1001      9             49.0         298.0                44.0
3       1001     10             65.0         374.0                53.0
4       1001     11             64.0         405.0                36.0
…        …      …               …             …                    …
2288    1349     12              0.0           0.0                61.0
2289    1361      5              0.0           0.0                 2.0
2290    1482     10              0.0           0.0                 2.0
2291    1108     12              0.0           0.0                 0.0
2292    1311      6              0.0           0.0                 0.0

      internet_used  first_name  last_name  age  \
```

```
0               2.0    Anamaria      Bauer   45
1               7.0      Mickey  Wilkerson   28
2              13.0      Mickey  Wilkerson   28
3              22.0      Mickey  Wilkerson   28
4              18.0      Mickey  Wilkerson   28
…               …         …         …  …
2288           13.0  Florentina       Diaz   69
2289            1.0     Jacelyn    Hoffman   45
2290            0.0      Armand      Glenn   70
2291            0.0    Porfirio       Kane   45
2292            1.0     Cherlyn   Saunders   69

                                        city  … plan_name churn_date  \
0         Atlanta-Sandy Springs-Roswell, GA MSA  …  ultimate     active
1               Seattle-Tacoma-Bellevue, WA MSA  …      surf     active
2               Seattle-Tacoma-Bellevue, WA MSA  …      surf     active
3               Seattle-Tacoma-Bellevue, WA MSA  …      surf     active
4               Seattle-Tacoma-Bellevue, WA MSA  …      surf     active
…                                            …  …       …        …
2288         Boston-Cambridge-Newton, MA-NH MSA  …      surf     active
2289                   Birmingham-Hoover, AL MSA  …      surf     active
2290  New York-Newark-Jersey City, NY-NJ-PA MSA  …  ultimate     active
2291                     Salt Lake City, UT MSA  …  ultimate     active
2292                      Memphis, TN-MS-AR MSA  …  ultimate     active

     messages_included  mb_per_month_included  minutes_included  \
0                 1000                  30720              3000
1                   50                  15360               500
2                   50                  15360               500
3                   50                  15360               500
4                   50                  15360               500
…                  …                      …                 …
2288                50                  15360               500
2289                50                  15360               500
2290              1000                  30720              3000
2291              1000                  30720              3000
2292              1000                  30720              3000

      usd_monthly_pay  usd_per_gb  usd_per_message  usd_per_minute  \
0                  70           7             0.01            0.01
1                  20          10             0.03            0.03
2                  20          10             0.03            0.03
3                  20          10             0.03            0.03
4                  20          10             0.03            0.03
…                 …          …              …               …
2288               20          10             0.03            0.03
2289               20          10             0.03            0.03
2290               70           7             0.01            0.01
```

```
2291                70           7              0.01            0.01
2292                70           7              0.01            0.01

      gb_per_month_included
0                      30.0
1                      15.0
2                      15.0
3                      15.0
4                      15.0
…                       …
2288                   15.0
2289                   15.0
2290                   30.0
2291                   30.0
2292                   30.0

[2293 rows x 21 columns]
```

[107]:
```
# Calcula el ingreso mensual para cada usuario

# Obtendremos el ingreso por tarifa de minutos excedidos
consumption_per_user['revenue_per_calls'] =␣
 ↪consumption_per_user['minutes_included'] -␣
 ↪consumption_per_user['minutes_used']
consumption_per_user['revenue_per_calls'] =␣
 ↪consumption_per_user['revenue_per_calls'].apply(lambda x:0 if x > 0 else x)
consumption_per_user['revenue_per_calls'] =␣
 ↪consumption_per_user['revenue_per_calls'] *␣
 ↪consumption_per_user['usd_per_minute'] * -1

# Obtendremos el ingreso por tarifa de mensajes excedidos
consumption_per_user['revenue_per_messages'] =␣
 ↪consumption_per_user['messages_included'] -␣
 ↪consumption_per_user['number_of_messages']
consumption_per_user['revenue_per_messages'] =␣
 ↪consumption_per_user['revenue_per_messages'].apply(lambda x:0 if x > 0 else␣
 ↪x)
consumption_per_user['revenue_per_messages'] =␣
 ↪consumption_per_user['revenue_per_messages'] *␣
 ↪consumption_per_user['usd_per_message'] * -1

# Obtendremos el ingreso por tarifa de gb excedidos
consumption_per_user['revenue_per_internet'] =␣
 ↪consumption_per_user['gb_per_month_included'] -␣
 ↪consumption_per_user['internet_used']
```

```
consumption_per_user['revenue_per_internet'] =␣
 ↪consumption_per_user['revenue_per_internet'].apply(lambda x:0 if x > 0 else␣
 ↪x)
consumption_per_user['revenue_per_internet'] =␣
 ↪consumption_per_user['revenue_per_internet'] *␣
 ↪consumption_per_user['usd_per_gb'] * -1

# Obtendremos el ingreso total
consumption_per_user['total_revenue'] = consumption_per_user['usd_monthly_pay']␣
 ↪+ consumption_per_user['revenue_per_calls'] +␣
 ↪consumption_per_user['revenue_per_messages'] +␣
 ↪consumption_per_user['revenue_per_internet']
display(consumption_per_user)
```

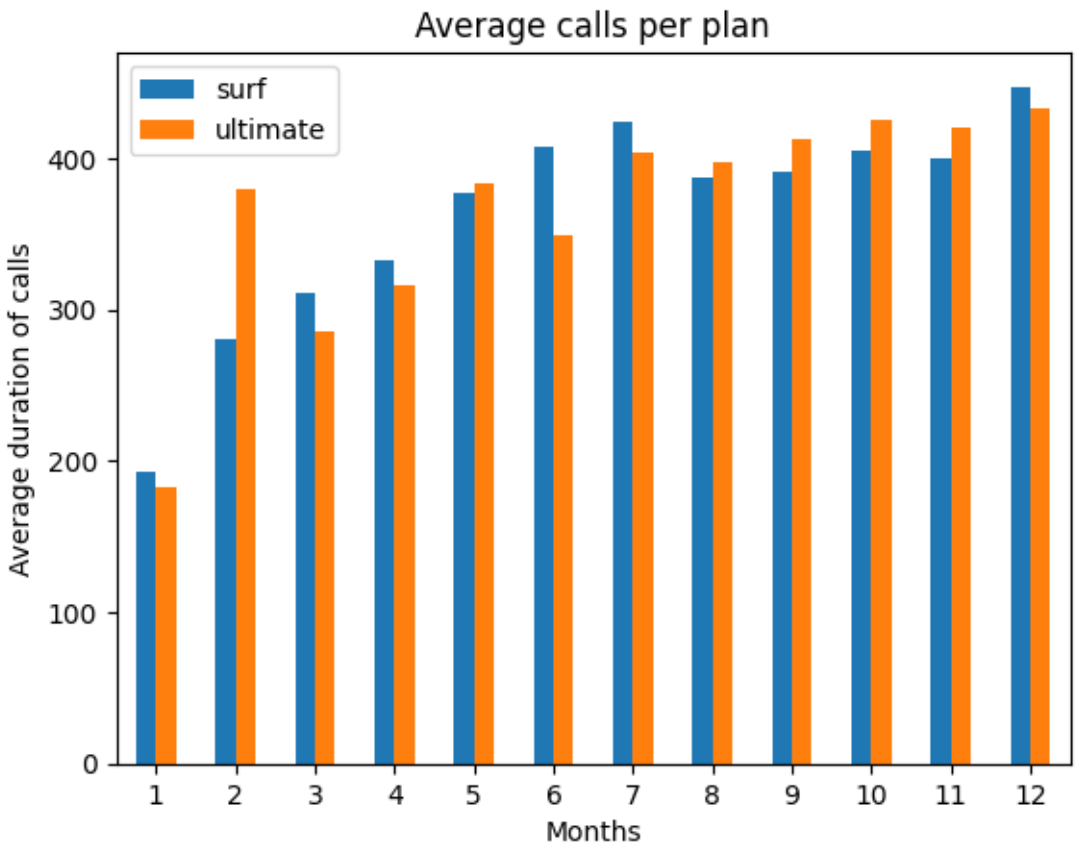
```
     user_id  month  number_of_calls  minutes_used  number_of_messages  \
0       1000     12             16.0         117.0                11.0
1       1001      8             27.0         171.0                30.0
2       1001      9             49.0         298.0                44.0
3       1001     10             65.0         374.0                53.0
4       1001     11             64.0         405.0                36.0
…        …      …                …             …                   …
2288    1349     12              0.0           0.0                61.0
2289    1361      5              0.0           0.0                 2.0
2290    1482     10              0.0           0.0                 2.0
2291    1108     12              0.0           0.0                 0.0
2292    1311      6              0.0           0.0                 0.0

      internet_used  first_name  last_name  age  \
0               2.0    Anamaria      Bauer   45
1               7.0      Mickey  Wilkerson   28
2              13.0      Mickey  Wilkerson   28
3              22.0      Mickey  Wilkerson   28
4              18.0      Mickey  Wilkerson   28
…               …          …          …  …
2288           13.0  Florentina       Diaz   69
2289            1.0     Jacelyn    Hoffman   45
2290            0.0      Armand      Glenn   70
2291            0.0    Porfirio       Kane   45
2292            1.0     Cherlyn   Saunders   69

                                      city  … minutes_included  \
0         Atlanta-Sandy Springs-Roswell, GA MSA  …              3000
1               Seattle-Tacoma-Bellevue, WA MSA  …               500
2               Seattle-Tacoma-Bellevue, WA MSA  …               500
3               Seattle-Tacoma-Bellevue, WA MSA  …               500
4               Seattle-Tacoma-Bellevue, WA MSA  …               500
…                                          …  …                 …
```

```
2288         Boston-Cambridge-Newton, MA-NH MSA  …              500
2289                   Birmingham-Hoover, AL MSA  …              500
2290  New York-Newark-Jersey City, NY-NJ-PA MSA  …             3000
2291                      Salt Lake City, UT MSA  …             3000
2292                       Memphis, TN-MS-AR MSA  …             3000

     usd_monthly_pay usd_per_gb  usd_per_message  usd_per_minute  \
0                 70          7             0.01            0.01
1                 20         10             0.03            0.03
2                 20         10             0.03            0.03
3                 20         10             0.03            0.03
4                 20         10             0.03            0.03
…               …         …              …               …
2288              20         10             0.03            0.03
2289              20         10             0.03            0.03
2290              70          7             0.01            0.01
2291              70          7             0.01            0.01
2292              70          7             0.01            0.01

      gb_per_month_included  revenue_per_calls  revenue_per_messages  \
0                      30.0               -0.0                 -0.00
1                      15.0               -0.0                 -0.00
2                      15.0               -0.0                 -0.00
3                      15.0               -0.0                  0.09
4                      15.0               -0.0                 -0.00
…                     …                 …                   …
2288                   15.0               -0.0                  0.33
2289                   15.0               -0.0                 -0.00
2290                   30.0               -0.0                 -0.00
2291                   30.0               -0.0                 -0.00
2292                   30.0               -0.0                 -0.00

      revenue_per_internet  total_revenue
0                     -0.0          70.00
1                     -0.0          20.00
2                     -0.0          20.00
3                     70.0          90.09
4                     30.0          50.00
…                    …             …
2288                  -0.0          20.33
2289                  -0.0          20.00
2290                  -0.0          70.00
2291                  -0.0          70.00
2292                  -0.0          70.00

[2293 rows x 25 columns]
```

1.14 Estudia el comportamiento de usuario

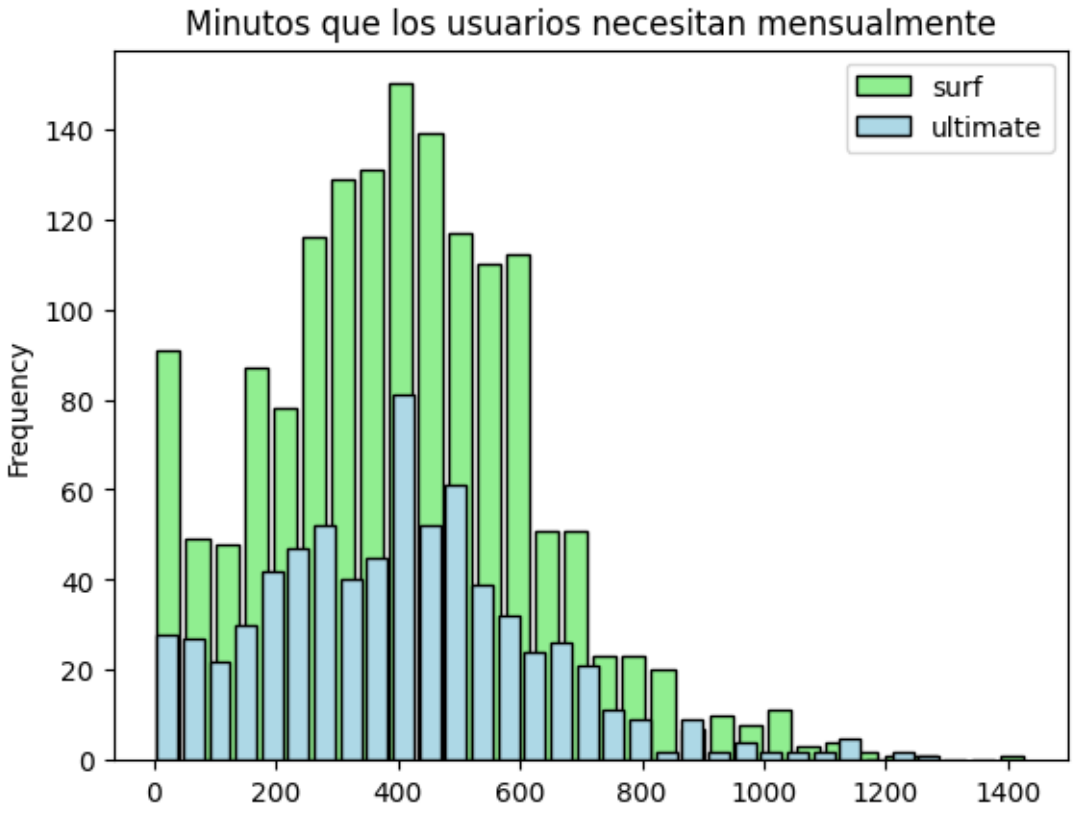
1.14.1 Llamadas

[108]:
```
# Compara la duración promedio de llamadas por cada plan y por cada mes. Traza un gráfico de barras para visualizarla.

# Agruparemos el dataframe por plan y por mes, para calcular la duración promedio de llamadas
df_avg_calls = consumption_per_user.groupby(['plan_name','month'])['minutes_used'].mean().reset_index()

# Crearemos el dataframe con columnas diferentes para cada plan, para así poder visualizar una columna por plan en nuestro gráfico
surf_plan = ['surf']
df_surf_plan = df_avg_calls[df_avg_calls['plan_name'].isin(surf_plan)]
ultimate_plan = ['ultimate']
df_ultimate_plan = df_avg_calls[df_avg_calls['plan_name'].isin(ultimate_plan)]

df_avg_calls_merged = df_surf_plan.merge(df_ultimate_plan, on='month', how='outer')
df_avg_calls_merged = df_avg_calls_merged.drop(['plan_name_x','plan_name_y'], axis = 1)
df_avg_calls_merged.columns = ['months','average_minutes_surf','average_minutes_ultimate']

# Crearemos nuestro gráfico de barras
df_avg_calls_merged.plot(
    x = 'months',
    kind = 'bar',
    title = 'Average calls per plan',
    xlabel = 'Months',
    ylabel = 'Average duration of calls',
    rot=0)

plt.legend(['surf','ultimate'])
plt.show()
```



[109]:
```
# Compara el número de minutos mensuales que necesitan los usuarios de cada␣
 ↪plan. Traza un histograma.

# Filtraremos por plan nuestros datos, para así trazar un histograma de los␣
 ↪minutos consumidos por cada plan
df_surf_minutes = consumption_per_user.query("plan_name ==␣
 ↪'surf'")[['user_id','plan_name','minutes_used']]
df_ultimate_minutes = consumption_per_user.query("plan_name ==␣
 ↪'ultimate'")[['user_id','plan_name','minutes_used']]

# Trazaremos el histograma de minutos mensuales que necesitan los usuarios del␣
 ↪plan surf y ultimate
df_surf_minutes['minutes_used'].plot(kind='hist',
                                     bins=30,
                                     title='Minutos que los usuarios necesitan␣
 ↪mensualmente',
                                     color='lightgreen',
                                     ec='black',
                                     rwidth=0.8)
```

```
df_ultimate_minutes['minutes_used'].plot(kind='hist',
                                         bins=30,
                                         title='Minutos que los usuarios necesitan␣
 ↪mensualmente',
                                         color='lightblue',
                                         ec='black',
                                         rwidth=0.8)

plt.legend(['surf','ultimate'])
plt.show()
```

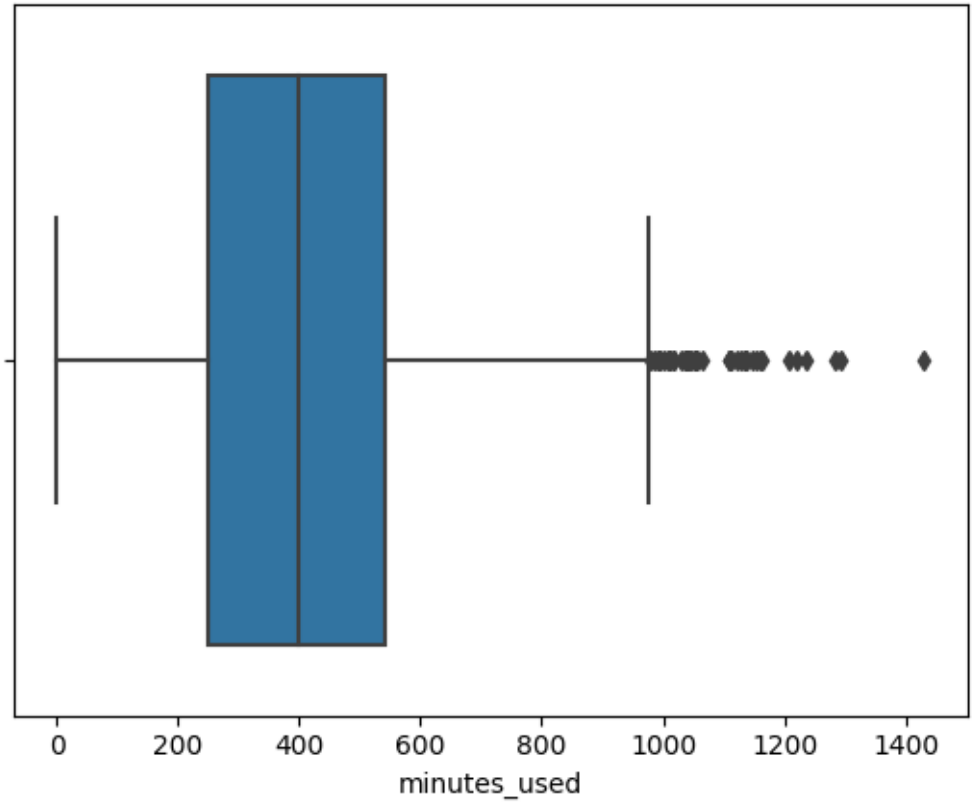


[110]:
```
# Calcula la media y la varianza de la duración mensual de llamadas.

print('La media de la duración mensual de llamadas es:',␣
 ↪consumption_per_user['minutes_used'].mean())
print('La varianza de la duración mensual de llamadas es:', np.
 ↪var(consumption_per_user['minutes_used']))
```

```
La media de la duración mensual de llamadas es: 405.1993022241605
La varianza de la duración mensual de llamadas es: 49878.178769683196
```

[111]:
```
# Traza un diagrama de caja para visualizar la distribución de la duración␣
↪mensual de llamadas

sns.boxplot(consumption_per_user['minutes_used'])
```

[111]: <AxesSubplot:xlabel='minutes_used'>

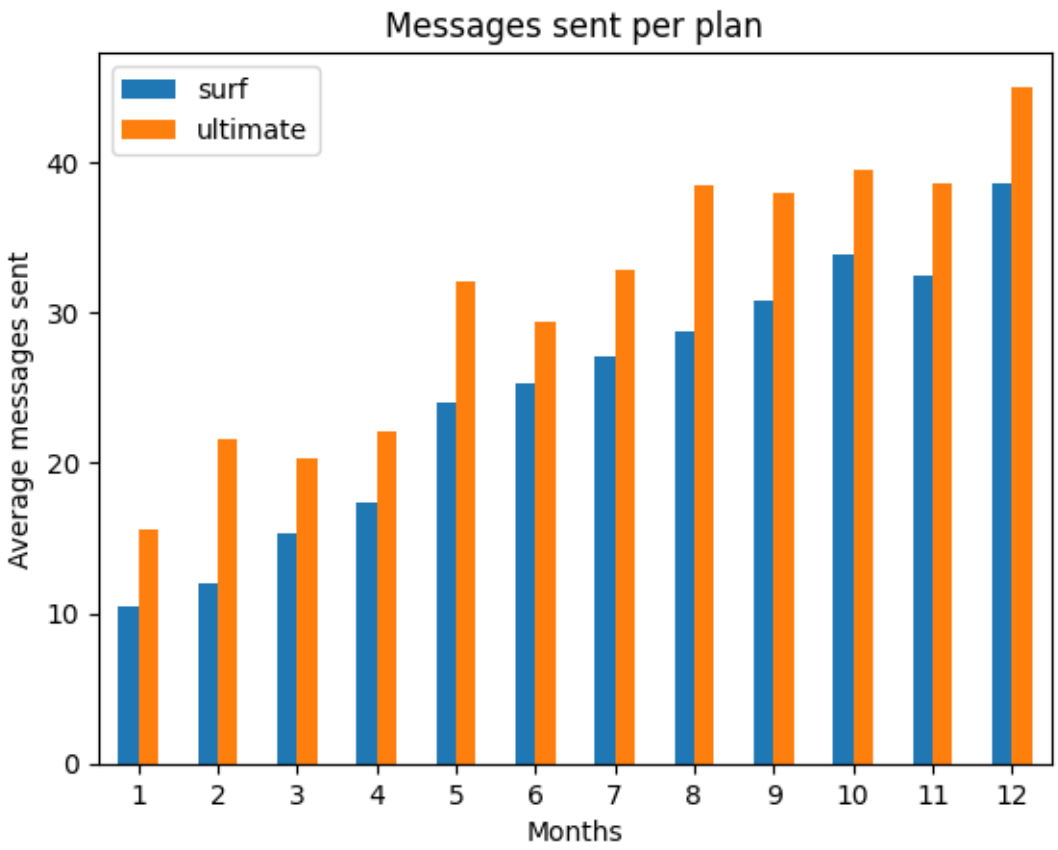


Analizando nuestros gráficos, podemos suponer que los usuarios de ambos planes tienen una media de minutos muy similar, alrededor de 400 minutos por mes. Cabe recalcar que existen muchos mas usuarios que cuentan con plan surf que ultimate, pero lo curioso es que ambos suelen utilizar una cantidad muy similar de minutos, a pesar de que el precio es mucho mayor para el plan ultimate.

1.14.2 Mensajes

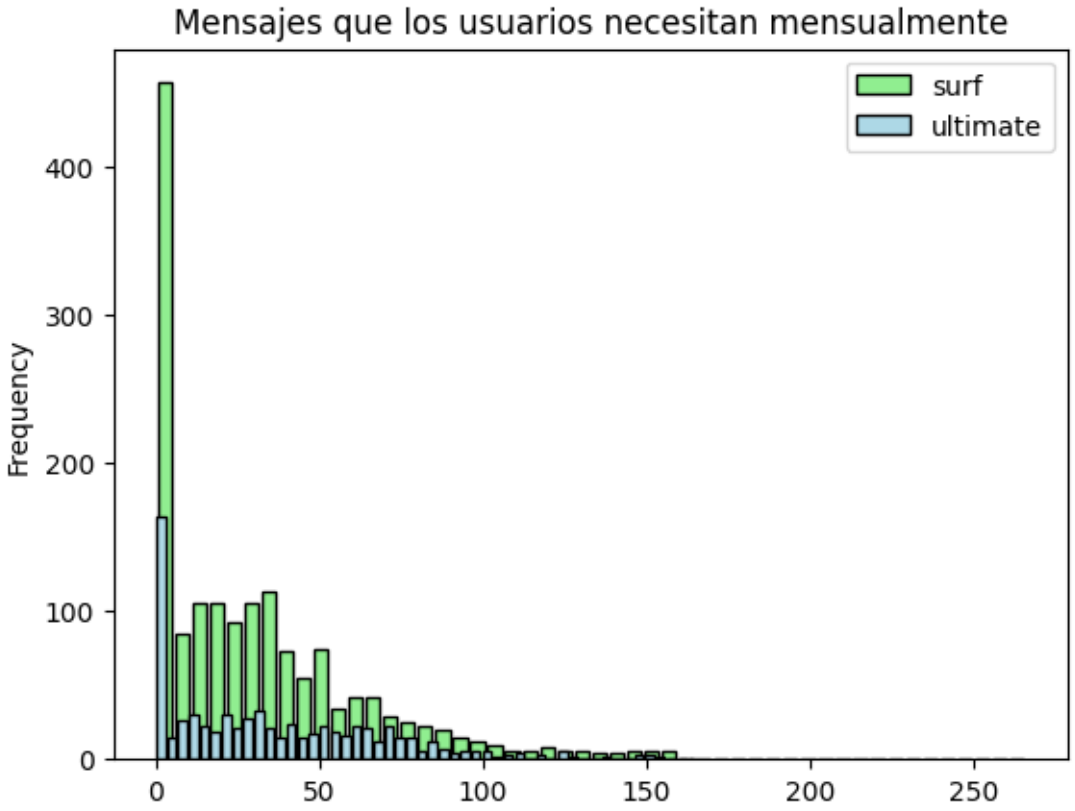
[112]:
```
# Comprara el número de mensajes que tienden a enviar cada mes los usuarios de␣
↪cada plan

# Agruparemos el dataframe por plan y por mes, para calcular el número de␣
↪mensajes que tienden a enviar los usuarios
df_avg_messages = consumption_per_user.
↪groupby(['plan_name','month'])['number_of_messages'].mean().reset_index()
```

```
# Crearemos el dataframe con columnas diferentes para cada plan, para así poder␣
 ↪visualizar una columna por plan en nuestro gráfico
surf_plan = ['surf']
df_surf_plan_m = df_avg_messages[df_avg_messages['plan_name'].isin(surf_plan)]
ultimate_plan = ['ultimate']
df_ultimate_plan_m = df_avg_messages[df_avg_messages['plan_name'].
 ↪isin(ultimate_plan)]

df_avg_messages_merged = df_surf_plan_m.merge(df_ultimate_plan_m, on='month',␣
 ↪how='outer')
df_avg_messages_merged = df_avg_messages_merged.
 ↪drop(['plan_name_x','plan_name_y'], axis = 1)
df_avg_messages_merged.columns =␣
 ↪['months','average_messages_surf','average_messages_ultimate']

# Crearemos nuestro gráfico de barras
df_avg_messages_merged.plot(
    x = 'months',
    kind = 'bar',
    title = 'Messages sent per plan',
    xlabel = 'Months',
    ylabel = 'Average messages sent',
    rot=0)

plt.legend(['surf','ultimate'])
plt.show()
```



[113]:
```
# Compara el número de mensajes por mes que necesitan los usuarios de cada plan.
 ↪ Traza un histograma.

# Filtraremos por plan nuestros datos, para así trazar un histograma de los␣
 ↪minutos consumidos por cada plan
df_surf_messages = consumption_per_user.query("plan_name ==␣
 ↪'surf'")[['user_id','plan_name','number_of_messages']]
df_ultimate_messages = consumption_per_user.query("plan_name ==␣
 ↪'ultimate'")[['user_id','plan_name','number_of_messages']]

# Trazaremos el histograma de mensajes mensuales que necesitan los usuarios del␣
 ↪plan surf y ultimate
df_surf_messages['number_of_messages'].plot(kind='hist',
                                            bins=50,
                                            title='Mensajes que los usuarios necesitan␣
 ↪mensualmente',
                                            color='lightgreen', ec='black',
                                            rwidth=0.8)
```

```
df_ultimate_messages['number_of_messages'].plot(kind='hist',
                                        bins=50,
                                        title='Mensajes que los usuarios necesitan␣
 ↪mensualmente',
                                        color='lightblue', ec='black',
                                        rwidth=0.8)

plt.legend(['surf','ultimate'])
plt.show()
```

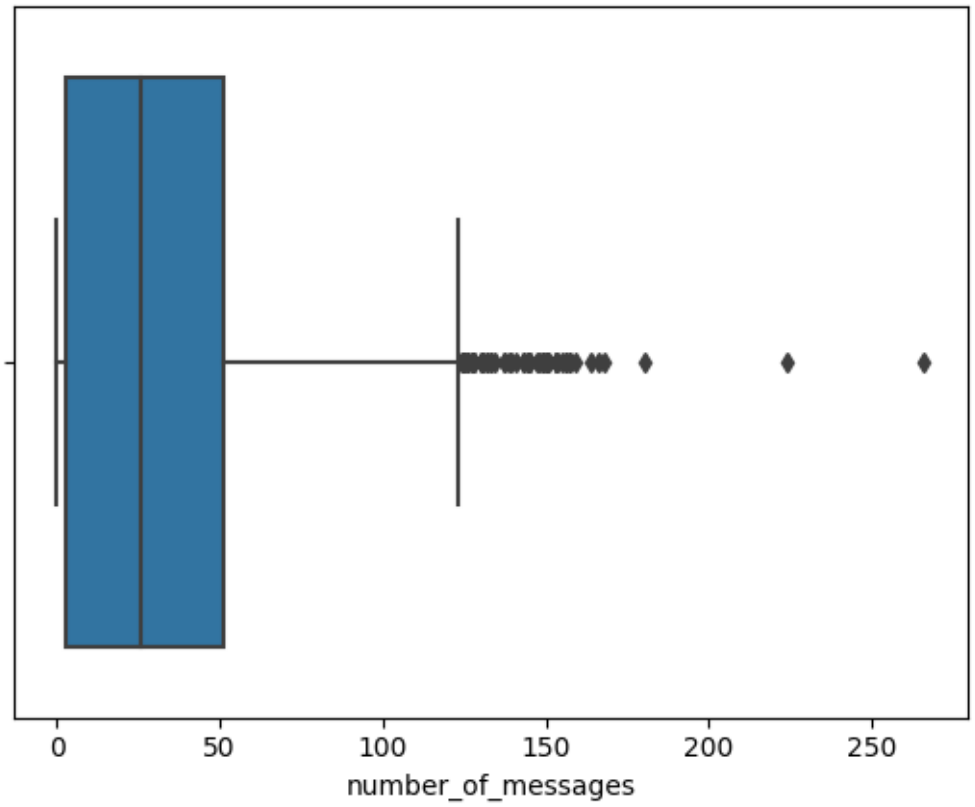


[114]:
```
# Calcula la media y la varianza de la cantidad de mensajes enviados al mes.

print('La media de los mensajes enviados al mes es:',␣
 ↪consumption_per_user['number_of_messages'].mean())
print('La varianza de los mensajes enviados al mes es:', np.
 ↪var(consumption_per_user['number_of_messages']))
```

```
La media de los mensajes enviados al mes es: 33.166593981683384
La varianza de los mensajes enviados al mes es: 1160.2644400780625
```

[115]:
```
# Traza un diagrama de caja para visualizar la distribución de los mensajes␣
↪enviados al mes

sns.boxplot(consumption_per_user['number_of_messages'])
```

[115]: <AxesSubplot:xlabel='number_of_messages'>

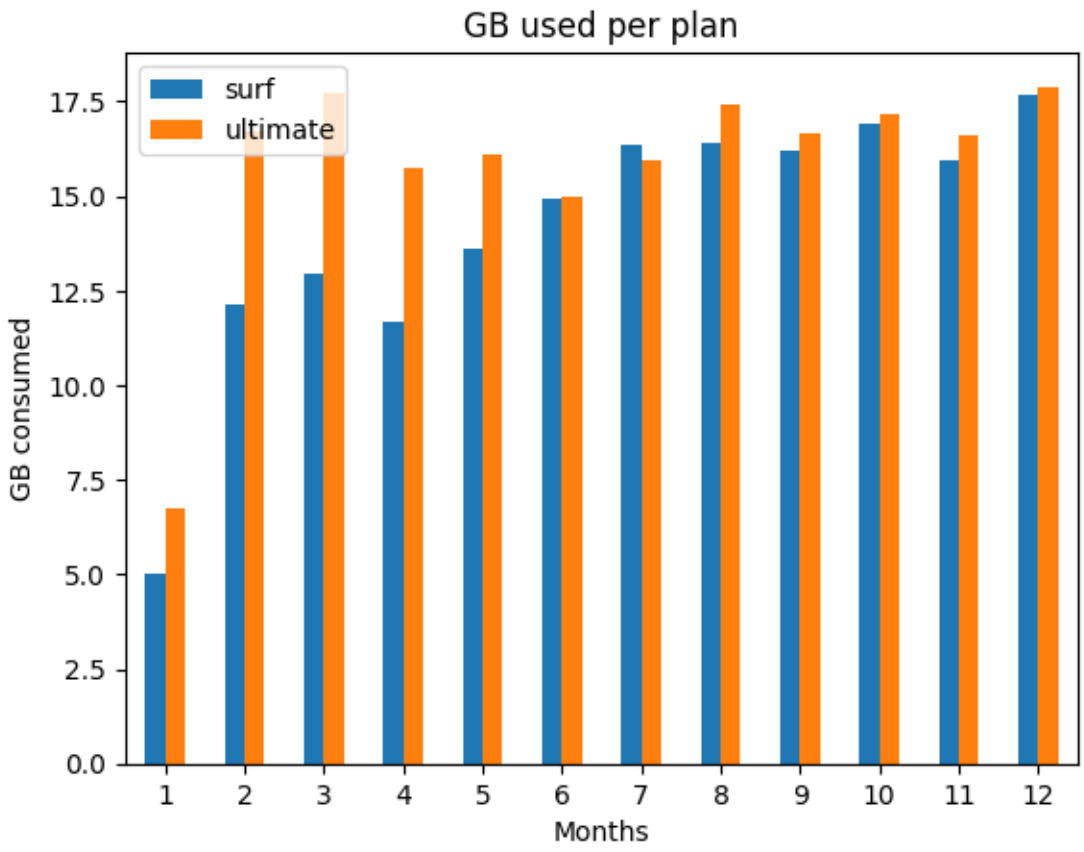


El uso de mensajes difiere un poco al de llamadas, y es que, los mensajes son mucho menos utilizados que las llamdas, ya que existe un gran porcentaje de los usuarios que no envían mensajes. Estas cifras no me sorprenden, ya que con los recientes servicios de mensajeria por aplicaciones como Whatsapp o Messenger, han desplazado este tipo de servicios. De igual manera, la media de mensajes enviados al mes es muy similar para ambos planes, al rededor de 30 mensajes por mes, recalcando que existen más usuarios por plan surf que por ultimate.

1.14.3 Internet

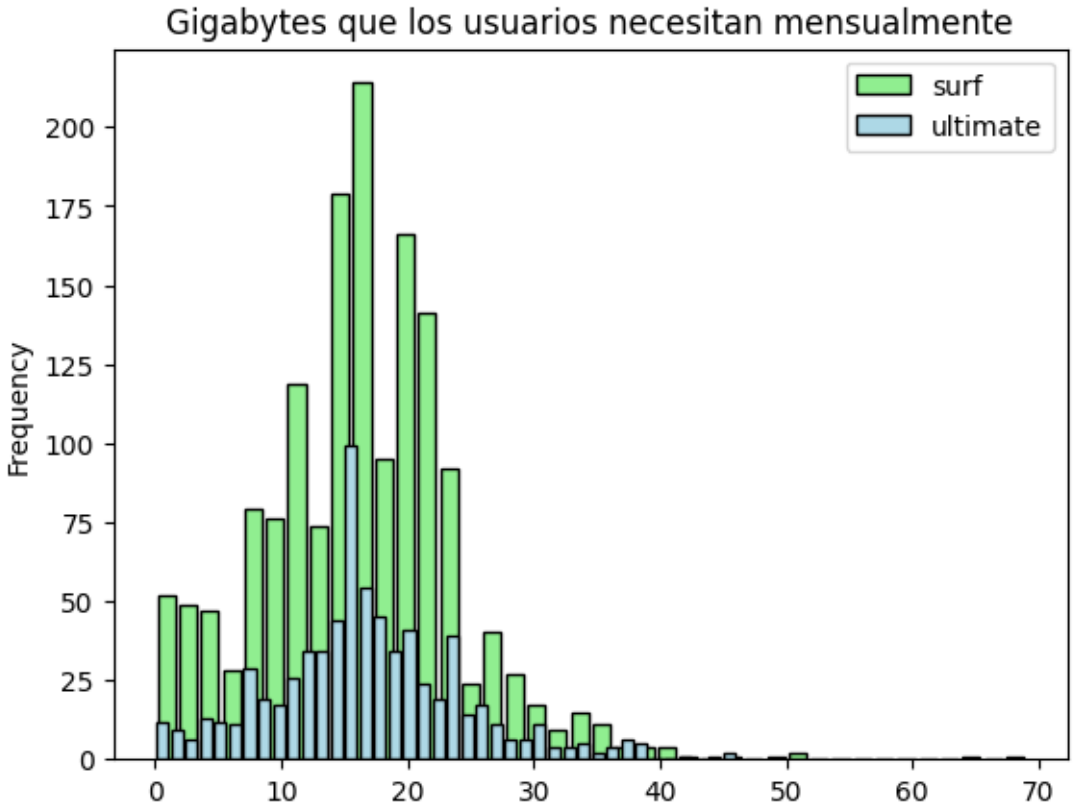
[116]:
```
# Compara la cantidad de tráfico de Internet consumido por usuarios por plan y␣
↪por mes

# Agruparemos el dataframe por plan y por mes, para calcular la cantidad de␣
↪tráfico de internet que consumen los usuarios
```

```
df_avg_internet = consumption_per_user.
 ↪groupby(['plan_name','month'])['internet_used'].mean().reset_index()

# Crearemos el dataframe con columnas diferentes para cada plan, para así poder␣
 ↪visualizar una columna por plan en nuestro gráfico
surf_plan = ['surf']
df_surf_plan_i = df_avg_internet[df_avg_internet['plan_name'].isin(surf_plan)]
ultimate_plan = ['ultimate']
df_ultimate_plan_i = df_avg_internet[df_avg_internet['plan_name'].
 ↪isin(ultimate_plan)]

df_avg_internet_merged = df_surf_plan_i.merge(df_ultimate_plan_i, on='month',␣
 ↪how='outer')
df_avg_internet_merged = df_avg_internet_merged.
 ↪drop(['plan_name_x','plan_name_y'], axis = 1)
df_avg_internet_merged.columns =␣
 ↪['months','average_gb_surf','average_gb_ultimate']

# Crearemos nuestro gráfico de barras
df_avg_internet_merged.plot(
    x = 'months',
    kind = 'bar',
    title = 'GB used per plan',
    xlabel = 'Months',
    ylabel = 'GB consumed',
    rot=0)

plt.legend(['surf','ultimate'])
plt.show()
```



[117]:
```
# Compara el flujo de internet por mes que necesitan los usuarios de cada plan.␣
 ↪Traza un histograma.

# Filtraremos por plan nuestros datos, para así trazar un histograma de los␣
 ↪gigabytes consumidos por cada plan
df_surf_internet = consumption_per_user.query("plan_name ==␣
 ↪'surf'")[['user_id','plan_name','internet_used']]
df_ultimate_internet = consumption_per_user.query("plan_name ==␣
 ↪'ultimate'")[['user_id','plan_name','internet_used']]

# Trazaremos el histograma de gigabyes mensuales que necesitan los usuarios del␣
 ↪plan surf y ultimate
df_surf_internet['internet_used'].plot(kind='hist',
                                       bins=40,
                                       title='Gigabytes que los usuarios␣
 ↪necesitan mensualmente',
                                       color='lightgreen', ec='black',
                                       rwidth=0.8)
```

```
df_ultimate_internet['internet_used'].plot(kind='hist',
                                           bins=40,
                                           title='Gigabytes que los usuarios␣
 ↪necesitan mensualmente',
                                           color='lightblue', ec='black',
                                           rwidth=0.8)

plt.legend(['surf','ultimate'])
plt.show()
```

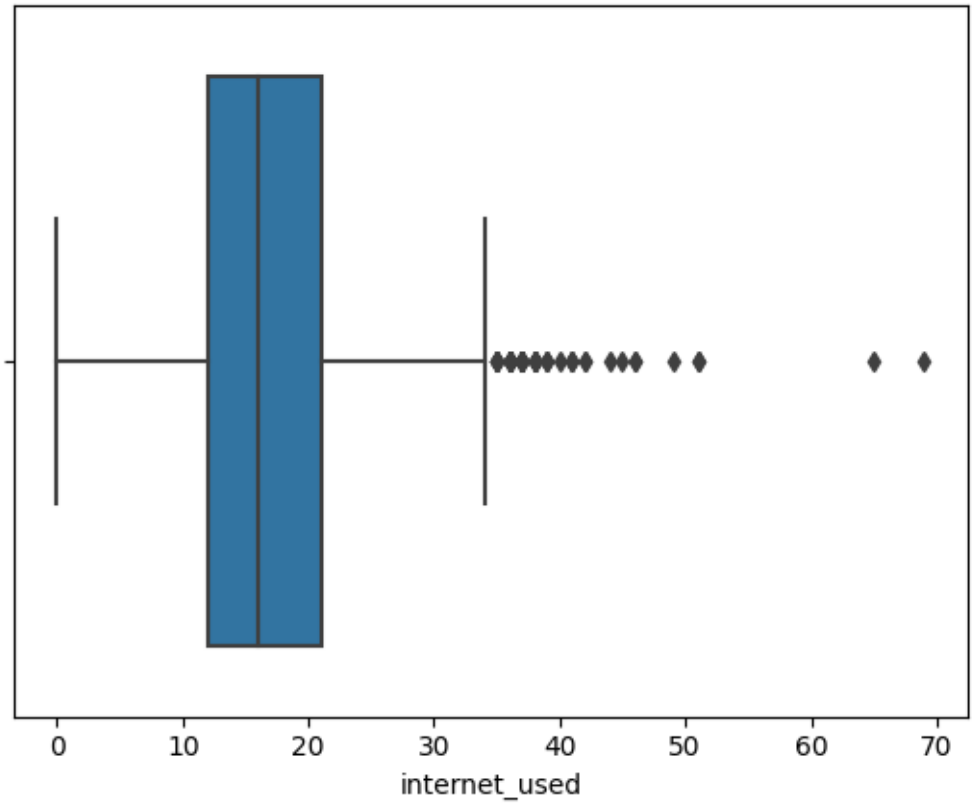


[118]:
```
# Calcula la media y la varianza de la cantidad de gigabytes consumidos al mes.

print('La media de los gigabytes consumidos al mes es:',␣
 ↪consumption_per_user['internet_used'].mean())
print('La varianza de los gigabytes consumidos al mes es:', np.
 ↪var(consumption_per_user['internet_used']))
```

```
La media de los gigabytes consumidos al mes es: 16.366332315743566
La varianza de los gigabytes consumidos al mes es: 60.49118223060418
```

[119]:
```
# Traza un diagrama de caja para visualizar el consumo de gigabytes al mes

sns.boxplot(consumption_per_user['internet_used'])
```

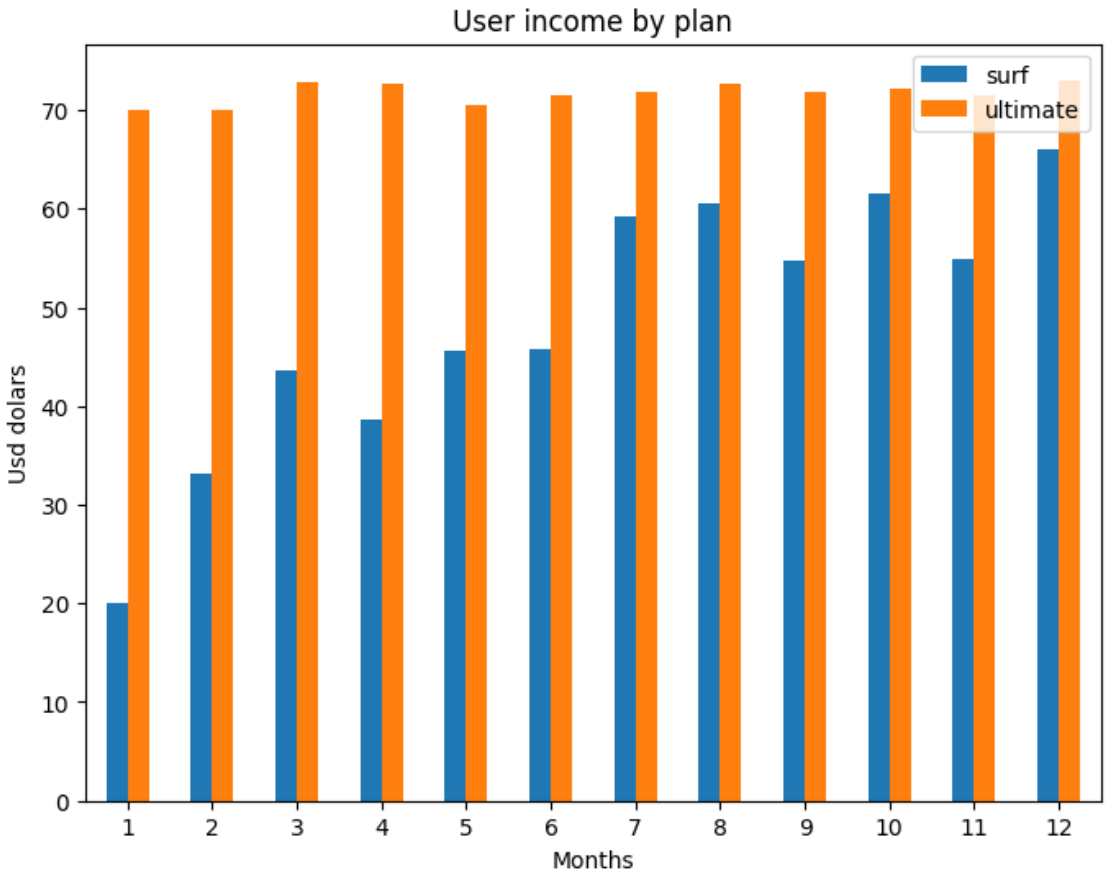
[119]: <AxesSubplot:xlabel='internet_used'>



Podemos suponer que al igual que los otros servicios, el uso de internet de los usuarios al mes es muy similar para ambos planes, manteniendo una media de consumo de alrededor de 16 gigabytes. Es un poco triste para los usuarios de ambos planes, ya que para los usuarios de surf, 16 gigabytes es una cantidad que rebasa el limite permitido en su plan, mientras que para los usuarios de ultimate apenas consumen un poco mas de la mitad de gigabytes que ofrece dicho plan. No obstante, existen muchos más usuarios con el plan surf. Ahora veo que estas cantidades fueron estrategicamente planeadas, ya que al conocer el promedio de consumo de internet de los usuarios, han establecido estos limites de modo que si tienes surf tendrás que pagar las tarifas de exceso de consumo, diez dólares por gigabyte, nada barato teniendo en cuenta que el valor mensual del plan es de 20 dólares; por otra parte si decides adquirir el plan ultimate la cantidad de gigabytes permitidos es mucho mayor, dandote más libertad del uso de datos de navegación, aunque el precio se eleva bastante del plan surf, setenta dólares al mes. Veamos como esta estrategia de precios afecta el ingreso de los usuarios.

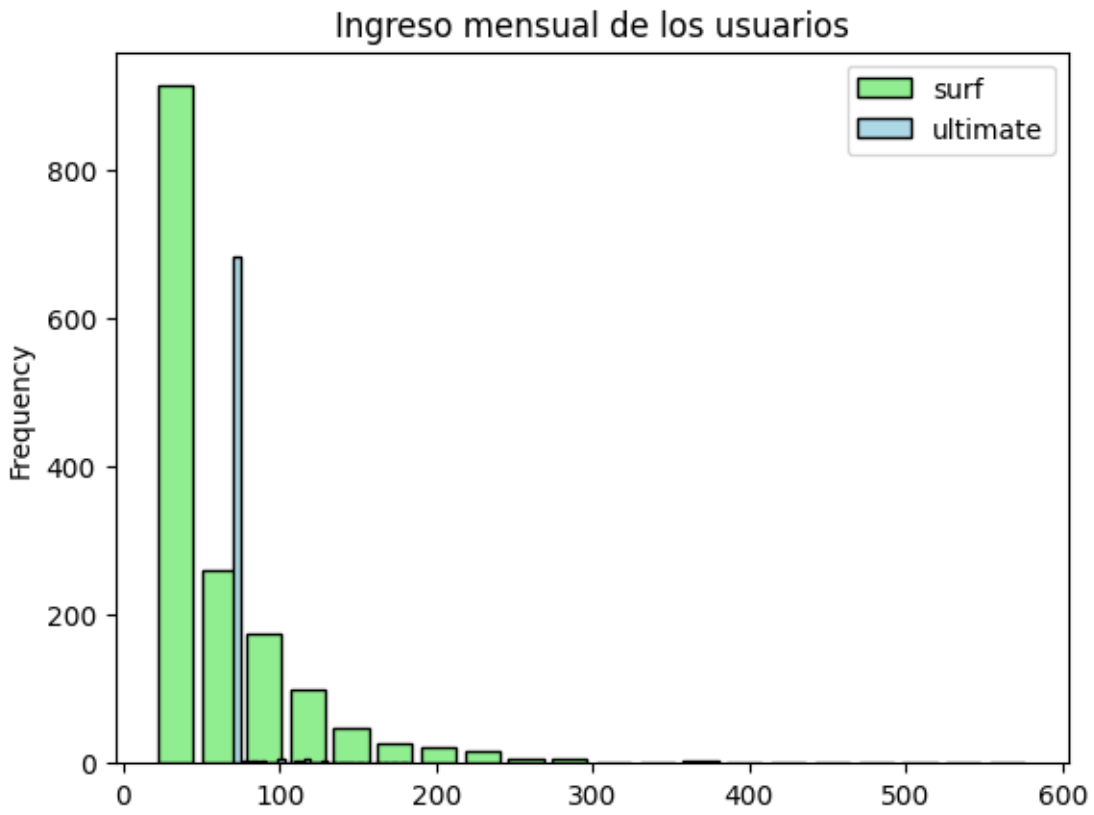
1.15 Ingreso

[120]:
```
# Compara el ingreso de los usuarios por plan y por mes

# Agruparemos el dataframe por plan y por mes, para calcular el ingreso mensual␣
 ↪de los usuarios
df_avg_income = consumption_per_user.
 ↪groupby(['plan_name','month'])['total_revenue'].mean().reset_index()

# Crearemos el dataframe con columnas diferentes para cada plan, para así poder␣
 ↪visualizar una columna por plan en nuestro gráfico
surf_plan = ['surf']
df_surf_users = df_avg_income[df_avg_income['plan_name'].isin(surf_plan)]
ultimate_plan = ['ultimate']
df_ultimate_users = df_avg_income[df_avg_income['plan_name'].
 ↪isin(ultimate_plan)]

df_avg_income_merged = df_surf_users.merge(df_ultimate_users, on='month',␣
 ↪how='outer')
df_avg_income_merged = df_avg_income_merged.drop(['plan_name_x','plan_name_y'],␣
 ↪axis = 1)
df_avg_income_merged.columns =␣
 ↪['months','average_income_surf','average_income_ultimate']

# Crearemos nuestro gráfico de barras
df_avg_income_merged.plot(
    x = 'months',
    kind = 'bar',
    title = 'User income by plan',
    xlabel = 'Months',
    ylabel = 'Usd dolars',
    rot=0,
    figsize=(8,6))

plt.legend(['surf','ultimate'])
plt.show()
```



[121]:
```
# Compara la cantidad monetaria por mes que necesitan los usuarios de cada plan.
 ↪ Traza un histograma.

# Filtraremos por plan nuestros datos, para así trazar un histograma del␣
 ↪ingreso mensual de los usuarios por cada plan
df_surf_income = consumption_per_user.query("plan_name ==␣
 ↪'surf'")[['user_id','plan_name','total_revenue']]
df_ultimate_income = consumption_per_user.query("plan_name ==␣
 ↪'ultimate'")[['user_id','plan_name','total_revenue']]

# Trazaremos el histograma de ingresos mensuales de los usuarios del plan surf␣
 ↪y ultimate
df_surf_income['total_revenue'].plot(kind='hist',
                                     bins=20,
                                     title='Ingreso mensual de los usuarios',
                                     color='lightgreen', ec='black',
                                     rwidth=0.8)

df_ultimate_income['total_revenue'].plot(kind='hist',
```

```
                                    bins=20,
                                    title='Ingreso mensual de los usuarios',
                                    color='lightblue', ec='black',
                                    rwidth=0.8)

plt.legend(['surf','ultimate'])
plt.show()
```



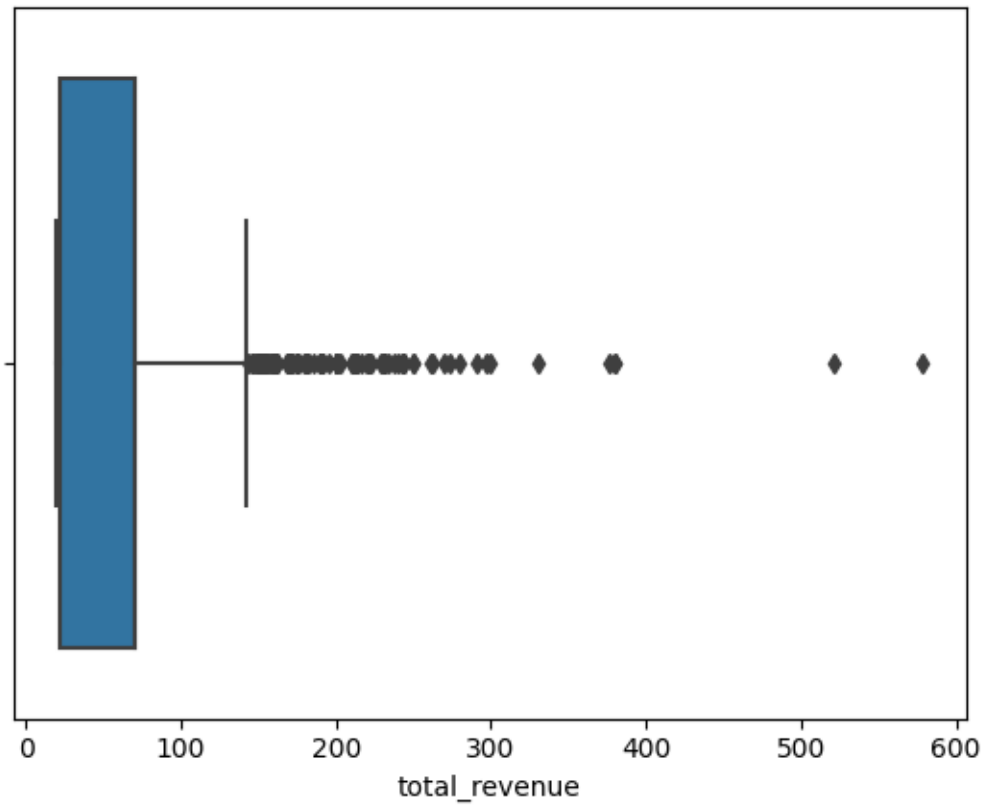
[122]:
```
# Calcula la media y la varianza del ingreso mensual de los usuarios.

print('La media del ingreso mensual es:', consumption_per_user['total_revenue'].
  ↪mean())
print('La varianza del ingreso mensual es:', np.
  ↪var(consumption_per_user['total_revenue']))
```

```
La media del ingreso mensual es: 61.86822503270824
La varianza del ingreso mensual es: 2062.6747979397655
```

[123]:
```
# Traza un diagrama de caja para visualizar la distribución del ingreso mensual␣
  ↪de los usuarios
```

```
sns.boxplot(consumption_per_user['total_revenue'])
```

[123]: <AxesSubplot:xlabel='total_revenue'>



Tal como se comentó anteriormente, si bien los usuarios del plan ultimate no suelen pagar tarifas por exceso de minutos, mensajes o internet esto no ocurre con los usuarios del plan surf, ya que es muy común que paguen tarifas por exceder los servicios ofrecidos, especilmente el último mes del año. Añadiendole que la cantidad de usuarios del plan surf es mucho mayor a la del plan ultimate pero a pesar de ello la media del ingreso mensual se acerca más al precio del plan ultimate que a la del plan surf, 61 dólares aproximadamente. Esto se explica ya que existen usuarios del plan surf que llegan a pagar más de 70 dólares mensuales debido a las tarifas de exceso de consumo. Una locura!

1.16 Prueba las hipótesis estadísticas

[124]:
```
# Prueba las hipótesis

# Filtraremos los datos por plan para poder tener dos poblaciones, una para␣
 ↪usuarios del plan surf y otra para usuarios del plan ultimate
array_1 = consumption_per_user.query("plan_name == 'surf'")['total_revenue']
array_2 = consumption_per_user.query("plan_name == 'ultimate'")['total_revenue']
```

```
# Comprobaremos que nuestras muestras tienen varianzas iguales

alpha = 0.05

levene_test = st.levene(array_1, array_2, center = 'median')

if levene_test.pvalue < alpha:
    print('Los grupos tienen varianzas iguales')
else:
    print('Los grupos no tienen varianzas iguales')

# Establecemos nuestra hipótesis nula y alternativa:
# H0 = El ingreso de los usuarios del plan Surf es igual al ingreso de los␣
 ↪usuarios del plan Ultimate
# H1 = El ingreso de los usuarios del plan Surf es diferente al ingreso de los␣
 ↪usuarios del plan Ultimate

# Probamos nuestra hipótesis nula con un valor alpha del 5%

results = st.ttest_ind(array_1, array_2, equal_var=True)

print('valor p:', results.pvalue)

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

```
Los grupos tienen varianzas iguales
valor p: 2.0640090656105099e-13
Rechazamos la hipótesis nula
```

[125]:
```
# Prueba las hipótesis

# Filtraremos los datos por área NY-NJ para poder tener dos muestras, una para␣
 ↪usuarios del área NY-NJ y otra para usuarios de las demás áreas
city = ['New York-Newark-Jersey City, NY-NJ-PA MSA']
sample_1 = consumption_per_user[consumption_per_user['city'].
 ↪isin(city)]['total_revenue']
sample_2 = consumption_per_user[~consumption_per_user['city'].
 ↪isin(city)]['total_revenue']

# Comprobaremos que nuestras muestras tienen varianzas iguales

alpha = 0.05
```

```
l_test = st.levene(array_1, array_2, center = 'median')

if l_test.pvalue < alpha:
    print('Los grupos tienen varianzas iguales')
else:
    print('Los grupos no tienen varianzas iguales')

# Establecemos nuestra hipótesis nula y alternativa:
# H0 = El ingreso de los usuarios del área NY-NJ es igual al ingreso de los␣
 ↪demás usuarios
# H1 = El ingreso de los usuarios del área NY-NJ es diferente al ingreso de los␣
 ↪demás usuarios

# Probamos nuestra hipótesis nula con un valor alpha del 5%

results = st.ttest_ind(sample_1, sample_2)

print('valor p:', results.pvalue)

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

```
Los grupos tienen varianzas iguales
valor p: 0.02645886317327806
Rechazamos la hipótesis nula
```

1.17 Conclusión general

Finalmente, después de limpiar, procesar, preparar y analizar nuestros datos, podemos suponer que los ingresos de los usuarios de cada plan ofrecido (Surf y Ultimate) son diferentes, incluso existe una diferencia de ingresos por área, tal como el área NY-NJ. Aunque es díficil hacer afirmaciones sobre que plan genera más ingresos, personalmente, creo que la estrategia de fijación de precios y tarifas por exceso de consumo les ha impactado muy efectivamente a la empresa, ya que por una parte tenemos una gran cantidad de usuarios que prefieren pagar una cifra mensual más baja pero al tener mayor límite de consumo se ven forzados a pagar las tarifas de exceso, las cuales elevan su costo mensual; y por otro lado tenemos a los usuarios que no quieren tener este limite reducido de consumo y prefieren pagar el plan que ofrece un límite mucho mayor de consumo pero eleva mucho su costo. Estimo que el plan Surf genera un mayor ingreso al tener una cantidad considerable mayor de usuarios, y a pesar de que no pagan una cantidad elevada mensual, su consumo es casi el mismo al de los otros usuarios, generando buenas utilidades para la empresa.