¿Cuál es un mejor plan?

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 los planes 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.

Observaciones:

- Megaline redondea los segundos a minutos y los megabytes a gigabytes.
- Para llamadas, cada llamada individual se redondea: incluso si la llamada duró solo un segundo, se contará como un minuto.
- Para tráfico web, las sesiones web individuales no se redondean. En vez de esto, el total del mes se redondea hacia arriba. Si alguien usa 1025 megabytes este mes, se le cobrarán 2 gigabytes.

OBJETIVOS

Para cada usuario, buscar:

1. El número de llamadas realizadas y la cantidad de minutos utilizados al mes.
2. La cantidad de los SMS enviados por mes.
3. El tráfico de internet por usuario y por mes.
4. Ganancias mensuales para el plan Surf y Ultimate.

Hipotesis a comprobar:

1. El ingreso promedio de los usuarios de las tarifas Ultimate y Surf difiere.
2. El ingreso promedio de los usuarios en el área de estados Nueva York–Nueva Jersey es diferente al de los usuarios de otras regiones.

Inicialización

In [711…

```
# Cargar todas las librerías
import pandas as pd
import numpy as np
from math import factorial
from scipy import stats as st

from matplotlib import pyplot as plt
import seaborn as sns
```

Cargar los datos

In [712…

```
# Carga los archivos de datos en diferentes DataFrames

calls = pd.read_csv('/datasets/megaline_calls.csv')
internet = pd.read_csv('/datasets/megaline_internet.csv')
messages = pd.read_csv('/datasets/megaline_messages.csv')
plans = pd.read_csv('/datasets/megaline_plans.csv')
users = pd.read_csv('/datasets/megaline_users.csv')
```

Preparar los datos

LLAMADAS

In [713…

```
calls.info()
calls.head()
```

```
<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
```

Out[713…

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

In [714…

```
calls.describe()
```

Out[714…

	user_id	duration
count	137735.000000	137735.000000
mean	1247.658046	6.745927
std	139.416268	5.839241
min	1000.000000	0.000000
25%	1128.000000	1.290000
50%	1247.000000	5.980000
75%	1365.000000	10.690000
max	1499.000000	37.600000

Observaciones:

1. No hay valores ausentes entre los datos.
2. Dtype de call_date debe ser cambiado al formato datetime.
3. La columna de duration tiene el tipo flotante (float64), pero de acuerdo a las instrucciones de la empresa Megaline se debe redondear los segundos a minutos, por lo que debemos convertir esta columna a números enteros.
4. Hay 137.735 llamadas en el conjunto de datos. La duración media y mediana es de unos 6-7 minutos. Los valores medio y mediano están próximos entre sí, lo que significa que la distribución es simétrica.
5. La duración mínima de una llamada es 0, lo que probablemente signifique unos segundos, según política de la empresa deberíamos redondearlo a 1 minuto.
6. La duración máxima de la llamada es de 38 minutos.

INTERNET

In [715…

```
internet.info()
internet.head()
```

```
<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
```

Out[715…

	id	user_id	session_date	mb_used
0	1000_13	1000	2018-12-29	89.86
1	1000_204	1000	2018-12-31	0.00
2	1000_379	1000	2018-12-28	660.40
3	1000_413	1000	2018-12-26	270.99
4	1000_442	1000	2018-12-27	880.22

Observaciones:

1. No hay valores ausentes entre los datos.
2. Dtype de session_date debe ser cambiado al formato datetime.
3. Para tráfico web, las sesiones web individuales no se redondean. En vez de esto, el total del mes se redondea hacia arriba y convertir esta columna a números enteros.

SMS - MENSAJES DE TEXTO

In [716…

```
messages.info()
messages.head()
```

```
<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
```

Out[716…

	id	user_id	message_date
0	1000_125	1000	2018-12-27
1	1000_160	1000	2018-12-31
2	1000_223	1000	2018-12-31
3	1000_251	1000	2018-12-27
4	1000_255	1000	2018-12-26

Observaciones:

1. No hay valores ausentes entre los datos.

2. Dtype de message_date debe ser cambiado al formato datetime.

USUARIOS

In [717…

```
users.info()
users.head()
```

```
<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
```

Out[717…

	user_id	first_name	last_name	age	city	reg_date	plan	churn_date
0	1000	Anamaria	Bauer	45	Atlanta-Sandy Springs-Roswell, GA MSA	2018-12-24	ultimate	NaN
1	1001	Mickey	Wilkerson	28	Seattle-Tacoma-Bellevue, WA MSA	2018-08-13	surf	NaN
2	1002	Carlee	Hoffman	36	Las Vegas-Henderson-Paradise, NV MSA	2018-10-21	surf	NaN
3	1003	Reynaldo	Jenkins	52	Tulsa, OK MSA	2018-01-28	surf	NaN
4	1004	Leonila	Thompson	40	Seattle-Tacoma-Bellevue, WA MSA	2018-05-23	surf	NaN

Observaciones:

1. Faltan valores en la columna 'churn_date'. Sin embargo, 'churn_date' es la fecha en que el usuario dejó de usar el servicio; si el valor es ausente, la tarifa se estaba usando cuando se generaron estos datos. En este caso cambiaremos NaN por 'in use'.
2. 'reg_date' y 'churn_date' son tipo objeto, lo cambiaremos al tipo 'datetime'.

Planes

In [718…

```
# Imprime la información general/resumen sobre el DataFrame de los planes
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
```

In [719…

```
# Imprime una muestra de los datos para los planes
plans.head()
```

Out[719…

	messages_included	mb_per_month_included	minutes_included	usd_monthly_pay	usd_per_
0	50	15360	500	20	
1	1000	30720	3000	70	

Nomenclatura de la tabla plans (datos sobre las tarifas):

- plan_name — nombre de la tarifa
- usd_monthly_pay — pago mensual en dólares estadounidenses
- minutes_included — minutos incluidos al mes
- messages_included — SMS incluidos al mes
- mb_per_month_included — datos incluidos al mes (en megabytes)
- usd_per_minute — precio por minuto tras exceder los límites del paquete (por ejemplo, si el paquete incluye 100 minutos el operador cobrará el minuto 101)
- usd_per_message — precio por SMS tras exceder los límites del paquete
- usd_per_gb — precio por gigabyte de los datos extra tras exceder los límites del paquete (1 GB = 1024 megabytes)

Observaciones:

1. No hay valores ausentes entre los datos.
2. No hay problemas con estos datos. El resultado muestra que sólo hay 2 planes en este conjunto de datos: 'Surf' y 'Ultimate' que corresponde con lo mencionado por la compañia Megaline.
3. Dado que Megaline redondea los megabytes a gigabytes, podemos convertir gb_per_month_included a mb_per_month_included

Corregir los datos

LLAMADAS

Problemas:

1. Dtype de call_date debe ser cambiado al formato datetime.
2. La columna de duration tiene el tipo flotante (float64), pero de acuerdo a las instrucciones de la empresa Megaline se debe redondear los segundos a minutos, por lo que debemos convertir esta columna a números enteros y redondear.
3. La duración mínima de una llamada es 0, lo que probablemente significa que la llamada duro unos segundos, según política de la empresa vamos a redondearlo a 1 minuto.

In [720…

```
# Convertir call_date a datetime
calls['call_date'] = pd.to_datetime(calls['call_date'], format ='%Y-%m-%d')

# Redondear 'duration' y convertir a números enteros
```

```
#calls['duration'] = calls['duration'].apply(np.ceil).astype(int)
calls['duration'] = np.ceil(calls['duration']).astype('int')

# También vamos a crear una columna 'month' que tenga solo la parte del mes
calls['month'] = calls['call_date'].dt.month
```

In [721…

```
# Comprobar cambios
calls.info()
calls.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 137735 entries, 0 to 137734
Data columns (total 5 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  datetime64[ns]
 3   duration   137735 non-null  int64
 4   month      137735 non-null  int64
dtypes: datetime64[ns](1), int64(3), object(1)
memory usage: 5.3+ MB
```

Out[721…

	id	user_id	call_date	duration	month
0	1000_93	1000	2018-12-27	9	12
1	1000_145	1000	2018-12-27	14	12
2	1000_247	1000	2018-12-27	15	12
3	1000_309	1000	2018-12-28	6	12
4	1000_380	1000	2018-12-30	5	12

In [722…

```
calls['duration'] = calls['duration'].replace(0, 1)
```

In [723…

```
calls.describe()
```

Out[723…

	user_id	duration	month
count	137735.000000	137735.000000	137735.000000
mean	1247.658046	7.341496	9.320797
std	139.416268	5.728989	2.412550
min	1000.000000	1.000000	1.000000
25%	1128.000000	2.000000	8.000000
50%	1247.000000	6.000000	10.000000
75%	1365.000000	11.000000	11.000000
max	1499.000000	38.000000	12.000000

INTERNET

Problemas:

1. Dtype de session_date debe ser cambiado al formato datetime.

2. Para tráfico web, las sesiones web individuales no se redondean. En vez de esto, el total del mes se redondea hacia arriba y convertir esta columna a números enteros.

In [724…
```
# Convertir los tipos de datos 'session_date' en datetime
internet['session_date'] = pd.to_datetime(internet['session_date'], format ='

# También vamos a crear una columna 'session_month' que tenga solo la parte d
internet['month'] = internet['session_date'].dt.month
```

In [725…
```
# Comprobar cambios
internet.info()
internet.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104825 entries, 0 to 104824
Data columns (total 5 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  datetime64[ns]
 3   mb_used       104825 non-null  float64       
 4   month         104825 non-null  int64         
dtypes: datetime64[ns](1), float64(1), int64(2), object(1)
memory usage: 4.0+ MB
```

Out[725…

	id	user_id	session_date	mb_used	month
0	1000_13	1000	2018-12-29	89.86	12
1	1000_204	1000	2018-12-31	0.00	12
2	1000_379	1000	2018-12-28	660.40	12
3	1000_413	1000	2018-12-26	270.99	12
4	1000_442	1000	2018-12-27	880.22	12

SMS - MENSAJES DE TEXTO

Problemas:

1. Dtype de message_date debe ser cambiado al formato datetime.

In [726…
```
# Convertir message_date a datetime
messages['message_date'] = pd.to_datetime(messages['message_date'], format ='

# También vamos a crear una columna 'month' que tenga solo la parte del mes
messages['month'] = messages['message_date'].dt.month
```

In [727…
```
# Comprobar cambios
messages.info()
messages.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 76051 entries, 0 to 76050
Data columns (total 4 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  datetime64[ns]
 3   month         76051 non-null  int64
dtypes: datetime64[ns](1), int64(2), object(1)
memory usage: 2.3+ MB
```

Out[727…

	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

USUARIOS

Problemas:

1. Faltan valores en la columna 'churn_date'. Sin embargo, 'churn_date' es la fecha en que el usuario dejó de usar el servicio; si el valor es ausente, la tarifa se estaba usando cuando se generaron estos datos. En este caso consideramos esta columna innecesaria.
2. 'reg_date' es de tipo objeto, lo cambiaremos al tipo 'datetime'.
3. Los resultados deberán ser entregados por usuario, consideramos nombre, apellido, edad, y ciudad columnas innecesarias, por ende serán eliminadas.

In [728…

```
# Convertir reg_date a datetime
users['reg_date'] = pd.to_datetime(users['reg_date'], format ='%Y-%m-%d')

# Crear una nueva columna para mostrar el departamento de los usuarios extraí
users['state'] = users['city'].str[-6:].str[:2]

# Eliminar columnas innecesaria
users = users.drop(['first_name', 'last_name','age','city','reg_date','churn_
```

In [729…

```
# Comprobar cambios
users.info()
users.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   user_id  500 non-null    int64 
 1   plan     500 non-null    object
 2   state    500 non-null    object
dtypes: int64(1), object(2)
memory usage: 11.8+ KB
```

Out[729…

	user_id	plan	state
0	1000	ultimate	GA
1	1001	surf	WA
2	1002	surf	NV
3	1003	surf	OK
4	1004	surf	WA

Estudiar las condiciones del plan

Es fundamental comprender cómo funcionan los planes y cómo se les cobra a los usuarios en función de su suscripción al plan. Observaremos la información del plan para ver sus condiciones una vez más.

In [730…
```
# Imprime las condiciones del plan y asegúrate de que te resulten claros
plans.head()
```

Out[730…

	messages_included	mb_per_month_included	minutes_included	usd_monthly_pay	usd_per_
0	50	15360	500	20	
1	1000	30720	3000	70	

Calcular el número de llamadas hechas por cada usuario por mes.

Para calcular el número de llamadas realizadas por cada usuario al mes, necesitamos unir los datos tanto de usuarios (users) como de las llamadas (calls). Vamos a crear un nuevo Dataframe - users_calls fusionando los dos.

In [731…
```
# Unir el número de llamadas hechas por cada usuario
users_calls = users.merge(calls, on=['user_id'], how='outer')
users_calls.head()
```

Out[731…

	user_id	plan	state	id	call_date	duration	month
0	1000	ultimate	GA	1000_93	2018-12-27	9.0	12.0
1	1000	ultimate	GA	1000_145	2018-12-27	14.0	12.0
2	1000	ultimate	GA	1000_247	2018-12-27	15.0	12.0
3	1000	ultimate	GA	1000_309	2018-12-28	6.0	12.0
4	1000	ultimate	GA	1000_380	2018-12-30	5.0	12.0

Ahora que tenemos los datos consolidados, crearemos una tabla dinámica con user_id y month como índices y apliquemos count() para obtener la cantidad de llamadas realizadas por cada usuario por mes:

In [732…
```
# crear una tabla dinámica con user_id and call_month
calls_per_user = users_calls.pivot_table(index=['user_id', 'month'], aggfunc=

# Asignar nombres a las columnas de la tabla dinámica
calls_per_user.columns = ['number_of_calls']
```

In [733…
```
# Comprobar
calls_per_user.head(10)
```

Out[733…

		number_of_calls
user_id	**month**	
1000	**12.0**	16
1001	**8.0**	27

		number_of_calls
user_id	month	
	9.0	49
	10.0	65
	11.0	64
	12.0	56
1002	10.0	11
	11.0	55
	12.0	47
1003	12.0	149

Calcular la cantidad de minutos que usó cada usuario por mes.

In [734…

```
# Crear una tabla dinámica en user_id y call_month y sumar la duración
minutes_per_user_per_month = users_calls.pivot_table(index=['user_id', 'month
minutes_per_user_per_month = minutes_per_user_per_month.reset_index()

# Cambiar nombre de 'duration'
minutes_per_user_per_month.rename(columns={'duration': 'total_min_month'}, in
```

In [735…

```
# Comprobar
minutes_per_user_per_month.head()
```

Out[735…

	user_id	month	total_min_month
0	1000	12.0	124.0
1	1001	8.0	187.0
2	1001	9.0	326.0
3	1001	10.0	411.0
4	1001	11.0	441.0

Calcula el número de mensajes enviados por cada usuario por mes.

In [736…

```
# Unir el número de mensajes de textos enviados por cada usuario
users_messages = users.merge(messages, on='user_id', how='outer')
```

In [737…

```
# Comprobar
users_messages.head()
```

Out[737…

	user_id	plan	state	id	message_date	month
0	1000	ultimate	GA	1000_125	2018-12-27	12.0
1	1000	ultimate	GA	1000_160	2018-12-31	12.0
2	1000	ultimate	GA	1000_223	2018-12-31	12.0
3	1000	ultimate	GA	1000_251	2018-12-27	12.0

	user_id	plan	state	id	message_date	month
4	1000	ultimate	GA	1000_255	2018-12-26	12.0

In [738…

```
# Crear una tabla dinámica en user_id y month
messages_per_user_per_month = users_messages.pivot_table(index=['user_id', 'm
messages_per_user_per_month = messages_per_user_per_month.reset_index()

# Cambiar nombre de 'id'
messages_per_user_per_month.rename(columns={'id': 'total_messages_month'}, in
```

In [739…

```
# Comprobar
messages_per_user_per_month.head()
```

Out[739…

	user_id	month	total_messages_month
0	1000	12.0	11
1	1001	8.0	30
2	1001	9.0	44
3	1001	10.0	53
4	1001	11.0	36

Calcula el volumen del trafico de internet usado por cada usuario por mes.

In [740…

```
# Unir la cantidad de internet usado por cada usuario
users_internet = users.merge(internet, on='user_id', how='outer')

# Comprobar
users_internet.head()
```

Out[740…

	user_id	plan	state	id	session_date	mb_used	month
0	1000	ultimate	GA	1000_13	2018-12-29	89.86	12.0
1	1000	ultimate	GA	1000_204	2018-12-31	0.00	12.0
2	1000	ultimate	GA	1000_379	2018-12-28	660.40	12.0
3	1000	ultimate	GA	1000_413	2018-12-26	270.99	12.0
4	1000	ultimate	GA	1000_442	2018-12-27	880.22	12.0

In [741…

```
# Crear una tabla dinámica en user_id y session_month y sumar gb_used
internet_per_user = users_internet.pivot_table(index=['user_id', 'month'], ag
internet_per_user = internet_per_user.reset_index()


# Cambiar nombre de 'gb_used'
internet_per_user.rename(columns={'mb_used': 'mb_used_month'}, inplace=True)

# Convertir megabyte to gigabyte, redondear el total del mes y convertir a nú
internet_per_user['gb_used_month'] = np.ceil((internet_per_user['mb_used_monti
```

In [742…
```
# Comprobar
internet_per_user.head()
```

Out[742…

	user_id	month	mb_used_month	gb_used_month
0	1000	12.0	1901.47	2
1	1001	8.0	6919.15	7
2	1001	9.0	13314.82	14
3	1001	10.0	22330.49	22
4	1001	11.0	18504.30	19

Datos combinados de llamadas, minutos, mensajes, internet para todos los usuarios

Pondremos los datos agregados juntos en un DataFrame para que un registro en él represente lo que un único usuario consumió en un mes determinado.

In [743…
```
# Combina los datos para las llamadas y minutos con base en el user_id y el m
calls_minutes_per_month = minutes_per_user_per_month.merge(calls_per_user, on
```

In [744…
```
len(calls_minutes_per_month)
```

Out[744… 2258

In [745…
```
len(minutes_per_user_per_month)
```

Out[745… 2258

In [746…
```
len(calls_per_user)
```

Out[746… 2258

In [747…
```
# Comprobar
calls_minutes_per_month.head()
```

Out[747…

	user_id	month	total_min_month	number_of_calls
0	1000	12.0	124.0	16
1	1001	8.0	187.0	27
2	1001	9.0	326.0	49
3	1001	10.0	411.0	65
4	1001	11.0	441.0	64

In [748…
```
len(messages_per_user_per_month)
```

Out[748… 1806

In [749…
```
# Revisar valores nulos
messages_per_user_per_month.isna().sum()
```

Out[749…
```
user_id                 0
month                   0
total_messages_month    0
dtype: int64
```

In [750…
```
len(internet_per_user)
```

Out[750…
```
2277
```

In [751…
```
# Revisar valores nulos
internet_per_user.isnull().sum()
```

Out[751…
```
user_id           0
month             0
mb_used_month     0
gb_used_month     0
dtype: int64
```

In [752…
```
# Combina los datos para las mensajes e internet con base en el user_id y el 
messages_internet_per_month = messages_per_user_per_month.merge(internet_per_
```

In [753…
```
len(messages_internet_per_month)
```

Out[753…
```
2292
```

In [754…
```
# Comprobar
messages_internet_per_month.head()
```

Out[754…

	user_id	month	total_messages_month	mb_used_month	gb_used_month
0	1000	12.0	11.0	1901.47	2.0
1	1001	8.0	30.0	6919.15	7.0
2	1001	9.0	44.0	13314.82	14.0
3	1001	10.0	53.0	22330.49	22.0
4	1001	11.0	36.0	18504.30	19.0

In [755…
```
# Revisar valores nulos
messages_internet_per_month.isna().sum()
```

Out[755…
```
user_id                   0
month                     0
total_messages_month    486
mb_used_month            15
gb_used_month            15
dtype: int64
```

Hemos encontrado que tenemos algunos valores nulos. Vamos a utilizar .fillna(0) para rellenar los valores ausentes porque, un valor ausente en este caso significa 0 consumo.

In [756…
```
# Rellenar valores nulos
for null in ['total_messages_month', 'mb_used_month','gb_used_month']:
    messages_internet_per_month[null] = messages_internet_per_month[null].fil
```

In [757…
```
#Comprobar
messages_internet_per_month.isna().sum()
```

Out[757…
```
user_id                0
month                  0
total_messages_month   0
mb_used_month          0
gb_used_month          0
dtype: int64
```

In [758…
```
# Combinar todo
user_consumption_per_month = calls_minutes_per_month.merge(messages_internet_

# Comprobar
user_consumption_per_month.head(10)
```

Out[758…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
0	1000	12.0	124.0	16.0	11.0	1901.47
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82
3	1001	10.0	411.0	65.0	53.0	22330.49
4	1001	11.0	441.0	64.0	36.0	18504.30
5	1001	12.0	422.0	56.0	44.0	19369.18
6	1002	10.0	62.0	11.0	15.0	6552.01
7	1002	11.0	393.0	55.0	32.0	19345.08
8	1002	12.0	393.0	47.0	41.0	14396.24
9	1003	12.0	1135.0	149.0	50.0	27044.14

In [759…
```
user_consumption_per_month = user_consumption_per_month.groupby(['user_id', '
    'total_min_month': 'sum',
    'number_of_calls': 'sum',
    'total_messages_month': 'sum',
    'mb_used_month': 'sum',
    'gb_used_month': 'sum'
}).reset_index()
```

In [760…
```
# Comprobar
user_consumption_per_month.head(10)
```

Out[760…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
0	1000	12.0	124.0	16.0	11.0	1901.47
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
3	1001	10.0	411.0	65.0	53.0	22330.49
4	1001	11.0	441.0	64.0	36.0	18504.30
5	1001	12.0	422.0	56.0	44.0	19369.18
6	1002	10.0	62.0	11.0	15.0	6552.01
7	1002	11.0	393.0	55.0	32.0	19345.08
8	1002	12.0	393.0	47.0	41.0	14396.24
9	1003	12.0	1135.0	149.0	50.0	27044.14

Calcula el ingreso mensual para cada usuario

Para calcular los ingresos mensuales de cada usuario restaremos el límite del paquete gratuito del número total de llamadas, SMS y datos; multiplicaremos el resultado por el valor de tarifa de llamadas; agregaremos el cargo mensual según la tarifa de llamadas.

In [761…
```
plans.head()
```

Out[761…

	messages_included	mb_per_month_included	minutes_included	usd_monthly_pay	usd_per_
0	50	15360	500	20	
1	1000	30720	3000	70	

In [762…
```
user_plan = users[['user_id','plan','state']]
user_plan = user_plan.merge(plans, left_on='plan', right_on='plan_name')
user_plan.head()
```

Out[762…

	user_id	plan	state	messages_included	mb_per_month_included	minutes_included	us
0	1000	ultimate	GA	1000	30720	3000	
1	1006	ultimate	CA	1000	30720	3000	
2	1008	ultimate	FL	1000	30720	3000	
3	1011	ultimate	OH	1000	30720	3000	
4	1013	ultimate	TN	1000	30720	3000	

In [763…
```
user_monthly = (calls_minutes_per_month
                .merge(messages_per_user_per_month, how='outer', on=['user_id',
                .merge(internet_per_user, how='outer', on=['user_id', 'month'])
                .merge(user_plan, on=['user_id'])
               )
```

In [764…
```
user_monthly.head(10)
```

Out[764…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
0	1000	12.0	124.0	16.0	11.0	1901.47
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
3	1001	10.0	411.0	65.0	53.0	22330.49
4	1001	11.0	441.0	64.0	36.0	18504.30
5	1001	12.0	422.0	56.0	44.0	19369.18
6	1002	10.0	62.0	11.0	15.0	6552.01
7	1002	11.0	393.0	55.0	32.0	19345.08
8	1002	12.0	393.0	47.0	41.0	14396.24
9	1003	12.0	1135.0	149.0	50.0	27044.14

In [765…

```
# Revisar valores nulos
user_monthly.isnull().sum()
```

Out[765…

```
user_id                    0
month                      0
total_min_month           35
number_of_calls           35
total_messages_month     487
mb_used_month             16
gb_used_month             16
plan                       0
state                      0
messages_included          0
mb_per_month_included      0
minutes_included           0
usd_monthly_pay            0
usd_per_gb                 0
usd_per_message            0
usd_per_minute             0
plan_name                  0
dtype: int64
```

Tenemos algunos valores nulos para aquellos usuarios que solo usaron 1 o 2 servicios. Es decir, solo mensajes e internet pero no llamadas e internet. Por tanto, reemplazaremos estos valores por 0.

In [766…

```
for null in ['total_min_month', 'number_of_calls','total_messages_month','mb_
    user_monthly[null] = user_monthly[null].fillna(0)
```

In [767…

```
# Comprobar
user_monthly.isnull().sum()
```

Out[767…

```
user_id                  0
month                    0
total_min_month          0
number_of_calls          0
total_messages_month     0
mb_used_month            0
gb_used_month            0
plan                     0
state                    0
messages_included        0
mb_per_month_included    0
minutes_included         0
usd_monthly_pay          0
usd_per_gb               0
usd_per_message          0
usd_per_minute           0
```

```
plan_name                0
dtype: int64
```

Finalmente, calculemos calcular los ingresos mensuales de cada usuario:

In [768…
```
user_monthly['gb_per_month_included']= np.ceil((user_monthly['mb_per_month_in
```

In [769…
```
user_monthly.head()
```

Out[769…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
0	1000	12.0	124.0	16.0	11.0	1901.47
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82
3	1001	10.0	411.0	65.0	53.0	22330.49
4	1001	11.0	441.0	64.0	36.0	18504.30

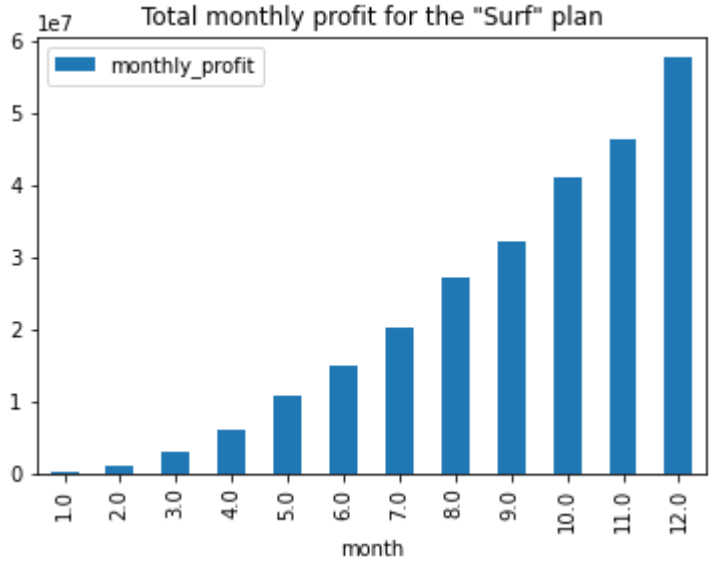
In [770…
```
user_monthly['monthly_profit'] = (((user_monthly['total_min_month']-user_month
                                   + ((user_monthly['total_messages_month
                                   + ((user_monthly['mb_used_month']-np.
                                   + user_monthly['usd_monthly_pay'])
```

In [771…
```
total_profit_month_plan = user_monthly.groupby(['plan', 'month'])['monthly_pr
total_profit_month_plan.head()
```

Out[771…

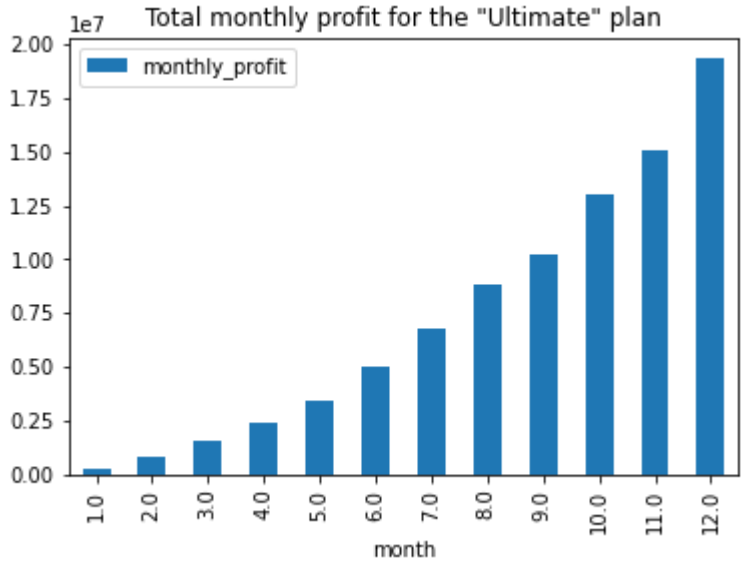
	plan	month	monthly_profit
0	surf	1.0	97197.37
1	surf	2.0	1094772.87
2	surf	3.0	3066095.84
3	surf	4.0	5984843.68
4	surf	5.0	10719946.66

In [772…
```
# Diagrama de barras for Surf
total_profit_month_plan[total_profit_month_plan['plan'] == 'surf'].plot(y='mo
plt.title('Total monthly profit for the "Surf" plan');
```



In [773…

```
# Diagrama de barras for Ultimate
total_profit_month_plan[total_profit_month_plan['plan'] == 'ultimate'].plot(y
plt.title('Total monthly profit for the "Ultimate" plan');
```



Conclusion intermediaria

1. Durante el mes de diciembre, el Plan Surf y Ultimate tienen la mayor parte de sus ganancias.
2. Surf y Ultimate presentan un comportamiento exponencial a lo largo del año.
3. Aunque Surf y Ultimate tienen el mismo comportamiento mes a mes, podemos deducir que Ultimate tiene más ganancias.

Estudia el comportamiento del usuario

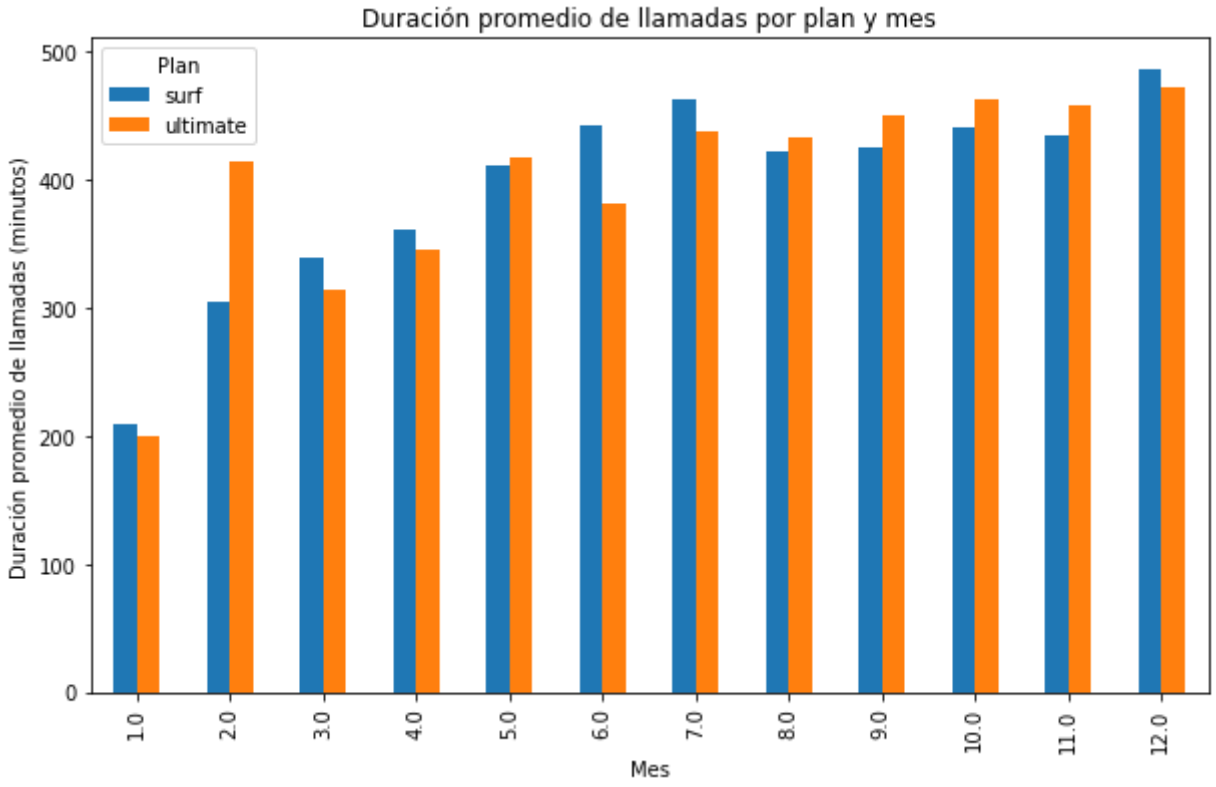
Calcularemos algunas estadísticas descriptivas para estudiar el comportamiento del usuario y muestren un panorama general captado por los datos.

Llamadas

In [774…

```
# Compara la duración promedio de llamadas por cada plan por cada mes.
mean_calls_duration = user_monthly.pivot_table(index='month', columns='plan',
```

```
# Dibuja una gráfica de barras para visualizarlo
mean_calls_duration.plot(kind='bar', figsize=(10, 6))
plt.title('Duración promedio de llamadas por plan y mes')
plt.xlabel('Mes')
plt.ylabel('Duración promedio de llamadas (minutos)')
plt.legend(title='Plan')
plt.show()
```



Conclusion intermediaria

1. Encontramos que los clientes del plan Surf hacen más llamadas que los clientes con Ultimate.
2. En Febrero, los clientes del plan Ultimate alcanzan a utilzar casi 500 minutos de los 3000 que brinda el plan.
3. El plan Surf parece tener más variación en la duración promedio de las llamadas que el plan Ultimate.
4. La mayor duración promedio de llamadas para ambos planes se registró en diciembre de 2018.

In [775…

```
# Compara el número de minutos que los usuarios de cada plan requieren cada m
surf_user_monthly = user_monthly[user_monthly['plan'] == 'surf']
ultimate_user_monthly = user_monthly[user_monthly['plan'] == 'ultimate']
```

In [776…

```
# Comprobar para clientes Surf
surf_user_monthly.head()
```

Out[776…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82
3	1001	10.0	411.0	65.0	53.0	22330.49

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
4	1001	11.0	441.0	64.0	36.0	18504.30
5	1001	12.0	422.0	56.0	44.0	19369.18

In [777…

```
# Comprobar para clientes Ultimate
ultimate_user_monthly.head()
```

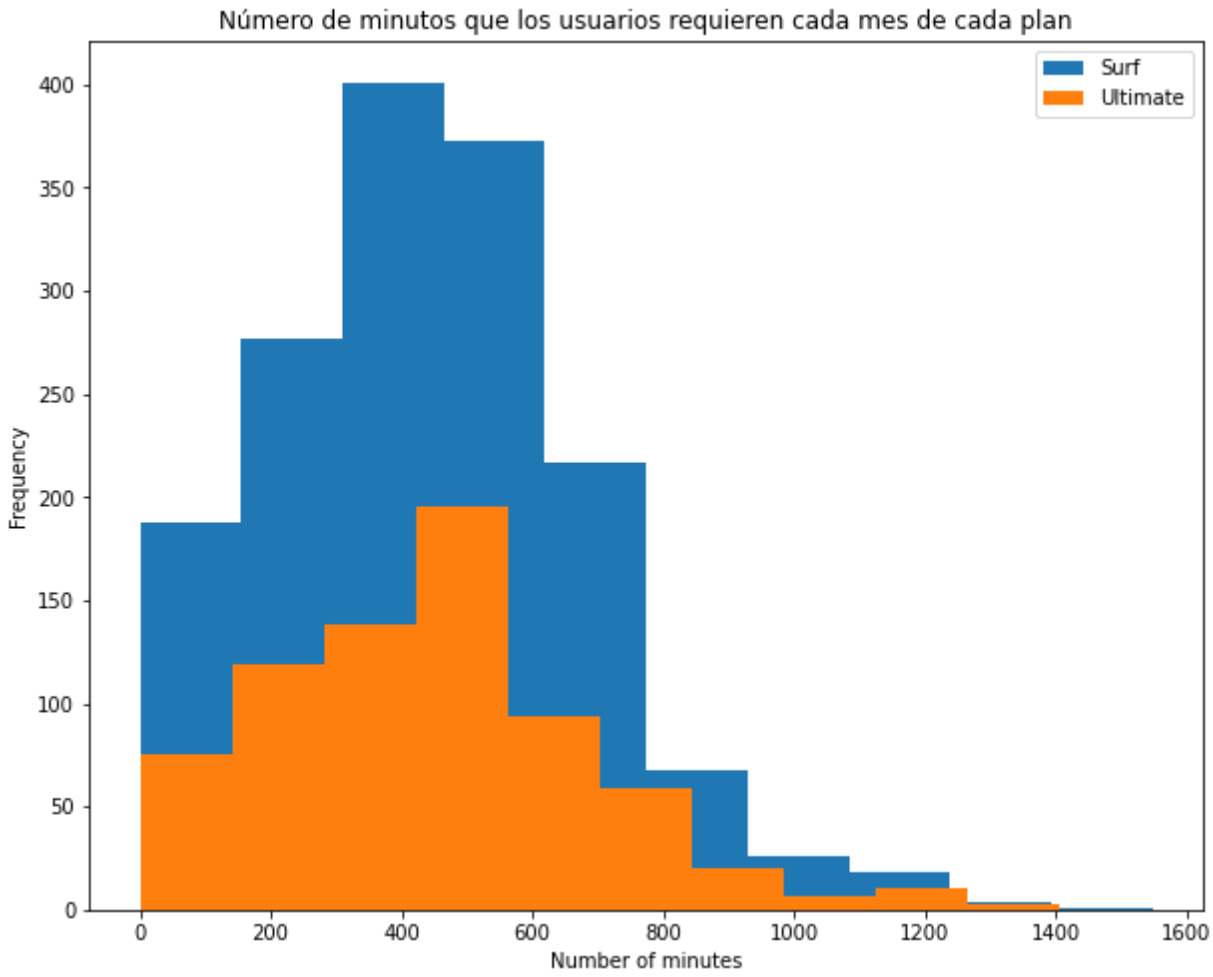
Out[777…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_mont
0	1000	12.0	124.0	16.0	11.0	1901.4
19	1006	11.0	10.0	2.0	15.0	2068.3
20	1006	12.0	61.0	9.0	139.0	32118.8
26	1008	10.0	493.0	71.0	21.0	17106.9
27	1008	11.0	459.0	63.0	37.0	23676.7

In [778…

```
# Trazar histogramas para comparar la cantidad de minutos que los usuarios de
surf_user_monthly['total_min_month'].plot(kind='hist',figsize=(10,8))
ultimate_user_monthly['total_min_month'].plot(kind='hist',figsize=(10,8))
plt.title('Número de minutos que los usuarios requieren cada mes de cada plan
plt.legend(['Surf', 'Ultimate'])
plt.xlabel('Cantidad de minutes')
plt.xlabel('Number of minutes')

plt.show()
```



Conclusion intermedia

1. Tanto Ultimate como Surf alcanzaron su punto máximo durante unos 500 minutos, con una distribución sesgada hacia la izquierda.
2. El límite de duración de la llamada lo superan un gran número de usuarios de Surf, pero ningún usuario de Ultimate.

Recordemos que Surf ofrece 500 minutos al mes y Ultimate, 3000 minutos al mes.

Vamos a calcular la media y la variable de la duración de la llamada para concluir si los usuarios de diferentes planes muestran comportamientos distintos para sus llamadas.

In [779…

```
# Calcula la media y la varianza de la duración mensual de llamadas para Surf

print('Duración promedio de las llamadas mensuales en minutos del plan "Surf"
print('Desviación estándar de la duración de las llamadas mensuales en minuto
```

```
Duración promedio de las llamadas mensuales en minutos del plan "Surf": 440
Desviación estándar de la duración de las llamadas mensuales en minutos del pl
an "Surf": 240
```

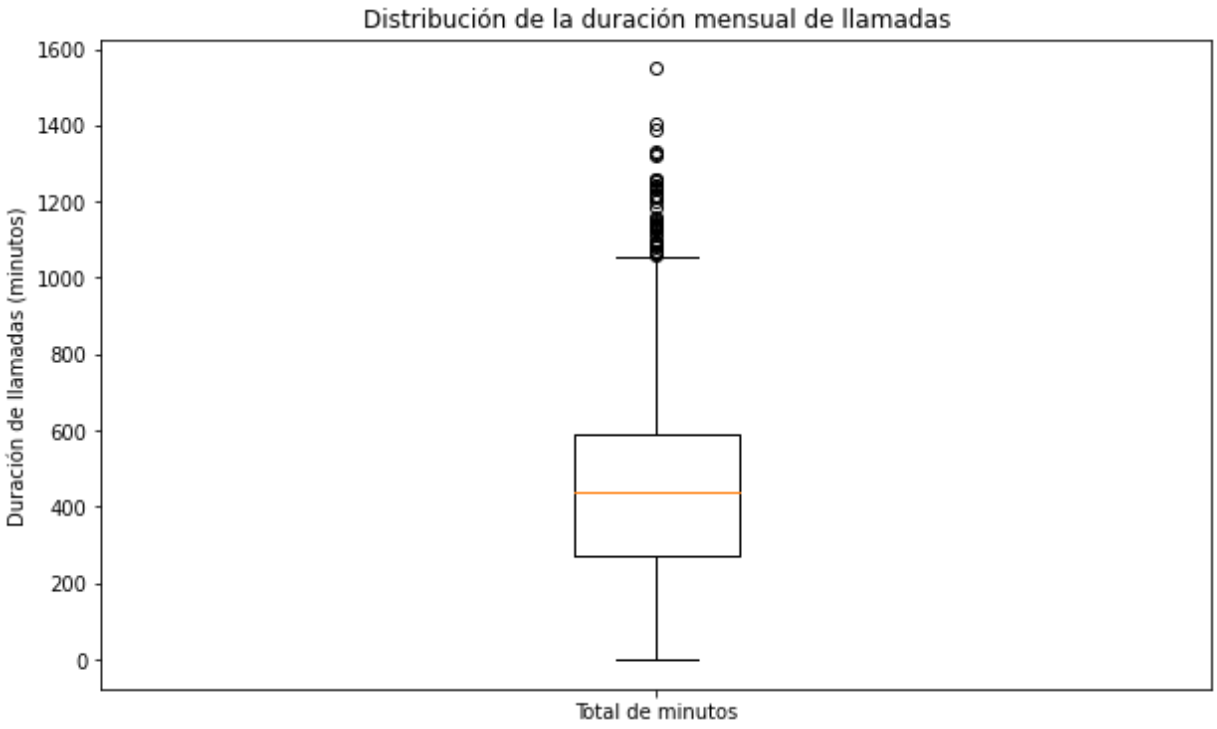
In [780…

```
# Calcula la media y la varianza de la duración mensual de llamadas para Ulti
print('Duración promedio de las llamadas mensuales en minutos del plan "Ultim
print('Desviación estándar de la duración de las llamadas mensuales en minuto
```

```
Duración promedio de las llamadas mensuales en minutos del plan "Ultimate": 44
2
Desviación estándar de la duración de las llamadas mensuales en minutos del pl
an "Ultimate": 247
```

In [781…

```
# Traza un diagrama de caja para visualizar la distribución de la duración me
plt.figure(figsize=(10, 6))
plt.boxplot(user_monthly['total_min_month'])
plt.title('Distribución de la duración mensual de llamadas')
plt.ylabel('Duración de llamadas (minutos)')
plt.xticks([1], ['Total de minutos'])
plt.show()
```



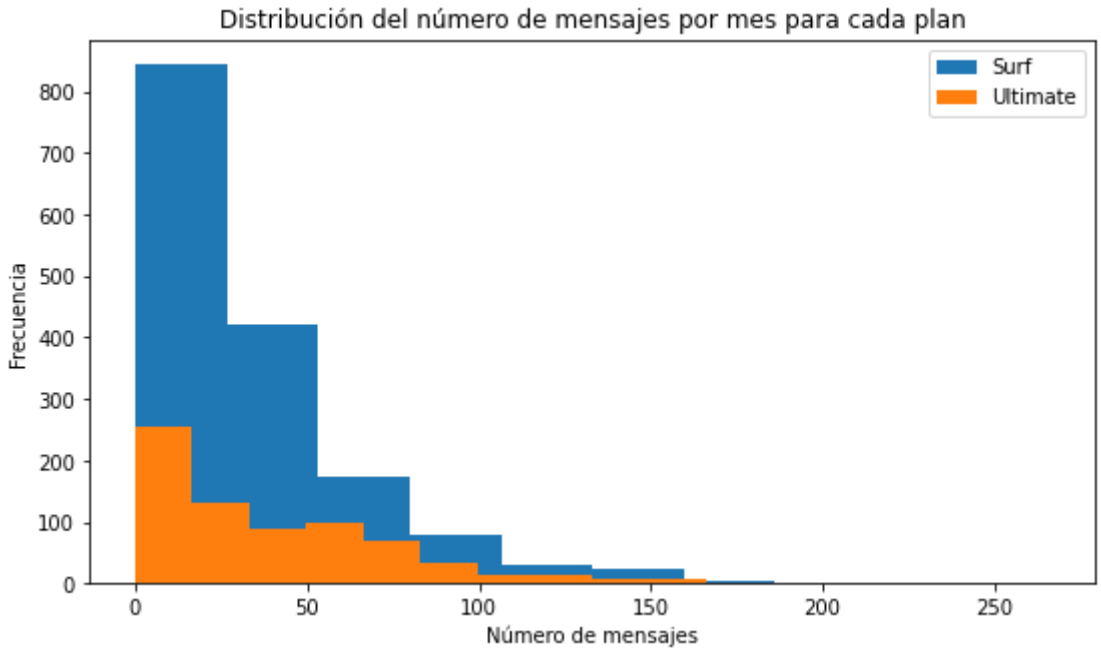
Conclusiones:

1. La duración media de las llamadas mensuales es casi la misma (alrededor de 430-450 minutos por mes) para ambos planes, lo que es una señal de una distribución cercana a la normal. Las desviaciones estándar también son bastante cercanas (alrededor de 220-230 minutos).
2. Del diagrama de cajas podemos ver que existen varios puntos por fuera del extremo superior; sin embargo se puede inferir que son usuarios del plan Ultimate, con hasta 3000 minutos al mes.
3. Del diagrama de cajas podemos ver que la mediana es similar a la media que hayamos anteriormente.

Mensajes

In [782…

```
# Comprara el número de mensajes que tienden a enviar cada mes los usuarios d
plt.figure(figsize=(10, 6))
surf_user_monthly['total_messages_month'].plot(kind='hist',figsize=(9,5))
ultimate_user_monthly['total_messages_month'].plot(kind='hist',figsize=(9,5))
plt.title('Distribución del número de mensajes por mes para cada plan')
plt.xlabel('Número de mensajes')
plt.ylabel('Frecuencia')
plt.legend(['Surf', 'Ultimate'])
plt.show()
```



In [783…
```
# Calcula la media y la varianza de mensajes enviados con Surf.

print('Número promedio mensual de mensajes enviados del plan "Surf": {:.0f}'.
print('Desviación estándar mensual de mensajes enviados del plan "Surf": {:.0
```

```
Número promedio mensual de mensajes enviados del plan "Surf": 31
Desviación estándar mensual de mensajes enviados del plan "Surf": 34
```

In [784…
```
# Calcula la media y la varianza de mensajes enviados con Ultimate.

print('Número promedio mensual de mensajes enviados del plan "Surf": {:.0f}'.
print('Desviación estándar mensual de mensajes enviados del plan "Surf": {:.0
```

```
Número promedio mensual de mensajes enviados del plan "Surf": 38
Desviación estándar mensual de mensajes enviados del plan "Surf": 35
```

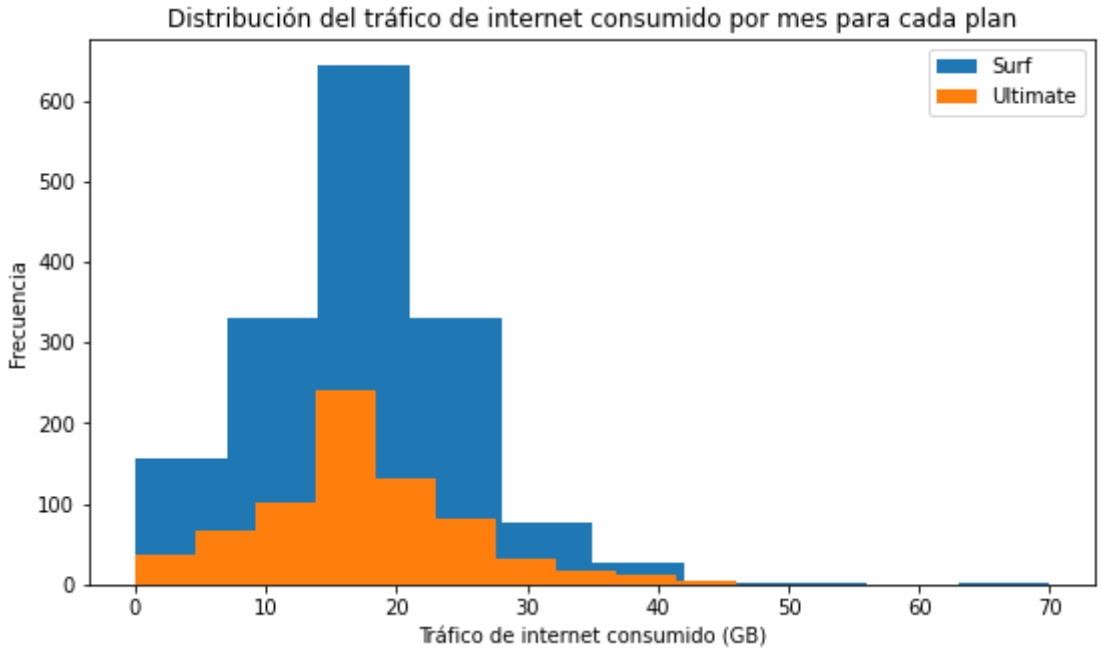
Conclusiones

1. Encontramos que el promedio mensual de mensajes enviados del plan Surf fueron 30; cabe recordar que Surf ofrece 50 SMS al mes, lo cual indica que la mayoria de usuarios intentan enviar mensajes dentro del limite del plan.
2. En el caso de Ultimate, el promedio mensual de mensajes enviados fueron 37; cabe recordar que Ultimate ofrece 1,000 SMS al mes, lo cual indica que la mayoria de usuarios no están enviando mensajes. Aquí es donde la empresa pierde ingresos extra ya que no habría recargo en este servicio. Podríamos recomendar a la compañia revisar esta cantidad.
3. Ambos planes muestran el mismo panorama: estas distribuciones están sesgadas positivamente.
4. Ambos planes muestran una desviación estándar bastante grande, casi igual a la media.

Internet

In [785…
```
# Compara la cantidad de tráfico de internet consumido por usuarios por plan
plt.figure(figsize=(10, 6))
surf_user_monthly['gb_used_month'].plot(kind='hist',figsize=(9,5))
ultimate_user_monthly['gb_used_month'].plot(kind='hist',figsize=(9,5))
plt.title('Distribución del tráfico de internet consumido por mes para cada p
```

```
plt.xlabel('Tráfico de internet consumido (GB)')
plt.ylabel('Frecuencia')
plt.legend(['Surf', 'Ultimate'])
plt.show()
```



In [786…

```
# Calcula la media y la varianza del tráfico de internet consumido por usuari

print('Número promedio mensual de tráfico de internet del plan "Surf": {:.0f}
print('Desviación estándar mensual de mensajes enviados del plan "Surf": {:.0
```

```
Número promedio mensual de tráfico de internet del plan "Surf": 17
Desviación estándar mensual de mensajes enviados del plan "Surf": 8
```

In [787…

```
# Calcula la media y la varianza del tráfico de internet consumido por usuari

print('Número promedio mensual de tráfico de internet del plan "Ultimate": {:
print('Desviación estándar mensual de mensajes enviados del plan "Ultimate":
```

```
Número promedio mensual de tráfico de internet del plan "Ultimate": 17
Desviación estándar mensual de mensajes enviados del plan "Ultimate": 8
```

Conclusiones

1. Encontramos que el promedio mensual del tráfico de internet consumido por usuarios con el plan Surf fueron 17; cabe recordar que Surf ofrece 15 GB al mes, lo cual indica que la mayoria de usuarios usan más de indica que los usuarios se están pasandose del plan. Aquí es donde la empresa genera ingresos extra ya que hay recargo en este servicio.

2. En el caso de Ultimate, el promedio mensual de tráfico de internet consumido por usuarios fueron 19, lo cual es muy similar a Surf. Cabe recordar que Ultimate ofrece 13GB al mes, lo cual indica que la mayoria de usuarios están pasandose del plan. Aquí es donde la empresa genera ingresos extra ya que hay recargo en este servicio.

3. Ambos planes muestran el mismo panorama: estas distribuciones están sesgadas hacia la izquierda.

4. Ambos planes muestran una desviación estándar normal de 8 GB.

Ingreso

In [788…

```
user_monthly.head()
```

Out[788…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
0	1000	12.0	124.0	16.0	11.0	1901.47
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82
3	1001	10.0	411.0	65.0	53.0	22330.49
4	1001	11.0	441.0	64.0	36.0	18504.30

In [789…

```
#Calculemos los ingresos mensuales de cada usuario.

def calculate_monthly_revenue(row):

    plan_monthly_charge = row.usd_monthly_pay

    # Calcular los ingresos por llamadas, si el usuario excedió el límite cub
    extra_call_mins = row.total_min_month - row.minutes_included
    if extra_call_mins > 0:
        call_revenue = extra_call_mins * row.usd_per_minute
    else:
        call_revenue = 0

    # Calcular los ingresos por mensajes, si el usuario excedió el límite cub
    extra_messages = row.total_messages_month - row.messages_included
    if extra_messages > 0:
        message_revenue = extra_messages * row.usd_per_message
    else:
        message_revenue = 0

    # Calcular los ingresos por el uso de Internet, si el usuario excedió el
    extra_internet_gb = (row.gb_used_month - row.gb_per_month_included)
    if extra_internet_gb > 0:
        internet_revenue = extra_internet_gb * row.usd_per_gb
    else:
        internet_revenue = 0

    monthly_revenue = plan_monthly_charge + call_revenue + message_revenue +

    return monthly_revenue
```

In [790…

```
# Calcular los ingresos mensuales de cada usuario.
user_monthly['usd_monthly_revenue'] = user_monthly.apply(calculate_monthly_re
```

In [791…

```
# Compare los ingresos promedio por cada plan por cada mes distinto
mean_revenue = user_monthly.pivot_table(index='month', columns='plan', aggfun
mean_revenue
```

Out[791…

plan month	surf	ultimate
1.0	20.000000	70.000000
2.0	34.456667	70.000000
3.0	45.996087	74.666667
4.0	40.832400	73.000000
5.0	47.834286	70.724138
6.0	49.455052	71.638298
7.0	62.904628	71.898305
8.0	64.118765	72.859155
9.0	58.444845	72.034884
10.0	65.618059	72.311321
11.0	58.185548	71.708661
12.0	70.783648	73.291391

In [792…

```
# Calcular la media y la varianza de los ingresos mensuales
monthly_revenue_stats = user_monthly.pivot_table(index='plan', values='usd_mo
monthly_revenue_stats.columns = ['mean_monthly_revenue', 'var_monthly_revenue
monthly_revenue_stats
```

Out[792…

plan	mean_monthly_revenue	var_monthly_revenue	std_monthly_revenue	median_monthly_
surf	60.899053	3075.367902	55.456000	
ultimate	72.313889	129.848486	11.395108	

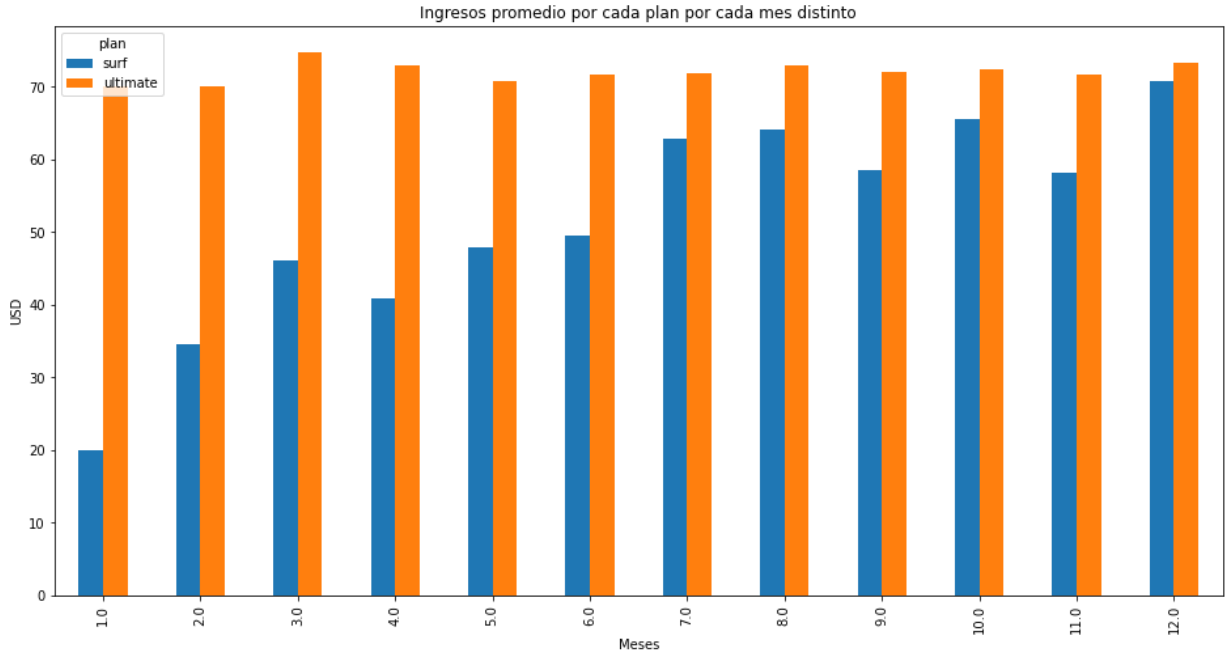
Observaciones

1. En promedio, los clientes del plan Ultimate generan mayores ingresos por mes en comparación con los del plan Surf.
2. La variación en los ingresos mensuales de Surf es significativamente mayor que la de Ultimate, lo que indica que hay más variabilidad en los ingresos mensuales del plan Surf en comparación con el plan Ultimate.
3. La desviación estándar de los ingresos mensuales de Surf es 51.16, que es mayor que la de Ultimate (12.07). Esto significa que los ingresos del plan Surf están más dispersos en comparación con los del plan Ultimate.
4. El ingreso mensual medio de Ultimate es 70, mientras que el de Surf es 40. Esto indica que al menos la mitad de los clientes del plan Ultimate generan ingresos de 70 por mes, mientras que la mitad de los clientes del plan Surf generan ingresos de 40 o menos por mes.

In [793…

```
# Trazar un diagrama de barras para visualizar 'mean_revenue'
mean_revenue.plot(kind='bar', figsize=(16,8))
plt.title('Ingresos promedio por cada plan por cada mes distinto')
plt.ylabel('USD')
```

```
plt.xlabel('Meses')
plt.show()
```



In [794…

```
mean_revenue.describe()
```

Out[794…

plan	surf	ultimate
count	12.000000	12.000000
mean	51.552499	72.011068
std	14.764229	1.364068
min	20.000000	70.000000
25%	44.705165	71.409758
50%	53.820300	71.966594
75%	63.208162	72.894366
max	70.783648	74.666667

Conclusiones:

1. La mediana del plan Surf es alrededor del precio del plan. Sin embargo, para Ultimate observamos que usuarios tienden a pagar recargo.
2. Surf tiene desviación estandar grande en comparación con Ultimate.
3. Se observa que usuarios han pagado recargo en ambos planes. Lo máximo que un usuario ha pagado para Surf es $65 aproximadamente y 74 aproximadamente con Ultimate.
4. Los ingresos promedio del plan Ultimate son consistentemente más altos que los del plan Surf en todos los meses.
5. En general, estas estadísticas sugieren que el plan Ultimate es más rentable para Megaline, ya que los clientes de este plan generan mayores ingresos en promedio y los ingresos son más consistentes en comparación con el plan Surf.

Probar las hipótesis estadísticas.

Hipótesis 1:

Probemos la hipótesis de que los ingresos promedio de los usuarios de los planes de llamadas Ultimate y Surf difieren. Para probar la hipótesis, formulemos la hipótesis nula y la alternativa.

Hipótesis nula: los ingresos medios de los usuarios de los planes de llamadas Ultimate y Surf son iguales.

Hipótesis alternativa: Los ingresos promedio de los usuarios del plan de llamadas Surf son diferentes a los del plan de llamadas Ultimate.

In [795…
```
user_monthly.head()
```

Out[795…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
0	1000	12.0	124.0	16.0	11.0	1901.47
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82
3	1001	10.0	411.0	65.0	53.0	22330.49
4	1001	11.0	441.0	64.0	36.0	18504.30

In [796…
```
# Ingresos medios de los usuarios del plan Surf
surf_mean_user_revenue = user_monthly[user_monthly['plan'] == 'surf'].groupby
surf_mean_user_revenue.head()
```

Out[796…
```
user_id
1001     50.018000
1002     33.333333
1003    159.050000
1004     76.250000
1005     40.150000
Name: usd_monthly_revenue, dtype: float64
```

In [797…
```
# Ingresos medios de los usuarios del plan Ultimate
ultimate_mean_user_revenue = user_monthly[user_monthly['plan'] == 'ultimate']
ultimate_mean_user_revenue.head()
```

Out[797…
```
user_id
1000    70.0
1006    77.0
1008    70.0
1011    70.0
1013    70.0
Name: usd_monthly_revenue, dtype: float64
```

In [798…
```
# Prueba las hipótesis 1
alpha = 0.05

results = st.ttest_ind(surf_mean_user_revenue, ultimate_mean_user_revenue)

print('p-value:', results.pvalue)

if (results.pvalue < alpha):
    print("Rechazamos la hipótesis nula")
```

```
else:
    print("No podemos rechazar la hipótesis nula")
```

```
p-value: 0.0001419783641714099
Rechazamos la hipótesis nula
```

Conclusión: En esta prueba se rechaza la hipótesis nula, lo que implica que los ingresos promedio de los usuarios de los planes telefónicos Ultimate y Surf no son iguales.

Hipótesis 2:

Comprobar la hipótesis de que el ingreso promedio de los usuarios del área NY-NJ es diferente al de los usuarios de otras regiones.

Hipótesis nula: Los ingresos promedio de los usuarios del área NY-NJ es igual al de los usuarios de otras regiones.

Hipótesis alternativa: Los ingresos promedio de los usuarios del área NY-NJ es diferente al de los usuarios de otras regiones.

In [799…
```
user_monthly['state'].unique()
```

Out[799…
```
array(['GA', 'WA', 'NV', 'OK', 'TX', 'CA', 'MI', 'FL', 'OH', 'WI', 'TN',
       'PA', 'WV', 'IN', 'IA', 'MD', 'AL', 'SC', 'CO', 'NM', 'NH', 'AZ',
       'NY', 'AR', 'MA', 'HI', 'CT', 'LA', 'NC', 'VA', 'IL', 'KS', 'UT'],
      dtype=object)
```

NY-NJ

In [800…
```
ny_nj_user_monthly = user_monthly[user_monthly['state'].str.contains('NY')]
ny_nj_user_monthly.head()
```

Out[800…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_mo
355	1076	7.0	30.0	4.0	9.0	113
356	1076	8.0	405.0	54.0	156.0	25517
357	1076	9.0	373.0	53.0	145.0	23134
358	1076	10.0	326.0	50.0	159.0	21310
359	1076	11.0	390.0	57.0	139.0	21318

In [801…
```
# Ingresos promedio de usuarios que se encuentran en el área de NY-NJ
ny_nj_mean_user_revenue = ny_nj_user_monthly.groupby('user_id')['usd_monthly_
ny_nj_mean_user_revenue.head()
```

Out[801…
```
user_id
1076     74.071667
1098    148.448000
1192     22.250000
1302     73.152500
1377     82.116667
Name: usd_monthly_revenue, dtype: float64
```

OTROS ESTADOS

In [802…
```
# Ingresos promedio de usuarios que no se encuentran en el área de Nueva York
other_states_user_monthly = user_monthly[~user_monthly['state'].str.contains(
other_states_user_monthly.head()
```

Out[802…

	user_id	month	total_min_month	number_of_calls	total_messages_month	mb_used_month
0	1000	12.0	124.0	16.0	11.0	1901.47
1	1001	8.0	187.0	27.0	30.0	6919.15
2	1001	9.0	326.0	49.0	44.0	13314.82
3	1001	10.0	411.0	65.0	53.0	22330.49
4	1001	11.0	441.0	64.0	36.0	18504.30

In [803…

```
# Ingresos promedio de usuarios que no se encuentran en el área de NY-NJ
other_states_mean_user_revenue = other_states_user_monthly.groupby('user_id')
other_states_mean_user_revenue.head()
```

Out[803…

```
user_id
1000     70.000000
1001     50.018000
1002     33.333333
1003    159.050000
1004     76.250000
Name: usd_monthly_revenue, dtype: float64
```

In [804…

```
# Prueba las hipótesis 2
alpha = 0.05

results = st.ttest_ind(ny_nj_mean_user_revenue, other_states_mean_user_revenu

print('p-value:', results.pvalue)

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

```
p-value: 0.6200856765260465
No podemos rechazar la hipótesis nula
```

Conclusión:

No podemos rechazar la hipótesis nula, lo que significa que el ingreso promedio de los usuarios del área NY-NJ es igual al de los usuarios de los otros estados.

Conclusión general

El objetivo de este proyecto es analizar el comportamiento de los clientes y determinar qué plan prepago genera más ingresos. Los resultados del análisis son:

1. Ambos planes muestran una tendencia general de aumento de la duración promedio de las llamadas de enero a diciembre, lo que podría indicar un efecto estacional o una tendencia en el comportamiento de los clientes.

2. La cantidad promedio de mensajes enviados por los usuarios del plan Surf y Ultimate es relativamente baja, y la mayoría de los meses tienen un promedio de menos de 40 mensajes por mes.

3. Enero parece ser el mes con menor uso de Internet para ambos planes.

4. El plan Ultimate es más rentable para Megaline, ya que los clientes del plan Ultimate probablemente están usando más funciones y servicios adicionales que ofrece el plan, como límites de datos más altos.

5. La mayoría de los usuarios de Surf tienden a superar su límite de uso mensual. Sin embargo, los ingresos promedio del plan Ultimate son consistentemente más altos que los del plan Surf en todos los meses.

6. Los ingresos promedio de los usuarios de los planes telefónicos Ultimate y Surf no son iguales.

7. Los ingresos promedio de los usuarios en la región NY-NJ son iguales a los ingresos de los usuarios de otros estados.

8. En promedio, los clientes del plan Ultimate generan mayores ingresos por mes en comparación con los del plan Surf.