8b0c61e8-9975-4818-84f4-e4ee6e4efb3d-Copy1

November 20, 2024

1 ¿Cuál es la mejor tarifa?

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

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

[Te proporcionamos algunos comentarios para orientarte mientras completas este proyecto. Pero debes asegurarte de eliminar todos los comentarios entre corchetes antes de entregar tu proyecto.]

[Antes de sumergirte en el análisis de datos, explica por tu propia cuenta el propósito del proyecto y las acciones que planeas realizar.]

[Ten en cuenta que estudiar, modificar y analizar datos es un proceso iterativo. Es normal volver a los pasos anteriores y corregirlos/ampliarlos para permitir nuevos pasos.]

1.1 Inicialización

[73]:
```
import pandas as pd
import numpy as np
import scipy as stats
import scipy.stats as st
import matplotlib.pyplot as plt
import seaborn as sns
import math as mt# Cargar todas las librerías
```

1.2 Cargar datos

[74]:
```
# Carga los archivos de datos en diferentes DataFrames
df_calls = pd.read_csv("/datasets/megaline_calls.csv")

df_internet=pd.read_csv("/datasets/megaline_internet.csv")

df_messages=pd.read_csv("/datasets/megaline_messages.csv")
```

```
df_plans=pd.read_csv("/datasets/megaline_plans.csv")

df_users=pd.read_csv("/datasets/megaline_users.csv")
```

1.3 Preparar los datos

[Los datos para este proyecto se dividen en varias tablas. Explora cada una para tener una comprensión inicial de los datos. Si es necesario, haz las correcciones requeridas en cada tabla.]

1.4 Tarifas

[75]:
```
df_plans.info()# Imprime la información general/resumida sobre el DataFrame de␣
↪las tarifas
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 8 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   messages_included      2 non-null      int64
 1   mb_per_month_included  2 non-null      int64
 2   minutes_included       2 non-null      int64
 3   usd_monthly_pay        2 non-null      int64
 4   usd_per_gb             2 non-null      int64
 5   usd_per_message        2 non-null      float64
 6   usd_per_minute         2 non-null      float64
 7   plan_name              2 non-null      object
dtypes: float64(2), int64(5), object(1)
memory usage: 256.0+ bytes
```

[76]:
```
print(df_plans.head())# Imprime una muestra de los datos para las tarifas
```

```
   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
```

[Describe lo que ves y observas en la información general y en la muestra de datos impresa para el precio de datos anterior. ¿Hay algún problema (tipos de datos no adecuados, datos ausentes, etc.) que pudieran necesitar investigación y cambios adicionales? ¿Cómo se puede arreglar?]

No considero que haya algo que corregir es un dataset bien detalledo, los tipos de datos son acordes, no tiene ni duplicados ni valores ausentes.

1.5 Corregir datos

[Corrige los problemas obvios con los datos basándote en las observaciones iniciales.]

[77]:
```
df_plans['gb_per_month_included'] = df_plans['mb_per_month_included'] / 1024
df_plans.drop('mb_per_month_included', axis=1, inplace=True)
print(df_plans.head())
```

```
   messages_included  minutes_included  usd_monthly_pay  usd_per_gb  \
0                 50               500               20          10
1               1000              3000               70           7

   usd_per_message  usd_per_minute plan_name  gb_per_month_included
0             0.03            0.03      surf                   15.0
1             0.01            0.01  ultimate                   30.0
```

1.6 Enriquecer los datos

[Agrega factores adicionales a los datos si crees que pudieran ser útiles.]

Expresar los valores en GB facilita la interpretación de los resultados y la comparación entre diferentes planes, tambien al convertir a GB, se trabaja con valores precisos y consistentes, evitando errores de redondeo o inconsistencias al comparar valores en diferentes unidades.

1.7 Usuarios/as

[78]:
```
df_users.info()# Imprime la información general/resumida sobre el DataFrame de␣
↪usuarios
```

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

[79]:
```
print(df_users.head())# Imprime una muestra de datos para usuarios
```

```
   user_id first_name  last_name  age                                  city  \
0     1000   Anamaria      Bauer   45  Atlanta-Sandy Springs-Roswell, GA MSA
```

```
1     1001     Mickey  Wilkerson   28        Seattle-Tacoma-Bellevue, WA MSA
2     1002     Carlee    Hoffman   36   Las Vegas-Henderson-Paradise, NV MSA
3     1003   Reynaldo    Jenkins   52                           Tulsa, OK MSA
4     1004    Leonila   Thompson   40        Seattle-Tacoma-Bellevue, WA MSA

     reg_date      plan churn_date
0  2018-12-24  ultimate        NaN
1  2018-08-13      surf        NaN
2  2018-10-21      surf        NaN
3  2018-01-28      surf        NaN
4  2018-05-23      surf        NaN
```

[Describe lo que ves y observas en la información general y en la muestra de datos impresa para el precio de datos anterior. ¿Hay algún problema (tipos de datos no adecuados, datos ausentes, etc.) que pudieran necesitar investigación y cambios adicionales? ¿Cómo se puede arreglar?]

En este caso, observo valores ausentes "NaN" y que la columna de fecha no tiene un formato adecuado

1.7.1 Corregir los datos

[Corrige los problemas obvios con los datos basándote en las observaciones iniciales.]

Sustitui los valores ausentes

[80]:
```
df_users["churn_date"].fillna(" ", inplace=True)
print(df_users.head())
```

```
   user_id first_name  last_name  age                                   city  \
0     1000   Anamaria      Bauer   45  Atlanta-Sandy Springs-Roswell, GA MSA
1     1001     Mickey  Wilkerson   28        Seattle-Tacoma-Bellevue, WA MSA
2     1002     Carlee    Hoffman   36   Las Vegas-Henderson-Paradise, NV MSA
3     1003   Reynaldo    Jenkins   52                           Tulsa, OK MSA
4     1004    Leonila   Thompson   40        Seattle-Tacoma-Bellevue, WA MSA

     reg_date      plan churn_date
0  2018-12-24  ultimate
1  2018-08-13      surf
2  2018-10-21      surf
3  2018-01-28      surf
4  2018-05-23      surf
```

1.7.2 Enriquecer los datos

[Agrega factores adicionales a los datos si crees que pudieran ser útiles.]

Puede ser util que cada columna tenga un tipo de dato correspondiente a su contenido

[81]:
```
df_users['reg_date'] = pd.to_datetime(df_users['reg_date'], format='%Y-%m-%d')
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    datetime64[ns]
 6   plan        500 non-null    object
 7   churn_date  500 non-null    object
dtypes: datetime64[ns](1), int64(2), object(5)
memory usage: 31.4+ KB
```

1.8 Llamadas

[82]:
```
df_calls.info()# Imprime la información general/resumida sobre el DataFrame de las llamadas
```

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

[83]:
```
print(df_calls.sample(10))# Imprime una muestra de datos para las llamadas
```

```
                id  user_id   call_date  duration
90001    1326_652     1326  2018-08-13      0.00
75408    1267_130     1267  2018-12-13      8.56
108439  1382_1803     1382  2018-08-12     13.52
127458   1456_346     1456  2018-09-26      0.00
74636    1263_620     1263  2018-09-21      4.16
36711    1139_373     1139  2018-12-17     17.83
50393    1181_790     1181  2018-06-27      0.00
87142    1320_811     1320  2018-07-20      0.00
6254      1030_28     1030  2018-10-11      0.00
128666    1462_79     1462  2018-10-19     13.71
```

[Describe lo que ves y observas en la información general y en la muestra de datos impresa para el

precio de datos anterior. ¿Hay algún problema (tipos de datos no adecuados, datos ausentes, etc.) que pudieran necesitar investigación y cambios adicionales? ¿Cómo se puede arreglar?]

De igual forma que en el anterior veo para corregir la columna de fecha, el resto de los datos condidero que estan ordenados y acordes

1.8.1 Corregir los datos

[Corrige los problemas obvios con los datos basándote en las observaciones iniciales.]

[84]:
```
df_calls['call_date'] = pd.to_datetime(df_calls['call_date'], format='%Y-%m-%d')
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  datetime64[ns]
 3   duration   137735 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 4.2+ MB
```

1.8.2 Enriquecer los datos

[Agrega factores adicionales a los datos si crees que pudieran ser útiles.]

[85]:
```
df_enriquecido_calls = df_calls.merge(df_users, on='user_id', how='left')
print(df_enriquecido_calls.sample(5))
```

```
               id  user_id  call_date  duration first_name  last_name  age  \
18978    1072_479     1072 2018-09-17      7.37    Seymour     Spence   66
116062   1406_288     1406 2018-08-29      3.39      Noble  Jefferson   54
21157   1077_1124     1077 2018-07-31      7.73       Chau    Webster   23
23022     1084_69     1084 2018-11-19      0.00      Wiley   Mckinney   21
80401      1291_1     1291 2018-09-01      9.43    Angeles      Mejia   65

                                              city   reg_date      plan  \
18978              Dallas-Fort Worth-Arlington, TX MSA 2018-07-12      surf
116062          San Diego-Chula Vista-Carlsbad, CA MSA 2018-03-04      surf
21157            Charlotte-Concord-Gastonia, NC-SC MSA 2018-01-14  ultimate
23022   Miami-Fort Lauderdale-West Palm Beach, FL MSA 2018-06-04      surf
80401             Indianapolis-Carmel-Anderson, IN MSA 2018-01-29      surf

        churn_date
18978
116062
```

```
21157
23022   2018-11-11
80401
```

[86]:
```
rangos_edad = {"joven": 0, "adulto": 0, "adulto_mayor": 0}

for age in df_enriquecido_calls["age"]:
    try:
        int_age = int(age)
        if int_age <= 25:
            rangos_edad["joven"] += 1
        elif int_age <= 59:
            rangos_edad["adulto"] += 1
        else:
            rangos_edad["adulto_mayor"] += 1
    except ValueError:

        print(f"Valor de edad no válido: {age}")


for result in rangos_edad:
    print(result, '-', rangos_edad[result])
```

```
joven - 22703
adulto - 76269
adulto_mayor - 38763
```

1.9 Mensajes

[87]:
```
df_messages.info()# Imprime la información general/resumida sobre el DataFrame␣
↪de los mensajes
```

```
<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
```

[88]:
```
print(df_messages.sample(10))# Imprime una muestra de datos para los mensajes
```

```
              id  user_id message_date
22299   1133_301     1133   2018-11-20
5271     1052_28     1052   2018-12-12
```

```
11901   1079_300     1079   2018-08-08
41197   1264_329     1264   2018-08-23
17875    1118_14     1118   2018-12-26
44195  1293_1132     1293   2018-08-18
39570   1257_123     1257   2018-09-28
59566    1379_43     1379   2018-12-26
9007    1066_180     1066   2018-08-26
67648    1439_42     1439   2018-09-28
```

[Describe lo que ves y observas en la información general y en la muestra de datos impresa para el precio de datos anterior. ¿Hay algún problema (tipos de datos no adecuados, datos ausentes, etc.) que pudieran necesitar investigación y cambios adicionales? ¿Cómo se puede arreglar?]

No posee datos ausentes y los tipos de datos son los adecuados, excepto la columna message_date

1.9.1 Corregir los datos

[Corrige los problemas obvios con los datos basándote en las observaciones iniciales.]

[89]:
```
df_messages['message_date'] = pd.to_datetime(df_messages['message_date'],␣
↪format='%Y-%m-%d')
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  datetime64[ns]
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 1.7+ MB
```

1.9.2 Enriquecer los datos

[Agrega factores adicionales a los datos si crees que pudieran ser útiles.]

[ ]:
```
```

1.10 Internet

[90]:
```
df_internet.info()# Imprime la información general/resumida sobre el DataFrame␣
↪de internet
```

```
<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
```

[91]:
```
print(df_internet.sample(10))# Imprime una muestra de datos para el tráfico de␣
 ↪internet
```

```
              id  user_id session_date  mb_used
47283   1211_160     1211   2018-10-29   440.58
102979   1493_97     1493   2018-12-23   264.36
83924   1391_168     1391   2018-10-02     0.00
4227     1027_55     1027   2018-10-28   337.49
90479    1417_12     1417   2018-10-26   273.38
52825    1238_59     1238   2018-12-06   407.68
96707   1454_163     1454   2018-07-27   604.65
45813   1203_138     1203   2018-12-05   302.47
11820   1057_160     1057   2018-12-15   230.71
19071   1085_145     1085   2018-12-31   125.72
```

[Describe lo que ves y observas en la información general y en la muestra de datos impresa para el precio de datos anterior. ¿Hay algún problema (tipos de datos no adecuados, datos ausentes, etc.) que pudieran necesitar investigación y cambios adicionales? ¿Cómo se puede arreglar?]

En esta muestra, en la columna "mb_used", veo valores atipicos o extremos. al igual que los anterios datasets, el tipo de dato para fecha hay que modificarlos y no tiene valores ausentes

1.10.1 Corregir los datos

[Corrige los problemas obvios con los datos basándote en las observaciones iniciales.]

[92]:
```
df_internet['session_date'] = pd.to_datetime(df_internet['session_date'],␣
 ↪format='%Y-%m-%d')
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  datetime64[ns]
 3   mb_used       104825 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 3.2+ MB
```

1.10.2 Enriquecer los datos

[Agrega factores adicionales a los datos si crees que pudieran ser útiles.]

Al igual que en el dataset de tarifas, se realizo el cambio de MB a GB para que los futuros calculos sean legibles y sin errores

En conclusion con el codigo que imprime el consumo de gigabytes de los clientes podemos ver un patron de consumo

[93]:
```
df_internet['gb_used'] = df_internet['mb_used']/1024
df_internet["gb_used"] = df_internet["gb_used"].round(2)
df_internet.drop('mb_used', axis=1, inplace=True)
print(df_internet.head())
```

```
         id  user_id session_date  gb_used
0   1000_13     1000   2018-12-29     0.09
1  1000_204     1000   2018-12-31     0.00
2  1000_379     1000   2018-12-28     0.64
3  1000_413     1000   2018-12-26     0.26
4  1000_442     1000   2018-12-27     0.86
```

[94]:
```
cantidad_gigas = {"bajo_uso": 0, "mediano_uso": 0, "alto_uso": 0}

for gigas in df_internet["gb_used"]:
    try:
        float_gigas = float(gigas)
        if float_gigas <= 5:
            cantidad_gigas["bajo_uso"] += 1
        elif float_gigas <= 10:
            cantidad_gigas["mediano_uso"] += 1
        else:
            float_gigas <= 15
            cantidad_gigas["alto_uso"] += 1
    except ValueError:

        print(f"Valor de megas no válido: {gigas}")


for result in cantidad_gigas:
    print(result, '-', cantidad_gigas[result])
```

```
bajo_uso - 104825
mediano_uso - 0
alto_uso - 0
```

1.11 Estudiar las condiciones de las tarifas

[Es sumamente importante entender cómo funcionan las tarifas, cómo se les cobra a los usuarios en función de su plan de suscripción. Así que te sugerimos imprimir la información de la tarifa para

ver una vez más sus condiciones.]

[95]:
```
print(df_plans.head())# Imprime las condiciones de la tarifa y asegúrate de que
 ↪te quedan claras
```

```
   messages_included  minutes_included  usd_monthly_pay  usd_per_gb  \
0                 50               500               20          10
1               1000              3000               70           7

   usd_per_message  usd_per_minute plan_name  gb_per_month_included
0             0.03            0.03      surf                   15.0
1             0.01            0.01  ultimate                   30.0
```

1.12 Agregar datos por usuario

[Ahora que los datos están limpios, agrega los datos por usuario y por periodo para que solo haya un registro por usuario y por periodo. Esto facilitará mucho el análisis posterior.]

[96]:
```
df_calls['mes'] = pd.to_datetime(df_calls['call_date']).dt.month
llamadas_por_usuario_mes = df_calls.groupby(['user_id', 'mes'])['id'].count()
print(llamadas_por_usuario_mes.sample(10))
# Calcula la cantidad de llamadas por cada usuario al mes. Guarda el resultado.
```

```
user_id  mes
1317     11      58
1001     8       27
1270     12      50
1069     9       26
1052     12     177
1440     9       96
1411     9       65
1456     8       67
1297     12      15
1389     10      69
Name: id, dtype: int64
```

[97]:
```
df_calls['total_minutos_por_llamada'] = df_calls['duration'] * df_calls.
 ↪groupby(['user_id', pd.to_datetime(df_calls['call_date']).dt.month])['id'].
 ↪transform('count')
print(df_calls.sample(10))

# Calcula la cantidad de minutos usados por cada usuario al mes. Guarda el
 ↪resultado.
```

```
              id  user_id  call_date  duration  mes  total_minutos_por_llamada
34429   1127_308     1127 2018-10-13     14.02   10                     378.54
63733   1231_447     1231 2018-12-27      7.36   12                     574.08
33605   1126_343     1126 2018-07-27      0.00    7                       0.00
22981    1084_28     1084 2018-10-25      4.15   10                     120.35
```

```
983       1007_4     1007 2018-12-01     12.74   12                  1108.38
82498   1299_156     1299 2018-08-17      6.05    8                   387.20
56228   1201_328     1201 2018-08-10      0.11    8                     8.91
51822    1187_15     1187 2018-05-12      0.00    5                     0.00
30954   1115_716     1115 2018-12-24      7.94   12                   913.10
131481  1472_507     1472 2018-05-30      0.00    5                     0.00
```

[98]:
```
df_messages['mes'] = pd.to_datetime(df_messages['message_date']).dt.month
mensajes_por_usuario_mes = df_messages.groupby(['user_id', 'mes'])['id'].count()
print(mensajes_por_usuario_mes.sample(5))

# Calcula el número de mensajes enviados por cada usuario al mes. Guarda el␣
 ↪resultado.
```

```
user_id  mes
1363     11     34
1460     11     54
1316     7      61
1147     9      18
1075     9      48
Name: id, dtype: int64
```

[99]:
```
df_internet['mes'] = pd.to_datetime(df_internet['session_date']).dt.month
trafico_internet_por_usuario_mes = df_internet.groupby(['user_id',␣
 ↪'mes'])['gb_used'].sum()
print(trafico_internet_por_usuario_mes.sample(5))

# Calcula el volumen del tráfico de Internet usado por cada usuario al mes.␣
 ↪Guarda el resultado.
```

```
user_id  mes
1341     11     22.32
1109     7      19.53
1004     6      20.22
1268     4      25.38
1059     6      10.65
Name: gb_used, dtype: float64
```

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

[100]:
```
df_messages['mes'] = pd.DatetimeIndex(df_messages['message_date']).month

df_calls['mes'] = pd.DatetimeIndex(df_calls['call_date']).month

df_internet["mes"] = pd.DatetimeIndex(df_internet["session_date"]).month
```

```
# Fusiona los datos de llamadas, minutos, mensajes e Internet con base en␣
 ↪user_id y month
```

[101]:
```
messages_pivot = df_messages.pivot_table(index = ['user_id', 'mes'], values =␣
 ↪'id', aggfunc = 'count')

calls_pivot = df_calls.pivot_table(index =  ['user_id', 'mes'], values = 'id',␣
 ↪aggfunc = 'count')

calls_minutos_pivot = df_calls.pivot_table(index = ['user_id', 'mes'], values =␣
 ↪'total_minutos_por_llamada', aggfunc = 'sum')

internet_pivot = df_internet.pivot_table(index =['user_id', 'mes'], values =␣
 ↪'gb_used', aggfunc='sum')
```

[102]:
```
calls_y_minutos = calls_pivot.merge(calls_minutos_pivot, on=['user_id','mes'],␣
 ↪how='outer')
```

[103]:
```
calls_y_messages = calls_y_minutos.merge(internet_pivot, on=['user_id','mes'],␣
 ↪how='outer')
```

[104]:
```
df_final = calls_y_messages.merge(messages_pivot, on = ['user_id','mes'],␣
 ↪how='outer')
print(df_final.sample(10))
```

```
              id_x  total_minutos_por_llamada  gb_used  id_y
user_id mes
1095    6     16.0                    1279.04    12.23   7.0
1090    11    62.0                   27627.20    20.86  24.0
1403    7     17.0                    2152.03    17.86   NaN
1090    6     47.0                   13921.40    15.12  27.0
1390    4     52.0                   17813.12    13.58   NaN
1278    11    52.0                   13561.08    14.45  32.0
1334    11   131.0                  127263.88    18.45  13.0
1041    8     58.0                   27006.54    24.16   NaN
1201    8     81.0                   48102.66    34.69   6.0
1238    12    76.0                   37287.12    22.25   NaN
```

[105]:
```
df_con_tarifas = df_final.merge(df_users, on="user_id", how="left")
df_con_tarifas.rename(columns={"id_x" : "conteo_llamadas", "id_y" :␣
 ↪"conteo_messages", "plan": "plan_name"}, inplace=True)
df_con_tarifas.drop(columns=['age', "churn_date"], inplace=True)
df_con_tarifas['mes'] = pd.to_datetime(df_con_tarifas['reg_date']).dt.month
```

[106]:
```
print(df_con_tarifas.sample(10))# Añade la información de la tarifa
```

```
      user_id  conteo_llamadas  total_minutos_por_llamada  gb_used  \
```

```
1830     1400             78.0                  42960.84    11.44
1016     1219             97.0                  65051.11    26.47
903      1193             29.0                   4879.25    13.83
1004     1215            123.0                 104244.96    30.27
1501     1333             61.0                  24235.91    18.79
372      1078             16.0                   1418.56     8.02
1399     1312             57.0                  20262.36    16.70
1205     1259             14.0                   1387.26     3.17
1511     1334            141.0                 142402.95    14.14
1262     1273             56.0                  22398.88    17.74

      conteo_messages first_name last_name  \
1830              NaN     Kenton   Hickman
1016             24.0      Gavin    Keller
903              51.0   Lacresha     Olsen
1004             96.0     Adelle     Knapp
1501              7.0       Macy     David
372              36.0    Earnest      Gray
1399             33.0       Kory   Emerson
1205             32.0     Etsuko     Perry
1511             17.0    Donovan    Horton
1262             52.0    Hermila      Ryan

                                                 city   reg_date plan_name  \
1830         Riverside-San Bernardino-Ontario, CA MSA 2018-03-08      surf
1016                   Detroit-Warren-Dearborn, MI MSA 2018-05-14      surf
903          Houston-The Woodlands-Sugar Land, TX MSA 2018-07-07      surf
1004        New York-Newark-Jersey City, NY-NJ-PA MSA 2018-07-01      surf
1501           Los Angeles-Long Beach-Anaheim, CA MSA 2018-11-24  ultimate
372                    Seattle-Tacoma-Bellevue, WA MSA 2018-02-11      surf
1399                                  Fresno, CA MSA 2018-01-26      surf
1205  Philadelphia-Camden-Wilmington, PA-NJ-DE-MD MSA 2018-03-16      surf
1511                   Seattle-Tacoma-Bellevue, WA MSA 2018-03-08      surf
1262                               Baton Rouge, LA MSA 2018-10-13  ultimate

      mes
1830    3
1016    5
903     7
1004    7
1501   11
372     2
1399    1
1205    3
1511    3
1262   10
```

[107]:
```
df_con_planes = df_con_tarifas.merge(df_plans, on="plan_name", how="left")

df_con_planes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2293 entries, 0 to 2292
Data columns (total 18 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   user_id                    2293 non-null   int64
 1   conteo_llamadas            2258 non-null   float64
 2   total_minutos_por_llamada  2258 non-null   float64
 3   gb_used                    2277 non-null   float64
 4   conteo_messages            1806 non-null   float64
 5   first_name                 2293 non-null   object
 6   last_name                  2293 non-null   object
 7   city                       2293 non-null   object
 8   reg_date                   2293 non-null   datetime64[ns]
 9   plan_name                  2293 non-null   object
 10  mes                        2293 non-null   int64
 11  messages_included          2293 non-null   int64
 12  minutes_included           2293 non-null   int64
 13  usd_monthly_pay            2293 non-null   int64
 14  usd_per_gb                 2293 non-null   int64
 15  usd_per_message            2293 non-null   float64
 16  usd_per_minute             2293 non-null   float64
 17  gb_per_month_included      2293 non-null   float64
dtypes: datetime64[ns](1), float64(7), int64(6), object(4)
memory usage: 340.4+ KB
```

[Calcula los ingresos mensuales por usuario (resta el límite del paquete gratuito del número total de llamadas, mensajes de texto y datos; multiplica el resultado por el valor del plan de llamadas; añade la tarifa mensual en función del plan de llamadas). Nota: Dadas las condiciones del plan, ¡esto podría no ser tan trivial como un par de líneas! Así que no pasa nada si dedicas algo de tiempo a ello.]

[108]:
```
minutos_mensual =df_con_planes['total_minutos_por_llamada'] -␣
 ↪df_con_planes['minutes_included']

cargos_minuto = minutos_mensual * df_con_planes['usd_per_minute']

# Calcula el ingreso mensual para cada usuario
```

[109]:
```
mensajes_mensual =  df_con_planes['conteo_messages'] -␣
 ↪df_con_planes['messages_included']

cargos_mensaje= mensajes_mensual * df_con_planes['usd_per_message']
```

[110]:
```
gb_mensual = df_con_planes['gb_used'] - df_con_planes['gb_per_month_included']

cargos_datos = gb_mensual * df_con_planes['usd_per_gb']
```

[111]:
```
cargos_suma = cargos_minuto + cargos_mensaje + cargos_datos

df_con_planes ["cargos_totales"] = cargos_suma +␣
 ↪df_con_planes["usd_monthly_pay"]

df_con_planes["cargos_totales"].fillna(0,inplace=True)

df_con_planes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2293 entries, 0 to 2292
Data columns (total 19 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   user_id                    2293 non-null   int64
 1   conteo_llamadas            2258 non-null   float64
 2   total_minutos_por_llamada  2258 non-null   float64
 3   gb_used                    2277 non-null   float64
 4   conteo_messages            1806 non-null   float64
 5   first_name                 2293 non-null   object
 6   last_name                  2293 non-null   object
 7   city                       2293 non-null   object
 8   reg_date                   2293 non-null   datetime64[ns]
 9   plan_name                  2293 non-null   object
 10  mes                        2293 non-null   int64
 11  messages_included          2293 non-null   int64
 12  minutes_included           2293 non-null   int64
 13  usd_monthly_pay            2293 non-null   int64
 14  usd_per_gb                 2293 non-null   int64
 15  usd_per_message            2293 non-null   float64
 16  usd_per_minute             2293 non-null   float64
 17  gb_per_month_included      2293 non-null   float64
 18  cargos_totales             2293 non-null   float64
dtypes: datetime64[ns](1), float64(8), int64(6), object(4)
memory usage: 358.3+ KB
```

1.13 Estudia el comportamiento de usuario

[Calcula algunas estadísticas descriptivas para los datos agregados y fusionados que nos sean útiles y que muestren un panorama general captado por los datos. Dibuja gráficos útiles para facilitar la comprensión. Dado que la tarea principal es comparar las tarifas y decidir cuál es más rentable, las estadísticas y gráficas deben calcularse por tarifa.]

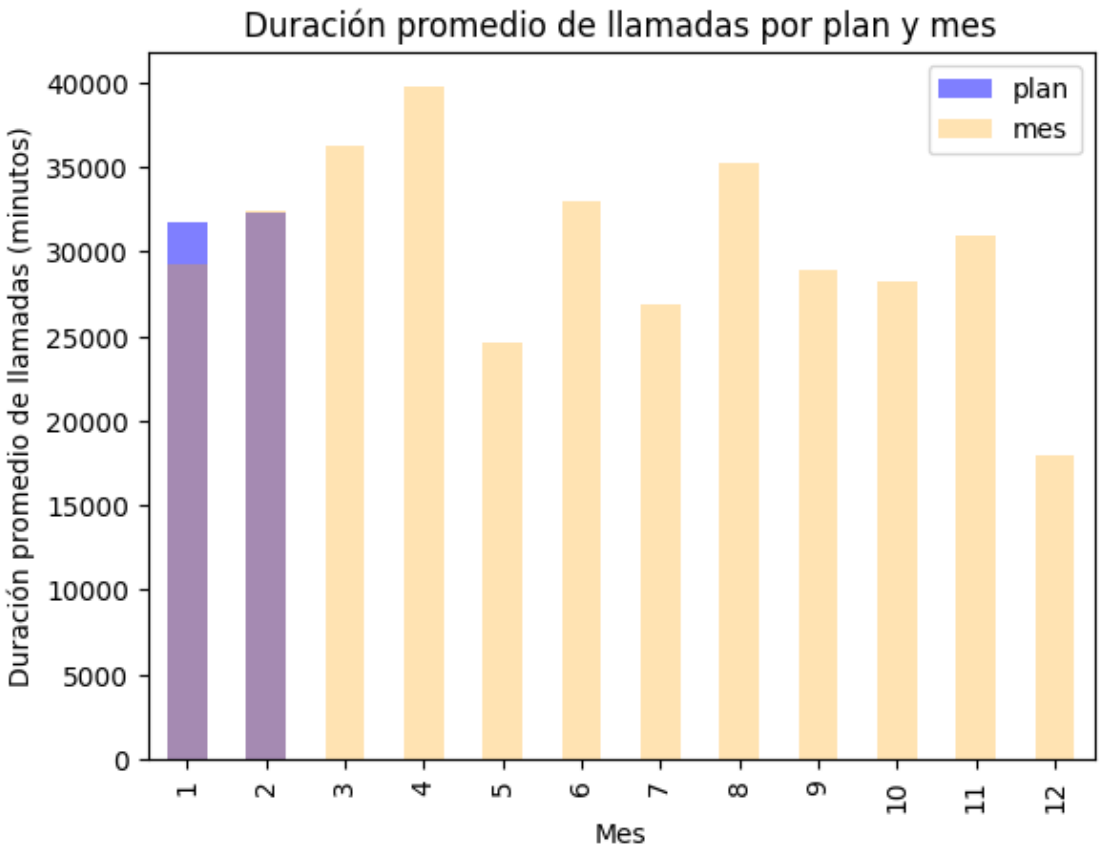
[En los comentarios hallarás pistas relevantes para las llamadas, pero no las hay para los mensajes

e Internet. Sin embargo, el principio del estudio estadístico que se aplica para ellos es el mismo que para las llamadas.]

1.13.1 Llamadas

[112]:
```
duracion_promedio = df_con_planes.groupby(['plan_name',␣
↪'mes'])['total_minutos_por_llamada'].mean()

# Compara la duración promedio de llamadas por cada plan y por cada mes. Traza␣
↪un gráfico de barras para visualizarla.
```

[113]:
```
promedio__llamadas_plan = df_con_planes.
↪groupby('plan_name')['total_minutos_por_llamada'].mean()
promedio_llamadas_mes = df_con_planes.
↪groupby('mes')['total_minutos_por_llamada'].mean()

colores = ['blue', 'orange']

promedio__llamadas_plan.plot(kind='bar', color= colores[0], label='plan',␣
↪alpha=0.5)
promedio_llamadas_mes.plot(kind='bar', color=colores[1], label='mes', alpha=0.3)

plt.xlabel('Mes')
plt.ylabel('Duración promedio de llamadas (minutos)')
plt.title("Duración promedio de llamadas por plan y mes")
plt.legend()
plt.show()
```



[114]:
```
minutos_mensuales = df_con_planes.pivot_table(index = ['user_id', 'mes'],␣
 ↪values = 'total_minutos_por_llamada', columns = "plan_name",aggfunc = 'sum')
minutos_mensuales.fillna(0,inplace=True)
print(minutos_mensuales)

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

```
plan_name          surf  ultimate
user_id mes
1000    12         0.00   1869.28
1001    8      91422.58      0.00
1002    10     37454.51      0.00
1003    1     155109.00      0.00
1004    5     126713.50      0.00
…                 …         …
1495    9     109062.17      0.00
1496    2      58721.12      0.00
1497    12         0.00  14932.62
1498    2     125284.36      0.00
```

```
1499    5     76210.71      0.00

[490 rows x 2 columns]
```

[ ]:

[Calcula la media y la variable de la duración de las llamadas para averiguar si los usuarios de los distintos planes se comportan de forma diferente al realizar sus llamadas.]

(1er gra)En general, la duración promedio de las llamadas es mayor para los planes más caros. Esto sugiere que los usuarios que pagan más por sus planes tienden a realizar llamadas más largas. La duración promedio de las llamadas varía a lo largo del año. Se observa un pico en la duración promedio de las llamadas durante los meses de noviembre y diciembre. Esto podría deberse a las fiestas navideñas, cuando las personas suelen realizar más llamadas a familiares y amigos.

[115]:
```
df_con_planes['fecha_mes'] = pd.to_datetime(df_con_planes['reg_date']).dt.
 ↪strftime('%Y-%m')
estadisticas_mensuales_llamadas = df_con_planes.groupby(['plan_name',␣
 ↪'fecha_mes'])['total_minutos_por_llamada'].agg(['mean', 'var'])
print(estadisticas_mensuales_llamadas)
# Calcula la media y la varianza de la duración mensual de llamadas.
```

```
                            mean           var
plan_name fecha_mes
surf      2018-01    27913.259307  5.740164e+08
          2018-02    39113.482775  1.414775e+09
          2018-03    37275.828101  1.932869e+09
          2018-04    37315.238436  1.021216e+09
          2018-05    24518.474809  3.255052e+08
          2018-06    29687.837931  7.876001e+08
          2018-07    27461.685766  6.324359e+08
          2018-08    25610.409364  3.746966e+08
          2018-09    26086.822121  6.795851e+08
          2018-10    27642.351951  8.270876e+08
          2018-11    36378.495952  1.617584e+09
          2018-12    22427.676400  6.812275e+08
ultimate  2018-01    31299.782481  5.313383e+08
          2018-02    13043.625000  1.424420e+08
          2018-03    33095.889091  4.486252e+08
          2018-04    48504.060000  1.462295e+09
          2018-05    24802.916329  3.975281e+08
          2018-06    37290.027315  2.146402e+09
          2018-07    25546.102391  9.608133e+08
          2018-08    53236.800678  3.173237e+09
          2018-09    33741.390263  4.063959e+08
          2018-10    30593.115455  7.133875e+08
          2018-11    21956.530000  5.715239e+08
          2018-12     4005.253750  2.375859e+07
```

[44]:
```
sns.boxplot(estadisticas_mensuales_llamadas)

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

[Elabora las conclusiones sobre el comportamiento de los usuarios con respecto a las llamadas. ¿Su comportamiento varía en función del plan?]

(media y varianza) La media (mean) es de 27913.26 minutos, lo que significa que, en promedio, los usuarios del plan "surf" en enero de 2018 hablaron durante 27913.26 minutos por mes. La varianza (var) es de 5.7, lo que indica que hay una variabilidad significativa en la duración de las llamadas entre los usuarios de este plan y mes. Un valor alto de varianza sugiere que algunos usuarios hablaron mucho más o mucho menos que el promedio. Plan "ultimate" en diciembre de 2018:

La media (mean) es de 4005.25 minutos, lo que significa que, en promedio, los usuarios del plan "ultimate" en diciembre de 2018 hablaron durante 4005.25 minutos por mes. La varianza (var) es de 2.4, que es considerablemente menor que la varianza del plan "surf" en enero de 2018. Esto sugiere que la duración de las llamadas en este plan y mes fue más homogénea entre los usuarios.

1.13.2 Mensajes

[116]:
```
mensajes_mensuales = df_con_planes.pivot_table(index = ['user_id', 'mes'],␣
 ↪values = 'conteo_messages', columns = "plan_name",aggfunc = 'sum')
mensajes_mensuales.fillna(0,inplace=True)
print(mensajes_mensuales)

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

```
plan_name     surf  ultimate
user_id mes
1000    12     0.0      11.0
1001    8    207.0       0.0
1002    10    88.0       0.0
1003    1     50.0       0.0
1004    5    177.0       0.0
…             …        …
1495    9      0.0       0.0
1496    2     65.0       0.0
1497    12     0.0      50.0
1498    2      0.0       0.0
1499    5      0.0       0.0

[490 rows x 2 columns]
```

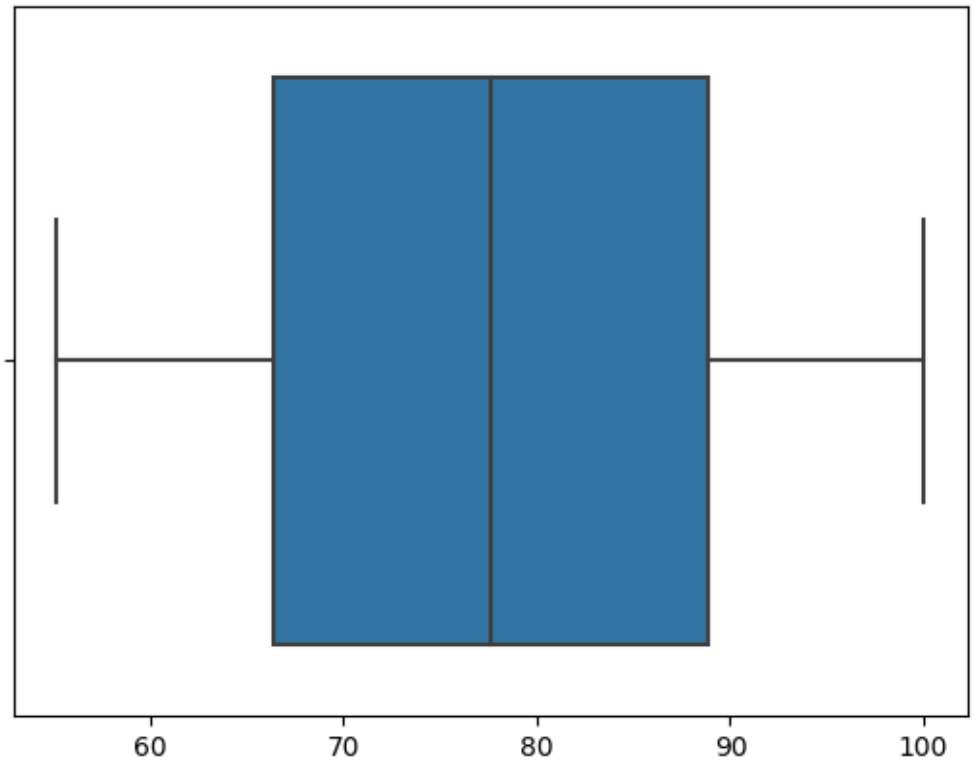
[117]:
```
media_anual_mensajes= mensajes_mensuales.mean()
print(media_anual_mensajes)
```

```
plan_name
```

```
surf        100.028571
ultimate     55.177551
dtype: float64
```

[118]:
```
sns.boxplot(media_anual_mensajes)
```

[118]: <AxesSubplot:>

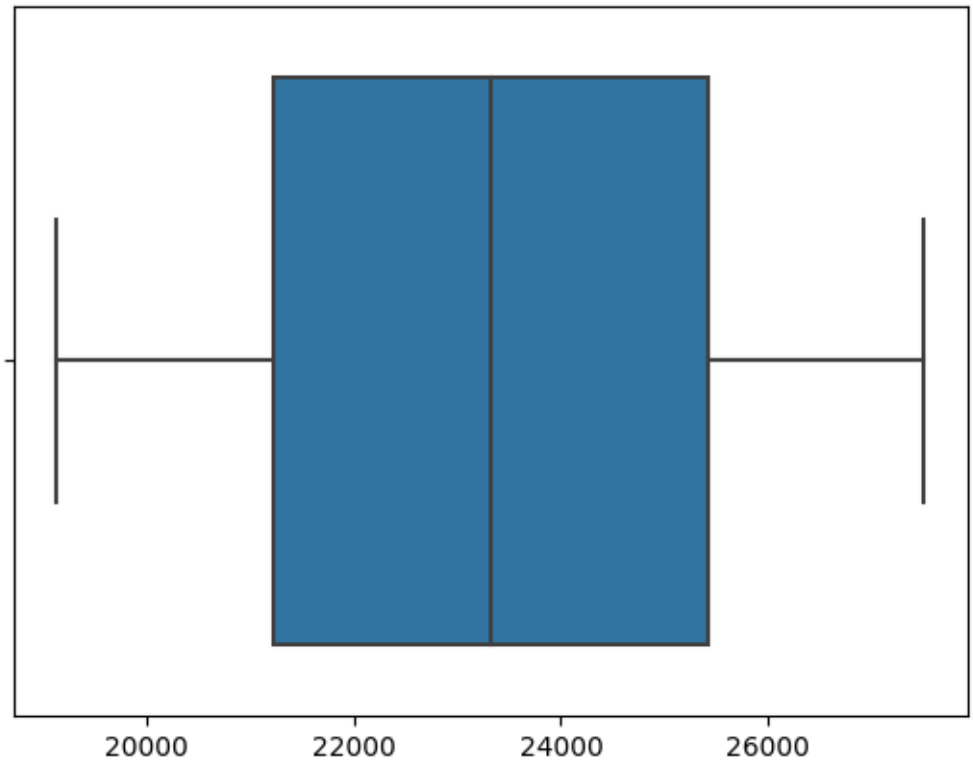


[119]:
```
varianza_anual_mensajes = mensajes_mensuales.var()
print(varianza_anual_mensajes)
```

```
plan_name
surf        27511.140286
ultimate    19129.577818
dtype: float64
```

[120]:
```
sns.boxplot(varianza_anual_mensajes)
```

[120]: <AxesSubplot:>



[Elabora las conclusiones sobre el comportamiento de los usuarios con respecto a los mensajes. ¿Su comportamiento varía en función del plan?]

Los usuarios del plan "surf" envían más mensajes en promedio que los usuarios del plan "ultimate", por otro lado la cantidad de mensajes enviados por los usuarios del plan "surf" es más variable que la cantidad de mensajes enviados por los usuarios del plan "ultimate".

1.13.3 Internet

[121]:
```
gb_mensuales = df_con_planes.pivot_table(index = ['user_id', 'mes'], values =␣
  ↪'gb_used', columns = "plan_name",aggfunc = 'sum')
gb_mensuales.fillna(0,inplace=True)
print(gb_mensuales)

# Compara la cantidad de tráfico de Internet consumido por usuarios por plan
```

```
plan_name      surf  ultimate
user_id mes
1000    12     0.00      1.85
1001    8     78.52      0.00
1002    10    39.38      0.00
1003    1     26.40      0.00
1004    5    152.85      0.00
```

```
…              …        …
1495    9      96.52      0.00
1496    2      62.78      0.00
1497    12      0.00     10.85
1498    2     222.25      0.00
1499    5      69.77      0.00

[490 rows x 2 columns]
```

[122]:
```
gb_anuales_varianza = gb_mensuales.var()
gb_anuales_varianza = gb_anuales_varianza.round(2)
print(gb_anuales_varianza)
```

```
plan_name
surf        3589.44
ultimate    2644.80
dtype: float64
```

[123]:
```
media_anual_gb= gb_mensuales.mean()
media_anual_gb = media_anual_gb.round(2)
print(media_anual_gb)
```

```
plan_name
surf        51.91
ultimate    24.70
dtype: float64
```

[Elabora las conclusiones sobre cómo los usuarios tienden a consumir el tráfico de Internet. ¿Su comportamiento varía en función del plan?]

Los usuarios del plan “surf” consumen más MB en promedio por año que los usuarios del plan “ultimate”. La media anual de MB para “surf” es más del doble que la de “ultimate” y por otro lado la cantidad de MB consumidos por los usuarios del plan “surf” es más variable a lo largo del año que la cantidad de MB consumidos por los usuarios del plan “ultimate”. La varianza anual de “surf” es considerablemente mayor que la de “ultimate”

1.14 Ingreso

[Del mismo modo que has estudiado el comportamiento de los usuarios, describe estadísticamente los ingresos de los planes.]

[124]:
```
estadisticas_mensuales_ingresos = df_con_planes.groupby(['plan_name',␣
↪'fecha_mes'])['cargos_totales'].agg(['mean', 'var'])
print(estadisticas_mensuales_ingresos)
```

```
                          mean           var
plan_name fecha_mes
surf      2018-01    740.734142  6.406577e+05
          2018-02    922.953812  1.463661e+06
```

```
           2018-03    863.842666  2.096225e+06
           2018-04    964.494900  1.033704e+06
           2018-05    435.519869  2.621727e+05
           2018-06    605.087265  5.993892e+05
           2018-07    776.951632  6.778681e+05
           2018-08    608.312335  4.048473e+05
           2018-09    528.068028  5.021132e+05
           2018-10    752.614336  8.579846e+05
           2018-11    791.786676  1.182663e+06
           2018-12    602.507056  7.011680e+05
ultimate   2018-01    217.314325  6.876143e+04
           2018-02     80.085590  1.041784e+04
           2018-03    103.816842  5.741926e+04
           2018-04    419.291918  1.576002e+05
           2018-05    127.459752  4.694693e+04
           2018-06    309.195345  2.177757e+05
           2018-07    150.911150  9.908981e+04
           2018-08    433.929544  3.566102e+05
           2018-09    206.471992  4.805559e+04
           2018-10    234.707973  7.093281e+04
           2018-11     72.330400  4.192643e+04
           2018-12    -51.871744  6.040093e+03
```

[125]:
```
ingresos_mensuales = df_con_planes.pivot_table(index = ['user_id', 'mes'],␣
 ↪values = 'cargos_totales', columns = "plan_name",aggfunc = 'sum')
ingresos_mensuales.fillna(0,inplace=True)
print(ingresos_mensuales)
```

```
plan_name         surf  ultimate
user_id mes
1000    12      0.0000 -148.2472
1001    8    2801.5874    0.0000
1002    10   1080.5753    0.0000
1003    1    4772.2700    0.0000
1004    5    4163.2150    0.0000
…                …        …
1495    9       0.0000    0.0000
1496    2    1658.8836    0.0000
1497    12      0.0000   45.7762
1498    2       0.0000    0.0000
1499    5       0.0000    0.0000

[490 rows x 2 columns]
```

[126]:
```
tamaño_muestra = 50
```

```
muestra_plan_surf = ingresos_mensuales["surf"].sample(n=tamaño_muestra,␣
 ↪replace=False)

muestra_plan_ultimate = ingresos_mensuales["ultimate"].sample(n=tamaño_muestra,␣
 ↪replace=False)
print(type(muestra_plan_surf))
```

```
<class 'pandas.core.series.Series'>
```

[127]:
```
print(muestra_plan_surf.mean())
```

```
1988.711098
```

[128]:
```
print(muestra_plan_ultimate.mean())
```

```
41.06143999999999
```

[129]:
```
media_anual_ingresos= ingresos_mensuales.mean()
media_anual_ingresos = media_anual_ingresos.round(2)
print(media_anual_ingresos)
varianza_anual_ingresos= ingresos_mensuales.var()
varianza_anual_ingresos = varianza_anual_ingresos.round(2)
print(varianza_anual_ingresos)
```

```
plan_name
surf        2435.18
ultimate     328.64
dtype: float64
plan_name
surf        21893152.14
ultimate     1218688.33
dtype: float64
```

[Elabora las conclusiones sobre cómo difiere el ingreso entre los planes.]

Podría haber una tendencia "estacional" en el uso del plan "surf". Los minutos tienden a ser más altos en los meses de primavera y verano (abril-agosto) y más bajos en los meses de otoño e invierno (octubre-marzo). Esto podría deberse a factores como el aumento de los viajes o las actividades al aire libre durante los meses más cálidos

1.15 Prueba las hipótesis estadísticas

[Prueba la hipótesis de que son diferentes los ingresos promedio procedentes de los usuarios de los planes de llamada Ultimate y Surf.]

[Elabora las hipótesis nula y alternativa, escoge la prueba estadística, determina el valor alfa.]

Para obtener los resultados de esta prueba de hipotesis se realizaron 2 muestras aleatorias, de 50 clientes cada una, una para el plan Surf y otra para el plan Ultimate, se establecio alpha y para aceptar o rechazar la hipotesis se hizo una prueba t sobre la igualdad de las medias de 2 poblaciones

Los resultados de la prueba t sugieren que el ingreso promedio para los usuarios del plan de llamadas Ultimate es probablemente diferente del ingreso promedio para los usuarios del plan de llamadas Surf. El valor p de 0.001582 indica que existe una probabilidad muy baja (menos del 0.15%) de observar tal diferencia si la hipótesis nula fuera cierta.

[130]:
```
# Hipotesis nula (H0)= "los ingresos de los planes surf y ultimate son iguales"

# Hipotesis alternativa (H1)= "Los ingresos promedio de los usuarios de los␣
 ↪planes de llamada Ultimate y Surf no son iguales"



# Prueba las hipótesis
```

[131]:
```
alpha= 0.05

results = st.ttest_ind(muestra_plan_surf, muestra_plan_ultimate,␣
 ↪equal_var=False)

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

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

```
valor p: 3.845282807704358e-06
Rechazamos la hipótesis nula
```

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

[Elabora las hipótesis nula y alternativa, escoge la prueba estadística, determina el valor alfa.]

[136]:
```
#Hipotesis Nula (H0) = "el ingreso promedio de los usuarios del área NY-NJ es␣
 ↪diferente al de los usuarios de otras regiones"

#Hipotesis Alternativa (H1) = "el ingreso promedio de los usuarios del área␣
 ↪NY-NJ no es diferente al de los usuarios de otras regiones""

# Prueba las hipótesis
```

[133]:
```
usuarios_nynj = df_con_planes[df_con_planes["city"] == "New York-Newark-Jersey␣
 ↪City, NY-NJ-PA MSA"]
usuarios_otras_regiones = df_con_planes[df_con_planes["city"] != "New␣
 ↪York-Newark-Jersey City, NY-NJ-PA MSA"]

promedio_cargos_nynj = usuarios_nynj["cargos_totales"].mean()
```

```
promedio_cargos_otras_regiones = usuarios_otras_regiones["cargos_totales"].
 ↪mean()
```

[135]:
```
results = st.
 ↪ttest_ind(usuarios_nynj["cargos_totales"],usuarios_otras_regiones["cargos_totales"]␣
 ↪, equal_var=False)


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

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

```
valor p: 0.38645520874311823
No podemos rechazar la hipótesis nula
```

1.16 Conclusión general

[En esta sección final, enumera tus conclusiones importantes. Asegúrate de que estas abarquen todas las decisiones (suposiciones) importantes que adoptaste y que determinaron la forma elegida para procesar y analizar los datos.]