Statistical Data Analysis Project

Comparative analysis of the telecom plans profitability

Table of Contents

Goal

Prepare a report for the telecom operator Megaline to analyze clients' behavior, determine which of the two prepaid plans is more profitable and test two statistical hypotheses.

Hypotheses

1. The average profit from users of Ultimate and Surf calling plans differs.
2. The average profit from users in NY-NJ area is different from that of the users from other regions.

Description of the data

Megaline rounds seconds up to minutes, and megabytes to gigabytes. For **calls**, each individual call is rounded up: even if the call lasted just one second, it will be counted as one minute. For **web traffic**, individual web sessions are not rounded up. Instead, the total for the month is rounded up. If someone uses 1025 megabytes this month, they will be charged for 2 gigabytes.

The `users` table (data on users):

- *user_id* — unique user identifier
- *first_name* — user's name
- *last_name* — user's last name
- *age* — user's age (years)
- *reg_date* — subscription date (dd, mm, yy)
- *churn_date* — the date the user stopped using the service (if the value is missing, the calling plan was being used when this database was extracted)
- *city* — user's city of residence
- *plan* — calling plan name

The `calls` table (data on calls):

- *id* — unique call identifier
- *call_date* — call date
- *duration* — call duration (in minutes)
- *user_id* — the identifier of the user making the call

The `messages` table (data on texts):

- *id* — unique text message identifier
- *message_date* — text message date
- *user_id* — the identifier of the user sending the text

The `internet` table (data on web sessions):

- *id* — unique session identifier
- *mb_used* — the volume of data spent during the session (in megabytes)
- *session_date* — web session date
- *user_id* — user identifier

The `plans` table (data on the plans):

- *plan_name* — calling plan name
- *usd_monthly_fee* — monthly charge in US dollars
- *minutes_included* — monthly minute allowance
- *messages_included* — monthly text allowance
- *mb_per_month_included* — data volume allowance (in megabytes)
- *usd_per_minute* — price per minute after exceeding the package limits (e.g., if the package includes 100 minutes, the 101st minute will be charged)
- *usd_per_message* — price per text after exceeding the package limits
- *usd_per_gb* — price per extra gigabyte of data after exceeding the package limits (1 GB = 1024 megabytes)

Imports

In [1]:

```
import pandas as pd
import numpy as np
import scipy
import matplotlib

from scipy import stats as st

import matplotlib.pyplot as plt
%matplotlib inline

import sys
import warnings
if not sys.warnoptions:
       warnings.simplefilter("ignore")

# pd.set_option('display.max_rows', None)

print("Setup Complete")
```

```
Setup Complete
```

Library version check and update

In [2]:

```
version_dict = {pd:'1.0.1', np:'1.18.1', scipy:'1.6.0', matplotlib:'3.1.3'}
```

In [3]:

```
def get_value(my_key):
    """
    If the val can be found in the dictinary.values() list,
    returns the key of the dictionary item in which the val was found.
    """
    wrong_val = []

    for key, value in version_dict.items():
        try:
            if key == my_key:
                return value
        except:
            wrong_val.append(key, value)
```

In [4]:

```
for lib in version_dict.keys():
    if lib.__version__ != get_value(lib):
        print("Warning: update", lib, 'to', get_value(lib))
```

In order to update scipy library to 1.6.0 version, run the following line:

In [5]:

```
!pip install scipy==1.6.0
```

```
Requirement already satisfied: scipy==1.6.0 in /anaconda3/lib/python
3.7/site-packages (1.6.0)
Requirement already satisfied: numpy>=1.16.5 in /anaconda3/lib/pytho
n3.7/site-packages (from scipy==1.6.0) (1.18.2)
```

Input data

In [6]:

```
try:
    df_users = pd.read_csv('megaline_users.csv')
    df_messages = pd.read_csv('megaline_messages.csv')
    df_calls = pd.read_csv('megaline_calls.csv')
    df_internet = pd.read_csv('megaline_internet.csv')
    df_plans = pd.read_csv('megaline_plans.csv')
except:
    df_users = pd.read_csv('/datasets/megaline_users.csv')
    df_messages = pd.read_csv('/datasets/megaline_messages.csv')
    df_calls = pd.read_csv('/datasets/megaline_calls.csv')
    df_internet = pd.read_csv('/datasets/megaline_internet.csv')
    df_plans = pd.read_csv('/datasets/megaline_plans.csv')
```

Overview

Users

In [7]:

```
df_users.head()
```

Out[7]:

	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

The `churn_date` is the date the user stopped using the service. If the value is missing, the calling plan was being used when this database was extracted. For the further analysis we will need to replace 'NaN' with the string 'in use'.

In [8]:

```
df_users.info()
```

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

No missing values, except for the `churn_date` column. The 2 date features have the object type, we will change it to the datetime type.

In [9]:

```
df_users.describe()
```

Out[9]:

	user_id	age
count	500.000000	500.000000
mean	1249.500000	45.486000
std	144.481833	16.972269
min	1000.000000	18.000000
25%	1124.750000	30.000000
50%	1249.500000	46.000000
75%	1374.250000	61.000000
max	1499.000000	75.000000

The dataset contains information about 500 users. Most of them are 45-46 years old. The age range is from 18 to 75 years old. No visible outliers here.

Messages

In [10]:

```
df_messages.head()
```

Out[10]:

	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

In [11]:

```
df_messages.info()
```

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

No missing values, `message_date` type should be changed to datetime format. `id` column has the object dtype which is harder to work with than integers. We may later decide to extract the message number for each user and group the table by user.

In [12]:

```
df_messages.describe()
```

Out[12]:

	user_id
count	76051.000000
mean	1245.972768
std	139.843635
min	1000.000000
25%	1123.000000
50%	1251.000000
75%	1362.000000
max	1497.000000

`user_id` starts from 1000 and ends with 1497. It means that 2 users were not messaging at all.

Calls

In [13]:

```
df_calls.head()
```

Out[13]:

	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 [14]:

```
df_calls.info()
```

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

No missing values, `call_date` should be changed to datetime format. The `duration` column has the float dtype but the policy of the company is to round up seconds to minutes, so we should convert this column to integers.

In [15]:

```
df_calls.describe()
```

Out[15]:

	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

There are 137 735 calls in the data set. The average and medium duration is about 6-7 minutes. Mean and median values are close to each other, meaning that the distribution does not have heavy tails. The minimum duration of a call is 0, which probably means a few seconds, according to the company's policy we should round it up to 1 minute. The maximum call duration is 38 minutes.

Internet

In [16]:

```
df_internet.head()
```

Out[16]:

	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

In [17]:

```
df_internet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 104825 entries, 0 to 104824
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            104825 non-null  object 
 1   user_id       104825 non-null  int64  
 2   session_date  104825 non-null  object 
 3   mb_used       104825 non-null  float64
dtypes: float64(1), int64(1), object(2)
memory usage: 3.2+ MB
```

No missing values, `session_date` type should be changed from object to datetime. For web traffic, individual web sessions are not rounded up. Instead, the total for the month is rounded up.

In [18]:

```
df_internet.describe()
```

Out[18]:

	user_id	mb_used
count	104825.000000	104825.000000
mean	1242.496361	366.713701
std	142.053913	277.170542
min	1000.000000	0.000000
25%	1122.000000	136.080000
50%	1236.000000	343.980000
75%	1367.000000	554.610000
max	1499.000000	1693.470000

There are 104 825 sessions in this data set. The mean and median values are close (around 350 mb), meaning this distribution does not have heavy tails either. The megabyte usage ranges from almost 0 to almost 1700.

Plans

In [19]:

```
df_plans
```

Out[19]:

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

In [20]:

```
df_plans.info()
```

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

There are only 2 plans in this data set - 'Surf' and 'Ultimate'. No issues with this data.

Preprocessing

Data type change

Dates

In [21]:

```
for df in [df_users, df_messages, df_calls, df_internet]:
    for col in ['reg_date','churn_date','message_date','call_date','session_dat
e']:
        if col in df.columns:
            df[col] = pd.to_datetime(df[col])
```

Call duration

Let's round up all calls to minutes as per the company's policy.

In [22]:

```
df_calls['total_min_month'] = np.ceil((df_calls['duration'].values))
```

Missing values

As discussed above, we will replace missing values in the `churn_date` column with the string 'in use'

In [23]:

```
df_users['churn_date'] = df_users['churn_date'].fillna('in use')
```

Duplicates

Let's check if any rows are duplicated in any data frames.

In [24]:

```
result = []
for df in [df_users, df_messages, df_calls, df_internet]:
    print(result.append(df.duplicated().sum()))
```

```
None
None
None
None
```

Check for artifacts in user dates

First, let's check if the `reg_date` is strictly less or equal than the `churn_date` in the df_users.

In [25]:

```
date_check = df_users.query('churn_date != "in use"')
sum(date_check['reg_date'] > date_check['churn_date'])
```

Out[25]:

```
0
```

We don't have any observations with wrong dates.

Now let's verify that all user activity stays inside the interval between the `reg_date` and the `churn_date` . For that purpose we will create a df_dates with all the datetime features.

In [26]:

```
df_dates = df_users.merge(df_messages, how='left').merge(df_calls,how='left').me
rge(df_internet, how='left')
```

In [27]:

```
len(df_dates[df_dates['reg_date'] > df_dates['message_date']])
```

Out[27]:

0

In [28]:

```
len(df_dates[df_dates['reg_date'] > df_dates['call_date']])
```

Out[28]:

0

In [29]:

```
len(df_dates[df_dates['reg_date'] > df_dates['session_date']])
```

Out[29]:

0

No artifacts found in dates.

Calculations

Calls per month

To calculate the number of calls made and minutes used per month for each user, first we will extract month from the `call_date` column and then make a pivot table grouping by `user_id` and `call_month` .

In [30]:

```
df_calls['month'] = df_calls['call_date'].dt.month
```

In [31]:

```
calls_per_month = df_calls.pivot_table(index=['user_id','month'], values='total_
min_month', aggfunc='sum')
calls_per_month = calls_per_month.reset_index()
```

In [32]:

```
calls_per_month.columns = ["user_id","month","total_min_month"]
```

In [33]:

```
calls_per_month.head()
```

Out[33]:

	user_id	month	total_min_month
0	1000	12	124.0
1	1001	8	182.0
2	1001	9	315.0
3	1001	10	393.0
4	1001	11	426.0

Messages per month

To calculate the number of text messages per month for each user, first we will extract month from the `message_date` column and then make a pivot table grouping by `user_id` and `message_month` .

In [34]:

```
df_messages['month'] = df_messages['message_date'].dt.month
```

In [35]:

```
messages_per_month = df_messages.pivot_table(index=['user_id','month'], values=
'id', aggfunc='count')
messages_per_month = messages_per_month.reset_index()
messages_per_month.columns = ['user_id','month','num_messages_month']
```

In [36]:

```
messages_per_month.head()
```

Out[36]:

	user_id	month	num_messages_month
0	1000	12	11
1	1001	8	30
2	1001	9	44
3	1001	10	53
4	1001	11	36

Volume of data per month

To calculate the volume of data per month for each user, first we will extract month from the `session_date` column and then make a pivot table grouping by `user_id` and `session_month`. For web traffic, the total volume in megabytes for the month is rounded up.

In [37]:

```
df_internet['month'] = df_internet['session_date'].dt.month
```

In [38]:

```
sessions_per_month = df_internet.pivot_table(index=['user_id','month'], values=
'mb_used', aggfunc='sum')
sessions_per_month = sessions_per_month.reset_index()

sessions_per_month.columns = ['user_id', 'month', "mb_volume_month"]
sessions_per_month['gb_volume_month'] = np.ceil((sessions_per_month['mb_volume_m
onth'].values) / 1000).astype(int)
```

In [39]:

```
sessions_per_month.head()
```

Out[39]:

	user_id	month	mb_volume_month	gb_volume_month
0	1000	12	1901.47	2
1	1001	8	6919.15	7
2	1001	9	13314.82	14
3	1001	10	22330.49	23
4	1001	11	18504.30	19

Monthly profit from each user

To calculate this we will subtract the free package limit from the total number of calls, text messages, and data. Then we will multiply the result by the calling plan value and add the monthly charge depending on the calling plan.

In [40]:

```
user_plan = df_users[['user_id','plan','city']]
user_plan = user_plan.merge(df_plans, left_on='plan', right_on='plan_name')
```

In [41]:

```
df_monthly = (calls_per_month
              .merge(messages_per_month, how='outer', on=['user_id','month'])
              .merge(sessions_per_month, how='outer', on=['user_id','month'])
              .merge(user_plan, on='user_id')
             )
```

In [42]:

```
df_monthly.head()
```

Out[42]:

	user_id	month	total_min_month	num_messages_month	mb_volume_month	gb_volume_mo
0	1000	12	124.0	11.0	1901.47	
1	1001	8	182.0	30.0	6919.15	
2	1001	9	315.0	44.0	13314.82	1
3	1001	10	393.0	53.0	22330.49	2
4	1001	11	426.0	36.0	18504.30	1

In [43]:

```
df_monthly.isnull().sum()
```

Out[43]:

```
user_id                    0
month                      0
total_min_month           35
num_messages_month       487
mb_volume_month           16
gb_volume_month           16
plan                       0
city                       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
```

We have a few null values for those users who only used 1 or 2 services - e.g. only messages and the internet but not calls and so on. We will thus replace these values with 0.

In [44]:

```
for col in ['total_min_month','num_messages_month','mb_volume_month','gb_volume_
month']:
    df_monthly[col] = df_monthly[col].fillna(0)
```

Finally, let's calculate the monthly profit per user according to the below formula:

In [45]:

```
df_monthly['monthly_profit'] = (((df_monthly['num_messages_month']-df_monthly['m
essages_included']) * df_monthly['usd_per_message'])
                                        + ((df_monthly['total_min_month']-df_mon
thly['minutes_included']) * df_monthly['usd_per_minute'])
                                        + ((df_monthly['gb_volume_month']-np.cei
l(df_monthly['mb_per_month_included'].values/1000)) * df_monthly['usd_per_gb'])
                                        + df_monthly['usd_monthly_pay'])
```

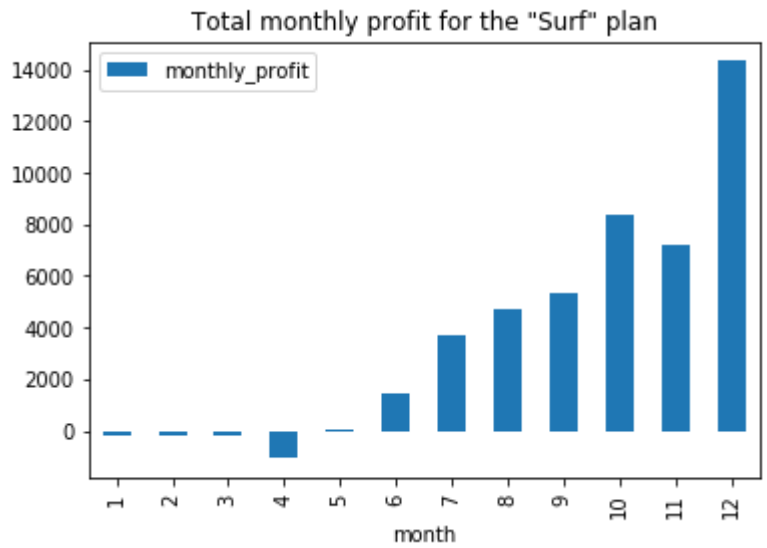
We will now group the table per plan and per month and visualize the results.

In [46]:

```
total_profit_month_plan = df_monthly.groupby(['plan','month'])['monthly_profit']
.sum().reset_index()
```

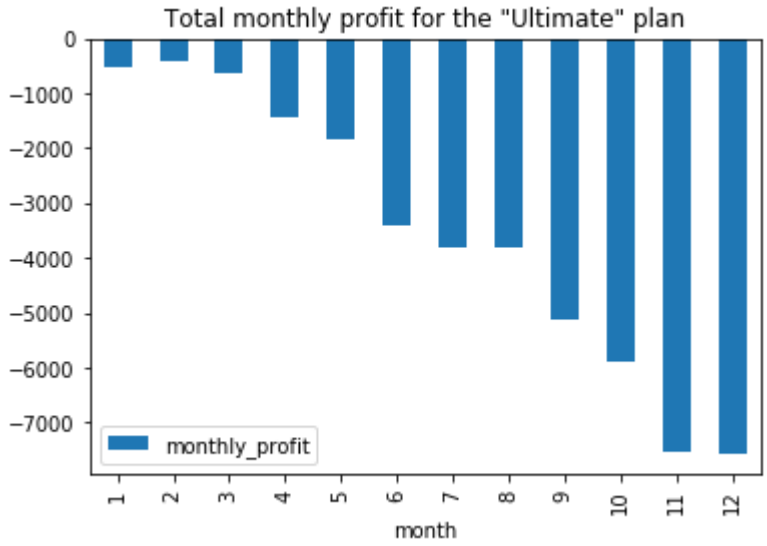
In [47]:

```
total_profit_month_plan[total_profit_month_plan['plan'] == 'surf'].plot(y='month
ly_profit', x='month', kind='bar')
plt.title('Total monthly profit for the "Surf" plan');
```



In [48]:

```
total_profit_month_plan[total_profit_month_plan['plan'] == 'ultimate'].plot(y='monthly_profit', x='month', kind='bar')
plt.title('Total monthly profit for the "Ultimate" plan');
```



We see that the "Surf" plan is mostly profitable throughout the year with a tendency of being more and more profit-making towards the end of the year. In contract, the "Ultimate" plan is mostly not profitable during the year, with the opposite tendency of being even more costly by the end of the year. It could be explained by the fact that the "Surf" plan does not include as much free calls, messages and gigabytes as the "Ultimate" plan, so these users more often exceed the limits of the plan. However, the "Ultimate" plan seems to have too much free data as almost all its users are far from the limits by the end of a month. According to the above plots, the "Ultimate" plan conditions should be revised.

EDA

Distribution analysis

Now we are going to further compare the two plans, so let's create separate data frames for them.

In [49]:

```
df_surf = df_monthly[df_monthly['plan'] == 'surf']
df_ultimate = df_monthly[df_monthly['plan'] == 'ultimate']
```

In [50]:

```
df_ultimate.shape
```

Out[50]:

```
(720, 17)
```

In [51]:

```
df_surf.shape
```

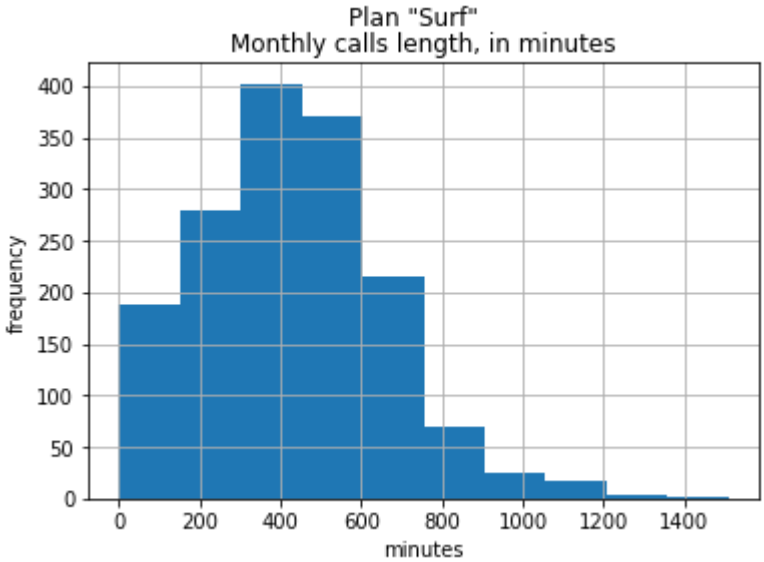
Out[51]:

```
(1573, 17)
```

Calls

In [52]:

```
df_surf.hist('total_min_month')
plt.title('Monthly calls length, in minutes')
plt.suptitle('Plan "Surf"')
plt.xlabel('minutes')
plt.ylabel('frequency');
```



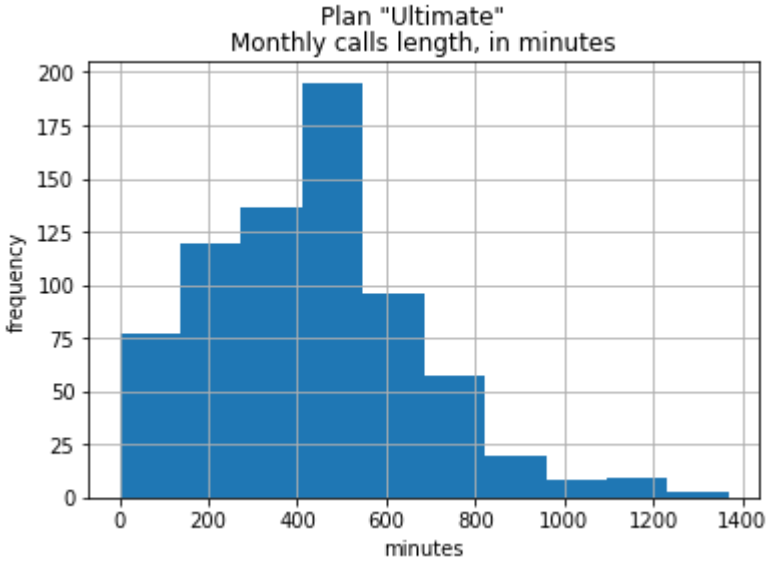
In [53]:

```
print('Average monthly calls length in minutes, plan "Surf": {:.0f}'.format(df_surf['total_min_month'].mean()))
print('Medium monthly calls length in minutes, plan "Surf":{:.0f}'.format(df_surf['total_min_month'].median()))
print('Standard deviation for monthly calls length in minutes, plan "Surf":{:.0f}'.format(df_surf['total_min_month'].std()))
```

```
Average monthly calls length in minutes, plan "Surf": 429
Medium monthly calls length in minutes, plan "Surf":425
Standard deviation for monthly calls length in minutes, plan "Surf":
234
```

In [54]:

```
df_ultimate.hist('total_min_month')
plt.title('Monthly calls length, in minutes')
plt.suptitle('Plan "Ultimate"')
plt.xlabel('minutes')
plt.ylabel('frequency');
```



In [55]:

```
print('Average monthly calls length in minutes, plan "Ultimate": {:.0f}'.format(
df_ultimate['total_min_month'].mean()))
print('Medium monthly calls length in minutes, plan "Ultimate":{:.0f}'.format(df
_ultimate['total_min_month'].median()))
print('Standard deviation for monthly calls length in minutes, plan "Ultimate":
{:.0f}'.format(df_ultimate['total_min_month'].std()))
```

```
Average monthly calls length in minutes, plan "Ultimate": 430
Medium monthly calls length in minutes, plan "Ultimate":424
Standard deviation for monthly calls length in minutes, plan "Ultima
te":241
```

Average and medium monthly calls length is almost the same (around 430-440 minutes per month) for both plans, which is a sign of a distribution close to normal, without any heavy tails. Standard deviations are also quite close (around 230-240 minutes).

In [56]:

```
print('People exceeded the "Surf" plan limit for calls: {:.0f}'.format(len(df_su
rf.query('total_min_month > 500'))))
print('Ratio of people exceeded the "Surf" plan limit for calls: {:.0%}'.format(
len(df_surf.query('total_min_month > 500')) / len(df_surf)))
```

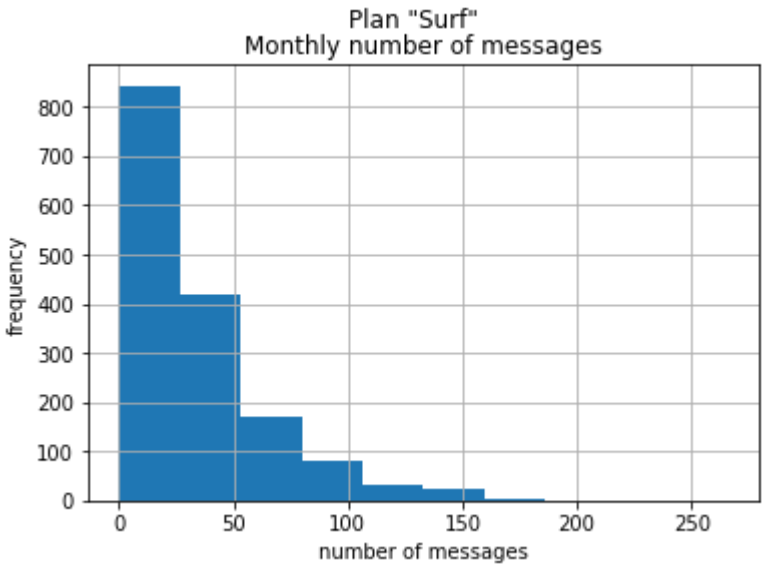
```
People exceeded the "Surf" plan limit for calls: 566
Ratio of people exceeded the "Surf" plan limit for calls: 36%
```

The "Surf" plan limit is 500 minutes per month. This value is close to the mean and median, so most users stay within this limit while others - 566 people (36%) which is almost a half! - exceed it. Those who exceed the limit create profit for the company. The "Ultimate" plan limit is 3000 minutes per month. No user is spending that many minutes - the highest number is even less than 1400 minutes per month. This is where the company looses much profit, so this limit must be reviewed.

Messages

In [57]:

```
df_surf.hist('num_messages_month')
plt.title('Monthly number of messages')
plt.suptitle('Plan "Surf"')
plt.xlabel('number of messages')
plt.ylabel('frequency');
```



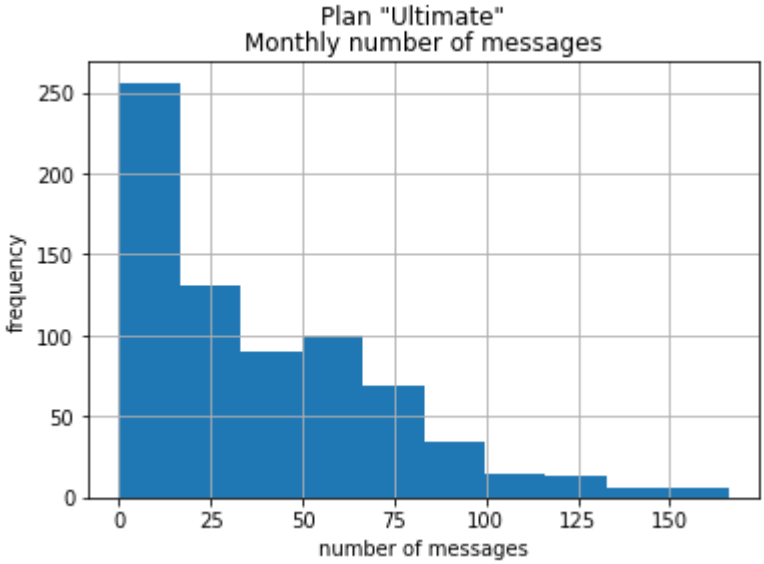
In [58]:

```
print('Average monthly number of messages, plan "Surf": {:.0f}'.format(df_surf[
'num_messages_month'].mean()))
print('Medium monthly number of messages, plan "Surf":{:.0f}'.format(df_surf['nu
m_messages_month'].median()))
print('Standard deviation for monthly number of messages, plan "Surf":{:.0f}'.fo
rmat(df_surf['num_messages_month'].std()))
```

```
Average monthly number of messages, plan "Surf": 31
Medium monthly number of messages, plan "Surf":24
Standard deviation for monthly number of messages, plan "Surf":34
```

In [59]:

```
df_ultimate.hist('num_messages_month')
plt.title('Monthly number of messages')
plt.suptitle('Plan "Ultimate"')
plt.xlabel('number of messages')
plt.ylabel('frequency');
```



In [60]:

```
print('Average monthly number of messages, plan "Ultimate": {:.0f}'.format(df_ul
timate['num_messages_month'].mean()))
print('Medium monthly number of messages, plan "Ultimate":{:.0f}'.format(df_ulti
mate['num_messages_month'].median()))
print('Standard deviation for monthly number of messages, plan "Ultimate":{:.0f}
'.format(df_ultimate['num_messages_month'].std()))
```

```
Average monthly number of messages, plan "Ultimate": 38
Medium monthly number of messages, plan "Ultimate":30
Standard deviation for monthly number of messages, plan "Ultimate":3
5
```

Both plans show the same picture: these distributions are positively skewed, standard deviation is quite large, almost the same as the mean. There are slightly - around 10 messages per month - less in the "Surf" plan which is only logical as users of this plan have a lower limit, so they are probably trying to stay inside the boundaries by sending less messages.

In [61]:

```
print('People exceeded the "Surf" plan limit for messages: {:.0f}'.format(len(df
_surf.query('num_messages_month > 50'))))
print('Ratio of people exceeded the "Surf" plan limit for messages: {:.0%}'.form
at(len(df_surf.query('num_messages_month > 50')) / len(df_surf)))
```

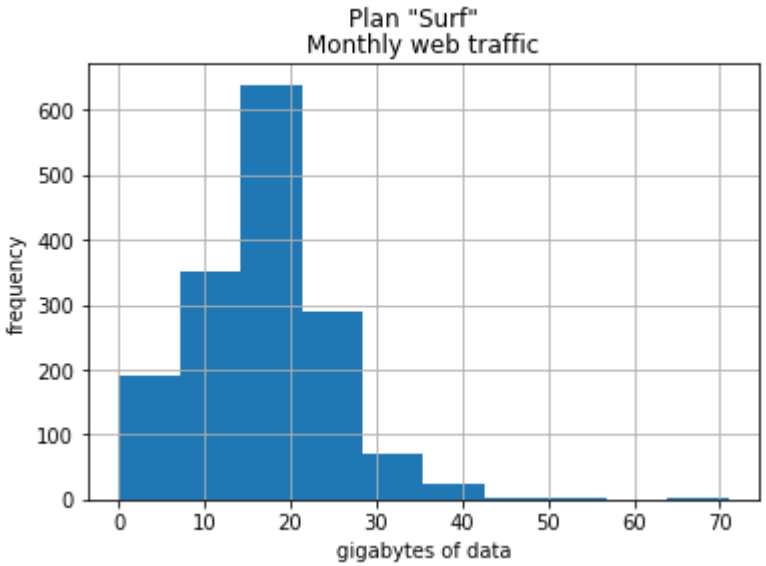
```
People exceeded the "Surf" plan limit for messages: 340
Ratio of people exceeded the "Surf" plan limit for messages: 22%
```

The "Surf" plan limit is 50 messages per month. Most users stay within this limit while others - 340 people (22%) - exceed it. Those who exceed the limit create profit for the company. The "Ultimate" plan limit is 1000 messages per month. No user is sending that many texts - the highest number is only 150 messages per month. This is where the company looses much profit, so this limit must be reviewed.

Internet

In [62]:

```
df_surf.hist('gb_volume_month')
plt.title('Monthly web traffic')
plt.suptitle('Plan "Surf"')
plt.xlabel('gigabytes of data')
plt.ylabel('frequency');
```



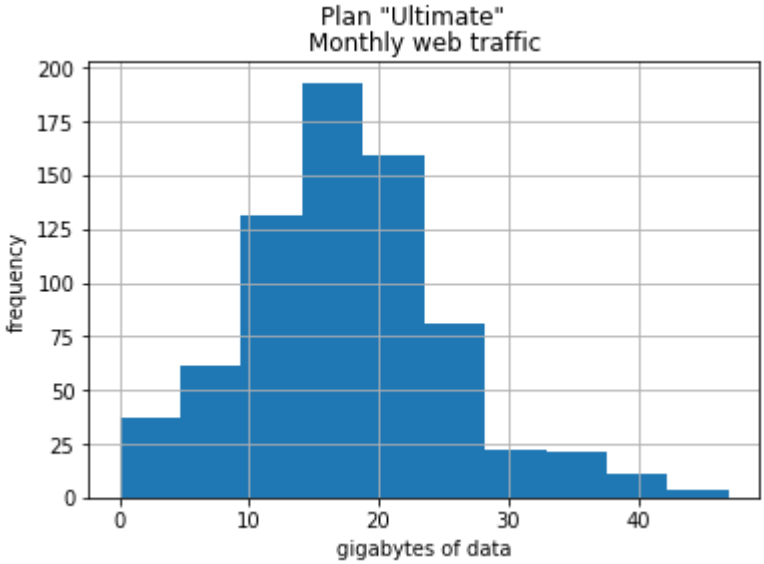
In [63]:

```
print('Average monthly volume of data, plan "Surf": {:.0f}'.format(df_surf['gb_volume_month'].mean()))
print('Medium monthly volume of data, plan "Surf":{:.0f}'.format(df_surf['gb_volume_month'].median()))
print('Standard deviation for monthly volume of data, plan "Surf":{:.0f}'.format(df_surf['gb_volume_month'].std()))
```

```
Average monthly volume of data, plan "Surf": 17
Medium monthly volume of data, plan "Surf":17
Standard deviation for monthly volume of data, plan "Surf":8
```

In [64]:

```
df_ultimate.hist('gb_volume_month')
plt.title('Monthly web traffic')
plt.suptitle('Plan "Ultimate"')
plt.xlabel('gigabytes of data')
plt.ylabel('frequency');
```



In [65]:

```
print('Average monthly volume of data, plan "Ultimate": {:.0f}'.format(df_ultima
te['gb_volume_month'].mean()))
print('Medium monthly volume of data, plan "Ultimate":{:.0f}'.format(df_ultimate
['gb_volume_month'].median()))
print('Standard deviation for monthly volume of data, plan "Ultimate":{:.0f}'.fo
rmat(df_ultimate['gb_volume_month'].std()))
```

```
Average monthly volume of data, plan "Ultimate": 18
Medium monthly volume of data, plan "Ultimate":17
Standard deviation for monthly volume of data, plan "Ultimate":8
```

The distributions of both plans are very similar: measures of location and dispersion are almost identical. The mean and median are very close (around 17-18 gb per month) which is a sign of a distribution close to normal. Standard deviation is not so large either - 8 gb. The number of observations with a mean/median volume is more than twice as large as the rest of values, that's why the picks of both histograms are so distinctive.

In [66]:

```
print('People exceeded the "Surf" plan limit for web traffic: {:.0f}'.format(len
(df_surf.query('gb_volume_month > 16'))))
print('Ratio of people exceeded the "Surf" plan limit for web traffic: {:.0%}'.f
ormat(len(df_surf.query('gb_volume_month > 16')) / len(df_surf)))
```

```
People exceeded the "Surf" plan limit for web traffic: 858
Ratio of people exceeded the "Surf" plan limit for web traffic: 55%
```

In [67]:

```
print('People exceeded the "Ultimate" plan limit for web traffic: {:.0f}'.format
(len(df_ultimate.query('gb_volume_month > 31'))))
print('Ratio of people exceeded the "Ultimate" plan limit for web traffic: {:.
0%}'.format(len(df_ultimate.query('gb_volume_month > 31')) / len(df_surf)))
```

```
People exceeded the "Ultimate" plan limit for web traffic: 39
Ratio of people exceeded the "Ultimate" plan limit for web traffic:
2%
```

The "Surf" plan's data limit is 16 gb which is slightly less than the average usage. Almost a half of users stay within this limit while others - 858 people (55%) - exceed it. Those who exceed the limit create profit for the company. As the number of people who exceed the limit is significant, profit from the internet source is probably the biggest among other sources for this plan.

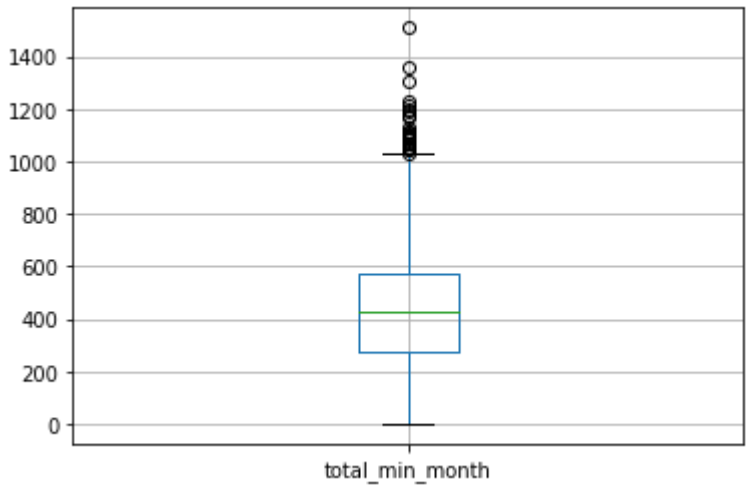
The "Ultimate" plan's data limit is 31 bg. There are only 39 people (2%) who exceeded this number, hence the data limit for this plan should be reconsidered.

Outliers

Calls

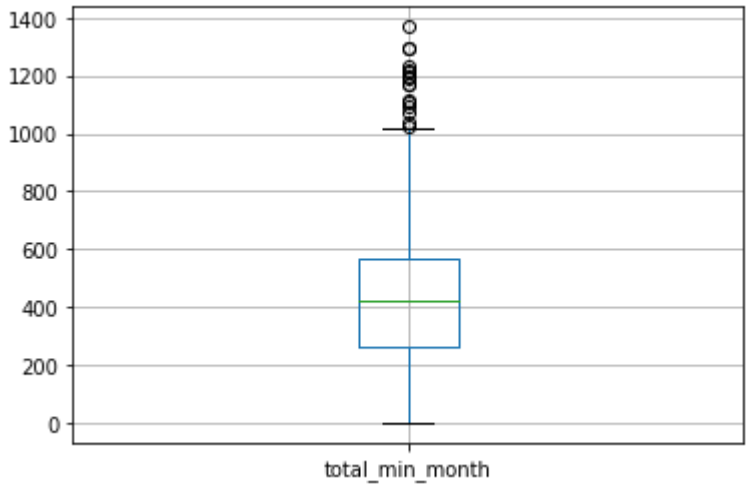
In [68]:

```
df_surf.boxplot('total_min_month');
```



In [69]:

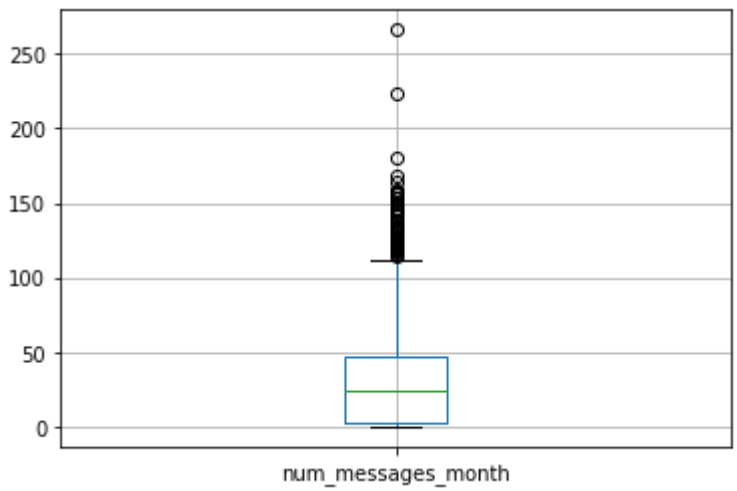
```
df_ultimate.boxplot('total_min_month');
```



Messages

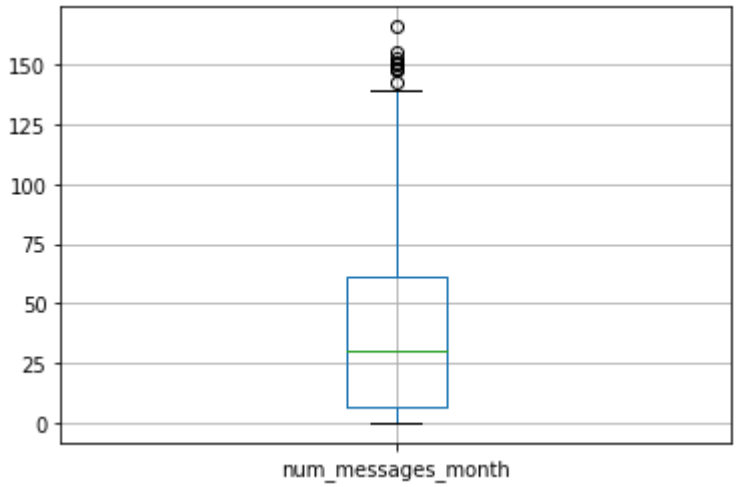
In [70]:

```
df_surf.boxplot('num_messages_month');
```



In [71]:

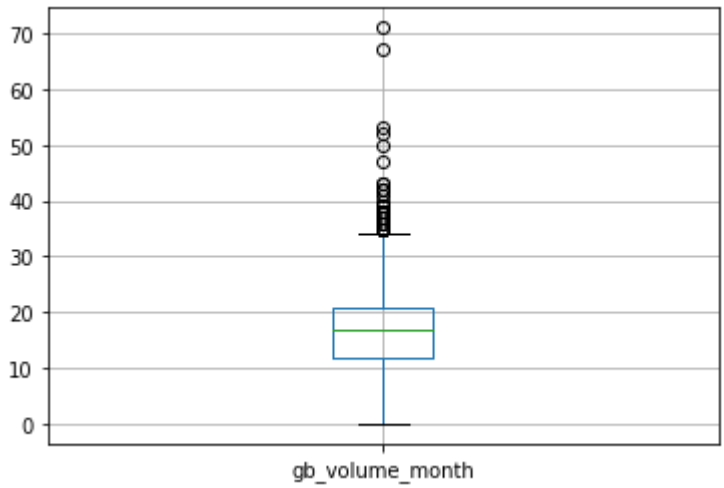
```
df_ultimate.boxplot('num_messages_month');
```



Internet

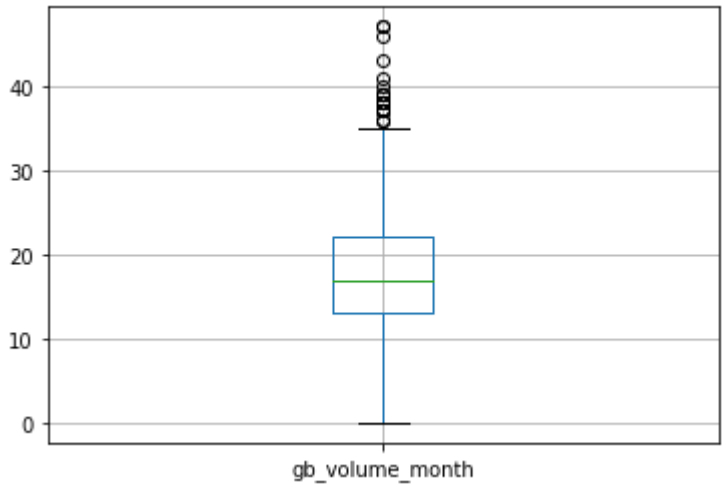
In [72]:

```
df_surf.boxplot('gb_volume_month');
```



In [73]:

```
df_ultimate.boxplot('gb_volume_month');
```



According to the above boxplots, we see that distributions for both plans and for both `total_min_month` and `gb_volume_month` variables are close to normal, whereas it is positively skewed for the `num_messages_month` variable. However we only see outliers above the upper whisker for all 3 variables and both plans. We will hence apply the Interquartile range method in order to remove them.

Outliers removal

First, let's calculate the upper whisker for each plan and for each variable.

In [74]:

```
df_surf.name = "Surf"
df_ultimate.name = "Ultimate"
for df in [df_surf,df_ultimate]:
    print()
    print(df.name)
    for feature in ['total_min_month','gb_volume_month','num_messages_month']:
        df[feature] = df[feature].astype(int)
        q25 = df[feature].quantile(0.25)
        q75 = df[feature].quantile(0.75)
        iqr = q75 - q25
        # calculate the outlier cutoff and upper limit
        cut_off = iqr * 1.5
        upper = q75 + cut_off
        print(feature, upper)
```

```
Surf
total_min_month 1032.0
gb_volume_month 34.5
num_messages_month 113.0

Ultimate
total_min_month 1023.125
gb_volume_month 35.5
num_messages_month 142.0
```

Next, we will only keep those rows for which each variable is under the upper limit.

In [75]:

```
df_surf = df_surf[(df_surf['total_min_month']<1032.0) & (df_surf['gb_volume_mont
h']<34.5) & (df_surf['num_messages_month']<113.0)]
df_surf.shape
```

Out[75]:

```
(1471, 17)
```

In [76]:

```
df_ultimate = df_ultimate[(df_ultimate['total_min_month']<1023.125) & (df_ultima
te['gb_volume_month']<35.5) & (df_ultimate['num_messages_month']<142.0)]
df_ultimate.shape
```

Out[76]:

```
(671, 17)
```

Statistical hypotheses testing

The average profit from users of Ultimate and Surf plans differs: bilateral hypothesis

Step 1: the null and alternative hypotheses

H0: The means of two statistical populations are equal. In our case it means that the average profit from users of "Surf" and "Ultimate" plans is the same.

H1: The means of two statistical populations are not equal. In our case it means that the average profit from users of "Surf" and "Ultimate" plans differs, although we do not specify here which one is more.

Step 2: Set the criteria for a decision

In behavioral science, the level of significance is typically set at 5% and we will choose this criteria as well. When the probability of obtaining a sample mean is less than 5% if the null hypothesis were true, then we reject the value stated in the null hypothesis.

Step 3: Compute the test statistic

In order to test our hypothesis that the means of two statistical populations are equal based on samples taken from them, we will apply the method `scipy.stats.ttest_ind()`.

The method takes the following parameters:

- `array1`, `array2` are arrays containing the samples. We will use the `monthly_profit` variables we calculated earlier for both plans;
- `equal_var` is an optional parameter that specifies whether or not the variances of the populations should be considered equal. To set this parameter let's test whether the variances of our samples are the same.

In [77]:

```
sample_1 = df_surf['monthly_profit']
sample_2 = df_ultimate['monthly_profit']
```

In [78]:

```
st.levene(sample_1, sample_2)
```

Out[78]:

```
LeveneResult(statistic=98.0356634273378, pvalue=1.24925160887314e-2
2)
```

The small p-value suggests that the populations do not have equal variances. Hence we will set the `equal_var` parameter to False.

In [79]:

```
alpha = .05 # critical statistical significance level
                        # if the p-value is less than alpha, we reject the hypot
hesis

results = st.ttest_ind(
                        sample_1,
                        sample_2,
                        equal_var=False)

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

if (results.pvalue < alpha):
        print("We reject the null hypothesis")
else:
        print("We retain the null hypothesis")
```

```
p-value:  6.754263687330053e-181
We reject the null hypothesis
```

Step 4: Make a decision

Based on the results of the test statistic we reach **significance**: the decision is to **reject the null hypothesis**. The equality of samples' means is associated with a low probability of occurrence (much less than 1%) when the null hypothesis is true.

Now let's see if this decision stands if we reformulate the alternative hypothesis, so it becomes unilateral.

The average profit from users of Ultimate and Surf plans differs: unilateral hypothesis

The "Surf" plan's average profit is greater than the "Ultimate" plan's average profit

Step 1: the null and alternative hypotheses

H0: The means of two statistical populations are equal. In our case it means that the average profit from users of "Surf" and "Ultimate" plans is the same.

H1: One of the means of two statistical populations is greater than the other. In our case it means that the average profit from users of the "Surf" plan is greater than that of the "Ultimate" plan.

Step 2: Set the criteria for a decision

We will keep the level of significance at 5% for this test as well.

Step 3: Compute the test statistic

In order to test our new hypothesis we will also apply the method `scipy.stats.ttest_ind()`. The first 3 parameters stay the same and we will add one more for unilateral hypotheses:

- `alternative` is an optional parameter that defines the alternative hypothesis. The following options are available (default is 'two-sided'). In this section we will set it to 'greater' for our purposes.

In [80]:

```
alpha = .05 # critical statistical significance level
                        # if the p-value is less than alpha, we reject the hypot
hesis

results = st.ttest_ind(
                        sample_1,
                        sample_2,
                        equal_var=False,
                        alternative='greater')

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

if (results.pvalue < alpha):
        print("We reject the null hypothesis")
else:
        print("We retain the null hypothesis")
```

```
p-value:  3.3771318436650265e-181
We reject the null hypothesis
```

Step 4: Make a decision

Based on the results of the test statistic we reach **significance**: the decision is to **reject the null hypothesis**. The equality of samples' means is associated with a low probability of occurrence (much less than 1%) when the null hypothesis is true.

Now let's see what results we would get if we changed again the alternative hypothesis.

The average profit from users of Ultimate and Surf plans differs: unilateral hypothesis

The "Surf" plan's average profit is less than the "Ultimate" plan's average profit

Step 1: the null and alternative hypotheses

H0: The means of two statistical populations are equal. In our case it means that the average profit from users of "Surf" and "Ultimate" plans is the same.

H1: One of the means of two statistical populations is less than the other. In our case it means that the average profit from users of the "Surf" plan is less than that of the "Ultimate" plan.

Step 2: Set the criteria for a decision

We will keep the level of significance at 5% for this test as well.

Step 3: Compute the test statistic

In order to test our new hypothesis we will also apply the method `scipy.stats.ttest_ind()`. The first 3 parameters stay the same and we will add one more for unilateral hypotheses:

- `alternative` is an optional parameter that defines the alternative hypothesis. The following options are available (default is ‘two-sided’). In this section we will set it to 'less' for our purposes.

In [81]:

```
alpha = .05 # critical statistical significance level
                        # if the p-value is less than alpha, we reject the hypot
hesis

results = st.ttest_ind(
                        sample_1,
                        sample_2,
                        equal_var=False,
                        alternative='less')

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

if (results.pvalue < alpha):
        print("We reject the null hypothesis")
else:
        print("We retain the null hypothesis")
```

```
p-value:  1.0
We retain the null hypothesis
```

Step 4: Make a decision

Based on the results of the test statistic we **failed to reach significance**: the decision is to **retain the null hypothesis**. The equality of samples' means against such an alternative hypothesis is associated with a 100% probability of occurrence when the null hypothesis is true.

This decision is probably an example of a Type II error. With each test we make, there is always some probability that the decision could be a Type II error. In this decision, we decide to retain previous notions of truth that are in fact false (based on previous analysis). While it's an error, we still did nothing; we retained the null hypothesis. We can always go back and conduct more studies.

The average profit from users in NY-NJ area is different from that of the users from other regions: bilateral hypothesis

First, let's from two populations according to the task.

In [82]:

```
ny_nj_users = df_monthly[df_monthly['city'] == 'New York-Newark-Jersey City, NY-
NJ-PA MSA']
```

In [83]:

```
other_regions_users = df_monthly[df_monthly['city'] != 'New York-Newark-Jersey C
ity, NY-NJ-PA MSA']
```

In [84]:

```
other_regions_users.shape
```

Out[84]:

```
(1916, 17)
```

In [85]:

```
ny_nj_users.shape
```

Out[85]:

```
(377, 17)
```

Step 1: the null and alternative hypotheses

H0: The means of two statistical populations are equal. In our case it means that the average profit from users in NY-NJ area and users in other regions is the same.

H1: The means of two statistical populations are not equal. In our case it means that the average profit from users in NY-NJ area and users in other regions differs, although we do not specify here which one is more.

Step 2: Set the criteria for a decision

We will keep the level of significance at 5% for this test as well.

Step 3: Compute the test statistic

In order to test our new hypothesis we will again apply the method scipy.stats.ttest_ind() but first let's test whether variances of these two populations are similar.

In [86]:

```
sample_3 = ny_nj_users['monthly_profit']
sample_4 = other_regions_users['monthly_profit']

st.levene(sample_3, sample_4)
```

Out[86]:

```
LeveneResult(statistic=6.608211517749522, pvalue=0.01021358406972762
4)
```

The small p-value (less that 5%) suggests that the populations do not have equal variances. Hence we will set the `equal_var` parameter to False.

In [87]:

```
alpha = .05 # critical statistical significance level
                        # if the p-value is less than alpha, we reject the hypot
hesis

results = st.ttest_ind(
                        sample_3,
                        sample_4,
                        equal_var=False)

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

if (results.pvalue < alpha):
        print("We reject the null hypothesis")
else:
        print("We retain the null hypothesis")
```

```
p-value:  0.0004452925660048938
We reject the null hypothesis
```

Step 4: Make a decision

Based on the results of the test statistic we reach **significance**: the decision is to **reject the null hypothesis**. The equality of samples' means is associated with a low probability of occurrence (much less than 1%) when the null hypothesis is true.

Now let's see if this decision stands if we reformulate the alternative hypothesis, so it becomes unilateral.

The average profit from users of Ultimate and Surf plans differs: unilateral hypothesis

The "Surf" plan's average profit is greater than the "Ultimate" plan's average profit

Step 1: the null and alternative hypotheses

H0: The means of two statistical populations are equal. In our case it means that the average profit from users in NY-NJ area and users in other regions is the same.

H1: One of the means of two statistical populations is greater than the other. In our case it means that the average profit from users in NY-NJ area is greater than that of users in other areas.

Step 2: Set the criteria for a decision

We will keep the level of significance at 5% for this test as well.

Step 3: Compute the test statistic

In order to test our new hypothesis we will also apply the method `scipy.stats.ttest_ind()`. The first 3 parameters stay the same and we will add one more for unilateral hypotheses:

- `alternative` is an optional parameter that defines the alternative hypothesis. The following options are available (default is 'two-sided'). In this section we will set it to 'greater' for our purposes.

In [88]:

```
alpha = .05 # critical statistical significance level
                        # if the p-value is less than alpha, we reject the hypot
hesis

results = st.ttest_ind(
                        sample_3,
                        sample_4,
                        equal_var=False,
                        alternative='greater')

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

if (results.pvalue < alpha):
        print("We reject the null hypothesis")
else:
        print("We retain the null hypothesis")
```

```
p-value:  0.0002226462830024469
We reject the null hypothesis
```

Step 4: Make a decision

Based on the results of the test statistic we reach **significance**: the decision is to **reject the null hypothesis**. The equality of samples' means is associated with a low probability of occurrence (much less than 1%) when the null hypothesis is true.

Now let's see what results we would get if we changed again the alternative hypothesis.

The average profit from users of Ultimate and Surf plans differs: unilateral hypothesis

The "Surf" plan's average profit is less than the "Ultimate" plan's average profit

Step 1: the null and alternative hypotheses

H0: The means of two statistical populations are equal. In our case it means that the average profit from users of "Surf" and "Ultimate" plans is the same.

H1: One of the means of two statistical populations is less than the other. In our case it means that the average profit from users of the "Surf" plan is less than that of the "Ultimate" plan.

Step 2: Set the criteria for a decision

We will keep the level of significance at 5% for this test as well.

Step 3: Compute the test statistic

In order to test our new hypothesis we will also apply the method `scipy.stats.ttest_ind()`. The first 3 parameters stay the same and we will add one more for unilateral hypotheses:

- `alternative` is an optional parameter that defines the alternative hypothesis. The following options are available (default is ‘two-sided’). In this section we will set it to 'less' for our purposes.

In [89]:

```
alpha = .05 # critical statistical significance level
                        # if the p-value is less than alpha, we reject the hypot
hesis

results = st.ttest_ind(
                        sample_3,
                        sample_4,
                        equal_var=False,
                        alternative='less')

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

if (results.pvalue < alpha):
        print("We reject the null hypothesis")
else:
        print("We retain the null hypothesis")
```

```
p-value:  0.9997773537169975
We retain the null hypothesis
```

Step 4: Make a decision

Based on the results of the test statistic we **failed to reach significance**: the decision is to **retain the null hypothesis**. The equality of samples' means against such an alternative hypothesis is associated with a 100% probability of occurrence when the null hypothesis is true.

This decision is probably an example of a Type II error as well.

Conclusion

In this report we have analyzed telecom clients' behavior in order to determine which of the two prepaid plans is more profitable and to test two statistical hypotheses.

First of all, we have familiarized ourselves with the data by performing the descriptive statistics. Based on that analysis, in the preprocessing step we have changed data type of all the date variables to datetime format.

Then we have rounded all the calls to minutes as per the company's policy.

Next, we filled some missing values in the `churn_date` column, assuming these clients are still using the service, that is why there is no exit date for them.

We have then checked for duplicates and any artifacts in user dates (we made sure the `churn_date` was always later than the `reg_date` and also that all user activity was placed between these two dates). No issues were found in these sections.

Next step was to perform a few calculations. For each user, we found:

- Total minutes used per month;
- The number of text messages sent per month;
- The volume of data per month;
- The monthly profit from each user (we subtracted the free package limit from the total number of calls, text messages, and data; multiplied the result by the numbers included in the plan and added the monthly plan's charge)

In the following section we have performed an exploratory data analysis in order to analyze customer's behavior for each plan and then compare both plans.

We found that the "Surf" plan is mostly profitable throughout the year with a tendency of being more and more profit-making towards the end of the year. In contract, the "Ultimate" plan is mostly not profitable during the year, with the opposite tendency of being even more costly by the end of the year.

Distributions for calls and web traffic are close to normal with the mean/median value for calls of 430-440 minutes per month and for web traffic - of 17-18 gb per month for both plans. The distributions for messages are positively skewed with a quite large standard deviation. It means that some people sent large amount of messages compared to the mean number of messages.

As for the plans' conditions, the "Surf" plan brings some profit because users often exceed the limits. In contract, the "Ultimate" plan's limits should be reconsidered as users quite rarely (in 0% for calls and messages and 2% for web traffic cases) reach the limits and usually are far away from them.

Next step was statistical hypotheses testing. We tested two hypotheses:

1. The average profit from users of Ultimate and Surf plans differs;
2. The average profit from users in NY-NJ area is different from that of the users from other regions.

For both cases we have tested a bilateral and 2 unilateral hypotheses:

- bilateral: in both cases we have rejected the null - each pair of populations does not bring the same average profit to the company;
- unilateral:
  - 'greater': in both cases we have rejected the nulls in favor of the alternative hypotheses:
    - The "Surf" plan brings more profit than "Ultimate", on average;

    - Users from NY-NJ area bring more profit than users from other regions, on average.
- 'less': in both cases we have retained the nulls over the alternative hypotheses:
    - The "Surf" plan does not bring less profit than "Ultimate", on average;
    - Users from NY-NJ area does not bring less profit than users from other regions, on average.