# Статистический анализ данных

## 1 Запрос бизнеса

Вы аналитик компании «Мегалайн» — федерального оператора сотовой связи.
Клиентам предлагают два тарифных плана: «Смарт» и «Ультра». Чтобы скорректировать рекламный бюджет, коммерческий департамент хочет понять, **какой тариф приносит больше денег**.

Вам предстоит сделать предварительный анализ тарифов на небольшой выборке клиентов. В вашем распоряжении данные 500 пользователей «Мегалайна»: кто они, откуда, каким тарифом пользуются, сколько звонков и сообщений каждый отправил за 2018 год. Нужно проанализировать поведение клиентов и сделать вывод — **какой тариф лучше**.

## 2 Описание данных

#### 2.0.1 Таблица users (информация о пользователях):

user_id — уникальный идентификатор пользователя
first_name — имя пользователя
last_name — фамилия пользователя
age — возраст пользователя (годы)
reg_date — дата подключения тарифа (день, месяц, год)
churn_date — дата прекращения пользования тарифом (если значение пропущено, то тариф ещё действовал на момент выгрузки данных)
city — город проживания пользователя
tarif — название тарифного плана

#### 2.0.2 Таблица calls (информация о звонках):

id — уникальный номер звонка
call_date — дата звонка
duration — длительность звонка в минутах
user_id — идентификатор пользователя, сделавшего звонок

#### 2.0.3 Таблица messages (информация о сообщениях):

id — уникальный номер звонка
message_date — дата сообщения
user_id — идентификатор пользователя, отправившего сообщение

#### 2.0.4 Таблица internet (информация об интернет-сессиях):

id — уникальный номер сессии
mb_used — объём потраченного за сессию интернет-трафика (в мегабайтах)
session_date — дата интернет-сессии
user_id — идентификатор пользователя

#### 2.0.5 Таблица tariffs (информация о тарифах):

tariff_name — название тарифа
rub_monthly_fee — ежемесячная абонентская плата в рублях
minutes_included — количество минут разговора в месяц, включённых в абонентскую плату
messages_included — количество сообщений в месяц, включённых в абонентскую плату

mb_per_month_included — объём интернет-трафика, включённого в абонентскую плату (в мегабайтах)
rub_per_minute — стоимость минуты разговора сверх тарифного пакета (например, если в тарифе 100 минут разговора в месяц, то со 101 минуты будет взиматься плата)
rub_per_message — стоимость отправки сообщения сверх тарифного пакета
rub_per_gb — стоимость дополнительного гигабайта интернет-трафика сверх тарифного пакета (1 гигабайт = 1024 мегабайта)

Примечание. Если объединение таблиц командой merge приводит к ошибке dead kernell, примените join.

# 3 Изучение данных из файла

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# importing a library for operating with a data type "NaN"
import numpy as np
from scipy import stats as st
```

In [2]:
```
# отключаем некритичные уведомления
import warnings
warnings.filterwarnings('ignore')
# показывать до 40ка колонок
pd.set_option('display.max.columns', 40)
# установка формата вывода на дисплей численных значений
pd.options.display.float_format = '{:,.2f}'.format
# Библиотека для отображения картинок
from IPython.display import Image
# бибилиотека для укоругления в бОльшую сторону
import math
```

считываем файлы

In [3]:
```
try:
    users = pd.read_csv('/dat...rs.csv') # Yandex path
    tariffs = pd.read_csv('/dat...ffs.csv')
    messages = pd.read_csv('/dat..ages.csv')
    internet = pd.read_csv('/dat...net.csv')
    calls = pd.read_csv('/data..ls.csv')

except:
    users = pd.read_csv(r"C:\Users\eddyd..ers.csv") # personal path
    tariffs = pd.read_csv(r"C:\Users\edd..ariffs.csv")
    messages = pd.read_csv(r"C:\Users\ed...ges.csv")
    internet = pd.read_csv(r"C:\Users\e...rnet.csv")
    calls = pd.read_csv(r"C:\Users\eddyd\Down...ls.csv")

```

## 3.1 Рассмотрим *users *

In [4]:
```
1 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   age         500 non-null    int64 
 2   churn_date  38 non-null     object
 3   city        500 non-null    object
 4   first_name  500 non-null    object
 5   last_name   500 non-null    object
 6   reg_date    500 non-null    object
 7   tariff      500 non-null    object
dtypes: int64(2), object(6)
memory usage: 31.4+ KB
```

In [5]:
```
1 users.describe()
```

Out[5]:

| | user_id | age |
|---|---|---|
| **count** | 500.00 | 500.00 |
| **mean** | 1,249.50 | 46.59 |
| **std** | 144.48 | 16.67 |
| **min** | 1,000.00 | 18.00 |
| **25%** | 1,124.75 | 32.00 |
| **50%** | 1,249.50 | 46.00 |
| **75%** | 1,374.25 | 62.00 |
| **max** | 1,499.00 | 75.00 |

In [6]:
```
1 users.head()
```

Out[6]:

| | user_id | age | churn_date | city | first_name | last_name | reg_date | tariff |
|---|---|---|---|---|---|---|---|---|
| **0** | 1000 | 52 | NaN | Краснодар | Рафаил | Верещагин | 2018-05-25 | ultra |
| **1** | 1001 | 41 | NaN | Москва | Иван | Ежов | 2018-11-01 | smart |
| **2** | 1002 | 59 | NaN | Стерлитамак | Евгений | Абрамович | 2018-06-17 | smart |
| **3** | 1003 | 23 | NaN | Москва | Белла | Белякова | 2018-08-17 | ultra |
| **4** | 1004 | 68 | NaN | Новокузнецк | Татьяна | Авдеенко | 2018-05-14 | ultra |

### 3.1.1 предварительно о users

- проверить user_id на уникальность
- age - данные хорошие. от 18 до 75 лет
- тип данных для age и user_id - целочисленные, что верно
- reg_data и churn_date следует **перевести** в формат даты
- проверить на неявные дубликаты **city** и **tariff**

### 3.1.2 Проверим user_id на уникальность

Подсчитаем уникальные значения
Применив функцию len()

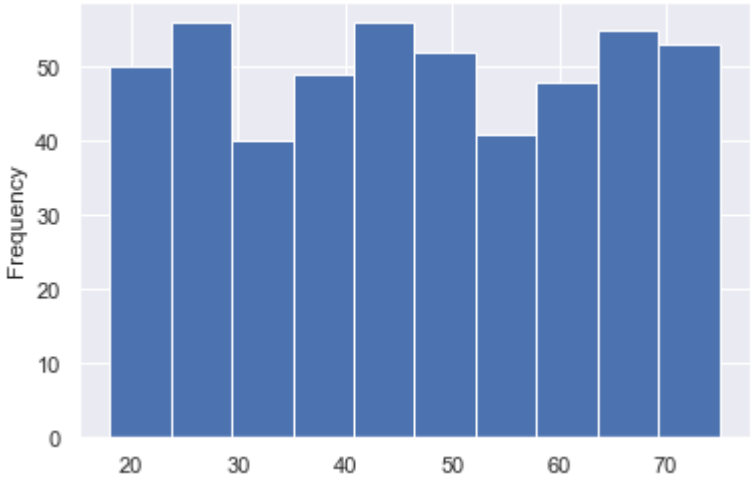
In [7]:
```
1 len(users.user_id.unique())
```

Out[7]: 500

В колоке user_id - **уникальные** значения (без повторов), так как количество уникальных значений совпадает с числом записей.

### 3.1.3 График распределения данных о возрасте

In [8]:
```
1 # setup size plot
2 sns.set(rc = {'figure.figsize':(10,5)})
3
4 users.age.plot(kind='hist');
```




В данных есть информация по всем возрастам почти в одинаковом количестве строк на каждый

### 3.1.4 переводим reg_data и churn_date в формат даты

Исходный формат **reg_data** 2018-05-25

In [9]:
```
1 users['reg_date'] = pd.to_datetime(users['reg_date'], format='%Y-%m-%d')
```

Смотрим, что получилось

In [10]:
```
1 users.reg_date.head()
```

Out[10]:
```
0   2018-05-25
1   2018-11-01
2   2018-06-17
3   2018-08-17
4   2018-05-14
Name: reg_date, dtype: datetime64[ns]
```

### 3.1.5 Значения успешно переведены во внутренний формат Питона datetime64

Посмотрим, какой исходный формат **churn_date** Для этого выведем строки без пропусков значений

In [11]:
```
# delete the lines with NaN and then read 5 lines
users_not_nan = users.dropna(subset= ['churn_date'])
print(users_not_nan['churn_date'].head())
```

```
19    2018-10-05
20    2018-12-18
34    2018-11-21
50    2018-10-03
51    2018-10-14
Name: churn_date, dtype: object
```

Исходный формат **churn_date** 2018-05-25

In [12]:
```
users['churn_date'] = pd.to_datetime(users['churn_date'], format='%Y-%m-%d')
```

Смотрим, что получилось

In [13]:
```
# delete the lines with NaN and then read 5 lines
users_not_nan = users.dropna(subset= ['churn_date'])
print(users_not_nan['churn_date'].head())
```

```
19   2018-10-05
20   2018-12-18
34   2018-11-21
50   2018-10-03
51   2018-10-14
Name: churn_date, dtype: datetime64[ns]
```

#### 3.1.6 Значения успешно переведены во внутренний формат Питона datetime64

#### 3.1.7 Проверим на неявные дубликаты city

Сортированный список уникальных значений

In [ ]:
```
sorted(users['city'].unique())
```

Вывод удалён из-за длинного принта

#### 3.1.8 Неявные дубликаты в city отсутствуют

#### 3.1.9 Проверим на неявные дубликаты tariff

Сортированный список уникальных значений

In [15]:
```
sorted(users['tariff'].unique())
```

Out[15]:
```
['smart', 'ultra']
```

#### 3.1.10 Неявные дубликаты в tariff отсутствуют

Значения

- smart

- ultra

#### 3.1.11 Закончили предварительную обработку данных users

- age - данные хорошие. от 18 до 75 лет
- - информация есть для всех возрастов примерно в равных количествах
- тип данных для age и user_id - целочисленные, что верно
- reg_data и churn_date **перевели** в формат даты
- отсутствуют неявные дубликаты **city** и **tariff**
- tariff
- - smart
- - ultra

## 3.2 Рассмотрим **tariffs* *

In [16]:
```
1 tariffs.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   rub_monthly_fee       2 non-null      int64 
 4   rub_per_gb            2 non-null      int64 
 5   rub_per_message       2 non-null      int64 
 6   rub_per_minute        2 non-null      int64 
 7   tariff_name           2 non-null      object
dtypes: int64(7), object(1)
memory usage: 256.0+ bytes
```

В таблице две строки.
Выведем их полностью на экран

In [17]:
```
1 tariffs
```

Out[17]:

| | messages_included | mb_per_month_included | minutes_included | rub_monthly_fee | rub_per_gb | rub_p |
|---|---|---|---|---|---|---|
| **0** | 50 | 15360 | 500 | 550 | 200 | |
| **1** | 1000 | 30720 | 3000 | 1950 | 150 | |

Из структуры и характера данных в таблице **tariffs** логично применить их трансформацию в формат **словаря**. Это позволит в дальнейшем легко и красиво менять параметры в экспериментах.

#### 3.2.1 Создаём новый объект tariffs_dict

In [18]:
```
1 tariffs_dict = pd.DataFrame() # def table
2 tariffs_dict = tariffs # write table
3 tariffs_dict = tariffs_dict.set_index('tariff_name') # set index as tariff_name
4 tariffs_dict = tariffs_dict.to_dict() # transform to dict
```

In [19]:
```
1 tariffs_dict # print dict
```

Out[19]:
```
{'messages_included': {'smart': 50, 'ultra': 1000},
 'mb_per_month_included': {'smart': 15360, 'ultra': 30720},
 'minutes_included': {'smart': 500, 'ultra': 3000},
 'rub_monthly_fee': {'smart': 550, 'ultra': 1950},
 'rub_per_gb': {'smart': 200, 'ultra': 150},
 'rub_per_message': {'smart': 3, 'ultra': 1},
 'rub_per_minute': {'smart': 3, 'ultra': 1}}
```

In [20]:
```
1 # print value(mb_per_month_included for ultra) from dict
2 print(tariffs_dict.get('mb_per_month_included').get('ultra'))
```

```
30720
```

Словарь **tariffs_dict** успешно создан.
Метод чтения данных проверен

- print(tariffs_dict.get('mb_per_month_included').get('ultra'))

#### 3.2.2 Закончили предварительную обработку данных tariffs

Данные в отличном состоянии и готовы сразу к работе.

tariff_name — название тарифа

- smart
- ultra
  rub_monthly_fee — ежемесячная абонентская плата в рублях
  minutes_included — количество минут разговора в месяц, включённых в абонентскую плату
  messages_included — количество сообщений в месяц, включённых в абонентскую плату
  mb_per_month_included — объём интернет-трафика, включённого в абонентскую плату (в мегабайтах)
  rub_per_minute — стоимость минуты разговора сверх тарифного пакета (например, если в тарифе 100 минут разговора в месяц, то со 101 минуты будет взиматься плата)
  rub_per_message — стоимость отправки сообщения сверх тарифного пакета
  rub_per_gb — стоимость дополнительного гигабайта интернет-трафика сверх тарифного пакета (1 гигабайт = 1024 мегабайта)

#### 3.2.3 использовать будем словарь tariffs_dict

## 3.3 Рассмотрим **messages* *

In [21]:
```
1 messages.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123036 entries, 0 to 123035
Data columns (total 3 columns):
 #   Column        Non-Null Count   Dtype 
---  ------        --------------   ----- 
 0   id            123036 non-null  object
 1   message_date  123036 non-null  object
 2   user_id       123036 non-null  int64 
dtypes: int64(1), object(2)
memory usage: 2.8+ MB
```

**пропусков** данных **нет**

In [22]:
```
1 messages.head()
```

Out[22]:

| | id | message_date | user_id |
|---|---|---|---|
| **0** | 1000_0 | 2018-06-27 | 1000 |
| **1** | 1000_1 | 2018-10-08 | 1000 |
| **2** | 1000_2 | 2018-08-04 | 1000 |
| **3** | 1000_3 | 2018-06-16 | 1000 |
| **4** | 1000_4 | 2018-12-05 | 1000 |

Значения в **message_date** следует перевести в формат дат Питона
Делаем это:

In [23]:
```
1 messages['message_date'] = pd.to_datetime(messages['message_date'], format='%Y-
```

Проверим результат

In [24]:
```
1 messages.message_date.head()
```

Out[24]:
```
0   2018-06-27
1   2018-10-08
2   2018-08-04
3   2018-06-16
4   2018-12-05
Name: message_date, dtype: datetime64[ns]
```

#### 3.3.1 Значения успешно переведены во внутренний формат Питона datetime64

#### 3.3.2 Закончили предварительную обработку данных messages

Данные в отличном состоянии и готовы к работе.

id — уникальный номер звонка
message_date — дата сообщения
user_id — идентификатор пользователя, отправившего сообщение

## 3.4 Рассмотрим **internet* *

In [25]:
```
1 internet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149396 entries, 0 to 149395
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   Unnamed: 0    149396 non-null  int64  
 1   id            149396 non-null  object 
 2   mb_used       149396 non-null  float64
 3   session_date  149396 non-null  object 
 4   user_id       149396 non-null  int64  
dtypes: float64(1), int64(2), object(2)
memory usage: 5.7+ MB
```

**пропусков** данных **нет**

In [26]:
```
1 internet.head()
```

Out[26]:

| | Unnamed: 0 | id | mb_used | session_date | user_id |
|---|---|---|---|---|---|
| **0** | 0 | 1000_0 | 112.95 | 2018-11-25 | 1000 |
| **1** | 1 | 1000_1 | 1,052.81 | 2018-09-07 | 1000 |
| **2** | 2 | 1000_2 | 1,197.26 | 2018-06-25 | 1000 |
| **3** | 3 | 1000_3 | 550.27 | 2018-08-22 | 1000 |
| **4** | 4 | 1000_4 | 302.56 | 2018-09-24 | 1000 |

Колонка **Unnamed: 0** дублирует **index** и для анализа может не использоваться

- Поэтому **удалим** её
- Данные в колонке **session_date** переведём в формат даты Питона

In [27]:
```
1 internet.drop('Unnamed: 0', axis=1, inplace=True)
```

In [28]:
```
1 internet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149396 entries, 0 to 149395
Data columns (total 4 columns):
 #   Column        Non-Null Count   Dtype  
---  ------        --------------   -----  
 0   id            149396 non-null  object 
 1   mb_used       149396 non-null  float64
 2   session_date  149396 non-null  object 
 3   user_id       149396 non-null  int64  
dtypes: float64(1), int64(1), object(2)
memory usage: 4.6+ MB
```

Лишний столбец удалён успешно

In [29]:
```
1 internet['session_date'] = pd.to_datetime(internet['session_date'], format='%Y-
```

Проверим результат

In [30]:
```
1 internet.session_date.head()
```

Out[30]:
```
0   2018-11-25
1   2018-09-07
2   2018-06-25
3   2018-08-22
4   2018-09-24
Name: session_date, dtype: datetime64[ns]
```

**Успешно**

### 3.4.1 Закончили предварительную обработку данных internet

Данные в отличном состоянии и готовы к работе.

id — уникальный номер сессии
mb_used — объём потраченного за сессию интернет-трафика (в мегабайтах)
session_date — дата интернет-сессии
user_id — идентификатор пользователя

## 3.5 Рассмотрим **calls** *

In [31]:
```
1 calls.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202607 entries, 0 to 202606
Data columns (total 4 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   id         202607 non-null  object
 1   call_date  202607 non-null  object
 2   duration   202607 non-null  float64
 3   user_id    202607 non-null  int64
dtypes: float64(1), int64(1), object(2)
memory usage: 6.2+ MB
```

- **Пропусков** нет
- call_date следует **перевести** в формат даты Питона

In [32]:
```
1 calls.duration.describe()
```

Out[32]:
```
count   202,607.00
mean          6.76
std           5.84
min           0.00
25%           1.30
50%           6.00
75%          10.70
max          38.00
Name: duration, dtype: float64
```

В связи со **снижением** стоимости звонков в среднем по рынку, **длительности** разговоров значительно **превышают** одну минуту.

- duration - есть значения "0". Это **пропущенные** звонки.

In [33]:
```
1 calls.head()
```

Out[33]:

| | id | call_date | duration | user_id |
|---|---|---|---|---|
| **0** | 1000_0 | 2018-07-25 | 0.00 | 1000 |
| **1** | 1000_1 | 2018-08-17 | 0.00 | 1000 |
| **2** | 1000_2 | 2018-06-11 | 2.85 | 1000 |
| **3** | 1000_3 | 2018-09-21 | 13.80 | 1000 |
| **4** | 1000_4 | 2018-12-15 | 5.18 | 1000 |

### 3.5.1 Данные в колонке call_date переведём в формат даты Питона

In [34]:
```
1 calls['call_date'] = pd.to_datetime(calls['call_date'], format='%Y-%m-%d')
```

Проверим результат преобразования

In [35]:
```
1 calls.call_date.head()
```

Out[35]:
```
0   2018-07-25
1   2018-08-17
2   2018-06-11
3   2018-09-21
4   2018-12-15
Name: call_date, dtype: datetime64[ns]
```

### 3.5.2 Отлично! Данные в calls готовы к работе

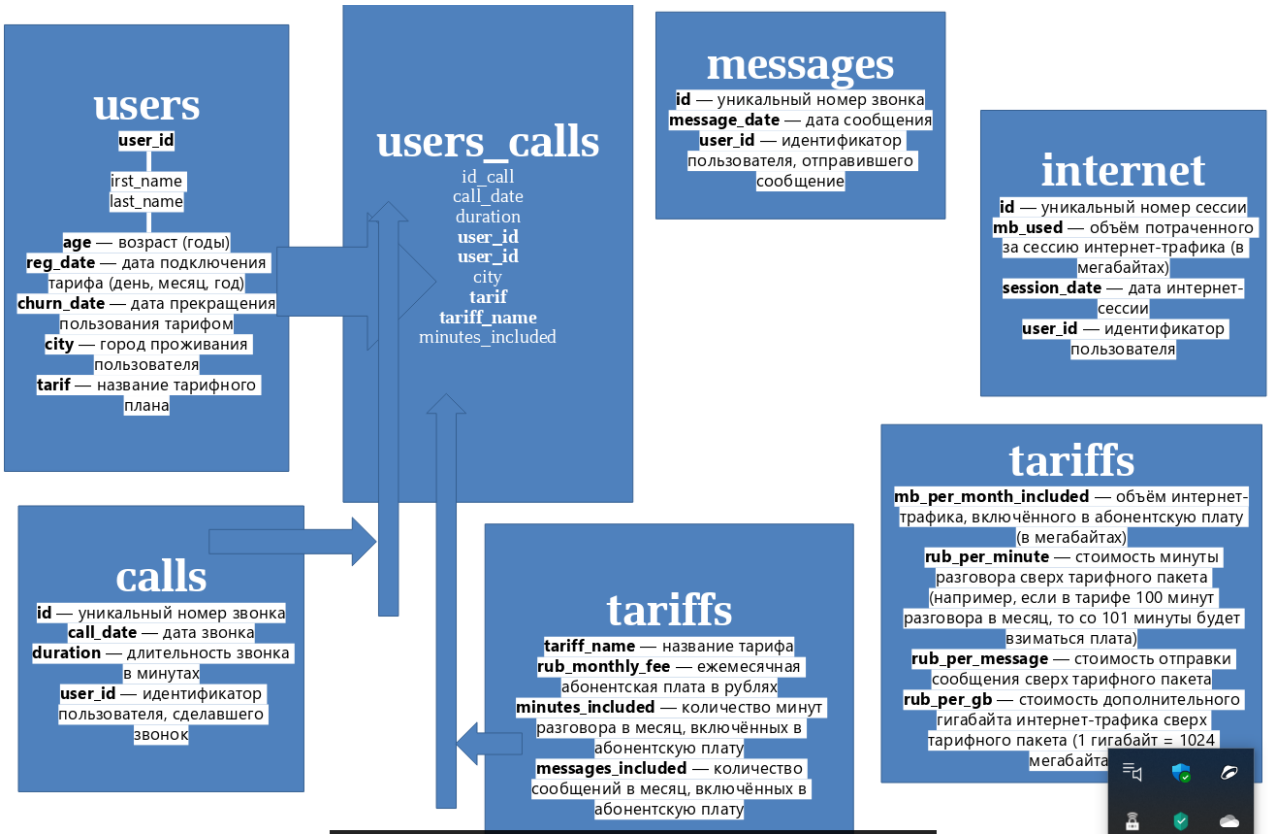
id — уникальный номер звонка call_date — дата звонка duration — длительность звонка в минутах

- "0" - пропущенный вызов user_id — идентификатор пользователя, сделавшего звонок

# 4 Проект сводной таблицы

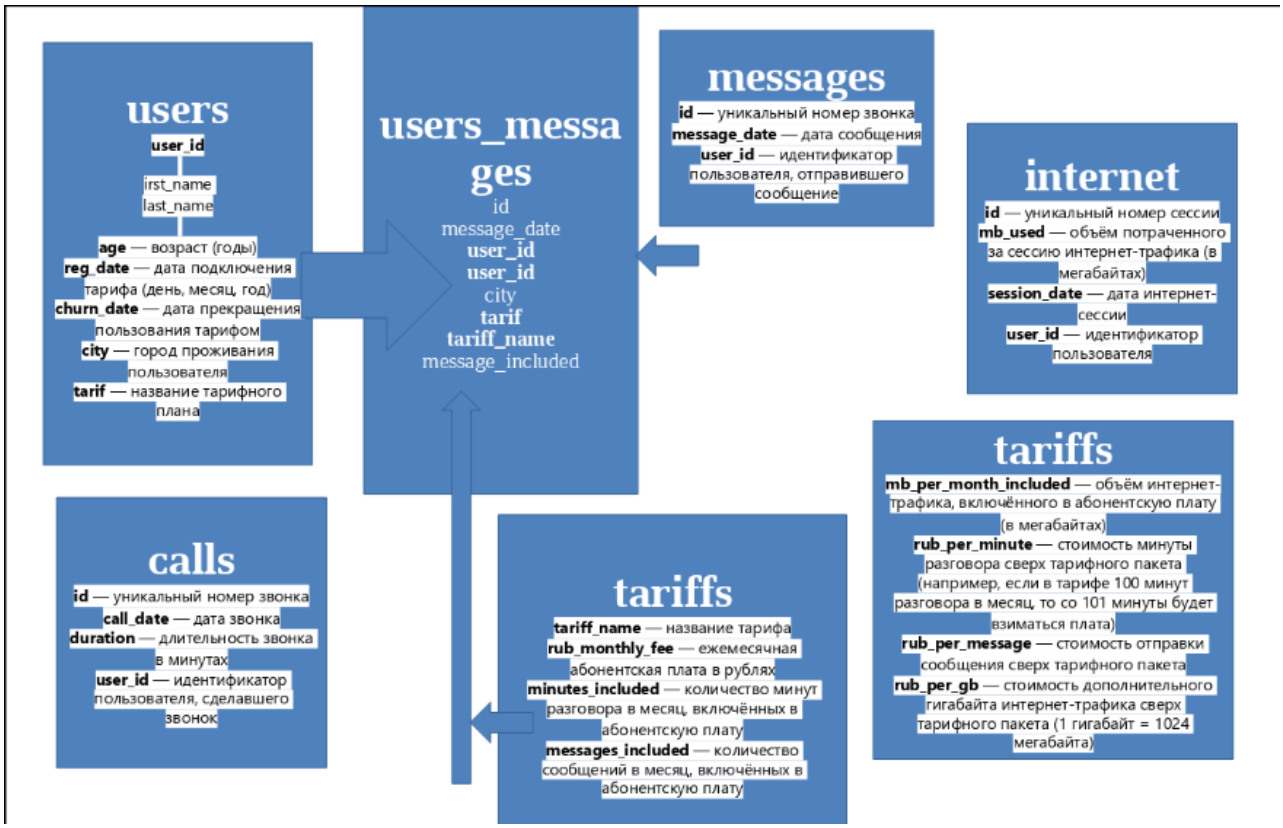
## 4.1 users_calls - для количество сделанных звонков и израсходованных минут разговора по месяцам

In [36]:
```
1 display(Image(url='http://dl3.joxi.net/drive/2022/02/07/0052/0684/3465900/00/14
```



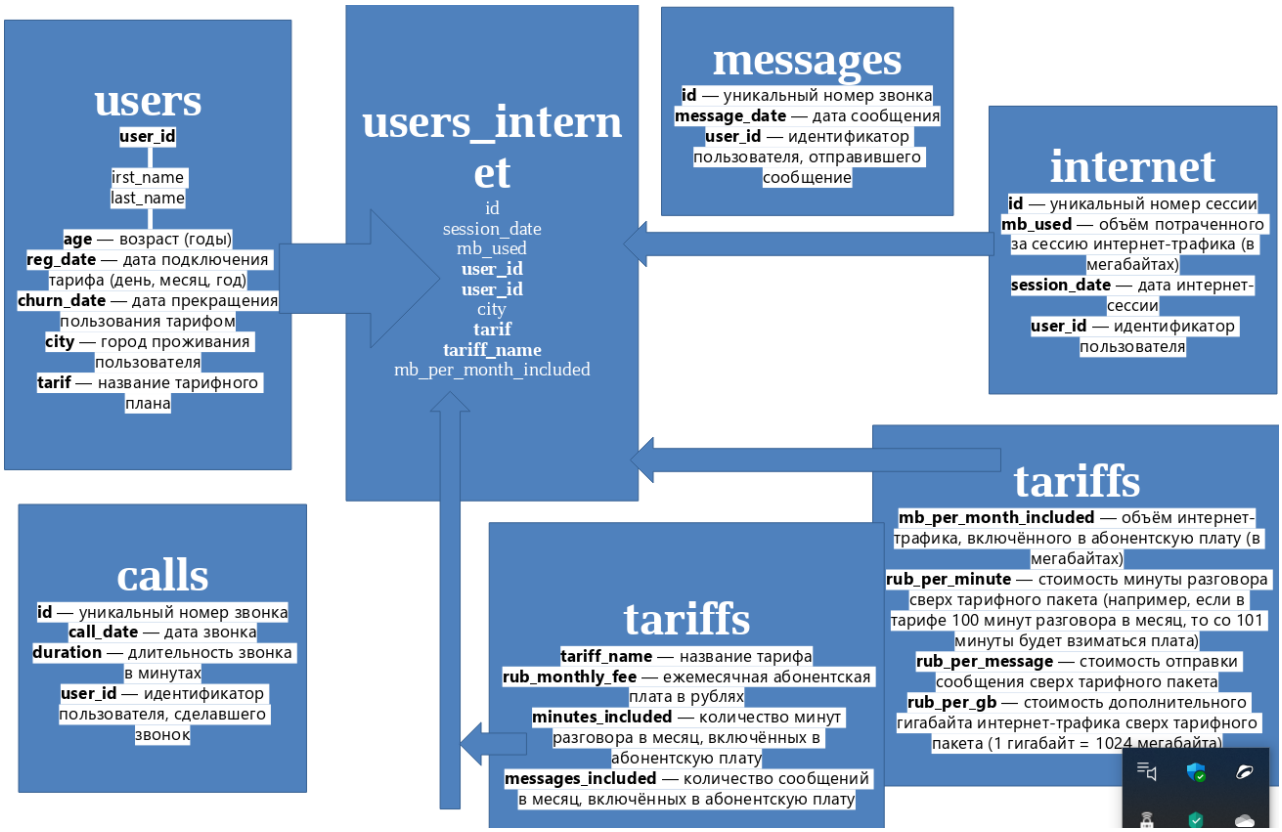

## 4.2 user_messages - количество отправленных сообщений по месяцам

In [37]:
```
1 display(Image(url='http://dl3.joxi.net/drive/2022/02/07/0052/0684/3465900/00/6b
```




## 4.3 users_internet - объем израсходованного интернет-трафика по месяцам

In [38]:
```
1 display(Image(url='http://dl4.joxi.net/drive/2022/02/07/0052/0684/3465900/00/94
```




## 4.4 users_monthly_revenue - помесячная выручка с каждого пользователя

а за какие года данные в таблицах?

In [39]:
```
1 calls['call_date'].dt.year.unique()
```

Out[39]:
```
array([2018], dtype=int64)
```

In [40]:
```
1 messages['message_date'].dt.year.unique()
```

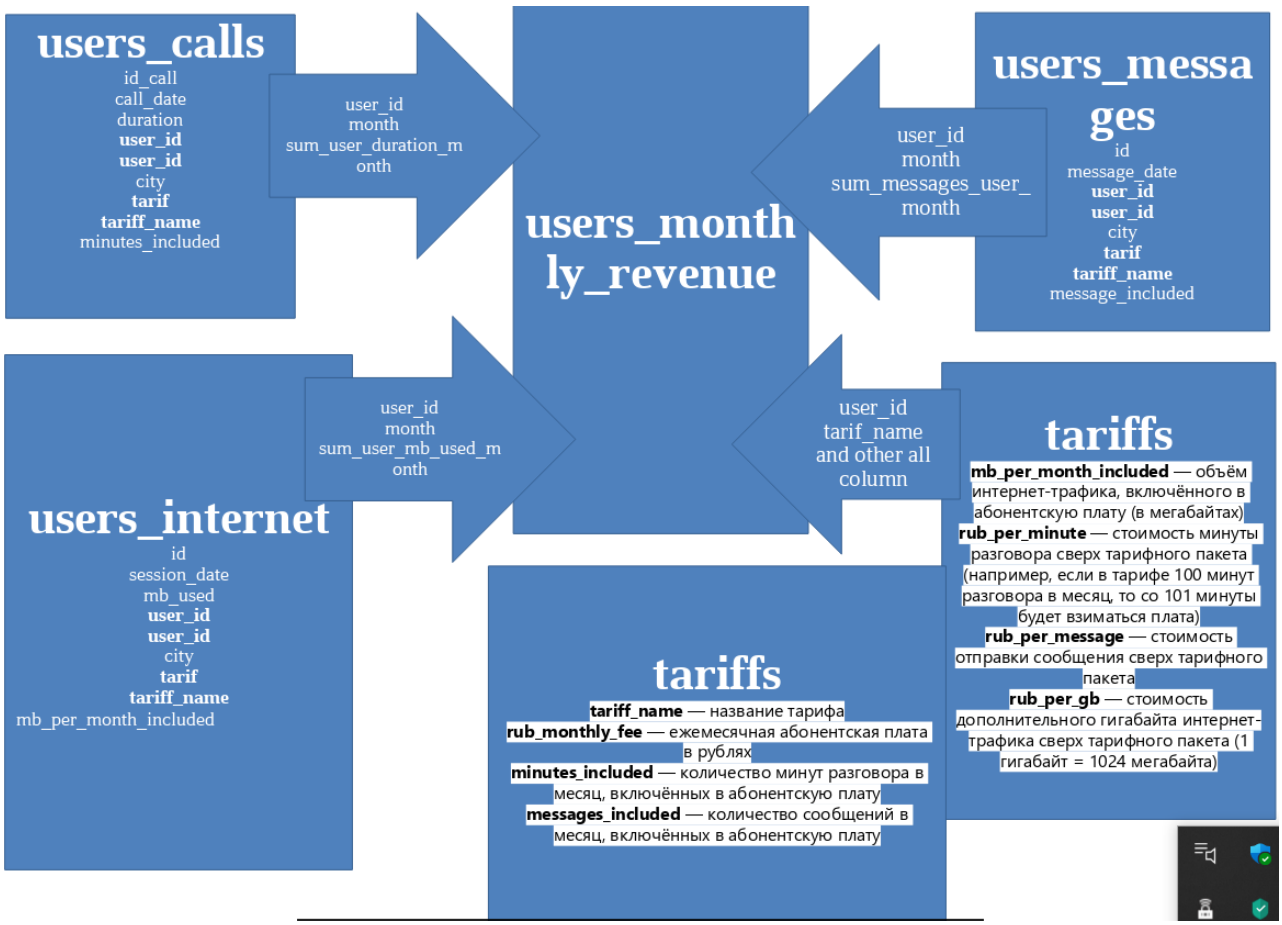
Out[40]:
```
array([2018], dtype=int64)
```

In [41]:
```
1 internet['session_date'].dt.year.unique()
```

Out[41]:
```
array([2018], dtype=int64)
```

**великолепно!** данные лишь за один, **2018**, год!

In [42]:
```
display(Image(url='http://dl3.joxi.net/drive/2022/02/07/0052/0684/3465900/00/1a
```




# 5 создаём сводные таблицы

## 5.1 users_calls

количество сделанных звонков и израсходованных минут разговора по месяцам

1 шаг - calls + users

### 5.1.1 calls

- id
- call_date
- duration
- user_id

### 5.1.2 users

- ключ - user_id
- city
- tarif

In [43]:
```
users_calls = calls.merge(users[['user_id','city', 'tariff']], on='user_id', ho
```

Посмотрим, что получилось

In [44]:
```
users_calls.head()
```

Out[44]:

| | id | call_date | duration | user_id | city | tariff |
|---|---|---|---|---|---|---|
| **0** | 1000_0 | 2018-07-25 | 0.00 | 1000 | Краснодар | ultra |
| **1** | 1000_1 | 2018-08-17 | 0.00 | 1000 | Краснодар | ultra |
| **2** | 1000_2 | 2018-06-11 | 2.85 | 1000 | Краснодар | ultra |
| **3** | 1000_3 | 2018-09-21 | 13.80 | 1000 | Краснодар | ultra |
| **4** | 1000_4 | 2018-12-15 | 5.18 | 1000 | Краснодар | ultra |

In [45]:
```
users_calls.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202607 entries, 0 to 202606
Data columns (total 6 columns):
 #   Column     Non-Null Count   Dtype
---  ------     --------------   -----
 0   id         202607 non-null  object
 1   call_date  202607 non-null  datetime64[ns]
 2   duration   202607 non-null  float64
 3   user_id    202607 non-null  int64
 4   city       202607 non-null  object
 5   tariff     202607 non-null  object
dtypes: datetime64[ns](1), float64(1), int64(1), object(3)
memory usage: 10.8+ MB
```

In [46]:
```
users_calls.describe()
```

Out[46]:

| | duration | user_id |
|---|---|---|
| **count** | 202,607.00 | 202,607.00 |
| **mean** | 6.76 | 1,253.94 |
| **std** | 5.84 | 144.72 |
| **min** | 0.00 | 1,000.00 |
| **25%** | 1.30 | 1,126.00 |
| **50%** | 6.00 | 1,260.00 |
| **75%** | 10.70 | 1,379.00 |
| **max** | 38.00 | 1,499.00 |

### 5.1.3 users_calls успешно создана

### 5.1.4 шаг 2 - количество сделанных звонков и израсходованных минут разговора по месяцам

группируем по user_id (index)
группируем по значениям call_date (columns)
подсчитываем duration (values)
функция - сложением (aggfunc)

**однако**

1. нас интересует группирповка не по дням, а по месяцам

** введём новый столбец **month**

2. длительность звонка менее ОДНОЙ минуты следует считать как полная минута

** введём новый столбец **reduced_duration**

3. **каждый** состоявшийся разговор кругляем по-минутно вверх

In [47]:
```
1 users_calls['month'] = users_calls['call_date'].dt.month
```

In [48]:
```
1 users_calls.month.describe()
```

Out[48]:
```
count   202,607.00
mean          8.57
std           2.79
min           1.00
25%           7.00
50%           9.00
75%          11.00
max          12.00
Name: month, dtype: float64
```

In [49]:
```
1 sorted(users_calls.month.unique())
```

Out[49]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

In [50]:
```
1 users_calls.month.value_counts()
```

Out[50]:
```
12    33987
11    29501
10    27146
9     24061
8     21488
7     18298
6     14716
5     12647
4      9272
3      6687
2      3328
1      1476
Name: month, dtype: int64
```

новый столбец **month** успешно добавлен
в течении года наблюдается неуклонное увеличение количества звонков
данные есть по всем месяцам

**reduced_duration**
Если значение == 0, проставляем 0
Если значение 0 <= 1, проставляем 1
Если значение >1, проставляем исходное значение

оформим в виде функции и применим её к таблице

Округляем **вверх** длительности разговоров

In [53]:
```
1 users_calls['reduced_duration'] = np.ceil(users_calls['duration']) # round_up
```

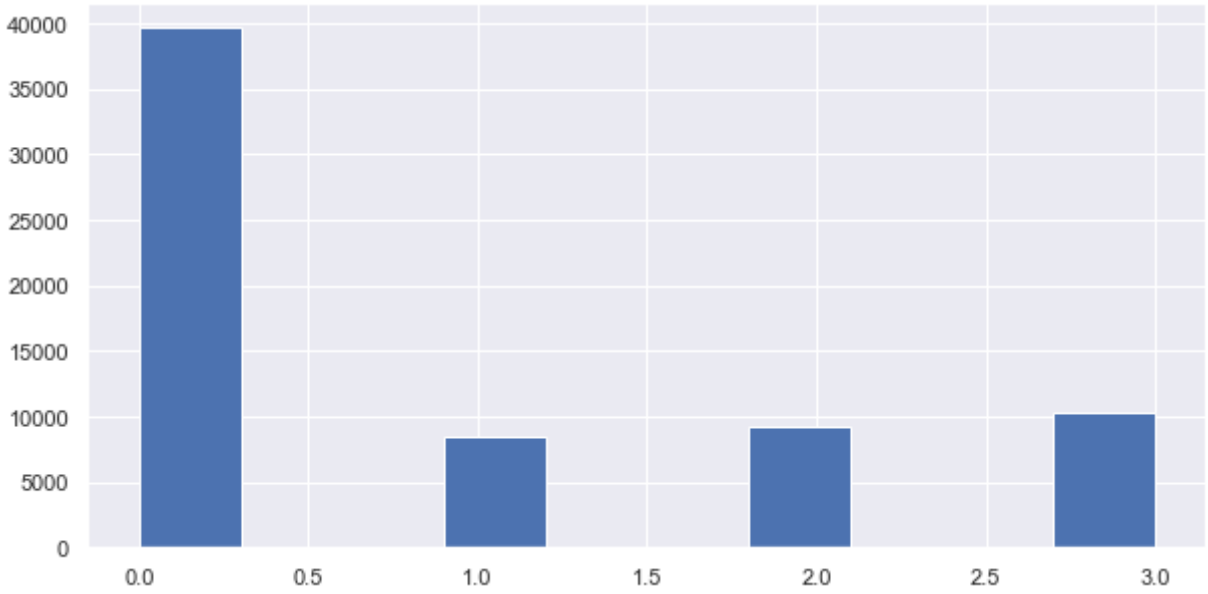
Проверяем результат

In [54]:
```
1 users_calls.reduced_duration.describe()
```

Out[54]:
```
count   202,607.00
mean          7.16
std           5.96
min           0.00
25%           2.00
50%           6.00
75%          11.00
max          38.00
Name: reduced_duration, dtype: float64
```

Удачное округление вверх

In [55]:
```
1 # setup size plot
2 sns.set(rc = {'figure.figsize':(10,5)})
3
4 users_calls.reduced_duration.hist(range=(0,3));
```




#### 5.1.5 Отлично!

Пропущенных звонков много - **спам** замучил всех!

Звонки менее минуты превратились в одноминутные!

## 5.2 данные готовы для создания сводной таблицы

группируем по user_id (index)
группируем по значениям month (columns)
подсчитываем reduced_duration (values)
функция - сложением - sum (aggfunc)

In [56]:
```
1 users_calls_month = users_calls.groupby(['user_id', 'month'])['reduced_duration
```

Посмотрим результат

In [57]:
```
1 users_calls_month.head(10)
```

Out[57]:
```
user_id  month
1000     5        159.00
         6        172.00
         7        340.00
         8        408.00
         9        466.00
         10       350.00
         11       338.00
         12       333.00
1001     11       430.00
         12       414.00
Name: reduced_duration, dtype: float64
```

In [58]:
```
1 users_calls_month.loc[1000,11]
2
```

Out[58]:
```
338.0
```

#### 5.2.1 это значение для пользователя 1000 за 11 месяц

Мы убедились в том, что получили то ,что хотели.

**великолепно!**

## 5.3 users_monthly_revenue - 1 шаг

помесячная выручка с каждого пользователя

#### 5.3.1 Первые данные - по сумме **времени звонков** *

1. user_id
2. month
3. users_calls_month.loc[user_id, month]

В **users_monthly_revenue** первая колонка - **user_id**
её берём из **users**

Вторая - **month** - **список** значений
sorted(users_calls.month.unique()):

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Третья - значение ячейки по user_id и месяцу
из сложного списка **users_calls_month**

users_calls_month.loc[user_id, month]

In [59]:

```
# definition DataFrame users_monthly_revenue
users_monthly_revenue = pd.DataFrame(columns =['user_id', 'month', 'sum_duratio
users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 0 entries
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype 
---  ------        --------------  ----- 
 0   user_id       0 non-null      object
 1   month         0 non-null      object
 2   sum_duration  0 non-null      object
dtypes: object(3)
memory usage: 0.0+ bytes
```

Пустая табличка создана

### 5.3.2 Цикл

для каждого значения user_id для каждого месяца вносим строку user_id///month///sum_duration

In [60]:

```
# list of users
users_list = []
users_list = (sorted(users['user_id'].unique()))

# list of months
months = sorted(users_calls.month.unique())

for user in users_list:
    for m in months:

        try:
            sum_duration = users_calls_month.loc[user, m]

        except:
            sum_duration = 0


        users_monthly_revenue.loc[len(users_monthly_revenue.index)] = [user, m,

users_monthly_revenue.head()
```

Out[60]:

| | user_id | month | sum_duration |
|---|---|---|---|
| **0** | 1,000.00 | 1.00 | 0.00 |
| **1** | 1,000.00 | 2.00 | 0.00 |
| **2** | 1,000.00 | 3.00 | 0.00 |
| **3** | 1,000.00 | 4.00 | 0.00 |
| **4** | 1,000.00 | 5.00 | 159.00 |

In [61]:
```
1 users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 3 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
dtypes: float64(3)
memory usage: 187.5 KB
```

In [62]:
```
1 users_monthly_revenue.describe()
```

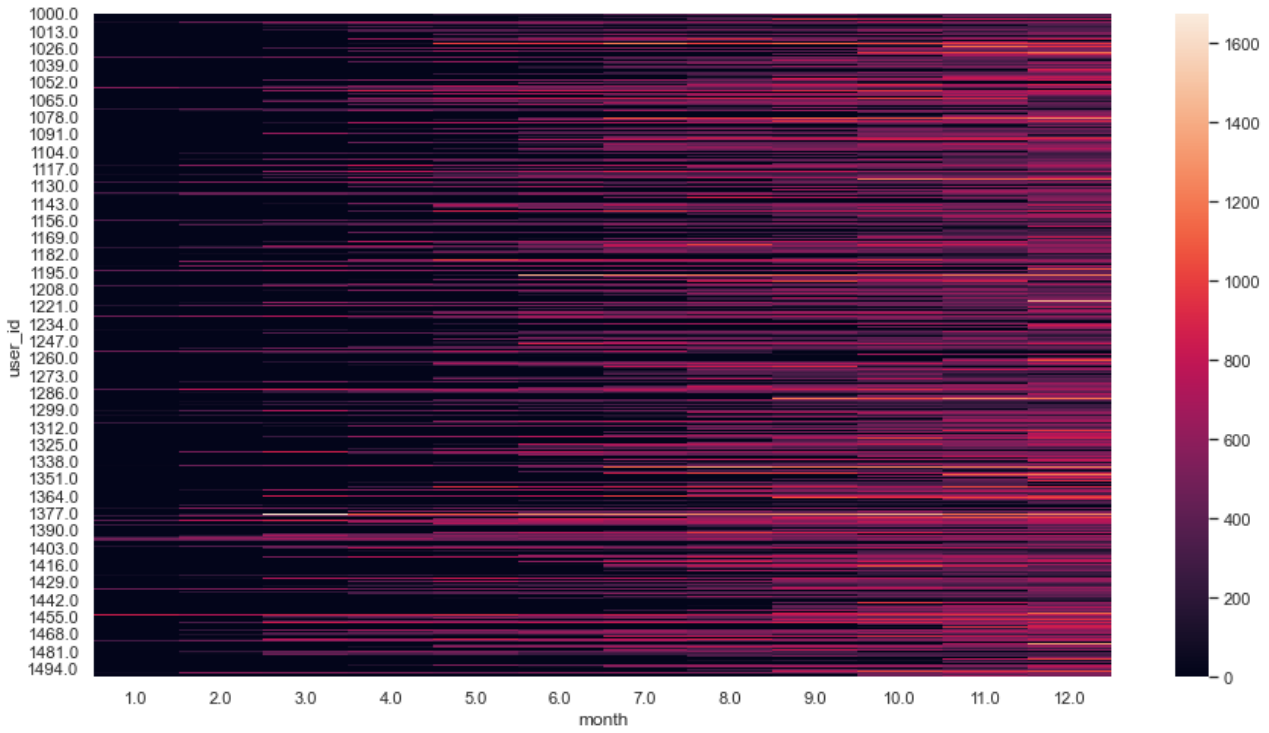
Out[62]:

| | user_id | month | sum_duration |
|---|---|---|---|
| **count** | 6,000.00 | 6,000.00 | 6,000.00 |
| **mean** | 1,249.50 | 6.50 | 241.72 |
| **std** | 144.35 | 3.45 | 286.35 |
| **min** | 1,000.00 | 1.00 | 0.00 |
| **25%** | 1,124.75 | 3.75 | 0.00 |
| **50%** | 1,249.50 | 6.50 | 98.00 |
| **75%** | 1,374.25 | 9.25 | 460.00 |
| **max** | 1,499.00 | 12.00 | 1,673.00 |

### 5.3.3 первый шаг выполнен

In [63]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(15,8)})

# prepare dataframe for heatmap
sns_users_monthly_revenue = users_monthly_revenue
# sns_users_monthly_revenue.fillna(0, inplace = True)
sns_users_monthly_revenue = (
    users_monthly_revenue.pivot_table(index=['user_id'],
                      columns='month', values='sum_duration', aggfunc='sum')
)

# plot heatmap
sns.heatmap(sns_users_monthly_revenue);
```




#### 5.3.4 Этот график - "тепловая карта" - показывает, как постепенно в течении года

- увеличивалось число абонентов
- большинство абонентов постепенно совершало больше и больше звонков

## 5.4 users_calls_month готова!

для каждого пользователя есть сумма длительности звонков для расчета стоимости, подсчитанная по-месячно

## 5.5 user_messages

- количество отправленных сообщений по месяцам

#### 5.5.1 1 шаг - messages + users

#### 5.5.2 messages

- id
- message_date

- user_id

### 5.5.3 users

- ключ - user_id
- city
- tarif

In [64]:
```
1 user_messages = messages.merge(users[['user_id','city', 'tariff']], on='user_id
```

Посмотрим, что получилось

In [65]:
```
1 user_messages.head()
```

Out[65]:

| | id | message_date | user_id | city | tariff |
|---|---|---|---|---|---|
| **0** | 1000_0 | 2018-06-27 | 1000 | Краснодар | ultra |
| **1** | 1000_1 | 2018-10-08 | 1000 | Краснодар | ultra |
| **2** | 1000_2 | 2018-08-04 | 1000 | Краснодар | ultra |
| **3** | 1000_3 | 2018-06-16 | 1000 | Краснодар | ultra |
| **4** | 1000_4 | 2018-12-05 | 1000 | Краснодар | ultra |

In [66]:
```
1 user_messages.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 123036 entries, 0 to 123035
Data columns (total 5 columns):
 #   Column        Non-Null Count   Dtype         
---  ------        --------------   -----         
 0   id            123036 non-null  object        
 1   message_date  123036 non-null  datetime64[ns]
 2   user_id       123036 non-null  int64         
 3   city          123036 non-null  object        
 4   tariff        123036 non-null  object        
dtypes: datetime64[ns](1), int64(1), object(3)
memory usage: 5.6+ MB
```

In [67]:
```
1 user_messages.describe()
```

Out[67]:

| | user_id |
|---|---|
| **count** | 123,036.00 |
| **mean** | 1,256.99 |
| **std** | 143.52 |
| **min** | 1,000.00 |
| **25%** | 1,134.00 |
| **50%** | 1,271.00 |
| **75%** | 1,381.00 |
| **max** | 1,499.00 |

### 5.5.4 user_messages успешно создана

### 5.5.5 шаг 2 - количество сообщений по месяцам

группируем по user_id (index)
группируем по значениям month (columns)
подсчитываем id (values)
функция - count (aggfunc)

** **month** предварительно создадим и заполним

In [68]:
```
# create and fill column 'month'
user_messages['month'] = user_messages['message_date'].dt.month
```

In [69]:
```
user_messages.month.describe()
```

Out[69]:
```
count   123,036.00
mean          8.63
std           2.75
min           1.00
25%           7.00
50%           9.00
75%          11.00
max          12.00
Name: month, dtype: float64
```

In [70]:
```
sorted(user_messages.month.unique())
```

Out[70]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

In [71]:
```
user_messages.month.value_counts()
```

Out[71]:
```
12    20555
11    18244
10    17114
9     14759
8     13088
7     11320
6      8983
5      7298
4      5179
3      3648
2      1924
1       924
Name: month, dtype: int64
```

новый столбец **month** успешно добавлен
в течении года наблюдается неуклонное увеличение количества сообщений
данные есть по всем месяцам

### 5.5.6 Создаём таблицы с количеством сообщений по месяцам

users_messages_month

In [72]:
```
users_messages_month = user_messages.groupby(['user_id', 'month'])['id'].count(
```

Посмотрим результат

In [73]:
```
1 users_messages_month.head(10)
```

Out[73]:
```
user_id  month
1000     5        22
         6        60
         7        75
         8        81
         9        57
         10       73
         11       58
         12       70
1002     6         4
         7        11
Name: id, dtype: int64
```

In [74]:
```
1 users_messages_month.loc[1000,11]
2
```

Out[74]:
```
58
```

#### 5.5.7 это значение для пользователя 1000 за 11 месяц

Мы убедились в том, что получили то ,что хотели.

## 5.6 users_monthly_revenue - 2 шаг

помесячная выручка с каждого пользователя

#### 5.6.1 Следующие данные - по количеству *сообщений *

1. user_id
2. month
3. users_messages_month.loc[user_id, month]

В **users_monthly_revenue** первая колонка - **user_id**

Вторая - **month** - **список** значений
sorted(users_calls.month.unique()):

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Третья - значение ячейки по user_id и месяцу
из сложного списка **users_messages_month**

users_messages_month.loc[user_id, month]

Цикл для каждого значения user_id для каждого месяца вносим строку
user_id///month///sum_messages

Добавим значения **users_messages_month** в **users_monthly_revenue**
методом **merge**

In [75]:
```
users_monthly_revenue = (
    users_monthly_revenue.merge(users_messages_month,
                on=['user_id', 'month'] , how='left')
)
```

In [76]:
```
users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   id            2717 non-null   float64
dtypes: float64(4)
memory usage: 234.4 KB
```

Переименуем колонку **id** в **sum_messages**

In [77]:
```
users_monthly_revenue = users_monthly_revenue.rename(columns={'id': 'sum_messag

```

Заполним пропуски значениями "0"

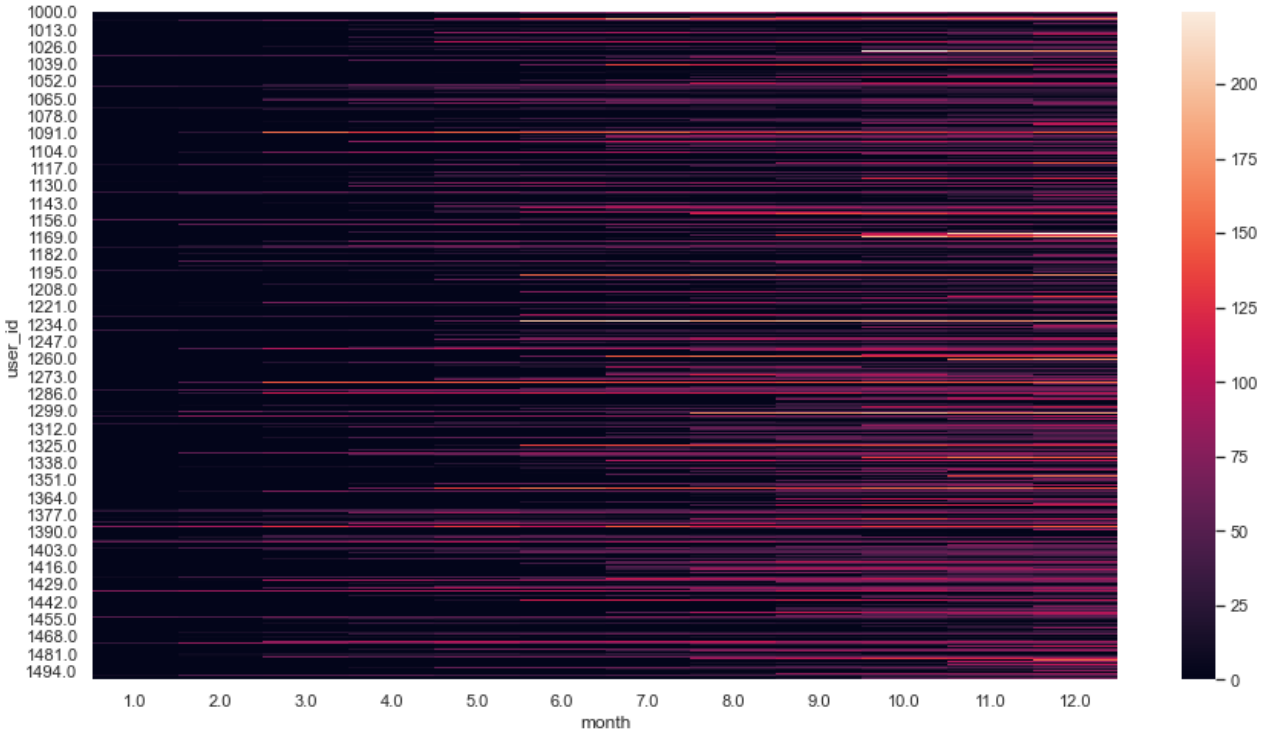
In [78]:
```
users_monthly_revenue.fillna(0, inplace = True)
```

In [79]:
```
users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 4 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   sum_messages  6000 non-null   float64
dtypes: float64(4)
memory usage: 234.4 KB
```

In [80]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(15,8)})

# prepare dataframe for heatmap
sns_users_monthly_revenue = users_monthly_revenue
sns_users_monthly_revenue = (
    users_monthly_revenue.pivot_table(index=['user_id'],
                columns='month', values='sum_messages', aggfunc='sum')
)

# plot heatmap
sns.heatmap(sns_users_monthly_revenue);
```




Этот график - "тепловая карта" - показывает, как постепенно в течении года

- увеличивалось число абонентов
- большинство абонентов постепенно отправляло больше и больше сообщений
- есть абоненты ,которые не пользуются отправкой сообщений

## 5.7 users_internet

- количество использованного траффика

### 5.7.1 1 шаг - internet + users

### 5.7.2 internet

- id
- session_date
- mb_used
- user_id

### 5.7.3 users

- ключ - user_id
- city
- tarif

In [81]:
```
1 users_internet = internet.merge(users[['user_id','city', 'tariff']], on='user_i
```

Посмотрим, что получилось

In [82]:
```
1 users_internet.head()
```

Out[82]:

| | id | mb_used | session_date | user_id | city | tariff |
|---|---|---|---|---|---|---|
| **0** | 1000_0 | 112.95 | 2018-11-25 | 1000 | Краснодар | ultra |
| **1** | 1000_1 | 1,052.81 | 2018-09-07 | 1000 | Краснодар | ultra |
| **2** | 1000_2 | 1,197.26 | 2018-06-25 | 1000 | Краснодар | ultra |
| **3** | 1000_3 | 550.27 | 2018-08-22 | 1000 | Краснодар | ultra |
| **4** | 1000_4 | 302.56 | 2018-09-24 | 1000 | Краснодар | ultra |

In [83]:
```
1 users_internet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 149396 entries, 0 to 149395
Data columns (total 6 columns):
 #   Column        Non-Null Count   Dtype         
---  ------        --------------   -----         
 0   id            149396 non-null  object        
 1   mb_used       149396 non-null  float64       
 2   session_date  149396 non-null  datetime64[ns]
 3   user_id       149396 non-null  int64         
 4   city          149396 non-null  object        
 5   tariff        149396 non-null  object        
dtypes: datetime64[ns](1), float64(1), int64(1), object(3)
memory usage: 8.0+ MB
```

In [84]:
```
1 users_internet.describe()
```

Out[84]:

| | mb_used | user_id |
|---|---|---|
| **count** | 149,396.00 | 149,396.00 |
| **mean** | 370.19 | 1,252.10 |
| **std** | 278.30 | 144.05 |
| **min** | 0.00 | 1,000.00 |
| **25%** | 138.19 | 1,130.00 |
| **50%** | 348.01 | 1,251.00 |
| **75%** | 559.55 | 1,380.00 |
| **max** | 1,724.83 | 1,499.00 |

### 5.7.4 users_internet успешно создана

### 5.7.5 шаг 2 - количество траффика по месяцам

группируем по user_id (index)
группируем по значениям month (columns)
подсчитываем mb_used (values)
функция - sum (aggfunc)

** **month** предварительно создадим и заполним

In [85]:
```
1 # create and fill column 'month'
2 users_internet['month'] = users_internet['session_date'].dt.month
```

In [86]:
```
1 users_internet.month.describe()
```

Out[86]:
```
count   149,396.00
mean          8.56
std           2.79
min           1.00
25%           7.00
50%           9.00
75%          11.00
max          12.00
Name: month, dtype: float64
```

In [87]:
```
1 sorted(users_internet.month.unique())
```

Out[87]:
```
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

In [88]:
```
1 users_internet.month.value_counts()
```

Out[88]:
```
12    24799
11    21817
10    20009
9     17512
8     16092
7     13548
6     11057
5      9408
4      6511
3      4885
2      2641
1      1117
Name: month, dtype: int64
```

новый столбец **month** успешно добавлен
в течении года наблюдается неуклонное увеличение количества интернет- сессий
данные есть по всем месяцам

### 5.7.6 Создаём таблицы с количеством траффика по месяцам

users_internet_month

In [89]:
```
1 users_internet_month = users_internet.groupby(['user_id', 'month'])['mb_used'].
```

Посмотрим результат

In [90]:
```
1 users_internet_month.head(10)
```

Out[90]:
```
user_id  month
1000     5        2,253.49
         6       23,233.77
         7       14,003.64
         8       14,055.93
         9       14,568.91
         10      14,702.49
         11      14,756.47
         12       9,817.61
1001     11      18,429.34
         12      14,036.66
Name: mb_used, dtype: float64
```

In [91]:
```
1 users_internet_month.loc[1000,11]
2
```

Out[91]:
```
14756.47
```

### 5.7.7 это значение для пользователя 1000 за 11 месяц

Мы убедились в том, что получили то ,что хотели.

## 5.8 users_monthly_revenue - 3 шаг

помесячная выручка с каждого пользователя

### 5.8.1 Следующие данные - по количеству *_траффика_ *

1. user_id
2. month
3. users_internet_month.loc[user_id, month]

В **users_monthly_revenue** первая колонка - **user_id**

Вторая - **month** - **список** значений
sorted(users_calls.month.unique()):

[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

Третья - значение ячейки по user_id и месяцу
из сложного списка **users_internet_month**

users_internet_month.loc[user_id, month]

Для каждого значения user_id для каждого месяца вносим строку
user_id///month///sum_mb_used

Добавим значения **users_internet_month** в **users_monthly_revenue**
методом **merge**

In [92]:
```
users_monthly_revenue = (
    users_monthly_revenue.merge(users_internet_month,
                on=['user_id', 'month'] , how='left')
)
```

In [93]:
```
users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   sum_messages  6000 non-null   float64
 4   mb_used       3203 non-null   float64
dtypes: float64(5)
memory usage: 281.2 KB
```

Переименуем колонку **mb_used** в **sum_mb_used**

In [94]:
```
users_monthly_revenue = users_monthly_revenue.rename(columns={'mb_used': 'sum_m

```

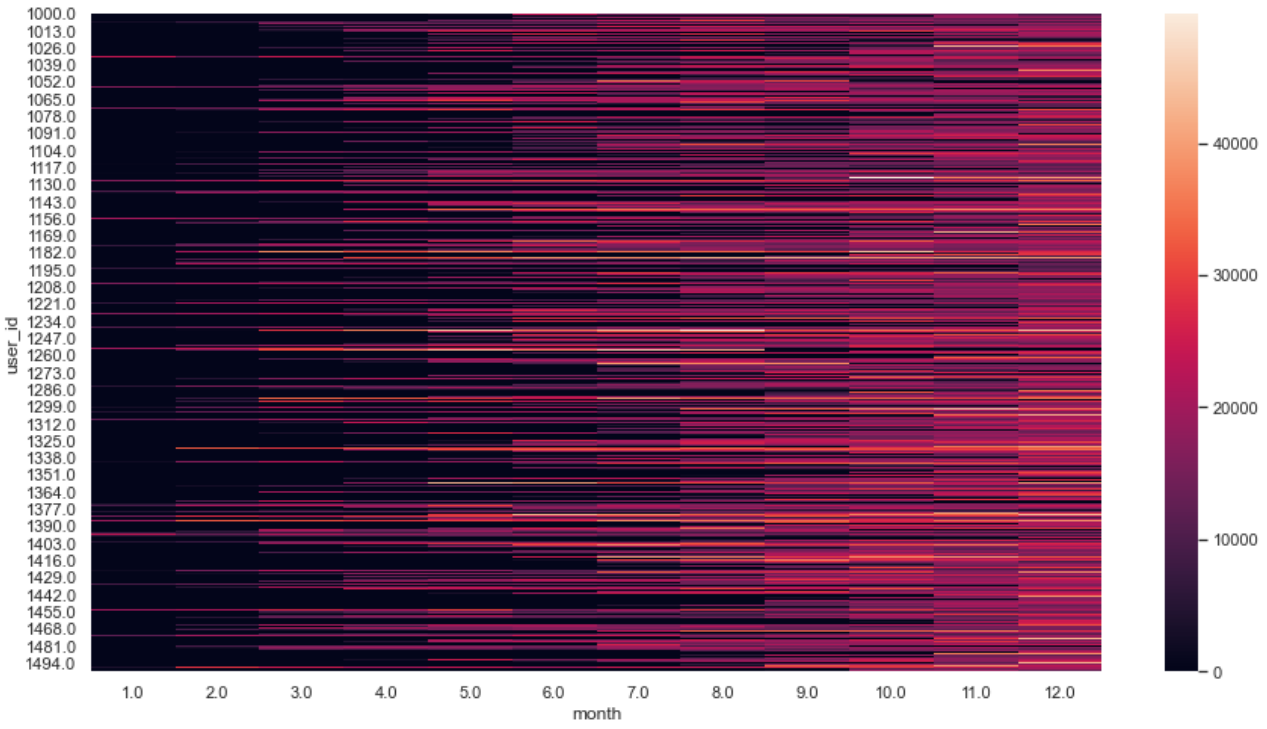
Заполним пропуски значениями "0"

In [95]:
```
users_monthly_revenue.fillna(0, inplace = True)
```

In [96]:
```
1 users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   sum_messages  6000 non-null   float64
 4   sum_mb_used   6000 non-null   float64
dtypes: float64(5)
memory usage: 281.2 KB
```

In [97]:
```
1  # setup size plot
2  sns.set(rc = {'figure.figsize':(15,8)})
3
4  # prepare dataframe for heatmap
5  sns_users_monthly_revenue = users_monthly_revenue
6  sns_users_monthly_revenue = (
7      users_monthly_revenue.pivot_table(index=['user_id'],
8                   columns='month', values='sum_mb_used', aggfunc='sum')
9  )
10
11 # plot heatmap
12 sns.heatmap(sns_users_monthly_revenue);
```




### 5.8.2 Этот график - "тепловая карта" - показывает, как постепенно в течении года

- увеличивалось число абонентов
- большинство абонентов постепенно больше и больше сообщений пользовалось мобильным интернетом
- есть абоненты, которые пользуются мобильным интернетом немного
- - может быть, сделать им предложение **по тарифу Wi-Fi**?

## 5.9 добавим значения

- **tariff**
- **city**

In [98]:
```
users_monthly_revenue = (
    users_monthly_revenue.merge(users[['user_id', 'city', 'tariff']] ,
                on=['user_id'] , how='left')
)
```

Проверка

In [99]:
```
users_monthly_revenue.head()
```

Out[99]:

| | user_id | month | sum_duration | sum_messages | sum_mb_used | city | tariff |
|---|---|---|---|---|---|---|---|
| **0** | 1,000.00 | 1.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **1** | 1,000.00 | 2.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **2** | 1,000.00 | 3.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **3** | 1,000.00 | 4.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **4** | 1,000.00 | 5.00 | 159.00 | 22.00 | 2,253.49 | Краснодар | ultra |

In [100]:
```
users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   sum_messages  6000 non-null   float64
 4   sum_mb_used   6000 non-null   float64
 5   city          6000 non-null   object 
 6   tariff        6000 non-null   object 
dtypes: float64(5), object(2)
memory usage: 375.0+ KB
```

In [101]:
```
1 users_monthly_revenue.reset_index(drop=True)
```

Out[101]:

| | user_id | month | sum_duration | sum_messages | sum_mb_used | city | tariff |
|---|---|---|---|---|---|---|---|
| **0** | 1,000.00 | 1.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **1** | 1,000.00 | 2.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **2** | 1,000.00 | 3.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **3** | 1,000.00 | 4.00 | 0.00 | 0.00 | 0.00 | Краснодар | ultra |
| **4** | 1,000.00 | 5.00 | 159.00 | 22.00 | 2,253.49 | Краснодар | ultra |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **5995** | 1,499.00 | 8.00 | 0.00 | 0.00 | 0.00 | Пермь | smart |
| **5996** | 1,499.00 | 9.00 | 70.00 | 11.00 | 1,845.75 | Пермь | smart |
| **5997** | 1,499.00 | 10.00 | 449.00 | 48.00 | 17,788.51 | Пермь | smart |
| **5998** | 1,499.00 | 11.00 | 612.00 | 59.00 | 17,963.31 | Пермь | smart |
| **5999** | 1,499.00 | 12.00 | 492.00 | 66.00 | 13,055.58 | Пермь | smart |

6000 rows × 7 columns

**добавилось удачно**

In [102]:
```
1 users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   sum_messages  6000 non-null   float64
 4   sum_mb_used   6000 non-null   float64
 5   city          6000 non-null   object 
 6   tariff        6000 non-null   object 
dtypes: float64(5), object(2)
memory usage: 375.0+ KB
```

## 5.10 users_monthly_revenue - 4 шаг

помесячную выручку с каждого пользователя (вычтите бесплатный лимит из суммарного количества звонков, сообщений и интернет-трафика;
остаток умножьте на значение из тарифного плана; прибавьте абонентскую плату, соответствующую тарифному плану).
**revenue**

1. проверим, являлся ли этот **user_id** абонентом в этом месяце

- если значения во всех трёх колонках == "0", то **revenue=0**
- иначе
  - revenue = **rub_monthly_fee** из tariffs
  - вычисляем сумму доплат **add_pay**
  - каждую доплату **прибавляем** к revenue

2. сравнить длительность звонков **sum_duration** с количеством, включённый в тариф

```
**minutes_included**
значение minutes_included зависит от тарифа
```

- тариф считываем из текущей строки
- - запишем в переменную **tariff**
- если sum_duration **больше** minutes_included
- - к revenue добавляем дополнительную стоимость
- для этого
- - из sum_duration вычитаем minutes_included и умножаем на
- - - стоимость дополнительной минуты **rub_per_minute** в tariffs_dict
    полученное значение **add_pay** прибавляем к **revenue**

3. сравнить количество сообщений **sum_messages** с количеством, включённый в тариф

```
**messages_included**
значение minutes_included зависит от тарифа
тариф для этого пользователя у нас хранится во временной переменной **tari
ff**, определённой на шаге 2.
```

- если sum_messages **больше** messages_included
- - к revenue добавляем дополнительную стоимость
- для этого
- - из sum_messages вычитаем messages_included и умножаем на
- - - стоимость дополнительных сообщений **rub_per_message** в tariffs_dict
    полученное значение **add_pay** прибавляем к **revenue**

4. сравнить количество траффика **sum_mb_used** с количеством, включённый в тариф

```
**mb_per_month_included**
значение mb_per_month_included зависит от тарифа
тариф для этого пользователя у нас хранится во временной переменной **tari
ff**, определённой на шаге 2.
```

- если sum_mb_used **больше** mb_per_month_included
- - к revenue добавляем дополнительную стоимость
- для этого

- - из sum_mb_used вычитаем mb_per_month_included (**add_traffic**) и умножаем на
- - - стоимость дополнительного трафиика **rub_per_gb** в tariffs_dict
- *предварительно**
- - переводим add_traffic в **gb_add_traffic**
- - - для этого add_traffic разделить на 1024

полученное значение **add_pay** прибавляем к **revenue**

### 5.10.1 Определяем функцию revenue_calculation

In [103]:

```python
def revenue_calculation(df, dict):
    # defining temporary variables
    revenue = 0
    add_pay = 0
    add_traffic = 0
    gb_add_traffic = 0
    gb_add_traffic = 0

    # get values from df
    user_id = df['user_id']
    tarif = df['tariff']
    sum_duration = df['sum_duration']
    sum_messages = df['sum_messages']
    sum_mb_used = df['sum_mb_used']

    #get values from dict
    rub_monthly_fee = dict.get('rub_monthly_fee').get(tarif)
    minutes_included = dict.get('minutes_included').get(tarif)
    rub_per_minute = dict.get('rub_per_minute').get(tarif)
    messages_included = dict.get('messages_included').get(tarif)
    rub_per_message = dict.get('rub_per_message').get(tarif)
    mb_per_month_included = dict.get('mb_per_month_included').get(tarif)
    rub_per_gb = dict.get('rub_per_gb').get(tarif)

    # set revenue as basic subscription fee in the tariff
    revenue = rub_monthly_fee

    if sum_duration == 0 and sum_messages == 0 and sum_mb_used == 0:
        revenue = 0
        return(revenue) # exit from def with "0"

    if sum_duration > minutes_included:
        # adding revenue for exceeding the call time

        add_pay = (sum_duration - minutes_included) * rub_per_minute

        revenue += add_pay

    if sum_messages > messages_included:
        # adding revenue for exceeding the number of messages
        add_pay = (sum_messages - messages_included) * rub_per_message
        revenue += add_pay

    if sum_mb_used > mb_per_month_included:
        # adding revenue for exceeding traffic
        add_traffic = sum_mb_used - mb_per_month_included
        gb_add_traffic = add_traffic / 1024
        gb_add_traffic = math.ceil(gb_add_traffic) # round up

        add_pay = gb_add_traffic *  rub_per_gb
        revenue += add_pay

    return(revenue)
```

In [104]:

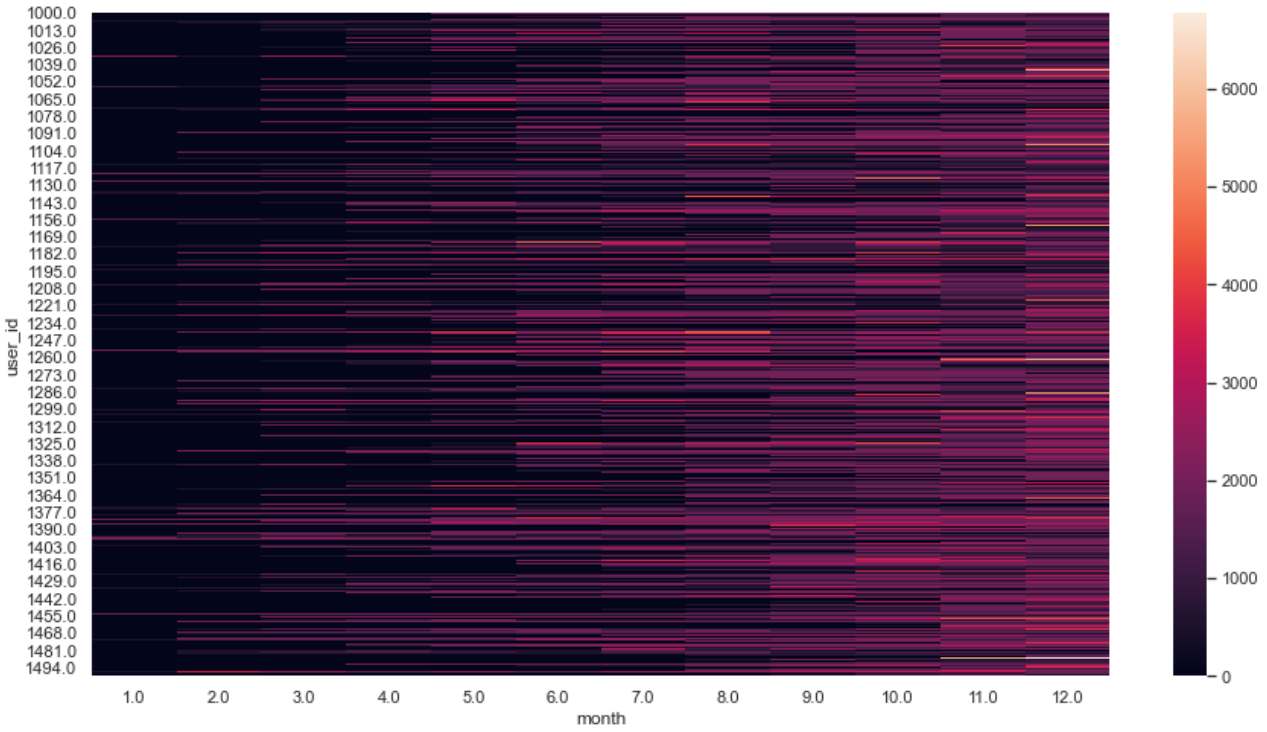
```python
users_monthly_revenue['revenue'] = users_monthly_revenue.apply(revenue_calculat
```

In [105]:
```
1 users_monthly_revenue.revenue.describe()
```

Out[105]:
```
count   6,000.00
mean      819.08
std       960.69
min         0.00
25%         0.00
50%       550.00
75%     1,806.25
max     6,770.00
Name: revenue, dtype: float64
```

## 5.11 Посмотрим на тепловой карте

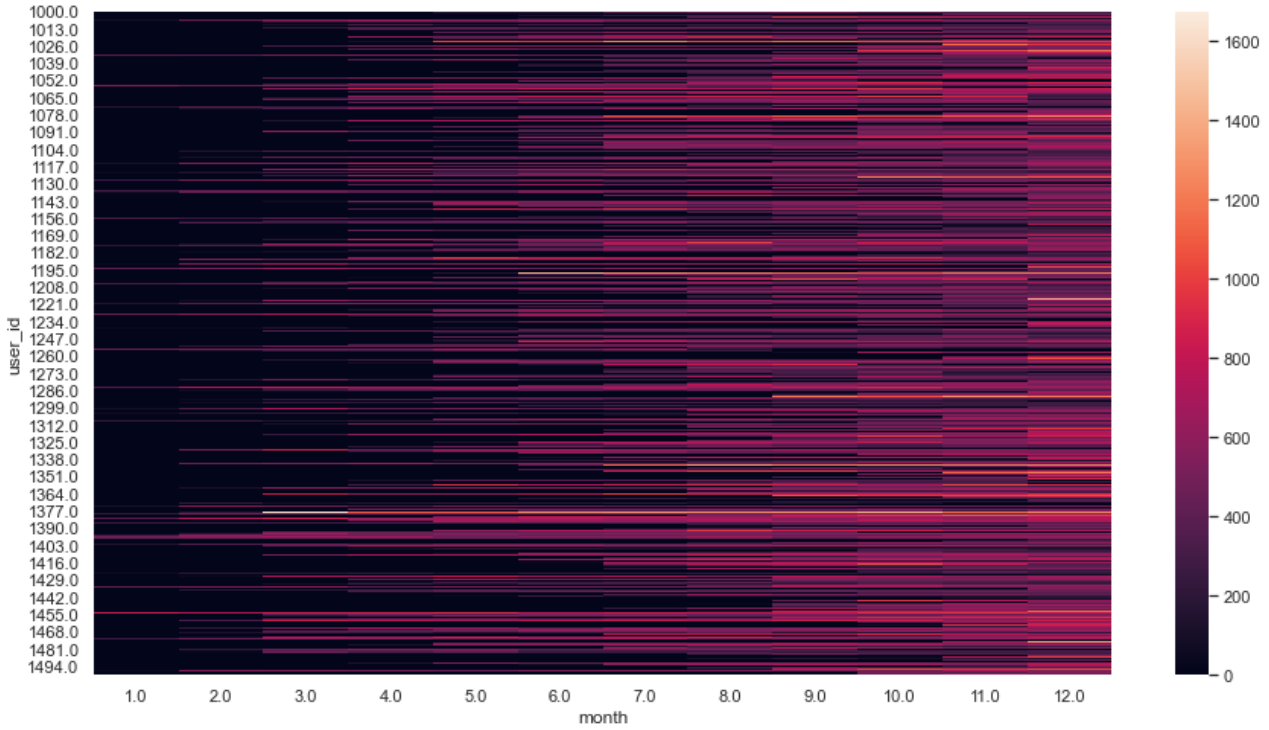
In [106]:
```
1  # setup size plot
2  sns.set(rc = {'figure.figsize':(15,8)})
3
4  # prepare dataframe for heatmap
5  sns_users_monthly_revenue = users_monthly_revenue
6  # sns_users_monthly_revenue.fillna(0, inplace = True)
7  sns_users_monthly_revenue = (
8      users_monthly_revenue.pivot_table(index=['user_id'],
9                        columns='month', values='revenue', aggfunc='sum')
10 )
11
12 # plot heatmap
13 sns.heatmap(sns_users_monthly_revenue);
```




Вспомним тепловую карту звонков

In [107]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(15,8)})

# prepare dataframe for heatmap
sns_users_monthly_revenue = users_monthly_revenue
# sns_users_monthly_revenue.fillna(0, inplace = True)
sns_users_monthly_revenue = (
    users_monthly_revenue.pivot_table(index=['user_id'],
                      columns='month', values='sum_duration', aggfunc='sum')
)

# plot heatmap
sns.heatmap(sns_users_monthly_revenue);
```




#### 5.11.1 общая картина совпадает, значит расчёты проведены верно!

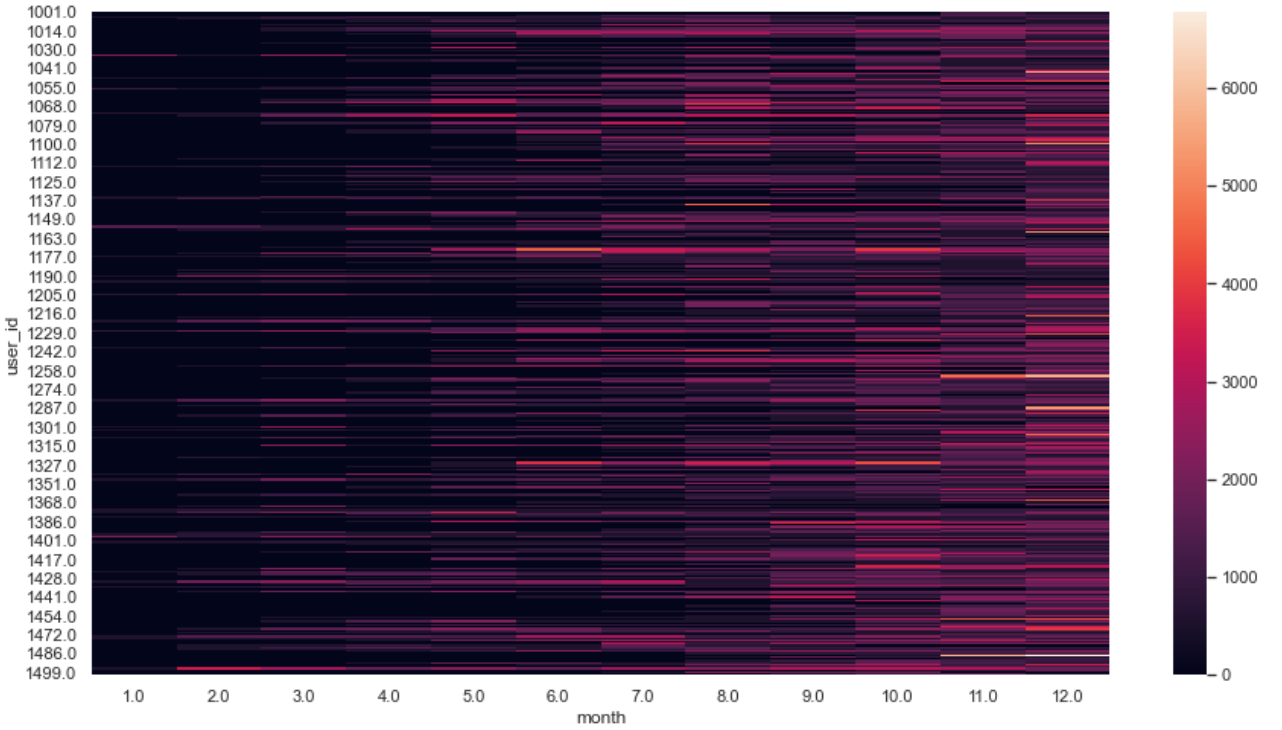
# 6 Анализируем данные

## 6.1 Посмотрим тепловые карты для всех пользователей, но отдельно по тарифам

**smart**

In [108]:

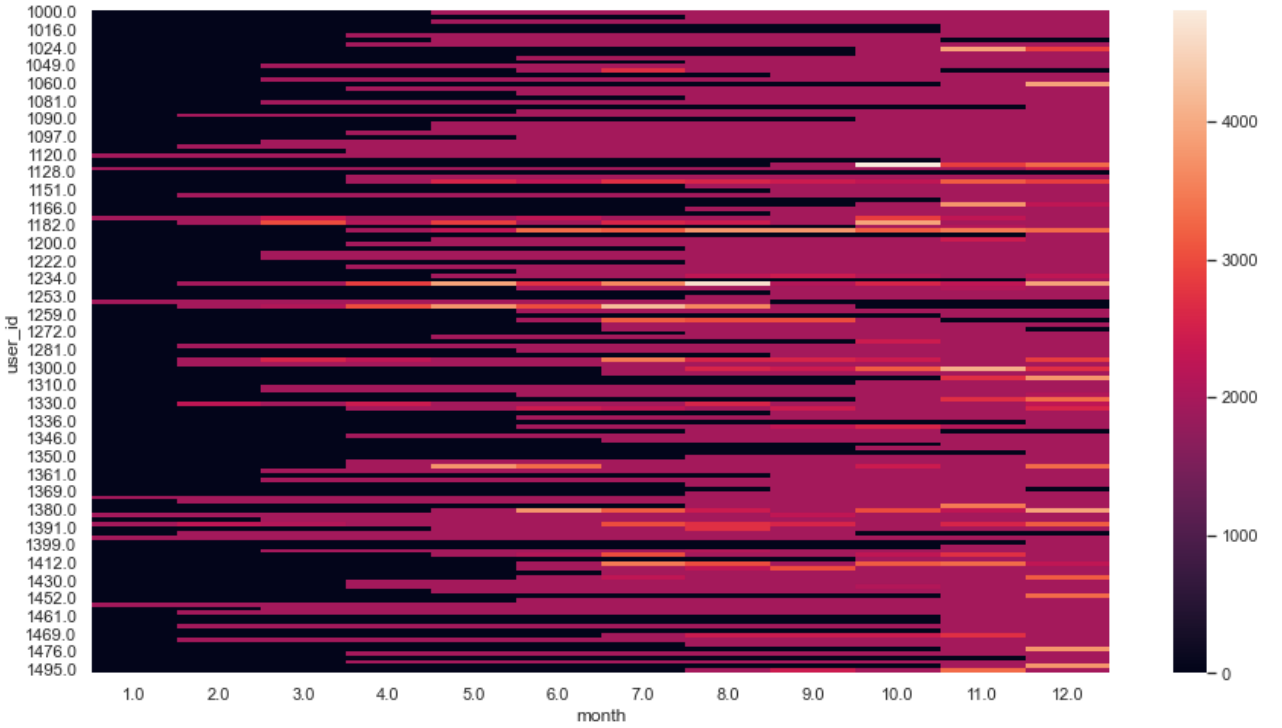
```
# setup size plot
sns.set(rc = {'figure.figsize':(15,8)})

# prepare dataframe for heatmap
sns_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart

# sns_users_monthly_revenue.fillna(0, inplace = True)
sns_smart_users_monthly_revenue = (
    sns_smart_users_monthly_revenue.pivot_table(index=['user_id'],
                    columns='month', values='revenue', aggfunc='sum')
)

# plot heatmap
sns.heatmap(sns_smart_users_monthly_revenue);
```




**ultra**

In [109]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(15,8)})

# prepare dataframe for heatmap
sns_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra

# sns_users_monthly_revenue.fillna(0, inplace = True)
sns_ultra_users_monthly_revenue = (
    sns_ultra_users_monthly_revenue.pivot_table(index=['user_id'],
                    columns='month', values='revenue', aggfunc='sum')
)

# plot heatmap
sns.heatmap(sns_ultra_users_monthly_revenue);
```

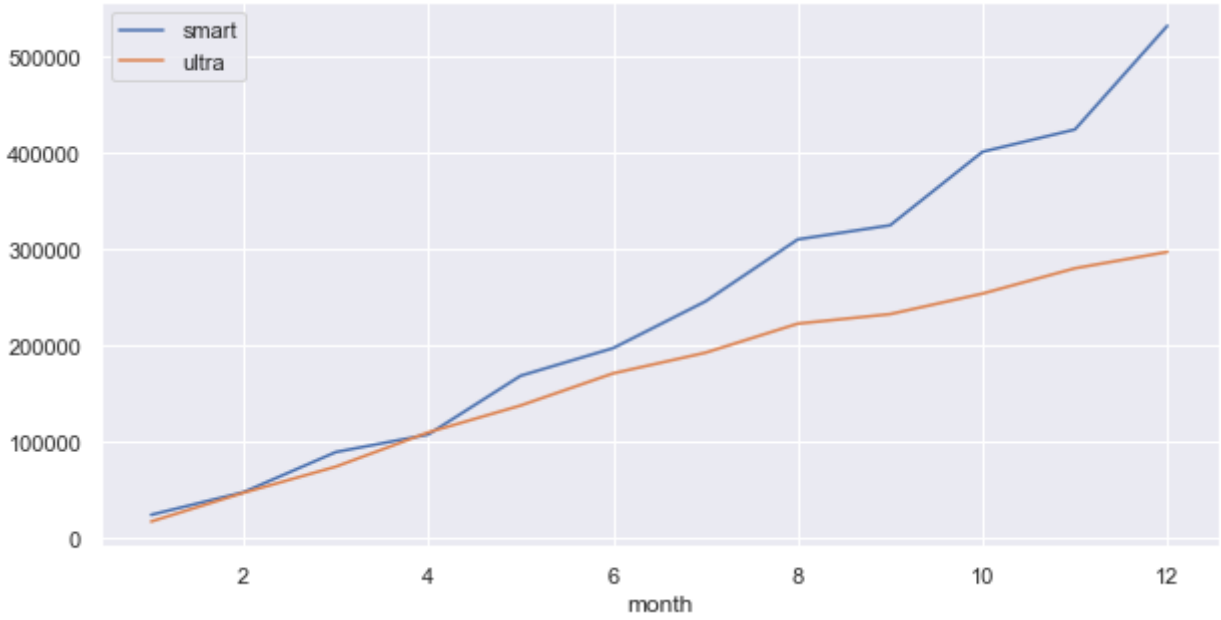



НА первый взгляд, в среднем выручка от абонента тарифа **ultra** намного **больше**, чем от от абонента тарифа **smart**

## 6.2 Посмотрим выручку в течении года в сумме по-месячно от всех абонентов, но раздельно по тарифам

In [110]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# smart
sum_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
sum_smart_users_monthly_revenue = (
    sum_smart_users_monthly_revenue.groupby(['month'])['revenue'].sum()
)
# ultra
sum_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
sum_ultra_users_monthly_revenue = (
    sum_ultra_users_monthly_revenue.groupby(['month'])['revenue'].sum()
)
# plot
ax = sum_smart_users_monthly_revenue.plot(label='smart', legend=True);
sum_ultra_users_monthly_revenue.plot(label='ultra', ax=ax, legend=True);

```

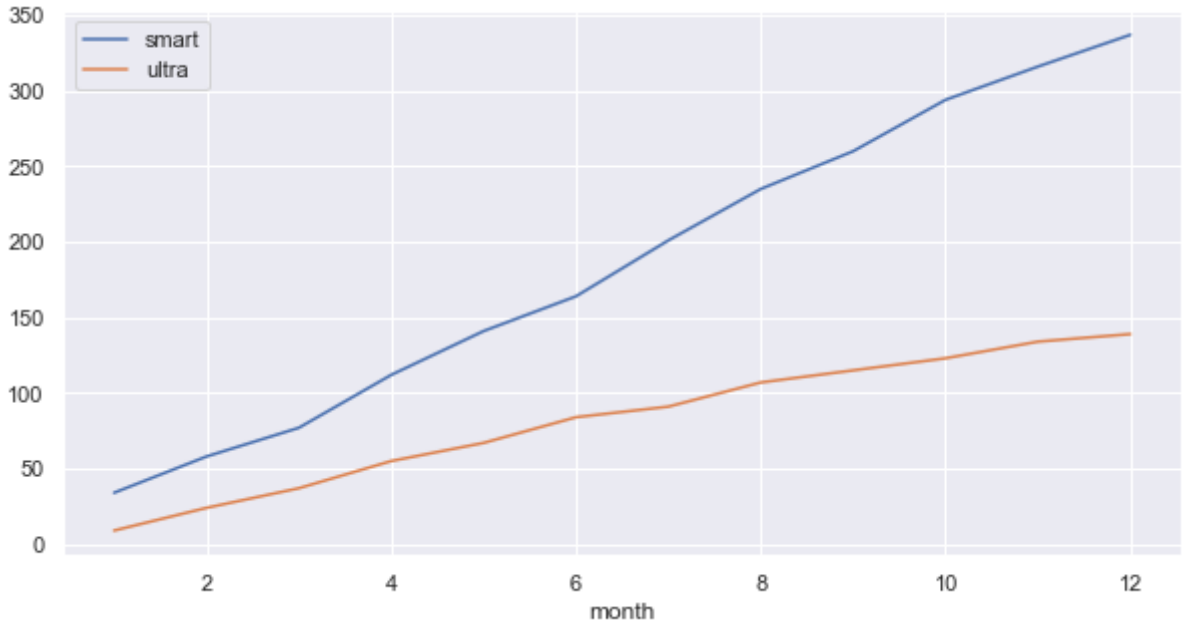



А вот "в среднем по больнице", абоненты тарифа smart в сумме приносят больше выручки компании засчет нарастающего прободания количества

## 6.3 Посмотрим, как в течении года изменялось количество абонентов по тарифам

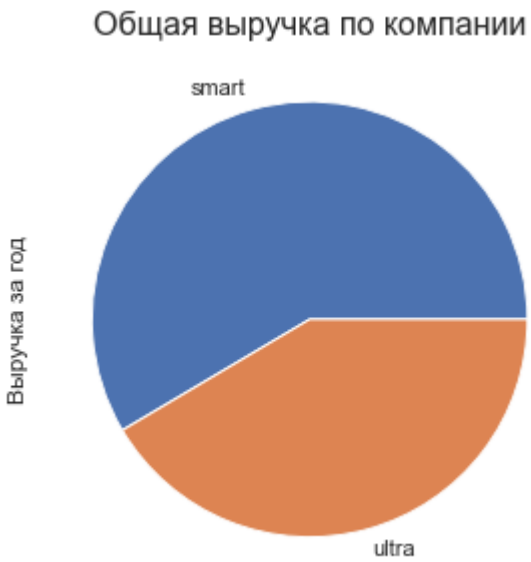
In [111]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# smart
qty_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
# only active abonents
qty_smart_users_monthly_revenue = qty_smart_users_monthly_revenue.query('revenu
qty_smart_users_monthly_revenue = (
    qty_smart_users_monthly_revenue.groupby(['month'])['user_id'].count()
)
# ultra
qty_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
# only active abonents
qty_ultra_users_monthly_revenue = qty_ultra_users_monthly_revenue.query('revenu

qty_ultra_users_monthly_revenue = (
    qty_ultra_users_monthly_revenue.groupby(['month'])['user_id'].count()
)
# plot
ax = qty_smart_users_monthly_revenue.plot(label='smart', legend=True);
qty_ultra_users_monthly_revenue.plot(label='ultra', ax=ax, legend=True);
```



In [112]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# amount per year
tarifs_revenue = (
    users_monthly_revenue.groupby(['tariff'])['revenue'].sum()
)
# plot
plt.ylabel("Выручка за год");
plt.title("Общая выручка по компании",  fontsize=16);

plt.pie(tarifs_revenue, labels = ['smart', 'ultra']);
```

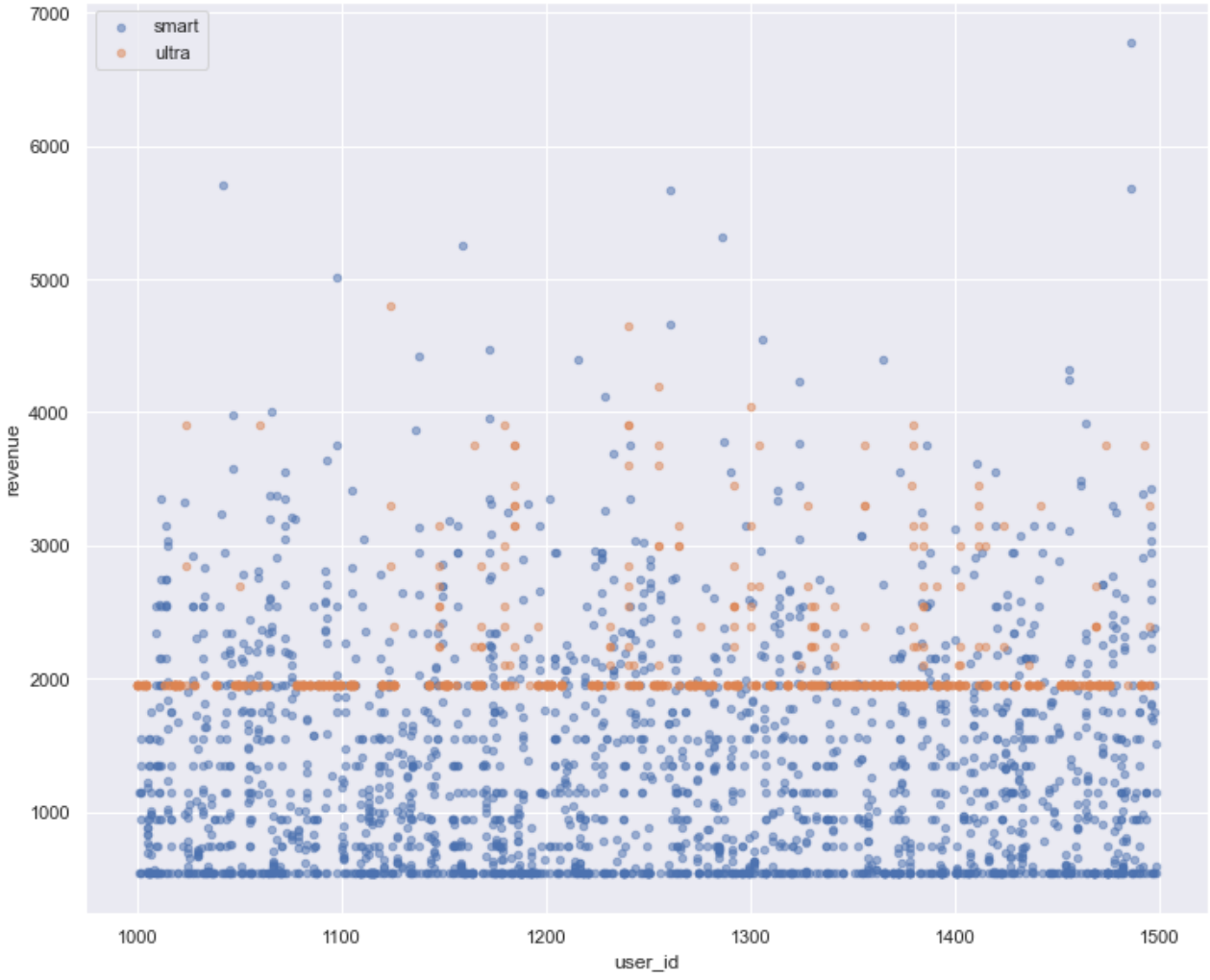



ДА, графически мы видим, что опережение выручки от тарифа smart начинается примерно с августа (8 месяц).
И с того же примерно времени доля пользователей тарифа smart также начинает расти по отношению к пользователям тарифа ultra

Построим скаттер - график по месячной выручке абонентов, раскрашенных по тарифам.

In [113]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(12,10)})
# smart
qty_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
# ultra
qty_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
# plot
ax = qty_smart_users_monthly_revenue.plot(kind='scatter', x='user_id', y='reven
qty_ultra_users_monthly_revenue.plot(kind='scatter', x='user_id', y='revenue',
```



Очень интересный результат.

- очень большая доля абонентов, кто оплачивал "минималку" и не вышел за её лимиты
  - бОльшая часть абонентов **smatr** , превысивших лимиты,оплатили **меньше** базовой стоимости **ultra** тарифа
  - уровень оплат выше базовой стоимости тарифа **ultra** практически одинаково распределён между абонентами **обоих** тарифов
  - *необычный* факт: некоторые абоненты **smart** оплачивают значительно больше всех абонентов

## 6.4 посмотрим на "среднего пользователя" в тарифах

Отсечём выручки == 0, добавив соответствующее условие в отбор.

### 6.4.1 взгляд с высоты птичьего полёта

#### *6.4.1.1 smart*

In [114]:
```
1 users_monthly_revenue.query('tariff == "smart" and revenue > 0').median()
```

Out[114]:
```
user_id         1,239.00
month               9.00
sum_duration      422.00
sum_messages       28.00
sum_mb_used    16,506.84
revenue         1,023.00
dtype: float64
```

Вот они, современные реалии: СМС-ки всё менее востребованны! )))

#### *6.4.1.2 Средний пользователь тарифа smart*

- выговаривает 420 минут
- использует ~16 GB интернет-трафика
- платит в **два** раза больше базовой абонентской платы

#### *6.4.1.3 ultra*

In [115]:
```
1 users_monthly_revenue.query('tariff == "ultra" and revenue > 0').median()
```

Out[115]:
```
user_id         1,277.00
month               9.00
sum_duration      518.00
sum_messages       38.00
sum_mb_used    19,308.01
revenue         1,950.00
dtype: float64
```

#### *6.4.1.4 Средний пользователь тарифа ultra*

#### *6.4.1.4 Средний пользователь тарифа ultra*

- выговаривает 510 минут
- использует ~19 GB интернет-трафика
- платит базовую абонентскую плату без доплат
- - остаётся в рамках предоставленных тарифом лимитов

## 6.5 Среднеарифметические показатели

### 6.5.1 smart

In [116]:
```
1 users_monthly_revenue.query('tariff == "smart" and revenue > 0').mean()
```

Out[116]:
```
user_id         1,246.51
month               8.35
sum_duration      417.93
sum_messages       33.38
sum_mb_used    16,208.39
revenue         1,289.97
dtype: float64
```

Отличия между медианными и средними значениями незначительные.
Это может означать то, что экстремальных "выбросов" не много.

### 6.5.2 ultra

In [117]:
```
1 users_monthly_revenue.query('tariff == "ultra" and revenue > 0').mean()
```

Out[117]:
```
user_id         1,263.09
month               8.25
sum_duration      526.62
sum_messages       49.36
sum_mb_used    19,468.81
revenue         2,070.15
dtype: float64
```

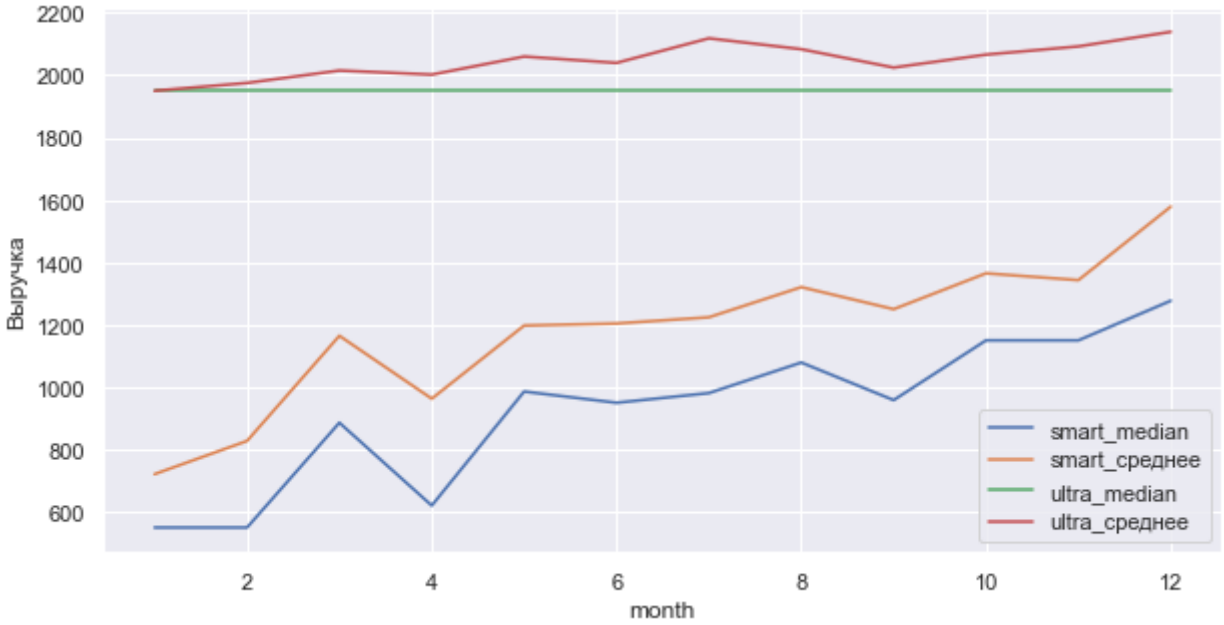
Отличия между медианными и средними значениями незначительные.
Это может означать то, что экстремальных "выбросов" не много.

## 6.6 Графическое представление данных

### 6.6.1 Выведем медианы и среднее раздельно по выручке

In [118]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# smart
med_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
med_smart_users_monthly_revenue = (
    med_smart_users_monthly_revenue.groupby(['month'])['revenue'].median()
)
mean_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smar
mean_smart_users_monthly_revenue = (
    mean_smart_users_monthly_revenue.groupby(['month'])['revenue'].mean()
)
# ultra
med_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
med_ultra_users_monthly_revenue = (
    med_ultra_users_monthly_revenue.groupby(['month'])['revenue'].median()
)
mean_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultr
mean_ultra_users_monthly_revenue = (
    mean_ultra_users_monthly_revenue.groupby(['month'])['revenue'].mean()
)
# plot
plt.ylabel("Выручка")
ax = med_smart_users_monthly_revenue.plot(label='smart_median', legend=True);
mean_smart_users_monthly_revenue.plot(label='smart_среднее', ax=ax, legend=True
med_ultra_users_monthly_revenue.plot(label='ultra_median', ax=ax, legend=True);
mean_ultra_users_monthly_revenue.plot(label='ultra_среднее', ax=ax, legend=True
```

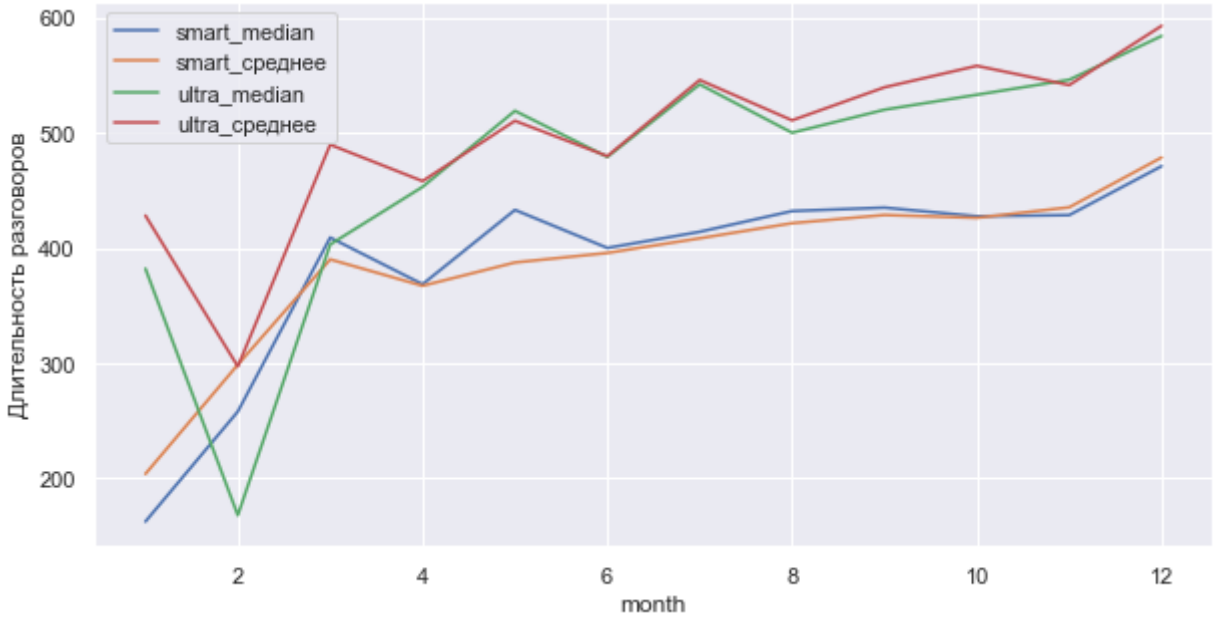



Среднеарифметическое выручки значительно превышает медианное значение в тарифе **smart**. Мы уже видели ранее, что в этом тарифе достаточно много абонентов с экстремальными показателями.

### 6.6.2 Выведем медианы и среднее раздельно по длительности разговоров

In [119]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# smart
med_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
med_smart_users_monthly_revenue = (
    med_smart_users_monthly_revenue.groupby(['month'])['sum_duration'].median()
)
mean_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smar
mean_smart_users_monthly_revenue = (
    mean_smart_users_monthly_revenue.groupby(['month'])['sum_duration'].mean()
)
# ultra
med_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
med_ultra_users_monthly_revenue = (
    med_ultra_users_monthly_revenue.groupby(['month'])['sum_duration'].median()
)
mean_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultr
mean_ultra_users_monthly_revenue = (
    mean_ultra_users_monthly_revenue.groupby(['month'])['sum_duration'].mean()
)
# plot
plt.ylabel("Длительность разговоров")
ax = med_smart_users_monthly_revenue.plot(label='smart_median', legend=True);
mean_smart_users_monthly_revenue.plot(label='smart_среднее', ax=ax, legend=True
med_ultra_users_monthly_revenue.plot(label='ultra_median', ax=ax, legend=True);
mean_ultra_users_monthly_revenue.plot(label='ultra_среднее', ax=ax, legend=True
```

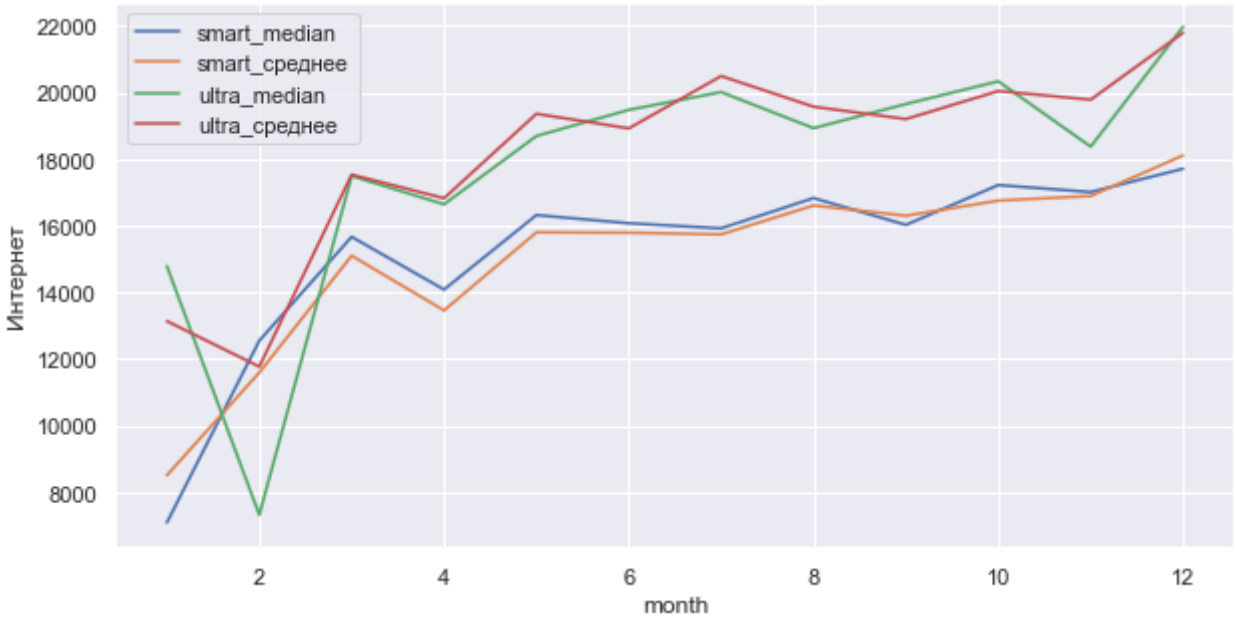



Среднеарифметические и медианные значения по длительности разговоров практически не отличаются.
Значит, выбросы в выручке обусловлены чем-то другим.

### 6.6.3 Выведем медианы и среднее раздельно по интернет-трафику

In [120]:

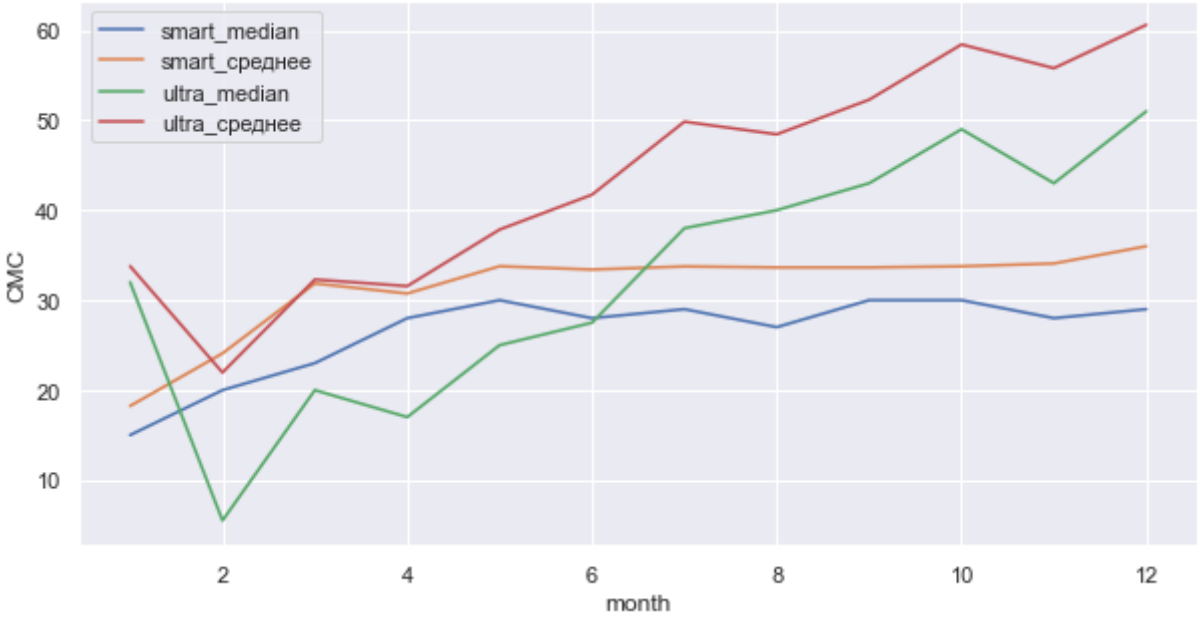
```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# smart
med_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
med_smart_users_monthly_revenue = (
    med_smart_users_monthly_revenue.groupby(['month'])['sum_mb_used'].median()
)
mean_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smar
mean_smart_users_monthly_revenue = (
    mean_smart_users_monthly_revenue.groupby(['month'])['sum_mb_used'].mean()
)
# ultra
med_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
med_ultra_users_monthly_revenue = (
    med_ultra_users_monthly_revenue.groupby(['month'])['sum_mb_used'].median()
)
mean_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultr
mean_ultra_users_monthly_revenue = (
    mean_ultra_users_monthly_revenue.groupby(['month'])['sum_mb_used'].mean()
)
# plot
plt.ylabel("Интернет")
ax = med_smart_users_monthly_revenue.plot(label='smart_median', legend=True);
mean_smart_users_monthly_revenue.plot(label='smart_среднее', ax=ax, legend=True
med_ultra_users_monthly_revenue.plot(label='ultra_median', ax=ax, legend=True);
mean_ultra_users_monthly_revenue.plot(label='ultra_среднее', ax=ax, legend=True
```




### 6.6.4 Выведем медианы и среднее раздельно по СМС

In [121]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# smart
med_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
med_smart_users_monthly_revenue = (
    med_smart_users_monthly_revenue.groupby(['month'])['sum_messages'].median()
)
mean_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smar
mean_smart_users_monthly_revenue = (
    mean_smart_users_monthly_revenue.groupby(['month'])['sum_messages'].mean()
)
# ultra
med_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
med_ultra_users_monthly_revenue = (
    med_ultra_users_monthly_revenue.groupby(['month'])['sum_messages'].median()
)
mean_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultr
mean_ultra_users_monthly_revenue = (
    mean_ultra_users_monthly_revenue.groupby(['month'])['sum_messages'].mean()
)
# plot
plt.ylabel("CMC")
ax = med_smart_users_monthly_revenue.plot(label='smart_median', legend=True);
mean_smart_users_monthly_revenue.plot(label='smart_среднее', ax=ax, legend=True
med_ultra_users_monthly_revenue.plot(label='ultra_median', ax=ax, legend=True);
mean_ultra_users_monthly_revenue.plot(label='ultra_среднее', ax=ax, legend=True
```




**Удивительное - рядом!**

Абоненты тарифа **ultra** шлёт море СМС-ок, и есть отдельные абоненты, которые делают это в немыслимых количествах.

In [122]:
```
1 users_monthly_revenue.query('tariff == "ultra" and revenue > 0').describe()
```

Out[122]:

| | user_id | month | sum_duration | sum_messages | sum_mb_used | revenue |
|---|---|---|---|---|---|---|
| **count** | 985.00 | 985.00 | 985.00 | 985.00 | 985.00 | 985.00 |
| **mean** | 1,263.09 | 8.25 | 526.62 | 49.36 | 19,468.81 | 2,070.15 |
| **std** | 140.69 | 2.87 | 317.61 | 47.80 | 10,087.17 | 376.19 |
| **min** | 1,000.00 | 1.00 | 0.00 | 0.00 | 0.00 | 1,950.00 |
| **25%** | 1,143.00 | 6.00 | 284.00 | 6.00 | 11,770.26 | 1,950.00 |
| **50%** | 1,277.00 | 9.00 | 518.00 | 38.00 | 19,308.01 | 1,950.00 |
| **75%** | 1,381.00 | 11.00 | 752.00 | 79.00 | 26,837.69 | 1,950.00 |
| **max** | 1,495.00 | 12.00 | 1,673.00 | 224.00 | 49,745.69 | 4,800.00 |

**224** - масимальное количество СМС-ок
Это почти в **шесть** раз больше медианного значения

## Комментарий ревьюера

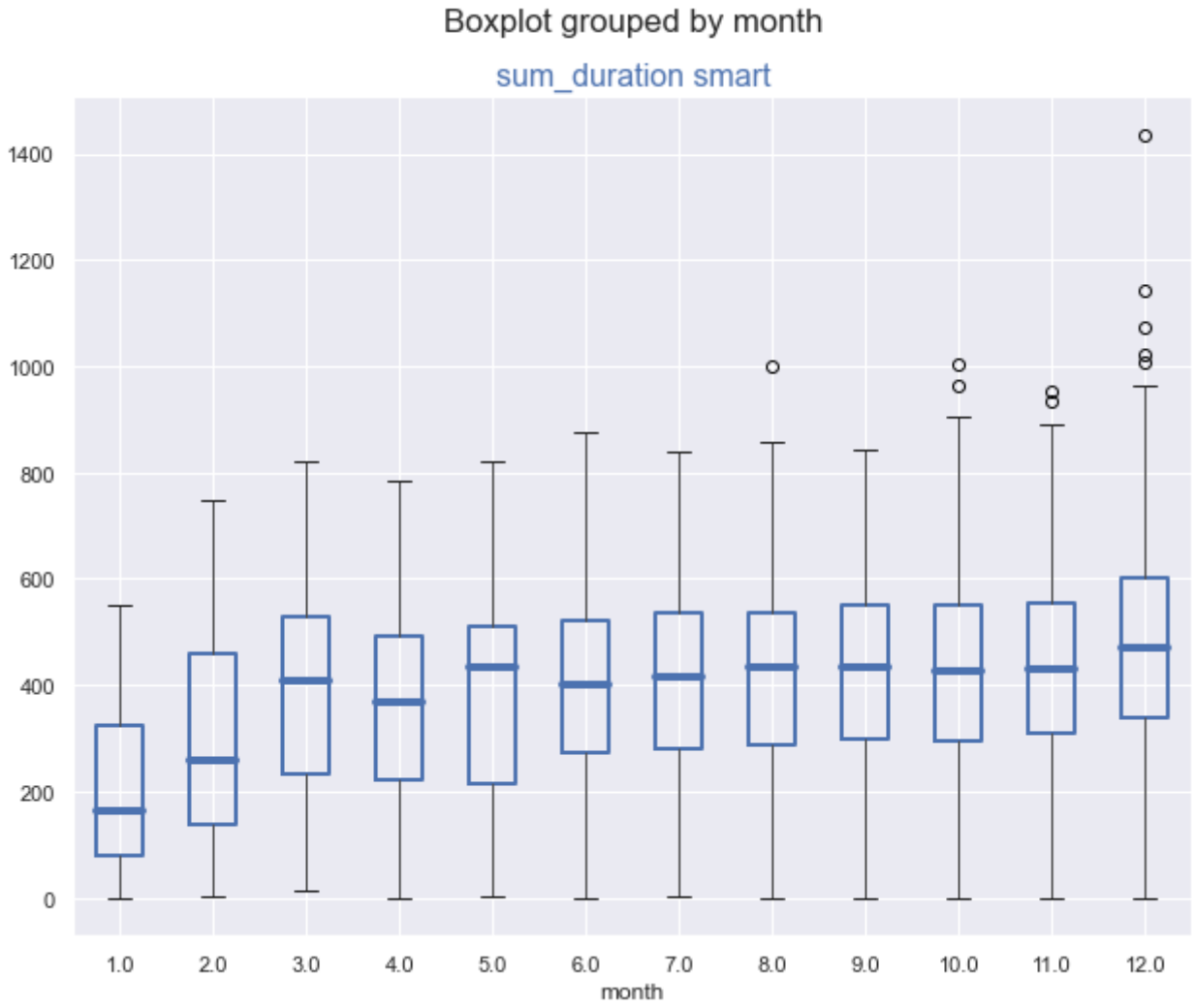
**Успех**

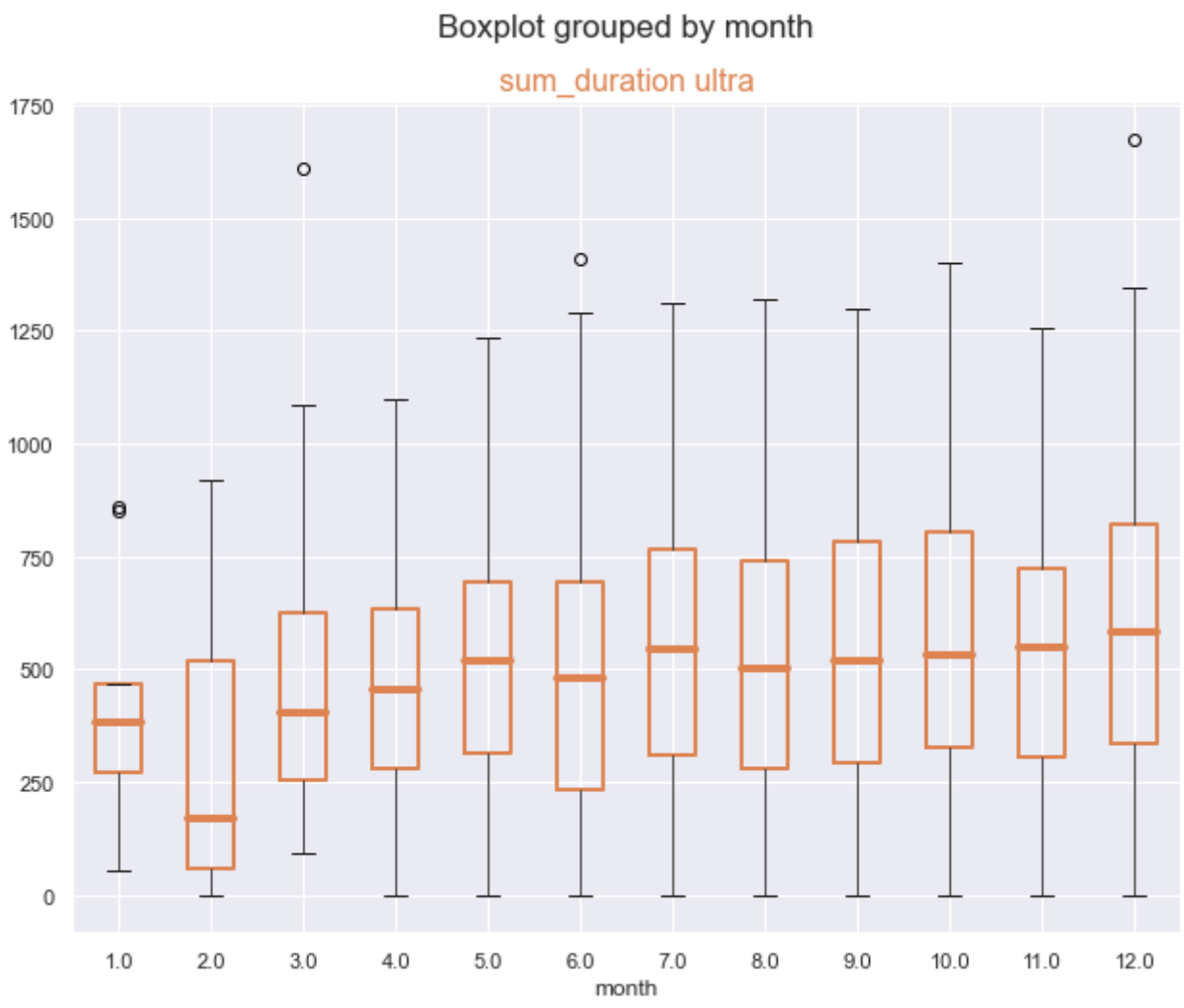
Видимо кто-то не пользуется мессенджерами, а просто СМС-сит всем подряд)

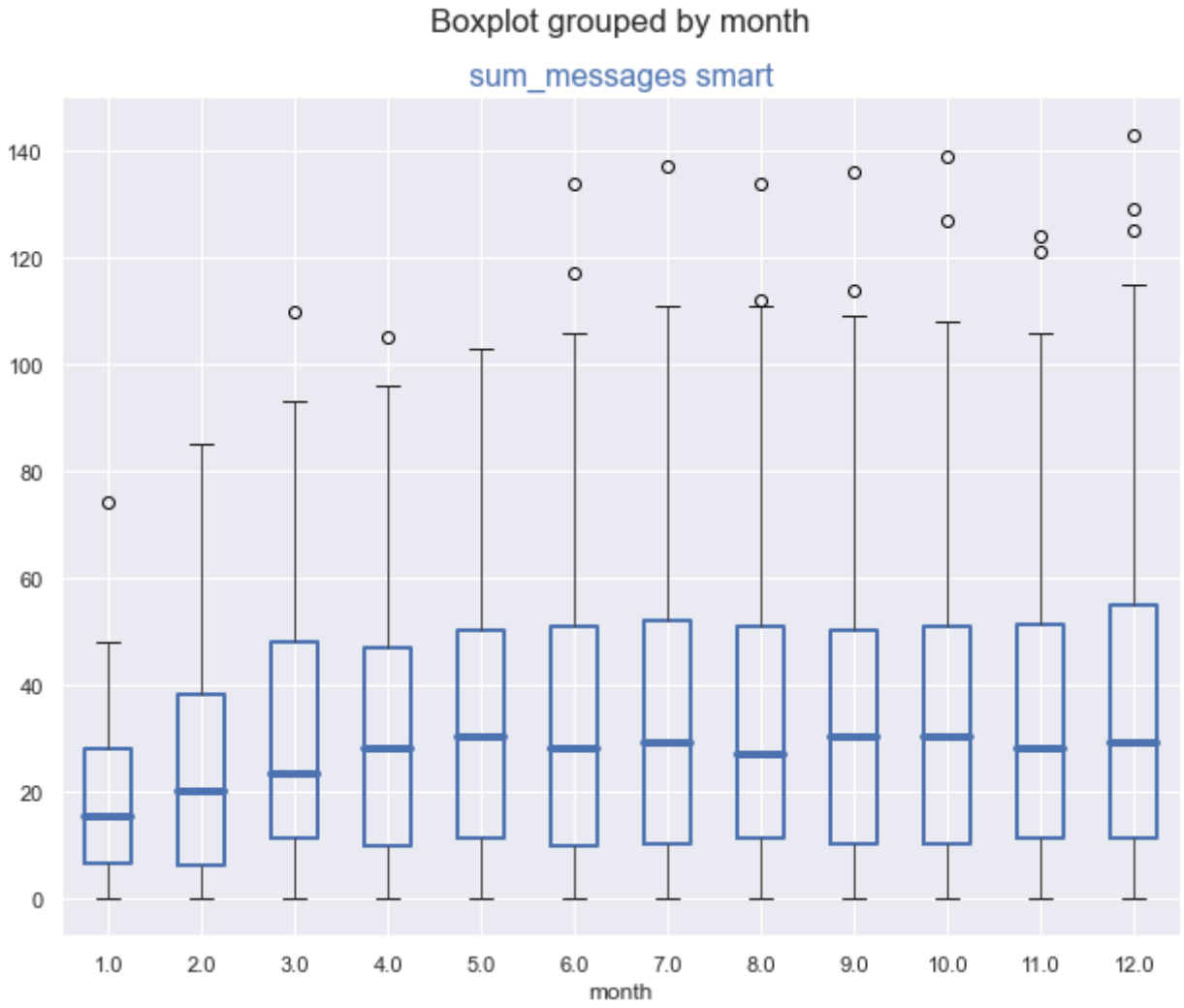
### 6.6.5 Посмотрим диаграммы размаха

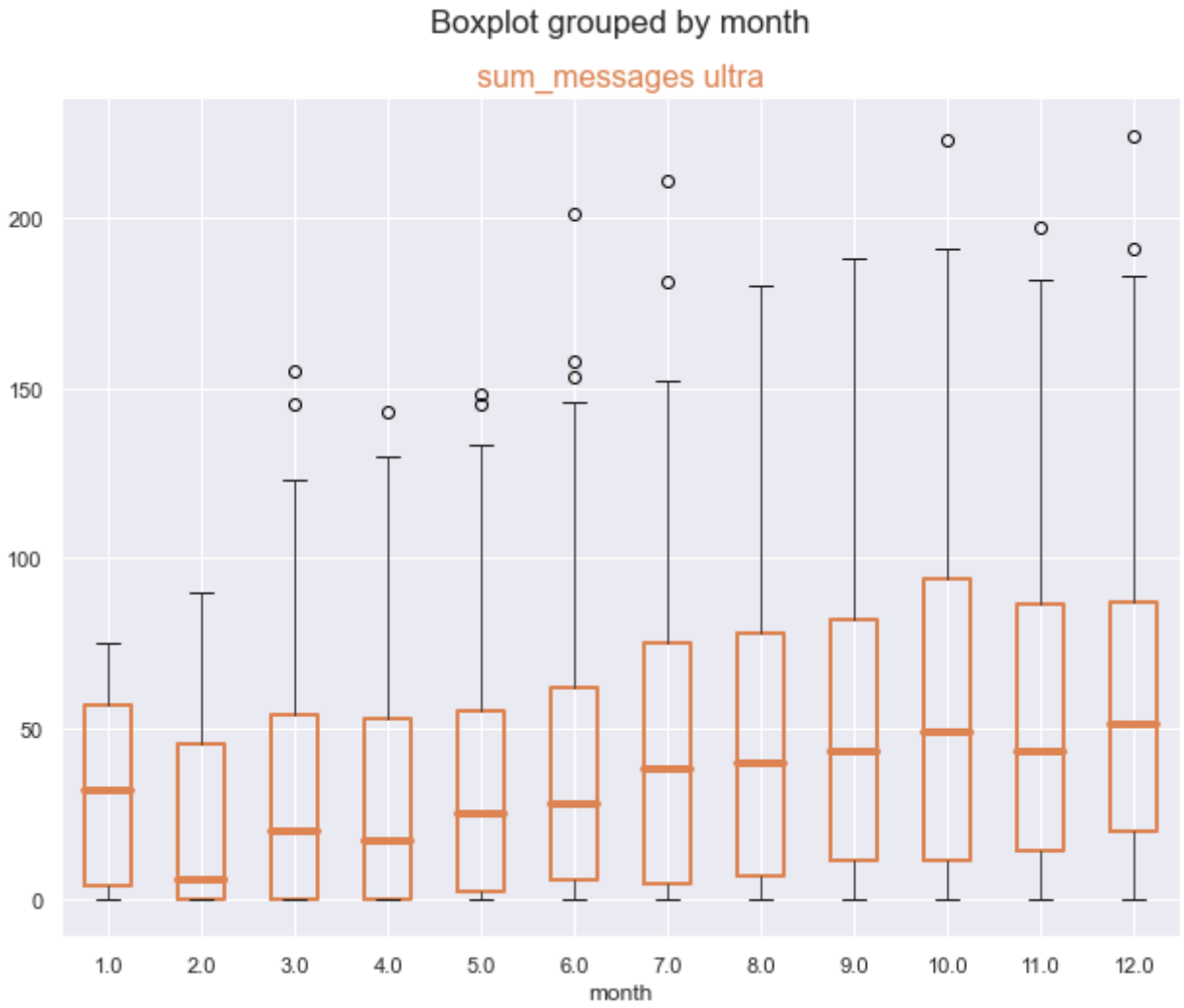
In [123]:

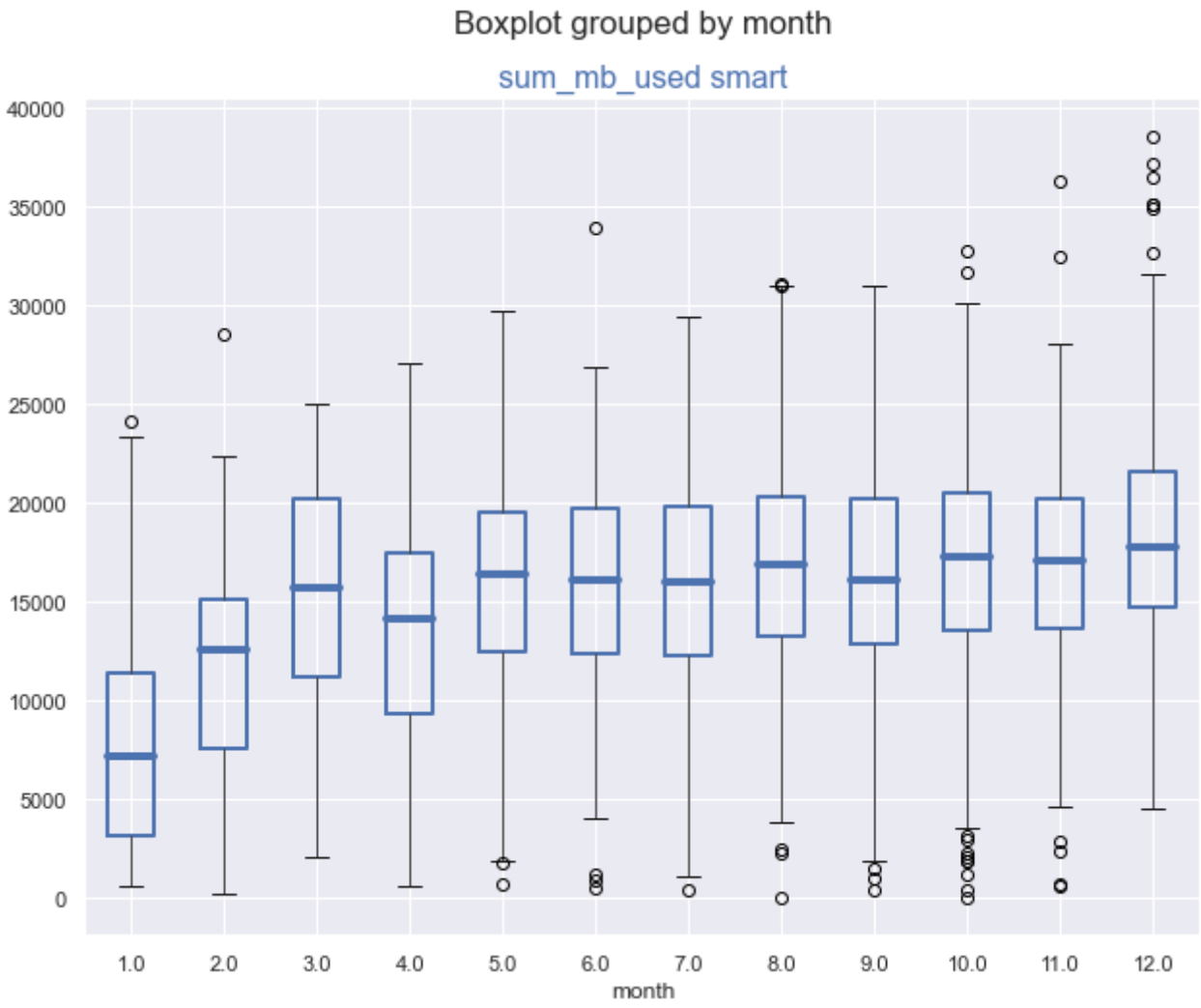
```
# setup size plot
sns.set(rc = {'figure.figsize':(10,8)})
plt.rcParams.update({'font.size': 14})
boxprops1 = dict(linestyle='-', linewidth=2, color='C0')
medianprops1 = dict(linestyle='-', linewidth=4, color='C0')
boxprops2 = dict(linestyle='-', linewidth=2, color='C1')
medianprops2 = dict(linestyle='-', linewidth=4, color='C1')

for col in ['sum_duration', 'sum_messages', 'sum_mb_used', 'revenue']:

    # smart
    smart_pivot = (
    users_monthly_revenue.query('tariff == "smart" and revenue > 0')
    .pivot_table(index=['month', 'user_id'], values=col)
    )

    # ultra
    ultra_pivot = (
    users_monthly_revenue.query('tariff == "ultra" and revenue > 0')
    .pivot_table(index=['month', 'user_id'], values=col)
    )

    # plot
    ax = smart_pivot.boxplot(column = col, by= 'month', boxprops=boxprops1, med
    ax2 = ultra_pivot.boxplot(column = col, by= 'month', boxprops=boxprops2, me

    ax.set_title(col + " smart", fontsize=16, color='C0');
    ax2.set_title(col + " ultra", fontsize=16, color='C1');

```

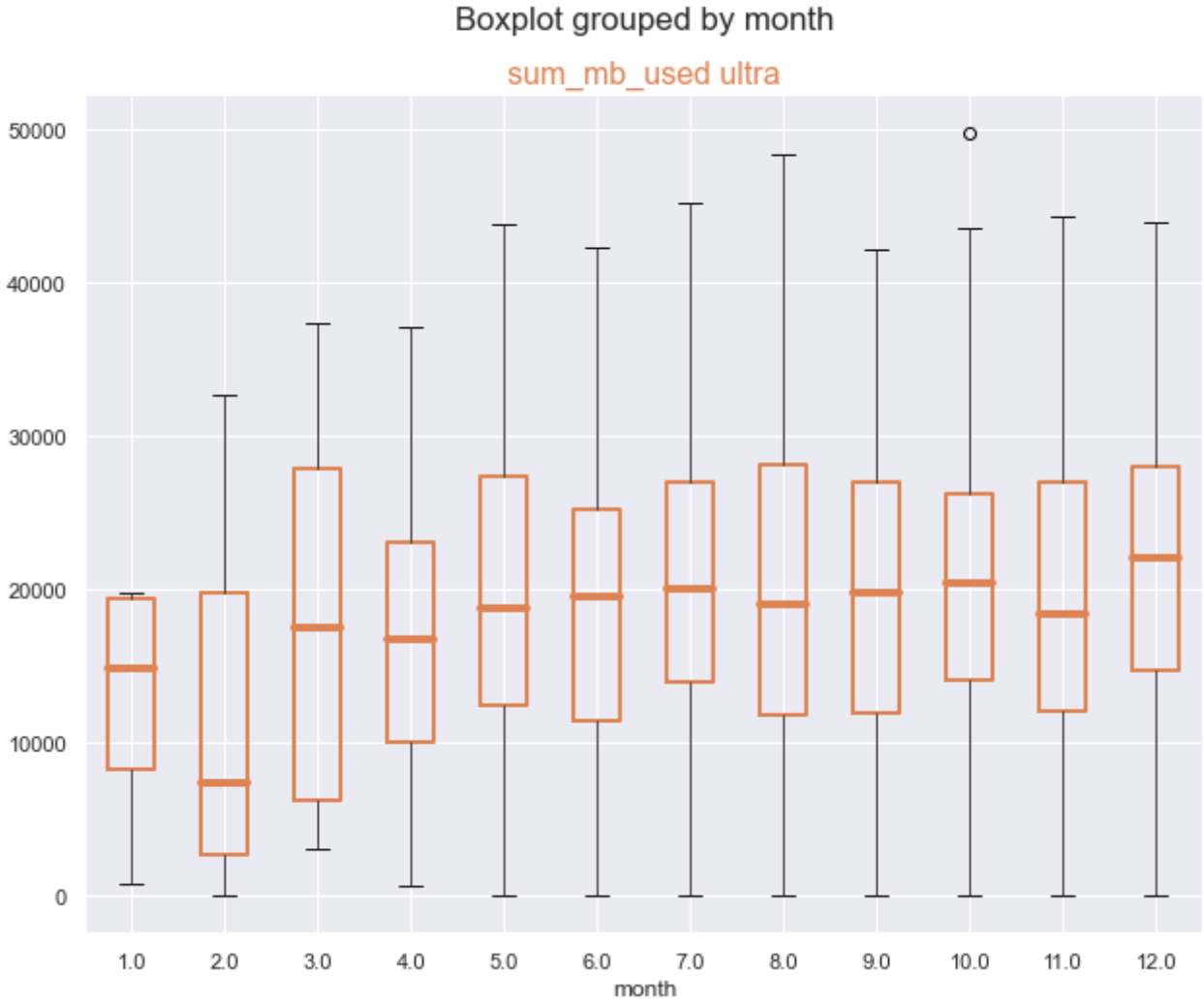


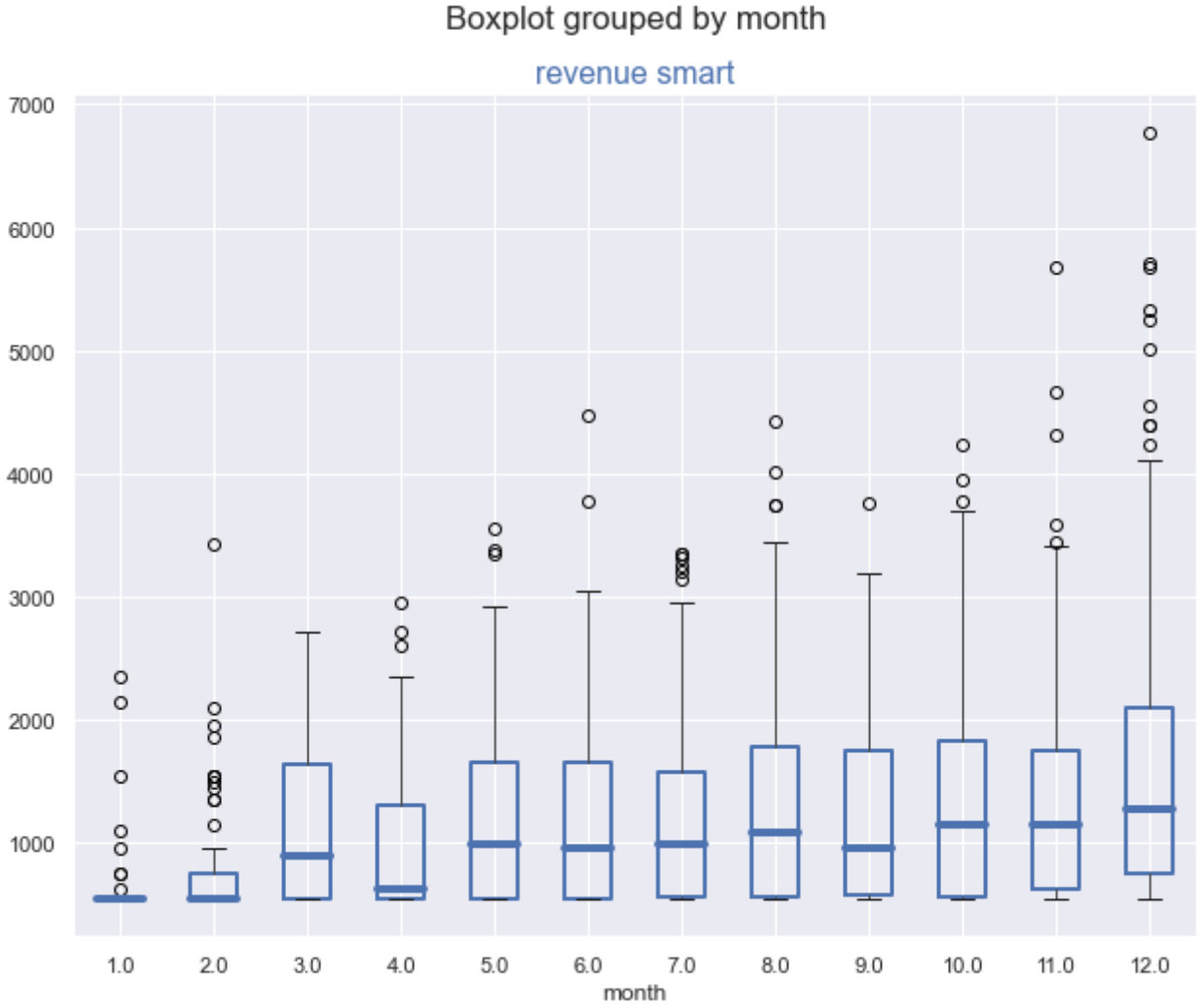

Boxplot grouped by month
sum_duration ultra
1750
1500
1250
1000
750
500
250
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
month

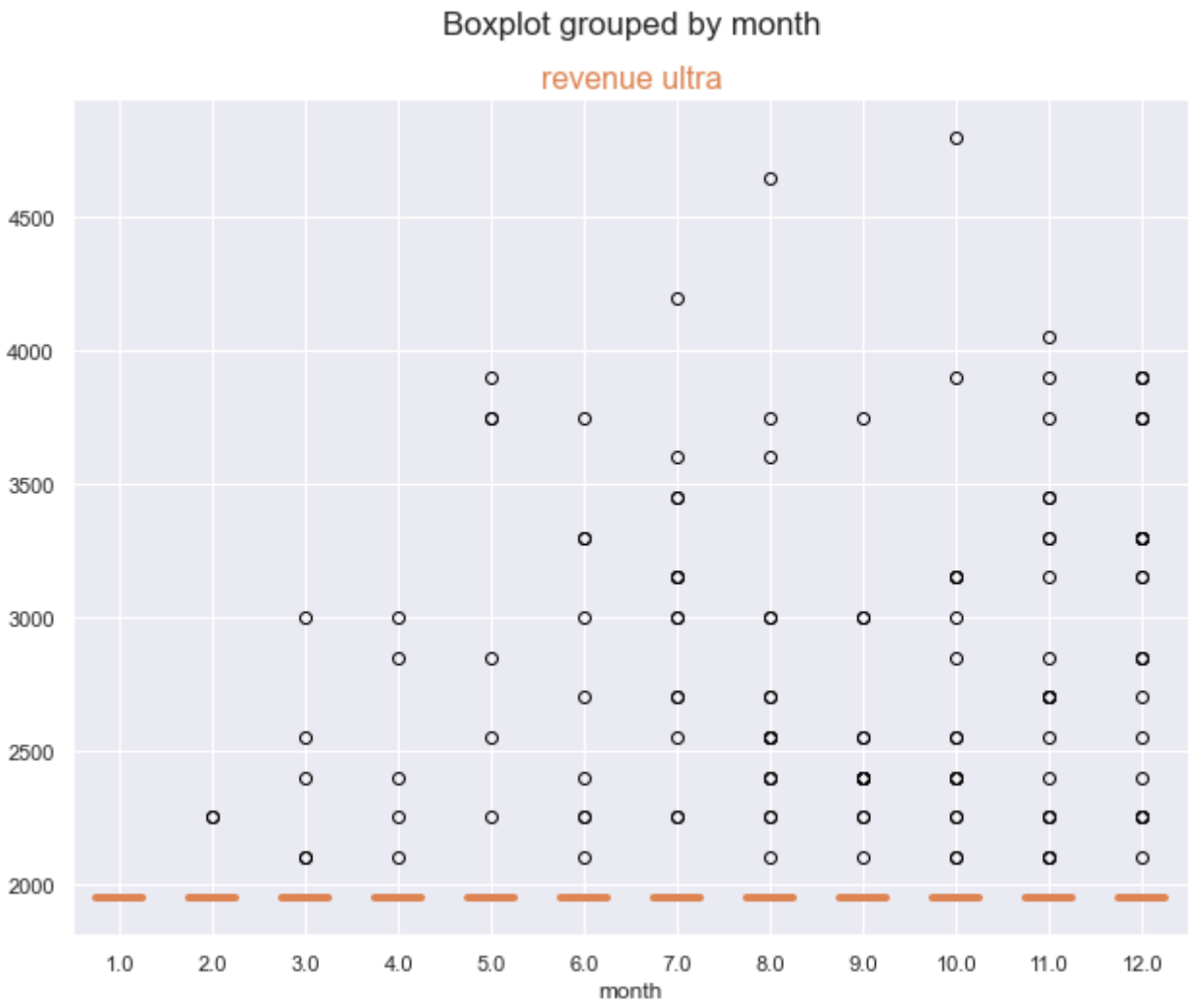

Boxplot grouped by month
sum_messages smart
140
120
100
80
60
40
20
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
month


Boxplot grouped by month
sum_messages ultra
200
150
100
50
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
month


Boxplot grouped by month
sum_mb_used smart
40000
35000
30000
25000
20000
15000
10000
5000
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
month


Boxplot grouped by month
sum_mb_used ultra
50000
40000
30000
20000
10000
0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
month

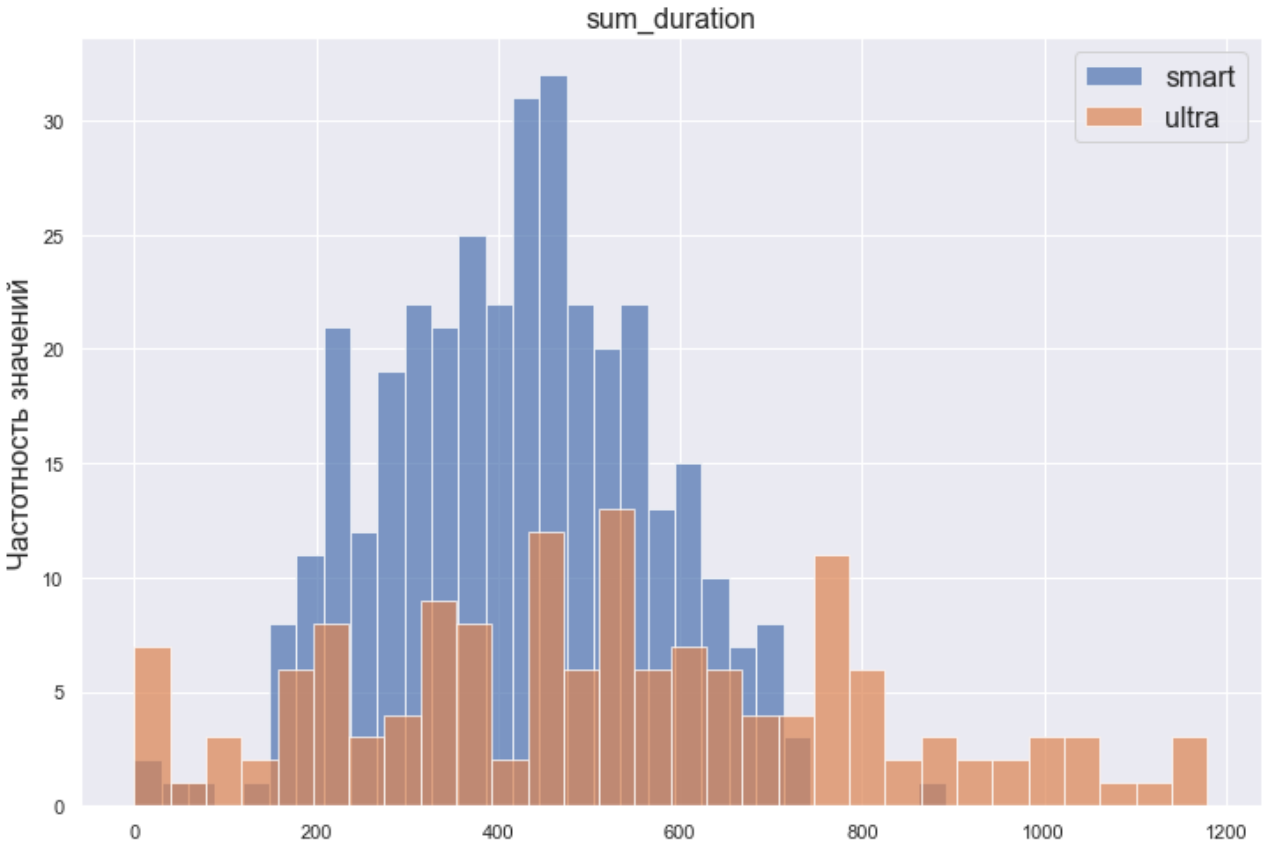

Boxplot grouped by month
revenue smart
7000
6000
5000
4000
3000
2000
1000
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
month

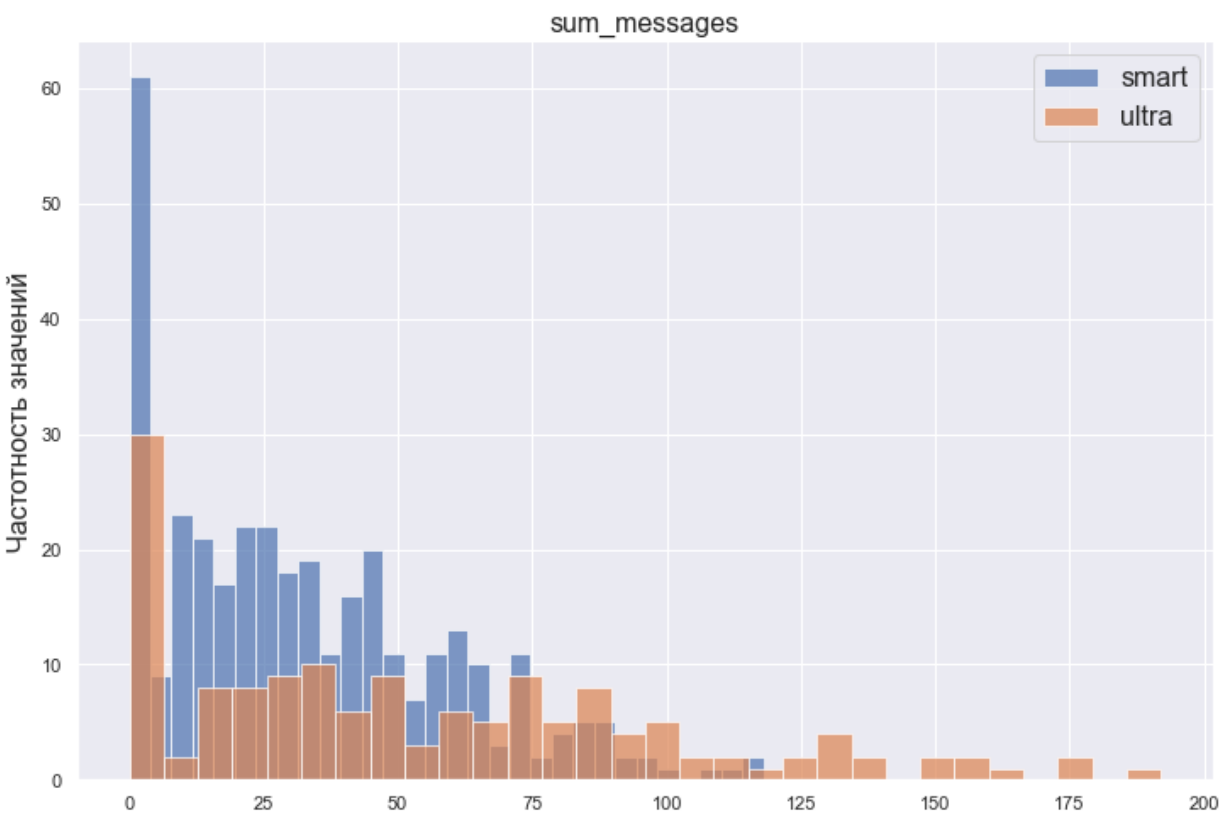

Boxplot grouped by month
revenue ultra
4500
4000
3500
3000
2500
2000
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
month

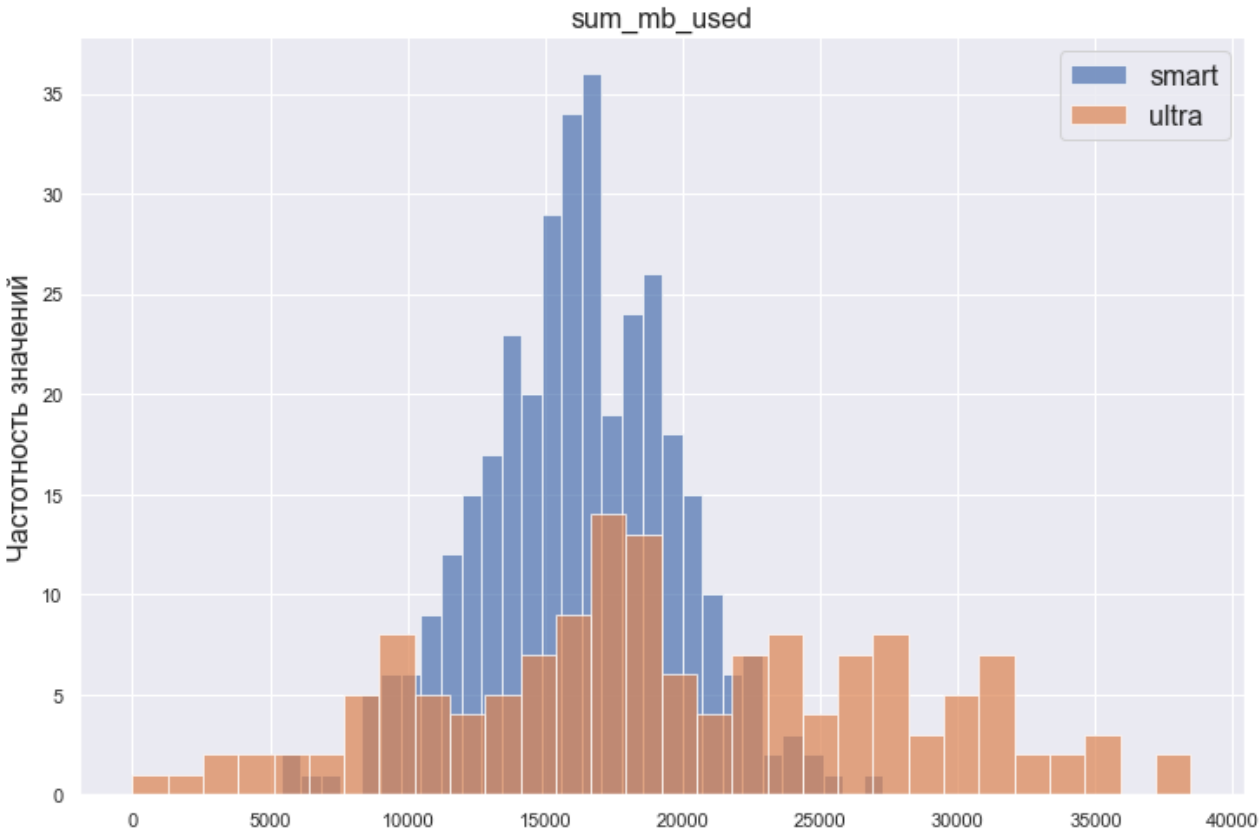
### 6.6.6 Посмотрим гистограммы в целом по году

In [124]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(12,8)})
plt.rcParams.update({'font.size': 14})
bins_const = 30

for col in ['sum_duration', 'sum_messages', 'sum_mb_used', 'revenue']:

    # smart
    smart_pivot = (
    users_monthly_revenue.query('tariff == "smart" and revenue > 0')
    .pivot_table(index=['user_id'], values=col)
    )

    # ultra
    ultra_pivot = (
    users_monthly_revenue.query('tariff == "ultra" and revenue > 0')
    .pivot_table(index=['user_id'], values=col)
    )

    # plot
    ax = smart_pivot.hist(bins=bins_const, color='C0', alpha=0.7, label="smart"
    ax2 = ultra_pivot.hist(bins=bins_const, color='C1', alpha=0.7, label="ultra

    plt.legend(fontsize=16)
    plt.title(col,  fontsize=16)
    plt.ylabel("Частотность значений", fontsize=16)

```




sum_messages
smart
ultra
Частотность значений
0
10
20
30
40
50
60
0
25
50
75
100
125
150
175
200


sum_mb_used
smart
ultra
Частотность значений
0
5
10
15
20
25
30
35
0
5000
10000
15000
20000
25000
30000
35000
40000


## 6.7 Пояснения и выводы по графикам

### 6.7.1 sum_duration - продолжительность разговоров (звонков)

#### *6.7.1.1 Диаграмма размаха*

Постепенно в течении года, диаграммы размаха обоих тарифов становятся симметричными.
Экстремальные выбросы вверх в декабре.
Оно и понятно - предпраздничная суета, завершение финансового года в большинстве компаний....

- лимит smart (500 минут) к концу года превышают примерно половина пользователей
- лимит ultra (3000 минут) не выговаривает никто

#### *6.7.1.2 Гистограмма*

Гистограмма построена в целом по году.

***smart***

Гистограмма практически имеет нормальный тип с небольшим смещением в сторону мЕньших значений.
Декабрьские аномалии сместили вершину (самое большое количество значений продолжительности разговоров) на уровень количества включённых в тариф минут (500).

- Превышение лимита встречается не редко

#### *6.7.1.3 ultra*

Гистограмма имеет очень растянутую форму.
Абоненты тарифа **ultra** не задумываются о том, сколько минут они "выговаривают".

- Количества включённых в тариф минут (3000) не достиг ни один абонент

### 6.7.2 sum_messages - количество СМС - сообщений)

#### *6.7.2.1 Диаграмма размаха*

В обоих тарифах есть уникальные выбросы и аномальные значения в бОльшую сторону в количестве СМС в месяц.
Что интересно, здесь такого большого количества экстремальных выбросов вверх в декабре, как наблюдаются в длительности звонков.
И тоже поведение абонентов разных тарифов различаются.

- лимит **smart** 50 сообщений

- - превышение лимита имеет не системный характер.
    основное количество пользователей редко используют весь лимит
- - есть небольшое число пользователей, превышающих лимит в **два** раза
- - изредка встречаются экстремальный превышения до **трёх** раз
- лимит **ultra** 1000 сообщений
- - лимит **никто** не использует в полном объёме
- - в целом абоненты ultra отправляют СМС больше, чем в тарифе smart

#### *6.7.2.2 Гистограмма*

Гистограмма построена в целом по году. **Обе** гистограммы с двумя и более пиками.
**Первый пик - это "0" и около "0"**

***smart***

Гистограмма практически имеет скошенный тип с длинным хвост вправо.
То есть с увеличением количества отправленных сообщений, уменьшается количество таких абонентов.

- Превышение лимита встречается.

#### *6.7.2.3 ultra*

Гистограмма имеет форму рваного плато.
Максимальные количества СМС абонентов тарифа **ultra** превышают максимальные количества СМС абонентов тарифа **smart** в **два** раза.

- Количества включённых в тариф СМС (1000) не достиг ни один абонент

### 6.7.3 sum_mb_used - количество интернет-трафика

#### *6.7.3.1 Диаграмма размаха*

В тарифе **smart** есть уникальные выбросы и аномальные значения и в ьОльшую сторону и в мЕньшую сторону в интерент-трафика в месяц.
В тарифе **ultra** всё выглядит намного спокойнее.

- лимит **smart** 15 Гб трафика
- - превышение лимита имеет **системный** характер.
- - основное количество пользователей останавливаются в потреблении трафика при достижении лимита.
- - есть небольшое число пользователей, превышающих лимит в **два** раза
- - изредка встречаются экстремальный превышения до **трёх** раз
- лимит **ultra** 30 Гб трафика
- - лимит в полном объёме использует небольшое количество абонентов
- - в основном превышение лимита ограничивается коэффициентом **1,5** раза
- - в целом абоненты ultra используют интернет-трафик больше, чем в тарифе smart

#### *6.7.3.2 Гистограмма*

Гистограмма построена в целом по году.

***smart***

Гистограмма практически имеет нормальный тип с небольшим смещением в сторону мЕньших значений.
Декабрьские аномалии сместили вершину (самое большое количество значений трафика) на уровень, **превышающий** количество, включённое в тариф.

- Превышение лимита встречается не редко

#### *6.7.3.3 ultra*

Гистограмма имеет форму рваного плато.
Максимальные количества интернет-трафика абонентов тарифа **ultra** превышают максимальные количества СМС абонентов тарифа **smart** почти в **два** раза.

- Количество включённого в тариф трафика превышается небольшим количеством абонентов

### 6.7.4 revenue - выручка

#### *6.7.4.1 Диаграмма размаха*

Графики тарифа **smart** и тарифа **ultra** очень разные.

- абонентская плата **smart** 550 рублей
- - переплата имеет **системный** характер.
- - основное количество пользователей оплачивают в **два** и более раз больше абоненсткой платы
- - изредка встречаются экстремальный превышения **более чем в десять** раз
- абонентская плата **ultra** 1950 рублей
- - переплата сверх абонентской платы встречается крайне редко
- - экстремальные переплаты бывают до **двукратного** размера
- - в целом абоненты ultra платят за услуги мобильной связи, чем в тарифе smart

#### *6.7.4.2 Гистограмма*

Гистограмма построена в целом по году.
**Обе** гистограммы имеют усечённое распределение, ограниченное слева базовой абонентской платой.

В тарифе **ultra** на первой позиции "небоскрёб". Он характеризует то, что практически все абоненты тарифа **ultra**, за редким исключением, не превышают включённых в тариф лимитов и не платят доплат.

# 7 Вывод по 3 шагу - анализ данных

## 7.1 Выручка

- при рассмотрении данных по **абонентам**
- - выручка больше в тарифе **ultra**
- рассмотрении данных в целом по компании
- - выручка больше от абонентов тарифа **smart**

Источник этого факта - значительное (**более, чем в два раза**), превышение количества абонентов тарифа **smart**.

### 7.2 Абонентская плата

- абоненты тарифа **ultra**, практически **переплачивают** за услуги мобильной сети.
  они крайне редко испокьзуют полностью предоставленные им лимиты в рамках абонентской платы
- абоненты тарифа **smart**, выходят за рамки предоставленных лимитов в **два и более** раз.
- - встречаются экстремальные случаи, когда абоненты тарифа **smart** платят больше, чем абоненты тарифа **ultra**

## Комментарий ревьюера

**Успех**

Супер обширные выводы, заказчик явно будет рад)

# 8 Гипотезы. Проверить

1. средняя выручка пользователей тарифов **различаются**
2. средняя выручка пользователей из Москвы **отличается** от средней выручки пользователей из других регионов

### 8.1 средние выручки пользователей тарифов различаются.

- альтернативная гипотеза:
- - средние выручки пользователей одинаковые

## Комментарий ревьюера

**Совет**

Все-таки нулевая гипотеза всегда имеет формулировку **равны**, а альтернативная - **не равны**

Вижу, что ниже ты так и делаешь, поэтому не стал подсвечивать это именно как ошибку

Средние выручки пользователей.

In [125]:

```python
# smart
mean_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smar
mean_smart_users_monthly_revenue = (
    mean_smart_users_monthly_revenue.groupby(['user_id'])['revenue'].mean()
)
# ultra
mean_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultr
mean_ultra_users_monthly_revenue = (
    mean_ultra_users_monthly_revenue.groupby(['user_id'])['revenue'].mean()
)
print("Средние выручки пользователей")
print("Тариф smart")
print(mean_smart_users_monthly_revenue.describe())
print()
print("Тариф ultra")
print(mean_ultra_users_monthly_revenue.describe())
```

```
Средние выручки пользователей
Тариф smart
count     350.00
mean    1,324.40
std       606.04
min       550.00
25%       867.55
50%     1,177.12
75%     1,667.25
max     4,333.33
Name: revenue, dtype: float64

Тариф ultra
count     148.00
mean    2,086.46
std       298.53
min     1,950.00
25%     1,950.00
50%     1,950.00
75%     1,990.62
max     3,225.00
Name: revenue, dtype: float64
```

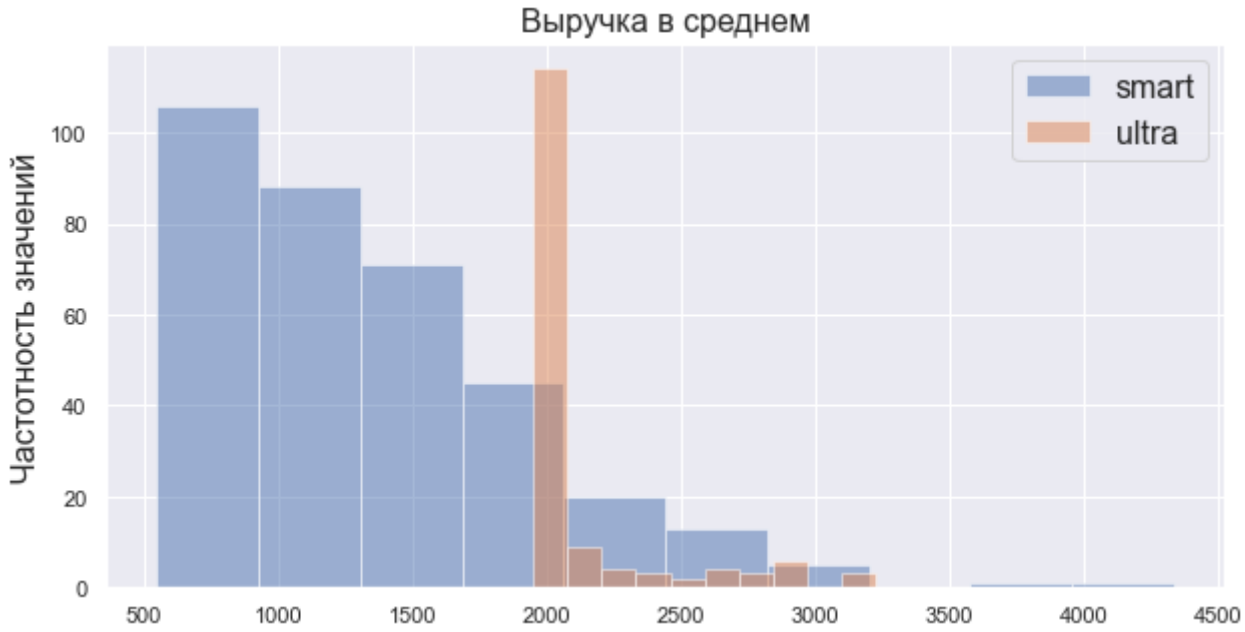
**Средние выручки пользователей тарифов различаются**

- smart среднее среднего 1324
- ultra среднее среднего 2086 Теперь проверим, верна ли гипотеза (существенное ли это различие)

Вспомним диаграмму распределения средних значений выручки по тарифам

In [126]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})

# plot
ax = mean_smart_users_monthly_revenue.plot(kind='hist',label='smart', legend=Tr
mean_ultra_users_monthly_revenue.plot(kind='hist',label='ultra', ax=ax, legend=

plt.legend(fontsize=16);
plt.title("Выручка в среднем",  fontsize=16)
plt.ylabel("Частотность значений", fontsize=16);
```




**вершинки в разных местах далеко друг от друга**
скорее всего, средние различаются существенно

Условие по проверке
**среднее.смарт == среднее.ультра**

За основу берём массивы данных

- smart mean_smart_users_monthly_revenue
- ultra mean_ultra_users_monthly_revenue
- 
  - критический уровень статистической значимости 0.05

### 8.1.1 Критерии для теста

1. генеральные совокупности **не зависят** друг от друга

2. выборочные средние **не** распределены нормально
3. выборки очень велики (350 и 148 значений)

- - в связи с большой выборкой, условие №2 можно опустить

### 8.1.2 статистические гипотезы

- нулевая: Средние выручки равны
- альтернативная: средняя выручка тарифа **ultra больше** средней выручки тарифа smart Гипотеза **односторонняя**

In [127]:
```
alpha = .05
results = st.ttest_ind(
    mean_smart_users_monthly_revenue,
    mean_ultra_users_monthly_revenue)
print('p-значение: ', results.pvalue)

if results.pvalue/2 < alpha:
    print("Средние выручки по тарифам различаются")
else:
    print("Не подтверждается различие средних выручек по тарифам")

```

```
p-значение:  2.983728825353049e-40
Средние выручки по тарифам различаются
```

### 8.1.3 средние выручки пользователей тарифов различаются.

Подтверждено статистически

## 8.2 средняя выручка пользователей из Москвы отличается от средней выручки пользователей из других регионов

- альтернативная гипотеза:
- - средние выручки пользователей одинаковые

Подготовим массивы данных для проверки гипотезы.
Вспомним, какие данные у нас есть:

In [128]:
```
users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   sum_messages  6000 non-null   float64
 4   sum_mb_used   6000 non-null   float64
 5   city          6000 non-null   object 
 6   tariff        6000 non-null   object 
 7   revenue       6000 non-null   float64
dtypes: float64(6), object(2)
memory usage: 421.9+ KB
```

Отлично! Всё, что надо есть.
Подготовим массивы данных.

Средняя выручка жителей Москвы
mean_moscow_users_revenue

In [129]:
```
mean_moscow_users_revenue = users_monthly_revenue.query('city == "Москва" and r
mean_moscow_users_revenue = (
    mean_moscow_users_revenue.groupby(['user_id'])['revenue'].mean()
)
```

Средняя выручка жителей других городов
mean_no_moscow_users_revenue

In [130]:
```
mean_no_moscow_users_revenue = users_monthly_revenue.query('city != "Москва" an
mean_no_moscow_users_revenue = (
    mean_no_moscow_users_revenue.groupby(['user_id'])['revenue'].mean()
)
```

In [131]:
```
print("Средние выручки пользователей")
print("Москвы")
print(mean_moscow_users_revenue.describe())
print()
print("Других городов")
print(mean_no_moscow_users_revenue.describe())
```

```
Средние выручки пользователей
Москвы
count      99.00
mean    1,498.47
std       606.48
min       550.00
25%     1,020.83
50%     1,600.86
75%     1,950.00
max     2,935.50
Name: revenue, dtype: float64

Других городов
count     399.00
mean    1,563.88
std       644.47
min       550.00
25%       993.39
50%     1,619.00
75%     1,950.00
max     4,333.33
Name: revenue, dtype: float64
```

**Средние выручки пользователей из Москвы и других городов различаются**

- Москва среднее среднего 1498
- Другие города среднее среднего 1563 Теперь проверим, верна ли гипотеза (существенное ли это различие)

Построим диаграмму распределения средних значений выручки по городам

In [132]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})

# plot
ax = mean_no_moscow_users_revenue.plot(kind='hist',label='Другие', legend=True,
mean_moscow_users_revenue.plot(kind='hist',label='Москва', ax=ax, legend=True,

plt.legend(fontsize=16);
plt.title("Выручка в среднем",  fontsize=16)
plt.ylabel("Частотность значений", fontsize=16);
```


**вершинки в одинаковых местах**
скорее всего, средние не существенно различаются

Условие по проверке
**среднее.Москва == среднее.Другие**

### 8.2.1 Критерии для теста

1. генеральные совокупности **не зависят** друг от друга
2. выборочные средние **не** распределены нормально
3. выборки очень велики (99 и 399 значений)

- ▪ в связи с большой выборкой, условие №2 можно опустить

### 8.2.2 статистические гипотезы

- нулевая: Средние выручки равны
- альтернативная: средняя выручка жителей **Москвы меньше** средней выручки жителей **других** городов Гипотеза **односторонняя**

За основу берём массивы данных

- Москва mean_moscow_users_revenue
- Другие mean_no_moscow_users_revenue

- - критический уровень статистической значимости 0.05

In [133]:
```
alpha = .05
results = st.ttest_ind(
    mean_moscow_users_revenue,
    mean_no_moscow_users_revenue)
print('p-значение: ', results.pvalue)

if results.pvalue/2 < alpha:
    print("Средние выручки по городам различаются")
else:
    print("Не подтверждается различие средних выручек по городам")

```

```
p-значение:  0.36094633224451433
Не подтверждается различие средних выручек по городам
```

#### 8.2.3 средняя выручка пользователей из Москвы отличается от средней выручки пользователей из других регионов

Статистически **опровергнуто**

# 9 Общий вывод

## 9.1 Запрос бизнеса

**какой тариф приносит больше денег**

за отчётный период (2018 год), больше денег принёс тариф

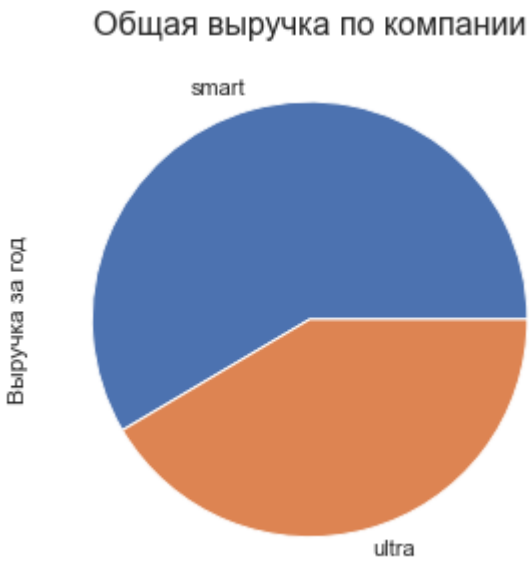
### 9.1.1 smart

In [134]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# smart
sum_smart_users_monthly_revenue = users_monthly_revenue.query('tariff == "smart
sum_smart_users_monthly_revenue = (
    sum_smart_users_monthly_revenue.groupby(['month'])['revenue'].sum()
)
# ultra
sum_ultra_users_monthly_revenue = users_monthly_revenue.query('tariff == "ultra
sum_ultra_users_monthly_revenue = (
    sum_ultra_users_monthly_revenue.groupby(['month'])['revenue'].sum()
)
# plot
plt.ylabel("Выручка")
plt.title("Общая выручка по компании",  fontsize=16)
ax = sum_smart_users_monthly_revenue.plot(label='smart', legend=True);
sum_ultra_users_monthly_revenue.plot(label='ultra', ax=ax, legend=True);
```

In [135]:

```
# setup size plot
sns.set(rc = {'figure.figsize':(10,5)})
# amount per year
tarifs_revenue = (
    users_monthly_revenue.groupby(['tariff'])['revenue'].sum()
)
# plot
plt.ylabel("Выручка за год");
plt.title("Общая выручка по компании",  fontsize=16);

plt.pie(tarifs_revenue, labels = ['smart', 'ultra']);
```




ДА, графически мы видим, что опережение выручки от тарифа smart начинается примерно с августа (8 месяц).
И с того же примерно времени доля пользователей тарифа smart также начинает расти по отношению к пользователям тарифа ultra

## 9.2 Запрос бизнеса

**какой тариф лучше**

### 9.2.1 Статистически подтверждено, что средние выручки абонентов различаются по тарифам.

In [136]:

```
alpha = .05
results = st.ttest_ind(
    mean_smart_users_monthly_revenue,
    mean_ultra_users_monthly_revenue)
if results.pvalue < alpha:
    print("Средние выручки по тарифам различаются")
else:
    print("Не подтверждается различие средних выручек по тарифам")

```

```
Средние выручки по тарифам различаются
```

### 9.2.2 в тарифе ultra средняя выручка больше

In [137]:

```
# smart
mean_smart_revenue = (
 users_monthly_revenue.query('tariff == "smart" and revenue > 0')['revenue'].me
)
# ultra
mean_ultra_revenue = (
 users_monthly_revenue.query('tariff == "ultra" and revenue > 0')['revenue'].me
)
print('В smart средняя выручка', round(mean_smart_revenue), "руб")
print()
print('В ultra средняя выручка', round(mean_ultra_revenue), "руб")
print()
```

```
В smart средняя выручка 1290 руб

В ultra средняя выручка 2070 руб
```

## 9.3 Гипотезы

## 9.4 Дайджест исследования

### 9.4.1 Исходные данные предоставлены в прекрасном виде.

Единственное, что потребовалось сделать - это конвертировать даты во внутренний формат программы.

### 9.4.2 Таблица с параметрами тарифов переформатирована

в формат **словаря** программы.
Этот шаг позволит в дальнейшем проще обращаться к значениям по их наименованию и менять данные тарифов при измении ввлдных или в качестве **экспериментов**

### 9.4.3 Служебные таблицы

Созданы служебные таблицы

- users_calls - для количество сделанных звонков и израсходованных минут разговора по месяцам
- user_messages - количество отправленных сообщений по месяцам
- users_internet - объем израсходованного интернет-трафика по месяцам И на их основе - финальная таблица, предназначенная для статистического анализа
- users_monthly_revenue - помесячная выручка с каждого пользователя

In [138]:
```
1 users_calls.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 202607 entries, 0 to 202606
Data columns (total 8 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   id                202607 non-null  object
 1   call_date         202607 non-null  datetime64[ns]
 2   duration          202607 non-null  float64
 3   user_id           202607 non-null  int64
 4   city              202607 non-null  object
 5   tariff            202607 non-null  object
 6   month             202607 non-null  int64
 7   reduced_duration  202607 non-null  float64
dtypes: datetime64[ns](1), float64(2), int64(2), object(3)
memory usage: 13.9+ MB
```

In [139]:
```
1 user_messages.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 123036 entries, 0 to 123035
Data columns (total 6 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            123036 non-null  object
 1   message_date  123036 non-null  datetime64[ns]
 2   user_id       123036 non-null  int64
 3   city          123036 non-null  object
 4   tariff        123036 non-null  object
 5   month         123036 non-null  int64
dtypes: datetime64[ns](1), int64(2), object(3)
memory usage: 6.6+ MB
```

In [140]:
```
1 users_internet.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 149396 entries, 0 to 149395
Data columns (total 7 columns):
 #   Column        Non-Null Count   Dtype
---  ------        --------------   -----
 0   id            149396 non-null  object
 1   mb_used       149396 non-null  float64
 2   session_date  149396 non-null  datetime64[ns]
 3   user_id       149396 non-null  int64
 4   city          149396 non-null  object
 5   tariff        149396 non-null  object
 6   month         149396 non-null  int64
dtypes: datetime64[ns](1), float64(1), int64(2), object(3)
memory usage: 9.1+ MB
```

In [141]:
```
1 users_monthly_revenue.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6000 entries, 0 to 5999
Data columns (total 8 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   user_id       6000 non-null   float64
 1   month         6000 non-null   float64
 2   sum_duration  6000 non-null   float64
 3   sum_messages  6000 non-null   float64
 4   sum_mb_used   6000 non-null   float64
 5   city          6000 non-null   object 
 6   tariff        6000 non-null   object 
 7   revenue       6000 non-null   float64
dtypes: float64(6), object(2)
memory usage: 421.9+ KB
```

#### 9.4.4 В соответствии с правилами тарифов,

рассчитана по-месячная плата за услуги мобильной связи для каждого клиента

##### *9.4.4.1 Алгоритм расчёта:*

1. проверим, являлся ли этот **user_id** абонентом в этом месяце

- если значения во всех трёх колонках == "0", то **revenue=0**
- иначе
  - revenue = **rub_monthly_fee** из tariffs
  - вычисляем сумму доплат **add_pay**
  - каждую доплату **прибавляем** к revenue

2. сравнить длительность звонков **sum_duration** с количеством, включённый в тариф

```
**minutes_included**
значение minutes_included зависит от тарифа
```

- тариф считываем из текущей строки
- - запишем в переменную **tariff**
- если sum_duration **больше** minutes_included
- - к revenue добавляем дополнительную стоимость
- для этого
- - из sum_duration вычитаем minutes_included и умножаем на
- - - стоимость дополнительной минуты **rub_per_minute** в tariffs_dict
    полученное значение **add_pay** прибавляем к **revenue**

3. сравнить количество сообщений **sum_messages** с количеством, включённый в тариф

```
**messages_included**
значение minutes_included зависит от тарифа
тариф для этого пользователя у нас хранится во временной переменной **tari
ff**, определённой на шаге 2.
```

- если sum_messages **больше** messages_included
- - к revenue добавляем дополнительную стоимость
- для этого
- - из sum_messages вычитаем messages_included и умножаем на

- - - - стоимость дополнительных сообщений **rub_per_message** в tariffs_dict
    полученное значение **add_pay** прибавляем к **revenue**

4. сравнить количество траффика **sum_mb_used** с количеством, включённый в тариф

```
**mb_per_month_included**
значение mb_per_month_included зависит от тарифа
тариф для этого пользователя у нас хранится во временной переменной **tari
ff**, определённой на шаге 2.
```

- если sum_mb_used **больше** mb_per_month_included
- - к revenue добавляем дополнительную стоимость
- для этого

- - из sum_mb_used вычитаем mb_per_month_included (**add_traffic**) и умножаем на
- - - стоимость дополнительного трафиика **rub_per_gb** в tariffs_dict
- *предварительно**
- - переводим add_traffic в **gb_add_traffic**
- - - для этого add_traffic разделить на 1024

полученное значение **add_pay** прибавляем к **revenue**

## 9.5 Анализ данных

подробно смотрим в разделе 6 Анализируем данные исследования
**основной вывод**:

### 9.5.1 Выручка

- при рассмотрении данных по **абонентам**
- - выручка больше в тарифе **ultra**
- рассмотрении данных в целом по компании
- - выручка больше от абонентов тарифа **smart**

Источник этого факта - значительное (**более, чем в два раза**), превышение количества абонентов тарифа **smart**.

### 9.5.2 Абонентская плата

- абоненты тарифа **ultra**, практически **переплачивают** за услуги мобильной сети.
  они крайне редко испокьзуют полностью предоставленные им лимиты в рамках абонентской платы
- абоненты тарифа **smart**, выходят за рамки предоставленных лимитов в **два и более** раз.
- - встречаются экстремальные случаи, когда абоненты тарифа **smart** платят больше, чем абоненты тарифа **ultra**

автор исследования Эдуард Дементьев

Кратко о себе

лет 10 помощником в элитных часах

лет 15 в корпоративных сувенирах (банки, телеком, спорт)

- ну там всё делал и учёт и отчётность и вёрстка и макетирование и продажи и контроль производства и т д и т п

5 лет https://silanavsegda.ru/ (https://silanavsegda.ru/)

Ах да! Ещё ведь водителем такси 17 лет. )))

Ещё сайтик https://eddydewrussia.ru/ (https://eddydewrussia.ru/)

Это - мой первый самостоятельный анализ данных.

https://clck.ru/Z5cgg (https://clck.ru/Z5cgg)

Это - небольшой рассказ про себя
https://www.youtube.com/watch?v=zJSsrUkJolo&t=1616s (https://www.youtube.com/watch?v=zJSsrUkJolo&t=1616s)