Определение перспективного тарифа для телеком-компании

Содержание

Компания «Мегалайн» — федеральный оператор сотовой связи. Клиентам предлагают два тарифных плана: «Смарт» и «Ультра». Чтобы скорректировать рекламный бюджет, коммерческий департамент хочет понять, какой тариф приносит больше денег.

Требуется сделать предварительный анализ тарифов на небольшой выборке клиентов «Мегалайна» в количестве 500 пользователей: 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 (выгоднее провайдеру).

Описание тарифов

Тариф «Смарт»

1. Ежемесячная плата: 550 рублей
2. Включено 500 минут разговора, 50 сообщений и 15 Гб интернет-трафика
3. Стоимость услуг сверх тарифного пакета:
   - минута разговора: 3 рубля
   - сообщение: 3 рубля
   - 1 Гб интернет-трафика: 200 рублей

Тариф «Ультра»

1. Ежемесячная плата: 1950 рублей
2. Включено 3000 минут разговора, 1000 сообщений и 30 Гб интернет-трафика
3. Стоимость услуг сверх тарифного пакета:
   - минута разговора: 1 рубль
   - сообщение: 1 рубль
   - 1 Гб интернет-трафика: 150 рублей

Примечания:

«Мегалайн» всегда округляет секунды до минут, а мегабайты — до гигабайт. Каждый звонок округляется отдельно: даже если он длился всего 1 секунду, будет засчитан как 1 минута.

Для веб-трафика отдельные сессии не считаются. Вместо этого общая сумма за месяц округляется в бóльшую сторону. Если абонент использует 1025 мегабайт в этом месяце, с него возьмут плату за 2 гигабайта.

Данные хранятся в файлах:

- /datasets/users.csv
- /datasets/calls.csv
- /datasets/messages.csv
- /datasets/internet.csv
- /datasets/tariffs.csv

Обзор данных

Начнём с обзора данных. Для этого последовательно загрузим данные из всех пяти файлов и изучим их.

Импотрируем необходимые библиотеки:

In [1]:
```
import pandas as pd
import math as mt
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats as st
```

Определим функцию обзора данных:

In [2]:
```
# определение функции обзора данных
# ==================================
# на вход подаётся датафрейм df
# на выходе выводится:
#     - 10 случайных строк df
#     - информация df.info()
#     - количество явных дубликатов в строках df
#     - процент пропусков данных в столбцах df
def data_observe(df):
    print('Произвольные 10 строк таблицы:')
    if len(df) >= 10:
        display(df.sample(10))
    else:
        display(df)

    print('\nИнформация о таблице:')
    df.info()

    print('\nКоличество явных дубликатов в таблице:')
    print(df.duplicated().sum())

    print('\nПроцент пропусков в столбцах:')
    display(pd.DataFrame(
                round((df.isna().mean()*100),2), columns=['NaNs, %'])
                .sort_values(by='NaNs, %', ascending=False
            )
        .style.format('{:.2f}')
        .background_gradient('coolwarm')
    )
```

Обзор данных о пользователях

Откроем файл `users.csv` и выведем случайные 10 строк:

In [3]:
```
try:
    users = pd.read_csv('/datasets/users.csv')
except:
    users = pd.read_csv('users.csv')

data_observe(users)
```

```
Произвольные 10 строк таблицы:
```

	user_id	age	churn_date	city	first_name	last_name	reg_date	tariff
294	1294	66	NaN	Владивосток	Дебора	Асафьева	2018-02-20	ultra
318	1318	53	NaN	Курган	Марьяна	Анищенко	2018-03-22	ultra
495	1495	65	NaN	Иркутск	Авксентий	Фокин	2018-08-28	ultra
466	1466	37	NaN	Новосибирск	Рубен	Шевцов	2018-02-28	ultra
183	1183	20	NaN	Владивосток	Снежана	Озерова	2018-07-12	smart
32	1032	31	NaN	Ульяновск	Инна	Игнатьева	2018-01-04	smart
12	1012	38	NaN	Санкт-Петербург	Варлам	Соловьев	2018-03-28	smart
158	1158	45	NaN	Новокузнецк	Игорь	Андрейчук	2018-02-03	smart
387	1387	74	NaN	Краснодар	Габриель	Зуев	2018-12-21	smart
212	1212	21	NaN	Саранск	Руслан	Пономарев	2018-11-09	smart

```
Информация о таблице:
<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

Количество явных дубликатов в таблице:
0

Процент пропусков в столбцах:
```

	NaNs, %
churn_date	92.40
user_id	0.00
age	0.00
city	0.00
first_name	0.00
last_name	0.00
reg_date	0.00
tariff	0.00

Описание данных таблицы `users` (информация о пользователях):

- user_id — уникальный идентификатор пользователя (подразумевается целое число)
- first_name — имя пользователя
- last_name — фамилия пользователя
- age — возраст пользователя (годы, целое)
- 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
- tariff — название тарифного плана

В таблице объёмом 500 строк, 8 столбцов содержатся следующие типы данных: int64, object.

Столбцы поименованы корректно. Дубликатов строк нет.

Пропуски наблюдаются только в столбце `'churn_date'`, что означает, что подавляющее большинство представленных пользователей действующие. В остальных столбцах пропусков нет, Устранять пропуски не требуется.

Для экономии места поля `'user_id'` и `'age'` можно привести к типу int32.

Поле `'reg_date'` и непустые значения поля `'churn_date'` требуется привести к формату даты на этапе предобработки.

Обзор данных о звонках

Откроем файл `calls.csv` и выведем случайные 10 строк:

In [4]:
```
try:
    calls = pd.read_csv('/datasets/calls.csv')
except:
    calls = pd.read_csv('calls.csv')

data_observe(calls)
```

```
Произвольные 10 строк таблицы:
```

	id	call_date	duration	user_id
58734	1148_197	2018-11-21	4.40	1148
2811	1007_14	2018-12-01	0.00	1007
56075	1143_139	2018-05-13	12.68	1143
74179	1186_1	2018-09-23	4.89	1186
66668	1169_127	2018-07-12	6.15	1169
64748	1167_199	2018-11-11	10.55	1167
176680	1436_125	2018-10-03	0.00	1436
153406	1381_281	2018-01-19	0.00	1381
163251	1402_74	2018-04-27	15.69	1402
191019	1469_638	2018-07-29	2.06	1469

```
Информация о таблице:
<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

Количество явных дубликатов в таблице:
0

Процент пропусков в столбцах:
```

	NaNs, %
id	0.00
call_date	0.00
duration	0.00
user_id	0.00

Описание данных таблицы `calls` (информация о звонках):

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

- user_id — идентификатор пользователя, сделавшего звонок

В таблице объёмом 202607 строк, 4 столбца содержатся следующие типы данных: int64, float64, object.

Столбцы поименованы корректно. Пропуски данных и дубликаты строк не наблюдаются.

Идентификатор звонка имеет строковый тип.

С учётом замечаний в контексте исследования о порядке округления звонков следует привести поле `'duration'` к целому типу с округлением "вверх".

Для экономии места поле `'user_id'` можно привести к типу int32.

Поле `'call_date'` следует привести к формату даты.

Обзор данных о сообщениях

Откроем файл `messages.csv` и выведем случайные 10 строк:

In [5]:
```
try:
    messages = pd.read_csv('/datasets/messages.csv')
except:
    messages = pd.read_csv('messages.csv')

data_observe(messages)
```

```
Произвольные 10 строк таблицы:
```

	id	message_date	user_id
106033	1430_108	2018-10-05	1430
71402	1302_230	2018-08-18	1302
7129	1030_40	2018-10-22	1030
116747	1474_133	2018-12-26	1474
55547	1245_57	2018-11-22	1245
20594	1089_597	2018-10-02	1089
5062	1021_218	2018-10-18	1021
63883	1277_560	2018-12-18	1277
111883	1452_272	2018-10-24	1452
71924	1302_752	2018-04-09	1302

```
Информация о таблице:
<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

Количество явных дубликатов в таблице:
0

Процент пропусков в столбцах:
```

	NaNs, %
id	0.00
message_date	0.00
user_id	0.00

Описание данных таблицы `messages` (информация о сообщениях):

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

В таблице объёмом 123036 строк, 3 столбца содержатся следующие типы данных: int64, object.

Столбцы поименованы корректно. Пропуски данных и дубликаты строк не наблюдаются.

Идентификатор сообщения имеет строковый тип.

Для экономии места поле `'user_id'` можно привести к типу int32.

Поле `'message_date'` следует привести к формату даты.

Обзор данных об интернет-сессиях

Откроем файл `internet.csv` и выведем случайные 10 строк:

In [6]:
```
try:
    internet = pd.read_csv('/datasets/internet.csv')
except:
    internet = pd.read_csv('internet.csv')

data_observe(internet)
```

```
Произвольные 10 строк таблицы:
```

	Unnamed: 0	id	mb_used	session_date	user_id
20217	20217	1067_318	0.00	2018-06-06	1067
24859	24859	1086_66	694.68	2018-06-09	1086
101921	101921	1341_36	664.39	2018-11-16	1341
89330	89330	1302_130	0.00	2018-11-04	1302
146026	146026	1489_35	320.54	2018-10-22	1489
36795	36795	1127_145	53.51	2018-09-26	1127
49936	49936	1172_218	451.67	2018-07-18	1172
128340	128340	1429_498	0.00	2018-05-28	1429
74825	74825	1251_387	239.42	2018-03-27	1251
9099	9099	1032_179	372.34	2018-12-01	1032

```
Информация о таблице:
<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

Количество явных дубликатов в таблице:
0

Процент пропусков в столбцах:
```

	NaNs, %
Unnamed: 0	0.00
id	0.00
mb_used	0.00
session_date	0.00
user_id	0.00

Описание данных таблицы `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

В таблице объёмом 149396 строк, 5 столбцов содержатся следующие типы данных: int64, float64, object.

Столбцы поименованы корректно, за исключением `'Unnamed: 0'`. Однако, такого столбца нет в описании данных. Вероятно, в процессе извлечения или сохранения данных были получены побочные значения, равные индексу строки в таблице.

Представляется целесообразным удалить столбец `'Unnamed: 0'` и ***уведомить о возможной технологической ошибке инженеров по данным.***

Пропуски данных и дубликаты строк не наблюдаются.

Идентификатор сессии имеет строковый тип.

С учётом замечаний в контексте исследования о порядке округления интернет-сессий следует оставить поле `'mb_used'` в вещественном типе.

Для экономии места поле `'user_id'` можно привести к типу int32.

Поле `'session_date'` следует привести к формату даты.

Обзор данных о тарифах

Откроем файл `tariffs.csv` и выведем первые 5 строк:

In [7]:
```
try:
    tariffs = pd.read_csv('/datasets/tariffs.csv')
except:
    tariffs = pd.read_csv('tariffs.csv')

data_observe(tariffs)
```

```
Произвольные 10 строк таблицы:
```

	messages_included	mb_per_month_included	minutes_included	rub_monthly_fee	rub_per_gb	rub_per_message	rub_per_minute	tariff_name
0	50	15360	500	550	200	3	3	smart
1	1000	30720	3000	1950	150	1	1	ultra

```
Информация о таблице:
<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

Количество явных дубликатов в таблице:
0

Процент пропусков в столбцах:
```

	NaNs, %
messages_included	0.00
mb_per_month_included	0.00
minutes_included	0.00
rub_monthly_fee	0.00
rub_per_gb	0.00
rub_per_message	0.00
rub_per_minute	0.00
tariff_name	0.00

Описание данных таблицы `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)

В таблице объёмом 2 строки, 8 столбцов содержатся следующие типы данных: int64, object.

Столбцы поименованы корректно. Пропуски данных и дубликаты строк не наблюдаются.

Для экономии места целочисленные поля можно привести к типу int32.

Выводы

1. Представленные для анализа данные, на первый взгляд, качественные. Необоснованных (ошибочных) пропусков данных не наблюдается.
2. Данные сгруппированы таблицы, описывающие отдельно самих пользователей, тарифы и поведение пользователей "Мегалайна" - очень удобно для дальнейшего анализа.
3. Предварительно можно утверждать, что данных достаточно для проведения исследования, в том числе, для проверки статистических гипотез.
4. На этапе предобработки целесообразно выполнить следующие действия:
   - Приведение типов:
     - к `int32` - все целочисленные столбцы int64 всех таблиц, а также поле `'duration'` таблицы `'calls'` (с округлением "вверх");
     - к `DateTime` - все столбцы с датами всех таблиц, включая неполный столбец `'churn_date'` таблицы `'users'`.
   - Удаление столбца `'Unnamed: 0'` в таблице `internet`.

Замечание: уведомить инженеров по данным о возможной технологической ошибке (извлечение лишнего столбца в таблице `internet`).

Предобработка данных

Зададим ряд функций для преобразования типов данных, а также проверки значений на уникальность:

In [8]:
```
# Задание функции для преобразования типов набора столбцов
# ========================================================
# На вход подаются
#     df - имя таблицы
#     columns - список столбцов
#     datatype - тип данных
# В процессе работы функция выводит новый тип данных
# столбца после изменения, или сообщение об ошибке
def set_columns_types(df, columns, datatype):
    try:
        for col in columns:
            df[col] = df[col].astype(datatype)
            print(df[col].dtype)
    except:
        print(f'Один или несколько столбцов невозможно привести к типу {datatype}')
```

In [9]:
```
# Задание функции для преобразования даты в наборе столбцов
# ========================================================
```

```
# На вход подаются
#     df - имя таблицы
#     columns - список столбцов
#     fmt - формат даты
# В процессе работы функция выводит новый тип данных
# столбца после изменения, или сообщение об ошибке
def set_date_type(df, columns, fmt):
    try:
        for col in columns:
            df[col] = pd.to_datetime(df[col], format=fmt)
            print(df[col].dtype)
    except:
        print(f'Один или несколько столбцов невозможно привести к формату {fmt}')
```

In [10]:
```
# Задание функции для проверки значений в столбце на уникальность
# ============================================================
# На вход подаются
#     df - имя таблицы
#     column - столбец
# В процессе работы функция выводит сообщение о результате проверки
def is_unique(df, column):
    if df[column].nunique() == len(df):
        print('Все значения уникальны!')
    else:
        print('Присутствуют дубликаты!')
```

Предобработка данных о пользователях

Для экономии места поля `'user_id'` и `'age'` приведём к типу int32:

In [11]:
```
set_columns_types(users, ['user_id', 'age'], 'int32')
```

```
int32
int32
```

Приведём поле `'reg_date'` и непустые значения поля `'churn_date'` к формату даты:

In [12]:
```
set_date_type(users, ['reg_date', 'churn_date'], '%Y-%m-%d')
```

```
datetime64[ns]
datetime64[ns]
```

Проверим, действительно ли уникальные идентификаторы пользователей уникальны:

In [13]:
```
is_unique(users, 'user_id')
```

```
Все значения уникальны!
```

Отлично!

Оценм, в каком диапазоне лежат значения возраста клиентов:

In [14]:
```
users['age'].sort_values().unique()
```

Out[14]:
```
array([18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75], dtype=int32)
```

Клиентами "Мегалайна" являются люди разных возрастов: от 18 до 75 лет. Возможно их придётся поделить по возрасту на категории.

Рассмотрим географию пользователей на предмет неявных дубликатов:

In [15]:
```
users['city'].sort_values().unique()
```

Out[15]:
```
array(['Архангельск', 'Астрахань', 'Балашиха', 'Барнаул', 'Белгород',
       'Брянск', 'Владивосток', 'Владикавказ', 'Владимир', 'Волгоград',
       'Волжский', 'Вологда', 'Воронеж', 'Грозный', 'Екатеринбург',
       'Иваново', 'Ижевск', 'Иркутск', 'Казань', 'Калининград', 'Калуга',
       'Кемерово', 'Киров', 'Кострома', 'Краснодар', 'Красноярск',
       'Курган', 'Курск', 'Липецк', 'Магнитогорск', 'Махачкала', 'Москва',
       'Мурманск', 'Набережные Челны', 'Нижневартовск', 'Нижний Новгород',
       'Нижний Тагил', 'Новокузнецк', 'Новороссийск', 'Новосибирск',
       'Омск', 'Оренбург', 'Орёл', 'Пенза', 'Пермь', 'Петрозаводск',
       'Подольск', 'Ростов-на-Дону', 'Рязань', 'Самара',
       'Санкт-Петербург', 'Саранск', 'Саратов', 'Севастополь', 'Смоленск',
       'Сочи', 'Ставрополь', 'Стерлитамак', 'Сургут', 'Тамбов', 'Тверь',
       'Тольятти', 'Томск', 'Тула', 'Тюмень', 'Улан-Удэ', 'Ульяновск',
       'Уфа', 'Хабаровск', 'Химки', 'Чебоксары', 'Челябинск', 'Череповец',
       'Чита', 'Якутск', 'Ярославль'], dtype=object)
```

География - широкая. Неявных дубликатов - не выявлено.

Поищем неявные дубликаты в столбце `'tariff'`:

In [16]:
```
users['tariff'].sort_values().unique()
```

Out[16]:
```
array(['smart', 'ultra'], dtype=object)
```

Здесь также нет неявных дубликатов.

На всякий случай убедимся, что явные дубликаты также отсутствуют:

In [17]:
```
users.duplicated().sum()
```

Out[17]:
```
0
```

Для обеспечения возможности объединения таблиц `users` и `tariffs` по названию тарифа переименуем столбец `'tariff'` в `'tariff_name'`:

In [18]:
```
users = users.rename(
    columns={
        'tariff' : "tariff_name"
    }
)
users.head()
```

Out[18]:

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

Поскольку все `'user_id'` в таблице уникальны, то для удобства дальнейших исследований проиндексируем таблицу `users` данным столбцом:

In [19]:
```
# переиндексируем таблицу
users.index = users['user_id']

# проверяем результат
users.head()
```

Out[19]:

	user_id	age	churn_date	city	first_name	last_name	reg_date	tariff_name
user_id								
1000	1000	52	NaT	Краснодар	Рафаил	Верещагин	2018-05-25	ultra
1001	1001	41	NaT	Москва	Иван	Ежов	2018-11-01	smart
1002	1002	59	NaT	Стерлитамак	Евгений	Абрамович	2018-06-17	smart
1003	1003	23	NaT	Москва	Белла	Белякова	2018-08-17	ultra
1004	1004	68	NaT	Новокузнецк	Татьяна	Авдеенко	2018-05-14	ultra

Осталось проверить есть ли клиенты, которые были зарегистрированы менее 1 месяца (30 дней), поскольку есть вероятность, что они могли не пользоваться услугами вообще, или создать статистические выбросы в дальнейшем исследовании:

In [20]:
```
# добавим столбец разницы окончания и начала пользования услугами
users['days'] = users['churn_date'] - users['reg_date']

# выведем информацию по интересующим пользователям
users[users['days'] < '30 days']
```

Out[20]:

	user_id	age	churn_date	city	first_name	last_name	reg_date	tariff_name	days
user_id									
1062	1062	24	2018-12-09	Москва	Александр	Коршунов	2018-11-16	smart	23 days
1128	1128	51	2018-12-23	Волжский	Ксения	Агаева	2018-12-15	ultra	8 days
1371	1371	50	2018-12-30	Омск	Ириней	Трофимов	2018-12-25	smart	5 days

Мы видим, что 2 клиента в возрасте 50+ были зарегистрированы, как пользователи, 8 и 5 дней соответственно. В этой связи вероятным представляется отсутствие пользования услугами .данными клиентами.

Рассмотрим количество совершённых выбранными клиентами звонков, отправленных сообщений и сессий передачи данных:

In [21]:
```
user_ids = [1062, 1128, 1371]

for usr_id in user_ids:
    print(f'Количество звонков пользователя {usr_id}:', len(calls.query("user_id == @usr_id")))
    print(f'Количество сообщений пользователя {usr_id}:', len(messages.query("user_id == @usr_id")))
    print(f'Количество интернет-сессий пользователя {usr_id}:', len(internet.query("user_id == @usr_id")))
```

```
Количество звонков пользователя 1062: 83
Количество сообщений пользователя 1062: 26
Количество интернет-сессий пользователя 1062: 36
Количество звонков пользователя 1128: 0
Количество сообщений пользователя 1128: 0
Количество интернет-сессий пользователя 1128: 0
Количество звонков пользователя 1371: 0
Количество сообщений пользователя 1371: 0
Количество интернет-сессий пользователя 1371: 0
```

Очевидно, соответствующие пользователям с id 1128 и 1371 строки в таблице `users` можно удалить:

In [22]:
```
users.drop(labels=[1128, 1371], axis=0, inplace=True)
```

Клиент из Москвы пользовался тарифом `smart` более полумесяца и совершил достаточное количество действий за этот период. оставим информацию о нём для дальнейших исследований.

Предобработка таблицы `users` завершена.

Предобработка данных о звонках

Для экономии места поле `'user_id'` приведём к типу int32:

In [23]:
```
set_columns_types(calls, ['user_id'], 'int32')
```

```
int32
```

Приведём поле `'call_date'` к формату даты:

In [24]:
```
set_date_type(calls, ['call_date'], '%Y-%m-%d')
```

```
datetime64[ns]
```

Приведём к `int32` поле `'duration'` таблицы `'calls'` (с округлением "вверх"). Для этого применим к нему metod `ceil` библиотеки `math` :

In [25]:
```
# выведем первые 5 строк таблицы
calls.head()
```

Out[25]:

	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

In [26]:
```
# применим функцию mt.ceil к столбцу 'duration'
calls['duration'] = np.ceil(calls['duration'])
```

In [27]:
```
# проверим результат
calls.head()
```

Out[27]:

	id	call_date	duration	user_id
0	1000_0	2018-07-25	0.0	1000
1	1000_1	2018-08-17	0.0	1000
2	1000_2	2018-06-11	3.0	1000
3	1000_3	2018-09-21	14.0	1000
4	1000_4	2018-12-15	6.0	1000

Итак длительность вызовов округлена до целых минут в большую сторону.

Отметим, что в таблице присутствуют звонки нулевой длительности. Это не ошибка: нулями обозначены пропущенные звонки, поэтому их не нужно удалять.

Осталось проверить идентификаторы вызовов на уникальность:

In [28]:
```
is_unique(calls, 'id')
```

```
Все значения уникальны!
```

На всякий случай убедимся, что явные дубликаты также отсутствуют:

In [29]:
```
calls.duplicated().sum()
```

Out[29]:
```
0
```

Предобработка таблицы `'calls'` закончена.

Предобработка данных о сообщениях

Для экономии места поле `'user_id'` приведём к типу int32:

In [30]:
```
set_columns_types(messages, ['user_id'], 'int32')
```

```
int32
```

Приведём поле `'message_date'` к формату даты:

In [31]:
```
set_date_type(messages, ['message_date'], '%Y-%m-%d')
```

```
datetime64[ns]
```

Осталось проверить идентификаторы сообщений на уникальность:

In [32]:
```
is_unique(messages, 'id')
```

```
Все значения уникальны!
```

На всякий случай убедимся, что явные дубликаты также отсутствуют:

In [33]:
```
messages.duplicated().sum()
```

Out[33]:
```
0
```

Предобработка таблицы `'messages'` закончена.

Предобработка данных об интернет-сессиях

На этапе обзора мы признали целесообразным удаление столбца `'Unnamed: 0'` в таблице `internet`. Данный столбец не несёт в себе осмысленной для исследования информации, более того - не описан в исходных данных. Скорее всего, он является плодом технологической ошибки извлечения данных.

Удалим его:

In [34]:
```
internet.drop('Unnamed: 0', axis=1, inplace=True)
internet.head(2)
```

Out[34]:

	id	mb_used	session_date	user_id
0	1000_0	112.95	2018-11-25	1000
1	1000_1	1052.81	2018-09-07	1000

Лишний столбец удалён.

Для экономии места поле `'user_id'` приведём к типу int32:

In [35]:
```
set_columns_types(internet, ['user_id'], 'int32')
```

```
int32
```

Приведём поле `'message_date'` к формату даты:

In [36]:
```
set_date_type(internet, ['session_date'], '%Y-%m-%d')
```

```
datetime64[ns]
```

Осталось проверить идентификаторы сессий на уникальность:

In [37]:
```
is_unique(internet, 'id')
```

```
Все значения уникальны!
```

На всякий случай убедимся, что явные дубликаты также отсутствуют:

In [38]:
```
internet.duplicated().sum()
```

Out[38]:
```
0
```

Предобработка таблицы `'internet'` закончена.

Предобработка данных о тарифах

Для экономии места приведём целочисленные поля к типу int32:

In [39]:
```
# выберем названия числовых столбцов таблицы tariffs (все целочисленные)
cols = tariffs.select_dtypes(include=np.number).columns

# приведём их к типу int32
set_columns_types(tariffs, cols, 'int32')
```

```
int32
int32
int32
int32
int32
int32
int32
```

Нетрудно видеть, что в тарифах перерасход оплачивается за гигабайт, а включённый трафик измеряется в мегабайтах. Поэтому для обеспечения единства измерений, с учётом замечания о тарификации в контексте исследования, переведём включённые мегабайты в гигабайты:

In [40]:
```
# переименуем столбец включённого трафика
tariffs = tariffs.rename(
        columns={
            'mb_per_month_included' : 'gb_per_month_included'
        }
    )

# заменим мегабайты на гигабайты
tariffs['gb_per_month_included'] = tariffs['gb_per_month_included'].apply(lambda x : x // 1024)

# проверим результат
tariffs.head()
```

Out[40]:

	messages_included	gb_per_month_included	minutes_included	rub_monthly_fee	rub_per_gb	rub_per_message	rub_per_minute	tariff_name
0	50	15	500	550	200	3	3	smart
1	1000	30	3000	1950	150	1	1	ultra

На этом предобработка данных таблицы `'tariffs'` окончена.

Выводы

1. В ходе предварительной обработки данных установлено, что в целом данные очень качественные:
   - отсутствуют явные и неявные дубликаты;
   - отсутствуют необоснованные пропуски данных.

1. Для экономии памяти и ускорения обработки, а также в силу ограниченной природы целочисленных значений в представленных таблицах принято решение привести их к 32-битному целому типу `int32`.

1. Значения дат приведены к типу `DateTime`.

1. Столбец неясного происхождения `'Unnamed: 0'` удалён из таблицы `nternet`.

1. Длительность звонка `'duration'` в таблице `'calls'` приведена к целому типу с округлением "верх".

1. Для облегчения дальнейших действий с данными, а также для обеспечения единства измерений произведены:
   - переименование столбцов `'tariff'` в таблице `users` а также `'mb_per_month_included'` в таблице `tariffs`;
   - перевод мегабайт в гигабайты в таблице `tariffs`.

1. Из таблицы `users` обоснованно удалены 2 клиента, вообще не использовавших услуги оператора.

Обогащение данных

Добавление месяца пользования услугами

Добавим в таблицы `calls`, `messages`, `internet` столбтец с первым днём месяца совершения действия. Заодно учтём и год:

In [41]:
```
# месяцы звонков
# calls['month'] = calls['call_date'].dt.month
calls['month'] = calls['call_date'].astype('datetime64[M]')

# месяцы сообщений
# messages['month'] = messages['message_date'].dt.month
messages['month'] = messages['message_date'].astype('datetime64[M]')

# месяцы пользования интернетом
```

```
# internet['month'] = internet['session_date'].dt.month
internet['month'] = internet['session_date'].astype('datetime64[M]')
```

In [42]:
```
# проверим результат
calls.head(2)
```

Out[42]:

	id	call_date	duration	user_id	month
0	1000_0	2018-07-25	0.0	1000	2018-07-01
1	1000_1	2018-08-17	0.0	1000	2018-08-01

In [43]:
```
# проверим результат
messages.head(2)
```

Out[43]:

	id	message_date	user_id	month
0	1000_0	2018-06-27	1000	2018-06-01
1	1000_1	2018-10-08	1000	2018-10-01

In [44]:
```
# проверим результат
internet.head(2)
```

Out[44]:

	id	mb_used	session_date	user_id	month
0	1000_0	112.95	2018-11-25	1000	2018-11-01
1	1000_1	1052.81	2018-09-07	1000	2018-09-01

Количественный расчёт пользования услугами

Посчитаем для каждого пользователя количество сделанных звонков и израсходованных минут разговора по месяцам:

In [45]:
```
# создадим сводную таблицу
user_calls = calls.pivot_table(
    index = ['user_id', 'month'],
    values = 'duration',
    aggfunc = ['count', 'sum']
)

# переименуем столбцы
user_calls.columns = ['calls_facts', 'calls_total']

# выведем результат
user_calls.head()
```

Out[45]:

		calls_facts	calls_total
user_id	month		
1000	2018-05-01	22	159.0
	2018-06-01	43	172.0
	2018-07-01	47	340.0
	2018-08-01	52	408.0
	2018-09-01	58	466.0

Посчитаем для каждого пользователя количество отправленных сообщений по месяцам (пропуски оставим, поскольку отсутствие сообщений в указанном месяце равносильно неиспользованию сервиса сообщений):

In [46]:
```
# создадим сводную таблицу
user_messages = messages.pivot_table(
    index = ['user_id', 'month'],
    values = 'id',
    aggfunc = ['count']
)

# переименуем столбцы
user_messages.columns = ['messages_facts']

# выведем результат
user_messages.head()
```

Out[46]:

		messages_facts
user_id	month	
1000	2018-05-01	22
	2018-06-01	60
	2018-07-01	75
	2018-08-01	81
	2018-09-01	57

Посчитаем для каждого пользователя объём израсходованного интернет-трафика по месяцам (пропуски оставим, поскольку отсутствие пользования интернетом в указанном месяце равносильно неиспользованию интернета в данном месяце):

In [47]:
```
# создадим сводную таблицу
user_internet = internet.pivot_table(
    index = ['user_id', 'month'],
    values = 'mb_used',
    aggfunc = ['count', 'sum']
)

# переименуем столбцы
user_internet.columns = ['internet_facts', 'internet_total']

# округлим ежемесячное потребление в гигабайтах согласно требуемым правилам
user_internet['internet_total'] = np.ceil(
```

```
    user_internet['internet_total'].apply(lambda x : x / 1024)
)

# выведем результат
user_internet.head()
```

Out[47]:

user_id	month	internet_facts	internet_total
1000	2018-05-01	5	3.0
	2018-06-01	49	23.0
	2018-07-01	29	14.0
	2018-08-01	29	14.0
	2018-09-01	27	15.0

Соберём полученные сводные таблицы в одну и проверим, все ли данные попали в итоговую таблицу:

In [48]:
```
# Соберём расчёты в одну таблицу и добавим данные по тарифам
user_total = (
    user_calls
    .join(user_messages,
          on=['user_id', 'month'],
          how='outer')
    .join(user_internet,
          on=['user_id', 'month'],
          how='outer')
    .join(users[['city', 'tariff_name']],
          on='user_id',
          how='left')
)

# проверим, все ли коиенты пользовались всеми услугами каждый месяц
user_total.isna().sum()
```

Out[48]:
```
calls_facts        40
calls_total        40
messages_facts    497
internet_facts     11
internet_total     11
city                0
tariff_name         0
dtype: int64
```

В сводной таблице часть значений пропущена. Это соответствует ситуации, когда отдельные клиенты не пользовались *всеми* услугами оператора каждый месяц.
Логично заполнить пропуски нулями и использовать их в дальнейшем при проверке статистических гипотез:

In [49]:
```
# заполним нулями пропуски пользования отдельными услугами в конкретные месяцы
user_total.fillna(value=0, axis=0, inplace=True)
user_total.head()
```

Out[49]:

user_id	month	calls_facts	calls_total	messages_facts	internet_facts	internet_total	city	tariff_name
1000	2018-05-01	22.0	159.0	22.0	5.0	3.0	Краснодар	ultra
	2018-06-01	43.0	172.0	60.0	49.0	23.0	Краснодар	ultra
	2018-07-01	47.0	340.0	75.0	29.0	14.0	Краснодар	ultra
	2018-08-01	52.0	408.0	81.0	29.0	14.0	Краснодар	ultra
	2018-09-01	58.0	466.0	57.0	27.0	15.0	Краснодар	ultra

Проверим, все ли данные вошли в итоговую таблицу:

In [50]:
```
# проверим данные по пользователям
if len(user_total.index.get_level_values(0).unique()) == len(users):
    print('В сводной таблице данные по всем пользователям.')
else:
    print('ОШИБКА! В сводной таблице пропущены данные о пользователях!')
```

```
В сводной таблице данные по всем пользователям.
```

In [51]:
```
# расчёт суммы по сводной таблице
user_total[['calls_facts', 'calls_total', 'messages_facts', 'internet_facts', 'internet_total']].sum()
```

Out[51]:
```
calls_facts        202607.0
calls_total       1450301.0
messages_facts     123036.0
internet_facts     149396.0
internet_total      55599.0
dtype: float64
```

In [52]:
```
# расчёт суммы по отдельным таблицам
print('original calls facts:', len(calls))
print('original salls total:', calls['duration'].sum())
print('original messages facts:', len(messages))
print('original internet facts:', len(internet))
print('original internet total:', internet['mb_used'].sum() / 1024)
```

```
original calls facts: 202607
original salls total: 1450301.0
original messages facts: 123036
original internet facts: 149396
original internet total: 54009.05051757812
```

Поскольку мы округляли гигабайты "к потолку", можно считать, что все данные по сумме трафика учтены в сводной таблице, равно как остальные данные по услугам.

Рассчёт затрат пользователей за услуги по месяцам

Добавим в сводную таблицу информацию о тарифах:

In [53]:
```
# добавим в таблицу user_total информацию о тарифах
```

```
user_total = (
    user_total
    .merge(tariffs,
           on='tariff_name',
           how='left')
)
# выведем результат
user_total.sample(5).T
```

Out[53]:

	1584	462	1587	2768	3001
calls_facts	59.0	83.0	78.0	72.0	75.0
calls_total	466.0	617.0	497.0	503.0	557.0
messages_facts	0.0	19.0	0.0	88.0	83.0
internet_facts	44.0	65.0	41.0	42.0	27.0
internet_total	20.0	22.0	18.0	14.0	14.0
city	Москва	Уфа	Москва	Подольск	Вологда
tariff_name	ultra	smart	ultra	smart	ultra
messages_included	1000	50	1000	50	1000
gb_per_month_included	30	15	30	15	30
minutes_included	3000	500	3000	500	3000
rub_monthly_fee	1950	550	1950	550	1950
rub_per_gb	150	200	150	200	150
rub_per_message	1	3	1	3	1
rub_per_minute	1	3	1	3	1

Рассчитаем для каждого пользователя выручку "Мегалайна" по месяцам.

Добавим в таблицу столбец суммарных помесячных затрат пользователей, после чего удалим информацию о тарифах:

In [54]:
```
# расчёт суммарных затрат
user_total['spendings'] = (
    # 1. вычислим стоимость превышения минут (отсекаем отрицательные значения нулём)
    (user_total['calls_total'] - user_total['minutes_included']).clip(lower=0) * user_total['rub_per_minute'] +
    # 2. вычислим стоимость превышения сообщений
    (user_total['messages_facts'] - user_total['messages_included']).clip(lower=0) * user_total['rub_per_message'] +
    # 3. вычислим стоимость превышения трафика
    (user_total['internet_total'] - user_total['gb_per_month_included']).clip(lower=0) * user_total['rub_per_gb'] +
    # 4. добавим абонплату
    user_total['rub_monthly_fee']
)

# удаление информации о тарифах
user_total.drop(
    ['messages_included', 'gb_per_month_included', 'minutes_included',
     'rub_monthly_fee', 'rub_per_gb', 'rub_per_message', 'rub_per_minute'],
    axis=1,
    inplace=True
)

# вывод результата
user_total.sample(5)
```

Out[54]:

	calls_facts	calls_total	messages_facts	internet_facts	internet_total	city	tariff_name	spendings
759	36.0	310.0	0.0	21.0	13.0	Краснодар	ultra	1950.0
1357	63.0	567.0	17.0	60.0	17.0	Рязань	smart	1151.0
3137	81.0	510.0	13.0	71.0	17.0	Санкт-Петербург	smart	980.0
248	77.0	541.0	137.0	43.0	21.0	Ярославль	ultra	1950.0
340	91.0	595.0	62.0	63.0	19.0	Омск	smart	1671.0

Выводы

В ходе обогащения данных мы выполнили следующие действия:

1. Извлекли из даты месяц пользования для каждого факта пользования услугами.
2. Рассчитали количественные объёмы пользования услугами для различных клиентов по месяцам.
3. Рассчитали общие затраты всех пользователей по месяцам.

Теперь таблица `user_total` содержит все статистические данные по всем 498 клиентам за все месяцы пользования услугами. Перейдём к анализу данных и описанию поведения клиентов "Мегалайна".

Анализ данных: описание поведения клиентов оператора

Определим, сколько минут разговора, сколько сообщений и какой объём интернет-трафика требуется пользователям каждого тарифа в месяц (посчитаем среднее количество, дисперсию и стандартное отклонение).

Оценим размеры выборок данных по тарифам для анализа:

In [55]:
```
len(user_total[user_total['tariff_name'] == 'ultra'])
```

Out[55]:
```
985
```

In [56]:
```
len(user_total[user_total['tariff_name'] == 'smart'])
```

Out[56]:
```
2229
```

Итак, для каждого параметра мы получили по 2 выборки длиной 985 и 2229 испытаний соответственно.

Зададим функции для автоматизации построения гистограмм и диаграмм размаха для набора из 3 столбцов датафрейма:

In [57]:
```
# функция вывода гистограмм для набора столбцов датафрейма
# ==========================================================
# на вход подаются:
#     df - датафрейм
#     columns - список названий колонок
def histplt(df, columns):
    fig, axes = plt.subplots(2, 2, figsize=(16,16))

    axe = axes.ravel()

    for i, column in enumerate(columns):
        df[column].hist(ax=axe[i], bins=10)
        axe[i].set_xlabel(column)
        axe[i].set_ylabel('Частота')
        axe[i].set_title('Распределение {}'.format(column))
```

In [58]:
```
# функция вывода диаграмм размаха для набора столбцов датафрейма
# ==========================================================
# на вход подаются:
#     df - датафрейм
#     columns - список названий колонок
def boxplt(df, columns):
    fig, axes = plt.subplots(2, 2, figsize=(16,16))

    axe = axes.ravel()

    for i, col in enumerate(columns):
        df.boxplot(ax=axe[i], column=col)
        axe[i].set_ylabel('Количество')
        axe[i].set_title('Диаграмма размаха {}'.format(col))
```

Рассчитаем среднее количество, дисперсию, стандартное отклонение для тарифов `smart` и `ultra`:

In [59]:
```
# вычисление среднего, стандартного отклонения и квартилей
user_total.pivot_table(
    index = 'tariff_name',
    values = ['calls_total', 'messages_facts', 'internet_total', 'spendings'],
    aggfunc= ['mean', 'var','std', 'max']).T
```

Out[59]:

	tariff_name	smart	ultra
mean	calls_total	417.934948	526.623350
	internet_total	16.328847	19.494416
	messages_facts	33.384029	49.363452
	spendings	1289.973531	2070.152284
var	calls_total	36219.315784	100873.633397
	internet_total	33.028705	97.091686
	messages_facts	796.812958	2285.266143
	spendings	669785.708006	141516.745079
std	calls_total	190.313730	317.606098
	internet_total	5.747061	9.853511
	messages_facts	28.227876	47.804457
	spendings	818.404367	376.187114
max	calls_total	1435.000000	1673.000000
	internet_total	38.000000	49.000000
	messages_facts	143.000000	224.000000
	spendings	6770.000000	4800.000000

Итак, пользователи тарифа `ultra` в среднем используют:

- 527 минут разговоров (17,6% от 3000 включённых);
- 49 сообщений (4,9% от 1000 включённых);
- 20 гигабайт интернета (66,7% от включённых 30).

Таким образом, с одной стороны, пользователи `ultra` в среднем переплачивают абонентску плату за неиспользуемые услуги. Услуга коротких сообщений вообще не востребована в таком количестве.

Если смотреть на максимальное потребление, то оно составляет:

- 1673 минуты разговоров (55,8% от 3000 включённых);
- 224 сообщений (22,4% от 1000 включённых);
- 49 гигабайт интернета (163.3% от включённых).

Действительно, что касается звонков и сообщений, пользователи оплачивают ненужные им в таком количестве услуги, а по интернету - даже, порой существенно переплачивают (при стоимости 150 руб/Гб - до 2850 рублей в месяц).

Большие значения стандартного отклонения и дисперсии свидетельствуют о большом разбросе в потреблении услуг. Особенно выражено это на примере разговоров.

Пользователи тарифа `smart` в среднем используют:

- 418 минут разговоров (83,6% от 500 включённых);
- 33 сообщения (66% от 50 включённых);
- 17 гигабайт интернета (113,3% от включённых 15).

Таким образом, с одной стороны, пользователи `smart` в среднем переплачивают только за услугу передачи данных, используя голосовую связь и сообщения в пределах тарифа.

Если смотреть на максимальное потребление, то оно составит:

- 1435 минуты разговоров (287% от 500 включённых);
- 143 сообщения (286% от 50 включённых);
- 38 гигабайт интернета (253.3% от 15 включённых).

Очевидно, максимальное потребление в среднем почти втрое превышает включённые лимиты. В рублях данное превышение выражается в 2805 рублей за минуты, 279 рублей за сообщения, 4600 рублей за трафик.

Отметим, что максимальное потребление по разным услугам, скорее всего, наблюдается у разных клиентов, однако в целом можно констатировать, что некоторое количество клиентов "выросло" из тарифа `smart` и приносит повышенный доход Мегалайну.

В целом, дисперсия по всем услугам тарифа `smart` ниже, чем для тарифа `ultra`. Это может свидетельствовать о том, что пользователи `smart`, обладая более скромными лимитами, стараются в большинстве своём в них вписаться, в то время как пользователи `ultra` чувствуют большую свободу в выборе объёмов месячного потребления.

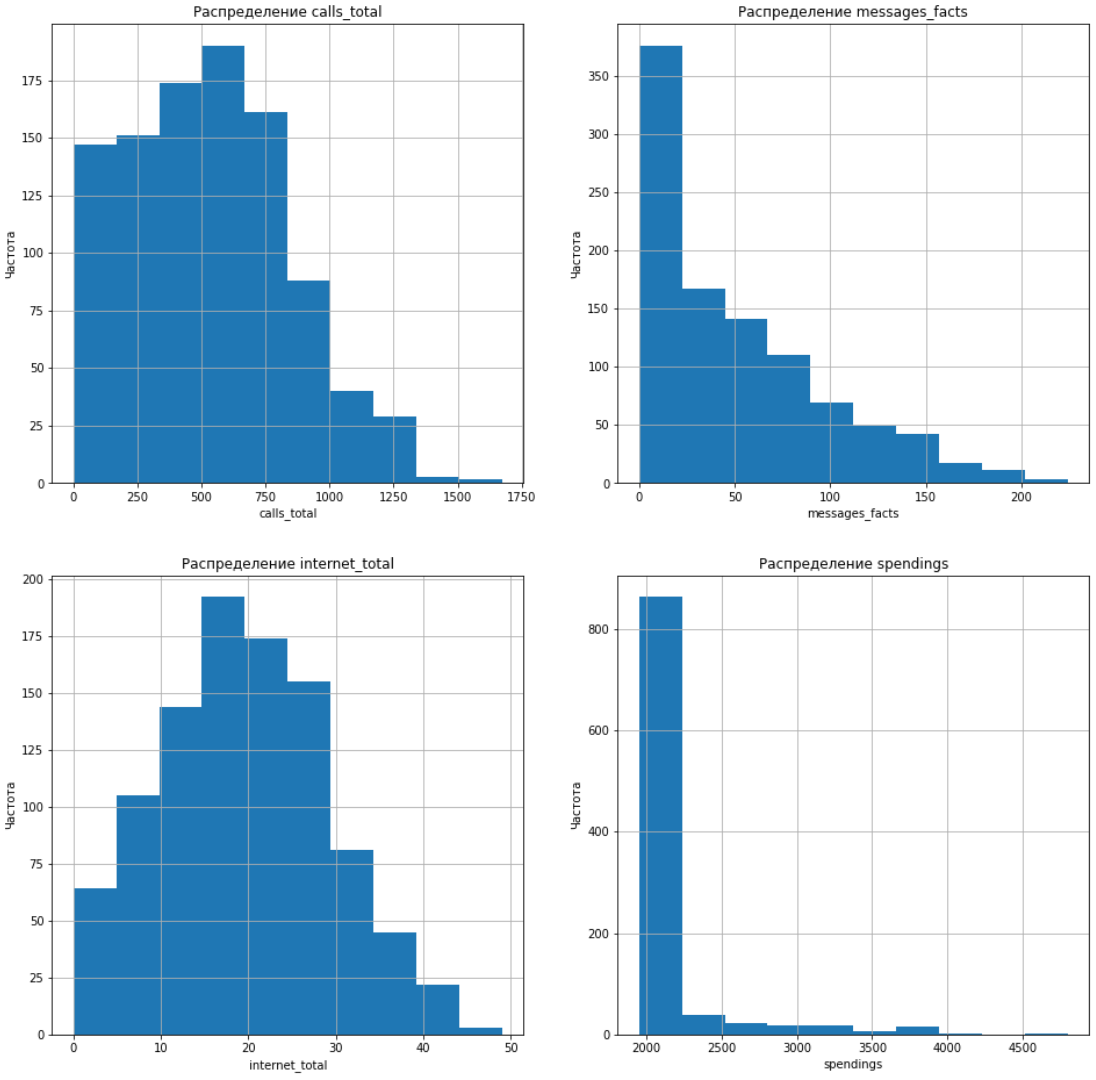
Если анализировать среднюю выручку, то она составит:

- 1290 рублей для пользователей `smart` (примерно в 2.3 раза превышает абонплату в 550 руб.);
- 2070 рублей для пользователей `ultra` (примерно на 6% превышает абонплату в 1950 руб.).

Однако, поскольку дисперсия затрат примерно в 2.2 раза выше для тарифа `smart`, можно предположить, что поведение пользователей данного тарифа более нестабильно.

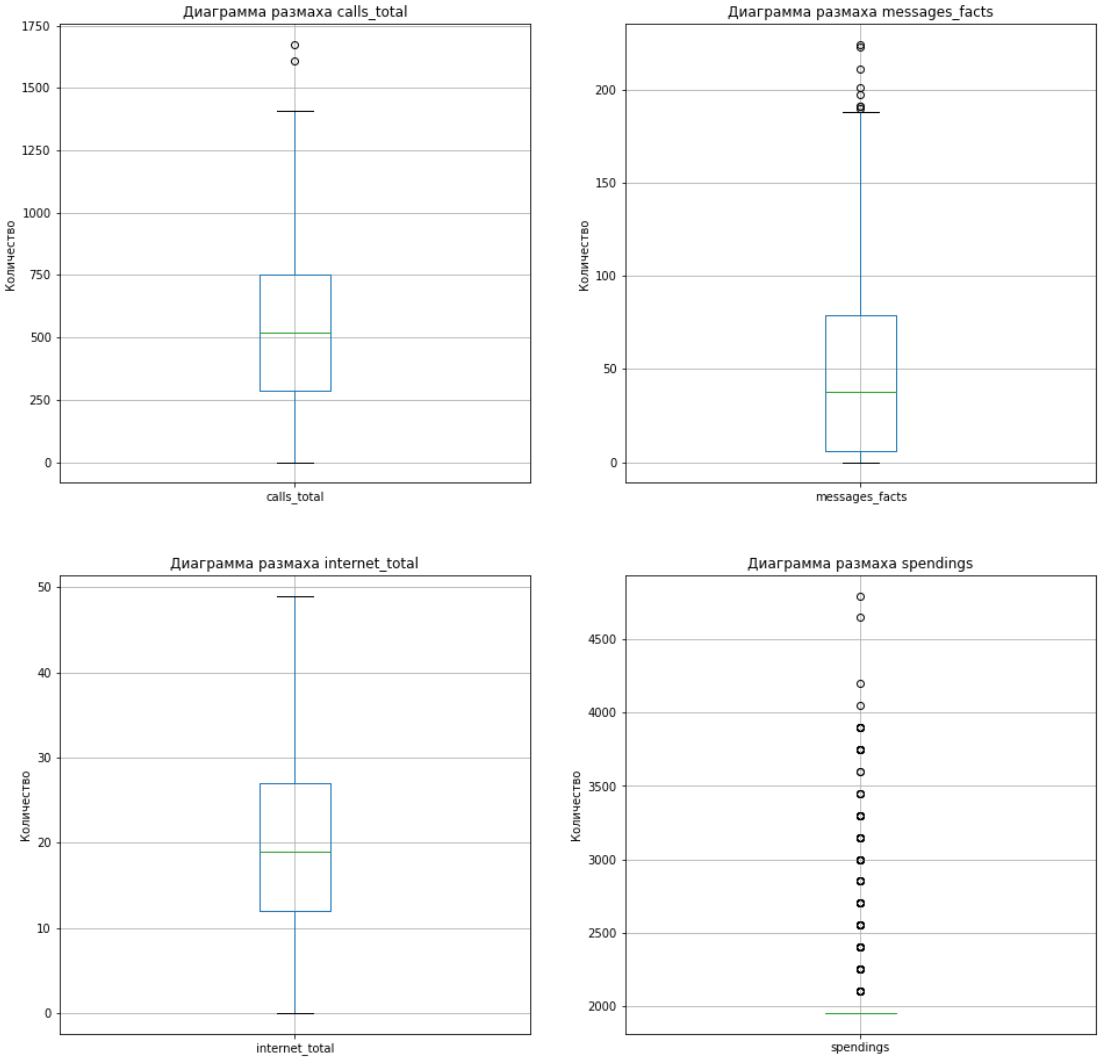
Построим гистограммы для минут, сообщений и трафика в тарифе `ultra`:

In [60]:
```
histplt(user_total.query('tariff_name == "ultra"'),
        ['calls_total', 'messages_facts', 'internet_total', 'spendings'])
```



Построим диаграммы размаха для минут, сообщений и трафика в тарифе `ultra`:

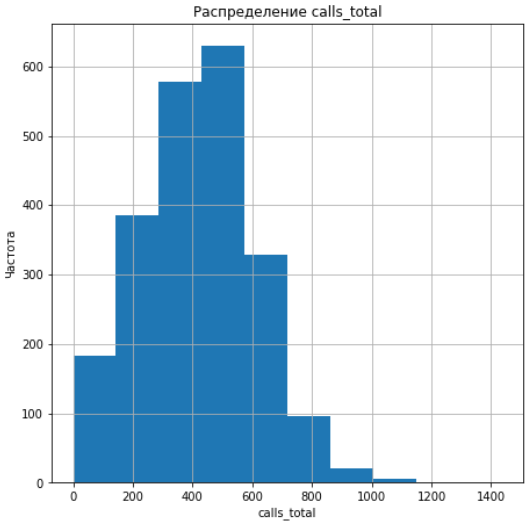
In [61]:
```
boxplt(user_total.query('tariff_name == "ultra"'),
       ['calls_total', 'messages_facts', 'internet_total', 'spendings'])
```

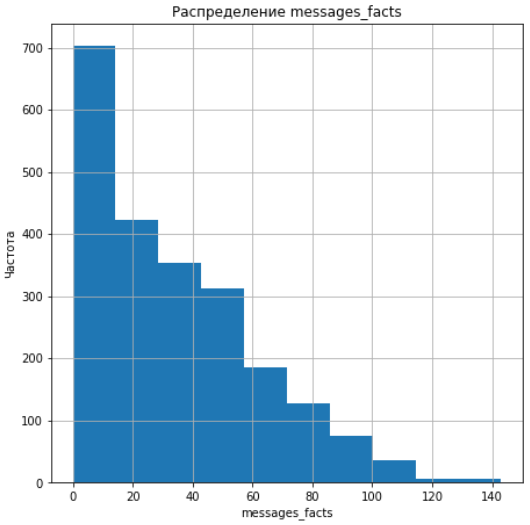


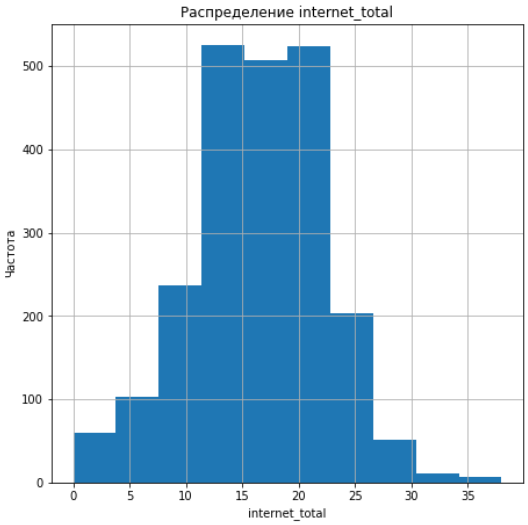
Диаграммы и гистограммы подтверждают сделанные выше заключения - распределения минут и сообщений в тарифе `ultra` существенно смещены влево - в сторону недобора использования услуг. Распределение потребления трафика ближе к нормальному со средним в районе 18-19 Гбайт.

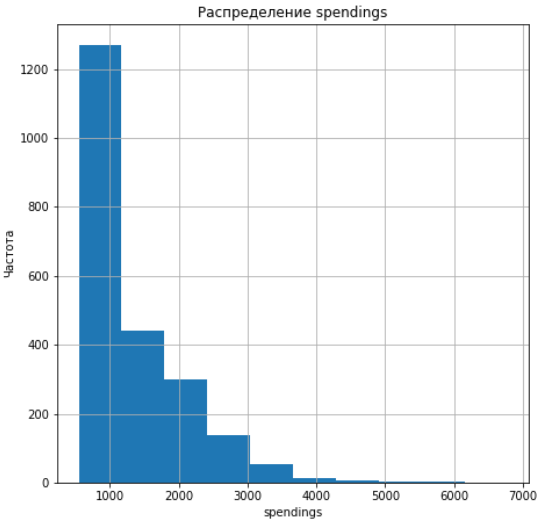
Построим гистограммы для минут, сообщений и трафика в тарифе `smart`:

In [62]:
```
histplt(user_total.query('tariff_name == "smart"'),
        ['calls_total', 'messages_facts', 'internet_total', 'spendings'])
```



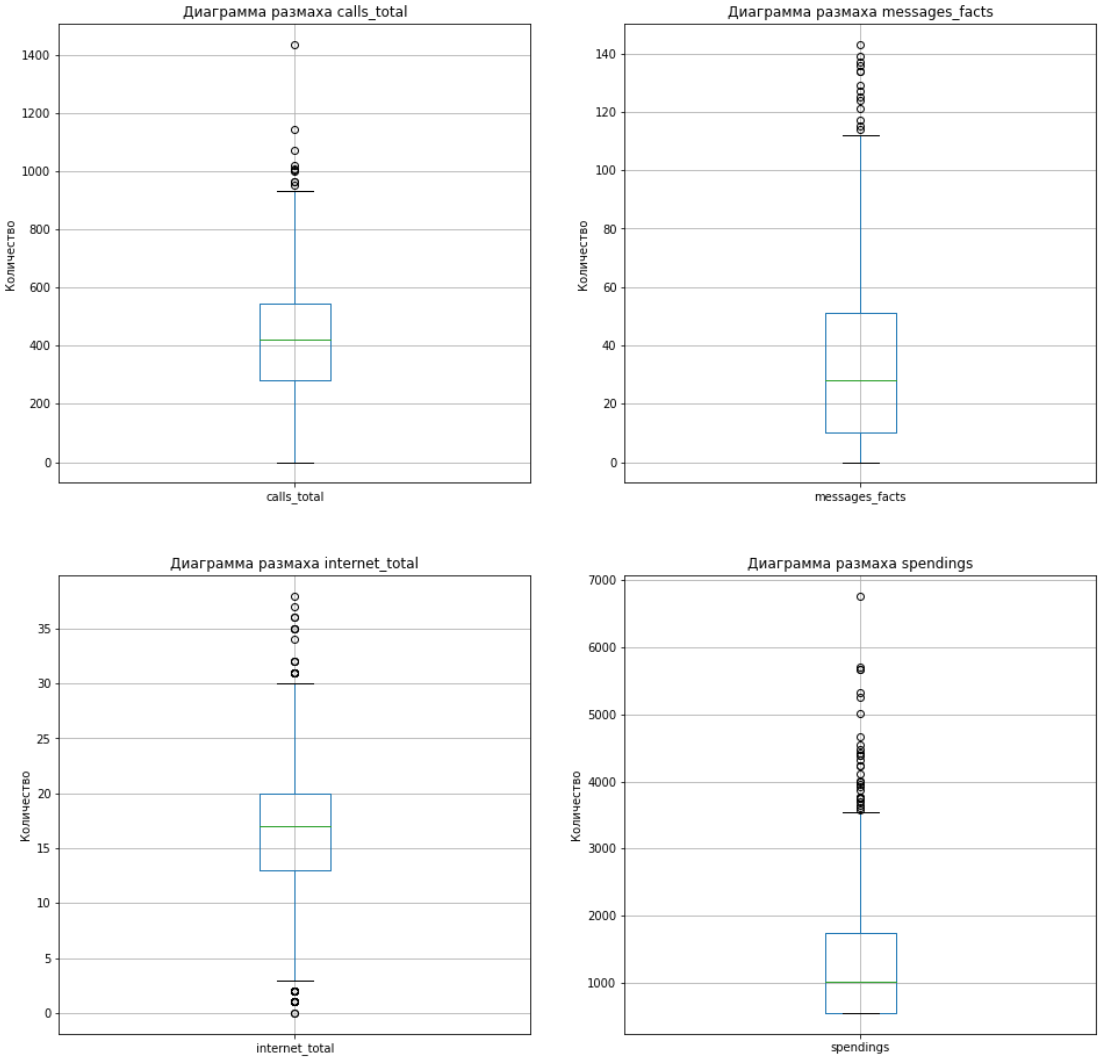






Построим диаграммы размаха для минут, сообщений и трафика в тарифе `smart`:

In [63]:
```
boxplt(user_total.query('tariff_name == "smart"'),
       ['calls_total', 'messages_facts', 'internet_total', 'spendings'])
```



Диаграммы и гистограммы подтверждают сделанные выше заключения - распределения минут и сообщений в тарифе `smart` имеют форму нормального и пуассоновского распределения соответственно, а распределение потребления трафика - существенно смещено вправо, в сторону перерасхода (причём таких клиентов немало).

Также следует отметить, что диаграммы размаха фиксируют существенно большее количество выбросов для клиентов данного тарифа, что подтверждает мысль о нестабильности их поведения.

Для ответа на главный вопрос исследования можно рассчитать среднюю общую выручку за 2018 год по каждому из тарифов:

In [64]:
```
print('Средняя общая выручка за 2018 год по тарифу smart:',
      round(user_total[user_total['tariff_name'] == 'smart']['spendings'].mean() *
      len(user_total[user_total['tariff_name'] == 'smart'])))
print('Средняя общая выручка за 2018 год по тарифу ultra:',
      round(user_total[user_total['tariff_name'] == 'ultra']['spendings'].mean() *
      len(user_total[user_total['tariff_name'] == 'ultra'])))
```

```
Средняя общая выручка за 2018 год по тарифу smart: 2875351
Средняя общая выручка за 2018 год по тарифу ultra: 2039100
```

Очевидно, пользователи тарифа `smart` в среднем принесли Мегалайну больше денег в 2018 году. Следовательно, тариф `smart` более выгоден провайдеру.

Выводы

Клиенты на тарифах `ultra` и `smart` ведут себя по-разному:

- большая часть клиентов `ultra` переплачивают абонплату за неиспользуемые звонки и сообщения, в то время как клиенты `smart` стараются использовать данные услуги в рамках тарифа;
- в то же время, 30 включённых Гбайт трафика клиентам `ultra` хватает чаще, нежели включённых 15 Гбайт - кдиентам `smart` ;
- максимальный перерасход трафика у клиентов `smart` выше - 23 Гбайт против 19 Гбайт в тарифе `ultra` .

Таким образом, можно заключить, что в тарифе `ultra` основной источник дохода Мегалайна в высокой абонплате за большие пакеты включённых, но не используемых абонентами услуг, а в тарифе `smart` - в высокой стоимости дополнительного трафика при ограниченном пакете включённых Гбайт.

За счёт большего количества абонентов тарифа `smart` , а также за счёт их склонности чаще перерасходовать более дорогой интернет-трафик, можно утверждать, что данный тариф более выгоден для Мегалайна по данным за 2018 год.

Проверка гипотез

В рамках исследования необходимо проверить 2 гипотезы:

1. Средняя выручка пользователей тарифов «Ультра» и «Смарт» различаются.
2. Средняя выручка пользователей из Москвы отличается от выручки пользователей из других регионов.

Проверка гипотезы №1

Необходимо проверить гипотезу "Средняя выручка пользователей тарифов `ultra` и `smart` различается".

Сформулируем соответствующие статистические гипотезы:

- H0 - средняя выручка пользователей `ultra` равна средней выручке пользователей `smart`.
- H1 - средняя выручка пользователей `ultra` не равна средней выручке пользователей `smart`.

Это статистическая гипотеза о равенстве средних двух генеральных совокупностей. Мы имеем выборки разного размера, вероятно, с разными дисперсиями. Поэтому при проверке гипотезы используем параметр `equal_var=False`:

In [65]:
```
# зададим пороговое значение
alpha = .05 # если p-value окажется меньше него - отвергнем гипотезу

# выполним статистический тест на равенство средних
# двух генеральных совокупностей
results = st.ttest_ind(
    user_total.query('tariff_name == "ultra"')['spendings'],
    user_total.query('tariff_name == "smart"')['spendings'],
    equal_var=False
)

print('статистика разности:', results.statistic)
print('p-значение:', results.pvalue)

if results.pvalue < alpha:
    print("Отвергаем H0")
else:
    print("Не получилось отвергнуть H0")
```

```
статистика разности: 37.019021231454644
p-значение: 4.2606313931076085e-250
Отвергаем H0
```

На имеющихся данных на 5% уровне значимости имеются основания отвергнуть гипотезу H0 в пользу альтернативы H1. Средняя выручка пользователей тарифов ultra и smart различается.

Проверка гипотезы №2

Необходимо проверить гипотезу "Средняя выручка пользователей из Москвы отличается от выручки пользователей из других регионов".

Сформулируем соответствующие статистические гипотезы:

- H0 - средняя выручка пользователей из Москвы равна средней выручке пользователей из других регионов.
- H1 - средняя выручка пользователей из Москвы не равна средней выручке пользователей из других регионов.

Для проверки гипотезы разделим выборки по полю `city`:

In [66]:
```
# оценим размеры выборок
print(len(user_total.query('city == "Москва"')))
print(len(user_total.query('~(city == "Москва")')))
```

```
611
2603
```

Итак, мы имеем статистическую гипотезу о равенстве средних двух генеральных совокупностей. Мы имеем выборки разного размера, вероятно, с разными дисперсиями. Поэтому при проверке гипотезы используем параметр `equal_var=False`:

In [67]:
```
# зададим пороговое значение
alpha = .05 # если p-value окажется меньше него - отвергнем гипотезу

# выполним статистический тест на равенство средних
# двух генеральных совокупностей
results = st.ttest_ind(
    user_total.query('city == "Москва"')['spendings'],
    user_total.query('~(city == "Москва")')['spendings'],
    equal_var=False
)

print('статистика разности:', results.statistic)
print('p-значение:', results.pvalue)

if results.pvalue < alpha:
    print("Отвергаем H0")
else:
    print("Не получилось отвергнуть H0")
```

```
статистика разности: 0.6347555055229303
p-значение: 0.5257376663729298
Не получилось отвергнуть H0
```

На имеющихся данных на 5% уровне значимости не имеется оснований отвергнуть гипотезу H0 в пользу альтернативы H1. Средние выручки пользователей из Москвы и других городов близки друг к другу.

Выводы

Мы проверили две статистические гипотезы:

1. Гипотеза о равестве средней выручки от пользователей тарифов `smart` и `ultra` была отвергнута. Таким образом, можно говорить о том что средние выручки от пользователей разных тарифов различаются.

1. Гипотеза о равестве средней выручки от пользователей г. Москва и остальных городов не была отвергнута. Следовательно нельзя говорить о том, средние выручки от пользователей из разных городов различаются.

Общий вывод

Проведённое исследование включает в себя:

- обзор данных;
- предобработку данных;
- обогащение данных;
- анализ поведения пользователей разных тарифов;
- проверку статистических гипотез.

Основные выводы по каждому этапу приведены в соответствующем разделе.

Основываясь на них, сформулируем ответ на основной вопрос исследования: какой тариф лучше (выгоднее Мегалайну)?

Проверка статистических гипотез показала, что средняя выручка оператора от пользователей тарифных планов `ultra` и `smart` различается, причём существенно.

Анализ поведения пользователей показал, что подавляющее большинство пользователей `ultra` переплачивают абонентскую плату за огромные пакеты звонков и СМС, в то время, как пользователи `smart` преимущественно расходуют данные услуги в рамках тарифа.

Переплата за интернет наблюдается у пользователей обоих тарифов примерно одинаково.

Однако за счёт большего количества абонентов тарифа `smart`, а также за счёт их склонности чаще перерасходовать более дорогие (дополнительная плата за превышение) услуги, можно утверждать, что именно **тариф `smart` более выгоден для Мегалайна по данным за 2018 год**.

Замечание: следует уведомить инженеров по данным о возможной технологической ошибке (наличие лишнего столбца неясного содержания в таблице `internet`).