

Определение перспективного тарифа для телеком компании

Проект на тему "Статистический анализ данных" учебного курса "Специалист по Data Science" от Яндекс.Практикум.

Выполнил **Денис Абрашин** ([e-mail](#)) Версия 1.0

Оглавление

- Задание
 - Описание тарифов
 - Тариф «Смарт»
 - Тариф «Ультра»
 - Инструкция по выполнению проекта
 - Задание 1. Открытие и первичный анализ файлов с данными
 - Задание 2. Подготовка данных
 - Задание 3. Анализ данных
 - Задание 4. Проверка гипотез
 - Задание 5. Напишите общий вывод
 - Общие требования
 - Описание полей набора данных
 - Таблица tariffs (информация о тарифах)
 - Таблица users (информация о пользователях)
 - Таблица calls (информация о звонках)
 - Таблица messages (информация о сообщениях)
 - Таблица internet (информация об интернет-сессиях)
 - Критерии проверки
- Подключение и настройка необходимых библиотек
 - Настройка pandas
 - Настройка matplotlib и seaborn
 - Проверка версий подключенных библиотек
- Шаг 1. Открытие и первичный анализ файлов с данными†
 - Замысел выполнения шага 1
 - Словарь полей данных и условий их автоматизированной проверки
 - Информация об одном поле
 - Информация о всех полях и их типах
 - Информация об одном условии проверки набора данных
 - Вспомогательный класс для загрузки и проверки файла данных
 - Замысел вспомогательного класса
 - Код вспомогательного класса
 - Наиболее употребимые элементы условий проверки
 - Специфические для полученного набора данных условия проверки
 - Параметры загрузки и открытия файлов
 - Загрузка файлов данных

- Шаблон операций с таблицами
- Вывод краткой информации о наборах данных
- Изучение распределений числовых значений
- Просмотр примеров записей из таблиц
- Вывод по шагу 1
- Шаг 2. Подготовка данных
 - Поиск пропусков, ошибок и аномалий в данных
 - Первая автоматизированная проверка типов полей
 - Первая автоматизированная проверка значений
 - Предобработка данных
 - Заполнение пропусков в датах прекращения пользования тарифом
 - Обработка нулевых длительностей звонков
 - Обработка нулевых интернет-сессий
 - Округление длительности звонков и приведение типа к целому
 - Округление интернет-трафика и приведение типа к целому
 - Контрольная автоматизированная проверка
 - Преобразование названия тарифного плана в категориальную переменную
 - Преобразование названия города в категориальную переменную
 - Оценка количества полных дубликатов
 - Ориентация во времени
 - Путешествие во времени
 - Получение биллинга
 - Проверка биллинга
 - Удаление оборванных месяцев
 - Вывод по шагу 2
- Шаг 3. Анализ данных
 - Инструменты анализа
 - Ареал обитания клиентов и релевантность выборки
 - Критерии выделения выборок для сравнения
 - Распределение количества звонков в месяц
 - Распределение минут разговора в месяц
 - Распределение количества отправляемых сообщений в месяц
 - Распределение объема интернет-трафика в месяц
 - Распределение стоимости дополнительных услуг в месяц
 - Распределение расходов пользователей в месяц
 - Эволюция пользователей по мере приобретения опыта
 - Москва и другие города
 - Сравнение тарифов
 - Тарифы в Москве и регионах
 - Вывод по шагу 3
- Шаг 4. Проверка гипотез
 - Инструмент проверки гипотез
 - Выбор уровня значимости
 - Проверка гипотезы о среднемесячной выручке по тарифам
 - Проверка гипотезы о среднемесячной выручке по регионам
 - Гипотезы о Москве и других городах
 - Гипотезы о разных тарифах

- Гипотезы о новых абонентах
- Вывод по шагу 4
- Шаг 5. Общий вывод
- Таблица контроля версий
- Бонус
 - Счет за услуги связи
 - Несчастливый клиент
 - Беззаботный пользователь
 - Король Интернета

Задание

Вы аналитик компании «Мегалайн» — федерального оператора сотовой связи. Клиентам предлагаются 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.

Описание тарифов

Тариф «Смарт»

Ежемесячная плата: 550 рублей.

Включено 500 минут разговора, 50 сообщений и 15 Гб интернет-трафика.

Стоимость услуг сверх тарифного пакета:

- минута разговора: 3 рубля;
- сообщение: 3 рубля;
- 1 Гб интернет-трафика: 200 рублей.

Тариф «Ультра»

Ежемесячная плата: 1950 рублей.

Включено 3000 минут разговора, 1000 сообщений и 30 Гб интернет-трафика.

Стоимость услуг сверх тарифного пакета:

- минута разговора: 1 рубль;
- сообщение: 1 рубль;
- 1 Гб интернет-трафика: 150 рублей.

Обратите внимание: «Мегалайн» всегда округляет вверх значения минут и мегабайтов.

Если пользователь проговорил всего 1 секунду, в тарифе засчитывается целая минута.

Инструкция по выполнению проекта

Задание 1. Открытие и первичный анализ файлов с данными

Откройте файлы с данными и изучите **общую информацию**:

- **о тарифах** - [описание](#));
- **о пользователях** - [описание](#));
- **о звонках** - [описание](#));
- **об интернет-сессиях** - [описание](#));
- **о сообщениях** - [описание](#)).

Задание 2. Подготовка данных

Приведите данные к нужным типам.

Найдите и исправьте ошибки в данных.

Поясните, какие ошибки вы нашли и как их исправили. Обратите внимание, что длительность многих звонков — 0.0 минут. Это могут быть пропущенные звонки. Обрабатывать ли эти нулевые значения, **решать вам** — оцените, как их отсутствие повлияет на результаты анализа.

Посчитайте для каждого пользователя:

- количество сделанных звонков и израсходованных минут разговора по месяцам;
- количество отправленных сообщений по месяцам;
- объем израсходованного интернет-трафика по месяцам;
- 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).

Задание 3. Анализ данных

Опишите поведение клиентов оператора, исходя из **выборки**.

Сколько минут разговора, сколько сообщений и какой объём интернет-трафика требуется пользователям каждого тарифа в месяц?

Посчитайте среднее количество, дисперсию и стандартное отклонение.

Постройте **гистограммы**.

Опишите распределения.

Задание 4. Проверка гипотез

Проверьте гипотезы:

- **средняя выручка** пользователей тарифов «Ультра» и «Смарт» различается;
- **средняя выручка** пользователей из Москвы отличается от выручки пользователей из других регионов.

Пороговое значение alpha **задайте** самостоятельно.

Поясните:

- как вы формулировали нулевую и альтернативную гипотезы;

- какой критерий использовали для проверки гипотез и почему.

Задание 5. Напишите [общий вывод](#)

Общие требования

Задание выполните в Jupyter Notebook. Программный код заполните в ячейках типа code, текстовые пояснения — в ячейках типа markdown. Примените форматирование и заголовки.

Если объединение таблиц методом `merge` приводит к ошибке `dead kernel`, примените метод `join` — это облегчит нагрузку на Jupyter Notebook.

Описание полей набора данных

Таблица tariffs (информация о тарифах)

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

Таблица users (информация о пользователях)

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

Таблица calls (информация о звонках)

Имя поля	Описание поля
<code>id</code>	уникальный номер звонка

Имя поля	Описание поля
call_date	дата звонка
duration	длительность звонка в минутах
user_id	идентификатор пользователя, сделавшего звонок

Таблица messages (информация о сообщениях)

Имя поля	Описание поля
id	уникальный номер сообщения
message_date	дата сообщения
user_id	идентификатор пользователя, отправившего сообщение

Таблица internet (информация об интернет-сессиях)

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

Критерии проверки

На что обращают внимание, проверяя проект:

- Как вы описываете выявленные в данных проблемы?
- Как готовите данные к анализу?
- Какие графики строите для распределений?
- Как интерпретируете полученные графики?
- Как рассчитываете стандартное отклонение и дисперсию?
- Формулируете ли альтернативную и нулевую гипотезы?
- Какие методы применяете для проверки гипотез?
- Интерпретируете ли результат проверки гипотезы?
- Соблюдаете структуру проекта и поддерживаете аккуратность кода?
- Какие выводы делаете?
- Оставляете ли комментарии к шагам?

Подключение и настройка необходимых библиотек

```
In [1]: # отключение предупреждений при желании
# import warnings; warnings.filterwarnings('once')
```

```
In [2]: import os
import errno
```

```
In [3]: from collections import namedtuple, defaultdict
```

```
In [4]: from urllib.request import urlretrieve
```

```
In [5]: from IPython.display import HTML, display
```

Настройка pandas

```
In [6]: # обновление библиотеки Pandas для исключения ошибок версий  
!pip3 install --upgrade --user pandas
```

```
Collecting pandas  
  Downloading pandas-1.1.4-cp37-cp37m-manylinux1_x86_64.whl (9.5 MB)  
    ██████████ | 9.5 MB 1.8 MB/s eta 0:00:01  
Requirement already satisfied, skipping upgrade: numpy>=1.15.4 in /opt/conda/lib/python3.7/site-packages (from pandas) (1.19.0)  
Requirement already satisfied, skipping upgrade: python-dateutil>=2.7.3 in /opt/conda/lib/python3.7/site-packages (from pandas) (2.8.1)  
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /opt/conda/lib/python3.7/site-packages (from pandas) (2020.1)  
Requirement already satisfied, skipping upgrade: six>=1.5 in /opt/conda/lib/python3.7/site-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)  
Installing collected packages: pandas  
Successfully installed pandas-1.1.4
```

```
In [7]: import numpy as np  
import pandas as pd
```

```
In [8]: pd.set_option('display.notebook_repr_html', True)  
pd.set_option('display.max_columns', 8)  
pd.set_option('display.max_rows', 10)  
pd.set_option('display.width', 80)
```

Настройка matplotlib и seaborn

```
In [9]: # обновление библиотек  
!pip3 install --upgrade --user matplotlib seaborn
```

```
Collecting matplotlib  
  Downloading matplotlib-3.3.3-cp37-cp37m-manylinux1_x86_64.whl (11.6 MB)  
    ██████████ | 11.6 MB 19.9 MB/s eta 0:00:01 | ██████████  
    ██████████ | 10.5 MB 1.3 MB/s eta 0:00:01  
Collecting seaborn  
  Downloading seaborn-0.11.0-py3-none-any.whl (283 kB)  
    ██████████ | 283 kB 29.0 MB/s eta 0:00:01  
Requirement already satisfied, skipping upgrade: pillow>=6.2.0 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (7.0.0)  
Requirement already satisfied, skipping upgrade: pyparsing!=2.0.4,!>2.1.2,!>2.1.6,>=2.0.3 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (2.4.7)  
Requirement already satisfied, skipping upgrade: kiwisolver>=1.0.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (1.2.0)  
Requirement already satisfied, skipping upgrade: cycler>=0.10 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (0.10.0)  
Requirement already satisfied, skipping upgrade: python-dateutil>=2.1 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (2.8.1)  
Requirement already satisfied, skipping upgrade: numpy>=1.15 in /opt/conda/lib/python3.7/site-packages (from matplotlib) (1.19.0)  
Requirement already satisfied, skipping upgrade: scipy>=1.0 in /opt/conda/lib/python3.7/site-packages (from seaborn) (1.4.1)  
Requirement already satisfied, skipping upgrade: pandas>=0.23 in /home/jovyan/.local/lib/python3.7/site-packages (from seaborn) (1.1.4)  
Requirement already satisfied, skipping upgrade: six in /opt/conda/lib/python3.7/site-packages (from cycler>=0.10->matplotlib) (1.15.0)  
Requirement already satisfied, skipping upgrade: pytz>=2017.2 in /opt/conda/lib/python3.7/site-packages (from pandas>=0.23->seaborn) (2020.1)  
Installing collected packages: matplotlib, seaborn  
Successfully installed matplotlib-3.3.3 seaborn-0.11.0
```

```
import matplotlib as mpl
```

```
In [10]: import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
```

```
In [11]: %matplotlib inline
```

```
In [12]: small, medium, large = 12, 16, 22
params = {'figure.figsize': (12, 6),
          'figure.titlesize': large,
          'legend.fontsize': medium,
          'axes.titlesize': medium,
          'axes.labelsize': medium,
          'xtick.labelsize': medium,
          'ytick.labelsize': medium,
          'legend.loc': 'best'}
plt.rcParams.update(params)
```

```
In [13]: # повышение четкости графиков для больших мониторов
%config InlineBackend.figure_format = 'retina'
```

```
In [14]: # включение цветового оформления Seaborn
#plt.style.use('seaborn-whitegrid')
sns.set_style("white")
```

Проверка версий подключенных библиотек

```
In [15]: def lib_versions(libs):
    for lib in libs: print('Версия', lib.__name__, '-', lib.__version__)
```

```
In [16]: lib_versions([np, pd, mpl, sns])
```

```
Версия numpy - 1.19.0
Версия pandas - 1.1.4
Версия matplotlib - 3.3.3
Версия seaborn - 0.11.0
```

Шаг 1. Открытие и первичный анализ файлов с данными¶

[Задание описано выше](#)

Замысел выполнения шага 1

Не будет преувеличением сказать, что прохождение учебного курса и приближение к реальным задачам сопровождается увеличением количества источников данных и их объемов. Проведение предобработки информации только вручную будет отнимать много времени и сил, что в свою очередь может привести к снижению качества анализа или срыва сроков сдачи проекта.

Для нивелирования описанной проблемы создадим вспомогательные структуры данных, классы и методы для автоматической загрузки массива данных, его первоначальной проверки и подготовки удобочитаемого отчета об обнаруженных ошибках в значениях полей.

Полученный опыт будем использовать не только в решении конкретных задач, но и в создании (совершенствовании) средств автоматической предобработки данных, которые, например, могут быть оформлены в виде отдельных библиотек.

Если Вам не терпится посмотреть результат, можете сразу перейти к [загрузке файлов данных](#).

Словарь полей данных и условий их автоматизированной проверки

Опишем словарь со списком всех полей нашего набора данных в соответствии с их [описанием в задании](#). Эх, получить бы сразу такую информацию от поставщика данных в удобном формате типа JSON:

Информация об одном поле

Используем `namedtuple` для описания структуры информации об отдельном поле набора данных. В коде программ к этим полям будем обращаться через точку:

- `desc` - смысловое описание поля;
- `desc_short` - смысловое описание поля в кратком формате для небольших графиков;
- `dtype` - тип поля;
- `try_to_fix` - если `True`, то попытаться автоматически исправить тип поля на указанный в `dtype`,
- `name_in_csv` - имя поля в файле данных.

```
In [17]: DataField = namedtuple('DataField', ['desc_short', 'desc', 'dtype', 'try_to_fix',
                                             defaults=[None, None, np.object, True,
```

Информация о всех полях и их типах

Сведем все имена полей, их описания и требования к типам в один словарь для последующей алгоритмизации. В учебных целях код оставим в этом ноутбуке. В будущем можем вынести его в отдельный файл.

```
In [18]: data_fields = \
{
    'Тарифы':
    {
        'tariff_name': DataField('тариф', 'название тарифа', np.object),
        'rub_monthly_fee': DataField('абонентская плата', 'ежемесячная абонентская'),
        'minutes_included': DataField('минут разговора', 'количество минут разговор'),
        'messages_included': DataField('сообщений', 'количество сообщений в месяц, в'),
        'mb_per_month_included': DataField('интернет (Мб)', 'объём интернет-трафика, вкл'),
        'rub_per_minute': DataField('стоимость минуты', 'стоимость минуты разгово'),
        'rub_per_message': DataField('стоимость сообщения', 'стоимость отправки со'),
        'rub_per_gb': DataField('стоимость 1 Гб', 'стоимость дополнительного')
    },
    'Пользователи':
    {
        'user_id': DataField('№ пользователя', '的独特ый идентификатор пользователя'),
        'last_name': DataField('фамилия пользователя', 'фамилия пользователя'),
        'first_name': DataField('имя пользователя', 'имя пользователя'),
        'tariff': DataField('тариф', 'название тарифного плана'),
        'reg_date': DataField('дата подключения', 'дата подключения тарифа (день, меся'),
        'churn_date': DataField('дата прекращения', 'дата прекращения пользования тарифо'),
        'city': DataField('город', 'город проживания пользователя'),
        'age': DataField('возраст (лет)', 'возраст пользователя (годы)', np.int64)
    },
    'Звонки':
}
```

```

{
    'id': DataField('№ звонка', 'уникальный номер звонка'),
    'user_id': DataField('№ пользователя', 'идентификатор пользователя, сделавшего'),
    'call_date': DataField('дата звонка', 'дата звонка', np.dtype('datetime64[ns]')),
    'duration': DataField('длительность', 'длительность звонка в минутах', np.int64
    ),
}

'Сообщения':
{
    'id': DataField('№ сообщения', 'уникальный номер сообщения'),
    'user_id': DataField('№ пользователя', 'идентификатор пользователя, отправи'),
    'message_date': DataField('дата сообщения', 'дата сообщения', np.dtype('datetime
    ),
}

'Интернет':
{
    'id': DataField('№ сессии', 'уникальный номер сессии'),
    'user_id': DataField('№ пользователя', 'идентификатор пользователя', np.int),
    'session_date': DataField('дата интернет-сессии', 'дата интернет-сессии', np.dty
    'mb_used': DataField('объём трафика', 'объём потраченного за сессию интерне
    },
}

```

Опишем вспомогательные функции, которые пригодятся при подписывании осей графиков и именовании столбцов таблиц:

```
In [19]: def get_desc_long(table_name, field_names):
    """Возвращает список смысловых описаний для переданного списка полей"""
    return list(map(lambda x: data_fields[table_name][x].desc, field_names))

def get_desc_short(table_name, field_names):
    """Возвращает список смысловых описаний в кратком формате для переданного списка
    return list(map(lambda x: data_fields[table_name][x].desc_short, field_names))
```

Информация об одном условии проверки набора данных

Аналогично опишем информацию об одном конкретном условии проверки значений набора данных:

- `fields` - список имен проверяемых полей;
- `desc_template` - шаблон смыслового описания условия проверки, которое будет выводится в дальнейшем на экран с использованием значения `fields` в качестве параметра к `str.format`;
- `query_template` - шаблон строки запроса на выборку неудовлетворяющих условию записей через `pandas.query` с использованием значения `fields` в качестве параметра к `str.format`;
- `fix` - функция корректировки найденных ошибок (получает на вход ссылку на набор данных). Если нет необходимости исправлять ошибки, то присвойте `fix` значение `None`. Функцию удобно задавать через `lambda`. `fix` можно применять не только для исправления ошибок, но и просто для вывода дополнительной информации после проверки очередного условия.

```
In [20]: DataChecking = namedtuple('DataChecking', ['fields', 'desc_template', 'query_templat
```

Вспомогательный класс для загрузки и проверки файла данных

Опишем класс, облегчающий предобработку одного файла данных. В учебных целях его

код оставим в этом ноутбуке. В будущем можем вынести его в отдельный файл. Заказчика кодом не удивишь, а только напугаешь!

Замысел вспомогательного класса

К разрабатываемому классу предъявим следующие требования:

- загрузка файла должна осуществляться с локального или сетевого источника (при отсутствии локального);
- параметры открытия файла через Pandas должны быть настраиваемыми;
- проверку файла по заданным критериям можно осуществлять повторно (вдруг в будущих проектах машинного обучения валидационную выборку нам сразу не предоставят или обучающую дополнят);
- отчет о проверке файла должен быть удобочитаемым.

Код вспомогательного класса

```
In [21]: class PandasFile(object):
    """Класс для описания одного файла, который необходимо загрузить для дальнейшего использования"""

    def __init__(self, name, file_path, file_url, pandas_params={}, data_fields=None):
        """Конструктор"""

        self.__name = name
        self.__file_path = file_path
        self.__file_url = file_url
        self.__pandas_params = pandas_params
        self.data_fields = data_fields
        self.data_checkings = data_checkings
        self.__df = None

    # проверить наличие файла, загрузить его при отсутствии
    if self.exists():
        print(f'{self.__name} ({self.__file_path}) был загружен ранее.')
    else:
        self.load_from_url()

    # создать датафрейм из файла
    self.get_data_frame(reread=True)

    def exists(self):
        """Проверка наличия локального файла"""
        return os.path.isfile(self.__file_path)

    def load_from_url(self):
        """Загрузка файла из сетевого источника"""
        # создаем папку для сохранения файла с данными, если она еще не создана
        folder_path = os.path.dirname(self.__file_path)
        if not os.path.exists(folder_path):
            print(f'Создаю папку {folder_path}')
            os.makedirs(folder_path)

        print(f'Загружаю "{self.__name}" из {self.__file_url} в {self.__file_path} .')
        result = urlretrieve(url=self.__file_url, filename=self.__file_path)
        print('OK')
        return result

    def get_data_frame(self, reread=False):
        """Загрузка датафрейма Pandas из локального файла"""
        if reread or self.__df is None:
            print(f'Открываю "{self.__name}" с помощью Pandas ... ', end=' ')
            self.__df = pd.read_csv(self.__file_path, **self.__pandas_params)
```

```

print('OK')
# переименование полей при необходимости
for field_name, field_info in self.data_fields.items():
    if field_info.name_in_csv is not None:
        print(f'Переименовываю поле "{field_info.name_in_csv}" в "{field_name}"')
        self._df.rename(columns={field_info.name_in_csv: field_name}, i
                        print('OK')
# пересортировка столбцов в порядке их следования в списке
self._df = self._df[self.data_fields.keys()]
return self._df

def check_data_frame_fields(self, check_try_to_fix=True, data_fields=None, prefix=''):
    """Проверка полей датасета на соответствие типам, указанным в data_fields.
    Возвращает количество несоответствий заданным требованиям.
    prefix - строка-префикс для создания ссылок на ошибки с использованием тега
    """
    display(HTML(f'  
Протокол автоматизированной проверки типов полей набора'))

    # выбираем список с информацией о полях для проверки
    if data_fields is None:
        data_fields = self.data_fields

    # если для набора данных не заданы описания полей...
    if data_fields is None:
        display(HTML(f'В наборе данных "{self.__name}" нет описания полей'))
        return -1

    # обнулим счетчик ошибок
    errors = 0

    # полный префикс ссылки тега <a>
    link_prefix = f'{self.__name}-{prefix}-'.replace(' ', '-')

    # для каждого описания поля...
    for field_name, field_info in data_fields.items():
        try:
            # проверка существования поля
            field_dtype = self._df[field_name].dtypes

            # если тип поля отличается от предполагаемого...
            if field_dtype != field_info.dtype:
                errors += 1
                display(HTML(f''
                            f'{errors} Тип {field\_dtype} поля {field\_name} '
                            f'\({{field\_info.desc}}\) не соответствует заявлен'
                            f' \({{{field\_info.dtype.\_\_name\_\_}}}\)'\)\)
        except Exception as e:
            display\(HTML\(f'При попытке изменения типа поля произошла'\)\)

            # проверяем тип поля после попытки его изменения
            field\_dtype = self.\_df.dtypes\[field\_name\]
            if field\_dtype == field\_info.dtype:
                display\(HTML\(f'Тип поля {field\_name} \({field\_info.dtype.\_\_name\_\_}\) '
                            f'изменен на {field\_info.dtype.\_\_name\_\_}'\)\)
            else:
                # "назад в будущее" - создадим гиперссылку на блок исправлен
                display\(HTML\(f'

См. и'
                            f'{errors}

'\)\)

```

```

# обработка исключительных ситуаций
except KeyError as e:
    errors += 1
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'<b>{errors}</b> Поле с именем <b>"{field_name}"</b>')

if errors == 0:
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'В наборе данных <b>"{self.__name}"</b> ошибок в типах не'))

# возвращаем общее количество обнаруженных замечаний
return errors

def check_data_frame_values(self, check_fix=True, example_count=3, data_fields=None,
                           prefix='value error'):
    """Проверка значений датасета на соответствие условиям, указанным в data_checkings.
    Возвращает количество несоответствий заданным требованиям."""
    display(HTML(f'<br><b>Протокол автоматизированной проверки значений набора данных</b>'))

    # выбираем список с информацией об условиях проверки значений
    if data_checkings is None:
        data_checkings = self.data_checkings

    # если для набора данных не заданы условия проверки значений...
    if data_checkings is None:
        display(HTML(f'В наборе данных <b>"{self.__name}"</b> нет условий проверки.' +
                    f'Проверка не проводилась.'))
        return -1

    # выбираем список с информацией о полях для проверки
    if data_fields is None:
        data_fields = self.data_fields

    # если для набора данных не заданы описания полей...
    if data_fields is None:
        display(HTML(f'В наборе данных <b>"{self.__name}"</b> нет описания полей.' +
                    f'Расширенные пояснения не будут выводиться.'))

    # обнулим счетчик ошибок
    errors = 0

    # полный префикс ссылки тега <a>
    link_prefix = f'{self.__name}-{prefix}'.replace(' ', '-')

    # для каждого условия проверки...
    for check in data_checkings.copy():
        try:
            # проверим существование полей
            for f in check.fields: field_type = self.__df[f].dtypes

            # отберем записи набора, которые не удовлетворяют условиям проверки
            check_df = self.__df.query(check.query_template.format(*check.fields))

            # определим их количество
            lines_count = check_df.shape[0]

            # если обнаружены нарушения...
            if check_df.shape[0] > 0:
                errors += 1

            # выведем текст ошибки с указанием количества записей и их доли
            percent = lines_count / self.__df.shape[0]

            # сформируем текст с перечислением
            display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                        f'<b>{percent}</b> % из {self.__df.shape[0]} записей в наборе <b>"{self.__name}"</b> '
                        f'не соответствуют условиям проверки <b>"{check.name}"</b>.'))

        except Exception as e:
            errors += 1
            display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                        f'<b>{errors}</b> Ошибка при проверке условия <b>"{check.name}"</b>.'))

    return errors

```

```

try:
    fields_list = ', '.join(map(lambda f: f"<b>'{f}'</b> " +
                                f"\"{data_fields[f].desc}\")", check.fields))
except:
    fields_list = ', '.join(map(lambda f: f"<b>'{f}'</b>", check.fields))

display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
            f'<b>{errors}</b> Обнаружены замечания к значениям ' +
            f'{check.desc_template.format(*check.fields)}.' +
            f'Количество записей с нарушением: <b>{lines_count}</b>')

# если обработчик ошибки задан...
if check_fix and check.fix is not None:
    # вызовем его, передав ссылку на условие проверки и датасет
    check.fix(check, check_df, self.__df)
else:
    # иначе выведем пример ошибок в табличном виде
    if example_count > 0:
        display(HTML('Пример:'))
        #display(check_df[check.fields].head(example_count))
        display(check_df.head(example_count))

    # "назад в будущее" - создадим гиперссылку на блок исправления ошибок
    display(HTML(f'<p><a href="#{link_prefix}{errors}-fix">См. исправление <b>{errors}</b></a></p>'))

# обработка исключительных ситуаций
except KeyError as e:
    errors += 1
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'<b>{errors}</b> Поле с именем <b>"{e}"</b> в датасете'))

except NameError as e:
    errors += 1
    # определим имя поля, совпадающее с зарезервированным словом,
    # из последних скобок сообщения NameError
    text = str(e)[::-1]
    reserved_name = text[text.find('(') + 1 : text.find(')')[::-1]]
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'<b>{errors}</b> Имя поля <b>{reserved_name}</b> совпадает с зарезервированным ' +
                f'словом. Переименуйте его для возможности использовать <b>pandas.query</b>. Проверка поля <b>{reserved_name}</b>'))

if errors == 0:
    display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                f'В наборе данных <b>"{self.__name}"</b> ошибок в значениях'))

# возвращаем общее количество обнаруженных замечаний
return errors

@property
def df(self):
    """Возвращает ссылку на последнюю созданную копию датасета Pandas"""
    return self.get_data_frame(reread=False)

@property
def name(self):
    return self.__name

```

Наиболее употребимые элементы условий проверки

Опишем наиболее часто употребимые элементы условий проверки. Вместо `{0}`, `{1}` и так далее программа (см. ниже) будет подставлять имя проверяемого поля из `fields` по индексу, начиная с нуля:

```
In [22]: # проверка на пустое значение одного поля
desc_template_nan      = 'Значение {0} не должно быть пустым'
query_template_nan     = '`{0}` != `{}`'

# проверка на пустое значение двух полей одновременно
desc_template_nan_2    = 'Значения {0} и {1} не должны быть пустыми одновременно'
query_template_nan_2   = '(`{0}` != `{}`) and (`{1}` != `{}`')

# другие частые проверки
desc_template_g_zero   = 'Значение "{0}" должно быть больше нуля'
query_template_g_zero  = '`{0}` <= 0'

desc_template_ge_zero  = 'Значение "{0}" должно быть не меньше нуля'
query_template_ge_zero = '`{0}` < 0'

desc_template_int       = 'значение "{0}" должно быть округлено'
query_template_int      = '(`{0}` % 1) != 0'
```

Специфические для полученного набора данных условия проверки

Проверочные условия зададим, исходя из здравого смысла и опыта специалистов в сфере телекоммуникации. Например, дата прекращения пользования тарифом должна быть не раньше даты подключения. В спецификации к массиву сказано об округлении некоторых значений и т.д.

Выявление подобных аномалий на ранней стадии очень важно как для анализа и генерации новых атрибутов по имеющимся данным. Их искаженные значения могут отрицательно сказаться на качестве машинного обучения, к которому мы когда-нибудь подберемся.

В следующих проектах попробуем реализовать запись условий проверки в форме, максимально приближенной к естественному языку. А пока воспользуемся описанной выше структурой `DataChecking`.

```
In [23]: # создадим словарь списков проверок для каждой таблицы
data_checkings = defaultdict(list)

# для всех таблиц...
for table_name in data_fields:
    # для всех полей таблицы...
    for field in data_fields[table_name].keys():
        # для всех полей проверим наличие пропусков
        data_checkings[table_name] += [DataChecking([field], desc_template_nan,
                                                    query_template_nan, None)]

        # все числовые поля должны быть положительными
        if data_fields[table_name][field].dtype in [np.int64, np.float64]:
            data_checkings[table_name] += [DataChecking([field], desc_template_g_zero,
                                                    query_template_g_zero, None)]

# по правилам оператора связи некоторые величины должны округляться
for table, field in (('Звонки', 'duration'), ('Интернет', 'mb_used')):
    data_checkings[table] += [DataChecking([field], desc_template_int,
                                         query_template_int, None)]

# 'Пользователи':
data_checkings['Пользователи'] += [DataChecking(['reg_date', 'churn_date'],
                                                'Дата прекращения пользования тарифом  
быть не раньше даты подключения',
                                                '`{0}` > `{1}`', None)]
```

Отсортируем наши условия проверки по названию полей для удобства их дальнейшей

обработки:

```
In [24]: # для всех списков проверок...
for checkings in data_checkings.values():
    checkings.sort(key = lambda x : ' '.join(x.fields))
```

Параметры загрузки и открытия файлов

Зададим необходимый набор параметров для открытия файла данных нашего проекта с помощью `pandas.read_csv`:

```
In [25]: pandas_params = \
{
    # в данном проекте особые параметры загрузки файлов пока не требуются
}
```

Зададим локальную папку для хранения файлов данных (по умолчанию `'datasets'`). При необходимости укажите более удобное место:

```
In [26]: local_folder = 'datasets'
```

Загрузка файлов данных

Зададим список всех файлов данных нашего проекта. В данном случае файл один, но в будущем их будет гораздо больше (тогда список можно будет легко пополнить).

Выполнение следующей ячейки должно привести к загрузке всех отсутствующих локально файлов и созданию датафреймов, как было описано в [замысле выполнения шага 1](#) практической работы:

```
In [27]: data_files = \
{
    'Тарифы':
        PandasFile
    (
        'Информация о тарифах',
        os.path.join(local_folder, 'tariffs.csv'),
        '',
        '# сетевой адрес файла
        {},
        data_fields['Тарифы'],
        data_checkings['Тарифы']
    ),
    'Пользователи':
        PandasFile
    (
        'Информация о пользователях',
        os.path.join(local_folder, 'users.csv'),
        '',
        {'parse_dates': ['reg_date', 'churn_date']},
        data_fields['Пользователи'],
        data_checkings['Пользователи']
    ),
    'Звонки':
        PandasFile
    (
        'Информация о звонках',
        os.path.join(local_folder, 'calls.csv'),
        '',
        {'parse_dates': ['call_date']},
        data_fields['Звонки'],
        data_checkings['Звонки']
    )
}
```

```

        ),
    'Сообщения':
        PandasFile
    (
        'Информация о сообщениях',
        os.path.join(local_folder, 'messages.csv'),
        '',
        {'parse_dates': ['message_date']},
        data_fields['Сообщения'],
        data_checkings['Сообщения']
    ),
    'Интернет':
        PandasFile
    (
        'Информация об интернет-сессиях',
        os.path.join(local_folder, 'internet.csv'),
        '',
        {'parse_dates': ['session_date']},
        data_fields['Интернет'],
        data_checkings['Интернет']
    )
}

```

"Информация о тарифах" (datasets/tariffs.csv) был загружен ранее.
Открываю "Информация о тарифах" с помощью Pandas ... OK
"Информация о пользователях" (datasets/users.csv) был загружен ранее.
Открываю "Информация о пользователях" с помощью Pandas ... OK
"Информация о звонках" (datasets/calls.csv) был загружен ранее.
Открываю "Информация о звонках" с помощью Pandas ... OK
"Информация о сообщениях" (datasets/messages.csv) был загружен ранее.
Открываю "Информация о сообщениях" с помощью Pandas ... OK
"Информация об интернет-сессиях" (datasets/internet.csv) был загружен ранее.
Открываю "Информация об интернет-сессиях" с помощью Pandas ... OK

Шаблон операций с таблицами

Опишем вспомогательную процедуру для выполнения шаблонных операций для каждой таблицы:

```
In [28]: def do_for_each_data_file(action, header=None, show_title=False, *args, **kwargs):
    """процедура для выполнения шаблонных операций для каждой таблицы"""
    # выведем общий заголовок если он задан
    if header:
        display(HTML(''.join(['<br><b>', header, '</b>'])))

    # для всех таблиц выводим их название и выполняем операцию action
    for data_file in data_files.values():
        if show_title:
            display(HTML(''.join(['<br><b>Таблица ', data_file.name, '"</b>'])))
        action(data_file, *args, **kwargs)
```

Пригодятся итераторы по всем таблицам и по всем полям:

```
In [29]: def tables():
    """Итератор по всем таблицам.
    Последовательно возвращает кортежи (датафрейм, название)"""
    for data_file in data_files.values():
        yield data_file.df, data_file.name
```

```
In [30]: def fields(include=None, exclude=None):
    """Итератор по всем столбцам (полям) с заданными типами.
    Последовательно возвращает кортежи (поле, описание поле)"""
    for data_file in data_files.values():
```

```
for field in data_file.df.select_dtypes(include, exclude):
    yield data_file.df[field], data_file.data_fields[field]
```

И создадим ссылки для быстрого обращения к таблицам:

```
In [31]: table_tariffs = data_files['Тарифы'].df
table_users = data_files['Пользователи'].df
table_calls = data_files['Звонки'].df
table_messages = data_files['Сообщения'].df
table_internet = data_files['Интернет'].df
```

Вывод краткой информации о наборах данных

Выведем краткую информацию о наборах данных с помощью `info` и убедимся в успешности загрузки:

```
In [32]: do_for_each_data_file(lambda f: f.df.info(), 'Информация о таблицах', True)
```

Информация о таблицах

Таблица "Информация о тарифах"

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

Таблица "Информация о пользователях"

```
<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   last_name        500 non-null     object  
 2   first_name       500 non-null     object  
 3   tariff           500 non-null     object  
 4   reg_date         500 non-null     datetime64[ns]
 5   churn_date       38 non-null     datetime64[ns]
 6   city              500 non-null     object  
 7   age               500 non-null     int64  
dtypes: datetime64[ns](2), int64(2), object(4)
memory usage: 31.4+ KB
```

Таблица "Информация о звонках"

```
<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   user_id          202607 non-null  int64
```

```
2   call_date  202607 non-null  datetime64[ns]
3   duration    202607 non-null  float64
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 6.2+ MB
```

Таблица "Информация о сообщениях"

```
<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   user_id     123036 non-null   int64   
 2   message_date 123036 non-null   datetime64[ns]
dtypes: datetime64[ns](1), int64(1), object(1)
memory usage: 2.8+ MB
```

Таблица "Информация об интернет-сессиях"

```
<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   user_id     149396 non-null   int64   
 2   session_date 149396 non-null   datetime64[ns]
 3   mb_used     149396 non-null   float64 
dtypes: datetime64[ns](1), float64(1), int64(1), object(1)
memory usage: 4.6+ MB
```

По беглому просмотру информации о таблицах можно сделать вывод, что **все файлы набора данных успешно загружены в память.**

Таблица тарифов действительно содержит только две записи.

В таблице информации о пользователях в 462 строках из 500 имеются **пропуски** в столбце `churn_date` (дата прекращения пользования тарифом).

Некоторые поля имеют неверный тип - **целый** вместо **вещественный** (цена на услуги в рублях может содержать и доли в копейках).

Изучение распределений числовых значений

```
In [33]: do_for_each_data_file(lambda f: display(f.df.describe()), None, True)
```

Таблица "Информация о тарифах"

	rub_monthly_fee	minutes_included	messages_included	mb_per_month_included	rub_per_min
count	2.000000	2.000000	2.000000	2.000000	2.000000
mean	1250.000000	1750.000000	525.000000	23040.000000	2.000000
std	989.949494	1767.766953	671.751442	10861.160159	1.414214
min	550.000000	500.000000	50.000000	15360.000000	1.000000
25%	900.000000	1125.000000	287.500000	19200.000000	1.500000
50%	1250.000000	1750.000000	525.000000	23040.000000	2.000000
75%	1600.000000	2375.000000	762.500000	26880.000000	2.500000

	rub_monthly_fee	minutes_included	messages_included	mb_per_month_included	rub_per_min
max	1950.000000	3000.000000	1000.000000	30720.000000	3.000000

Таблица "Информация о пользователях"

	user_id	age
count	500.000000	500.000000
mean	1249.500000	46.58800
std	144.481833	16.66763
min	1000.000000	18.00000
25%	1124.750000	32.00000
50%	1249.500000	46.00000
75%	1374.250000	62.00000
max	1499.000000	75.00000

Таблица "Информация о звонках"

	user_id	duration
count	202607.000000	202607.000000
mean	1253.940619	6.755887
std	144.722751	5.843365
min	1000.000000	0.000000
25%	1126.000000	1.300000
50%	1260.000000	6.000000
75%	1379.000000	10.700000
max	1499.000000	38.000000

Таблица "Информация о сообщениях"

	user_id
count	123036.000000
mean	1256.989410
std	143.523967
min	1000.000000
25%	1134.000000
50%	1271.000000
75%	1381.000000
max	1499.000000

Таблица "Информация об интернет-сессиях"

	user_id	mb_used
count	149396.000000	149396.000000
mean	1252.099842	370.192426
std	144.050823	278.300951
min	1000.000000	0.000000
25%	1130.000000	138.187500
50%	1251.000000	348.015000
75%	1380.000000	559.552500
max	1499.000000	1724.830000

Просмотр примеров записей из таблиц

Выведем по три первых записи каждой таблицы:

```
In [34]: do_for_each_data_file(lambda f: display(f.df.head(3)), None, True)
```

Таблица "Информация о тарифах"

	tariff_name	rub_monthly_fee	minutes_included	messages_included	mb_per_month_included	rub
0	smart	550	500	50		15360
1	ultra	1950	3000	1000		30720
	◀				▶	

Таблица "Информация о пользователях"

	user_id	last_name	first_name	tariff	reg_date	churn_date	city	age	
0	1000	Верещагин		Rafaил	ultra	2018-05-25	NaN	Краснодар	52
1	1001	Ежов		Иван	smart	2018-11-01	NaN	Москва	41
2	1002	Абрамович		Евгений	smart	2018-06-17	NaN	Стерлитамак	59

Таблица "Информация о звонках"

	id	user_id	call_date	duration
0	1000_0	1000	2018-07-25	0.00
1	1000_1	1000	2018-08-17	0.00
2	1000_2	1000	2018-06-11	2.85

Таблица "Информация о сообщениях"

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

	<code>id</code>	<code>user_id</code>	<code>message_date</code>
2	1000_2	1000	2018-08-04

Таблица "Информация об интернет-сессиях"

	<code>id</code>	<code>user_id</code>	<code>session_date</code>	<code>mb_used</code>
0	1000_0	1000	2018-11-25	112.95
1	1000_1	1000	2018-09-07	1052.81
2	1000_2	1000	2018-06-25	1197.26

Обратим внимание на то, что имеются записи о звонках с нулевой длительностью. Необходимо разобраться с их природой.

Нам очень повезло увидеть в первых записях **неокругленные минуты** (2.85) и **дробные мегабайты** (112.95), хотя в спецификации [сказано](#), что «Мегалайн» всегда округляет вверх значения минут и мегабайтов.

Поля `id` в таблицах с информацией о звонках, сообщениях и интернет-сессиях имеют строковый тип, но фактически хранят две числовые величины, разделенные символом подчеркивания: идентификатор пользователя и номер операции (уникальный для пользователя). На первый взгляд, для нашей задачи поле `id` избыточно. Если по ходу решения оно понадобится, то его будет целесообразно разбить на два и привести к целому типу для удобства обработки.

Вывод по шагу 1

- Полученные файлы имеют корректный формат, удобный для обработки с помощью библиотеки Pandas.
- Файлы успешно загружены в память.
- Названия столбцов не искажены, соответствуют полученному в задании описанию и могут быть использованы для упрощенной адресации Pandas в виде `data.first_name`, а не только `data['first_name']`.
- Массив данных в целом пригоден для анализа, но требует предобработки, включая как минимум:
 - устранение пропусков;
 - приведение целых типов к вещественным для стоимости;
 - контроль правильности значений в соответствии с их смыслом. Так, например, в протоколах действий абонентов имеются неокругленные вверх длительности звонков и объемы интернет-трафика.
- Для облегчения выполнения предобработки был подготовлен класс, который в соответствии со списком ограничений на данные построит отчет о найденных ошибках и аномалиях. Их устранение целесообразно осуществить на следующем шаге.
- После корректировки значений набора данных следует проверить наличие дубликатов и принять решение об их удалении или способе дальнейшего использования.

Шаг 2. Подготовка данных

Задание описано выше

Поиск пропусков, ошибок и аномалий в данных

Первая автоматизированная проверка типов полей

Выполнив [ручную проверку](#), мы выявили некоторое количество замечаний к массиву данных. Настало время проверить эффективность придуманной выше автоматизированной проверки. Начнем с изучения типов полей загруженных файлов с помощью разработанного выше класса:

```
In [35]: do_for_each_data_file(lambda f: f.check_data_frame_fields(check_try_to_fix=False))
```

Протокол автоматизированной проверки типов полей набора данных "Информация о тарифах":

1. Тип **int64** поля "**rub_monthly_fee**" (ежемесячная абонентская плата в рублях) не соответствует заявленному в описании **float64**.

[См. исправления и комментарии по п. 1](#)

2. Тип **int64** поля "**rub_per_minute**" (стоимость минуты разговора сверх тарифного пакета) не соответствует заявленному в описании **float64**.

[См. исправления и комментарии по п. 2](#)

3. Тип **int64** поля "**rub_per_message**" (стоимость отправки сообщения сверх тарифного пакета) не соответствует заявленному в описании **float64**.

[См. исправления и комментарии по п. 3](#)

4. Тип **int64** поля "**rub_per_gb**" (стоимость дополнительного гигабайта интернет-трафика сверх тарифного пакета) не соответствует заявленному в описании **float64**.

[См. исправления и комментарии по п. 4](#)

Протокол автоматизированной проверки типов полей набора данных "Информация о пользователях":

В наборе данных "**Информация о пользователях**" ошибок в типах не обнаружено.

Протокол автоматизированной проверки типов полей набора данных "Информация о звонках":

1. Тип **float64** поля "**duration**" (длительность звонка в минутах) не соответствует заявленному в описании **int64**.

[См. исправления и комментарии по п. 1](#)

Протокол автоматизированной проверки типов полей набора данных "Информация о сообщениях":

В наборе данных "**Информация о сообщениях**" ошибок в типах не обнаружено.

Протокол автоматизированной проверки типов полей набора данных "Информация об интернет-сессиях":

1. Тип **float64** поля "**mb_used**" (объём потраченного за сессию интернет-трафика (в мегабайтах)) не соответствует заявленному в описании **int64**.

[См. исправления и комментарии по п. 1](#)

Получили список замечаний по набору данных, поиск которых вручную мог бы затянуться.

Вот и первые подтверждения, что оправдались временные затраты на реализацию **замысла выполнения первого шага**. На следующих этапах предобработки данных после исправления очередной порции ошибок и пропусков будем периодически осуществлять автоматическую проверку и радостно наблюдать за сокращением протокола.

Первая автоматизированная проверка значений

Проверим теперь значения массивов данных на соответствие заданным [выше](#) ограничениям, сопроводив результаты пояснениями при необходимости.

Обнаруженные замечания будем автоматически помечать перекрестными гиперссылками с идентификаторами в формате `test 1 value error-{N}` (обнаруженная ошибка) и `test 1 value error-{N}-fix` (обработка ошибки), где `{N}` - номер позиции в протоколе. Это позволит нам удобно продолжить предобработку данных, переходя по этим ссылкам.

```
In [36]: do_for_each_data_file(lambda f: f.check_data_frame_values(check_fix=True, prefix='te
```

Протокол автоматизированной проверки значений набора данных "Информация о тарифах":

В наборе данных "Информация о тарифах" ошибок в значениях не обнаружено.

Протокол автоматизированной проверки значений набора данных "Информация о пользователях":

1. Обнаружены замечания к значениям '`churn_date`' (дата прекращения пользования тарифом). Значение `churn_date` не должно быть пустым. Количество записей с нарушением: **462 (92.40%)**.

Пример:

	<code>user_id</code>	<code>last_name</code>	<code>first_name</code>	<code>tariff</code>	<code>reg_date</code>	<code>churn_date</code>	<code>city</code>	<code>age</code>
0	1000	Верещагин	Рафаил	ultra	2018-05-25	NaT	Краснодар	52
1	1001	Ежов	Иван	smart	2018-11-01	NaT	Москва	41
2	1002	Абрамович	Евгений	smart	2018-06-17	NaT	Стерлитамак	59

[См. исправления и комментарии по п. 1](#)

Протокол автоматизированной проверки значений набора данных "Информация о звонках":

1. Обнаружены замечания к значениям '`duration`' (длительность звонка в минутах). Значение "`duration`" должно быть больше нуля. Количество записей с нарушением: **39613 (19.55%)**.

Пример:

	<code>id</code>	<code>user_id</code>	<code>call_date</code>	<code>duration</code>
0	1000_0	1000	2018-07-25	0.0
1	1000_1	1000	2018-08-17	0.0
5	1000_5	1000	2018-11-02	0.0

[См. исправления и комментарии по п. 1](#)

2. Обнаружены замечания к значениям '**duration**' (длительность звонка в минутах).

Значение "duration" должно быть округлено. Количество записей с нарушением: **161447 (79.68%)**.

Пример:

	id	user_id	call_date	duration
2	1000_2	1000	2018-06-11	2.85
3	1000_3	1000	2018-09-21	13.80
4	1000_4	1000	2018-12-15	5.18

[См. исправления и комментарии по п. 2](#)

Протокол автоматизированной проверки значений набора данных "Информация о сообщениях":

В наборе данных "**Информация о сообщениях**" ошибок в значениях не обнаружено.

Протокол автоматизированной проверки значений набора данных "Информация об интернет-сессиях":

1. Обнаружены замечания к значениям '**mb_used**' (объём потраченного за сессию интернет-трафика (в мегабайтах)). Значение "mb_used" должно быть больше нуля.

Количество записей с нарушением: **19598 (13.12%)**.

Пример:

	id	user_id	session_date	mb_used
11	1000_11	1000	2018-08-28	0.0
38	1000_38	1000	2018-11-27	0.0
46	1000_46	1000	2018-06-30	0.0

[См. исправления и комментарии по п. 1](#)

2. Обнаружены замечания к значениям '**mb_used**' (объём потраченного за сессию интернет-трафика (в мегабайтах)). Значение "mb_used" должно быть округлено.

Количество записей с нарушением: **128462 (85.99%)**.

Пример:

	id	user_id	session_date	mb_used
0	1000_0	1000	2018-11-25	112.95
1	1000_1	1000	2018-09-07	1052.81
2	1000_2	1000	2018-06-25	1197.26

[См. исправления и комментарии по п. 2](#)

Предобработка данных

[Заполнение пропусков в датах прекращения пользования тарифом](#)

Автоматизированная проверка обнаружила 462 (92.40%) пропуска в поле 'churn_date' (дата прекращения пользования тарифом). Пропуски означают, что пользователи продолжают пользоваться тарифным планом. В дальнейших расчетах стоимости предоставленных услуг нам будет неудобно работать с пропусками. Целесообразно заменить их на гарантированно большую дату, чем существующие значения во всём массиве данных. Заодно проверим диапазон дат на адекватность.

```
In [37]: date_min = min([field.min() for field, _ in fields(include=np.dtype('datetime64[ns]'))])
date_max = max([field.max() for field, _ in fields(include=np.dtype('datetime64[ns]'))])
display(HTML('В наборе данных представлены сведения за период '
f'с <b>{date_min:%d.%m.%Y}</b> по <b>{date_max:%d.%m.%Y}</b>.'))
```

В наборе данных представлены сведения за период с **01.01.2018** по **31.12.2018**.

Данные соответствуют условию **задания** и не выходят за 2018 год, следовательно, мы можем заполнить пропуски, например, следующим за периодом днем:

```
In [38]: table_users.churn_date.fillna(date_max + pd.DateOffset(days=1), inplace=True)
```

Обработка нулевых длительностей звонков

Автоматизированная проверка выявила 39613 записей с нулевой длительностью звонка в минутах. Доля этих значений (19.55%) слишком большая, чтобы просто отбросить их.

Выдвинем возможные причины появления в массиве нулевых значений:

- у пользователей на момент совершения звонков ещё не был активирован тарифный план;
- на счете было недостаточное количество денежных средств (к сожалению, мы не сможем проверить это, так как в полученном наборе нет информации о балансе на момент попыток получить услуги связи);
- произошла ошибка округления до целых значений - вместо округления в большую сторону (по правилам оператора) производили округление до ближайшего целого;
- нулевые значения возникли из-за технической ошибки системы регистрации и характерны лишь для абонентов определенной категории (по тарифу, городу, времени и т.п.) - в этом случае придется проверить еще и релевантность выборки;
- звонки были пропущены адресатом или он находился вне зоны доступа, процедура установления связи оборвалась до начала разговора (примем этот вариант, если перечисленные выше не подтверждаются).

У нас нет оснований полагать, что нулевые значения являются следствием ошибки округления в большую сторону, как это установлено оператором. Судя по наличию в таблице **161447 (79.68%) дробных значений**, к ней ранее вообще не применялась операция округления.

Для проверки других версий создадим временный набор данных, объединив информацию о звонках с информацией о пользователях по общему идентификатору:

```
In [39]: temp = pd.merge(table_calls.reset_index(), table_users.reset_index(),
                    on='user_id', right_index=True, how='inner', sort=False,
                    suffixes=('_calls', '_users')) \
[[ 'user_id', 'call_date', 'duration', 'tariff', 'reg_date', 'churn_date', 'c
```

Посчитаем количество попыток звонков, когда тарифный план ещё не был активирован

или был уже отключен:

```
In [40]: ((temp.call_date < temp.reg_date) | (temp.call_date > temp.churn_date)).sum()
```

```
Out[40]: 0
```

Значит у всех пользователей на момент совершения звонков был активирован тарифный план. Значит причина нулевых длительностей в другом.

Вычислим новую категориальную переменную, чтобы отличать звонки с нулевой длительностью от остальных:

```
In [41]: temp['zero_duration'] = (temp.duration == 0)
```

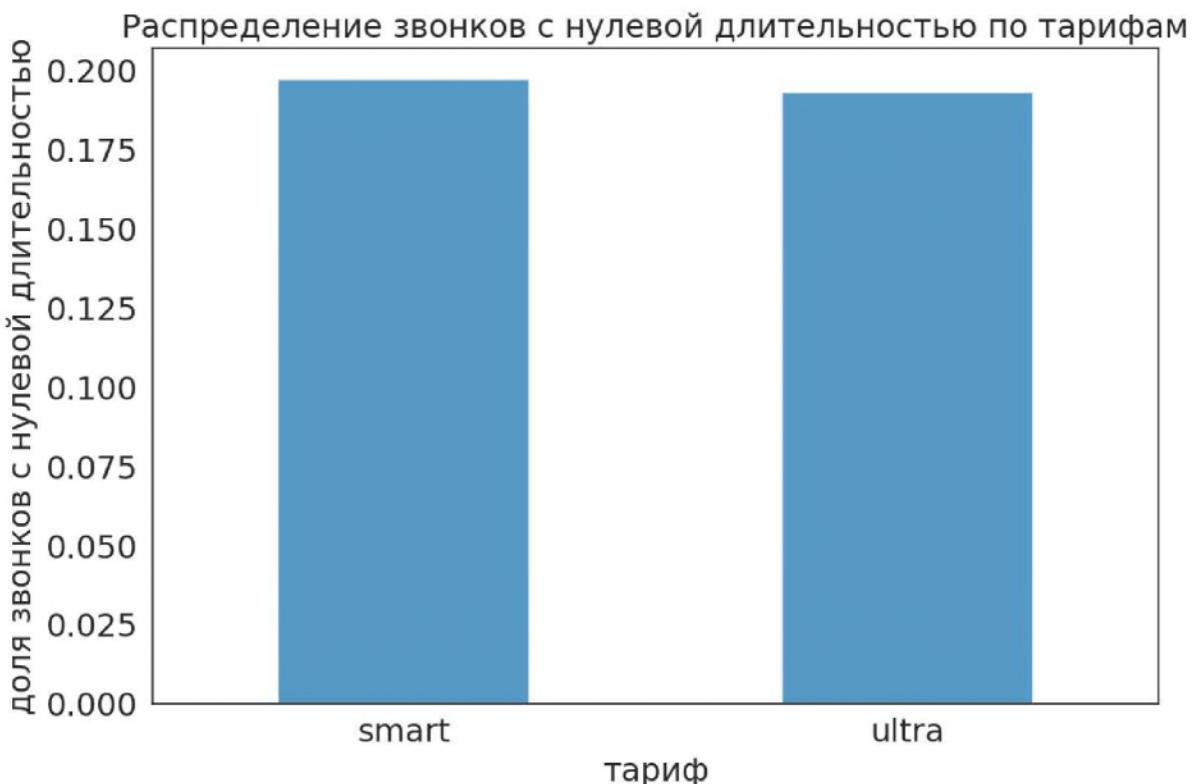
Определим влияние дня недели на долю звонков с нулевой длительностью:

```
In [42]: temp['weekday'] = temp.call_date.dt.weekday + 1
temp.groupby('weekday').zero_duration.mean().plot.bar(alpha=0.75)
plt.xlabel('день недели')
plt.ylabel('доля звонков с нулевой длительностью')
plt.title('Распределение звонков с нулевой длительностью по дням недели')
plt.show()
```



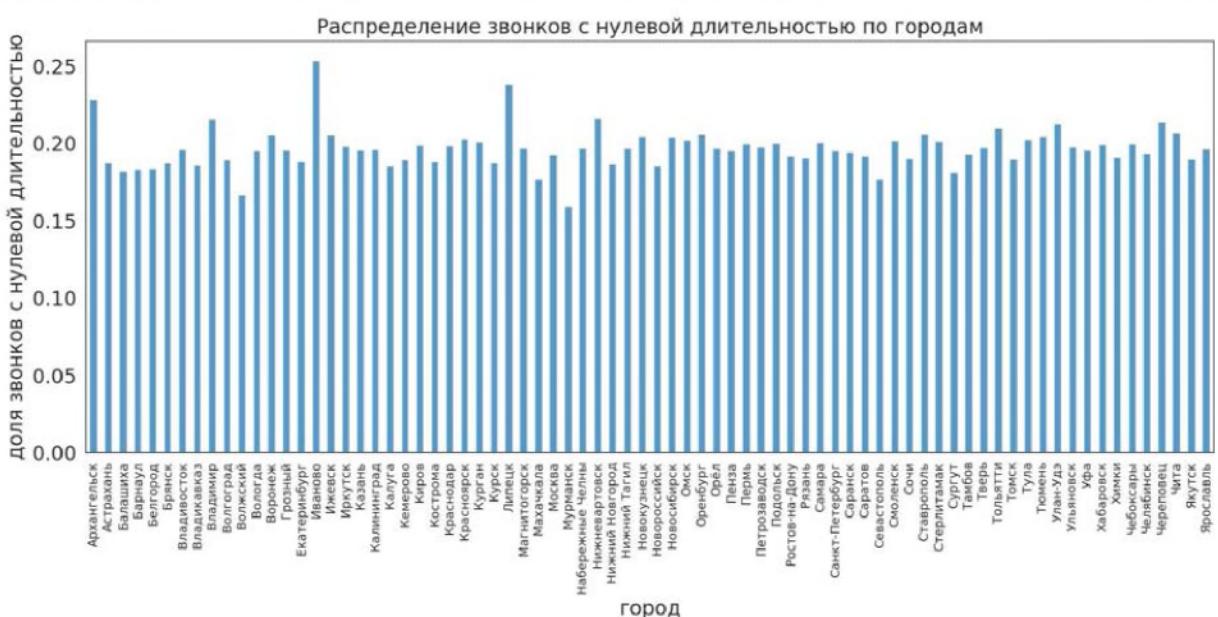
Диаграмма выше показывает, что день недели не влияет на долю звонков с нулевой длительностью. Они с одинаковой вероятностью регистрируются в любой день.

```
In [43]: fig, ax = plt.subplots(figsize=(9, 6))
temp.groupby('tariff').zero_duration.mean().plot.bar(alpha=0.75)
plt.xlabel('тариф')
ax.xaxis.set_tick_params(rotation=0)
plt.ylabel('доля звонков с нулевой длительностью')
plt.title('Распределение звонков с нулевой длительностью по тарифам')
plt.show()
```



Не вызывает нареканий и зависимость доли нулевой длительности звонков от города абонента:

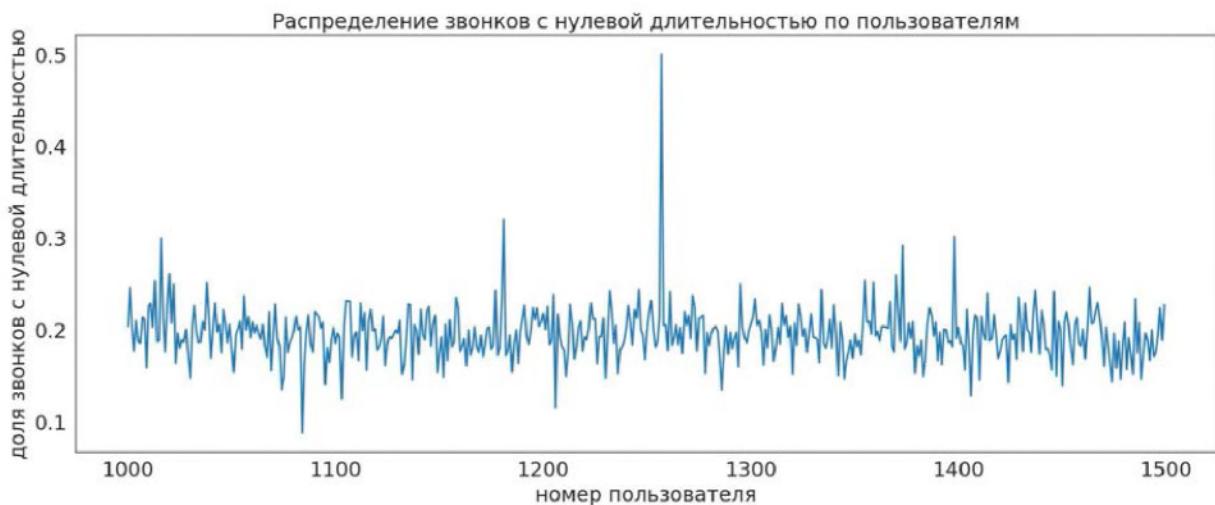
```
In [44]: fig, ax = plt.subplots(figsize=(16, 6))
temp.groupby('city').zero_duration.mean().plot.bar(alpha=0.75)
plt.xlabel('город')
ax.xaxis.set_tick_params(labelsize=10, rotation=90)
plt.ylabel('доля звонков с нулевой длительностью')
plt.title('Распределение звонков с нулевой длительностью по городам')
plt.show();
```



Бросим взгляд на распределение нулевых длительностей звонков по пользователям:

```
In [45]: fig, ax = plt.subplots(figsize=(16, 6))
temp.groupby('user_id').zero_duration.mean().plot()
plt.xlabel('номер пользователя')
ax.xaxis.set_tick_params(rotation=0)
plt.ylabel('доля звонков с нулевой длительностью')
```

```
plt.title('Распределение звонков с нулевой длительностью по пользователям')
plt.show();
```



Обнаруживается лишь один выброс, если смотреть на каждого пользователя в отдельности. Габриэлю Жданову сложно дозвониться до знакомых:

```
In [46]: unlucky_user_id = temp.groupby('user_id').zero_duration.mean().idxmax()
```

```
In [47]: table_users[table_users.user_id == unlucky_user_id]
```

```
Out[47]:   user_id  last_name  first_name  tariff  reg_date  churn_date          city  age
257      1257    Жданов     Габриель   ultra  2018-06-17  2019-01-01  Санкт-Петербург   21
```

Но и звонков за весь период исследования он сделал всего-лишь 14 - ничтожно мало, чтобы нам обращать внимание на его высокую долю соединений с нулевой длительностью. Пусть об этом подумают ([спойлер](#)) компетентные органы - взрослый парень делает в месяц 2-3 звонка, половина из них неудачных, а абонентскую плату вносит регулярно. Прошу прощения, Габриэль, за мою возможную мнительность, если Вам просто оплачивает телефон Ваша компания.

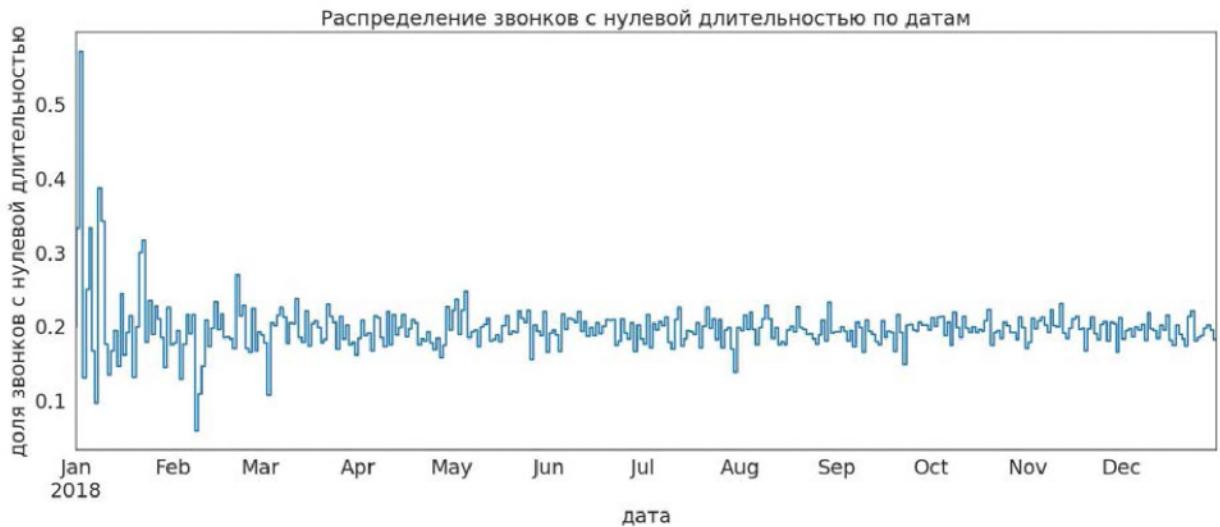
```
In [48]: table_calls[table_calls.user_id == unlucky_user_id]
```

```
Out[48]:      id  user_id  call_date  duration
100603  1257_0    1257  2018-08-13      0.00
100604  1257_1    1257  2018-11-28      0.00
100605  1257_2    1257  2018-10-02     14.16
100606  1257_3    1257  2018-06-26     18.49
100607  1257_4    1257  2018-12-19      0.91
...
100612  1257_9    1257  2018-09-21      0.00
100613  1257_10   1257  2018-06-20      0.00
100614  1257_11   1257  2018-08-12      0.00
100615  1257_12   1257  2018-12-22      7.03
100616  1257_13   1257  2018-11-07     5.42
```

14 rows × 4 columns

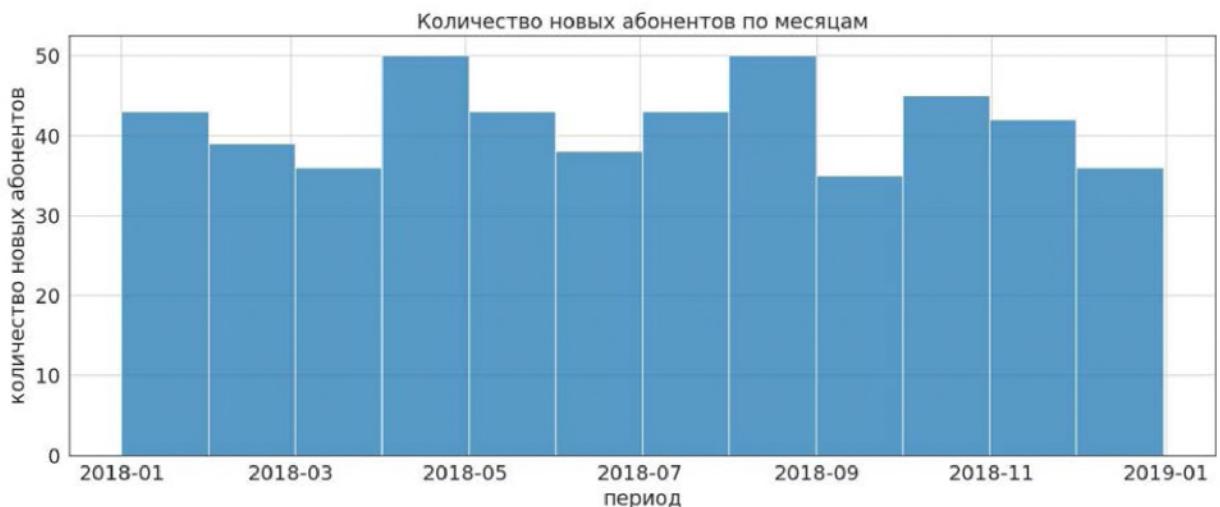
Может быть есть связь нулей в длительности с датой? Давайте проверим:

```
In [49]: fig, ax = plt.subplots(figsize=(16, 6))
temp.groupby('call_date').zero_duration.mean().plot(drawstyle='steps')
plt.xlabel('дата')
ax.xaxis.set_tick_params(rotation=0)
plt.ylabel('доля звонков с нулевой длительностью')
plt.title('Распределение звонков с нулевой длительностью по датам')
plt.show();
```

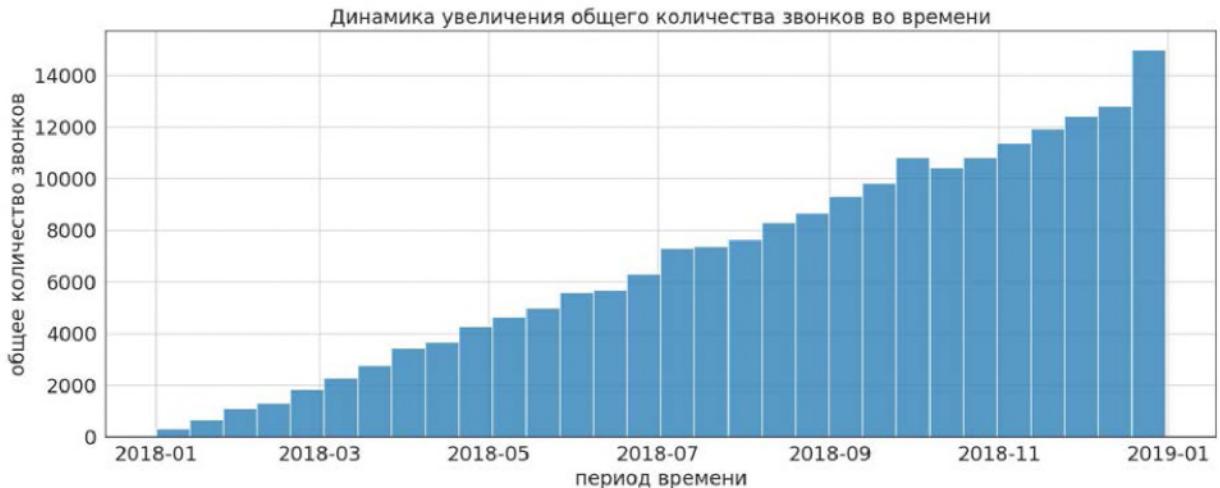


Вот, кажется появилась зацепка в начале 2018 года! Но посмотрев на количество звонков за периоды, отбрасываем и её - резкие колебания доли звонков с нулевой длительностью в начале года связаны не только с праздниками, но и с относительно малым количеством абонентов. В нашем учебном проекте компания заключала новые договоры на обслуживание на всём протяжении 2018 года. Постоянный рост количества абонентов приносил линейное увеличение общего количества звонков в единицу времени. Выборка отражает и эту тенденцию:

```
In [50]: fig, ax = plt.subplots(figsize=(16, 6))
table_users['reg_date'].hist(bins=12, alpha=0.75)
plt.xlabel('период')
ax.xaxis.set_tick_params(rotation=0)
plt.ylabel('количество новых абонентов')
plt.title('Количество новых абонентов по месяцам')
plt.show();
```



```
In [51]: fig, ax = plt.subplots(figsize=(16, 6))
temp['call_date'].hist(bins=30, alpha=0.75)
plt.xlabel('период времени')
ax.xaxis.set_tick_params(rotation=0)
plt.ylabel('общее количество звонков')
plt.title('Динамика увеличения общего количества звонков во времени')
plt.show();
```



Таким образом, записи с нулевой длительностью звонка случайно и равномерно распределены по всем категориям пользователей и по времени, и по месту регистрации, и по тарифу, и, поверьте на слово (проверено), по возрасту и сроку действия тарифного плана. Скорее всего причина кроется в пропущенных вызовах и нахождении абонентов вне зоны действия сети. Даже если это не так, в нашем учебном проекте оператор связи не взимает плату за установление соединения, т.е. нулевая длительность не тарифицируется. Так как в задачу исследования входит оценка прибыли, на которую не влияют неудавшиеся попытки соединения, мы можем смело их удалить. Как было установлено выше, выборка не перестанет быть репрезентативной, а результат исследования не ухудшится.

Удалим 39613 (19.55%) записей с нулевой длительностью звонка в минутах:

```
In [52]: before = table_calls.shape[0]
table_calls.drop(table_calls[table_calls.duration <= 0].index, inplace=True)
print('Удалено записей с нулевой длительностью звонка:', before - table_calls.shape[0])
```

Удалено записей с нулевой длительностью звонка: 39613

Обработка нулевых интернет-сессий

По аналогии с [обработкой звонков](#) с нулевой длительностью **удалим 19598 (13.12%) записей с пустыми объёмами интернет-трафика:**

```
In [53]: before = table_internet.shape[0]
table_internet.drop(table_internet[table_internet.mb_used <= 0].index, inplace=True)
print('Удалено записей с нулевым объемом интернет-трафика:', before - table_internet.shape[0])
```

Удалено записей с нулевым объемом интернет-трафика: 19598

Округление длительности звонков и приведение типа к целому

Автоматизированная проверка [обнаружила](#), что в **161447 (79.68%) записях с информацией о звонках** указана длительность звонка в минутах без округленная по

правилам сотового оператора. Вероятно, нами были получены данные не из финансового подразделения, а от поставщика связи, который ведет лишь учет биллинга и округлением не занимается (это делают другие [эксперты](#)). Округлим обнаруженные значения вверх до ближайшего целого и изменим тип столбца с [вещественного](#) на целый:

```
In [54]: table_calls['duration'] = table_calls['duration'].apply(np.ceil).astype(np.int64)
```

Округление интернет-трафика и приведение типа к целому

Автоматизированная проверка [обнаружила 128462 \(85.99%\) записи без округления объёма потраченного за сессию интернет-трафика \(в мегабайтах\)](#). Округлим обнаруженные значения вверх до ближайшего целого и изменим тип столбца с [вещественного](#) на целый:

```
In [55]: table_internet['mb_used'] = table_internet['mb_used'].apply(np.ceil).astype(np.int64)
```

Контрольная автоматизированная проверка

Вызовем повторно функцию проверки и автоматического преобразования типов столбцов наших таблиц:

```
In [56]: do_for_each_data_file(lambda f: f.check_data_frame_fields(check_try_to_fix=True))
```

Протокол автоматизированной проверки типов полей набора данных "Информация о тарифах":

1. Тип **int64** поля "**rub_monthly_fee**" (ежемесячная абонентская плата в рублях) не соответствует заявленному в описании **float64**.

Тип поля "**rub_monthly_fee**" (ежемесячная абонентская плата в рублях) изменен на **float64**.

2. Тип **int64** поля "**rub_per_minute**" (стоимость минуты разговора сверх тарифного пакета) не соответствует заявленному в описании **float64**.

Тип поля "**rub_per_minute**" (стоимость минуты разговора сверх тарифного пакета) изменен на **float64**.

3. Тип **int64** поля "**rub_per_message**" (стоимость отправки сообщения сверх тарифного пакета) не соответствует заявленному в описании **float64**.

Тип поля "**rub_per_message**" (стоимость отправки сообщения сверх тарифного пакета) изменен на **float64**.

4. Тип **int64** поля "**rub_per_gb**" (стоимость дополнительного гигабайта интернет-трафика сверх тарифного пакета) не соответствует заявленному в описании **float64**.

Тип поля "**rub_per_gb**" (стоимость дополнительного гигабайта интернет-трафика сверх тарифного пакета) изменен на **float64**.

Протокол автоматизированной проверки типов полей набора данных "Информация о пользователях":

В наборе данных "**Информация о пользователях**" ошибок в типах не обнаружено.

Протокол автоматизированной проверки типов полей набора данных "Информация о звонках":

В наборе данных "**Информация о звонках**" ошибок в типах не обнаружено.

Протокол автоматизированной проверки типов полей набора данных "Информация о сообщениях":

В наборе данных "Информация о сообщениях" ошибок в типах не обнаружено.

Протокол автоматизированной проверки типов полей набора данных "Информация об интернет-сессиях":

В наборе данных "Информация об интернет-сессиях" ошибок в типах не обнаружено.

Повторно запустим функцию автоматизированной проверки значений полей. По идее, у неё больше не должно возникнуть вопросов к данным. Убедимся, что все ошибки устранены:

```
In [57]: do_for_each_data_file(lambda f: f.check_data_frame_values(check_fix=True, prefix='te
```

Протокол автоматизированной проверки значений набора данных "Информация о тарифах":

В наборе данных "Информация о тарифах" ошибок в значениях не обнаружено.

Протокол автоматизированной проверки значений набора данных "Информация о пользователях":

В наборе данных "Информация о пользователях" ошибок в значениях не обнаружено.

Протокол автоматизированной проверки значений набора данных "Информация о звонках":

В наборе данных "Информация о звонках" ошибок в значениях не обнаружено.

Протокол автоматизированной проверки значений набора данных "Информация о сообщениях":

В наборе данных "Информация о сообщениях" ошибок в значениях не обнаружено.

Протокол автоматизированной проверки значений набора данных "Информация об интернет-сессиях":

В наборе данных "Информация об интернет-сессиях" ошибок в значениях не обнаружено.

```
In [58]: temp['days_since_reg'] = (temp['call_date'] - temp['reg_date']) / np.timedelta64(1, '
```

Преобразование названия тарифного плана в категориальную переменную

Операции объединения наборов данных Pandas (и не только), содержащих большое количество значений строкового типа, может происходить достаточно долго и расходовать память неэффективно из-за появления дубликатов. Возможным вариантом решения этой проблемы является использование **категориальных переменных**.

Преобразуем название тарифного плана в категориальную переменную средствами `pandas.CategoricalDtype`.

Опишем новый тип данных:

```
In [59]: tariff_dtype = pd.CategoricalDtype(['Смарт', 'Ультра'], ordered=False)
```

Опишем порядок преобразования названий тарифов и их перевода на русский язык:

```
In [60]: def convert_tariff(field):
    return field.map({'smart': 'Смарт', 'ultra': 'Ультра'}).astype(tariff_dtype)
```

Исправим таблицу тарифов:

```
In [61]: table_tariffs['tariff_name'] = convert_tariff(table_tariffs['tariff_name'])
```

```
In [62]: display(table_tariffs.tariff_name.dtype)
```

```
CategoricalDtype(categories=['Смарт', 'Ультра'], ordered=False)
```

Исправим таблицу пользователей:

```
In [63]: table_users['tariff'] = convert_tariff(table_users['tariff'])
```

```
In [64]: display(table_users.tariff.dtype)
```

```
CategoricalDtype(categories=['Смарт', 'Ультра'], ordered=False)
```

Преобразование названия города в категориальную переменную

Опишем новый тип данных:

```
In [65]: city_dtype = pd.CategoricalDtype(sorted(list(table_users.city.unique())), ordered=True)
```

Исправим таблицу пользователей:

```
In [66]: table_users['city'] = table_users['city'].astype(city_dtype)
```

```
In [67]: display(table_users.city.dtype)
```

```
CategoricalDtype(categories=['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', 'Петrozаводск',
    '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'],
    ordered=True)
```

Оценка количества полных дубликатов

```
In [68]: do_for_each_data_file(lambda f: print(f.name, '-', f.df.duplicated().sum()),
    'Количество полных дубликатов в таблицах')
```

Количество полных дубликатов в таблицах

Информация о тарифах - 0
Информация о пользователях - 0
Информация о звонках - 0
Информация о сообщениях - 0
Информация об интернет-сессиях - 0

Полных дубликатов в нашем наборе данных нет.

Теперь освободим память, занятую временным набором данных:

```
In [69]: del temp
```

Ориентация во времени

Перед тем как приступить к вычислению стоимостей оказанных услуг для каждого пользователя сориентируемся во времени.

Нам необходимо посчитать для каждого пользователя:

- количество сделанных звонков и израсходованных минут разговора **по месяцам**;
- количество отправленных сообщений **по месяцам**;
- объем израсходованного интернет-трафика **по месяцам**;
- **помесечную** выручку с каждого пользователя.

Ключевое значение имеет понятие **месяца**. Мы знаем, что сотовый оператор округляет минуты и мегабайты вверх. А как она относится к абонентам, заключившим контракт в разные дни месяца. Надеюсь, она не взимает с несчастных новых клиентов, подключившихся в конце месяца, абонентскую плату за предыдущие дни. Вопрос можно и так поставить: для каждого клиента месяц начинается в день заключения договора на обслуживание? При учете этого фактора исследование будет точнее, не будет зависеть от распределения пользователей по дню выставления счета. Также проще будет формулировать гипотезы.

Важность вопроса подтверждается и копией информационного сообщения с сайта очень похожего на Мегалайн оператора:

- В первый месяц абонентская плата списывается в полном размере сразу после подключения / перехода на тариф и предоставляется 100% трафика.
- Далее абонентская плата списывается через календарный месяц после даты подключения/ перехода на тариф в то же время, что и первое списание АП (минута в минуту).
- Если при подключении / переходе на тариф у абонента на балансе недостаточно средств для списания АП, то смена ТП не происходит, АП не списывается, приходит нотификация о недостаточности средств для перехода.
- Если абонент подключился/ перешел на тариф 29, 30, 31 числа, а в новом календарном периоде нет такой даты, то АП списывается в последний день месяца. В следующий месяц списание происходит в дату подключения.



Если не учитывать разницу между началом календарного месяца и отчетного месяца по биллингу для пользователей с абонентской платой и лимитами, мы можем обмануть

уважаемый Мегалайн не меньше, чем он своих клиентов 😊. Достаточно взглянуть на следующую диаграмму:

```
In [70]: fig, ax = plt.subplots(figsize=(16, 6))
table_users.reg_date.dt.day.value_counts(normalize=True, sort=False) \
    .plot.bar(label='доля абонентов', alpha=0.75)
plt.xlabel('день месяца для подведения баланса (день заключения договора)')
ax.xaxis.set_tick_params(rotation=0)
plt.ylabel('доля абонентов')
plt.title('Распределение абонентов по дням подведения баланса')
plt.legend()
plt.show()
```



Как видим, в нашем проекте пользователи приходят в офисы сотовой компании совсем не только по первым числам месяца. Значит считать выручку по отдельному клиенту, ориентируясь на календарный месяц нельзя. **Мы должны брать за точку отсчета день подключения тарифа.** Это несколько усложнит алгоритмизацию, но позволит не упасть в грязь лицом перед заказчиком.

Поступим просто - для каждого пользователя определим виртуальное время, которое будет отсчитываться с момента активации его тарифного плана. Соответствующие данные хранятся в поле `reg_date` - дата подключения тарифа (день, месяц, год).

В дальнейшем все статистики пользователя **за месяц** будем рассчитывать, опираясь на начало его персональной "эпохи".

Путешествие во времени

Опишем алгоритм перевода календарной даты в виртуальное время абонента сотовой связи:

- удаляем все записи, у которых дата выходит за пределы контракта на обслуживание;
- для каждой даты оказания услуги (звонок, отправка сообщения, выход в Интернет) вычисляем количество полных календарных месяцев, прошедших с даты активации тарифа - даты первого списания денежных средств и включения счетчиков, подключенных к тарифу услуг;
- прибавляем 1 для перевода номера месяца в привычный для нас вид, при котором нумерация начинается с единицы.

В результате всем действиям пользователя будут сопоставлены номера отчетных периодов (в нашем случае месяцев). В дальнейшем, группируя записи по этим номерам, получаем

ежемесячные статистики для каждого пользователя.

Опишем соответствующую функцию:

```
In [71]: from pandas.tseries.offsets import DateOffset
def get_billing(table_users, table_tariffs, table_calls, table_messages, table_inter
    """Возвращает датафрейм с информацией о биллинге каждого клиента
    по месяцам, начиная с даты заключения договора на обслуживание"""

    def month_diff(a, b):
        """Возвращает разницу между датами a и b в месяцах"""
        return 12 * (a.dt.year - b.dt.year) + (a.dt.month - b.dt.month)

    def add_months(a, d):
        """добавляет d месяцев к дате a"""
        return a + DateOffset(months=d)

    def start_time_machine(df, date_field, agg_field, agg_func, new_field=None, as_i
        """Вспомогательная функция.
        Приводит таблицу информации о действиях абонента (df) к виду, удобному для
        подсчета стоимости оказанных услуг, применяя к полю agg_field
        функцию agg_func и помещая результат в поле new_field"""

        nonlocal table_users

        # присоединяем к таблице информации о действиях абонента сроки его контракта
        temp = df.join(table_users[['reg_date', 'churn_date']], on='user_id', lsuffix

        # удаляем действия с датами за пределами контракта
        # в нашем массиве их нет (проверено на этапе предобработки)
        # реализуем, когда появятся (не терпится уже биллинг получить :-)
        pass

        #####
        # если раскомментировать этот код, то задача будет решена без учета
        # даты выставления счета (поверьте, результат проекта изменится в
        # худшую сторону - упадут сборы за услуги свыше лимитов по тарифу :-)
        # Мегалайн потеряет деньги!
        # temp['reg_date'] = pd.to_datetime('2018-01-01')
        #####
        # совершают скачок во времени - вычисляем по дате операции номер месяца
        # с момента заключения контракта абонентом
        temp['month'] = month_diff(temp[date_field], temp['reg_date']) + 1

        # группируем операции по пользователю и месяцу,
        # применяя к полю agg_field функцию agg_func
        # функцию agg_func и помещая результат в поле new_field
        temp = temp.groupby(['user_id', 'month'], as_index=as_index).agg({agg_field:

            # помещаем результат групповой операции в поле new_field
            if new_field:
                temp.columns = [new_field]

            # возвращаем результат
            return temp

    def stop_time_machine(billing):
        """Вспомогательная функция.
        Считает итоговые денежные суммы по данным биллинга и информации о тарифах"""

        nonlocal table_users, table_tariffs, table_calls, table_messages, table_inter
        # сбрасываем индекс таблицы биллинга
```

```

billing.reset_index(inplace=True)
# индексируем биллинг по идентификатору пользователя
billing.set_index('user_id', inplace=True)

# присоединяем к биллингу столбец с названием тарифа, даты его активации и т.д.
billing['tariff'] = table_users['tariff']
billing['reg_date'] = table_users['reg_date']
billing['city'] = table_users['city']

#####
# если раскомментировать этот код, то задача будет решена без учета
# даты выставления счета (проверьте, результат проекта изменится в
# худшую сторону - упадут сборы за услуги свыше лимитов по тарифу :-( )
# Мегалайн потеряет деньги!
# billing['reg_date'] = pd.to_datetime('2018-01-01')
#####

# вычисляем даты отчетного периода в один месяц
billing['date_from'] = billing[['reg_date', 'month']]. \
    apply(lambda x: add_months(x[0], x[1]-1), axis=1, r
billing['date_to'] = billing[['reg_date', 'month']]. \
    apply(lambda x: add_months(x[0], x[1]), axis=1, r
DateOffset(days=1)

# сбрасываем текущие индексы у биллинга и тарифов
billing.reset_index(inplace=True)
billing.set_index('tariff', inplace=True)
billing['minutes_included'] = table_tariffs['minutes_included']
billing['messages_included'] = table_tariffs['messages_included']
billing['mb_per_month_included'] = table_tariffs['mb_per_month_included']
billing['rub_per_minute'] = table_tariffs['rub_per_minute']
billing['rub_per_message'] = table_tariffs['rub_per_message']
billing['rub_per_gb'] = table_tariffs['rub_per_gb']
billing['rub_monthly_fee'] = table_tariffs['rub_monthly_fee']

# сбрасываем индекс таблицы биллинга
billing.reset_index(inplace=True)
# индексируем биллинг по идентификатору пользователя и месяцу
# для дальнейшего расчета ежемесячных статистик
billing.set_index(['user_id', 'month'], inplace=True)

# преобразуем вещественные типы к целым
for f in ['calls_total', 'minutes_total', 'messages_total', 'mb_total']:
    billing[f] = billing[f].astype(np.int64)

# определяем объемы услуг, входящих в пакеты тарифа
billing['minutes_free'] = billing['minutes_total'].clip(upper=billing['minu
billing['messages_free'] = billing['messages_total'].clip(upper=billing['mes
billing['mb_free'] = billing['mb_total'].clip(upper=billing['mb_per_mo

# вычитаем бесплатные услуги из суммарного количества звонков, сообщений и т.д.
# для получения объемов, подлежащих дополнительной оплате
billing['minutes_charged'] = billing['minutes_total'] - billing['minutes_f
billing['messages_charged'] = billing['messages_total'] - billing['messages_f
billing['mb_charged'] = billing['mb_total'] - billing['mb_free']

# вычисляем стоимости услуг сверх бесплатных лимитов тарифного плана, округл
billing['pay_minutes'] = (billing['minutes_charged'] * billing['rub_per_mi
billing['pay_messages'] = (billing['messages_charged'] * billing['rub_per_
billing['pay_internet'] = (billing['mb_charged']) / 1024 * billing['rub_per_

```

```

        billing['pay_extra']      = billing['pay_minutes'] + \
                                billing['pay_messages'] + \
                                billing['pay_internet']

    # Вычисляем итоговую сумму
    billing['pay_total'] = billing['rub_monthly_fee'] + \
                           billing['pay_minutes'] + \
                           billing['pay_messages'] + \
                           billing['pay_internet']

    # переведем мегабайты в гигабайты
    billing['gb_total'] = billing['mb_total'] / 1024

    # Возвращаем датасет с полями в удобном для просмотра порядке
    return billing[['calls_total', 'minutes_total', 'minutes_included', 'minutes',
                    'minutes_charged', 'rub_per_minute', 'pay_minutes',
                    'messages_total', 'messages_included', 'messages_free',
                    'messages_charged', 'rub_per_message', 'pay_messages',
                    'mb_total', 'gb_total', 'mb_per_month_included', 'mb_free',
                    'mb_charged', 'rub_per_gb', 'pay_internet',
                    'rub_monthly_fee', 'pay_extra', 'pay_total',
                    'date_from', 'date_to', 'reg_date',
                    'tariff', 'city']]

    # сбрасываем индекс таблицы пользователей
    table_users.reset_index(inplace=True)
    # индексируем пользователей по идентификатору
    table_users.set_index('user_id', inplace=True)

    # считаем количество звонков пользователя по месяцам
    b1 = start_time_machine(table_calls, 'call_date', 'duration', 'count', 'calls_to')

    # считаем суммы длительностей звонков пользователя по месяцам
    b2 = start_time_machine(table_calls, 'call_date', 'duration', 'sum', 'minutes_to')

    # считаем количество сообщений пользователя по месяцам
    b3 = start_time_machine(table_messages, 'message_date', 'message_date', 'count', 'messages_to')

    # считаем суммы интернет-трафика пользователя по месяцам
    b4 = start_time_machine(table_internet, 'session_date', 'mb_used', 'sum', 'mb_to')

    # объединяем полученные значения, заполняя пропуски нулями
    billing = b1.join(b2, how='outer').join(b3, how='outer').join(b4, how='outer').fillna(0)

    # возвращаемся из путешествия во времени :-)
    # считаем итоговые денежные суммы за каждый месяц
    billing = stop_time_machine(billing)

    # сбросим индекс у таблицы пользователей
    if table_users.index.name == 'user_id':
        table_users.reset_index(inplace=True)

return billing

```

Получение биллинга

Воспользуемся разработанной выше функцией для получения биллинга для всех пользователей по месяцам:

```
In [72]: billing = get_billing(table_users, table_tariffs, table_calls, table_messages, table_internet)
```

```
In [73]: display(billing)
```

user_id	month																	
1000	1	17	159	3000	159	...	2018-06-24	2018-05-25	Ул									
	2	28	172	3000	172	...	2018-07-24	2018-05-25	Ул									
	3	41	340	3000	340	...	2018-08-24	2018-05-25	Ул									
	4	42	408	3000	408	...	2018-09-24	2018-05-25	Ул									
	5	46	466	3000	466	...	2018-10-24	2018-05-25	Ул									
...									
1498	4	30	247	500	247	...	2018-11-18	2018-07-19	Си									
1499	1	8	70	500	70	...	2018-10-26	2018-09-27	Си									
	2	44	449	500	449	...	2018-11-26	2018-09-27	Си									
	3	62	612	500	500	...	2018-12-26	2018-09-27	Си									
	4	56	492	500	492	...	2019-01-26	2018-09-27	Си									

3214 rows × 28 columns



Получили объемную, но не менее удобную, сводную таблицу в следующем формате:

Имя столбца	Описание столбца
Информация о пользователе и периоде времени	
user_id	уникальный идентификатор пользователя
reg_date	дата активации тарифного плана
month	номер месяца с момента заключения абонентом договора на обслуживание
date_from	дата начала отчетного периода
date_to	дата завершения отчетного периода (1 месяц с даты начала)
Информация о тарифе	
tariff	название тарифа
city	город пользователя
Информация о звонках	
calls_total	количество телефонных звонков, совершенных в отчетный месяц
minutes_included	максимальное количество минут разговора в месяц, включённых в абонентскую плату по тарифу
minutes_total	общее количество минут разговора в месяц

Имя столбца	Описание столбца
minutes_free	количество минут разговора в месяц, предоставленных без дополнительной платы
minutes_charged	количество минут разговора в месяц, предоставленных за дополнительную плату
rub_per_minute	цена одной минуты разговора сверх тарифного пакета (например, если в тарифе 100 минут разговора в месяц, то со 101 минуты будет взиматься плата)
pay_minutes	стоимость минут разговора сверх тарифного пакета
Информация о сообщениях	
messages_included	максимальное количество сообщений в месяц, включённых в абонентскую плату
messages_total	общее количество сообщений в месяц
messages_free	количество сообщений в месяц, отправленных без дополнительной платы
messages_charged	количество сообщений в месяц, отправленных за дополнительную плату
rub_per_message	цена отправки одного сообщения сверх тарифного пакета
pay_messages	стоимость сообщений сверх тарифного пакета
Информация о пользовании сетью Интернет	
mb_total	общий объём интернет-трафика (в мегабайтах)
gb_total	общий объём интернет-трафика (в гигабайтах)
mb_per_month_included	максимальный объём интернет-трафика, включённого в абонентскую плату (в мегабайтах)
mb_free	объём интернет-трафика, предоставленный без дополнительной платы (в мегабайтах)
mb_charged	объём интернет-трафика, предоставленный за дополнительную плату (в мегабайтах)
rub_per_gb	цена дополнительного гигабайта интернет-трафика сверх тарифного пакета (1 гигабайт = 1024 мегабайта)
pay_internet	стоимость интернет-трафика сверх тарифного пакета
Итоговые суммы	
rub_monthly_fee	ежемесячная абонентская плата в рублях
pay_extra	сумма дополнительных услуг, не включенных в пакеты тарифа
pay_total	общая сумма к оплате

Проверка биллинга

Можем ли мы доверять полученному биллингу? Думаю, что нет. Перефразируя известную цитату: "И запомните, что верить в наше время нельзя никому, порой даже самому себе." [Автору](#) работы — можно.

Убедимся в этом. Сравним итоговые суммы биллинга со статистикой по исходным массивам:

```
In [74]: compare_dict = {
    'общего количества телефонных звонков':
```

```

{
    'по массиву': table_calls.shape[0],
    'по биллингу': billing.calls_total.sum()
},
'общей продолжительности телефонных звонков':
{
    'по массиву': table_calls.duration.sum(),
    'по биллингу': billing.minutes_total.sum(),
    'сумма платных и бесплатных': billing.minutes_free.sum() + billing.minutes_charg
},
'общего количества сообщений':
{
    'по массиву': table_messages.shape[0],
    'по биллингу': billing.messages_total.sum(),
    'сумма платных и бесплатных': billing.messages_free.sum() + billing.messages_cha
},
'общего объема интернет-трафика':
{
    'по массиву': table_internet.mb_used.sum(),
    'по биллингу': billing.mb_total.sum(),
    'сумма платных и бесплатных': billing.mb_free.sum() + billing.mb_charged.sum()
},
'суммы услуг, предоставленных за отдельную плату':
{
    'сумма по звонкам, сообщениям и интернет':
        (billing.pay_minutes.sum() + billing.pay_messages.sum() + billing.pay_intern
    'сумма по биллингу': billing.pay_extra.sum().round(2),
},
'общей суммы услуг':
{
    'сумма цены тарифа и платных услуг':
        (billing.rub_monthly_fee.sum() + billing.pay_extra.sum()).round(2),
    'сумма по биллингу': billing.pay_total.sum().round(2),
}
}

for test_name, data in compare_dict.items():
    display(HTML(f'Проверка {test_name}'))
    for var_name, var_value in data.items():
        print(f'{var_name}: {var_value}')

    if len(frozenset(data.values())) == 1:
        print(f'*52}ok')
    else:
        print(f'*48}ошибка')

```

Проверка общего количества телефонных звонков

по массиву:	162994
по биллингу:	162994
	ok

Проверка общей продолжительности телефонных звонков

по массиву:	1450301
по биллингу:	1450301
сумма платных и бесплатных:	1450301
	ok

Проверка общего количества сообщений

по массиву:	123036
по биллингу:	123036
сумма платных и бесплатных:	123036
	ok

Проверка общего объема интернет-трафика

по массиву:	55369459
по биллингу:	55369459
сумма платных и бесплатных:	55369459
	ok

Проверка суммы услуг, предоставленных за отдельную плату

сумма по звонкам, сообщениям и интернет: 1632057.89
сумма по биллингу: 1632057.89
ok

Проверка общей суммы услуг

сумма цены тарифа и платных услуг: 4778757.89
сумма по биллингу: 4778757.89
ok

Итоговые суммы по биллингу совпадают со значениями, полученными на основе предобработанного массива. Это дает нам основание полагать, что ошибки в биллинг не закрались.

Удаление оборванных месяцев

Выше мы уже вычислили период времени, за который нам предоставили информацию о действиях пользователей.

```
In [75]: display(HTML('В наборе данных представлены сведения за период '  
f'с <b>{date_min:{%d.%m.%Y}</b> по <b>{date_max:{%d.%m.%Y}</b>.'))
```

В наборе данных представлены сведения за период с **01.01.2018** по **31.12.2018**.

Не всю информацию из этого периода мы можем принимать во внимание для оценки усредненных статистик по месяцам. Например, если пользователь заключил контракт в 2017 году, мы должны отбросить его звонки в январе 2018 года до даты выставления счета из-за его неопределенности. Мы не знаем, как использовалась связь в последний месяц прошлого года. Как мы уже определили, абонентская плата часто списывается совсем не в первое число календарного месяца. Аналогично необходимо отбросить звонки (сообщения и интернет-сессии), счет по которым будет выставлен лишь в следующем 2019 году - за это время пользователь может успеть совершить множество действий сверх лимитов по тарифу.

Удивительно, но в нашем массиве данных нет звонков, сообщений и DDOS-атак из прошлого. Видимо, заботливые преподаватели Яндекс.Практикума не хотели поставить нас в неловкое положение 🤪:

```
In [76]: to_drop = (billing.date_to < '2018-01-15')  
to_drop.sum()
```

Out[76]: 0

Вот, например, господин Безруков из Тамбова заключил контракт 1 января 2018 года...

```
In [77]: table_users[table_users.user_id == table_calls.loc[table_calls.call_date.idxmin()].u
```

Out[77]:

	last_name	city	reg_date
193	Безруков	Тамбов	2018-01-01

193 Безруков Тамбов 2018-01-01

... и только после этого совершил первый звонок:

```
In [78]: print(table_calls[table_calls.user_id == table_calls.loc[table_calls.call_date.idxmi
```

2018-01-01

А вот оборванные счета, платить за которые придется в следующем году, в некотором количестве имеются:

```
In [79]: to_drop = (billing.date_to > '2019-01-15')
to_drop.sum()
```

```
Out[79]: 235
```

При принятии решения об их удалении мы [не будем полагаться на случай](#), а прежде оценим их долю:

```
In [80]: print(f'{to_drop.mean():.02%}')
```

```
7.31%
```

Как бы жалко не было терять данные, но брать во внимание только половину месячной активности пользователей в конце года мы не можем, ведь в задачу проекта входит как раз узучить среднемесячные величины.

```
In [81]: billing.shape
```

```
Out[81]: (3214, 28)
```

```
In [82]: billing.drop(billing[to_drop].index, inplace=True)
```

```
In [83]: billing.shape
```

```
Out[83]: (2979, 28)
```

Вывод по шагу 2

- Данные [приведены](#) к нужным типам:
 - длительности разговоров, количество сообщений и объемы интернет-трафика - целые;
 - цены - вещественные (в общем случае они могут содержать копейки);
 - для ускорения операций с большими массивами данных в категориальные переменные преобразованы [названия тарифного плана](#) и [названия городов](#).
- [Найдены](#) и [исправлены](#) ошибки в данных (пропуски в информации о сроках контрактов, [неокругленные величины](#) длительности звонка и объема интернет-трафика).
Оценено, как их отсутствие повлияет на результаты анализа. Принято [обоснованное решение](#) удалить эти записи.
- [Обнаружены](#) нулевые значения в длительностях звонков и объемах интернет-трафика. Оценено, как их отсутствие повлияет на результаты анализа. Принято [обоснованное решение](#) удалить эти записи.
- [Посчитаны](#) и [проверены](#) все статистические данные (билинг) об объемах оказанных услуг и их стоимости.
- Таким образом, получены новые данные, которые позволяют приступить к решению основной задачи проекта.

Шаг 3. Анализ данных

[Задание описано выше](#)

Инструменты анализа

Настало время для описания функции, которая поможет нам удобно визуализировать данные о поведении клиентов:

```
In [84]: # словарь для переименования полей результата работы функции describe
stats_dict = {'count': 'количество образцов', 'mean': 'среднее значение',
              'std': 'среднеквадратичное отклонение',
              'min': 'минимум', 'max': 'максимум'}
```

```
In [85]: def plot_billing(field_name, slice_names, force='', hist=False, min_x=0, max_x=None,
                    start_color=0, plot_sigma=True, fill=False, step=None, grid=True,
                    loc='upper right', figsize=(16, 6)):
    """Вспомогательная функция для вычисления характеристик распределения, вывода
    отчета и построения графиков зависимости от force значений среза биллинга
    по полю field_name и категорий пользователей slice_names"""
    # создаем полотно
    fig, ax = plt.subplots(figsize=figsize)

    # словарь для заполнения характеристиками распределений
    stat = dict()

    # задаем начальный номер цвета
    color = start_color

    # выполняем для всех заданных срезов
    for i, slice_name in enumerate(slice_names):
        # надрезаем биллинг
        data = billing[filters[slice_name]]

        # находим параметры распределения
        desc = data[field_name].describe()
        desc.loc['дисперсия'] = desc.loc['std'] ** 2
        stat[slice_name] = desc
        # считаем по выборке для перекрестной проверки
        x_mean = data[field_name].mean()
        sigma = data[field_name].std()

        # формируем подпись кривой
        label=f'{parameters[field_name]} ({slice_name})\n${mu=${x_mean:.02f}}, ${si
        if hist:
            # чертим гистограмму индивидуальным цветом
            sns.histplot(data=data, x=field_name, ax=ax, fill=fill,
                          label=label, color=f'C{color}', lw=2, kde=True)
        else:
            # чертим график индивидуальным цветом
            sns.kdeplot(data=data, x=field_name, ax=ax, fill=fill,
                         label=label, color=f'C{color}', lw=2)
        # выбираем следующий цвет, если рисуем всего один график
        color += len(slice_names) == 1
        # чертим среднее значение
        ax.axvline(x_mean, color=f'C{color}', ls='--', lw=1.5,
                   label=f'среднее: ${mu=${x_mean:.02f}}' if i == 0 else None)
        # выбираем следующий цвет, если рисуем всего один график
        color += len(slice_names) == 1
        # чертим медиану
        x = data[field_name].median()
        ax.axvline(x, color=f'C{color}', ls='-.', lw=1.5,
                   label=f'медиана: {x:.02f}' if i == 0 else None)
        # для первого среза чертим линии трех сигм
        if plot_sigma and i == 0:
            # выбираем следующий цвет, если рисуем всего один график
            color += len(slice_names) == 1
            for j in (-3, -2, -1, 1, 2, 3):
                dx = j * sigma
                x = x_mean + dx
```

```

        if j >= 1:
            interval = f'\sigma=[{max(0, x_mean-dx):.02f}, {x:.02f}]'
            if j == 1:
                label = f'$\mu\pm{interval}'
            else:
                label = f'$\mu\pm{j}\cdot{interval}$
            else:
                label = None
            ax.axvline(x, color=f'C{color}', ls=':', lw=1.5*abs(j), label=label)
        # выбираем следующий цвет
        color += 1

    # настраиваем ось x
    ax.set_xlim(min_x, max_x)
    if step:
        ax.xaxis.set_major_locator(ticker.MultipleLocator(step))

    # настраиваем общий вид чертежа
    plt.xlabel(f'{parameters[field_name]} в месяц для одного абонента')
    if hist:
        plt.ylabel('количество наблюдений')
    else:
        plt.ylabel('плотность вероятности')
    plt.title(f'Как распределяется {parameters[field_name]}'{force})
    plt.grid(grid)

    # выводим подписи к графикам
    handles, labels = ax.get_legend_handles_labels()
    if labels[0].startswith('среднее'):
        labels = labels[5:6] + labels[0:5] + labels[6:]
        handles = handles[5:6] + handles[0:5] + handles[6:]
    plt.legend(handles, labels, loc='best', framealpha=1)

    # выводим параметры распределения в удобном виде
    stat = pd.DataFrame(stat).rename(index=stats_dict)
    display(HTML(f'{  
Изучим, как распределяется {parameters[field_name]}'{force}}  
'))
    display(stat)

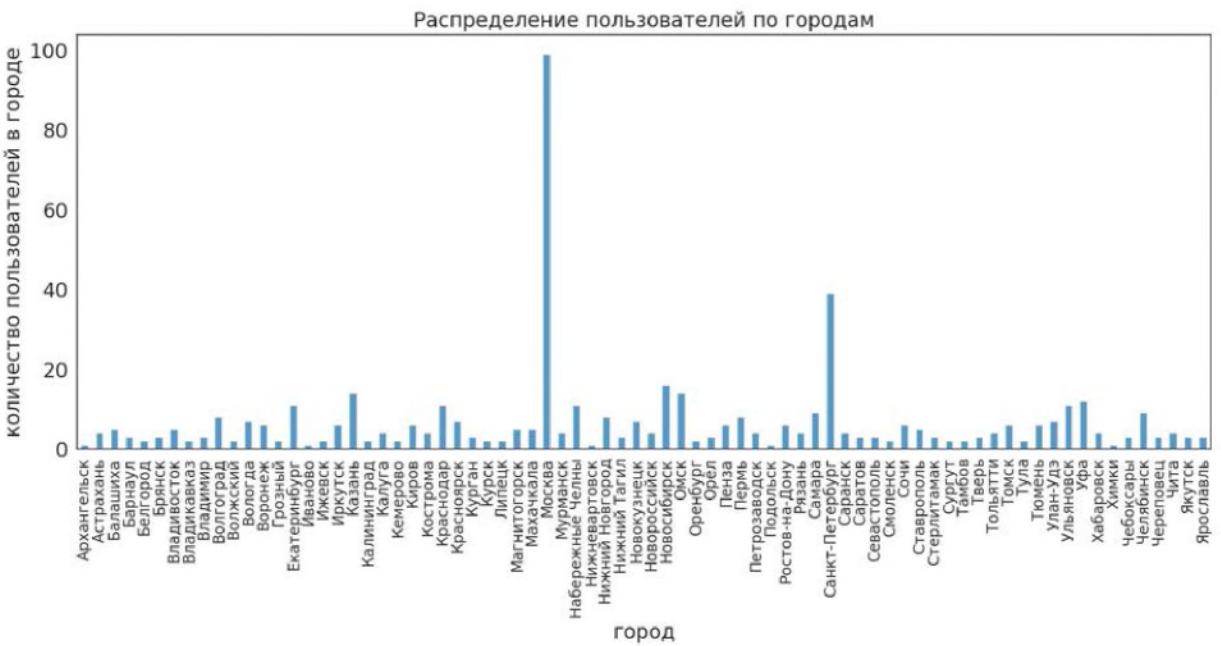
    # отображаем чертеж
    plt.show();

```

Ареал обитания клиентов и релевантность выборки

Из телевизионных передач о живых организмах мы знаем, что на их повадки большое влияние оказывает ареал обитания. Начнем изучение поведения клиентов именно с этого:

```
In [86]: fig, ax = plt.subplots(figsize=(16, 6))
table_users.groupby('city').user_id.count().plot.bar(alpha=0.75, ax=ax)
plt.xlabel('город')
ax.xaxis.set_tick_params(labelsize=12, rotation=90)
plt.ylabel('количество пользователей в городе')
plt.title('Распределение пользователей по городам')
plt.show();
```



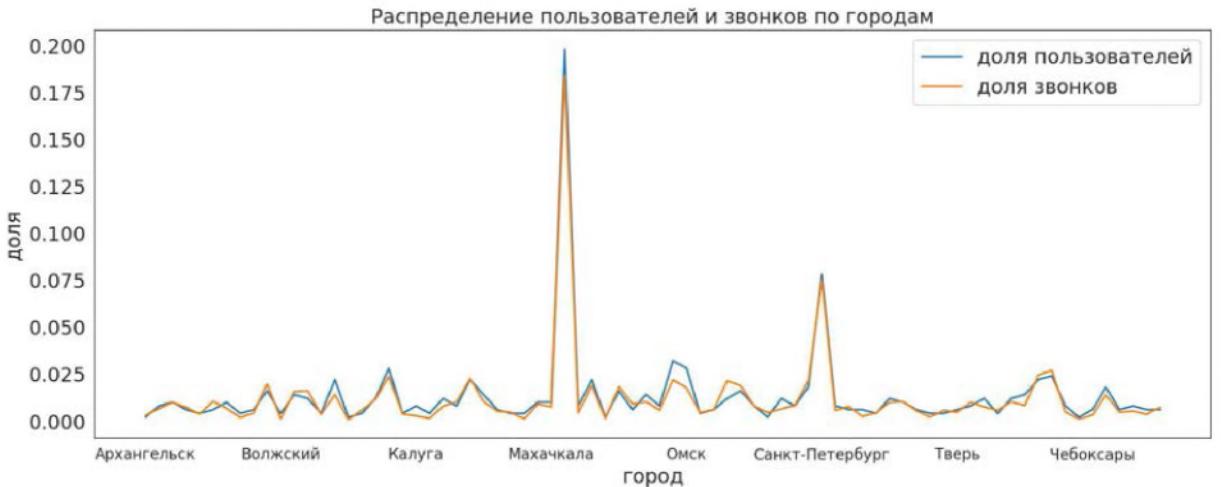
Как и следовало ожидать, большинство абонентов проживает в крупных городах. Это говорит о высокой релевантности выборки, которую предоставила нам сотовая компания. А как обстоят дела со звонками? Сравним доли пользователей по городам и доли звонков от них из общей массы:

```
In [87]: fig, ax = plt.subplots(figsize=(16, 6))

data_users = table_users.groupby('city')[['user_id']].count() / table_users.shape[0]
data_calls = billing.groupby('city').calls_total.sum() / table_calls.shape[0]
data = data_users.join(data_calls, how='outer')

data.plot(ax=ax)

plt.xlabel('город')
ax.xaxis.set_tick_params(labelsize=12, rotation=0)
plt.ylabel('доля')
plt.title('Распределение пользователей и звонков по городам')
plt.legend(['доля пользователей', 'доля звонков'])
plt.show();
```



Сомнений не осталось: выборка релевантна - доли пользователей по городам коррелируют с долями звонков. Доверять можно не только автору работы, но и поставщику данных.

Критерии выделения выборок для сравнения

```
In [88]: # сбросим индекс биллинга  
billing.reset_index(inplace=True)
```

Опишем критерии отбора данных для сравнения и присвоим им символические имена для дальнейшего использования при построении графиков:

```
In [89]: filters = {  
    'все пользователи': billing.month == billing.month,  
  
    'тариф "Смарт)": billing.tariff == 'Смарт',  
    'тариф "Ультра": billing.tariff == 'Ультра',  
  
    'Москва': billing.city == 'Москва',  
  
    'первый месяц': billing.month == 1,  
    'второй месяц': billing.month == 2,  
    'опытные пользователи': billing.month > 2  
}  
  
filters['не Москва'] = ~filters['Москва']  
  
filters['"Смарт" в Москве'] = filters['тариф "Смарт"] & filters['Москва']  
filters['"Смарт" не в Москве'] = filters['тариф "Смарт"] & filters['не Москва']  
  
filters['"Ультра" в Москве'] = filters['тариф "Ультра"] & filters['Москва']  
filters['"Ультра" не в Москве'] = filters['тариф "Ультра"] & filters['не Москва']  
  
filters['первый месяц со Смартом'] = filters['первый месяц'] & filters['тариф "Смарт']  
filters['второй месяц со Смартом'] = filters['второй месяц'] & filters['тариф "Смарт']  
filters['опытные со Смартом'] = filters['опытные пользователи'] & filters['тариф "Смарт']  
  
filters['первый месяц с Ультра'] = filters['первый месяц'] & filters['тариф "Ультра"]  
filters['второй месяц с Ультра'] = filters['второй месяц'] & filters['тариф "Ультра"]  
filters['опытные с Ультра'] = filters['опытные пользователи'] & filters['тариф "Ультра"]
```

```
In [90]: def billing_slice(slice_name):  
    """Возвращает срез биллинга по его имени"""  
    return billing(filters[slice_name])
```

Теперь нам будет легко разрезать биллинг на куски. Если слышим фразу "Как там сборы по Смарту в регионах?", выполняем следующий код, а сэкономленное время тратим на самосовершенствование:

```
In [91]: billing_slice('"Смарт" не в Москве')[['user_id', 'city', 'pay_total', 'pay_extra']]  
        .groupby('city').agg(users=('user_id', 'nunique'), pay_total=('pay_total', 'sum'))
```

```
Out[91]:      users  pay_total  pay_extra  
          city  
-----  
Архангельск     1   13537.12    7487.12  
Астрахань       3   26423.91   14323.91  
Балашиха       4   23734.38   12184.38  
Барнаул        2   4089.53    789.53  
Белгород        2   10769.59   2519.59  
...             ...      ...      ...  
Челябинск       5   37388.79   22538.79
```

	users	pay_total	pay_extra
city			
Череповец	2	20683.20	15183.20
Чита	4	15024.60	6774.60
Якутск	2	21597.67	13347.67
Ярославль	2	21273.87	12473.87

76 rows × 3 columns

Можно придумать что-нибудь и посложнее. Например, отберём только информацию о пользовании услугами тарифа "Ультра" москвичами со сроком контракта больше двух месяцев:

In [92]: `billing[filters['"Ультра" в Москве'] & filters['опытные пользователи']]`

	user_id	month	calls_total	minutes_total	...	date_to	reg_date	tariff	city
17	1003	3	97	855	...	2018-11-16	2018-08-17	Ультра	Москва
18	1003	4	95	824	...	2018-12-16	2018-08-17	Ультра	Москва
277	1049	3	63	525	...	2018-06-09	2018-03-10	Ультра	Москва
278	1049	4	66	520	...	2018-07-09	2018-03-10	Ультра	Москва
279	1049	5	72	653	...	2018-08-09	2018-03-10	Ультра	Москва
...
2845	1476	4	54	492	...	2018-08-29	2018-04-30	Ультра	Москва
2846	1476	5	50	456	...	2018-09-29	2018-04-30	Ультра	Москва
2847	1476	6	59	622	...	2018-10-29	2018-04-30	Ультра	Москва
2848	1476	7	60	540	...	2018-11-29	2018-04-30	Ультра	Москва
2849	1476	8	59	556	...	2018-12-29	2018-04-30	Ультра	Москва

149 rows × 30 columns

И ещё создадим словарь с осмысленными описаниями параметров биллинга, которые подлежат анализу:

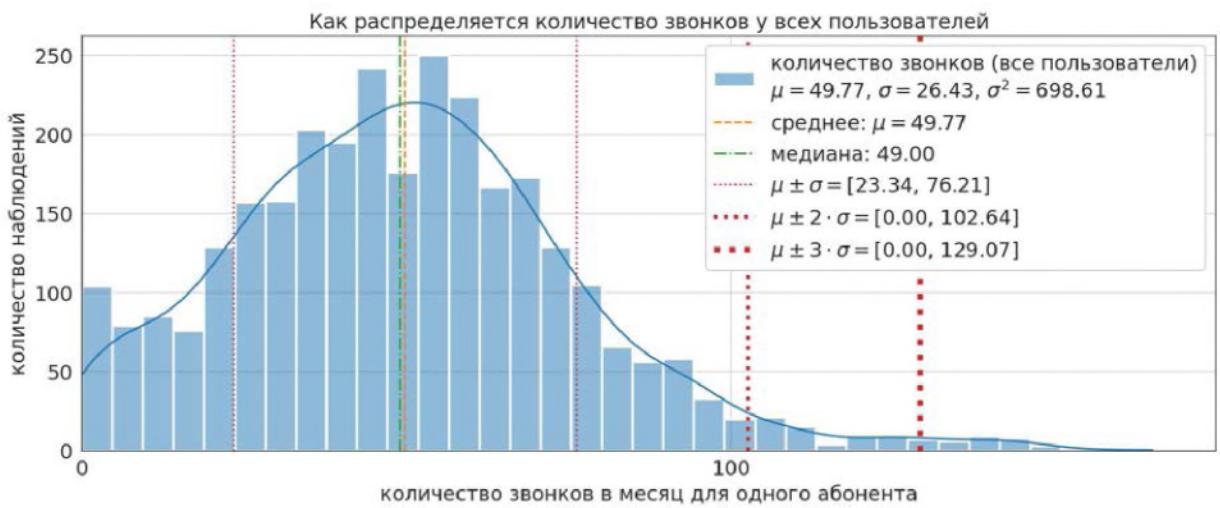
```
In [93]: parameters = {
    'calls_total': 'количество звонков',
    'minutes_total': 'количество минут',
    'minutes_charged': 'количество минут сверх тарифа',
    'messages_total': 'количество сообщений',
    'messages_charged': 'количество сообщений сверх тарифа',
    'mb_total': 'объём трафика в Мб',
    'gb_total': 'объём трафика в Гб',
    'mb_charged': 'объём трафика в Мб сверх тарифа',
    'pay_total': 'выручка',
    'pay_extra': 'выручка по услугам сверх тарифа',
    'pay_minutes': 'выручка по звонкам сверх тарифа',
    'pay_messages': 'выручка по сообщениям сверх тарифа',
    'pay_internet': 'выручка по трафику сверх тарифа'
}
```

Распределение количества звонков в месяц

```
In [94]: plot_billing('calls_total', ['все пользователи'], 'у всех пользователей', hist=True)
```

Изучим, как распределяется количество звонков у всех пользователей:

все пользователи	
количество образцов	2979.000000
среднее значение	49.774757
среднеквадратичное отклонение	26.431265
минимум	0.000000
25%	31.000000
50%	49.000000
75%	65.000000
максимум	165.000000
дисперсия	698.611774



В среднем пользователи говорят по телефону около 49-50 раз в месяц. На это указывает и среднее значение и медиана.

Распределение больше похоже на [распределение Пуассона](#).

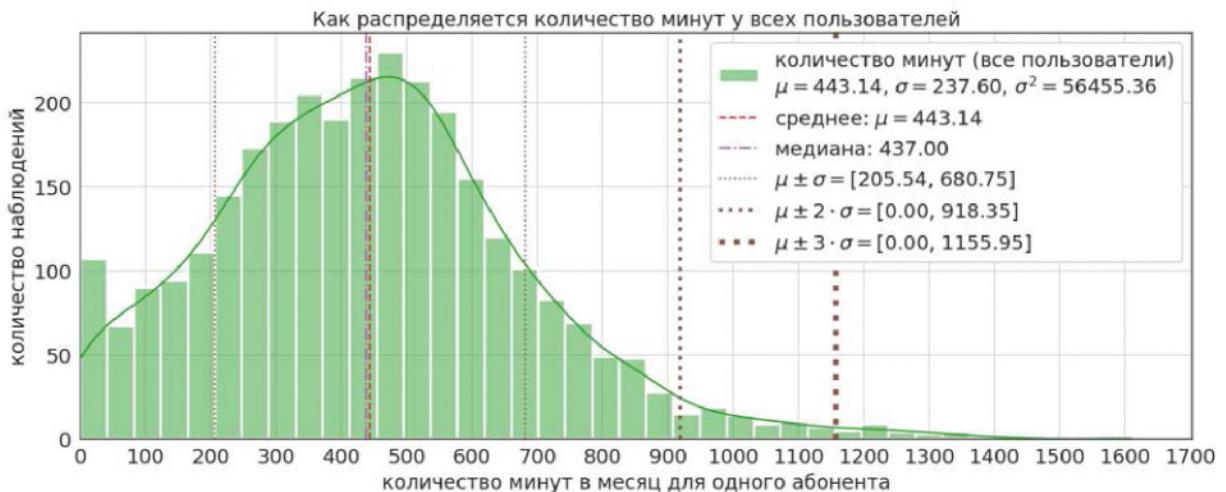
Распределение минут разговора в месяц

```
In [95]: plot_billing('minutes_total', ['все пользователи'], 'у всех пользователей', hist=True)
```

Изучим, как распределяется количество минут у всех пользователей:

все пользователи	
количество образцов	2979.000000
среднее значение	443.143001
среднеквадратичное отклонение	237.603376
минимум	0.000000

все пользователи	
25%	278.500000
50%	437.000000
75%	581.000000
максимум	1609.000000
дисперсия	56455.364366



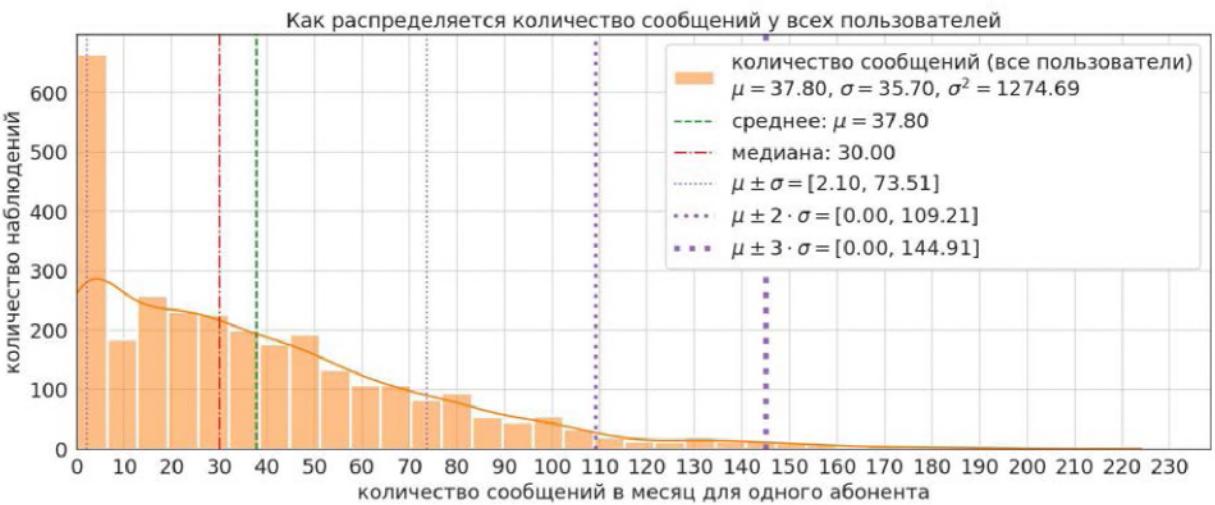
Средняя общая продолжительность разговоров в месяц у одного абонента достигает 443 минуты. Распределение имеет правый хвост - есть абоненты, которые говорят значительно больше остальной массы клиентов сотовой компании.

Распределение количества отправляемых сообщений в месяц

```
In [96]: plot_billing('messages_total', ['все пользователи'], 'у всех пользователей', hist=T)
```

Изучим, как распределяется количество сообщений у всех пользователей:

все пользователи	
количество образцов	2979.000000
среднее значение	37.802954
среднеквадратичное отклонение	35.702822
минимум	0.000000
25%	9.000000
50%	30.000000
75%	56.000000
максимум	224.000000
дисперсия	1274.691516



Четверть клиентов отправляет не больше 9 сообщений в месяц, половина - не более 30. Смещение среднего значения вправо относительно медианы и большая дисперсия говорит о существовании больших любителей коротких сообщений. Максимальное количество в месяц - 224.

Распределение объема интернет-трафика в месяц

```
In [97]: plot_billing('gb_total', ['все пользователи'], 'у всех пользователей', hist=False)
```

Изучим, как распределяется объём трафика в Гб у всех пользователей:

все пользователи	
количество образцов	2979.000000
среднее значение	16.563140
среднеквадратичное отклонение	7.360693
минимум	0.000000
25%	11.945312
50%	16.336914
75%	20.733398
максимум	48.622070
дисперсия	54.179806



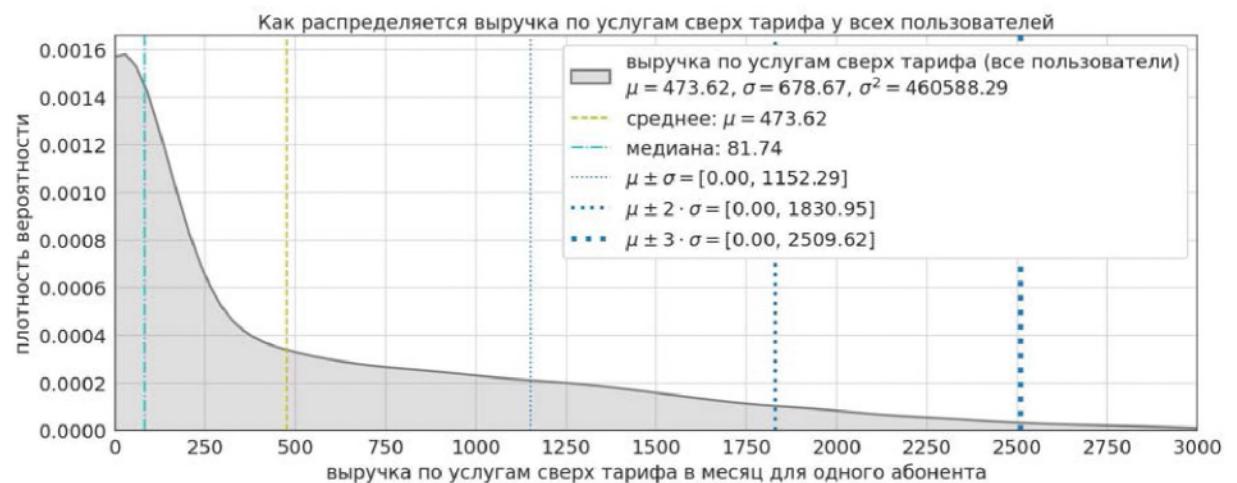
Средний объем трафика в месяц - около 16.6 Гб. Это больше, чем лимит у тарифа "Смарт", и меньше "Ультры". Наверняка есть пользователи, которые платят дополнительную плату за трафик у более дешевого "Смarta" и не выбирают пакеты у премиального "Ультра".

Распределение стоимости дополнительных услуг в месяц

```
In [98]: plot_billing('pay_extra', ['все пользователи'], 'у всех пользователей', hist=False, fill=True, start_color=7, max_x=3000, step=250)
```

Изучим, как распределяется выручка по услугам сверх тарифа у всех пользователей:

все пользователи	
количество образцов	2979.000000
среднее значение	473.620695
среднеквадратичное отклонение	678.666556
минимум	0.000000
25%	0.000000
50%	81.740000
75%	803.365000
максимум	5023.550000
дисперсия	460588.293705



Тяжелый правый хвост и дисперсия, говорят, что разброс значений достаточно высок, но, как минимум, четверть пользователей не выходит за лимиты пакетов.

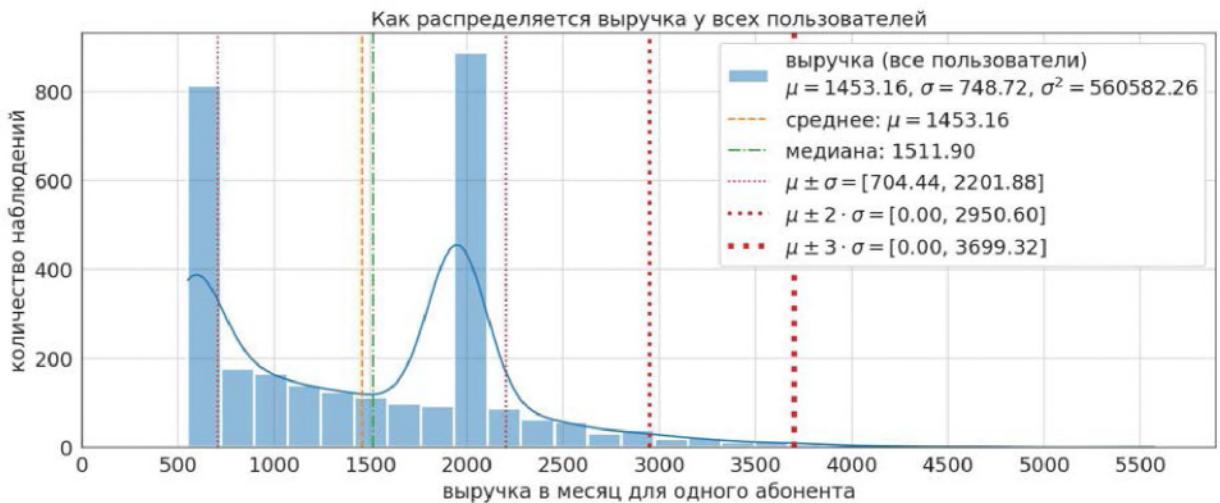
Распределение расходов пользователей в месяц

```
In [99]: plot_billing('pay_total', ['все пользователи'], 'у всех пользователей', hist=True,
```

Изучим, как распределяется выручка у всех пользователей:

все пользователи	
количество образцов	2979.000000
среднее значение	1453.160809

все пользователи	
среднеквадратичное отклонение	748.720413
минимум	550.000000
25%	671.500000
50%	1511.900000
75%	1950.000000
максимум	5573.550000
дисперсия	560582.257325



Два горба на графике являются абонентской платой по двум тарифам, вокруг них и распределяются пользователи "Смарт" и "Ультра".

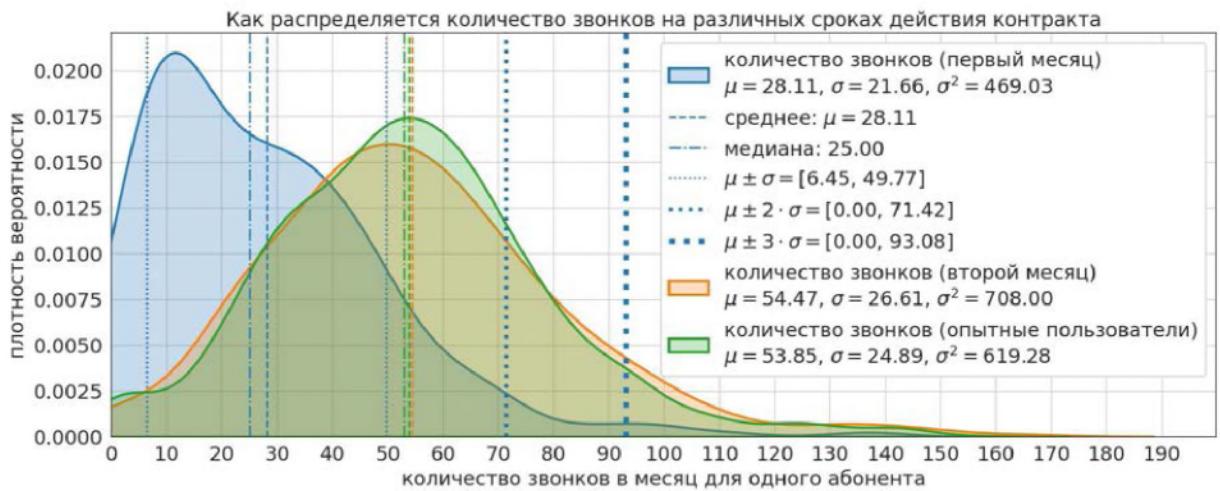
Эволюция пользователей по мере приобретения опыта

Изучим изменение поведения клиентов по мере привыкания к услугам сотовой связи:

```
In [100...]: plot_billing('calls_total', ['первый месяц', 'второй месяц', 'опытные пользователи'],
                           'на различных сроках действия контракта', fill=True, step=10)
```

Изучим, как распределяется количество звонков на различных сроках действия контракта:

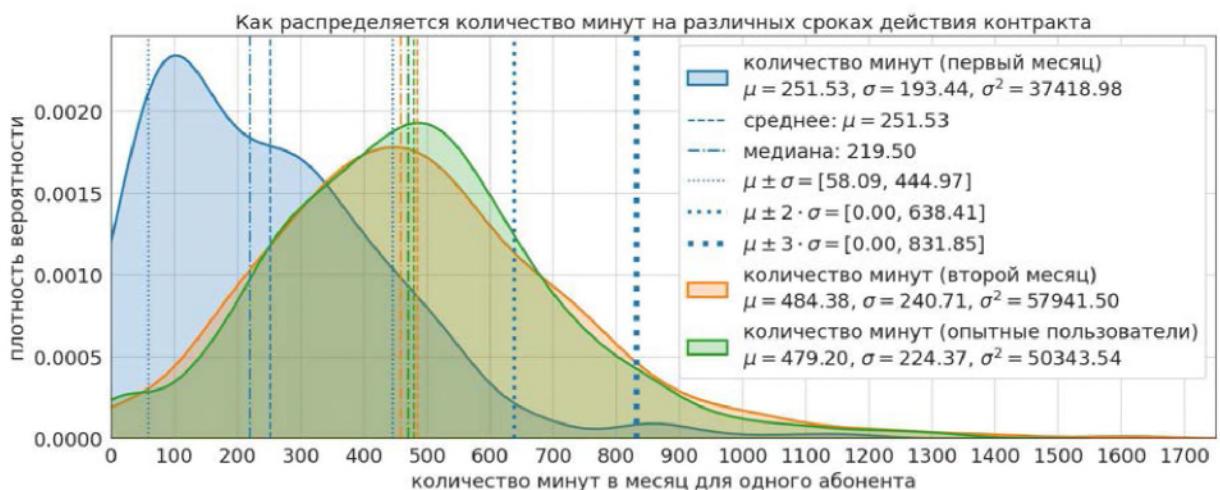
	первый месяц	второй месяц	опытные пользователи
количество образцов	482.000000	448.000000	2049.000000
среднее значение	28.107884	54.466518	53.845778
среднеквадратичное отклонение	21.657147	26.608248	24.885375
минимум	0.000000	0.000000	0.000000
25%	11.000000	37.000000	37.000000
50%	25.000000	53.000000	53.000000
75%	40.000000	69.000000	68.000000
максимум	140.000000	165.000000	157.000000
дисперсия	469.031996	707.998876	619.281868



```
In [101...]: plot_billing('minutes_total', ['первый месяц', 'второй месяц', 'опытные пользователи'],
                           'на различных сроках действия контракта', fill=True, max_x=1750, step
```

Изучим, как распределяется количество минут на различных сроках действия контракта:

	первый месяц	второй месяц	опытные пользователи
количество образцов	482.000000	448.000000	2049.000000
среднее значение	251.533195	484.381696	479.200098
среднеквадратичное отклонение	193.439865	240.710413	224.373668
минимум	0.000000	0.000000	0.000000
25%	98.000000	326.750000	329.000000
50%	219.500000	458.500000	470.000000
75%	358.250000	618.750000	606.000000
максимум	1186.000000	1609.000000	1401.000000
дисперсия	37418.981224	57941.502751	50343.542949

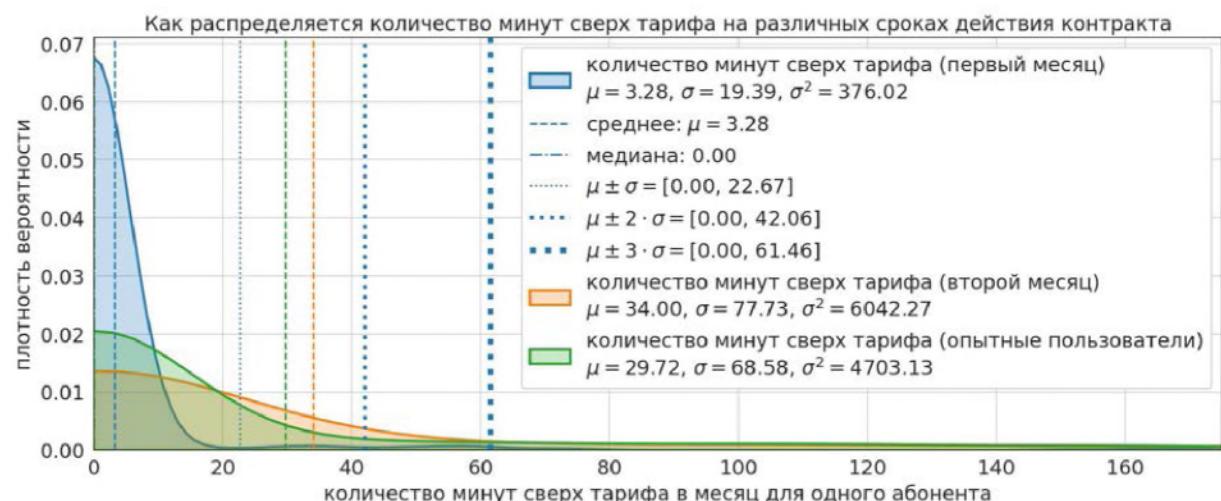


Интересное наблюдение - в первый месяц после заключения контракта абоненты совершают меньше звонков, чем в последующие периоды.

```
In [102...]: plot_billing('minutes_charged', ['первый месяц', 'второй месяц', 'опытные пользователи'],
                           'на различных сроках действия контракта', fill=True, max_x=175)
```

Изучим, как распределяется количество минут сверх тарифа на различных сроках действия контракта:

	первый месяц	второй месяц	опытные пользователи
количество образцов	482.000000	448.000000	2049.000000
среднее значение	3.282158	34.004464	29.722792
среднеквадратичное отклонение	19.391128	77.732058	68.579364
минимум	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.000000	0.000000
75%	0.000000	8.000000	9.000000
максимум	205.000000	465.000000	505.000000
дисперсия	376.015856	6042.272911	4703.129173



Привыкшие к сотовой связи клиенты чаще выходят за лимиты по минутам разговоров в месяц, чем новые.

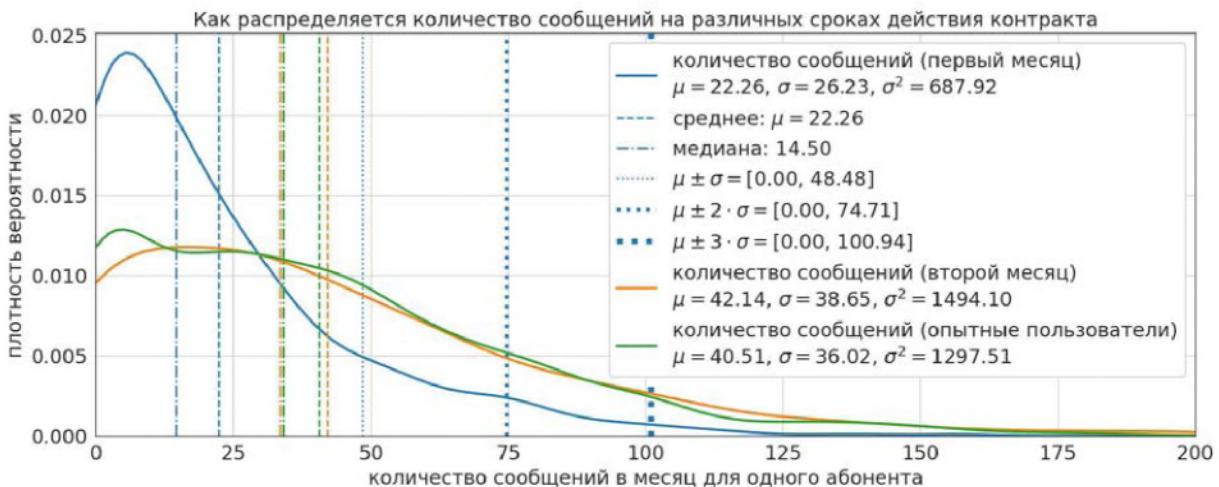
```
In [103]: plot_billing('messages_total', ['первый месяц', 'второй месяц', 'опытные пользователи',
                                         'на различных сроках действия контракта', fill=False, max_x=200])
```

Изучим, как распределяется количество сообщений на различных сроках действия контракта:

	первый месяц	второй месяц	опытные пользователи
количество образцов	482.000000	448.000000	2049.000000
среднее значение	22.255187	42.138393	40.512445
среднеквадратичное отклонение	26.228233	38.653611	36.020962
минимум	0.000000	0.000000	0.000000
25%	3.000000	12.750000	12.000000
50%	14.500000	33.500000	34.000000
75%	32.000000	61.250000	60.000000
максимум	223.000000	201.000000	224.000000

первый месяц второй месяц опытные пользователи

дисперсия	687.920191	1494.101610	1297.509733
-----------	------------	-------------	-------------



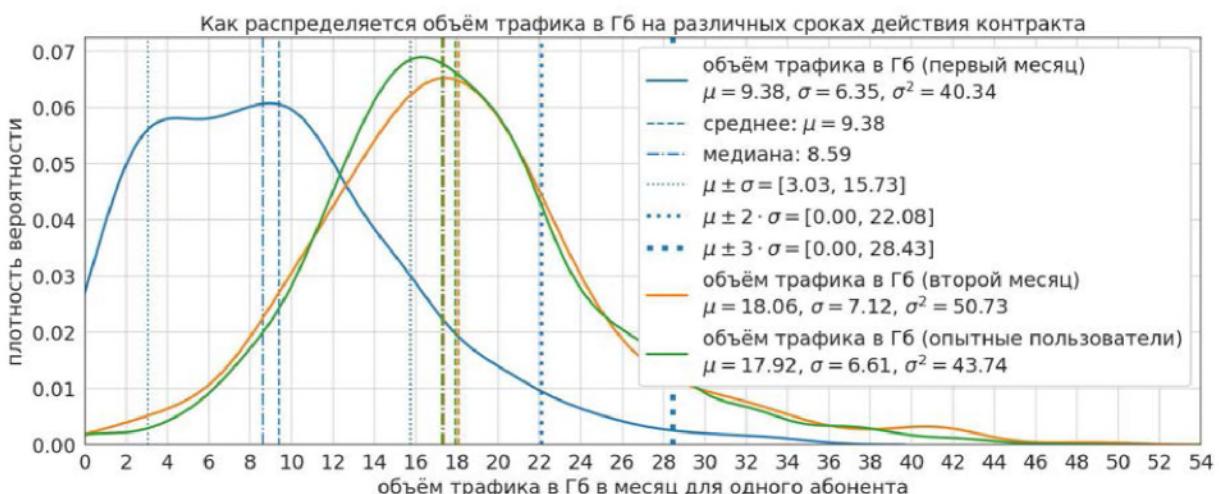
Среднее количество сообщений от абонентов в первый месяц почти вдвое меньше, чем у более опытных пользователей.

```
In [104]: plot_billing('gb_total', ['первый месяц', 'второй месяц', 'опытные пользователи'],
                    'на различных сроках действия контракта', max_x=54, step=2)
```

Изучим, как распределяется объём трафика в Гб на различных сроках действия контракта:

первый месяц второй месяц опытные пользователи

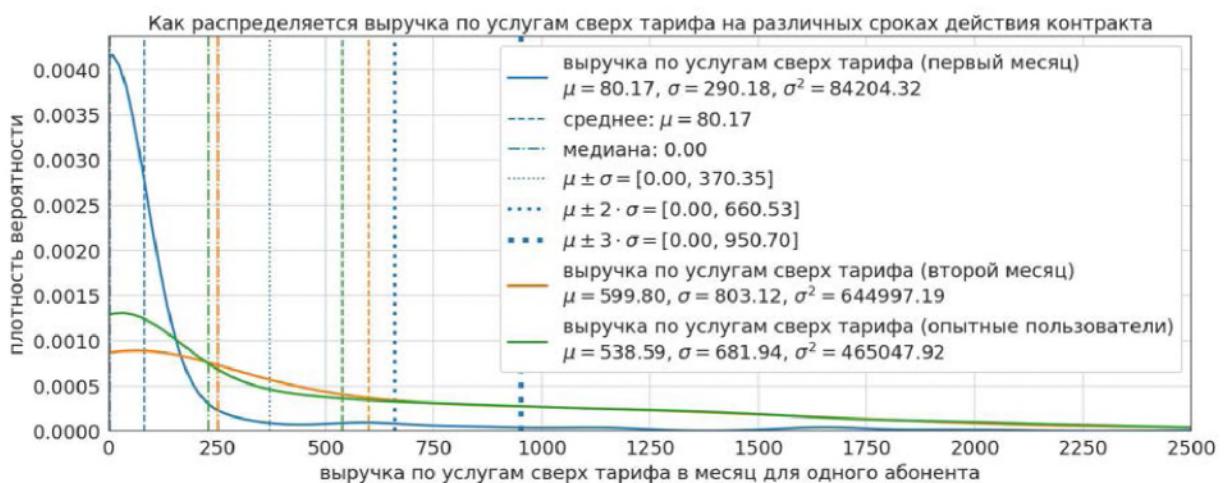
количество образцов	482.000000	448.000000	2049.000000
среднее значение	9.380065	18.062986	17.924931
среднеквадратичное отклонение	6.351626	7.122619	6.613942
минимум	0.000000	0.000000	0.000000
25%	4.263672	13.652588	13.687500
50%	8.589355	17.322754	17.280273
75%	12.917969	21.338867	21.232422
максимум	34.105469	48.622070	47.260742
дисперсия	40.343147	50.731694	43.744230



```
In [105...]: plot_billing('pay_extra', ['первый месяц', 'второй месяц', 'опытные пользователи'],
                           'на различных сроках действия контракта', max_x=2500, step=250)
```

Изучим, как распределяется выручка по услугам сверх тарифа на различных сроках действия контракта:

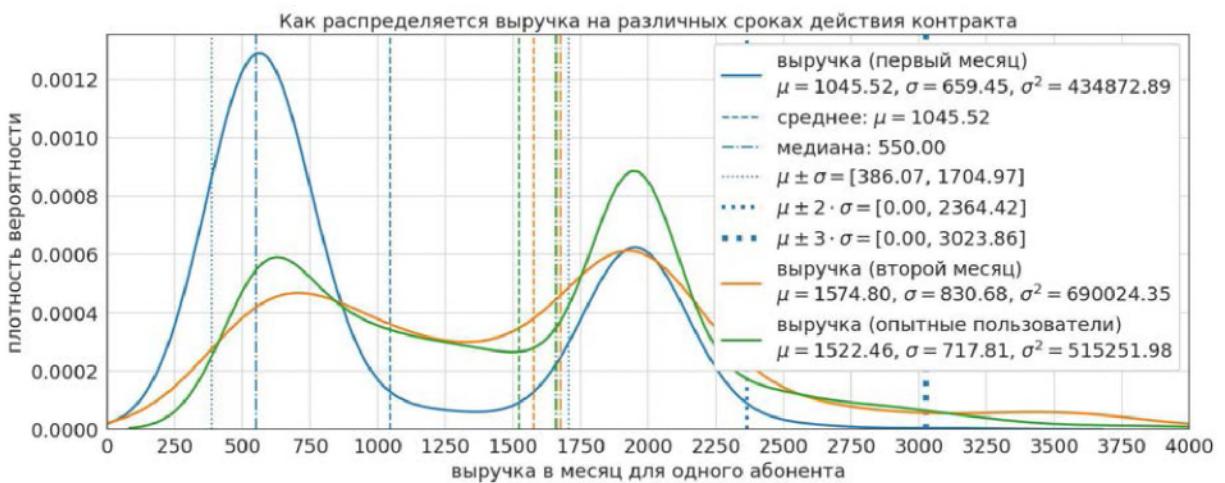
	первый месяц	второй месяц	опытные пользователи
количество образцов	482.000000	448.000000	2049.000000
среднее значение	80.165394	599.797612	538.588092
среднеквадратичное отклонение	290.179807	803.117169	681.944217
минимум	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	251.975000	228.710000
75%	0.000000	971.367500	922.710000
максимум	2557.570000	5023.550000	3642.980000
дисперсия	84204.320539	644997.187549	465047.915161



```
In [106...]: plot_billing('pay_total', ['первый месяц', 'второй месяц', 'опытные пользователи'],
                           'на различных сроках действия контракта', max_x=4000, step=250)
```

Изучим, как распределяется выручка на различных сроках действия контракта:

	первый месяц	второй месяц	опытные пользователи
количество образцов	482.000000	448.000000	2049.000000
среднее значение	1045.518091	1574.797612	1522.458272
среднеквадратичное отклонение	659.448929	830.677043	717.810548
минимум	550.000000	550.000000	550.000000
25%	550.000000	841.000000	839.150000
50%	550.000000	1675.020000	1659.280000
75%	1950.000000	1950.000000	1950.000000
максимум	3107.570000	5573.550000	4539.110000
дисперсия	434872.890617	690024.349741	515251.982706



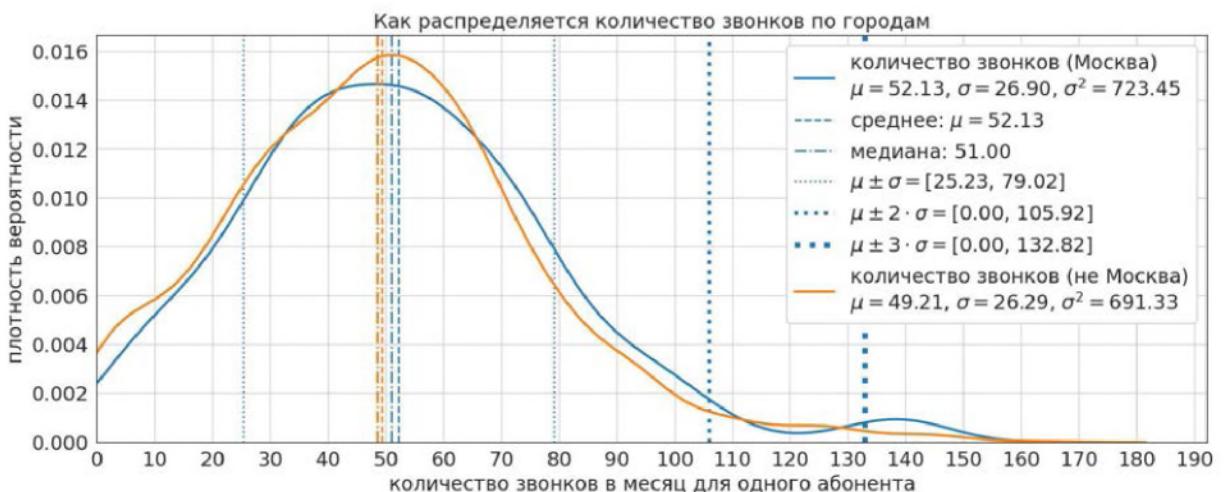
Средняя выручка от новых клиентов в месяц меньше.

Москва и другие города

```
In [107...]: plot_billing('calls_total', ['Москва', 'не Москва'], 'по городам', step=10)
```

Изучим, как распределяется количество звонков по городам:

	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	52.126957	49.212146
среднеквадратичное отклонение	26.897007	26.293138
минимум	0.000000	0.000000
25%	33.500000	31.000000
50%	51.000000	48.500000
75%	69.000000	65.000000
максимум	146.000000	165.000000
дисперсия	723.449011	691.329091

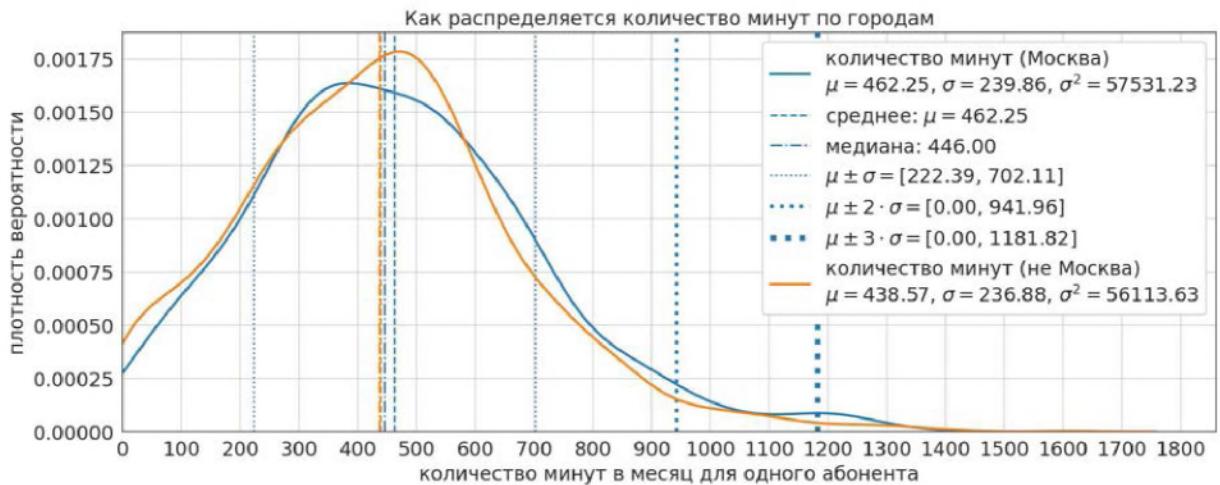


Москвичи звонят чуть чаще: 52 звонка против 49 в месяц, но эта разница абсолютно не существенна. Проверим эту гипотезу ниже ([спойлер](#)).

```
In [108...]: plot_billing('minutes_total', ['Москва', 'не Москва'], 'по городам', step=100)
```

Изучим, как распределяется количество минут по городам:

	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	462.248696	438.573211
среднеквадратичное отклонение	239.856684	236.883160
минимум	0.000000	0.000000
25%	296.500000	275.750000
50%	446.000000	436.000000
75%	611.500000	575.250000
максимум	1321.000000	1609.000000
дисперсия	57531.228983	56113.631342

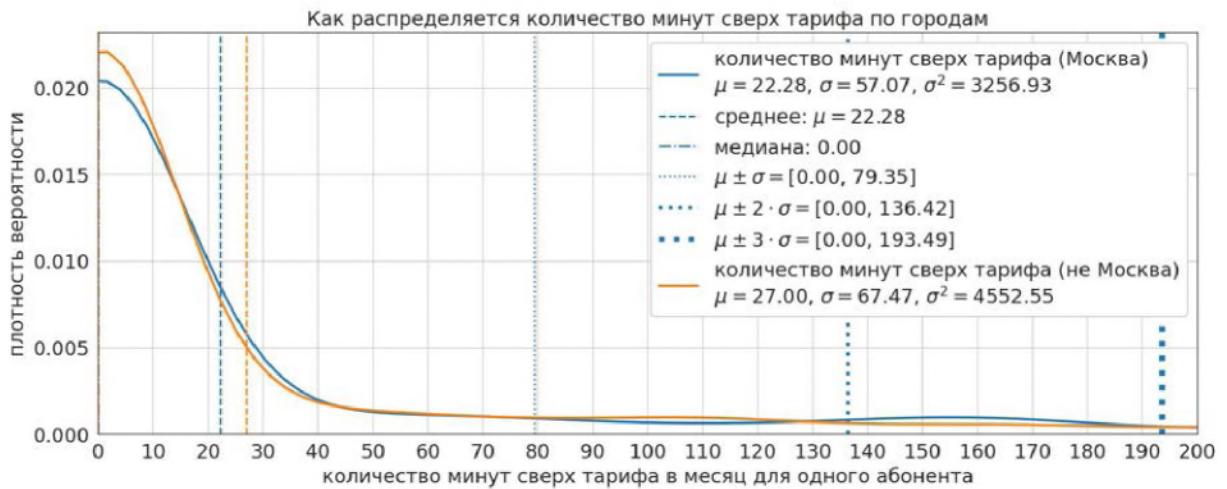


Нет существенной разницы между средним количеством минут телефонных разговоров в месяц в Москве и регионах.

```
In [109]: plot_billing('minutes_charged', ['Москва', 'не Москва'], ' по городам', max_x=200, s
```

Изучим, как распределяется количество минут сверх тарифа по городам:

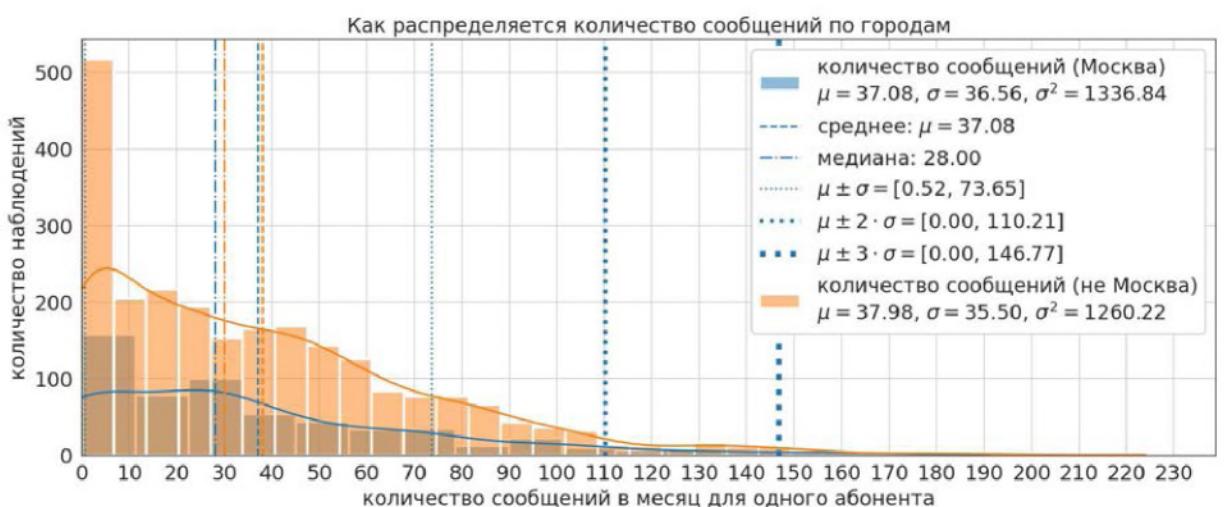
	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	22.278261	27.000000
среднеквадратичное отклонение	57.069544	67.47260
минимум	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
максимум	375.000000	505.000000
дисперсия	3256.932889	4552.55181



```
In [110]: plot_billing('messages_total', ['Москва', 'не Москва'], ' по городам', hist=True, fi
```

Изучим, как распределяется количество сообщений по городам:

	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	37.083478	37.975042
среднеквадратичное отклонение	36.562819	35.499623
минимум	0.000000	0.000000
25%	6.000000	10.000000
50%	28.000000	30.000000
75%	54.000000	56.000000
максимум	191.000000	224.000000
дисперсия	1336.839709	1260.223264



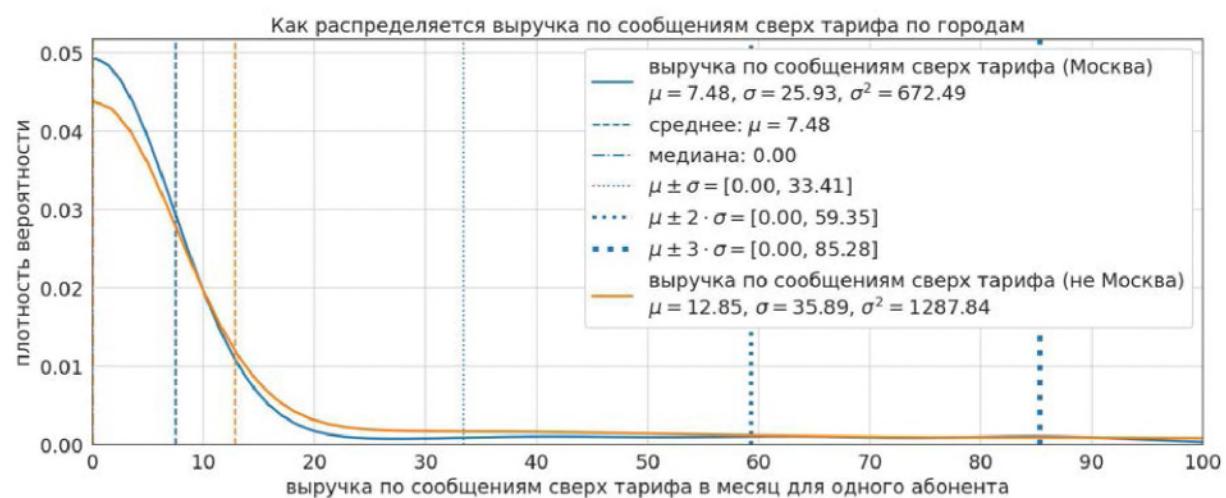
По среднему количеству сообщений разница заметна ещё меньше.

```
In [111]: plot_billing('pay_messages', ['Москва', 'не Москва'], ' по городам', max_x=100, step
```

Изучим, как распределяется выручка по сообщениям сверх тарифа по городам:

	Москва	не Москва
--	--------	-----------

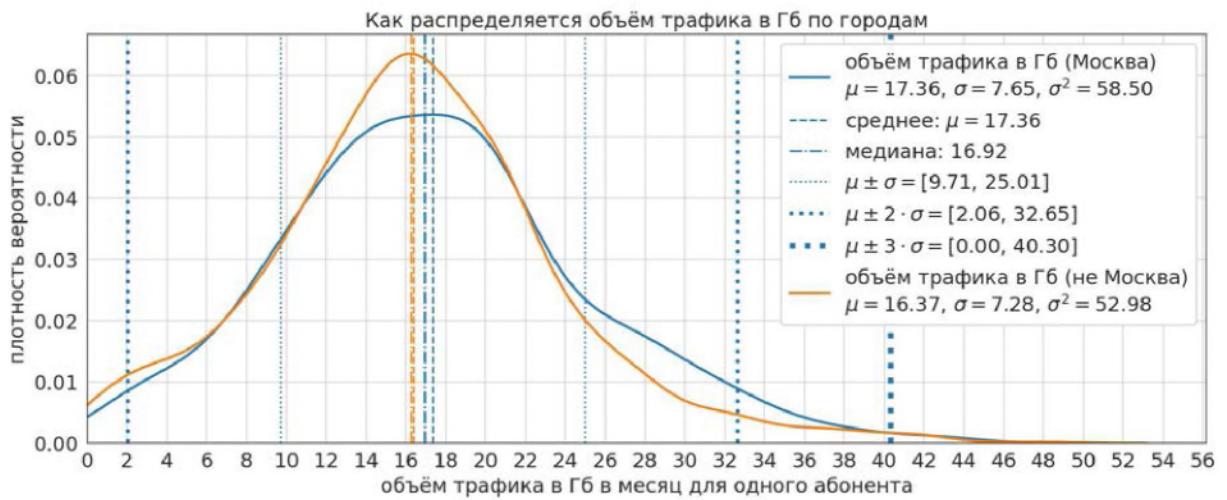
	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	7.481739	12.849834
среднеквадратичное отклонение	25.932355	35.886530
минимум	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	0.000000	0.000000
максимум	177.000000	279.000000
дисперсия	672.487035	1287.843025



```
In [112]: plot_billing('gb_total', ['Москва', 'не Москва'], ' по городам', step=2)
```

Изучим, как распределяется объём трафика в Гб по городам:

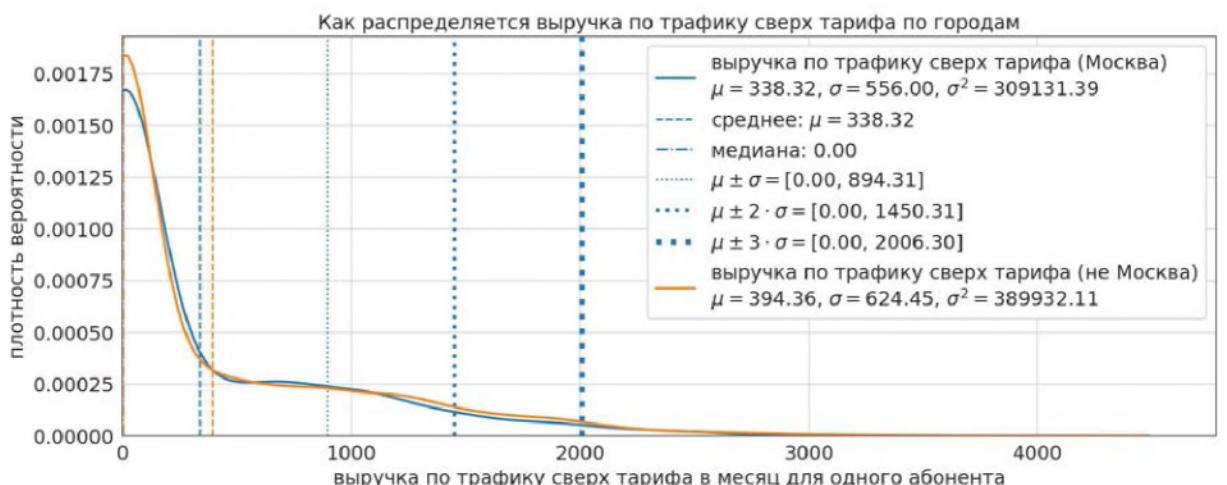
	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	17.357189	16.373215
среднеквадратичное отклонение	7.648422	7.278999
минимум	0.357422	0.000000
25%	12.200195	11.865234
50%	16.922852	16.257324
75%	21.436035	20.504883
максимум	43.301758	48.622070
дисперсия	58.498358	52.983830



```
In [113]: plot_billing('pay_internet', ['Москва', 'не Москва'], ' по городам')
```

Изучим, как распределяется выручка по трафику сверх тарифа по городам:

	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	338.316365	394.358211
среднеквадратичное отклонение	555.995850	624.445438
минимум	0.000000	0.000000
25%	0.000000	0.000000
50%	0.000000	0.000000
75%	587.325000	651.270000
максимум	2600.780000	4093.550000
дисперсия	309131.385420	389932.105476

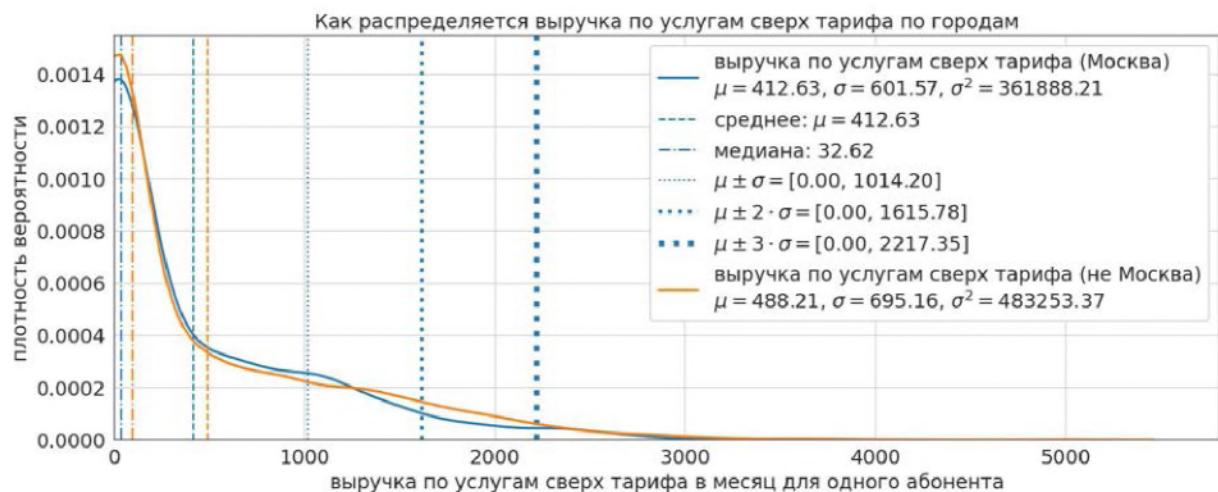


```
In [114]: plot_billing('pay_extra', ['Москва', 'не Москва'], ' по городам')
```

Изучим, как распределяется выручка по услугам сверх тарифа по городам:

	Москва	не Москва
количество образцов	575.000000	2404.000000

	Москва	не Москва
среднее значение	412.632887	488.208045
среднеквадратичное отклонение	601.571449	695.164273
минимум	0.000000	0.000000
25%	0.000000	0.000000
50%	32.620000	96.000000
75%	696.900000	822.260000
максимум	2825.780000	5023.550000
дисперсия	361888.208270	483253.365900

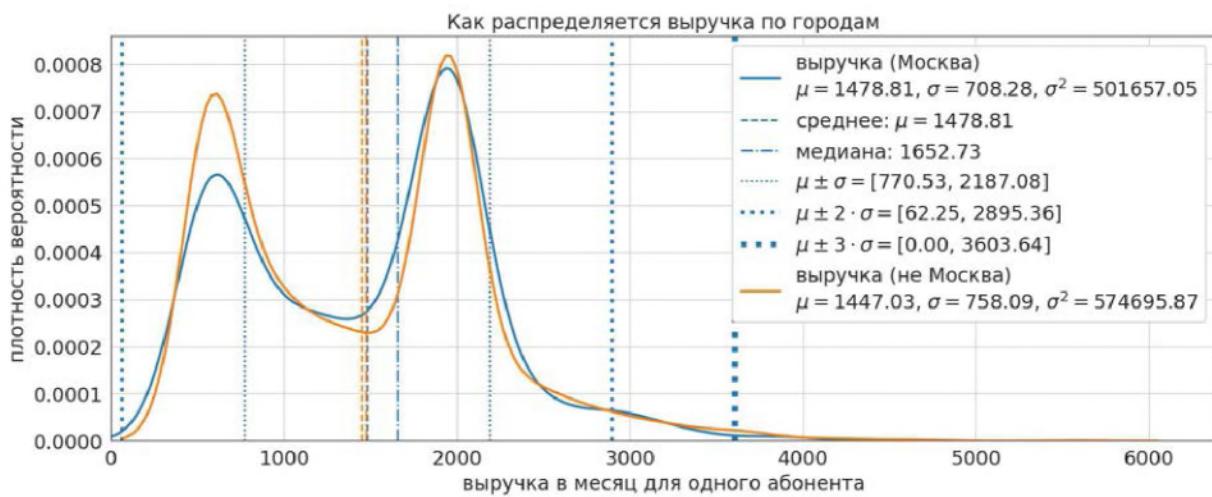


Следующие графики также имеют непосредственное отношение к проверке одной из гипотез, предложенных в задании. Они иллюстрируют распределение величин, которые мы сравним ниже ([спойлер](#)).

```
In [115]: plot_billing('pay_total', ['Москва', 'не Москва'], ' по городам')
```

Изучим, как распределяется выручка по городам:

	Москва	не Москва
количество образцов	575.000000	2404.000000
среднее значение	1478.806800	1447.026681
среднеквадратичное отклонение	708.277525	758.086979
минимум	550.000000	550.000000
25%	719.500000	658.000000
50%	1652.730000	1473.240000
75%	1950.000000	1950.000000
максимум	3945.260000	5573.550000
дисперсия	501657.053064	574695.867697

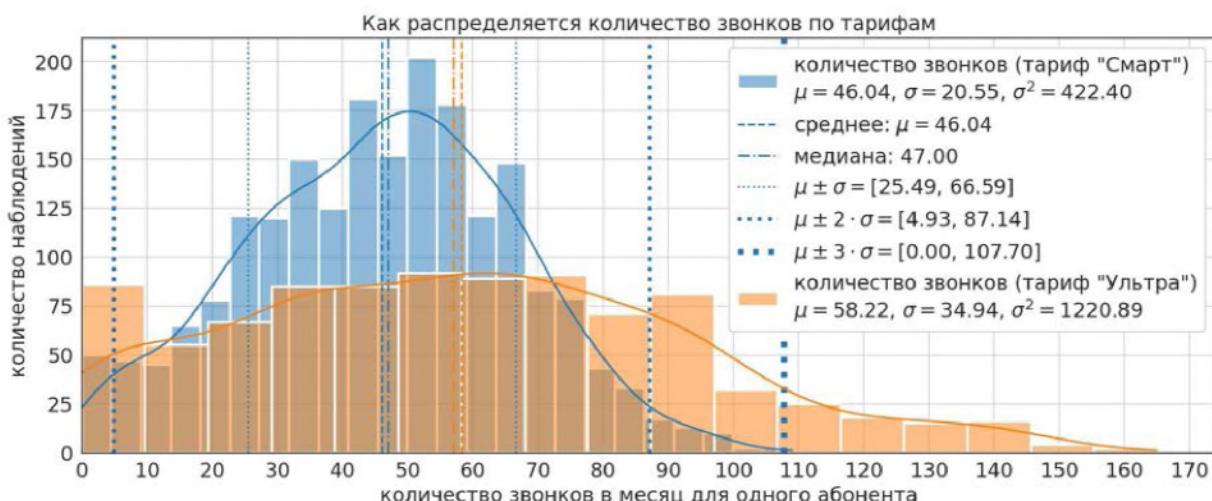


Сравнение тарифов

```
In [116]: plot_billing('calls_total', ['тариф "Смарт"', 'тариф "Ультра"'], ' по тарифам', hist)
```

Изучим, как распределяется количество звонков по тарифам:

	тариф "Смарт"	тариф "Ультра"
количество образцов	2065.000000	914.000000
среднее значение	46.038257	58.216630
среднеквадратичное отклонение	20.552329	34.941182
минимум	0.000000	0.000000
25%	31.000000	32.000000
50%	47.000000	57.000000
75%	61.000000	82.000000
максимум	109.000000	165.000000
дисперсия	422.398245	1220.886207

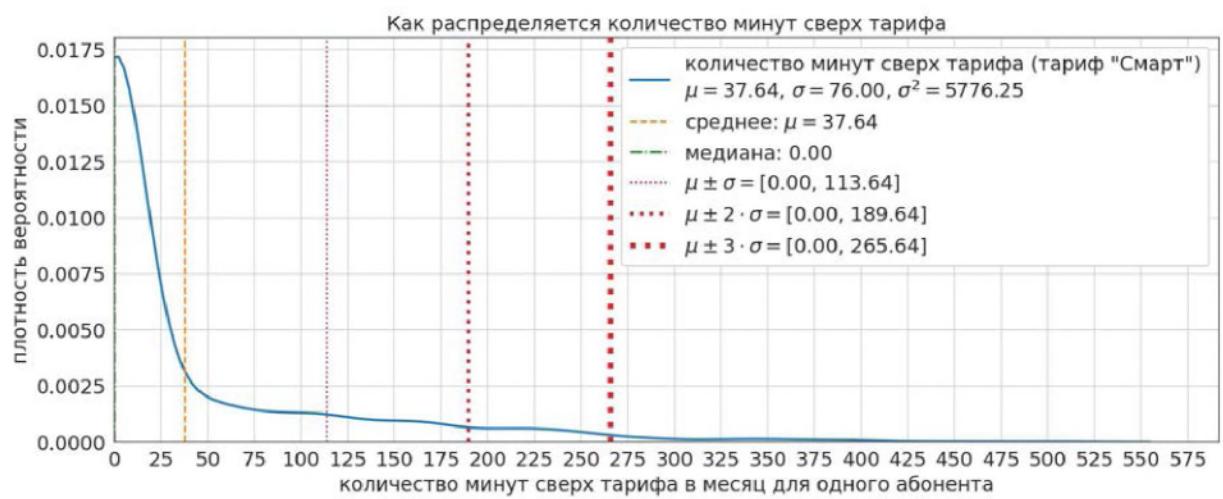


По гистограмме и графикам аппроксимации плотности вероятности видим, что среднее количество звонков в месяц от абонентов различных тарифов отличается: Смарт - около 46, Ультра - около 58. Да и разброс значений у "богатого" Ультра шире.

```
In [117]: plot_billing('minutes_charged', ['тариф "Смарт"'], step=25)
```

Изучим, как распределяется количество минут сверх тарифа:

тариф "Смарт"	
количество образцов	2065.000000
среднее значение	37.635835
среднеквадратичное отклонение	76.001645
минимум	0.000000
25%	0.000000
50%	0.000000
75%	39.000000
максимум	505.000000
дисперсия	5776.250072

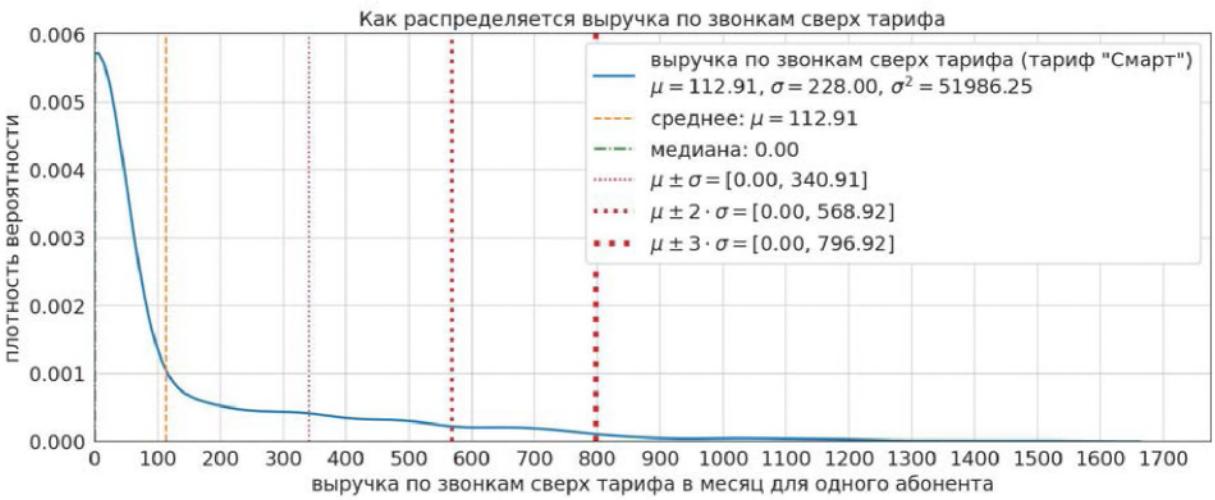


Пользователи тарифа "Ультра" с большими пакетами услуг, включенными в абонентскую плату, совершенно не выходят за лимиты по минутам.

```
In [118]: plot_billing('pay_minutes', ['тариф "Смарт"'], step=100)
```

Изучим, как распределяется выручка по звонкам сверх тарифа:

тариф "Смарт"	
количество образцов	2065.000000
среднее значение	112.907506
среднеквадратичное отклонение	228.004936
минимум	0.000000
25%	0.000000
50%	0.000000
75%	117.000000
максимум	1515.000000
дисперсия	51986.250646

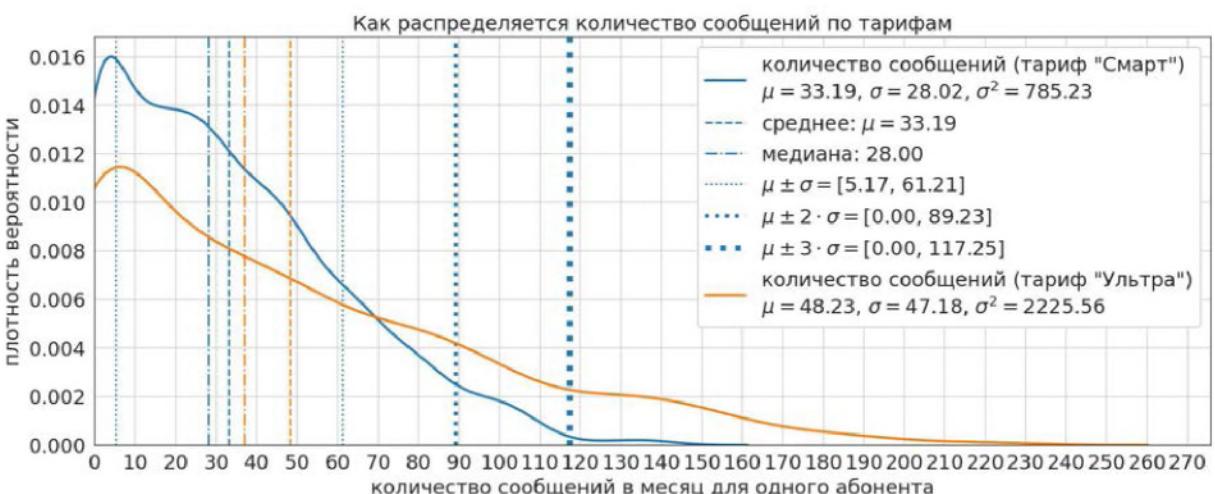


За дополнительные минуты в месяц платят пользователи тарифа "Смарт" с меньшими пакетами услуг, включенными в абонентскую плату.

```
In [119...]: plot_billing('messages_total', ['тариф "Смарт"', 'тариф "Ультра"'], 'по тарифам', s
```

Изучим, как распределяется количество сообщений по тарифам:

	тариф "Смарт"	тариф "Ультра"
количество образцов	2065.000000	914.000000
среднее значение	33.187409	48.230853
среднеквадратичное отклонение	28.021971	47.175818
минимум	0.000000	0.000000
25%	10.000000	6.000000
50%	28.000000	37.000000
75%	50.000000	76.000000
максимум	143.000000	224.000000
дисперсия	785.230849	2225.557820



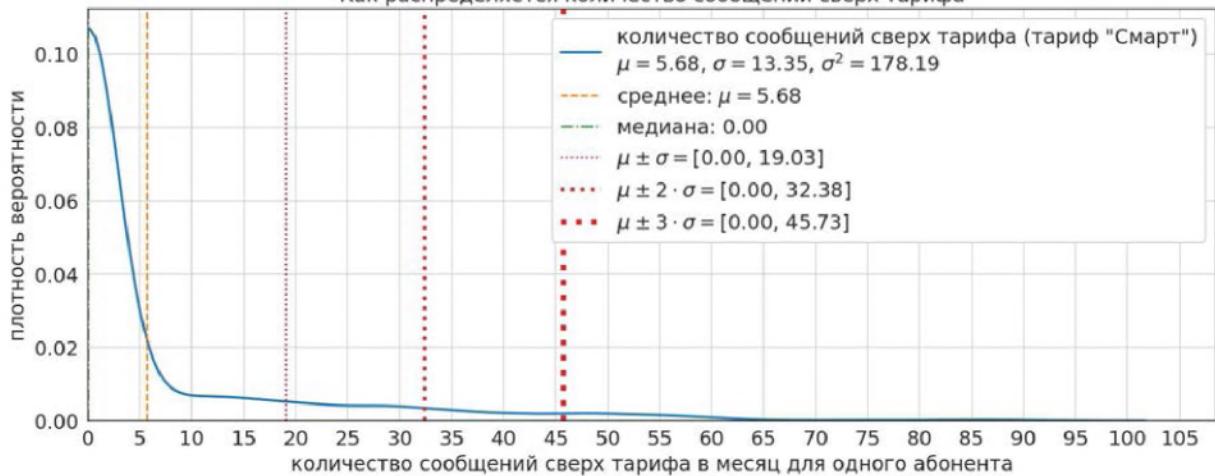
```
In [120...]: plot_billing('messages_charged', ['тариф "Смарт"'], step=5)
```

Изучим, как распределяется количество сообщений сверх тарифа:

тариф "Смарт"

количество образцов	2065.000000
среднее значение	5.680872
среднеквадратичное отклонение	13.348756
минимум	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
максимум	93.000000
дисперсия	178.189290

Как распределяется количество сообщений сверх тарифа



```
In [121]: plot_billing('pay_messages', ['тариф "Смарт"'], step=25)
```

Изучим, как распределяется выручка по сообщениям сверх тарифа:

тариф "Смарт"

количество образцов	2065.000000
среднее значение	17.042615
среднеквадратичное отклонение	40.046268
минимум	0.000000
25%	0.000000
50%	0.000000
75%	0.000000
максимум	279.000000
дисперсия	1603.703609

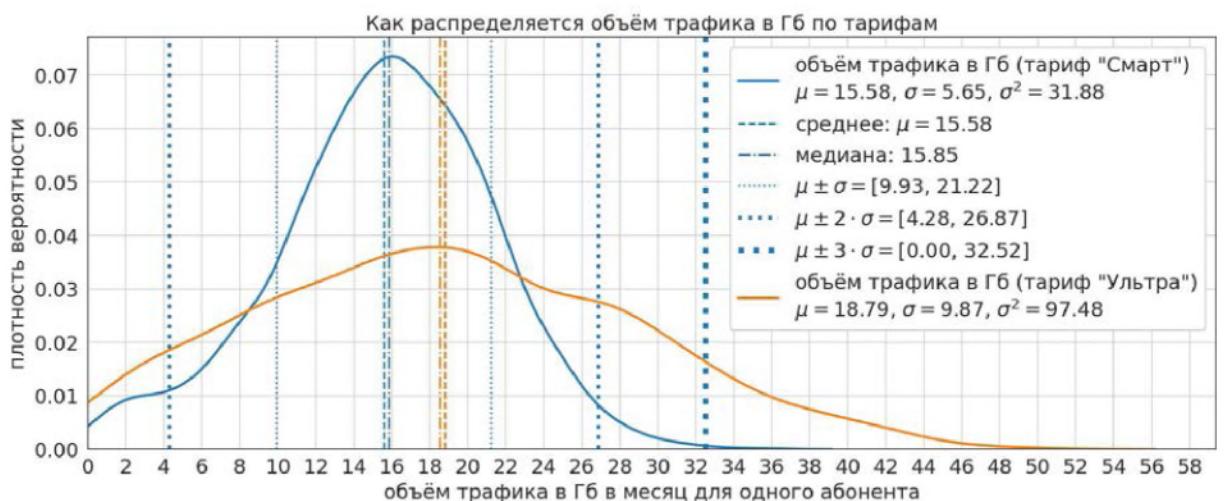


```
In [122...]: plot_billing('gb_total', ['тариф "Смарт"', 'тариф "Ультра"'], 'по тарифам', step=2)
```

Изучим, как распределяется объём трафика в Гб по тарифам:

тариф "Смарт" тариф "Ультра"

	тариф "Смарт"	тариф "Ультра"
количество образцов	2065.000000	914.000000
среднее значение	15.576786	18.791610
среднеквадратичное отклонение	5.646409	9.872967
минимум	0.000000	0.000000
25%	12.165039	11.171875
50%	15.845703	18.527832
75%	19.449219	26.158936
максимум	35.467773	48.622070
дисперсия	31.881940	97.475482



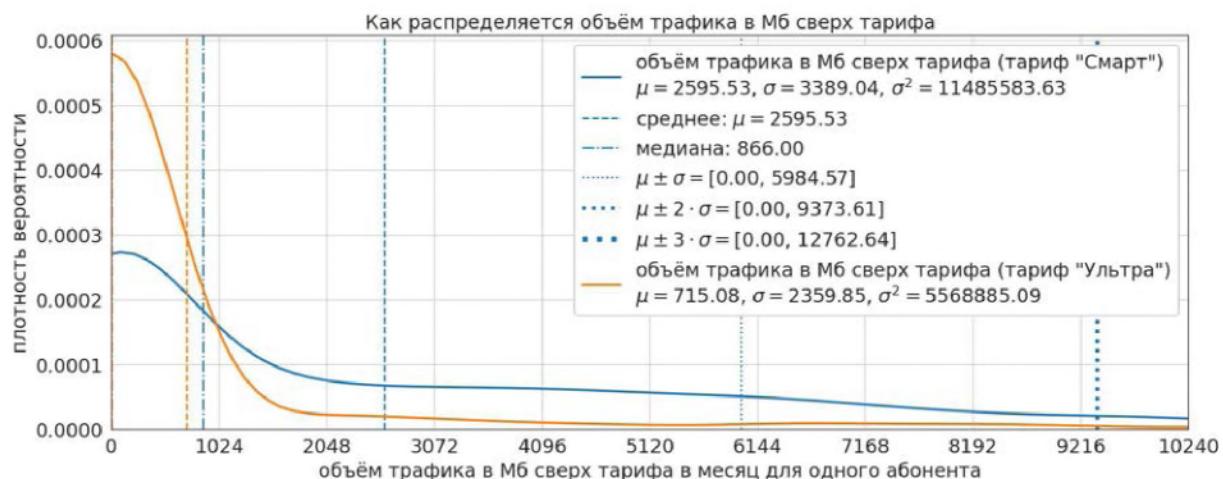
```
In [123...]: plot_billing('mb_charged', ['тариф "Смарт"', 'тариф "Ультра"'], max_x=10240, step=10)
```

Изучим, как распределяется объём трафика в Мб сверх тарифа:

тариф "Смарт" тариф "Ультра"

	тариф "Смарт"	тариф "Ультра"
количество образцов	2.065000e+03	9.140000e+02

	тариф "Смарт"	тариф "Ультра"
среднее значение	2.595528e+03	7.150810e+02
среднеквадратичное отклонение	3.389039e+03	2.359849e+03
минимум	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00
50%	8.660000e+02	0.000000e+00
75%	4.556000e+03	0.000000e+00
максимум	2.095900e+04	1.906900e+04
дисперсия	1.148558e+07	5.568885e+06

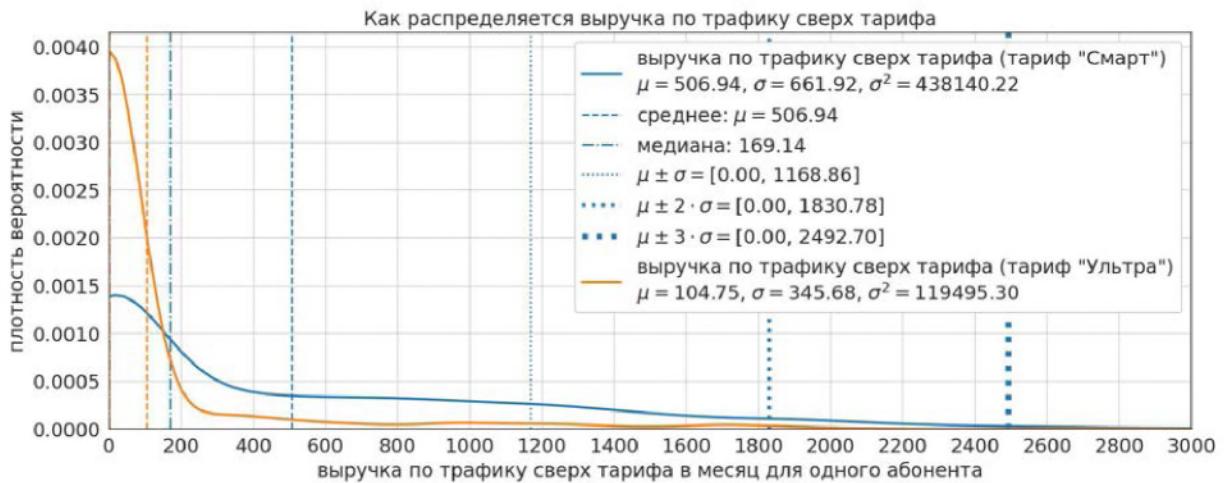


Пользователи тарифа "Ультра" имеют шанс превысить лимит лишь по интернет-трафику.

```
In [124]: plot_billing('pay_internet', ['тариф "Смарт"', 'тариф "Ультра"'], step=200, max_x=30)
```

Изучим, как распределяется выручка по трафику сверх тарифа:

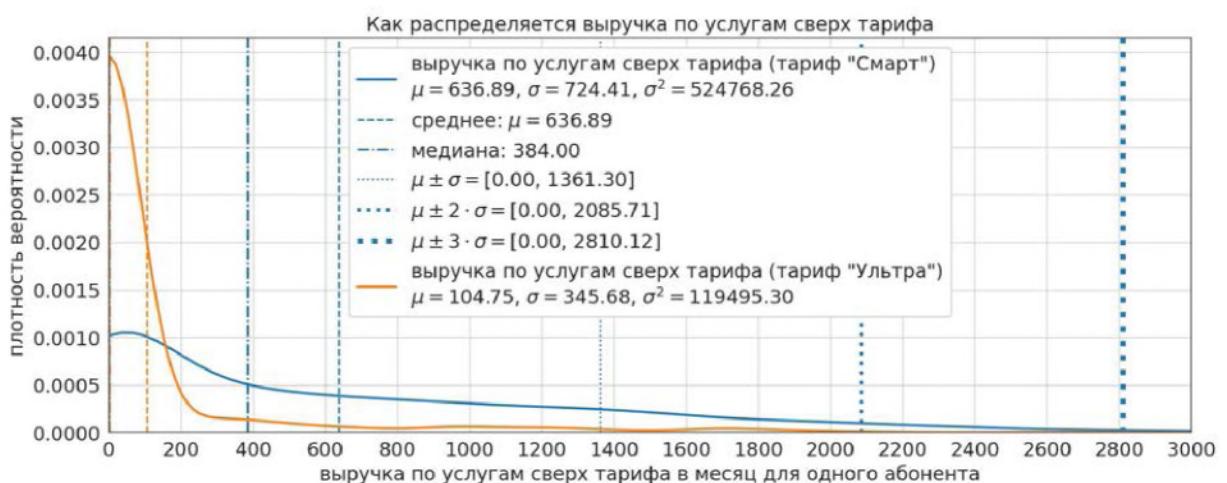
	тариф "Смарт"	тариф "Ультра"
количество образцов	2065.000000	914.000000
среднее значение	506.939070	104.748217
среднеквадратичное отклонение	661.921614	345.680928
минимум	0.000000	0.000000
25%	0.000000	0.000000
50%	169.140000	0.000000
75%	889.840000	0.000000
максимум	4093.550000	2793.310000
дисперсия	438140.222862	119495.303805



```
In [125...]: plot_billing('pay_extra', ['тариф "Смарт"', 'тариф "Ультра"'], step=200, max_x=3000)
```

Изучим, как распределяется выручка по услугам сверх тарифа:

	тариф "Смарт"	тариф "Ультра"
количество образцов	2065.000000	914.000000
среднее значение	636.889191	104.748217
среднеквадратичное отклонение	724.408905	345.680928
минимум	0.000000	0.000000
25%	0.000000	0.000000
50%	384.000000	0.000000
75%	1055.860000	0.000000
максимум	5023.550000	2793.310000
дисперсия	524768.261607	119495.303805

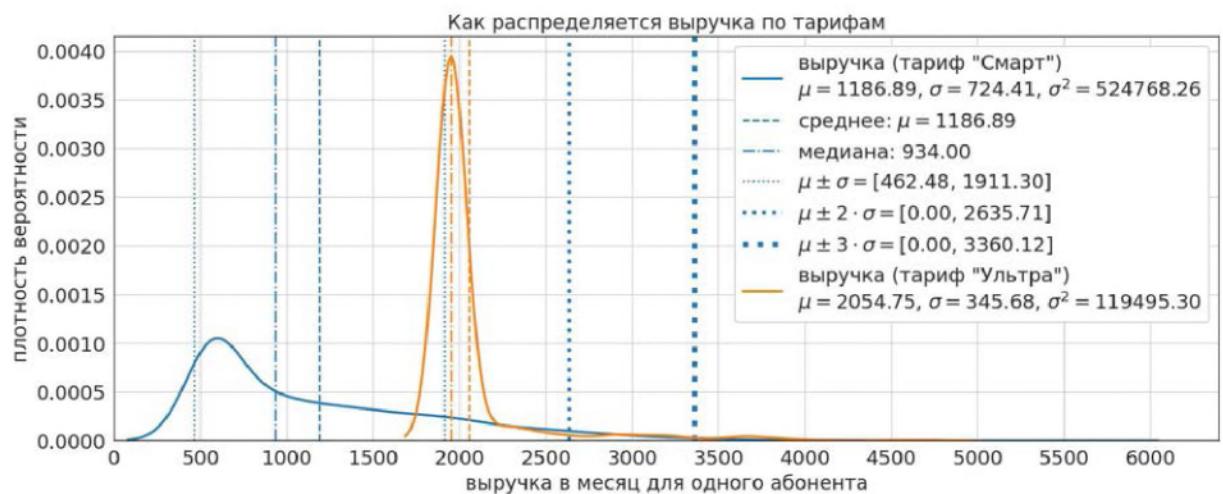


Следующий график займет особое место в нашем проекте, так как он иллюстрирует величины, которые требуется сравнить по заданию с помощью проверки гипотез. Мы сделаем это чуть позже ([спойлер](#)).

```
In [126...]: plot_billing('pay_total', ['тариф "Смарт"', 'тариф "Ультра"'], 'по тарифам', step=5)
```

Изучим, как распределяется выручка по тарифам:

	тариф "Смарт"	тариф "Ультра"
количество образцов	2065.000000	914.000000
среднее значение	1186.889191	2054.748217
среднеквадратичное отклонение	724.408905	345.680928
минимум	550.000000	1950.000000
25%	550.000000	1950.000000
50%	934.000000	1950.000000
75%	1605.860000	1950.000000
максимум	5573.550000	4743.310000
дисперсия	524768.261607	119495.303805



Интересно, подтвердится ли сказанное при проверке гипотез? ([спойлер](#))

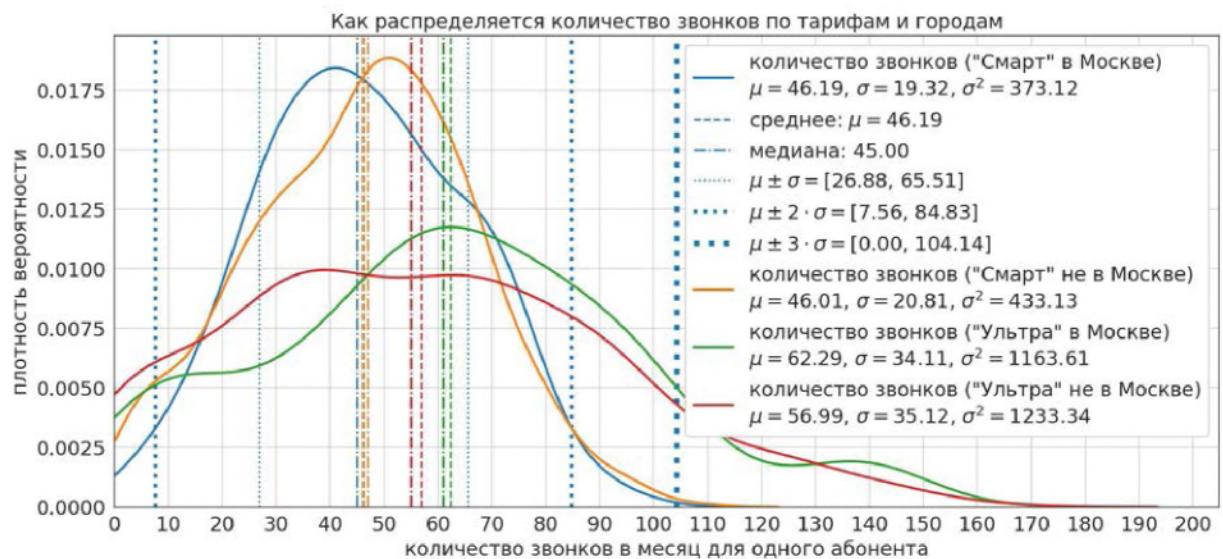
Тарифы в Москве и регионах

```
In [127...]: plot_billing('calls_total', ['"Смарт" в Москве', '"Смарт" не в Москве',
                                '"Ультра" в Москве', '"Ультра" не в Москве'],
                                ' по тарифам и городам', step=10, figsize=(16, 7))
```

Изучим, как распределяется количество звонков по тарифам и городам:

	"Смарт" в Москве	"Смарт" не в Москве	"Ультра" в Москве	"Ультра" не в Москве
количество образцов	363.000000	1702.000000	212.000000	702.000000
среднее значение	46.192837	46.005288	62.287736	56.987179
среднеквадратичное отклонение	19.316247	20.811731	34.111710	35.118972
минимум	2.000000	0.000000	0.000000	0.000000
25%	32.000000	31.000000	38.750000	31.000000
50%	45.000000	47.000000	61.000000	55.000000
75%	60.500000	61.000000	84.250000	81.000000
максимум	99.000000	109.000000	146.000000	165.000000

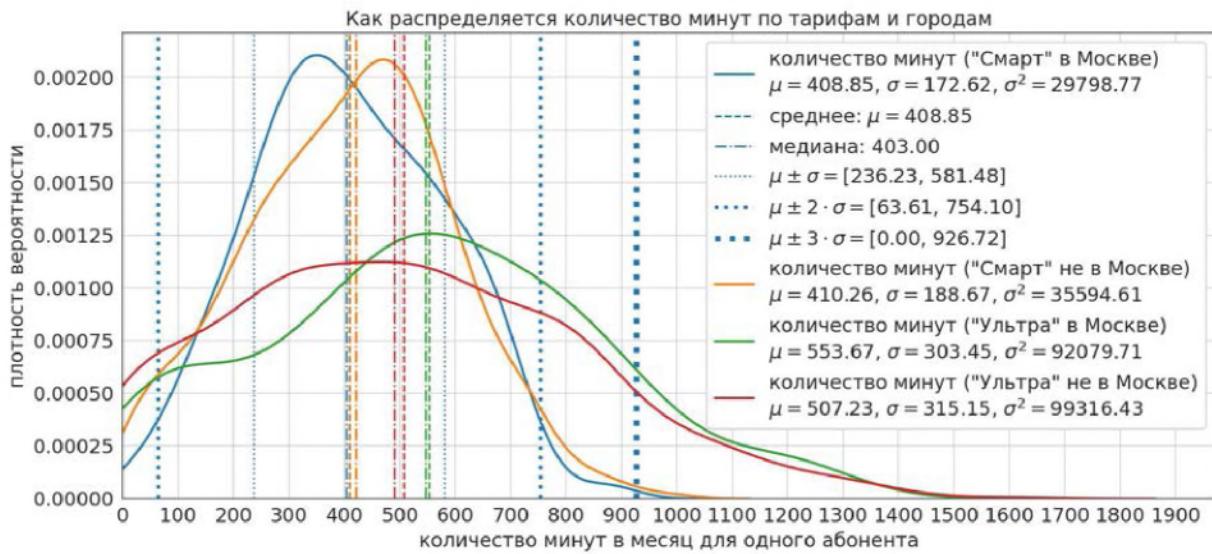
	"Смарт" в Москве	"Смарт" не в Москве	"Ультра" в Москве	"Ультра" не в Москве
дисперсия	373.117407	433.128132	1163.608759	1233.342203



```
In [128]: plot_billing('minutes_total', ['"Смарт" в Москве', '"Смарт" не в Москве',
                                    '"Ультра" в Москве', '"Ультра" не в Москве'],
                           ' по тарифам и городам', step=100, figsize=(16, 7))
```

Изучим, как распределяется количество минут по тарифам и городам:

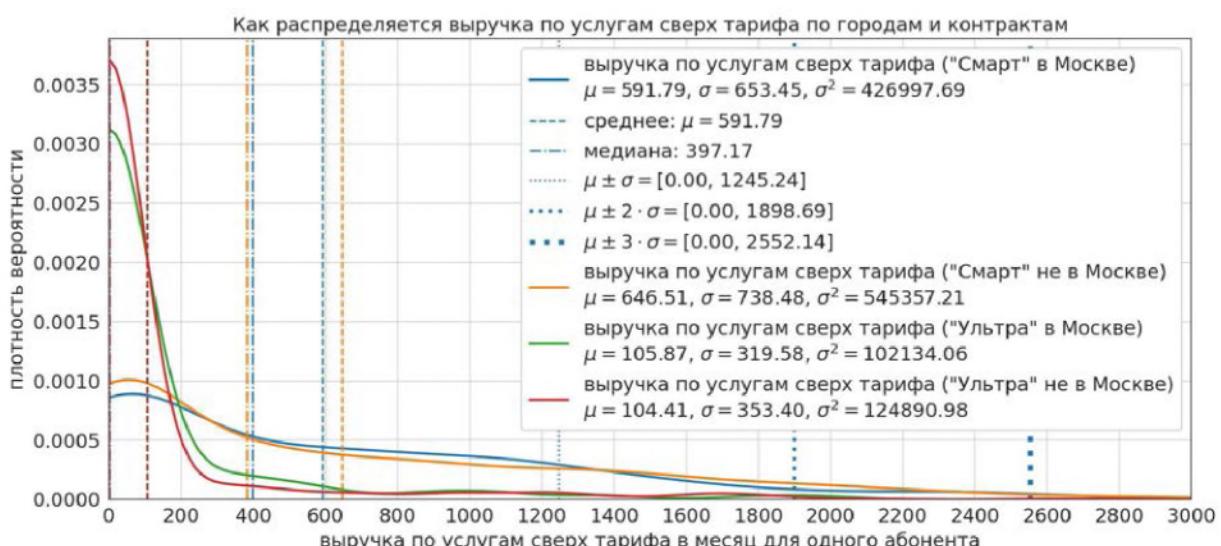
	"Смарт" в Москве	"Смарт" не в Москве	"Ультра" в Москве	"Ультра" не в Москве
количество образцов	363.000000	1702.000000	212.000000	702.000000
среднее значение	408.853994	410.255582	553.674528	507.229345
среднеквадратичное отклонение	172.623207	188.665342	303.446385	315.145091
минимум	12.000000	0.000000	0.000000	0.000000
25%	280.000000	277.250000	354.000000	271.250000
50%	403.000000	420.500000	547.000000	490.000000
75%	541.000000	538.000000	759.750000	732.000000
максимум	875.000000	1005.000000	1321.000000	1609.000000
дисперсия	29798.771441	35594.611300	92079.708732	99316.428068



```
In [129]: plot_billing('pay_extra', ['"Смарт" в Москве', '"Смарт" не в Москве',
                                '"Ультра" в Москве', '"Ультра" не в Москве'],
                                ' по городам и контрактам', max_x=3000, step=200, figsize=(16, 7))
```

Изучим, как распределяется выручка по услугам сверх тарифа по городам и контрактам:

	"Смарт" в Москве	"Смарт" не в Москве	"Ультра" в Москве	"Ультра" не в Москве
количество образцов	363.000000	1702.000000	212.000000	702.000000
среднее значение	591.787025	646.508514	105.873679	104.408333
среднеквадратичное отклонение	653.450604	738.483050	319.584206	353.399175
минимум	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000	0.000000
50%	397.170000	381.645000	0.000000	0.000000
75%	1005.055000	1074.257500	0.000000	0.000000
максимум	2825.780000	5023.550000	1995.260000	2793.310000
дисперсия	426997.692113	545357.214856	102134.064968	124890.977228

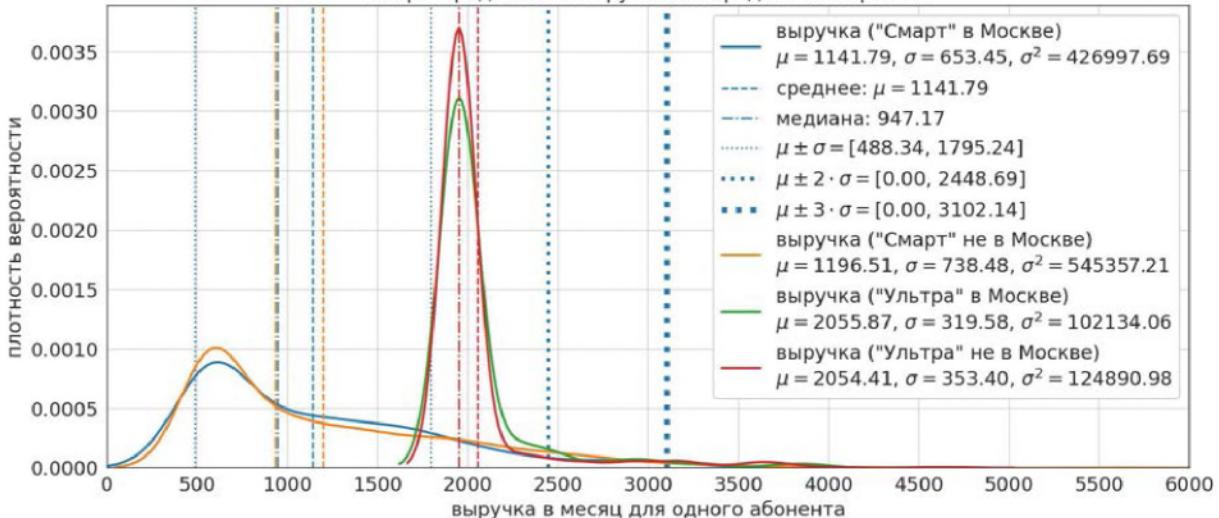


```
In [130...]: plot_billing('pay_total', ['"Смарт" в Москве', '"Смарт" не в Москве',
                                '"Ультра" в Москве', '"Ультра" не в Москве'],
                                ' по городам и контрактам', max_x=6000, step=500, figsize=(16, 7))
```

Изучим, как распределяется выручка по городам и контрактам:

	"Смарт" в Москве	"Смарт" не в Москве	"Ультра" в Москве	"Ультра" не в Москве
количество образцов	363.000000	1702.000000	212.000000	702.000000
среднее значение	1141.787025	1196.508514	2055.873679	2054.408333
среднеквадратичное отклонение	653.450604	738.483050	319.584206	353.399175
минимум	550.000000	550.000000	1950.000000	1950.000000
25%	550.000000	550.000000	1950.000000	1950.000000
50%	947.170000	931.645000	1950.000000	1950.000000
75%	1555.055000	1624.257500	1950.000000	1950.000000
максимум	3375.780000	5573.550000	3945.260000	4743.310000
дисперсия	426997.692113	545357.214856	102134.064968	124890.977228

Как распределяется выручка по городам и контрактам



Вывод по шагу 3

Как и следовало ожидать, большинство абонентов проживает в крупных городах. Это говорит о высокой релевантности выборки, которую предоставила нам сотовая компания.

В среднем пользователи говорят по телефону около 49-50 раз в месяц. На это указывает и среднее значение и медиана.

Средняя общая продолжительность разговоров в месяц у одного абонента достигает 443 минуты. Распределение имеет правый хвост - есть абоненты, которые говорят значительно больше остальной массы клиентов сотовой компании.

Четверть клиентов отправляет не больше 9 сообщений в месяц, половина - не более 30. Смещение среднего значения вправо относительно медианы и большая дисперсия говорит о существовании больших любителей коротких сообщений. Максимальное количество в месяц - 224.

Средний объем трафика в месяц - около 16.6 Гб. Это больше, чем лимит у тарифа "Смарт", и меньше "Ультры", поэтому имеются пользователи, которые платят дополнительную плату за трафик у более дешевого "Смarta" и не выбирают пакеты у премиального "Ультра". Разброс значений достаточно высок, но, как минимум, четверть пользователей не выходит за лимиты пакетов.

Интересное наблюдение - в первый месяц после заключения контракта абоненты совершают меньше звонков, чем в последующие периоды. Привыкшие к сотовой связи клиенты чаще выходят за лимиты по минутам разговоров в месяц, чем новые. Среднее количество сообщений от абонентов в первый месяц почти вдвое меньше, чем у более опытных пользователей. Средняя выручка от новых клиентов в месяц меньше.

Среднее количество звонков в месяц от абонентов различных тарифов отличается: Смарт - около 46, Ультра - около 58. Разброс значений у "богатого" Ультра шире.

Шаг 4. Проверка гипотез

[Задание описано выше](#)

Маркетологи компании Мегалайн, я уверен, постоянно думают, как оптимизировать свою стратегию управления тарифными планами. На начальных этапах планирования шагов по увеличению количества конверсий часто возникает необходимость в сравнении средней выручки от абонентов разных категорий. Мы им поможем!

Инструмент проверки гипотез

Опишем функцию, которая позволит автоматизировать проверку гипотез:

```
In [131...]: from scipy import stats as st
```

```
In [132...]: def check_mean_hypothesis(hypothesis, sample_1, sample_2, alpha=0.05, digits=2, *args):
    """Проверяет гипотезу о равенстве средних выборок sample_1, sample_2
    alpha - критический уровень статистической значимости
    если p-value окажется меньше него - гипотеза отвергается
    digits - количество цифр в дробной части для вывода на экран

    Возвращает:
        true - гипотеза не отвергнута
        false - гипотеза отвергнута"""

    results = st.ttest_ind(sample_1, sample_2, *args, **kwargs)

    if results.pvalue < alpha:
        verdict = 'отвергнута'
    else:
        verdict = 'не может быть отвергнута'

    display(HTML(f'Гипотеза {hypothesis} {verdict} по результатам проверки '
                f'с критическим уровнем статистической значимости {alpha} '
                f'(получено p-значение {results.pvalue:.{digits}f).')))

    return results.pvalue >= alpha
```

Проверим правильность её работы на тестовом примере из урока от [Яндекс.Практикум](#).

```
In [133...]: sample_1 = [3071, 3636, 3454, 3151, 2185, 3259, 1727, 2263, 2015,
                  2582, 4815, 633, 3186, 887, 2028, 3589, 2564, 1422, 1785,
                  3180, 1770, 2716, 2546, 1848, 4644, 3134, 475, 2686,
```

```
1838, 3352]
sample_2 = [1211, 1228, 2157, 3699, 600, 1898, 1688, 1420, 5048, 3007,
509, 3777, 5583, 3949, 121, 1674, 4300, 1338, 3066,
3562, 1010, 2311, 462, 863, 2021, 528, 1849, 255,
1740, 2596]
```

Для указанных значений гипотеза не должна быть отвергнута при критическом уровне статистической значимости в 0.001:

```
In [134...]: check_mean_hypothesis('о равенстве средних значений двух проверочных выборок', samp
```

Гипотеза о равенстве средних значений двух проверочных выборок не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.001** (получено р-значение **0.191**).

```
Out[134...]: True
```

Наш "инструмент проверки гипотез" возвратил правильное значение. Скорее применим его, а то заказчик мог уже и утомиться ожидать результат!

Выбор уровня значимости

Уровень значимости статистического теста — допустимая для данной задачи вероятность ошибки первого рода (ложноположительного решения, false positive), то есть вероятность отклонить **нулевую гипотезу**, когда на самом деле она верна.

Другая интерпретация: **уровень значимости** — это такое (достаточно малое) значение вероятности события, при котором событие уже можно считать неслучайным.

Уровень значимости обычно обозначают греческой буквой α (альфа).

Обычно рекомендуется выбирать уровень значимости из априорных соображений. Однако на практике не вполне ясно, какими именно соображениями надо руководствоваться, и выбор часто сводится к назначению одного из популярных вариантов $\alpha = 0.005, 0.01, 0.05, 0.1$. В докомпьютерную эпоху эта стандартизация позволяла сократить объём справочных статистических таблиц. Теперь нет никаких специальных причин для выбора именно этих значений.

Наш дивиз остается прежним: "**Мы не будем полагаться на случай !**". В нашем проекте будем минимизировать вероятность отклонить нулевую гипотезу, когда на самом деле она верна. Для этого присвоим α маленькое значение:

```
In [135...]: alpha=0.01
```

Проверка гипотезы о среднемесячной выручке по тарифам

Сформулируем гипотезы:

- нулевая гипотеза H_0 : **средняя выручка в месяц от пользователей тарифов «Ультра» и «Смарт» одинакова;**
- альтернативная гипотеза H_1 : **средняя выручка в месяц от пользователей тарифов «Ультра» и «Смарт» существенно отличается.**

Проверим гипотезы:

```
In [136...]: _ = check_mean_hypothesis('о равенстве средней выручки пользователей тарифов «Ультра
```

```
billing_slice('тариф "Ультра"').pay_total,  
billing_slice('тариф "Смарт"').pay_total,  
alpha=alpha, digits=10)
```

Гипотеза о равенстве средней выручки пользователей тарифов «Ультра» и «Смарт» отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0000000000**).

Значит у нас нет оснований говорить, что пользователи тарифов «Ультра» и «Смарт» тратят одинаковые суммы в месяц. Судя по р-значению, размер оплаты значительно отличается. Подтверждается это и исследованием, проведенным [выше](#). Основная причина кроется в разнице абонентской платы в месяц у сравниваемых [тарифов](#), размер которой равен 1400 рублям.

Проверка гипотезы о среднемесячной выручке по регионам

Сформулируем гипотезы:

- нулевая гипотеза H_0 : **средняя выручка в месяц, получаемая от абонентов из Москвы, не отличается от аналогичного показателя в других регионах;**
- альтернативная гипотеза H_1 : **средняя выручка в месяц от жителей столицы и регионов отличается.**

Проверим гипотезы:

```
In [137...]  
_ = check_mean_hypothesis('о равенстве средней выручки в месяц от абонентов из Москв  
billing_slice('Москва').pay_total,  
billing_slice('не Москва').pay_total,  
alpha=alpha, digits=2)
```

Гипотеза о равенстве средней выручки в месяц от абонентов из Москвы и других регионов не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.36**).

Значит, скорее всего, для генеральной совокупности всех пользователей среднее значение выручки в месяц от абонентов в Москве и других городах отличается совершенно несущественно. Даже на [графиках](#) плотности распределения выглядят очень похоже. **Но в чем же тогда природа отличий?** Попробуем разобраться через один абзац.



Правильно! Наши выборки на основе места жительства абонентов **нестратифицированные** в смысле соотношения тарифных планов. Рассмотрим пример на одном из тарифов. В столице доля платежей от абонентов с тарифным планом "Ультра" составляет около 37%, а в регионах - чуть более 29%.

```
In [138...]: ultra_portion = (billing_slice('Москва').tariff == 'Ультра').mean()  
ultra_portion
```

```
Out[138...]: 0.36869565217391304
```

```
In [139...]: ultra_portion = (billing_slice('не Москва').tariff == 'Ультра').mean()
```

Центральная предельная теорема нивелировала этот дисбаланс и мы получили похожие распределения средней выручки в Москве и других городах для указанного выше сочетания. Но вот в случае, если бы мы окончательно смешали "*красное с тяжелым*"...

```
In [140...]: _ = check_mean_hypothesis('о равенстве средней выручки в месяц от абонентов из Москвы  
        с тарифным планом "Ультра" из других городов',  
        billing_slice('Москва').pay_total,  
        billing_slice('"Ультра" не в Москве').pay_total,  
        alpha=alpha, digits=10)
```

Гипотеза о равенстве средней выручки в месяц от абонентов из Москвы и клиентов с тарифным планом "Ультра" из других городов отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0000000000**).

... то, возможно, имели бы красное лицо и получили бы по голове чем-нибудь тяжелым. **Мы не будем полагаться на случай!** Стратифицируем наши выборки и обеспечим баланс классов абонентов по тарифным планам.

Выборку по Москве оставим прежнюю (она меньше), а из абонентов, живущих в других городах, отберем данные так, чтобы доли тарифов были, как в столице. Найдем необходимое количество платежей по Смарту в регионах из пропорции:

```
In [141... new_smart_count = round((billing_slice('не Москва').tariff == 'Ультра').sum() *  
                           (billing_slice('Москва').tariff == 'Смарт').sum() /  
                           (billing_slice('Москва').tariff == 'Ультра').sum())
```

Набор для тарифа Ультра оставляем без изменений:

```
In [142... b = billing_slice('не Москва')
```

```
In [143... ultra_indices = b[b.tariff == 'Ультра'].index
```

А дальше проведем четыре проверки гипотезы и посмотрим, как будет меняться результат в зависимости от того, какие платежи по Смарту в регионах совершенно случайным образом попадут в выборку с помощью `np.random.choice`:

```
In [144... for i, s in enumerate([18112020, 0, 555, 777]):  
    # настраиваем генератор случайных чисел  
    np.random.seed(s)  
  
    # отбираем платежи случайным образом  
    smart_indices = np.random.choice(b[b.tariff == 'Смарт'].index,  
                                      size=new_smart_count,  
                                      replace=False)  
    balanced_indices = ultra_indices.to_list()  
  
    # объединяем Ультра и Смарт в регионах  
    balanced_indices.extend(smart_indices)  
  
    # получаем итоговую стратифицированную выборку по регионам  
    non_moscow_balanced = b.loc[balanced_indices]  
  
    # Выполняем проверку гипотезы для очередной выборки  
    _ = check_mean_hypothesis('о равенстве средней выручки в месяц от абонентов из М  
                                ф'стратифицированной выборки № {i + 1} из других регио  
                                billing_slice('Москва').pay_total,  
                                non_moscow_balanced.pay_total,  
                                alpha=alpha, digits=2)
```

Гипотеза о равенстве средней выручки в месяц от абонентов из Москвы и стратифицированной выборки № 1 из других регионов не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.41**).

Гипотеза о равенстве средней выручки в месяц от абонентов из Москвы и стратифицированной выборки № 2 из других регионов не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.38**).

Гипотеза о равенстве средней выручки в месяц от абонентов из Москвы и стратифицированной выборки № 3 из других регионов не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.32**).

Гипотеза о равенстве средней выручки в месяц от абонентов из Москвы и стратифицированной выборки № 4 из других регионов не может быть отвергнута

по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.45**).

Обратим внимание на изменение р-значения для разных выборок - алгоритм с разной степенью уверенности не отвергал нашу гипотезу.

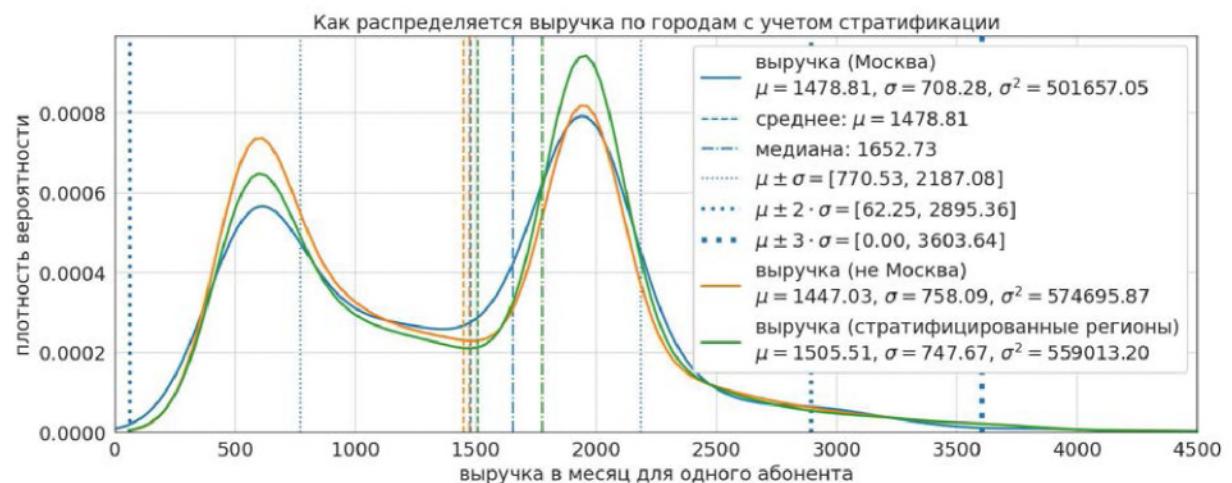
А мы теперь с еще большей уверенностью можем заключить, что в целом **средняя выручка в месяц от абонентов из Москвы и регионов мало отличается**.

Убедимся, что порядок формирования выборки и стратификация оказывает определенное влияние на графики распределения:

```
In [145...]: filters['стратифицированные регионы'] = billing.index.isin(balanced_indices)
plot_billing('pay_total', ['Москва', 'не Москва', 'стратифицированные регионы'],
             ' по городам с учетом стратификации', max_x=4500)
```

Изучим, как распределяется выручка по городам с учетом стратификации:

	Москва	не Москва	стратифицированные регионы
количество образцов	575.000000	2404.000000	1904.000000
среднее значение	1478.806800	1447.026681	1505.509554
среднеквадратичное отклонение	708.277525	758.086979	747.671854
минимум	550.000000	550.000000	550.000000
25%	719.500000	658.000000	724.000000
50%	1652.730000	1473.240000	1775.840000
75%	1950.000000	1950.000000	1950.000000
максимум	3945.260000	5573.550000	5573.550000
дисперсия	501657.053064	574695.867697	559013.201653



Мы добавили к уже [знакомому](#) чертежу график плотности распределения месячной выручки по стратифицированной выборке из регионов. Зеленый график "отыграл" корректировку пропорции клиентов с разными тарифами предсказуемым изменением высоты своих "горбов". Но средние значения выборок (пунктирные линии) так и остались поблизости друг от друга. Гипотеза сформулирована, проверена, доказана, перепроверена и теперь не вызывает сомнений, по крайней мере, у [автора](#)!

Гипотезы о Москве и других городах

Сформулируем и проверим ряд дополнительных гипотез, которые помогут нам лучше понять поведение пользователей сотовой связи:

```
In [146...]  
_ = check_mean_hypothesis('о равенстве среднего количества звонков от абонентов Москва  
billing_slice('Москва').calls_total,  
billing_slice('не Москва').calls_total,  
alpha=alpha, digits=2)
```

Гипотеза о равенстве среднего количества звонков от абонентов Москвы и других городов в месяц не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.02**).

```
In [147...]  
_ = check_mean_hypothesis('о равенстве среднего количества минут разговоров от або  
billing_slice('Москва').minutes_total,  
billing_slice('не Москва').minutes_total,  
alpha=alpha, digits=2)
```

Гипотеза о равенстве среднего количества минут разговоров от абонентов Москвы и других городов в месяц не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.03**).

```
In [148...]  
_ = check_mean_hypothesis('о равенстве среднего количества сообщений от абонентов Мо  
billing_slice('Москва').messages_total,  
billing_slice('не Москва').messages_total,  
alpha=alpha, digits=2)
```

Гипотеза о равенстве среднего количества сообщений от абонентов Москвы и других городов в месяц не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.59**).

```
In [149...]  
_ = check_mean_hypothesis('о равенстве среднего объема интернет-трафика в месяц у аб  
billing_slice('Москва').mb_total,  
billing_slice('не Москва').mb_total,  
alpha=alpha, digits=10)
```

Гипотеза о равенстве среднего объема интернет-трафика в месяц у абонентов Москвы и других городов отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0039657665**).

Гипотезы о разных тарифах

```
In [150...]  
_ = check_mean_hypothesis('о равенстве среднего количества звонков в месяц от абонен  
billing_slice('тариф "Смарт"').calls_total,  
billing_slice('тариф "Ультра"').calls_total,  
alpha=alpha, digits=2)
```

Гипотеза о равенстве среднего количества звонков в месяц от абонентов различных тарифов отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.00**).

```
In [151...]  
_ = check_mean_hypothesis('о равенстве средней выручки в месяц от абонентов тарифа "  
billing_slice('"Смарт" в Москве").pay_total,  
billing_slice('"Смарт" не в Москве").pay_total,  
alpha=alpha, digits=2)
```

Гипотеза о равенстве средней выручки в месяц от абонентов тарифа "Смарт" из Москвы и других регионов не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.19**).

```
In [152...]: _ = check_mean_hypothesis('о равенстве средней выручки в месяц от абонентов тарифа "Ультра" в Москве и не в Москве',  
                                billing_slice('"Ультра" в Москве").pay_total,  
                                billing_slice('"Ультра" не в Москве").pay_total,  
                                alpha=alpha, digits=2)
```

Гипотеза о равенстве средней выручки в месяц от абонентов тарифа "Ультра" из Москвы и других регионов не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.96**).

```
In [153...]: _ = check_mean_hypothesis('о равенстве среднего объема интернет-трафика в месяц у абонентов тарифа "Смарт" Москвы и других городов',  
                                billing_slice('"Смарт" в Москве").mb_total,  
                                billing_slice('"Смарт" не в Москве").mb_total,  
                                alpha=alpha, digits=2)
```

Гипотеза о равенстве среднего объема интернет-трафика в месяц у абонентов тарифа "Смарт" Москвы и других городов не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.60**).

```
In [154...]: _ = check_mean_hypothesis('о равенстве среднего объема интернет-трафика в месяц у абонентов тарифа "Ультра" Москвы и других городов',  
                                billing_slice('"Ультра" в Москве").mb_total,  
                                billing_slice('"Ультра" не в Москве").mb_total,  
                                alpha=alpha, digits=10)
```

Гипотеза о равенстве среднего объема интернет-трафика в месяц у абонентов тарифа "Ультра" Москвы и других городов отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0017547336**).

```
In [155...]: _ = check_mean_hypothesis('о равенстве среднего количества минут телефонных разговоров у абонентов разных тарифов',  
                                billing_slice('тариф "Смарт"').minutes_total,  
                                billing_slice('тариф "Ультра"').minutes_total,  
                                alpha=alpha, digits=10)
```

Гипотеза о равенстве среднего количества минут телефонных разговоров в месяц у абонентов разных тарифов отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0000000000**).

Гипотезы о новых абонентах

```
In [156...]: _ = check_mean_hypothesis('о равенстве средней выручки в месяц от новых абонентов и длительностью более двух месяцев',  
                                billing_slice('первый месяц').pay_total,  
                                billing_slice('опытные пользователи').pay_total,  
                                alpha=alpha, digits=2)
```

Гипотеза о равенстве средней выручки в месяц от новых абонентов и контрактов длительностью более двух месяцев отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.00**).

```
In [157...]: _ = check_mean_hypothesis('о равенстве среднего количества звонков в месяц от новых абонентов и длительностью более двух месяцев',  
                                billing_slice('первый месяц').calls_total,  
                                billing_slice('опытные пользователи').calls_total,  
                                alpha=alpha, digits=2)
```

Гипотеза о равенстве среднего количества звонков в месяц от новых абонентов и

контрактов длительностью более двух месяцев отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.00**).

In [158...]

```
_ = check_mean_hypothesis('о равенстве среднего объема интернет-трафика в месяц от новых длительностью более двух месяцев',
                           billing_slice('первый месяц').mb_total,
                           billing_slice('опытные пользователи').mb_total,
                           alpha=alpha, digits=2)
```

Гипотеза о равенстве среднего объема интернет-трафика в месяц от новых абонентов и контрактов длительностью более двух месяцев отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.00**).

Вывод по шагу 4

- разработан [инструмент](#) автоматизированной проверки гипотез о средних значениях срезов биллинга;
- уровень статистической значимости [задан](#) в интересах минимизации вероятности ошибок первого рода ($\alpha = 0.01$);
- в результате проверки заданных в задании гипотез установлено:
 - [средняя выручка](#) пользователей тарифов «Ультра» и «Смарт» различается;
 - [средняя выручка](#) пользователей из Москвы не отличается от выручки пользователей из других регионов;
- в качестве маркетингового исследования проведена проверка дополнительной [серии гипотез](#);
- результаты исследования готовы к обобщению в [общем выводе](#).

Шаг 5. Общий вывод

Полученные для анализа файлы имеют корректный формат. Массив был проанализирован после минимальной предобработки.

Как и следовало ожидать, большинство абонентов проживает в крупных городах. Это говорит о высокой релевантности выборки, которую предоставила нам сотовая компания.

В среднем пользователи говорят по телефону около 49-50 раз в месяц. На это указывает и среднее значение и медиана.

Среднее количество звонков в месяц от абонентов различных тарифов отличается: Смарт - около 46, Ультра - около 58. Разброс значений у "богатого" Ультра шире.

Средняя общая продолжительность разговоров в месяц у одного абонента достигает 443 минуты. Распределение имеет правый хвост - есть абоненты, которые говорят значительно больше остальной массы клиентов сотовой компании.

Четверть клиентов отправляет не больше 9 сообщений в месяц, половина - не более 30. Смещение среднего значения вправо относительно медианы и большая дисперсия говорит о существовании больших любителей коротких сообщений (максимальное количество в месяц достигает 224).

Средний объем трафика в месяц - около 16.6 Гб. Это больше, чем лимит у тарифа "Смарт", и меньше "Ультры", поэтому имеются пользователи, которые платят дополнительную плату за трафик у более дешевого "Смarta" и не выбирают пакеты у премиального "Ультра". Разброс значений достаточно высок, но, как минимум, четверть пользователей не выходит за лимиты пакетов.

В результате проверки [заданных заказчиком](#) гипотез установлено:

- [средняя выручка](#) пользователей тарифов «Ультра» (2054.75 руб.) и «Смарт» (1186.89 руб.) [различаются](#) существенно;
- [средняя выручка](#) пользователей из Москвы (1478.81 руб.) существенно [не отличается](#) от выручки пользователей из других регионов (1447.03 руб.).

Проверка дополнительной [серии гипотез](#) показала следующее:

- средние выручки в месяц от абонентов одинакового тарифа из Москвы и других регионов практически равны;
- средний объем интернет-трафика в месяц у абонентов Москвы (17.36 Гб) и других городов (16.37 Гб) отличается, но:
 - средний объем интернет-трафика в месяц у абонентов тарифа "Смарт" Москвы и других городов практически одинаков;
 - средний объем интернет-трафика в месяц у абонентов тарифа "Ультра" Москвы и других городов отличается;
- среднее количество звонков в месяц от абонентов различных тарифов отличается (Смарт - около 46, Ультра - около 58).

Интересное наблюдение - поведение клиентов меняется после первого месяца пользования услугами связи:

- в первый месяц после заключения контракта абоненты совершают меньше звонков (в среднем около 28), чем в последующие периоды (54);
- привыкшие к сотовой связи клиенты чаще выходят за лимиты по минутам разговоров в месяц, чем новые;
- среднее количество сообщений от абонентов в первый месяц (22) почти вдвое меньше, чем у более опытных пользователей (40);
- как следствие средняя выручка в месяц от новых абонентов (1045.52 руб.) и контрактов длительностью более двух месяцев (1522.46 руб.) значимо отличаются.

При определении перспективности двух исследуемых тарифов следует также принимать во внимание следующее:

- пользователи тарифа "Ультра" с большими пакетами услуг,ключенными в абонентскую плату, совершенно не выходят за лимиты по минутам и сообщениям;
- за дополнительные минуты и сообщения в месяц платят пользователи тарифа "Смарт" с меньшими пакетами услуг,ключенными в абонентскую плату;
- пользователи тарифа "Ультра" имеют шанс превысить лимит лишь по интернет-трафику.

Оба тарифа являются достаточно перспективными, но их продвижение целесообразно осуществлять по различным маркетинговым каналам, разделенным по возрастанию

уровня дохода конечного потребителя, так как [средние выручки](#) в месяц от пользователей тарифов «Ультра» (2054.75 руб.) и «Смарт» (1186.89 руб.) существенно [различаются](#).

Таблица контроля версий

Версия	Дата	Описание
1.0	19.11.2020	Направлено на первичную проверку

Планы на будущие проекты

Реализовать универсальную функцию проверки гипотез о средних значениях выборок с одновременным выводом графиков, параметров распределения и текстового заключения в виде, удобном для включения в общий вывод исследования (Данный проект выполнялся по шагам в соответствии с заданием. Использование подобной функции на данном этапе обучения могло затруднить проверку.)

[К оглавлению](#)

Бонус

По ходу повествования у [автора](#) и читателей неоднократно возникал вопрос: "А стоило ли так заморачиваться с учетом даты выставления счета индивидуально для каждого клиента?"

Хотелось бы напомнить, что ответ преподавателей и студентов [Яндекс.Практикума](#) отличается от медианных значений, [сгенерированных](#) этим героям:



Учет даты выставления счета абоненту позволил не только избежать искажений в расчетах, но и сделал возможным автоматизированную подготовку отчетных документов для сверки с заказчиком в лице сотовой компании Мегалайн. А что может лучше убедить его, чем учет каждой его копейки?!

Счет за услуги связи

Опишем функцию выставления счетов за услуги связи:

```
In [159...]: def print_bill(billing, user_id):
    """Печатает счета за услуги связи по номеру пользователя"""
    def print_package(package, total, included, free, charged, price, unit, pay):
        if total > 0:
            print(f'{package:>24}: {total:>8}, в т.ч. по пакету {free} из {included}')
            if charged > 0:
                print(f' и за отдельную плату {charged}')
            else:
                print()
        if pay > 0:
            print('{:>24s}: {:.02f} руб. по цене {:.02f} руб. за {:s}'.format(
                'к оплате', pay, price, unit))

    user_info = table_users[table_users.user_id == user_id]
    if user_info.shape[0] == 0:
        print(f'Пользователь с номером {user_id} не найден.')
        return

    user_info = next(user_info.itertuples())

    user_bills = billing[billing.user_id == user_id]
    user_bills.set_index('month', inplace=True)
    for row in user_bills.itertuples():
        print(f'\t\tСчет за услуги, оказанные с {row.date_from:%d.%m.%Y} по {row.dat
print('{:>24s}: {} {} ({})'.format('абонент', user_info.last_name, user_info
print('{:>24s}: {}, договор заключен {:%d.%m.%Y}'.format('тариф', row.tariff
        if row.calls_total > 0:
            print('{:>24s}: {:8d}'.format('звонков', row.calls_total))
            print_package('минут разговора', row.minutes_total, row.minutes_included,
                          row.minutes_free, row.minutes_charged, row.rub_per_minute,
                          '1 минуту', row.pay_minutes)
            print_package('сообщений', row.messages_total, row.messages_included,
                          row.messages_free, row.messages_charged, row.rub_per_message,
                          '1 сообщение', row.pay_messages)
            print_package('интернет-трафик (Мб)', row.mb_total, row.mb_per_month_include
                          row.mb_free, row.mb_charged, row.rub_per_gb, '1 Гб', row.pay_i
        print('{:>24s}: {:.02f} руб.'.format('абонентская плата', row.rub_monthly_f
        if row.pay_extra:
            print('{:>24s}: {:.02f} руб.'.format('за дополнительные услуги', row.pa
        print('{:>24s}: {:.02f} руб.'.format('ИТОГО К ОПЛАТЕ', row.pay_total))
        print()
```

Несчастливый клиент

Помните [пользователя](#), которому сложно дозвониться до знакомых. Причина ясна - он и его друзья не пользуются сотовой связью, а сидят в интернет-мессенджерах (трафик каждый месяц). А сообщения, скорее всего, Габриэль отправляет лишь для подтверждения перевода денежных средств через онлайн-банкинг.

```
In [160...]: print_bill(billing, unlucky_user_id)
```

```
СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 17.06.2018 ПО 16.07.2018
абонент: Жданов Габриэль (Санкт-Петербург)
тариф: Ультра, договор заключен 17.06.2018
звонков: 2
минут разговора: 35, в т.ч. по пакету 35 из 3000
сообщений: 62, в т.ч. по пакету 62 из 1000
интернет-трафик (Мб): 11787, в т.ч. по пакету 11787 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.
```

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 17.07.2018 ПО 16.08.2018
абонент: Жданов Габриель (Санкт-Петербург)
тариф: Ультра, договор заключен 17.06.2018
сообщений: 146, в т.ч. по пакету 146 из 1000
интернет-трафик (Мб): 20034, в т.ч. по пакету 20034 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 17.08.2018 ПО 16.09.2018
абонент: Жданов Габриель (Санкт-Петербург)
тариф: Ультра, договор заключен 17.06.2018
сообщений: 157, в т.ч. по пакету 157 из 1000
интернет-трафик (Мб): 25506, в т.ч. по пакету 25506 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 17.09.2018 ПО 16.10.2018
абонент: Жданов Габриель (Санкт-Петербург)
тариф: Ультра, договор заключен 17.06.2018
сообщений: 153, в т.ч. по пакету 153 из 1000
интернет-трафик (Мб): 22777, в т.ч. по пакету 22777 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 17.10.2018 ПО 16.11.2018
абонент: Жданов Габриель (Санкт-Петербург)
тариф: Ультра, договор заключен 17.06.2018
звонков: 2
минут разговора: 16, в т.ч. по пакету 16 из 3000
сообщений: 131, в т.ч. по пакету 131 из 1000
интернет-трафик (Мб): 20359, в т.ч. по пакету 20359 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 17.11.2018 ПО 16.12.2018
абонент: Жданов Габриель (Санкт-Петербург)
тариф: Ультра, договор заключен 17.06.2018
звонков: 1
минут разговора: 6, в т.ч. по пакету 6 из 3000
сообщений: 133, в т.ч. по пакету 133 из 1000
интернет-трафик (Мб): 26274, в т.ч. по пакету 26274 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

Беззаботный пользователь

Есть люди, которые не очень думают о сохранении своего бюджета. Найдем клиента, который больше всех остальных заплатил за дополнительные услуги (сверх лимитов минут, сообщений и интернет-трафика по своему тарифному плану). Кто ты беззаботный пользователь, на котором держится весь Мегалайн?

```
In [161]: print_bill(billing, billing.groupby('user_id').pay_extra.sum().idxmax())
```

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.01.2018 ПО 26.02.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 6
минут разговора: 48, в т.ч. по пакету 48 из 500
сообщений: 9, в т.ч. по пакету 9 из 50
интернет-трафик (Мб): 3766, в т.ч. по пакету 3766 из 15360
абонентская плата: 550.00 руб.
ИТОГО К ОПЛАТЕ: 550.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.02.2018 ПО 26.03.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018

94 звонков: 65
минут разговора: 594, в т.ч. по пакету 500 из 500 и за отдельную плату
к оплате: 282.00 руб. по цене 3.00 руб. за 1 минуту
сообщений: 45, в т.ч. по пакету 45 из 50
интернет-трафик (Мб): 28523, в т.ч. по пакету 15360 из 15360 и за отдельную плату 13163
к оплате: 2570.90 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 2852.90 руб.
ИТОГО К ОПЛАТЕ: 3402.90 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.03.2018 ПО 26.04.2018
57 абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 67
минут разговора: 557, в т.ч. по пакету 500 из 500 и за отдельную плату
к оплате: 171.00 руб. по цене 3.00 руб. за 1 минуту
сообщений: 46, в т.ч. по пакету 46 из 50
интернет-трафик (Мб): 24990, в т.ч. по пакету 15360 из 15360 и за отдельную плату 9630
к оплате: 1880.86 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 2051.86 руб.
ИТОГО К ОПЛАТЕ: 2601.86 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.04.2018 ПО 26.05.2018
21 абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 52
минут разговора: 521, в т.ч. по пакету 500 из 500 и за отдельную плату
к оплате: 63.00 руб. по цене 3.00 руб. за 1 минуту
сообщений: 49, в т.ч. по пакету 49 из 50
интернет-трафик (Мб): 21098, в т.ч. по пакету 15360 из 15360 и за отдельную плату 5738
к оплате: 1120.70 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 1183.70 руб.
ИТОГО К ОПЛАТЕ: 1733.70 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.05.2018 ПО 26.06.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 50
минут разговора: 484, в т.ч. по пакету 484 из 500
сообщений: 40, в т.ч. по пакету 40 из 50
интернет-трафик (Мб): 23740, в т.ч. по пакету 15360 из 15360 и за отдельную плату 8380
к оплате: 1636.72 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 1636.72 руб.
ИТОГО К ОПЛАТЕ: 2186.72 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.06.2018 ПО 26.07.2018
55 абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 57
минут разговора: 555, в т.ч. по пакету 500 из 500 и за отдельную плату
к оплате: 165.00 руб. по цене 3.00 руб. за 1 минуту
сообщений: 35, в т.ч. по пакету 35 из 50
интернет-трафик (Мб): 20163, в т.ч. по пакету 15360 из 15360 и за отдельную плату 4803
к оплате: 938.09 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 1103.09 руб.
ИТОГО К ОПЛАТЕ: 1653.09 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.07.2018 ПО 26.08.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 63
минут разговора: 582, в т.ч. по пакету 500 из 500 и за отдельную плату
82
к оплате: 246.00 руб. по цене 3.00 руб. за 1 минуту
сообщений: 42, в т.ч. по пакету 42 из 50
интернет-трафик (Мб): 24395, в т.ч. по пакету 15360 из 15360 и за отдельную плату 9035
к оплате: 1764.65 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 2010.65 руб.
ИТОГО К ОПЛАТЕ: 2560.65 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.08.2018 ПО 26.09.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 62
минут разговора: 527, в т.ч. по пакету 500 из 500 и за отдельную плату
27
к оплате: 81.00 руб. по цене 3.00 руб. за 1 минуту
сообщений: 41, в т.ч. по пакету 41 из 50
интернет-трафик (Мб): 23560, в т.ч. по пакету 15360 из 15360 и за отдельную плату 8200
к оплате: 1601.56 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 1682.56 руб.
ИТОГО К ОПЛАТЕ: 2232.56 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.09.2018 ПО 26.10.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 58
минут разговора: 486, в т.ч. по пакету 486 из 500
сообщений: 29, в т.ч. по пакету 29 из 50
интернет-трафик (Мб): 26970, в т.ч. по пакету 15360 из 15360 и за отдельную плату 11610
к оплате: 2267.58 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 2267.58 руб.
ИТОГО К ОПЛАТЕ: 2817.58 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.10.2018 ПО 26.11.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 56
минут разговора: 450, в т.ч. по пакету 450 из 500
сообщений: 35, в т.ч. по пакету 35 из 50
интернет-трафик (Мб): 28215, в т.ч. по пакету 15360 из 15360 и за отдельную плату 12855
к оплате: 2510.74 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.
за дополнительные услуги: 2510.74 руб.
ИТОГО К ОПЛАТЕ: 3060.74 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 27.11.2018 ПО 26.12.2018
абонент: Блохин Трифон (Вологда)
тариф: Смарт, договор заключен 27.01.2018
звонков: 66
минут разговора: 662, в т.ч. по пакету 500 из 500 и за отдельную плату
162
к оплате: 486.00 руб. по цене 3.00 руб. за 1 минуту
сообщений: 46, в т.ч. по пакету 46 из 50
интернет-трафик (Мб): 25092, в т.ч. по пакету 15360 из 15360 и за отдельную плату 9732
к оплате: 1900.78 руб. по цене 200.00 руб. за 1 Гб
абонентская плата: 550.00 руб.

за дополнительные услуги: 2386.78 руб.
ИТОГО К ОПЛАТЕ: 2936.78 руб.

Король Интернета

А кому мы предложим скидки на онлайн-телевидение от нашего партнера [Яндекс.Эфир?](#)
Правильно, лидеру по объему трафика:

```
In [162]: print_bill(billing, billing.groupby('user_id').mb_total.sum().idxmax())
```

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.01.2018 ПО 12.02.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 30
минут разговора: 270, в т.ч. по пакету 270 из 3000
сообщений: 72, в т.ч. по пакету 72 из 1000
интернет-трафик (Мб): 19297, в т.ч. по пакету 19297 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.02.2018 ПО 12.03.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 36
минут разговора: 277, в т.ч. по пакету 277 из 3000
сообщений: 90, в т.ч. по пакету 90 из 1000
интернет-трафик (Мб): 32655, в т.ч. по пакету 30720 из 30720 и за отдельную плату 1935
к оплате: 283.45 руб. по цене 150.00 руб. за 1 Гб
абонентская плата: 1950.00 руб.
за дополнительные услуги: 283.45 руб.
ИТОГО К ОПЛАТЕ: 2233.45 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.03.2018 ПО 12.04.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 36
минут разговора: 310, в т.ч. по пакету 310 из 3000
сообщений: 123, в т.ч. по пакету 123 из 1000
интернет-трафик (Мб): 31278, в т.ч. по пакету 30720 из 30720 и за отдельную плату 558
к оплате: 81.74 руб. по цене 150.00 руб. за 1 Гб
абонентская плата: 1950.00 руб.
за дополнительные услуги: 81.74 руб.
ИТОГО К ОПЛАТЕ: 2031.74 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.04.2018 ПО 12.05.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 39
минут разговора: 348, в т.ч. по пакету 348 из 3000
сообщений: 96, в т.ч. по пакету 96 из 1000
интернет-трафик (Мб): 28968, в т.ч. по пакету 28968 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.05.2018 ПО 12.06.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 41
минут разговора: 343, в т.ч. по пакету 343 из 3000
сообщений: 127, в т.ч. по пакету 127 из 1000
интернет-трафик (Мб): 29612, в т.ч. по пакету 29612 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.06.2018 ПО 12.07.2018

абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 37
минут разговора: 356, в т.ч. по пакету 356 из 3000
сообщений: 103, в т.ч. по пакету 103 из 1000
интернет-трафик (Мб): 27373, в т.ч. по пакету 27373 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.07.2018 ПО 12.08.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 41
минут разговора: 383, в т.ч. по пакету 383 из 3000
сообщений: 142, в т.ч. по пакету 142 из 1000
интернет-трафик (Мб): 37589, в т.ч. по пакету 30720 из 30720 и за отдельную плату 6869
к оплате: 1006.20 руб. по цене 150.00 руб. за 1 Гб
абонентская плата: 1950.00 руб.
за дополнительные услуги: 1006.20 руб.
ИТОГО К ОПЛАТЕ: 2956.20 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.08.2018 ПО 12.09.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 35
минут разговора: 326, в т.ч. по пакету 326 из 3000
сообщений: 117, в т.ч. по пакету 117 из 1000
интернет-трафик (Мб): 35724, в т.ч. по пакету 30720 из 30720 и за отдельную плату 5004
к оплате: 733.01 руб. по цене 150.00 руб. за 1 Гб
абонентская плата: 1950.00 руб.
за дополнительные услуги: 733.01 руб.
ИТОГО К ОПЛАТЕ: 2683.01 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.09.2018 ПО 12.10.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 47
минут разговора: 374, в т.ч. по пакету 374 из 3000
сообщений: 113, в т.ч. по пакету 113 из 1000
интернет-трафик (Мб): 34253, в т.ч. по пакету 30720 из 30720 и за отдельную плату 3533
к оплате: 517.53 руб. по цене 150.00 руб. за 1 Гб
абонентская плата: 1950.00 руб.
за дополнительные услуги: 517.53 руб.
ИТОГО К ОПЛАТЕ: 2467.53 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.10.2018 ПО 12.11.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 38
минут разговора: 368, в т.ч. по пакету 368 из 3000
сообщений: 127, в т.ч. по пакету 127 из 1000
интернет-трафик (Мб): 27298, в т.ч. по пакету 27298 из 30720
абонентская плата: 1950.00 руб.
ИТОГО К ОПЛАТЕ: 1950.00 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.11.2018 ПО 12.12.2018
абонент: Дубинина Алиса (Москва)
тариф: Ультра, договор заключен 13.01.2018
звонков: 31
минут разговора: 273, в т.ч. по пакету 273 из 3000
сообщений: 126, в т.ч. по пакету 126 из 1000
интернет-трафик (Мб): 34360, в т.ч. по пакету 30720 из 30720 и за отдельную плату 3640
к оплате: 533.20 руб. по цене 150.00 руб. за 1 Гб
абонентская плата: 1950.00 руб.
за дополнительные услуги: 533.20 руб.

ИТОГО К ОПЛАТЕ: 2483.20 руб.

СЧЕТ ЗА УСЛУГИ, ОКАЗАННЫЕ С 13.12.2018 ПО 12.01.2019

абонент: Дубинина Алиса (Москва)

тариф: Ультра, договор заключен 13.01.2018

звонков: 38

минут разговора: 333, в т.ч. по пакету 333 из 3000

сообщений: 144, в т.ч. по пакету 144 из 1000

интернет-трафик (Мб): 38347, в т.ч. по пакету 30720 из 30720 и за отдельную плату 7627

к оплате: 1117.24 руб. по цене 150.00 руб. за 1 Гб

абонентская плата: 1950.00 руб.

за дополнительные услуги: 1117.24 руб.

ИТОГО К ОПЛАТЕ: 3067.24 руб.

[К оглавлению](#)

Спасибо за проверку!