Анализ тарифов на небольшой выборке клиентов
из генеральной совокупности.

В 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

«Мегалайн» всегда округляет вверх значения минут и мегабайтов.

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

План

1. Получение файлов с данными и изучение общей информации

- Описание данных
- Изучение общей информации о предоставляемых данных

2. Подготовка данных

- Приведите данные к нужным типам
- Исправление ошибок в данных
- Добавление информации

3. Анализ данных

- Описание поведений клиентов оператора, исходя из выборки.

4. Проверка гипотез

- Средняя выручка пользователей тарифов «Ультра» и «Смарт» различается
- Средняя выручка пользователей из Москвы отличается от выручки пользователей из других регионов

5. Общий вывод

---

1. Получение файлов с данными и изучение общей информации

Описание данных

Таблица **`users`** (информация о пользователях):

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

Таблица **`calls`** (информация о звонках):

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

Таблица **`messages`** (информация о сообщениях):

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

Таблица **`internet`** (информация об интернет-сессиях):

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

Таблица **tariffs** (информация о тарифах):

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

Изучение общей информации о предоставляемых данных и приведение типов данных

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import scipy.stats as st
import numpy as np
import os
from pathlib import Path
import urllib
import plotly.express as px

pd.options.display.float_format = '{:.2f}'.format
```

In [2]:
```
def get_file(name, url):
    if not os.path.exists(name):
        print(name, 'не найден. Будет загружен из сети')
        _ = urllib.request.urlretrieve(url, name)

files = {
    'calls': ('datasets/calls.csv', 'https://code.s3.yandex.net/datasets/calls.csv'),
    'internet': ('datasets/internet.csv', 'https://code.s3.yandex.net/datasets/internet
    'messages': ('datasets/messages.csv', 'https://code.s3.yandex.net/datasets/messages
    'tariffs': ('datasets/tariffs.csv', 'https://code.s3.yandex.net/datasets/tariffs.cs
    'users': ('datasets/users.csv', 'https://code.s3.yandex.net/datasets/users.csv')
}


Path('datasets').mkdir(parents=True, exist_ok=True)
```

```
for key in files:
    get_file(*files[key])
```

In [3]:
```
df_calls, df_internet, df_messages, df_tariffs, df_users = [pd.read_csv(files[key][0])
base_data = df_calls, df_internet, df_messages, df_tariffs, df_users
```

In [4]:
```
print('#' * 100)
for df, name in zip(base_data, files):
    print('{: >20}'.format(name.upper()))
    df.info()
    print('#' * 100)
```

```
####################################################################################################
#############
               CALLS
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202607 entries, 0 to 202606
Data columns (total 4 columns):
id           202607 non-null object
call_date    202607 non-null object
duration     202607 non-null float64
user_id      202607 non-null int64
dtypes: float64(1), int64(1), object(2)
memory usage: 6.2+ MB
####################################################################################################
#############
            INTERNET
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149396 entries, 0 to 149395
Data columns (total 5 columns):
Unnamed: 0      149396 non-null int64
id              149396 non-null object
mb_used         149396 non-null float64
session_date    149396 non-null object
user_id         149396 non-null int64
dtypes: float64(1), int64(2), object(2)
memory usage: 5.7+ MB
####################################################################################################
#############
            MESSAGES
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123036 entries, 0 to 123035
Data columns (total 3 columns):
id              123036 non-null object
message_date    123036 non-null object
user_id         123036 non-null int64
dtypes: int64(1), object(2)
memory usage: 2.8+ MB
####################################################################################################
#############
             TARIFFS
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 8 columns):
messages_included        2 non-null int64
mb_per_month_included    2 non-null int64
minutes_included         2 non-null int64
rub_monthly_fee          2 non-null int64
rub_per_gb               2 non-null int64
rub_per_message          2 non-null int64
rub_per_minute           2 non-null int64
tariff_name              2 non-null object
dtypes: int64(7), object(1)
memory usage: 256.0+ bytes
```

```
################################################################################################
############
              USERS
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
user_id       500 non-null int64
age           500 non-null int64
churn_date    38 non-null object
city          500 non-null object
first_name    500 non-null object
last_name     500 non-null object
reg_date      500 non-null object
tariff        500 non-null object
dtypes: int64(2), object(6)
memory usage: 31.4+ KB
################################################################################################
############
```

Вывод

Данные, полученные от компании, не содержат пустых значений, за исключением столбца `churn_data` таблицы `users`
о чём было упомянуто в первом пункте (если значение пропущено, то тариф ещё действовал на момент выгрузки данных)

2. Подготовка данных

Приведите данные к нужным типам

In [5]:
```
def describe_updown(data, list_cols=False):
    '''Поиск значений low_iqr и up_iqr для солбцов в list_cols DataFrame data
    и занесений значений в DataFrame метода .describe()'''
    def get_lowest_uppest(col):
        '''Получение нижнего и верхнего "усов" данных - то,
        что будет добавлено в DataFrame метода .describe()'''
        col_info = dict(col.describe())
        if col_info.get('75%', None) is None:
            return None
        iqr = col_info['75%'] - col_info['25%']
        lowest = col_info['25%'] - 1.5 * iqr
        lowest = lowest if lowest >= 0 else 0
        uppest = col_info['75%'] + 1.5 * iqr
        return lowest, uppest

    if list_cols is False:
        list_cols = data.columns
    cols_to_add = []
    descr = pd.DataFrame(data[list_cols].describe())
    lowers, uppers = [], []
    for item in list_cols:
        temp = get_lowest_uppest(data[item])
        if temp:
            cols_to_add.append(item)
            lowers.append(temp[0])
            uppers.append(temp[1])
    to_add = pd.DataFrame([lowers, uppers], index=(('low_iqr', 'up_iqr')))
    to_add.columns = cols_to_add
    descr = descr.append(to_add)
    return descr
```

```
def del_anomal_values(data, info_descr, list_cols):
    for col in list_cols:
        low, up = info_descr[col]['low_iqr'], info_descr[col]['up_iqr']
        data = data[(low < data[col]) & (data[col] < up)]
    return data
```

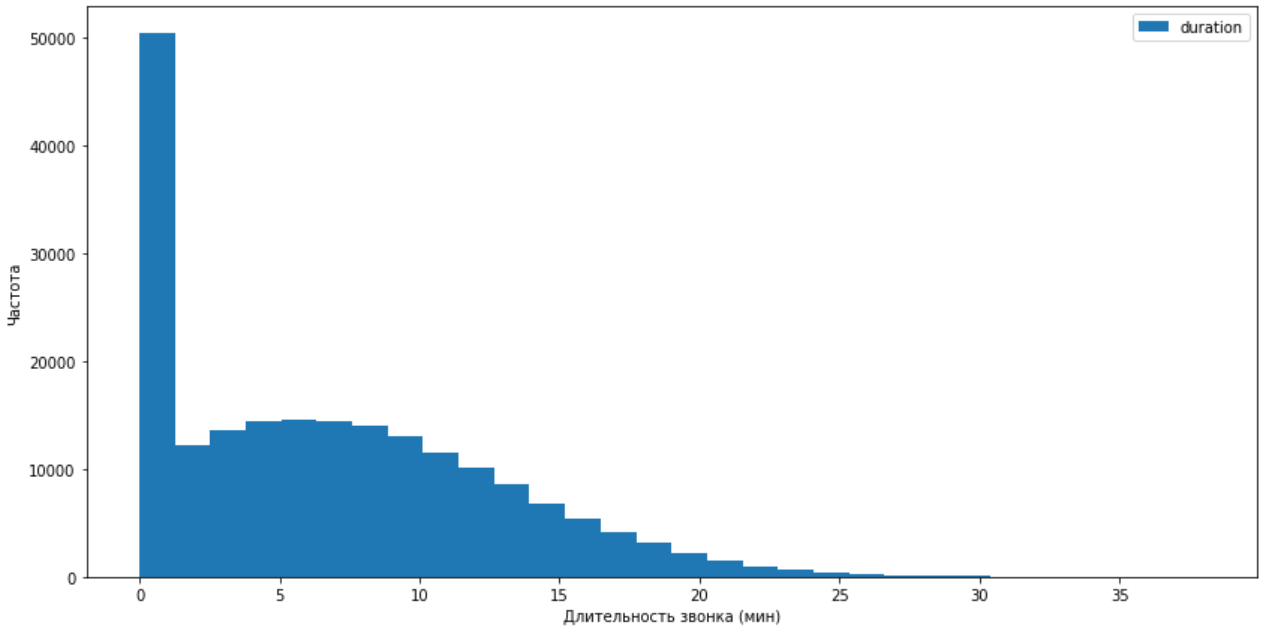
Таблица calls

In [6]:
```
df_calls_info = describe_updown(df_calls)
display(df_calls_info.T)
display(df_calls.sample())
```

	count	mean	std	min	25%	50%	75%	max	low_iqr	up_iqr
duration	202607.00	6.76	5.84	0.00	1.30	6.00	10.70	38.00	0.00	24.80
user_id	202607.00	1253.94	144.72	1000.00	1126.00	1260.00	1379.00	1499.00	746.50	1758.50

	id	call_date	duration	user_id
161662	1396_363	2018-09-23	1.72	1396

In [7]:
```
df_calls.plot(kind='hist', y='duration', bins=30, figsize=(14, 7))
plt.xlabel('Длительность звонка (мин)')
plt.ylabel('Частота')
plt.show()
```



In [8]:
```
print('Длительность в 0 мин составляет {: .0%}'.format((df_calls['duration'] == 0).sum(
```

```
Длительность в 0 мин составляет  20%
```

20% много для удаления из датасета, предполагаем, что пользователь такого рода звонки будут рассмотрены компанией

как состоявшиеся (с целью получения максимальной прибыли) Следовательно, заменим на 1

In [9]:
```
df_calls.loc[df_calls['duration'] == 0, 'duration'] = 1
df_calls.loc[:, 'duration'] = np.ceil(df_calls['duration']).astype(np.uint8)
```

```
df_calls.loc[:, 'user_id'] = df_calls['user_id'].astype(np.uint16)
```

In [10]:
```
df_calls.loc[:, 'call_date'] = pd.to_datetime(df_calls['call_date'])
```

In [11]:
```
df_calls.duplicated().sum()
```

Out[11]:
```
0
```

Таблица **internet**

In [12]:
```
df_internet_info = describe_updown(df_internet)
display(df_internet_info.T)
display(df_internet.sample(3))
```

	count	mean	std	min	25%	50%	75%	max	low_iqr	
Unnamed: 0	149396.00	74697.50	43127.05	0.00	37348.75	74697.50	112046.25	149395.00	0.00	22
mb_used	149396.00	370.19	278.30	0.00	138.19	348.01	559.55	1724.83	0.00	
user_id	149396.00	1252.10	144.05	1000.00	1130.00	1251.00	1380.00	1499.00	755.00	

	Unnamed: 0	id	mb_used	session_date	user_id
137923	137923	1464_147	655.99	2018-03-20	1464
16420	16420	1056_224	226.41	2018-07-18	1056
134822	134822	1453_43	571.90	2018-07-31	1453

In [13]:
```
df_internet.loc[:, 'session_date'] = pd.to_datetime(df_internet['session_date'])
```

In [14]:
```
df_internet.loc[:, 'gb_used'] = np.ceil(df_internet['mb_used'] / 1024).astype(np.uint8)
del df_internet['mb_used']
```

In [15]:
```
df_internet.loc[:, 'user_id'] = df_internet['user_id'].astype(np.uint16)
```

In [16]:
```
df_internet.duplicated().sum()
```

Out[16]:
```
0
```

Таблица **messages**

In [17]:
```
df_meassages_info = describe_updown(df_messages)
display(df_meassages_info.T)
display(df_messages.sample())
```

	count	mean	std	min	25%	50%	75%	max	low_iqr	up_iqr
user_id	123036.00	1256.99	143.52	1000.00	1134.00	1271.00	1381.00	1499.00	763.50	1751.50

	id	message_date	user_id
25587	1104_779	2018-02-23	1104

In [18]:
```
df_messages.loc[:, 'message_date'] = pd.to_datetime(df_messages['message_date'])
```

In [19]:
```
df_messages.loc[:, 'user_id'] = df_messages['user_id'].astype(np.uint16)
```

In [20]:
```
df_messages.duplicated().sum()
```

Out[20]:
```
0
```

Таблица **tariffs**

In [21]:
```
df_tariffs.columns = 'messages_included', 'mb_per_month_included', 'minutes_included',\
       'rub_monthly_fee', 'rub_per_gb', 'rub_per_message', 'rub_per_minute', 'tariff'
display(df_tariffs)
```

	messages_included	mb_per_month_included	minutes_included	rub_monthly_fee	rub_per_gb	rub_per
0	50	15360	500	550	200	
1	1000	30720	3000	1950	150	

In [22]:
```
for col in set(df_tariffs.columns) - {'tariff'}:
    df_tariffs.loc[:, col] = df_tariffs[col].astype(np.uint16)
```

Таблица **users**

In [23]:
```
df_users_info = describe_updown(df_users)
display(df_users_info.T)
display(df_users.sample())
```

	count	mean	std	min	25%	50%	75%	max	low_iqr	up_iqr
user_id	500.00	1249.50	144.48	1000.00	1124.75	1249.50	1374.25	1499.00	750.50	1748.50
age	500.00	46.59	16.67	18.00	32.00	46.00	62.00	75.00	0.00	107.00

	user_id	age	churn_date	city	first_name	last_name	reg_date	tariff
25	1025	56	NaN	Уфа	Матвей	Акинин	2018-03-15	smart

In [24]:
```
df_users.loc[:, 'reg_date'] = pd.to_datetime(df_users['reg_date'])
df_users.loc[:, 'churn_date'] = pd.to_datetime(df_users['churn_date'])
```

In [25]:
```
df_users.loc[:, 'user_id'] = df_users['user_id'].astype(np.uint16)
df_users.loc[:, 'age'] = df_users['age'].astype(np.uint8)
```

In [26]:
```
df_users.duplicated().sum()
```

Out[26]:
```
0
```

Вывод

Предобработали данные, заменили типы хранимых таблицами данных для оптимизации хранения и использования.

---

1) df_calls.loc[:, 'duration'] = np.ceil(df_calls['duration']).astype(np.uint8)
2) df_internet.loc[:, 'gb_used'] = np.ceil(df_internet['mb_used'] / 1024).astype(np.uint8)

In [27]:
```
print('#' * 100)
for df, name in zip(base_data, files):
    print('{: >20}'.format(name.upper()))
    df.info()
    print('#' * 100)
```

```
####################################################################################################
#############
               CALLS
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 202607 entries, 0 to 202606
Data columns (total 4 columns):
id           202607 non-null object
call_date    202607 non-null datetime64[ns]
duration     202607 non-null uint8
user_id      202607 non-null uint16
dtypes: datetime64[ns](1), object(1), uint16(1), uint8(1)
memory usage: 3.7+ MB
####################################################################################################
#############
            INTERNET
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149396 entries, 0 to 149395
Data columns (total 5 columns):
Unnamed: 0      149396 non-null int64
id              149396 non-null object
session_date    149396 non-null datetime64[ns]
user_id         149396 non-null uint16
gb_used         149396 non-null uint8
dtypes: datetime64[ns](1), int64(1), object(1), uint16(1), uint8(1)
memory usage: 3.8+ MB
####################################################################################################
#############
            MESSAGES
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 123036 entries, 0 to 123035
Data columns (total 3 columns):
id              123036 non-null object
message_date    123036 non-null datetime64[ns]
user_id         123036 non-null uint16
dtypes: datetime64[ns](1), object(1), uint16(1)
memory usage: 2.1+ MB
####################################################################################################
#############
             TARIFFS
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 8 columns):
messages_included        2 non-null uint16
mb_per_month_included    2 non-null uint16
minutes_included         2 non-null uint16
rub_monthly_fee          2 non-null uint16
rub_per_gb               2 non-null uint16
rub_per_message          2 non-null uint16
rub_per_minute           2 non-null uint16
```

```
tariff                   2 non-null object
dtypes: object(1), uint16(7)
memory usage: 172.0+ bytes
################################################################################################
#############
               USERS
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 8 columns):
user_id       500 non-null uint16
age           500 non-null uint8
churn_date    38 non-null datetime64[ns]
city          500 non-null object
first_name    500 non-null object
last_name     500 non-null object
reg_date      500 non-null datetime64[ns]
tariff        500 non-null object
dtypes: datetime64[ns](2), object(4), uint16(1), uint8(1)
memory usage: 25.0+ KB
################################################################################################
#############
```

Исправление ошибок в данных

In [28]:
```
if len(df_internet[df_internet['Unnamed: 0'] == df_internet.index]) == df_internet.shap
    print('Столбец \'Unnamed: 0\' является копией индексов в этой таблице, удалим его'
    del df_internet['Unnamed: 0']
```

```
Столбец 'Unnamed: 0' является копией индексов в этой таблице, удалим его
```

Добавление информации

In [29]:
```
df_calls.loc[:, 'month'] = df_calls['call_date'].dt.month.astype(np.uint8)
df_calls_month = df_calls.groupby(['user_id', 'month']).agg({'duration': 'sum', 'call_d
df_calls_month.columns = ['user_id', 'month', 'calls_sum', 'calls_cnt']
display(df_calls_month.head(3))
# display(len(df_calls_month))
```

	user_id	month	calls_sum	calls_cnt
0	1000	5	164.00	22
1	1000	6	187.00	43
2	1000	7	346.00	47

In [30]:
```
df_messages.loc[:, 'month'] = df_messages['message_date'].dt.month.astype(np.uint8)
df_messages_month = df_messages.groupby(['user_id', 'month']).agg({'message_date': 'cou
df_messages_month.columns = ['user_id', 'month', 'msg_cnt']
display(df_messages_month.head(3))
# display(len(df_messages_month))
```

	user_id	month	msg_cnt
0	1000	5	22
1	1000	6	60
2	1000	7	75

In [31]:
```
df_general = df_calls_month.merge(df_messages_month, on=['user_id', 'month'], how='oute
# df_general.shape[0]
```

In [32]:
```
df_internet.loc[:, 'month'] = df_internet['session_date'].dt.month.astype(np.uint8)
df_internet_month = df_internet.groupby(['user_id', 'month']).agg({'gb_used': 'sum'}).r
df_internet_month.columns = ['user_id', 'month', 'gb_used_sum']
display(df_internet_month.head())
```

	user_id	month	gb_used_sum
0	1000	5	4
1	1000	6	49
2	1000	7	28
3	1000	8	27
4	1000	9	26

In [33]:
```
df_general = df_general.merge(df_internet_month, on=['user_id', 'month'], how='outer')
```

In [34]:
```
df_general = df_general.merge(df_users[['user_id', 'tariff', 'city']], on='user_id')
```

In [35]:
```
df_general = df_general.merge(df_tariffs, on='tariff', how='outer')
```

In [36]:
```
def get_price_month(row):
    price = row['rub_monthly_fee']
    over_cll_m = row['calls_sum'] - row['minutes_included']
    over_msg_m = row['msg_cnt'] - row['messages_included']
    over_int_gb = row['gb_used_sum'] - row['mb_per_month_included']
    if over_cll_m > 0:
        price += over_cll_m * row['rub_per_minute']
    if over_msg_m > 0:
        price += over_msg_m * row['rub_per_message']
    if over_int_gb > 0:
        price += over_int_gb * row['rub_per_gb']
    return price


df_general.loc[:, 'price'] = df_general.apply(get_price_month, axis=1)
```

In [37]:
```
df_price = df_general[['user_id', 'month', 'calls_sum', 'calls_cnt', 'msg_cnt', 'gb_use
                                    'tariff', 'price', 'city']]
```

Вывод

Для получения таблцы `df_price` в которой содержатся данные о звонках, сообщениях и потречнном объёме ГБ интернета,
а также название тарифа, цене, потраченной пользователем тарифа за определённый месяц пришлось сгруппировать данные
из входных таблиц по месяцам и использвоать метод `merge` с именованным параметров `how` аргумент которму задали как `out`
с целью максимального охвата входных данных для последующего анализа.

---

3. Анализ данных

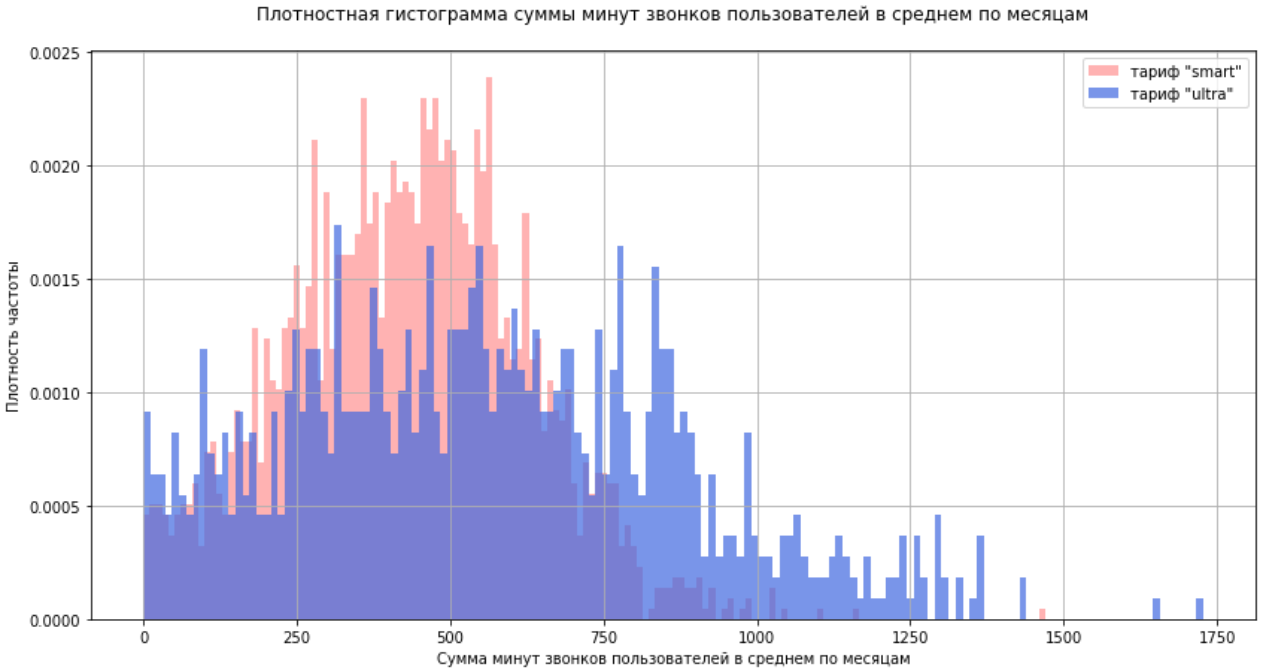
Описание поведений клиентов оператора, исходя из выборки

In [38]:
```
calls_info = df_price.pivot_table(index='tariff', values='calls_sum', aggfunc=['mean',
calls_info.columns = 'тариф', 'срднее', 'станд. откл.', 'дисперсия'
print('\nСуммарное кол-во звонков по месяцам')
display(calls_info)
print()

# df_price.groupby('tariff')['calls_sum'].plot(kind='hist', bins=150,  alpha=0.7, grid=
ax = df_price.query('tariff == "smart"')['calls_sum'].plot(kind='hist', bins=150,\
                                    alpha=0.3, grid=True, figsize=(14, 7), color='r', d
df_price.query('tariff == "ultra"')['calls_sum'].plot(kind='hist', bins=150,\
                                    alpha=0.7, grid=True, figsize=(14, 7), color='royal
plt.legend(['тариф "smart"', 'тариф "ultra"'])
plt.title('Плотностная гистограмма суммы минут звонков пользователей в среднем по месяц
plt.xlabel('Сумма минут звонков пользователей в среднем по месяцам')
plt.ylabel('Плотность частоты')
plt.show()
```

Суммарное кол-во звонков по месяцам

	тариф	срднее	станд. откл.	дисперсия
0	smart	430.61	193.85	37577.97
1	ultra	560.11	314.74	99062.63



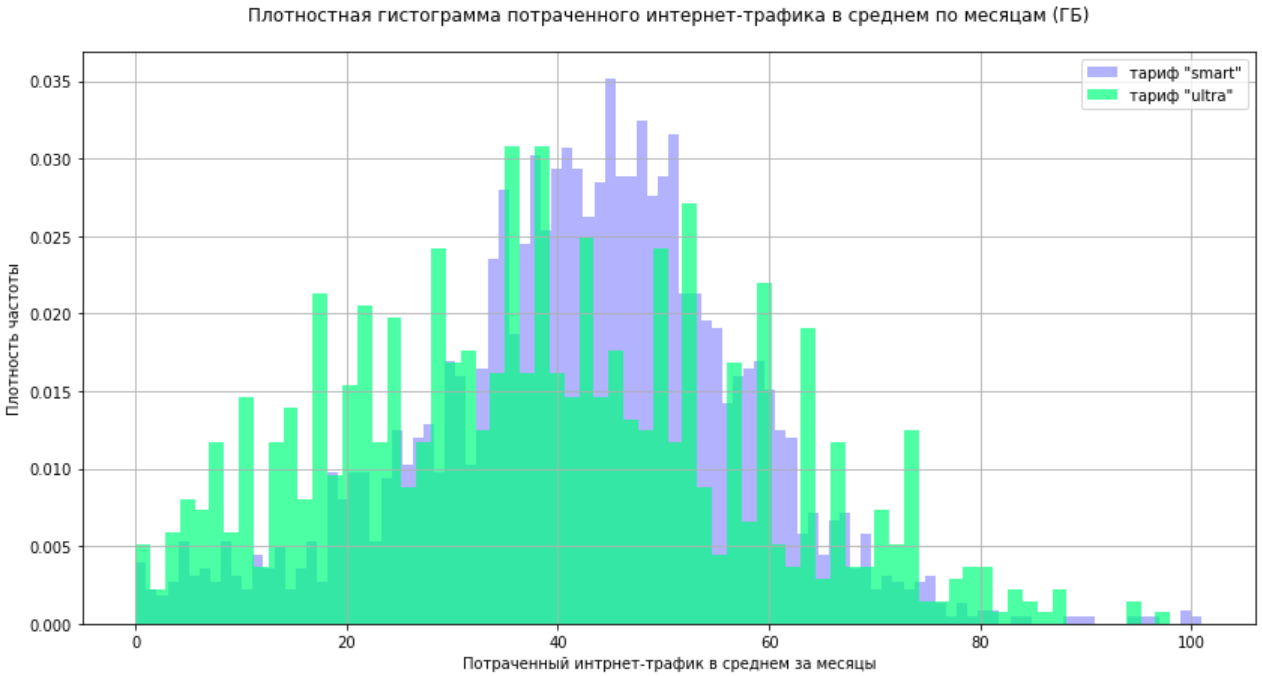
In [39]:
```
calls_info = df_price.pivot_table(index='tariff', values='gb_used_sum', aggfunc=['mean'
calls_info.columns = 'тариф', 'срднее', 'станд. откл.', 'дисперсия'
print('\nИнтернет-трафик в среднем по месяцам (ГБ)')
display(calls_info)
print()

# df_price.groupby('tariff')['calls_sum'].plot(kind='hist', bins=150,  alpha=0.7, grid=
ax = df_price.query('tariff == "smart"')['gb_used_sum'].plot(kind='hist', bins=100,\
                                    alpha=0.3, grid=True, figsize=(14, 7), color='b', d
df_price.query('tariff == "ultra"')['gb_used_sum'].plot(kind='hist', bins=70,\
                                    alpha=0.7, grid=True, figsize=(14, 7), color='sprin
```

```
plt.legend(['тариф "smart"', 'тариф "ultra"'])
plt.title('Плотностная гистограмма потраченного интернет-трафика в среднем по месяцам (
plt.xlabel('Потраченный интрнет-трафик в среднем за месяцы')
plt.ylabel('Плотность частоты')
plt.show()
```

Интернет-трафик в среднем по месяцам (ГБ)

	тариф	срднее	станд. откл.	дисперсия
0	smart	42.44	14.92	222.59
1	ultra	38.56	19.28	371.60



Вывод

По значениями дисперсии и стандартного отклонения суммарного количества минут и потраченного интернет-трафика обоих тарифов,
можно утверждать, что в пользовании тарифа ultra имеет место гораздо больший разброс значений от среднего.
Пользователям тарифа smart количества ГБ чаще не хватает, чем ползьователям тарифа ultra.
В случае же мобильной связи ситуация иная: в среднем количества минут хватает пользователям обоих тарифов.
Гистограммы тарифа "smart" более похоже на нормальное распределение, чем тарифа ultra

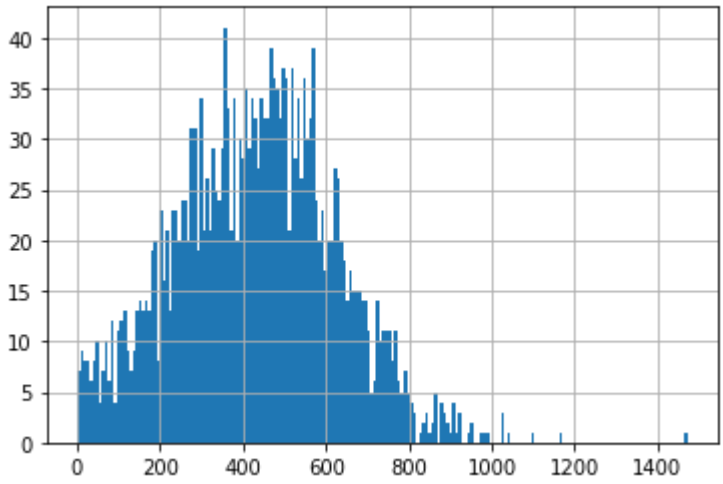
После анализа среднего, дисперсии и стандартного отклонения параметров двух тарифов smart и ultra.
Можно сделать вывод, что пользователи тарифа smart переплачивают больше, хотя и отклонения от заданных
оператором ограничений носят не такой большой характер (то есть пользователи тарифа ultra гораздо более
вариативны в использовании трафика). Так как переплата за доп трафик у тарифа "smart"

выше чем у "ultra".

У пользователей первого тарифа выходит гораздо больше трат на "связь".

In [40]:
```
df_price.query('tariff=="smart"')['calls_sum'].hist(bins = 200)
print('Среднее')
display(df_price.query('tariff=="smart"')['calls_sum'].mean())
print('Стандартное отклонение')
display(df_price.query('tariff=="smart"')['calls_sum'].std())
print('Дисперсия')
display(df_price.query('tariff=="smart"')['calls_sum'].var())
```

```
Среднее
430.60998650472334
Стандартное отклонение
193.85037292443653
Дисперсия
37577.96708294311
```



4. Проверка гипотез

Средняя выручка пользователей тарифов «Ультра» и «Смарт» различается

За нулевую и альтернативную гипотезы выберем и обозначим:

- H0: Средняя выручка пользователей тарифов «Ультра» и «Смарт» одинаковая
- H1: Средняя выручка пользователей тарифов «Ультра» и «Смарт» различается.

Используя выборки из Генеральной Совокупности (далее ГС) найдём вероятность того, что средние двух выборок одинаковы
с помощью теста Стьюдента. (Критический уровень статистической значимости примем 0.05)

In [41]:
```
alpha = .05
first_one = df_price.query('tariff == "smart"')['price']
second_one = df_price.query('tariff == "ultra"')['price']

f_var = first_one.describe()['std']
s_var = second_one.describe()['std']

is_true = np.abs(f_var - s_var) < 0.00001

distr = st.ttest_ind(first_one, second_one, equal_var=is_true)
if distr.pvalue < alpha:
```

```
    print('Отбрасываем нулевую гипотезу, принимаем альтернативную')
else:
    print('Принимаем нулевую гипотезу')
```

```
Отбрасываем нулевую гипотезу, принимаем альтернативную
```

Средняя выручка пользователей из Москвы отличается от выручки пользователей из других регионов

За нулевую и альтернативную гипотезы выберем и обозначим:

- H0: Средняя выручка оператора от пользователей из Москвы равна средней выручке пользователей из регионов.
- H1: Средняя выручка оператора от пользователей из Москвы отличается от выручки пользователей из регионов.

Используя выборки из Генеральной Совокупности (далее ГС) найдём вероятность того, что средние двух выборок одинаковы
с помощью теста Стьюдента. (Критический уровень статистической значимости примем 0.05)

In [42]:
```
alpha = .05
first_one = df_price.query('tariff == "smart" and city == "Москва"')['price']
second_one = df_price.query('tariff == "ultra" and city != "Москва"')['price']

f_var = first_one.describe()['std']
s_var = second_one.describe()['std']

is_true = np.abs(f_var - s_var) < 0.00001

distr = st.ttest_ind(first_one, second_one, equal_var=is_true)
if distr.pvalue < alpha:
    print('Отбрасываем нулевую гипотезу, принимаем альтернативную')
else:
    print('Принимаем нулевую гипотезу')
```

```
Отбрасываем нулевую гипотезу, принимаем альтернативную
```

Вывод

Таким образом, полученный тест Стьюдента позволил убедиться в том, что данные не позволяют утверждать
состоятельность нулевых гипотез.

---

5. Общий вывод

Вывод

После анализа среднего, дисперсии и стандартного отклонения параметров двух тарифов smart и ultra. Можно сделать вывод, что пользователи тарифа smart переплачивают больше, хотя и отклонения от заданных
оператором ограничений носят не такой большой характер (то есть пользователи тарифа ultra гораздо более
вариативны в использовании трафика). Так как переплата за доп трафик у тарифа "smart" выше чем у

"ultra".

У пользователей первого тарифа выходит гораздо больше трат на "связь".

Проведённый тест Стьюдента позволил утверждать то, что выборки из генеральной совокупности не равны.

Средние занчения из двух выборок отличаются на значительную величину.
