

Описание проекта

Клиентам предлагают два тарифных плана: «Смарт» и «Ультра». Чтобы скорректировать рекламный бюджет, коммерческий департамент хочет понять, какой тариф приносит больше денег. Нам предстоит сделать предварительный анализ тарифов на небольшой выборке клиентов. В нашем распоряжении данные 500 пользователей: кто они, откуда, каким тарифом пользуются, сколько звонков и сообщений каждый отправил за 2018-й год. Нужно проанализировать поведение клиентов и сделать вывод — какой тариф лучше. Описание тарифов

Тариф «Смарт»

- 1. Ежемесячная плата: 550 рублей
- 2. Включено 500 минут разговора, 50 сообщений и 15 Гб интернет-трафика
- 3. Стоимость услуг сверх тарифного пакета:
  - Минута разговора — 3 рубля. Количество использованных минут и мегабайтов «Мегалайн» всегда округляет вверх. Если пользователь проговорил всего 1 секунду, в тарифе засчитывается целая минута.
  - Сообщение — 3 рубля.
  - 1 Гб интернет-трафика — 200 рублей.

Тариф «Ультра»

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

In [1]: `import pandas as pd`

Откроем файл с данными и изучим общую информацию

Откроем файл `/datasets/calls.csv` , сохраним датафрейм в переменную `calls` .

In [2]: `calls = pd.read_csv('datasets/calls.csv')`

Выведем первые 5 строк датафрейма `calls` .

In [3]: `calls.head(5)`

Out[3]:

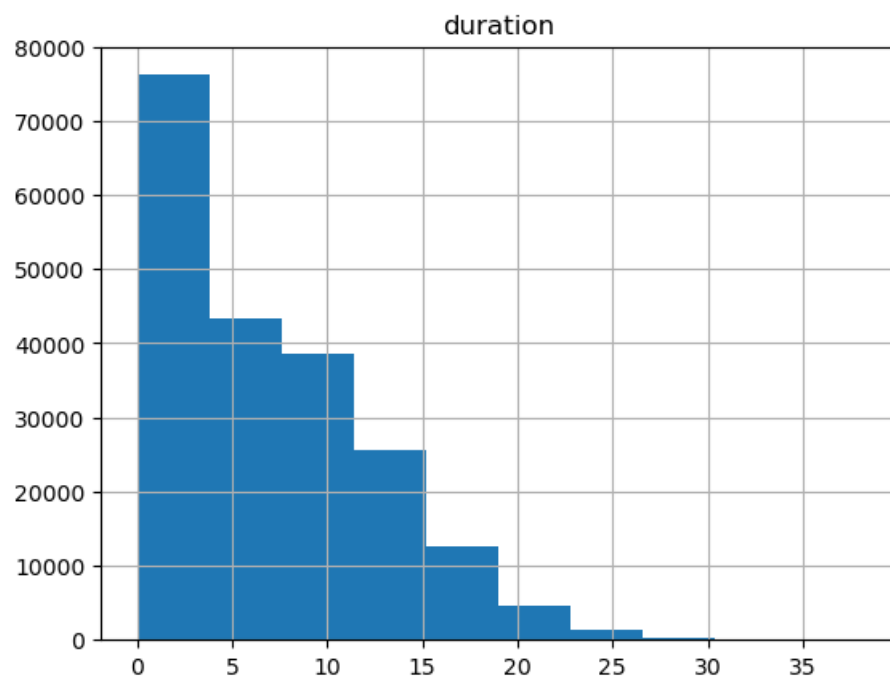
	id	call_date	duration	user_id
0	1000_0	2018-07-25	0.00	1000
1	1000_1	2018-08-17	0.00	1000
2	1000_2	2018-06-11	2.85	1000
3	1000_3	2018-09-21	13.80	1000
4	1000_4	2018-12-15	5.18	1000

Посмотрим основную информацию для датафрейма `calls` с помощью метода `info()` .

In [4]: `calls.info()`

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

In [5]: `calls.hist('duration');`



Откроем файл /datasets/internet.csv , сохраним датафрейм в переменную sessions .

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

Выведем первые 5 строк датафрейма sessions .

```
In [7]: sessions.head(5)
```

Out[7]:

	Unnamed: 0	id	mb_used	session_date	user_id
0	0	1000_0	112.95	2018-11-25	1000
1	1	1000_1	1052.81	2018-09-07	1000
2	2	1000_2	1197.26	2018-06-25	1000
3	3	1000_3	550.27	2018-08-22	1000
4	4	1000_4	302.56	2018-09-24	1000

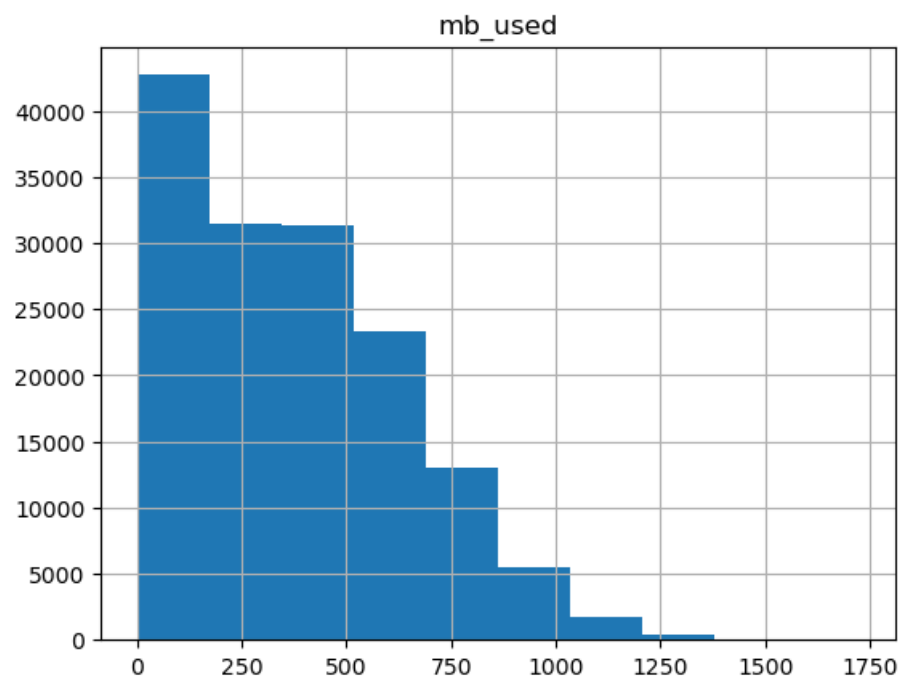
Посмотрим основную информацию для датафрейма sessions с помощью метода info() .

```
In [8]: sessions.info()
```

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

С помощью метода hist() выведем гистограмму для столбца с количеством потраченных мегабайт.

```
In [9]: sessions.hist('mb_used');
```



Откроем файл /datasets/messages.csv , сохраним датафрейм в переменную messages .

```
In [10]: messages = pd.read_csv('datasets/messages.csv')
```

Выведем первые 5 строк датафрейма messages .

```
In [11]: messages.head(5)
```

Out[11]:

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

Посмотрим основную информацию для датафрейма messages с помощью метода info() .

```
In [12]: messages.info()
```

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

Откроем файл /datasets/tariffs.csv , сохраним датафрейм в переменную tariffs .

```
In [13]: tariffs = pd.read_csv('datasets/tariffs.csv')
```

Выведем весь датафрейм tariffs .

```
In [14]: tariffs
```

Out[14]:

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

Посмотрим основную информацию для датафрейма tariffs с помощью метода info() .

```
In [15]: tariffs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 2 entries, 0 to 1
```

```
Data columns (total 8 columns):
```

```
#   Column      Non-Null Count  Dtype
```

```
---  ---
0   messages_included    2 non-null    int64
1   mb_per_month_included 2 non-null    int64
2   minutes_included     2 non-null    int64
3   rub_monthly_fee      2 non-null    int64
4   rub_per_gb           2 non-null    int64
5   rub_per_message      2 non-null    int64
6   rub_per_minute       2 non-null    int64
7   tariff_name          2 non-null    object
```

```
dtypes: int64(7), object(1)
```

```
memory usage: 256.0+ bytes
```

Откроем файл /datasets/users.csv , сохраним датафрейм в переменную users .

```
In [16]: users = pd.read_csv('datasets/users.csv')
```

Выведем первые 5 строк датафрейма users .

```
In [17]: users.head(5)
```

```
Out[17]:
```

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

Посмотрим основную информацию для датафрейма users с помощью метода info() .

```
In [18]: users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 500 entries, 0 to 499
```

```
Data columns (total 8 columns):
```

```
#   Column      Non-Null Count  Dtype
```

```
---  ---
0   user_id    500 non-null    int64
1   age        500 non-null    int64
2   churn_date 38 non-null    object
3   city        500 non-null    object
4   first_name  500 non-null    object
5   last_name   500 non-null    object
6   reg_date    500 non-null    object
7   tariff      500 non-null    object
```

```
dtypes: int64(2), object(6)
```

```
memory usage: 31.4+ KB
```

## Подготовим данные

Приведем столбцы

- reg\_date из таблицы users
- churn\_date из таблицы users
- call\_date из таблицы calls
- message\_date из таблицы messages
- session\_date из таблицы sessions

к новому типу с помощью метода to\_datetime() .

```
In [19]: users['reg_date'] = pd.to_datetime(users['reg_date'], format='%Y-%m-%d') # обработка столбца reg_date
users['churn_date'] = pd.to_datetime(users['churn_date'], format='%Y-%m-%d') # обработка столбца churn_date
```

```
calls['call_date'] = pd.to_datetime(calls['call_date'], format='%Y-%m-%d') # обработка столбца call_date
```

```
messages['message_date'] = pd.to_datetime(messages['message_date'], format='%Y-%m-%d') # обработка столбца message_date
```

```
sessions['session_date'] = pd.to_datetime(sessions['session_date'], format='%Y-%m-%d') # обработка столбца session_date
```

В данных мы найдём звонки с нулевой продолжительностью. Это не ошибка: нулями обозначены пропущенные звонки, поэтому их не нужно удалять.

Однако в столбце duration датафрейма calls значения дробные. Округлим значения столбца duration вверх с помощью метода numpy.ceil() и приведем столбец duration к типу int .

```
In [20]: import numpy as np
```

```
calls['duration'] = np.ceil(calls['duration']).astype('int')
```

```
# округление значений столбца duration с помощью np.ceil() и приведение типа к int
```

Удалим столбец Unnamed: 0 из датафрейма sessions . Столбец с таким названием возникает, когда данные сохраняют с указанием индекса ( df.to\_csv(..., index=column) ). Он сейчас не понадобится.

```
In [21]: sessions = sessions.drop(columns='Unnamed: 0')
```

Создадим столбец `month` в датафрейме `calls` с номером месяца из столбца `call_date` .

```
In [22]: calls['month'] = pd.DatetimeIndex(calls['call_date']).month
```

Создадим столбец `month` в датафрейме `messages` с номером месяца из столбца `message_date` .

```
In [23]: messages['month'] = pd.DatetimeIndex(messages['message_date']).month
```

Создадим столбец `month` в датафрейме `sessions` с номером месяца из столбца `session_date` .

```
In [24]: sessions['month'] = pd.DatetimeIndex(sessions['session_date']).month
```

Посчитаем количество сделанных звонков разговора для каждого пользователя по месяцам.

```
In [25]: calls_per_month = calls.groupby(['user_id', 'month']).agg(calls=('duration', 'count'))  
# подсчёт количества звонков для каждого пользователя по месяцам
```

```
In [26]: calls_per_month.head(30) # вывод 30 первых строк на экран
```

```
Out[26]:
```

calls		
user_id	month	
1000	5	22
	6	43
	7	47
	8	52
	9	58
	10	57
	11	43
	12	46
1001	11	59
	12	63
1002	6	15
	7	26
	8	42
	9	36
	10	33
	11	32
	12	33
1003	8	55
	9	134
	10	108
	11	115
	12	108
1004	5	9
	6	31
	7	22
	8	19
	9	26
	10	29
	11	19
	12	21

Посчитаем количество израсходованных минут разговора для каждого пользователя по месяцам и сохраним в переменную `minutes_per_month` . Нам понадобится

- сгруппировать датафрейм с информацией о звонках по двум столбцам — с идентификаторами пользователей и номерами месяцев;
- после группировки выбрать столбец `duration`
- затем применить метод для подсчёта суммы.

Выведем первые 30 строчек `minutes_per_month` .

```
In [27]: minutes_per_month = calls.groupby(['user_id', 'month']).agg(minutes=('duration', 'sum'))  
# подсчёт израсходованных минут для каждого пользователя по месяцам
```

```
In [28]: minutes_per_month.head(30) # вывод первых 30 строк на экран
```

Out[28]:

		minutes
user_id	month	
1000	5	159
	6	172
	7	340
	8	408
	9	466
	10	350
	11	338
	12	333
1001	11	430
	12	414
1002	6	117
	7	214
	8	289
	9	206
	10	212
	11	243
	12	236
1003	8	380
	9	961
	10	855
	11	824
	12	802
1004	5	35
	6	171
	7	135
	8	137
	9	117
	10	145
	11	117
	12	130

Посчитаем количество отправленных сообщений по месяцам для каждого пользователя и сохраним в переменную `messages_per_month` . Нам понадобится

- сгруппировать датафрейм с информацией о сообщениях по двум столбцам — с идентификаторами пользователей и номерами месяцев;
- после группировки выбрать столбец `message_date` ;
- затем применить метод для подсчёта количества.

Выведем первые 30 строчек `messages_per_month` .

```
In [29]: messages_per_month = messages.groupby(['user_id', 'month']).agg(messages=('message_date', 'count'))
# подсчёт количества отправленных сообщений для каждого пользователя по месяцам
```

```
In [30]: messages_per_month.head(30) # вывод первых 30 строк на экран
```

Out[30]:

		messages
user_id	month	
1000	5	22
	6	60
	7	75
	8	81
	9	57
	10	73
	11	58
	12	70
1002	6	4
	7	11
	8	13
	9	4
	10	10
	11	16
	12	12
1003	8	37
	9	91
	10	83
	11	94
	12	75
1004	5	95
	6	134
	7	181
	8	151
	9	146
	10	165
	11	158
	12	162
1005	1	7
	2	38

Посчитаем количество потраченных мегабайт по месяцам для каждого пользователя и сохраним в переменную `sessions_per_month` . Нам понадобится

- сгруппировать датафрейм с информацией о сообщениях по двум столбцам — с идентификаторами пользователей и номерами месяцев;
- затем применить метод для подсчёта суммы: `.agg({'mb_used': 'sum'})`

```
In [31]: sessions_per_month = sessions.groupby(['user_id', 'month']).agg({'mb_used': 'sum'})
# подсчёт потраченных мегабайт для каждого пользователя по месяцам
```

```
In [32]: sessions_per_month.head(30) # вывод первых 30 строк на экран
```

Out[32]:

			mb_used
	user_id	month	
	1000	5	2253.49
		6	23233.77
		7	14003.64
		8	14055.93
		9	14568.91
		10	14702.49
		11	14756.47
		12	9817.61
	1001	11	18429.34
		12	14036.66
	1002	6	10856.82
		7	17580.10
		8	20319.26
		9	16691.08
		10	13888.25
		11	18587.28
	1003	12	18113.73
		8	8565.21
		9	12468.87
		10	14768.14
		11	11356.89
	1004	12	10121.53
		5	13403.98
		6	17600.02
		7	22229.58
		8	28584.37
		9	15109.03
		10	18475.44
		11	15616.02
		12	18021.04

Анализ данных и подсчёт выручки

Объединяем все посчитанные выше значения в один датафрейм user\_behavior . Для каждой пары "пользователь - месяц" будут доступны информация о тарифе, количестве звонков, сообщений и потраченных мегабайтах.

In [33]: users['churn\_date'].count() / users['churn\_date'].shape[0] \* 100

Out[33]:7.6  
Расторгли договор 7.6% клиентов из датасета

In [34]: user\_behavior = calls\_per\_month\  
 .merge(messages\_per\_month, left\_index=True, right\_index=True, how='outer')\  
 .merge(sessions\_per\_month, left\_index=True, right\_index=True, how='outer')\  
 .merge(minutes\_per\_month, left\_index=True, right\_index=True, how='outer')\  
 .reset\_index()\  
 .merge(users, how='left', left\_on='user\_id', right\_on='user\_id')\  
  
user\_behavior.head()

Out[34]:

	user_id	month	calls	messages	mb_used	minutes	age	churn_date	city	first_name	last_name	reg_date	tariff
0	1000	5	22.0	22.0	2253.49	159.0	52	NaT	Краснодар	Рафаил	Верещагин	2018-05-25	ultra
1	1000	6	43.0	60.0	23233.77	172.0	52	NaT	Краснодар	Рафаил	Верещагин	2018-05-25	ultra
2	1000	7	47.0	75.0	14003.64	340.0	52	NaT	Краснодар	Рафаил	Верещагин	2018-05-25	ultra
3	1000	8	52.0	81.0	14055.93	408.0	52	NaT	Краснодар	Рафаил	Верещагин	2018-05-25	ultra
4	1000	9	58.0	57.0	14568.91	466.0	52	NaT	Краснодар	Рафаил	Верещагин	2018-05-25	ultra

Проверим пропуски в таблице user\_behavior после объединения:

In [35]: user\_behavior.isna().sum()



```
Out[35]: user_id      0
         month        0
         calls       40
         messages    497
         mb_used      11
         minutes     40
         age          0
         churn_date  3027
         city         0
         first_name   0
         last_name    0
         reg_date     0
         tariff       0
         dtype: int64
```

Заполним образовавшиеся пропуски в данных:

```
In [36]: user_behavior['calls'] = user_behavior['calls'].fillna(0)
         user_behavior['minutes'] = user_behavior['minutes'].fillna(0)
         user_behavior['messages'] = user_behavior['messages'].fillna(0)
         user_behavior['mb_used'] = user_behavior['mb_used'].fillna(0)
```

Присоединяем информацию о тарифах

```
In [37]: # переименование столбца tariff_name на более простое tariff
```

```
tariffs = tariffs.rename(
    columns={
        'tariff_name': 'tariff'
    }
)
```

```
In [38]: user_behavior = user_behavior.merge(tariffs, on='tariff')
```

Считаем количество минут разговора, сообщений и мегабайт, превышающих включенные в тариф

```
In [39]: user_behavior['paid_minutes'] = user_behavior['minutes'] - user_behavior['minutes_included']
         user_behavior['paid_messages'] = user_behavior['messages'] - user_behavior['messages_included']
         user_behavior['paid_mb'] = user_behavior['mb_used'] - user_behavior['mb_per_month_included']
```

```
for col in ['paid_messages', 'paid_minutes', 'paid_mb']:
    user_behavior.loc[user_behavior[col] < 0, col] = 0
```

Переводим превышающие тариф мегабайты в гигабайты и сохраняем в столбец paid\_gb

```
In [40]: user_behavior['paid_gb'] = np.ceil(user_behavior['paid_mb'] / 1024).astype(int)
```

Считаем выручку за минуты разговора, сообщения и интернет

```
In [41]: user_behavior['cost_minutes'] = user_behavior['paid_minutes'] * user_behavior['rub_per_minute']
         user_behavior['cost_messages'] = user_behavior['paid_messages'] * user_behavior['rub_per_message']
         user_behavior['cost_gb'] = user_behavior['paid_gb'] * user_behavior['rub_per_gb']
```

Считаем месячную выручку с каждого пользователя, она будет храниться в столбец total\_cost

```
In [42]: user_behavior['total_cost'] = \
         user_behavior['rub_monthly_fee'] \
         + user_behavior['cost_minutes'] \
         + user_behavior['cost_messages'] \
         + user_behavior['cost_gb']
```

Датафрейм stats\_df для каждой пары "месяц-тариф" будет хранить основные характеристики

```
In [43]: # сохранение статистических метрик для каждой пары месяц-тариф
         # в одной таблице stats_df (среднее значение, стандартное отклонение, медиана)
```

```
stats_df = user_behavior.pivot_table(
    index=['month', 'tariff'], \
    values=['calls', 'minutes', 'messages', 'mb_used'], \
    aggfunc=['mean', 'std', 'median'] \
).round(2).reset_index()
```

```
stats_df.columns=['month', 'tariff', 'calls_mean', 'sessions_mean', 'messages_mean', 'minutes_mean',
                  'calls_std', 'sessions_std', 'messages_std', 'minutes_std',
                  'calls_median', 'sessions_median', 'messages_median', 'minutes_median']
```

```
stats_df.head(10)
```

Out[43]:	month	tariff	calls_mean	sessions_mean	messages_mean	minutes_mean	calls_std	sessions_std	messages_std	minutes_std	calls_median	se
0	1	smart	27.68	8513.72	18.24	203.85	20.81	6444.68	16.20	154.23	20.5	
1	1	ultra	59.44	13140.68	33.78	428.11	41.64	6865.35	30.67	269.76	51.0	
2	2	smart	40.19	11597.05	24.09	298.69	25.39	6247.35	21.75	190.82	38.5	
3	2	ultra	41.54	11775.94	21.96	297.12	40.97	10644.64	26.77	296.51	25.0	
4	3	smart	54.32	15104.16	31.86	390.05	25.54	5828.24	26.80	191.89	59.0	
5	3	ultra	67.68	17535.55	32.30	489.65	44.84	10951.79	41.62	333.74	57.0	
6	4	smart	51.31	13462.18	30.74	367.13	25.70	5698.25	24.54	186.49	52.0	
7	4	ultra	64.09	16828.13	31.56	458.02	36.27	9718.65	37.51	267.68	61.0	
8	5	smart	55.24	15805.18	33.77	387.36	25.38	5978.23	27.04	186.60	59.0	
9	5	ultra	72.51	19363.15	37.85	510.33	41.08	10046.11	40.31	289.60	75.0	

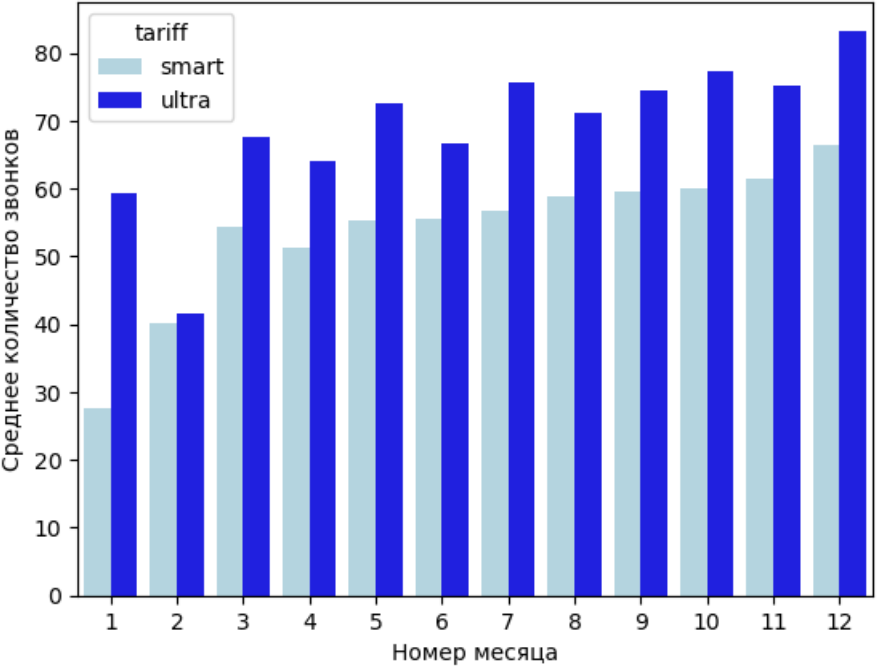
Распределение среднего количества звонков по видам тарифов и месяцам

In [44]: **import** seaborn **as** sns

```
ax = sns.barplot(x='month',
                 y='calls_mean',
                 hue="tariff",
                 data=stats_df,
                 palette=['lightblue', 'blue'])

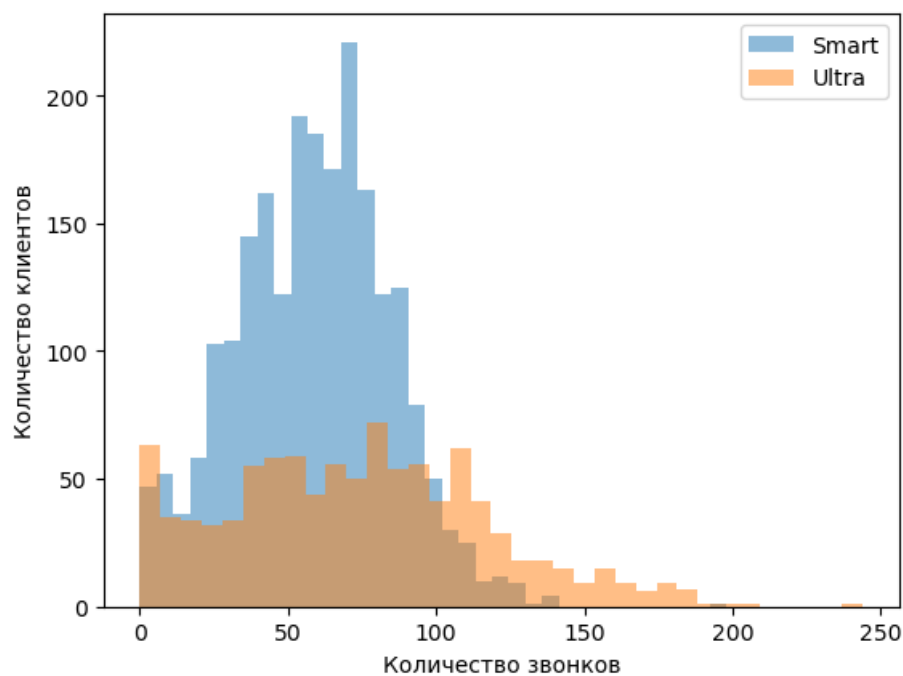
ax.set_title('Распределение количества звонков по видам тарифов и месяцам')
ax.set(xlabel='Номер месяца', ylabel='Среднее количество звонков');
```

Распределение количества звонков по видам тарифов и месяцам



In [45]: **import** matplotlib.pyplot **as** plt

```
user_behavior.groupby('tariff')['calls'].plot(kind='hist', bins=35, alpha=0.5)
plt.legend(['Smart', 'Ultra'])
plt.xlabel('Количество звонков')
plt.ylabel('Количество клиентов')
plt.show()
```

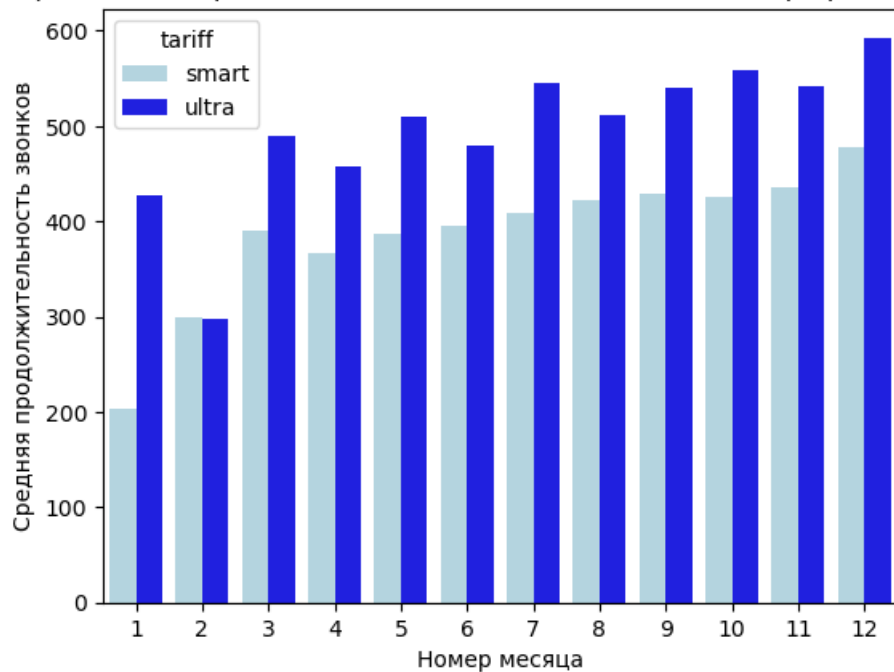


Распределение средней продолжительности звонков по видам тарифов и месяцам

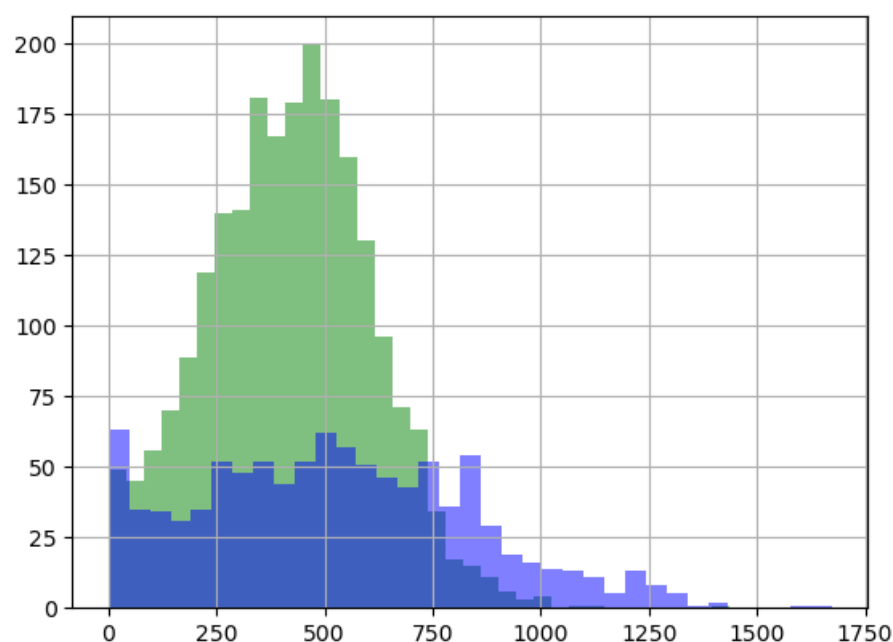
```
In [46]: ax = sns.barplot(x='month',
                        y='minutes_mean',
                        hue="tariff",
                        data=stats_df,
                        palette=['lightblue', 'blue'])

ax.set_title('Распределение продолжительности звонков по видам тарифов и месяцам')
ax.set(xlabel='Номер месяца', ylabel='Средняя продолжительность звонков');
```

Распределение продолжительности звонков по видам тарифов и месяцам



```
In [47]: user_behavior[user_behavior['tariff'] == 'smart']['minutes'].hist(bins=35, alpha=0.5, color='green')
user_behavior[user_behavior['tariff'] == 'ultra']['minutes'].hist(bins=35, alpha=0.5, color='blue');
```



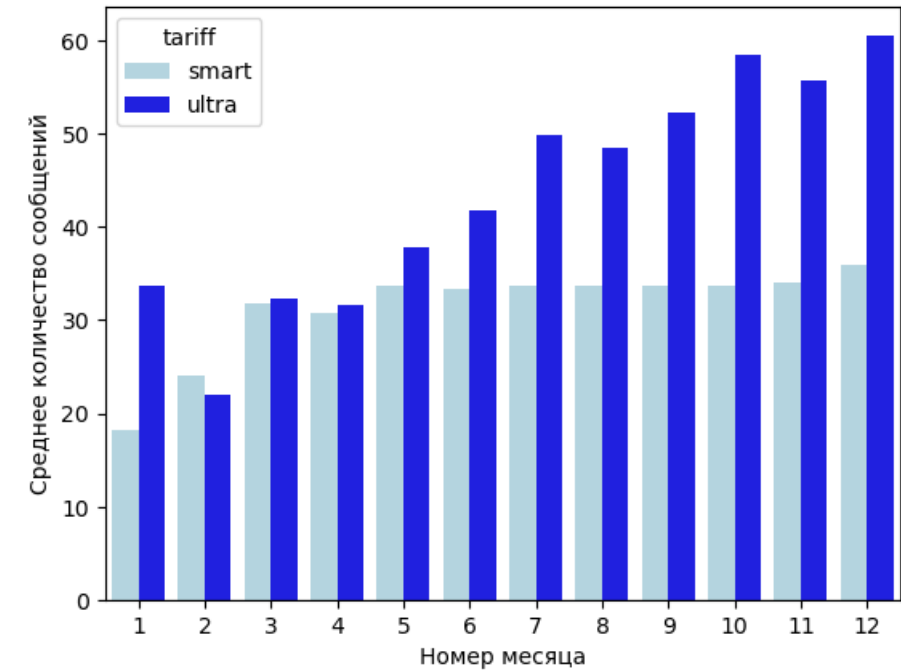
Средняя длительность разговоров у абонентов тарифа Ultra больше, чем у абонентов тарифа Smart. В течение года пользователи обоих тарифов увеличивают среднюю продолжительность своих разговоров. Рост средней длительности разговоров у абонентов тарифа Smart равномерный в течение года. Пользователи тарифа Ultra не проявляют подобной линейной стабильности. Стоит отметить, что феврале у абонентов обоих тарифных планов наблюдались самые низкие показатели.

Распределение среднего количества сообщений по видам тарифов и месяцам

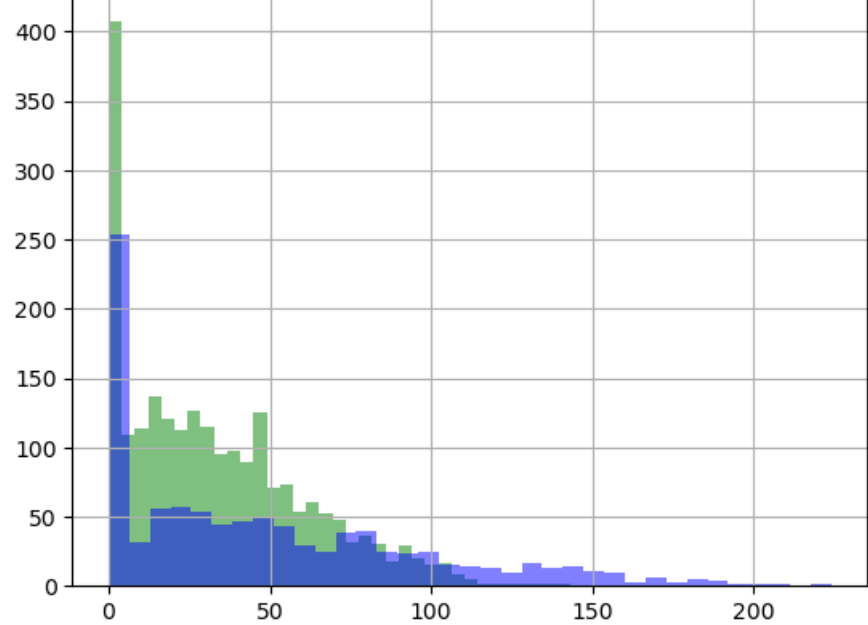
```
In [48]: ax = sns.barplot(x='month',
                        y='messages_mean',
                        hue="tariff",
                        data=stats_df,
                        palette=['lightblue', 'blue'])

ax.set_title('Распределение количества сообщений по видам тарифов и месяцам')
ax.set(xlabel='Номер месяца', ylabel='Среднее количество сообщений');
```

Распределение количества сообщений по видам тарифов и месяцам



```
In [49]: user_behavior[user_behavior['tariff'] == 'smart']['messages'].hist(bins=35, alpha=0.5, color='green')
user_behavior[user_behavior['tariff'] == 'ultra']['messages'].hist(bins=35, alpha=0.5, color='blue');
```

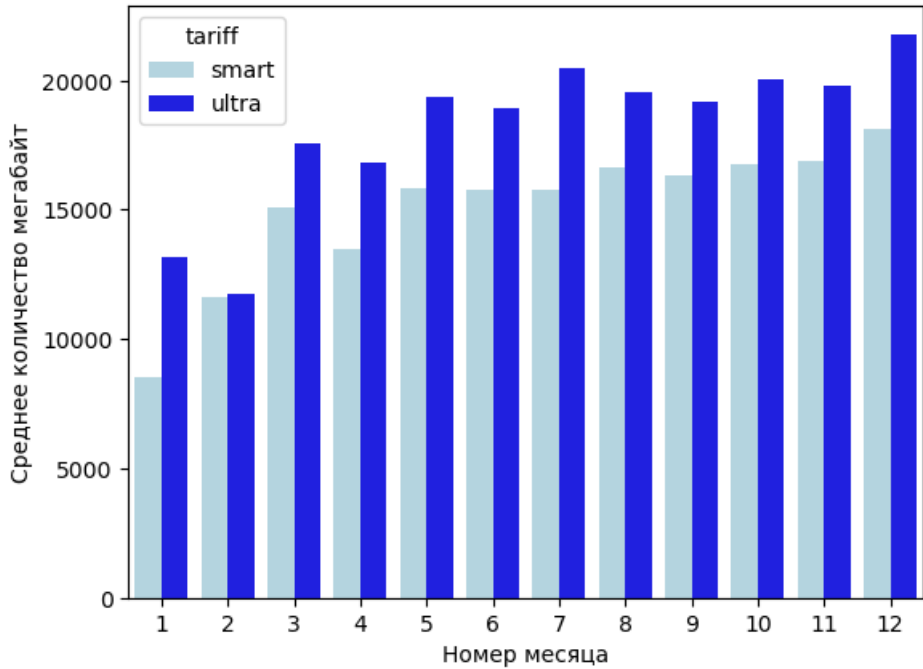


В среднем количество сообщений пользователи тарифа Ultra отправляют больше - почти на 20 сообщений больше, чем пользователи тарифа Smart. Количество сообщений в течение года на обоих тарифах растет. Динамика по отправке сообщений схожа с тенденциями по длительности разговоров: в феврале отмечено наименьшее количество сообщений за год и пользователи тарифа Ultra также проявляют нелинейную положительную динамику.

```
In [50]: ax = sns.barplot(x='month',
                        y='sessions_mean',
                        hue="tariff",
                        data=stats_df,
                        palette=['lightblue', 'blue'])

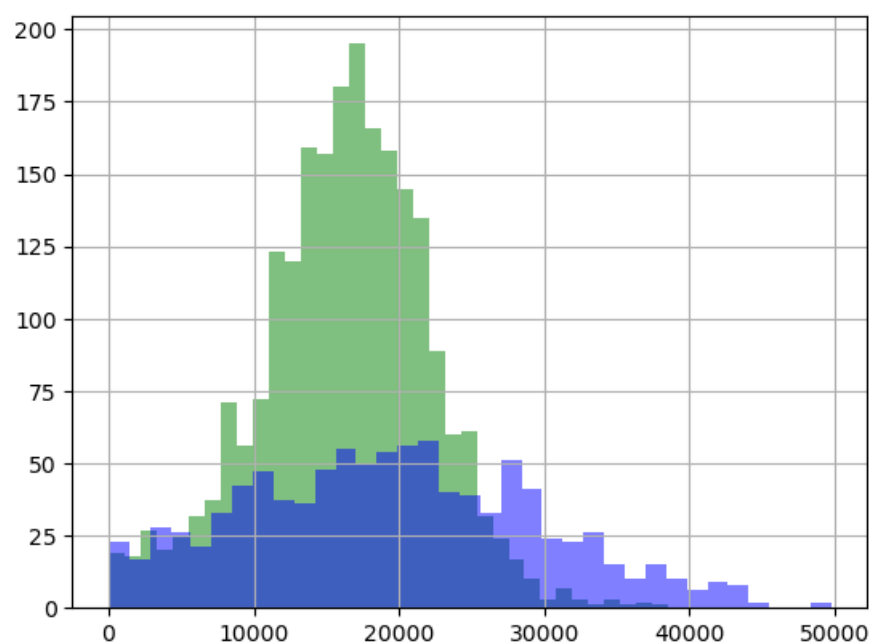
ax.set_title('Распределение количества потраченного трафика (Мб) по видам тарифов и месяцам')
ax.set(xlabel='Номер месяца', ylabel='Среднее количество мегабайт');
```

Распределение количества потраченного трафика (Мб) по видам тарифов и месяцам



Сравнение потраченных мегабайт среди пользователей тарифов Smart и Ultra

```
In [51]: user_behavior[user_behavior['tariff'] == 'smart']['mb_used'].hist(bins=35, alpha=0.5, color='green')
         user_behavior[user_behavior['tariff'] == 'ultra']['mb_used'].hist(bins=35, alpha=0.5, color='blue');
```



Меньше всего пользователи использовали интернет в январе, феврале и апреле. Чаще всего абоненты тарифа Smart тратят 15-17 Гб, а абоненты тарифного плана Ultra - 19-21 Гб.

### Проверка гипотез

Проверка гипотезы: средняя выручка пользователей тарифов «Ультра» и «Смарт» различаются;

H\_0: Выручка (total\_cost) пользователей "Ультра" = выручка (total\_cost) пользователей "Смарт"  
H\_1: Выручка (total\_cost) пользователей "Ультра" ≠ выручка (total\_cost) пользователей "Смарт"  
alpha = 0.05

```
In [52]: from scipy import stats as st

In [53]: results = st.ttest_ind(user_behavior[user_behavior['tariff'] == 'smart']['total_cost'],
                                user_behavior[user_behavior['tariff'] == 'ultra']['total_cost'],
                                equal_var=False)

#вызов метода для проверки гипотезы

alpha = 0.05
#зададим значение уровня значимости

print('p-значение:', results.pvalue)

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

p-значение: 4.2606313931076085e-250  
Отвергаем нулевую гипотезу  
Проверка гипотезы: пользователи из Москвы приносят больше выручки, чем пользователи из других городов;

H\_0: Выручка (total\_cost) пользователей из Москвы = выручка (total\_cost) пользователей не из Москвы`  
H\_1: Выручка (total\_cost) пользователей из Москвы ≠ выручка (total\_cost) пользователей не из Москвы`  
alpha = 0.05

```
In [54]: results = st.ttest_ind(user_behavior[user_behavior['city'] == 'Москва']['total_cost'],
                                user_behavior[user_behavior['city'] != 'Москва']['total_cost'],
                                equal_var=False)

alpha = 0.05
#зададим значение уровня значимости

print('p-значение:', results.pvalue)

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

p-значение: 0.5257376663729298  
Не получилось отвергнуть нулевую гипотезу