Проект Тарифы сотовой связи

### **Откройте файл с данными и изучите общую информацию**

**Задание 1.** Откройте файл /datasets/calls.csv, сохраните датафрейм в переменную calls.

Ввод [153]:

calls **=** pd.read\_csv('/datasets/calls.csv')

**Задание 2.** Выведите первые 5 строк датафрейма calls.

Ввод [154]:

calls.head()

Out[154]:

|  | **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 |

Ввод [ ]:

​

**Задание 3.** Выведите основную информацию для датафрейма calls с помощью метода info().

Ввод [155]:

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

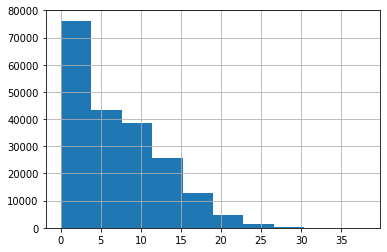
**Задание 4.** С помощью метода hist() выведите гистограмму для столбца с продолжительностью звонков. Подумайте о том, как распределены данные.

Ввод [156]:

calls['duration'].hist()

Out[156]:

<AxesSubplot:>



**Задание 5.** Откройте файл /datasets/internet.csv, сохраните датафрейм в переменную sessions.

Ввод [157]:

sessions **=** pd.read\_csv('/datasets/internet.csv')

**Задание 6.** Выведите первые 5 строк датафрейма sessions.

Ввод [158]:

sessions.head()

Out[158]:

|  | **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 |

**Задание 7.** Выведите основную информацию для датафрейма sessions с помощью метода info().

Ввод [159]:

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

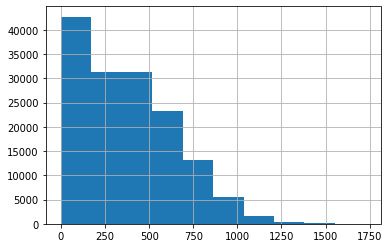
**Задание 8.** С помощью метода hist() выведите гистограмму для столбца с количеством потраченных мегабайт.

Ввод [160]:

sessions['mb\_used'].hist()

Out[160]:

<AxesSubplot:>



**Задание 9.** Откройте файл /datasets/messages.csv, сохраните датафрейм в переменную messages.

Ввод [161]:

messages **=** pd.read\_csv('/datasets/messages.csv')

**Задание 10.** Выведите первые 5 строк датафрейма messages.

Ввод [162]:

messages.head()

Out[162]:

|  | **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 |

**Задание 11.** Выведите основную информацию для датафрейма messages с помощью метода info().

Ввод [163]:

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

**Задание 12.** Откройте файл /datasets/tariffs.csv, сохраните датафрейм в переменную tariffs.

Ввод [164]:

tariffs **=** pd.read\_csv('/datasets/tariffs.csv')

**Задание 13.** Выведите весь датафрейм tariffs.

Ввод [165]:

tariffs

Out[165]:

|  | **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 |

**Задание 14.** Выведите основную информацию для датафрейма tariffs с помощью метода info().

Ввод [166]:

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

**Задание 15.** Откройте файл /datasets/users.csv, сохраните датафрейм в переменную users.

Ввод [167]:

users **=** pd.read\_csv('/datasets/users.csv')

**Задание 16.** Выведите первые 5 строк датафрейма users.

Ввод [168]:

users.head()

Out[168]:

|  | **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 |

**Задание 17.** Выведите основную информацию для датафрейма users с помощью метода info().

Ввод [169]:

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

### **Подготовьте данные**

**Задание 18.** Приведите 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().

Ввод [170]:

users['reg\_date'] **=** pd.to\_datetime(users['reg\_date'], format**=**'%Y-%m-%d %H:%M')*# обработка столбца reg\_date*

users['churn\_date'] **=** pd.to\_datetime(users['churn\_date'], format**=**'%Y-%m-%d %H:%M')*# обработка столбца churn\_date*

calls['call\_date'] **=** pd.to\_datetime(calls['call\_date'], format**=**'%Y-%m-%d %H:%M')*# обработка столбца call\_date*

messages['message\_date'] **=** pd.to\_datetime(messages['message\_date'], format**=**'%Y-%m-%d %H:%M')*# обработка столбца message\_date*

sessions['session\_date'] **=** pd.to\_datetime(sessions['session\_date'], format**=**'%Y-%m-%d %H:%M')*# обработка столбца session\_date*

**Задание 19.** В данных вы найдёте звонки с нулевой продолжительностью. Это не ошибка: нулями обозначены пропущенные звонки, поэтому их не нужно удалять.

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

Ввод [171]:

**import** numpy **as** np

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

calls['duration'] **=** np.ceil(calls['duration'])

calls['duration'] **=** calls['duration'].astype(int)

**Задание 20.** Удалите 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.

Ввод [172]:

sessions **=** sessions.drop(columns**=**['Unnamed: 0'], axis**=**1)

**Задание 21.** Создайте столбец month в датафрейме calls с номером месяца из столбца call\_date.

Ввод [176]:

calls['month'] **=** calls['call\_date'].dt.month

**Задание 22.** Создайте столбец month в датафрейме messages с номером месяца из столбца message\_date.

Ввод [180]:

messages['month'] **=** messages['message\_date'].dt.month

**Задание 23.** Создайте столбец month в датафрейме sessions с номером месяца из столбца session\_date.

Ввод [196]:

sessions['month'] **=** sessions['session\_date'].dt.month

**Задание 24.** Посчитайте количество сделанных звонков разговора для каждого пользователя по месяцам.

Ввод [201]:

calls\_per\_month **=** calls.groupby(['user\_id', 'month']).agg(calls**=**('duration', 'count')) *# подсчёт количества звонков для каждого пользователя по месяцам*

Ввод [202]:

calls\_per\_month.head(30) *# вывод 30 первых строк на экран*

Out[202]:

|  |  |  |
| --- | --- | --- |

**Задание 25.** Посчитайте количество израсходованных минут разговора для каждого пользователя по месяцам и сохраните в переменную minutes\_per\_month. Вам 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.

Выведите первые 30 строчек minutes\_per\_month.

Ввод [203]:

minutes\_per\_month **=** calls.groupby(['user\_id', 'month']).agg(minutes**=**('duration', 'sum'))*# подсчёт израсходованных минут для каждого пользователя по месяцам*

Ввод [205]:

minutes\_per\_month.head(30) *# вывод первых 30 строк на экран*

Out[205]:

**Задание 26.** Посчитайте количество отправленных сообщений по месяцам для каждого пользователя и сохраните в переменную messages\_per\_month. Вам 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.

Выведите первые 30 строчек messages\_per\_month.

Ввод [207]:

messages\_per\_month **=** messages.groupby(['user\_id', 'month']).agg(messages**=**('message\_date', 'count')) *# подсчёт количества отправленных сообщений для каждого пользователя по месяцам*

Ввод [208]:

messages\_per\_month.head(30)*# вывод первых 30 строк на экран*

Out[208]:

**Задание 27.** Посчитайте количество потраченных мегабайт по месяцам для каждого пользователя и сохраните в переменную sessions\_per\_month. Вам 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'})

Ввод [209]:

sessions\_per\_month **=** sessions.groupby(['user\_id', 'month']).agg({'mb\_used': 'sum'})*# подсчёт потраченных мегабайт для каждого пользователя по месяцам*

Ввод [210]:

sessions\_per\_month.head(30) *# вывод первых 30 строк на экран*

Out[210]:

### **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.

Ввод [211]:

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

Out[211]:

7.6

Расторгли договор 7.6% клиентов из датасета

Ввод [212]:

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[212]:

|  | **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:

Ввод [213]:

user\_behavior.isna().sum()

Out[213]:

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:

Ввод [214]:

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

Ввод [215]:

*# переименование столбца tariff\_name на более простое tariff*

​

tariffs **=** tariffs.rename(

columns**=**{

'tariff\_name': 'tariff'

}

)

Ввод [216]:

user\_behavior **=** user\_behavior.merge(tariffs, on**=**'tariff')

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

Ввод [217]:

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

Ввод [218]:

user\_behavior['paid\_gb'] **=** np.ceil(user\_behavior['paid\_mb'] **/** 1024).astype(int)

Считаем выручку за минуты разговора, сообщения и интернет

Ввод [219]:

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

Ввод [220]:

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

Ввод [221]:

*# 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[221]:

|  | **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** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 1 | smart | 27.68 | 8513.72 | 18.24 | 203.85 | 20.81 | 6444.68 | 16.20 | 154.23 | 20.5 | 7096.18 | 15.0 | 162.5 |

Распределение среднего количества звонков по видам тарифов и месяцам

Ввод [222]:

**import** seaborn **as** sns

​

ax **=** sns.barplot(x**=**'month',

y**=**'calls\_mean',

hue**=**"tariff",

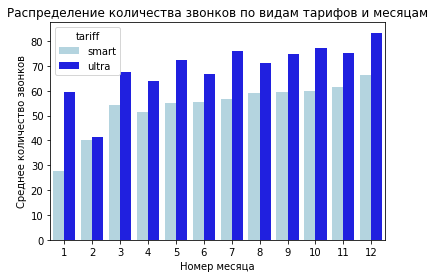
data**=**stats\_df,

palette**=**['lightblue', 'blue'])

​

ax.set\_title('Распределение количества звонков по видам тарифов и месяцам')

ax.set(xlabel**=**'Номер месяца', ylabel**=**'Среднее количество звонков');



Ввод [223]:

**import** matplotlib.pyplot **as** plt

​

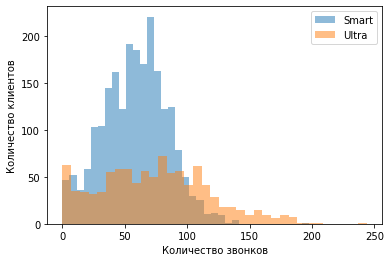
user\_behavior.groupby('tariff')['calls'].plot(kind**=**'hist', bins**=**35, alpha**=**0.5)

plt.legend(['Smart', 'Ultra'])

plt.xlabel('Количество звонков')

plt.ylabel('Количество клиентов')

plt.show()



Распределение средней продолжительности звонков по видам тарифов и месяцам

Ввод [224]:

ax **=** sns.barplot(x**=**'month',

y**=**'minutes\_mean',

hue**=**"tariff",

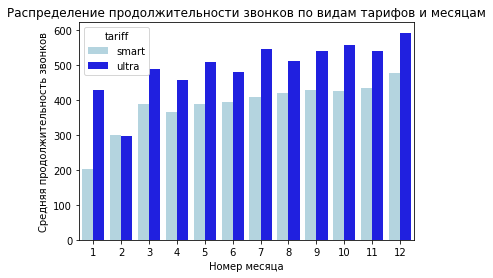
data**=**stats\_df,

palette**=**['lightblue', 'blue'])

​

ax.set\_title('Распределение продолжительности звонков по видам тарифов и месяцам')

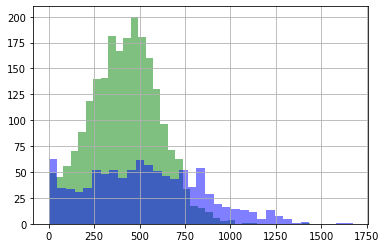
ax.set(xlabel**=**'Номер месяца', ylabel**=**'Средняя продолжительность звонков');



Ввод [225]:

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

Ввод [226]:

ax **=** sns.barplot(x**=**'month',

y**=**'messages\_mean',

hue**=**"tariff",

data**=**stats\_df,

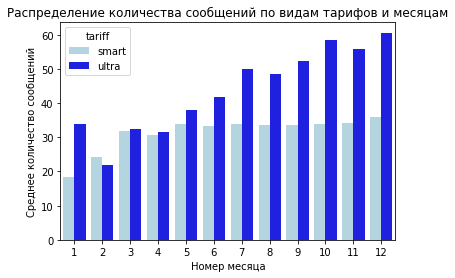
palette**=**['lightblue', 'blue']

)

​

ax.set\_title('Распределение количества сообщений по видам тарифов и месяцам')

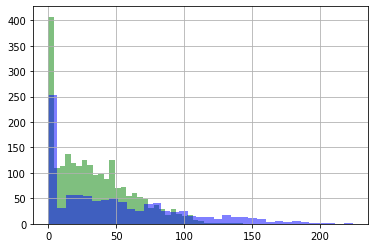
ax.set(xlabel**=**'Номер месяца', ylabel**=**'Среднее количество сообщений');



Ввод [227]:

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 тарифак 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.

Ввод [228]:

ax **=** sns.barplot(x**=**'month',

y**=**'sessions\_mean',

hue**=**"tariff",

data**=**stats\_df,

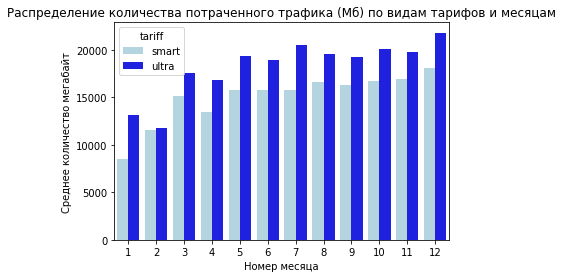
palette**=**['lightblue', 'blue']

)

​

ax.set\_title('Распределение количества потраченного трафика (Мб) по видам тарифов и месяцам')

ax.set(xlabel**=**'Номер месяца', ylabel**=**'Среднее количество мегабайт');

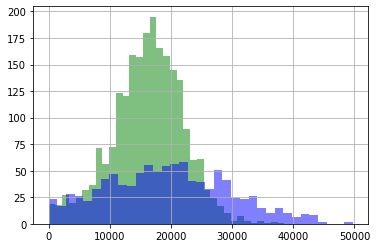


Сравнение потраченных мегабайт среди пользователей тарифов Smart и Ultra

Ввод [229]:

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**

**Задание 28.** Проверка гипотезы: средняя выручка пользователей тарифов «Ультра» и «Смарт» различаются;

H\_0: Выручка (total\_cost) пользователей "Ультра" = выручка (total\_cost) пользователей "Смарт"`

H\_a: Выручка (total\_cost) пользователей "Ультра" ≠ выручка (total\_cost) пользователей "Смарт"`

alpha = 0.05

Ввод [ ]:

**from** scipy **import** stats **as** st

Ввод [231]:

total\_cost\_smart **=** user\_behavior[user\_behavior['tariff'] **==**'smart']['total\_cost']

total\_cost\_ultra **=** user\_behavior[user\_behavior['tariff'] **==**'ultra']['total\_cost']

*# results = вызов метода для проверки гипотезы*

results **=** st.ttest\_ind(total\_cost\_smart, total\_cost\_ultra,equal\_var**=False**)

*# alpha = задайте значение уровня значимости*

alpha **=** .05

*# вывод значения p-value на экран*

*#print('p-значение:', results.pvalue)*

*# условный оператор с выводом строки с ответом*

**if** results.pvalue **<** alpha:

print(results.pvalue)

print("Отвергаем нулевую гипотезу")

**else**:

print(results.pvalue)

print("Не получилось отвергнуть нулевую гипотезу")

4.2606313931076085e-250

Отвергаем нулевую гипотезу

**Задание 29.** 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

Ввод [232]:

total\_cost\_mos **=** user\_behavior[user\_behavior['city'] **==**'Москва']['total\_cost']

total\_cost\_reg **=** user\_behavior[user\_behavior['city'] **!=**'Москва']['total\_cost']

*# results = вызов метода для проверки гипотезы*

results **=** st.ttest\_ind(total\_cost\_mos, total\_cost\_reg, equal\_var**=False**)

*# alpha = задайте значение уровня значимости*

alpha **=** .05

*# вывод значения p-value на экран*

print('p-значение:', results.pvalue)

*# условный оператор с выводом строки с ответом*

**if** results.pvalue **<** alpha:

print(results.pvalue)

print("Отвергаем нулевую гипотезу")

**else**:

print(results.pvalue)

print("Не получилось отвергнуть нулевую гипотезу")

p-значение: 0.5257376663729298

0.5257376663729298

Не получилось отвергнуть нулевую гипотезу