В [1]:
```
#загружаем необходимые библиотеки
 импортировать  pandas  как  pd
 импортировать  matplotlib.pyplot  как  plt
 из  plotly  импортировать  graph_objects  как  go
 импортировать  plotly.express  как  px
 импорт  plotly.io  как  pio
 pio.templates.default = "plotly_white"
 импортировать  numpy  как  np
 импортировать  seaborn  как  sns
 импортировать   itertools
 из  sklearn.metrics  импортировать  silhouette_score
 из  sklearn.кластерный  импорт  KMeans
 из  sklearn.импорт метрик   accuracy_score, precision_score, recall_score, f1_score,mean_absol
 из  sklearn.предварительная обработка  импортируйте  стандартный масштаб
 из  sklearn.model_selection  импортируйте  train_test_split
 из  sklearn.linear_model  импортируйте  Лассо,  гребень,LogisticRegression
 из  sklearn.tree  импорт  DecisionTreeRegressor
 из  sklearn.ensemble  импорт  RandomForestRegressor,  GradientBoostingRegressor, RandomForestC
 импорт  matplotlib.pyplot  как  plt
 из  scipy.cluster.hierarchy  импорт  дендрограммы,  привязки
 sns.set(style="whitegrid")
цвета  = ["#ef476f", "#ffd166", "#06d6a0","#118ab2","#073b4c"]
sns.set_palette(sns.color_palette(цвета))
импортировать  re
 из  scipy  импортировать  статистику  как  st
 импортировать  математику  как  mth
 импортировать  предупреждения
 warnings.filterwarnings('игнорировать')
```

В [2]:
```
#Загружаем данные:
df = pd.read_csv('/datasets/gym_churn.csv')
```

Проведем исследовательский анализ данных (EDA)

В [3]:
```
#Выведем 5 строк датафрейма:
df.head()
```

Выход [...

	пол	Near_Location рядом с местоположением	Партнер	Промо - Друзья	Телефон	Контрактный период	Групповые посещения	Возраст	Avg_additional_cl
0	1	1	1	1	0	6	1	29	
1	0	1	0	0	1	12	1	31	
2	0	1	1	0	1	1	0	28	
3	0	1	1	1	1	12	1	33	
4	1	1	1	1	1	1	0	26	

В [4]:
```
#Выведем размер
print(df.shape)
```

```
(4000, 14)
```

In [5]:
```
# Получаем информацию:
df.info()
```

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

```
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 14 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   gender                             4000 non-null   int64  
 1   Near_Location                      4000 non-null   int64  
 2   Partner                            4000 non-null   int64  
 3   Promo_friends                      4000 non-null   int64  
 4   Phone                              4000 non-null   int64  
 5   Contract_period                    4000 non-null   int64  
 6   Group_visits                       4000 non-null   int64  
 7   Age                                4000 non-null   int64  
 8   Avg_additional_charges_total       4000 non-null   float64
 9   Month_to_end_contract              4000 non-null   float64
 10  Lifetime                           4000 non-null   int64  
 11  Avg_class_frequency_total          4000 non-null   float64
 12  Avg_class_frequency_current_month  4000 non-null   float64
 13  Churn                              4000 non-null   int64  
dtypes: float64(4), int64(10)
memory usage: 437.6 KB
```

- Имеем столбцы:
  1 gender - пол
  2 Near_Location - проживание или работа в районе, где находится фитнес-центр
  3 Partner - информация о работодателе клиента
  4 Promo_friends - факт первоначальной записи в рамках акции «приведи друга»
  5 Phone - наличие контактного телефона
  6 Contract_period - длительность текущего действующего абонемента (месяц, 6 месяцев, год)
  7 Group_visits - факт посещения групповых занятий
  8 Age - возраст
  9 Avg_additional_charges_total - суммарная выручка от других услуг фитнес-центра: кафе, спорттовары, косметический и массажный салон
  10 Month_to_end_contract - срок до окончания текущего действующего абонемента (в месяцах)
  11 Lifetime - время с момента первого обращения в фитнес-центр (в месяцах)
  12 Avg_class_frequency_total - средняя частота посещений в неделю за все время с начала действия абонемента
  13 Avg_class_frequency_current_month - средняя частота посещений в неделю за предыдущий месяц
  14 Churn - факт оттока в текущем месяце.

Имеем датафрейм в 4000 строк, 14 столбцов. Тип данных - int64(10), float64(4)

Предобработка данных

In [6]:
```
# Названия столбцов:
df.columns
```

Out[6]:
```
Index(['gender', 'Near_Location', 'Partner', 'Promo_friends', 'Phone',
       'Contract_period', 'Group_visits', 'Age',
       'Avg_additional_charges_total', 'Month_to_end_contract', 'Lifetime',
       'Avg_class_frequency_total', 'Avg_class_frequency_current_month',
       'Churn'],
      dtype='object')
```

In [7]:
```
#переведем в нижний регистр
df.columns = df.columns.str.lower()
```

In [8]:
```
# Проверим:
df.columns
```

Out[8]:
```
Index(['gender', 'near_location', 'partner', 'promo_friends', 'phone',
       'contract_period', 'group_visits', 'age',
```

```
       'avg_additional_charges_total', 'month_to_end_contract', 'lifetime',
       'avg_class_frequency_total', 'avg_class_frequency_current_month',
       'churn'],
      dtype='object')
```

In [9]:
```
# проверим на дубликаты
df.duplicated(subset=['gender', 'near_location', 'partner', 'promo_friends', 'phone','contract_
       'avg_additional_charges_total', 'month_to_end_contract', 'lifetime',
       'avg_class_frequency_total', 'avg_class_frequency_current_month',
       'churn']).sum()
```

Out[9]:
```
0
```

Дубликатов не обнаружено.

In [10]:
```
# проверим на пропуски
df.isna().sum()
```

Out[10]:
```
gender                             0
near_location                      0
partner                            0
promo_friends                      0
phone                              0
contract_period                    0
group_visits                       0
age                                0
avg_additional_charges_total       0
month_to_end_contract              0
lifetime                           0
avg_class_frequency_total          0
avg_class_frequency_current_month  0
churn                              0
dtype: int64
```

Пропусков не обнаружено.

In [11]:
```
# Переименуем некоторые столбцы для удобства:
df = df.rename(columns={'avg_class_frequency_current_month':'a_month'})
df = df.rename(columns={'avg_class_frequency_total':'a_total'})
df = df.rename(columns={'avg_additional_charges_total':'a_charges'})
df = df.rename(columns={'month_to_end_contract':'month_end'})
df = df.rename(columns={'contract_period':'period'})
```

In [12]:
```
df.head()
```

Out[12]:

	gender	near_location	partner	promo_friends	phone	period	group_visits	age	a_charges	month_end	lif
0	1	1	1	1	0	6	1	29	14.227470	5.0	
1	0	1	0	0	1	12	1	31	113.202938	12.0	
2	0	1	1	0	1	1	0	28	129.448479	1.0	
3	0	1	1	1	1	12	1	33	62.669863	12.0	
4	1	1	1	1	1	1	0	26	198.362265	1.0	

Вывод.

- В процессе предобработки:
  - привели к нижнему регистру текстовые данные в столбцах
  - проверили на дубли и отсутствующие значения
  - переименовали некоторые столбцы.

- Тип данных в каждой колонке — числовые. Дубликатов нет. Пропущенных значений нет.
- Данные не имеют временную структуру.

EDA. Формулировка гипотез.

In [13]:
```
# Изучим средние значения и стандартные отклонения:
df.describe()
```

Out[13]:

	gender	near_location	partner	promo_friends	phone	period	group_visits	a
count	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.000000	4000.0000
mean	0.510250	0.845250	0.486750	0.308500	0.903500	4.681250	0.412250	29.1842
std	0.499957	0.361711	0.499887	0.461932	0.295313	4.549706	0.492301	3.2583
min	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000	18.0000
25%	0.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.000000	27.0000
50%	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	0.000000	29.0000
75%	1.000000	1.000000	1.000000	1.000000	1.000000	6.000000	1.000000	31.0000
max	1.000000	1.000000	1.000000	1.000000	1.000000	12.000000	1.000000	41.0000

-Вывод из использования функции describe:

- Мужчин и женщин в фитнес-клубе одинаковое количество
- Больше тех кто проживает вблизи фитнеса - 85%
- Почти половина клиентов сотрудники компаний-партнеров
- 30 % приводят друзья
- 90 % пользователей оставляют свой номер телефона
- в основном абонемент покупают на 6 месяцев
- более 41% посещают груповые занятия
- основной возраст клиентов 29 лет, самому молодому 18 лет, самому старшему 41 год.
- в среднем дополнительные покупки делают на 146 у.е., медианное значение 136 у.е.
- среднее количество месяцев посещения 3,7
- среднее количество посещений в неделю за все время практически совпадает с количеством за текущий месяц и составляет ~2 посещения в неделю
- средний отток составляет 26 %
- у переменных довольно разные по величине стандартные отклонения,их возможно нужно будет стандартизировать перед обучением.

In [14]:
```
# Разделим посетителей на оставшихся и отток и построим графики:
ottok = df.groupby('churn').mean()
live = ottok.transpose()
live
```

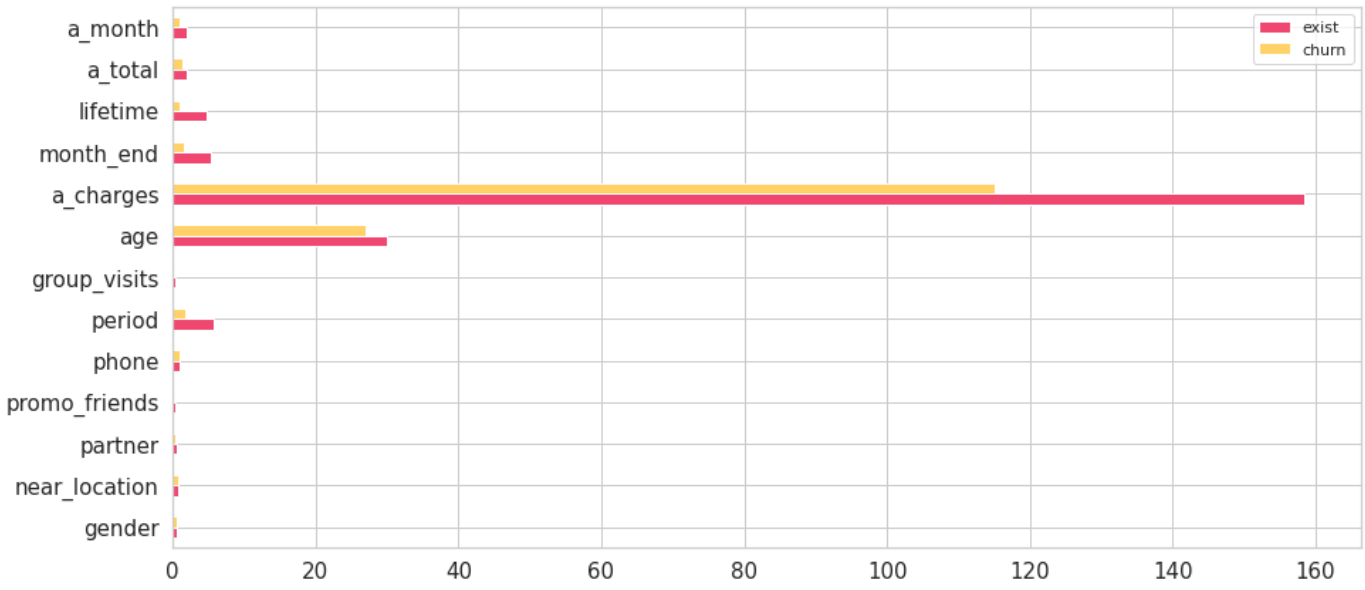
Out[14]:

churn	0	1
gender	0.510037	0.510839
near_location	0.873086	0.768143
partner	0.534195	0.355325
promo_friends	0.353522	0.183789
phone	0.903709	0.902922

churn	0	1
period	5.747193	1.728558
group_visits	0.464103	0.268615
age	29.976523	26.989632
a_charges	158.445715	115.082899
month_end	5.283089	1.662582
lifetime	4.711807	0.990575
a_total	2.024876	1.474995
a_month	2.027882	1.044546

- Вывод:
  - Среди тех кто остался, больше всего тех, кто:
    - живет рядом
    - сотрудник компании-партнера
    - пришел с другом/друзьями
    - давно (около полугода) в клубе
    - посещает групповые занятия
    - кто постарше
    - кто больше тратит на доп.услуги
    - у кого до конца абонемента еще полгода
    - у постоянных клиентов (кто давно в клубе)
    - посещает фитнес от 2-х раз в неделю

In [15]:
```
live.columns = ['exist', 'churn']
live.plot(kind='barh',figsize=(15, 7), fontsize=15)
plt.show()
```

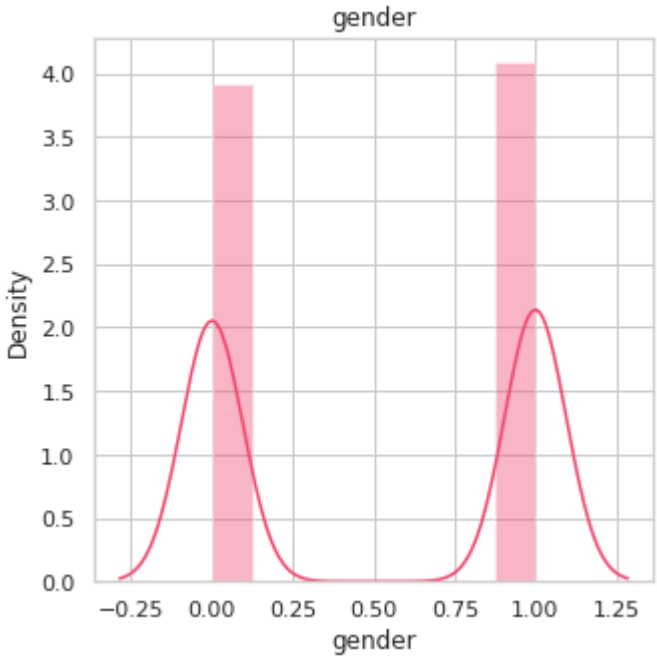


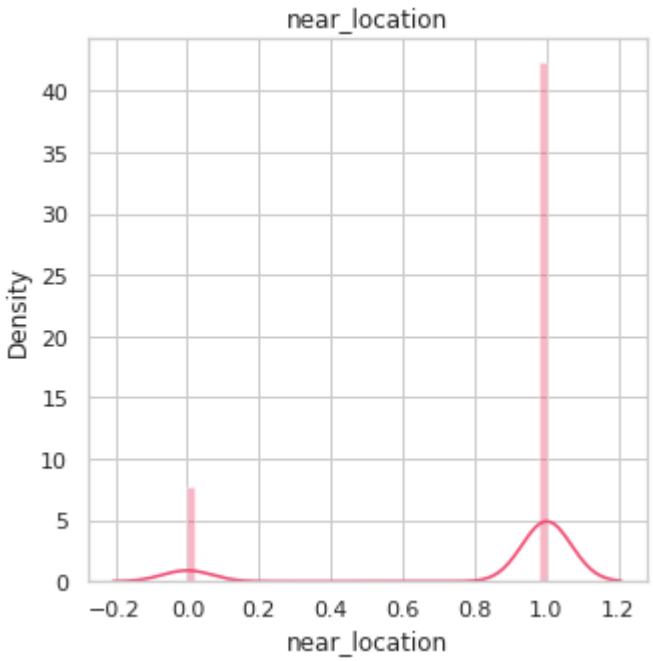
- Вывод:
  - соотношение отток - оставшиеся более заметны в колонках: "Покупки", "Длительность текущего действующего абонемента", "Срок до окончания текущего действующего абонемента", "Время с момента первого обращения в фитнес-центр"
  - в остальных колонках наименее заметно

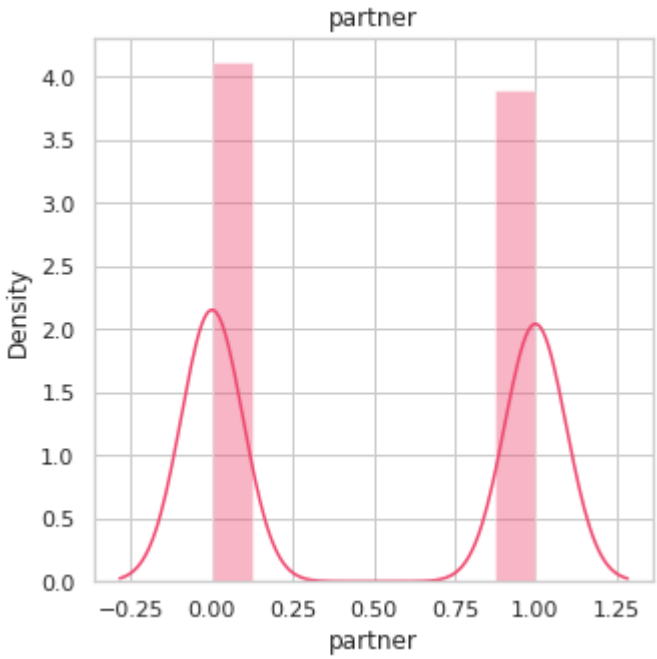
Построим столбчатые гистограммы

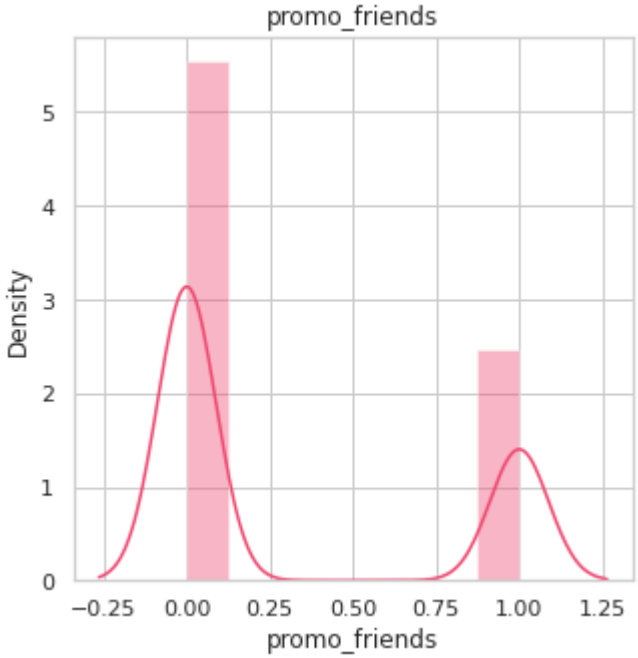
In [16]:
```
for column in df.columns:
```

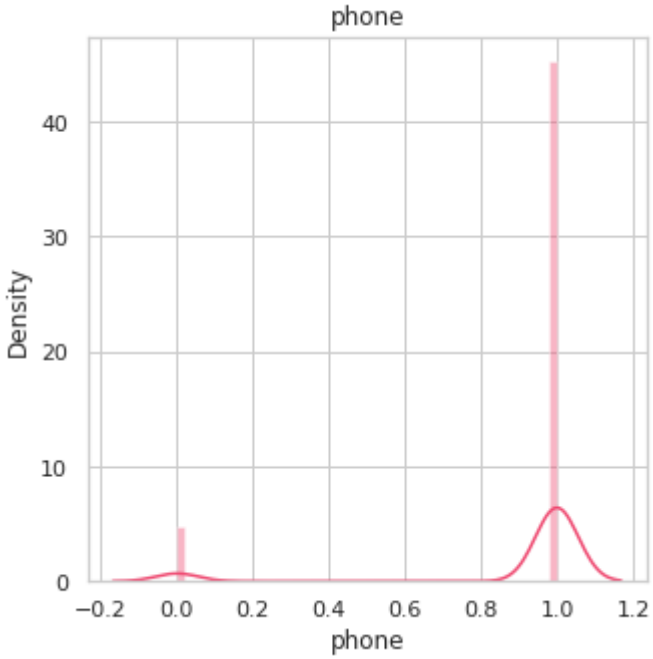
```
    plt.figure(figsize=(5, 5))
    df.groupby('churn')[column]
    sns.distplot(df[column])
    plt.title(column)
    plt.show()
```

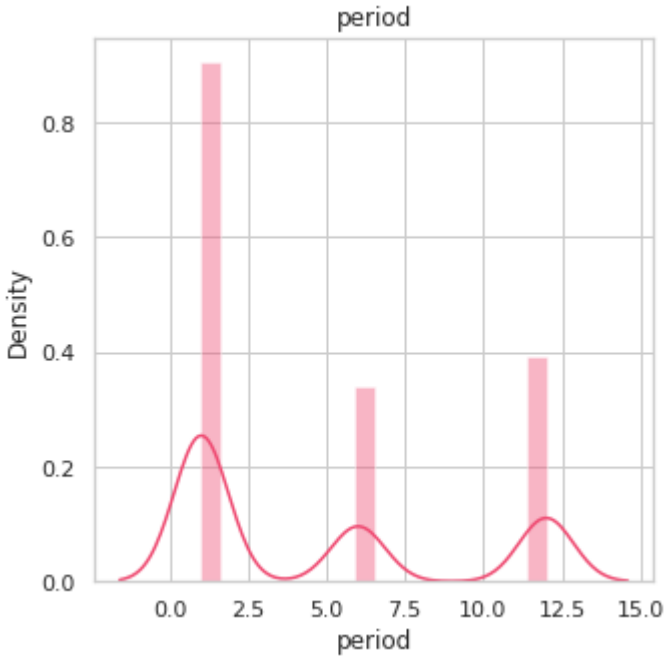


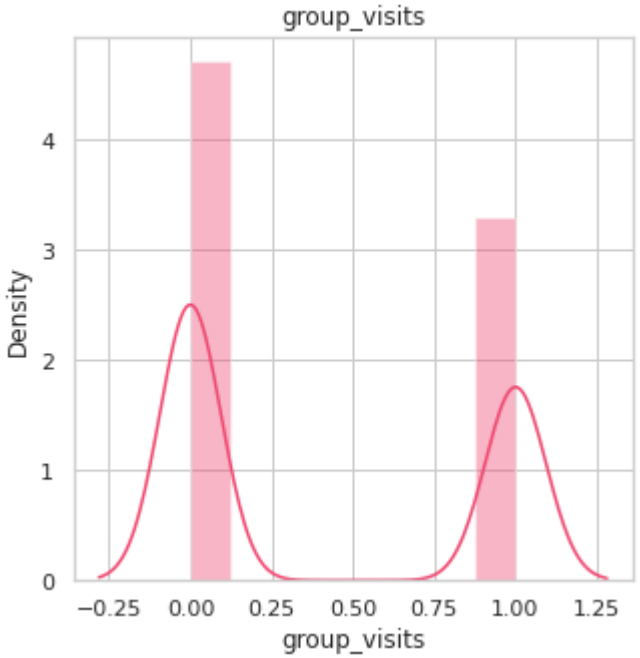


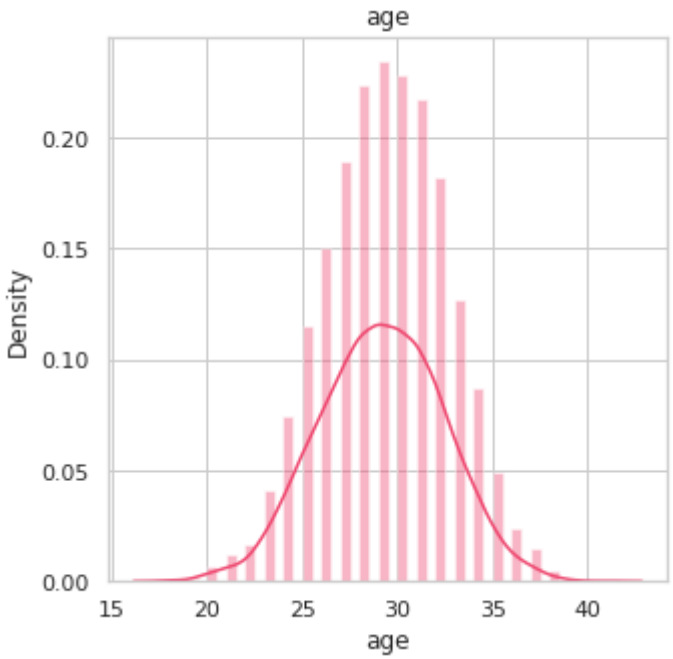

partner
Density
partner

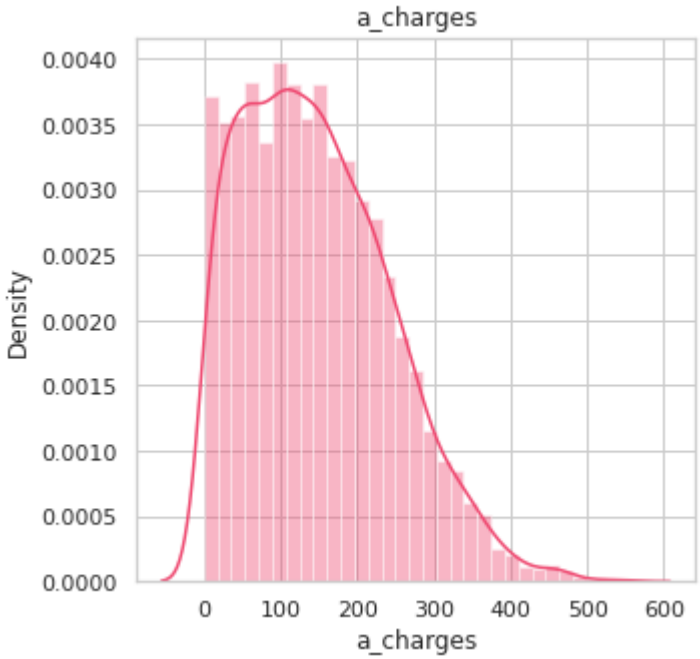

promo_friends
Density
promo_friends

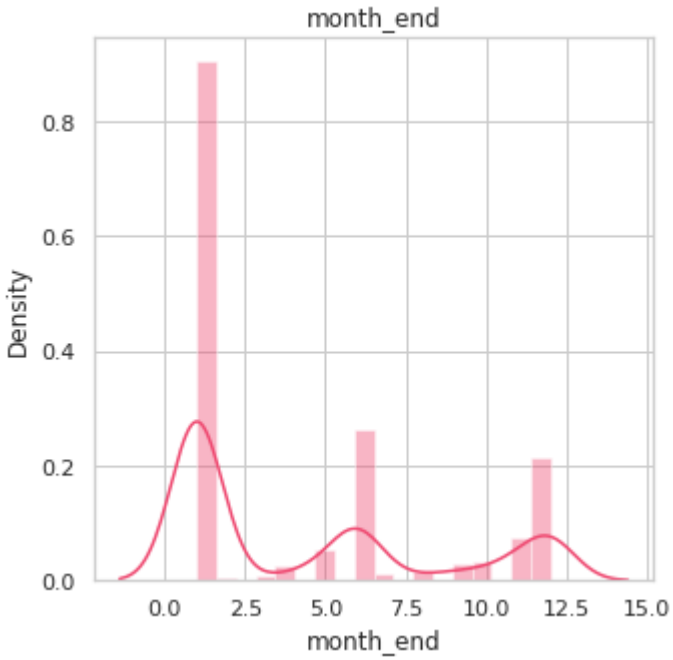

phone
Density
phone

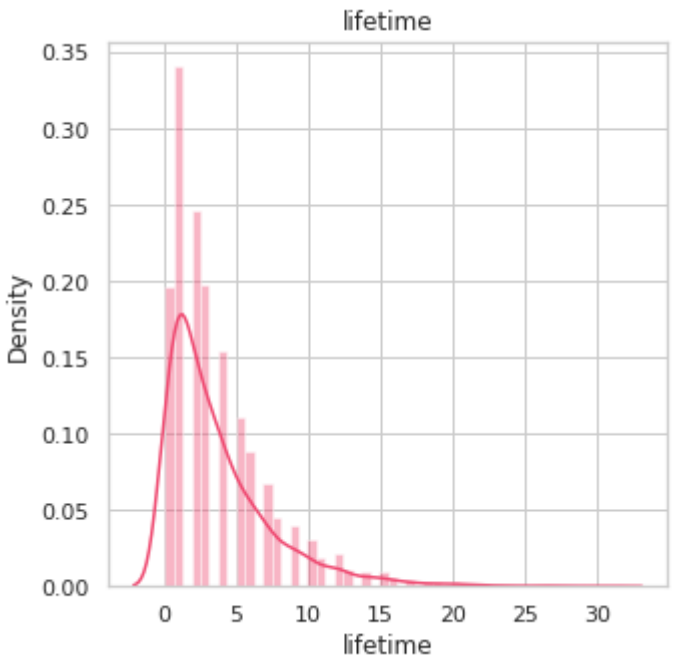

period
Density
0.0
0.2
0.4
0.6
0.8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
period

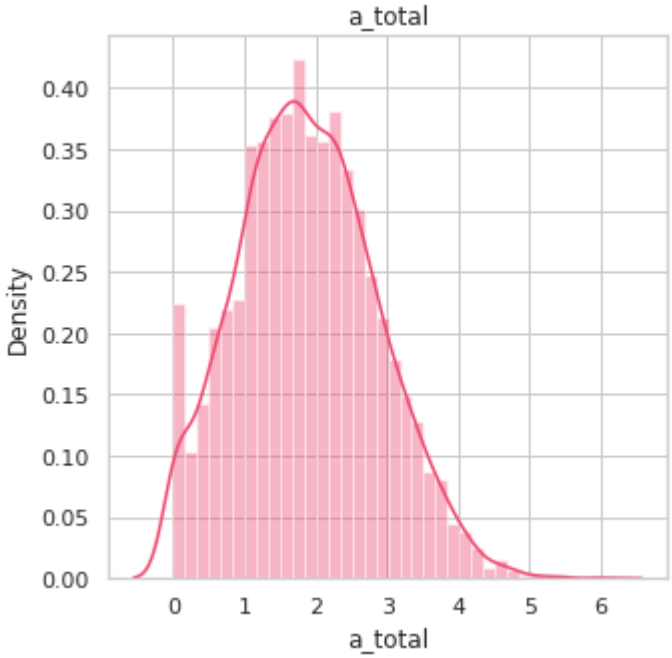

group_visits
Density
0
1
2
3
4
−0.25
0.00
0.25
0.50
0.75
1.00
1.25
group_visits

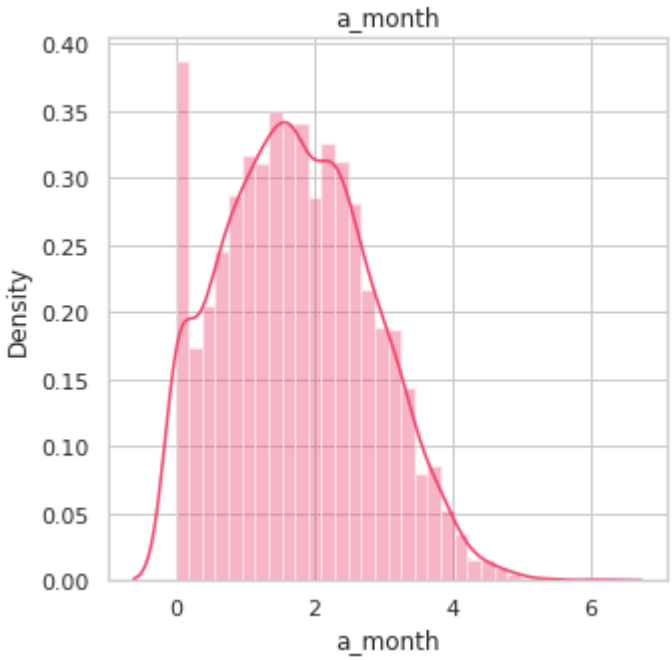

age
Density
0.00
0.05
0.10
0.15
0.20
15
20
25
30
35
40
age

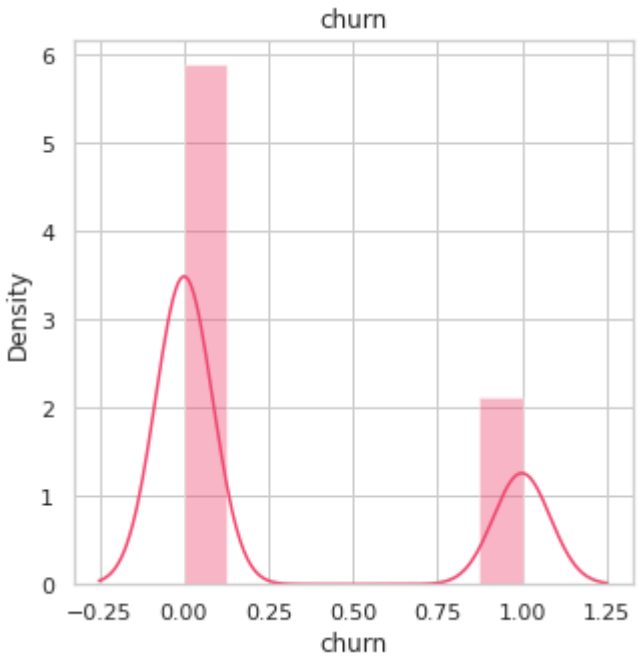

a_charges
Density
0.0040
0.0035
0.0030
0.0025
0.0020
0.0015
0.0010
0.0005
0.0000
0
100
200
300
400
500
600
a_charges


month_end
Density
0.8
0.6
0.4
0.2
0.0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
month_end


lifetime
Density
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
0
5
10
15
20
25
30
lifetime



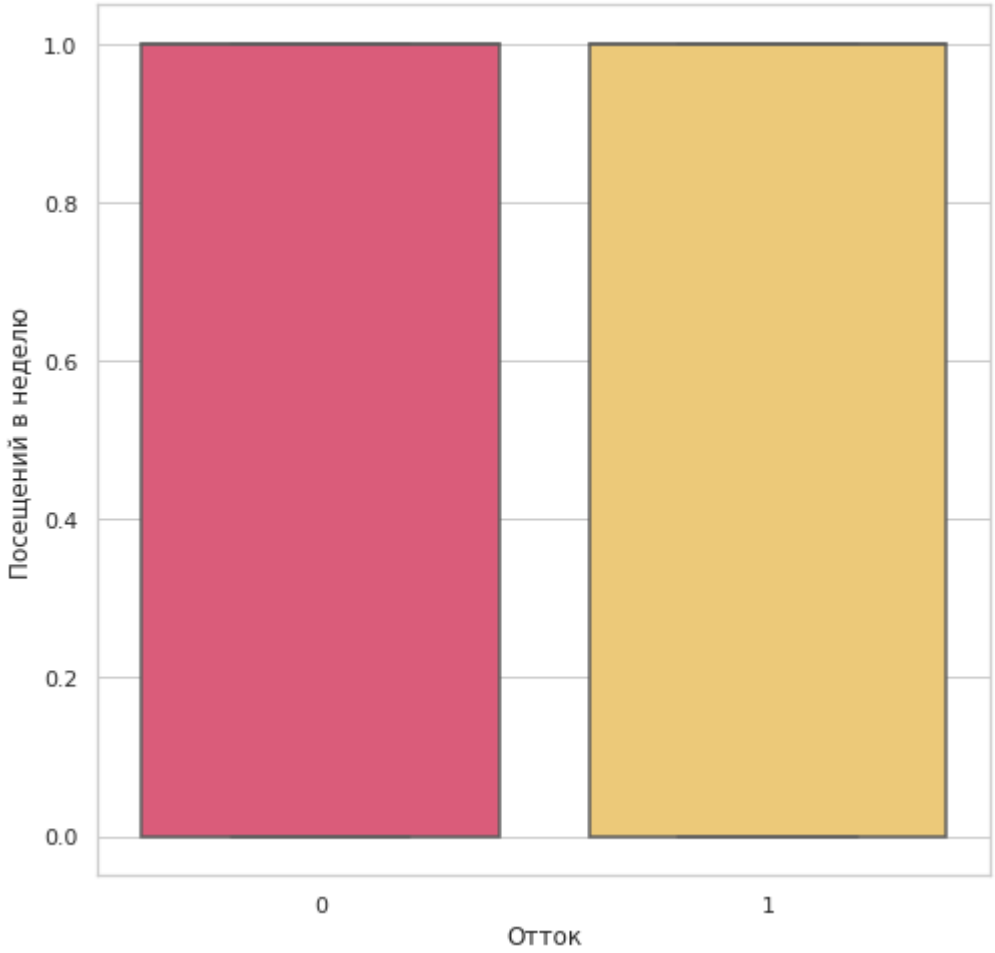


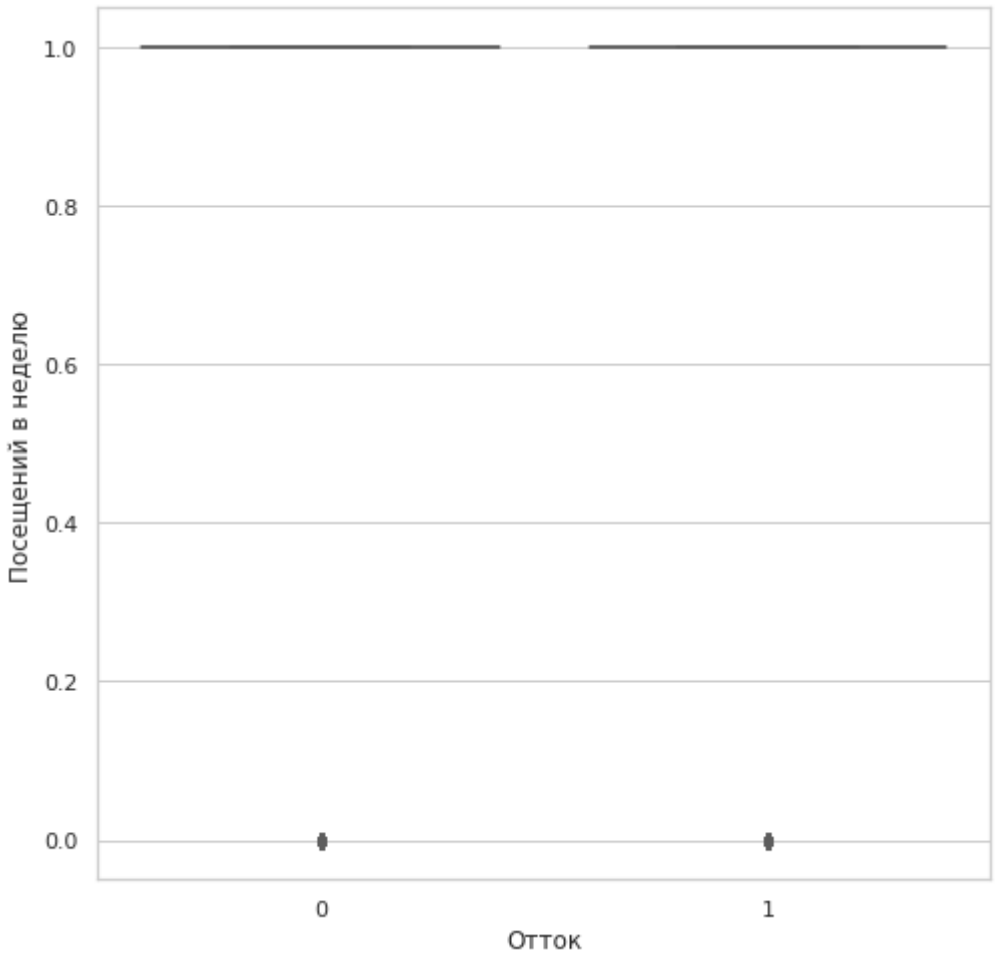


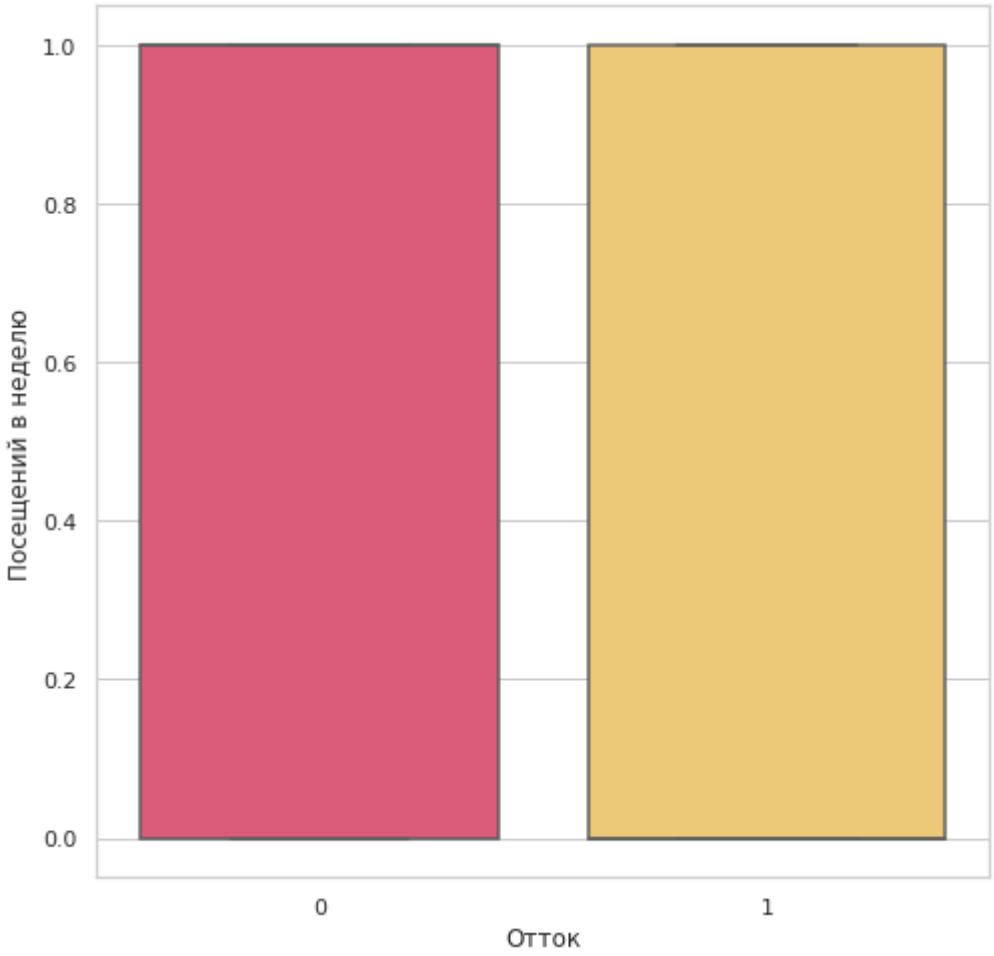
In [17]:

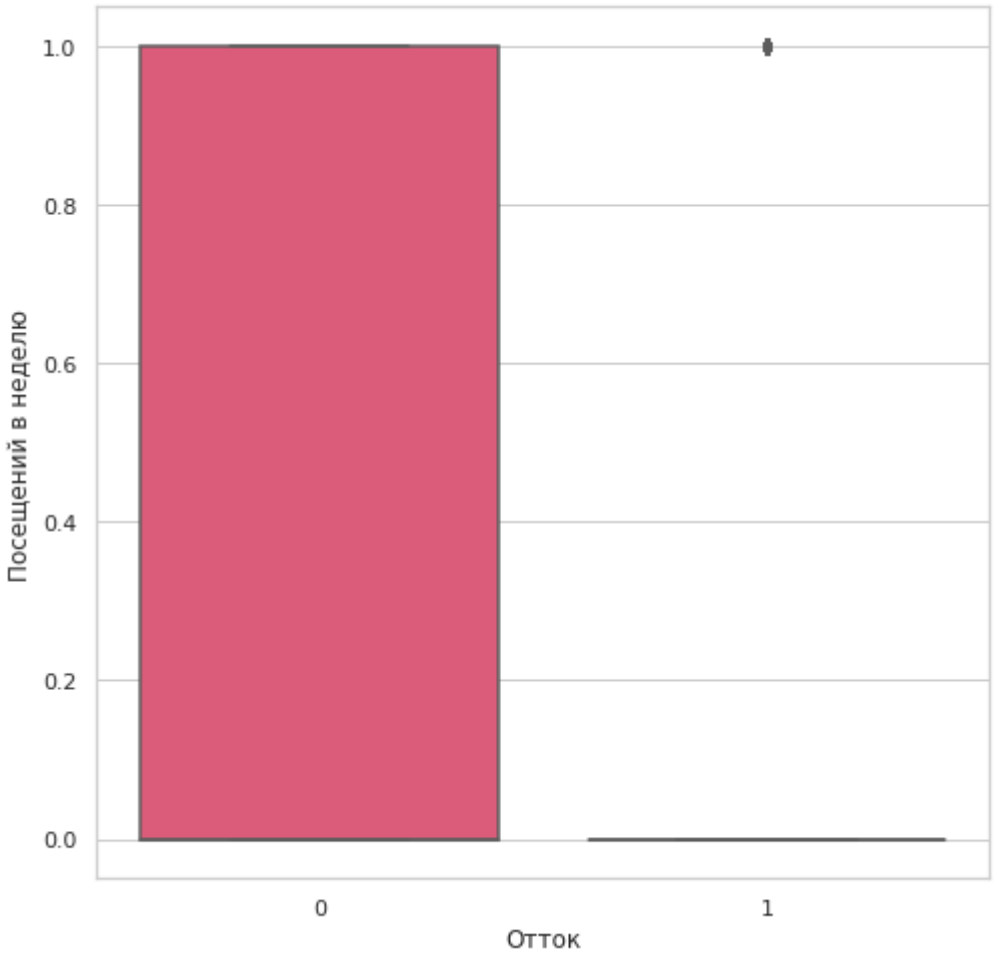
```
for column in df.columns:
    plt.figure(figsize=(8, 8))
    sns.boxplot(x = 'churn', y = df[column], data = df)
    plt.xlabel("Отток")
```

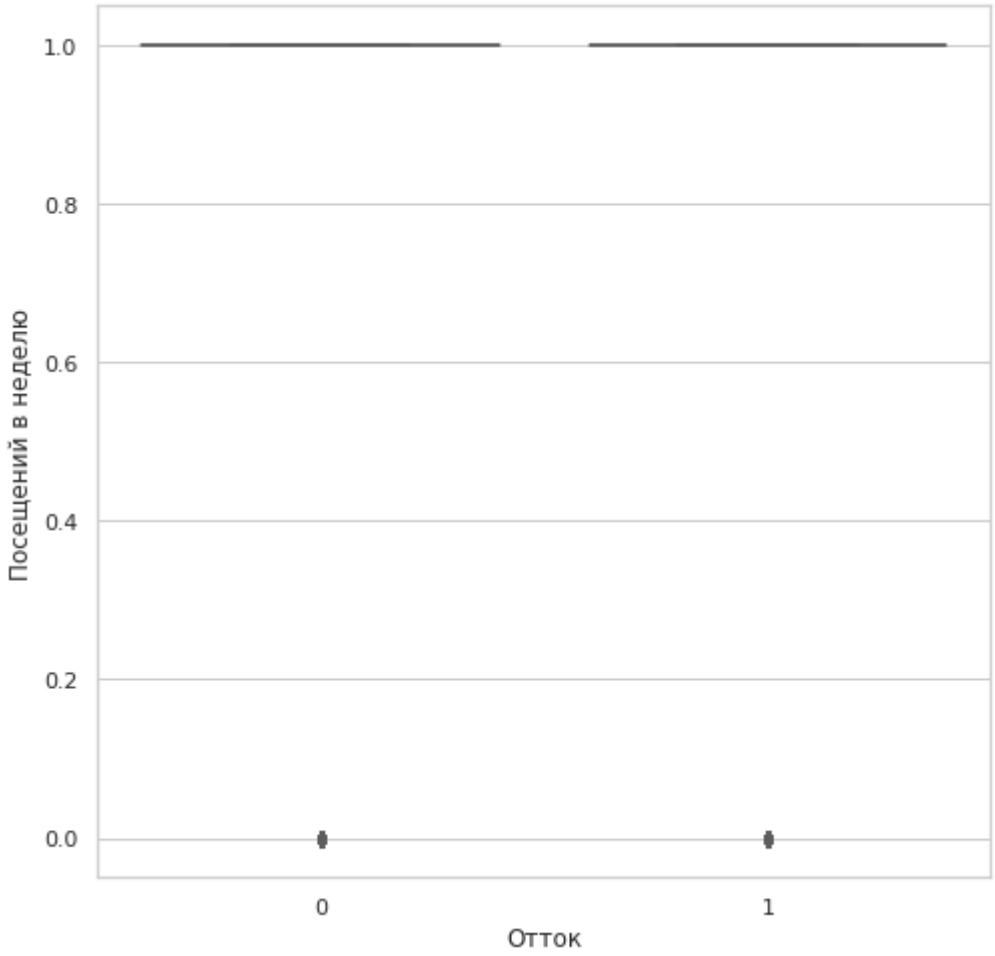
```
    plt.ylabel("Посещений в неделю")
    plt.show()
```

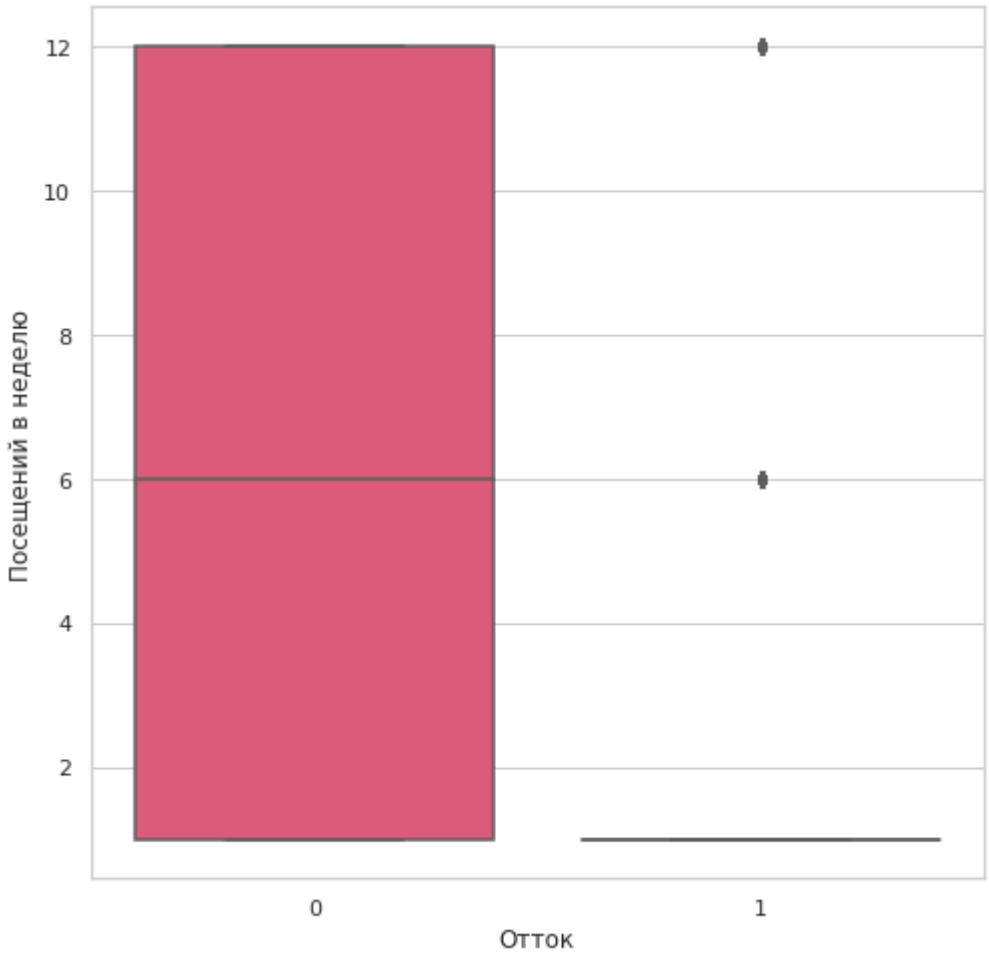


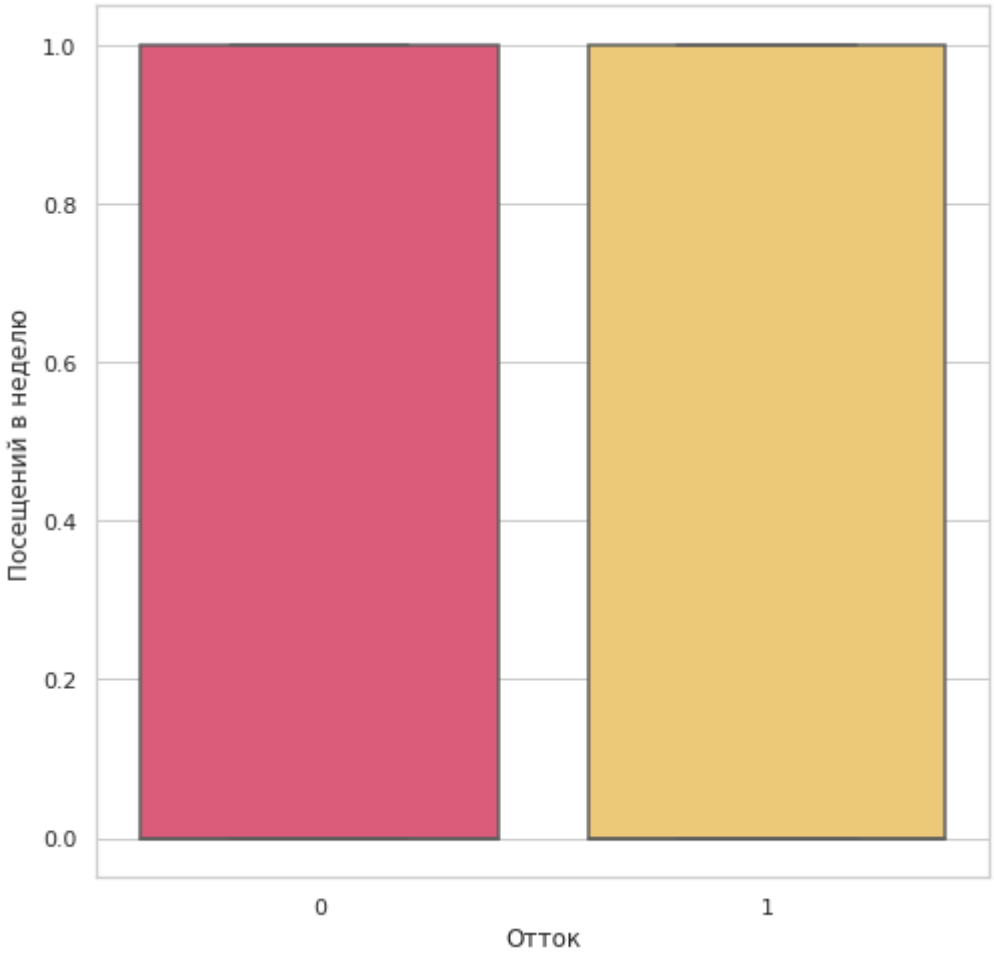


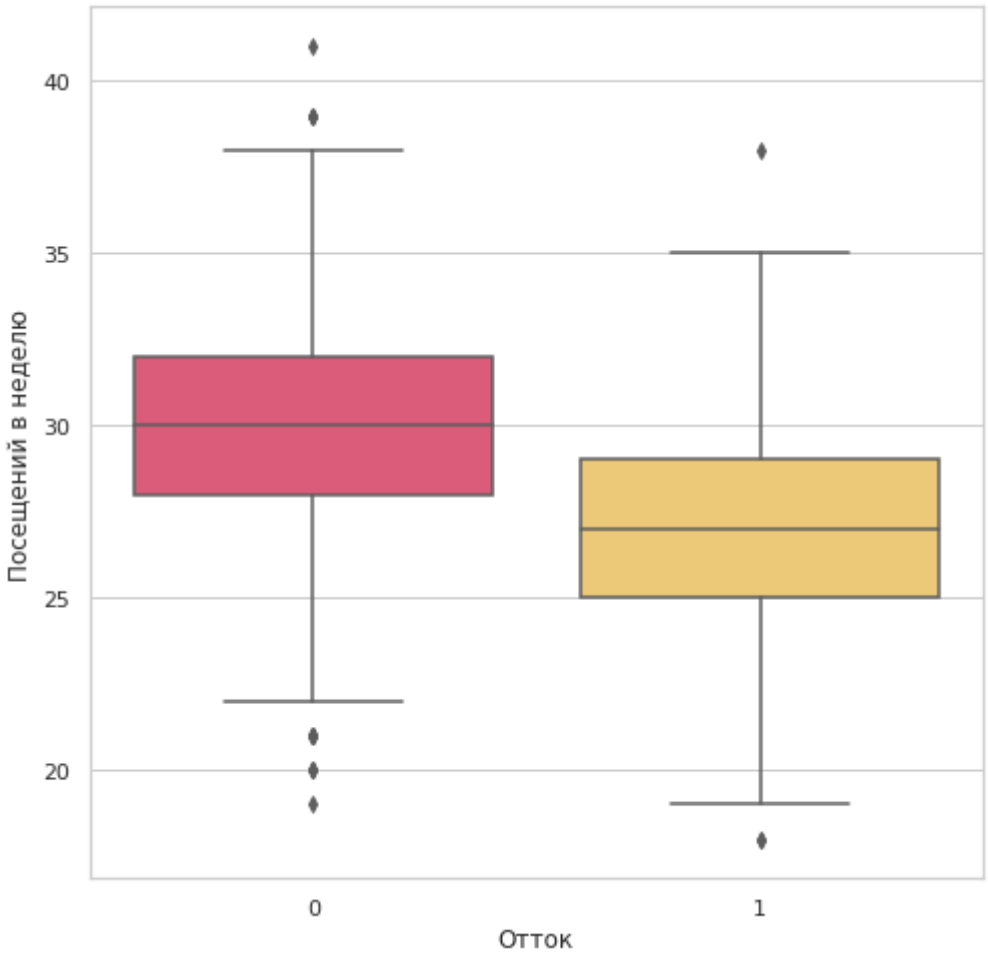

1.0
0.8
0.6
0.4
0.2
0.0
Посещений в неделю
0
1
Отток

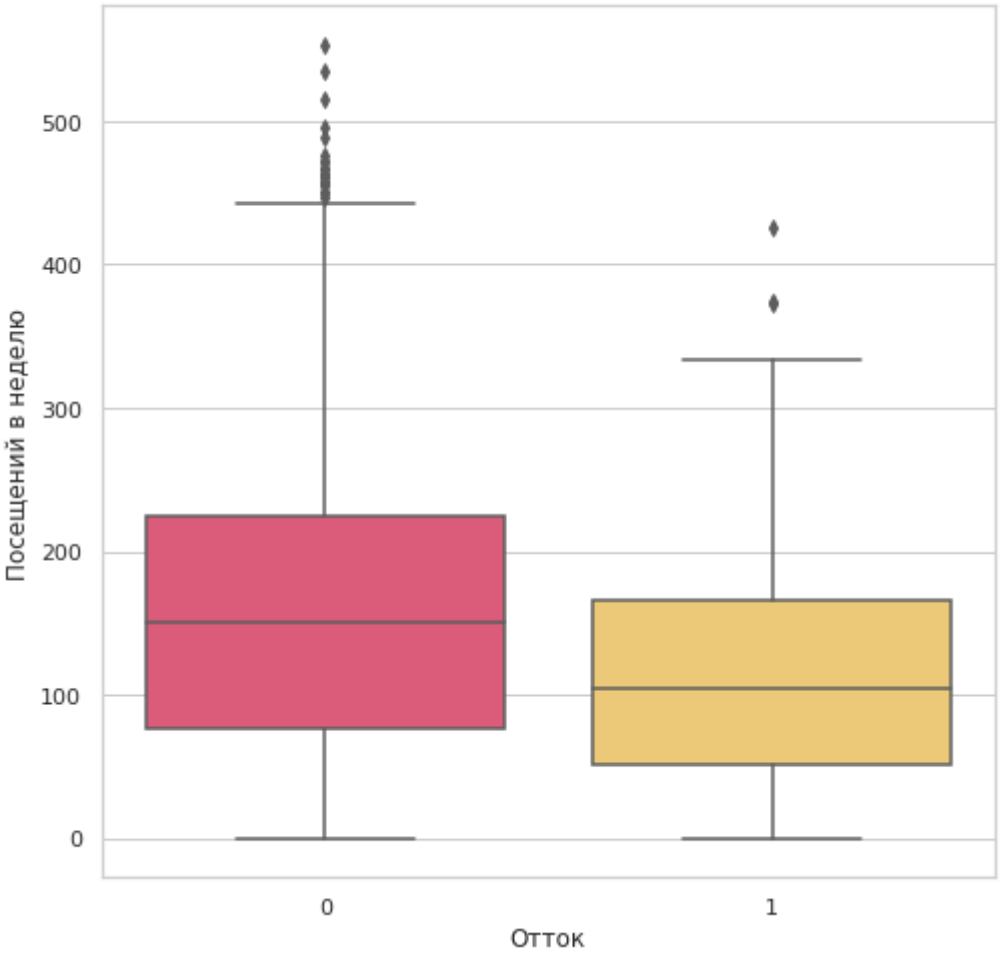

1.0
0.8
0.6
0.4
0.2
0.0
Посещений в неделю
0
1
Отток

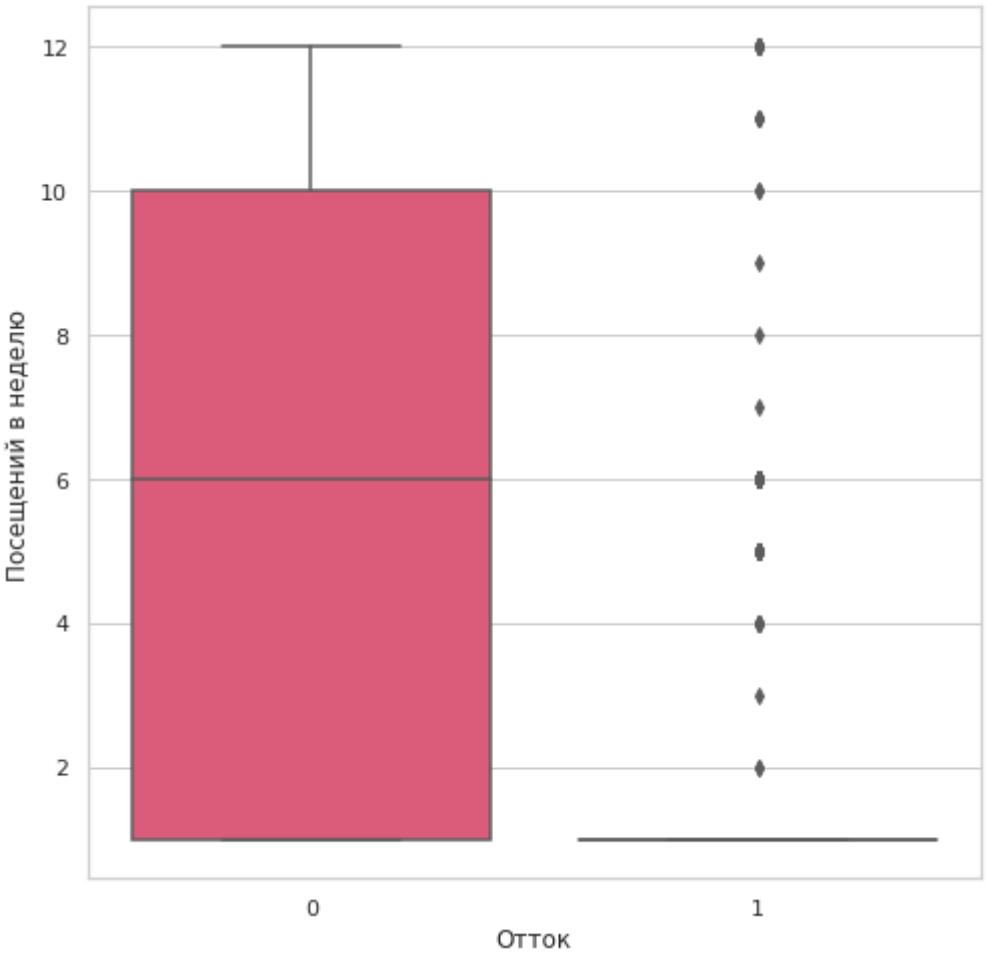

1.0
0.8
0.6
0.4
0.2
0.0
Посещений в неделю
0
1
Отток

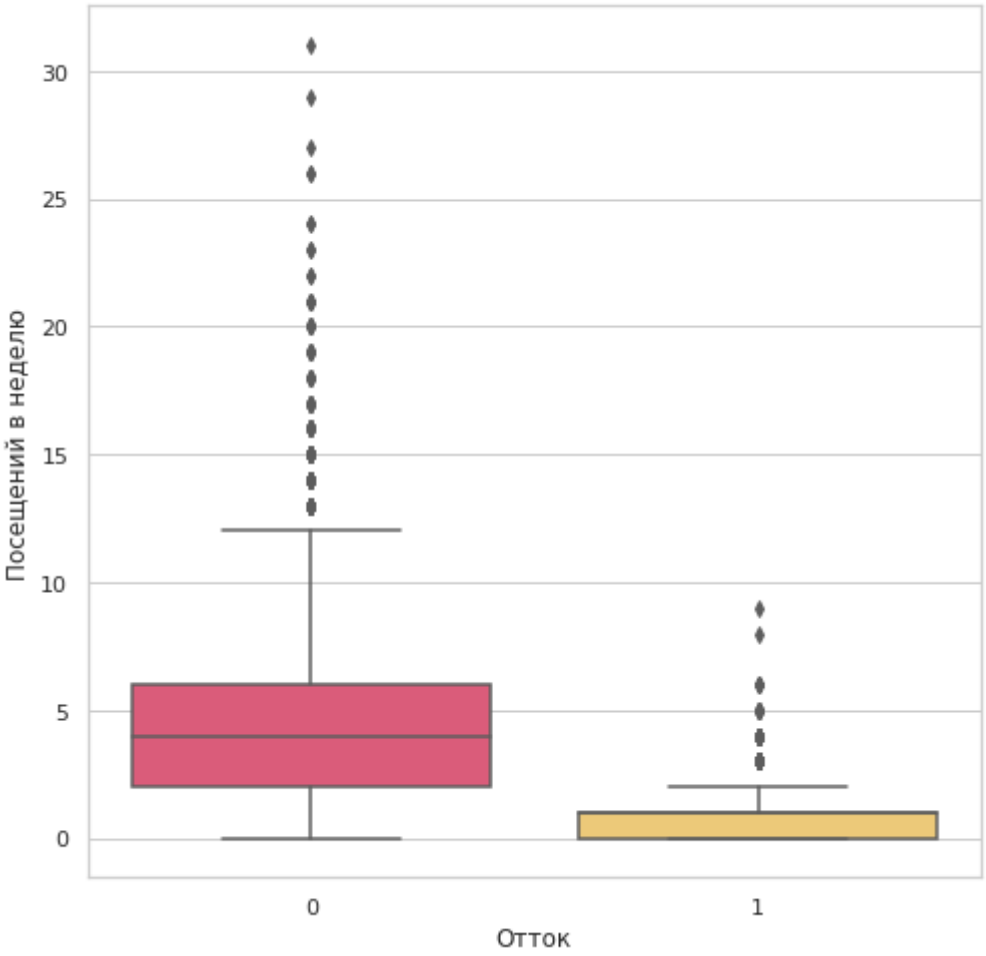

12
10
8
6
4
2
Посещений в неделю
0
1
Отток

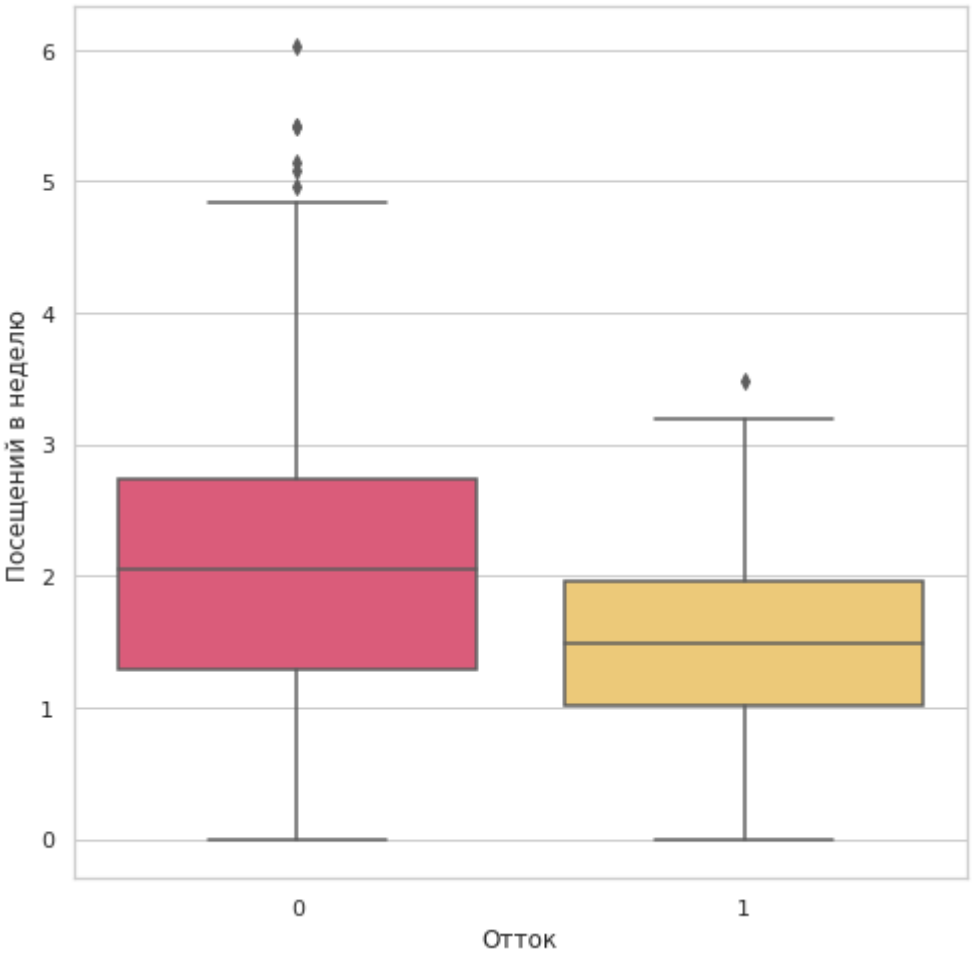

1.0
0.8
0.6
0.4
0.2
0.0
Посещений в неделю
0
1
Отток

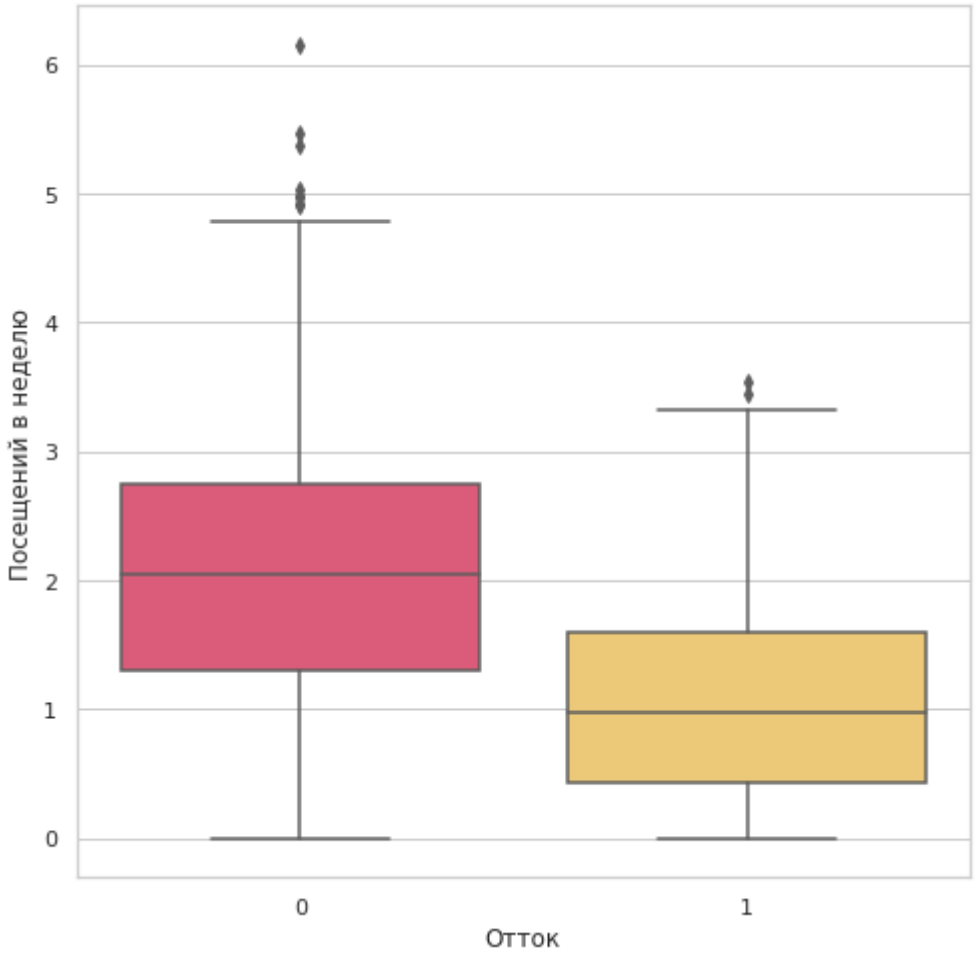

40
35
30
25
20
Посещений в неделю
0
1
Отток

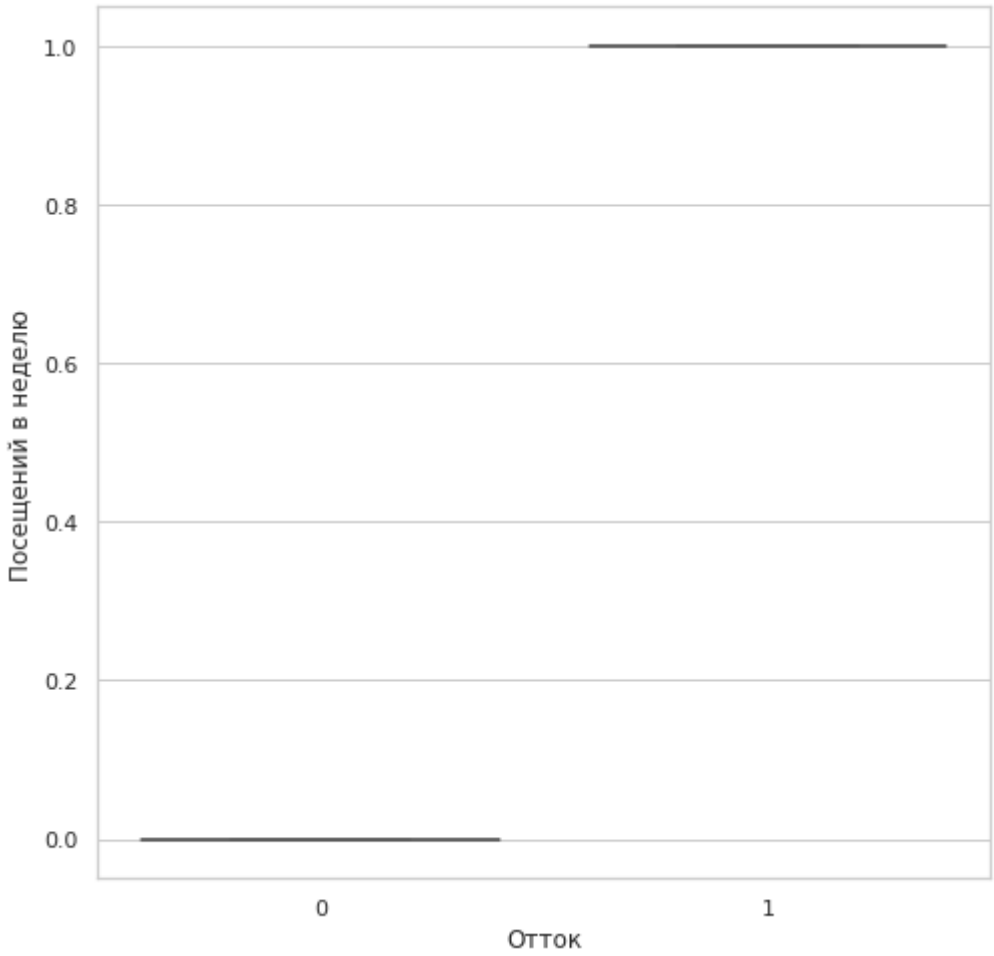

500
400
300
200
100
0
Посещений в неделю
0
1
Отток


12
10
8
6
4
2
Посещений в неделю
0
1
Отток


Посещений в неделю
30
25
20
15
10
5
0
0
1
Отток


Посещений в неделю
6
5
4
3
2
1
0
0
1
Отток





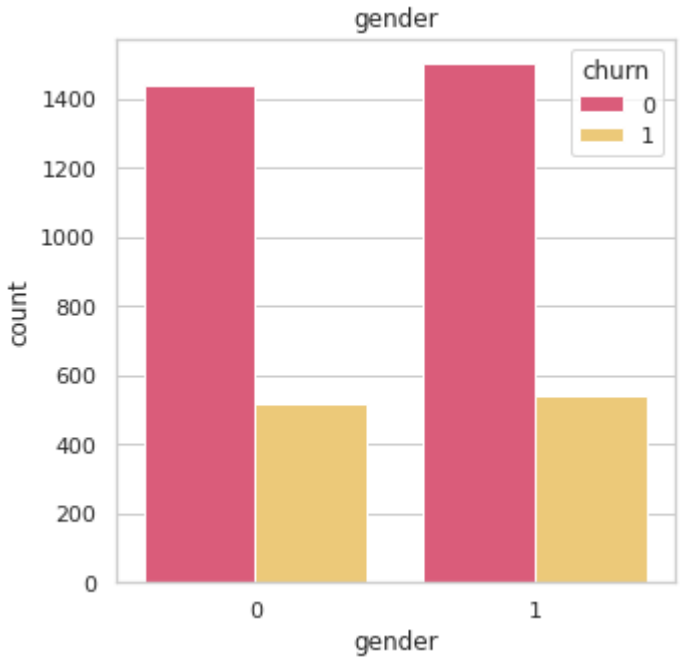
In [18]:
```
# Посмотрим на средние значения признаков:
df.groupby('churn')[column].agg('count').describe()
```

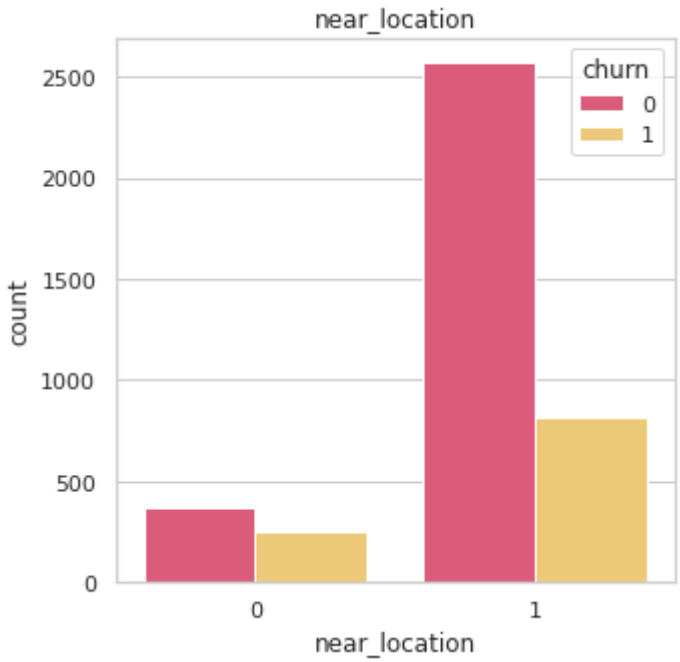
Out[18]:
```
count       2.000000
mean     2000.000000
std      1327.946535
min      1061.000000
```

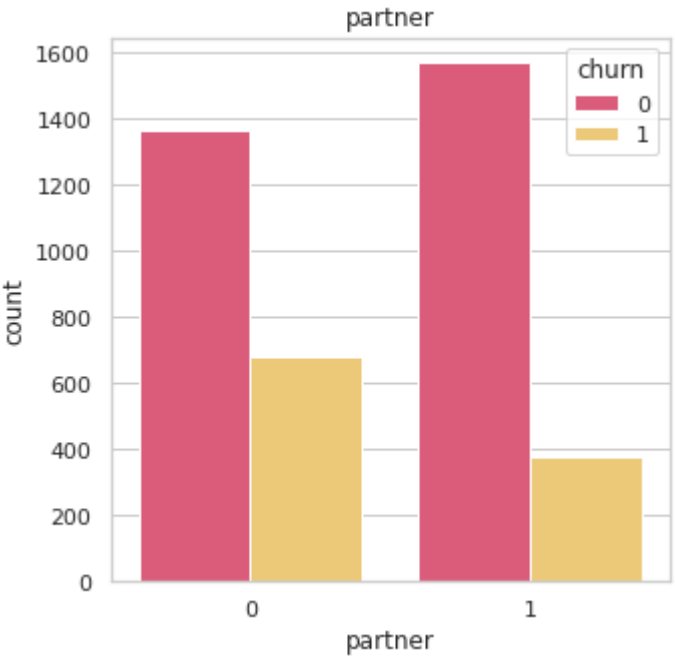
```
25%      1530.500000
50%      2000.000000
75%      2469.500000
max      2939.000000
Name: churn, dtype: float64
```

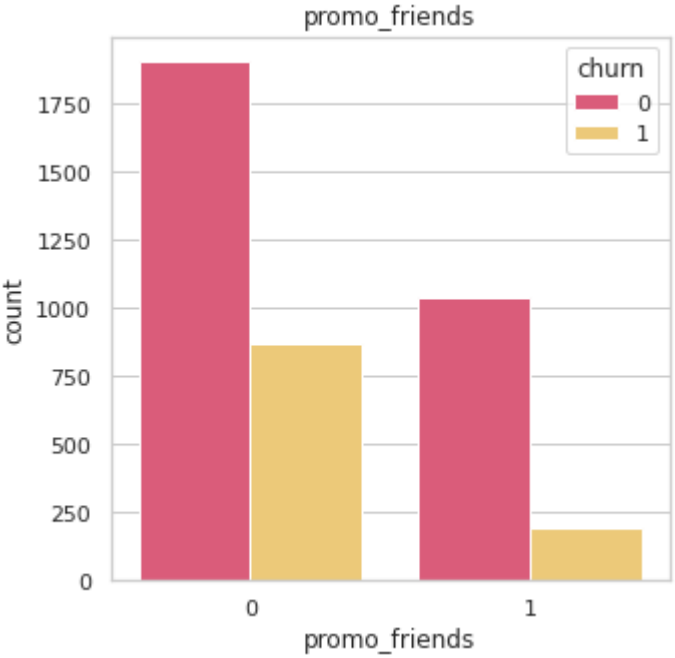
In [19]:

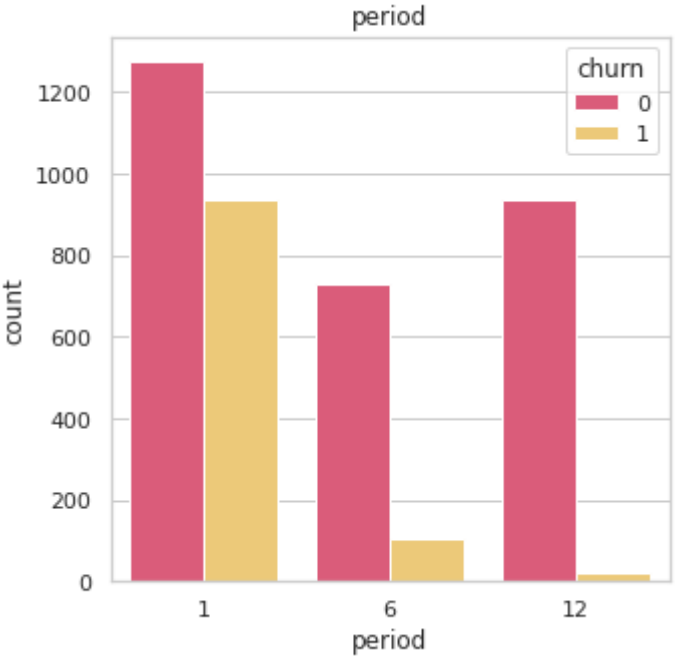
```
for column in ['gender', 'near_location', 'partner', 'promo_friends', 'period']:
    plt.figure(figsize=(5, 5))
    sns.countplot(x = df[column], hue='churn', data=df)
    plt.title(column)
    plt.show()
```











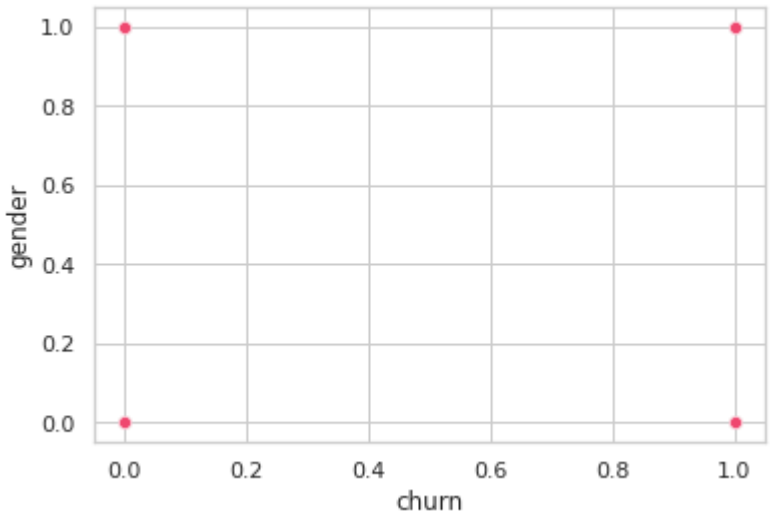
- Вывод:
  - отток не зависит от пола, одинаков у мужчин и женщин
  - отток зависит от расположения- среди приезжих доля покинувших очень велика

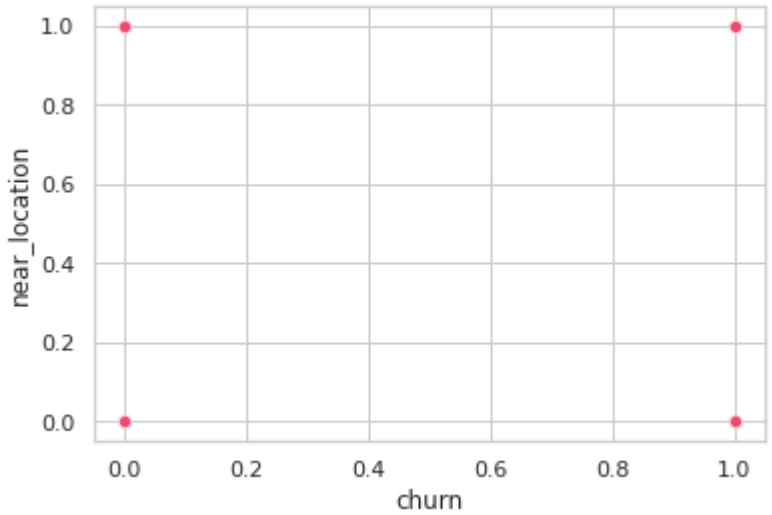
- доля покинувших среди клиентов фирм- партнеров значительно меньше
- среди пришедших по промо-коду доля покинувших незначительна
- наибольшее число покинувших у клиентов с месячным абонементом, с годовым - близка к 0

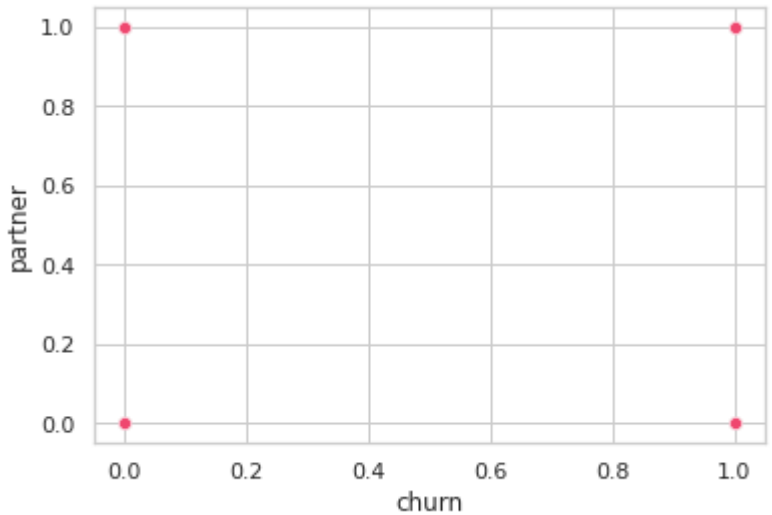
Построим графики распределения признаков.

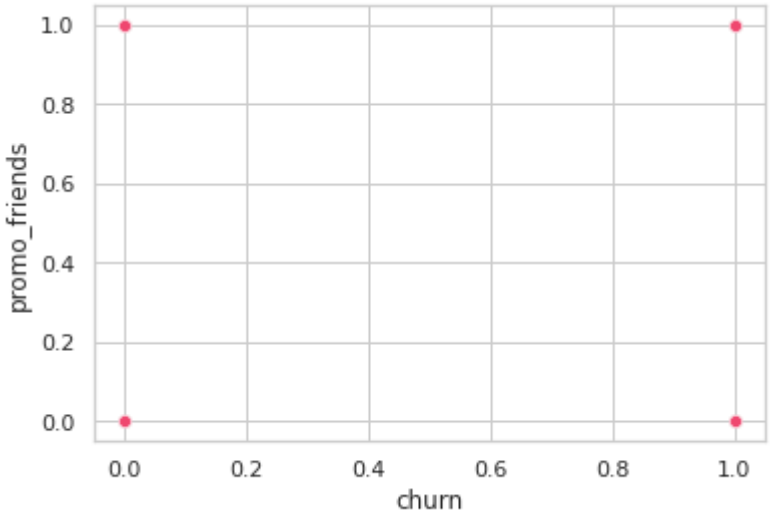
In [20]:

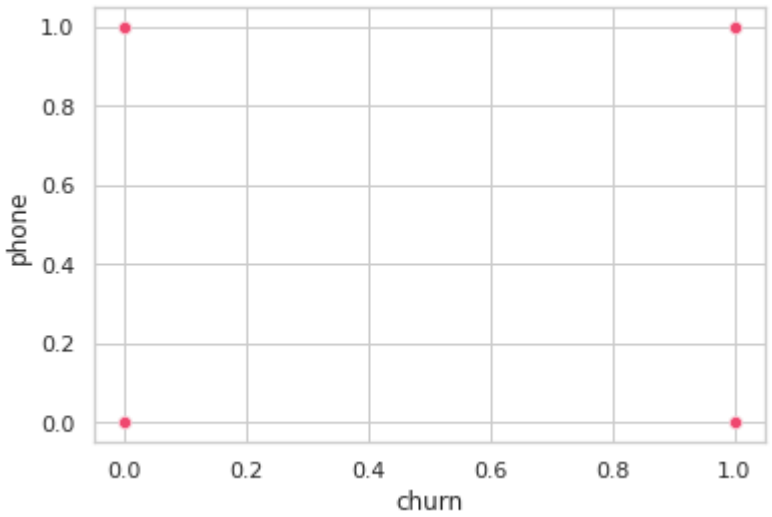
```
# Диаграммы распределения признаков для тех, кто ушёл (отток) и тех, кто остался (не попали в о
for col in df.drop('churn', axis = 1).columns:
    sns.scatterplot(x=df['churn'], y=df[col])
    plt.show()
```

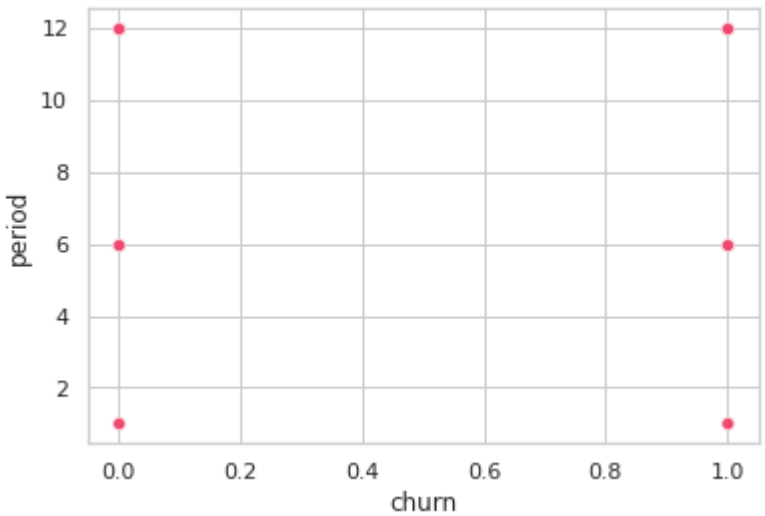


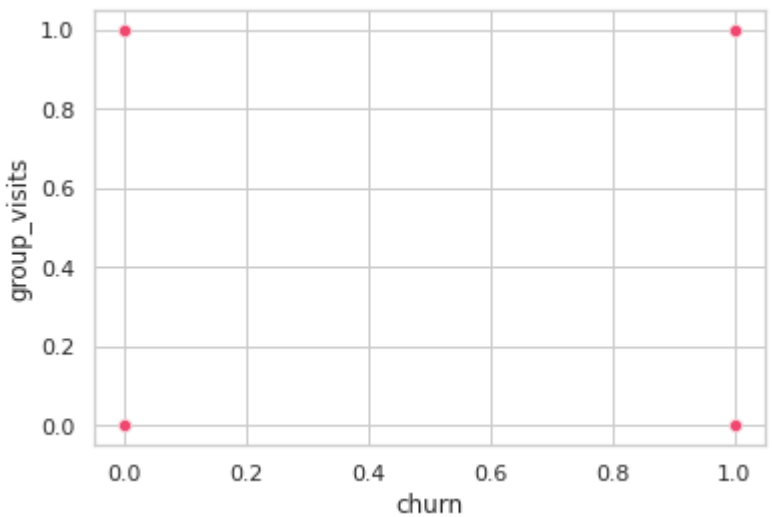


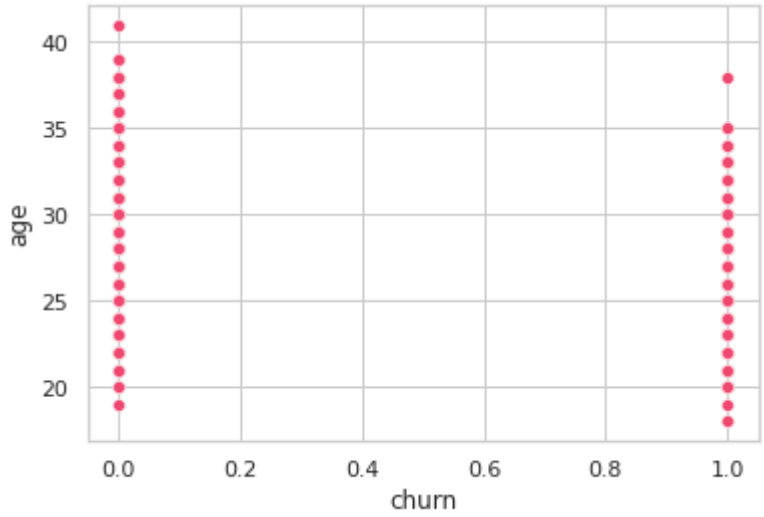


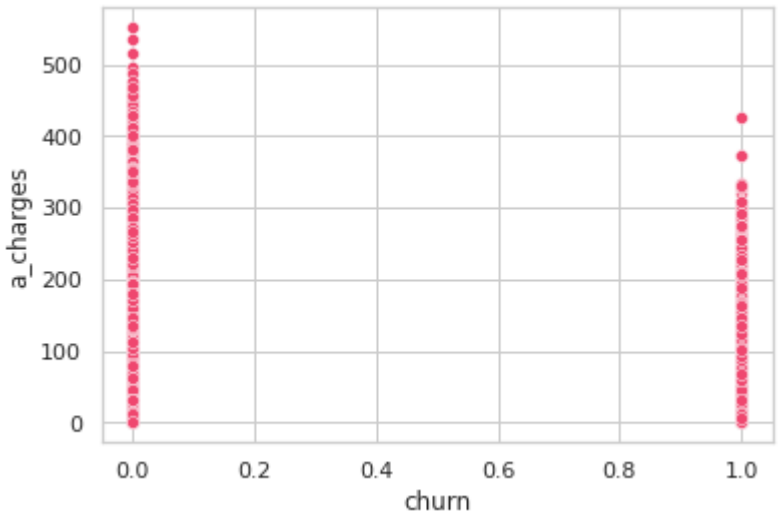

promo_friends
churn
0.0
0.2
0.4
0.6
0.8
1.0

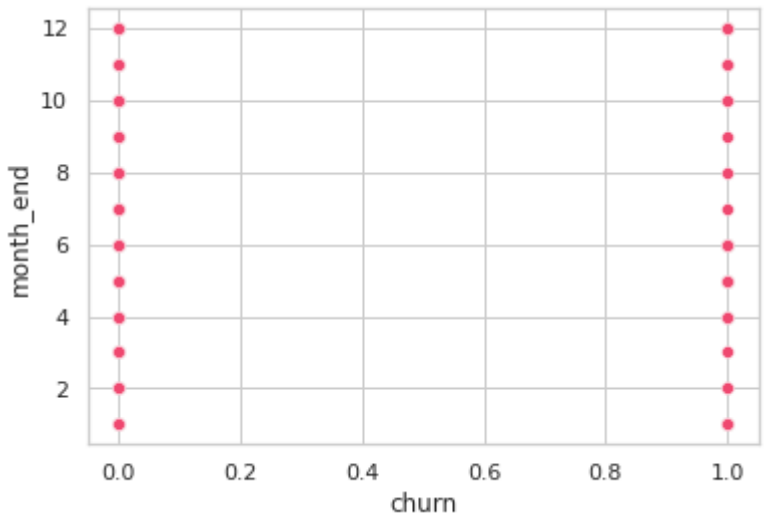

phone
churn
0.0
0.2
0.4
0.6
0.8
1.0

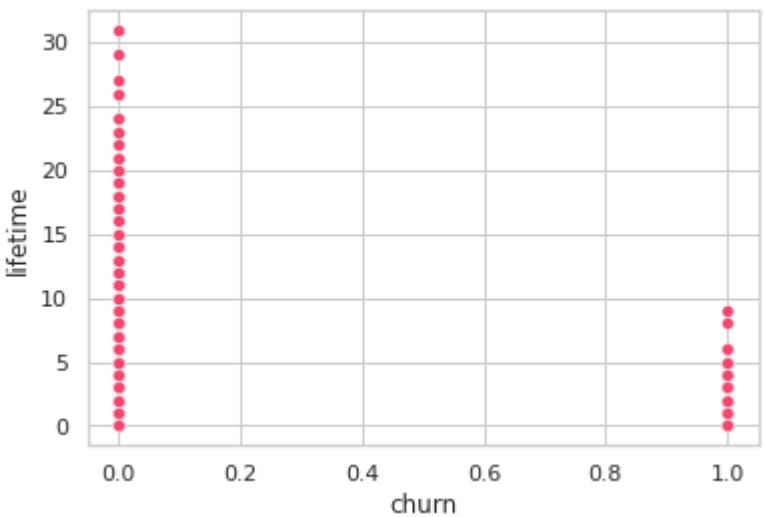

period
churn
2
4
6
8
10
12
0.0
0.2
0.4
0.6
0.8
1.0

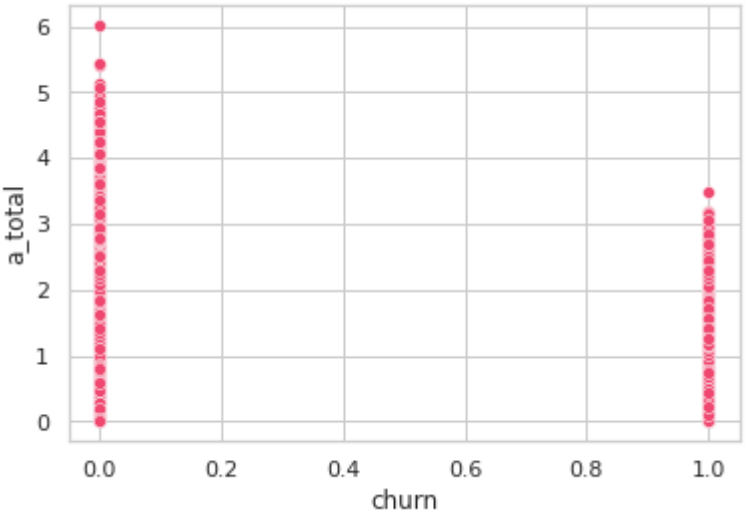

group_visits
churn
0.0
0.2
0.4
0.6
0.8
1.0

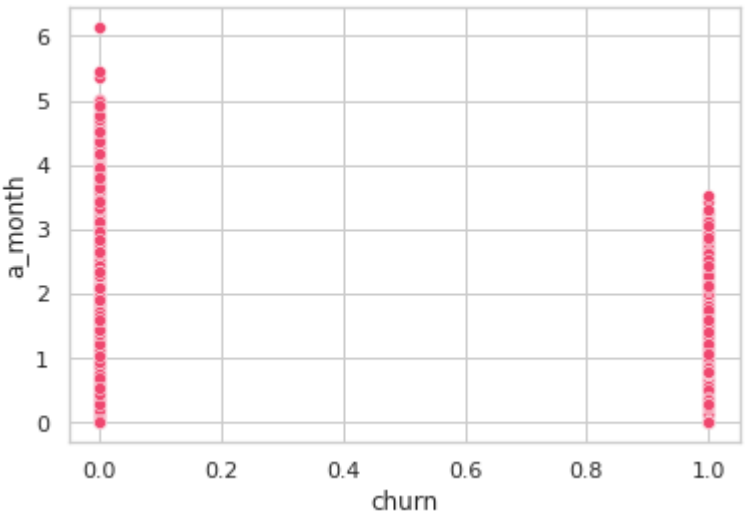

age
churn
40
35
30
25
20
0.0
0.2
0.4
0.6
0.8
1.0


a_charges
churn
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0


month_end
churn
12
10
8
6
4
2
0.0
0.2
0.4
0.6
0.8
1.0


lifetime
churn
30
25
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

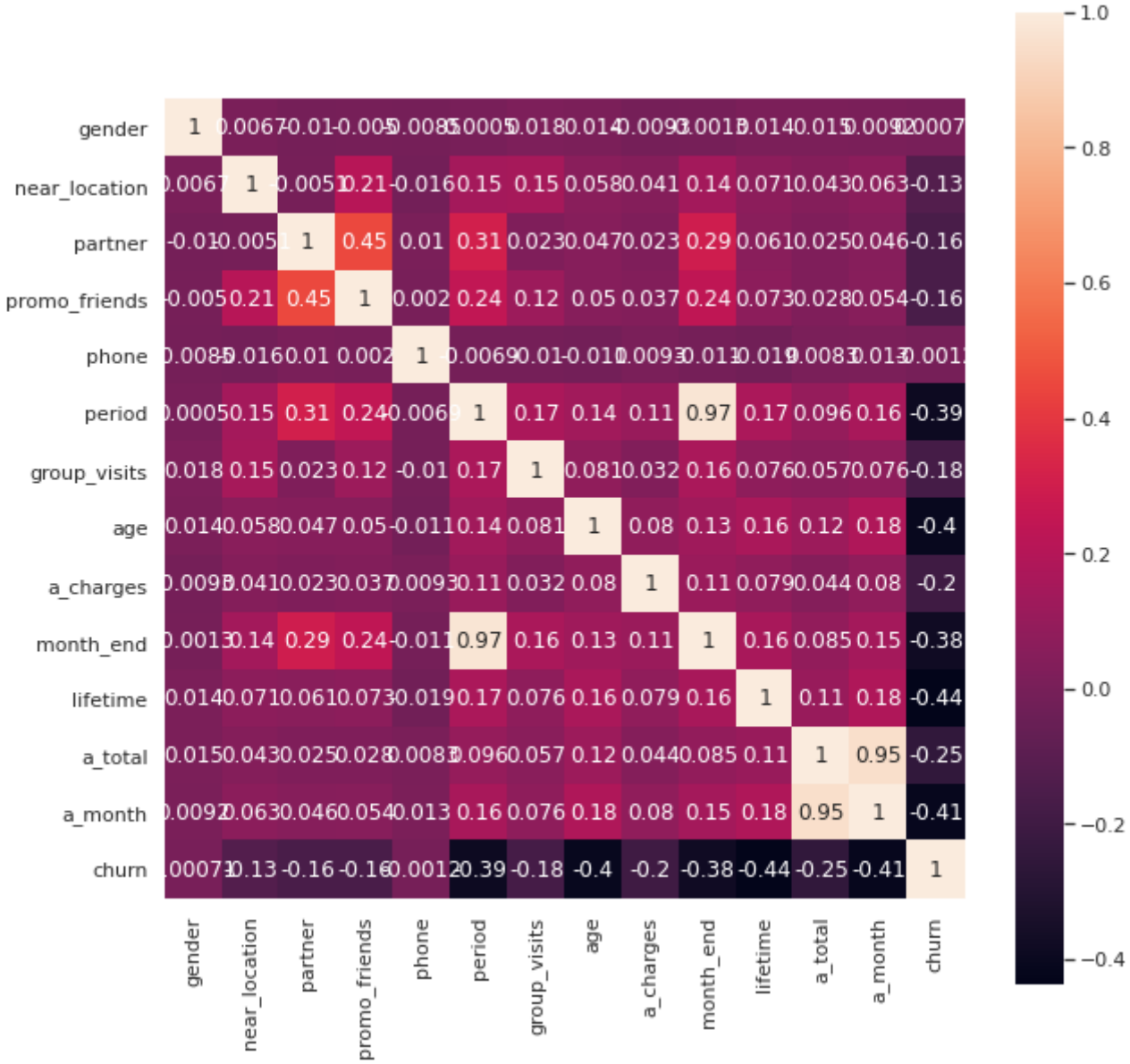




Построим матрицу корреляций и отобразим её.

In [21]:

```
#вычисляем матрицу корреляций
cm = df.corr()
plt.figure(figsize = (10,10))
sns.heatmap(cm, annot=True, square=True)#ваш код здесь
plt.show()
```



Выводы:

- По графикам можно сделать выводы об оставшихся и оттоке:
  - отток почти не зависит от пола;
  - большинство посетителей проживает поблизости;
  - посетители,котрые могут получать скидки на абонемент от работодателей, чаще остаются;
  - среди ушедших большинство имело месячный абонемент.У оставшихся - на 6 мес и 12 мес;
  - среди ушедших большинство одиночки, у оставшихся много групповых занятий;
  - среди ушедших большинство 26-28 лет,у оставшихся около 30;
  - суммарную выручку от других услуг дают почти одинаковую;
  - большинство уходит в первый месяц, оставшиеся посещают до 6 мес и более;
  - средняя частота посещения в неделю за весь период - около 2 раз;
  - средняя частота посещения в неделю за предыдущий месяц различается. У ушедших - около 1 раза, у оставшихся - около 2.5 раз;
- Признаки наиболее сильно коррелирующие с целевой переменной (отток):
  - средняя частота посещения в неделю за предыдущий месяц;
  - возраст;
  - время с момента первого обращения в фитнес-центр;
- Признаки наиболее сильно коррелирующие между собой:
  - срок до окончания текущего действующего абонемента и длительность текущего действующего абонемента;
  - средняя частота посещений в неделю за предыдущий месяц и средняя частота посещений в неделю за все время с начала действия абонемента;

- Каких - то ярких выбросов и перекосов не наблюдается. На этих данных можно строить модель. Но предварительно нужно удалить сильно коррелирующие столбцы.

In [22]:
```
#Удалим столбцы
df1=df.drop(['month_end','a_total'],axis = 1)
```

Построим модель прогнозирования оттока клиентов

Разделим наши данные на признаки (матрица X) и целевую переменную (y)

In [23]:
```
#разделим наши данные на признаки (матрица X) и целевую переменную (y)
X = df1.drop('churn', axis = 1)
y = df1['churn']
#разделяем модель на обучающую и валидационную выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
#создадим объект класса StandardScaler и применим его к обучающей выборке
scaler = StandardScaler()
scaler.fit(X_train)
#обучаем scaler и одновременно трансформируем матрицу для обучающей выборки
X_train_st = scaler. fit_transform(X_train)
print(X_train_st[:5])
#применяем стандартизацию к матрице признаков для тестовой выборки
X_test_st = scaler.transform(X_test)
```

```
[[-1.01511421  0.4175068   1.03175391  1.4800097   0.31628211  1.60502986
  -0.84769226  0.57944798  0.37161711  1.12734972  1.61822807]
 [-1.01511421  0.4175068  -0.96922337 -0.67567125  0.31628211 -0.81299073
  -0.84769226  0.27046055 -1.09697378  5.88138322 -0.01340886]
 [ 0.98511083  0.4175068   1.03175391 -0.67567125  0.31628211  1.60502986
  -0.84769226 -0.65650171 -1.18374157  0.3350108  -0.80541199]
 [-1.01511421  0.4175068   1.03175391  1.4800097   0.31628211  0.28610954
  -0.84769226 -0.96548914 -0.95158829  1.39146269  0.64605224]
 [-1.01511421  0.4175068  -0.96922337 -0.67567125 -3.16173427 -0.81299073
  -0.84769226 -0.03852687  0.97190435  0.07089783 -0.16038147]]
```

Обучим модель на train-выборке двумя способами:логистической регрессией и случайным лесом.

Зададим список моделей и напишем цикл, который выводит метрики по списку моделей

In [24]:
```
# Для логистической регрессии
models = [

    LogisticRegression()
]
# функция, которая вычисляет MAPE
def mape(y_true, y_pred):
    y_error = y_true - y_pred
    y_error_abs = [abs(i) for i in y_error]
    perc_error_abs = y_error_abs / y_true
    mape = perc_error_abs.sum() / len(y_true)
    return mape
# функция, которая принимает на вход модель и данные и выводит метрики
def make_prediction(m, X_train, y_train, X_test, y_test):
    model = m
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    print(
        'MAE:{:.2f} MSE:{:.2f} MAPE:{:.2f} R2:{:.2f} '.format(
            mean_absolute_error(y_test, y_pred),
            mean_squared_error(y_test, y_pred),
```

```
            mape(y_test, y_pred),
            r2_score(y_test, y_pred),
        )
    )
# напишем цикл, который выводит метрики по списку моделей
for i in models:
    print(i)
    make_prediction(i, X_train, y_train, X_test, y_test)
```

```
LogisticRegression()
MAE:0.12 MSE:0.12 MAPE:inf R2:0.36
```

In [25]:

```
#Для случайного леса
models = [
    RandomForestClassifier()

]
rf_model = RandomForestClassifier(n_estimators = 100,random_state = 0) # Ваш код здесь
# обучим модель случайного леса
rf_model.fit(X_train, y_train)

# воспользуемся уже обученной моделью, чтобы сделать прогнозы
rf_predictions = rf_model.predict(X_test)# Ваш код здесь
rf_probabilities = rf_model.predict_proba(X_test)[:, 1]# Ваш код здесь
# выведем все метрики
for i in models:
    print(i)
    make_prediction(i, X_train, y_train, X_test, y_test)
```

```
RandomForestClassifier()
MAE:0.10 MSE:0.10 MAPE:inf R2:0.47
```

Оценим метрики accuracy, precision и recall для обеих моделей на валидационной выборке

In [26]:

```
# зададим алгоритм для нашей модели логистической регрессии
model = LogisticRegression(random_state=0)
# обучим модель
model.fit(X_train_st,y_train)
#воспользуемся уже обученной моделью, чтобы сделать прогнозы
predictions = model.predict(X_test_st)
probabilities = model.predict_proba(X_test_st)[:,1]
# выведем значения predictions и probabilities на экран
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, predictions)))
print('Precision: {:.2f}'.format(precision_score (y_test, predictions)))
print('Recall: {:.2f}'.format(recall_score (y_test, predictions)))
```

```
Accuracy: 0.90
Precision: 0.79
Recall: 0.82
```

In [27]:

```
# зададим алгоритм для новой модели на основе алгоритма случайного леса
rf_model = RandomForestClassifier(n_estimators = 100, random_state = 0) # Ваш код здесь
rf_model.fit(X_train_st, y_train)# обучим модель случайного леса
# воспользуемся уже обученной моделью, чтобы сделать прогнозы
rf_predictions = rf_model.predict(X_test_st)
rf_probabilities =rf_model.predict_proba(X_test_st)[:, 1]
# выведем значения predictions и probabilities на экран
print('Accuracy: {:.2f}'.format(accuracy_score(y_test, rf_predictions)))
print('Precision: {:.2f}'.format(precision_score (y_test, rf_predictions)))
print('Recall: {:.2f}'.format(recall_score (y_test, rf_predictions)))
```

```
Accuracy: 0.90
Precision: 0.81
Recall: 0.75
```

- Вывод:
  - у моделей - логистической регрессии и случайного леса разница между precision и recall довольна ощутимая.

Обучим финальные модели и получим метрики классификации на основе значений прогнозного класса

In [28]:
```
# обучим финальную модель
final_model = RandomForestRegressor( random_state = 0)
final_model.fit(X_train, y_train)
y_pred = final_model.predict(X_test)
```

In [29]:
```
# создадим датафрейм с именами признаков и их важностью и выведем его по убыванию важности
fi_df = pd.DataFrame(data={'feature': X.columns, 'importance': final_model.feature_importances_
fi_df.sort_values('importance', ascending=False)
```

Out[29]:

	feature	importance
9	lifetime	0.376948
10	a_month	0.205809
5	period	0.121373
8	a_charges	0.121324
7	age	0.107472
6	group_visits	0.014927
3	promo_friends	0.012737
0	gender	0.012571
2	partner	0.010339
1	near_location	0.009955
4	phone	0.006545

In [30]:
```
# обучим финальную модель
final_model = LogisticRegression( random_state = 0)
final_model.fit(X_train, y_train)
y_pred = final_model.predict(X_test)
```

In [31]:
```
# создадим датафрейм с именами признаков и их важностью и выведем его по убыванию важности
fi_df = pd.DataFrame(data={'feature': X.columns, 'coeff': final_model.coef_[0]})
fi_df.sort_values('coeff', ascending=False)
```

Out[31]:

	feature	coeff
4	phone	1.398690
0	gender	0.332506
1	near_location	0.106621
2	partner	0.076518
8	a_charges	-0.007576
7	age	-0.119972
5	period	-0.202932
6	group_visits	-0.594174

	feature	coeff
3	promo_friends	-1.029141
9	lifetime	-1.246773
10	a_month	-1.436661

Вывод:

- исходя из анализа данных датафреймов, выбираем модель "Случайный лес", т.к. по важности признаков он более отвечает исследованию.

Сделаем кластеризацию клиентов

Стандартизируем данные

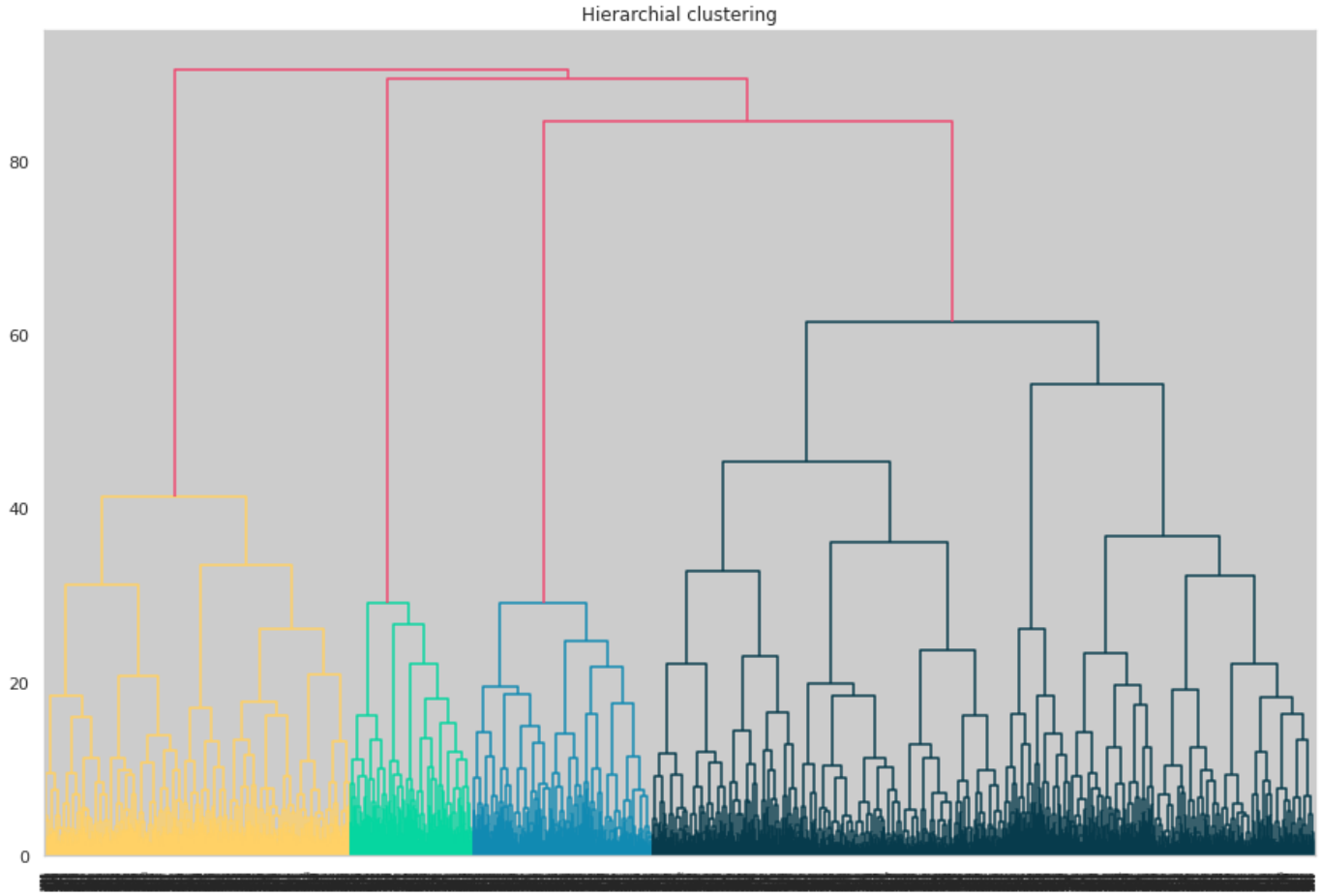
In [32]:
```
#Удалим столбец
df1=df1.drop(['churn'],axis = 1)
```

In [33]:
```
# создаём объект класса scaler (нормализатор)
scaler = StandardScaler()
x_sc = scaler.fit_transform(df1)
print(x_sc[:10])
```

```
[[ 0.97970588  0.42788074  1.02686062  1.49716101 -3.05985201  0.28989014
   1.19403206 -0.0565538  -1.37753121 -0.19332863 -1.67847198]
 [-1.0207145   0.42788074 -0.973842   -0.66793083  0.32681319  1.60882159
   1.19403206  0.55732732 -0.35021325  0.87368001  0.1360137 ]
 [-1.0207145   0.42788074  1.02686062 -0.66793083  0.32681319 -0.8092194
  -0.83749845 -0.36349436 -0.1815923  -0.46008079 -0.02901851]
 [-1.0207145   0.42788074  1.02686062  1.49716101  0.32681319  1.60882159
   1.19403206  1.17120844 -0.87472237 -0.46008079  1.51045005]
 [ 0.97970588  0.42788074  1.02686062  1.49716101  0.32681319 -0.8092194
  -0.83749845 -0.97737548  0.5336998  -0.19332863 -0.61454183]
 [ 0.97970588  0.42788074 -0.973842   -0.66793083  0.32681319 -0.8092194
   1.19403206  1.478149    3.15330097 -0.19332863  0.82634551]
 [ 0.97970588  0.42788074  1.02686062  1.49716101 -3.05985201  0.28989014
   1.19403206  0.86426788 -0.54676556 -0.46008079 -0.46525669]
 [-1.0207145   0.42788074 -0.973842   -0.66793083  0.32681319 -0.8092194
  -0.83749845  0.25038676  0.73531552 -0.99358511 -0.68168915]
 [ 0.97970588  0.42788074  1.02686062  1.49716101  0.32681319 -0.8092194
   1.19403206 -1.89819716 -1.05387243 -0.72683295 -0.67224189]
 [-1.0207145   0.42788074 -0.973842   -0.66793083  0.32681319 -0.8092194
  -0.83749845  0.55732732 -0.84731033  1.94068865  0.17507634]]
```

Построим матрицу расстояний и нарисуем дендрограмму

In [34]:
```
# Построим матрицу расстояний функцией linkage()
linked = linkage(x_sc, method = 'ward')
# Дендрограмма
plt.figure(figsize=(15, 10))
dendrogram(linked, orientation='top')
plt.title('Hierarchial clustering')
plt.show()
```



Предложенное оптимальное число кластеров 4 — четыре разные цвета на графике

Обучим модель кластеризации на основании алгоритма K-Means и спрогнозируем кластеры клиентов

In [35]:
```
# задаём модель k_means с числом кластеров 5
km = KMeans(n_clusters = 5, random_state=0)
# прогнозируем кластеры для наблюдений
labels = km.fit_predict(x_sc)
df['cluster_km'] = labels
```

Посчитаем кластеры

In [36]:
```
# Отсортируем кластеры по убыванию.
df.cluster_km.value_counts()
```

Out[36]:
```
2    1064
3    1007
0     985
1     558
4     386
Name: cluster_km, dtype: int64
```

- Вывод:
  - самый многочисленный 2 кластер - 1064 чел
  - самый маленький 4 - 386 чел

Посчитаем метрику силуэта для нашей кластеризации

In [37]:
```
# посчитаем метрику силуэта для нашей кластеризации
print('Silhouette_score: {:.2f}'.format(silhouette_score(x_sc, labels)))
```

```
Silhouette_score: 0.14
```

Значение метрики силуэта принимает значения от -1 до 1. Чем ближе к 1, тем качественнее кластеризация.

Средние значения признаков для кластеров

In [38]:
```
# Средние значения признаков для кластеров
df.groupby('cluster_km').mean().T
```

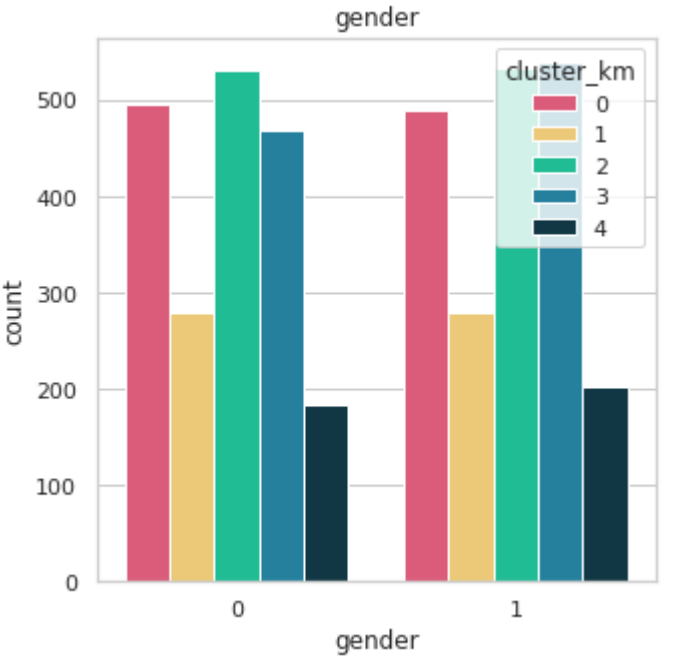
Out[38]:

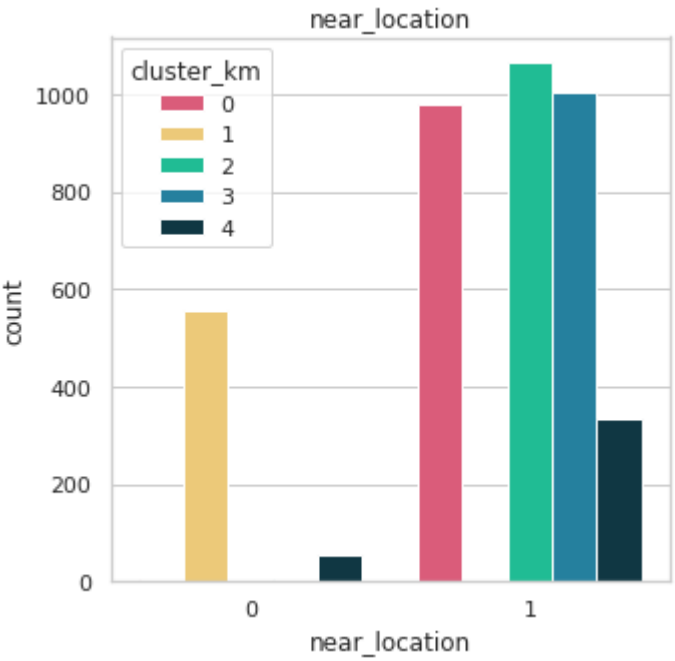
cluster_km	0	1	2	3	4
gender	0.496447	0.500000	0.500940	0.534260	0.523316
near_location	0.995939	0.000000	1.000000	0.996028	0.862694
partner	0.892386	0.489247	0.217105	0.379345	0.471503
promo_friends	1.000000	0.078853	0.072368	0.009930	0.305699
phone	1.000000	1.000000	1.000000	1.000000	0.000000
period	6.922843	2.994624	2.010338	6.208540	4.777202
group_visits	0.524873	0.232975	0.277256	0.538232	0.427461
age	29.606091	28.679211	27.583647	30.699106	29.297927
a_charges	153.424651	137.125763	119.339956	176.259567	144.208179
month_end	6.332995	2.818996	1.941729	5.650447	4.466321
lifetime	4.283249	2.974910	1.922932	5.415094	3.940415
a_total	1.962217	1.764122	1.451098	2.322960	1.854211
a_month	1.919520	1.597146	1.203319	2.324220	1.723967
churn	0.119797	0.403226	0.563910	0.014896	0.266839

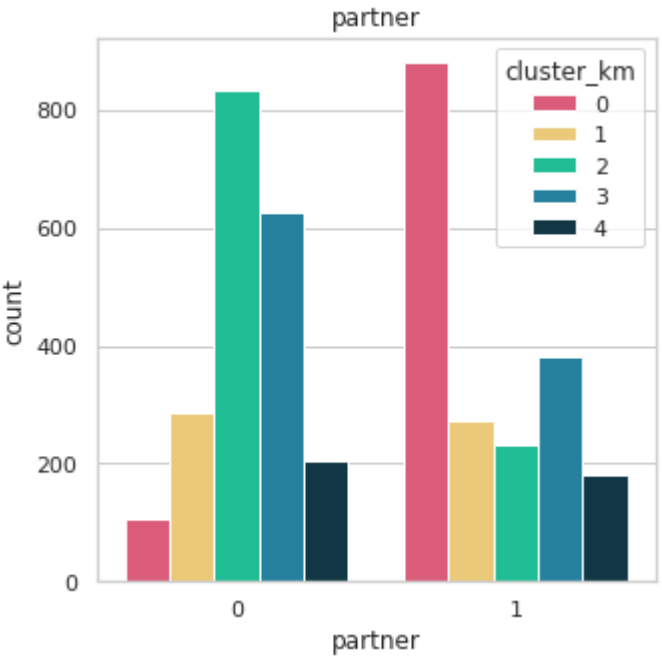
- Выводы из средних значений признаков для кластеров:
  - по полу кластеры почти не отличаются,в 0 доля мужчин минимальна,в 3 их больше
  - во 2 кластере живущих рядом 100%, в 1 все приезжие
  - в 0 больше из фирм- партнеров - более 89%, во 2 минимальна - около 22%
  - в 0 кластере больше пришедших по акции "друзья" - 100%, в 3 - всего около 1%
  - самая большая длительность текущего действующего абонемента в 0 кластере - почти 7 мес.
  - самая маленькая длительность текущего действующего абонемента во 2 - 2 мес
  - меньше всего групповых занятий в 1 кластере - 23%
  - больше всего групповых занятий в 3 кластере - более 53%
  - по возрасту кластеры почти не отличаются, во 2 чуть меньше, в 3 чуть больше
  - меньше всех покупок совершают во 2 кластере, больше - в 3
  - срок до окончания текущего действующего абонемента самый большой в 0 кластере - более 6 мес, самый маленький - во 2 кластере - менее 2 мес
  - в кластере 2 собрались новички - время с момента первого обращения в фитнес-центр менее 2 мес
  - в кластере 3 старожилы - время с момента первого обращения в фитнес-центр более 5 мес
  - в кластере 3 самые мотивированные - здесь значительно больше средняя частота посещений в неделю
  - в кластере 2 минимальное посещение занятий - 1.2 в неделю
  - отток в текущем месяце больше во 2 кластере более 56%, минимальный в 3 - менее 1.5%

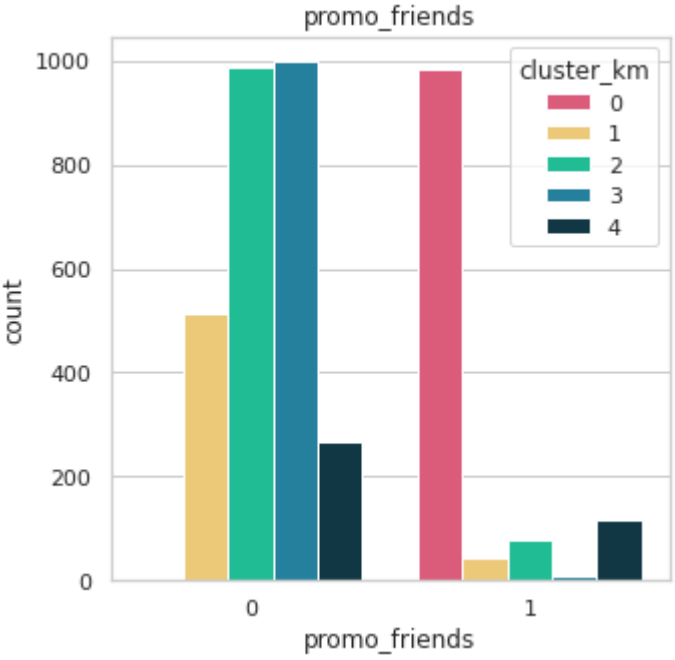
Построим диаграммы для признаков кластеров

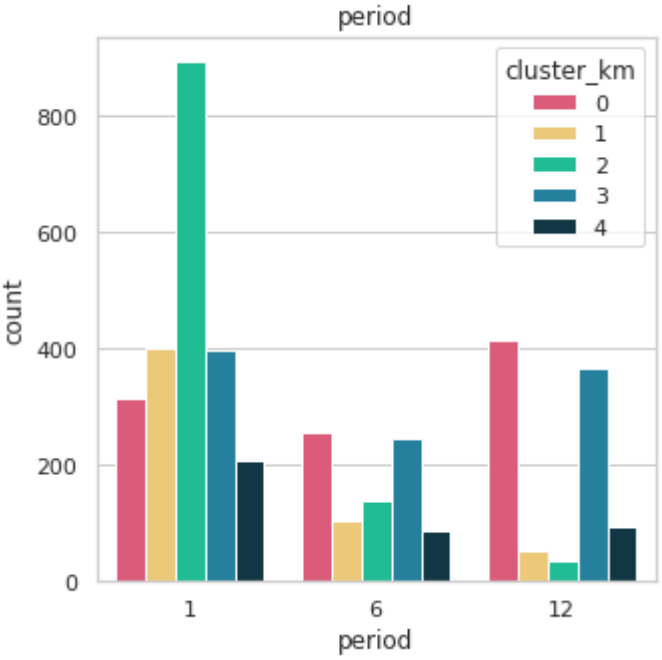
In [39]:
```
for column in ['gender', 'near_location', 'partner', 'promo_friends', 'period','month_end']:
    plt.figure(figsize=(5, 5))
    sns.countplot(x = df[column], hue='cluster_km', data=df)
    plt.title(column)
    plt.show()
```

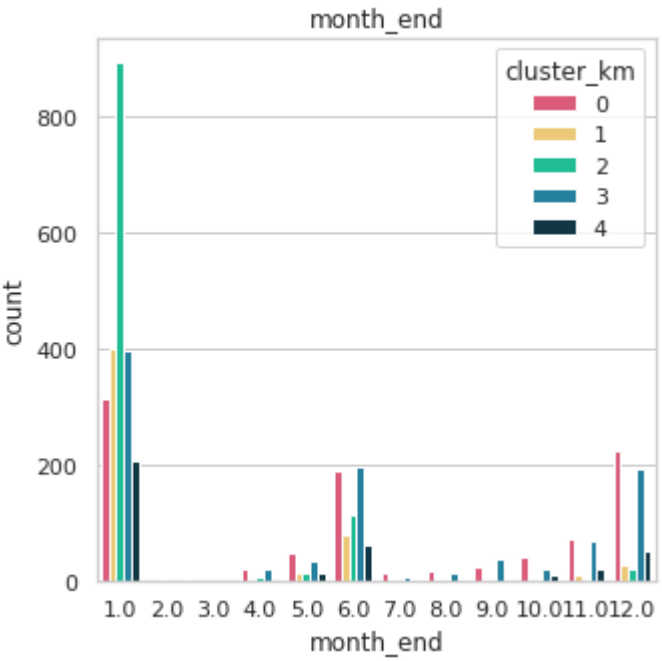





partner
cluster_km
0
1
2
3
4
800
600
400
200
0
count
0
1
partner

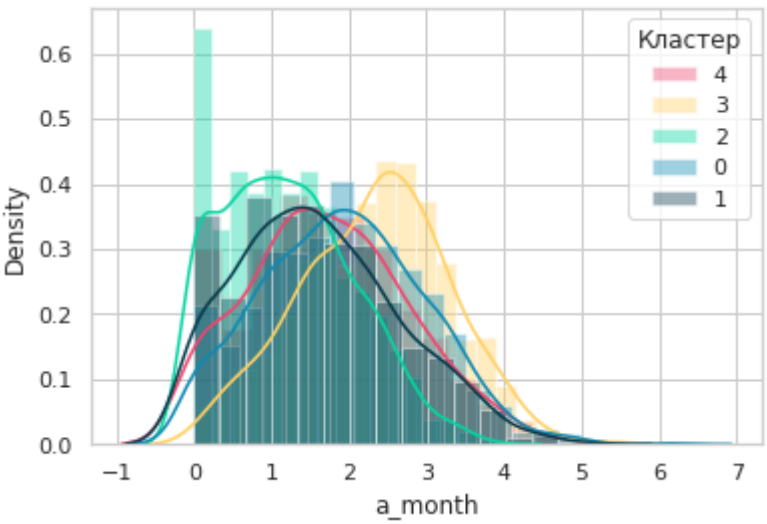

promo_friends
cluster_km
0
1
2
3
4
1000
800
600
400
200
0
count
0
1
promo_friends


period
cluster_km
0
1
2
3
4
800
600
400
200
0
count
1
6
12
period

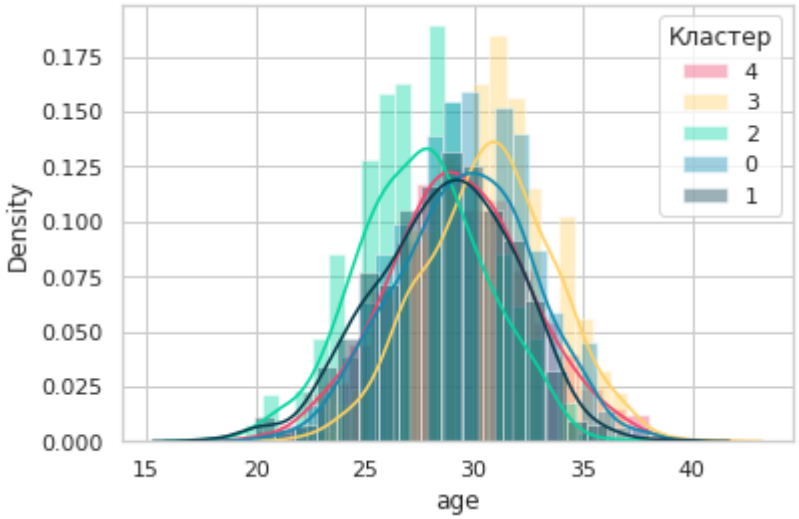


- Выводы:
  - по полу кластеры почти не различаются. в 3 меньше женщин
  - клиенты 1 кластера преимущественно приезжие, 0,2и3 - местные
  - меньше всего от фирм- партнеров во 2 и 3 кластере, в 1 их большинство
  - меньше всего по акции пришли клиенты 1,2 и 3 кластеров, в 0 их большинство
  - длительность текущего действующего абонемента:
    - самая короткая во 2 кластере
    - самая большая в 0 и 3 кластерах
  - месячных абонементов больше во 2 и 1 кластере, годовых в 0 и 3

In [40]:
```
for cluster in df['cluster_km'].unique():
    sns.distplot(df.query('cluster_km ==@cluster')['a_month'],label = cluster)
    plt.legend(title = 'Кластер')
plt.show()
```

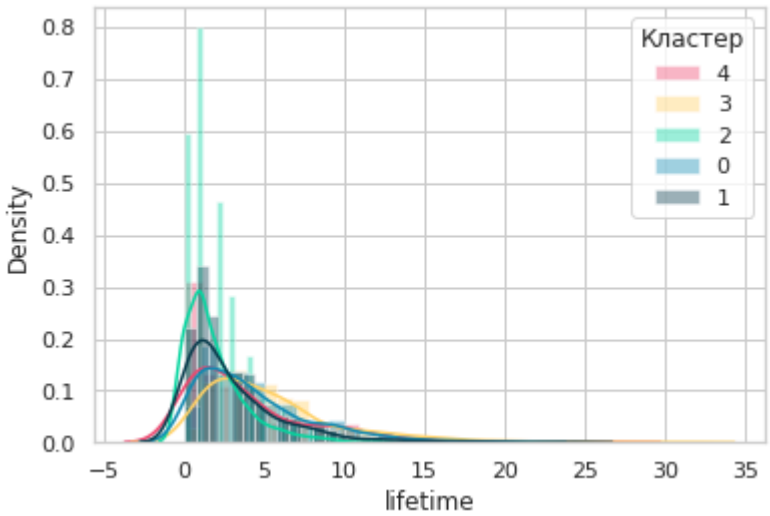


In [41]:
```
for cluster in df['cluster_km'].unique():
    sns.distplot(df.query('cluster_km ==@cluster')['age'],label = cluster)
    plt.legend(title = 'Кластер')
plt.show()
```



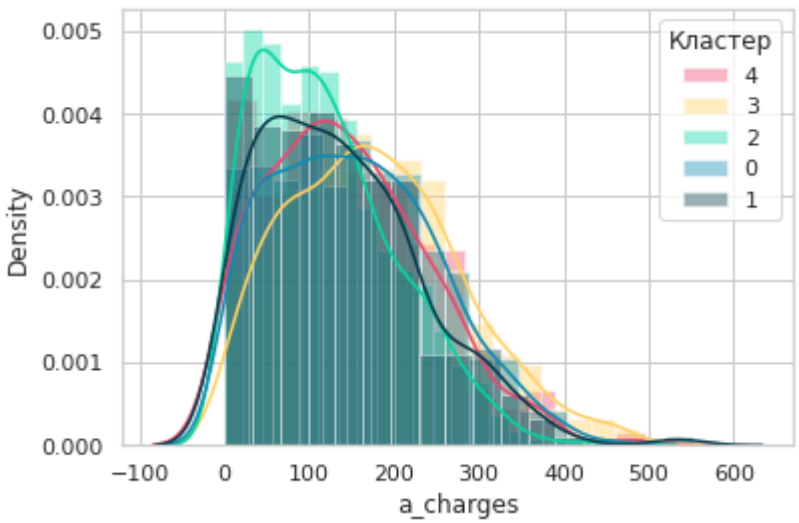
In [42]:

```
for cluster in df['cluster_km'].unique():
    sns.distplot(df.query('cluster_km ==@cluster')['lifetime'],label = cluster)
    plt.legend(title = 'Кластер')
plt.show()
```



In [43]:

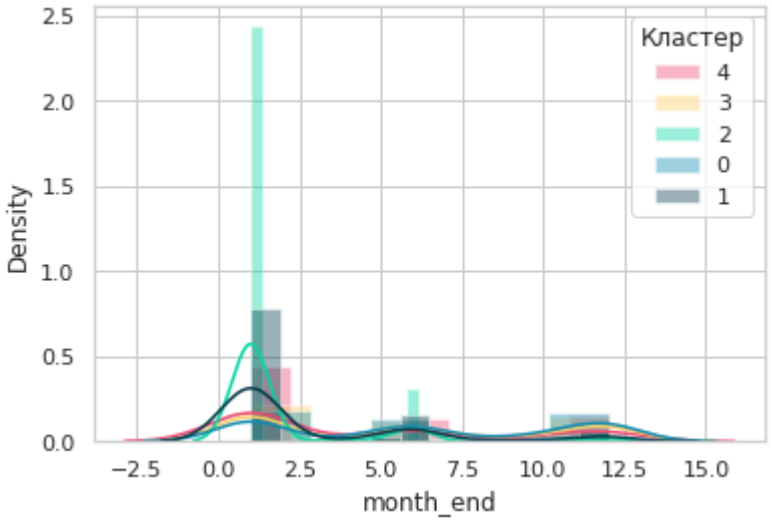
```
for cluster in df['cluster_km'].unique():
    sns.distplot(df.query('cluster_km ==@cluster')['a_charges'],label = cluster)
    plt.legend(title = 'Кластер')
plt.show()
```



In [44]:

```
for cluster in df['cluster_km'].unique():
    sns.distplot(df.query('cluster_km ==@cluster')['month_end'],label = cluster)
```

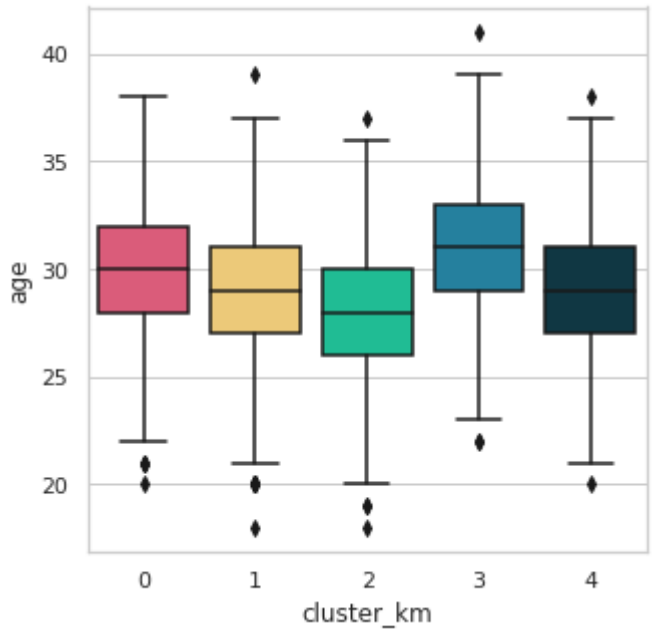
```
    plt.legend(title = 'Кластер')
plt.show()
```

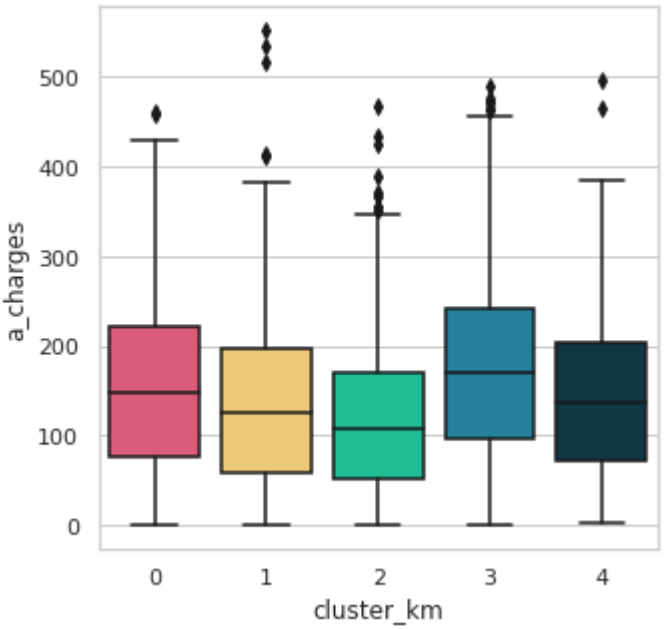


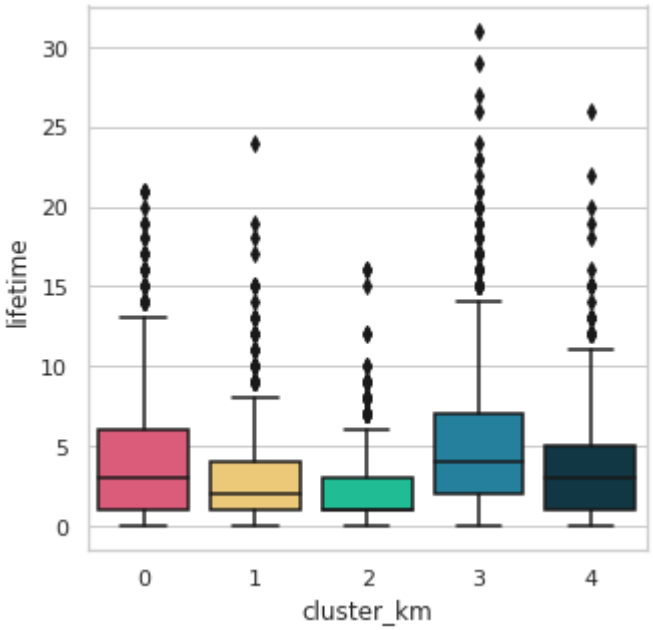
- Вывод:
  - в 3 кластере больше посещений занятий в неделю, во 2 - минимально
  - самые возрастные в 3, самые молодые во 2
  - во 2 кластере минимальное время с момента первого обращения в фитнес-центр
  - количество покупок больше в 3 кластере, но сумма меньше, во 2 наоборот- покупок больше, сумма меньше
  - во 2 меньше срок до окончания текущего действующего абонемента

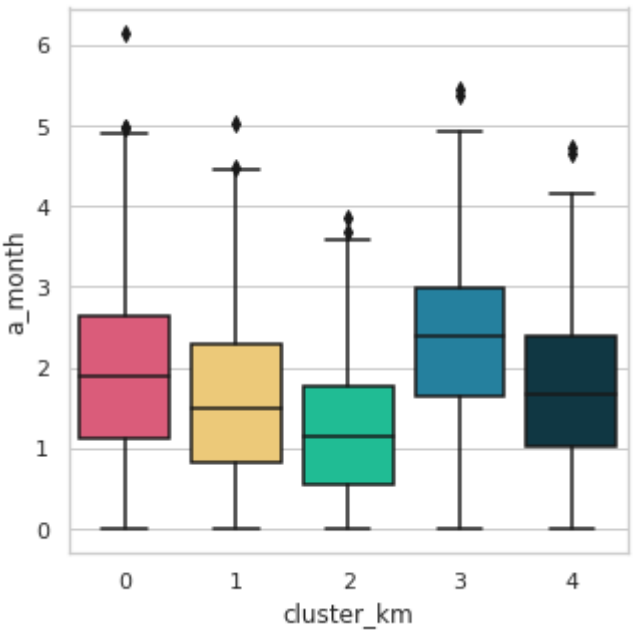
Построим ящики с усами

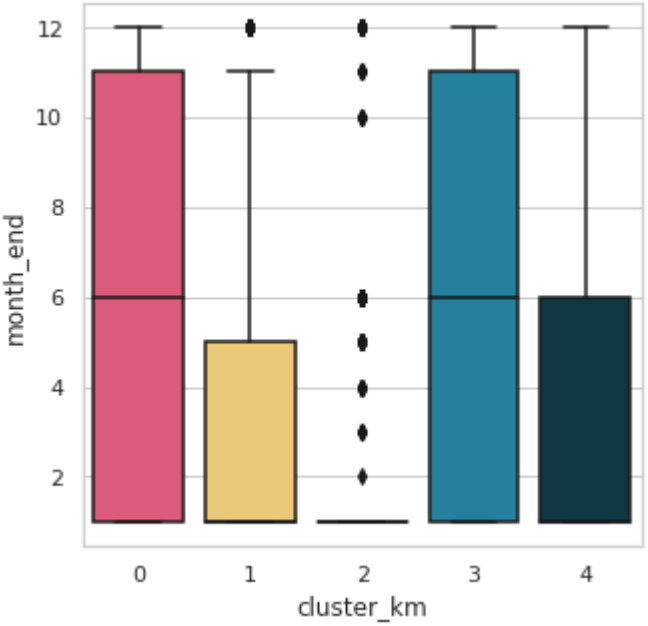
In [45]:
```
for column in ['age', 'a_charges', 'lifetime', 'a_month','month_end' ]:
    plt.figure(figsize=(5, 5))
    sns.boxplot(x = 'cluster_km', y = df[column], data = df)
    plt.show()
```




a_charges
0
100
200
300
400
500
0
1
2
3
4
cluster_km


lifetime
0
5
10
15
20
25
30
0
1
2
3
4
cluster_km


a_month
0
1
2
3
4
5
6
0
1
2
3
4
cluster_km



- Выводы:
  - средний медианный возраст ниже у клиентов во 2 кластере
  - они же совершают меньше покупок
  - время с момента первого обращения в фитнес-центр меньше у 2 кластера
  - больше всего занятий в неделю у 1 кластера, меньше у 2
  - срок до окончания текущего действующего абонемента больше в 0 и 3 кластерах

Посчитаем долю оттока для каждого кластера

In [46]:
```
# Для каждого полученного кластера посчитайте долю оттока
df['cluster_km'] = labels
df.groupby('cluster_km').agg({'churn':'mean'})
```

Out[46]:

cluster_km	churn
0	0.119797
1	0.403226
2	0.563910
3	0.014896
4	0.266839

- Вывод:
  - самый верный 3 кластер- минимальная доля оттока менее 1.5%
  - самый ненадежный - 2 кластер- более 56%

Выводы и базовые рекомендации по работе с клиентами

Выводы

- Надежные кластеры 0 и 3
- Самые лучшие показатели в 0 кластере:
  - жители района
  - сотрудники компаний-партнеров

    - занимаются с друзьями
    - большая длительность действующего абонемента
    - групповые занятия
    - совершают много покупок
    - большое время с момента первого обращения в фитнес-центр
    - среднее число занятий в неделю
    - небольшой отток.
- Самый ненадежный - 2:
    - хуже по всем показателям.

Рекомендации по работе с клиентами

- Нам необходимо:
    - Поддерживать и мотивировать "хорошие кластеры"
    - Глубже сегментировать оттекающих, к каждому сегменту тестировать индивидуальные предложения
    - Выстроить предиктивную систему аналитики и действовать не после оттока, а до него.
    - Привлечение новых клиентов по акции "приведи друга"
    - Активная реклама клуба в районе его расположения
    - Повышение вовлеченности на групповых занятиях (например внедрение элементов геймификации), поощрение их посещения
    - Расширение программы привлечения сотрудников компаний-партнеров и списка самих партнеров