# Тема 1. Первичный анализ данных с Pandas

Домашние задания в курсе

Демонстрация основных методов Pandas

Первые попытки прогнозирования оттока

Домашнее задание №1

Необходимо навёрстывать:

* базовую математику (математический анализ, линейную алгебру, оптимизацию, теорвер и статистику) ;
* по Python хватит небольшого интерактивного туториала на любой образовательной платформе;
* что касается машинного обучения, на русском языке есть отличная специализация МФТИ и Яндекса «Машинное обучение и анализ данных». Есть книга и на русском языке – "Глубокое обучение: погружение в мир нейронных сетей" (Николенко С. И., Кадурин А. А., Архангельская Е. О.).

import pandas as pd

import numpy as np

Прочитаем данные (метод read\_csv) и посмотрим на первые 5 строк с помощью метода head:

df = pd.read\_csv('telecom\_churn.csv')

df.head()

print(df.head())

По умолчанию Pandas выводит всего 20 столбцов и 60 строк, поэтому если ваш датафрейм больше, воспользуйтесь функцией set\_option:

Каждая строка представляет собой одного клиента – это объект исследования.  
Столбцы – признаки объекта.

| Название | Описание | Тип |
| --- | --- | --- |
| State | Буквенный код штата | номинальный |
| Account length | Как долго клиент обслуживается компанией | количественный |
| Area code | Префикс номера телефона | количественный |
| International plan | Международный роуминг (подключен/не подключен) | бинарный |
| Voice mail plan | Голосовая почта (подключена/не подключена) | бинарный |
| Number vmail messages | Количество голосовых сообщений | количественный |
| Total day minutes | Общая длительность разговоров днем | количественный |
| Total day calls | Общее количество звонков днем | количественный |
| Total day charge | Общая сумма оплаты за услуги днем | количественный |
| Total eve minutes | Общая длительность разговоров вечером | количественный |
| Total eve calls | Общее количество звонков вечером | количественный |
| Total eve charge | Общая сумма оплаты за услуги вечером | количественный |
| Total night minutes | Общая длительность разговоров ночью | количественный |
| Total night calls | Общее количество звонков ночью | количественный |
| Total night charge | Общая сумма оплаты за услуги ночью | количественный |
| Total intl minutes | Общая длительность международных разговоров | количественный |
| Total intl calls | Общее количество международных разговоров | количественный |
| Total intl charge | Общая сумма оплаты за международные разговоры | количественный |
| Customer service calls | Число обращений в сервисный центр | количественный |

Целевая переменная: Churn – Признак оттока, бинарный признак (1 – потеря клиента, то есть отток). Потом мы будем строить модели, прогнозирующие этот признак по остальным, поэтому мы и назвали его целевым.

Посмотрим на размер данных, названия признаков и их типы.

print(df.shape)

(3333, 20)

Видим, что в таблице 3333 строки и 20 столбцов. Выведем названия столбцов:

print(df.columns)

Index(['State', 'Account length', 'Area code', 'International plan',

'Voice mail plan', 'Number vmail messages', 'Total day minutes',

'Total day calls', 'Total day charge', 'Total eve minutes',

'Total eve calls', 'Total eve charge', 'Total night minutes',

'Total night calls', 'Total night charge', 'Total intl minutes',

'Total intl calls', 'Total intl charge', 'Customer service calls',

'Churn'],

dtype='object')

Чтобы посмотреть общую информацию по датафрейму и всем признакам, воспользуемся методом info:

print(df.info())

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

RangeIndex: 3333 entries, 0 to 3332

Data columns (total 20 columns):

State 3333 non-null object

Account length 3333 non-null int64

Area code 3333 non-null int64

International plan 3333 non-null object

Voice mail plan 3333 non-null object

Number vmail messages 3333 non-null int64

Total day minutes 3333 non-null float64

Total day calls 3333 non-null int64

Total day charge 3333 non-null float64

Total eve minutes 3333 non-null float64

Total eve calls 3333 non-null int64

Total eve charge 3333 non-null float64

Total night minutes 3333 non-null float64

Total night calls 3333 non-null int64

Total night charge 3333 non-null float64

Total intl minutes 3333 non-null float64

Total intl calls 3333 non-null int64

Total intl charge 3333 non-null float64

Customer service calls 3333 non-null int64

Churn 3333 non-null bool

dtypes: bool(1), float64(8), int64(8), object(3)

memory usage: 498.1+ KB

None

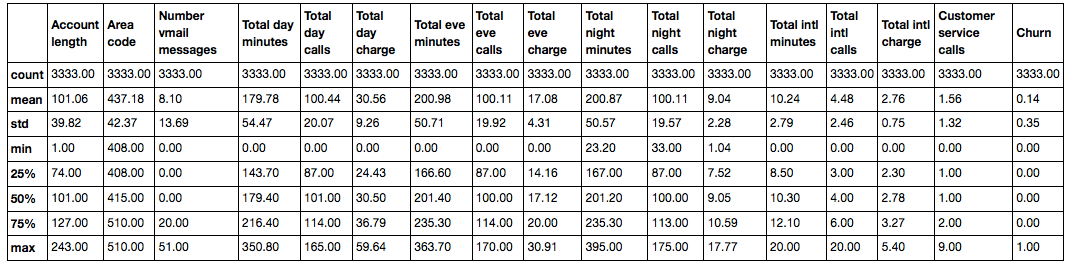
bool, int64, float64 и object — это типы признаков. Видим, что 1 признак — логический (bool), 3 признака имеют тип object и 16 признаков — числовые. Также с помощью метода info удобно быстро посмотреть на пропуски в данных, в нашем случае их нет, в каждом столбце по 3333 наблюдения.

Изменить тип колонки можно с помощью метода astype. Применим этот метод к признаку Churn и переведём его в int64:

df['Churn'] = df['Churn'].astype('int64')

Метод describe показывает основные статистические характеристики данных по каждому числовому признаку (типы int64 и float64): число непропущенных значений, среднее, стандартное отклонение, диапазон, медиану, 0.25 и 0.75 квартили.

df.describe()

Чтобы посмотреть статистику по нечисловым признакам, нужно явно указать интересующие нас типы в параметре include.

df.describe(include=['object', 'bool'])

|  | State | International plan | Voice mail plan |
| --- | --- | --- | --- |
| count | 3333 | 3333 | 3333 |
| unique | 51 | 2 | 2 |
| top | WV | No | No |
| freq | 106 | 3010 | 2411 |

Для категориальных (тип object) и булевых (тип bool) признаков можно воспользоваться методом value\_counts. Посмотрим на распределение данных по нашей целевой переменной — Churn:

df['Churn'].value\_counts()

0 2850

1 483

Name: Churn, dtype: int64

2850 пользователей из 3333 — лояльные, значение переменной Churn у них — 0.

Посмотрим на распределение пользователей по переменной Area code. Укажем значение параметра normalize=True, чтобы посмотреть не абсолютные частоты, а относительные.

df['Area code'].value\_counts(normalize=True)

415 0.496550

510 0.252025

408 0.251425

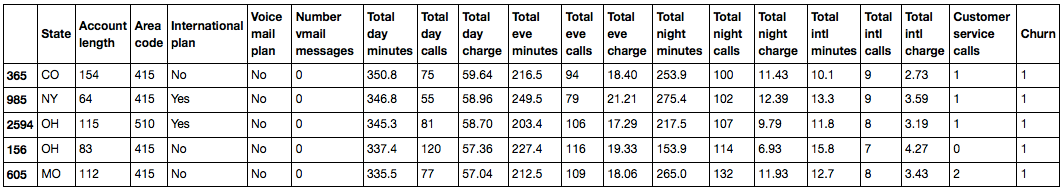
Name: Area code, dtype: float64

Сортировка

DataFrame можно отсортировать по значению какого-нибудь из признаков. В нашем случае, например, по Total day charge (ascending=False для сортировки по убыванию):

df.sort\_values(by='Total day charge',

ascending=False).head()



### Индексация и извлечение данных

DataFrame можно индексировать по-разному. В связи с этим рассмотрим различные способы индексации и извлечения нужных нам данных из датафрейма на примере простых вопросов.

Для извлечения отдельного столбца можно использовать конструкцию вида DataFrame['Name']. Воспользуемся этим для ответа на вопрос: какова доля людей нелояльных пользователей в нашем датафрейме?

df['Churn'].mean(). # выводит: 0.14491449144914492

14,5% — довольно плохой показатель для компании, с таким процентом оттока можно и разориться.

Очень удобной является логическая индексация DataFrame по одному столбцу. Выглядит она следующим образом: df[P(df['Name'])], где P — это некоторое логическое условие, проверяемое для каждого элемента столбца Name. Итогом такой индексации является DataFrame, состоящий только из строк, удовлетворяющих условию P по столбцу Name.

Воспользуемся этим для ответа на вопрос: каковы средние значения числовых признаков среди нелояльных пользователей?

# надо откорректировать выдавать среднее по всему df при лояльности df['Churn'] == 1 иначе выдает ошибку при команде ниже

# df[df['Churn'] == 1].mean()

Account length 102.664596

Number vmail messages 5.115942

Total day minutes 206.914079

Total day calls 101.335404

Total day charge 35.175921

Total eve minutes 212.410145

Total eve calls 100.561077

Total eve charge 18.054969

Total night minutes 205.231677

Total night calls 100.399586

Total night charge 9.235528

Total intl minutes 10.700000

Total intl calls 4.163561

Total intl charge 2.889545

Customer service calls 2.229814

Churn 1.000000

dtype: float64

Скомбинировав предыдущие два вида индексации, ответим на вопрос: сколько в среднем в течение дня разговаривают по телефону нелояльные пользователи?

df[df['Churn'] == 1]['Total day minutes'].mean() # выводит: 206.91407867494823

Какова максимальная длина международных звонков среди лояльных пользователей (Churn == 0), не пользующихся услугой международного роуминга ('International plan' == 'No')?

df[(df['Churn'] == 0) & (df['International plan'] == 'No')]['Total intl minutes'].max() # выводит: 18.899999999999999

Датафреймы можно индексировать как по названию столбца или строки, так и по порядковому номеру. Для индексации по названию используется метод loc, по номеру — iloc.

В первом случае мы говорим «передай нам значения для id строк от 0 до 5 и для столбцов от State до Area code», а во втором — «передай нам значения первых пяти строк в первых трёх столбцах».

df.loc[0:5, 'State':'Area code']

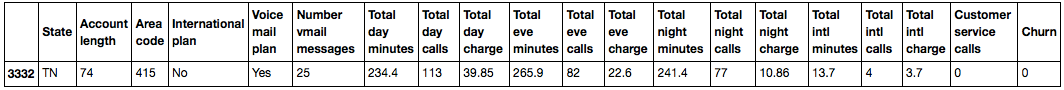
|  | State | Account length | Area code |
| --- | --- | --- | --- |
| 0 | KS | 128 | 415 |
| 1 | OH | 107 | 415 |
| 2 | NJ | 137 | 415 |
| 3 | OH | 84 | 408 |
| 4 | OK | 75 | 415 |
| 5 | AL | 118 | 510 |

df.iloc[0:5, 0:3]

|  | State | Account length | Area code |
| --- | --- | --- | --- |
| 0 | KS | 128 | 415 |
| 1 | OH | 107 | 415 |
| 2 | NJ | 137 | 415 |
| 3 | OH | 84 | 408 |
| 4 | OK | 75 | 415 |

Если нам нужна первая или последняя строчка датафрейма, пользуемся конструкцией df[:1] или df[-1:]:

df[-1:]



### Применение функций к ячейкам, столбцам и строкам

Применение функции к каждому столбцу: apply

df.apply(np.max)

State WY

Account length 243

Area code 510

International plan Yes

Voice mail plan Yes

Number vmail messages 51

Total day minutes 350.8

Total day calls 165

Total day charge 59.64

Total eve minutes 363.7

Total eve calls 170

Total eve charge 30.91

Total night minutes 395

Total night calls 175

Total night charge 17.77

Total intl minutes 20

Total intl calls 20

Total intl charge 5.4

Customer service calls 9

Churn True

dtype: object

Метод apply можно использовать и для того, чтобы применить функцию к каждой строке. Для этого нужно указать axis=1.

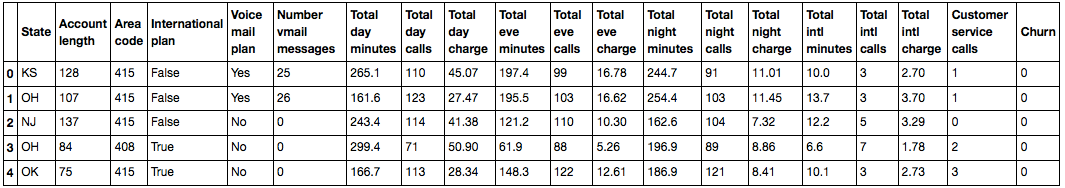
Применение функции к каждой ячейке столбца: map

Например, метод map можно использовать для замены значений в колонке, передав ему в качестве аргумента словарь вида {old\_value: new\_value}:

d = {'No' : False, 'Yes' : True}

df['International plan'] = df['International plan'].map(d)

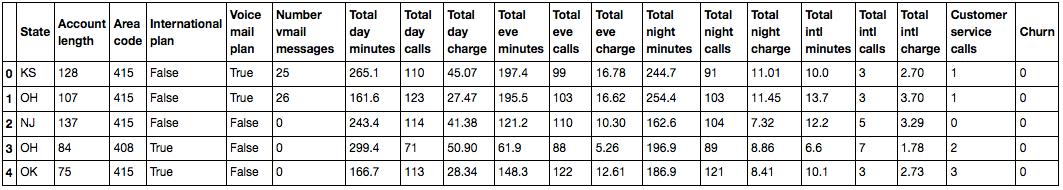
df.head()



Аналогичную операцию можно провернуть с помощью метода replace:

df = df.replace({'Voice mail plan': d})

df.head()



### Группировка данных

В общем случае группировка данных в Pandas выглядит следующим образом:

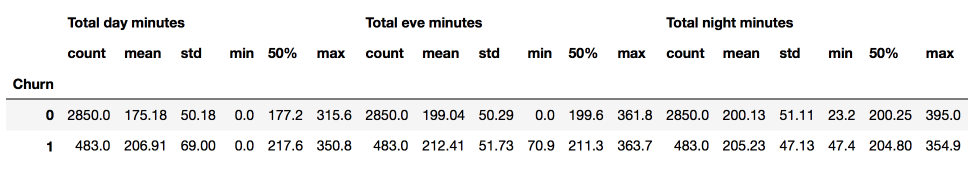
df.groupby(by=grouping\_columns)[columns\_to\_show].function()

1. К датафрейму применяется метод groupby, который разделяет данные по grouping\_columns – признаку или набору признаков.
2. Выбираем нужные нам столбцы (columns\_to\_show).
3. К полученным группам применяется функция или несколько функций.

Группирование данных в зависимости от значения признака Churn и вывод статистик по трём столбцам в каждой группе.

columns\_to\_show = ['Total day minutes', 'Total eve minutes', 'Total night minutes']

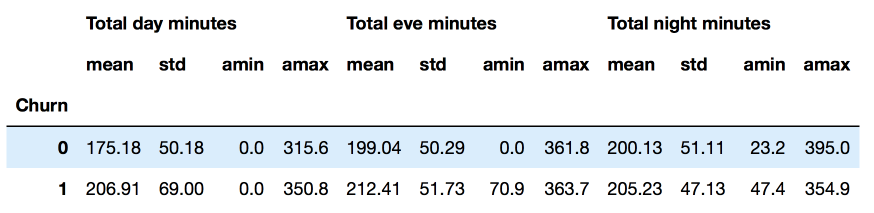
df.groupby(['Churn'])[columns\_to\_show].describe(percentiles=[])



Сделаем то же самое, но немного по-другому, передав в agg список функций:

columns\_to\_show = ['Total day minutes', 'Total eve minutes', 'Total night minutes']

df.groupby(['Churn'])[columns\_to\_show].agg([np.mean, np.std, np.min, np.max])



Допустим, мы хотим посмотреть, как наблюдения в нашей выборке распределены в контексте двух признаков — Churn и International plan. Для этого мы можем построить таблицу сопряженности, воспользовавшись методом crosstab:

pd.crosstab(df['Churn'], df['International plan'])

| International plan | No | Yes |
| --- | --- | --- |
| Churn |  |  |
| 0 | 2664 | 186 |
| 1 | 346 | 137 |

pd.crosstab(df['Churn'], df['Voice mail plan'], normalize=True)

| Voice mail plan | No | Yes |
| --- | --- | --- |
| Churn |  |  |
| 0 | 0.602460 | 0.252625 |
| 1 | 0.120912 | 0.024002 |

Мы видим, что большинство пользователей лояльны и при этом пользуются дополнительными услугами (международного роуминга / голосовой почты).

Продвинутые пользователи Excel наверняка вспомнят о такой фиче, как сводные таблицы (pivot tables). В Pandas за сводные таблицы отвечает метод pivot\_table, который принимает в качестве параметров:

* values – список переменных, по которым требуется рассчитать нужные статистики,
* index – список переменных, по которым нужно сгруппировать данные,
* aggfunc — то, что нам, собственно, нужно посчитать по группам — сумму, среднее, максимум, минимум или что-то ещё.

Давайте посмотрим среднее число дневных, вечерних и ночных звонков для разных Area code:

df.pivot\_table(['Total day calls', 'Total eve calls', 'Total night calls'],

['Area code'], aggfunc='mean').head(10)

|  | Total day calls | Total eve calls | Total night calls |
| --- | --- | --- | --- |
| Area code |  |  |  |
| 408 | 100.496420 | 99.788783 | 99.039379 |
| 415 | 100.576435 | 100.503927 | 100.398187 |
| 510 | 100.097619 | 99.671429 | 100.601190 |

### Преобразование датафреймов

Как и многое другое в Pandas, добавление столбцов в DataFrame осуществимо несколькими способами.

Например, мы хотим посчитать общее количество звонков для всех пользователей. Создадим объект total\_calls типа Series и вставим его в датафрейм:

total\_calls = df['Total day calls'] + df['Total eve calls'] + \

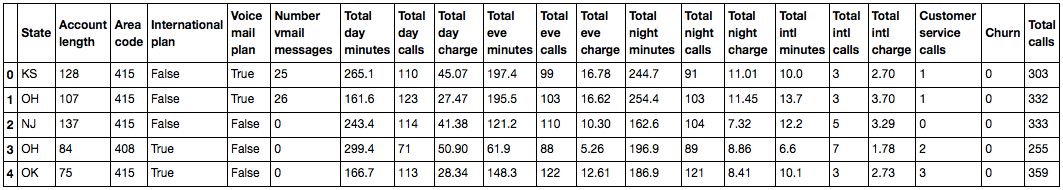
df['Total night calls'] + df['Total intl calls']

df.insert(loc=len(df.columns), column='Total calls', value=total\_calls)

# loc - номер столбца, после которого нужно вставить данный Series

# мы указали len(df.columns), чтобы вставить его в самом конце

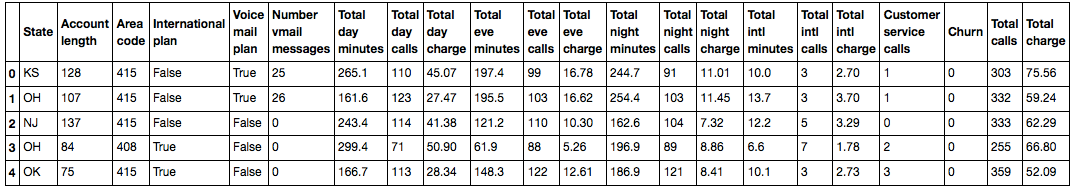
df.head()



Добавить столбец из имеющихся можно и проще, не создавая промежуточных Series:

df['Total charge'] = df['Total day charge'] + df['Total eve charge'] + df['Total night charge'] + df['Total intl charge']

df.head()

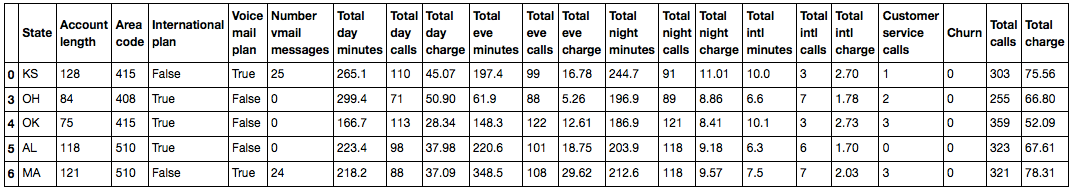


Чтобы удалить столбцы или строки, воспользуйтесь методом drop, передавая в качестве аргумента нужные индексы и требуемое значение параметра axis (1, если удаляете столбцы, и ничего или 0, если удаляете строки):

# избавляемся от созданных только что столбцов

df = df.drop(['Total charge', 'Total calls'], axis=1)

df.drop([1, 2]).head() # а вот так можно удалить строчки

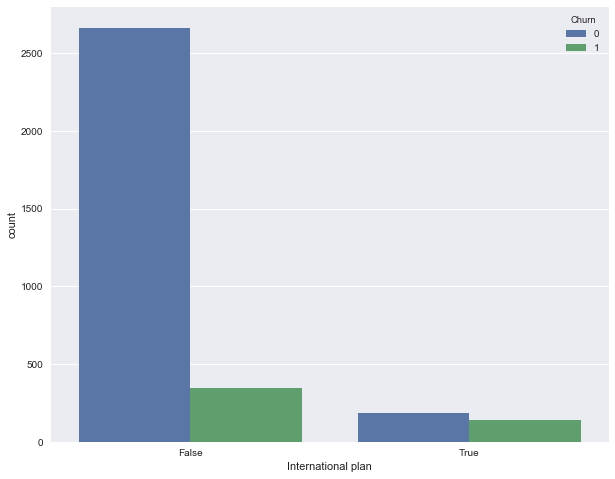


## 4. Первые попытки прогнозирования оттока

Посмотрим, как отток связан с признаком "Подключение международного роуминга" (International plan). Сделаем это с помощью сводной таблички crosstab, а также путем иллюстрации с Seaborn.

pd.crosstab(df['Churn'], df['International plan'], margins=True)

| International plan | False | True | All |
| --- | --- | --- | --- |
| Churn |  |  |  |
| 0 | 2664 | 186 | 2850 |
| 1 | 346 | 137 | 483 |
| All | 3010 | 323 | 3333 |

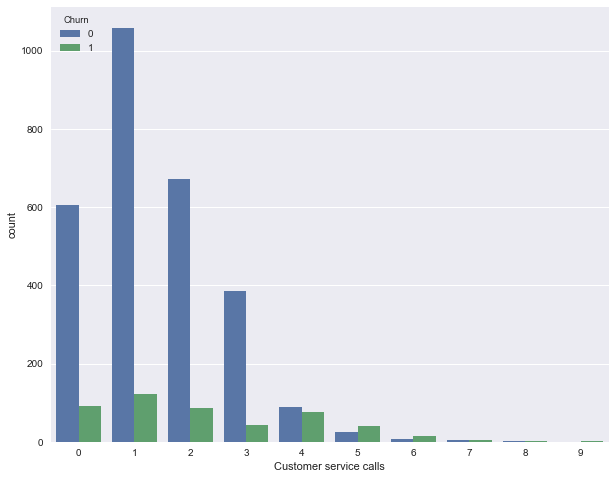


Видим, что когда роуминг подключен, доля оттока намного выше – интересное наблюдение! Возможно, большие и плохо контролируемые траты в роуминге очень конфликтогенны и приводят к недовольству клиентов телеком-оператора и, соответственно, к их оттоку.

Далее посмотрим на еще один важный признак – "Число обращений в сервисный центр" (Customer service calls). Также построим сводную таблицу и картинку.

pd.crosstab(df['Churn'], df['Customer service calls'], margins=True)

| Customer service calls | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | All |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Churn |  |  |  |  |  |  |  |  |  |  |  |
| 0 | 605 | 1059 | 672 | 385 | 90 | 26 | 8 | 4 | 1 | 0 | 2850 |
| 1 | 92 | 122 | 87 | 44 | 76 | 40 | 14 | 5 | 1 | 2 | 483 |
| All | 697 | 1181 | 759 | 429 | 166 | 66 | 22 | 9 | 2 | 2 | 3333 |

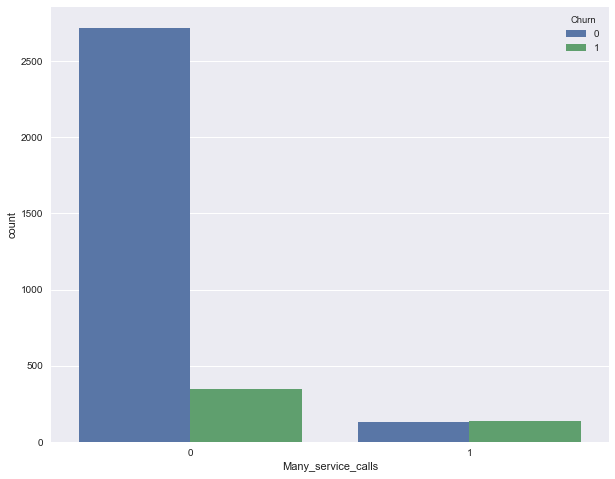
Может быть, по сводной табличке это не так хорошо видно (или скучно ползать взглядом по строчкам с цифрами), а вот картинка красноречиво свидетельствует о том, что доля оттока сильно возрастает начиная с 4 звонков в сервисный центр.

Добавим теперь в наш DataFrame бинарный признак — результат сравнения Customer service calls > 3. И еще раз посмотрим, как он связан с оттоком.

df['Many\_service\_calls'] = (df['Customer service calls'] > 3).astype('int')

pd.crosstab(df['Many\_service\_calls'], df['Churn'], margins=True)

| Churn | 0 | 1 | All |
| --- | --- | --- | --- |
| Many\_service\_calls |  |  |  |
| 0 | 2721 | 345 | 3066 |
| 1 | 129 | 138 | 267 |
| All | 2850 | 483 | 3333 |

Объединим рассмотренные выше условия и построим сводную табличку для этого объединения и оттока.

pd.crosstab(df['Many\_service\_calls'] & df['International plan'] , df['Churn'])

| Churn | 0 | 1 |
| --- | --- | --- |
| row\_0 |  |  |
| False | 2841 | 464 |
| True | 9 | 19 |

Значит, прогнозируя отток клиента в случае, когда число звонков в сервисный центр больше 3 и подключен роуминг (и прогнозируя лояльность – в противном случае), можно ожидать около 85.8% правильных попаданий (ошибаемся всего 464 + 9 раз). Эти 85.8%, которые мы получили с помощью очень простых рассуждений – это неплохая отправная точка (baseline) для дальнейших моделей машинного обучения, которые мы будем строить.

В целом до появления машинного обучения процесс анализа данных выглядел примерно так. Прорезюмируем:

* Доля лояльных клиентов в выборке – 85.5%. Самая наивная модель, ответ которой "клиент всегда лоялен" на подобных данных будет угадывать примерно в 85.5% случаев. То есть доли правильных ответов (accuracy) последующих моделей должны быть как минимум не меньше, а лучше, значительно выше этой цифры;
* С помощью простого прогноза, который условно можно выразить такой формулой: "International plan = True & Customer Service calls > 3 => Churn = 1, else Churn = 0", можно ожидать долю угадываний 85.8%, что еще чуть выше 85.5%. Впоследствии мы поговорим о деревьях решений и разберемся, как находить подобные правила автоматически на основе только входных данных;
* Эти два бейзлайна мы получили без всякого машинного обучения, и они служат отправной точной для наших последующих моделей. Если окажется, что мы громадными усилиями увеличиваем долю правильных ответов всего, скажем, на 0.5%, то возможно, мы что-то делаем не так, и достаточно ограничиться простой моделью из двух условий;
* Перед обучением сложных моделей рекомендуется немного покрутить данные и проверить простые предположения. Более того, в бизнес-приложениях машинного обучения чаще всего начинают именно с простых решений, а потом экспериментируют с их усложнениями.