Практические задания по первому модулю

1. Google Data Studio

Описание:

Компания «Product Gallery», которая занимается дистрибьюцией продуктов по всему миру. Ассортимент «Product Gallery» включает фрукты, овощи, молочные продукты, мясо и птицу, морепродукты, крупы, приправы и кондитерские изделия.

Владимир – владелец «Product Gallery» и фанат путешествий.

Владимир много времени проводит за изучением отчетов вместо того, чтобы активнее развивать бизнес и больше путешествовать.

Именно поэтому Владимир решил использовать систему класса Business Intelligence, чтобы быстро получать все необходимые данные для анализа и больше времени посвящать любимым делам.

Владимир проанализировал демо-примеры приложений для розничных компаний (Retail) на https://demos.qlik.com/qliksense?category=Retail, https://powerbi.microsoft.com/en-us/partner-showcase/agile-analytics-retail-insights/, https://www.tableau.com/solutions/topic/retail-wholesale и попросил разработать для него приложение в **Google Data Studio**.

Постоянно Владимир хочет быть в курсе:

1. Какие продажи у компании за произвольный период и с начала года?
2. Насколько прибыльна компания за произвольный период и с начала года?
3. Идет ли рост компании по сравнению с прошлыми периодами?
4. Выполняет ли компания планы по продажам и прибыли на конкретную дату и с начала года? (в %). Вы можете задать план самостоятельно, увеличив текущие показатели на 10% либо можете опустить данный пункт. **(необязательный).**

5. Какова средняя скидка по товарам на конкретную дату и с начала года?

Это данные «Product Gallery», которые предоставил Владимир: **https://docs.google.com/spreadsheets/d/1Y-AorVy9o7Go23KlAzCiFpD54pScIVjlrxgldv2lUNc/edit?usp=sharing**

Глоссарий по данным и формулам «Product Gallery»:

Поставщик = Supplier.

Стоимость единицы товара при закупке у поставщика (себестоимость) = UnitCost.

Цена единицы товара для клиента = Unit Price.

Количество = Quantity.

Дата заказа = Дата продажи = OrderDate.

Номер заказа = OrderID.

Скидка = Discount.

Количество единиц товара на складе = UnitsInStock.

Количество единиц товара в заказах от поставщика = UnitsOnOrder.

Продажи должны рассчитываться как в деньгах, так и в количестве проданных единиц товара.

Прибыль рассчитывается как разница между продажей и себестоимостью.

Маржа, % (в процентах) рассчитывается как отношение между прибылью и продажами.

Задание:

Разработайте аналитическое приложение, состоящее из нескольких дашбордов, для компании Владимира «Product Gallery» в Google Data Studio, по каждому дашборду необходимо сделать краткий вывод, “поделитесь” сделайте ссылку доступной всем для просмотра у кого она есть.

Критерии оценки приложения:

- Эргономика интерфейса (создание эффективного интерфейса).
- Логика интерфейса (логика расположения элементов в приложении для удобной аналитики).
- Качество визуализации (качество исполнения диаграмм, выбор типа диаграмм в соответствии с типом анализируемых данных)
- Качество дизайна приложения.
- Использование разработанных мер
- Полнота информации.
- Применение методов машинного обучения на уровне визуальных элементов
- Оценка предлагаемых вариантов управленческих решений и 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.

2. Демонстрация основных методов Pandas

Задание: Выполнить в Google Colaboratory (“поделиться” работой с преподавателем)

Введение:

Pandas — это библиотека Python, предоставляющая широкие возможности для анализа данных. Данные, с которыми работают датасаентисты, часто хранятся в форме табличек — например, в форматах .csv, .tsv или .xlsx. С помощью библиотеки Pandas такие табличные данные очень удобно загружать, обрабатывать и анализировать с помощью SQL-подобных запросов. А в связке с библиотеками Matplotlib и Seaborn Pandas предоставляет широкие возможности визуального анализа табличных данных.

Основными структурами данных в Pandas являются классы **Series** и **DataFrame**. Первый из них представляет собой одномерный индексированный массив данных некоторого фиксированного типа. Второй – это двухмерная структура данных, представляющая собой таблицу, каждый столбец которой содержит данные одного типа. Можно представлять её как словарь объектов типа Series. Структура DataFrame отлично подходит для представления реальных данных: строки соответствуют признаковым описаниям отдельных объектов, а столбцы соответствуют признакам.

```
# импортируем Pandas и Numpy
import pandas as pd
import numpy as np
```

Будем показывать основные методы в деле, анализируя **набор данных** по оттоку клиентов телеком-оператора (скачивать не нужно, он есть в репозитории). Прочитаем данные (метод read_csv) и посмотрим на первые 5 строк с помощью метода head:

```
df = pd.read_csv('../../data/telecom_churn.csv')
```

```
df.head()
```

	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
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	False
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	False
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	False
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	False
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	False

Про вывод датафрейма в тетрадке Jupyter

В Jupyter-ноутбуках датафреймы Pandas выводятся в виде вот таких красивых табличек, и print(df.head()) выглядит хуже.

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

```
pd.set_option('display.max_columns', 100)
pd.set_option('display.max_rows', 100)
```

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

Описание признаков

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()
```

	Account length	Area code	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
count	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00	3333.00
mean	101.06	437.18	8.10	179.78	100.44	30.56	200.98	100.11	17.08	200.87	100.11	9.04	10.24	4.48	2.76	1.56	0.14
std	39.82	42.37	13.69	54.47	20.07	9.26	50.71	19.92	4.31	50.57	19.57	2.28	2.79	2.46	0.75	1.32	0.35
min	1.00	408.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	23.20	33.00	1.04	0.00	0.00	0.00	0.00	0.00
25%	74.00	408.00	0.00	143.70	87.00	24.43	166.60	87.00	14.16	167.00	87.00	7.52	8.50	3.00	2.30	1.00	0.00
50%	101.00	415.00	0.00	179.40	101.00	30.50	201.40	100.00	17.12	201.20	100.00	9.05	10.30	4.00	2.78	1.00	0.00
75%	127.00	510.00	20.00	216.40	114.00	36.79	235.30	114.00	20.00	235.30	113.00	10.59	12.10	6.00	3.27	2.00	0.00
max	243.00	510.00	51.00	350.80	165.00	59.64	363.70	170.00	30.91	395.00	175.00	17.77	20.00	20.00	5.40	9.00	1.00

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()
```

	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
365	CO	154	415	No	No	0	350.8	75	59.64	216.5	94	18.40	253.9	100	11.43	10.1	9	2.73	1	1
985	NY	64	415	Yes	No	0	346.8	55	58.96	249.5	79	21.21	275.4	102	12.39	13.3	9	3.59	1	1
2594	OH	115	510	Yes	No	0	345.3	81	58.70	203.4	106	17.29	217.5	107	9.79	11.8	8	3.19	1	1
156	OH	83	415	No	No	0	337.4	120	57.36	227.4	116	19.33	153.9	114	6.93	15.8	7	4.27	0	1
605	MO	112	415	No	No	0	335.5	77	57.04	212.5	109	18.06	265.0	132	11.93	12.7	8	3.43	2	1

Сортировать можно и по группе столбцов:

```
df.sort_values(by=['Churn', 'Total day charge'],
               ascending=[True, False]).head()
```

	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
688	MN	13	510	No	Yes	21	315.6	105	53.65	208.9	71	17.76	260.1	123	11.70	12.1	3	3.27	3	0
2259	NC	210	415	No	Yes	31	313.8	87	53.35	147.7	103	12.55	192.7	97	8.67	10.1	7	2.73	3	0
534	LA	67	510	No	No	0	310.4	97	52.77	66.5	123	5.65	246.5	99	11.09	9.2	10	2.48	4	0
575	SD	114	415	No	Yes	36	309.9	90	52.68	200.3	89	17.03	183.5	105	8.26	14.2	2	3.83	1	0
2858	AL	141	510	No	Yes	28	308.0	123	52.36	247.8	128	21.06	152.9	103	6.88	7.4	3	2.00	1	0

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?**

```
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»*.

Когда мы передаём slice object в **iloc**, датафрейм слайсится как обычно. Однако в случае с **loc** учитываются и начало, и конец.

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

	State	Account length	Area code
0	KS	128	415

	State	Account length	Area code
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:]
```

	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
3332	TN	74	415	No	Yes	25	234.4	113	39.85	265.9	82	22.6	241.4	77	10.86	13.7	4	3.7	0	0

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()
```

	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
0	KS	128	415	False	Yes	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	0
1	OH	107	415	False	Yes	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	0
2	NJ	137	415	False	No	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	0
3	OH	84	408	True	No	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	0
4	OK	75	415	True	No	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	0

Аналогичную операцию можно провернуть с помощью метода replace:

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

	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
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	0
1	OH	107	415	False	True	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	0
2	NJ	137	415	False	False	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	0
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	0
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	0

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=[])
```

	Total day minutes						Total eve minutes						Total night minutes					
	count	mean	std	min	50%	max	count	mean	std	min	50%	max	count	mean	std	min	50%	max
Churn																		
0	2850.0	175.18	50.18	0.0	177.2	315.6	2850.0	199.04	50.29	0.0	199.6	361.8	2850.0	200.13	51.11	23.2	200.25	395.0
1	483.0	206.91	69.00	0.0	217.6	350.8	483.0	212.41	51.73	70.9	211.3	363.7	483.0	205.23	47.13	47.4	204.80	354.9

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])
```

	Total day minutes				Total eve minutes				Total night minutes			
	mean	std	amin	amax	mean	std	amin	amax	mean	std	amin	amax
Churn												
0	175.18	50.18	0.0	315.6	199.04	50.29	0.0	361.8	200.13	51.11	23.2	395.0
1	206.91	69.00	0.0	350.8	212.41	51.73	70.9	363.7	205.23	47.13	47.4	354.9

```
column_to_show = ['Total day minutes','Total eve minutes', 'Total night minutes']
```

Сводные таблицы

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'],normalize=False)
```

International plan	**No**	**Yes**
Churn		
0	2664	186
1	346	137

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

International 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()
```

	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	Total calls
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	0	303
1	OH	107	415	False	True	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	0	332
2	NJ	137	415	False	False	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	0	333
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	0	255
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	0	359

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()
```

	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	Total calls	Total charge
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	0	303	75.56
1	OH	107	415	False	True	26	161.6	123	27.47	195.5	103	16.62	254.4	103	11.45	13.7	3	3.70	1	0	332	59.24
2	NJ	137	415	False	False	0	243.4	114	41.38	121.2	110	10.30	162.6	104	7.32	12.2	5	3.29	0	0	333	62.29
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	0	255	66.80
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	0	359	52.09

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
```

	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	Total calls	Total charge
0	KS	128	415	False	True	25	265.1	110	45.07	197.4	99	16.78	244.7	91	11.01	10.0	3	2.70	1	0	303	75.56
3	OH	84	408	True	False	0	299.4	71	50.90	61.9	88	5.26	196.9	89	8.86	6.6	7	1.78	2	0	255	66.80
4	OK	75	415	True	False	0	166.7	113	28.34	148.3	122	12.61	186.9	121	8.41	10.1	3	2.73	3	0	359	52.09
5	AL	118	510	True	False	0	223.4	98	37.98	220.6	101	18.75	203.9	118	9.18	6.3	6	1.70	0	0	323	67.61
6	MA	121	510	False	True	24	218.2	88	37.09	348.5	108	29.62	212.6	118	9.57	7.5	7	2.03	3	0	321	78.31

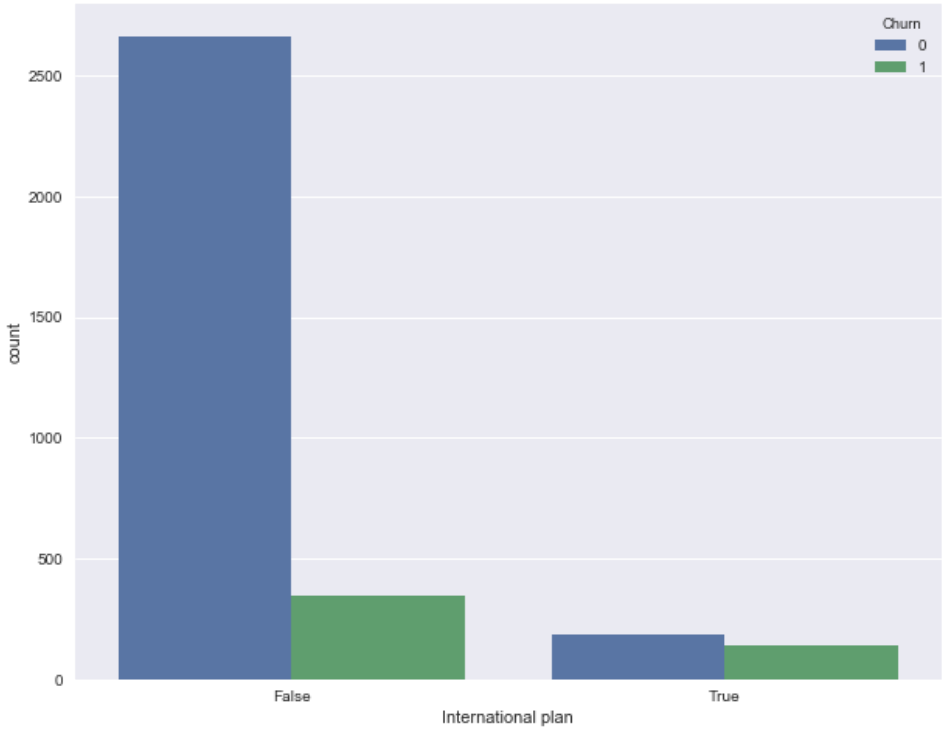
Первые попытки прогнозирования оттока

Посмотрим, как отток связан с признаком *"Подключение международного роуминга" (International plan)*. Сделаем это с помощью сводной

таблички *crosstab*, а также путем иллюстрации с Seaborn (как именно строить такие картинки и анализировать с их помощью графики – материал следующей статьи).

```
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



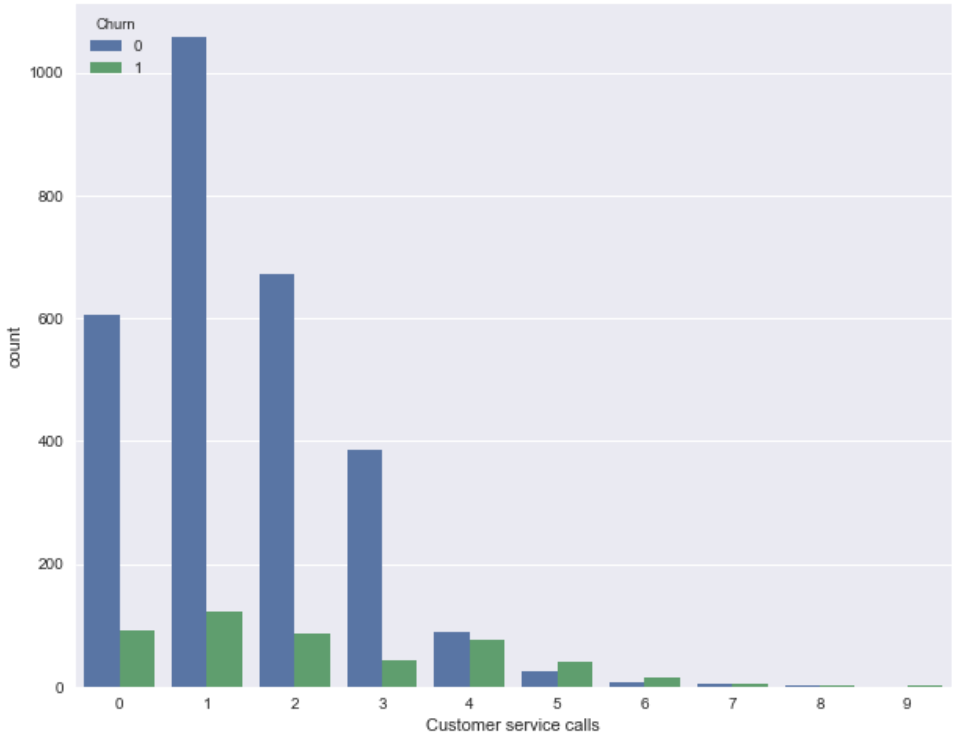
Видим, что когда роуминг подключен, доля оттока намного выше – интересное наблюдение! Возможно, большие и плохо контролируемые траты в роуминге очень конфликтогенны и приводят к недовольству клиентов

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

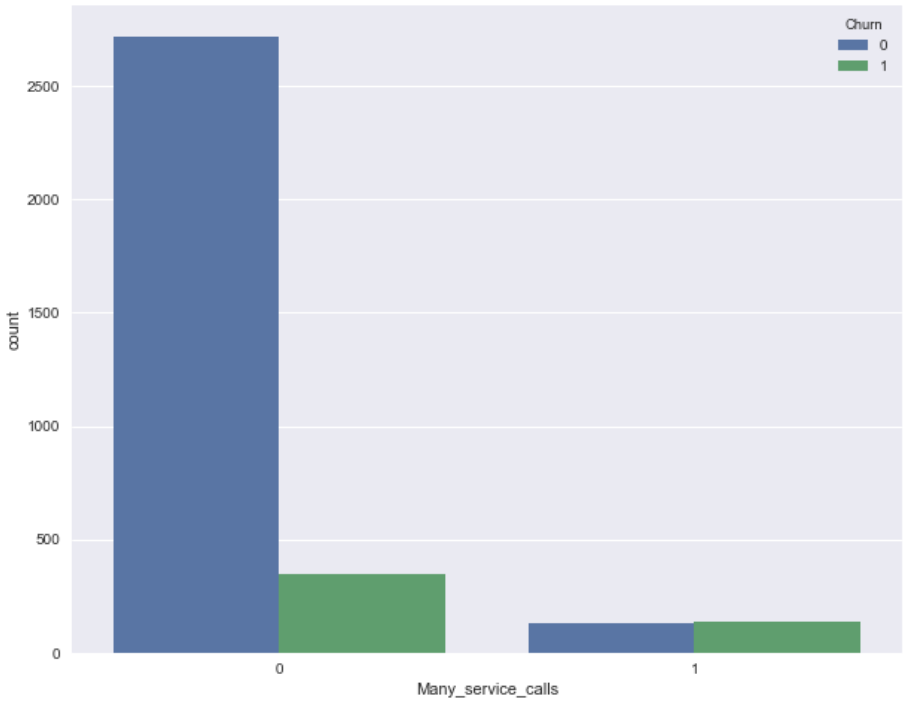
в сервисный центр.

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.