Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«Уральский федеральный университет имени первого Президента России Б. Н. Ельцина»

УТВЕРЖДАЮ

Директор по образовательной деятельности

_____ С.Т. Князев

«7» сентября 2023 г.

Сбор и верификация данных

Учебно-методические материалы по направлению подготовки
09.03.03 Прикладная информатика
Образовательная программа «Прикладной искусственный интеллект»

Екатеринбург

РАЗРАБОТЧИКИ УЧЕБНО-МЕТОДИЧЕСКИХ МАТЕРИАЛОВ

Доцент, канд.техн.наук _____ Корелин Иван Андреевич

СОДЕРЖАНИЕ

1. Основные принципы сбора и первичной обработки данных 5
   1.1. Сигналы 5
   1.2. Общие сведения о сигналах 5
   1.3. Классификация сигналов 6
   1.4. Первичные эмпирические сведения о социальных и экономических фактах 6
   1.5. Парсинг (скраппинг) сайтов 8
2. Создание данных. 14
   2.1. Дискретизация и квантование. 14
   2.2. Первичный анализ данных с Pandas 16
   2.3. Базовые операции в Pandas 17
3. Математические модели, методы и алгоритмы сбора и верификации наукометрических данных. 19
   3.1. Применение функций к ячейкам, столбцам и строкам 19
   3.2. Фильтрация кадров данных 20
   3.3. Объединение кадров данных 21
   3.4. Изучение данных 22
   3.5. Сортировка 27
   3.6. Группировка данных 28
4. Поиск и устранение ошибок в данных. Заполнение недостающих данных. 32
   4.1. Поиск ошибок в данных 32
   4.2. Очистка данных 37
   4.3. Заполнение недостающих данных в pandas DataFrame 39
5. Форматы данных. 40
   5.1. Источники данных поддерживаемые в Pandas 40
   5.3. Чтение из SQL баз данных 42
   5.4. Запись в базу 45
6. Качество данных. 47
   6.1. Определения 47
   6.2. Измерения качества данных 48
   6.3. Контроль качества данных 54
   6.4. Оптимальное использование качества данных 56
7. Разметка данных. Сценарии применения размеченных данных в задачах компьютерного зрения и обработки естественного языка 58
   7.1. Семантическая сегментация изображений в CVAT 58
   7.2. Разметка именованных сущностей в Label Studio 60
   7.3. Сценарии применения размеченных данных в задачах компьютерного зрения и обработки естественного языка 65
8. Обработка данных. Закономерности и аномалии. Визуализация данных. 71

8.1. Обзор набора данных 71
8.2. Seaborn 74
8.1. Plotly 80
8.2. Пример визуального анализа данных 84

1. Основные принципы сбора и первичной обработки данных

1.1. Сигналы

Несмотря на интуитивную понятность термина сигнал, найти точное определение этого понятия достаточно сложно. Обычно под сигналом понимают величину, отражающую каким-либо образом состояние физической системы. В этом смысле естественно рассматривать сигнал как результат некоторых измерений, проводимых над физической системой в процессе её наблюдения. ***Сигнал*** — это изменяющаяся во времени физическая величина, описываемая функцией времени $f(t)$. Один из параметров этой функции (чаще всего это значение функции) содержит информацию о некоторой физической величине. Такой параметр сигнала (функции) называют информативным, а физическую величину, которой представлен сигнал, — носителем сигнала (несущей сигнала); сигнал имеет размерность этой величины. Пример непрерывного аналогового сигнала представлен на рисунке 1.

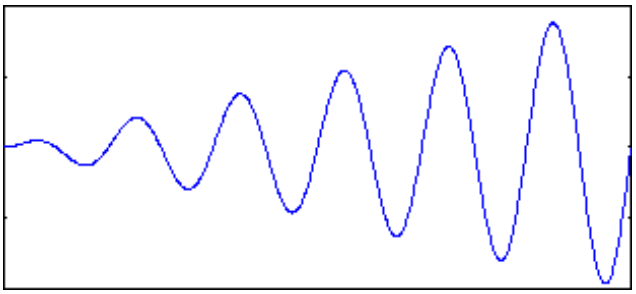

Рис. 1 Аналоговый сигнал

Сигналом обычно называют то, что несёт в себе какие-то данные.

1.2. Общие сведения о сигналах

Сигнал может генерироваться, но его приём не обязателен, в отличие от сообщения, которое рассчитано на принятие принимающей стороной, иначе оно не является сообщением. Сигналом может быть любой физический процесс, параметры которого изменяются (или находятся) в соответствии с передаваемым сообщением.

Сигнал, детерминированный или случайный, описывают математической моделью, функцией, характеризующей изменение параметров сигнала. Математическая модель представления сигнала, как функции времени, является основополагающей концепцией теоретической радиотехники, оказавшейся плодотворной как для анализа, так и для синтеза радиотехнических устройств и систем. В радиотехнике альтернативой сигналу, который несёт полезную информацию, является шум — обычно случайная функция времени, взаимодействующая (например, путём сложения) с сигналом и искажающая его. Основной задачей теоретической радиотехники является извлечение полезной информации из сигнала с обязательным учётом шума.

Понятие сигнал позволяет абстрагироваться от конкретной физической величины, например тока, напряжения, акустической волны и рассматривать вне физического контекста явления связанные кодированием информации и извлечением её из сигналов, которые обычно искажены шумами. В исследованиях сигнал часто представляется функцией времени, параметры которой могут нести нужную информацию. Способ записи этой функции, а также способ записи мешающих шумов называют математической моделью сигнала.

В связи с понятием сигнала формулируются такие базовые принципы кибернетики, как понятие о пропускной способности канала связи, разработанное Клодом Шенноном и об оптимальном приёме, разработанная В. А. Котельниковым.

1.3. Классификация сигналов

По физической природе носителя информации:

- электрические;
- электромагнитные;
- оптические;
- акустические
- и другие;

По способу задания сигнала:

- регулярные (детерминированные), заданные аналитической функцией;
- нерегулярные (случайные), принимающие произвольные значения в любой момент времени. Для описания таких сигналов используется аппарат теории вероятностей.

В зависимости от функции, описывающей параметры сигнала, выделяют:

- непрерывные (аналоговые),
- непрерывно-квантованные,
- дискретно-непрерывные и
- дискретно-квантованные (цифровые) сигналы.

1.4. Первичные эмпирические сведения о социальных и экономических фактах

В отличии от технических систем, где физическое явление описывается сигналом, чаще всего для сбора сведений о социальных явлениях используются инструменты ручного сбора данных. Так, например, имя,

даваемое человеку при рождении, записывается в некоторую систему хранения и обработки данных вручную. Эмпирическая оценка уровней запаса полезных ископаемых и экономической целесообразности их добычи зависит не только от физических остатков, но и труднодоступности и стоимость на рынке. А это значит, что по-сути целесообразность добычи имеет не физическую природу, и описать её сигналами невозможно. Но это вовсе не означает, что нельзя попытаться работать с данными, описывающими социальные факты. Одним из инструментов является социологический опрос.

Социологический опрос (соцопрос) — метод социологического исследования, заключающийся в сборе и получении первичных эмпирических сведений об определённых мнениях, знаниях и социальных фактах, составляющих предмет исследования, путём устного или письменного взаимодействия исследователя (интервьюера) и заданной совокупности опрашиваемых (интервьюируемые, респонденты).

«Метод опроса — самый распространенный из социологических методов, определяющий „образ“ социологии в глазах непосвященных и, к тому же, имеющий самую богатую и давнюю историю. Утверждение о том, что почти невозможно дать строгое и исчерпывающее определение того, что такое опрос, на первый взгляд кажется нелепостью. Однако в действительности представления о том, каким должен быть хороший социологический опрос, менялись так часто, что любая попытка свести определение опроса к конкретной технике сбора информации, плану исследования, типу анализа данных или характеру использования полученных сведений наверняка столкнется с трудностями», – И. Ф. Девятко, Методы социологического исследования, 1998.[2]
Социологический опрос — один из самых распространённых способов сбора необходимой информации в современной социологии и маркетинге.
Результаты опроса в цифровом виде фиксируются в системах управления Базами данных. В таких системах опросы можно анализировать и обрабатывать.

1.5. Парсинг (скраппинг) сайтов

Иногда владелец данных не предоставляет их для прямого использования. Для примера мы рассмотрим следующую задачу:
Задача будет состоять в том, чтобы выгрузить данные о просмотренных фильмах на КиноПоиске: название фильма (русское, английское), дату и время просмотра, оценку пользователя.
Инструменты
Для отправки http-запросов есть немало python-библиотек, наиболее известные urllib/urllib2 и Requests. На мой вкус Requests удобнее и лаконичнее, так что, буду использовать ее. Также необходимо выбрать библиотеку для парсинга html, небольшой research дает следующие варианты:

re
Регулярные выражения, конечно, нам пригодятся, но использовать только их, на мой взгляд, слишком хардкорный путь, и они немного не для этого. Были придуманы более удобные инструменты для разбора html, так что перейдем к ним.
BeatifulSoup, **lxml**
Это две наиболее популярные библиотеки для парсинга html и выбор одной из них, скорее, обусловлен личными предпочтениями. Более того, эти библиотеки тесно переплелись: BeautifulSoup стал использовать lxml в качестве внутреннего парсера для ускорения, а в lxml был добавлен модуль soupparser. Подробнее про плюсы и минусы этих библиотек можно почитать в обсуждении. Для сравнения подходов я буду парсить данные с помощью BeautifulSoup и используя XPath селекторы в модуле lxml.html.
scrapy
Это уже не просто библиотека, а целый open-source framework для получения данных с веб-страниц. В нем есть множество полезных функций: асинхронные запросы, возможность использовать XPath и CSS селекторы для обработки данных, удобная работа с кодировками и многое другое (подробнее можно почитать тут). Если бы моя задача была не разовой выгрузкой, а production процессом, то я бы выбрала его. В текущей постановке это overkill.

Загрузка данных
Приступим к выгрузке данных. Для начала, попробуем просто получить страницу по url и сохранить в локальный файл.

```
import requests
user_id = 12345
url =
'http://www.kinopoisk.ru/user/%d/votes/list/ord/date/page/2/#list' %
(user_id) # url для второй страницы
r = requests.get(url)
with open('test.html', 'w') as output_file:
  output_file.write(r.text.encode('cp1251'))
```

Открываем полученный файл и видим, что все не так просто: сайт распознал в нас робота и не спешит показывать данные.

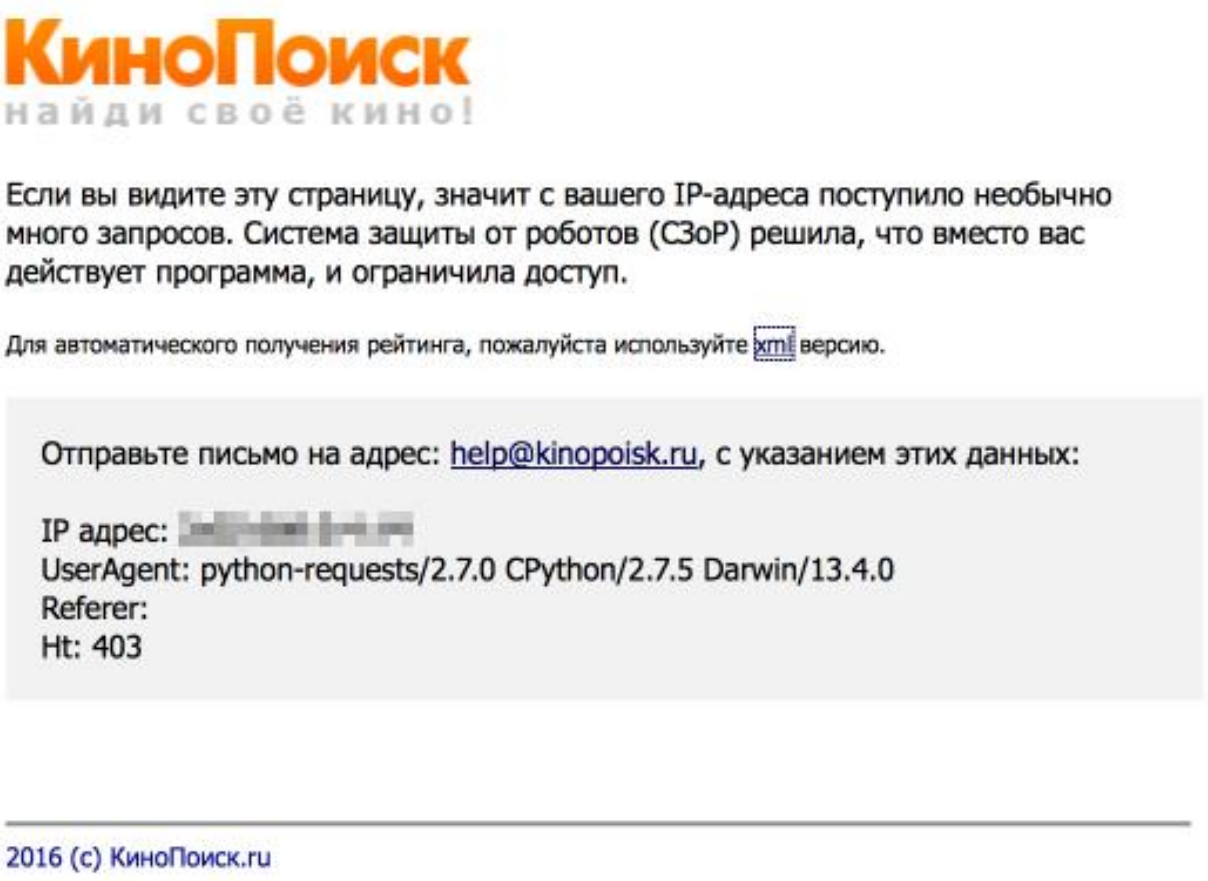

Рис. 2 Содержимое скачанного файла

Разберемся, как работает браузер
Однако, у браузера отлично получается получать информацию с сайта. Посмотрим, как именно он отправляет запрос. Для этого воспользуемся панелью "Сеть" в "Инструментах разработчика" в браузере (я использую для этого Firebug), обычно нужный нам запрос — самый продолжительный.

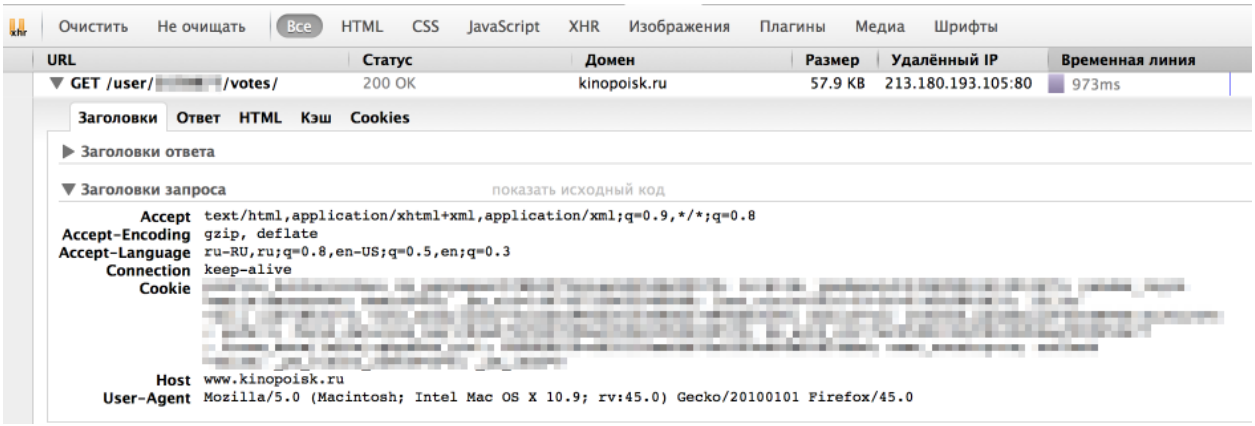

Рис. 3 Просмотр заголовков

Как мы видим, браузер также передает в headers UserAgent, cookie и еще ряд параметров. Для начала попробуем просто передать в header корректный UserAgent.

```
headers = {
    'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:45.0) Gecko/20100101
Firefox/45.0'
    }
r = requests.get(url, headers = headers)
```

На этот раз все получилось, теперь нам отдаются нужные данные. Стоит отметить, что иногда сайт также проверяет корректность cookie, в таком случае помогут sessions в библиотеке Requests.

Скачаем все оценки

Теперь мы умеем сохранять одну страницу с оценками. Но обычно у пользователя достаточно много оценок и нужно проитерироваться по всем страницам. Интересующий нас номер страницы легко передать непосредственно в url. Остается только вопрос: "Как понять сколько всего страниц с оценками?" Я решила эту проблему следующим образом: если указать слишком большой номер страницы, то нам вернется вот такая страница без таблицы с фильмами. Таким образом мы можем итерироваться по страницам до тех, пор пока находится блок с оценками фильмов ( ).

Рис.4 Содержимое скачанного файла

```
import requests
# establishing session
s = requests.Session()
s.headers.update({
        'User-Agent': 'Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:45.0) Gecko/20100101 Firefox/45.0'
    })

def load_user_data(user_id, page, session):
    url = 'http://www.kinopoisk.ru/user/%d/votes/list/ord/date/page/%d/#list' % (user_id, page)
    request = session.get(url)
    return request.text

def contain_movies_data(text):
    soup = BeautifulSoup(text)
    film_list = soup.find('div', {'class': 'profileFilmsList'})
    return film_list is not None

# loading files
page = 1
while True:
    data = load_user_data(user_id, page, s)
    if contain_movies_data(data):
        with open('./page_%d.html' % (page), 'w') as output_file:
            output_file.write(data.encode('cp1251'))
            page += 1
    else:
```

```
        break
```

Парсинг

Немного про XPath

XPath — это язык запросов к xml и xhtml документов. Мы будем использовать XPath селекторы при работе с библиотекой lxml (документация). Рассмотрим небольшой пример работы с XPath

```
from lxml import html
test = '''
   <html>
      <body>
         <div class="first_level">
            <h2 align='center'>one</h2>
            <h2 align='left'>two</h2>
         </div>
         <h2>another tag</h2>
      </body>
   </html>
'''
tree = html.fromstring(test)
tree.xpath('//h2') # все h2 теги
tree.xpath('//h2[@align]') # h2 теги с атрибутом align
tree.xpath('//h2[@align="center"]') # h2 теги с атрибутом align равным "center"

div_node = tree.xpath('//div')[0] # div тег
div_node.xpath('.//h2') # все h2 теги, которые являются дочерними div ноде
```

Подробнее про синтаксис XPath также можно почитать на W3Schools.

Вернемся к нашей задаче

Теперь перейдем непосредственно к получению данных из html. Проще всего понять как устроена html-страница используя функцию "Инспектировать эле мент" в браузере. В данном случае все довольно просто: вся таблица с оценка ми заключена в теге . Выделим эту ноду:

```
from bs4 import BeautifulSoup
from lxml import html

# Beautiful Soup
soup = BeautifulSoup(text)
film_list = soup.find('div', {'class': 'profileFilmsList'})

# lxml
tree = html.fromstring(text)
film_list_lxml = tree.xpath('//div[@class = "profileFilmsList"]')[0]
```

Каждый фильм представлен как или . Рассмотрим, как вытащить русское название фильма и ссылку на страни цу фильма (также узнаем, как получить текст и значение атрибута).

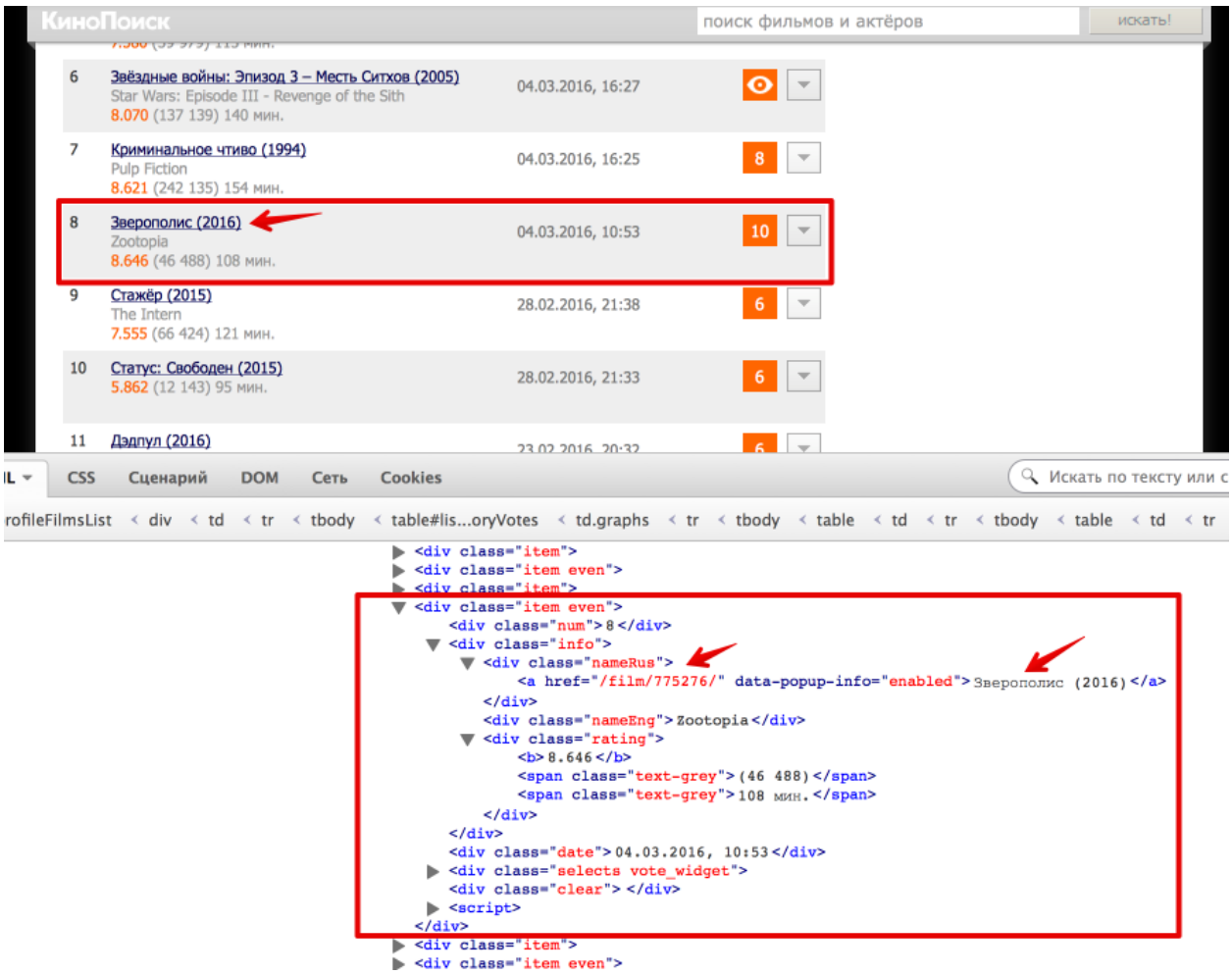

Рис.5 Содержимое скачанного файла

```
# Beatiful Soup
movie_link = item.find('div', {'class': 'nameRus'}).find('a').get('href')
movie_desc = item.find('div', {'class': 'nameRus'}).find('a').text

# lxml
movie_link = item_lxml.xpath('.//div[@class = "nameRus"]/a/@href')[0]
movie_desc = item_lxml.xpath('.//div[@class = "nameRus"]/a/text()')[0]
```

Еще небольшой хинт для debug'a: для того, чтобы посмотреть, что внутри вы бранной ноды в BeautifulSoup можно просто распечатать ее, а в lxml воспольз оваться функцией tostring() модуля etree.

```
# BeatifulSoup
print item

#lxml
from lxml import etree
print etree.tostring(item_lxml) default.
```

2. Создание данных

2.1. Дискретизация и квантование

Все сигналы можно разделить на 4 категории: непрерывные (аналоговые), непрерывно-квантованные, дискретно-непрерывные и дискретно-квантованные сигналы (цифровые).

Аналоговые сигналы.

Большинство сигналов имеют непрерывную зависимость от независимой переменной (например, изменяются непрерывно во времени) и могут принимать любые значения на некотором интервале. «Сигналы в непрерывном времени и с непрерывным диапазоном амплитуд также называются аналоговыми сигналами». Аналоговые сигналы (АС) оказывается возможным описать некоторой непрерывной математической функцией времени.

Пример АС — гармонический сигнал: $s(t) = A \cdot \cos(\omega \cdot t + \varphi)$.

Аналоговые сигналы используются в телефонии, радиовещании, телевидении. Ввести такой сигнал в цифровую систему для обработки невозможно, так как на любом интервале времени он может иметь бесконечное множество значений, и для точного (без погрешности) представления его значения требуются числа бесконечной разрядности. Поэтому очень часто необходимо преобразовывать аналоговый сигнал так, чтобы можно было представить его последовательностью чисел заданной разрядности.

Среди экспертов существует мнение, что термин «аналоговый сигнал» следует считать неудачным и устаревшим, а вместо него следует использовать термин «непрерывный сигнал».

Дискретно-непрерывный (дискретный) сигнал

«Дискретные сигналы (сигналы в дискретном времени) определяются в дискретные моменты времени и представляются последовательностью чисел». Дискретизация аналогового сигнала состоит в том, что сигнал представляется в виде последовательности значений, взятых в дискретные моменты времени t_i (где i — индекс). Обычно промежутки времени между последовательными отсчётами ($\Delta t_i = t_i - t_{i-1}$) постоянны; в таком случае, Δt называется интервалом дискретизации. Сами же значения сигнала $x(t)$ в моменты измерения, то есть $x_i = x(t_i)$, называются отсчётами.

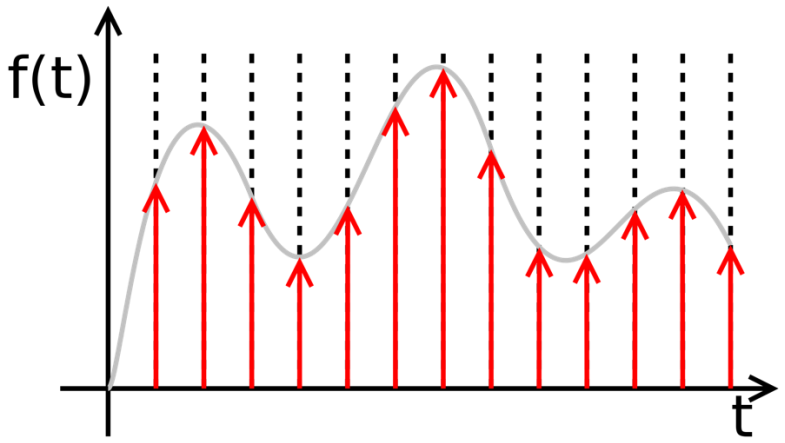

Рис.6 Дискретный сигнал

Непрерывно-квантованный сигнал

При квантовании вся область значений сигнала разбивается на уровни, количество которых должно быть представлено в числах заданной разрядности. Расстояния между этими уровнями называется шагом квантования Δ. Число этих уровней равно N (от 0 до $N-1$). Каждому уровню присваивается некоторое число. Отсчёты сигнала сравниваются с уровнями квантования и в качестве сигнала выбирается число, соответствующее некоторому уровню квантования. Каждый уровень квантования кодируется двоичным числом с n разрядами. Число уровней квантования N и число разрядов n двоичных чисел, кодирующих эти уровни, связаны соотношением $n \geq \log2(N)$.

В соответствии с ГОСТ 26.013-81, такие сигналы обозначены термином «многоуровневый сигнал».

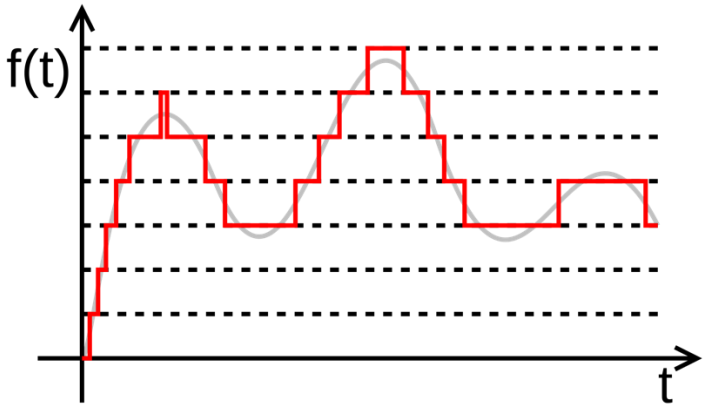

Рис.7 Квантованный сигнал

Цифровой сигнал

К цифровым сигналам относят те, у которых дискретны как независимая переменная (например, время), так и уровень.

Для того, чтобы представить аналоговый сигнал последовательностью чисел конечной разрядности, его следует сначала превратить в дискретный сигнал, а затем подвергнуть квантованию. Квантование является частным случаем дискретизации, когда дискретизация происходит по одинаковой величине, называемой квантом. В результате сигнал будет представлен таким образом, что на каждом заданном промежутке времени известно приближённое (квантованное) значение сигнала, которое можно записать целым числом. Последовательность таких чисел и будет являться цифровым сигналом.

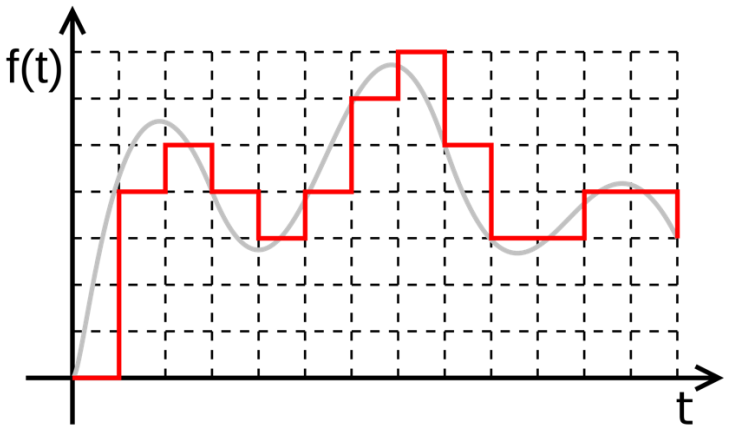

Рис.8 Цифровой сигнал

2.2. Первичный анализ данных с Pandas

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.

```
!wget https://datahub.io/core/airport-codes/r/airport-codes.csv -O /content/sample_data/airport-codes.csv -q
```

```
!head -n 10 /content/sample_data/airport-codes.csv
```

```
ident,type,name,elevation_ft,continent,iso_country,iso_region,municipality,gps_code,iata_code,local_code,coordinates
00A,heliport,Total Rf Heliport,11,NA,US,US-PA,Bensalem,00A,,00A,"-74.93360137939453, 40.07080078125"
00AA,small_airport,Aero B Ranch Airport,3435,NA,US,US-KS,Leoti,00AA,,00AA,"-101.473911, 38.704022"
00AK,small_airport,Lowell Field,450,NA,US,US-AK,Anchor Point,00AK,,00AK,"-151.695999146, 59.94919968"
00AL,small_airport,Epps Airpark,820,NA,US,US-AL,Harvest,00AL,,00AL,"-86.77030181884766, 34.86479949951172"
00AR,closed,Newport Hospital & Clinic Heliport,237,NA,US,US-AR,Newport,,,,"-91.254898, 35.6087"
```

```
00AS,small_airport,Fulton Airport,1100,NA,US,US-OK,Alex,00AS,,00AS,"-97.81801
94, 34.9428028"
00AZ,small_airport,Cordes Airport,3810,NA,US,US-AZ,Cordes,00AZ,,00AZ,"-112.16
500091552734, 34.305599212646484"
00CA,small_airport,Goldstone /Gts/ Airport,3038,NA,US,US-CA,Barstow,00CA,,00C
A,"-116.888000488, 35.350498199499995"
00CL,small_airport,Williams Ag Airport,87,NA,US,US-CA,Biggs,00CL,,00CL,"-121.
763427, 39.427188"
```

2.3. Базовые операции в Pandas

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

airports = pd.read_csv('sample_data/airport-codes.csv')
airports.head(200)
```

```
    ident           type                                        name  \
0     00A       heliport                         Total Rf Heliport
1    00AA  small_airport                       Aero B Ranch Airport
2    00AK  small_airport                              Lowell Field
3    00AL  small_airport                              Epps Airpark
4    00AR         closed          Newport Hospital & Clinic Heliport
..    ...            ...                                       ...
195  03AZ  small_airport  Thompson International Aviation Airport
196  03CA       heliport                 Grossmont Hospital Heliport
197  03CO  small_airport                       Kugel-Strong Airport
198  03FA  small_airport                   Lake Persimmon Airstrip
199  03FD         closed                              Tharpe Airport

     elevation_ft continent iso_country iso_region  municipality gps_code  \
0            11.0       NaN          US      US-PA      Bensalem      00A
1          3435.0       NaN          US      US-KS         Leoti     00AA
2           450.0       NaN          US      US-AK  Anchor Point     00AK
3           820.0       NaN          US      US-AL       Harvest     00AL
4           237.0       NaN          US      US-AR       Newport      NaN
..            ...       ...         ...         ...           ...      ...
195        4275.0       NaN          US      US-AZ      Hereford     03AZ
196         634.0       NaN          US      US-CA       La Mesa     03CA
197        4950.0       NaN          US      US-CO   Platteville     03CO
198          70.0       NaN          US      US-FL   Lake Placid     03FA
199         115.0       NaN          US      US-FL       Bonifay      NaN

    iata_code local_code                             coordinates
0         NaN        00A       -74.93360137939453, 40.07080078125
1         NaN       00AA                   -101.473911, 38.704022
2         NaN       00AK              -151.695999146, 59.94919968
3         NaN       00AL    -86.77030181884766, 34.86479949951172
4         NaN        NaN                      -91.254898, 35.6087
..        ...        ...                                      ...
195       NaN       03AZ  -110.08399963378906, 31.433399200439453
196       NaN       03CA                   -117.006952, 32.779484
197       NaN       03CO            -104.744003296, 40.2125015259
198       NaN       03FA   -81.40809631347656, 27.353099822998047
199       NaN        NaN                      -85.731003, 30.8288
```

```
[200 rows x 12 columns]
```

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

```
print(airports.head())
```

```
  ident          type                                name  elevation_ft  \
0   00A      heliport                Total Rf Heliport          11.0
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR        closed  Newport Hospital & Clinic Heliport         237.0

  continent iso_country iso_region  municipality gps_code iata_code  \
0       NaN          US      US-PA      Bensalem      00A       NaN
1       NaN          US      US-KS         Leoti     00AA       NaN
2       NaN          US      US-AK  Anchor Point     00AK       NaN
3       NaN          US      US-AL       Harvest     00AL       NaN
4       NaN          US      US-AR       Newport      NaN       NaN

  local_code                          coordinates
0        00A     -74.93360137939453, 40.07080078125
1       00AA                -101.473911, 38.704022
2       00AK            -151.695999146, 59.94919968
3       00AL  -86.77030181884766, 34.86479949951172
4        NaN                    -91.254898, 35.6087
```

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)
```

Посмотрим на размер данных, названия признаков и их типы.

```
print(airports.shape)
```

```
(57421, 12)
```

Видим, что в таблице 57421 строки и 12 столбцов. Выведем названия столбцов:

```
print(airports.columns)
```

```
Index(['ident', 'type', 'name', 'elevation_ft', 'continent', 'iso_country',
       'iso_region', 'municipality', 'gps_code', 'iata_code', 'local_code',
       'coordinates'],
      dtype='object')
```

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

```
print(airports.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 57421 entries, 0 to 57420
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   ident         57421 non-null  object
 1   type          57421 non-null  object
 2   name          57421 non-null  object
 3   elevation_ft  49608 non-null  float64
 4   continent     28978 non-null  object
 5   iso_country   57175 non-null  object
 6   iso_region    57421 non-null  object
 7   municipality  51527 non-null  object
 8   gps_code      41561 non-null  object
 9   iata_code     9225 non-null   object
 10  local_code    30030 non-null  object
 11  coordinates   57421 non-null  object
dtypes: float64(1), object(11)
memory usage: 5.3+ MB
None
```

3. Математические модели, методы и алгоритмы сбора и верификации наукометрических данных

3.1. Применение функций к ячейкам, столбцам и строкам

Как видно колонка coordinates не распозналась. Попробуем извлечь из неё данные, для этого воспользуемся методом apply и анонимными функциями. Для того, чтобы удалить лишний столбец воспользуемся методом drop с параметром axis=1.

```
airports['latitude'] = airports['coordinates'].apply(lambda x : float(x.split
(',')[0]))
airports['altitude'] = airports['coordinates'].apply(lambda x : float(x.split
(',')[1]))
airports = airports.drop(['coordinates'], axis=1)
airports.head()
```

```
  ident          type                                name  elevation_ft  \
0   00A      heliport                Total Rf Heliport          11.0
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR        closed  Newport Hospital & Clinic Heliport         237.0

  continent iso_country iso_region  municipality gps_code iata_code  \
0       NaN          US      US-PA      Bensalem      00A       NaN
1       NaN          US      US-KS         Leoti     00AA       NaN
2       NaN          US      US-AK  Anchor Point     00AK       NaN
3       NaN          US      US-AL       Harvest     00AL       NaN
4       NaN          US      US-AR       Newport      NaN       NaN

  local_code    latitude   altitude
0        00A  -74.933601  40.070801
```

```
1       00AA -101.473911  38.704022
2       00AK -151.695999  59.949200
3       00AL  -86.770302  34.864799
4        NaN  -91.254898  35.608700
```

3.2. Фильтрация кадров данных

Часто данные требуется отфильтровать. Причём фильтрация может потребоваться как по атрибутам (колонкам) так и по кортежам (строкам) фильтры регулируются по оси. Можно фильтровать кадры данных можно без удаления filter или с удалением drop.

```
airports.filter(items=['ident', 'type', 'continent', 'name']).head() # фильтрация колонок
```

```
  ident           type continent                                name
0   00A       heliport                                Total Rf Heliport
1  00AA  small_airport                           Aero B Ranch Airport
2  00AK  small_airport                                   Lowell Field
3  00AL  small_airport                                   Epps Airpark
4  00AR         closed            Newport Hospital & Clinic Heliport
```

```
airports.filter(like='00', axis=0).head() # фильтрация строк
```

```
    ident           type               name  elevation_ft continent  \
100  01NE  small_airport     Detour Airport        3000.0
200  03FL       heliport    Ranger Heliport          20.0
300  04TX  small_airport     Pocock Airport         565.0
400  06MO  small_airport  Noahs Ark Airport         755.0
500  08ID  small_airport      Symms Airport        2680.0

    iso_country iso_region     municipality gps_code iata_code local_code  \
100          US      US-NE        Wellfleet     01NE         0       01NE
200          US      US-FL  West Palm Beach     03FL         0       03FL
300          US      US-TX     China Spring     04TX         0       04TX
400          US      US-MO          Waldron     06MO         0       06MO
500          US      US-ID          Marsing     08ID         0       08ID

       latitude   altitude
100 -100.653000  40.843601
200  -80.187302  26.683701
300  -97.368896  31.732201
400  -94.804398  39.230598
500 -116.777000  43.569302
```

```
airports.drop(['continent'], axis=1).head()
```

```
  ident           type                                name  elevation_ft  \
0   00A       heliport                Total Rf Heliport          11.0
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                       Lowell Field         450.0
3  00AL  small_airport                       Epps Airpark         820.0
4  00AR         closed  Newport Hospital & Clinic Heliport         237.0

  iso_country iso_region  municipality gps_code iata_code local_code  \
0          US      US-PA      Bensalem      00A         0        00A
1          US      US-KS         Leoti     00AA         0       00AA
```

```
2          US      US-AK  Anchor Point     00AK         0       00AK
3          US      US-AL       Harvest     00AL         0       00AL
4          US      US-AR       Newport      NaN         0        NaN

     latitude   altitude
0  -74.933601  40.070801
1 -101.473911  38.704022
2 -151.695999  59.949200
3  -86.770302  34.864799
4  -91.254898  35.608700
```

```
airports.drop(index=0, axis=0).head()
```

```
  ident          type                                name  elevation_ft  \
1  00AA  small_airport               Aero B Ranch Airport        3435.0
2  00AK  small_airport                      Lowell Field         450.0
3  00AL  small_airport                      Epps Airpark         820.0
4  00AR        closed  Newport Hospital & Clinic Heliport         237.0
5  00AS  small_airport                     Fulton Airport        1100.0

  continent iso_country iso_region  municipality gps_code iata_code  \
1                    US      US-KS         Leoti     00AA         0
2                    US      US-AK  Anchor Point     00AK         0
3                    US      US-AL       Harvest     00AL         0
4                    US      US-AR       Newport      NaN         0
5                    US      US-OK          Alex     00AS         0

  local_code    latitude   altitude
1       00AA -101.473911  38.704022
2       00AK -151.695999  59.949200
3       00AL  -86.770302  34.864799
4        NaN  -91.254898  35.608700
5       00AS  -97.818019  34.942803
```

3.3. Объединение кадров данных

Распространённая задача — это обогащение данных, т.е. объединение кадра с источником приносящим дополнительную информацию. Объединение кадров производится по ключевому атрибуту. Рассмотрим пример:

```
poles = pd.DataFrame({'continent':['Unknown', 'Oceania', 'Africa', 'Africa',
'Antarctida', 'Europe', 'Asia', 'South America'], 'pole':['Unknown','South','
South','North','South','North','North','South']})
poles.head()
```

```
    continent     pole
0     Unknown  Unknown
1     Oceania    South
2      Africa    South
3      Africa    North
4  Antarctida    South
```

```
airports.set_index('continent').join(poles.set_index('continent'), on='contin
ent',how='left').filter(like='Africa', axis=0).head()
```

```
          ident          type              name  elevation_ft iso_country  \
continent
```

```
Africa      AAD  small_airport     Adado Airport        1001.0          SO
Africa      AAD  small_airport     Adado Airport        1001.0          SO
Africa      ADV  small_airport  El Daein Airport        1560.0          SD
Africa      ADV  small_airport  El Daein Airport        1560.0          SD
Africa      AEE  small_airport   Adareil Airport        1301.0          SS

          iso_region municipality gps_code iata_code local_code   latitude  \
continent
Africa         SO-GA        Adado      NaN       AAD        NaN  46.637500
Africa         SO-GA        Adado      NaN       AAD        NaN  46.637500
Africa         SD-DE     El Daein      NaN       ADV        NaN  26.118600
Africa         SD-DE     El Daein      NaN       ADV        NaN  26.118600
Africa         SS-23          NaN      NaN       AEE        NaN  32.959444

            altitude   pole
continent
Africa      6.095802  South
Africa      6.095802  North
Africa     11.402300  South
Africa     11.402300  North
Africa     10.053611  South
```

Таким образом можно дополнять и **обогащать** выборку данных новым информативным смыслом. На самом деле в реальных задачах это гораздо сложнее, чем в продемонстрированном примере. Основная мысль этой демонстрации — это sql подобный синтаксис работы с pandas DataFrame.

3.4. Изучение данных

Не всегда данные находятся сразу в csv. иногда их требуется предварительно распарсить, например, их html. Это можно сделать как представлено в примере:

```
import html5lib
telecom = pd.read_html('https://github.com/Yorko/mlcourse_open/blob/master/da
ta/telecom_churn.csv', header = 0)[0]
telecom.head()
```

```
   Unnamed: 0 State  Account length  Area code International plan  \
0         NaN    KS             128        415                 No
1         NaN    OH             107        415                 No
2         NaN    NJ             137        415                 No
3         NaN    OH              84        408                Yes
4         NaN    OK              75        415                Yes

  Voice mail plan  Number vmail messages  Total day minutes  Total day calls
\
0             Yes                     25              265.1              110
1             Yes                     26              161.6              123
2              No                      0              243.4              114
3              No                      0              299.4               71
4              No                      0              166.7              113

   Total day charge  Total eve minutes  Total eve calls  Total eve charge  \
0             45.07              197.4               99             16.78
```

```
1             27.47             195.5             103            16.62
2             41.38             121.2             110            10.30
3             50.90              61.9              88             5.26
4             28.34             148.3             122            12.61

   Total night minutes  Total night calls  Total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   Total intl minutes  Total intl calls  Total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   Customer service calls  Churn
0                       1  False
1                       1  False
2                       0  False
3                       2  False
4                       3  False

telecom.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3333 entries, 0 to 3332
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Unnamed: 0              0 non-null      float64
 1   State                   3333 non-null   object
 2   Account length          3333 non-null   int64
 3   Area code               3333 non-null   int64
 4   International plan      3333 non-null   object
 5   Voice mail plan         3333 non-null   object
 6   Number vmail messages   3333 non-null   int64
 7   Total day minutes       3333 non-null   float64
 8   Total day calls         3333 non-null   int64
 9   Total day charge        3333 non-null   float64
 10  Total eve minutes       3333 non-null   float64
 11  Total eve calls         3333 non-null   int64
 12  Total eve charge        3333 non-null   float64
 13  Total night minutes     3333 non-null   float64
 14  Total night calls       3333 non-null   int64
 15  Total night charge      3333 non-null   float64
 16  Total intl minutes      3333 non-null   float64
 17  Total intl calls        3333 non-null   int64
 18  Total intl charge       3333 non-null   float64
 19  Customer service calls  3333 non-null   int64
 20  Churn                   3333 non-null   bool
dtypes: bool(1), float64(9), int64(8), object(3)
memory usage: 524.2+ KB
```

bool, int64, float64 и object — это типы аттрибутов (колонок, или признаков). Видим, что 1 атрибут — логический (bool), 3 атрибута 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:

Отдельно замечу, что **признаки** (или переменные) могут обладать различной природой, а значит и *статистическим распределением*. Глобально отличия природы распределения переменных бывает двух типов: количественные и категориальные. Переменная, принимающая значения из некоторого ограниченного набора категорий, называется **категориальной**. Обычно связана с неисчисляемыми атрибутами, такими как названия (товаров, услуг и др.), имена людей, исходы событий (да/нет), пункты выбора в меню, тарифные планы, статусы и т.д. Синонимом может быть дискретный или качественный признак. Если категорий такое множество, что нам приходится задавать диапазон значений, попадающих в эту категорию, то принято считать распределение такой величины квазинепрерывным (в статистике случайные величины могут быть не только дискретными и непрерывными, а например, в ограниченной областью рассеивания). Для таких признаков говорят, что они **количественные**. Например, возраст, длина, высота, доход, задолженность и т.д.

```
telecom['Churn'] = telecom['Churn'].astype('int64')
```

```
telecom = telecom.drop(telecom.columns[0], axis=1) # уберем не информативную колонку
telecom.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
0	KS	128	415	No	Yes	25	265.1	110	45.07	197.4	99	16.78
1	OH	107	415	No	Yes	26	161.6	123	27.47	195.5	103	16.62
2	NJ	137	415	No	No	0	243.4	114	41.38	121.2	110	10.30
3	OH	84	408	Yes	No	0	299.4	71	50.90	61.9	88	5.26
4	OK	75	415	Yes	No	0	166.7	113	28.34	148.3	122	12.61

```
   Total night minutes  Total night calls  Total night charge  \
0                244.7                 91               11.01
1                254.4                103               11.45
2                162.6                104                7.32
3                196.9                 89                8.86
4                186.9                121                8.41

   Total intl minutes  Total intl calls  Total intl charge  \
0                10.0                 3               2.70
1                13.7                 3               3.70
2                12.2                 5               3.29
3                 6.6                 7               1.78
4                10.1                 3               2.73

   Customer service calls  Churn
0                       1      0
1                       1      0
2                       0      0
3                       2      0
4                       3      0
```

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.

Допущение о том, что распределение данных в признаке нормальное очень ненадёжное. Вы можете вызвать метод на категориальной переменной, в которой хранятся статусы, но это **первое приближение**. На самом деле после более тщательного анализа распределения данных в категориальной переменной стоит уточнить, что для неё понятия квартилей, среднего и дисперсии выдаваемые pandas не имеют никакого смысла.

```
telecom.describe()
```

```
       Account length    Area code  Number vmail messages  Total day minutes
\
count     3333.000000  3333.000000            3333.000000        3333.000000
mean       101.064806   437.182418               8.099010         179.775098
std         39.822106    42.371290              13.688365          54.467389
min          1.000000   408.000000               0.000000           0.000000
25%         74.000000   408.000000               0.000000         143.700000
50%        101.000000   415.000000               0.000000         179.400000
75%        127.000000   510.000000              20.000000         216.400000
max        243.000000   510.000000              51.000000         350.800000

       Total day calls  Total day charge  Total eve minutes  Total eve calls
\
count      3333.000000       3333.000000        3333.000000      3333.000000
mean        100.435644         30.562307         200.980348       100.114311
std          20.069084          9.259435          50.713844        19.922625
min           0.000000          0.000000           0.000000         0.000000
25%          87.000000         24.430000         166.600000        87.000000
50%         101.000000         30.500000         201.400000       100.000000
75%         114.000000         36.790000         235.300000       114.000000
```

```
max         165.000000         59.640000         363.700000       170.000000

       Total eve charge  Total night minutes  Total night calls  \
count       3333.000000          3333.000000        3333.000000
mean          17.083540           200.872037         100.107711
std            4.310668            50.573847          19.568609
min            0.000000            23.200000          33.000000
25%           14.160000           167.000000          87.000000
50%           17.120000           201.200000         100.000000
75%           20.000000           235.300000         113.000000
max           30.910000           395.000000         175.000000

       Total night charge  Total intl minutes  Total intl calls  \
count         3333.000000         3333.000000       3333.000000
mean             9.039325           10.237294          4.479448
std              2.275873            2.791840          2.461214
min              1.040000            0.000000          0.000000
25%              7.520000            8.500000          3.000000
50%              9.050000           10.300000          4.000000
75%             10.590000           12.100000          6.000000
max             17.770000           20.000000         20.000000

       Total intl charge  Customer service calls        Churn
count        3333.000000             3333.000000  3333.000000
mean            2.764581                1.562856     0.144914
std             0.753773                1.315491     0.352067
min             0.000000                0.000000     0.000000
25%             2.300000                1.000000     0.000000
50%             2.780000                1.000000     0.000000
75%             3.270000                2.000000     0.000000
max             5.400000                9.000000     1.000000
```

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

```
telecom.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:

```
telecom['Churn'].value_counts()
```

```
0    2850
1     483
Name: Churn, dtype: int64
```

3.5. Сортировка

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

```
telecom.sort_values(by='Total day charge', ascending=False).head()
```

```
     State  Account length  Area code International plan Voice mail plan  \
365     CO             154        415                 No              No
985     NY              64        415                Yes              No
2594    OH             115        510                Yes              No
156     OH              83        415                 No              No
605     MO             112        415                 No              No

      Number vmail messages  Total day minutes  Total day calls  \
365                       0              350.8               75
985                       0              346.8               55
2594                      0              345.3               81
156                       0              337.4              120
605                       0              335.5               77

      Total day charge  Total eve minutes  Total eve calls  Total eve charge
\
365              59.64              216.5               94             18.40
985              58.96              249.5               79             21.21
2594             58.70              203.4              106             17.29
156              57.36              227.4              116             19.33
605              57.04              212.5              109             18.06

      Total night minutes  Total night calls  Total night charge  \
365                 253.9                100               11.43
985                 275.4                102               12.39
2594                217.5                107                9.79
156                 153.9                114                6.93
605                 265.0                132               11.93

      Total intl minutes  Total intl calls  Total intl charge  \
365                 10.1                 9               2.73
985                 13.3                 9               3.59
2594                11.8                 8               3.19
156                 15.8                 7               4.27
605                 12.7                 8               3.43

      Customer service calls  Churn
365                        1      1
985                        1      1
2594                       1      1
156                        0      1
605                        2      1
```

Сортировать можно и по группе столбцов:

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

```
     State  Account length  Area code International plan Voice mail plan  \
688     MN              13        510                 No             Yes
2259    NC             210        415                 No             Yes
534     LA              67        510                 No              No
575     SD             114        415                 No             Yes
2858    AL             141        510                 No             Yes

      Number vmail messages  Total day minutes  Total day calls  \
688                      21              315.6              105
2259                     31              313.8               87
534                       0              310.4               97
575                      36              309.9               90
2858                     28              308.0              123

      Total day charge  Total eve minutes  Total eve calls  Total eve charge
\
688              53.65              208.9               71             17.76
2259             53.35              147.7              103             12.55
534              52.77               66.5              123              5.65
575              52.68              200.3               89             17.03
2858             52.36              247.8              128             21.06

      Total night minutes  Total night calls  Total night charge  \
688                 260.1                123               11.70
2259                192.7                 97                8.67
534                 246.5                 99               11.09
575                 183.5                105                8.26
2858                152.9                103                6.88

      Total intl minutes  Total intl calls  Total intl charge  \
688                 12.1                 3               3.27
2259                10.1                 7               2.73
534                  9.2                10               2.48
575                 14.2                 2               3.83
2858                 7.4                 3               2.00

      Customer service calls  Churn
688                        3      0
2259                       3      0
534                        4      0
575                        1      0
2858                       1      0
```

3.6. 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 min
utes']

telecom.groupby(['Churn'])[columns_to_show].describe(percentiles=[])
```

```
      Total day minutes                                              \
                  count        mean        std  min    50%    max
Churn
0                2850.0  175.175754  50.181655  0.0  177.2  315.6
1                 483.0  206.914079  68.997792  0.0  217.6  350.8

      Total eve minutes                                              \
                  count        mean        std   min    50%    max
Churn
0                2850.0  199.043298  50.292175   0.0  199.6  361.8
1                 483.0  212.410145  51.728910  70.9  211.3  363.7

      Total night minutes
                   count        mean        std   min     50%    max
Churn
0                 2850.0  200.133193  51.105032  23.2  200.25  395.0
1                  483.0  205.231677  47.132825  47.4  204.80  354.9
```

```
columns_to_show = ['Total day minutes', 'Total eve minutes', 'Total night min
utes']

telecom.groupby(['State'])[columns_to_show].describe(percentiles=[])
```

```
      Total day minutes                                                \
                  count        mean        std   min     50%    max
State
AK                 52.0  178.384615  49.640430  58.2  177.25  278.4
AL                 80.0  186.010000  51.466249  68.7  190.25  308.0
AR                 55.0  176.116364  50.368831  55.3  170.70  273.4
AZ                 64.0  171.604688  51.941907  58.9  171.45  281.1
CA                 34.0  183.564706  47.742484  92.8  183.20  280.0
CO                 66.0  178.712121  59.805856  30.9  180.90  350.8
CT                 74.0  175.140541  60.523424  37.8  176.50  321.6
DC                 54.0  171.379630  57.157338  51.5  169.20  306.2
DE                 61.0  174.583607  52.060645  46.5  179.90  334.3
FL                 63.0  179.533333  57.468499  47.7  181.80  288.1
GA                 54.0  185.025926  53.736275  71.2  193.30  299.5
HI                 53.0  175.962264  54.834311  41.9  181.40  291.6
IA                 44.0  177.613636  48.400925  88.1  168.80  308.6
ID                 73.0  178.619178  52.794622  55.6  181.60  274.4
IL                 58.0  173.591379  49.802932  69.1  180.35  269.6
IN                 71.0  196.525352  51.956157  49.9  203.80  300.4
KS                 70.0  191.555714  58.143148  27.0  191.25  321.3
KY                 59.0  173.754237  54.943583  73.8  170.50  314.6
LA                 51.0  178.376471  45.435139  58.4  179.30  310.4
MA                 65.0  180.103077  51.288790  58.9  178.10  293.7
MD                 70.0  197.228571  58.031576  78.1  198.15  321.1
ME                 62.0  185.262903  52.707427  58.8  193.80  322.3
MI                 73.0  180.593151  54.873206  18.9  185.30  314.1
```

MN	84.0	183.354762	56.625260	50.6	178.60	317.8
MO	63.0	170.506349	58.076573	45.0	165.90	335.5
MS	65.0	177.929231	61.631895	70.7	166.50	313.2
MT	68.0	174.007353	48.848851	89.8	162.30	273.2
NC	68.0	185.145588	56.222470	54.7	189.80	322.3
ND	62.0	187.338710	45.251481	82.5	191.25	295.3
NE	61.0	177.465574	52.599645	34.0	180.90	272.7
NH	56.0	177.328571	59.963106	17.6	182.85	322.4
NJ	68.0	196.225000	48.608661	40.9	193.05	301.5
NM	62.0	171.429032	44.931695	69.1	169.05	286.7
NV	66.0	176.425758	56.561785	67.4	168.70	303.9
NY	83.0	175.114458	56.786981	60.6	166.40	346.8
OH	78.0	183.274359	55.755483	7.8	185.60	345.3
OK	61.0	179.909836	61.730340	2.6	179.20	329.8
OR	78.0	176.246154	56.013219	12.5	186.40	324.7
PA	45.0	188.375556	55.137556	35.1	205.10	288.7
RI	65.0	167.478462	55.418410	40.4	167.80	286.2
SC	60.0	166.441667	63.585043	19.5	157.75	322.5
SD	60.0	189.690000	55.366666	0.0	186.55	328.1
TN	53.0	175.771698	50.608282	54.8	170.10	305.2
TX	72.0	181.516667	57.146528	59.5	181.30	326.5
UT	72.0	183.569444	53.796365	63.2	187.40	285.7
VA	77.0	177.244156	49.273203	44.9	174.50	283.4
VT	73.0	182.031507	52.048788	0.0	188.40	307.1
WA	66.0	178.742424	56.412167	37.7	166.40	289.1
WI	78.0	179.130769	57.867821	7.9	177.25	326.3
WV	106.0	173.950943	53.920065	58.0	176.25	312.0
WY	77.0	180.170130	53.975375	25.9	178.30	296.0

	Total eve minutes						\
	count	mean	std	min	50%	max	
State							
AK	52.0	184.282692	49.160213	58.6	179.95	314.4	
AL	80.0	195.462500	50.909648	77.9	201.25	299.9	
AR	55.0	201.047273	50.957484	120.5	192.30	350.9	
AZ	64.0	187.748438	49.070513	72.9	191.10	328.7	
CA	34.0	198.970588	40.361770	114.0	195.45	281.3	
CO	66.0	206.884848	53.306802	75.3	211.20	341.3	
CT	74.0	203.828378	55.971033	66.0	209.85	335.0	
DC	54.0	196.272222	47.422479	65.2	198.10	337.1	
DE	61.0	208.247541	46.393479	132.5	206.60	328.2	
FL	63.0	210.276190	54.921782	69.2	209.00	318.8	
GA	54.0	204.140741	47.011053	73.2	199.95	304.4	
HI	53.0	191.343396	50.612090	90.0	192.40	305.8	
IA	44.0	206.400000	55.424480	102.2	200.45	329.3	
ID	73.0	194.610959	44.761049	98.3	195.70	292.7	
IL	58.0	196.798276	52.539016	48.1	198.00	319.3	
IN	71.0	202.559155	55.354715	83.9	199.40	361.8	
KS	70.0	202.512857	48.809158	95.1	197.35	310.6	
KY	59.0	196.244068	50.654953	87.6	195.70	324.8	
LA	51.0	197.819608	57.194972	31.2	205.10	351.6	
MA	65.0	214.664615	52.439425	95.6	211.40	348.5	
MD	70.0	196.061429	50.489311	90.2	194.15	354.2	
ME	62.0	200.514516	44.995424	79.3	206.60	278.2	
MI	73.0	208.172603	51.870258	89.1	208.60	336.0	
MN	84.0	199.334524	46.627874	42.5	203.05	322.2	

MO	63.0	200.141270	57.256014	60.8	208.60	332.1
MS	65.0	200.009231	50.163423	103.2	202.80	313.7
MT	68.0	201.526471	52.860757	88.1	210.65	312.6
NC	68.0	202.536765	47.720895	80.8	204.10	363.7
ND	62.0	207.775806	45.635307	89.3	209.85	289.9
NE	61.0	203.111475	54.304328	56.0	206.70	282.6
NH	56.0	198.158929	51.762225	58.9	201.40	303.2
NJ	68.0	198.289706	53.385299	88.3	198.10	303.8
NM	62.0	212.193548	46.616795	122.9	211.40	327.1
NV	66.0	202.915152	50.859687	106.1	196.45	339.9
NY	83.0	196.993976	58.252500	64.3	193.00	332.8
OH	78.0	206.441026	49.864079	61.9	210.50	301.5
OK	61.0	193.018033	57.107987	42.2	195.00	317.0
OR	78.0	201.496154	47.566428	75.9	205.85	317.5
PA	45.0	191.653333	45.323271	101.5	186.60	294.3
RI	65.0	211.038462	49.939531	102.2	207.20	350.5
SC	60.0	207.456667	49.462711	83.4	213.90	304.6
SD	60.0	202.723333	45.108319	118.7	193.15	295.7
TN	53.0	210.513208	45.766754	93.4	206.90	296.5
TX	72.0	199.787500	51.865041	49.2	194.80	327.0
UT	72.0	195.343056	51.600630	0.0	197.90	319.0
VA	77.0	204.216883	44.592478	91.2	205.20	313.4
VT	73.0	205.368493	48.275120	71.0	209.90	304.9
WA	66.0	203.810606	49.061353	52.9	203.45	285.9
WI	78.0	197.458974	53.884652	97.7	193.80	322.3
WV	106.0	188.413208	52.769371	67.0	185.60	315.4
WY	77.0	205.828571	52.124729	60.0	212.80	330.6

	Total night minutes					
	count	mean	std	min	50%	max
State						
AK	52.0	192.326923	53.663297	23.2	200.50	303.5
AL	80.0	187.285000	42.355416	94.1	185.40	287.6
AR	55.0	205.454545	56.696498	96.4	205.50	367.7
AZ	64.0	194.004687	54.387325	77.3	191.40	297.9
CA	34.0	198.508824	58.502574	71.1	204.90	345.8
CO	66.0	189.898485	50.131426	83.9	193.45	308.2
CT	74.0	205.997297	59.124952	63.6	205.95	325.6
DC	54.0	206.348148	48.437521	95.3	210.85	302.0
DE	61.0	203.900000	43.651063	111.7	200.90	313.2
FL	63.0	196.147619	50.501983	73.2	197.40	286.3
GA	54.0	193.746296	56.564876	65.8	186.15	364.9
HI	53.0	203.713208	43.058596	112.3	203.30	318.3
IA	44.0	191.490909	48.004466	43.7	189.90	271.8
ID	73.0	202.595890	49.006208	89.3	200.40	377.5
IL	58.0	197.605172	46.280500	67.7	192.70	281.9
IN	71.0	210.242254	52.449659	99.0	213.40	350.2
KS	70.0	203.970000	46.138045	50.1	208.95	289.9
KY	59.0	198.355932	53.771045	79.9	202.10	314.1
LA	51.0	201.396078	51.633872	103.7	196.00	352.2
MA	65.0	204.007692	55.335701	72.4	204.80	332.7
MD	70.0	198.614286	51.707685	76.4	194.00	294.8
ME	62.0	198.833871	48.369012	114.2	193.85	313.4
MI	73.0	192.657534	47.490048	63.3	195.20	302.2
MN	84.0	209.680952	45.807222	77.2	201.30	296.3
MO	63.0	209.146032	54.135820	50.1	210.00	349.7

MS	65.0	200.996923	41.684953	112.9	203.40	309.6
MT	68.0	197.205882	45.569900	87.4	194.85	328.5
NC	68.0	197.144118	51.714106	54.5	198.40	329.3
ND	62.0	199.796774	53.604532	61.4	207.00	281.3
NE	61.0	206.427869	60.281739	82.4	211.60	381.9
NH	56.0	208.614286	48.423373	107.3	214.00	334.7
NJ	68.0	206.383824	51.385458	57.5	210.65	309.1
NM	62.0	200.193548	53.324426	82.3	209.00	310.1
NV	66.0	208.645455	43.134524	126.3	206.55	320.7
NY	83.0	203.268675	47.042381	94.9	200.80	323.5
OH	78.0	204.491026	55.067958	64.2	204.90	352.5
OK	61.0	196.947541	50.258414	47.4	192.50	312.1
OR	78.0	199.925641	51.042301	53.3	199.55	322.2
PA	45.0	195.864444	53.891090	56.6	186.90	342.8
RI	65.0	204.052308	56.367225	91.6	200.60	315.0
SC	60.0	195.136667	46.713639	73.2	186.00	325.9
SD	60.0	201.310000	45.437361	116.3	198.45	303.5
TN	53.0	210.426415	51.282487	104.7	219.50	317.8
TX	72.0	195.288889	40.665577	115.9	193.10	306.6
UT	72.0	190.519444	51.344801	90.9	188.20	344.3
VA	77.0	212.963636	57.260802	54.0	214.70	395.0
VT	73.0	206.989041	48.835198	78.1	205.70	333.5
WA	66.0	200.045455	52.908182	84.8	203.95	304.2
WI	78.0	199.229487	56.392127	77.9	192.55	364.3
WV	106.0	201.055660	50.275578	75.8	202.55	326.4
WY	77.0	199.167532	45.849655	45.0	203.30	285.3

4. Поиск и устранение ошибок в данных. Заполнение недостающих данных

4.1. Поиск ошибок в данных

Очистка данных включает различные методы, основанные на проблемах и типах данных. Различные методы могут быть применены с каждым атрибутом данных, но имеют свои собственные особенности применения.

В целом, неверные данные либо удаляются, исправляются, либо присваиваются.

Нерелевантные данные

Нерелевантные данные — это те, которые на самом деле не нужны и не вписываются в контекст проблемы, которую мы пытаемся решить.

Например, если бы мы анализировали данные об общем состоянии здоровья населения, номер телефона не был бы необходим.

Точно так же, если бы вы были заинтересованы только в одной конкретной стране, вы не хотели бы включать в набор данных все другие страны. Или изучать только тех пациентов, которые ходили на операцию, мы бы не включали всех пациентов клиники.

Только если вы уверены, что часть данных не важна, вы можете удалить ее. В противном случае изучите матрицу корреляции между атрибутами объекта.

И даже если вы не заметили никакой корреляции, вы должны спросить кого-то, кто является экспертом в области. Вы никогда не знаете, что функция,

которая кажется неактуальной, может быть очень актуальной с точки зрения предметной области, например с клинической точки зрения.

Дубликаты

Дубликаты — это значения, которые повторяются в вашем наборе данных (повторяться могут как транзакции, так и сущности в различных справочниках).

Дублирование часто случается, когда, например,

Данные объединены из разных источников

Пользователь может дважды нажать кнопку «Отправить», думая, что форма фактически не была отправлена.

Запрос на онлайн-бронирование был подан дважды, исправляя неверную информацию, которая была введена случайно в первый раз.

Распространенным симптомом является случай, когда два пользователя имеют одинаковый идентификационный номер. Или одна и та же статья была отменена дважды.

И поэтому они просто должны быть удалены.

Преобразования типа

Убедитесь, что числа хранятся в виде числовых типов данных. Дата должна храниться в виде объекта даты или метки времени Unix (количество секунд) и т.д.

Категориальные значения могут быть преобразованы в/из чисел при необходимости.

Это можно быстро определить, взглянув на типы данных каждого столбца в сводке (мы обсуждали выше).

Предупреждение: значения, которые нельзя преобразовать в указанный тип, следует преобразовать в значение NA (или любое другое) с отображением предупреждения. Это указывает на неправильное значение, которое должно быть исправлено.

Синтаксические ошибки

Удалить пробелы: следует удалить лишние пробелы в начале или конце строки.

```
"   hello world  " => "hello world"
```

Строки заполнения: строки могут быть дополнены пробелами или другими символами до определенной ширины. Например, некоторые числовые коды часто представлены с предшествующими нулями, чтобы они всегда имели одинаковое количество цифр.

```
313 => 000313 (6 digits)
```

Исправьте опечатки: строки могут быть введены разными способами, и неудивительно, что могут быть ошибки.

```
Gender

m

Male
```

```
fem.

FemalE

Femle
```

Считается, что эта категориальная переменная имеет 5 разных классов, а не 2, как ожидалось: мужской и женский, поскольку каждое значение различно.

Отсутствующие данные

Работа с отсутствующими значениями – одна из самых сложных, но и самых распространенных проблем очистки. Большинство моделей не предполагают пропусков. Рассмотрим три метода обнаружения отсутствующих данных в наборе.

Тепловая карта пропущенных значений

Когда признаков в наборе не очень много, визуализируйте пропущенные значения с помощью тепловой карты.

```
cols = df.columns[:30] # первые 30 колонок
# определяем цвета
# желтый - пропущенные данные, синий - не пропущенные
colours = ['#000099', '#ffff00']
sns.heatmap(df[cols].isnull(),
cmap=sns.color_palette(colours))
```

Приведенная ниже карта демонстрирует паттерн пропущенных значений для первых 30 признаков набора. По горизонтальной оси расположены признаки, по вертикальной – количество записей/строк. Желтый цвет соответствует пропускам данных.

Заметно, например, что признак life_sq имеет довольно много пустых строк, а признак floor – напротив, всего парочку – около 7000 строки.

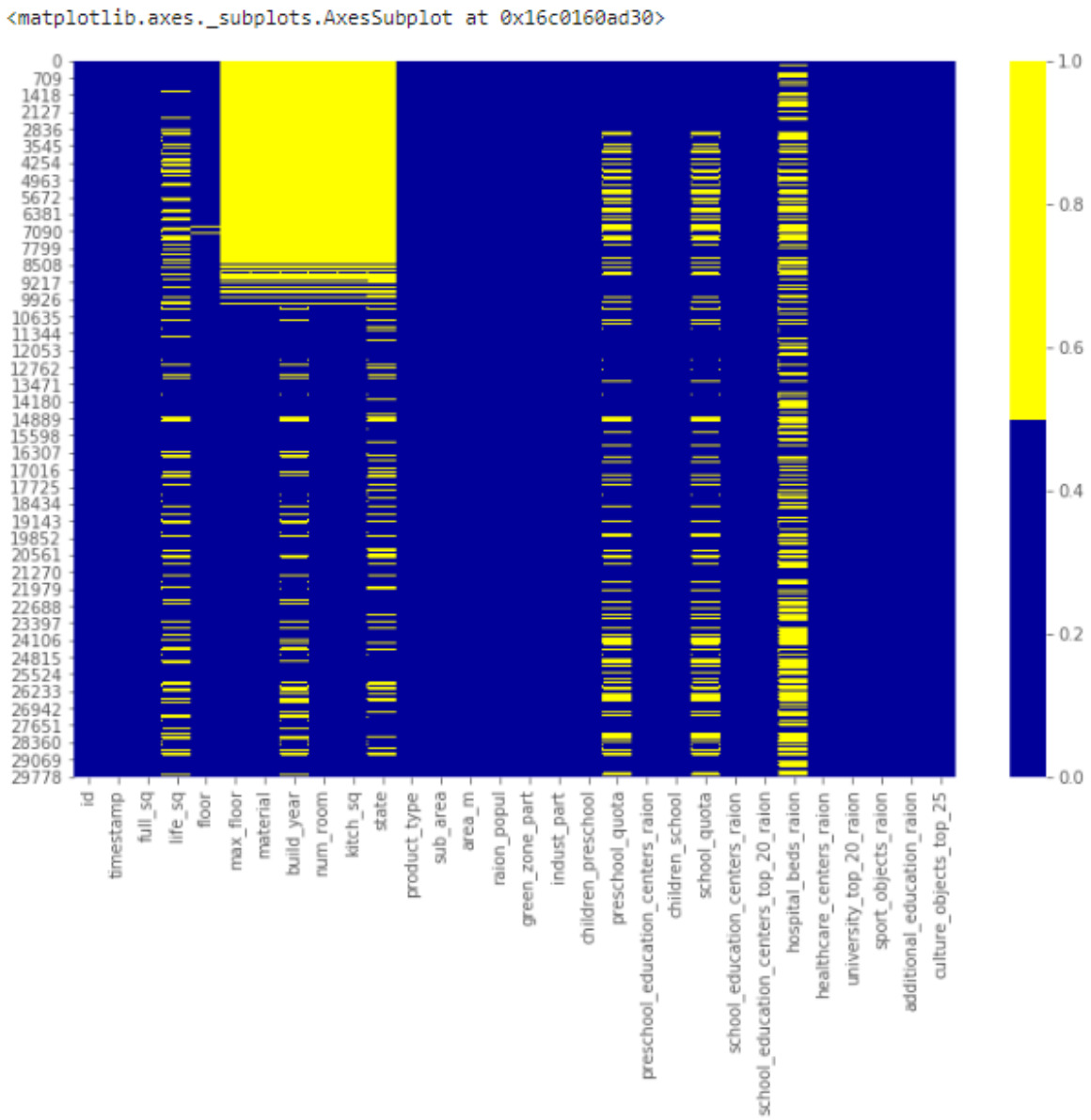

Рис.9 Тепловая карта признаков

Процентный список пропущенных данных

Если в наборе много признаков и визуализация занимает много времени, можно составить список долей отсутствующих записей для каждого признака.

```
for col in df.columns:
    pct_missing = np.mean(df[col].isnull())
    print('{}                    -                    {}%'.format(col,
round(pct_missing*100)))
```

Такой список для тех же 30 первых признаков выглядит следующим образом:

У признака life_sq отсутствует 21% значений, а у floor – только 1%.

Этот список является полезным резюме, которое может отлично дополнить визуализацию тепловой карты.

Гистограмма пропущенных данных

Еще одна хорошая техника визуализации для наборов с большим количеством признаков – построение гистограммы для числа отсутствующих значений в записи.

```
# сначала создаем индикатор для признаков с пропущенными данными
for col in df.columns:
    missing = df[col].isnull()
    num_missing = np.sum(missing)

    if num_missing > 0:
        print('created missing indicator for: {}'.format(col))
        df['{}_ismissing'.format(col)] = missing


# затем на основе индикатора строим гистограмму
ismissing_cols = [col for col in df.columns if 'ismissing' in col]
df['num_missing'] = df[ismissing_cols].sum(axis=1)

df['num_missing'].value_counts().reset_index().sort_values(by='index').plot.bar(x='index', y='num_missing')
```

Отсюда понятно, что из 30 тыс. записей более 6 тыс. строк не имеют ни одного пропущенного значения, а еще около 4 тыс.– всего одно. Такие строки можно использовать в качестве «эталонных» для проверки различных гипотез по дополнению данных.

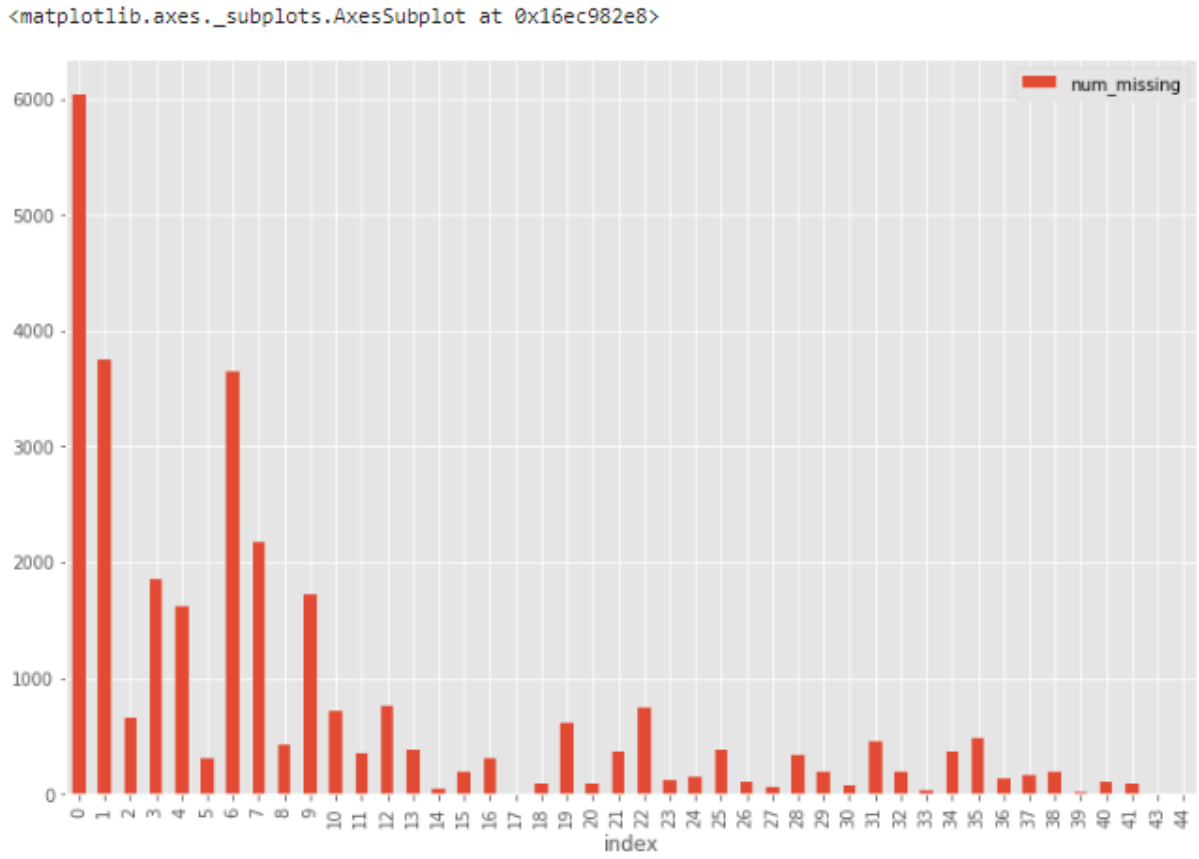

Рис.10 Гистограмма признаков

4.2. Очистка данных

Не существует общих решений для проблемы отсутствующих данных. Для каждого конкретного набора приходится искать наиболее подходящие методы или их комбинации.

Разберем четыре самых распространенных техники. Они помогут в простых ситуациях, но, скорее всего, придется проявить творческий подход и поискать нетривиальные решения, например, промоделировать пропуски.

Отбрасывание записей

Первая техника в статистике называется методом удаления по списку и заключается в простом отбрасывании записи, содержащей пропущенные значения. Это решение подходит только в том случае, если недостающие данные не являются информативными.

Для отбрасывания можно использовать и другие критерии. Например, из гистограммы, построенной в предыдущем разделе, мы узнали, что лишь небольшое количество строк содержат более 35 пропусков. Мы можем создать новый набор данных df_less_missing_rows, в котором отбросим эти строки.

```
# отбрасываем строки с большим количеством пропусков
ind_missing = df[df['num_missing'] > 35].index
df_less_missing_rows = df.drop(ind_missing, axis=0)
```

Отбрасывание признаков

Как и предыдущая техника, отбрасывание признаков может применяться только для неинформативных признаков.

В процентном списке, построенном ранее, мы увидели, что признак hospital_beds_raion имеет высокий процент недостающих значений – 47%. Мы можем полностью отказаться от этого признака:

```
cols_to_drop = ['hospital_beds_raion']
df_less_hos_beds_raion = df.drop(cols_to_drop, axis=1)
```

Внесение недостающих значений

Для численных признаков можно воспользоваться методом принудительного заполнения пропусков. Например, на место пропуска можно записать среднее или медианное значение, полученное из остальных записей.

Для категориальных признаков можно использовать в качестве заполнителя наиболее часто встречающееся значение.

Возьмем для примера признак life_sq и заменим все недостающие значения медианой этого признака:

```
med = df['life_sq'].median()
print(med)
df['life_sq'] = df['life_sq'].fillna(med)
```

Одну и ту же стратегию принудительного заполнения можно применить сразу для всех числовых признаков:

```
# impute the missing values and create the missing value
indicator variables for each numeric column.
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values

for col in numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)

    if num_missing > 0:  # only do the imputation for the
columns that have missing values.
        print('imputing     missing     values     for:
{}'.format(col))
        df['{}_ismissing'.format(col)] = missing
        med = df[col].median()
        df[col] = df[col].fillna(med)
```

К счастью, в нашем наборе не нашлось пропусков в категориальных признаках. Но это не мешает нам продемонстрировать использование той же стратегии:

```
df_non_numeric = df.select_dtypes(exclude=[np.number])
```

```
non_numeric_cols = df_non_numeric.columns.values

for col in non_numeric_cols:
    missing = df[col].isnull()
    num_missing = np.sum(missing)

    if num_missing > 0:  # only do the imputation for the
columns that have missing values.
        print('imputing     missing     values     for:
{}'.format(col))
        df['{}_ismissing'.format(col)] = missing

        top = df[col].describe()['top'] # impute with the
most frequent value.
        df[col] = df[col].fillna(top)
```

Замена недостающих значений

Можно использовать некоторый дефолтный плейсхолдер для пропусков, например, новую категорию _MISSING_ для категориальных признаков или число -999 для числовых.

Таким образом, мы сохраняем данные о пропущенных значениях, что тоже может быть ценной информацией.

```
# категориальные признаки
df['sub_area'] = df['sub_area'].fillna('_MISSING_')

# численные признаки
df['life_sq'] = df['life_sq'].fillna(-999)
```

4.3. Заполнение недостающих данных в pandas DataFrame

Метод apply можно использовать и для того, чтобы применить функцию к каждой строке. Для этого нужно указать axis=0. Колонки **continent** и **iata_code** заполнились не очень хорошо. Применим функции к каждой ячейке столбца с помощью fillna или replace для np.nan:

```
airports['continent']        =        airports['continent'].fillna('')
airports['iata_code']   =   airports['iata_code'].replace(np.nan,   '0')
airports.head()
```

```
  ident          type                                name  elevation_f
t                                                        \
0   00A       heliport                   Total Rf Heliport          11.
0
1  00AA  small_airport                Aero B Ranch Airport        3435.
0
2  00AK  small_airport                        Lowell Field         450.
0
```

```
3  00AL  small_airport                        Epps Airpark         820.
0
4  00AR         closed  Newport Hospital & Clinic Heliport         237.
0

  continent iso_country iso_region  municipality gps_code iata_code  \
0                    US       US-PA       Bensalem       00A          0
1                    US       US-KS          Leoti      00AA          0
2                    US       US-AK  Anchor Point      00AK          0
3                    US       US-AL        Harvest      00AL          0
4                    US       US-AR        Newport       NaN          0

  local_code                          latitude                 altitude
0                                 00A        -74.933601        40.070801
1                                 00AA     -101.473911          38.704022
2                                 00AK     -151.695999          59.949200
3                               00AL         -86.770302          34.864799
4        NaN  -91.254898  35.608700
```

Для того, чтобы понять какие бывают уникальные значения колонки можно воспользоваться методом unique

```
airports['continent'].unique()
```

```
array(['', 'OC', 'AF', 'AN', 'EU', 'AS', 'SA'], dtype=object)
```

Иногда поле требуется перекодировать. Для этого можно воспользоваться словарём и методом map. Для подсчёта частоты встречаемости различных значений можно воспользоваться методом value_counts.

```
d = {'SA':'South America', 'EU':'Europe', 'AS' : 'Asia','AF':'Africa', 'OC':'
Oceania', 'AN':'Antarctida', '':'Unknown'}
airports['continent'] = airports['continent'].map(d)
airports['continent'].value_counts()
```

```
Unknown          28443
South America     8443
Europe            8404
Asia              5619
Africa            3361
Oceania           3123
Antarctida          28
Name: continent, dtype: int64
```

5. Форматы данных

5.1. Источники данных поддерживаемые в Pandas

CSV (от англ. Comma-Separated Values — значения, разделённые запятыми) — текстовый формат, предназначенный для представления табличных данных. Строка таблицы соответствует строке текста, которая содержит одно или несколько полей, разделенных запятыми.

Формат CSV стандартизирован не полностью. Идея использовать запятые для разделения полей очевидна, но при таком подходе возникают проблемы, если исходные табличные данные содержат запятые или переводы строк. Возможным решением проблемы запятых и переносов строк является заключение данных в кавычки, однако исходные данные могут содержать кавычки. Помимо этого, термином «CSV» могут обозначаться похожие форматы, в которых разделителем является символ табуляции (TSV) или точка с запятой. Многие приложения, которые работают с форматом CSV, позволяют выбирать символ разделителя и символ кавычек. Существует RFC 4180, предназначенный для стандартизации и упрощения обмена данными в формате CSV.

Помимо csv, текстовых файлов и html Pandas поддерживает большое количество различных источников:

- json

  JSON (англ. JavaScript Object Notation) — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом.

  Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

```
pd.read_json('sample_data/anscombe.json').head(10)
```

	Series	X	Y
0	I	10	8.04
1	I	8	6.95
2	I	13	7.58
3	I	9	8.81
4	I	11	8.33
5	I	14	9.96
6	I	6	7.24
7	I	4	4.26

- excel

  В Microsoft Excel вплоть до 2003 версии включительно использовался свой собственный бинарный формат файлов (BIFF) в качестве основного. Excel 2007 использует Microsoft Office Open XML в качестве своего основного формата.

  Несмотря на то, что Excel 2007 поддерживает и направлен на использование новых XML-форматов в качестве основных, он по-прежнему совместим с традиционными бинарными форматами.

- sql
  SQL декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционной базе данных, управляемой соответствующей системой управления базами данных.
  Является, прежде всего, информационно-логическим языком, предназначенным для описания, изменения и извлечения данных, хранимых в реляционных базах данных. В общем случае SQL (без ряда современных расширений) считается языком программирования не полным по Тьюрингу, но вместе с тем стандарт языка спецификацией SQL/PSM предусматривает возможность его процедурных расширений.
- parquet
  Apache Parquet — это бинарный, колоночно-ориентированный формат хранения больших данных, изначально созданный для экосистемы Hadoop, позволяющий использовать преимущества сжатого и эффективного колоночно-ориентированного представления информации. Паркет позволяет задавать схемы сжатия на уровне столбцов и добавлять новые кодировки по мере их появления. Вместе с Apache Avro, Parquet является очень популярным форматом хранения файлов Big Data и часто используется в Kafka, Spark и Hadoop.
- google-bigquery
  BigQuery — это бессерверное, масштабируемое облачное хранилище данных с мощной инфраструктурой от Google, которое имеет на борту RESTful веб-сервис. Имеет тесное взаимодействие с другими сервисами от Google. Создатели обещают молниеносное выполнение запросов с максимальной задержкой в RESTful до 1 секунды. BigQuery поддерживает диалект Standard SQL. Имеется возможность контроля доступа к данным и разграничение прав пользователей. Также есть возможность задавать квоты и лимиты для операций с БД. Доступ к BigQuery возможен через Google Cloud Console, с помощью внутренней консоли BigQuery, а также через вызовы BigQuery REST API как напрямую, так и через различные клиентские библиотеки java, python, .net и многие другие.

5.3. Чтение из SQL баз данных

Загрузить данные из SQL базы можно с помощью функции pd.read_sql. read_sql автоматически преобразует столбцы SQL в столбцы DataFrame.

read_sql принимает 2 аргумента: запрос SELECT, и connection. Это здорово, так как это означает, что можно читать из любого вида базы данных - неважно, MySQL, SQLite, PostgreSQL, или другая.

В этом примере мы читаем из базы SQLite, но другие читаются точно также. БД возьмём из официального руководства.

```
#!wget 'https://github.com/jvns/pandas-cookbook/raw/master/data/weather_2012.sqlite' -O /content/sample_data/weather_2012.sqlite -q
```

```
!wget https://www.sqlitetutorial.net/wp-content/uploads/2018/03/chinook.zip -
O 'sample_data/chinook.zip' -q
!unzip "sample_data/chinook.zip" -d "sample_data/"

Archive:  sample_data/chinook.zip
  inflating: sample_data/chinook.db
```

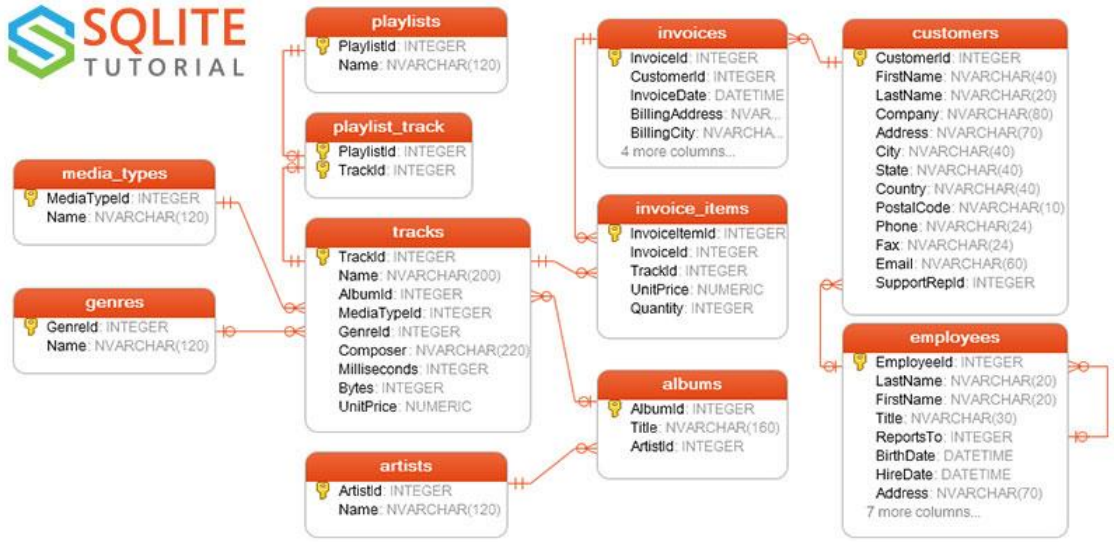

Рис.11 Схема данных РСУБД

```
import pandas as pd
from sqlalchemy import create_engine

engine = create_engine("sqlite:///sample_data/chinook.db")

with engine.connect() as conn, conn.begin():
    employees = pd.read_sql_table("employees", conn)
    customers = pd.read_sql("SELECT * FROM customers LIMIT 20", conn)
employees.head()
```

```
   EmployeeId LastName FirstName                Title  ReportsTo  BirthDate
\
0           1    Adams    Andrew      General Manager        NaN 1962-02-18
1           2  Edwards     Nancy        Sales Manager        1.0 1958-12-08
2           3  Peacock      Jane  Sales Support Agent        2.0 1973-08-29
3           4     Park  Margaret  Sales Support Agent        2.0 1947-09-19
4           5  Johnson     Steve  Sales Support Agent        2.0 1965-03-03

    HireDate              Address      City State Country PostalCode  \
0 2002-08-14  11120 Jasper Ave NW  Edmonton    AB  Canada     T5K 2N1
1 2002-05-01         825 8 Ave SW   Calgary    AB  Canada     T2P 2T3
2 2002-04-01        1111 6 Ave SW   Calgary    AB  Canada     T2P 5M5
3 2003-05-03     683 10 Street SW   Calgary    AB  Canada     T2P 5G3
4 2003-10-17         7727B 41 Ave   Calgary    AB  Canada     T3B 1Y7

                Phone                 Fax                   Email
0  +1 (780) 428-9482  +1 (780) 428-3457  andrew@chinookcorp.com
1  +1 (403) 262-3443  +1 (403) 262-3322   nancy@chinookcorp.com
```

```
2  +1 (403) 262-3443  +1 (403) 262-6712      jane@chinookcorp.com
3  +1 (403) 263-4423  +1 (403) 263-4289  margaret@chinookcorp.com
4   1 (780) 836-9987   1 (780) 836-9543     steve@chinookcorp.com
```

```
customers.head()
```

```
   CustomerId  FirstName     LastName  \
0           1       Luís    Gonçalves
1           2     Leonie       Köhler
2           3   François     Tremblay
3           4      Bjørn       Hansen
4           5  František  Wichterlová

                                        Company  \
0  Embraer - Empresa Brasileira de Aeronáutica S.A.
1                                              None
2                                              None
3                                              None
4                                 JetBrains s.r.o.

                          Address                 City State         Country
\
0  Av. Brigadeiro Faria Lima, 2170  São José dos Campos    SP          Brazil
1          Theodor-Heuss-Straße 34            Stuttgart  None         Germany
2                1498 rue Bélanger             Montréal    QC          Canada
3                 Ullevålsveien 14                 Oslo  None          Norway
4                    Klanova 9/506               Prague  None  Czech Republic

  PostalCode               Phone                 Fax  \
0  12227-000  +55 (12) 3923-5555  +55 (12) 3923-5566
1      70174    +49 0711 2842222                None
2    H2G 1A7   +1 (514) 721-4711                None
3       0171     +47 22 44 22 22                None
4      14700    +420 2 4172 5555    +420 2 4172 5555

                     Email  SupportRepId
0      luisg@embraer.com.br             3
1     leonekohler@surfeu.de             5
2       ftremblay@gmail.com             3
3     bjorn.hansen@yahoo.no             4
4  frantisekw@jetbrains.com             4
```

read_sql не устанавливает первичный ключ (id) в качестве индекса. Можно это сделать вручную, добавив аргумент index_col к read_sql.

Если вы много использовали read_csv, вы могли заметить, что у него также есть аргумент index_col. И ведёт он себя точно также.

```
with engine.connect() as conn, conn.begin():
    employees = pd.read_sql_table("employees", conn, index_col='EmployeeId')
    customers = pd.read_sql("SELECT * FROM customers LIMIT 20", conn, index_c
ol='SupportRepId')
customers.head()
```

```
              CustomerId  FirstName     LastName  \
SupportRepId
3                      1       Luís    Gonçalves
```

```
5                     2     Leonie       Köhler
3                     3   François     Tremblay
4                     4      Bjørn       Hansen
4                     5  František  Wichterlová

                                                    Company  \
SupportRepId
3             Embraer - Empresa Brasileira de Aeronáutica S.A.
5                                                       None
3                                                       None
4                                                       None
4                                            JetBrains s.r.o.

                                    Address                 City State  \
SupportRepId
3             Av. Brigadeiro Faria Lima, 2170  São José dos Campos    SP
5                    Theodor-Heuss-Straße 34            Stuttgart  None
3                          1498 rue Bélanger             Montréal    QC
4                           Ullevålsveien 14                 Oslo  None
4                              Klanova 9/506               Prague  None

                     Country PostalCode               Phone  \
SupportRepId
3                     Brazil  12227-000  +55 (12) 3923-5555
5                    Germany      70174    +49 0711 2842222
3                     Canada    H2G 1A7   +1 (514) 721-4711
4                     Norway       0171     +47 22 44 22 22
4             Czech Republic      14700    +420 2 4172 5555

                           Fax                      Email
SupportRepId
3             +55 (12) 3923-5566      luisg@embraer.com.br
5                           None     leonekohler@surfeu.de
3                           None       ftremblay@gmail.com
4                           None     bjorn.hansen@yahoo.no
4               +420 2 4172 5555  frantisekw@jetbrains.com
```

5.4. Запись в базу

Поддерживается использование sqlite без использования SQLAlchemy. Для этого режима требуется адаптер базы данных Python, который поддерживает Python DB-API. Вы можете создавать такие подключения: Запись производится с помощью метода to_sql (по аналогии с CSV):

```
import sqlite3
con = sqlite3.connect("sample_data/test_db.sqlite")
con.execute("DROP TABLE IF EXISTS employees")
employees.to_sql("employees", con)
```

```
!ls sample_data/test_db.sqlite
```

```
sample_data/test_db.sqlite
```

Теперь мы можем загрузить записанные данные:

```
con = sqlite3.connect("sample_data/test_db.sqlite")
df = pd.read_sql("SELECT * FROM employees LIMIT 5", con)
df
```

```
   EmployeeId LastName FirstName               Title  ReportsTo  \
0           1    Adams    Andrew     General Manager        NaN
1           2  Edwards     Nancy       Sales Manager        1.0
2           3  Peacock      Jane  Sales Support Agent        2.0
3           4     Park  Margaret  Sales Support Agent        2.0
4           5  Johnson     Steve  Sales Support Agent        2.0

             BirthDate             HireDate              Address      City  \
0  1962-02-18 00:00:00  2002-08-14 00:00:00  11120 Jasper Ave NW  Edmonton
1  1958-12-08 00:00:00  2002-05-01 00:00:00         825 8 Ave SW   Calgary
2  1973-08-29 00:00:00  2002-04-01 00:00:00        1111 6 Ave SW   Calgary
3  1947-09-19 00:00:00  2003-05-03 00:00:00     683 10 Street SW   Calgary
4  1965-03-03 00:00:00  2003-10-17 00:00:00         7727B 41 Ave   Calgary

  State Country PostalCode              Phone                Fax  \
0    AB  Canada    T5K 2N1  +1 (780) 428-9482  +1 (780) 428-3457
1    AB  Canada    T2P 2T3  +1 (403) 262-3443  +1 (403) 262-3322
2    AB  Canada    T2P 5M5  +1 (403) 262-3443  +1 (403) 262-6712
3    AB  Canada    T2P 5G3  +1 (403) 263-4423  +1 (403) 263-4289
4    AB  Canada    T3B 1Y7   1 (780) 836-9987   1 (780) 836-9543

                     Email
0    andrew@chinookcorp.com
1     nancy@chinookcorp.com
2      jane@chinookcorp.com
3  margaret@chinookcorp.com
4     steve@chinookcorp.com
```

Главное преимущество хранения данных в базе в том, что можно напрямую делать SQL запросы. Это особенно хорошо, если SQL для вас более родной язык. Например, можно отсортировать по колонке 'Weather' с помощью лишь SQL:

```
df = pd.read_sql("SELECT * FROM employees ORDER BY HireDate LIMIT 5", con)
df
```

```
   EmployeeId LastName FirstName               Title  ReportsTo  \
0           3  Peacock      Jane  Sales Support Agent        2.0
1           2  Edwards     Nancy       Sales Manager        1.0
2           1    Adams    Andrew     General Manager        NaN
3           4     Park  Margaret  Sales Support Agent        2.0
4           5  Johnson     Steve  Sales Support Agent        2.0

             BirthDate             HireDate              Address      City  \
0  1973-08-29 00:00:00  2002-04-01 00:00:00        1111 6 Ave SW   Calgary
1  1958-12-08 00:00:00  2002-05-01 00:00:00         825 8 Ave SW   Calgary
2  1962-02-18 00:00:00  2002-08-14 00:00:00  11120 Jasper Ave NW  Edmonton
3  1947-09-19 00:00:00  2003-05-03 00:00:00     683 10 Street SW   Calgary
4  1965-03-03 00:00:00  2003-10-17 00:00:00         7727B 41 Ave   Calgary

  State Country PostalCode              Phone                Fax  \
0    AB  Canada    T2P 5M5  +1 (403) 262-3443  +1 (403) 262-6712
```

```
1    AB  Canada    T2P 2T3  +1 (403) 262-3443  +1 (403) 262-3322
2    AB  Canada    T5K 2N1  +1 (780) 428-9482  +1 (780) 428-3457
3    AB  Canada    T2P 5G3  +1 (403) 263-4423  +1 (403) 263-4289
4    AB  Canada    T3B 1Y7   1 (780) 836-9987   1 (780) 836-9543

                      Email
0      jane@chinookcorp.com
1     nancy@chinookcorp.com
2    andrew@chinookcorp.com
3  margaret@chinookcorp.com
4     steve@chinookcorp.com
```

Не забываем отключаться от БД.

con.close()

Выводы

1. Pandas мощная библиотека для обработки данных из стандартных форматов. Позволяющая выполнять очистку, обогащение, группировку, изучение и сохранение данных.
2. Все операции производятся в памяти над объектами кадров данных и последовательностей.
3. Если вы только начинающий инженер данных или готовите небольшую выборку для прототипирования (этап разработки модели), то знания Pandas могут очень ускорить вашу работу.

6. Качество данных

6.1. Определения

Определить качество данных сложно из-за множества контекстов, в которых используются данные, а также из-за различных точек зрения конечных пользователей, производителей и хранителей данных.

С точки зрения потребителя качество данных:

- «данные, пригодные для использования потребителями данных»
- данные «соответствуют или превосходят ожидания потребителей»
- данные, которые «удовлетворяют требованиям предполагаемого использования»

С точки зрения бизнеса качество данных — это:

- данные, которые «пригодны для использования» в предполагаемых операционных целях, при принятии решений и в других целях» или демонстрируют «соответствие установленным стандартам», так что достигается пригодность для использования»
- данные, которые «пригодны для предполагаемого использования в операциях, принятии решений и планировании»
- «способность данных удовлетворять заявленным бизнес-, системным и техническим требованиям предприятия»

С точки зрения стандартов качество данных:

- «степень, в которой набор неотъемлемых характеристик (размеров качества) объекта (данных) соответствует требованиям»
- «полезность, точность и правильность данных для его применения»

Можно утверждать, что во всех этих случаях «качество данных» представляет собой сравнение фактического состояния конкретного набора данных с желаемым состоянием, при этом желаемое состояние обычно называют «пригодным для использования», «соответствует спецификации», «соответствие ожиданиям потребителей», «без дефектов» или «соответствие требованиям». Эти ожидания, спецификации и требования обычно определяются одним или несколькими лицами или группами, организациями по стандартизации, законами и правилами, бизнес-политиками или политиками разработки программного обеспечения.

6.2. Измерения качества данных

Данные имеют ограниченный срок годности

Первичные данные всегда актуальны на какой-то конкретный момент времени в прошлом и очень редко актуальны в течение какого-либо длительного периода. *Это одна из проблем качества*: цифровые данные, как регистрация исторического состояния объекта или системы постоянно теряют свою актуальность со временем и их приходится обновлять. **Качество данных – характеристика наборов цифровых данных, показывающая степень их пригодности к обработке и анализу и соответствия обязательным и специальным требованиям, в связи с этим к ним предъявляемым.**

А что может составлять такое понятие как «качество публичных данных»? Выделим девять показателей.

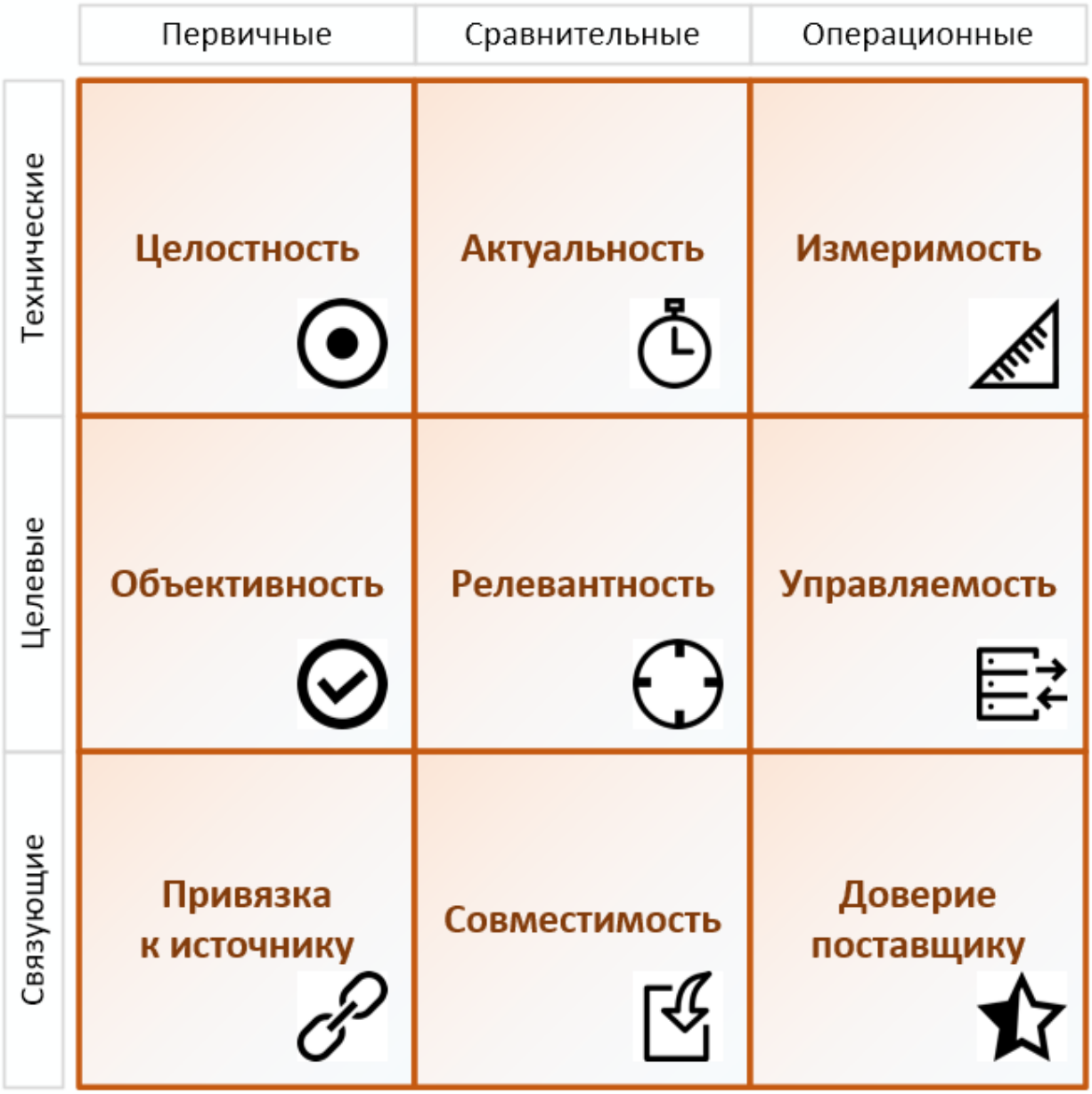

Рис.12 Схема измерений качества данных

1. Актуальность данных

Обозначенный или косвенно определяемый момент времени, на который данные отражают реальное состояние целевого субъекта (объекта, системы, явления, модели, события и т.п.). Актуальность данных также может быть обозначена через период времени, в течение которого они сохраняют свою значимость. Учитывая постоянные изменения экономических систем, публичные экономические данные имеют достаточно короткие сроки актуальности.

Актуальность данных чаще всего устанавливается **поставщиком**, в дополнение к которой он также может «дать обещание» периодического их обновления для её поддержания. **Получатель** данных может самостоятельно оценивать их актуальность на основании информации от поставщика или иными способами.

2. Объективность данных

Точность отражения данными реального состояния целевого субъекта (объекта, системы, явления, модели, события и т.п.). Объективность напрямую зависит от применяемого метода и процедур сбора информации, а также от плотности регистрируемых данных. В процессе обработки наборов цифровых данных они теряют свою объективность и обогащаются агрегированными, округленными, приведенными и расчетными показателями. Однако за счет этого данные «насыщаются» знаниями, тем самым позволяя в последующем сокращать последовательность операций по извлечению из них значимых для практики сведений. **Поставщик** может указать объективность публичных данных охарактеризовав их первичность и описав процедуру их получения. **Получатель** вправе критично отнестись к вторичным данным, особенно если их объективность не доказана применяемыми формулами и математическими расчетными моделями.

4. **Целостность данных**

   Полнота отражения данными реального состояния целевого субъекта (объекта, системы, явления, модели, события и т.п.). В отличии от объективности целостность показывает, насколько полными и безошибочными являются данные как в части смыслового непротиворечия, так и в части соответствия заданной структуре или выбранного формата. Целостность зависит от корректного разделения на элементарные неделимые единицы, сохранения их неделимости, правильной идентификации и взаимной связанности. Данные, публикуемые добросовестным **поставщиком** по умолчанию, должны являться целостными. **Получатель** определяет целостность специальными проверочными методами оценивая смысловое содержание, корректность определения структуры и технически проверяя формат.

5. **Релевантность данных**

   Соответствие данных о реальном состоянии целевого субъекта (объекта, системы, явления, модели, события и т.п.) решаемой задачи (поставленной цели) и возможность их применения с учетом имеющегося содержания, структуры и формата. Понимание релевантности напрямую увязывается с целью пользователя данных и конкретной исполняемой им задачи, а значит и с располагаемым исходным набором данных. **Поставщик** не может повлиять на релевантность данных, но может существенно упростить понимание данного показателя качества с помощью расширенных метаданных, применения распространенных форматов и традиционных структур, а также указанием рекомендаций по их использованию. **Получатель** в каждом конкретном случае оценивает релевантность наборов данных исходя из тематики и рабочего формата (т.е. используемых инструментов).

6. **Совместимость данных**

   Совместная обработка данных о реальном состоянии целевого субъекта (объекта, системы, явления, модели, события и т.п.) с имеющимися в рамках решаемой задачи (поставленной цели). В отличие от релевантности, совместимость — это процедурный показатель, который характеризует

возможность включить данные в обрабатываемый массив для дальнейшего анализа и не связан напрямую с сутью и критериями текущей задачи. С другой стороны, совместимость на содержательном уровне с тематикой исполняемой задачи важна для эффективной обработки цифровых данных. Публичные данные должны особенно тщательно оцениваться на совместимость, в том числе с точки зрения их разновидности. Допустимо ли для конкретных целей совмещение – взаимное использование — открытых данных и разделяемых данных или разделяемы и делегируемых данных зависит от оценки аналитика. Чаще всего необходимо соблюдать условия раздельного хранения и контроля разных видов публичных данных. **Поставщик** публичных данных задает совместимость через метаданные и ссылки на контекст. **Получатель** определяет возможность совместного использования данных для каждого набора как по содержанию и структуре, так и по формату. Но в отличие от релевантности, несовместимые данные можно попытаться привести к совместимому с помощью различных операций трансформации, перекодирования, перевода и т.п.

7. **Измеримость данных**
   Присутствие в данных обрабатываемых качественных или количественных характеристик реального состояния целевого субъекта (объекта, системы, явления, модели, события и т.п.), а также подсчитанный конечный объем набора цифровых данных. Содержательная измеримость данных является основой для выполнения последующих процедур их обработки и анализа. Измерение же общего объема данных необходимо для выбора инструментария и контроля их целостности в процессе обработки и по итогам анализа. **Поставщик** может явно указывать «измерения», включенные в данные, как количественные, так и качественные. Как минимум, сопровождение наборов публичных данных записью об итоговом или пофайловом их размере в байтах почти является общепринятым стандартом. **Получатель** публичных данных восстанавливает измеримость в содержании данных анализируя их и исследуя структуру и всегда точно или бегло проверяет насколько их физический размер соответствует заявленному.
8. **Управляемость данных**
   Возможность целевым и осмысленным образом обработать, передать и контролировать данные о реальном состоянии целевого субъекта (объекта, системы, явления, модели, события и т.п.). Управляемость обусловлена необходимостью изменять, исправлять, структурировать, организовывать, фильтровать, сохранять, пересылать, оценивать, распределять данные. Она во многом основывается на правильно выбранной структуре и формате. **Поставщик** может заявить об управляемости данных через сопровождение их специальными метаданными, но **получатель**, как правило, самостоятельно проводит её оценку исходя из имеющихся у него компетенций и инструментов.
9. **Привязка к источнику данных**
   Связанная и достоверная идентификация цепочки поставки данных о реальном состоянии целевого субъекта (объекта, системы, явления, модели,

события и т.п.). При этом в описание «цепочки поставки публичных данных» лучше включить указания на все субъекты, которые исполняли основные роли трансфера данных: генератор (автор), владелец, поставщик. Привязка к источнику позволяет поставщику и получателю сослаться и восстановить авторство, правоотношения, достоверность источника, доверие к распространителям. Публичные данные почти всегда распространяются с указанием владельца и **поставщика**. И более того, одним из ограничений использования данных является необходимость указать первоисточник при их последующей публикации или использовании. Следует учитывать, что хорошая привязка данных позволяет по необходимости получить её повторно с уточнениями, дополнительной актуализацией или с восстановленной целостностью, т.е. – с повышенным качеством.

10. **Доверие к поставщику данных**

    Оценка получателем деловых качеств поставщика публичных данных о целевом состоянии субъекта (объекта, системы, явления, модели, события и т.п.), как ответственного, авторитетного, организованного и относительно независимого издателя цифровой информации высокого качества. Данный показатель выступает некоторой интегрированной ретроспективной оценкой всех предыдущих трансферов данных поставщика – репутация издателя публичных данных. **Получатель** всегда исходит из внутренней убежденности при определении такого показателя качества данных, но у поставщика есть несколько путей по формированию и поддержанию нужного ему уровня доверия. К ним можно, например, отнести: тщательную подготовку данных для публичного трансфера, высокий уровень организации процессов издания «цифры», поддержку обратной связи с получателями, своевременную актуализацию и извещение об обнаруженных в данных проблемах, специальные мероприятия, участие в независимой оценке и ассоциациях. **Любой из указанных показателей качества данных субъективен**, как в части смыслового содержания данных, так и в части его восприятия разными поставщиками и получателями. Тем не менее все показатели можно разделить на:

    условно-объективные – это показатели, значения которых слабо зависят от мнения поставщика или получателя данных и устанавливаются в соответствии с контролируемыми и частично проверяемыми критериями, *к ним относятся: актуальность, целостность, измеримость, совместимость, привязка к источнику*.

    условно-субъективные – это показатели, значения которых напрямую зависят от мнения поставщика или получателя данных и устанавливаются в соответствии с внутренней «убежденностью» как некоторая допустимая критериальная оценка, *к ним относятся: объективность, релевантность, управляемость, доверие к поставщику*.

    Формальная оценка каждого из показателей качества может осуществляться как в баллах (в заданном интервале), так и в процентах. Причем бальная оценка может даваться экспертным путем, а процент может высчитываться как доля данных, отвечающих заданному показателю качества

к общему объему данных. В последнем случае задача выглядит много более сложная и требует специальных инструментов, хотя и будет давать взвешенную, но всё-таки экспертную оценку качества. Одним их важных аспектов формальной оценки показателей качества является их контроль по мере работы с наборами цифровых данных. В динамике качество данных не должно ухудшаться, т.е. экспертная оценка данных не должна неуправляемо снижаться после отдельных операций или целой серии обработок. Общая проблема качества публичных данных зависит как от каждого из перечисленных показателей, так и от **интегрированной субъективной оценки получателя**. В любом случае, качество важно в первую очередь получателю, как лицу, выполняющему операции обработки и анализа.

В случае завершения обратной связи стороннего результативного пользователя данных с поставщиком «проблема» качества данных возвращается последнему «бумерангом». Если данные были предоставлены «плохие» или с ошибками, то ожидать от тех, кто их использовал, сколь-либо хороших и адекватных итогов не приходится. Тогда утрачивается весь смысл усилий по выбору, подготовке и публикации данных – поставщик не получает никаких новых полезных решений и знаний (продуктов или сервисов). **Важнейший показатель качества данных – это их целостность** Он оказывает сильное влияние на *совместимость* и *управляемость* данных. А неоднократная публикация данных с нарушением целостности обязательно скажется на доверии к их поставщику. Целостность данных не является чем-то обособленным от смысла, структуры или формата и должна соблюдать на всех уровнях цифровой информации. Нарушение целостности данных возможно:

на смысловом уровне – при сборе допущена ошибка в полноте или записи данных так, что становиться непонятным само значение, которое описывают такие данные;

на структурном уровне – при упорядочивании элементов данных или при обработке данных допущена ошибка в полноте или записи данных так, что становиться «непонятной» часть или целая структура;

на уровне кодирования – при записи, хранении или чтении данных допущена ошибка на уровне преобразования отдельных символов и понятий так, что данные не удается прочитать и (или) присутствуют пропуски;

на уровне нотации – при записи, хранении или чтении данных допущена ошибка на уровне преобразования отдельных элементов цифровых данных или их совместной записи так, что в данных невозможно правильно установить обособленные отдельные единицы и связи между ними;

на уровне схемы – при записи, хранении или чтении данных допущена ошибка на уровне логики или формата отдельных элементов цифровых данных или их взаимосвязи так, что из данных невозможно извлечь значимую информацию о предметной области. *Аналогично, по каждому из уровней – смысл, структура, формат – можно рассматривать каждый показатель качества данных.* За качество публикуемых данных, конечно же, отвечает поставщик. Но получатель вынужден выполнять проверку и по

необходимости корректировать сами данные. Если публичные данные оказываются низкого качества, то имеет смысл отказаться от их использования и направить подробное уведомление поставщику. Добросовестный и заинтересованный поставщик обязательно предпримет усилия по исправлению ситуации. Он как минимум должен закрыть доступ к некачественным данным на время разбирательства и маркировать их соответствующим образом. Адресованная поставщику претензия относительно качества данных, в условиях максимальной открытости сетевого общения, вынуждает в обязательном порядке помещать специальный заявительный отказ от принятия претензии с обоснованием такого отказа, либо повышать качество данных и повторно их издавать с соответствующими разъяснениями. А в случае, если поддерживается адресная связь с получателями – уведомлять их специальным образом. Поставщик, который не готов отвечать за качество данных достаточно быстро переходит в разряд «безответственных» и теряет все преимущества, предоставляемые сообществом аналитиков и экспертов, занятых в соответствующей предметной области. Из вышесказанного вытекает необходимость постоянного контроля качества данных как со стороны получателя, так и со стороны поставщика. Что в свою очередь вынуждает разрабатывать и применять специальные контрольно-измерительные инструменты. **Исследование проблемы качества цифровых данных, а особенно качества открытых, разделяемых и делегируемых данных должно осуществляться аналитиками и экспертами как на микроуровне заинтересованных бизнесов, так и на макроуровне сообществ и государственных структур. Во многом безопасность будущей цифровой экономики будет базироваться на активном мониторинге качества используемых данных.**

6.3. Контроль качества данных

За оценку качества данных отвечают инженеры Data Quality, которые управляют информационными массивами, проверяют их поведение в текущих и новых условиях, контролируют релевантность, достаточность и актуальность. Как правило, обязанности Data Quality инженера не ограничиваются только рутинными проверками записей в таблицах СУБД, а требуют глубокого понимания бизнес-потребностей, чтобы трансформировать имеющиеся данные в пригодную к практическому использованию информацию. Для этого решаются следующие задачи:

- автоматизированная подготовка тестовых данных;
- загрузка подготовленного датасета в исходный источник, например, озеро данных (Data Lake);
- запуск ETL-процессов для обработки набора данных из исходного источника и отправку в окончательное или промежуточное хранилище с возможностью конфигурации параметров ETL-задачи, например, с помощью Apache Airflow;

- верификация данных после ETL-обработки на предмет их качества и соответствие бизнес-требованиям.

Для организации data chain – цепочки проверочных тестов на каждой стадии обработки данных от источника до пункта финального использования могут использоваться легковесные SQL-запросы. Они помогают оценить отдельные атрибуты качества данных, например, tables metadata, blank lines, NULLs, Errors in syntax. Для регрессионного тестирования, когда используются уже готовые неизменяемые датасеты, код автотестов уже хранит готовые шаблоны проверки данных на соответствие качеству, такие как описания ожидаемых метаданных таблиц, строчных выборочных объектов для случайного выбора и т.д. Иногда в ходе тестирования Big Data Quality инженер пишет тестовые ETL-процессы с помощью Apache Spark или Airflow, используя уже готовые операторы, в частности, GCP BigQuery или создавая собственные. Про операторы Apache Airflow мы писали здесь, на примере Kubernetes Operator.

Разумеется, все это инженеры Data Quality выполняют не вручную. Современный рынок ПО предлагает множество специализированных инструментов для проверки качества данных и их улучшения. В частности, Informatica Data Quality, Microsoft Data Quality Services, Oracle Enterprise Data Quality, SAP Data Services, Talend Open Studio for Data Quality и другие коммерческие продукты, а также открытые сервисы и библиотеки типа Great Expectations. Аналитическое агентство Gartner составило список ТОП-10 таких решений, проранжировав их по удобству использования, функциональным возможностям и отзывам профессионалов.

Как правило, большинство специализированных систем управления качеством данных автоматизируют следующие процессы Data Quality Management:

профилирование – первоначальная оценка данных, чтобы понять их текущее состояние, в т.ч. распределение значений;

стандартизация – механизм бизнес-правил, обеспечивающий соответствие данных стандартам;

геокодирование адресов, которое корректирует данные в соответствии с географическими стандартами;

сопоставление или связывание – способ сравнения данных для выявления одинаковых по смыслу, но разных по виду представления записей. Сопоставление может использовать нечеткую логику для поиска дубликатов в данных. Например, «Петр» и «Птер» может быть одним и тем же человеком, который проживает по одинаковому адресу.

мониторинг – отслеживание качества данных с течением времени и отчетность об изменениях с автоматическим исправлением изменений на основе предварительно определенных бизнес-правил;

пакетная и потоковая обработка – первичная очистка данных в пакетном режиме с последующей интеграцией в корпоративные приложения.

Обеспечение качества данных не сводится лишь к технической задаче устранения дублирующихся или пропущенных значений. Важна также организационная сторона этого процесса, где задействован не только Data Quality инженер. В следующей статье мы поговорим про ответственность за

качество данных и профессию дата стюарда (Data Steward). А о лучших практиках повышения Data Quality в компании Airbnb с помощью организационных изменений и архитектурно-технических решений читайте здесь.

6.4. Оптимальное использование качества данных

Качество данных (DQ) — это ниша, необходимая для обеспечения целостности управления данными путем устранения пробелов в данных. Это одна из ключевых функций, которые помогают управлению данными путем мониторинга данных для поиска исключений, не обнаруженных текущими операциями управления данными. Проверки качества данных могут быть определены на уровне атрибутов, чтобы иметь полный контроль над этапами исправления.

Проверки DQ и бизнес-правила могут легко перекрываться, если организация не уделяет должного внимания своему объему DQ. Бизнес-команды должны тщательно понимать объем DQ, чтобы избежать дублирования. Проверки качества данных являются излишними, если бизнес-логика охватывает ту же функциональность и выполняет ту же цель, что и DQ. Объем DQ организации должен быть определен в стратегии DQ и должным образом реализован. Некоторые проверки качества данных могут быть преобразованы в бизнес-правила после неоднократных исключений в прошлом.

Ниже приведены несколько областей потоков данных, которые могут нуждаться в постоянных проверках DQ:

Проверки полноты и точности DQ для всех данных могут выполняться в точке ввода для каждого обязательного атрибута из каждой исходной системы. Несколько значений атрибутов создаются после первоначального создания транзакции; в таких случаях администрирование этих проверок становится сложным и должно выполняться сразу же после того, как определенное событие источника этого атрибута и других основных условий атрибута транзакции выполнены.

Все данные, имеющие атрибуты, относящиеся к справочным данным в организации, могут быть проверены на соответствие набору четко определенных допустимых значений справочных данных для обнаружения новых или несоответствующих значений посредством проверки достоверности DQ. Результаты могут использоваться для обновления справочных данных, администрируемых в рамках управления основными данными (MDM).

Все данные, полученные от третьих лиц для внутренних команд организации, могут пройти проверку на точность (DQ) по сравнению с данными третьих лиц. Эти результаты проверки DQ полезны при администрировании данных, которые совершили несколько переходов после точки ввода этих данных, но до того, как эти данные будут авторизованы или сохранены для корпоративной аналитики.

Все столбцы данных, которые ссылаются на основные данные, могут быть проверены на предмет их согласованности. Проверка DQ, проводимая для

данных в точке входа, обнаруживает новые данные для процесса MDM, но проверка DQ, проводимая после точки входа, обнаруживает нарушение (не исключения) согласованности.
По мере преобразования данных фиксируются несколько временных меток и положения этих временных меток, которые можно сравнивать друг с другом и с их запасом для проверки их значения, затухания, эксплуатационной значимости в соответствии с определенным SLA (соглашением об уровне обслуживания). Эту проверку своевременности DQ можно использовать для уменьшения скорости затухания значений данных и оптимизации политик временной шкалы перемещения данных.
В организации сложная логика обычно разделяется на более простую логику в нескольких процессах. Разумность Проверки DQ такой сложной логики, приводящие к логическому результату в определенном диапазоне значений или статических взаимосвязей (агрегированные бизнес-правила), могут быть проверены для обнаружения сложных, но важных бизнес-процессов и выбросов данных, их отклонения от BAU (обычный бизнес).) ожидания и могут создавать возможные исключения, которые в конечном итоге приводят к проблемам с данными. Эта проверка может быть простым общим правилом агрегации, охватываемым большим блоком данных, или сложной логикой группы атрибутов транзакции, относящейся к основному бизнесу организации. Эта проверка DQ требует высокой степени деловых знаний и сообразительности. Обнаружение проблем разумности может помочь в изменении политики и стратегии либо бизнесом, либо управлением данными, либо тем и другим.
Проверки соответствия и проверки целостности не должны охватывать все потребности бизнеса, это строго на усмотрение архитектуры базы данных.
Есть много мест в движении данных, где проверки DQ могут не потребоваться. Например, проверка DQ на полноту и точность ненулевых столбцов является избыточной для данных, полученных из базы данных. Точно так же данные должны быть проверены на их точность относительно времени, когда данные объединяются из разрозненных источников. Однако это бизнес-правило, и оно не должно входить в область DQ.
К сожалению, с точки зрения разработки программного обеспечения DQ часто рассматривается как нефункциональное требование. Таким образом, ключевые проверки/процессы качества данных не учитываются в окончательном программном решении. В сфере здравоохранения носимые технологии или сети Body Area Network генерируют большие объемы данных. Уровень детализации, необходимый для обеспечения качества данных, чрезвычайно высок и часто недооценивается. Это также верно для подавляющего большинства приложений мобильного здравоохранения, электронных медицинских карт и других программных решений, связанных со здоровьем. Однако существуют некоторые инструменты с открытым исходным кодом, которые проверяют качество данных. Основная причина этого связана с дополнительными затратами, связанными с добавлением более высокой степени строгости в архитектуру программного обеспечения.

7. Разметка данных. Сценарии применения размеченных данных в задачах компьютерного зрения и обработки естественного языка

7.1. Семантическая сегментация изображений в CVAT

Общие сведения

Computer Vision Annotation Tool (CVAT) – это инструмент с открытым исходным кодом для разметки цифровых изображений и видео. Основной его задачей является предоставление пользователю удобных и эффективных средств разметки наборов данных. Мы создаем CVAT как универсальный сервис, поддерживающий разные типы и форматы разметки.

Для конечных пользователей CVAT – это web-приложение, работающее в браузере. Он поддерживает разные сценарии работы и может быть использован как для персональной, так и для командной работы. Основные задачи машинного обучения с учителем в области обработки изображений можно разбить на три группы:

- Детектирование объектов
- Классификация изображений
- Сегментация изображений

CVAT пригоден во всех этих сценариях.

Преимущества:

1. Отсутствие установки у конечных пользователей. Для создания задачи или разметки данных достаточно открыть определенную ссылку в браузере.
2. Возможность совместной работы. Существует возможность сделать задачу общедоступной для пользователей и распараллелить работу над ней.
3. Простота развертывания. Установка CVAT в локальной сети осуществляется парой команд за счет применения Docker.
4. Автоматизация процесса разметки. Интерполяция, к примеру, позволяет получить разметку на множестве кадров, при реальной работе лишь над некоторыми ключевыми.
5. Опыт профессионалов. Инструмент разрабатывался с участием аннотационной и нескольких алгоритмических команд.
6. Возможность интеграции. CVAT пригоден для встраивания в платформы более широкого назначения. Например Onepanel.
7. Опциональная поддержка различных инструментов:
    a. Deep Learning Deployment Toolkit (компонент в составе OpenVINO)
    b. Tensorflow Object Detection API (RCNN и MaskRCNN)
    c. Алгоритм Deep Extreme Cut
    d. ELK (Elasticsearch + Logstash + Kibana) система аналитики
    e. NVIDIA CUDA Toolkit
8. Поддержка разных аннотационных сценариев.

9. Продвинутая система стриминга изображений и видео с сервера на клиент.
10. Открытый исходный код под простой и свободной лицензией MIT.
11. Демо сервер, предоставляющий возможность создать небольшие задачи и опробовать основные инструменты CVAT.

Недостатки:

1. Ограниченная поддержка браузеров. Мы не тестируем CVAT в большом количестве браузеров. Рекомендуется использовать Google Chrome последней версии или другой браузер на движке Chromium. Ожидается работа в Mozilla Firefox. Однако, например в Safari на данный момент существуют проблемы.
2. Не проработана система автоматических тестов. Разработаны некоторые модульные тесты, но они не покрывают значительную часть функционала.
3. Отсутствует документация по исходному коду. Включиться в разработку может быть достаточно трудно.
4. Ограничения по производительности. С увеличением требований по объемам разметки мы столкнулись с разными проблемами, как например, ограничение Chrome Sandbox по использованию оперативной памяти.

Конечно, эти списки не исчерпывающие, но содержат основные положения.

Как было сказано ранее, CVAT поддерживает ряд дополнительных компонентов, реализованных в виде serverless функций. Среди них:

- Deep Learning Deployment Toolkit в составе OpenVINO Toolkit – позволяет запускать пользовательские модели нейронных сетей для получения автоматической аннотации.
- Tensorflow Object Detection API – используется для автоматической разметки объектов. CVAT дает возможность использовать разные модели среди обученных для задач детектирования и сегментации.
- Алгоритм Deep Extreme Cut добавляет возможность получения маски объекта из его крайних (extreme) точек.
- Logstash, Elasticsearch, Kibana – позволяют визуализировать и анализировать накопленные клиентами логи. Это может применяться, например, для мониторинга процесса разметки или поиска ошибок и причин их возникновения.

Разметка данных

Процесс начинается с постановки задачи для разметки. Постановка включает в себя:

1. Указание названия задачи
2. Перечисление классов, подлежащих разметке и их атрибутов
3. Указание файлов на загрузку
4. Данные загружаются с локальной файловой системы; распределенной, смонтированной в контейнере файловой системы; либо по прямым удаленным ссылкам

5. Задача может содержать один архив с изображениями, одно видео, набор изображений и даже структуру каталогов с изображениями при загрузке через распределенное хранилище

Опционально могут быть заданы:

- Ссылка на подробную спецификацию разметки, а также любую другую дополнительную информацию (Bug Tracker)
- Ссылка на удаленный Git-репозиторий для хранения аннотации (Dataset Repository)
- Размер сегмента (Segment Size). Загружаемая задача может быть разбита на несколько подзадач для параллельной работы
- Область пересечения сегментов (Overlap). Используется в видео для слияния аннотации в разных сегментах
- Уровень качества при конвертации изображений (Image Quality)
- Размер чанка медиаданных при передаче с сервера на клиент

7.2. Разметка именованных сущностей в Label Studio

Label Studio поддерживает следующие форматы файлов:
— Text - txt
— Audio - wav, aiff, mp3, au, flac, m4a, ogg
— Images - jpg, png, gif, bmp, svg, webp
— HTML - html, htm, xml
— Time Series - csv, tsv
— Common Formats - csv, tsv, txt, json
Для NER-разметки мы создаем txt-файл, в который предварительно вставляем все спарсенный тексты.
Далее переходим в Labeling Setup. Там нам нужно выбрать раздел Natural Language Processing и в нем уже Named Entity Recognition.

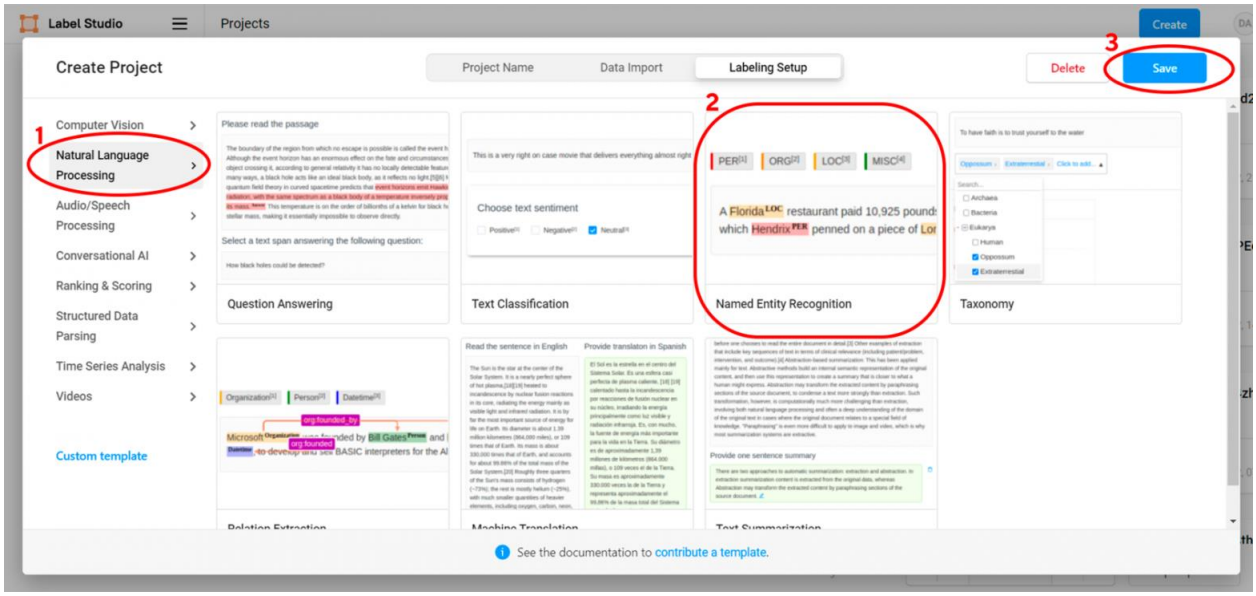

Рис.13 Пользовательский интерфейс проекта в Label Studio

Теперь необходимо задать классы извлекаемых сущностей. Но прежде, чем мы это сделаем, давайте разберемся, что из себя представляют именованных сущности и какие они бывают.

Именованные сущности — это своего рода категории слов и словосочетаний, сгруппированных по значению. Например, самые распространенные сущности, выделяемые в текстах: имена людей, названия организаций, даты и так далее. Хотя в большинстве случаев всё зависит от сферы применения и конкретных задач, которые должна решать нейросеть.

Теперь возвращаемся к настройке нашего проекта. После того, как мы выбрали вид разметки, перед нами откроется окно настройки классов сущностей.

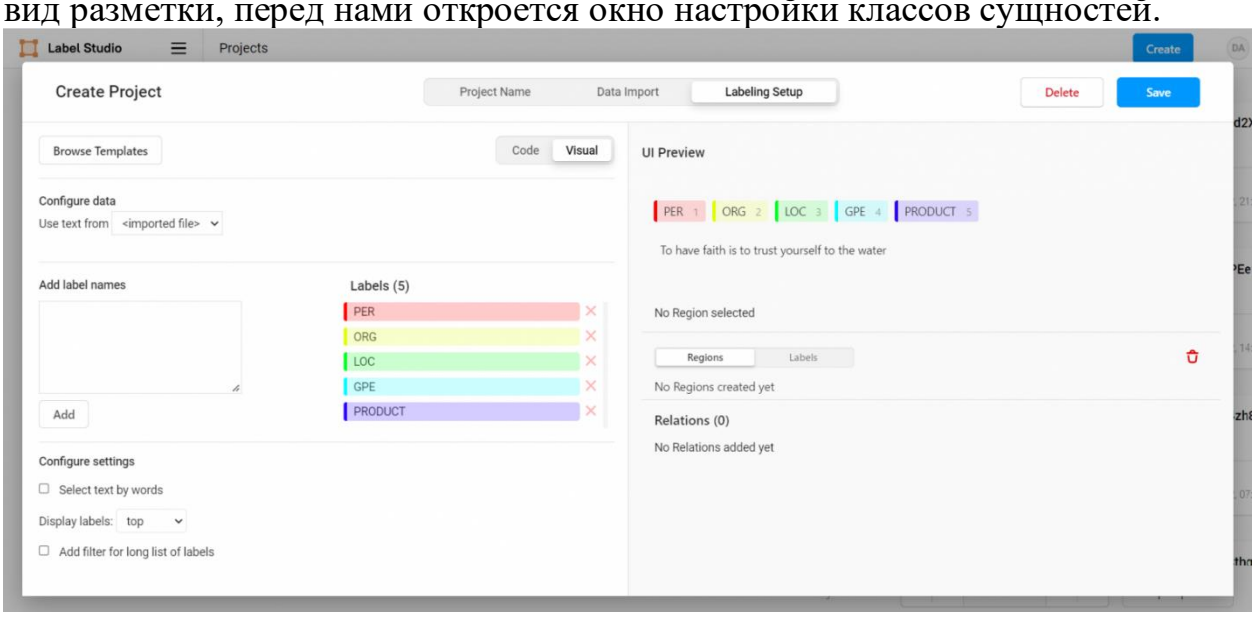

Рис.14 Окно настройки классов сущностей в Label Studio

Как видно на скриншоте, мы выбрали для наших текстов:

PER - люди, имена

ORG - название компаний, организаций

LOC - не географические локации

GPE - страны, города, населенные пункты

PRODUCT - товары, продающиеся объекты (кроме услуг)

Для удобства назначаем разные цвета для каждой сущности и нажимаем на **Save**.

Разметка именованных сущностей

После того, как нажали на Save, нас перекидывает на страницу проекта.

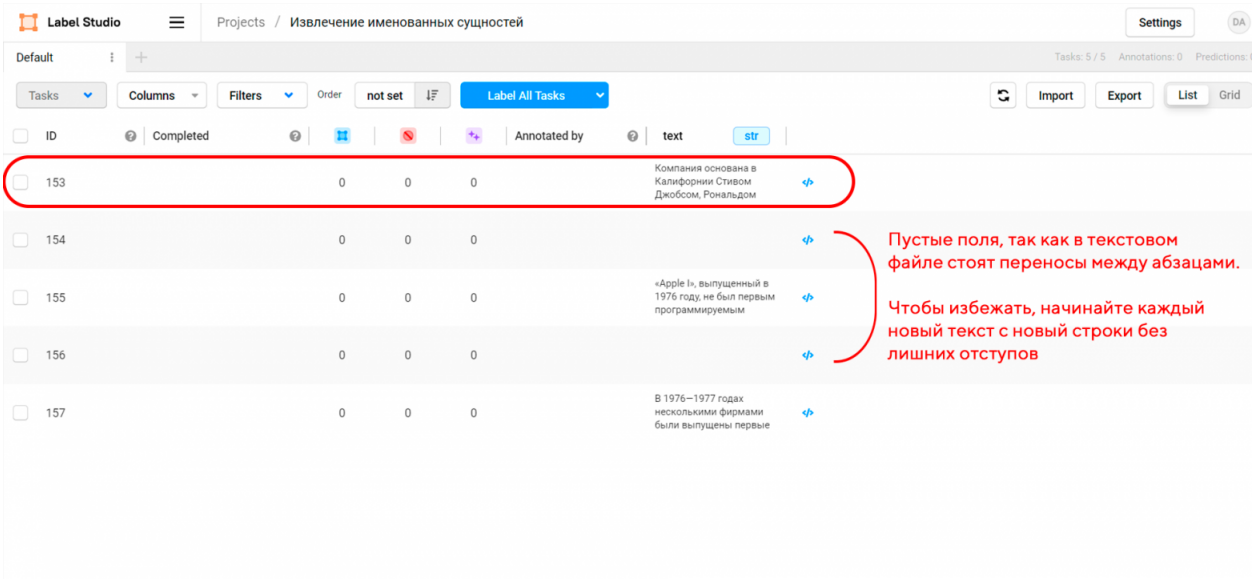

Рис.15 Окно разметки сущностей в Label Studio

Для примера я взял небольшой фрагмент из статьи про Apple на Википедии. Label Studio автоматически выносит каждую текстовую строку из файла в отдельный джоб. У меня на скриншоте между ними пробел, так как и в текстовом файле между ними есть отступ.

Теперь кликаем на любой из текстов и начинаем размечать.

Стоит отдать должное Label Studio: интерфейс интуитивно понятен и удобен. Просто кликаешь на нужный класс сущности, а затем выделяешь слово или словосочетание. Повторяешь до тех пор, пока не выделишь все сущности.

А что делать, если забыл какой-то из классов? Прям, как я забыл про сущности дат. Всё просто. Нажимаешь на **Submit**, чтобы сохранить текущий прогресс, а затем возвращаемся к настройкам проекта. Далее кликаем на **Settings** в правом верхнем углу экрана.

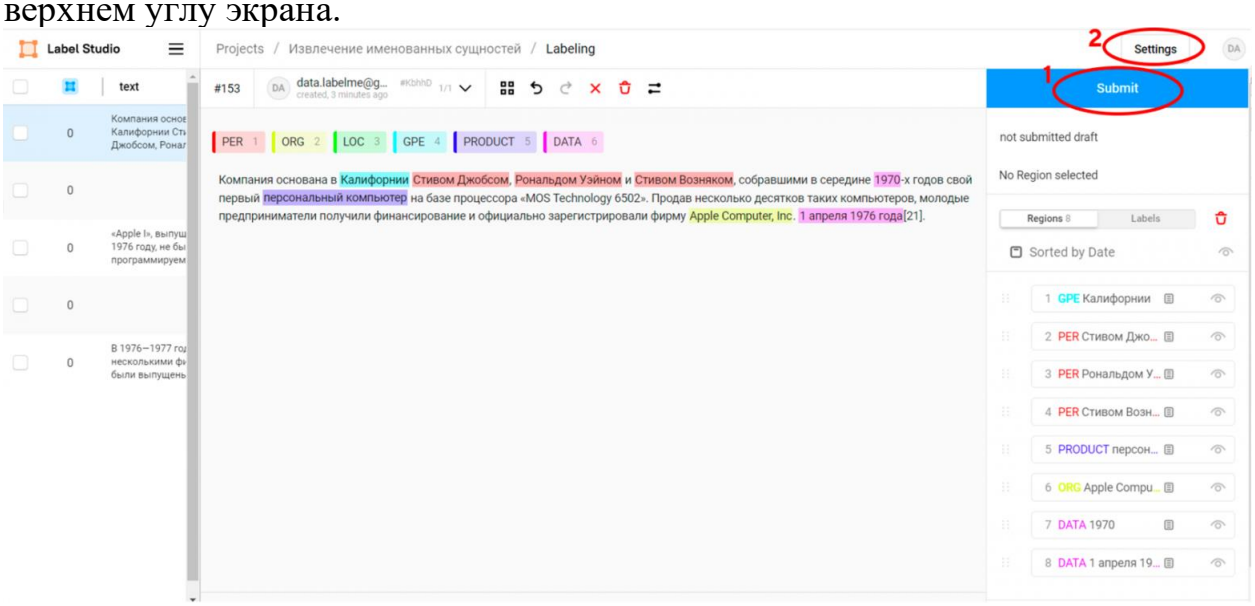

Рис.16 Окно разметки сущностей в Label Studio

Теперь переходим во вкладку **Labeling Interface** и добавляем нужный класс, как в самом начале настройки проекта.

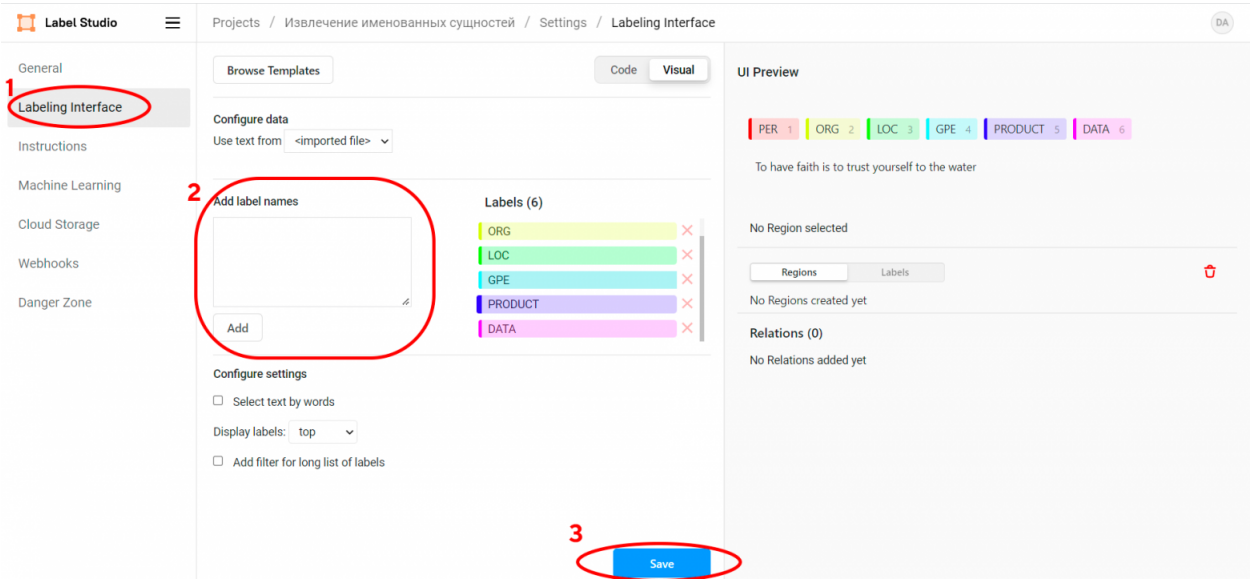

Рис.17 Окно разметки классов в Label Studio

Следующий важный нюанс, с которым мы можем столкнуться в ner-разметке, это неоднородность сущностей в тексте. Самый лучший пример - сущности даты. Только в рассматриваемых трех абзацах мы встречаем разные вариации:
— *в середине 1970-х годов*
— *1 апреля 1976 года*
— *в 1976 году*
— *в 1974—1975 годах*
— *в 1976—1977 годах*
— *с 1977 года*
— *с 1977 по 1993 годы*
— *с конца 1970-х и начале 1980-х годов*
Чтобы неоднородность формулировок не повлияла на точность модели, важно максимально стандартизировать принципы аннотации каждой сущности.
Я столкнулся с этой проблемой лично, когда моя компания по разметке делала первые шаги. Каждый разметчик выделял начало и конец сущности по-разному, из-за датасет получался довольно мусорным.
Решением стало написание подробного ТЗ и инструкции с разбором конкретных примеров. То есть под каждую сущность мы разбирали различные примеры написания и аннотации.
Есть и другой способ решения проблемы начала-конца многосоставной сущности — **BIOES-схема**. Её суть заключается в присвоении метке сущности префиксов, указывающих на расположении внутри класса. Схема BIOES включает 4 подобных префикса:
— **B (beginning)** – первое слово в сущности
— **I (inside)** – все слова между первым и последним словом в сущности
— **O (out)** — слово не относящееся к сущности
— **E (ending)** — последнее слово в сущности
— **S (single)** — сущность состоящая из одного слова

Использование BIOES позволяет нам представить разметку сущности в виде токенов и точно определять начало и конец каждой сущности. Пример ниже:

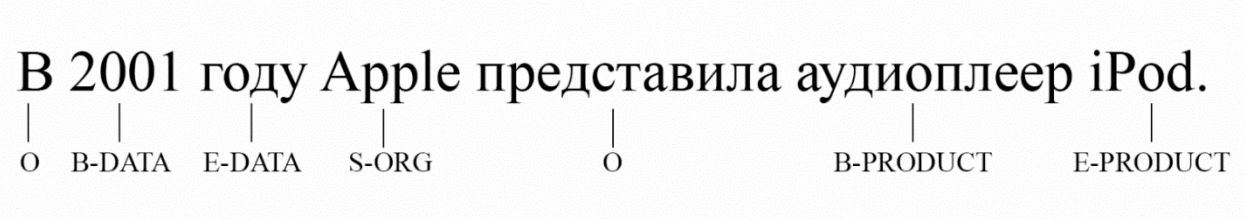

Рис.18 Пример **BIOES-схемы**.

Этот метод более трудоемок, да и в Label Studio нет этого функционала, но и не упомянуть о нем нельзя.
Но это лирическое отступление. Для большинства задач по автоматизации достаточно описанной выше NER-разметки.
Шаг 4: Экспорт размеченных данных
Итак, мы разметили все сущности в имеющихся текстах. Теперь нужно их выгрузить. Для этого выделяем нужные джобы или все сразу, кликнув на **ID**. Затем жмем на **Export** в правой части экрана.
Label предложит один из нескольких форматов. Для задачи по извлечению именованных сущностей чаще всего это:
— JSON
— JSON-MIN
— CSV
— TSV
— CONLL2003

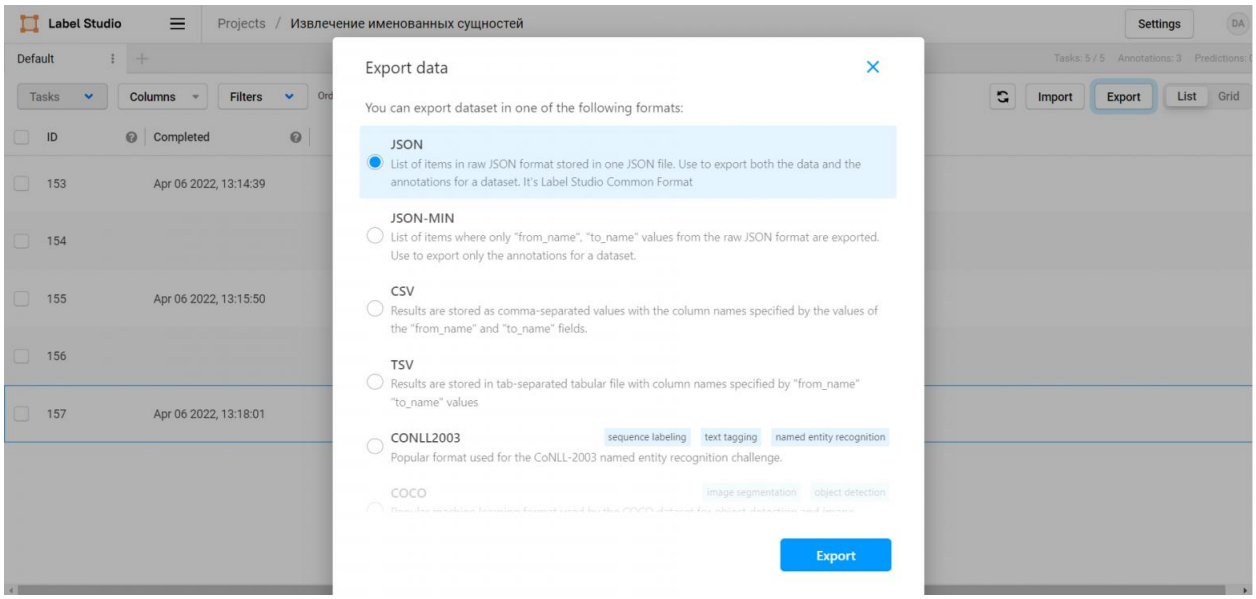

Рис.19 Сохранение результатов

Ставите поинт на нужный и нажимаете Export. Готово.

7.3. Сценарии применения размеченных данных в задачах компьютерного зрения и обработки естественного языка

Под данными для обучения понимается исходный набор данных, передаваемый модели машинного обучения, на котором обучается модель. Люди лучше всего учатся на примерах, и машинам тоже нужен набор данных, чтобы узнавать из него паттерны.

Данные обучения — это данные, которые мы используем для обучения алгоритма машинного обучения. В большинстве случаев данные обучения содержат пары «входящие данные: аннотация», собранные из различных источников, которые используются для обучения модели выполнению конкретной задачи с высоким уровнем точности. Они могут состоять из сырых данных (изображений, текстов или звука), содержащих аннотации, например, ограничивающие прямоугольники, метки или связи. Модели машинного обучения изучают аннотации данных обучения, чтобы в будущем обрабатывать новые, неразмеченные данные.

Данные обучения при контролируемом и неконтролируемом обучении

В чём разница датасета для контролируемого и неконтролируемого обучения? При контролируемом обучении люди размечают данные и сообщают модели именно то, что ей нужно найти. Например, в сфере распознавания спама входящими данными является любой текст, а метка даёт понять, является ли сообщение спамом. Контролируемое обучение строже, потому что мы не позволяем модели делать собственные выводы на основе данных вне пределов, аннотированных нашими метками. При неконтролируемом обучении люди передают модели сырые данные без меток, а модель находит в данных паттерны. Например, распознавая уровень схожести или различия двух примеров данных на основании общих извлечённых признаков. Это помогает модели делать заключения и приходить к выводам, например, разделять похожие изображения или объединять их в кластеры.

Обучение с **частичным контролем** (semi-supervised learning) — это сочетание двух вышеперечисленных типов: данные частично размечаются людьми, а часть прогнозов оставляют на усмотрение модели.

Обучение с частичным контролем часто используется, когда люди могут направить модель в нужном направлении работы, но сами прогнозы становится сложно аннотировать из-за слишком большого количества нюансов.

В реальности нет такого понятия, как полностью контролируемое или неконтролируемое обучение, существуют только различные степени контроля.

Контролируемое обучение: процесс работы с данными обучения

Все методики обучения начинаются со сбора сырых данных из различных источников. Сырые данные могут иметь любой вид: текст, изображения, звуки, видео и т.д. Однако, чтобы сообщить модели, что необходимо находить в этих данных, вы должны добавить аннотации. Эти аннотации позволяют контролировать обучение, гарантировав, что модель сфокусируется на

указанных вами признаках, а не будет экстраполировать выводы из других коррелированных (но не обусловленных) элементов данных.
Все входящие данные должны иметь соответствующую метку, позволяющую машине двигаться в направлении того, как должен выглядеть прогноз. Такой обработанный набор данных можно получить при помощи людей, а иногда и других моделей ML, достаточно точных для надёжного проставления меток. После того, как размеченный набор данных готов к передаче ИИ, начинается этап обучения. На нём модель пытается выявить важные признаки, общие для всех примеров, которым вы назначили метки. Например, если вы сегментировали несколько легковых автомобилей на снимках, то она поймёт, что колёса, зеркала заднего вида и ручки дверей являются признаками, коррелирующими с легковым автомобилем. Модели непрерывно тестируют сами себя на наборе данных валидации, подготовленном перед этапом обучения. После завершения модели выполняют последнюю проверку на тестовом датасете (наборе, который модель ранее никогда не видела); это даёт нам понимание о качестве работы модели на релевантных новых примерах. Наборы данных для обучения, валидации и тестирования являются частями данных обучения. Чем больше данных обучения у вас есть, тем выше точность модели. Теперь давайте дадим определения некоторым популярным терминам, которые вы можете встретить при работе с датасетами.
Размеченные данные — это данные, дополненные метками/классами, содержащими значимую информацию. Вот несколько примеров размеченных данных: изображения с меткой «кошка»/«собака», электронные письма/сообщения, размеченные как спам, прогнозы цен фондовых рынков (меткой является состояние в будущем), определение злокачественности узелков с их выделением многоугольником, аудиофайлы с информацией о том, какие слова в них произнесены.
Точно размеченные данные позволяют машине распознавать паттерны в соответствии с задачей, поэтому они широко используются в решении сложных задач.
Процесс «**оператор в контуре управления**» (human in the loop, HITL) используется тогда, когда модель только частично способна решить задачу, и часть работы передаётся живому человеку. Примером human in the loop является разметка данных при помощи модели, когда модель ML создаёт первоначальные прогнозы, а человек дополняет их метками, исправлениями или другими типами аннотаций, не поддерживаемыми моделью. Люди обеспечивают непрерывную обратную связь, улучшающую качество работы модели. Обычно люди используют инструменты аннотирования для разметки сырых данных, чтобы помочь машинам обучаться и делать прогнозы. Они валидируют полученные моделью результаты и проверяют прогнозы, когда машина не уверена в своих результатах, чтобы убедиться, что обучение модели происходит в нужном направлении. Однако иногда люди постоянно остаются в контуре управления, добавляя в данные новые метки, создание которых нельзя полностью доверить моделям. Например, во многих автоматизированных системах медицинской диагностики или

системах идентификации личности задействуются операторы в контуре управления, чтобы не оставлять на долю алгоритмов машинного обучения выбор окончательного важного решения. В этом контуре машины и люди работают рука об руку.

Выборки для обучения, валидации и тестирования

Ни одну ИИ-модель невозможно обучить и протестировать на одних и тех же данных.

Оценка модели будет смещенной, потому что модель тестируется на том, что она уже знает. Это аналогично тому, как если бы мы давали на экзамене студентам те же вопросы, на которые они уже ответили в классе. Так мы не узнаем, запомнил ли студент ответы или действительно понял тему.

Те же правила применимы и к моделям машинного обучения.

Вот их процентные соотношения объёмов данных:

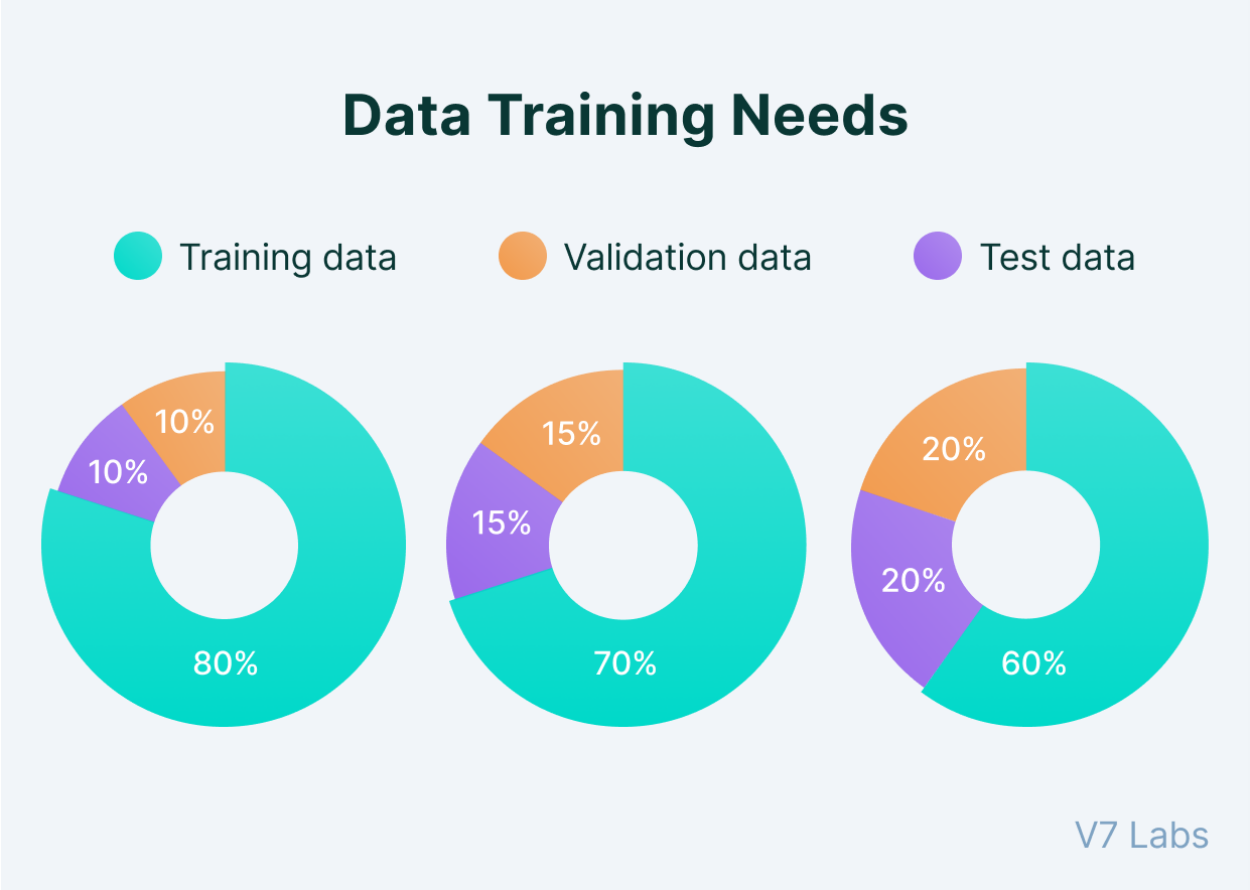

Рис.20 Распределение данных по выборкам

Данные обучения (Training data) — не менее 60% данных должно использоваться для обучения.

Данные валидации (Validation data) — выборка (10-20%) из общего набора данных, используемая для валидации и периодически проверяемая на модели во время обучения. Этот набор данных валидации должен представлять собой репрезентативную выборку из набора данных обучения.

Данные тестирования (Test data) — этот набор данных используется для тестирования модели после её полного обучения. Он отделяется и от набора обучения, и от набора валидации. После обучения и валидации модель

тестируется на наборе тестирования. Данные в наборе тестирования должны выглядеть точно так же, как будут выглядеть реальные данные после развёртывания модели. В общем наборе данных может быть несколько наборов тестирования.

Каждый набор тестирования можно использовать для проверки того, достаточно ли модель обучилась для конкретного сценария применения. Например, модель беспилотного вождения, наученная распознавать пешеходов, может обучаться на видео, снятом по всей территории США.

Её основной набор тестирования может состоять из перемешанных данных всех штатов страны, однако вам может потребоваться создать отдельные наборы тестирования для конкретных сценариев. Например:

- Набор тестирования для вождения на закате
- Набор тестирования для снежного климата
- Набор тестирования для вождения в сильные бури
- Набор тестирования для ситуаций, когда объектив камеры загрязнился или поцарапан.

Эти наборы тестирования обычно хранятся в системе управления наборами данных и вручную подбираются дата-саентистами. Поэтому они являются свидетельством того, что вы полностью понимаете, как выглядят ваши данные и что вы соответствующим образом пометили все краевые сценарии, чтобы из них можно было создавать наборы тестирования.

Наборы тестирования используются не только для оценки качества работы ИИ-моделей. Иногда они применяются для тестирования качества работы и живых аннотаторов.

Такие наборы называются «золотыми наборами» (Gold Set).

Gold Set — идеальный золотой стандарт

Выборка тщательно размеченных изображений, точно описывающих то, как выглядит эталонно верная разметка, называется gold set.

Эти наборы изображений используются как уменьшенные наборы тестирования для живых аннотаторов, или как часть вводного обучения, или их примешивают к задачам разметки, чтобы убедиться, что качество работы аннотатора не ухудшается из-за его плохой работы или из-за изменения инструкций. Gold sets обычно позволяют проверять следующие аспекты:

Время выполнения задачи.

Точность каждой аннотации

Рост качества работы с повышением опыта

Ухудшение качества работы после изменения инструкций

Слепые этапы — несколько проходов, выполняемых разными аннотаторами

Слепые этапы — это задачи аннотирования, при которых несколько людей (или моделей) присваивают метки независимо друг от друга; при этом этап считается пройденным, только если все они сойдутся на одном результате.

Слепые этапы используются для создания сверхточных данных обучения и автоматизации проверок контроля качества. Аннотаторы очень часто пропускают объекты, но двое или несколько аннотаторов пропустят один и тот же объект с меньшей вероятностью.

Разметка на слепых этапах выполняется параллельно, и каждый участник не может видеть прогресс остальных.
Когда все аннотаторы завершат свою версию задания, оно проходит через проверку консенсусом, валидирующую, что аннотаторы сошлись на одном решении. Если это не так или если решения недостаточно совпадают друг с другом пространственно, то задача передаётся живому контролёру, которые вносит исправления; при этом совершивший ошибку аннотатор уведомляется, чтобы он мог улучшить свою работу.
Если у вас есть 1 миллион размеченных примеров каких-нибудь данных, то вы будете одним из лидеров среди команд разработчиков ИИ. Некоторые компании сегодня обучают модели на миллиардах изображений, видео и аудиосэмплов. Эти наборы данных имеют множество наборов тестирования и многократно размечены для увеличения масштабов задач. Да, теоретически вы можете обучить модель на 100 примерах каких-то данных. Например, V7 позволяет обучить модель всего на 100 примерах, однако на новых примерах качество будет довольно низким.
Высококачественные модели обучаются на больших объёмах данных обучения, и на то есть причины — современные архитектуры нейронных сетей великолепно работают, потому что способны эффективно хранить множество весов (параметров). Однако если у вас мало данных обучения, то вы сможете использовать только долю потенциала своей модели. Размер набора данных также зависит от области вашей задачи и дисперсии каждого класса. Если вы планируете распознать каждый батончик Mars в мире, то дисперсия исчерпается после 10000 примеров. Модель научится каждому возможному углу, типу освещения и степени помятости упаковки. Однако если вам требуется обобщённый распознаватель людей, то набор из 10000 сэмплов будет только небольшой частью вариаций размеров, поз и типов внешностей и одежды. Поэтому для класса с высокой дисперсией, например, «человек», требуется гораздо больше данных обучения.

Размер имеющегося корпуса сырых данных

Если сырые данные отсутствуют, то обеспечьте возможность их сбора в том масштабе, который необходим в вашем случае. Например, если вы работаете на логистическую компанию, обрабатывающую по 10000 счетов в день, то это хороший показатель того, каким должен быть набор данных тестирования, то есть набор данных обучения должен состоять не менее чем из 100000 файлов, чтобы можно было точно сравнивать его с качеством работы человека.

Дисперсия классов

Насколько разрежена репрезентация меток, которые вам нужно распознавать? Если ваша цель заключается в распознавании очень равномерного набора объектов, то вы можете обойтись несколькими тысячами примеров. Если вам нужно идентифицировать что-то разнообразное, например, автомобильные номера всех стран при различных погодных и световых условиях, то для достижения надёжных результатов ваш набор данных должен содержать несколько сотен примеров каждого возможного сценария. Количество классов пропорционально увеличивает требования к размеру наборов данных.

Тип классификации

Вы выполняете объектов или семантическую сегментацию? Каждый тип задачи классификации требует своего количества данных обучения. В задачах распознавания объектов необходимо вычислить количество ожидаемых экземпляров объектов. Если вы ожидаете только одного объекта на файл, то будете обучать модель чуть медленнее, чем при 4-5 объектах.

Текущие изменения

Будет ли ваше распределение (содержимое того, что должен представлять собой набор данных) меняться со временем? Например, если вы создаёте распознаватель мобильных телефонов, то вам придётся периодически добавлять новые данные обучения для учёта новых моделей. Если вы делаете распознаватель лиц, то нужно учесть использование масок. Инструменты тоже меняются со временем — камеры современных телефонов делают HDR-снимки в высоком разрешении, и модели, обученные на таких данных, будут работать качественнее.

Сложность вашей модели

Чем больше весов в вашей модели, тем больше требуется данных обучения. Другими словами, с увеличением сложности модели растёт и необходимый размер датасета. Современные глубокие нейронные сети могут хранить миллионы или даже миллиарды параметров. Следовательно, сложность модели почти никогда не является узким местом — всегда стремитесь к увеличению объёмов данных обучения, чтобы улучшить качество работы вашей модели.

Если вы используете классическое машинное обучение или работаете на устройствах с ограниченными ресурсами, то выгода от многомилионных наборов данных будет минимальной.

Качественные данные обучения:

- В конце давайте повторим всё то, что мы узнали из этого руководства по качественным данным обучения:
- Данными обучения называются данные, применяемые для тренировки алгоритма машинного обучения.
- Точность модели зависит от используемых данных — подавляющую часть времени дата-инженер занимается подготовкой качественных данных обучения.
- Контролируемое обучение использует размеченные данные, а неконтролируемое использует сырые, неразмеченные данные.
- Необходимы высококачественные наборы данных для обучения и валидации и отдельный независимый набор данных для тестирования.
- Золотой набор (gold set) — это выборка точно размеченных данных, идеально отображающая то, как должна выглядеть правильная разметка.
- Чтобы достичь качественных результатов, необходим существенный объём данных обучения для репрезентации всех возможных случаев с не менее чем 1000 сэмплов данных.

- 4 характеристики качественных данных обучения: релевантность содержимого, постоянство, однородность и полнота.
- Используемые вами инструменты очистки, разметки и аннотирования данных играют важнейшую роль в том, чтобы готовую модель можно было использовать в реальных условиях.

8. Обработка данных. Закономерности и аномалии. Визуализация данных

8.1. Обзор набора данных

В начале настроим окружение: импортируем все необходимые библиотеки и немного настроим дефолтное отображение картинок.

```
# отключим предупреждения Anaconda
import warnings
warnings.simplefilter('ignore')

# будем отображать графики прямо в jupyter'e
%matplotlib inline
import seaborn as sns
import matplotlib.pyplot as plt
#графики в svg выглядят более четкими
%config InlineBackend.figure_format = 'svg'

#увеличим дефолтный размер графиков
from pylab import rcParams
rcParams['figure.figsize'] = 8, 5
import pandas as pd
```

После этого загрузим в DataFrame данные, с которыми будем работать. Для примеров я выбрала данные о продажах и оценках видеоигр из Kaggle Datasets. (https://www.kaggle.com/rush4ratio/video-game-sales-with-ratings)

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16719 entries, 0 to 16718
Data columns (total 16 columns):
Name            16717 non-null object
Platform        16719 non-null object
Year_of_Release    16450 non-null float64
Genre           16717 non-null object
Publisher       16665 non-null object
NA_Sales        16719 non-null float64
EU_Sales        16719 non-null float64
JP_Sales        16719 non-null float64
Other_Sales     16719 non-null float64
```

```
Global_Sales      16719 non-null float64
Critic_Score      8137 non-null float64
Critic_Count      8137 non-null float64
User_Score        10015 non-null object
User_Count        7590 non-null float64
Developer         10096 non-null object
Rating            9950 non-null object
dtypes: float64(9), object(7)
memory usage: 2.0+ MB
```

Некоторые признаки, которые pandas считал, как object, явно приведем к типам float или int.

```
df['User_Score'] = df.User_Score.astype('float64')
df['Year_of_Release'] = df.Year_of_Release.astype('int64')
df['User_Count'] = df.User_Count.astype('int64')
df['Critic_Count'] = df.Critic_Count.astype('int64')
```

Данные есть не для всех игр, поэтому давайте оставим только те записи, в которых нет пропусков, с помощью метода dropna.

```
df = df.dropna()
print(df.shape)
```

```
(6825, 16)
```

Всего в таблице 6825 объектов и 16 признаков для них. Посмотрим на несколько первых записей с помощью метода head, чтобы убедиться, что все распарсилось правильно. Для удобства я оставила только те признаки, которые мы будем в дальнейшем использовать.

```
useful_cols = ['Name', 'Platform', 'Year_of_Release', 'Genre',
               'Global_Sales', 'Critic_Score', 'Critic_Count',
               'User_Score', 'User_Count', 'Rating'
              ]
df[useful_cols].head()
```

	Name	Platform	Year_of_Release	Genre	Global_Sales	Critic_Score	Critic_Count	User_Score	User_Count	Rating
0	Wii Sports	Wii	2006	Sports	82.53	76.0	51.0	8.0	322.0	E
2	Mario Kart Wii	Wii	2008	Racing	35.52	82.0	73.0	8.3	709.0	E
3	Wii Sports Resort	Wii	2009	Sports	32.77	80.0	73.0	8.0	192.0	E
6	New Super Mario Bros.	DS	2006	Platform	29.80	89.0	65.0	8.5	431.0	E
7	Wii Play	Wii	2006	Misc	28.92	58.0	41.0	6.6	129.0	E

Прежде чем мы перейдем к рассмотрению методов библиотек seaborn и plotly, обсудим самый простой и зачастую удобный способ визуализировать данные из pandas DataFrame — это воспользоваться функцией plot.

Для примера построим график продаж видео игр в различных странах в зависимости от года. Для начала отфильтруем только нужные нам столбцы, затем посчитаем суммарные продажи по годам и у получившегося DataFrame вызовем функцию plot без параметров.

```
sales_df = df[[x for x in df.columns if 'Sales' in x] + ['Year_of_Release']]
sales_df.groupby('Year_of_Release').sum().plot()
```

Реализация функции plot в pandas основана на библиотеке matplotlib.

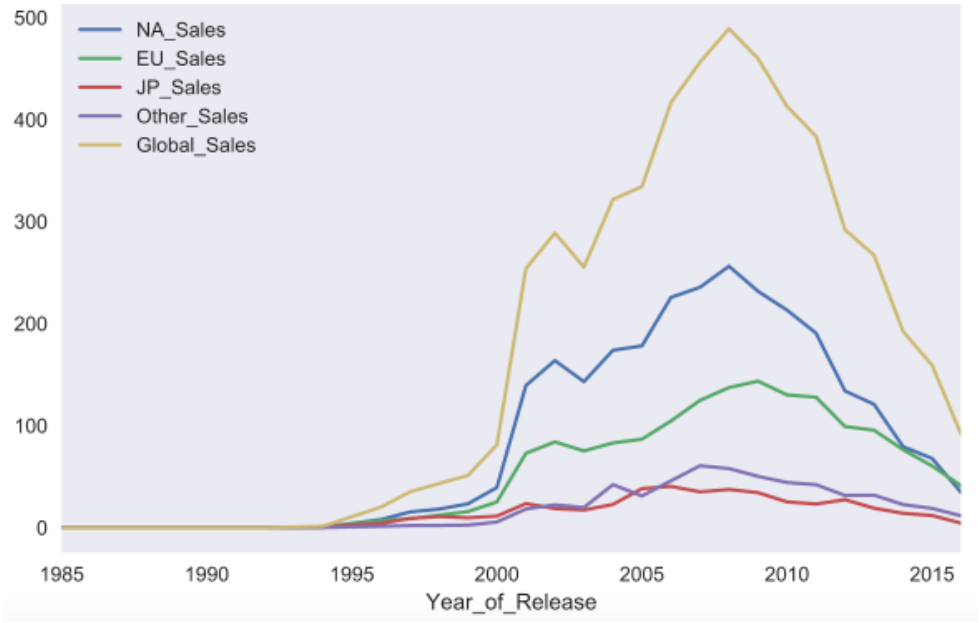

Рис.21 График функциональной зависимости

С помощью параметра kind можно изменить тип графика, например, на bar chart. Matplotlib позволяет очень гибко настраивать графики. На графике можно изменить почти все, что угодно, но потребуется порыться в документации и найти нужные параметры. Например, параметр rot отвечает за угол наклона подписей к оси x.

```
sales_df.groupby('Year_of_Release').sum().plot(kind='bar', rot=45)
```

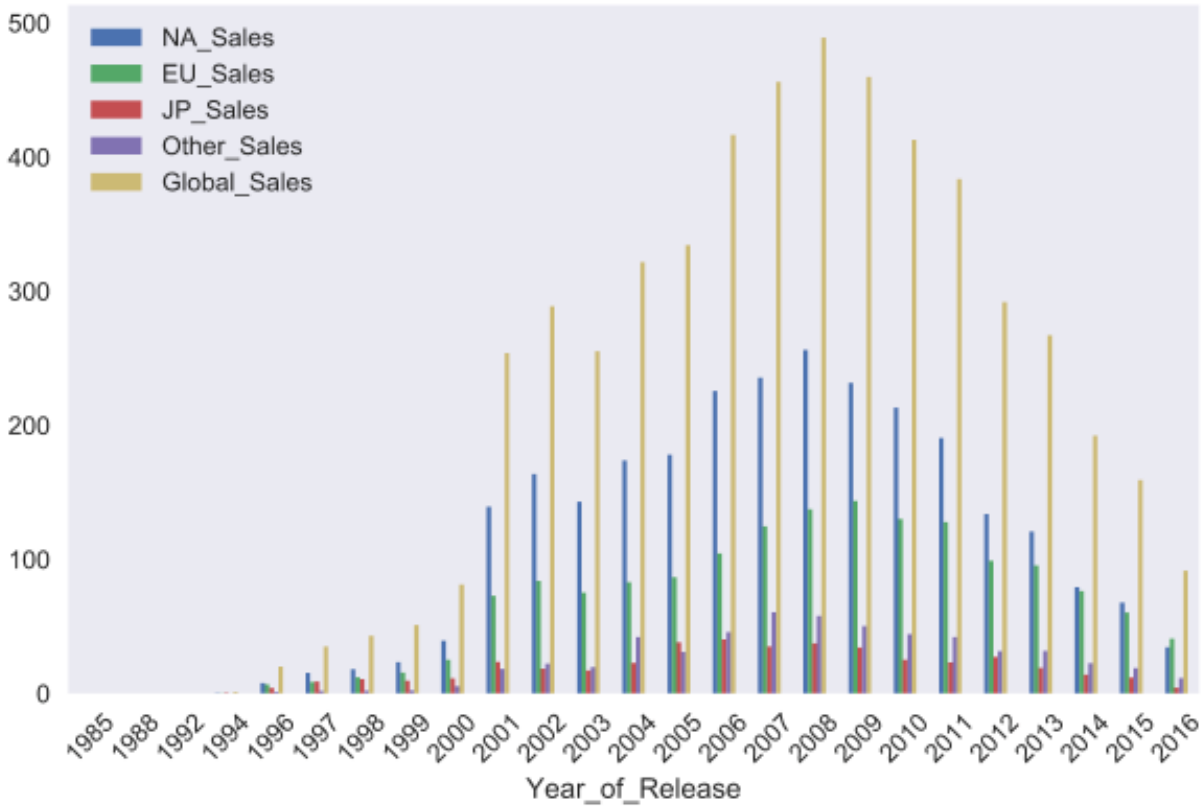

Рис.22 Столбцовый график функциональной зависимости

8.2. Seaborn

Теперь давайте перейдем к библиотеке seaborn. Seaborn — это, по сути, более высокоуровневое API на базе библиотеки matplotlib. Seaborn содержит более адекватные дефолтные настройки оформления графиков. Также в библиотеке есть достаточно сложные типы визуализации, которые в matplotlib потребовали бы большого количество кода.
Познакомимся с первым таким "сложным" типом графиков pair plot (scatter plot matrix). Эта визуализация поможет нам посмотреть на одной картинке, как связаны между собой различные признаки.

```
cols = ['Global_Sales', 'Critic_Score', 'Critic_Count', 'User_Score', 'User_Count']
sns_plot = sns.pairplot(df[cols])
sns_plot.savefig('pairplot.png')
```

Как можно видеть, на диагонали матрицы графиков расположены гистограммы распределений признака. Остальные же графики — это обычные scatter plots для соответствующих пар признаков.
Для сохранения графиков в файлы стоит использовать метод savefig.

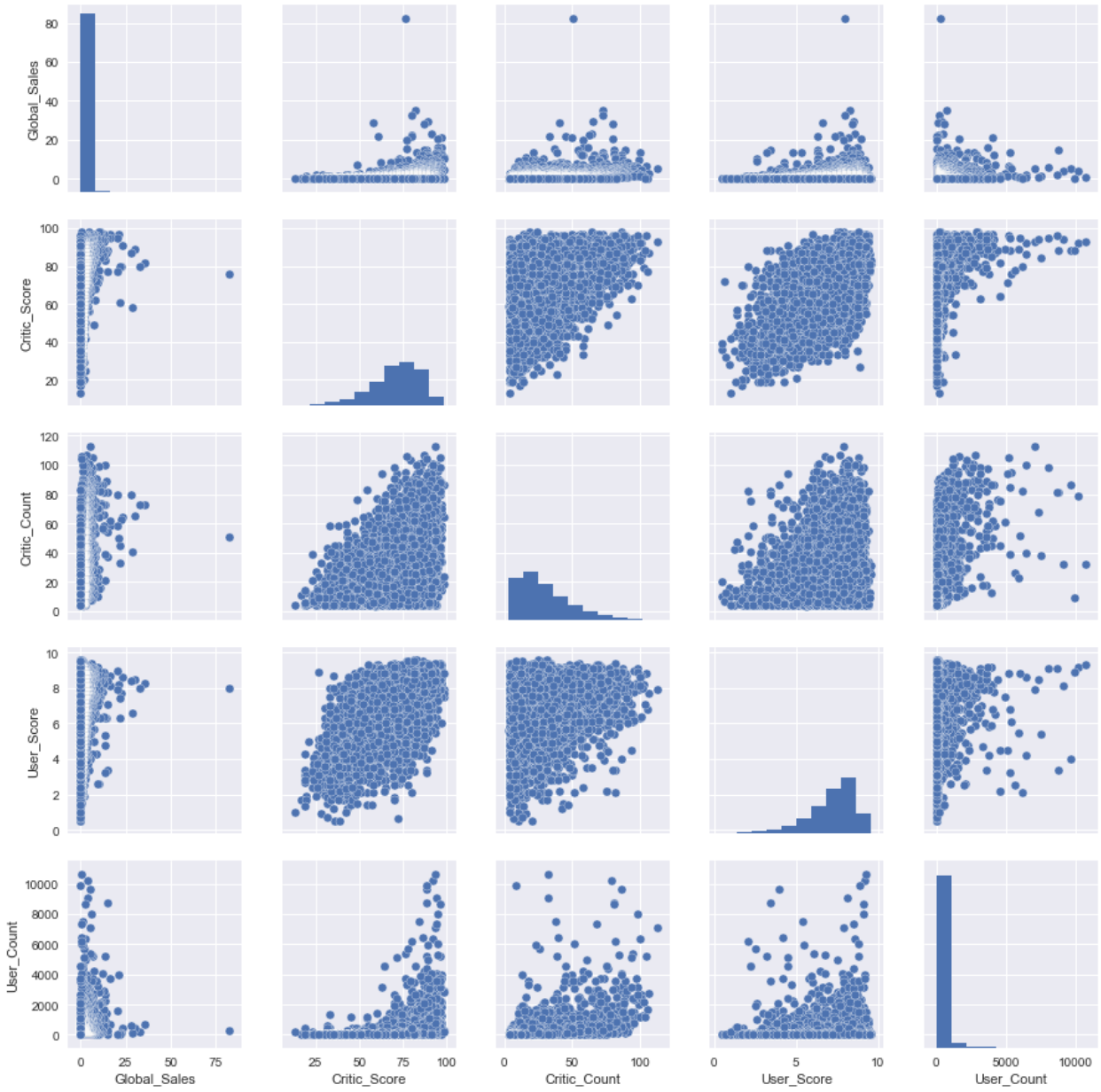

Рис.23 Гистограммы распределения по признакам

С помощью seaborn можно построить и распределение dist plot. Для примера посмотрим на распределение оценок критиков Critic_Score. По умолчанию на графике отображается гистограмма и kernel density estimation.

```
sns.distplot(df.Critic_Score)
```

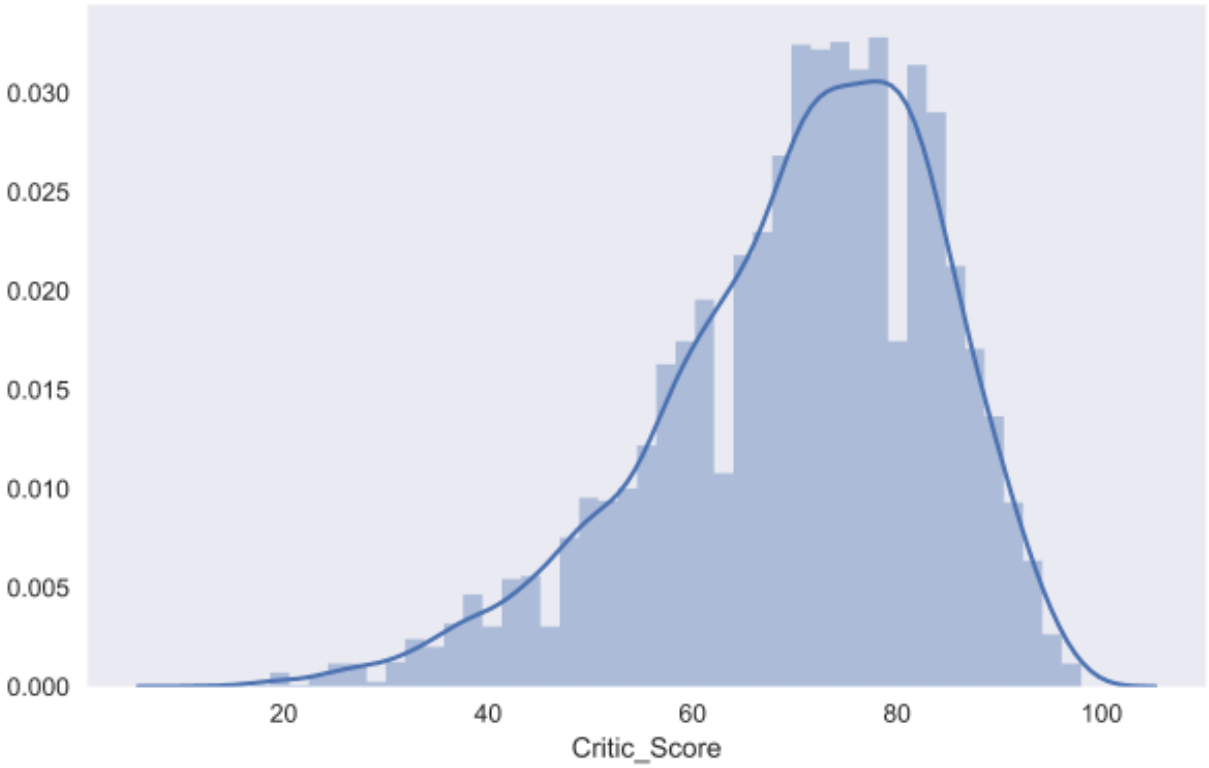

Рис.24 Гистограмма распределения признака

Для того, чтобы подробнее посмотреть на взаимосвязь двух численных признаков, есть еще и joint plot — это гибрид scatter plot и histogram. Посмотрим на то, как связаны между собой оценка критиков Critic_Score и оценка пользователя User_Score.

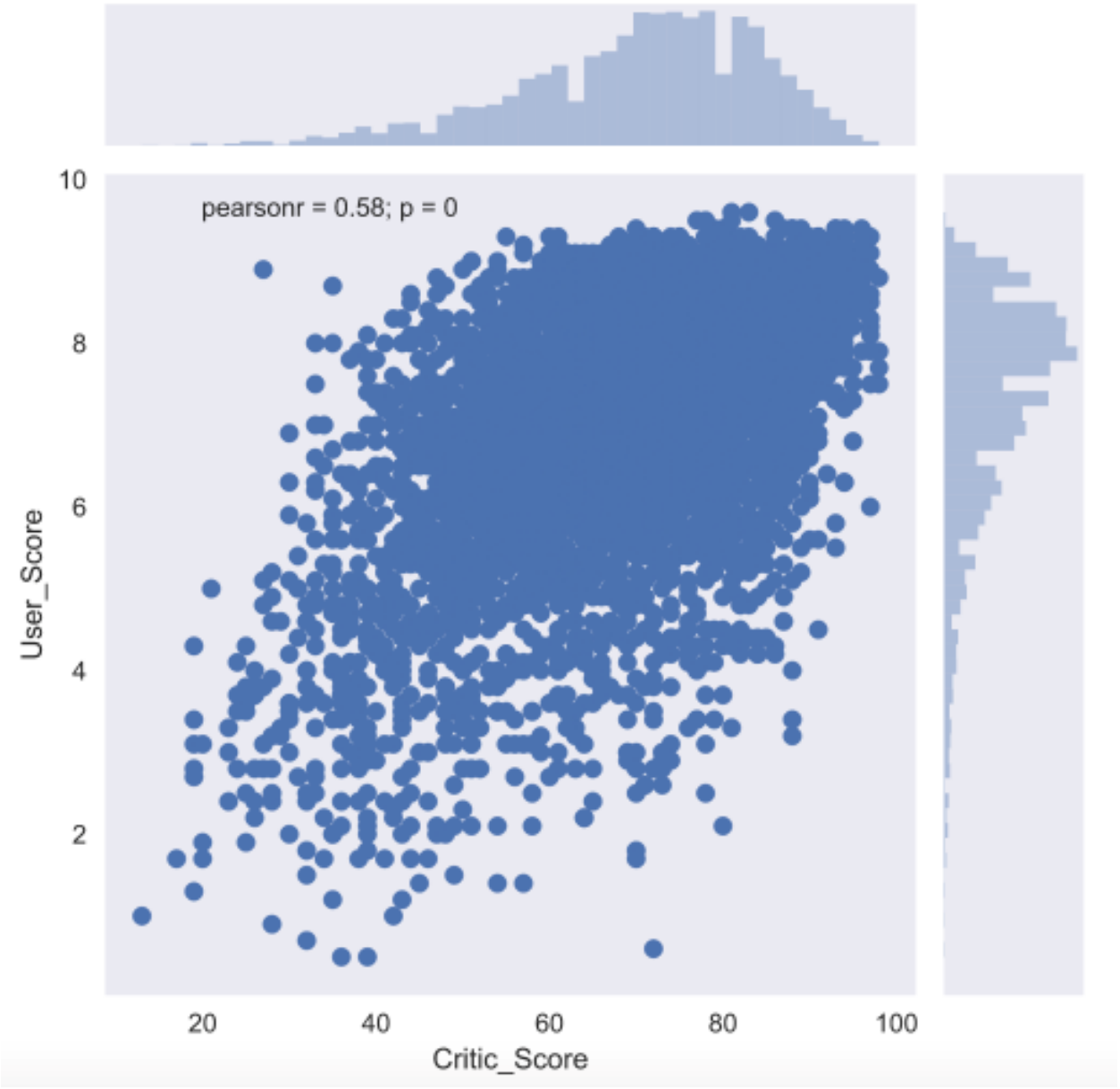

Рис.25 Гистограммы по двум признакам

Еще один полезный тип графиков — это box plot. Давайте сравним оценки игр от критиков для топ-5 крупнейших игровых платформ.

```
top_platforms = df.Platform.value_counts().sort_values(ascending = False).head(5).index.values
sns.boxplot(y="Platform", x="Critic_Score", data=df[df.Platform.isin(top_platforms)], orient="h")
```

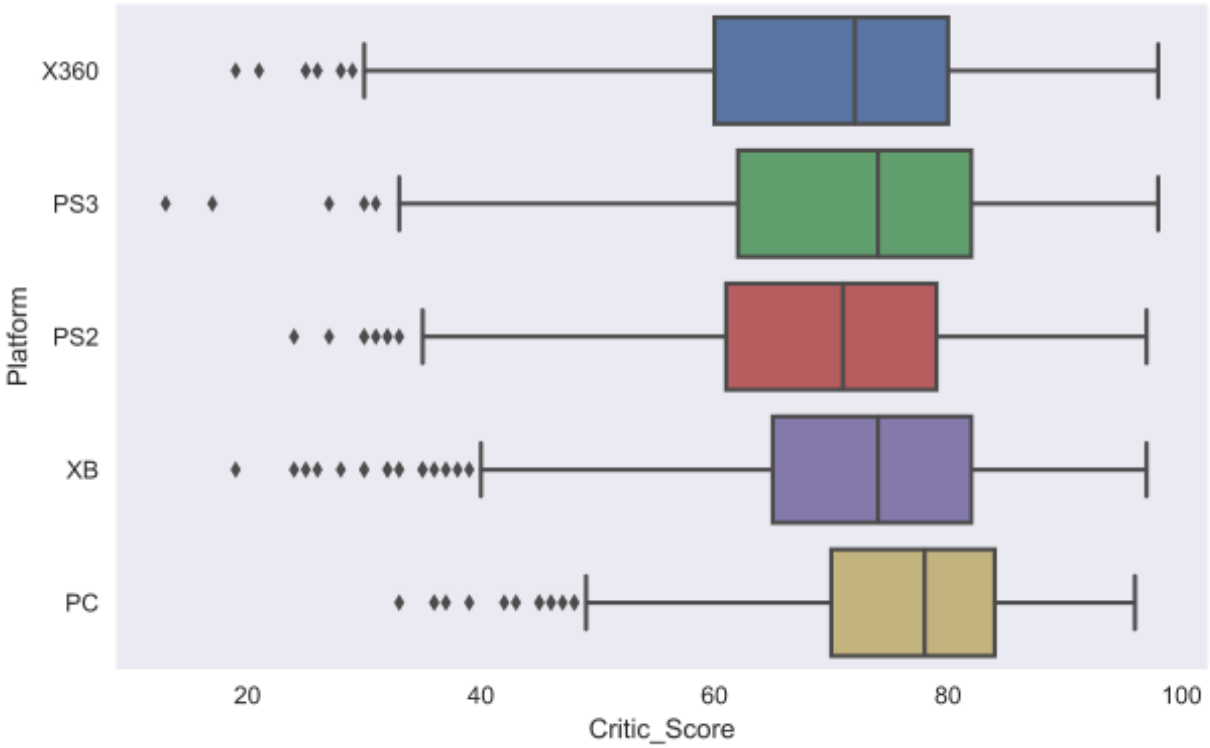

Рис.26 Графики типа «свеча» или «коробка»

Стоит обсудить немного подробнее, как же понимать box plot. Box plot состоит из коробки (поэтому он и называется box plot), усиков и точек. Коробка показывает интерквартильный размах распределения, то есть соответственно 25% (Q1) и 75% (Q3) перцентили. Черта внутри коробки обозначает медиану распределения.

С коробкой разобрались, перейдем к усам. Усы отображают весь разброс точек кроме выбросов, то есть минимальные и максимальные значения, которые попадают в промежуток (Q1 - 1.5*IQR, Q3 + 1.5*IQR), где IQR = Q3 - Q1 — интерквартильный размах. Точками на графике обозначаются выбросы (outliers) — те значения, которые не вписываются в промежуток значений, заданный усами графика.

Для понимания лучше один раз увидеть, поэтому вот еще и картинка с Wikipedia:

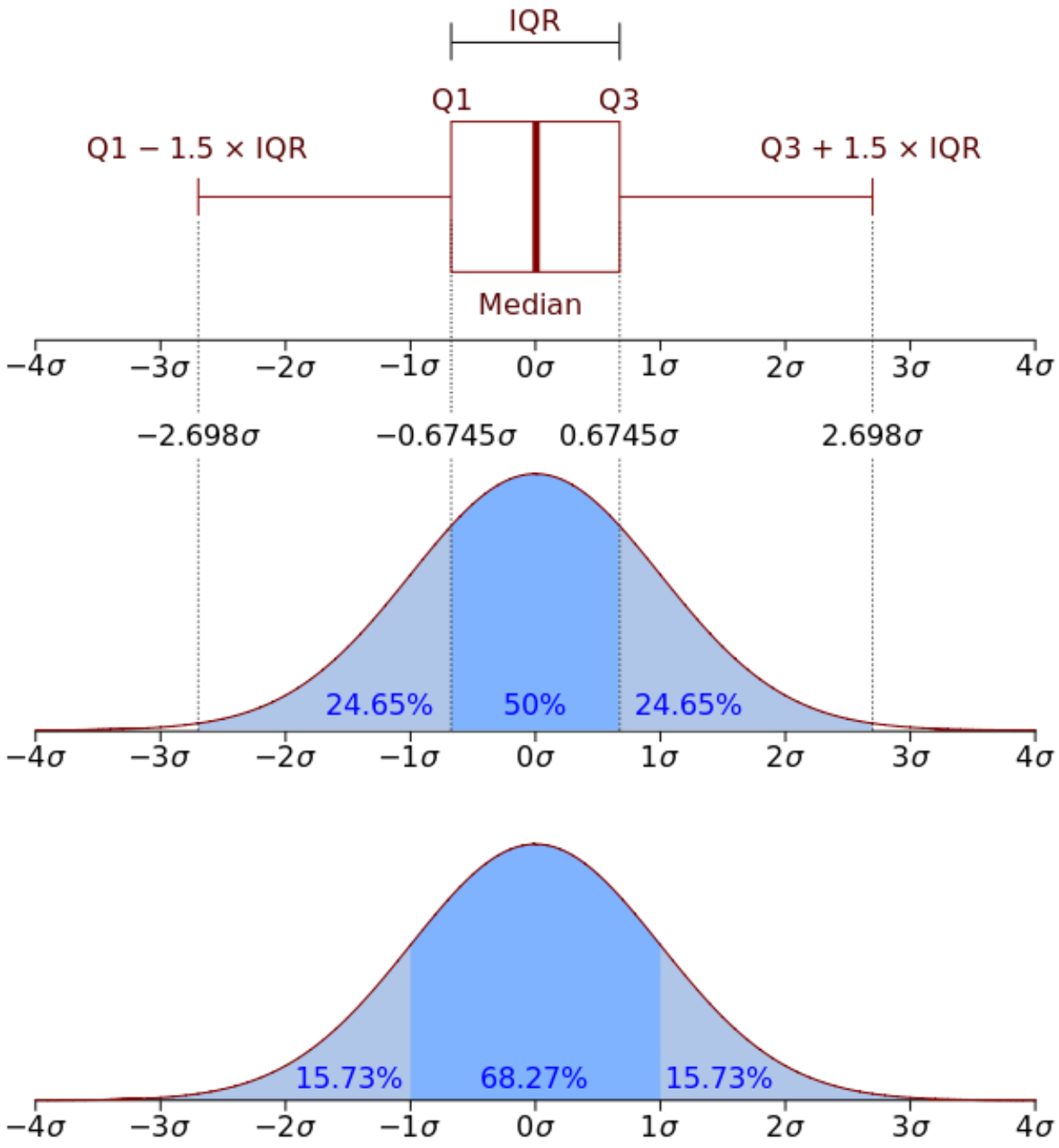

Рис.27 Медианы и квартили

И еще один тип графиков (последний из тех, которые мы рассмотрим в этой статье) — это heat map. Heat map позволяет посмотреть на распределение какого-то численного признака по двум категориальным. Визуализируем суммарные продажи игр по жанрам и игровым платформам.

```
platform_genre_sales = df.pivot_table(
                        index='Platform',
                        columns='Genre',
                        values='Global_Sales',
                        aggfunc=sum).fillna(0).applymap(float)
```

```
sns.heatmap(platform_genre_sales, annot=True, fmt=".1f", linewidths=.5)
```

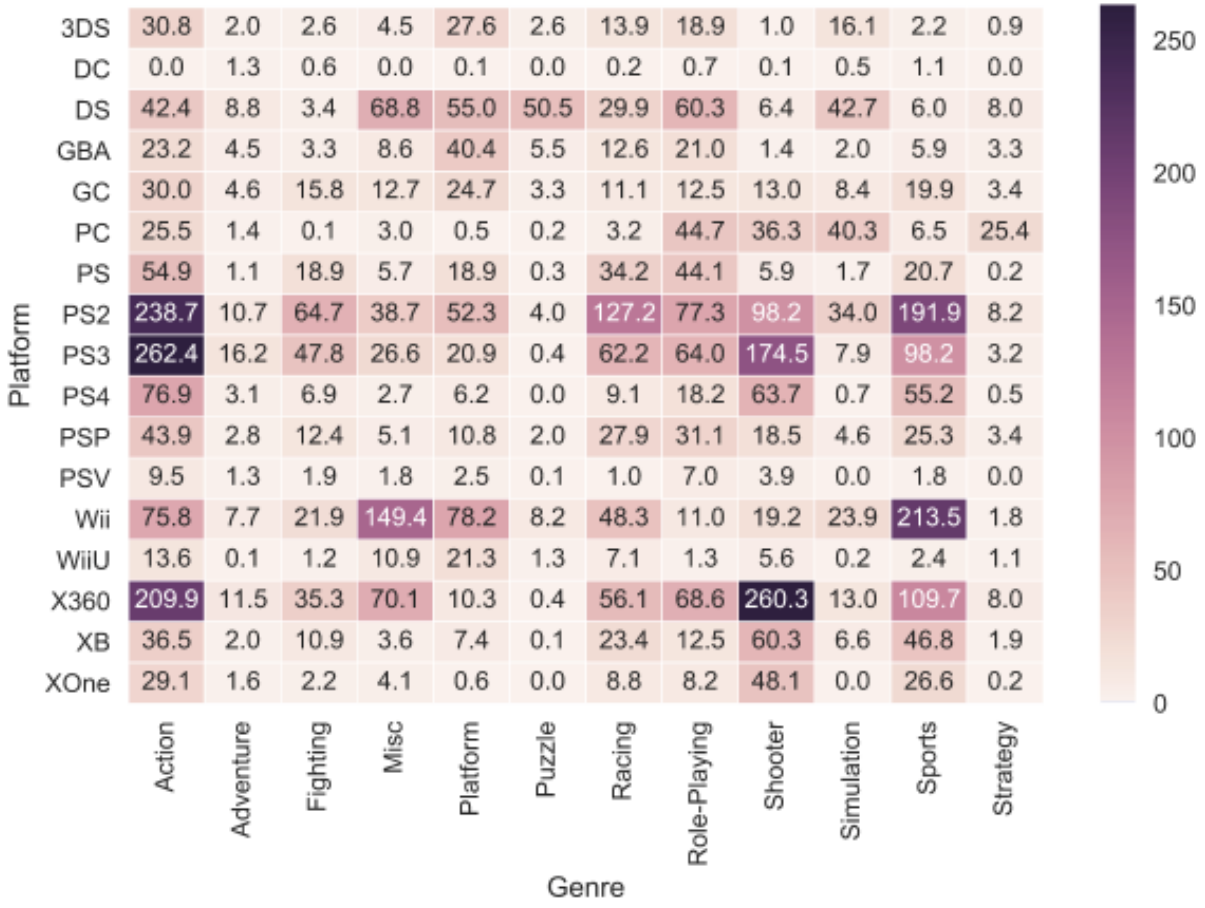

Рис.28 Тепловая карта по признакам

8.1. Plotly

Мы рассмотрели визуализации на базе библиотеки matplotlib. Однако это не единственная опция для построения графиков на языке python. Познакомимся также с библиотекой plotly. Plotly — это open-source библиотека, которая позволяет строить интерактивные графики в jupyter.notebook'e без необходимости зарываться в javascript код.

Прелесть интерактивных графиков заключается в том, что можно посмотреть точное численное значение при наведении мыши, скрыть неинтересные ряды в визуализации, приблизить определенный участок графика и т.д.

Перед началом работы импортируем все необходимые модули и инициализируем plotly с помощью команды init_notebook_mode.

```
from plotly.offline import download_plotlyjs, init_notebook_mode, plot, iplot
import plotly
import plotly.graph_objs as go

init_notebook_mode(connected=True)
```

Для начала построим line plot с динамикой числа вышедших игр и их продаж по годам.

```
# посчитаем число вышедших игр и проданных копий по годам
years_df = df.groupby('Year_of_Release')[['Global_Sales']].sum().join(
    df.groupby('Year_of_Release')[['Name']].count()
)
years_df.columns = ['Global_Sales', 'Number_of_Games']

# создаем линию для числа проданных копий
trace0 = go.Scatter(
    x=years_df.index,
    y=years_df.Global_Sales,
    name='Global Sales'
)

# создаем линию для числа вышедших игр
trace1 = go.Scatter(
    x=years_df.index,
    y=years_df.Number_of_Games,
    name='Number of games released'
)

# определяем массив данных и задаем title графика в layout
data = [trace0, trace1]
layout = {'title': 'Statistics of video games'}

# создаем объект Figure и визуализируем его
fig = go.Figure(data=data, layout=layout)
iplot(fig, show_link=False)
```

В plotly строится визуализация объекта Figure, который состоит из данных (массив линий, которые в библиотеке называются traces) и оформления/стиля, за который отвечает объект layout. В простых случаях можно вызывать функцию iplot и просто от массива traces.
Параметр show_link отвечает за ссылки на online-платформу plot.ly на графиках. Поскольку обычно это функциональность не нужна, то я предпочитаю скрывать ее для предотвращения случайных нажатий.

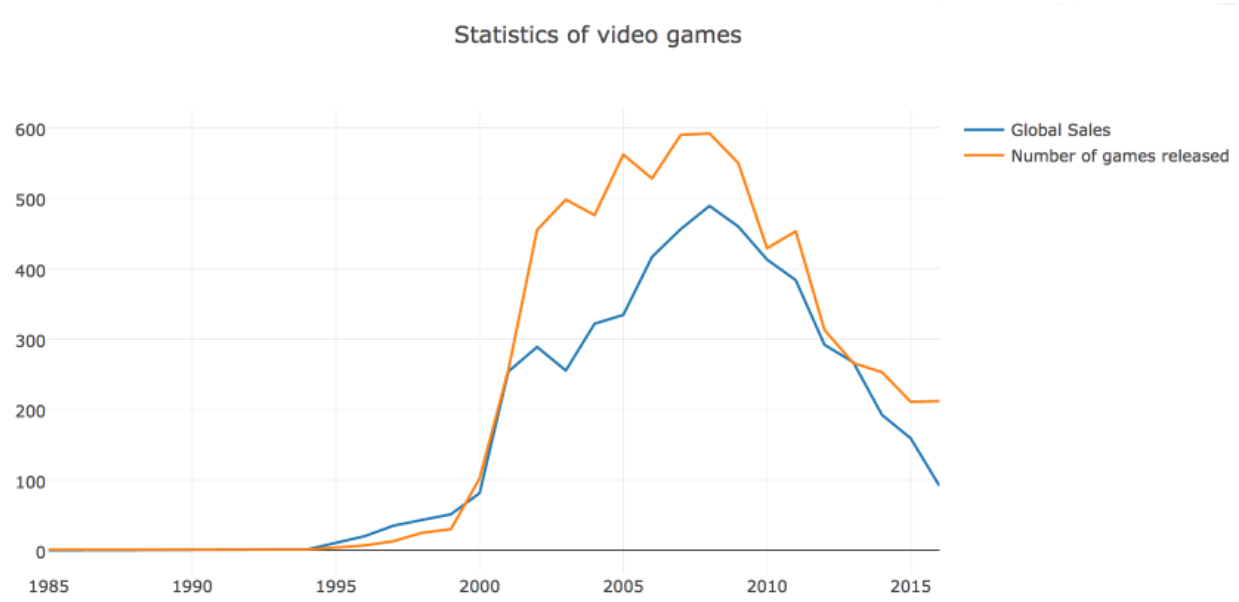

Рис.29 График функциональной зависимости

Можно сразу сохранить график в виде html-файла.

```
plotly.offline.plot(fig, filename='years_stats.html', show_link=False)
```

Посмотрим также на рыночную долю игровых платформ, рассчитанную по количеству выпущенных игр и по суммарной выручке. Для этого построим bar chart.

```
# считаем число проданных и вышедших игр по платформам
platforms_df = df.groupby('Platform')[['Global_Sales']].sum().join(
    df.groupby('Platform')[['Name']].count()
)
platforms_df.columns = ['Global_Sales', 'Number_of_Games']
platforms_df.sort_values('Global_Sales', ascending=False, inplace=True)

# создаем traces для визуализации
trace0 = go.Bar(
    x=platforms_df.index,
    y=platforms_df.Global_Sales,
    name='Global Sales'
)

trace1 = go.Bar(
    x=platforms_df.index,
    y=platforms_df.Number_of_Games,
    name='Number of games released'
)

# создаем массив с данными и задаем title для графика и оси x в layout
data = [trace0, trace1]
```

```
layout = {'title': 'Share of platforms', 'xaxis': {'title': 'platform'}}

# создаем объект Figure и визуализируем его
fig = go.Figure(data=data, layout=layout)
iplot(fig, show_link=False)
```

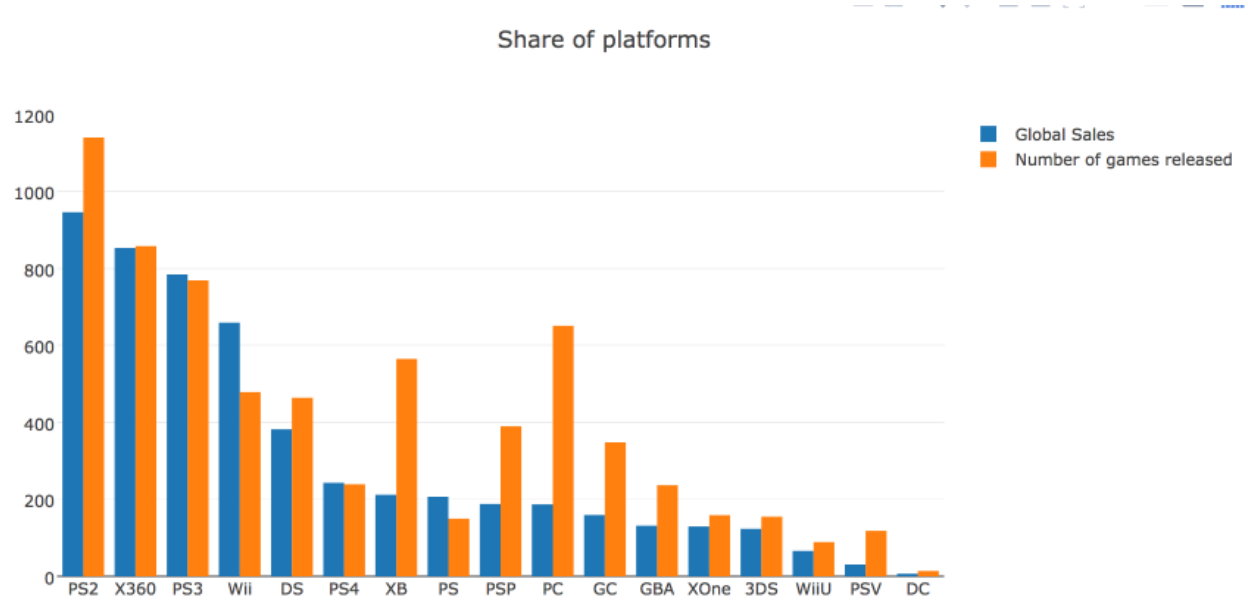

Рис.30 Столбцовый график функциональной зависимости

В plotly можно построить и box plot. Рассмотрим распределения оценок критиков в зависимости от жанра игры.

```
# создаем Box trace для каждого жанра из наших данных
data = []
for genre in df.Genre.unique():
    data.append(
        go.Box(y=df[df.Genre==genre].Critic_Score, name=genre)
    )

# визуализируем данные
iplot(data, show_link = False)
```

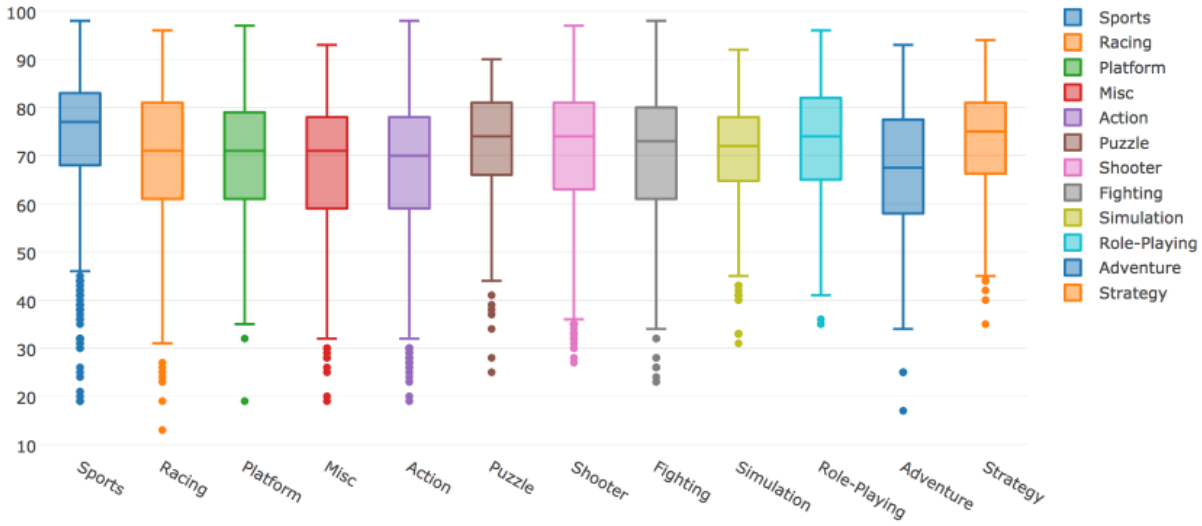

Рис.31 Графики типа «свеча» или «коробка»

С помощью plotly можно построить и другие типы визуализаций. Графики получаются достаточно симпатичными с дефолтными настройками. Однако библиотека позволяет и гибко настраивать различные параметры визуализации: цвета, шрифты, подписи, аннотации и многое другое.

8.2. Пример визуального анализа данных

Считываем в DataFrame знакомые нам по первой статье данные по оттоку клиентов телеком-оператора.

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

Проверим, все ли нормально считалось – посмотрим на первые 5 строк (метод head).

```
df.head()
```

Таблица.1 Вывод данных

	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

Число строк (клиентов) и столбцов (признаков):

```
df.shape
```

```
(3333, 20)
```

Посмотрим на признаки и убедимся, что пропусков ни в одном из них нет – везде по 3333 записи.

```
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
```

Посмотрим на распределение целевого класса – оттока клиентов.

```
df['Churn'].value_counts()
```

```
False    2850
True      483
```

```
Name: Churn, dtype: int64
```

```
df['Churn'].value_counts().plot(kind='bar', label='Churn')
plt.legend()
plt.title('Распределение оттока клиентов');
```

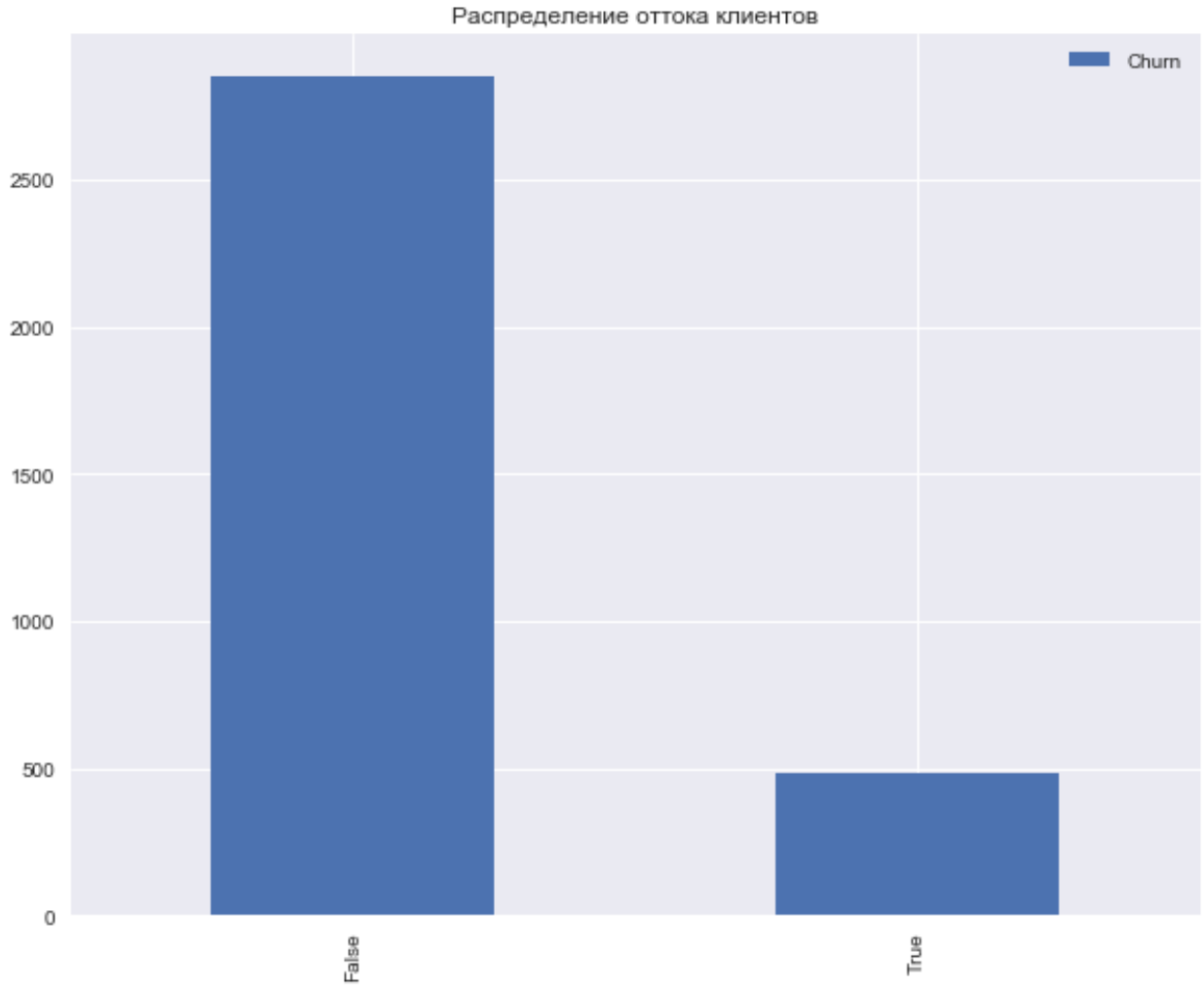

Рис.32 Гистограмма гаспределения целевой переменной

Выделим следующие группы признаков (среди всех кроме *Churn*):

- бинарные: *International plan, Voice mail plan*
- категориальные: *State*
- порядковые: *Customer service calls*
- количественные: все остальные

Посмотрим на корреляции количественных признаков. По раскрашенной матрице корреляций видно, что такие признаки как *Total day charge* считаются по проговоренным минутам (*Total day minutes*). То есть 4 признака можно выкинуть, они не несут полезной информации.

```
corr_matrix = df.drop(['State', 'International plan', 'Voice mail plan',
                       'Area code'], axis=1).corr()
```

```
sns.heatmap(corr_matrix);
```

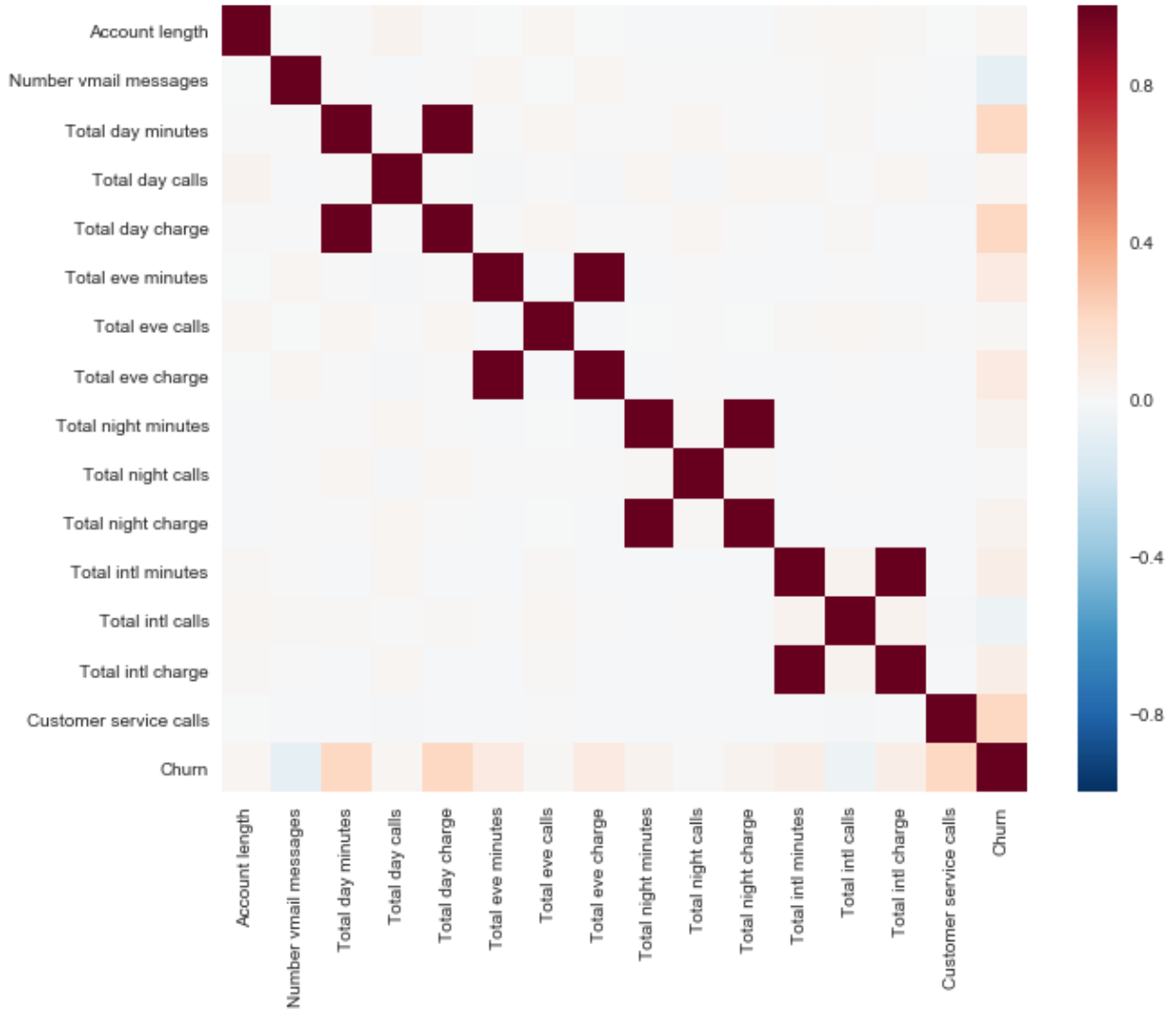

Рис.33 Тепловая карта признаков

Теперь посмотрим на распределения всех интересующих нас количественных признаков. На бинарные/категориальные/порядковые признакие будем смотреть отдельно.

```
features = list(set(df.columns) - set(['State', 'International plan', 'Voice mail plan',  'Area code',
                                       'Total day charge',   'Total eve charge',   'Total night charge',
                                        'Total intl charge', 'Churn']))
```

```
df[features].hist(figsize=(20,12));
```

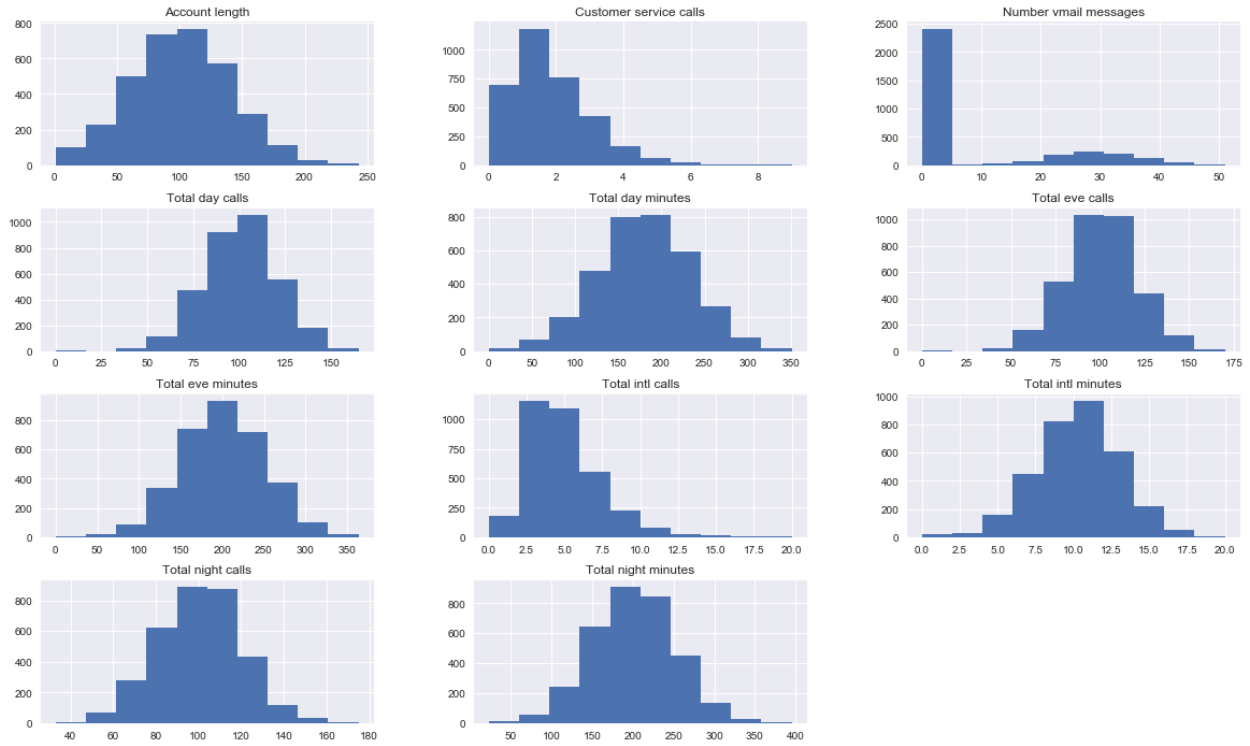

Рис.34 Гистограммы по признакам

Видим, что большинство признаков распределены нормально. Исключения – число звонков в сервисный центр (*Customer service calls*) (тут больше подходит пуассоновское распределение) и число голосовых сообщений (*Number vmail messages*, пик в нуле, т.е. это те, у кого голосовая почта не подключена). Также смещено распределение числа международных звонков (*Total intl calls*).
Еще полезно строить вот такие картинки, где на главной диагонали рисуются распределения признаков, а вне главной диагонали – диаграммы рассеяния для пар признаков. Бывает, что это приводит к каким-то выводам, но в данном случае все примерно понятно, без сюрпризов.

```
sns.pairplot(df[features + ['Churn']], hue='Churn');
```

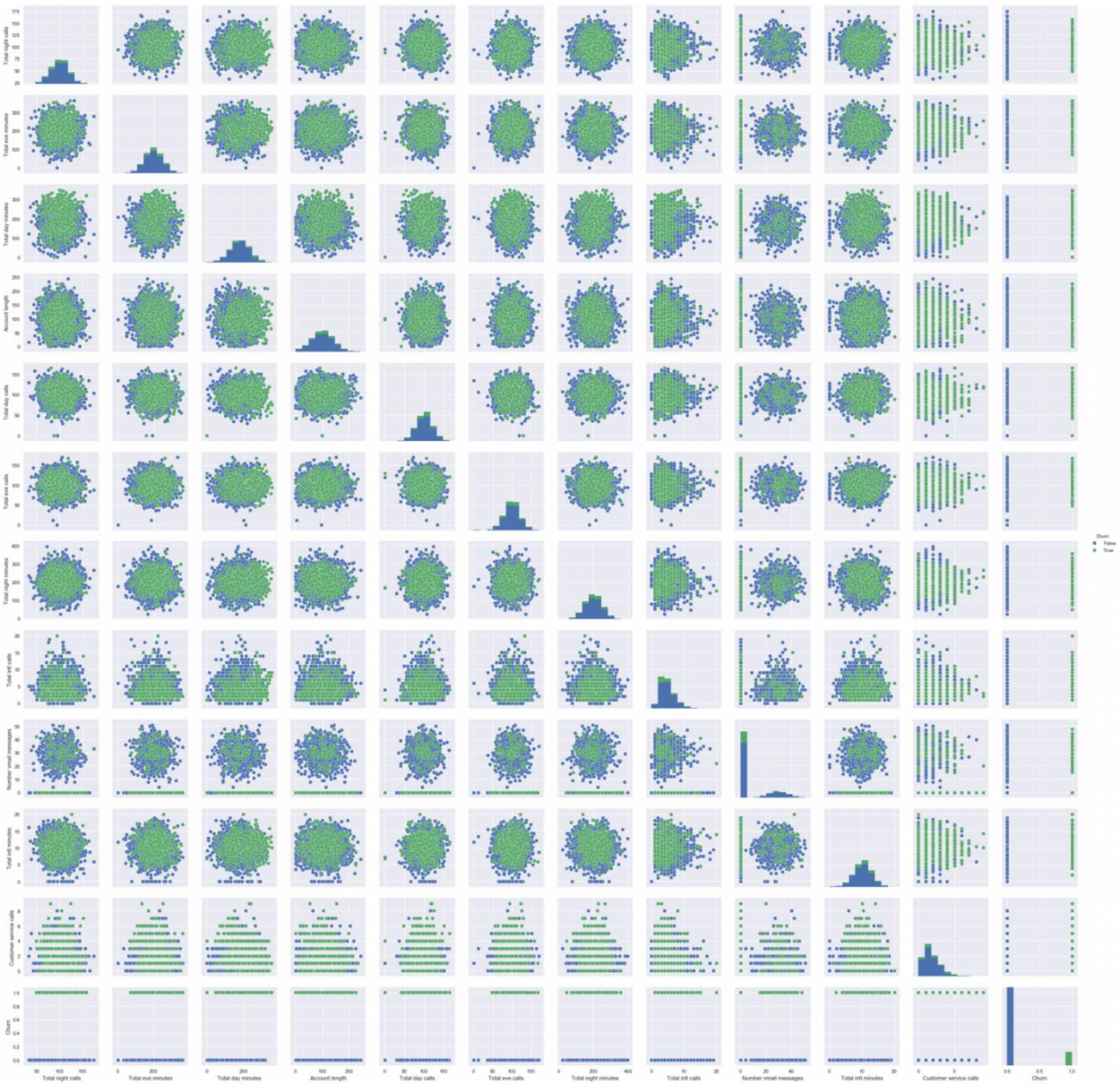

Рис.35 Парные гистограммы

Дальше посмотрим, как признаки связаны с целевым – с оттоком.
Построим boxplot-ы, описывающее статистики распределения количественных признаков в двух группах: среди лояльных и ушедших клиентов.

```
fig, axes = plt.subplots(nrows=3, ncols=4, figsize=(16, 10))

for idx, feat in  enumerate(features):
    sns.boxplot(x='Churn', y=feat, data=df, ax=axes[idx / 4, idx % 4])
    axes[idx / 4, idx % 4].legend()
    axes[idx / 4, idx % 4].set_xlabel('Churn')
    axes[idx / 4, idx % 4].set_ylabel(feat);
```

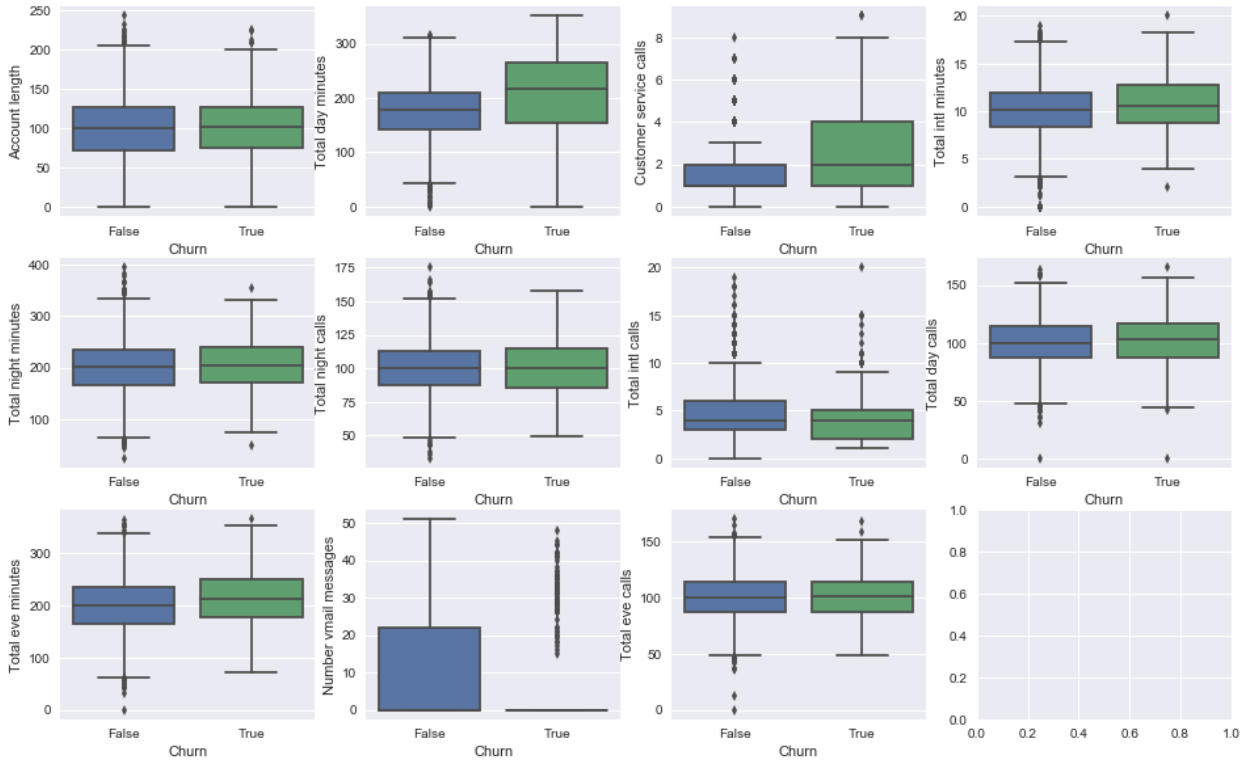

Рис.36 Графики типа «свеча» или «коробка»

На глаз наибольшее отличие мы видим для признаков *Total day minutes*, *Customer service calls* и *Number vmail messages*. Впоследствии мы научимся определять важность признаков в задаче классификации с помощью случайного леса (или градиентного бустинга), и окажется, что первые два – действительно очень важные признаки для прогнозирования оттока.

Посмотрим отдельно на картинки с распределением кол-ва проговоренных днем минут среди лояльных/ушедших. Слева — знакомые нам боксплоты, справа – сглаженные гистограммы распределения числового признака в двух группах (скорее просто красивая картинка, все и так понятно по боксплоту).

Интересное **наблюдение:** в среднем ушедшие клиенты больше пользуются связью. Возможно, они недовольны тарифами, и одной из мер борьбы с оттоком будет понижение тарифных ставок (стоимости мобильной связи). Но это уже компании надо будет проводить дополнительный экономический анализ, действительно ли такие меры будут оправданы.

```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(16,6))

sns.boxplot(x='Churn', y='Total day minutes', data=df, ax=axes[0]);
sns.violinplot(x='Churn', y='Total day minutes', data=df, ax=axes[1]);
```

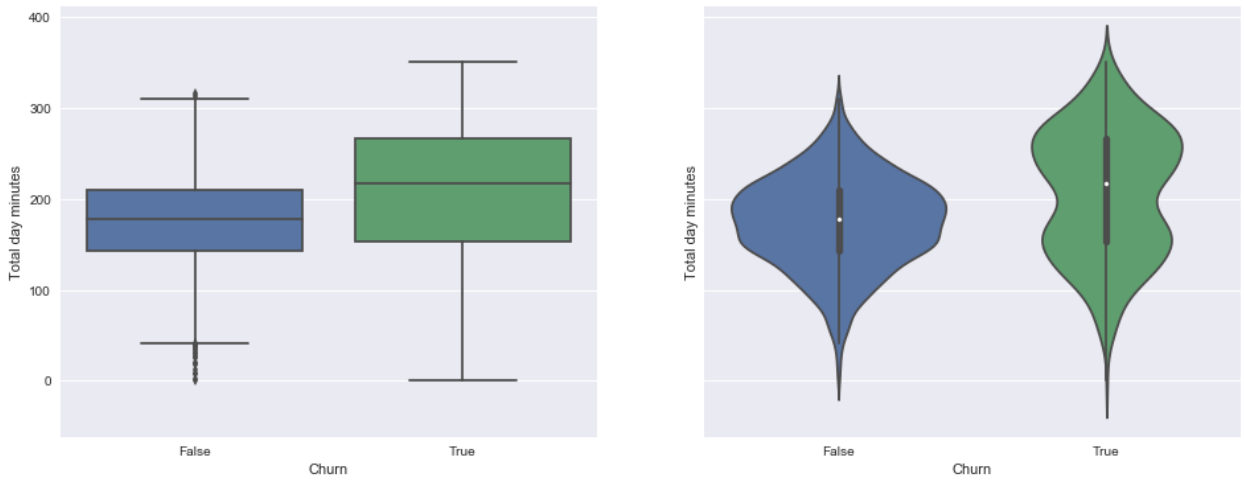

Рис.37 Графики типа «коробка» и «скрипка»

Теперь изобразим распределение числа обращений в сервисный центр (такую картинку мы строили в первой статье). Тут уникальных значений признака не много (признак можно считать как количественным целочисленным, так и порядковым), и наглядней изобразить распределение с помощью countplot.
Наблюдение: доля оттока сильно возрастает начиная с 4 звонков в сервисный центр.

```
sns.countplot(x='Customer service calls', hue='Churn', data=df);
```

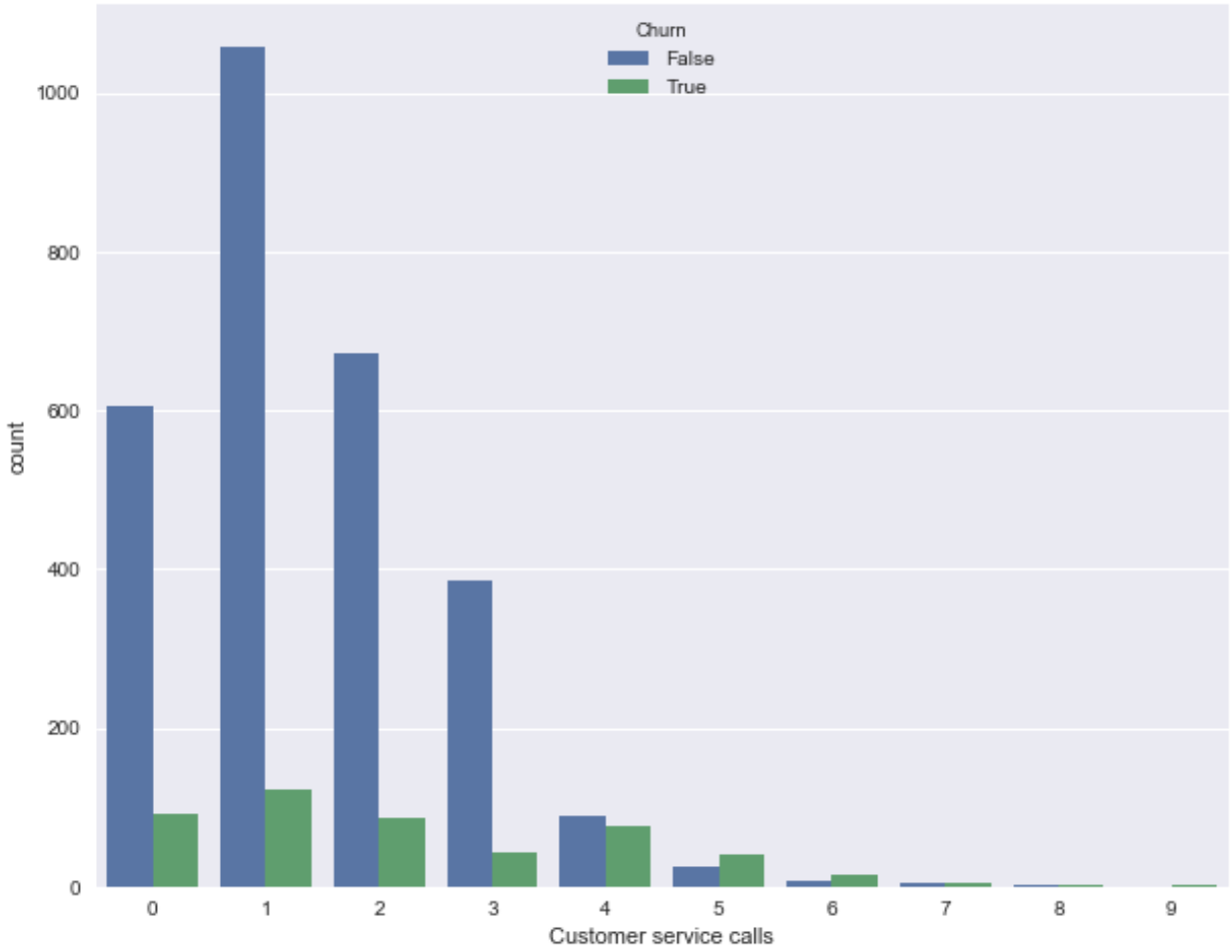

Рис.38 Столбцовый график функциональной зависимости

Теперь посмотрим на связь бинарных признаков *International plan* и *Voice mail plan* с оттоком.

Наблюдение: когда роуминг подключен, доля оттока намного выше, т.е. наличие международного роуминга – сильный признак. Про голосовую почту такого нельзя сказать.

```
_, axes = plt.subplots(1, 2, sharey=True, figsize=(16,6))

sns.countplot(x='International plan', hue='Churn', data=df, ax=axes[0]);
sns.countplot(x='Voice mail plan', hue='Churn', data=df, ax=axes[1]);
```

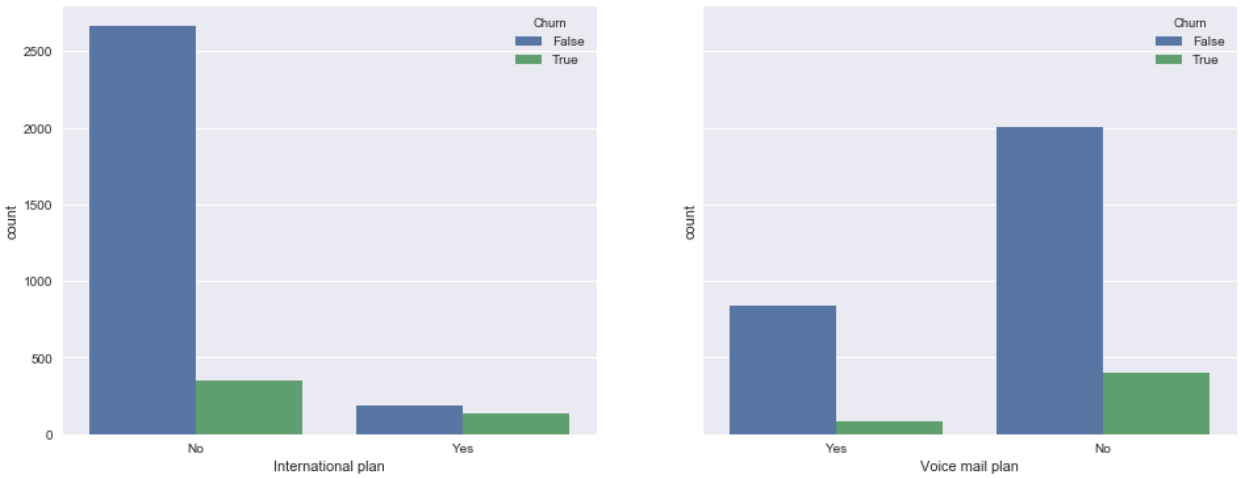

Рис.39 Столбцовый график функциональной зависимости

Наконец, посмотрим, как с оттоком связан категориальный признак *State*. С ним уже не так приятно работать, поскольку число уникальных штатов довольно велико – 51. Можно в начале построить сводную табличку или посчитать процент оттока для каждого штата. Но данных по каждом штату по отдельности маловато (ушедших клиентов всего от 3 до 17 в каждом штате), поэтому, возможно, признак *State* впоследствии не стоит добавлять в модели классификации из-за риска *переобучения* (но мы это будем проверять на *кросс-валидации*, stay tuned!).
Доли оттока для каждого штата:

```
df.groupby(['State'])['Churn'].agg([np.mean]).sort_values(by='mean', ascending=False).T
```

State	NJ	CA	TX	MD	SC	MI	MS	NV	WA	ME
mean	0.264706	0.264706	0.25	0.242857	0.233333	0.219178	0.215385	0.212121	0.212121	0.209677

RI	WI	IL	NE	LA	IA	VA	AZ	AK	HI
0.092308	0.089744	0.086207	0.081967	0.078431	0.068182	0.064935	0.0625	0.057692	0.056604

Видно, что в Нью-Джерси и Калифорнии доля оттока выше 25%, а на Гавайях и в Аляске меньше 5%. Но эти выводы построены на слишком скромной статистике и возможно, это просто особенности имеющихся данных (тут можно и гипотезы попроверять про корреляции Мэтьюса и Крамера, но это уже за рамками данной статьи).