

Определение перспектив продукта игровой компьютерной индустрии

Сборный проект № 1 учебного курса "Специалист по Data Science" от Яндекс.Практикум.

Выполнил **Денис Абрашин** (e-mail) Версия 1.0

Оглавление

- Описание проекта
 - Инструкция по выполнению проекта
 - Шаг 1. Откройте файл с данными и изучите общую информацию
 - Шаг 2. Подготовьте данные
 - Шаг 3. Проведите исследовательский анализ данных
 - Шаг 4. Составьте портрет пользователя каждого региона
 - Шаг 5. Проверьте гипотезы
 - Шаг 6. Напишите общий вывод
 - Требования к оформлению
 - Описание данных
 - Критерии проверки
- Подключение и настройка необходимых библиотек
 - Настройка pandas
 - Настройка matplotlib, seaborn и jouru
 - Проверка версий подключенных библиотек
- Шаг 1. Открытие и первичный анализ файлов с данными[†]
 - Замысел выполнения шага 1
 - Словарь полей данных и условий их автоматизированной проверки
 - Информация об одном поле
 - Информация о всех полях и их типах
 - Информация об одном условии проверки набора данных
 - Вспомогательный класс для загрузки и проверки файла данных
 - Замысел вспомогательного класса
 - Код вспомогательного класса
 - Специфические для полученного набора данных условия проверки
 - Параметры загрузки и открытия файлов
 - Загрузка файлов данных
 - Шаблон операций с таблицами
 - Вывод краткой информации о наборах данных
 - Изучение распределений числовых значений
 - Просмотр примеров записей из таблиц
 - Создание таблицы с информацией об игровых платформах
 - Преобразование платформы к категориальному типу
 - Пропуски и признаки аномалий
 - Вывод по шагу 1
- Шаг 2. Подготовка данных
 - Поиск пропусков, ошибок и аномалий в данных
 - Первая автоматизированная проверка типов полей
 - Аномалия "одного в темноте" и новый кошмар от сеньора
 - Конец аномалии "одного в темноте"
 - Обработка TBD в оценках пользователей
 - Контрольная проверка типов полей
 - "Идентификация Борна"
 - Первая автоматизированная проверка значений
 - Предобработка данных
 - Обработка пропусков в оценках игр
 - Удаление игр без названия и жанра
 - Преобразование названий игр в категориальную переменную
 - Корректировка аномалий рейтинга
 - Вычисление общего объема продаж игры
 - Контрольная автоматизированная проверка значений
 - Размер таблицы после предобработки
 - Вывод по шагу 2
- Шаг 3. Анализ данных
 - Инструменты анализа и визуализации
 - Палитра цветов
 - Перевод названий столбцов на русский язык
 - Процедура построения тепловой карты
 - Выпуск игр в разные годы
 - Тепловая карта количества новых игр по платформам и годам
 - Сроки жизни игровых платформ
 - Выпуск новых игр по годам
 - Тепловая карта количества новых игр по жанрам и годам
 - Тепловая карта количества выпущенных игр по платформам и жанрам

- Доли жанров в общей структуре игрового программного обеспечения
 - Доли рейтингов в общей структуре игрового программного обеспечения
 - Продажи игр
 - Взрыв на макаронной фабрике
 - Продажи игр по годам
 - Продажи игр по годам для различных поколений игровых платформ
 - Распределение оценок
 - Функция построения сравнительных графиков распределений
 - Распределение оценок пользователей
 - Распределение оценок критиков
 - Изменение оценок во времени
 - Распределение продаж за актуальный период
 - Гистограмма продаж
 - "Ящик с усами" по глобальным продажам игр в разбивке по платформам
 - Анализ зависимости продаж от оценок
 - Нужны ли оценки?
 - Отличия в продажах игр различных жанров
 - Вывод по шагу 3
- Шаг 4. Составьте портрет пользователя каждого региона
- Инструменты анализа
 - Самые популярные платформы по регионам
 - Самые популярные жанры по регионам
 - Влияние рейтинга ESRB на продажи в регионах
 - Выводы по шагу 4
- Шаг 5. Проверка гипотез
- Инструмент проверки гипотез
 - Выбор уровня значимости
 - Проверка гипотезы о средних пользовательских оценках платформ Xbox One и PC
 - Проверка гипотезы о средних пользовательских оценках жанров Action и Sports
 - Вывод по шагу 5
- Шаг 6. Общий вывод
- Таблица контроля версий
- Бонус
 - Суп от сеньора
 - Рецепт супа сеньора

Описание проекта

Вы работаете в интернет-магазине «**Стримчик**», который продаёт по всему миру **компьютерные игры**.

Из открытых источников доступны исторические данные о продажах игр, оценки пользователей и экспертов, жанры и платформы (например, **Xbox** или **PlayStation**). Вам нужно выявить определяющие успешность игры закономерности. Это позволит **сделать ставку на потенциально популярный продукт и спланировать рекламные кампании**.

Перед вами данные до 2016 года. Представим, что **сейчас декабрь 2016 г., и вы планируете кампанию на 2017-й**. Нужно отработать принцип работы с данными. Неважно, прогнозируете ли вы продажи на 2017 год по данным 2016-го или же 2027-й — по данным 2026 года.

В наборе данных попадается аббревиатура **ESRB** (Entertainment Software Rating Board) — это ассоциация, определяющая возрастной рейтинг компьютерных игр. ESRB оценивает игровой контент и присваивает ему подходящую возрастную категорию, например, «Для взрослых», «Для детей младшего возраста» или «Для подростков».

Инструкция по выполнению проекта

Шаг 1. Откройте файл с данными и изучите общую информацию

Путь к файлу: /datasets/games.csv. ([к выполнению](#))

Шаг 2. Подготовьте данные

Замените названия столбцов ([приведите к нижнему регистру](#));

Преобразуйте данные в нужные типы. [Опишите](#), в каких столбцах заменили тип данных и [почему](#);

Обработайте пропуски при необходимости:

- Объясните, почему [заполнили](#) пропуски определённым образом или почему [не стали](#) это делать;
- Опишите [причины](#), которые могли привести к пропускам;
- [Обратите](#) внимание на аббревиатуру 'tbd' в столбцах с рейтингом.
- Отдельно разберите это значение и [опишите](#), как его обработать.

[Посчитайте](#) суммарные продажи во всех регионах и запишите их в отдельный столбец.

К шагу 2

Шаг 3. Проведите исследовательский анализ данных

[Посмотрите](#), сколько игр выпускалось в разные годы. [Важны ли](#) данные за все периоды?

Посмотрите, как [менялись](#) продажи по платформам. Выберите платформы с наибольшими суммарными продажами и постройте распределение по годам. [За какой](#) характерный срок появляются новые и исчезают старые платформы?

Возьмите данные за соответствующий **актуальный период**. Актуальный период [определите](#) самостоятельно в результате исследования предыдущих вопросов. Основной фактор — эти данные помогут построить прогноз на 2017 год.

Не учитывайте в работе данные за **предыдущие годы**.

Какие платформы лидируют по продажам, растут или падают? [Выберите](#) несколько потенциально прибыльных платформ.

[Постройте](#) график «ящик с усами» по глобальным продажам игр в разбивке по платформам. Опишите результат.

Посмотрите, как влияют на продажи внутри одной популярной платформы отзывы пользователей и критиков. Постройте диаграмму рассеяния и [посчитайте](#) корреляцию между отзывами и продажами. Сформулируйте выводы. Соотнесите выводы с продажами игр на других платформах.

[Посмотрите](#) на общее распределение игр по жанрам. Что можно сказать о самых прибыльных жанрах? [Выделяются](#) ли жанры с высокими и низкими продажами?

К шагу 3

Шаг 4. Составьте портрет пользователя каждого региона

[Определите](#) для пользователя каждого региона (NA, EU, JP):

- Самые популярные [платформы](#) (топ-5). Опишите различия в долях продаж.
- Самые популярные [жанры](#) (топ-5). Поясните разницу.
- Влияет ли [рейтинг](#) ESRB на продажи в отдельном регионе?

Шаг 5. Проверьте гипотезы

Средние пользовательские рейтинги [платформ](#) Xbox One и PC одинаковые;

Средние пользовательские рейтинги [жанров](#) Action (англ. «действие», экшен-игры) и Sports (англ. «спортивные соревнования») разные.

[Задайте](#) самостоятельно пороговое значение alpha.

Поясните:

- Как вы сформулировали нулевую и альтернативную гипотезы;
- Какой [критерий](#) применили для проверки гипотез и почему.

Шаг 6. Напишите общий вывод

([к общему выводу](#))

Требования к оформлению

Задание выполните в Jupyter Notebook. Программный код заполните в ячейках типа code, текстовые пояснения — в ячейках типа markdown. Примените форматирование и заголовки.

Описание данных

Имя поля	Описание поля
Name	название игры
Platform	платформа
Year_of_Release	год выпуска
Genre	жанр игры
NA_sales	продажи в Северной Америке (миллионы проданных копий)
EU_sales	продажи в Европе (миллионы проданных копий)
JP_sales	продажи в Японии (миллионы проданных копий)
Other_sales	продажи в других странах (миллионы проданных копий)
Critic_Score	оценка критиков (максимум 100)
User_Score	оценка пользователей (максимум 10)
Rating	рейтинг от организации ESRB (англ. Entertainment Software Rating Board - эта ассоциация определяет рейтинг компьютерных игр и присваивает им подходящую возрастную категорию)

Данные за 2016 год могут быть неполными.

Критерии проверки

На что обращают внимание при проверке проекта:

- Как вы описываете выявленные в данных проблемы?
- Как готовите датасет к анализу?
- Какие строите графики для распределений и как их объясняете?
- Как рассчитываете стандартное отклонение и дисперсию?
- Формулируете ли альтернативную и нулевую гипотезы?
- Какие методы применяете, чтобы их проверить?
- Объясняете результат проверки гипотезы или нет?
- Соблюдаете ли структуру проекта и поддерживаете аккуратность кода?

- Какие выводы делаете?
- Оставляете ли комментарии к шагам?

Выполнение работы

Подключение и настройка необходимых библиотек

```
In [1]: # отключение предупреждений
import warnings; warnings.filterwarnings("ignore", category=DeprecationWarning)

In [2]: import os
import errno

In [3]: from collections import namedtuple, defaultdict

In [4]: from urllib.request import urlretrieve

In [5]: from IPython.display import HTML, display

In [6]: from functools import partial

In [7]: from scipy import stats as st
```

Настройка pandas

```
In [8]: # обновление библиотеки Pandas для исключения ошибок версий
!pip3 install --upgrade --quiet --user pandas

In [9]: import numpy as np
import pandas as pd

In [10]: pd.set_option('display.notebook_repr_html', True)
pd.set_option('display.max_columns', 8)
pd.set_option('display.max_rows', 10)
pd.set_option('display.width', 80)
pd.set_option('display.float_format', '{:.3f}'.format)
```

Настройка matplotlib, seaborn и joypy

```
In [11]: # обновление библиотек
!pip3 install --upgrade --quiet --user matplotlib seaborn joypy

In [12]: import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.ticker as ticker
import seaborn as sns
from joypy import joyplot

In [13]: %matplotlib inline

In [14]: small, medium, large = 12, 16, 22
params = {'figure.figsize': (16, 6),
           'figure.titlesize': large,
           'legend.fontsize': medium,
           'axes.titlesize': medium,
           'axes.labelsize': medium,
           'xtick.labelsize': medium,
           'ytick.labelsize': medium,
           'legend.loc': 'best'}
plt.rcParams.update(params)

In [15]: # повышение четкости графиков для больших мониторов
%config InlineBackend.figure_format = 'retina'

In [16]: # включение цветового оформления Seaborn
# plt.style.use('seaborn-whitegrid')
sns.set_style("white")
```

Проверка версий подключенных библиотек

```
In [17]: def lib_versions(libs):
    for lib in libs: print('Версия', lib.__name__, '-', lib.__version__)

In [18]: lib_versions([np, pd, mpl, sns])

Версия numpy - 1.19.0
Версия pandas - 1.1.4
Версия matplotlib - 3.3.3
Версия seaborn - 0.11.0
```

Шаг 1. Открытие и первичный анализ файлов с данными¶

Задание описано выше

Замысел выполнения шага 1

Не будет преувеличением сказать, что прохождение учебного курса и приближение к реальным задачам сопровождается увеличением количества источников данных и их объемов. Проведение предобработки информации только вручную будет отнимать много времени и сил, что в свою

очередь может привести к снижению качества анализа или срыва сроков сдачи проекта.

Для нивелирования описанной проблемы создадим вспомогательные структуры данных, классы и методы для автоматической загрузки массива данных, его первоначальной проверки и подготовки удобочитаемого отчета об обнаруженных ошибках в значениях полей.

Полученный опыт будем использовать не только в решении конкретных задач, но и в создании (совершенствовании) средств автоматической предобработки данных, которые, например, могут быть оформлены в виде отдельных библиотек.

Если Вам не терпится посмотреть результат, можете сразу перейти к [загрузке файлов данных](#).

Словарь полей данных и условий их автоматизированной проверки

Создадим словарь со списком всех полей нашего набора данных в соответствии с их [описанием в задании](#). Эх, получать бы сразу такую информацию от поставщика данных в удобном формате типа [JSON](#) или [XML](#)!

Информация об одном поле

Используем `namedtuple` для описания структуры информации об отдельном поле набора данных. В коде программ к этим полям будем обращаться через точку:

- `desc_short` - смысловое описание поля в кратком формате для небольших графиков;
- `desc` - смысловое описание поля;
- `dtype` - тип поля;
- `try_to_fix` - если `True`, то попытаться автоматически исправить тип поля на указанный в `dtype`,
- `name_in_csv` - имя поля в файле данных (оно будет переименовано в значение ключа словаря).

```
In [19]: DataField = namedtuple('DataField', ['desc_short', 'desc', 'dtype', 'try_to_fix', 'name_in_csv'],
                           defaults=[None, None, np.object, True, None])
```

Информация о всех полях и их типах

Сведём все имена полей, их описания и требования к типам в один словарь для последующей алгоритмизации. В учебных целях код оставим в этом ноутбуке. В будущем можем вынести его в отдельный файл.

```
In [20]: data_fields = \
{
    'Игры':
    {
        'name': DataField('название игры',
                          'название игры',
                          np.object, True, 'Name'),
        'platform': DataField('платформа',
                              'платформа игры',
                              np.object, True, 'Platform'),
        'year_of_release': DataField('год выпуска',
                                    'год выпуска игры',
                                    np.int64, True,
                                    'Year_of_Release'),
        'genre': DataField('жанр игры',
                           'жанр игры',
                           np.object, True, 'Genre'),
        'na_sales': DataField('в Северной Америке',
                              'продажи в Северной Америке в миллионах копий',
                              np.float64, True, 'NA_sales'),
        'eu_sales': DataField('в Европе',
                              'продажи в Европе в миллионах копий',
                              np.float64, True, 'EU_sales'),
        'jp_sales': DataField('в Японии',
                              'продажи в Японии в миллионах копий',
                              np.float64, True, 'JP_sales'),
        'other_sales': DataField('в других странах',
                                 'продажи в других странах в миллионах копий',
                                 np.float64, True, 'Other_sales'),
        'critic_score': DataField('оценка критиков',
                                  'оценка критиков - максимум 100',
                                  np.float64, True, 'Critic_Score'),
        'user_score': DataField('оценка пользователей',
                               'оценка пользователей - максимум 10',
                               np.float64, True, 'User_Score'),
        'rating': DataField('рейтинг ESRB',
                            'рейтинг от организации ESRB',
                            np.object, True, 'Rating')
    }
}
```

Опишем вспомогательные функции, которые пригодятся при подписывании осей графиков и именовании столбцов таблиц:

```
In [21]: def get_desc_long(table_name, field_names):
    """Возвращает список смысловых описаний для переданного списка полей"""
    return list(map(lambda x: data_fields[table_name][x].desc, field_names))

def get_desc_short(table_name, field_names):
    """Возвращает список смысловых описаний в кратком формате для переданного списка полей"""
    return list(map(lambda x: data_fields[table_name][x].desc_short, field_names))
```

Информация об одном условии проверки набора данных

Аналогично опишем информацию об одном конкретном условии проверки значений набора данных:

- `fields` - список имен проверяемых полей;
- `desc_template` - шаблон смыслового описания условия проверки, которое будет выводится в дальнейшем на экран с использованием значения `fields` в качестве параметра к `str.format`;
- `query_template` - шаблон строки запроса на выборку неудовлетворяющих условию записей через `pandas.query` с использованием значения `fields` в качестве параметра к `str.format`;
- `fix` - функция корректировки найденных ошибок (получает на вход ссылку на набор данных). Если нет необходимости исправлять ошибки, то присвойте `fix` значение `None`. Функцию удобно задавать через `lambda`. `fix` можно применять не только для исправления ошибок, но и

просто для вывода дополнительной информации после проверки очередного условия.

In [22]: DataChecking = namedtuple('DataChecking', ['fields', 'desc_template', 'query_template', 'fix'])

Вспомогательный класс для загрузки и проверки файла данных

Опишем класс, облегчающий предобработку одного файла данных. В учебных целях его код оставим в этом ноутбуке. В будущем можем вынести его в отдельный файл. Заказчика кодом не удивишь, а только напугаешь!

Замысел вспомогательного класса

К разрабатываемому классу предъявим следующие требования:

- загрузка файла должна осуществляться с локального или сетевого источника (при отсутствии локального);
- параметры открытия файла через Pandas должны быть настраиваемыми;
- проверку файла по заданным критериям можно осуществлять повторно (вдруг в будущих проектах машинного обучения валидационную выборку нам сразу не предоставят или обучающую дополнят);
- отчет о проверке файла должен быть удобочитаемым.

Код вспомогательного класса

```
In [23]: class PandasFile(object):  
    """Класс для описания одного файла, который необходимо загрузить для дальнейшего анализа с помощью Pandas"""\n\n    def __init__(self, name, file_path, file_url, pandas_params={}, data_fields=None, data_checkings=None):  
        """Конструктор"""\n\n        self.__name = name # назначение файла  
        self.__file_path = file_path # локальный путь к файлу  
        self.__file_url = file_url # сетевой адрес файла  
        self.__pandas_params = pandas_params # список параметров pandas.read_csv  
        self.data_fields = data_fields # словарь информации о полях и их типах  
        self.data_checkings = data_checkings # список условий автоматизированной проверки значений полей  
        self.__df = None # ссылка на созданный из файла датафрейм Pandas\n\n        # проверить наличие файла, загрузить его при отсутствии  
        if self.exists():  
            print(f'{self.__name} ({self.__file_path}) был загружен ранее.')  
        else:  
            self.load_from_url()  
  
        # создать датафрейм из файла  
        self.get_data_frame(reread=True)  
  
    def exists(self):  
        """Проверка наличия локального файла"""\n        return os.path.isfile(self.__file_path)  
  
    def load_from_url(self):  
        """Загрузка файла из сетевого источника"""\n        # создаем папку для сохранения файла с данными, если она ещё не создана  
        folder_path = os.path.dirname(self.__file_path)  
        if not os.path.exists(folder_path):  
            print(f'Создаю папку {folder_path}')  
            os.makedirs(folder_path)  
  
        print(f'Загружаю "{self.__name}" из {self.__file_url} в {self.__file_path} ... ', end=' ')  
        result = urlretrieve(url=self.__file_url, filename=self.__file_path)  
        print('OK')  
        return result  
  
    def get_data_frame(self, reread=False):  
        """Загрузка датафрейма Pandas из локального файла"""\n        if reread or self.__df is None:  
            print(f'Открываю "{self.__name}" с помощью Pandas ... ', end=' ')  
            self.__df = pd.read_csv(self.__file_path, **self.__pandas_params)  
            print('OK')  
            # переименование полей при необходимости  
            for field_name, field_info in self.data_fields.items():  
                if field_info.name_in_csv is not None:  
                    print(f'Переименовываю поле "{field_info.name_in_csv}" в "{field_name}" ... ', end=' ')  
                    self.__df.rename(columns={field_info.name_in_csv: field_name}, inplace=True)  
                    print('OK')  
            # пересортировка столбцов в порядке их следования в списке  
            self.__df = self.__df[self.data_fields.keys()]  
        return self.__df  
  
    def check_data_frame_fields(self, check_try_to_fix=True, data_fields=None, prefix='type error'):  
        """Проверка полей датасета на соответствие типам, указанным в data_fields.  
        Возвращает количество несоответствий заданным требованиям.  
        prefix - строка-префикс для создания ссылок на ошибки с использованием тега <a>  
        """\n\n        display(HTML(f'\n        Протокол автоматизированной проверки типов полей набора данных "{self.__name}"'))\n\n        # выбираем список с информацией о полях для проверки  
        if data_fields is None:  
            data_fields = self.data_fields  
  
        # если для набора данных не заданы описания полей...  
        if data_fields is None:  
            display(HTML(f'В наборе данных <b>{self.__name}</b> нет описания полей. Проверка не проводилась.'))  
            return -1  
  
        # обнулим счетчик ошибок  
        errors = 0  
  
        # полный префикс ссылки тега <a>  
        link_prefix = f'{self.__name}-{prefix}'.replace(' ', '-')
```

```

# для каждого описания поля...
for field_name, field_info in data_fields.items():
    try:
        # проверка существования поля
        field_dtype = self.__df[field_name].dtypes

        # если тип поля отличается от предполагаемого...
        if field_dtype != field_info.dtype:
            errors += 1
            display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                        f'<b>{errors}</b> Тип <b>{field_dtype}</b> поля <b>{field_name}</b> ' +
                        f'<b>({field_info.desc})</b> не соответствует заявленному в описании ' +
                        f'<b>({field_info.dtype._name_})</b>.'))

        # если необходимо попытаться изменить тип поля...
        if check_try_to_fix and field_info.try_to_fix:
            try:
                # попытаемся изменить тип поля
                self.__df[field_name] = self.__df[field_name].astype(field_info.dtype)
            except Exception as e:
                display(HTML(f'При попытке изменения типа поля произошла ошибка: <i>{e}</i>'))

            # проверяем тип поля после попытки его изменения
            field_dtype = self.__df.dtypes[field_name]
            if field_dtype == field_info.dtype:
                display(HTML(f'Тип поля <b>{field_name}</b> ({field_info.desc}) ' +
                            f'изменен на <b>({field_info.dtype._name_})</b>.'))

            else:
                # "назад в будущее" - создадим гиперссылку на блок исправления ошибок
                display(HTML(f'<p><a href="#{link_prefix}{errors}-fix">М. исправления и комментарии по п. ' +
                            f'{errors}</a></p>'))

        # обработка исключительных ситуаций
    except KeyError as e:
        errors += 1
        display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                    f'<b>{errors}</b> Поле с именем <b>{field_name}</b> в датасете не найдено.'))

    if errors == 0:
        display(HTML(f'<a id="{link_prefix}{errors}"></a>' +
                    f'В наборе данных <b>{self.__name}</b> ошибок в типах не обнаружено.'))

# возвращаем общее количество обнаруженных замечаний
return errors

def check_data_frame_values(self, check_fix=True, example_count=3, data_fields=None, data_checkings=None,
                           prefix='value error'):
    """Проверка значений датасета на соответствие условиям, указанным в data_checkings.
    Возвращает количество несоответствий заданным требованиям."""
    display(HTML(f'<br><b>Протокол автоматизированной проверки значений набора данных <b>{self.__name}</b>:</b>'))

    # выбираем список с информацией об условиях проверки значений
    if data_checkings is None:
        data_checkings = self.data_checkings

    # если для набора данных не заданы условия проверки значений...
    if data_checkings is None:
        display(HTML(f'В наборе данных <b>{self.__name}</b> нет условий проверки значений. ' +
                    f'Проверка не проводилась.'))

    return -1

    # выбираем список с информацией о полях для проверки
    if data_fields is None:
        data_fields = self.data_fields

    # если для набора данных не заданы описания полей...
    if data_fields is None:
        display(HTML(f'В наборе данных <b>{self.__name}</b> нет описания полей. ' +
                    f'Расширенные пояснения не будут выводиться.'))

    # обнулим счетчик ошибок
    errors = 0

    # полный префикс ссылки тега <a>
    link_prefix = f'{self.__name}-{prefix}'.replace(' ', '-')

    # для каждого условия проверки...
    for check in data_checkings.copy():
        try:
            # проверим существование полей
            for f in check.fields: field_type = self.__df[f].dtypes

            # отберем записи набора, которые не удовлетворяют условиям проверки
            check_df = self.__df.query(check.query_template.format(*check.fields))

            # определим их количество
            lines_count = check_df.shape[0]

            # если обнаружены нарушения...
            if check_df.shape[0] > 0:
                errors += 1

            # выведем текст ошибки с указанием количества записей и их доли в процентах
            percent = lines_count / self.__df.shape[0]

            # сформируем текст с перечислением
            try:
                fields_list = ', '.join(map(lambda f: f"<b>'{f}'</b> " +
                                            f"({data_fields[f].desc})", check.fields))
            except:
                fields_list = ', '.join(map(lambda f: f"<b>'{f}'</b>", check.fields))

        except:
            fields_list = ', '.join(map(lambda f: f"<b>'{f}'</b>", check.fields))

```

```

        display(f'<a id="{link_prefix}{errors}"></a>')
        f'<b>{errors}</b> Обнаружены замечания к значениям {fields_list}. '
        f'{check.desc_template.format(*check.fields)}.' 
        f'Количество записей с нарушением: <b>{lines_count} ({percent:.02%})</b>.')
    )

    # если обработчик ошибки задан...
    if check_fix and check.fix is not None:
        # вызовем его, передав ссылку на условие проверки и датасет с ошибочными значениями
        check.fix(check, check_df, self._df)
    else:
        # иначе выведем пример ошибок в табличном виде
        if example_count > 0:
            display(HTML('Пример:'))
            #display(check_df[check.fields].head(example_count))
            display(check_df.head(example_count))

        # "назад в будущее" - создадим гиперссылку на блок исправления ошибок
        display(HTML(f'<p><a href="#{link_prefix}{errors}-fix">См. исправления и комментарии по п. '
                    f'{errors}</a></p>'))

    # обработка исключительных ситуаций
    except KeyError as e:
        errors += 1
        display(HTML(f'<a id="{link_prefix}{errors}"></a>')
                f'<b>{errors}</b> Поля с именем <b>{e}</b> в датасете не найдено. '))

    except NameError as e:
        errors += 1
        # определим имя поля, совпадающее с зарезервированным словом,
        # из последних скобок сообщения NameError
        text = str(e)[-1]
        reserved_name = text[text.find('(') + 1 : text.find(')')[-1]]
        display(HTML(f'<a id="{link_prefix}{errors}"></a>')
                f'<b>{errors}</b> Имя поля <b>{reserved_name}</b> совпадает с зарезервированным '
                f'словом. Переименуйте его для возможности использовать в запросах '
                f'<b>pandas.query</b>. Проверка поля <b>{reserved_name}</b> не выполнена. '))

    if errors == 0:
        display(HTML(f'<a id="{link_prefix}{errors}"></a>')
                f'В наборе данных <b>{self._name}</b> ошибок в значениях не обнаружено. '))

# возвращаем общее количество обнаруженных замечаний
return errors

@property
def df(self):
    """Возвращает ссылку на последнюю созданную копию датасета Pandas"""
    return self.get_data_frame(reread=False)

@property
def name(self):
    return self._name

```

Опишем наиболее часто употребляемые элементы условий проверки. Вместо `{0}`, `{1}` и так далее программа (см. ниже) будет подставлять имя проверяемого поля из `fields` по индексу, начиная с нуля:

```
In [24]: # проверка на пустое значение одного поля
desc_template_nan      = 'Значение {0} не должно быть пустым'
query_template_nan     = '`{0}` != `{}{0}`'

# проверка на пустое значение двух полей одновременно
desc_template_nan_2    = 'Значения {0} и {1} не должны быть пустыми одновременно'
query_template_nan_2   = '`{0}` != `{}{0}` and (`{1}` != `{}{1}`)'

# другие частные проверки
desc_template_g_zero   = 'Значение "{0}" должно быть больше нуля'
query_template_g_zero  = '`{0}` >= 0'

desc_template_ge_zero  = 'Значение "{0}" должно быть не меньше нуля'
query_template_ge_zero = '`{0}` < 0'

desc_template_int       = 'Значение "{0}" должно быть округлено'
query_template_int      = '`{0}` % 1 != 0'
```

Специфические для полученного набора данных условия проверки

Проверочные условия зададим, исходя из здравого смысла и опыта специалистов в сфере разработки игрового программного обеспечения. Например, в спецификации к массиву сказано о максимальных значениях оценок критиков (100) пользователей (10).

На странице с информацией о рейтинге ESRB узнаём, что он должен принимать только определённые символические значения:

- «**EC** («**Early childhood**») — «Для детей младшего возраста»: Игра подходит для детей от 3 лет и старше и не содержит материалов, которые родители могли бы счесть неподходящими. Продукты, получившие данный рейтинг, изначально разрабатываются для детей и обычно представляют собой развивающие игры. Некоторые усложнённые развивающие игры могут иметь рейтинг «Everyone».
- «**E** («**Everyone**») — «Для всех»: Содержание вполне подходит для возрастной категории от 6 лет и старше; такие игры могут понравиться и взрослым. Игры с этим рейтингом могут содержать минимальное насилие, в основном «мультишного» характера. Первой игрой, которая получила данный рейтинг, стала The Simpsons Cartoon Studio, выпущенная в 1996 году. Первоначально "**K-A**" ("Kids to Adults").
- «**E10+**» («**Everyone 10 and older**») — «Для всех от 10 лет и старше»: Проекты с данным рейтингом могут содержать немного больше мультипликационного или мягкого насилия, или несколько откровенные сцены или минимальное количество крови. Рейтинг был принят ESRB 2 марта 2005 года. Первой игрой, которая получила данный рейтинг, стала Donkey Kong: Jungle Beat.
- «**T**» («**Teen**») — «Подросткам»: Игра подходит для лиц от 13 лет и старше. Проекты из данной категории могут содержать насилие, непристойные сцены, грубый юмор, в меру откровенное сексуальное содержимое, кровь или нечестное использование ненормативной лексики.
- «**M**» («**Mature**») — «Для взрослых»: Материалы игры не подходят для подростков младше 17 лет. Проекты с данным рейтингом могут содержать достаточно жестокое насилие, большое количество крови с расчленением, непристойные сексуальные сцены или грубую ненормативную лексику, нежелательную для младшей аудитории.

- **«AO» («Adults Only 18+»)** — «Только для взрослых»: Содержание игры только для взрослых старше 18 лет. Продукты из данной категории могут содержать длительные сцены жестокого насилия и/или очень откровенное сексуальное содержимое, а также сцены с обнажением. Большинство таких игр предназначены для персональных компьютеров под управлением Microsoft Windows и Apple Macintosh. Рейтинг «Только для взрослых» является предметом многочисленных дискуссий, так как накладывает серьёзные ограничения на продажи игры.
 - **«RP» («Rating Pending»)** — «Рейтинг ожидается»: Продукт был отправлен в ESRB и ожидает присвоения рейтинга. Данный логотип используется только на рекламных презентациях и в демо-версиях игр до официальной даты выпуска в продажу.

Оценки критиков и пользователей тоже имеют ряд ограничений - подробности [здесь](#). Для каждого продукта вычисляются две средних численных оценки: **стобалльная** — на основе обзоров профессиональных изданий и **десетибалльная** — на основе пользовательских обзоров. Отрывок из каждого обзора предоставляется вместе с гиперссылкой на источник. Это даёт представление об общей привлекательности продукта среди рецензентов и, в меньшей степени, общественности. Порой эти две оценки различаются вплоть до полной противоположности.

Взглянем на пример оценок одной игры с общепризнанного сайта **Metacritic** - источника информации и для нашего проекта:



1. Hades

Platform: Switch
September 17, 2020

Defy the god of the dead as you hack and slash out of the Underworld in this rogue-like dungeon crawler from the creators of Bastion, Transistor, and Pyre. Hades is a god-like rogue-like dungeon crawler that combines the best aspects of Supergiant's critically acclaimed titles, including the fast-paced action of Bastion, the rich atmosphere and depth of Transistor, and the character-driven storytelling of Pyre. **BATTLE OUT OF HELL** As the immortal Prince of the Underworld, you'll wield the powers and mythic weapons of Olympus to break free from the clutches of the god of the dead himself, while growing stronger and unraveling more of the story with each unique escape attempt. **UNLEASH THE FURY OF OLYMPUS** The Olympians have your back! Meet Zeus, Athena, Poseidon, and many more, and choose from their dozens of powerful Boons that enhance your abilities. There are thousands of viable character builds to discover as you go. **BEFRIEND GODS, GHOSTS, AND MONSTERS** A fully-voiced cast of colorful, larger-than-life characters is waiting to meet you! Grow your relationships with them, and experience thousands of unique story events as you learn about what's really at stake for this big, dysfunctional family. **BUILT FOR REPLAYABILITY** New surprises await each time you delve into the ever-shifting Underworld, whose guardian bosses will remember you. Use the powerful Mirror of Night to grow permanently stronger, and give yourself a leg up the next time you run away from home. **NOTHING IS IMPOSSIBLE** Permanent upgrades mean you don't have to be a god yourself to experience the exciting combat and gripping story. Though, if you happen to be one, crank up the challenge and get ready for some white-knuckle action that will put your well-practiced skills to the test. **SIGNATURE SUPERGIANT STYLE** The rich....

Metascore:

User Score:

93

89

[Collapse](#)

Обращаем внимание на то, что **оценки критиков должны принимать целочисленные значения**. Учтем этот факт при проверке данных.

Выявление аномалий в данных на ранней стадии очень важно как для анализа и генерации новых атрибутов по известным значениям. Искажения могут, например, отрицательно сказаться на качестве машинного обучения, к которому мы когда-нибудь подберемся.

В следующих проектах попробуем реализовать запись условий проверки в форме, максимально приближенной к естественному языку. А пока воспользуемся описанной [выше](#) структурой `DataChecking`.

```
In [25]: # создадим словарь списков проверок для каждой таблицы  
data_checkings = defaultdict(list)
```

```

# наложим ограничения на год выпуска игры (по условию задания)
max_release_year = 2016
data_checkings['Игры'] += [DataChecking(['year_of_release']),
                           'Значение "{0}" должно быть не больше ' + f'{max_release_year}',
                           '{0} > ' + f'{max_release_year}', None]

# наложим ограничения на год выпуска игры (с начала эпохи компьютерных игр)
min_release_year = 1962
data_checkings['Игры'] += [DataChecking(['year_of_release']),
                           'Значение "{0}" должно быть не меньше ' + f'{min_release_year}',
                           '{0} < ' + f'{min_release_year}', None]

# наложим ограничения на оценки критиков
data_checkings['Игры'] += [DataChecking(['critic_score']),
                           'Значение "{0}" должно быть не больше 100',
                           '{0} > 100', None]

# наложим ограничения на оценки пользователей
data_checkings['Игры'] += [DataChecking(['user_score']),
                           'Значение "{0}" должно быть не больше 10',
                           '{0} > 10', None]

# наложим ограничения на рейтинг ESRB (RP считаем недопустимым)
data_checkings['Игры'] += [DataChecking(['rating']),
                           'Рейтинг ESRB должен принимать только определённые '
                           'символические значения',
                           "(`{0}` == `{'EC', 'E', 'E10+', 'T', 'M', 'AO'}`) & "
                           "(`{0}` not in ['EC', 'E', 'E10+', 'T', 'M', 'AO'])", None]

```

Отсортируем наши условия проверки по названию полей для удобства их дальнейшей обработки:

```
In [26]: # для всех списков проверок...
for checkings in data_checkings.values():
    checkings.sort(key = lambda x : ' '.join(x.fields))
```

Параметры загрузки и открытия файлов

Зададим необходимый набор параметров для открытия файла данных нашего проекта с помощью `pandas.read_csv`:

```
In [27]: pandas_params = \
{
    # в данном проекте особые параметры загрузки файлов пока не требуются
}
```

Зададим локальную папку для хранения файлов данных (по умолчанию `'datasets'`). При необходимости укажите более удобное место:

```
In [28]: local_folder = 'datasets'
```

Загрузка файлов данных

Зададим список всех файлов данных нашего проекта. В данном случае файл один, но в будущем их будет гораздо больше (тогда список можно будет легко пополнить). Выполнение следующей ячейки должно привести к загрузке всех отсутствующих локально файлов и созданию датафреймов, как было описано в [замысле выполнения шага 1](#) практической работы:

```
In [29]: data_files = \
{
    'Игры':
        PandasFile
        (
            'Информация о продажах игровых программ',           # назначение файла
            os.path.join(local_folder, 'games.csv'),              # локальный путь к файлу
            ',',          # сетевой адрес файла скрыт из-за правовых ограничений
            {},          # список параметров pandas.read_csv
            data_fields['Игры'],                                # словарь информации о полях
            data_checkings['Игры']                            # список условий проверки значений
        )
}
```

Загружаю "Информация о продажах игровых программ" из ... в datasets/games.csv ... OK
 Открываю "Информация о продажах игровых программ" с помощью Pandas ... OK
 Переименовываю поле "Name" в "name" ... OK
 Переименовываю поле "Platform" в "platform" ... OK
 Переименовываю поле "Year_of_Release" в "year_of_release" ... OK
 Переименовываю поле "Genre" в "genre" ... OK
 Переименовываю поле "NA_sales" в "na_sales" ... OK
 Переименовываю поле "EU_sales" в "eu_sales" ... OK
 Переименовываю поле "JP_sales" в "jp_sales" ... OK
 Переименовываю поле "Other_sales" в "other_sales" ... OK
 Переименовываю поле "Critic_Score" в "critic_score" ... OK
 Переименовываю поле "User_Score" в "user_score" ... OK
 Переименовываю поле "Rating" в "rating" ... OK

Шаблон операций с таблицами

Опишем вспомогательную процедуру для выполнения шаблонных операций для каждой таблицы:

```
In [30]: def do_for_each_data_file(action, header=None, show_title=False, *args, **kwargs):
    """процедура для выполнения шаблонных операций для каждой таблицы"""
    # выведем общий заголовок если он задан
    if header:
        display(HTML(''.join(['<br><b>', header, '</b>'])))

    # для всех таблиц выводим их название и выполняем операцию action
    for data_file in data_files.values():
        if show_title:
            display(HTML(''.join(['<br><b>Таблица ', data_file.name, '</b>'])))
        action(data_file, *args, **kwargs)
```

Пригодятся итераторы по всем таблицам и по всем полям:

```
def tables():
```

```
In [31]: """Итератор по всем таблицам.  
Последовательно возвращает кортежи (датафрейм, название)"""\n    for data_file in data_files.values():\n        yield data_file.df, data_file.name
```

```
In [32]: def fields(include=None, exclude=None):\n    """Итератор по всем столбцам (полям) с заданными типами.  
Последовательно возвращает кортежи (поле, описание поля)"""\n    for data_file in data_files.values():\n        for field in data_file.df.select_dtypes(include, exclude):\n            yield data_file.df[field], data_file.data_fields[field]
```

И создадим ссылки для быстрого обращения к таблицам:

```
In [33]: table_games = data_files['Игры'].df
```

Вывод краткой информации о наборах данных

Выведем краткую информацию о наборах данных с помощью `info` и убедимся в успешности загрузки:

```
In [34]: do_for_each_data_file(lambda f: f.df.info(), 'Информация о таблицах', True)
```

Информация о таблицах

Таблица "Информация о продажах игровых программ"

```
<class 'pandas.core.frame.DataFrame'\nRangeIndex: 16715 entries, 0 to 16714\nData columns (total 11 columns):\n #   Column      Non-Null Count  Dtype \n--- \n 0   name        16713 non-null   object \n 1   platform    16715 non-null   object \n 2   year_of_release  16446 non-null   float64 \n 3   genre       16713 non-null   object \n 4   na_sales    16715 non-null   float64 \n 5   eu_sales    16715 non-null   float64 \n 6   jp_sales    16715 non-null   float64 \n 7   other_sales 16715 non-null   float64 \n 8   critic_score 8137 non-null   float64 \n 9   user_score   10014 non-null   object \n 10  rating      9949 non-null   object \n dtypes: float64(6), object(5)\nmemory usage: 1.4+ MB
```

По беглому просмотру информации о таблицах можно сделать вывод, что **все файлы данных проекта успешно загружены**.

В таблице с информацией о продажах игровых программ содержится 16715 строк.

Некоторые поля имеют неверный тип, например **строковый** вместо **вещественный** в поле `user_score` (оценка пользователей) и **вещественный** вместо **целого** в поле `year_of_release` (год выпуска игры). Разберемся с этими аномалиями на [шаге подготовки данных](#).

В некоторых полях имеются **пропуски** с неодинаковым распределением по полям. Их анализ и принятие решения о способе обработки отложим до [первой автоматизированной проверки значений](#).

Запомним для будущего использования первоначальный размер нашей таблицы:

```
In [35]: shape_0 = table_games.shape\nnan_count_0 = table_games.isna().sum().sum()\nmemory_0 = table_games.memory_usage(deep=True).sum()\nprint(f'В исходной таблице с информацией об играх {shape_0[0]} строк и {shape_0[1]} столбцов.\n    Таблица занимает в памяти {memory_0} байт.\n    Попущены значения в {nan_count_0} ячейках.')
```

В исходной таблице с информацией об играх 16715 строк и 11 столбцов. Таблица занимает в памяти 5839242 байт. Попущены значения в 22318 ячейках.

Изучение распределений числовых значений

```
In [36]: do_for_each_data_file(lambda f: display(f.df.describe()), None, True)
```

Таблица "Информация о продажах игровых программ"

	year_of_release	na_sales	eu_sales	jp_sales	other_sales	critic_score
count	16446.000	16715.000	16715.000	16715.000	16715.000	8137.000
mean	2006.485	0.263	0.145	0.078	0.047	68.968
std	5.877	0.814	0.503	0.309	0.187	13.938
min	1980.000	0.000	0.000	0.000	0.000	13.000
25%	2003.000	0.000	0.000	0.000	0.000	60.000
50%	2007.000	0.080	0.020	0.000	0.010	71.000
75%	2010.000	0.240	0.110	0.040	0.030	79.000
max	2016.000	41.360	28.960	10.220	10.570	98.000

На первый взгляд серьезных замечаний к значениям нет:

- **продажи** неотрицательные, но встречаются **нулевые** (это странновато, особенно в контексте нашего исследования);
- максимальная оценка критиков (98) не выходит за верхний предел (100), а вот распределение **оценок пользователей** нужно будет проверить после изменения типа (в таблице сверху столбец `user_score` не выведен, так как пока имеет строковый тип).

Просмотр примеров записей из таблиц

Выведем по несколько примеров из каждой таблицы:

```
In [37]: do_for_each_data_file(lambda f: display(f.df.sample(10)), None, True)
```

Таблица "Информация о продажах игровых программ"

	name	platform	year_of_release	genre	...	other_sales	critic_score	user_score	rating
4040	FIFA Soccer 64	N64	1997.000	Sports	...	0.030	nan	NaN	NaN
13499	Art of Fighting Anthology	PS2	2006.000	Fighting	...	0.010	60.000	7	T
15533	Battle Princess of Arcadias	PS3	2013.000	Role-Playing	...	0.000	69.000	7.9	T
470	Killzone 2	PS3	2009.000	Shooter	...	0.470	91.000	8.1	M
4558	The Sims	XB	2003.000	Simulation	...	0.010	84.000	6.9	T
2412	Wizards of Waverly Place	DS	2009.000	Misc	...	0.080	58.000	tbd	E
15599	Winning Post 8 2016	PS4	2016.000	Simulation	...	0.000	nan	NaN	NaN
11686	Scooby-Doo! Who's Watching Who?	PSP	2006.000	Adventure	...	0.010	nan	NaN	NaN
10911	The Smurfs 2	Wii	2013.000	Platform	...	0.010	nan	tbd	E
2201	Tony Hawk: RIDE	Wii	2009.000	Sports	...	0.080	47.000	4.5	E10+

10 rows × 11 columns

Обратим внимание на то, что поля `platform`, `genre` и `rating`, скорее всего, имеют конечные списки значений, которые удобно укладываются в **категориальный тип** данных Pandas. Убедимся в этом:

```
In [38]: print('Список платформ:', ', '.join(sorted(table_games.platform.unique()))))
```

Список платформ: 2600, 3DO, 3DS, DC, DS, GB, GBA, GC, GEN, GG, N64, NES, NG, PC, PCF/X, PS, PS2, PS3, PS4, PSP, PSV, SAT, SCD, SNES, T616, W/S, Wii, WiiU, X360, XB, XOne

Какая прелесть! В списке есть даже "[Atari 2600](#)" - [Марк Цукерберг](#) ещё не родился, а эпоха этой приставки уже прошла.



При прогнозировании успешности новых игр нам не стоит брать во внимание такие раритеты. Или следует ориентироваться на них, но с поправкой на веяния времени в виде обоснованного математически коэффициента, так как имеющиеся технические возможности, несомненно, оказывают влияние на вариативность и привлекательность того или иного [жанра](#). Например, в эпоху первых консолей некоторые жанры ещё не существовали. Это актуально, даже если [представить](#), что [сейчас декабрь 2016 г., и мы планируем кампанию на 2017-й](#).

Создание таблицы с информацией об игровых платформах

Некоторые платформы современные, а некоторые представляют уже лишь музейную ценность. Необходимо отделить одних от других. Для этого обратимся к другим источникам данных. Но где бы нам взять информацию об актуальных игровых платформах? Какие есть варианты? Видимо, правильные и неправильные:

- отправить джуна в компьютерный игровой клуб за информацией - на неделю останемся без разносчика кофе;
- отправить джуна в библиотеку - он вместо решения этой серьезной задачи начитается книг по машинному обучению и уйдет от нас;
- пусть джун гуглит яндикс информацию - обрушим поисковый сервер, искривим [Яндекс.Метрики](#), поднимем цены на "Брелок Atari Console & Joystick 3D" на [Яндекс.Маркете](#);
- спросим у мидла из соседнего отдела - пусть все мидлы других отделов над нами за обедом посмеются;
- пожалуемся своему сеньору - вместе поплачем.

Выхода нет! Попытаемся уговорить [автора этой работы](#) бросить своё теплое местечко ради спасения индустрии компьютерных игр. На минуту представим, что нам это удалось, и он пообещал предоставить необходимые данные всего через четверть часа. Хе, хе, хе, оптимист! Как он умудрится это сделать?

[Автор](#) выполнил своё обещание и, как сказал джун, "распарсил ВЕСЬ Интернет" (именно с такими глазами), чтобы получить следующую таблицу:

Название в массиве	Полное название	Производитель	Поколение	Начало выпуска	Дополнительно
2600	Atari 2600	Atari	2	1977	wiki

Название в массиве	Полное название	Производитель	Поколение	Начало выпуска	Дополнительно
3DO	3DO Interactive Multiplayer	Panasonic	5	1993	wiki
3DS	Nintendo 3DS	Nintendo	8	2011	wiki
DC	Dreamcast	Dreamcast	6	1998	wiki
DS	Nintendo DS	Nintendo	7	2004	wiki
GB	Game Boy	Nintendo	4	1989	wiki
GBA	Game Boy Advance	Nintendo	6	2001	wiki
GC	Nintendo GameCube	Nintendo	6	2001	wiki
GEN	Sega Mega Drive	Sega	4	1988	wiki
GG	Sega Game Gear	Sega	4	1990	wiki
N64	Nintendo 64	Nintendo	5	1996	wiki
NES	Nintendo Entertainment System	Nintendo	3	1983	wiki
NG	Neo-Geo	SNK	4	1990	wiki
PC	Персональный компьютер	Мультибренд	8	1985	wiki
PC	PC Engine	NEC	4	1987	wiki
PCFX	PC-FX	NEC	5	1994	wiki
PS	PlayStation	Sony	5	1994	wiki
PS2	PlayStation 2	Sony	6	2000	wiki
PS3	PlayStation 3	Sony	7	2006	wiki
PS4	PlayStation 4	Sony	8	2013	wiki
PSP	PlayStation Portable	Sony	7	2004	wiki
PSV	PlayStation Vita	Sony	8	2011	wiki
SAT	Sega Saturn	Sega	5	1994	wiki
SCD	Sega Mega-CD	Sega	4	1992	wiki
SNES	Super Nintendo Entertainment System	Nintendo	4	1990	wiki
TG16	TurboGrafx-16	NEC	4	1987	wiki
WS	WonderSwan	Bandai	5	1999	wiki
Wii	Wii	Nintendo	7	2006	wiki
WiiU	Wii U	Nintendo	8	2012	wiki
X360	Xbox 360	Microsoft	7	2005	wiki
XB	Xbox	Microsoft	6	2001	wiki
XOne	Xbox One	Microsoft	8	2013	wiki

Есть только одна неопределенность с платформой **PC**: это может быть **PC Engine** (поддержка прекращена в 1995 году) или **игровой персональный компьютер** (к 1993 году стал совместимым стандартом для игр). Разберемся с этим вопросом чуть позже.

Подготовим отдельную таблицу `table_platforms` для информации о платформах:

```
In [39]: table_platforms = pd.DataFrame(np.array([
    ['2600', 'Atari 2600', 'Atari', 2, 1977],
    ['3DO', '3DO Interactive Multiplayer', 'Panasonic', 5, 1993],
    ['3DS', 'Nintendo 3DS', 'Nintendo', 8, 2011],
    ['DC', 'Dreamcast', 'Dreamcast', 6, 1998],
    ['DS', 'Nintendo DS', 'Nintendo', 7, 2004],
    ['GB', 'Game Boy', 'Nintendo', 4, 1989],
    ['GBA', 'Game Boy Advance', 'Nintendo', 6, 2001],
    ['GC', 'Nintendo GameCube', 'Nintendo', 6, 2001],
    ['GEN', 'Sega Mega Drive', 'Sega', 4, 1988],
    ['GG', 'Sega Game Gear', 'Sega', 4, 1990],
    ['N64', 'Nintendo 64', 'Nintendo', 5, 1996],
    ['NES', 'Nintendo Entertainment System', 'Nintendo', 3, 1983],
    ['NG', 'Neo-Geo', 'SNK', 4, 1990],
    ['PC', 'Персональный компьютер', 'Мультибренд', 8, 1985],
    #['PC', 'PC Engine', 'NEC', 4, 1987], # пока держим в уме этот призрак прошлого
    ['PCFX', 'PC-FX', 'NEC', 5, 1994],
    ['PS', 'PlayStation', 'Sony', 5, 1994],
    ['PS2', 'PlayStation 2', 'Sony', 6, 2000],
    ['PS3', 'PlayStation 3', 'Sony', 7, 2006],
    ['PS4', 'PlayStation 4', 'Sony', 8, 2013],
    ['PSP', 'PlayStation Portable', 'Sony', 7, 2004],
    ['PSV', 'PlayStation Vita', 'Sony', 8, 2011],
    ['SAT', 'Sega Saturn', 'Sega', 5, 1994],
    ['SCD', 'Sega Mega-CD', 'Sega', 4, 1992],
    ['SNES', 'Super Nintendo Entertainment System', 'Nintendo', 4, 1990],
    ['TG16', 'TurboGrafx-16', 'NEC', 4, 1987],
    ['WS', 'WonderSwan', 'Bandai', 5, 1999],
    ['Wii', 'Wii', 'Nintendo', 7, 2006],
    ['WiiU', 'Wii U', 'Nintendo', 8, 2012],
    ['X360', 'Xbox 360', 'Microsoft', 7, 2005],
    ['XB', 'Xbox', 'Microsoft', 6, 2001],
    ['XOne', 'Xbox One', 'Microsoft', 8, 2013]]),
    columns=['platform', 'description', 'producer', 'generation', 'birth'])
# преобразуем типы чисел из строк в int
table_platforms.generation = table_platforms.generation.astype(int)
table_platforms.birth = table_platforms.birth.astype(int)
```

Определим годы активной жизни платформ по минимальным и максимальным годам выпуска игр для них:

```
In [40]: platform_life = table_games.groupby('platform') \
    .agg(year_min=('year_of_release', 'min'), year_max=('year_of_release', 'max')).astype(int)
```

Сравним с годом выпуска платформы. Может обнаружим какие-нибудь странности:

```
In [41]: platform_life = table_platforms[['platform', 'birth']].join(platform_life, on='platform')
platform_life[platform_life.birth > platform_life.year_min]
```

```
Out[41]:   platform  birth  year_min  year_max
4        DS  2004    1985    2013
5        GB  1989    1988    2001
6       GBA  2001    2000    2007
29       XB  2001    2000    2008
```

Очень странно! В нашем массиве есть игры, появившиеся раньше их платформы. Разберемся в причинах:

```
In [42]: temp = table_games.join(table_platforms.set_index('platform')[['birth']], on='platform')
temp[temp.year_of_release < temp.birth]
```

```
Out[42]:      name  platform  year_of_release  genre  ...  critic_score  user_score  rating  birth
1340  Disney's DuckTales      GB    1988.000  Platform  ...        nan     NaN  NaN  1989
2076  NFL Fever 2002       XB    2000.000  Sports  ...     79.000    8.5    E  2001
12300 ESPN Winter X-Games: Snowboarding 2002      GBA    2000.000  Sports  ...        nan     NaN  NaN  2001
15957 Strongest Tokyo University Shogi DS      DS    1985.000 Action  ...        nan     NaN  NaN  2004
```

4 rows × 12 columns

Обнаружено всего четыре "выкидыши" с ошибками в свидетельстве о рождении:

- [Disney's DuckTales](#)) на самом деле родились в 1989 году;
- [NFL Fever 2002](#) - в 2001 году;
- [ESPN Winter X-Games: Snowboarding 2002](#) - в 2001 году;
- [Strongest Tokyo University Shogi DS](#) - в 2005 году.

Исправим досадные недоразумения:

```
In [43]: table_games.loc[1340, 'year_of_release'] = 1989
table_games.loc[2076, 'year_of_release'] = 2001
table_games.loc[12300, 'year_of_release'] = 2011
table_games.loc[15957, 'year_of_release'] = 2005
```

Теперь можем присоединить активные годы жизни игр к их платформам:

```
In [44]: platform_life = table_games.groupby('platform') \
    .agg(year_min=('year_of_release', 'min'), year_max=('year_of_release', 'max')).astype(int)
```

```
In [45]: table_platforms = table_platforms.join(platform_life, on='platform')
```

```
In [46]: table_platforms
```

```
Out[46]:   platform  description  producer  generation  birth  year_min  year_max
0        2600  Atari 2600  Atari          2  1977    1980    1989
1         3DO  3DO Interactive Multiplayer  Panasonic        5  1993    1994    1995
2         3DS  Nintendo 3DS  Nintendo        8  2011    2011    2016
3          DC  Dreamcast  Dreamcast        6  1998    1998    2008
4          DS  Nintendo DS  Nintendo        7  2004    2004    2013
...        ...
26         Wii        Wii  Nintendo        7  2006    2006    2016
27        WiiU        Wii U  Nintendo        8  2012    2012    2016
28        X360  Xbox 360  Microsoft        7  2005    2005    2016
29          XB        Xbox  Microsoft        6  2001    2001    2008
30        XOne  Xbox One  Microsoft        8  2013    2013    2016
```

31 rows × 7 columns

Есть ощущение, что в дальнейшем нам пригодится инструмент определения новейших платформ как по поколению, так и по году выпуска. Вот, например, только последнее поколение игровых платформ (не забывайте, 2016 год на дворе):

```
In [47]: table_platforms[table_platforms.generation == table_platforms.generation.max()]
```

```
Out[47]:   platform  description  producer  generation  birth  year_min  year_max
2        3DS  Nintendo 3DS  Nintendo        8  2011    2011    2016
13        PC  Персональный компьютер  Мультибренд        8  1985    1985    2016
18        PS4  PlayStation 4  Sony          8  2013    2013    2016
20        PSV  PlayStation Vita  Sony          8  2011    2011    2016
27        WiiU        Wii U  Nintendo        8  2012    2012    2016
30        XOne  Xbox One  Microsoft        8  2013    2013    2016
```

А вот платформы, для которых в отчетном году еще выпускаются новые игры:

```
In [48]: table_platforms[table_platforms.year_max == table_platforms.year_max.max()]
```

	platform	description	producer	generation	birth	year_min	year_max
2	3DS	Nintendo 3DS	Nintendo	8	2011	2011	2016
13	PC	Персональный компьютер	Мультибренд	8	1985	1985	2016
17	PS3	PlayStation 3	Sony	7	2006	2006	2016
18	PS4	PlayStation 4	Sony	8	2013	2013	2016
20	PSV	PlayStation Vita	Sony	8	2011	2011	2016
26	Wii	Wii	Nintendo	7	2006	2006	2016
27	WiiU	Wii U	Nintendo	8	2012	2012	2016
28	X360	Xbox 360	Microsoft	7	2005	2005	2016
30	XOne	Xbox One	Microsoft	8	2013	2013	2016

Преобразование платформы к категориальному типу

Преобразуем платформы к категориальному типу, отсортировав их по поколению и году появления:

```
In [49]: platform_dtype = \
    pd.CategoricalDtype(table_platforms.sort_values(['generation', 'birth'])['platform'].to_list(),
    ordered=True)
```

```
In [50]: table_platforms['platform'] = table_platforms['platform'].astype(platform_dtype)
table_games['platform'] = table_games['platform'].astype(platform_dtype)
```

```
In [51]: display(table_games.platform.dtype)
```

```
CategoricalDtype(categories=['2600', 'NES', 'TG16', 'GEN', 'GB', 'GG', 'NG', 'SNES',
    'SCD', '3DO', 'PCFX', 'PS', 'SAT', 'N64', 'WS', 'DC', 'PS2',
    'GBA', 'GC', 'XB', 'DS', 'PSP', 'X360', 'PS3', 'Wii', 'PC',
    '3DS', 'PSV', 'WiiU', 'PS4', 'XOne'],
    ordered=True)
```

```
In [52]: # сообщим системе автоматизированного контроля об изменении типа
data_fields['Игры']['platform'] = data_fields['Игры']['platform'].replace(dtype = platform_dtype)
```

Теперь подумаем о будущем использовании информации о платформе. Нас, аналитиков, будет больше интересовать не название платформы, а её поколение. Зная поколение платформы игры, мы сможем попробовать выявить какие-нибудь исторические закономерности в целом. Ведь игра часто выходит для нескольких платформ и не один год. Взглянем на чемпионов по количеству платформ:

```
In [53]: table_games[table_games.name == table_games.groupby('name')['platform'].count().idxmax()]
```

	name	platform	year_of_release	genre	...	other_sales	critic_score	user_score	rating
253	Need for Speed: Most Wanted	PS2	2005.000	Racing	...	0.470	82.000	9.1	T
523	Need for Speed: Most Wanted	PS3	2012.000	Racing	...	0.580	nan	NaN	NaN
1190	Need for Speed: Most Wanted	X360	2012.000	Racing	...	0.150	83.000	8.5	T
1591	Need for Speed: Most Wanted	X360	2005.000	Racing	...	0.100	83.000	8.5	T
1998	Need for Speed: Most Wanted	XB	2005.000	Racing	...	0.050	83.000	8.8	T
...
5972	Need for Speed: Most Wanted	PC	2005.000	Racing	...	0.040	82.000	8.5	T
6273	Need for Speed: Most Wanted	WiiU	2013.000	Racing	...	0.020	nan	NaN	NaN
6410	Need for Speed: Most Wanted	DS	2005.000	Racing	...	0.020	45.000	6.1	E
6473	Need for Speed: Most Wanted	GBA	2005.000	Racing	...	0.000	nan	8.3	E
11715	Need for Speed: Most Wanted	PC	2012.000	Racing	...	0.020	82.000	8.5	T

12 rows × 11 columns

Создадим в таблице игр новое поле с информацией о поколении платформы, на которой она устанавливалась:

```
In [54]: table_games['generation'] = table_games.platform.replace(table_platforms.set_index('platform').generation.to_dict())
```

Зададим описание нового поля для вывода на графиках и в отчетах в будущем:

```
In [55]: data_fields['Игры']['generation'] = \
    DataField('поколение', 'поколение игровой платформы', np.int64)
```

Пропуски и признаки аномалий

В массиве есть игры без указания жанра (nan). На русский бы ещё их перевести. Я слышал, что у заказчика исследования в совете директоров сплошь патриоты. Если видят "англицкие" слова в отчетах, дальше даже не читают:

```
In [56]: table_games.genre.unique()
```

```
Out[56]: array(['Sports', 'Platform', 'Racing', 'Role-Playing', 'Puzzle', 'Misc',
    'Shooter', 'Simulation', 'Action', 'Fighting', 'Adventure',
    'Strategy', 'nan'], dtype=object)
```

Обращает на себя внимание и дисбаланс классов по рейтингу. Некоторые типы игр представлены крайне мало:

```
In [57]: table_games.rating.value_counts()
```

```
Out[57]: E      3990
T      2961
M      1563
E10+    1420
EC      8
RP      3
K-A      3
AO      1
Name: rating, dtype: int64
```

Не просто будет оценивать перспективы игры для взрослых ("AO") на примере выборки из одного значения!

Ну, а к категориальному типу лучше конечно же перейти. Это сэкономит память и увеличит скорость выполнения группировок и индексации.

Вывод по шагу 1

- Полученный файл имеет корректный формат, удобный для обработки с помощью библиотеки Pandas.
- Файл успешно загружен в память.
- Названия столбцов не искажены, соответствуют полученному в задании описанию и могут быть использованы для упрощенной адресации Pandas в виде `data.first_name`, а не только `data['first_name']`.
- Названия столбцов автоматически приведены к нижнему регистру, как этого требует второй пункт задания.
- По дополнительным источникам создана таблица с информацией об игровых платформах, годах их выпуска и поколениях. Каждой игре сопоставлено поколение игровой платформы, это облегчит получение срезов только по актуальным платформам на этапе анализа.
- Исправлены ошибки в годах выпуска нескольких игр.
- Массив данных в целом пригоден для анализа, но требует предобработки, включая как минимум:
 - устранение пропусков;
 - изменение типов некоторых столбцов, в том числе на категориальные;
 - контроль правильности значений в соответствии с их смыслом. Так, например, оценки игр должны лежать в определенных диапазонах.
- Для облегчения выполнения предобработки был подготовлен класс, который в соответствии со списком ограничений на данные построит отчет о найденных ошибках и аномалиях. Их устранение целесообразно осуществить на следующем шаге.
- После корректировки значений набора данных следует проверить наличие дубликатов и принять решение об их удалении или способе дальнейшего использования.

Шаг 2. Подготовка данных

[Задание описано выше](#)

Поиск пропусков, ошибок и аномалий в данных

Первая автоматизированная проверка типов полей

Выполнив [ручную проверку](#), мы выявили некоторое количество замечаний к массиву данных. Настало время проверить эффективность придуманной выше автоматизированной проверки (предыдущие проекты не в счет, проверим ещё раз). Начнем с изучения типов полей загруженных файлов с помощью разработанного выше класса и попытаемся изменить неправильные типы данных автоматически:

```
In [58]: do_for_each_data_file(lambda f: f.check_data_frame_fields(check_try_to_fix=True))
```

Протокол автоматизированной проверки типов полей набора данных "Информация о продажах игровых программ":

1. Тип `float64` поля "`year_of_release`" (год выпуска игры) не соответствует заявленному в описании `int64`.

При попытке изменения типа поля произошла ошибка: `Cannot convert non-finite values (NA or inf) to integer`.

2. Тип `object` поля "`user_score`" (оценка пользователей - максимум 10) не соответствует заявленному в описании `float64`.

При попытке изменения типа поля произошла ошибка: `could not convert string to float: 'tbd'`.

Получили список замечаний по набору данных, поиск которых вручную мог бы затянуться.

Вот и первые подтверждения, что оправдались временные затраты на реализацию [замысла выполнения первого шага](#). На следующих этапах предобработки данных после исправления очередной порции ошибок и пропусков будем периодически осуществлять автоматическую проверку и радостно наблюдать за сокращением протокола.

Аномалия "одного в темноте" и новый кошмар от сеньюра

Год выпуска не удалось преобразовать к целому типу, так как для 269 игр не указано соответствующее значение:

```
In [59]: table_games[table_games.year_of_release.isna()].sample(10)
```

```
Out[59]:   name  platform  year_of_release  genre  ...  critic_score  user_score  rating  generation
12734  Mobile Ops: The One Year War     X360      nan  Simulation  ...        nan      NaN      NaN       7
  183          Madden NFL 2004      PS2      nan    Sports  ...     94.000      8.5      E       6
10993  The Daring Game for Girls      Wii      nan Adventure  ...        nan      tbd      E       7
11455          The Hidden      3DS      nan Adventure  ...        nan      4.2  E10+       8
  2169            Yakuza 4      PS3      nan   Action  ...     78.000      8      M       7
  6293  Disgaea 3: Absence of Detention    PSV      nan Role-Playing  ...     78.000      7.6      T       8
  5932          Shrek the Third      DS      nan   Action  ...     70.000      6.5      E       7
   377          FIFA Soccer 2004      PS2      nan    Sports  ...     84.000      6.4      E       6
  8364  Sword of the Samurai      PS2      nan Fighting  ...        nan      NaN      NaN       6
10258          GIFTPiA       GC      nan Role-Playing  ...        nan      NaN      NaN       6
```

10 rows × 12 columns

```
In [60]: print('Список платформ, игры которых имеют пропуски в годах выпуска:')
print(', '.join(sorted(table_games[table_games.year_of_release.isna()].platform.unique())))
```

Список платформ, игры которых имеют пропуски в годах выпуска:
2600, 3DS, DS, GB, GCA, N64, PC, PS, PS2, PS3, PSP, PSV, Wii, X360, XB

Попробуем восстановить пропуски в значениях года выпуска игр по уже известной информации. Для начала опишем вспомогательную функцию:

```
In [61]: def most_often_index(x):
    """Безопасно возвращает индекс наиболее часто встречающегося элемента в Series x"""
    try:
        return x.value_counts().idxmax()
    except:
        return np.nan
```

Заполним пропуски для игр с одинаковым названием и поколением игровой платформы наиболее часто встречающимся известным значением года выпуска. Логично предположить, что игра выходила одновременно для всех платформ-ровесников:

```
In [62]: table_games['year_of_release'] = table_games['year_of_release'].fillna(
    table_games.groupby(by=['name', 'platform'], dropna=False)[['year_of_release']].transform(most_often_index))
```

Количество пропусков уменьшилось совершенно незначительно (если восстановление одного пропуска можно так назвать):

```
In [63]: table_games.year_of_release.isna().sum()
```

```
Out[63]: 268
```

Ужас! Просто выбросить эти образцы очень жалко. Мы пока не определили актуальный период. А ещё сеньор, услышав предложение избавиться от **Alone in the Dark: The New Nightmare**, почему-то весь красными пятнами пошёл и слюной брызгал больше обычного. Прямо новый кошмар! Со стороны могло показаться, что чердак у него совсем поехал. Засыпал джуна кулинарными терминами - тот убежал в магазин **за супом**, непременно **"красивым"**. Как выяснилось потом, представления о красоте супа у сеньора, мидла и джуна оказались разными. **Версия джуна**:



Наивный **мидл** пытался суп через интернет-доставку [заказать](#):

Top Results for "Aquaman Battle for Atlantis"

- All Items
- Movies
- Games
- sort by
 - Relevancy
 - Score
 - Most Recent
 - platforms
 - All
 - PlayStation 4



Aquaman: Battle for Atlantis

GC Game, 2003

27

Here's your chance to wield the power of Aquaman, the DC Comics superhero, as he defends the deep-sea city of Atlantis against such enemies as Ocean Master, Black Manta, and the Lava Lord of the...



Aquaman: Battle for Atlantis

XBOX Game, 2003

26

Here's your chance to wield the power of Aquaman, the DC Comics superhero, as he defends the deep-sea city of Atlantis against such enemies as Ocean Master, Black Manta, and the Lava Lord of the...

А вот **версия сеньора**, которую, по его словам, "даже **этот** знает", с комментариями самого сеньора (не судите строго) описана в [приложении](#). Воспользуемся пока полученным результатом:

```
In [64]: restored_years = { 183:2003, 377:2003, 456:2008, 475:2005, 609:2009, 627:2007, 657:2001, 678:2008, 719:2006, 805:2007, 1131:2010, 1142:2007, 1301:1998
restored_years = pd.DataFrame.from_dict(restored_years, orient='index', columns=['year'])
```

```
In [65]: restored_years
```

```
Out[65]:   year
183  2003
377  2003
456  2008
475  2005
609  2009
...
16288 2009
16373 2009
16405 2003
```

year

16448 2012

16522 2005

234 rows × 1 columns

В нашем распоряжении словарь с ключами из индексов в таблице с информацией об играх, а в значениях - год выпуска игры, найденный в Интернет.

Заполним пропуски:

In [66]: `table_games.loc[restored_years.index, 'year_of_release'] = restored_years['year']`

В супе сеньора было найдено уже немало для восстановления пропусков в годах выхода игр. Пробелов осталось совсем чуть-чуть:

In [67]: `table_games.year_of_release.isna().sum()`

Out[67]: 35

А если серьезно, то парсинг сайтов с помощью библиотеки BeautifulSoup помог заполнить большинство пропусков в году выпуска игр. Нетерпеливый маркетолог может сказать: "Зачем они нужны, половина из них старые?". А мы ответим лишь молчаливым упорством - придет время и спасенные подобным образом данные помогут нам улучшить на заветную тысячечную скор в конкурсе по машинному обучению "Яндекс Мега Старт После Практикум - 2021".

Почему же о где выпуска некоторых игр не знают специализированные сайты и поисковые системы? Знают! Просто эти игры так и не дошли до продажи. От таких образцов можно смело избавиться, оценки пользователей у них отсутствуют.



Brothers in Arms: Furious 4



Компьютерная игра

Brothers in Arms: Furious 4 — компьютерная игра, разрабатывавшаяся компанией Gearbox Software; изначально планировалась к выходу в 2012 году, позже выход переносился, в 2015 году стало известно о том, что разработка отменена.

[Википедия](#)

Серия: Brothers in Arms

Дата выпуска: Отменена

Не будем терять время, продолжим пока изучение нашего массива:



Monster Hunter 2

PS2 Game, Canceled

tbd

Monster Hunter 2 is an online third-person action role-playing game from Capcom.

Вот и аббревиатура "tbd" в оценках встретилась. Происходит она от английского **to be determined** и в данном контексте означает, что рейтинг неизвестен или пока не может быть рассчитан, возможно, по причине малого количества отзывов пользователей. Это особенно характерно для слишком старых игр, слишком новых или совсем не выходивших на рынок. Вряд ли в исследовании нам могут пригодится игры без рейтинга.

Некоторые игры выходили только на азиатском рынке ([Umineko no Naku Koro ni San](#)), их подробное описание имеется исключительно на национальном языке. Для простого европейского обывателя это не интересно, а для аналитика, готовящего на стол руководству доклад, который определит решение о стратегии продвижения инновационного игрового продукта на новый рынок, будет преступлением проигнорировать факт популярности [манги](#) в Японии.

У некоторых игр год фигурирует в названии. Попытаемся восстановить пропуски с помощью регулярного выражения:

In [68]: `table_games.year_of_release.fillna(table_games.name.str.extract(r'(\d\d\d\d\d)')[0], inplace=True)`In [69]: `unknown_games = sorted(table_games[(~table_games.name.isna()) & table_games.year_of_release.isna()].name.unique())`In [70]: `print(f'Год рождения {len(unknown_games)} игр остается неизвестным.\nСреди них такие возможные шедевры, как:')`
`print(*join(unknown_games))`

Год рождения 35 игр остается неизвестным. Среди них такие возможные шедевры, как:

AKB1/48: Idol to Guam de Koishitara..., Action Man-Operation Extreme, Agarest Senki: Re-appearance, Atsumare! Power Pro Kun no DS Koushien, B.L.U.E.: Legend of Water, Bikuriman Daijiten, Breakaway IV, Brothers in Arms: Furious 4, Charm Girls Club: My Fashion Mall, Cubix Robots for Everyone: Clash 'n' Bash, Dragon Ball Z: Budokai Tenkaichi 2 (JP sales), Fullmetal Alchemist: Brotherhood, Hakuouki: Shinsengumi Kitan, Half-Minute Hero 2, Home Run, Housekeeping, Jet X20, Luminous Arc 2 (JP sales), Maze Craze: A Game of Cops 'n Robbers, Monster Hunter Frontier Online, Move Fitness, Our House Party!, Prinny: Can I Really Be The Hero? (US sales), Samurai Spirits: Tenkaichi Kenkakuden, Star Frontiers, The Hidden, The Legend of Zelda: The Minish Cap (weekly JP sales), Tom Clancy's Rainbow Six: Critical Hour, Umineko no Naku Koro ni San: Shijinjutsu to Gensou no Yosaoukyoku, Valkyria Chronicles III: Unrecorded Chronicles, Wii de Asobu: Metroid Prime, Writing and Speaking Beautiful Japanese DS, Yu-Gi-Oh! 5D's Wheelee Breakers (JP sales)

Конец аномалии "одного в темноте"

К сожалению, год рождения небольшого количества игр остается неизвестным, а времени заполнить пустые значения вручную у нас нет. Но вдруг среди них современные азиатские хиты?! Опять джуна интересная работа привалила. Пока он учит японский, мы применим другую технику восстановления года выпуска (Кстати, джун добил сеньора, уверенно предложив заполнить год средним значением игр по первой букве названия. Выручил, как всегда, мидл, предложив вместо среднего медиану по региону ☺).

Вспомним, что в наборе данных присутствует также информация о названиях **игровых платформ**. Не зря же сеньор так возбудился, увидев своего **рөвесника**. Мы можем заменить пропуски датой выпуска игровой платформы. Так как нам не придется анализировать динамику спроса на игры по годам (у нас есть данные об общих продажах игр, но нет по периодам), то результат исследования не ухудшится. Список платформ, у которых в массиве есть игры с неизвестным годом выпуска, уже не вызывает ужас:

```
In [71]: print(' ', join(table_games[table_games.year_of_release.isna()].platform.unique()))  
2600, GBA, Wii, PS2, PS3, PSP, DS, PS, 3DS, XB, X360
```

Заполним оставшиеся пропуски в годах выпуска игр годами выпуска их игровой платформы:

```
In [72]: table_games.year_of_release.fillna(  
    table_games.platform.replace(table_platforms.set_index('platform')[['birth']],  
    inplace=True)
```

```
In [73]: # проследим, чтобы восстановленные пропуски не вышли за годы жизни платформы  
temp = table_games[['platform']].join(table_platforms.set_index('platform')[['birth', 'year_max']], on='platform')  
table_games.year_of_release = \  
    table_games.year_of_release.clip(  
        lower=temp.birth,  
        upper=temp.year_max)  
temp = None
```

Теперь можем обновить активные годы платформ по годам выпуска игр к ним:

```
In [74]: platform_life = table_games.groupby('platform') \  
    .agg(year_min=('year_of_release', 'min'), year_max=('year_of_release', 'max')).astype(int)
```

```
In [75]: table_platforms.drop(columns=['year_min', 'year_max'], inplace=True)  
table_platforms = table_platforms.join(platform_life, on='platform')
```

Вычислим срок жизни каждой платформы в годах:

```
In [76]: table_platforms['age'] = table_platforms['year_max'] - table_platforms['year_min'] + 1
```

Обработка TBD в оценках пользователей

В нашем исследовании **TBD** означает неизвестную оценку игры. Приведем это значение к **NaN**, более удобному для обработки в Pandas:

```
In [77]: table_games.user_score.replace('tbd', np.nan, inplace=True)
```

Контрольная проверка типов полей

После обработки аномальных значений выполним контрольную проверку типов данных, при необходимости преобразуем их по требованиям спецификации:

```
In [78]: do_for_each_data_file(lambda f: f.check_data_frame_fields(check_try_to_fix=True))
```

Протокол автоматизированной проверки типов полей набора данных "Информация о продажах игровых программ":

1. Тип **float64** поля **"year_of_release"** (год выпуска игры) не соответствует заявленному в описании **int64**.

Тип поля **"year_of_release"** (год выпуска игры) изменен на **int64**.

2. Тип **object** поля **"user_score"** (оценка пользователей - максимум 10) не соответствует заявленному в описании **float64**.

Тип поля **"user_score"** (оценка пользователей - максимум 10) изменен на **float64**.

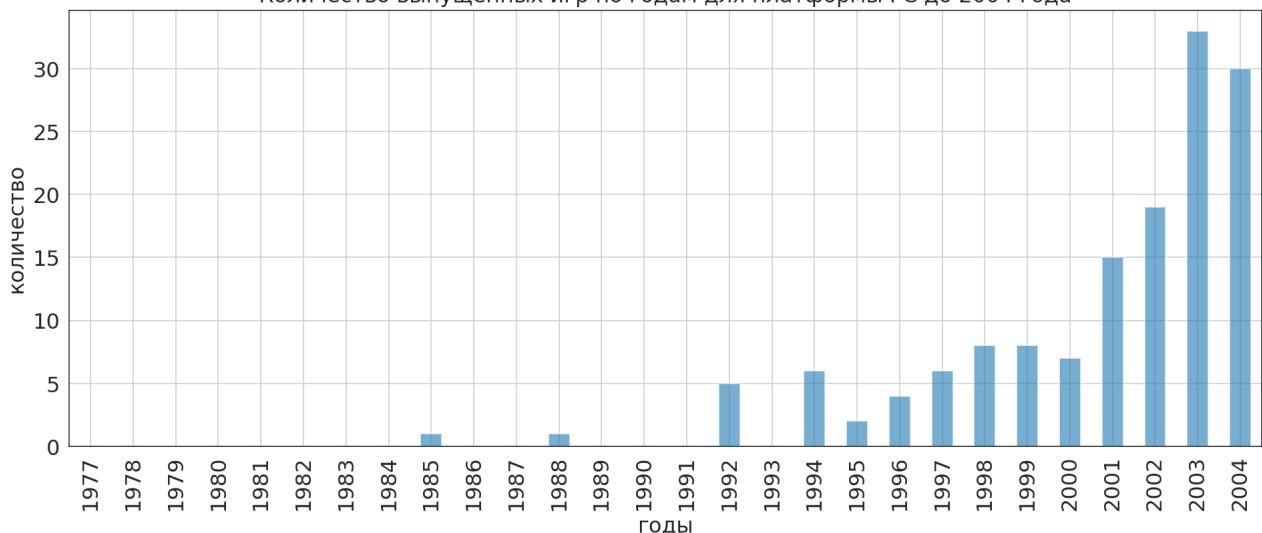
К типам данных претензий больше нет, они были преобразованы к нужному типу: год выпуска стал целым, а оценки пользователей - вещественными.

"Идентификация Борна"

Настало время снять маску с платформы **PC**: это может быть **PC Engine** (поддержка прекращена в 1995 году) или **игровой персональный компьютер** (к 1993 году стал общепризнанным совместимым стандартом для игр). Как видим, эти два класса не разделимы во времени - закат игровой приставки сопровождался рассветом игрового компьютера:

```
In [79]: year_lim = 2004  
tmp = table_games[table_games.year_of_release <= year_lim] \  
    .groupby(['platform', 'year_of_release'], as_index=False)[['name']].count()  
tmp[tmp.platform == 'PC'].plot.bar(x='year_of_release',  
    legend=False, title=f'Количество выпущенных игр по годам для платформы PC до {year_lim} года',  
    xlabel='годы', ylabel='количество', grid=True, alpha=0.6, figsize=(16, 6));
```

Количество выпущенных игр по годам для платформы PC до 2004 года



На черновой диаграмме сверху, которую мы не будем прикладывать к отчету, виден рубеж 1992 года, когда смысл сокращения **PC**, возможно, поменялся с игровой приставки на игровой компьютер. Снимем неопределенность просто - посмотрим на названия игр, выпущенных до 1995 года:

```
In [80]: print('; '.join(table_games[(table_games.platform == 'PC') & (table_games.year_of_release < 1995)].name.to_list()))
```

Doom II: Hell on Earth; Myst; Monopoly; SimCity 2000; Warcraft: Orcs & Humans; Star Wars: Dark Forces; The 7th Guest; Syndicate; Alter Ego; Doom; SimCity; Mortal Kombat; Empire Deluxe

Сомнений не осталось: **PC** - это персональный компьютер.

Первая автоматизированная проверка значений

Проверим теперь значения массивов данных на соответствие заданным [выше](#) ограничениям, сопроводив результаты пояснениями при необходимости.

Обнаруженные замечания будем автоматически помечать перекрестными гиперссылками с идентификаторами в формате `test 1 value error-{N}` (обнаруженная ошибка) и `test 1 value error-{N}-fix` (обработка ошибки), где `{N}` - номер позиции в протоколе. Это позволит нам удобно продолжить предобработку данных, переходя по этим ссылкам.

```
In [81]: do_for_each_data_file(lambda f: f.check_data_frame_values(check_fix=True, prefix='test 1 value error'))
```

Протокол автоматизированной проверки значений набора данных "Информация о продажах игровых программ":

1. Обнаружены замечания к значениям '**critic_score**' (оценка критиков - максимум 100). Значение `critic_score` не должно быть пустым. Количество записей с нарушением: **8578 (51.32%)**.

Пример:

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
1	Super Mario Bros.	NES	1985	Platform	...	nan	nan	NaN	3
4	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	...	nan	nan	NaN	4
5	Tetris	GB	1989	Puzzle	...	nan	nan	NaN	4

3 rows × 12 columns

[См. исправления и комментарии по п. 1](#)

2. Обнаружены замечания к значениям '**genre**' (жанр игры). Значение `genre` не должно быть пустым. Количество записей с нарушением: **2 (0.01%)**. Пример:

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
659	NaN	GEN	1993	NaN	...	nan	nan	NaN	4
14244	NaN	GEN	1993	NaN	...	nan	nan	NaN	4

2 rows × 12 columns

[См. исправления и комментарии по п. 2](#)

3. Обнаружены замечания к значениям '**name**' (название игры). Значение `name` не должно быть пустым. Количество записей с нарушением: **2 (0.01%)**.

Пример:

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
659	NaN	GEN	1993	NaN	...	nan	nan	NaN	4
14244	NaN	GEN	1993	NaN	...	nan	nan	NaN	4

2 rows × 12 columns

[См. исправления и комментарии по п. 3](#)

4. Обнаружены замечания к значениям '**rating**' (рейтинг от организации ESRB). Значение `rating` не должно быть пустым. Количество записей с нарушением: **6766 (40.48%)**.

Пример:

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
--	------	----------	-----------------	-------	-----	--------------	------------	--------	------------

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
1	Super Mario Bros.	NES	1985	Platform	...	nan	nan	NaN	3
4	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	...	nan	nan	NaN	4
5	Tetris	GB	1989	Puzzle	...	nan	nan	NaN	4

3 rows × 12 columns

[См. исправления и комментарии по п. 4](#)

5. Обнаружены замечания к значениям '**rating**' (рейтинг от организации ESRB). Рейтинг ESRB должен принимать только определённые символические значения. Количество записей с нарушением: **6 (0.04%)**.

Пример:

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
656	Theme Hospital	PC	1997	Strategy	...	nan	9.000	K-A	8
903	PaRappa The Rapper	PS	1996	Misc	...	92.000	7.400	K-A	5
13672	Clockwork Empires	PC	2016	Strategy	...	58.000	3.800	RP	8

3 rows × 12 columns

[См. исправления и комментарии по п. 5](#)

6. Обнаружены замечания к значениям '**user_score**' (оценка пользователей - максимум 10). Значение user_score не должно быть пустым. Количество записей с нарушением: **9125 (54.59%)**.

Пример:

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
1	Super Mario Bros.	NES	1985	Platform	...	nan	nan	NaN	3
4	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	...	nan	nan	NaN	4
5	Tetris	GB	1989	Puzzle	...	nan	nan	NaN	4

3 rows × 12 columns

[См. исправления и комментарии по п. 6](#)

Предобработка данных

Обработка пропусков в оценках игр

Автоматизированная проверка обнаружила замечания к оценкам игр:

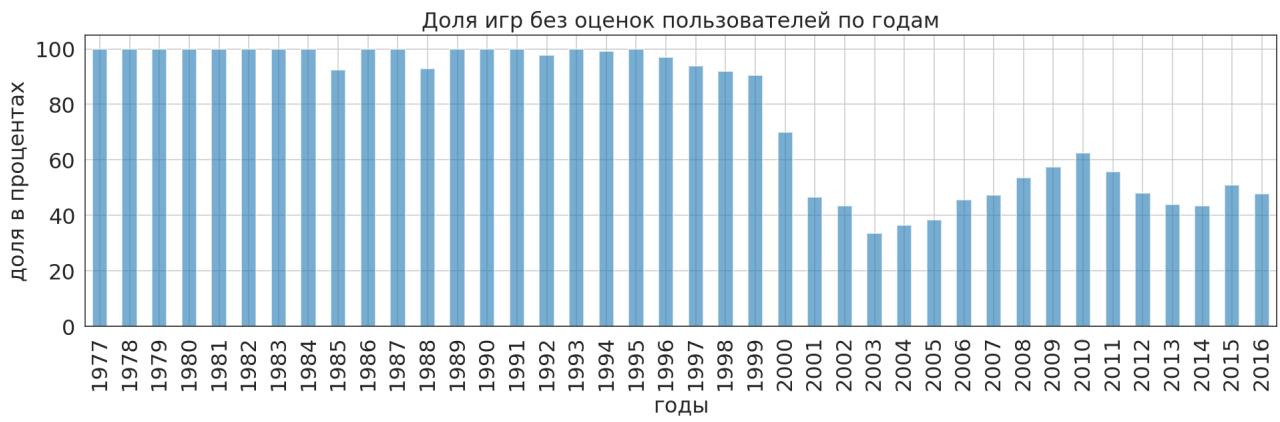
- **8578 программ (51.32% от общего количества)** не имеют оценок [критиков](#);
- для **9125 игр (54.59%)** нет информации об оценках [пользователей](#).

Для лучшего понимания причин пропусков построим две гистограммы:

```
In [82]: table_games.groupby('year_of_release') \
    .agg({'critic_score': lambda x: x.isnull().mean()*100}) \
    .plot.bar(legend=False, title='Доля игр без оценок критиков по годам',
              xlabel='годы', ylabel='доля в процентах', grid=True, alpha=0.6, figsize=(16, 4));
```



```
In [83]: table_games.groupby('year_of_release') \
    .agg({'user_score': lambda x: x.isnull().mean()*100}) \
    .plot.bar(legend=False, title='Доля игр без оценок пользователей по годам',
              xlabel='годы', ylabel='доля в процентах', grid=True, alpha=0.6, figsize=(16, 4));
```



Обратим внимание на то, что доля игр без оценок начала заметно снижаться в 2000 году. Это объясняется тем, что в 2001 году был создан [Metacritic](#) - крупнейший англоязычный сайт-агрегатор, собирающий отзывы о музыкальных альбомах, играх, фильмах, телевизионных шоу, DVD-дисках и играх. Именно с момента создания платформы оценивания игровых программ голос экспертов и пользователей стал учитываться. Наиболее популярные, культовые программы прошлых лет тоже могли получить свою оценку на этом сайте, но больший интерес всё же вызывали новинки:

```
In [84]: table_games.groupby('year_of_release') \
    .agg({'user_score': 'count'}) \
    .plot.bar(legend=False, title='Количество оцененных пользователями игр по годам',
              xlabel='годы', ylabel='количество', grid=True, alpha=0.6, figsize=(16,4));
```



Ряд игр, выходивших исключительно в азиатском или европейском регионе, мог остаться вне поля зрения Metacritic, владельцем которого является американский медиахолдинг [CBS Corporation](#).

Таким образом, заполнять пропуски в оценках игр не имеет смысла - большинство неоцененных программ канули в Лету. Анализ популярности игрового программного обеспечения целесообразно производить по имеющимся достоверным сведениям. В реальной задаче их можно было бы дополнить из других источников, например, через парсинг сайтов (что заняло бы какое-то время) или заказ у профильных организаций. В учебном проекте ограничимся имеющимся массивом.

```
In [85]: # удалим созданные ранее ограничения на пустые значения оценок игр
for check in data_checkings['Игры']:
    if ((check.fields == ['critic_score'] or check.fields == ['user_score']) and
        check.desc_template == desc_template_nan):
        data_checkings['Игры'].remove(check)
```

Удаление игр без названия и жанра

Автоматизированная проверка обнаружила **2 игры (0.01% от общего количества) без названия и жанра**. Удалим эту ничтожную долю случайных ошибок массива данных:

```
In [86]: table_games.drop(table_games[table_games.name.isna()].index, inplace=True)
```

Теперь в массиве нет пропусков в жанрах и мы можем преобразовать поле `genre` в категориальную переменную без упорядочивания. Это сократит расход памяти и ускорит индексацию, а также расширит возможности по анализу игр. Для начала опишем новый тип данных `genre_dtype`:

```
In [87]: # поместим 'Misc' (прочее) в конец списка жанров
genres = sorted(table_games.genre.unique())
genres.remove('Misc')
genres.append('Misc')
print('Жанры:', ', '.join(genres))
```

Жанры: Action, Adventure, Fighting, Platform, Puzzle, Racing, Role-Playing, Shooter, Simulation, Sports, Strategy, Misc

```
In [88]: genre_dtype = pd.CategoricalDtype(genres, ordered=False)
```

Некоторые Джуны не верят/знают, что категориальная переменная может занимать меньше памяти. Запомним, сколько байтов было выделено под неоптимизированные жанры:

```
In [89]: before = table_games.memory_usage(deep=True)[['genre']]
```

Изменим тип столбца:

```
In [90]: table_games.genre = table_games.genre.astype(genre_dtype)
```

```
In [91]: display(table_games.genre.dtype)
```

CategoricalDtype(categories=['Action', 'Adventure', 'Fighting', 'Platform', 'Puzzle',

```
'Racing', 'Role-Playing', 'Shooter', 'Simulation', 'Sports',
'Strategy', 'Misc'],
ordered=False)
```

```
In [92]: # сообщим системе автоматизированного контроля об изменении типа
data_fields['Игры']['genre'] = data_fields['Игры']['genre'].replace(dtype = genre_dtype)
```

Сравним расходы памяти до и после:

```
In [93]: after = table_games.memory_usage(deep=True)[ 'genre' ]
```

```
In [94]: print(f'Было {before}, стало {after}, сократили расходы памяти на {before - after} (байт).')
```

Было 1071971, стало 17807, сократили расходы памяти на 1054164 (байт).

Представляю себе экономию на действительно больших данных!

Преобразование названий игр в категориальную переменную

Для начала удалим незначащие пробелы в названиях:

```
In [95]: table_games.name = table_games.name.str.replace('\s+', ' ')
table_games.name = table_games.name.str.strip()
```

Опишем новый тип данных `game_names_dtype` с перечислением всех возможных названий:

```
In [96]: game_names_dtype = pd.CategoricalDtype(sorted(table_games.name.unique()), ordered=True)
```

```
In [97]: table_games.name = table_games.name.astype(game_names_dtype)
```

```
In [98]: display(table_games.name.dtype)
```

```
CategoricalDtype(categories=[''98 Koshien', '.hack//G.U. Vol.1//Rebirth',
'.hack//G.U. Vol.2//Reminisce',
'.hack//G.U. Vol.2//Reminisce (jp sales)',
'.hack//G.U. Vol.3//Redemption', '.hack//Infection Part 1',
'.hack//Link', '.hack//Mutation Part 2',
'.hack//Outbreak Part 3',
'.hack//Quarantine Part 4: The Final Chapter',
...
'nail'd', 'pro evolution soccer 2011', 'th!nk Logic Trainer',
'thinkSMART', 'thinkSMART FAMILY!',
'thinkSMART: Chess for Kids', 'uDraw Studio',
'uDraw Studio: Instant Artist', 'wwe Smackdown vs. Raw 2006',
';Shin Chan Flipa en colores!'],
ordered=True)
```

```
In [99]: # сообщим системе автоматизированного контроля об изменении типа
data_fields['Игры']['name'] = data_fields['Игры']['name'].replace(dtype = game_names_dtype)
```

Корректировка аномалий рейтинга

Автоматизированная обратила внимание на **6 игр (0.04%)** с **недопустимым рейтингом**. Среди них:

- **"K-A"** ("Kids to Adults") - устаревшее название «**E**» (**«Everyone»**) (**«Для всех»**);
- **«RP»** (**«Rating Pending»**) — «Рейтинг ожидается»: Продукт был отправлен в ESRB и ожидает присвоения рейтинга. Данный логотип используется только на рекламных презентациях и в демо-версиях игр до официальной даты выпуска в продажу.

Заменим "**K-A**" на "**E**:

```
In [100]: table_games.loc[table_games.rating == 'K-A', 'rating'] = 'E'
```

«**RP**» заменим на более логичное пустое значение:

```
In [101...]: table_games['rating'].where(table_games.rating != 'RP', inplace=True)
```

Автоматизированная проверка **обнаружила 6766 (40.48%) игр без указания рейтинга**. Это слишком много, чтобы закрыть глаза на пропуски.

Возможно что, ESRB не имеет к этим играм претензий, и мы вправе заполнить пропуски категорией "для всех". Рейтинг может отсутствовать также по причине старости игры или её реализации в другом регионе (рейтинговых организаций в мире несколько, например, CERO, PEGI, USK и RARS).

Как заполнить пропуски? Идею о средних оставим джуналам, о медианах - мидлам, о парсинге - сензорам. Студенты Яндекс.Практикума не летают в облаках, а живут в реальном мире, где действует одно суровое правило - **"не насмеши ревьюера"**, иначе **"академ"**. В связи с этим бросим взгляд на игры с одинаковым названием, изданные для разных платформ. Возьмем для примера игру, установленную на максимальном количестве игровых приставок:

```
In [102...]: table_games[table_games.name == table_games.groupby('name')[ 'platform' ].count().idxmax()]
```

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
253	Need for Speed: Most Wanted	PS2	2005	Racing	...	82.000	9.100	T	6
523	Need for Speed: Most Wanted	PS3	2012	Racing	...	nan	nan	NaN	7
1190	Need for Speed: Most Wanted	X360	2012	Racing	...	83.000	8.500	T	7
1591	Need for Speed: Most Wanted	X360	2005	Racing	...	83.000	8.500	T	7
1998	Need for Speed: Most Wanted	XB	2005	Racing	...	83.000	8.800	T	6
...
5972	Need for Speed: Most Wanted	PC	2005	Racing	...	82.000	8.500	T	8
6273	Need for Speed: Most Wanted	WiiU	2013	Racing	...	nan	nan	NaN	8
6410	Need for Speed: Most Wanted	DS	2005	Racing	...	45.000	6.100	E	7
6473	Need for Speed: Most Wanted	GBA	2005	Racing	...	nan	8.300	E	6

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
11715	Need for Speed: Most Wanted	PC	2012	Racing	...	82.000	8.500	T	8

12 rows × 12 columns

Попался гоночный рекордсмен! Да, еще так удачно - с пропусками в рейтинге. Видим, что рейтинг - понятие неустойчивое. Для одной игры он может принимать разные значения в разные годы и для разных платформ (а платформы зависят от производителя, а производитель от страны, а в каждой стране своя система оценки, как было предположено выше).

А вот пример, когда для игры вообще нет ни рейтинга, ни оценок:

```
In [103...]: table_games[table_games.name == 'Winning Post 8 2016']
```

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
15599	Winning Post 8 2016	PS4	2016	Simulation	...	nan	nan	NaN	8
16714	Winning Post 8 2016	PSV	2016	Simulation	...	nan	nan	NaN	8

2 rows × 12 columns

Всё понятно - она произведена в Японии компанией [Koei](#), поэтому у нас для неё нет рейтинга от ESRB:

Winning Post 8 2016

Видеоигра



Вам понравилась эта видеоигра?



Дата выпуска: 19 февраля 2016 г.

Издатель: Koei

Платформы: Windows, PlayStation 3

При заполнении пропусков у таких игр мы [не будем полагаться на случай](#), а максимально учтём закономерности изменения рейтинга по принципам ESRB.

Заполним пропуски для игр с одинаковым названием и годом выпуска наиболее часто встречающимся известным значением для этого же года. Логично предположить, что игра в год выхода имела одинаковый рейтинг на всех платформах:

```
In [104...]: table_games['rating'] = table_games['rating'].fillna(
    table_games.groupby(by=['name', 'year_of_release'])['rating'].transform('most_frequent'))
```

Количество пропусков уменьшилось:

```
In [105...]: table_games.rating.isna().sum()
```

Out[105...]: 6545

И во второй строчке гоночной игры у PS3 появилось заветное значение T, вычисленное по X360 в 2012 году:

```
In [106...]: table_games[table_games.name == table_games.groupby('name')['platform'].count().idxmax()]
```

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
253	Need for Speed: Most Wanted	PS2	2005	Racing	...	82.000	9.100	T	6
523	Need for Speed: Most Wanted	PS3	2012	Racing	...	nan	nan	T	7
1190	Need for Speed: Most Wanted	X360	2012	Racing	...	83.000	8.500	T	7
1591	Need for Speed: Most Wanted	X360	2005	Racing	...	83.000	8.500	T	7
1998	Need for Speed: Most Wanted	XB	2005	Racing	...	83.000	8.800	T	6
...
5972	Need for Speed: Most Wanted	PC	2005	Racing	...	82.000	8.500	T	8
6273	Need for Speed: Most Wanted	WiiU	2013	Racing	...	nan	nan	NaN	8
6410	Need for Speed: Most Wanted	DS	2005	Racing	...	45.000	6.100	E	7
6473	Need for Speed: Most Wanted	GBA	2005	Racing	...	nan	8.300	E	6
11715	Need for Speed: Most Wanted	PC	2012	Racing	...	82.000	8.500	T	8

12 rows × 12 columns

Теперь заполним пропуски для игр с одинаковым названием наиболее часто встречающимся известным значением рейтинга этой игры за все годы:

```
In [107...]: table_games.rating = table_games.rating.fillna(
    table_games.groupby(['name'])['rating'].transform('most_frequent'))
```

Количество пропусков снова уменьшилось:

```
In [108]: table_games.rating.isna().sum()
```

```
Out[108]: 6329
```

И для гоночной игры в 2013 году появился рейтинг:

```
In [109]: table_games[table_games.name == table_games.groupby('name')['platform'].count().idxmax()]
```

```
Out[109]:
```

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
253	Need for Speed: Most Wanted	PS2	2005	Racing	...	82.000	9.100	T	6
523	Need for Speed: Most Wanted	PS3	2012	Racing	...	nan	nan	T	7
1190	Need for Speed: Most Wanted	X360	2012	Racing	...	83.000	8.500	T	7
1591	Need for Speed: Most Wanted	X360	2005	Racing	...	83.000	8.500	T	7
1998	Need for Speed: Most Wanted	XB	2005	Racing	...	83.000	8.800	T	6
...
5972	Need for Speed: Most Wanted	PC	2005	Racing	...	82.000	8.500	T	8
6273	Need for Speed: Most Wanted	WiiU	2013	Racing	...	nan	nan	T	8
6410	Need for Speed: Most Wanted	DS	2005	Racing	...	45.000	6.100	E	7
6473	Need for Speed: Most Wanted	GBA	2005	Racing	...	nan	8.300	E	6
11715	Need for Speed: Most Wanted	PC	2012	Racing	...	82.000	8.500	T	8

12 rows × 12 columns

Теперь заполним пропуски рейтинга для игр с одинаковым жанром, платформой и годом выпуска наиболее часто встречающимся известным значением рейтинга. Логично предположить, что игры определенного жанра, вышедшие в один год для одной платформы, вероятнее всего, имели одинаковый рейтинг:

```
In [110]: table_games.rating = table_games.rating.fillna(  
        table_games.groupby(by=['genre', 'platform', 'year_of_release'])['rating'].transform(lambda x: x.value_counts().index[0]))
```

Теперь заполним пропуски рейтинга для игр с одинаковым жанром и годом выпуска наиболее часто встречающимся известным значением рейтинга. Логично предположить, что игры определенного жанра, вышедшие в один год, вероятнее всего, имели одинаковый рейтинг:

```
In [111]: table_games.rating = table_games.rating.fillna(  
        table_games.groupby(by=['genre', 'year_of_release'])['rating'].transform(lambda x: x.value_counts().index[0]))
```

Теперь японские скачки имеют адекватный сюжету игры рейтинг "для всех":

```
In [112]: table_games[table_games.name == 'Winning Post 8 2016']
```

```
Out[112]:
```

	name	platform	year_of_release	genre	...	critic_score	user_score	rating	generation
15599	Winning Post 8 2016	PS4	2016	Simulation	...	nan	nan	E	8
16714	Winning Post 8 2016	PSV	2016	Simulation	...	nan	nan	E	8

2 rows × 12 columns

И пропусков осталось совсем мало:

```
In [113]: table_games.rating.isna().sum()
```

```
Out[113]: 607
```

В отношении оставшихся пропусков исходим из следующего: что не запрещено - разрешено:

```
In [114]: table_games['rating'].fillna('E', inplace=True)
```

Преобразуем поле `rating` в категориальную переменную с возрастными ограничениями, отсортированными по степени строгости. Это позволит эффективно расходовать память, ускорит индексацию, а также расширит возможности по анализу игр. Для начала опишем новый тип данных:

```
In [115]: rating_dtype = pd.CategoricalDtype(['EC', 'E', 'E10+', 'T', 'M', 'AO'], ordered=True)
```

Преобразуем тип поля:

```
In [116]: table_games['rating'] = table_games['rating'].astype(rating_dtype)
```

```
In [117]: display(table_games.rating.dtype)
```

CategoricalDtype(categories=['EC', 'E', 'E10+', 'T', 'M', 'AO'], ordered=True)

```
In [118]: # сообщим системе автоматизированного контроля об изменении типа  
data_fields['Игры']['rating'] = data_fields['Игры']['rating'].replace(dtype = rating_dtype)
```

Создадим новое поле с более понятным описанием возрастного рейтинга для отображения в таблицах и графиках:

```
In [119]: rating_dict = {  
    'EC': 'EC (для детей младшего возраста)',  
    'E': 'E (для всех)',  
    'E10+': 'E10+ (для всех от 10 лет и старше)',  
    'T': 'T (подросткам)',  
    'M': 'M (для взрослых)',  
    'AO': 'AO (только для взрослых)'}  
In [120]: rating_rus_dtype = pd.CategoricalDtype(list(map(lambda x: rating_dict[x], rating_dtype.categories)), ordered=True)
```

```
In [121]: table_games['rating_rus'] = table_games['rating'].replace(rating_dict).astype(rating_rus_dtype)
```

Зададим описание нового поля для вывода на графиках и в отчетах в будущем:

```
In [122...]: data_fields['Игры']['rating_rus'] = \
    DataField('возрастной рейтинг', 'возрастной рейтинг игры от ESRB', rating_rus_dtype)
```

Убедимся в правильности преобразования значений:

```
In [123...]: display(table_games.rating_rus.dtype)

CategoricalDtype(categories=['EC (для детей младшего возраста)', 'E (для всех)',
    'E10+ (для всех от 10 лет и старше)', 'T (подросткам)',
    'M (для взрослых)', 'AO (только для взрослых)'],
    ordered=True)
```

```
In [124...]: table_games[['rating', 'rating_rus']].sample(10)
```

	rating	rating_rus
12308	E	E (для всех)
14835	E	E (для всех)
981	T	T (подросткам)
2765	E	E (для всех)
10672	E10+	E10+ (для всех от 10 лет и старше)
16594	T	T (подросткам)
8962	E	E (для всех)
10801	T	T (подросткам)
11638	E10+	E10+ (для всех от 10 лет и старше)
9385	E10+	E10+ (для всех от 10 лет и старше)

Вычисление общего объема продаж игры

По имеющимся значениям объемов продаж игры в различных регионах мира вычислим общую сумму, занесем её в отдельный столбец с именем `total_sales`:

```
In [125...]: table_games['total_sales'] = (table_games['eu_sales'] +
    table_games['na_sales'] +
    table_games['jp_sales'] +
    table_games['other_sales'])
```

Убедимся, что в новом поле нет пустых значений:

```
In [126...]: table_games.total_sales.isna().sum()
```

```
Out[126...]: 0
```

Изучим нулевые и отрицательные значения:

```
In [127...]: table_games[table_games.total_sales <= 0]
```

	name	platform	year_of_release	genre	...	rating	generation	rating_rus	total_sales
16676	G1 Jockey 4	PS3	2008	Sports	...	E	7	E (для всех)	0.000
16709	SCORE International Baja 1000: The Official Game	PS2	2008	Racing	...	E	6	E (для всех)	0.000

2 rows × 14 columns

Обнаруживаем две провальные игры. Оставляем их в качестве напоминания о рискованности инвестиций в игровую индустрию без предварительного изучения рынка и предпочтений пользователей.

Зададим описание нового поля для вывода на графиках в будущем:

```
In [128...]: data_fields['Игры']['total_sales'] = \
    DataField('в мире', 'продажи в мире в миллионах копий', np.float64)
```

Контрольная автоматизированная проверка значений

Вызовем повторно функцию автоматизированной проверки значений и убедимся, что все обнаруженные ранее замечания устраниены, а новые не появились:

```
In [129...]: do_for_each_data_file(lambda f: f.check_data_frame_values(check_fix=True, prefix='test 2 value error'))
```

Протокол автоматизированной проверки значений набора данных "Информация о продажах игровых программ":

В наборе данных "Информация о продажах игровых программ" ошибок в значениях не обнаружено.

Размер таблицы после предобработки

Вычислим, как изменился размер таблицы после предобработки:

```
In [130...]: shape_1 = table_games.shape
nan_count_1 = table_games.isna().sum().sum()
memory_1 = table_games.memory_usage(deep=True).sum()
print('В таблице с информацией об играх после предобработки \
    {} строк и {} столбцов, \
    {} удалено {} строк и создано {} новых столбца.')
print('Объем занимаемой памяти сократился с {} до {}, оптимизировано {} (байт).')
print('Имеются обоснованные пропуски в {} ячейках, заполнено {}.'.format(shape_1[0] - nan_count_1, shape_1[1], shape_1[1] - shape_1[0], memory_1 - memory_0, memory_0 - memory_1, nan_count_1, nan_count_0 - nan_count_1))
```

В таблице с информацией об играх после предобработки 16713 строк и 14 столбцов, удалено 2 строки и создано 3 новых столбца.

Объем занимаемой памяти сократился с 5839242 до 2718279, оптимизировано 3120963 (байт).
Имеются обоснованные пропуски в 17699 ячейках, заполнено 4619.

Потери информации минимальны.

Вывод по шагу 2

Задание шага 2 выполнено:

- названия столбцов ([приведены к нижнему регистру](#));
- данные [преобразованы](#) в нужные типы, выполнение операций сопровождалось необходимыми [пояснениями](#);
- [заполнены](#) пропуски в рейтинге игр;
- пропуски в оценках игр обоснованно [оставлены](#) без изменений;
- описаны [причины](#), которые могли привести к пропускам;
- [обращено](#) внимание на аббревиатуру 'tbd' в столбцах с рейтингом, аномальное значение [обосновано заменено](#) на пустое значение;
- [посчитаны](#) суммарные продажи во всех регионах, новые значения записаны в отдельный столбец.

[Удалены](#) две игры без названия и жанра.

С помощью парсинга сайтов и использования известных значений для разных платформ [заполнены](#) пропуски в годах выпуска игр.

Ряд полей [преобразован](#) к категориальному типу для [сокращения](#) расходов [памяти](#) и ускорения индексации.

Таким образом, данные предобработаны и готовы к использованию для решения основной аналитической задачи проекта.

Шаг 3. Анализ данных

[Задание описано выше](#)

Инструменты анализа и визуализации

Опишем процедуры и функции, которые облегчат проведение анализа, построение графиков и диаграмм.

Палитра цветов

```
In [131...]: fill_colors = ['#1f77b4', '#aec7e8', '#ff7f0e', '#ffbb78', '#2ca02c', '#98df8a', '#d62728', '#ff9999', '#9467bd', '#c5b0d5', '#8c564b', '#c49c94', '#e377c2', '#f7b6d2', '#7f7f7f', '#c7c7c7', '#bcbd22', '#dbdb8d', '#17becf', '#9edae5']
```

Перевод названий столбцов на русский язык

```
In [132...]: def translate_field_name(field_name, short=False):
    """Безопасно переводит название столбца набора данных на русский язык"""
    for df in data_fields.values():
        try:
            return df[field_name].desc_short if short else df[field_name].desc
        except:
            pass
    return field_name
```

Выполним проверку работы функции:

```
In [133...]: translate_field_name('total_sales')
```

```
Out[133]: 'продажи в мире в миллионах копий'
```

```
In [134...]: translate_field_name('total_sales', True)
```

```
Out[134]: 'в мире'
```

```
In [135...]: def translate_field_names(field_names_list, short=False):
    """Безопасно переводит список названий столбцов набора данных на русский язык"""
    return list(map(partial(translate_field_name, short=short), field_names_list))
```

```
In [136...]: translate_field_names(['total_sales', 'year_of_release', 'unknown field', 'platform'])
```

```
Out[136]: ['продажи в мире в миллионах копий',
           'год выпуска игры',
           'unknown field',
           'платформа игры']
```

```
In [137...]: translate_field_names(['total_sales', 'year_of_release', 'unknown field', 'platform'], short=True)
```

```
Out[137]: ['в мире', 'год выпуска', 'unknown field', 'платформа']
```

```
In [138...]: old_names_stack = []

def push_old_names(names, push):
    """Сохраняет старые названия в стек при необходимости"""
    if push:
        old_names_stack.append(names)

def translate_data_frame(data, short=False, push=False):
    """Переводит названия столбцов и индексов набора данных Pandas на русский язык,
    сохраняя старые в стек при необходимости (push).
    Возвращает набор данных с новыми названиями столбцов и индексов."""
    if isinstance(data, pd.DataFrame):
        if isinstance(data.columns, pd.core.indexes.base.Index):
            push_old_names(data.columns, push)
            data.columns = translate_field_names(data.columns, short)
        elif isinstance(data.columns, pd.core.indexes.multi.MultiIndex):
            push_old_names(data.columns.names, push)
```

```

data.columns.names = translate_field_names(data.columns.names, short)

push_old_names(data.index.names, push)
data.index.names = translate_field_names(data.index.names, short)
return data

def pop_old_names(data):
    """Восстанавливает предыдущие названия столбцов и индексов набора данных"""
    if isinstance(data, pd.DataFrame):
        data.index.names = old_names_stack.pop()

    if isinstance(data.columns, pd.core.indexes.multi.MultiIndex):
        data.columns.names = old_names_stack.pop()
    elif isinstance(data.columns, pd.core.indexes.base.Index):
        data.columns = old_names_stack.pop()

    return data

```

Процедура построения тепловой карты

```

In [139]: def plot_heat_map(title, index, column, value, aggfunc='sum', digits=2, rot=0, labelsize=12,
                      vmin=None, cmap='coolwarm', figsize=(16.5, 13)):
    """Строит тепловую карту сводной таблицы по index (строки) и column (столбцы),
    применяя групповую операцию aggfunc к полю value.
    Чертежу присваивается заголовок title"""
    plt.figure(figsize=figsize)
    tmp = table_games.pivot_table(values=[value], index=index,
                                   columns=[column], aggfunc=aggfunc, margins=False)[value]
    tmp = translate_data_frame(tmp)
    sns.heatmap(tmp, linewidths=.5, annot=True, fmt=f'{digits}f', vmin=vmin, cmap=cmap)
    plt.tick_params(axis='both', which='major', labelsize=labelsize,
                    labelleft=True, labelright = True, labelbottom = True, labeltop=True)
    plt.xticks(rotation=rot)
    plt.xlabel(data_fields['Игры'][column].desc)
    plt.yticks(rotation=0)
    plt.grid(True)
    plt.title(title)
    plt.show();

```

Выпуск игр в разные годы

Посмотрим, сколько игр выпускалось в разные годы. Важны ли данные за все периоды? (См. задание)

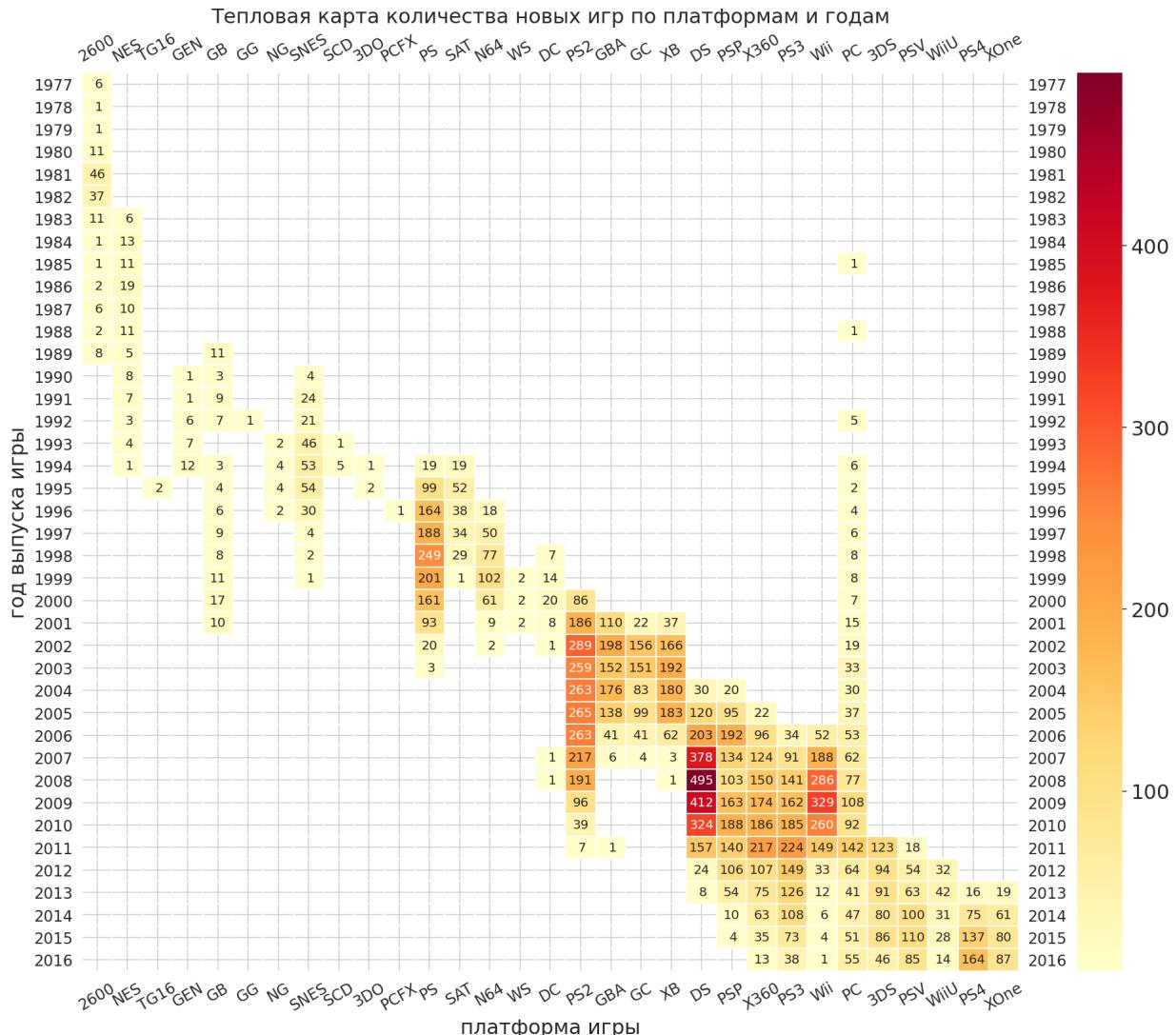
Тепловая карта количества новых игр по платформам и годам

Попробуем отобразить информацию о жизненном цикле игровых платформ и появлении новых игр по годам в наглядном, концентрированном виде:

```

In [140]: plot_heat_map('Тепловая карта количества новых игр по платформам и годам',
                  'year_of_release', 'platform', 'total_sales', 'count', digits=0, rot=30, cmap='YlOrRd')

```



Видим, что у каждой платформы есть "годы жизни":

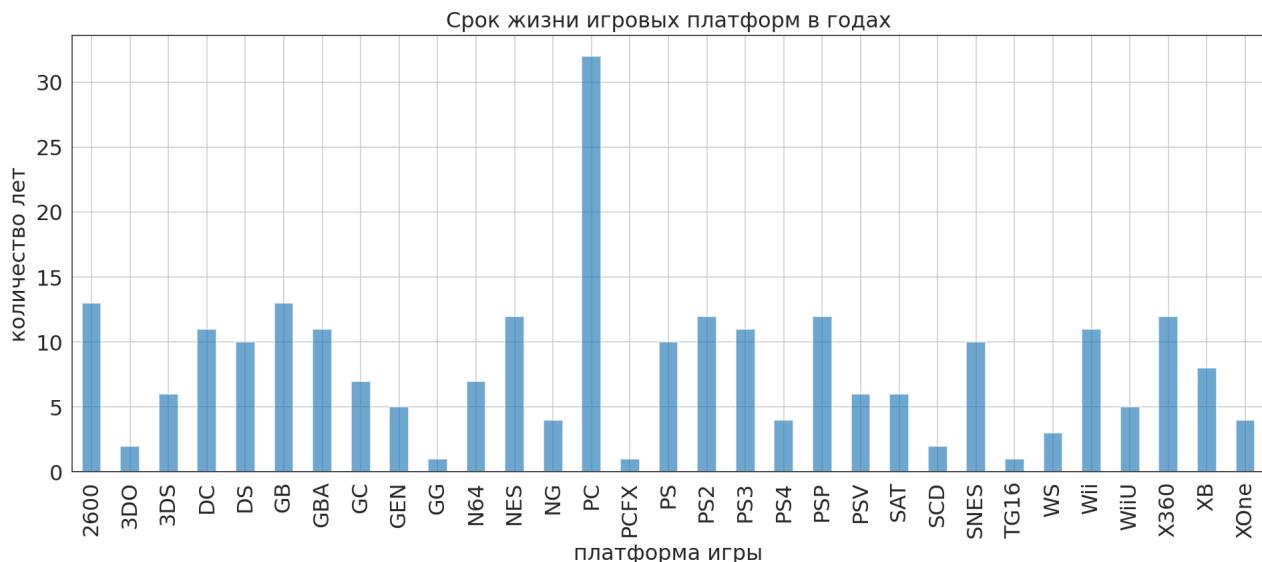
- Первая приставка (**2600**) появилась в 1977 году, последние игры для неё были выпущены в 1989 году, пик продуктивности разработчиков игр пришелся на период 1981-1982 годов.
- Следующая платформа (**NES**) вышла в свет в 1983 году, и количество новых игр для **2600** стало от года к году сокращаться ("Совпадение?! Не думаю.").
- По тепловой карте можно увидеть даже появление новых поколений - об этом свидетельствуют ступеньки. Выбивается из общего ряда лишь универсальный старожил игровой индустрии - персональный компьютер, который сам прошел через несколько модернизаций. Можно сказать, что в статистику для **PC** вносят вклад и первый домашний компьютер и его дети, внуки, правнуки и т.д.
- Складывается ощущение, что пик количества новых игр в год для игровой платформы, вероятнее всего, приходится на середину её жизненного цикла.
- Рекорд по количеству новинок в год (495) поставил в 2008 году **Nintendo DS**.
- В 2016 году новые игры появились для девяти платформ. Лидирует **PS4** (164), его догнать пытаются **XOne** (87) и **PSV** (85), следом кряхтит **PC** (55). Но из девяти в росте по новинкам только **PS4**, **XOne** и с натягом **PC**, у остальных пик пройден ранее.

На вопрос "Важны ли данные за все периоды?" ответим: "Важны, но, разумеется, в разной степени. В руках опытного биатлониста даже палка раз в год стреляет". Изучение тепловой карты за весь исторический период натолкнул нас на мысль, что у жизненного цикла игровых платформ и программного обеспечения к ним есть закономерности, знание которых может помочь в прогнозировании спроса. Не это ли задача для машинного обучения? А, Илон Маск?!

Срок жизни игровых платформ

In [141...]

```
translate_data_frame(table_platforms.set_index('platform')[['age']]).plot.bar(  
    title='Срок жизни игровых платформ в годах', ylabel='количество лет', alpha=0.65, grid=True, legend=False);
```



Игровые платформы, как живые организмы, тоже не знают секрета вечной молодости (**PC** в расчет не берем, там статистика за всё семейство персональных компьютеров с момента создания). Вспомним тех, кого уже нет в наших квартирах:

In [142...]

```
table_platforms[table_platforms.year_max < table_platforms.year_max.max()].nlargest(7, 'age')
```

Out[142...]

	platform	description	producer	generation	birth	year_min	year_max	age
0	2600	Atari 2600	Atari	2	1977	1977	1989	13
5	GB	Game Boy	Nintendo	4	1989	1989	2001	13
11	NES	Nintendo Entertainment System	Nintendo	3	1983	1983	1994	12
16	PS2	PlayStation 2	Sony	6	2000	2000	2011	12
19	PSP	PlayStation Portable	Sony	7	2004	2004	2015	12
3	DC	Dreamcast	Dreamcast	6	1998	1998	2008	11
6	GBA	Game Boy Advance	Nintendo	6	2001	2001	2011	11

Вот ещё одна особенность проявилась - максимальный срок жизни платформ от поколения к поколению сокращается. Долгожитель "Atari 2600" принадлежит первому известному нам поколению (второму в абсолютном исчислении).

А кто самый новенький на сегодняшнем рынке?

In [143...]

```
table_platforms[table_platforms.year_max == table_platforms.year_max.max()].sort_values('age')
```

Out[143...]

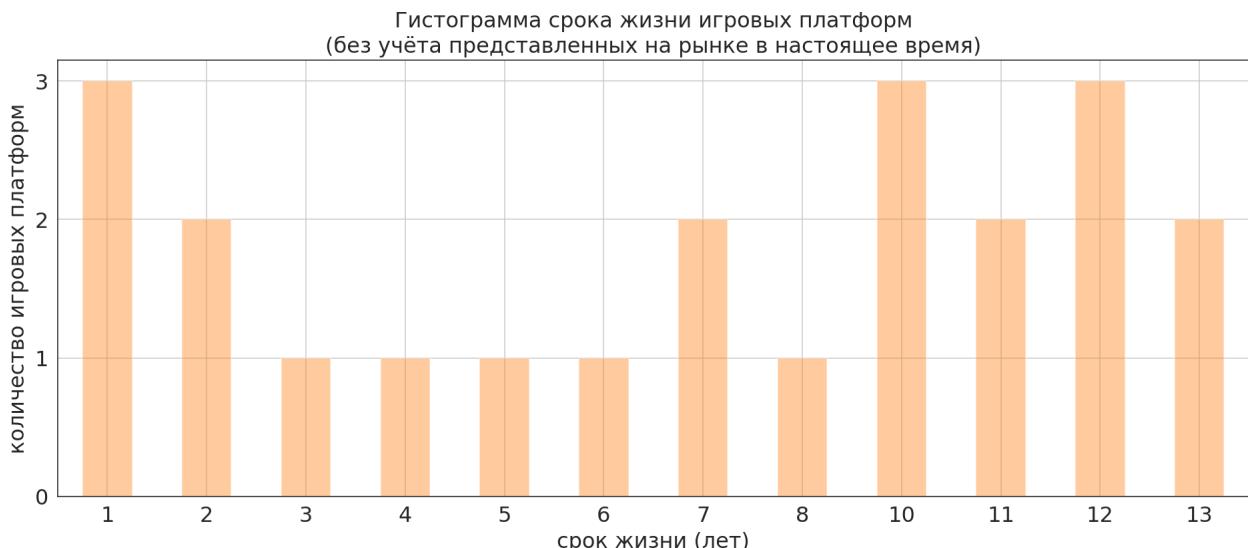
	platform	description	producer	generation	birth	year_min	year_max	age
18	PS4	PlayStation 4	Sony	8	2013	2013	2016	4
30	XOne	Xbox One	Microsoft	8	2013	2013	2016	4
27	WiiU	Wii U	Nintendo	8	2012	2012	2016	5
2	3DS	Nintendo 3DS	Nintendo	8	2011	2011	2016	6
20	PSV	PlayStation Vita	Sony	8	2011	2011	2016	6
17	PS3	PlayStation 3	Sony	7	2006	2006	2016	11
26	Wii	Wii	Nintendo	7	2006	2006	2016	11
28	X360	Xbox 360	Microsoft	7	2005	2005	2016	12

platform	description	producer	generation	birth	year_min	year_max	age	
13	PC	Персональный компьютер	Мультибренд	8	1985	1985	2016	32

Владельцы "Xbox 360", "Wii" и "PlayStation 3" скоро сами придут в магазины за новыми игровыми приставками, а игроманам с "Nintendo 3DS" и "PlayStation Vita" можно предложить скидку на утилизацию стареющей платформы при покупке новинки.

Посмотрим гистограмму срока жизни игровых платформ, на примере ретро-консолей (действующие в рассмотрение не берем):

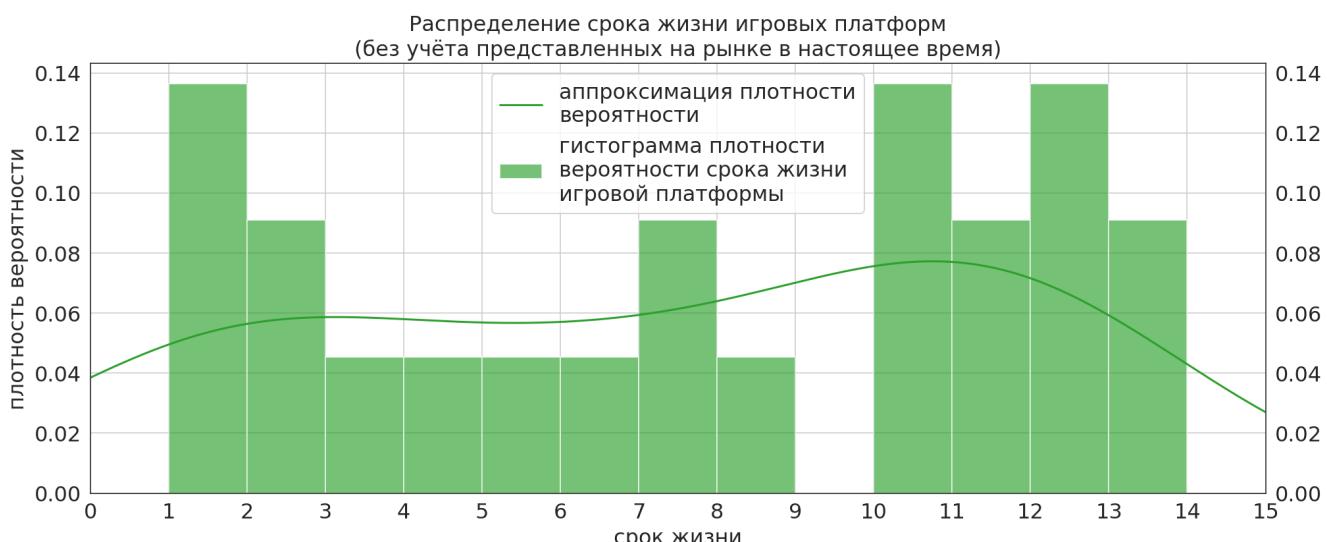
```
In [144...]: temp = table_platforms[table_platforms.year_max < table_platforms.year_max.max()][['age']].value_counts().sort_index()
ax = temp.plot.bar(color='C1', alpha=0.4, xlim=(0, None))
plt.title('Гистограмма срока жизни игровых платформ\n(без учёта представленных на рынке в настоящее время)')
plt.xlabel('срок жизни (лет)')
plt.xticks(rotation=0)
plt.ylabel('количество игровых платформ')
plt.yticks(list(range(0, temp.max()+1)))
plt.grid(True)
plt.show()
```



Чаще всего игровые платформы доживают до возраста 10-12 лет. 9 лет отсутствует в статистике, назовём этот феномен "кризисом второго [нибла](#)" - риск умереть в восемь лет у приставок очень высок. Но, если дотянул до девяти, то имеешь все шансы увидеть триумф следующего поколения.

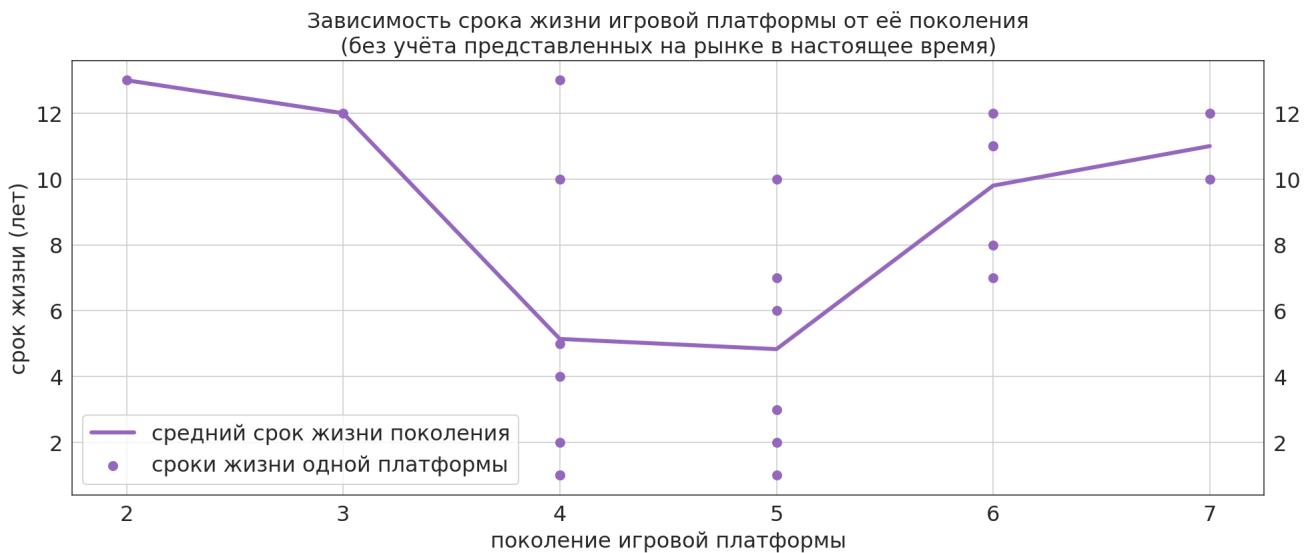
Как же выглядит распределение времени присутствия приставок и программ к ним на рынке?

```
In [145...]: temp = table_platforms[table_platforms.year_max < table_platforms.year_max.max()].set_index('platform')[['age']]
ax = temp.plot.hist(bins=list(range(1, temp.age.max()+2)), color='C2', alpha=0.65, density=True, xlim=(0, None))
temp.plot.kde(ax=ax, color='C2', xlim=(0, None))
plt.tick_params(axis='both', which='both', labelleft=True, labelright=True, labelbottom=True, labeltop=False);
plt.legend(['аппроксимация плотности вероятности', 'гистограмма плотности'])
plt.title('Распределение срока жизни игровых платформ\n(без учёта представленных на рынке в настоящее время)')
plt.xlabel('срок жизни')
plt.ylabel('плотность вероятности')
plt.xticks(list(range(0, temp.age.max()+3)))
plt.grid(True)
plt.show()
```



```
In [146...]: temp = table_platforms[table_platforms.year_max < table_platforms.year_max.max()][['generation', 'age']]
ax = temp.plot.scatter(x='generation', y='age', color='C4', lw=3)
temp = table_platforms[table_platforms.year_max < table_platforms.year_max.max()][['generation', 'age']] \
    .groupby('generation').mean()
temp.plot(color='C4', lw=3, ax=ax)
plt.tick_params(axis='both', which='both', labelleft=True, labelright=True, labelbottom=True, labeltop=False);
plt.legend(['средний срок жизни поколения', 'сроки жизни одной платформы'])
plt.title('Зависимость срока жизни игровой платформы от её поколения')
plt.xlabel(translate_field_name('generation'))
```

```
plt.ylabel('срок жизни (лет)')
plt.grid(True)
plt.show()
```



Времена быстрой смены поколений (четвертое и пятое) постепенно уходят, средний срок жизни игровой платформы стал увеличиваться. Это значит, что мы можем планировать выпуск игр на более долгую перспективу, до десяти лет вперёд, естественно, учитывая, что к концу этого периода платформа будет терять свою привлекательность.

Выпуск новых игр по годам

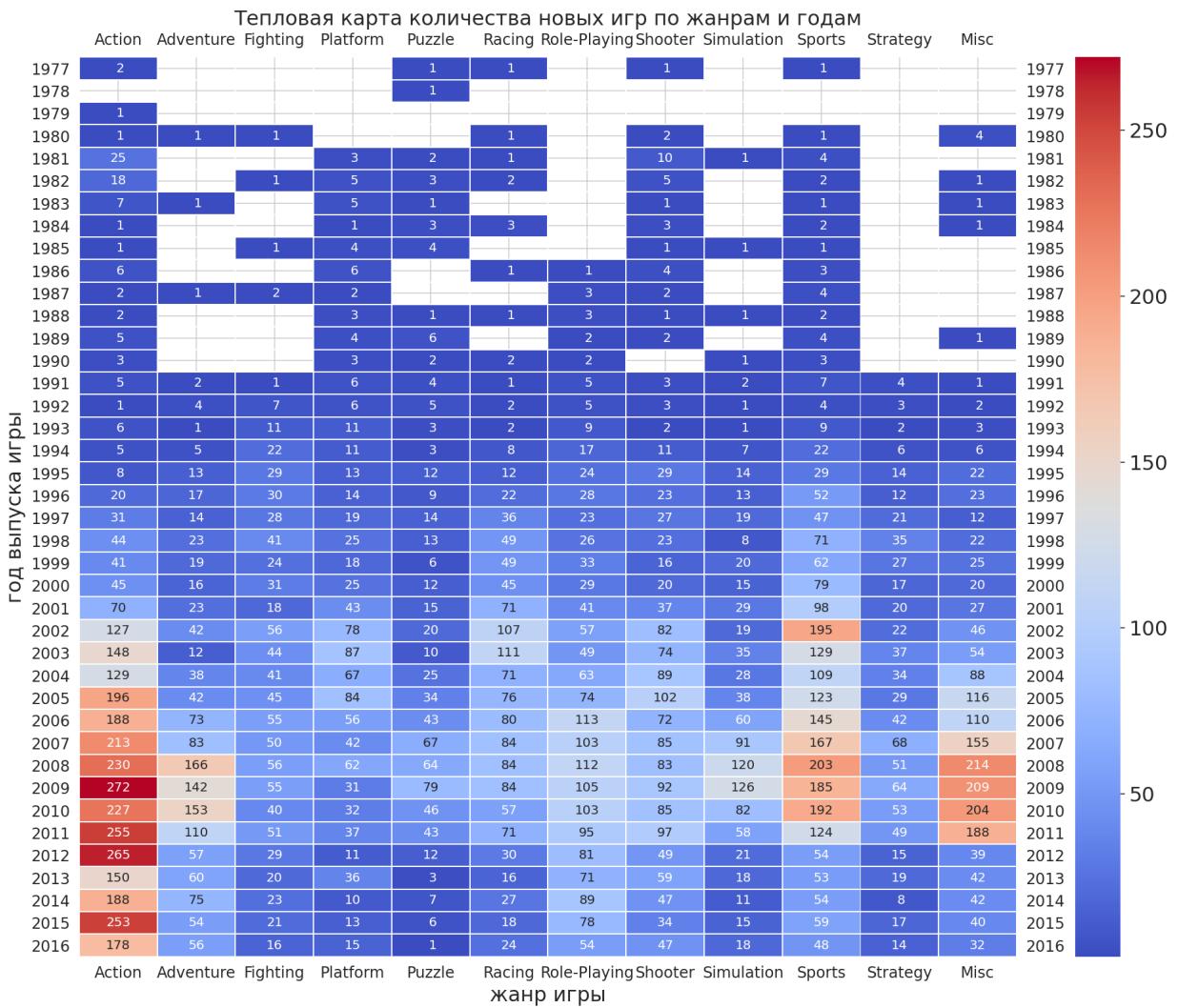
```
In [147...]: tmp = translate_data_frame(table_games[['year_of_release', 'name']].groupby('year_of_release').count())
tmp.plot.bar(color='C0', alpha=0.65, legend=False, title='Динамика выпуска новых игр по годам', grid=True)
plt.ylabel('количество новых игр')
plt.show()
```



Интенсивность выпуска новых игр неуклонно росла вплоть до 2008 года. Затем новые игры стали появляться реже. Значит ли это, что индустрия игрового программного обеспечения сдаёт позиции? Не уверен. Игры могли стать меньше, но они могли стать интереснее, технологичнее, длиннее с точки зрения прохождения и т.д. Вот поэтому очень важно **правильно** проанализировать продажи. Дойдем и до этого.

Тепловая карта количества новых игр по жанрам и годам

```
In [148...]: plot_heat_map('Тепловая карта количества новых игр по жанрам и годам',
                     'year_of_release', 'genre', 'total_sales', 'count', digits=0, cmap='coolwarm')
```

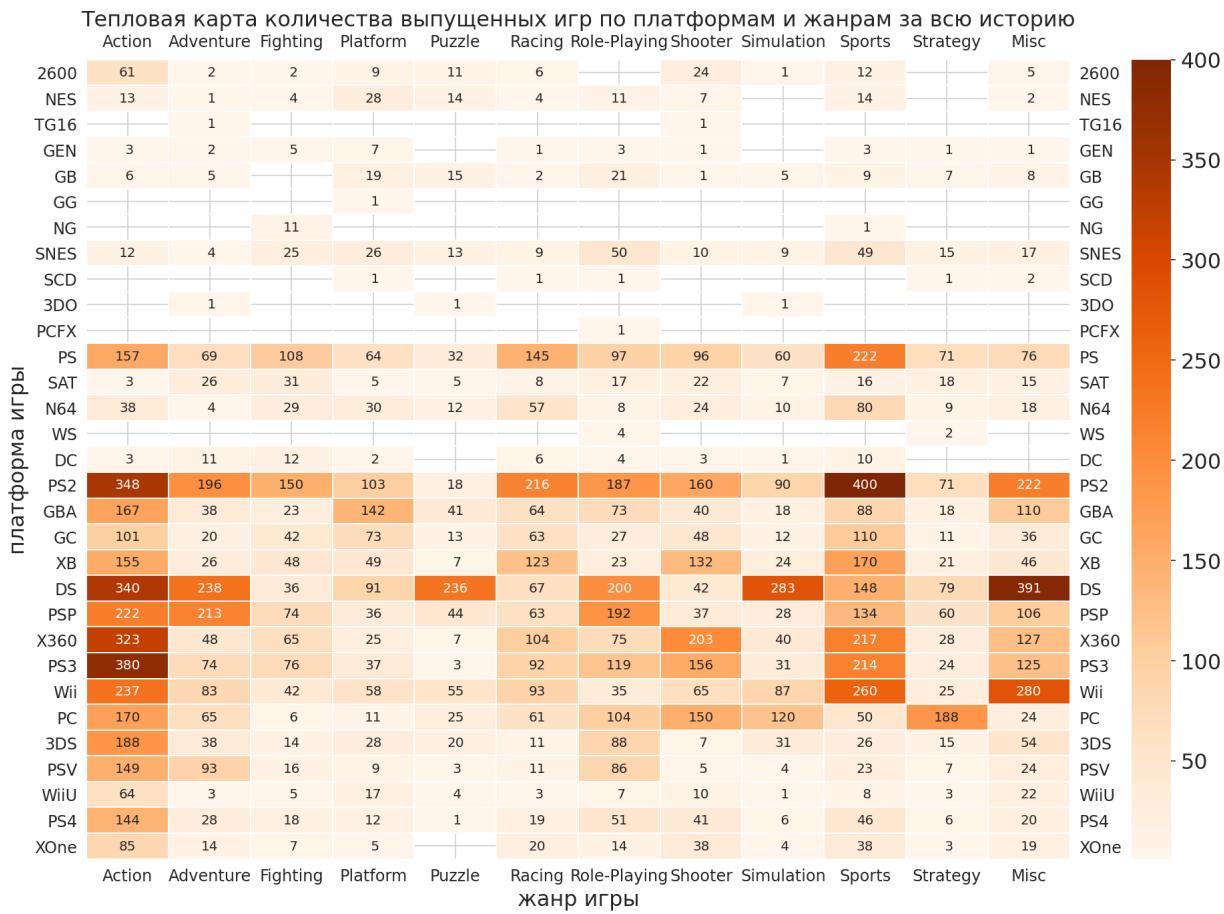


Начиная с 2003 года, игры жанра 'Action' удерживают пальму первенства по новинкам в год, выхвачив её у спортивных игр, которые лидировали с 1996 по 2002 годы. Наиболее яркие всплески в выпуске новых игр, как мы можем видеть на тепловой карте, происходили в жанре "Action" в 2008-2012 годах. Примерно в этот же период, если сравнивать с другими годами, появлялось много спортивных ("Sports") и прочих ("Misc") игр.

Катализатором, возможно, стало увеличение вычислительной мощи игровых платформ.

Тепловая карта количества выпущенных игр по платформам и жанрам

```
In [149]: plot_heat_map('Тепловая карта количества выпущенных игр по платформам и жанрам за всю историю',
                     'platform', 'genre', 'total_sales', 'count', digits=0, cmap='Oranges', figsize=(16.5, 11))
```



Невооруженным взглядом видно, что больше всего игр было выпущено для "Sony PlayStation 2", а программисты реализовывали свои таланты чаще в жанре "Action".

Доли жанров в общей структуре игрового программного обеспечения

Посмотрим, как распределяли свои усилия разработчики игр по жанрам за всю историю отрасли:

```
In [150]: tmp = table_games.groupby('genre').size()
tmp.plot(kind='pie', subplots=True, colors=fill_colors, autopct='%1.1f%')
plt.title('Доли жанров игр за всю историю')
plt.ylabel(''); plt.show()
```



Жанры **Action** и **Sports** больше всего привлекают продюсеров. А как менялись их вкусы со временем?

```
In [151]: tmp = table_games.pivot_table(index='year_of_release', columns='genre', values='name', aggfunc='count',
                                         fill_value=0, margins=True, margins_name='total')
# переводим в проценты
tmp = tmp.divide(tmp.loc[:, 'total'], axis='rows') * 100
# отсекаем итоги
tmp = translate_data_frame(tmp.iloc[:-1, :-1])
tmp.loc[:, ['Action', 'Sports', 'Shooter', 'Misc', 'Adventure', 'Role-Playing']].plot(grid=True)
plt.tick_params(axis='both', which='major', labelleft=True, labelright = True, labelbottom = True, labeltop=False)
plt.xticks(ticks=tmp.index.to_list(), rotation=90)
plt.ylabel('Доля в процентах')
plt.yticks(rotation=0)
plt.grid(True)
plt.title('Доли основных игровых жанров в новых играх по годам')
plt.show();
```



А вот и еще одно подтверждение, что с 2003 года разработчики игр стали сосредотачивать свои усилия на 'Action' вместо 'Sports'.

```
In [152...]
tmp = table_games.pivot_table(index='year_of_release', columns='genre', values='name', aggfunc='count')
tmp = translate_data_frame(tmp)
tmp.plot.bar(alpha=0.7, legend=True, title='Выпуск новых игр по годам с учетом жанров',
             grid=True, stacked=True, color=fill_colors)
plt.ylabel('количество новых игр')
plt.show()
```

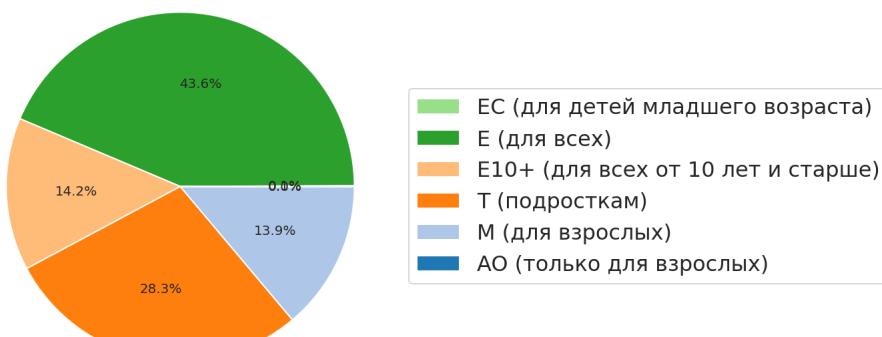


Доли рейтингов в общей структуре игрового программного обеспечения

Посмотрим, какие известные рейтинги присваивались новым играм:

```
In [153...]
tmp = table_games.groupby('rating_rus').size()
tmp.plot(kind='pie', subplots=True, colors=fill_colors[5:-1], autopct='%1.1f%%', labeldistance=None)
plt.title('Доли известных рейтингов игр за всю историю')
plt.ylabel('')
plt.legend(loc='center left', bbox_to_anchor=(1.0, 0.5))
plt.show()
```

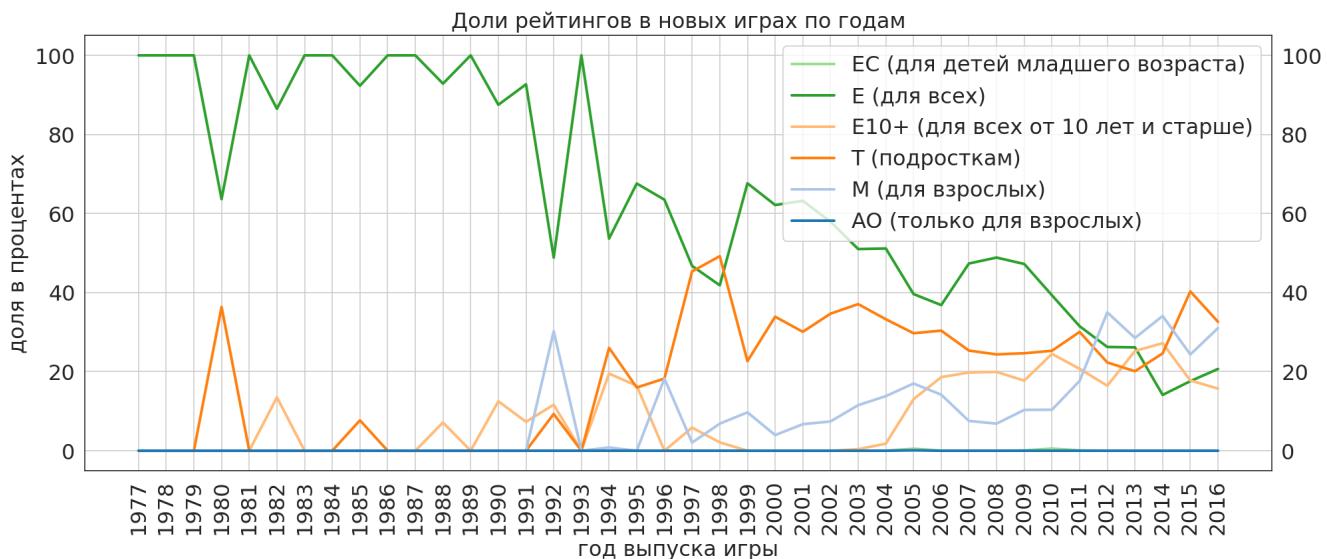
Доли известных рейтингов игр за всю историю



Доли игр с рейтингом **для самых маленьких** и **самых взрослых** ничтожно малы. Посмотрим, как менялись доли во времени:

```
In [154...]
tmp = table_games.pivot_table(index='year_of_release', columns='rating_rus', values='name', aggfunc='count',
                               fill_value=0, margins=True, margins_name='total')
# переводим в проценты
tmp = tmp.divide(tmp.loc[:, 'total'], axis='rows') * 100
```

```
# отсекаем итоги
tmp = translate_data_frame(tmp.iloc[:-1, :-1])
tmp.plot(grid=True, lw=2, color=fill_colors[5:-1])
plt.tick_params(axis='both', which='major', labelleft=True, labelright = True, labelbottom = True, labeltop=False)
plt.xticks(ticks=tmp.index.to_list(), rotation=90)
plt.ylabel('доля в процентах')
plt.yticks(rotation=0)
plt.grid(True)
plt.title('Доли рейтингов в новых играх по годам')
plt.show();
```



В последнее время верх в рейтингах берут игры для взрослых и подростков.

```
In [155...]
tmp = table_games.pivot_table(index='year_of_release', columns='rating_rus', values='name', aggfunc='count')
tmp = translate_data_frame(tmp)
tmp.plot.bar(alpha=0.8, legend=True, title="Выпуск новых игр по годам с учетом рейтингов",
            grid=True, stacked=True, color=fill_colors[5:-1])
plt.ylabel('количество новых игр'); plt.show()
```



Продажи игр

Дойдя до этого пункта, читатель, наверняка, в предвкушении подумал: "Ну, наконец-то! Анализ, анализ..."

Так оно бы и было, если бы не автор с его нездоровым педантизмом, аналитическим мышлением (нужное подчеркнуть ☺).

В задании заказчик (уверен, это не Яндекс.Практикум) неоднократно упоминает связь продаж и календарных лет. Например:

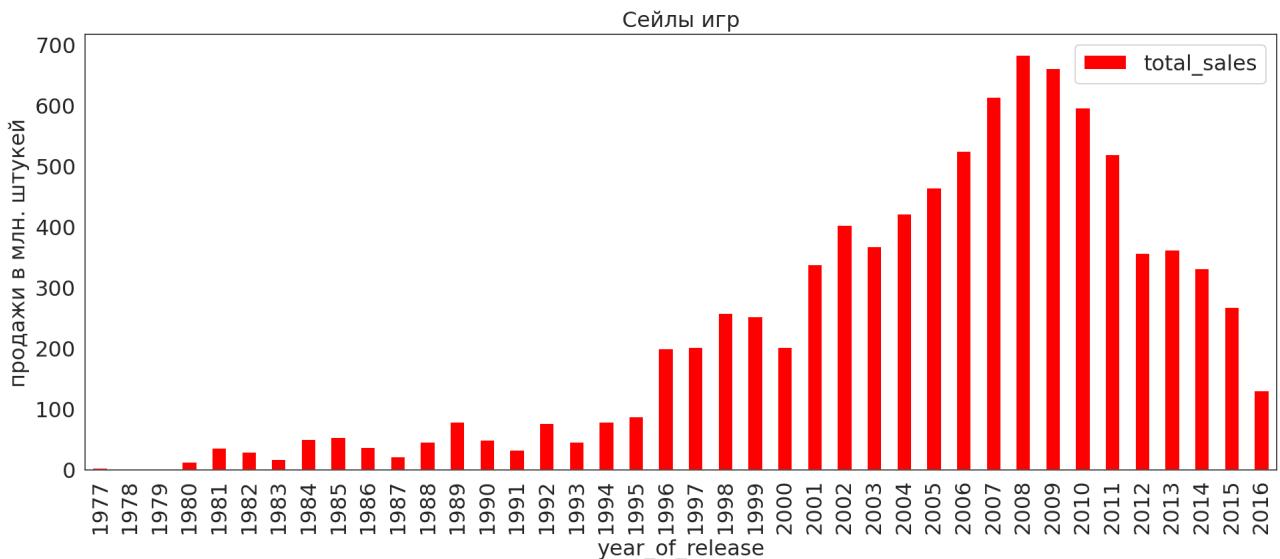
"Выберите платформы с наибольшими суммарными продажами и постройте распределение по годам."

"Какие платформы лидируют по продажам, растут или падают? Выберите несколько потенциально прибыльных платформ."

Скажу сразу, провести дальнейший анализ мы не сможем без преобразования нашего набора данных, в котором на текущий момент все продажи привязаны к году выпуска игры и не учитывают срок её присутствия на рынке... Плохая игра, появившаяся раньше, за 5-6 лет может собрать такую же статистику по продажам, как более новая, но хорошая за год. Последствия такого анализа нетрудно предугадать - фиаско. Дисперсии выборок объемов продаж по годам, платформам, регионам, жанрам, оценкам и т.п. будут завышены, так как для одних игр итог реализации будет считаться, например, за 1-2 года, а для других за 5-6 лет просто потому, что они появились на разных этапах жизненного цикла своей платформы...

Рассмотрим пример неправильного графика продаж, сделанного нашими конкурентами из аналитического агентства "Рога и копыта". Они посчитали итоговые продажи без учета многолетнего периода реализации отдельной игры в торговых сетях, привязав всё к году появления продукта на рынке. По их понятиям о бизнесе, игра выпустилась... и продалась вся в первый же год целиком:

```
In [156...]
tmp = table_games[['year_of_release', 'total_sales']].groupby('year_of_release').sum()
tmp.plot.bar(color='red', title='Сейлы игр')
plt.ylabel('продажи в млн. штукей')
plt.show()
```



Не будем строго судить "рогокопытцев" за мерзотный цвет, подписи графика и другие недостатки. Главное не в этом, а в том, что они сделали **совершенно неправильный вывод: "Общие продажи игр неуклонно падают. Отрасль загибается. Переводим активы в производство надувных шариков."**



Взрыв на макаронной фабрике

Мы пойдем другим путём и воспользуемся функцией `explode` (англ. взрываться).

План таков:

- Для каждой игры определяем годы её присутствия на рынке:
 - начало - поле `year_of_release` (год выпуска);
 - завершение - `year_max` (год окончания жизни игровой платформы, не зря же мы [вычислили](#) это поле).
- Посчитаем также продолжительность присутствия игры на рынке в годах - срок жизни.
- Разбиваем статистику продаж игры по годам её жизни, вычисляя продажи в отдельный год, как среднее (т.е. делим `total_sales` и продажи по регионам на срок жизни программы). Вместо среднего можно было бы выбрать и другой способ. Например, определять долю продаж в конкретный год случайной величиной из [распределения Вейбулла](#), т.е. учесть продаваемость в первый, второй и последующие годы. Совсем, как в жизни (хотя выявление этой закономерности может стать хорошей задачей для машинного обучения):

Super Mario Galaxy (Wii)

Updates Summary Reviews Screens FAQs Cheats Extras Forum Sales

Latest Sales Updates

12,800,000	Marth	07th May 2020
12,790,000	Marth	25th Apr 2019
12,780,000	Machina	08th Jan 2019

Sales History

Total Sales	1.20m	6.06m	3.38m	0.76m	11.40m
	Japan	NA	Europe	Others	Total
1	260,993	n/a	n/a		250,553
2	78,563	n/a	n/a		75,420
3	44,750	526,910	153,992	92,403	818,055
4	35,917	319,004	126,500	59,915	541,336
5	38,671	143,964	126,099	35,151	343,885
6	46,118	184,818	124,341	40,449	395,726
7	73,211	229,869	148,235	49,009	500,324
8	118,987	463,364	176,808	84,114	843,273
9	72,913	193,581	132,022	41,847	440,363
10	73,000	93,947	79,587	21,143	267,677

Судя по информации с сайта об играх, продажи в разные периоды могут отличаться. Но в нашей задаче это не так важно, мы усредним показатели по годам (приведем продажи к одинаковому временному масштабу), а [центральная предельная теорема](#) только укрепит уверенность в здравости нашей идеи.

Преобразуем нашу таблицу с играми:

```
In [157...]: # присоединим последний год жизни платформы к играм
table_games = table_games.join(table_platforms.set_index('platform')[['year_max']], on='platform')
```

```
In [158...]: # Вычислим возраст игры
table_games['age'] = table_games['year_max'] - table_games['year_of_release'] + 1
```

```
In [159...]: # делим все продажи на возраст игры, получаем усредненные продажи в год, заносим в отдельные поля
name_suffix = '_annual'
old_names = table_games.filter(like='sales').columns
for field in old_names:
    table_games[field + name_suffix] = table_games[field] / table_games['age']
```

```
In [160...]: # создаем копию таблицы игр
sales = table_games.copy()
# удаляем из неё общие продажи
sales.drop(columns=old_names, inplace=True)

# Вычисляем список лет жизни программы
sales['year'] = sales[['year_of_release', 'year_max']].apply(lambda x: list(range(x[0],x[1]+1)), axis=1)
# отбрасываем у полей суффикс
sales.rename(lambda x: str.replace(x, name_suffix, ''), axis='columns', inplace=True)
display(sales)
```

	name	platform	year_of_release	genre	...	jp_sales	other_sales	total_sales	year
0	Wii Sports	Wii	2006	Sports	...	0.343	0.768	7.504	[2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013]
1	Super Mario Bros.	NES	1985	Platform	...	0.681	0.077	4.024	[1985, 1986, 1987, 1988, 1989, 1990, 1991, 1992]
2	Mario Kart Wii	Wii	2008	Racing	...	0.421	0.366	3.947	[2008, 2009, 2010, 2011, 2012, 2013, 2014, 2015]
3	Wii Sports Resort	Wii	2009	Sports	...	0.410	0.369	4.096	[2009, 2010, 2011, 2012, 2013, 2014, 2015, 2016]
4	Pokemon Red/Pokemon Blue	GB	1996	Role-Playing	...	1.703	0.167	5.230	[1996, 1997, 1998, 1999, 2000, 2001]
...
16710	Samurai Warriors: Sanada Maru	PS3	2016	Action	...	0.010	0.000	0.010	[2016]
16711	LMA Manager 2007	X360	2006	Sports	...	0.000	0.000	0.001	[2006, 2007, 2008, 2009, 2010, 2011, 2012, 2013]

					genre	...	jp_sales	other_sales	total_sales	year
16712	Haitaka no Psychedelica	PSV	2016	Adventure	...	0.010	0.000	0.010		[2016]
16713	Spirits & Spells	GBA	2003	Platform	...	0.000	0.000	0.001	[2003, 2004, 2005, 2006, 2007, 2008, 2009, 201...	
16714	Winning Post 8 2016	PSV	2016	Simulation	...	0.010	0.000	0.010		[2016]

16713 rows × 17 columns

Обращаем внимание на столбец `year` в таблице сверху, он содержит список лет, которые могла продаваться игра. Убедившись, что там всё нормально, наблюдаем за "взрывом на макаронной фабрике" - разбиением записей таблицы по годам жизни программы:

```
In [161...]: sales = sales.explode(column='year')
sales['year'] = sales['year'].astype(np.int64)
```

Обязательно проверим равенство итоговых показателей в исходной и преобразованной таблицах с играми:

```
In [162...]: for field in sales.filter(like='sales').columns:
    print(f'Разница по полу {translate_field_name(field)}: '
          f'{abs(1000000 * (table_games[field].sum() - sales[field].sum())):.00f} копий')
```

Разница по полу "продажи в Северной Америке в миллионах копий": 0 копий
 Разница по полу "продажи в Европе в миллионах копий": 0 копий
 Разница по полу "продажи в Японии в миллионах копий": 0 копий
 Разница по полу "продажи в других странах в миллионах копий": 0 копий
 Разница по полу "продажи в мире в миллионах копий": 0 копий

Получили новую таблицу со скорректированными объемами продаж по годам жизни игровых программ:

```
In [163...]: display(sales)
```

	name	platform	year_of_release	genre	...	jp_sales	other_sales	total_sales	year
0	Wii Sports	Wii	2006	Sports	...	0.343	0.768	7.504	2006
0	Wii Sports	Wii	2006	Sports	...	0.343	0.768	7.504	2007
0	Wii Sports	Wii	2006	Sports	...	0.343	0.768	7.504	2008
0	Wii Sports	Wii	2006	Sports	...	0.343	0.768	7.504	2009
0	Wii Sports	Wii	2006	Sports	...	0.343	0.768	7.504	2010
...
16713	Spirits & Spells	GBA	2003	Platform	...	0.000	0.000	0.001	2008
16713	Spirits & Spells	GBA	2003	Platform	...	0.000	0.000	0.001	2009
16713	Spirits & Spells	GBA	2003	Platform	...	0.000	0.000	0.001	2010
16713	Spirits & Spells	GBA	2003	Platform	...	0.000	0.000	0.001	2011
16714	Winning Post 8 2016	PSV	2016	Simulation	...	0.010	0.000	0.010	2016

106025 rows × 17 columns

Зададим описания новых полей для вывода на графиках:

```
In [164...]: data_fields['Продажи'] = data_fields['Игры'].copy()
```

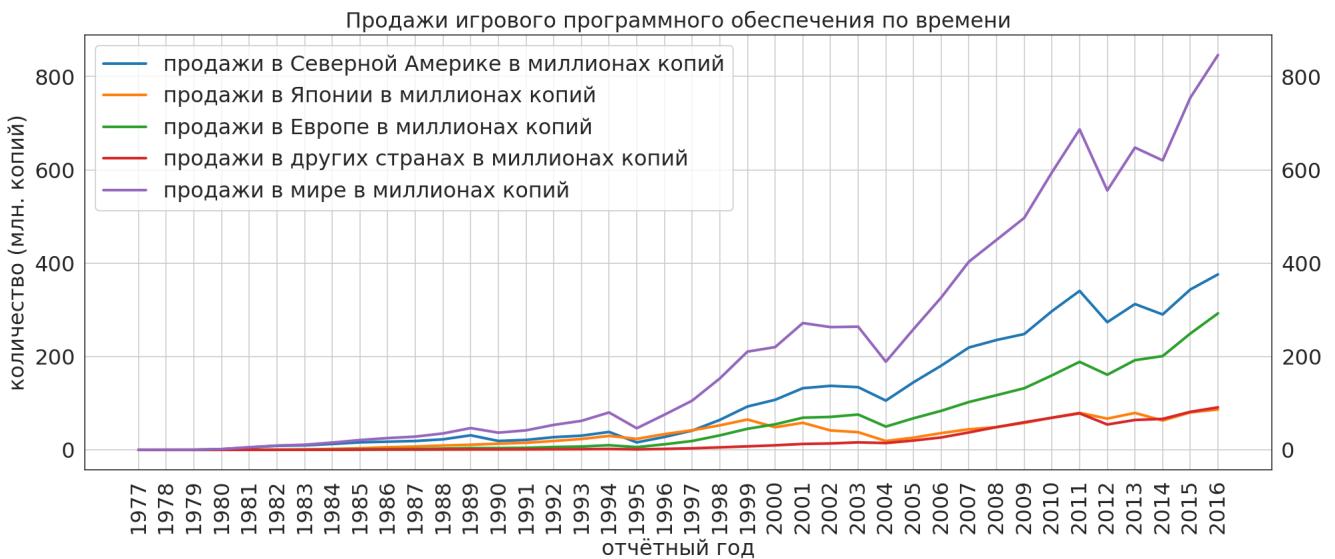
```
In [165...]: data_fields['Продажи']['year'] = DataField('отчётный год', 'отчётный год', np.int64)
```

Продажи игр по годам

И вот теперь проанализируем рынок игрового программного обеспечения, оценив динамику изменения общего количества продаж дистрибутивов игр по регионам и в целом:

```
In [166...]: tmp = translate_data_frame(sales.groupby('year')[['na_sales', 'jp_sales', 'eu_sales', 'other_sales', 'total_sales']].sum())
display(tmp.tail(10))
tmp.plot(lw=2, grid=True);
plt.tick_params(axis='both', which='major', labelleft=True, labelright = True, labelbottom = True, labeltop=False)
plt.xticks(ticks=tmp.index.to_list(), rotation=90)
plt.ylabel('количество (млн. копий)')
plt.title('Продажи игрового программного обеспечения по времени')
plt.show();
```



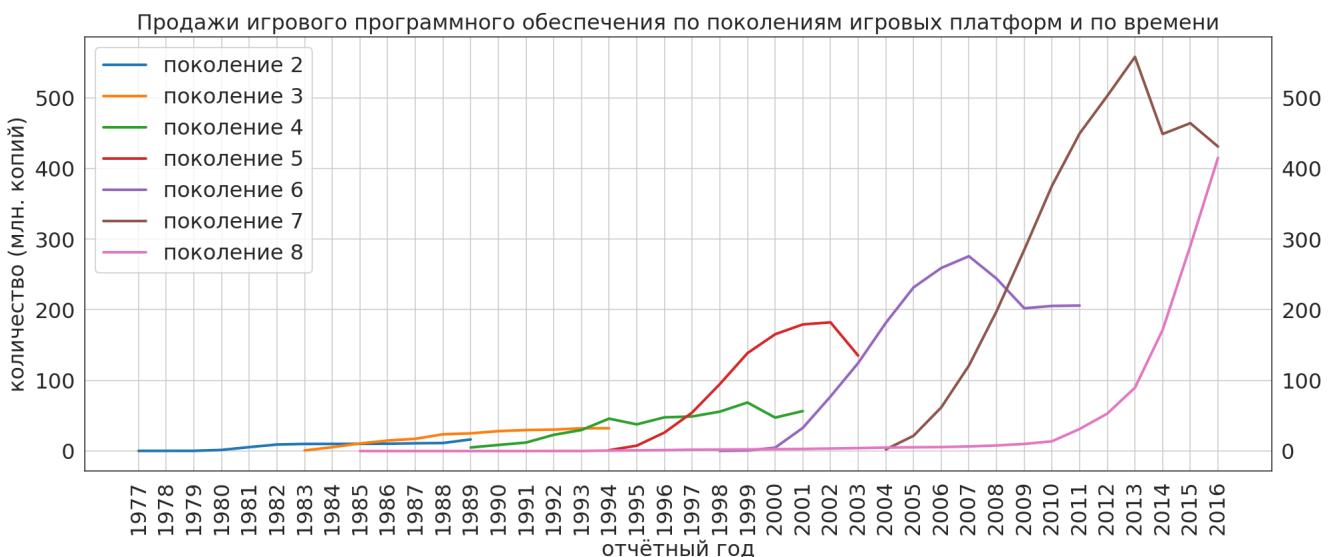


Делаем вывод: "Общие продажи игр уверенно растут во всех регионах (темперы выше в Северной Америке). Спрос стимулирует развитие отрасли. Переводим активы из производства надувных шариков в создание игр для перспективных платформ." и сравниваем с конкурентами из "Рогов и копыт". Как тебе такое, Илон Маск?!"

Продажи игр по годам для различных поколений игровых платформ

И вот теперь проанализируем рынок игрового программного обеспечения, оценив динамику изменения общего количества продаж дистрибутивов игр по регионам и в целом:

```
In [167]: tmp = sales.pivot_table(index='year', columns='generation', values='total_sales', aggfunc='sum')
tmp = translate_data_frame(tmp)
tmp.plot(lw=2, grid=True)
plt.tick_params(axis='both', which='major', labelleft=True, labelright = True, labelbottom = True, labeltop=False)
plt.xticks(ticks=tmp.index.to_list(), rotation=90)
plt.ylabel('количество (млн. копий)')
plt.legend(labels=list(map('поколение {:d}'.format, tmp.columns)))
plt.title('Продажи игрового программного обеспечения по поколениям игровых платформ и по времени')
plt.show();
```



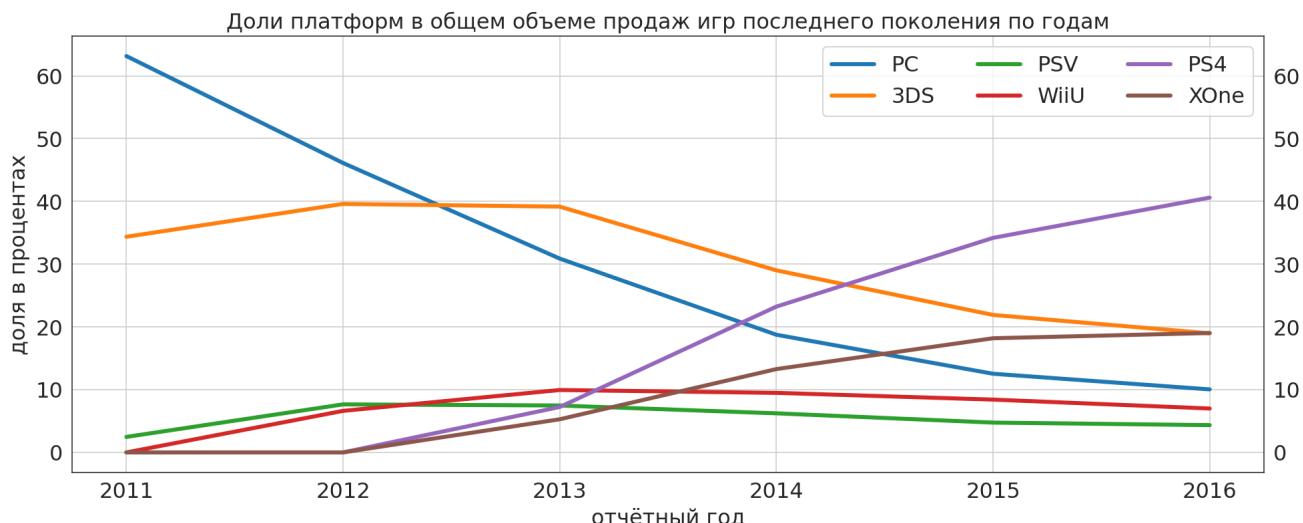
Делаем вывод: "Программы для каждого нового поколения игровых устройств продаются в большем количестве, по сравнению с предыдущим. Игровые платформы последнего (восьмого) поколения еще далеко не исчерпали потенциал для роста. Седьмое поколение сдаёт позиции, связывать свои перспективы с ним уже не стоит."

```
In [168]: # определяем номер последнего поколения
generation = table_platforms.generation.max()
# определяем год рождения первого представителя платформы, кроме PC (он жил при всех режимах)
min_year = table_platforms[(table_platforms.generation == generation) &
                           (table_platforms.platform != 'PC')].birth.min()
display(HTML(f'<br><b>Актуальный период ограничим {min_year} годом, с этого момента ' +
            f'начало развиваться современное ({generation}) поколение игровых платформ. Изучим ' +
            f'доли каждой из них в общем объеме продаж игр последнего поколения по годам</b>'))
# отбираем продажи по последнему поколению
perspective_sales = sales[(sales.generation == generation) & (sales.year >= min_year)]
tmp = perspective_sales.pivot_table(index='year', columns='platform', values='total_sales', aggfunc='sum',
                                      fill_value=0, margins=True, margins_name='total')
tmp = translate_data_frame(tmp)
# переводим в проценты
tmp = tmp.divide(tmp.loc[:, 'total'], axis='rows') * 100
# отсекаем итого
tmp = translate_data_frame(tmp.iloc[:-1, :-1])
display(tmp)
tmp.plot(lw=3, grid=True)
plt.tick_params(axis='both', which='major', labelleft=True, labelright = True, labelbottom = True, labeltop=False)
plt.ylabel('доля в процентах')
plt.title('Доля платформ в общем объеме продаж игр последнего поколения по годам')
```

```
plt.legend(ncol=3)
plt.show();
```

Актуальный период ограничим 2011 годом, с этого момента начало развиваться современное (8) поколение игровых платформ. Изучим доли каждой из них в общем объеме продаж игр последнего поколения по годам

	PC	3DS	PSV	WiiU	PS4	XOne
отчётный год						
2011	63.158	34.370	2.473	0.000	0.000	0.000
2012	46.121	39.600	7.657	6.621	0.000	0.000
2013	30.887	39.168	7.477	9.945	7.241	5.282
2014	18.751	29.014	6.228	9.490	23.237	13.280
2015	12.538	21.910	4.753	8.417	34.189	18.194
2016	10.046	18.976	4.349	6.997	40.603	19.029



Очевидно, что **PlayStation 4** (игровая приставка) стремительно завоевывает рынок, её доля в общей структуре продаж в 2016 году достигла 40.6%. Конкурентам в лице **Nintendo 3DS** (портативная консоль) и **Xbox One** (игровая приставка) лидера не догнать, но их ресурса может хватить ещё года на четыре (на двоих они пока владеют около 38% рынка). **Персональный компьютер** продолжит пользоваться своим преимуществом универсального устройства, но их доля вряд ли сможет подняться выше 10% (пока заметен только явный тренд на снижение).

Сформируем **рейтинг перспективности**:

1. **PlayStation 4**
2. **Xbox One**
3. **Nintendo 3DS**
4. **Персональный компьютер**

Распределение оценок

Функция построения сравнительных графиков распределений

Настало время для описания функции, которая поможет нам удобно визуализировать данные о поведении клиентов:

```
In [169...]: filters = {
    'все': perspective_sales.year == perspective_sales.year,
    'PlayStation 4': perspective_sales.platform == 'PS4',
    'Xbox One': perspective_sales.platform == 'XOne',
    'Nintendo 3DS': perspective_sales.platform == '3DS',
    'ПК': perspective_sales.platform == 'PC',
    'жанр Action': perspective_sales.genre == 'Action',
    'жанр Sports': perspective_sales.genre == 'Sports',
    '2011 год': perspective_sales.year == 2011,
    '2015 год': perspective_sales.year == 2015
}
```

```
In [170...]: # словарь для переименования полей результата работы функции describe
stats_dict = {'count': 'количество образцов', 'mean': 'среднее значение',
              'std': 'среднеквадратичное отклонение',
              'min': 'минимум', 'max': 'максимум'}
```

```
In [171...]: def plot_dist_info(df, field_name, slice_names, force='', hist=False, min_x=0, max_x=None,
                      start_color=0, plot_sigma=True, fill=False, step=None, grid=True,
                      loc='upper right', figsize=(16, 6), show_table=True):
    """Вспомогательная функция для вычисления характеристик распределения, вывода отчета и построения графиков зависимости от force значений среза набора данных df по полю field_name и abk slice_names"""
    # создаем полотно
    fig, ax = plt.subplots(figsize=figsize)

    # словарь для заполнения характеристиками распределений
    stat = dict()

    # задаем начальный номер цвета
```

```

color = start_color

# выполняем для всех заданных срезов
for i, slice_name in enumerate(slice_names):
    # нарезаем набор данных
    data = df[filters[slice_name]]

    # находим параметры распределения
    desc = data[field_name].describe()
    desc.loc['дисперсия'] = desc.loc['std'] ** 2
    stat[slice_name] = desc
    # считаем по выборке для перекрестной проверки
    x_mean = data[field_name].mean()
    sigma = data[field_name].std()

    # формируем подпись кривой
    label=f'{translate_field_name(field_name, short=True)} ({slice_name})\n\mu={x_mean:.02f}, \sigma={sigma:.02f}, \sigma^2={sigma**2:.02f}'

    if hist:
        # чертим гистограмму индивидуальным цветом
        sns.histplot(data=data, x=field_name, ax=ax, fill=fill,
                      label=label, color=f'C{color}', lw=2, kde=True)
    else:
        # чертим график индивидуальным цветом
        sns.kdeplot(data=data, x=field_name, ax=ax, fill=fill,
                      label=label, color=f'C{color}', lw=2)

    # выбираем следующий цвет, если рисуем всего один график
    color += len(slice_names) == 1

    # чертиим среднее значение
    ax.axvline(x_mean, color=f'C{color}', ls='--', lw=1.5,
                label=f'среднее: $\mu={x_mean:.02f}' if i == 0 else None)
    # выбираем следующий цвет, если рисуем всего один график
    color += len(slice_names) == 1

    # чертиим медиану
    x = data[field_name].median()
    ax.axvline(x, color=f'C{color}', ls='-.', lw=1.5,
                label=f'медиана: {x:.02f}' if i == 0 else None)

    # для первого среза чертим линии трех сигм
    if plot_sigma and i == 0:
        # выбираем следующий цвет, если рисуем всего один график
        color += len(slice_names) == 1
        for j in (-3, -2, -1, 1, 2, 3):
            dx = j * sigma
            x = x_mean + dx
            if j >= 1:
                interval = f'\sigma=[\max(0, x_mean-dx):.02f, {x:.02f}]'
            if j == 1:
                label = f'$\mu\pm{interval}'
            else:
                label = f'$\mu\pm{j}\cdot\sigma{interval}'

            else:
                label = None
            ax.axvline(x, color=f'C{color}', ls=':', lw=1.5*abs(j), label=label)
        # выбираем следующий цвет
        color += 1

    # настраиваем ось x
    ax.set_xlim(min_x, max_x)
    if step:
        ax.xaxis.set_major_locator(ticker.MultipleLocator(step))

    # настраиваем общий вид чертежа
    plt.xlabel(f'{translate_field_name(field_name, short=True)}')
    if hist:
        plt.ylabel('количество наблюдений')
    else:
        plt.ylabel('плотность вероятности')
    plt.title(f'Как распределяется {translate_field_name(field_name, short=True)}{force}')
    plt.grid(grid)

    # выводим подписи к графикам
    handles, labels = ax.get_legend_handles_labels()
    if labels[0].startswith('среднее'):
        labels = labels[5:6] + labels[0:5] + labels[6:]
        handles = handles[5:6] + handles[0:5] + handles[6:]
    plt.legend(handles, labels, loc='best', framealpha=1)

    # выводим параметры распределения в удобном виде
    if show_table:
        stat = pd.DataFrame(stat).rename(index=stats_dict)
        display(HTML(f'  
Изучим, как распределяется {translate_field_name(field_name, short=True)}{force}:'))
        display(stat)

    # отображаем чертеж
    plt.show();

```

Распределение оценок пользователей

In [172...]:

```
plot_dist_info(perspective_sales, 'user_score', ['все'], 'для игр последнего поколения (гистограмма)', hist=True, fill=True, step=1, show_table=False, max_x=10)
```



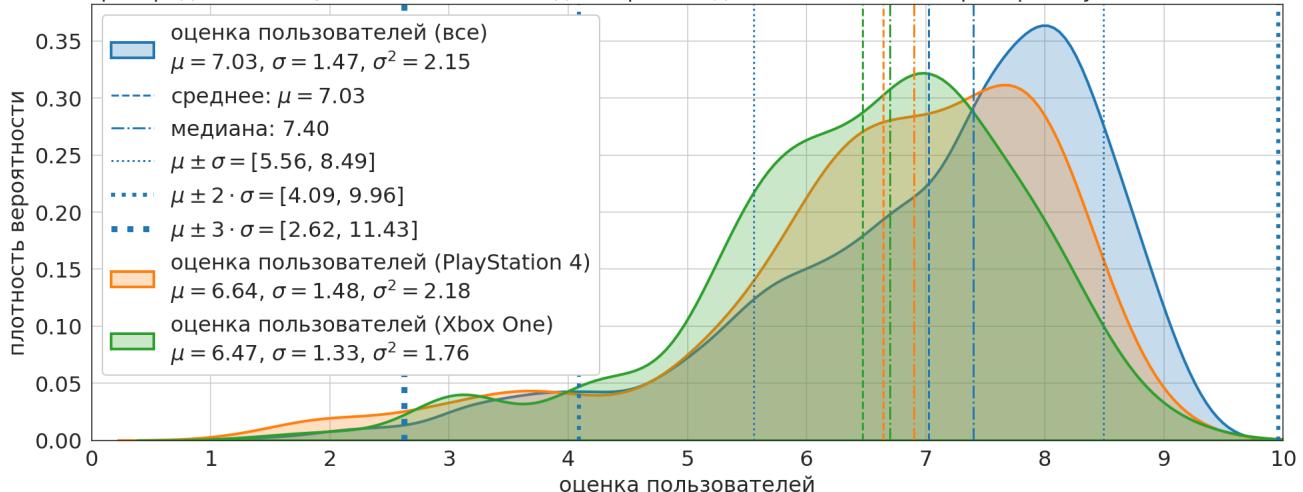
Пользователи, в целом, высоко оценивают игры, которые покупают. Для актуальных платформ средние оценки выше 7 по 10-ти бальной шкале. Но распределение имеет тяжелый левый хвост, что свидетельствует о наличии крайне раздосадованных игроков. Отличаются ли оценки для разных платформ?

```
In [173]: plot_dist_info(perspective_sales, 'user_score', ['все', 'PlayStation 4', 'Xbox One'],
                     'для игр последнего поколения на примере PlayStation 4 и Xbox One',
                     hist=False, fill=True, max_x=10, step=1)
```

Изучим, как распределяется оценка пользователей для игр последнего поколения на примере PlayStation 4 и Xbox One:

	все	PlayStation 4	Xbox One
количество образцов	6451.000	495.000	376.000
среднее значение	7.025	6.644	6.469
среднеквадратичное отклонение	1.468	1.475	1.326
минимум	1.400	1.500	1.600
25%	6.200	6.100	5.800
50%	7.400	6.900	6.700
75%	8.100	7.700	7.400
максимум	9.300	9.200	9.200
дисперсия	2.154	2.176	1.758

Как распределяется оценка пользователей для игр последнего поколения на примере PlayStation 4 и Xbox One

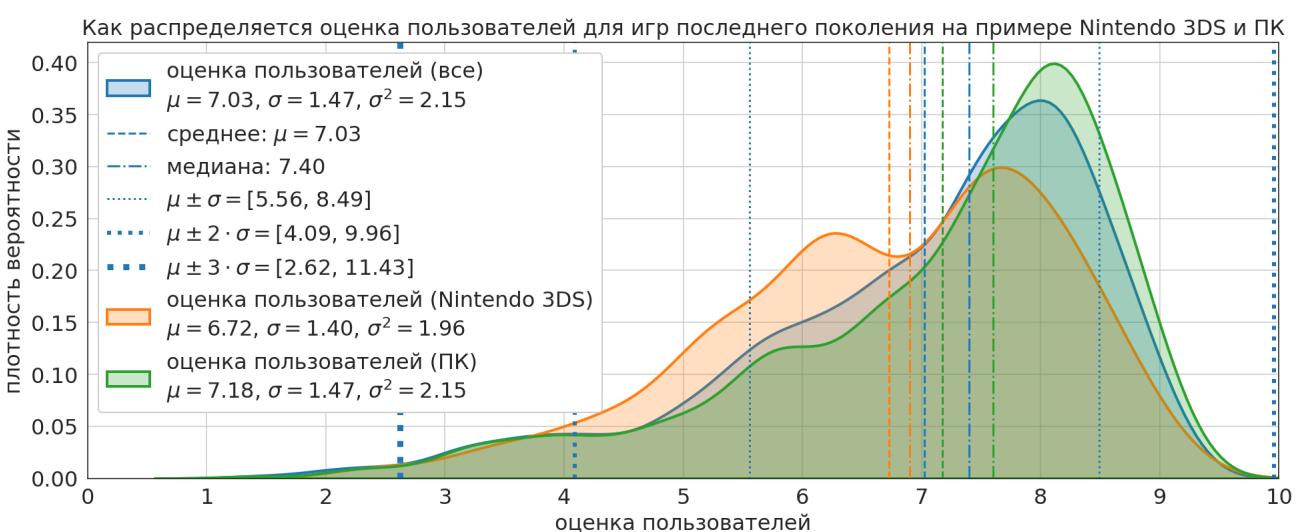


```
In [174]: plot_dist_info(perspective_sales, 'user_score', ['все', 'Nintendo 3DS', 'ПК'],
                     'для игр последнего поколения на примере Nintendo 3DS и ПК', hist=False, fill=True, max_x=10, step=1)
```

Изучим, как распределяется оценка пользователей для игр последнего поколения на примере Nintendo 3DS и ПК:

	все	Nintendo 3DS	ПК
количество образцов	6451.000	750.000	3990.000
среднее значение	7.025	6.725	7.177
среднеквадратичное отклонение	1.468	1.401	1.466
минимум	1.400	2.400	1.400
25%	6.200	5.900	6.400
50%	7.400	6.900	7.600
75%	8.100	7.800	8.200

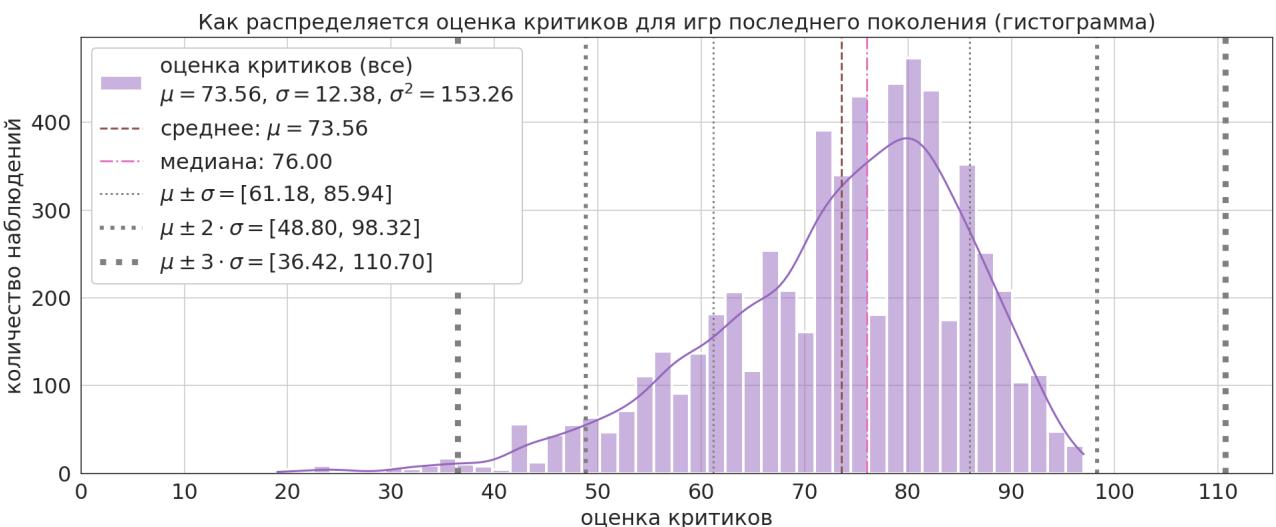
	все	Nintendo 3DS	ПК
максимум	9.300	9.100	9.300
дисперсия	2.154	1.964	2.150



Средние оценки и разброс оценок игр для различных платформ, как и ожидалось, отличаются. Лидирует по средним оценкам пользователей персональный компьютер. Создается впечатление, что на разброс мнений пользователей влияет очень много факторов, пренебрегать которыми нельзя.

Распределение оценок критиков

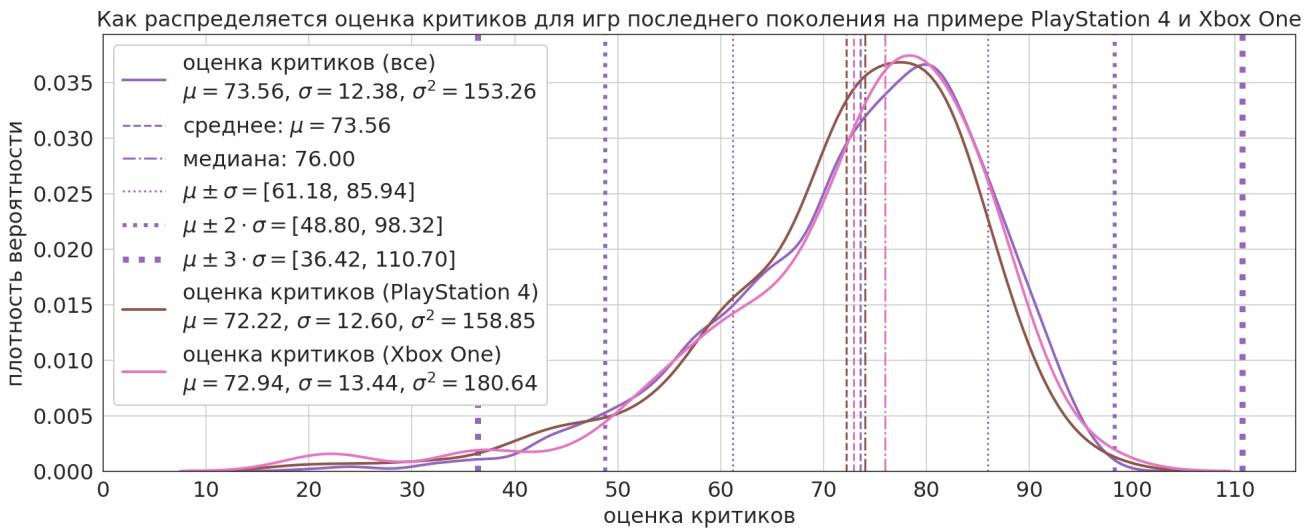
```
In [175...]: plot_dist_info(perspective_sales, 'critic_score', ['все'], 'для игр последнего поколения (гистограмма)', start_color=4, hist=True, fill=True, step=10, show_table=False)
```



```
In [176...]: plot_dist_info(perspective_sales, 'critic_score', ['все', 'PlayStation 4', 'Xbox One'], 'для игр последнего поколения на примере PlayStation 4 и Xbox One', start_color=4, hist=False, fill=False, step=10)
```

Изучим, как распределяется оценка критиков для игр последнего поколения на примере PlayStation 4 и Xbox One:

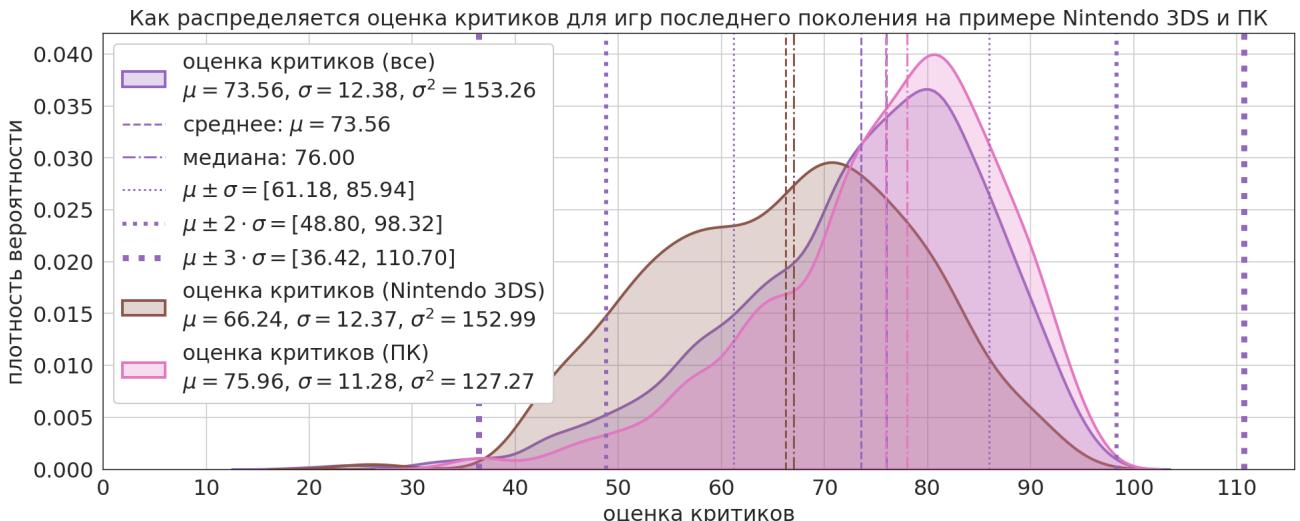
	все	PlayStation 4	Xbox One
количество образцов	6013.000	488.000	349.000
среднее значение	73.564	72.215	72.940
среднеквадратичное отклонение	12.380	12.603	13.440
минимум	19.000	19.000	20.000
25%	66.000	66.000	66.000
50%	76.000	74.000	76.000
75%	82.000	81.000	82.000
максимум	97.000	97.000	97.000
дисперсия	153.265	158.847	180.637



```
In [177]: plot_dist_info(perspective_sales, 'critic_score', ['все', 'Nintendo 3DS', 'ПК'],
                     'для игр последнего поколения на примере Nintendo 3DS и ПК', start_color=4, hist=False, fill=True, step=10)
```

Изучим, как распределяется оценка критиков для игр последнего поколения на примере Nintendo 3DS и ПК:

	все	Nintendo 3DS	ПК
количество образцов	6013.000	743.000	3686.000
среднее значение	73.564	66.242	75.956
среднеквадратичное отклонение	12.380	12.369	11.281
минимум	19.000	26.000	33.000
25%	66.000	57.000	70.000
50%	76.000	67.000	78.000
75%	82.000	75.000	84.000
максимум	97.000	92.000	96.000
дисперсия	153.265	152.987	127.267



А вот критики в среднем на удивление одинаково оценивают игры для PlayStation 4, Xbox One и персональных компьютеров. На их фоне Nintendo 3DS получает заметно меньше положительных отзывов.

Изменение оценок во времени

```
In [178]: plot_dist_info(perspective_sales, 'user_score', ['2011 год', '2015 год'],
                     'для игр последнего поколения за 2011 и 2015 год', hist=False, fill=True, max_x=10, step=1)
```

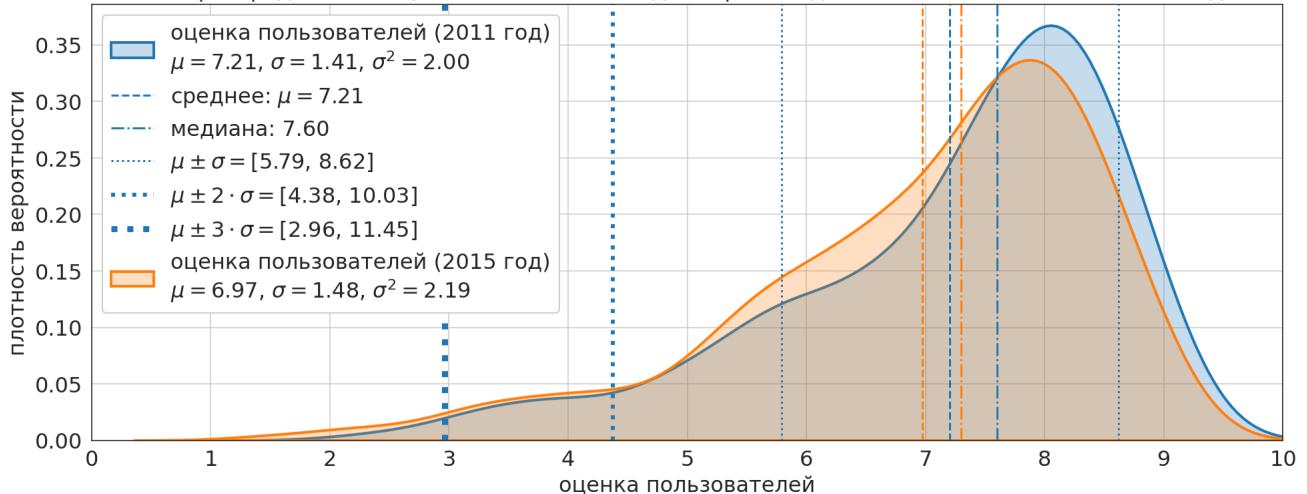
Изучим, как распределяется оценка пользователей для игр последнего поколения за 2011 и 2015 год:

	2011 год	2015 год
количество образцов	628.000	1379.000
среднее значение	7.205	6.973
среднеквадратичное отклонение	1.415	1.478
минимум	2.200	1.400
25%	6.400	6.200
50%	7.600	7.300
75%	8.200	8.100
максимум	9.300	9.300

2011 год 2015 год

дисперсия 2.001 2.185

Как распределяется оценка пользователей для игр последнего поколения за 2011 и 2015 год



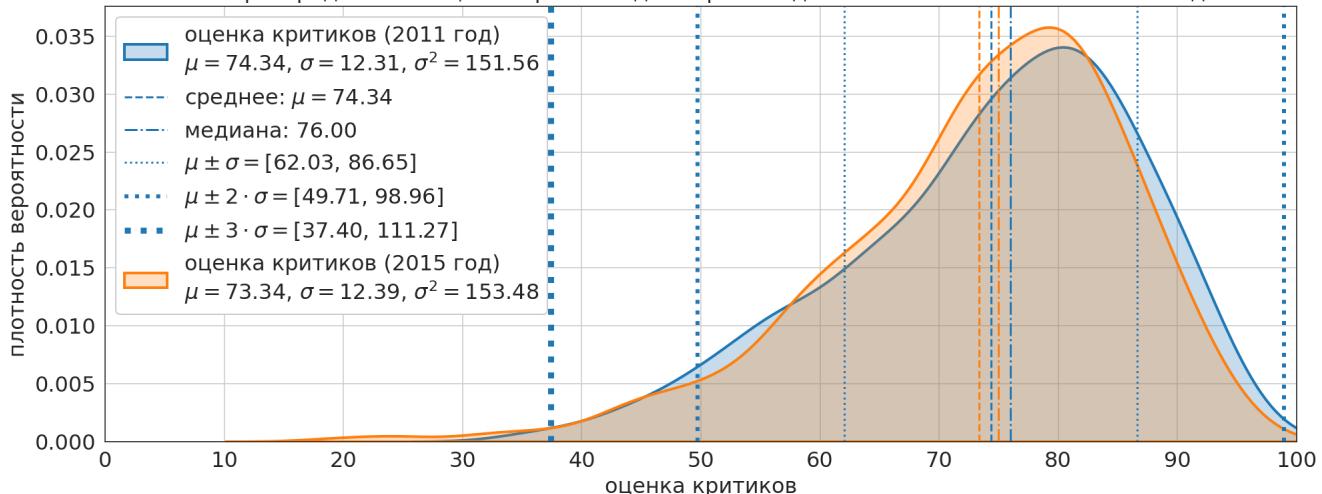
В 2015 году пользовательские оценки игр стали ниже, чем в 2011 году. Рынок ждет нашего выхода с новой прорывной игрой.

```
In [179]: plot_dist_info(perspective_sales, 'critic_score', ['2011 год', '2015 год'],
'для игр последнего поколения за 2011 и 2015 год', hist=False, fill=True, max_x=100, step=10)
```

Изучим, как распределяется оценка критиков для игр последнего поколения за 2011 и 2015 год:

	2011 год	2015 год
количество образцов	585.000	1281.000
среднее значение	74.337	73.343
среднеквадратичное отклонение	12.311	12.389
минимум	36.000	19.000
25%	66.000	66.000
50%	76.000	75.000
75%	83.000	82.000
максимум	96.000	97.000
дисперсия	151.563	153.483

Как распределяется оценка критиков для игр последнего поколения за 2011 и 2015 год



Критики солидарны с пользователями - игры в 2015 году в среднем хуже, чем в 2011 году на 1 балл.

Распределение продаж за актуальный период

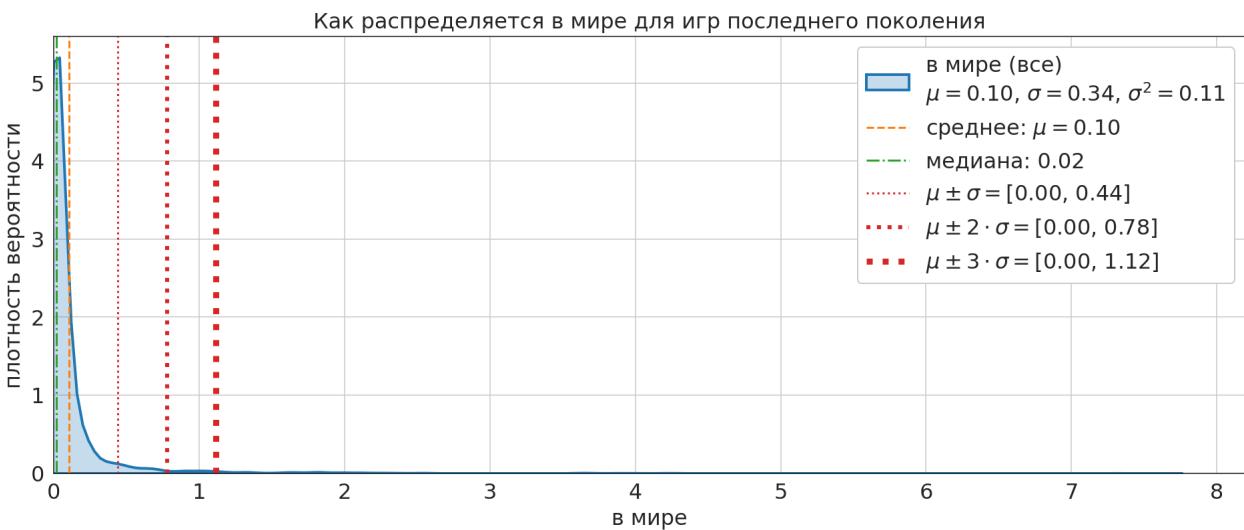
Гистограмма продаж

```
In [180]: plot_dist_info(perspective_sales, 'total_sales', ['все'],
'для игр последнего поколения', hist=False, fill=True)
```

Изучим, как распределяется в мире для игр последнего поколения:

все
количество образцов 10067.000
среднее значение 0.104
среднеквадратичное отклонение 0.337
минимум 0.001

всё	
25%	0.005
50%	0.020
75%	0.070
максимум	7.600
дисперсия	0.114



В распределении случайной величины количества продаж одной игры в год обращают на себя внимание небольшая дисперсия и наличие ярких выбросов в лице игр которые на фоне остальных, собравшихся ближе к средним значениям, показывают феноменальные результаты продаж. Пока 25% программ не достигают рубежа в 5 тыс. копий, а 75% мечтают о 70 тыс. в год, некоторые устанавливаются в каждый "утюг". Например:

```
In [181...]: table_games[table_games.generation==table_games.generation.max()] \
    .sort_values('total_sales_annual', ascending=False) \
    [[['name', 'platform', 'year_of_release', 'total_sales_annual']].head(10)
```

```
Out[181...]:
```

	name	platform	year_of_release	total_sales_annual
94	FIFA 17	PS4	2016	7.600
31	Call of Duty: Black Ops 3	PS4	2015	7.315
108	Pokemon Sun/Moon	3DS	2016	7.140
171	Uncharted 4: A Thief's End	PS4	2016	5.390
245	Call of Duty: Infinite Warfare	PS4	2016	4.470
77	FIFA 16	PS4	2015	4.290
42	Grand Theft Auto V	PS4	2014	4.207
289	Battlefield 1	PS4	2016	4.070
87	Star Wars Battlefront (2015)	PS4	2015	3.990
47	Pokemon Omega Ruby/Pokemon Alpha Sapphire	3DS	2014	3.893

"Ящик с усами" по глобальным продажам игр в разбивке по платформам

Опишем функцию построения "ящика с усами":

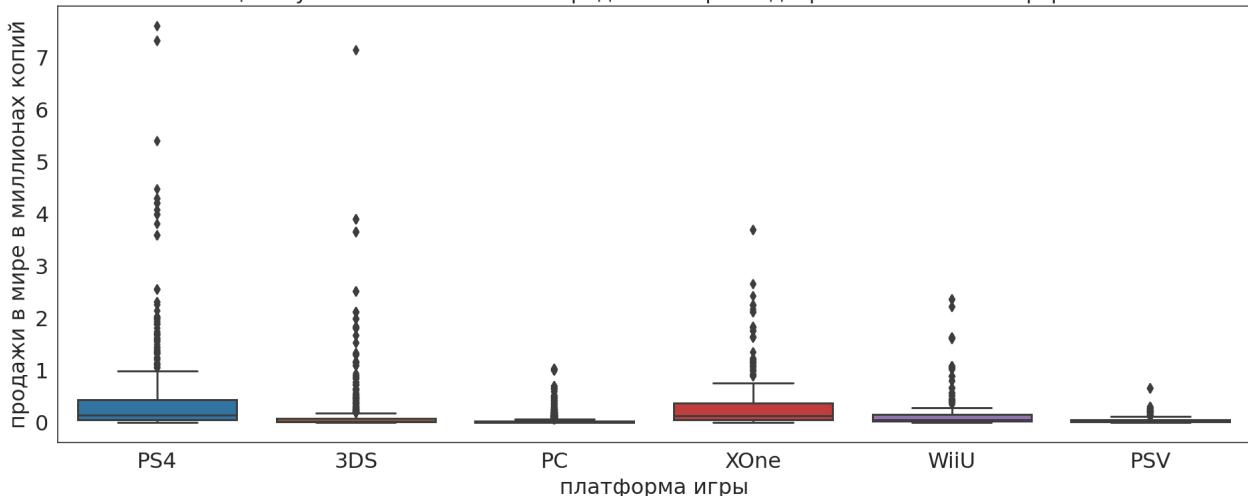
```
In [182...]: def plot_sales_box(hue=None, *args, **kwagrs):
    """Строит ящик с усами по продажам"""
    if hue is not None:
        hue = translate_field_name(hue)

    tmp = translate_data_frame(perspective_sales, push=True)
    try:
        sns.boxplot(data=tmp, x=translate_field_name('platform'), y=translate_field_name('total_sales'),
                    hue=hue, order=tmp[translate_field_name('platform')].unique(), *args, **kwagrs)
    finally:
        pop_old_names(perspective_sales)
    plt.title("Ящик с усами" по глобальным продажам игр в год с разбивкой по платформам")
    plt.show()
```

Построим "ящик с усами" по продажам игр для платформ последнего поколения:

```
In [183...]: plot_sales_box()
```

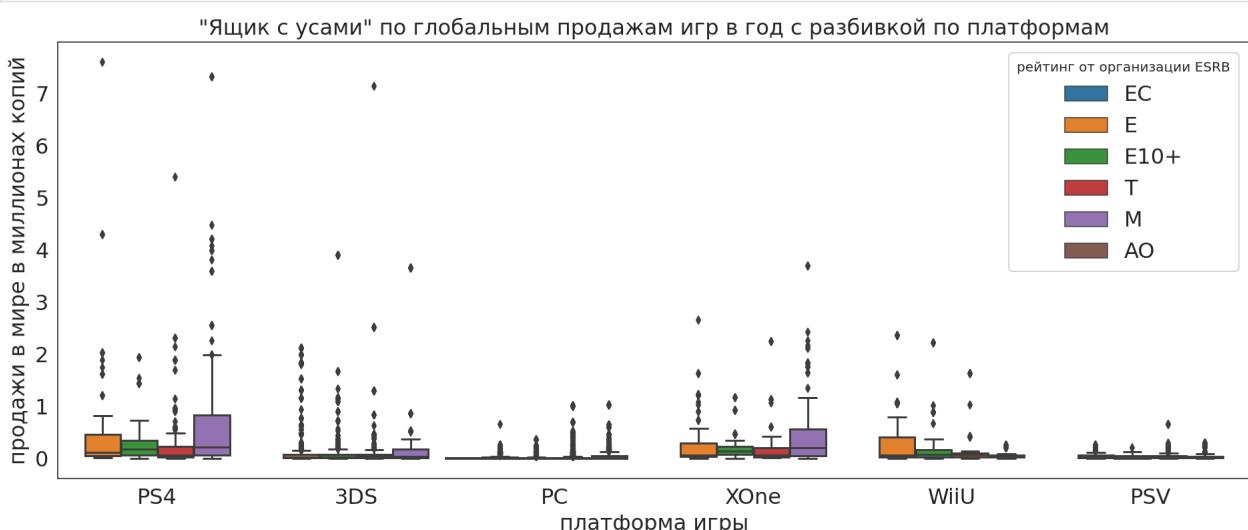
"Ящик с усами" по глобальным продажам игр в год с разбивкой по платформам



Обратим внимание, что разброс значений продаж игры в год (дисперсия) невысокий, но для каждой платформы имеются игры-хиты, которые демонстрировали аномально высокие продажи по сравнению с конкурентами. Наибольшую выручку приносят игры для **PlayStation 4**.

Построим "ящик с усами" по продажам игр для платформ последнего поколения с разбивкой еще и по возрастному рейтингу:

In [184...]: `plot_sales_box(hue='rating', width=1.1, fliersize=4)`



Теперь видим, что в структуре продаж платформ **PS4**, **3DS** и **XOne** преобладают игры с возрастным ограничением "**для взрослых (M)**", а для платформы **WiiU** - игры "**для всех (E)**".

Анализ зависимости продаж от оценок

Опишем функции для вычисления корреляций и построения графиков:

In [185...]:

```
corr_fields = ['user_score', 'critic_score', 'total_sales', 'na_sales', 'eu_sales', 'jp_sales', 'other_sales']
corr_fields_desc = translate_field_names(corr_fields, short=True)
def display_corr_for_platform(df, slice_name):
    """Выводит таблицу корреляции оценок игр и объемов их продаж"""
    data = df[filters[slice_name]]
    display(HTML(f'{<br>}<b>Таблицы корреляций оценок игр и объемов их продаж ({slice_name})</b>'))
    display(translate_data_frame(data[corr_fields], short=True).corr().iloc[0:2, 2:])
```

In [186...]:

```
def draw_corr_for_platform(df, slice_name):
    """Рисует диаграмму рассеяния оценок игр и объемов их продаж"""
    display(HTML(f'{<br>}<b>Диаграмма рассеяния оценок игр и объемов их продаж ({slice_name})</b>'))
    data = df[filters[slice_name]]
    sns.pairplot(translate_data_frame(data[corr_fields], short=True), diag_kind='None',
                 kind='reg', plot_kws={'line_kws': {'color': 'red'}, 'scatter_kws': {'s': 5, 'alpha': 0.7}},
                 y_vars=corr_fields_desc[0:2], x_vars=corr_fields_desc[2:]);
```

In [187...]:

```
def plot_score_scatter(field_name, title):
    """Строит укрупненную диаграмму рассеяния оценок (field_name) и продаж"""
    tmp = perspective_sales[[field_name, 'total_sales', 'rating_rus']].dropna()
    sns.scatterplot(data=translate_data_frame(tmp), x=translate_field_name(field_name),
                    y=translate_field_name('total_sales'),
                    hue=translate_field_name('rating_rus'), alpha=0.65);
    plt.title(title); plt.show()
```

In [188...]:

```
def corr_power(x):
    """Возвращает описание степени корреляции по её числовому значению"""
    if x >= 0.9: return 'очень высокая'
    elif x >= 0.7: return 'высокая'
    elif x >= 0.5: return 'средняя'
    elif x >= 0.3: return 'слабая'
    else: return 'очень слабая'
```

In [189...]:

```
def normalize(df):
    """Масштабирует значения набора данных в шкалу от 0 до 1"""
```

```
return (df - df.min()) / (df.max() - df.min())
```

Посчитаем корреляции для всех платформ последнего поколения:

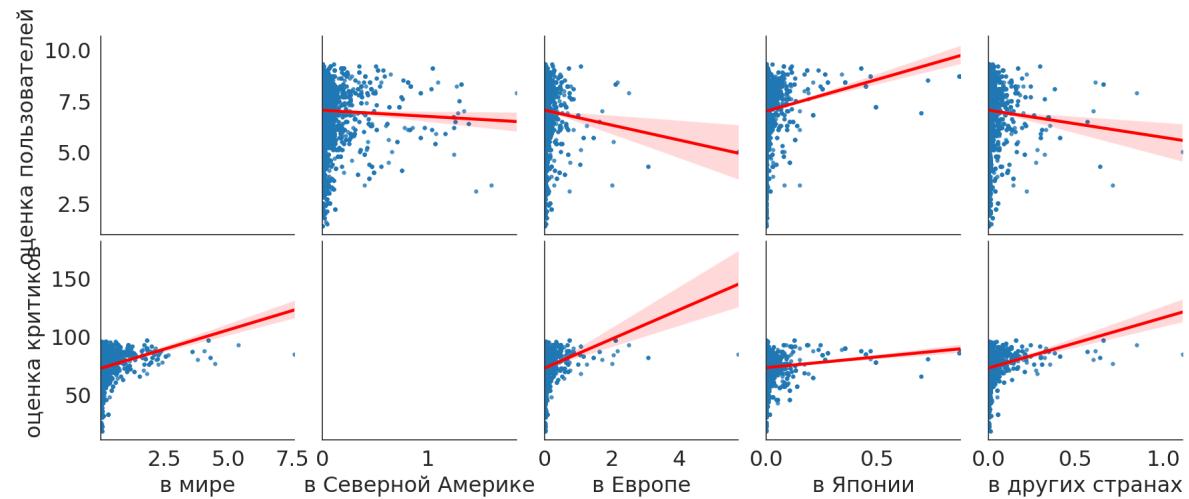
```
In [190... display_corr_for_platform(perspective_sales, 'все')
```

Таблицы корреляций оценок игр и объемов их продаж (все):

	в мире	в Северной Америке	в Европе	в Японии	в других странах
оценка пользователей	-0.020	-0.027	-0.040	0.105	-0.040
оценка критиков	0.185	0.177	0.168	0.076	0.160

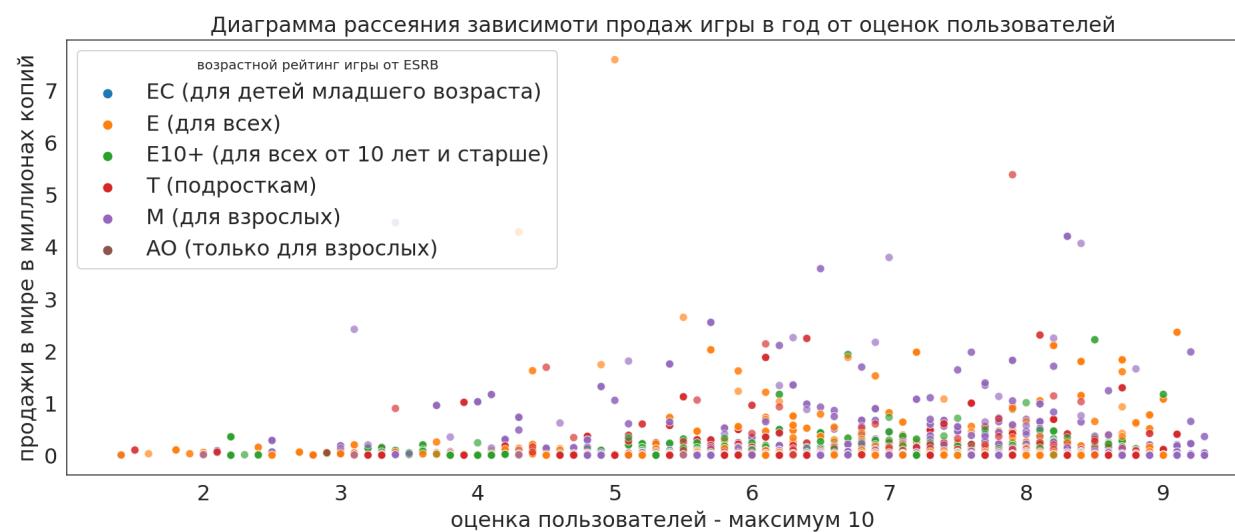
```
In [191... draw_corr_for_platform(perspective_sales, 'все')
```

Диаграмма рассеяния оценок игр и объемов их продаж (все):

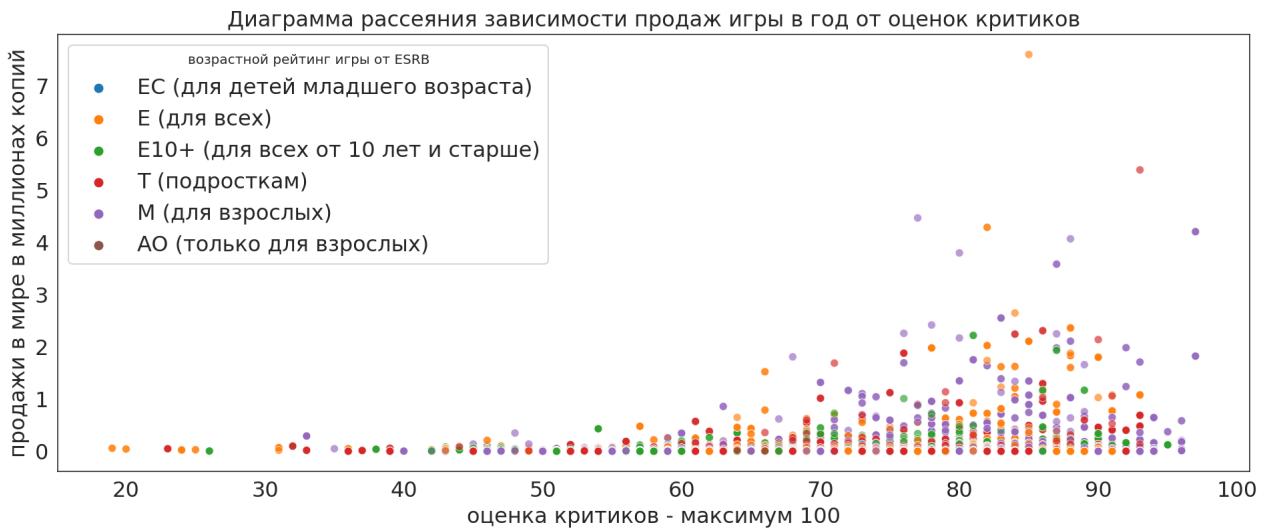


Связь оценок с общими продажами игр для всех платформ последнего поколения не прослеживается, хотя критики лучше "угадывают" тенденции - корреляция их оценок с продажами в выше, чем у пользователей. Отрицательные пользовательские корреляции говорят, скорее, о том, что потребители активнее выражают именно негативное мнение, а позитивное держат при себе. Но это требует подтверждения через проверку методики сбора пользовательских отзывов.

```
In [192... plot_score_scatter('user_score', 'Диаграмма рассеяния зависимости продаж игры в год от оценок пользователей')
```



```
In [193... plot_score_scatter('critic_score', 'Диаграмма рассеяния зависимости продаж игры в год от оценок критиков')
```



Посмотрим, как влияют на продажи внутри одной популярной платформы отзывы пользователей и критиков. Построим диаграмму рассеяния и посчитаем корреляцию между отзывами и продажами.

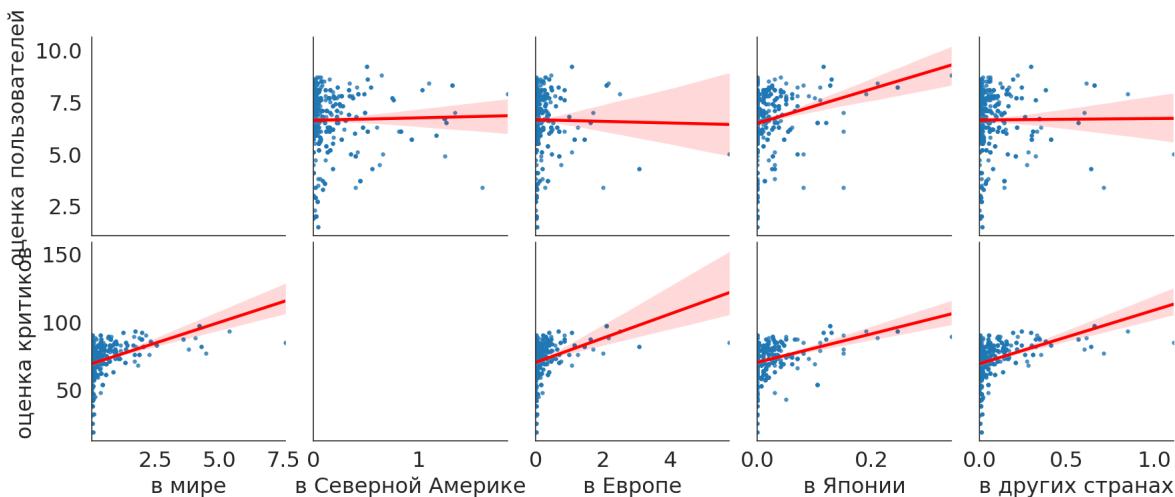
In [194...]: `display_corr_for_platform(perspective_sales, 'PlayStation 4')`

Таблицы корреляций оценок игр и объемов их продаж (PlayStation 4):

	в мире	в Северной Америке	в Европе	в Японии	в других странах
оценка пользователей	0.010	0.022	-0.012	0.202	0.006
оценка критиков	0.403	0.425	0.330	0.303	0.407

In [195...]: `draw_corr_for_platform(perspective_sales, 'PlayStation 4')`

Диаграмма рассеяния оценок игр и объемов их продаж (PlayStation 4):



In [196...]: `display_corr_for_platform(perspective_sales, 'Nintendo 3DS')`

Таблицы корреляций оценок игр и объемов их продаж (Nintendo 3DS):

	в мире	в Северной Америке	в Европе	в Японии	в других странах
оценка пользователей	0.283	0.268	0.221	0.307	0.247
оценка критиков	0.407	0.405	0.366	0.370	0.388

In [197...]: `display_corr_for_platform(perspective_sales, 'Xbox One')`

Таблицы корреляций оценок игр и объемов их продаж (Xbox One):

	в мире	в Северной Америке	в Европе	в Японии	в других странах
оценка пользователей	-0.045	-0.047	-0.032	0.091	-0.047
оценка критиков	0.440	0.416	0.370	0.222	0.438

In [198...]: `display_corr_for_platform(perspective_sales, 'ПК')`

Таблицы корреляций оценок игр и объемов их продаж (ПК):

	в мире	в Северной Америке	в Европе	в Японии	в других странах
оценка пользователей	-0.092	-0.063	-0.092	0.071	-0.087

оценка критиков	0.238	0.199	0.218	0.100	0.211
------------------------	-------	-------	-------	-------	-------

Пользовательские оценки, увы, мало связаны с количеством продаж. Скорее всего, пользователи сначала покупают игру, а потом её оценивают, и палитра эмоций пестрит различными мнениями от восторга до полного неприятия.

Критики прогнозируют продажи точнее для всех платформ. Хуже всего экспертные оценки коррелируют с продажами для персональных компьютеров, возможно, из-за широкой номенклатуры технических спецификаций этих устройств. Оценки критиков слабее коррелируют с продажами игр в Японии. Видимо, западное лекало не совсем учитывает менталитет людей Страны восходящего солнца.

По анализу корреляции в нашем проекте получилось, что сила связи между оценками пользователей и объемами продаж игры в год очень слабая, сила критиков иногда претендует на слабую. Таким образом, для прогнозирования продаж игрового программного обеспечения недостаточно использовать оценки в том виде, как они были нам предоставлены. Пользовательский рейтинг в нашем массиве "заморожен", т.е. имеет одно значение на весь период продажи игры, а он растягивается на несколько лет, за которые могут меняться и оценки, реагируя на многие факторы, в том числе появление игр-конкурентов. Для более точного предсказывания продаж по оценкам необходимо собирать их чаще и сопоставлять с продажами за более короткие периоды времени, в идеале около месяца, как это делается в рейтинговых агентствах типа [VGChartz](#).

Нужны ли оценки?

Кстати, наш конкурент, аналитическое агентство с иностранным капиталом "Soy, Det & Tuck", успело разместить в научном журнале статью с выводом об бесполезности учета оценок критиков и пользователей при планировании инвестиций из-за низкой корреляции оценок игры с её среднегодовыми продажами. Статья получила множество стикеров в Slack-сообществе начинающих аналитиков данных, их с энтузиазмом поддержали программисты-стартаперы, бросившиеся переделывать свои курсовые по машинной графике в игру-бродилку.

Не будем торопиться с выводами. Поставим себя на место руководства крупной компании с большим штатом программистов разного уровня подготовки, которые способны наводнить рынок программами разного уровня качества. Основная задача - максимизировать общую прибыль. Как руководству решить эту задачу оптимизации использования ресурсов и выбрать правильное стратегическое направление развития? Изучим корреляции оценок игр и общих продаж игр с этими оценками. Руководство компании мало интересует судьба одной программы, важнее общий успех. Дадим ему информацию для обоснования своего решения:

```
In [199...]: def get_total_sales_parts(field_name):
    """Определяет общие продажи с группировкой по оценкам, выводит их доли и печатает корреляции по регионам"""
    tmp = perspective_sales.groupby(field_name) \
        [[['total_sales', 'na_sales', 'eu_sales', 'jp_sales', 'other_sales']].sum()
    tmp = translate_data_frame(tmp / tmp.sum() * 100, short=True)
    tmp.plot(grid=True, lw=2, alpha=0.7)
    plt.tick_params(axis='both', which='major', labelleft=True, labelright = True, labelbottom = True, labeltop=False)
    plt.ylabel('доля в процентах')
    plt.title(f'Как связана {tmp.index.name} с долей игр в общем объеме продаж за актуальный период (с {min_year} года)')
    plt.show()
    print(f'Корреляция с общими объемами продаж игр за актуальный период с {min_year}:')
    for index, value in tmp.corrwith(pd.Series(tmp.index)).iteritems():
        print(f'{index}>20s}: {value:.03f} ({corr_power(value)})')
    return tmp
```

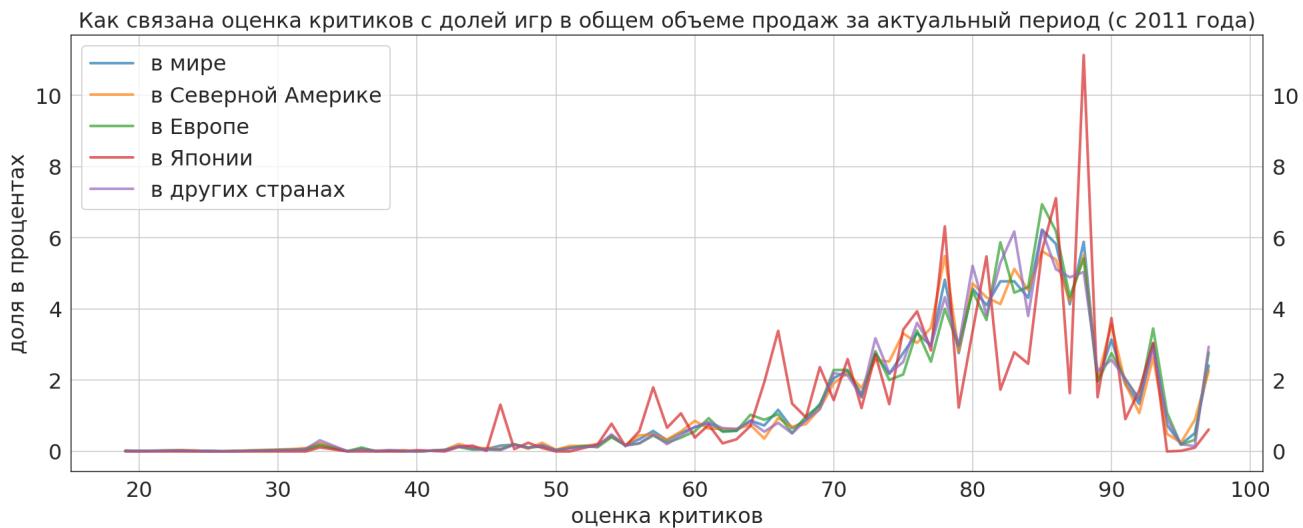
```
In [200...]: sales_for_users = get_total_sales_parts('user_score')
```



Корреляция с общими объемами продаж игр за актуальный период с 2011:

в мире: 0.801 (высокая)
в Северной Америке: 0.890 (высокая)
в Европе: 0.605 (средняя)
в Японии: 0.795 (высокая)
в других странах: 0.611 (средняя)

```
In [201...]: sales_for_critic = get_total_sales_parts('critic_score')
```



Корреляция с общими объемами продаж игр за актуальный период с 2011:

в мире: 0.766 (высокая)
в Северной Америке: 0.766 (высокая)
в Европе: 0.727 (высокая)
в Японии: 0.654 (средняя)
в других странах: 0.720 (высокая)

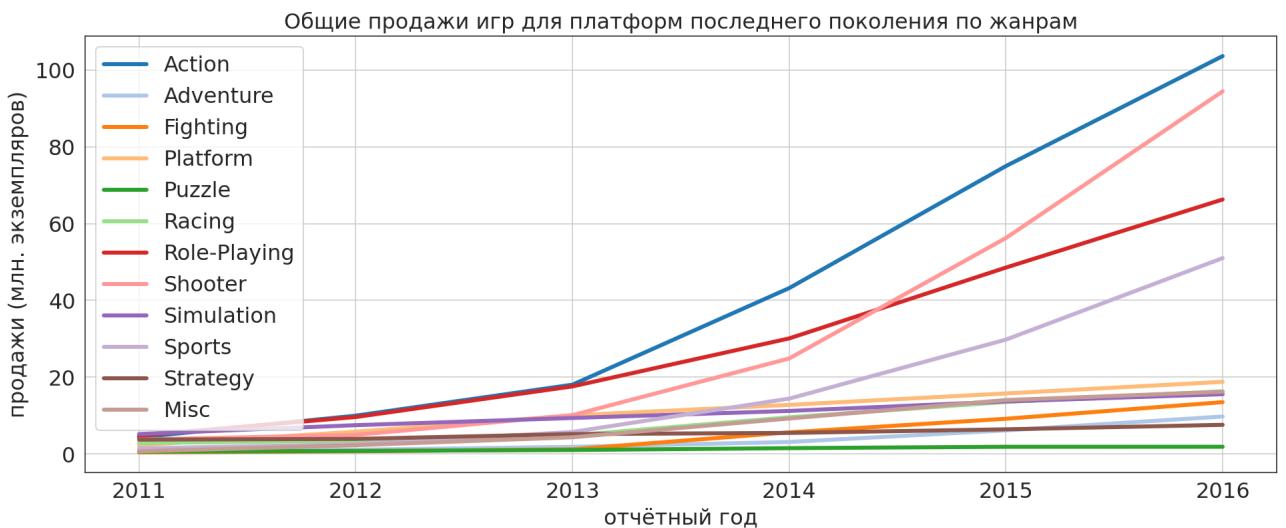
Получили в основном высокие степени корреляции оценок пользователей и критиков с общим объемом продаж. Особое место опять заняла Япония (наши оценки, скорее всего, учитывают мнение пользователей за пределами Страны восходящего солнца). Таким образом, **оценка отдельно взятой игры не влияет на её продажи (большой вклад вносят и другие факторы).** А вот общие продажи списка игр, которые выпущены, например, одной компанией, уже будут более чувствительны к реакции пользователей и критиков (в дело вступает центральная предельная теорема).

И вот теперь становится понятно, что для крупных компаний из игровой индустрии есть пути получения гарантированно высокой прибыли. Например, оставить в разработке 3-4 игры из сегментов (по жанру, платформе, возрастному рейтингу) с умеренными оценками (70-80 от критиков) и 1-2 из сегмента рискованных, но способных выстрелить большими продажами (с оценками 80-87). Возможны и другие варианты, которые целесообразно рассчитывать уже с помощью машинного обучения, принимая во внимание другие факторы.

Отличия в продажах игр различных жанров

На вопросы **задания** "Что можно сказать о самых прибыльных жанрах? Выделяются ли жанры с высокими и низкими продажами?" однозначно можно ответить только на второй. Оценить прибыль от продажи игр мы не имеем возможности, так как не знаем затраты на их разработку. Косвенным показателем прибыльности могут выступить объемы продаж по жанрам. Изучим их для платформ последнего поколения, построив графики:

```
In [202...]:  
tmp = perspective_sales.pivot_table(index='year', columns='genre', values='total_sales', aggfunc='sum')  
tmp = translate_data_frame(tmp)  
tmp.plot(lw=3, legend=True, grid=True, color=fill_colors,  
         title='Общие продажи игр для платформ последнего поколения по жанрам')  
plt.ylabel('продажи (млн. экземпляров)')  
plt.show()
```



В последнее время, что особенно актуально для планирования деятельности на ближайшие пару лет, лидируют по количеству продаж четыре явных фаворита игровой индустрии:

1. Action
2. Shooter
3. Role-Playing
4. Sports

Самый малоподаваемый жанр - **Puzzle**.

Производители учитывают этот факт, о чем свидетельствуют тепловые карты, построенные [выше](#).

Вывод по шагу 3

Шаг 3 выполнен:

- Изучено, сколько игр выпускалось в разные годы. Оценена важность данных за все периоды.
- Проанализирована динамика изменения продаж по платформам. Построены графики распределения суммарных продаж по годам. Определены характерные сроки появления новых платформ.
- Обоснован актуальный период времени для исследования данных в интересах построения прогноза на 2017 год.
- Выбраны несколько потенциально прибыльных платформ.
- Построен график «ящик с усами» по глобальным продажам игр в разбивке по платформам. Описан результат.
- Изучено, как влияют на продажи внутри одной популярной платформы отзывы пользователей и критиков. Построена диаграмма рассеяния и посчитаны корреляции между отзывами и продажами. Выводы соотнесены с продажами игр на других платформах.
- Проанализировано общее распределение игр по жанрам. Выделены жанры с высокими и низкими продажами.
- Заключения по вопросам задания подготовлены к включению в общий вывод проекта.

Шаг 4. Составьте портрет пользователя каждого региона

Задание описано выше

Изучим показатели рынка игрового программного обеспечения в регионах мира. Напомню, что ранее мы определили тренды продаж для платформ различных поколений. Перспективы есть только у последнего (восьмого), именно только его и будем учитывать в дальнейшем.

Инструменты анализа

```
In [203...]: def show_sales_parts(data, index, columns, values, title, norm=True, loc='best', color=None):
    """Выводит долю значений values в общем объеме с группировкой по
    полям index (строки) и columns (столбцы) набора данных data,
    а также строит график зависимости долей values от index"""
    tmp=pd.crosstab(index=data[index], columns=data[columns], values=data[values],
                     aggfunc='sum', normalize='index').mul(100)
    tmp=translate_data_frame(tmp)
    display(HTML(f'{title} в процентах'))
    display(tmp)
    display(HTML(f'Рейтинг в {tmp.index.max()} году:'))
    display(HTML(', '.join([
        f'{i+1} место - {v}' for i, v in enumerate(tmp.iloc[-1].sort_values(ascending=False).index)])))
    tmp.plot(grid=True, lw=2.5)
    plt.tick_params(axis='both', which='major', labelleft=True, labelright = True, labelbottom = True, labeltop=False)
    plt.xticks(ticks=tmp.index.to_list())
    plt.ylabel('доля в процентах')
    plt.yticks(rotation=0)
    plt.legend(ncol=3, loc=loc)
    plt.title(title)
    plt.show();
```

Самые популярные платформы по регионам

```
In [204...]: show_sales_parts(perspective_sales, 'year', 'platform', 'na_sales',
                      'Доли платформ в общей структуре продаж игр в Северной Америке')
```

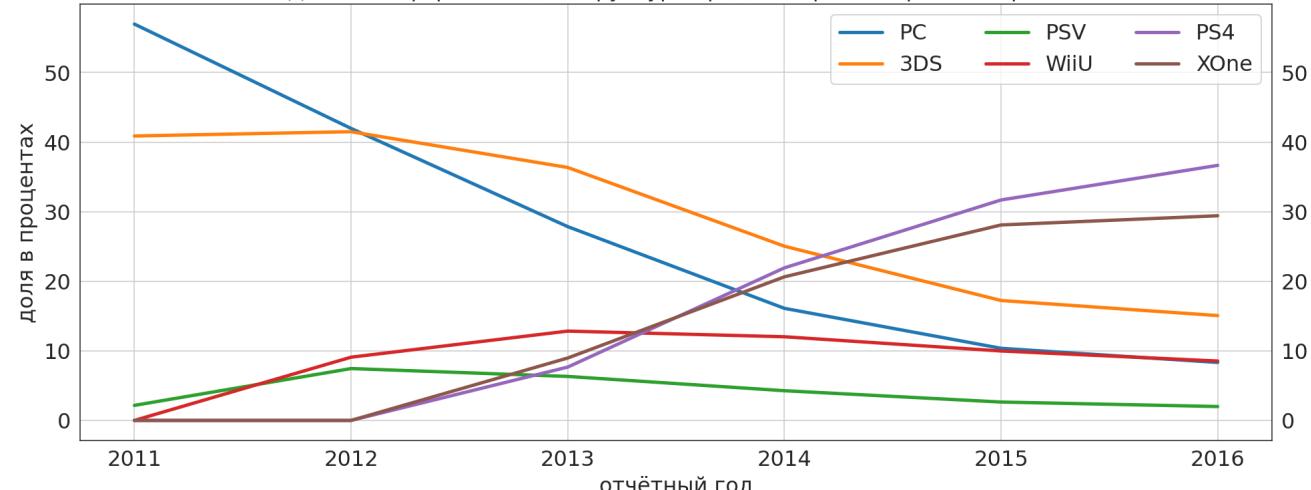
Доли платформ в общей структуре продаж игр в Северной Америке в процентах

	PC	3DS	PSV	WiiU	PS4	XOne
отчётный год						
2011	56.960	40.879	2.161	0.000	0.000	0.000
2012	41.963	41.495	7.454	9.088	0.000	0.000
2013	27.852	36.354	6.322	12.837	7.667	8.968
2014	16.108	25.037	4.269	12.023	21.926	20.637
2015	10.366	17.235	2.642	9.981	31.683	28.093
2016	8.333	15.067	1.997	8.533	36.656	29.414

Рейтинг в 2016 году:

1 место - PS4, 2 место - XOne, 3 место - 3DS, 4 место - WiiU, 5 место - PC, 6 место - PSV

Доли платформ в общей структуре продаж игр в Северной Америке



```
In [205...]: show_sales_parts(perspective_sales, 'year', 'platform', 'eu_sales',
                      'Доли платформ в общей структуре продаж игр в Европе')
```

Доли платформ в общей структуре продаж игр в Европе в процентах

PC 3DS PSV WiiU PS4 XOne

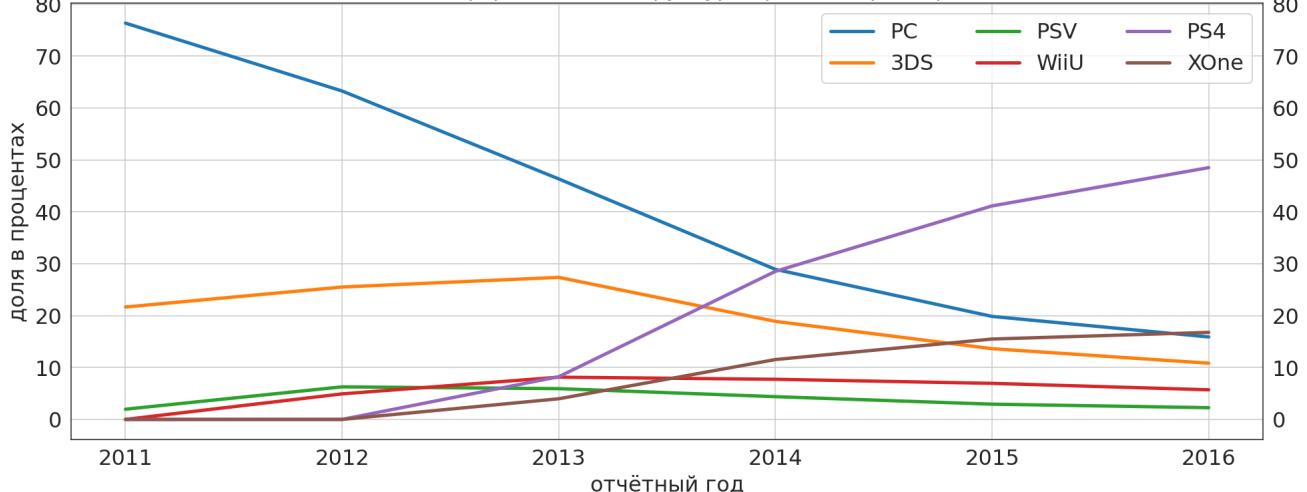
отчётный год

	PC	3DS	PSV	WiiU	PS4	XOne
2011	76.374	21.660	1.966	0.000	0.000	0.000
2012	63.290	25.514	6.276	4.920	0.000	0.000
2013	46.346	27.376	5.929	8.139	8.233	3.978
2014	28.915	18.897	4.384	7.735	28.531	11.537
2015	19.850	13.632	2.937	6.949	41.142	15.490
2016	15.893	10.836	2.263	5.711	48.521	16.776

Рейтинг в 2016 году:

1 место - **PS4**, 2 место - **XOne**, 3 место - **PC**, 4 место - **3DS**, 5 место - **WiiU**, 6 место - **PSV**

Доли платформ в общей структуре продаж игр в Европе



```
In [206]: show_sales_parts(perspective_sales, 'year', 'platform', 'jp_sales',
'Dоли платформ в общей структуре продаж игр в Японии')
```

Доли платформ в общей структуре продаж игр в Японии в процентах

PC 3DS PSV WiiU PS4 XOne

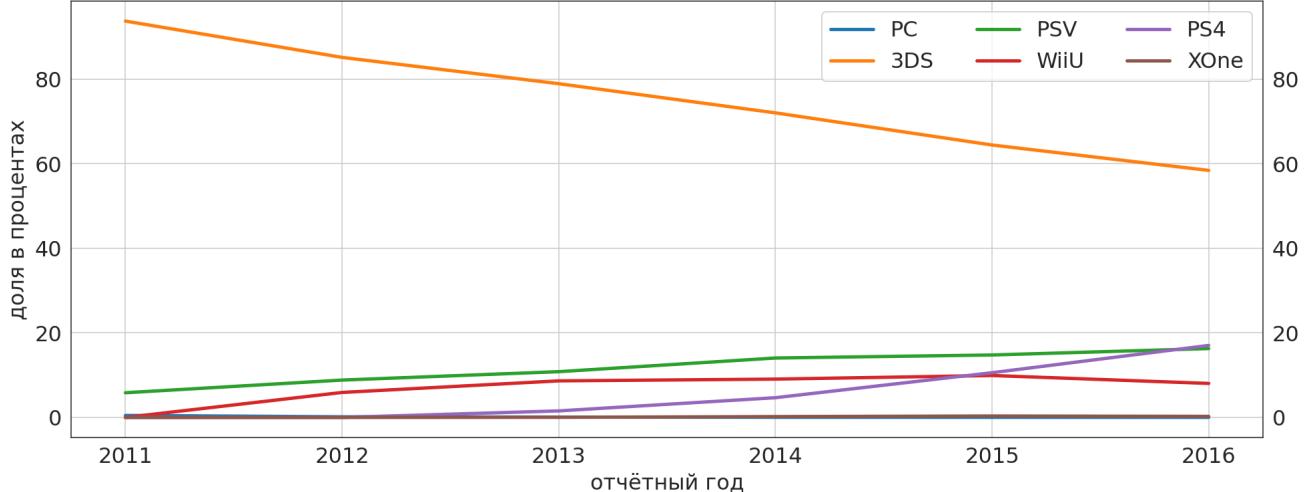
отчётный год

	PC	3DS	PSV	WiiU	PS4	XOne
2011	0.466	93.700	5.833	0.000	0.000	0.000
2012	0.148	85.120	8.830	5.902	0.000	0.000
2013	0.070	78.899	10.815	8.643	1.540	0.033
2014	0.041	72.001	14.045	9.058	4.659	0.196
2015	0.026	64.405	14.762	9.909	10.568	0.330
2016	0.018	58.402	16.278	8.034	17.021	0.247

Рейтинг в 2016 году:

1 место - **3DS**, 2 место - **PS4**, 3 место - **PSV**, 4 место - **WiiU**, 5 место - **XOne**, 6 место - **PC**

Доли платформ в общей структуре продаж игр в Японии



```
In [207]: show_sales_parts(perspective_sales, 'year', 'platform', 'other_sales',
'Dоли платформ в общей структуре продаж игр за пределами Америки, Европы и Японии')
```

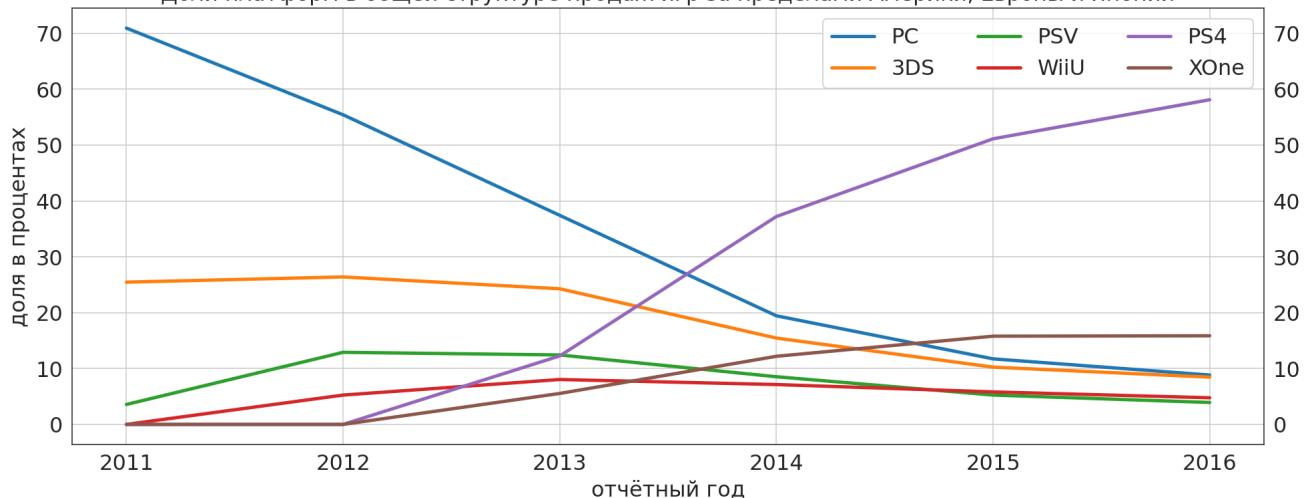
Доли платформ в общей структуре продаж игр за пределами Америки, Европы и Японии в процентах

	PC	3DS	PSV	WiiU	PS4	XOne
отчётный год						
2011	70.954	25.472	3.574	0.000	0.000	0.000
2012	55.434	26.404	12.907	5.254	0.000	0.000
2013	37.430	24.285	12.438	8.041	12.261	5.544
2014	19.447	15.458	8.536	7.139	37.222	12.198
2015	11.738	10.259	5.253	5.820	51.131	15.798
2016	8.841	8.462	3.932	4.771	58.125	15.869

Рейтинг в 2016 году:

1 место - PS4, 2 место - XOne, 3 место - PC, 4 место - 3DS, 5 место - WiiU, 6 место - Psv

Доли платформ в общей структуре продаж игр за пределами Америки, Европы и Японии



Популярность платформ с разных регионах отличается: PS4 - фаворит везде кроме Японии, там предпочитают 3DS. Во всём мире наблюдается устойчивый тренд на увеличение популярности PS4.

Самые популярные жанры по регионам

```
In [208]: show_sales_parts(perspective_sales, 'year', 'genre', 'na_sales',
                      'доля жанров в общей структуре продаж игр в Северной Америке', color=fill_colors)
```

Доли жанров в общей структуре продаж игр в Северной Америке в процентах

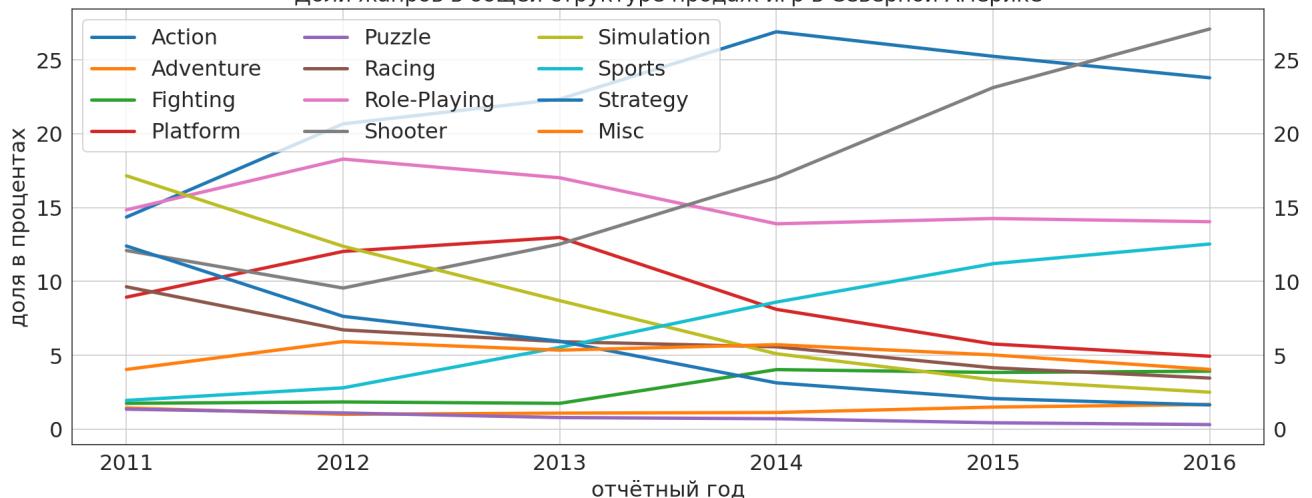
	Action	Adventure	Fighting	Platform	...	Simulation	Sports	Strategy	Misc
отчётный год									
2011	14.362	1.440	1.752	8.939	...	17.159	1.951	12.405	4.046
2012	20.670	1.009	1.847	12.037	...	12.386	2.802	7.642	5.933
2013	22.346	1.090	1.752	12.980	...	8.706	5.542	5.957	5.353
2014	26.906	1.134	4.037	8.115	...	5.109	8.606	3.139	5.724
2015	25.244	1.499	3.845	5.770	...	3.343	11.206	2.074	5.031
2016	23.789	1.678	3.927	4.944	...	2.507	12.538	1.647	4.050

6 rows x 12 columns

Рейтинг в 2016 году:

1 место - Shooter, 2 место - Action, 3 место - Role-Playing, 4 место - Sports, 5 место - Platform, 6 место - Misc, 7 место - Fighting, 8 место - Racing, 9 место - Simulation, 10 место - Adventure, 11 место - Strategy, 12 место - Puzzle

Доли жанров в общей структуре продаж игр в Северной Америке



```
In [209...]: show_sales_parts(perspective_sales, 'year', 'genre', 'eu_sales',
    'Доли жанров в общей структуре продаж игр в Европе', color=fill_colors)
```

Доли жанров в общей структуре продаж игр в Европе в процентах

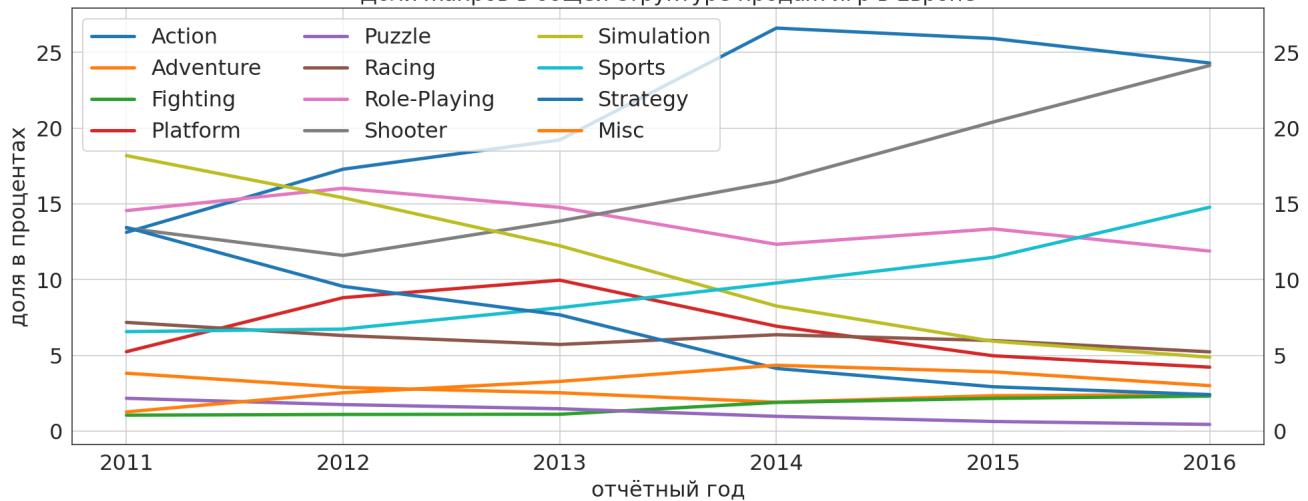
	Action	Adventure	Fighting	Platform	...	Simulation	Sports	Strategy	Misc
отчётный год									
2011	13.120	3.824	1.053	5.245	...	18.182	6.567	13.443	1.269
2012	17.285	2.890	1.103	8.807	...	15.403	6.737	9.555	2.535
2013	19.214	2.534	1.115	9.962	...	12.236	8.146	7.676	3.283
2014	26.597	1.912	1.897	6.925	...	8.258	9.775	4.139	4.351
2015	25.909	2.343	2.162	4.975	...	5.933	11.466	2.930	3.914
2016	24.291	2.412	2.313	4.223	...	4.880	14.777	2.412	3.004

6 rows × 12 columns

Рейтинг в 2016 году:

1 место - **Action**, 2 место - **Shooter**, 3 место - **Sports**, 4 место - **Role-Playing**, 5 место - **Racing**, 6 место - **Simulation**, 7 место - **Platform**, 8 место - **Misc**, 9 место - **Strategy**, 10 место - **Adventure**, 11 место - **Fighting**, 12 место - **Puzzle**

Доли жанров в общей структуре продаж игр в Европе



```
In [210...]: show_sales_parts(perspective_sales, 'year', 'genre', 'jp_sales',
    'Доли жанров в общей структуре продаж игр в Японии', color=fill_colors)
```

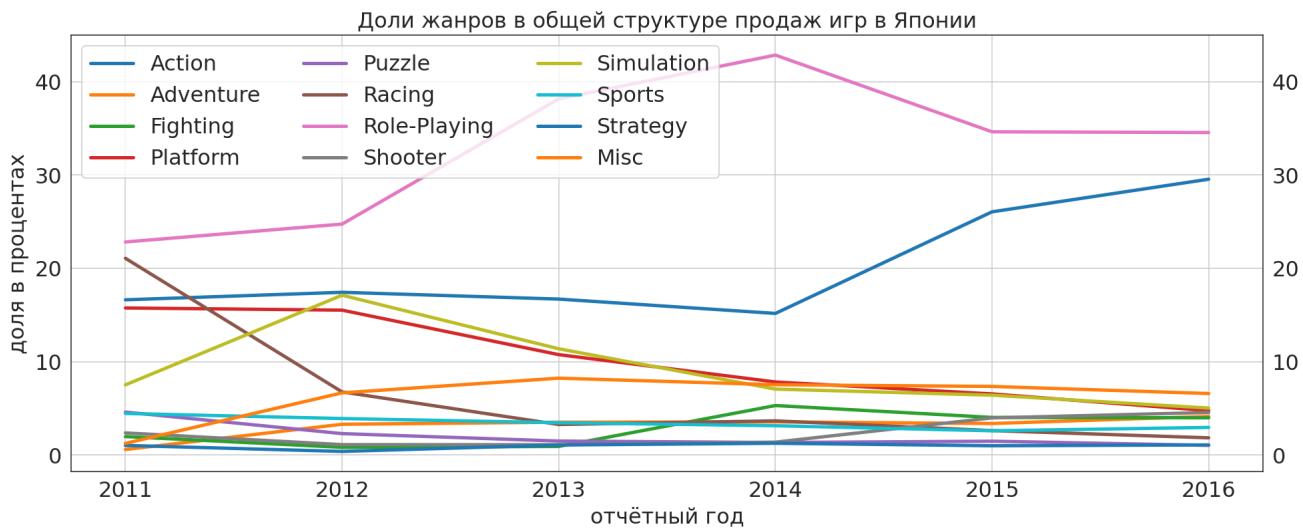
Доли жанров в общей структуре продаж игр в Японии в процентах

	Action	Adventure	Fighting	Platform	...	Simulation	Sports	Strategy	Misc
отчётный год									
2011	16.625	0.583	1.969	15.750	...	7.511	4.448	1.021	1.240
2012	17.429	3.288	0.817	15.518	...	17.120	3.902	0.379	6.655
2013	16.703	3.506	0.911	10.750	...	11.378	3.452	1.064	8.229
2014	15.166	3.542	5.295	7.821	...	7.057	3.131	1.251	7.525
2015	26.054	3.370	4.023	6.543	...	6.408	2.590	0.987	7.344
2016	29.548	4.132	3.982	4.746	...	5.022	2.949	1.075	6.583

6 rows × 12 columns

Рейтинг в 2016 году:

1 место - **Role-Playing**, 2 место - **Action**, 3 место - **Misc**, 4 место - **Simulation**, 5 место - **Platform**, 6 место - **Shooter**, 7 место - **Adventure**, 8 место - **Fighting**, 9 место - **Sports**, 10 место - **Racing**, 11 место - **Strategy**, 12 место - **Puzzle**



```
In [211]: show_sales_parts(perspective_sales, 'year', 'genre', 'other_sales',
   'Доли жанров в общей структуре продаж игр за пределами Америки, Европы и Японии', color=fill_colors)
```

Доли жанров в общей структуре продаж игр за пределами Америки, Европы и Японии в процентах

отчётный год	Action	Adventure	Fighting	Platform	Puzzle	Racing	Role-Playing	Simulation	Sports	Strategy	Misc
2011	14.284	2.241	1.333	6.283	...	15.245	6.593	12.360	1.132		
2012	20.677	1.623	1.567	9.479	...	11.757	6.332	7.988	2.737		
2013	21.470	1.874	1.840	10.386	...	8.635	8.442	6.376	3.152		
2014	29.113	1.621	2.721	6.552	...	4.810	10.855	3.087	4.874		
2015	27.260	1.972	3.077	4.337	...	3.158	13.004	1.923	4.020		
2016	25.222	2.163	3.280	3.893	...	2.453	15.157	1.474	3.021		

6 rows x 12 columns

Рейтинг в 2016 году:

1 место - **Shooter**, 2 место - **Action**, 3 место - **Sports**, 4 место - **Role-Playing**, 5 место - **Platform**, 6 место - **Racing**, 7 место - **Fighting**, 8 место - **Misc**, 9 место - **Simulation**, 10 место - **Adventure**, 11 место - **Strategy**, 12 место - **Puzzle**



На популярность жанров в различных регионах мира оказывают влияние социокультурные и психологические особенности населения: запад предпочитает стрельбу и действие, восток в лице Японии - ролевые игры.

Влияние рейтинга ESRB на продажи в регионах

```
In [212]: show_sales_parts(perspective_sales, 'year', 'rating_rus', 'na_sales',
   'Доли рейтингов в общей структуре продаж игр в Северной Америке', loc='lower center')
```

Доли рейтингов в общей структуре продаж игр в Северной Америке в процентах

отчётный год	EC (для детей младшего возраста)	E (для всех)	E10+ (для всех от 10 лет и старше)	T (подросткам)	M (для взрослых)	
2011	0.112	34.677		12.936	35.480	16.795
2012	0.066	34.899		16.377	27.755	20.903
2013	0.039	34.339		17.564	19.270	28.789
2014	0.020	25.878		18.078	19.238	36.787
2015	0.012	21.773		16.158	18.206	43.851

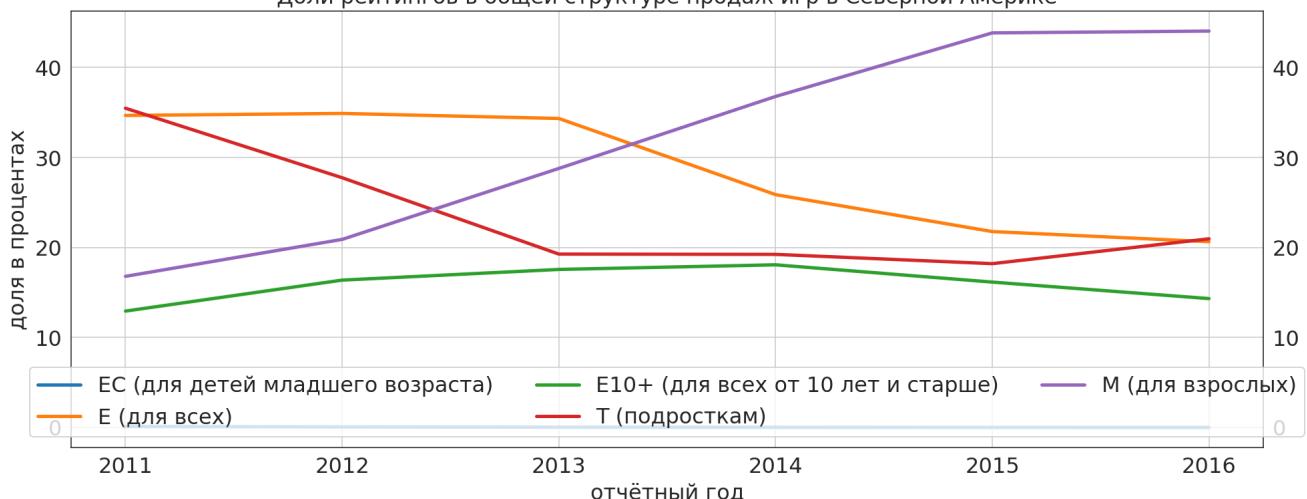
отчёtnый год

2016	0.008	20.648	14.325	20.968	44.051
------	-------	--------	--------	--------	--------

Рейтинг в 2016 году:

1 место - **M (для взрослых)**, 2 место - **T (подросткам)**, 3 место - **E (для всех)**, 4 место - **E10+ (для всех от 10 лет и старше)**, 5 место - **EC (для детей младшего возраста)**

Доли рейтингов в общей структуре продаж игр в Северной Америке



```
In [213]: show_sales_parts(perspective_sales, 'year', 'rating_rus', 'eu_sales',
'Dоли рейтингов в общей структуре продаж игр в Европе', loc='lower center')
```

Доли рейтингов в общей структуре продаж игр в Европе в процентах

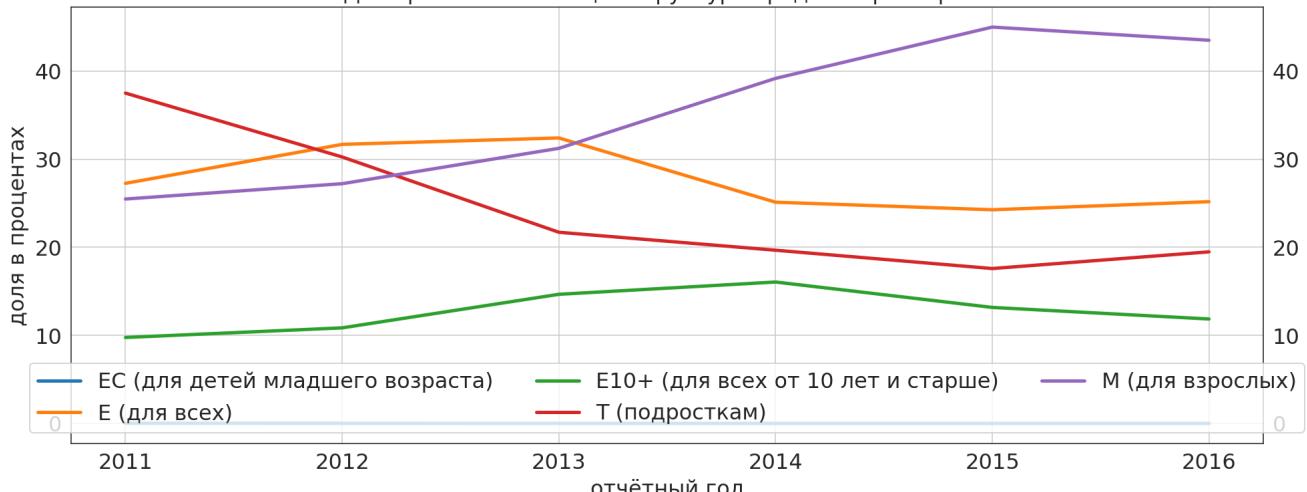
отчёtnый год

2011	0.020	27.253	9.757	37.497	25.474
2012	0.013	31.677	10.849	30.247	27.214
2013	0.009	32.406	14.658	21.701	31.227
2014	0.004	25.112	16.053	19.662	39.169
2015	0.003	24.255	13.170	17.583	44.990
2016	0.002	25.170	11.855	19.476	43.497

Рейтинг в 2016 году:

1 место - **M (для взрослых)**, 2 место - **E (для всех)**, 3 место - **T (подросткам)**, 4 место - **E10+ (для всех от 10 лет и старше)**, 5 место - **EC (для детей младшего возраста)**

Доли рейтингов в общей структуре продаж игр в Европе



```
In [214]: show_sales_parts(perspective_sales, 'year', 'rating_rus', 'jp_sales',
'Dоли рейтингов в общей структуре продаж игр в Японии', loc='lower center')
```

Доли рейтингов в общей структуре продаж игр в Японии в процентах

отчёtnый год

2011	0.000	53.231	15.459	29.532	1.779
2012	0.000	52.603	15.287	26.421	5.690
2013	0.000	42.849	11.116	18.843	27.192

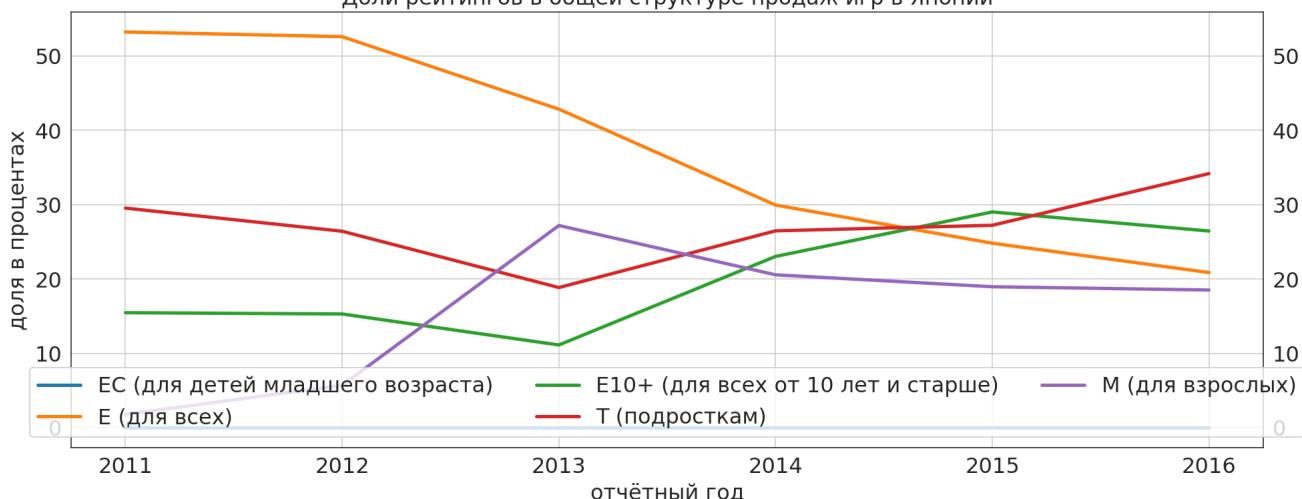
отчёtnый год

2014	0.000	29.942	23.030	26.473	20.554
2015	0.000	24.812	29.016	27.220	18.952
2016	0.000	20.862	26.450	34.179	18.509

Рейтинг в 2016 году:

1 место - T (подросткам), 2 место - E10+ (для всех от 10 лет и старше), 3 место - E (для всех), 4 место - M (для взрослых), 5 место - EC (для детей младшего возраста)

Доли рейтингов в общей структуре продаж игр в Японии



```
In [215...]: show_sales_parts(perspective_sales, 'year', 'rating_rus', 'other_sales',
    'Доли рейтингов в общей структуре продаж игр за пределами Америки, Европы и Японии', loc='lower center')
```

Доли рейтингов в общей структуре продаж игр за пределами Америки, Европы и Японии в процентах

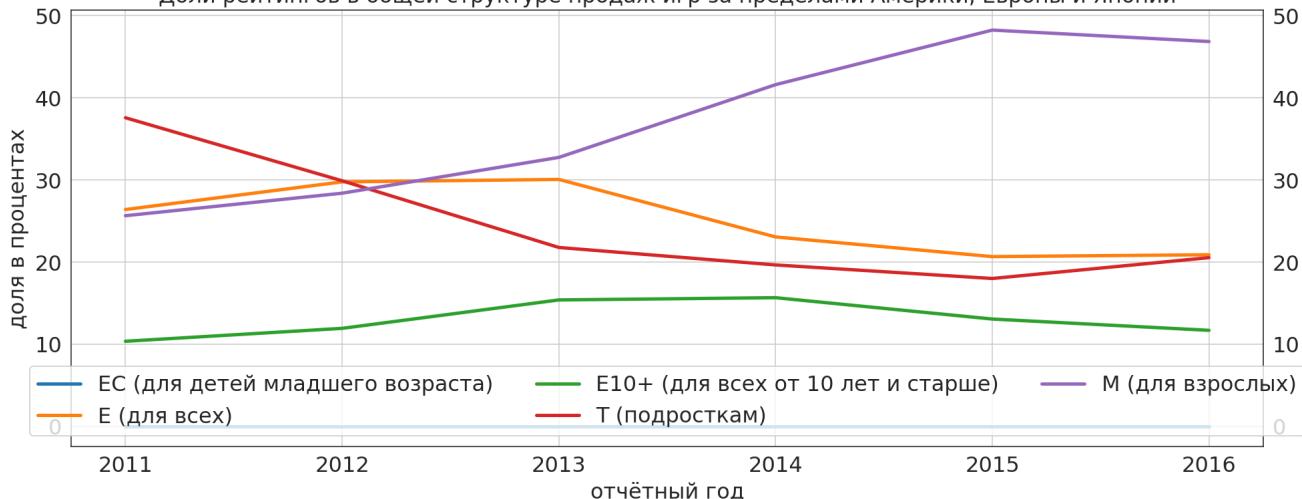
отчёtnый год

2011	0.000	26.403	10.367	37.578	25.652
2012	0.000	29.769	11.939	29.900	28.392
2013	0.000	30.069	15.398	21.785	32.748
2014	0.000	23.067	15.670	19.658	41.604
2015	0.000	20.672	13.072	18.008	48.248
2016	0.000	20.898	11.698	20.547	46.857

Рейтинг в 2016 году:

1 место - M (для взрослых), 2 место - E (для всех), 3 место - T (подросткам), 4 место - E10+ (для всех от 10 лет и старше), 5 место - EC (для детей младшего возраста)

Доли рейтингов в общей структуре продаж игр за пределами Америки, Европы и Японии



Рейтинг ESRB оказывает влияние на продажи игр в отдельном регионе. Везде кроме Японии основную долю продаж имеет отметка M (для взрослых), в Японии она лишь на четвертом месте, но и разница между остальными рейтингами в Стране восходящего солнца несущественная. Общая закономерность - игры для детей младшего возраста (EC) имеют очень малую долю продаж.

Нигде не представлены игры «AO» («Adults Only 18+») — «Только для взрослых». Возможно, они просто не попали в рейтинг из-за законодательного ограничения на их продвижение. Кто знает, может эта чёрная дыра в скором времени станет для кого-то золотой жилой?

Выводы по шагу 4

1. Список самых популярных платформ в разных регионах отличается: PS4 - фаворит везде кроме Японии, там предпочитают 3DS. Во всём мире

наблюдается устойчивый тренд на увеличение популярности **PS4**.

2. На популярность **жанров** в различных регионах мира оказывают влияние социокультурные и психологические особенности населения: запад предпочитает стрельбу и действие, восток в лице Японии - ролевые игры.
3. **Рейтинг ESRB** оказывает влияние на продажи игр в отдельном регионе. Везде кроме Японии основную долю продаж имеет отметка **M (для взрослых)**, в Японии она лишь на четвертом месте, но и разница между остальными рейтингами в Стране восходящего солнца несущественная. Общая закономерность - игры для детей младшего возраста (**EC**) имеют очень малую долю продаж.

Шаг 5. Проверка гипотез

Задание описано выше

Инструмент проверки гипотез

```
In [216]: def actual_game_slice(games_slice):  
    """Возвращает срез игр актуального периода и платформ"""  
    return table_games[(table_games.year_of_release >= min_year) &  
                       (table_games.generation == table_games.generation.max()) &  
                       games_slice].dropna()
```

```
In [217]: def plot_games_means(field_to_mean, other, title, figsize=None):  
    """Строит схематично плотности распределения для иллюстрации проверки гипотез"""  
    data = table_games[(table_games.year_of_release >= min_year) &  
                        (table_games.generation >= table_games.generation.max())][[field_to_mean, other]].dropna()  
    data[other] = data[other].astype(str)  
    joyplot(data=data, by=other, alpha=0.2, color='C9', overlap=0, figsize=figsize, grid=True);  
  
    for i, value in enumerate(sorted(data[other].unique())):  
        mi = data[data[other]==value][field_to_mean].mean()  
        plt.axvline(mi, c=fill_colors[i], lw=2.5, ls='--', label=f'средняя для {value} ({mi:.03f})')  
    plt.legend(loc='upper left')  
    plt.title(title)  
    plt.plot()
```

Для проверки гипотез о равенстве средних применим **t-критерий Стьюдента**, принимая во внимание, что сравниваемые независимые случайные величины распределены по закону, близкому к нормальному, но могут иметь разные дисперсии. Поэтому перед сравнением средних будем ещё проверять похожесть распределений по **U-критерию Манна-Уитни**. Опишем функцию, которая позволит автоматизировать проверку гипотез на основе библиотечных функций `scipy.stats.mannwhitneyu` и `scipy.stats.ttest_ind`:

```
In [218]: def check_mean_hypothesis(hypothesis, sample_1, sample_2, alpha=0.05, digits=2, *args, **kwargs):  
    """Проверяет гипотезу о равенстве средних выборок sample_1, sample_2  
    alpha - критический уровень статистической значимости  
    если p-value окажется меньше него - гипотеза отвергается  
    digits - количество цифр в дробной части для вывода на экран  
  
    Возвращает:  
        true - гипотеза не отвергнута  
        false - гипотеза отвергнута"""  
  
    # проверим схожесть распределений  
    results = st.mannwhitneyu(sample_1, sample_2)  
  
    # если распределения похожи, продолжаем тест  
    if results.pvalue >= alpha:  
        results = st.ttest_ind(sample_1, sample_2, *args, **kwargs)  
  
    if results.pvalue < alpha:  
        verdict = 'отвергнута'  
    else:  
        verdict = 'не может быть отвергнута'  
  
    display(HTML(f'Гипотеза {hypothesis} {verdict} по результатам проверки '  
        f'с критическим уровнем статистической значимости {alpha}' '  
        f'(получено p-значение {results.pvalue:.{digits}f}).'))  
  
    return results.pvalue >= alpha
```

Проверим правильность её работы на тестовом примере из урока от [Яндекс.Практикум](#).

```
In [219]: sample_1 = [3071, 3636, 3454, 3151, 2185, 3259, 1727, 2263, 2015,  
              2582, 4815, 633, 3186, 887, 2028, 3589, 2564, 1422, 1785,  
              3180, 1770, 2716, 2546, 1848, 4644, 3134, 475, 2686,  
              1838, 3352]  
sample_2 = [1211, 1228, 2157, 3699, 600, 1898, 1688, 1420, 5048, 3007,  
           509, 3777, 5583, 3949, 121, 1674, 4300, 1338, 3066,  
           3562, 1010, 2311, 462, 863, 2021, 528, 1849, 255,  
           1740, 2596]
```

Для указанных значений гипотеза не должна быть отвергнута при критическом уровне статистической значимости в 0.001:

```
In [220]: check_mean_hypothesis('о равенстве средних значений двух проверочных выборок', sample_1, sample_2, alpha=0.001, digits=3,  
                           equal_var=False)
```

Гипотеза о равенстве средних значений двух проверочных выборок не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.001** (получено p-значение **0.192**).

Out[220]: True

Наш "инструмент проверки гипотез" возвратил правильное значение. Скорее применим его, а то заказчик мог уже и утомиться ожидать результат!

Выбор уровня значимости

Уровень значимости статистического теста — допустимая для данной задачи вероятность ошибки первого рода (ложноположительного решения, *false positive*), то есть вероятность отклонить **нулевую гипотезу**, когда на самом деле она верна.

Другая интерпретация: **уровень значимости** — это такое (достаточно малое) значение вероятности события, при котором событие уже можно считать неслучайным.

Уровень значимости обычно обозначают греческой буквой α (альфа).

Обычно рекомендуется выбирать уровень значимости из априорных соображений. Однако на практике не вполне ясно, какими именно соображениями надо руководствоваться, и выбор часто сводится к назначению одного из популярных вариантов $\alpha = 0.005, 0.01, 0.05, 0.1$. В докомпьютерную эпоху эта стандартизация позволяла сократить объём справочных статистических таблиц. Теперь нет никаких специальных причин для выбора именно этих значений.

Наш дивиз остается прежним: "Мы не будем полагаться на случай!". В нашем проекте будем минимизировать вероятность отклонить нулевую гипотезу, когда на самом деле она верна. Для этого присвоим α маленькое значение:

In [221... alpha=0.01

Проверка гипотезы о средних пользовательских оценках платформ Xbox One и PC

Сформулируем гипотезы:

- нулевая гипотеза H_0 : средняя оценка, данной пользователями играм для платформ "Xbox One" и "PC" в актуальном периоде исследования, одинакова;
- альтернативная гипотеза H_1 : средняя оценка, данная пользователями играм для платформ "Xbox One" и "PC" в актуальном периоде исследования, существенно отличается.

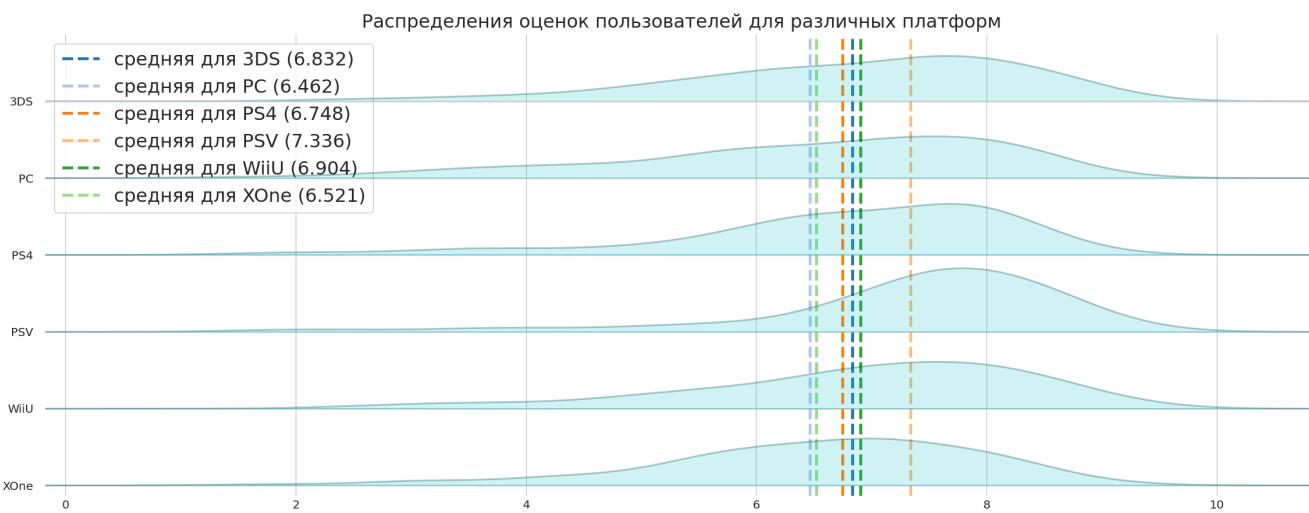
Проверим гипотезы:

In [222... _ = check_mean_hypothesis('о равенстве средней оценки, данной пользователями играм для платформ "Xbox One" и "PC" в актуальном периоде исследования, ',
 'и "PC" в актуальном периоде исследования,',
 actual_game_slice(table_games.platform == 'XOne').user_score,
 actual_game_slice(table_games.platform == 'PC').user_score,
 alpha=alpha, digits=2, equal_var=False)

Гипотеза о равенстве средней оценки, данной пользователями играм для платформ "Xbox One" и "PC" в актуальном периоде исследования, не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.68**).

Значит у нас нет оснований говорить, что пользователи в среднем по-разному относятся к платформ "Xbox One" и "PC". Судя по р-значению, оценки близки. Подтверждается это и иллюстрацией ниже (см. на две самые левые вертикальные линии пунктиром).

In [223... plot_games_means('user_score', 'platform', 'Распределения оценок пользователей для различных платформ')



А теперь сравним пользовательские оценки для других платформ. Выдвинем ряд гипотез и проверим их по схеме и с помощью подготовленной функции:

In [224... _ = check_mean_hypothesis('о равенстве средней оценки, данной пользователями играм для платформ "PC" и "PS4" в актуальном периоде исследования,',
 'и "PS4" в актуальном периоде исследования,',
 actual_game_slice(table_games.platform == 'PC').user_score,
 actual_game_slice(table_games.platform == 'PS4').user_score,
 alpha=alpha, digits=2, equal_var=False)

Гипотеза о равенстве средней оценки, данной пользователями играм для платформ "PC" и "PS4" в актуальном периоде исследования, не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.04**).

In [225... _ = check_mean_hypothesis('о равенстве средней оценки, данной пользователями играм для платформ "Xbox One" и "PSV" в актуальном периоде исследования,',
 'и "PSV" в актуальном периоде исследования,',
 actual_game_slice(table_games.platform == 'XOne').user_score,
 actual_game_slice(table_games.platform == 'PSV').user_score,
 alpha=alpha, digits=2, equal_var=False)

Гипотеза о равенстве средней оценки, данной пользователями играм для платформ "Xbox One" и "PSV" в актуальном периоде исследования, отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.00**).

In [226... _ = check_mean_hypothesis('о равенстве средней оценки, данной пользователями играм для платформ "3DS" и "WiiU" в актуальном периоде исследования,',
 'и "WiiU" в актуальном периоде исследования,',
 actual_game_slice(table_games.platform == '3DS').user_score,
 actual_game_slice(table_games.platform == 'WiiU').user_score,
 alpha=alpha, digits=2, equal_var=False)

Гипотеза о равенстве средней оценки, данной пользователями играм для платформ "3DS" и "WiiU" в актуальном периоде исследования, не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.56**).

Проверка гипотезы о средних пользовательских оценках жанров Action и Sports

Сформулируем гипотезы:

- нулевая гипотеза H_0 : **средняя оценка, данная пользователями играм жанров "Action" и "Sports" в актуальном периоде исследования, одинакова;**
- альтернативная гипотеза H_1 : **средняя оценка, данная пользователями играм жанров "Action" и "Sports" в актуальном периоде исследования, существенно отличается.**

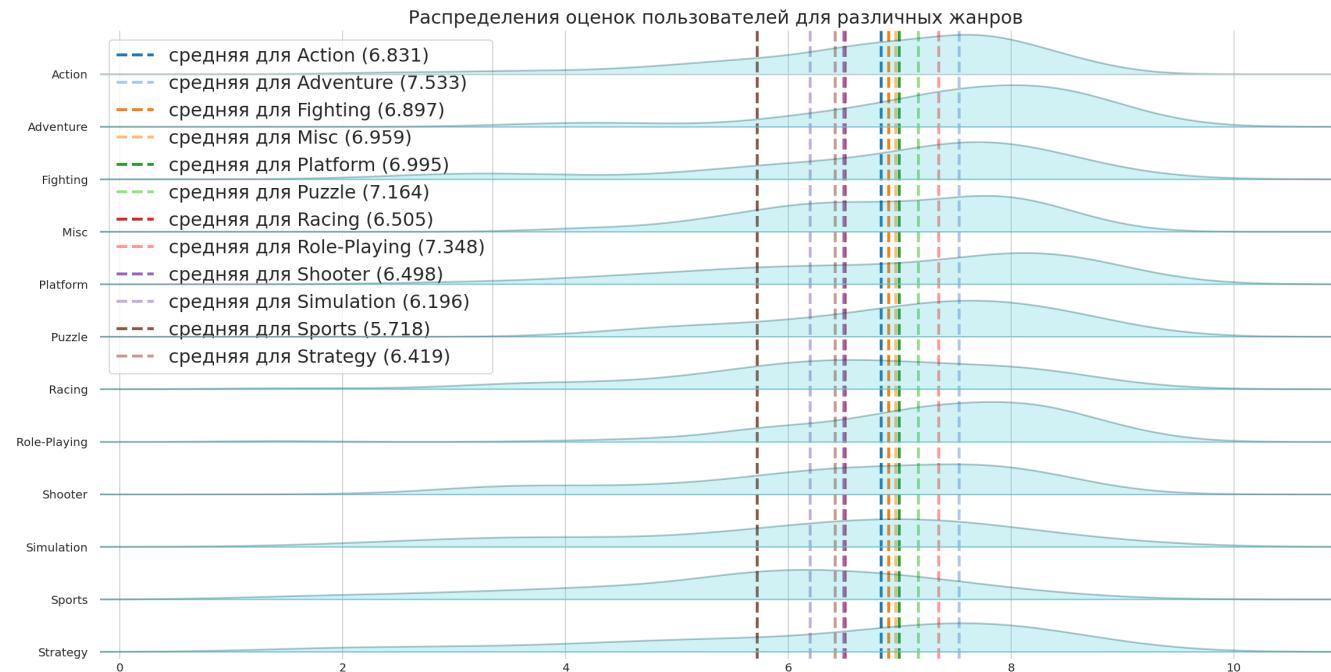
Проверим гипотезы:

```
In [227...]: _ = check_mean_hypothesis('о равенстве средней оценки, данной пользователями играм жанров "Action" и "Sports" ' + 'в актуальном периоде исследования,', actual_game_slice(table_games.genre == 'Action').user_score, actual_game_slice(table_games.genre == 'Sports').user_score, alpha=alpha, digits=10, equal_var=False)
```

Гипотеза о равенстве средней оценки, данной пользователями играм жанров "Action" и "Sports" в актуальном периоде исследования, отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0000000001**).

Значит средние оценки пользователей для этих игр существенно отличаются.

```
In [228...]: plot_games_means('user_score', 'genre', 'Распределения оценок пользователей для различных жанров', figsize=(16, 8))
```



Проверим несколько других гипотез о средних:

```
In [229...]: _ = check_mean_hypothesis('о равенстве средней оценки, данной пользователями играм жанров "Action" и "Shooter" ' + 'в актуальном периоде исследования,', actual_game_slice(table_games.genre == 'Action').user_score, actual_game_slice(table_games.genre == 'Shooter').user_score, alpha=alpha, digits=10, equal_var=False)
```

Гипотеза о равенстве средней оценки, данной пользователями играм жанров "Action" и "Shooter" в актуальном периоде исследования, не может быть отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0303694990**).

```
In [230...]: _ = check_mean_hypothesis('о равенстве средней оценки, данной критиками играм жанров "Action" и "Shooter" ' + 'в актуальном периоде исследования,', actual_game_slice(table_games.genre == 'Action').critic_score, actual_game_slice(table_games.genre == 'Shooter').critic_score, alpha=alpha, digits=10, equal_var=False)
```

Гипотеза о равенстве средней оценки, данной критиками играм жанров "Action" и "Shooter" в актуальном периоде исследования, отвергнута по результатам проверки с критическим уровнем статистической значимости **0.01** (получено р-значение **0.0000501763**).

Интересное наблюдение: пользователи оценивают жанр "Action" и "Shooter" в среднем одинаково, а критики по-разному.

Вывод по шагу 5

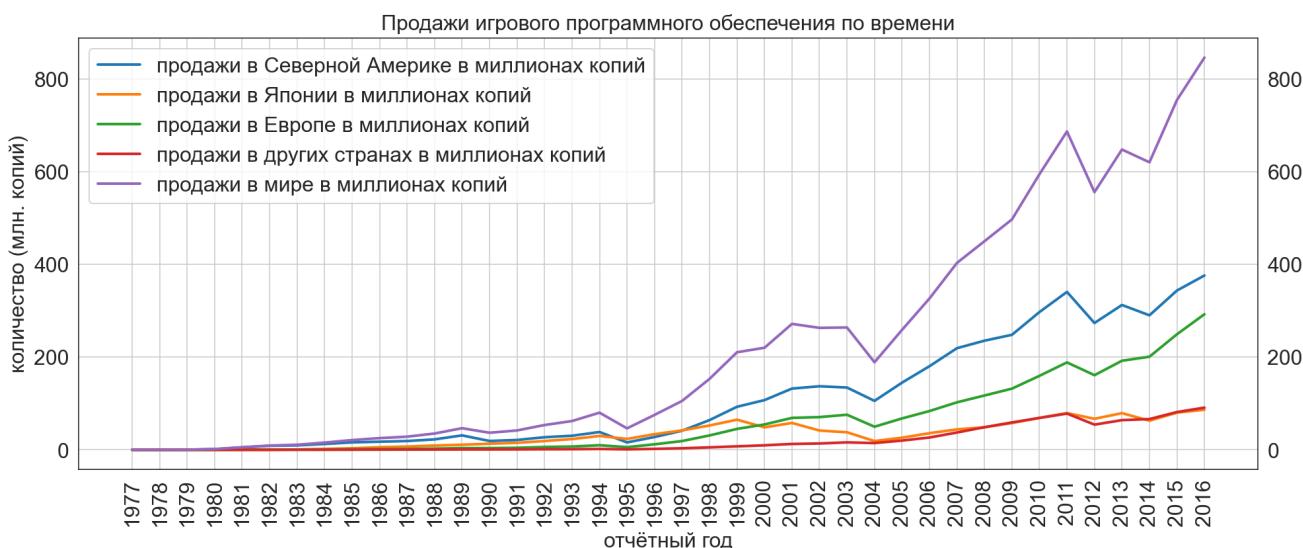
- разработан [инструмент](#) автоматизированной проверки гипотез о средних значениях выборок;
- обоснован выбор [критерия](#) проверки гипотез;
- уровень статистической значимости [задан](#) в интересах минимизации вероятности ошибок первого рода ($\alpha = 0.01$);
- в результате проверки заданных в задании гипотез установлено:
 - средняя оценка, данная пользователями играм для платформ "Xbox One" и "PC" в актуальном периоде исследования, [одинакова](#);
 - средняя оценка, данная пользователями играм жанров "Action" и "Sports" в актуальном периоде исследования, существенно [отличается](#);
- результаты исследования готовы к обобщению в [общем выводе](#).

Шаг 6. Общий вывод

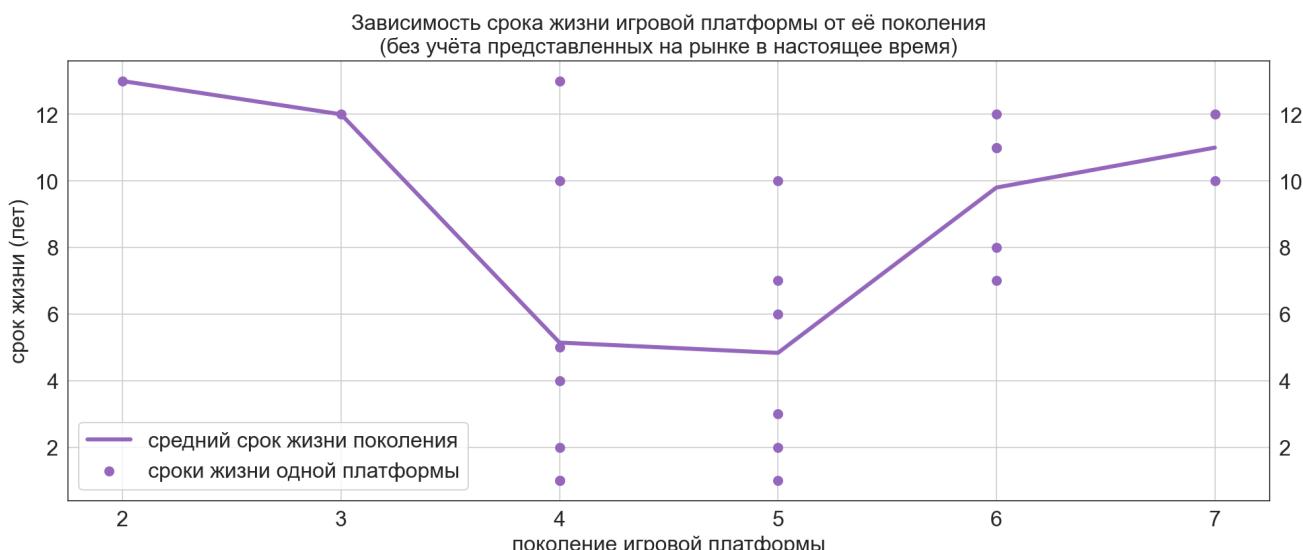
Полученные для анализа файлы имеют корректный формат. Массив был проанализирован после минимальной предобработки. По дополнительным источникам [создана](#) таблица с информацией об игровых платформах, годах их выпуска и поколениях. Каждой игре [сопоставлено](#) поколение игровой платформы, что облегчило получение срезов только по актуальным платформам на этапе анализа. Были исправлены [ошибки](#) в годах выпуска

нескольких игр. С помощью парсинга сайтов и использования известных значений для разных платформ **заполнены** пропуски в годах выпуска игр. Пропуски в оценках игр обоснованно **оставлены** без изменений. Описаны **причины**, которые могли привести к пропускам. **Обращено** внимание на аббревиатуру 'tbd' в столбцах с рейтингом, аномальное значение **обосновано заменено** на пустое значение. **Посчитаны** суммарные продажи во всех регионах, новые значения записаны в отдельный столбец.

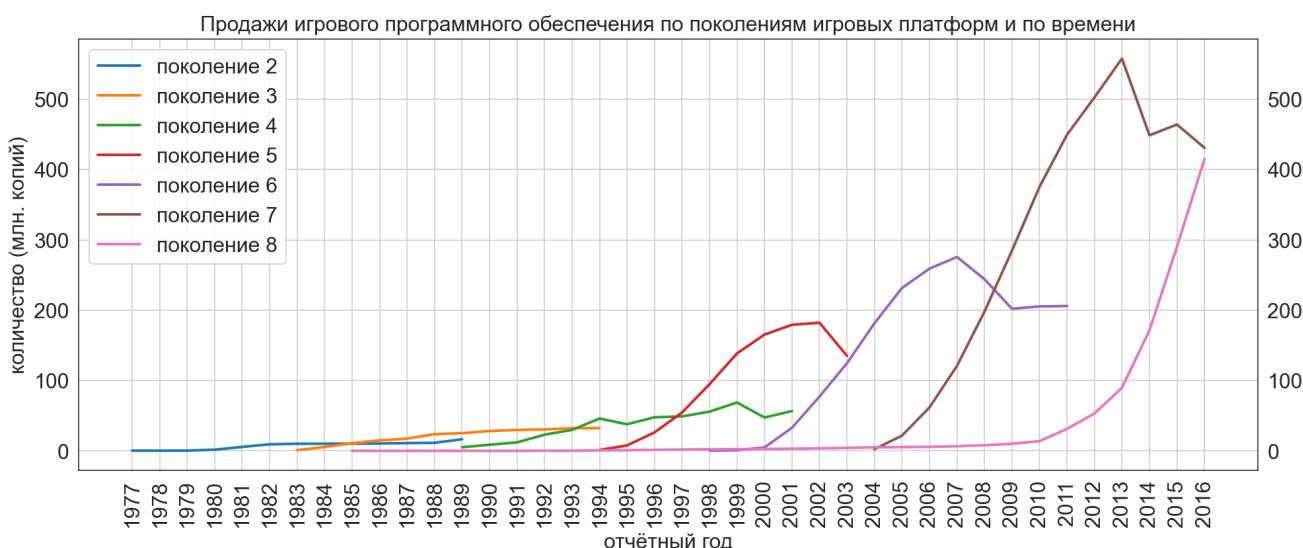
Анализ рынка игрового программного обеспечения, проведенный с **учетом** его продолжительности жизненного цикла, показал, что **общие продажи игр уверенно растут во всех регионах (темперы выше в Северной Америке), спрос стимулирует развитие отрасли и гарантирует получение прибыли от инвестиций в перспективные направления.**



Времена быстрой смены поколений (четвертое и пятое) постепенно уходят, средний срок жизни игровой платформы стал увеличиваться. Это значит, что мы можем планировать выпуск игр на более долгую перспективу, до десяти лет вперёд, естественно, учитывая, что к концу этого периода платформа будет терять свою привлекательность.



Программы для каждого нового поколения игровых устройств продаются в большем количестве, по сравнению с предыдущим. Игровые платформы последнего (восьмого) поколения ещё далеко не исчерпали потенциал для роста. Седьмое поколение сдаёт позиции, связывать свои перспективы с ним уже не стоит.

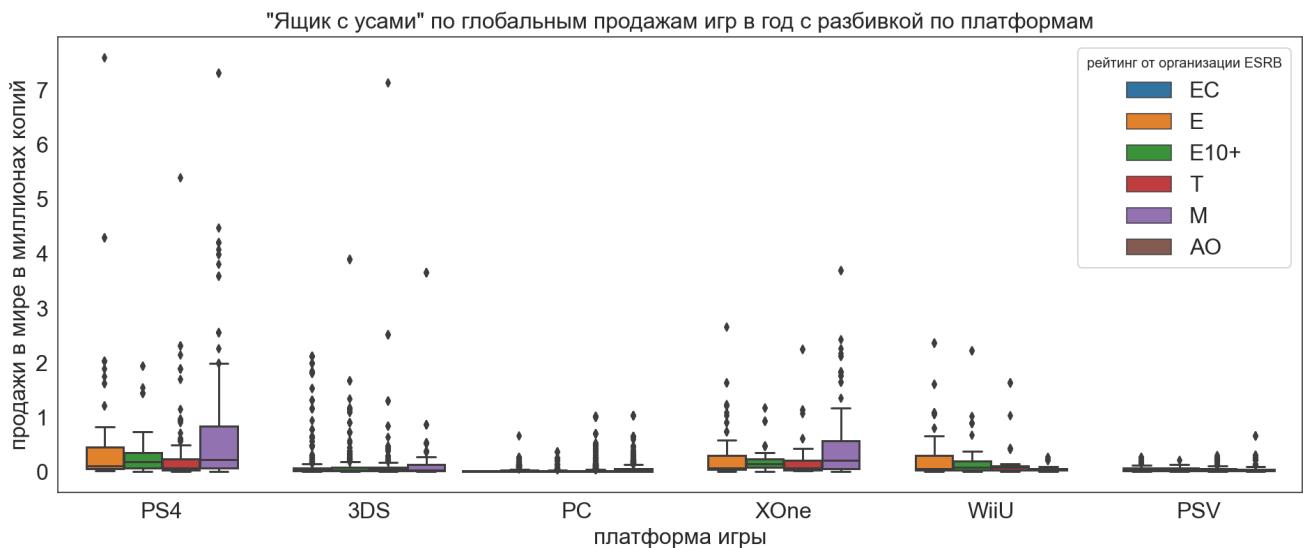


Очевидно, что PlayStation 4 (игровая приставка) стремительно завоевывает рынок, её доля в общей структуре продаж в 2016 году достигла 40.6%.

Конкурентам в лице **Nintendo 3DS** (портативная консоль) и **Xbox One** (игровая приставка) лидера не догнать, но их ресурса может хватить ещё года на четыре (на двоих они пока владеют около 38% рынка). **Персональный компьютер** продолжит пользоваться своим преимуществом универсального устройства, но их доля вряд ли сможет подняться выше 10% (пока заметен только явный тренд на снижение). На основе анализа данных за актуальный период (с 2011 года) можно составить следующий **рейтинг перспективности платформ**:

1. **PlayStation 4**
2. **Xbox One**
3. **Nintendo 3DS**
4. **Персональный компьютер**

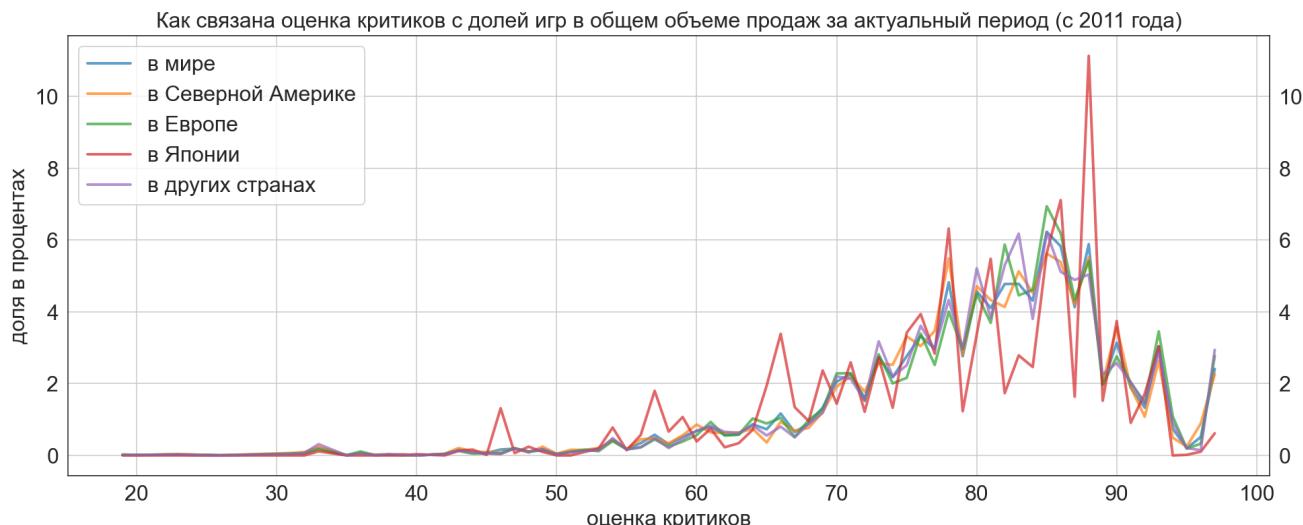
Разброс значений продаж одной игры в год невысокий, но для каждой платформы имеются игры-хиты, которые демонстрировали аномально высокие продажи по сравнению с конкурентами. Наибольшую выручку приносят игры для **PlayStation 4**. В структуре продаж платформ **PS4**, **3DS** и **XOne** преобладают игры с возрастным ограничением "для взрослых (M)", а для платформы **WiiU** - игры "для всех (E)".



Пользовательские оценки мало связаны с количеством продаж. Скорее всего, пользователи сначала покупают игру, а потом её оценивают, и палитра эмоций пестрит различными мнениями от восторга до полного неприятия. Критики прогнозируют продажи одной игры точнее для всех платформ. Хуже всего экспертные оценки коррелируют с продажами для персональных компьютеров, возможно, из-за широкой номенклатуры технических спецификаций этих устройств. Оценки критиков слабее коррелируют с продажами игр в Японии. Видимо, западное лекало не совсем учитывает менталитет людей Страны восходящего солнца.

По анализу корреляции в нашем проекте получилось, что сила связи между оценками пользователей и объемами продаж игры в год очень слабая. Таким образом, для прогнозирования продаж игрового программного обеспечения недостаточно использовать оценки в том виде, как они были нам предоставлены. Пользовательский рейтинг в нашем массиве "заморожен", т.е. имеет одно значение на весь период продажи игры, а он растягивается на несколько лет, за которые могут меняться и оценки, реагируя на многие факторы, в том числе появление игр-конкурентов. Для более точного предсказывания продаж по оценкам необходимо собирать их чаще и сопоставлять с продажами за более короткие периоды времени, в идеале около месяца, как это делается в рейтинговых агентствах типа [VGChartz](#).

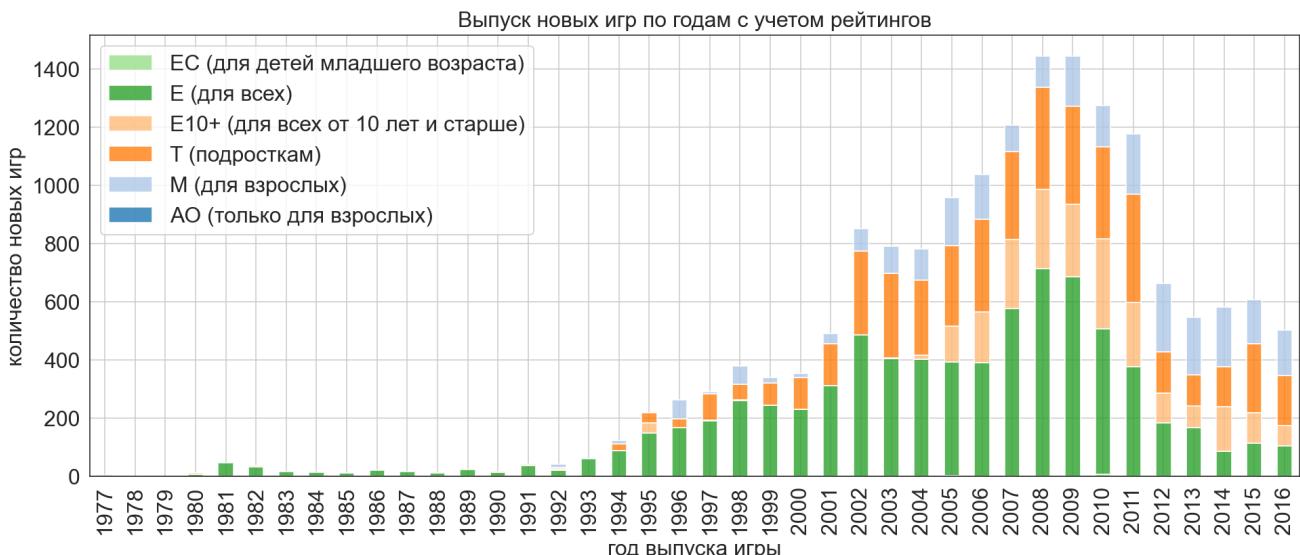
Высокие степени корреляции оценок пользователей и критиков просматриваются с общим объемом продаж. Оценка отдельно взятой игры не влияет на её продажи (большой вклад вносят и другие факторы). А вот общие продажи списка игр, которые выпущены, например, одной компанией, уже будут более чувствительны к реакции пользователей и критиков.



В связи с этим у крупных компаний игровой индустрии есть пути получения гарантированно высокой прибыли. Например, оставить в разработке 3-4 игры из сегментов (по жанру, платформе, возрастному рейтингу) с умеренными оценками (70-80 от критиков) и 1-2 из сегмента рискованных, но способных выстрелить большими продажами (с оценками 80-87). Возможны и другие варианты, которые целесообразно рассчитывать уже с помощью машинного обучения, принимая во внимание другие факторы.

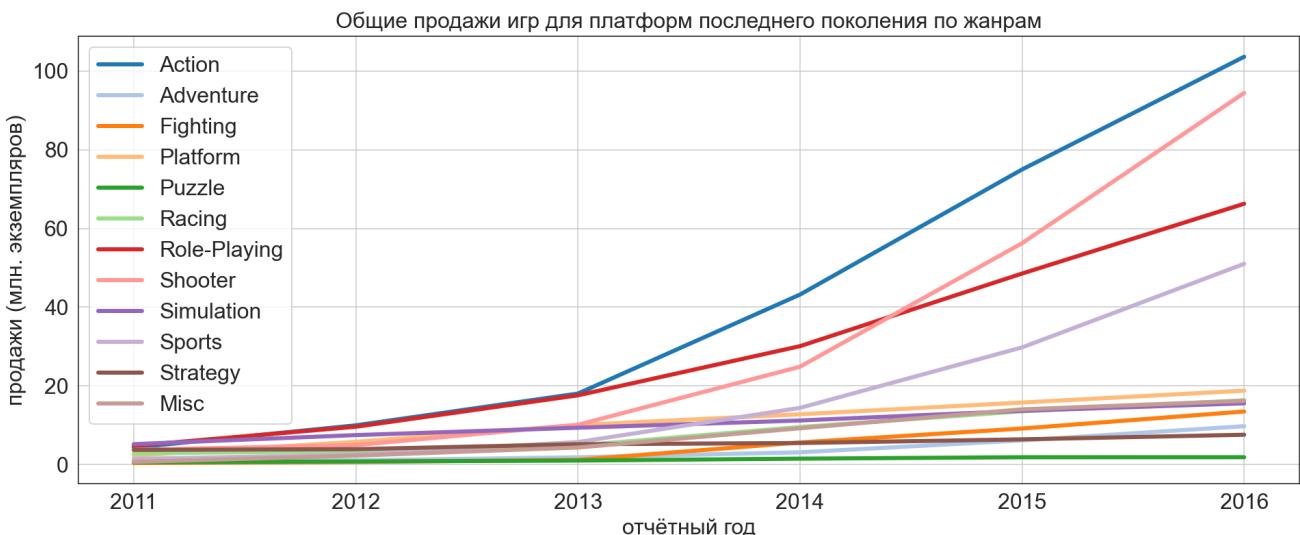
Начиная с 2003 года, игры жанра 'Action' удерживают пальму первенства по новинкам в год, выхватив её у спортивных игр, которые лидировали с 1996 по 2002 годы. Наиболее яркие всплески в выпуске новых игр, как мы можем видеть на [тепловой карте](#), происходили в жанре "Action" в 2008-2012 годах. Примерно в этот же период, если сравнивать с другими годами, появлялось много спортивных ("Sports") и прочих ("Misc") игр. Катализатором, возможно, стало увеличение вычислительной мощи игровых платформ.

В последнее время больше игр появляется с возрастным рейтингом для взрослых и подростков.

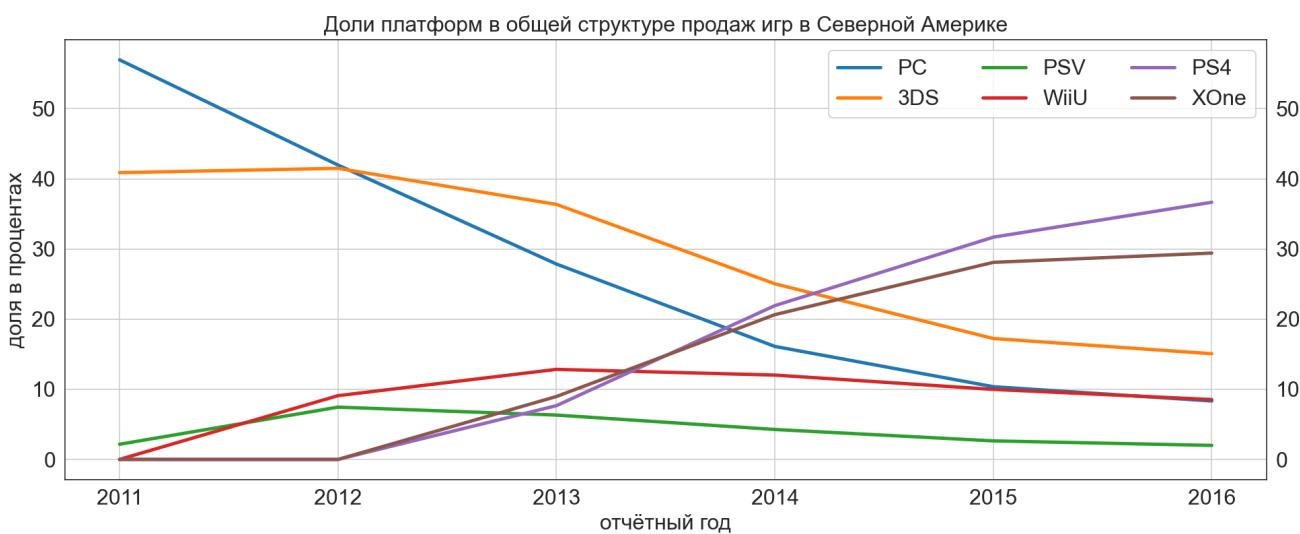


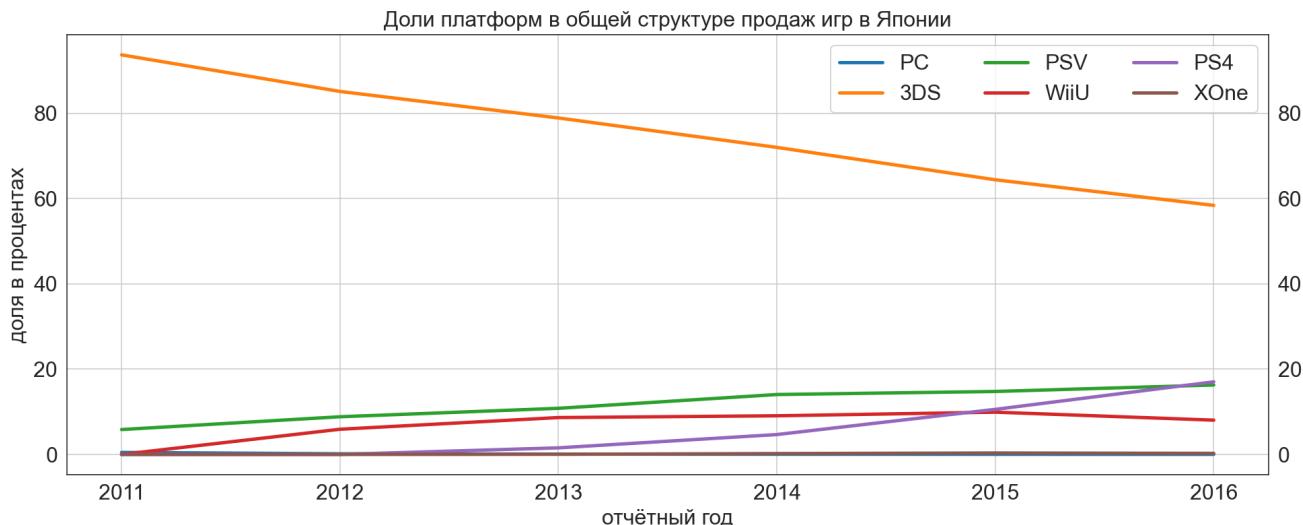
В последнее время, что особенно актуально для планирования деятельности на ближайшие пару лет, лидируют по количеству продаж четыре явных фаворита игровой индустрии:

1. Action
2. Shooter
3. Role-Playing
4. Sports



Список самых популярных [платформ](#) в разных регионах отличается: PS4 - фаворит везде кроме Японии, там предпочитают 3DS. Во всём мире наблюдается устойчивый тренд на увеличение популярности PS4.





На популярность [жанров](#) в различных регионах мира оказывают влияние социокультурные и психологические особенности населения: запад предпочитает стрельбу и действие, восток в лице Японии - ролевые игры.

Рейтинг ESRB оказывает влияние на продажи игр в отдельном регионе. Везде кроме Японии основную долю продаж имеет отметка **M (для взрослых)**, в Японии она лишь на четвертом месте, но и разница между остальными рейтингами в Стране восходящего солнца несущественная. Общая закономерность - игры для детей младшего возраста (**EC**) имеют очень малую долю продаж.

Применение разработанного [инструмента](#) автоматизированной проверки гипотез о средних значениях выборок показало, что:

- средняя оценка, данная пользователями играм для платформ "Xbox One" и "PC" в актуальном периоде исследования, [одинакова](#);
- средняя оценка, данная пользователями играм жанров "Action" и "Sports" в актуальном периоде исследования, существенно [отличается](#);
- пользователи оценивают жанр "Action" и "Shooter" в среднем одинаково, а критики по-разному.

Таблица контроля версий

Версия	Дата	Описание
1.0	02.12.2020	Направлено на первичную проверку

Планы на будущие проекты

Реализовать универсальную функцию проверки гипотез о средних значениях выборок с одновременным выводом графиков, параметров распределения и текстового заключения в виде, удобном для включения в общий вывод исследования.

Обновить массив данных, используя технологии скрапинга сайтов о рейтинге игр, их продажах и оценке (например, с [VGChartz](#)). Выполнить исследование с учетом актуальных данных о современных платформах, в том числе мобильных и онлайн. Оформить в виде выпускного проекта, нагружив его машинным обучением. Выбрать перспективное направление с большим потенциалом роста и малой конкуренцией. Заполнить собой найденную нишу. На какие деньги? Вопрос... ☺ Есть идеи?

[К оглавлению](#)

Бонус

Суп от сеньора

Здесь приведен протокол удачного выполнения парсинга для восстановления пропуска в годах выпуска игр. Текст показан на случай, если ячейки в конце ноутбука не выполнились. Если это так, я сожалею. Но причин может быть много, в том числе проблемы с подключением к Интернет, бан от Google, файрвол и т.д. Протокол иллюстрирует парсинг сайта [Metacritic](#) с информацией об играх и поисковика [Google](#). На его основе был получен [словарь](#) соответствий индекса в таблице с играми и правильного года выпуска игры:

Наберитесь терпения, пожалуйста! Суп будет готов минут через 10:

1. Madden NFL 2004 выпущена ... в 2003 году
2. FIFA Soccer 2004 выпущена ... в 2003 году
3. LEGO Batman: The Videogame выпущена ... в 2008 году
4. WWE Smackdown vs. Raw 2006 выпущена ... в 2005 году
5. Space Invaders выпущена ... в 2009 году
6. Rock Band выпущена ... в 2007 году
7. Frogger's Adventures: Temple of the Frog выпущена ... в 2001 году
8. LEGO Indiana Jones: The Original Adventures выпущена ... в 2008 году
9. Call of Duty 3 выпущена ... в 2006 году
10. Rock Band выпущена ... в 2007 году
11. Call of Duty: Black Ops выпущена ... в 2010 году
12. Rock Band выпущена ... в 2007 году
13. Triple Play 99 выпущена ... в 1998 году
14. Adventure выпущена ... в 1980 году
15. LEGO Batman: The Videogame выпущена ... в 2008 году
16. Combat выпущена ... в 1977 году
17. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
18. NASCAR Thunder 2003 выпущена ... в 2002 году
19. Hitman 2: Silent Assassin выпущена ... в 2002 году
20. Rock Band выпущена ... в 2007 году
21. Legacy of Kain: Soul Reaver выпущена ... в 1999 году
22. Donkey Kong Land III выпущена ... в 1997 году

23. Air-Sea Battle выпущена ... в 1977 году
24. Suikoden III выпущена ... в 2002 году
25. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
26. Wheel of Fortune выпущена ... в 2010 году
27. Yakuza 4 выпущена ... в 2011 году
28. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
29. Namco Museum выпущена ... в 2001 году
30. Rhythm Heaven выпущена ... в 2009 году
31. The Lord of the Rings: War in the North выпущена ... в 2011 году
32. Madden NFL 07 выпущена ... в 2006 году
33. MLB SlugFest 20-03 выпущена ... в 2002 году
34. The Lord of the Rings: War in the North выпущена ... в 2011 году
35. Shaun White Snowboarding выпущена ... в 2008 году
36. PES 2009: Pro Evolution Soccer выпущена ... в 2008 году
37. WarioWare: Twisted! выпущена ... в 2005 году
38. Madden NFL 11 выпущена ... в 2010 году
39. Test Drive Unlimited 2 выпущена ... в 2011 году
40. The Chronicles of Narnia: The Lion, The Witch and The Wardrobe выпущена ... в 2005 году
41. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
42. Monster Hunter 2 выпущена ... в 2006 году
43. Metal Gear Solid 2: Substance выпущена ... в 2003 году
44. Test Drive Unlimited 2 выпущена ... в 2011 году
45. Advance Wars: Days of Ruin выпущена ... в 2008 году
46. The Golden Compass выпущена ... в 2007 году
47. Madden NFL 06 выпущена ... в 2005 году
48. NASCAR: Dirt to Daytona выпущена ... в 2002 году
49. Madden NFL 2002 выпущена ... в 2001 году
50. Def Jam: Fight for NY выпущена ... в 2004 году
51. NBA Street Vol. 2 выпущена ... в 2003 году
52. Fishing Derby выпущена ... в 2013 году
53. Wet выпущена ... в 2009 году
54. Sonic the Hedgehog выпущена ... в 2006 году
55. Karate выпущена ... в 1982 году
56. Tiger Woods PGA Tour 07 выпущена ... в 2006 году
57. Circus Atari выпущена ... в 2012 году
58. The Chronicles of Riddick: Escape from Butcher Bay выпущена ... в 2004 году
59. Maze Craze: A Game of Cops 'n Robbers выпущена ... в неизвестном году
60. Silent Hill: Homecoming выпущена ... в 2008 году
61. Super Breakout выпущена ... в 2008 году
62. Robert Ludlum's The Bourne Conspiracy выпущена ... в 2008 году
63. NHL Slapshot выпущена ... в 2010 году
64. TERA выпущена ... в 2012 году
65. NFL GameDay 2003 выпущена ... в 2002 году
66. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
67. Harvest Moon: Save the Homeland выпущена ... в 2001 году
68. Robert Ludlum's The Bourne Conspiracy выпущена ... в 2008 году
69. Silent Hill: Homecoming выпущена ... в 2008 году
70. Hangman выпущена ... в 1978 году
71. The Golden Compass выпущена ... в 2007 году
72. NBA Live 2003 выпущена ... в 2002 году
73. Cubix Robots for Everyone: Clash 'n' Bash выпущена ... в неизвестном году
74. Dragon Ball Z: Budokai Tenkaichi 2 (JP sales) выпущена ... в неизвестном году
75. Tropico 4 выпущена ... в 2011 году
76. Tomb Raider (2013) выпущена ... в 2013 году
77. Bejeweled 3 выпущена ... в 2010 году
78. Custom Robo выпущена ... в 2004 году
79. Final Fantasy XI выпущена ... в 2004 году
80. Singularity выпущена ... в 2010 году
81. Dragster выпущена ... в 1980 году
82. All-Star Baseball 2005 выпущена ... в 2004 году
83. Star Wars Jedi Knight II: Jedi Outcast выпущена ... в 2002 году
84. Slot Machine выпущена ... в 1979 году
85. The Dukes of Hazzard II: Daisy Dukes It Out выпущена ... в 2000 году
86. Harvest Moon: The Tale of Two Towns выпущена ... в 2011 году
87. NBA Live 2003 выпущена ... в 2002 году
88. Shrek the Third выпущена ... в 2007 году
89. Nicktoons: Battle for Volcano Island выпущена ... в 2006 году
90. Haven: Call of the King выпущена ... в 2002 году
91. Unreal Championship 2: The Liandri Conflict выпущена ... в 2005 году
92. The Chronicles of Narnia: The Lion, The Witch and The Wardrobe выпущена ... в 2005 году
93. Pac-Man Fever выпущена ... в 2002 году
94. The Legend of Zelda: The Minish Cap(weekly JP sales) выпущена ... в неизвестном году
95. Indy 500 выпущена ... в 1977 году
96. Disgaea 3: Absence of Detention выпущена ... в 2012 году
97. Flag Capture выпущена ... в 2015 году

98. Gun выпущена ... в 2005 году
99. Rock Revolution выпущена ... в 2008 году
100. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
101. College Hoops 2K6 выпущена ... в 2005 году
102. Jonah Lomu Rugby Challenge выпущена ... в 2011 году
103. Mega Man X Collection выпущена ... в 2006 году
104. BioShock 2 выпущена ... в 2010 году
105. Singularity выпущена ... в 2010 году
106. Danganronpa: Trigger Happy Havoc выпущена ... в 2016 году
107. Jet X20 выпущена ... в неизвестном году
108. Tony Hawk's Downhill Jam выпущена ... в 2006 году
109. Tribes: Aerial Assault выпущена ... в 2002 году
110. Big Beach Sports 2 выпущена ... в 2010 году
111. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
112. Yu Yu Hakusho: Dark Tournament выпущена ... в 2004 году
113. Ghostbusters II выпущена ... в 1989 году
114. Breakaway IV выпущена ... в неизвестном году
115. Robotech: Battlecry выпущена ... в 2002 году
116. Move Fitness выпущена ... в неизвестном году
117. Valkyria Chronicles III: Unrecored Chronicles выпущена ... в неизвестном году
118. DanceDanceRevolution II выпущена ... в 2011 году
119. WRC: FIA World Rally Championship выпущена ... в 2006 году
120. Famista 64 выпущена ... в 1997 году
121. Dead Space 3 выпущена ... в 2013 году
122. Test Drive Unlimited 2 выпущена ... в 2011 году
123. Pet Zombies выпущена ... в 2011 году
124. Star Trek: Legacy выпущена ... в 2006 году
125. Backbreaker выпущена ... в 2010 году
126. Twisted Metal: Small Brawl выпущена ... в 2001 году
127. Otomedius Excellent выпущена ... в 2011 году
128. NBA Starting Five выпущена ... в 2002 году
129. Teen Titans выпущена ... в 2006 году
130. Trauma Team выпущена ... в 2010 году
131. Backbreaker выпущена ... в 2010 году
132. James Cameron's Dark Angel выпущена ... в 2002 году
133. Sword of the Samurai выпущена ... в 1990 году
134. Splatterhouse выпущена ... в 2010 году
135. Alone in the Dark: The New Nightmare выпущена ... в 2001 году
136. Vegas Party выпущена ... в 2009 году
137. Jurassic Park: The Game выпущена ... в 2011 году
138. Home Run выпущена ... в неизвестном году
139. eJay Clubworld выпущена ... в 2003 году
140. All-Star Baseball 2005 выпущена ... в 2004 году
141. Bejeweled 3 выпущена ... в 2010 году
142. Our House Party! выпущена ... в неизвестном году
143. WCW Backstage Assault выпущена ... в 2000 году
144. Bejeweled 3 выпущена ... в 2010 году
145. Disney's Cinderella: Magical Dreams выпущена ... в 2005 году
146. Transworld Surf выпущена ... в 2002 году
147. Street Fighter IV выпущена ... в 2009 году
148. Nintendo Puzzle Collection выпущена ... в 2003 году
149. Charm Girls Club: My Fashion Mall выпущена ... в неизвестном году
150. Record of Agarest War Zero выпущена ... в 2011 году
151. Rocksmith выпущена ... в 2012 году
152. Super Robot Wars OG Saga: Masou Kishin II - Revelation of Evil God выпущена ... в неизвестном году
153. Saru! Get You! Million Monkeys выпущена ... в неизвестном году
154. Street Hoops выпущена ... в 2002 году
155. WRC: FIA World Rally Championship выпущена ... в 2006 году
156. Godzilla: Destroy All Monsters Melee выпущена ... в 2002 году
157. The Daring Game for Girls выпущена ... в 2009 году
158. Major League Baseball 2K6 выпущена ... в 2006 году
159. Star Trek: Conquest выпущена ... в 2007 году
160. GiFTPiA выпущена ... в 2003 году
161. Happy Feet Two выпущена ... в 2011 году
162. Disney's Chicken Little: Ace In Action выпущена ... в 2006 году
163. Atsumare! Power Pro Kun no DS Koushien выпущена ... в неизвестном году
164. My Healthy Cooking Coach выпущена ... в 2009 году
165. Happy Feet Two выпущена ... в 2011 году
166. Luminous Arc 2 (JP sales) выпущена ... в неизвестном году
167. Happy Feet Two выпущена ... в 2011 году
168. Egg Monster Hero выпущена ... в 2005 году
169. The Daring Game for Girls выпущена ... в 2009 году
170. Demon Chaos выпущена ... в 2005 году
171. Samurai Shodown Anthology выпущена ... в 2009 году
172. Action Man-Operation Extreme выпущена ... в неизвестном году

173. Super Puzzle Fighter II выпущена ... в 2003 году
174. Charm Girls Club: My Fashion Show выпущена ... в 2009 году
175. Face Racers: Photo Finish выпущена ... в 2011 году
176. Zero: Tsukihami no Kamen выпущена ... в 2008 году
177. The Hidden выпущена ... в неизвестном году
178. Get Fit with Mel В выпущена ... в 2010 году
179. Rock Revolution выпущена ... в 2008 году
180. Happy Feet Two выпущена ... в 2011 году
181. Mega Man Battle Network: Operation Shooting Star выпущена ... в 2009 году
182. Smashing Drive выпущена ... в 2002 году
183. Port Royale 3 выпущена ... в 2012 году
184. Dream Trigger 3D выпущена ... в 2011 году
185. Dead Island: Riptide выпущена ... в 2013 году
186. Yoostar on MTV выпущена ... в 2011 году
187. Tornado выпущена ... в 2008 году
188. McFarlane's Evil Prophecy выпущена ... в 2004 году
189. Drake of the 99 Dragons выпущена ... в 2003 году
190. Build-A-Bear Workshop: Friendship Valley выпущена ... в 2010 году
191. Rayman Arena выпущена ... в 2002 году
192. Port Royale 3 выпущена ... в 2012 году
193. National Geographic Challenge! выпущена ... в 2011 году
194. Alex Rider: Stormbreaker выпущена ... в 2006 году
195. Chou Soujū Mecha MG выпущена ... в 2006 году
196. Prinny: Can I Really Be The Hero? (US sales) выпущена ... в неизвестном году
197. Combat Elite: WWII Paratroopers выпущена ... в 2005 году
198. Flip's Twisted World выпущена ... в 2010 году
199. Mobile Ops: The One Year War выпущена ... в 2008 году
200. Tom Clancy's Rainbow Six: Critical Hour выпущена ... в неизвестном году
201. Jewel Link Chronicles: Mountains of Madness выпущена ... в 2012 году
202. Captain America: Super Soldier выпущена ... в 2011 году
203. Mountain Bike Adrenaline выпущена ... в 2007 году
204. Drill Dozer выпущена ... в 2006 году
205. Captain America: Super Soldier выпущена ... в 2011 году
206. GRID выпущена ... в 2008 году
207. Tour de France 2011 выпущена ... в 2011 году
208. Reader Rabbit 2nd Grade выпущена ... в 2012 году
209. Monster Hunter Frontier Online выпущена ... в неизвестном году
210. RollerCoaster Tycoon выпущена ... в 1999 году
211. Battle vs. Chess выпущена ... в 2011 году
212. The History Channel: Great Battles - Medieval выпущена ... в 2011 году
213. Clockwork Empires выпущена ... в 2016 году
214. B.L.U.E.: Legend of Water выпущена ... в неизвестном году
215. GRID выпущена ... в 2008 году
216. NHL Hitz Pro выпущена ... в 2003 году
217. Luxor: Pharaoh's Challenge выпущена ... в 2007 году
218. Sega Rally 2006 выпущена ... в 2006 году
219. Half-Minute Hero 2 выпущена ... в неизвестном году
220. Housekeeping выпущена ... в неизвестном году
221. Major League Baseball 2K8 выпущена ... в 2008 году
222. Sabre Wulf выпущена ... в 2004 году
223. Swords выпущена ... в 2011 году
224. Beyond the Labyrinth выпущена ... в 2012 году
225. Bikkuriman Daijiten выпущена ... в неизвестном году
226. Majesty 2: The Fantasy Kingdom Sim выпущена ... в 2009 году
227. Fullmetal Alchemist: Brotherhood выпущена ... в неизвестном году
228. Combat Elite: WWII Paratroopers выпущена ... в 2005 году
229. Samurai Spirits: Tenkaichi Kenkakuden выпущена ... в неизвестном году
230. World of Tanks выпущена ... в 2011 году
231. Battle vs. Chess выпущена ... в 2011 году
232. Tom and Jerry in War of the Whiskers выпущена ... в 2002 году
233. Super Duper Sumos выпущена ... в 2003 году
234. The King of Fighters: Maximum Impact - Maniax выпущена ... в 2005 году
235. Combat Wings: The Great Battles of WWII выпущена ... в 2012 году
236. Tube Slider выпущена ... в 2003 году
237. Umineko no Naku Koro ni San: Shinjitsu to Gensou no Yasoukyoku выпущена ... в неизвестном году
238. Payout Poker & Casino выпущена ... в 2006 году
239. Wii de Asobu: Metroid Prime выпущена ... в неизвестном году
240. Legacy of Ys: Books I & II выпущена ... в 2009 году
241. Saint выпущена ... в 2009 году
242. Steal Princess выпущена ... в 2009 году
243. Mario Tennis выпущена ... в 2000 году
244. Runaway: A Twist of Fate выпущена ... в 2011 году
245. Yu-Gi-Oh! 5D's Wheelie Breakers (JP sales) выпущена ... в неизвестном году
246. Cabela's Alaskan Adventure выпущена ... в 2006 году
247. Writing and Speaking Beautiful Japanese DS выпущена ... в неизвестном году

248. Virtua Quest выпущена ... в 2005 году
249. Shonen Jump's Yu-Gi-Oh! GX Card Almanac выпущена ... в неизвестном году
250. Without Warning выпущена ... в 2005 году
251. PDC World Championship Darts 2008 выпущена ... в 2009 году
252. Dinotopia: The Sunstone Odyssey выпущена ... в 2003 году
253. Jet Impulse выпущена ... в 2007 году
254. Dream Dancer выпущена ... в 2009 году
255. Dance! It's Your Stage выпущена ... в 2010 году
256. Football Manager 2007 выпущена ... в 2006 году
257. Ferrari: The Race Experience выпущена ... в 2011 году
258. Aquaman: Battle for Atlantis выпущена ... в 2003 году
259. WRC: FIA World Rally Championship выпущена ... в 2006 году
260. Homeworld Remastered Collection выпущена ... в 2015 году
261. Shorts выпущена ... в 2009 году
262. AKB1/48: Idol to Guam de Koishitara... выпущена ... в неизвестном году
263. Brothers in Arms: Furious 4 выпущена ... в неизвестном году
264. Agarest Senki: Re-appearance выпущена ... в неизвестном году
265. PDC World Championship Darts 2008 выпущена ... в 2009 году
266. Freaky Flyers выпущена ... в 2003 году
267. Inversion выпущена ... в 2012 году
268. Hakuouki: Shinsengumi Kitan выпущена ... в неизвестном году
269. Virtua Quest выпущена ... в 2005 году

Восстановлены годы выпуска игр в количестве 234 шт. из 269 с пропусками. Wall time: 15min 8s

```
print('restored_years = {', ', '.join([f'{i}:{year}' for i, year in years.items()]), '}')
```

```
restored_years = {  
183:2003,377:2003,456:2008,475:2005,609:2009,627:2007,657:2001,678:2008,719:2006,805:2007,1131:2010,1142:2007,1301:1998,1506:1980,1538:2008,1585:1977,1609:2011,1650:2  
}
```

Рецепт супа сеньора

А это рецепты от сеньора с его комментариями:

```
In [231...]  
# закупим ингредиенты  
ingredients = \  
{  
    'Игры':  
        PandasFile  
        (  
            'Информация о продажах игровых программ',           # назначение файла  
            os.path.join(local_folder, 'games.csv'),             # локальный путь к файлу  
            ',',          # сетевой адрес файла скрыт из-за правовых ограничений  
            {},           # список параметров pandas.read_csv  
            {'name': DataField('название игры',  
                               'название игры',  
                               np.object, True, 'Name'),  
             'platform': DataField('платформа',  
                                   'платформа игры',  
                                   np.object, True, 'Platform'),  
             'year_of_release': DataField('год выпуска',  
                                           'год выпуска игры',  
                                           np.int64, True,  
                                           'Year_of_Release')},  
            None  
        )  
}
```

"Информация о продажах игровых программ" (datasets/games.csv) был загружен ранее.
Открываю "Информация о продажах игровых программ" с помощью Pandas ... OK
Переименовываю поле "Name" в "name" ... OK
Переименовываю поле "Platform" в "platform" ... OK
Переименовываю поле "Year_of_Release" в "year_of_release" ... OK

```
In [232...]  
# снимем упаковку  
table_names = ingredients['Игры'].df
```

```
In [233...]  
# помоем продукты  
table_names.drop(table_names[table_names.name.isna()].index, inplace=True)  
table_names.name = table_games.name.str.replace('\s+', ' ')  
table_names.name = table_games.name.str.strip()
```

```
In [234...]  
# обновим обои кухни  
!pip3 install --upgrade --quiet --user lxml html5lib requests bs4
```

```
In [235...]  
# возьмем кастрюлю...  
import requests  
# ... ложку ...  
from urllib.parse import quote  
# ... часы, чтобы засекать время кипения...  
from time import sleep  
# ... и пакет с красивым супом  
from bs4 import BeautifulSoup
```

```
In [236...]  
%time  
print('Наберитесь терпения, пожалуйста! Суп будет готов минут через 15:')
```

```
# наденем фартук...  
headers = {'User-Agent':  
           'Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 '  
           '(KHTML, like Gecko) Chrome/86.0.4240.77 YaBrowser/20.11.0.817 '  
           'Yowser/2.5 Safari/537.36'}
```

```

# приготовимся к дегустации
count = 0
years = {}

# зададим интервал помешиваний ложкой в секундах
# часто мешать вредно, можно у кастриюли эмаль отбить
request_interval = 5

# зададим количество помешиваний ложкой
max_pages = 5

# для всех рецептов супов, которые ещё не пробовали...
for n, row in enumerate(table_names[(~table_names.name.isna()) &
                                      table_names.year_of_release.isna()].itertuples(index=True)):
    # произнесём магическое заклинание для вкуса
    print(f'{n+1}. {row.name} выпущена ... ', end=' ')
    pattern = row.name.lower()
    page_no = 1

    # выпьем воды, чтобы успокоить вкусовые рецепторы
    year = 0

    # пока суп не готов и терпение барить не лопнуло...
    while year == 0 and page_no <= max_pages:
        # найдем номер страницы рецепта в оглавлении...
        link = f'https://www.metacritic.ru/search/game/{quote(row.name)}/results'
        if page_no > 1:
            link = link + f'?page={page_no}'

        # попытаемся найти страницу...
        website_url = requests.get(link, headers=headers, allow_redirects=True, timeout=5)

        # если страницу не вырвали джуны-вандалы...
        if website_url.ok:
            # нальём воды в кастриюлю...
            soup = BeautifulSoup(website_url.text, 'lxml')

            try:
                # забросим в кастриюлю все ингредиенты...
                for i in soup.find_all('div', {'class': 'main_stats'}):
                    # если суп забурлил, помешаем его ложкой...
                    if i.find('h3', {'class': 'product_title basic_stat'}).text.strip().lower() == pattern:
                        # добавляем суп до кипения...
                        platform = i.find('span', {'class': 'platform'})
                        y = int(platform.parent.text.strip()[-4:])
                        # снимем пробу, если суп уже готов...
                        if y > min_release_year and y <= max_release_year:
                            # ...снимем его с плиты
                            year = y
                            break
            except:
                # если просыпалась соль, то нет и супа, тихо уходим
                pass

            # тщательно помоем кастриюлю
            sleep(request_interval)
            page_no += 1

        # проверим наличие воды в кастриюле, если нет, то гасим огонь
        if soup.find('a', class_='page_num') is None:
            page_no = max_pages + 1

    # если суп не удался, выпьем его и пойдем подглядим, как его готовит соседка
    if year == 0:
        link = f'https://www.google.com/search?q=%22{quote(row.name.replace(" ", "+"))}%22' \
               f'+game+for%22{quote(row.platform.replace(" ", "+"))}%22'

        # смутился к соседке...
        website_url = requests.get(link, headers=headers, allow_redirects=True, timeout=5)

        # если открыли дверь...
        if website_url.ok:
            # промтигаем кастриюлю...
            soup = BeautifulSoup(website_url.text, 'lxml')

            # забросим в кастриюлю все ингредиенты...
            info_tag = soup.find('div',
                                  {'class': 'mod',
                                   'data-atrid': "kc:/cvg/computer_videogame:release date"})
            # если суп забурлил, помешаем его ложкой...
            if info_tag is not None:
                # снимем пробу...
                try:
                    y = int(info_tag.text.strip().rstrip(' г.')[ -4:])
                    # если суп уже готов...
                    if y > min_release_year and y <= max_release_year:
                        # ...снимем его с плиты
                        year = y
                except:
                    # если просыпалась соль, то нет и супа, тихо уходим
                    pass
            else:
                # если соседка не хочет разговаривать...
                print('даже соседка говорит, что ', end='')

        if year != 0:
            # если суп удался, перепишем рецепт в свою книжечку :-)
            print('в', year, 'году')
            table_names.loc[row.Index, 'year_of_release'] = year
            years[row.Index] = year
            count += 1
    else:

```

```
# если суп не удался, грустно вздохнем
print('в неизвестном году')

display(HTML(f'{<br><b>Восстановлены годы выпуска игр в количестве {count} шт. из {n+1} с пропусками.</b>}'))
```

Наберитесь терпения, пожалуйста! Суп будет готов минут через 15:

1. Madden NFL 2004 выпущена ... в 2003 году
2. FIFA Soccer 2004 выпущена ... в 2003 году
3. LEGO Batman: The Videogame выпущена ... в 2008 году
4. WWE Smackdown vs. Raw 2006 выпущена ... в 2005 году
5. Space Invaders выпущена ... в 1978 году
6. Rock Band выпущена ... в неизвестном году
7. Frogger's Adventures: Temple of the Frog выпущена ... в 2001 году
8. LEGO Indiana Jones: The Original Adventures выпущена ... в 2008 году
9. Call of Duty 3 выпущена ... в 2006 году
10. Rock Band выпущена ... в неизвестном году
11. Call of Duty: Black Ops выпущена ... в 2010 году
12. Rock Band выпущена ... в неизвестном году
13. Triple Play 99 выпущена ... в 1998 году
14. Adventure выпущена ... в 1980 году
15. LEGO Batman: The Videogame выпущена ... в неизвестном году
16. Combat выпущена ... в 1977 году
17. LEGO Harry Potter: Years 5-7 выпущена ... в 2011 году
18. NASCAR Thunder 2003 выпущена ...

И так далее...

А вот и листочек с записями рецептов:

```
In [237]: print('restored_years = ', ', '.join([f'{i}:{year}' for i, year in years.items()]), ')')
restored_years = { 183:2003, 377:2003, 456:2008, 475:2005, 609:1978, 657:2001, 678:2008, 719:2006, 1131:2010, 1301:1998, 1506:1980, 1585:1977, 1609:2011 }
```

[Отдадим](#) рецепт джунам...

[К оглавлению](#)

Спасибо за проверку!