Изучение индустрии игр начала XXI века

- Автор: Егорова Ольга

Цели и задачи проекта:

Цель:

- изучить развитие игровой индустрии с 2000 по 2013 год на основе исторических данных, собранных из открытых источников.

Задачи:

- познакомиться с данными;
- проверить их корректность;
- провести предотбработку данных;
- сформировать срез данных за период;
- провести категоризацию данных по оценкам пользователей и экспертов;
- сформировать выводы.

Описание данных:

Данные `games.csv` содержат информацию о продажах игр разных жанров и платформ, а также пользовательские и экспертные оценки игр:

- `Name` — название игры.
- `Platform` — название платформы.
- `Year of Release` — год выпуска игры.
- `Genre` — жанр игры.
- `NA sales` — продажи в Северной Америке (в миллионах проданных копий).
- `EU sales` — продажи в Европе (в миллионах проданных копий).
- `JP sales` — продажи в Японии (в миллионах проданных копий).
- `Other sales` — продажи в других странах (в миллионах проданных копий).
- `Critic Score` — оценка критиков (от 0 до 100).
- `User Score` — оценка пользователей (от 0 до 10).
- `Rating` — рейтинг организации ESRB, которая определяет рейтинг компьютерных игр и присваивает им подходящую возрастную категорию.

 **Справочная информация о категориях рейтинга ESRB (*кликни для раскрытия*)**

▶ Подробнее

Содержание проекта:

1. Загрузка и знакомство с данными

Загрузим необходимые библиотеки для анализа данных и данные датасета `games.csv`. Затем выведем основную информацию о данных с помощью метода `info()` и первые строки датафрейма с помощью метода `head()`.

In [1]:
```
# Импортируем библиотеку pandas
import pandas as pd
```

In [2]:
```
# Выгружаем данные из датасета 'games.csv' в датафрейм 'games'
games = pd.read_csv('https://drive.google.com/uc?export=download&id=14qb_AECZr3en9wM1fE1a2ZvwbbZWa6jU')
```

In [3]:
```
# Выводим информацию о датафрейме
games.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16956 entries, 0 to 16955
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Name             16954 non-null  object
 1   Platform         16956 non-null  object
 2   Year of Release  16681 non-null  float64
 3   Genre            16954 non-null  object
 4   NA sales         16956 non-null  float64
 5   EU sales         16956 non-null  object
 6   JP sales         16956 non-null  object
 7   Other sales      16956 non-null  float64
 8   Critic Score     8242 non-null   float64
 9   User Score       10152 non-null  object
 10  Rating           10085 non-null  object
dtypes: float64(4), object(7)
memory usage: 1.4+ MB
```

In [4]:
```
# Выводим первые строки датафрейма на экран
games.head()
```

Out[4]:

	Name	Platform	Year of Release	Genre	NA sales	EU sales	JP sales	Other sales	Critic Score	User Score	Rating
0	Wii Sports	Wii	2006.0	Sports	41.36	28.96	3.77	8.45	76.0	8	E
1	Super Mario Bros.	NES	1985.0	Platform	29.08	3.58	6.81	0.77	NaN	NaN	NaN
2	Mario Kart Wii	Wii	2008.0	Racing	15.68	12.76	3.79	3.29	82.0	8.3	E
3	Wii Sports Resort	Wii	2009.0	Sports	15.61	10.93	3.28	2.95	80.0	8	E
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	11.27	8.89	10.22	1.00	NaN	NaN	NaN

Датасет `games.csv` содержит 11 столбцов и 16956 строк, в которых представлена информация о продажах игр разных жанров и платформ, в разных странах с указанием пользовательских и экспертных оценок.

Изучим типы данных и их корректность:

- **Строковые данные (object).** Семь столбцов имеют тип данных `object`:
  - `Name`, `Platform` содержат текстовую информацию - название игры и платформы. Здесь тип данных `object` определен корректно. Однако если набор значений в столбце `Platform` ограничен, то можно рассматривать их как категориальные и привести тип к `category`. Для начала убедимся, что данные в столбце корректны и не содержат неявных дубликатов.
  - `Genre` и `Rating` также содержат текстовые данные - наименование жанра игры и рейтинг компьютерной игры, представленный категориями. Тип данных `object` определен корректно. Данные столбцов можно рассматривать как категориальные признаки. Если набор значений для этих столбцов корректен и не содержит неявных дубликатов, можно их привести к типу `category`, чтобы улучшить производительность и оптимизировать память.
  - `EU sales` и `JP sales` хранят информацию о количестве продаж. Для таких данных рекомендуется использовать тип `float64`, который позволит выполнять вычисления и анализ числовых данных.
  - `User Score` хранят информацию об оценках пользователей. Как видно из 5 первых строк, значение оценки пользователя может быть дробным числом. Для таких данных рекомендуется использовать тип `float64`, который позволит выполнять различные вычисления, например, находить среднюю оценку по группам и т.д
- **Числовые значения с плавающей запятой (float64).** Четыре столбца имеют тип данных `float64`:
  - `Year of Release` содержит информацию о годе выпуска игры. Для таких данных корректным будет тип `datetime64`. Кроме того, год можно рассматривать как категориальный признак, в этом случае подойдет тип `category`. Также возможно привести к типу целочисленных значений `integer`. Для удобства расчётов оставим тип `float64` и вернемся к вопросу смены типа данных после предобработки.
  - `NA sales` и `Other sales` содержат информацию о количестве продаж игр в Северной Америке и в друих странах. Для таких данных рекомендуется оставить тип данных `float64`.
  - `Critic Score` содержит информацию об оценках критиков. Первые 5 строк не дают полного представления о всех возможных значениях этого столбца, т.е. не понятно являются ли значения целочисленными или дробными. Рекомендуется оставить и использовать тип `float64`, который позволит выполнять различные вычисления, например, находить среднюю оценку по группам и т.д.
- **Булевые значения (bool)** не представлены в датафрейме.
- **Целочисленные значения (int64)** не представлены в датафрейме.

После анализа типов данных видно, что для 8 столбцов из 11: `Name`, `Platform`, `Year of Release`, `Genre`, `NA sales`, `Other sales`, `Critic Score`, `Rating` типы данных определены корректно. Для 3 столцов: `EU sales`, `JP sales` и `User Score` требуется преобразование типов на `float64`. А также с целью оптимизации для всех числовых столбцов уменьшим разрядность.

В датафрейме 6 столбцов содержат пропуски - это `Name`, `Year of Release`, `Genre`, `Critic Score`, `User Score`, `Rating`

2. Проверка ошибок в данных и их предобработка

2.1 Названия столбцов датафрейма

Выведем названия всех столбцов датафрейма с помощью атрибута `columns` и оценим их написание

In [5]:
```
#Выводим имена столбцов в виде объекта Index
games.columns
```

Out[5]:
```
Index(['Name', 'Platform', 'Year of Release', 'Genre', 'NA sales', 'EU sales',
       'JP sales', 'Other sales', 'Critic Score', 'User Score', 'Rating'],
      dtype='object')
```

Названия столбцов представлены смешанным регистром с пробелами.

Приведём названия столбцов к стилю snake_casek, то есть приведём к нижнему регистру и заменим пробелы на нижнее подчёркивание:

- с помощью цикла созданим новый список с названиями столбцов в нижнем регистре без пробелов;
- с помощью функции `zip()` создадим словарь, где в качестве ключей будет список старых названий столцов, а в качестве значений - список новых названий столбцов;
- передадим параметру `columns` словарь с новыми названиями столбцов
- результат сохраним в новом датафрейме 'games_new'

In [6]:
```
# Создаём пустой список new_names_colomns
new_names_colomns = []
```

```
# Сохраняем в список new_names_colomns новые названия столбцов в нижнем регистре и в стиле snake case
for name_colomn in games.columns.tolist():
    new_names_colomns.append(name_colomn.lower().replace(' ', '_'))
print(f'Список новых имен столбцов: {new_names_colomns}')
print()

# Создаём словарь, где в качестве ключей будет список старых названий столцов, а в качестве значений - список новых названий столбцов
dict_names_colomns = dict(zip(games.columns.tolist(), new_names_colomns))

# Передаём параметру columns словарь с новыми названиями столбцов
games_new = games.rename(columns=dict_names_colomns)
```

```
Список новых имен столбцов: ['name', 'platform', 'year_of_release', 'genre', 'na_sales', 'eu_sales', 'jp_sales', 'other_sales', 'critic_score', 'user_score', 'rating']
```

Проверим результат преобразования названий столбцов и выведем 1 строку датафрейма `games_new`

In [7]:
```
# Датафрейм с новыми названиями столбцов, выведим 1 строку
games_new.head(1)
```

Out[7]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
0	Wii Sports	Wii	2006.0	Sports	41.36	28.96	3.77	8.45	76.0	8	E

2.2 Типы данных

На стадии знакомства было установлено, что 3 столбца `eu_sales`, `jp_sales` и `user_score` датафрейма представлены с некорректными типами данных:

- `eu_sales`: Тип данных определен как `object`. Причиной могут служить наличие текстовых данные в некоторых записях. Согласно описанию, в столбце хранится информация о количестве продаж не в единицах, а в миллионах, т.е. представлена дробным числом. Для возможности выполнения точных вычислений и анализа числовых данных необходимо привести к типу `float64`. Однако предварительно потребуется обработать имеющиеся строковые данные.
- `jp_sales`: Тип данных определен как `object`. Причиной могут служить наличие текстовых данные в некоторых записях. В столбце хранится информация о количестве продаж не в единицах, а в миллионах, т.е. представлена дробным числом. Для возможности выполнения точных вычислений и анализа числовых данных необходимо привести к типу `float64`. Однако предварительно потребуется обработать имеющиеся строковые данные.
- `user_score`: Тип данных определен как `object`. Причиной могут служить наличие текстовых данные в некоторых записях. Из предоставленной информации известно, что оценка пользователей принимает значения от 0 до 10. А знакомство с данными датафрейма показало, что эти значения могут быть в том числе и дробными. Для возможности выполнять вычисления и анализировать числовые данные необходимо привести их к типу `float64`. Кроме того, для оптимизации можно уменьшить разрядность до `float32`. Однако предварительно потребуется обработать имеющиеся строковые данные.

Преобразуем типы данных в столбцах `eu_sales`, `jp_sales`, `user_score` к типу `float` с помощью метода `to_numeric()`, при этом в случае невозможности преобразования строковых данных в числовой тип заменим эти значения на NaN с помощью параметра `errors='coerce'`, а с помощью `downcast='float'` понизим разрядность. Полученные значения округлим до 2 знаков после запятой.

In [8]:
```
# Выводим информацию о количестве непустых строк в столбцах и типы до преобразования
for column in ['eu_sales', 'jp_sales', 'user_score']:
    print(f'Количество непустых строк до преобразования в {column}: {games_new[column].count()}, тип: {games_new[column].dtypes}')

print()

for column in ['eu_sales', 'jp_sales', 'user_score']:
    # Преобразуем в числовой тип
    games_new[column] = pd.to_numeric(games_new[column], errors='coerce', downcast = 'float')
    # Округлим значения до 2 знаков после запятой
    games_new[column] = games_new[column].round(2)
    # Выведим информацию о количестве непустых строк и типе данных после преобразования
    print(f'Количество непустых строк после преобразования в {column}: {games_new[column].count()}, тип: {games_new[column].dtypes}')

print()
```

```
Количество непустых строк до преобразования в eu_sales: 16956, тип: object
Количество непустых строк до преобразования в jp_sales: 16956, тип: object
Количество непустых строк до преобразования в user_score: 10152, тип: object

Количество непустых строк после преобразования в eu_sales: 16950, тип: float32
Количество непустых строк после преобразования в jp_sales: 16952, тип: float32
Количество непустых строк после преобразования в user_score: 7688, тип: float32
```

Понизим разрядность типа данных в столбцах `na_sales`, `other_sales`, `critic_score` с помощью метода `to_numeric()` и параметра `downcast='float'`. Полученные значения округлим до 2 знаков после запятой.

In [9]:
```
# Выводим информацию о количестве непустых строк в столбцах и типы до преобразования
for column in ['na_sales', 'other_sales', 'critic_score']:
    print(f'Количество непустых строк до преобразования в {column}: {games_new[column].count()}, тип: {games_new[column].dtypes}')

print()

for column in ['na_sales', 'other_sales', 'critic_score']:
    games_new[column] = pd.to_numeric(games_new[column], downcast = 'float')
    # Округлим значения до 2 знаков после запятой
    games_new[column] = games_new[column].round(2)
    # Выведим информацию о количестве непустых строк и типе данных после преобразования
    print(f'Количество непустых строк после преобразования в {column}: {games_new[column].count()}, тип: {games_new[column].dtypes}')

print()
```

```
Количество непустых строк до преобразования в na_sales: 16956, тип: float64
Количество непустых строк до преобразования в other_sales: 16956, тип: float64
Количество непустых строк до преобразования в critic_score: 8242, тип: float64

Количество непустых строк после преобразования в na_sales: 16956, тип: float32
Количество непустых строк после преобразования в other_sales: 16956, тип: float32
Количество непустых строк после преобразования в critic_score: 8242, тип: float32
```

Для оптимизации работы с данными в датафрейме были сделаны такие изменения типов данных:

- `eu_sales`, `jp_sales`, `user_score`: тип данных изменён с `object` на `float32`. При этом уменьшилось количество непустых строк в столбцах. Это значит, что некоторые строковые значения не были представленны числовыми знаками и их неудалось преобразовать.
- `na_sales`, `other_sales`, `critic_score`: понижена разрядность с `float64` на `float32`.

2.3 Пропуски в данных

Оценим количество пропусков в датафрейме, для этого выведем общее количество пропусков, а так же количество пропусков в каждом столбце в абсолютной и относительной величине.

In [10]:
```
# Выводим общее количество пропусков в датафрейма
print('Общее количество пропусков в датафрейме:', games_new.isna().sum().sum())
```

```
Общее количество пропусков в датафрейме: 25142
```

In [11]:
```
# Выводим количество пропусков в каждом столбце датафрейма
print('Количество пропусков в каждом столбце датафрейма(абсолютные значения):')
games_new.isna().sum()
```

```
Количество пропусков в каждом столбце датафрейма(абсолютные значения):
```

Out[11]:
```
name                 2
platform             0
year_of_release    275
genre                2
na_sales             0
eu_sales             6
jp_sales             4
other_sales          0
critic_score      8714
user_score        9268
rating            6871
dtype: int64
```

In [12]:
```
# Выводим количество пропусков в относительной величине
# Округлим до 2 знаков после запятой
print('Количество пропусков в каждом столбце датафрейма(относительные значения):')
(games_new.isna().mean() * 100).round(2)
```

```
Количество пропусков в каждом столбце датафрейма(относительные значения):
```

Out[12]:
```
name               0.01
platform           0.00
year_of_release    1.62
genre              0.01
na_sales           0.00
eu_sales           0.04
jp_sales           0.02
other_sales        0.00
critic_score      51.39
user_score        54.66
rating            40.52
dtype: float64
```

Выводы о причинах появления пропусков в столбцах датафрейма, возможных последствиях и способах их обработки

В данных наблюдаются пропуски в 6 столбцах:

Больше всего пропусков 54,66% (9268 пропусков) в столбце с оценками пользователей `User Score`.
Возможные причины пропусков:

- гипотеза о том, что оценки пользователей не были размещены в открытом доступе(или скрыты) маловероятна. Оценки могут быть анонимными, но вряд-ли скрытыми от остальных;
- человеческий фактор - коллеги пропустили/не увидели оценки пользователей или забыли внести данные;
- пользователи могли еще не оценить игру, такое возможно при условии, если не продано ни одной копии игры.

Возможные последствия:

- пропуски могут затруднить анализ и исказить результат исследования на основе оценок пользователей. Для большей части игр невозможно определить нравятся они пользователям или нет.

Большое количество пропусков 51,39% (8714 пропусков) в столбце с оценками критиков `Critic Score`.
Возможные причины пропусков:

- оценки критиков не были размещены в открытом доступе, например, оценка была заказана для внутреннего пользования в компании, или намеренно скрыты, например, если оценка ниже ожидаемой. Затрудняюсь сказать насколько это реально, но оставим эту причину как одну из возможных;
- человеческий фактор - коллеги пропустили/не увидели оценки критиков или забыли внести данные;
- критики ещё не оценили игру, например, игра новая.

Возможные последствия:

- пропуски могут затруднить анализ и исказить результат исследования на основе оценок критиков, например, при изучении ТОП-100 игр по оценкам критиков. Игра не войдет в ТОП, хотя возможно имеет высокую оценку критиков. Таким образом, рейтинг на основе оценок критиков не будет отражать реальную картину.

В столбце рейтинга ESRB `Rating` 40,52% пропусков (6871 пропуск) данных.
Возможные причины пропусков:

- гипотеза о том, что рейтинг не был размещен в открытом доступе маловероятна, поскольку суть рейтинга как раз в общедоступности;
- человеческий фактор - коллеги пропустили/не увидели присвоенный рейтинг или забыли внести данные;
- можно предположить, что игра новая и рейтинг еще не присвоен, однако для таких случаев есть категория RP (Rating Pending) - игра в процессе присвоения рейтинга.

Таким образом, наиболее вероятная причина пропусков - человеческий фактор

Возможные последствия:

- пропуски могут затруднить анализ и исказить результат исследования на основе рейтинга ESRB. Для более чем 40% игр невозможно определить возрастную категорию. Например, корректно собрать данные для рекомендаций и предоставить полную информацию об играх, например, для детей до 13 лет, не представляется возможным.

Рекомендации для пропусков в `User Score`, `Critic Score` и `Rating`:

- определить являются ли пропуски случайными или зависят от других значений датафрейма;
- удалять пропуски нельзя, иначе потеряется значительная часть данных;
- если пропуски зависят от других данных, и гипотеза подтверждается данными, то следует заменить данные на основе этой гипотезы, например, заменить средними значениями в рамках групп, которые связаны с появлением пропусков;
- замена пропусков средними значениями или популярными значениями в разрезе групп может привести к искажению результатов, например, игра реально имеющая оценку 10,0 может ошибочно получить оценку 6,0 или игре для дошкольников может по ошибке получить категорию 18+;
- если восстановить оценки невозможным, можно заменить пропуски значением-индикатором для `User Score` и `Critic Score` значением `-1`, а для `Rating` значением `not_rating`

В столбце с годом выпуска игры `Year of Release` 1,62% пропусков (275 пропусков).
Возможные причины пропусков:

- намеренное скрытие информации о годе выпуска игры не имеет смысла, т.е. эту гипотезу можно не рассматривать;
- можно предположить, что еще не состоялся официальный релиз игры (В этом случаем, для проверки гипотезы стоит обратить внимание на столбцы с продажами. Если они ненулевые, значит релиз состоялся и гипотезу можно отвергнуть);
- наиболее вероятен человеческий фактор - коллеги пропустили/не увидели присвоенный рейтинг или забыли внести данные.

Возможные последствия:

- игра без указанного года выпуска может не попасть в выборку, например, имея высокие показатели объёмов продаж игра не попадёт в ТОП-20 игр 2008 года

В столбцах с продажами в Европе `EU sales` 0,04% пропусков (6 пропусков), в Японии `JP ales` 0,02% пропусков (4 пропуска).
Возможные причины пропусков:

- наиболее вероятная причина пропусков - это отсутствие продаж;
- однако не стоит отвергать и человеческий фактор.

Возможные последствия:

- игра может не попасть в выборки по продажам имея при этом высокие показатели объёмов продаж.

Рекомендации для пропусков в `EU sales`, `JP sales` и `Year of Release`:

- попытаться восстановить данные;
- заменить пропуски значением-индикатором для `EU sales` и `JP sales` значением `-1`, а для `Year of Release` значением `0`

Наименьшее количество пропусков содержат столбцы с названием игры `Name` и её жанром `Genre` по 0,01% (по 2 пропуска).
Возможные причины пропусков:

- игра без названия существовать не может, поэтому наиболее вероятная причина - это человеческий фактор, данные просто забыли внести.

Возможные последствия:

- отсутствие данных затруднит идентификацию игр.

Рекомендации:

- если отсутствующие значения названия и жанра принадлежат одним записям, то удалить эти записи;
- если отсутствует наименование жанра, а название игры есть, то восстановить данные из открытых источников по названию игры;
- если в записи отсутствует название игры, то удалить её запись.

2.3.1 Работа с пропусками в столбце с оценками пользователей `user_score`.

Выведем несколько строк датафрейма, в которых имеются пропуски в столбце `user_score`. Возможно получится проследить зависимость этих пропусков от других данных. Посмотрим на разнообразие данных в столбцах, чтобы оценить распределение пропусков. Примем решение о способе обработки пропусков

In [13]:
```
# Выводим количество пропусков
print('Количество пропусков в столбце user_score до обработки:', games_new['user_score'].isna().sum())
```

```
Количество пропусков в столбце user_score до обработки: 9268
```

In [14]:
```
# Выводим первые 5 строк датфрейма с пропусками в столбце 'user_score'
games_new[games_new['user_score'].isna()].head()
```

Out[14]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
1	Super Mario Bros.	NES	1985.0	Platform	29.080000	3.58	6.81	0.77	NaN	NaN	NaN
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	11.270000	8.89	10.22	1.00	NaN	NaN	NaN
5	Tetris	GB	1989.0	Puzzle	23.200001	2.26	4.22	0.58	NaN	NaN	NaN
9	Duck Hunt	NES	1984.0	Shooter	26.930000	0.63	0.28	0.47	NaN	NaN	NaN
10	Nintendogs	DS	2005.0	Simulation	9.050000	10.95	1.93	2.74	NaN	NaN	NaN

Пропуски в столбце `user_score` появляются в разных строках датафрейма и явной зависимости от других данных не наблюдается. Таким образом, будем считать, что пропуски являюся случайными. Заменим пропуски на индикатор `-1`.

In [15]:
```
# Заменяем каждый пропуск на индикатор '-1'
games_new['user_score'] = games_new['user_score'].fillna(-1)
```

Проверим результат обработки пропусков и выведем количество пропусков в столбце `user_score`

In [16]:
```
# Выводим количество пропусков  столбце 'user_score'
print('Количество пропусков в столбце user_score после обработки:', games_new['user_score'].isna().sum())
```

```
Количество пропусков в столбце user_score после обработки: 0
```

Итак, 9268 пропусков в столбце `user_score` заменены на индикатор `-1`.

2.3.2 Работа с пропусками в столбце с оценками критиков `critic_score`.

Выведем несколько строк датафрейма, в которых имеются пропуски в столбце `critic_score`. Возможно получится проследить зависимость этих пропусков от других данных. Посмотрим на разнообразие данных в столбцах, чтобы оценить распределение пропусков и примем решение о способе их обработки.

In [17]:
```
# Выводим количество пропусков
print('Количество пропусков в столбце critic_score до обработки:', games_new['critic_score'].isna().sum())
```

```
Количество пропусков в столбце critic_score до обработки: 8714
```

In [18]:
```
# Выводим первые 5 строк датфрейма с пропусками в столбце 'critic_score'
games_new[games_new['critic_score'].isna()].head()
```

Out[18]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
1	Super Mario Bros.	NES	1985.0	Platform	29.080000	3.58	6.81	0.77	NaN	-1.0	NaN
4	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	11.270000	8.89	10.22	1.00	NaN	-1.0	NaN
5	Tetris	GB	1989.0	Puzzle	23.200001	2.26	4.22	0.58	NaN	-1.0	NaN
9	Duck Hunt	NES	1984.0	Shooter	26.930000	0.63	0.28	0.47	NaN	-1.0	NaN
10	Nintendogs	DS	2005.0	Simulation	9.050000	10.95	1.93	2.74	NaN	-1.0	NaN

Пропуски в столбце `critic_score` появляются в разных строках датафрейма, зависимости от остальных данных не наблюдается. Таким образом, будем считать, что пропуски являются случайными. Заменим пропуски на индикатор `-1`.

In [19]:
```
# Заменяем каждый пропуск на индикатор '-1'
games_new['critic_score'] = games_new['critic_score'].fillna(-1)
```

Проверим результат обработки пропусков и выведем количество пропусков в столбце `critic_score`

In [20]:
```
# Выводим количество пропусков в столбце 'critic_score' после обработки
print('Количество пропусков в столбце critic_score после обработки:', games_new['critic_score'].isna().sum())
```

```
Количество пропусков в столбце critic_score после обработки: 0
```

Итак, 8714 пропусков в столбце `critic_score` заменены индикатором `-1`

2.3.3 Работа с пропусками в столбце с рейтингом `rating`.

In [21]:
```
# Выводим количество пропусков в столбце 'rating'
print('Количество пропусков в столбце rating до обработки:', games_new['rating'].isna().sum())
```

```
Количество пропусков в столбце rating до обработки: 6871
```

Пропуски в столбце `rating` не возможно корректно заполнить на основе других данных, поэтому заменим пропуски значением `not_rating`

In [22]:
```
# Заменяем пропуски в столбце 'rating' на значение 'not_rating'
games_new['rating'] = games_new['rating'].fillna('not_rating')
```

Проверим результат обработки пропусков и выведим количество пропусков в столбце `rating`

In [23]:
```
# Выводим количество пропусков в столбце 'rating'
print('Количество пропусков в столбце rating после обработки:', games_new['rating'].isna().sum())
```

```
Количество пропусков в столбце rating после обработки: 0
```

Итак, 6871 пропуск в столбце `rating` заменен на индикатор `not_rating`

2.3.4 Работа с пропусками в столбце с годом релиза игр `year_of_release`

Выведем несколько строк датафрейма, в которых имеются пропуски в столбце `year_of_release`. Попробуем проследить зависимость этих пропусков от других данных. Посмотрим на разнообразие данных в столбцах, чтобы оценить распределение пропусков.

In [24]:
```
# Выводим количество пустых строк в столбце 'year_of_release'
print('Количество пропусков в столбце year_of_release до обработки:', games_new['year_of_release'].isna().sum())
```

```
Количество пропусков в столбце year_of_release до обработки: 275
```

In [25]:
```
# Выводим уникальным значениям в столбце 'year_of_release' отсортированные по возрастанию
games_new['year_of_release'].sort_values().unique()
```

Out[25]:
```
array([1980., 1981., 1982., 1983., 1984., 1985., 1986., 1987., 1988.,
       1989., 1990., 1991., 1992., 1993., 1994., 1995., 1996., 1997.,
       1998., 1999., 2000., 2001., 2002., 2003., 2004., 2005., 2006.,
       2007., 2008., 2009., 2010., 2011., 2012., 2013., 2014., 2015.,
       2016.,   nan])
```

In [26]:
```
# Выводим первые 5 строк датфрейма с пропусками в столбце 'year_of_release'
games_new[games_new['year_of_release'].isna()].head()
```

Out[26]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
183	Madden NFL 2004	PS2	NaN	Sports	4.26	0.26	0.01	0.71	94.0	8.5	E
379	FIFA Soccer 2004	PS2	NaN	Sports	0.59	2.36	0.04	0.51	84.0	6.4	E
458	LEGO Batman: The Videogame	Wii	NaN	Action	1.80	0.97	0.00	0.29	74.0	7.9	E10+
477	wwe Smackdown vs. Raw 2006	PS2	NaN	Fighting	1.57	1.02	0.00	0.41	-1.0	-1.0	not_rating
611	Space Invaders	2600	NaN	Shooter	2.36	0.14	0.00	0.03	-1.0	-1.0	not_rating

Столбец `year_of_release` содержит 275 пропусков. Остальные значения представляю собой года от 1980 до 2016. Аномальные значения отсутствуют.

Интересное наблюдение.

В названиях некоторых игр в столбце `name` присутствуют числовые подстроки, которые напоминают номер года. Если эти данные действительно являются годами, то их можно использовать для заполнения пропущенных значений в столбце `year_of_release`.

План:

1. Из каждой строки столбца `name` выделим последние 4 знака.
2. Изменим для полученных подстрок тип данных на числовой. При этом, в случае невозможности изменить тип (то есть в случае, если подстрока не состоит из числовых знаков) оставим строку пустой. Пустые строки заменим нулями.
3. Сохраним полученные данные в новом столбце `year_name`.
4. Оценим корректность значений столбца `year_name`:
   - если числовое значение является годом из интервала 2000-2013, то оставим это значение;
   - если числовое значение не подходит под формат года или выходит за рамки интересующего интервала, то заменим это значение нулём.
5. Заменим пропущенные значения в столбце `year_of_release` на соответствующие значения из столбца `year_name`.
6. Удалим вспомогательные столбец `year_name`

In [27]:
```
# Выделяем из строк с названием игр столбца 'name' подстроку - последние 4 знака , меняем тип на числовой с параметром errors='coerce'.
# Результат сохраняем во временном столбце 'year_name'
games_new['year_name'] = pd.to_numeric(games_new['name'].str.slice(-4), errors='coerce')

# Заменяем пропуски на "0": fillna(0)
games_new['year_name'] = games_new['year_name'].fillna(0)

# Выводим уникальные значения и частоту их вхождения
games_new['year_name'].value_counts()
```

Out[27]:
```
year_name
0.0       16227
2002.0       69
2000.0       58
2004.0       55
2010.0       46
2001.0       46
2003.0       45
2005.0       42
2011.0       41
2009.0       38
2008.0       31
2015.0       31
2014.0       29
2006.0       29
2012.0       29
2016.0       26
2013.0       22
2007.0       20
2017.0       13
3.0           6
14.0          4
2.0           4
2033.0        2
1996.0        2
2025.0        2
1945.0        2
81.0          2
1999.0        2
2100.0        2
1942.0        2
300.0         2
575.0         1
2142.0        1
1.5           1
9.0           1
2.5           1
1946.0        1
121.0         1
100.0         1
1943.0        1
1969.0        1
79.0          1
2070.0        1
1692.0        1
2049.0        1
2020.0        1
2600.0        1
2205.0        1
776.0         1
500.0         1
626.0         1
122.0         1
2048.0        1
101.0         1
7.0           1
215.0         1
1886.0        1
3000.0        1
Name: count, dtype: int64
```

Получили временный/вспомогательный столбец `year_name`. Не все данные этого столбца представляют собой номер года. Однако группа наиболее частовстречающихся значений, очевидно, относится к годам начала 21 века. Оставим период с 2000 по 2013 года, а остальные значения заменяем нулями.

In [28]:
```
# В столбце 'year_name' значения < 2000 и > 2013 заменяем на "0"
games_new['year_name'] = games_new['year_name'].mask(games_new['year_name'] < 2000, 0)
games_new['year_name'] = games_new['year_name'].mask(games_new['year_name'] > 2013, 0)

# Выводим уникальные значения и частоту их вхождения после преобразования
games_new['year_name'].value_counts()
```

Out[28]:
```
year_name
0.0       16385
2002.0       69
2000.0       58
2004.0       55
2010.0       46
2001.0       46
2003.0       45
2005.0       42
2011.0       41
2009.0       38
2008.0       31
2006.0       29
2012.0       29
2013.0       22
2007.0       20
Name: count, dtype: int64
```

В результате получили временный столбец `year_name` содержащий только года от 2000 до 2013 включительно.

Заменим пропущенные значения в столбце `year_of_release` на соответствующие значения из временного столбца `year_name`. И преобразуем тип столбца `year_of_release` к целочисленному.

In [29]:
```
# Заменяем пропущенные значения из столбца 'year_of_release' на соответствующие значения из столбца 'year_name'
games_new['year_of_release'] = games_new['year_of_release'].fillna(games_new['year_name'])
```

In [30]:
```
# Преобразуем 'year_of_release' к целочисленному типу с наименьшим разрядом
games_new['year_of_release'] = pd.to_numeric(games_new['year_of_release'], downcast = 'integer')
```

Проверим результат работы: выведем уникальные значения столбца `year_of_release` и количество пропусков.

In [31]:
```
# выводим уникальные значения в столбце 'year_of_release' отсортированные по возрастанию
games_new['year_of_release'].sort_values().unique()
```

Out[31]:
```
array([   0, 1980, 1981, 1982, 1983, 1984, 1985, 1986, 1987, 1988, 1989,
       1990, 1991, 1992, 1993, 1994, 1995, 1996, 1997, 1998, 1999, 2000,
       2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008, 2009, 2010, 2011,
       2012, 2013, 2014, 2015, 2016], dtype=int16)
```

In [32]:
```
# Выводим количество пропусков в столбце'year_of_release'
print('Количество пропусков в столбце year_of_release после преобразований:', games_new['year_of_release'].isna().sum())
```

```
Количество пропусков в столбце year_of_release после преобразований: 0
```

Удалим вспомогательный столбец `year_name` из датафрейма и проверим удаление, выведя названия всех оставшихся столбцов датафрейма.

In [33]:
```
# Удаляем вспомогательный столбец 'year_name' из датафрейма
del games_new['year_name']
```

In [34]:
```
# Выводим названия столбцов датафрейма
games_new.columns
```

Out[34]:
```
Index(['name', 'platform', 'year_of_release', 'genre', 'na_sales', 'eu_sales',
       'jp_sales', 'other_sales', 'critic_score', 'user_score', 'rating'],
      dtype='object')
```

В результате из 275 пропущенных значений столбца `year_of_release` удалось восстановить 15 значений благодаря данным столбца `name`. Остальные 260 пропусков были заменены нулями. Данные столбца `year_of_release` к целочисленному типу `int16`

2.3.5 Работа с пропусками в столбце с продажами в Европе `eu_sales`

In [35]:
```
# Выводим количество пропусков в столбце eu_sales
print('Количество пропусков в столбце eu_sales до преобразования:', games_new['eu_sales'].isna().sum())
```

```
Количество пропусков в столбце eu_sales до преобразования: 6
```

Проанализируем данные представленные в столбце с продажами игр в Европе. Выведем все строки датафрейма с пропусками в столбце `eu_sales`, их всего 6. Попробуем восстановить данные с помощью информации из открытых источников. В противном случае, заполним пропуски соответствующими средними значениями, рассчитанными для каждого жанра и года.

In [36]:
```
# Выводим все строки датафрейма с пропусками в столбце 'eu_sales'
games_new[games_new['eu_sales'].isna()].head(6)
```

Out[36]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
446	Rhythm Heaven	DS	2008	Misc	0.55	NaN	1.93	0.13	83.0	9.0	E
802	Dead Rising	X360	2006	Action	1.16	NaN	0.08	0.20	85.0	7.6	M
1131	Prince of Persia: Warrior Within	PS2	2004	Action	0.54	NaN	0.00	0.22	83.0	8.5	M
1132	Far Cry 4	XOne	2014	Shooter	0.80	NaN	0.01	0.14	82.0	7.5	M
1394	Sonic Advance 3	GBA	2004	Platform	0.74	NaN	0.08	0.06	79.0	8.4	E
1612	Ratatouille	DS	2007	Action	0.49	NaN	0.00	0.14	-1.0	-1.0	not_rating

По данным сайтов arcadetemple.com и vgchartz.com объёмы продаж игр в Европе:

1. Игра `Rhythm Heaven` 2008 года, платформа DS - 510 тысяч копий

2. Игра `Dead Rising` 2006 года, платформа X360 - 650 тысяч копий.

3. Игра `Prince of Persia Warrior Within` 2004 года, платформа PS2 - 880 тысяч копий.

4. Игра `Far Cry 4` 2014 года, платформа XOne - 0,68 миллионов копий.

5. Игра `Sonic Advance 3` 2004 года, платформа GBA - 520 тысяч копий.

6. Игра `Ratatouille` 2007 года, платформа DS - 620 тысяч копий.

Внесем эти данные в датафрейм. Но предварительно рассчитаем объёмы продаж для этих игр на основе данные датафрейма и сравним их с данными из открытых источников.

In [37]:
```
# Создаём списки с названиями, жанрами, годами релизов и объёмами продаж, полученными из открытых источников
games_name = ['Rhythm Heaven', 'Dead Rising','Prince of Persia Warrior Within', 'Far Cry 4', 'Sonic Advance 3', 'Ratatouille']
games_age = [2008, 2006, 2004, 2014, 2004, 2007]
games_genre = ['Misc', 'Action', 'Action', 'Shooter', 'Platform', 'Action']
games_sales_internet = [0.51, 0.65, 0.88, 0.68, 0.52, 0.62]
```

In [38]:
```
# Создаём пустой список для рассчётных значений
games_sales_calc = []
```

In [39]:
```
# Рассчитываем средние значения объёмов продаж для каждого жанра и года релиза.
# Результат сохраняем в 'games_sales'
games_sales = games_new.groupby(['genre', 'year_of_release'])['eu_sales'].mean().reset_index()
```

In [40]:
```
# Для каждой игры находим соответствующее значение в 'games_sales' и сохраняем в новом списке 'games_sales'
for i in range(6):
    k = 0
    k = games_sales[(games_sales['genre'] == games_genre[i]) & (games_sales['year_of_release'] == games_age[i])].iloc[0, 2]
    # Округляем до 2 знаков после запятой
    games_sales_calc.append(k.round(2))

# Выводим найденные значения объёмов продаж
games_sales_calc
```

Out[40]:
```
[0.1, 0.08, 0.12, 0.53, 0.17, 0.12]
```

Для удобства сравнения сохраним исходные данные `games_name`, `games_age`, `games_genre`, `games_sales_internet` и полученные данные `games_sales_calc` в датафрей `EU`. Добавим столбец `delta_%`, показывающий на сколько процентов средние значения меньше данных из открытых источников.

In [41]:
```
# Создаем датафрейм 'EU'
EU = pd.DataFrame({'games_name': games_name,
                   'games_age': games_age,
                   'games_genre': games_genre,
                   'games_sales_internet': games_sales_internet,
                   'games_sales_calc': games_sales_calc})

# Добавляем столбец с расчётами разницы в процентах
EU['delta_%'] = ((1 - EU['games_sales_calc'] / EU['games_sales_internet']) * 100).round(2)

# Выводим полученный датафрейм
EU
```

Out[41]:

	games_name	games_age	games_genre	games_sales_internet	games_sales_calc	delta_%
0	Rhythm Heaven	2008	Misc	0.51	0.10	80.39
1	Dead Rising	2006	Action	0.65	0.08	87.69
2	Prince of Persia Warrior Within	2004	Action	0.88	0.12	86.36
3	Far Cry 4	2014	Shooter	0.68	0.53	22.06
4	Sonic Advance 3	2004	Platform	0.52	0.17	67.31
5	Ratatouille	2007	Action	0.62	0.12	80.65

Средние значения, полученные на основе данных датафрейма, в основном значительно ниже (ниже на 67%-87%), чем реальные данные. Исключение составляет игра `Far Cry 4`, для которой погрешность сотавляет 22%. Все эти игры входят в интересующий нас интервал 2000-2013гг, то есть будут использоваться для анализа. Таким образом, внося реальные значения объёмов продаж в датафреём мы исключаем возможные ошибки при анализе данных.

Внесём реальные значения объёмов продаж в датафрейм `games_new`: получим индексы строк с играми и заменим эти строки значениями из списка `games_sales_interne`

In [42]:
```
# Список значений объёмов продаж
games_sales_internet = [0.51, 0.65, 0.88, 0.68, 0.52, 0.62]

# Выводим индексы строк с пропусками в столбце 'eu_sales', преобразовав их в список
eu_index = games_new[games_new['eu_sales'].isna()].index.to_list()
eu_index
```

Out[42]:
```
[446, 802, 1131, 1132, 1394, 1612]
```

In [43]:
```
# Заменяем пропуски в столбце 'eu_sales' на значения списка games_sales_internet
games_new.loc[eu_index, 'eu_sales'] = games_sales_internet
```

Проверим результат работы после внесения данных в датафрей. Убедимся, что пропусков в столбце больше нет

In [44]:
```
# Выводим количество пропусков после преобразований
print('Количество пропусков в столбце eu_sales после преобразования:', games_new['eu_sales'].isna().sum())
```

```
Количество пропусков в столбце eu_sales после преобразования: 0
```

Итак, в результате 6 пропущенных значений столбца `eu_sales` были восстановлены.

Соберем небольшую статистику значений столбца `eu_sales`

In [45]:
```
# Выводим количество нулевых значений столбца 'eu_sales'
games_new['eu_sales'][games_new['eu_sales'] == 0].count()
```

Out[45]: 5950

In [46]:
```
# Выводим процент нулевых значений столбца 'eu_sales'
games_new['eu_sales'][games_new['eu_sales'] == 0].count() / games_new['eu_sales'].count() * 100
```

Out[46]: 35.090823307383815

In [47]:
```
# Разбиваем данные (исключая нулевые значения) на 3 интервала и выводим процентное соотношение
games_new['eu_sales'][games_new['eu_sales'] != 0].value_counts(normalize=True, bins=3)
```

Out[47]:
```
(-0.0199, 9.66]    0.999637
(9.66, 19.31]      0.000273
(19.31, 28.96]     0.000091
Name: proportion, dtype: float64
```

In [48]:
```
# Получаем статистику по столбцу 'eu_sales'
games_new['eu_sales'].describe()
```

Out[48]:
```
count    16956.000000
mean         0.144291
std          0.500365
min          0.000000
25%          0.000000
50%          0.020000
75%          0.110000
max         28.959999
Name: eu_sales, dtype: float64
```

В столбце `eu_sales` больше трети значений представлены нулями, а именно 35,1%. Значит 5950 игр не продавались в Европе. Значительная часть игр 99,96% имеет объём продаж не превышающий 9.66 миллионов копий. Максимальное количество продаж 28,96 миллионов копий.

2.3.6 Работа с пропусками в столбце с продажами в Японии `jp_sales`

In [49]:
```
# Выводим количество пропусков в столбце 'jp_sales'
print('Количество пропусков в столбце jp_sales до преобразования:', games_new['jp_sales'].isna().sum())
```

```
Количество пропусков в столбце jp_sales до преобразования: 4
```

Проанализируем данные представленные в столбце `jp_sales` с продажами игр в Японии. Выведем все строки датафрейма с пропусками в столбце `jp_sales`, их всего 4. Попробуем восстановить данные с помощью информации из открытых источников. В противном случае, заполним пропуски соответствующими средними значениями, рассчитанными для каждого жанра и года.

In [50]:
```
# Выводим все строки датафрейма с пропусками в столбце 'jp_sales'
games_new[games_new['jp_sales'].isna()].head()
```

Out[50]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
467	Saints Row 2	X360	2008	Action	1.94	0.79	NaN	0.28	81.0	8.1	M
819	UFC 2009 Undisputed	X360	2009	Fighting	1.48	0.39	NaN	0.19	83.0	7.9	T
1379	Hello Kitty Party	DS	2007	Misc	0.78	0.51	NaN	0.12	-1.0	-1.0	E
4732	Castlevania: The Dracula X Chronicles	PSP	2007	Platform	0.22	0.09	NaN	0.07	80.0	7.8	T

По данным сайта arcadetemple.com объёмы продаж игр в Японии:

1. Игра `Saints Row 2` 2008 года, платформа X360 - всего 20 тысяч копий
2. Игра `UFC 2009 Undisputed` 2009 года, платформа X360 - 0 копий
3. Игра `Hello Kitty Party` 2007 года, платформа DS - 0 копий
4. Игра `Castlevania: The Dracula X Chronicles` 2007 года, платформа PSP - 40 тысяч копий

Внесем эти данные в датафрейм. Предварительно рассчитаем объёмы продаж для этих игр на основе данные датафрейма и сравним их с данными из открытых источников.

In [51]:
```
# Создаём списки с названиями, жанрами, годами релизов и объёмами продаж, полученными из открытых источников
games_name = ['Saints Row 2', 'UFC 2009 Undisputed','Hello Kitty Party', 'Castlevania: The Dracula X Chronicles']
games_age = [2008, 2009, 2007, 2007]
games_genre = ['Action', 'Fighting', 'Misc', 'Platform']
games_sales_internet = [0.02, 0, 0, 0.04]
```

In [52]:
```
# Создаём пустой список для рассчётных значений
games_sales_calc = []
```

In [53]:
```
# Рассчитываем средние значения объёмов продаж для каждого жанра и года релиза.
# Результат сохраняем в games_sales
games_sales = games_new.groupby(['genre', 'year_of_release'])['jp_sales'].mean().reset_index()
```

In [54]:
```
# Для каждой игры находим соответствующее значение в 'games_sales' и сохраняем в новом списке 'games_sales'
for i in range(4):
    k = 0
    k = games_sales[(games_sales['genre'] == games_genre[i]) & (games_sales['year_of_release'] == games_age[i])].iloc[0, 2]
    # Округляем до 2 знаков после запятой
    games_sales_calc.append(k.round(2))

# Выводим найденные значения объёмов продаж
games_sales_calc
```

Out[54]: [0.03, 0.06, 0.08, 0.07]

Для удобства сравнения сохраним исходные данные `games_name`, `games_age`, `games_genre`, `games_sales_internet` и полученные данные `games_sales_calc` в датафрей `JP`. Добавим столбец `delta_%`, показывающий на сколько процентов средние значения больше данных из открытых источников

In [55]:
```
# Создаём датафрейм 'JP'
JP = pd.DataFrame({'games_name': games_name,
                   'games_age': games_age,
                   'games_genre': games_genre,
                   'games_sales_internet': games_sales_internet,
                   'games_sales_calc': games_sales_calc})
```

```
# Добавляем столбец с расчётами разницы в процентах
JP['delta_%'] = ((1- JP['games_sales_internet'] / JP['games_sales_calc']) * 100).round(2)
JP
```

Out[55]:

	games_name	games_age	games_genre	games_sales_internet	games_sales_calc	delta_%
0	Saints Row 2	2008	Action	0.02	0.03	33.33
1	UFC 2009 Undisputed	2009	Fighting	0.00	0.06	100.00
2	Hello Kitty Party	2007	Misc	0.00	0.08	100.00
3	Castlevania: The Dracula X Chronicles	2007	Platform	0.04	0.07	42.86

Средние значения, полученные на основе данных датафрейма, выше на 33%-43%, чем реальные данные. Для двух игр, в случае использования расчётных данных, мы бы совершили ошибку, внеся количество продаж при их реальном отсутствии. Все эти игры входят в интересующий нас интервал 2000-2013гг, то есть будут использоваться для анализа. Таким образом, внося реальные значения объёмов продаж в датафреём мы исключаем возможные ошибки при анализе данных.

Внесём реальные значения объёмов продаж в датафрейм `games_new`: получим индексы строк с играми и заменим эти строки значениями из списка `games_sales_interne`

In [56]:
```
# Получаем индексы строк с пропусками в столбце 'jp_sales', преобразовав их в список
jp_index = games_new[games_new['jp_sales'].isna()].index.to_list()
jp_index
```

Out[56]: `[467, 819, 1379, 4732]`

In [57]:
```
# Заменяем пропуски в столбце 'jp_sales' на значения списка games_sales_internet
games_new.loc[jp_index, 'jp_sales'] = games_sales_internet
```

Проверим результат работы после внесения данных в датафрей. Убедимся, что пропусков в столбце больше нет

In [58]:
```
# Выводим количество пропусков в столбце 'jp_sales' после преобразований
print('Количество пропусков в столбце jp_sales после преобразования:', games_new['jp_sales'].isna().sum())
```

```
Количество пропусков в столбце jp_sales после преобразования: 0
```

Итак, в результате 4 пропущенные значения столбца `jp_sales` были восстановлены.

Соберем небольшую статистику значений столбца `jp_sales`

In [59]:
```
# Выводим процент нулевых значений столбца 'jp_sales'
games_new['jp_sales'][games_new['jp_sales'] == 0].count() / games_new['jp_sales'].count() * 100
```

Out[59]: `62.99834866713847`

In [60]:
```
# Выводим количество нулевых значений столбца 'jp_sales'
games_new['jp_sales'][games_new['jp_sales'] == 0].count()
```

Out[60]: `10682`

In [61]:
```
# Количество значений в интервалах без учёта нулей
games_new['jp_sales'][games_new['jp_sales'] != 0].value_counts(normalize=True, bins=3)
```

Out[61]:
```
(-0.0012100000000000001, 3.413]    0.994900
(3.413, 6.817]                     0.004782
(6.817, 10.22]                     0.000319
Name: proportion, dtype: float64
```

In [62]:
```
# Получаем статистику по столбцу 'jp_sales'
games_new['jp_sales'].describe()
```

Out[62]:
```
count    16956.000000
mean         0.077156
std          0.307139
min          0.000000
25%          0.000000
50%          0.000000
75%          0.040000
max         10.220000
Name: jp_sales, dtype: float64
```

В столбце `jp_sales` почти 63% значений представлены нулями, то есть 10682 игр из датафрейма не продавались в Японии. Из тех игр, которые были проданы в Японии 99,5% имеет объём продаж не превышающий 3.413 миллионов копий. Максимальное количество продаж 10,22 миллионов копий.

2.3.7 Работа с пропусками в столбце с названием игры `Name`

In [63]:
```
# Выводим количество пропусков в столбце 'name'
print('Количество пропусков в столбце name до преобразования:', games_new['name'].isna().sum())
```

```
Количество пропусков в столбце name до преобразования: 2
```

Выведем строки с пропусками в названии игр и решим, что с ними делать.

In [64]:
```
# Выводим все строки датафрейма с пропусками в столбце 'name'
games_new[games_new['name'].isna()].head()
```

Out[64]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
661	NaN	GEN	1993	NaN	1.78	0.53	0.00	0.08	-1.0	-1.0	not_rating
14439	NaN	GEN	1993	NaN	0.00	0.00	0.03	0.00	-1.0	-1.0	not_rating

Кроме пропущенных названий игр, в этих строках также отсутствую наименования жанров, по ним нет оценок пользователей и критиков, они не имеют рейтинга. Кроме того, эти игры относятся к 1993 году, который мы не будем рассматривать при анализе. Удалим эти строки из датафрейма.

In [65]:
```
# Удаляем строки датафрейма с пропусками в столбце 'name'
games_new = games_new.dropna(subset=['name'])
```

In [66]:
```
# Выводим количество пропусков в столбце 'name'
```

```
print('Количество пропусков в столбце name после удаления:', games_new['name'].isna().sum())
```

```
Количество пропусков в столбце name после удаления: 0
```

Итак, 2 пропуска в столбце `name` были удалены.

2.3.8 Работа с пропусками в столбце с жанрами `Genre`

В столбце `genre` с названием жанра в первоначальном датафрейме существовало 2 пропуска. Они находились в тех же строках, что и пропуски названий игр, которые мы удалили на предыдущем этапе. Проверим, что в столбце `genre` больше нет пропусков.

In [67]:
```
# Выводим количество пропусков в столбце 'genre'
print('Количество пропусков в столбце genre:', games_new['genre'].isna().sum())
```

```
Количество пропусков в столбце genre: 0
```

2.4 Явные и неявные дубликаты

2.4.1 Неявные дубликаты

Посмотрим статистику столбцов, содержащих строковые данные: `name`, `platform`, `genre`, `rating`

In [68]:
```
# Получаем статистику по столбцу
games_new[['name', 'platform', 'genre', 'rating']].describe()
```

Out[68]:

	name	platform	genre	rating
count	16954	16954	16954	16954
unique	11559	31	24	9
top	Need for Speed: Most Wanted	PS2	Action	not_rating
freq	12	2189	3405	6869

- Столбец `name` содержит 11559 уникальных значений из 16954
- Столбец `platform` содержит 31 уникальное значение.
- Столбец `genre` содержит 24 уникальные значения
- Столбец `rating` содержит 9 уникальных значений

Для выявления неявных дубликатов выполним ряд действий:

1. Переведём названия игр, платформ, жанров и категорий к одному регистру:
   - `name`, `platform` и `rating` к верхнему регистру;
   - `genre` к нижниму регистру.
2. Удалим возможные пробелы в начале и в конце строк.
3. Проверим корректность уникальных значений.
4. Проверим, не появились ли после преобразования явные дубликаты.

In [69]:
```
# Приводим к верхнему регистру данные столбцов 'name', 'platform', 'rating'
# Удаляем пробелы в начале и в конце строки
for col in ['name', 'platform', 'rating']:
    games_new[col] = games_new[col].str.upper().str.strip()
```

In [70]:
```
# Приводим к нижнему регистру данные столбца 'genre'
# Удаляем пробелы в начале и в конце строки
games_new['genre'] = games_new['genre'].str.lower().str.strip()
```

Проверим результат работы и повторно получим статистику:

In [71]:
```
# Получаем статистику по столбцу
games_new[['name', 'platform', 'genre', 'rating']].describe()
```

Out[71]:

	name	platform	genre	rating
count	16954	16954	16954	16954
unique	11559	31	12	9
top	NEED FOR SPEED: MOST WANTED	PS2	action	NOT_RATING
freq	12	2189	3418	6869

В результате преобразований:

- количество уникальных значений в столбце `name` не изменилось и осталось равное 11559;
- количество уникальных значений в столбце `genre` изменилось с 24 на 12;
- количество уникальных значений в столбце `platform` не изменилось и осталось равное 31;
- количество уникальных значений в столбце `rating` не изменилось и осталось равное 9.

Убедимся, что уникальные значения из `platform`, `genre`, `rating` не являются неявными дубликатами, для этого выведем списки уникальных значений для каждого их указанных столбцов.

In [72]:
```
# Выводим уникальные значения из 'platform'
games_new['platform'].unique()
```

Out[72]:
```
array(['WII', 'NES', 'GB', 'DS', 'X360', 'PS3', 'PS2', 'SNES', 'GBA',
       'PS4', '3DS', 'N64', 'PS', 'XB', 'PC', '2600', 'PSP', 'XONE',
       'WIIU', 'GC', 'GEN', 'DC', 'PSV', 'SAT', 'SCD', 'WS', 'NG', 'TG16',
       '3DO', 'GG', 'PCFX'], dtype=object)
```

In [73]:
```
# Выводим уникальные значения из 'genre'
games_new['genre'].unique()
```

Out[73]:
```
array(['sports', 'platform', 'racing', 'role-playing', 'puzzle', 'misc',
       'shooter', 'simulation', 'action', 'fighting', 'adventure',
       'strategy'], dtype=object)
```

In [74]:
```
# Выводим уникальные значения из 'rating'
games_new['rating'].unique()
```

Out[74]:
```
array(['E', 'NOT_RATING', 'M', 'T', 'E10+', 'K-A', 'AO', 'EC', 'RP'],
      dtype=object)
```

Значения из `platform`, `genre` и `rating` корректны и действительно уникальны.

2.4.2 Явные дубликаты

Проверим наличие явных дубликатов в датафрейме `games_new` и при необходимости удалим их.

In [75]:
```
# Выводим количество явных дубликатов
games_new.duplicated().sum()
```

Out[75]:
```
241
```

In [76]:
```
# Удаляем явные дубликаты
games_new.drop_duplicates(inplace=True)
```

In [77]:
```
# Выводим количество строк в датафрейме
games_new.shape[0]
```

Out[77]:
```
16713
```

Осталось определить возможные причины большого количества дубликатов (5213 дубликатов) в столбце `name`:

- игра имеет несколько релизов;
- игра выпущена несколькими платформами;
- игре по ошибке указали различные жанры;
- и т.д.

Отфильтруем строки датафрейма и оставим данные по самой частовстречающейся игре `NEED FOR SPEED: MOST WANTED`, она встречаетсся 12 раз.

In [78]:
```
# Фильтруем строки по значению в столбце 'name'
# Сортируем по 'platform', 'year_of_release', 'genre'
games_new[games_new['name'] == 'NEED FOR SPEED: MOST WANTED'].sort_values(by=[ 'year_of_release', 'platform', 'genre'])
```

Out[78]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
6493	NEED FOR SPEED: MOST WANTED	DS	2005	racing	0.24	0.01	0.00	0.02	45.0	6.1	E
6557	NEED FOR SPEED: MOST WANTED	GBA	2005	racing	0.19	0.07	0.00	0.00	-1.0	8.3	E
3619	NEED FOR SPEED: MOST WANTED	GC	2005	racing	0.43	0.11	0.00	0.02	80.0	9.1	T
6051	NEED FOR SPEED: MOST WANTED	PC	2005	racing	0.02	0.23	0.00	0.04	82.0	8.5	T
253	NEED FOR SPEED: MOST WANTED	PS2	2005	racing	2.03	1.79	0.08	0.47	82.0	9.1	T
1605	NEED FOR SPEED: MOST WANTED	X360	2005	racing	1.00	0.13	0.02	0.10	83.0	8.5	T
2017	NEED FOR SPEED: MOST WANTED	XB	2005	racing	0.53	0.46	0.00	0.05	83.0	8.8	T
11873	NEED FOR SPEED: MOST WANTED	PC	2012	racing	0.00	0.06	0.00	0.02	82.0	8.5	T
525	NEED FOR SPEED: MOST WANTED	PS3	2012	racing	0.71	1.46	0.06	0.58	-1.0	-1.0	NOT_RATING
2067	NEED FOR SPEED: MOST WANTED	PSV	2012	racing	0.33	0.45	0.01	0.22	-1.0	-1.0	NOT_RATING
1197	NEED FOR SPEED: MOST WANTED	X360	2012	racing	0.62	0.78	0.01	0.15	83.0	8.5	T
6356	NEED FOR SPEED: MOST WANTED	WIIU	2013	racing	0.13	0.12	0.00	0.02	-1.0	-1.0	NOT_RATING

В 2005 году игра `NEED FOR SPEED: MOST WANTED` вышла на 7 различных платформах, а в 2012 на 5 платформах. Для каждого года и каждой платформы игра имеет свои объёмы продаж, оценки.

Предположим, что записи могут быть неявными дубликатами в случае, если строки совпадают по 4 столбцам: `name`, `platform`, `genre` и `rating`. Выгрузим количество таких дубликатов, и если они есть посмотрим на эти строки.

In [79]:
```
# Определяем количество дубликатов в датафрейме по 4 столбцам
games_new.duplicated(subset=['name', 'year_of_release', 'platform', 'genre']).sum()
```

Out[79]:
```
1
```

Игра `MADDEN NFL 13` имеет 1 неявный дубликат. При этом строка с индексом `606` имеет более полные данные: указаны объёмы продаж в Америке 2.11 миллионов копий и объёмы продаж в других странах 0,23 миллиона копий. Удалим вторую строку с индексом `16465`

In [80]:
```
# Выводим строки с дубликатами по 4 столбцам
# при этом выведим все повторяющиеся записи keep=False
# для удобства отсортирум столбцы
games_new[games_new.duplicated(subset=['name', 'year_of_release', 'platform', 'genre'], keep=False)].sort_values(by=['name', 'year_of_release', 'platform', 'genre'])
```

Out[80]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
606	MADDEN NFL 13	PS3	2012	sports	2.11	0.22	0.0	0.23	83.0	5.5	E
16465	MADDEN NFL 13	PS3	2012	sports	0.00	0.01	0.0	0.00	83.0	5.5	E

In [81]:
```
# Удаляем неявный дубликат в исходном датафрейме
games_new.drop_duplicates(subset=['name', 'year_of_release', 'platform', 'genre'], keep='first', inplace=True)
```

In [82]:
```
# Выводим количество строк в датафрейме
games_new.shape[0]
```

Out[82]:
```
16712
```

Все дубликаты удалены. Осталось оптимизировать данные из `platform`, `genre` и `rating` и заменить их тип на `category`

In [83]:
```
# Меняем тип на 'category'
for column in ['platform', 'genre', 'rating']:
    games_new[column] = games_new[column].astype('category')
    print(f'Тип {column}: {games_new[column].dtypes}')

print()
```

```
Тип platform: category
Тип genre: category
Тип rating: category
```

In [84]:
```
# Выводим информацию о датафрейме
games_new.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 16712 entries, 0 to 16955
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype   
---  ------           --------------  -----   
 0   name             16712 non-null  object  
 1   platform         16712 non-null  category
 2   year_of_release  16712 non-null  int16   
 3   genre            16712 non-null  category
 4   na_sales         16712 non-null  float32 
 5   eu_sales         16712 non-null  float32 
 6   jp_sales         16712 non-null  float32 
 7   other_sales      16712 non-null  float32 
 8   critic_score     16712 non-null  float32 
 9   user_score       16712 non-null  float32 
 10  rating           16712 non-null  category
dtypes: category(3), float32(6), int16(1), object(1)
memory usage: 736.5+ KB
```

Выводы по разделу 2:

На этапе проверки ошибок в данных и их предобработке были проведены следующие действия:

1. Проведена нормализация названий столбцов датафрейма `games_new`.
2. Три столбца с некорректными типами данных `eu_sales`, `jp_sales` `user_score` приведены к типу float32, а для столбцов `na_sales`, `other_sales`, `critic_score` понижена разрядность.
3. Обработано 25142 пропуска из 8 столбцов:
   - пропуски из столбцов `user_score`, `critic_score` заменены на индикатор `-1`
   - пропуски в столбце `rating` заменены на индикатор `not_rating`
   - из 275 пропусков в столбце `year_name` - 15 восстановлено, 260 заменено индикатором `0`
   - 6 пропусков с продажами в столбце `eu_sales` восстановлены из открытых источников
   - 4 пропуска с продажами в столбце `jp_sales` восстановлены из открытых источников
   - 2 пропуска в столбце `name` удалены вместе с 2 пропусками в столбце `genre`
4. Удалено 182 явных дубликат. После приведения данных к одному регистру, удалено еще 59 дубликатов. А также удален 1 неявный дубликат с неполными данными.

Итого, в процессе преобразования количество строк датафрейма уменьшилось с 16956 до 16712, то есть на 244 строки.

3. Фильтрация данных

Для изучения истории продаж игр в начале XXI века, отберём данные за период с 2000 по 2013 год включительно. Сохраним новый срез данных в отдельном датафрейме `games_actual`

In [85]:
```
# Извлекаем из датафрейма 'games_new' строки с годами релизов от 2000 до 2023 вкл
# Результат сохраняем в 'games_actual'
games_actual = games_new[(games_new['year_of_release'] >= 2000) & (games_new['year_of_release'] <= 2013)]
```

In [86]:
```
# Выводим информацию о новом датафрейме 'games_actual'
games_actual.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 12795 entries, 0 to 16954
Data columns (total 11 columns):
 #   Column           Non-Null Count  Dtype   
---  ------           --------------  -----   
 0   name             12795 non-null  object  
 1   platform         12795 non-null  category
 2   year_of_release  12795 non-null  int16   
 3   genre            12795 non-null  category
 4   na_sales         12795 non-null  float32 
 5   eu_sales         12795 non-null  float32 
 6   jp_sales         12795 non-null  float32 
 7   other_sales      12795 non-null  float32 
 8   critic_score     12795 non-null  float32 
 9   user_score       12795 non-null  float32 
 10  rating           12795 non-null  category
dtypes: category(3), float32(6), int16(1), object(1)
memory usage: 564.3+ KB
```

Сбросим индексы для получения последовательных числовых индексов. Добавим столбец с общим объёмом продаж `total_sales` по каждой игре.

In [87]:
```
# Сбрасываем индексы 'games_actual' с парметром drop=True
games_actual = games_actual.reset_index(drop=True)
```

In [88]:
```
# Добавляем новый столбец с общим объёмом продаж
games_actual['total_sales'] = games_actual['na_sales'] + games_actual['eu_sales'] + games_actual['jp_sales'] + games_actual['other_sales']
```

In [89]:
```
# Выводим статистические данные по датафрейму 'games_actual'
games_actual.describe()
```

Out[89]:

	year_of_release	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	total_sales
count	12795.000000	12795.000000	12795.000000	12795.000000	12795.000000	12795.000000	12795.000000	12795.000000
mean	2007.103791	0.254328	0.142205	0.056492	0.050679	38.006020	3.149472	0.503705
std	3.443759	0.741345	0.519153	0.254193	0.204670	36.054508	4.221810	1.509229
min	2000.000000	0.000000	0.000000	0.000000	0.000000	-1.000000	-1.000000	0.000000
25%	2004.000000	0.010000	0.000000	0.000000	0.000000	-1.000000	-1.000000	0.060000
50%	2008.000000	0.090000	0.020000	0.000000	0.010000	50.000000	2.800000	0.160000
75%	2010.000000	0.240000	0.110000	0.020000	0.040000	72.000000	7.500000	0.450000
max	2013.000000	41.360001	28.959999	6.500000	10.570000	98.000000	9.700000	82.539993

In [90]:
```
# Выводим количество уникальных значений в каждом столбце датафрейма
games_actual.nunique()
```

Out[90]:
```
name               8826
platform             20
year_of_release      14
genre                12
na_sales            347
eu_sales            270
jp_sales            174
other_sales         150
critic_score         82
user_score           96
rating                8
total_sales         845
dtype: int64
```

В результате фильтрации датафрейма `games_new` было потеряно 3917 строк. Определим какие данные в процессе фильтрации мы потеряли. Например, количество уникальных платформ `platform` в датафрейме `games_new` было 31, а в новом датафрейме `games_actual` их 20; уникальных рейтингов в датафрейме `games_new` было 9, в новом датафрейме `games_actual` их 8. А вот количество жанров `genre` не изменилось, их по-прежнему 12.

Узнаем, какие уникальные значения столбцов `rating` и `platform` были в датафрейма `games_new`, но отсутствуют в датафрейме `games_actual`. Для этого найдем разность множеств уникальных значений столбцов `rating` и `platform` двух датафреймов.

In [91]:
```
# Находим разность двух множеств, состоящих из уникальных значений столбцов rating в games_new и games_actual
set(games_new['rating'].unique()) - set(games_actual['rating'].unique())
```

Out[91]:
```
{'K-A'}
```

Столбец `rating` датафрейма `games_actual` не содержит значение рейтинга `K-A (Kids to Adults)`. Это вполне ожидаемо, поскольку этот рейтинг действовал до 1998 года, а позже ему на смену пришел рейтинг `E (Everyone)`. В процессе фильтрации, все года до 2000 мы отфильтровали , в том числе и записи с этим рейтингом.

In [92]:
```
# Находим разность двух множеств, состоящих из уникальных значений столбцов platform в games_new и games_actual
set(games_new['platform'].unique()) - set(games_actual['platform'].unique())
```

Out[92]:
```
{'2600', '3DO', 'GEN', 'GG', 'NES', 'NG', 'PCFX', 'SAT', 'SCD', 'SNES', 'TG16'}
```

Столбец `platform` датафрейма `games_actual` не содержит 11 наименований игровых платформ. Все эти платформы были популярны до начала 21 века. Естественно, в процессе фильтрации датафрейма записи с играми и этими платформами были отфильтрованы.

По данным сайта wikipedia.org периоды жизни указанных платформ следующие:

1. `Atari 2600 (2600)` 1977-1992
2. `Interactive Multiplayer (3DO)` 1993-1993
3. `Sega Genesis (GEN)` 1988-1999
4. `Sega Game Gear (GG)` 1990-1997
5. `Nintendo Entertainment System (NES)` 1985-2003
6. `Neo Geo (NG)` 1990-1997
7. `PC-FX` 1994-1998
8. `Sega Saturn (SAT)` 1994-2000
9. `Super Nintendo (SCD)` 1992-2003
10. `Super Nintendo Entertainment System (SNET)` 1990-2003
11. `TurboGrafx-16 (TG16)` 1987- 1994

Выведем уникальные значения столбцов `platform` и `rating` датафрейма `games_actual`:

In [93]:
```
# Выводим множество уникальных значений platform в games_actual
set(games_actual['platform'].unique())
```

Out[93]:
```
{'3DS',
 'DC',
 'DS',
 'GB',
 'GBA',
 'GC',
 'N64',
 'PC',
 'PS',
 'PS2',
 'PS3',
 'PS4',
 'PSP',
 'PSV',
 'WII',
 'WIIU',
 'WS',
 'X360',
 'XB',
 'XONE'}
```

In [94]:
```
# Выводим множестов уникальных значений platform в games_actual
set(games_actual['genre'].unique())
```

Out[94]:
```
{'action',
 'adventure',
 'fighting',
 'misc',
 'platform',
 'puzzle',
 'racing',
 'role-playing',
 'shooter',
 'simulation',
 'sports',
 'strategy'}
```

Определим минимальные значения оценок пользователей `user_score` и критиков `critic_score` датафрейма `games_actual`, при этом не будем учитывать значения индикаторов `-1`:

In [95]:
```
# Находим минимальное значение оценки критиков
games_new['critic_score'][games_new['critic_score'] != -1].min()
```

Out[95]:
```
13.0
```

In [96]:
```
# Находим минимальное значение оценки пользователей
games_new['user_score'][games_new['user_score'] != -1].min()
```

Out[96]:
```
0.0
```

In [97]:
```
# Выводим информацию о новом датафрейме 'games_actual'
games_actual.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12795 entries, 0 to 12794
Data columns (total 12 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   name             12795 non-null  object
 1   platform         12795 non-null  category
 2   year_of_release  12795 non-null  int16
 3   genre            12795 non-null  category
 4   na_sales         12795 non-null  float32
 5   eu_sales         12795 non-null  float32
 6   jp_sales         12795 non-null  float32
 7   other_sales      12795 non-null  float32
 8   critic_score     12795 non-null  float32
 9   user_score       12795 non-null  float32
 10  rating           12795 non-null  category
 11  total_sales      12795 non-null  float32
dtypes: category(3), float32(7), int16(1), object(1)
memory usage: 514.5+ KB
```

Итак, мы получили новый датафрейм `games_actual`, который содержит информацию об играх вышедших в начале 21 века.

- 12795 строк без пропусков;
- 12 столбцов с корректными типами данных: `category`, `float32`, `int16`, `object`;
- столбец `name` содержит 8826 уникальных значений;
- столбец `platform` содержит 20 уникальных значений: `3DS`, `DC`, `DS`, `GB`, `GBA`, `GC`, `N64`, `PC`, `PS`, `PS2`, `PS3`, `PS4`, `PSP`, `PSV`, `WII`, `WIIU`, `WS`, `X360`, `XB`, `XONE`;
- столбец `year_of_release` содержит 14 уникальных значений, представляющих года с 2000 по 2013;
- столбец `genre` содержит 12 уникальных значений жанров: `sports`, `platform`, `racing`, `role-playing`, `puzzle`, `misc`, `shooter`, `simulation`, `action`, `fighting`, `adventure`, `strategy`;
- столбец `na_sales` содержит значения объёмов продаж игр в Северной Америке от 0 до 41,36 миллионов копий;
- столбец `eu_sales` содержит значения объёмов продаж игр в Европе от 0 до 28,96 миллионов копий;
- столбец `jp_sales` содержит значения объёмов продаж игр в Японии от 0 до 6,5 миллионов копий;
- столбец `other_sales` содержит значения объёмов продаж игр в остальных странах от 0 до 10,57 миллионов копий;
- столбец `critic_score` содержит оценки критиков от 13 до 98, значение `-1` - является индикатором и означает отсутствие оценки
- столбец `user_score` содержит оценки пользователей от 0 до 9,7, значение `-1` - является индикатором и означает отсутствие оценки
- столбец `rating` содержит 8 уникальных значений: `E`, `NOT_RATING`, `M`, `T`, `E10+`, `AO`, `EC`, `RP`.
- столбец `total_sales` содержит общие объёмы продаж по всем странам.

4. Категоризация данных

Проведём категоризацию данных:

1. Разделим все игры по оценкам пользователей `user_score` и выделите 4 категории:
   - `высокая оценка` (от 8 до 10 включительно),
   - `средняя оценка` (от 3 до 8, не включая правую границу интервала),
   - `низкая оценка` (от 0 до 3, не включая правую границу интервала),
   - `без оценки`
2. Разделим все игры по оценкам критиков `critic_score` и выделите 4 категории:
   - `высокая оценка` (от 80 до 100 включительно),
   - `средняя оценка` (от 30 до 80, не включая правую границу интервала),
   - `низкая оценка` (от 0 до 30, не включая правую границу интервала),
   - `без оценки`
3. Выделим топ-7 платформ `platform` по количеству игр, выпущенных за весь актуальный период.

4.1 Категоризация по оценкам пользователей

In [98]:
```
# Создаём категории для оценок пользователей
category_user_score = ['без оценки', 'низкая оценка', 'средняя оценка', 'высокая оценка']

# Создаём интервалы для категорий
value_user_score = [-1, 0, 3, 8, 11]
```

```
# Создаём новый столбец 'category_user' с категориями на основе оценок
games_actual['category_user'] = pd.cut(games_actual['user_score'], bins = value_user_score, labels = category_user_score, right=False)
```

In [99]:
```
# Группируем данные по столбцсе ли у 'category_user'
# выводим статистические данные по столбцу 'name'
games_actual.groupby('category_user', observed=False)['name'].describe()
```

Out[99]:

category_user	count	unique	top	freq
без оценки	6304	5276	FIFA 12	7
низкая оценка	116	103	THRILLVILLE: OFF THE RAILS	4
средняя оценка	4082	2700	FIFA 14	7
высокая оценка	2293	1824	NEED FOR SPEED: MOST WANTED	8

- `Высокую оценку` пользователей, то есть оценку от 8 до 10 (вкл) получили **2293 игры** - это 1824 игры имеющих по 1 и более релизов и вышедших на 1 или более платформах. Самая популярная игра в группе `NEED FOR SPEED: MOST WANTED`
- `Среднюю оценку` пользователей, то есть оценку от 3 до 8 (не вкл) получила **4082 игры** - это 2700 игр имеющих по 1 и более релизов и вышедших на 1 или более платформах. Самая популярная игра в группе `FIFA 14`
- `Низкую оценку` пользователей, то есть оценку от 0 до 3 (не вкл) получили **116 игр** - это 103 игр имеющих по 1 и более релизов и вышедших на 1 или более платформах. Самая популярная игра в группе `THRILLVILLE: OFF THE RAILS`
- `Без оценки` пользователей остались **6304 игры**.

4.2 Категоризация по оценкам критиков

In [100]:
```
# Создаём категории для оценок критоков
category_critic_score = ['без оценки', 'низкая оценка', 'средняя оценка', 'высокая оценка']

# Создаём интервалы для категорий
value_critic_score = [-1, 0, 30, 80, 101]

# Создаём новый столбец 'category_critic' с категориями на основе оценок
games_actual['category_critic'] = pd.cut(games_actual['critic_score'], bins = value_critic_score, labels = category_critic_score, right=False)
```

In [101]:
```
# Группируем данные по столбцу 'category_critic'
# считаем количество по столбцу 'name'
# сортируем по возрастанию
# выводим первые 7 значений
games_actual.groupby('category_critic', observed=False)['name'].describe()
```

Out[101]:

category_critic	count	unique	top	freq
без оценки	5616	4710	FIFA 12	7
низкая оценка	55	49	LEISURE SUIT LARRY: BOX OFFICE BUST	2
средняя оценка	5427	3620	CARS	8
высокая оценка	1697	1110	NEED FOR SPEED: MOST WANTED	7

- `Высокую оценку` критиков, то есть оценку от 80 до 100 (вкл) получили **1697 игр** - это 1110 игр, имеющих по 1 и более релизов и вышедших на 1 или более платформах. Самая популярная игра в группе `NEED FOR SPEED: MOST WANTED`.
- `Среднюю оценку` критиков, то есть оценку от 30 до 80 (не вкл) получили **5427 игр** - это 3620 игр, имеющих по 1 и более релизов и вышедших на 1 или более платформах. Самая популярная игра в группе `CARS`.
- `Низкую оценку` критиков, то есть оценку от 0 до 30 (не вкл) получили **55 игр** - это 49 игр имеющих, по 1 и более релизов и вышедших на 1 или более платформах. Самая популярная игра в группе `LEISURE SUIT LARRY: BOX OFFICE BUST`.
- `Без оценки` критиков остались **5616 игр**

In [102]:
```
# Выводим 3 строки датафрейма
games_actual.head(5)
```

Out[102]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating	total_sales	category_user	category_critic
0	WII SPORTS	WII	2006	sports	41.360001	28.959999	3.77	8.45	76.0	8.0	E	82.539993	высокая оценка	средняя оценка
1	MARIO KART WII	WII	2008	racing	15.680000	12.760000	3.79	3.29	82.0	8.3	E	35.520000	высокая оценка	высокая оценка
2	WII SPORTS RESORT	WII	2009	sports	15.610000	10.930000	3.28	2.95	80.0	8.0	E	32.770000	высокая оценка	высокая оценка
3	NEW SUPER MARIO BROS.	DS	2006	platform	11.280000	9.140000	6.50	2.88	89.0	8.5	E	29.799999	высокая оценка	высокая оценка
4	WII PLAY	WII	2006	misc	13.960000	9.180000	2.93	2.84	58.0	6.6	E	28.910000	средняя оценка	средняя оценка

4.3 ТОП-7 платформ по количеству игр, выпущенных за весь актуальный период

Сгруппируем данные по столбцу `platform` и посчитаем количество игр в каждой группе, а также общий объём продаж по группам. С помощью метода `assign()` добавим новый столбец `share`, в котором рассчитаем долю игр платформы от общего числа игр. Долю рассчитаем как отношение количества игр по каждой платформе к общему количеству игр, при этом общее количество игр равно количеству строк датафрейма. Для получения ТОП-7 отсортируем платформы по количеству игр с помощью `sort_values()` по убыванию и оставим только первые 7 строк.

In [103]:
```
# Группируем данные по столбцу 'platform'
# по столбцу 'name' считаем количество
# по столбцу 'total_sales' считаем сумму
# Добавляем новый столбец 'share' с помощью метода assign()
# Сортируем по убыванию значений столбца 'name' с помощью sort_values()
# Выводим первые 7 значений с помощью head(7)
games_actual.groupby('platform', as_index=False, observed=False).agg({'name': 'count',
                                                                      'total_sales': 'sum'}).assign(share=lambda x: x['name'] * 100 / games_actual.shape[0]).sort_values(by
```

Out[103]:

	platform	name	total_sales	share
16	PS2	2134	1247.280029	16.678390
4	DS	2121	802.780029	16.576788
25	WII	1275	886.109985	9.964830
19	PSP	1181	289.179993	9.230168
28	X360	1123	913.090027	8.776866
17	PS3	1086	863.149963	8.487691
6	GBA	811	312.880005	6.338413

- Лидер по количеству игр является платформа `PS2 (PlayStation 2)` , ей принадлежит почти **16,7%** всех игр - **2134 игры**. Общий объём продаж 1247,3 миллионов копий. Игровую консоль выпускали почти 13 лет, это один из самых длинных жизненных циклов в истории игровой индустрии.
- На втором месте карманная игровая консоль `DS (Nintendo)` с долей игр **16,6%** - **2121 игра**. Общий объём продаж 802,8 миллионов копий. Выпускалась на протяжении порядка 9 лет.
- На третьем месте с большим отрывом по количеству игр разместилась игровая приставка, являющаяся прямым конкурентом Microsoft Xbox 360 и Sony PlayStation 3 - `WII (Nintendo). Доля игр чуть меньше **10%** - **1275 игр**. Общий объём продаж 886,1 миллионов копий.
- Четвёртое место занимает портативная консоль `PSP (PlayStation Portable)` Доля игр составляет **9,2%** - **1181 игра**. Общий объём продаж самый низкий в ТОПе - 289,2 миллионов копий.
- На пятом месте разместилась вторая по счёту игровая приставка от компании Microsoft `X360 (Xbox 360)` , ей принадлежат **8,8% игр** - **1086 игр**. Общий объём продаж 913,1 миллионов копий.
- Лишь на шестом место платформа `PS3 (PlayStation 3)` с **8,5%** игр- **1086 игр**. Однако эта платформа имеет достаточно высокий показателей объёмов продаж 863,1 миллионов копий. С помощью `PS3` можно играть, просматривать фото, смотреть фильмы, слушать музыку, отправлять электронную почту и просматривать веб-страницы. Производилась с ноября 2006 по май 2017 года
- На седьмом месте `GBA (Game Boy Advance)` — портативная игровая система от компании Nintendo: **811 игр**, что составляет 6,3% от общего числа игр. Общий объём продаж один из самых низких в ТОПе - 312.9 миллионов копий.

5. Выводы

Для изучения истории продаж игр в начале 21 века был предоставлен датасет `new_games.csv` , данные выгружены в датафрейм `games` и проанализированы.
Исходный датафрейм представлял собой:

- количество строк: 16956,
- количество пропущенных значений: 25142,
- типы данных столбцов и их количество: float64(4), object(7), их низ 3 столбца с некорректными данными,
- объём используемой памяти: 1.4+ MB

Данные датафрейма `games` были очищены и оптимизированы. На этом этапе были выполнены следующие работы:

- проведена нормализация всех названий столбцов;
- преобразованы типы данных столбцов и понижены их разрядности;
- обработаны пропуски данных (заполнены индикаторами, восстановлены, удалены);
- удалены явные и неявные дубликаты.

В результате был создан новый оптимизированный датафрейм `games_new` с чистыми данными:

- количество строк: 16712
- типы данных столбцов и их количество: category(3), float32(6), int16(1), object(1)
- объём используемой памяти: 736.5+ KB

На основе датафрейма `games_new` был подготовлен срез данных `games_actual` за период 2000-2013г. Таким образом, получили датафрейм готовый к аналитической работе:

- количество строк: 12795
- типы данных столбцов и их количество: category(5), float32(7), int16(1), object(1)
- объём используемой памяти: 539,9+ KB

На этом этапе были проведены следующие работы:

- категоризация на основе оценок пользователей и критиков с добавлением новых полей `category_user` и `category_critic`
- добавлено поле с общим объёмом продаж `total_sales`
- сформирован ТОП-7 платформ по количеству игр, выпущенных за весь актуальный период.

В ТОП-7 собралась тройка самых известных компаний по производству игрового оборудования:

- компания `Sony` с 3 продуктами: стационарные `PlayStation 2` и `PlayStation 3` , портативная `PlayStation Portable` ;
- компания `Nintendo` с тремя продуктами: стационарные `DS` и `WII` , портативная `Game Boy Advance` ;
- компании `Microsoft` с 1 продуктом: `X360 (Xbox 360)` - стационарный.

Лидерами же по количеству игр является: платформа `PlayStation 2` с долей игр 16,7% (2134 игры) и карманная игровая консоль `DS Nintendo` с долей игр 16,6% (2121 игра).
