Проект: Развитие игровой индустрии

Автор: Котов Алексей

Почта: alexkotov1001@yandex.ru

Описание проекта

Цели и задачи проекта

Цель: Изучить развитие игровой индустрии с 2000 по 2013 год.

Задачи:

- Познакомиться с данными, проверить их корректность и провести предобработку.
- Отобрать данные по времени выхода игры: нужен период с 2000 по 2013 год включительно.
- Категоризовать игры по оценкам пользователей и экспертов.
- Выделить топ-7 платформ по количеству игр, выпущенных за весь требуемый период.

Описание данных

Датасет `new_games.csv` содержит информацию о продажах игр разных жанров и платформ, а также пользовательские и экспертные оценки игр: <...>

Содержание проекта

1. Загрузка данных и знакомство с ними

Загружаем необходимую библиотеку `pandas`.

In [53]:
```
# Импортируем библиотеку pandas
import pandas as pd
```

Загружаем данные датасета `new_games.csv` в датафрейм `df`.

In [ ]:
```
# Выгружаем данные из датафрейма new_games.csv в датафрейм df
df = pd.read_csv('.../new_games.csv')
```

Познакомимся с данными: выведем результат метода `info()` и первые строки.

In [55]:
```
# Выводим информацию о датафрейме
df.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 [56]:
```
# Выводим первые строки датафрейма на экран
df.head()
```

Out[56]:

	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

Вывод о полученных данных:

- В датасете 16956 строк и 11 столбцов.
- Данные соответствуют описанию.
- В данных присутствуют пропуски.
- В данных не во всех полях используются верные типы данных.
- В данных возможны дубликаты.
- Названия столбцов прописаны в неудобном для работы виде.

Стек решения данных проблем и поставленной аналитической задачи будет следующий:

Проверка ошибок в данных и их предобработка -> Фильтрация данных -> Категоризация данных -> Итоговый вывод.

2. Проверка ошибок в данных и их предобработка

2.1. Названия столбцов датафрейма

Выведем на экран названия всех столбцов датафрейма и проверьте их стиль написания.

In [57]:
```
# Для вывода названий столбцов используем атрибут columns
df.columns
```

Out[57]:
```
Index(['Name', 'Platform', 'Year of Release', 'Genre', 'NA sales', 'EU sales',
       'JP sales', 'Other sales', 'Critic Score', 'User Score', 'Rating'],
      dtype='object')
```

Приведем все столбцы к стилю snake case. Названия должны быть в нижнем регистре, а вместо пробелов — подчёркивания.

In [58]:
```
# Используем rename(), чтобы не запутаться в последовательности столбцов
df = df.rename(columns={'Name': 'name',
                        'Platform': 'platform',
                        'Year of Release': 'year_of_release',
                        'Genre': 'genre',
                        'NA sales': 'na_sales',
                        'EU sales': 'eu_sales',
                        'JP sales': 'jp_sales',
                        'Other sales': 'other_sales',
                        'Critic Score': 'critic_score',
                        'User Score': 'user_score',
                        'Rating': 'rating'})
# Выведем снова названия столбцов для проверки замены названий
df.columns
```

Out[58]:
```
Index(['name', 'platform', 'year_of_release', 'genre', 'na_sales', 'eu_sales',
       'jp_sales', 'other_sales', 'critic_score', 'user_score', 'rating'],
      dtype='object')
```

2.2. Типы данных

2.2.1. Информация о типах данных

Проверим наличие некорректных типов данных.

In [59]:
```
# Выводим информацию о типах данных
df.dtypes
```

Out[59]:
```
name              object
platform          object
year_of_release    float64
genre             object
na_sales          float64
eu_sales          object
jp_sales          object
other_sales       float64
critic_score      float64
user_score        object
rating            object
dtype: object
```

- Поле `name` типа `object` - корректно, т. к. поле хранит текстовую информацию.
- Поле `platform` типа `object` - допустимо, но поле хранит ограниченный список значений, рекомендуется использовать тип `category`.
- Поле `year_of_release` типа `float64` - ***некорректно***, т. к. поле хранит только целые значения года, требуется использовать тип `int`, рекомендуется уменьшить битовый разряд.
- Поле `genre` типа `object` - допустимо, но поле хранит ограниченный список значений, рекомендуется использовать тип `category`.
- Поле `na_sales` типа `float64` - корректно, т. к. поле хранит дробные числа (количество в миллионах), рекомендуется уменьшить битовый разряд.
- Поле `eu_sales` типа `object` - ***некорректно***, т. к. поле хранит дробные числа (количество в миллионах), требуется использовать тип `float`, рекомендуется уменьшить битовый разряд.
- Поле `jp_sales` типа `object` - ***некорректно***, т. к. поле хранит дробные числа (количество в миллионах), требуется использовать тип `float`, рекомендуется уменьшить битовый разряд.
- Поле `other_sales` типа `float64` - корректно, т. к. поле хранит дробные числа (количество в миллионах), рекомендуется уменьшить битовый разряд.
- Поле `critic_score` типа `float64` - допустимо, т. к. поле хранит числа от 0 до 100, рекомендуется изменить поле на `int` и уменьшить битовый разряд.
- Поле `user_score` типа `object` - ***некорректно***, т. к. поле хранит дробные числа от 0 до 10, требуется использовать тип `float`, рекомендуется уменьшить битовый разряд.
- Поле `rating` типа `object` - допустимо, но поле хранит ограниченный список значений, рекомендуется использовать тип `category`.

2.2.2. Преобразование типа поля `year_of_release`

Преобразуем тип данных поля `year_of_release` на целочисленный с помощью метода `astype()` с выводом сообщения при обнаружении ошибки. Используем конструкцию `try-except`, чтобы не прерывалось выполнение кода при ошибках.

In [60]:
```
try:
    # пытаемся преобразовать тип данных поля в целочисленный
    df['year_of_release'] = df['year_of_release'].astype('int16', errors='raise')
except Exception as err:
    # сообщение об ошибке
    print(f'Произошла ошибка: "{err}"')
else:
    # выведем тип данных, если не было ошибок
    print(f"Успешно изменили тип на: {df['year_of_release'].dtypes}")
finally:
    # выведем сообщение, что код выполнился
    print("Операция завершена")
```

```
Произошла ошибка: "Cannot convert non-finite values (NA or inf) to integer"
Операция завершена
```

Сообщение об ошибке показало, что имеются пропуски.

Посчитаем количество пропусков в абсолютном и относительном значениях.

In [61]:
```
# Выводим количество пропусков
df['year_of_release'].isna().sum()
```

Out[61]:
```
275
```

In [62]:
```
# Подсчитываем процент строк с пропусками
df['year_of_release'].isna().mean() * 100
```

Out[62]:
```
1.6218447747110165
```

Поле `year_of_release` содержит 275 пропусков (1.6% от общего числа строк).

Для каждой игры заменим пропуски на медианное значение года выхода той же игры на других платформах.

In [63]:
```
# создаем словарь с медианами для каждого названия игры
medians = df.groupby('name')['year_of_release'].median()

# Заполняем пропуски медианными значениями
df['year_of_release'] = df['year_of_release'].fillna(df['name'].map(medians))

# проверим количество пропусков
df['year_of_release'].isna().sum()
```

Out[63]: 148

Количество пропусков уменьшилось с 275 до 148. Остальные пропуски заменим нулями.

In [64]:
```
# Заполним пропуски нулями
df['year_of_release'] = df['year_of_release'].fillna(0)

# Проверим количество пропусков
df['year_of_release'].isna().sum()
```

Out[64]: 0

Преобразуем тип поля `year_of_release` с помощью метода `to_numeric()` с понижением размерности.

In [65]:
```
# Преобразуем тип данных в поле year_of_release с понижением размерности для целочисленных значений
df['year_of_release'] = pd.to_numeric(df['year_of_release'], downcast='integer')

# Проверим тип данных
df['year_of_release'].dtypes
```

Out[65]: dtype('float64')

Тип поля `year_of_release` остался `float64`.

Преобразуем тип данных поля `year_of_release` на целочисленный с помощью метода `astype()` с выводом сообщения при обнаружении ошибки. Используем конструкцию `try-except`, чтобы не прерывалось выполнение кода при ошибках.

In [66]:
```
try:
    # пытаемся преобразовать тип данных поля в целочисленный
    df['year_of_release'] = df['year_of_release'].astype('int16', errors='raise')
except Exception as err:
    # сообщение об ошибке
    print(f'Произошла ошибка: "{err}"')
else:
    # выведем тип данных, если не было ошибок
    print(f"Успешно изменили тип на: {df['year_of_release'].dtypes}")
finally:
    # выведем сообщение, что код выполнился
    print("Операция завершена")
```

```
Успешно изменили тип на: int16
Операция завершена
```

Тип поля `year_of_release` успешно преобразовали в `int16`. Выведем первые строки поля.

In [67]:
```
df['year_of_release'].head()
```

Out[67]:
```
0    2006
1    1985
2    2008
3    2009
4    1996
Name: year_of_release, dtype: int16
```

Теперь значения поля `year_of_release` имеют корректный тип данных.

2.2.3. Преобразование типа полей `eu_sales`, `jp_sales`, `critic_score` и `user_score`

- Преобразуем тип полей `eu_sales`, `jp_sales`, `critic_score` и `user_score` на вещественный.
- Преобразование типов реализуем с помощью метода `to_numeric()` с понижением размерности, в случае невозможности преобразовать тип заменим данные на пропуски.

In [68]:
```
# Преобразуем тип данных в необходимых слобцах с понижением размерности для вещественных значений
for column in ['eu_sales', 'jp_sales', 'critic_score', 'user_score']:
    df[column] = pd.to_numeric(df[column], errors='coerce', downcast='float')

# Проверим тип данных
df[['eu_sales', 'jp_sales', 'critic_score', 'user_score']].dtypes
```

Out[68]:
```
eu_sales        float32
jp_sales        float32
critic_score    float32
user_score      float32
dtype: object
```

Успено изменили тип полей `eu_sales`, `jp_sales`, `critic_score` и `user_score` на вещественный и понизили размерность до 32-бит.

2.3. Пропуски в данных

2.3.1. Общая информация о пропусках

Посчитаем количество пропусков в каждом столбце в абсолютных и относительных значениях.

In [69]:
```
# Выводим количество пропущенных строк в датафрейме с сортировкой по возрастанию пропусков
df.isna().sum().sort_values()
```

Out[69]:
```
platform              0
year_of_release       0
na_sales              0
other_sales           0
name                  2
genre                 2
jp_sales              4
eu_sales              6
rating             6871
critic_score       8714
user_score         9268
dtype: int64
```

In [70]:
```
# Подсчитываем процент строк с пропусками с сортировкой по возрастанию пропусков
df.isna().mean().sort_values() * 100
```

Out[70]:
```
platform            0.000000
year_of_release     0.000000
na_sales            0.000000
other_sales         0.000000
name                0.011795
genre               0.011795
jp_sales            0.023590
eu_sales            0.035386
rating             40.522529
critic_score       51.391838
user_score         54.659118
dtype: float64
```

- Поле `name` содержит 2 пропуска (0.01% от общего числа строк).
- Поле `genre` содержит 2 пропуска (0.01% от общего числа строк).
- Поле `eu_sales` содержит 6 пропусков (0.04% от общего числа строк).
- Поле `jp_sales` содержит 4 пропусков (0.02% от общего числа строк).
- Поле `critic_score` содержит 8714 пропусков (51.39% от общего числа строк).
- Поле `user_score` содержит 9268 пропусков (54.66% от общего числа строк).
- Поле `rating` содержит 6871 пропусков (40.52% от общего числа строк).

В ходе анализа пропусков возможно их удаление, выведем снова исходное количество записей датафреймйма.

In [71]:
```
# вывод количества записей
df.shape[0]
```

Out[71]:
```
16956
```

В датафреймйме 16956 записей.

2.3.2. Пропуски в полях `name` и `genre`

Проверим записи, где поле `name` или поле `genre` содержит пропуски.

In [72]:
```
# выведем записи в пропусками в полях name или genre
df[df['name'].isna() | df['genre'].isna()]
```

Out[72]:

	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	NaN	NaN	NaN
14439	NaN	GEN	1993	NaN	0.00	0.00	0.03	0.00	NaN	NaN	NaN

Только две одни и те же игры имеют пропуски в полях `name` и `genre`. Также в записях пропуски в полях `critic_score`, `user_score` и `rating`. Возможно, данные записи имеют пропуски в результате технической ошибки. Фрагментарной информацией из данных двух записей в ходе дальнейшего анализа можно пренебречь, поэтому удалим их, новый датафреймймйм запишем в новую переменную `df

In [73]:
```
# удалим записи с пропусками в поле name или genre
# результат поместим в новую переменную df
df = df.dropna(subset=['name', 'genre'])

# проверим количество оставшихся записей
df.shape[0]
```

Out[73]:
```
16954
```

Количество записей уменьшилось на 2, т. е. стало 16954. Проверим, что данные записи отсутсвуют в новом датафрейме df.

In [74]:
```
# выведем количество записей с пропусками в полях name или genre
df[df['name'].isna() | df['genre'].isna()].shape[0]
```

Out[74]:
```
0
```

2.3.3. Пропуски в полях `eu_sales` и `jp_sales`

Изучим записи с пропусками в полях `eu_sales` или `jp_sales`.

In [75]:
```
# выведем записи с пропусками в полях eu_sales или jp_sales
#df[df['eu_sales'].isna() | df['jp_sales'].isna()]

# можно вывести все записи игр на разных платформах с названиями из names_nan_eu_sales и сортировкой по полю name
# в переменную names_nan_sales сохраним серию из названий игр, у которых пропуск в полях eu_sales или jp_sales
names_nan_sales = df[df['eu_sales'].isna() | df['jp_sales'].isna()]['name'].unique()
df[df['name'].isin(names_nan_sales)].sort_values('name')
```

Out[75]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
4732	Castlevania: The Dracula X Chronicles	PSP	2007	Platform	0.22	0.09	NaN	0.07	80.0	7.8	T
16375	Dead Rising	XOne	2016	Action	0.01	0.00	0.00	0.00	NaN	8.8	M
14064	Dead Rising	PS4	2016	Action	0.02	0.00	0.01	0.01	78.0	6.6	M
802	Dead Rising	X360	2006	Action	1.16	NaN	0.08	0.20	85.0	7.6	M
3219	Far Cry 4	PC	2014	Shooter	0.15	0.44	0.00	0.04	80.0	6.7	M
2319	Far Cry 4	X360	2014	Shooter	0.45	0.37	0.00	0.08	NaN	7.0	M
1531	Far Cry 4	PS3	2014	Shooter	0.34	0.68	0.08	0.19	NaN	7.3	M
1132	Far Cry 4	XOne	2014	Shooter	0.80	NaN	0.01	0.14	82.0	7.5	M
299	Far Cry 4	PS4	2014	Shooter	1.13	2.18	0.10	0.63	85.0	7.7	M
1379	Hello Kitty Party	DS	2007	Misc	0.78	0.51	NaN	0.12	NaN	NaN	E
7795	Prince of Persia: Warrior Within	GC	2004	Action	0.15	0.04	0.00	0.01	83.0	8.6	M
1131	Prince of Persia: Warrior Within	PS2	2004	Action	0.54	NaN	0.00	0.22	83.0	8.5	M
2591	Prince of Persia: Warrior Within	XB	2004	Action	0.48	0.28	0.00	0.04	83.0	7.5	M
9139	Ratatouille	GC	2007	Action	0.11	0.03	0.00	0.00	60.0	5.6	E
7783	Ratatouille	GBA	2007	Action	0.14	0.05	0.00	0.00	65.0	NaN	E
6481	Ratatouille	X360	2007	Action	0.23	0.02	0.00	0.02	56.0	NaN	E
3992	Ratatouille	Wii	2007	Action	0.43	0.03	0.00	0.04	62.0	6.7	E
3946	Ratatouille	PS3	2007	Action	0.09	0.32	0.00	0.10	55.0	4.8	E
2466	Ratatouille	PS2	2007	Action	0.31	0.00	0.00	0.53	65.0	8.0	E
14714	Ratatouille	PC	2007	Action	0.01	0.01	0.00	0.00	NaN	7.9	E
1612	Ratatouille	DS	2007	Action	0.49	NaN	0.00	0.14	NaN	NaN	NaN
3102	Ratatouille	PSP	2007	Action	0.22	0.27	0.00	0.16	64.0	7.1	E
446	Rhythm Heaven	DS	2008	Misc	0.55	NaN	1.93	0.13	83.0	9.0	E
2383	Rhythm Heaven	Wii	2008	Misc	0.11	0.00	0.77	0.01	NaN	NaN	NaN
467	Saints Row 2	X360	2008	Action	1.94	0.79	NaN	0.28	81.0	8.1	M
1082	Saints Row 2	PS3	2008	Action	0.88	0.54	0.02	0.25	82.0	7.9	M
14251	Saints Row 2	PC	2009	Action	0.00	0.03	0.00	0.01	72.0	6.8	M
1394	Sonic Advance 3	GBA	2004	Platform	0.74	NaN	0.08	0.06	79.0	8.4	E
819	UFC 2009 Undisputed	X360	2009	Fighting	1.48	0.39	NaN	0.19	83.0	7.9	T
999	UFC 2009 Undisputed	PS3	2009	Fighting	1.07	0.44	0.01	0.24	84.0	8.1	T

В 10 записях пропуски в полях `eu_sales` или `jp_sales` могут быть или из-за технической ошибки, или из-за отсуствия данной информации в доступных источниках, или пренебрежимо малых продажах, или вообще запрета продаж в данном регионе.

Несмотря на то, что всего 10 записей имеют пустые значения в полях `eu_sales` или `jp_sales`, удалить эти записи не целесообразно, т. к. остальные поля несут полезную информацию. Поля `eu_sales` и `jp_sales` можно заполнить нулями или средним значением поля отдельно для каждой игры на других платформах, а также можно оставить поля пустыми.

В данной ситуации отдельные игры на различных платформах в различных регионах мира могут продаваться совершенно по-разному. Например, в Японии семейство Xbox непопулярно, и заполняя пропуски продаж какой-то игры на Xbox средним значением по столбцу, куда войдут высокие продажи на PS3 или PS4, явно завысит реальные данные. Также разные игры, но одного и того же жанра и одного года выхода могут в одном регионе, могут иметь абсолютно разные продажи, то усреднение по данным полям также может сильно исказить продажи.

Принято решение оставить пропуски пустыми. Агрегирующие функции просто пропустят данные пропуски в расчетах, если они потребуются.

2.3.4. Пропуски в полях `critic_score` и `user_score`

In [76]:
```
# Подсчитываем процент строк с пропусками с сортировкой по возрастанию пропусков
df[['critic_score', 'user_score']].isna().mean().sort_values() * 100
```

Out[76]:
```
critic_score    51.386104
user_score      54.653769
dtype: float64
```

Поля `critic_score` и `user_score` содержат пропусков более 50% от общего числа строк. Удалить такое количество записей нельзя. Замена пустых полей на меры центральной тенденции (среднее значение, медиана, мода) приведет к искажению данных. Поэтому можно оставить данные поля пустыми или заменить на значения-индикаторы.

Решено пустые записи полей `critic_score` и `user_score` заполнить значением `-1`. Это позволит поле `critic_score` преобразовать в целочисленный тип, а для вещественного поля `user_score` понизить размерность.

In [77]:
```
# Заполняем пропуски critic_score значением -1
df['critic_score'] = df['critic_score'].fillna(-1)

# Преобразуем тип данных в поле critic_score с понижением размерности для целочисленных значений
df['critic_score'] = pd.to_numeric(df['critic_score'], downcast='integer')

# Проверка типа поля
df['critic_score'].dtypes
```

Out[77]:
```
dtype('int8')
```

В поле `critic_score` пропуски успешно заменили на индикатор `-1`, преобразовали в тип `int8` с понижением размерности. *(Целочисленный тип в `pandas` не может хранить пропуски, поэтому это гарантирует их отсуствие.)*

In [78]:
```
# Заполняем пропуски user_score значением -1
df['user_score'] = df['user_score'].fillna(-1)

# проверим количество пропусков user_score
df['user_score'].isna().sum()
```

Out[78]:
```
0
```

В поле `user_score` пропусков больше нет.

In [79]:
```
# Преобразуем тип данных в поле user_score с понижением размерности для целочисленных значений
df['user_score'] = pd.to_numeric(df['user_score'], downcast='float')

# Проверка типа поля user_score
df['user_score'].dtypes
```

Out[79]:
```
dtype('float32')
```

Поле `user_score` осталось с типом `float32`.

2.3.5. Пропуски в поле `rating`

In [80]:
```
# Выводим количество пропусков
df['rating'].isna().sum()
```

Out[80]:
```
6869
```

In [81]:
```
# Подсчитываем процент строк с пропусками
df['rating'].isna().mean() * 100
```

Out[81]:
```
40.515512563406865
```

- В поле `rating` пропусков 6869 (40.5% от общего количества записей).
- Пропуски в поле `rating` будут у старых игр, т. к. самые первые системы рейтингов вводились только с 1994 года.
- У игр моложе 1994 года пропуски могут быть из-за ошибок или из-за отсутствия информации в источнике.
- Пропуск в рейтинге также может означать доступность игры без возрастных ограничений.
- Также рейтинг одной и той же игры на разных платформах может быть различным.
- Заполним пропуски в поле `rating` строковым значение `unknown`, т. к. для поставленной аналитической задачи данное поле не имеет важного значения.

In [82]:
```
# Заполняем пустые значения
df['rating'] = df['rating'].fillna('unknown')

# Проверяем количество пустых значений
df['rating'].isna().sum()
```

Out[82]:
```
0
```

2.4. Явные и неявные дубликаты в данных

In [83]:
```
# Сортируем датафрейм по всем столбцам
df = df.sort_values(by=df.columns.tolist())
# Выведем первые строки
df.head()
```

Out[83]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
15191	Beyblade Burst	3DS	2016	Role-Playing	0.00	0.00	0.03	0.00	-1	-1.0	unknown
15192	Beyblade Burst	3DS	2016	Role-Playing	0.00	0.00	0.03	0.00	-1	-1.0	unknown
1086	Fire Emblem Fates	3DS	2015	Role-Playing	0.81	0.23	0.52	0.11	-1	-1.0	unknown
3394	Frozen: Olaf's Quest	3DS	2013	Platform	0.27	0.27	0.00	0.05	-1	-1.0	unknown
3906	Frozen: Olaf's Quest	DS	2013	Platform	0.21	0.26	0.00	0.04	-1	-1.0	unknown

2.4.1. Неявные дубликаты

Изучаем уникальные значения в категориальных данных: названия жанра, платформы, рейтинга и года выпуска. Проверим, встречаются ли среди данных неявные дубликаты, связанные с опечатками или разным способом написания.

2.4.1.1. Проверка поля `platform`

In [84]:
```
# Выводим количество уникальных значений для выбранного поля platform с сортировкой по названию
df['platform'].value_counts().sort_index()
```

Out[84]:
```
2600     135
3DO        3
3DS      530
DC        52
DS      2177
GB        98
GBA      837
GC       563
GEN       27
GG         1
N64      323
NES      100
NG        12
PC       990
PCFX       1
PS      1215
PS2     2189
PS3     1355
PS4      395
PSP     1229
PSV      435
SAT      174
SCD        6
SNES     241
TG16       2
WS         6
Wii     1340
WiiU     147
X360    1281
XB       839
XOne     251
Name: platform, dtype: int64
```

В поле `platform` дубликатов нет.

2.4.1.2. Проверка поля `year_of_release`

In [85]:
```
# Выводим количество уникальных значений для выбранного поля year_of_release с сортировкой по году
df['year_of_release'].value_counts().sort_index()
```

Out[85]:
```
0       148
1980      9
1981     46
1982     37
1983     18
1984     14
1985     14
1986     22
1987     17
1988     15
1989     17
1990     16
1991     42
1992     43
1993     60
1994    121
1995    220
1996    267
1997    293
1998    384
1999    343
2000    358
2001    495
2002    854
2003    798
2004    774
2005    959
2006   1033
2007   1223
2008   1460
2009   1455
2010   1293
2011   1167
2012    679
2013    557
2014    584
2015    612
2016    507
Name: year_of_release, dtype: int64
```

В поле `year_of_release` дубликатов нет. Есть 148 строк с нулевым значением (неопределенным годом выхода).

2.4.1.3. Проверка поля `genre`

In [86]:
```
# Выводим количество уникальных значений для выбранного поля genre с сортировкой по названию
df['genre'].value_counts().sort_index()
```

Out[86]:
```
ACTION           13
ADVENTURE         4
Action         3405
Adventure      1319
FIGHTING          6
Fighting        856
MISC              3
Misc           1769
PLATFORM          3
PUZZLE            2
Platform        901
Puzzle          588
RACING            6
ROLE-PLAYING      6
Racing         1267
Role-Playing   1510
SHOOTER           5
SIMULATION        2
SPORTS            8
STRATEGY          1
Shooter        1341
Simulation      882
Sports         2367
Strategy        690
Name: genre, dtype: int64
```

В поле `genre` дубликаты есть. Названия жанров приведем к нижнему регистру.

In [87]:
```
# приведем все названия поля genre к нижнему регистру
df.loc[:, 'genre'] = df['genre'].str.lower()

# Выводим количество уникальных значений для выбранного поля genre с сортировкой по названию
df['genre'].value_counts().sort_index()
```

Out[87]:
```
action         3418
adventure      1323
fighting        862
misc           1772
platform        904
puzzle          590
racing         1273
role-playing   1516
shooter        1346
simulation      884
sports         2375
strategy        691
Name: genre, dtype: int64
```

В поле `genre` дубликатов теперь нет.

2.4.1.4. Проверка поля `rating`

In [88]:
```
# Выводим количество уникальных значений для выбранного поля rating с сортировкой по названию
df['rating'].value_counts().sort_index()
```

Out[88]:
```
AO            1
E          4037
E10+       1441
EC            8
K-A           3
M          1587
RP            3
T          3005
unknown    6869
Name: rating, dtype: int64
```

В поле `rating` дубликаты есть. Категория `E` до 1998 года обозначалась `K-A`. Заменим значения `K-A` на новое `E`.

In [89]:
```
# Заменяем значения K-A на E в поле rating
df.loc[:, 'rating'] = df['rating'].str.replace('K-A', 'E')

# Выводим количество уникальных значений для выбранного поля rating с сортировкой по названию
df['rating'].value_counts().sort_index()
```

Out[89]:
```
AO            1
E          4040
E10+       1441
EC            8
M          1587
RP            3
T          3005
unknown    6869
Name: rating, dtype: int64
```

В поле `rating` дубликатов больше нет.

2.4.1.5. Проверка поля `name`

Избавляемся от неявных дубликатов в поле `name`.

In [90]:
```
# В названии все буквы заглавные
df['name'] = df['name'].str.upper()

# Удаляем пробелы с обеих сторон строки
df['name'] = df['name'].str.strip()

# Заменяем пробелы внутри строки на нижнее подчеркивание
#df['name'] = df['name'].str.replace(' ', '_')

# Заменяем все символы кроме букв на нижнее подчеркивание (несколько знаков подряд на одно подчеркивание) с помощью регулярных выражений
df['name'] = df['name'].str.replace(r'[^\w]+', '_', regex=True)

# Выведем первые строки датафрейма
df.head()
```

Out[90]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
15191	BEYBLADE_BURST	3DS	2016	role-playing	0.00	0.00	0.03	0.00	-1	-1.0	unknown
15192	BEYBLADE_BURST	3DS	2016	role-playing	0.00	0.00	0.03	0.00	-1	-1.0	unknown
1086	FIRE_EMBLEM_FATES	3DS	2015	role-playing	0.81	0.23	0.52	0.11	-1	-1.0	unknown
3394	FROZEN_OLAF_S_QUEST	3DS	2013	platform	0.27	0.27	0.00	0.05	-1	-1.0	unknown
3906	FROZEN_OLAF_S_QUEST	DS	2013	platform	0.21	0.26	0.00	0.04	-1	-1.0	unknown

В записях поля `name` заменили все буквы на заглавные буквы и заменили все символы на нижнее подчеркивание.

2.4.2. Явные дубликаты

После того, как нормализовали данные и устранили неявные дубликаты, проверим наличие явных дубликатов в данных. Выведем количество записей в датафрейме до того, как удалим явные дубликаты.

In [91]:
```
# Выводим количество записей в датафрейме
df.shape[0]
```

Out[91]: 16954

До удаления дубликатов в датафрейме 16954 записей.

In [92]:
```
# создадим новый датафрейм и удалим явные дуликаты по всем полям, сохраняя только первый дубликат
df_uniq = df.drop_duplicates(subset=None, keep='first', inplace=False).copy()

# выведем новое количество оставшихся записей
df_uniq.shape[0]
```

Out[92]: 16713

После удаления явных дубликатов в датафрейме 16713 записей. Выведем количество неявных дубликатов по названию, платформе и году выхода игры.

In [93]:
```
# выведем количество дубликатов по названию, платформе и году выхода игры
df_uniq[df_uniq.duplicated(subset=['name', 'platform', 'year_of_release'], keep=False)].sort_values('name').shape[0]
```

Out[93]: 4

Таких неявных дубликатов всего только 4. Выведем их на экран.

In [94]:
```
# Выведем дубликаты по названию, платформе и году выхода игры
df_uniq[df_uniq.duplicated(subset=['name', 'platform', 'year_of_release'], keep=False)].sort_values('name')
```

Out[94]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
16465	MADDEN_NFL_13	PS3	2012	sports	0.00	0.01	0.00	0.00	83	5.5	E
606	MADDEN_NFL_13	PS3	2012	sports	2.11	0.22	0.00	0.23	83	5.5	E
4173	SONIC_THE_HEDGEHOG	PS3	2006	platform	0.00	0.48	0.00	0.00	43	4.1	E10+
1760	SONIC_THE_HEDGEHOG	PS3	2006	platform	0.41	0.06	0.04	0.66	43	4.1	E10+

Видим, что две игры имеют по две записи. Информация по продажам у неявных дубликатов различается, возможно, это были разные издания. Поэтому добавим в датафрейм новые записи, в которых будет сумма продаж из полей неявных дубликатов, а сами неявные дубликаты удалим.

In [95]:
```
# сохраняем дубликаты отдельно
dup = df_uniq[df_uniq.duplicated(subset=['name', 'platform', 'year_of_release'], keep=False)].sort_values('name')

# применим агрегацию, чтобы продажи у дубликатов сложить, остальные поля оставить по первому вхождению
df_agg = dup.groupby(['name', 'platform', 'year_of_release'], as_index=False).agg({
    'name': 'first',
    'platform': 'first',
    'year_of_release': 'first',
    'genre': 'first',
    'na_sales': 'sum',
    'eu_sales': 'sum',
    'jp_sales': 'sum',
    'other_sales': 'sum',
    'critic_score': 'first',
    'user_score': 'first',
    'rating': 'first'})
# выведем новые записи
df_agg
```

Out[95]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
0	MADDEN_NFL_13	PS3	2012	sports	2.11	0.23	0.00	0.23	83	5.5	E
1	SONIC_THE_HEDGEHOG	PS3	2006	platform	0.41	0.54	0.04	0.66	43	4.1	E10+

Видим, что правильно сложили продажи, теперь можем удалить неявные дубликаты и добавим вместо них новые записи.

In [96]:
```
# удаляем дубликаты
df_uniq = df_uniq.drop_duplicates(subset=['name', 'platform', 'year_of_release'], keep=False, inplace=False)
```

```
# добавим новые записи
df_uniq = pd.concat([df_uniq, df_agg], ignore_index=True)

# проверим, что добавлены записи вместо старых дубликатов
df_uniq[df_uniq['name'].isin(df_agg['name']) & df_uniq['platform'].isin(df_agg['platform']) & df_uniq['year_of_release'].isin(df_agg['year_of_release'])]
```

Out[96]:

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
16709	MADDEN_NFL_13	PS3	2012	sports	2.11	0.23	0.00	0.23	83	5.5	E
16710	SONIC_THE_HEDGEHOG	PS3	2006	platform	0.41	0.54	0.04	0.66	43	4.1	E10+

Корректно удалили четыре дубликата и заменили их двумя новыми записями. Выведем общее количество записей в датафрейме.

In [97]:
```
# выведем количество оставшихся записей
df_uniq.shape[0]
```

Out[97]: 16711

После удаления явных дубликатов в датафрейме 16711 записей: 16713 - 4 + 2 = 16711 - всё верно.

In [98]:
```
# абсолютное количество удаленных записей
df.shape[0] - df_uniq.shape[0]
```

Out[98]: 243

In [99]:
```
# относительное количество удаленных записей
(df.shape[0] - df_uniq.shape[0]) / df.shape[0] * 100
```

Out[99]: 1.4332900790373952

Удалили 243 записи, что составило 1.43% от общего числа.

2.5. Промежуточный вывод после предобработки данных

- Изначально в датасете было 16956 строк и 11 столбцов.
- Привели все столбцы к стилю "snake case".
- Проверили некорректные типы данных:
  - Преобразовали поле `year_of_release` к целочисленному типу с уменьшением размерности, в 127 пропусках восстановили настоящие значения и 148 пропусков заменили нулями.
  - Преобразовали поля `eu_sales`, `jp_sales`, `critic_score` и `user_score` к вещественному типу с уменьшением размерности, значения все данные, которые не удалось преобразовать в вещественные числа, заменили пропусками.
- Обработали пропуски в данных:
  - Пропуски в полях `name` и `genre` обнаружены лишь в 2 записях, которые были удалены. Осталось 16954 строки.
  - Пропуски в полях `eu_sales` и `jp_sales` обнаружены в 10 записях, которые оставили пустыми.
  - Пропуски в полях `critic_score` и `user_score` содержало порядка 54.6% записей. Пропускам присвоили значение-индикатор `-1`.
  - Поле `critic_score` преобразовали в целочисленный тип `int8`.
  - Пропуски в поле `rating` составляли порядка 41% записей, пропускам присвоили значение `unknown`.
- Выявили и устранили явные и неявные дубликаты:
  - В полях `platform` и `year_of_release` неявных дубликатов не было.
  - Поле `genre` имело 24 различных значения, после приведения к нижнему регистру осталось 12 уникальных значений.
  - Поле `rating` имело 9 различных значений, включая `unknown`. Заменили 3 значения старого обозначения `K-A` на новое обозначение `E`, осталось 8 различных уникальных значения.
  - В поле `name` все значения привели в верхнему регистру, удалили пробелы перед и после строки, с помощью регулярных выражений заменили все символы на нижнее подчеркивание.
  - До удаления дубликатов записей было 16954.
  - Были выявлены 2 пары записей являющихся неявными дубликатами по названию, платформе и году релиза. Каждую пару неявных дубликатов заменили на одну запись.
  - После выявления возможных неявных дубликатов были удалены явные дубликаты.
  - Итого было удалено 243 строк (порядка 1.4% от общего числа), осталось 16711 строк в датафрейме.

3. Фильтрация данных

Коллеги хотят изучить историю продаж игр в начале XXI века, и их интересует период с 2000 по 2013 год включительно. Отберем данные по этому показателю. Сохраним новый срез данных в отдельном датафрейме, например `df_actual`.

In [100…
```
# отфильтруем записи с годом от 2000 до 2013 включительно и сохраним в отдельный датафрейм
df_actual = df_uniq[(df_uniq['year_of_release'] >= 2000) & (df_uniq['year_of_release'] <= 2013)].copy()

# выведем первые строки нового датафрейма
df_actual.head()
```

Out[100…

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating
2	FROZEN_OLAF_S_QUEST	3DS	2013	platform	0.27	0.27	0.00	0.05	-1	-1.0	unknown
3	FROZEN_OLAF_S_QUEST	DS	2013	platform	0.21	0.26	0.00	0.04	-1	-1.0	unknown
5	TALES_OF_XILLIA_2	PS3	2012	role-playing	0.20	0.12	0.45	0.07	71	7.9	T
7	_HACK_G_U_VOL_1_REBIRTH	PS2	2006	role-playing	0.00	0.00	0.17	0.00	-1	-1.0	unknown
8	_HACK_G_U_VOL_2_REMINISCE	PS2	2006	role-playing	0.11	0.09	0.00	0.03	-1	-1.0	unknown

In [101…
```
# выведем количество записей в новом датафрейме
df_actual.shape[0]
```

Out[101… 12900

В новом датафрейме со срезом по году релиза игры 12900 записей.

4. Категоризация данных

4.1. Категоризация данных по полю `user_score`

Разделим все игры по оценкам пользователей и выделим такие категории: высокая оценка (от 8 до 10 включительно), средняя оценка (от 3 до 8, не включая правую границу интервала) и низкая оценка (от 0 до 3, не включая правую границу интервала).

In [102…
```
# Функция для категоризации
def user_categorize(value):
    if 0 <= value < 3:
        return 'low'
```

```
    elif 3 <= value < 8:
        return 'medium'
    elif 8 <= value <= 10:
        return 'high'
    else:
        return 'unknown'

# Применение функции к данным
df_actual.loc[:, 'user_rating'] = df_actual['user_score'].apply(user_categorize)

# выведем первые пять строк
df_actual.head()
```

Out[102…

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating	user_rating
2	FROZEN_OLAF_S_QUEST	3DS	2013	platform	0.27	0.27	0.00	0.05	-1	-1.0	unknown	unknown
3	FROZEN_OLAF_S_QUEST	DS	2013	platform	0.21	0.26	0.00	0.04	-1	-1.0	unknown	unknown
5	TALES_OF_XILLIA_2	PS3	2012	role-playing	0.20	0.12	0.45	0.07	71	7.9	T	medium
7	_HACK_G_U_VOL_1_REBIRTH	PS2	2006	role-playing	0.00	0.00	0.17	0.00	-1	-1.0	unknown	unknown
8	_HACK_G_U_VOL_2_REMINISCE	PS2	2006	role-playing	0.11	0.09	0.00	0.03	-1	-1.0	unknown	unknown

После категоризации данных проверим результат: группируем данные по выделенной категории и считаем количество игр в каждой категории.

In [103…
```
# выведем количество записей по каждой категории
df_actual['user_rating'].value_counts().sort_values()
```

Out[103…
```
low         118
high       2308
medium     4132
unknown    6342
Name: user_rating, dtype: int64
```

In [104…
```
# выведем процент записей по каждой категории
df_actual['user_rating'].value_counts(normalize=True).sort_values() * 100
```

Out[104…
```
low         0.914729
high       17.891473
medium     32.031008
unknown    49.162791
Name: user_rating, dtype: float64
```

Видим следующее распределение по категории пользовательской оценки: `high` - 2308 (18%); `medium` - 4132 (32%); `low` - 118 (1%); `unknown` - 6342 (49%).

4.2. Категоризация данных по полю `critic_score`

Разделим все игры по оценкам критиков и выделите такие категории: высокая оценка (от 80 до 100 включительно), средняя оценка (от 30 до 80, не включая правую границу интервала) и низкая оценка (от 0 до 30, не включая правую границу интервала).

In [105…
```
# Функция для категоризации
def critic_categorize(value):
    if 0 <= value < 30:
        return 'low'
    elif 30 <= value < 80:
        return 'medium'
    elif 80 <= value <= 100:
        return 'high'
    else:
        return 'unknown'

# Применение функции к данным
df_actual['critic_rating'] = df_actual['critic_score'].apply(critic_categorize)

# выведем первые пять строк
df_actual.head()
```

Out[105…

	name	platform	year_of_release	genre	na_sales	eu_sales	jp_sales	other_sales	critic_score	user_score	rating	user_rating	critic_rating
2	FROZEN_OLAF_S_QUEST	3DS	2013	platform	0.27	0.27	0.00	0.05	-1	-1.0	unknown	unknown	unknown
3	FROZEN_OLAF_S_QUEST	DS	2013	platform	0.21	0.26	0.00	0.04	-1	-1.0	unknown	unknown	unknown
5	TALES_OF_XILLIA_2	PS3	2012	role-playing	0.20	0.12	0.45	0.07	71	7.9	T	medium	medium
7	_HACK_G_U_VOL_1_REBIRTH	PS2	2006	role-playing	0.00	0.00	0.17	0.00	-1	-1.0	unknown	unknown	unknown
8	_HACK_G_U_VOL_2_REMINISCE	PS2	2006	role-playing	0.11	0.09	0.00	0.03	-1	-1.0	unknown	unknown	unknown

После категоризации данных проверим результат: группируем данные по выделенной категории и считаем количество игр в каждой категории.

In [106…
```
# выведем количество записей по каждой категории
df_actual['critic_rating'].value_counts().sort_values()
```

Out[106…
```
low          57
high       1711
medium     5491
unknown    5641
Name: critic_rating, dtype: int64
```

In [107…
```
# выведем процент записей по каждой категории
df_actual['critic_rating'].value_counts(normalize=True).sort_values() * 100
```

Out[107…
```
low         0.441860
high       13.263566
medium     42.565891
unknown    43.728682
Name: critic_rating, dtype: float64
```

Видим следующее распределение по категории оценки критиков: `high` - 1711 (13%); `medium` - 5491 (43%); `low` - 57 (менее 1%); `unknown` - 5641 (44%).

4.3. Сравнение оценок критиков и пользователей

Напомним распределения оценок.

Оценки пользователей: `high` - 2308 (18%); `medium` - 4132 (32%); `low` - 118 (1%); `unknown` - 6342 (49%).

Оценки критиков: `high` - 1711 (13%); `medium` - 5491 (43%); `low` - 57 (менее 1%); `unknown` - 5641 (44%).

Можно сделать следующие выводы:

- Играм критики и пользователи очень редко ставят крайне низкую оценку (порядка 1%), т. е. подавляющее большинство игр находит свою аудиторию.
- Критики (13%) более сдержаны в высоких оценках игр нежели игроки (18%).
- Средние оценки критики поставили 43% всех игр, а игроки - 32%.
- Существенная часть данных не имеет информации об оценках пользователями (49%) и критиками (44%).
- Если бы заполнили пропуски средними значениями, то это очевидно просто увеличило бы количество средних оценок.

- Рекомендуется провести дополнительное исследование о причине такого большого количества пропусков в оценках игр: связано ли это с непопулярностью игр (низкими продажами в мире).

4.4. Топ-7 платформ по количеству игр

Выделим топ-7 игровых платформ по количеству игр, выпущенных за весь актуальный период.

In [108…
```
# создадим серию total_count_by_platform с количеством игр на каждой платформы
total_count_by_platform = df_actual['platform'].value_counts()

# отсортируем по убыванию количества игр
total_count_by_platform = total_count_by_platform.sort_values(ascending=False)

# выводим топ-7 игровых платформ
total_count_by_platform.head(7)
```

Out[108…
```
PS2     2140
DS      2128
Wii     1290
PSP     1190
X360    1139
PS3     1099
XB       817
Name: platform, dtype: int64
```

In [109…
```
# создадим серию total_per_by_platform с процентом игр на каждой платформы
total_per_by_platform = df_actual['platform'].value_counts(normalize=True) * 100

# отсортируем по убыванию количества игр
total_per_by_platform = total_per_by_platform.sort_values(ascending=False)

# выводим топ-7 игровых платформ
total_per_by_platform.head(7)
```

Out[109…
```
PS2     16.589147
DS      16.496124
Wii     10.000000
PSP      9.224806
X360     8.829457
PS3      8.519380
XB       6.333333
Name: platform, dtype: float64
```

В топе-7 места распределились следующий образом:

1. PS2 - 2140 шт. - 16.6%
2. DS - 2128 шт. - 16.5%
3. Wii - 1290 шт. - 10.0%
4. PSP - 1190 шт. - 9.2%
5. X360 - 1139 шт. - 8.8%
6. PS3 - 1099 шт. - 8.5%
7. XB - 817 шт. - 6.3%.

Результат закономерный:

- Лидерами являются консоли PS2 и DS, чей жизненный цикл почти целиком укладывается в срез с 2000 по 2013, у консолей Wii, PS3 и Xbox360 большое количество игр выходило после 2013.
- 1 место занимает PS2, но отрыв от DS на 2 месте составляет лишь 0.1%.
- 3-6 места также распеределены с небольшими разрывами друг от друга с размахом в 1.5%.
- Интересно сравнить между собой одинаковые поколения консолей:
  - PS2 (1 место, 16.6%) и XB (7 место, 6.3%): их жизненный цикл укладывается в срез, но игр на PS2 было существенно больше, Microsoft только заходило на рынок игровых консолей.
  - Wii (3 место, 10.0%), Xbox360 (5 место 8.8%) и PS3 (6 место, 8.5%): Wii имеет больше эксклюзивов за счет уникального управления с пультами, Xbox360 изменил свою стратегию на игровом рынке и увеличил долю эксклюзивов, сумев несколько обогнать PS3.
  - DS (2 место, 16.5%) и PSP (4 место, 9.2%): на DS было существенно больше эксклюзивов, когда как на PSP было много портированных игр с PS и PS2.
- PC в топ-7 по количеству игр не попал из-за отсутствия эксключивов.
- Именно наличие своих эксклюзивов для каждой консоли стимулировало продажи самих консолей, в будущем у компаний поменялась стратегия.

---

5. Итоговый вывод

Были загружены данные датасета `new_games.csv`. Они содержали 11 столбцов и 16956 строк, в которых представлена информация о компьютерных играх.

При первичном знакомстве с данными и их предобработке получили такие результаты:

- Привели все столбцы к стилю "snake case".
- Проверили некорректные типы данных:
  - Преобразовали поле `year_of_release` к целочисленному типу с уменьшением размерности, в 127 пропусках восстановили настоящие значения и 148 пропусков заменили нулями.
  - Преобразовали поля `eu_sales`, `jp_sales`, `critic_score` и `user_score` к вещественному типу с уменьшением размерности, значения все данные, которые не удалось преобразовать в вещественные числа, заменили пропусками.
- Обработали пропуски в данных:
  - Пропуски в полях `name` и `genre` обнаружены лишь в 2 записях, которые были удалены. Осталось 16954 строки.
  - Пропуски в полях `eu_sales` и `jp_sales` обнаружены в 10 записях, которые оставили пустыми.
  - Пропуски в полях `critic_score` и `user_score` содержало порядка 54.6% записей. Пропускам присвоили значение-индикатор `-1`.
  - Поле `critic_score` преобразовали в целочисленный тип `int8`.
  - Пропуски в поле `rating` составляли порядка 41% записей, пропускам присвоили значение `unknown`.
  - Все данные изменения записаны в датафрейме `df`.
- Выявили и устранили явные и неявные дубликаты:
  - В полях `platform` и `year_of_release` неявных дубликатов не было.
  - Поле `genre` имело 24 различных значения, после приведения к нижнему регистру осталось 12 уникальных значений.
  - Поле `rating` имело 9 различных значений, включая `unknown`. Заменили 3 значения старого обозначения `K-A` на новое обозначение `E`, осталось 8 различных уникальных значения.
  - В поле `name` все значения привели в верхнему регистру, удалили пробелы перед и после строки, с помощью регулярных выражений заменили все символы на нижнее подчеркивание.
  - До удаления дубликатов записей было 16954.
  - Были выявлены 2 пары записей являющихся неявными дубликатами по названию, платформе и году релиза. Каждую пару неявных дубликатов заменили на одну запись.
  - После выявления возможных неявных дубликатов были удалены явные дубликаты.
  - Итого было удалено 243 строк (порядка 1.4% от общего числа), осталось 16711 строк.
  - Записали все строки без дубликатов датафрейм `df_uniq`.

Отфильтровали данные за период с 2000 по 2013 год включительно: срез сохранили в датафрейм `df_actual`.

Категоризовали данные:

- Категории по пользовательским оценкам `user_score`: создали новое поле `user_rating`: `high` (от 8 до 10 включительно), `medium` (от 3 до 8, не включая правую границу интервала), `low` (от 0 до 3, не включая правую границу интервала) и `unknown` (в остальных случаях).

- Категории по пользовательским оценкам `critic_score`: создали новое поле `critic_rating`: `high` (от 80 до 100 включительно), `medium` (от 30 до 80, не включая правую границу интервала), `low` (от 0 до 30, не включая правую границу интервала) и `unknown` (в остальных случаях).
- Распределения оценок было следующим:
  - Оценки пользователей: `high` - 2544 (20%); `medium` - 5475 (42%); `low` - 138 (1%); `unknown` - 4743 (37%).
  - Оценки критиков: `high` - 1897 (15%); `medium` - 6177 (48%); `low` - 83 (менее 1%); `unknown` - 4743 (37%).
- Можно сделать следующие выводы:
  - Играм критики и пользователи очень редко ставят крайне низкую оценку (порядка 1%), т. е. подавляющее большинство игр находит свою аудиторию.
  - Критики (13%) более сдержаны в высоких оценках игр нежели игроки (18%).
  - Средние оценки критики поставили 43% всех игр, а игроки - 32%.
  - Существенная часть данных не имеет информации об оценках пользователями (49%) и критиками (44%).
  - Если бы заполнили пропуски средними значениями, то это очевидно просто увеличило бы количество средних оценок.
  - Рекомендуется провести дополнительное исследование о причине такого большого количества пропусков в оценках игр: связано ли это с непопулярностью игр (низкими продажами в мире).
- Новые поля записаны были в датафрейм `df_actual`.

Составили топ-7 платформ по продажам игр:

- В топе-7 места распределились следующий образом:
  1. PS2 - 2140 шт. - 16.6%
  2. DS - 2128 шт. - 16.5%
  3. Wii - 1290 шт. - 10.0%
  4. PSP - 1190 шт. - 9.2%
  5. X360 - 1139 шт. - 8.8%
  6. PS3 - 1099 шт. - 8.5%
  7. XB - 817 шт. - 6.3%.
- Результат закономерный: лидерами являются консоли PS2 и DS, чей жизненный цикл почти целиком укладывается в срез с 2000 по 2013, у консолей Wii, PS3 и Xbox360 большое количество игр выходило после 2013.
- 1 место занимает PS2, но отрыв от DS на 2 месте составляет лишь 0.1%.
- 3-6 места также распеределены с небольшими разрывами друг от друга с размахом в 1.5%.
- Интересно сравнить между собой одинаковые поколения консолей:
  - PS2 (1 место, 16.6%) и XB (7 место, 6.3%): их жизненный цикл укладывается в срез, но игр на PS2 было существенно больше, Microsoft только заходило на рынок игровых консолей.
  - Wii (3 место, 10.0%), Xbox360 (5 место 8.8%) и PS3 (6 место, 8.5%): Wii имеет больше эксклюзивов за счет уникального управления с пультами, Xbox360 изменил свою стратегию на игровом рынке и увеличил долю эксклюзивов, сумев несколько обогнать PS3.
  - DS (2 место, 16.5%) и PSP (4 место, 9.2%): на DS было существенно больше эксклюзивов, когда как на PSP было много портированных игр с PS и PS2.
- PC в топ-7 по количеству игр не попал из-за отсутствия эксключивов.
- Именно наличие своих эксклюзивов для каждой консоли стимулировало продажи самих консолей, в будущем у компаний поменялась стратегия.