1. The ten best-selling video games

Photo by Dan Schleusser on Unsplash.

Video games are big business: the global gaming market is projected to be worth more than $300 billion by 2027 according to Mordor Intelligence. With so much money at stake, the major game publishers are hugely incentivized to create the next big hit. But are games getting better, or has the golden age of video games already passed?

In this project, we'll explore the top 400 best-selling video games created between 1977 and 2020. We'll compare a dataset on game sales with critic and user reviews to determine whether or not video games have improved as the gaming market has grown.

Our database contains two tables. We've limited each table to 400 rows for this project, but you can find the complete dataset with over 13,000 games on Kaggle.

`game_sales`

column	type	meaning
`game`	varchar	Name of the video game
`platform`	varchar	Gaming platform
`publisher`	varchar	Game publisher
`developer`	varchar	Game developer
`games_sold`	float	Number of copies sold (millions)
`year`	int	Release year

`reviews`

column	type	meaning
`game`	varchar	Name of the video game
`critic_score`	float	Critic score according to Metacritic
`user_score`	float	User score according to Metacritic

Let's begin by looking at some of the top selling video games of all time!

In [2]:
```
%%sql
postgresql:///games

-- Select all information for the top ten best-selling games
-- Order the results from best-selling game down to tenth best-selling
SELECT *
FROM game_sales
ORDER BY games_sold DESC
LIMIT 10;
```

10 rows affected.

Out[2]:

game	platform	publisher	developer	games_sold	year
Wii Sports for Wii	Wii	Nintendo	Nintendo EAD	82.90	2006
Super Mario Bros. for NES	NES	Nintendo	Nintendo EAD	40.24	1985
Counter-Strike: Global Offensive for PC	PC	Valve	Valve Corporation	40.00	2012
Mario Kart Wii for Wii	Wii	Nintendo	Nintendo EAD	37.32	2008
PLAYERUNKNOWN'S BATTLEGROUNDS for PC	PC	PUBG Corporation	PUBG Corporation	36.60	2017
Minecraft for PC	PC	Mojang	Mojang AB	33.15	2010
Wii Sports Resort for Wii	Wii	Nintendo	Nintendo EAD	33.13	2009
Pokemon Red / Green / Blue Version for GB	GB	Nintendo	Game Freak	31.38	1998
New Super Mario Bros. for DS	DS	Nintendo	Nintendo EAD	30.80	2006
New Super Mario Bros. Wii for Wii	Wii	Nintendo	Nintendo EAD	30.30	2009

In [3]:
```
%%nose
from decimal import Decimal as D
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (10, 6), \
    "The results should have six columns and ten rows."
    assert results.columns.tolist() == ["game", "platform", "publisher", "developer", "games_sold", "year"], \
    'The results should have columns named "game", "platform", "publisher", "developer", "games_sold", and "year".'
    assert _.DataFrame().loc[0, 'games_sold'] == D('82.90')
    "The top selling game should be Wii Sports with 82.90 million copies sold."
```

Out[3]: 2/2 tests passed

2. Missing review scores

Wow, the best-selling video games were released between 1985 to 2017! That's quite a range; we'll have to use data from the `reviews` table to gain more insight on the best years for video games.

First, it's important to explore the limitations of our database. One big shortcoming is that there is not any `reviews` data for some of the games on the `game_sales` table.

In [4]:
```
%%sql

-- Join games_sales and reviews
-- Select a count of the number of games where both critic_score and user_score are null
SELECT COUNT(gs.game)
FROM game_sales as gs
LEFT JOIN reviews as r
ON gs.game = r.game
WHERE r.critic_score IS NULL AND r.user_score IS NULL;
```

 * postgresql:///games
1 rows affected.

Out[4]:

count
31

In [5]:
```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (1, 1), \
    "The query should return just one value, a count of games where both critic_score and user_score are null."
    assert results.columns.tolist() == ["count"], \
    'The results should have just one column, called "count".'
    assert last_output.DataFrame().loc[0, 'count'] == 31, \
    "There should be 31 games where both critic_score and user_score are null."
```

Out[5]: 2/2 tests passed

3. Years that video game critics loved

It looks like a little less than ten percent of the games on the `game_sales` table don't have any reviews data. That's a small enough percentage that we can continue our exploration, but the missing reviews data is a good thing to keep in mind as we move on to evaluating results from more sophisticated queries.

There are lots of ways to measure the best years for video games! Let's start with what the critics think.

In [6]:
```
%%sql

-- Select release year and average critic score for each year, rounded and aliased
-- Join the game_sales and reviews tables
-- Group by release year
-- Order the data from highest to lowest avg_critic_score and limit to 10 results
SELECT year, ROUND(AVG(r.critic_score),2) AS avg_critic_score
FROM game_sales as gs
LEFT JOIN reviews as r
ON gs.game = r.game
GROUP BY year
ORDER BY avg_critic_score DESC
LIMIT 10;
```

 * postgresql:///games
10 rows affected.

Out[6]:

year	avg_critic_score
1990	9.80
1992	9.67
1998	9.32
2020	9.20
1993	9.10
1995	9.07
2004	9.03
1982	9.00
2002	8.99
1999	8.93

In [7]:
```
%%nose
from decimal import Decimal as D
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (10, 2), \
    "Make sure to limit the query to only ten results."
    assert results.columns.tolist() == ["year", "avg_critic_score"], \
    'The results should have two columns, called "year" and "avg_critic_score".'
    assert last_output.DataFrame().loc[0, 'year'] == 1990, \
    "The year with the highest score should be 1990."
    assert last_output.DataFrame().loc[0, 'avg_critic_score'] == D('9.80'), \
    "The highest average critic score should be 9.80."
```

Out[7]: 2/2 tests passed

4. Was 1982 really that great?

The range of great years according to critic reviews goes from 1982 until 2020: we are no closer to finding the golden age of video games!

Hang on, though. Some of those `avg_critic_score` values look like suspiciously round numbers for averages. The value for 1982 looks especially fishy. Maybe there weren't a lot of video games in our dataset that were released in certain years.

Let's update our query and find out whether 1982 really was such a great year for video games.

In [8]:
```
%%sql

-- Paste your query from the previous task; update it to add a count of games released in each year called num_games
-- Update the query so that it only returns years that have more than four reviewed games
SELECT gs.year, COUNT(gs.game) AS num_games, ROUND(AVG(r.critic_score),2) AS avg_critic_score
FROM game_sales as gs
INNER JOIN reviews as r
ON gs.game = r.game
GROUP BY gs.year
HAVING COUNT(gs.game) > 4
ORDER BY avg_critic_score DESC
LIMIT 10;
```

 * postgresql:///games
10 rows affected.

Out[8]:

year	num_games	avg_critic_score
1998	10	9.32
2004	11	9.03
2002	9	8.99
1999	11	8.93
2001	13	8.82
2011	26	8.76
2016	13	8.67
2013	18	8.66
2008	20	8.63
2017	13	8.62

In [9]:
```
%%nose
from decimal import Decimal as D
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (10, 3), \
    "Make sure to limit the query to only ten results."
    assert set(last_output.DataFrame().columns) == set(["year", "num_games", "avg_critic_score"]), \
    'The results should have three columns: "year", "num_games", and "avg_critic_score".'
    assert last_output.DataFrame().loc[0, 'year'] == 1998, \
    "The year with the highest score should be 1998."
    assert last_output.DataFrame().loc[0, 'num_games'] == 10, \
    "In the year with the highest critic score, there were 10 games released."
    assert last_output.DataFrame().loc[0, 'avg_critic_score'] == D('9.32'), \
    "The highest average critic score should be 9.32."
```

Out[9]: 2/2 tests passed

5. Years that dropped off the critics' favorites list

That looks better! The `num_games` column convinces us that our new list of the critics' top games reflects years that had quite a few well-reviewed games rather than just one or two hits. But which years dropped off the list due to having four or fewer reviewed games? Let's identify them so that someday we can track down more game reviews for those years and determine whether they might rightfully be considered as excellent years for video game releases!

It's time to brush off your set theory skills. To get started, we've created tables with the results of our previous two queries:

`top_critic_years`

column	type	meaning
`year`	int	Year of video game release
`avg_critic_score`	float	Average of all critic scores for games released in that year

`top_critic_years_more_than_four_games`

column	type	meaning
`year`	int	Year of video game release
`num_games`	int	Count of the number of video games released in that year
`avg_critic_score`	float	Average of all critic scores for games released in that year

In [10]:
```
%%sql

-- Select the year and avg_critic_score for those years that dropped off the list of critic favorites
-- Order the results from highest to lowest avg_critic_score
SELECT year, avg_critic_score
FROM top_critic_years
EXCEPT
SELECT year, avg_critic_score
FROM top_critic_years_more_than_four_games
ORDER BY avg_critic_score DESC;
```

 * postgresql:///games
6 rows affected.

Out[10]:

year	avg_critic_score
1990	9.80
1992	9.67
2020	9.20
1993	9.10
1995	9.07
1982	9.00

In [11]:
```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (6, 2), \
    "There should be six years that dropped off the critics' favorite list after implementing the criteria that the year had to have at least five games released to be considered."
    assert results.columns.tolist() == ["year", "avg_critic_score"], \
    'The results should have two columns: "year" and "avg_critic_score".'
    assert last_output.DataFrame().loc[5, 'year'] == 1982, \
    "The last year returned by the query should be 1982."
    assert last_output.DataFrame().loc[5, 'avg_critic_score'] == 9.00, \
    "1982's average critic score should be 9.00."
```

Out[11]: 2/2 tests passed

6. Years video game players loved

Based on our work in the task above, it looks like the early 1990s might merit consideration as the golden age of video games based on `critic_score` alone, but we'd need to gather more games and reviews data to do further analysis.

Let's move on to looking at the opinions of another important group of people: players! To begin, let's create a query very similar to the one we used in Task Four, except this one will look at `user_score` averages by year rather than `critic_score` averages.

In [12]:
```
%%sql

-- Select year, an average of user_score, and a count of games released in a given year, aliased and rounded
-- Include only years with more than four reviewed games; group data by year
-- Order data by avg_user_score, and limit to ten results
SELECT gs.year, COUNT(gs.game) AS num_games, ROUND(AVG(r.user_score),2) AS avg_user_score
FROM game_sales as gs
INNER JOIN reviews as r
ON gs.game = r.game
GROUP BY gs.year
HAVING COUNT(gs.game) > 4
ORDER BY avg_user_score DESC
LIMIT 10;
```

 * postgresql:///games
10 rows affected.

Out[12]:

year	num_games	avg_user_score
1997	8	9.50
1998	10	9.40
2010	23	9.24
2009	20	9.18
2008	20	9.03
1996	5	9.00
2005	13	8.95
2006	16	8.95
2000	8	8.80
2002	9	8.80

In [13]:
```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (10, 3), \
    "Don't forget to limit the query results to ten."
    assert set(results.columns.tolist()) == set(["year", "num_games", "avg_user_score"]), \
    'The results should have three columns: "year", "num_games", and "avg_user_score".'
    assert last_output.DataFrame().loc[0, 'year'] == 1997, \
    "The year with the highest user score should be 1997."
    assert last_output.DataFrame().loc[0, 'num_games'] == 8, \
    "In the year with the highest user score, there were eight games released."
    assert last_output.DataFrame().loc[0, 'avg_user_score'] == 9.50, \
    "The highest average user score should be 9.50."
```

Out[13]: 2/2 tests passed

7. Years that both players and critics loved

Alright, we've got a list of the top ten years according to both critic reviews and user reviews. Are there any years that showed up on both tables? If so, those years would certainly be excellent ones!

Recall that we have access to the `top_critic_years_more_than_four_games` table, which stores the results of our top critic years query from Task 4:

`top_critic_years_more_than_four_games`

column	type	meaning
`year`	int	Year of video game release
`num_games`	int	Count of the number of video games released in that year
`avg_critic_score`	float	Average of all critic scores for games released in that year

We've also saved the results of our top user years query from the previous task into a table:

`top_user_years_more_than_four_games`

column	type	meaning
`year`	int	Year of video game release
`num_games`	int	Count of the number of video games released in that year
`avg_user_score`	float	Average of all user scores for games released in that year

In [14]:
```
%%sql

-- Select the year results that appear on both tables
SELECT year
FROM top_critic_years_more_than_four_games
INTERSECT
SELECT year
FROM top_user_years_more_than_four_games;
```

 * postgresql:///games
3 rows affected.

Out[14]:

year
1998
2008
2002

In [15]:
```
%%nose
last_output = _

def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (3, 1), \
    "There should be three years present in both tables."
    assert results.columns.tolist() == ["year"], \
    'The results should just have one column: "year".'
    assert last_output.DataFrame().loc[0, 'year'] == 1998, \
    "The first year returned by the query should be 1998."
```

Out[15]: 2/2 tests passed

8. Sales in the best video game years

Looks like we've got three years that both users and critics agreed were in the top ten! There are many other ways of measuring what the best years for video games are, but let's stick with these years for now. We know that critics and players liked these years, but what about video game makers? Were sales good? Let's find out.

This time, we haven't saved the results from the previous task in a table for you. Instead, we'll use the query from the previous task as a subquery in this one! This is a great skill to have, as we don't always have write permissions on the database we are querying.

In [16]:
```
%%sql

-- Select year and sum of games_sold, aliased as total_games_sold; order results by total_games_sold descending
-- Filter game_sales based on whether each year is in the list returned in the previous task
SELECT year, SUM(games_sold) as total_games_sold
FROM game_sales
WHERE year IN (SELECT year
FROM top_critic_years_more_than_four_games
INTERSECT
SELECT year
FROM top_user_years_more_than_four_games)
GROUP BY year
ORDER BY total_games_sold DESC;
```

 * postgresql:///games
3 rows affected.

Out[16]:

year	total_games_sold
2008	175.07
1998	101.52
2002	58.67

In [17]:
```
%%nose
from decimal import Decimal as D
last_output = _


def test_output_type():
    assert str(type(last_output)) == "<class 'sql.run.ResultSet'>", \
    "Please ensure an SQL ResultSet is the output of the code cell."

results = last_output.DataFrame()

def test_results():
    assert results.shape == (3, 2), \
    "There should be games sales data for three years: the same three years from the previous query."
    assert results.columns.tolist() == ["year", "total_games_sold"], \
    'The results should have two columns: "year" and "total_games_sold".'
    assert last_output.DataFrame().loc[0, 'year'] == 2008, \
    "Just like in the last query, the first year returned should be 2008."
    assert last_output.DataFrame().loc[0, 'total_games_sold'] == D('175.07'), \
    "In 2008, the total_games_sold value should be 175.07."
```

Out[17]: 2/2 tests passed