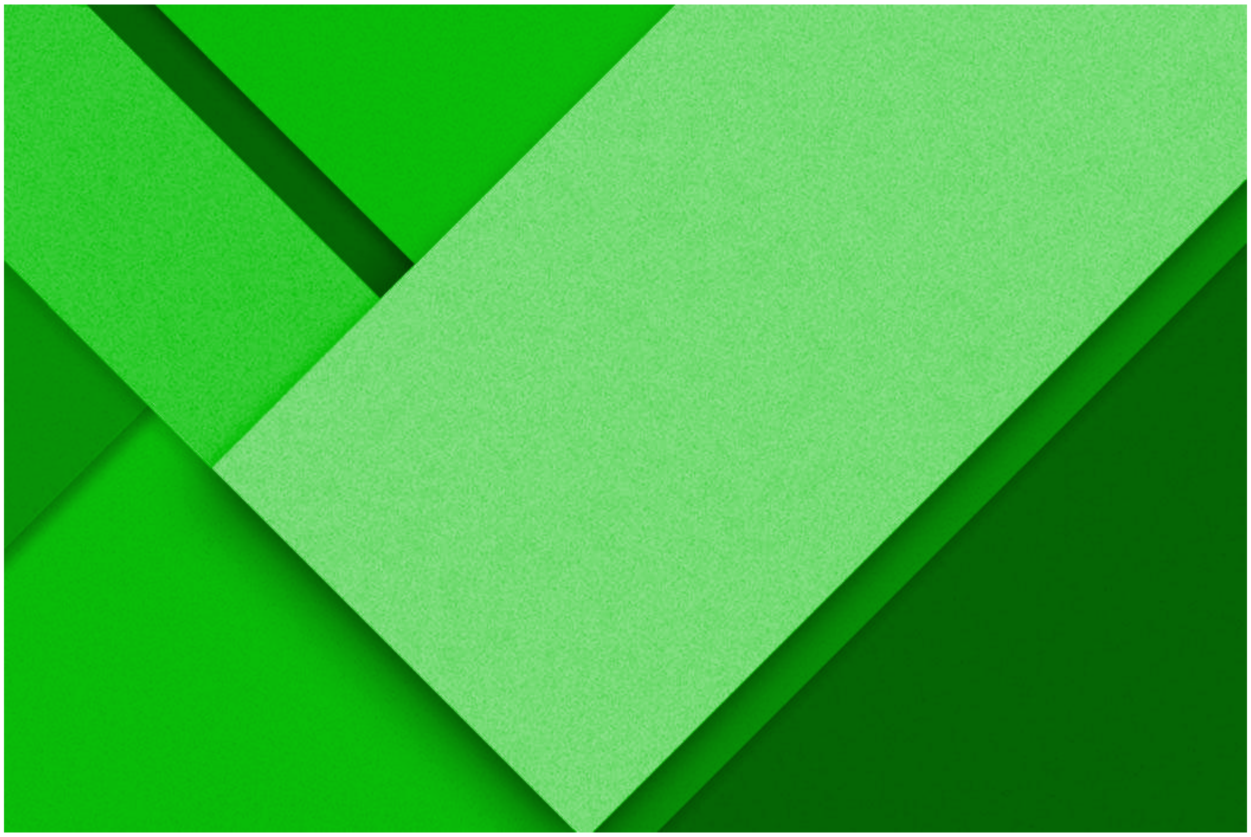


GamifyDB

09.02.2024

Application developed as a part of the Information Integration course at the University of Stuttgart by:

Vedant Puranik (3703786)

Jan Bothmann (3634237)

Mugdha Asgekar (3697957)

Overview

This project was developed as an effort to consolidate data about video games released between 1980 to 2016. Additionally, we also combine the sales figures across continents for each game to create an application that allows stakeholders to query and analyze the data for making informed decisions.

Goals

1. Extract data about the games from IMDb (Internet Movie Database)
2. Combine the sales and information about the games into a single view.
3. Extract data about the developers from IGDB (Internet Game Database)
4. Clean the data.
5. Develop an application to serve the data.
6. Provide visualization options to get quick insights into the game.

Installation Guide

Skip to the next page in case you have prerequisites installed.

(Prerequisite) Setup PostgreSQL:

1. Visit the official PostgreSQL website: https://www.postgresql.org/download/
2. Select the OS as per your current workstation
3. Download the latest version of PostgreSQL (Version 16 at the time of writing of this document).
4. Follow the installer guide to complete setup.
5. Ensure that you set the port to 5432 (default port) while installing Postgres and remember the password.
6. Once installed, verify the installation by using the following command: “psql -U postgres”. Then enter the password that you used while installing. This should start a postgres prompt. This indicates installation was successful.

(Prerequisite) Setup Python:

1. Visit the official Python download page: https://www.python.org/downloads/
2. Download the latest version of Python (3.12.0 at the time of writing of this document) by choosing the OS as per your current workstation.
3. For Windows:
    a. Set “Use admin privileges when installing py.exe” and “Add python.exe to PATH”.
    b. Click on Install Now or Customize Installation as per your requirement.
    c. In case of Customize Installation, Select “Install Python 3.xx for all users”, “Precompile standard library” in addition to existing set options under Advanced Options.
    d. Click on Install.
    e. Verify Installation by opening the command prompt and writing “python –version”.
4. For other Linux: Python is already installed
5. For mac: Follow this guide: https://macpaw.com/how-to/install-python-mac#:~:text=The%20Python%20programming%20language%20is,need%20to%20install%20it%20yourself.

Install GamifyDB application:

1. Clone the official git repository to your system: `git clone https://github.com/VedantKP/GamifyDB`
2. Replace the "<password>" string with your postgres password in the following files:
   a. data-extractor/create_schema.py: in the dbDetails object on line 3.
   b. data-extractor/insert_data_db.py: in the connString on line 7.
   c. data-extractor/readAndMergeIntoCsv.py: in the db_url string on line.
   d. flask-app/app.py: in the app.config['DATABASE_URI'] string on line 6.
3. Open a command prompt.
4. Ensure your working directory is the cloned repository's folder: `GamifyDB`
5. Install python's virtualenv package: `pip install virtualenv`
6. Create a virtual environment: `virtualenv env`
7. Activate virtual environment:
   a. For Windows: `.\env\Scripts\activate`
   b. For Mac and Linux: `source env/bin/activate`
8. Install all required packages: `pip install -r requirements.txt`
9. Change working directory to data-extractor: `cd data-extractor`
10. Create Schema in postgres: `python create_schema.py`
11. Integrate all the data (Includes duplicate detection and fusion): `python integrate_data.py`
12. Insert data into the database: `python insert_data_db.py`
13. Change working directory to flask-app: `cd ..\flask-app\`
14. Run the flask app: `python app.py`
15. Click on the link given in the console output or paste https://localhost:5000 in the browser to open the interface to the application.

Details of files used to perform integration and run the application

Files related to data integration

1. data-extractor/create_schema.py: This file holds the code necessary to create the schema in the postgres database, including the database, tables and comments.
2. data-extractor/integrate_data.py: All the code to read the data, clean the data, find duplicates and integrate the data is included in the file.
3. data-extractor/insert_data_db.py: This file reads the integrated data and inserts the data into the game, company and game_developers tables.
4. data-extractor/pull_igdb.py: This file contains the code to extract developer data using the IGDB API. Note: This data was extracted in parts over a period of time due to API call restrictions posed by IGDB. All the extracted data was then concatenated to form the final dataset. DO NOT execute this file as all the extracted data is available in datasets/allCompanyData.csv file.
5. data-extractor/readAndMergeIntoCsv.py: This file creates a CSV file from the extract of the data in the database, later used for visualization purposes.
6. datasets/: This folder has:
    a. Source datasets: imdb-videogames.csv, vgsales.csv
    b. Datasets created after integration: globalDF.csv, allCompanyData.csv
    c. All intermediate datasets created during IGDB Extraction and testing of the code (Retained only for information purposes).

Files related to running the application

1. flask-app/app.py: This is the main Flask application file to run the entire application.
2. flask-app/templates/: This folder contains all the html files
3. flask-app/static/: This folder contains the css and javascript files used by the html files when rendering data.
4. requirements.txt: This file has all the python modules required to execute the project.

Description of data sources

#	Name	Source	Type	Data
1	Internet Game Database	IDGB	API	JSON
2	IMDb Video Games	Kaggle	Flat file	CSV
3	Sales	DataCamp	Flat File	CSV

1. **Internet Game Database**: For each game, we extracted the data of the developers behind the company from IGDB. For this, an initial app setup is necessary on Twitch developers. Then a secret key is generated which is then used to make API calls to a total of 3 endpoints:
    a. Game endpoint: This returns IDs of all the companies involved in the development of the game.
    b. Involved companies: This endpoint returns a JSON object with the actual ID of the company.
    c. Companies: This endpoint returns the actual data about the company as explained in the data schema section of this document.
2. **IMDb Video Games**: This is a CSV file hosted on Kaggle for free use by developers. This data was extracted from IMDb and consolidated into a CSV file. The details about the content are mentioned in the data schema section of this document.
3. **Sales**: This is a CSV file hosted on DataCamp. It has information about the sales figures of several games by continent. All sales are in USD Millions. More details in the data schema section.

Schema of Data Sources

#	Name	Schema
1	Internet Game Database	[‘id’, ‘country’, ‘name’, ’start_date’, ‘url’] (Final data that we extract)
2	IMDb Video Games	['Name', 'URL', 'Certificate', 'Rating', 'Votes', 'Plot', 'Action', 'Adventure', 'Comedy', 'Crime', ‘Family, 'Fantasy', 'Mystery', 'Sci-Fi', 'Action', 'Thriller ']
3	Sales	['Rank', 'Name', 'Platform', 'Year', 'Genre', 'Publisher', 'NA_Sales', 'EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales']

IGDB endpoint response examples

Endpoint	Type	URL	Query	Response
Access token (To be executed only once)	GET	https://id.twitch.tv/oauth2/token	Headers: { “client_id”: “<client_id>”, “client_secret”: “<client_secret>”, “grant_type”: “client_credentials” } Body: NA	{ "access_token": "<token number>", "expires_in": 4888406, "token_type": "bearer" }
Games	POST	https://api.igdb.com/v4/games	Headers: { “client_id”: “<client_id>”, “Authorization”: “Bearer <access_token>” } Body: { “fields *; where slug=<slug>;” }	{... "genres": [<genreID1>, <genreID2>], "involved_companies": [<id1>, <id2>, <id3>], "keywords": ... }
Involved Companies	POST	https://api.igdb.com/v4/involved_companies	Headers: { “client_id”: “<client_id>”, “Authorization”: “Bearer <access_token>” } Body: { “fields company; where id=<id>;” }	[{ "id": <id>, "company": <companyId> }]
Companies	POST	https://api.igdb.com/v4/companies	Headers: { “client_id”: “<client_id>”, “Authorization”: “Bearer <access_token>” } Body: { “fields country,name,url,start_date; where id=<companyId>;” }	[{ "id": <id>, "country": <countryID>, "name": <companyName>, "start_date": <posixTimestamp> , "url": "<companyUrl>" }]

Data Integration Relevance

Holistic Insights: Gain a comprehensive view of the gaming landscape, incorporating information on development, sales, ratings and more. This view then allows for better-informed decisions and insights.

Accurate Insights: Combining data from multiple sources reduces bias or inaccuracies that might be present in individual datasets and provides a more reliable foundation for analysis.

Combine desired information: Specific queries cannot be answered by only one source e.g. rank the games by ranking of sales and find the company which developed them

Interesting Data Characteristics

Accessibility : Accessing data through CSV files is fairly straightforward. Although accessing data through the APIs is relatively tougher (involves app and secrets setup), it is doable, with no hindrances on cost or number of requests.

Completeness : All the required data elements needed for analysis are present including video game details, sales, ratings, prices.

Uniqueness: each video game record is uniquely represented in each data source.

Possibility to explore and analyze sales distribution of video games across different regions. (EU_Sales', 'JP_Sales', 'Other_Sales', 'Global_Sales)

Discover the popularity of games across different regions, genres over time. ('Rating', 'Plot', 'Action', 'Adventure', 'Comedy', 'Crime', ‘Family, 'Fantasy', 'Mystery', 'Sci-Fi', 'Action', 'Thriller)

Possible to identify potential relationships between ratings, genres and sales.

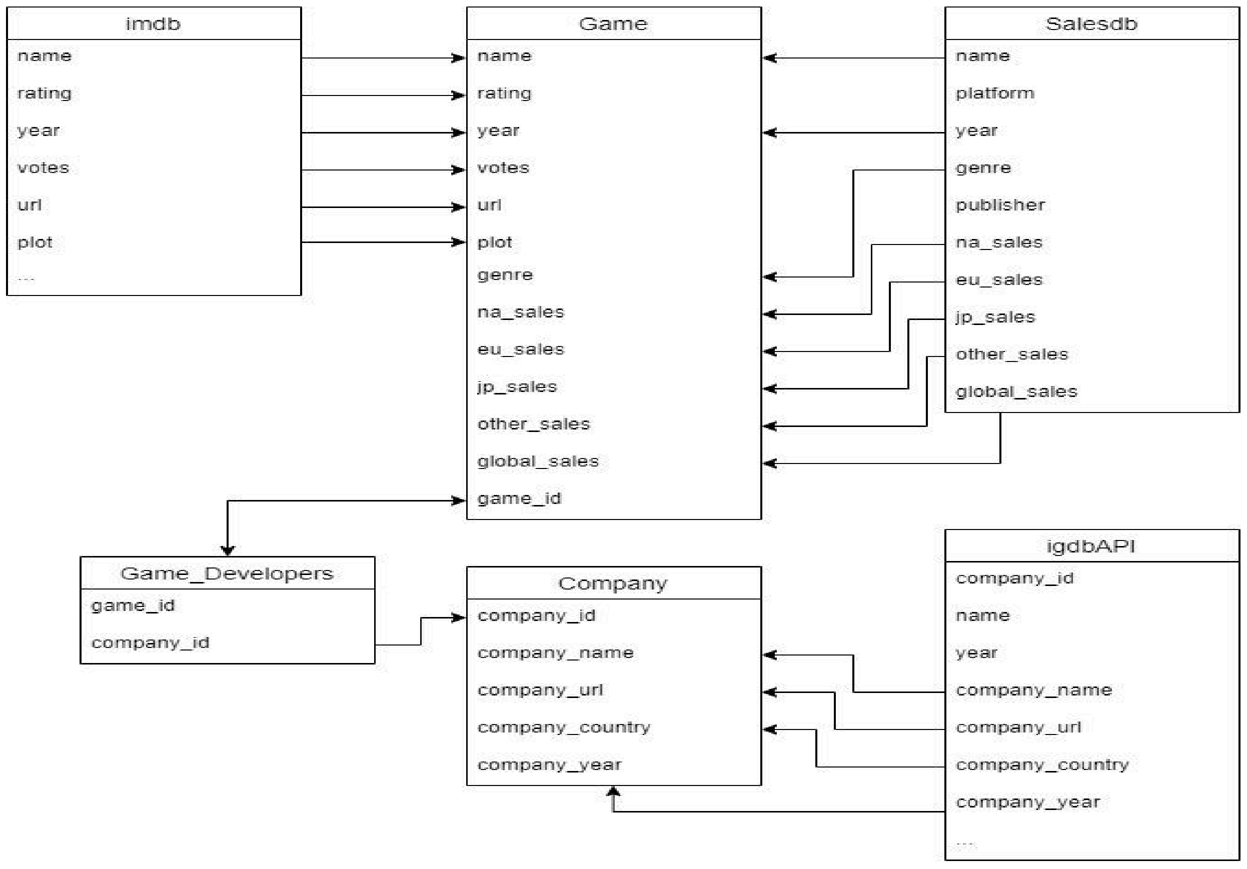
Challenges with Data Extraction

We encountered several challenges during the data extraction process. While two of our sources were CSV files, which facilitated further processing, we encountered difficulties when extracting data from the IGDB API. Initially, one had to obtain access to the API by requesting a key. Then, we needed to determine which endpoints provided by the API were relevant to our specific needs.

We evaluated all available endpoints to ascertain how we could retrieve the desired information about companies involved in various games. We noticed that companies were linked by IDs, necessitating a search for each game to determine the associated companies involved in its development. Subsequently, we had to query another API to obtain detailed information about each game.

We soon realized that this approach resulted in a high number of API request calls, and frequent querying of the IGDB API was restricted unless a subscription fee was paid. To mitigate this, we needed to limit the number of calls by only querying games that would be utilized in subsequent integration steps. Consequently, we refrained from checking for new information about games/companies in the IGDB API, thereby reducing the frequency of API queries and enabling continued operation within the constraints of the API usage policy.

Global Schema



Schema Mapping

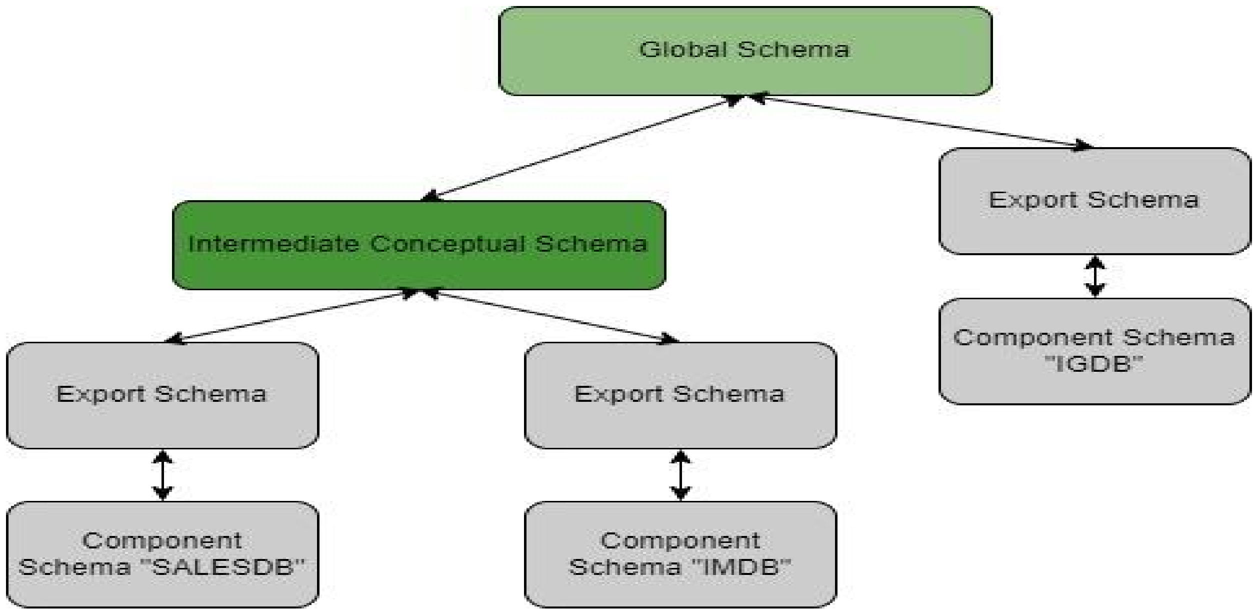
Our schema mapping approach comprised three key steps:

Global Schema Definition:
We defined a comprehensive global schema to structure the integration of data from "salesdb" and "imdb," as well as additional information from the IGDB API (see Global Schema)

Mapping Component Schemas using Jaccard Coefficient:
Next, we employed the Jaccard Coefficient to map the export schemas from the "salesdb" and "imdb" datasets to an intermediate schema. By applying this coefficient, we could identify common attributes between the component schemas and align them with corresponding attributes in the intermediate schema.

Add information about companies to each game

For each tuple representing a game entity, we initiated requests to the IGDB API to retrieve additional information about the companies involved in the development of the respective game. This additional information typically included details about the game developers, publishers, and other relevant entities. Upon receiving the API responses, we extended the intermediate schema by incorporating these additional attributes.



Data Standardization (Preprocessing)

Sales

Out of all the data available in the CSV file, we only extract the data that we need as per our global schema:

salesSchema = ['Name','Year','Genre','NA_Sales','EU_Sales','JP_Sales','Other_Sales','Global_Sales']

The Year column is represented as an “Object” internally by Pandas. To ensure this is not an issue when merging with the IMDb data, we convert it to “Int”.

IMDb

Similar to what we did in Sales, we only extract the data that we need as per our global schema:

imdbSchema = ['name','url','year','rating','votes','plot']

As we later merge this data with the data in the sales csv, we convert the “name” and “year” to “string” and “int” respectively. Also we convert the rating of each game to type “float”.

IGDB

In order to fetch a unique object for each game, we converted the name to its equivalent slug using the python module `slugify`. In case we query using the name, then IGDB returns all the games with the matching pattern. E.g. querying with name=”batman” would return all batman games as IGDB uses name pattern matching, whereas “batman” as slug is only a single game.

Country names are returned as ISO 3166-1 codes. We converted this to obtain their names using the python module `pycountry`.

Similarly, start_date of each company is returned as a POSIX timestamp. To get the actual start year we used inbuilt modules `datetime` and `timedelta`.

Duplicate Detection

We use the Sorted Neighborhood Algorithm to detect duplicates in the datasets. We detect duplicates in two different datasets: Sales and IMDb. Both of these are CSV files. The data that we extract from IGDB has only unique values as we do that after we have resolved the duplicates in the other datasets and we fetch entries for each unique game.

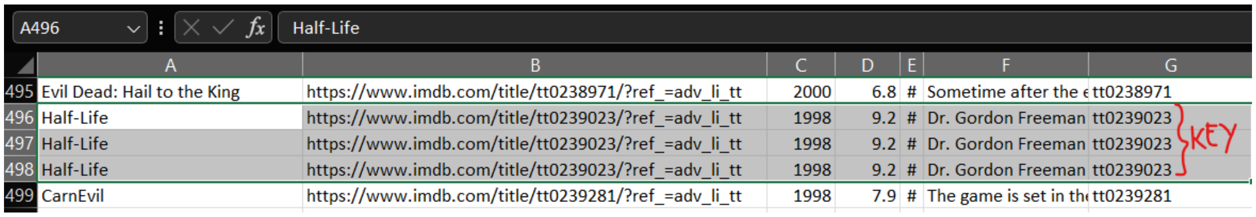
We use a window size of 3, as having a smaller and bigger window size both have their own shortcomings.

Implementation:

1. Create a key: We do this differently for both the datasets:
    a. IMDb: Used the unique identifier in the url
    b. Sales: The key is composed of the following three fields
        i. First three characters of each word in name
        ii. Last two digits of the year
        iii. First three characters of the genre
2. Sort the dataset on the basis of the key.
3. Run a window on the sorted keys.
4. Detect duplicates

Example of Duplicates:

IMDb:

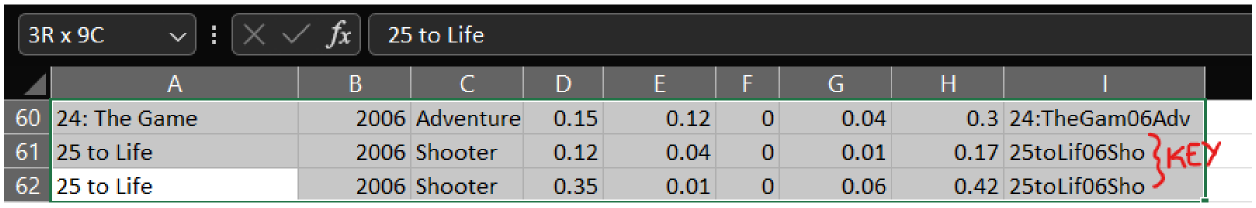


A496 | Half-Life

	A	B	C	D	E	F	G
495	Evil Dead: Hail to the King	https://www.imdb.com/title/tt0238971/?ref_=adv_li_tt	2000	6.8	#	Sometime after the e	tt0238971
496	Half-Life	https://www.imdb.com/title/tt0239023/?ref_=adv_li_tt	1998	9.2	#	Dr. Gordon Freeman	tt0239023
497	Half-Life	https://www.imdb.com/title/tt0239023/?ref_=adv_li_tt	1998	9.2	#	Dr. Gordon Freeman	tt0239023
498	Half-Life	https://www.imdb.com/title/tt0239023/?ref_=adv_li_tt	1998	9.2	#	Dr. Gordon Freeman	tt0239023
499	CarnEvil	https://www.imdb.com/title/tt0239281/?ref_=adv_li_tt	1998	7.9	#	The game is set in th	tt0239281

KEY

Sales:



3R x 9C | 25 to Life

	A	B	C	D	E	F	G	H	I
60	24: The Game	2006	Adventure	0.15	0.12	0	0.04	0.3	24:TheGam06Adv
61	25 to Life	2006	Shooter	0.12	0.04	0	0.01	0.17	25toLif06Sho
62	25 to Life	2006	Shooter	0.35	0.01	0	0.06	0.42	25toLif06Sho

KEY

Optimization of duplicate detection

There are two probable methods to optimize the runtime of the Sorted Neighborhood algorithm:

1. Use a larger window size: This will ensure that the total number of window slides are lesser and hence time required in sliding is reduced. However, the downside to this is that there could be multiple different keys' duplicates being present in the same window frame at once and hence, a more complicated way of checking for duplicates would be required.
2. Terminate algorithm earlier: This is possible in case we only need to check if there exist *any* duplicates in the data. In case there are duplicates then the algorithm can be terminated as the overall purpose of at least one duplicate detection is satisfied. However, this technique would only be applicable to particular application types/needs. It will not be possible in all cases.

According to our particular use case, we just need to detect if duplicates exist in the dataset. Once we find out the existence of the duplicates we can end the run of the algorithm. This presence can be confirmed by a single occurrence of duplicates as well. Hence, for us, it is not necessary to keep looking for each and every duplicate value by using the sorted neighborhood algorithm and then handle the data at the same point. Hence, we use the second strategy of optimization to reduce the overall runtime of the algorithm. This helps the algorithm to finish faster, helping the overall runtime of the program.

Once the duplicates are detected they are handled respectively using `pandas'` optimized built in methods as explained in the data fusion section of this document.

Data Fusion Strategy

For both the datasets, IMDb and Sales, we handle duplicates in different ways

IMDb:

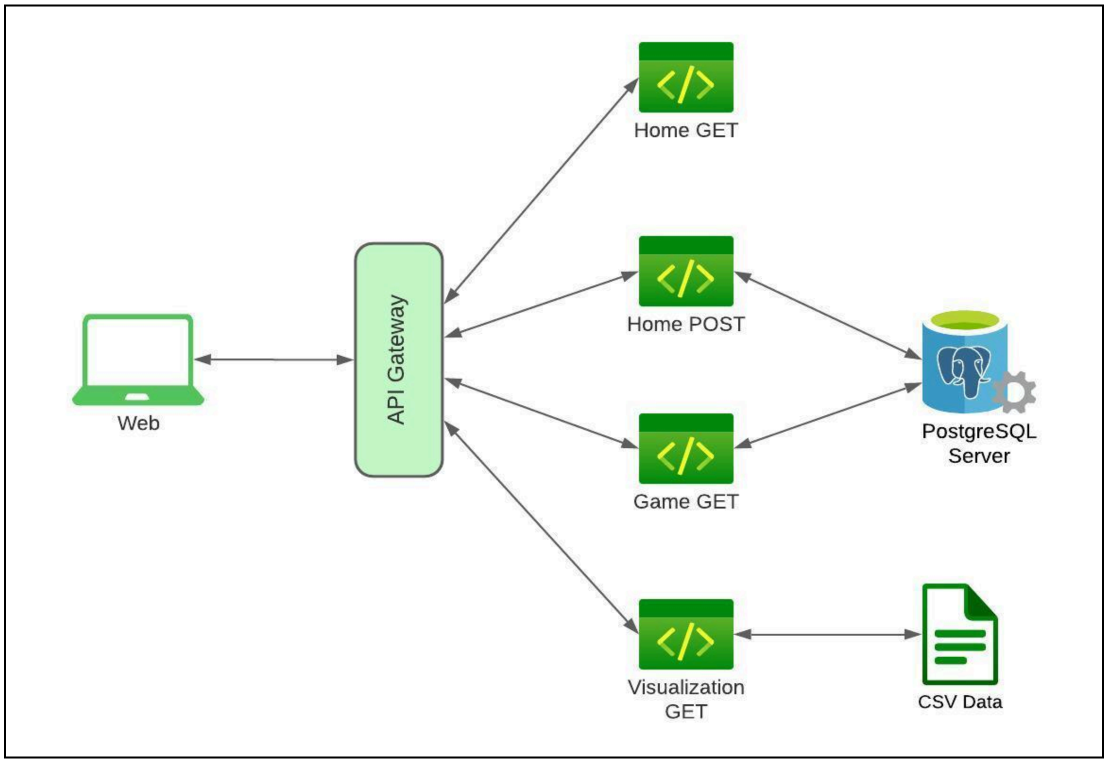
Each record in IMDb that has the same key as detected by the sorted neighborhood algorithm, is the exact same entity. The data in all the columns of the instance is exactly the same. Hence, dropping the duplicates and retaining only one instance does not result in loss of any information. These duplicates are dropped using the pandas module as the data is represented as a dataframe in the program. Dataframe is a standard object that pandas works with for data manipulation.

Sales:

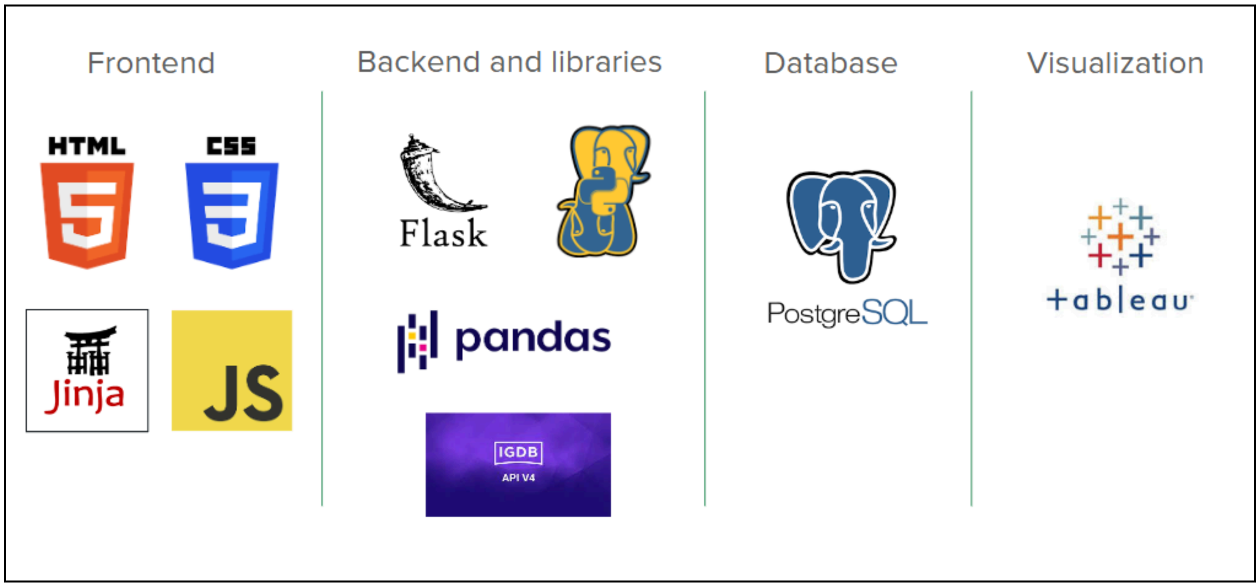
The duplication that is created in the Sales dataset is because of the platform that the game is being played on e.g. PlayStation, PC, etc. As the objective of this application is to explore the insights of the game, retaining the information about the platform is not relevant for our use case. Hence, to capture all the sales figures for each platform, the data is not dropped. The data is aggregated by summing all the sales columns: 'NA_Sales','EU_Sales','JP_Sales','Other_Sales','Global_Sales'.

The resultant instance created is a single record of the game with the sales figures summed up for all the platforms in a single row under the respective column names. For performing the aggregation, we use the pandas' builtin agg function that sums up the data in the most optimized way possible for the dataframe.

Final Application Architecture



Technology Stack



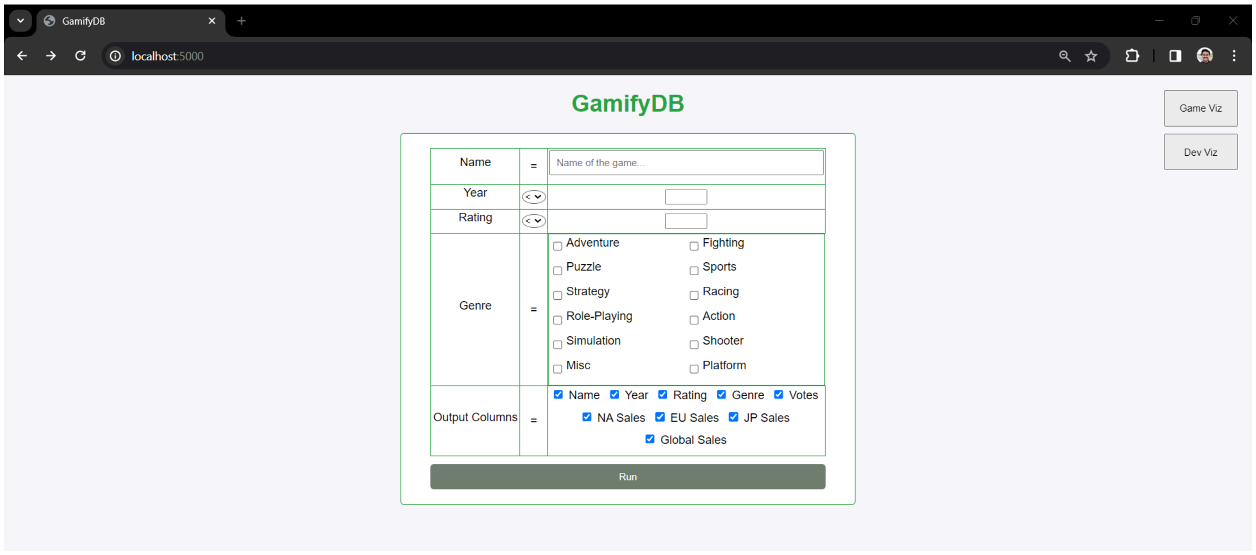
Detailed Working - API

The API has been developed using Python's microframework for web development called "Flask". Flask was chosen as, given the scope of the application, Django would have been an overkill as Django is better for a full fledged application with user authentication, payments and other facilities. But as our application performs multiple simple functions, Flask serves as the perfect choice to help ease backend development.

There are a total of 3 API endpoints that we use to serve the data:

Home GET

The GET request for the Home endpoint presents the home page of the application. This page allows the user to enter inputs by choosing from multiple options. An example is as shown below:



Name	=	Name of the game...
Year	<	
Rating	<	
Genre	=	☐ Adventure ☐ Fighting ☐ Puzzle ☐ Sports ☐ Strategy ☐ Racing ☐ Role-Playing ☐ Action ☐ Simulation ☐ Shooter ☐ Misc ☐ Platform
Output Columns	=	☑ Name ☑ Year ☑ Rating ☑ Genre ☑ Votes ☑ NA Sales ☑ EU Sales ☑ JP Sales ☑ Global Sales

*Requirement: It is necessary that at least one field is entered so that we do not fetch all the data from the database. This has been handled in the HTML page itself, where the button to execute the query is grayed out till at least one of the available inputs are chosen/entered information into. A second check is provided in the backend to ensure that a select * query is not executed.

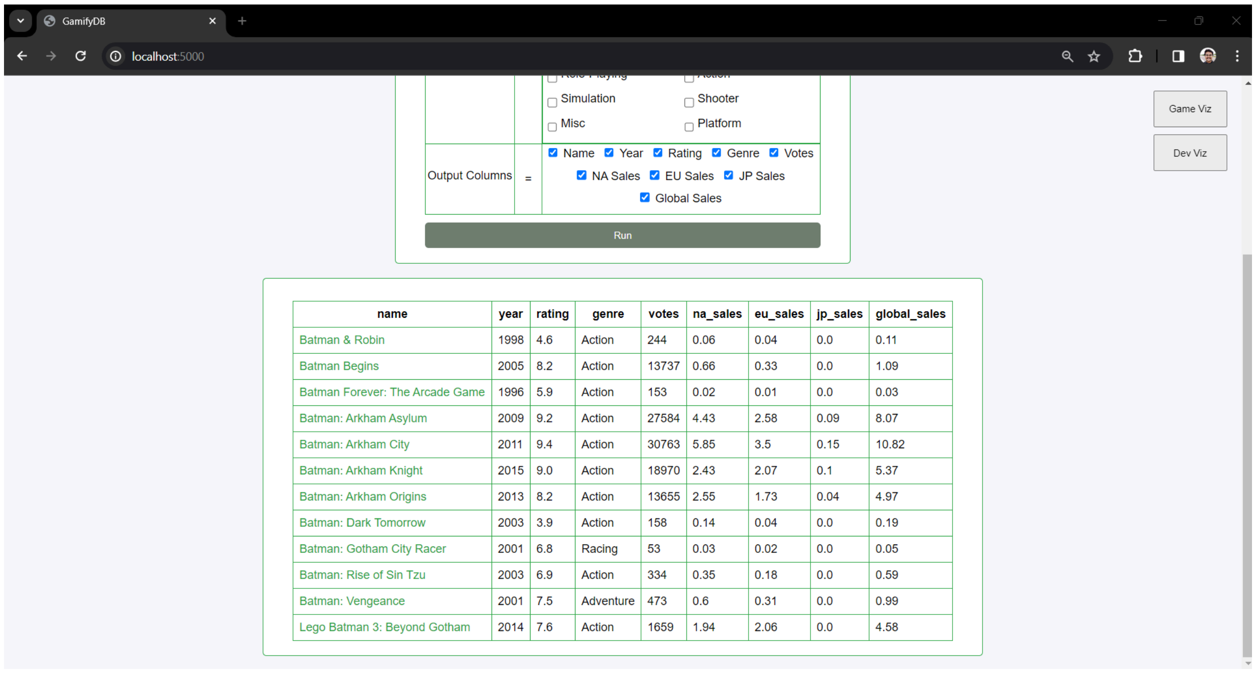
There is no compulsion to choose all the options. It depends on the person querying the information as to what it is that he wants to filter the data based on. So it is completely possible to only enter the name of the game and nothing else as well as give inputs in all the fields.

Home POST

These inputs are then submitted as form data and then handled by the same endpoint to formulate a query. This query is then executed against the Postgres database to fetch the data that matches the input given.

Once this output of the query is received in the backend, it is formatted as a class of objects, made of a custom Python class that stores all the data that the database returns. All of this data is then packaged into a JSON object and passed to the frontend that renders the data in a tabular format. The HTML output utilizes the Jinja template engine that is built into the Flask framework which allows “Python-like” code to be integrated into an HTML page.

The output that you see when the request is completed and rendered on the screen looks something like this:



name	year	rating	genre	votes	na_sales	eu_sales	jp_sales	global_sales
Batman & Robin	1998	4.6	Action	244	0.06	0.04	0.0	0.11
Batman Begins	2005	8.2	Action	13737	0.66	0.33	0.0	1.09
Batman Forever: The Arcade Game	1996	5.9	Action	153	0.02	0.01	0.0	0.03
Batman: Arkham Asylum	2009	9.2	Action	27584	4.43	2.58	0.09	8.07
Batman: Arkham City	2011	9.4	Action	30763	5.85	3.5	0.15	10.82
Batman: Arkham Knight	2015	9.0	Action	18970	2.43	2.07	0.1	5.37
Batman: Arkham Origins	2013	8.2	Action	13655	2.55	1.73	0.04	4.97
Batman: Dark Tomorrow	2003	3.9	Action	158	0.14	0.04	0.0	0.19
Batman: Gotham City Racer	2001	6.8	Racing	53	0.03	0.02	0.0	0.05
Batman: Rise of Sin Tzu	2003	6.9	Action	334	0.35	0.18	0.0	0.59
Batman: Vengeance	2001	7.5	Adventure	473	0.6	0.31	0.0	0.99
Lego Batman 3: Beyond Gotham	2014	7.6	Action	1659	1.94	2.06	0.0	4.58

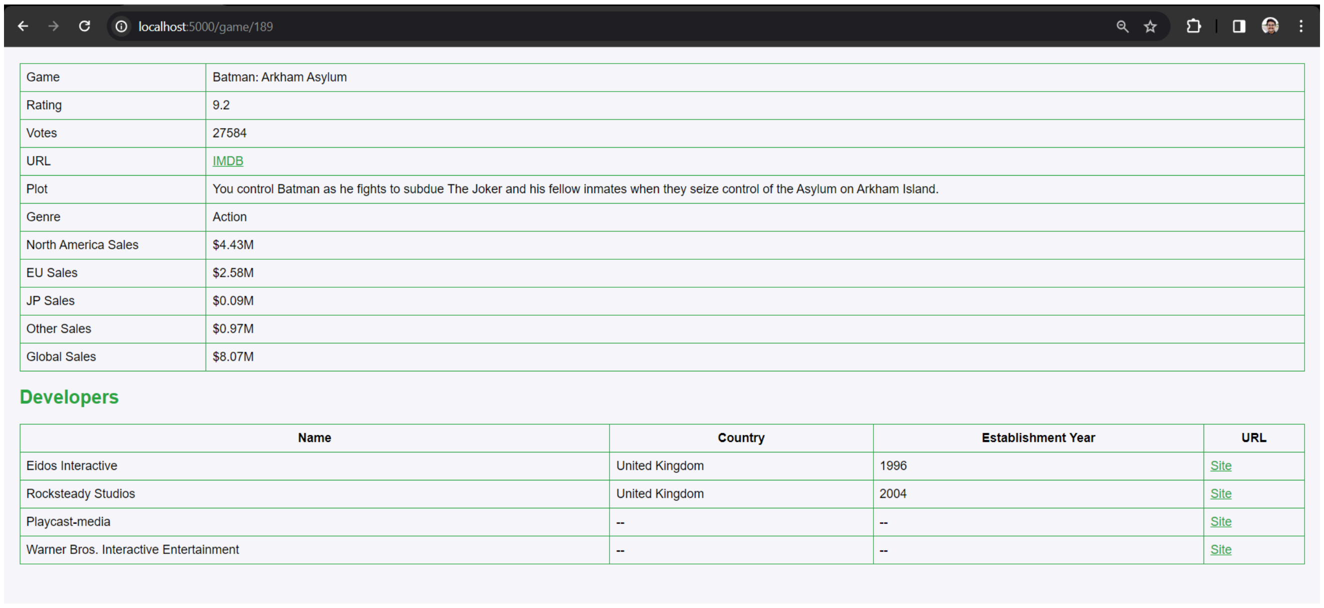
<u>Note</u>: This is a sample output visible when only Batman is entered in the name field and the query is run.

Game GET

The tabular output has the name of the game being displayed always. This name has a link embedded into it which stores the id of the game associated with that link. Once the link is clicked, the id of the game is passed to the Game endpoint that fetches all the data associated with the game from the database. A join is created over the game, game_developers and company table to ensure that all the information is picked up.

This information is then packaged into a JSON object and a new HTML page dedicated to that game is loaded. This data is then rendered as two tables: game information and company information. In case the company information is not available, the Company table is not rendered.

A sample output looks like the one below:



localhost:5000/game/189

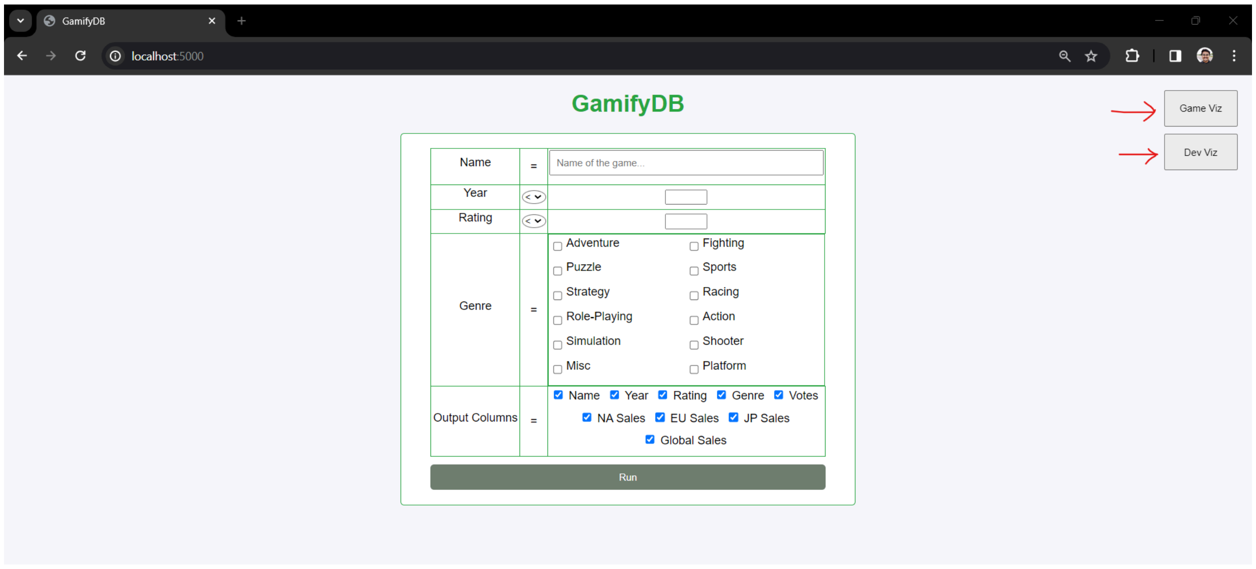
Game	Batman: Arkham Asylum
Rating	9.2
Votes	27584
URL	IMDB
Plot	You control Batman as he fights to subdue The Joker and his fellow inmates when they seize control of the Asylum on Arkham Island.
Genre	Action
North America Sales	$4.43M
EU Sales	$2.58M
JP Sales	$0.09M
Other Sales	$0.97M
Global Sales	$8.07M

Developers

Name	Country	Establishment Year	URL
Eidos Interactive	United Kingdom	1996	Site
Rocksteady Studios	United Kingdom	2004	Site
Playcast-media	--	--	Site
Warner Bros. Interactive Entertainment	--	--	Site

Visualization GET

To get an insight into the major attributes and trends in the data, we created visualizations using Tableau. These visualizations are hosted and available on Tableau public (and can be accessed by the links in the next section). We have separately written HTML code and embedded the visualizations in the page to be visible to the user. These are interactive visualizations and hence can be played with on-the-go.



Name	=	Name of the game...	
Year	<		
Rating	<		
Genre	=	☐ Adventure ☐ Puzzle ☐ Strategy ☐ Role-Playing ☐ Simulation ☐ Misc	☐ Fighting ☐ Sports ☐ Racing ☐ Action ☐ Shooter ☐ Platform
Output Columns	=	☑ Name ☑ Year ☑ Rating ☑ Genre ☑ Votes ☑ NA Sales ☑ EU Sales ☑ JP Sales ☑ Global Sales	

Two buttons in the top right corner of the Home page (Red arrows in the above screenshot) generate a GET request to the visualizations which is then rendered on a new HTML page with the embedded Tableau generated dashboards respectively. One button "Game Viz" shows visualizations about the games for which data about developers is available. The other button "Dev Viz" shows visualizations about the developers behind the games. Details about the visualizations and snapshots can be found in the next section.

Business queries and Visualization

Queries

1. Which company is behind the creation of the highly voted game (popularity)?
    - The Companies behind the highly popular game : “The Last of Us” are Naughty Dog and Sony Computer Entertainment.

2. Which game developers have developed the highest number of games in a specific genre?
    - Action: Capcom
    - Racing: Nintendo
    - Shooter: Electronic Arts

3. Which top 3 genres of games are popular and generate the highest sales? This could further help to understand the customer preference and develop similar games.
    - Action, Shooter, Platform

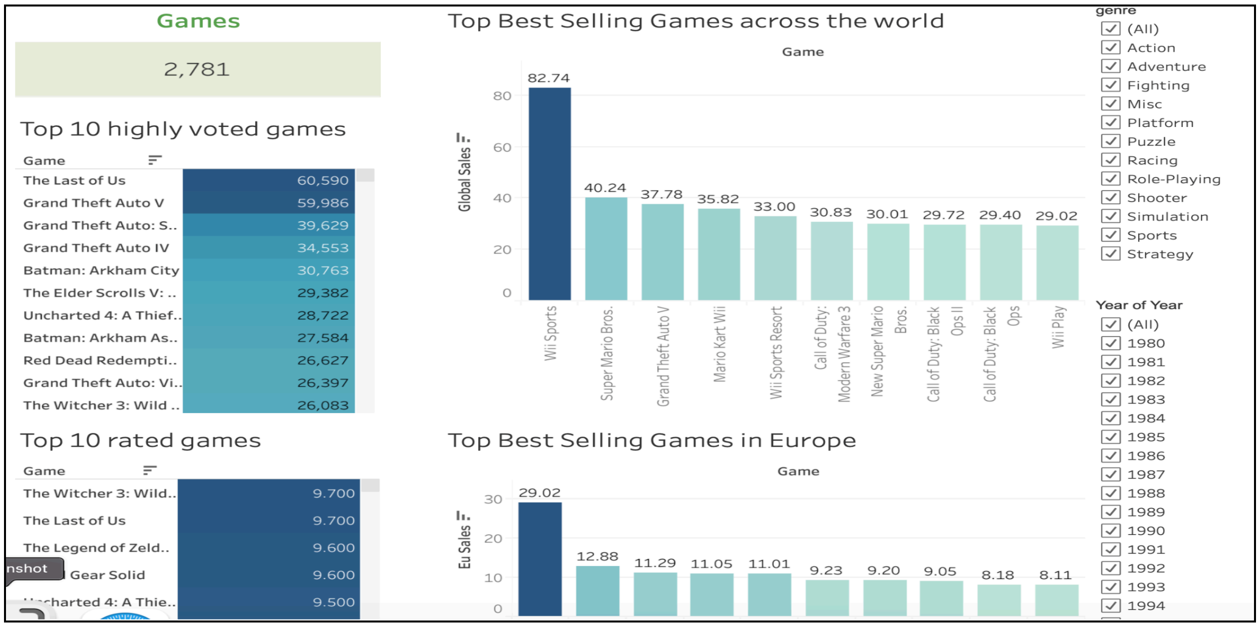
Each query mentioned above cannot be answered by a single source independently.
The integrated application will unify data from multiple heterogeneous sources and provide results for the above mentioned queries.

We designed two dashboards on Tableau to answer the above queries and provided a holistic view for the stakeholders to understand the gaming market through total sales, popularity and ratings, etc. These visualizations will help to make data informed decisions.

Dashboard link:
https://public.tableau.com/views/GamifyDBvisuals/Dashboard12?:language=en-US&:display_count=n&:origin=viz_share_link

https://public.tableau.com/views/GamifyVisuals3/Dashboard1?:language=en-US&:display_count=n&:origin=viz_share_link

Screenshots of Dashboards



Games

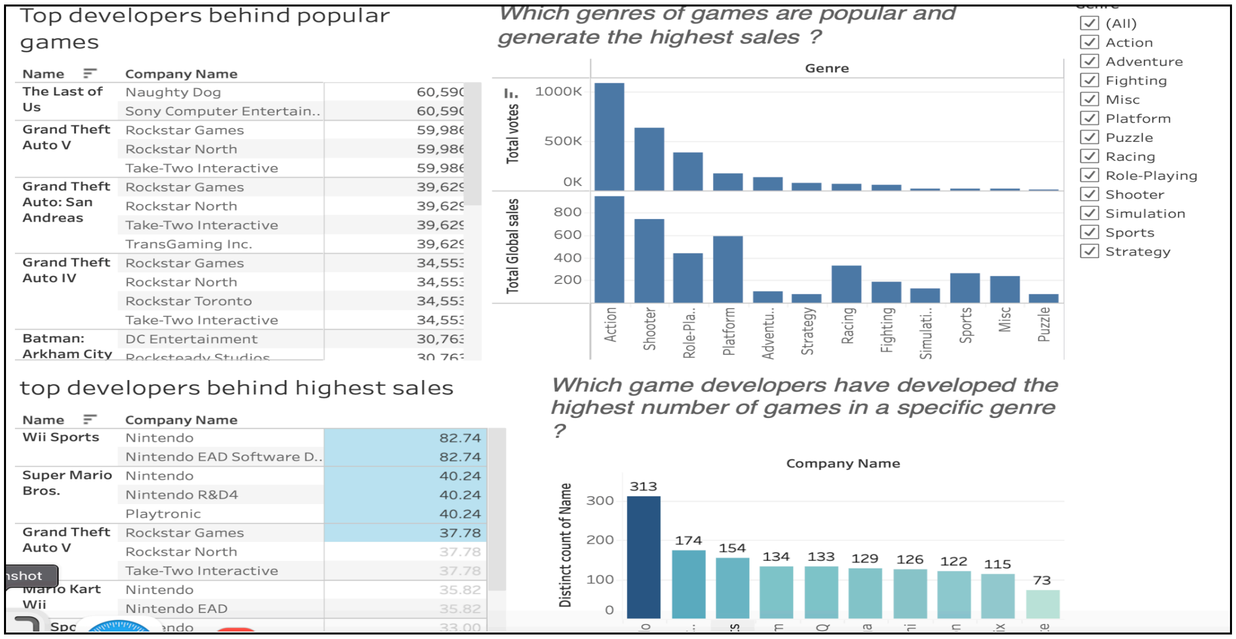
2,781

Top 10 highly voted games

Game	
The Last of Us	60,590
Grand Theft Auto V	59,986
Grand Theft Auto: S..	39,629
Grand Theft Auto IV	34,553
Batman: Arkham City	30,763
The Elder Scrolls V: ..	29,382
Uncharted 4: A Thief..	28,722
Batman: Arkham As..	27,584
Red Dead Redempti..	26,627
Grand Theft Auto: Vi..	26,397
The Witcher 3: Wild ..	26,083

Top 10 rated games

Game	
The Witcher 3: Wild..	9.700
The Last of Us	9.700
The Legend of Zeld..	9.600
Gear Solid	9.600
..charted 4: A Thie..	9.500



Top developers behind popular games

Name	Company Name	
The Last of Us	Naughty Dog	60,590
	Sony Computer Entertain..	60,590
Grand Theft Auto V	Rockstar Games	59,986
	Rockstar North	59,986
	Take-Two Interactive	59,986
Grand Theft Auto: San Andreas	Rockstar Games	39,629
	Rockstar North	39,629
	Take-Two Interactive	39,629
	TransGaming Inc.	39,629
Grand Theft Auto IV	Rockstar Games	34,553
	Rockstar North	34,553
	Rockstar Toronto	34,553
	Take-Two Interactive	34,553
Batman: Arkham City	DC Entertainment	30,763
	Rocksteady Studios	30,763

top developers behind highest sales

Name	Company Name	
Wii Sports	Nintendo	82.74
	Nintendo EAD Software D..	82.74
Super Mario Bros.	Nintendo	40.24
	Nintendo R&D4	40.24
	Playtronic	40.24
Grand Theft Auto V	Rockstar Games	37.78
	Rockstar North	37.78
	Take-Two Interactive	37.78
Mario Kart Wii	Nintendo	35.82
	Nintendo EAD	35.82
Spo..	..endo	33.00

Links of online resources used for project work

- IGDB: https://www.igdb.com/api
- Kaggle: https://www.kaggle.com/
- IMDB: https://www.imdb.com/
- Datacamp: https://www.datacamp.com/
- Tableau: https://www.tableau.com/
- Flask tutorial: https://www.youtube.com/watch?v=Z1RJmh_OqeA

Closing remarks

We would like to extend our gratitude to Prof. Dr. Melanie Herschel and Mr. Nico Lässig and the entire IPVS team for the conduction of the project.

This project was built during the Winter Semester 2023/24 at the University of Stuttgart under the Information Integration course by Vedant Puranik, Jan Bothmann and Mugdha Asgekar.