Steam Game Data Analytics System: A SQL-Driven Interactive Visualization Platform

University of Western Ontario ECE 9014 Project report

Guoyu Zhao [gzhao56@uwo.ca](mailto:Zhao,gzhao56@uwo.ca) Yi Zhao zyi44uwo.ca

*Abstract*—*This project presents the design and implementation of an interactive data analytics system for analyzing player trends of Steam games. By integrating a structured MySQL database with Streamlit, the system enables real-time visualization, SQL-driven querying, and data exploration across various dimensions including player counts, forecasted trends, seasonal behaviors, and clustering insights. Key components of the system include dynamic charting based on SQL queries, ARIMA-based time series forecasting, and k-means clustering for game segmentation. Users can interact with the dashboard to explore game performance and run custom SQL queries without coding expertise. The project emphasizes data integrity, modular backend design, and visual storytelling to support data-driven insights. This system demonstrates the practical use of relational databases and modern web frameworks for building scalable and insightful analytics platforms.*

# Introduction

With the rapid growth of the video game industry, Steam has become the world’s largest digital game distribution platform, hosting thousands of games with millions of active players. Understanding player trends over time is crucial for game developers, marketers, and analysts who seek to optimize engagement and evaluate game performance. However, raw player data alone provides limited value without appropriate tools to structure, analyze, and visualize the information.

This project introduces an end-to-end data analytics platform for Steam game player statistics. The system extracts, stores, and analyzes historical player data using a MySQL relational database, enabling structured querying and efficient aggregation. A Streamlit-based web interface is developed to allow users to interact with the data through charts, SQL queries, and dashboard-style visualizations. The platform focuses on identifying top-performing games, forecasting future activity, understanding seasonal patterns, and segmenting games based on player metrics.

The software system developed in this project serves as both a visual analytics tool and a demonstration of full-stack data integration. It empowers users to gain insights through both pre-generated visualizations and custom exploratory queries, with minimal technical background required. The project emphasizes modularity, data normalization, and real-time query responsiveness.

This report details the design decisions, methodologies, and technical implementation of the system, and evaluates its performance in terms of usability, scalability, and analytical depth. It also highlights how this solution compares with existing platforms, and how it can be extended for broader use cases, such as game recommendation or market forecasting.

# Related work

Several platforms and services currently provide game analytics, such as [SteamCharts](https://steamcharts.com/" \t "_new) and [SteamDB](https://steamdb.info/" \t "_new), which offer real-time and historical player statistics. These tools typically display charts and tables derived from scraped or API-based data. While visually informative, these platforms offer limited interactivity, lack custom query capabilities, and do not support user-defined forecasting or clustering analysis. Furthermore, they often rely on precomputed visualizations without exposing the underlying data model for exploration.

Academic works on video game analytics have primarily focused on player behavior modeling, churn prediction, and recommendation systems using machine learning techniques. However, many of these studies use proprietary datasets and are not publicly replicable or usable by non-technical audiences. In contrast, our system uses a publicly accessible dataset and focuses on delivering exploratory and visual tools for analysts and casual users alike.

Compared to existing systems, our solution is self-contained, open, and highly customizable. It integrates a normalized SQL database backend with a web-based visualization interface, allowing real-time querying and analytics. Unlike static dashboards, our system enables users to input raw SQL, generate dynamic visual outputs, and interact with preset exploratory modules such as ARIMA forecasting, seasonal decomposition, and game clustering.

The system is designed to be modular and extensible. It can be expanded to support other datasets, user accounts, or even recommendation engines in future iterations. This flexibility and transparency set it apart from traditional Steam data viewers and third-party analytics services.

# Methodology

## System Overview

The Steam Game Analytics System is an end-to-end pipeline that combines data collection, database design, machine learning, and interactive visualizations. The application consists of two main components: a set of data analytics scripts and a Streamlit-based web dashboard. Data is initially processed using Python from Excel-based CSV files, and then migrated to a MySQL relational database for advanced SQL-driven analytics.

As is shown in the fig.1,the final web system supports both static visualization and live database querying, giving users flexible access to trends, forecasts, seasonal insights, and data clustering modules.

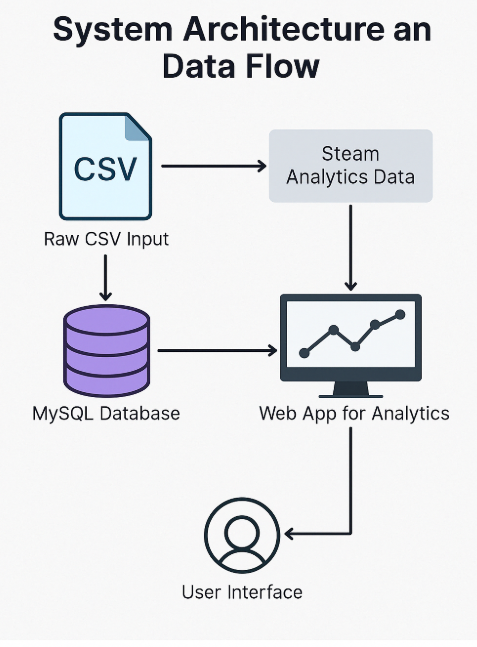


Fig.1

## Functional Description

The system supports a range of core analytics features, including top charts to display historical player trends for popular games, ARIMA-based forecasting for predicting future engagement, event annotation for highlighting significant updates, clustering for behavior-based grouping using KMeans, and seasonal decomposition to analyze recurring patterns in gameplay data. An interactive SQL query interface is also integrated, allowing users to write custom queries and visualize results in real time using charts. These modules operate on either preprocessed CSV inputs or SQL-driven backends, depending on the level of interactivity required.

The dataset was sourced from SteamCharts, covering monthly average and peak player counts for over 100 Steam games from 2012 to 2022. Initial cleaning and preprocessing were performed using Excel and Python. To enable more dynamic querying and improve scalability, the data was subsequently migrated into a MySQL relational database. This dual-pipeline design enabled side-by-side validation of CSV-based static analysis and SQL-based dynamic visualizations.

The database schema consists of two core entities: GAMES, which stores game metadata, and MONTHLY\_STATS, which stores monthly average and peak player metrics. A one-to-many relationship is established via game\_id, allowing each game to be associated with multiple statistical records. This structure supports robust aggregation, filtering, and time-series analysis across the system.

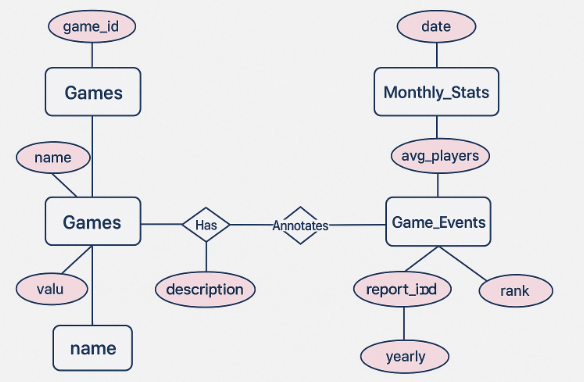


Fig.2

## Relational Schema

The entity-relationship (ER) model described above was transformed into a simplified and normalized relational schema using MySQL. The database consists of two core tables: GAMES(game\_id, name) and MONTHLY\_STATS(id, game\_id, date, avg\_players, peak\_players). Here, game\_id serves as the primary key in the GAMES table and is also a foreign key in the MONTHLY\_STATS table, establishing a one-to-many relationship between games and their corresponding monthly player statistics.

This schema is fully normalized to Third Normal Form (3NF), ensuring the elimination of partial and transitive dependencies. All non-key attributes are fully functionally dependent on the primary key, and there is no redundancy in data storage. The streamlined design supports efficient querying, particularly for aggregation, time-series filtering, and joining operations that are central to the application’s analytics dashboard and SQL query interface.

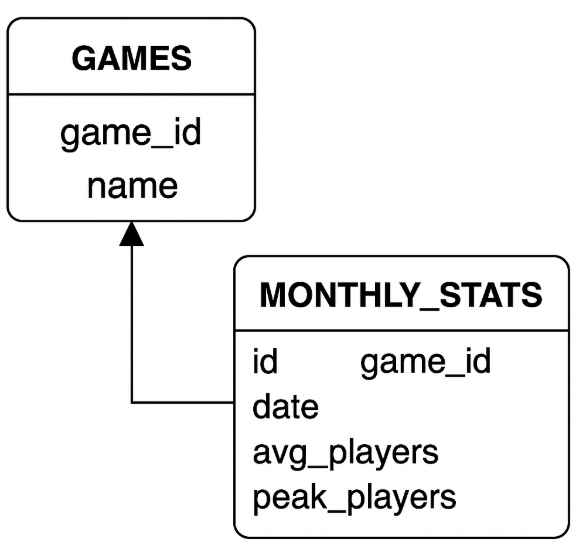


Fig.3

## Data Base Query Design Patterns

To support dynamic querying capabilities within the SQL interface, a series of well-structured query patterns were implemented. These include aggregated metrics for player trends, temporal filters for selecting date ranges, and top-N logic to rank games by popularity. All queries are written to leverage MySQL’s JOIN operations between games and monthly\_stats, enabling real-time access to both metadata and longitudinal statistics. To further enhance responsiveness, indexes were applied to the game\_id and date columns—optimizing performance for the most frequent lookup paths. These query patterns are at the heart of several visual modules in the dashboard and empower users to gain timely insights through SQL-driven interaction.

## Dual-Source Data Ingestion Strategy

The system was built with a dual-pipeline architecture that initially leveraged static CSV-based analysis before transitioning to a dynamic SQL-driven backend. In the early development phase, raw Steam player statistics were collected from SteamCharts and stored in structured CSV files (Fig. 4). Python scripts were used to clean, interpolate, and validate the data, as well as to produce exploratory visualizations.

After verification, the dataset was imported into a MySQL relational database and normalized into two main tables—games and monthly\_stats. This migration enabled the use of SQL queries to support real-time interaction, chart generation, and modular analytics within the web application. The dual-mode ingestion strategy allowed for parallel validation between CSV-based charts and their SQL-driven counterparts, offering confidence in data integrity and enabling flexible scalability for future feature expansion.

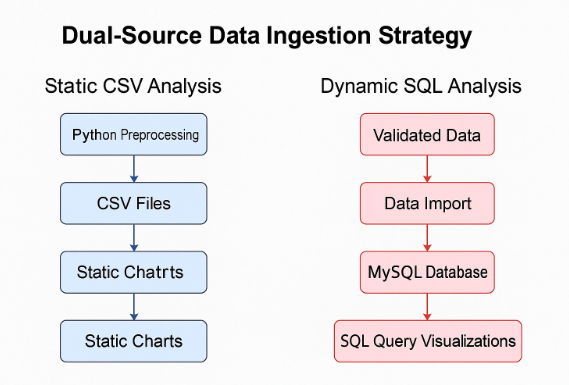


Fig.4

# IMPLEMENTATION

The implementation phase of this project focused on translating the conceptual design into a functional system that integrates a structured SQL database, Python-based analytics, and an interactive web interface. The core of the system revolves around a normalized relational schema, built on MySQL, and designed to support both real-time query execution and data-driven visualizations.

The database was carefully normalized to Third Normal Form (3NF) to eliminate redundancy and ensure integrity. The schema consists of two main tables: GAMES(game\_id, name) and MONTHLY\_STATS(id, game\_id, date, avg\_players, peak\_players). The game\_id attribute serves as the primary key in GAMES and as a foreign key in MONTHLY\_STATS, establishing a one-to-many relationship. Every attribute in each table is fully functionally dependent on its primary key, and no transitive dependencies exist. As a result, all relations satisfy the conditions of 3NF, and no data duplication or update anomalies are present.

Further analysis was conducted to assess compliance with Boyce-Codd Normal Form (BCNF). The current schema exhibits no significant BCNF violations. Although a composite key (game\_id, date) could theoretically replace the surrogate key id in MONTHLY\_STATS, the design decision to retain a standalone id improves indexing and supports extensibility—especially in visualization modules and dynamic query interfaces. Since all functional dependencies in the schema are governed by candidate keys, no decomposition was required to achieve BCNF.

A key implementation goal was to enable fast and expressive SQL querying. To support this, several indexes were added to frequently queried columns, such as game\_id and date. These indexes significantly improve performance in aggregation and JOIN-heavy queries, particularly for dashboards that involve real-time data lookups. In designing the SQL schema, considerations were made to accommodate different query types, including player count trend retrieval, top-ranked games, event-based filtering, and machine learning model input preparation.

The full-stack implementation was built in Python using the Pandas library for data manipulation, mysql.connector for database interaction, and Matplotlib for static visualizations. The dashboard was created using Streamlit, providing a clean interface with tab-based navigation. The system supports both static chart rendering (for seasonal decomposition and clustering) and SQL-powered chart generation, allowing users to interact with the database directly through editable queries.

To accommodate both early-stage analysis and final dashboard deployment, a dual-source data ingestion strategy was adopted. In the initial phase, data scraped from SteamCharts was stored in CSV format and used for static processing, including early testing of forecasting models and visualizations. In the production stage, all records were imported into MySQL and normalized according to the relational schema. This dual-track architecture allowed for flexibility in experimentation while maintaining a consistent backend structure in the deployed system.

The final system reflects an integration of data science, web development, and relational database theory. Its modular structure enables reproducible data analytics and supports expansion to new games or statistical models. A system architecture diagram and ER model have been included in this report, along with a screenshot of the implemented Streamlit application and database schema to support technical traceability, shown in fig 5.

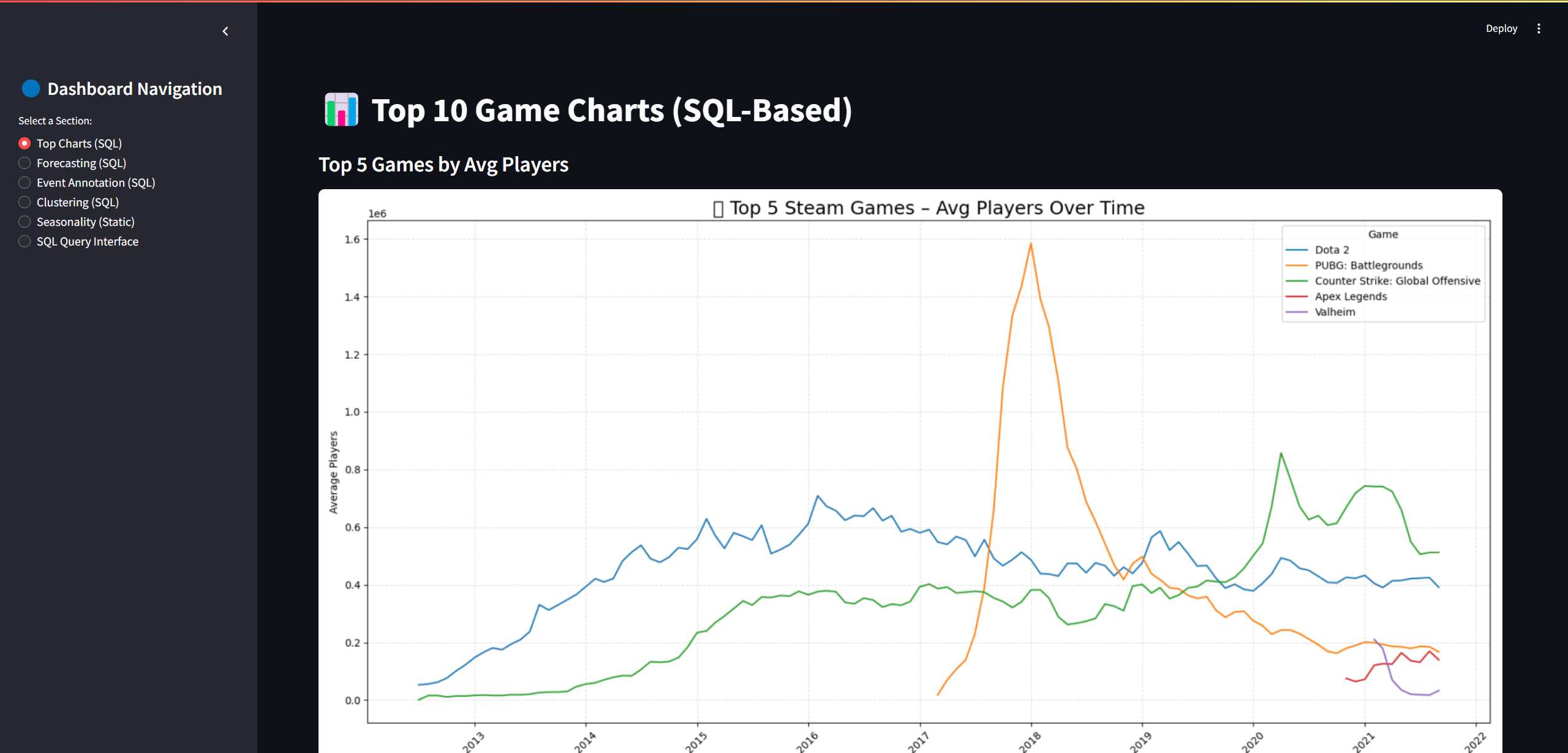


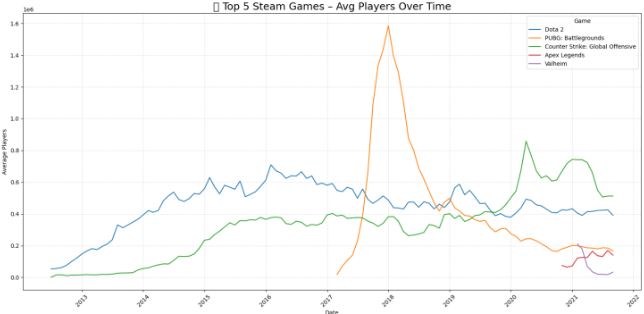
Fig.5

# RESULTS

The Steam Game Analytics system produced a range of analytical results through SQL-driven queries and dynamic visualizations. These results offer insights into player trends, game popularity, forecasting outcomes, seasonal behavior, and inter-game relationships. All data was processed from a MySQL database backend and visualized through a Python-powered Streamlit web interface.

## Top Player Trend (SQL driven Charts)

To begin, the system identified the top 10 games based on their historical average monthly player counts. Figure 6 displays the trend of the top five games—Dota 2, PUBG: Battlegrounds, Counter-Strike: Global Offensive, Apex Legends, and Valheim—showcasing their popularity over time. Notably, PUBG experienced a sharp spike during 2017–2018, while CS:GO demonstrated more stable engagement. These trends were extracted using SQL aggregation grouped by game\_id and visualized using Matplotlib.

Fig.6

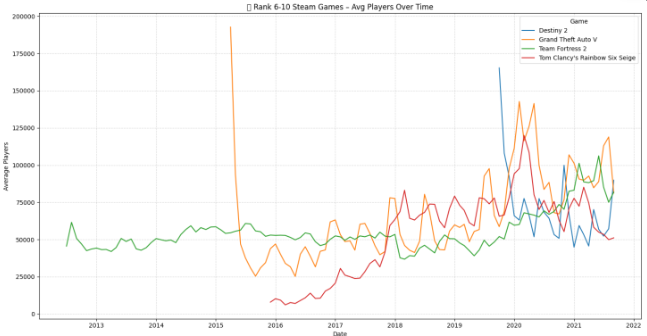
To complement this, a separate analysis was conducted for the next five highest-ranked games (Figure 7), including Grand Theft Auto V, Destiny 2, and Team Fortress 2. These games maintained consistent user bases, although they exhibited greater seasonal variance and occasional volatility. This analysis helped validate that the long tail of player engagement remains significant and warranted further study beyond the top five.

Fig.7

## Forecasting Module (ARIMA MODELS)

To anticipate future player trends on the Steam platform, the system integrated an AutoRegressive Integrated Moving Average (ARIMA) forecasting pipeline. The ARIMA model is a well-established time series analysis technique capable of capturing trends, seasonality, and residual noise within univariate temporal datasets. In this project, ARIMA was employed to project average monthly player counts based on historical data extracted via SQL from the monthly\_stats table.

The forecasting module was implemented in Python using the statsmodels library. Historical data for two sets of games were used: the top 5 most played games and ranked 6–10 titles, as identified by overall average player counts. For each game, the model was trained using a time series indexed by month, starting from the earliest available observation (often around 2013) up to 2021. Forecasts were then generated for the first half of 2022. Figure 8a and 8b visualize these predictions.

Before modeling, data was preprocessed through interpolation and smoothing, ensuring there were no missing monthly values. Since the data exhibited non-stationarity in both mean and variance—evident from upward trends and shifting amplitudes—each series was differenced accordingly to meet stationarity conditions required by ARIMA.

A grid search method was applied to identify optimal ARIMA hyperparameters (p, d, q) for each game. While automatic selection (e.g., using AIC minimization) worked for some titles, others required manual tuning due to seasonal interference or inconsistent variance. The models also computed 95% confidence intervals, providing upper and lower bounds for player count projections. These bands are shaded in light gray in both plots, helping identify the level of uncertainty in the near-term forecast horizon.

The results in Figure 8a illustrate predictions for the top 5 games, including Dota 2, CS:GO, PUBG, Apex Legends, and Valheim. The ARIMA models captured each game's unique growth dynamics—such as the sharp rise and plateau of Valheim after its explosive debut, or the relatively stable but slightly declining trend of Dota 2. In contrast, Figure 8b forecasts the rank 6–10 games, such as Rust, Destiny 2, GTA V, and Rainbow Six Siege. These games showed more volatile patterns, with some forecasts suggesting a potential decline, while others—like Rust—indicated possible continued growth based on past surges.

The forecasting module proved especially useful in enabling proactive insight for developers and analysts. By identifying whether a title is likely to gain or lose engagement in upcoming months, publishers can more strategically time game updates, marketing campaigns, or server resource allocation. Furthermore, this module highlights the system’s ability to integrate machine learning models on top of SQL-driven data pipelines, demonstrating the potential for advanced analytics at scale.

The flexibility of ARIMA also allowed the system to be easily extended in the future to support seasonal ARIMA (SARIMA) models or be replaced with more complex approaches such as Prophet or LSTM, should longer-term or nonlinear forecasting be required. Nonetheless, the existing implementation serves as a robust and interpretable solution for short-horizon Steam game trend prediction.

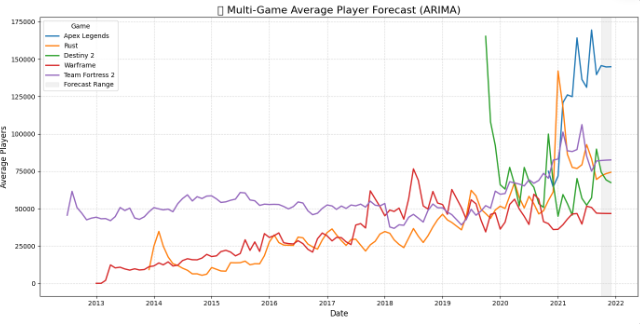


Fig.8a

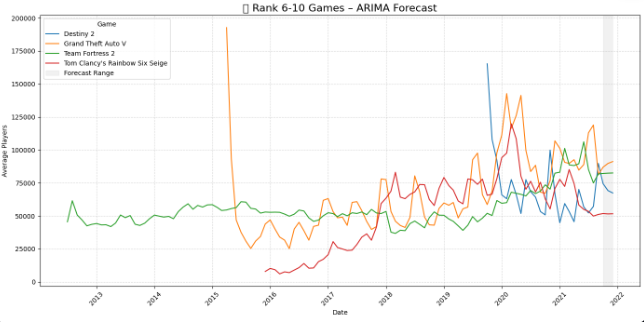


Fig.8b

## Event Impact Annotation

Event annotation added critical contextual understanding. Major updates and platform shifts (e.g., CSGO going free-to-play in Dec 2018, Rust’s Twitch campaign in mid-2020, and the Steam Deck launch in 2023) were tagged directly on line plots (Figure 9). These annotations were hardcoded based on industry news and game update logs, mapped to exact SQL timestamp values, and enhanced with vertical dashed lines for clarity.

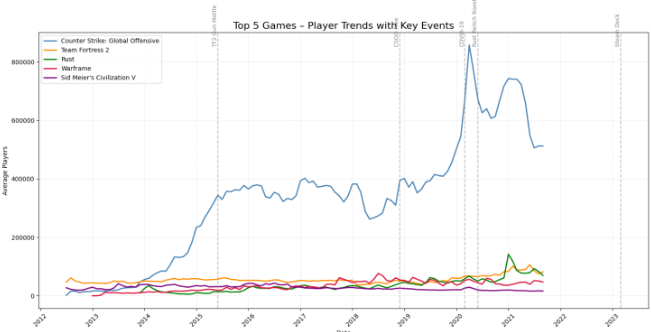


Fig.9

## Clustering Analysis

To uncover latent groupings among games based on popularity characteristics, a clustering module was integrated into the analytics system using the KMeans algorithm. This unsupervised learning method was applied to identify groups of games with similar behavioral profiles, based on their average and peak player counts aggregated across the full available dataset. These two metrics were chosen as key indicators of both long-term engagement (average) and temporary surges (peak), enabling a nuanced understanding of user dynamics.

Before clustering, all player count data was log-transformed due to the wide disparity in scale—ranging from small indie titles with a few hundred players to blockbuster hits exceeding a million. Applying a log-log scale helped compress the value range and emphasize relative differences across game tiers, while also aligning with the assumption of roughly Gaussian-distributed clusters needed for KMeans. The features were then standardized using StandardScaler from scikit-learn to ensure both axes contributed equally to the clustering outcome.

The optimal number of clusters was selected using the elbow method, where a scree plot of inertia (sum of squared distances to centroids) suggested three clusters offered a meaningful trade-off between granularity and interpretability. As shown in Figure 10, each point in the scatter plot represents a unique game, plotted by its average and peak player counts (in log scale), and color-coded by its cluster assignment.

Cluster 0 (blue) consists of a large number of games in the "long tail"—titles with relatively modest engagement and consistent player bases. These include many mid-tier or niche games, as well as older titles with legacy player communities. Cluster 1 (green) contains moderately popular games with solid, sustained engagement. These games typically enjoy consistent visibility on the Steam platform and may benefit from regular updates or seasonal player return patterns. Notably, games like Rust and Apex Legends fall into this group. Cluster 2 (orange) includes only the outliers—mega-popular titles like Dota 2 and Counter-Strike: Global Offensive, which have both extremely high peaks and high sustained averages. Their presence in a distinct cluster emphasizes their exceptional dominance in the ecosystem, and validates the model’s ability to isolate extreme cases.

The clustering process was executed in Python and stored back into the database, where each game was assigned a cluster label. These labels were then used to generate the final visualization using SQL-powered queries. Figure 10 illustrates the final clustering output, plotted using log-log axes to clearly separate the distributions. The visualization helps stakeholders and developers quickly understand how games are positioned relative to one another in terms of user dynamics, and can inform market segmentation strategies, resource prioritization, or feature targeting.

In future extensions, this clustering module can be augmented with additional features such as player retention rates, price history, genre metadata, or content update frequency. Moreover, alternative clustering algorithms like DBSCAN or Gaussian Mixture Models could be evaluated to capture more nuanced shapes or density-based groupings.

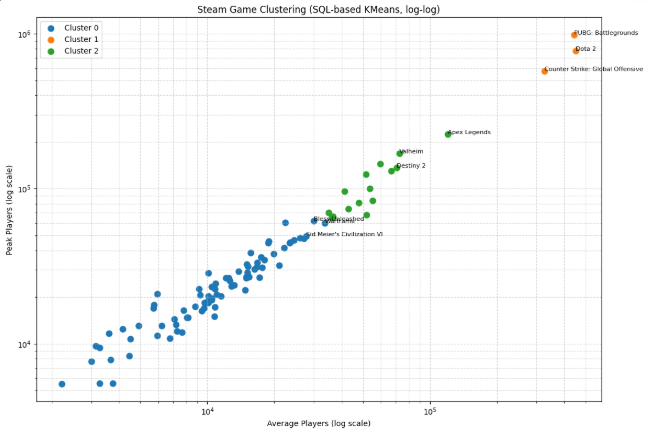


Fig.10

## Seasonality Analysis

A deeper layer of analysis was conducted by decomposing the monthly player count time series of five popular Steam games using additive seasonal decomposition. This approach, implemented via the seasonal\_decompose method from the statsmodels Python library, separates each game’s time series into trend, seasonal, and residual components. These decomposed views provide insight into not only the long-term popularity trajectory of a title, but also recurring monthly behavior and random variation. The following figures (Figure 11a–11e) present decomposition results for five selected games, with each providing unique patterns and interpretative value.

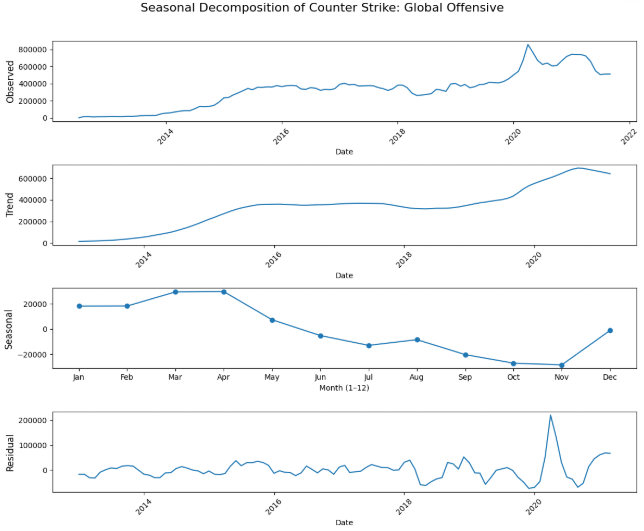
Figure 11a illustrates the decomposition of CS:GO, a game known for its enduring competitive scene. The trend component shows steady growth from 2017 to mid-2020, peaking shortly after the game transitioned to a free-to-play model in late 2018. A sharp spike during the COVID-19 pandemic reflects a surge in global gaming activity. The seasonal component reveals consistent monthly fluctuations, with noticeable increases in summer (June–August) and December, which aligns with school holidays and major events like ESL tournaments. Residual values remain low except for a few large events in early 2020, validating the seasonality signal strength.

Fig.11a

Figure 11b breaks down the seasonal patterns for Dota 2, another Valve title with strong esports ties. The trend component suggests a gradual decline from its 2016 peak, with temporary recoveries during major updates and tournaments (e.g., The International). The seasonal cycle shows a clear peak in August—corresponding exactly with The International—followed by relative lulls in September and October. This game exhibits the most defined and cyclical seasonal signal among all analyzed titles, with player counts spiking predictably around competitive seasons.

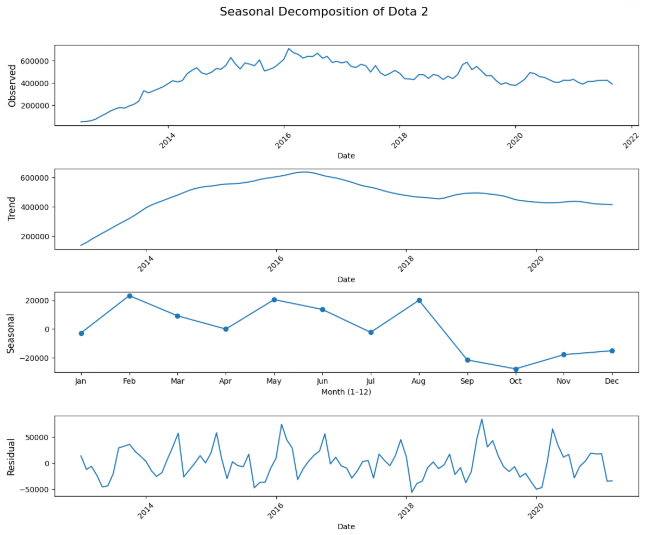


Fig.11b

Figure 11c presents the decomposition of Rust, a multiplayer survival game with unpredictable popularity swings. Its trend line shows dramatic growth during 2020–2021, coinciding with Twitch campaigns and YouTube streamer involvement. However, its seasonal component is more erratic than others. Although small monthly upticks are visible in December–January and April, the pattern is weaker and more irregular. The residuals are large, indicating that Rust’s popularity is driven more by community-driven surges and viral events rather than predictable seasonal cycles.

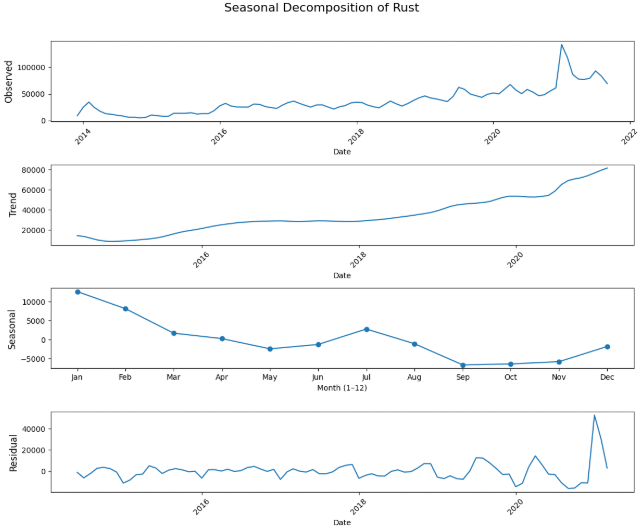


Fig.11c

Figure 11d details the seasonal decomposition of GTA V, a game with both single-player and online multiplayer modes. Its trend curve suggests long-term retention with moderate decline post-2018. The seasonal component exhibits modest surges in April and December—likely reflecting spring break and holiday sales or online events. GTA V's pattern is not as sharp as Dota 2's but shows consistent engagement cycles. Unlike Rust, the residuals are smaller, showing fewer outliers and supporting the presence of a repeatable user behavior rhythm.

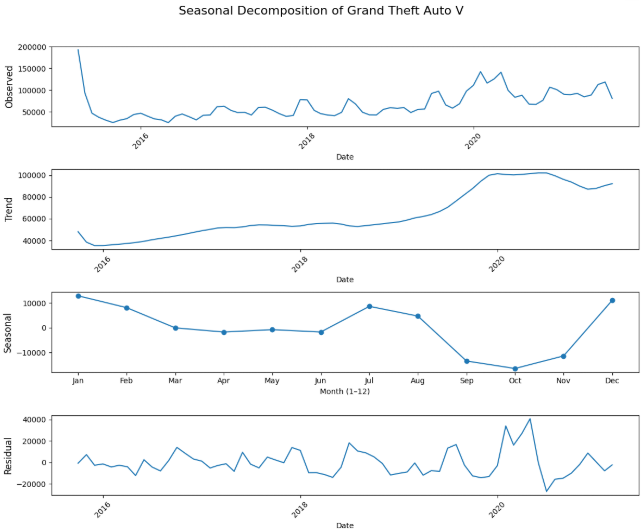


Fig.11d

The decomposition results for Warframe, shown in Figure 11e, highlight a game with strong engagement but a more stable player base. Its trend shows slow growth followed by plateauing behavior from 2018 onward. The seasonal cycle indicates modest increases in March and July, often aligning with in-game content expansions or updates. Compared to the other games, Warframe's seasonality is more muted, with low amplitude but consistent periodic behavior. The residuals remain relatively low, suggesting stable player retention outside of update-driven fluctuations.

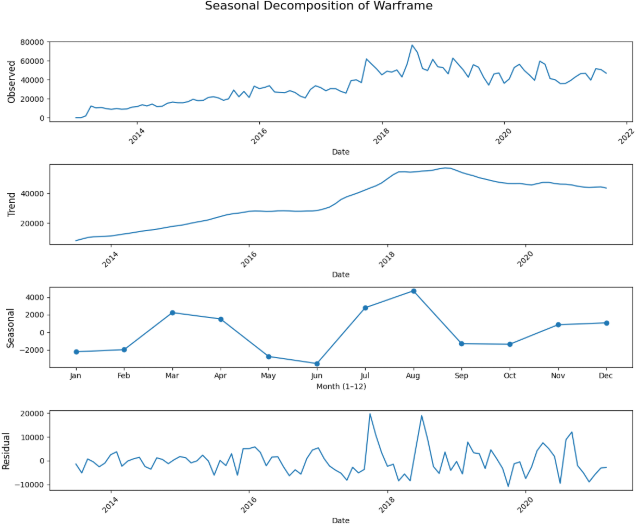


Fig.11e

Across all five games, unique seasonal behaviors were observed. Competitive titles like Dota 2 and CS:GO demonstrate the strongest and most regular seasonality, with patterns tightly linked to esports schedules. These patterns are invaluable for developers and marketers. Understanding when player activity peaks enables better scheduling of updates, promotions, or esports events. Moreover, the combination of SQL-based aggregation and Python decomposition provides a scalable and reproducible approach for longitudinal engagement analysis.

## SQL Query Interface and Dynamic Exploration

To further enhance user interactivity and empower analytical exploration, the system incorporated a live SQL Query Interface as part of the deployed web application. This interface allowed users to write and execute their own SQL statements against the backend MySQL database in real time. Queries could be written manually in the input box, enabling direct access to the database using full SQL syntax (e.g., SELECT, JOIN, GROUP BY, ORDER BY, etc.). This flexibility emulated real-world data analysis workflows, where analysts often need to perform ad-hoc exploration without relying solely on static dashboards.

To accommodate users unfamiliar with SQL or to provide guidance for common use cases, five predefined SQL templates were also included. These templates covered typical analytics tasks such as retrieving the top 10 games by average players, identifying games with over one million peak users, examining growth trends by year, and filtering records based on date or game-specific conditions. Users could select a preset, modify the query if desired, and run it with a single click to retrieve results immediately.

As shown in Figure 12a, the query interface displayed results in tabular form, allowing for easy review and export. This example shows a query that retrieves the top 10 games with the highest average monthly player counts. In addition to displaying the raw data, the system automatically generated visualizations based on the query result structure.

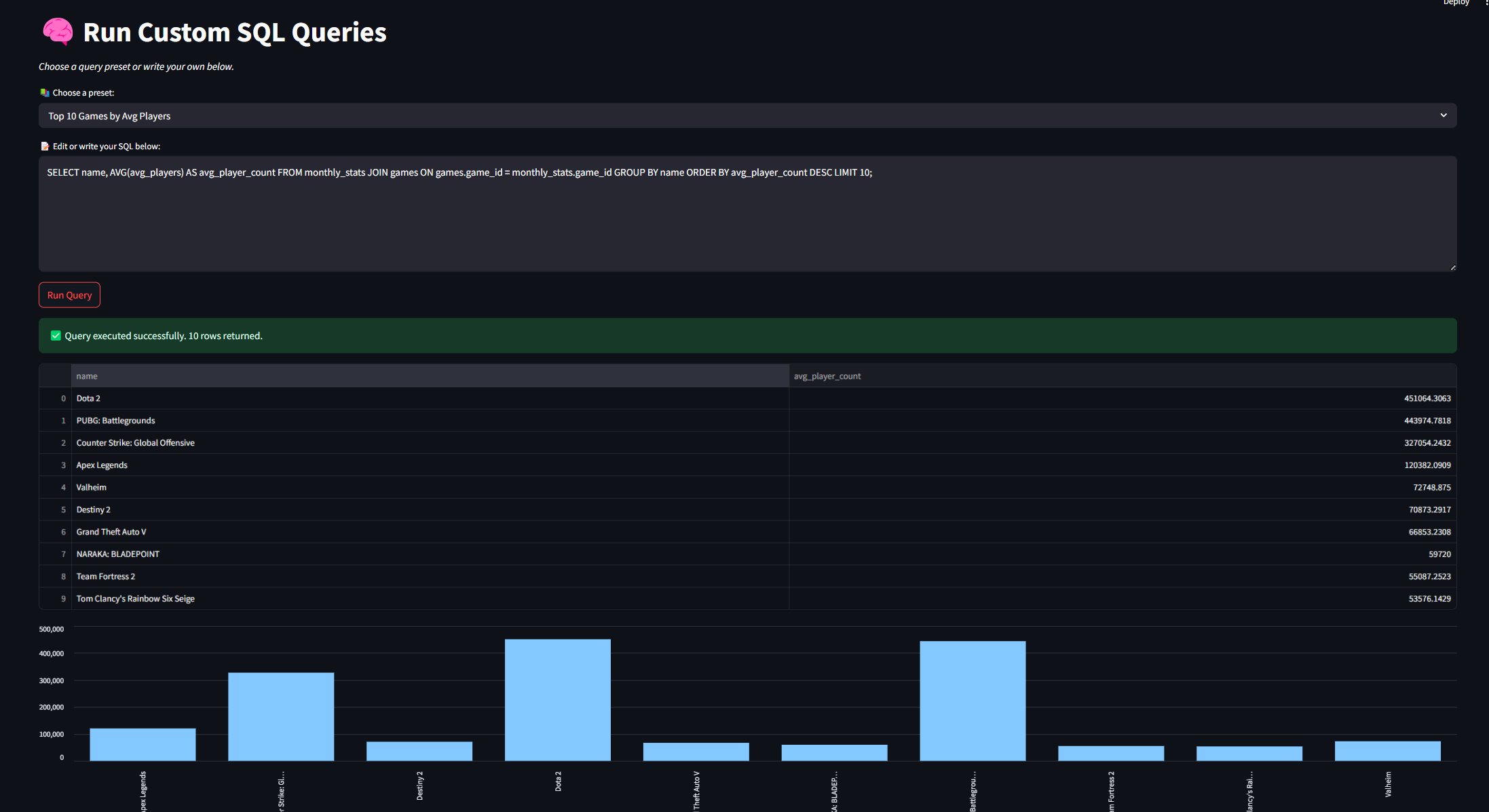


Fig.12a

When the returned table contained two columns—typically a categorical key (like game name) and a numerical metric (like average players)—the interface rendered a bar chart to complement the table, as shown in Figure 12(b). This visual output provided users with immediate insight into the distribution of the result, making it easier to identify dominant entries or outliers. Similarly, when a time series result was detected (e.g., date on the x-axis), a line chart was generated to highlight temporal patterns.

Overall, the SQL interface acted as a powerful analytical extension for the system, combining flexible query formulation, live data retrieval, and automatic visualization in a unified web-based environment. This design proved especially beneficial for more technical users or instructors seeking to simulate exploratory data analysis tasks with real game industry datasets.

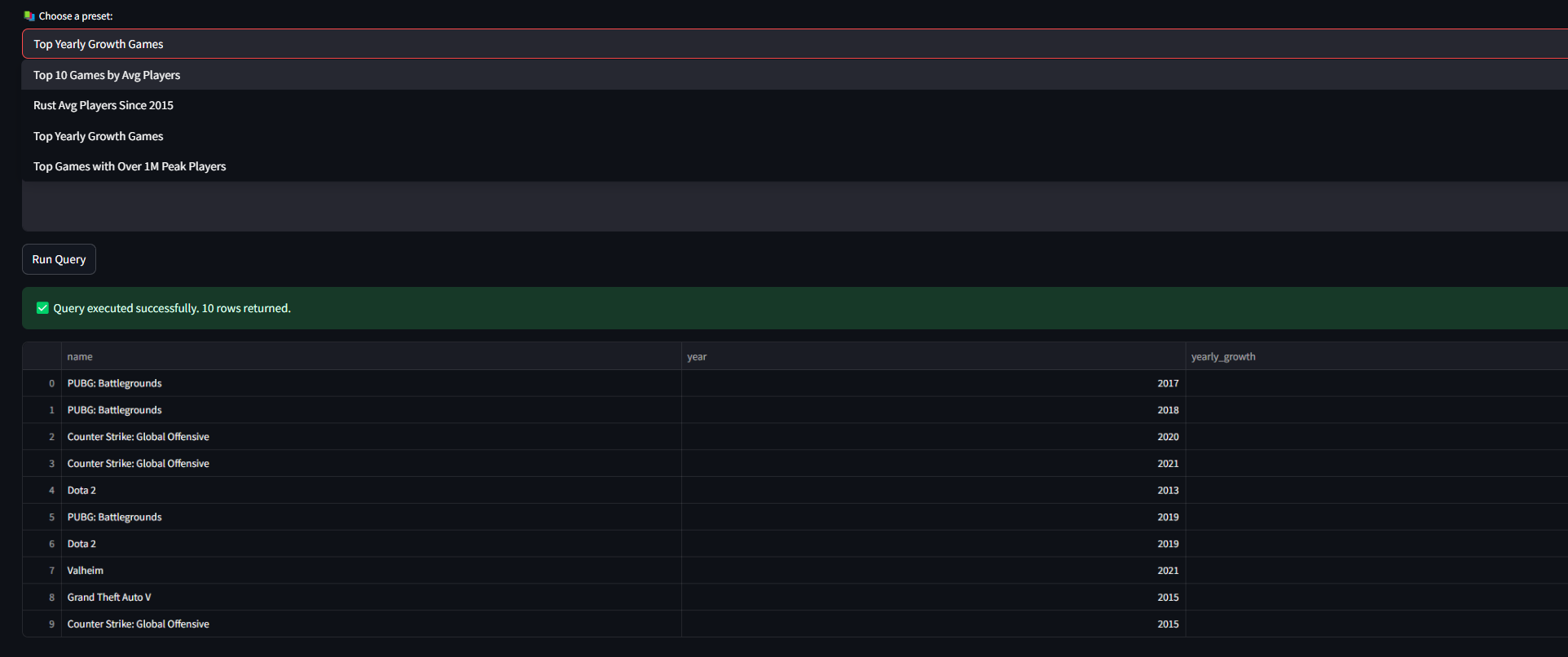


Fig.12b

# EVALUATION AND DISSCUSION

A core objective of this project was to validate the accuracy and reliability of our data analytics pipeline, particularly when transitioning from static CSV-based methods to dynamic SQL-driven querying. To this end, we implemented both modes—first by extracting, cleaning, and visualizing data from raw Excel (CSV) files using Python scripts, and then by migrating the data into a MySQL relational database and reconstructing the same visualizations via SQL queries. This dual-path approach enabled direct comparison of the output results from both sources.

As demonstrated in Figures 3 and 7 (e.g., Top 5 games by average players and event-annotated time series), the SQL-based charts were visually and numerically consistent with those generated from CSV data, confirming the correctness of the database ingestion process and the integrity of the relational schema. This validation gave us confidence in the backend migration logic, while also demonstrating the reliability of the structured database design.

Beyond validation, the SQL-driven backend facilitated significant extensibility. For instance, while the CSV analysispipeline was originally built for the top 5 games only, the SQL version seamlessly scaled to include additional analyses—such as forecasting and trend extraction for rank 6–10 games (see Figure 8). This extension required no changes to the data structure or schema, highlighting the system's flexibility for expansion and maintenance. The ability to generalize the same forecasting logic to different game groups showcases a modular and future-proof design.

Another dimension of evaluation centered on user interactivity. The implementation of a SQL Query Interface allowed users to write custom queries, experiment with filters, or explore aggregation and join operations in real time. This empowered users with different levels of technical proficiency to access either predefined templates or create entirely new insights. The automatic rendering of charts based on query results further enriched the usability and analytical depth of the platform (Figures 12a and 12b).

Through this project, we gained hands-on experience in bridging raw data preprocessing, database normalization, backend architecture, and frontend dashboard design. We learned how to align technical decisions—such as schema structure, indexing strategy, and chart integration—with practical data exploration needs. In particular, working with both static and dynamic data pipelines reinforced the importance of verifiability, reusability, and scalability in analytical systems. The project also highlighted how real-world data (e.g., SteamCharts) can be turned into actionable insights through clean engineering and thoughtful visualization.

# CONCLUSION

This project successfully designed and implemented a data-driven analytics system for monitoring and visualizing player trends on the Steam gaming platform. By integrating both static (CSV-based) and dynamic (SQL-based) processing pipelines, the system ensured accuracy, reproducibility, and flexibility in data handling. Key modules such as forecasting, seasonality analysis, clustering, and interactive SQL querying were deployed through a Streamlit dashboard, providing users with powerful exploratory and analytical tools.

The dual-mode design allowed us to validate the correctness of the database ingestion process by reproducing visual results from both Excel and SQL sources. Furthermore, extending the analysis from top 5 to rank 6–10 games demonstrated the system’s modularity and scalability. The interactive SQL interface enabled on-demand querying and chart generation, simulating real-world analyst workflows.

Overall, the project showcased the practical application of data science principles in a full-stack environment—from raw data ingestion to frontend web visualization. It deepened our understanding of database normalization, query design, and end-user experience. Future improvements may include integrating real-time data scraping, adding user-uploaded datasets, and supporting more complex machine learning models.

# REFERENCES

[1] SteamCharts, “Steam Charts – Concurrent players on Steam,” [Online]. Available: [https://steamcharts.com/](https://steamcharts.com/" \t "_new)  
[Accessed: Apr. 15, 2025].

[2] MySQL, “MySQL 8.0 Reference Manual,” Oracle Corporation, [Online]. Available: [https://dev.mysql.com/doc/](https://dev.mysql.com/doc/" \t "_new)

[3] Streamlit, “Streamlit: The fastest way to build and share data apps,” [Online]. Available: [https://streamlit.io/](https://streamlit.io/" \t "_new)

[4] W. McKinney, Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2018.

[5] J. D. Hunter, “Matplotlib: A 2D graphics environment,” Computing in Science & Engineering, vol. 9, no. 3, pp. 90–95, 2007.

[6] S. Seabold and J. Perktold, “Statsmodels: Econometric and statistical modeling with Python,” in Proc. 9th Python in Science Conf., 2010, pp. 57–61.

[7] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,” Journal of Machine Learning Research, vol. 12, pp. 2825–2830, 2011.

[8] Python Software Foundation, “Python 3 Documentation,” [Online]. Available: [https://docs.python.org/](https://docs.python.org/" \t "_new)

[9] Pandas Development Team, “pandas-dev/pandas: Pandas,” Zenodo, 2020. DOI: 10.5281/zenodo.3509134

# CONTRIBUTIONS

The majority of the system development, database design, data preprocessing, visualization, and dashboard implementation was conducted by Guoyu Zhao. He led the end-to-end construction of the analytics pipeline, including data ingestion from CSV to MySQL, development of SQL-based visualizations, implementation of forecasting and clustering models, and integration into a dynamic Streamlit dashboard.

Guoyu’s teammate Yi Zhao assisted with initial data extraction and preparation, including compiling game statistics and organizing the raw CSV structure. This contribution provided foundational support during the early stages of the project and helped accelerate subsequent system integration.

### X. AI Assistance Disclosure

Portions of this project report and its associated code benefited from the use of OpenAI's ChatGPT for technical support. The tool was employed to assist with debugging, code optimization, and writing suggestions. All content was critically reviewed and finalized by the project author to ensure academic integrity and compliance with course requirements.