An Examination and Exploration of the Steam API and the Data Available for Public Use

William Loving

December 8, 2024

Affiliation(s):

The University of Virginia

Email Address:

wloving77@gmail.com

Keywords

1. Steam
2. Gaming
3. User Data
4. API
5. Metadata
6. Public
7. Scrape

Abstract

This report summarizes the work done for the DS6600 course at the University of Virginia and encompasses the content of the final project assigned in this course. The course led to us learning and exploring many modern

data engineering and data gathering techniques. The semester began with simple environment setup in Python and exploring techniques like virtual environments and ended with a fully functional environment running in a docker container that could be dynamically spun-up and shutdown at the user's need. We then moved on to data wrangling techniques that focus on the usage of Web APIs to collect and modify publicly available data. This led to webscraping using the BeautifulSoup Python module to pull data from public websites for analysis and to build the foundations of our skills for the project ahead. The semester ended with an overview of database design concepts with database normalization and a final product for the in-class project that gave all students a repository of code and implementation to begin their own projects.

This leads to the project discussed in this report. I wrote and designed a Python Dashy app that wrangles and displays data all sourced from Valve's Steam website/app in one way or another. The code is available online on Github but the bulk appears as Web API requests with a few functions that required more careful execution (Selenium for JavaScript execution). The data were generally formatted as JSON and if it was not it was converted to JSON for ease of use and interoperability with the database.

The data is highly reusable, and the functions are designed to both operate on their own but to also be easily integrated into a larger application. In the future, I hope to build out more initialization functions to get the app running in a single script with the data loaded locally. The only thing preventing this from happening currently is the scale of the problem.

Specifications Table

Subject	Data Engineering and Data Mining and Statistical Analysis
Specific Subject Area	The specific subject area of this report is the user data and gaming space. The goal was to explore and familiarize myself with the data and the challenges associated with pulling that data from different end services.
Type of data	Data is tabular and formatted as JSON in almost all scenarios (API Responses)

Data collection	Data was collected programmatically using Python and the HTTP requests library alongside other modules for modification of the received data (BeautifulSoup, Pandas, etc...)
Data source location	Located on Steam's servers distributed throughout the USA and likely the world because of CDN deployments.
Data accessibility **Raw Data Hosted by Steam**	• Repository name: Steam • Data identification number: N/A • Direct URL to data: All Apps • Instructions for accessing these data: Make API requests programmatically with programming language of choice. See implementation here on my Github: Github Link
Related research article	Python Web Scraping Article: link

Value of Data

- Application Development:
  - Programmers could design applications around the use of Steam user information as well as game data.
- Personal Exploration:
  - If a user has played a game or games on steam they can explore their own personal data and compare to others or add this data to a larger repository of their own user data.
- Why are these data valuable?
  - Data contains current up to date information from Steam's databases and is able to be updated in real time. Applications can be built

using this data and much analysis can be done.

- How can these data be reused by other researchers?
  - The data are general to Steam and can be used extensively for other applications. Data on every app on Steam is a powerful thing and many tools can be used to analyze the broader PC gaming industry as well as specifically Steam.

Background

My motivation for using this as the topic for my project was two-fold. I wanted to explore common data engineering and data wrangling techniques with Python as well as explore a topic I was intimately familiar with. I have been using Steam for the majority of my life at this points (¿ 10 years) and have always been curious how it works under the hood and what eh public API actually offered to end users. With this project I hoped to discover how Steam stored some of their data and how complicated the process of using their API actually was. It so happens Valve has written the Steam website very professionally and the API is both well documented and generally easy to use. Aside from that, scraping the website and getting data from their search engines is also very doable if a little slow. Overall, the motivation was to learn and experiment.

Data Description

The data is aggregated separately but is eventually congregated into a single normalized database schema. The endpoints and tables created are as follows:

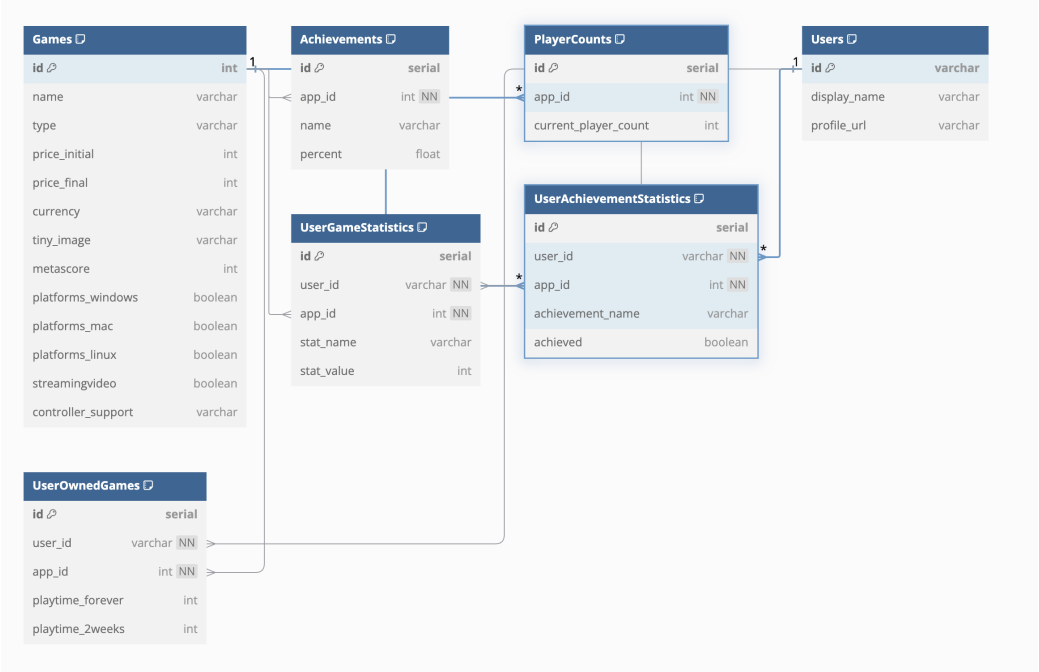
1. Games:
   - Data pertaining to any application that Steam allows end users to install. This data is gathered using the ‘getAllSteamApps‘ function.
2. Achievements:
   - Data pertaining to the achievements for a given steam app. This data is gathered by first querying ‘getAllSteamApps‘ and then choosing a steam app and querying ‘getGameAchievementData‘
3. PlayerCounts

- Data pertaining to the 24hr peak player counts for a given steam app. Similarly to Achievements you first pick a steam app and then query the ‘getGamePlayerCount‘ function to get this data.

4. Users
    - Data pertaining to an individual user or account on Steam. This data is acquired through a function wrapping the Steam seach engine and can be found by running the ‘searchSteamDisplayNames‘ function. There are other related helper functions but that one is the core.
5. UserAchievementStatistics
    - Data bridging the gap between a given user and a steam app. The data is essentially a foreign key mapping between an achievement, a user, and a steam app. This data can be initially aggregated using the ‘getUserAchievementStats‘ function.
6. UserGameStatistics
    - Data gathering a users statistics about a given game. Similar to achievement stats this function returns general statistics about a player in a given game. This data can be found by using the ‘getUserGameStats‘ function.
7. UserOwnedGames
    - The final table mapping games to users. This table contains the relationship of what games a given user owns or what games are present in a users account library. The data can be gathered by using the ‘getUserOwnedGames‘ function.

Below is the E-R Diagram created for this database schema hosted on dbdocs.io. The link to this diagram is here: link

An image of the diagram:



The bulk of this data is currently accessed through public API use and saving of the data to a database is not strictly necessary. The E-R Diagram above is still pertinent however and does illustrate the relationships that exist between the currently existing data. All of the functions in the Github are functional and will work provided an API key has been issues by Steam for the programmers Steam Account.

Experimental Design, Materials and Methods

All of the data was acquired through usage of Python and Juypter Notebooks. Some of the data was available through a public API offered by Steam and some was scraped manually using BeautifulSoup or Selenium depending on the specific scenario. There are twelve functions available in the core 'steam-api-helper.py' module and I will enumerate them now:

1. **getAllSteamApps(self)**
    - Functionality: Returns all of the Steam Apps currently hosted on SteamPowered.com
    - Parameters:
        - Only self

2. **searchSteamApps(self, game-title)**
    - Functionality: Allows a user to search a given Steam App.
    - Parameters:
        - game-title: The title of the game the user wants to search
3. **getGamePlayerCount(self, game-title)**
    - Functionality: Returns the 24hr peak player count for the given game.
    - Parameters:
        - game-title: The title of the game the user wants to search
4. **getGameAchievementData(self, game-title)**
    - Functionality: Returns the global achievement percentages for the entire playerbase.
    - Parameters:
        - game-title: The title of the game the user wants to search
5. **getGameNews(self, game-title, count=5)**
    - Functionality: Returns news articles related to a specific steam app.
    - Parameters:
        - game-title: The title of the game the user wants to search
        - count: The number of articles to return
6. **getUserAchievementStats(self, user-id, app-id, game-title)**
    - Functionality: The achievement completion of a specific user for a specific steam app.
    - Parameters:
        - user-id: The steam user-id for the desired user
        - app-id: The specific app-id of the desired app
        - game-title: The title of the game the user wants to search
7. **getUserGameStats(self, user-id, app-id, game-title)**
    - Functionality: Returns stats for a given steam app for a given steam user.

- Parameters:
  - user-id: The steam user-id for the desired user
  - app-id: The specific app-id of the desired app
  - game-title: The title of the game the user wants to search

8. **getUserOwnedGames(self, user-id, include-playtime=True)**
   - Functionality: Returns the games owned by a specific user.
   - Parameters:
     - user-id: The steam user-id for the desired user
     - include-playtime: A boolean specifying whether to include the user's playtime (in hours)
9. **getSteamIDFromVanity(self, vanity-name, api-key)**
   - Functionality: A helper function to convert a Steam vanity username to a Steam ID usable for other functions.
   - Parameters:
     - vanity-name: The vanity name to convert
     - api-key: Your Steam API Key for authentication.
10. **searchSteamDisplayNames(self, display-name)**
    - Functionality: Returns a list of users with links to their profiles.
    - Parameters:
      - display-name: The username to search
11. **extractSteamID(self, steam-url)**
    - Functionality: Extracts the SteamID from a user's profile URL, primarily a helper function.
    - Parameters:
      - steam-url: URL to a user's profile
12. **isVanityURL(self, steam-url)**
    - Functionality: Helper function to check if a Steam profile URL is a vanity URL or a standard URL.
    - Parameters:
      - steam-url: URL to a user's profile

These functions are all used in tandem with a Python Dashy app that produces a dashboard hosted locally. The dashboard has 4 text inputs and 4 buttons that allow users to type in steam apps they wish to search and the dashboard then populates the search results returned by the above functions. To run the app locally, see the github link below and follow the ReadMe.md in the top level directoy.

The code is hosted on Github and the link to the repository is here: Link

Limitations

There are relatively few limitations but those that do exist do prevent certain interesting features from being available "out of the box". The player count data is only available for the given 24hr day in which the function is run, historical data does not exist and must be build day by day iteratively. This is possible but the lack of historical data is unfortunate. Other than that the data only encompasses Steam App's and information the developers of those app's choose to include on Steam which restricts the possibilities. For example, developers choose the names for their achievements and sometimes those names are far from human readable.

Ethics Statement

The authors has read and follow the ethical requirements for publication in Data in Brief and confirms that the current work does not involve human subjects, animal experiments, or any data collected from social media platforms.

CRediT Author Statement

N/A, Only One Author

Acknowledgements

Jonathon Kropko (Professor of DS6600) for teaching and explaining many of the technologies used to complete this report and corresponding project.

Competing Interests

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

Data Source: source