



THE UNIVERSITY OF  
MELBOURNE

**COMP90068**

**Computer Science Research  
Project**

**on**

**Online Game Analytics,**

by

Niloy Sarkar - [sarkarn@student.unimelb.edu.au](mailto:sarkarn@student.unimelb.edu.au) - 991245

Supervised by

Professor Richard Sinnott

# Table of Contents

---

1. Abstract
2. Introduction
3. Background
  - a) Online Marketplace
4. The Game Clients
  - a) Uplay
  - b) Origin
  - c) Xlive
  - d) Steam
  - e) Epic Games Store
5. Literature Review
6. System Architecture
  - a) Python
    - i. JSON
    - ii. IJSON
    - iii. CouchDB
    - iv. Matplotlib
    - v. Numpy
    - vi. Requests
  - B) NECTAR Research Cloud/Melbourne Research Cloud
  - C) Apache CouchDB
7. Data Collection
  - a) Approach-1
  - b) Approach-2
8. Data Storage
9. Data Refining
10. Results
11. Conclusion

# Abstract

---

This project aims to check the viability of data obtained from video game portals in terms of behaviour analysis. Due to fake profiles, twitter data analysis can lead to false positive results. In order to circumvent that issue, data from online game portals is considered for analysis. While it is acknowledged that fake profiles can exist on online game portals, the games that are bought and the amount of time spent on them cannot be fudged.

A number of online game clients are targeted for data collection and two strategies are developed in order to obtain a complete dataset. The data is then stored in a CouchDB cluster, hosted on Dockers containers installed on Virtual machines, hosted on NECTAR Cloud resource. The data obtained is then updated further with more precise player locations. It is then retrieved from cluster, applied MAP and REDUCE functions manually using python, and then compared with a range of scenarios in order to check is viability.

# Introduction

---

With the dawn of the modern age of technology, the world has transitioned online. The internet as a service has changed from a luxury to a necessity, and social media has been incorporated in our lives in one way or the other. With such drastic changes in our world, companies like Facebook and Twitter have risen to great heights by collecting the data generated by their users, analysing it and selling it to various businesses. These businesses can then further utilize this data to understand what products people prefer or search for, how they react to events, how well they receive a particular product and so on, helping the company to optimize their business analytics, customer engagement, retention and conversion.

Most of these social media companies allow developers access to their API (Application Programming interface) so they may be able to perform the appropriate analytics to improve their systems. These APIs present researchers with the opportunity to collect and build a dataset with millions of users to refine their research and produce accurate results.

When a user browses online, their activities- such as what sites they visit, what pages they spend time- on are recorded. This record is in the form of an 'event' with certain 'properties'. In terms of social media, when one makes a post, the post is the 'event' data that is generated, with data such as text, hashtags, media, as its 'properties'. This behaviour data is what tells companies like Amazon what products an individual user would like to buy, and hence, caters the advertisements accordingly to them.

The analysis of such data to yield results is what is known as 'behaviour analysis'. Social media plays a big role in behaviour analysis. With easily accessible APIs and millions of users to collect data from, it seems like an ideal place for researchers and analysts to build their datasets, However, the data of our 'digital selves' cannot always be trusted.

While most people just use social media to have an online presence, so they may be able to connect with or keep in touch with people, there are some individuals who would rather use these services differently. Since all of us are free to control how we represent our "digital selves", some people see this opportunity to portray themselves

differently from their real-world selves. They might act in ways contrary to how they really are- say or do things they would not normally do- as they feel safer behind the walls of the internet. Some people even go as far as to create fake profiles of other users and masquerade as them. The ethical conundrums of such activities aside, this poses a major hurdle for analysis as well. Such padded/imprecise data would yield false positive results, reducing the accuracy of analysis. Lower is the accuracy, the further one strays from the true value they are aiming to obtain.

In order to circumvent this problem, the focus of this project is on an entirely new dataset, namely, data acquired from video games.

The goal of this project to verify the validity of Online video game data in terms of behavior analytics by comparison to real world data obtained from AURIN(Australian Urban Research Infrastructure Network).

## Background

---

Video games as a hobby have existed for many years. However, it is only with the advent of online gaming as its subsequent features that the video gaming industry has skyrocketed to new heights. From online market places containing games with a multitude of subscription models, to multi-platform releases, live services and downloadable content, this multi-billion-dollar industry garnered over a billion customers, each of whom produce data that is accessible and usable.

It is estimated that Australia's 12.4 million players have spent \$1.3 billion in 2018, making it the world's 14<sup>th</sup> biggest games market (newzoo.com, 2018). Many academic works have focused primarily on characterizing the gamers that make up this market, examining factors such as age, gender, health conditions and economic status. For example, in 2014, a study was conducted on 2,550 gamers to investigate the legitimacy of the stereotype of the unhealthy gamer (Kowert et al., 2013). The study struggled to draw definitive conclusions as the players surveyed were a very diverse group, with a proportion of users having a close relationship to the stereotypes, and other proportions not matching the stereotypes at all. The study did find that the players most correlated to the stereotypical attributes were those more "involved" with the games, and those who dedicated the most time to playing.

The large size of this population has made the in-depth study of player behavior a difficult obstacle to overcome. Surveys, interviews and large-scale data analysis have been carried out to understand player behavior. Data analysis completes this work by allowing researchers to accurately and directly characterize players' playing behaviors on the basis of statistics, such as the player's time spent playing games. In addition, large data sets, and publicly available API frameworks provided by game developers and distributors allow enquiry into the behavior of millions of players at once.

### Online marketplaces:

Initially a luxury, it was only in the early 2000s, when the internet capabilities and computers processors started improving rapidly, that the cost of such technologies dropped enough to be accessible by common households. This led to the birth of online marketplaces, through which people could purchase and, subsequently, play games.

For Authentication, Authorization and Anti-Piracy reasons, these marketplaces take the form of an online portal or game clients, through which a game's online activity can be accessed. These portals also provide a range of features to keep the users engaged enough to not complain about having a game's online access restricted to, and from, that specific portal.

These features, combined with the users' activities, generate a variety of data which, through the rights means and methods, can be utilized for analysis. While there are a number of different platforms that people play games on, each with its own set of portals, the focus of this project is towards PC gaming due to the easier accessibility of data.

# The Game Clients

---

The focus on PC game clients is due to the massive availability of data. While a significant percentage of people do play video games on consoles, a vast majority that population is said to be on the PC as well. Due to the restricted nature of game clients specific to consoles- such as Xbox Live for the Xbox, Playstation Net for the Playstation, and so on- accessing data for consoles is difficult. This factor, combined with the sheer number of PC gaming clients, each with huge amounts of data and relatively open APIs, is what drives this project towards PC gaming for data collection.

While there are many game clients out there, the purpose of this project is to collect players' behaviour data, requiring such information as their play time. For this purpose, we look at the biggest names in the PC gaming industry, namely, Uplay, Origin, XLive, Steam and Epic Games.

## 1. Uplay

With many companies wanting to branch out of the already present and established gaming marketplaces, a range of gaming clients were formed in order to keep 100% of the sales made from their games. One such company was Ubisoft, with its game client- Uplay.

Ubisoft is considered as one of the giants of the gaming industry, with big names under its belt, resulting in them amassing a huge player base. However, being a fairly new client (set up in 2012), Uplay does not provide an open API. To circumvent this, individuals have developed sub-level APIs which allow us to obtain game-specific play data from Uplay.

## 2. Origin

Origin was Electronic Arts (EA) Inc's answer to having their own game client. EA dominates the sports-based games market in the industry, with millions of sports fans accessing their services. Unfortunately, it does not have any open APIs available, making it difficult to access any information.



### 3. XLive

With Microsoft already having a major presence in the console industry, XLive is its attempt to branch out to the world of PC Gaming. Being relatively new, XLive currently uses the same code that Xbox Live does on consoles. However, Microsoft has made many advancements, due to which we can access play data based on certain game characteristics by using the API.

### 4. Steam

The first, and perhaps the biggest gaming client of them all, Steam was created by Valve Corporation back in 2003 and for years faced no competition whatsoever. This allowed them to garner millions of players onto their service. Steam further provides a very open API, which is ideal for collecting such player information.

### 5. Epic Games Store

The newest and biggest competitor to Steam, Epic Games Store was developed by Epic Games as a niche marketplace, i.e. they only hold games published by big companies. This provides an opportunity for data collection, as the more famous a game is, the larger dataset of players we can accumulate.

# Literature Review

---

Being a fairly new form of data, and given its lack of availability until fairly recently, real world data analytics as compared to video game analytics have mostly been two separate areas of study. Early research on video game behaviour had little to no ways of amassing player data through any databases, hence, studies regarding real world behaviour- such as sleep patterns, level of aggression, and the like- had mostly been based on either focus groups or surveys.

It was only from 2016 onwards when research based on analytics performed on data obtained from databases started emerging. Even then, most early research on this was based on what could be inferred from the data, or the relationship between different datasets. It was only after 2018 that data analytics based on real world data, such as the link between bullying and cyber-bullying based on video game analytics, were seen.

A paper published in 2016 called ‘Condensing Steam: Distilling the Diversity of Gamer Behavior’ is of special consideration here. Their project revolved around collecting data from Steam to examine player behavior using play time, game ownership, genre affinity, among others, and characterize it by using heavy-tailed distributions. This paper highlighted the fact that every player on the Steam client is assigned a 64bit Steam ID and using that ID they could obtain a dataset of 108.7 million Steam user profiles for analysis.

A project done by the group Steamdb.info, who specialize in Steam data analytics, was made public in January of 2020. This project further broke down how valve assigns a user their specific 64bit Steam ID. They further built upon it by explaining that every 64bit ID has a 32bit representation as well. This 32bit representation is not only easier to generate, but also be converted to the 64bit version so that it may be used to make

API calls to Steam API and obtain that players’ information.

# System Architecture

---

## Python:

Python is an object-oriented, high level programming language. Due to the ease of use, object-oriented design, easy debugging and availability of a number of modules helpful in data analysis, it proves to be an ideal candidate for this project. A few the modules used are:

### 1. JSON (JavaScript Object Notation)

JSON is an open standard File format that is used to transmit data. Due to its light-weight format for data storage, it is almost universally used for data storage and transportation. Python's JSON library is used to read JSON files, load them into a suitable format (Python dictionaries), convert dictionaries into a JSON format and store JSON formats into a JSON file.

Given that a large part of the project involves data retrieval, JSON proves incredibly necessary and useful to accomplish the same.

### 2. IJSON (Iterative JSON)

The IJSON module is used to iterate over a JSON file rather than loading it all directly. This module is useful when the JSON file to be deciphered is too big for memory to load into, which would either result in an out of memory error or the machine getting stuck. This module circumvents that issue by iterating over the JSON file, line by line in the form of a string. While this does cause an added overhead of converting the obtained string into a dictionary format, its benefits outweigh the disadvantages.

### 3. Couchdb

The Apache CouchDB is a noSQL document-based database that is ideal for storage of large amounts of data. The data stored is in the form of documents, and Python's Couchdb module is needed to perform those operations. The module allows one to create/connect to a database, create/access documents, retrieve/store data from/onto the documents.

It even allows Python to go a step further and perform map reduce queries.

#### **4. Matplotlib**

Matplotlib is a comprehensive library for the creation of static or animated visualization of data in Python. This module helps creating a virtual representation of the data that is obtained to be able to properly visualize the relationships between different datasets.

#### **5. Numpy**

An immensely powerful and widely used library, Numpy is used for scientific computing in Python. In this project, it is utilized perform operations of large datasets, such as a list of all play times of all the cities in Australia. Numpy can perform the computation significantly faster than traditional methods.

#### **6. Requests**

Most of data that is obtained is present on databases on the internet. In order to make appropriate HTTP calls to those databases to get a response data in return, the Requests module is utilized. It is used to make the API requests to different APIs and it receives responses in a JSON format.

#### **NECTAR Research Cloud/Melbourne Research Cloud:**

NECTAR is a platform that offers free, on-demand computing resources to researchers and students at the University of Melbourne. It uses OpenStack Infrastructure as a Service (IAAS) which is an open source cloud technology.

Melbourne Research Cloud is an extension of NECTAR that provides flexible, scalable computing power. By utilizing this computing infrastructure, software, and services, data can be stored and accessed remotely, rapidly and autonomously.

For this project, the following NECTAR resources were utilized:

- 4 Instances (Virtual Machines) each with 30GB inbuilt volume
- 8 VCPUs
- 36GB RAM

## ● 250GB of Volume Storage

The biggest concerns regarding cloud services are regarding trust and security. Not only do you have to trust that the cloud provider will always allow you access to your data and resources (i.e. that they won't have power failures or server corruptions), but also that they will keep your data secure from physical and digital threats. A further issue is in the case of technical difficulties; if one experiences technical difficulties while using a cloud service, they have no choice but to contact the service's support team and hope the issue will be fixed promptly.

NECTAR addresses each of these concerns in an appropriate way for this project. Data security is provided physically by the research cloud, as the infrastructure resides within the University of Melbourne's Data Centre. Interestingly, this does imply that there is in fact a single point of failure for the database, i.e. the Data Centre itself. If the Data Centre were to succumb to some physical disaster and the infrastructure is damaged, the database and instances could be lost. However, this seems highly unlikely.

Data security is provided digitally through the research cloud, as a user has to be connected to the University VPN to access the instances. Furthermore, access to the virtual machines is only granted to those with the valid SSH key. Finally, accessing the database is only possible if a user possesses the correct username and login to the database. Technical support is offered by submitting a ticket or contacting support using 'service-now' or via the support email.

Additional security can be added by configuring security groups, which are attached to each instance. To ensure that only those connected to the University's VPN can access the instances, the various security groups (such as 22/tcp for SSH, 5984/tcp and 4369/tcp for CouchDB) can only be accessed from the IP 128.250.0.0/16.

One interesting issue faced with using the cloud instances was found when attempting to leave a script running on the instance. In the case of the Steam user ID generator and API call, this has to run on the instance indefinitely to harvest as much data as possible. However, the script would stop once the instance terminal was closed. To avoid this issue, a virtual desktop environment was set up on the instance, and from within this environment the harvester was run. Once the harvester was running, the virtual desktop could be closed and the harvester would continue to run indefinitely.

## Apache CouchDB

Couchdb, as discussed above, is a noSQL document database built with Erlang. Erlang applications are ideal for scalable distributed systems which require high availability and fault tolerance, which makes CouchDB an excellent option for a clustered database. CouchDB stores documents in JSON or XML type documents, unlike what one would come across in a relational database. Users can access data through the JavaScript based web UI 'Fauxton', or by using HTTP requests such as 'GET' to retrieve JSON documents and views. Documents can also be updated using HTTP requests such as 'POST' or 'PUT'.

An appealing feature of CouchDB is its omni-directional replication capabilities. This means data contained on any one node is synchronized with other nodes within the cluster via replication. This greatly increases system availability, as when one node is busy with various requests, the other nodes (which contain identical data) can easily handle the new requests. This also allows businesses to route users to nodes which are geographically closest to the user, thus improving latency. The high level of data redundancy achieved by this replicated system makes the database very durable and fault tolerant. If one node fails, data is not lost, and users can still be routed to the remaining nodes while the failed node is repaired and restarted. In effect, a cluster of nodes hosting a CouchDB database has no one single point of failure.

CouchDB suits our needs for this project through its use of Multi-Version Concurrency Control (MVCC), which chooses Availability and Partition-tolerance from the CAP theorem. MVCC is a structure which ensures a high level of availability (nodes in the cluster will never reject a request), and the ability to recover data from a partition by assigning the single databases with revision numbers.

# Data Collection

---

One major problem one would face during video game data collection is the fact that in most cases, the API's are in the form:

“`http://api.steampowered.com/ISteamUser/GetPlayerSummaries/v0002/?key=XXXXXXXXXXXXXXXXXXXXXXXX&steamids=76561197960435530`”

The above example intends to highlight the fact that unlike APIs such as Twitter, which allow one to either track live tweets or get all twitter posts from over a period of time, video game portals do not have such data to present.

Originally, they are marketplaces which also act as a hub to play games online. Due to this, there is no ‘post’ data to look up, or find the author/user from and access their data. Video game API calls, such as Steam’s, involve knowing the ID of the user one needs information about. Given that there is no consolidated database containing such information, this would prove to be difficult. Hence, there are two different approaches for data collection in this project.

## Approach 1

Steam’s Open API requires a user’s Steam ID to be able to fetch their information. However, local APIs for particular games require the players TAG (in game name) for their information to be looked up. The API provided by ‘Tracker.gg’ allows us to track player information from games such as Fortnite, Valorant, Apex Legends, Destiny 2, Halo MCC, TeamFight Tactics, Rainbow Six and Rocket League.

The idea behind the first approach was a bit complex. The player data from each of these games is to be retrieved, and since they cover all the major gaming clients- Fortnite, Valorant and Apex Legends belong to Epic Games; Destiny 2, Halo MCC, Rocket League belong to Steam; and TeamFight Tactics, Rainbow Six belong to Uplay- through that players’ data, information from each of the gaming clients can be retrieved through their friends’ data.

Taking Fortnite as an example, the API call URL is in this form:

```
GET https://api.fortnitetracker.com/v1/powerrankings/{region}/{epic}
```

Since our project focuses on data from Australia, the region code is simply 'AU'.

The problem arises with the Username ({epic}). Step 1 is to acquire usernames. Since each of these games are of a competitive genre and are to be played online, there exist several leaderboards which store the ranking, progress of the top players per region. To utilize such sites, a Python program is written that would crawl through these leaderboards and retrieve the players' name. Since the project is Australia-specific, the program would only filter the ones that are from Australia by parsing through the source of the web page and looking for the specific tags that would contain the above information.

```
<tr class="trn-table_row trn-lb-entry ">
<td class="trn-lb-entry_rank">7</td>
<td class="trn-lb-entry_player">
<a class="trn-lb-entry_name" href="/profile/all/twitchtv eXzacT7">twitchtv eXzacT7</a>
<span class="trn-lb-entry_platform">
<ftr-platform-icon :platform="100" />
</span>
<a class="trn-lb-entry_twitch " href="https://www.twitch.tv/exzact7" target="_blank" rel="noreferrer noopener">
<i class="ion-social-twitch-outline"></i>
</a>

</td>
<td class="trn-lb-entry_stat trn-text--right">8,606</td>
<td class="trn-lb-entry_stat trn-text--right">18,537</td>
</tr>
```

Each player's entry is enclosed in tags (as given above), so this helps in figuring out what parts of the source to look up. The highlighted part is the player's name, and title in the <img class> tag refers to the country code.

Armed with this information, the Python program to retrieve player names is as:



```

url = 'https://fortnitetracker.com/leaderboards/kbm/Top1?mode=all&country=Australia'
page = requests.get(url)
data = page.text.encode('utf8')
data2 = str(data).split('\n')
for x in range(len(data2)):
    if data2[x] != '<td class="trn-lb-entry__player">':
        continue
    temp = data2[x+1].split('>')
    data3 = temp[1].split('<')
    f.write(data3[0])

```

The URL provided takes care of our region restriction, and the rest of the code crawls through the webpage source, takes out all the player names and stores them into a file. This process is then automated so that all pages of the leaderboard are accessed in order to get the maximum number of player names.

Once the player's name is obtained, the appropriate information is provided into the API request link in <http://api.steampowered.com/ISteamUser/GetPlayerSummaries/v0002/?key=XXXXX XXXXXXXXXXXXXXXXXXXX&steamids=76561197960435530> to obtain a JSON containing the player's data Summary for that game, including, but not limited to- time spent on game, play statistics, items owned.

Not only is the data obtained for the player useful, we also obtain their Friend data, using which API calls are made to further enhance the dataset.

## Approach- 2

Utilizing the information obtained from the literature survey, a Python program is made to generate Steam IDs.

```

class SteamID:
    account_id = 0
    instance = 1
    steam_2 = ''
    steam_64 = 0
    type = 1
    universe = 0

    def __init__(self, value):
        self.steam_2 = value
        match = re.match(r"^(?P<universe>[0-4]):(?P<reminder>[0-1]):(?P<id>[0-9]{1,10})$", value)

        if not match:
            print('Incorrect STEAM ID')
        else:
            self.accountId = int(match.group('id'))
            steam32 = (self.accountId << 1) | int(match.group('reminder'))
            self.universe = int(match.group('universe'))
            if self.universe == 0:
                self.universe = 1
            self.steam_64 = (self.universe << 56) | (self.type << 52) | (self.instance << 32) | steam32

```

Based on prior work, we know that the Steam IDs may sometimes be represented as [U:1:22202] which is simply the 64-bit Steam ID represented in a different way. Here U indicated the account type, 1 is the Universe and 22202 is the Account ID. In order to obtain a user's account information, a query needs to be sent to the Steam API with their particular Steam ID.

It is also known that the Steam IDs are assigned in a sequential manner with the base value being '76561197960265728'. The program first generates the 32-bit format Steam ID based on the information obtained about the individual bit values. This 32-bit format is then converted into the 64-bit version so that API calls can be made.

```

def get_player_data(steam_id, db, db2):
    global api_count
    print(cs(f'Fetching player data for steamid {steam_id}', '#b4d3fa'))
    url = f'http://api.steampowered.com/ISteamUser/GetPlayerSummaries/v0002/?key={STEAM_API_KEY}&steamids={steam_id}'
    api_count += 1
    resp = requests.get(url)
    try:
        info = resp.json()['response']['players'][0]
        info.pop('communityvisibilitystate', None)
        info.pop('profilestate', None)
        info.pop('profileurl', None)
        info.pop('avatar', None)
        info.pop('avatarmedium', None)
        info.pop('avatarfull', None)
        info.pop('avatarhash', None)
        info.pop('personastate', None)
        info.pop('primaryclanid', None)
        info.pop('personastateflags', None)
        info = get_owned_games_data(steam_id, info)
        save_to_db(db, steam_id, info, db2)
    except IndexError:
        print(cs(f'Skipping {steam_id} because no data of player', '#ff0000'))

```

Using the Steam ID generated, an API call to Steam is Made that returns the information of that user in a JSON format such as:

```
{'steamid': '76561198043097120', 'communityvisibilitystate': 3, 'profilestate': 1, 'personaname': 'BatDan', 'profileurl': 'https://steamcommunity.com/profiles/76561198043097120/', 'avatar': 'https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/11/11140411cfc5680aa4b31afa54b930c43da23f4.jpg', 'avatarmedium': 'https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/11/11140411cfc5680aa4b31afa54b930c43da23f4_medium.jpg', 'avatarfull': 'https://steamcdn-a.akamaihd.net/steamcommunity/public/images/avatars/11/11140411cfc5680aa4b31afa54b930c43da23f4_full.jpg', 'avatarhash': '11140411cfc5680aa4b31afa54b930c43da23f4', 'lastlogoff': 1592524608, 'personastate': 3, 'realname': 'Sunandan Choudhury', 'primaryclanid': '103582791431031078', 'timecreated': 1307731046, 'personastateflags': 0, 'loccountrycode': 'AU', 'locstatecode': 'NSW', 'loccityid': 5038}
```

The problem with Steam API is that not all relevant information can be obtained about a user from just one API call. In the data seen above, the only relevant information is in the tags-

‘steamid’, ‘personaname’, ‘lastlogoff’, ‘realname’, ‘loccountrycode’ and ‘loccityid’.

Out of this information, it is a common occurrence that a user might not have filled this data in at all.

In order to make the data more readable, all the unwanted keys and their values are popped out of the dictionary done in the code.

The success of this API call confirms that this user exists and the Steam ID generated was correct. Once that is confirmed, the next API call is made to get the actual game related data for that user.

```
def get_owned_games_data(steam_id, info):
    global api_count
    print(cs(f'Fetching owned game data for steamid {steam_id}', '#b4d3fa'))
    url = f'http://api.steampowered.com/IPlayerService/GetOwnedGames/v0001/?key={STEAM_API_KEY}&steamid={steam_id}'
    resp = requests.get(url)
    api_count += 1
    try:
        ownedgames = resp.json()['response']['games']
        tottime = 0
        listy = []
        for x in ownedgames:
            x.pop('playtime_windows_forever', None)
            x.pop('playtime_mac_forever', None)
            x.pop('playtime_linux_forever', None)
            tottime = tottime + x['playtime_forever']
            listy.append(x)
        info['total_playtime'] = tottime
        info['ownedgames'] = listy
    except KeyError:
        print(cs(f'Skipping {steam_id} because no data on owned games', '#ff0000'))
    return info
```

Another problem with the API is the fact that only the data that the user makes available for public to view can be accessed (? Plz confirm this). Therefore, not always does the 'GetOwnedGames' API call yield a positive result. Such results are ignored or not stored at all.

This API call to get the information of all the games a user owns, along with how much time they have spent on them is highly relevant and we append out existing player information dictionary with this data along with calculating and storing the total play time ( total time they have spent on video games) to help during the analysis.

```
def start_program(account_id=0):
    global api_count
    db, db2, dbread = init_couch_db()
    for d_id in dbread.view('_all_docs'):
        doc_id = d_id['id']
        url2 = f'http://api.steampowered.com/ISteamUser/GetFriendList/v0001/?key={STEAM_API_KEY}&steamid={doc_id}&relationship=friend'
        api_count += 1
        if api_count >= 100:
            api_count = 0
            print('waiting for a minute...')
            time.sleep(60)
        resp = requests.get(url2)
        friend = resp.json()
        if 'friendslist' in friend and 'friends' in friend['friendslist']:
            listfriends = friend['friendslist']['friends']
            for x in listfriends:
                steam_id = x['steamid']
                get_player_data(steam_id, db, db2)
```

In order to further enhance the dataset, another Python script is created that accesses the documents that are already stored in the database. From that data, an API call is made to obtain the data of all the friends that particular user has, since these IDs are in the friend data. Since they already exist, the IDs obtained from the friend data is further utilized to get more data.

## Data Storage

---

Utilizing NECTAR, 4 virtual machines running Ubuntu with Docker pre-installed on each are created. These Docker containers are used to host the CouchDB nodes in a clustered set-up. This ensures availability, redundancy and fault tolerance for the data stored.

A shell script is used to first declare an array containing the IP addresses for each of the virtual machine instances. It assigns one of the nodes as a 'master' to optimize the formatting of the cluster.

The shell script pulls the docker image 'ibmcom/couchdb3:3.0.0'. This is done to avoid the extra overhead of configuring the names and their unique Identifier in the cluster. Following that, a container is created using the 'ibmcom' image with the name of 'couchdb<node>' where node is the IP address of the current instance.

Once the Docker is created, the variable 'cont' is exported with the command:

```
$export cont=` echo $(sudo docker ps -aq)`
```

this is defined as the container id.

The container is then started with

```
$sudo docker start ${cont} .
```

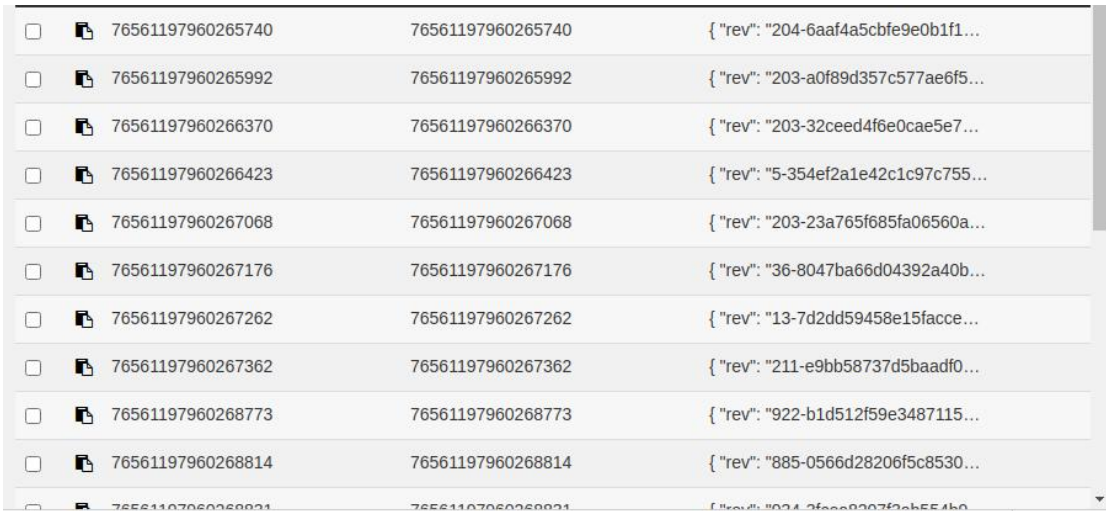
Once the creation of the Docker containers with the couchDB nodes are finalized, the nodes are then clustered one at a time by adding them to the 'master' node's CouchDB 'cluster\_setup' URI with the '- XPOST' HTTP request.












This is preceded by an 'if' statement to check if the current node is a master node or not before the request can be made. The nodes are added using the port 5984 as it is the standard clustered port used by CouchDB 3.0.0 for HTTP API requests.

A final 'if' statement is used to check if current node is the last in the array of nodes. If true, that means all the nodes have been clustered and the setup is finalized by posting the JSON containing the key-value pair "action": "finish\_cluster".

Now that the database is ready, the data is stored into them by simply adding a few more lines to the Python's scripts that originally collect the data.

The CouchDB server is initialized and for every Steam ID a separate document is created with the Steam ID itself being the document ID and key values. (see FIG)



<input type="checkbox"/>		76561197960265740	76561197960265740	{ "rev": "204-6aaf4a5cbfe9e0b1f1..." }
<input type="checkbox"/>		76561197960265992	76561197960265992	{ "rev": "203-a0f89d357c577ae6f5..." }
<input type="checkbox"/>		76561197960266370	76561197960266370	{ "rev": "203-32ceed4f6e0cae5e7..." }
<input type="checkbox"/>		76561197960266423	76561197960266423	{ "rev": "5-354ef2a1e42c1c97c755..." }
<input type="checkbox"/>		76561197960267068	76561197960267068	{ "rev": "203-23a765f685fa06560a..." }
<input type="checkbox"/>		76561197960267176	76561197960267176	{ "rev": "36-8047ba66d04392a40b..." }
<input type="checkbox"/>		76561197960267262	76561197960267262	{ "rev": "13-7d2dd59458e15facce..." }
<input type="checkbox"/>		76561197960267362	76561197960267362	{ "rev": "211-e9bb58737d5baadf0..." }
<input type="checkbox"/>		76561197960268773	76561197960268773	{ "rev": "922-b1d512f59e3487115..." }
<input type="checkbox"/>		76561197960268814	76561197960268814	{ "rev": "885-0566d28206f5c8530..." }
<input type="checkbox"/>		76561197960268821	76561197960268821	{ "rev": "204-2f6e88207f2eb554b0..." }

## Database refining

---

The data obtained for each player contains a few flaws. While the country code obtained from the 'GetPlayerSummaries' API call being 'AU' lets us filter our data for Australia, the 'loccityid' key contains a code for the city's location and not the actual name. Similarly, the 'GetOwnedGames' API call gives us the App ID of the games and not their actual information. One more flaw is that since not all players provide all the information or keep it private, when we request for the information there are no key values returned for the parts that are unavailable. This might prove to be a problem during the retrieval and analysis of the data.

For the refining process, the first step is to create a simple python script that makes the API call 'GetAppInfo' by iteratively generating all possible Steam App IDs starting from 1, the script ends when no further information can be obtained and all the data obtained from this script is stored into its own JSON file.

To obtain the city information we refer to the 'github.com/RudeySH/SteamCountries' Repository. This Repo contains a JSON with all the City Details for each country, This JSON is stripped down to only contain information about Australia since that is what is relevant to the data obtained.

Once all the information is collected, the original Python script that collects the data is modified to have the above obtained JSON files for reference. When the API call returns a value, the JSON returned is parsed through to check if all the key values are present. If not, a 'NULL' keyword is added for reference during the analysis. If present, the 'loccityid' key is checked and based on the ID value it returns, the subsequent city information is retrieved from the modified City Details JSON and added to the user information dictionary. Similarly, all the App ID of all the Owned games for the user are checked, their respective information is fetched from the AppInfo JSON and added to the Dictionary.

Once the Refining process is complete, the Document is stored on CouchDB as below.



```
[
  {
    "_id": "76561197960269493",
    "_rev": "999-6521c5e93b22d59b0fcc683275fc74f0",
    "steamid": "76561197960269493",
    "personaname": "SquattingCow",
    "commentpermission": 1,
    "realname": "Matt",
    "timecreated": 1063347797,
    "loccountrycode": "AU",
    "locstatecode": "ACT",
    "loccityid": 4922,
    "name": "Canberra",
    "coordinates": "-35.308235,149.124224"
  }
]
```

## Data Analysis

---

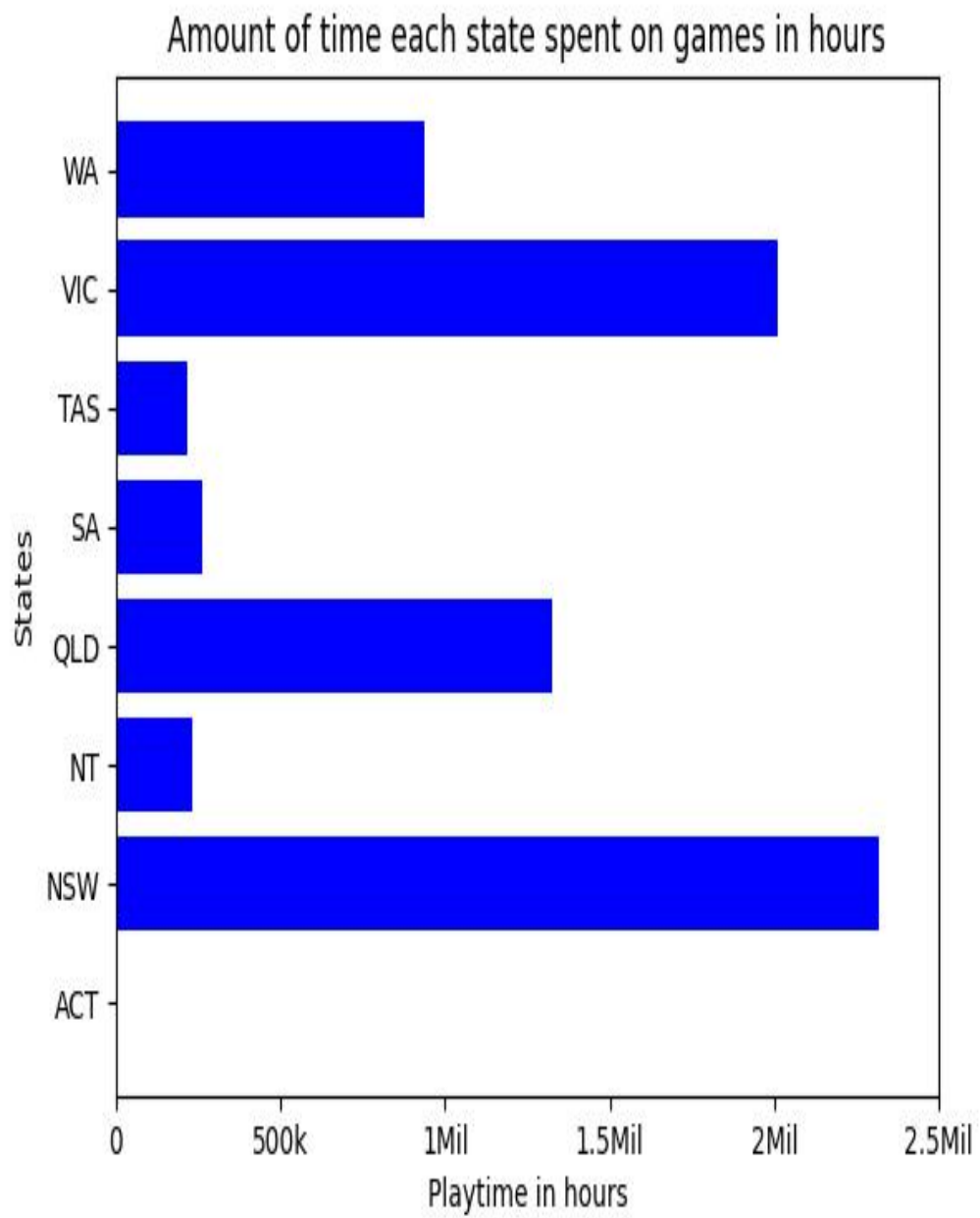
For the analysis part of our code, the data must first be retrieved from the CouchDB clusters, mapped and reduced and then used for analysis.

Each document retrieved is in the form of a JSON. The first task is to filter out all the NULL values. From the remaining data that is left, based on city data present in the document, a consolidated dataset for each city containing the number of players, the number of games, their playtimes and expenses is created. From there, the data is further refined into smaller more precise datasets based on the locations.

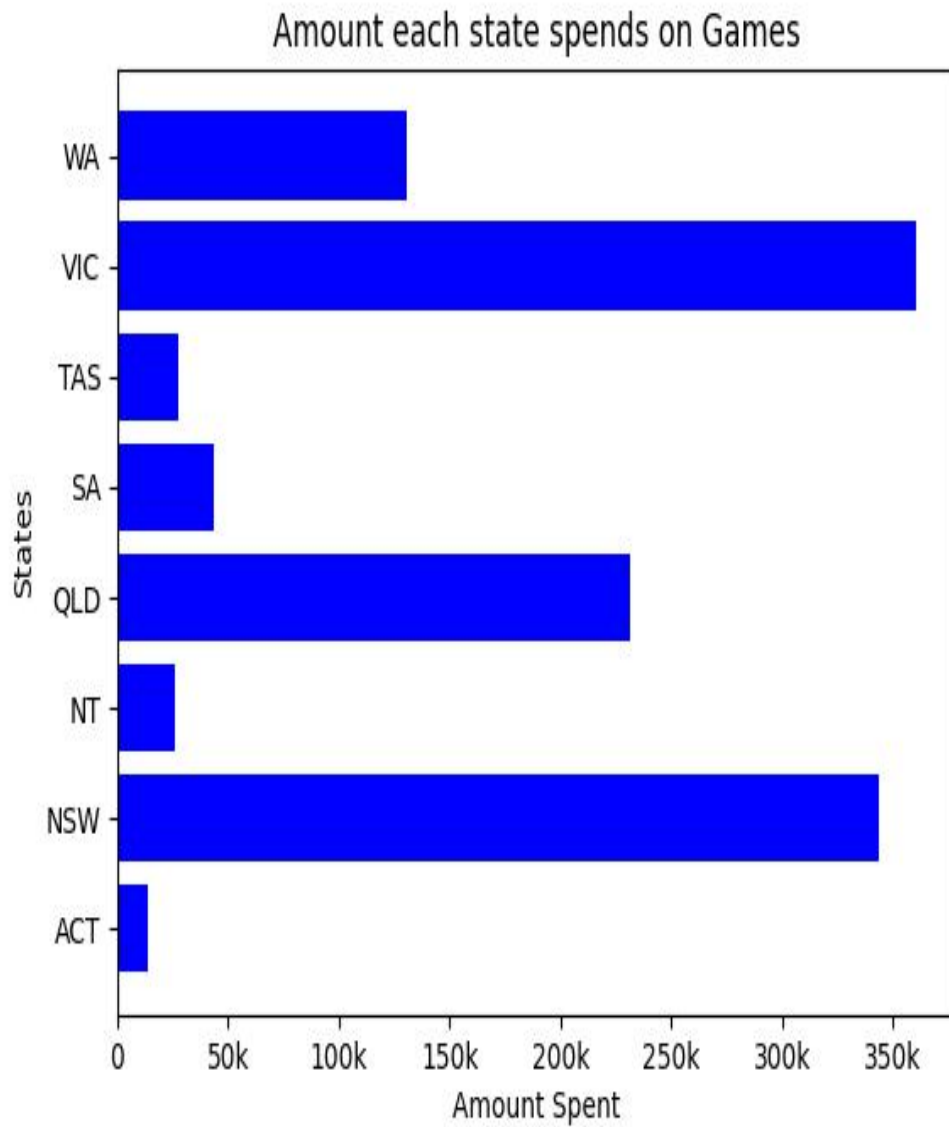
From the resulting data, a few comparison scenarios are generated to check the viability of data. These are comparisons between the playtimes between different states, the amount of money each state spends on games, the top 5 Cities that spend the most on games and the top 5 cities that spend the most time on games.



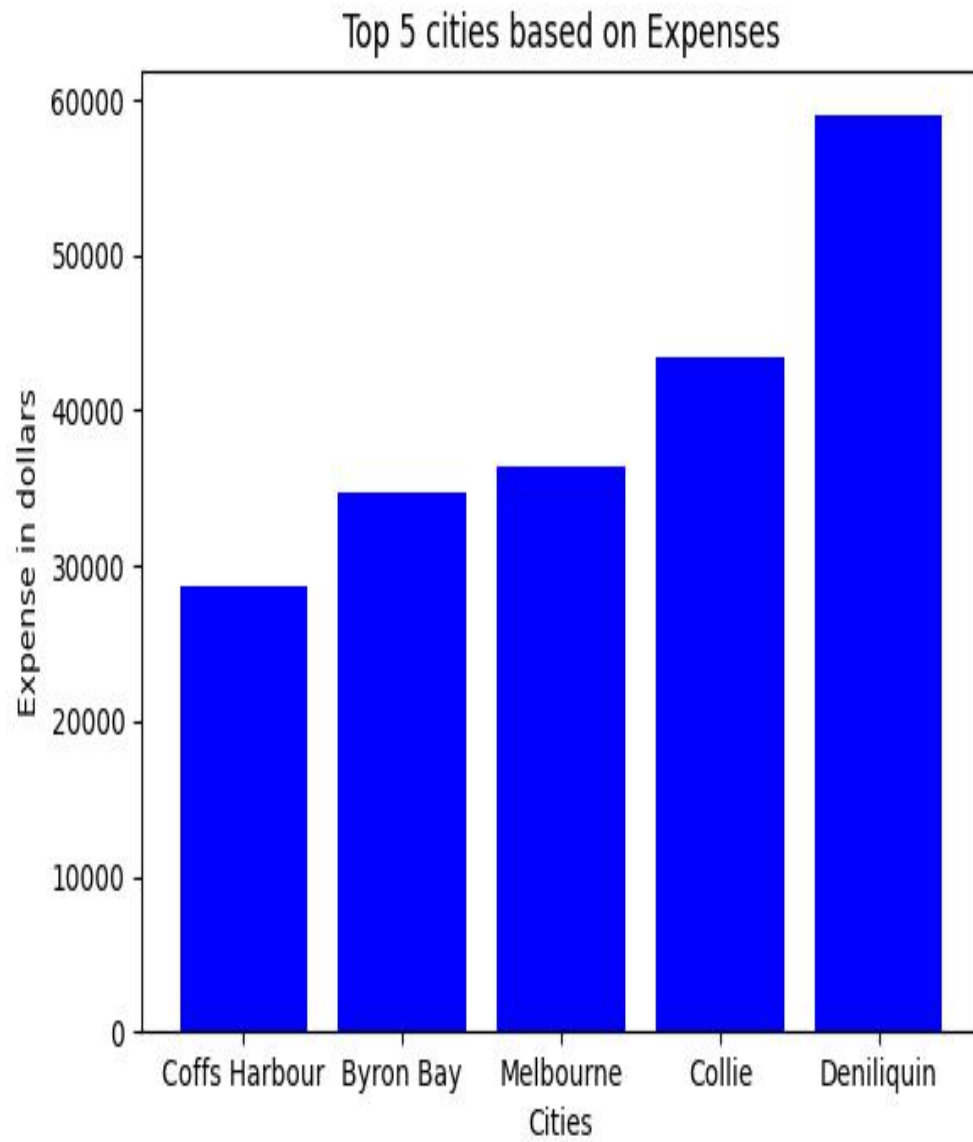
**The number of hours spent on games based on States.**



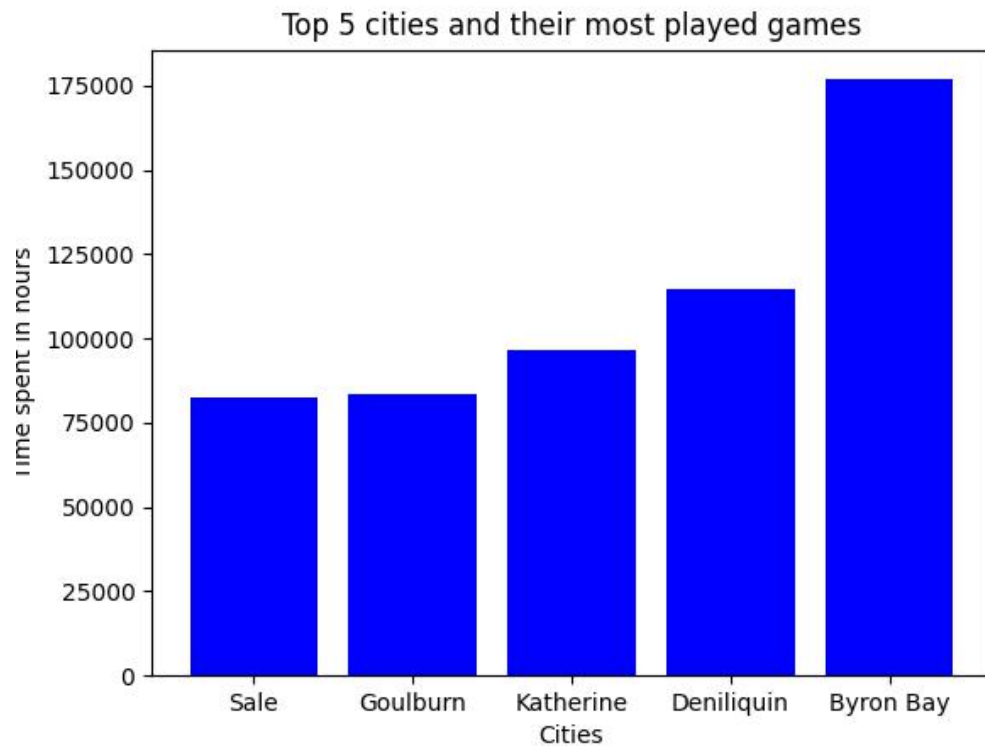
### Amount of money Each state spends on Games



### Top 5 Cities based on their expenses on games



## The Top 5 cities with the most hours spent on video games



In the Above chart the game played the most in each of these cities along with their genres are:

In Sale-Half Life 2, Genre- Action

Goulburn-SimCity™ 4 Deluxe Edition, Genre Simulation

Kathrine-Raycatcher, Genre-Action

Deniliquin-Left 4 Dead 2, Genre-Action

Byron Bay-Left 4 Dead, Genre-Action

These are just a few of the number data combinations that can be made from the sheer amount of data that has been obtained.

# Results

---

For the analysis, we consider 2 scenarios.

## 1. Academics

According to an analysis of 12,000 children in Australia, it was found out that children who play online video games tend to do better in academics.

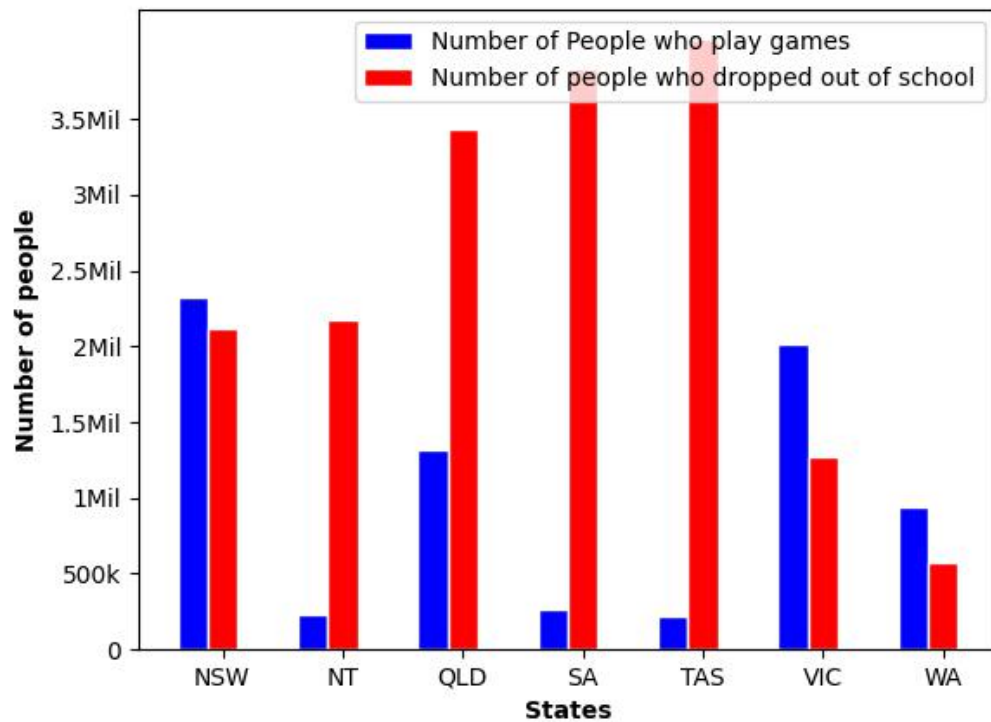
It was noted that students who play games tend to score 15-17 points above average in school tests.

It can be noted that the inverse could be true as well, students who play less video games would tend to perform bad in school.

For the validation, a dataset of number of students who dropped out of school was obtained from AURIN.

This dataset was then cross-referenced by the number of people who play video games in these states. By doing so the following graph is

obtained:



From the above a certain relationship can be noted.

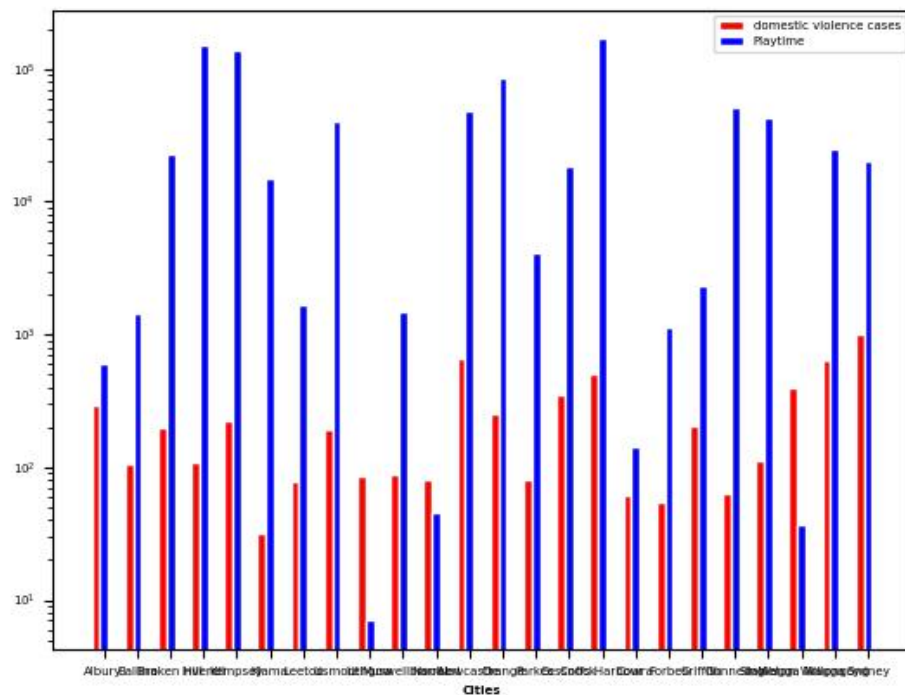
In states that have a high number of students who drop out of schools, the number of people who play games is significantly lower than in the ones where people who play games is high.

## 2. Violence

The age-old argument against video games is that “video games cause violence”. The long-standing debate of whether violence depicted in video games can trigger real-world violence has had research conducted that has proven both sides. There have been a slew of studies claiming to find a link between violence in video games and real-world aggression, although this when analyzed was revealed to be a very casual relationship.

There have been several counter studies to prove otherwise as no link has been found between the number of people who play games and the number of violent incidents.

In order to pick a side, a dataset from AURIN is pulled containing all the number of violent incidents and crimes conducted across different cities of Australia, all the crime statistics are combined into one value for each city for ease of representation, and then plotted against the number of people who play games across those cities. The following graph was obtained.



From the chart above it can be noted that no clear relationship can be pulled from the two datasets. Cities with a higher number of people who play games do not necessarily have more cases of violent crimes conducted, and vice versa. It can also be noted that upon analysis it was observed that the majority of the cities play action/violence-oriented games, however the data is too varied for a relationship to be obtained, proving that there is no clear relationship between video games and violent activities.

## Conclusion

---

From the research conducted in this project, it can be seen that data obtained from online game portals can indeed be useful when analyzed and represented appropriately.

However, given the paucity of research conducted using these datasets, they are in a haphazard state as mentioned above. Not all the data for a single user can be obtained with one API call. Furthermore, there is a restriction of 100 API calls per minute. For this project, on an average, three to five API calls need to be made to obtain all the information required. This severely hinders the speed of progress.

If more researchers attempt to utilize video game datasets for their research purposes, as observed in recent trends, perhaps the clients that provide them would try and improve ease of access to aid further research.

## References

---

1. H. Kwak, C. Lee, H. Park, and S. Moon. What is twitter, a social network or a news media? In Proceedings of the 19th international conference on World wide web, pages 591--600. ACM, 2010.
1. Mark O'Neill, Elham Vaziripour, Justin Wu, and Daniel Zappala. 2016. Condensing Steam: Distilling the Diversity of Gamer Behavior. In Proceedings of the 2016 Internet Measurement Conference (IMC '16). Association for Computing Machinery, New York, NY, USA, 81–95.
2. "Scanning all possible Steam IDs, and what we have found". SteamDB, <https://steamdb.info/blog/scanning-all-steam-ids/>
3. R. Becker, Y. Chernihov, Y. Shavitt, and N. Zilberman. An analysis of the Steam community network evolution. In Electrical, Electronics Engineers in Israel (IEEEI), 2012 IEEE 27th Convention of, pages 1--5. IEEE, 2012.
4. Drummond, Aaron, and James D Sauer. "Video-games do not negatively impact adolescent academic performance in science, mathematics or



reading.” *PloS one* vol. 9,4 e87943. 3 Apr. 2014,  
doi:10.1371/journal.pone.0087943

5. “Australia Games Market 2018.” Newzoo,  
newzoo.com/insights/infographics/australia-games-market-2018/.

6. “Steam: Game and Player Statistics.” Welcome to Steam ,  
store.steampowered.com/stats/.

7. D. J. Kuss and M. D. Griffiths. Internet gaming addiction: A systematic review of empirical research. *International Journal of Mental Health and Addiction*, 10(2):278--296, 2012.

8. “Fortnite LeaderBoards” , <https://fortnitetracker.com/leaderboards/>.

9. “Tracker Network”, [tracker.gg](https://tracker.gg).