IndieP_Data_Collection

February 19, 2020

1 IndieP: Predicting the Success of Indie Games on Steam

Note: See github.com/argwood for full project with nicer formatting (as well as other projects)!

2 Part 1: Data Collection

If you haven't been to a video game convention such as PAX (the Penny Arcade Expo) before, let me paint a picture for you. After you push through the thousands of fans and cosplayers crowding around the massive Nintendo, Sony, Ubisoft or Square Enix booths with hundred-inch flat screen TVs and LEDs flashing from signs hanging off the ceiling, at the very back of the convention hall is the Indie Mega Booth, where developers of around 80 indie games are chosen to showcase their work and get feedback from con-goers. And then, maybe upstairs, in another corner of the show floor, are 10 more indie games highlighted as the PAX 10, the indie games of the year that have been designated the "best" by some committee. But what do they mean by "best"? What factors go into determining which of the thousand games released each year will succeed, which will be remembered for years to come, which will fade away with time, and which will never even make it into the (albeit somewhat faded) spotlight?

Almost everyone has a sense of what big name games have been successful over the years, even if you aren't a hard-core gamer - Overwatch and Fortnite both had their moments of fame not too long ago, and there was Minecraft, Final Fantasy, and World of Warcraft before that - but indie games are a whole different world. How many people outside of that industry can tell you which indie game was the most successful last year? In the world of indie game development, there's no massive corporation deciding what kind of game to make based on what their analytics department has told them will sell the most copies. Instead, there's just a few nerds (maybe a programmer, an artist and a writer) with a cool idea and the skills to create a product out of it.

In this project, my goal is to explore which factors play into the success or failure of an Indie game, thinking about how this may differ from games that are created by larger corporations. While full explorations of video game data are not uncommon (there's some datasets on Kaggle, for example), most focus on the most popular games without putting thought into the games that are lesser known but potentially just as qualified to be considered the "best" by someone's metric.

This project will be split into 4 parts: First, we will collect a messy dataset of all of the games in the "Indie" genre of the Steam store. Then, we will clean up the data, decide which features to keep, and create a high-quality dataset suitable for analysis and as input for Machine Learning algorithms. Next, we will analyze our data by exploring the correlations between features to see what we can learn without using Machine Learning. Finally, we will create a training set and test set from our data, and create a Machine Learning framework to measure "success" of a game. In this final section, we will first address what metrics may be used to measure success, and what Machine Learning algorithms are best suited for the job at hand.

2.1 1. Data Collection

The goal of the first part of this project is to create a data set that includes details for every indie game available on Steam. As we ultimately want to gain insight into what features have the greatest effect on the success of a game, we will want to include qualities of the games themselves (price, genres, game descriptions, number of achievements, etc), but also metrics that may be useful for quantifying sucess, such as ratings, number of users, and number of copies purchased.

We'll be collecting this data in three parts, and combining everything together later as part of the data cleaning. First, we will scrape a filtered list of Indie games from Steam's online store to get basic information such as the title and AppID, which is the unique identifier for each game (or other medium). We will then use the Steam Store API to get as much information as we can about each game in our list. Lastly, since the Steam Store doesn't provide information about how many (active or inactive) users each game has, we will use an external API called SteamSpy, which has this information readily available.

2.1.1 1.0 Useful Resources for Data Collection

Below are a few of the resources I used to conduct Part 1 of this project, including inspiring/helpful webpages and API documentation:

1. Steam API Documentation
2. User-made Steam Storefront API Documentation
3. SteamSpy API Documentation
4. Scrapy Tutorial
5. Nik Davis' Data Science Blog
6. A note on the accuracy of SteamSpy

2.1.2 1.1 Gathering Initial Data from Steam Store

While the genre tags of each Steam game are accessible through Steamfront, the Steam API only provides a list of *all* available apps and no way to filter through them efficiently. A few alternative user-made APIs offer a wider variety of API features and could be filtered by genre, but as they are user-maintained and not always online, I chose to avoid these alternatives for this initial step and make sure our data comes directly from the main source (though we will use one user-made API, SteamSpy, later to collect data on estimated game sales/popularity since as far as I know there's no other way to get this information). Rather than make an API request for every single app (~70,000 requests) and check if it fits our genre, it will be more efficient to scrape a list of Indie games from Steam's online store:

https://store.steampowered.com/search/?sort_by=Name_ASC&tags=492&category1=998

Here, I've filtered Steam's full game list to include only apps with the tag "indie," further narrowed the list down to include only games (excluding other software, downloadable content, demos, and soundtracks), and sorted the list alphabetically. There's a total of 27,271 results at the time of writing this.

I've learned to love (and hate) BeautifulSoup for prior web scraping projects, but for this project I've chosen to use the simpler and elegant Scrapy which will have a much easier time crawling through the 1,085 pages of results while also providing more flexibility since Steam updates their store frequently (plus, I've never used it and I like trying new things!). The ability to easily control a rate limiter for Scrapy is also a plus, since I want to ensure that the scraper remains well-behaved.

For the initial data collection, I scraped the following features for each game, as they are universal, easy to understand, and all accessible without needing to follow additional links (thus increasing the number of requests): * AppID: * Url: * Title: * Release Date: * Price:

When I initially gathered this data, I only collected the Title, Release Date, and Price - but quickly discovered that there's no guarantee that these three features would be unique and there could easily be multiple games with the same title. The AppID, however, is a unique identifier and will be used to distinguish between data points throughout.

The spider I wrote to extract this data can be found in `./steam-scrapy/steam_scrape/spiders/scrapy_indie_all.py`. Here, I will just load in the data obtained from this step.

[4]:
```
import csv
import json
import time
import os

import pandas as pd
import requests
import numpy as np

pd.set_option("max_columns", 1000)
```

To start, let's load in the basic data we gathered into a Pandas dataframe - for now, all we really need here is the AppIDs, but we'll load in everything to begin. We'll use the rest of the columns later when we're cleaning up the full dataset, in the case of apps that had unsuccessful API calls.

[9]:
```
def get_scrape_data(f):
    ''' Load in json data that we previously collected to extract a list of␣
 ↪appIDs for all Indie games '''

    data = pd.read_json(f)
    return data
```

[10]:
```
scrapy_data = get_scrape_data('./steam-scrapy/steam_scrape/output/
 ↪indie_all_final.json')
```

[11]:
```
scrapy_data.head()
```

[11]:
```
      appid  price  release_date                   title  \
0  1034230         Mar 17, 2019                     ***
1   603750  $2.99  Mar 10, 2017        - Arcane Raise -
```

```
2   729370  $14.99  Jan 23, 2019                 -KLAUS-
3   638510                 2019                      .Age
4  1091520   $0.99   Sep 9, 2019  (Malicious Dinner)

                                                 url
0  https://store.steampowered.com/app/1034230/_/?...
1  https://store.steampowered.com/app/603750/_Arc...
2  https://store.steampowered.com/app/729370/KLAU...
3  https://store.steampowered.com/app/638510/Age/...
4  https://store.steampowered.com/app/1091520/Mal...
```

Right off the bat, we can see that some of the titles are questionable, so we'll need to later dig in to the data to ensure that everything we include in our dataset would be considered a game (no demos/soundtracks/videos/etc - we filtered these out initially but you never know what crept through). We can also see that not all of the release dates have the full day, month, and year, and that not all games have prices (of course some are free to play). We'll keep everything as is until we have all of the data collected, and then later work on cleaning it up into a nice dataset.

[14]:
```
applist=scrapy_data['appid']
```

[15]:
```
applist.head()
```

[15]:
```
0    1034230
1     603750
2     729370
3     638510
4    1091520
Name: appid, dtype: object
```

[16]:
```
applist.describe()
```

[16]:
```
count      22758
unique     22753
top       434260
freq           2
Name: appid, dtype: object
```

For some reason, it looks likes there are a few duplicates in the applist, which is suprising because the AppID should be a unique identifier. Let's look under the hood to see if we can figure out why:

[17]:
```
scrapy_data[scrapy_data.duplicated(subset='appid')]
```

[17]:
```
         appid         price  release_date  \
4612    230820        $24.99  May 28, 2013
12707   430280         $4.99  Dec 21, 2015
12782  1155470          Free   Oct 7, 2019
12833   434260         $0.99  Feb 15, 2016
```

```
15982   557260  Free to Play   Jan 16, 2017

                                         title  \
4612   The Night of the Rabbit Premium Edition
12707                         Nature Defenders
12782                   Mythic Ocean: Prologue
12833                          My Name is Mayo
12833                                     iREC

                                                     url
4612   https://store.steampowered.com/sub/28005/?snr=...
12707  https://store.steampowered.com/app/430280/Natu...
12782  https://store.steampowered.com/app/1155470/Myt...
12833  https://store.steampowered.com/app/434260/My_N...
15982  https://store.steampowered.com/app/557260/iREC...
```

We can see that, for some reason, these 5 AppIDs have been included more than once. Before just deleting the duplicates, we want to see if the full rows are duplicated, not just the AppID:

[18]:
```
scrapy_data.loc[scrapy_data['appid'] == '230820']
```

[18]:
```
       appid   price  release_date                                    title  \
4611  230820          May 28, 2013                     The Night of the Rabbit
4612  230820  $24.99  May 28, 2013  The Night of the Rabbit Premium Edition

                                                    url
4611  https://store.steampowered.com/app/230820/The_...
4612  https://store.steampowered.com/sub/28005/?snr=...
```

[19]:
```
scrapy_data.loc[scrapy_data['appid'] == '430280']
```

[19]:
```
        appid  price  release_date             title  \
12658  430280  $4.99  Dec 21, 2015  Nature Defenders
12707  430280  $4.99  Dec 21, 2015  Nature Defenders

                                                     url
12658  https://store.steampowered.com/app/430280/Natu...
12707  https://store.steampowered.com/app/430280/Natu...
```

[20]:
```
scrapy_data.loc[scrapy_data['appid'] == '1155470']
```

[20]:
```
         appid price release_date                   title  \
12727  1155470  Free  Oct 7, 2019  Mythic Ocean: Prologue
12782  1155470  Free  Oct 7, 2019  Mythic Ocean: Prologue

                                                     url
12727  https://store.steampowered.com/app/1155470/Myt...
```

```
12782  https://store.steampowered.com/app/1155470/Myt...
```

So, it looks like in the first case, both the original and premium versions of the game are given the same AppID - this could be something important to keep in mind for later, but since we'll be making API calls based on a specific AppID, it doesn't make sense to keep the duplicates in our list since you can't have two different request responses from the same API call for an AppID - presumably, info on both versions (or at least that multiple versions exist) will be encoded into the same request, but we'll want to check this later. Regardless, the remaining four duplicates seem to just have printed twice, so we can go ahead and delete them.

[21]:
```
scrapy_data = scrapy_data.drop_duplicates(subset='appid')
```

Before we finish and write our applist to file, there's one other caveat from the scraped data: it turns out (after a handful of confusing error messages from the Steam API later on...) that in the cases where the game includes some DLC packages, soundtracks or other media, the AppID that was scraped from the store was actually a string of concatenated AppIDs for the main game as well as the other media. For example, you can see the first 5 instances of this below, but there's actually quite a few of these! In all cases, the first AppID listed is the game, and those following it are additional media. Since we only want the games and aren't interested in the DLC or soundtracks (in fact, we'll later collect just a "yes or no" answer as to whether the game has DLC, etc), we want to keep only part of these strings up until the first comma (we'll likely use this technique again later in the project for cleaning up and transforming our messy data, as well as creating new features out of existing ones). Then, we'll want to check for duplicates again.

[22]:
```
stop = 5
c = 0
for index, row in scrapy_data.iterrows():
    if ',' in row['appid'] and c<stop:
        print(row['appid'])
        c+=1
```

```
258090,262141,262142,262230,262300
220820,220822
92300,92302,92303
15500,15520
273700,259830
```

[23]:
```
scrapy_data['appid']=scrapy_data['appid'].str.split(',', n=1, expand=True)
```

[24]:
```
applist=scrapy_data['appid']
```

[25]:
```
applist.describe()
```

[25]:
```
count      22753
unique     22695
top       209190
freq           4
```

```
Name: appid, dtype: object
```

[26]:
```python
applist = applist.drop_duplicates()
```

[27]:
```python
applist.describe()
```

[27]:
```
count       22695
unique      22695
top       1163290
freq            1
Name: appid, dtype: object
```

Now, we have a dataset of AppIDs with no duplicates and no concatenated strings that will throw errors upon API requrests. We'll now write the data to file for safe keeping, take this list of apps to the Steam Store API, and start filling in our dataset.

[322]:
```python
applist.to_csv('./data/applist.csv')
```

2.2 1.2 Using Steam Store API to Get Game Data

Now that we have a list of AppIDs for all of the Indie games on Steam, we can use the Steam API to obtain as much information as we can about each app. We start with some functions to make and parse through API requests, and take a look at the Steam API data that's returned for one appID before iterating through our whole list.

[28]:
```python
def get_api_request(url, parameters=None):

    try:
        response = requests.get(url=url, params=parameters)
    except SSLError as err:
        print('SSL Error:', err)

        for i in range(5, 0, -1):
            print('Waiting... ({})'.format(i))
            time.sleep(1)
        print('Retrying...')

        # recusively try again if errored
        return get_api_request(url, parameters)

    if response:
        return response.json()
    else:
        # If there's no response it usually means too many requests in a given␣
↪time
        print('No response, waiting 10 seconds...')
        time.sleep(10)
        print('Retrying...')
        return get_api_request(url, parameters)
```

[29]:
```
def parse_steam_request(appid):
    '''Makes a request to the Steam Store API and returns all data for a given␣
 ↪AppID.

    Returns : data in json format
    '''

    url = "http://store.steampowered.com/api/appdetails/"
    param = {"appids": appid}

    data = get_api_request(url, parameters=param)

    json_app_data = data[str(appid)]

    if json_app_data['success']:
        data = json_app_data['data']
    else:
        data = {'steam_appid': appid}

    return data
```

Pick an arbitrary appID to make sure the API call to Steam works, and see what kind of data we're getting.

[30]:
```
test_data=parse_steam_request(1091520)
```

[31]:
```
test_data.keys()
```

[31]:
```
dict_keys(['type', 'name', 'steam_appid', 'required_age', 'is_free',
'detailed_description', 'about_the_game', 'short_description',
'supported_languages', 'header_image', 'website', 'pc_requirements',
'mac_requirements', 'linux_requirements', 'developers', 'publishers',
'price_overview', 'packages', 'package_groups', 'platforms', 'categories',
'genres', 'screenshots', 'movies', 'achievements', 'release_date',
'support_info', 'background', 'content_descriptors'])
```

And we can take a peak at the full data for one game (trying not to think too much about this particular game I picked with questionable content but no age restrictions...I guess this is an indication of the full scope of Indie games we're going to find...)

[32]:
```
print(test_data['name'])
print(test_data['about_the_game'])
print(test_data['genres'])
```

```
(Malicious Dinner)
```

```
Malicious Dinner is an AVG indie game I made <strong>(not a Gal or sexual
game)</strong> after work. It's aslo my 1st diy game. Ur choices will drive the
entire plot. Pls do it wisely. Different choices will have influence on ur
quality, popularity, game difficulty and results. <br><img src="https://steamcdn
-a.akamaihd.net/steam/apps/1091520/extras/games.jpg?t=1572422069" ><br>Including
1 bonus egg, 4 casual games, 5 careers, 6 results and 100+ hidden items. This is
only 1st chapter of the story. Will optimize, update and add 4 more chapters of
DLC later.<h2 class="bb_tag"><strong>Chapter 1</strong></h2><img
src="https://steamcdn-a.akamaihd.net/steam/apps/1091520/extras/-
_1-000.png?t=1572422069" ><br>Now U play a school girl who is invited to a
volunteer-reward active, only to find it's actually a mailcious dinner. How to
survive from it?<br>When u r toasted, drink or reject?<br>When u r framed, stay
or hide?<br>When u r harassed, hesitate or refuse?<br>When u r abused, keep
quiet or resist?<h2 class="bb_tag"><strong>Addition</strong></h2><img src="htt
ps://steamcdn-a.akamaihd.net/steam/apps/1091520/extras/en.png?t=1572422069"
><br>click the top left button of start menu can switch Chinese/English
language.<br><a href="https://store.steampowered.com/app/1099400/_Evkworld/"
target="_blank" rel="noreferrer"  id="dynamiclink_1"
>https://store.steampowered.com/app/1099400/_Evkworld/</a><br>It's a game made
by Evkworld game engine ↑↑↑<br>E-mail <a
href="mailto:meyoudian14@yeah.net">meyoudian14@yeah.net</a> or leave a
comment if u have any idea. Please do NOT do advertising / result spoiling /
flaming or abusing in review. THX!
[{'id': '4', 'description': 'Casual'}, {'id': '23', 'description': 'Indie'},
{'id': '3', 'description': 'RPG'}, {'id': '2', 'description': 'Strategy'}]
```

For now, let's keep everything except: `detailed_description` (because, if you look, it can be *really long* and there are two other categories with very similar information), `pc_requirements`, `mac_requirements`, `linux_requirements` (because there's a feature for `platforms` which will give us plenty of information there), `legal_notice` (because who wants to deal with that...), and `email` (probably won't be relevant). We'll keep all of the images such as screenshots and assets for now, even if we end up not using all, because as a stretch goal we may want to do some image analysis later to see if we can learn anything about the art styles being used in each game.

[33]:
```
cols_to_keep = [
    'type', 'name', 'steam_appid', 'required_age', 'is_free', 'about_the_game',␣
 ↪'short_description', 'supported_languages',
    'header_image', 'website', 'developers', 'publishers', 'price_overview',␣
 ↪'packages', 'package_groups', 'platforms',
    'categories', 'genres', 'screenshots', 'movies', 'achievements',␣
 ↪'release_date', 'background', 'content_descriptors'
]
```

2.3 1.3 Collecting, Formatting and Writing Steam Store Data

With the framework set up to collect the data we want from the Steam Store API, we can go ahead and iterate through the full list of AppIDs, writing to a CSV file. Since we have a large number of API requests to make, in order to avoid catastrophic data loss or massive overloading of the API, we'll want to split up the requests into batches and make sure to pause briefly between requests (Thanks to Nik Davis for this tip!).

```
[34]: def make_app_data_list(start, stop, parse_func, applist, pause, output=True):
          """Return list of app data generated from parser.

          parse_func : function to handle request
          """
          app_data = []

          # iterate through each row of app_list between start and stop
          for index, appid in applist[start:stop].iteritems():
              if output==True: #print details, useful for short runs but should be␣
       ↪supressed for full production
                  print('Index: {}; Appid: {}'.format(index, appid))

              # retrive app data for a row, handled by supplied parser, and append to␣
       ↪list
              data = parse_func(appid)
              app_data.append(data)

              time.sleep(pause) # prevent overloading API with requests

          return app_data
```

First, check on a small subset of apps to make sure the above function works as intended:

```
[191]: app_data = make_app_data_list(0,2,parse_steam_request, applist, 5)
```

```
Index: 0; Appid: 1034230
Index: 1; Appid: 603750
```

Next, create and initialize a CSV file to which we will iteratively write our data.

```
[2]: def init_csv_file(path, filename, columns):
         '''Create file and write header row'''
         file = os.path.join(path, filename)
         with open(file, 'w', newline='') as f:
             writer = csv.DictWriter(f, fieldnames=columns)
             writer.writeheader()
```

```
[333]: datafile = 'steam_data.csv'
       data_path = './data/'
       init_csv_file(data_path, datafile, cols_to_keep)
```

Finally, collect and write the entire dataset in a batch process, saving to file every 100 (or whatever you chose) lines. This process takes quite a while, as there's over 22,000 apps to be processed), so as an example I've reduced the `"stop"` parameter to only 500. Change this to "-1" or use the default to rerun the full dataset.

[45]:
```
def get_and_write_data_batches(path, filename, columns, applist, parse_func,␣
↪start=0, stop=-1, batchsize=100, pause=2):
    '''Run the data collection process in batches by making API requests and␣
↪writing to file every batch'''

    if stop==-1: #process entire app list
        stop=len(applist)+1

    batches = np.arange(start, stop, batchsize) #array to be used for "start"␣
↪and "stop" values for batch requests
    batches = np.append(batches, stop)
    start_time = time.time()

    for i in range(len(batches)-1):
        start_time_batch = time.time()

        batch_start = batches[i]
        batch_stop = batches[i+1]
        print('Batch #: {}'.format(i+1))

        #get data for batch
        print('Collecting data from API...')
        app_data = make_app_data_list(batch_start, batch_stop, parse_func,␣
↪applist, pause, output=False)

        #write batch to file
        print('Writing data to file...')
        file = os.path.join(path, filename)
        with open(file, 'a', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=columns, extrasaction='ignore')
            time.sleep(1)
            writer.writerows(app_data)

        #get time info and estimate remaining time
        total_time = time.time()-start_time
        batch_time = time.time()-start_time_batch
        estimated_time_remaining = (stop-start)/batchsize*batch_time-total_time
        print('Elapsed time: {} seconds'.format(round(total_time,3)))
        print('Estimated time remaining: {} minutes'.
↪format(round(estimated_time_remaining/60,3)))
        print(' ')
    print('Done!')
```

[334]:
```
get_and_write_data_batches(path=data_path, filename=datafile,␣
↪columns=cols_to_keep, applist=applist, parse_func = parse_steam_request,␣
↪start=0, stop=200, batchsize=100, pause=2)
```

```
Batch #: 1
Collecting data from API...
Writing data to file...
Elapsed time: 231.513 seconds
Estimated time remaining: 3.859 minutes

Batch #: 2
Collecting data from API...
Writing data to file...
Elapsed time: 462.392 seconds
Estimated time remaining: -0.011 minutes

Done!
```

2.4 1.4 Using SteamSpy to Get Game Popularity Data

The final step to our data collection process is to use the SteamSpy API to obtain game sales/rating information that isn't available directly from Steam. This API is much simpler to use than the Steam Storefront API, and has really straightforward documentation on its website (however, sometimes it goes down and the information we need isn't available...).

Luckily, we can reuse most of the functions we created for the Steam API data collection - we'll just feed a different parser into the functions to collect and write the data. The function to make a API request to SteamSpy is below. Like we did previously, before we go collecting 22,000+ data points, we want to take a look at an arbitrary AppID and see what kind of information SteamSpy will give us.

[38]:
```
def parse_steamspy_request(appid):
    '''Parser to handle SteamSpy API data.'''

    url = "https://steamspy.com/api.php"
    parameters = {"request": "appdetails", "appid": appid}

    json_data = get_api_request(url, parameters)
    return json_data
```

[39]:
```
parse_steamspy_request(603750)
```

[39]:
```
{'appid': 603750,
 'name': '- Arcane Raise -',
 'developer': 'Arcane Raise',
 'publisher': 'WAX Publishing',
 'score_rank': '',
 'positive': 54,
 'negative': 91,
 'userscore': 0,
 'owners': '20,000 .. 50,000',
 'average_forever': 225,
 'average_2weeks': 0,
 'median_forever': 233,
 'median_2weeks': 0,
 'price': '299',
 'initialprice': '299',
 'discount': '0',
 'languages': 'English',
 'genre': 'Adventure, Casual, Indie, RPG, Strategy',
 'ccu': 0,
 'tags': {'Adventure': 66,
  'RPG': 48,
  'Strategy': 40,
  'Casual': 34,
  'RPGMaker': 30,
  'JRPG': 26,
  'Fantasy': 24,
  'Indie': 24,
  'Story Rich': 19,
  'Rogue-like': 5}}
```

We can quickly see that a lot of the categories on SteamSpy are duplicates of what we get from Steam, such as `name`, `developer/publisher`, `genre`, `languages`, etc, but that we also have some intriguing categories such as `positive`, `negative`, and `owners`. In the example above, the `score_rank` and `userscore` categories are blank, but maybe those will be useful as well for some AppIDs. The `average_forever` and similar categories may be of interest, as these give the average (or median) playtime (in minutes) and is a very different metric from just how many people bought a game and never touched it after the first day. For now, we'll go ahead and collect all of this data, and later decide if we want to remove anything redundant or unhelpful when we clean the data and merge the Steam and SteamSpy datasets.

We'll start by initializing a data file with the columns we want, much the same as we did for the Steam API data:

```
[44]: cols_steamspy = ['appid', 'name', 'developer', 'publisher', 'score_rank',␣
       ↪'positive', 'negative', 'userscore', 'owners',
                        'average_forever', 'average_2weeks', 'median_forever',␣
       ↪'median_2weeks', 'price', 'initialprice', 'discount',
                        'languages', 'genre', 'ccu', 'tags']

      datafile_steamspy = 'steamspy_data.csv'
      data_path_steamspy = './data/'
      init_csv_file(data_path_steamspy, datafile_steamspy, cols_steamspy)
```

Then, we'll use the same data collection function as before, supplying this time the parser function to the SteamSpy API as well as the appropriate columns. The SteamSpy API has a higher polling rate than the Steam API, so we can also safely reduce the valueof `pause` to speed up the process.

[46]:
```
get_and_write_data_batches(path=data_path_steamspy, filename=datafile_steamspy,␣
↪columns=cols_steamspy, applist=applist, parse_func = parse_steamspy_request,␣
↪start=0, stop=100, batchsize=50, pause=0.5)
```

```
Batch #: 1
Collecting data from API...
Writing data to file...
Elapsed time: 44.18 seconds
Estimated time remaining: 0.736 minutes

Batch #: 2
Collecting data from API...
Writing data to file...
Elapsed time: 84.463 seconds
Estimated time remaining: -0.065 minutes

Done!
```

2.4.1 Concluding Thoughts

And that's it for Part 1! So far, we have two datasets, saved as CSV files, that contain different information pertaining to each Indie game. We can look at both the Steam data and SteamSpy data and see that it's really quite a mess, and we have a lot to do in the next section on data cleaning! In addition to cleaning, in Part 2 we will decide which categories we want to keep in our main dataset, and combine the information from Steam and SteamSpy.

[47]:
```
steam_data = pd.read_csv('./data/steam_data.csv')
steamspy_data = pd.read_csv('./data/steamspy_data.csv')
```

[48]:
```
steam_data.head(2)
```

[48]:
```
   type               name  steam_appid  required_age is_free  \
0  game                ***      1034230           0.0   False
1  game  - Arcane Raise -        603750           0.0   False

                                      about_the_game  \
0  *** is a small game.<br />\r\nIt is easy to pl...
1  <img src="https://steamcdn-a.akamaihd.net/stea...

                                   short_description  \
0  *** is a small game. It is easy to play,hope y...
1  Arcane Raise is a role-playing video game fran...

                                 supported_languages  \
0  English<strong>*</strong>, French<strong>*</st...
1  English<strong>*</strong><br><strong>*</strong...
```

```
                                        header_image website  \
0  https://steamcdn-a.akamaihd.net/steam/apps/103...     NaN
1  https://steamcdn-a.akamaihd.net/steam/apps/603...     NaN

              developers               publishers  \
0  ['Kenshin Game Studio']  ['Kenshin Game Studio']
1         ['Arcane Raise']       ['WAX Publishing']

                                      price_overview  packages  \
0  {'currency': 'USD', 'initial': 99, 'final': 99...  [344162]
1  {'currency': 'USD', 'initial': 299, 'final': 2...  [158866]

                                      package_groups  \
0  [{'name': 'default', 'title': 'Buy ***', 'desc...
1  [{'name': 'default', 'title': 'Buy - Arcane Ra...

                                        platforms  \
0  {'windows': True, 'mac': False, 'linux': False}
1    {'windows': True, 'mac': True, 'linux': True}

                                          categories  \
0        [{'id': 2, 'description': 'Single-player'}]
1  [{'id': 2, 'description': 'Single-player'}, {'...

                                              genres  \
0  [{'id': '4', 'description': 'Casual'}, {'id': ...
1  [{'id': '25', 'description': 'Adventure'}, {'i...

                                         screenshots  \
0  [{'id': 0, 'path_thumbnail': 'https://steamcdn...
1  [{'id': 0, 'path_thumbnail': 'https://steamcdn...

                                              movies  \
0  [{'id': 256744684, 'name': 'V0.3', 'thumbnail'...
1  [{'id': 256680620, 'name': 'Arcane Raise - Lau...

                                        achievements  \
0                                                NaN
1  {'total': 37, 'highlighted': [{'name': 'Letter...

                                     release_date  \
0  {'coming_soon': False, 'date': 'Mar 17, 2019'}
1  {'coming_soon': False, 'date': 'Mar 10, 2017'}

                                          background  \
0  https://steamcdn-a.akamaihd.net/steam/apps/103...
1  https://steamcdn-a.akamaihd.net/steam/apps/603...
```

```
          content_descriptors
0  {'ids': [], 'notes': None}
1  {'ids': [], 'notes': None}
```

[49]:
```
steamspy_data.head(2)
```

[49]:
```
     appid               name           developer           publisher  \
0  1034230                ***  Kenshin Game Studio  Kenshin Game Studio
1   603750   - Arcane Raise -          Arcane Raise      WAX Publishing

   score_rank  positive  negative  userscore            owners  \
0         NaN         3         2          0       0 .. 20,000
1         NaN        54        91          0  20,000 .. 50,000

   average_forever  average_2weeks  median_forever  median_2weeks  price  \
0                0               0               0              0     99
1              225               0             233              0    299

   initialprice  discount                                      languages  \
0            99         0  English, French, Italian, German, Spanish - Sp...
1           299         0                                        English

                                  genre  ccu  \
0                         Casual, Indie    0
1  Adventure, Casual, Indie, RPG, Strategy    0

                                          tags
0         {'Indie': 21, 'Casual': 21, 'Puzzle': 12}
1  {'Adventure': 66, 'RPG': 48, 'Strategy': 40, '...
```