ROBLOX ETL Project



# Project Proposal

**Business Opportunity**: Roblox is an interactive gaming platform that allows developers to create games millions of users (predominantly children) can play. With a vast and ever growing array of games, first time users might find it a little daunting finding the right game to play and how to play it.

**Proposal**: We will create a service that will allow users to view the most popular Roblox game and display the top youtube tutorials on how to play the Roblox game. We will accomplish this by scraping information from the Roblox site for the most engaging game, query the Youtube API for the top 5 videos of the search for each game, and display each game with the top 5 tutorial videos on our service.

**Benefit**: By creating a service that consolidates the most popular Roblox games and some of the best tutorial videos of those games into one easy to use location, we will create a resource that new and existing gamers will turn to when looking for their next Roblox game to play and have the resources to play the game well.

# Extract

## Roblox site scrape

Roblox has multiple categories for that are available for users of the Roblox site to select from with approximately 60 games in each category:

1. Most Engaging
2. Recommended for You
3. Sponsored
4. Up and Coming
5. Popular
6. Top Rated
7. Learn & Explore
8. Top Grossing (Top Earning)
9. (ETC.)

We will scrape information from only one category to limit the scope of our project. After discussing which category we want to examine, we focused on the Most Engaging category. Games from the Most Engaging category are the games that will likely draw in the most players, be most enjoyable and lead to the largest usage of our service to learn the fun games.

The Roblox site uses Javascript. When viewing the site, more content is available than is available immediately using a simple scrape. As a result, it was necessary to enable the Web Driver for the browser to allow faster loading speeds to facilitate scraping.

We pulled the URL for the Most Engaging category as our baseline URL. We created a beautiful soup item with the results of our web scrape. Once prettified, we identified the “game-card game-tile” class as the class that would provide us with the key information about the top games within the category.

Once we have the results of our scrape in the beautiful soup object, we identified the following key information:

* Game title
* Number of users
* Overall rating
* Game thumbnail URL (for later use with website)
* Game link (for later use with website)

When thinking about the end user, we gathered the above information to best inform the gamer using our service to determine which game they want to play. Game name, the image of the game, and the link to said game all are key to let the customer know what game they are looking at. On top of that information, gamers want to know how other gamers enjoyed their experience so including an overall rating will give our users a sense of how enjoyable the new game will be. Lastly, providing the number of users for each game helps balance out the overall rating of the game. A game might be super popular with the users playing, but not many people playing could inform our users that the game might not be as great as rated.

The information we gathered from the object isn’t always complete so we filtered our results further to only include the games that had all of the following:

* Game title
* Number of users
* Overall rating
* A link to the game

## Youtube API query



Once the results from the Roblox webscrape are successfully stored in our dataframe, we leveraged the game names in our results to query the Youtube API for tutorial videos on how to play those games.

We determined that the top 5 results provided by the API would be sufficient for our service as too many videos would be overwhelming to our users. Youtube as a website, must have a large and complex algorithm to determine the top results of any search you enter so we elected to trust that algorithm for establishing which results we pulled from our query. The rough idea was to query the API with the following baseline key word search: f’learn roblox {game\_name}’, with the game\_name being the result returned from the Roblox web scrape and “learn roblox” being the beginning of our Youtube search.

We began our query with a stud game name and key word search. Adopt Me is an extremely popular Roblox game so we used it as our baseline game. By beginning our query with a search with our baseline game and key word search, we established that our connection with the Youtube API was successful. We jsonified the results of our stud query to establish what information our query returns and determine what portions are useful to us.

As Youtube users ourselves, we know some of the metrics we would find most useful when deciding what video to select. We gathered the following information for each video result in our query:

* Video title
* Video duration (publication time)
* A thumbnail of the video for our website
* A URL link to the video for our website
* Video statistics
  + Likes Count
  + Views Count
  + Comments Count

Providing the video titles and durations, users on our website will be able to know at a high level what the subject of the video is and how long those videos are. Some users might appreciate a more in-depth and detailed tutorial so they will look for videos with a longer duration. Others will appreciate a quick run through of the game before they learn by playing so a quick tutorial suits them better.

Once we had the URL links to the returned videos, we further queried the API to find the statistics of said videos to better inform our end users on the quality of the videos (vamp).

Once we established that the stud pulled all the information needed for the our test game, we updated our query to pull the information needed of all games provided by the Roblox webscrape. We created a for loop to query each game name and return the information into its own dataframe so we could load this, along with our Roblox dataframe created in the web scrape to send results to our service.

# Transform

## Transforming Roblox site scrape results

With the finalized and filtered list of game results, we created a pandas dataframe to further manipulate the results. The results of the beautiful soup object kicked out the overall rating of each game as a percent liked vs disliked. The Percentage Approval rating was reformatted to remove the “%” and enable sorting by this figure in the dataframe. The text was converted to a float.

Similarly, the Number of Users was reported in numbers up to 1,000, then with a “K.” In order to ensure that the numbers were reflected appropriately and enable sorting by these figures, the strings were transformed to the actual numbers and converted to a float. This resulted in the ability to sort descending by this statistic. The higher Number of Users reflects the relative popularity of the games in the “Most Engaging” category.

Finally, we formatted the game titles to make our Youtube API query more successful. We stripped the game titles of their spaces, removed any special characters and emojis, and reformatted them to only be lower-cased. We found a few duplicate titles in our search so reformatting the game titles ensured any duplicate results were easy to identify and delete.

With our finalized results transformed to our liking, we saved the dataframe to combine later with our results from the Youtube API scrape.

# Load

## PGAdmin Database

Before loading the information we gathered about the Roblox games we want to display and the tutorial videos on said games, we weighed our options on what type of database we will use for loading our results.

Our initial thoughts were to go with either a MongoDB database or a PGAdmin database. Each comes with it’s own set of advantages and difficulties.

* PGAdmin databases
  + Pros:
    - Tend to be easier to create with solid pre-existing databases
    - More user-friendly to the end users
    - Able to easily identify the relationship between multiple databases
  + Cons:
    - People that want to connect with our service will need to establish a PGAdmin connection to our dataframes
    - More complex establishing the databases with raw data
* MongoDB
  + Pros:
    - More user-friendly for us when creating the databases
    - More versatile in terms of the data that can be entered into a database
  + Cons:
    - Users of our service need to be exact in their searches. Minor errors return no results
    - Identifying the relationship between multiple databases is more difficult

After deliberation, we determined a PGAdmin database would best suit our data. Our Youtube and Roblox databases already formatted our data into tabular format, the preferred formatting of SQL databases. This means that we would need to manipulate our existing databases less and we would be able to quickly establish the relationships between our 2 databases.

The following code was used to create the table for Roblox game in our PGAdmin database:

CREATE TABLE public.games

(

game\_id integer NOT NULL,

game\_title character varying COLLATE pg\_catalog."default",

user\_count double precision,

positive\_ratings double precision,

game\_url character varying COLLATE pg\_catalog."default",

game\_image\_url character varying COLLATE pg\_catalog."default",

CONSTRAINT games\_pkey PRIMARY KEY (game\_id)

)

To eliminate any discrepancies between the game title and the video results, we identified the game\_id as the primary key from our Roblox database and tied that key to the Youtube database so the Roblox game titles drove the results returned for the API query.

With our games table established, we created the videos table to support our Youtube API results with game\_id serving as the foreign key tying our 2 tables together.

CREATE TABLE public.videos

(

game\_id integer NOT NULL,

game\_title character varying COLLATE pg\_catalog."default",

video\_name character varying COLLATE pg\_catalog."default",

yt\_views integer,

yt\_likes integer,

yt\_comments integer,

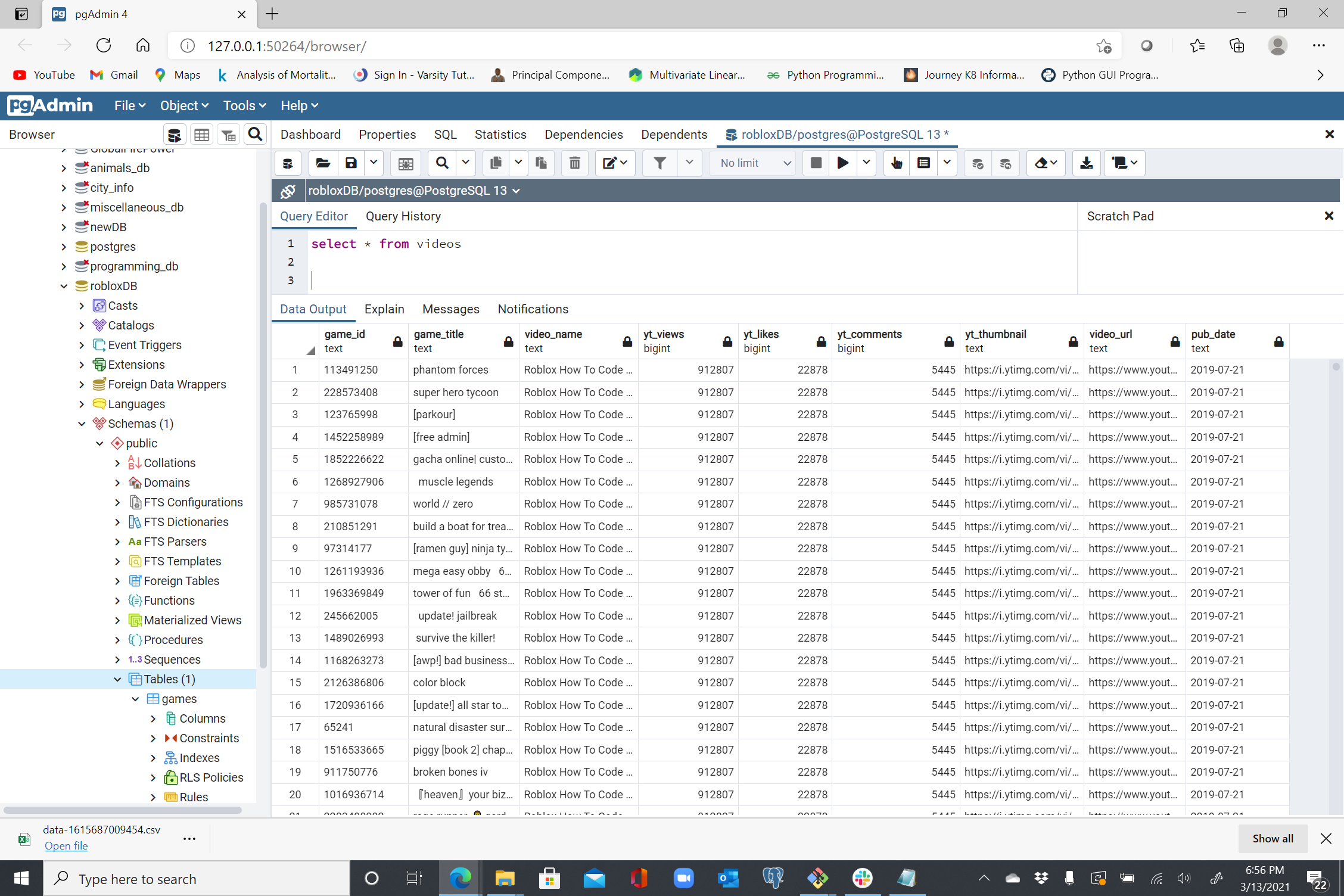
video\_url character varying COLLATE pg\_catalog."default",

pub\_date date,

CONSTRAINT videos\_pkey PRIMARY KEY (game\_id)

)

With our 2 tables established, queries of our database returned all the key information on the Roblox game and the youtube videos associated with that game.

****

# Lessons Learned & Next Steps

## Lessons Learned

Through this project, we learned a lot about gathering information from multiple web sources and how to best display the results of our searches. Although we were able to successfully scrape the web for our data, transform the data to better suit our needs, and display those results in a unique service, we ran into issues along the way that we would have addressed differently if we had a do-over.

1. The timing of our Beautiful Soup object was being created too quickly so when we tried to connect our database to the website, no results were rendered. Need to find a way to delay the soup object.
   * When we used the time.sleep function to create the soup object, we were able to successfully stop our soup object from rendering before the full page was loaded.
2. One potential issue of our setup is that each hit to the API could return different results.
   * Although it might be nice for returning users to see different video results each visit, it could also lead to the removal of the tutorial video they enjoyed best, stopping them from coming back to our site.
3. Despite successfully establishing the relationship between our 2 databases easily, we didn’t create a schema for the database relationships.
   * This didn’t impact us since the relationship between the 2 databases was relatively straightforward, creating the databases without a schema could lead to an incorrect relationship between them.
     + A more complex relationship needs to be mapped out in order to properly set up the databases in the future.

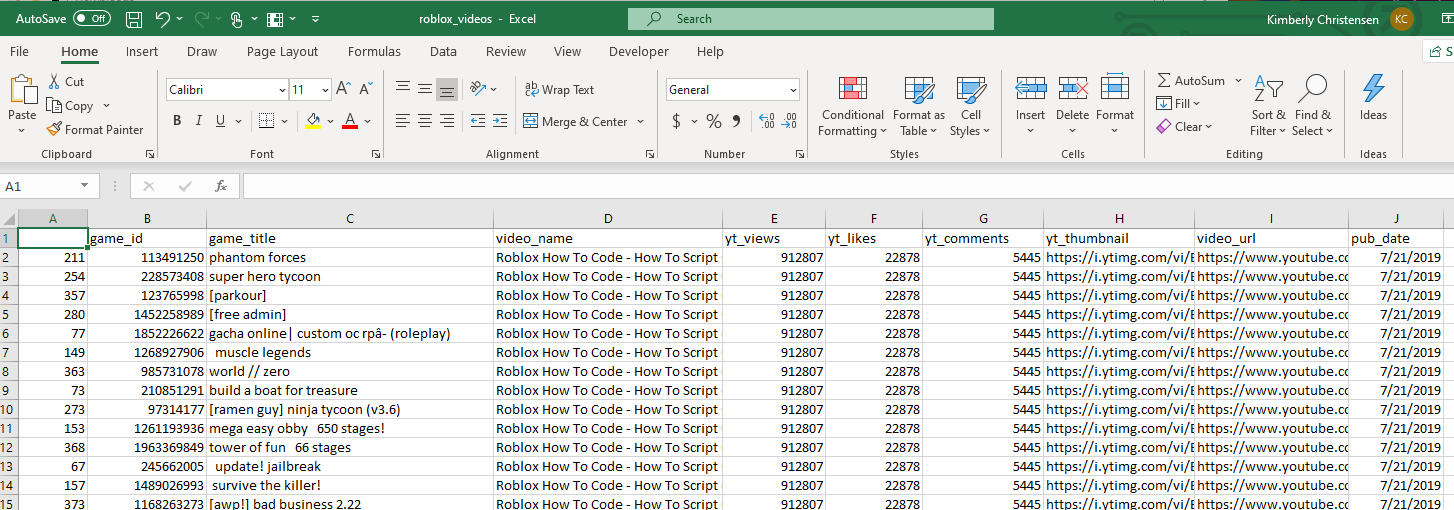
## Next Steps

Although we accomplish all objectives of this project, we have larger ambitions that someone picking up where we left off could reach for.

1. Our service could be put into an HTML that would allow for a more user-friendly experience. This would make our service a powerful resource for all gamers.
   1. This option could also expand the results we pull from the Youtube API to include more than just the videos and to include more metrics for the games from the Roblox web scrape.
2. We used a static keyword search for our Youtube API query with the Roblox web scrape providing the game titles. We could allow the end users to provide us with the key words to leverage in our query to better tailor our results for their needs.
3. We elected to pull games from only one category for simplicity sake. To enhance our service more, we could expand our scrape to pull games from multiple categories.
   1. This would increase the value of our service for gamers (identifying up and coming games, games that aren’t as popular, etc.) as well as increase the value of the service to game developers (showing the top grossing games to emulate or the most engaging games to aspects from when creating new games).

# References

Games CSV



Youtube video CSV

