

Silviu Popovici

Computer Science Capstone Project

9/20/2019

Web-Based Computer Game Recommendation System

with Analytics Dashboard

Table of Contents

- Letter of Transmittal – Page 2
- Project Proposal – Page 4
- Technical Executive Summary – Page 8
- Post-implementation Report – Page 18
- Installation Guide– Page 28
- User Guide – Page 29
- Summary of Learning Experience – Page 35
- Bibliography – Page 37

Letter of Transmittal

Silviu Popovici

Software Engineer

AwesomeGames

Sept. 13, 2019

Mr. John Smith

CEO

AwesomeGames

1234 Great St Suite 101 Phoenix, AZ

Subject: Web Recommender System with Analytics

Dear Mr.Smith, I am writing in reference to the need to increase game sales on our online platform and also to figure out what a good release date and price point would be if we were to release our own game on our platform. I recommend building an all new web application that allows our users to search for and browse games and which combines an intuitive user interface with a recommendation system that will suggest games to our users that are similar to the one they're viewing.

The system will also contain an analytics tab for managers which will help them monitor the users base's satisfaction with the recommender system and also allow them to use powerful algorithms to analyze our game catalog data and make inferences that would help our company decide if we should release our own game, and what a good release date and price would be.

A system that can recommend other similar games would help increase our sales greatly by putting relevant game suggestions right in front of our users as they browse our game catalog, and that will be the main objective of this project. The secondary objective will be to provide managers with analytics that will help them make good decisions in regards to a potential game release, which should greatly increase our profit should we choose to release one.

The funding requirement for this project will be around \$21,000, which represents one developer (myself) working for three months at around \$5000 monthly salary, and a two man QA team testing for one month in the later stages of development at \$3000 each for the month. There will also be the additional cost of hosting the web app once we are ready to release it to the public, which will come to around \$300-500 per month.

I've worked with the relevant technologies and programming languages for around three years and have built similar apps before, and I have a good grasp on what it will take to complete the project on time. Feel free to contact me with any questions.

Sincerely,

Silviu Popovici

Project Proposal

Problem Summary

The issue our company is currently facing is that our web based game marketplace is old and doesn't work well. It doesn't track user data and it also doesn't have any way to suggest games to users which makes it difficult for them to browse around for new things they might want. It also doesn't take advantage of modern security features. As a result, our sales have been in decline this past quarter.

On top of that, our company has been considering potentially launching our own game(s) on our platform, so that we will no longer be merely a distributor but also a game development company. However, we don't currently have any good metrics by which to judge how this would play out. We need some good analytics tools to help us decide what price point we should be targeting and what release date, among other things. The solution is clear- we need a new web based system that can recommend games to users, is capable of performing analytics on our game catalog data, and has an intuitive user interface all while utilizing modern security features.

Application Benefits

The main benefit of a web based recommendation system will be that we will increase the visibility of our games by taking the game the user is currently viewing, and giving them ten recommendations of similar games they might like. This will lead to users discovering games that they didn't know about but are quite likely to be interested in. It will also increase user engagement as users will feel at home as they smoothly browse from game to game, getting a new set of recommendations for each game. Our sales will increase significantly, as will our brand loyalty since users will be much more likely to use the site again if they have a good experience.

The analytics portion of the system will have the benefit of giving decision makers a better idea of how we should go about developing our own game for the platform. It will do so by providing graphs and charts that show relationships in the data, such as the relationship between game release date and the average playtime that users

have played that game for every game in our catalog. These analytic tools will make it much more likely that the game development venture will be profitable.

Application Description

The application will be a web based system that will consist of three different main parts. First, a console-based data parsing and cleaning utility that will be run offline and will take the raw games data and filter and combine various parts of it into one file of cleaned data ready to be used by the recommender system.

Second, there will be a recommendation system that is also run offline to generate the top ten recommendations for each game and store them in a file. It will use a custom item to item similarity algorithm to calculate the similarity of each game to every other game.

Third, a web application that takes that recommendation data and serves it to the user in a user friendly web interface, among other typical web app functionalities such as the ability to search the games catalog and a user authentication system allowing users to log in. This will be divided into a back end and a front end.

The back end of the web app will be written in Python using the Flask framework, and the front end will be done in Javascript, HTML and CSS with the Bootstrap library for easy styling. The recommendation system itself will be written in Python, as will the console based data utility. Data used by the web app will be stored in SQL format using a SQLite database.

Data Description

The data that will be used is a catalog of information about thousands of different computer games in .csv format. The data includes categories such as game name, developer, publisher, price, positive and negative ratings, and average playtime in minutes for each game. The data comes in three different files- one for general data such as name, developer, and price. One for game description data, and one for game media data such as cover image and screenshots. The data parsing utility will parse this data and combine data from all three files into one while filtering out unusable data- such as games without ratings or without English language support.

In addition, there will also be user data that is generated by the app and stored in an SQL database. This data will represent user accounts that are created by users signing

up for the new system. It will also include a table for user opinions of the recommendation system which will record the user, the game, and the opinion. This will be used by managers to gauge the quality and accuracy of the recommender system according to the users themselves.

Objectives and Hypothesis

There are two main objectives for this new system: a recommendation system that can recommend games to users, and an analytics tool that enables managers to study the game catalog data. My hypothesis is that if a web based recommendation system is built, sales of games will greatly increase, and users will be happier and more engaged with our platform. My second hypothesis is that if a few analytics tools are made to help managers study the product data, they will be better able to make a decision as to how to price and release a potential in-house developed game.

Methodology

The methodology that would be best for this project is Agile. Agile is best because it breaks things down into smaller parts and is good at adapting to change. Each major portion of the project (the data utility, recommender system, and front end and back end of the web app) will be broken into smaller parts and completed within a two week sprint, and my code will regularly be reviewed by an IT manager.

Funding Requirements

The funding requirements for this project are simply the salary of one developer (myself) over a three month period, which is around \$5000 per month, so \$15,000 total, and the salary of two quality assurance team members, at \$3000 a month each, for one month, which comes to \$6,000 total, so \$21,000 total human resource cost. QA is only needed for one month because they only need to start testing in the later stages of the app development. Testing in earlier stages will be done by the developer. There will also be some hosting fees for Heroku once the app is ready to be released which will come to around \$300-500 per month depending on traffic to the site.

Since all programming languages, IDEs and other technologies to be used in this project are free, there are no extra costs. One modern desktop computer with internet access is all that is needed for development, and our company has plenty for use already so there are no extra costs there either.

Stakeholders Impact

There are three main stakeholders that stand to benefit from this project: our company, AwesomeGames, all of our investors, and our customers. Our company and its investors will benefit greatly from the increased sales provided by a better online system that provides product recommendations. Our customers will benefit by having a more enjoyable and engaging experience with our platform. They will have an easier time navigating through our games catalog and purchasing the games they want. Our company and investors will also benefit by having higher brand loyalty due to the customers' happiness.

Data Precautions

The game catalog data does not contain any sensitive information about people, only information about games such as game name, developer, price, etc. However, the Users database that will be created along with the application as part of the user authentication system will need to store usernames and passwords. These passwords need to be encrypted before they are stored and this will be done using an algorithm called 'bcrypt' which is an industry standard. Also, the system will need to be able to process payments, which would normally require careful protections of payment information according to the PCI-DSS standard, However, we will not be storing payment information locally. All of our transactions will be handled by a payment API called Stripe and our users' payment data will not be exposed.

Developer Expertise

I, Silviu Popovici, will be the primary developer of this system. I have been studying computer science and programming for over four years now, and have also built many projects for fun on the side, including a full-stack blogging web app. I've worked with the primary programming language for this system, Python, for a few years now. I have a good understanding of web systems, both front and back end, as well as SQL databases, and I am fully qualified to build this new system from the ground up.

Technical Executive Summary

Problem Statement

The main problem our company is facing is that our sales are low due to the inefficiency of our current online computer game sales platform. We currently don't have a way of recommending new games to customers, which makes it quite difficult to expose them to new games and difficult to browse our catalog in general. The second problem we have is that our managers and other decision makers don't have the analytical tools necessary to make data-driven business decisions.

All of the games on our online distribution platform are made by other developers that we have made distribution deals with. But lately some executives have expressed interest in developing a game ourselves and adding it our platform, but they don't have the ability to analyze when and for what price a game should be released if we do decide to do it.

On top of that, our current system has no real metric by which to judge its accuracy and performance. We need an all new web based system that can recommend games to customers, allow customers to search for games and make accounts to log in and out with, and that has analytical tools built in that managers can access to help with making some business decisions about the data. All while adhering to modern security standards. The project summarized in this document will meet all these points and more.

Customer Summary

There will be two types of users for this new system- customers and managers. Manager accounts will be special accounts that are created internally only for use by high ranking members of our company, and will have access to the analytic tools the system provides in addition to all of the functionality that customer accounts have. Our customers are the people who purchase games on our online platform, and customers will be able to create their own accounts to log in with.

Customer accounts will have access to the basic functions of the website, such as searching for and browsing games, which includes having games recommended to them

based on the one they are viewing, and adding a game to their cart. The environment for the app will be the user's local web browser that sends and receives calls from our server, and the interface will be intuitive and responsive so that anyone can browse it from any device and screen size without any special technical skills. The intuitiveness of the UI and the ability to browse recommendations will make our customers happy and definitely increase our sales, and the higher-ups will be very happy with the analytic tools the system will provide.

Existing System Analysis

Our existing web platform consists of a very basic and old Java based system that has no recommender system in place and consists only of being able to browse a game list and add to cart. The UI is outdated and users have reported difficulty navigating the website. Payments are handled through a third party API using Stripe just like our new system will, but this old system doesn't record any user data on our end beyond the number of thumbs-up or thumbs-down given to a game and the average playtime for that game, and it does not have the ability to log in and out.

We will replace this old system with a new one that will have a recommender system, a much better UI and a proper user account system, and that will be written in Python using the Flask framework, because Python is faster to develop with and has more concise, readable code than Java. Python also has better support for machine learning and data analysis algorithms which we will need for our analytics functionality. It will still use Stripe to process transactions just like the old system. We should be able to simply link our Stripe account to the new system.

Data Summary

The game data that we have comes from our large catalog of games that we sell and is formatted in comma separated values (.csv) format. However, this data will be cleaned to only include the games that have received ratings, and any games with missing or faulty data will be dropped, ensuring that our system is not cluttered by games we have been unable to sell or by incomplete data, and also that every single game in the cleaned data set will be usable by the recommender system. Every game that is missing data in a column necessary for the recommender system to perform its similarity calculations will be dropped. All of this data cleaning will be done by a console based data utility built as part of the new system.

The original data is spread across three different files. One file is the main games data file and contains information such as game name, developer, publisher, price, etc. Another file contains the game description data, with both long and short descriptions. And the third contains game media data, such as screenshots and cover image. From these three files, the data cleaning utility will put together one cleaned file ready to be used by the recommendation system and web app.

From the main games file, it will take the Name, Release Date, Developer, Publisher, Platform, Genre, Tags, Positive Ratings, Negative Ratings, Average Playtime, and Price columns. From the descriptions file it will take the short description column, as it makes for faster similarity calculations and the long descriptions are simply too large and not needed, and from the media file it will take the cover image link. It will combine all of these columns into one file, while filtering out games that either do not have English language support, have an average playtime of zero, or have zero positive ratings, as we no longer want these games to be part of our sales catalog. The games that make it into the cleaned data set will be given new Game IDs based on the order they were parsed, as the old IDs are no longer needed. All of this cleaned data will also be in .csv format for ease of use by the recommender system.

Once the cleaned games data file is created, the recommender system will need to be run using a console based recommender utility built as part of the project. This recommender system will take the cleaned data, calculate the similarity score between each game and every other game, and generate a large similarity matrix in .csv format. It will then sort each row in the matrix by similarity score in descending order, and the top ten of these will be exported to a file that will hold the top ten recommendations for each game, also in .csv format. The large similarity matrix can then be optionally deleted as it is no longer needed. The only time this process will need to be done over again is when the game catalog changes.

Finally, the cleaned games data and top ten recommendations data for each game will need to be imported into the web app and turned into SQL form. This data wrangling will be handled by the Flask backend itself using SQLite3, which is a lightweight SQL database tool that allows us to use a highly portable database file embedded into our web app. It also allows us to create a database in RAM upon server startup, which is a much faster solution than validating/populating a SQL database stored in a file. Since the games data is a csv format which is based on a game catalog

that could change regularly, it is faster to simply generate the SQL database in memory directly from the CSV.

In addition to the games data, there will also be user data which will be stored in a sqlite database file. We don't have any user specific data to begin with because our old system did not adequately record such data and did not have a user account system. We will need to create a Users database in SQL that has a Users table that stores data such as username, password (encrypted), and account type. There will also need to be a table for user opinions of the recommendation system that will store userId, gameId, and opinion which can include the values 'Great', 'Okay' and 'Bad'. This will help us gauge the accuracy of our recommender system by the best real world metric- the real opinions of users.

Project Methodology

The development methodology used for this project will be Agile. Each of the main components of the system will be created in a two week sprint, and each will be tested at the unit level before being combined with the other parts. Testing will be mainly black/gray box manual testing. In the first two months or so of development, testing will be done by the developer and in the last month or so it will be done by the developer and by a two person software quality assurance team.

The first sprint will be for creating the data cleaning utility. It will be tested by ensuring that the output of combining and cleaning the three data files contains the right data columns and rows, and that there were no errors in copying over the data. The second sprint will be for the recommender utility. This utility is responsible for running the recommender system and producing a similarity matrix, and it will be tested by ensuring that the data is read in properly, and that the output appears correct- namely that the similarity percentages are floating point numbers between 0 and 100.

The third sprint will be for creating the back-end of the web system. It will involve creating all of the RESTful routes for the site and the API for retrieving games data from the database. The two SQL databases that need to be created will be considered part of the back-end sprint. The back-end and databases will be tested using a basic placeholder front end with minimal styling. Tests will include ensuring that the databases can be built and populated without errors, that the top ten recommendations can be sent properly to the front-end through a GET request, and that the user is

capable of searching for games by name using a search bar without errors. The back end will also include the user authentication functionality, which will include creating a special manager account

The fourth sprint will consist of creating and styling the front end. The styling will be done using the Bootstrap library along with custom CSS and HTML, and the front end will be rendered with the help of the Jinja templating engine built into Flask. The front end will be tested by ensuring that every button and navigation link works properly, and that everything is displaying properly, even when the screen size is changed.

Then there will be a fifth sprint that consists of creating the analytical tools of the system. These will include a k-means cluster analysis that will attempt to find patterns when graphing games' release dates to average time played, a scatter plot graphing price to number of positive ratings, and a pie chart that represents user opinions of the recommendation system. These tools will be tested by ensuring that the graphs display properly and that they show relevant data from the database.

Finally, there will be a sixth sprint that will focus on integrating all these parts while also performing integration testing along the way, ensuring that as each piece is added the whole application still works. Once everything is integrated, system and acceptance testing will be done across the entire web app.

This will include testing every navigation link and button, the search functionality, ensuring that the recommendations pop-up properly on a game's page and that the user can click through them, testing logging in and out, and that the analytics tab graphs work properly and will enable managers to make better business decisions. The last test will be a system-wide test of the entire application flow from start to finish, starting with only the app and its utilities and the original data.

The final system flow to be tested will go like this- open and run the console-based data utility. A cleaned data file should be output with appropriate data. Then run the recommender utility on the cleaned data- a similarity matrix in csv format should be output and then a list of top ten recommendations for each game also output based on the matrix.

Then, run the web application using the Python server that comes with Flask, open the web app in a browser and proceed to test the front end, UI, user authentication, searching, and analytics functionality as described earlier. Also check the

SQL database for anomalies. Once this final system check is completed without errors, the project will be considered complete, barring any potential last minute changes, and the only thing left to do will be to deploy it to a production environment.

Project Outcomes

There will be two main categories of deliverables as an outcome of this project- written project deliverables and software product deliverables. The project deliverables will include documentation such as test plans for each sprint, an installation guide and user manual, and a post-implementation report. The product deliverables will include the data cleaning utility, the recommender system utility, the back end of the web app, the front end of the web app, the analytics functionality built in to the app, and the SQL database and cleaned .csv data files ready for use with the app.

The test plans will be written documents that describe black box testing methods for each of the six Agile sprints for the project. They will include detailed tests for every aspect of each component of the app and have a column for indicating whether the test passed or failed. For example, some tests for the data cleaning utility might be, “Data files open in the utility without error”, and “Cleaned data file contains all required columns.”

The installation guide and user manual will be documents that inform a user on how to set up an environment in which to run the application, and how to actually use the application- from the console based utilities to the actual web app, all functionality will be covered. Finally, there will be a post implementation report summarizing what the project accomplished and how.

As far as software product deliverables, the first will be the data cleaning utility. This will be a console-based program that will be run from the command line. It will be capable of combining and cleaning the data from the three original source .csv files and will output one cleaned games data file.

The second product deliverable will be the recommender utility. This utility is also console based and will be capable of taking the cleaned data and calculating the similarity score of each game to every other game. It will do this using a custom item to item similarity algorithm that gives a similarity score from 0 to 100, using different weights for different data columns. For example, the 'game developer' attribute may have a weight of 10%, so if the developer of two games is the same then that's 10 points

added to similarity score. For more complex things like game name and game description, a variant of the Hamming Distance is used to find how many words are the same between the two strings, which is then converted to a percentage and multiplied by the weight given to that attribute. The recommender algorithm will do this for all games and output a similarity matrix in .csv format.

The recommender utility will also be capable of taking the similarity matrix, sorting each row by similarity score in descending order, and then extracting the top 10 recommendations for each game into a csv file ready to be imported by the web app. The recommender utility and the data cleaning utility will both be accessed by a single console based interface that combines the functionality of both, so that the user only has to run one utility to get all the functionality, and all of these utilities will be written in Python.

The next product deliverable will be the backend of the web app. This will be written in Python with the Flask framework and will also make use of multiple libraries, such as flask-bcrypt for encrypting passwords and flask-login for handling user authentication. The backend will define multiple RESTful routes by which to receive and send data to the front-end, including sending the top ten recommendations for a game, responding to a search query of a user searching for a game by name, and the ability to create accounts and log in. It will also help render templates to the front end using a templating engine called Jinja.

Another product deliverable will be the front end and UI of the web app. This will include a navigation bar at the top of each page with links to each part of the app as well as a search bar. This will be created with HTML5 and styled with Bootstrap and some custom CSS3. It will be responsive, meaning that it will look good on all screen sizes. Bootstrap will help greatly with this, providing among other things a collapsible navigation bar that collapses into a single drop down button on small screen sizes.

The SQL database needed for the app is another deliverable. It will consist of a Users database with two tables- one for storing user data and one for storing user opinions of the recommendation system. There is also another SQL database in the form of the cleaned games data being rendered from csv to SQL format upon server startup, but this will be rendered in RAM instead of a file using SQLite3, so it is more part of the back end than a separate deliverable.

The final product deliverable for this system will be the analytics functionality. This will consist of an 'Analytics' tab on the navigation bar that will only be accessible when logged in as a Manager type account. Clicking on this tab will take the user to an Analytics page, where multiple graphs rendered in real time will be displayed. They will include a pie chart representing user opinions of the recommendation system, a scatter plot with a k-means clustering analysis of Release Date versus Average Playtime, and a scatter plot plotting price versus positive ratings. These will help answer business questions such as “When is the best time to release a game?”, “What's a good price point for a game?”, and “How accurate is our recommender system according to our users?”

Implementation Plan

Once the system is complete it will need to be implemented in a production environment. The implementation strategy will involve deploying the entire application to Heroku using our company's Heroku account and running it from the cloud. Every software deliverable described earlier will be deployed to Heroku all at once, after the application has been thoroughly tested on the development server. This will take place as part of the final sprint after final system and acceptance testing is finished on the development server.

The application will also need to be tested again in production to make sure nothing has broken in the process of migrating. Regression testing will be done starting at the unit level by our QA team. Finally, the app will need to be linked up with the Stripe API using our company's Stripe account so that payment transactions can be processed. This Stripe integration will also need to be tested thoroughly by our QA team, and once it's complete the app will be ready for our customers to use.

Evaluation Plan

The main method of assessing the accuracy of the app will be the user opinions of the recommender system, because the most important metric by which to measure a recommender system is user engagement with the system in the real world. On each game's product page, below the ten recommendations there will be a section asking the user their opinion on the quality of the recommendations with three options – Great, Okay, and Bad. These will be stored in the Users database in the RecommenderOpinions table as numbers (0,1,2), and they will be visualized in a three-section pie chart at the

top of the analytics page in order to give managers an idea of how the recommender system is being received by users. The recommender system will be considered successful as long as less than one third of the user opinions are in the 'bad' category after 6 months of the app being released. If more than a third of the votes are 'bad' then we would have to take a serious look at adjusting the recommender algorithm.

As far as quality assurance and the evaluation of the app as a whole, there will be unit tests for each piece of the app, corresponding to each two week sprint. There will also be a logging functionality that logs errors and important information from the app to a log file, recording the timestamp, severity of the warning or error, and the error itself. There will also be system and acceptance tests for when all the pieces come together.

All of these unit and system tests must pass before the app can be moved into production, and once the app is in production a regression test must be run and all of these tests must pass again before it is officially released to the public.

Resources and Costs

The programming environment will include mainly free and open source tools, so there isn't any cost associated with them. The data and recommender utilities will be written in Python, as will the back end of the website (Flask) and the analytics tools. The front end of the website will be written with HTML5, CSS3 and Javascript. All of the programming will be done in an open source editor called Atom, and the development server will be the one that comes with Flask.

The costs of the environment will be minimal. All that is needed is a PC with an internet connection which our company has many of already. The app will not require a lot of memory or processing power to run, although the recommender utility may take some time to generate the similarity matrix, so a faster processor is always better. The development server can be run on just about any computer from the command line. However, the Heroku production server will have costs for hosting depending on how many concurrent users we have. Currently we have an estimated 100,000 customers, so the server costs for Heroku will probably be around \$300-500 per month.

As far as human resources, there is only one developer for the project who will be working 40 hours per week for 3 months at \$5000 per month salary, which comes to \$15,000 total. There will also be two QA team members working full time for one month

at \$3000 a month each. That's \$6000 total for QA, so \$21,000 human resources cost in total for the three month project.

Timeline and Milestones

Data Cleaning Utility development	Sprint 1 – 2 weeks
Recommender utility development	Sprint 2- 2 weeks
Web Back End Development	Sprint 3- 2 weeks
Web Front End Development	Sprint 4 - 2weeks
Analytics Tools Development	Sprint 5 – 2 weeks
Integration of all parts and extensive integration/system testing by QA team	Sprint 6 – 2 weeks
Migrate system to production and perform regression tests by devs and QA team	End of Sprint 6

Post-Implementation Report

Project Purpose

The main purpose of this project was to create an online system where users can browse and shop for computer games while having games recommended to them via a recommendation system in order to increase sales and customer engagement with our platform. The second purpose was to give managers and decision makers some analytical tools so they could use our data to make better business decision, especially pertaining to answering questions about a potential game release by our company.

Both of these goals were met, as the recommendation system works properly and recommends ten similar games to the user when they are viewing a game's page, for every game. And the analytics tab provides a bird's eye view of how the recommendation system is doing based on the best real world metric, the opinions of the users, and it does so through a live generated pie chart.

The analytics page also shows three different scatter plots- the first two are showing the relationship between release date and average playtime. The first chart shows the plain relationship, while the second is the result of k-means cluster analysis with ten clusters. The third chart shows the relationship between price and number of positive ratings, and helps answer the question of what a good price point would be for a game.

Datasets

The original data is in the 'recommender_app/data/original data' folder and is comprised of three files. The first is the games.csv file, which contains the main game data such as game name, developer, price, etc. The second is the game_description_data.csv file which contains the game descriptions, and the third is the games_media_data.csv file which contains media such as a link to the game's cover image.

In the Data folder there are also two other data files: the cleaned data in 'games_clean.csv' and the top ten recommendations for each game in 'topTenRecs.csv'. The cleaned data file was generated by the data cleaning utility and involved taking the

original data from three files, and condensing them into one file with all of the data columns relevant to the recommender system and/or the web app. Games without any positive ratings or playtime or that do not have English language support were excluded.

Another data file, the similarity matrix in .csv format, was generated by the recommender utility but was deleted because once the top ten recommendations file is generated the original similarity matrix is no longer needed and it was taking up over 660MB. The only time the similarity matrix, along with the cleaned games data and top ten recs data need to be re-created is when either the original game data changes, or new recommendations need to be made because the similarity calculation algorithm has been adjusted.

Data Product Code

The code for the entire system is contained in the recommender_app folder. The descriptive method is the k-means cluster analysis that's done in the Analytics section of the web app. The code is contained in the recommender_app.py file, under the Analytics Route about halfway down the file. This code uses the kMeans algorithm from the sklearn.cluster library, which divides the data into N clusters to help us visualize relationships better, and in this case N is ten clusters. This is visualized on the Analytics page through the matplotlib library and a library called mpld3 which allows matplotlib graphs to be embedded into a website. The resulting graph shows us that if we want to release a large game that we expect to be popular (around 200-400 hours of gameplay on average) then it would be good to release it during the spring time, for example. This visualization helps us answer the business question “what is a good time for a game to be released if we were to develop one ourselves?”.

The non-descriptive method is the recommendation system, and it is a prescriptive method, prescribing ten recommendations for the game the user is viewing based on their similarity rankings to the current game. The code for this is split between the data cleaning utility in 'data_cleaning.py', the recommender utility in 'content_recommender.py', and the main web app code in 'recommender_app.py'. The 'rec_utility.py' file is the file that combines the data cleaning and recommender utilities into one console based application with a text interface.

This recommender system is based on a custom item to item similarity algorithm that assigns different weights to different game attributes and then compares one game

to another, adding similarity points based on how similar the attribute is between the two games. The weights must add up to 100 and each game gets a similarity ranking of 0 to 100 to every other game within the cleaned games dataset. For example, the current algorithm gives a weight of 20 to the game name, 15 to the genre, 20 to the tags, 20 to the description, 5 to the platform, 5 to the publisher, 10 to the developer, and 5 to the year of release. For the simpler fields like developer and publisher, the algorithm simply checks if the two strings are exactly the same and if they are, add points according to that attribute's weight to the similarity score.

For the more complex string comparisons such as name, genre, tags, and description, the algorithm checks how many words are the same between the two strings. It does this by storing one string's words in a dictionary then comparing the second string to that dictionary and counting how many words are the same, incrementing the counter by two for each word that's the same since it appears in both strings. Then, it takes the number of words that were the same and divides them by the total number of words in both strings, and that fraction is then multiplied by the weight given to that particular field and added to the similarity score.

The algorithm does all of this for every game in the cleaned games data set, and builds an $N \times N$ matrix of similarity scores where N is the number of games, which can then be written to a file and/or used to generate the top ten recommendations file. The top ten recommendations are generated by going through every row in the matrix and sorting it by similarity score using Python's built in sort algorithm. Then, the first ten scores are extracted and written to a dictionary with the key being the game's ID and the value being a list of ten tuples, with each tuple containing the similarity score and the game ID of the recommendation. Game IDs are not explicitly stored in the similarity matrix, but they are implicitly stored by position. Meaning, the first game in the list is the one with ID 0, the second one is the one with ID 1, etc. This top ten recommendations dictionary can then be written to a .csv file so it is ready to be imported by the web application.

The web application then imports this top ten recommendations data from the csv file and puts it into a dictionary, which is then used to get game data from the database and display the top ten recommendations on each game's page. There is also an extra visualization of the recommendations available by hitting the “visualize recommendations” button at the bottom of a game page. This brings up a bubble chart

of the recommendations where size indicates that game's popularity based on number of positive ratings.

Hypothesis Verification

There were two hypotheses made regarding this system: the first was that sales and customer engagement would greatly increase with this new platform and recommendation system, and the second was that decision makers in our company would be better able to decide when and for what price we should release a game on our own platform if the company decides to go through with developing our own game.

The first hypothesis is one that cannot be fully verified yet, as the product has just been completed and has not been in the hands of actual users for very long yet. Once the customers have had a chance to shop on the new platform for at least a few months we will be able to verify this with certainty. Considering that the old system didn't have any way of suggesting new products to users as they shopped, this system is practically guaranteed to increase sales and user satisfaction, but officially this hypothesis is rejected until we can observe sales numbers over the next quarter or two.

The second hypothesis is definitely verified as accepted. Using the graphs and k-means analysis provided on the Analytics section of the web app, decision makers will be able to analyze the data in ways that would be very difficult without such graphs, and there's no doubt they will be better able to make decisions pertaining to the potential release of our own game on our platform.

Effective Visualizations and Reporting

There are four main types of visualizations in this project. The first is the display of the top ten recommendations at the bottom of each game page. This is the main visualization of the prescriptive recommendation system prescribing the top ten similar games to a user. The second type of visualization is the bubble chart visualization of the top ten recommendations for any given game. This can be accessed by clicking the "Visualize Recommendations" button at the bottom of any game's page, and the bubble chart visualizes the top ten recommendations in such a way that bubble size corresponds to the popularity of that recommended game based on number of positive ratings, and it allows users to see the relative popularity of the games they've been recommended.

The other two types of visualizations appear in the Analytics page which is only accessible by Manager-type accounts, and they are pie chart and scatter plot. The pie chart at the top of the analytics page represents the user opinions of the recommendation system, which come in three categories: Great, Okay, and Bad. This pie chart allows managers to instantly see a real world accuracy metric of how the recommender system is doing.

There are also three scatter plots on the Analytics page. The first plots day of release versus average playtime, but without any cluster analysis it's difficult to draw conclusions and the default graph is also too zoomed in and using average playtime in minutes instead of hours. In the second scatter plot, a k-means analysis has been done with ten clusters and we can see that there is a cluster of games in the 200-400 hours played range that has some noticeable gaps in release date, such as the very beginning of the year, meaning we shouldn't release at that time. The third scatter plot is a plot of price versus number of positive ratings, and although there is no cluster analysis done we can see that around \$15 there are quite a few games with high numbers of positive ratings, so that might be a good price point. These scatter plots help our business leaders make better decisions.

Accuracy Analysis

For the recommender system, since it is an item to item similarity algorithm that doesn't factor in user data when making recommendations, the best way to define accuracy is what the impact is on the user. This accuracy can be measured by observing the real world feedback from users actually using the system. There is a section below the recommendations on game's page where the user can give feedback on what they think of the recommendations they got, which has three options: Great, Okay, and Bad. When the user selects an option a record is made in the RecommenderOpinions table of the Users database and the opinion is recorded numerically (2,1, or 0). These opinions are then visualized in the pie chart at the top of the Analytics page so managers can see how the recommender system is doing in real time.

If the recommender system ends up getting a lot of bad ratings, the similarity algorithm can be adjusted easily. For example, if we wanted to give the user a wider variety of recommendations, we could put less weight on the game name attribute and more weight on the genres and tags and then observe customer feedback over another few months.

For the k-means cluster graph, we can see that the clusters have a very accurate real world representation with ten clusters. For games with less than about 100 hours played on average, there are 6 different clusters corresponding to about a two month release window each, then there's another cluster right around the 200 hour mark that is arguably the dividing line between long games with a lot of playtime and those without. Then there's another cluster roughly around the 200-600 hours played mark, which can be considered very long games, and yet another above that in the 800+ range that are extremely long outliers. So because the clustering corresponds well to real world relationships we can say it's fairly accurate.

Application Testing

During development, the application was routinely and iteratively tested. Each section of the application was tested individually, as well as together when they were integrated. Finally, there was a system test done testing the entire system flow from start to finish. Here are some samples of the test plans that were used:

Data Cleaning Utility Unit Tests

Data files open without error	Pass / Fail
Data files are parsed without error	Pass / Fail
Output file contains the following columns: GameID, Name, ReleaseDate, Developer, Publisher, Platform, Genre, Tags, PositiveRatings, NegativeRatings, AveragePlaytime, Price, Description, Image, PercentageRating	Pass / Fail
Output file does not have any empty or null rows or columns	Pass / Fail

Recommender Utility Unit Tests

Cleaned Data is imported without error	Pass / Fail
Similarity scores for all games are calculated successfully, and a similarity matrix is output	Pass / Fail
Writing the similarity matrix to file works properly	Pass / Fail
Similarity matrix is N x N, where N is the number of games in the cleaned data set	Pass / Fail
Reading similarity matrix into memory from file works	Pass / Fail
Writing the top ten recommendations to file works properly	Pass / Fail

Web Back End Unit Tests

Requesting any one of the site pages (home, about, analytics, recommender, etc.) sends back an HTML template	Pass / Fail
Cleaned game data is read from csv file and loaded into memory in SQL format using SQLite	Pass / Fail
Users database is created on server startup with Users and RecommenderOpinions tables if does not exist	Pass / Fail

Web Front End Unit Tests

Navigation bar displays properly	Pass / Fail
Search bar displays properly within navigation bar	Pass / Fail
On any game's page, the top ten recommendations display in a neat fashion at the bottom of the page	Pass / Fail
The recommender opinions section appears below the recommendations on a game's page	Pass/Fail

System Tests

Run the data utility and output the cleaned data file	Pass / Fail
Run the recommender utility to build the similarity matrix, then write it to a file	Pass / Fail
From the recommender utility, create the top ten recommendations csv file	Pass / Fail
Start up the server- it should create and populate the Games database in memory, and create the Users database only if it doesn't exist	Pass / Fail
Open the home page in a browser, and search for a game. Search page with list of results should be displayed	Pass / Fail

Click on one of the search results and the game's page should display properly	Pass / Fail
Scroll down to the top ten recommendations and click on one of them. That game's page should now load	Pass / Fail
Create an account and log in with it without errors	Pass / Fail
While logged in, the “Account” link should be visible on the right of the nav bar. Visit this link and ensure the account page displays properly	Pass / Fail
Log out of the new account and log in with the test Manager account- username 'Manager', password 'test'. The 'Analytics' tab should now be visible on the nav bar	Pass / Fail
Click on the Analytics tab in the Navbar. Analytics page should display with a pie char at the top and three scatter plots, including one with k-means cluster analysis with colored clusters	Pass / Fail
Open the Users database in a SQLite database viewer such as 'DB Browser for SQLite' and check for anomalies. Passwords should be encrypted and recommender opinions should have one of three values: 0, 1 or 2	Pass / Fail

Application Files

All the application files needed for the app to run are in the `recommender_app` folder. At the top level of the hierarchy, there are all of the python files such as the data cleaning utility, the recommender algorithm, and the back end of the website. The `data` folder contains the cleaned games data and top ten recommendations data, and also contains a folder with the original three data files, and a `databases` folder for the Users database file. The `log` folder contains an application log where the app writes server information, warnings and errors to with a timestamp. This is used for monitoring and maintaining the health of the application.

The `models` folder contains the Python class model of a User that is stored in the Users database. The `static` folder contains javascript scripts and CSS stylesheets for the web app, and the `templates` folder holds all of the HTML template files that are served with Flask's Jinja templating engine. Here is a table of all the main python files at the top level and what they do:

<code>content_recommender.py</code>	The main file for the recommender system functionality. Contains the similarity algorithm and the algorithms for getting top ten recs and writing them and/or the similarity matrix to file
<code>data_cleaning.py</code>	The data cleaning utility. Contains the algorithm that cleans and combines the data from the original 3 files and outputs one cleaned file
<code>db_utility.py</code>	A database utility file containing functions that create/populate/modify the databases needed for the application
<code>forms.py</code>	The file that defines the registration and login forms to be used with the flask-wtf forms library

rec_utility.py	This file provides a console based interface for interacting with the data cleaning utility and the recommender utility. This is the file to open to work with either of those in the console
recommender_app.py	The main file of the web application. Defines the Flask backend and all relevant routes. This is the file to run to start up the Python server

Installation Guide

In order to install this application, you must have Python 3.7.2 or later installed on your system. To do this, go to the python.org website and download the latest installer for your system. After running the installer, open a command line/terminal window and type “python --version”, and you should get back the version number you installed. If not, then Python may not have been added to your PATH variable/environment variables. To do this automatically, re-run the installer and click on “Modify Installation” and check the box that says something like “Add Python to environment variables”. Also try typing 'pip --version' in the command line to ensure that you have the pip package manager. It should have come with your python installation.

Once you have Python, open a terminal window and navigate to the same directory as this document. There should be a file here called 'requirements.txt'. This file contains the names of all of the dependencies that the app needs in order to run. Now run the command “pip install -r requirements.txt” to install them all. This may take 5-10 minutes.

Once the dependencies are installed the app is ready to be run. In your terminal navigate to the recommender_app folder and then type the command “python recommender_app.py”. This should start the development server that comes with Flask. Wait about 60 seconds, and then open your internet browser and go to localhost:5000. If it says page unable to load, wait another minute or so and try refreshing again. If you see the app's homepage, the installation succeeded!

Console-based Utilities User Guide

This guide documents the functionality of the data cleaning and recommender console based utilities. The project submission already comes with the output of these utilities so there is no need to run them unless you want to personally see the flow of how they work.

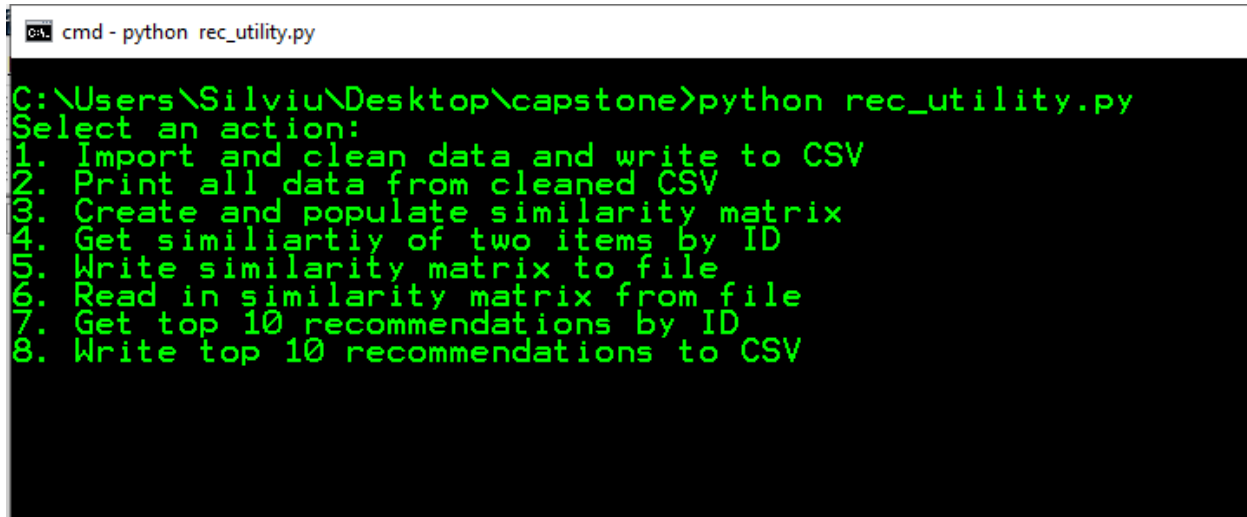
*Note that the generation of the similarity matrix by the recommender system can take around 25 to 45 minutes to complete depending on processor speed, will use up to 1GB of RAM, and, if you choose to write the matrix to a file, will produce a 660MB file in the data directory.

*Doing any of this requires Python 3.7.2 installed on your system, although any version above 3.6 should work

Here are the steps for how to produce all the output the web app needs, starting with only the original data files in the 'data/original_data' directory:

1. Open a terminal window and navigate to the project directory containing the rec_utility.py file
2. Run the rec_utility.py file with the command 'python rec_utility.py'
3.Type '1' for 'Import and clean data and write to CSV' and press Enter
4.Type '3' for 'Create and populate similarity matrix' and press Enter (this may take around 30 minutes)
5. Optional – Type '5' for 'Write Similarity Matrix to CSV'
5.Type '8' for 'Write top 10 recommendations to CSV' and press Enter.

Here is a screenshot of what the console utility looks like:

A screenshot of a Windows command prompt window. The title bar reads "cmd - python rec_utility.py". The command prompt shows the directory "C:\Users\Silviu\Desktop\capstone" and the command "python rec_utility.py" has been executed. The output is a green text menu on a black background. It starts with "Select an action:" followed by a numbered list of 8 options. The options are: 1. Import and clean data and write to CSV, 2. Print all data from cleaned CSV, 3. Create and populate similarity matrix, 4. Get similiarity of two items by ID, 5. Write similarity matrix to file, 6. Read in similarity matrix from file, 7. Get top 10 recommendations by ID, and 8. Write top 10 recommendations to CSV.

```
cmd - python rec_utility.py
C:\Users\Silviu\Desktop\capstone>python rec_utility.py
Select an action:
1. Import and clean data and write to CSV
2. Print all data from cleaned CSV
3. Create and populate similarity matrix
4. Get similiarity of two items by ID
5. Write similarity matrix to file
6. Read in similarity matrix from file
7. Get top 10 recommendations by ID
8. Write top 10 recommendations to CSV
```

The other options were for internal debugging purposes only and may not work as expected. For example, number 2 'Print all data from CSV' will only work after running number 1 while the data that was just printed to csv is still in memory, it doesn't actually read from the csv. If you closed the utility after writing the similarity matrix to file, you will have to use option 6 'Read in Similarity Matrix from file' to work with it again.

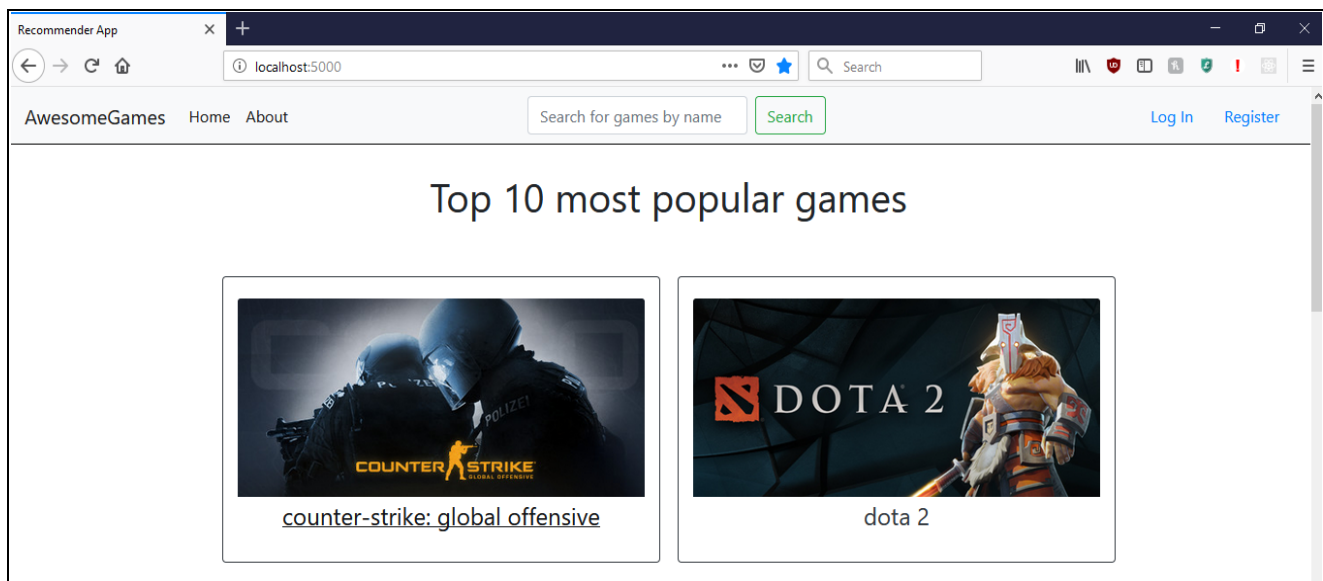
Web Application User Guide

This is a guide to all of the functionality of the web application. If the installation detailed in the installation guide was successful, then go ahead and open a terminal window and navigate to the project directory. Type 'python recommender_app.py' to start the development server. For some reason the server seems to restart once upon starting from the command line for the first time, but it's not an issue. It should look something like this:

```
cmd - python recommender_app.py

C:\Users\Silviu\Desktop\capstone>python recommender_app.py
Users database is ready
Re-populating games database
Games database is ready
* Serving Flask app "recommender_app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
Users database is ready
Re-populating games database
Games database is ready
```

Now open up a browser window (Firefox recommended as that's what I used during development, but any modern browser should work) and go to 'localhost:5000' which is where the application should be hosted by default. If that doesn't work, try waiting a minute and refreshing the page. The home screen will look like this:



The home page lists the top 10 most popular games in the game catalog and each of them can be clicked to go to that game's page.

At the top is the navigation bar, which appears on every page of the app. There are links to the Home page and About page. The About page just contains a short few

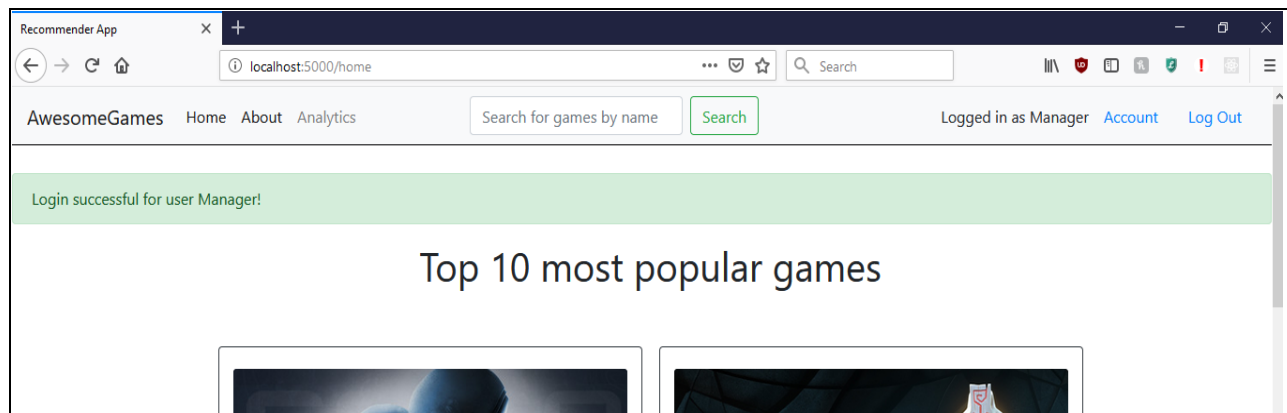
paragraphs about the fictional company AwesomeGames that this project was made for.

There is a search bar in the middle that allows you to search for games by name. Typing something in here and pressing search will take you to the search page, where up to 100 results for your query will be listed in a single column. You can click on any of them to go to that game's page.

On the right of the navigation bar are the Register and Log In links. Register will take you to a registration form where you can create a new account. Log In will take you to a log in form. Listed below are two test accounts you can use to test out the app:

Username	Password	Account Type
Manager	test	Manager
Test	aaa	Customer

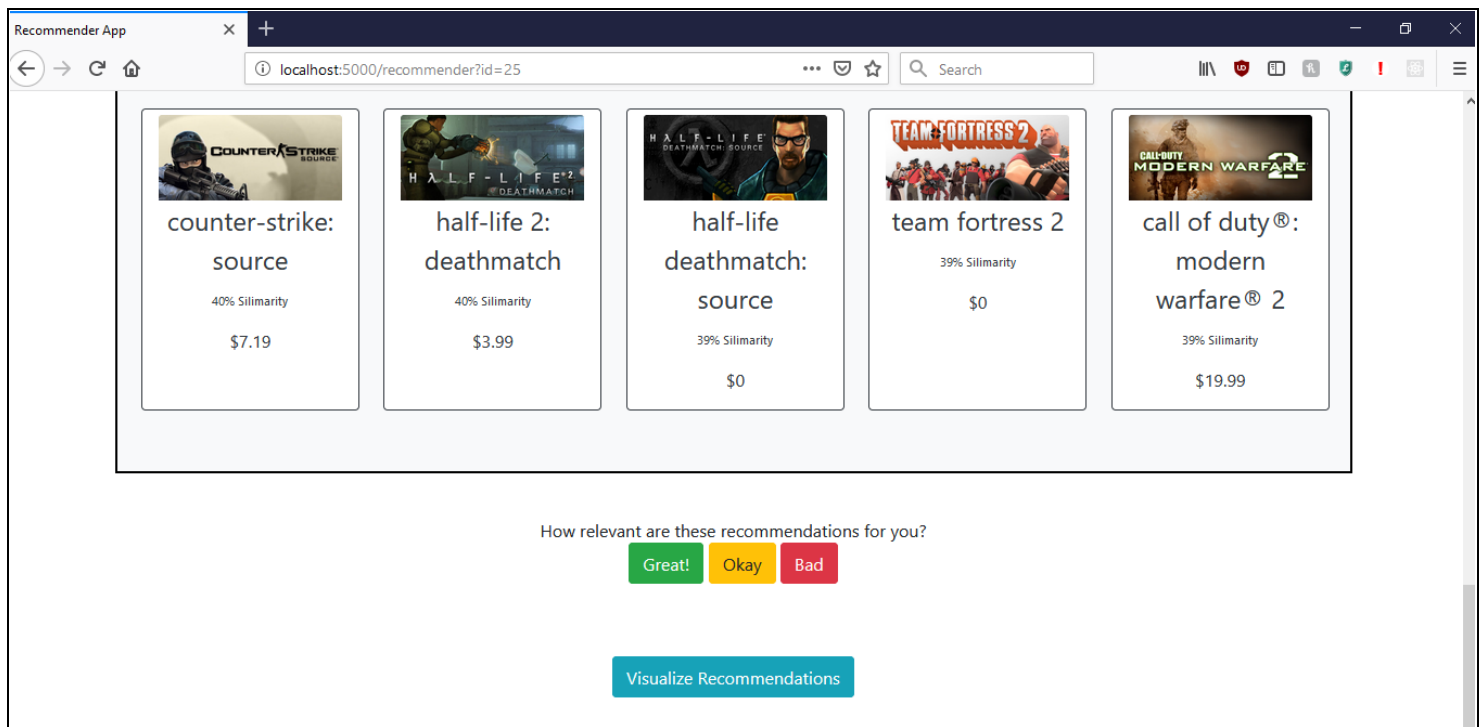
A manager type account is needed to access the Analytics tab of the application, so if you want full access that's the best account to use. Once you're logged in, the home screen will look like this:



The Log In and Register links on the right of navigation bar will be replaced with Account and Log Out links and, if you're logged in with a Manager account the Analytics

link will appear next to the About link. The Account link will take you to a page showing some basic account info like username and password, and Log Out will log the user out.

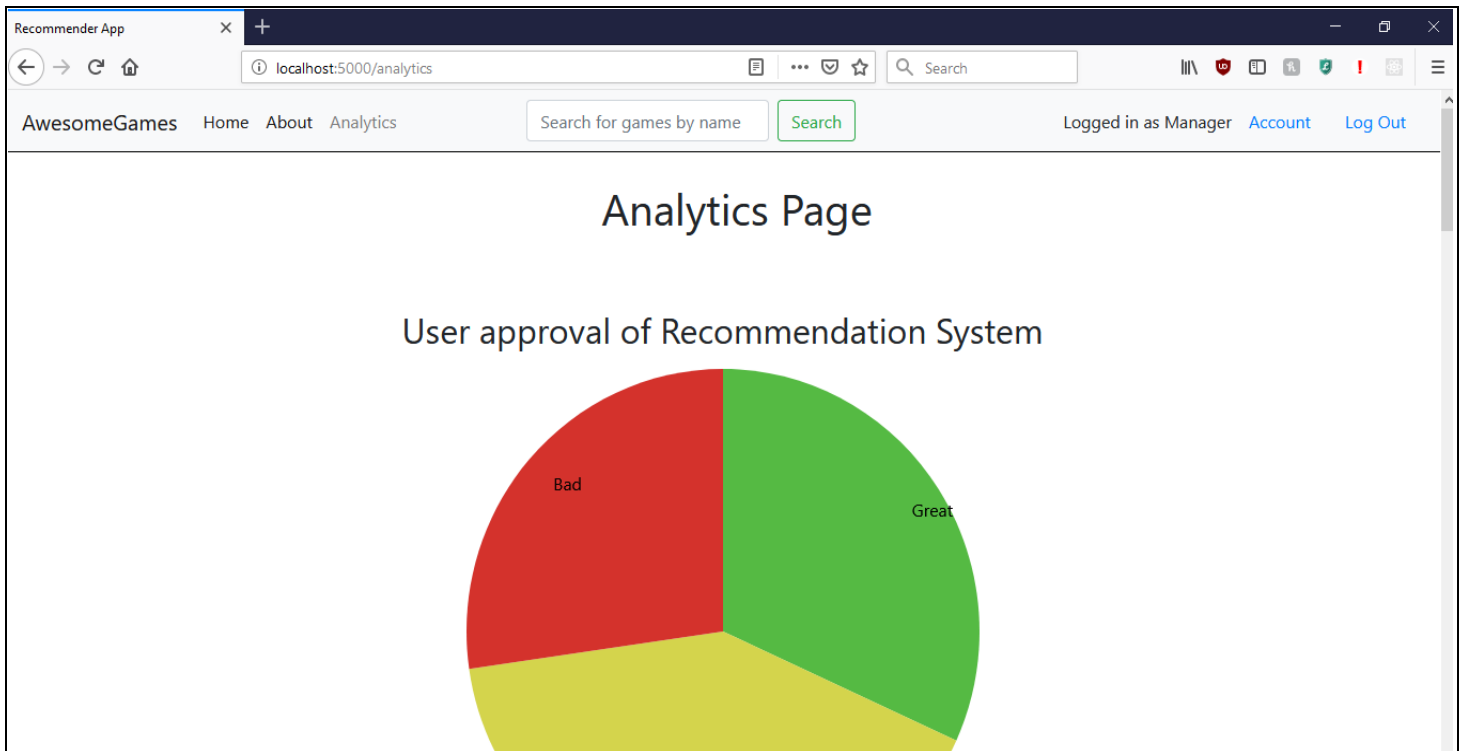
Clicking on a game either from the home screen or after searching will take you to that game's screen, which contains an image and a bunch of information about the game, and below that will be the top 10 recommendations for that game, which are also clickable links. Below that will be a feedback section for the user to give their opinion of the quality of recommendations they received, which will only appear if logged in. there is also a button to visualize the recommendations, which will open up a bubble chart of the 10 recommendations where bubble size is relative to the popularity of that game based on number of positive ratings. The bottom of the recommendations section will look like this:



(The 'Add to Cart' button on this page doesn't do anything, because it isn't a real store. The next step for AwesomeGames if they were a real company would be to create cart functionality and then link the app to something like Stripe to process payments.)

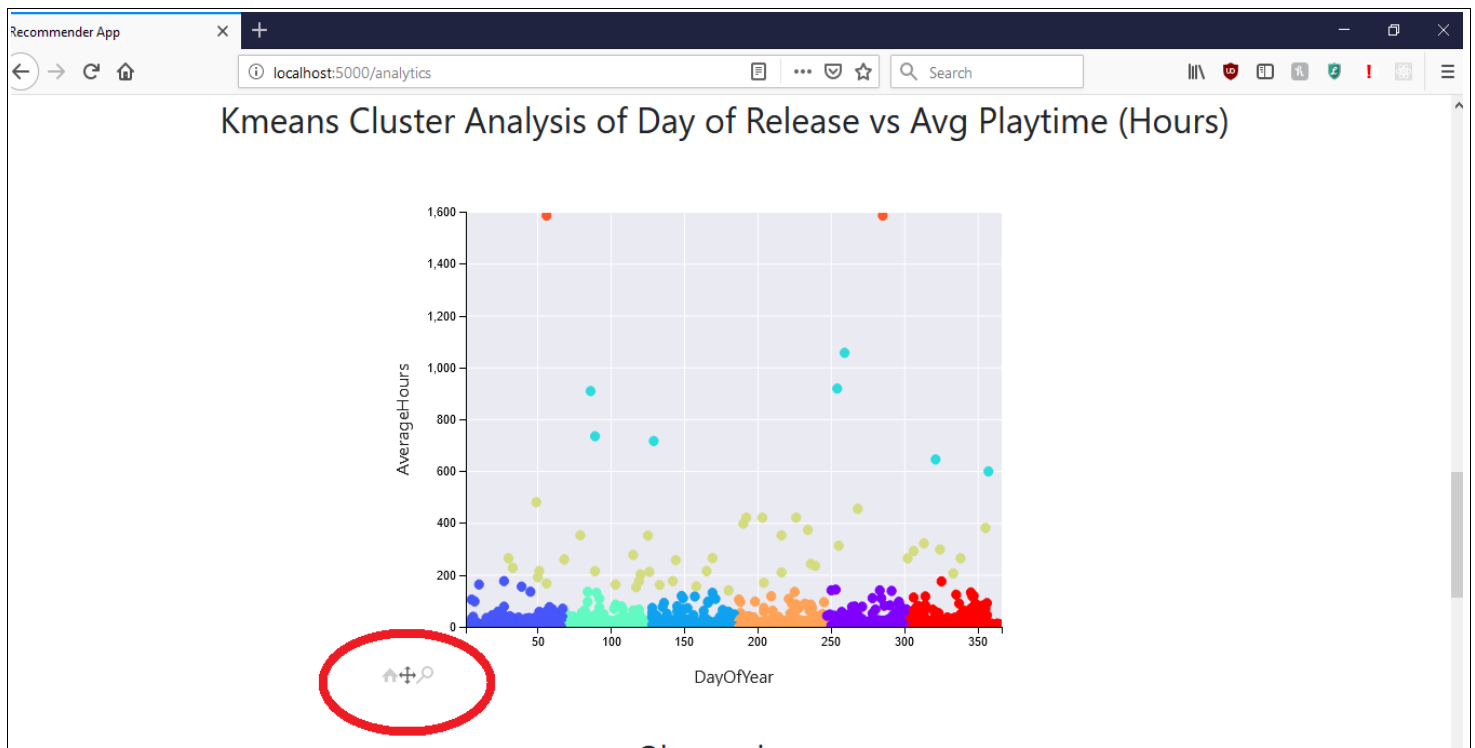
If you're logged in to a manager account, you'll have access to the Analytics tab which will allow you to see certain data visualizations that help answer business

questions. When you click on the Analytics link in the navigation bar, the page may take a few seconds to load and will look something like this:



There is a rare bug that can sometimes make this page fail to load and give a 404 type error. If this happens just refresh the page and it should work. At the top will be the pie chart representing the user feedback of the recommendation system. Currently the opinions come from a few test accounts that I went through and clicked on the different feedback links on a few different games to get some test opinion data. Below that you will see three scatter plots, including one with a color coded k-means cluster analysis and some descriptions that explain how these algorithms and data visualizations would help decision makers make better business decisions in relation to the games data.

These scatter plots are interactive, and if you hover at just the right spot in the bottom left of the plots, you will see three tools that will look something like this:



The arrows will allow you to move around the graph and the hourglass will allow you to zoom. The home icon will reset the graph back to default. Note that these tools might be a bit laggy.

Summation of Learning Experience

Most of my prior experience with Python and programming in general had been from taking courses either at WGU or at my local community college, and I had built web apps before but never a project this large and involved. Overall, I was pretty comfortable working with Python and creating web apps but still had to learn a ton about Flask and Python especially about machine learning and analytical tools like k-means analysis. Most of the web apps I'd built before were mostly front-end websites, so I had to read a lot about back-end routing and AJAX.

I spent a ton of time looking at both Python and Flask documentation, and the documentation of many different libraries such as sqlite3, and also Googling bugs I came across, often landing at a StackOverflow post. One great video series I watched was a tutorial series for Flask apps on YouTube made by user Corey Schafer, which helped me a lot in structuring my Flask app.

This project introduced me to working with big data, and I spent a lot of time in the early stages of the project reading articles and watching videos about different ways of creating recommendation systems and thinking about how I could create one with the data that I had, which is a games dataset scraped from Steam that I found on Kaggle.

Overall, this project was awesome and quite challenging and I learned a ton about Python, Flask, AJAX, and the way web apps are structured, as well as a ton about working with big data and creating recommendation systems. This is the largest app I've ever built and I now feel very confident going out and applying for software engineer jobs, which was my goal with getting this degree.

Bibliography

M, Madasamy. (2019, February 16). Introduction to recommendation systems and How to design Recommendation system,that resembling the Amazon. Retrieved from <https://medium.com/@madasamy/introduction-to-recommendation-systems-and-how-to-design-recommendation-system-that-resembling-the-9ac167e30e95>

Schafer, Corey. (n.d). Home [YouTube Channel]. Retrieved from <https://www.youtube.com/user/schafer5/>

Sharma, P. (2018, June 21). Comprehensive Guide to build Recommendation Engine from scratch. Retrieved from <https://www.analyticsvidhya.com/blog/2018/06/comprehensive-guide-recommendation-engine-python/>

Teichmann, J. (2019, April 6). How to build a Recommendation Engine quick and simple. Retrieved from <https://towardsdatascience.com/how-to-build-a-recommendation-engine-quick-and-simple-aec8c71a823e>