

Department of Computer Science

Individual Project – CS3IP16

Predictive Analytics for Film User Ratings

|  |  |
| --- | --- |
| **Student:** | Kane Small |
| **Student Number:** | 23013043 |
| **Supervisor:** | Jonathan Boyle |
| **Submission Date:** | 20th April 2018 |
| **Word Count:** | 19,095 |

Abstract

Machine learning is being utilised more and more as of late, to help with assisting and automating both simple and complex tasks and for finding patterns amongst datasets. The film critic industry is often perceived as a biased collection of reviews that do not take into account a broad enough selection of opinions, which ultimately makes it difficult for the viewer to discern whether to watch a film solely based on an average rating of critic reviews. Primarily, this is all that is currently available in the way of platforms that provide information on unreleased films and often, the ratings are not even present before the official release date. This project aims to solve this exact problem, by providing users with a singular value that analyses existing film metadata to find the relationships and ascertain the underlying features that result in a film reaching critical acclaim or failing at the box office; as well as everything in between. Different machine learning implementations will be investigated to determine the best model to make predictions and once a viable option has been selected, the results will be displayed to the user via a front-end web application interface.

# Acknowledgements

First and foremost, I would like to express my gratitude toward Jonathan Boyle for his continued support throughout my project’s lifetime and helping to guide the project in the right direction.

Secondly, I would also like to thank my Mother for her wisdom, support and strength throughout this entire year. I could not have done it without her.

Lastly, I would like to thank my friends for assisting me with carrying out testing and providing valuable feedback toward improving my project along the way.

Table of Contents

[Acknowledgements i](#_Toc512075517)

[Table of contents ii](#_Toc512074832)

[Glossary of Terms and Abbreviations iv](#_Toc512075518)

[1 Introduction 1](#_Toc512075519)

[2 Problem Articulation and Technical Specification 3](#_Toc512075520)

[2.1 Problem statement 3](#_Toc512075521)

[2.2 Stakeholders 3](#_Toc512075522)

[2.3 Technical specification 3](#_Toc512075523)

[3 Literature Review 6](#_Toc512075524)

[3.1 Similar projects 6](#_Toc512075525)

[3.2 Rating platforms 7](#_Toc512075526)

[3.3 Data sources 8](#_Toc512075527)

[3.4 Machine learning technologies 10](#_Toc512075528)

[3.5 Feature selection 11](#_Toc512075529)

[3.6 Back-end technologies 13](#_Toc512075530)

[3.7 Front-end technologies 13](#_Toc512075531)

[4 Solution Approach 16](#_Toc512075532)

[4.1 Similar project takeaways 16](#_Toc512075533)

[4.2 Data sources 16](#_Toc512075534)

[4.3 Machine learning technologies 16](#_Toc512075535)

[4.4 Feature selection 16](#_Toc512075536)

[4.5 Back-end technologies 17](#_Toc512075537)

[4.6 Front-end technologies 17](#_Toc512075538)

[4.7 Chosen solution 17](#_Toc512075539)

[5 Design and Implementation 19](#_Toc512075540)

[5.1 Environment configuration 19](#_Toc512075541)

[5.2 Data acquisition 22](#_Toc512075542)

[5.3 Data cleansing/pre-processing 26](#_Toc512075543)

[5.4 Regression algorithm and predictions 28](#_Toc512075544)

[5.5 API routing 28](#_Toc512075545)

[5.6 Front-end design and user experience 30](#_Toc512075546)

[6 Testing Verification and Validation 31](#_Toc512075547)

[6.1 Unit testing 31](#_Toc512075548)

[6.2 Compatibility testing 37](#_Toc512075549)

[6.3 Usability testing 38](#_Toc512075550)

[6.4 Regression metrics 38](#_Toc512075551)

[7 Discussion: Contribution and Reflection 40](#_Toc512075552)

[7.1 Discussion 40](#_Toc512075553)

[7.2 Reflection 45](#_Toc512075554)

[8 Social, Legal, Health and Safety and Ethical Issues 47](#_Toc512075555)

[8.1 Social, legal and ethical issues 47](#_Toc512075556)

[8.2 Health and safety risks 47](#_Toc512075557)

[9 Conclusion and Future Improvements 48](#_Toc512075558)

[9.1 Conclusion 48](#_Toc512075559)

[9.2 Future Improvements 49](#_Toc512075560)

[10 References 51](#_Toc512075561)

[11 Appendices 55](#_Toc512075562)

[11.1 Appendix 1: Project Initiation Document 55](#_Toc512075563)

[11.2 Appendix 2: Logbook 64](#_Toc512075564)

[11.3 Appendix 3: Questionnaire 12](#_Toc512075565)

[11.4 Appendix 4: Scikit-learn algorithm cheat sheet 15](#_Toc512075566)

# Glossary of Terms and Abbreviations

API – Application Programming Interface

CA – Certificate Authority

CSS – Cascading Style Sheets

FTP – File Transfer Protocol

HTML – Hypertext Markup Language

IMDb – The Internet Movie Database

JS – JavaScript

JSON – JavaScript Object Notation

MAE – Mean Absolute Error

OMDb – The Open Movie Database

PCA – Principal Component Analysis

PID – Project Initiation Document

RDBMS – Relational Database Management System

RDP – Remote Desktop Protocol

SSH – Secure Shell

TMDb – The Movie Database

UI – User Interface

VFX – Visual Effects

VPN – Virtual Private Network

# Introduction

Machine learning is still in a very primitive state at this point in time but is being utilised a lot more frequently across different industries, from entertainment, to finance, to transportation services and many more (Sas.com, 2018). The process for machine learning models is to formulate patterns and understand the inter-linking features within the dataset being analysed, that can then be used against previously unseen datasets in order to predict values to a certain degree of accuracy. But ultimately, the end goal for using this new technology is to make better decisions without the need for human intervention.

This is where this project comes into play. The application uses machine learning regression algorithm to train a model that can successfully predict rating values for films that have yet to be released, based on the most important key attributes (the features) that influence viewer’s decisions. The main purpose behind building this application is to essentially make it easier for anyone to decide whether to see a film, based solely on a single value. This singular rating percentage is the result of, finding the inter-linking patterns and relationships of the most important film attributes and is ultimately as unbiased as possible by using only factual information.

The project commences by identifying the problem that is trying to be solved, listing the stakeholders that will form the main target audience of the application and more importantly, the main objectives of the project. In order to accomplish these objectives, it is necessary for the project to be split into five separate parts, data acquisition, data cleansing/pre-processing, regression algorithm implementation, API routing and front-end design/user experience. The first phase focuses on acquiring data from a viable source that meets the requirements (i.e. relevant metadata), the next phase transforms the acquired data into a correct format for the input to the algorithm and stores it into the database, the thirds phase implements the regression algorithm and train a model; the fourth phase revolves around building an API that will interface with the database and display data to the front-end and the last phase will concentrate on the user-experience and ease-of-use of the application.

Furthermore, the motivation and reasoning behind undertaking this project is brought to light, followed by the constraints that illustrate potential challenges for the project.

The application must then look next at any existing applications that carry out either the same or similar tasks and see how it can be improved upon, from both a machine learning accuracy and a user-experience perspective. This is accomplished by reviewing existing literature and from the findings, justify an informed solution approach on how best to meet the objectives outlined. The solution approach itself will also aim to cover any tools and technologies utilised throughout the development process in brief.

The design and implementation of the application will be explored in detail, illustrating exactly how the technologies and any libraries aided in the development of the product. The implementation will build all the way to the result of the model, which will be examined in detail as part of the discussion. This will also address the success of the solution, as well as any limitations that have been identified along the way. A reflective piece will lead on from here, bringing to light what the challenges were, what has been carried out well and how the project would be approached differently if attempted a second time.

Any potential social, legal, ethical and health and safety risks of the project are reviewed in short at this point, which will support the continued development and improvement of the application in the future.

Lastly, the report comes to an with a conclusion that quickly restates the main project objectives and discusses whether the solution that has been implemented has in fact met them. Nevertheless, this outcome leads into a list of future improvements that could potentially elevate both the project’s effectiveness and use.

# Problem Articulation and Technical Specification

## Problem statement

The number of variables that determine whether a film will be a high user-rated success has increased dramatically over the years, in-part due to the advancements made within the film industry allowing production companies to offer a wider variety of films to the viewers, specifically in the Visual Effects (VFX) category. So, with this comes the pressure of leaving it to the viewer to determine whether a film is going to be enjoyable based on these attributes. But, a viewer can only use what they already know about an upcoming film such as the genre of the film, or the actors cast, or the trailer and so on and these attributes will simply *influence* their decisions; but will not tell them whether or not this film will result in a decent or meagre film-viewing experience.

This project aims to alleviate that pressure on viewers by taking all of the factual data already present and using it to make accurate predictions on the overall rating of unreleased films, but by taking into account each of the most important film attributes and even finding relationships between them.

## Stakeholders

A total of three stakeholders were identified for this particular project, revealing how they would be affected by or provide assistance in achieving the outcome of the project.

### Developer – Kane Small

The developer, Kane Small, is responsible for advancing the project through all phases of the development lifecycle. This involves making sure that the problem statement outlined in section *3.1* is resolved and this will be accomplished by fulfilling the objectives outlined in section *3.3*. Due to time constraints and other important engagements throughout the year, the developer will also be responsible for maintaining an effective pace through time management and organisation, to ensure that project goals are met on time.

### Project supervisor – Jonathan Boyle

The project supervisor, Jonathan Boyle, is the person who will provide continued support and assistance toward guiding the project in the right direction at all times. This will be accomplished by scheduling weekly and/or bi-weekly meetings whereby a detailed discussion will comprise of the work that has been carried out and any problems encountered since the last meeting ending with a summary of what will be tackled by the next meeting.

### User – Film Viewers

The user of the finished product will be, any film viewer that is uncertain on whether or not to see an upcoming film. They will expect to use a system that is easy-to-view and understand, where the content has been organised in a concise and logical manner but most importantly that the prediction result being displayed is as accurate as possible; as they will be relying on this factor to make their decisions for them.

## Technical specification

The application to be developed will reside on a server that will deliver the content to the internet. There will be two pages, the *main* application page that will display all of the film metadata including the prediction result and a *trends* page that will provide users with updated trend graphs for the data being held.

Utilising the previously created PID, the project must satisfy the objectives outlined. However, the objectives that are listed below have been altered in order to conform to realistic time constraints and developer experience:

**Data acquisition**

* Implement a viable API data source
* Utilise the YouTube API for film trailer acquisition
* Design, build and configure the database to store the film data
* Implement a viable Python framework
* Build the scripts that will perform the data acquisition
* Setup and configure a server to host the scripts
* Create a questionnaire to aid with feature selection

**Data cleansing/pre-processing**

* Use imputation to format NULL data values
* Use one-hot encoding to format categorial features
* Use Principal Component Analysis (PCA) to reduce dimensions

**Machine learning implementation**

* Select an appropriate machine learning algorithm
* Fit the formatted data to the algorithm
* Train the algorithm to produce a model that can output a set of predictions
* Test the accuracy of the model using different regression metrics

**API routing**

* Create API routes to serve data to the front-end
* Implement basic API route authentication

**Front-end design and user experience**

* Create a functional front-end interface for the user to interact with
* Implement the typeahead and bloodhound autocomplete and suggestion engine
* Allow the user to query for films
* Implement charting/graphing capabilities to display data trends

### Project motivation

The main motivation behind carrying out this project is, the developer’s passion for film. Having had the idea to implement some form of film aggregation and search platform in the past, but not knowing exactly how to make the application stand-out, combining a field of interest such as machine learning has enabled the project to reach a state that is worthy of development.

The application itself, also has a lot of possibility for future implementations such as, including TV show support, or allowing custom search parameters to be user-inputted to test different film recipes – which would be extremely useful for production companies.

### Project constraints

The biggest constraints to consider are scheduling and scope. Ensuring that, within the given time it will be possible to research, learn and implement a successful machine learning algorithm that will be able to output a set of predictions. But more importantly, guaranteeing that the project is not over-scoped and that the objectives outlined are focused on first and foremost, where additional features and functionality can be implemented if there is additional time at the end of the development lifecycle.

# Literature Review

In order to produce a viable solution to the problem articulated in section 2.1, first research must be conducted into existing projects that are similar to this, in order to learn from what they accomplished well but also to avoid the limitations they presented. Existing rating platforms will be analysed to determine what rating information is often presented to users and how it is calculated. A viable data source must be selected to handle the acquisition of film metadata, so a comparison of available sources will be carried out. In order to implement a machine learning model, the different categories and problems within the field must be properly examined so that an informed decision can be made on the best technique to utilise. Lastly, the front and back-end technologies will be explored to decide what will be the best setup for a project like this and will ultimately satisfy the problem presented.

## Similar projects

A paper released by students at the University of Salford, aimed to implement data mining techniques to analyse and predict film ratings (Saraee, et al., 2004). The paper outlines IMDb as their data source, retrieving film data in a collection of un-formatted text documents that were all linked by the film title. This is definitely not a viable source for acquiring data, as each crucial film attribute would have to be extracted manually from the relevant text documents and combined into a single structured format for each film. As mentioned in the paper, this would require some form of natural language processing (NLP) to extract the key pieces of information.

The paper split the evaluation phase into four separate sub-phases, pre-selection, cleansing and integration, selection and transformation and data mining. Pre-selection, involved filtering out the unwanted files that contained data that was unnecessary or where the information was already present in an alternate file such as, cinematographers, or actors and so on. The cleaning and integration sub-phase revolved around processing each file into comma separated value (CSV) file so that these could later be read by their choice of database management system, which happened to be Microsoft Access. Within Microsoft Access, tables could be formed from the CSV files linking all attribute tables (actors, ratings and so on) to a table with the unique film entries.

Next came the selection and transformation sub-phase, centring around calculating numerical rating values for fields such as directors and actors, as it was to their understanding that these particular film attributes would heavily influence a film’s success. In order to calculate these values, Structured Query Language (SQL) queries were constructed that computed the average user rating for every film an actor or director had been in or directed, respectively. This produced a singular rating value for each individual actor and director stored in the database.

The final sub-phase could only be accomplished once the data had been cleaned and transformed, via the previous two steps. This project trained a classifier which would be used to predict a class or each film into one of the following categories:

|  |  |
| --- | --- |
| Scale | Category |
| 7.5 – 10 | Excellent |
| 5 – 7.4 | Average |
| 2.5 – 4.9 | Poor |
| 1 – 2.4 | Terrible |

Table - Rating categories from the academic paper - A data mining approach to analysis and prediction of movie ratings

So, this method focused on generalising the prediction rating instead of providing a continuous numerical value as the output. It could be argued that this provides users with a much simpler method of seeing whether a film has been predicted to do well or not, to influence their decision on whether to see it upon release.

|  |  |
| --- | --- |
| Film | Rating |
| 5-25-77 (2005) | Average |
| Ask the Dust (2005) | Average |
| Batman Begins (2005) | Excellent |
| Because of Winn-Dixie (2005) | Poor |
| Bewitched (2005) | Average |
| Bridget Jones: The Edge of Reason (2004) | Average |
| Cars (2005) | Excellent |
| … | … |

Table - Predicted film ratings from the academic paper - A data mining approach to analysis and prediction of movie ratings

A subset of the results they achieved can be found in Table 2 above. Taking a couple of the films as an example, Batman Begins which was set to release in 2005 at the time the paper had been written, sits at a certified fresh rating of 84% on Rotten Tomatoes (Rotten Tomatoes, 2018). On IMDb itself, Batman Begins has an average user rating of 8.3/10 (IMDb, 2018). On the opposite end of the scale, Because of Winn-Dixie has a *rotten* rating of 54% on Rotten Tomatoes (Rotten Tomatoes, 2018) and an average user rating of 6.4/10 on IMDb (IMDb, 2018). Consequently, these results do illustrate at least some level of accuracy even 14 years on, but a larger sample of films would need to be tested to confirm the accuracy.

Ultimately, this academic paper has highlighted a lot of interesting points to consider when carrying out this project. They had a lot of trouble with data source selected, so locating one that returns the film metadata in a viable format that is still structured would be recommended. They also did not take into account a large selection of film attributes and therefore will potentially be biased toward the actors and directors of each film.

## Rating platforms

### Rotten tomatoes

Rotten tomatoes is one of the top aggregator of film and TV show reviews from expert critics in the industry (Rotten Tomatoes, 2018). They are well known for the *Tomatometer*, which essentially is a scoring system calculated based on the opinions of hundreds of film and TV show critics. The value itself is a percentage rating that illustrates how positively or how negatively a film or TV show has been reviewed by critics, but is only displayed after at least five reviews have been acquired.

There are different categories within the Rotten Tomatoes platform, such as the *Certified Fresh* and *Rotten* statuses that reveal to a user whether a film or TV show has either scored above 75% or below 60% in total positive reviews. This is useful to the user for establishing a visually eye-catching marker that distinguishes a truly terrible film or TV show, from a critically acclaimed film or TV show.

Lastly, there is the *Audience rating* which reveals the percentage of Rotten Tomatoes users that have rated the show. The interesting value here is the value that is displayed for an unreleased film or TV show, which indicates how many users have added an upcoming film to their *Want to-See* list. This however, is a biased rating as it only takes into account whether or not a user *wants* to see a film and not any additional factual information that can be acquired. This is a value that would be useful as a feature when training a model, but does not reveal to the user a predicted value of whether or not the film *will* be good. But, it isn’t as biased as a critic rating, as the number of critic reviews is always smaller than the total number of users that have added a film to their *Want to-See* list.

### Internet Movie Database (IMDb)

IMDb is another (IMDb, 2018) is another film and TV content aggregator, tailored more toward providing the content and metadata for films and TV shows as opposed to average critic review ratings. But the platform does provide a popularity score on each film and TV show page, indicating to users how popular that film or TV show has been by the number of visitors, for that week. They also have a user rating value, that represents the mean value of all users that have casted a vote for a film or TV show, on a rating scale of 1-10.

IMDb’s user rating value, isn’t as biased as Rotten Tomatoes’ critic rating, as there is a much larger audience or sample to calculate a mean rating from. A lot of content on IMDb, films especially, can receive hundreds of thousands of votes. However, as IMDb mentions here (IMDb, 2018), the vote averages that are displayed on a film or TV show page are weighted averages as opposed to raw data averages. So, the values have been filtered to try to reduce *vote stuffing*, whereby users create multiple accounts to purposefully boost or reduce a rating. Though, the methods by which IMDb carries out the filtering are not released to the public, most likely to prevent attackers from creating bots that can overcome the filtering.

### Metacritic

The third and final rating platform (Metacritic, 2018), is similar to the previous two in that it provides user with most of the film or TV show metadata as well as a *Metascore*, but it also does this for both video games and music. The Metascores are calculated very similarly to both Rotten Tomatoes and IMDb, by utilising a weighted average of critic reviews, but for film and music they also normalise the end result (Metacritic, 2018).

### Summary

So, with each rating platforms there are limitations as well as benefits. Rotten Tomatoes is the only one out of the three platforms to display some form of indicator as to what it’s users think about an upcoming film by means of the *Want to-See* list, which could be useful as a feature for training purposes. IMDb has a lot more votes on average, per film or TV show and because these are *supposedly* all user ratings, the end result should be a much less biased one. Metacritic is very similar to Rotten Tomatoes, except it offers generated ratings for video games and music, which neither Rotten Tomatoes or IMDb offer.

## Data sources

A variety of data sources were researched to see which could be used as the primary origin to obtain film metadata from. The most important feature was how easy it would be to retrieve the data from the source, so the preferred method would be via an Application Programming Interface (API). An API essentially, abstracts the complex code from the user and reveal simple and formatted data instead (MDN Web Docs, 2018). They also interface with the database so that the front-end never has to directly make database calls, which is slow and insecure.

### The Open Movie Database (OMDb)

The OMDb was the first data source considered, simply because it happened to be one that the developer had heard of in the past. It is an open source service that offer an API to retrieve film metadata (OMDb, 2018). The site however, does limit the number of requests that a developer can make using an API key to 1,000 a day. This limit is then increased and eventually removed, depending on the reward tier you are subscribed to if you become a patron. This involves paying a monthly donation to the developer of the service (Patreon, 2018). At $1 per month, users receive basic API access and 100,000 daily requests. At $5 per month, users receive access to the basic API as well as the poster API and are limited to 250,000 requests a day. Lastly, at $10 per month users are provided with their own server that has no limits and access to the API’s mentioned above.

### Rotten Tomatoes

Rotten Tomatoes was also considered and has been analysed as a rating platform above in section 3.2.1. It provides access to the Tomatometer and Audience rating values, as well as a subset of critic reviews per film but does not provide detailed film or TV show metadata (Rotten Tomatoes - Developer Network, 2018). The access to the API is also granted on a per proposal application basis and therefore, depending on their turnaround time may not have been reviewed before the project commenced.

This source would not be viable for the project, but could be used for the ratings it provides, which could be used for feature selection.

### Internet Movie Database (IMDb)

IMDb does not provide an official API. They do however, offer subsets of their data that can be downloaded manually (IMDb - Datasets, 2018). These separate tab-separated-values (TSV) files would have to be formatted and then combined before being stored in a database. Due to time constraints, this might not be the most viable data source solution. The data would also have to be filtered, i.e. filtering out adult films, after the data had been stored in a database.

### The Movie Database (TMDb)

Lastly, there is TMDb. Another open source service that offer an API to gain access to their data. They offer an extremely detailed documentation, that lists each of the routes available and how to use them including information on the query strings and the responses given (TMDb - API, 2018). The API also provides a range of film metadata that satisfy the project requirements.

The API is rate-limited, whereby developers can only perform 40 requests every 10 seconds, but this is not as much of a problem as the OMDb which had a maximum number of daily requests. The rate limit will just have to be observed so that the application doesn’t exceed it. This can easily be accomplished by reading the *X-RateLimit* request headers and keeping track of the *X-RateLimit-Remaining* value. If this value is exceeded then, an HTTP status will be returned with the *Retry-After* response header and after this cool down period expires requests can continue (TMDb - Rate Limiting, 2018).

The response from the API is in a JavaScript Object Notation (JSON) format, which enables the key information to be extracted with ease. JSON is format for displaying/storing structured data and is most commonly used for data transmission within web applications (MDN Web Docs, 2018). Data is stored within a JSON file in a key/value pair format, where the keys are always strings and the value is a supported data type.

If selected, this option would be able to provide the required metadata, in a format that would be easy to manipulate and without any serious performance/time limitations.

### Summary

Again, each data source in this list have their defining strengths and weaknesses but ultimately, the source selected will be the one that offers the best API support and satisfies the requirements/objectives of the project. The OMDb definitely offers a wider selection of metadata such as reward information and Rotten Tomatoes, Metacritic and IMDb ratings. But, with the recurring monthly payment, it would have to be decided whether the additional metadata would be worth the cost. TMDb also did say that they were in the process of providing API support for awards in 2017, so this could be added very soon (TMDb, 2018).

## Machine learning technologies

### Algorithm/model selection

Machine learning and data mining often get confused and was something that warranted additional research. Machine learning is the process of building models using certain data mining techniques, in order to predict future outcomes on a given dataset. Data mining, is the process of finding patterns and/or relationships within datasets.

Before selecting an appropriate model, the different categories in which machine learning problems are organised must first be understood.

There are two main categories that machine learning algorithms can be split into, supervised and unsupervised learning (Scikit-learn, 2018). Supervised learning is focused on finding the relationships in a dataset between the target value that is trying to be predicted and the remaining features that influence the target. Problems within this category will be one of two types, classification or regression. The easiest way to distinguish the two is to look at the target value, if it is a class value then predicting the target will be a classification problem, whereas if the target value is a continuous value then it is a regression problem.

Unsupervised learning concentrates on, not being trained alongside any target values and predicting an output based on those attributes. Two of the common problems within this category are, clustering and association. Clustering problems look to find and group data values that have similarities. A common real-world example of a clustering problem is Netflix’s recommendation engine. They cluster groups of similar films together in order to better suggest films to users based on what they have viewed in the past. Association problems on the other hand, looks at a collection of items and produces association rules that will attempt to predict the occurrence of one item based on the occurrence of other similar items. A common real world example of this type of problem being used is for the Amazon *Frequently bought together* section, which will recommend related items based on the item currently being searched as well previous user purchases.

Data often needs to be pre-processed before fitting it to the correct model. This step is often just as, if not more important than training the model itself as, poorly formatted data can affect the predicted results of a model. Data pre-processing can be split into the following categories:

**Data cleaning**

* Often primarily focused on removing duplicates and handling NULL values

**Data transformation**

* Often focused on implementing standardisation in order to format all attributes in a given dataset onto a uniform scale, where 0 is the baseline

**Data reduction**

* Often focused on implementing techniques such as principal component analysis (PCA), which reduces down the number of dimensions (columns) within a given dataset

There is a lot to consider when it comes to selecting the correct model, but once the problem being addressed has been filtered into one of the main categories it makes the process somewhat easier to manage.

### Useful libraries

**Tensorflow**

Tensorflow is an open source machine learning and data graphing library originally built by Google (Tensorflow, 2018) and is being used by a lot of big-name brands such as AMD, Dropbox, Snapchat, Intel and many more. From an initial overview, it seems as if Tensorflow is not as welcoming to newcomers in the field of machine learning. The documentation is somewhat complicated and it isn’t too clear what is the best way to start using the library. It does however, offer guides on how to install it on at least three unique platform operating systems and even offers a simplified interface that is supposed to mimic the more established scikit-learn library to easy users through the transition (GitHub - Tensorflow, 2018).

**Scikit-learn**

Scikit-learn is another popular open source machine learning library for python. It does have a few dependencies such as NumPy, SciPy and matplotlib as it was built around these tools, but these are again all open source libraries (Scikit-learn, 2018).

From a cursory overview, scikit-learn is a lot more tailored to inviting users who are both new to machine learning and data analytics as well as experienced industry professionals. The documentation is thorough and provides detailed and walked through examples in the form of tutorials, which makes implementing the library easier to understand. It also provides information on how to select the appropriate model for the dataset you have access to, which again for someone starting out in the field is extremely helpful.

## Feature selection

Feature selection is an important part of machine learning and centres around finding the most important features that directly influence the value being targeted for prediction.

### Questionnaire

To assist with feature selection, a questionnaire was carried out to determine what a subset of real world viewers thought their most important film attributes were and to rank those in order of how they influence the decision of whether or not to see a film.

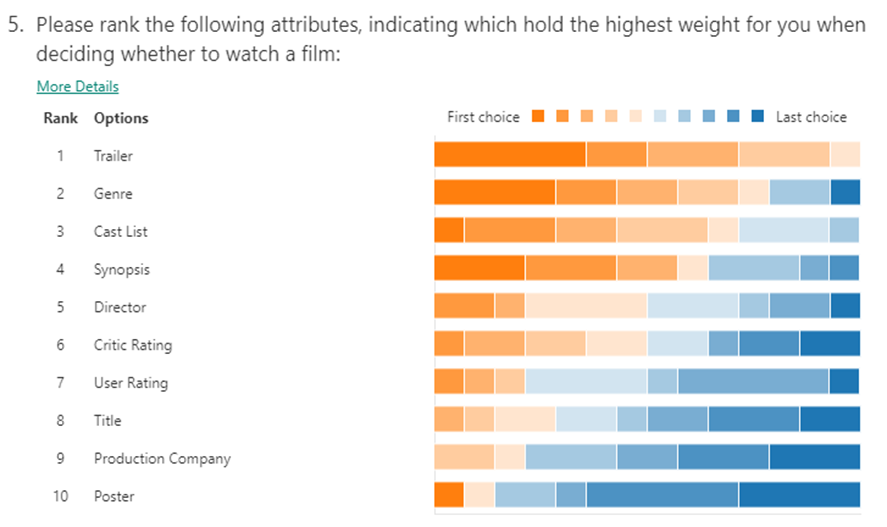
The full list of both questions and answers can be found under 11.3 Appendix 3. But the most important results are those from question 5 which asked users to rank a predefined list of film attributes in order of how they influence your film-watching decisions.

Figure - Film attribute ranking questionnaire results

Figure 1 above illustrates, from the small subset of users who responded, that the most important film attribute feature is the trailer. Therefore, taking into account a film trailer’s view, like and dislike count as model features could help increase the predictions accuracy. Surprisingly, both the critic and user ratings came in at positions 6 and 7 respectively, proving to a certain degree that the current platforms that present these values to users are not helping viewers decide on whether an upcoming film will be worth watching. Taking into account as many features as possible however, will provide a much more accurate and unbiased prediction result.

### Sentiment analysis

Sentiment analysis is a technique that focuses around natural language processing (NLP), being able to train a computer to understand human language (Bird, et al., 2009) and from that, decide whether or not the language being analysed is positive or negative (or neutral).

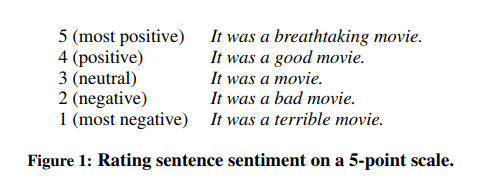
The following academic paper uses sentiment analysis to rate film reviews on a scale of 1-5, see Figure 2 for the full scale categories (Roberts & Yan, 2014). They utilised a dataset of ~10,000 film reviews/sentences and split those into an 80%, 20% training and test set respectively to train their model to classify film reviews.

Figure - Sentiment rating scale (Academic paper)

This technique could be extremely valuable in this project, by performing sentiment analysis on tweets from Twitter, or comments on Facebook posts in order to gain a popularity rating for upcoming films and provide an additional feature when training the model.

### Summary

In the end, the features selected will be the ones that can be pre-processed correctly and will satisfy the input requirements of a given machine learning model.

## Back-end technologies

### Database

**NoSQL**

NoSQL is a database system that was researched as a newer technology to familiar ones such as, Structured Query Language (SQL). NoSQL are non-relational database systems that are focused more toward performance and scaling (Amazon Web Services, Inc, 2018). Unlike traditional relational database models, that define the relationships between tables NoSQL databases store data in document-style formats like JSON for example. NoSQL databases can also be queried a lot more efficiently than traditional relational database management systems (RDBMSs) to the JSON or equivalent format data is stored in.

Depending on how the data is acquired and from which source, a NoSQL database system could work for this project but, the underlying relationships between the film data attributes would first have to be considered.

**Elasticsearch**

Elasticsearch is another NoSQL database system by Elastic but is primarily targeted at the searching and analytics of datasets (Elastic, 2018). So, data is still stored in a document format, most commonly JSON and will index the fields automatically. The Elasticsearch engine itself provides full-text querying capabilities and the results are exceedingly fast, which would be useful for the querying of specific film titles using the input field on the front-end of the application.

Elasticsearch however, does have a bit of a learning curve so may not be achievable within the given timeframe of the project and also when it comes to deploying Elasticsearch due to the complex sorting and aggregation that it has to compute, it requires a lot of powerful hardware to sustain it and that comes at a cost (Elastic, 2018).

**SQL**

SQL database systems rely on more traditional methods of storing data into tables with rows and columns and finding relationships between the tables, linking them by primary and foreign keys. SQL queries can then be sent to an RDBMS for execution.

RDBMSs are something the developer is a lot more confident with and has experience implementing, however depending on the hardware being utilised storing the required amount of film metadata might prove both challenging to store and/or query later on.

## Front-end technologies

### Web frameworks

Web frameworks allow developers to build web applications and utilise a specific back-end programming language. They often allow additional modules to easily be connected and maintained for extra features to be implemented within the application. For this project, the developer would like to use their most experienced language which is python, so the following libraries were considered:

**Pyramid**

* A lot newer that the other two, so may have less technical support and less plugin support
* Intimidating with the sheer number of setup options available to the user

**Django**

* Targeted toward large scale projects
* Is the oldest framework of the three and as a result, has a lot more user feedback
* The documentation is extremely detailed

**Flask**

* The fastest to setup and start prototyping with, can be up and running with 7 lines of code
* Targeted toward smaller projects

The framework utilised will ultimately be the one that satisfies the requirements and appeals to the developer.

### Application programming interface (API)

As mentioned above in section 3.3, an API is used to interface and retrieve data directly from a web application without exposing access to the code behind it. An API will also have to be created to interact with the data stored in the database that is selected. This will enable the front-end web pages to request data from an intermediary point instead of calling directly from the database, which is an insecure process.

Python web frameworks all provide methods that enable API routes to be constructed alongside templates that can render the data returned from the database, so this should not be a problem.

### Grid systems

**Bootstrap**

Grid systems are extremely useful for the alignment and positioning of content on a web page. Bootstrap has been around for a long time now and was originally developed by Twitter. It is an open source tool that allows developers to build websites and web applications with a responsive grid system, that transforms and adapts to fit the resolution of the device that is being used to view the content on (Otto, 2018).

Utilising Bootstrap would allow for the front-end user experience to incorporate a set well-organised and laid out styles. The documentation is also extremely detailed, which is exactly what the developer is after in this case to ensure that the front-end can be built with ease leaving more time on the core features of the project.

### Autocomplete/suggestion engine

**Typeahead JS and Bloodhound**

Lastly, an autocomplete engine will be required for the searching of films within the application. Typeadhead.js is an autocomplete library by Twitter, that combines the user interface (UI) view and a suggestion engine, Bloodhound to make informed suggestions to users (GitHub, 2018). The suggestion engine, Bloodhound, will be perfect for providing efficient lookups on the film data provided and offers additional features such as prefetching, which allows the browser to cache subsets of data so that the suggestions can be returned to the user at a much faster rate (GitHub, 2018).

# Solution Approach

In this section, a solution will be clearly outlined and influenced heavily by the research that was conducted in the literature review (section 3) above. Each individual section defined in the same section above will be briefly summarised and end with a conclusive approach to assist with the development of the solution.

## Similar project takeaways

The academic paper found during the research phase provided a lot of insight into how a project of this calibre could be tackled. Principally the way in which data will be acquired. This process should be one of the simplest phases of the project, so spending too much time retrieving data in a format that must be structured manually will not suit the project timeline. Secondly, ensuring to utilise a viable spread of different film attributes should aid with the overall accuracy of the model and reduce any biases toward specific values.

## Data sources

Utilising the results from the research carried out in section 3.3, the data source that will be best suited for this project is TMDb API. This option provides the means to interface with the required data via an extremely well documented API, it does not limit the number of queries that can be executed on a daily basis and provides a very active forum that can be used to ask any questions that may arise during the development of the project.

Some of the other sources also had their advantages such as the OMDb, which provided additional metadata that the other sources did not. If these additional attributes were required at a later date, making the switch to this API would not be a problem.

## Machine learning technologies

For the machine learning implementation, based on the dataset that will be acquired from the data source selected above a supervised learning technique will be the most well-matched option for the problem presented. A continuous user rating value is what the model will attempt to predict, so a regression algorithm will need to be implemented here.

Following on, scikit-learn offers a variety of regression methods but a decision tree regressor would benefit the problem and dataset well. This type of algorithm is able to handle data features that are both categorical and numerical in nature such as, genres and budget respectively (Scikit-learn, 2018).

## Feature selection

The questionnaire completed as part of the research was helpful at identifying what the general public believed were the most important film attributes when deciding whether or not to see a film. Still, the results only influenced the overall feature selection. The features that are to be used for model training will be:

* genres
* budget
* runtime
* trailer view count
* trailer like count
* trailer dislike count
* user rating

These are the values that can be used as the inputs to the regression algorithm selected and will not require an extensive amount of pre-processing.

## Back-end technologies

Mainly due to time constraints, the most viable solution here for the back-end database technology is a RDBMS utilising SQL and as the developer is most familiar with MySQL, this will be the RDMBS of choice. A NoSQL solution could be the preferred option in the future, especially with Elasticsearch handling the querying of data but at this time the learning curve for these technologies do not outweigh the end results.

## Front-end technologies

The most beginner-friendly python web framework researched appeared to be Flask. This enabled an application to be prototyped in 7 lines of code which is something that is useful when trying to test different functions for unit testing for example, as quickly and as efficiently as possible. The framework also provides a lot of useful features such as templating, whereby HTML pages can be created and used directly within the Flask application and python variables are able to be passed straight into the HTML syntax (Flask, 2018).

For the main web page development on the front-end, a grid system provides a lot of advantageous options for laying out the structure of the pages and allowing for predefined styling templates to be utilised. Bootstrap is the best option here, primarily as the developer has experience with this particular framework. As for any other additions to the front-end, libraries that deliver additional functionality and interactivity to the overall user experience will be added as and when they are required.

Lastly, providing autocomplete and suggestion functionality to users is a key part of this application so that they can locate a specific film in seconds, with ease. To accomplish this task, a UI view and suggestion engine bundle is required and the best choice from the research conducted was typeahead.js and the Bloodhound engine. These tools will be paramount in accurately indexing and tokenising custom search queries to the API so that relevant results can be displayed back to the user.

## Chosen solution

Based on the sections listed above the chosen solution will utilise TMDb for the acquisition of data, a MySQL RDBMS to store the film metadata retrieved from the API into the relevant tables, the features that have been selected (genres, budget, runtime, trailer view, like and dislike counts and user rating) will provide a decent spread of attributes for the training of the model, a supervised machine learning technique will be implemented in the form of a decision tree regression algorithm and Flask will be used as the web framework to assist with building the application and creating the main API routes to interface with the application database.

In addition to the main tool and techniques outlined above, GitHub will also serve as the main git repository providing the ability to commit any code written for the application to a remote server, which in turn also acts as a form of redundancy in the case of data loss from hardware failure or by other means. GitHub provides a concise and well-organised user experience that enables the developer to quickly locate any commit that has been made (GitHub, 2018), which is useful if a new feature is added through the creation of a new function for example but in turn renders an existing function non-operational. This is carried out through the use of the *diff* tool that renders code files side-by-side and highlights the parts of the code that have either been added or removed. However, to ensure that any code or documents created as part of the development process the repository utilised will be private and only made public after the submission of the report.

A lot of different methods, techniques and technologies could be applied to this project, but the solution outlined above will satisfy the problem articulated in section 2.1.

# Design and Implementation

In this section of the report, the development lifecycle of the application will be covered revealing where the application was installed and run from, what actual techniques were used to acquire the data, how the data was then pre-processed, how the regressor algorithm was implemented, how the API routes were generated and how the front-end design and user experience was handled.

## Environment configuration

When it came to configuring the best environment for this application a variety of options were tested before landing on the most viable option.

### Raspberry pi

Initially, it was decided to setup the project on a raspberry pi. This was primarily due to the fact that, a raspberry pi is lightweight and a fresh installation of Raspbian OS can be configured in next to no time. It also has a very small footprint, so leaving the pi running on a 24/7 basis would not be a problem when compared with leaving my desktop machine on all of the time.

In order to connect to the pi, a remote desktop protocol (RDP) service called *XRDP* was installed and would allow a remote machine to connect to it providing the correct credentials. Setting up a MySQL database is also extremely easy and can be completed by entering the following command into the terminal:

* **sudo apt-get install mysql-server**

After following the setup options such as, creating an administrator user, MySQL had been successfully configured and would now be able to have databases created through its interface.

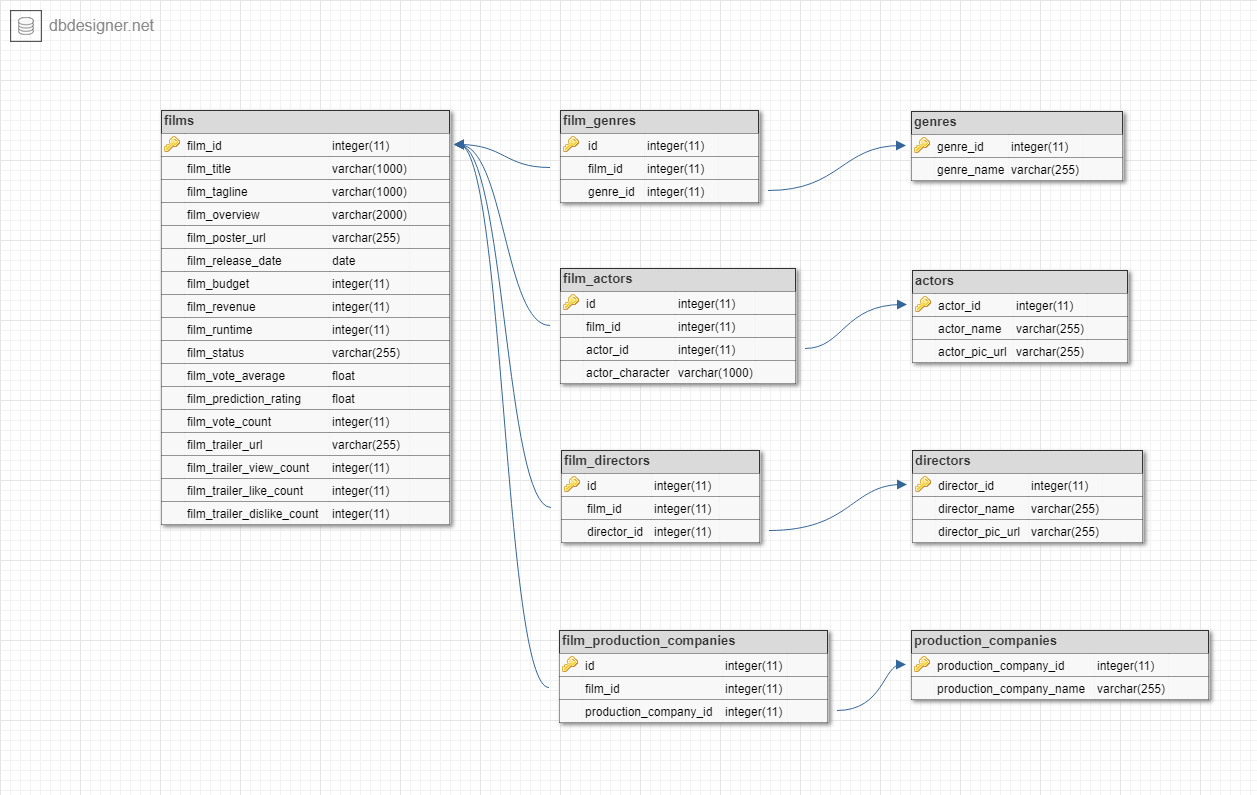
At this stage it was an appropriate time to consider database design. It was already clear that from the solution approach, TMDb would be the data source used which returns data through JSON responses. So, the database had to be built around the structure of the JSON object.

Figure - Final database design

The database was designed using a free online database scheme designer called DbDesigner (DbDesigner, 2018). The final result can be seen in Figure 3 above, illustrating the main tables that would be used and how they would link together. The database had to be normalised and to accomplish this task, the relationships between the tables had to be established. Each film entry for example, can have multiple genres associated to it and since table rows in SQL cannot contain multiple values, they would need to be stored in a separate table. To achieve this in relational databases an *association table* is required, which will intersect between this many-to-many relationship link breaking it into what is effectively two separate many-to-one relationships instead. The same process was carried out for the actor, director and production company tables.

DbDesigner also allows for the schema to be exported to a SQL file that can then be imported into the MySQL database.

For continuity, even though this configuration option was used periods of the data acquisition phase later on, because it proved to be too restrictive and resulted in quite poor performance when connecting to and querying the database the application was later moved to a HostPresto web server.

### HostPresto web host

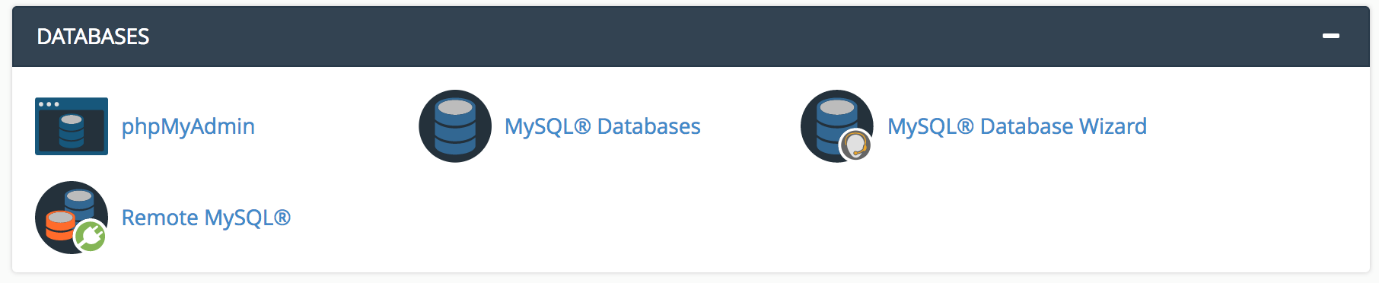
HostPresto is a website hosting company and at the time of writing this, hosted the developer’s personal website. The hosting options provided access to two separate databases, so it was simple to move the existing database from the raspberry pi to the web server. HostPresto provide users with the ability to access their web server configurations through cPanel, which is a control panel/dashboard for interfacing with the server configuration (cPanel, Inc, 2018). Through cPanel, there are certain database options like the ones shown in Figure 4 such as, the ability to identify the hosts that are allowed to connect to a MySQL database stored on the web server remotely.

Figure - cPanel database options

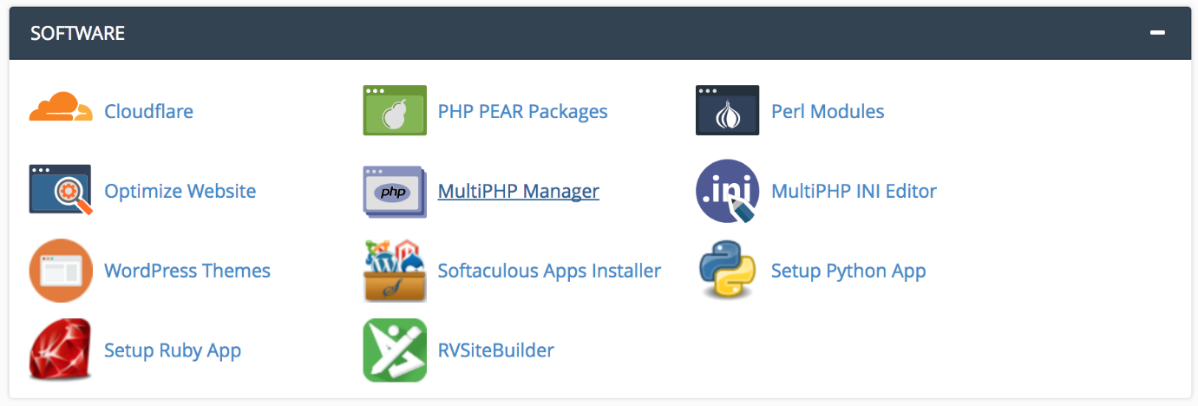
It also provides the capability to setup and configure python applications directly on the web server (see Figure 5 below). This will be useful later on when implementing the Flask web framework.

Figure - cPanel setup python application

But again, for continuity this option was not where the application ended up again mainly due to performance issues with remotely accessing the database. This is a server hardware problem as opposed to an internet connection one. To resolve this, the application in its entirety was relocated to a DigitalOcean Droplet.

### DigitalOcean Droplet

DigitalOcean is a popular platform for setting up and configuring scalable server instances as quickly as possible (DigitalOcean, 2018). These instances are often referred to as *droplets*. Through the GitHub student developer pack (GitHub Education, 2018) it is possible to obtain $50 worth of DigitalOcean credit, which roughly translates to one month of free hosting on their platform depending on the power (amount of memory and number of virtual central processing units (VCPUs)) of the virtual machine (VM) selected.

Configuring the droplet was super easy through the use of PuTTY, which is a terminal emulator that can communicate to a server via the Secure Shell (SSH) protocol (PuTTY, 2018) and also by utilising the development guides that DigitalOcean offer (DigitalOcean, 2018). FileZilla is another tool used and is a free file transfer protocol (FTP) client (FileZilla, 2018), for interacting with the files stored on a server remotely, so all of the Flask application files and folders from the web server could simply be transferred to the droplet.

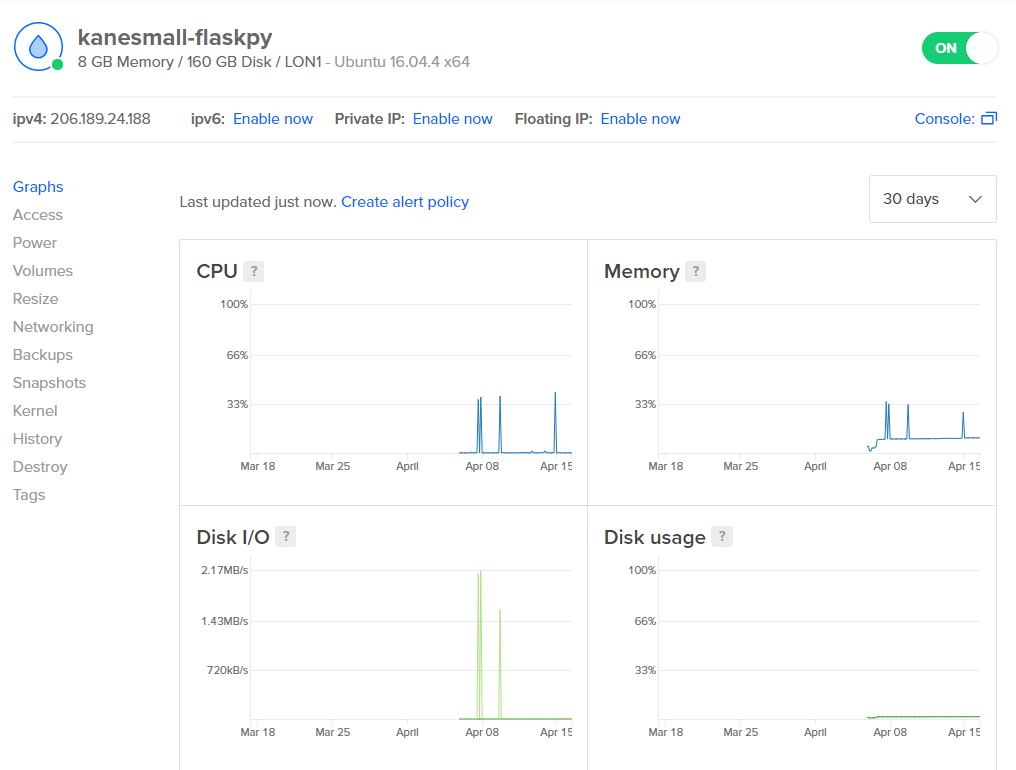
As far as development environments go, DigitalOcean droplets provide the developer with a lot more freedom to configure the VM to their specific requirements unlike a web host and also run on much more reliable, robust and powerful hardware than web hosts and/or a raspberry pi. The management console as illustrated below in Figure 6 shows just how many options the user has, especially when it comes to monitoring and is handy for keeping an eye on both the top processes running by total Central Processing Unit (CPU) or memory usage.

Figure - DigitalOcean configuration and management console

This is where the application currently resides and today and is in some ways a step closer to future-proofing the application, such as having an increase in the number of users who use the platform.

## Data acquisition

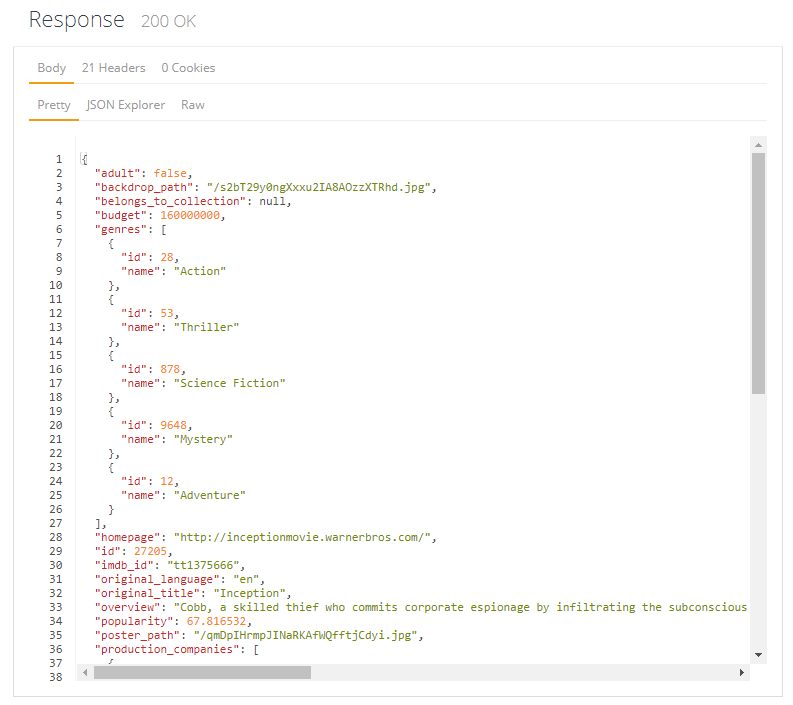
As outlined in section 4.7, TMDb is the data source being used to acquire the film metadata. The data is returned from the API via an HTTP response and formatted as JSON. Figure 7 below provides an example response snippet for the *GET /movie/{movie\_id}* route. It also illustrates the fact that each entry is stored in a key/value pair format, so accessing the specific data required would be possible using python and a few libraries.

Figure - Example JSON response snippet

### Libraries utilised

The following libraries illustrated in Table 3 were used for the data acquisition phase.

|  |  |
| --- | --- |
| Library | Usage |
| requests | Handling the HTTP GET requests to TMDb API |
| pymysql | Establishing a connection to a MySQL database from within python |
| progressbar | Providing a visual aid for time-consuming operations, to monitor progress |
| time | To handle sleeping executions to satisfy TMDb API rate limiting |

Table - Libraries used for data acquisition

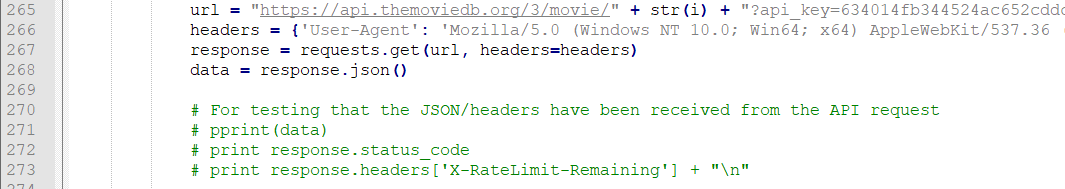
The main library here is the *requests* library (Docs.python, 2018), which allows python to communicate with web pages. An early example of this library in use can be seen in Figure 8 below.

Figure - Python request example

To explain this in more detail, a Uniform Resource Locator (URL) is stored in a variable and then it is passed to the requests library’s *get()* function which performs an HTTP GET request to the given URL. This data is stored in a variable so that it can be manipulated with the request library’s *json()* decoder function, which returns the data in a format that python can understand and read. Finally, as can be seen in Figure 8, the commented out code was essentially being used for testing purposes to ensure that both the JSON response and headers were received from the API request. Manipulating the response header object ot display the *X-RateLimit-Remaining* value was very useful when determining how best to avoid exceeding the rate limit.

Essentially, to handle the rate limiting enforced by TMDb, the *time* python library was utilised to prevent the code from continuing to make requests to the API after the HTTP response code *429* had been received, which essentially lets us know that the rate limit has been reached and that requests should be attempted again after a cool down period.

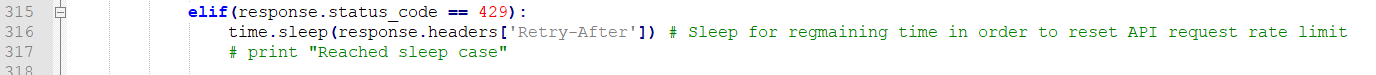


Figure - Python time library usage example

Figure 9 above, illustrates how the *time* library was implemented. As part of the library there is another function called *sleep()* which can pause the execution of the program for the specific time passed as a parameter. In this example, the parameter being passed is the value returned from the API’s response headers, as this is an exact length of time remaining as part of the cool down period.

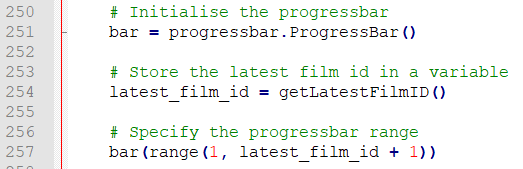
Lastly, *progressbar* is a library (PyPi, 2018) that was located as part of the research carried out and essentially enables a visual representation of how far through an iterative process the program is. Figure 10 below illustrates how the library can be setup and implemented.

Figure - Python progressbar library usage example

Firstly, a bar variable must be initialised and will be used as the point of access to the library throughout the code. Next, the progressbar’s range must be specified, this always starts at a value of *1* and goes all the way up to the latest film id (which is acquired by a function and explained in section 5.2.2) that the API has stored *+ 1* to ensure that the last id is accounted for in the progressbar display. Finally, within the main while loop, using the *progressbar* library’s *update()* function will update the visual progressbar within the terminal by the iteration value.

The libraries listed in Table 4 below were used to help with handling errors and storing them in a CSV file so that the code execution isn’t terminated to display the error.

|  |  |
| --- | --- |
| Library | Usage |
| csv | Creating and storing execution errors to CSV files |
| traceback | To allow for full stack trace errors to be printed (primarily to CSV files) |

Table - Libraries used for error handling

In order to write to a CSV file, one must be opened for writing outside of the main loop, thus to prevent a new file being created upon each iteration. Figure 11 below illustrates that a CSV file must first be created with the file mode of *wb* (writing). This can then be passed to the *csv* library’s *writer()* function that will write to the specified file, with the delimiter of a comma, hence *CSV*.

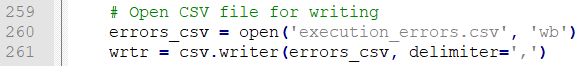
From here, during the while loop iteration by utilising a try-except method, any exceptions caught will be sent directly to the *wrtr* object created in Figure 11 with a full stack trace of the exception and the iteration number that the exception was raised. This is evident in Figure 12 below.

Figure - Python csv library usage example 1

### Functions created

Figure - Python csv library usage example 2

Table 5 below outlines a full list of the functions that were created for the data acquisition phase, along with their purpose and outputs.

|  |  |  |
| --- | --- | --- |
| Function | Purpose | Output |
| getLatestFilmID() | Querying TMDb API for the latest film id on record | Returns the latest film id |
| insertData(sqlQuery, params) | Inserting data into the database | Returns the response from the MySQL query execution |
| selectData(sqlQuery, params) | Selecting data from the database | Returns the selected data |
| insertGenres() | Querying the API for a full list of genre categories and inserting them into the database | No return method |
| getTrailerData(film\_id) | Querying the YouTube API for trailer details relating to a specific film id passed to it | Returns the trailer URL and view, like and dislike counts |
| matchGenres(film\_id, genre\_ids, genre\_names) | Checking if a new genre has already been added to the database | No return method |
| matchProdComps(film\_id, data) | Checking if a production company has already been added to the database | No return method |
| matchActors(film\_id) | Checking if an actor has already been added to the database | No return method |
| matchDirectors(film\_id) | Checking if a director has already been added to the database | No return method |
| checkIfPresent(data) | Checking if a data value is present within a JSON object | Returns either the data if present, othewise None |
| insertFilms() | Querying TMDb API for each film it has on record to return all of the metadata | No return method |

Table - Functions created for data acquisition

Due to how the functions were created, there are a lot of similarities between them. So only the following libraries will be explained in detail.

**getTrailerData() function**

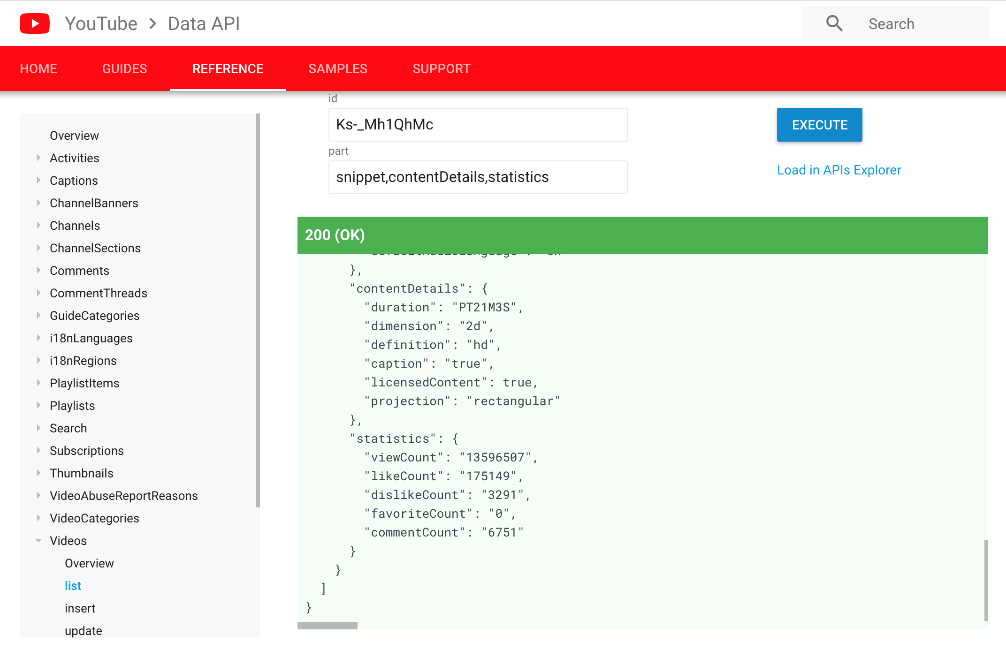
This function was implemented by taking in a specific film id and concatenating that onto TMDb API route that revolved around returning videos for a given film. The JSON response was then analysed to check whether any videos for the provided film id were on record, by using an if statement to check the status code of the HTTP request. If a video was present then the YouTube key stored within the first entry of the JSON object was extracted and using the same technique mentioned above, was concatenated onto the YouTube API route this time providing a video id to search for. Table 6 below illustrates an example response from YouTube API.

Table - Example YouTube API response

The specific data that is useful here is the *statistics* item of the JSON response. Utilising a set of nested if statements, each element that was required (viewCount, likeCount, dislikeCount) was located in the JSON object. If it was present, the data was stored into a variable, if it wasn’t then the variable corresponding to each attribute was set to None. The function could then return all of the variables acquired for this film id and perform the operation again on the next.

**matchActors() function**

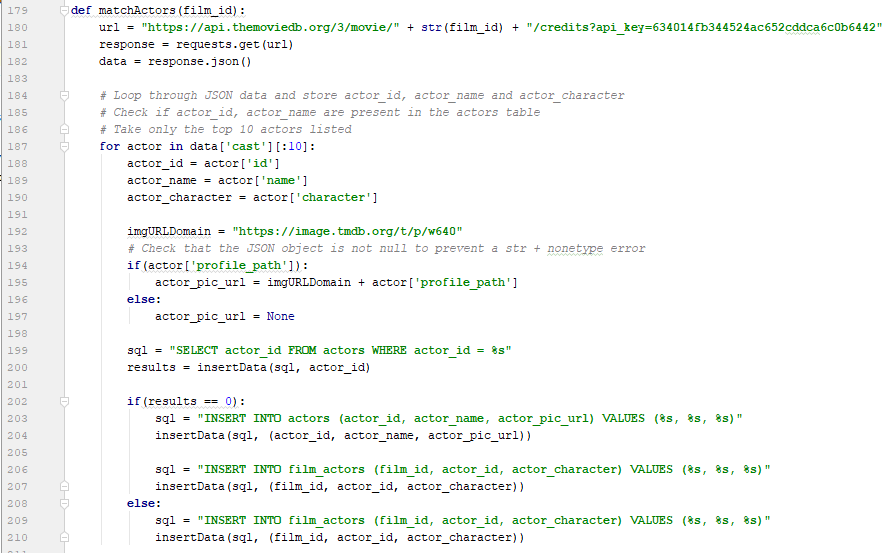
This function centres around looping through the JSON response received from TMDb API for the *GET /movie/{movie\_id}/credits* route and storing the *actor\_id*, *actor\_name, actor\_character* and *actor\_pic\_url* values. This iterative loop is limited to 10 however, so that the entire credits list is not acquired. A SQL query then checks to see if the *actor\_id* stored within the current iteration is present within the actors table in the database. If it is, then only the association table is updated with the relevant data, if it isn’t then this is a new actor that has not been added to the master actors table in the database and needs adding. A sample of the code can be seen below in Figure 13.

Figure - matchActors function

**insertFilms() function**

This is the main function that performs the actual data acquisition. It does so by performing HTTP GET requests to TMDb API within a while loop, starting at *1* and looping all the way to the latest film id TMDb has on record. For each iteration the response headers are checked to ensure that the *X-RateLimit-Remaining* value is not equal to 0 and that films with both an *adult* tag and a non-English *original\_language* tag are filtered out.

Utilising the functions listed in Table 5 above, all of the relevant film metadata is extracted from the JSON object returned and/or additional requests are made to either TMDb API or the YouTube API. Once all of the data has been acquired it is then stored into the database. Any errors that are raised during the execution are printed to a CSV file that can be accessed after the code has finished executing.

Lastly, whenever a response status code of *429* is received the code sleeps for the remaining cool down length of time that is returned in the headers under the *Retry-After* item.

## Data cleansing/pre-processing

Due to the specific input requirements for the regression algorithm that is implemented (see section 5.4), certain pre-processing techniques had to be carried out and are listed below.

### Data imputation

The first technique was a data cleaning technique whereby NULL values were handled. There are three ways to handle NULL values in machine learning training datasets, remove those entries from the dataset, ignore those entries in the dataset, or impute the entries in the dataset. The third option was carried out on the dataset that had been acquired and imputation essentially, replaces all NULL values with an arbitrary value that isn’t used elsewhere in the dataset. So, for this case *-1* was selected, as none of the film attributes being used as training features contain negative values.

With the *pandas* library, carrying out imputation of NULL values is straightforward. Simply utilise the *fillna()* function (Pandas.pydata, 2018) on an existing dataframe structure to replace all NULL values.

### One-hot encoding

In order to again, satisfy the input requirements of the regression algorithm selected the genres feature which as it stands is categorical in nature, needs to be converted into individual features. To accomplish this, the *pandas* library was utilised again and this time using the *get\_dummies()* function takes a *pandas* series object as an input and converts each of the genres into it’s own category with the prefix of *genre*.

A visual representation of how this is achieved is illustrated in both Table 7 and Table 8 below.

|  |  |  |  |
| --- | --- | --- | --- |
| Inception | Interstellar | Iron Man | … |
| Action | Adventure | Action | … |
| Thriller | Drama | Science Fiction | … |
| Science Fiction | Science Fiction | Adventure | … |
| Mystery |  |  | … |
| Adventure |  |  | … |

Table - Genres pre-one-hot encoding

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | genre\_Action | genre\_Adventure | genre\_Drama | genre\_Mystery | … |
| Inception | 1 | 1 | 0 | 1 | … |
| Interstellar | 0 | 1 | 1 | 0 | … |
| Iron Man | 1 | 1 | 0 | 0 | … |
| … | … | … | … | … | … |

Table - Genres post-one-hot encoding

Essentially, where a film normally contains a single list of all genres that are associated with it (as shown in Table 7) one-hot encoding will convert the genre categories into separate columns and place a binary-represented value under each category for each film (as shown in Table 8). So, taking *Inception* as an example from Table 8, it has the genre *Action* and therefore a value of *1* is placed under that category but, it does not have the genre *Drama* and therefore a value of *0* is placed under that category instead.

### Principal component analysis (PCA)

The last pre-processing technique implemented is a data reduction strategy called principal component analysis (PCA). PCA focuses on reducing the number of dimensions (columns) down to the satisfy the required input. It does this by stripping off the target column (the set of values that are trying to be predicted) and with the remaining features reduces the values to their main dimensions of variation.

Figure 14 below illustrates how this is achieved through actual code logic. This is primarily through the usage of the *scikit-learn* library and specifically the *pca()* function within the library (Scikit-learn, 2018).

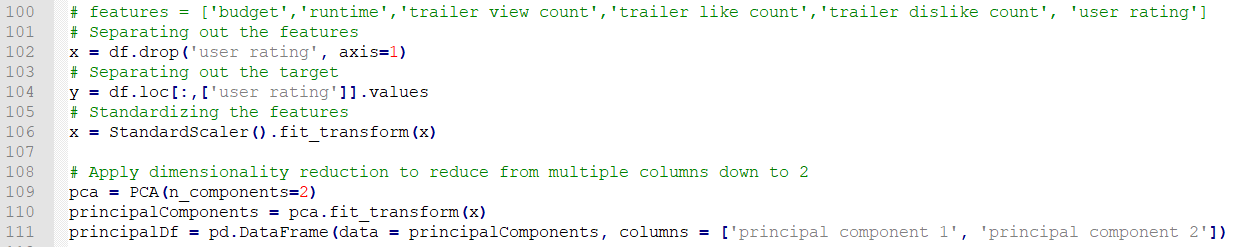
The number of components (columns) to keep must be specified using the *n\_components­* parameter and in this case was set to *2*, as this is what the regression algorithm requires from the *X* input.

Figure - PCA code logic

## Regression algorithm and predictions

### Regressor fitting

The algorithm selected in section 4.7 as part of the solution approach, was a decision tree regressor. The regressor requires an *X* and a *y* input, where the *X* is a list of lists each containing *principal\_component\_1* and ­*principal\_component\_2* values and *y* is a list of user ratings in their raw form (Scikit-learn, 2018). When implementing the *DecisionTreeRegressor()* function, the *max\_depth* parameter should be considered, as this value represents the maximum depth of the tree that the nodes should be expanded to. This value was set to *2* initially, but later removed altogether to see if this would result in a more accurate model.

### Model persistence

This is simply saving the model after it has been trained so that it can be loaded up again at a later time. This is mainly to prevent the model from having to be retrained if the accuracy of the model simply needed calculating for example. To accomplish the model persistence, the *joblib* library was used which is better suited to larger models (Scikit-learn, 2018).

## API routing

Setting up and configuring the Flask web framework was quick and easy, especially since DigitalOcean provide a development guide for serving Flask applications with Gunicorn (web server gateway) and Nginx (web server) on Ubuntu (DigitalOcean, 2018). Once configured, API routes can be created. The following table (Table 9) is a list of all API routes that were created for this application along with their purpose and outputs.

|  |  |  |
| --- | --- | --- |
| API Route | Purpose | Output |
| “/” | Rendering the index.html template to the user | Serves index.html page |
| “/trends” | Rendering the trends.html template to the user | Serves trends.html page |
| “/api/get\_trends/<int:trend\_id>” | Takes a trend id value and queries the database for trend data, storing the results in a JSON file | JSON file with trend data |
| “/api/show\_trends/<int:trend\_id>” | Loads a trend JSON file by trend id | Serves JSON to the specified route |
| “/api/search/films?%QUERY%” | Takes a query parameter passed to the route and queries the database for films similar to that term by film\_title and maps the results to a FilmSearch() object | Returns a JSON formatted FilmSearch() object |
| “/api/film/<int:film\_id>” | Takes a film\_id parameter passed to the route and queries the database for all film metadata related to that specific id and maps the results to the relevant class objects, storing instances of each into the master Film() object | Returns a combined JSON formatted Film() object |

Table - List of API routes

Each of the routes serves a different purpose as outlined in Table 9, for example <https://ksmall.me/api/search/films?query=deadpool> will return a JSON object with just the required fields to populate the results of the search field. Whereas, <https://ksmall.me/api/film/58> will return a full list of results for the film id *58* so that the metadata can then be displayed to the page.

|  |  |
| --- | --- |
| Class | Purpose |
| FilmSearch() | To map the results of a database query to a class object specific to the requirements of the search input field |
| Film() | To map an entire set of film metadata to a single class object, including instances of the other classes |
| Genre() | To map the results of a database query to a class object specific to genres |
| Actor() | To map the results of a database query to a class object specific to actors |
| Director() | To map the results of a database query to a class object specific to directors |
| Production\_Company() | To map the results of a database query to a class object specific to production companies |
| Trends() | To map the results of a database query to a class object specific to the trend id parameter that was passed to the API route |

Table - List of classes created for API routing

Each of the classes listed above represent a single database table (accept for *FilmSearch()* and *Trends()*) because, each set of data has to be acquired via a separate SQL query. So, when a single film is queried via the API each table that returns results such as the actors, or genres table will be mapped to a relevant class object and then stored within a *Film()* object. An example of how a class has been structured can be seen in Table 15 below.

## Front-end design and user experience

Figure - Example API class

Again, from the solution approach it was clear that a grid system would be useful when implementing the front-end design of the application. Bootstrap was the grid system implemented and provides an entire selection of customisable options to help provide a premium user experience. Most notably, responsiveness of web pages through the use of media queries and both column and row definitions within HTML pages themselves.

From Figure 16 - Final front-end design below, it’s clear that both the positioning of elements on the page and formatting of textual data were considered greatly. The fonts and font-sizes were selected to help the user read the information more clearly and without straining. Additional libraries such as, the *progressbar.js* (Progressbar.js, 2018) and *chart.js* (Chart.js, 2018) libraries were utilised to improve the users experience through interactive/animated methods.

Figure - Final front-end design

# Testing Verification and Validation

## Unit testing

Unit testing is a method that involves testing the individual components of an application, such as functions, classes and so on, to ensure that they carry out their intended purpose (ISTQB Exam Certification, 2018). This project however, did not conduct any automated unit testing with specialist testing software, instead the unit tests were carried out manually. This involved utilising Python’s *print* function, which provides a full traceback detailing where exactly an exception has been raised and this helps identity and locate the erroneous code as quickly as possible. A collection of the vital unit tests carried out are provided below inTable 11.

**Back-end code (Data acquisition)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Component | Expected Outcome | Actual Outcome | Action Taken |
| 1 | getLatestFilmID() function | Retrieves the latest ID from TMDb API | The latest film ID is returned from TMBd API | None |
| 2 | insertData() function | Takes 2 parameters, a SQL query and the values being inserted into the database, queries the database and returns the results | Inserts the data passed and returns the results of the execution when called | None |
| 3 | selectData() function | Takes a SQL query and selects data from the database, returning the results | Selects data from the database and returns the results | None |
| 4 | insertGenres() function | Fetches a list of genres and inserts them into the database | Fetched genres are successfully inserted into the database | None |
| 5 | getTrailerData() function | Takes in a film id, queries TMDb API for a YouTube video key, queries the YouTube API with that key and returns all trailer details related to the key | Returns the trailer URL, view, like and dislike counts as expected | None |
| 6 | matchGenres() function | Checks to make sure if genres related to the current film iteration are already in the genres table, if they aren’t add them and if they are do not | Adds genres to the genres table if not present, otherwise updates the association table with the genres for the current film iteration | None |
| 7 | matchProdComps() function | Checks to make sure if production companies related to the current film iteration are already in the production companies table, if they aren’t add them and if they are do not | Adds production companies to the production companies table if not present, otherwise updates the association table with the production companies for the current film iteration | None |
| 8 | matchActors() function | Checks to make sure if actors related to the current film iteration are already in the actors table, if they aren’t add them and if they are do not | Adds actors to the actors table if not present, otherwise updates the association table with the actors for the current film iteration | None |
| 9 | matchDirectors() function | Checks to make sure if directors related to the current film iteration are already in the directors table, if they aren’t add them and if they are do not | Adds directors to the directors table if not present, otherwise updates the association table with the directors for the current film iteration | None |
| 10 | checkIfPresent() function | Simply performs a check to see if the data being passed to it is there or not, if it is return the data, if it isn’t return *None* | Returns the data if present, returns *None* if it isn’t | None |
| 11 | insertFilms() function | The main function that runs via a while loop across a list of all film ids and acquires all of the film metadata utilising each function listed above and stores the data in the database | Executes via a while loop, displays a progressbar, sleeps to avoid exceeding rate limitations, filters out adult and foreign films, prints any errors to a csv file | None |

Table - Back-end code (Data acquisition) unit test results

**Back-end code (Regression modelling and predictions)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Component | Expected Outcome | Actual Outcome | Action Taken |
| 1 | Training data mapping | Training data csv is mapped correctly to a pandas dataframe structure | A pandas dataframe is created, populated with the csv data and with the relevant header names | None |
| 2 | Genre categories mapping | Genre categories are read from a csv file and mapped to a Python dictionary object | Genre categories successfully stored in Python dictionary | None |
| 3 | Film genres acquisition | A list of genres for each film are acquired, using the genre categories dictionary as a reference | List of genres for each film are successfully acquired and stored in a list | None |
| 4 | One-hot encoding genres | Genres are subjected to a formatting technique called one-hot encoding | Each genre is given its own category and a binary value represents a genres relation to a certain film | None |
| 5 | Handling NULL values | NULL values are imputed using the *fillna()* function within pandas | All NULL values are replaced with an arbitrary -1 value | None |
| 6 | PCA | The dataset is subjected to PCA, which reduces the number of dimensions | The number of dimensions are reduced successful and the results and the target are appended back into a single dataframe | None |
| 7 | Regression fit | The formatted dataframe is fitted to the decision tree regressor | The decision tree regressor is trained successfully | None |
| 8 | Saving the model | The model is then saved to an external file, so that it can be loaded at any point | Model is saved to an external file successfully | None |
| 9 | Regressor predictions | The regressor makes predictions on the test data | Predictions on test data are stored successfully in a list | None |
| 10 | Storing prediction ratings | Predicted ratings are inserted into the database | Predicted ratings are successfully stored in the database | None |
| 11 | Regression metrics | The explained variance and mean absolute error values are printed to the console | Regression metrics are successfully printed to the console | None |

Table - Back-end code (Regression modelling and predictions) unit test results

**Front-end code (API routing)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Component | Expected Outcome | Actual Outcome | Action Taken |
| 1 | Genre() class | The required variables are created and the appropriate setters and getters are created | Class created to store genre data as objects | None |
| 2 | Actor() class | The required variables are created and the appropriate setters and getters are created | Class created to store actor data as objects | None |
| 3 | Director() class | The required variables are created and the appropriate setters and getters are created | Class created to store director data as objects | None |
| 4 | Production\_Company() class | The required variables are created and the appropriate setters and getters are created | Class created to store production company data as objects | None |
| 5 | Film() class | The required variables are created and the appropriate setters and getters are created, including instances of the genre, actor, director and production company classes as lists | Class created to store film data as objects | None |
| 6 | Trends() class | The required variables are created and the appropriate setters and getters are created | Class created to store trend data as objects | None |
| 7 | index route (“/”) | The home route that simply returns the index.html template | Route directs users to the index.html template page | None |
| 8 | trends route (“/trends”) | The home route that simply returns the trends.html template | Route directs users to the trends.html template page | None |
| 9 | getTrends route (“/api/get\_trends/<int:trend\_id>”) | The route that queries the database, maps the returned data to the trends object and stores the data into a json file | Queries database for trends, maps results to trend object, saves object to json file | None |
| 10 | getTrends route (“/api/show\_trends/<int:trend\_id>”) | The route that loads a trend json file by the provided id and returns the results to the page as an HTTP response | Returns loaded json data to the page as an HTTP response | None |
| 11 | getFilms route (“/api/search/films”) | The route responsible for taking in an additional query parameter and querying the results returned from the database, returning an HTTP json response | Uses query parameter in SQL query, returns an HTTP json response with the results | None |
| 12 | getFilm route (“/api/film/<int:film\_id>”) | The route takes a single film id and queries the database for the required metadata | Returns an HTTP json response with the metadata related to that single film id | None |

Table - Front-end code (API routing) unit test results

**Front-end code (HTML/CSS/JS)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Component | Expected Outcome | Actual Outcome | Action Taken |
| 1 | Navigation links function as intended upon user selection | User relocated to alternate page | Alternate page loads | None |
| 2 | Navbar menu dropdown button toggles on smaller devices when clicked | Toggled button reveals dropdown menu | Nothing happens when selected | See section 7.1.1 |
| 3 | Input field produces results when a query is provided | A list of (up to 10) results are displayed to the user | Results are displayed to the user | None |
| 4 | Selecting an item from the search results updates the page content | Film data should update dynamically | Film data is updated dynamically | None |
| 5 | YouTube video trailer is playable | The YouTube video should be playable | The YouTube video can be played | None |
| 6 | YouTube controls are accessible/usable | All of the default YouTube controls should be visible and usable by the user | The user can see and select any of the default YouTube controls | None |
| 7 | Trend graphs display data to the user upon page load | When the user visits the trends page, the graphs should load | Upon visiting the trends page the trend graphs load immediately | None |
| 8 | The trend graphs should display additional data when hovered over | If a user hovers over the trend graph additional data should be displayed | When the user hovers over the trend graph additional data is displayed | None |
| 9 | selectFilmStatus() JS function | Takes in a film status value and based on that value returns the relevant status badge class | Successfully returns status badge class | None |
| 10 | Bloodhound suggestion engine assignment | The bloodhound engine is initialised, returning a mapped JS array of film attributes provided by the API | Bloodhound variable assigned to a list of film attributes from the API | None |
| 11 | User/predicted rating bar assignments | The progressbar.js variables are assigned, configuring the options to the developers requirements | Progressbar.js variables assigned | None |
| 12 | grabFilmData() JS function | Takes a film id parameter, calls the API, returns the results, formats them accordingly and passes the data to an HTML page | Formats results returned from API call and displays the data to an HTML page | None |
| 13 | Typeahead UI instantiation | The typeahead.js dropdown that will display the search results is instantiated, defining the appropriate options | typeahead.js UI object instantiated | None |
| 14 | getJSON() asynchronous functions | For the trend JS, the getJSON() functions take an API route as an input and use the returned JSON object to map the results to a newly defined chart.js object | getJSON() function calls from the API, uses JSON object response to map to chart.js object and display to an HTML page | None |

Table - Front-end code (HTML/CSS/JS) unit test results

All of the functions listed above in Table 11, Table 12, Table 13 and Table 14 completed their unit tests successfully, For Table 11, this resulted in integrating the functions tested into the *main()* method for use in the acquisition of the entire film dataset.

## Compatibility testing

It is always essential to test any web application on a variety of operating systems, devices and browsers, to ensure that it reaches and is accessible to the widest range of users. Due to the nature of the project, the front-end interface is the platform in which users will interact with, therefore only this part of the application has been compatibility tested and the prediction modelling has not. At the time of writing this, the most up-to-date versions of the operating systems and browsers were used for each individual test.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Device | Operating System | Web Browser | Pass/Fail |
| 1 | Desktop | Windows 10 | Google Chrome | Pass |
| 2 | Desktop | Windows 10 | Mozilla Firefox Quantum | Pass |
| 3 | Desktop | Windows 10 | Microsoft Edge | Pass |
| 4 | Desktop | Windows 10 | Internet Explorer 11 | Fail |
| 5 | Laptop | macOS High Sierra | Safari | Pass |
| 6 | Laptop | macOS High Sierra | Google Chrome | Pass |
| 7 | Laptop | macOS High Sierra | Mozilla Firefox | Pass |
| 8 | Mobile | Android 8.0 “Oreo” | Google Chrome | Pass |
| 9 | Mobile | Android 8.0 “Oreo” | Mozilla Firefox Focus | Pass |
| 10 | Mobile | iOS 11 | Safari | Pass |
| 11 | Mobile | iOS 11 | Google Chrome | Pass |

Table - Application compatibility test results

*Table 3* clearly identifies that almost every compatibility test passed, but happens to fail when tested on a Windows 10 desktop device running Internet Explorer 11. This test result was inspected further and the results are discussed in section 7.1.1.

## Usability testing

Usability testing revolves around making sure that the user experience when performing tasks using an application is an easy and problem-free process. Therefore, the following set of tests outlined in *Table 4* reveal the results of two external users that the developer knew and asked to test the application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test ID | Description | Pass/Fail | | Action Taken |
| **User 1** | **User 2** |
| 1 | Can you search for a film using the search input field? | Pass | Pass | None |
| 2 | Can you play the YouTube video trailer loaded for a film? | Pass | Pass | None |
| 3 | Can you full screen the YouTube video trailer loaded for a film? | Pass | Pass | None |
| 4 | Can you locate a film of your choice via the search input field with ease? | Pass | Fail | See section 7.1.1 |
| 5 | Can you hover over the trend graphs to display additional information? | Pass | Pass | None |
| 6 | Can you load a film within a reasonable amount of time? | Pass | Fail | See section 7.1.1 |
| 7 | Is the site easily accessible via the URL provided? | Pass | Pass | See section 7.1.1 |

Table - Front-end application usability test results

The results from the usability tests were fairly reasonable, with only two failures relating mainly to the querying of films. The outcome of these tests and how they were resolved are covered in detail in section 7.1.1.

## Regression metrics

For a regression algorithm, there are a few metrics that can be used to gather information relating to how accurate the model that has been trained is. The two metrics that were used for this project were, explained variance and mean absolute error.

The explained variance metric helps identify how spread apart are the prediction results from their mean value. So, a regression model like the decision tree regressor essentially looks for the inter-linking relationships between the independent variables (the features used to train the model) and the dependant variable (the target value that is being predicted). The closer the value is to 1, the more accurate the model should be. There are however other factors that can affect a model’s explained variance.

The mean absolute error on the other hand, is the average spread of errors from the dataset, the differences between the actual results and the prediction results. If these values were to be plotted on a graph whereby the X values were the actual (true) results and the Y values were the prediction results, in machine learning the mean is usually represented by 0. So, imagine a straight-line being drawn where both X and Y = 0, the prediction values closer to that line illustrate a higher level of accuracy as the mean spread of errors will be lower. A MAE that is negative (below the mean) reveals an underestimation in the predicted values, whereas a positive MAE represents an overestimation in the predicted values.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test ID | Regression Metric | Result | Training Data Used | Additional Regressor Parameters | Action Taken |
| 1 | Explained variance | 0.1535 | All training data | max\_depth = 2 | Retrain |
| 2 | Explained variance | 0.0512 | All training data, with NULL values ignored | max\_depth = 2 | Retrain |
| 3 | Explained variance | -0.5685 | All training data | max\_depth not set | None |
| 4 | Mean Absolute Error | 32.3410 | All training data | max\_depth not set | None |

Table - Regression metric results table

Both metrics assisted with helping the developer better understand the accuracy of the model that had been trained and from Table 17, it is evident that the model is overestimating the prediction values and that a lot of work needs to be carried out in improving the overall accuracy of the model. The details of the accuracy predictions are discussed in section 7.1.2.

# Discussion: Contribution and Reflection

## Discussion

The section prior to this illustrated the results of unit, compatibility and usability testing that was carried out during the development of this project. This also included, utilising regression metrics to identify the accuracy of the model selected. This section will aim to discuss the discoveries of the tests in more detail and look to fix any major flaws that were identified.

### Known issues

The testing that was conducted as part of section 6, exposed some important features of the application that needed improving, mainly to do with the usability of the application.

|  |  |  |  |
| --- | --- | --- | --- |
| Reference ID | Description | Priority | Action |
| 1 (Unit – Front-end) | On mobile devices, the navigation menu reduces to a togglable button, but this was not functioning as intended | High | Completed |
| 4 (Compatibility) | Display of film data and input field functionality not functioning on Windows 10 Desktops using Internet Explorer 11 | Low | To do |
| 4 (Usability) | Difficult to locate certain films using the input field | Medium | Completed |
| 6 (Usability) | Initial loading of the search results and autocomplete/suggestion features are sometimes slow | Low | Completed |
| 7 (Usability) | The site does not have secure communication via HTTPS | High | Completed |

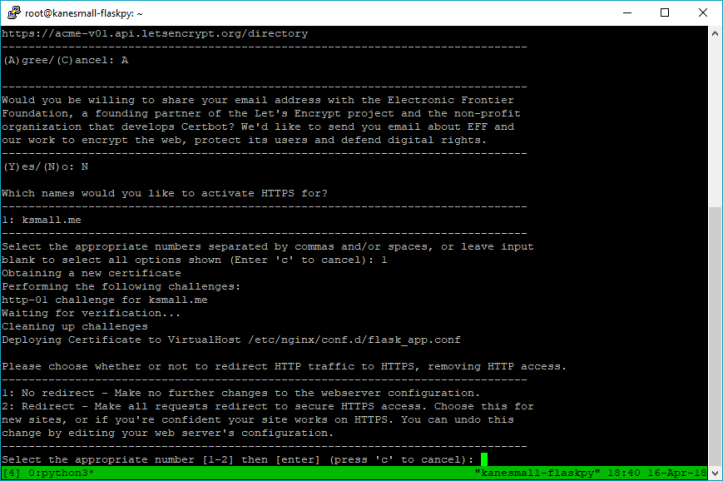
Table - Known issues table

When it came to correcting the issues referenced in *Table 5* above, the higher priority tasks were resolved first. This is because, these issues are more likely to result in missing or reduced application functionality or potential security risks.

To begin with, the first issue related to being able to access the menu items (home/trend page links and search bar) on mobile devices. On mobile devices, the menu button that is supposed to toggle when click and reveal the navigation items and search bar was not functioning. This is a major problem, as it prevents mobile users from being able to use the application for its intended purpose. To rectify the issue however, was an extremely easy fix. It was a matter of making sure that the *div* element containing the menu items had an id value that matched the *data-target­* element attached to the togglable button.

Upon configuring the Flask application on a DigitalOcean droplet, a valid certificate had not been acquired from a Certificate Authority (CA) at that time. However, there are plenty of sources that offer free certificates, the most notable being Let’s Encrypt (Let's Encrypt, 2018). As this application does have shell access to the server over SSH, an automated certificate issuance and installation tool can be used called Certbot (Certbot, 2018). The instructions are straight forward to follow and allowed for an extremely smooth installation process.

Figure 17 below illustrates a selection of the type of questions that were asked during the setup and installation process. Most notably the last question, which asks whether or not all HTTP traffic should be redirected to HTTPS, removing HTTP access as a result. Option 2 was selected and this was for two reasons. The first being that, HTTPS access is more secure and should be used across the web. In July 2018, Chrome will be one of the first browsers to mark non-HTTP as *not secure* with the release of Chrome 68 (Google, 2018). The second reason behind implementing HTTPS access is due to the way in which in the API handles credentials that are used to connect to the database. When passed in plaintext and using an HTTP connection, the credentials would be readily available for any experienced hacker to grab them in transit. However, by implementing an HTTPS connection and forcing all users to do the same, it is not possible to access the credentials in transit thus implementing basic API authentication.

Figure - Certbot installation via PuTTY

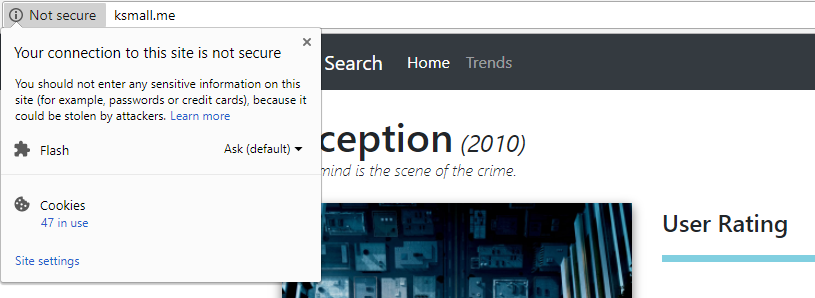
The browser that the user is viewing the application will update from saying that the connection is not secure to the site as illustrated in Figure 18.

Figure - HTTP (Not secure) connection

To a message stating that the certificate for this site is valid and that the connection *is* secure. The URL will now also include the HTTPS protocol and is illustrated in Figure 19 below.

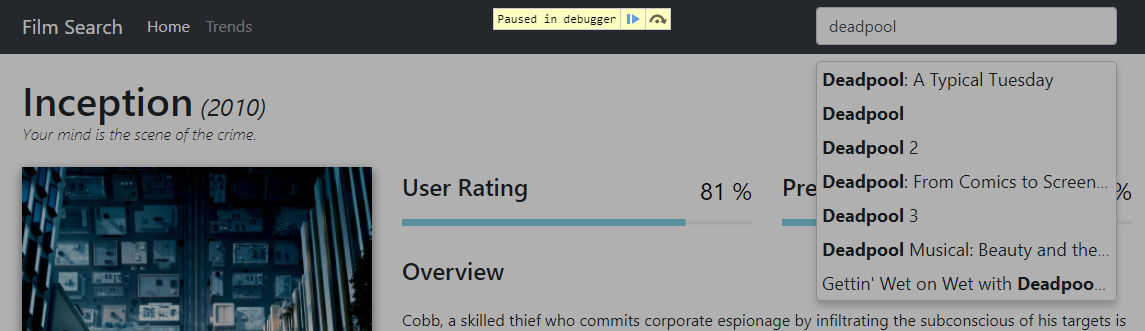
Next up was the issue relating to locating certain films using the input field. The user who marked this usability test as a fail informed the developer that this was due to the way in which the results were returned after a query had been entered making it difficult to locate certain films. To rectify this issue, additional film metadata was added to the query results so that users could better pinpoint films with similar titles, or films with the same name due to a rebooting of the series for example. This is illustrated by the comparison of Figure 20 and Figure 21.

Figure - HTTPS (Secure) connection

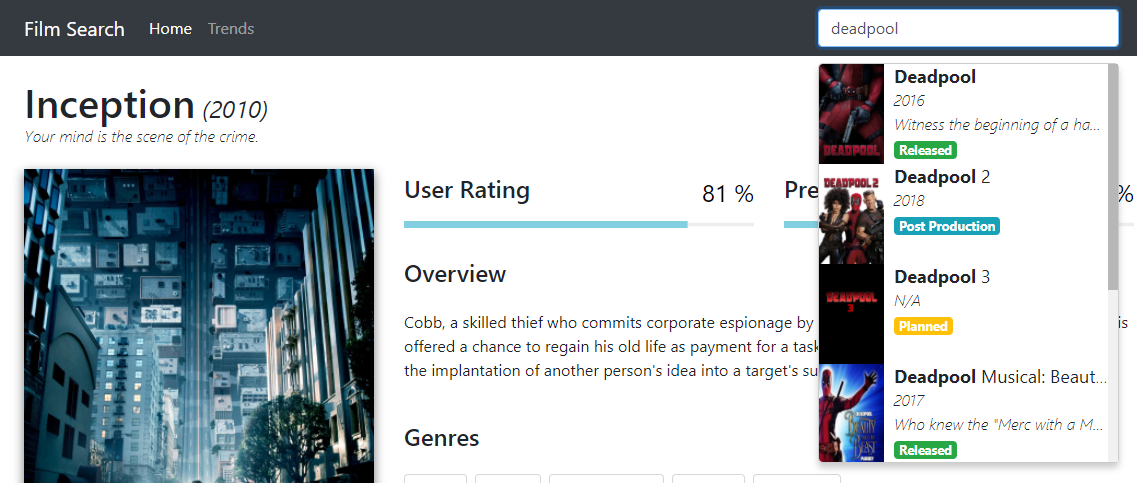
Figure - Query results (old)

Figure - Query results (new)

Next, the issue relating to the load time of the query results which was only documented by one of the users as a fail. The user in question stated to the developer that it was not an issue all of the time, but did occur on a rather regular basis. They also mentioned that the load times were already more than reasonable, but that an increase in speed would result in a much more fluid experience for the user.

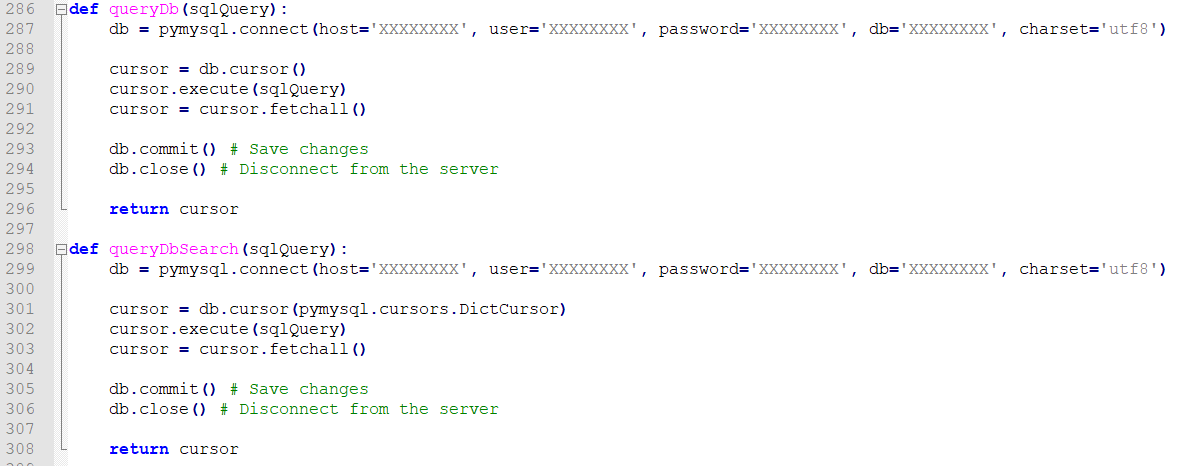
This issue required a bit more attention than the last, mainly due to the way in which the search results were queried via the API. But after a bit of research, a pymysql has a class (pymysql.cursors.DictCursor) that can be passed to the *cursor()* object as a parameter allowing the cursor to return the results as a python dictionary (pymysql.readthedocs, 2018).

Figure - Database query function comparison

As illustrated by Figure 22 above, by duplicating the already existing database connection function and altering the cursor parameters, it would now be possible to utilise this function within the correct API route. Now, as part of the */api/search/films* route, because the *queryDbSearch()* function will now return a python dictionary of results, this can simply be serialised using jsonpickle and returned as a response to the web page. As evident in Figure 23, the code that has been commented out was responsible for mapping each element returned from the SQL query to a python class. This was the part of the code that was rather expensive and was causing slow load times for the query results.

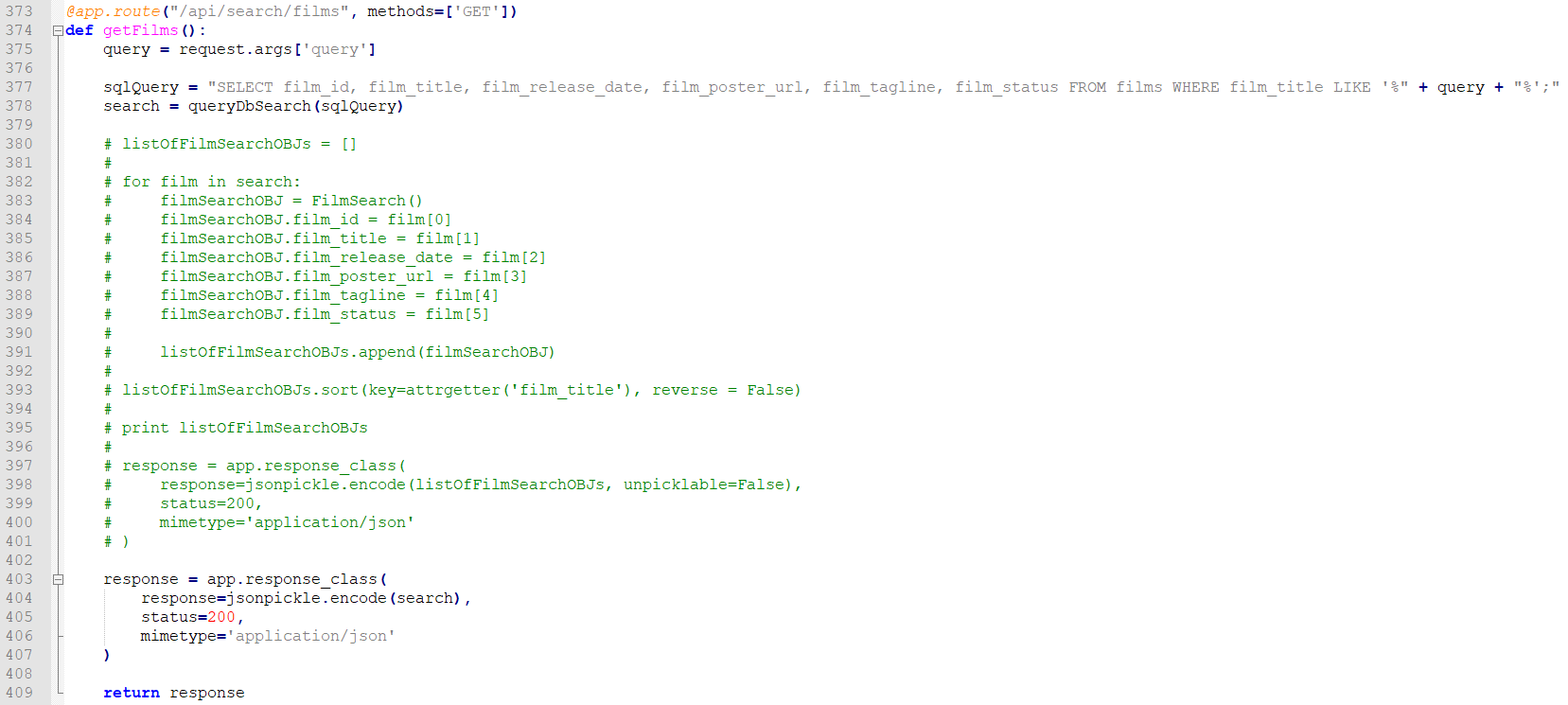
Lastly, there is the issue relating to the displaying of data and input field querying not functioning as intended whilst using a Windows 10 desktop running Internet Explorer 11. From a cursory overview within Internet Explorer 11 using the in-built developer tools, it appeared that the *progressbar.js* library being utilised to display the animated user and predicted rating progress bars, is not supported for the Internet Explorer 11 browser. Due to the fact that it would have taken a while to locate a replacement library for generating animated progress bars that was compatible with Internet Explorer, this issue has been left with the status of *To-do*. It was also more important to ensure that support/compatibility for the most popular browsers was the main priority and as can be seen on *statcounter* (statcounter, 2018), Internet Explorer is only used by 3.13% of total users, whereas Chrome, Safari and Firefox total of 77.88% of users worldwide.

Figure - /api/search/films route

### Prediction accuracy

The results shown are difficult to make assumptions on by themselves, but it is clear that the model trained isn’t exceedingly accurate and does overestimate the prediction values. This could be a result of a number of factors and when it comes to machine learning in general, there are so many factors to consider that may have influenced the end results.

**Model selection**

When it comes to selecting the appropriate model for a particular dataset, there isn’t necessarily a full proof method, lots of different variables affect the decision. Scikit-learn provide a *cheat sheet* (Appendix 4: Scikit-learn algorithm cheat sheet), which can be used as a starting point for understanding which types are better suited toward specific datasets. However, this should not be the only tool used when selecting a model.

Even today, one of the best approaches to selecting the best model for your dataset is trial and error (Sief, 2018). Implementing, training and then comparing the prediction accuracy results of each will give a better understanding of the factors that improve accuracy. But, due to time constraints, training different models was simply not possible considering how long it takes to do just that.

That is why, in the future it would be best to try implementing different models and continually attempting to improve feature selection and pre-processing techniques to learn which methods and which combination of methods, result in the most accurate results. A potential neural network solution could even be something to consider, as they are exceedingly effective at modelling highly complex non-linear relationships (Sief, 2018) and that is what this dataset has a lot of. More details on the future improvements that have been considered however, can be found in section 9.2.1.

**Feature selection**

Another reason for the results attained could simply be a consequence of the features that have been selected. The features, budget, runtime, trailer view/like/dislike counts and user rating, might just not have been the right attributes for making predictions on the user ratings. Given more time and a better understanding of feature selection and pre-processing, the model would have also been trained using additional metadata values as features such as, actors, directors, production companies and other film-related attributes that were not acquired for this project. Again, more information on additional feature selection can be found in sections 9.2.5 and 9.2.8.

**Summary**

Ultimately, this isn’t the end of the project even after submission. There are a variety of additional methods to try, new features to acquire and use to train models and even entirely different machine learning solutions to learn about. The main objective of, implementing a regression algorithm to train a model that could make predictions on previously unseen data has been accomplished. It is now the responsibility of the developer to improve the accuracy of those predictions over time, utilising the infrastructure that has been developed over the course of this projects lifetime.

### Application limitations

There are definitely still limitations related to this application, primarily with the absence of interactivity whereby users can create custom search parameters in order to receive a prediction output. But, one could argue that this was never the intended purpose of the project and is simply an idea that has presented itself over the course and nearer to the end of the project’s lifetime. Section 9.2 addresses in detail, a number of improvements that could be made given more time and if the project were to be continued even after university.

## Reflection

As a final year student completing his/her degree, you are required to undertake a final year project that must be completed throughout the last year and alongside other pieces of coursework. I will be looking at how I personally felt at each point during the project’s lifetime and ultimately, what I learned from this experience and how I could improve if given a second chance.

My initial feelings for the project were ambivalent, mainly due to the sheer size of the undertaking and because we had to continually work on the project throughout the year alongside multiple pieces of coursework being set by each module lecturer. However, once I had settled on a project idea and started working on it, because I was working on something that I find of interest to me I think this helped in viewing the project as an enjoyable set of tasks as opposed to mandatory *work*.

One of the worst parts of the project for me was before the year had even started. A few months before beginning our final year, we were tasked with either selecting a pre-defined project from a list that was provided or coming up with our own idea and submitting a PID for it. For my placement I worked at a cyber security company and so for the best part of that year I had been thinking of cyber security related projects that could have been interesting to complete. However, I eventually decided that cyber security was no longer a field I wished to pursue, as I wasn’t enjoying it as much as I was in the beginning. Therefore, completing a project related to that field no longer made sense. So, this left me with a few weeks of having to think of an entirely new project.

The stress and anxiety that the final year project caused me was severe and at times, very overwhelming. Having to juggle separate pieces of coursework for different modules I found challenging enough in the first two years of university, but having this additional project weighing down on you and also knowing that it’s worth a third of the entire year in terms of marks was again, at times too much to handle.

That being said, I have learned with the help of my incredible Mother, certain coping techniques and strategies to alleviate some of the pressures of work similar to this. So, there were definitely good experiences that I had whilst completing my project. I thoroughly enjoyed improving my programming ability with every piece of code I had to write. I decided to build my application in Python, because I believe that I have the most experience with this language over any other that I have used in the past, yet after having completed this project I feel as if I’ve learned so much more about the language then I could have imagined.

Learning how to implement machine learning has been a very interesting journey. I didn’t quite anticipate the level of difficulty that using these tools would provide but learning how to implement them as well as *why* to implement a specific algorithm for example has been really enjoyable. I have only just touched the surface when it comes to machine learning but, with so many industries beginning to adopt this technology it could be a very interesting field to break into after university.

In conclusion, from my experience I now know that it is vital to carry out research first and at the earliest point in development as possible. Before you can tackle any task, you need to have all of the tools, including the knowledge of how to implement *with* those tools.

Looking at my personal experiences and my unique style of working I would say that, industry experts spend months and months, sometimes even years, attempting to perfect machine learning models. I can’t expect to have achieved outstanding results on my first attempt and that shouldn’t hinder me from trying to get better results, trying different models and comparing accuracy scores. Just because this is the end of my project from an academic-perspective doesn’t mean it *has* to be the end of my journey of continuing to expand my knowledge in this vast field.

I think, if I were to undertake this or a similar project again I would try to focus on asking more questions. Making sure that I came prepared to each supervisor meeting with detailed questions, where we could then have discussions about. I would also ensure to utilise strategies to help me, such as recording feedback my supervisor gives me so that I don’t forget what has been said by the time I get home. I need to focus on my worst qualities first next time, such as time management and organisation. If I create a plan of action of *how* I am going to tackle these obstructions, it will be easier when it comes to starting.

# Social, Legal, Health and Safety and Ethical Issues

## Social, legal and ethical issues

When constructing the PID initially no social, legal or ethical issues were identified. However, upon reflection there are a couple of social and ethical issues that could in fact relate to this project.

Firstly, one of the social issues concerning the usage of the application itself relates to users who speak another native language other than English and would therefore not be able to understand the content that is being displayed to them. A simple solution to this problem would be to implement some form of translation feature, possibly with the help of an existing library that supports hundreds of languages and give the user the option to change the language manually. However, a more streamlined approach would be to automatically detect the user’s location via their browser session and to convert the language for them. But, having the manual option present would still be required for users that are visiting the application whilst using a Virtual Private Network (VPN).

Secondly, an additional social issue which again, concerns the usage of the application itself, would be the fact that the application does not cater toward users with disabilities such as visual impairments. There are a variety of solutions of for this type of issue including but not limited to, ensuring that all *<img>* tags are making proper use of the ‘alt’ tag that describes what an image (often used for screen readers), ensuring that the web application is fully-operable via *­just* the keyboard, providing users with the control over text sizing for all elements and having an option for users to be able to alter the contrast levels of the application or include specific colour-blindness modes.

Finally, an ethical implication of this project could be that if the prediction accuracy of the ratings was high enough some could argue that there would no longer be a need for film critics, thus putting a lot of people out of work. A second ethical issue, that follows on closely from the last relates to the emotional side of films. A lot of people believe that computers are not capable of replicating human emotion and therefore, they will never be as accurate as a human critic. There are often films that do draw heavily on a viewer’s emotion, regardless of the known variables such as the genres, budget, trailer view count, etc and therefore it is hard to envisage a scenario in which a computer would consider these parameters when formulating predictions. Such film’s that are considered *cult-classics* would also be extremely difficult to predict accurately for, since a lot of the time these films are liked by viewers regardless of critic ratings/reviews, or factual metadata such as the actors in the film.

## Health and safety risks

There were a few risks that were identified during the creation of the PID relating to the work area and these were eye strain, hardware failure and repetitive strain injury. Eye strain was reduced by taking regular breaks and by utilising an application call *f.lux* (f.lux, 2018), which reduces blue light exposure. To reduce the risk of hardware failure, all documents and code were backed up to a private GitHub repository as well as Google Drive. Lastly, the risk of repetitive strain injury was reduced by taking regular breaks, using an ergonomic chair with lumbar support and height adjustments and by simply adjusting the workstation to appropriately conform to the user’s requirements.

# Conclusion and Future Improvements

## Conclusion

To conclude the report and the project as a whole, the overarching objective to produce an application that makes predictions on user ratings for unreleased films was a success. In order to carry out the objectives, research had to be conducted on the technique of regression modelling to implement, as well as the libraries that would accommodate machine learning algorithms into a python coding environment, various underlying technologies/frameworks were also researched in order to determine how the application could be built and accessed by the end user.

By completing the sections within this report, the main objectives have been achieved. Data has been acquired from a viable source that provided all of the metadata required, this data was then stored in an appropriate normalised database structure, a suitable python framework was selected that would form the base of the entire application, the application itself was configured on a robust and reliable DigitalOcean Droplet which at the same time future-proofed the application and lastly; the features selected were influenced by the results of a questionnaire.

For the data cleansing/pre-processing phase, three separate pre-processing techniques were used to properly transform the data into a format that the regression algorithm could handle. Imputation of NULL values, replaced all NULL entries throughout the dataset with the arbitrary value of -1, one-hot encoding converted the categorial genres feature into separate unique fields that were then represented by binary values and PCA reduced the number of dimensions from a total of 42 down to 2, resulting in an appropriate X input value for the regression algorithm. The regression algorithm itself was selected after research had been conducted, with a decision tree learner being chosen as part of the scikit-learn library for its benefits when working with large and complex datasets that are attempting to predict a numerical output. This was then trained and then used to make predictions, by learning from the scikit-learn library and making reasonable adjustments when needed.

For a first attempt at implementing any form of machine learning logic, the developer has managed to produce a working regression model and that in its own right is a success. From these initial results and using what has been learned throughout the entire project, the results can be improved upon as the project transitions past this deadline.

Researching the techniques used to implement a strong and robust API, along with the routes to access it was a challenging process, but like the rest of the objectives this one also resulted in success. The API can serve data from the database to the front-end web pages, either by predefined identifiers, or by unique queries. It was also secured with basic authentication as a result of carrying out useful usability testing, which makes it more difficult for attackers to access the credentials being transmitted for use when interfacing with the database.

Lastly, the front-end design and user experience objective has been satisfied. This in-part was due to the use of the bootstrap grid system, which handles different user configurations well, such as different resolutions and helps with making the application responsive. But, also thanks to the layout and positioning of the elements on the page, making sure to house the most important information at the top of the page and sticking with this method for the rest of the metadata.

As a final point, there is a lot to take away from this project, not only from a technical stand point but also from a soft skill-perspective. It has enabled the developer to improve their abilities on all fronts, from programming efficiency and code management, to organisation and time management; all of which can be utilised moving forward into a career within the field of computer science.

## Future Improvements

A lot of time and thought has gone into how this project could be improved if more time had been allocated, or simply as the project continues to grow past a university project.

### Improving prediction model accuracy

One of the most important considerations is improving the accuracy of the prediction model. This is the core feature of the application and being able to specify to your users that this application is capable of predicting user ratings for unreleased films, to a certain high-level degree of accuracy, would encourage a lot more people to use the platform.

In machine learning it is common practice to implement multiple algorithms to see which is best suited toward your specific dataset. So, with more time it would be possible to train different models and compare the accuracy scores (explained variance, MAE) of each to see which is the most precise at predicting user ratings.

### TV show support

Implementing TV show support as well as film, would allow for the application to have a lot more range and to appeal to a wider audience. TMDb (TMDb, 2018) already allows developers to retrieve metadata for TV shows, so by utilising the same scripts that have already been created for the film data acquisition and altering them slightly all TV show metadata could be acquired.

### Video game support

Further to the addition of TV show support, once an effective model has been trained and can produce accurate prediction ratings on a regular basis for film and TV, video game support could also be added to the application. This would again, increase the range of the audience by tapping into an ever-growing and fast-paced industry. If users and game developers alike, had the ability to predict how well a game would do, it would provide users with the knowledge of whether or not the game is worth purchasing and developers the information to decide whether or not to put time and money into creating a game that may or may not be well-received.

### Additional trend graphs

It would be advantageous to provide the user with additional trend graphs via the *trends* page, but also allow for more user interactivity by having some form of filtering option available on the existing and any future graphs. An added interactive feature could include the option to search for a specific production company for example and to show a variety of trend analytics specifically for that search query.

### External ratings

Requesting ratings from other external sources such as Rotten Tomatoes and/or Metacritic would not only allow users to see all ratings for a film in a singular location but could also help influence prediction models by providing these specific ratings as additional features. Rotten Tomatoes for example, has a *want-to see list*, which indicates how many users have added an unreleased film to their want-to see list. So, utilising this feature could help increase the accuracy of the model by revealing the percentage of users that are already interested in seeing the film.

A similar idea could be applied to both Twitter and Facebook pages whereby, users can follow or like pages that have been created for upcoming films. But, this could be taken a step further and tweets and/or Facebook comments could be analysed using sentiment analysis to determine whether the content is positive or negative and again used as additional features for training the model.

### Updater function

This is an important function to have for an application like this one, whereby on a set schedule (once a week, or once a month) a function would trigger that would be responsible for looking at all existing data in the database and comparing it against The Movie Database API to see if any new films have been added, but more importantly to check if any existing films have been updated or altered in anyway. If a film’s metadata has changed, then the function will retrieve the new data and replace the old data in the database.

Once updated or additional data has been acquired, the function will then be responsible for re-training the model to account for the new data. This will ensure that as new data enters the system, all prediction rating values are also updated to better reflect any changes in the relationships that the model had previously been trained on.

### Targeting businesses

By adding additional functionality to the application whereby allowing users to configure their own custom test cases for unreleased films, this application could be targeted toward production companies. These industry experts are always trying to find the *winning* formula for a film and one that will generate the highest amount of revenue. If they had the ability to compute different combinations of variables such as, which actors, or which director would best suit an upcoming film based on the predicted rating output, then it would be extremely beneficial and lucrative for them.

### Oscar nominees and winners

A very interesting model feature was considered toward the end of the project and as such, has been constructed into a future improvement. The idea centred around utilising Oscar nomination, winner and potentially other film award or accolade data, as an additional feature when training the model; which could have a huge advantage on the outcome of the prediction values. It would work so that, whenever a film is either nominated or wins, a unique category would then be populated for each attribute of the film. Taking this year’s Academy Awards for example, Blade Runner 2049 won an Oscar in best visual effects category (Donnelly, 2018). So, each film attribute, the genres, the actors, the director, the production company and so on; would also receive numerical value appended onto this unique category mentioned above. Each value could also be weighted differently depending on the category in which it has originated from and depending on whether the end result was simply a nomination or an actual win.

# References

Amazon Web Services, Inc, 2018. *What is NoSQL? – Amazon Web Services (AWS).* [Online]   
Available at: https://aws.amazon.com/nosql/

Bird, S., Klein, E. & Loper, E., 2009. Preface. In: *Natural Language Processing with Python: Analyzing Text with the Natural Toolkit.* s.l.:O'Reilly Media, p. ix.

Certbot, 2018. *Nginx on Ubuntu 16.04 (xenial) - Install.* [Online]   
Available at: https://certbot.eff.org/lets-encrypt/ubuntuxenial-nginx

Chart.js, 2018. *Chart.js documentation.* [Online]   
Available at: http://www.chartjs.org/docs/latest/

cPanel, Inc, 2018. *The Hosting Platform of Choice.* [Online]   
Available at: https://cpanel.com/

DbDesigner, 2018. *About DbDesigner.* [Online]   
Available at: https://dbdesigner.net/

DigitalOcean, 2018. *DigitalOcean: Cloud Computing, Simplicity at Scale.* [Online]   
Available at: https://www.digitalocean.com/

DigitalOcean, 2018. *How To Serve Flask Applications with Gunicorn and Nginx on Ubuntu 16.04 | DigitalOcean.* [Online]   
Available at: https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-16-04

Docs.python, 2018. *Requests: HTTP for Humans — Requests 2.18.4 documentation.* [Online]   
Available at: http://docs.python-requests.org/en/master/

Donnelly, J., 2018. *Oscar Winners 2018: See the Full list.* [Online]   
Available at: http://oscar.go.com/news/2018/oscar-winners-2018-see-the-full-list

Elastic, 2018. *Elasticsearch: RESTful, Distributed Search & Analytics | Elastic.* [Online]   
Available at: https://www.elastic.co/products/elasticsearch

Elastic, 2018. *Hardware | Elasticsearch: The Definitive Guide [2.x] | Elastic.* [Online]   
Available at: https://www.elastic.co/guide/en/elasticsearch/guide/current/hardware.html

f.lux, 2018. *justgetflux.* [Online]   
Available at: https://justgetflux.com/

FileZilla, 2018. *FileZilla - The free FTP solution.* [Online]   
Available at: https://filezilla-project.org/

Flask, 2018. *Step 7: The Templates — Flask Documentation (0.12).* [Online]   
Available at: http://flask.pocoo.org/docs/0.12/tutorial/templates/

GitHub - Tensorflow, 2018. *tensorflow/skflow.* [Online]   
Available at: https://github.com/tensorflow/skflow

GitHub Education, 2018. *GitHub Student Developer Pack.* [Online]   
Available at: https://education.github.com/pack

GitHub, 2018. *kanesmall | Profile.* [Online]   
Available at: https://github.com/kanesmall

GitHub, 2018. *twitter/typeahead.js.* [Online]   
Available at: https://github.com/twitter/typeahead.js

GitHub, 2018. *twitter/typeahead.js - Bloodhound.* [Online]   
Available at: https://github.com/twitter/typeahead.js/blob/master/doc/bloodhound.md

Google, 2018. *A secure web is here to stay - Google Online Security Blog.* [Online]   
Available at: https://security.googleblog.com/2018/02/a-secure-web-is-here-to-stay.html

IMDb - Datasets, 2018. *IMDb Datasets.* [Online]   
Available at: https://www.imdb.com/interfaces/

IMDb, 2018. *Batman Begins.* [Online]   
Available at: https://www.imdb.com/title/tt0372784/

IMDb, 2018. *Because of Winn-Dixie.* [Online]   
Available at: https://www.imdb.com/title/tt0317132/?ref\_=fn\_al\_tt\_1

IMDb, 2018. *IMDb | Help - Weighted Average Ratings.* [Online]   
Available at: https://help.imdb.com/article/imdb/track-movies-tv/weighted-average-ratings/GWT2DSBYVT2F25SK?ref\_=cons\_tt\_rt\_wtavg#

IMDb, 2018. *IMDb | Help - What is IMDb?.* [Online]   
Available at: https://help.imdb.com/article/imdb/general-information/what-is-imdb/G836CY29Z4SGNMK5?ref\_=helpsect\_cons\_1\_1#

ISTQB Exam Certification, 2018. *What is unit testing?.* [Online]   
Available at: http://istqbexamcertification.com/what-is-unit-testing/

Let's Encrypt, 2018. *Getting Started.* [Online]   
Available at: https://letsencrypt.org/getting-started/

MDN Web Docs, 2018. *Introduction to web APIs.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side\_web\_APIs/Introduction

MDN Web Docs, 2018. *Working with JSON.* [Online]   
Available at: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON

Metacritic, 2018. *About Us - Metacritic.* [Online]   
Available at: http://www.metacritic.com/about-metacritic

Metacritic, 2018. *How We Create the Metascore Magic - Metacritic.* [Online]   
Available at: http://www.metacritic.com/about-metascores

OMDb, 2018. *OMDb API - The Open Movie Database.* [Online]   
Available at: http://www.omdbapi.com/

Otto, M., 2018. *Bootstrap.* [Online]   
Available at: https://getbootstrap.com/

Pandas.pydata, 2018. *pandas.DataFrame.fillna — pandas 0.22.0 documentation.* [Online]   
Available at: https://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.fillna.html

Patreon, 2018. *Brian Fritz is creating The Open Movie Database API | Patreon.* [Online]   
Available at: https://www.patreon.com/omdb

Progressbar.js, 2018. *Progress bars with JavaScript.* [Online]   
Available at: https://kimmobrunfeldt.github.io/progressbar.js/

PuTTY, 2018. *Download PuTTY: latest release (0.70).* [Online]   
Available at: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

pymysql.readthedocs, 2018. *Cursor Objects - PyMySQL 0.7.2 documentation.* [Online]   
Available at: http://pymysql.readthedocs.io/en/latest/modules/cursors.html#pymysql.cursors.DictCursor

PyPi, 2018. *progressbar2.* [Online]   
Available at: https://pypi.org/project/progressbar2/

Roberts, I. & Yan, L., 2014. *Vector-based Sentiment Analysis of Movie Reviews,* s.l.: Stanford University.

Rotten Tomatoes - Developer Network, 2018. *Fandango - Welcome to the Rotten Tomatoes® Developer Network.* [Online]   
Available at: https://developer.fandango.com/rotten\_tomatoes

Rotten Tomatoes, 2018. *Batman Begins.* [Online]   
Available at: https://www.rottentomatoes.com/m/batman\_begins/

Rotten Tomatoes, 2018. *Because of Winn-Dixie.* [Online]   
Available at: https://www.rottentomatoes.com/m/because\_of\_winndixie

Rotten Tomatoes, 2018. *Rotten Tomatoes: About.* [Online]   
Available at: https://www.rottentomatoes.com/about/

Saraee, M., White, S. & Eccleston, J., 2004. A Data Mining Approach To Analysis And Prediction Of Movie Ratings. *WIT Transactions on Information and Communication Technologies,* Volume 33, p. 10.

Sas.com, 2018. *Machine Learning - What it is and why it matters.* [Online]   
Available at: https://www.sas.com/en\_gb/insights/analytics/machine-learning.html

Scikit-learn, 2018. *1.10. Decision Trees - Regression — scikit-learn 0.19.1 documentation.* [Online]   
Available at: http://scikit-learn.org/stable/modules/tree.html#regression

Scikit-learn, 2018. *1.10. Decision Trees — scikit-learn 0.19.1 documentation.* [Online]   
Available at: http://scikit-learn.org/stable/modules/tree.html#tree

Scikit-learn, 2018. *3.4. Model persistence — scikit-learn 0.19.1 documentation.* [Online]   
Available at: http://scikit-learn.org/stable/modules/model\_persistence.html

Scikit-learn, 2018. *An introduction to machine learning with scikit-learn.* [Online]   
Available at: http://scikit-learn.org/stable/tutorial/basic/tutorial.html

Scikit-learn, 2018. *Documentation of scikit-learn 0.19.1.* [Online]   
Available at: http://scikit-learn.org/stable/documentation.html

Scikit-learn, 2018. *sklearn.decomposition.PCA — scikit-learn 0.19.1 documentation.* [Online]   
Available at: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

Sief, G., 2018. *Selecting the best Machine Learning algorithm for your regression problem - Towards Data Science.* [Online]   
Available at: https://towardsdatascience.com/selecting-the-best-machine-learning-algorithm-for-your-regression-problem-20c330bad4ef

statcounter, 2018. *StatCounter Global Stats - Browser, OS, Search Engine including Mobile Usage Share.* [Online]   
Available at: http://gs.statcounter.com/

Tensorflow, 2018. *An open source machine learning framework for everyone.* [Online]   
Available at: https://www.tensorflow.org/

TMDb - API, 2018. *API Documentation - TMDb.* [Online]   
Available at: https://developers.themoviedb.org/3/getting-started/introduction

TMDb - Rate Limiting, 2018. *API Documentation - Request Rate Limiting.* [Online]   
Available at: https://developers.themoviedb.org/3/getting-started/request-rate-limiting

TMDb, 2018. *Get TV Details API Documentation.* [Online]   
Available at: https://developers.themoviedb.org/3/tv/get-tv-details

TMDb, 2018. *New APIs for external reviews and awards — Talk — The Movie Database (TMDb).* [Online]   
Available at: https://www.themoviedb.org/talk/5904943092514169c000c5f8

# Appendices

## Appendix 1: Project Initiation Document

Individual Project (CS3IP16)

Department of Computer Science

University of Reading

Project Initiation Document

**PID Sign-Off**

|  |  |
| --- | --- |
| **Student No.** | **23013043** |
| **Student Name** | **Kane Small** |
| **Email** | **k.small@student.reading.ac.uk** |
| **Degree programme** (BSc CS/BSc IT) | **BSc CS** |
|  |  |
| **Supervisor Name** | **Jonathan Boyle** |
| **Supervisor Signature** | **JONATHAN BOYLE** |
| **Date** | **04/10/2017** |

**SECTION 1 – General Information**

**Project Identification**

|  |  |
| --- | --- |
| **1.1** | **Project ID**  (as in handbook) |
|  | Own project |
| **1.2** | **Project Title** |
|  | Data mining film data for trend analysis |
| **1.3** | **Briefly describe the main purpose of the project in no more than 25 words** |
|  | The main purpose of this project is to develop a web application that collects film data and provides trend analysis via automated graphing techniques. |

**Student Identification**

|  |  |
| --- | --- |
| **1.4** | **Student Name(s), Course, Email address(s)**  e.g. Anne Other, BSc CS, a.other@student.reading.ac.uk |
|  | Kane Small, BSc CS, k.small@student.reading.ac.uk |

**Supervisor Identification**

|  |  |
| --- | --- |
| **1.5** | **Primary Supervisor Name, Email address**  e.g. Prof Anne Other, a.other@reading.ac.uk |
|  | Jonathan Boyle, j.n.boyle@reading.ac.uk |
| **1.6** | **Secondary Supervisor Name, Email address**  Only fill in this section if a secondary supervisor has been assigned to your project |
|  |  |

**Company Partner (only complete if there is a company involved)**

|  |  |
| --- | --- |
| **1.7** | **Company Name** |
|  |  |
| **1.8** | **Company Address** |
|  |  |
| **1.9** | **Name, email and phone number of Company Supervisor or Primary Contact** |
|  |  |

**SECTION 2 – Project Description**

|  |  |
| --- | --- |
| **2.1** | **Summarise the background research for the project in about 400 words. You must include references in this section but don’t count them in the word count.** |
|  | The background research for the project will begin by comparing the existing film aggregation and review platforms available today and gathering statistics on each site such as the total number of films stored, the number of unique attributes listed, as well as information about the users who visit on a daily basis. From this initial information, it will give a better understanding of the trends of film accumulation and the popularity of the film industry as a whole.  Research into the top open source film database API’s (Application Programming Interface) and how much data they provide access to will be conducted, providing an overview on which film trends to analyse. As it stands, the most prominent candidate is ‘The Movie Database’, which has a vast collection of films at over 400,000 and provides access to information such as cast, crew, plot keywords, release information, reviews and more. The disadvantage to utilising this API is that it does not allow of the entire film database to be dumped/extracted and would therefore have to be queried incrementally across the film ids, which would take a long time due to the requests cooldown period.  Furthermore, there are multiple options for handling the data requested via the API. All of the required data could be pulled into a singular local database, which would have the advantage of reducing the number of requests having to be made however, the disadvantage would be that the data would not be displayed in real time. Another option would be a tool such as ‘Elasticsearch’, which not only handles large volumes of data well and queries datasets almost instantly but is highly scalable; so would be able to handle the ever-increasing additions of new films to the database. Though, as this is a small-scale project as opposed to a business application implementing Elasticsearch may not be required and/or manageable in the given timeframe.  Finally, research will need to be carried out with regards to the different data mining techniques available. Currently, the four main techniques that will be compared are association, classification, clustering and prediction. Prediction is the most versatile method and would allow for additional features to be added to the project at a later date. Once a more detailed overview of each technique has been obtained, a selection will be made on which technique(s) to employ, primarily based on the advantages and disadvantages of each. From the selected technique(s), a method to present the data in a meaningful format will be considered. This will include textual data alongside such mediums as graphs and charts.  IMDb: <http://www.imdb.com>  The Move Database API: <https://www.themoviedb.org/documentation/api>  Data mining techniques: <https://www.ibm.com/developerworks/library/ba-data-mining-techniques/> |

|  |  |
| --- | --- |
| **2.2** | **Summarise the project objectives and outputs in about 400 words.** These objectives and outputs should appear as tasks, milestones and deliverables in your project plan. In general, an objective is something you can do and an output is something you produce – one leads to the other. |
|  | **Objectives:**   * Compare existing film aggregation and review platforms. * Learn how to use The Movie Database API, or an equivalent service. * Learn how to implement Amazon Web Services Lambda Functions. * Compare and select the required data mining technique(s). * Identify and analyse the limitations of the data mining technique(s) selected. * Identify how these systems can be exploited and ultimately mitigated against, for example preventing too many requests within a certain time period. * Design, build and configure the database to store the film data. * Design and build the front-end web interface. * Test the system’s functionality. * Review all tests and make any required improvements/fixes. * Compile a report of the entire process from planning to completion.   **Outputs:**   * Produce a document detailing the comparison between the film aggregation and review platforms. * Produce a web application that illustrates the current trends among film attributes. * Produce a testing plan. * Produce a final report detailing each stage of the development process of this program. * Output a reflection analysing how the project has been executed and what has been successful/unsuccessful. |

|  |  |
| --- | --- |
| **2.3** | **Initial project specification - list key features and functions of your finished project.** Remember that a specification should not usually propose the solution. For example, your project may require open source datasets so add that to the specification but don’t state how that data-link will be achieved – that comes later. |
|  | The following key features and functions will be present in this project:   * The system will deliver trend analysis on film data queried via an open source API. * A web front-end interface will be created to showcase the trend analytics. * Data will be presented to the user in both a graphical and textual format. * The analytics graphs will update automatically whenever new data is received from the API. * Users will have the option to filter the analytics based on certain information such as genre, budget and user rating. * The system will function across all popular web browsers, from Google Chrome to Safari. |
| **2.4** | **Describe the social, legal and ethical issues that apply to your project. Does your project require ethical approval?** |
|  | The project will not require any ethical approval, as all of the film data will be acquired via an open source API. There will also be no social or legal issues that apply to this project, due to the nature of the source of the data that will be acquired and the fact that said data will only be used for analytic purposes and not for any monetary gains. |
| **2.5** | **Identify and lists the items you expect to need to purchase for your project. Specify the cost (include VAT and shipping if known) of each item as well as the supplier.** e.g. item 1 name, supplier, cost |
|  | All of the content required for this project is open source and can be found online. |
| **2.6** | **State whether you need access to specific resources within the department or the University e.g. special devices and workshop** |
|  | All of the content required for this project is open source and can be found online, therefore there is no need for any specific departmental resources. |

**SECTION 3 – Project Plan**

|  |  |  |  |
| --- | --- | --- | --- |
| **3.1** | **Project Plan**  Split your project work into sections/categories/phases and add tasks for each of these sections. It is likely that the high-level objectives you identified in section 2.2 become sections here. The outputs from section 2.2 should appear in the Outputs column here. Remember to include tasks for your project presentation, project demos, producing your poster, and writing up your report. | | |
|  | | | |
| **Task No.** | **Task description** | **Effort**  **(weeks)** | **Outputs** |
| **1** | **Background Research** | **2.6** |  |
| 1.1 | Research existing film aggregation and review platforms | 2 days | Research document |
| 1.2 | Research and compare data mining techniques | 2 days | Comparison document |
| 1.3 | Research database design | 1 | Research document |
| 1.4 | Research front-end frameworks | 1 | Research document |
| 1.5 | Research limitations to current data mining techniques | 2 days | Research document |
| **2** | **Analysis and design** | **2.3** |  |
| 2.1 | Context analysis | 3 days | Case model diagram |
| 2.2 | Use cases | 2 days | Use case diagram |
| 2.3 | Functional workflow | 2 days | Flow diagram |
| 2.4 | Back-end database design | 5 days | Database design |
| 2.5 | Trend algorithm design | 5 days | Flowchart/pseudocode |
| **3** | **Develop prototype** | **8** |  |
| 3.1 | Develop the database | 2 | A database to store the film data. |
| 3.2 | Develop the front-end interface | 3 | A web interface. |
| 3.3 | Develop the logic for trend analysis | 3 | The logic for the trend analysis in code form. |
| **4** | **Testing, evaluation/validation** | **4.2** |  |
| 4.1 | Unit testing | 1 | Testing of each individual component. |
| 4.2 | System testing | 1 | Testing carried out for the entire system, from end-to-end. |
| 4.3 | Functional testing | 1 | Testing of all requirements. |
| 4.4 | Usability testing | 1 | Testing of the user experience. |
| 4.5 | Live-testing with users | 2 days | Testing the application with actual end-users. |
| **5** | **Assessments** | **5.3** |  |
| 5.1 | Write-up project report | 4 | Project Report |
| 5.2 | Produce poster | 3 days | Poster |
| 5.3 | Prepare presentation/demonstration | 1 | Presentation/demo |
| **TOTAL** | **Sum of total effort in weeks** | **23** |  |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **SECTION 4 - Time Plan for the proposed Project work** | | | | | | | | | | | | | | | | | | |
| For each task identified in 3.1, please *shade* the weeks when you’ll be working on that task. You should also mark target milestones, outputs and key decision points. To shade a cell in MS Word, move the mouse to the top left of cell until the curser becomes an arrow pointing up, left click to select the cell and then right click and select ‘borders and shading’. Under the shading tab pick an appropriate grey colour and click ok. | | | | | | | | | | | | | | | | | | |
| * + - * 1. **Project stage** | * + - * 1. **START DATE: /10/2017**         2. **Project Weeks** | | | | | | | | | | | | | | | | | |
| 0-3 | | 3-6 | | 6-9 | 9-12 | 12-15 | | 15-18 | | 18-21 | 21-24 | | 24-27 | 27-30 | 30-33 | 33-36 | 36-39 |
| **1 Background Research** |  |  |  | |  |  |  | |  | |  |  | |  |  |  |  |  |
|  |  |  |  | |  |  |  | |  | |  |  | |  |  |  |  |  |
|  |  |  |  | |  |  |  | |  | |  |  | |  |  |  |  |  |
| **2 Analysis/Design** |  |  |  |  |  |  |  | |  | |  |  | |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  | |  |  | |  |  |  |  |  |
|  |  |  |  |  |  |  |  | |  | |  |  | |  |  |  |  |  |
| **3 Develop prototype.** |  | |  |  |  |  |  |  |  | |  |  | |  |  |  |  |  |
|  |  | |  |  |  |  |  |  |  | |  |  | |  |  |  |  |  |
|  |  | |  |  |  |  |  |  |  | |  |  | |  |  |  |  |  |
| **4 Testing, evaluation/validation** |  | |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |
|  |  | |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |
|  |  | |  | |  |  |  |  |  |  |  |  | |  |  |  |  |  |
| **5 Assessments** |  | |  | |  |  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  | |  | |  |  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  | |  | |  |  |  | |  |  |  |  |  |  |  |  |  |  |
|  |  | |  | |  |  |  | |  |  |  |  |  |  |  |  |  |  |

**Risk Assessment Form**

|  |  |  |  |
| --- | --- | --- | --- |
| **Assessment Reference No.** |  | **Area or activity assessed:** |  |
| **Assessment date** |  |
| **Persons who may be affected by the activity (i.e. are at risk)** |  |

**SECTION 1: Identify Hazards -** *Consider the activity or work area and identify if any of the hazards listed below are significant (tick the boxes that apply).*

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Fall of person (from work at height) |  |  | Lighting levels |  |  | Use of portable tools / equipment |  |  | Vehicles / driving at work |  |  | Hazardous fumes,  chemicals, dust |  |  | Occupational stress |  |
|  | Fall of objects |  |  | Heating & ventilation |  |  | Fixed machinery or lifting equipment |  |  | Outdoor work / extreme weather |  |  | Hazardous biological agent |  |  | Violence to staff / verbal assault |  |
|  | Slips, Trips & Housekeeping |  |  | Layout, storage, space, obstructions |  |  | Pressure vessels |  |  | Fieldtrips / field work |  |  | Confined space / asphyxiation risk |  |  | Work with animals |  |
|  | Manual handling operations |  |  | Welfare facilities |  |  | Noise or Vibration |  |  | Radiation sources |  |  | Condition of Buildings & glazing |  |  | Lone working / work out of hours |  |
| 1. **55** | Display screen equipment | **✓** |  | Electrical Equipment |  |  | Fire hazards & flammable material |  |  | Work with lasers |  |  | Food preparation |  |  | Other(s) - specify | **✓** |

**SECTION 2: Risk Controls** *- For each hazard identified in Section 1, complete Section 2.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Hazard No**. | Hazard Description | Existing controls to reduce risk | **Risk Level** (tick one) | | | Further action needed to reduce risks |
|  | High | Med | Low | *(provide timescales and initials of person responsible)* |
| 5 | Extended periods of time looking at the computer screen | Regular breaks, computer glasses that prevent glare and/or eye strain. Potentially look into using a program such as *f*.*lux*, to reduce blue light exposure. |  | X |  |  |
| 30 | Hardware failure | Back up device on a regular basis. |  | X |  | Backup all project work on cloud services as well as external HDD’s to minimise the risk of irretrievable data loss as much as possible. |
| 30 | Repetitive strain injury | Take regular breaks, use good posture, ensure that your workstation is ergonomically designed/setup and make sure not to over-exert yourself whilst using the computer, i.e. stretching for hard-to-reach keys. |  | X |  |  |
| **Name of Assessor(s)** | |  | **SIGNED** | | | |
| **Review date** | |  |

## Appendix 2: Logbook

**Logbook**

**Week 1 (09/10/2017 – 15/10/2017)**

**Objectives**

* Research existing film aggregation and review platforms

**Description**

Throughout the course of this week I researched all of the existing platforms that implement some form of aggregation and/or review systems. I found the following to be the top platforms available today:

* [IMDb](http://www.imdb.com/)
* [Rotten Tomatoes](https://www.rottentomatoes.com/)
* [Netflix](https://www.netflix.com/browse)
* [Metacritic](http://www.metacritic.com/)
* [The Movie Database](https://www.themoviedb.org/?language=en)

IMDb unfortunately doesn’t have a proper API but instead allows users to download [subsets of data](http://www.imdb.com/interfaces/) such as the titles of movies for example. This will not be viable for my project.

Rotten Tomatoes does have an actual [API](https://developer.fandango.com/Rotten_Tomatoes), but limits the amount of information that developers are able to access/request from the API.

Finally, The Movie Database is a free [API](https://developers.themoviedb.org/3/movies/get-movie-details) service that allows users to request a variety of data on films and TV shows from, cast details, ratings, trailers, genres, etc. This is the service that I have decided to utilise for my project. There was one other alternative that was just as good as my selection, The Open Movie Database, however it was limited to 1,000 free daily request rate otherwise you had to become a patron and pay a monthly subscription.

**Problems encountered**

Not really any specific problems encountered, other than it took longer than expected to finalise my decision on which API service to use. There also weren’t any open source options available that I could find and although this doesn’t affect my project directly it would’ve been nice to see how a service manages to aggregate all of the film data like The Movie Database does for example.

**Tools used**

* Google Chrome

**References**

* <https://developers.themoviedb.org/3/movies/get-movie-details>

**Keywords**

API, request rate, patreon, open source

**Week 2 (16/10/2017 – 22/10/2017)**

**Objectives**

* Research and compare data mining techniques
* Select a suitable algorithm for this project

**Description**

This week I looked into what data mining is and how I will use it for my project. I started off by researching the main data mining techniques and found that they were association, classification, clustering and prediction.

**Association:** This is where correlations between two or more data items are made, in order to identify patterns.

**Classification:** Is used to build up an idea of an object by describing multiple attributes that identify a particular class. For example, cars can be identified into different types (hatchback, coupe, 4x4) by identifying different attributes (number of seats, car shape, forward/rear/all-wheel drive).

**Clustering:** Is used to correlate and group attributes that have similar defining features.

**Prediction:** Often used in conjunction with one or more of the other data mining techniques, involves analysing trends, patter matching and relation in order to predict a value or class.

Prediction is definitely the technique that I will be utilising the most, feeding in all of the film data attributes such as genre, actors, directors, budget, etc and then predicting an overall rating based on this data.

**Problems encountered**

Having already planned to build my application in Python, I hope that there is a data mining/machine learning Python library that I can incorporate into my project.

**Tools used**

* Google Chrome

**References**

* <https://www.kdnuggets.com/2015/05/top-10-data-mining-algorithms-explained.html>
* <https://www.ibm.com/developerworks/library/ba-data-mining-techniques/>
* <https://www.technologyreview.com/s/538701/data-mining-reveals-the-surprising-factors-behind-successful-movies/>

**Keywords**

Data mining, association, classification, clustering, prediction, class

**Week 3 (23/10/2017 – 29/10/2017)**

**Objectives**

* Research database design
* Select an appropriate database management system

**Description**

This week I looked into database design and how best to build the database that will house my training data. I started off by looking on Google to see if I could find any online schema designers and the best one I could find was [DbDesigner.net](https://dbdesigner.net/). This web application was extremely versatile and although it doesn’t allow non-premium users to save their designs it did allow me to save my SQL script and I even had the choice of PostgresSQL, SQLite, MySQL, MSSql or Oracle; which I was very impressed with.

Early on in the project planning phase, I new that I wanted to build an application that utilised a database in some way. I’ve had experience with MySQL in the past and decided that because of this reason and also because of the ease-of-use when it comes to linking a MySQL database to a front-end application of some sort that this would be the best database management system to implement.

Finally, I researched what the best visual database management system tool on the market was. I came across quite a few contenders such as HeidiSQL, dbForge Studio, DBTools Manager, etc but most of the options were not open source. I settled on MySQL Workbench which has a lot of documentation and is free to use. My favourite GUI database tool is by far Sequel Pro, but unfortunately is for MacOS only. So I will be using MySQL Workbench when working on my Windows PC at home and Sequel Pro whenever I am at university and am working on my MacBook Pro.

**Problems encountered**

Sequel Pro only available on MacOS.

**Tools used**

* Google Chrome
* MySQL
* MySQL Workbench
* Sequel Pro

**References**

* <https://dbdesigner.net/>

**Keywords**

Database, DBMS, MySQL

**Week 4 (30/10/2017 – 05/11/2017)**

**Objectives**

* Research similar projects for influence and references

**Description**

This week involved a lot of research into existing projects that are similar in mine. This was something that I’d wanted to do for quite a while to see if anyone had undertaken a similar task, mainly to identify if they had run into any major issues and how I could overcome or avoid the same pitfalls altogether.

I managed to find a total of 5 very similar projects that would be an aid toward my project. The [first](https://nikhilwins.wordpress.com/2015/09/18/movie-recommendations-how-does-netflix-do-it-a-9-step-coding-intuitive-guide-into-collaborative-filtering/) talked a lot about film recommendations and how Netflix implements their suggestion algorithm, but also outlined how to build a collaborative filtering application in Python. The [second](https://nycdatascience.com/blog/student-works/machine-learning/movie-rating-prediction/) focused more on scraping films from IMDb instead of utilising an API like my project is going to and also was trying to answer the question *“Will the number of human faces in a movie poster correlate with the movie rating?”*. They focused on the IMDb film ratings but also pulled in other attributes such as an actor’s Facebook popularity. The [third](http://www.diva-portal.org/smash/get/diva2:352538/FULLTEXT01.pdf) project focused on answering the question *“How do movie producers identify the genre shifting trend?”*, so might not be as useful as previously thought. The [fourth](http://predictive.analyticsight.com/wp-content/uploads/2013/01/movie_genre_TimSchellhase.pdf) project focused on predicting the genre of a film by implementing a regression model. They analysed each film’s summary using topic clustering and then accurately predicted the genre of the film. Finally, the [fifth](http://usir.salford.ac.uk/18838/1/Wessex_movie.pdf) and most interesting project was a very similar project to mine and focused on the prediction of film ratings. They achieved this by selecting the data, cleaning and integrating it, transforming it and then finally performing the actual data mining. But instead of predicting a numeric value, they generalised the ratings into four categories: excellent, average, poor and terrible.

**Problems encountered**

Only one project was very similar to mine, but I will be utilising slightly different data mining and machine learning techniques as well as displaying the data in a front-end application format.

**Tools used**

* Google Chrome

**References**

* <http://usir.salford.ac.uk/18838/1/Wessex_movie.pdf>
* <https://nikhilwins.wordpress.com/2015/09/18/movie-recommendations-how-does-netflix-do-it-a-9-step-coding-intuitive-guide-into-collaborative-filtering/>
* <https://nycdatascience.com/blog/student-works/machine-learning/movie-rating-prediction/>
* <http://predictive.analyticsight.com/wp-content/uploads/2013/01/movie_genre_TimSchellhase.pdf>
* <http://www.diva-portal.org/smash/get/diva2:352538/FULLTEXT01.pdf>

**Keywords**

Regression model

**Week 5 (06/11/2017 – 12/11/2017)**

**Objectives**

* Research front-end frameworks
* Compare existing frameworks
* Select a suitable framework for this project

**Description**

This week I focused on researching and selecting a suitable front-end framework for my project. Having decided early on to build my back-end in Python, I had to look for a front-end framework that would incorporate Python as the back-end. Django immediately came to mind, as I had heard of it a few times in the past and whilst on my placement. However, I had to ensure that I properly compared the existing frameworks to find the one that best suited my project and my experience with both Python and front-end web technologies.

I found a very good article which compared Django, Flask and Pyramid, three popular Python front-end frameworks. What I concluded from reading the article was that Django was the most popular framework, with the most StackOverflow questions relating to that framework having been asked, but was more suitable for mid-large applications. Pyramid is the most flexible, but can be intimidating for new users as it presents a lot of pre-configuration options that can be daunting. Flask however, is really useful for developers who are working on smaller projects and need a fast yet powerful Python front-end application. Flask also has the smallest footprint and requires a lot less code to perform the same task as the other frameworks. Django and Pyramid both have in-built tools for bootstrapping for example, meaning that it is very simple to setup a skeleton of the project quickly. They also support ORM (Object Relational Mapping) which makes it easier to connect to and manipulate a database from within Python code – instead of using basic SQL, tables are mapped to objects within Python. I discussed this with my supervisor and it was something that he hadn’t heard of himself, but in the end I decided that learning about and implementing ORM didn’t seem like an advantage.

Therefore, this is why I decided that Flask would be my framework of choice for this project.

**Problems encountered**

Not knowing where to start with front-end Python frameworks.

**Tools used**

* Google Chrome

**References**

* <https://www.airpair.com/python/posts/django-flask-pyramid#1-introduction>

**Keywords**

Framework, ORM

**Week 6 (13/11/2017 – 19/11/2017)**

**Objectives**

* Create an application architecture design diagram
* Design the database to hold the training data

**Description**

This week I wanted to design and create an application architecture diagram, which will illustrate how my project will be setup.

I decided that I wanted to have my Python code that would retrieve the film data from the API, my database that would store all of the film data, an API which would retrieve the data from the database and my front-end website which would simply display the data. It is necessary for the front-end to ever only display data and to not request it directly from the database.

I designed a simple draft diagram in [Draw.io](https://www.draw.io/), having used the tool previously and allowing for the diagram to be saved directly to the cloud (Google Drive).

I started simple with my design and built the ***films*** table, that would house all of the film data, ensuring to specify which of the values would be my primary and foreign keys, what their types are, whether they should be unique values, whether they should auto increment or whether they should allow null values. For film data attributes such as ***genres****,* ***actors****,* ***production companies***it became apparent that I wouldn’t be able to store these directly in the ***films*** table, because a single film can have multiple iterations of these values. I solved this problem by storing these attributes in their own tables, but I then had to work on normalising the tables by adding in association tables between the ***films*** table and these newly created tables. For example, the ***genres***table stores the ***genre\_id***and ***genre\_title*** and is linked to the ***films*** table by an association table that stores the ***id****,* ***film\_id***and **­*genre\_id****.* I repeated these steps for the actors and production company tables.

**Problems encountered**

Having never created an application architecture diagram, it took quite a bit of research before I was happy with the final work I had produced.

**Tools used**

* Google Chrome
* [Draw.io](https://www.draw.io/)
* [DbDesigner.net](https://dbdesigner.net/)

**References**

* <https://slidemodel.com/templates/software-diagrams-powerpoint/four-layers-modern-web-application-architecture-diagram/>

**Keywords**

Application architecture diagram, primary/foreign keys, auto increment, normalisation, association table

**Week 7 (20/11/2017 – 26/11/2017)**

**Objectives**

* Create the database
* Think about storage/hosting options

**Description**

This week I focused on actually building the database. I began by creating a simple Debian virtual machine with VirtualBox and installed MySQL. I created a simple database called ***training\_data*** via the console in Debian and then worked on setting up MySQL Workbench to connect to my newly created database. Connecting to the database actually proved rather challenging, as I eventually found out that I had to ensure that the ***bind-address*** option within the config file (/etc/my.cnf) was set to my computer’s IP address. Once this had been resolved, I started on the actual table creation. Considering I already had my schema and SQL downloaded from DbDesigner.net, creating the tables was as simple as running the SQL in the query editor within MySQL Workbench. It however, didn’t work immediately, as a lot of the values were not correct and I found myself dropping and recreating the tables several times. I eventually managed to get all of the tables created, with the correct values and value types.

At this point I started to think about storage/hosting options, as having the database reside on a VM on my PC locally meant that I could not access and/or work on the database externally (i.e. at university). This would prove problematic in the coming weeks when I would be at university all week and during my free periods would want to work on my project. I decided that I would move the database to my Raspberry Pi (which first needed Raspbian to be reinstalled and MySQL to also be installed and configured) by exporting the ***training\_data*** database. I then installed a service called ***XRDP*** which is a remote desktop service for Raspbian OS. This would allow me to remotely connect to it from (unfortunately) my local PC. This meant that I had to (using TeamViewer) remotely connect to my Windows PC at home, from my MacBook at university and then RDP from my Windows PC to the Raspberry Pi. This was a very long, tedious and expensive workflow process which I do not intend to keep indefinitely. I will continue to search for a better solution.

I do host my personal website with HostPresto and they allow me, with the package I have purchased, to setup two databases and two websites (with separate domain names). So, I might be able to host my entire application with my web hosting company. I will look into this in the next few weeks, or when I next have time.

**Problems encountered**

Hosting the database on my Raspberry Pi proved to be a very expensive process.

**Tools used**

* VirtualBox
* MySQL Workbench/Sequel Pro
* Raspberry Pi (Raspbian OS)
* XDRP

**References**

* <http://www.circuitbasics.com/access-raspberry-pi-desktop-remote-connection/>

**Keywords**

Virtual machine, Hosting, IP Address, RDP, SQL query

**Week 8 (27/11/2017 – 03/12/2017)**

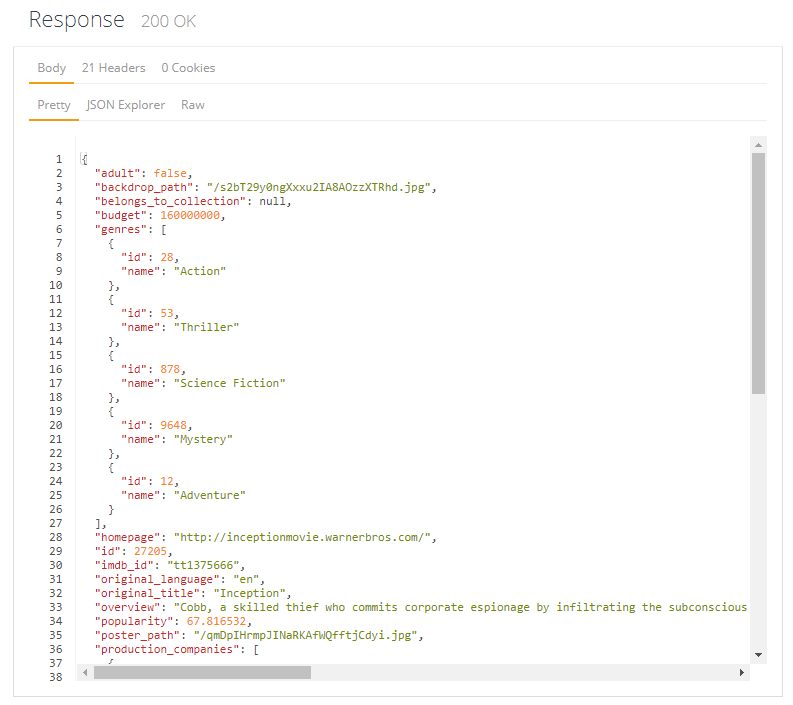
**Objectives**

* Program the film data acquisition program (utilising TMDB API)

**Description**

This week I began development on my Python film data acquisition code. I first had to set myself up an account with The Movie Database, which would then provide me with an API key. This was a very straight forward process. Next, I began simply attempting to retrieve data from TMDB using the Python [requests library](http://docs.python-requests.org/en/master/user/quickstart/). Using the film Inception as an example (grabbing the film ID from TMDB’s search tool) I managed to return film data in JSON format for this particular film, just like it shows in the API documentation.

The documentation pages allow you to actually test requests in the browser to illustrate what you should see returned when performing this task externally.

I then managed to start working on and complete the function that takes the JSON file, locates the relevant film attributes, stores them in variables and then passes them to an SQL query called [***INSERT INTO***](https://www.w3schools.com/sql/sql_insert.asp), which takes a set of parameters and inserts them into an existing table. These parameters can even be Python variables as long as you pass them to the SQL query as [string formatters](https://www.tutorialspoint.com/python/python_strings.htm).

**Problems encountered**

I did encounter one problem, where I was receiving a warning for every request made to TMDB API. I mitigated this issue by adding a disable warning piece of code:

* requests.packages.urllib3.disable\_warnings()

This, according to a StackOverflow post, was not an error and was perfectly fine to disable.

**Tools used**

* [TMDB API](https://developers.themoviedb.org/3/getting-started/introduction)
* [Python 2.7](https://www.python.org/download/releases/2.7/)
* [Jetbrains PyCharm](https://www.jetbrains.com/pycharm/)

**References**

* <https://developers.themoviedb.org/3/getting-started/introduction>
* <https://www.themoviedb.org/talk/5406a1670e0a261889000376?language=en>
* <https://www.themoviedb.org/talk/54f41aafc3a3683455000991?language=en>
* <http://docs.python-requests.org/en/master/user/quickstart/>
* <https://www.themoviedb.org/movie/27205-inception>
* <https://www.w3schools.com/sql/sql_insert.asp>
* <https://www.tutorialspoint.com/python/python_strings.htm>
* <https://stackoverflow.com/questions/42839363/python-disable-warnings-for-securitywarning-certificate-has-no-subjectaltnam>

**Keywords**

Python, JSON, INSERT INTO

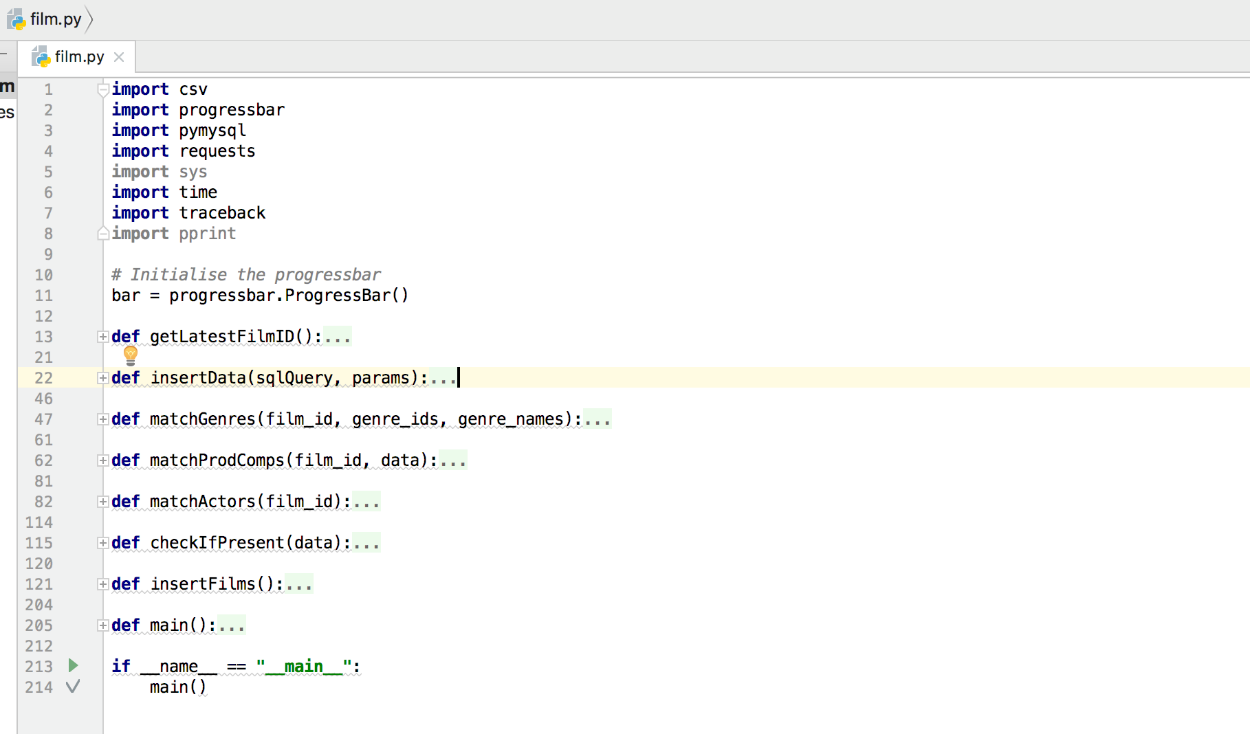
**Week 9 (04/12/2017 – 10/12/2017)**

**Objectives**

* Continue building the Python application

**Description**

This week I continued with my Python development. I found two very useful forum posts, where a couple of TMDB users had asked how to retrieve all film data. The simple answer was to iterate over the API, which I had already concluded. So, I began to build out the functions that I would need:

* insertData()
* matchGenres()
* matchProdComps()
* matchActors()
* insertFilms()

**Problems encountered**

N/A

**Tools used**

* [TMDB API](https://developers.themoviedb.org/3/getting-started/introduction)
* [Python 2.7](https://www.python.org/download/releases/2.7/)
* [Jetbrains PyCharm](https://www.jetbrains.com/pycharm/)

**References**

* <https://developers.themoviedb.org/3/getting-started/introduction>
* <https://www.themoviedb.org/talk/5406a1670e0a261889000376?language=en>
* <https://www.themoviedb.org/talk/54f41aafc3a3683455000991?language=en>
* <https://www.themoviedb.org/movie/27205-inception>

**Keywords**

N/A

**Week 10 (11/12/2017 – 17/12/2017)**

**Objectives**

* Continue with Python development
* Design a questionnaire to gather statistics about the most important film attributes

**Description**

This week I focused on making the main function iterable via a local variable (***latest\_film\_id***) which is a variable returned from the ***getLatestFilmID()*** function and that function jut queries TMDB API which has a ***GET /movie/latest*** URL.

I also implemented a [progressbar](https://pypi.python.org/pypi/progressbar2), utilising the Python library which would (using the latest film id as a maximum value) display a progressbar in the console so that I could identify the percentage of completion when my application was executing. This would be extremely useful when it came to running my final execution.

This week I also managed to design and create a questionnaire using Microsoft Forms. I thought the questionnaire would be a good way to gather statistics on people who watch films, whilst at the same time allow me to determine how people who watch films rank the film attributes regarding how much of an effect said attribute has on whether they see a film. The questionnaire ended up having a total of 4 questions:

* Please specify your age
  + Dropdown box with a ‘Prefer not to say’ option.
* How many films do you watch on average, per month?
  + Radio buttons.
* Which platform(s) do you use to watch films? (Select all that apply)
  + Multi-select radio buttons.
* Please rank the following attributes, indicating which hold the highest weight for you when deciding whether to watch a film
  + List of film attributes that can be ordered by dragging and dropping into place.

**Problems encountered**

* <https://developers.themoviedb.org/3/getting-started/introduction>
* <https://www.themoviedb.org/talk/5406a1670e0a261889000376?language=en>
* <https://www.themoviedb.org/talk/54f41aafc3a3683455000991?language=en>
* <https://www.themoviedb.org/movie/27205-inception>
* <https://pypi.python.org/pypi/progressbar2>

Tools used

* [TMDB API](https://developers.themoviedb.org/3/getting-started/introduction)
* [Python 2.7](https://www.python.org/download/releases/2.7/)
* [Jetbrains PyCharm](https://www.jetbrains.com/pycharm/)
* [Microsoft Forms](https://forms.office.com/Pages/DesignPage.aspx#FormId=xDv6T_zswEiQgPXkP_kOX7zAW1OgPQRFlgBCrgMdZ1BUN0pRQzlLQzlHTlpGWjNUS0JTMlFLM1pYMi4u)

**References**

* N/A

**Keywords**

N/A

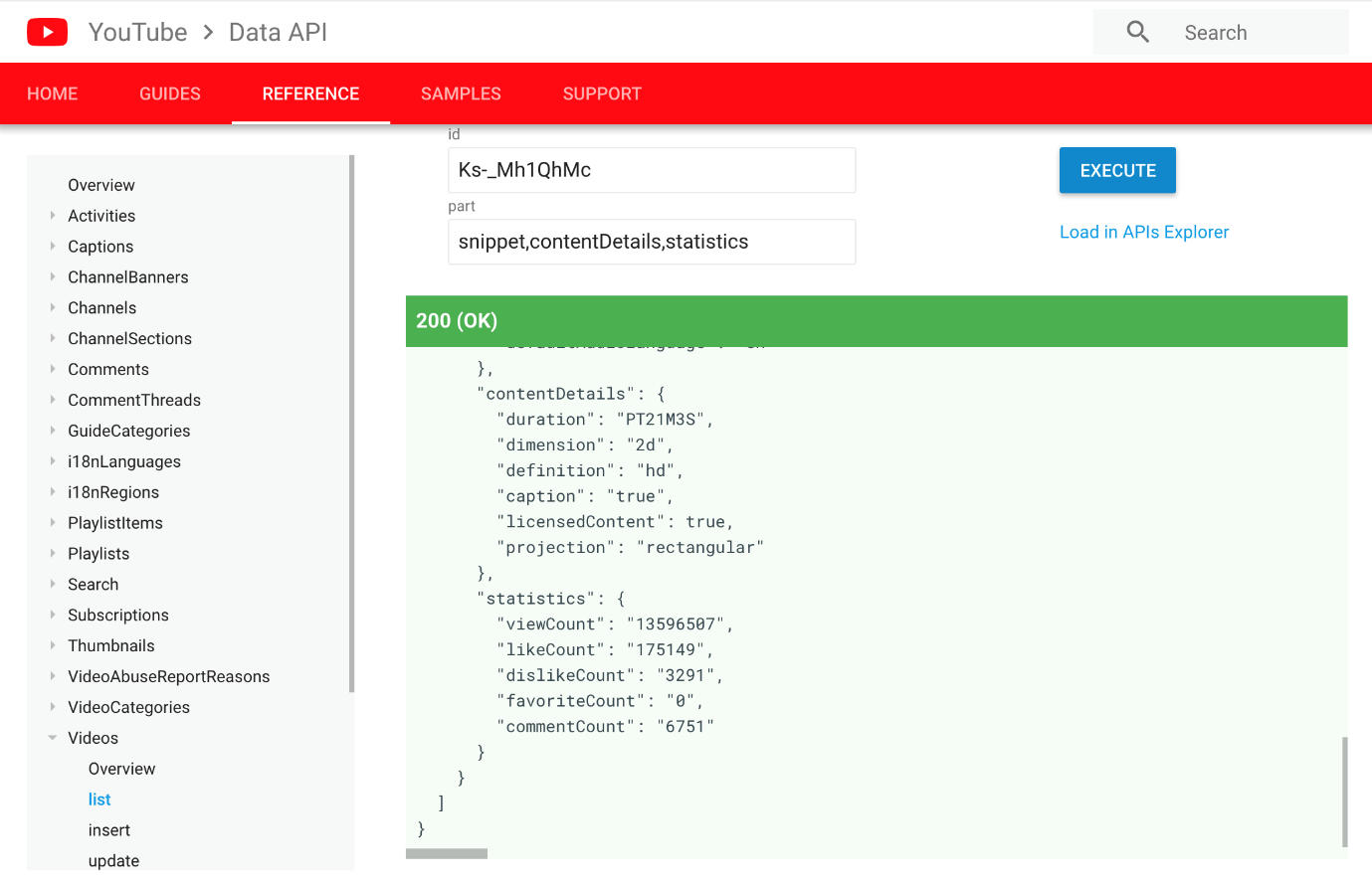
**Week 11 (18/12/2017 – 24/12/2017)**

**Objectives**

* Consider implementing the YouTube API to gather trailer statistics
* Research if an API exists to gather information on Oscar winnings

**Description**

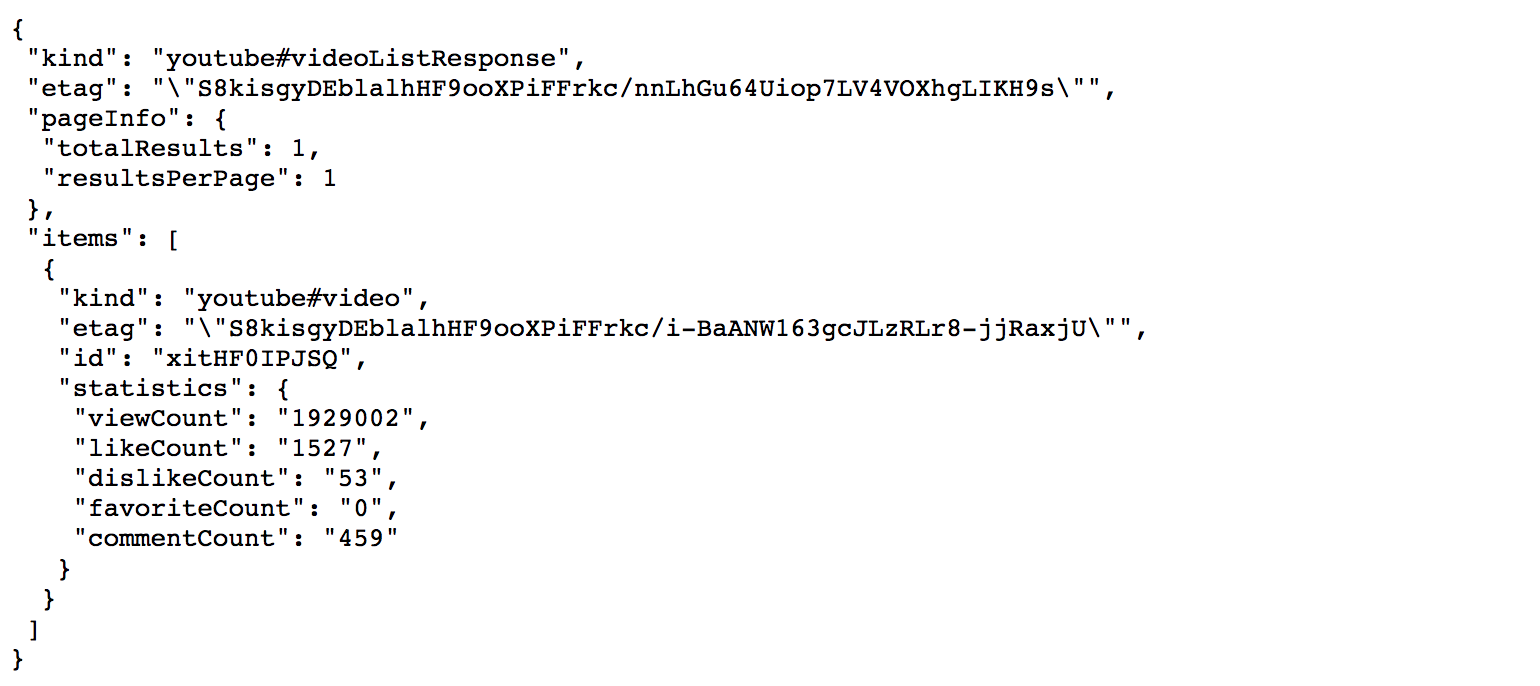
This week I worked on gathering data on trailers as per a suggestion from my supervisor. I started by researching the YouTube API, reading the documentation and see how simple it would be to gather the data I required with just the video id that I had acquired from querying TMDB API.

After some research, I found that I was after the ***Videos: list*** section of the YouTube API. Running the in-built browser request tool, I found that I would need to locate the specific JSON object, ***items[statistics]***.

This would provide me with all of the video data that I was after – total view count and like/dislike count.

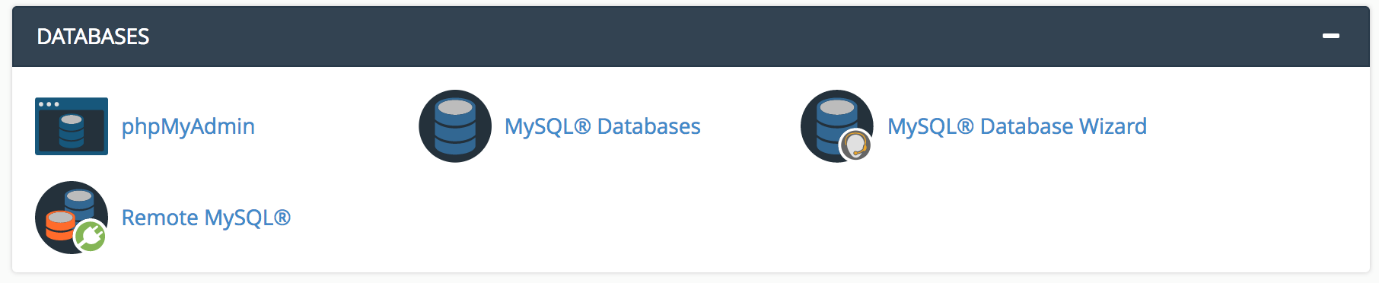
Implementing this function in Python was a lot simpler than expected, I called it ***getTrailerViewCount()*** and simply had to use the same responses library that I used to gather the film data from TMDB API. I did however, also require a key via the YouTube API service for developers. Using the Google APIs Developer dashboard, I created an API key without any restrictions for now and assigned it the access to the YouTube API service.

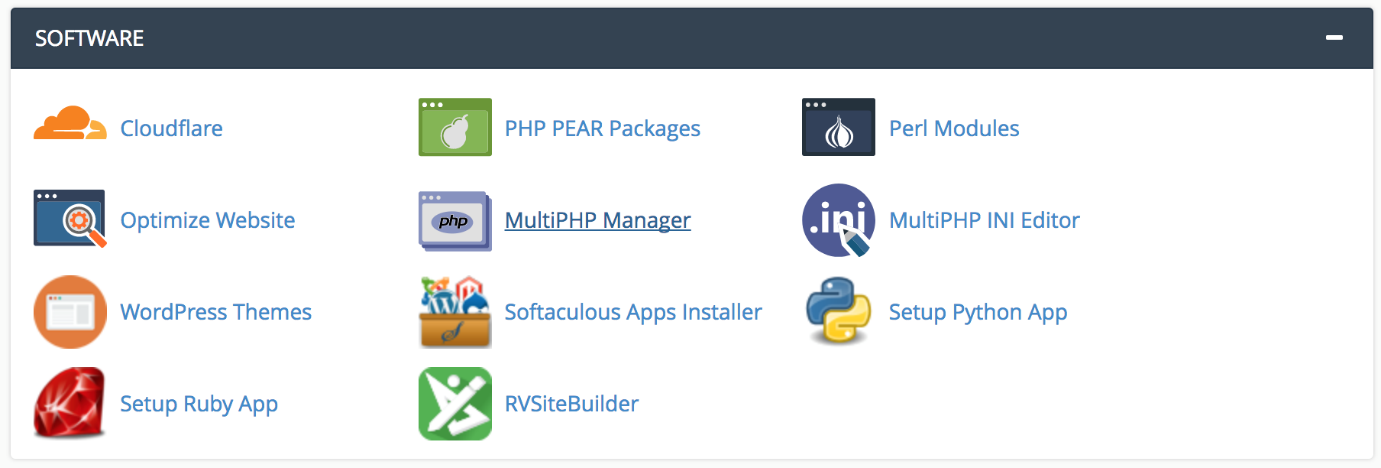
From here, I now had access to the YouTube API service my API key and a simple test in the browser demonstrated this.

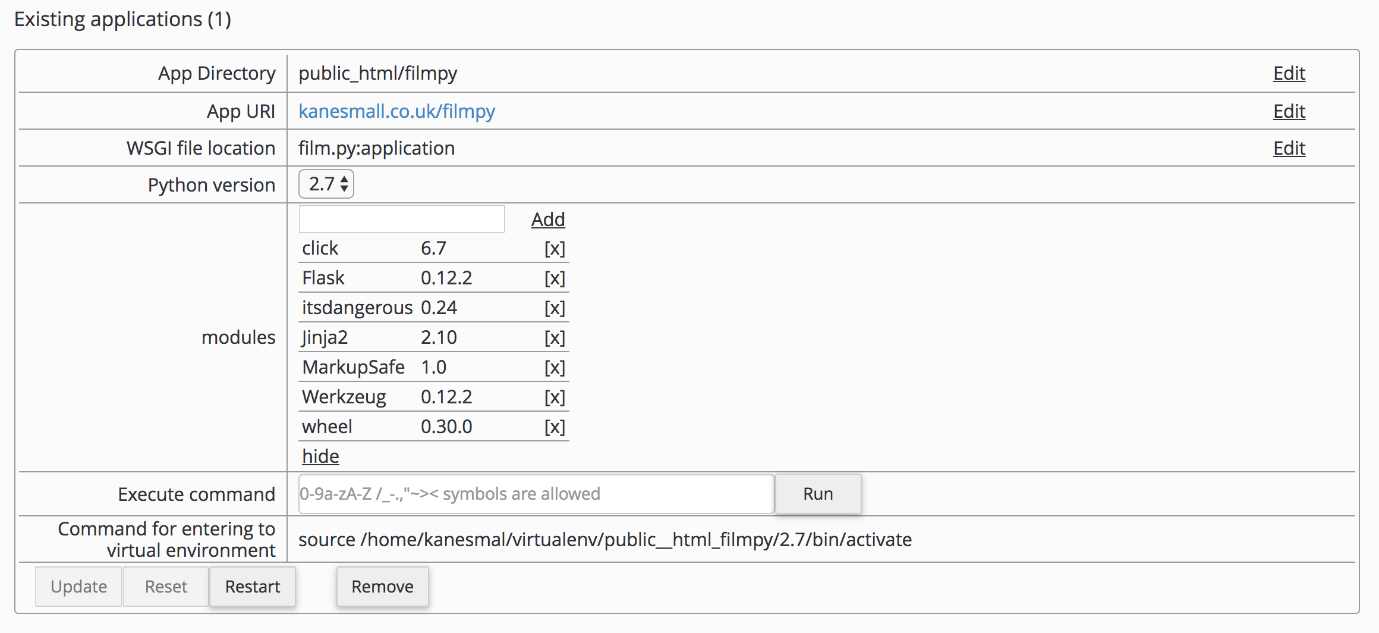


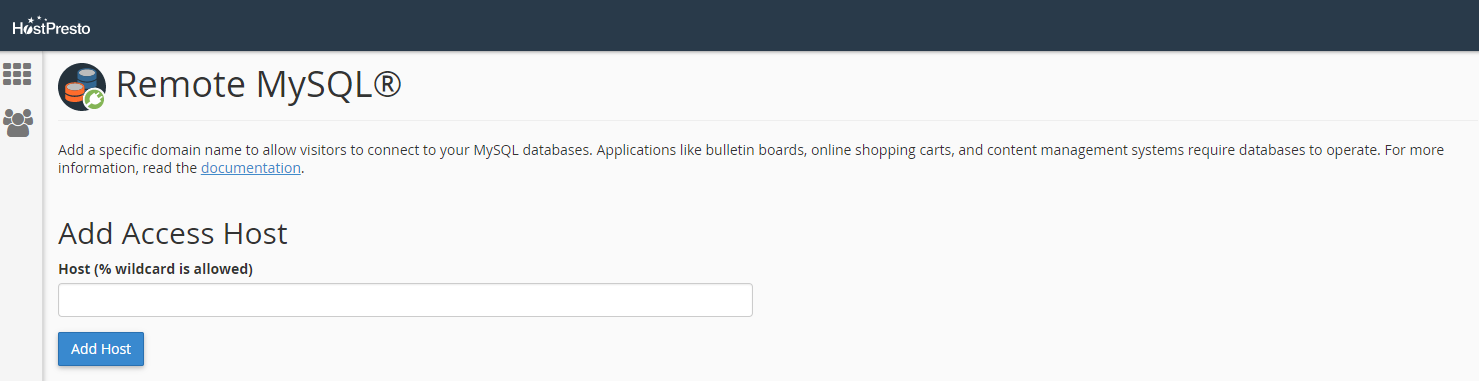
Which returned the JSON without any issues.

From here I was able to build up a simple function that traversed the JSON, stored the required statistic values into variables and then passed these variables to the main function that would later insert the values into the database; ensuring to return null for any values that were not present in the JSON object.

I also ended up uploading my populated database to my web hosting company, after first having created a new database via the [CPanel](https://cpanel.com/) dashboard that HostPresto provide access to. I also had to allow remote connection to the MySQL server, which is carried out by adding access hosts (my device IP addresses) to the ***Remote MySQL*** section of the dashboard.

Then, under the software tab I decided to see how difficult it would be to setup a Flask application with my web hosting as well. CPanel actually provides a lot of simple and straightforward tools for setting up a Python application. I simply had to go to the ***Setup Python App*** section of the dashboard, select the Python version I would be using, the application directory, the domain (which I set as my personal domain for now) and then any additional modules that I would require. I just installed the main ***Flask*** module.

It was then accessible via the app URI, which is just my personal domain followed by the app directory (filmpy).

I also made sure to limit the number of hosts that could access the MySQL database remotely, by specifying the access hosts explicitly.

**Problems encountered**

I had a problem with the YouTube statistics gathering when traversing the JSON that was returned from YouTube’s API. This ended up being a problem with the way in which I was looking for specific data within the JSON returned from the YouTube API. It was erroring because I was asking for it to return data that didn’t exist, i.e. when a YouTube video has likes/dislikes disabled or is no longer available. To mitigate these issues, I just performed simple if statement checks around these JSON search queries that would check first if the JSON object existed and the second, if the object was null. I later moved this to a separate function (***checkIfPresent()***) as I found that I was having to perform this check regularly throughout the application.

**Tools used**

* CPanel
* Google Chrome

**References**

* <https://developers.google.com/youtube/v3/docs/videos#resource>
* <https://opendata.stackexchange.com/questions/3668/is-there-an-api-for-the-oscars-academy-awards-that-lists-past-winners-as-well-as>
* <https://cpanel.com/>
* <http://calderonroberto.com/blog/how-to-deploy-a-flask-python-app-for-cheap/>

**Keywords**

CPanel, Access Host, % Wildcard

**Week 13 (01/01/2018 – 07/01/2018)**

**Objectives**

* Develop SQL queries to transform the downloaded film data

**Description**

This week I have been working on developing the queries to transform the data. This involves taking film attributes such as ***genre****,* ***actors****,* ***production companies***, etc and calculating a numerical value for them. However, it will be useful to take the answers from the questionnaire as an indicator as to the most important attributes (subjectively) that incorporate a film’s success.

I started by thinking about the way in which I wanted to provide these values with a numerical rating. I decided that the best way would be to take every film that had a specific genre for example and to average those ratings in order to provide each individual genre with its own rating.

Firstly, I tried to calculate these values by an ***INNER JOIN*** on the ***films*** and ***film\_genres*** tables, but found that this was not giving me the correct results. So, I changed the query to join three tables together, the ***films****,* ***film\_genres*** *and* ***genres***tables. This would also allow me to take the actor’s name and use it in the output. I ended up matching the ***films*** and ***film\_genres*** tables via the ***film\_id*** value that was present in both tables and then for the ***genres*** and ***film\_genres*** tables via the ***genre\_id*** value that was present in both tables. This logic was basically saying that, for a genre with x id, return the results of where it is present in both the ***films*** and ***film\_genres***tables and then the same logic applied to the other join. I finally used a ***WHERE*** clause, which only returns the data from the tables where the ***film\_vote\_count*** value (how many people have actually voted) is above 999. This would hopefully eliminate any potential outliers, such as with actors who have only been in two small and potentially unheard-of films that have both received a 10/10 rating. Subjectively, these actors should not receive an individual rating of 10.

**Problems encountered**

My initial calculations suggested that running the query I had written would take 2hrs 50mins given how large my dataset is. But this did not end up being the case. I am however, still unsure as to whether the results it produced are accurate.

I also had a few issues with the number of requests my application was making to the YouTube API service. I found a post on [StackOverflow](https://stackoverflow.com/questions/15568405/youtube-api-limitations) that detailed the exact number of requests that users were allowed to make (the quota), but after checking my API dashboard I was not going over this limit. In the end, the only way to mitigate the issue was to recreate the API and to use the new API key.

At the end of last week, I also executed the program for the first time on the Raspberry Pi.



**Tools used**

* MySQL Workbench

**References**

* <http://usir.salford.ac.uk/18838/1/Wessex_movie.pdf>
* <https://stackoverflow.com/questions/15568405/youtube-api-limitations>

**Keywords**

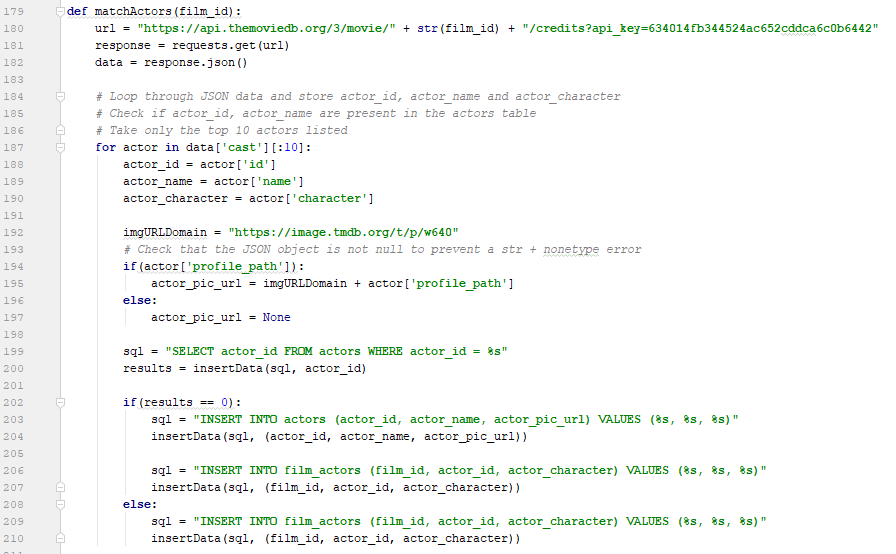
INNER JOIN, WHERE

**Week 15 (15/01/2018 – 21/01/2018)**

**Objectives**

* Acquire director data for each film (limit to top 10)
* Reacquire actor data for each film (limit to top 10)

**Description**

This week I focused primarily on the acquisition of director metadata for each film as well as the reacquisition of actor data for each film, but this time limiting the maximum number of actors acquired to simply the top 10. This is mainly due to these often being the most important actors/actresses of the film and acquiring anymore would simply require a lot more storage and could potentially skew the regression algorithm results/accuracy.

**Problems encountered**

I realised that I had forgotten to retrieve details for the directors for each film. This was most likely an omission due to the fact that each film can have multiple directors and would therefore also require a separate ***directors*** table, joined to the ***films*** table by an association table (***film\_directors***). I also have decided that (even though I previously had concluded that this would not be a problem) acquiring all actors that appeared in a film is unnecessary and will be expensive for any future calculations. I will therefore re-run this specific section (acquisition of actors) at the same time as grabbing the directors. Because of the way in which I built my application, it is quite modular and will be able to run and only retrieve these values. My supervisor suggested that it would be a good idea to justify the reasoning behind limiting actors top 10.

**Tools used**

* PyCharm
* MySQL Workbench

**References**

* N/A

**Keywords**

N/A

**Week 17 (29/01/2018 – 04/02/2018)**

**Objectives**

* Start considering how to build API routes
* Fix actor/director data acquisition

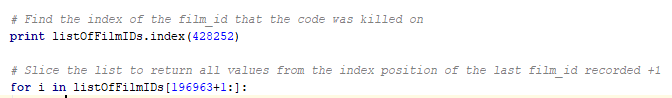
**Description**

This week I focused on researching how to build API routes within Flask. This had me primarily looking at how to utilise POST and/or GET parameters within the Flask framework and how to use the various route registrations to allow either variables to be passed through the API route or queries.

From my research it seems as though, variable sections of a route URL are defined using angular brackets like so: **/films/<film\_id>** and by default accepts a string. However, I needed to accept an integer value in this position, so in order to do just that I need to use a different *converter* like so: **/films/<int: film\_id>**.

My script that was running on the raspberry pi also happened to be killed without any proper indication as to why. So, this required fixing (more information can be found in the *problems* section).

**Problems encountered**

Toward the end of the actor/director data acquisition, the program froze and then was killed what I can only presume was due to the program being inactive for such a long period of time or due to a lack of memory that the raspberry pi has. I managed to restart it and alter the code to only collect the data for the remaining film\_id’s.

This was mitigated using Python’s list slice functionality.

There are also quite a few errors in the csv file, ~313 all of which errored for various reasons one of the main one being that my host could not connect to my database which is hosted with my web host provider. To resolve this issue, once I’ve completed more of the front end I will update the program and create a function that continuously monitors and updates each film’s data, retrieving any missing data automatically.

**Tools used**

* PyCharm
* MySQL Workbench

**References**

* <http://blog.luisrei.com/articles/flaskrest.html>
* <http://flask.pocoo.org/docs/0.12/api/#url-route-registrations>

**Keywords**

N/A

**Week 18 (05/02/2018 – 11/02/2018)**

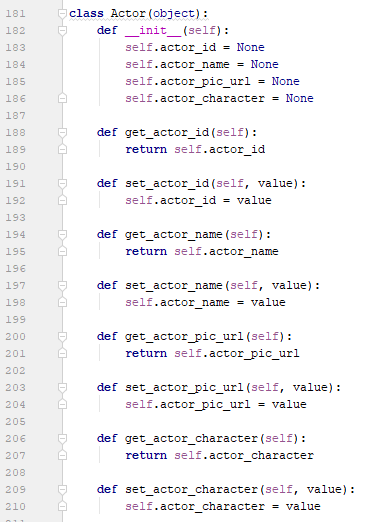
**Objectives**

* Build up the API routes.
* Create classes to hold the film data in Python.

**Description**

This week was an extremely busy week as it was focused on creating the API routes that would interface with the front-end of the application.

Each route essentially required the creation of a class that would store all of the relevant data in the correct format. So for example, for the **/film/<int:film\_id>** route, each film that was searched by it’s id would need to return all of the following information:

* film\_id
* film\_title
* film\_tagline
* film\_overview
* film\_poster\_url
* film\_release\_date
* film\_budget
* film\_revenue
* film\_runtime
* film\_status
* film\_vote\_average
* film\_prediction\_rating
* film\_vote\_count
* film\_trailer\_url
* film\_trailer\_view\_count
* film\_trailer\_like\_count
* film\_trailer\_dislike\_count
* genres
* actors
* production\_companies
* directors

This data also had to be converted into a JSON-format so that the front-end would be able to actually read the data.

Each class was created similar to the example above, where the variables were defined and then the setters and getters were initiated.

**Problems encountered**

The biggest problem encountered was converting my complex Python classes into a JSON format. I continually received the error “Object is not JSON serializable”, which took me a very long time to decipher. Eventually I was directed to a Python library called ‘jsonpickle’ which essentially translates complex Python classes to JSON objects.

**Tools used**

* PyCharm
* MySQL Workbench
* Google Chrome

**References**

* <https://stackoverflow.com/questions/2627002/whats-the-pythonic-way-to-use-getters-and-setters>
* <https://www.w3schools.com/sql/sql_like.asp>
* <http://blog.luisrei.com/articles/flaskrest.html>
* <https://jsonpickle.github.io/>

**Keywords**

Class, API, Routing

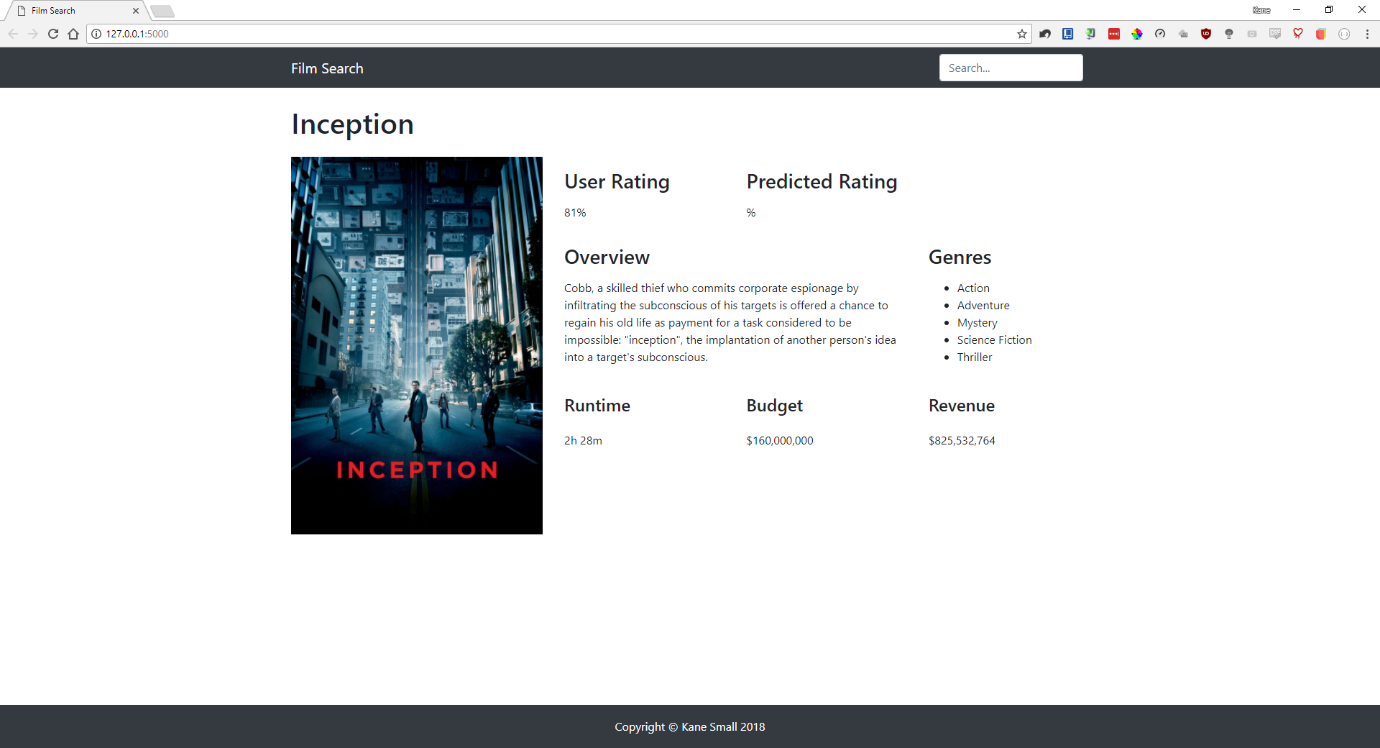
**Week 19 (12/02/2018 – 18/02/2018)**

**Objectives**

* Finalise demo for Tuesday.
* Make a proper start on the prediction algorithm and training the classifier.

**Description**

During my preliminary demo with my supervisor this week, I showcased what I currently had and then we talked about where I could take it from here. We decided that it would be a good idea to have other film data analytics such as which genre has the highest number of well-rated films, or which genre has the highest average revenue, etc. I also suggested that I have an about page that details (at a very low level) how my application functions and how the prediction algorithm will work. I would also like to have a piece of text that will update dynamically, letting the users know that the system (API) has last been updated 2 minutes ago for example. Finally, we discussed the possibility of pulling in additional ratings, i.e. from Rotten Tomatoes and Metacritic.

We also talked about the report and how best to structure it. Primarily to ensure that I have a table of figures after the table of contents, a glossary, an individual reflection, etc. I also decided that I would start using git version control (GitHub), so that I could write up about the many benefits of using a system like that.

**Problems encountered**

* Certain films in my database have stored data that should not be null.
* Need to loop over genres, actors, directors and production companies and update/populate the webpage accordingly for each search.
* Every poster path URL in my database (per film) is incorrect and this part ***w640*** needs replacing with ***w1280***. The same goes for the actor pic URL and director pic URL.
* I also decided that I would like the backdrop image for each film, so in my update function I will request this additional field.

**Tools used**

* PyCharm
* MySQL Workbench
* Google Chrome
* GitHub

**References**

* <https://github.com/kanesmall/final_year_project>
* <https://www.atlassian.com/git/tutorials/what-is-git>

**Keywords**

Version control, repo, commit, merge, branch, master, clone

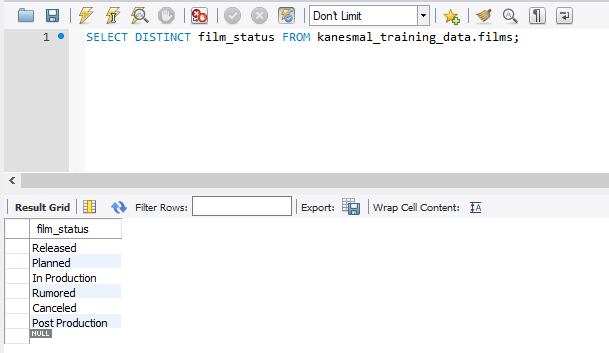
**Week 20 (19/02/2018 – 25/02/2018)**

**Objectives**

* Remove films from the database with the film\_status ‘cancelled’

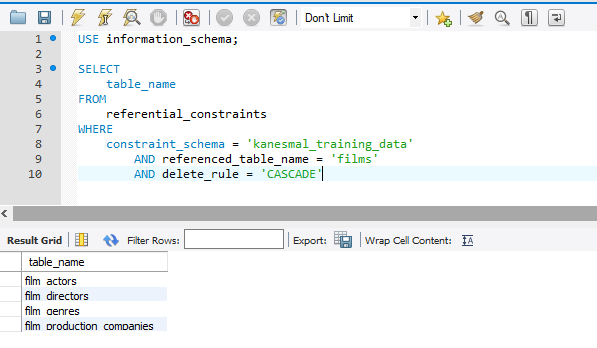
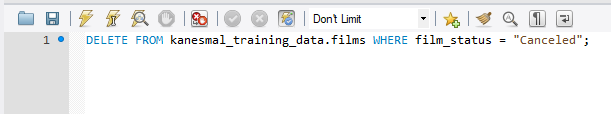
**Description**

I had discussed with my supervisor that it would be a good idea to remove all films that had the film\_status ‘cancelled’, as these films would not be useful for the training or testing phases of the regression algorithm.

First, I had to check the number of distinct film\_status values that there were and carried this task out by using the following SQL:

Initially I attempted to delete the films that had the specific film\_status (cancelled), but before realising that I would also need to delete the referenced tables that had foreign key links to the films table I executed the statement. Luckily, MySQL Workbench prevents the user from deleting any content that is referenced in another table. To mitigate this issue, I essentially utilised the ‘CASCADE DELETE’ option which deletes all of the data from the specified table as well as any referenced table data.

In order to utilise this function though, the existing tables had to be altered to allow this operation to be performed on foreign keys. To do so I simply navigated to **alter table > foreign keys > foreign key options > on delete** and changed this value to ‘CASCADE’. Now by running the following SQL statement, I can ensure that all of the tables that have referenced data appear:

Finally, utilising the following two statements I can see how many films have the ‘cancelled’ status and ensure that only that number of films have been removed from the database:

**Problems encountered**

* Cannot delete films that have references to other tables
* Cascade delete function is confusing at first

**Tools used**

* PyCharm
* MySQL Workbench
* Google Chrome
* GitHub

**References**

* <https://stackoverflow.com/questions/9220821/how-to-set-up-cascading-deletes-in-mysql-workbench>
* <http://www.mysqltutorial.org/mysql-on-delete-cascade/>

**Keywords**

CASCADE DELETE

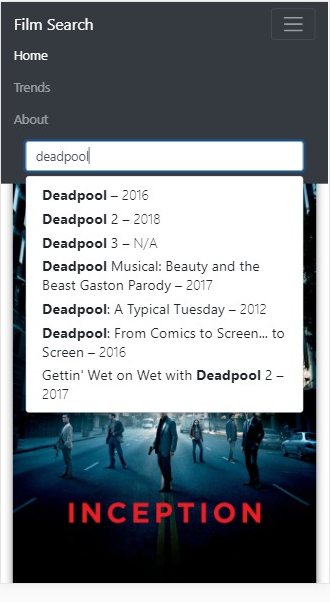
**Week 22 (05/03/2018 – 11/03/2018)**

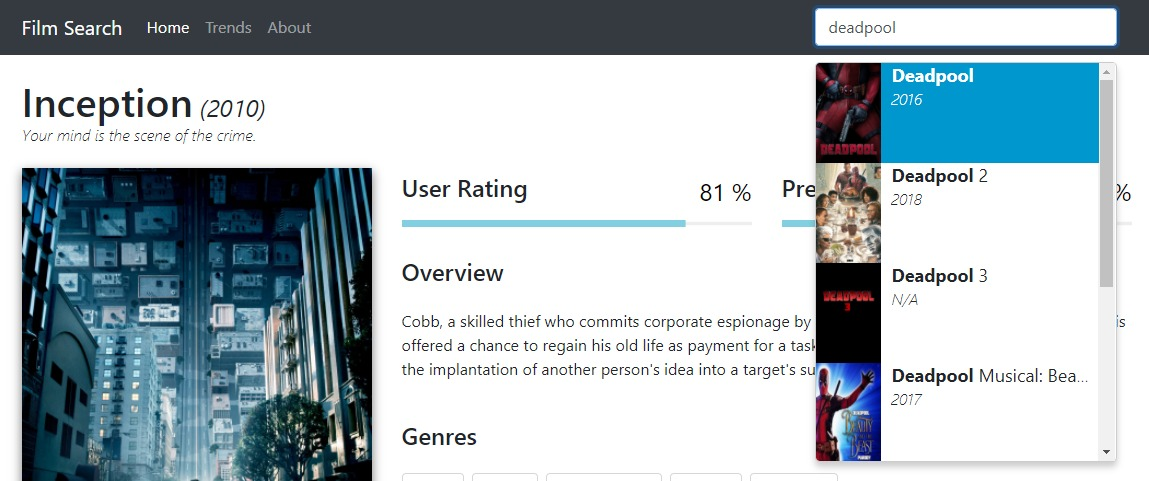
Objectives

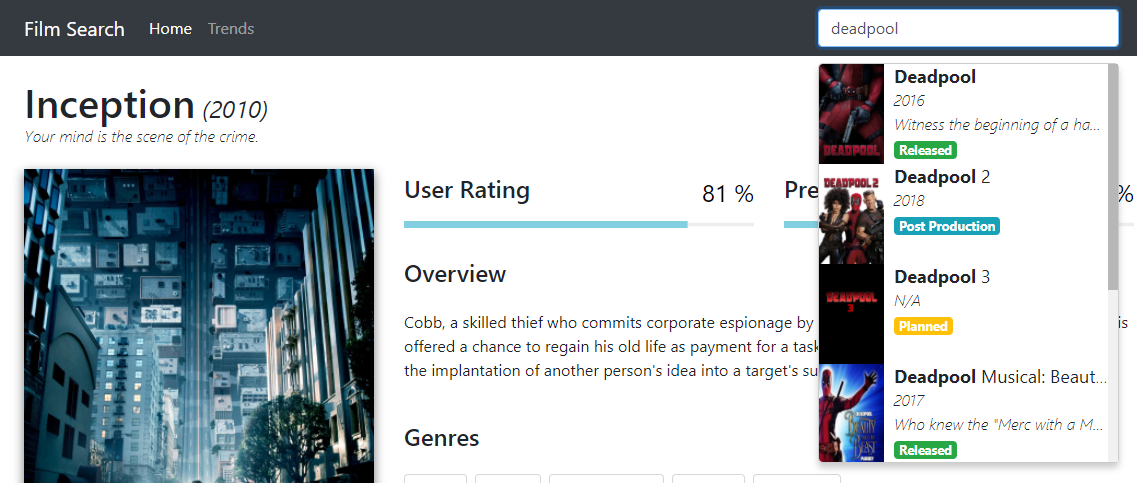
* Search improvements

**Description**

This week I focused on improving the searching experience for users. This primarily involved the actual data that was being displayed to the user within the search results, but also included improvements to the styling and to the ordering of the results.

The end result was a transformation from the following screenshot:

To the screenshot below:

With the final design looking like this:

Essentially, I decided to include more information in the search results which would aid the user in finding exactly the right film that they were searching for. This included, the release year, film tagline (where the overflow was set to *ellipsis* to avoid any positioning errors) and the film status.

**Problems encountered**

* Styling
* Positioning
* Ordering a jsonpickle formatted list

**Tools used**

* PyCharm
* MySQL Workbench
* Google Chrome
* GitHub

**References**

* <https://stackoverflow.com/questions/403421/how-to-sort-a-list-of-objects-based-on-an-attribute-of-the-objects>

**Keywords**

N/A

**Week 23 (12/03/2018 – 18/03/2018)**

**Objectives**

* Implementing a regression algorithm
* Looking at the results/accuracy

**Description**

This week (and some of last week) was aimed at implementing the regression algorithm. I started by building up a script that would be responsible for handling the acquisition of the data from the database, pre-processing the data so that it was in the correct format and training the model so that predictions could be made.

I first had to decide on the right estimator for the job. I ended up selecting a decision tree learner regression model as it is best suited toward projects that have a lot of samples and needs a quantity value to be predicted as opposed to a class or categorical value. A big advantage of decision trees is that there are able to handle features that are both numerical and categorical, which is something that my dataset has.

The DecisionTreeRegressor class function takes in two parameters, a list of X values which are 2-D lists and a list of y values (***X = [[0, 0], [2, 2]], y = [0.5, 2.5]***), so before I could fit the data to the model the data had to first be pre-processed.

The features that had been selected, influenced by the results of the questionnaire, were as follows:

* genres
* budget
* runtime
* trailer\_view\_count
* trailer\_like\_count
* trailer\_dislike\_count
* user\_rating

Due to the method of acquiring the data, I had already ensured that I did not store any duplicate entries into the database. However, I did have a lot of NULL values so these had to be dealt with. When using machine learning algorithms certain algorithms will be able to handle NULL values automatically, but unfortunately the decision tree learner is not one of them. So, this left me with three options:

1. Remove the rows with NULL data
2. Ignore the rows with NULL data
3. Impute the NULL values

I opted for option 3, as this allowed me to keep all of the other data intact and allow it to still be used for the training of the model. Essentially, imputation replaces all NULL values in the dataset with an arbitrary value that isn’t used anywhere else. So, I decided on the value ***-1***, as none of my values in my dataset are negative so the model would be able to distinguish this value from the rest and understand that it effectively represents a NULL value.

Next, I had to encode the genres because each film can have multiple genres attached to it, but the input requires singular feature attributes. So, to mitigate this problem I utilised a technique that I had learned in one of my data mining lectures whereby any features similar to genres are essentially encoded using *one-hot encoding*, which creates a category for every genre there is (all 37 genres) and then represents whether or not a film has that genre by setting the value to 0 or 1 (binary representation).

So, the following table, which represents all of the genres for three separate films:

|  |  |  |  |
| --- | --- | --- | --- |
| Inception | Interstellar | Iron Man | … |
| Action | Adventure | Action | … |
| Thriller | Drama | Science Fiction | … |
| Science Fiction | Science Fiction | Adventure | … |
| Mystery |  |  | … |
| Adventure |  |  | … |

Would be converted into the following format:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | genre\_Action | genre\_Adventure | genre\_Drama | genre\_Mystery | … |
| Inception | 1 | 1 | 0 | 1 | … |
| Interstellar | 0 | 1 | 1 | 0 | … |
| Iron Man | 1 | 1 | 0 | 0 | … |
| … | … | … | … | … | … |

Lastly, I had to ensure that I now reduced my entire formatted feature set to just two dimensions. To accomplish this, I used a technique called PCA (Principal Component Analysis), which reduces the data down to its basic components and strips away any unnecessary parts not required by the model.

So, the following data (genres excluded for visual purposes only):

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| budget | runtime | trailer\_view\_count | trailer\_like\_count | trailer\_dislike\_count | user\_rating |
| 4000000 | 98 | 407034 | 496 | 28 | 6.5 |
| 0 | 110 | 26589 | 71 | 5 | 6.4 |
| 42000 | 80 | 17347 | 36 | 2 | 6.4 |
| 11000000 | 121 | 696398 | 3296 | 82 | 8.1 |
| … | … | … | … | … | … |

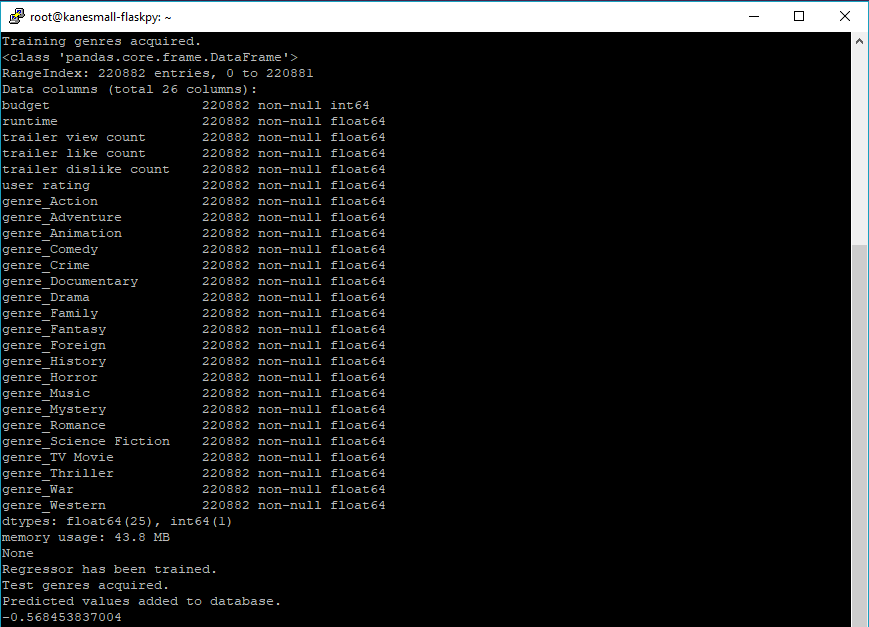
Becomes this, after the target column has been stripped off:

**+**

|  |  |
| --- | --- |
| **principal\_component\_1** | **principal\_component\_2** |
| -0.900681 | 1.032057 |
| -1.143017 | -0.124958 |
| 1.385353 | 0.337848 |
| -1.506521 | 0.106445 |
| … | … |

|  |
| --- |
| **user\_rating** |
| 6.5 |
| 6.4 |
| 6.4 |
| 8.1 |
| … |

So, now all the values have been reduced down revealing the main dimensions of variation and the user ratings can be appended back on. Now the data is ready to be fitted to the model for training purposes.

Using the DecisionTreeRegressor class as part of the scikit-learn library, the inputs are taken in and the regressor begins to build the tree.

The screenshot above is the output from the console, which shows what is happening at each stage, until the regressor has been trained and the prediction values are set in the database. It ends with the explained variance value being printed to the console.

Finally, we have the results. I trained the model three separate times using different training data based on certain parameters. Here are the results:

* Explained variance: 0.1535
  + All training data
  + 4-5 distinct prediction results
* Explained variance: 0.0512
  + All training data, without NULL values
  + 4-5 distinct prediction results
* Explained variance: -0.5685
  + Max tree depth undefined
  + A lot more variation in predicted values

A negative explained variance as can be seen from the final result and is from the final model, can mean that the data being used is imbalanced. This is where the data being trained is extremely different from the data being tested on and is actually the case with the data acquired, as a lot of the test data is NULL. Sometimes negative explained variances can occur even when there is a positive outcome with the values being predicted, which could be the case here, but it is hard to tell.

Using an additional regression metric, MAE (Mean Absolute Error), can tell us whether we have overestimated or underestimated the predictions values based on the actual values. So, if we were to plot the results on a graph, a straight line would be drawn where X=Y and the closer to this line the values are the more accurate the model is. The MAE essentially tells us what the average spread of the errors is, which is this case indicates a positive value that represents an overestimation of the prediction values in general.

**Problems encountered**

* Explained variance is negative
* Could not implement a technique to utilise the actors as a feature with the time provided
* Could not compare other models to see determine which is the most accurate fit for my dataset

Tools used

* PyCharm
* MySQL Workbench
* Google Chrome
* GitHub

References

* <http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html>
* <http://scikit-learn.org/stable/modules/tree.html#tree>
* <http://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>
* <http://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics>
* <http://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_absolute_error.html>

**Keywords**

Decision tree classifier, one-hot encoding, PCA, regression, MAE

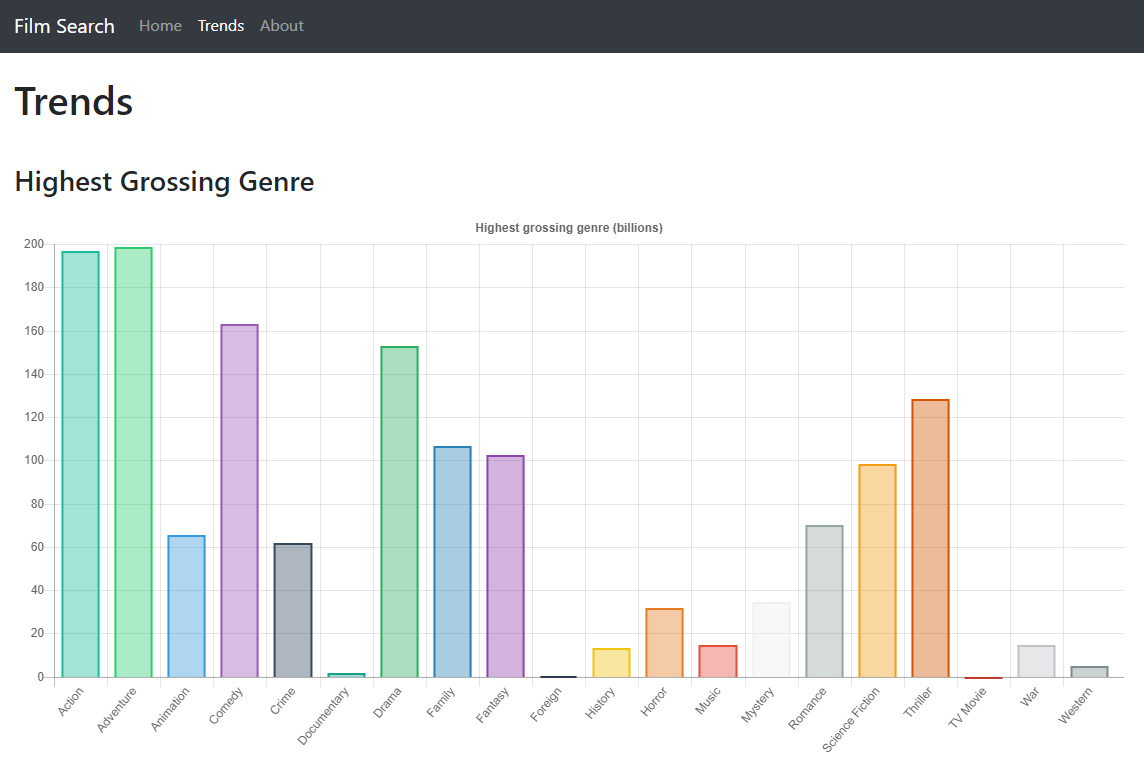
**Week 24 (19/03/2018 – 25/03/2018)**

Objectives

* Focus on building trend graphs

**Description**

This week I focused on building the trend graphs that would be displayed on the **/trends** page. I utilised a library that I had used in the past called *chart.js*, which is an extremely powerful charting and graphing library with a lot of configurable options available.

I started by again, creating additional API routes and classes within my Python Flask script that would be responsible for retrieving the relevant/required data from the database, formatting it with the class, translating it to JSON and then displaying the data to the front-end for the user to see.

The end result is the bar chart illustrated above.

**Problems encountered**

The main problem encountered was the load times of the chart elements. This was to the best of my knowledge, a result of the API having to run the SQL statements on the database which took a long time considering the amount of data it had to query. To mitigate this issue, I essentially created two separate routes for the trend graphs. The first ***/get\_trends/<int:trend\_id>*** would essentially acquire the data from the database by running the required SQL statements and store the JSON formatted results into a .json file on the server (ensuring to overwrite any existing files). The second ***/show\_trends/<int:trend\_id>*** would locate the relevant file based on the ***trend\_id*** that had been passed and load the data directly from the json file without having to perform any expensive SQL operations.

**Tools used**

* PyCharm
* MySQL Workbench
* Google Chrome
* GitHub

**References**

* <http://www.chartjs.org/samples/latest/>

**Keywords**

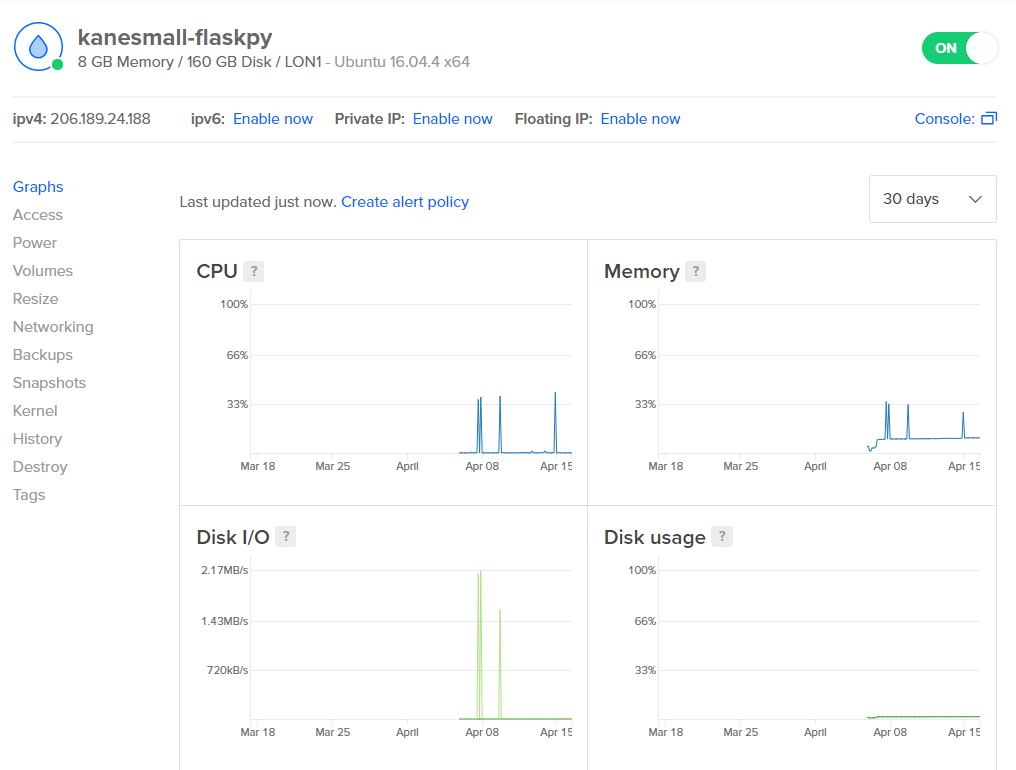
N/A

**Week 25 (26/03/2018 – 01/04/2018)**

**Objectives**

* Move existing infrastructure over to a DigitalOcean Droplet

**Description**

I decided to move all of my existing Flask application infrastructure and database over to a DigitalOcean droplet. I discovered that as a student through GitHub’s student developer pack I was able to receive $50 worth of credits to use on DigitalOcean and I was also able to acquire a free domain for a year with Namecheap. This enabled me to move my application to a much more powerful, stable and reliable platform. This would also provide a much more maintainable solution for the future, as having my raspberry pi on consistently is not the most viable solution especially since power cuts can occur and I do not have a redundant backup power supply such as a UPS.

In order to connect to the server I utilised two applications, Filezilla and PuTTY. Filezilla is an FTP (File Transfer Protocol) client and PuTTY is a terminal emulator that can communicate to the server via SSH (Secure Shell). Filezilla is used to allow me to update the copies of files stored on the server with files that have been updated on my local machine. SSH allows me to run Python scripts on the server, configure the Nginx web server, configure the Flask application and also configure the database.

**Problems encountered**

Setting up an Apache web server proved difficult, but in the end was resolved by instead setting up and configuring an Nginx web server on the droplet following the tutorial/guide that DigitalOcean provides.

**Tools used**

* PyCharm
* MySQL Workbench
* Google Chrome
* GitHub
* Filezilla
* PuTTY

**References**

* <https://education.github.com/pack>
* <https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-16-04>
* <https://filezilla-project.org/>
* <https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html>

**Keywords**

DigitalOcean Droplet, UPS, SSH, FTP, Nginx

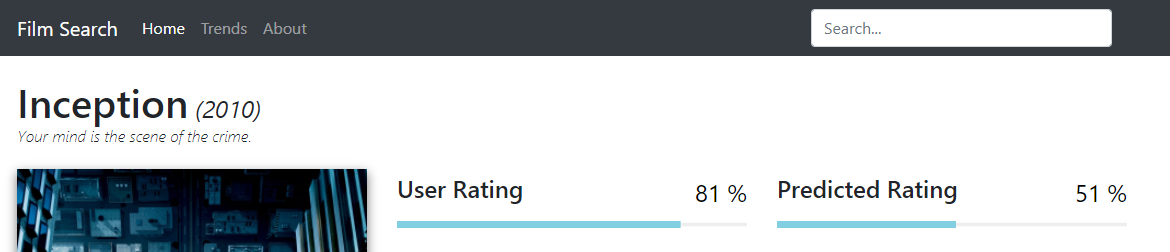
**Week 26 (02/04/2018 – 08/04/2018)**

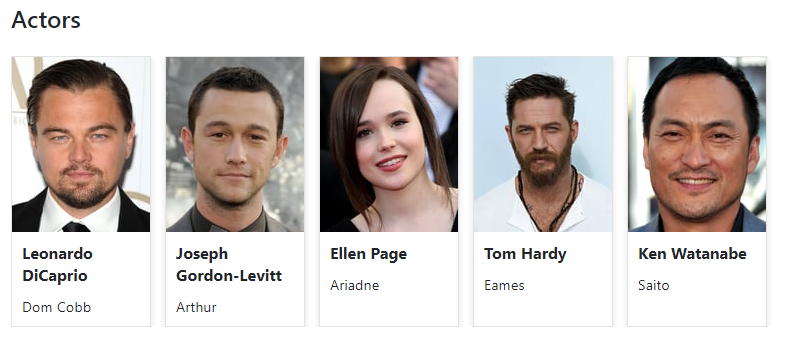
**Objectives**

* Improving design

**Description**

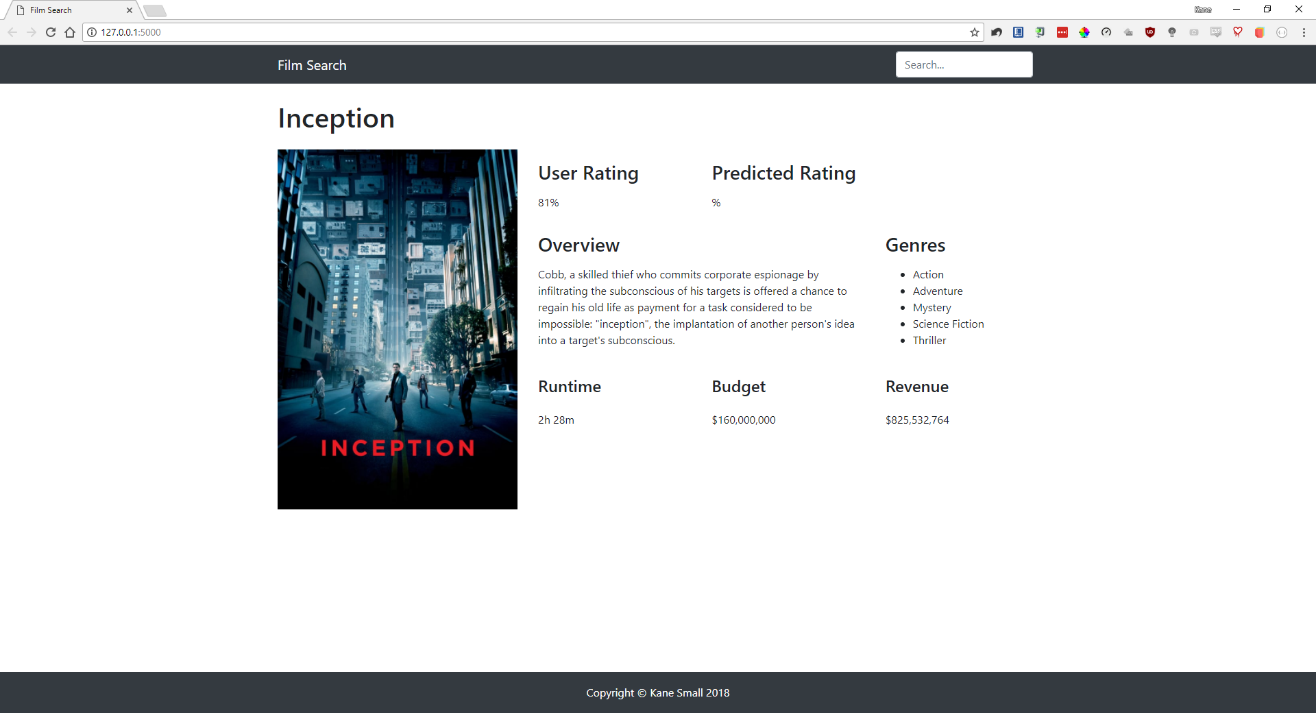
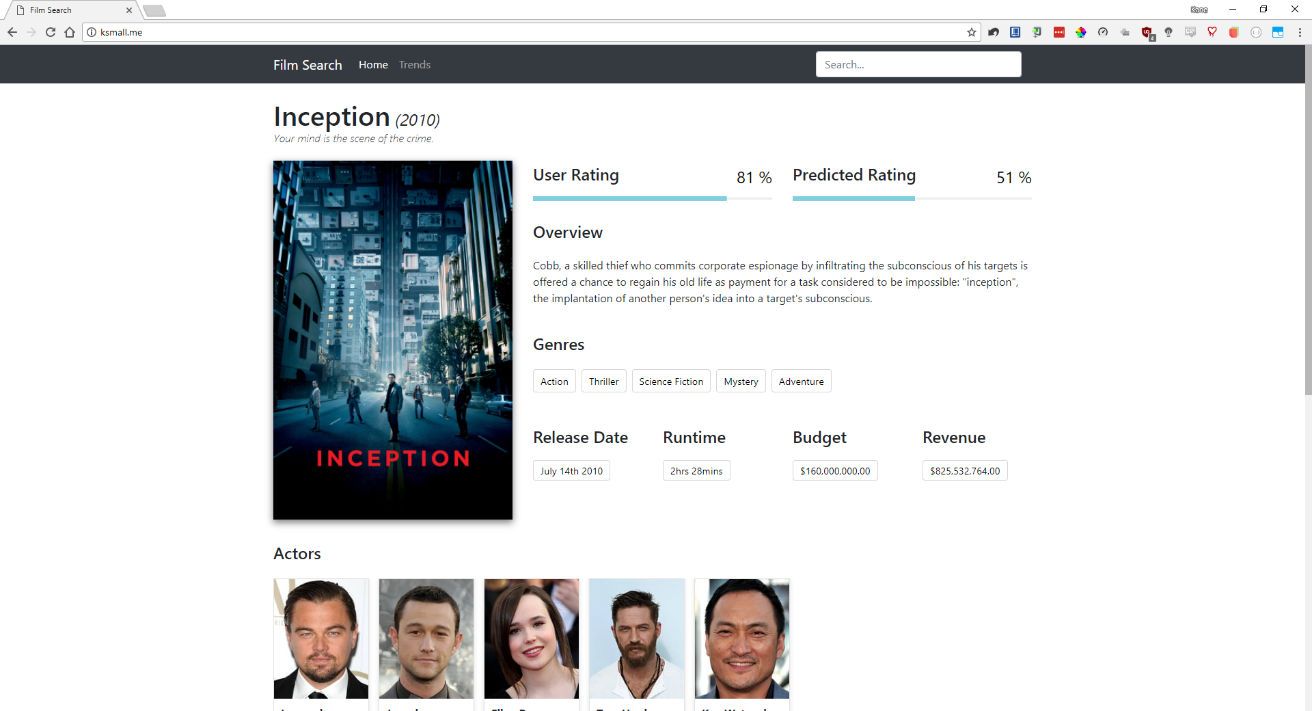
This week I primarily focused on improving my overall front-end design. This revolved around making the information on the page a lot more user-friendly and laid out in a format that was concise and flowed in a more design and user-friendly manner.

This was mainly a matter of working with the CSS to style the elements on the page, but also involved the use of external libraries such as *progressbar.js*, which provides a sense of a more polished and professional-looking application.

Both the user and prediction ratings utilised the progress bar library, which also included animations whenever the data is updated.

As depicted above, I also worked on displaying the top 5 actors in a succinct manner, ensuring that the actor pictures were easy to see and not blurry and that the names and characters were also readily available to the viewer, but were also separated.

Ultimately, my application ended up transitioning away from this screenshot below:

To this screenshot below:

**Problems encountered**

* Formatting the progress bars was extremely difficult, especially when it came to positioning the dynamic percentage value.
* Styling can sometimes overlap, so ensuring that each element has a unique class and/or id value is essential.

**Tools used**

* PyCharm
* MySQL Workbench
* Google Chrome
* GitHub
* DigitalOcean
* PuTTY
* Filezilla

**References**

* <https://kimmobrunfeldt.github.io/progressbar.js/>

**Keywords**

N/A

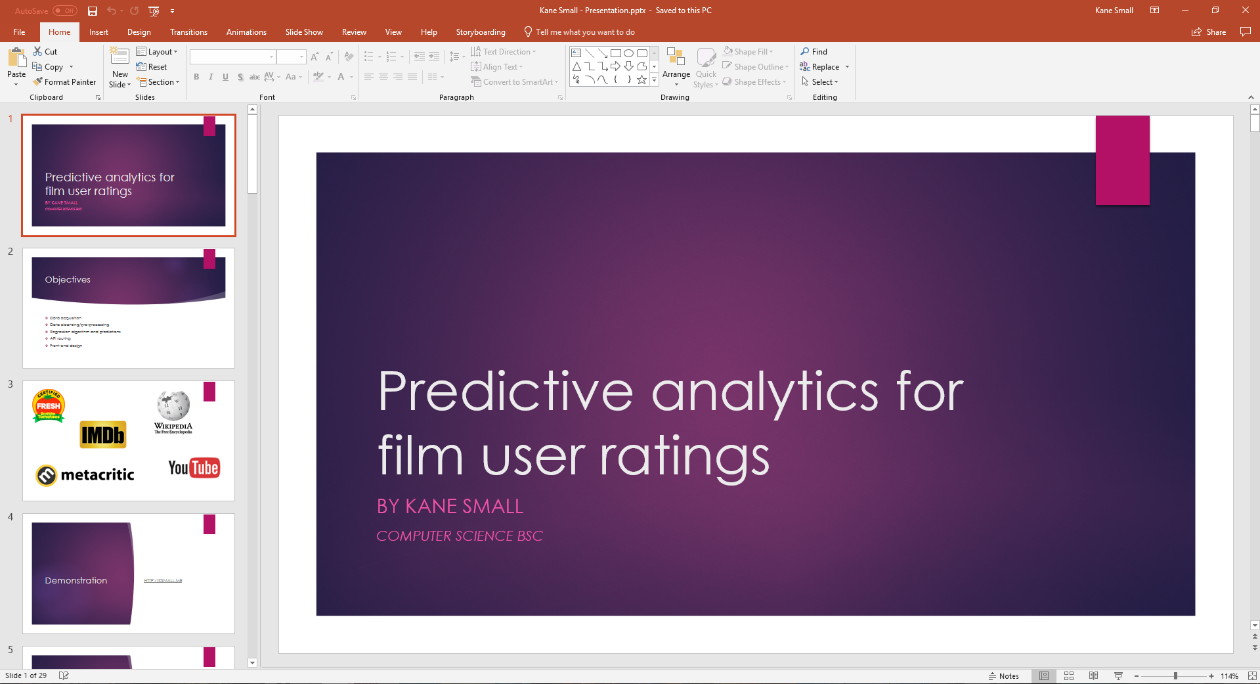
**Week 27 (09/04/2018 – 15/04/2018)**

**Objectives**

* Create PowerPoint presentation

**Description**

This week has been focused on creating a presentation for the presentation/demonstration next week. On Tuesday I had a practice presentation with my supervisor, which helped me to receive some extremely valuable feedback. I recorded all feedback using a voice recorder so that I could play it back when I got home, as I knew I would forget what had been said during the session otherwise.

I managed, to my surprise, to speak for 25 minutes during the practice session, which was obviously too long but, in my opinion,, it is easier to cut down time than to have to think of more to say. Using the 50 minutes of feedback that I received from my supervisor I wrote down all of the parts of the presentation that either needed removing or amending and then carried out these changes.

**Problems encountered**

* Needed to cutdown my presentation
* Went into too much detail regarding code and/or functions or classes that had been created

**Tools used**

* Google Chrome
* Microsoft PowerPoint

**References**

* N/A

**Keywords**

N/A

**Week 28 (16/04/2018 – 20/04/2018)**

**Objectives**

* Giving my presentation
* Finalising the report

**Description**

This week has primarily been focused on finalising the report, but on Tuesday I also had to give my presentation to a panel of 4 people.

From my perspective, I gave a detailed presentation which outlined the main objectives of the project and then dove deeper into how I achieved them on-by-one.

**Problems encountered**

The stress has been extremely overwhelming this year, especially in the last few weeks and it has been causing me to shut down due to my disabilities.

**Tools used**

* Google Chrome
* Microsoft Word
* Microsoft PowerPoint

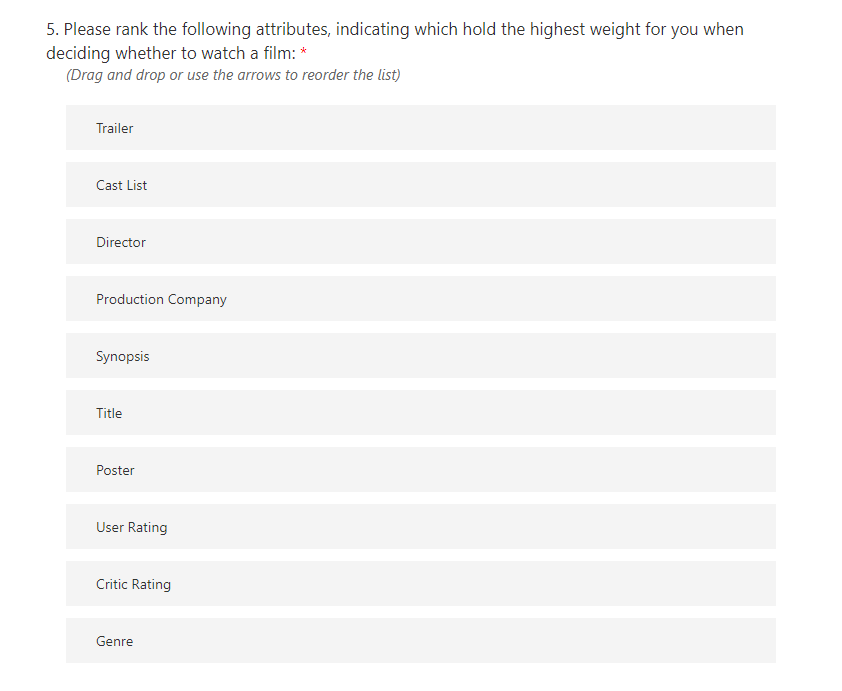
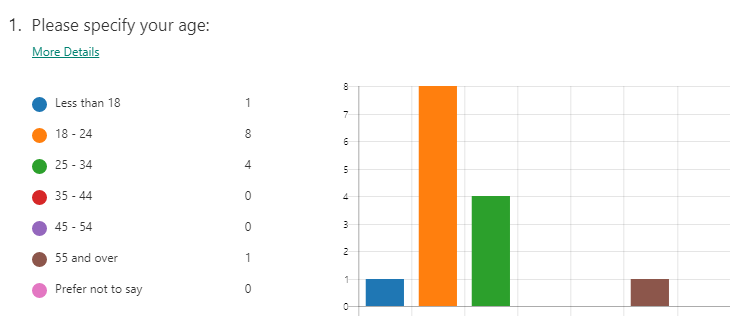
**References**

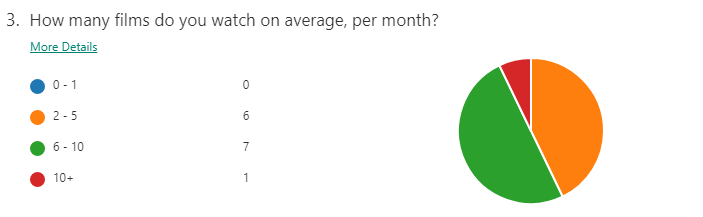
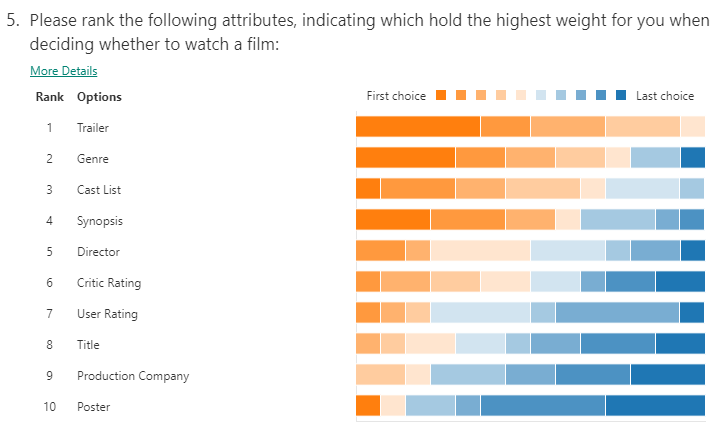
* N/A

**Keywords**

N/A

## Appendix 3: Questionnaire





## Move mouse over imageAppendix 4: Scikit-learn algorithm cheat sheet