## Solution description

Our solution is to create an analysis of bodies of text by classifying keywords in the text that signify the text belongs to or is describing a specific category or genre of writing. We will do this using the sklearn python library, namely the Pipeline module for training and the GridSearchCV module for testing.

We will likely select a list of common stopwords such as ‘and’, ‘or’, ‘then’, ‘this’, ‘that’ in order to avoid confusing the model and decreasing the margin for error/confusion. GridSearchCV should tell us which stop words (i.e. none, English or custom) result in the highest accuracy.

The data will require pre-processing as there are missing or incomplete values for many of the records, and some of the records have multiple genres, which will not only require us to split the genre string by commas into a list of strings, but design our model so that it can potentially receive multiple classifiers for one body of text; one solution might be to split one record with three categories into three separate records with one category each. A multilabel binarizer is another solution. As well as pre-processing our data so that the data only contains one label per entry. It has also not been taken off the table that we could mix multiple datasets together in order to have a larger variation of writing styles and (hopefully) more accurate predictions.

We have a choice of classifier algorithms available for us to choose from, some examples include: naïve bayes, random forest, logistic regression and a few others. It will become clearer to us which one to use as development continues.

We will be building a web API for our model, this will either be built using flask, or Django – both of which are python based web frameworks. Our GUI will either be built using either python’s ktinter GUI library or a web based GUI which could potentially be a lone html file making ajax requests to the api.

## Data Description

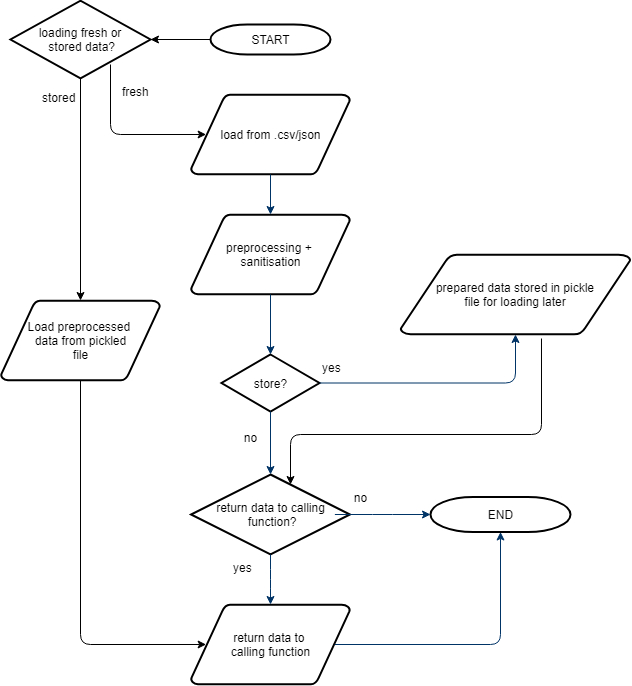
Our data consists of bodies of text (in the case of our data, movie synopsis or a “plot”), and matching genres for the movies, movies which have a genre will be used as a training set. Some movies have missing categories, these records can be separated off and used as a practical test set as opposed to a prediction accuracy test set. The data includes these features:

* Release year
* Title
* Origin/ethnicity
* Director
* Cast
* Genre
* Wiki Page
* Plot

Some of the records do not include a plot, these records will be cleansed in the data pre-processing. There are 34894 records in total. 17% of the records belong to the drama genre, 17% have unknown genres, there are supposedly 2263 unique genres, however, this number counts two, comma separated genres as an entirely new genre. For example, if a record has “historical, thriller” as its genre, this will not count as thriller or historical, but an entirely new entity, the same rule applies for “short action/crime western” for example. All features apart from genre and plot will be dropped from the table.

## Solution diagrams

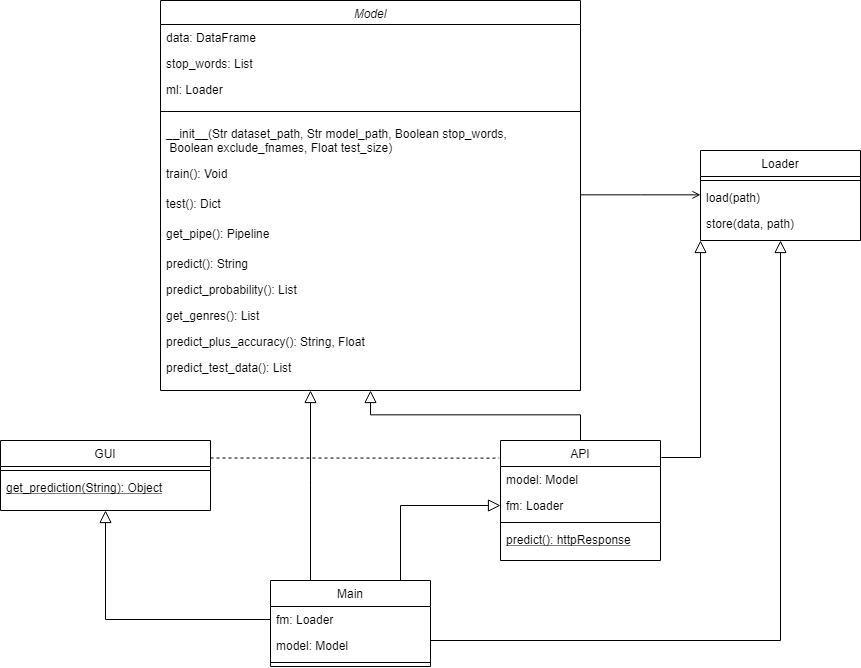
### Data loader



### Model



### Class diagram (main, api and gui are rather entities/non-class-scripts than classes)



## Solution Motivation

### Object oriented approach

Breaking our code down into separate files and classes allows us to take an object oriented approach that makes our code more modular. This allows other scripts to import functionality from our code and saves us from re-writing code as well. Classes can contain their own global variables too which makes them extremely useful for bilateral use (say for example you want two different models with the same data in them. Using an object oriented approach also allows us to have a “main” script (see figure above) that can instantiate/call all the individual pieces of code in our project to work in tandem while being able to skip unnecessary steps and pass different variables as parameters so that we don’t have to rewrite hardcoded variables each time.

### Library choice

Sklearn is a brilliant library for the problem we have chosen to solve, for text categorisation a deep learning approach for example would be overkill and would also not allow . Sklearn comes built in with many of the most prominent text classification algorithms and allows us to easily switch between them.

Flask and Django are both acceptable choices for API libraries, we decided we will use one of them as they are python based, like the rest of our code, allowing for easier integration.

Pickle allows us to store models and pre-processed datasets in file form for later use. This allows us to run the API for example without having to train the model first which would both be a waste of processing power and time in many cases.

### Pipeline & GridSearchCV

GridSearchCV can be very useful for working out what the best settings to use on our pipeline are. It will give feedback one elements we choose such as which stop words to use, all the way up to which classifier to use.

Pipelining allows us to very easily swap out our vectorizers, our transformers, feature selection methods and our classifiers. This pairs up nicely with grid search as it allows us to quickly change a massive part of our model such as the classification algorithm (usually in one line) and then give us instant feedback on the change in accuracy.