## 2.3. Existing FYPs

For each project:

* Author
* Web address
* Screengrab (options)
* 100-200 word description of proejct

100-200 word description of how it inspires your project.

# Design

This section details the technical architecture chosen for this project, a diagram of it and all other design documents including Use Case and Class diagrams and an Entity Relationship Diagram.

## Technical Architectures

### Model View Controller

The Model View Controller (MVC) architecture is used across a wide range of applications where there is a need to provide a User Interface through a desktop or web front-end. It is a three-tier architecture which uses the Controller, comprised of several classes such as a Command Factory class and Command, Service, and DAO classes, to pass information between the View, i.e. the front end, and the Model, i.e. the backend.[44]

This ensures the separation of roles between the different sections of code in a project. This makes it easier to divide up the work in a project as team members can focus on specific sections without worrying too much about the other parts enabling better development and testing.

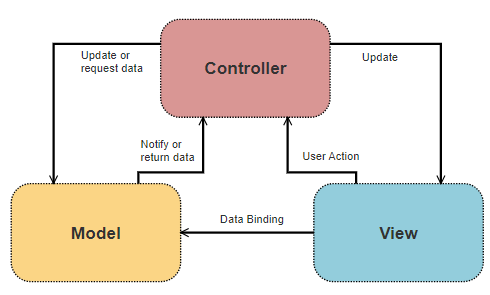


Figure 12 Model View Controller

### Model View Template

For this project I have chosen to use the Django framework which uses its own modified version of the MVC called the Model View Template (MVT). In this adaption Django takes care of the Controller role and replaces it with the Template section, which takes the role of the presentation layer by containing all the HTML, CSS and Forms files while the View section deals with all business logic and handles all requests from and responses to the User. The Model section stays the same and deals with everything to do with the database.[45]

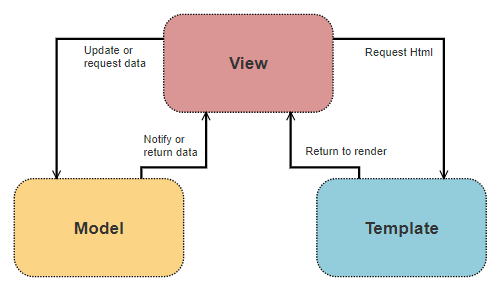


Figure 13 Model View Template

## Technical Architecture Diagram

As this is a web-based data mining application I have adapted the MVT architecture to suit this project by adding another layer between View and Model, called Mining, to consider the data mining and machine learning aspects of this project. This layer deals with everything from data pre-processing to the creation and evaluation of the machine-learning models to analysis of new data passed to it.

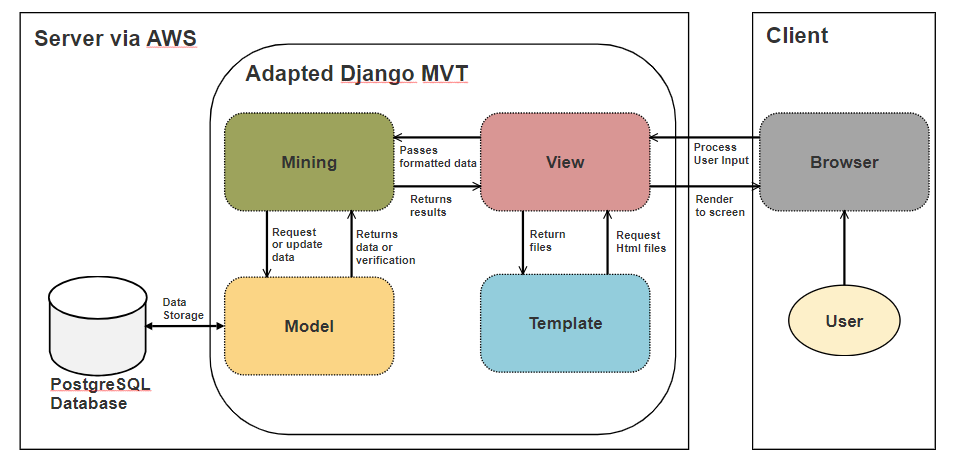


Figure 14 Application Technical Architecture

## Other Design Documents

### Use Case Diagram

The use case below details how a user will interact with the system. The user can enter in their own Twitter username or any other one they wish, be it a celebrity’s, one of their friends or any other account they know off. They will then be able to view the results about how likely that account is a bot and be able to either share those results on social media or download them into a file.

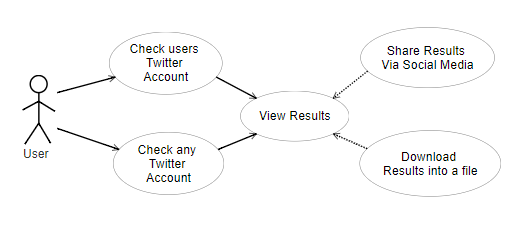


Figure 15 User Use Case Diagram

### Entity Relationship Diagram

This diagram is subject to change due to trying to achieve a more accurate result in further iterations of the development cycle.

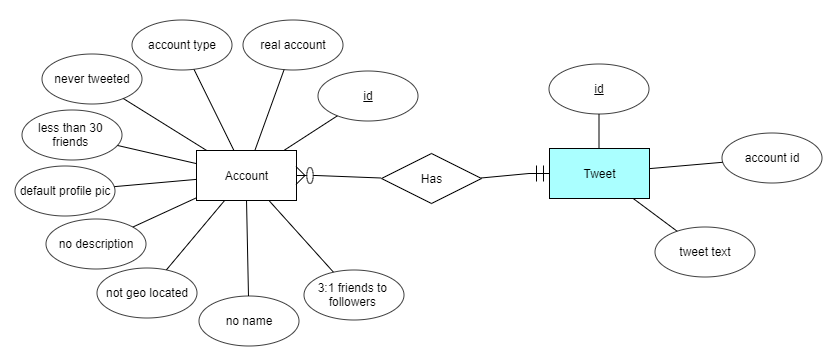


Figure 16 Entity Relationship Diagram

# Prototyping and Development

This section explains what prototyping and development has been completed to date, giving details on building the web front-end and the creation of a basic machine-learning model.

## Vertical Prototype

The prototyping for this project revolves around creating a vertical prototype, which shows the basic structure and functionality of both sub-sections of the project. This will then be reviewed over the December break and if found satisfactory, built upon heavily to create the final application.

### Web Front-End

For this section of the vertical prototype, the goal was to have a working web-app that could connect to the Twitter API and retrieve data from it, in this case the twenty most recent tweets from the account linked to the username chosen by the web-app user.

Below are two screenshots of the working web page which asks for the user to input a Twitter username then retrieves and outputs the data received back from the Twitter API to screen.

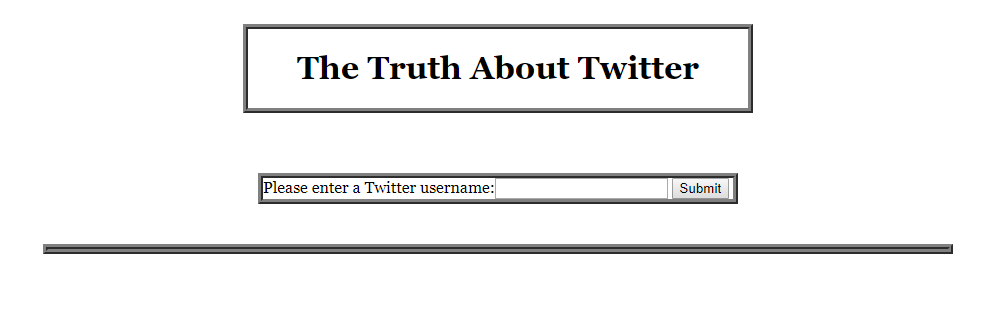


Figure 17 Web Front End

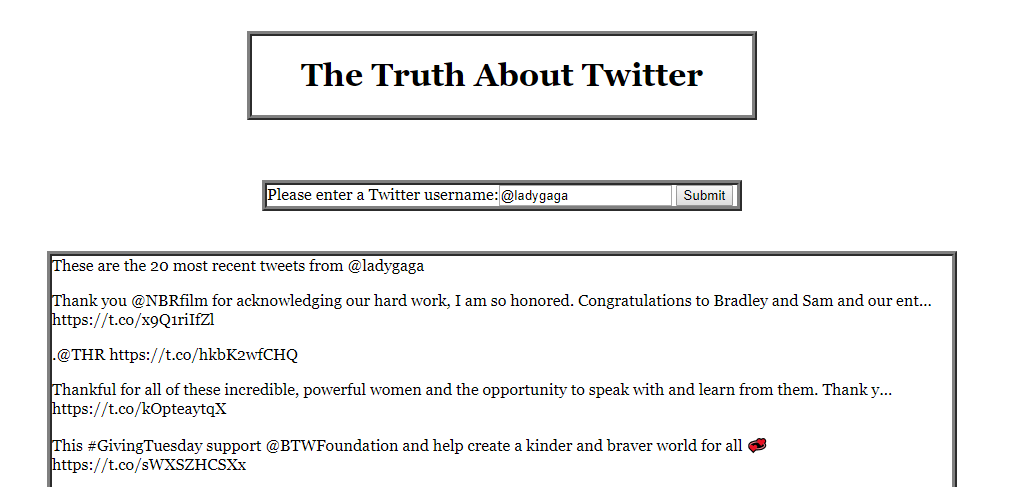


Figure 18 Web Front End with returned tweets

### Basic Machine-Learning Model

For this section of the vertical prototype, the goal was to have a basic machine-learning model that uses data from the acquired datasets and using k-fold cross validation, uses the entirety of the selected data as training and testing sets, outputting, to console, the average accuracy result across the partitions, repeated five times.

Seven features were chosen for this initial model and will be re-evaluated and changed further into development: Whether the account has the default profile picture, has a screen name, has a description, has less than 30 friends, has more than 1000 friends, has never tweeted, the account is geo located and the ratio of friends to followers is 3:1.

A Naïve Bayes classifier with Bernoulli distribution was chosen for this as it a good classifier to start with any data mining project and the inputs are of a binary format, 0 and 1’s with 2000 accounts chosen, 1000 random genuine accounts and 1000 random traditional bot accounts. Below is the output from this completed section the accuracy sitting at around 60% depending on which accounts are selected at the start.

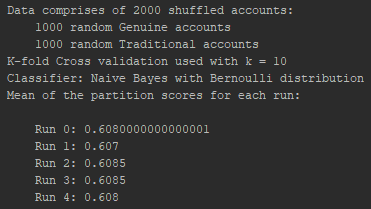


Figure 19 Results of Model being run

## Development

This section deals with all the development done to date in the creation of the vertical prototype using the Django framework and various python libraries such as Tweepy and Scikit-learn.

### Web Front-End

There were several steps to creating the web front-end shown above in the previous section and each will be explained with code snippets where needed.

A new Django project was created within PyCharm Professional, allowing a lot of the tedious groundwork for a web application to be taken care. This meant a bare-bones skeleton app was ready for use and to be built upon.

Next a Twitter developer account was created using my own Twitter account and a Twitter app created, noting its Consumer Key and Secret Key. An Access Token and Access Token Secret were created and noted next. These were outputted to a json file: twitter\_credentials.json, using twitter\_credentials.py, for use later.

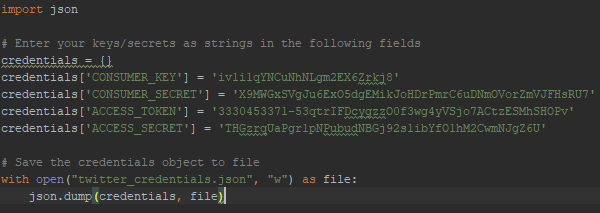


Figure 20 Saving Twitter credentials to file

Next are all changes made to the Django skeleton code:

* Creating a basic form called UsernameForm, in forms.py, to take in user input.



Figure 21 Basic form

* This form was added to views.py within the index method. If the request method was POST, the existing form instance was read in and the user input read into username for use later otherwise a new form instance was created.

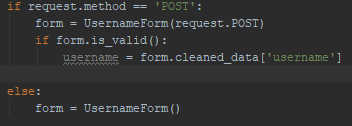


Figure 22 Creating or reading in form

* The values from twitter\_credentials.json are read in and are used in combination with the username inputted in the form to access the Twitter API to return the most recent twenty tweets from that username.[22]

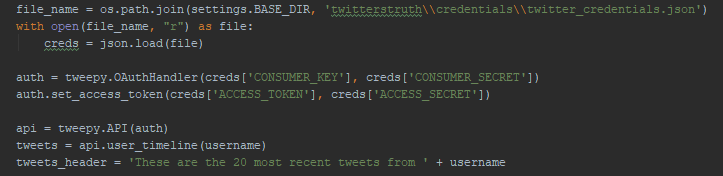


Figure 23 Twitter Authentication and tweet retrieval

* Then a response is returned, asking to render index.html with passed variables: username, tweets and tweets\_header for use in the Html file.



Figure 24 Render Index.html with variables passed

* The templates section of settings.py was altered so to know where to look for template files suck as index.html

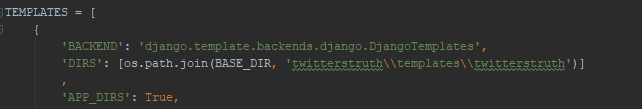


Figure 25 Template directory added

* Index.html was been altered to show the form and tweets using the variables passed to it via views.py

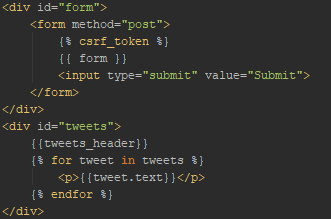


Figure 26 Index.html

* Index.py uses a CSS file web\_style.css, which is located within the twitterstruth/staticfiles/css sub-directory. [46]



Figure 27 Load web\_style.css

* To enable this CSS file, and others within the twitterstruth/staticfiles sub-directory, to be found urls.py was altered [46] :

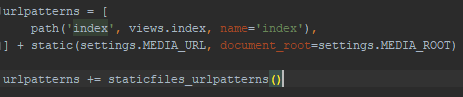


Figure 28 Enabling CSS file load part 1

* As was settings.py [46]:

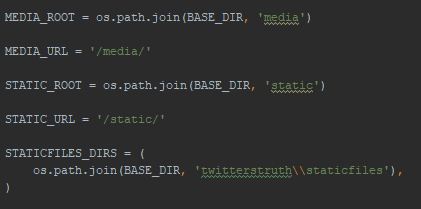


Figure 29 Enabling CSS file load part 2

### Basic Machine-Learning Model

There were several steps to creating the basic machine-learning model shown above in the previous section and each will be explained with code snippets where needed.

* As the files for this section are being run separately but still need access to certain files within the Django framework, the environmental variable, DJANGO\_SETTINGS\_MODULE, must be set and Django setup within each file:

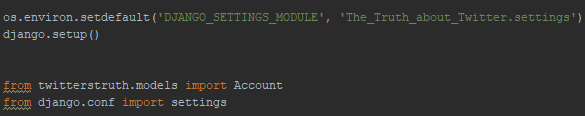


Figure 30 Enabling Django file use

* The file read\_store.py deals with all reading in and storing of the datasets.
* Each dataset is read in from their CSV files one at a time using the pandas library [18], with the tweets files being ignored until the next phase of development.

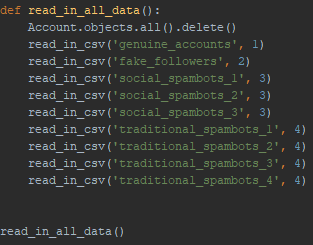


Figure 31 Read in all data

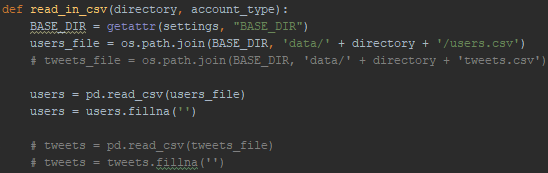


Figure 32 Read in single dataset from CSV

* Checks are done on certain columns in the dataset and binary outputs given depending on the result to form the data going into the database:

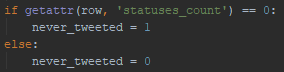


Figure 33 Data check example

* The data is then read into the database:

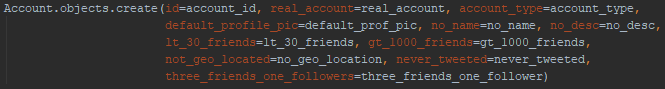


Figure 34 Adding data to database

* In machine\_learning.py, data is read out from the database and passed to through a model, giving output to the console of the model’s accuracy with each run.
* All the randomly chosen, accounts are read out from the database, being split into features and corresponding targets lists:



Figure 35 Get 2000 accounts

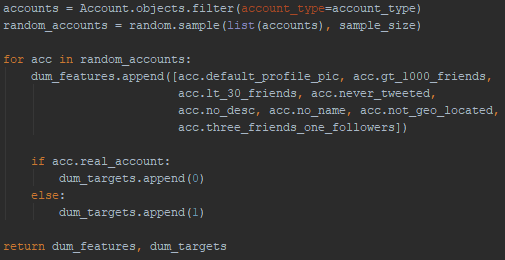


Figure 36 Return random sub-set of accounts from database

* The lists were converted into NumPy arrays [19] and the sklearn library used for k-fold cross validation and classifiers initialisation [17]:

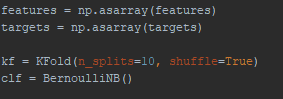


Figure 37 Convert arrays and initialise model

* This model was run five times, with the mean accuracy score across the partitions outputted each time:



Figure 38 Run model and output results to screen

# Testing

This section will explain how testing for the various parts of this project will be done. The testing is split up into 3 parts: The data mining and machine learning section, the web front-end and lastly the fully integrated combination of these two parts.

## Data Mining & Machine Learning section

This part will be employing the K-fold cross validation procedure, talked about during the research stage of this document, to perform my testing on all the models that I build or tweak in this project.

This method involves partitioning up a dataset into K partitions of equal size and for each one, taking that as the testing set with the remaining partitions as the training set.

The dataset I am using is already divided up into multiple sub sets. One of these represents a collection of real Twitter accounts while the rest represent different types of bot accounts. In turn, each of the bot datasets will be mixed separately with the dataset of real accounts and K-fold cross validation will be applied ensuring that the models are trained and tested using the entire mixed dataset each time.

## Web Front-End

This part will be relatively simple and as such, until the previous part is integrated with this one, taking on the role of tester and trying to break every part of it should suffice.

## Web-based Data Mining Application

Once everything is integrated together I will again take on the role of tester while also asking for several testers from my friends and family to help find any bugs that might arise from the integration stage or missed at an earlier stage.

# Bibliography

1. Gamma, Erich; Vlissides, John; Johnson, Ralph; Helm, Richard; (1994), “*Design Patterns: Elements of Reusable Object-Oriented Software*”, Boston Massachusetts, Addison-Wesley Professional
2. The Django Book; (2018), The Model-View-Controller Design Pattern, djangobook.com/model-view-controller-design-pattern/, Date Accessed: October 2018
3. Catherine; (2013), Include CSS and Javascript in my django template, stackoverflow.com/questions/15491727/include-css-and-javascript-in-my-django-template, Date Accessed: November 2018
4. Azevedo, A. I. R. L.; Santos, M. F.; "*KDD, SEMMA and CRISP-DM: a parallel overview*", IADS-DM, (2008).