

# **Predicting the results of future UK political opinion polls using sentiment analysis of tweets**

Author: Samuel Wright

Year of Study: 3rd Year BSc

Project Supervisor: Torsten Mütze

August 16, 2021

# Abstract

Machine learning is a relatively new method of data analysis that has gained a lot of traction in many fields with its ability to make accurate, intelligent predictions on trends in data over time. One such field being the political sphere, as the main method of gauging public political opinion is through polls and elections. This project takes current Twitter data (user tweets) and quantifies them through sentiment analysis as a measure of the public's current opinion on prominent UK political parties. Tweets from the UK are collected, tokenized and evaluated using Python's Natural Language Toolkit (NLTK) and a custom Naive Bayes classifier. The resulting sentiment figures are combined with past poll results to train a regression model that maps the current general sentiment for a party to a percentage vote said party might receive if a poll were to happen on that day. The model will improve in accuracy over time as more polls take place and more tweets are collected, however the current linear regression model shows the predicted linear relationship between positive sentiment and increased vote percentage.

**Keywords:** UK opinion poll, Python, NLTK, Naive Bayes Classifier, Linear Regression, Sentiment Analysis, Twitter

# Contents

<b>1</b>	<b>Introduction</b>	<b>6</b>
1.1	Context . . . . .	6
1.2	Project aims and contribution . . . . .	6
1.3	Motivation . . . . .	7
1.4	Related works . . . . .	7
<b>2</b>	<b>Requirements</b>	<b>8</b>
2.1	Functional requirements . . . . .	8
2.2	Nonfunctional requirements . . . . .	8
<b>3</b>	<b>Constraints</b>	<b>10</b>
3.1	Hardware . . . . .	10
3.2	Software . . . . .	10
3.3	Data . . . . .	10
<b>4</b>	<b>Background Research</b>	<b>11</b>
4.1	Social media influence and validity . . . . .	11
4.2	Data quality assessment . . . . .	11
4.2.1	Accuracy . . . . .	11
4.2.2	Completeness . . . . .	11
4.2.3	Consistency . . . . .	12
4.2.4	Time-related Dimensions: Currency, Volatility, and Timeliness . . . . .	12
4.2.5	Subjective value . . . . .	12
4.3	Political science and poll influence . . . . .	13
4.4	Sentiment analysis and classification algorithms . . . . .	13
4.5	Regression models . . . . .	14
4.6	Polynomial regression . . . . .	15

<b>5</b>	<b>Project Management</b>	<b>18</b>
5.1	Methodology . . . . .	18
5.1.1	Time constraints . . . . .	18
5.1.2	Flexibility . . . . .	18
5.1.3	Project size . . . . .	19
5.1.4	Past experience . . . . .	19
5.2	Schedule . . . . .	20
5.3	Tools and packages . . . . .	21
5.3.1	Software . . . . .	21
5.3.2	Hardware . . . . .	22
5.4	Risk management . . . . .	23
5.4.1	Data loss . . . . .	23
5.4.2	Time constraints . . . . .	24
5.4.3	Software restrictions . . . . .	24
<b>6</b>	<b>Design</b>	<b>25</b>
6.1	Overview . . . . .	25
6.2	Stream listener . . . . .	27
6.3	Tweet sorting . . . . .	29
6.4	Web scraper . . . . .	29
6.5	Sentiment analysis . . . . .	29
6.6	Relational table manager . . . . .	30
6.7	Prediction model . . . . .	31
6.8	Database design . . . . .	32
<b>7</b>	<b>Implementation</b>	<b>35</b>
7.1	Preliminary steps . . . . .	35
7.2	Database system . . . . .	36
7.3	Twitter API . . . . .	37
7.4	Web scraper . . . . .	38
7.5	Tweet Sorting . . . . .	38
7.6	Sentiment analysis . . . . .	38
7.7	Relational table manager . . . . .	42
7.8	Linear and Polynomial Regression . . . . .	42
<b>8</b>	<b>Testing</b>	<b>43</b>

---

8.1	Model testing . . . . .	43
8.1.1	Sentiment analysis . . . . .	43
8.1.2	Regression . . . . .	44
8.2	Unit testing . . . . .	51
8.2.1	Database management . . . . .	51
8.2.2	Stream listener . . . . .	52
8.2.3	Tweet sorting . . . . .	52
8.2.4	Poll updating . . . . .	52
8.2.5	Sentiment analysis . . . . .	53
8.2.6	Relational table management . . . . .	53
8.2.7	Prediction model . . . . .	54
8.3	Integration testing . . . . .	54
<b>9</b>	<b>Results and Discussion</b>	<b>56</b>
9.1	Model accuracy . . . . .	56
9.1.1	Linear regression . . . . .	56
9.1.2	Polynomial regression . . . . .	57
9.1.3	Follower scaling . . . . .	57
<b>10</b>	<b>Evaluation</b>	<b>58</b>
10.1	Requirements . . . . .	58
10.2	Project management . . . . .	58
<b>11</b>	<b>Conclusion</b>	<b>61</b>
11.1	Summary . . . . .	61
11.2	Future work . . . . .	61
11.2.1	Predictions . . . . .	61
11.2.2	Sentiment analysis . . . . .	61
11.2.3	Website . . . . .	62
<b>12</b>	<b>Legal, Social and Ethical Issues</b>	<b>63</b>
	<b>Bibliography</b>	<b>64</b>
<b>A</b>	<b>Appendix A: Important code</b>	<b>66</b>
A.1	Stream listener . . . . .	66
A.2	Tweet sorting . . . . .	68

0	Contents	5
A.3	Poll updating . . . . .	69
A.4	Sentiment analysis . . . . .	70
A.5	Relational table manager . . . . .	73
A.6	Prediction model . . . . .	74
<b>B</b>	<b>Appendix B: Additional predictions</b>	<b>78</b>
<b>C</b>	<b>Appendix C: Git</b>	<b>84</b>

# 1. Introduction

## 1.1 Context

The world of politics is ever changing, and it is influenced hugely by public opinion. There are constant opinion polls that show which parties the people would vote for if a general election were to be held at any given moment. Predicting the outcome of future polls would be useful to industries that rely on the state of the government such as finance and the stock market, science and research funding, defence and the military, medicine and pharmaceutical companies, as well as to the parties themselves to determine strategies and to gauge the public's opinion on their recent actions. Presently, Twitter plays a large role in spreading information and opinion to the general public, especially where politics is concerned. Furthermore, it is extremely far reaching, with approximately 18.3 million users in the UK alone in 2021 [1]. Out of all of the most widely used social media sites, Twitter is one of the more suitable for sentiment analysis as user tweets are generally short, and consist of opinionated language that lends itself to strong sentiment evaluation.

In the recent past, computer science and the statistical methods that are suitable for analysing trends in voting information have evolved rapidly. The format of poll data is relatively simple, and is therefore a prime subject for statistical analysis. Past poll results coupled with quantified public opinion has the potential to produce useful predictions about the specific voting intention of the population in future polls.

## 1.2 Project aims and contribution

The goal of this project is to use sentiment analysis to evaluate the public's tweets and the results of past opinion polls to create a model that can produce reliable predictions about the results of the next opinion poll. The desired final results are to show the exact vote percentages each major political party would receive if there was a general election at the time of the next poll. Previously, there have been instances of people using Twitter analysis to predict the results of elections [2][3], mostly in the US. However, the goal of this project differs in that the aim is to produce an application that can be accessed at any time, that constantly collects and processes the most current data to produce an exact result in the next opinion poll for each major political party in the UK.

## 1.3 Motivation

Politics influences a huge portion of our day to day lives, and therefore the ability to predict trends and relationships in the current political state is extremely useful and potentially profitable information in a plethora of different fields. Fields including, chiefly: finance and the stock market; “The evidence presented concerning short-term share price movements indicates that the stock market responds both to the findings of opinion polls in the run up to elections and to elections themselves” [4]; as well as the managers of the political parties themselves, as there is evidence that the outcomes of polls influence voting intentions; “Voter expectations concerning the outcome of an election on the performance of a party or candidate have come to play an increasingly important role in the study of voting behavior. Whenever there are more than two candidates or parties, the voter may base his decision not only on his own preference but on expectations of what other voters will do.” [5].

## 1.4 Related works

The use of social media sites such as Twitter to measure relationships between public opinion and many different opinion influenced metrics is far from a new idea and there are a surfeit of papers that support this method of data analysis, this is discussed further in the section 4.



## 2. Requirements

### 2.1 Functional requirements

1. The system must have a database of sufficient size to store past poll results and enough tweets to produce an accurate sentiment for each poll.
2. For each poll result stored, the system must identify and implement a relationship that tracks which tweets are relevant to which political party.
3. The system must actualize a stream listener that collects tweets in real time that are relevant to the Conservative Party, Labour Party, Liberal Democrat Party and the Green Party, storing only those tweets that contain certain party specific phrases.
4. The system must carry out tokenizing and cleaning of tweets so that they can be effectively analysed for sentiment.
5. The system must implement a sentiment analysis model that classifies tweets as positive or negative sentiment.
6. The system must have a method of measuring the strength of the dependency of a party's poll result on their total Twitter sentiment.
7. The system must be able to compile a value for the total sentiment towards a party, inside certain time parameters, from the sentiment of each related tweet.
8. The system must allow a user to input a particular date and receive an exact vote percentage prediction for each party given the sentiment towards the party running up to said date.

### 2.2 Nonfunctional requirements

1. The sentiment analysis model should be a custom trained Naive Bayes classifier that, when tested, produces a greater accuracy than NLTK's built in VADER sentiment analysis feature.
2. The system should be usable at any time.
3. The system should present the predictions in a user friendly, understandable manner.
4. The system should outperform the average human in its predictions for poll results.

- 
5. The system's predictions should improve over time, learning from actual poll results as they are released.
  6. The system should be integrated into a website for ease of access.

## 3. Constraints

### 3.1 Hardware

The computational power available during the development of this project is limited to a standard HP laptop and custom built but outdated desktop PC. This has affected the speed at which the collected data can be processed, and more importantly the time taken to perform unit testing during each step of the development cycle. Utilisation of parallel processing and multi-threading helped alleviate some of these limitations - however this itself created new unforeseen constraints and compatibility issues, such as Python's nltk package not being threadsafe and requiring multiprocessing instead.

Furthermore, the large volume of data that was collected, and would need to continually be collected during the lifetime of the end product quickly began to overwhelm the limited storage capacity of the two systems as well as Google Drive's cloud storage. Continued uploading of new backups and deleting of old data consumed a significant portion of development and testing time. Ideally, parallel processing and storage of these large volumes of data would be assigned to a dedicated external server.

### 3.2 Software

Due to hardware and time constraints, much of the application relies on prebuilt python packages and libraries. Any incompatibilities had to be managed during development as they arose, which took longer to resolve than predicted.

### 3.3 Data

Because the project relies heavily on data from different organisations and what they are willing to release to the public, the resources available are adequate but limited. Twitter's application programming interface (API) only allows realtime scraping of tweets. For an independent developer to gain access to Twitter's store of past tweets they must pay a significant fee, as this service is aimed towards large corporations and analysts. Furthermore the tweet listener API used to collect tweets only allows two instances executing at once from one IP address, as well as a maximum of 200 target words per instance. This severely limits the volume of data that an individual can collect.

## 4. Background Research

### 4.1 Social media influence and validity

Past works have found significant potential value in using social media as a reliable information market [6]. Furthermore, similar projects involving analysing tweets to predict poll and election results have identified political sentiment in tweets as a valuable resource for prediction model training [7][8].

### 4.2 Data quality assessment

It is vital when performing any kind of data analysis to assess the quality of the information in order to ascertain how reliable, and thus potentially useful, the results of the analysis might be. Although there are no formally agreed upon dimensions by which to objectively assess the quality of gathered data, one can form a basic set of requirements that data must meet to be considered credible by combining research from several trusted papers in the matter, these being: accuracy, completeness, consistency, and timeliness. The definitions of these dimensions are taken specifically from the paper “Methodologies for data quality assessment and improvement” [9] as the definitions in this report draw from many trusted sources.

#### 4.2.1 Accuracy

We are concerned only with syntactic accuracy, that is the closeness of a value,  $v$ , to the elements of the corresponding definition domain,  $D$ . In this case,  $D$  is either a positive sentiment value (1) or a negative sentiment value (-1) and  $v$  is the value of each tweet after it has been cleaned and fed into a sentiment analysis algorithm. The algorithm used only produces either a 1 or -1 value, so while the intensity of each tweet’s sentiment is lost, the overall sentiment value can be considered accurate.

#### 4.2.2 Completeness

Completeness of a dataset can be defined in several ways, in the case of Twitter data the most logical definition being: “[the] ability of an information system to represent every meaningful state of a real world system” [10], e.g. how well the tweets used for analysis represent the whole of the UK general public. First of all, Twitter’s reach is surprisingly broad as mentioned in the introduction, and secondly, information gathered from social media does not necessarily

need to span the whole of a population to be considered useful. For example, one paper states: “Although the population of social media users is not representative of one country’s citizenry, there are still some doubts about whether such bias could affect the predictive skills of social media analysis compared to traditional offline surveys. Indeed, the former aspect (the predictive skills of social-media analysis) does not necessarily require the previous factor (the issue of representation) to hold true in order to effectively apply.” [11]. Even though the whole population might not share their opinions online, the politically active internet users act like “opinion makers” who anticipate the public’s preferences and speak for a wider audience.

### 4.2.3 Consistency

Due to the limited semantic nature of tweets and the process of tokenizing and cleaning each individual tweet, resulting semantic values are extremely consistent. Tokenizing is the process of dividing a corpus (in this case each individual tweet) into its basic meaningful entities, and significant development time has gone into making sure each tweet is stripped down to only the language that contributes to a sentiment value, such as removing links, punctuation and stop words (words that exist only for syntactic reasons such as and/or/etc).

### 4.2.4 Time-related Dimensions: Currency, Volatility, and Timeliness

The temporal aspect of the data is fundamental to the project as its value stems from allowing markets, industries and political figures to immediately respond to changes in public opinion. This is why so much importance is placed on the ability to predict poll results in real time using only the most relevant and up to date sources. When it comes to currency, the final product relates only tweets from the 3 days running up to each poll (2 days before and the day the results are released) to train the regression model.

In relation to volatility and timeliness, there is research that shows the stability of social network information despite sudden surges in political discussions, evidenced in the paper “Predicting elections from social media: a three-country, three-method comparative study” [12]. This stability suggests that the quality of data collected from Twitter in the run up to a poll is not likely to be adversely affected by unpredictable, potentially artificial or malicious, swings in current events.

### 4.2.5 Subjective value

Separately to the aforementioned basic metrics, there is a need to assess the specific value of social media information and its practicality in predicting how changes in public opinion affect the political sphere. It is generally agreed that in this use case the information has a high subjective value, summarised clearly by this passage from the report “Wisdom of the Crowds”: “In line with the findings of Shah (2010) and Gloor et al. (2009), social media are here found to be able to effectively aggregate public opinions regarding political matters and to be reliable measures of public opinion swings.” [13].

### 4.3 Political science and poll influence

The results of this project would be rendered effectively useless if there was no evidence of poll results having a noticeable effect on the outcomes of future polls, and more importantly, official elections. There has been much prior research into the existence of a phenomenon called the “bandwagon effect” - essentially, does a positive result in the polls for a specific party lead to a higher voting intention in subsequent polls? A significant portion of this research suggests that the bandwagon effect does exist in some capacity, although it is not necessarily the product of a simple change in voter intention. However, poll results are in many cases accompanied by further qualifying text framing a party as “winning” in the polls, which has consistently shown to positively affect this party’s performance in subsequent votes [14]. For this reason, as well as the commonly accepted assumption in political and analytical circles that polls do affect voter intention, the conclusions drawn in this paper are considered to be genuinely useful and potentially advantageous to politically influenced industries.

### 4.4 Sentiment analysis and classification algorithms

Regarding the feasibility of analysing the sentiment of tweets to gauge public opinion, this practice is not unconventional. Social media is designed to encourage anyone to share their opinions on almost any matter. Twitter has a character limit for single tweets of 280 characters, and the average tweet is only 33 characters in length. This short format, especially biased style of writing, is ideal input for sentiment analysis algorithms, as it is possible to reduce these texts even further to purely semantically relevant words and produce a +1/-1 sentiment value for the whole text with a high level of accuracy.

This project abstracts the sentiment analysis of tweets into a classification problem, i.e. sorting tweets into those that consist of overall positive language and conversely those that use overall negative language. A Naive Bayes classifier is an easy to implement but efficient and robust method of text classification [15] and thus is implemented into the classification algorithm used in the final application. These properties of NB classifiers are especially useful considering the limited power of the hardware available. There are many examples of projects using a Naive Bayes model to classify text, such as social media posts including tweets [16].

Bayes’ theorem finds the probability of an event occurring given the likelihood that another specific event has occurred. The mathematical equation for Bayes’ theorem is as such:

$$P(A|B) = \frac{P(A)P(B|A)}{P(B)} \quad (4.1)$$

$A$  and  $B$  are events.  $P(A|B)$  is the probability of  $A$  given  $B$ .  $P(A)$  and  $P(B)$  are the independent probabilities of events  $A$  and  $B$  respectively.

This formula can be modified to predict the probability that a set of events (in the case of this project, a tweet) belongs to a certain class:

$$P(y|X) = \frac{P(y) \times \prod_{i=1}^n P(x_i|y)}{\prod_{i=1}^n P(x_i)} \quad (4.2)$$

$y$  is a class variable and  $X$  is a vector of  $n$  independent features. Also, since the denominator remains constant for a given input it can be ignored, giving:

$$P(y|X) = P(y) \times \prod_{i=1}^n P(x_i|y) \quad (4.3)$$

Probability of a class  $y$  given a set of events  $X$  is calculated for each different class and whichever one produces the maximum value is selected as the predicted classification for the set of events. For example, two values would be calculated using words in a tweet as the set of events  $X$ , and two classes for sentiment: positive and negative. The tweet would be labelled with the class that produced the greater probability,  $P(y|X)$ .

Classification of a corpus is most effective when the text is reduced to only its basic meaningful entities, which is why each collected tweet must undergo thorough “cleaning”. Firstly, each individual tweet is tokenized, that is, broken down into its most basic useful components such as words, punctuation, hashtags and emojis. This allows all tokens that don’t affect overall sentiment values to be removed. Furthermore it is important to normalize each token into its root form, this is called lemmatization. For example, tokens such as “walked”, “walking” and “walks” are reduced to simply the word “walk” so that they can be analysed as a single item, as they are essentially the same word.

## 4.5 Regression models

The ultimate aim of this project is to identify a relationship between voter intention in polls and the sentiment of the public towards a party in the days leading up to said polls. One of the most suitable models for calculating and visualising this relationship is a regression model. Two types of regression implemented in this study are linear regression and polynomial regression. One would logically expect to find, if such a relationship exists, it is most closely replicated using linear regression; a non linear relationship would suggest an extreme (potentially erroneous) sentiment value might unrealistically override any other factor affecting voter intention. A linear relationship is evidenced by research showing that parties framed as losing or on a downwards trend in the polls do not suffer from all of their supporters “jumping ship” [14].

Linear regression is a method of statistical analysis that can use the dependence of a variable  $y$  on another variable  $x$  to predict a new value of  $y_i$  given some  $x_j$ . There is assumed a linear relationship between the dependent variable  $y$  and the set of independent variables  $X = (x_1, x_2, \dots, x_n)$  which leads to a model described by the equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \dots + \beta_n x_n + \varepsilon \quad (4.4)$$

$y$  is the predicted value of the dependent variable.  $X$  is a set of regressors or predictor variables.  $0, 1, 2, \dots, n$  are known as the regression coefficients.  $\varepsilon$  is the error term or disturbance term.

The equation above is known as multiple linear regression, and when there is only one independent variable it is called simple linear regression:

$$y = \beta_0 + \beta_1 x + \varepsilon \quad (4.5)$$

When given a set of values for the dependent variable  $y$  and the independent variable  $x$ , the regression function calculates values for the regression coefficients that minimise the residuals - the differences between the actual values of  $y$  and the predicted values of  $y$  ( $\hat{y}$ ). From these values, the error sum of squares (SSE) can be calculated. The reason for squaring this value is that, since the regression function minimises these residuals, or in layman terms “goes through the middle of the observed points”, the sum of all the residuals would equal zero. Here is the formula for SSE:

$$SSE = \sum_{i=1}^n (y_i - \hat{y}_i)^2 \quad (4.6)$$

$y_1 \dots y_n$  are the observed  $y$  values and  $\hat{y}$  is the predicted  $y$  values.

SSE measures the unexplained error between the model’s predicted values ( $\hat{y}$ ) and the actual values. There is another calculation that measures the explained error between the model’s predicted values ( $\hat{y}$ ) and the mean value ( $\bar{y}$ ) - the expected value of  $y$  if it had no dependence on  $x$ . This value is known as the regression sum of squares (SSR) and is given by the formula:

$$SSR = \sum_{i=1}^n (\hat{y}_i - \bar{y})^2 \quad (4.7)$$

SSE and SSR can be summed to find the total sum of squares (SST) which quantifies how much the data points  $y_1 \dots y_n$  vary around their mean:

$$SST = SSR + SSE \quad (4.8)$$

These measures of error/variance can be used to produce a formal metric of how accurate the regression model is, known as the coefficient of determination,  $r^2$ :

$$r^2 = \frac{SSR}{SST} = 1 - \frac{SSE}{SST} \quad (4.9)$$

Fundamentally, linear regression is all about calculating the best predicted weights that result in the smallest SSE and therefore an  $r^2$  closest to 1. Figure 4.1 is an example of what a linear regression function might look like and displays how the errors relate to the function when graphed. It can be seen that the disturbance term  $\varepsilon$  corresponds with the y axis intercept.

## 4.6 Polynomial regression

Despite the relationship between vote intention and influence likely being linear, there may arise some interesting insight from experimentation with a non linear model. Polynomial regression is a generalised form of linear regression, for example the quadratic regression model is given by the formula:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x^2 + \varepsilon \quad (4.10)$$



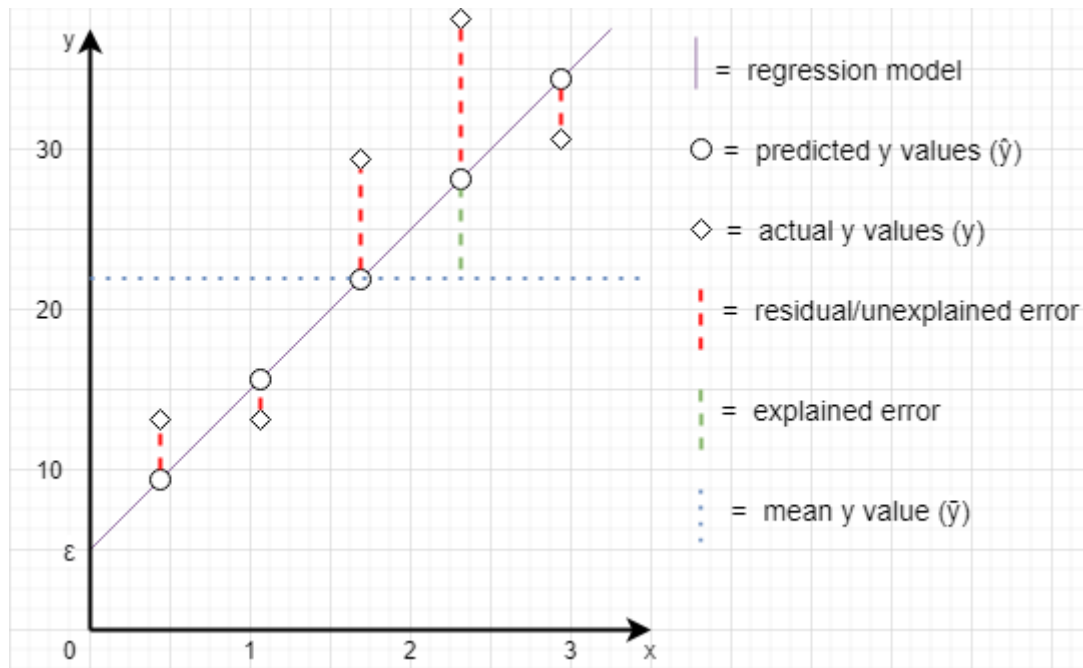


Figure 4.1: An example linear regression model

In all probability this model will overfit the data, especially if the dataset is small. One can take an educated guess that if a  $2n$  order polynomial is used for regression, at some point the vote intention will increase with decreasing sentiment or the vote intention will decrease with increasing sentiment since a turning point will be present in any quadratic model. On the other hand if a  $2n - 1$  order polynomial is used, sentiment will affect the vote percentage in extreme proportions before the first turning point and after the last turning point.

Figure 4.2 is an illustration of how different regression functions might fit a dataset. Graph A is an example of linear regression that seems to greatly underfit the data. Linear regression is relatively inflexible and is therefore best suited to interpolating linear relationships. Graph B is a second order polynomial regression that fits the data much better but still slightly underfits the last two points. Graph C is a third order polynomial that fits the data even better than graph B, and without further analysis seems to be the most suitable model shown here. Graph D shows a fourth order polynomial function that overfits the data, consequently it is overly sensitive to noise and error.

An SSE value can be calculated for both linear and polynomial regression and as stated before, a common method of measuring the performance of a regression function is to find its coefficient of determination, known as  $r^2$ . This value quantifies the fit of the function by quantifying what amount of variation in the dependent variable  $y$  can be explained by variances in the independent variable  $x$ . A larger  $r^2$  value corresponds to a better fit, although in the case of polynomial regression it can also indicate overfitting. Graph D would have a smaller SSE than graph C and therefore an  $r^2$  value closer to 1, however visually it is obvious that D has likely been overfit. This is one downside to using polynomial regression over linear regression.

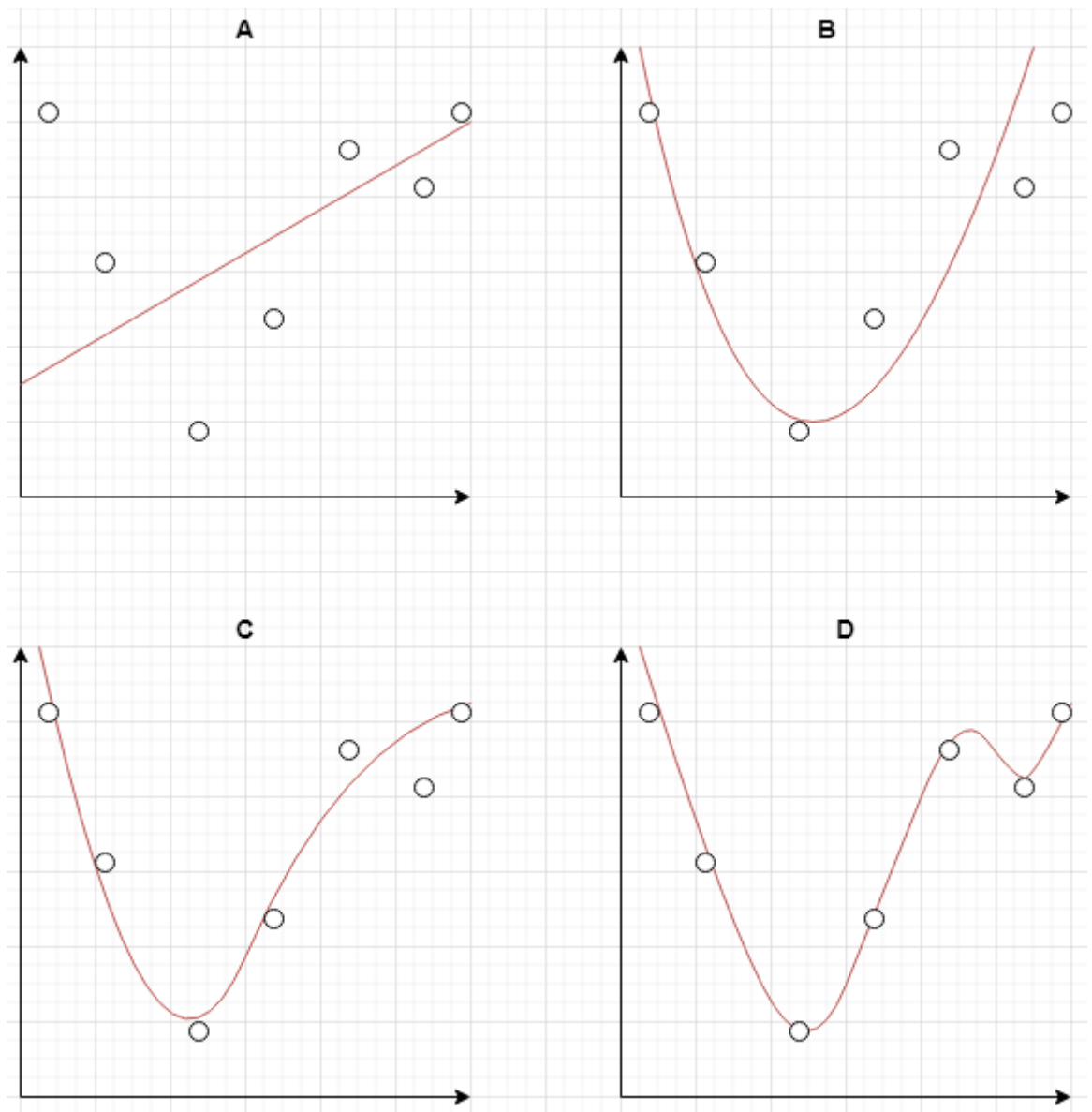


Figure 4.2: Four increasing order regression models

## 5. Project Management

### 5.1 Methodology

Due to the many constraints and generally small scope of this project, a feature driven development approach was adopted when working on the application. This approach is especially suitable given the clear cut requirements and features needed to satisfy a minimum viable product. Almost all of the essential features relied partly on the functionality of each prior feature in order to be effectively implemented and tested, which naturally produced a logical order in feature development.

An adapted agile software development methodology was implemented throughout the development cycle of this project since it most suitably fits the majority of characteristics of this specific task.

#### 5.1.1 Time constraints

Due to the relatively short span of time allocated to the research, design, development and evaluation of the product, and furthermore the writing of this paper, it was essential for the project's requirements to be flexible in case any unforeseen challenges caused a significant unavoidable increase in development time. Furthermore feature driven development lends itself to an agile methodology, since one of agile's primary tenets is to ensure working software over comprehensive documentation and responding to change over following a plan [17]. Essentially, this allowed spontaneous time allocation to problematic essential features and removal of non essential requirements without compromising the integrity of the end product.

#### 5.1.2 Flexibility

As mentioned in the previous point, flexibility of requirements was a primary concern. Although research into similar projects and papers was extremely insightful, it is not possible to predict all of the potential roadblocks, and new information gathered during development can completely change the priority and relevance of certain features. This is especially present in a project that relies on existing APIs provided by private companies, and real time expansion of a knowledge base through collection of social media data. There are three foundational aspects to empirical process control for software development projects [18]. As there was only one developer: the visibility of disturbances to the process was inherent; the inspection of each disturbance was carried out during the continual testing of each component feature; every requirement was adapted according to how fundamental it was to the completion of a minimum viable product.

### 5.1.3 Project size

Given that overall this was a small project completed entirely by one person, such software products are difficult to manage using defined process control and tend to be more manageable with empirical process control involving feed-forward and feedback mechanisms [19]. It would have been wasteful to use project management methodologies more suited to larger teams of engineers.

### 5.1.4 Past experience

It has been found in practice that while the technical requirements play a large role in selecting a methodology, the first process used by a software company is based principally on the prior experience of the person appointed as Software Development Manager [18]. As the sole researcher I have the most experience in adapted agile methodologies in regards to process formation and development, specifically taking part in scrum and sprint sessions. It is hardly a coincidence that this approach fulfils all of the previous attributes, and is also the most familiar to me, since the projects I have contributed to previously were of similar scope.

# 5.2 Schedule

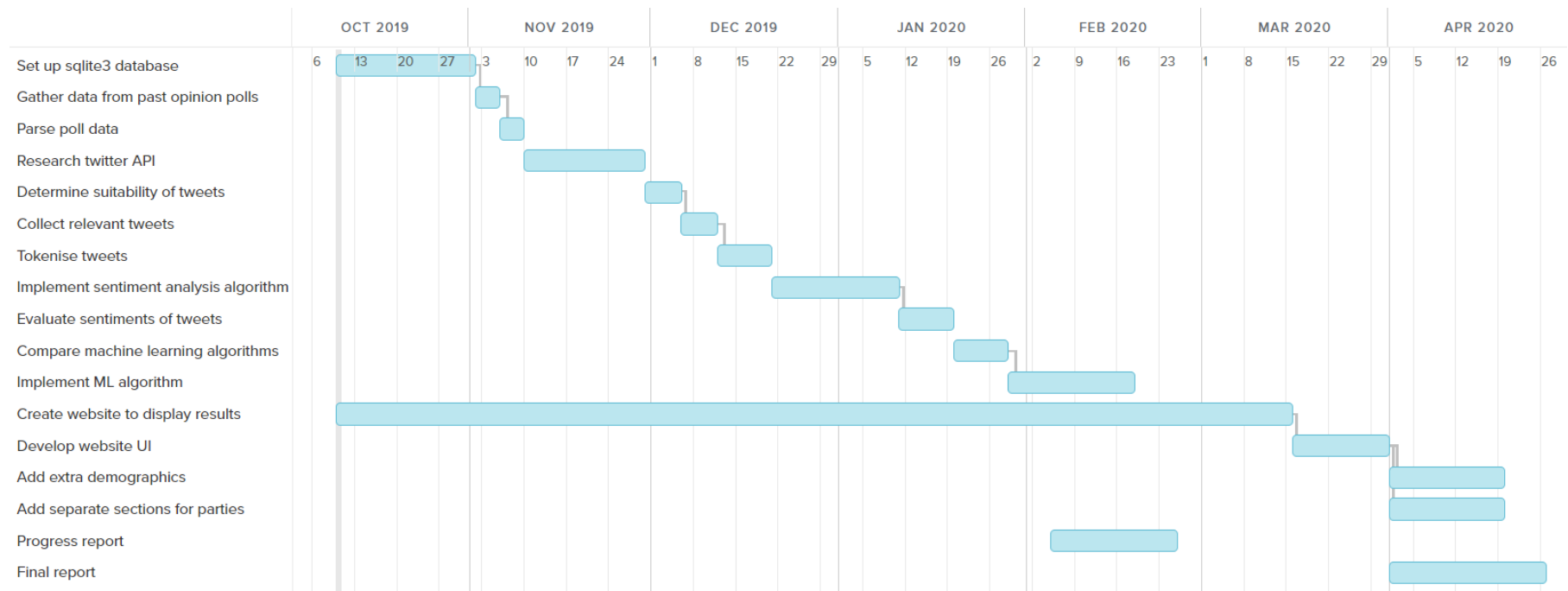


Figure 5.1: Gantt chart of development cycle

Following initial research into the scope, identification of functional and nonfunctional requirements, tools and packages, as well as which development methodology most suited these combined attributes, a Gantt chart (fig 5.1) has been produced depicting the expected progression in development of each feature, and additionally the report components of the overall project. However, midway through development the progress was evaluated against the original schedule and found to be sufficiently behind schedule, with some tasks taking significantly longer than planned. As such the plan was re-evaluated to ensure the critical functional requirements could be met within the time constraints of the project.

## 5.3 Tools and packages

### 5.3.1 Software

This section lists all of the existing technologies employed during development of the application described in this paper, and the justification for each choice.

#### Python

To ensure consistency and minimise the risk of incompatibility between features, practically all of the code that makes up the application will be written in Python 3.9.2. The Pip package installer will be utilised during development as a simple way of integrating all of the external Python packages into the code, negating the need to spend excess development time ensuring each module is up to date and doesn't obstruct the function of other modules. Additionally, use of Python's venv feature allows complete separation of Python dependencies between application development and any other concurrent projects using Python. Before each development session, a virtual environment will be initialised so as to emulate a system with only the external packages relevant to this project installed.

The database system chosen to manage storage and retrieval of data is the open source SQLite3 library. It is a lightweight, flexible database library that can be relatively easily integrated into Python. SQLite3 is most suitable for this scale of project because, without public release, the application should not have to handle a high number of concurrent users, one of the strengths of SQLite3 being that it is particularly lightweight when compared to competitors. However, if the project were to grow further, a more scalable database such as MySQL would likely be more appropriate since it is more efficient at handling larger volumes of stored data and greater degrees of simultaneous access to said data by multiple users.

Some basic web scraping is required to gather past poll results, as well as new results as they are released. Python's BeautifulSoup4 module is more than capable of parsing html from public web pages.

Considering sentiment analysis is the cornerstone of the project, thorough research has gone into choosing the most suitable package for this purpose. The final decision was between Natural Language Toolkit (NLTK), SpaCy, and TextBlob. Although SpaCy boasts high performance when analysing large corpora, it is more suited to larger scale applications and is less accessible to an inexperienced developer. TextBlob and NLTK are both perfectly reasonable choices, however some developers have experienced issues with TextBlob interacting with unicode characters so NLTK was decided to be the safer option. NLTK is a very popular choice

for sentiment analysis, especially when paired with Twitter, exemplified by its inclusion of a Twitter corpus (sample set of tweets) and tweet oriented tokenizer, designed to format short, informal texts that incorporate emojis and hyperlinks.

Scikit-Learn or sklearn will be employed for the purpose of statistical analysis on account of its widespread use and extensive documentation. Once the training data is correctly configured, there will be the option to train and compare the accuracy of multiple machine learning models.

The Tweepy library will be used to allow the system's Python code to interface with Twitter's API. There are many public resources detailing how to employ the StreamListener method to maintain a constant connection to Twitter's services for real time tweet collection.

In addition, the final application makes use of the following libraries for various purposes such as formatting, data handling, parallelisation etc: numpy, pandas, matplotlib, statistics, math, pickle, multiprocessing, requests, datetime.

## **Visual Studio Code**

Due to the feature driven development methodology chosen, there will be a large emphasis on modularity in the application code. VSCode is an extremely useful IDE for smaller projects that will help to keep track of all of the component features, since each feature will likely consist of several smaller scripts imported into a larger main program. VSCode supports various quality of life plugins for all major programming languages, and in this case Kite, an AI powered auto complete plugin, will be helpful in maintaining good programming practices and general consistency.

## **Backups and version control**

Integration of Git into the development workflow is essential for streamlining version control and makes backing up application code much more simple, especially when paired with a remote repository on GitHub. Additionally, it provides a timeline of changes made to each different feature, and facilitates development of different features without conflicts when separated out into different branches. Furthermore, these branches can easily be merged once features are complete, and merges reversed if conflicts arise.

Git is insufficient for backing up large volumes of data, so this project will make use of Google Drive to back up databases containing tweets and poll data.

## **Report writing**

For rough notes and drafts, Google Docs will suffice as it is user friendly and automatically backed up to the cloud. LaTeX will be used for formatting the final report since it is a much more professionally oriented report writing tool.

### **5.3.2 Hardware**

Much of the development will be carried out using a HP 250 G5 Notebook PC, although collection of tweets will likely have to run in parallel to application development. This task will be assigned to a custom built PC continually executing the Twitter API stream listener.

## Specifications

### Laptop

- CPU: Intel Core i5-7200u @2.5GHz
- Memory: 8GB DDR3 RAM
- Storage: 1TB HDD
- Operating System: Windows 10

### PC

- CPU: Intel Core i5-4690k @3.5GHz
- Memory: 2x4GB DDR3 RAM
- Storage: 1TB HDD
- Operating System: Windows 10

## 5.4 Risk management

It is essential to prepare for problems that may arise during the development cycle of the system so that, firstly, preventative measures can be put in place to ensure a problematic scenario can't realistically arise or is greatly reduced in scale, and secondly for in the unfortunate event that an issue significantly hinders progress, there is a prepared procedure to mitigate as much of the loss as possible and proceed development with an alternate plan.

### 5.4.1 Data loss

#### Preventative measures

- Employ git and github to regularly backup code so that hardware failure doesn't mean lost work.
- Check the database regularly and frequently upload new versions to Google Drive.
- Use Google Docs for drafts and notes so that they are automatically backed up.

#### Alternate plans

- In the event of hardware failure, make use of the university's PCs and immediately contact technical support.
- Loss of tweets is a minor issue as they will be constantly collected over time.



### 5.4.2 Time constraints

#### Preventative measures

- Abide primarily by the gantt chart, however prioritise momentum over schedule.
- Once a functional requirement is completed, move on. Leave non functional requirements to the end.

#### Alternate plans

- If development of a functional requirement overruns the schedule, reduce the time allotted to less important features.
- Spend a short amount of time producing a rudimentary new gantt chart.
- Contact the project supervisor and ask for professional advice.

### 5.4.3 Software restrictions

#### Preventative measures

- Apply early for a Twitter developer account and carefully abide by their terms of service.
- Thoroughly research compatibility between Python libraries and refer to documentation periodically.
- Perform frequent unit tests, particularly immediately before and after merging a newly completed feature.

#### Alternate plans

- If the Twitter developer account is revoked, immediately inquire and apply for a new developer key.
- During the research phase, create a list of alternate Python libraries with similar functionality to the original packages.

# 6. Design

## 6.1 Overview

Before beginning development, it is important to visualise the basic flow of data into and out of the application and how a user might interact with the system as a whole. One of the simplest methods of doing so is to produce a system context diagram (a level 0 data flow diagram) for the prediction application, outlining how the system will interact with external entities and vice versa.

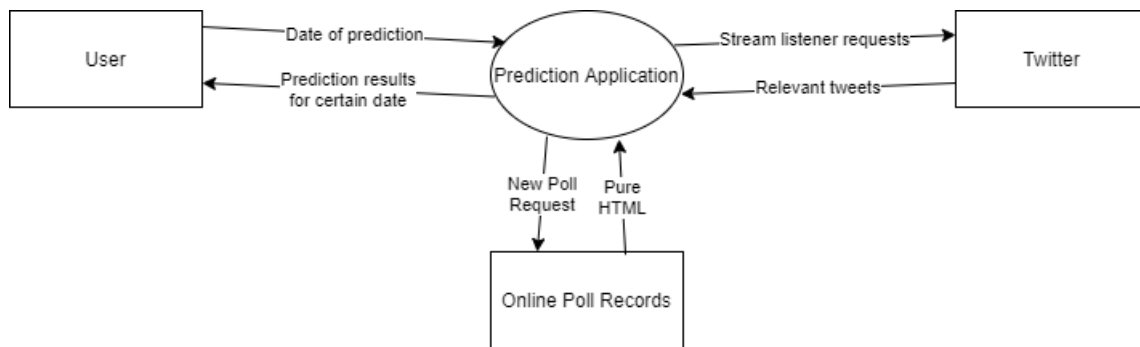


Figure 6.1: System context diagram

Figure 6.1 is a very rudimentary view of the system; the application itself is abstracted to a single process and the flow of data is kept to only that which is needed to understand how the application uses external entities to collect necessary data. This is all that the user will need to understand: they input a date up to the present day that they would like to know a prediction for and the application, after a short amount of time, will provide them with an exact value for the predicted poll results for said date. The stream listener will maintain a persistent connection with the Twitter API, which sends a constant stream of new tweets to the system, since the primary requirement is to predict results using current public opinion. Periodically, the system will scrape the internet to retrieve the relevant information on newly published poll results. Constant updates to real poll results will allow the prediction algorithm to be trained using the most up to date official data.

As previously stated, most components of the prediction application will rely on some part of previously developed components to be able to properly function, or may require information produced by such components in order to be adequately tested. Therefore it is required that the single process in the system context diagram be broken down into separate processes that better represent individual features. This is shown in a level 1 data flow diagram.

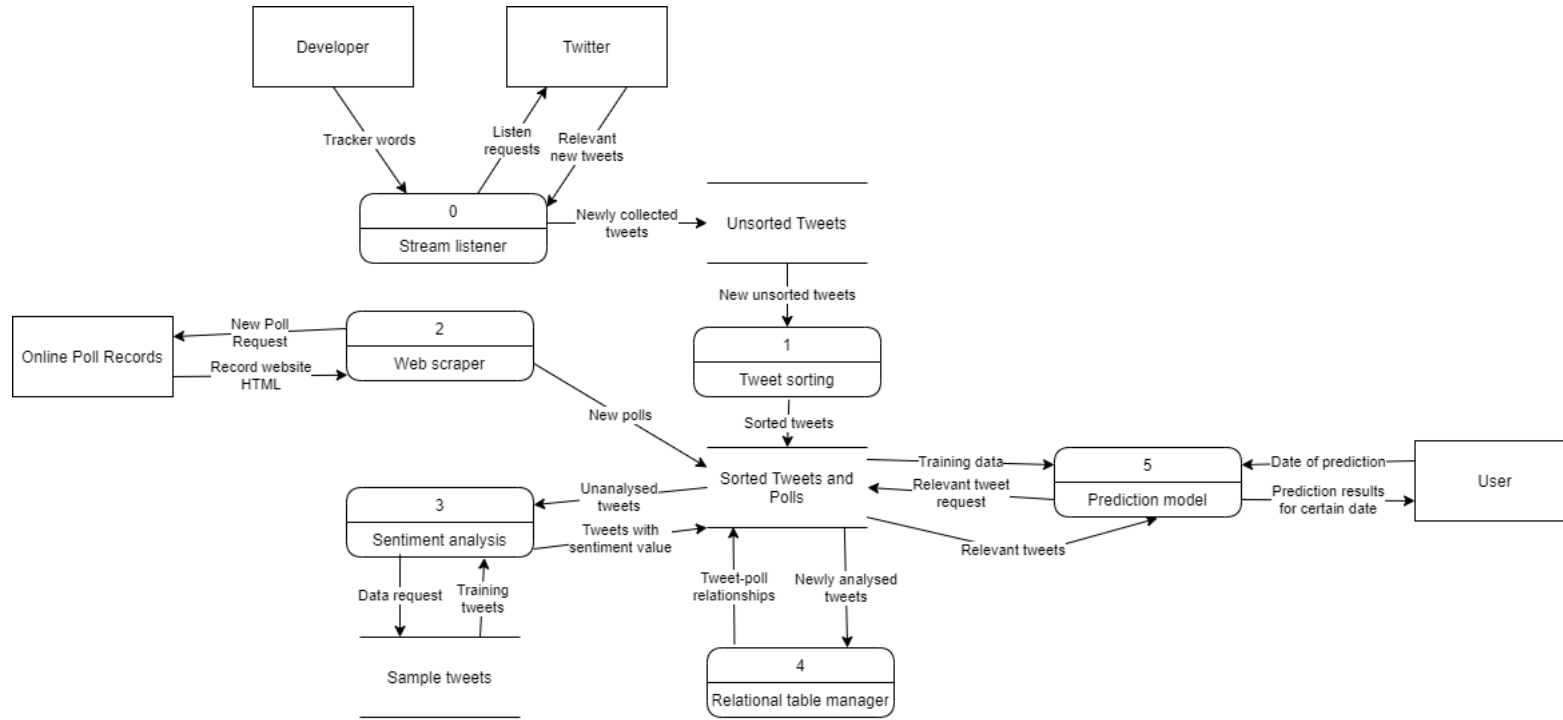


Figure 6.2: Level 1 data flow diagram

Figure 6.2 is a more detailed visualisation of every feature and how they will interact with each other to produce a working system - including where data is sourced from and stored, and how it generally flows during normal operation of the system. The numbering of the processes gives an idea of the order of operations, however some of these processes will execute in parallel for greater efficiency. The following is a further explanation of the intricacies of the fundamental processes.

## 6.2 Stream listener

The stream listener will be constantly executing and storing tweets in the unsorted tweet database. It will be mostly unsupervised, so some automatic error handling will need to be implemented.

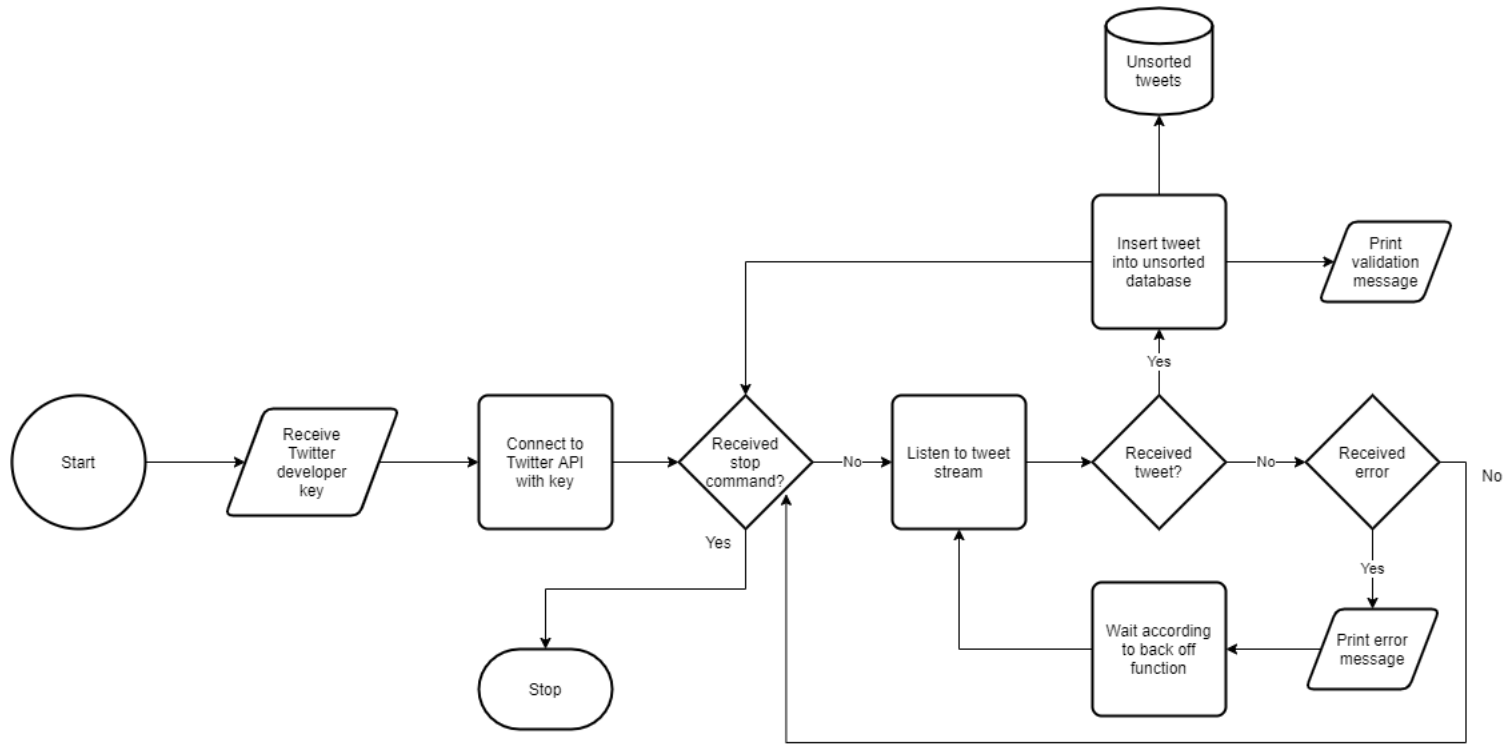


Figure 6.3: Stream listener flowchart

### 6.3 Tweet sorting

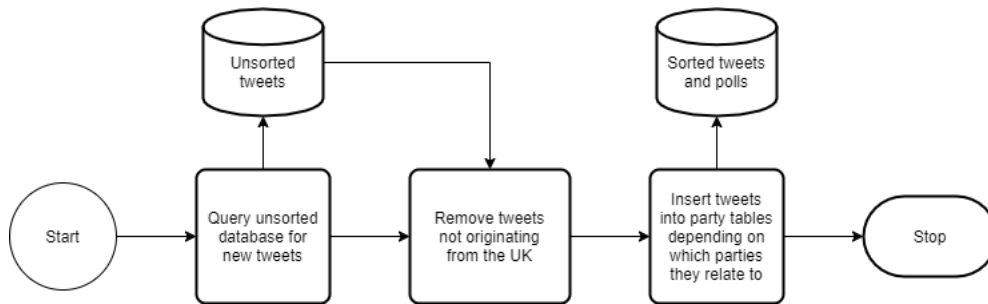


Figure 6.4: Stream listener flowchart

Periodically, newly collected tweets will be sorted: they will be more thoroughly analysed and unnecessary tweets will be discarded, such as those that aren't tweeted from within the UK. Unfortunately not all tweets contain location data, so some tweets from other countries may still be saved, although only tweets containing English will be collected so foreign tweets shouldn't make up a significant enough volume to noticeably skew predictions.

### 6.4 Web scraper

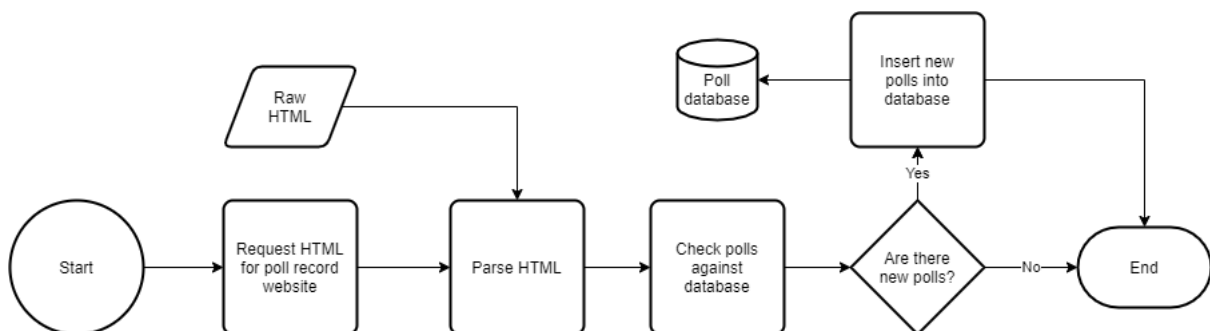


Figure 6.5: Web scraper flowchart

To maintain autonomy, a web scraper will regularly check a poll record website for new official poll results. It should work by parsing the HTML of the site to extract only the poll results that aren't already stored in the database. There are risks to this method of data extraction as it is solely dependent on a site not under the control of this system's developer, most importantly if the website HTML is modified by even a small amount the scraper might not parse the text correctly. Consequently, during the life cycle of the application the developer will have to occasionally test this feature, or acquire a more stable source of data.

### 6.5 Sentiment analysis

Before the sentiment of a tweet can be accurately interpreted, redundant data and noise must be removed. This can only feasibly be achieved by dividing the tweet into its fundamental

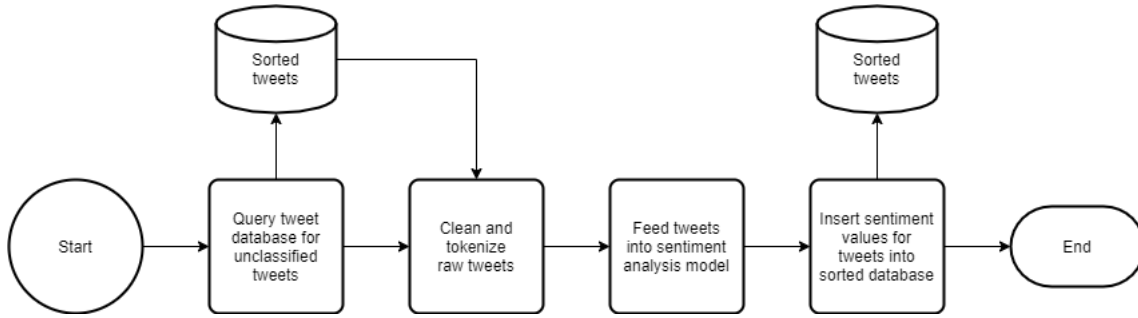


Figure 6.6: Sentiment analysis flowchart

structural elements, called “tokens”. These tokens might include words, punctuation, special characters (emoji), hyperlinks and hashtags. Punctuation is not worth considering since it primarily concerns the syntax of a text, whereas sentiment is usually quantified by considering semantics and the decision to use certain words that carry particularly positive or negative emotional weight. Hyperlinks are potentially useful in determining sentiment, for example links that lead to pictures or videos that are highly emotional. However, analysing those would add sizable complexity that would not fit the time scale of the project.

Once the tweets are “cleaned” of unnecessary information, they can be formatted and fed into the pre-trained sentiment analysis model that will label each tweet with one of two values: positive sentiment or negative sentiment.

Prior studies strongly suggest a Naive Bayes classifier should be implemented as a model in relation to sentiment analysis of tweets; this has been justified in the research section 4.4. A sample set of pre classified tweets will be used to train the model; this set should be as large and varied as possible in order for the model to learn the sentiment of a larger vocabulary of tokens, leading to more accurate classification. A method called k-fold cross validation will be utilised to split the sample tweets into training and validation sets. K-fold cross validation is a very popular method of minimising the prediction error when training a natural language processing model in a supervised fashion, because it is simple to implement and outperforms other techniques such as a simple train/test split. One picks a number  $k$ , e.g. 10, and the training data is split into  $k$  groups. This number should be at least 5; any less results in an increased prediction error [16]. Each portion of data is in turn used as the test data, the other  $k - 1$  portions are used to train the model, then the test portion is used as input for the newly trained model and the accuracy measured. This is repeated  $k$  times using a new portion as test data each time and finally a value for the total accuracy is produced from the mean of all results.

## 6.6 Relational table manager

Data storage and retrieval will play a major role in the flow of data and performance of the overall system. Establishing a normalized database will be a primary goal during development, and one important aspect of protecting data integrity is to employ the use of relational database techniques. Specifically, once the tweets have been sorted into tables for each party, tweets collected in a set period of time before a real poll result will need to be associated with this poll in some manner. As such, poll-tweet pairs will be inserted into an intermediate relational table containing a composite key consisting of a foreign key from a poll as well as a foreign

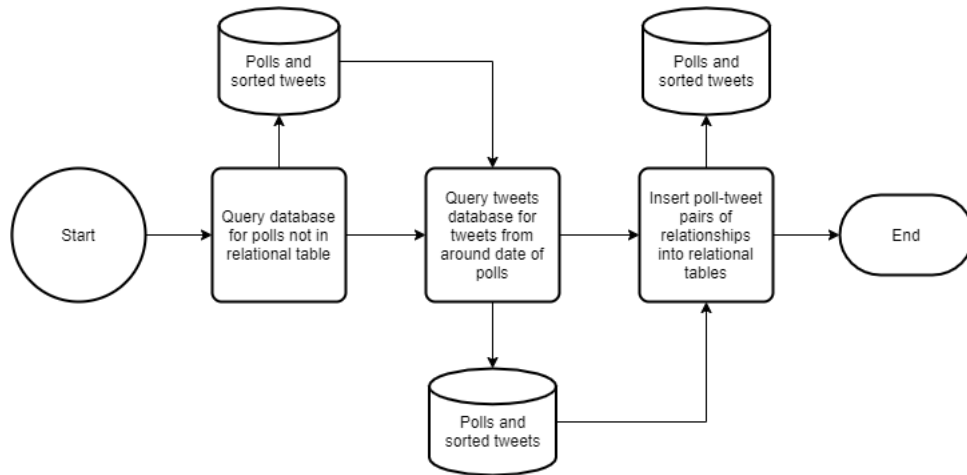


Figure 6.7: Relational table manager flowchart

key from a related tweet. Not only will this preserve data integrity, it should also increase the speed of queries that select tweets relating to given polls.

## 6.7 Prediction model

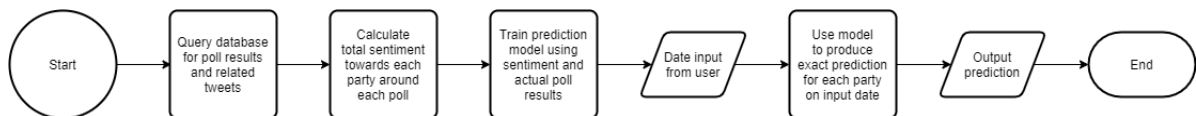


Figure 6.8: Prediction model flowchart

Finally, the feature that delivers the actual quantifiable prediction that the user expects, is the prediction model.

One of the approaches will be to use a linear regression model trained with real poll results to interpolate new poll results. The system will first select the polls which have enough associated tweets collected from a short time before each poll's release. Unfortunately Twitter has hidden data for past tweets behind a paywall. Since these past tweets are unavailable, poll results from before the first tweets are collected will not be available to use for training the model. For simplicity, the next steps will concern only one party, since the process of predicting vote intention for each of the parties will just use different relevant tweets. By this point all of the tweets will have been classified into positive or negative sentiment, so the next step is to collate these values into a combined sentiment towards the party in the individual polls. The metric for sentiment will be a relative value so that parties of different sizes can be compared more equally. To gain further insight into social media's ability to spread and manipulate public opinion, separate tests will include scaling a tweet's sentiment by the popularity of the poster of the tweet. Popularity will be determined using the Twitter user's number of followers. It would be potentially insightful to use a tweet's number of retweets and likes, however the tweets will be collected as they are posted and as such will not have had the time to gather these attributes. Followers seems like an appropriate metric to determine popularity since the active choice to follow a user means a much higher likelihood of seeing their tweets.



Actual voting intention for the party in each poll will be paired with its respective sentiment and fed into the model as training data. This is an example of supervised learning; this process is detailed further in section 4.5. The model's prediction accuracy will improve over time through machine learning as more poll results are released publicly.

The regression model will then be ready to make predictions. When given a date between the first collection of tweets and the current day, the system should produce a sentiment value for the party on said date using opinions tweeted less than a week before. After this sentiment value is fed into the linear regression model, it should output an exact predicted value for the variable dependent on this sentiment - the vote intention.

Once the linear regression model has been developed it shouldn't take much extra time to implement and test polynomial regression, as the latter is a generalised version of the former. Users will have a choice of which order of regression to use in the finished application.

## 6.8 Database design

As pictured in the level 1 data flow diagram, it makes the most sense to separate the storage of tweets into two domains: one database in which tweets are stored as they are collected, and a second database which is a more permanent storage location containing more carefully curated information. Separation will allow the constant collection of new tweets to run uninterrupted in parallel to the function of the rest of the system, and new tweets will only be sorted into the second database when absolutely necessary. This will ensure that the stream listener experiences as little downtime as possible, producing a more accurate picture of public sentiment from a larger volume of tweets. Although the stream listener is depicted as the first step in the flow of data, it is essentially an external agent in relation to the remainder of the application, allowing for a higher degree of automation.

Ensuring normalized databases is important to maximising the integrity of data and general ease of use when querying the system. The original schemata for the two databases are depicted in figure 6.9 and figure 6.10.

tweets	
newId	INTEGER PRIMARY KEY
id	INTEGER
user	TEXT
text	TEXT
hashtags	TEXT
location	TEXT
coordinates	TEXT
date	TEXT
followers	INTEGER
retweets	INTEGER
favourites	INTEGER
replyToId	INTEGER
party	TEXT

Figure 6.9: Unsorted tweet database schema

These schemata have been constructed by analysing the sources of data. The most reliable

conTweets		labTweets		libdemTweets	
conId	INTEGER PRIMARY KEY	labId	INTEGER PRIMARY KEY	libdemId	INTEGER PRIMARY KEY
id	INTEGER	id	INTEGER	id	INTEGER
user	TEXT	user	TEXT	user	TEXT
text	TEXT	text	TEXT	text	TEXT
hashtags	TEXT	hashtags	TEXT	hashtags	TEXT
location	TEXT	location	TEXT	location	TEXT
coordinates	TEXT	coordinates	TEXT	coordinates	TEXT
date	TEXT	date	TEXT	date	TEXT
followers	INTEGER	followers	INTEGER	followers	INTEGER
retweets	INTEGER	retweets	INTEGER	retweets	INTEGER
favourites	INTEGER	favourites	INTEGER	favourites	INTEGER
replyToId	INTEGER	replyToId	INTEGER	replyToId	INTEGER
sentiment	TEXT	sentiment	TEXT	sentiment	TEXT

greenTweets		polls	
greenId	INTEGER PRIMARY KEY	id	INTEGER PRIMARY KEY
id	INTEGER	pollster	TEXT
user	TEXT	date	TEXT
text	TEXT	con	TEXT
hashtags	TEXT	lab	TEXT
location	TEXT	libdem	TEXT
coordinates	TEXT	green	TEXT
date	TEXT	positiveTweets	TEXT
followers	INTEGER	negativeTweets	TEXT
retweets	INTEGER	sentiment	TEXT
favourites	INTEGER		
replyToId	INTEGER		
sentiment	TEXT		

Figure 6.10: Sorted tweets and polls database schema

source for new poll results was Wikipedia because it is almost immediately updated with new results from all of the various political opinion pollsters in the UK in a very consistent format [20]. Twitter’s API documentation for the tweet object [21] has been consulted to decide on the columns in the tweets table, and more information than is immediately needed will be stored: such as followers, retweets, favourites and the id of the parent tweet when the tweet itself is a reply. Inclusion of some fields may seem unnecessary, and although it will increase the storage capacity needed to maintain the system, it will more importantly result in a greater degree of scalability; future development could include these metrics to improve the accuracy of the prediction model or even include completely different, equally useful analytics. Therefore, collecting some extra data is deemed a worthwhile tradeoff.

After further consideration, it has been identified that relational tables, between each party’s table of sorted tweets and the table of polls, are essential. This has been detailed in section 6.8 and is further visualised in an entity relationship diagram, figure 6.11.

The relationship between a party tweet entity (conTweets, labTweets, libdemTweets, greenTweets) and the intermediate entity (conPollTweets, labPollTweets, libdemPollTweets, greenPollTweets) is one to many, and the relationship between the polls entity and the intermediate entity is also one to many. This allows one tweet to be related to multiple polls (several polls may be carried out in a small time span) and one poll can also be related to multiple tweets (this is a fundamental concept of the project).

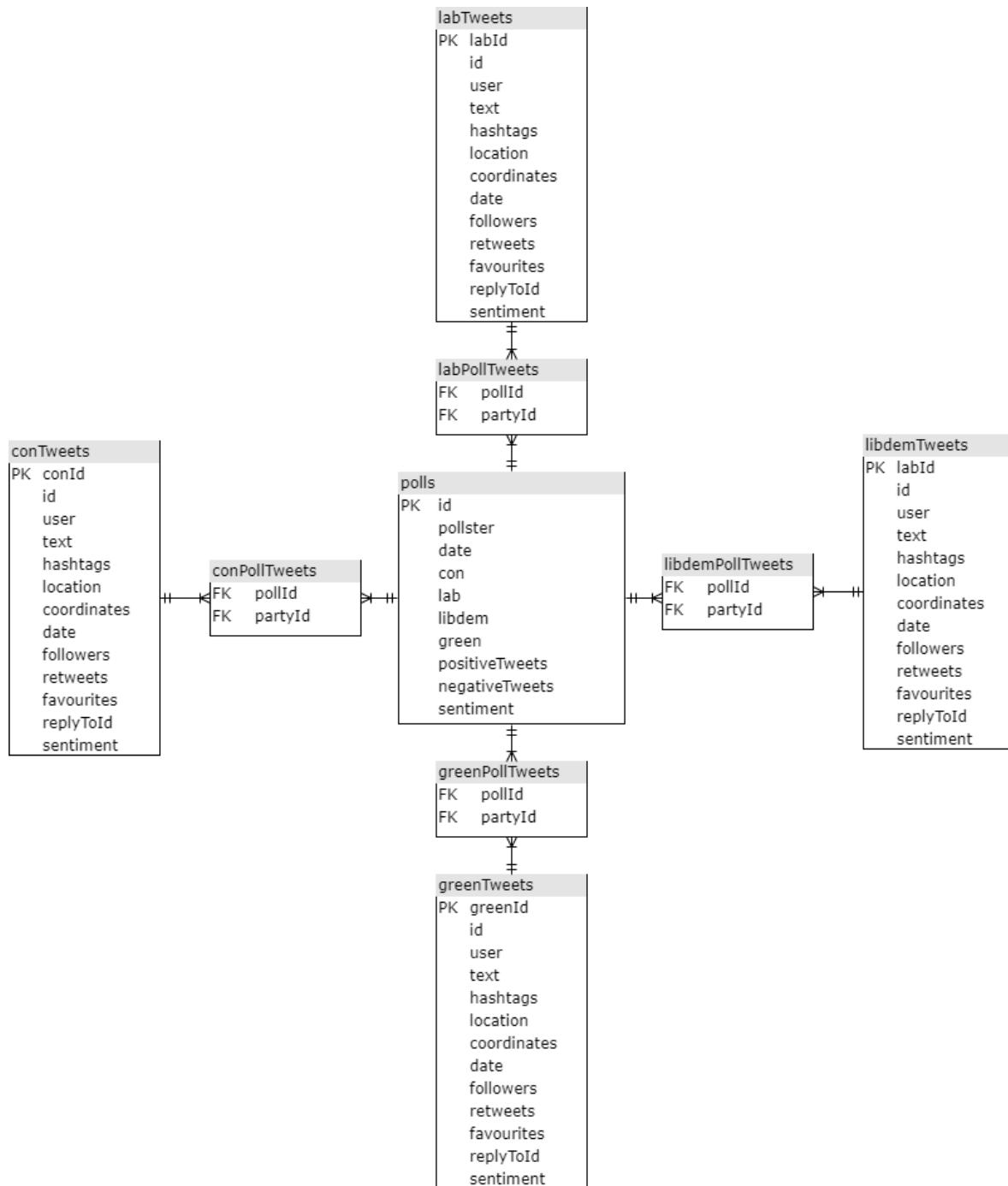


Figure 6.11: Entity relationship diagram for the sorted tweets and polls database

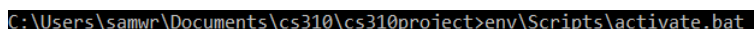
# 7. Implementation

## 7.1 Preliminary steps

Before the bulk of development could be approached, it was necessary to configure the development environment that would encourage good programming habits and instinctive backing up of important data. The IDE utilised was Visual Studio Code because it lets the developer quickly change between all of a project's scripts easily, and the search function allows efficient scanning of all the files in a project with ease. Additionally, the abundance of quality of life plugins, including Kite's AI autocompletion, helped to streamline workflow and cut down on time wasted trying to remember the names of every variable and function. This in turn reduced human error leading to syntactic bugs.

Backing up python code and tracking changes was made infinitely easier by using git, and further linking the local repository to a remote repository hosted on github. Whenever development began on a new feature, a new branch was initialised and appropriately named to accommodate said work. Once the feature was completed and thoroughly tested, the offshoot branch would be merged back into the main branch and further tested for compatibility. Git is not optimised for storing large files, so a .gitignore file was created and maintained to specify which files, such as the databases and csv files, shouldn't be included in git commits. The final summary of all commits can be found in the appendix.

To ensure complete separation of this project from other simultaneous work, a Python virtual environment was initialised at the start of each development session. Ensuring that only the necessary libraries were installed made it easier to identify and solve conflicts between Python dependencies. The virtual environment was also significant in regards to portability; a file detailing the libraries installed inside a virtual environment can be exported to a different machine and an exact copy of the original virtual environment can be constructed. This feature was utilised during the setup of the desktop PC to constantly run the stream listener module. Initialising the environment each time was as simple as executing a script provided by the virtual environment package (see fig 7.1).

A terminal window with a black background and white text. The command entered is: C:\Users\samwr\Documents\cs310\project>env\Scripts\activate.bat

```
C:\Users\samwr\Documents\cs310\project>env\Scripts\activate.bat
```

Figure 7.1: Command for initialising the virtual environment

The full list of Python dependencies are displayed in figure 7.2.

```
(env) C:\Users\samwr\Documents\cs310\cs310project>pip list
Package            Version
-----
beautifulsoup4     4.9.3
certifi             2020.12.5
chardet             4.0.0
click               7.1.2
cyclor              0.10.0
idna                2.10
joblib              1.0.1
kiwisolver          1.3.1
lxml                4.6.2
matplotlib          3.3.4
nltk                3.5
numpy               1.19.5
oauthlib            3.1.0
pandas              1.2.0
Pillow              8.1.2
pip                 20.3.3
pyparsing           2.4.7
PySocks             1.7.1
python-dateutil     2.8.1
pytz                2020.5
regex               2020.11.13
requests            2.25.1
requests-oauthlib   1.3.0
scikit-learn        0.24.1
scipy               1.6.1
setuptools          51.1.1
six                 1.15.0
sklearn             0.0
soupsieve           2.1
threadpoolctl       2.1.0
tqdm                4.58.0
tweepy              3.10.0
urllib3             1.26.2
wheel               0.36.2
```

Figure 7.2: List of Python dependencies for the system

## 7.2 Database system

In order to be able to store the tweets and the poll results used to train the prediction model, an adequate database system was the first requirement to be met. The first step was to become familiarised with Python’s SQLite3 library which allows Python functions to interact with an SQLite3 database. The database itself is simply a local .db file that is connected to by storing the connection in a variable, which for clarity has been named `conn`. The state of the database is manipulated by a cursor object that contains the `execute()` method, which queries the database when passed a string of SQL. After any SQL operation is executed that modifies the state of the database, `conn.commit()` is used to commit the changes, and once the connection is no longer needed, `conn.close` is used to close the connection. This sequence was implemented to minimise the risk of any accidental changes to the database and so that the Database Management System didn’t need to manage so many concurrent requests.

The unsorted tweets were originally stored in a single .db file, which was later on split into two to facilitate two Twitter streams listening and saving tweets in parallel. The first method of having one database file encountered several errors when the two streams attempted to insert tweets into the same table at the same time. Rather than spend time that wasn’t available troubleshooting and configuring concurrent inserts, a more feasible solution was to have two separate database files, with one stream connecting to the first and the other stream connecting to the second. This was not a compromise since these tweets weren’t yet sorted so it mattered little where they were stored.

The second database was created to store tweets sorted into tables of the parties they relate to and past poll results ranging back to 2015. For the prediction model to be trained, for each poll, for each of the four parties: the system needed to query this database to select each tweet associated with the poll. It was decided that this included tweets posted from 3 days before the poll up to the date the poll finished. Adding relational tables between the tweet tables and the poll tables to relate tweets to polls was primarily a method of reducing the time it took for the system to carry out these queries; the process of relating tweets and polls could be done at any time before a user needs a prediction.

## 7.3 Twitter API

In order to gain access to the tweet stream feature of Twitter’s API, authorisation is required through signing up for a developer account. Once the terms of use were accepted and some brief details of the project were submitted to Twitter, they approved the request. Developers are given access to the API through a set of keys associated with the developer account. In the script that handled tweet streaming, the first part of the code sent the keys to Twitter for verification. Interfacing with the Twitter API was done through methods provided by the Tweepy library.

A `StreamListener` class was created that inherits from the Tweepy stream listener, though the methods provided were incomplete and need to be extended. The `on_data` method collected raw JSON data which it then identified and passed to the appropriate method. The class has many methods but the ones this system was concerned with were `on_status()`, `on_error()` and `on_timeout()`.

`on_status()` was executed on receipt of a tweet. The parent method was overridden and the new method extracted the relevant data (shown in the database schema) and called a function from the database control program to insert the tweet into the database, after which a message was printed to the terminal confirming a successful insertion.

`on_error()` was executed on receipt of an error message. For most errors this function was designed to print the error to the terminal and to continue normal function, except if it received an error code of 420 in the form of a 0 byte packet, which was to tell the developer that the listener was being rate limited for sending too many requests over a short period of time. On receiving this error, a sleep function was used to force the listener object to wait 60 seconds before making any more requests to the API.

If for some reason the error handler didn’t catch a 420 error and the listener kept making requests, the listener would receive a timeout signal from Twitter. The `on_timeout()` method was designed to print this information and force the listener to wait another 60 seconds.

The stream listener needed a list of “tracker” words to specify which tweets it should collect. If the listener identified any of the words from the tracker list in a tweet, it would save said tweet. The tracker words chosen consisted of the popular hashtags for each party (found at [22] and [23] plus each elected MP’s Twitter handle. The handles were sourced from the internet [24] as a csv; a small script was developed using pandas to parse the csv into an array of strings. Unfortunately, the Twitter streaming API only allowed for a maximum of 400 tracker words associated with a single stream listener, and there are 650 members of parliament. The simple solution was to run two instances of the tweet collecting script, with the tracker words split between the two. There was still a limit to the number of tracker words, and an obvious route

for future expansion would be to run more instances of the streamer. This was infeasible at the time because Twitter only allows two instances per IP address and a ban on the Twitter developer account was not an acceptable risk.

In future development, topic modelling would be considered as a further measure of more consistently removing irrelevant tweets. A smaller number of irrelevant tweets would in theory produce more accurate predictions given the relationship is modelled correctly.

## 7.4 Web scraper

The library used to build the web scraper was BeautifulSoup4. It saved a lot of development time that would have been used for tedious parsing of HTML. Once the HTML of the poll record site was retrieved, it was stored as a BeautifulSoup object, which is a nested data structure that facilitates parsing HTML using standard tags such as `<head>` and `<body>`. Locating the table and which tags it was nested in required human input, however bs4 made the actual extraction of exactly the required text much easier. Once the table was assigned to a variable it was iterated over and each poll was further divided into an array of the same length as the table it would be inserted into.

## 7.5 Tweet Sorting

Although the tracker words were chosen to collect tweets specific to UK political parties, some irrelevant tweets are still tracked and stored. This was due to phrases shared with parties in other countries, such as “conservative” tracking tweets about the UK and the US Conservative parties. One measure put in place was to discard tweets with a location tag from outside of the UK when sorting tweets into their relevant party tables. This was not fully effective since not all tweets had a location tag. It was decided that tweets without a location tag would be kept since preserving a large volume of data was more important than having a few irrelevant tweets. It was infeasible to sort the tweets into which political parties they referenced when they were being collected because any processing applied when streaming slowed down the collection enough for the program to receive a time out signal after just a short period. Tweets were sorted by a separate process using a library called FTS5. This is an extension for SQLite3 that facilitates creation of virtual tables that can be searched efficiently using a full text search. In this instance, all of the unsorted tweets were loaded into a virtual table in memory for faster processing. Then all tweets that had a location tag outside of the UK were deleted. The full list of UK place names was sourced online and imported from a CSV. Then, each party’s tracker words were used to search the remaining tweets, and any that matched the terms were inserted into that party’s sorted tweet table.

## 7.6 Sentiment analysis

One of the vital functional requirements of the system was that it could reasonably accurately assign a positive or negative sentiment value to real life, unclassified tweets. There were two major sub requirements of this feature: “cleaning tweets” and training a classification model.

“Cleaning” refers to the system’s ability to receive a raw tweet from the database, isolating the main text body, removing unnecessary data and noise, and formatting the text to align with the classifier’s input domain. The first part of this process was relatively simple since the text body is a distinct attribute in the party tweet tables, and the SQLite3 select query returns records as a 2D array, each element of the 1st dimension is a record and the 2nd dimension is an array of fields.

The next part of the process, removing unnecessary data, was broken down into a series of functions:

- Tokenize
- Normalize
- Lemmatize
- Clean

Tokenization was handled by the `nltk.tokenize.casual.casual_tokenize()` method that takes a string of text as input, which it then breaks down into the constituent elements of the text, and returns an array of “tokens”. This method is optimised for the tokenization of texts that contain a higher concentration of casual, emotive language and additional elements such as emoji and hyperlinks. This perfectly fits the description of tweets and was ideal for this requirement. For example, the input tweet: “Our reach should always exceed our grasp. We’ve crossed the Rubicon. #Labour :’(“ would be output as:

```
[‘Our’, ‘reach’, ‘should’, ‘always’, ‘exceed’, ‘our’, ‘grasp’, ‘.’, ‘We’, ‘’, ‘ve’, ‘crossed’, ‘the’, ‘Rubicon’, ‘.’, ‘#Labour’, ‘:’(“]
```

Next came the process of normalization and lemmatization. Lemmatization is actually a form of normalization that considers the context of the word in the sentence and the word’s structure to convert it into its normalized form, a process known as morphological analysis. In order for tweets to be normalized in this way it was necessary to assign each token a tag that denoted its context in the sentence. Using the previous tokenized example, the normalization function would output:

```
[(‘Our’, ‘PRP$’), (‘reach’, ‘NN’), (‘should’, ‘MD’), (‘always’, ‘RB’), (‘exceed’, ‘VB’), (‘our’, ‘PRP$’), (‘grasp’, ‘NN’), (‘.’, ‘.’), (‘We’, ‘PRP’), (‘’, ‘VBP’), (‘ve’, ‘NNS’), (‘crossed’, ‘VBD’), (‘the’, ‘DT’), (‘Rubicon’, ‘NNP’), (‘.’, ‘.’), (‘#Labour’, ‘CD’), (‘:’(“ , ‘NN’)]
```

A full list of tags and their definitions can be found at the University of Pennsylvania’s website [25]. There are faster methods of normalization and stemming such as naively removing prefixes and suffixes, however there is evidence that using parts-of-speech tags is more effective for analysing sentiment [26][16]. Once the a tweet’s tokens had been assigned tags, they could be lemmatized, i.e. converted into their root forms (explained in more detail in section 6.5). The output of this function would be:

```
[‘Our’, ‘reach’, ‘should’, ‘always’, ‘exceed’, ‘our’, ‘grasp’, ‘.’, ‘We’, ‘’, ‘ve’, ‘cross’, ‘the’, ‘Rubicon’, ‘.’, ‘#Labour’, ‘:’(“]
```

Only one word in this example was in need of normalization: “crossed” was converted into “cross”. This ultimately increased the speed and accuracy when classifying tweets. Finally, as depicted in the flowchart, unnecessary data was removed from the tweets. Regular expressions were employed to parse tokens and remove links.



```
for word in lemmatizedTweet:
    word = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&#]|[*\(\),])|\'\'|\'(?:%[\0-9a-fA-F][0-9a-fA-F]))+\'', '', word)
    word = re.sub("@[A-Za-z0-9_]+", "", word)
```

The regular expressions in figure 7.6 are commonly used expressions for parsing links and are not unique to this project. Furthermore, empty strings, stop words, and punctuation were removed using the simple if statement pictured in figure 7.6.

```
if len(word)>0 and (word.lower() not in nltk.corpus.stopwords.words('english')) and word not in string.punctuation:
    cleanedWords.append(word.lower())
```

NLTK comes packaged with several sample texts called corpora, and the one used here is a file containing stop words in English. The complete list can be found on GitHub [27].

The final form of the example tweet would then be:

```
['reach', 'always', 'exceed', 'grasp', '', 'cross', 'rubicon', '#labour', ':(']
```

It is evident that the original tweets were greatly reduced in length. This led to the model being able to produce a more accurate sentiment classification as only elements likely to contain emotional semantics were retained. Furthermore the algorithm executed faster seeing as there were less tokens to analyse.

NLTK's NaiveBayesClassifier class was implemented for the actual task of tweet classification. In the sentiment analysis module of this system, there were two classes: positive and negative. The algorithm was trained using a dataset of 1.6 million tweets, 800,000 positive and 800,000 negative. This meant that the  $P(y)$  of both classes were equal at 0.5 and could essentially be ignored. This dataset is hosted at Kaggle [28]. K-fold cross validation was used to split the sample tweets into training and test sets with  $k = 10$  (see section 6.5).

For example, the system collects a tweet with the text "Labour is bad". This would be cleaned and tokenized to become ["labour", "bad"]. The equations would look like this:

$$P(\text{positive} | ["labour", "bad"]) = P(\text{positive}) \times P("labour" | \text{positive}) \times P("bad" | \text{positive}) \quad (7.1)$$

$$P(\text{negative} | ["labour", "bad"]) = P(\text{negative}) \times P("labour" | \text{negative}) \times P("bad" | \text{negative}) \quad (7.2)$$

The associated values might be something like:  $P(\text{positive}) = 0.5$   $P(\text{negative}) = 0.5$   $P(\backslash \text{Labour} | \text{positive}) = 0.05$   $P(\backslash \text{Labour} | \text{negative}) = 0.05$   $P(\backslash \text{bad} | \text{positive}) = 0.001$   $P(\backslash \text{bad} | \text{negative}) = 0.09$  Substituting these values into the two equations would give:

$$P(\text{positive} | ["labour", "bad"]) = 0.5 \times 0.05 \times 0.001 = 0.000025 \quad (7.3)$$

$$P(\text{negative} | ["labour", "bad"]) = 0.5 \times 0.05 \times 0.09 = 0.00225 \quad (7.4)$$

Since  $0.00225 > 0.000025$ , the tweet would be classified as negative. Obviously these figures are artificial, they exist only to show how the algorithm works in the context of tweet classification/sentiment analysis. One consideration is that this implementation of Bayes' theorem is described as naive, in that it assumes each event (word in the tweet) is independent. This is not

necessarily the case in the real world, it is quite obvious that “positive” or “negative” words are more likely to be followed by other “positive” or “negative” words respectively. This shortcoming could possibly affect the accuracy of classification, although Naive Bayes Classification is still considered a valuable tool for sentiment analysis given its ease of use and speed. In future development this would be considered and compared to other classification algorithms.

Some formula was required to combine these sentiments together to create an overall single value sentiment for each poll from their related tweets. Some of the parties have a larger following and media presence than others, i.e. the Conservative Party is a “bigger” party with more elected MPs than the Green Party. It follows that there were many more tweets collected related to the Conservatives than the Greens. This suggested that it would be much more useful to compare the relative sentiment of the parties rather than using absolute values. Consequently, the “total” sentiment figure towards a party was actually given by calculating the relative proportional difference using the formula:

$$\textit{Sentimenttowardsaparty} = \frac{P - N}{P + N} \quad (7.5)$$

$P$  is the total number of positive tweets towards the party and  $N$  is the total number of negative tweets towards the party. The bounds of this equation are  $-1 \leq \textit{sentiment} \leq 1$ , with 1 implying there were only positive tweets and -1 implying there were only negative tweets. If there were a similar number of positive tweets and negative tweets, the value was close to 0, a reasonable representation of there being no positive or negative majority. Another implementation of total sentiment included scaling by number of followers. This was achieved by simply multiplying the sentiment of each tweet (-1 or 1) by the follower count of the account that posted the tweet.

NLTK comes packaged with a pre-trained sentiment analysis model called Valence Aware Dictionary for sEntiment Reasoning (VADER), that is sensitive to positive/negative sentiment as well as intensity. The accuracies of the Naive Bayes classifier and VADER are compared in the section 8.1 and section 9.1.

Having a single process handle the cleaning and sentiment analysis proved to take quite a significant amount of time when processing a large volume of tweets. To remedy this, multithreading was attempted. Problems arose when the program attempted to remove stopwords from tweets during cleaning, it kept giving the exception `’’WordNetCorpusReader’ object has no attribute ’_LazyCorpusLoader__args’’`. After further research in to the NLTK WordNetCorpusReader documentation it was discovered that when this class is initialised to load a corpus, it actually instantiates a proxy object LazyCorpusLoader in its place that stands in for a corpus object before the corpus is loaded. Then when the object is first accessed in any way, the corpus is properly loaded. The reason for this is to allow NLTK to load an object for each corpus while deferring the cost associated with actually loading these corpora until they are actually used. This is not usually a problem in single threaded applications and exceptions are usually solved by using the `ensure_loaded()` function that forces the corpus to be properly loaded. It was discovered though, that even with the use of the `ensure_loaded()` function the WordNetCorpusReader class was not thread safe and the corpora weren’t loaded for each thread. This was solved relatively easily by instead implementing multiprocessing.

## 7.7 Relational table manager

In order to select all of the tweets that were relevant to a poll, the date of the poll was selected from the table. Python's datetime module was used to correctly generate the dates of the two days before the date of the poll. Then, the three dates were used to select all tweets from a party's sorted tweet table that were collected in this timespan. For each tweet selected, the primary key of the poll and the primary key of the tweet were inserted into an intermediate relational table as foreign keys of a new record. This process was followed for every new poll.

## 7.8 Linear and Polynomial Regression

Implementation of the linear regression model closely followed the order described in the design section (6.7). Sklearn contains a class called LinearRegression which greatly reduced development time rather than having to reinvent basic statistical operations. It was decided that tweets from 3 days up to and including the day a poll result is released would be associated with this poll. This time span was chosen primarily to maximise the number of polls available to use as tweet collection began relatively late due to some personal issues. In future development it may be useful to compare different time spans from more than a week to maybe even as short as 18 minutes. Studies have shown a tweet's lifespan is extremely short, with an average half life (time taken for a tweet to garner half of its total retweets) of about 18 minutes [29].

As before, these next steps will describe the operations performed when predicting vote intention for one party. Polls and tweets were related previously into intermediate tables, for example conPollTweets stores pairs of poll primary keys and Conservative relevant tweet primary keys as composite foreign keys. This would happen in parallel to the prediction function during real world use. Following the selection of the related tweets, their sentiment values were aggregated to determine the total sentiment value towards the party in each poll - including the option to scale each tweet's sentiment by follower count by simple multiplication. Then the poll results were inserted into a numpy array called y and the sentiments inserted into second numpy array named x. These arrays were passed to the fit() function of the instantiated LinearRegression model. The regression coefficients were calculated internally to minimise residuals and after a short time the model was trained. To reduce the processing overhead of retraining the model each time, the regression object was saved in a pickle file to preserve its state. This was not a long term solution as the model needed to be retrained on new data frequently.

When a user wanted a prediction for the percentage vote intention of a party on a date, this date would be used to select tweets leading up to this date. Next, the total sentiment was calculated and passed to the model as the independent variable using the predict() function. The model would then output a predicted value of the dependent variable: a percentage vote intention. This process was repeated for each party and the results returned to the user. After some analysis, the linear model was used for the official predictions yet the user had the option to switch to a polynomial model of an order of their choosing.

Sklearn's regression model objects have the ability to output the exact values of their  $r^2$ ,  $\epsilon$  and regression coefficients. For each party's models, these were collected and used to calculate root mean squared error and mean absolute error to allow appropriate evaluation of performance.

## 8. Testing

### 8.1 Model testing

#### 8.1.1 Sentiment analysis

In order to decide on which sentiment analysis model to use, the Naive Bayes classifier or VADER, some tests were carried out using the sentiment 140 dataset to quantify the error in classification for both models. Tweets in the sample dataset are pre classified into positive and negative, so to produce a value for accuracy they were input into the models minus their classification and the output predicted class was compared to their actual class.

VADER is pre trained and therefore the whole dataset was fed into the model. This produced an accuracy of 0.5209990756244223, meaning that VADER only correctly classified tweets 52.1% of the time, barely more accurate than a 50:50 guess.

VADER Classifier Accuracy
0.5209990756244223

Figure 8.1:

To test the Naive Bayes classifier, 10-fold cross validation was used to split the dataset into training and test portions. Each iteration, a different tenth of the data was excluded from the training set. After the model was trained, these excluded tweets were input into the model as a test set. The results for each of the 10 train-test splits are shown in this table:

Naive Bayes Classifier Accuracy
0.75803125
0.75718125
0.7598625
0.75721875
0.7589125
0.759125
0.75873125
0.7577
0.7582625
0.758898493115582

Figure 8.2:

The average accuracy calculated from the mean of these values is 0.7583923493115583, which is 50% better at classifying tweets than VADER. This made the decision to use the Naive Bayes classifier in the final product much more justified. There would be several methods of increasing the accuracy of classification, discussed in section 11.2.

### 8.1.2 Regression

Regression models were trained for all of the four parties, including first (linear), second and third order polynomial regressions for each. Data was collected on the performance of these models:

**Conservative party**

Order of regression function	y intercept	Regression coefficients	$r^2$	Root Mean Squared Error	Mean Absolute Error	Scaled by followers
1	41.93586172	0.2835592281	0.1969465579	1.011848737	0.9264995403	N
2	38.57347459	[1.52613379, -0.10235749]	0.2610604577	0.9831902471	0.9285906488	N
3	5.233629073	[19.90218758, -3.21396146, 0.16576495]	0.7037828458	0.9700175959	0.7082167921	N
1	39.16141222	0.2075963862	0.275642371	1.759177855	1.282232383	Y
2	37.92078277	[0.3158333, -0.00231685]	0.2758894476	1.625824052	1.221051726	Y
3	-119.7955958	[ 2.08300483e+01, -8.82367336e-01, 1.24526052e-02]	0.2980163329	12.448133	5.418435836	Y

Figure 8.3:

**Labour party**

Order of regression function	y intercept	Regression coefficients	$r^2$	Root Mean Squared Error	Mean Absolute Error	Scaled by followers
1	34.415931	-0.2267449422	0.191648963	1.854796126	1.541500807	N
2	34.45536855	[-0.45623381, -0.02902269]	0.271383176	1.700926799	1.162143274	N
3	34.15050776	[-0.45112828, 0.00359692, 0.0028319 ]	0.2753164284	1.812250985	1.378325221	N
1	35.6452304	-0.1060550184	0.04810132053	1.958788871	1.60657003	Y
2	35.68526224	[0.47335215, -0.06199603]	0.3783349801	5.839954368	3.235725103	Y
3	34.93651613	[ 0.3543802, 0.09224461, -0.01223647]	0.4383779593	11.0596826	5.321331032	Y

Figure 8.4:

**Liberal Democrat party**

Order of regression function	y intercept	Regression coefficients	r <sup>2</sup>	Root Mean Squared Error	Mean Absolute Error	Scaled by followers
1	7.433669787	-0.0456750399	0.3804804877	0.9399779498	0.7183828684	N
2	7.806941057	[-0.0201118, -0.00184803]	0.4663928935	1.08131458	0.8367703473	N
3	7.634846886	[-3.71980010e-03, -5.13990005e-04, -6.55901209e-05]	0.476883141	1.250806126	0.9094208111	N
1	7.236992653	-0.01638972894	0.08919093101	1.045191184	0.8428420477	Y
2	7.465428808	[0.00728733, -0.00087574]	0.1422183877	1.131378354	0.9905005918	Y
3	6.351900775	[ 0.08088471, 0.00714167, -0.00024195]	0.9912780911	3.835366604	1.516166259	Y

Figure 8.5:

**Green party**

Order of regression function	y intercept	Regression coefficients	r <sup>2</sup>	Root Mean Squared Error	Mean Absolute Error	Scaled by followers
1	-2.756281382	0.1592790524	0.05674927581	0.9986502867	0.8241302274	N
2	76.67585881	[-3.21373786, 0.03576314]	0.05755795188	0.9774301413	0.7924269721	N
3	315508.5539	[-1.99054675e+04, 4.18380429e+02, -2.92961933e+00]	0.4351115162	30.48913913	12.12015119	N
1	3.458434285	0.0636386502	0.1184294878	0.938939485	0.6491701656	Y
2	-5.270584517	[ 1.05165476, -0.02523013]	0.3762972173	0.8038981017	0.691307207	Y
3	-5.91325987	[-5.91325987, 0.34830551, -0.00637825]	0.7232045108	0.5203603437	0.4092269545	Y

Figure 8.6:

These results were self computed by the model using internal functions. It is useful to visualise the models as well as the predictions to gain a reference point for these figures. The three regression functions for the Conservative party (tweets not scaled by followers) are graphed in figure 8.7.

The actual and predicted poll results for the Conservative party interpolated using the three models are also graphed in figure 8.8, figure 8.9 and figure 8.10.

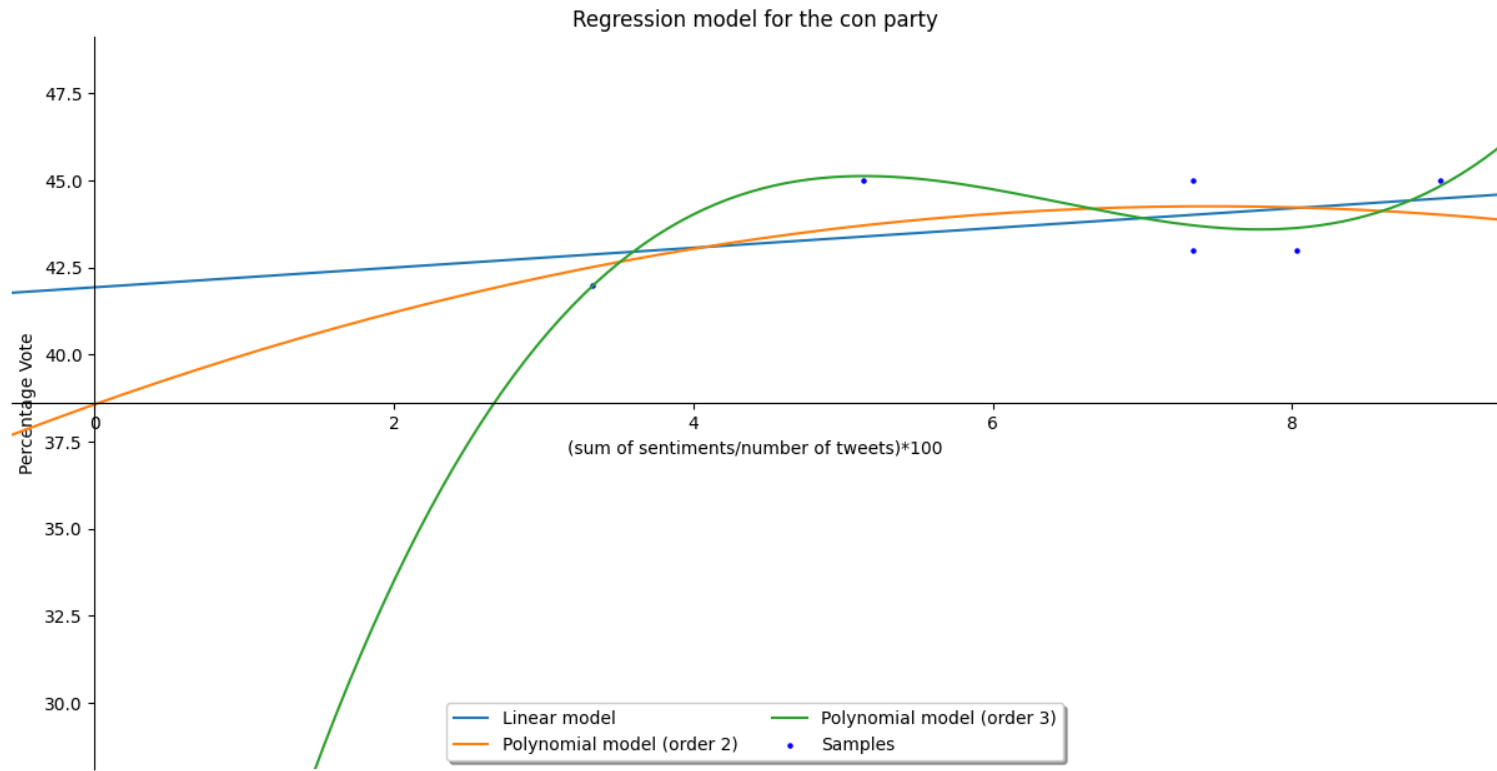


Figure 8.7:



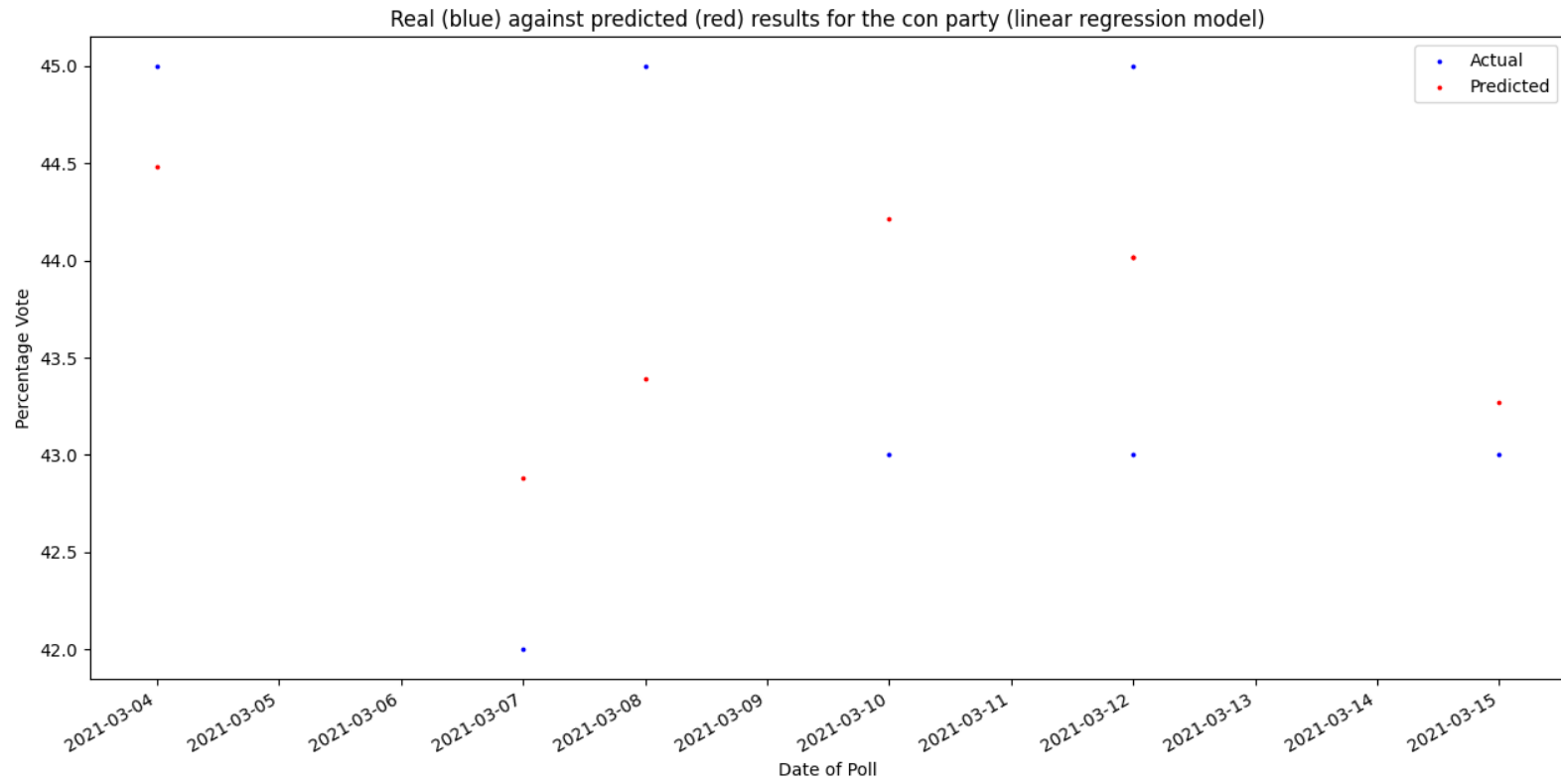


Figure 8.8:

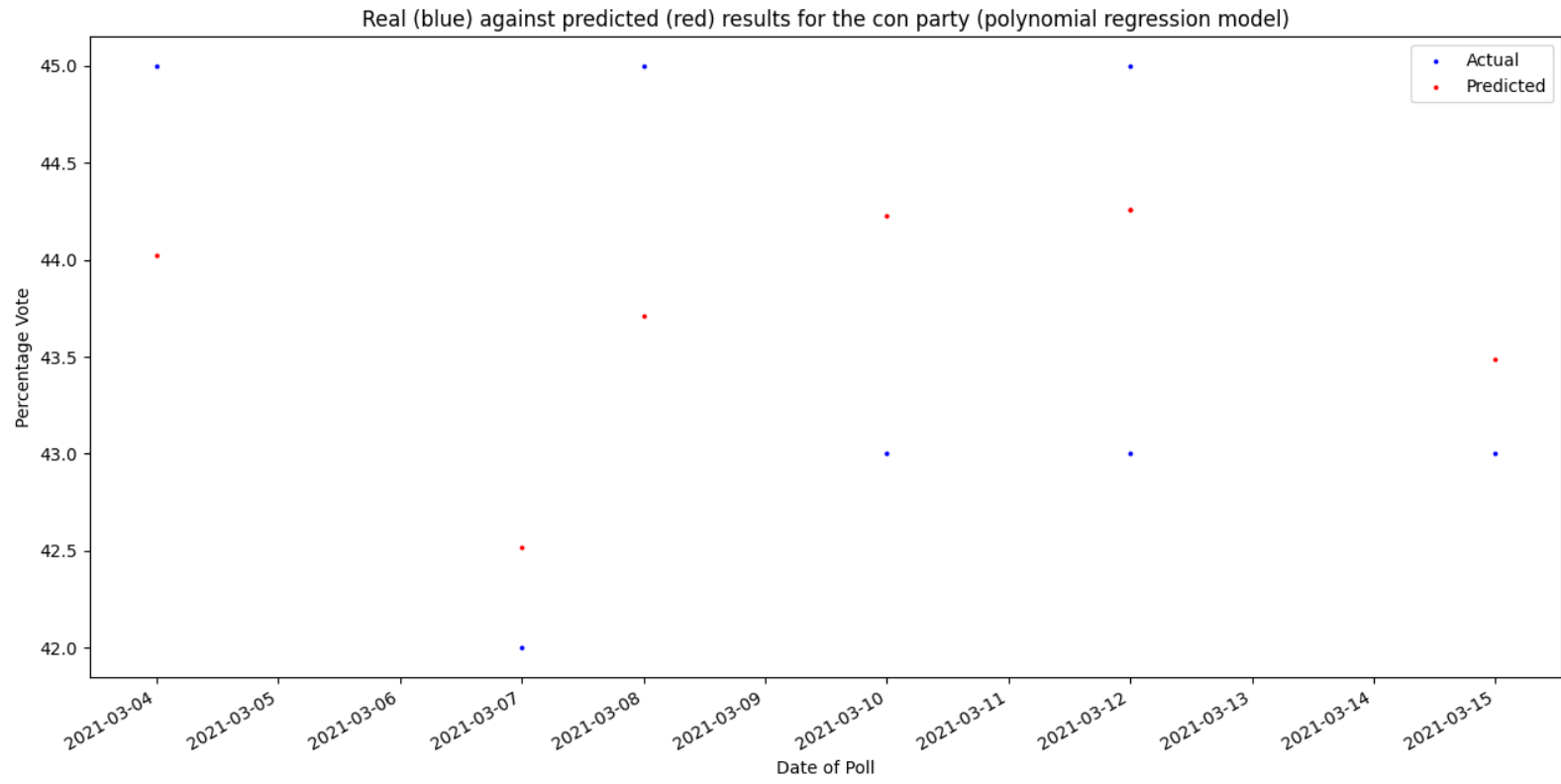


Figure 8.9:

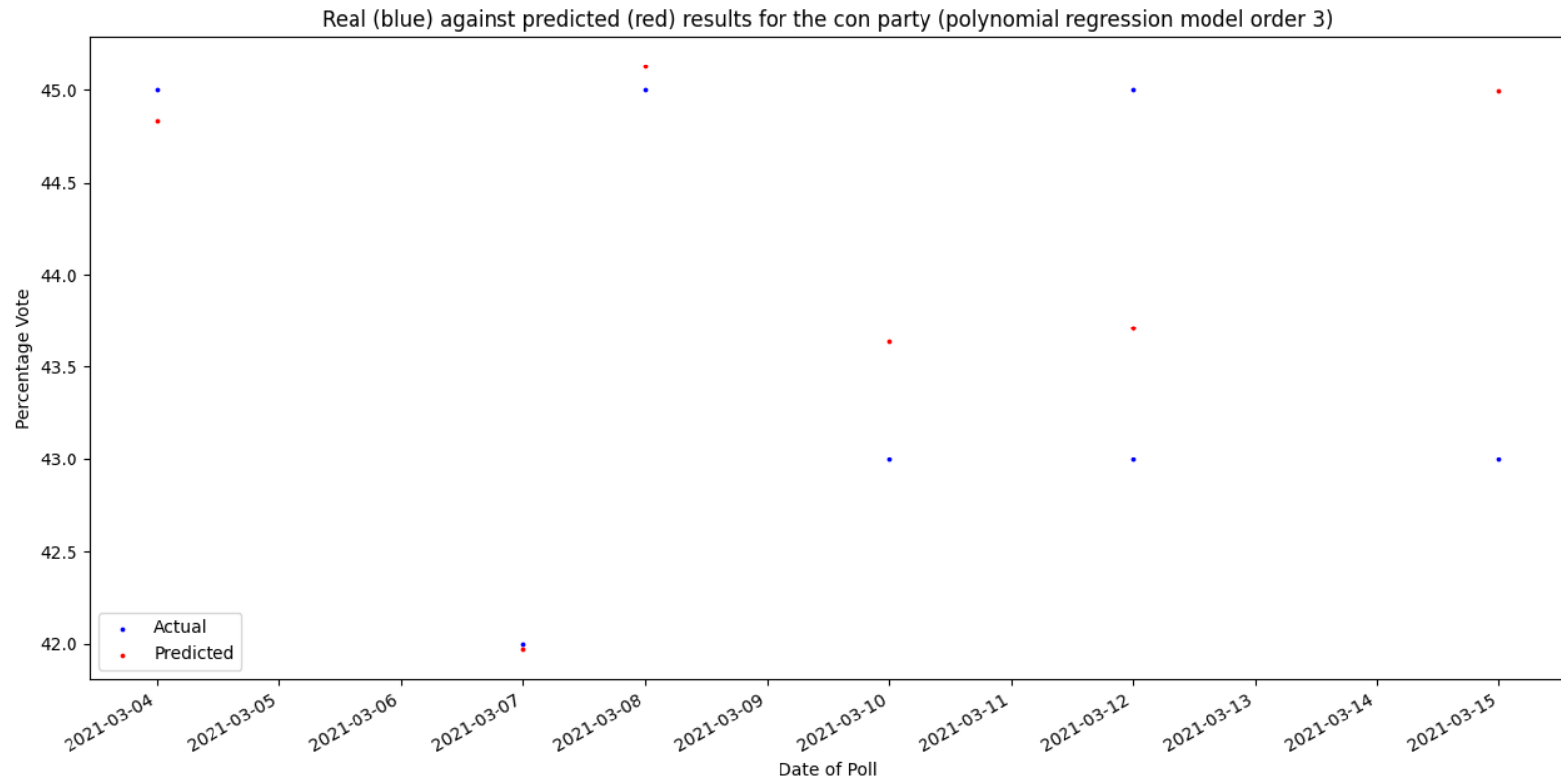


Figure 8.10:

It is clear by observing the graphed regression models that a lack of training data has led to inaccuracies; the polynomial regressions seem to overfit the data and the linear regression seems to greatly underfit. The implications of the predictions are discussed further in section 9.1.

## 8.2 Unit testing

Before being combined as a complete application, the individual modules were tested to ensure that the implementation was faithful to the design and able to meet the requirements detailed earlier in this paper. These tests don't contain any interaction between the different modules and used sample data consisting of expected data, erroneous data and boundary data in order to highlight any errors that might arise with extended use.

### 8.2.1 Database management

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Table creation	All tables created according to the schema SQL		Pass
2	Insertion, deletion and modification of records	DBMS recognises the SQL commands and changes the state of the database		Pass
3	Selection of multiple records	DBMS recognises queries and returns correct records		Pass
4	Concurrent access to the database	DBMS handles two processes modifying data in the DB at once	This is not consistent, sometimes fails due to the database being locked by one process	Fail

### 8.2.2 Stream listener

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Connection to Twitter streaming API	Authorisation codes sent to Twitter and subsequent successful receipt of tweets		Pass
2	Stream listener stays connected on error	Print error to terminal and back off for 60 seconds		Pass
3	Correct tweets received	Only tweets containing tracker words collected	Success (but some tweets are irrelevant even though they contain trackers)	Pass
4	Relevant data extracted from tweets	Tweet object correctly parsed to retain only relevant data		Pass

### 8.2.3 Tweet sorting

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Successful virtual table full text search	Unsorted tweets inserted into table in memory, full text search identifies UK locations		Pass
2	Non UK tweet removal	All non UK relevant tweets removed during sorting	Only some irrelevant tweets removed (not all location tagged)	Adequate pass
3	Tweets sorted into party tables	Tweets matching party specific trackers inserted into party's sorted tweets table		Pass

### 8.2.4 Poll updating

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Gather new polls	Parse HTML from poll record website and retain only new results		Pass
2	Update polls table	Insertion query used to store new polls in the database		Pass

### 8.2.5 Sentiment analysis

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Tokenize tweets	Tweets broken down into individual tokens, e.g. words and punctuation		Pass
2	Tag tokens	Label each token with the correct Part-Of-Speech tag		Pass
3	Lemmatize tokens	Convert each word token into its root form		Pass
4	Prepare training and test data	Use 10-fold cross validation to split the classified tweets into 10 chunks		Pass
5	Train and test Naive Bayes model	Use the chunks to train the model in 10 different ways and test for accuracy each time		Pass
6	Classify tweets	Output sentiment values for any input tweets		Pass

### 8.2.6 Relational table management

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Collect tweets relevant to a poll	Find the date of the poll and use it to select tweets from up to 2 days before		Pass
2	Relate tweets to poll	For each tweet related to a poll, insert a record into an intermediate table consisting of both primary keys		Pass

### 8.2.7 Prediction model

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Calculate total sentiment	Take a collection of classified tweets as input and calculate their relative proportional difference		Pass
2	Train and test prediction model	Given pairs of polls and sentiment values, train and test the accuracy of the model		Pass
3	Produce predictions	Given a date, calculate sentiment for this date and use it as input for the model. Output an exact percentage vote intention		Pass

## 8.3 Integration testing

Once a new feature was implemented and thoroughly tested, it was integrated into the main application. Further testing was required to confirm that the separate features worked together to produce the expected result. If there was any interference, git made it simple to revert the changes and solve the newly discovered problems. Integration testing loosely followed the interactions pictured in the level 1 data flow diagram to confirm data flowed through the system correctly from collection to prediction.

Test No.	Description	Expected result	Actual result (if different from expected)	Evaluation
1	Tweet streamer connects to DB	All collected tweets inserted into the unsorted tweet database		Pass
2	Unsorted tweets sorted into second DB	All new tweets selected from DB and inserted into party specific tables in another DB		Pass
3	Real sorted tweets related to real polls	Tweets selected by date of poll and IDs of tweets and polls inserted into relational table as foreign key		Pass
4	Real tweets classified by sentiment	Tweets selected from sorted table and classified. Tweets in sorted <u>table</u> modified to give them sentiment values.		Pass 
5	Regression models trained using real sentiment and poll data	Sentiment totalled for each poll. Total sentiment and poll pairs fed <u>into model</u> as training data.		Pass
6	Prediction for current date made	All components interact to produce a prediction for each party using current real world data.		Pass



## 9. Results and Discussion

### 9.1 Model accuracy

Order of model	Scaled by followers	Mean $r^2$
1	N	0.20645632109905393
2	N	0.26409861978489857
3	N	0.4727734828243334
1	Y	0.13284102759171657
2	Y	0.29318500816773396
3	Y	0.612719223511212

Figure 9.1: Average accuracies of each model

#### 9.1.1 Linear regression

Earlier in the research section (4.5) of the paper it was hypothesised that the dependency of vote intention for a party on tweet sentiment would most likely be best modelled as a linear relationship. Underwhelmingly, the mean coefficient of determination ( $r^2$ ) value of the relationship (not scaled by followers) is 0.20645632109905393. Predictions for the Green party are a large contributor to this, possessing an  $r^2$  value of 0.05674927581025835. Poor performance in this case is most likely explained by the presence of a larger proportion of irrelevant tweets. Tracker words for the Green party included very common non-political terms such as “green” - this word on its own caused the collection of many tweets that were not necessarily concerned with the Green party. Since linear regression attempts to model the specific relationship of political tweets on vote intention, inclusion of non-political tweets will have almost certainly caused relatively large inconsistencies in this relationship. A higher proportion of irrelevant tweets has likely led to the disappointingly low coefficient of determination.

Despite the obvious inaccuracies, the linear regression function for the Conservative party pictured earlier shows promise. Unlike the 2nd and 3rd order polynomial functions it shows no point at which an increasing sentiment leads to lower predicted vote intention and vice versa, whereas the polynomial models much more quickly show unrealistically extreme dependencies at the higher and lower vote intentions. Furthermore this suggests that the linear models are less prone to erroneous data points.

### 9.1.2 Polynomial regression

Observing the outcomes for the Conservative party without follower scaling gives a general idea of the overall results. As predicted, the  $r^2$  value for the 3rd order polynomial model is significantly larger than for the linear and 2nd order model and more importantly this doesn't seem to imply a greater general accuracy. It can be seen in the graphing of the models that as the total sentiment approaches 0, the percentage vote intention sharply declines implying an extreme relationship between tweet sentiment and vote intention, which is a prime example of overfitting. The y intercept has an unrealistically low value of 5.233629073170825. It is reasonable to state that a neutral overall sentiment towards a party shouldn't totally destroy the party's performance in the polls. This statement is backed by findings presented in section 4.3. Polynomial regression's unreliability is further evidenced by the RMSE of the 3rd order polynomial model for the Green party. Although the coefficient of determination is 7.5 times larger than for the 1st and 2nd order regression models, the root mean squared error is over 30 times larger than for the other two models.

It is very possible that once more data has been acquired, a section of a higher order polynomial function might more accurately describe the observed relationship between tweet sentiment and vote intention. This was not explored due to the limited time and computational power available. Another potential option going forward would be to research a wider array of statistical models that may be more suited to efficiently modelling higher order relationships.

### 9.1.3 Follower scaling

In regards to linear regression, scaling the sentiments of tweets by the account's number of followers seems to negatively affect the predictions. Despite this, the additional dimension shouldn't be dismissed until further data is collected and analysed.

On the other hand, the polynomial regression models boast consistently higher  $r^2$  values when scaled. This improved performance is perhaps related to political Twitter accounts generally having large numbers of followers, so when weighted by this metric, these accounts have more of an influence on the total sentiment than smaller accounts having general unrelated discussions. This does somewhat take away from the original aim of this project; if greater importance is placed on more popular accounts there is a risk of under-representing the general public.

# 10. Evaluation

## 10.1 Requirements

With development and testing of each fundamental process and successful integration into a complete system, evaluation of the finished application can be carried out. This involves investigating how successful the project has been, objectively and subjectively. Success is partly quantifiable through comparing the abilities of the final product with the original requirements set out in the design phase.

All of the functional requirements have been achieved (fig 10.1) well enough for the application to work as designed, as well as some nonfunctional requirements (fig 10.2). The requirements not achieved primarily concern the user experience. UX was not a priority while functionality of the back end had potential for improvement and therefore was mostly neglected.

## 10.2 Project management

Adopting a feature driven development style was ideal for this project. Having to ensure the correct functionality of each process in the system (detailed in the level 1 dfd) helped to maintain modularity in the code. Furthermore, since the completed components could provide data in its appropriately processed form as soon as they were finished, thorough unit and integration testing could be carried out all throughout development.

Splitting the requirements into functional and nonfunctional proved to be necessary for the completion of the project. If time had been spent fine tuning the UX, it is very possible that the underlying functionality behind the prediction model would be unfinished.

Requirement	Achieved	Description
1. The system must have a database of sufficient size to store past poll results and enough tweets to produce an accurate sentiment for each poll.	Yes	The SQLite3 database system does have some limitations but it has been sufficient for the current scope of the project. All of the necessary SQL operations have been implemented successfully.
2. For each poll result stored, the system must identify and implement a relationship that tracks which tweets are relevant to which political party.	Yes	The relational database design, plus data normalisation, have allowed tweets to be related to polls in an intermediate table.
3. The system must actualize a stream listener that collects tweets in real time that are relevant to the Conservative Party, Labour Party, Liberal Democrat Party and the Green Party, storing only those tweets that contain certain party specific phrases.	Partially	The stream listener is functional and collects tweets according to tracker words, however some irrelevant tweets are collected as some of the trackers aren't necessarily confined to just political language.
4. The system must carry out tokenizing and cleaning of tweets so that they can be effectively analysed for sentiment.	Yes	Every tweet is tokenized and normalised using part-of-speech tags and lemmatization for the improvement in sentiment analysis performance.
5. The system must implement a sentiment analysis model that classifies tweets as positive or negative sentiment.	Yes	A Naive Bayes classifier has been trained on a sample tweet dataset to classify real world tweets as positive or negative.
6. The system must have a method of measuring the strength of the dependency of a party's poll result on their total Twitter sentiment.	Partially	The linear and polynomial regression models attempt to calculate the strength of the dependency, but the lack of training data stemming from having to collect tweets in real time has led to the models being inaccurate. The models will improve the accuracy of their predictions over time as they are periodically retrained on up to date information.
7. The system must be able to compile a value for the total sentiment towards a party, inside certain time parameters, from the sentiment of each related tweet.	Yes	Although the output values may be somewhat inaccurate, they are still predictions based on sentiment. Because the tweet/poll collection, sentiment analysis, database management etc. are constantly running, the only processing that is required when a user requests a prediction is to train the model, the output is relatively quick.
8. The system must allow a user to input a particular date and receive an exact vote percentage prediction for each party given the sentiment towards the party running up to said date.	Yes	All of the actual poll results and tweets are constantly being updated to ensure temporally relevant predictions.

Figure 10.1: Functional requirement evaluation

Requirement	Achieved	Description
1. The sentiment analysis model should be a custom trained Naive Bayes classifier that, when tested, produces a greater accuracy than NLTK's built in VADER sentiment analysis feature.	Yes	The accuracy of the Naive Bayes classifier is ~50% greater than the VADER module.
2. The system should be usable at any time.	Partially	If one possesses the program code, the prediction model script can be executed at any time. This ties in to the original plan to develop a web app, which was unachievable.
3. The system should present the predictions in a user friendly, understandable manner.	No	There is little user interface since development was focused on the back end function of each feature.
4. The system should outperform the average human in its predictions for poll results.	Not measured	Over time this should become more likely.
5. The system's predictions should improve over time, learning from actual poll results as they are released.	Yes	The models are retrained on new data periodically.
The system should be integrated into a website for ease of access.	No	There was not enough time to begin development of the web app.

Figure 10.2: Nonfunctional requirement evaluation

# **11. Conclusion**

## **11.1 Summary**

The project successfully achieved the main goal of predicting the outcomes of future polls. A user can input the current date and the program will give an exact percentage vote intention for each party calculated using current Twitter based sentiment. The underlying hypothesis that the opinion of the general public can affect performance in the polls has been measured in a way that allows informed predictions. The most significant weakness of the current system is the inaccuracy of these predictions. Currently, it is unlikely to be accurate enough to provide much value to real world professional fields. However due to the system's ability to update its models with new data, it will continue to improve over time. Additionally, there are a lot of potential improvements that could be made to the prediction model implementation. With more funding and time the scope of the application could be significantly increased. Most importantly, the fundamental base of each functional requirement has been met, and the system as a whole has great potential for expansion in several areas detailed in the next section.

## **11.2 Future work**

### **11.2.1 Predictions**

The accuracy of the regression model should improve over time. However, improvements in the application's accuracy could be achieved by testing several other models. It is possible that other statistical models may have increased accuracy and performance when analysing these kinds of trends.

### **11.2.2 Sentiment analysis**

There are several possible changes that could be made to improve the sentiment analysis feature, listed in figure 11.1.

Potential change	Explanation
Use a different classification algorithm.	There may be algorithms more suited to classifying this form of data with a higher accuracy. One example is a feedback algorithm that would be able to learn from its own inaccuracies. In addition, some models may be able to measure intensity of sentiments.
Use more training data.	The more training data provided, the more the algorithm can learn which tokens are more associated with positive or negative sentiments.
Use a greater variety of training data.	If trained with a more varied set of tweets, the algorithm would more accurately recognise the sentiment of a larger range of tokens.
Allow classification of neutral tweets.	The accuracy of the algorithm may be skewed as some tweets may not have any discernible positive or negative sentiment. Currently they must either be classified as positive or negative.
Implement topic modelling.	The system would more consistently remove irrelevant tweets from the database.
Collect data from other social media sites.	This would theoretically lead to a better representation of the UK population's political opinions.

Figure 11.1: Potential changes to the sentiment analysis algorithm

### 11.2.3 Website

The website never came to fruition due to time concerns so it is an obvious next step in future development. It would add much needed usability and clarity to the system and would be an easy way of distributing the software.

## 12. Legal, Social and Ethical Issues

All of the data that will be used in this project is public, however the people may not have intended for their tweets to be analysed in such a manner, and furthermore Twitter may deem the application's use of their data to be harmful to their business or against their terms of service. Political opinion can be a sensitive subject, and people can find even objective political fact personally offensive.

Any future expansion of the application, especially releasing the project for public use or licensing privately, would require further communication with Twitter for approval. If the company deemed it an unfair use of their product, the entire project could be compromised.

To prevent interference from Twitter and backlash from its users, use of their data will always follow the Twitter developer terms and conditions, and any personal data will be removed from collected tweets before they are processed. The prediction results, being the product of large quantities of data manipulated by several mathematical algorithms, will be abstracted to the point where it would be impossible to trace any data back to an individual. This should be sufficient to satisfy General Data Protection Regulation (GDPR) [30].

In addition, there are personal ethical obstacles that might impede the development of the application into a larger entity. Use of tools like this by political parties could undoubtedly help them to change aspects of their campaigns or their social media presence, and while this is not directly interfering with the democratic process, it could certainly be used as a tool to aid in malicious manipulation and propaganda.



# Bibliography

- [1] Statista Research Department. *Leading countries based on number of Twitter users as of July 2021*. URL: <https://www.statista.com/statistics/242606/number-of-active-twitter-users-in-selected-countries/>.
- [2] Pritee Salunkhe and Sachin Deshmukh. "Twitter Based Election Prediction and Analysis". In: (2017). URL: <https://www.irjet.net/archives/V4/i10/IRJET-V4I1094.pdf>.
- [3] Ramteke et al. "Election result prediction using Twitter sentiment analysis". In: (2016). URL: <https://ieeexplore.ieee.org/document/7823280>.
- [4] Robert Hudson, Kevin Keasey, and Mike Dempsey. "Share prices under Tory and Labour governments in the UK since 1945". In: (2010). URL: <https://www.tandfonline.com/doi/abs/10.1080/096031098332925>.
- [5] Galen A. Irwin and Joop J. M. van Holsteyn. "According to the Polls: The Influence of Opinion Polls on Expectations". In: (2002). URL: <https://www.jstor.org/stable/3078698>.
- [6] Andranik Tumasjan et al. "Predicting Elections with Twitter: What 140 Characters Reveal about Political Sentiment". In: (2010). URL: <https://ojs.aaai.org/index.php/ICWSM/article/view/14009/13858>.
- [7] Tsakalidis et al. "Predicting Elections for Multiple Countries Using Twitter and Polls". In: (2015). URL: <https://ieeexplore.ieee.org/abstract/document/7021854>.
- [8] Pete Burnap et al. "140 characters to victory?: Using Twitter to predict the UK 2015 General Election". In: (2016). URL: <https://www.sciencedirect.com/science/article/pii/S0261379415002243>.
- [9] Batini et al. "Methodologies for data quality assessment and improvement". In: (2009). URL: <https://dl.acm.org/doi/10.1145/1541880.1541883>.
- [10] Yair Wand and Richard Y. Wang. "Anchoring Data Quality Dimensions in Ontological Foundations". In: (1996). URL: <http://web.mit.edu/tdqm/www/tdqmpub/WandWangCACMNov96.pdf>.
- [11] Ceron A et al. "Every tweet counts? How sentiment analysis of social media can improve our knowledge of citizens' political preferences with an application to Italy and France". In: (2013). URL: <https://journals.sagepub.com/doi/epub/10.1177/1461444813480466>.
- [12] Kokil Jaidka et al. "Predicting elections from social media: a three-country, three-method comparative study". In: (2017). URL: <https://www.tandfonline.com/doi/abs/10.1080/01292986.2018.1453849?scroll=top&needAccess=true&journalCode=rajc20>.

- [13] Fabio Franch. “(Wisdom of the Crowds)2: 2010 UK Election Prediction with Social Media”. In: (2013). URL: <https://www.tandfonline.com/doi/abs/10.1080/19331681.2012.705080?journalCode=witp20>.
- [14] Tom W. G. van der Meer, Armen Hakhverdian, and Loes Aaldering. “Off the Fence, Onto the Bandwagon? A Large-Scale Survey Experiment on Effect of Real-Life Poll Outcomes on Subsequent Vote Intentions”. In: (2015). URL: <https://academic.oup.com/ijpor/article/28/1/46/2357273?login=true#37522011>.
- [15] Sebastian Raschka. “Naive Bayes and Text Classification I: Introduction and Theory”. In: (2014). URL: <https://arxiv.org/pdf/1410.5329.pdf>.
- [16] Hassan Saif, Yulan He, and Harith Alani. “Semantic Sentiment Analysis of Twitter”. In: (2012). URL: <https://link.springer.com/content/pdf/10.1007%2F978-3-642-35176-1.pdf>.
- [17] Kent Beck et al. *Manifesto for Agile Software Development*. 2001. URL: <https://agilemanifesto.org/>.
- [18] Gerard Coleman. “Investigating Software Process in Practice: A Grounded Theory Perspective”. In: (2006). URL: [http://doras.dcu.ie/17296/1/gerard\\_coleman\\_20120705132657.pdf](http://doras.dcu.ie/17296/1/gerard_coleman_20120705132657.pdf).
- [19] Philip S. Taylor et al. “Preparing Small Software Companies for Tailored Agile Method Adoption: Minimally Intrusive Risk Assessment”. In: (2008). URL: <http://www.cs.qub.ac.uk/~Des.Greer/SPIPTaylorGreer.pdf>.
- [20] *Opinion polling for the next United Kingdom general election*. URL: [https://en.wikipedia.org/wiki/Opinion\\_polling\\_for\\_the\\_next\\_United\\_Kingdom\\_general\\_election](https://en.wikipedia.org/wiki/Opinion_polling_for_the_next_United_Kingdom_general_election).
- [21] *Data dictionary: Standard v1.1*. URL: <https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/object-model/tweet>.
- [22] *Best Hashtags*. URL: <https://best-hashtags.com/>.
- [23] *RiteTag*. URL: <https://ritetag.com/>.
- [24] *List of MPs with active Twitter accounts organised by party*. URL: <https://www.politics-social.com/list/party>.
- [25] *Alphabetical list of part-of-speech tags used in the Penn Treebank Project*. URL: [https://www.ling.upenn.edu/courses/Fall\\_2003/ling001/penn\\_treebank\\_pos.html](https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html).
- [26] Apoorv Agarwal et al. “Sentiment Analysis of Twitter Data”. In: (2011). URL: <https://aclanthology.org/W11-0705.pdf>.
- [27] *NLTK’s list of English stopwords*. URL: <https://gist.github.com/sebleier/554280>.
- [28] *Sentiment140 dataset with 1.6 million tweets*. URL: <https://www.kaggle.com/kazanova/sentiment140>.
- [29] Peter Bray. *When Is My Tweet’s Prime of Life? (A brief statistical interlude.)* 2012. URL: <https://moz.com/blog/when-is-my-tweets-prime-of-life>.
- [30] *General Data Protection Regulation*. 2018. URL: <https://gdpr-info.eu/>.

## A. Appendix A: Important code

### A.1 Stream listener

```
# Variables that contains the credentials to access Twitter API
ACCESS_TOKEN = '3099117383-ybROCSwGlrqSNgNWYAGU1PlZ5FNFkzdyYDruplo'
ACCESS_SECRET = 'HPGbi7jHo8WSDMSz1EYL4dhFucmnz2roTqVwidfbNoM5E'
CONSUMER_KEY = 'GMTtHcXjvMnpFhs8BVHP0RcSs'
CONSUMER_SECRET = 'sO7DyfPs3cve3vzb1shhEM30vWewJEf3gADi7MAk6qpntTUmz'

# Setup access to API
def connect_to_twitter_OAuth():
    auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
    auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)

    api = tweepy.API(auth, wait_on_rate_limit=True, wait_on_rate_limit_notify=True, compression=True)
    return api
```

Figure A.1: Set up connection to Twitter API

```
# Create API object
api = connect_to_twitter_OAuth()

class StreamListener(tweepy.StreamListener):
    #Inherit from tweepy streamlistener and override methods because they are stubs
    def on_status(self, status):
        #print(status.entities["text"])
        tweet = self.extractData(status)
        insertTweet(tweet)
        print("tweet inserted")

    def on_error(self, status_code):
        if status_code == 420:
            print("420 error")
            time.sleep(60)
            return True # continue listening
        logger.info('Error: status %s', str(status_code))
        return True # continue listening

    def on_timeout(self):
        logger.info('Timeout: pausing 60s')
        time.sleep(60)
        return True # continue listening
```

Figure A.2: First part of the stream listener class

```
def extractData(self, status):
    data = []
    data.append(status.id)
    data.append(status.user.id)
    data.append(status.text)

    hashtags = ""
    for tag in status.entities["hashtags"]:
        hashtags = hashtags + tag["text"] + ","
    data.append(hashtags)

    data.append(str(status.user.location))
    data.append(str(status.coordinates))
    data.append(str(status.created_at))
    data.append(status.user.followers_count)
    data.append(status.retweet_count)
    data.append(status.favorite_count)
    data.append(status.in_reply_to_user_id)
    data.append('')

    return data
```

Figure A.3: Second part of the stream listener class

## A.2 Tweet sorting

```
#Selects all of the tweets that haven't been sorted and loads them into a virtual table in memory so the full text search is faster
def sortTweets():
    maxId = getLargestExistingId()
    conn = sqlite3.connect('ukpoliticstweets.db')
    c = conn.cursor()
    c.execute('SELECT * FROM tweets WHERE id > ?;',(maxId,))
    tweets = c.fetchall()
    conn.close()
    conn = sqlite3.connect('ukpoliticstweets2.db')
    c = conn.cursor()
    c.execute('SELECT * FROM tweets WHERE id > ?;',(maxId,))
    tweets += c.fetchall()
    for tweet in tweets:
        tweet = list(tweet)
        tweet[5] = tweet[5].replace('?', '').replace('""', '').replace(',', '')
    conn = sqlite3.connect([':memory:'])
    c = conn.cursor()
    c.execute('CREATE VIRTUAL TABLE IF NOT EXISTS vTweets using fts5(newId,id,user,text,hashtags,location,coordinates,date,followers,retweets,favourites,replyToId,party, tokenize="porter unicode61");')
    c.execute('CREATE VIRTUAL TABLE IF NOT EXISTS vTweets2 using fts5(newId,id,user,text,hashtags,location,coordinates,date,followers,retweets,favourites,replyToId,party, tokenize="porter unicode61");')
    c.executemany('INSERT INTO vTweets (newId,id,user,text,hashtags,location,coordinates,date,followers,retweets,favourites,replyToId,party) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);', tweets)
    local = placeSearch(c, conn)
    c.execute('DELETE FROM vTweets;')
    c.executemany('INSERT INTO vTweets (newId,id,user,text,hashtags,location,coordinates,date,followers,retweets,favourites,replyToId,party) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?);', local)
    partySearch('con', c, conn)
    print('con')
    partySearch('lab', c, conn)
    print('lab')
    partySearch('libdem', c, conn)
    print('libdem')
    partySearch('green', c, conn)
    print('green')
    conn.commit()
    conn.close()
```

Figure A.4: Main function for sorting tweets and removing irrelevant data

## A.3 Poll updating

```
#Turns the 2020/2021 table into an array
def newTableToArray(rows,index):
    headers = rows[0].find_all('th')
    array = []

    for header in range(len(headers)):
        headers[header] = headers[header].text.replace('\n', '')
    rows.pop(0)

    for row in rows:
        data = row.find_all('td')
        if len(data) == len(headers):
            for datum in range(len(data)):
                if datum == 2:
                    data[datum] = parseDate(data[datum].text.replace('\n',''),2021-index)
                else:
                    data[datum] = data[datum].text.replace('\n', '').replace('%','')
            array.append(data)
    return array

#Converts the array to a dataframe
def newPollsToDF(index):
    soup = getHTML('https://en.wikipedia.org/wiki/Opinion_polling_for_the_next_United_Kingdom_general_election')
    table = getNewTable(soup,index)
    data = newTableToArray(table,index)
    df = pd.DataFrame(data,columns=['pollster','client','date','area','samplesize','con','lab','libdem','snp','green','others','lead'])
    df = df.drop(['client','area','samplesize','snp','others','lead'],axis=1)
    return df
```

Figure A.5: Scrape and parse polls from the internet

```
#Takes a dataframe of polls and inserts them into the database
def pollsToDB(polls):
    conn = sqlite3.connect('sortedTweets.db')
    c = conn.cursor()
    for index, row in polls.iterrows():
        c.execute('INSERT INTO polls (pollster,date,con,lab,libdem,green) VALUES (?, ?, ?, ?, ?, ?);',(row['pollster'],row['date'],row['con'],row['lab'],row['libdem'],row['green']))
    conn.commit()
    conn.close()

#Takes brand new polls and inserts them into the database
def newPollsToDB():
    newPolls = newPollsToDF(0)
    conn = sqlite3.connect('sortedTweets.db')
    c = conn.cursor()
    earliest2021Id = 2137
    c.execute('SELECT pollster,date,con,lab,libdem,green FROM polls WHERE id > ?',(earliest2021Id,))
    existing = dbToDf(c.fetchall())
    complement = newPolls.merge(existing, how = 'outer', indicator=True).loc[lambdax : x['_merge']=='left_only']
    complement.drop(['_merge'],axis=1)
    print('Polls to be added to db:')
    print(complement)
    pollsToDB(complement.iloc[:-1])
```

Figure A.6: Insert new polls into the database

## A.4 Sentiment analysis

```
#Parallel processing safe tweet cleaning functions
def tweetCleanerP(tweet):
    wn.ensure_loaded()
    tokenizedTweet = tokenizeP(tweet)
    normalizedTweet = normalizeP(tokenizedTweet)
    lemmatizedTweet = lemmatizeP(normalizedTweet)
    cleanedTweet = cleanP(lemmatizedTweet)
    return cleanedTweet

#First the tweet is broken down into tokens
def tokenizeP(tweet):
    return nltk.tokenize.casual.casual_tokenize(tweet)

#The words are converted to their canonical forms
def normalizeP(tokenizedTweet):
    return nltk.tag.pos_tag(tokenizedTweet)

def lemmatizeP(normalizedTweet):
    lemmatizer = nltk.stem.wordnet.WordNetLemmatizer()
    lemmatizedWords = []
    for word, tag in normalizedTweet:
        if tag.startswith('NN'):
            pos = 'n'
        elif tag.startswith('VB'):
            pos = 'v'
        else:
            pos = 'a'
        lemmatizedWords.append(lemmatizer.lemmatize(word, pos))
    return lemmatizedWords
```

Figure A.7: Tokenize, tag and lemmatize tweet

```
#The links are stripped out and the stopwords and punctuation removed
def cleanP(lemmatizedTweet):
    cleanedWords = []
    for word in lemmatizedTweet:
        word = re.sub('http[s]?://(?:[a-zA-Z]|[0-9]|[$-_@.&+]|[*\(\)\,])'\
            '(?:%[0-9a-fA-F][0-9a-fA-F])+', '', word)
        word = re.sub("([A-Za-z0-9_]+)", "", word)

        if len(word)>0 and (word.lower() not in nltk.corpus.stopwords.words('english')) and word not in string.punctuation:
            cleanedWords.append(word.lower())
    return cleanedWords

#Takes a newly collected tweet and cleans it
def collectedTweetCleaner(tweet):
    wn.ensure_loaded()
    tweet[1] = tokenizeP(tweet[1])
    tweet[1] = normalizeP(tweet[1])
    tweet[1] = lemmatizeP(tweet[1])
    tweet[1] = cleanP(tweet[1])
    return tweet
```

Figure A.8: Clean tweet

```

#Trains the naive bayes classifier
def trainNaiveBayes():
    cleanPosTweets, cleanNegTweets = getCleanTweets('cleanPosTweets.pkl','cleanNegTweets.pkl')
    modelPosTweets = tweetsForModel(cleanPosTweets)
    modelNegTweets = tweetsForModel(cleanNegTweets)
    posDataset, negDataset = labelTweets(modelPosTweets,modelNegTweets)
    global dataset
    dataset = posDataset + negDataset
    random.shuffle(dataset)
    return trainModel(dataset)

#Trains the naive bayes model
def trainModel(dataset):
    classifier = nltk.NaiveBayesClassifier.train(dataset)
    return classifier

```

Figure A.9: Train Naive Bayes classifier

```

#Uses k fold cross validation to split the tweets in to training and test tweets with a 9:1 split
#Utilises multiprocessing to create a pool of 4 processes which severely reduces running time
def testNaiveBayes():
    cleanPosTweets, cleanNegTweets = getCleanTweets('cleanPosTweets.pkl','cleanNegTweets.pkl')
    modelPosTweets = tweetsForModel(cleanPosTweets)
    modelNegTweets = tweetsForModel(cleanNegTweets)
    posDataset, negDataset = labelTweets(modelPosTweets,modelNegTweets)
    global dataset
    dataset = posDataset + negDataset

    testResults = []
    for x in range(20):
        kfold = KFold(10, shuffle=True, random_state=1)
        splitData = kfold.split(dataset)

        traintest = []
        for train, test in splitData:
            traintest.append([train,test])

        pool = ThreadPool(4)
        results = []
        results = pool.map(trainTestModel,traintest)
        testResults.append(["test"+str(x),results])

    with open('testResultsNaiveBayes.txt','w') as f:
        f.write(str(testResults))

```

Figure A.10: Test Naive Bayes classifier



```
#Takes the newly collected tweets and feeds them into the classifier, returns the sentiment of each tweet
def analyseCollectedTweets(tweets):
    classifier = loadClassifier('naiveBayesClassifier.pkl')
    tweets = collectedTweetsForModel(tweets)
    for tweet in range(len(tweets)):
        tweets[tweet].append(classifier.classify(tweets[tweet][1]))
    return tweets

#Updates the newly evaluated sentiment to the sortedTweets database
def updateSentiment(tweets,party,file):
    conn = sqlite3.connect(file)
    c = conn.cursor()
    for tweet in tweets:
        newSentiment = 0
        if tweet[3] == 'positive':
            newSentiment = '1'
        elif tweet[3] == 'negative':
            newSentiment = '-1'
        c.execute('UPDATE '+party+'Tweets SET sentiment = ? WHERE '+party+'Id = ?;',(newSentiment,tweet[0]))
    conn.commit()
    conn.close()

#Classifies all new tweets
def collectedTweetProcessor(party,file):
    tweets = cleanCollectedTweets(file,party) #gets tweets with no sentiment and cleans them
    print('cleaned ' + party + ' tweets examples:')
    print(tweets[:5])
    tweets = collectedTweetsForModel(tweets) #formats tweets for model input
    tweets = analyseCollectedTweets(tweets) #uses classifier to classify tweets
    updateSentiment(tweets,party,file) #updates the sentiment in sortedTweets.db
```

Figure A.11: Classify new tweets

## A.5 Relational table manager

```
#Gets all new polls and relates tweets to them
def fillRelationalTable(party):
    conn = sqlite3.connect('sortedTweets.db')
    c = conn.cursor()
    c.execute('SELECT id,date FROM polls WHERE id > ?;',(2304,))
    allPolls = c.fetchall()
    c.execute('SELECT DISTINCT pollId FROM '+party+'PollTweets;')
    existingPolls = c.fetchall()
    for poll in range(len(existingPolls)):
        existingPolls[poll] = existingPolls[poll][0]
    polls = []
    for poll in allPolls:
        if poll not in existingPolls:
            polls.append(poll)
    for poll in polls:
        tweetIds = getPollRelevantTweets(poll[1],conn,c,party)
        for id in tweetIds:
            try:
                c.execute('INSERT INTO '+party+'PollTweets (pollId,partyId) VALUES (?,?)',(poll[0],id))
            except:
                print('pollId = '+str(poll[0])+' '+party+'Id = '+str(id))
        conn.commit()
    conn.close()
```

Figure A.12: Relate tweets to new polls

```
#Selects only the ids of the tweets from up to 3 days before the specified poll
def getPollRelevantTweets(date,conn,c,party):
    dates = []
    date1 = datetime.strptime(date, '%Y-%m-%d')
    date2 = date1 - timedelta(days=1)
    date3 = date1 - timedelta(days=2)
    date4 = date1 - timedelta(days=3)
    dates.append(date1.strftime('%Y')+'-'+date1.strftime('%m')+'-'+date1.strftime('%d'))
    dates.append(date2.strftime('%Y')+'-'+date2.strftime('%m')+'-'+date2.strftime('%d'))
    dates.append(date3.strftime('%Y')+'-'+date3.strftime('%m')+'-'+date3.strftime('%d'))
    dates.append(date4.strftime('%Y')+'-'+date4.strftime('%m')+'-'+date4.strftime('%d'))
    c.execute('SELECT '+party+'Id FROM '+party+'Tweets WHERE date IN (?, ?, ?, ?);',dates)
    tweets = c.fetchall()
    tweetIds = []
    for tweet in tweets:
        tweetIds.append(tweet[0])
    return tweetIds
```

Figure A.13: Get tweets relevant to new polls

## A.6 Prediction model

```
#Selects the vote percentage and sentiment for each poll
def getPercentageAndSentiment(pollId,party,conn,c):
    sql = 'SELECT '+party+'Tweets.sentiment,polls.'+party+' FROM polls INNER JOIN '+party+'PollTweets ON '+party+'PollTweets.pollId = polls.id INNER JOIN '+party+'Tweets ON '+party+'Tweets.'+party+'Id = '+party+'pollId'
    c.execute(sql,(pollId,))
    sentAndPercent = c.fetchall()
    return sentAndPercent

#Takes all sentiments and calculates the overall sentiment
def calculateTotalSentiment(sentAndPercent):
    pos = 0
    neg = 0
    for data in sentAndPercent:
        if data[0] == '1':
            pos += 1
        elif data[0] == '-1':
            neg += 1
    return ((pos-neg)/(pos+neg))*100

#Selects sentiment, followers and vote percentage for each poll's tweets
def getPercentageAndSentimentScaledByFollowers(pollId,party,conn,c):
    sql = 'SELECT '+party+'Tweets.sentiment,polls.'+party+', '+party+'Tweets.followers FROM polls INNER JOIN '+party+'PollTweets ON '+party+'PollTweets.pollId = polls.id INNER JOIN '+party+'Tweets ON '+party+'Tweets.'+party+'Id = '+party+'pollId'
    c.execute(sql,(pollId,))
    sentAndPercent = c.fetchall()
    return sentAndPercent

#Takes all sentiments multiplied by the followers of the poster and calculates the overall sentiment
def calculateTotalSentimentScaledByFollowers(sentAndPercent):
    pos = 0
    neg = 0
    for data in sentAndPercent:
        if data[0] == '1':
            pos += int(data[0])*data[2]
        elif data[0] == '-1':
            neg += int(data[0])*data[2]*-1
    return ((pos-neg)/(pos+neg))*100
```

Figure A.14: Calculate total sentiment for a party in a poll

```
#Gets the data needed for predictions, overall sentiment and the percentage the party got in each poll
def collectData(party):
    conn = sqlite3.connect('sortedTweets.db')
    c = conn.cursor()
    c.execute('SELECT DISTINCT pollId from '+party+'PollTweets')
    pollIds = c.fetchall()
    results = []
    for id in pollIds:
        sentAndPercent = getPercentageAndSentiment(id[0],party,conn,c)
        totalSentiment = calculateTotalSentiment(sentAndPercent)
        results.append([totalSentiment,sentAndPercent[0][1]])
    conn.close()
    return results

#Does the same as collectData but the overall sentiment is scaled by the no. of followers of the poster of each tweet
def collectDataScaledByFollowers(party):
    conn = sqlite3.connect('sortedTweets.db')
    c = conn.cursor()
    c.execute('SELECT DISTINCT pollId from '+party+'PollTweets')
    pollIds = c.fetchall()
    results = []
    for id in pollIds:
        sentAndPercent = getPercentageAndSentimentScaledByFollowers(id[0],party,conn,c)
        totalSentiment = calculateTotalSentimentScaledByFollowers(sentAndPercent)
        results.append([totalSentiment,sentAndPercent[0][1]])
    conn.close()
    return results
```

Figure A.15: Select and format data for the regression model

```
#Formats the sentiments and percentage votes and uses them to train a linear regression model
def trainModel(data):
    x = []
    y = []
    for pair in data:
        x.append(pair[0])
        y.append(int(pair[1]))
    x = np.array(x)
    y = np.array(y)
    x = x.reshape((-1, 1))
    model = LinearRegression().fit(x,y)
    r_sq = model.score(x, y)
    modelMetrics = []
    modelMetrics.append(r_sq)
    modelMetrics.append(model.intercept_)
    modelMetrics.append(model.coef_[0])
    return model, modelMetrics

#Formats the sentiments and percentage votes and uses them to train a polynomial regression model
def trainPolynomialModel(data,degree):
    x = []
    y = []
    for pair in data:
        x.append(pair[0])
        y.append(int(pair[1]))
    x = np.array(x)
    y = np.array(y)
    x = x.reshape((-1, 1))
    x_ = PolynomialFeatures(degree=degree, include_bias=False).fit_transform(x)
    model = LinearRegression().fit(x_,y)
    r_sq = model.score(x_, y)
    modelMetrics = []
    modelMetrics.append(r_sq)
    modelMetrics.append(model.intercept_)
    modelMetrics.append(model.coef_)
    return model, modelMetrics
```

Figure A.16: Train linear or polynomial model

```
#Trains the appropriate model and then uses it to predict the outcome of polls on given dates
def predictPercentage(party,dates,polynomial,degree):
    data = collectData(party)
    if not polynomial:
        model, modelMetrics = trainModel(data)
    else:
        model, modelMetrics = trainPolynomialModel(data,degree)
    conn = sqlite3.connect('sortedTweets.db')
    c = conn.cursor()
    predictions = []
    for date in dates:
        tweets = getPollRelevantTweets(date,conn,c,party)
        sentiments = []
        for tweet in tweets:
            c.execute('SELECT sentiment FROM '+party+'Tweets WHERE '+party+'Id = ?;',(tweet,))
            sentiments.append(c.fetchone())
        totalSentiment = calculateTotalSentiment(sentiments)
        x = np.array([[totalSentiment]])
        if polynomial:
            x = PolynomialFeatures(degree=degree, include_bias=False).fit_transform(x)
        predictions.append(model.predict(x))
    return predictions, modelMetrics
```

Figure A.17: Make prediction using a model

```
#The same as predictPercentage but with follower scaled sentiment
def predictPercentageScaledByFollowers(party,dates,polynomial,degree):
    data = collectDataScaledByFollowers(party)
    if not polynomial:
        model, modelMetrics = trainModel(data)
    else:
        model, modelMetrics = trainPolynomialModel(data,degree)
    conn = sqlite3.connect('sortedTweets.db')
    c = conn.cursor()
    predictions = []
    for date in dates:
        tweets = getPollRelevantTweets(date,conn,c,party)
        sentiments = []
        for tweet in tweets:
            c.execute('SELECT sentiment,date,followers FROM '+party+'Tweets WHERE '+party+'Id = ?;',(tweet,))
            sentiments.append(c.fetchone())
        totalSentiment = calculateTotalSentimentScaledByFollowers(sentiments)
        x = np.array([[totalSentiment]])
        if polynomial:
            x = PolynomialFeatures(degree=degree, include_bias=False).fit_transform(x)
        predictions.append(model.predict(x))
    return predictions, modelMetrics

#Main function that gets the predicted percentages for each party on given dates
def finalPredictions(dates,polynomial,degree):
    for party in ['con','lab','libdem','green']:
        predictions, modelMetrics = predictPercentage(party, dates, polynomial, degree)
        for prediction in range(len(predictions)):
            print(party + ' party percentage vote prediction for ' + dates[prediction] + ': ' + str(round(predictions[prediction][0],2)) + '%')
```

Figure A.18: More prediction options

# B. Appendix B: Additional predictions

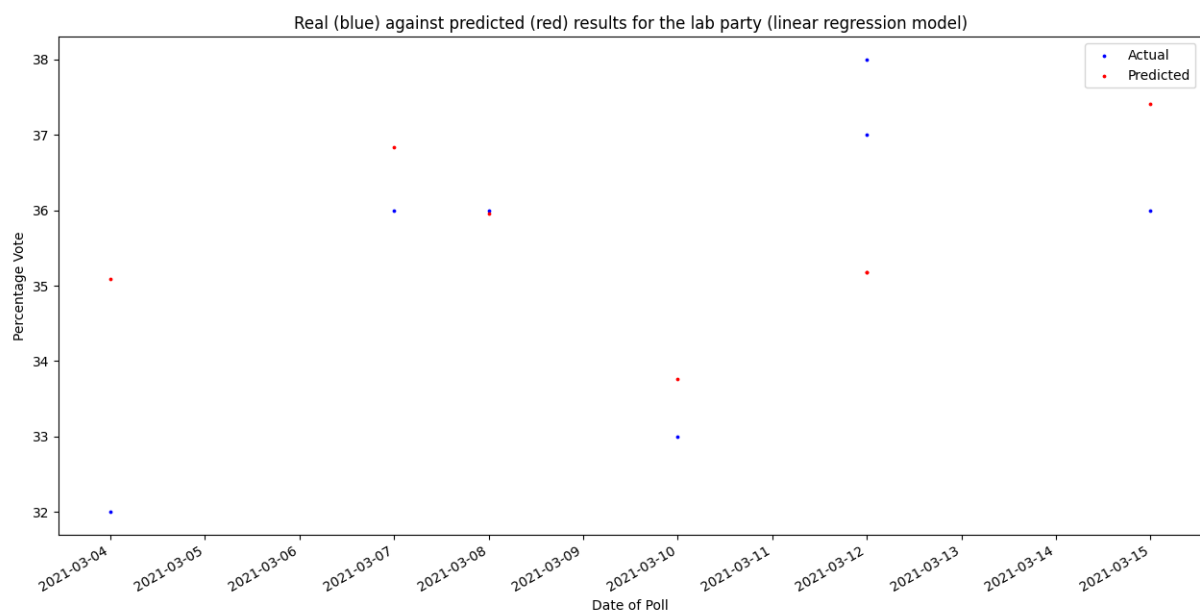


Figure B.1:

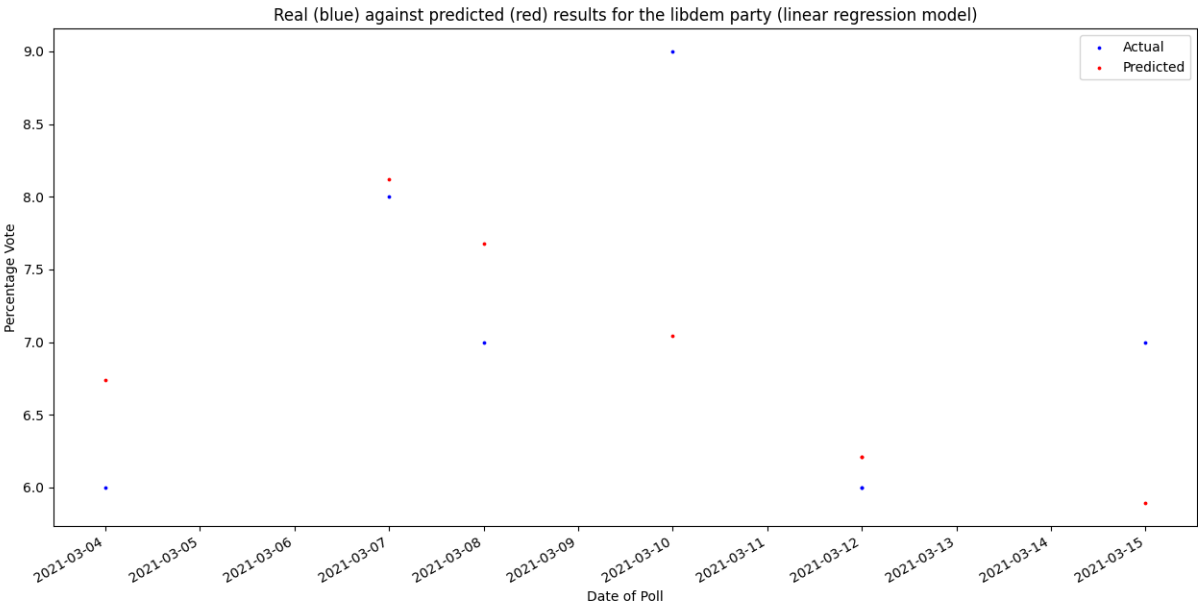


Figure B.2:

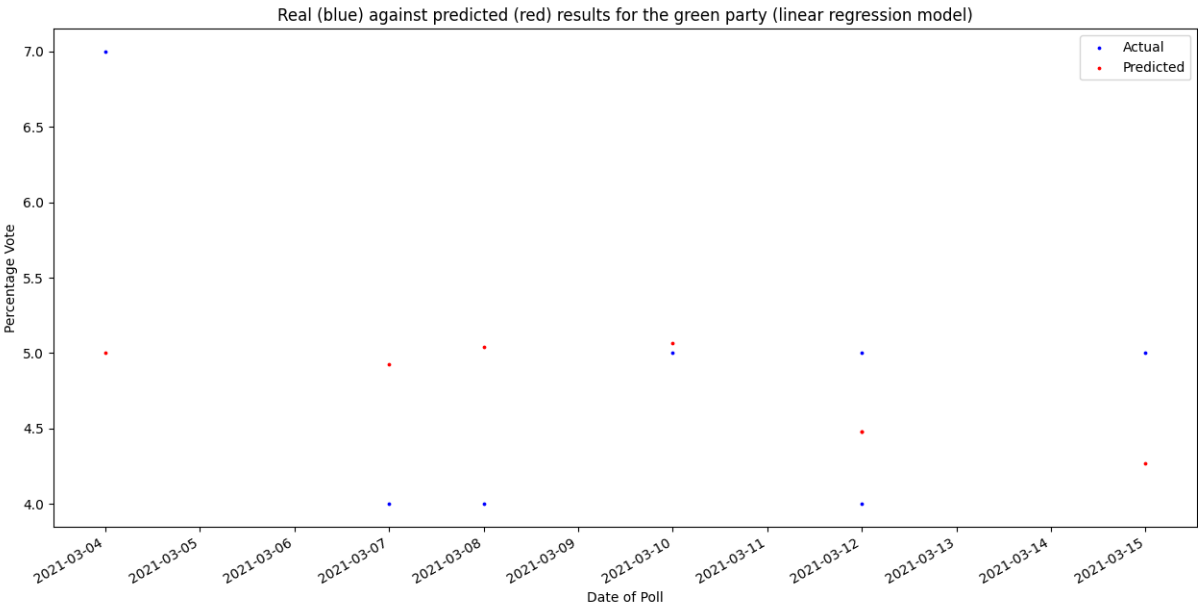


Figure B.3:



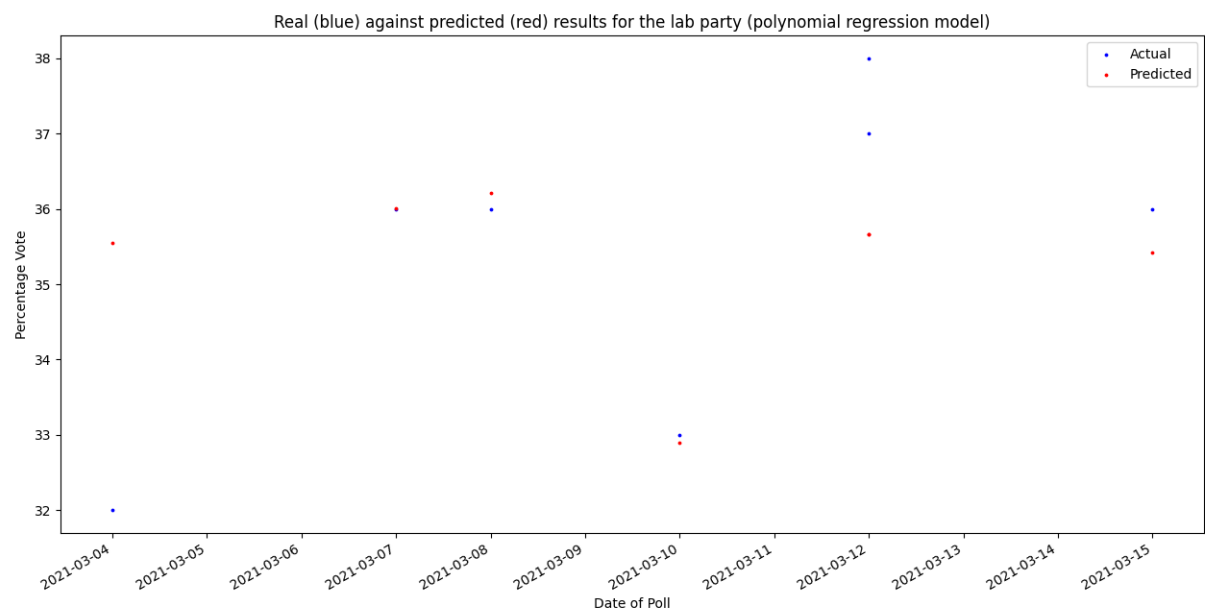


Figure B.4:

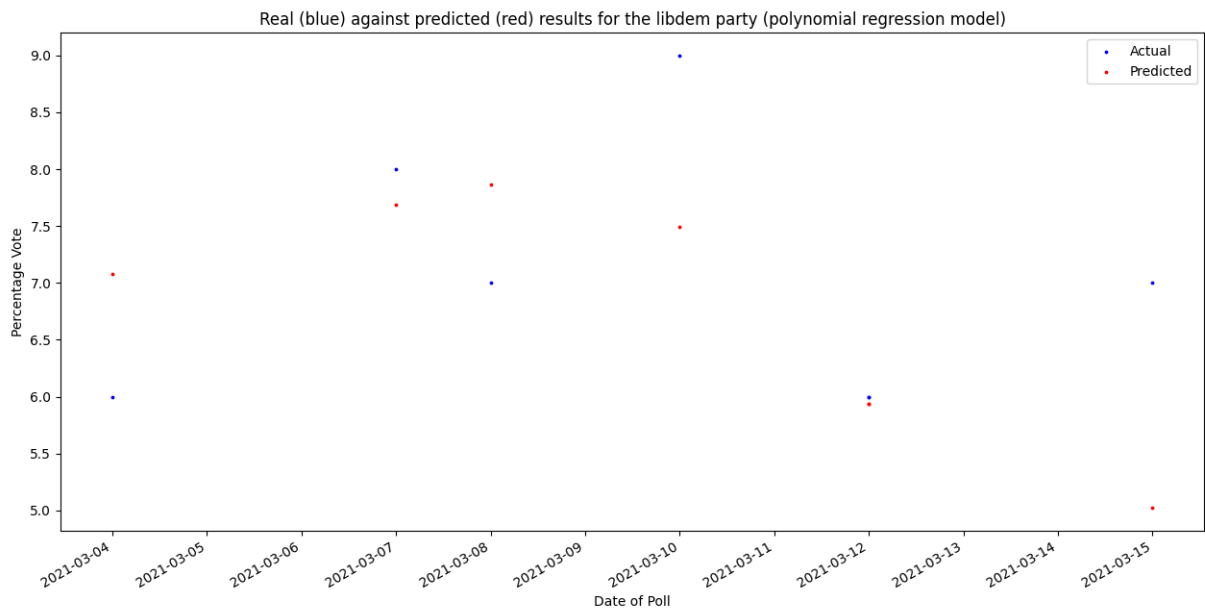


Figure B.5:

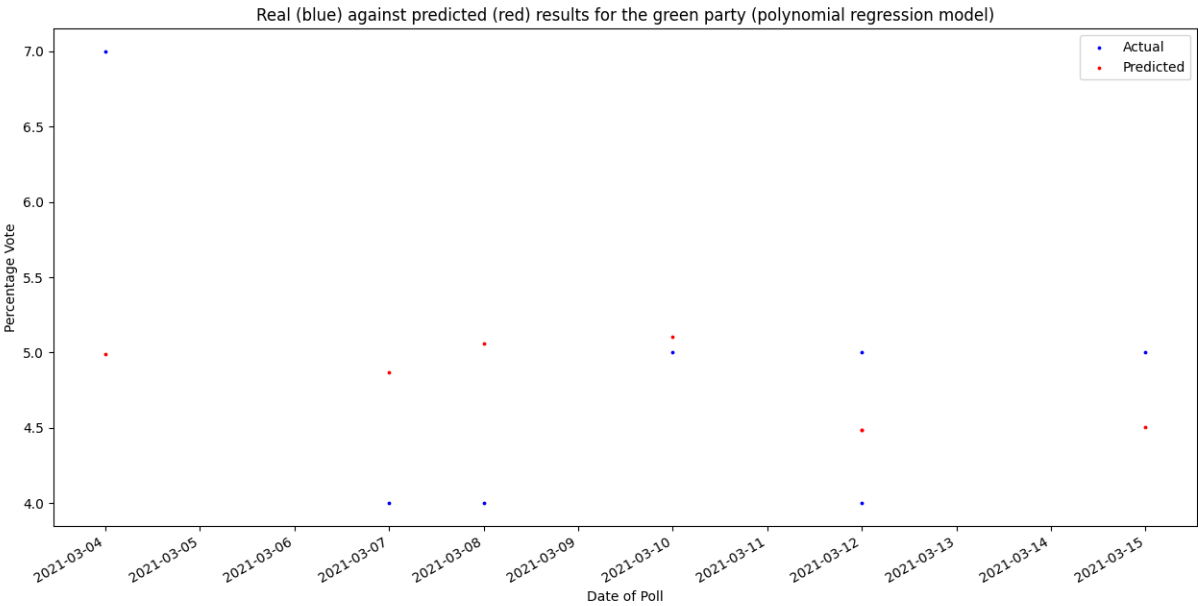


Figure B.6:

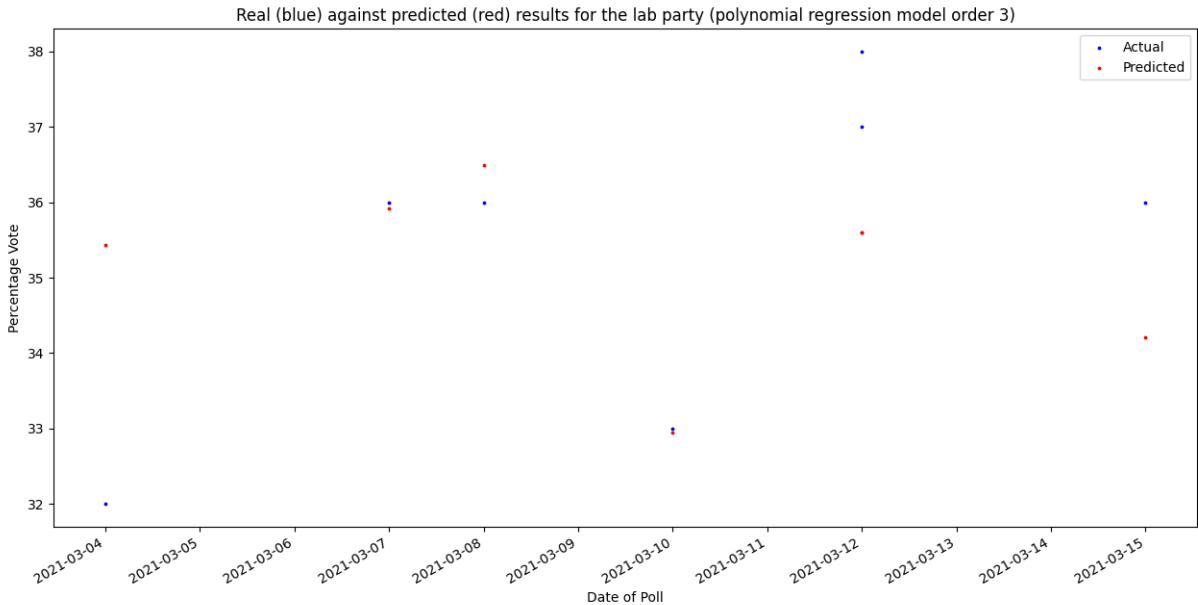


Figure B.7:

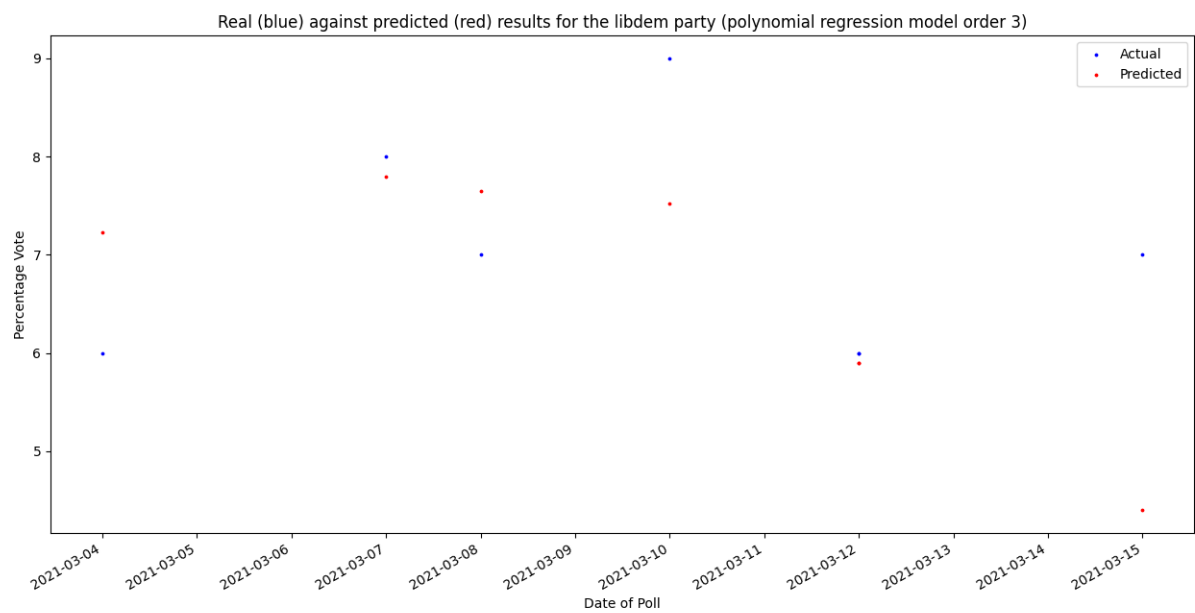


Figure B.8:

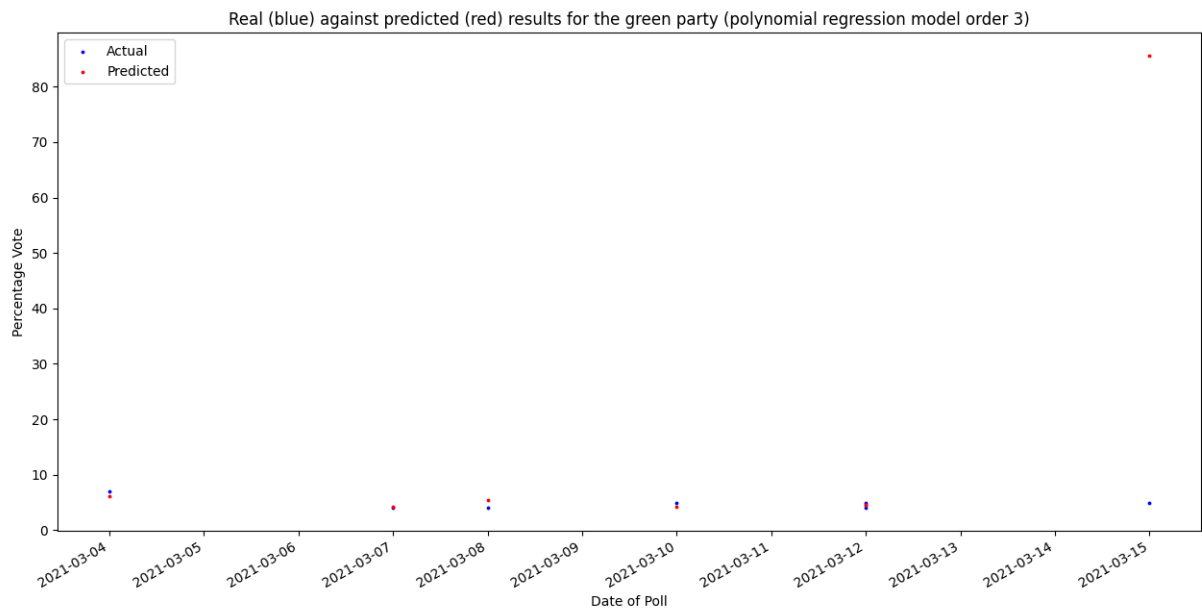


Figure B.9:

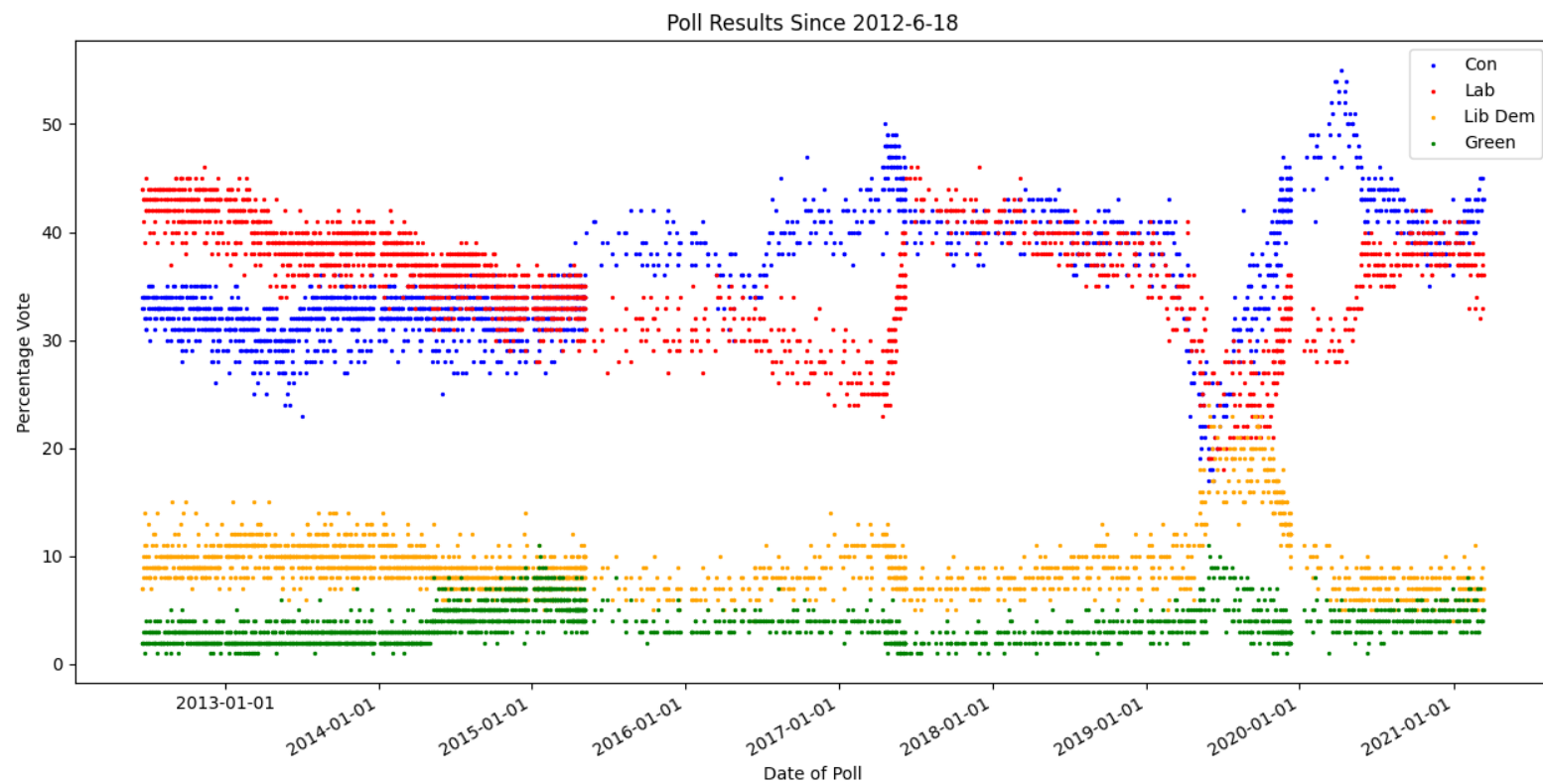


Figure B.10:

## C. Appendix C: Git

```
6f6b433aa92e5953db9cf4ac7e37494d1e40c38f (HEAD -> main, origin/main) Cleaning up
4d7348a6c5015fd4e9a6056559fece72ae1c8634 Ready for presentation
15ba93c200835d38058b8238de350d8b7170c2ae Added final vote predictions
f5f2309c438aeab6b3e2bd496ca537e009c85ac Plotted regression models for the con party
5f8cf35f3168974ee100689e821710939a32bad9 Done linear regression analysis
5b1e49110ac79578f960a852f8a1cc3fd6bffee4 Deleting old tweets
956a7a23b1015b703bfcca5fdae16301a3f13fd3 Made delete old tweets function
ace55cc7273e62e2752762f454841d76498be389 GOT PREDICTIONS!!!!!!!!!!!!!!
310ece51b748864551dbe12e5f1932504ab7d55c Initial ml model commit
c993c1934788a19c6b39ee386fa3d911a5fb65d3 Created relational tables between polls and each party's tweets
7fcd489a51029968e87fd23d3820105f2405c86b Updated gitignore to ignore.pkl
1117b18378f79c89366089c1fbaa9442ae712587 Git keeps playing up
b334ecb47992f04d7882b0a1076e2779526dcebb Made selectTweets.py more efficient
226d49126b9c0a54ba5685ce0605a29faa9574b0 Can now analyse sentiment of collected tweets!!!!
fab015d2f35b273eba08276474609950ef5151a3c Got all polls into a db
1a2d9b043545ccf60fb55a716dbd4215fea547aa Cleaned up a bit and saved classifier as pickle
27aec470f1534e7625d45dd40f28ea593558b16f Can read/write cleaned tweets and train model
32e86eaf4cd1d9e789bea5723569aedae5c03e74 Created txt of cleaned tweets
e84b1173783479a5034334be18e505818744eb29 Updated to send cleaned tweets to txt
1b605c75c7b9e10c0936def5ec484b2977963d43 Formatting tweets for model
110e0502f5d19dff99d55e5d8dc29049e838cb0c Finished tweet cleaning script
45b63aa429756e1c941918e7969a0a1dedfbcadf Updated gitignore
f0f051870388a21359b5d98d569bfa56c95cbdff8 Tokenized tweets
5c5864b2b840dc7aac6f9e536115c5a4ad325542 Finished sorting tweets into party and location
1acd4089016952ba6f50f4cd6072383aff93165a Created new db for party sorted tweets
9d8413c30409daaa7c37d87ea6f6ade8647a0a0 Working selection of tracker relevant tweets
7c89cf4d7b2f083a3833de3b3dbb1146cc3fdac2 selectTweets gets phrases from txt files
59ba50b91371f8fccc0c7b8daf5eb66ec07c90a New file selectTweets.py creates a virtual table in memory for a full text search
f93140d24ed77493895feabca3be452ddc71ca9f Downloaded nltk corpora
e4359aa1a4e097f3edc6465b6d54c3bce271d071 Added both db to gitignore
500e4de590a69a6e4045794131c9aae15cab2008 (origin/twitter) Added sleep functions and a try except block to avoid errors
50369382fac36a3029ac64c842d4e99975adeced Commit before bed
5584ac6276b71c38c2e7efa2cf4af493b614490c (twitter) Completely separated the two streams including a second dbcontrol and second db file
b1dcab5d7701c10bbfc5710a190bf37d5aaaa981 removed *.csv from gitignore
4368aa275960b20479575cd9b19a45230f6d96a5 Created requirements.txt to copy the venv packages to pc
77263e9abf6899a959c338844a4bfb832e35ce6b Copied twitter streaming program to allow 2 twitter streams to run at once
fc2ad0d39ef49050ffe622445a9dfd433bc2133a Read csv to get mp twitter handles rather than their names
462002f422657e584eeca5b5650789e31309a1c Fixed connecting to db and finished adding mp names to tracker list
172c6b1c14cfb31c89592056fad51a952af274ae MP names are now added to the relevant party's phrase list
9531e1fba926edf7e7b3721a34be27ea7be234a5 Output mp names and parties to text file
e0991089ab318e34ee8be3528669c8bfc87dd326 Implemented scraping of MP names
32802a830498efa4275ca344b664b3d516c345dc Collecting key phrases and initialised mpparser which will scrape relevant mp names
bf656a477ea39eaa56ff3775c996b1ae78c24d30 Implemented basic tweet stream to db
c51d4e58fb54253d79c043837697b5b395c4ae6f Added drop table for testing purposes
59b62e574690acbd4dc3fdadd1242c6b87719b38 Tested stream listener and added replyToId to db
b194fdcfa50174a57974dcba8a551383c173d38c Merge twitter into main
484cabdd077fe4740fe6ade9ebbe0d43d668e6a5 Gave up inserting tweets because of bug and added db to gitignore
579f3dfcbdd11735e0466fae35db758fcaaf6c59 twitter authentication and csv gitignore
ad1d5519f7f9b488384827a2a21355e1be2f4e4c added basic dbcontrol.py
```

Figure C.1: Git commit history