UK Parliamentary Data Analysis

For DE2 MEng Data Science Module

Maximilian Matthews[a], Chinene Chukwuma[a], Owain Pill[a], William Jiang[a]

[a] *Dyson School of Design Engineering, Imperial College London*

June, 2022

Abstract

The project we embarked on was to determine the political, social, and economic situation of constituencies by look at data that was collected from the constituency dashboard on the Parliament website and Census data. Each group member looked at a different aspect of these questions so that we would get an overall view on what features of a constituency influenced it the most and use this to decide where to live. The team all used at least one of the following methods: Ordinary Linear Regression, Logistic Regression, Decision Trees. After each member had conducted their analysis, the results were that it was possible to get a picture of the factors in question (accuracy of all models was above 0.8). Most factors could be well predicted with fewer than 5 features and the most features in a final model was 10.

Contents

Figure 1: UK Parliament Website

1. Introduction

With the rise of globalization and continuous technology advancement, the level of flexibility and freedom for general public to move from one region or locale to another to seek gainful employment in their field and better standard or quality of living have become increasingly higher.

However, the escalation of geographical mobility has proposed a new challenge to us – which place is better for living and working than the others?

1.1. Project Aim

In this project, our group attempts to create a model to predict the quality of living of each constituency in the UK by analysing its social, political and economic conditions. The results could be used as a reference for one that is seeking to transfer from one constituency to another.

2. Related Work

SAS Paradise Found project uses data science and machine learning to calculate the "best place on earth" to live. In comparison to city rankings that are often determined by editorial choices based on limited statistics and criteria, SAS uses a self-learning algorithm that uses a dataset with 5 million data points from thousands of sources that cover 193 countries and 148,233 cities. The sources are compromised largely of publicly available data, including city studies, social media sites, review sites like TripAdvisor, geodata and reports from statistical services and international agencies.

The first city ranking using AI determined that the inner suburb of West Perth in Australia is the algorithm's choice for best place to live. SAS said it is often too easy for unconscious bias to affect results when selecting the criteria to use when determining which data should be collected and analysed so they processed all the available data and allowed machine learning algorithms to decide which criteria were important.

Using a variable reduction technique, the key criteria was cut down to eight, namely: living expenses; safety and infrastructure; healthcare; restaurants and shopping; the environment; culture; attractiveness to families; and education and employment.

3. Methodology

Given the strain the cost-of-living crisis has put on households, many people currently living in big cities such as London are considering whether to move to another, more affordable, location. We aim to predict key indicators such as the number of businesses and house prices in an area to allow people to determine whether they want to live there.

We chose to use a group of datasets freely available on the UK Parliament website. They offer data on key metrics broken down by UK Parliamentary Constituency. This gives us the geographic variation needed for our investigation. The data is originally sources from either the UK Census or other ONS studies.

A combined dataset was created where data on election results, housing stock, economy, deprivation, social mobility, education, religion, prevalence of health conditions, age distribution and ethnic group breakdown were put alongside the Parliamentary Constituency name. This was ordered from A-Z.

Most of this data was in the form of absolute numbers which given the differences in constituency size, is not a good comparative metric. Thus, additional data was "engineered" including percentage results for political parties and voter turnout; the ratio of median house price to median salary; and number of businesses per capita: preventing dataset imbalance.

It was found that data for some metrics including social mobility index, median house price and school funding was only available for England: not the rest of the UK. Additionally, the major UK political parties do not operate in Northern Ireland. Thus, non-English constituencies were removed from the dataset.

In order to mitigate overfitting issues the dataset was randomly split into training, testing and validation sets. This was done to make our models usable on other future datasets including the 2021 Census data once it releases.

4. Local Politics - Maximilian Matthews

4.1. Aim

Using the UK Parliamentary Constituency dataset, I aim to predict whether a constituency could vote

Figure 2: Heatmap showing the correlation of the data-set's features with Binary Con (whether the Conservative Party win the constituency, 1 = win)

for the Conservative Party or another political party.

4.2. Data Preparation

4.2.1. Feature Engineering

Initially the predictor, Conservative vote proportion was in the form of continuous data (0-1). Since this is a classification problem this was converted into a binary data (BIN_CON).

If the Conservative Party on the constituency this was set to 1, else it is equal to 0.

4.2.2. Attribute Removal

In order to reduce the number of useless features the forward selection algorithm has to test, certain features were removed.

- *Constituency Name.* This is not relevant to creating a predictive model, and since it's a series of strings it interrupts with numeric analysis algorithms.

- *Electorate, Number of Businesses.* These two features had already been used to engineer per capita features and are thus no longer useful.

- *Conservative votes, Labour votes, Lib Dem votes and vote proportions.* Voting data is directly affected by the size of our predictor (if Conservatives do well, Labour and Lib Dem automatically do more poorly), thus they are too dependant.

4.2.3. Data Splitting

To prevent overfitting, the dataset was split at random into training, validation and testing subgroups. This was done at a ratio of 60:20:20. Initially an 80:10:10 ratio was tested however this resulted in an excessive gap (0.35) between the training and validation groups).

4.2.4. Data Balancing

The Conservative Party came first in English Constituencies 63% of the time causing an imbalance in the training data which would have caused poor results in the test data.

Thus the majority class (BIN_CON = 1) was undersampled at random and assigned to a new dataset variable. This was only done for the training data since validation and test should be an accurate depiction of reality.

4.3. Dataset Overview

In order to obtain a first understanding of the relationships between BIN_CON and other features, a correlation heatmap was generated (Figure 2. In

```
            coefficient    std  p-value  [0.025   0.975]
intercept         0.307  0.377    0.415  -0.436     1.05
20-29            -2.172  3.005    0.470  -8.093     3.75
Mixed            -0.769  8.438    0.927 -17.397    15.86
-------------------------------------------------------------
Confusion Matrix (total:226)      Accuracy:          0.721
  TP: 102 | FN: 11
  FP: 52 | TN: 61
```

Figure 3: Confusion matrix using features selected by the automatic forwards selection algorithm (training data).

```
            coefficient     std  p-value   [0.025   0.975]
intercept         0.307   0.586    0.600   -0.855    1.469
20-29            -2.172   5.137    0.672  -12.357    8.014
Mixed            -0.769  14.204    0.957  -28.933   27.396
-------------------------------------------------------------
Confusion Matrix (total:107)      Accuracy:          0.804
  TP: 63 | FN: 7
  FP: 14 | TN: 23
```

Figure 4: Final model on test data.

addition to this a correlation metric was obtained for each feature (with BIN_CON).

Table 1: Correlation between features and BIN_CON.

Feature	Correlation
20 – 29	-0.5095
Unemployment Rate	−0.4164
Are Christian (proportion)	0.3617
50 – 59	0.5415
Home Ownership (proportion of households)	0.5625

Correlation numbers in Table 1 shows both negative and positive correlation between features. These confirm expectations that area with older, white, more affluent residents tend to vote Conservative.

4.4. Logistic Regression

From the initial correlation analysis it was obvious that there were moderately strong correlations between multiple features and BIN_CON. Since this is present and this is a categorisation problem, logistic regression was identified as the most suitable model. This is where a sigmoid function is fitted between two binaries.

4.5. Feature Selection

4.5.1. Forward Selection Algorithm

In order to select the best features for the model, without adding too many and risking overfitting, a forward selection algorithm was used.

Initially a model with no features was created. In an iterative process the feature with the next strongest correlation to BIN_CON was identified and added to the model. If the addition of this feature improved the validation accuracy of the model by 0.005 (5%) then it would be included, else it would be rejected. This was repeated until no more features were added, which occurred after 2-3 variables typically.

Compared to selecting the features manually, this produces a more accurate model and the 0.005 inclusion criteria prevents overfitting.

After iterating through all columns, the forwards selection algorithm selected features listed in Figure 3. Despite only two features being used, adding additional ones would have improved accuracy by ¡2% risking overfitting. Although the p-values are fairly high the features selected scored well in the prior correlation test: -0.5095 and -0.4458 respectively.

4.6. Testing

A final logistic regression model was created using the selected features and this was trialed using the test data. See Figure 4 and Table 2.

Table 2: Performance metrics from running test data on model

Performance Matrix	Value
Accuracy	0.804
Precision	0.818
Recall	0.9
Δ Accuracy (Val - Test)	0.065

4.7. Discussion

4.7.1. Successes

The performance metrics obtained via the testing dataset show that, with both a high precision and accuracy, my model has high predictive power in determining whether an area would vote for the Conservative party.

Additionally through data engineering, splitting, attribute removal and balancing, as well as a thorough feature selection process, the model does not suffer from overfitting. By obtaining a high accuracy with only two features, a Δ Accuracy of only 0.065 was obtained.

Both the age distribution as well as ethic group distribution, which the model is based on, are provided regularly via the census meaning this model is repeatable in the future.

4.7.2. Limitations

Since this model was only trained on English Constituencies geography is a key limitation. Due to differences in which political parties operate there and the political circumstances, this model could not be extrapolated onto Scotland, Wales or Northern Ireland.

Furthermore, there have been multiple political realignments in the last decade. This includes more affluent, older traditionally Conservative areas becoming more open to voting for other parties. Thus a model based on demographics could lose its predict power very quickly.

5. Local Unemployment - William Jiang

5.1. Aim

Predicting whether a constituency has a low unemployment rate or not.

5.2. Initial Dataset Overview

The predictive model was built upon the dataset extracted from 2021 census dataset. The dataset consists of 533 constituencies with 55 features (1 ID, 1 categorical, 53 continuous).

5.3. Predictor Variable

The unemployment rate, a widely recognized key indicator of the economic performance of the labor market of a specific region, was chosen to be the predictor variable. The objective of the predictive model was to predict whether a given constituency would have a low unemployment rate at the present time. A low unemployment rate would be considered desirable as a result of more job opportunities available and higher economic output of the constituency. The unemployment rate provided within the database was measured on a scale of 0-100 (0: total employment; 100: total unemployment). See Figure 5.

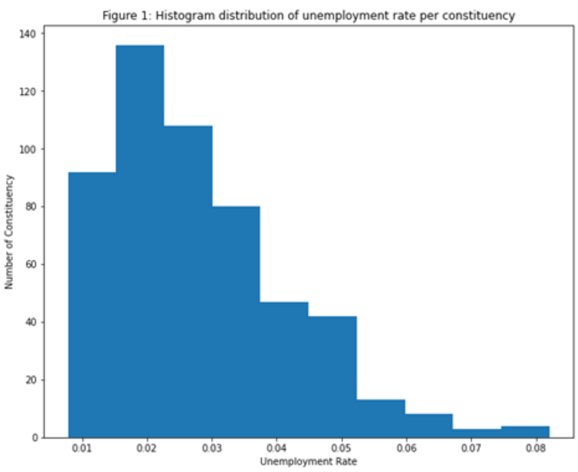

Figure 5: Histogram distribution of unemployment rate per constituency.

5.4. Dataset Preparation

5.4.1. Feature Engineering

As the predictor variable was in the form of continuous data ranging from 0-100, a threshold value was necessary to binarize the predictor attribute.

The mean unemployment rate was calculated to equal 0.028256. Constituencies with mean unemployment rate or below (<= 0.0028256) were assigned a value of 1 and those above (> 0.0028256) were assigned a value of 0. A new column. "UR_B", was used to record the binary values. By using mean as a threshold value, imbalance of dataset with respect to y classes could be avoided.

5.4.2. Attribute Removal

The following attributes were removed for the models:

- *Constituency Name.* Served as id and was a non-predictive attribute.
- *Conservative votes, Labour votes, Lib Dem votes.* Proportion of voters for each party in the dataset was considered to be a better representation of political preference of a constituency.
- *Religion.* This attribute was not considered to have causal relationship with the predictor variable.
- *Unemployment rate.* This column was no longer useful after binarization, and was replaced by "UR_B".

5.4.3. Final Pre-processing

The updated dataset was split into training, validation and test (60%, 20%, 20%). The purpose of training set was to train the model, with the validation set for accuracy, recall, and precision measurement. Finally, the test set would be used as a final measure to check the accuracy was obtained to prevent overfitting of data.

All dataset was standardized and rescaled to have the means of 0 and the standard deviation of 1 for requirement of model construction.

5.4.4. Context for Performance Metrics

Three metrics were used to compare the performance of the predictive models:

- *Accuracy.* The proportion of constituencies that have been correctly predicted with high or low unemployment rate.
- *Precision.* The proportion of constituencies that have been predicted with low unemployment rate were correct.

- *Recall.* The proportion of actual constituencies with low unemployment rate were correctly predicted by the model to be with low unemployment rate.

After analyzing the different cost of mistakes, predictive models with high recall rate were prioritized. This was due to False Positive predictions were considered to be the most undesirable outcome among all. Imagine a person chose to move to a constituency to work based on our prediction of the constituency is low in unemployment rate, but in fact the constituency is high in unemployment and is very hard to find jobs.

5.5. Predictive Models

5.5.1. Logistic Regression with LASSO

Even after attribute removal, there are still more than 40 available features. And hence, it was vital to first identify which features have more significant correlation with the predictor variable. The LASSO method was then implemented to select the "best" variables to use for a logistic regression model using the principled way. See Figures 6, 7, 8.

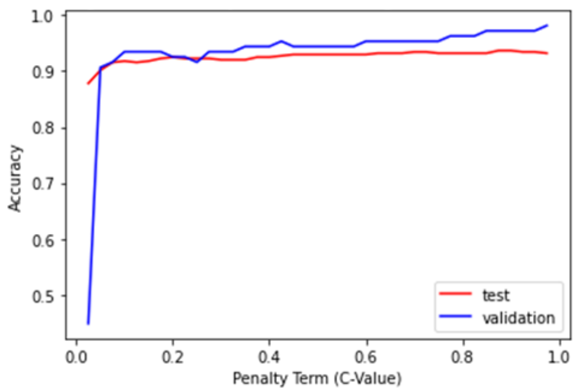

Figure 6: Accuracy Score of training and validation set.

According to these Figures, the penalty term (C-Value) peaked at 0.1 for both accuracy and recall. And as mentioned before, the performance metrics of recall to minimize false positive was prioritize, so the value of 0.1 of C was identified to be the optimum value.

Table 3: Summary of metrics of performance for logistic regression with LASSO at C=0.1.

Type of dataset	Accuracy	Precision	Recall
Accuracy	0.918	0.922	0.941
Precision	0.916	0.931	0.915
Recall	0.935	0.933	0.949

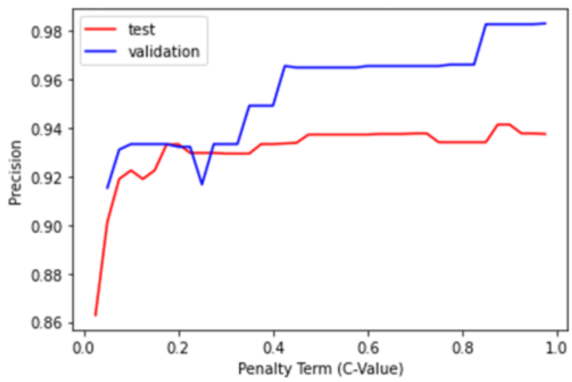

Figure 7: Precision Score of training and validation set.

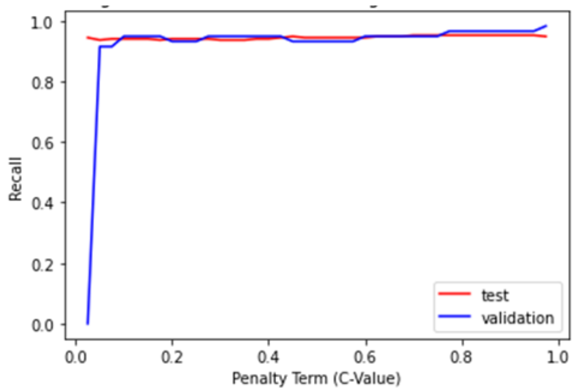

Figure 8: Recall Score of training and validation set.

5 features were selected to have relatively significant correlationship with predictor variable with C = 0.1, as seen in table 2. The 5 features would be used for further decision tree predictive model construction.

5.5.2. Decision Tree

Decision Tree was employed to construct a predictive model built on Gini Impurity, an useful measurement of the probability of a randomly assigned datapoint is incorrect. Graphs with Recall plotted against the Maximum Depth and Minimum impurity was constructed for better identification of the optimal value for both depth and impurity, and to

Table 4: Summary of chosen features with LASSO at C=0.1.

Features	Coefficients
Turnout (proportion)	0.303
Share of LSOAs (small areas)	-1.092
Obesity	-0.026
Schizophrenia, bipolar, psychoses	-0.336
Age group (0-9)	-0.048

prevent overfitting of the model. See Figure 9.

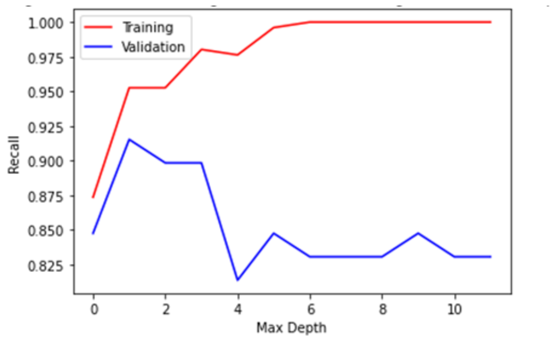

Figure 9: Recall of Training and Validation Data against Max Tree Depth.

Figure 10 shows the validation recall peaked at the range of 1-3 and drops significantly after the depth of 3. This notable drop in recall on validation data demonstrates the overfitting of the training data.

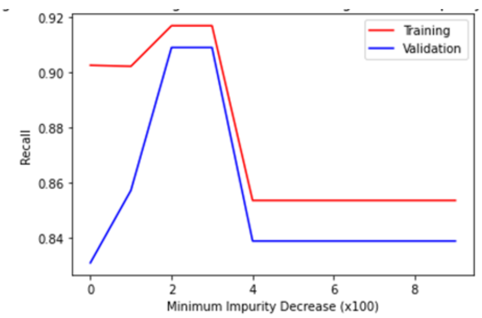

Figure 10: Recall of Training and Validation Data against Min Impurity Decrease.

Figure 11 shows the recall peaked at a minimum impurity decrease of 0.02. To improve the predictive nature of our model, the minimum impurity decrease of 0.02 was chosen to maximise recall rate while a max tree depth of 3 was used to prevent overfitting.

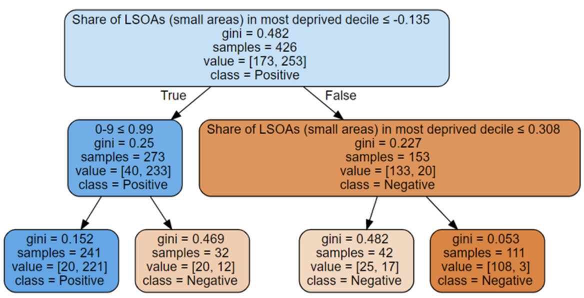

Figure 11: Decision Tree of the Final Model (max depth = 3, minimum impurity increase = 0.02

5.6. Discussion & Conclusion

5.6.1. Results

The two chosen machine learning models were tested against the test dataset as shown in Table 5.

Table 5: Summary of metrics of performance for logistic regression with LASSO at C=0.1.

Performance Metric	Logistic Regression (LASSO)	Decision Tree
Accuracy	0.935	0.925
Precision	0.933	0.947
Recall	0.949	0.915
Δ Recall	+0.034	+0.068

The results showed both chosen models achieve considerably good predictive performance, with accuracy of 92.5% or higher during final testing. The results also suggested both models were not overfitting owing to the considerably insignificant difference between validation and test sets (<=0.068).

The Logistic Regression with LASSO was selected as the final predictive model to identify whether a constituency is with low unemployment or not due to its exceptional recall rate of 94.9%. Despite its lower precision than the decision tree model, the ultimate aim of this model is to minimize False Positive predictive outcome as explained in section 3.4. And hence, with higher accuracy and recall, the logistic regression with LASSO was considered to be a better and more suitable predictive model.

5.6.2. Evaluation

Limitation of the predictive models of this study include not using the most up-to-date data. The models were constructed on the 2021 census dataset of the UK Parliament[2], and hence, the results may differ due to the time accuracy of the data. Other limitations include not considering factors such as number of business, economic output, cost of living that may also have a sizeable impact on unemployment rate of a constituency. The predictive models in this study are solely built upon the available datasets from the common library of UK Parliament.

6. Brexit Voting Patterns - Owain Pill

6.1. Aim

Predicting whether a constituency voted for Brexit based on data about it.

6.2. Data Preparation

6.2.1. Dataset Observations

From Figure 12, the data looks to be slightly negatively distributed with a spike in values around 0.55.

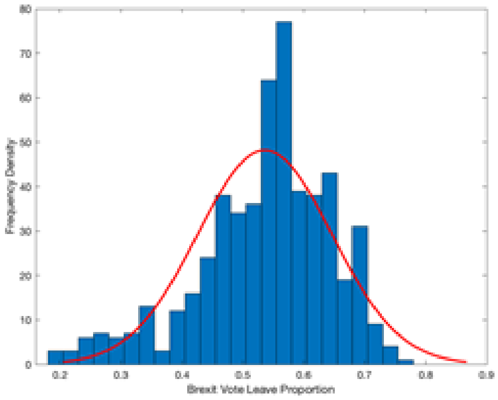

Figure 12: Histogram of Brexit voting data with normal distribution fitted.

6.2.2. Linear Regression

The R^2 value was used to see correlation between Brexit votes and individual features. The 5 features with the highest correlation are shown in Table A.9.

Table 6: R^2 between features and Brexit voting ratios.

Feature	Correlation
Brexit	1.000
Brexit Predicted	0.9919
CON%Lev_4_Qual	0.7681
CON%PROF	0.7655
CON%Lev_1_Qual	0.719

The most closely correlated feature that gives an informative result is *CON%Lev_4_qual*, a graph of which is shown in Figure 13.

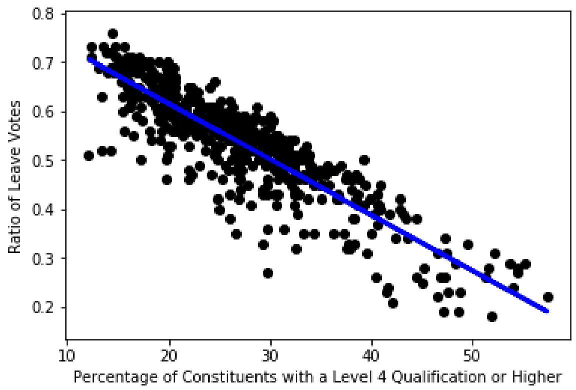

Figure 13: Relationship between Leave vote ratio and Level 4 Qualifications

6.2.3. Feature Engineering

The dependent variable (Brexit) was converted from a ratio to a binary value (1 or 0).

The threshold for this conversion was decided to be 0.5 as this would decide whether a constituency as a whole contributed to voting to leave the EU. These values were assigned to a new column 'Brexit binary'.

As the median of Brexit voting ratios was 0.55, there would be fewer constituencies that voted to remain. This number was found to be 172 vs 394 constituencies that voted to leave. The larger group was randomly undersampled so that both groups would match thereby avoiding dataset imbalance. Random selection with no variable should lead to 0.50 accuracy.

6.2.4. Attribute Removal

After binarization, a number of columns were removed from the dataset to make predictions better.

- *Constituency Name.* This was just the id for each row.
- *Conservative votes, Labour votes, Lib Dem votes.* Proportional figures for this data was included.
- *Brexit, Brexit Predicted.* This data had been binarised so needed to be removed.

6.2.5. Data Splitting

This updated dataset was then split into training, validation, and test. The split was 50%, 25%, 25%. This larger validation set helped minimize overfitting to the validation set in forward selection.

6.2.6. Performance Metrics

To measure the performance of the datasets, the cost of different sorts of mistakes were analyzed.

Considering that the data was nearly normally distributed and centered around 0.55 which is close to the threshold value and that the penalty for either false negatives or false positives is not fatal or costly, accuracy was chosen as the performance metric. This prioritizes getting the greatest proportion of correct predictions on a scale from 0 to 1.

6.3. Logistic Regression

Logistic regression was chosen as the method because it is useful for finding the correlation between binary dependent variables and many independent variables. To evaluate the effectiveness of any model that would be created, baseline values for accuracy were needed given different conditions.

A graph of accuracy vs number of features included in the model was created. Each iteration had completely different variables so there was a lot of noise in the signal. The training data accuracy (0.90 overall average) and validation data accuracy (0.84 overall average) were well above randomly picking (0.50). The average accuracy of training and validation data is achieved after less than 10 features.

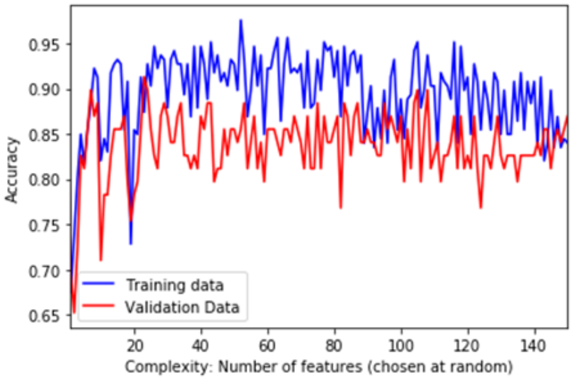

Figure 14: Accuracy vs random features

The same graphing method was used with an ordered list of the R^2 values from highest to lowest obtained from the linear regression. The results showed that there was an initial performance increase when there were fewer variables as these were highly correlated and that adding more variables did not improve either the training or validation accuracies beyond 30 variables as shown in Figures 15 and 16.

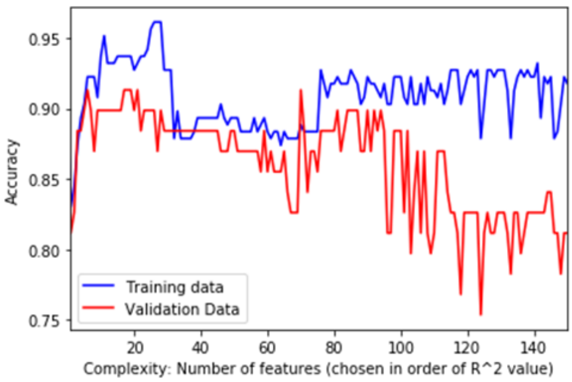

Figure 15: Accuracy vs features by order of R^2

A 'Forward Selection' algorithm was created to automatically pick a specified amount of the best features based on validation accuracy. Every cycle, the variable with the highest accuracy was added. This meant that accuracy could stay the same or drop after a cycle. The accuracy was for train and validation was higher (0.93 average and 0.92 average respectively, Figure 17) than for the previous two models so was taken forward.

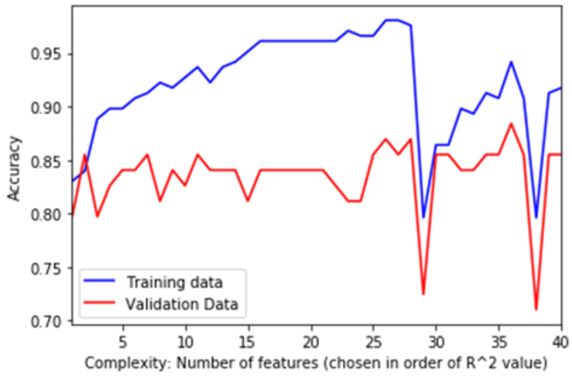

Figure 16: Accuracy vs features by order of R^2

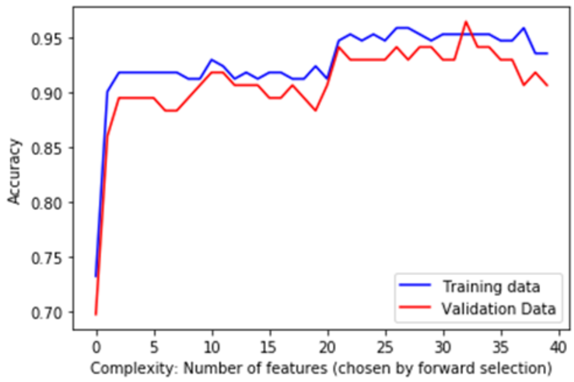

Figure 17: Accuracy vs features by forward selection

The model had many variables with high p-values which meant there was a lot of prediction overlap between variables. To mitigate this, the data was normalized and the Lasso method (penalty = 'l1', C = 0.1) was used. This removed 33 variables causing a training accuracy drop from 0.948 to 0.89 and a rise in validation accuracy from 0.90 to 0.93.

A final measure was to manually remove features that were clearly related to each other. An example of this was removing 'Labour Vote (proportion)' as 'Conservative Vote (proportion)' was already included in the model. After each removal, the models' accuracies were checked to prevent underfitting. The final model is shown below in Figure 18 and Table A.9.

This model was then run without the data being undersampled and achieved a test accuracy of 0.926 which confirmed its robustness.

6.4. Discussion

The model selected provides very good predictive power of whether a constituency will vote for Brexit

```
                                     coefficient     std  p-value  [0.025  \
intercept                                  2.184   2.505    0.383  -2.761
CON%Lev_4_qual                            -0.521   0.103    0.000  -0.724
Conservative Vote (proportion)            15.518   4.133    0.000   7.359
High blood pressure (hypertension)        53.742  15.575    0.001  22.998

                                     0.975]
intercept                             7.130
CON%Lev_4_qual                       -0.318
Conservative Vote (proportion)       23.677
High blood pressure (hypertension)   84.487
--------------------------------------------------------------
Confusion Matrix (total:172)     Accuracy:        0.901
  TP: 74 | FN: 10
  FP: 7 | TN: 81
```

Figure 18: Features, pvals and Confusion Matrix for final model

Table 7: Accuracy of final model

Performance Metric	Value
Training Accuracy	0.901
Validation Accuracy	0.907
Test Accuracy	0.907

or not if there were to be a second referendum (accuracy > 0.9). The fact that it is only made up of three features means that it is less prone to overfitting. This is reinforced by the fact that train, validation, and test accuracies are all very similar. Blood pressure might seem an unlikely feature but probably has to do with providing a good age range as older people are far more likely to suffer from high blood pressure (see Seaborne plot) – other age ranges are present in the data but only in increments of 10 years.

A limitation is that while the results from the final model seem very high, the dataset lends itself to high accuracies as demonstrated by the fact that randomly picking more than 10 variables leads to accuracies of over 80% so the bar for performance is not 0.5.

The data is from different years – some from the 2011 census while other data is from a 2019 constituency review. This will lead to discrepancies with newer data as society changes over time. The dataset is also small which led to there being differences in final results after every time running the code. This was offset by the low number of features in the model and after running the entire code 10 times, the average final undersampled test accuracy was 0.91 and for the whole dataset, the accuracy was 0.93.

In terms of deciding where to live, we can only predict constituency level support. However most people will live in much smaller communities so it difficult to generalise from our results to a granular level. Social policy is also largely made at the federal level in the UK so political leanings of an area matter less than the economic situation when it comes to quality of life. When compared to somewhere like the USA where states have a lot more control over an individual's life (for example trigger bans on abortion), it can be seen that knowing the political leanings of an area can be vital when deciding where to move.

7. Number of Businesses - Chinene Chukwuma

7.1. The Dependent Variable

The number of businesses per capita (NBC) is an important factor to consider when looking for an ideal place to live. A higher number of businesses per capita mean that, proportionally, there are more job opportunities per person.

7.2. Data Processing

7.2.1. Dataset Choice

There are many factors to consider when selecting a dataset as the characteristics of the dataset can determine whether the model is overfitted. Overfitting is displayed when the accuracy of the training set is too high. This means the model will not work for other concepts and unseen datasets so scores poorly on the test data. The 2011 census was chosen with these considerations in mind.

7.2.2. Clean Up

The following variables present in the dataset were removed as, though there may be a correlation with the number of businesses per capita, they were deemed to not have a causal relationship:

1. Election Results
2. Health Conditions
3. Religion
4. Ethnic Groups

A brief exploratory analysis of the dataset was conducted with respect to the dependent variable. Analysis revealed there to be an anomaly in the data, the 'Cities of London and Westminster', which had a value of 1.4 while other constituencies had NBC values below 0.4. See Figures 19 and 20. This was removed to not skew the data and negatively affect the accuracy of the prediction. Other studies on the number of businesses per capita were noted to have taken the same course of action [1].

7.2.3. Binarisation

Originally, the NBC was in continuous form but was changed as a linear regression model to predict future values was not the purpose of this report. Consequently, the mean number of businesses per capita, 0.0570 (3 s.f.), was used to binarise the predictor attribute.

If the number of businesses per capita for a constituency was within the range of 0 ¡ x ¡= 0.0570

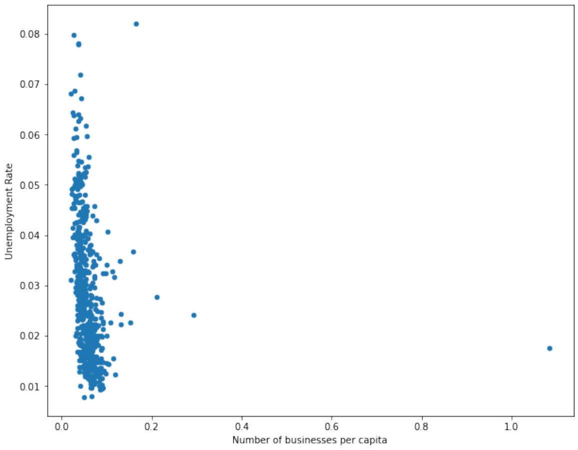

Figure 19: NBC (before data cleanup)

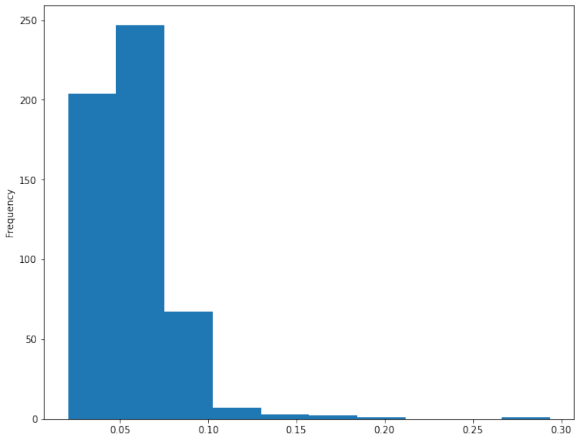

Figure 20: NBC histogram (before data cleanup)

(3 s.f.), it was classified as 0 (low NBC), otherwise they were classified as 1 (high NBC).

In addition to being a balanced dataset, using the mean to binarise the NBC mitigates instances of model bias as everything is balanced. This means accuracy was a suitable and effective metric to assess the competency of a model. See Figure 21

7.2.4. Predictive Model

Due to the chosen predictive model and its coding method, changes were made to the dataset including the removal of null values. The dataset was also subject to label encoding. This ensured that no errors would occur when running vital code.

The dataset was also split up into 80% training data, 10% validation data, and 10% test data.

7.2.5. Performance Metrics

The three metrics used the asses the chosen predictive model included:

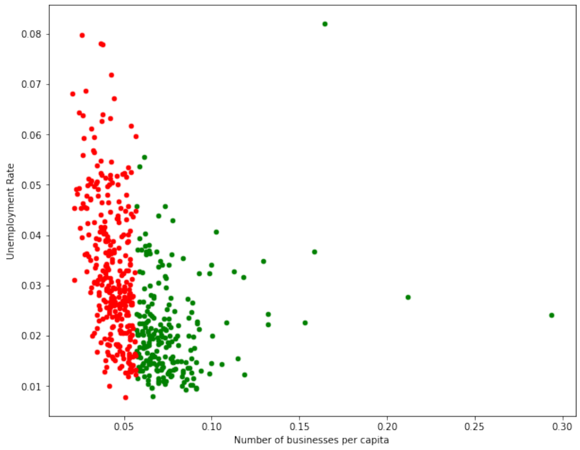

Figure 21: NBC to unemployment post cleanup and binarised

1. Accuracy: The proportion of constituencies correctly predicted to have a high NBC.
2. Precision: Out of all the constituencies predicted to have a high NBC, what percentage was correct.
3. Recall: The proportion of high NBC constituencies that were predicted to have a high NBC.

7.3. Method Selection

7.3.1. Comparison

Classification methods such as the SVM classifier were not considered as the parameters for what counts as a high NBC and low NBC were clearly defined.

Decision trees and random forests find the variable with the minimum weighted impurity level. Random Forests use many randomly generated decision trees to produce the same variable. They have greater accuracy than a single decision tree that is prone to overfitting. The problem with this method is that the final model only considers one variable and is most effective when used with datasets with multiclass classifications.

Forward and backward selection, though effective at finding a combination of features with high predictive power, are generally incapable of finding the global maximum. On the other hand, they are more realistic as they take into account the effect of different combinations of variables.

7.3.2. Justification

Choosing whether to live somewhere is binary in nature so logistic regression was deemed most suitable. The logistic distribution restricts the estimated probabilities (y) between 0 (low NBC) and 1 (high NBC).

The chosen predictive model was backward selection. This method was chosen instead of forward selection as there aren't many features to consider. Considering as many features as possible provides a holistic view of the causes and effects of different variables on the NBC.

7.4. Logistic Regression Backward Selection

7.4.1. Description

Logistic Regression Backward Selection first starts by considering all the features in a dataset, then calculating the accuracy before removing the feature with the least predictive power. This process is repeated until there are no more variables to remove or the validation accuracy has decreased by more than a certain amount.

7.4.2. Model Tuning

At first, the accuracy of the model was 0.96 (3 s.f.) so to avoid overfitting, the 'Number of Businesses' variable was removed as it was too closely related to the NBC, increasing the accuracy. The accuracy was reduced to a maximum of 0.814 (3 s.f.). See Figures 22 and 23

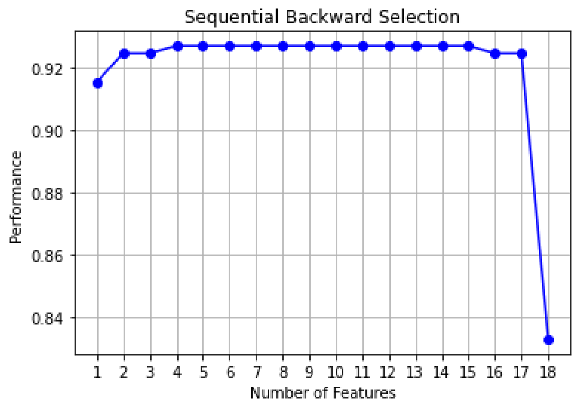

Figure 22: Backwards selection with number of businesses

7.4.3. Adjusting Hyperparameters

The hyperparameter called *k_feature*, which, in this case, is the minimum number of features your model will select, was defined as 'best' to find the best combination of features. Changing this hyperparameter only resulted in a loss of information.

The greatest change was visible when the cross-validation hyperparameter (cv) was increased from 0 to 5. CV determines the splitting strategy of the model [2] so results in more averaged values, reducing the risk of overfitting. See Figure 24.

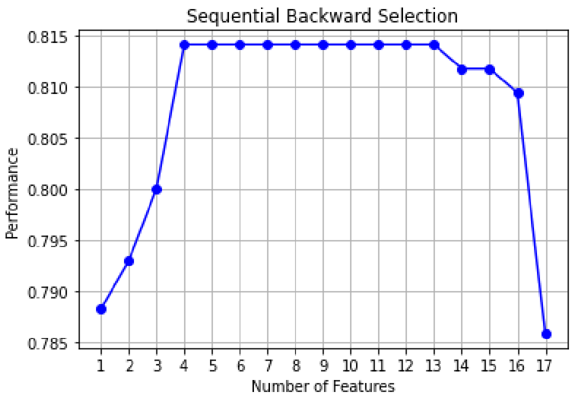

Figure 23: Backwards selection without number of businesses

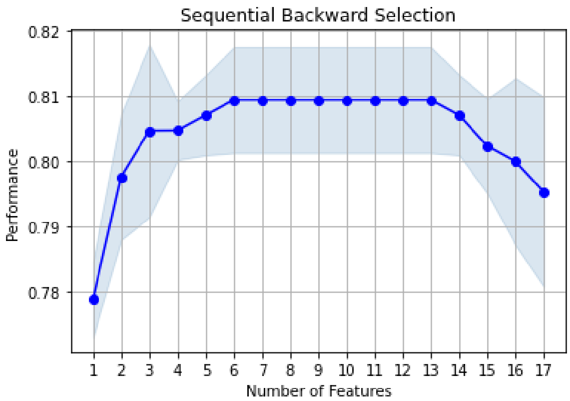

Figure 24: Backwards selection with increased cross-validation

7.5. Results Summary

7.6. Discussion

Nine out of seventeen features were determined to be the 'best' feature combination with the most predictive power. These features included: Unemployment Rate, Social Mobility Index, and Home Ownership. Potential observations one could make from this include that areas with a high number of businesses per capita can be closely associated with a high social mobility index, making them ideal places to live long-term.

In terms of the evaluation metrics, though the recall values are much lower in comparison to the

Table 8: Final model results summary

Performance Metric	Training	Validation	Test
Accuracy	80.941	75.472	83.333
Validation Accuracy	80.941	75.472	88.889
Test Accuracy	40.235	50.943	46.296

```
    coefficient     std  p-value  [0.025  0.975]
0        -5.199   2.429    0.032  -9.973  -0.424
1         0.638   0.089    0.000   0.463   0.813
2        -0.082   1.855    0.965  -3.728   3.564
3        -0.376  22.974    0.987 -45.533  44.781
4        -0.805   2.574    0.754  -5.865   4.255
5         0.006   0.004    0.111  -0.001   0.014
6        -0.002   0.001    0.170  -0.004   0.001
7        -0.297  10.779    0.978 -21.485  20.890
8        -0.045  17.557    0.998 -34.556  34.466
------------------------------------------------------------
Confusion Matrix (total:425)      Accuracy:         0.809
  TP: 125 | FN: 46
  FP: 35 | TN: 219
```

Figure 25: Final model confusion matrix

accuracy and precision values, there is still an increase after training data results for all values. This proves that the selected feature subset improves the performance of the model in all aspects. Lower recall values imply that proportionally, it is a model with lower false positives (35). For this study, a lower number of false positives is better so one can be sure there is a higher chance of a constituency identified as ideal, actually being ideal.

Overall, this is a good model that is not overfitted and has a high chance of correctly predicting whether an area has a high or low number of businesses per capita.

8. Conclusion

Combined, the team has created a series of models that accurately predict the major characteristics of a constituency: political leanings, unemployment, and the number of businesses per capita. Logistic regression was the primary base used throughout as choosing whether or not to live in a constituency is a binary decision.

The study prioritised having a lower number of false positives so constituencies identified by the models as ideal, had a higher likelihood of being ideal in reality.

A logistic regression model was built to predict whether the Conservative Party would win the next election in an area. By using an optimised automatic forward selection method, a model obtained where predictions could be made with 80.4% accuracy while only using two features. This avoids the risk of overfitting meaning the model is widely applicable.

While the model to predict voting tendencies in a constituency used forward selection, the Brexit votes predictive model applied forward selection. The model selected provides very good predictive power of whether a constituency will vote for Brexit or not if there were to be a second referendum (accuracy ¿ 0.9). The fact that it is only made up of three features means that it is less prone to overfitting. After many iterations, due to a high infection in data, the average final test accuracy was 91%.

LASSO was used to predict unemployment. The selection model carried out effective features selection with a large penalty term applied until 5 features remained. The selected features were considered to have a significant correlation with the predictor variable (unemployment rate) and were used for Decision Tree model for further predictive and comparison purposes. The model has a high training accuracy of 93.5%.

On the other hand, the model for the number of businesses per capita utilised Backward Selection so all features were considered before the feature combination with the most predictive power was selected. Using backward selection means you are more likely to end up with a local maximum that contains a longer list of features. This provides a more holistic view of the causes and effects of different variables on the NBC, resulting in a model with 83.3% accuracy.

In conclusion, the predictive models classified constituents as ideal and unideal effectively. In practice, they would be combined to create an algorithm where individuals have the power to define whether more or less of a feature is ideal for them and returns a list of constituents.

Appendix A. Number of Businesses - Chinene Chukwuma

Appendix A.1. Predictive Methods Comparison

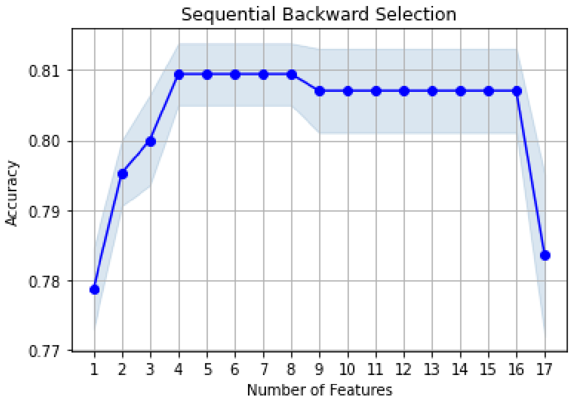

Figure A.26: Training accuracy of backwards selection

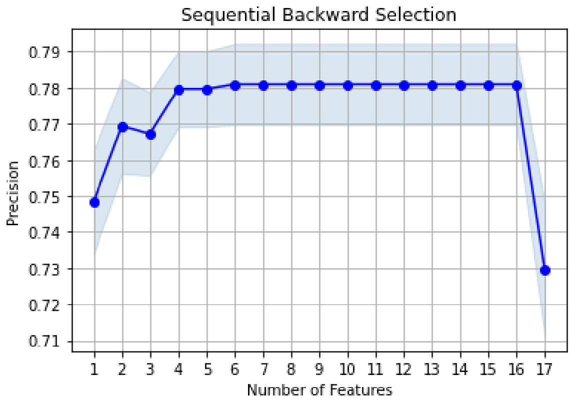

Figure A.27: Training precision of backwards selection

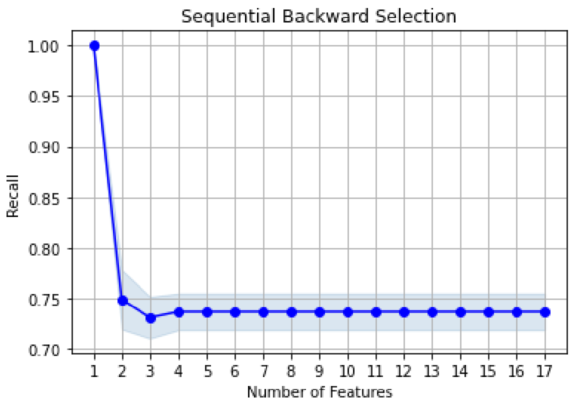

Figure A.28: Training recall of backwards selection

	feature_idx	avg_score
17	(0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13,...	0.783529
16	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	0.807059
15	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14...	0.807059
14	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14)	0.807059
13	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13)	0.807059
12	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12)	0.807059
11	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11)	0.807059
10	(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)	0.807059
9	(1, 2, 3, 4, 5, 6, 7, 8, 9)	0.807059
8	(1, 2, 3, 4, 5, 6, 8, 9)	0.809412
7	(1, 2, 3, 4, 5, 6, 8)	0.809412
6	(1, 2, 3, 4, 5, 6)	0.809412
5	(1, 2, 4, 5, 6)	0.809412
4	(1, 4, 5, 6)	0.809412
3	(1, 4, 5)	0.8
2	(1, 5)	0.795294
1	(1,)	0.778824

Figure A.29: Backward selection process

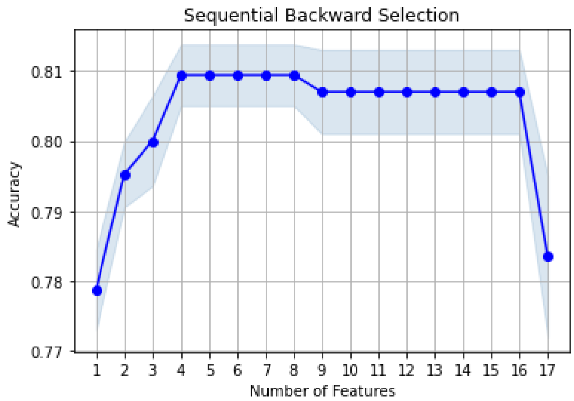

Figure A.30: Features with the most predictive power

Table A.9: Accuracy of final model

Method	Outcome	Advantages	Disadvantages
LR Forward Selection	Finds the combination of features with the most predictive power working forwards.	More efficient than backward selection when there is a large number of data variables [3].	Does not result in the global maximum.
LR Backward Selection	Finds the combination of features with the most predictive power working backward.	More effective than forward selection for a small number of variables [3].	Finds the local optimum combination, not the best combination.
LASSO	It helps selects the best variables for predictive analysis.	Useful if there are several insignificant variables.	Does not show how variables behave when combined.
Decision Tree	The variable with the minimum weighted impurity level.	The resulting variable is likely to give accurate results.	Prone to overfitting[4]. Cannot guarantee decision tree is optimum.
Random Forest	Out of many decision trees, the variable with the minimum weighted impurity level.	Takes many randomly generated decision trees into account. Greater accuracy than a single decision tree	Harder to interpret than decision trees and cannot view the impurity level of individual variables[4].
SVM Classifier	Finds the best separation between classes.	Works well when there is a clear distinction between classes.	Will be harder to execute if there is a lot of overlapping between classes [5].

Appendix B. Code

Appendix A - Table Feature Descriptions

Column	Description	Type	Instance
Constituency	Name of constituency	String	Aldershot
Electorate	Number of people in constituency	int	72617
Conservative Votes	-	int	27980
Conservative Vote (proportion)	-	float	0.385309
Labour Votes	-	int	11282
Labour Vote (proportion)	-	float	0.155363
Lib Dem Votes	-	int	6920
Lib Dem Vote (proportion)	-	float	0.095294
Turnout (Proportion)	-	float	0.660066
Median House Price	-	int	302000
Ratio of median house price to median salary	-	float	8.9
Home Ownership (proportion of households)	-	float	0.65426
Unemployment Rate	-	float	0.015521
Share of LSOAs (small areas) in most deprived decile	SOAs were designed to improve the reporting of small area statistics and are built up from groups of output areas (OA).	float	0
Number of Businesses	-	int	3825
Number of businesses per capita	-	float	0.052674
Standardised Weighted Overall Social Mobility Index	-	float, not ratio	-24.9463
School Funding Per Pupil (Real)	-	int	6972.00
Are Religous (proportion)	-	float	0.664787
Are Not Religious (proportion)	-	float	0.265362
Are Christian (proportion)	-	float	0.581417
Are Muslim (proportion)	-	float	0.013818

Average Internet Speed (Mb/s)	-	float, not ratio	114.8
Asthma	Proportion of people with certain disease	float	0.057579
Atrial fibrillation	-	float	0.019893
Cancer diagnosis since 2003	-	float	0.03061
Cardiovascular disease (primary prevention)	-	float	0.012002
Chronic kidney disease	-	float	0.034918
COPD	-	float	0.018407
Coronary heart disease	-	float	0.026682
Dementia	-	float	0.006853
Depression	-	float	0.133935
Diabetes	-	float	0.078472
Epilepsy	-	float	0.008037
Heart failure	-	float	0.008008
High blood pressure (hypertension)	-	float	0.147026
Learning disabilities	-	float	0.004241
Obesity	-	float	0.116832
Osteoporosis	-	float	0.013875
Peripheral arterial disease	-	float	0.0054
Rheumatoid arthritis	-	float	0.006285
Schizophrenia, bipolar disorder & psychoses	-	float	0.008153
Stroke and transient ischaemic attack	-	float	0.014896
0-9	-	float	0.129569
10-19	-	float	0.109031
20-29	-	float	0.121766
30-39	-	float	0.151904
40-49	-	float	0.143236
50-59	-	float	0.139938
60-69	-	float	0.091857
70-79	-	float	0.072326
80+	-	float	0.040373
"White"	-	float	0.855334
"Mixed"	-	float	0.021343
"Asian"	-	float	0.097737
"Black"	-	float	0.019293

"Other"	-	float	0.006293
vgh	-	float	50.03175
gh	-	float	35.64885
fh	-	float	10.74363
bh	-	float	2.774196
vbh	-	float	0.801563
CON%1_ps	One person	float	25.09516
CON%1_ps_65	One person over 65	float	9.399179
CON%1_ps_Oth	One person under 65	float	15.69598
CON%1_fm	one family	float	65.79674
CON%1_fm_lp_dc	one family lone parent disparate conditions	float	6.227143
CON%Own	**Owned**	float	65.42605
CON%Own_out	Owned outright	float	24.48563
CON%Own_mort	Owned with a mortgage or loan	float	40.94042
CON%Share	**Shared ownership (part owned & rented)**	float	1.686777
CON%Soc_r	**Social rented**[1]	float	15.26558
CON%Soc_r_LA	Rented from council (Local Authority)	float	1.970394
CON%Soc_r_Other	Other social rented	float	13.29519
CON%Private_rent	**Private rented**	float	16.8006
CON%Private_r_land	Private landlord or letting agency	float	12.25526
CON%Private_r_Oth	Other private rented	float	4.545341
CON%Rent_free	**Living rent free**	float	0.820998
CON%NO	No cars or vans in household	float	15.82535
CON%1CV	1 car or van in household	float	42.29631
CON%2CV	2 cars or vans in household	float	32.03881
CON%3CV	3 cars or vans in household	float	7.269561
CON%4+CV	4 or more cars or vans in household	float	2.569971
CON%No_qual	No qualifications	float	18.51107
CON%Lev_1_qual	Level 1 qualifications	float	16.35943
CON%Lev_2_qual	Level 2 qualifications	float	17.14327
CON%Appren	Apprenticeship[1]	float	3.722924
CON%Lev_3_qual	Level 3 qualifications	float	12.51731
CON%Lev_4_qual	Level 4 qualifications and above	float	24.99247
CON%Oth_qual	Other qualifications[1]	float	6.753519
CON%AC_ALL	**Economically active**	float	77.51202
CON%AC_EMP	Employee: Part-time	float	70.69799
CON%AC_EMP_E_P	Employee: Part-time	float	13.53477
CON%AC_EMP_E_F	Employee: Full-time	float	48.94207

CON%AC_SELF	Self-employed	float	8.221148
CON%AC_UNE	Unemployed	float	3.637
CON%AC_F_STUD	Full-time student	float	3.17703
CON%INA_ALL	**Economically inactive**	float	22.48798
CON%INA_RET	Retired	float	10.32535
CON%INA_STUD	Student (including full-time students)	float	3.718628
CON%INA_CARE	Looking after home or family	float	3.924643
CON%INA_LONG SICK	Long-term sick or disabled	float	2.542142
CON%INA_OTHER	Other	float	1.977222
CON%UN_16_24	Unemployed: Age 16 to 24	float	1.107814
CON%UN_50_74	Unemployed: Age 50 to 74	float	0.725586
CON%UN_NEVER	Unemployed: Never worked	float	0.463857
CON%UN_LONG	Unemployed: Long-term unemployed	float	1.373431
CON%A_FOR_FISH	Agriculture, forestry and fishing	float	0.100864
CON%M_Q	Mining and quarrying	float	0.088477
CON%MANF	Manufacturing	float	7.056908
CON%E_G_S_AIR	Electricity, gas, steam and air conditioning supply	float	0.41938
CON%W_SEW_WAST_REM	Water supply [1]	float	0.72551
CON%CONST	Construction	float	7.479827
CON%WT_RT_REPM	Wholesale and retail trade [2]	float	15.95767
CON%TRAN_STOR	Transport and storage	float	5.075028
CON%ACCOM_FOOD	Accommodation and food service activities	float	5.280294
CON%INF_COM	Information and communication	float	6.400411
CON%FIN_INS	Financial and insurance activities	float	3.6594
CON%REAL	Real estate activities	float	0.998018
CON%PROF_SCI_TECH	Professional, scientific and technical activities	float	6.057121
CON%ADMIN_SUPP	Administrative and support service activities	float	6.70654
CON%ADMIN_DEF	Public administration and defence [3]	float	11.10561
CON%EDUC	Education	float	7.309952
CON%HEAL_SOC	Human health and social work activities	float	11.0012
CON%OTHER	Other	float	4.577789
CON%MAN_DIR_SEN	Managers, directors and senior officials	float	10.07397
CON%PROF	Professional occupations	float	14.92072
CON%ASSOC_PROF_TECH	Associate professional and technical occupations	float	15.92759
CON%ADMIN_SEC	Administrative and secretarial occupations	float	11.85766
CON%SKILL	Skilled trades occupations	float	11.33211

CON%CAR_LEI_	Caring, leisure and other service occupations	float	9.819153
CON%SAL_SERV	Sales and customer service occupations	float	8.642412
CON%OPS	Process, plant and machine operatives	float	6.207531
CON%ELEM	Elementary occupations	float	11.21886
CON%H_MAN_AD_PROF	Higher managerial, administrative and professional	float	10.71924
CON%LG_E_H_MAN_AD	Higher managerial and administrative [1]	float	2.707991
CON%H_PROF	Higher professional occupations	float	8.011247
CON%L_MAN_AD_PROF	Lower managerial, administrative and professional	float	22.50612
CON%I_OCC	Intermediate	float	15.4796
CON%S_EMP_OWN	Small employers and own account workers	float	8.054004
CON%L_SUP_TECH	Lower supervisory and technical	float	8.086397
CON%SEMI	Semi-routine occupations	float	14.53634
CON%ROUT	Routine	float	9.927571
CON%N_WKD_L_EMP	Never worked and long-term unemployed	float	3.999793
CON%N_WKD	Never worked	float	2.626362
CON%L_EMP	Long-term unemployed	float	1.373431
CON%NON_C	Not classified	float	6.690939
CON%FT_STUD	Full-time students	float	6.690939
Brexit	Percentage of Constituents that voted for Brexit	float	0.581
Brexit Predicted	Prediction of how many constituents voted for Brexit	float	0.579

Appendix B – Code

Max Matthews---

```
# %% [markdown]

# # Predicting the probability of a UK Parliamentary constituency voting for the Conservative Party



# %% [markdown]

# ## Environment Setup



# %%

# Import Libraries

from logging import warning
```

```python
from sklearn.linear_model import LogisticRegression as logreg

from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix,
precision_score, recall_score

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import warnings


# Surpress warnings

warnings.filterwarnings("ignore")


# Read csv

dataset = pd.read_csv("combined-dataset-england-only.csv")




# %% [markdown]

# ## Model Summary Class

#

# Python class provided in class which summarises a created sklearn model


# %%

from scipy import stats

import numpy as np

import pandas as pd

from sklearn.metrics import confusion_matrix, accuracy_score


class ModelSummary:

    """ This class extracts a summary of the model


    Methods

    -------
```

```
get_se()
    computes standard error
get_ci(SE_est)
    computes confidence intervals
get_pvals()
    computes p-values
get_summary(name=None)
    prints the summary of the model
"""

def __init__(self, clf, X, y):
    """
    Parameters
    ----------
    clf: class
        the classifier object model
    X: pandas Dataframe
        matrix of predictors
    y: numpy array
        matrix of variable
    """
    self.clf = clf
    self.X = X
    self.y = y
    pass

def get_se(self):
    """Computes the standard error

    Returns
    -------
        numpy array of standard errors
```

```
        """
        # from here https://stats.stackexchange.com/questions/89484/how-to-compute-the-standard-errors-of-a-logistic-regressions-coefficients
        predProbs = self.clf.predict_proba(self.X)
        X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
        V = np.diagflat(np.product(predProbs, axis=1))
        covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
        return np.sqrt(np.diag(covLogit))


    def get_ci(self, SE_est):
        """Computes the confidence interval


        Parameters
        ----------
        SE_est: numpy array
            matrix of standard error estimations


        Returns
        -------
        cis: numpy array
            matrix of confidence intervals
        """
        p = 0.975
        df = len(self.X) - 2
        crit_t_value = stats.t.ppf(p, df)
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        upper = coefs + (crit_t_value * SE_est)
        lower = coefs - (crit_t_value * SE_est)
        cis = np.zeros((len(coefs), 2))
        cis[:,0] = lower
        cis[:,1] = upper
        return cis
```

```
    def get_pvals(self):
        """Computes the p-value

        Returns
        -------
        p: numpy array
            matrix of p-values
        """
        # from here https://stackoverflow.com/questions/25122999/scikit-learn-how-to-check-coefficients-significance
        p = self.clf.predict_proba(self.X)
        n = len(p)
        m = len(self.clf.coef_[0]) + 1
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        se = self.get_se()
        t =  coefs/se
        p = (1 - stats.norm.cdf(abs(t))) * 2
        return p

    def get_summary(self, names=None):
        """Prints the summary of the model

        Parameters
        ----------
        names: list
            list of the names of predictors
        """
        ses = self.get_se()
        cis = self.get_ci(ses)
        lower = cis[:, 0]
        upper = cis[:, 1]
```

```
        pvals = self.get_pvals()
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        data = []
        for i in range(len(coefs)):
            currlist = []
            currlist.append(np.round(coefs[i], 3))
            currlist.append(np.round(ses[i], 3))
            currlist.append(np.round(pvals[i], 3))
            currlist.append(np.round(lower[i], 3))
            currlist.append(np.round(upper[i], 3))
            data.append(currlist)
        cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
        sumdf = pd.DataFrame(columns=cols, data=data)
        if names is not None:
            new_names = ['intercept']*(len(names) + 1)
            new_names[1:] = [i for i in names]
            sumdf.index = new_names
        else:
            try:
                names = list(self.X.columns)
                new_names = ['intercept']*(len(names) + 1)
                new_names[1:] = [i for i in names]
                sumdf.index = new_names
            except:
                pass
        print(sumdf)
        acc = accuracy_score(self.y, self.clf.predict(self.X))
        confmat = confusion_matrix(self.y, self.clf.predict(self.X))
        print('-'*60)
        print('Confusion Matrix (total:{}) \t Accuracy: \t {}'.format(len(self.X),np.round(acc, 3)))
        print('  TP: {} | FN: {}'.format(confmat[1][1],confmat[1][0]))
```

```
        print('  FP: {} | TN: {}'.format(confmat[0][1],confmat[0][0]))


# %% [markdown]
# ## Split the dataset into training, validation, and testing group
#
# In order to prevent overfitting the model must be validated and tested on data which it is not trained on. Hence we randomly split the data into 3 subgroups.


# %%
# Put 60% of the data into a training set and 40% into a combined testing set
dataset_train, dataset_splitdata = train_test_split(
    dataset, test_size = 0.4, random_state = 0
)


# Split the combined testing set into validation and testing sets
dataset_val, dataset_test = train_test_split(
    dataset_splitdata, test_size = 0.5, random_state = 0
)


# %% [markdown]
# ## Balancing the dataset
#
# In order to get a better model we want there to be an equal number of consituencies which voted Tory as those who didn't


# %%
# Counting num of constituencies that voted Tory and for another party
total = len(dataset_train)
num_CON = dataset_train["Binary_Con (1 = win)"].sum()
num_notCON = total - num_CON
```

```
# Adjusting dataset

dataset_voteCON = dataset_train.loc[dataset_train["Binary_Con (1 = win)"] ==
1].sample(num_notCON)

dataset_votenotCON = dataset_train.loc[dataset_train["Binary_Con (1 = win)"]
== 0]


resampled_dataset_train = pd.concat((dataset_voteCON, dataset_votenotCON))


# %% [markdown]

# ## Splitting features and predictors & Data preparation

# `X` and `y` values will be split into sperae variables to make later code
easier.

#

# Consitutuency names were dropped since they are strings and not relevent.
Voting percentages (other than conservative) were not included since they are
directly dependent on the convervative vote %. Absolute numbers were removed
since per capita features were developed.


# %%

X_train = resampled_dataset_train.drop(columns = [

    "Binary_Con (1 = win)", "Constituency Name", "Electorate", "Number of
Businesses", "Conservative Votes",

    "Conservative Vote (proportion)", "Labour Votes", "Labour Vote
(proportion)", "Lib Dem Votes", "Lib Dem Vote (proportion)"

])

y_train = resampled_dataset_train["Binary_Con (1 = win)"].values.reshape(-1,
1)


X_val = dataset_val.drop(columns = [

    "Binary_Con (1 = win)", "Constituency Name", "Electorate", "Number of
Businesses", "Conservative Votes",

    "Conservative Vote (proportion)", "Labour Votes", "Labour Vote
(proportion)", "Lib Dem Votes", "Lib Dem Vote (proportion)"

])

y_val = dataset_val["Binary_Con (1 = win)"].values.reshape(-1, 1)
```

```
X_test = dataset_test.drop(columns = [
    "Binary_Con (1 = win)", "Constituency Name", "Electorate", "Number of Businesses", "Conservative Votes",
    "Conservative Vote (proportion)", "Labour Votes", "Labour Vote (proportion)", "Lib Dem Votes", "Lib Dem Vote (proportion)"
])
y_test = dataset_test["Binary_Con (1 = win)"].values.reshape(-1, 1)


# %% [markdown]
# ## Seaborn Correlation Heatmap
#
# A seaborn correleation heatmap is cretaed to see the correlations between the various features


# %%
seaborn_data = dataset.drop(columns = [
    "Constituency Name", "Electorate", "Number of Businesses", "Conservative Votes",
    "Conservative Vote (proportion)", "Labour Votes", "Labour Vote (proportion)", "Lib Dem Votes", "Lib Dem Vote (proportion)"
])


corrmat = seaborn_data.corr()
heatmap = plt.figure(figsize = (16, 10))


sns.heatmap(corrmat, vmax = 0.8, cmap = "RdYlGn_r")
plt.show()


corrT = dataset.corr(method = "pearson").round(4)
corrT = corrT.sort_values(by = ["Binary_Con (1 = win)"])
corrT["Binary_Con (1 = win)"]


# %% [markdown]
```

```
# # Feature Selection
#
# A logisitcal regrerssion prediction model was established using the training data.
#
# In order to prevent overfitting in the model, an autoatic forwards selection algorithm is used.


# %% [markdown]
# First a function is created to chose the next feature to be added. The feature is only added if it improves the accuracy of the model by a certain criteria


# %%
stopping_criteria = 0.005


def add_feature(X_train, y_train, X_val, y_val, current_features, features_to_test):
    """
    Function which adds a feature if it improves accuracy by 0.5%
    """


    # Set the next feature to be added to the model to be None
    next_feature = None


    # Convert current_features to list
    current_features = list(current_features)


    # If there are no current features, accuracy is 0
    if len(current_features) == 0:
        best_accuracy = 0


    # If there is 1 feature, it needs to be reshaped to be a nested list
```

```
    elif len(current_features) == 1:
        # make and fit logreg model
        temp_model = logreg().fit(
            X_train[current_features].values.reshape(-1, 1), y_train
        )
        # test model on validation data
        val_prediction =
temp_model.predict(X_val[current_features].values.reshape(-1, 1))
        best_accuracy = accuracy_score(y_val, val_prediction)

    # Normal behaviour for > 1 feature
    else:
        temp_model = logreg().fit(X_train[current_features], y_train)
        val_prediction = temp_model.predict(X_val[current_features])
        best_accuracy = accuracy_score(y_val, val_prediction)

    # Test new features and identfies the next which increases accuracy by
more than stopping_criteria
    for feature in features_to_test:
        temp_model = logreg().fit(X_train[current_features + [feature]],
y_train)
        y_prediction = temp_model.predict(X_val[current_features +
[feature]])
        accuracy = accuracy_score(y_val, y_prediction)
        # print(f"Feature being tested is {feature}")
        # print(f"Accuracy from test is {accuracy}")

        if accuracy - best_accuracy >= stopping_criteria:
            best_accuracy = accuracy
            next_feature = feature

    if next_feature != None:
        # Update the user on what is happening
```

```python
        print(f"{next_feature} has been added to the model")
        print(f"The new validation accuracy is {best_accuracy}")

        # Add new feature to feature list
        new_feature_list = current_features + [next_feature]

    else:
        print("No features were added to the model")
        new_feature_list = current_features

    return new_feature_list, best_accuracy

# %% [markdown]
# The previous function is now run in a loop to chose the features used in the model. The number of chosen features is specified.

# %%
def forward_selection(X_train, y_train, X_val, y_val, max_num_features):
    """
    Function which runs the forward selection algorithm for feature selection.
    Uses the add_feature function for chosing whether/which feature to chose.
    """

    # Extract list of possible features
    available_features = list(X_train.columns)

    # Set variable defaults
    model_features = []
    model_accuracy = 0

    # Forward selection algorithm
```

```
    for i in range(0, max_num_features):
        model_features, best_accuracy = add_feature(X_train, y_train, X_val, 
y_val, model_features, available_features)
        if best_accuracy == model_accuracy:
            break
        else:
            for feature in available_features:
                if feature in model_features:
                    available_features.remove(feature)

    # Print updated list of features
    # print(model_features, best_accuracy)
    return model_features, best_accuracy


# %% [markdown]
# This is now run


# %%
num_features = 5
model_features, best_accuracy = forward_selection(X_train, y_train, X_val, 
y_val, num_features)


# %% [markdown]
# ## Creating the model & testing
#
# A logistic regression model is created using the previosuly selected 
features.
#
# These are then tested on the test data and a confusion matrix is generated.


# %%
## Make model
# Curating training & test data
```

```python
final_training = X_train[model_features]
final_val = X_val[model_features]
final_test = X_test[model_features]

# Fit logreg model to data
mylr = logreg().fit(final_training, y_train)

# Model prediction with train, validation and test data
train_prediction = mylr.predict(final_training)
val_prediction = mylr.predict(final_val)
test_prediction = mylr.predict(final_test)

## Generate confusion matrix
# Training data
print("Training data\n")
conf_matrix = confusion_matrix(y_train, train_prediction, labels = [1, 0])
print(conf_matrix)

print(accuracy_score(y_train, train_prediction))
print(precision_score(y_train, train_prediction))
print(recall_score(y_train, train_prediction))

# Validation data
print("\nValidation data\n")
conf_matrix = confusion_matrix(y_val, val_prediction, labels = [1, 0])
print(conf_matrix)

print(accuracy_score(y_val, val_prediction))
print(precision_score(y_val, val_prediction))
print(recall_score(y_val, val_prediction))
```

```
# Test data
print("\nTest data\n")
conf_matrix = confusion_matrix(y_test, test_prediction, labels = [1, 0])
print(conf_matrix)

print(accuracy_score(y_test, test_prediction))
print(precision_score(y_test, test_prediction))
print(recall_score(y_test, test_prediction))

modsummary = ModelSummary(mylr, X_test[model_features], y_test)
modsummary.get_summary()
```

Owain Pill --
-----r2 values

```
#!/usr/bin/env python
# coding: utf-8


# In[53]:



##############################################################################
##################
##############################################################################
##################
##############################################################################
##################
##############################################################################
##################
####
#R2 score and graphs Owain Pill

import sklearn.linear_model
import pandas as pd
```

```
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

brexit = pd.read_csv('brexit1.csv')

column_names = list(brexit)
column_names.pop(0)
#print(column_names)

#create histogram of brexit
plt.hist(brexit['Brexit'], bins=50)  # plotting the historgam with 50 bins
plt.xlabel('Ratio of leave votes')  # setting the label for the x axis
plt.ylabel('Frequency Density')  # setting the label for the x axis
plt.show()

r2_all = {}

for column in column_names:
    X = brexit[column].values.reshape(-1,1)
    y = brexit['Brexit'].values.reshape(-1,1)

    linearModel = sklearn.linear_model.LinearRegression()
    linearModel.fit(X, y)

    #diabetes_y_pred = regr.predict(diabetes_X_test)
    y_pred = linearModel.predict(X)

    #print(r2_score(y, y_pred))
    r2_all[column] = r2_score(y, y_pred)

    if column == 'CON%Lev_4_qual':
```

```
        plt.scatter(X, y, color="black")

        plt.plot(X, y_pred, color="blue", linewidth=3)


        plt.ylabel('Ratio of Leave Votes')

        plt.xlabel('Percentage of Constituents with a Level 4 Qualification
or Higher')


        plt.show()


r2_all = {k: v for k, v in sorted(r2_all.items(), key=lambda item: item[1],
reverse=True)}




#print(r2_all)


df = pd.DataFrame.from_dict(r2_all, orient='index')


#with pd.option_context('display.max_rows', None, 'display.max_columns',
None):  # more options can be specified also

    #print(df)


plt.scatter(X, y, color="black")

plt.plot(X, y_pred, color="blue", linewidth=3)


plt.ylabel('Ratio of Leave Votes')

plt.xlabel('Percentage of Constituents with a Level 4 Qualification or
Higher')


plt.show()




# In[56]:
```

```
cols_list_r2_order = list(r2_all.keys())
cols_list_r2_order=cols_list_r2_order[2:]#remove brexit stuff
print(cols_list_r2_order)


# In[55]:


from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
import numpy as np


train_accuracy = []
validation_accuracy = []
test_accuracy = []


for i in range(1,150):
    print(i)
    selected_columns = cols_list_r2_order[0:i]
    print(selected_columns)
    model = LogisticRegression(C=1e9).fit(X_train[selected_columns], y_train)

    y_train_predicted = model.predict(X_train[selected_columns])
    y_val_predicted = model.predict(X_val[selected_columns])
    y_test_predicted = model.predict(X_test[selected_columns])
    #print(len(X_test[selected_columns]))
```

```
    train_accuracy.append(accuracy_score(y_train, y_train_predicted))
    validation_accuracy.append(accuracy_score(y_val, y_val_predicted))
    test_accuracy.append(accuracy_score(y_test, y_test_predicted))
    '''
    print('======= Accuracy  table =======')
    print('Training accuracy is:    {}'.format(accuracy_score(y_train,
y_train_predicted)))
    print('Validation accuracy is:  {}'.format(accuracy_score(y_val,
y_val_predicted)))
    print('Test accuracy is:  {}'.format(accuracy_score(y_test,
y_test_predicted)))
    '''
plt.figure()
plt.plot(train_accuracy,'b')
plt.plot(validation_accuracy,'r')
#plt.plot(test_accuracy,'y')
plt.legend(['Training data', 'Validation Data'])
plt.xlabel('Complexity: Number of features (chosen at random)')
plt.ylabel('Accuracy')



# In[ ]:
```

--------logistic regression

```
#!/usr/bin/env python
# coding: utf-8


# In[2051]:




from scipy import stats
import numpy as np
import pandas as pd
```

```python
from sklearn.metrics import confusion_matrix, accuracy_score

class ModelSummary:
    """ This class extracts a summary of the model

    Methods
    -------
    get_se()
        computes standard error
    get_ci(SE_est)
        computes confidence intervals
    get_pvals()
        computes p-values
    get_summary(name=None)
        prints the summary of the model
    """

    def __init__(self, clf, X, y):
        """
        Parameters
        ----------
        clf: class
            the classifier object model
        X: pandas Dataframe
            matrix of predictors
        y: numpy array
            matrix of variable
        """
        self.clf = clf
        self.X = X
        self.y = y
        pass
```

```python
    def get_se(self):
        # from here https://stats.stackexchange.com/questions/89484/how-to-compute-the-standard-errors-of-a-logistic-regressions-coefficients
        predProbs = self.clf.predict_proba(self.X)
        X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
        V = np.diagflat(np.product(predProbs, axis=1))
        covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
        return np.sqrt(np.diag(covLogit))

    def get_ci(self, SE_est):
        """
        Parameters
        ----------
        SE_est: numpy array
            matrix of standard error estimations
        """
        p = 0.975
        df = len(self.X) - 2
        crit_t_value = stats.t.ppf(p, df)
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        upper = coefs + (crit_t_value * SE_est)
        lower = coefs - (crit_t_value * SE_est)
        cis = np.zeros((len(coefs), 2))
        cis[:,0] = lower
        cis[:,1] = upper
        return cis

    def get_pvals(self):
        # from here https://stackoverflow.com/questions/25122999/scikit-learn-how-to-check-coefficients-significance
        p = self.clf.predict_proba(self.X)
```

```python
        n = len(p)
        m = len(self.clf.coef_[0]) + 1
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        se = self.get_se()
        t =  coefs/se
        p = (1 - stats.norm.cdf(abs(t))) * 2
        return p

    def get_summary(self, names=None):
        ses = self.get_se()
        cis = self.get_ci(ses)
        lower = cis[:, 0]
        upper = cis[:, 1]
        pvals = self.get_pvals()
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        data = []
        for i in range(len(coefs)):
            currlist = []
            currlist.append(np.round(coefs[i], 3))
            currlist.append(np.round(ses[i], 3))
            currlist.append(np.round(pvals[i], 3))
            currlist.append(np.round(lower[i], 3))
            currlist.append(np.round(upper[i], 3))
            data.append(currlist)
        cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
        sumdf = pd.DataFrame(columns=cols, data=data)
        if names is not None:
            new_names = ['intercept']*(len(names) + 1)
            new_names[1:] = [i for i in names]
            sumdf.index = new_names
        else:
            try:
```

```
                names = list(self.X.columns)
                new_names = ['intercept']*(len(names) + 1)
                new_names[1:] = [i for i in names]
                sumdf.index = new_names
            except:
                pass
        print(sumdf)
        acc = accuracy_score(self.y, self.clf.predict(self.X))
        confmat = confusion_matrix(self.y, self.clf.predict(self.X))
        print('-'*60)
        print('Confusion Matrix (total:{}) \t Accuracy: \t
{}'.format(len(self.X),np.round(acc, 3)))
        print('  TP: {} | FN: {}'.format(confmat[1][1],confmat[1][0]))
        print('  FP: {} | TN: {}'.format(confmat[0][1],confmat[0][0]))



# In[2052]:




from sklearn.linear_model import LogisticRegression as logreg
from sklearn.metrics import accuracy_score
import pandas as pd
import warnings
warnings.filterwarnings('ignore') # prevents future version warnings


#read csv and remove first column which is constituency name
brexit = pd.read_csv('brexit1.csv')
list_cols = list(brexit)[1:]
#print(list_cols)


interest_column = 'Brexit'
interest_column_binary = interest_column + ' binary'
```

```
#set threshhold and convert to binary
brexit[interest_column_binary] = brexit[interest_column] > 0.5
brexit[interest_column_binary] = brexit[interest_column_binary].astype(int)

print(brexit[interest_column].median())
print(len(brexit[brexit[interest_column_binary] == 0]))

#undersample majority in this case
voted_leave = brexit.loc[brexit[interest_column_binary] == 1].sample(172)
brexit = pd.concat((brexit.loc[brexit[interest_column_binary] == 0],
voted_leave))

#print(brexit[interest_column_binary])
print(len(voted_leave))


col_list = list_cols[5:20]#['CON%Lev_4_qual', 'gh']

#test with just the highest r2 variable
X = brexit['CON%Lev_4_qual'].values.reshape(-1,1)
y = brexit['Brexit binary'].values.reshape(-1,1)

mylr = logreg()
mylr.fit(X,y)

model_summary = ModelSummary(mylr,X,y)
model_summary.get_summary()


# In[2053]:
```

```
#splitting columns into train validate and test

from sklearn.model_selection import train_test_split

#split into train validate test, 0.5,0.25,0.25
train, other = train_test_split(brexit, test_size=0.5, random_state=None)

validation, test = train_test_split(other, test_size=0.5, random_state=None)

#random_state=0 default here

#get rid of columns that are not useful
for df in [brexit, train, validation, test]:
    del df['Constituency']
    del df['Brexit']
    del df['Brexit Predicted']
    del df['Conservative Votes']
    del df['Labour Votes']
    del df['Lib Dem Votes']



#create the X and y by dropping or keeping columns
X_train = train.drop(columns=[interest_column_binary])
y_train = train[interest_column_binary]

X_val = validation.drop(columns=[interest_column_binary])
y_val = validation[interest_column_binary]

X_test = test.drop(columns=[interest_column_binary])
```

```
y_test = test[interest_column_binary]


#save a clean version for the very end
X_train_final = train.drop(columns=[interest_column_binary])
y_train_final = train[interest_column_binary]


X_val_final = validation.drop(columns=[interest_column_binary])
y_val_final = validation[interest_column_binary]


X_test_final = test.drop(columns=[interest_column_binary])
y_test_final = test[interest_column_binary]


print(len(X_train))
print(len(X_val))
print(len(X_test))




# In[ ]:







# In[2054]:



#cols in order of r2 value
r2order_cols = ['CON%Lev_4_qual', 'CON%PROF', 'CON%Lev_1_qual',
'CON%L_SUP_TECH', 'CON%SEMI', 'CON%PROF_SCI_TECH', 'CON%Lev_2_qual', 'vgh',
'CON%OPS', 'CON%SKILL', 'CON%H_PROF', 'CON%No_qual', 'fh', 'CON%INF_COM',
'CON%ROUT', 'CON%CONST', 'CON%ASSOC_PROF_TECH', 'High blood pressure
(hypertension)', 'Epilepsy', 'CON%WT_RT_REPM', 'Obesity',
'CON%H_MAN_AD_PROF', 'CON%Appren', 'Rheumatoid arthritis', 'CON%MANF',
```

```
'CON%AC_EMP_E_P', 'COPD', 'Coronary heart disease', 'Median House Price',
'CON%OTHER', 'Diabetes', 'CON%Private_r_land', 'Ratio of median house price
to median salary', 'CON%Private_rent', 'CON%CAR_LEI_', '"Mixed"',
'CON%W_SEW_WAST_REM', 'CON%INA_RET', 'CON%1_fm', 'gh', '"Other"', 'Peripheral
arterial disease', 'Stroke and transient ischaemic attack', 'CON%REAL',
'Chronic kidney disease', 'CON%NO_ADEM_NCHILD', 'CON%INA_STUD', 'Standardised
Weighted Overall Social Mobility Index', 'CON%NO_ADEM', 'Asthma',
'CON%NON_C', 'CON%FT_STUD', 'Heart failure', 'CON%L_MAN_AD_PROF', '60-69',
'CON%1_ps_Oth', 'Are Christian (proportion)', 'Home Ownership (proportion of
households)', 'CON%Own', '"Black"', '30-39', '50-59', 'Learning
disabilities', '70-79', 'Dementia', 'bh', 'CON%EDUC', 'Cardiovascular disease
(primary prevention)', 'Depression', 'CON%1_ps_65', 'Atrial fibrillation',
'CON%AC_F_STUD', '"White"', 'CON%Own_mort', '20-29', 'Labour Votes',
'CON%ELEM', 'CON%FIN_INS', 'CON%I_OCC', 'Number of Businesses',
'CON%Own_out', 'CON%NO', 'CON%UN_16_24', 'CON%TRAN_STOR', 'Conservative Vote
(proportion)', 'CON%Share', 'Labour Vote (proportion)', 'CON%SAL_SERV',
'Cancer diagnosis since 2003', 'CON%Oth_qual', 'Turnout (Proportion)',
'CON%1CV', '80+', 'Number of businesses per capita', 'CON%AC_SELF',
'CON%MAN_DIR_SEN', 'Schizophrenia, bipolar disorder & psychoses', 'CON%2CV',
'CON%LG_E_H_MAN_AD', 'vbh', 'Conservative Votes', 'Lib Dem Votes', 'Lib Dem
Vote (proportion)', 'CON%INA_LONG SICK', 'CON%E_G_S_AIR', 'CON%1_ps', 'Are
Muslim (proportion)', 'CON%3CV', 'CON%Soc_r_Other', '40-49', 'CON%ADMIN_SEC',
'"Asian"', 'CON%ACCOM_FOOD', 'CON%UN_50_74', 'Electorate', 'Share of LSOAs
(small areas) in most deprived decile', 'Unemployment Rate', 'CON%AC_ALL',
'CON%INA_ALL', 'CON%4+CV', 'CON%INA_OTHER', 'CON%HEAL_SOC', 'Average Internet
Speed (Mb/s)', 'CON%Soc_r', 'Are Religous (proportion)', 'CON%A_FOR_FISH',
'CON%Lev_3_qual', 'CON%ADMIN_SUPP', 'CON%ADMIN_DEF', 'CON%M_Q',
'CON%UN_LONG', 'CON%L_EMP', 'CON%Private_r_Oth', 'CON%AC_EMP_E_F',
'CON%N_WKD', 'CON%AC_UNE', 'CON%S_EMP_OWN', 'Are Not Religious (proportion)',
'CON%Rent_free', 'CON%N_WKD_L_EMP', 'CON%1_fm_lp_dc', 'CON%AC_EMP',
'CON%INA_CARE', '19-Oct', '0-9', 'CON%Soc_r_LA', 'CON%NO_ADEM_WCHILD',
'School Funding Per Pupil (Real)', 'CON%UN_NEVER', 'Osteoporosis']


# In[2055]:


##ORDERED SELECTOR

from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
import numpy as np
import matplotlib.pyplot as plt
```

```python
#initialise lists for graphs
train_accuracy = []
validation_accuracy = []
test_accuracy = []

#set range of number of columns to try
lower_range = 1
upper_range = 40
#hacky way of getting the right ranges to show up
for i in range(lower_range):
    train_accuracy.append(0.9)
    validation_accuracy.append(0.9)
    test_accuracy.append(0.9)



#select first i columns from ordered columns to train and validate model and graph
for i in range(lower_range, upper_range+1):
    #print(i)
    selected_columns = r2order_cols[0:i]
    #print(selected_columns)
    model = LogisticRegression(C=1e9).fit(X_train[selected_columns], y_train)

    y_train_predicted = model.predict(X_train[selected_columns])
    y_val_predicted = model.predict(X_val[selected_columns])
    y_test_predicted = model.predict(X_test[selected_columns])
    #print(len(X_test[selected_columns]))

    train_accuracy.append(accuracy_score(y_train, y_train_predicted))
    validation_accuracy.append(accuracy_score(y_val, y_val_predicted))
    test_accuracy.append(accuracy_score(y_test, y_test_predicted))
```

```
'''
    print('======= Accuracy  table =======')
    print('Training accuracy is:    {}'.format(accuracy_score(y_train, y_train_predicted)))
    print('Validation accuracy is:  {}'.format(accuracy_score(y_val, y_val_predicted)))
    print('Test accuracy is:  {}'.format(accuracy_score(y_test, y_test_predicted)))
'''


#plot everything
plt.figure()
#plt.set_xlim(xmin=lower_range, xmax=upper_range)
#fig, ax = plt.subplots()


plt.plot(train_accuracy,'b')
#fig.plot(train_accuracy,'b')
plt.plot(validation_accuracy,'r')
plt.plot(test_accuracy,'y')
plt.legend(['Training data', 'Validation Data'])
plt.xlabel('Complexity: Number of features (chosen in order of R^2 value)')
plt.ylabel('Accuracy')
plt.locator_params(axis="both", integer=True, tight=True)
#fig, ax = plt.subplots()
plt.xlim((lower_range,upper_range))
#plt.ylim((0.7,1))




# In[2056]:




##RANDOM SELECTOR - same as r2cols except with random
```

```
train_accuracy = []
validation_accuracy = []
test_accuracy = []


#X_train_noconst = X_train.drop(columns=['Constituency'])

lower_range = 1
upper_range = 40
#hacky way of getting the right ranges to show up
for i in range(lower_range):
    train_accuracy.append(0.9)
    validation_accuracy.append(0.9)
    test_accuracy.append(0.9)



for i in range(lower_range,upper_range+1):
    #print(i)
    X_train_cols = X_train.sample(n=i,axis='columns')#method that returns
random columns
    #print(len(list(X_train)))
    selected_columns = list(X_train_cols)
    #print(selected_columns)
    model = LogisticRegression(C=1e9).fit(X_train[selected_columns], y_train)

    y_train_predicted = model.predict(X_train[selected_columns])
    y_val_predicted = model.predict(X_val[selected_columns])
    y_test_predicted = model.predict(X_test[selected_columns])
```

```
    #print(len(X_test[selected_columns]))


    train_accuracy.append(accuracy_score(y_train, y_train_predicted))
    validation_accuracy.append(accuracy_score(y_val, y_val_predicted))
    test_accuracy.append(accuracy_score(y_test, y_test_predicted))
    '''
    print('======= Accuracy  table =======')
    print('Training accuracy is:    {}'.format(accuracy_score(y_train,
y_train_predicted)))
    print('Validation accuracy is:  {}'.format(accuracy_score(y_val,
y_val_predicted)))
    print('Test accuracy is:  {}'.format(accuracy_score(y_test,
y_test_predicted)))
    '''
plt.figure()
#plt.set_xlim(xmin=lower_range, xmax=upper_range)
#fig, ax = plt.subplots()


plt.plot(train_accuracy,'b')
#fig.plot(train_accuracy,'b')
plt.plot(validation_accuracy,'r')
plt.plot(test_accuracy,'y')
plt.legend(['Training data', 'Validation Data'])
plt.xlabel('Complexity: Number of features (chosen at random)')
plt.ylabel('Accuracy')
plt.locator_params(axis="both", integer=True, tight=True)
#fig, ax = plt.subplots()
plt.xlim((lower_range,upper_range))
#plt.ylim((0.7,1))




# In[2057]:
```

```
#########FORWARD SELECTION

import numpy as np
from sklearn.metrics import accuracy_score
from sklearn.linear_model import LogisticRegression
import warnings
warnings.filterwarnings('ignore')

#forward selection
train_accuracy = []
validation_accuracy = []
test_accuracy = []

def select_column_to_add(X_train, y_train, X_val, y_val, columns_in_model, columns_to_test):

    column_best = None
    columns_in_model = list(columns_in_model)

    if len(columns_in_model) == 0:
        acc_best = 0
    elif len(columns_in_model) == 1:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model].values.reshape(-1, 1), y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model].values.reshape(-1, 1)))/2
    else:
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model], y_train)
        acc_best = accuracy_score(y_val, mod.predict(X_val[columns_in_model]))/2
```

```
        #divide by 2 to allow as many columns as specified - acc for first column added will always be greater



    for column in columns_to_test:
        #print(column)
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model+[column]], y_train)
        y_pred_val = mod.predict(X_val[columns_in_model+[column]])
        y_pred_train = mod.predict(X_train[columns_in_model+[column]])
        #acc = accuracy_score(y_val, y_pred_val)*2 + accuracy_score(y_train, y_pred_train)
        #acc = acc/2
        acc = accuracy_score(y_val, y_pred_val)

        #if acc - acc_best >= 0.005:  # one of our stopping criteria
        if acc >= acc_best + 0.005:
            #print(acc)
            acc_best = acc
            column_best = column

    if column_best is not None:  # the other stopping criteria
        print('Adding {} to the model'.format(column_best))
        print('The new best validation accuracy is {}'.format(acc_best))
        columns_in_model_updated = columns_in_model + [column_best]
        #print(columns_in_model_updated)
        #add to graphs
        mod = LogisticRegression(C=1e9).fit(X_train[columns_in_model+[column]], y_train)
        y_pred_val = mod.predict(X_val[columns_in_model+[column]])
        y_pred_train = mod.predict(X_train[columns_in_model+[column]])
        train_accuracy.append(accuracy_score(y_train, y_pred_train))
        validation_accuracy.append(accuracy_score(y_val, y_pred_val))
```

```
        #print(train_accuracy)

    else:
        print('Did not add anything to the model')
        columns_in_model_updated = columns_in_model

    return columns_in_model_updated, acc_best




####Forward selector itself

def auto_forward_selection(X_train, y_train, X_val, y_val, max_num_features):

    columns_to_test = list(X_train.columns)
    columns_in_model = []
    current_acc = 0
    same_acc_buffer = 0

    for i in range(0, max_num_features):
        columns_in_model, acc_best = select_column_to_add(X_train, y_train,
X_val, y_val, columns_in_model, columns_to_test)
        if acc_best <= current_acc:
            same_acc_buffer+=1
            columns_in_model = columns_in_model[1:]
            print('Worse')
        else:
            for feature in columns_to_test:
                if feature in columns_in_model:
                    columns_to_test.remove(feature)
        if same_acc_buffer == 4:
```

```
            break

    print(columns_in_model, acc_best)
    return columns_in_model



test1 = auto_forward_selection(X_train, y_train, X_val, y_val, 5)#30


selected_columns = test1


print(selected_columns)


#selected_columns = ['CON%L_SUP_TECH', 'Conservative Vote (proportion)',
'CON%H_MAN_AD_PROF']


model = LogisticRegression(penalty='l1',C=1e9,
solver='liblinear').fit(X_train[selected_columns], y_train)#default C=1e9
only


y_train_predicted = model.predict(X_train[selected_columns])
y_val_predicted = model.predict(X_val[selected_columns])
y_test_predicted = model.predict(X_test[selected_columns])
#print(len(X_test[selected_columns]))
'''
train_accuracy.append(accuracy_score(y_train, y_train_predicted))
validation_accuracy.append(accuracy_score(y_val, y_val_predicted))
test_accuracy.append(accuracy_score(y_test, y_test_predicted))
'''


print('======= Accuracy  table =======')
print('Training accuracy is:    {}'.format(accuracy_score(y_train,
y_train_predicted)))
```

```
print('Validation accuracy is:  {}'.format(accuracy_score(y_val,
y_val_predicted)))

print('Test accuracy is:  {}'.format(accuracy_score(y_test,
y_test_predicted)))




# In[2058]:




#one step out with the others

test_accuracy = test_accuracy[1:]


#print(train_accuracy)

plt.figure()

#plt.set_xlim(xmin=lower_range, xmax=upper_range)

#fig, ax = plt.subplots()


plt.plot(train_accuracy,'b')

#fig.plot(train_accuracy,'b')

plt.plot(validation_accuracy,'r')

plt.plot(test_accuracy,'y')

plt.legend(['Training data', 'Validation Data'])

plt.xlabel('Complexity: Number of features (chosen by forward selection)')

plt.ylabel('Accuracy')

plt.locator_params(axis="both", integer=True, tight=True)

#fig, ax = plt.subplots()

#plt.xlim((lower_range,upper_range))

#plt.ylim((0.7,1))




# In[2059]:
```

```
####Model summary for chosen forward logistic regression


model = LogisticRegression(C=1e9).fit(X_train[selected_columns],
y_train)#penalty='l1',C=1e9, solver='liblinear'


X = X_train[selected_columns]

y = y_train


model_summary = ModelSummary(model,X,y)

model_summary.get_summary()




# In[2060]:




######LASSO To avoid too many features

X_train_standardised = (X_train)


for df in [X_train, X_val, X_test]:

    for col in df.columns:

        #print(df[col])

        #print((df[col]-df[col].mean())/df[col].std())

        df[col] = (df[col]-df[col].mean())/df[col].std()


#print(X_train.head)


####Model summary for chosen forward logistic regression WITH PENALTY


model = LogisticRegression(penalty='l1',C=0.1,
solver='liblinear').fit(X_train[selected_columns], y_train)
```

```python
X = X_train[selected_columns]
y = y_train

model_summary = ModelSummary(model,X,y)
model_summary.get_summary()

#get pvalues from model summary
ps = model_summary.get_pvals()[1:]
ps = list(ps)
#print(ps)
#print(list(model_summary.X))
col_names_final = list(model_summary.X)

#print(np.where(ps = 1))

#indices of where p value is 1
indices1 = [i for i, x in enumerate(ps) if x == 1]#x==1
print(indices1)

#remove columns where p value is too high
def delete_multiple_element(list_object, indices):
    indices = sorted(indices, reverse=True)
    for idx in indices:
        if idx < len(list_object):
            #print(idx)
            list_object.pop(idx)

delete_multiple_element(col_names_final, indices1)

print(col_names_final)
```

```
# In[2061]:


#FINAL MODEL Results
#['CON%L_SUP_TECH', 'CON%Lev_4_qual', 'Conservative Vote (proportion)', 'High blood pressure (hypertension)']

selected_columns = col_names_final
selected_columns = ['CON%Lev_4_qual', 'Conservative Vote (proportion)', 'High blood pressure (hypertension)']
#print(selected_columns)
model = LogisticRegression(C=1e9).fit(X_train_final[selected_columns], y_train_final)

#print(X_train_final.head())

y_train_predicted = model.predict(X_train_final[selected_columns])
y_val_predicted = model.predict(X_val_final[selected_columns])
y_test_predicted = model.predict(X_test_final[selected_columns])
#print(len(X_test[selected_columns]))

print('======= Accuracy  table =======')
print('Training accuracy is:    {}'.format(accuracy_score(y_train_final, y_train_predicted)))
print('Validation accuracy is:  {}'.format(accuracy_score(y_val_final, y_val_predicted)))
print('Test accuracy is:  {}'.format(accuracy_score(y_test_final, y_test_predicted)))


# In[2062]:
```

```
X = X_train_final[selected_columns]
y = y_train_final

model_summary = ModelSummary(model,X,y)
model_summary.get_summary()



# In[ ]:
```

Chinene Chukwuma

-----exploration and binarisation

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:



import pandas as pd
import numpy as np
import scipy as sp
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split



# In[2]:
```

```python
clean_dataset = pd.read_csv('Dataset.txt')
clean_dataset.head()
len(clean_dataset)


# In[3]:


NBC = clean_dataset['Number of businesses per capita']
meanNBC = NBC.mean()
print(meanNBC)


# In[4]:


# using a conditional statement, list bool of NBC samples
NBC < meanNBC


# In[5]:


# new dataframe with only the data about the dead
lowNBC = clean_dataset[NBC <= meanNBC]
lowNBC.head()
```

```
# In[6]:


highNBC = clean_dataset[NBC > meanNBC]


# In[7]:


#create scatter graph to check binarised properly
clean_dataset.plot.scatter(x='Number of businesses per capita', 
y='Unemployment Rate', figsize=(10, 8))
my_canvas = highNBC.plot.scatter(x='Number of businesses per capita', 
y='Unemployment Rate', figsize=(10, 8), color = 'green')
lowNBC.plot.scatter(x='Number of businesses per capita', y='Unemployment 
Rate', figsize=(10, 8), ax=my_canvas, color='red')


# In[8]:


#create histogram to see distribution
the_figure = highNBC.plot.hist(figsize=(10, 8), color='green')
lowNBC.plot.hist(figsize=(10, 8), color='red', ax = the_figure)
the_figure = NBC.plot.hist(figsize=(10, 8))
```

----LR Backward Selection

```
#!/usr/bin/env python
# coding: utf-8
```

```
# In[1]:


import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.linear_model import LogisticRegression as LGR
from sklearn.datasets import load_iris
from mlxtend.plotting import plot_learning_curves


# In[2]:


#classification and confusion matrix
import warnings
warnings.filterwarnings('ignore')


# In[3]:


df = pd.read_csv('FinalDataset.txt')
df = df.drop(columns = 'Number of businesses per capita')
df = df.drop(columns = 'Number of Businesses')
#df.head()
```

```
# In[4]:



# Remove all null value
df.dropna(inplace=True)

# drop the uninformatica column("Loan_ID")
df.drop(labels=['Constituency'],axis=1,inplace=True)
df.reset_index(drop=True,inplace=True)


# In[5]:



from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
cols = df.columns.tolist()
for column in cols:
    if df[column].dtype == 'object':
        df[column] = le.fit_transform(df[column])


# In[6]:



X = df.iloc[:,1:]
y = df["Binary Value: NBC"]


# In[7]:
```

```python
X.head()


# In[8]:


print(X.shape)
print(y.shape)
print(type(X))
print(type(y))


# In[9]:


feature_names=tuple(X.columns)
feature_names


# In[10]:


X.shape, y.shape


# In[11]:
```

```
X_train, X_other, y_train, y_other = train_test_split(X, y, test_size=0.20,
random_state=0)

X_val, X_test, y_val, y_test= train_test_split(X_other, y_other,
test_size=0.50, random_state=0)




# In[12]:




model = LGR(max_iter=1000)

sfs_code = SFS(model,

           k_features='best',

           forward=False,

           floating=False,

           verbose=0,

               scoring='accuracy',

               #scoring='precision',

           #scoring='recall',

            n_jobs=-1,

            cv=5)


sfs1 = sfs_code.fit(X_train, y_train,custom_feature_names=feature_names)




# In[13]:




X_train_sele = sfs1.transform(X_train)

X_val_sele = sfs1.transform(X_val)

X_test_sele = sfs1.transform(X_test)


model.fit(X_train_sele, y_train)
```

```python
print('Training accuracy:', np.mean(model.predict(X_train_sele) ==
y_train)*100)

print('Validation accuracy:', np.mean(model.predict(X_val_sele) ==
y_val)*100)

print('Test accuracy:', np.mean(model.predict(X_test_sele) == y_test)*100)


# In[14]:


# look at the selected feature indices at each step

sfs1.subsets_


# In[15]:


sfs1.get_metric_dict()


# In[16]:


from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs

fig1 = plot_sfs(sfs1.get_metric_dict(confidence_interval=0.95),ylabel =
'Accuracy', kind='std_err')

plt.grid()

plt.title('Sequential Backward Selection')


# In[17]:
```

```python
# the best features
sfs1.k_feature_names_, sfs1.k_feature_idx_



# In[18]:



df = pd.DataFrame.from_dict(sfs1.get_metric_dict()).T
df[["feature_idx","avg_score"]]



# In[19]:



from scipy import stats
from sklearn.metrics import confusion_matrix, accuracy_score

class ModelSummary:
    """ This class extracts a summary of the model

    Methods
    -------
    get_se()
        computes standard error
    get_ci(SE_est)
        computes confidence intervals
    get_pvals()
        computes p-values
    get_summary(name=None)
```

```
        prints the summary of the model
    """

    def __init__(self, clf, X, y):
        """
        Parameters
        ----------
        clf: class
            the classifier object model
        X: pandas Dataframe
            matrix of predictors
        y: numpy array
            matrix of variable
        """
        self.clf = clf
        self.X = X
        self.y = y
        pass

    def get_se(self):
        # from here https://stats.stackexchange.com/questions/89484/how-to-compute-the-standard-errors-of-a-logistic-regressions-coefficients
        predProbs = self.clf.predict_proba(self.X)
        X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
        V = np.diagflat(np.product(predProbs, axis=1))
        covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
        return np.sqrt(np.diag(covLogit))

    def get_ci(self, SE_est):
        """
        Parameters
        ----------
```

```
        SE_est: numpy array
            matrix of standard error estimations
        """
        p = 0.975
        df = len(self.X) - 2
        crit_t_value = stats.t.ppf(p, df)
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        upper = coefs + (crit_t_value * SE_est)
        lower = coefs - (crit_t_value * SE_est)
        cis = np.zeros((len(coefs), 2))
        cis[:,0] = lower
        cis[:,1] = upper
        return cis

    def get_pvals(self):
        # from here https://stackoverflow.com/questions/25122999/scikit-
learn-how-to-check-coefficients-significance
        p = self.clf.predict_proba(self.X)
        n = len(p)
        m = len(self.clf.coef_[0]) + 1
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        se = self.get_se()
        t =  coefs/se
        p = (1 - stats.norm.cdf(abs(t))) * 2
        return p

    def get_summary(self, names=None):
        ses = self.get_se()
        cis = self.get_ci(ses)
        lower = cis[:, 0]
        upper = cis[:, 1]
        pvals = self.get_pvals()
```

```
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        data = []
        for i in range(len(coefs)):
            currlist = []
            currlist.append(np.round(coefs[i], 3))
            currlist.append(np.round(ses[i], 3))
            currlist.append(np.round(pvals[i], 3))
            currlist.append(np.round(lower[i], 3))
            currlist.append(np.round(upper[i], 3))
            data.append(currlist)
        cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
        sumdf = pd.DataFrame(columns=cols, data=data)
        if names is not None:
            new_names = ['intercept']*(len(names) + 1)
            new_names[1:] = [i for i in names]
            sumdf.index = new_names
        else:
            try:
                names = list(self.X.columns)
                new_names = ['intercept']*(len(names) + 1)
                new_names[1:] = [i for i in names]
                sumdf.index = new_names
            except:
                pass
        print(sumdf)
        acc = accuracy_score(self.y, self.clf.predict(self.X))
        confmat = confusion_matrix(self.y, self.clf.predict(self.X))
        print('-'*60)
        print('Confusion Matrix (total:{}) \t Accuracy: \t
{}'.format(len(self.X),np.round(acc, 3)))
        print('  TP: {} | FN: {}'.format(confmat[1][1],confmat[1][0]))
        print('  FP: {} | TN: {}'.format(confmat[0][1],confmat[0][0]))
```

```
# In[20]:



#output code
modsummary = ModelSummary(model, X_train_sele, y_train)
modsummary.get_summary()
```

William Jiang --------------------------------

```
# -*- coding: utf-8 -*-
"""data_science_individual_submission

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1LPSwqpafo1odS4BE2NHr63ncZQ7FnCAD

**1. Initial Dataset Overview**
"""

# Dataset Overview

# read the csv file
import pandas as pd
england_data = pd.read_csv('combined-dataset-england-only.csv')  # read the csv file
england_data.head()
england_data.describe()
```

```
# unemployment rate column
ur = england_data['Unemployment Rate']
# get basic info of unemployment rate
ur.describe()


"""**2. Predictor Variable: Unemployment Rate**"""


# histogram of unemployment rate
figure1 = england_data['Unemployment Rate'].plot.hist(figsize=(10, 8))
figure1.set_ylabel('Number of Constituency')
figure1.set_xlabel('Unemployment Rate')
figure1.set_title('Figure 1: Histogram distribution of unemployment rate per
constituency')


"""**3. Data Preparation**"""


# binaritize unemployment rate with respect to threshold value (mean =
0.028256)
england_data['UR_B'] = england_data['Unemployment Rate'] < 0.028256
england_data['UR_B'] = england_data['UR_B'].astype(float)


# revise dataset
england_data['UR_B'].describe()


# attribute removal
updated_columns = england_data.drop(columns=['Constituency Name',
'Conservative Votes', 'Labour Votes', 'Lib Dem Votes','Are Religous
(proportion)', 'Are Not Religious (proportion)','Are Christian
(proportion)','Are Muslim (proportion)', 'School Funding Per Pupil
(Real)','Unemployment Rate'])
updated_columns.head()
```

```
# spliting dataset into train, validation and test (60%, 20%, 20%)

from sklearn.model_selection import train_test_split


train, validation = train_test_split(updated_columns, test_size=0.2,
random_state=0)

train, test = train_test_split(updated_columns, test_size=0.2,
random_state=0)


# checking size for train, validation and test

print('The sizes for train, validation, and test should be
{}'.format((len(train), len(validation), len(test))))


# dividing dataset into features (X variables) and predictor (y variable)

X_train = train.drop(columns = 'UR_B')

y_train = train['UR_B']


X_validation = validation.drop(columns = 'UR_B')

y_validation = validation['UR_B']


X_test = test.drop(columns = 'UR_B')

y_test = test['UR_B']


# checking of divided dataset

y_train.head()


# standardising training, validation, test data

X_means = X_train.mean(axis=0)

X_stds = X_train.std(axis=0)


X_train_s = (X_train - X_means) / X_stds

X_validation_s = (X_validation - X_means) / X_stds

X_test_s = (X_test - X_means) / X_stds
```

```
# checking data, mean of standardised data should be 0 with std = 1
X_train_s.head()


"""**4. Predictive Models**


**4.1 LogReg LASSO Code**
"""


from sklearn.linear_model import LogisticRegression
from scipy import stats
import numpy as np
import pandas as pd
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score


class ModelSummary:
    """ This class extracts a summary of the model


    Methods
    -------
    get_se()
        computes standard error
    get_ci(SE_est)
        computes confidence intervals
    get_pvals()
        computes p-values
    get_summary(name=None)
        prints the summary of the model
    """


    def __init__(self, clf, X, y):
```

```
        """
        Parameters
        ----------
        clf: class
            the classifier object model
        X: pandas Dataframe
            matrix of predictors
        y: numpy array
            matrix of variable
        """
        self.clf = clf
        self.X = X
        self.y = y
        pass

    def get_se(self):
        # from here https://stats.stackexchange.com/questions/89484/how-to-compute-the-standard-errors-of-a-logistic-regressions-coefficients
        predProbs = self.clf.predict_proba(self.X)
        X_design = np.hstack([np.ones((self.X.shape[0], 1)), self.X])
        V = np.diagflat(np.product(predProbs, axis=1))
        covLogit = np.linalg.inv(np.dot(np.dot(X_design.T, V), X_design))
        return np.sqrt(np.diag(covLogit))

    def get_ci(self, SE_est):
        """
        Parameters
        ----------
        SE_est: numpy array
            matrix of standard error estimations
        """
        p = 0.975
```

```
        df = len(self.X) - 2
        crit_t_value = stats.t.ppf(p, df)
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        upper = coefs + (crit_t_value * SE_est)
        lower = coefs - (crit_t_value * SE_est)
        cis = np.zeros((len(coefs), 2))
        cis[:,0] = lower
        cis[:,1] = upper
        return cis

    def get_pvals(self):
        # from here https://stackoverflow.com/questions/25122999/scikit-learn-how-to-check-coefficients-significance
        p = self.clf.predict_proba(self.X)
        n = len(p)
        m = len(self.clf.coef_[0]) + 1
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        se = self.get_se()
        t =  coefs/se
        p = (1 - stats.norm.cdf(abs(t))) * 2
        return p

    def get_summary(self, names=None):
        ses = self.get_se()
        cis = self.get_ci(ses)
        lower = cis[:, 0]
        upper = cis[:, 1]
        pvals = self.get_pvals()
        coefs = np.concatenate([self.clf.intercept_, self.clf.coef_[0]])
        data = []
        for i in range(len(coefs)):
            currlist = []
```

```
            currlist.append(np.round(coefs[i], 3))
            currlist.append(np.round(ses[i], 3))
            currlist.append(np.round(pvals[i], 3))
            currlist.append(np.round(lower[i], 3))
            currlist.append(np.round(upper[i], 3))
            data.append(currlist)
        cols = ['coefficient', 'std', 'p-value', '[0.025', '0.975]']
        sumdf = pd.DataFrame(columns=cols, data=data)
        if names is not None:
            new_names = ['intercept']*(len(names) + 1)
            new_names[1:] = [i for i in names]
            sumdf.index = new_names
        else:
            try:
                names = list(self.X.columns)
                new_names = ['intercept']*(len(names) + 1)
                new_names[1:] = [i for i in names]
                sumdf.index = new_names
            except:
                pass
        print(sumdf)
        acc = accuracy_score(self.y, self.clf.predict(self.X))
        pre = precision_score(self.y, self.clf.predict(self.X))
        re = recall_score(self.y, self.clf.predict(self.X))


        confmat = confusion_matrix(self.y, self.clf.predict(self.X))
        print('-'*60)
        print('Confusion Matrix (total:{}) \t Accuracy: \t
{}'.format(len(self.X),np.round(acc, 3)))
        print('  TP: {} | FN: {} \t Precision: \t
{}'.format(confmat[1][1],confmat[1][0],np.round(pre,3)))
        print('  FP: {} | TN: {} \t Recall: \t
{}'.format(confmat[0][1],confmat[0][0],np.round(re,3)))
```

```python
import numpy as np
from sklearn.metrics import confusion_matrix


# finding accuracy, precision, recall using different penalty terms (C-value)
with train dataset
X = X_train_s
y = y_train


penalty_term_list = []
acc_test_list = []
pre_test_list = []
rec_test_list = []


# run through penalty term ranging from 0.025 to 1 with step of 0.025
for i in np.arange(0.025, 1, 0.025):
  mod = LogisticRegression(penalty='l1', solver='liblinear', C=i).fit(X, y)
  ModelSummary(mod, X, y).get_summary()
  predicted_y = mod.predict(X)  # this creates our model prediction
  con_mat = confusion_matrix(y, predicted_y, labels=[1, 0])  # the labels
correspond to what is a positive (1) and negative (0) in our dataset
  acc = (con_mat[0, 0] + con_mat[1, 1]) / con_mat.sum()
  pre = con_mat[0,0]/ (con_mat[0,0] + con_mat[1,0])
  rec = con_mat[0,0]/ (con_mat[0,0] + con_mat[0,1])
  penalty_term_list.append(i)
  acc_test_list.append(acc)
  pre_test_list.append(pre)
  rec_test_list.append(rec)


# print all list
print(penalty_term_list)
print(acc_test_list)
```

```python
print(pre_test_list)

print(rec_test_list)


import numpy as np

from sklearn.metrics import confusion_matrix


# finding accuracy, precision, recall using different penalty terms (C-value)
with validation dataset

X = X_validation_s

y = y_validation




penalty_term_list = []

acc_validation_list = []

pre_validation_list = []

rec_validation_list = []


# run through penalty term ranging from 0.025 to 1 with step of 0.025

for i in np.arange(0.025, 1, 0.025):

  mod = LogisticRegression(penalty='l1', solver='liblinear', C=i).fit(X, y)

  ModelSummary(mod, X, y).get_summary()

  predicted_y = mod.predict(X)  # this creates our model prediction

  con_mat = confusion_matrix(y, predicted_y, labels=[1, 0])  # the labels
correspond to what is a positive (1) and negative (0) in our dataset

  acc = (con_mat[0, 0] + con_mat[1, 1]) / con_mat.sum()

  pre = con_mat[0,0]/ (con_mat[0,0] + con_mat[1,0])

  rec = con_mat[0,0]/ (con_mat[0,0] + con_mat[0,1])

  penalty_term_list.append(i)

  acc_validation_list.append(acc)

  pre_validation_list.append(pre)

  rec_validation_list.append(rec)
```

```
# print all list
print(penalty_term_list)
print(acc_validation_list)
print(pre_validation_list)
print(rec_validation_list)


import numpy as np
from sklearn.metrics import confusion_matrix


# finding accuracy, precision, recall using different penalty terms (C-value) with test dataset
X = X_test_s
y = y_test


penalty_term_list = []
acc_test_list = []
pre_test_list = []
rec_test_list = []


# run through penalty term ranging from 0.025 to 1 with step of 0.025
for i in np.arange(0.025, 1, 0.025):
  mod = LogisticRegression(penalty='l1', solver='liblinear', C=i).fit(X, y)
  ModelSummary(mod, X, y).get_summary()
  predicted_y = mod.predict(X)  # this creates our model prediction
  con_mat = confusion_matrix(y, predicted_y, labels=[1, 0])  # the labels correspond to what is a positive (1) and negative (0) in our dataset
  acc = (con_mat[0, 0] + con_mat[1, 1]) / con_mat.sum()
  pre = con_mat[0,0]/ (con_mat[0,0] + con_mat[1,0])
  rec = con_mat[0,0]/ (con_mat[0,0] + con_mat[0,1])
  penalty_term_list.append(i)
  acc_test_list.append(acc)
  pre_test_list.append(pre)
```

```python
  rec_test_list.append(rec)

# print all list
print(penalty_term_list)
print(acc_test_list)
print(pre_test_list)
print(rec_test_list)

# Importing libraries
import matplotlib.pyplot as plt
import numpy as np

# Using pernalty_term_list (C-value) for x-axis value
X = penalty_term_list

# Assign variables to the y axis part of the curve
y = acc_test_list
z = acc_validation_list

# Plotting both the curves simultaneously
plt.plot(X, y, color='r', label='test')
plt.plot(X, z, color='b', label='validation')

# Naming the x-axis, y-axis and the whole graph
plt.xlabel("Penalty Term (C-Value)")
plt.ylabel("Accuracy")
plt.title("Figure 2: Accuracy Score of training and validation set")

# Adding legend, which helps us recognize the curve according to it's color
plt.legend()
```

```python
# To load the display window
plt.show()

# Using pernalty_term_list (C-value) for x-axis value
X = penalty_term_list

# Assign variables to the y axis part of the curve
y = pre_test_list
z = pre_validation_list

# Plotting both the curves simultaneously
plt.plot(X, y, color='r', label='test')
plt.plot(X, z, color='b', label='validation')

# Naming the x-axis, y-axis and the whole graph
plt.xlabel("Penalty Term (C-Value)")
plt.ylabel("Precision")
plt.title("Figure 3: Precision Score of training and validation set")

# Adding legend, which helps us recognize the curve according to it's color
plt.legend()

# To load the display window
plt.show()

# Using pernalty_term_list (C-value) for x-axis value
X = penalty_term_list

# Assign variables to the y axis part of the curve
y = rec_test_list
z = rec_validation_list
```

```
# Plotting both the curves simultaneously
plt.plot(X, y, color='r', label='test')
plt.plot(X, z, color='b', label='validation')

# Naming the x-axis, y-axis and the whole graph
plt.xlabel("Penalty Term (C-Value)")
plt.ylabel("Recall")
plt.title("Figure 4: Recall Score of training and validation set")

# Adding legend, which helps us recognize the curve according to it's color
plt.legend()

# To load the display window
plt.show()

"""**4.2 Decision Tree**"""

# dividing training and validation sets into features and predictor variable
important_features = ['Obesity', 'Turnout (Proportion)', 'Share of LSOAs (small areas) in most deprived decile', 'Schizophrenia, bipolar disorder & psychoses', '0-9']
X_tree = train[important_features]
y_tree = train['UR_B']

X_tree_validation = validation[important_features]
y_tree_validation = validation['UR_B']

X_tree_test = test[important_features]
y_tree_test = test['UR_B']

# standardising training, validation, test data
```

```
X_means = X_tree.mean(axis=0)
X_stds = X_tree.std(axis=0)

X_tree_s = (X_tree - X_means) / X_stds
X_tree_validation_s = (X_tree_validation - X_means) / X_stds
X_tree_test_s = (X_tree_test - X_means) / X_stds


# checking data, mean of standardised data should be 0 with std = 1
X_tree_s.head()

# from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, precision_score, recall_score

# creating training and validation list for graph plotting data
rec_tree_train = []
rec_tree_val = []

for i in range (2,14):
  dt = tree.DecisionTreeClassifier(max_depth = i, min_impurity_decrease =0)
  dt = dt.fit(X_tree_s, y_tree)
  rec_tree_train.append(recall_score(y_tree, dt.predict(X_tree_s)))
  rec_tree_val.append(recall_score(y_tree_validation,
dt.predict(X_tree_validation_s)))

plt.figure()
plt.plot(rec_tree_train, "r", label='Training')
plt.plot(rec_tree_val, "b", label="Validation")
plt.title('Figure 5: Recall of Training and Validation Data against Max Tree
Depth')
plt.xlabel("Max Depth")
```

```
plt.ylabel("Recall")
plt.legend()
plt.show()


# from sklearn.tree import DecisionTreeClassifier
from sklearn import tree
from sklearn.metrics import accuracy_score, precision_score, recall_score


# creating training and validation list for graph plotting data
rec_tree_train = []
rec_tree_val = []


for i in np.arange (0,10):
  dt = tree.DecisionTreeClassifier(max_depth = 3, min_impurity_decrease
=i/100)
  dt = dt.fit(X_tree_s, y_tree)
  rec_tree_train.append(precision_score(y_tree, dt.predict(X_tree_s)))
  rec_tree_val.append(precision_score(y_validation,
dt.predict(X_tree_validation_s)))


plt.figure()
plt.plot(rec_tree_train, "r", label='Training')
plt.plot(rec_tree_val, "b", label="Validation")
plt.title('Figure 6: Recall of Training and Validation Data against Min
Impurity Decrease')
plt.xlabel("Minimum Impurity Decrease (x100)")
plt.ylabel("Recall")
plt.legend()
plt.show()


# # setting max depth = 6 and min impurity decrease = 0.04
dt = tree.DecisionTreeClassifier(max_depth = 3, min_impurity_decrease=0.02)
dt = dt.fit(X_tree_test_s, y_tree_test)
```

```
# visualising decision tree
import sklearn.tree as tree
import graphviz
dot_data = tree.export_graphviz(dt, out_file=None)
graph = graphviz.Source(dot_data)


predictors = X_tree_test_s.columns
dot_data = tree.export_graphviz(dt, out_file=None,
                                feature_names = predictors,
                                class_names = ('Negative', 'Positive'),
                                filled = True, rounded = True,
                                special_characters = True)
graph = graphviz.Source(dot_data)
graph


print('\nFor the test set:')
print('Accuracy: \t{}'.format(accuracy_score(y_tree_validation,
dt.predict(X_tree_validation_s))))
print('Precision: \t{}'.format(precision_score(y_tree_validation,
dt.predict(X_tree_validation_s))))
print('Recall: \t{}'.format(recall_score(y_tree_validation,
dt.predict(X_tree_validation_s))))
```