An Analysis of Mid-Term Elections in the U.S.

Years 1976-2016

Report

Victoria Bateman

Kaushik Sharma

Jyotika Khullar

Table of Contents

Objective

The U.S. Midterm elections are an important event in the U.S. history. They are regarded as a referendum on the current President and his~~/her~~ party's performance. Since World War II the President's party has lost an average of 26 seats in the House and an average of 4 seats in the senate. This year, the 2018 midterm elections were in the limelight at an international level since this year Mid-term results would prove if Donald J Trump's presidency has changed the country and the Republican wave is on the rise. And in the history of the nation it had only happened 6 times that the President's party gained seats in the House or Senate, until now, when Donald Trump's party kept its power in the Senate.

While the current elections were so heated and controversial, our curiosity grew about the past elections and we landed on the MIT Election Data and Science Lab, a website that offers data on U.S. Elections. We accessed two separate data files for the Midterm elections- one at the state level and at the second at district level. We aimed to analyse and infer the age old questions of – Democrats vs Republicans? Taking a deep dive into the datasets, we worked around with the many factors influencing their success at state and district levels.

Our primary goal is to analyze the dataset and predict the winners of both House and Senate elections efficiently. Within the datasets we are wanting to determine the most efficient features that would help us improve the overall predictability and hoping to see that the variables provided have an impact on the candidates winning or losing in the elections. Additionally, we are also aiming to perform analysis and compare results from 3 different machine learning models, in order to find which model will yield the most accurate results and which model will be the most effective model for a classification problem of this kind.

Data Preparation

Data Source

MIT Election Data and Science Lab is a website that offers data on U.S. Election and supports advances in election science by collecting, analyzing, and sharing core data and findings. The site contains data from elections from 1976 until 2018, which is still under analysis. The site also offers new scientific research to be applied to the practice of democracy in the United States.

We accessed two data sets from their lab - U.S. Senate (senate.csv) (1976-2016) and U.S. House (house.csv) (1976-2016). The two sources of data contain details about the names of the candidates, their party name, the candidate votes and total votes. The senate data set contains data of 3,270 candidates and the house data set includes data for 28,272 candidates. Links to the sources can be found in the appendices.

The data sets were forked, cleaned and split into Test and Train Datasets. We performed Multivariate Logistic Regression, Random Forrest Classifier and Bernoulli Naïve Bayesian Analysis on both the Test and Train models for the House and Senate Datasets. The analysis for the House dataset was implement using stats modules Python and the analysis for the Senate Test and Train Dataset was implemented by means of sklearn modules.

Data Quality and Data Cleaning

The Data consisted of good amount of information in the form of at least 28,272 records in the House data file and 3,270 records in the senate. The variables for both are as per below.

	House and Senate
1.	Year – year in which election was held
2.	Office – U.S. House
3.	State- State name
4.	State-po – State Postal Code
5.	State fips – State FIPS Code
6.	State – cen - U.S Census State Code

7.	State ic - ICPSR State Code
8.	District - District number
9.	Stage – Electoral Stage – the different stages - "gen" – general | "pri"- primary
10.	Special – Special Election
11.	Candidate – Name of the Candidate
12.	Party – Party of the Candidate
13.	Write-in – Vote totals associated with write-in candidates
14.	Candidate votes - votes received by this candidate for this particular party
15.	Total Votes – Total number of votes cast for this election
16.	Version – 20171101

Our analysis mainly focuses on the candidate votes and the total votes during the election. To prepare the data for analysis, we choose to drop the following 10 columns, as they do not have any impact on the analysis and the final results: office, state, state po, state cen, state ic, district, stage, special, write-in and version.

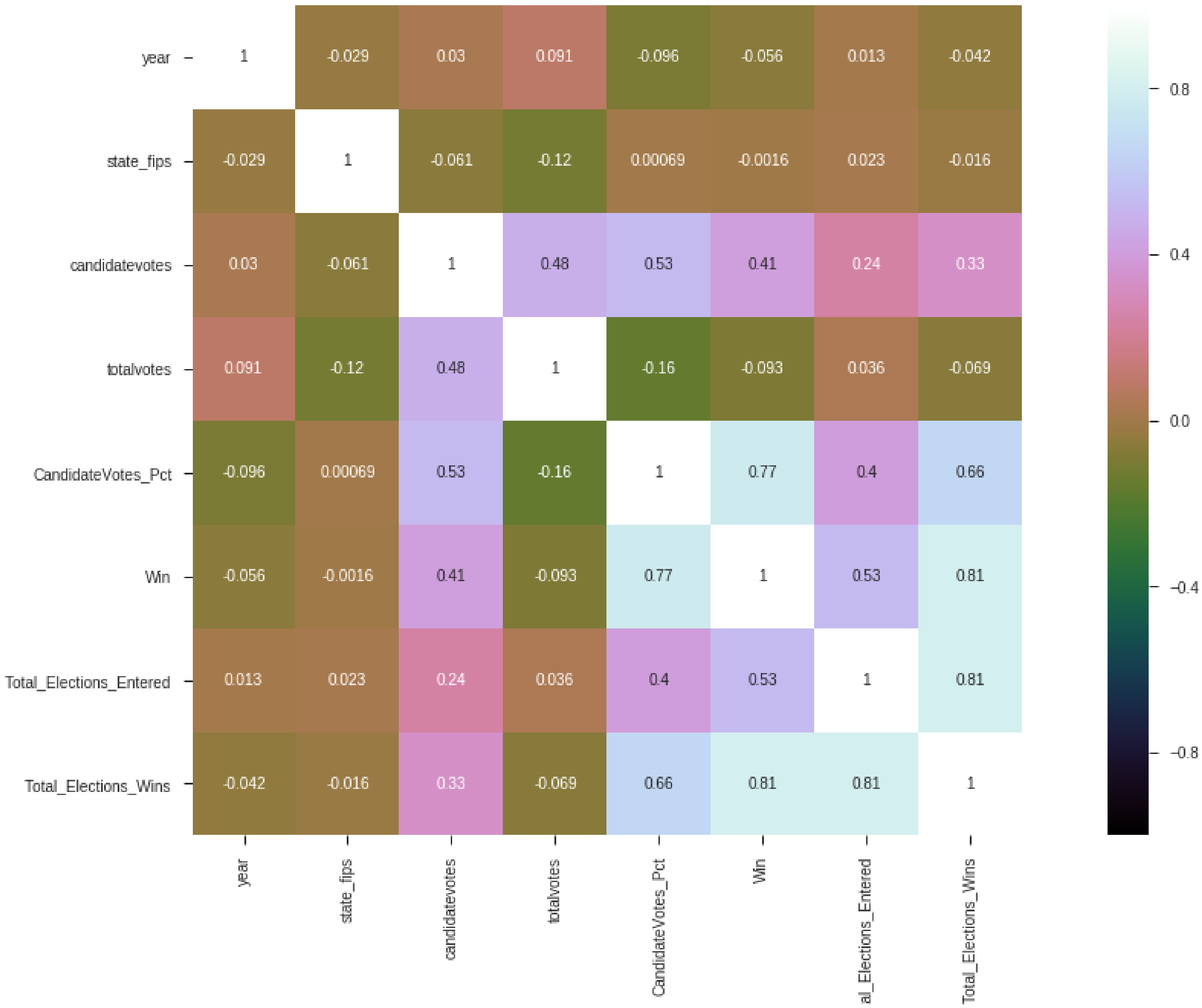
❖ Senate Dataset –

The Exploratory Data Analysis for Senate Dataset included the following:

- The first step in the process was to search for missing values. Having found a few hidden in the 'party' and 'Candidate' columns, we proceeded to remove them by first replacing it with "NaN" and then dropped them all together.
- Once all the missing values were removed and the data set was cleaned up, feature engineering was performed:
  - Using the "Total Votes" and "Candidate Votes" columns, a new column called "CandidateVotes_Pct" was created.
  - Using the "CandidateVotes_Pct", a new column called "Win" was created by assigning a 1 to any candidate that got more than 50% of the votes.
  - 'Candidate' column was used to create a new column labeled- 'total elections entered'. This helps us makes the dataset easier to analyze further as the values are precise.
  - Using the data from columns – 'Candidate' and 'Wins' - a new column is created labelled 'Total elections Win'.
- Once we have the new engineered dataset, we further go on to remove any candidates that got less than 10% of the total votes. This helps remove the outliers while performing further analysis.
- Under the column entitled 'Party', data was categorical and therefore was not precise to perform statistical analysis. To resolve that, One-Hot Encoding was performed, and the values were changed to binary values, i.e. 0 and 1, and the different columns were added back to the dataset.

At the end of the EDA process, having started off with 3,269 entries, we were left with 2,755 entries and 160 columns.

The Heatmap below, for the Senate data below is a quick look of the data used for analysis –



❖ <u>House Dataset-</u>

The Exploratory Data Analysis for House Dataset included the following:

- Due to the huge volume of the data, major parts of the cleanup were done on the csv file itself. Data was organized and sorted and then uploaded on the notebook for further cleaning.
- Due to the cluttered makeup of the dataset, we organised the following:
    - It was observed the ‘Party’ had the most missing values, and they were further replaced with ‘other’.
    - All the votes under “primary elections” were removed, given that our focus is only on general elections.
    - Several of the write in candidates have been removed since they tend to be not recognised at future stages.
    - Candidates with scatter votes are dropped from the dataset and only candidates with clean votes are retained.
    - Candidates with less than 5% votes are also grouped together at this time and excluded from the main data for analysis.
- Once all the missing values were removed and the data set was cleaned up, feature engineering was performed:
    - We create a “Win” feature for each candidate and a rank for each state, year and district based on the candidate’s percentage of votes. We assign rank ‘win’ to those ranked ‘1’ for each year, state and district.
    - To recognize candidates that won uncontested, we created a feature that separates all the candidates with rank ‘1’ – all uncontested candidates. This will help us while running the test data
    - In order to find the candidates that have recently won any election, we create an “incumbent” feature. This helps us calculate the probability of a candidate winning the elections based on recent wins.
    - For a detailed outcome, we also created two new columns - “Republican Win” and “Democrat Win”, to help create additional features that the model can use efficiently.

The Heatmap below, for the House data below provides us with a quick summary of the data used for analysis –

	year	state_fips	state_cen	state_ic	district	candidatevotes	candidatevotes %	totalvotes	Rank	Win	Uncontested	Cum_Elections	Cum_Wins	Republican	Democratic	Republican_Win	Democratic_Win
year		-0.0202228	0.0778732	0.0712712	0.0124257	0.262587	-0.0515659	0.461525	0.0306595	-0.0168211	-0.0460027	0.207225	0.165966	-0.000236987	-0.0262979	0.0586208	-0.0771362
state_fips	-0.0202228		-0.297859	-0.252415	-0.199301	-0.00460227	-0.0165435	-0.00426056	0.00752279	-0.00388838	-0.0100178	0.000451351	0.0397179	-0.00779477	-0.00810616	0.000331251	-0.00624473
state_cen	0.0778732	-0.297859		[illegible]	0.192685	-0.0133135	0.0501176	-0.0651024	-0.0302746	0.0108727	-0.0108881	-0.0263778	-0.0170238	0.0156499	0.00594651	0.0272999	-0.0101736
state_ic	0.0712712	-0.252415	[illegible]		0.165712	-0.00274409	0.0451652	-0.0505248	-0.0311026	0.00849711	-0.0260684	-0.0231546	-0.0125194	0.0177259	0.00577925	0.0243608	-0.0106949
district	0.0124257	-0.199301	0.192685	0.165712		-0.130942	-0.0504107	-0.15092	0.0364071	-0.0183076	-0.0346405	0.0366054	-0.0607394	-0.0164307	-0.00483928	-0.0123577	-0.00346934
special	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
writein	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
andidatevotes	0.262587	-0.00460227	-0.0133135	-0.00274409	-0.130942		0.650985	0.589321	-0.641097	0.61614	0.0377908	0.409104	0.433774	0.0704669	0.114572	0.411687	0.295034
andidatevotes %	-0.0515659	-0.0165435	0.0501176	0.0451652	-0.0504107	0.650985		-0.134981	-0.83714	0.820518	0.45267	0.418525	0.466011	0.00816151	0.219743	0.408757	0.533228
totalvotes	0.461525	-0.00426056	-0.0651024	-0.0505248	-0.15092	0.589321	-0.134981		0.0552533	-0.0586208	-0.271839	0.0652475	0.0509383	0.0298292	-0.0250624	0.0830809	-0.151795
Rank	0.0306595	0.00752279	-0.0302746	-0.0311026	0.0364071	-0.641097	-0.83714	0.0552533		-0.918297	-0.197837	-0.413994	-0.457545	-0.0672007	-0.175249	-0.501648	-0.553352
Win	-0.0168211	-0.00388838	0.0108727	0.00849711	-0.0183076	0.61614	0.820518	-0.0586208	-0.918297		0.215439	0.474884	0.495046	0.00576915	0.123597	0.546281	0.602586
Uncontested	-0.0460027	-0.0100178	-0.0108881	-0.0260684	-0.0346405	0.0377908	0.45267	-0.271839	-0.197837	0.215439		0.112656	0.12691	-0.0499086	0.0789439	0.0571084	0.18891
um_Elections	0.207225	0.000451351	-0.0263778	-0.0231546	0.0366054	0.409104	0.418525	0.0652475	-0.413994	0.474884	0.112656		0.831125	-0.0309689	0.0677533	0.231644	0.310909
Cum_Wins	0.165966	0.0397179	-0.0170238	-0.0125194	-0.0607394	0.433774	0.466011	0.0509383	-0.457545	0.495046	0.12691	0.831125		-0.0263508	0.100731	0.237647	0.335132
Republican	-0.000236987	-0.00779477	0.0156499	0.0177259	-0.0164307	0.0704669	0.00816151	0.0298292	-0.0672007	0.00576915	-0.0499086	-0.0309689	-0.0263508		-0.864776	0.595519	-0.552038
Democratic	-0.0262979	-0.00810616	0.00594651	0.00577925	-0.00483928	0.114572	0.219743	-0.0250624	-0.175249	0.123597	0.0789439	0.0677533	0.100731	-0.864776		-0.51499	0.63836
epublican_Win	0.0586208	0.000331251	0.0272999	0.0243608	-0.0123577	0.411687	0.408757	0.0830809	-0.501648	0.546281	0.0571084	0.231644	0.237647	0.595519	-0.51499		-0.328749
emocratic_Win	-0.0771362	-0.00624473	-0.0101736	-0.0106949	-0.00346934	0.295034	0.533228	-0.151795	-0.553352	0.602586	0.18891	0.310909	0.335132	-0.552038	0.63836	-0.328749	

Challenges and Opportunities

Challenges that presented itself was mostly the large and cluttered data on the House dataset, and extremely limited data on the Senate dataset. This made cleaning and performing analysis a little tricky. Further, the initial attempts at plotting graphs were constantly faced with an error whereby the rows(x-axis) were taken as a single dependent variable and the columns(y-axis) were read as multiple variables.

Despite that, analyzing the data set and learning about the various categories and its implications on the candidate wins was very interesting. The process to remove write-ins and their influence on each candidate win was intriguing. Since the write-ins do not affect as outliers they were not even useful for the analysis, yet somehow, they are still form a part of the U.S Elections.

Analysis and Key Findings

Machine Learning Prediction Models

Defining the type of problem to be solved was the first step in model selection. Midterm election prediction is dynamic in the sense it is possible to treat it as a regression or a classification problem. As a regression problem it is necessary to predict the percentage of candidate votes for each candidate. From there it is possible to convert the predicted vote percentage to a binary win/loss variable. Alternatively, elections can be treated directly as a classification problem with a binary outcome as a win/loss for each candidate.

Initially it was envisaged this would be treated as a regression problem. However, when attempting to train and test multivariate linear regression and Random Forest Regressor models it became evident that this approach was problematic. The models do not understand the concept of percentages, therefore, the model generated, in some cases, generated nonsensical candidate voting percentages, for example, negative percentages and percentages above 100.

In addition, using regression models for binary outcome prediction requires an additional step to turn the predicted percentages into a binary win/loss variable. With elections it is not always as simply as implementing a simple cutoff for a win/loss across all seats as there may be multiple viable candidates in a given race. This step would require further modelling on top of the regression analysis. Consequently, it was more efficient to directly treat it as a binary classification problem.

With this in mind, appropriate binary classification algorithms were chosen for both the Senate and the House models. The following algorithms were used to train and test the models – Logistic Regression, Random Forest Classifier and Naïve Bayes.

Logistic Regression uses the logistic function and a linear combination of independent variables to tackle binary classification problems. It is in the same family as Linear Regression and is underpinned by the frequentist approach to statistics. As a comparison, Naïve Bayes was included as an Bayesian alternative to frequentist Logistic Regression. Bernoulli Naïve Bayes was chosen because it is a fairly simple implementation for binary classification problems.

Finally, Random Forest Classifier was added to the modelling mix as an ensemble algorithm. Random Forest models are easy to implement and provide information about the importance of each independent variable in the model.

The remaining subsections discuss the results of these modelling approaches for the Senate and House elections data.

❖ Senate Dataset –

Logistic Regression Model – The dataset is broken into Train and Test and Logistic Regression is performed on both sets. A confusion Matrix was produced for both as well and the results for which are as follows:

Train Data Set

Accuracy – 97%

	Predicted Success	Predicted Failure
True Success	1347	18
True Failure	30	450

Test Data Set

Accuracy – 98%

	Predicted Success	Predicted Failure
True Success	670	6
True Failure	12	222

- Random Forest Classifier- We begin by splitting the data into test and train models and then defining our dependent and independent variables. For the Train dataset, we obtain an accuracy of 99%. Running a Confusion Matrix, we can see below the Predicted Success and Predicted Failures for the Train Dataset:

	Predicted Success	Predicted Failure
True Success	1362	3
True Failure	1	479

Further the same functions are performed on the Test dataset and we observe that the model yields an accuracy of 92%. Below is the confusion matrix for the Test dataset predicting the false success and failure:

	Predicted Success	Predicted Failure
True Success	663	13
True Failure	6	228

The mean squared error for the dataset is 0.02.

- Bernoulli Naïve Bayes Model- For the Bayesian Analysis we followed similar model as we did for the Random Forest Classifier. We begin by splitting the data into test and train, followed by fitting in the independent and dependent variable. Then we calculated the BernoulliNB score for each model and produced the confusion matrix.

Train Model – Accuracy of BernoulliNB model – 97 %

	Predicted Success	Predicted Failure
True Success	1313	52
True Failure	2	478

Test Model – Accuracy of BernoulliNB Score – 96%

	Predicted Success	Predicted Failure
True Success	650	26
True Failure	2	232

The mean squared error for the dataset is 0.03.

❖ <u>House Dataset –</u>

The house election dataset was split into train and test data. The same train and test datasets were used for all three models with the only addition being adding a constant term for the Logistic Regression model. Using the same train and test datasets across all three models allowed for a direct comparison of performance.

➢ Logistic Regression – the scikit-learn implementation of Binomial GLM was used to train and test the model. The y predicted output was a set of probabilities indicating the probability of the candidate winning the election.

The standard error, Z score and p-value is calculated for the test and train models. The results of the trained model as applied to the trained data are below:

Generalized Linear Model Regression Results

Dep. Variable:	Win	**No. Observations:**	12235
Model:	GLM	**Df Residuals:**	12229
Model Family:	Binomial	**Df Model:**	5
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	nan
Date:	Sun, 02 Dec 2018	**Deviance:**	nan
Time:	18:47:22	**Pearson chi2:**	8.44e+08
No. Iterations:	100	**Covariance Type:**	nonrobust

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.8550	0.187	-20.572	0.000	-4.222	-3.488
Uncontested	38.8609	2.89e+06	1.34e-05	1.000	-5.67e+06	5.67e+06
Cum_Elections	0.2650	0.021	12.870	0.000	0.225	0.305
Cum_Wins	1.2507	0.046	27.298	0.000	1.161	1.340
Republican	2.5744	0.181	14.190	0.000	2.219	2.930
Democratic	2.6745	0.182	14.708	0.000	2.318	3.031

Based on the p-value for the train data, the ‘uncontested’ variable was dropped from the model as it was not statistically significant. After the ‘uncontested’ variable was dropped from the data the model was retrained. The results for which are as follows:

Generalized Linear Model Regression Results

Dep. Variable:	Win	**No. Observations:**	12235
Model:	GLM	**Df Residuals:**	12230
Model Family:	Binomial	**Df Model:**	4
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-5442.8
Date:	Sun, 02 Dec 2018	**Deviance:**	10886.
Time:	18:47:23	**Pearson chi2:**	1.54e+09
No. Iterations:	8	**Covariance Type:**	nonrobust

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-3.7679	0.181	-20.761	0.000	-4.124	-3.412
Cum_Elections	0.2624	0.020	12.883	0.000	0.222	0.302
Cum_Wins	1.2906	0.046	28.198	0.000	1.201	1.380
Republican	2.5288	0.176	14.389	0.000	2.184	2.873
Democratic	2.6793	0.176	15.216	0.000	2.334	3.024

The mean squared error for the Test model is 0.132 and the accuracy score is 84%.
The mean squared error for the Train model: 0.133 and the accuracy score is 83%.

We further ran confusion matrix on both the test and train data to show false positives (Type I errors) in the bottom left quadrant and false negatives (Type II errors) in the upper right quadrant:

Train Dataset

	Predicted Success	Predicted Failure
True Success	5865	244
True Failure	1777	4349

Test Dataset

	Predicted Success	Predicted Failure
True Success	2919	109
True Failure	869	2130

- Random Forest Classifier - the model was trained with resulting importance weights for each independent variable. The most important independent variable in the model was total previous election wins ('Cum_Win'):

Independent Variable	**Importance Weight**
Cum_Win	0.690788
Cum_Elections	0.260157
Republican	0.027702
Democratic	0.021353

The mean squared error for the Test Dataset is 0.16127 and for the Train dataset is 0.16264.

For the Test dataset, we obtain an accuracy score of 0.83872. Further the same functions are performed on the Train dataset and we obtain an accuracy score of 0.83735.

Below is confusion matrix for both the test and the train dataset:

Train Dataset

Test Dataset

	Predicted Success	Predicted Failure
True Success	5841	268
True Failure	1722	4404

	Predicted Success	Predicted Failure
True Success	2899	129
True Failure	843	2156

➢ Bernoulli Naïve Bayes Model- For the Bayesian Analysis we followed similar model as we did for the Random Forest Classifier. We begin by splitting the data into test and train, followed by fitting in the independent and dependent variable. The Mean Squared Error Was calculated for both the datasets and the results were as follows – Train Dataset – 0.17188; Test Dataset- 0.16791. We then calculated the BernoulliNB score for each model and produced the confusion matrix.

Train Dataset

Accuracy – 82%

	Predicted Success	Predicted Failure
True Success	5893	216
True Failure	0	6126

Test Dataset

Accuracy – 83%

	Predicted Success	Predicted Failure
True Success	2935	93
True Failure	0	2999

Comparison of Models

All models for both House and Senate elections were measured against three core evaluation metrics. Each evaluation metric measures a different aspect of model performance as explain below:

- **Accuracy Score:** this is the percentage of all correctly predicted Y values.
- **Type I and II Errors:** this is presented as a confusion matrix showing the total number of false positives, Type I errors, and false negatives, Type II errors.
- **Mean Squared Error:** is the mean of the squared difference between Y and Y predicted.

A comparison of the results from the analysis performed on the Senate Dataset are as follows:

	Score - Train Data	Score - Test Data	Mean Squared Error - Test Data
Logistic Regression	0.973984	0.980220	0.019780
Random Forest	0.997832	0.979121	0.020879
Bernoulli Naive Bayes	0.970732	0.969231	0.030769

For the Train Model we see that our accuracy is between 97% - 99% for each of the predictive models and for the Test Data the accuracy is ranging between 96% - 98%.

The high accuracy values clearly show that the model is overfitting and will not generalize to other datasets. This is due to 2 major reasons:

a. Small dataset - Though our dataset is from 1976 - 2016 and has 3,270 rows and 10 columns, the total wins were fewer than 700 over a 20 year period. This is due to the fact that in any given election cycle, the Senate has just 33 out of the 100 seats, up for election. Which results in the dataset being extremely small.
b. Lack of relevant features - Having dropped a few columns that were not relevant to the overall data process, we added 2 new columns by using feature engineering principles. This resulted in a total of just 5 independent features, before the

one hot encoding process. As a result of this limited feature set, the model is overfitting and producing results that will not hold when introduced to a new dataset.

A comparison of the evaluation metrics for each model run on the House elections data are as follows:

Model	Score = Train Data	Score = Test	MSE = Test
Logistic Regression	0.834818	0.837730	0.132919
Random Forest Classifier	0.837515	0.838892	0.161108
Naive Bayes	0.828116	0.832089	0.167911

As can be seen above, all models achieved a similar accuracy score - approximately 83% across the board. The key difference between the models was the MSE, although Logistic Regression had a slightly better MSE with a result of 0.13 compared to Random Forest and Naive Bayes.

Conclusion

An initial analysis shows that future performance of a candidate or his party can be predicted using the data from past elections. However, stronger model can be built to improve predictability and increase the accuracy scores.

We do see that each dataset presents its own challenges. The volume of data within the Senate dataset prevents from yielding a tur accuracy score for any of the regression models. While the House dataset contains such a huge volume of data that the cleaning process becomes tedious and time consuming.

In order to treat the data as a regression problem it was necessary to have a uniform measure of the candidates votes. We achieved that by converting the percentage of candidate votes into binary win/loss variables. After splitting the data sets into Test and train models and attempting the machine learning analysis on each, we discover that the models do not acknowledge the candidate percentages as a valid variable. The models tend to produce illogical candidate vote percentages ranging from negative to values above 100.

The results suggest that all the three models generalise well which may be partly explained by the fact both the train and test datasets were relatively large samples. However, there is still room for improved accuracy scores as there were only four viable features. Further exploration of macro and micro social and economic data, national and regional as well as polling data and other political market research are complimentary for election prediction models.

Appendix

1. MIT Election Data and Science Lab, 2017, "U.S. House 1976-2016", https://doi.org/10.7910/DVN/IG0UN2 Harvard Dataverse, V2 [last accessed October 2018].

2. MIT Election Data and Science Lab, 2017, "U.S. Senate 1976-2016", https://doi.org/10.7910/DVN/PEJ5QU Harvard Dataverse, V2 [last accessed October 2018].

3. Z. (n.d.). Write-in Votes. Retrieved December 12, 2018, from https://electoral-vote.com/evp2018/Feature_stories/write-ins.html [last accessed November 2018].

PYTHON CODE

University of Toronto School of Continuing Studies

SCS3251 - 016 Statistics for Data Science

Predicting U.S. Midterm Elections: 1976 to 2014

- Bateman, Victoria
- Khullar, Jyotika
- Sharma, Kaushik

Utility Code

Below is utility code including import of relevant Python libraries and functions for reuse throughout the project code.

Import

In [387]:
```
import pandas as pd
import numpy as np
import sklearn as sk
import statsmodels.api as sm
import matplotlib.pyplot as plt
import scipy.stats as stats
import seaborn as sns
from sklearn.metrics import roc_curve, auc
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier as RFC
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import BernoulliNB as BNB
```

Functions

In [388]:

```
# Display the unique values for each of the feature
# or column in the dataframe passed to the function
def get_unique_values ( df ) :
    for column in df :
        print( column, ':' )
        print( df[ column ].unique() )
        print( '\n' )

# Select all categorical features/columns from the data
# Drop them from the dataset

def drop_categorical ( df, data_type, ax = 1 ) :
    cat_columns = df.select_dtypes( include = data_type )

    df = df.drop( cat_columns, axis=ax )

    return df

# *Dropping the following functions for now as this approach to splitting is b
ias *

# Generate test and train datasets by year without constant
#def get_train_test (input_df, y_variable):

    # Split into train and test
#     test_year = input_df['year'].max()
#     test = input_df[input_df['year'] == test_year]
#     train = input_df[input_df['year'] < test_year]
#     y_train, y_test = train[ y_variable ], test[ y_variable ]
#     test = test.drop([ y_variable ,'year','const'],axis=1)
#     train = train.drop([ y_variable ,'year','const'],axis=1)
#     X_train, X_test = train, test

    # Looking at the shape of the train and test data

#     return  X_train, X_test, y_train, y_test

# Generate test and train datasets by year with contant for linear regression
#def get_train_test_lm (input_df, y_variable):

    # Split into train and test
#     test_year = input_df['year'].max()
#     test = input_df[input_df['year'] == test_year]
#     train = input_df[input_df['year'] < test_year]
#     y_train, y_test = train[ y_variable ], test[ y_variable ]
#     test = test.drop([ y_variable ,'year'],axis=1)
#     train = train.drop([ y_variable ,'year'],axis=1)
#     X_train, X_test = train, test

    # Looking at the shape of the train and test data

#     return  X_train, X_test, y_train, y_test

# Clean up data removing categories of data from the DataFrame
def clean_votes (df, col, excl):
    for i in excl:
```

```
        df = df[df[ col ] != i]
    return df
```

Data Cleaning

In [389]:

```
# Import the data as a DataFrame
df = pd.read_excel('1976-2016-house clean.csv.xlsx')
```

In [390]:

```
# Dataframe columns with volume count and data types at a glance
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26762 entries, 0 to 26761
Data columns (total 16 columns):
year                     26762 non-null int64
state                    26762 non-null object
state_po                 26762 non-null object
state_fips               26762 non-null int64
state_cen                26762 non-null int64
state_ic                 26762 non-null int64
office                   26762 non-null object
district                 26762 non-null int64
stage                    26732 non-null object
special                  26762 non-null bool
candidate                26762 non-null object
party                    25041 non-null object
writein                  26762 non-null bool
candidatevotes           26762 non-null int64
candidatevotes %         26761 non-null float64
totalvotes               26762 non-null int64
dtypes: bool(2), float64(1), int64(7), object(6)
memory usage: 2.9+ MB
```

In [391]:

```
# Extensive research was done into missing party values
# The majority of missing data followed no pattern with some candididates
# Correctly attributed to one party in some records and not in others
# Some corrections have been made at the data file level
# For the remainder a decision was made to impute missing party with 'Other'
df['party'] = df['party'].fillna('Other')
```

In [392]:
```
# Determining unique values of the columns in the dataframe
# This is used to explore the make-up of the variables
get_unique_values( df )
```

```
year :
[1976 1978 1980 1982 1984 1986 1988 1990 1992 1994 1996 1998 2000 2002
 2004 2006 2008 2010 2012 2014 2016]


state :
['Alabama' 'Alaska' 'Arizona' 'Arkansas' 'California' 'Colorado'
 'Connecticut' 'Delaware' 'Florida' 'Georgia' 'Hawaii' 'Idaho' 'Illinois'
 'Indiana' 'Iowa' 'Kansas' 'Kentucky' 'Louisiana' 'Maryland'
 'Massachusetts' 'Michigan' 'Minnesota' 'Mississippi' 'Missouri' 'Montana'
 'Nebraska' 'Nevada' 'New Hampshire' 'New Jersey' 'New Mexico' 'New York'
 'North Carolina' 'North Dakota' 'Ohio' 'Oklahoma' 'Oregon' 'Pennsylvania'
 'Rhode Island' 'South Carolina' 'South Dakota' 'Tennessee' 'Texas' 'Utah'
 'Vermont' 'Virginia' 'Washington' 'West Virginia' 'Wisconsin' 'Wyoming'
 'Maine']


state_po :
['AL' 'AK' 'AZ' 'AR' 'CA' 'CO' 'CT' 'DE' 'FL' 'GA' 'HI' 'ID' 'IL' 'IN'
 'IA' 'KS' 'KY' 'LA' 'MD' 'MA' 'MI' 'MN' 'MS' 'MO' 'MT' 'NE' 'NV' 'NH'
 'NJ' 'NM' 'NY' 'NC' 'ND' 'OH' 'OK' 'OR' 'PA' 'RI' 'SC' 'SD' 'TN' 'TX'
 'UT' 'VT' 'VA' 'WA' 'WV' 'WI' 'WY' 'ME']


state_fips :
[ 1  2  4  5  6  8  9 10 12 13 15 16 17 18 19 20 21 22 24 25 26 27 28 29
 30 31 32 33 34 35 36 37 38 39 40 41 42 44 45 46 47 48 49 50 51 53 54 55
 56 23]


state_cen :
[63 94 86 71 93 84 16 51 59 58 95 82 33 32 42 47 61 72 52 14 34 41 64 43
 81 46 88 12 22 85 21 56 44 31 73 92 23 15 57 45 62 74 87 13 54 91 55 35
 83 11]


state_ic :
[41 81 61 42 71 62  1 11 43 44 82 63 21 22 31 32 51 45 52  3 23 33 46 34
 64 35 65  4 12 66 13 47 36 24 53 72 14  5 48 37 54 49 67  6 40 73 56 25
 68  2]


office :
['US House']


district :
[ 1  2  3  4  5  6  7  0  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
 48 49 50 51 52 53]


stage :
['gen' 'pri' nan]


special :
```

```
[False  True]


candidate :
['Jack Edwards' 'Bill Davenport' 'William L. "Bill" Dickinson' ...
 'Liz Cheney' 'Ryan Greene' 'Lawrence Gerard Struempf']


party :
['republican' 'democrat' 'prohibition' 'national democrat' 'libertarian'
 'independent' 'Other' 'peace and freedom' 'american independent'
 'socialist workers' 'u.s. labor' 'la raza unida' 'american' 'indpendent'
 'communist' 'conservative' 'socialist labor'
 'independents for godly government' "people's" 'workers' 'white power'
 'human rights' 'independent american' 'new majority' 'labor'
 'regular democracy' 'pro-life' 'restoration' 'individual needs center'
 'politicians are crooks' "independent taxpayer's watchdog"
 'jobs, equality, peace' 'consumer action'
 'individual americans independence' 'bring us together' 'silent majority'
 "people's independent" 'liberal' 'mayflower' 'coequal citizens'
 'independent conservatives' 'revolutionary workers' 'constitution'
 'citizens for haas' 'united states labor' 'aloha democratic' 'socialist'
 "worker's party" 'united labor' 'betsy ross'
 'peoples independent coalition' 'united taxpayers'
 'independent neighbors party' "taxpayer's citizens" 'national statesman'
 'liberty union' 'statesman' 'national democratic party of alabama'
 'citizens' 'constitution party of illinois' 'by petition' 'no party'
 'new union' 'industrial government' 'j.e.b. party inc' 'no slogan'
 'contempt of court' 'pro-life independent' 'human rights ratification'
 'youth against draft' 'the independent alternative'
 'independent for congress' 'action talks'
 "people's independent coalition" 'none' 'right to life' 'new alliance'
 'independent neighbors' 'free libertarian' 'workers world'
 'foglietta (democrat)' 'independent congressional' 'morris for congress'
 'young socialist alliance' 'unaffiliated-american' 'anti-drug'
 'bipartisan good government' 'socialist party of iowa' 'workers league'
 'world federalist' 'the unbossed independent' '"mr. liberty"'
 'independence' 'popular' 'nuclear freeze' 'consumer' 'milton street'
 'reef for congress' 'krill for congress' 'small is beautiful'
 'free peoples' 'independent political choice' 'people before profits'
 'concerns of people' 'constitutionalist' 'rainbow coalition'
 'tisch independent citizens' 'christian american' 'citizens-socialist'
 'constitutional freedom' 'ratepayers against lilco' 'ivs'
 'american eagle' 'populist' 'labor and farm'
 'quality congressional representation' 'labor for maine' 'awg' 'citizen'
 'stop financing communism' 'inflation fighting housewife'
 'port authority=crooks' 'let freedom ring'
 'public power alternative to lilco' 'concerned citizens against lilco'
 'rate payers against lilco' 'nei' 'effective congress' 'fair trade'
 'land-water-legacy' 'war against aids' 'solidarity'
 'nominated by petition' 'peace, jobs,  justice'
 'workers against concessions' 'democratic-farmer-labor' 'grassroots'
 'pro-life conservative' "poor man's" "people's choice" 'time for change'
 '"all-peoples congress"' 'citizens against rising electric rates'
 "vote children '88" 'drug fighter' 'independent voter'
 'independent progressive line' 'jobs' 'liberty' 'jim wham party'
 'no party affiliation' 'tisch independent citizen' 'pride and honesty'
```

'god we trust' 'back to basics' 'reform' 'world without war'
'better affordable government' 'right to vote' 'tax brake' 'bronx voters'
'american system independent' 'alaskan independence' 'green'
'natural law' 'american grass roots alternative' 'a connecticut party'
'concerned citizens' 'petitioning candidate' 'louanner peters party'
'recovery' 'economic recovery' 'peace, jobs, justice'
'pro-democracy reform' 'freedom for larouche' 'for the people'
'independent voters' 'unenrolled' 'independent-republican'
'independents for perot' 'perot choice' 'term limits candidate'
'pro-life pro-family veteran' 'pro-life independent conservative'
'american first populist' 'anti-tax' 'freedom, equality, prosperity'
'donald of moorestown' 'the independent party'
'basic reformed government' 'equality, brotherhood, justice'
'first populist' "the people's candidate" 'independent for freedom'
'you gotta believe' 'capitalist' 'no nonsense government'
"people's congressional preference" 'independent for change'
'independents for change' "independent people's network"
'restore public trust' 'independents' 'clean up congress'
'an independent voice' 'stop tax increases' 'long island first'
'independent fusion' 'voter rights' 'common sense' 'economic justice'
'change congress' 'none of above' 'magerman for congress'
'new independent' 'ross perot independent' 'independent thinking'
'a delaware party' 'best' 'united independents' 'taxpayers'
'independent maine greens' 'mississippi taxpayers' 'united we serve'
'democracy in action' 'larouche was right' 'fascist' 'we the people'
'fed up party' 'perot hispano american' 't.b.a. green'
'independence fusion' 'ax taxes' 'independent nomination'
'cash for congress' 'citizens with szabo' 'patriot' 'gun control'
'u.s. taxpayers' 'americans' 'non partisan' 'working class'
'socialist equality' 'independent grass roots' 'save medicare' 'freedom'
'protect seniors' 'francis worley congress' 'unaffiliated' 'term limits'
'indepdendence' 're' 'minnesota taxpayers' 'legal marijuana now'
'anti-federalist' 'indepdendent' 'star tax cut' 'fusion' 'pacific'
'workers campaign' 'constitutional' 'vermont grassroots'
'american heritage' 'american constitution' 'earth federation' 'other'
'other candidates' 'citizens first' 'working families' 'school choice'
'socialist worker' 'pacific green' 'conscience for congress'
'united citizens' 'one earth' 'no new taxes' 'american first'
'honesty, humanity, duty' 'lower tax independent' 'human rights advocate'
'anti-corruption doctor' 'the american party' 'republican/democrat'
'independent home protection' 'constitution party of florida'
'nebraska party' 'indepdence' 'fair' 'centrist' 'peace and justice'
'nonpartisan' 'randolph for congress' 'healthcare' 'personal choice'
'mountain' 'wisconsin green' 'impeach now' 'concerns of the people'
'pirate' 'unity' 'progressive' 'preserve green space' 'a new direction'
'socialist party usa' 'the patriot movement' 'remove medical negligence'
'diversity is strength' 'withdraw troops now' 'the moderate choice'
'pro life conservative' 'impeach bush now' 'independent green'
'unity party of america' 'american constitution party'
'term limits for the united states congress' 'think independently'
'lindsay for congress' 'rock the boat' 'hsing for congress'
'all-day breakfast party' 'prosperity not war' 'common sense ideas'
'eliminate the primary' 'vote people change' 'energy independence'
'socialist action' 'independent party of delaware' 'blue enigma'
'tea party' 'florida whig party' 'tax revolt independent'
'citizen legislator' 'bring home troops' 'party free'
'independent progressive' 'defend american constitution' 'marklovett.us'

```
 'american labor' 'new jersey tea party' 'your country again'
 'american renaissance movement' 'for americans' 'be determined'
 'green tea patriots' 'action no talk' 'agent of change'
 'truth vision hope' 'gravity buoyancy solution' 'tax revolt'
 'independence, vote people change' 'american congress party'
 'towne for congress' 'indepdent citizen for constitutional government'
 'coalition on government reform' 'independent no war bailout'
 'americans elect' 'constitutional conservative' "the people's agenda"
 'conservative, compassionate, creative' 'legalize marijuana'
 'bob?s for jobs' 'none of them' 'overthrow all incumbents'
 'independent reform candidate' 'restoring america?s promise'
 'unity is strength' 'abundant america' 'change, change, change'
 'opposing congressional gridlock' 'bednarski for congress' 'votekiss'
 'country party' 'marketing managers' 'jos_ peÃ\x90alosa'
 'no party preference' 'natural law party' 'we deserve better'
 'stop boss politics' 'change is needed' 'of the people' 'd-r party'
 'wake up usa' '911 truth needed' 'seeking inclusion'
 'bullying breaks hearts' 'future.vision' 'start the conversation'
 'allen 4 congress' 'flourish every person' 'mr. smith goes to congress'
 'nonaffiliated' 'veterans party of america' 'americanindependents'
 'make government work' 'representing the 99%'
 "people's independent progressive" 'for political revolution'
 'economic growth' 'wake up america' 'nsa did 911' 'women of power'
 "new beginning's" 'financial independence' 'legalize marijuana party'
 'teddy roosevelt progressive' "women's equality" 'haris bhatti party'
 'blue lives matter' 'stop iran deal' 'transparent government'
 'upstate jobs' 'trump conservative']


writein :
[False  True]


candidatevotes :
[98257 58906 90069 ... 75466 10362  6621]


candidatevotes %  :
[62.51638353 37.47916269 57.60287026 ...  3.49050188  2.55846484
  0.15108892]


totalvotes :
[157170 156362 108048 ... 362271 363780 258788]
```

In [393]:
```
df['special'] = df['special'].apply(lambda x: 1 if x == 'True' else 0)
```

In [394]:
```
# This project is focussed on general elections
# Remove any candidate records relating to primaries
df = df[df['stage'] != 'pri']
```

In [395]:

```
# Drop the scatter votes as these are not relevant to the predictive model
# Scatter votes are where voters spoil their ballot
# For further information on scattering votes visit: http://www.renewamerica.c
om/columns/contrada/121116
df = df[df['candidate'] !='scatter']
```

In [396]:

```
# Several write-in candidates have won an election, but are rarely known in ad
vance
# In some cases write-in candidates may not be recognised or even actually peo
ple
# Nor, are even intending to run - more info at https://electoral-vote.com/evp
2018/Feature_stories/write-ins.html
# On the basis that write-ins cannot be predicted with regularity they have be
en removed
df = df[df['writein'] == False]
```

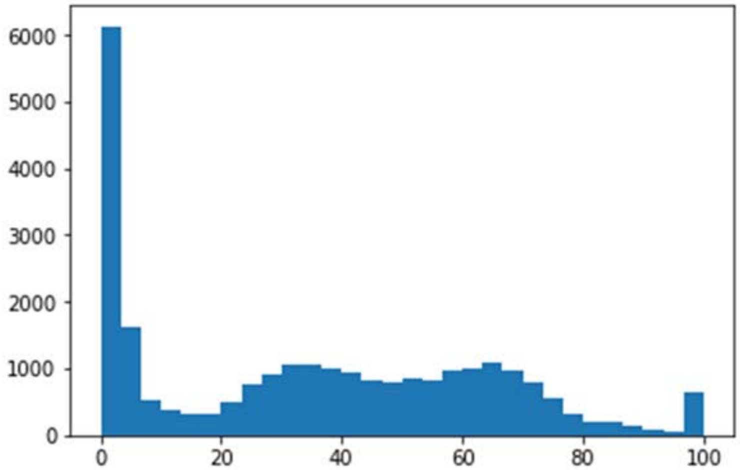
In [397]:

```
# Running descriptive statistics on the DataFrame
df.describe()
```

Out[397]:

	year	state_fips	state_cen	state_ic	district	special	candidat
count	25882.000000	25882.000000	25882.000000	25882.000000	25882.000000	25882.0	25882.0
mean	1996.694150	28.605131	50.945136	37.019743	10.135268	0.0	68962.6
std	11.966991	15.050682	26.827559	22.197526	10.164284	0.0	60183.0
min	1976.000000	1.000000	11.000000	1.000000	0.000000	0.0	0.0
25%	1986.000000	17.000000	23.000000	14.000000	3.000000	0.0	7003.5
50%	1998.000000	31.000000	47.000000	35.000000	6.000000	0.0	63927.5
75%	2006.000000	39.000000	74.000000	53.000000	14.000000	0.0	111114.0
max	2016.000000	56.000000	95.000000	82.000000	53.000000	0.0	322514.0

In [398]:
```
# There appears a significant number of candidates with zero or near zero vote
s
# Let's look at the distribution of the percentage of votes for all election y
ears
plt.hist(df['candidatevotes % '],bins=30)
plt.show()
```



In [399]:

```
# Show the number of times a candidate appears in the histroical data
# There appears to be a number of candidates that are not relevant
# For example, blank votes is not relevant for this model
df.groupby(['candidate'])['year'].aggregate('count').sort_values(ascending=False)
```

Out[399]:
```
candidate
Other                               415
Blank Vote/Scattering               373
Blank Vote                           65
Blank Vote/Void Vote/Scattering      54
Charles B. Rangel                    47
Peter T. King                        37
Gary L. Ackerman                     35
Eliot L. Engel                       32
Carolyn B. Maloney                   30
Michael R. McNulty                   28
John J. LaFalce                      28
Maurice D. Hinchey                   27
Edolphus Towns                       26
James T. Walsh                       26
Jerrold Nadler                       25
Major R. Owens                       24
Louise McIntosh Slaughter            24
Nita M. Lowey                        24
Carolyn McCarthy                     24
John M. McHugh                       21
Gerald B. H. Solomon                 21
Rosa L. DeLauro                      20
George Miller                        19
Don Young                            19
Christopher H. Smith                 19
C. W. Bill Young                     19
Edward J. Markey                     19
John D. Dingell                      19
Ike Skelton                          18
Gene Taylor                          18
                                   ...
Margaret Chapman                      1
Margaret B. Buhrmaster                1
Margaret A. Palms                     1
Margaret "Peggy" Miller               1
Marek Tyszkiewicz                     1
Margie Akin                           1
Marguerite A. Page                    1
Marilyn K. Stone                      1
Marianna Blume                        1
Marilyn Fowler                        1
Marilyn D. Clancy                     1
Marilinda Garcia                      1
Marihelen Wheeler                     1
Marielle Hammett Kronberg             1
Marie Richey                          1
Marie G. Delany                       1
Marie Agnes Fese                      1
Marianna Wertz                        1
Mariana Blume                         1
Marguerite Chandler                   1
Marian S. Henry                       1
Maria Selva                           1
Maria M. Passa                        1
Maria M. Hustace                      1
Maria Karczewski                      1
```

```
Maria Guadalupe Garcia           1
Maria Green                      1
Maria Elena Milton               1
Maria Armoudian                  1
 David L. Miller                 1
Name: year, Length: 14335, dtype: int64
```

In [400]:

```
# Drop any records that relate to non-candidates/spoiled ballots based on the
 table above
non_cand = ['Blank Vote/Scattering', 'Blank Vote','Blank Votes','Blank Vote/Vo
id Vote/Scattering','Other']
df = clean_votes(df, 'candidate', non_cand)
```

In [401]:

```
# Review number of candidates with less than 5% votes
# A large proportion of candidates have low candidate votes
df_low = df[df['candidatevotes % '] < 5]
df_low.describe()
```

Out[401]:

	year	state_fips	state_cen	state_ic	district	special	candidatevotes
count	6711.000000	6711.000000	6711.000000	6711.000000	6711.000000	6711.0	6711.000000
mean	1997.604828	29.686783	48.288184	34.670094	10.618239	0.0	4017.394129
std	11.467001	14.183467	27.897622	23.007456	10.606116	0.0	3029.864984
min	1976.000000	1.000000	11.000000	1.000000	0.000000	0.0	1.000000
25%	1990.000000	20.000000	21.000000	13.000000	3.000000	0.0	1703.500000
50%	1998.000000	34.000000	35.000000	24.000000	7.000000	0.0	3284.000000
75%	2007.000000	37.000000	74.000000	54.000000	15.000000	0.0	5561.500000
max	2016.000000	56.000000	95.000000	82.000000	53.000000	0.0	19333.000000

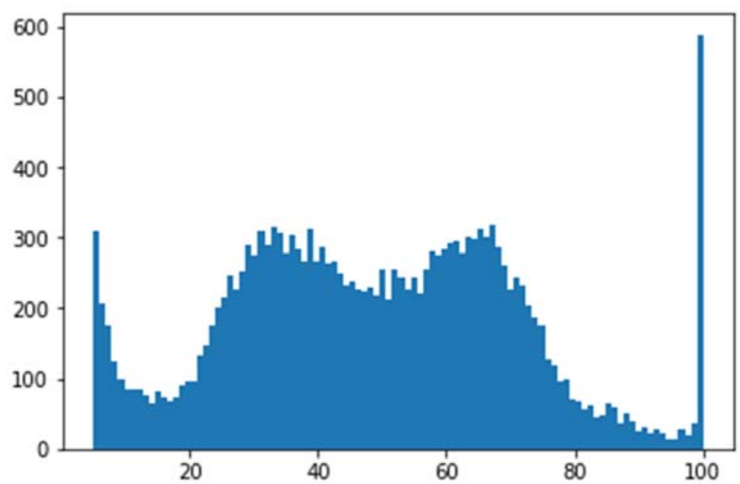
In [402]:
```
# Let's sort the number of candidates with less than 5% of the votes
# There are a few candidates for democrats and republicans
# Mostly these are small/fringe parties
df_low.groupby(['party'])['candidate'].count().sort_values(ascending=False)
```

```
Out[402]: party
libertarian                         2112
independent                          822
green                                401
natural law                          355
conservative                         344
liberal                              228
right to life                        217
independence                         184
working families                     179
peace and freedom                    139
reform                               129
socialist workers                    129
constitution                         116
american independent                  96
american                              65
u.s. taxpayers                        62
Other                                 56
none                                  48
republican                            47
no party affiliation                  40
independent american                  39
u.s. labor                            33
democrat                              30
citizens                              23
other                                 23
freedom                               22
socialist                             22
labor                                 22
new alliance                          21
populist                              21
                                    ...
people's independent progressive       1
perot hispano american                 1
nei                                    1
pirate                                 1
poor man's                             1
popular                                1
port authority=crooks                  1
preserve green space                   1
pride and honesty                      1
pro life conservative                  1
party free                             1
pacific                                1
overthrow all incumbents               1
other candidates                       1
new beginning's                        1
new independent                        1
new jersey tea party                   1
new majority                           1
new union                              1
no new taxes                           1
no nonsense government                 1
nonaffiliated                          1
none of above                          1
none of them                           1
nonpartisan                            1
```

```
nsa did 911                          1
of the people                        1
one earth                            1
opposing congressional gridlock      1
"all-peoples congress"               1
Name: candidate, Length: 385, dtype: int64
```

In [403]:
```python
# Let's drop these candidates as they may create noise in our data
# When we apply this model in future elections it may be better
# to apply only to key parties
df = df[df['candidatevotes % '] >= 5]
```

In [404]:
```python
# Now let's look at the spread of vote % after dropping candidates with less t
han 5% of the vote
# It's clear that this has cut out the peak at the low vote %
# The candidiates with 100% or near 100% of the votes have been left in
# Because they are likely to be uncontested elections
plt.hist(df['candidatevotes % '],bins=100)
plt.show()
```



Feature Engineering

In [405]:
```python
# Create a win feature for each candidate
# This will be the y variable - it will be used to determine the efficacy of t
he model

# First let's create a rank for each year, state and district based on the can
didate's % of votes
df['Rank'] = df.groupby(['year','state','district'])['candidatevotes % '].rank
(ascending=False)

# Then using rank assign win to those ranked 1 for each year, state and distri
ct
df['Win'] = df['Rank'].apply(lambda x: 1 if x == 1 else 0)
```

In [406]:
```
# Create an uncontested feature to confirm that all uncontested candidates win


# Create a col comparing the rank of each row with the row+1
df['Shift'] = df['Rank'].shift(-1)
# Find the difference between the two rows
df['Diff'] = df['Win']-df['Shift']
# If both rows have rank 1 then the content for the current row is uncontested
 (i.e. there is only rank 1 for that district)
df['Uncontested'] = df['Diff'].apply(lambda x: 1 if x == 0 else 0)
# Drop the Diff and Shift columns as they're not longer required
df = df.drop(['Diff','Shift'], axis=1)
```

In [407]:
```
# Create an incumbent feature - intuitively if a candidate is currently
# in office then they are more likely to win the next election

# Find all winning candidates from the most recent year
#incum = df[(df['year'] == df['year'].max()) & (df['Win'] == 1)]
# For each candidate identify if they are the winner in the most recent electi
on
# If so, then they are the (current) incumbent
#df['Incumbent'] = df['candidate'].apply(lambda x: 1 if x in (incum['candidat
e'].unique()) else 0)

# This feature was dropped as it had a poor p-value
```

In [408]:
```
# Next let's find out how many times each candidate has entered an election be
fore

ent = df[['candidate','year']].sort_values(['candidate','year'])

# Count the number of times a candidate has entered by year then take the
# cumulative sum of the years the candidate has entered an election
ent = ent.groupby(['candidate','year'])['year'].aggregate('count')
ent = pd.DataFrame(ent)
ent['Cum_Elections'] = ent.groupby(['candidate','year'])['year'].cumsum()
ent = ent.drop(['year'],axis=1)
ent = ent.reset_index().sort_values(['candidate','year'])

# Add this new feature to the original dataframe
df = df.merge(ent, how='left')

# Drop the duplicate variables
df = df.loc[:,~df.columns.duplicated()]
```

```
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\ipykernel_l
auncher.py:9: FutureWarning: 'year' is both an index level and a column labe
l.
Defaulting to column, but this will raise an ambiguity error in a future vers
ion
  if   name__== '  main  ':
```

In [409]:
```
# Next let's find out how many times each candidate has won an election before

# Count the number of times a candidate has won by year then take the
# cumulative sum of the years the candidate has won an election
won = df[['candidate','year','state_ic','district','party','Win']].sort_values
(['candidate','state_ic','district','party','year'])
won = won.groupby(['candidate','state_ic','district','party','year'])['Win'].a
ggregate('sum')
won = pd.DataFrame(won)
won['Cum_Wins'] = won.groupby(['candidate','state_ic','district','party','yea
r']).sum().groupby(level=[0,1,2,3]).cumsum()

# Next shift the results by one otherwise the election result of that year wil
l be included
won['Cum_Wins_2'] = won.groupby(['candidate','state_ic','district','party'])[
'Cum_Wins'].shift(1)
won['Cum_Wins_2'] = won['Cum_Wins_2'].fillna(0)
won['Cum_Wins'] = won['Cum_Wins_2']
won = won.drop(['Cum_Wins_2'],axis=1)
won = won.reset_index().sort_values(['candidate','state_ic','district','party'
,'year'])

# Add this new feature to the original dataframe
df = df.merge(won, on=['candidate','state_ic','district','year','party','Win'
],how='left')

# Drop the duplicate variables
df = df.loc[:,~df.columns.duplicated()]
```

In [410]:
```
# Fill any candidate that do not have CUM_WINS
# Only two as many data quality issues have been fixed in the data cleaning se
ction
df['Cum_Wins'] = df['Cum_Wins'].fillna(df['Cum_Wins'].mean())
```

In [411]:
```
# Create dummy variables for candidates in the republican and democrat party
df['Republican'] = df['party'].apply(lambda x: 1 if x == 'republican' else 0)
df['Democratic'] = df['party'].apply(lambda x: 1 if x == 'democrat' else 0)
```

In [412]:
```
# Next let's create variables to show where republican and democratic candidia
tes have won
df['Republican_Win'] = df['Republican']+df['Win']
df['Democratic_Win'] = df['Democratic']+df['Win']
df['Republican_Win'] = df['Republican_Win'].apply(lambda x: 1 if x == 2 else 0
)
df['Democratic_Win'] = df['Democratic_Win'].apply(lambda x: 1 if x == 2 else 0
)
```

In [413]:
```
# Convert the bool special election field to binary
df['special'] = df['special'].apply(lambda x: 1 if x == 'True' else 0)
```

In [414]:
```
# Drop all remaining categorical variables before feeding data into the model
df = drop_categorical(df,object)
```

Explore

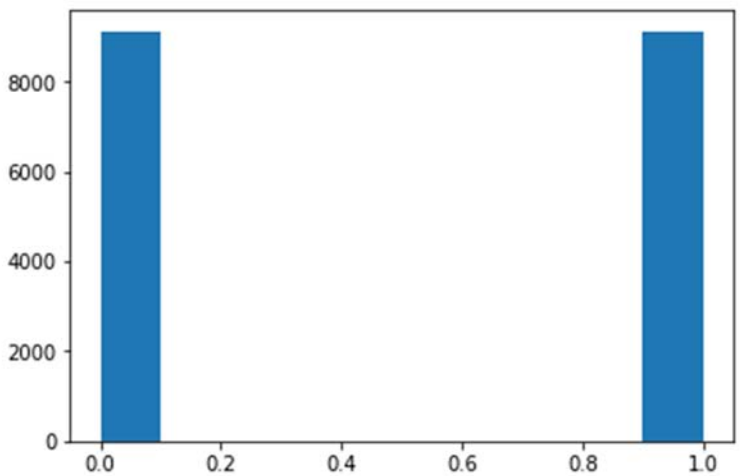
In [415]:

```
# Show how many candidiates have won and lost
# This appears to be approximately evenly distributed

wins = round(100*df['Win'].sum()/df['Win'].count(),2)
losses = round(100*(df['Win'].count()-df['Win'].sum())/df['Win'].count(),2)
print("Candidiate wins:",wins,"Candidate losses",losses)

#Plot a histogram of win losses
plt.hist(df['Win'],bins=10)
plt.show()
```

```
Candidiate wins: 49.97 Candidate losses 50.03
```



In [416]:
```
# Run a correlation matrixt check for multicollinearity
corr = df.corr()
corr.style.background_gradient()
```

```
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\matplotlib
\colors.py:512: RuntimeWarning: invalid value encountered in less
  xa[xa < 0] = -1
```

Out[416]:

	year	state_fips	state_cen	state_ic	district	special	writei
year	1	-0.0202228	0.0778732	0.0712712	0.0124257	nan	na
state_fips	-0.0202228	1	-0.297859	-0.252415	-0.199301	nan	na
state_cen	0.0778732	-0.297859	1	0.977965	0.192685	nan	na
state_ic	0.0712712	-0.252415	0.977965	1	0.165712	nan	na
district	0.0124257	-0.199301	0.192685	0.165712	1	nan	na
special	nan	nan	nan	nan	nan	nan	na
writein	nan	nan	nan	nan	nan	nan	na
candidatevotes	0.262587	-0.00460227	-0.0133135	-0.00274409	-0.130942	nan	na
candidatevotes %	-0.0515659	-0.0165435	0.0501176	0.0451652	-0.0504107	nan	na
totalvotes	0.461525	-0.00426056	-0.0651024	-0.0505248	-0.15092	nan	na
Rank	0.0306595	0.00752279	-0.0302746	-0.0311026	0.0364071	nan	na
Win	-0.0168211	-0.00388838	0.0108727	0.00849711	-0.0183076	nan	na
Uncontested	-0.0460027	-0.0100178	-0.0108881	-0.0260684	-0.0346405	nan	na
Cum_Elections	0.207225	0.000451351	-0.0263778	-0.0231546	0.0366054	nan	na
Cum_Wins	0.165966	0.0397179	-0.0170238	-0.0125194	-0.0607394	nan	na
Republican	-0.000236987	-0.00779477	0.0156499	0.0177259	-0.0164307	nan	na
Democratic	-0.0262979	-0.00810616	0.00594651	0.00577925	-0.00483928	nan	na
Republican_Win	0.0586208	0.000331251	0.0272999	0.0243608	-0.0123577	nan	na
Democratic_Win	-0.0771362	-0.00624473	-0.0101736	-0.0106949	-0.00346934	nan	na

In [417]:

```
# Show correlations of all variables with win
# This indicates strength of relationship with y
cor = df.corr()['Win'].sort_values()
cor
```

Out[417]:

```
Rank                -0.918297
totalvotes          -0.058621
district            -0.018308
year                -0.016821
state_fips          -0.003888
Republican           0.005769
state_ic             0.008497
state_cen            0.010873
Democratic           0.123597
Uncontested          0.215439
Cum_Elections        0.474884
Cum_Wins             0.495046
Republican_Win       0.546281
Democratic_Win       0.602586
candidatevotes       0.616140
candidatevotes %     0.820518
Win                  1.000000
special                   NaN
writein                   NaN
Name: Win, dtype: float64
```

In [435]:

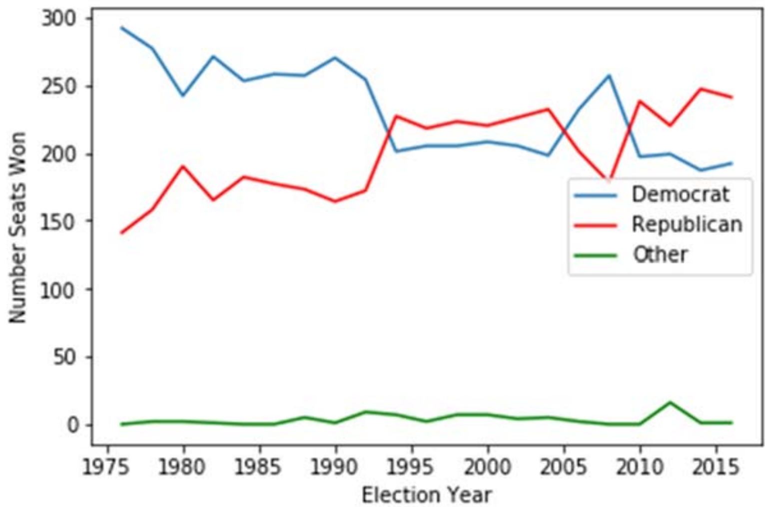
```
# Let's look at the split of Democrat, Republican and Other wins over time

year_wins = df.groupby(['year'])['Win'].sum()
year_wins = pd.DataFrame(year_wins)
year_wins = year_wins.rename({'Win':'Total_Seats'},axis=1)
year_wins['Republican_Win'] = df.groupby(['year'])['Republican_Win'].sum()
year_wins['Democratic_Win'] = df.groupby(['year'])['Democratic_Win'].sum()
year_wins['Other_Win'] = year_wins['Total_Seats']-(year_wins['Democratic_Win']
+year_wins['Republican_Win'])
```

In [447]:

```
# Plot the results for Repub, Demo and Other over time
plt.plot(year_wins['Democratic_Win'],label='Democrat')
plt.plot(year_wins['Republican_Win'],label='Republican',color='r')
plt.plot(year_wins['Other_Win'],label='Other',color='g')

# Add details to the plot
plt.legend()
plt.ylabel('Number Seats Won')
plt.xlabel('Election Year')
plt.show()
```



In [383]:

```
# Drop all of the following unnecessary columns - these columns are in fact ca
tegorical data hidden as integers
df = df.drop(['state_fips','state_cen', 'state_ic','district','writein','speci
al'],axis=1)

# Drop the following columns as this data will not be available for prediction
 purposes
df = df.drop(['candidatevotes','candidatevotes % ','totalvotes','Republican_Wi
n','Democratic_Win','Rank'],axis=1)
```

Select and Run Model

1. Multivariate Logistic Regression

In [330]:
```
# Add constant ready for logistic regression
df = sm.add_constant(df)

# Split into train and test
# y = win
y = df['Win']

# Drop y from the X data set
X = df
X = X.drop(['Win','year'],axis=1)

# Split the data into test and train
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, rand
om_state=48)
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[330]: ((12235, 6), (6027, 6), (12235,), (6027,))

In [331]:
```
# Run and fit GLM using binomial
model_GLM = sm.GLM(y_train, X_train, family=sm.families.Binomial())
results_GLM = model_GLM.fit()
```

```
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmodels
\genmod\families\family.py:880: RuntimeWarning: invalid value encountered in
true_divide
  n_endog_mu = self._clean((1. - endog) / (1. - mu))
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmodels
\genmod\families\family.py:881: RuntimeWarning: invalid value encountered in
log
  resid_dev = endog * np.log(endog_mu) + (1 - endog) * np.log(n_endog_mu)
```

In [332]:
```
# Generate evaluation summary for GLM model
results_GLM.summary()
```

```
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmodels
\genmod\families\family.py:932: RuntimeWarning: divide by zero encountered in
true_divide
  special.gammaln(n - y + 1) + y * np.log(mu / (1 - mu)) +
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmodels
\genmod\families\family.py:933: RuntimeWarning: divide by zero encountered in
log
  n * np.log(1 - mu)) * var_weights
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\statsmodels
\genmod\families\family.py:933: RuntimeWarning: invalid value encountered in
add
  n * np.log(1 - mu)) * var_weights
```

Out[332]:

Generalized Linear Model Regression Results

Dep. Variable:	Win	**No. Observations:**	12235
Model:	GLM	**Df Residuals:**	12229
Model Family:	Binomial	**Df Model:**	5
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	nan
Date:	Sun, 02 Dec 2018	**Deviance:**	nan
Time:	18:47:22	**Pearson chi2:**	8.44e+08
No. Iterations:	100	**Covariance Type:**	nonrobust

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-3.8550	0.187	-20.572	0.000	-4.222	-3.488
Uncontested	38.8609	2.89e+06	1.34e-05	1.000	-5.67e+06	5.67e+06
Cum_Elections	0.2650	0.021	12.870	0.000	0.225	0.305
Cum_Wins	1.2507	0.046	27.298	0.000	1.161	1.340
Republican	2.5744	0.181	14.190	0.000	2.219	2.930
Democratic	2.6745	0.182	14.708	0.000	2.318	3.031

In [333]:
```
# Based on the above p-values drop uncontested
X_test = X_test.drop(['Uncontested'],axis=1)
X_train = X_train.drop(['Uncontested'],axis=1)
```

In [334]:
```
# Refit the model
model_GLM2 = sm.GLM(y_train, X_train, family=sm.families.Binomial())
results_GLM2 = model_GLM2.fit()
predict_GLM2 = results_GLM2.predict(X_train)
```

In [335]:
```
# Generate evaluation summary for GLM model 2
results_GLM2.summary()
```

Out[335]:

Generalized Linear Model Regression Results

Dep. Variable:	Win	**No. Observations:**	12235
Model:	GLM	**Df Residuals:**	12230
Model Family:	Binomial	**Df Model:**	4
Link Function:	logit	**Scale:**	1.0000
Method:	IRLS	**Log-Likelihood:**	-5442.8
Date:	Sun, 02 Dec 2018	**Deviance:**	10886.
Time:	18:47:23	**Pearson chi2:**	1.54e+09
No. Iterations:	8	**Covariance Type:**	nonrobust

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	-3.7679	0.181	-20.761	0.000	-4.124	-3.412
Cum_Elections	0.2624	0.020	12.883	0.000	0.222	0.302
Cum_Wins	1.2906	0.046	28.198	0.000	1.201	1.380
Republican	2.5288	0.176	14.389	0.000	2.184	2.873
Democratic	2.6793	0.176	15.216	0.000	2.334	3.024

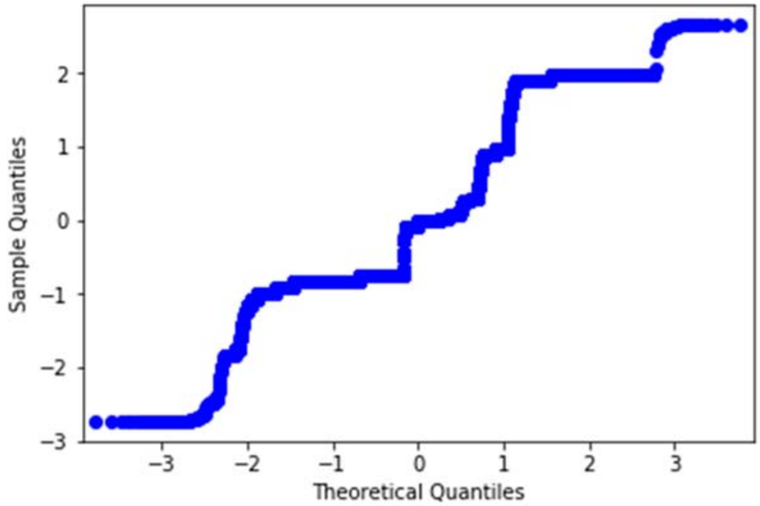
In [336]:
```
# Calculate the Mean Squared Error (MSE) for both the train and test data
mse_GLM_train = sk.metrics.mean_squared_error(y_train,predict_GLM2)
predict_GLM2_test = results_GLM2.predict(X_test)
mse_GLM_test = sk.metrics.mean_squared_error(y_test,predict_GLM2_test)
print("Train MSE:", mse_GLM_train,",","Test MSE:",mse_GLM_test)
```

```
Train MSE: 0.13371736397496445 , Test MSE: 0.13291898512856354
```

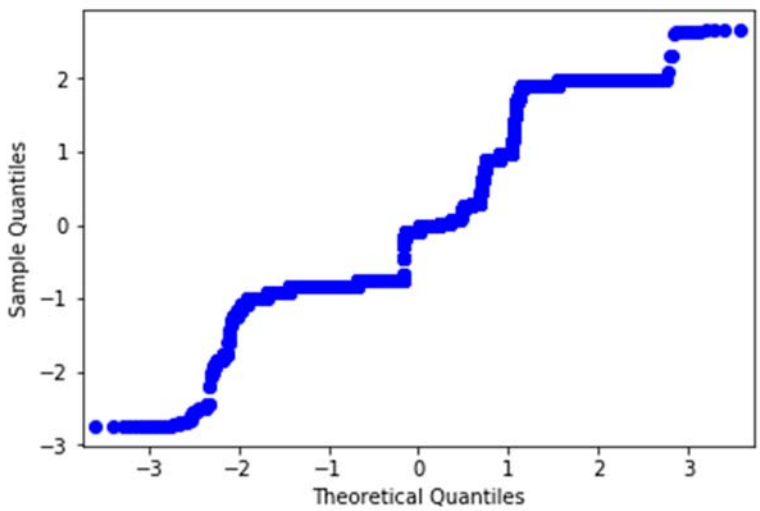
In [337]:
```
# Calculcate the accuracy score for both the train and test data
score_train_GLM = 1-(sum(abs(y_train-predict_GLM2.round()))/y_train.count())
score_test_GLM = 1-(sum(abs(y_test-predict_GLM2_test.round()))/y_train.count
())
print("Train accuracy score:",score_train_GLM,",","Test accuracy score:", scor
e_test_GLM)
```

```
Train accuracy score: 0.8348181446669392 , Test accuracy score: 0.92006538618
71679
```

In [338]:
```
# Evaluate residuals for train using qqplot
sm.qqplot(results_GLM2.resid_response, fit=True)
plt.show()
```



In [339]:
```
# Evaluate residuals for test using qqplot
sm.qqplot((y_test-predict_GLM2_test), fit=True)
plt.show()
```



In [340]:
```
# Run confusion matrix on test data to show false successes and false failures
train_confusion = pd.DataFrame(
    confusion_matrix(y_train, predict_GLM2.round()),
    columns=['Predicted Success', 'Predicted Failure'],
    index=['True Success', 'True Failure']
)

train_confusion
```

Out[340]:

	Predicted Success	Predicted Failure
True Success	5865	244
True Failure	1777	4349

In [341]:
```
# Run confusion matrix on test data to show false successes and false failures
test_confusion = pd.DataFrame(
    confusion_matrix(y_test, predict_GLM2_test.round()),
    columns=['Predicted Success', 'Predicted Failure'],
    index=['True Success', 'True Failure']
)

test_confusion
```

Out[341]:

	Predicted Success	Predicted Failure
True Success	2919	109
True Failure	869	2130

2. Random Forest

In [342]:
```
# Drop constant from X
X_train = X_train.drop(['const'],axis=1)
X_test = X_test.drop(['const'],axis=1)
```

In [343]:
```
# Fit and run the random forest classifier using selection of independent vari
ables based on the sqrt
# of the total available indpendent variables
rf = RFC( max_features = 'sqrt')
rf = rf.fit( X_train, y_train )
predict_rf = rf.predict( X_train )
```

```
C:\Users\Tori\AppData\Local\Continuum\anaconda3\lib\site-packages\sklearn\ens
emble\forest.py:246: FutureWarning: The default value of n_estimators will ch
ange from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [344]:
```
# Showing the most important independent variable in the RFC model
feature_importances = pd.DataFrame(
    rf.feature_importances_,
    index = X_train.columns,
    columns = [ 'y' ]
).sort_values( 'y', ascending = False )

print( feature_importances )
```

```
                      y
Cum_Wins       0.584728
Cum_Elections  0.368384
Democratic     0.025163
Republican     0.021725
```

In [345]:
```
# Calculate the accurarcy score for the train data
score_train_rf = rf.score( X_train, y_train )

# Calculate the accurarcy score for the test data
predict_rf_test = rf.predict( X_test )
score_test_rf = rf.score( X_test , y_test )

print("Train accuracy score:",score_train_rf,",","Test accuracy score:", score
_test_rf)
```

```
Train accuracy score: 0.8373518594196976 , Test accuracy score: 0.83872573419
61175
```

In [346]:
```
# Calculate the MSE for train
mse_train_rf = sk.metrics.mean_squared_error(y_train,predict_rf)

# Calculate the MSE for test
mse_test_rf = sk.metrics.mean_squared_error(y_test,predict_rf_test)
print("Train MSE:", mse_train_rf,",","Test MSE:",mse_test_rf)
```

```
Train MSE: 0.1626481405803024 , Test MSE: 0.16127426580388252
```

In [347]:
```
# Run confusion matrix on train data to show false successes and false failure
s

train_confusion = pd.DataFrame(
    confusion_matrix(y_train, predict_rf),
    columns=['Predicted Success', 'Predicted Failure'],
    index=['True Success', 'True Failure']
)

train_confusion
```

Out[347]:

	Predicted Success	Predicted Failure
True Success	5841	268
True Failure	1722	4404

In [348]:
```
# Run confusion matrix on test data to show false successes and false failures

test_confusion = pd.DataFrame(
    confusion_matrix(y_test, predict_rf_test),
    columns=['Predicted Success', 'Predicted Failure'],
    index=['True Success', 'True Failure']
)

test_confusion
```

Out[348]:

	Predicted Success	Predicted Failure
True Success	2899	129
True Failure	843	2156

3. Naive Bayes

In [349]:
```
# Run and fit Bernoulli Naive Bayes
clf = BNB()
clf.fit(X_train, y_train)
```

Out[349]: BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)

In [350]:
```
# Calculate the accurarcy score for the train data
predict_nb = clf.predict(X_train)
score_train_nb = clf.score(X_train,y_train)

# Calculate the accurarcy score for the test data
predict_nb_test = clf.predict(X_test)
score_test_nb = clf.score(X_test,y_test)

print("Train accuracy score:",score_train_nb,",","Test accuracy score:", score
_test_nb)
```

```
Train accuracy score: 0.8281160604822231 , Test accuracy score: 0.83208893313
42293
```

In [351]:
```
# Calculate the MSE for train
mse_train_nb = sk.metrics.mean_squared_error(y_train,predict_nb)

# Calculate the MSE for train
mse_test_nb = sk.metrics.mean_squared_error(y_test,predict_nb_test)


print("Train MSE:",mse_train_nb,",","Test MSE:", mse_test_nb)
```

```
Train MSE: 0.17188393951777686 , Test MSE: 0.1679110668657707
```

In [352]:
```
# Run confusion matrix on train data to show false successes and false failure
s

train_confusion = pd.DataFrame(
    confusion_matrix(y_train, predict_nb),
    columns=['Predicted Success', 'Predicted Failure'],
    index=['True Success', 'True Failure']
)

train_confusion
```

Out[352]:

	Predicted Success	Predicted Failure
True Success	5893	216
True Failure	1887	4239

In [353]:
```
# Run confusion matrix on test data to show false successes and false failures

test_confusion = pd.DataFrame(
    confusion_matrix(y_test, predict_nb_test),
    columns=['Predicted Success', 'Predicted Failure'],
    index=['True Success', 'True Failure']
)

test_confusion
```

Out[353]:

	Predicted Success	Predicted Failure
True Success	2935	93
True Failure	919	2080

Evaluation

In [354]:
```
# Store the accuracy scores and MSE for each model
GLM_score = ['Logistic Regression',score_train_GLM, score_test_GLM,mse_GLM_tes
t]
RF_score = ['Random Forest Classifier',score_train_rf, score_test_rf,mse_test_
rf]
NB_score = ['Naive Bayes',score_train_nb, score_test_nb,mse_test_nb]

# Display the comparative accuracy score and MSE test results for each model
results = pd.DataFrame([GLM_score,RF_score, NB_score],columns=['Model','Score
 = Train Data','Score = Test','MSE = Test'])
results = results.set_index(['Model'])
results
```

Out[354]:

Model	Score = Train Data	Score = Test	MSE = Test
Logistic Regression	0.834818	0.920065	0.132919
Random Forest Classifier	0.837352	0.838726	0.161274
Naive Bayes	0.828116	0.832089	0.167911

Preamble: Module Imports

In [0]:
```
import numpy as np
import pandas as pd
from sklearn.linear_model import LogisticRegression
#from sklearn.cross_validation import cross_val_score, cross_val_predict
from sklearn import metrics
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import BernoulliNB
from sklearn.metrics import confusion_matrix
from sklearn.metrics import mean_squared_error
from pandas import DataFrame,Series
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="ticks", color_codes=True)
%matplotlib inline
```

Loading, Reading, and Describing the Data

In [6]:

```
#Reading the csv file into a dataframe 'senda' and calling head to see the top
 5 rowd of data.
senda = pd.read_csv('1976-2016-senate-ks.csv')
senda.head()
```

Out[6]:

	year	state	state_po	state_fips	state_cen	state_ic	office	district	stage	special	can
0	1976	Arizona	AZ	4	86	61	US Senate	statewide	gen	False	De
1	1976	Arizona	AZ	4	86	61	US Senate	statewide	gen	False	
2	1976	Arizona	AZ	4	86	61	US Senate	statewide	gen	False	Bo
3	1976	Arizona	AZ	4	86	61	US Senate	statewide	gen	False	
4	1976	Arizona	AZ	4	86	61	US Senate	statewide	gen	False	M F

In [7]:
```
# Running descriptive statistics on the dataframe
senda.describe()
```

Out[7]:

	year	state_fips	state_cen	state_ic	candidatevotes	totalvotes	
count	3270.000000	3270.000000	3270.000000	3270.000000	3.270000e+03	3.270000e+03	3270.
mean	1997.975535	28.856575	53.183180	39.208563	3.972308e+05	2.165188e+06	0.
std	12.254840	15.459612	26.003031	22.700531	7.550154e+05	2.101348e+06	0.
min	1976.000000	1.000000	11.000000	1.000000	1.000000e+00	1.000000e+00	0.
25%	1988.000000	17.000000	33.000000	21.000000	4.726250e+03	6.450260e+05	0.
50%	2000.000000	29.000000	54.000000	41.000000	5.508500e+04	1.526782e+06	0.
75%	2010.000000	41.000000	74.000000	56.000000	4.733212e+05	2.743023e+06	0.
max	2016.000000	56.000000	95.000000	82.000000	7.864624e+06	1.257851e+07	1.

In [8]:
```
senda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3270 entries, 0 to 3269
Data columns (total 18 columns):
year                3270 non-null int64
state               3270 non-null object
state_po            3270 non-null object
state_fips          3270 non-null int64
state_cen           3270 non-null int64
state_ic            3270 non-null int64
office              3270 non-null object
district            3270 non-null object
stage               3270 non-null object
special             3270 non-null bool
candidate           3083 non-null object
party               2755 non-null object
writein             3270 non-null bool
candidatevotes      3270 non-null int64
totalvotes          3270 non-null int64
CandidateVotes %    3270 non-null object
Win                 3270 non-null int64
version             3270 non-null int64
dtypes: bool(2), int64(8), object(8)
memory usage: 415.2+ KB
```

Data Cleaning

In [0]:
```
#Dropping columns with repetitve or unneccasary data
senda.drop(columns = ['state_po','version','state','state_cen', 'state_ic', 'o
ffice','district','stage','special','writein'], axis = 1, inplace = True)
```

In [10]:
```
#Adding NaN as a replacement for missing data in the candidate column
senda['candidate'].replace('',np.nan)
```

Out[10]:

```
0                 Dennis DeConcini
1                      Sam Steiger
2                        Bob Field
3                    Allan Norwitz
4             Wm. Mathews Feighan
5              S. I. (Sam) Hayakawa
6                   John V. Tunney
7                       David Wald
8                       Jack McCoy
9                       Omari Musa
10           Lowell P. Weicker, Jr.
11                 Gloria Schaffer
12                Robert Barnabei
13                         scatter
14            William V. Roth, Jr.
15               Thomas C. Maloney
16                 Donald G. Gies
17             Joseph F. McInerney
18              John A. Massimilla
19                   Lawton Chiles
20                      John Grady
21                         scatter
22              Spark M. Matsunaga
23                  William Quinn
24               Anthony N. Hodges
25                 James D. Kimmel
26                  Rockne Johnson
27                Richard G. Lugar
28                   Vance Hartke
29                      Don L. Lee
                ...
3240         Edward T. Clifford III
3241                      Tim Scott
3242                  Thomas Dixon
3243                  Thomas Dixon
3244                  Bill Bledsoe
3245                  Thomas Dixon
3246                  Bill Bledsoe
3247     Rebel Michael Scarborough
3248                            NaN
3249                 John R. Thune
3250                 Jay Williams
3251                     Mike Lee
3252                Misty K. Snow
3253                 Stoney Fonua
3254                  Bill Barron
3255              Patrick J. Leahy
3256                  Scott Milne
3257                 Cris Ericson
3258                   Blank Vote
3259                Jerry Trudell
3260             Pete Diamondstone
3261                    Void Vote
3262                          NaN
3263                 Patty Murray
3264                  Chris Vance
3265                  Ron Johnson
```

```
3266                 uss Feingold
3267           Philip N. Anderson
3268                      scatter
3269                 John Schiess
Name: candidate, Length: 3270, dtype: object
```

In [11]:
```
#Adding NaN as a replacement for missing data, in the party column
senda['party'].replace('',np.nan)
```

Out[11]: 0 democrat

1. republican
2. independent
3. libertarian
4. independent
5. republican
6. democrat
7. peace and freedom
8. american independent
9. independent
10. republican
11. democrat
12. american independent
13. NaN
14. republican
15. democrat
16. american
17. none
18. prohibition
19. democrat
20. republican
21. NaN
22. democrat
23. republican
24. prohibition
25. NaN
26. libertarian
27. republican
28. democrat
29. NaN

```
                        ...
3240              libertarian
3241               republican
3242                 democrat
3243         working families
3244              libertarian
3245                    green
3246             constitution
3247                 american
3248                      NaN
3249               republican
3250                 democrat
3251               republican
```

```
3252                   democrat
3253       independent american
3254                       none
3255                   democrat
3256                 republican
3257    united states marijuana
3258                        NaN
3259                independent
3260              liberty union
3261                        NaN
3262                        NaN
3263                   democrat
3264                 republican
3265                 republican
```

```
3266                     democrat
3267                  libertarian
3268                          NaN
3269                   republican
Name: party, Length: 3270, dtype: object
```

In [0]:
```
#Dropping all missing columns from across the dataset
senda.dropna(inplace = True)
```

In [0]:
```
#Updating the Column name to make it more easier to process
senda = senda.rename({'CandidateVotes %':'CandidateVotes_Pct'}, axis='columns'
)
```

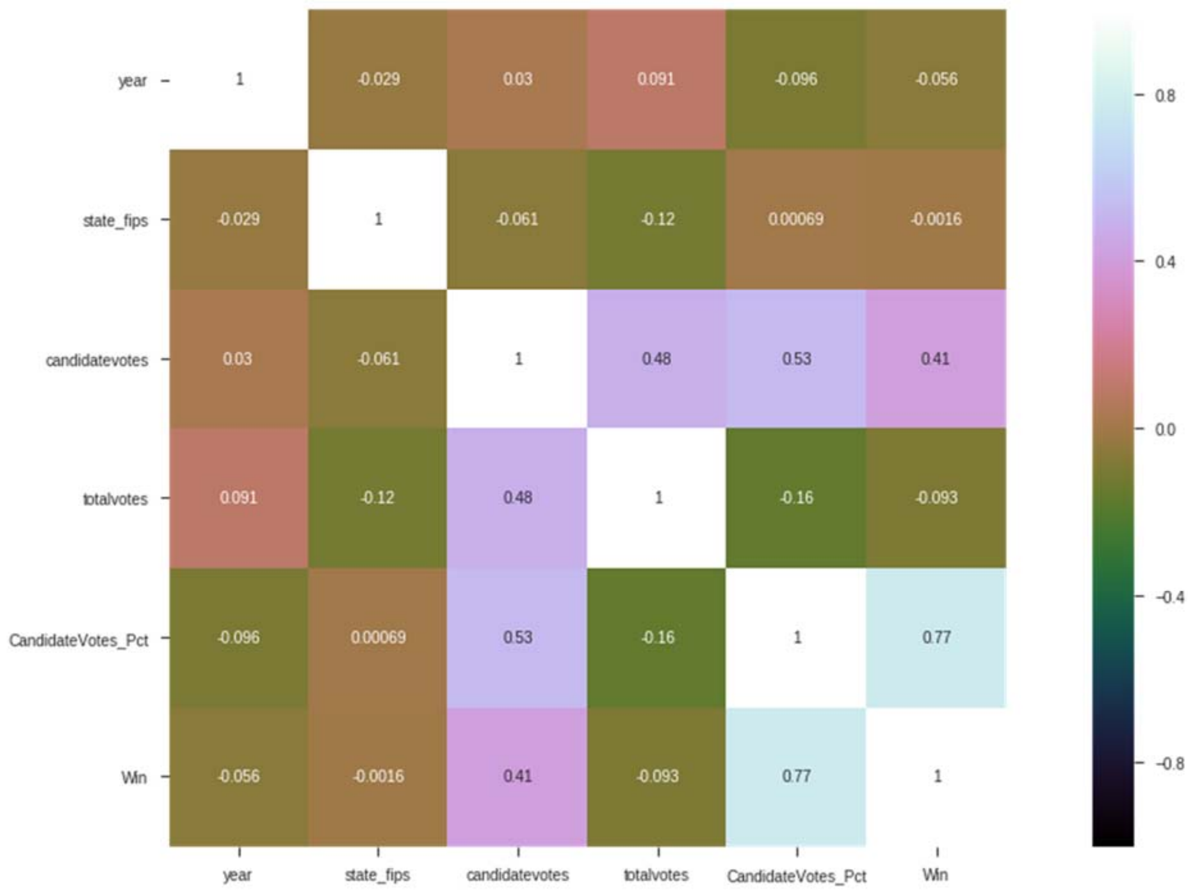
In [0]:
```
#Updating CandidateVotes_Pct from datatype object to datatype integer
senda['CandidateVotes_Pct'] = senda.CandidateVotes_Pct.str.extract('(\d+)', ex
pand=True).astype(int)
```

In [16]:
```
#Checking for any missing values across the updated dataset
senda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2755 entries, 0 to 3269
Data columns (total 8 columns):
year                  2755 non-null int64
state_fips            2755 non-null int64
candidate             2755 non-null object
party                 2755 non-null object
candidatevotes        2755 non-null int64
totalvotes            2755 non-null int64
CandidateVotes_Pct    2755 non-null int64
Win                   2755 non-null int64
dtypes: int64(6), object(2)
memory usage: 193.7+ KB
```

In [17]:
```
# Creating a Correllation matrix, using the numerical data only
corr_mat=senda.corr(method='pearson')
plt.figure(figsize=(20,10))
sns.heatmap(corr_mat,vmax=1,square=True,annot=True,cmap='cubehelix')
```

Out[17]: <matplotlib.axes._subplots.AxesSubplot at 0x7f02b6990048>



Feature Engineering

In [0]:
```
# Creating a new column, Total Elections Entered, by using the data from the C
andidate column
senda['Total_Elections_Entered'] = senda.groupby('candidate')['candidate'].tra
nsform(lambda x: x.count())
```

In [0]:
```
# Creating a new column, Total Elections Wins, by using the data from columsn
 Candidate and Win
senda['Total_Elections_Wins'] = senda.groupby('candidate')['Win'].transform(la
mbda x: x.sum())
```

In [20]:
```
# List of columns, along with datatypes, after creating two new columns from e
xisting data
senda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2755 entries, 0 to 3269
Data columns (total 10 columns):
year                       2755 non-null int64
state_fips                 2755 non-null int64
candidate                  2755 non-null object
party                      2755 non-null object
candidatevotes             2755 non-null int64
totalvotes                 2755 non-null int64
CandidateVotes_Pct        2755  non-null int64
Win                        2755 non-null int64
Total_Elections_Entered  2755  non-null  int64
Total_Elections_Wins   2755   non-null   int64
dtypes: int64(8), object(2)
memory usage: 236.8+ KB
```

In [21]:
```
# Removing any candidates that won less than 10% of votes; this helps to furth
er clean the data
senda.loc[lambda senda: senda.CandidateVotes_Pct >= 10,:]
```

Out[21]:

	year	state_fips	candidate	party	candidatevotes	totalvotes	CandidateVotes_Pct	Wi
0	1976	4	Dennis DeConcini	democrat	400334	741210	54	
1	1976	4	Sam Steiger	republican	321236	741210	43	
5	1976	6	S. I. (Sam) Hayakawa	republican	3748973	7470586	50	
6	1976	6	John V. Tunney	democrat	3502862	7470586	46	
10	1976	9	Lowell P. Weicker, Jr.	republican	785683	1361666	57	
11	1976	9	Gloria Schaffer	democrat	561018	1361666	41	
14	1976	10	William V. Roth, Jr.	republican	125454	224795	55	
15	1976	10	Thomas C. Maloney	democrat	98042	224795	43	
19	1976	12	Lawton Chiles	democrat	1799518	2857534	62	
20	1976	12	John Grady	republican	1057886	2857534	37	
22	1976	15	Spark M. Matsunaga	democrat	162305	302092	53	
23	1976	15	William Quinn	republican	122724	302092	40	
27	1976	18	Richard G. Lugar	republican	1275833	2161187	59	
28	1976	18	Vance Hartke	democrat	868522	2161187	40	
31	1976	23	Edmund S. Muskie	democrat	292704	486193	60	
32	1976	23	Robert A. G. Monks	republican	193489	486193	39	
33	1976	24	Paul S. Sarbanes	democrat	772101	1365290	56	
34	1976	24	J. Glenn Beall, Jr.	republican	530439	1365290	38	
36	1976	25	Edward M. Kennedy	democrat	1726657	2491255	69	
37	1976	25	Michael S. Robertson	republican	722641	2491255	29	
41	1976	26	Donald W. Riegle, Jr.	democrat	1831031	3490412	52	
42	1976	26	Marvin L. Esch	republican	1635087	3490412	46	
48	1976	27	Hubert H. Humphrey	democrat	1290736	1912020	67	

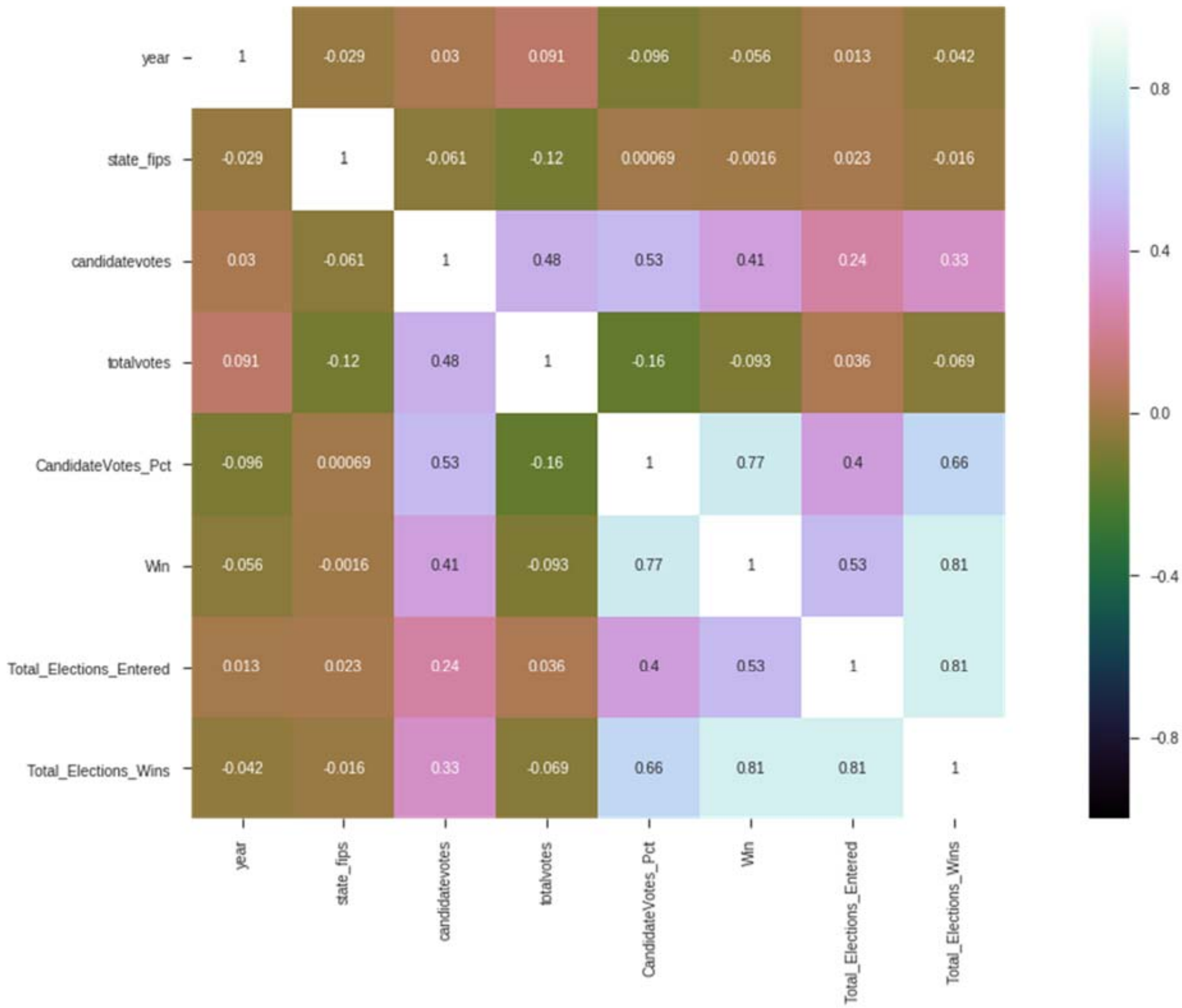
	year	state_fips	candidate	party	candidatevotes	totalvotes	CandidateVotes_Pct	Wi
49	1976	27	Jerry Brekke	republican	478602	1912020	25	
54	1976	28	John C. Stennis	democrat	554433	554433	100	
55	1976	29	John C. Danforth	republican	1090067	1914460	56	
56	1976	29	Warren E. Hearnes	democrat	813571	1914460	42	
58	1976	30	John Melcher	democrat	206232	321445	64	
59	1976	30	Stanley C. Burger	republican	115213	321445	35	
60	1976	31	Edward Zorinsky	democrat	313805	593310	52	
...	...	...	...	...	...	...	...	.
3188	2016	32	Catherine Cortez Masto	democrat	521994	1108294	47	
3189	2016	32	Joseph J. Heck	republican	495079	1108294	44	
3195	2016	33	Maggie Hassan	democrat	354649	739140	47	
3196	2016	33	Kelly Ayotte	republican	353632	739140	47	
3200	2016	36	Charles E. Schumer	democrat	4784220	7800725	61	
3201	2016	36	Wendy Long	republican	1723927	7800725	22	
3212	2016	37	Richard Burr	republican	2395376	4691133	51	
3213	2016	37	Deborah K. Ross	democrat	2128165	4691133	45	
3215	2016	38	John Hoeven	republican	268788	342501	78	
3216	2016	38	Eliot Glassheim	democrat	58116	342501	16	
3220	2016	39	Rob Portman	republican	3118567	5374164	58	
3221	2016	39	Ted Strickland	democrat	1996908	5374164	37	
3226	2016	40	James Lankford	republican	980892	1448047	67	
3227	2016	40	Mike Workman	democrat	355911	1448047	24	
3231	2016	41	Ron Wyden	democrat	1105119	1952478	56	

	year	state_fips	candidate	party	candidatevotes	totalvotes	CandidateVotes_Pct	Wi
3232	2016	41	Mark Callahan	republican	651106	1952478	33	
3238	2016	42	Patrick J. Toomey	republican	2951702	6051856	48	
3239	2016	42	Katie McGinty	democrat	2865012	6051856	47	
3241	2016	45	Tim Scott	republican	1241609	2049893	60	
3242	2016	45	Thomas Dixon	democrat	704540	2049893	34	
3249	2016	46	John R. Thune	republican	265516	369656	71	
3250	2016	46	Jay Williams	democrat	104140	369656	28	
3251	2016	49	Mike Lee	republican	760220	1115583	68	
3252	2016	49	Misty K. Snow	democrat	301858	1115583	27	
3255	2016	50	Patrick J. Leahy	democrat	192243	320467	59	
3256	2016	50	Scott Milne	republican	103637	320467	32	
3263	2016	53	Patty Murray	democrat	1913979	3243317	59	
3264	2016	53	Chris Vance	republican	1329338	3243317	40	
3265	2016	55	Ron Johnson	republican	1479471	2948741	50	
3266	2016	55	Russ Feingold	democrat	1380335	2948741	46	

1442 rows × 10 columns

In [22]:
```
# Creating a Correllation matrix, after data cleaning
corr_mat=senda.corr(method='pearson')
plt.figure(figsize=(20,10))
sns.heatmap(corr_mat,vmax=1,square=True,annot=True,cmap='cubehelix')
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x7f02b697b518>



In [0]:
```
# Because the party column is a categorical data column, we are going to use O
ne-Hot encoding to convert it to
# integer, either 0 or 1.
senda_two = pd.get_dummies(senda['party'])
```

In [0]:
```
# Add the one-hot encoded party column back to the 'senda' dataset.
senda = pd.concat([senda, senda_two], axis = 1)
senda.head()
```

Out[0]:

	year	state_fips	candidate	party	candidatevotes	totalvotes	CandidateVotes_Pct	Win
0	1976	4	Dennis DeConcini	democrat	400334	741210	54	1
1	1976	4	Sam Steiger	republican	321236	741210	43	0
2	1976	4	Bob Field	independent	10765	741210	1	0
3	1976	4	Allan Norwitz	libertarian	7310	741210	0	0
4	1976	4	Wm. Mathews Feighan	independent	1565	741210	0	0

5 rows × 166 columns

In [0]:
```
senda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2755 entries, 0 to 3269
Columns: 322 entries, year to working families
dtypes: int64(8), object(2), uint8(312)
memory usage: 1.1+ MB
```

In [0]:

Preparing dataset for Modelling

In [0]:
```
# Transfer the dependent variable, 'Win', to a dataframe senda_target
senda_target = senda['Win']
senda_target.head()
```

Out[0]:
```
0    1
1    0
2    0
3    0
4    0
Name: Win, dtype: int64
```

In [0]:
```
# Drop a few other columns that will not add any major value to the modelling
 process
senda.drop(columns = ['candidate','party','candidatevotes','totalvotes', 'Cand
idateVotes_Pct', 'Win'], axis = 1, inplace = True)
```

In [0]:
```
senda.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2755 entries, 0 to 3269
Columns: 166 entries, year to working families
dtypes: int64(8), object(2), uint8(156)
memory usage: 656.5+ KB
```

In [0]:
```
```

Modelling Dataset

Model A: Logistic Regression

In [0]:
```
# Splitting the dataset into train and test using train_test_split from sklear
n

X_train, X_test, y_train, y_test = train_test_split(senda, senda_target, test_
size=0.33, random_state=42)

print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(1845, 160) (1845,)
(910, 160) (910,)
```

In [0]:
```
# Create Logistic regression object
lr = LogisticRegression(random_state=0, solver='lbfgs',multi_class='multinomia
l')
```

In [0]:
```
#fit the model using dependent and independent variables from the training dat
aset
lr.fit( X_train, y_train)
```

Out[0]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='multinomial',
          n_jobs=1, penalty='l2', random_state=0, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

In [0]:
```
# Score the model using the training data
score_LogRegression_train = lr.score( X_train, y_train )
print(score_LogRegression_train)
```

```
0.9739837398373984
```

In [0]:
```
# Running confusion matrix on train data to show false successes and false fai
lures

pd.DataFrame (
    confusion_matrix( y_train, lr.predict( X_train ) ),
    columns = [ 'Predicted Success', 'Predicted Failure' ],
    index = [ 'True Success', 'True Failure' ]
)
```

Out[0]:

	Predicted Success	Predicted Failure
True Success	1347	18
True Failure	30	450

In [0]:
```
#Predicting the dependent variable, by running the test data
y_pred = lr.predict( X_test )
```

In [0]:
```
# Determining the accuracy of applying the random forest model on the test dat
a

score_LogRegression_test = lr.score( X_test, y_test )
print(score_LogRegression_test)
```

```
0.9802197802197802
```

In [0]:
```
# Running confusion matrix on test data to show false successes and false fail
ures
pd.DataFrame (
    confusion_matrix( y_test, y_pred ),
    columns = [ 'Predicted Success', 'Predicted Failure' ],
    index = [ 'True Success', 'True Failure' ]
)
```

Out[0]:

	Predicted Success	Predicted Failure
True Success	670	6
True Failure	12	222

In [0]:
```
# The mean squared error
Mean_Squared_Error_LogRegression = mean_squared_error(y_test, y_pred)
print("Mean squared error: %.2f"% Mean_Squared_Error_LogRegression)
```

```
Mean squared error: 0.02
```

Model B: Random Forest Classifier

In [0]:
```
# Splitting the dataset into train and test using train_test_split from sklear
n

X_train, X_test, y_train, y_test = train_test_split(senda, senda_target, test_
size=0.33, random_state=42)

print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(1845, 160) (1845,)
(910, 160) (910,)
```

In [0]:
```
# Create Random Forest Classifier Object called rf
rf = RandomForestClassifier( max_features = 'sqrt' )
```

In [0]:
```
#fit the model using dependent and independent variables from the training dat
aset
rf.fit( X_train, y_train)
```

Out[0]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='sqrt', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [0]:
```
# Showing the most important x variables/features in the model
feature_importances = pd.DataFrame(
    rf.feature_importances_,
    index = X_train.columns,
    columns = [ 'Win' ]
).sort_values( 'Win', ascending = False )

print( feature_importances )
```

```
                                            Win
                Total_Elections_Wins   0.605319
             Total_Elections_Entered   0.186105
                            democrat   0.045166
                          state_fips   0.042345
                                year   0.040927
                          republican   0.018917
                         libertarian   0.013491
                    working families   0.007319
                         independent   0.007281
                        conservative   0.004246
                               green   0.004120
                             liberal   0.003856
                   socialist workers   0.002959
                        independence   0.002336
                       liberty union   0.001556
                     u.s. taxpayers    0.001430
                            american   0.001286
                       workers world   0.001273
                         natural law   0.001197
                              reform   0.000961
                                none   0.000945
                        constitution   0.000820
                       right to life   0.000573
democrat (not identified on  ballot)   0.000550
           connecticut for lieberman   0.000496
                         prohibition   0.000432
                        god we trust   0.000422
                american independent   0.000415
                           new union   0.000392
                        new alliance   0.000370
                                 ...        ...
              american shopping party  0.000000
         american constitution party   0.000000
                            i.d.e.a.   0.000000
                  independence fusion  0.000000
               poor people's campaign  0.000000
                    independent green  0.000000
                            petition   0.000000
                     personal choice   0.000000
                perot's independents   0.000000
                 people before profits 0.000000
                 peace and prosperity  0.000000
                             patriot   0.000000
                        pacific green  0.000000
                             pacific   0.000000
                         nonpartisan   0.000000
                           no slogan   0.000000
                       nebraska party  0.000000
                   alaska libertarian  0.000000
national democratic party of  alabama  0.000000
        minnesota open progressives    0.000000
                     marijuana reform  0.000000
                alaskan independence   0.000000
                        la raza unida  0.000000
                   keep america first  0.000000
                             justice   0.000000
```

```
      jersey strong independents        0.000000
              independent reform        0.000000
    independent progressive line        0.000000
  independent party of delaware         0.000000
              national statesman        0.000000

            [160 rows x 1 columns]
```

In [0]:
```
# Score the model using the training data
score_Random_Forest_train = rf.score( X_train, y_train )
print(score_Random_Forest_train)
```

```
0.9978319783197832
```

In [0]:
```
# Running confusion matrix on train data to show false successes and false failures

pd.DataFrame (
    confusion_matrix( y_train, rf.predict( X_train ) ),
    columns = [ 'Predicted Success', 'Predicted Failure' ],
    index = [ 'True Success', 'True Failure' ]
)
```

Out[0]:

	Predicted Success	Predicted Failure
True Success	1362	3
True Failure	1	479

In [0]:
```
#Predicting the dependent variable, by running the test data
y_pred = rf.predict( X_test )
```

In [0]:
```
# Determining the accuracy of applying the random forest model on the test data

score_Random_Forest_test = rf.score( X_test, y_test )
print(score_Random_Forest_test)
```

```
0.9791208791208791
```

In [0]:
```
# Running confusion matrix on test data to show false successes and false failures
pd.DataFrame (
    confusion_matrix( y_test, y_pred ),
    columns = [ 'Predicted Success', 'Predicted Failure' ],
    index = [ 'True Success', 'True Failure' ]
)
```

Out[0]:

	Predicted Success	Predicted Failure
True Success	663	13
True Failure	6	228

In [0]:
```
# The mean squared error
Mean_Squared_Error_Random_Forest = mean_squared_error(y_test, y_pred)
print("Mean squared error: %.2f"% Mean_Squared_Error_Random_Forest)
```

```
Mean squared error: 0.02
```

Model C: Bernoulli Naive Bayes

In [0]:
```
# Splitting the dataset into train and test using train_test_split from sklear
n

X_train, X_test, y_train, y_test = train_test_split(senda, senda_target, test_
size=0.33, random_state=42)

print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)
```

```
(1845, 160) (1845,)
(910, 160) (910,)
```

In [0]:
```
# Create Bernoulli Naive Bayes object
bnb = BernoulliNB()
```

In [0]:
```
# Fit the independent and dependent variable to the model
bnb.fit(X_train, y_train)
```

Out[0]:
```
BernoulliNB(alpha=1.0, binarize=0.0, class_prior=None, fit_prior=True)
```

In [0]:
```
# Score the model using the traing data
score_BernoulliNB_train = bnb.score( X_train, y_train )
print(score_BernoulliNB_train)
```

```
0.9707317073170731
```

In [0]:
```
# Running confusion matrix on train data to show false successes and false fai
lures

pd.DataFrame (
    confusion_matrix( y_train, bnb.predict( X_train ) ),
    columns = [ 'Predicted Success', 'Predicted Failure' ],
    index = [ 'True Success', 'True Failure' ]
)
```

Out[0]:

	Predicted Success	Predicted Failure
True Success	1313	52
True Failure	2	478

In [0]:
```
#Predicting the dependent variable, by running the test data
y_pred = bnb.predict( X_test )
```

In [0]:
```
# Determining the accuracy of applying the random forest model on the test dat
a
score_BernoulliNB_test = bnb.score( X_test, y_test )
print(score_BernoulliNB_test)
```

```
0.9692307692307692
```

In [0]:
```
# Running confusion matrix on test data to show false successes and false fail
ures
pd.DataFrame (
    confusion_matrix( y_test, y_pred ),
    columns = [ 'Predicted Success', 'Predicted Failure' ],
    index = [ 'True Success', 'True Failure' ]
)
```

Out[0]:

	Predicted Success	Predicted Failure
True Success	650	26
True Failure	2	232

In [0]:
```
# The mean squared error
Mean_Squared_Error_BernoulliNB = mean_squared_error(y_test, y_pred)
print("Mean squared error: %.2f"% Mean_Squared_Error_BernoulliNB)
```

```
Mean squared error: 0.03
```

Model Comparison

In [0]:
```
score_test_data = [score_LogRegression_test,score_Random_Forest_test,score_Ber
noulliNB_test]
score_train_data = [score_LogRegression_train,score_Random_Forest_train,score_
BernoulliNB_train]
Mean_Squared_Error = [Mean_Squared_Error_LogRegression,Mean_Squared_Error_Rand
om_Forest,Mean_Squared_Error_BernoulliNB]

col = {'Score - Train Data':score_train_data,'Score - Test Data':score_test_da
ta,'Mean Squared Error - Test Data': Mean_Squared_Error}
models = ['Logistic Regression','Random Forest','Bernoulli Naive Bayes']
compare = DataFrame(data=col,index=models)
compare
```

Out[0]:

	Score - Train Data	Score - Test Data	Mean Squared Error - Test Data
Logistic Regression	0.973984	0.980220	0.019780
Random Forest	0.997832	0.979121	0.020879
Bernoulli Naive Bayes	0.970732	0.969231	0.030769

```
In [0]:
```