State Legislature Election Modeling

In this project I aim to create models able to predict the outcome of political campaigns for state legislative seats in all 50 states. Through the work below I created models that predict when a seat is likely to be a tossup. This is especially useful to political organizers who can use this information in order to prioritize resource allocation in favor of close elections.

But first, a definition of terms. 49 of 50 states have bicameral legislatures, i.e. a higher house with fewer members and a lower house with more. Nebraska stands alone as the only state with a unicameral legislature. Its legislature is treated as an upper house in the data I used in this project. Every other state calls their upper house the Senate.

However, for lower houses things get a bit more creative. Most states simply call them Houses of Representatives, mirroring the federal legislature. A number of states, mostly older east coast states, have more unique names such as Assemblies or Houses of Delegates. The differences are largely vestigial, harkening back to colonial governments

There are other similar political bodies included in the census data used, including the District of Columbia Council and Puerto Rico Legislative Assembly. The purpose of this project is simply to model state legislatures in US states and these other bodies have their own separate dynamics that aren't necessarily relevant."

In [1]:

```
Requirement already satisfied: censusdata in /usr/local/lib/python3.7/
dist-packages (1.15)
Requirement already satisfied: category_encoders in /usr/local/lib/pyt
hon3.7/dist-packages (2.3.0)
Requirement already satisfied: geopandas in /usr/local/lib/python3.7/d
ist-packages (0.10.2)
Requirement already satisfied: boruta in /usr/local/lib/python3.7/dist
-packages (0.3)
Requirement already satisfied: catboost in /usr/local/lib/python3.7/di
st-packages (1.0.3)
Requirement already satisfied: shap in /usr/local/lib/python3.7/dist-p
ackages (0.40.0)
Requirement already satisfied: requests in /usr/local/lib/python3.7/di
st-packages (from censusdata) (2.23.0)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist
-packages (from censusdata) (1.1.5)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.
7/dist-packages (from category_encoders) (1.4.1)
Requirement already satisfied: patsy>=0.5.1 in /usr/local/lib/python3.
7/dist-packages (from category_encoders) (0.5.2)
Requirement already satisfied: numpy>=1.14.0 in /usr/local/lib/python
3.7/dist-packages (from category_encoders) (1.19.5)
Requirement already satisfied: scikit-learn>=0.20.0 in /usr/local/lib/
python3.7/dist-packages (from category_encoders) (1.0.1)
Requirement already satisfied: statsmodels>=0.9.0 in /usr/local/lib/py
thon3.7/dist-packages (from category_encoders) (0.10.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.
7/dist-packages (from pandas->censusdata) (2018.9)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/li
b/python3.7/dist-packages (from pandas->censusdata) (2.8.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-pa
ckages (from patsy>=0.5.1->category_encoders) (1.15.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/
python3.7/dist-packages (from scikit-learn>=0.20.0->category_encoders)
(3.0.0)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.
7/dist-packages (from scikit-learn>=0.20.0->category_encoders) (1.1.0)
Requirement already satisfied: pyproj>=2.2.0 in /usr/local/lib/python
3.7/dist-packages (from geopandas) (3.2.1)
Requirement already satisfied: shapely>=1.6 in /usr/local/lib/python3.
7/dist-packages (from geopandas) (1.8.0)
Requirement already satisfied: fiona>=1.8 in /usr/local/lib/python3.7/
dist-packages (from geopandas) (1.8.20)
Requirement already satisfied: click-plugins>=1.0 in /usr/local/lib/py
thon3.7/dist-packages (from fiona>=1.8->geopandas) (1.1.1)
Requirement already satisfied: munch in /usr/local/lib/python3.7/dist-
packages (from fiona>=1.8->geopandas) (2.5.0)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/
dist-packages (from fiona>=1.8->geopandas) (57.4.0)
Requirement already satisfied: cligj>=0.5 in /usr/local/lib/python3.7/
dist-packages (from fiona>=1.8->geopandas) (0.7.2)
Requirement already satisfied: attrs>=17 in /usr/local/lib/python3.7/d
ist-packages (from fiona>=1.8->geopandas) (21.2.0)
Requirement already satisfied: click>=4.0 in /usr/local/lib/python3.7/
dist-packages (from fiona>=1.8->geopandas) (7.1.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dis
```

```
t-packages (from fiona>=1.8->geopandas) (2021.10.8)
Requirement already satisfied: plotly in /usr/local/lib/python3.7/dist
-packages (from catboost) (4.4.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.7/di
st-packages (from catboost) (0.10.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/
dist-packages (from catboost) (3.2.2)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python
```

In [2]:

```python
# enable text wrapping on Google Colab
from IPython.display import HTML, display

def set_css():
  display(HTML('''
  <style>
    pre {
        white-space: pre-wrap;
    }
  </style>
  '''))
get_ipython().events.register('pre_run_cell', set_css)

from warnings import simplefilter

# ignore all warnings
simplefilter(action='ignore')

import pandas as pd
import censusdata
import openpyxl
import re
import os
import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import *
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV, train_test_split, validation_curve
from sklearn.metrics import classification_report, plot_confusion_matrix, recall_score

from datetime import datetime

import category_encoders as ce

import geopandas as gpd

from boruta import BorutaPy

from imblearn.pipeline import Pipeline

import xgboost as xgb
```

```
import lightgbm as lgb
```

Data Exploration

I started with election result data gathered by Carl Klarner and available on online from [Harvard University (https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/3WZFK9&widget=dataverse@harvard)](https://dataverse.harvard.edu/dataset.xhtml?persistentId=doi:10.7910/DVN/3WZFK9&widget=dataverse@harvard). This dataset aims to capture every state-level legislative election from 1967 to 2016. I decided to limit the scope of this project to within the last 10 years for both technical and theoretical reasons. On the technical side, the way in which the Census Bureau presents their American Community Survey data products has changed since their earliest offering on their API in 2005, making it difficult to compare past and present data. In terms of theory, I subscribe to the notion that the ways in which demographics and politics interact are changing. [One example of this is the sharp turn towards the Democratic Party in American suburbs which used to be reliably Republican. (https://fivethirtyeight.com/features/why-the-suburbs-have-shifted-blue/)](https://fivethirtyeight.com/features/why-the-suburbs-have-shifted-blue/)

As a result of these two factors I fear including too much historical data could compromise my models' ability to understand the data in such a way as to usefully predict future election results.

In [3]:

```
harvard_data = pd.read_csv('/content/drive/MyDrive/capstone/harvard_data_2000+.csv', index_col=0)
```

Out[3]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
2825	2002	al	1	0	g	starkey, nelson jr.	democrat	inc	8540.0	w
2826	2002	al	1	0	g	franklin, joey	libertarian	none	304.0	l
2827	2002	al	1	0	g	mcnatt, william dale	modernrepublican	none	4772.0	l
2828	2002	al	1	0	g	writein	99999	none	28.0	l
2829	2002	al	2	0	g	pettus, mary	democrat	none	5718.0	l
...	...	...	...	...	...	...	...	...	...	...
378340	2016	wy	28	1	g	anderson, james lee	republican	inc	5216.0	w
378341	2016	wy	28	1	g	scattering	writein	none	22.0	l
378342	2016	wy	30	1	g	ford, robert	democrat	none	1521.0	l
378343	2016	wy	30	1	g	scott, charles k.	republican	inc	5831.0	w
378344	2016	wy	30	1	g	scattering	writein	none	45.0	l

142905 rows × 10 columns

In [4]:

Out[4]:

```
Index(['year', 'sab', 'ddez', 'sen', 'etype', 'cand', 'party', 'exper
', 'vote',
       'outcome'],
```

In [5]:

```
harvard_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 142905 entries, 2825 to 378344
Data columns (total 10 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   year     142905 non-null  int64
 1   sab      142905 non-null  object
 2   ddez     142905 non-null  object
 3   sen      142905 non-null  int64
 4   etype    142905 non-null  object
 5   cand     142905 non-null  object
 6   party    142748 non-null  object
 7   exper    142905 non-null  object
 8   vote     137078 non-null  float64
 9   outcome  142905 non-null  object
dtypes: float64(1), int64(2), object(7)
memory usage: 12.0+ MB
```

In [6]:

```
harvard_data.describe()
```

Out[6]:

	year	sen	vote
count	142905.000000	142905.000000	137078.000000
mean	2009.537287	0.225730	10098.985629
std	5.239091	0.418064	15806.774287
min	2000.000000	0.000000	0.000000
25%	2005.000000	0.000000	1583.250000
50%	2010.000000	0.000000	4885.000000
75%	2014.000000	0.000000	12522.000000
max	2017.000000	1.000000	326755.000000

In [7]:

```
harvard_data[(harvard_data['year'] == 2012) & (harvard_data['sab'] == 'ny') & (harvard_data['sen'] == 1)
 & (harvard_data['ddez'] == '1')
```

Out[7]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
246627	2012	ny	1	1	g	fleming, bridget m.	democrat	none	46783.0	l
246628	2012	ny	1	1	g	fleming, bridget m.	workingfamilies	none	4240.0	l
246629	2012	ny	1	1	g	lavalle, kenneth p.	independent	inc	4191.0	w

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
246630	2012	ny	1	1	g	lavalle, kenneth p.	conservative	inc	10462.0	w
246631	2012	ny	1	1	g	lavalle, kenneth p.	modernrepublican	inc	61130.0	w

In [8]:

```
harvard_data[(harvard_data['sab'] == 'al') & (harvard_data['sen'] == 0)
 & (harvard_data['ddez'] == '1')
```

Out[8]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
2825	2002	al	1	0	g	starkey, nelson jr.	democrat	inc	8540.0	w
2826	2002	al	1	0	g	franklin, joey	libertarian	none	304.0	l
2827	2002	al	1	0	g	mcnatt, william dale	modernrepublican	none	4772.0	l
2828	2002	al	1	0	g	writein	99999	none	28.0	l
3105	2006	al	1	0	g	irons, tammy	democrat	none	8410.0	w
3106	2006	al	1	0	g	smith, william	modernrepublican	none	4511.0	l
3257	2010	al	1	0	g	burdine, greg	democrat	none	7083.0	w
3258	2010	al	1	0	g	hanson, quinton	modernrepublican	none	6877.0	l
3259	2010	al	1	0	g	writein	independentwritein	none	14.0	l
3517	2014	al	1	0	dpfsettled	burdine, greg	democrat	inc	NaN	w
3518	2014	al	1	0	rpfsettled	pettus, phillip	modernrepublican	none	1172.0	w
3519	2014	al	1	0	rpfsettled	statom, sterling (josh)	modernrepublican	none	909.0	l
3727	2014	al	1	0	g	burdine, greg	democrat	inc	4652.0	l
3728	2014	al	1	0	g	pettus, phillip	modernrepublican	none	4933.0	w
3729	2014	al	1	0	g	scattering	9999	none	10.0	l

In [9]:

```
dupes = harvard_data[(harvard_data['cand'] != 'scattering') & (harvard_data['etype'] == 'g')&(harvard_data.drop(['vote', 'party'], axis=1).duplicated(keep=False))].dropna(subset=['vote'])
```

Out[9]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
3730	2014	al	2	0	g	betterton, andew (andy)	democrat	none	549.0	l

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
3731	2014	al	2	0	g	betterton, andew (andy)	democrat	none	4675.0	l
3732	2014	al	2	0	g	greer, lynn	modernrepublican	inc	1428.0	w
3733	2014	al	2	0	g	greer, lynn	modernrepublican	inc	7133.0	w
3736	2014	al	3	0	g	black, marcel	democrat	inc	262.0	w
...	...	...	...	...	...	...	...	...	...	...
378322	2016	wy	20	1	g	agar, wyatt	republican	none	1225.0	w
378323	2016	wy	20	1	g	agar, wyatt	republican	none	1817.0	w
378324	2016	wy	20	1	g	agar, wyatt	republican	none	3157.0	w
378330	2016	wy	22	1	g	kinskey, dave	republican	none	3746.0	w
378331	2016	wy	22	1	g	kinskey, dave	republican	none	3857.0	w

In [10]:

Out[10]:

```
ny    7839
ct    1590
va    1046
me    1021
in     929
il     914
wv     884
nc     747
nj     735
ky     715
Name: sab, dtype: int64
```

In [11]:

Out[11]:

```
47
```

The data as collected presents a number of issue's we have to overcome. First, individual rows appear to reflect individual candidates but split between their vote totals in different counties or in the case of fusion voting in New York, accross different parties. Rows will have to be combined in order to reflect individual elections so we can project elections outcomes.

In [12]:

```
nj_multimember = harvard_data[(harvard_data['outcome'] == 'w') &
(harvard_data['sab'] == 'nj')][harvard_data.duplicated(subset=['year'
'sab', 'ddez','sen', 'etype', 'outcome'], keep=False)]
```

Out[12]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
217767	2001	nj	1	0	g	vandrew, jeff	democrat	none	32271.0	w
217768	2001	nj	1	0	g	asselta, nicholas	modernrepublican	inc	36392.0	w
217773	2001	nj	2	0	g	blee, frank	modernrepublican	inc	29010.0	w

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
217774	2001	nj	2	0	g	damato, paul r.	modernrepublican	none	29427.0	w
217775	2001	nj	3	0	g	burzichelli, john j.	democrat	none	30213.0	w
...	...	...	...	...	...	...	...	...	...	...
221114	2013	nj	40	1	g	otoole, kevin j.	modernrepublican	inc	3751.0	w
221115	2013	nj	40	1	g	otoole, kevin j.	modernrepublican	inc	14674.0	w
221116	2013	nj	40	1	g	otoole, kevin j.	modernrepublican	inc	16752.0	w
221118	2015	nj	5	1	gs	cruzperez, nilsa	democrat	pastother	7979.0	w
221119	2015	nj	5	1	gs	cruzperez, nilsa	democrat	pastother	11171.0	w

1037 rows × 10 columns

Second, some states use mutli-member districts, i.e. more than one winner per election. The original dataset accounted for this through a dtype feature which aimed to keep track of individual seats by assigning arbitrary numbers to them. However, given that each seat in multimember districts has the exact same electorate, any artificial consistency on individual seats seems to me to be unnecessary information as far as the type of demographic analysis I aim to do goes.

In [13]:

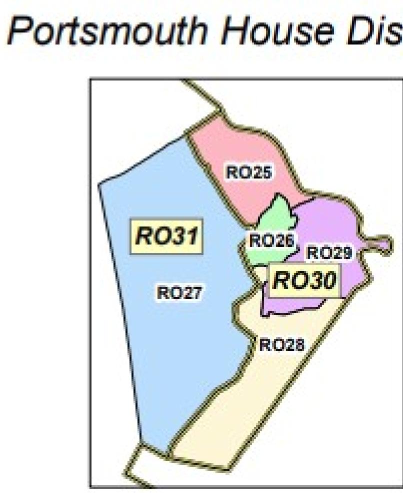
```
[illegible]
```

Out[13]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
140623	2000	ma	First Barnstable	0	g	george, thomas n.	modernrepublican	inc	19302.0	w
140624	2000	ma	Second Barnstable	0	g	atsalis, demetrius	democrat	inc	17044.0	w
140625	2000	ma	Second Barnstable	0	g	zalis, lawrence a.	modernrepublican	none	6327.0	l
140626	2000	ma	Third Barnstable	0	g	patrick, matthew	democrat	none	13285.0	w
140627	2000	ma	Third Barnstable	0	g	vieira, david joseph	modernrepublican	none	10508.0	l
...	...	...	...	...	...	...	...	...	...	...
145423	2016	ma	Worcester & Norfolk	1	g	fattman, ryan c.	republican	inc	64665.0	w
145424	2016	ma	Worcester & Norfolk	1	g	scattering	scattering	none	1021.0	l
145425	2016	ma	Plymouth & Norfolk	1	s	meschino, joan	democratic	none	8108.0	l

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
145426	2016	ma	Plymouth & Norfolk	1	s	oconnor, patrick m.	republican	none	9047.0	w
145427	2016	ma	Plymouth & Norfolk	1	s	scattering	scattering	none	30.0	l

Third, certain New England states don't number their districts, preferring county or locality designations. This is further complicated by New Hampshire's floterial districts. Put simply, some districts in New Hampshire's lower house consist of numerous other lower house districts put together. For example, see this map of Portsmouth's house districts.



In [14]:
```python
harvard_data[harvard_data['outcome'] ==
```

Out[14]:
```
modernrepublican          35132
democrat                  32710
republican                 5121
workingfamilies            1341
conservative               1104
independent                 998
democratfarmerlabor         892
democratic                  709
independence                342
democraticparty             337
indep                       247
4002                        225
reform                      157
working                     110
republicananddemocrat        95
liberal                      93
unspecifiednonmajor          90
democraticfarmerlabor        90
libertarian                  72
democratandrepublican        67
Name: party, dtype: int64
```

Yet another issue is the way candidates' parties are tracked. As previously mentioned, New York uses a fusion voting system allowing for a single candidate to run with multiple parties in the same race. The results record the number of votes each candidate gets with each respective party but as far as determining a winner the only thing that matters is the individual running, not their disparate party-lines.

However, other states do seem to allow for cross endorsement. One especially common feature in this data is candidates running as both democrats and republicans in Pennsylvania. They might be listed as "republicananddemocrat", "democratandrepublican" or other less common combinations.

Furthermore even pulling a list of people who only ran as Democrats isn't straight forward. Democrats in this data were recorded as "democrat," "democratfarmlabor" (Minnesota), "democratic", "democraticparty", and "democraticfarmlabor" (also Minnesota). And that's just among the 20 most common winning party labels.

To deal with this I had to do some research on various candidates that had uncommon party labels that by definition seemed to imply one of the two major parties but was written in some different way. I assembled lists of ways I would accept party listings as either with the Democratic or Republican parties.

Candidates running as a member of one of the major parties as well as some minor parties were counted as running with that major party. However, candidates running with endorsements from both major parties were excluded from the data.

In [15]:

```
republican = ['republican',
               'republican/independent',
               'republicanwritein',
               'writeinrepublican',
               'modernrepublican',
               'independentrepublicanparty',
               'independentrepublican',
               'gop/independentparty',
               'independentandrepublican'
               ]

democratic = ['progressive/democratic/workingfamily',
               'progressive/democratic',
               'progressivedemocrat',
               'writeindemocrat',
               'independentanddemocrat',
               'independentdemocrat',
               'independentdemparty','democrat',
               'democrat/progressive',
               'democrat/workingfamilies',
               'democratfarmerlabor',
               'democratic',
               'democratic/progressive',
               'democraticfarmerlabor',
               'democraticparty',
               'democraticwritein',
               'democratparty'
             ]
```

First we'll deal with the issue of results being split across counties and parties as a result of different reporting methods. For sanity's sake, I've highlighted a candidate whose rows need merging to make sure the function works as intended.

In [16]:

Out[16]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
237384	2010	ny	1	0	g	losquadro, daniel p.	unspecifiednonmajor	none	353.0	w
237385	2010	ny	1	0	g	losquadro, daniel p.	conservative	none	4752.0	w
237386	2010	ny	1	0	g	losquadro, daniel p.	modernrepublican	none	18755.0	w
238124	2012	ny	2	0	g	losquadro, daniel p.	independent	inc	1941.0	w
238125	2012	ny	2	0	g	losquadro, daniel p.	conservative	inc	5206.0	w
238126	2012	ny	2	0	g	losquadro, daniel p.	modernrepublican	inc	27158.0	w

In [17]:

```python
def fusion_county_voting(row):
    '''Takes in groupby-ed rows, returns single row with single standard major party ID.'''
    if row['party'].isin(democratic).any() and not row['party'].isin(republican).any():
        row['party'] = 'democratic'
    elif row['party'].isin(republican).any() and not row['party'].isin(democratic).any():
        row['party'] = 'republican'
    else:
        row['party'] = np.nan

    row['vote'] = row['vote'].sum()
    return row

merged_df = dupes.groupby(['year', 'cand', 'sab', 'ddez', 'sen', 'outcome'], as_index=False).apply(fusion_county_voting).drop_duplicates()
```

Out[17]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
3730	2014	al	2	0	g	betterton, andew (andy)	democratic	none	5224.0	l
3732	2014	al	2	0	g	greer, lynn	republican	inc	8561.0	w
3736	2014	al	3	0	g	black, marcel	democratic	inc	7993.0	w
3739	2014	al	3	0	g	joly, fred	republican	none	5357.0	l
3745	2014	al	4	0	g	hammon, micky	republican	inc	8473.0	w

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
...	...	...	...	...	...	...	...	...	...	...
378303	2016	wy	16	1	g	kusaba, richard	democratic	none	1989.0	l
378306	2016	wy	16	1	g	dockstader, dan	republican	inc	7208.0	w
378315	2016	wy	20	1	g	norskog, mary jane	democratic	none	1546.0	l
378320	2016	wy	20	1	g	agar, wyatt	republican	none	6893.0	w
378330	2016	wy	22	1	g	kinskey, dave	republican	none	7603.0	w

And after...

In [18]:

Out[18]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
237384	2010	ny	1	0	g	losquadro, daniel p.	republican	none	23860.0	w
238124	2012	ny	2	0	g	losquadro, daniel p.	republican	inc	34305.0	w

In [19]:

Out[19]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
3730	2014	al	2	0	g	betterton, andew (andy)	democratic	none	5224.0	l
3732	2014	al	2	0	g	greer, lynn	republican	inc	8561.0	w
3736	2014	al	3	0	g	black, marcel	democratic	inc	7993.0	w
3739	2014	al	3	0	g	joly, fred	republican	none	5357.0	l
3745	2014	al	4	0	g	hammon, micky	republican	inc	8473.0	w
...	...	...	...	...	...	...	...	...	...	...
5771	2014	al	29	1	g	mcclendon, melinda	republican	none	16145.0	w
5777	2014	al	30	1	g	morgan	NaN	none	5653.0	l
5782	2014	al	30	1	g	chambliss, clyde	republican	none	22916.0	w
5792	2014	al	31	1	g	greenwood, larry	democratic	none	8627.0	l
5796	2014	al	31	1	g	holley, jimmy w.	republican	inc	23067.0	w

125 rows × 10 columns

Looks good to me! Let's merge these results with the results that didn't have this duplicate problem.

In [20]:

```
election_data_merged = pd.concat([merged_df, harvard_data.drop(dupes.index)])
```

Out[20]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome
3730	2014	al	2	0	g	betterton, andew (andy)	democratic	none	5224.0	l
3732	2014	al	2	0	g	greer, lynn	republican	inc	8561.0	w
3736	2014	al	3	0	g	black, marcel	democratic	inc	7993.0	w
3739	2014	al	3	0	g	joly, fred	republican	none	5357.0	l
3745	2014	al	4	0	g	hammon, micky	republican	inc	8473.0	w
...	...	...	...	...	...	...	...	...	...	...
378340	2016	wy	28	1	g	anderson, james lee	republican	inc	5216.0	w
378341	2016	wy	28	1	g	scattering	writein	none	22.0	l
378342	2016	wy	30	1	g	ford, robert	democrat	none	1521.0	l
378343	2016	wy	30	1	g	scott, charles k.	republican	inc	5831.0	w
378344	2016	wy	30	1	g	scattering	writein	none	45.0	l

123816 rows × 10 columns

In [21]:

```
election_data_merged[(election_data_merged['cand'] != 'scattering') & (election_data_merged['etype'] == 'g')&(election_data_merged.drop(['vote', 'party'],
```

Out[21]:

year	sab	ddez	sen	etype	cand	party	exper	vote	outcome

Now I'll combine rows so each one represents one election. At this point I'll be dropping exclusively third party candidates. While they do occasionally win elections and in some states are even viable rivals, they remain rare exceptions. Where they do exist they tend to be regional and canvass with one of the two major parties. Including them would do little to inform about the dynamics of demographics as they influence partisan American elections.

In [22]:

```
election_data_merged['district'] = election_data_merged.apply(lambda row: ('u' if row['sen'] == 1 else 'l') + '{:0>3}'.format(row['ddez']), axis=1)
```

Out[22]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
3730	2014	al	2	0	g	betterton, andew (andy)	democratic	none	5224.0	l	l002
3732	2014	al	2	0	g	greer, lynn	republican	inc	8561.0	w	l002
3736	2014	al	3	0	g	black, marcel	democratic	inc	7993.0	w	l003

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
3739	2014	al	3	0	g	joly, fred	republican	none	5357.0	l	l003
3745	2014	al	4	0	g	hammon, micky	republican	inc	8473.0	w	l004
...	...	...	...	...	...	...	...	...	...	...	...
378340	2016	wy	28	1	g	anderson, james lee	republican	inc	5216.0	w	u028
378341	2016	wy	28	1	g	scattering	writein	none	22.0	l	u028
378342	2016	wy	30	1	g	ford, robert	democrat	none	1521.0	l	u030
378343	2016	wy	30	1	g	scott, charles k.	republican	inc	5831.0	w	u030
378344	2016	wy	30	1	g	scattering	writein	none	45.0	l	u030

For rows not included in the county/fusion voting cleanup, they need their party names standardized.

In [23]:

```
party_map = {}

for each in democratic:
    party_map[each] = 'democratic'
for each in republican:
    party_map[each] = 'republican'

election_data_merged['party'] =
election_data_merged['party'].map(party_map)
```

In [24]:

```
# drop non-general elections
election_data_merged =
election_data_merged[election_data_merged['etype'] ==
'g'].reset_index(drop=True)
```

In [25]:

```
election_data_merged
```

Out[25]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
0	2014	al	2	0	g	betterton, andew (andy)	democratic	none	5224.0	l	l002
1	2014	al	2	0	g	greer, lynn	republican	inc	8561.0	w	l002
2	2014	al	3	0	g	black, marcel	democratic	inc	7993.0	w	l003
3	2014	al	3	0	g	joly, fred	republican	none	5357.0	l	l003
4	2014	al	4	0	g	hammon, micky	republican	inc	8473.0	w	l004
...	...	...	...	...	...	...	...	...	...	...	...
106945	2016	wy	28	1	g	anderson, james lee	republican	inc	5216.0	w	u028

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
106946	2016	wy	28	1	g	scattering	NaN	none	22.0	l	u028
106947	2016	wy	30	1	g	ford, robert	democratic	none	1521.0	l	u030
106948	2016	wy	30	1	g	scott, charles k.	republican	inc	5831.0	w	u030
106949	2016	wy	30	1	g	scattering	NaN	none	45.0	l	u030

106950 rows × 11 columns

Feature Generation

Restricting Data by Years

At this point I decided to create a date cutoff for my data. District lines are generally redrawn every 10 years and in the year following the decennial census. As a result years ending in 2 tend to be the first ones featuring updated maps that last for roughly the next 10 years. I know I want this project to reflect current election trends first and foremost so it was never an option to use the completed 2010s decade cycle.

For the end date unfortunately no readily available database appears to exist that lists results for state-level legislative elections past 2016. I opted to collect some further data about New York (my home state) to use as holdout data but it would take a significant effort to do the same for all remaining data.

In [26]:

```
years = [2012, 2013, 2014, 2015, 2016]
states = election_data_merged['sab'].unique()
final_election_data = pd.DataFrame()

for year in years:
    for state in states:
        # districts vary by state
        districts = election_data_merged[(election_data_merged['sab'] == state) & (election_data_merged['year'] == year)]['district'].unique()
        for district in districts:
            # excludes third party exclusive candidates. multiparty candidates should already be combined into their largest representing party
            election = election_data_merged[(election_data_merged['year'] == year) & (election_data_merged['sab'] == state) & (election_data_merged['district'] == district) & (election_data_merged['party'].isin(democratic+republican))]
            # skip if election dataframe doesn't have at least 2 parties and at least 1 winner
            if election.empty or 'w' not in election['outcome'].unique() or len(election['party'].unique()) == 1:
                pass
            else:

```

```
                # number of seats is defined by the number of winning major party candidates per district per year
                num_of_seats = election['outcome'].value_counts()['w'
                if len(election) < num_of_seats * 2 or (num_of_seats != election['party'].isin(republican).value_counts()[True] or num_of_seats != election['party'].isin(democratic).value_counts()[True]):
                    pass
                else:
                    for seat in range(num_of_seats):
                        dem_vote = election[election['party'].isin(democratic)].iloc[seat]['vote'] / (election[election['party'].isin(democratic)].iloc[seat]['vote'] + election[election['party'].isin(republican)].iloc[seat]['vote'])
                        dem_incumbent = election[election['party'].isin(democratic)]['exper'].iloc[0]
                        if dem_incumbent == 'inc':
                            dem_incumbent = True
                        else:
                            dem_incumbent = False
                        rep_incumbent = election[election['party'].isin(republican)]['exper'].iloc[0]
                        if rep_incumbent == 'inc':
                            rep_incumbent = True
                        else:
                            rep_incumbent = False

                        if dem_incumbent == True:
                            previous_winner_dem = True
                        elif rep_incumbent == True:
                            previous_winner_dem = False
                        else:
                            # if no incumbent - check who won 2 years ago unless that would be a different district map
                            if year - 2 > 2011:
                                prev_election = election_data_merged[(election_data_merged['year'] == year - 2) & (election_data_merged['sab'] == state) & (election_data_merged['district'] == district)]
                                # west virginia has multimember staggered state senate elections
                                if prev_election.empty or state == 'wv' and district.startswith('u'):
                                    # or in some cases, 4 years ago
                                    if year - 4 > 2011:
                                        prev_election = election_data_merged[(election_data_merged['year'] == year - 4) & (election_data_merged['sab'] == state) & (election_data_merged['district'] == district)]
                                        if prev_election.empty:
                                            # or in the case of certain Montana Senate seats, 6 years
                                            if state == 'mt' and year - 6 > 2011:

```

```
                        prev_election = election_data_merged[(election_data_merged['year'] == year - 6) & (election_data_merged['sab'] == state) & (election_data_merged['district'] == district)]
                        if prev_election.empty:
                            previous_winner_dem = np.nan
                    else:
                        previous_winner_dem = np.nan
                # we only care about democratic vs republican
                elif prev_election[(prev_election['party'].isin(democratic+republican))].empty:
                    previous_winner_dem = np.nan
                else:
                    if len(prev_election[prev_election['outcome'] == 'w'])-1 < seat:
                        previous_winner_dem = np.nan
                    else:
                        party_of_prev_winner = prev_election[prev_election['outcome'] == 'w'].iloc[seat]['party']
                    if party_of_prev_winner in democratic:
                        previous_winner_dem = True
                    elif party_of_prev_winner in republican:
                        previous_winner_dem = False
            else:
                previous_winner_dem = np.nan
        elif prev_election[(prev_election['party'].isin(democratic+republican))].empty:
            previous_winner_dem = np.nan
        else:
            if len(prev_election[prev_election['outcome'] == 'w'])-1 < seat:
                previous_winner_dem = np.nan
            else:
                party_of_prev_winner = prev_election[prev_election['outcome'] == 'w'].iloc[seat]['party']
                if party_of_prev_winner in democratic:
                    previous_winner_dem = True
                elif party_of_prev_winner in republican:
                    previous_winner_dem = False
                else:
```

```
                                                if
len(prev_election[prev_election['outcome'] == 'l'])-1 < seat:
                                                    previous_winner_dem =
np.nan
                                                else:
                                                    if
prev_election[prev_election['outcome'] == 'l'].iloc[seat]['party'] i
democratic:
                                                        
 previous_winner_dem = False
                                                    else:
                                                        
 previous_winner_dem = np.nan
                                    else:
                                        previous_winner_dem = np.nan
                        # keep both incumbency info and previous winner
info as they're two related but distinct pieces of info
                        final_election_data =
final_election_data.append({'year': year,
                                                'state': state,
                                                'temp_district': district
                                                'district': district + '-
+ str(seat),
                                                'dem_vote': dem_vote,
                                                'previous_winner_dem':
previous_winner_dem,
                                                'dem_incumbent':
dem_incumbent,
                                                'rep_incumbent':
rep_incumbent
                                                }, ignore_index=True)
```

Out[26]:

	dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist
0	0.0	0.459653	l006-1	0.0	1.0	az	l
1	0.0	0.476306	l008-1	0.0	1.0	az	l
2	0.0	0.532791	l010-1	NaN	0.0	az	l
3	0.0	0.389172	l014-1	0.0	1.0	az	l
4	0.0	0.005714	l016-1	NaN	0.0	az	l
...	...	...	...	...	...	...	
9686	0.0	0.528484	u009-0	0.0	1.0	hi	u
9687	1.0	0.813933	u011-0	1.0	0.0	hi	u
9688	0.0	0.738679	u013-0	1.0	0.0	hi	u
9689	1.0	0.616010	u019-0	1.0	0.0	hi	u
9690	1.0	0.673618	u025-0	1.0	0.0	hi	u

9691 rows × 8 columns

In [27]:

```
# check to make sure no rows contain no vote data
```

Out[27]:

dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_district

In [28]:

```
# cases where there was no major party representative winner in the prior run for the seat
```

Out[28]:

	dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist
2	0.0	0.532791	l010-1	NaN	0.0	az	l
4	0.0	0.005714	l016-1	NaN	0.0	az	l
6	0.0	0.480042	l020-1	NaN	0.0	az	l
8	0.0	0.583422	l026-1	NaN	0.0	az	l
10	0.0	0.287712	u005-0	NaN	0.0	az	u
...	...	...	...	...	...	...	
9483	0.0	0.428948	u6-1-0	NaN	0.0	wv	u
9484	0.0	0.588194	u7-1-0	NaN	0.0	wv	u
9486	0.0	0.481926	u9-1-0	NaN	0.0	wv	u
9487	0.0	0.452720	u10-1-0	NaN	0.0	wv	u1
9488	0.0	0.488941	u11-1-0	NaN	0.0	wv	u1

1300 rows × 8 columns

It looks like some previous_winner_dem values are NaN. My intention is to exclude these when the prior winner isn't labeled as either a Democrat or Republican or the seat has changed due to redistricting. Let's spot check these to make sure that the only reason these are Nans.

In [29]:

```
election_data_merged[(election_data_merged['year'] < 2011) & (election_data_merged['sab'] == 'ms') &
```

Out[29]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
52460	2003	ms	40	0	g	barnett, ron	democratic	none	1164.0	l	l040
52461	2003	ms	40	0	g	mayhall, w. t. (ted)	republican	none	3173.0	w	l040
52640	2007	ms	40	0	g	mayhall, w. t. (ted)	NaN	inc	1847.0	w	l040

In [30]:

```
election_data_merged[(election_data_merged['year'] < 2011) & (election_data_merged['sab'] == 'va') &
```

Out[30]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
97888	2001	va	59	0	g	hagenau, h. p.	democratic	none	6829.0	l	l059

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
97889	2001	va	59	0	g	abbitt, watkins m. jr.	NaN	inc	11782.0	w	l059
98045	2003	va	59	0	g	hale, a. m.	democratic	none	5786.0	l	l059
98046	2003	va	59	0	g	abbitt, watkins m. jr.	NaN	inc	11834.0	w	l059
98196	2005	va	59	0	g	abbitt, watkins m. jr.	NaN	inc	16398.0	w	l059
98348	2007	va	59	0	g	brennan, connie	democratic	none	9136.0	l	l059
98349	2007	va	59	0	g	abbitt, watkins m. jr.	NaN	inc	13874.0	w	l059
98575	2009	va	59	0	g	abbitt, watkins m.	NaN	inc	16896.0	w	l059

In [31]:

```
# drop rows where prior winner couldn't be determined or the election was before 2012
final_election_data = final_election_data.dropna(subset=['previous_winner_dem'])
final_election_data = final_election_data[final_election_data['year']
```

Combining with census data

In order to match election data with census data by legislative district some examples need to be converted in specific ways. To accomplish this I'll be making a census matching column

In [32]:

```
final_election_data['census_matching_temp_district'] = final_election_data.apply(lambda row: 'l' + '{:0>3}'.format(row['temp_district'][1:-2]) if row['temp_district'][0 == 'l' and row['state'] == 'id' else
                                (row['temp_district'][0]+'{:0>2}'.format(row['temp_district'][1:2]) + row['temp_district'][-1] if row['temp_district'][0] == 'l' and row['state'] == 'mn' else
                                (row['temp_district'][:-2] + row['temp_district'][-1] if row['temp_district'][-1] in ['A', 'B'] an
row['state'] == 'sd' else
                                (row['temp_district'][0] + '{:0>3}'.format(row['temp_district'][1]) if row['temp_district'][0] =
'l' and row['state'] == 'wa' else
                                (row['temp_district'][0]+'{:0>3}'.format(row['temp_district'][1:-2]) if row['temp_district'][-2] == '-' and row['state'] == 'wv' else
                                ('l' + '{:0>2}'.format(row['temp_district'][1:-2]) + row['temp_district'][-1]  if row['temp_district'][-1] in ['A', 'B', 'C'] and row['state'] == 'md' else

```

```
                              (row['temp_district'].replace('lGrandisle', 'lGrand-Isle') if row['temp_district'][:10] == 'lGrandisle' and row['state'] == 'vt' else
                              (row['temp_district'].replace(' ', '-') if row['temp_district'][0] == 'l' and row['state'] == 'vt' else
```

In [33]:

```
# American Community Survey coded data to grab
values_to_grab = {'B01001001': "Total Pop",
                  'B01001002': "Male",
                  'B01001026': "Female",
                  'B02001002':"White alone",
                  'B02001003':"Black or African American alone",
                  'B02001004':"American Indian and Alaska Native alone",
                  'B02001005':"Asian alone",
                  'B02001006':"Native Hawaiian and Other Pacific Islander alone",
                  'B02001007':"Some other race alone",
                  'B02001008':"Two or more races:",
                  'B02001009':"Two races including Some other race",
                  'B02001010':"Two races excluding Some other race, and three or more races",
                  'B05001002':"U.S. citizen, born in the United States",
                  'B05001003':"U.S. citizen, born in Puerto Rico or U.S. Island Areas",
                  "B06007002":"Speak only English",
                  "B06007003":"Speak Spanish",
                  "B06007006":"Speak other languages",
                  "B06008002":"Never married",
                  "B06008003":"Now married, except separated",
                  "B06008004":"Divorced",
                  "B06008005":"Separated",
                  "B06008006":"Widowed",
                  "B06009002":"Less than high school graduate",
                  "B06009003":"High school graduate (includes equivalency)",
                  "B06009004":"Some college or associate's degree",
                  "B06009005":"Bachelor's degree",
                  "B06009006":"Graduate or professional degree",
                  "B06010002":"No income",
                  "B06010004":"$1 to $9,999 or loss",
                  "B06010005":"$10,000 to $14,999",
                  "B06010006":"$15,000 to $24,999",
                  "B06010007":"$25,000 to $34,999",
                  "B06010008":"$35,000 to $49,999",
                  "B06010009":"$50,000 to $64,999",
                  "B06010010":"$65,000 to $74,999",
                  "B06010011":"$75,000 or more",
                  "B16004002":"5 to 17 years",
                  "B16004024":"18 to 64 years",
                  "B16004046":"65 years and over"
```

In [34]:

```
# for converting Massachusetts lower house names
ordinal_dict = {
```

```
    '1st': 'First',
    '2nd': 'Second',
    '3rd': 'Third',
    '4th': 'Fourth',
    '5th': 'Fifth',
    '6th': 'Sixth',
    '7th': 'Seventh',
    '8th': 'Eighth',
    '9th': 'Ninth',
    '10th': 'Tenth',
    '11th': 'Eleventh',
    '12th': 'Twelfth',
    '13th': 'Thirteenth',
    '14th': 'Fourteenth',
    '15th': 'Fifteenth',
    '16th': 'Sixteenth',
    '17th': 'Seventeenth',
    '18th': 'Eighteenth',
    '19th': 'Nineteenth',
    '20th': 'Twentieth',
    '21st': 'Twenty-First',
    '22nd': 'Twenty-Second',
    '23rd': 'Twenty-Third',
    '24th': 'Twenty-Fourth',
    '25th': 'Twenty-Fifth',
    '26th': 'Twenty-Sixth',
    '27th': 'Twenty-Seventh',
    '28th': 'Twenty-Eighth',
    '29th': 'Twenty-Ninth',
    '30th': 'Thirtieth',
    '31st': 'Thirty-First',
    '32nd': 'Thirty-Second',
    '33rd': 'Thirty-Third',
    '34th': 'Thirty-Fourth',
    '35th': 'Thirty-Fifth',
    '36th': 'Thirty-Sixth',
    '37th': 'Thirty-Seventh'
```

In [35]:

```
# states and their census bureau numbers

states = {
'28': ["Mississippi","ms"],
'29': ["Missouri","mo"],
'30': ["Montana","mt"],
'31': ["Nebraska","ne"],
'32': ["Nevada","nv"],
'33': ["New Hampshire","nh"],
'34': ["New Jersey","nj"],
'35': ["New Mexico","nm"],
'36': ["New York","ny"],
'37': ["North Carolina","nc"],
'38': ["North Dakota","nd"],
'39': ["Ohio","oh"],
'40': ["Oklahoma","ok"],
'41': ["Oregon","or"],
'42': ["Pennsylvania","pa"],
```

```
'44': ["Rhode Island","ri"],
'45': ["South Carolina","sc"],
'46': ["South Dakota","sd"],
'47': ["Tennessee","tn"],
'48': ["Texas","tx"],
'50': ["Vermont","vt"],
'49': ["Utah","ut"],
'51': ["Virginia","va"],
'53': ["Washington","wa"],
'54': ["West Virginia","wv"],
'55': ["Wisconsin","wi"],
'56': ["Wyoming","wy"],
'01': ["Alabama","al"],
'02': ["Alaska","ak"],
'04': ["Arizona","az"],
'05': ["Arkansas","ar"],
'06': ["California","ca"],
'08': ["Colorado","co"],
'10': ["Delaware","de"],
'13': ["Georgia","ga"],
'09': ["Connecticut","ct"],
'12': ["Florida","fl"],
'16': ["Idaho","id"],
'15': ["Hawaii","hi"],
'17': ["Illinois","il"],
'18': ["Indiana","in"],
'19': ["Iowa","ia"],
'20': ["Kansas","ks"],
'21': ["Kentucky","ky"],
'22': ["Louisiana","la"],
'23': ["Maine","me"],
'24': ["Maryland","md"],
'25': ["Massachusetts","ma"],
'26': ["Michigan","mi"],
'27': ["Minnesota","mn"]
```

In [36]:

```
from drive.MyDrive.capstone.api_keys import census

census_data = pd.DataFrame()
for year in [int(str(x)[:4]) for x in final_election_data['year'].value_counts().index]:
    for state in states.keys():
        demo_data_senate = censusdata.download('acs5', year, censusdata.censusgeo([('state', state), ('state%20legislative%20district%20(upper%20chamber)', '*')]), [each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], key=census)
        # convert columns from acs codes
        demo_data_senate = demo_data_senate.rename(columns=dict(zip([each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], values_to_grab.values())))
        demo_data_senate = demo_data_senate.reset_index(drop=False)
        # Massachusetts specific rules
        if state == '25':

```

```
            demo_data_senate['index'] = demo_data_senate['index'].apply(lambda x: 'u' + x.name[:x.name.find('District')-1].replace(',', ''))
        # Vermont specific Rules
        elif state == '50':
            demo_data_senate['index'] = demo_data_senate['index'].apply(lambda x: 'u' + x.name.split()[0])
        else:
            demo_data_senate['index'] = demo_data_senate['index'].apply(lambda x: 'u'+x.geo[1][1].replace(',' ''))
        demo_data_senate = demo_data_senate.rename(columns={'index': 'census_matching_temp_district'})
        demo_data_senate['year'] = year
        demo_data_senate['state'] = states[state][1]
        # Nebraska has no lower house
        if state != '31':
            demo_data_assembly = censusdata.download('acs5', year, censusdata.censusgeo([('state', state), ('state%20legislative%20district%20(lower%20chamber)', '*')]), [each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], key=census)
            demo_data_assembly = demo_data_assembly.rename(columns=dict(zip([each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], values_to_grab.values())))
            demo_data_assembly = demo_data_assembly.reset_index(drop=False)

            # Massachusetts specific rules
            if state == '25':
                demo_data_assembly['index'] = demo_data_assembly['index'].apply(lambda x: 'l' + x.name.split('(')[0].strip()[:-9].replace(x.name.split()[0], ordinal_dict[x.name.split()[0]]) if x.name[0].isnumeric() else 'l' + x.name[:x.name.find('District')-1].replace(',', ''))
            # New Hampshire specific rules
            elif state == '33':
                demo_data_assembly['index'] = demo_data_assembly['index'].apply(lambda x: 'l' + ' '.join(x.name.split()[3:7:3]).replace(',', ''))
            # Vermont specific rules
            elif state == '50':
                demo_data_assembly['index'] = demo_data_assembly['index'].apply(lambda x: 'l' + x.name.split()[0].replace(',', ''))
            else:
                demo_data_assembly['index'] = demo_data_assembly['index'].apply(lambda x: 'l'+x.geo[1][1].replace(',', ''))
            demo_data_assembly = demo_data_assembly.rename(columns={'index': 'census_matching_temp_district'})
            demo_data_assembly['year'] = year
            demo_data_assembly['state'] = states[state][1]



```

```
            census_data = pd.concat([census_data, demo_data_assembly,
demo_data_senate])
        else:
            census_data = pd.concat([census_data, demo_data_senate])
```

Out[36]:

	census_matching_temp_district	Total Pop	Male	Female	White alone	Black or African American alone	American Indian and Alaska Native alone	Asian alone	H
0	l090	24519	12017	12502	16048	8229	3	29	
1	l091	24156	11596	12560	9804	14118	6	6	
2	l092	24381	12026	12355	19174	4806	9	237	
3	l093	25147	12755	12392	20440	3903	115	86	
4	l094	24227	12325	11902	7044	16098	33	55	
...	...	...	...	...	...	...	...	...	
62	u024	79732	40138	39594	72959	2718	312	703	
63	u025	79685	38793	40892	69584	3299	105	4235	
64	u026	80323	39463	40860	70061	3850	162	3735	
65	u027	79187	39239	39948	74425	1323	206	936	
66	u028	79064	38863	40201	76235	710	199	875	

33615 rows × 42 columns

In [37]:

```
# mapping New Hampshire's floterial districts to their component part
in order to assemble entire districts
nh_floterial_map = {
    'lBelknap' : [['8', '5', '7'],
                  ['9', '3', '6']],

    'lCarroll'  : [['7', '1','2', '3'],
                  ['8', '4','5']],

    'lCheshire': [['14', '11','9'],
                  ['15','10','12','13'],
                  ['16','4','5','6','7','8']],

    'lCoos'    : [['7', '2', '4', '5']],

    'lGrafton' : [['14','1', '2'],
                  ['15','3', '4'],
                  ['16','11', '6'],
                  ['17','10', '9']],

    'lHillsborough' : [['38','1','3','4'],
                       ['39','2','6'],
                       ['40','23','27','5'],
```

```
                        ['41','22','7'],
                        ['42','10','8','9'],
                        ['43','11','12','13','14'],
                        ['44','15','16','20'],
                        ['45','17','18','19']],
    'lMerrimack' : [['25','1','7'],
                     ['26','3','8','9'],
                     ['27','11','12','13','14','15','16'],
                     ['28','17','18','19'],
                     ['29','21','22']],
    'lRockingham': [['30','25','26','28','29'],
                     ['31','22','23','27'],
                     ['32','1','2'],
                     ['33','10','11','12'],
                     ['34','13','14'],
                     ['35','15','16'],
                     ['36','17','18','19'],
                     ['37','20','21']],
    'lStrafford': [['19','13','14'],
                    ['20','15','16'],
                    ['21','17','18'],
                    ['22','7','8'],
                    ['23','10','9'],
                    ['24','11','12'],
                    ['25','4','5']],
    'lSullivan': [['9','1','2','6'],
                   ['10','3','4','5'],
                   ['11','7','8']]
```

In [38]:

```
def nh_floterial_district_census_combiner(nh_floterial_map, census_data):
    '''Takes census_data and the floterial district map, returns census_data rows for the floterial districts'''
    flot_dists = pd.DataFrame()
    for county, numbers in nh_floterial_map.items():
        for floterial in range(len(numbers)):
            new_district = census_data[(census_data['census_matching_temp_district'].isin((county + ' ' + c for c in numbers[floterial][1:])))].groupby('year').sum()
            new_district['census_matching_temp_district'] = county + ' ' + numbers[floterial][0]
            new_district['state'] = 'nh'
            new_district['made_from'] = str(numbers[floterial][1:])
            new_district = new_district.reset_index(drop=False)
            flot_dists = flot_dists.append(new_district, ignore_index=True)

    return(flot_dists)
flots = nh_floterial_district_census_combiner(nh_floterial_map, census_data)
```

In [39]:

```
# add floterial districts to census_data
```

In [40]:

Out[40]:

	dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist
0	0.0	0.459653	l006-1	0.0	1.0	az	l
1	0.0	0.476306	l008-1	0.0	1.0	az	l
3	0.0	0.389172	l014-1	0.0	1.0	az	l
5	0.0	0.471326	l018-1	0.0	1.0	az	l
7	0.0	0.395057	l021-1	0.0	1.0	az	l
...	...	...	...	...	...	...	
9686	0.0	0.528484	u009-0	0.0	1.0	hi	u
9687	1.0	0.813933	u011-0	1.0	0.0	hi	u
9688	0.0	0.738679	u013-0	1.0	0.0	hi	u
9689	1.0	0.616010	u019-0	1.0	0.0	hi	u
9690	1.0	0.673618	u025-0	1.0	0.0	hi	u

8391 rows × 9 columns

In [41]:

```
sample = pd.merge(left = final_election_data,
          right = census_data_plus,
         how = 'left',
         on = ['census_matching_temp_district', 'year', 'state'])

```

Out[41]:

	dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist
0	0.0	0.459653	l006-1	0.0	1.0	az	l
1	0.0	0.476306	l008-1	0.0	1.0	az	l
2	0.0	0.389172	l014-1	0.0	1.0	az	l
3	0.0	0.471326	l018-1	0.0	1.0	az	l
4	0.0	0.395057	l021-1	0.0	1.0	az	l
...	...	...	...	...	...	...	
8386	0.0	0.528484	u009-0	0.0	1.0	hi	u
8387	1.0	0.813933	u011-0	1.0	0.0	hi	u
8388	0.0	0.738679	u013-0	1.0	0.0	hi	u
8389	1.0	0.616010	u019-0	1.0	0.0	hi	u

dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist

Now that we've combined our election data with census data, let's double-check to make sure every district we're interested in was properly mapped to census data.

In [42]:

Out[42]:

dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_district

---

The generic ballot is an indicator that is commonly used by political observers to gauge the general partisan sway of the country as a whole. It's gauged in political polls by asking voters which party they'd like to see control congress. It's generally believed that a very Republican-friendly environment can help Republican candidates down the ballot and vice versa in a Democratic-friendly environment. For this reason I'm including the national generic ballot averaged over the handful of polls preceding each year's November election as reflected on [RealClearPolitics' website (https://www.realclearpolitics.com/epolls/other/2022-generic-congressional-vote-7361.html)](https://www.realclearpolitics.com/epolls/other/2022-generic-congressional-vote-7361.html).

In [43]:

```
generic_ballot = {2011: 1.8,
                  2012: -0.2,
                  2013: -0.2,
                  2014: -2.4,
                  2015: 0,
                  2016: 0.6,
                  2018: 7.3,
                  2020: 6.8}

sample['generic_ballot'] = sample.apply(lambda row:
```

Another factor thought to be a strong indicator of partisanship is population density. To understand this I've grabbed the shapefiles for every legislative district from 2018 - well after each map should be settled.

In [44]:

```
leg_dist_areas = pd.DataFrame()

for state in states.keys():
    if state not in ['72', '11']:
#           senates

```

```
        shapefile = gpd.read_file("drive/MyDrive/capstone/SLDU/tl_2018_" + state + "_sldu/tl_2018_"+ state + "_sldu.shp")
        # must change projection to cea for accurate area
        shapefile = shapefile.to_crs({'proj': 'cea'})
        shapefile["area"] = shapefile['geometry'].area/ 10**6
        # Massachusetts naming
        if state == '25':
            shapefile['census_matching_temp_district'] = shapefile['NAMELSAD'].apply(lambda x: 'u' + x[:x.find('District')-1].replace(',', ''))
        # Vermont naming
        elif state == '50':
            shapefile['census_matching_temp_district'] = shapefile['NAMELSAD'].apply(lambda x: 'u' + x.split()[0])
        else:
            shapefile['census_matching_temp_district'] = shapefile['SLDUST'].apply(lambda x: 'u'+x.replace(',', ''))




        shapefile['state'] = shapefile['STATEFP'].apply(lambda x: states[state][1])
        leg_dist_areas = leg_dist_areas.append(shapefile[['area', 'census_matching_temp_district', 'state']], ignore_index=True)
#     lower houses
        # excluding Nebraska which has no lower house
        if state != '31':
            shapefile = gpd.read_file("drive/MyDrive/capstone/SLDL/tl_2018_" + state + "_sldl/tl_2018_"+ state + "_sldl.shp")
            shapefile = shapefile.to_crs({'proj': 'cea'})
            shapefile["area"] = shapefile['geometry'].area/ 10**6
            # Massachusetts naming
            if state == '25':
                shapefile['census_matching_temp_district'] = shapefile['NAMELSAD'].apply(lambda x: 'l' + x.split('(')[0].strip()[:-9].replace(x.split()[0], ordinal_dict[x.split()[0]]) if x[0].isnumeric() else 'l' + x[:x.find('District')-1].replace(',', ''))
            # New Hampshire naming
            elif state == '33':
                shapefile['census_matching_temp_district'] = shapefile['NAMELSAD'].apply(lambda x: 'l' + ' '.join(x.split()[3:7:3]).replace(',', ''))
            # Vermont naming
            elif state == '50':
                shapefile['census_matching_temp_district'] = shapefile['NAMELSAD'].apply(lambda x: 'l' + x.split()[0].replace(',', ''))
            else:
                shapefile['census_matching_temp_district'] = shapefile['SLDLST'].apply(lambda x: 'l'+x.replace(',', ''))
            shapefile['state'] = shapefile['STATEFP'].apply(lambda x: states[state][1])
            leg_dist_areas = leg_dist_areas.append(shapefile[['area', 'census_matching_temp_district', 'state']], ignore_index=True)

```

Out[44]:

	area	census_matching_temp_district	state
0	495.375785	u049	ms
1	1930.382029	u017	ms
2	947.814366	u006	ms
3	912.550467	u020	ms
4	3303.149472	u031	ms
...	...	...	...
6731	258.593410	l54B	mn
6732	46.139066	l53A	mn
6733	4587.942110	l06B	mn
6734	1047.503175	l03B	mn
6735	19.052332	l64A	mn

6736 rows × 3 columns

In [45]:

```
# similar to the census function, I needed to create a function to combine districts to account for NH's floterial districts
def nh_floterial_district_area_combiner(nh_floterial_map, leg_dist_areas):
    '''Takes district areas and the floterial district map, returns census_data rows for the floterial districts'''

    flot_dists = pd.DataFrame()
    for county, numbers in nh_floterial_map.items():
        for floterial in range(len(numbers)):
            new_district = leg_dist_areas[(leg_dist_areas['census_matching_temp_district'].isin(county + ' ' + c for c in numbers[floterial][1:]))))]
            new_district['census_matching_temp_district'] = county + ' ' + numbers[floterial][0]
            new_district['state'] = 'nh'
            new_district['made_from'] = str(numbers[floterial][1:])
            new_district = new_district.reset_index(drop=True)
            flot_dists = flot_dists.append(new_district, ignore_index=True)

    return(flot_dists)
flots = nh_floterial_district_area_combiner(nh_floterial_map, leg_dist_areas)
```

In [46]:

```
flots
```

Out[46]:

	area	census_matching_temp_district	state	made_from
0	369.821472	lBelknap 8	nh	['5', '7']
1	116.391494	lBelknap 8	nh	['5', '7']
2	82.575713	lBelknap 9	nh	['3', '6']
3	67.631465	lBelknap 9	nh	['3', '6']

	area	census_matching_temp_district	state	made_from
4	406.823180	lCarroll 7	nh	['1', '2', '3']
...	...	...	...	...
107	19.273284	lSullivan 10	nh	['3', '4', '5']
108	36.040656	lSullivan 10	nh	['3', '4', '5']
109	58.775621	lSullivan 10	nh	['3', '4', '5']
110	410.128183	lSullivan 11	nh	['7', '8']
111	98.522632	lSullivan 11	nh	['7', '8']

112 rows × 4 columns

In [47]:

```
leg_dist_areas_plus =
leg_dist_areas.append(flots.drop(columns='made_from'))
```

Out[47]:

	area	census_matching_temp_district	state
0	495.375785	u049	ms
1	1930.382029	u017	ms
2	947.814366	u006	ms
3	912.550467	u020	ms
4	3303.149472	u031	ms
...	...	...	...
107	19.273284	lSullivan 10	nh
108	36.040656	lSullivan 10	nh
109	58.775621	lSullivan 10	nh
110	410.128183	lSullivan 11	nh
111	98.522632	lSullivan 11	nh

6848 rows × 3 columns

In [48]:

```
sample = pd.merge(left=sample,
         right=leg_dist_areas_plus,
         how='left',
```

In [49]:

```
# confirm that every row has an area
```

Out[49]:

```
array([], dtype=object)
```

In [50]:

```
# derive population density in population per kilometer

sample['pop_density_km'] = sample['Total Pop']/sample['area']
```

Out[50]:

	dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist
0	0.0	0.459653	l006-1	0.0	1.0	az	l
1	0.0	0.476306	l008-1	0.0	1.0	az	l
2	0.0	0.389172	l014-1	0.0	1.0	az	l
3	0.0	0.471326	l018-1	0.0	1.0	az	l
4	0.0	0.395057	l021-1	0.0	1.0	az	l
...	...	...	...	...	...	...	
8539	0.0	0.528484	u009-0	0.0	1.0	hi	u
8540	1.0	0.813933	u011-0	1.0	0.0	hi	u
8541	0.0	0.738679	u013-0	1.0	0.0	hi	u
8542	1.0	0.616010	u019-0	1.0	0.0	hi	u
8543	1.0	0.673618	u025-0	1.0	0.0	hi	u

8544 rows × 51 columns

I'd like to convert my population counting rows into proportions of the total population. Otherwise my results will likely be skewed by the fact that senate seats will always be bigger than house seats within a given state.

In [51]:

```
for each in list(values_to_grab.values())[1:]:
    sample[each] = sample[each]/sample['Total Pop']
```

Out[51]:

	dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist
0	0.0	0.459653	l006-1	0.0	1.0	az	l
1	0.0	0.476306	l008-1	0.0	1.0	az	l

	dem_incumbent	dem_vote	district	previous_winner_dem	rep_incumbent	state	temp_dist
2	0.0	0.389172	l014-1	0.0	1.0	az	l
3	0.0	0.471326	l018-1	0.0	1.0	az	l
4	0.0	0.395057	l021-1	0.0	1.0	az	l
...	...	...	...	...	...	...	
8539	0.0	0.528484	u009-0	0.0	1.0	hi	u

In [52]:

```
# as Total Pop and area are completely implied by other features, I'm dropping them
```

In [53]:

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 8544 entries, 0 to 8543
Data columns (total 49 columns):
 #   Column                                                               Non
-Null Count  Dtype
---  ------
--------------  -----
 0   dem_incumbent                                                        854
4 non-null   float64
 1   dem_vote                                                             854
4 non-null   float64
 2   district                                                             854
4 non-null   object
 3   previous_winner_dem                                                  854
4 non-null   float64
 4   rep_incumbent                                                        854
4 non-null   float64
 5   state                                                                854
4 non-null   object
 6   temp_district                                                        854
4 non-null   object
 7   year                                                                 854
4 non-null   float64
 8   census_matching_temp_district                                        854
4 non-null   object
 9   Male                                                                 854
4 non-null   float64
 10  Female                                                               854
4 non-null   float64
 11  White alone                                                          854
4 non-null   float64
 12  Black or African American alone                                      854
4 non-null   float64
 13  American Indian and Alaska Native alone                              854
4 non-null   float64
 14  Asian alone                                                          854
4 non-null   float64
 15  Native Hawaiian and Other Pacific Islander alone                     854
4 non-null   float64
 16  Some other race alone                                                854
4 non-null   float64
 17  Two or more races:                                                   854
4 non-null   float64
 18  Two races including Some other race                                  854
4 non-null   float64
 19  Two races excluding Some other race, and three or more races         854
4 non-null   float64
 20  U.S. citizen, born in the United States                              854
4 non-null   float64
 21  U.S. citizen, born in Puerto Rico or U.S. Island Areas               854
4 non-null   float64
 22  Speak only English                                                   854
4 non-null   float64
 23  Speak Spanish                                                        854
4 non-null   float64
 24  Speak other languages                                                854
```

```
4 non-null   float64
 25   Never married                                                  854
4 non-null   float64
 26   Now married, except separated                                  854
4 non-null   float64
 27   Divorced                                                       854
4 non-null   float64
 28   Separated                                                      854
4 non-null   float64
 29   Widowed                                                        854
4 non-null   float64
 30   Less than high school graduate                                 854
4 non-null   float64
 31   High school graduate (includes equivalency)                    854
4 non-null   float64
 32   Some college or associate's degree                             854
4 non-null   float64
 33   Bachelor's degree                                              854
```

In [54]:

Out[54]:

	dem_incumbent	dem_vote	previous_winner_dem	rep_incumbent	year	l
count	8544.000000	8544.000000	8544.000000	8544.000000	8544.000000	8544.00
mean	0.372074	0.485887	0.445342	0.454588	2014.216175	0.49
std	0.483386	0.157199	0.497033	0.497963	1.580333	0.01
min	0.000000	0.000000	0.000000	0.000000	2012.000000	0.41
25%	0.000000	0.371840	0.000000	0.000000	2012.000000	0.48
50%	0.000000	0.466854	0.000000	0.000000	2014.000000	0.49
75%	1.000000	0.586498	1.000000	1.000000	2016.000000	0.50
max	1.000000	0.999818	1.000000	1.000000	2016.000000	0.59

Ultimately what I hope to be able to predict is which seats are likely to be tossups, i.e. could go either Republican or Democratic. I've defined that here as within 5 percentage points of 50%.

In [55]:

```
sample['dem_vote'] = sample['dem_vote'].apply(lambda x: -1 if x<= .45
```

In [56]:

Holdout Data

Generating some holdout data from after 2016

In [57]:
```
def process_ny_boe_data(document_name):
    output = pd.DataFrame()

    for district in pd.read_excel('drive/MyDrive/capstone/ny_data/' + document_name, sheet_name=None).keys():
        sheet = pd.read_excel('drive/MyDrive/capstone/ny_data/' + document_name, sheet_name=district, header=1)
        if 'Candidate Name (Party)' in sheet.columns:
            sheet = sheet.dropna()
            sheet['party'] = sheet['Candidate Name (Party)'].apply(lambda x: 'democratic' if x[-4:-1] == 'DEM' else ('republican' if x[-4:-1] == 'REP' else None))
            sheet = sheet[['Candidate Name (Party)', 'Total Votes by Candidate', 'party']]
            dnumber, dtype = district.split()
            dnumber = re.findall(r'\d+', dnumber)[0]
            if dtype == 'AD':
                district = 'l' + '{:0>3}'.format(dnumber)
            elif dtype == 'SD':
                district = 'u' + '{:0>3}'.format(dnumber)
            sheet['district'] = pd.Series([district, district])
            year = int(document_name[:4])
            sheet['year'] = pd.Series([year, year])
            output = pd.concat([output, sheet])
        else:
            pass


    output = output.rename(columns={'Candidate Name (Party)': 'cand', 'Total Votes by Candidate': "vote"})
```

In [58]:
```
# importing election data downloaded from NY's Board of Elections
holdout_ny = pd.DataFrame()

# because of covid-imposed limitations on census data, 2020 had to be excluded
for file in ['2018Assembly.xlsx', '2018NYSenate.xlsx']:
    holdout_ny = holdout_ny.append(process_ny_boe_data(file), ignore_index=True)

holdout_ny['sab'] = 'ny'
```

Out[58]:

	cand	vote	party	district	year	sab
0	Fred W. Thiele, Jr. (DEM)	31961.0	democratic	l001	2018.0	ny
1	Patrick M. O'Connor (REP)	19953.0	republican	l001	2018.0	ny
2	Rona Smith (DEM)	21533.0	democratic	l002	2018.0	ny
3	Anthony H. Palumbo (REP)	31242.0	republican	l002	2018.0	ny
4	Clyde E. Parker (DEM)	17822.0	democratic	l003	2018.0	ny

	cand	vote	party	district	year	sab
...	...	...	...	...	...	...
396	Joan Elizabeth Seamans (DEM)	51471.0	democratic	u061	2018.0	ny
397	Michael H. Ranzenhofer (REP)	60780.0	republican	u061	2018.0	ny
398	Robert G. Ortt (REP)	69118.0	republican	u062	2018.0	ny
399	Peter A. Diachun (GRE)	10539.0	None	NaN	NaN	ny
400	Timothy M. Kennedy (DEM)	70221.0	democratic	u063	2018.0	ny

401 rows × 6 columns

In [59]:

```
# grabbing data for the 2016 election in order to generate previous_win_dem column
ny_2016 = election_data_merged[(election_data_merged['sab'] == 'ny') (election_data_merged['year'] == 2016)]
ny_2016
```

Out[59]:

	year	sab	ddez	sen	etype	cand	party	exper	vote	outcome	district
5777	2016	ny	1	0	g	thiele, fred w. jr.	democratic	inc	35246.0	w	l001
5778	2016	ny	2	0	g	palumbo, anthony h.	republican	inc	39795.0	w	l002
5779	2016	ny	3	0	g	murray, dean	republican	inc	29087.0	w	l003
5780	2016	ny	4	0	g	englebright, steven	democratic	inc	31941.0	w	l004
5781	2016	ny	4	0	g	weissbard, steven	republican	none	21994.0	l	l004
...	...	...	...	...	...	...	...	...	...	...	...
71336	2016	ny	61	1	g	scattering	NaN	none	15.0	l	u061
71337	2016	ny	62	1	g	scattering	NaN	none	18.0	l	u062
71338	2016	ny	62	1	g	scattering	NaN	none	56.0	l	u062
71339	2016	ny	62	1	g	scattering	NaN	none	295.0	l	u062
71340	2016	ny	63	1	g	scattering	NaN	none	0.0	l	u063

750 rows × 11 columns

In [60]:

```
# largely a duplicate of the code that generated final_election_data but specific to the needs of this dataset
years = [2018]
states = ['ny']
holdout_data = pd.DataFrame()

for year in years:
    for state in states:
        districts = holdout_ny[(holdout_ny['sab'] == state) & (holdout_ny['year'] == year)]['district'].unique()
```

```
        for district in districts:
            election = holdout_ny[(holdout_ny['year'] == year) & (holdout_ny['sab'] == state) & (holdout_ny['district'] == district) & (holdout_ny['party'].isin(democratic+republican))]
            if len(election[(election['party'].isin(republican)) | (election['party'].isin(democratic))]) == 1:
                pass
            else:
                dem_net_vote = election[election['party'].isin(democratic)].iloc[0]['vote'] / (election[election['party'].isin(democratic)].iloc[0]['vote'] + election[election['party'].isin(republican)].iloc[0]['vote'])
                election = election.sort_values('vote', ascending=False).reset_index(drop=True)
                election['outcome'] = 'l'
                election.at[0, 'outcome'] = 'w'
                prev_election = ny_2016[(ny_2016['year'] == year - 2) & (ny_2016['sab'] == state) & (ny_2016['district'] == district)]

                party_of_prev_winner = prev_election[prev_election['outcome'] == 'w'].iloc[0]['party']
                if party_of_prev_winner in democratic:
                    previous_winner_dem = True
                elif party_of_prev_winner in republican:
                    previous_winner_dem = False
                else:
                    if len(prev_election[prev_election['outcome'] == 'l'])-1 < 0:
                        previous_winner_dem = np.nan
                    else:
                        if prev_election[prev_election['outcome'] == 'l'].iloc[0]['party'] in democratic:
                            previous_winner_dem = False
                        else:
                            previous_winner_dem = np.nan

                holdout_data = holdout_data.append({'year': year,
                                        'state': state,
                                        'temp_district': district,
                                        'district': district + '-' + str(0),
                                        'dem_net_vote': dem_net_vote,
                                        'previous_winner_dem': previous_winner_dem,
                                        'dem_incumbent': dem_incumbent,
                                        'rep_incumbent': rep_incumben
                                       }, ignore_index=True)

```

Out[60]:

	dem_incumbent	dem_net_vote	district	previous_winner_dem	rep_incumbent	state	temp_
0	1.0	0.615653	I001-0	1.0	0.0	ny	
1	1.0	0.408015	I002-0	0.0	0.0	ny	
2	1.0	0.460065	I003-0	0.0	0.0	ny	

	dem_incumbent	dem_net_vote	district	previous_winner_dem	rep_incumbent	state	temp_(
3	1.0	0.606159	l004-0	1.0	0.0	ny	
4	1.0	0.413354	l005-0	0.0	0.0	ny	
...	...	...	...	...	...	...	
125	1.0	0.480922	u055-0	0.0	0.0	ny	
126	1.0	0.444585	u056-0	0.0	0.0	ny	
127	1.0	0.405021	u058-0	0.0	0.0	ny	
128	1.0	0.442403	u060-0	0.0	0.0	ny	
129	1.0	0.458535	u061-0	0.0	0.0	ny	

In [61]:

```python
census_data = pd.DataFrame()
# acs data doesn't exist 2020 due to COVID
for year in [2018]:
    # NY is state 36 in the Census codebook
    for state in ['36']:
        demo_data_senate = censusdata.download('acs5', year, censusdata.censusgeo([('state', state), ('state%20legislative%20district%20(upper%20chamber)', '*')]), [each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], key=census)
        demo_data_senate = demo_data_senate.rename(columns=dict(zip([each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], values_to_grab.values())))
        demo_data_senate = demo_data_senate.reset_index(drop=False)
        demo_data_senate['index'] = demo_data_senate['index'].apply(lambda x: 'u'+x.geo[1][1].replace(',', ''))
        demo_data_senate = demo_data_senate.rename(columns={'index': 'temp_district'})
        demo_data_senate['year'] = year
        demo_data_senate['state'] = 'ny'

        demo_data_assembly = censusdata.download('acs5', year, censusdata.censusgeo([('state', state), ('state%20legislative%20district%20(lower%20chamber)', '*')]), [each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], key=census)
        demo_data_assembly = demo_data_assembly.rename(columns=dict(zip([each[:6] + '_' + each[6:] + 'E' for each in values_to_grab.keys()], values_to_grab.values())))
        demo_data_assembly = demo_data_assembly.reset_index(drop=False)
        demo_data_assembly['index'] = demo_data_assembly['index'].apply(lambda x: 'l'+x.geo[1][1].replace(',', ''))
        demo_data_assembly = demo_data_assembly.rename(columns={'index': 'temp_district'})
        demo_data_assembly['year'] = year
        demo_data_assembly['state'] = 'ny'

```

```
        census_data = pd.concat([census_data, demo_data_assembly,
demo_data_senate])
```

Out[61]:

	temp_district	Total Pop	Male	Female	White alone	Black or African American alone	American Indian and Alaska Native alone	Asian alone	Native Hawaiian and Other Pacific Islander alone	Some other race alone
0	l032	136376	63997	72379	6779	82696	443	15281	86	2648
1	l061	120894	59189	61705	64496	32340	371	9255	12	919
2	l140	128611	61121	67490	108673	7542	463	3629	18	374
3	l018	131400	63047	68353	30913	60487	1223	2330	0	2656
4	l029	129461	61024	68437	6302	87277	674	17719	9	1362
...	...	...	...	...	...	...	...	...	...	..
58	u042	293926	148790	145136	229571	26549	1184	6375	79	1794
59	u043	296988	146472	150516	275265	7422	457	5732	25	216
60	u044	293263	140757	152506	210092	44241	532	19138	133	502
61	u045	298350	154072	144278	272849	10505	4315	2591	160	358
62	u046	291782	145530	146252	257200	13296	532	6521	17	643

213 rows × 42 columns

In [62]:

```
holdout_data = pd.merge(left = holdout_data,
          right = census_data,
        how = 'left',
        on = ['temp_district', 'year', 'state'])
```

Out[62]:

	dem_incumbent	dem_net_vote	district	previous_winner_dem	rep_incumbent	state	temp_
0	1.0	0.615653	l001-0	1.0	0.0	ny	
1	1.0	0.408015	l002-0	0.0	0.0	ny	
2	1.0	0.460065	l003-0	0.0	0.0	ny	
3	1.0	0.606159	l004-0	1.0	0.0	ny	

	dem_incumbent	dem_net_vote	district	previous_winner_dem	rep_incumbent	state	temp_(
4	1.0	0.413354	l005-0	0.0	0.0	ny	
...	...	...	...	...	...	...	
125	1.0	0.480922	u055-0	0.0	0.0	ny	

In [63]:

```
generic_ballot = {2011: 1.8,
                  2012: -0.2,
                  2013: -0.2,
                  2014: -2.4,
                  2015: 0,
                  2016: 0.6,
                  2018: 7.3,
                  2020: 6.8}
holdout_data['generic_ballot'] = holdout_data.apply(lambda row:
```

In [64]:

```
leg_dist_areas = pd.DataFrame()

for state in ['36']:
    # NY senate
    shapefile = gpd.read_file("drive/MyDrive/capstone/SLDU/tl_2018_" + state + "_sldu/tl_2018_"+ state + "_sldu.shp")
    # change projection to cea for accurate area
    shapefile = shapefile.to_crs({'proj': 'cea'})
    shapefile["area"] = shapefile['geometry'].area/ 10**6
    shapefile['temp_district'] = shapefile['SLDUST'].apply(lambda x: 'u'+x.replace(',', ''))




    shapefile['state'] = "ny"
    leg_dist_areas = leg_dist_areas.append(shapefile[['area', 'temp_district', 'state']], ignore_index=True)

    # NY assembly
    shapefile = gpd.read_file("drive/MyDrive/capstone/SLDL/tl_2018_" + state + "_sldl/tl_2018_"+ state + "_sldl.shp")
    shapefile = shapefile.to_crs({'proj': 'cea'})
    shapefile["area"] = shapefile['geometry'].area/ 10**6
    shapefile['temp_district'] = shapefile['SLDLST'].apply(lambda x: 'l'+x.replace(',', ''))
    shapefile['state'] = "ny"
    leg_dist_areas = leg_dist_areas.append(shapefile[['area', 'temp_district', 'state']], ignore_index=True)

```

Out[64]:

	area	temp_district	state
0	10253.021636	u049	ny
1	16.347054	u017	ny
2	131.168312	u006	ny
3	13.213151	u020	ny
4	1622.232442	u056	ny
...	...	...	...
208	7.380144	l053	ny
209	996.896394	l105	ny
210	174.316577	l129	ny
211	5.627293	l042	ny
212	162.113485	l014	ny

213 rows × 3 columns

In [65]:

```
holdout_data = pd.merge(left=holdout_data,
```

```python
        right=leg_dist_areas,
        how='left',
        on=['temp_district', 'state'])
holdout_data['pop_density_km'] = holdout_data['Total Pop']/holdout_data['area']
```

Out[65]:

	dem_incumbent	dem_net_vote	district	previous_winner_dem	rep_incumbent	state	temp_(
0	1.0	0.615653	l001-0	1.0	0.0	ny	
1	1.0	0.408015	l002-0	0.0	0.0	ny	
2	1.0	0.460065	l003-0	0.0	0.0	ny	
3	1.0	0.606159	l004-0	1.0	0.0	ny	
4	1.0	0.413354	l005-0	0.0	0.0	ny	
...	...	...	...	...	...	...	
125	1.0	0.480922	u055-0	0.0	0.0	ny	
126	1.0	0.444585	u056-0	0.0	0.0	ny	
127	1.0	0.405021	u058-0	0.0	0.0	ny	
128	1.0	0.442403	u060-0	0.0	0.0	ny	
129	1.0	0.458535	u061-0	0.0	0.0	ny	

130 rows × 50 columns

In [66]:

```python
# converting census data to percentage of total pop
for each in list(values_to_grab.values())[1:]:
    holdout_data[each] = holdout_data[each]/holdout_data['Total Pop']

```

Out[66]:

	dem_incumbent	dem_net_vote	district	previous_winner_dem	rep_incumbent	state	temp_(
0	1.0	0.615653	l001-0	1.0	0.0	ny	
1	1.0	0.408015	l002-0	0.0	0.0	ny	
2	1.0	0.460065	l003-0	0.0	0.0	ny	
3	1.0	0.606159	l004-0	1.0	0.0	ny	

	dem_incumbent	dem_net_vote	district	previous_winner_dem	rep_incumbent	state	temp_(
4	1.0	0.413354	l005-0	0.0	0.0	ny	
...	...	...	...	...	...	...	
125	1.0	0.480922	u055-0	0.0	0.0	ny	

In [67]:

```
holdout_data = holdout_data.drop(columns=['Total Pop', 'area',
'district'])
```

In [68]:

```
holdout_data['dem_net_vote'] =
holdout_data['dem_net_vote'].apply(lambda x: -1 if x<= .45 else(1 if x
>= .55 else 0))
holdout_data = holdout_data.dropna()
holdout_data
```

Out[68]:

	dem_incumbent	dem_net_vote	previous_winner_dem	rep_incumbent	state	temp_district
0	1.0	1	1.0	0.0	ny	l001
1	1.0	-1	0.0	0.0	ny	l002
2	1.0	0	0.0	0.0	ny	l003
3	1.0	1	1.0	0.0	ny	l004
4	1.0	-1	0.0	0.0	ny	l005
...	...	...	...	...	...	...
125	1.0	0	0.0	0.0	ny	u055
126	1.0	-1	0.0	0.0	ny	u056
127	1.0	-1	0.0	0.0	ny	u058
128	1.0	-1	0.0	0.0	ny	u060
129	1.0	0	0.0	0.0	ny	u061

129 rows × 47 columns

Model and Feature Selection

In [138]:

```
import pickle

class BorutaFeatureSelection():
    def __init__(self):
        pass

    def fit(self, X, y):
        self.target_encoder = ce.target_encoder.TargetEncoder().fit(X[['state', 'temp_district']], y)
        Xtr = self.target_encoder.transform(X[['state', 'temp_district']])
        # print(Xtr)
        X['state'] = Xtr['state']
        X['temp_district'] = Xtr['temp_district']
        # print(X['state'])
        perc = 100
        while True:
            boruta = BorutaPy(
                    estimator = RandomForestClassifier(),
                    n_estimators = 'auto',
                    perc=perc
                ).fit(np.array(X), pd.Series(y))

            self.green_area = X.columns[boruta.support_].to_list()
            if perc == 10:
                break
            if len(self.green_area) >= 5:
                break
            perc -= 10

        return(self)


    def transform(self, X):
        if X['state'].dtype == 'O':
            Xtr = self.target_encoder.transform(X[['state', 'temp_district']])
            X['state'] = Xtr['state']
            X['temp_district'] = Xtr['temp_district']
        return(X[self.green_area])

    def fit_transform(self, X, y):
        self.fit(X, y)
        return(self.transform(X))

X = sample.drop(["census_matching_temp_district", 'district', 'dem_vote'], axis=1)
y = pd.Series(sample['dem_vote'])

b = BorutaFeatureSelection()

X = b.fit_transform(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
random_state=12)

X_holdout = holdout_data.drop(['dem_net_vote'], axis=1)
y_holdout = holdout_data['dem_net_vote']

target_encoder = ce.target_encoder.TargetEncoder().fit(X[['state',
'temp_district']], y)
Xtr = b.target_encoder.transform(X_holdout[['state',
'temp_district']])
# print(Xtr)
X_holdout['state'] = Xtr['state']
X_holdout['temp_district'] = Xtr['temp_district']

```

Baseline model - Guessing the mode

In [70]:

```
from scipy import stats

# "Learn" the mode from the training data
mode_train = stats.mode(y_train)
# Get predictions on the test set
baseline_predictions = np.ones(y_test.shape) * mode_train.mode
# Compute MAE
print(classification_report(y_test, baseline_predictions))
# Recall_baseline = recall_score(y_test, baseline_predictions,
average='micro')
```

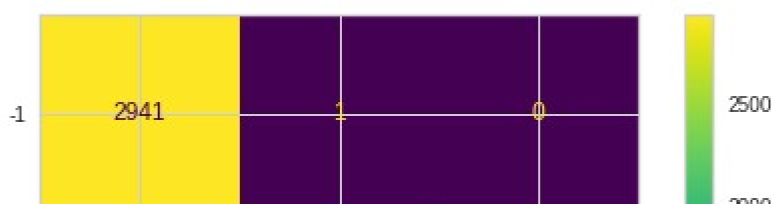
```
              precision    recall  f1-score   support

          -1       0.46      1.00      0.63       980
           0       0.00      0.00      0.00       473
           1       0.00      0.00      0.00       683

    accuracy                           0.46      2136
   macro avg       0.15      0.33      0.21      2136
weighted avg       0.21      0.46      0.29      2136
```

In [71]:

```
national_models = []

models = [
          RandomForestClassifier(),
          DecisionTreeClassifier(),
          LogisticRegression(),
          xgb.XGBClassifier(),
          lgb.LGBMClassifier(),
          CatBoostClassifier(verbose=False)
          ]

for model in models:

    pipe = Pipeline([
        ('scaler', StandardScaler()),
        ("classifier", model)
```

```
    ])

    start = datetime.now()
    pipe.fit(X_train, y_train)
    print('Fit time: ', datetime.now() - start)

    train_preds = pipe.predict(X_train)
    test_preds = pipe.predict(X_test)
    holdout_preds = pipe.predict(X_holdout)

    print('Train Results')

    print(classification_report(y_train, train_preds))
    plot_confusion_matrix(pipe, X_train, y_train)
    plt.show()

    print('Test Results')

    print(classification_report(y_test, test_preds))
    plot_confusion_matrix(pipe, X_test, y_test)
    plt.show()

    print('Holdout Results')

    print(classification_report(y_holdout, holdout_preds))
    plot_confusion_matrix(pipe, X_holdout, y_holdout)
    plt.show()

```

```
Fit time:  0:00:03.474945
Train Results
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00      2942
           0       1.00      0.99      1.00      1419
           1       1.00      1.00      1.00      2047

    accuracy                           1.00      6408
   macro avg       1.00      1.00      1.00      6408
weighted avg       1.00      1.00      1.00      6408
```



In [72]:

Out[72]:

```
{'auto_class_weights': 'None',
 'bagging_temperature': 1,
 'bayesian_matrix_reg': 0.10000000149011612,
 'best_model_min_trees': 1,
 'boost_from_average': False,
 'boosting_type': 'Plain',
 'bootstrap_type': 'Bayesian',
 'border_count': 254,
 'class_names': [-1, 0, 1],
 'classes_count': 0,
 'depth': 6,
 'eval_metric': 'MultiClass',
 'feature_border_type': 'GreedyLogSum',
 'force_unit_auto_pair_weights': False,
 'grow_policy': 'SymmetricTree',
 'iterations': 1000,
 'l2_leaf_reg': 3,
 'leaf_estimation_backtracking': 'AnyImprovement',
 'leaf_estimation_iterations': 1,
 'leaf_estimation_method': 'Newton',
 'learning_rate': 0.08698900043964386,
 'loss_function': 'MultiClass',
 'max_leaves': 64,
 'min_data_in_leaf': 1,
 'model_shrink_mode': 'Constant',
 'model_shrink_rate': 0,
 'model_size_reg': 0.5,
 'nan_mode': 'Min',
 'penalties_coefficient': 1,
 'pool_metainfo_options': {'tags': {}},
 'posterior_sampling': False,
 'random_seed': 0,
 'random_strength': 1,
 'rsm': 1,
```

All these classifiers are performing pretty well but most are showing serious overfitting, as indicated by the fact that the train scores in all but the Logistic Regression model are significantly better than either the test or holdout scores. Given these results I feel that both the XGB and LGBM models show the strongest likelihood of improving given hyperparamter tuning.

Model Itteration

In [78]:

Out[78]:

```
{'auto_class_weights': 'None',
 'bagging_temperature': 1,
 'bayesian_matrix_reg': 0.10000000149011612,
 'best_model_min_trees': 1,
 'boost_from_average': False,
 'boosting_type': 'Plain',
 'bootstrap_type': 'Bayesian',
 'border_count': 254,
 'class_names': [-1, 0, 1],
 'classes_count': 0,
 'depth': 6,
 'eval_metric': 'MultiClass',
 'feature_border_type': 'GreedyLogSum',
 'force_unit_auto_pair_weights': False,
 'grow_policy': 'SymmetricTree',
 'iterations': 1000,
 'l2_leaf_reg': 3,
 'leaf_estimation_backtracking': 'AnyImprovement',
 'leaf_estimation_iterations': 1,
 'leaf_estimation_method': 'Newton',
 'learning_rate': 0.08698900043964386,
 'loss_function': 'MultiClass',
 'max_leaves': 64,
 'min_data_in_leaf': 1,
 'model_shrink_mode': 'Constant',
 'model_shrink_rate': 0,
 'model_size_reg': 0.5,
 'nan_mode': 'Min',
 'penalties_coefficient': 1,
 'pool_metainfo_options': {'tags': {}},
```


Seeing as the CatBoost model default is overfit, let's try reducing the number of trees through the iterations parameter.

In [79]:

```python

pipe = Pipeline([
('scaler', StandardScaler())])


param_range = [1, 10,100, 500, 1000]

cb = national_models[-1]

train_scores, test_scores = validation_curve(estimator = cb,
                                             X = pipe.fit_transform(X, y),
                                             y = y,
                                             param_name='iterations',
                                             param_range=param_range)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.title("Validation Curve")
```

```python

plt.ylabel("Score")

lw = 2
plt.semilogx(param_range, train_scores_mean, label="Training score",
             color="darkorange", lw=lw)
plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.2,
                 color="darkorange", lw=lw)
plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",
             color="navy", lw=lw)
plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.2,
                 color="navy", lw=lw)
plt.legend(loc="best")
```

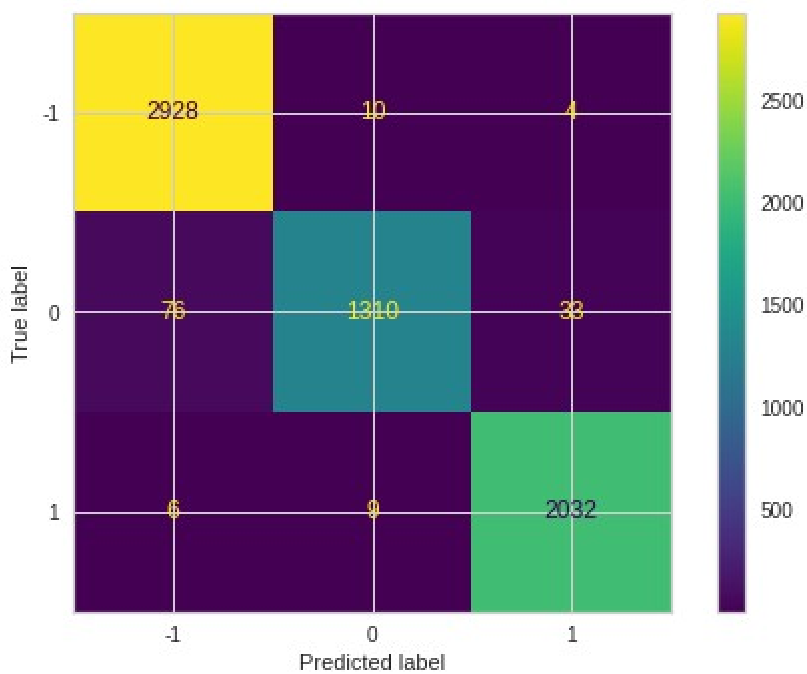
The cross-validation score looks to peak around 700 iterations. Let's see how that performs.

In [98]:

```python
cb = CatBoostClassifier(verbose = False, iterations=700)

pipe = Pipeline([
        ('scaler', StandardScaler()),
        ("classifier", cb)
    ])

start = datetime.now()
pipe.fit(X_train, y_train)
print('Fit time: ', datetime.now() - start)

train_preds = pipe.predict(X_train)
test_preds = pipe.predict(X_test)
holdout_preds = pipe.predict(X_holdout)
```

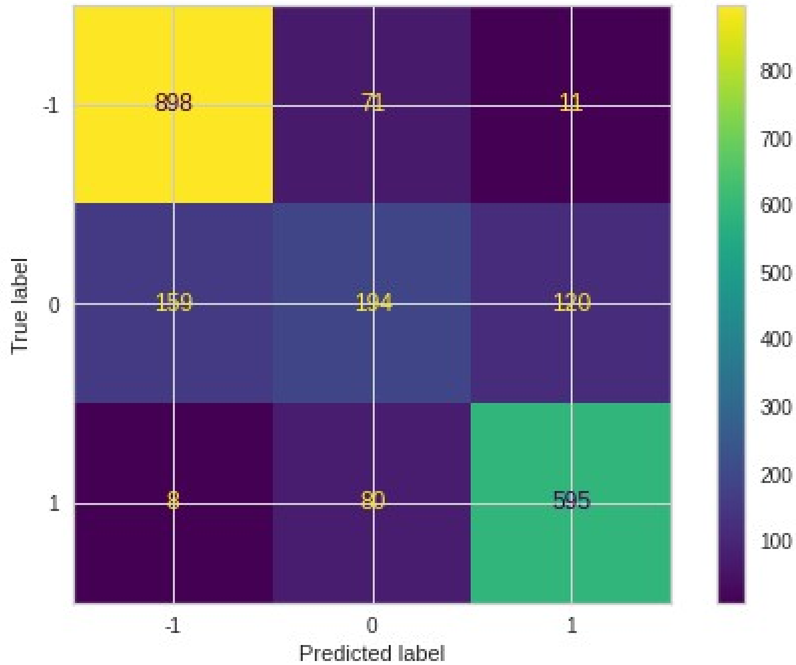
```
print('Train Results')

print(classification_report(y_train, train_preds))
plot_confusion_matrix(pipe, X_train, y_train)
plt.show()

print('Test Results')

print(classification_report(y_test, test_preds))
plot_confusion_matrix(pipe, X_test, y_test)
plt.show()

print('Holdout Results')

print(classification_report(y_holdout, holdout_preds))
plot_confusion_matrix(pipe, X_holdout, y_holdout)
```

```
Fit time:  0:00:17.490548
Train Results
              precision    recall  f1-score   support

          -1       0.97      1.00      0.98      2942
           0       0.99      0.92      0.95      1419
           1       0.98      0.99      0.99      2047

    accuracy                           0.98      6408
   macro avg       0.98      0.97      0.97      6408
weighted avg       0.98      0.98      0.98      6408
```

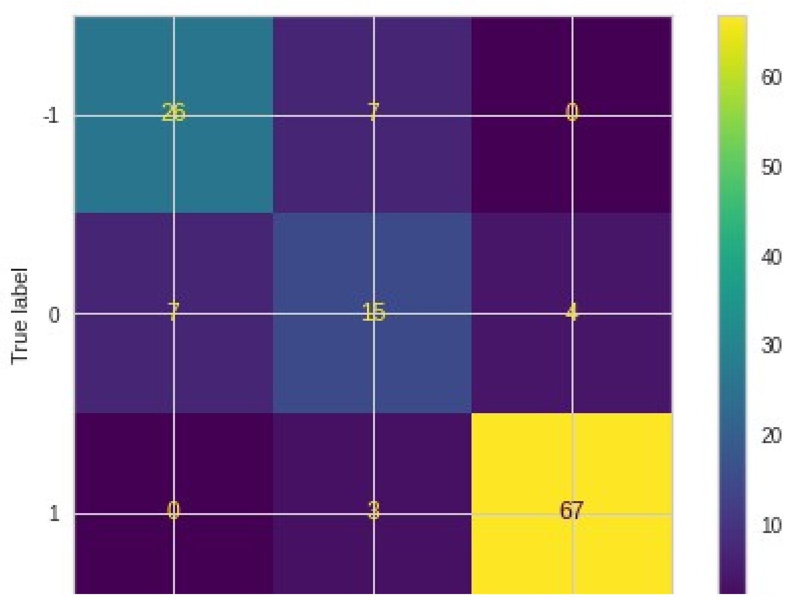



```
Test Results
              precision    recall  f1-score   support

          -1       0.84      0.92      0.88       980
           0       0.56      0.41      0.47       473
```

```
           1       0.82      0.87      0.84       683

    accuracy                           0.79      2136
   macro avg       0.74      0.73      0.73      2136
weighted avg       0.77      0.79      0.78      2136
```




```
Holdout Results
              precision    recall  f1-score   support

          -1       0.79      0.79      0.79        33
           0       0.60      0.58      0.59        26
           1       0.94      0.96      0.95        70

    accuracy                           0.84       129
   macro avg       0.78      0.77      0.78       129
weighted avg       0.83      0.84      0.84       129
```



The test score stayed the same but the holdout score went down a little - from 88 to 84. There's not a lot to read into here other than the fact that overfitting is still present. Per CatBoost's documentation, let's try decreasing the learning rate.

In [100]:

```
# pre-existing parameters
```

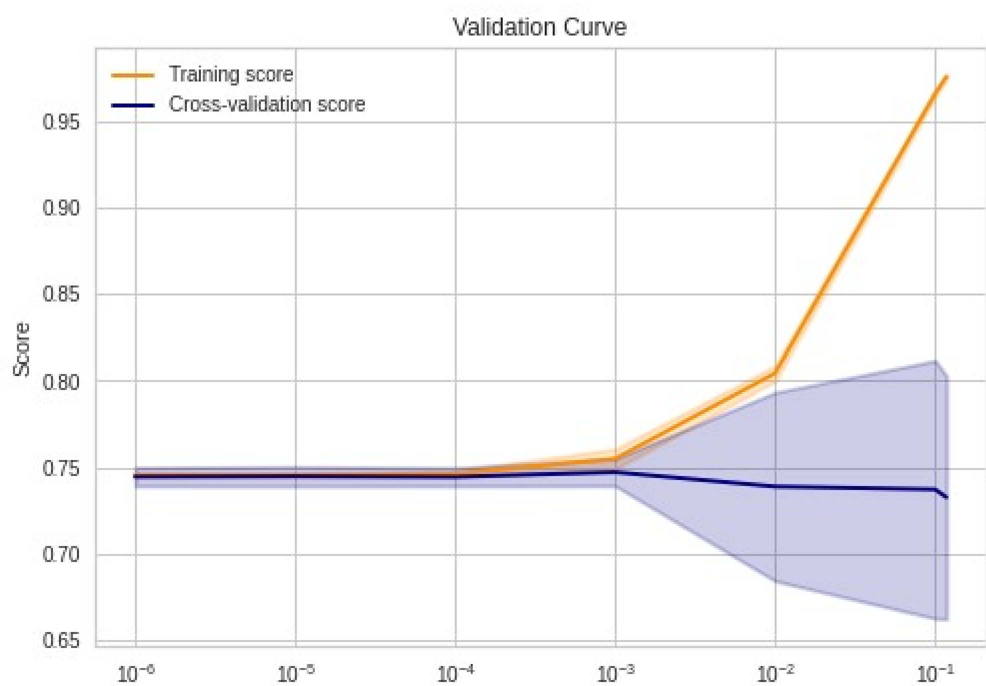
Out[100]:

```
{'auto_class_weights': 'None',
 'bagging_temperature': 1,
 'bayesian_matrix_reg': 0.10000000149011612,
 'best_model_min_trees': 1,
 'boost_from_average': False,
 'boosting_type': 'Plain',
 'bootstrap_type': 'Bayesian',
 'border_count': 254,
 'class_names': [-1, 0, 1],
 'classes_count': 0,
 'depth': 6,
 'eval_metric': 'MultiClass',
 'feature_border_type': 'GreedyLogSum',
 'force_unit_auto_pair_weights': False,
 'grow_policy': 'SymmetricTree',
 'iterations': 700,
 'l2_leaf_reg': 3,
 'leaf_estimation_backtracking': 'AnyImprovement',
 'leaf_estimation_iterations': 1,
 'leaf_estimation_method': 'Newton',
 'learning_rate': 0.11758700013160706,
 'loss_function': 'MultiClass',
 'max_leaves': 64,
 'min_data_in_leaf': 1,
 'model_shrink_mode': 'Constant',
 'model_shrink_rate': 0,
 'model_size_reg': 0.5,
 'nan_mode': 'Min',
 'penalties_coefficient': 1,
 'pool_metainfo_options': {'tags': {}},
 'posterior_sampling': False,
 'random_seed': 0,
 'random_strength': 1,
 'rsm': 1,
 'sampling_frequency': 'PerTree',
 'score_function': 'Cosine',
 'sparse_features_conflict_fraction': 0,
 'task_type': 'CPU',
 'use_best_model': False}
```

In [101]:

```
pipe = Pipeline([
('scaler', StandardScaler())])


param_range = [0.11758700013160706, .1, .01, .001, .0001, .00001, .000001]

```

```
cb = CatBoostClassifier(verbose=False, iterations=700)

train_scores, test_scores = validation_curve(estimator = cb,
                                             X = pipe.fit_transform(X, y),
                                             y = y,
                                             param_name='learning_rate',
                                             param_range=param_range)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.title("Validation Curve")

plt.ylabel("Score")

lw = 2
plt.semilogx(param_range, train_scores_mean, label="Training score",
             color="darkorange", lw=lw)
plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.2,
                 color="darkorange", lw=lw)
plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",
             color="navy", lw=lw)
plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.2,
                 color="navy", lw=lw)
plt.legend(loc="best")
```

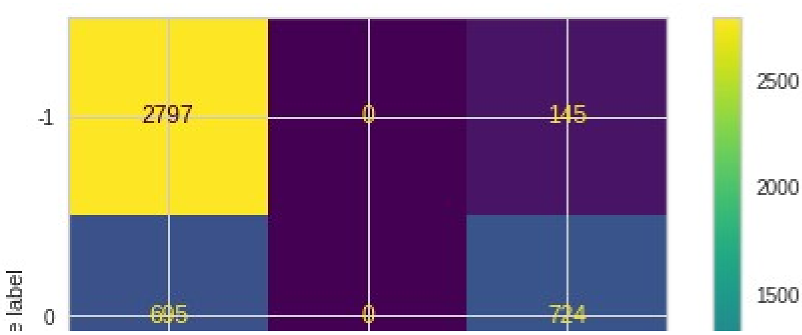


It looks like the cross-validation score is highest around .0001. Let's see where that gets us.

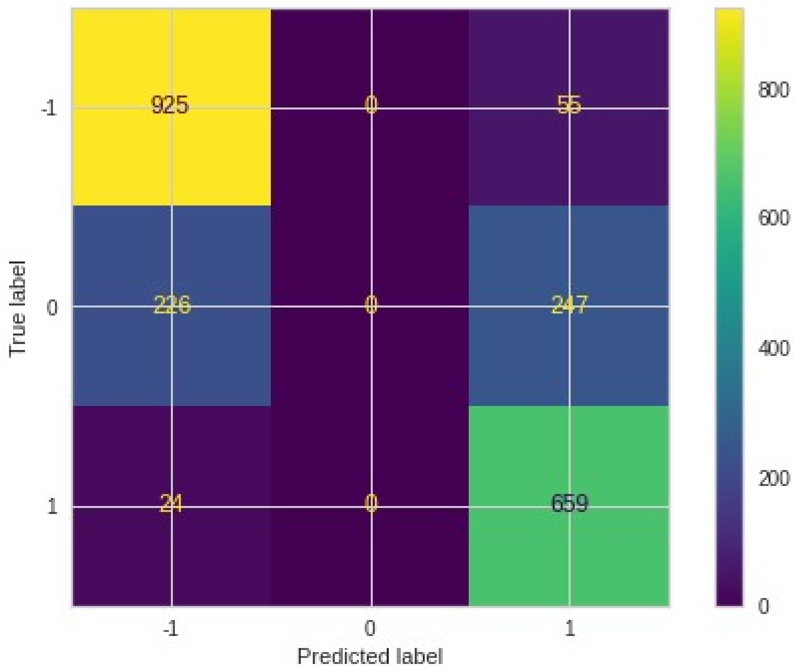
In [139]:

```
cb = CatBoostClassifier(verbose = False, iterations=700,
learning_rate=.0001)

pipe = Pipeline([
        ('scaler', StandardScaler()),
        ("classifier", cb)
    ])

start = datetime.now()
pipe.fit(X_train, y_train)
print('Fit time: ', datetime.now() - start)

train_preds = pipe.predict(X_train)
test_preds = pipe.predict(X_test)
holdout_preds = pipe.predict(X_holdout)

print('Train Results')

print(classification_report(y_train, train_preds))
plot_confusion_matrix(pipe, X_train, y_train)
plt.show()

print('Test Results')

print(classification_report(y_test, test_preds))
plot_confusion_matrix(pipe, X_test, y_test)
plt.show()

print('Holdout Results')

print(classification_report(y_holdout, holdout_preds))
plot_confusion_matrix(pipe, X_holdout, y_holdout)
```

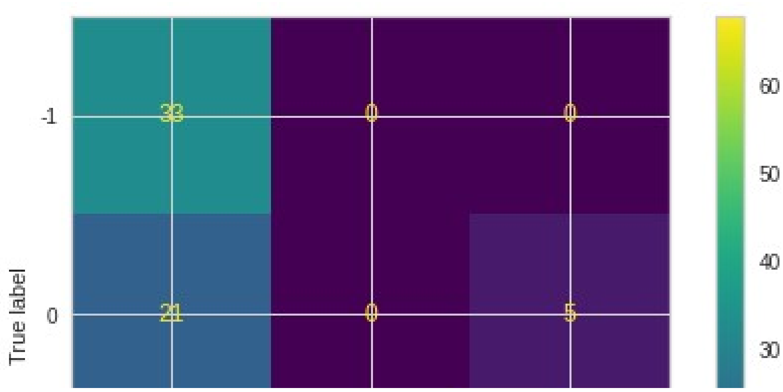
```
Fit time:  0:00:16.956833
Train Results
              precision    recall  f1-score   support

          -1       0.78      0.95      0.86      2942
           0       0.00      0.00      0.00      1419
           1       0.69      0.96      0.81      2047

    accuracy                           0.74      6408
   macro avg       0.49      0.64      0.56      6408
weighted avg       0.58      0.74      0.65      6408
```



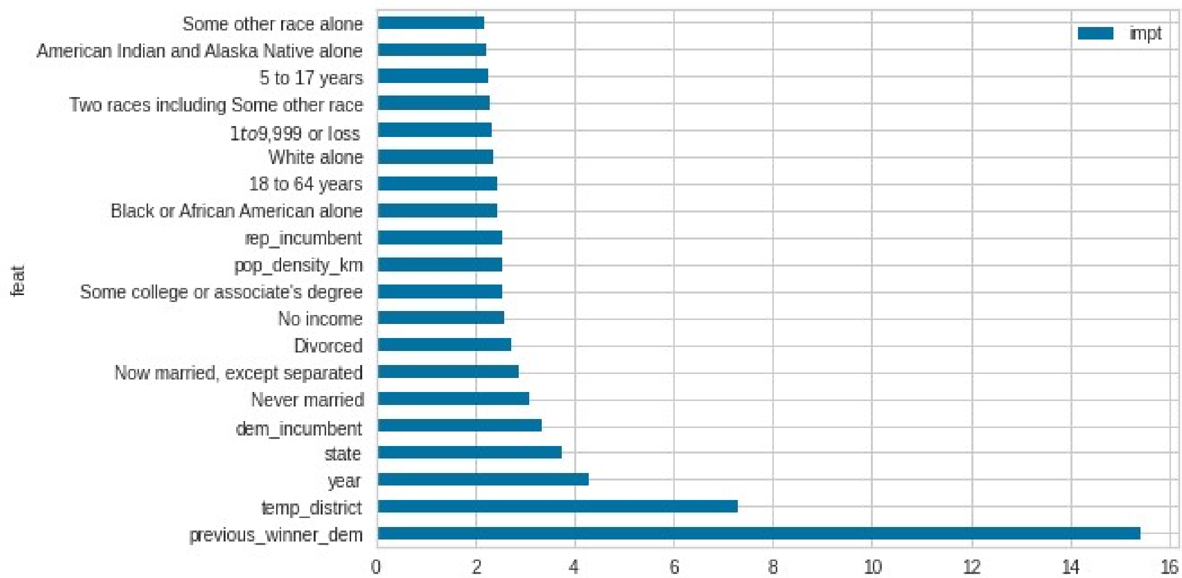

```
Test Results
              precision    recall  f1-score   support

          -1       0.79      0.94      0.86       980
           0       0.00      0.00      0.00       473
           1       0.69      0.96      0.80       683

    accuracy                           0.74      2136
   macro avg       0.49      0.64      0.55      2136
weighted avg       0.58      0.74      0.65      2136
```




```
Holdout Results
              precision    recall  f1-score   support

          -1       0.59      1.00      0.74        33
           0       0.00      0.00      0.00        26
           1       0.93      0.97      0.95        70

    accuracy                           0.78       129
   macro avg       0.51      0.66      0.56       129
weighted avg       0.66      0.78      0.71       129
```



In [99]:
```
# feature importances

importances = sorted(list(zip(cb.get_feature_importance(), X.columns)))

impts = pd.DataFrame(importances, columns=['impt', 'feat'])
impts = impts.set_index('feat')
impts = impts.sort_values(['impt'], ascending=False)
# graph 20 most impactful words to given model
impts.iloc[0:20].plot(kind='barh')
```



In [140]:
```
from sklearn.preprocessing import scale

def class_feature_importance(X, Y, feature_importances):
    N, M = X_test.shape
    X = scale(X_test)

    out = {}
    for c in set(test_preds):
        out[c] = dict(
            zip(range(N), np.mean(X.test[test_preds==c, :], axis=0)*feature_importances)
        )

```

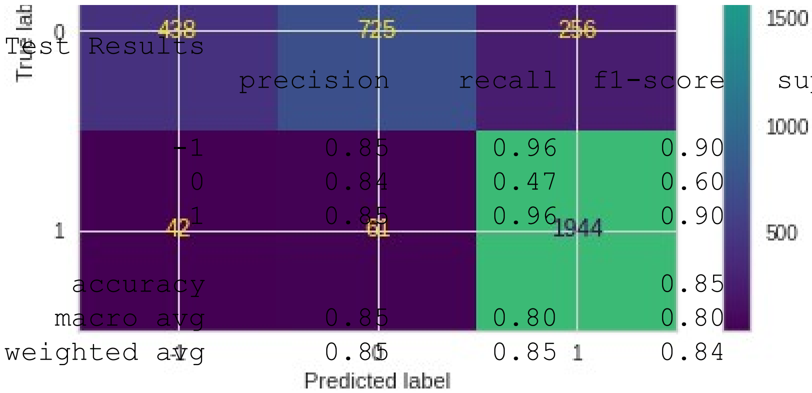
The model's now more generalizable but also undoubtedly worse. It's completely avoiding the toss-up option and is therefor useless.

Let's try working with Random Forest, another model that did pretty well when we first ran it.

In [103]:
```
params = [{'criterion': ['gini', 'entropy'],
          'bootstrap': [True, False],
          'max_depth': [10, 50, 100],
          'max_features': ['auto', 'sqrt'],
```

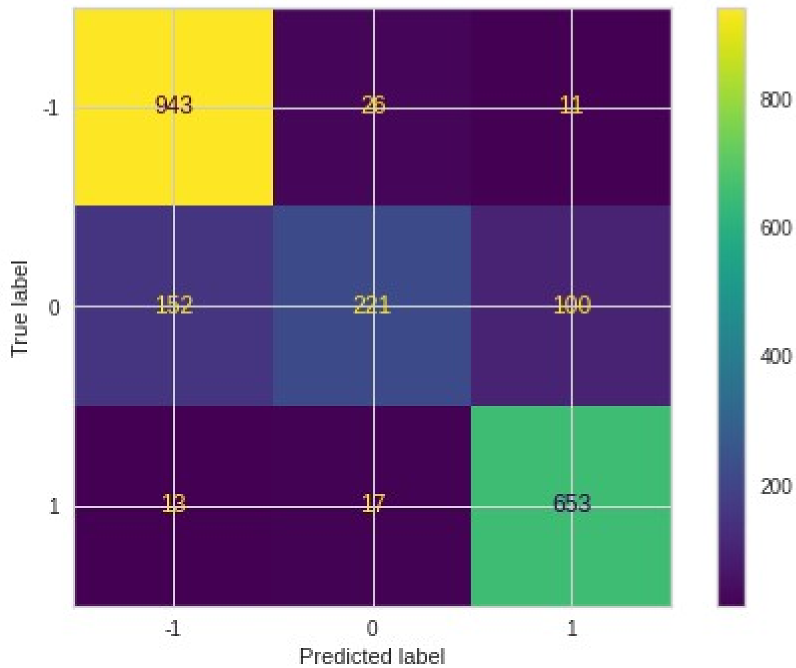
```
        'min_samples_split': [2, 20, 40],
        'n_estimators': [10, 100, 1000]
        }]

grid_search = RandomizedSearchCV(estimator=RandomForestClassifier(),
            param_distributions=params,
            scoring='recall_micro',
            n_jobs=-1,
            verbose =4
            )

pipe = Pipeline([
        ('scaler', StandardScaler()),
        ("classifier", grid_search)
    ])

start = datetime.now()
pipe.fit(X, y)
print('Fit time: ', datetime.now() - start)

train_preds = pipe.predict(X_train)
test_preds = pipe.predict(X_test)
holdout_preds = pipe.predict(X_holdout)

print('Train Results')

print(classification_report(y_train, train_preds))
plot_confusion_matrix(pipe, X_train, y_train)
plt.show()

print('Test Results')

print(classification_report(y_test, test_preds))
plot_confusion_matrix(pipe, X_test, y_test)
plt.show()

print('Holdout Results')

print(classification_report(y_holdout, holdout_preds))
plot_confusion_matrix(pipe, X_holdout, y_holdout)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Fit time:  0:16:35.883930
Train Results
              precision    recall  f1-score   support

          -1       0.86      0.96      0.91      2942
           0       0.86      0.51      0.64      1419
           1       0.87      0.95      0.91      2047

    accuracy                           0.86      6408
   macro avg       0.86      0.81      0.82      6408
weighted avg       0.86      0.86      0.85      6408
```



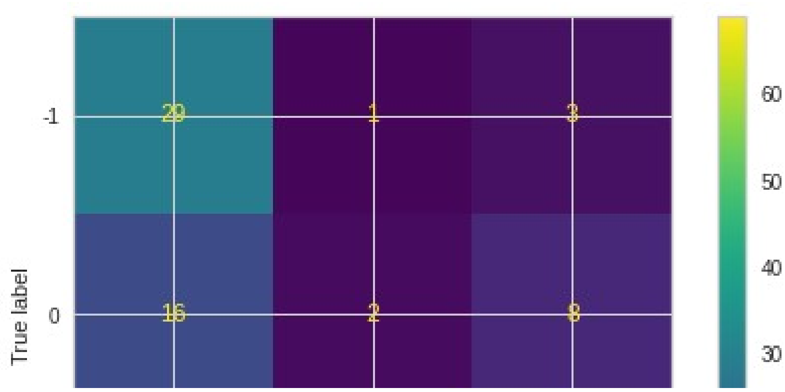
Test Results

	precision	recall	f1-score	support
-1	0.85	0.96	0.90	980
0	0.84	0.47	0.60	473
1	0.85	0.96	0.90	683
accuracy			0.85	2136
macro avg	0.85	0.80	0.80	2136
weighted avg	0.85	0.85	0.84	2136



Holdout Results

	precision	recall	f1-score	support
-1	0.63	0.88	0.73	33
0	0.67	0.08	0.14	26
1	0.86	0.99	0.92	70
accuracy			0.78	129
macro avg	0.72	0.65	0.60	129
weighted avg	0.76	0.78	0.71	129



The improvements here, if they can be called that, are extremely marginal. Let's see if manual adjustments can do better.

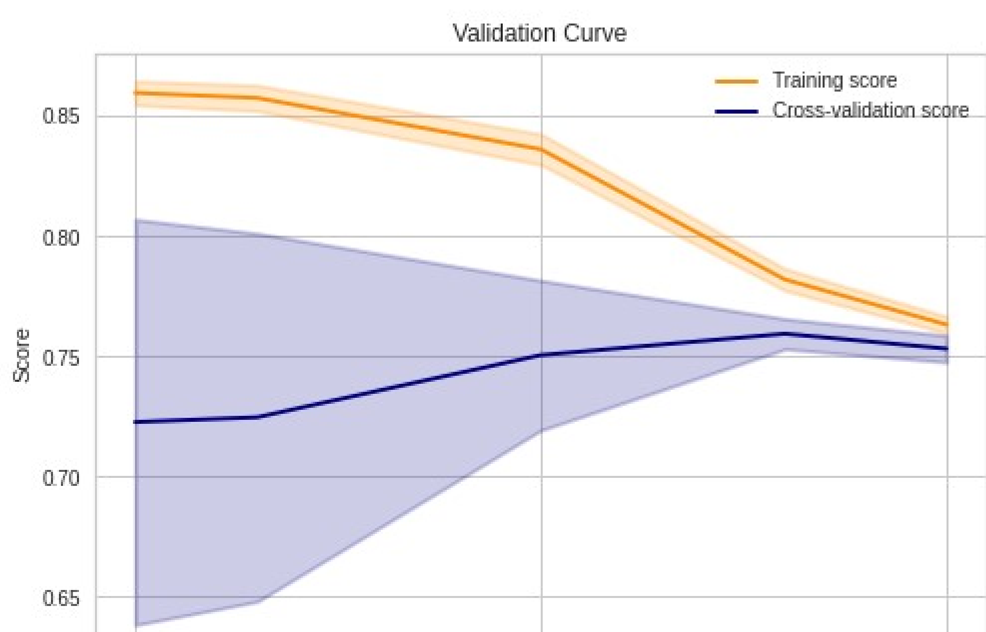
In [104]:

Out[104]:

```
{'bootstrap': True,
 'criterion': 'entropy',
 'max_depth': 100,
 'max_features': 'sqrt',
 'min_samples_split': 40,
 'n_estimators': 1000}
```

In [105]:

```
pipe = Pipeline([
('scaler', StandardScaler())])


param_range = [1, 2, 10, 40, 100]

rf = RandomForestClassifier().set_params(**grid_search.best_params_)

train_scores, test_scores = validation_curve(estimator = rf,
                                             X = pipe.fit_transform(X, y),
                                             y = y,
                                             param_name='min_samples_leaf',
                                             param_range=param_range)

train_scores_mean = np.mean(train_scores, axis=1)
train_scores_std = np.std(train_scores, axis=1)
test_scores_mean = np.mean(test_scores, axis=1)
test_scores_std = np.std(test_scores, axis=1)

plt.title("Validation Curve")

plt.ylabel("Score")

lw = 2
plt.semilogx(param_range, train_scores_mean, label="Training score",
             color="darkorange", lw=lw)
plt.fill_between(param_range, train_scores_mean - train_scores_std,
                 train_scores_mean + train_scores_std, alpha=0.2,
                 color="darkorange", lw=lw)
plt.semilogx(param_range, test_scores_mean, label="Cross-validation score",
             color="navy", lw=lw)
plt.fill_between(param_range, test_scores_mean - test_scores_std,
                 test_scores_mean + test_scores_std, alpha=0.2,
                 color="navy", lw=lw)
plt.legend(loc="best")
```

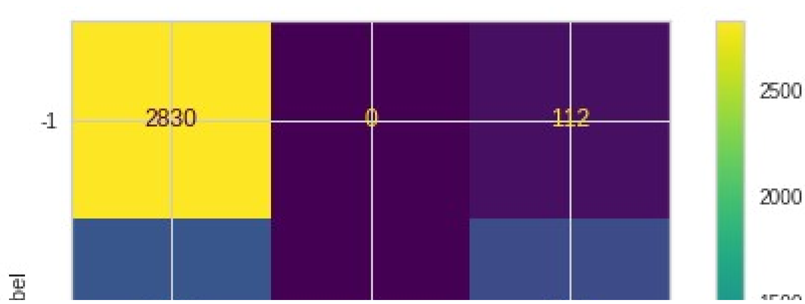


It looks like the cross-validation score peaks around the 40 parameter. Let's run it!

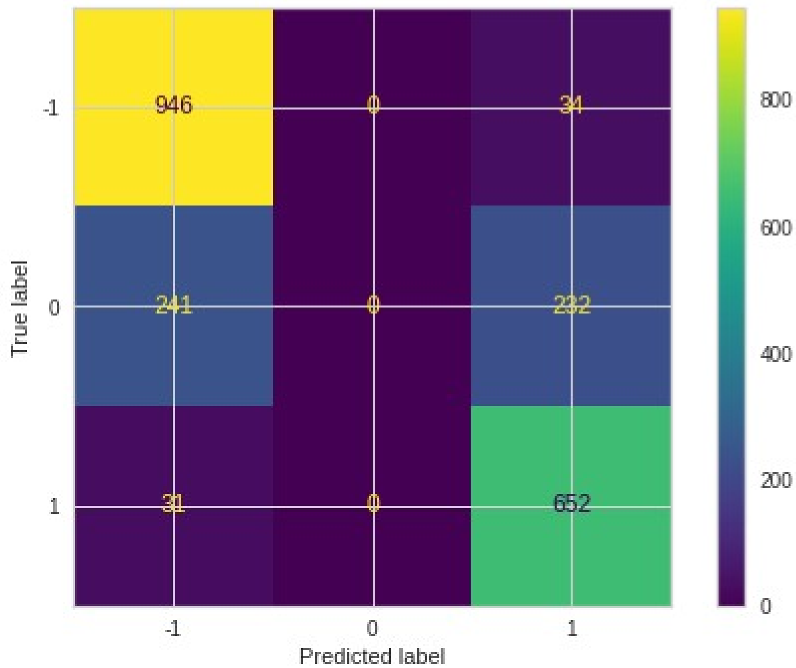
In [124]:

```
rf = 
RandomForestClassifier(min_samples_leaf=400).set_params(**grid_search
best_params_)

pipe = Pipeline([
        ('scaler', StandardScaler()),
        ("classifier", rf)
    ])

start = datetime.now()
pipe.fit(X_train, y_train)
print('Fit time: ', datetime.now() - start)

train_preds = pipe.predict(X_train)
test_preds = pipe.predict(X_test)
holdout_preds = pipe.predict(X_holdout)

print('Train Results')

print(classification_report(y_train, train_preds))
plot_confusion_matrix(pipe, X_train, y_train)
plt.show()

print('Test Results')

print(classification_report(y_test, test_preds))
plot_confusion_matrix(pipe, X_test, y_test)
plt.show()

print('Holdout Results')

print(classification_report(y_holdout, holdout_preds))
plot_confusion_matrix(pipe, X_holdout, y_holdout)
```

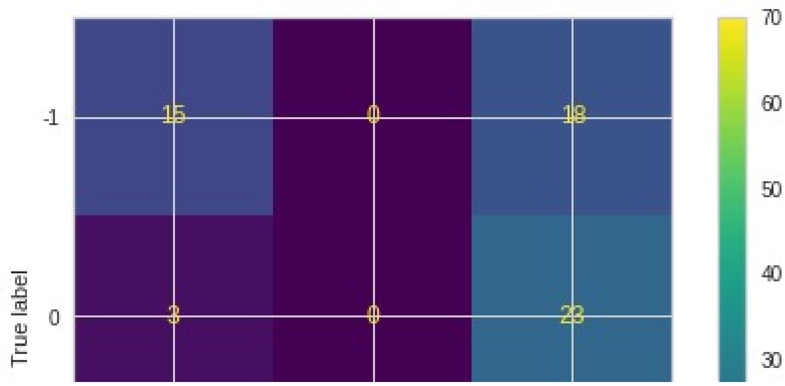
```
Fit time:  0:00:17.348502
Train Results
              precision    recall  f1-score   support

          -1       0.77      0.96      0.85      2942
           0       0.00      0.00      0.00      1419
           1       0.72      0.95      0.82      2047

    accuracy                           0.75      6408
   macro avg       0.49      0.64      0.56      6408
weighted avg       0.58      0.75      0.65      6408
```




```
Test Results
              precision    recall  f1-score   support

          -1       0.78      0.97      0.86       980
           0       0.00      0.00      0.00       473
           1       0.71      0.95      0.81       683

    accuracy                           0.75      2136
   macro avg       0.50      0.64      0.56      2136
weighted avg       0.58      0.75      0.66      2136
```




```
Holdout Results
              precision    recall  f1-score   support

          -1       0.83      0.45      0.59        33
           0       0.00      0.00      0.00        26
           1       0.63      1.00      0.77        70

    accuracy                           0.66       129
   macro avg       0.49      0.48      0.45       129
weighted avg       0.56      0.66      0.57       129
```



The scores are across the board worse. While we have fixed our overfitting problem we're once again seeing a model chose to ignore the smallest class, the toss-ups.

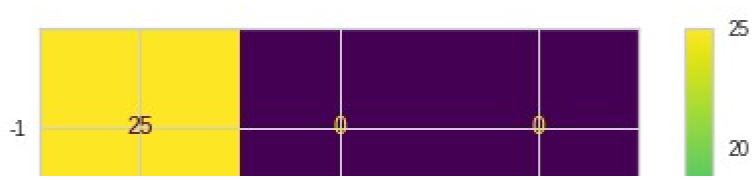
One final thing I want to try is simply modeling each state individually to see if those work better than one national model.

In [126]:

```
state_models = pd.DataFrame()

models = [
            DecisionTreeClassifier(),
            RandomForestClassifier(),
            LogisticRegression(),
            xgb.XGBClassifier(),
            lgb.LGBMClassifier(),
            CatBoostClassifier(verbose=False)
            ]


for state in sample['state'].unique():
    print(state)
    X = sample[sample['state'] == state].drop(["census_matching_temp_district", 'district', 'dem_vote'] axis=1)
    y = pd.Series(sample[sample['state'] == state]['dem_vote'])
    if state == 'ny':
        X_holdout = holdout_data.drop(['dem_net_vote'], axis=1)
        y_holdout = holdout_data['dem_net_vote']

    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, random_state=12)


    for model in models:


        pipe = Pipeline([
            ('feature_selection', BorutaFeatureSelection()),
            ('scaler', StandardScaler()),
            ("classifier", model)
        ])

        start = datetime.now()
        pipe.fit(X_train, y_train)
        print('Fit time: ', datetime.now() - start)

        train_preds = pipe.predict(X_train)
        test_preds = pipe.predict(X_test)



        print('Train Results')

        print(classification_report(y_train, train_preds))
        plot_confusion_matrix(pipe, X_train, y_train)
```

```
        plt.show()

        print('Test Results')

        print(classification_report(y_test, test_preds))
        plot_confusion_matrix(pipe, X_test, y_test)
        plt.show()

        if state == 'ny':
            target_encoder =
ce.target_encoder.TargetEncoder().fit(X[['state', 'temp_district']],
y)
            Xtr = b.target_encoder.transform(X_holdout[['state',
'temp_district']])
            X_holdout['state'] = Xtr['state']
            X_holdout['temp_district'] = Xtr['temp_district']

            holdout_preds = pipe.predict(X_holdout)
            print('Holdout Results')

            print(classification_report(y_holdout, holdout_preds))
            plot_confusion_matrix(pipe, X_holdout, y_holdout)
            plt.show()
        state_models = state_models.append(pd.Series({'estimator':
pipe['classifier'], 'test_recall': recall_score(y_test, test_preds,
average='micro'), 'state': state
                                                                    }),
```

```
az
Fit time:  0:00:19.393298
Train Results
              precision    recall  f1-score   support

          -1       1.00      1.00      1.00        25
           0       0.92      1.00      0.96        11
           1       1.00      0.91      0.95        11

    accuracy                           0.98        47
   macro avg       0.97      0.97      0.97        47
weighted avg       0.98      0.98      0.98        47
```



In [127]:

Out[127]:

	estimator	state	test_recall
0	DecisionTreeClassifier()	az	0.812500
1	(DecisionTreeClassifier(max_features='auto', r...	az	0.875000

	estimator	state	test_recall
2	LogisticRegression()	az	0.812500
3	XGBClassifier(objective='multi:softprob')	az	0.875000
4	LGBMClassifier()	az	0.812500
...	...	...	...
283	(DecisionTreeClassifier(max_features='auto', r...	ms	0.909091
284	LogisticRegression()	ms	0.818182
285	XGBClassifier(objective='multi:softprob')	ms	0.818182
286	LGBMClassifier()	ms	0.636364
287	<catboost.core.CatBoostClassifier object at 0x...	ms	0.818182

In [133]:

```
best_models = pd.DataFrame()

for state in sample['state'].unique():
    row = state_models[state_models['state'] ==
state].sort_values(['test_recall'], ascending=False).iloc[0]
```

In [136]:

Out[136]:

	estimator	state	test_recall
199	(DecisionTreeClassifier(max_features='auto', r...	tn	0.939394
16	LGBMClassifier()	ca	0.934783
163	(DecisionTreeClassifier(max_features='auto', r...	ok	0.923077
282	DecisionTreeClassifier()	ms	0.909091
215	<catboost.core.CatBoostClassifier object at 0x...	ut	0.909091
44	LogisticRegression()	ga	0.909091
109	(DecisionTreeClassifier(max_features='auto', r...	mo	0.905660
49	(DecisionTreeClassifier(max_features='auto', r...	id	0.900000
207	XGBClassifier(objective='multi:softprob')	tx	0.897436
259	(DecisionTreeClassifier(max_features='auto', r...	nj	0.892857
138	DecisionTreeClassifier()	ny	0.886364
62	LogisticRegression()	in	0.880000
97	(DecisionTreeClassifier(max_features='auto', r...	mi	0.879518
1	(DecisionTreeClassifier(max_features='auto', r...	az	0.875000
155	<catboost.core.CatBoostClassifier object at 0x...	nd	0.869565
190	LGBMClassifier()	sc	0.863636
253	(DecisionTreeClassifier(max_features='auto', r...	hi	0.857143
225	XGBClassifier(objective='multi:softprob')	wa	0.857143

	estimator	state	test_recall
115	(DecisionTreeClassifier(max_features='auto', r...	mt	0.854839
177	XGBClassifier(objective='multi:softprob')	pa	0.844156
265	(DecisionTreeClassifier(max_features='auto', r...	va	0.842105
31	(DecisionTreeClassifier(max_features='auto', r...	de	0.842105
248	LogisticRegression()	ak	0.833333
157	(DecisionTreeClassifier(max_features='auto', r...	oh	0.833333
58	LGBMClassifier()	il	0.829268
235	(DecisionTreeClassifier(max_features='auto', r...	wi	0.818182
93	XGBClassifier(objective='multi:softprob')	ma	0.818182
145	(DecisionTreeClassifier(max_features='auto', r...	nc	0.813559
20	LogisticRegression()	co	0.804878
277	(DecisionTreeClassifier(max_features='auto', r...	md	0.800000
270	DecisionTreeClassifier()	al	0.800000
29	<catboost.core.CatBoostClassifier object at 0x...	ct	0.800000
73	(DecisionTreeClassifier(max_features='auto', r...	ks	0.790323
105	XGBClassifier(objective='multi:softprob')	mn	0.780702
169	(DecisionTreeClassifier(max_features='auto', r...	or	0.771429
70	LGBMClassifier()	ia	0.769231
131	<catboost.core.CatBoostClassifier object at 0x...	nh	0.763158
7	(DecisionTreeClassifier(max_features='auto', r...	ar	0.760000
122	LogisticRegression()	nv	0.750000
133	(DecisionTreeClassifier(max_features='auto', r...	nm	0.740741
240	DecisionTreeClassifier()	wy	0.739130
37	(DecisionTreeClassifier(max_features='auto', r...	fl	0.714286
221	<catboost.core.CatBoostClassifier object at 0x...	vt	0.689655
85	(DecisionTreeClassifier(max_features='auto', r...	me	0.679245
83	<catboost.core.CatBoostClassifier object at 0x...	ky	0.666667
192	DecisionTreeClassifier()	sd	0.652174
228	DecisionTreeClassifier()	wv	0.550000

In [137]:

```
[illegible]
```

Out[137]:

	test_recall
count	48.000000
mean	0.812041
std	0.091557

	test_recall
min	0.538462
25%	0.767713
50%	0.831301
75%	0.879639
max	0.939394

The results here are very much mixed. The mean micro recall (and, due to the multiclass nature of the problem, the accuracy) is lower than we had for our best RandomForestClassifier. Though certain states did outperform this, we can't conclude that individual state models would outperform one national model.

Conclusion

Election modeling is a challenging field with near endless directions to explore. What's clear from my analysis is that demographics and certain types of political context are useful in understanding how an election might go, it's far from the whole story. With the vast and ever growing ways people's behavior is tracked and analyzed the amount of features that could be considered useful is hard to imagine. While my models definitely passed their benchmark of 46% accuracy, their inability to reliably identify toss-ups indicates there's still much more work to be done.

Further steps include:

- more exhaustive data manipulation including Feature Unions
- expanding the number of years analyzed
- adding demographic data from more sources, especially commercial ones
- analysis of the impact that other simultaneous races may have