Import what we need.

In [64]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model as lm
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.svm import SVR
from sklearn.preprocessing import StandardScaler
import warnings #this is to turn off DataConversionWarnings from
#normalizing in Sklearn from sklearn.exceptions import DataConversionWarning warnings.
#filterwarnings(action='ignore', category=DataConversionWarning)
```

This is a classification problem. Logistic regression or Random Forest would be two great ways to go. I will start with a logistic regression. We can also look to Random Forest if time allows. To create a different version of the logistic regression as per the problem instructions, we can use one version with certain features, and one version with other features. We will split the data into a train / validation / trest. We can use a cross_val score for our validation tuning.

Load and clean data.

In [65]:
```
voters = pd.read_csv("model_data.csv")
```

In [66]:
```
voters.shape
```

Out[66]:
```
(12325, 29)
```

In [67]:
```
pd.options.display.max_columns = 30
voters.head()
```

Out[67]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburbanp
0	1	0.0	3	2	0	69.0	2	1.000000	
1	2	1.0	3	1	0	43.0	1	1.000000	
2	3	NaN	1	7	1	52.0	1	0.167832	
3	4	NaN	1	3	1	38.0	5	0.167832	
4	5	1.0	3	2	1	60.0	1	1.000000	

In [68]:
```
voters.support_democrat.isna().sum()
```

Out[68]:
```
3719
```

In [69]:
```
#percentage of nulls
print(round((voters.support_democrat.isna().sum() / 12325)*100,2))
```

```
30.17
```

You can fillna with an average or any type of assumption you want when dealing with N/A's, but here, my issue is that the label, the dependent variable, is support_democrat. So I can't train a model on this. This might be

handly later on as a test of the model, but for now it will stand in the way of training. Further after we take out the 3719 nulls we'll have a lot of data left.

In [70]:

```
voters2 = voters.dropna(subset=['support_democrat'])
print(voters2.shape)
voters2.head()
```

```
(8606, 29)
```

Out[70]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburbanp
0	1	0.0	3	2	0	69.0	2	1.0	0.000(
1	2	1.0	3	1	0	43.0	1	1.0	0.000(
4	5	1.0	3	2	1	60.0	1	1.0	0.000(
5	6	0.0	2	8	1	37.0	5	0.0	0.695
6	7	1.0	3	5	0	60.0	1	1.0	0.000(

We need the data to be continuous or binary, so we have to hot-code the categoricals.

In [71]:

```
voters3 = pd.get_dummies(voters2)
print(voters3.columns)
voters3.cong_district_region.value_counts()
```

```
Index(['id', 'support_democrat', 'cong_district_region', 'occupation_code',
       'gender_female', 'age', 'party', 'census_urbanpcnt',
       'census_suburbanpcnt', 'census_ruralpcnt', 'census_collegepcnt',
       'census_unemprate', 'census_medianincome', 'density_sq_km',
       'on_email_list', 'avg_dem_performance', 'pet_owner', 'golf', 'hunting',
       'random', 'likes_cheese', 'protestant', 'catholic', 'jewish', 'afam',
       'latino', 'id_d', 'id_r', 'score_demo'],
      dtype='object')
```

Out[71]:

```
2    3857
3    2856
1    1893
Name: cong_district_region, dtype: int64
```

We have not caught all the categoricals because some categoricals already listed numbers. We have to deal with congressional distrcit, occupational code, and party.

In [72]:
```
voters3['occupat_managerial'] = voters3['occupation_code'].apply(lambda x: 1 if x == 1 else 0)
voters3['occupat_professional'] = voters3['occupation_code'].apply(lambda x: 1 if x == 2 else 0)
voters3['occupat_service'] = voters3['occupation_code'].apply(lambda x: 1 if x == 3 else 0)
voters3['occupat_clerical'] = voters3['occupation_code'].apply(lambda x: 1 if x == 4 else 0)
voters3['occupat_technical'] = voters3['occupation_code'].apply(lambda x: 1 if x == 5 else 0)
voters3['occupat_agriculture'] = voters3['occupation_code'].apply(lambda x: 1 if x == 6 else 0)
voters3['occupat_industrial'] = voters3['occupation_code'].apply(lambda x: 1 if x == 7 else 0)
voters3['occupat_technology'] = voters3['occupation_code'].apply(lambda x: 1 if x == 8 else 0)
voters3['occupat_retail'] = voters3['occupation_code'].apply(lambda x: 1 if x == 9 else 0)
voters3['occupat_other'] = voters3['occupation_code'].apply(lambda x: 1 if x == 0 else 0)
voters3['cong_district_1'] = voters3['cong_district_region'].apply(lambda x: 1 if x == 1 else 0)
voters3['cong_district_2'] = voters3['cong_district_region'].apply(lambda x: 1 if x == 2 else 0)
voters3['cong_district_3'] = voters3['cong_district_region'].apply(lambda x: 1 if x == 3 else 0)
voters3['party_dem'] = voters3['party'].apply(lambda x: 1 if x == 1 else 0)
voters3['party_repub'] = voters3['party'].apply(lambda x: 1 if x == 2 else 0)
voters3['party_green'] = voters3['party'].apply(lambda x: 1 if x == 3 else 0)
voters3['party_libert'] = voters3['party'].apply(lambda x: 1 if x == 4 else 0)
voters3['party_ind'] = voters3['party'].apply(lambda x: 1 if x == 5 else 0)

pd.options.display.max_columns = 60
voters3.head()
```

Out[72]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburbanp
0	1	0.0	3	2	0	69.0	2	1.0	0.000
1	2	1.0	3	1	0	43.0	1	1.0	0.000
4	5	1.0	3	2	1	60.0	1	1.0	0.000
5	6	0.0	2	8	1	37.0	5	0.0	0.695
6	7	1.0	3	5	0	60.0	1	1.0	0.000

In [73]:
```
#one more NaN check
print(voters3.gender_female.isna().sum())
print(voters3.census_medianincome.isna().sum())
print(voters3.age.isna().sum())
print(voters3.party_dem.isna().sum())
#you can decided to fill with an average. because it's so small I am going to drop it.
voters4 = voters3.dropna(subset=['census_medianincome','age'])
print(voters4.gender_female.isna().sum())
print(voters4.census_medianincome.isna().sum())
print(voters4.age.isna().sum())
print(voters4.party_dem.isna().sum())
```

```
0
13
21
0
0
0
0
0
```

Split up the data. We will start with a few simple features. If we had more time we will test more.

In [74]:
```
features = voters4[['gender_female', 'census_medianincome', 'age']]
outcome = voters4[['support_democrat']]
```

In [75]:
```
#get test set
intermediate_features, test_features, intermediate_labels, test_labels = train_test_split(features
```

In [76]:
```
#get validation and train set
train_features, validation_features, train_labels, validation_labels = train_test_split(intermedi
```

In [77]:
```
#check
print("train")
print(train_features.shape)
print(train_labels.shape)
print("validate")
print(validation_features.shape)
print(validation_labels.shape)
print("test - hold till very end")
print(test_features.shape)
print(test_labels.shape)
```

```
train
(5485, 3)
(5485, 1)
validate
(1372, 3)
(1372, 1)
test - hold till very end
(1715, 3)
(1715, 1)
```

Normalize and train model.

In [78]:
```
#normalize this, since sklearn's logistic regression uses regularization
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    scaler = StandardScaler()
    train_features = scaler.fit_transform(train_features)
    validation_features = scaler.transform(validation_features) #we do NOT want to fit to the vali
log_model = LogisticRegression(solver="liblinear") #to remove warning
#print('Accuracy Score: {}'.format(log_model.score(train_features, train_labels)))
```

In [79]:
```
log_model.fit(train_features, train_labels.values.ravel()) #the ravel removes an error mesage her
log_model.score(train_features, train_labels)
#I will address the reshaping if there is time
```

Out[79]:
```
0.5560619872379216
```

In [80]:
```
#We also score on the validation to see if we are overfitting or under fitting.
log_model.score(validation_features, validation_labels)
```

Out[80]:
```
0.5364431486880467
```

Cross validation check

In [81]:
```
print("Cross-Validation Scoring") #need to add solver="liblinear" #to remove warning
print('Accuracy Score: {}'.format(round(cross_val_score(LogisticRegression(solver="liblinear"), t
```

```
Cross-Validation Scoring
Accuracy Score: 0.556
```

We can check if with a better selection of features, the model does better.

In [82]:
```
features_v2 = voters4[['gender_female', 'census_medianincome', 'age','party_dem']]
outcome_v2 = voters4[['support_democrat']]
```

In [83]:
```
intermediate_features_v2, test_features_v2, intermediate_labels_v2, test_labels_v2 = train_test_s
```

In [84]:
```
train_features_v2, validation_features_v2, train_labels_v2, validation_labels_v2 = train_test_spl
```

In [85]:
```python
#check
print("train_v2")
print(train_features_v2.shape)
print(train_labels_v2.shape)
print("validate_v2")
print(validation_features_v2.shape)
print(validation_labels_v2.shape)
print("test - hold till very end_v2")
print(test_features_v2.shape)
print(test_labels_v2.shape)
```

```
train_v2
(5485, 4)
(5485, 1)
validate_v2
(1372, 4)
(1372, 1)
test - hold till very end_v2
(1715, 4)
(1715, 1)
```

In [86]:
```python
#normalize this, since sklearn's logistic regression uses regularization
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    scaler = StandardScaler()
    train_features_v2 = scaler.fit_transform(train_features_v2)
    validation_features_v2 = scaler.transform(validation_features_v2) #we do NOT want to fit to t
log_model_v2 = LogisticRegression(solver="liblinear") #to remove warning
```

In [87]:
```python
log_model_v2.fit(train_features_v2, train_labels_v2.values.ravel())
log_model_v2.score(train_features_v2, train_labels_v2)
```

Out[87]:
```
0.8264357338195077
```

In [88]:
```python
#We also score on the validation to see if we are overfitting or under fitting.
log_model_v2.score(validation_features_v2, validation_labels_v2)
```

Out[88]:
```
0.814868804664723
```

In [89]:
```python
print("Cross-Validation Scoring_v2") #need to add solver="liblinear" #to remove warning
print('Accuracy Score_v2: {}'.format(round(cross_val_score(LogisticRegression(solver="liblinear")
```

```
Cross-Validation Scoring_v2
Accuracy Score_v2: 0.826
```

We can see that adding in the Dem Party as a feature drastically improved the model's predicatability, of course this is to be expected. This was just to show how I might go about modeling. With more time I'd also use a .reshape(-1,1) method to fix the error box.

In [97]:
```python
#let's try to get a data frame together with the model score for everything
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    voters4["model_v2_score"]=log_model_v2.predict_proba(scaler.transform(features_v2))[:,1]
```

In [29]:
```
voters4.head()
print(voters4.model_v2_score.head())
print(voters4.model_v2_score.count())
print(voters4.shape)
```

```
0    0.193366
1    0.885680
4    0.875037
5    0.273198
6    0.868088
Name: model_v2_score, dtype: float64
8572
(8572, 48)
```

In [30]:
```
voter_model_scores_df = voters4[['id','model_v2_score']]
voter_model_scores_df.head()
```

Out[30]:

	id	model_v2_score
0	1	0.193366
1	2	0.885680
4	5	0.875037
5	6	0.273198
6	7	0.868088

In [31]:
```
scored_voters = pd.merge(voters, voter_model_scores_df, left_on='id', right_on='id', how='left')
#this left merges voter_model_scores ON TO voters
#so base table goes first
#merging-onto table goes second
```

In [32]:
```
scored_voters.head()
```

Out[32]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburbanp
0	1	0.0	3	2	0	69.0	2	1.000000	
1	2	1.0	3	1	0	43.0	1	1.000000	
2	3	NaN	1	7	1	52.0	1	0.167832	
3	4	NaN	1	3	1	38.0	5	0.167832	
4	5	1.0	3	2	1	60.0	1	1.000000	

In [33]:
```
scored_voters.to_csv("scored_output.csv",index=False)
```

In [34]:
```
voters.head()
```

Out[34]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburbanp
0	1	0.0	3	2	0	69.0	2	1.000000	
1	2	1.0	3	1	0	43.0	1	1.000000	
2	3	NaN	1	7	1	52.0	1	0.167832	
3	4	NaN	1	3	1	38.0	5	0.167832	
4	5	1.0	3	2	1	60.0	1	1.000000	

In [35]:
```
#df1 = df[df['Sales'] >= s]
unlabeled_df = voters[voters['support_democrat'].isnull()]
```

In [36]:
```
unlabeled_df.head()
```

Out[36]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburban
2	3	NaN	1	7	1	52.0	1	0.167832	0.00
3	4	NaN	1	3	1	38.0	5	0.167832	0.00
9	10	NaN	2	6	1	60.0	2	0.000000	0.71
10	11	NaN	3	2	0	47.0	5	1.000000	0.00
15	16	NaN	2	6	1	55.0	2	0.316286	0.00

In [37]:
```
unlabeled_df.isnull().values.any()
```

Out[37]:
```
True
```

In [38]:
```
#null_counts = unlabeled_df.isnull().sum()
#null_counts[null_counts > 0].sort_values(ascending=False)
#the below plugs the first line into the second
unlabeled_df.isnull().sum()[unlabeled_df.isnull().sum() > 0].sort_values(ascending=False)
```

Out[38]:
```
support_democrat        3719
density_sq_km             14
score_demo                11
census_unemprate          11
age                        8
census_collegepcnt         6
census_ruralpcnt           6
census_suburbanpcnt        6
census_urbanpcnt           6
census_medianincome        1
dtype: int64
```

In [39]:
```
unlabeled_df = unlabeled_df.dropna(subset=['density_sq_km','score_demo',\
                                            'census_unemprate','age','census_collegepcnt',\
                                           'census_ruralpcnt','census_suburbanpcnt',\
                                           'census_urbanpcnt','census_medianincome'])
```

In [40]:
```
unlabeled_df.isnull().sum()[unlabeled_df.isnull().sum() > 0].sort_values(ascending=False)
```

Out[40]:
```
support_democrat    3692
dtype: int64
```

In [41]:
```
unlabeled_df['party_dem'] = unlabeled_df['party'].apply(lambda x: 1 if x == 1 else 0)
```

In [42]:
```
unlabeled_df.head()
```

Out[42]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburban
2	3	NaN	1	7	1	52.0	1	0.167832	0.00
3	4	NaN	1	3	1	38.0	5	0.167832	0.00
9	10	NaN	2	6	1	60.0	2	0.000000	0.71
10	11	NaN	3	2	0	47.0	5	1.000000	0.00
15	16	NaN	2	6	1	55.0	2	0.316286	0.00

In [43]:
```
unlabeled_features = unlabeled_df[['gender_female', 'census_medianincome', 'age','party_dem']]
```

In [44]:
```
with warnings.catch_warnings():
    warnings.simplefilter("ignore")
    unlabeled_df["model_v2_score"]=log_model_v2.predict_proba\
    (scaler.transform(unlabeled_features))[:,1]
```

In [45]:
```
unlabeled_df.head()
```

Out[45]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburban
2	3	NaN	1	7	1	52.0	1	0.167832	0.00
3	4	NaN	1	3	1	38.0	5	0.167832	0.00
9	10	NaN	2	6	1	60.0	2	0.000000	0.71
10	11	NaN	3	2	0	47.0	5	1.000000	0.00
15	16	NaN	2	6	1	55.0	2	0.316286	0.00

In [46]:
```
unlabeled_model_scores_df = unlabeled_df[['id','model_v2_score']]
unlabeled_model_scores_df.head()
```

Out[46]:

	id	model_v2_score
2	3	0.895262
3	4	0.291095
9	10	0.251523
10	11	0.211985
15	16	0.190866

In [47]:
```
scored_voters_all = pd.merge(scored_voters, unlabeled_model_scores_df, left_on='id', right_on='id
```

In [48]:
```
scored_voters_all.head()
```

Out[48]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburbanp
0	1	0.0	3	2	0	69.0	2	1.000000	
1	2	1.0	3	1	0	43.0	1	1.000000	
2	3	NaN	1	7	1	52.0	1	0.167832	
3	4	NaN	1	3	1	38.0	5	0.167832	
4	5	1.0	3	2	1	60.0	1	1.000000	

In [49]:
```
#trying a differnent way...concat the voter socers Nan and non Nan first, then merge them in
```

In [50]:
```
score_file = pd.concat([voter_model_scores_df,unlabeled_model_scores_df])
```

In [51]:
```
score_file = score_file.sort_values(by=['id'])
```

In [52]:
```
score_file.head()
```

Out[52]:

	id	model_v2_score
0	1	0.193366
1	2	0.885680
2	3	0.895262
3	4	0.291095
4	5	0.875037

In [53]:
```
total_output_scores = pd.merge(voters, score_file, left_on='id',\
                                          right_on='id', how='left')
```

In [54]:
```
total_output_scores.head()
```
Out[54]:

	id	support_democrat	cong_district_region	occupation_code	gender_female	age	party	census_urbanpcnt	census_suburbanp
0	1	0.0	3	2	0	69.0	2	1.000000	
1	2	1.0	3	1	0	43.0	1	1.000000	
2	3	NaN	1	7	1	52.0	1	0.167832	
3	4	NaN	1	3	1	38.0	5	0.167832	
4	5	1.0	3	2	1	60.0	1	1.000000	

In [55]:
```
total_output_scores.to_csv("total_output_scores.csv",index=False)
```
In [57]:
```
total_output_scores.to_excel("total_output_scores.xlsx",index=False)
```
In [ ]:
```

```