Nathan He, Yushen Li, Samujjwaal Dey 11/18/2019
CS418 - Fall 2019

Project 2 Report

Task 1&2

```
#import dataset
original_election_data = pd.read_csv("merged_train.csv")
#original_election_data.head()
```

```
#import dataset
test_data = pd.read_csv("demographics_test.csv")
#test_data.head()
```

```
demo_data = original_election_data[original_election_data['Party'] == 1]
repub_data = original_election_data[original_election_data['Party'] != 1]
```

Task 1 : Partition dataset into training and validation sets ¶

```
#using Holdout method
demo_x_train, demo_x_val, demo_y_train, demo_y_val = train_test_split(demo_data.iloc[:,:-3], demo_data['Democratic'], test_size = 0.25, random_state=0)
repub_x_train, repub_x_val, repub_y_train, repub_y_val = train_test_split(repub_data.iloc[:,:-3], repub_data['Republican'], test_size = 0.25, random_state=0)
#demo_x_train.head()
#repub_x_train.iloc[:,3:]
```

```
x_train, x_val, y_train, y_val = train_test_split(original_election_data.iloc[:,:-3], original_election_data['Party'], test_size = 0.25, random_state=0)
```

First, we imported the dataset and we split the data into two datasets: demo_data for the democratic data and repub_data for republican, on the basis of the column 'Party'. Then we choose holdout method to partition the datasets with 75% of the data for training and 25% for testing and set the parameter random_state to 0.

Task 2 : Standardizing training and validation sets

[7]:
```
scaler = StandardScaler()
scaler.fit(demo_x_train.iloc[:,3:])
demo_x_train_scaled = scaler.transform(demo_x_train.iloc[:,3:])
demo_x_train_scaled = pd.DataFrame(demo_x_train_scaled)
demo_x_val_scaled = scaler.transform(demo_x_val.iloc[:,3:])
demo_x_val_scaled = pd.DataFrame(demo_x_val_scaled)
#demo_x_train_scaled
```

[8]:
```
scaler = StandardScaler()
scaler.fit(repub_x_train.iloc[:,3:])
repub_x_train_scaled = scaler.transform(repub_x_train.iloc[:,3:])
repub_x_train_scaled = pd.DataFrame(repub_x_train_scaled)
repub_x_val_scaled = scaler.transform(repub_x_val.iloc[:,3:])
repub_x_val_scaled = pd.DataFrame(repub_x_val_scaled)
```

[9]:
```
scaler.fit(x_train.iloc[:,3:])
x_train_scaled = scaler.transform(x_train.iloc[:,3:])
x_train_scaled = pd.DataFrame(x_train_scaled)
x_val_scaled = scaler.transform(x_val.iloc[:,3:])
x_val_scaled = pd.DataFrame(x_val_scaled)
```

[10]:
```
test_data_scaled = scaler.transform(test_data.iloc[:,3:])
test_data_scaled = pd.DataFrame(test_data_scaled)
```

Then we standardize the data and ignored the first 3 columns because they are categorical data attributes. This is because standardization can be performed on only numeric data attributes and the first three columns are not numeric.

Task 3. Regression Model

For Democratic

The best performing multiple linear regression model is when considering the attributes 'Total population','Percent Age 29 and Under','Percent Unemployed' and 'Less than Bachelor's Degree'. We get an R-Square of 0.9167 and adjusted R-Square of 0.9134.

Testing by selecting diffferent attributes

In [17]:
```
#cols = [0,1,3,6,7,8,10,11,12]
demo_cols = [0,6,9,11]
```

In [18]:
```
model = linear_model.LinearRegression()
demo_fitted_model = model.fit(X = demo_x_train_scaled.iloc[:,demo_cols], y = demo_y_train)
print("Coefficients are : ",demo_fitted_model.coef_)
print("\nIntercept is : ",demo_fitted_model.intercept_)
```

```
Coefficients are :  [103165.80239653  -6096.93418642   5266.84827431 -20585.67995092]

Intercept is :  67233.13168724279
```

In [19]:
```
predicted = demo_fitted_model.predict(demo_x_val_scaled.iloc[:, demo_cols])
predicted[predicted < 0] = 0
```

In [20]:
```
corr_coef = np.corrcoef(predicted,demo_y_val)[1,0]
print("Coefficient of Correlation = ",corr_coef)
R_squared = corr_coef ** 2
print("R square = ",R_squared)
adj_R_squared =1 - (1 - R_squared)*(len(demo_y_val)-1)/(len(demo_y_val)-len(demo_x_train_scaled.iloc[:, de
mo_cols].columns-1))
print("Adjusted R square = ",adj_R_squared)
```

```
Coefficient of Correlation =  0.9574454925112419
R square =  0.9167018711300946
Adjusted R square =  0.9134980969427906
```

For Republican

The best performing multiple linear regression model is when considering the attributes 'Total population','Percent white', 'Percent Age 29 and Under', 'Percentage Age 65 and Older', 'Median Household Income', 'Percent Unemployed', 'Less than Bachelor's Degree' and 'Percent Rural'.
We get an R-Square of 0.966 and adjusted R-Square of 0.9647.

Testing by selecting diffferent attributes

In [26]:
```
#cols = [0,1,3,6,7,8,10,11,12]
repub_cols = [0,1,6,7,8,9,10,11,12]
```

In [27]:
```
model = linear_model.LinearRegression()
repub_fitted_model = model.fit(X = repub_x_train_scaled.iloc[:,repub_cols], y = repub_y_train)
print("Coefficients are : ",repub_fitted_model.coef_)
print("\nIntercept is : ",repub_fitted_model.intercept_)
```

```
Coefficients are :  [20913.28053894   519.9673001   -800.41422896  1152.1470663
  1607.28872585   326.25606032   766.35971703  -542.34904965
  -335.11290375]

Intercept is :  12655.983128834356
```

In [28]:
```
predicted = repub_fitted_model.predict(repub_x_val_scaled.iloc[:, repub_cols])
predicted[predicted < 0] = 0
```

In [29]:
```
corr_coef = np.corrcoef(predicted,repub_y_val)[1,0]
print("Coefficient of Correlation = ",corr_coef)
R_squared = corr_coef**2
print("R square = ",R_squared)
adj_R_squared =1 - (1 - R_squared)*(len(repub_y_val)-1)/(len(repub_y_val)-len(repub_x_train_scaled.iloc[:,
repub_cols].columns-1))
print("Adjusted R square = ",adj_R_squared)
```

```
Coefficient of Correlation =  0.982889063828049
R square =  0.9660709117927785
Adjusted R square =  0.9647721907130762
```

We performed LASSO regression for both cases but since the coefficients obtained are large, LASSO does not drop any variable and takes all attributes for training.

Task 4. Classification Methods

The two best performing classification models are SVM and K-Nearest Neighbors.

Support Vector Machine

The best model is SVM but it comes with the drawback of being computationally expensive. By including all the attributes, we get result of 0.8562 accuracy and F1 score of 0.909 & 0.65. However, including all the attributes is not a good idea.

The parameters used are : kernel = 'rbf', gamma = 'auto'

Using all attributes

```
In [41]: classifier = SVC(kernel = 'rbf', gamma = 'auto')
         classifier.fit(x_train_scaled,y_train)

Out[41]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)

In [42]: y_pred = classifier.predict(x_val_scaled)

In [44]: accuracy = metrics.accuracy_score(y_val,y_pred)
         error = 1 - accuracy
         precision = metrics.precision_score(y_val, y_pred, average = None)
         recall = metrics.recall_score(y_val, y_pred, average = None)
         F1_score = metrics.f1_score(y_val, y_pred, average = None)
         print("Accuracy = ",accuracy, "\nError = ",error, "\nPrecision =",precision,"\nRecall = ", recall,"\nF1 Sc
         ore = ", F1_score)

         Accuracy =  0.8561872909698997
         Error =  0.14381270903010035
         Precision = [0.85375494 0.86956522]
         Recall =  [0.97297297 0.51948052]
         F1 Score =  [0.90947368 0.6504065 ]
```

Therefore, we run several combinations with different attributes and less amount of them and the result (in the image below) shows that 'Percent Black, not Hispanic or Latino', 'Present Hispanic or Latino',' Percent Age 29 and Under', 'Percentage Age 65 and Older', 'Median Household Income', 'Percent Unemployed', 'Less than Bachelor's Degree' and 'Percent Rural' are the most important attributes.

The end result returns the highest accuracy of 0.8562 and F1 score of 0.9087 & 0.6614. We tried different combinations and the end result of none of them are as strong as this particular combination of attributes.

Testing by selecting diffferent attributes

```
In [45]: SVM_cols = [2,3,6,7,8,9,11,12]
         SVM_classifier = SVC(kernel = 'rbf', gamma = 'auto')
         SVM_classifier.fit(x_train_scaled.iloc[:,SVM_cols],y_train)

Out[45]: SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
             decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
             max_iter=-1, probability=False, random_state=None, shrinking=True,
             tol=0.001, verbose=False)

In [46]: y_pred = SVM_classifier.predict(x_val_scaled.iloc[:,SVM_cols])

In [47]: accuracy = metrics.accuracy_score(y_val,y_pred)
         error = 1 - accuracy
         precision = metrics.precision_score(y_val, y_pred, average = None)
         recall = metrics.recall_score(y_val, y_pred, average = None)
         F1_score = metrics.f1_score(y_val, y_pred, average = None)
         print("Accuracy = ",accuracy, "\nError = ",error, "\nPrecision =",precision,"\nRecall = ", recall,"\nF1 Sc
         ore = ", F1_score)

         Accuracy =  0.8561872909698997
         Error =  0.14381270903010035
         Precision = [0.85943775 0.84      ]
         Recall =  [0.96396396 0.54545455]
         F1 Score =  [0.90870488 0.66141732]
```

K-Nearest Neighbors

For K-nearest neighbors, we did similar tasks by including all the attribute first and we selected the total number of neighbors here to find out what is the best way to classify them. Here, we found out when n_neighbor is 11, the accuracy and F1 Score are the highest (Image down below). The parameter 'weights' was also set to 'distance' to give higher priority to closer points.

k Nearest

```
In [861]: classifier = KNeighborsClassifier(n_neighbors = 11, weights ='distance')
          classifier.fit(x_train_scaled,y_train)

Out[861]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                               metric_params=None, n_jobs=None, n_neighbors=11, p=2,
                               weights='distance')

In [862]: y_pred = classifier.predict(x_val_scaled)
```

```
cols = [3,5,6,7,8,9,11,12]
for i in range(1,20):
    classifier = KNeighborsClassifier(n_neighbors = i,weights='distance')
    classifier.fit(x_train_scaled.iloc[:,cols],y_train)
    y_pred = classifier.predict(x_val_scaled.iloc[:,cols])
    accuracy = metrics.accuracy_score(y_val,y_pred)
    error = 1 - accuracy
    precision = metrics.precision_score(y_val, y_pred, average = None)
    recall = metrics.recall_score(y_val, y_pred, average = None)
    F1_score = metrics.f1_score(y_val, y_pred, average = None)
    print([i,accuracy, error, precision, recall, F1_score])
```

```
[1, 0.7692307692307693, 0.23076923076923073, array([0.83406114, 0.55714286]), array([0.86036036, 0.50649351]), array([0.84700665, 0.53061224])]
[2, 0.7692307692307693, 0.23076923076923073, array([0.83406114, 0.55714286]), array([0.86036036, 0.50649351]), array([0.84700665, 0.53061224])]
[3, 0.8193979933110368, 0.1806020066889632, array([0.836     , 0.73469388]), array([0.94144144, 0.46753247]), array([0.88559322, 0.57142857])]
[4, 0.802675585284281, 0.19732441471571904, array([0.83265306, 0.66666667]), array([0.91891892, 0.46753247]), array([0.87366167, 0.54961832])]
[5, 0.8093645484949833, 0.1906354515050167, array([0.83673469, 0.68518519]), array([0.92342342, 0.48051948]), array([0.87794433, 0.5648855 ])]
[6, 0.8160535117056856, 0.18394648829431437, array([0.83805668, 0.71153846]), array([0.93243243, 0.48051948]), array([0.88272921, 0.57364341])]
[7, 0.8260869565217391, 0.17391304347826086, array([0.84274194, 0.74509804]), array([0.94144144, 0.49350649]), array([0.8893617, 0.59375  ])]
[8, 0.842809364548495, 0.15719063545150502, array([0.84860558, 0.8125    ]), array([0.95945946, 0.50649351]), array([0.90063425, 0.624     ])]
[9, 0.8561872909698997, 0.14381270903010035, array([0.85657371, 0.85416667]), array([0.96846847, 0.53246753]), array([0.90909091, 0.656     ])]
[10, 0.8394648829431438, 0.1605351170568562, array([0.8452381 , 0.80851064]), array([0.95945946, 0.49350649]), array([0.89873418, 0.61290323])]
[11, 0.8461538461538461, 0.15384615384615385, array([0.852     , 0.81632653]), array([0.95945946, 0.51948052]), array([0.90254237, 0.63492063])]
[12, 0.8394648829431438, 0.1605351170568562, array([0.8452381 , 0.80851064]), array([0.95945946, 0.49350649]), array([0.89873418, 0.61290323])]
[13, 0.842809364548495, 0.15719063545150502, array([0.8458498 , 0.82608696]), array([0.96396396, 0.49350649]), array([0.90105263, 0.61788618])]
[14, 0.842809364548495, 0.15719063545150502, array([0.84860558, 0.8125    ]), array([0.95945946, 0.50649351]), array([0.90063425, 0.624     ])]
[15, 0.842809364548495, 0.15719063545150502, array([0.8458498 , 0.82608696]), array([0.96396396, 0.49350649]), array([0.90105263, 0.61788618])]
[16, 0.8394648829431438, 0.1605351170568562, array([0.83984375, 0.8372093 ]), array([0.96846847, 0.46753247]), array([0.89958159, 0.6       ])]
[17, 0.842809364548495, 0.15719063545150502, array([0.84313725, 0.84090909]), array([0.96846847, 0.48051948]), array([0.90146751, 0.61157025])]
[18, 0.8394648829431438, 0.1605351170568562, array([0.83984375, 0.8372093 ]), array([0.96846847, 0.46753247]), array([0.89958159, 0.6       ])]
[19, 0.8294314381270903, 0.1705685618729097, array([0.83794466, 0.7826087 ]), array([0.95495495, 0.46753247]), array([0.89263158, 0.58536585])]
```

Then we tried to use different combinations of attributes instead of all the attributes to see which one is accurate, we conclude that 'Percent Hispanic or Latino', 'Percent Female', 'Percent Age 29 and under', 'Percent Age 65 and Older', 'Median Household Income', 'Percent Unemployed', 'Less than Bachelor's Degree', and 'Percent Rural' are the most important attributes to determine whether the state is democratic or republican with an accuracy of 0.8562 and F1 Score of 0.9087 & 0.6614.

We again ran the for loop like before and found that, K = 9 gives the best performance for this set of columns.

Testing by selecting diffferent attributes

```
In [37]: KNN_cols = [3,5,6,7,8,9,11,12]
         KNN_classifier = KNeighborsClassifier(n_neighbors = 9)
         KNN_classifier.fit(x_train_scaled.iloc[:,KNN_cols],y_train)

Out[37]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=9, p=2,
                              weights='uniform')

In [38]: y_pred = KNN_classifier.predict(x_val_scaled.iloc[:,KNN_cols])
```

```
In [39]: accuracy = metrics.accuracy_score(y_val,y_pred)
         error = 1 - accuracy
         precision = metrics.precision_score(y_val, y_pred, average = None)
         recall = metrics.recall_score(y_val, y_pred, average = None)
         F1_score = metrics.f1_score(y_val, y_pred, average = None)
         print("Accuracy = ",accuracy, "\nError = ",error, "\nPrecision =",precision,"\nRecall = ", recall,"\nF1 Sc
         ore = ", F1_score)

Accuracy =  0.8561872909698997
Error =  0.14381270903010035
Precision = [0.85943775 0.84      ]
Recall =  [0.96396396 0.54545455]
F1 Score =  [0.90870488 0.66141732]
```

Task 5. Clustering Methods

The two best performing classification models we found for this dataset are K-Means and DBSCAN. We used Adjusted Random Index as the supervised evaluation metric and Silhouette Coefficient as the unsupervised evaluation metric.

k-Means

K Means turned out to be the best performing clustering model. Initially, we included all the attributes of the training dataset. The Adjusted Random Index was 0.1975 and Silhouette Coefficient was 0.2664 for this model. The parameters used are : n_clusters = 2, init = 'random',n_init = 10, random_state = 0
n_clusters is 2 as we need to predict only 2 clusters.

Using all attributes

```
In [49]: clustering = KMeans(n_clusters = 2,init = 'random',n_init = 10, random_state = 0).fit(data_x_scaled)
         clusters = clustering.labels_
         print("Clusters are : ",np.unique(clusters))
         print("Number of clusters = ",len(clusters))

Clusters are :  [0 1]
Number of clusters =  1195

In [50]: adjusted_rand_index = metrics.adjusted_rand_score(data_y,clusters)
         silhouette_coefficient = metrics.silhouette_score(data_x_scaled.iloc[:,cols],clusters)
         print("Adjusted Random Index =",adjusted_rand_index, "\nSilhouette Coefficient = ",silhouette_coefficient)

Adjusted Random Index = 0.19751656022671712
Silhouette Coefficient =  0.26647094360113965
```

Then we tried to use different combinations of attributes instead of all the attributes to see which set of attributes gives well defined true clusters. We gave priority to Adjusted Random Index in an attempt to obtain true clusters. We concluded that ‘Percent Black, not Hispanic or Latino’,‘Percent Foreign Born’, ‘Percent Age 65 and Older’, ‘Median Household Income’, ‘Percent Unemployed’, ‘Less than HIgh School’, and ‘Less than Bachelor's Degree’ are the most important attributes to determine the cluster of a county with an Adjusted Random Index of 0.2712 and Silhouette Coefficient of 0.3329 .

Testing by selecting diffferent attributes

```
In [52]: cols = [2,4,7,8,10,11]
         clustering = KMeans(n_clusters = 2,init = 'random',n_init = 10, random_state = 0).fit(data_x_scaled.iloc
         [:,cols])
         clusters = clustering.labels_
         print("Clusters are : ",np.unique(clusters))
         print("Number of clusters = ",len(clusters))

Clusters are :  [0 1]
Number of clusters =  1195

In [53]: adjusted_rand_index = metrics.adjusted_rand_score(data_y,clusters)
         silhouette_coefficient = metrics.silhouette_score(data_x_scaled.iloc[:,cols],clusters)
         print("Adjusted Random Index =",adjusted_rand_index, "\nSilhouette Coefficient = ",silhouette_coefficient)

Adjusted Random Index = 0.2712192205336508
Silhouette Coefficient =  0.3329964236009124
```

DBSCAN

For DBSCAN, initially we included all the attributes of the training dataset. The Adjusted Random Index was 0.1577 and Silhouette Coefficient was 0.3204 for this model.

The parameters used are: eps = 2, min_samples = 5, metric='euclidean'

Using all attributes

```
In [55]: clustering = DBSCAN(eps = 2, min_samples = 5, metric='euclidean').fit(data_x_scaled)
         clusters = clustering.labels_
         print("Clusters are : ",np.unique(clusters))
         print("Number of clusters = ",len(clusters))

         Clusters are :  [-1  0  1]
         Number of clusters =  1195

In [56]: adjusted_rand_index = metrics.adjusted_rand_score(data_y,clusters)
         silhouette_coefficient = metrics.silhouette_score(data_x_scaled.iloc[:,cols],clusters)
         print("Adjusted Random Index =",adjusted_rand_index, "\nSilhouette Coefficient = ",silhouette_coefficient)

         Adjusted Random Index = 0.1577583255755803
         Silhouette Coefficient =  0.32046886952277354
```

Then we tried to use different combinations of attributes instead of all the attributes to see which set of attributes gives well defined true clusters. We gave priority to Adjusted Random Index in an attempt to obtain true clusters. We concluded that attributes 'Total Population','Percent White','Percent Black' and 'Percent Foreign Born' are the most important attributes to determine the cluster of a county with an Adjusted Random Index of 0.1577 and Silhouette Coefficient of 0.4887 .

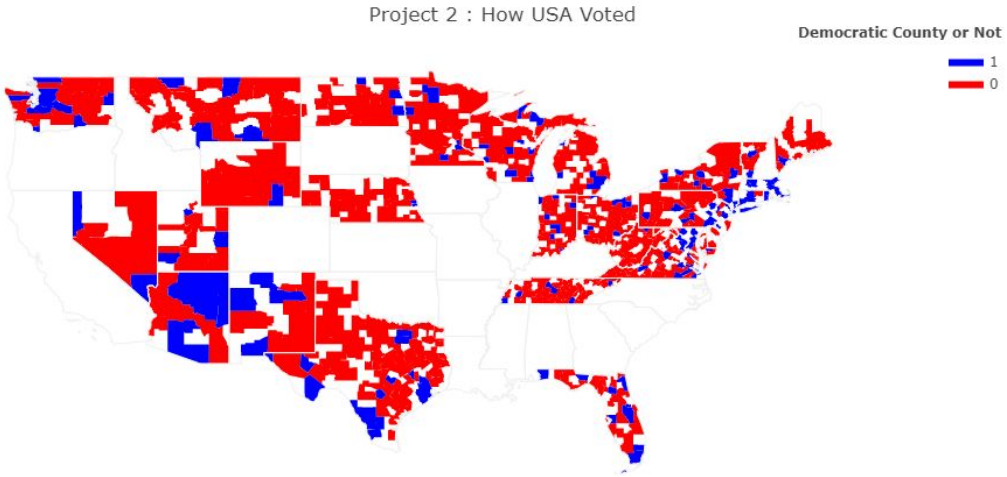
Testing by selecting diffferent attributes

```
In [112]: #cols = [2,4,7,8,10,11]
          cols = [0,1,2,4]
          clustering = DBSCAN(eps = 2, min_samples = 5, metric='euclidean').fit(data_x_scaled.iloc[:,cols])
          print("Clusters are : ",np.unique(clusters))
          print("Number of clusters = ",len(clusters))

          Clusters are :  [-1  0  1]
          Number of clusters =  1195

In [113]: adjusted_rand_index = metrics.adjusted_rand_score(data_y,clusters)
          silhouette_coefficient = metrics.silhouette_score(data_x_scaled.iloc[:,cols],clusters)
          print("Adjusted Random Index =",adjusted_rand_index, "\nSilhouette Coefficient = ",silhouette_coefficient)

          Adjusted Random Index = 0.1577583255755803
          Silhouette Coefficient =  0.48874944443295015
```

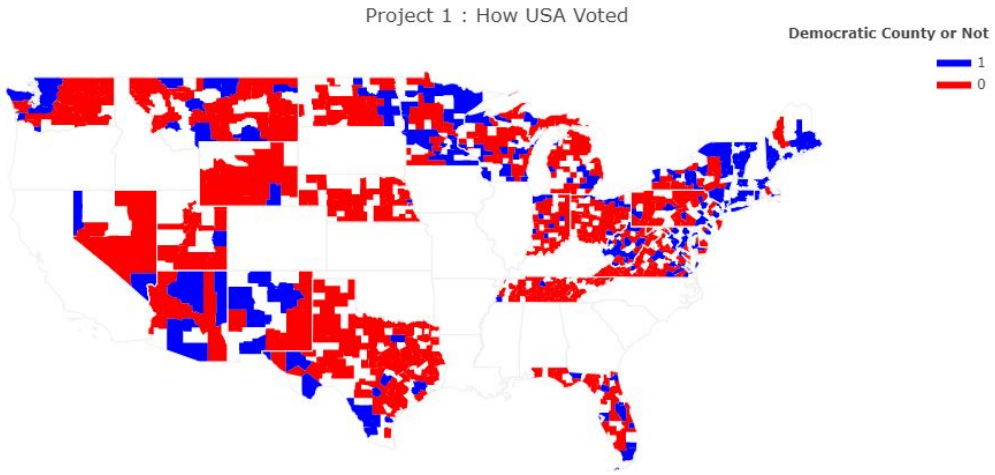
Task 6. Creating choropleth map

We found SVM to be the best performing classifier. So we used SVM classifier to classify counties based on all the demographic attributes of the merged dataset.

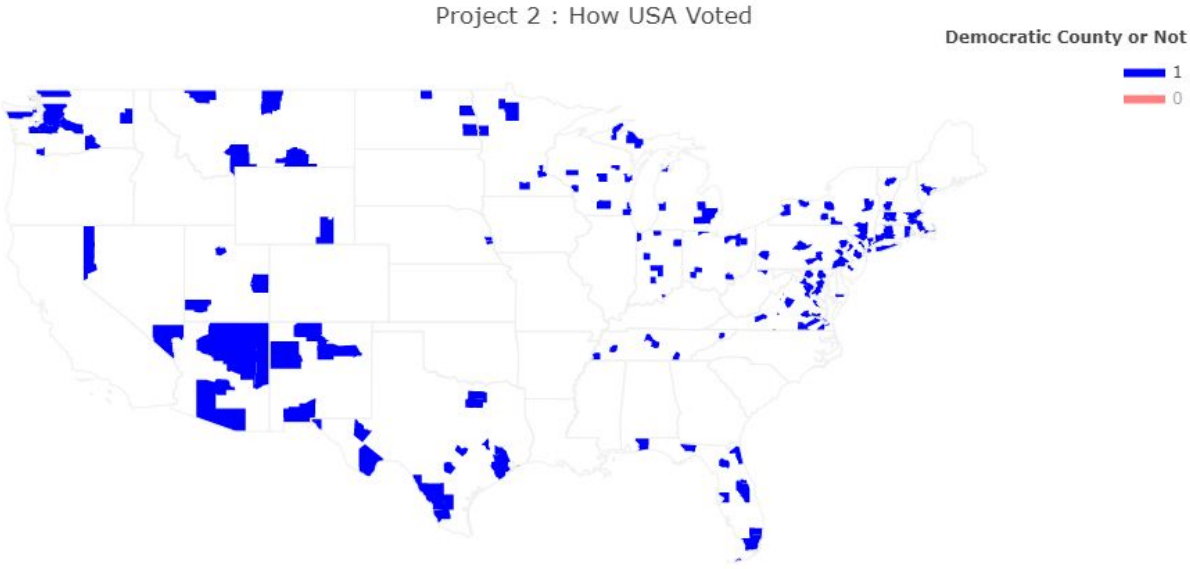
Map for Project 2:



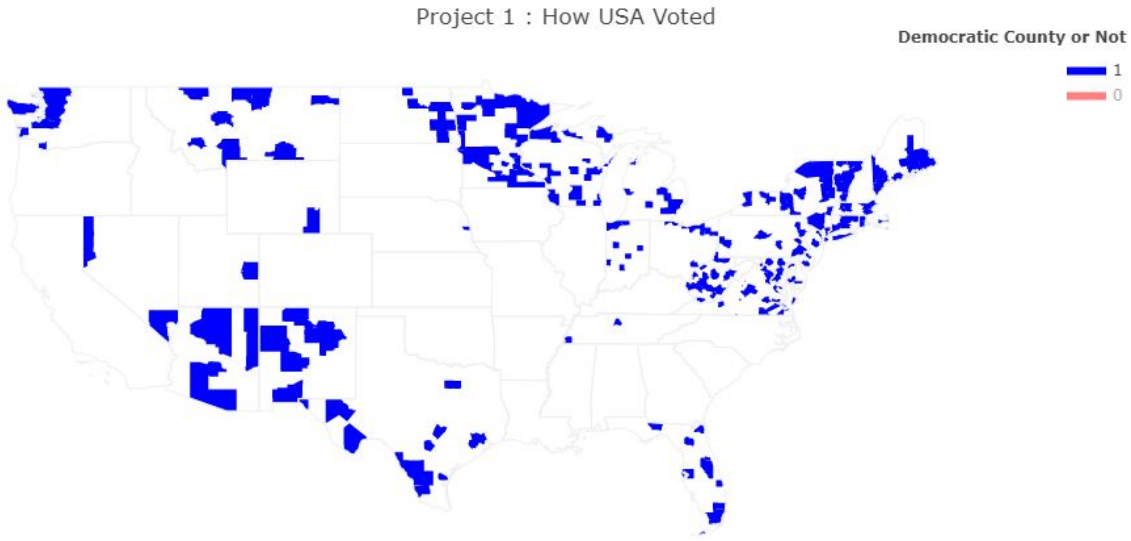
Map for Project 1:



Democratic counties found in Project 2:



Democratic counties found in Project 1:



7. Using best performing models

We used the best performing regression models after tuning the parameters for both the Democratic party and Republican party data.

SVM is our best performing model, So we used SVM to predict the class of the county based on the test dataset given to us.

We made a pandas dataframe to store the predicted votes and classes and to create the output CSV file.

```
In [69]: prediction = test_data[['State','County']]
```

Predict Democratic Party votes

```
In [70]: predicted = demo_fitted_model.predict(test_data_scaled.iloc[:, demo_cols])

In [71]: prediction['Democratic'] = predicted
```

Predict Democratic Party votes

```
In [72]: predicted = repub_fitted_model.predict(test_data_scaled.iloc[:, repub_cols])

In [73]: prediction['Republican'] = predicted
```

Predict winning Party (using SVM)

```
In [74]: class_pred = SVM_classifier.predict(test_data_scaled.iloc[:,SVM_cols])

In [75]: prediction['Party'] = class_pred

In [77]: prediction.head(10)
```

Out[77]:

	State	County	Democratic	Republican	Party
0	NV	eureka	10348.863597	4340.081194	1
1	TX	zavala	0.000000	0.000000	0
2	VA	king george	49592.687958	6261.083197	1
3	OH	hamilton	422219.367375	78704.729051	1
4	TX	austin	25431.101865	5579.948775	1
5	MI	barry	37938.948938	8093.733811	1
6	NM	valencia	46451.932733	7679.898972	1
7	TX	ellis	83341.506653	16900.949218	1
8	NJ	mercer	229639.199389	40011.561586	1
9	PA	cambria	83333.533575	15990.291551	1

```
In [78]: prediction.to_csv('project2_output.csv')
```