

```
In [1]: ┏━━━
      ┏━▶ from scipy.cluster.hierarchy import linkage, fcluster
      ┏━▶ from sklearn.model_selection import train_test_split
      ┏━▶ from sklearn.preprocessing import StandardScaler
      ┏━▶ from sklearn.tree import DecisionTreeClassifier
      ┏━▶ from sklearn.naive_bayes import GaussianNB
      ┏━▶ from sklearn.pipeline import Pipeline
      ┏━▶ import plotly.figure_factory as ff
      ┏━▶ from sklearn.cluster import KMeans, DBSCAN
      ┏━▶ from sklearn import linear_model
      ┏━▶ import matplotlib.pyplot as plt
      ┏━▶ from sklearn.svm import SVC
      ┏━▶ from sklearn import metrics
      ┏━▶ import seaborn as sns
      ┏━▶ import pandas as pd
      ┏━▶ import numpy as np
      ┏━▶ import os
```

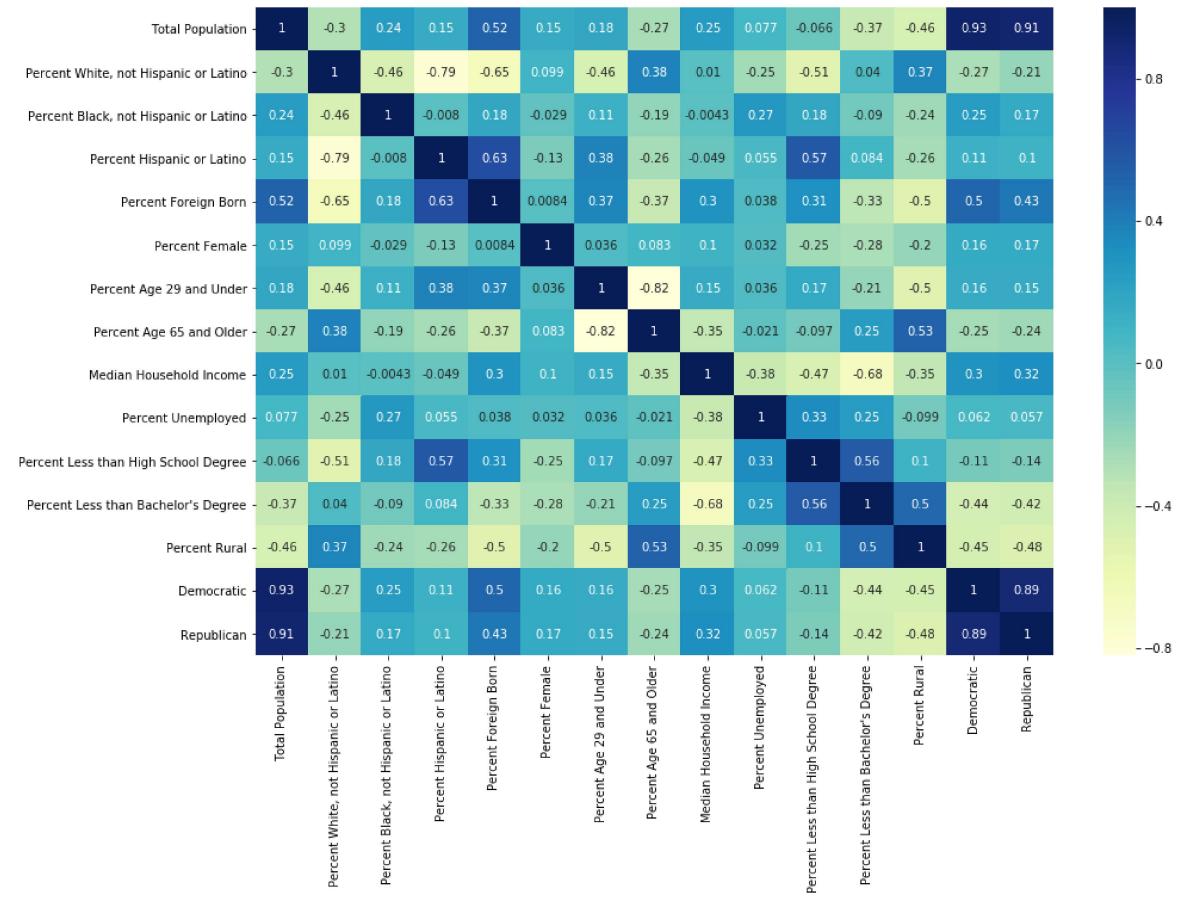
```
In [2]: ┏━━━
      ┏━▶ data = pd.read_csv(os.getcwd() + '\data\merged_train.csv')
      ┏━▶ # data.head()
```

Regression

Regression model to predict the votes cast for Republican and Democratic parties in each county

Choosing variables

```
In [3]: ┏ correlation = data.iloc[:, np.array([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(correlation,cmap="YlGnBu",annot=True, ax=ax)
plt.show()
```



```
In [4]: ► reg_cols = ['Total Population', 'Percent Foreign Born', 'Percent Less than Ba  
k = len(reg_cols)
```

Method for model evaluation

```
In [5]: ► def eval_regression_model(y_test, y_pred):  
    eval = ''  
    eval += 'Root Mean Square Error: {}'.format(metrics.mean_squared_error(y_...  
eval += 'Mean Absolute Error: {}'.format(metrics.mean_absolute_error(y_te...  
n = len(y_test)  
r2 = metrics.r2_score(y_test, y_pred)  
adjusted_r2 = 1 - ((1-r2)*(n-1)/(n-k-1))  
eval += 'R-Squared: {}'.format(r2) + '\n'  
eval += 'Adjusted R-Squared: {}'.format(adjusted_r2)  
return eval
```

Method for building and training a regression model

```
In [6]: ► def train_regression_model(x, y):  
    model = Pipeline([  
        ('scalar', StandardScaler()),  
        ('clf', linear_model.LassoCV(cv = 3))  
    #        ('clf', linear_model.LinearRegression())  
    ])  
    model.fit(x, y)  
    return model
```

Question 1: How was the dataset partitioned?

The dataset was partitioned using hold-out method

Question 2: Standardize the test and training datasets

Standardization was done using StandardScaler() in the pipeline

Predicting the votes cast for the democratic party

In [7]: ► # Splitting using hold-out method

```
x_train, x_test, y_train, y_test = train_test_split(data[reg_cols], data['Dem'])

dem_model = train_regression_model(x_train, y_train)
y_pred = dem_model.predict(x_test)
print(eval_regression_model(y_test, y_pred))
```

Root Mean Square Error: 25217.405065673698
 Mean Absolute Error: 7234.5728263734045
 R-Squared: 0.9089917345293794
 Adjusted R-Squared: 0.9077535268359016

Predicting the votes cast for the Republican party

In [8]: ► # Splitting using hold-out method

```
x_train, x_test, y_train, y_test = train_test_split(data[reg_cols], data['Rep'])

rep_model = train_regression_model(x_train, y_train)
y_pred = rep_model.predict(x_test)
print(eval_regression_model(y_test, y_pred))
```

Root Mean Square Error: 17258.664370356815
 Mean Absolute Error: 7822.2449374999105
 R-Squared: 0.8826640432665407
 Adjusted R-Squared: 0.8810676356919358

Question 3: What is the best performing linear regression model? What is the performance of the model? How did you select the parameters of the model? How did you select the variables of the model?

The best performing model in the LinearRegression. The parameters of this model were the default parameters. The performance of the model is - Root Mean Square Error: 17429.572586937353, Mean Absolute Error: 8085.641246178764, R-Squared: 0.880328639790789, Adjusted R-Squared: 0.8782864664083793. The variables were chosen such that each of them had a strong correlation with the target variables while also had the least multi-collinearity.

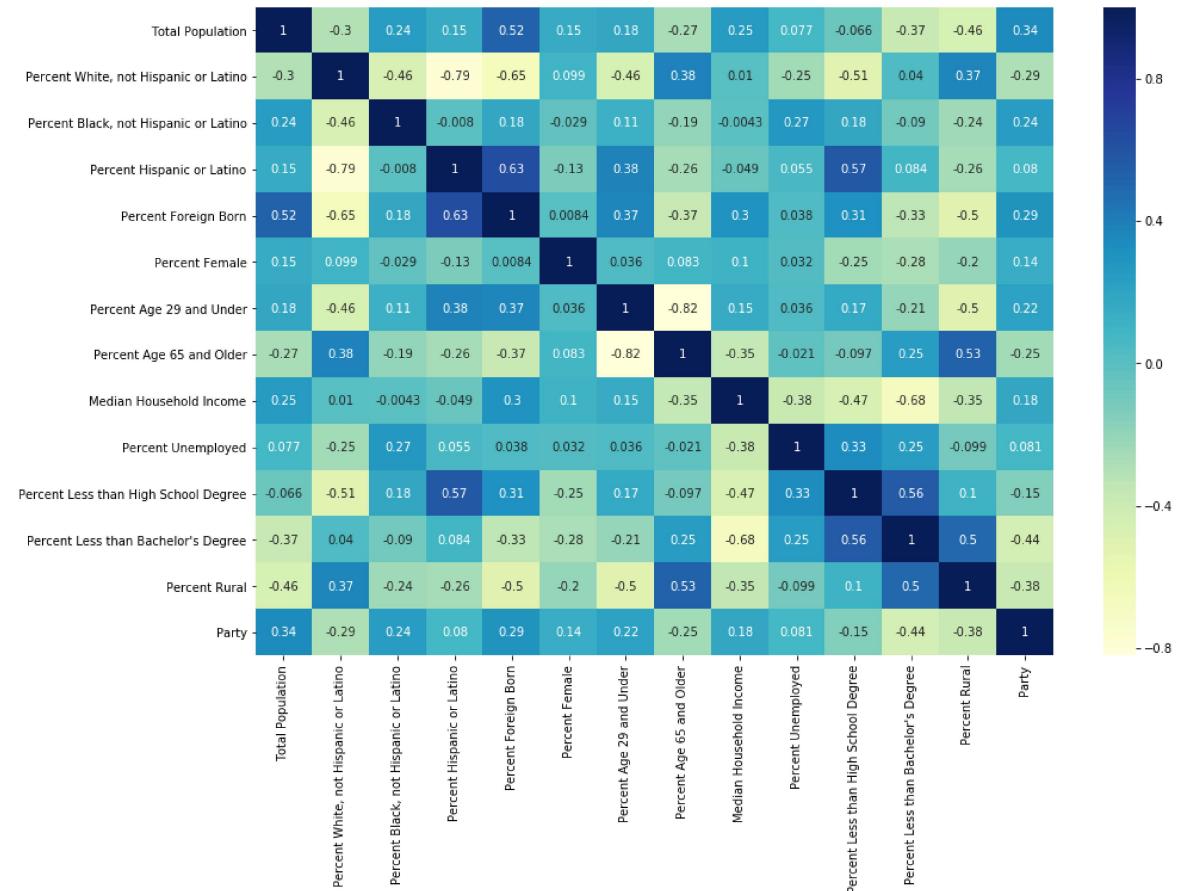
Classification

Classification models to classify each county as either republican or democratic

Choosing variables

Plot correlation heatmap and choose the variables that have a strong negative or positive correlation with the target variable

```
In [9]: ┏ correlation = data.iloc[:, np.array([3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14])
fig, ax = plt.subplots(figsize=(15,10))
sns.heatmap(correlation,cmap="YlGnBu",annot=True, ax=ax)
plt.show()
```



```
In [10]: ┏ class_cols = ['Total Population', 'Percent White, not Hispanic or Latino', 'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino', 'Percent Foreign Born', 'Percent Female', 'Percent Age 29 and Under', 'Percent Age 65 and Older', 'Median Household Income', 'Percent Unemployed', 'Percent Less than High School Degree', 'Percent Less than Bachelor's Degree', 'Percent Rural', 'Party']
# Splitting the dataset using hold-out method
x_train, x_test, y_train, y_test = train_test_split(data[class_cols], data['Party'], test_size=0.3, random_state=42)
```

Method for evaluating a classifier

```
In [11]: ┏ def eval_classifier(y_test, y_pred):
    return_str = ''
    accuracy = metrics.accuracy_score(y_test, y_pred)
    return_str += 'Accuracy: {}'.format(accuracy) + '\n'
    return_str += 'Error: {}'.format(1 - accuracy) + '\n'
    return_str += 'Precision: {}'.format(metrics.precision_score(y_test, y_pred)) + '\n'
    return_str += 'Recall: {}'.format(metrics.recall_score(y_test, y_pred, average='weighted')) + '\n'
    return_str += 'F1 Score: {}'.format(metrics.f1_score(y_test, y_pred, average='weighted')) + '\n'
    return return_str
```

Method for building and training a classifier

```
In [12]: def train_classifier(clf, x_train, y_train):
    classifier = Pipeline([('scalar', StandardScaler()),
                           ('clf', clf)])
    classifier.fit(x_train, y_train)
    return classifier
```

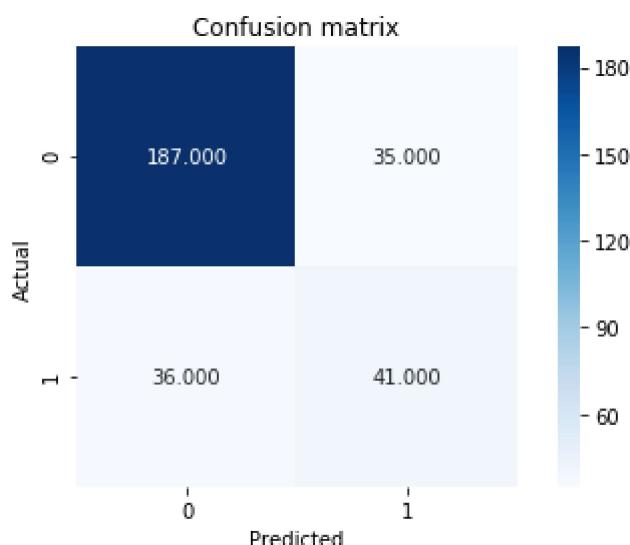
Using Decision Tree for classification

```
In [13]: # We are using 'entropy' as the splitting criterion and increasing the class
# because the data is skewed

dt_classifier = train_classifier(DecisionTreeClassifier(criterion = "entropy"))

y_pred = dt_classifier.predict(x_test)
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(eval_classifier(y_test, y_pred))
```

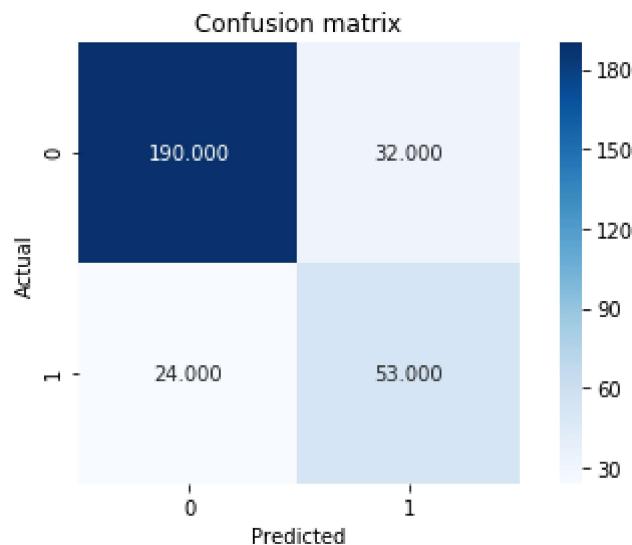
Accuracy: 0.7625418060200669
Error: 0.2374581939799306
Precision: [0.83856502 0.53947368]
Recall: [0.84234234 0.53246753]
F1 Score: [0.84044944 0.53594771]



Using SVM for classification

```
In [14]: # We are using 'rbf' kernel
svm_classifier = train_classifier(SVC(kernel='rbf', class_weight={1: 2}), x_t
y_pred = svm_classifier.predict(x_test)
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(eval_classifier(y_test, y_pred))
```

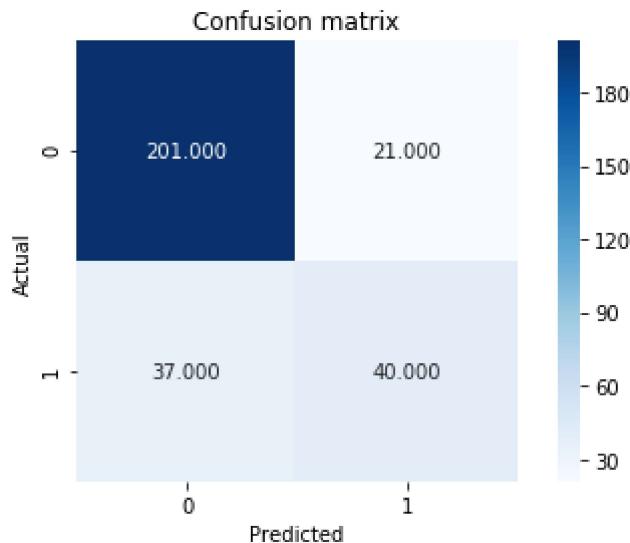
Accuracy: 0.8127090301003345
Error: 0.18729096989966554
Precision: [0.88785047 0.62352941]
Recall: [0.85585586 0.68831169]
F1 Score: [0.87155963 0.65432099]



Using Naive Bayes Classifier

```
In [15]: nb_classifier = train_classifier(GaussianNB(), x_train, y_train)
y_pred = nb_classifier.predict(x_test)
conf_matrix = metrics.confusion_matrix(y_test, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(eval_classifier(y_test, y_pred))
```

Accuracy: 0.8060200668896321
Error: 0.19397993311036787
Precision: [0.84453782 0.6557377]
Recall: [0.90540541 0.51948052]
F1 Score: [0.87391304 0.57971014]



Question 4: What is the best performing classification model? What is the performance of the model? How did you select the parameters of the model? How did you select the variables of the model?

The best performing classification model is the SVM classifier. The performance of this model is - Accuracy: 0.8394648829431438, Error: 0.1605351170568562, Precision: [0.89908257 0.67901235], Recall: [0.88288288 0.71428571], F1 Score: [0.89090909 0.69620253]. We chose

the rbf kernel because the decision boundaries seem to be non-linear. The class 1 was given more weight because, there are approximately twice the amount of observations of class 0 with respect to the ones of class 1. Other parameters are left to be default. The variables selected for classification model have a relatively stronger correlation with the target variable with respect to others.

Clustering

Clustering models to cluster the counties based on the party

Choosing variables

```
In [16]: ┏ clus_cols = ['Percent White, not Hispanic or Latino', 'Median Household Income',  
x = data[clus_cols]  
y = data['Party']
```

```
In [17]: ┏ scaler = StandardScaler()  
scaler.fit(x)  
x = scaler.transform(x)
```

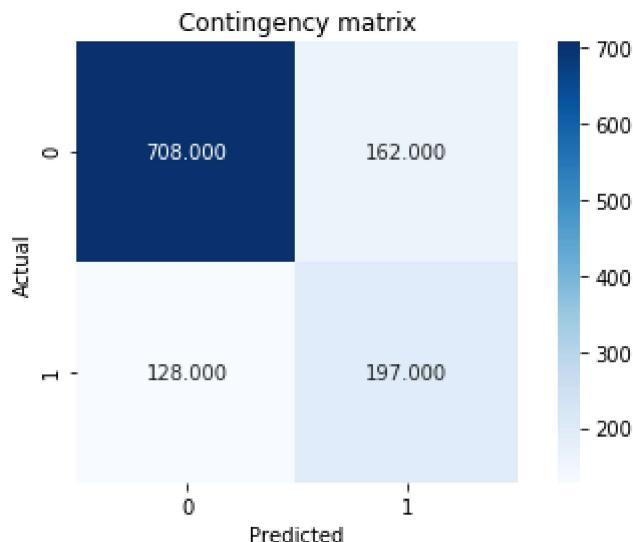
Method for evaluating each clustering technique

```
In [18]: ┏ def evaluate_clusters(X, y_test, y_pred):  
    ret_str = ''  
    ret_str += 'Adjusted Rand Index: {}'.format( metrics.adjusted_rand_score(y_test, y_pred))  
    ret_str += 'Silhouette Coefficient: {}'.format( metrics.silhouette_score(X, y_pred))  
    return ret_str
```

K-Means Clustering

```
In [19]: k_means_clusterer = KMeans(n_clusters = 2, n_init = 5, init = 'random', max_i  
clusters = k_means_clusterer.labels_  
cont_matrix = metrics.cluster.contingency_matrix(y, clusters)  
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt  
plt.ylabel('Actual')  
plt.xlabel('Predicted')  
plt.title('Contingency matrix')  
plt.tight_layout()  
print(evaluate_clusters(x, y, clusters))
```

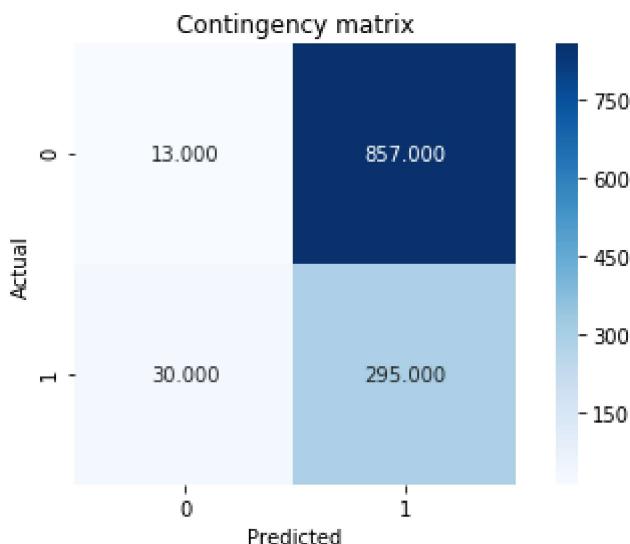
Adjusted Rand Index: 0.23922206669857454
Silhouette Coefficient: 0.28803789275218056



Hierarchical Clustering with ward method

```
In [20]: ► hierarchical_clusterer = linkage(x, method = 'complete', metric = 'euclidean')
clusters = fcluster(hierarchical_clusterer, 2, criterion = 'maxclust')
cont_matrix = metrics.cluster.contingency_matrix(y, clusters)
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
print(evaluate_clusters(x, y, clusters))
```

Adjusted Rand Index: 0.06765562549186784
 Silhouette Coefficient: 0.42953640479988603



Question 5: What is the best performing clustering model? What is the performance of the model? How did you select the parameters of model? How did you select the variables of the model?

The best performing clustering model is K-Means clustering. The performance of this model is Adjusted Rand Index: 0.24 and Silhouette Coefficient: 0.29. It seems to cluster more than 60% of the observations of each class correctly. We need 2 clusters, so n_clusters=2. When we run K-Means more than twice with random initialization of centroids, it increases the chances of getting clusters that are close to the true clusters. max_iter is 10 because we want to cluster as many observations properly in each run of the algorithm. The variables were selected based on how scattered they are with respect to the target variable.

Creating a map of Democratic and Republican Counties

```
In [21]: rep_fips = data['FIPS'].to_list()
values = np.where(data['Party']==1, 'Democratic', 'Republican')
fig = ff.create_choropleth(fips=rep_fips, values=values, legend_title='County'
fig.layout.template = None
fig.show()
```

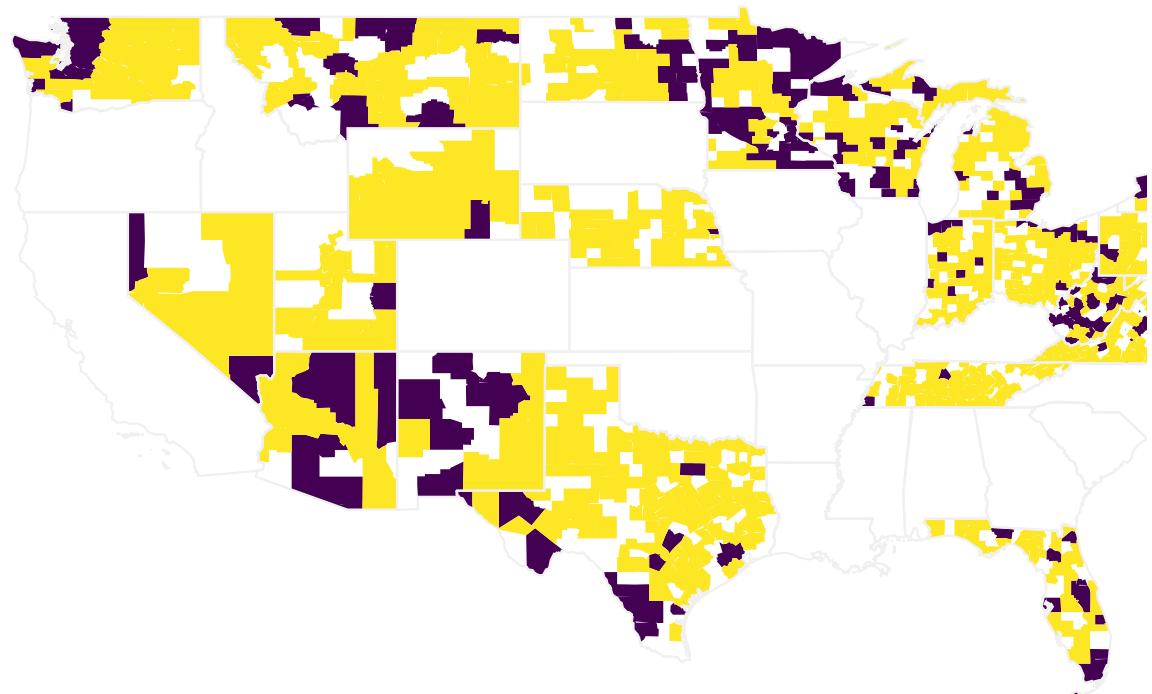
C:\Users\vigne\Applications\Anaconda3\lib\site-packages\pandas\core\frame.py:6692: FutureWarning:

Sorting because non-concatenation axis is not aligned. A future version of pandas will change to not sort by default.

To accept the future behavior, pass 'sort=False'.

To retain the current behavior and silence the warning, pass 'sort=True'.

Counties By Party



Testing the regression and classification models

The best performing regression and classification modes are tested below

Question 7

```
In [22]: ┏▶ test_data = pd.read_csv(os.getcwd() + '\data\demographics_test.csv')
reg_x = test_data[reg_cols]
class_x = test_data[class_cols]
```

```
In [23]: ┏▶ test_data['Democratic'] = dem_model.predict(reg_x)
```

```
In [24]: ┏▶ test_data['Republican'] = rep_model.predict(reg_x)
```

```
In [25]: ┏▶ test_data['Party'] = svm_classifier.predict(class_x)
```

```
In [26]: ┏▶ output = test_data[['State', 'County', 'Democratic', 'Republican', 'Party']]
output.head()
```

Out[26]:

	State	County	Democratic	Republican	Party
0	NV	eureka	-705.756837	2126.027513	0
1	TX	zavala	-1957.457775	303.672135	1
2	VA	king george	11927.735772	13033.062287	1
3	OH	hamilton	168214.800142	121880.192192	1
4	TX	austin	6707.873949	3374.323364	0

```
In [27]: ┏▶ output.to_csv('predictions.csv')
```