In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import KFold
import seaborn as sas
import matplotlib.pyplot as plt
import sklearn
from sklearn.model_selection import train_test_split
from sklearn import datasets, linear_model
from sklearn.preprocessing import StandardScaler
from scipy.cluster.hierarchy import linkage,fcluster
from sklearn import metrics
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
import seaborn as sns
from sklearn.linear_model import LinearRegression
import statsmodels.formula.api as smf
from sklearn.cluster import KMeans, DBSCAN
from sklearn import metrics
import plotly.figure_factory as ff
```

Task 1

In [2]:
```
# Import the CSV file
import pandas as pd
df = pd.read_csv("merged_train.csv")
df.head()
```

Out[2]:

	State	County	FIPS	Total Population	Percent White, not Hispanic or Latino	Percent Black, not Hispanic or Latino	Percent Hispanic or Latino	Percent Foreign Born	Percent Female	Pe A l
0	AZ	apache	4001	72346	18.571863	0.486551	5.947806	1.719515	50.598513	45.8
1	AZ	cochise	4003	128177	56.299492	3.714395	34.403208	11.458374	49.069646	37.9
2	AZ	coconino	4005	138064	54.619597	1.342855	13.711033	4.825298	50.581614	48.9
3	AZ	gila	4007	53179	63.222325	0.552850	18.548675	4.249798	50.296170	32.2
4	AZ	graham	4009	37529	51.461536	1.811932	32.097844	4.385942	46.313518	46.3

In [3]:
```
df.shape
```

Out[3]: (1195, 19)

In [4]:
```
#*****HOLDOUT METHOD*********
#Partitioning the data into Training(75%) and Validation set(25%)
X_train, X_val, Y_train, Y_val = train_test_split(df[df.columns[3:-3]], df[df.
columns[-3:]], test_size = 0.25, random_state=0)
```

In [5]:
```
print(X_train.shape)
print(Y_train.shape)
```

```
(896, 13)
(896, 3)
```

In [6]:
```
print(X_val.shape)
print(Y_val.shape)
```

```
(299, 13)
(299, 3)
```

In [7]:
```
Y_train.head()
```

Out[7]:

	Democratic	Republican	Party
589	2115	2916	0
702	3439	9365	0
182	1915	3541	0
655	6945	11210	0
1062	8159	18333	0

In [8]:
```
X_train.head()
```

Out[8]:

	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	H
589	14857	92.468197	3.163492	0.794238	0.511543	50.245675	34.832066	19.250185	
702	42406	94.168278	1.844079	1.903033	1.568174	50.853653	38.666698	17.426779	
182	10461	96.138037	0.248542	1.386101	1.720677	49.555492	21.202562	34.470892	
655	53460	96.127946	0.729517	1.060606	0.746352	50.787505	32.850730	19.943883	
1062	92530	55.510645	0.652761	40.121042	19.251054	49.605533	46.740517	12.576462	

Task 2

In [9]:
```
#-----Task 2-----
# Standardizing the demographic variables of train and Validation set
scaler = StandardScaler()
scaler.fit(X_train)
x_train_scaled = scaler.transform(X_train)
x_val_scaled = scaler.transform(X_val)
x_train_scaled
```

Out[9]:
```
array([[-0.3287239 ,  0.67151144, -0.26350303, ..., -0.17062413,
         1.01406194,  0.77555586],
       [-0.24936845,  0.75868846, -0.40186143, ...,  1.0578614 ,
         1.06814008,  0.60330794],
       [-0.34138666,  0.85969401, -0.56917509, ..., -0.30261512,
         0.76501978,  1.31092049],
       ...,
       [-0.04939121, -0.60217403,  0.44814651, ...,  0.32989637,
         0.21930393, -0.22759351],
       [-0.22119984,  0.87722062, -0.51847681, ..., -0.3554369 ,
         0.9134463 ,  0.30929678],
       [-0.25241604,  0.49667986, -0.2376449 , ...,  0.20906802,
         0.2665856 ,  0.41106862]])
```

In [10]:
```
Y_train.iloc[:1,0]
```

Out[10]:
```
589    2115
Name: Democratic, dtype: int64
```

In [11]:
```
x_train_r=np.concatenate([x_train_scaled,Y_train],axis=1)
```

In [12]:
```
x_train_r  = pd.DataFrame(x_train_r, index=X_train.index, columns=df.columns[3
:])
```

In [13]:
```
x_train_r.head()
```

Out[13]:

	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	Ho
589	-0.328724	0.671511	-0.263503	-0.615359	-0.755821	0.169311	-0.329466	0.235729	-(
702	-0.249368	0.758688	-0.401861	-0.541263	-0.581859	0.424608	0.328073	-0.140923	-(
182	-0.341387	0.859694	-0.569175	-0.575808	-0.556751	-0.120504	-2.666568	3.379791	-1
655	-0.217527	0.859177	-0.518738	-0.597559	-0.717162	0.396831	-0.669213	0.379022	-(
1062	-0.104985	-1.223603	-0.526787	2.012669	2.329427	-0.099491	1.712521	-1.142828	(

Task 3

In [14]:
```
#Predicting the number of democratic votes of validation set
#Building the model for Democratic votes using Total population as one of the
 attribute
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,0].reshape(-1,1),y=Y_train.iloc
[:,0])
print(fitted_model.coef_)
```

```
[74711.50206856]
```

In [15]:
```
#Predicting Democratic votes using Total population
predicted = fitted_model.predict(X=x_val_scaled[:,0].reshape(-1,1))
```

In [16]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,0],predicted)[1, 0]
R_squared = corr_coef**2
print('R-Square',R_squared)
```

```
R-Square 0.9436415220931651
```

In [17]:
```
# R-Square using Model Score
RSquare = model.score(X = x_val_scaled[:,0].reshape(-1,1),y = Y_val.iloc[:,0])
print('R-Square using model score',RSquare)
```

```
R-Square using model score 0.9168242212210275
```

In [18]:
```
# Building the model for Democratic votes using attributes initially identifie
d in project 1
# Total population, White Population,Age under 29,Population with less than Ba
chelor's degree
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,[0,2,6,11]],y=Y_train.iloc[:,0])
print(fitted_model.coef_)
```

```
[70974.39559794  2081.86202157 -3066.18177846 -9907.59785116]
```

In [19]:
```
## Predicting Democratic votes using using attributes initially identified in
 project 1
predicted = fitted_model.predict(X=x_val_scaled[:,[0,2,6,11]])
```

In [20]:
```
R_squared = corr_coef**2
print('R-Square',R_squared)
adjusted_r_squared = 1 - ((1-R_squared)*(len(Y_train)-1)/(len(Y_train)-x_val_s
caled[:,[0,2,6,11]].shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared)
# R-Square using Model Score
RSquare = model.score(X = x_val_scaled[:,[0,2,6,11]],y = Y_val.iloc[:,0])
print('R-Square using model score',RSquare)
adjusted_r_squared_model = 1 - ((1-RSquare)*(len(Y_train)-1)/(len(Y_train)-x_v
al_scaled[:,[0,2,6,11]].shape[1]-1))
print('Adjusted R-Square using model score',adjusted_r_squared_model)
```

```
R-Square 0.9436415220931651
Adjusted R-Square 0.9433885098466698
R-Square using model score 0.9018476013040684
Adjusted R-Square using model score 0.9014069620282168
```

Out of all the attributes,"Total Population" performed well as a single predictor for democratic votes with R-Square 0.9168 using model score

Performance choosing the initial set of attributes: Total population, White Population,Age under 25,Population with less than Bachelor's degree'

R-Square 0.944 and Adjusted R-Square 0.943

In [21]:
```
# Now building the model for Democratic votes using all the attributes
#to see how the R square changes with size of attributes
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled,y=Y_train.iloc[:,0])
print(fitted_model.coef_)
```

```
[ 69224.38708039  -3209.1591268   -1023.23488454  -6931.14708179
   3973.74580741    194.19056985  -5299.5676761   -1853.22320472
   1471.25963216   1467.0213699    4037.7699931  -10519.02638282
   -158.13004477]
```

In [22]:
```
#Predicting Democratic votes using all the attributes
predicted = fitted_model.predict(X=x_val_scaled)
```

In [23]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,0],predicted)[1, 0]
R_squared = corr_coef**2
print('R-Square',R_squared)
adjusted_r_squared = 1 - ((1-R_squared)*(len(Y_train)-1)/(len(Y_train)-x_val_s
caled.shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared)
```

```
R-Square 0.9338361960241593
Adjusted R-Square 0.932860992564198
```

In [24]:
```
# R-Square using Model Score
RSquare = model.score(X = x_val_scaled,y = Y_val.iloc[:,0])
print('R-Square using model score',RSquare)
adjusted_r_squared_model = 1 - ((1-RSquare)*(len(Y_train)-1)/(len(Y_train)-x_v
al_scaled.shape[1]-1))
print('Adjusted R-Square using model score',adjusted_r_squared_model)
```

```
R-Square using model score 0.867055068427187
Adjusted R-Square using model score 0.8650955626330299
```

Considering all the attributes for predicting the number of democratic votes, we observe the R-Square 0.9338 and Adjusted R-Square 0.9328

In [25]:
```
# Predicting Democratic votes using multiple attributes
# Attributes considered - Total Population, Black population,
#Population with less than a bachelor's degree, and the unemployed population.
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,[0, 11, 9, 2]],y=Y_train.iloc[:,
0])
print(fitted_model.coef_)
```

```
[70484.68208406 -9860.1194576   1648.91035669  1392.98545067]
```

In [26]:
```
predicted = fitted_model.predict(X=x_val_scaled[:,[0,  11, 9, 2]])
```

In [27]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,0],predicted)[1, 0]
R_squared = corr_coef**2
print('R-Square',R_squared)
adjusted_r_squared = 1 - ((1-R_squared)*(len(Y_train)-1)/(len(Y_train)-x_val_s
caled[:,[0,  11, 9, 2]].shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared)
```

```
R-Square 0.9506205107948219
Adjusted R-Square 0.9503988295862689
```

In [28]:
```
# R-Square using Model Score
RSquare = model.score(X = x_val_scaled[:,[0,  11, 9, 2]],y = Y_val.iloc[:,0])
print('R-Square',RSquare)
adjusted_r_squared_model = 1 - ((1-RSquare)*(len(Y_train)-1)/(len(Y_train)-x_v
al_scaled[:,[0, 11, 9, 2]].shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared_model)
```

```
R-Square 0.9043235928344125
Adjusted R-Square 0.9038940691209868
```

Using the initial set of attributes as a starting point,We added and dropped variables based on how R square changed.We found the optimal set of attribute that outputed a high R square value and Adjusted R square value of 0.9506 and 0.9503 respectivelyfor predicting number of democratic votes

Regression model for Republican votes

In [29]:
```
#Building the model using all the attributes
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled,y=Y_train.iloc[:,1])
print(fitted_model.coef_)
```

```
[45467.5097118   1769.95034533 -3141.4206375   1167.17323402
 -6463.65917143 -1121.73432851  -955.67013341  2580.74056065
  5910.97457236  2037.10575397  3530.42010898 -3156.11275644
 -5992.05181735]
```

In [30]:
```
#Predicting Republican votes using all the attributes
predicted = fitted_model.predict(X=x_val_scaled)
```

In [31]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,1],predicted)[1, 0]
R_squared = corr_coef**2
print('R-Square',R_squared)
adjusted_r_squared = 1 - ((1-R_squared)*(len(Y_train)-1)/(len(Y_train)-x_val_s
caled.shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared)
RSquare = model.score(X = x_val_scaled,y = Y_val.iloc[:,1])
print('R-Square',RSquare)
adjusted_r_squared_model = 1 - ((1-RSquare)*(len(Y_train)-1)/(len(Y_train)-x_v
al_scaled.shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared_model)
```

```
R-Square 0.7239014362949742
Adjusted R-Square 0.7198319563310679
R-Square 0.7004235899502084
Adjusted R-Square 0.6960080646320141
```

Considering all the attributes for predicting the number of Republican votes, we observe the R-Square 0.7239 and Adjusted R-Square 0.7198

In [32]:
```
#Building model for predicting Republican votes using Total population
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,0].reshape(-1,1),y=Y_train.iloc
[:,1])
print(fitted_model.coef_)
```

```
[45306.87897032]
```

In [33]:
```
#Predicting Republican votes using Total population
predicted = fitted_model.predict(X=x_val_scaled[:,0].reshape(-1,1))
```

In [34]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,1],predicted)[1, 0]
R_squared = corr_coef**2
print('R-Square',R_squared)
RSquare = model.score(X = x_val_scaled[:,0].reshape(-1,1),y = Y_val.iloc[:,1])
print('R-Square using model score',RSquare)
```

```
R-Square 0.6718468162068596
R-Square using model score 0.6567852066304897
```

In [35]:
```
# Building the model for Republican votes using attributes initially identifie
d in project 1
#Total population, White population,Age above 65,Population with less than a h
igh school degree
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,[0,1,7,10]],y=Y_train.iloc[:,1])
print(fitted_model.coef_)
```

```
[45659.94144049  2177.72938456  -703.90624058 -2722.92493514]
```

In [36]:
```
#Predicting Republican votes using initial set of attributes
predicted = fitted_model.predict(X=x_val_scaled[:,[0,1,7,10]])
```

In [37]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,1],predicted)[1, 0]
R_squared = corr_coef**2
print('R Square',R_squared)
adjusted_r_squared = 1 - ((1-R_squared)*(len(Y_train)-1)/(len(Y_train)-x_val_s
caled[:,[0,1,7,10]].shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared)
# R-Square using Model Score
RSquare = model.score(X = x_val_scaled[:,[0,1,7,10]],y = Y_val.iloc[:,1])
print('R-Square using model score',RSquare)
adjusted_r_squared_model = 1 - ((1-RSquare)*(len(Y_train)-1)/(len(Y_train)-x_v
al_scaled[:,[0,1,7,10]].shape[1]-1))
print('Adjusted R-Square using model score',adjusted_r_squared_model)
```

```
R Square 0.6802566781515079
Adjusted R-Square 0.678821242363187
R-Square using model score 0.6621606644934113
Adjusted R-Square using model score 0.6606439895865355
```

In [38]:
```
# Building the model for Republican votes using best set of attributes
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,[0,1,3,4,7,8,9,10,11,12]],y=Y_tr
ain.iloc[:,1])
print(fitted_model.coef_)
```

```
[45220.29033736  5065.79031785  3840.00607902 -5872.48026158
  2839.30463983  6066.05645897  2063.52147931  2586.73805496
 -2432.72004918 -5176.17669614]
```

In [39]:
```
# Predicting Republican votes using best set of attributes
predicted = fitted_model.predict(X=x_val_scaled[:,[0,1,3,4,7,8,9,10,11,12]])
```

In [40]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,1],predicted)[1, 0]
R_squared = corr_coef**2
print('R-square',R_squared)
adjusted_r_squared = 1 - ((1-R_squared)*(len(Y_train)-1)/(len(Y_train)-x_val_s
caled[:,[0,1,3,4,7,8,9,10,11,12]].shape[1]-1))
print('Adjusted R-Square',adjusted_r_squared)
```

```
R-square 0.7323279131009545
Adjusted R-Square 0.729303369746163
```

In [41]:
```
RSquare = model.score(X = x_val_scaled[:,[0,1,3,4,7,8,9,10,11,12]],y = Y_val.i
loc[:,1])
print('R-Square using model score',RSquare)
adjusted_r_squared_model = 1 - ((1-RSquare)*(len(Y_train)-1)/(len(Y_train)-x_v
al_scaled[:,[0,1,3,4,7,8,9,10,11,12]].shape[1]-1))
print('Adjusted R-Square using model score',adjusted_r_squared_model)
```

```
R-Square using model score 0.7115853695297715
Adjusted R-Square using model score 0.7083264471515768
```

Using the initial set of attributes determined in project 1 as a starting point,We added and dropped variables based on how R square changed.We found the optimal set of attribute that outputed a high R square value and Adjusted R square value of 0.7323 and 0.729 respectively for predicting number of republican votes

In [42]:
```
#Second type of linear regrsseion model to narrow down search
#Building using LASSO Regularization to check if any attributes are dropped
model = linear_model.Lasso(alpha = 1)
fitted_model = model.fit(X = x_train_scaled,y=Y_train.iloc[:,1])
```

In [43]:
```
print(fitted_model.coef_)
```

```
[45464.11625996  1763.84615535 -3141.51363944  1160.39910811
 -6454.91877737 -1119.19972956  -956.20034133  2577.09105238
  5906.62715265  2034.44712921  3523.56962737 -3151.08771664
 -5989.09353181]
```

In [44]:
```
predicted = fitted_model.predict(X=x_val_scaled)
```

In [45]:
```
corr_coef = np.corrcoef(Y_val.iloc[:,1],predicted)[1, 0]
R_squared = corr_coef**2
print(R_squared)
```

```
0.72388866630169
```

Using LASSO regression, we found that no attributes were dropped

- What is the best performing linear regression model?

The Best performing linear model for predicting number of democratic votes is the one that has the following predictor variables: Total Population, Population that is Black, Population with less than a bachelor's degree, and the unemployed population.

- What is the performance of the model?

For **Democratic** R square value-0.9506 and Adjusted R square -0.9503

We were able to determine that total population would be a default predictor based on a linear regression single test. We then ended up with 8 combination and form those combinations we started to drop values based on democratic parties intuition. We also dropped predictors based on democratic influences and confirmed by performing multiple linear regression tests.

- What is the performance of the model?

For **Republican** the R-squared: 0.7323 & Adjusted R-Square 0.7293

- How did you select the variables of the model?

Based on our analysis in project 1, we chose a initial set of attributes for both democratic and republican votes.Then, using Linear regression, we added and dropped attributes according to the adjusted R square value.We kept the attribute that increased the R-Square and droppedd the one's, that reduced the model's performance

Task 4

Decision Tree

In [46]:
```
#Using Decision Tree as a classifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(x_train_scaled,Y_train['Party'])
```

Out[46]:
```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=Non
e,
                       max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=0, splitter='best')
```

In [47]:
```
print(classifier.tree_.__getstate__()['nodes'])
len(classifier.tree_.__getstate__()['nodes'])
```

```
[(  1, 100, 11, -0.08037002, 0.85103407, 896, 896.)
 (  2,   7, 12, -1.57463121, 0.99675236, 328, 328.)
 (  3,   4,  1,  0.13988956, 0.28290479,  61,  61.)
 ( -1,  -1, -2, -2.        , 0.        ,  57,  57.)
 (  5,   6, 11, -2.27329141, 0.81127812,   4,   4.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
 (  8,  15,  0, -0.34515437, 0.98895258, 267, 267.)
 (  9,  10, 11, -0.18558561, 0.30337484,  37,  37.)
 ( -1,  -1, -2, -2.        , 0.        ,  29,  29.)
 ( 11,  12,  9, -0.91212842, 0.81127812,   8,   8.)
 ( -1,  -1, -2, -2.        , 0.        ,   5,   5.)
 ( 13,  14,  5,  0.40603571, 0.91829583,   3,   3.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( 16,  17,  1, -1.39712286, 1.        , 230, 230.)
 ( -1,  -1, -2, -2.        , 0.        ,  10,  10.)
 ( 18,  99, 10,  0.07734165, 0.9985091 , 220, 220.)
 ( 19,  42,  4, -0.37913467, 0.99998349, 209, 209.)
 ( 20,  25,  8, -0.14533475, 0.8890349 ,  62,  62.)
 ( 21,  24,  7,  0.36920083, 0.86312057,  14,  14.)
 ( 22,  23,  2,  0.18771875, 0.65002242,  12,  12.)
 ( -1,  -1, -2, -2.        , 0.        ,  10,  10.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( 26,  27,  1,  0.62267354, 0.69621226,  48,  48.)
 ( -1,  -1, -2, -2.        , 0.        ,  16,  16.)
 ( 28,  41, 11, -0.1682383 , 0.85714844,  32,  32.)
 ( 29,  40,  6,  0.67939885, 0.73550858,  29,  29.)
 ( 30,  39,  8,  0.51208383, 0.60518658,  27,  27.)
 ( 31,  34,  4, -0.49395105, 0.83664074,  15,  15.)
 ( 32,  33, 11, -0.27043697, 0.46899559,  10,  10.)
 ( -1,  -1, -2, -2.        , 0.        ,   9,   9.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( 35,  36,  6, -0.97654212, 0.97095059,   5,   5.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( 37,  38,  8,  0.11499928, 0.91829583,   3,   3.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,  12,  12.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
 ( 43,  92,  3,  0.05903428, 0.97903461, 147, 147.)
 ( 44,  83,  8,  2.13687468, 0.93925472, 118, 118.)
 ( 45,  82, 12, -0.01478512, 0.88247445, 103, 103.)
 ( 46,  53, 11, -1.47726208, 0.93255384,  89,  89.)
 ( 47,  48,  8,  1.62924671, 0.42622866,  23,  23.)
 ( -1,  -1, -2, -2.        , 0.        ,  17,  17.)
 ( 49,  52,  5,  0.71812838, 0.91829583,   6,   6.)
 ( 50,  51, 11, -2.57235062, 0.91829583,   3,   3.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
 ( 54,  57, 10, -1.00663757, 0.98937558,  66,  66.)
 ( 55,  56, 11, -1.36946547, 0.43949699,  11,  11.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( -1,  -1, -2, -2.        , 0.        ,  10,  10.)
```

```
( 58,  69,  0,  0.06232688, 0.99976152,  55,  55.)
( 59,  60,  0, -0.26622814, 0.90592822,  28,  28.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
( 61,  62,  9, -0.40477926, 0.79504028,  25,  25.)
( -1,  -1, -2, -2.        , 0.        ,  10,  10.)
( 63,  64,  0, -0.21745228, 0.97095059,  15,  15.)
( -1,  -1, -2, -2.        , 0.        ,   4,   4.)
( 65,  66,  7, -0.58334582, 0.99403021,  11,  11.)
( -1,  -1, -2, -2.        , 0.        ,   4,   4.)
( 67,  68, 10, -0.69751415, 0.86312057,   7,   7.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( -1,  -1, -2, -2.        , 0.        ,   5,   5.)
( 70,  81, 12, -1.04157078, 0.91829583,  27,  27.)
( 71,  80,  6,  0.92913273, 0.99277445,  20,  20.)
( 72,  75,  7, -0.46444936, 0.89603823,  16,  16.)
( 73,  74, 10, -0.92671674, 0.50325833,   9,   9.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( -1,  -1, -2, -2.        , 0.        ,   8,   8.)
( 76,  79,  6, -0.03350545, 0.98522814,   7,   7.)
( 77,  78,  9,  0.89954698, 0.81127812,   4,   4.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
( -1,  -1, -2, -2.        , 0.        ,   4,   4.)
( -1,  -1, -2, -2.        , 0.        ,   7,   7.)
( -1,  -1, -2, -2.        , 0.        ,  14,  14.)
( 84,  85,  3, -0.4157771 , 0.83664074,  15,  15.)
( -1,  -1, -2, -2.        , 0.        ,   7,   7.)
( 86,  87,  4,  0.10013912, 1.        ,   8,   8.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( 88,  89,  4,  0.30861057, 0.91829583,   6,   6.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
( 90,  91,  4,  0.71367368, 0.91829583,   3,   3.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( 93,  96,  9,  0.22570178, 0.92936363,  29,  29.)
( 94,  95,  8,  3.66894639, 0.54356444,  16,  16.)
( -1,  -1, -2, -2.        , 0.        ,  14,  14.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( 97,  98,  7,  0.72845934, 0.9612366 ,  13,  13.)
( -1,  -1, -2, -2.        , 0.        ,   8,   8.)
( -1,  -1, -2, -2.        , 0.        ,   5,   5.)
( -1,  -1, -2, -2.        , 0.        ,  11,  11.)
(101, 110,  1, -1.93926793, 0.55336838, 568, 568.)
(102, 103,  3,  2.75355005, 0.99613448,  41,  41.)
( -1,  -1, -2, -2.        , 0.        ,  13,  13.)
(104, 105,  1, -3.12038493, 0.90592822,  28,  28.)
( -1,  -1, -2, -2.        , 0.        ,   7,   7.)
(106, 109,  1, -2.37764275, 0.45371634,  21,  21.)
(107, 108,  3,  3.70863354, 0.91829583,   6,   6.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( -1,  -1, -2, -2.        , 0.        ,   4,   4.)
( -1,  -1, -2, -2.        , 0.        ,  15,  15.)
(111, 196,  3,  0.07042832, 0.45868582, 527, 527.)
(112, 193,  5,  1.17786229, 0.53017385, 424, 424.)
(113, 158, 10, -0.27290677, 0.50723272, 418, 418.)
(114, 115,  2, -0.58004016, 0.66366113, 168, 168.)
```

```
( -1,  -1, -2, -2.        , 0.        ,  24,  24.)
(116, 141,  6, -0.33900376, 0.72469719, 144, 144.)
(117, 140,  9,  0.75768894, 0.86853396,  69,  69.)
(118, 139, 10, -0.29305258, 0.93484902,  57,  57.)
(119, 120, 12, -0.6469022 , 0.89865338,  54,  54.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
(121, 138,  7,  1.93100727, 0.84786175,  51,  51.)
(122, 131,  2, -0.52002695, 0.80309098,  49,  49.)
(123, 124,  1,  0.77033558, 0.56650951,  30,  30.)
( -1,  -1, -2, -2.        , 0.        ,  15,  15.)
(125, 126,  7,  0.24093483, 0.83664074,  15,  15.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
(127, 128, 10, -0.54269454, 0.61938219,  13,  13.)
( -1,  -1, -2, -2.        , 0.        ,  10,  10.)
(129, 130,  3, -0.59259918, 0.91829583,   3,   3.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
(132, 135, 10, -0.52860704, 0.98194079,  19,  19.)
(133, 134,  0, -0.35401343, 0.54356444,   8,   8.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( -1,  -1, -2, -2.        , 0.        ,   7,   7.)
(136, 137,  5, -1.3264969 , 0.43949699,  11,  11.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( -1,  -1, -2, -2.        , 0.        ,  10,  10.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
( -1,  -1, -2, -2.        , 0.        ,  12,  12.)
(142, 155,  9,  1.04082423, 0.52936087,  75,  75.)
(143, 154,  9, -0.2200299 , 0.41786426,  71,  71.)
(144, 149, 12, -0.17743065, 0.6098403 ,  40,  40.)
(145, 146,  3, -0.42516482, 1.        ,   8,   8.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
(147, 148,  9, -1.51887619, 0.72192809,   5,   5.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( -1,  -1, -2, -2.        , 0.        ,   4,   4.)
(150, 151, 12,  1.07181042, 0.33729007,  32,  32.)
( -1,  -1, -2, -2.        , 0.        ,  24,  24.)
(152, 153,  6, -0.20031215, 0.81127812,   8,   8.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( -1,  -1, -2, -2.        , 0.        ,   6,   6.)
( -1,  -1, -2, -2.        , 0.        ,  31,  31.)
(156, 157,  4, -0.40317059, 0.81127812,   4,   4.)
( -1,  -1, -2, -2.        , 0.        ,   3,   3.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
(159, 170,  4, -0.71062896, 0.37334332, 250, 250.)
(160, 169,  8, -0.68556282, 0.74248757,  38,  38.)
(161, 164, 12,  0.77434984, 0.91829583,  24,  24.)
(162, 163,  8, -0.79894298, 0.81127812,   8,   8.)
( -1,  -1, -2, -2.        , 0.        ,   6,   6.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
(165, 168,  7, -0.18964487, 0.54356444,  16,  16.)
(166, 167,  2, -0.32445248, 0.91829583,   3,   3.)
( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
( -1,  -1, -2, -2.        , 0.        ,  13,  13.)
( -1,  -1, -2, -2.        , 0.        ,  14,  14.)
(171, 180, 11,  0.20588898, 0.27425064, 212, 212.)
```

```
 (172, 173,  9, -0.30048504, 0.73550858,  29,  29.)
 ( -1,  -1, -2, -2.        , 0.        ,  12,  12.)
 (174, 177,  0, -0.24804918, 0.93666738,  17,  17.)
 (175, 176, 12,  1.22702581, 0.86312057,   7,   7.)
 ( -1,  -1, -2, -2.        , 0.        ,   5,   5.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 (178, 179, 11, -0.00373855, 0.46899559,  10,  10.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( -1,  -1, -2, -2.        , 0.        ,   9,   9.)
 (181, 192,  5, -0.15476202, 0.15174889, 183, 183.)
 (182, 183,  8, -1.90594471, 0.32275696,  68,  68.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 (184, 191, 11,  0.63255814, 0.26377744,  67,  67.)
 (185, 190, 11,  0.58593363, 0.65002242,  18,  18.)
 (186, 187,  0, -0.12371872, 0.33729007,  16,  16.)
 ( -1,  -1, -2, -2.        , 0.        ,  14,  14.)
 (188, 189,  4, -0.45904274, 1.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( -1,  -1, -2, -2.        , 0.        ,   1,   1.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,  49,  49.)
 ( -1,  -1, -2, -2.        , 0.        , 115, 115.)
 (194, 195,  0, -0.34596957, 0.91829583,   6,   6.)
 ( -1,  -1, -2, -2.        , 0.        ,   2,   2.)
 ( -1,  -1, -2, -2.        , 0.        ,   4,   4.)
 ( -1,  -1, -2, -2.        , 0.        , 103, 103.)]
```

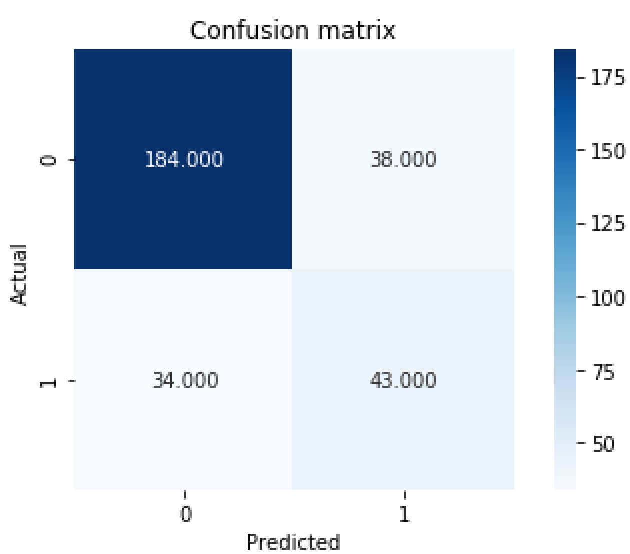
Out[47]: 197

In [48]:

```
y_pred = classifier.predict(x_val_scaled)
```

In [49]:

```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [50]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall])
```

```
[0.7591973244147158, 0.24080267558528423, 0.5308641975308642, 0.5584415584415
584]
```

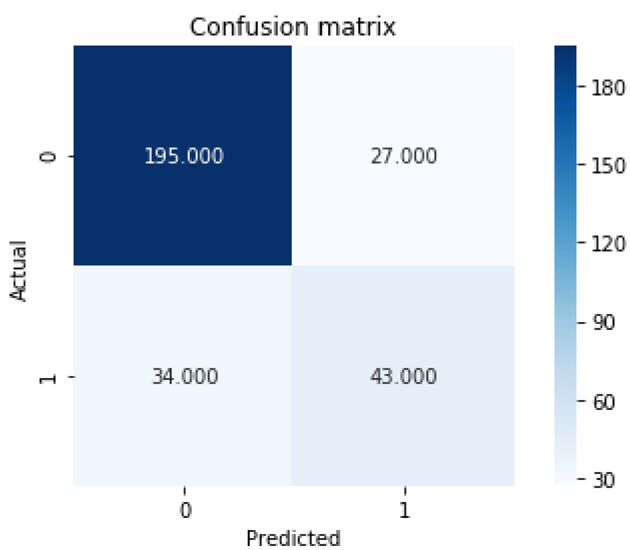
Naive Bayes Classifier

In [51]:
```
#-----------------Naive Bayes Classifier--------------------
#Using all the attributes
classifier = GaussianNB()
classifier.fit(x_train_scaled,Y_train['Party'])
```

Out[51]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [52]:
```
y_pred = classifier.predict(x_val_scaled)
```

In [53]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [54]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

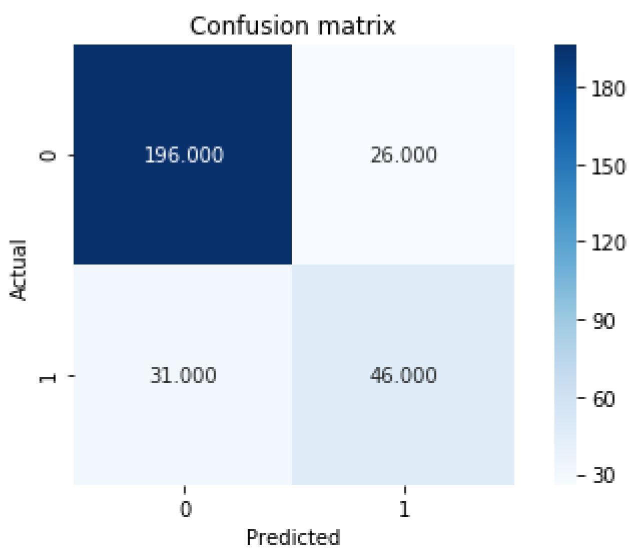
```
[0.7959866220735786, 0.20401337792642138, 0.6142857142857143, 0.5584415584415
584, 0.7927124130729645]
```

In [55]:
```
#-----------------Naive Bayes Classifier using different attributes ----------
----------
classifier = GaussianNB()
classifier.fit(x_train_scaled[:,[0,1,2,3,4,6,7,9,11,12]],Y_train['Party'])
```

Out[55]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [56]:
```
y_pred = classifier.predict(x_val_scaled[:,[0,1,2,3,4,6,7,9,11,12]])
```

In [57]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [58]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred,average=None)
recall = metrics.recall_score(Y_val['Party'], y_pred,average=None)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8093645484949833, 0.1906354515050167, array([0.86343612, 0.63888889]), arr
ay([0.88288288, 0.5974026 ]), 0.8072274117013812]
```

Considering our initial analysis in project 1

Attributes **'White population', ' Less than Bachelor's degree','Age 65 and over'** favored **Republican**

Attributes **'Black population','Foreign Born','Age under 29**' favored **Democrats**

Attributes **'Median Household income', 'Unemployment','Hispanic or Latino','Percent Rural'** do not favor any party

So choosing these attributes to classify the labels with higher accuracy and f1 score

Naive Bayes with multiple attributes

Accuracy = 80.936 F1 Score = 0.617

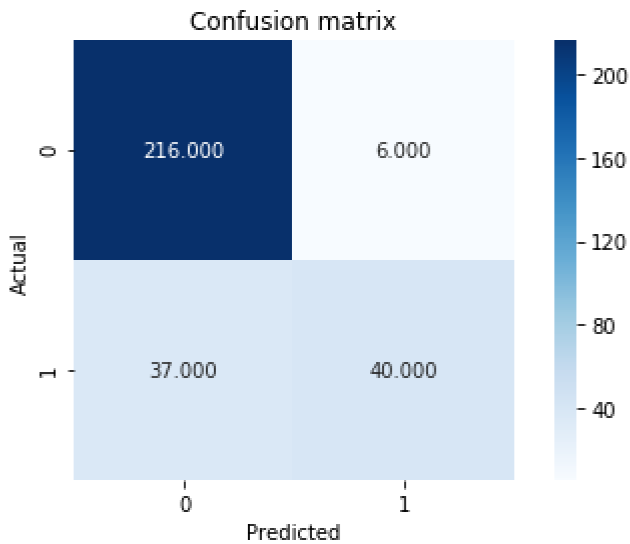
SVM Classifier

In [59]:
```
#-----------------------SVM----------------------------------------
#Using all the attributes
classifier = SVC(kernel = 'rbf')
classifier.fit(x_train_scaled,Y_train['Party'])
```

Out[59]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [60]:
```
y_pred = classifier.predict(x_val_scaled)
```

In [61]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [62]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

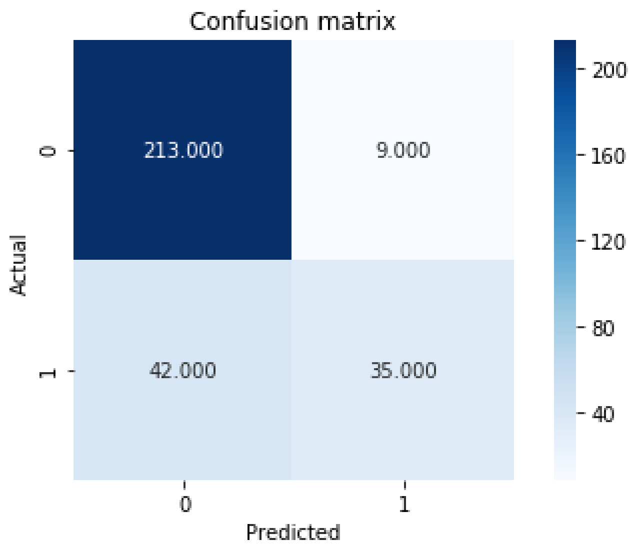
```
[0.8561872909698997, 0.14381270903010035, 0.8695652173913043, 0.5194805194805
194, 0.8427573869824246]
```

In [63]:
```
#--------------------SVM using selected multiple attributes with Linear Kerne
l------------------------------------
classifier = SVC(kernel = 'linear')
classifier.fit(x_train_scaled[:,[0,2,3,4,5,6,7,8,9,10,11,12]],Y_train['Party'
])
```

Out[63]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='linear', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [64]:
```
y_pred = classifier.predict(x_val_scaled[:,[0,2,3,4,5,6,7,8,9,10,11,12]])
```

In [65]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [66]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='micro')
print([accuracy, error, precision, recall, F1_score])
```

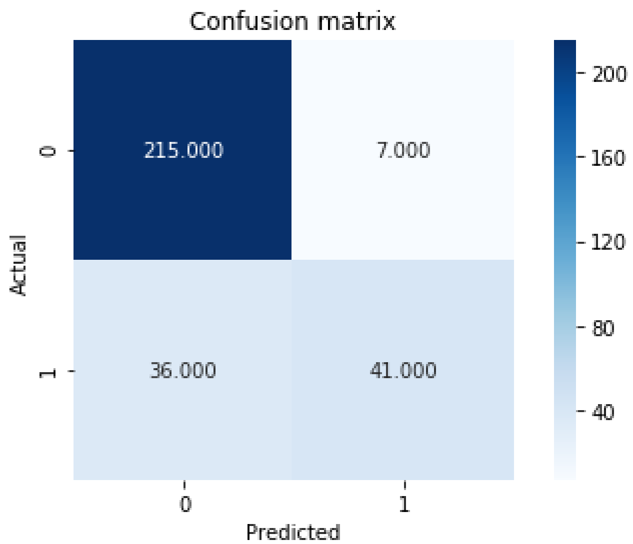
```
[0.8294314381270903, 0.1705685618729097, 0.7954545454545454, 0.45454545454545
453, 0.8294314381270903]
```

In [67]:
```
#----------------------SVM using multiple attributes using RBF---------------
-----------------------
classifier = SVC(kernel = 'rbf')
classifier.fit(x_train_scaled[:,[0,2,3,4,5,6,7,8,9,10,11,12]],Y_train['Party'
])
```

Out[67]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [68]:
```
y_pred = classifier.predict(x_val_scaled[:,[0,2,3,4,5,6,7,8,9,10,11,12]])
```

In [69]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [70]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8561872909698997, 0.14381270903010035, 0.8541666666666666, 0.5324675324675
324, 0.8439136515658254]
```

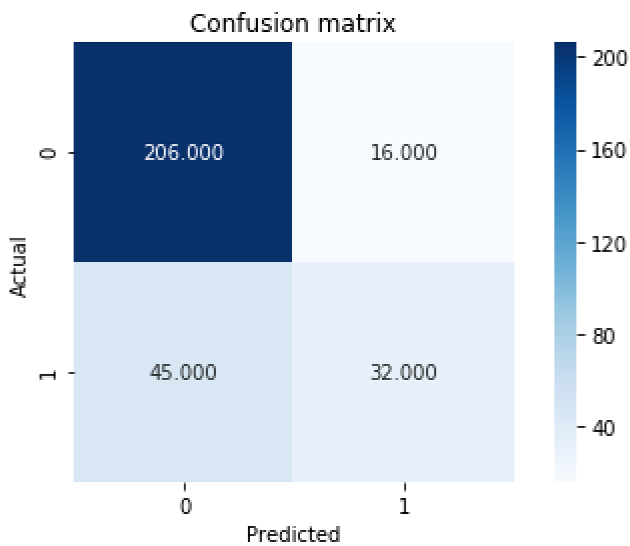
K Nearest Neighbours

In [71]:
```
#K-Nearest Neighbours with n=5
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train_scaled,Y_train['Party'])
```

Out[71]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [72]:
```
y_pred = classifier.predict(x_val_scaled)
```

In [73]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [74]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

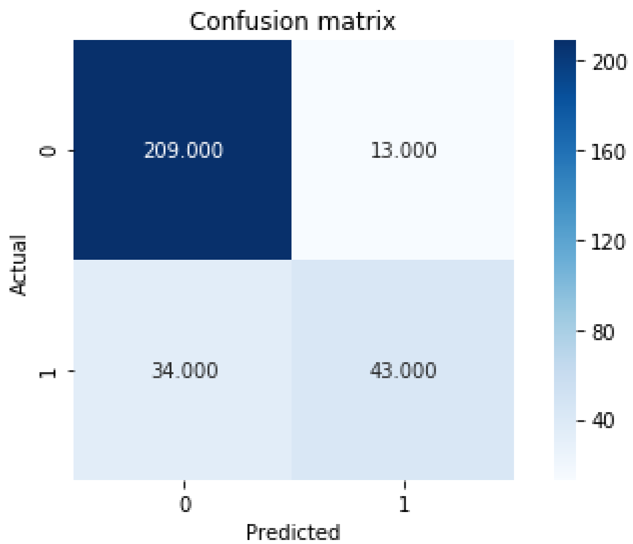
```
[0.7959866220735786, 0.20401337792642138, 0.6666666666666666, 0.4155844155844
156, 0.7785751801282641]
```

In [75]:
```
#K-Nearest Neighbours using multiple variables and 5
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train_scaled[:,[0,1,6,11]],Y_train['Party'])
```

Out[75]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [76]:
```
y_pred = classifier.predict(x_val_scaled[:,[0,1,6,11]])
```

In [77]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [78]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

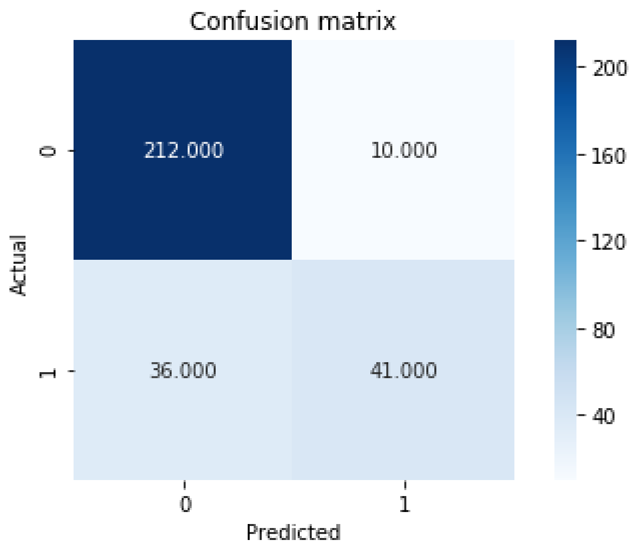
```
[0.842809364548495, 0.15719063545150502, 0.7678571428571429, 0.55844155844155
84, 0.8339490435009738]
```

In [79]:
```
#K-Nearest Neighbours using multiple variables and n=5
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train_scaled[:,[0,1,4,6,11]],Y_train['Party'])
```

Out[79]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

In [80]:
```
y_pred = classifier.predict(x_val_scaled[:,[0,1,4,6,11]])
```

In [81]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [82]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

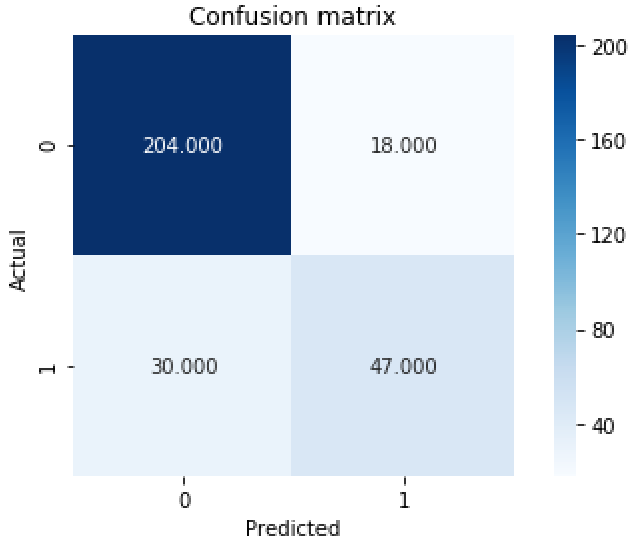
```
[0.8461538461538461, 0.15384615384615385, 0.803921568627451, 0.53246753246753
24, 0.8347841653027823]
```

In [83]:
```
#K-Nearest Neighbours using multiple variables and n=3
classifier = KNeighborsClassifier(n_neighbors = 3)
classifier.fit(x_train_scaled[:,[1,4,6,11]],Y_train['Party'])
```

Out[83]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [84]:
```
y_pred = classifier.predict(x_val_scaled[:,[1,4,6,11]])
```

In [85]:
```
conf_matrix = metrics.confusion_matrix(Y_val['Party'], 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()
```



In [86]:
```
accuracy = metrics.accuracy_score(Y_val['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(Y_val['Party'], y_pred)
recall = metrics.recall_score(Y_val['Party'], y_pred)
F1_score = metrics.f1_score(Y_val['Party'], y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8394648829431438, 0.1605351170568562, 0.7230769230769231, 0.61038961038961
04, 0.8347940131547956]
```

Considering our initial analysis in project 1

Attributes 'White population', ' Less than Bachelor's degree','Age 65 and over' favored Republican

Attributes 'Black population','Foreign Born','Age under 29' favored Democrats

Attributes 'Median Household income', 'Unemployment','Hispanic or Latino','Percent Rural' do not favor any party

So choosing these attributes to classify the labels with higher accuracy and f1 score

Classifier- SVM with parameters 'rbf'

Attributes- All except 'Percent white population' since it heavily favors Democrats

-What is the best performing classification model?

SVM with Kernel -RBF and attributes- All except 'Percent white population' since it heavily favors Democrats

-What is the performance of the model?

Accuracy -0.8562 F1 Score -0.8439

-How did you select the parameters of the model?

We chose the parameters by considering the ones that increase the accuracy and f1 score on a selectedd set of attributes

-How did you select the variables of the model? Considering our initial analysis in project 1

Attributes 'White population', ' Less than Bachelor's degree','Age 65 and over' favored Republican

Attributes 'Black population','Foreign Born','Age under 29' favored Democrats

Attributes 'Median Household income', 'Unemployment','Hispanic or Latino','Percent Rural' do not favor any party

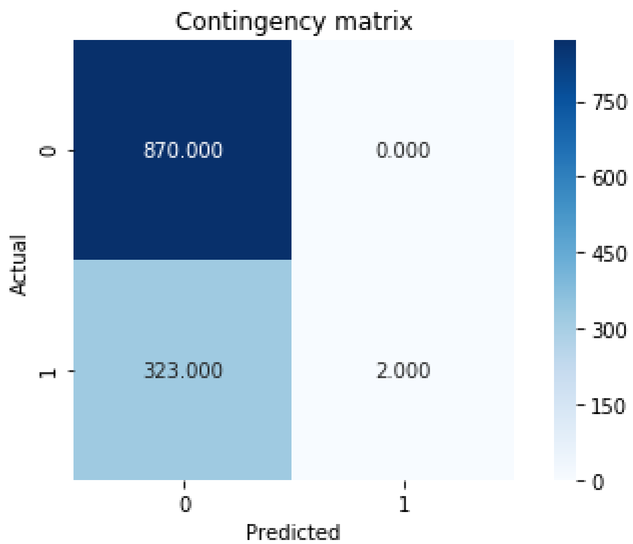
So choosing these attributes to classify the labels with higher accuracy and f1 score And selecting other attributes that increase the accuracy and f1 score

In [87]:
```
#************************ TASK 5 ******************************
X=df[df.columns[3:-3]]
Y=df[df.columns[-1]]
```

In [88]:
```
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

In [89]:
```
clustering = linkage(X_scaled[:,[0,1,2,3,4,9,10,11,12]], method="single", metr
ic="euclidean")
clusters = fcluster(clustering, 2, criterion="maxclust")
```

In [90]:
```
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()
```



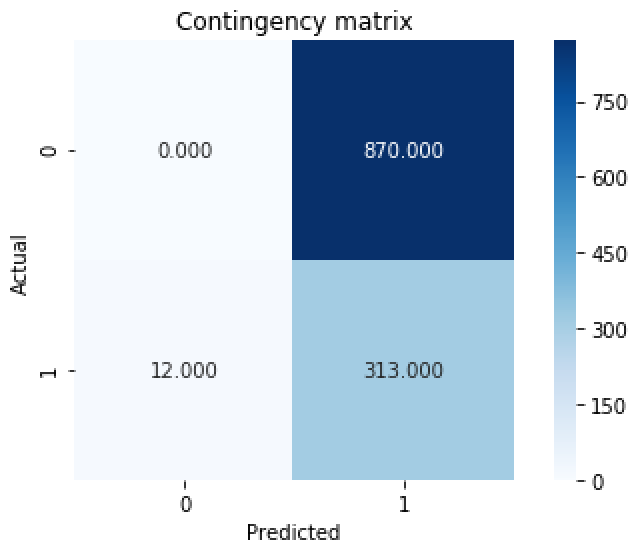
In [91]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X_scaled[:,[0,1,2,3,4,9,10,1
1,12]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.005608925119335567, 0.730755188271051]
```

In [92]:
```
#ax=df.plot(kind="scatter", x="Party", y="Total Population", c="clusters", col
ormap=plt.cm.brg)
```

In [93]:
```
clustering = linkage(X_scaled[:,[0,1,2,3,4,9,10,11,12]], method="complete", me
tric="euclidean")
clusters = fcluster(clustering, 2, criterion="maxclust")
```

In [94]:
```
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()
```



In [95]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X_scaled[:,[0,1,2,3,4,9,10,1
1,12]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

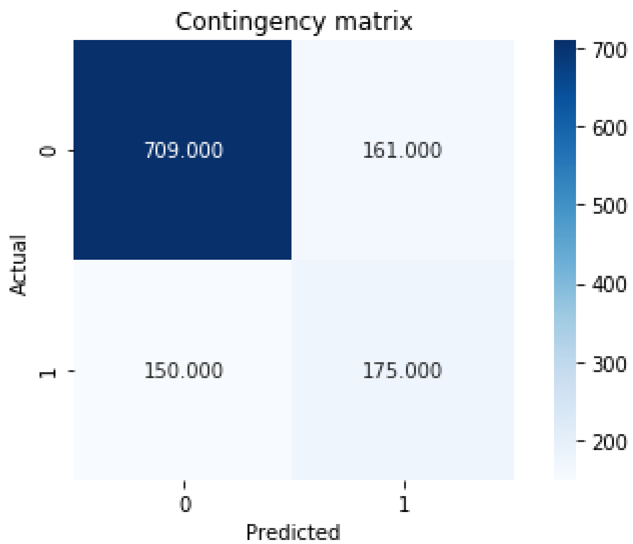
```
[0.03369805816713174, 0.6273018740347583]
```

In [96]:
```
#df["clusters"] = clusters
#ax=df.plot(kind="scatter", x="Party", y="Total Population", c="clusters", col
ormap=plt.cm.brg)
```

In [97]:
```
#----------------Kmeans---------------------------
```

In [98]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10,random_state=
0).fit(X_scaled)
clusters = clustering.labels_
```

In [99]:
```
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()
```



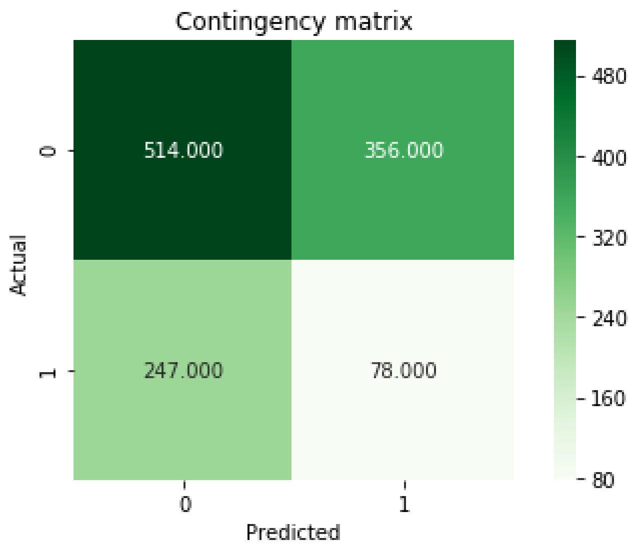
In [100]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X_scaled, clusters, metric =
"euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.19751656022671712, 0.30700290833697047]
```

In [101]:
```
# Plot clusters found using K-Means clustering
#df['clusters'] = clusters
#ax=df.plot(kind="scatter", x="Party", y="Total Population", c="clusters", col
ormap=plt.cm.brg)
```

In [102]:
```
clustering = DBSCAN(eps = 1, min_samples = 6, metric = "euclidean").fit(X_scal
ed)
clusters = clustering.labels_
```

In [103]:
```
cont_matrix = metrics.cluster.contingency_matrix(Y,clusters)
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Greens)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
```



In [104]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X_scaled, clusters, metric =
"euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[-0.016368494834228664, 0.05066332497562545]
```

In [105]:
```
#df['clusters'] = clusters
#ax=df.plot(kind="scatter", x="Party", y="Total Population", c="clusters", col
ormap=plt.cm.brg)
```

Task 6

In [106]:
```
x_train_scaled6=df.iloc[:,3:16]
y_train=df.iloc[:,-1]
```

In [107]:
```
x_train_scaled6.head()
```

Out[107]:

	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	M Hous In
0	72346	18.571863	0.486551	5.947806	1.719515	50.598513	45.854643	13.322091	
1	128177	56.299492	3.714395	34.403208	11.458374	49.069646	37.902276	19.756275	
2	138064	54.619597	1.342855	13.711033	4.825298	50.581614	48.946141	10.873943	
3	53179	63.222325	0.552850	18.548675	4.249798	50.296170	32.238290	26.397638	
4	37529	51.461536	1.811932	32.097844	4.385942	46.313518	46.393456	12.315809	

In [108]:
```
y_train.head()
```

Out[108]:
```
0    1
1    0
2    1
3    0
4    0
Name: Party, dtype: int64
```

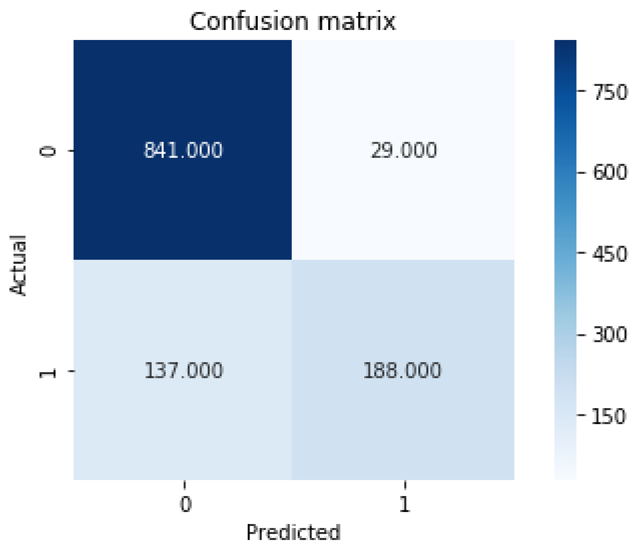
In [109]:
```
#Scaling numerical variables
# Standardize test data
scaler = StandardScaler()
scaler.fit(x_train_scaled6)
x_train_scaled6 = scaler.transform(x_train_scaled6)
```

In [110]:
```
#----------------------SVM using mltiple attributes-------------------------
------------
classifier = SVC(kernel = 'rbf')
classifier.fit(x_train_scaled6[:,[0,2,3,4,5,6,7,8,9,10,11,12]],y_train)
```

Out[110]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [111]:
```
y_pred = classifier.predict(x_train_scaled6[:,[0,2,3,4,5,6,7,8,9,10,11,12]])
```

In [112]:
```
conf_matrix = metrics.confusion_matrix(y_train, 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()
```

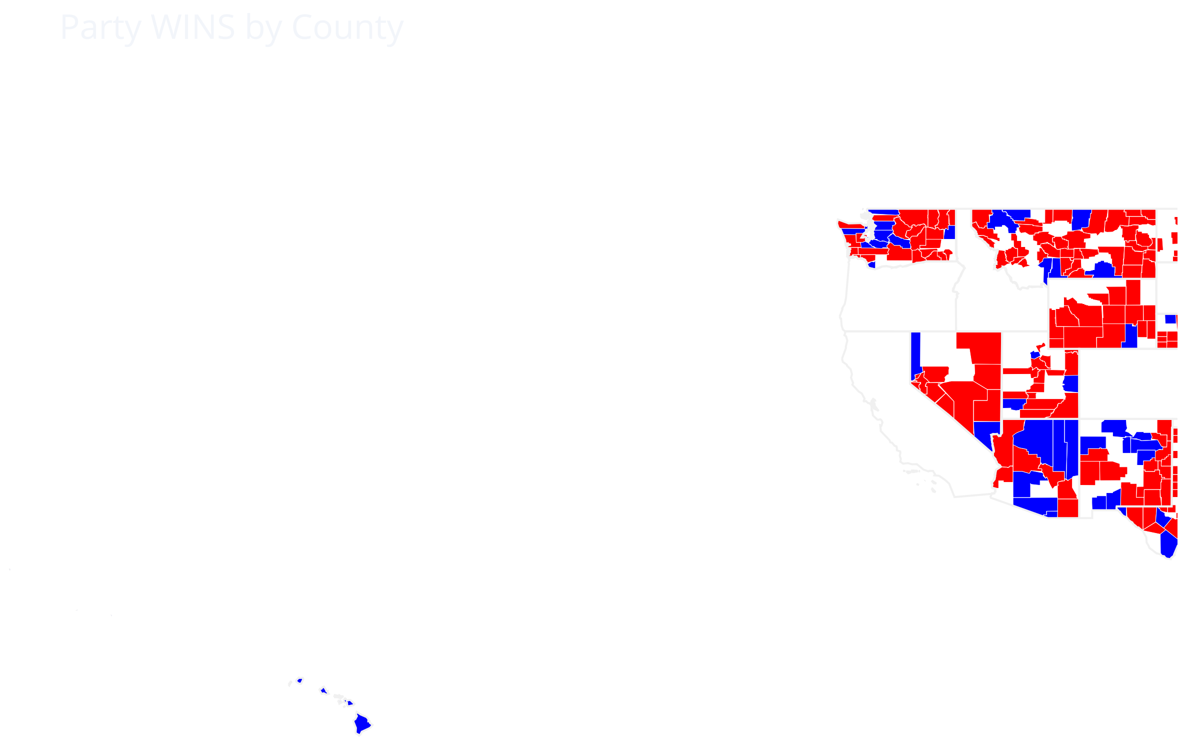


In [113]:
```
accuracy = metrics.accuracy_score(y_train, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_train, y_pred)
recall = metrics.recall_score(y_train, y_pred)
F1_score = metrics.f1_score(y_train, y_pred,average='weighted')
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8610878661087866, 0.13891213389121337, 0.8663594470046083, 0.5784615384615
385, 0.8513070326051845]
```

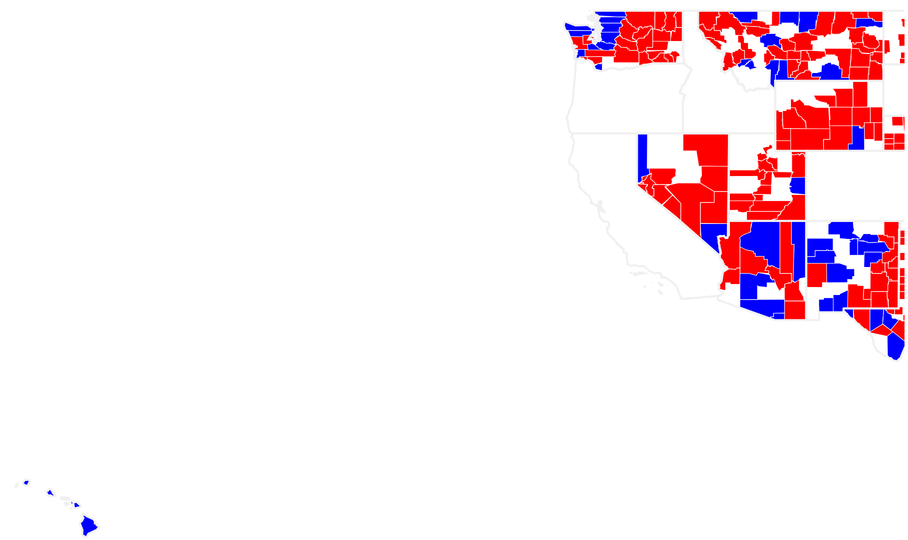
In [114]:
```
fips = df['FIPS'].tolist()
values=y_pred.tolist()

colorscale = [
    'rgb(255,0,0)',
    'rgb(0,0,255)'
]
fig=ff.create_choropleth(
                         fips=fips,colorscale = colorscale,values=values,scope
=['USA','HI'],
                         county_outline={'color':'white','width':0.2}

                        )

fig.layout['title'] = 'Party WINS by County'
fig.layout.template='plotly_dark'

fig.show()
#The next folowing line of code is only for rendering purposes
#fig.write_image("images/fig1.pdf")
```

```
C:\Users\ubemi\Anaconda3\lib\site-packages\pandas\core\frame.py:6692: FutureW
arning:

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'.
```



In [115]:
```
#For Original dataset
fips = df['FIPS']
values=df['Party']

colorscale = [
    'rgb(255,0,0)',
    'rgb(0,0,255)'
]

fig=ff.create_choropleth(
                         fips=fips,colorscale = colorscale,values=values,scope
=['USA','HI'],
                            county_outline={'color':'white','width':0.2}
                        )

fig.layout['title'] = 'Party WINS by County'
fig.layout.template= 'plotly_dark'


fig.show()
#The next folowing line of code is only for rendering purposes
#fig.write_image("images/fig1.pdf")
```

Party WINS by County



Classifying the Test set using the chosen model with parameters and attributes

In [116]:
```
import pandas as pd
data = pd.read_csv("demographics_test.csv")
data.head()
```

Out[116]:

	State	County	FIPS	Total Population	Percent White, not Hispanic or Latino	Percent Black, not Hispanic or Latino	Percent Hispanic or Latino	Percent Foreign Born	Percent Female	P A
0	NV	eureka	32011	1730	98.265896	0.057803	0.462428	0.346821	51.156069	27.1
1	TX	zavala	48507	12107	5.798299	0.594697	93.326175	9.193029	49.723301	49.3
2	VA	king george	51099	25260	73.804434	16.722090	4.441805	2.505938	50.166271	40.1
3	OH	hamilton	39061	805965	66.354867	25.654340	2.890944	5.086945	51.870615	40.7
4	TX	austin	48015	29107	63.809393	8.479060	25.502456	9.946061	50.671660	37.3

In [117]:
```
data.shape
```

Out[117]: (400, 16)

In [118]:
```
x_test_scaled=data.iloc[:,3:]
```

In [119]:
```
x_test_scaled.head()
```

Out[119]:

	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	M Hous In
0	1730	98.265896	0.057803	0.462428	0.346821	51.156069	27.109827	15.606936	
1	12107	5.798299	0.594697	93.326175	9.193029	49.723301	49.302057	12.480383	
2	25260	73.804434	16.722090	4.441805	2.505938	50.166271	40.186065	11.868567	
3	805965	66.354867	25.654340	2.890944	5.086945	51.870615	40.779686	14.161657	
4	29107	63.809393	8.479060	25.502456	9.946061	50.671660	37.351840	17.799842	

In [120]:
```
#Scaling numerical variables
# Standardize test data
scaler = StandardScaler()
scaler.fit(X_train)
x_train_scaled = scaler.transform(X_train)
x_test_scaled = scaler.transform(x_test_scaled)
x_test_scaled.shape
```

Out[120]: (400, 13)

In [121]:
```
#Merging scaled and other value
x_test_merge=np.concatenate([np.array(data.iloc[:,0:3]),x_test_scaled],axis=1)
```

In [122]:
```
x_test_merge.shape
```

Out[122]:
```
(400, 16)
```

In [123]:
```
#Predicting Democratic votes using selected attributes
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,[0, 11, 9, 2]],y=Y_train.iloc[:,
0])
print(fitted_model.coef_)
predicted_demo = fitted_model.predict(X=x_test_scaled[:,[0, 11, 9, 2]])
```

```
[70484.68208406 -9860.1194576   1648.91035669  1392.98545067]
```

In [124]:
```
predicted_demo.shape
```

Out[124]:
```
(400,)
```

In [125]:
```
predicted_demo[0:50]
```

Out[125]:
```
array([ -6024.19745779,  -6969.50285341,  22049.55580997, 183916.24158932,
         6705.1164912 ,   9702.31461902,  13947.69864219,  34278.96273618,
       100413.7281802 ,  28674.05187982, -11253.02270416,  -5347.44993859,
        -3446.95628052,  20649.49585335, 146790.7196577 ,  -4606.35407089,
         4227.32667003,  71177.07659226,  39776.44738931,   6111.817256  ,
        -2410.20884863,  -6610.79983302,  -5723.3989571 ,   1951.03312637,
         -891.76941812,  -3664.79926432,   7135.30796894,  50165.9284927 ,
         5876.69864661,  29797.24296704,  88406.35340686,  12270.97372962,
        66583.31453208,  -3272.82222137,   2887.8473437 ,   9785.19752946,
         8212.57663177,   2396.64880875,   4472.53810407,  -2604.53721651,
        15165.97887305,  20390.0093405 ,  46656.4432557 ,  37986.49126741,
        -4857.76849921, 120051.66116874,  -8331.0472549 ,  -4137.50193053,
        27744.11477966,  14226.33789101])
```

In [126]:
```
result_demo=np.concatenate([x_test_merge[:,0:2],predicted_demo[:,None]],axis=1
)
result_demo1=np.concatenate([x_test_merge[:,0:3],predicted_demo[:,None]],axis=
1)
```

In [127]:
```
result_demo[0:10]
```

Out[127]:
```
array([['NV', 'eureka', -6024.197457785998],
       ['TX', 'zavala', -6969.502853408365],
       ['VA', 'king george', 22049.555809972666],
       ['OH', 'hamilton', 183916.24158932114],
       ['TX', 'austin', 6705.1164911999185],
       ['MI', 'barry', 9702.314619023346],
       ['NM', 'valencia', 13947.698642187906],
       ['TX', 'ellis', 34278.96273617705],
       ['NJ', 'mercer', 100413.72818020337],
       ['PA', 'cambria', 28674.05187982477]], dtype=object)
```

In [128]:
```
#Predicting Republican votes using selected attributes
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled[:,[0,1,3,4,7,8,9,10,11,12]],y=Y_train.iloc[:,1])
print(fitted_model.coef_)
predicted_rep = fitted_model.predict(X=x_test_scaled[:,[0,1,3,4,7,8,9,10,11,12]])
```

```
[45220.29033736  5065.79031785  3840.00607902 -5872.48026158
  2839.30463983  6066.05645897  2063.52147931  2586.73805496
 -2432.72004918 -5176.17669614]
```

In [129]:
```
predicted_rep.shape
```

Out[129]:
```
(400,)
```

In [130]:
```
predicted_rep[0:50]
```

Out[130]:
```
array([  8448.1371131 ,   4049.96280682,  19807.21462961, 114298.5293142 ,
         6386.57639088,  14633.71966504,  17024.35221377,  29730.06181932,
        55620.65955875,  28811.48695693,   -537.17941533,  13890.97176422,
         6510.78435913,   4635.54292584,  98752.76986795,   3427.20111128,
         2255.83320775,  62426.94677451,  35551.94834098,  16331.98393355,
         4962.67008321,  -9272.37385646,  -3130.04361176,   5891.94681573,
         7839.25142585,   7293.13354901,  10765.87948668,  36570.61900946,
         1590.84514293,  25271.53746993,  59375.56752416,  -1174.14000816,
        42806.99337793,    930.76625423,   7056.81273428,  16351.59219585,
        11264.75821966,   8939.45281196,   7114.71408548,   6124.06342319,
        10763.11165727,  15779.1448899 ,  37157.25556503,  22357.32611469,
        -6993.00830573,  84845.64134817,   2816.01904595, -13088.03253914,
        24898.27443261,   9188.98431561])
```

In [131]:
```
result_votes=np.concatenate([result_demo,predicted_rep[:,None]],axis=1)
result_votes1=np.concatenate([result_demo1,predicted_rep[:,None]],axis=1)
```

In [132]:
```
result_votes[0:10]
```

Out[132]:
```
array([['NV', 'eureka', -6024.197457785998, 8448.13711310336],
       ['TX', 'zavala', -6969.502853408365, 4049.962806815296],
       ['VA', 'king george', 22049.555809972666, 19807.21462960806],
       ['OH', 'hamilton', 183916.24158932114, 114298.52931419964],
       ['TX', 'austin', 6705.1164911999185, 6386.576390883141],
       ['MI', 'barry', 9702.314619023346, 14633.719665042463],
       ['NM', 'valencia', 13947.698642187906, 17024.35221377254],
       ['TX', 'ellis', 34278.96273617705, 29730.061819320897],
       ['NJ', 'mercer', 100413.72818020337, 55620.659558749976],
       ['PA', 'cambria', 28674.05187982477, 28811.48695692587]],
      dtype=object)
```

Using SVM to predict the Party label with selected attributes and parameters

In [133]:
```
classifier = SVC(kernel = 'rbf')
classifier.fit(x_train_scaled[:,[0,2,3,4,5,6,7,8,9,10,11,12]],Y_train['Party'
])
```

Out[133]:
```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
    kernel='rbf', max_iter=-1, probability=False, random_state=None,
    shrinking=True, tol=0.001, verbose=False)
```

In [134]:
```
pred_labels = classifier.predict(x_test_scaled[:,[0,2,3,4,5,6,7,8,9,10,11,12
]])
```

In [135]:
```
pred_labels[0:10]
```

Out[135]:
```
array([0, 1, 0, 1, 0, 0, 0, 0, 1, 0], dtype=int64)
```

In [136]:
```
FinalResult =np.concatenate([result_votes,pred_labels[:,None]],axis=1)
FinalResult1 =np.concatenate([result_votes1,pred_labels[:,None]],axis=1)
```

In [137]:
```
FinalResult[0:50]
```

Out[137]:
```
array([['NV', 'eureka', -6024.197457785998, 8448.13711310336, 0],
       ['TX', 'zavala', -6969.502853408365, 4049.962806815296, 1],
       ['VA', 'king george', 22049.555809972666, 19807.21462960806, 0],
       ['OH', 'hamilton', 183916.24158932114, 114298.52931419964, 1],
       ['TX', 'austin', 6705.1164911999185, 6386.576390883141, 0],
       ['MI', 'barry', 9702.314619023346, 14633.719665042463, 0],
       ['NM', 'valencia', 13947.698642187906, 17024.35221377254, 0],
       ['TX', 'ellis', 34278.96273617705, 29730.061819320897, 0],
       ['NJ', 'mercer', 100413.72818020337, 55620.659558749976, 1],
       ['PA', 'cambria', 28674.05187982477, 28811.48695692587, 0],
       ['IN', 'switzerland', -11253.022704164916, -537.179415334198, 0],
       ['NV', 'lander', -5347.449938586022, 13890.97176421507, 0],
       ['NE', 'cherry', -3446.9562805248133, 6510.7843591255805, 0],
       ['VA', 'radford city', 20649.49585335145, 4635.542925836518, 1],
       ['FL', 'lee', 146790.71965769678, 98752.76986794514, 0],
       ['MI', 'arenac', -4606.354070889967, 3427.2011112836262, 0],
       ['TX', 'shackelford', 4227.326670028375, 2255.833207747317, 0],
       ['NJ', 'gloucester', 71177.07659225757, 62426.946774506796, 0],
       ['OH', 'trumbull', 39776.447389311674, 35551.94834097943, 0],
       ['OH', 'lawrence', 6111.8172560003295, 16331.983933554318, 0],
       ['ND', 'burke', -2410.2088486253742, 4962.67008321255, 0],
       ['TX', 'hardeman', -6610.799833020032, -9272.373856460184, 0],
       ['NE', 'keya paha', -5723.398957104731, -3130.043611760877, 0],
       ['VA', 'norton city', 1951.033126371036, 5891.946815732452, 1],
       ['ND', 'bowman', -891.7694181218612, 7839.251425850991, 0],
       ['UT', 'duchesne', -3664.7992643152866, 7293.133549005066, 0],
       ['MN', 'carlton', 7135.307968940106, 10765.879486679254, 0],
       ['FL', 'okaloosa', 50165.92849270195, 36570.619009456495, 1],
       ['TX', 'oldham', 5876.698646612975, 1590.8451429272645, 0],
       ['MT', 'lewis and clark', 29797.24296703842, 25271.537469930496,
        0],
       ['NY', 'rockland', 88406.3534068598, 59375.567524162645, 1],
       ['TX', 'waller', 12270.973729624407, -1174.1400081634565, 0],
       ['VA', 'falls church city', 66583.31453208286, 42806.99337793428,
        1],
       ['PA', 'potter', -3272.822221371145, 930.7662542251637, 0],
       ['MI', 'gratiot', 2887.8473437005123, 7056.8127342763655, 0],
       ['MI', 'shiawassee', 9785.197529456611, 16351.592195850704, 0],
       ['MN', 'polk', 8212.576631771899, 11264.758219664274, 0],
       ['ND', 'billings', 2396.648808748123, 8939.452811962861, 0],
       ['ND', 'mckenzie', 4472.538104065949, 7114.714085481639, 0],
       ['WY', 'weston', -2604.5372165075823, 6124.06342318819, 0],
       ['MT', 'jefferson', 15165.97887305365, 10763.111657272366, 0],
       ['VT', 'bennington', 20390.009340504996, 15779.144889896157, 0],
       ['FL', 'clay', 46656.443255695434, 37157.25556503171, 1],
       ['AZ', 'yuma', 37986.49126741439, 22357.32611469449, 1],
       ['TX', 'terrell', -4857.76849921095, -6993.008305726038, 0],
       ['MA', 'bristol', 120051.6611687351, 84845.64134817488, 1],
       ['WV', 'webster', -8331.047254895453, 2816.0190459522346, 0],
       ['TX', 'collingsworth', -4137.501930530125, -13088.032539137017,
        0],
       ['NY', 'chautauqua', 27744.11477965668, 24898.274432609353, 0],
       ['TN', 'fayette', 14226.337891005938, 9188.98431561122, 0]],
      dtype=object)
```

In [138]:
```
#FinalResult  = pd.DataFrame(FinalResult, index=X_train.index, columns=data.co
lumns[0,[0,1,14,15,16]])
Final=pd.DataFrame({'State':FinalResult[:,0],'County':FinalResult[:,1],'Democr
atic Votes':FinalResult[:,2],'Republican Votes':FinalResult[:,3],'Party':Final
Result[:,4]})
```

In [139]:
```
#FinalResult  = pd.DataFrame(FinalResult, index=X_train.index, columns=data.co
lumns[0,[0,1,14,15,16]])
Final1=pd.DataFrame({'State':FinalResult1[:,0],'County':FinalResult1[:,1],'FIP
S':FinalResult1[:,2],'Democratic Votes':FinalResult1[:,3],'Republican Votes':F
inalResult1[:,4],'Party':FinalResult1[:,5]})
```

In [140]:
```
Final['Democratic Votes'] = np.where(Final['Democratic Votes'] < 0, 0, Final[
'Democratic Votes'])
Final['Republican Votes'] = np.where(Final['Republican Votes'] < 0, 0, Final[
'Republican Votes'])
```

In [141]:
```
Final.head()
```

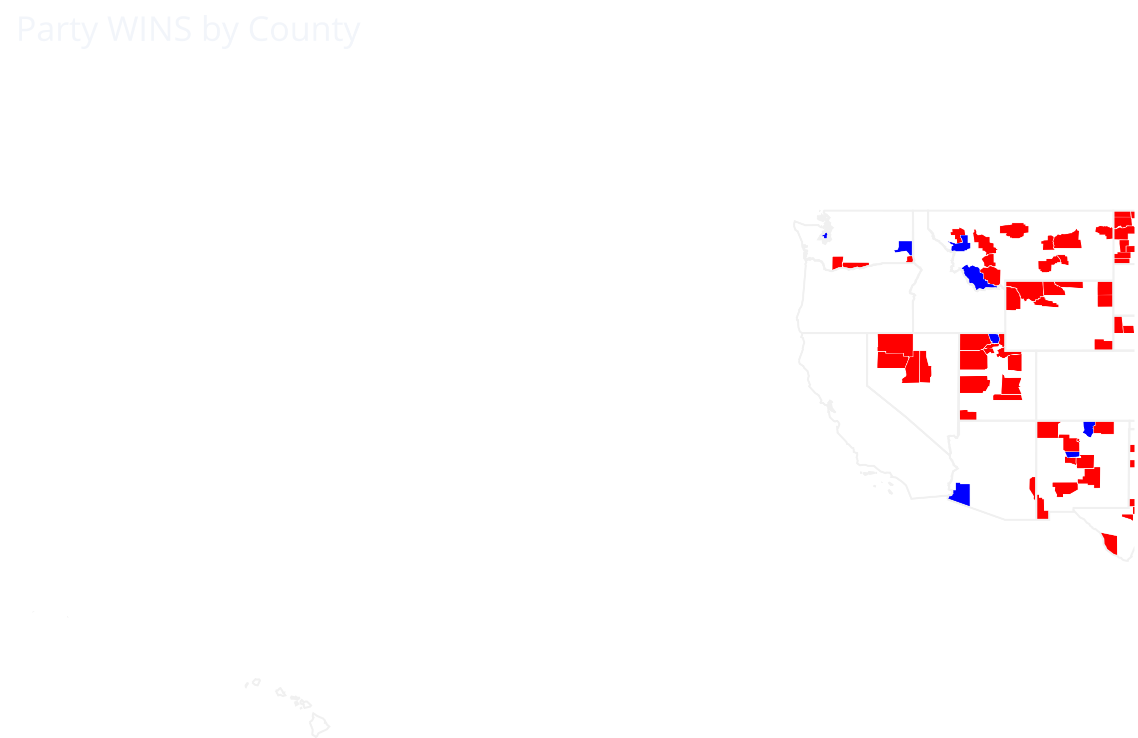
Out[141]:

	State	County	Democratic Votes	Republican Votes	Party
0	NV	eureka	0	8448.14	0
1	TX	zavala	0	4049.96	1
2	VA	king george	22049.6	19807.2	0
3	OH	hamilton	183916	114299	1
4	TX	austin	6705.12	6386.58	0

In [142]:
```
Final.to_csv("output.csv",index=False)
```

In [143]:
```
#******************************Task 10***************************

fips = Final1['FIPS']
values=Final1['Party']

colorscale = [
    'rgb(255,0,0)',
    'rgb(0,0,255)'
]

fig=ff.create_choropleth(
                         fips=fips,colorscale = colorscale,values=values,scope
=['USA','HI'],
                         county_outline={'color':'white','width':0.2})

fig.layout['title'] = 'Party WINS by County'
fig.layout.template= 'plotly_dark'


fig.show()
#The next folowing line of code is only for rendering purposes
#fig.write_image("images/fig1.pdf")
```

```
C:\Users\ubemi\Anaconda3\lib\site-packages\pandas\core\frame.py:6692: FutureW
arning:

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'.
```



In [ ]: