In [1]:
```
import pandas as pd
import numpy
import matplotlib.pyplot as plt
import seaborn as sns
import statistics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn import preprocessing
from sklearn.linear_model import LinearRegression
from scipy import stats
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import metrics
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.cluster import KMeans, DBSCAN
```

In [2]:
```
# Load dataset
data = pd.read_csv("merged_train.csv")
test= pd.read_csv("demographics_test.csv")
```

Task 01 (of 07): Performed hold out split

In [3]:
```
#task 1 Performed hold out split
x_train, x_val, y_train, y_val = train_test_split(data[['FIPS','Total Populati
on','Percent White, not Hispanic or Latino',
                                                       'Percent Black, not
 Hispanic or Latino','Percent Hispanic or Latino',
                                                       'Percent Foreign Bor
n','Percent Female','Percent Age 29 and Under',
                                                       'Percent Age 65 and
 Older','Median Household Income','Percent Unemployed',
                                                       'Percent Less than H
igh School Degree',"Percent Less than Bachelor's Degree",
                                                       'Percent Rural','Dem
ocratic']],
                                                    data['Democratic'], test_s
ize=0.25, random_state = 0)
x1_train, x1_val, y1_train, y1_val = train_test_split(data[['FIPS','Total Popu
lation','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 th
an High School Degree',"Percent Less than Bachelor's Degree",
                                                         'Percent Rural',
'Republican']],
                                                    data['Republican'], test_s
ize=0.25, random_state = 0)


x_test = test[['FIPS','Total Population','Percent White, not Hispanic or Latin
o',
               'Percent Black, not Hispanic or Latino','Percent Hispanic or La
tino',
               'Percent Foreign Born','Percent Female','Percent Age 29 and Und
er',
               'Percent Age 65 and Older','Median Household Income','Percent U
nemployed',
               'Percent Less than High School Degree',"Percent Less than Bache
lor's Degree",
               'Percent Rural']]
```

Task 02 (of 07): standardizing the training and test set for both democratic and republic

In [4]:
```
#task2 - standardizing the training and test set for both democratic and repub
lic
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_train_scaled = pd.DataFrame(x_train_scaled, columns = x_train.columns)
x_val_scaled = scaler.transform(x_val)
x_val_scaled = pd.DataFrame(x_val_scaled, columns = x_val.columns)
scaler = StandardScaler()
scaler.fit(x1_train)
x1_train_scaled = scaler.transform(x1_train)
x1_train_scaled = pd.DataFrame(x1_train, columns = x1_train.columns)
x1_val_scaled = scaler.transform(x1_val)
x1_val_scaled = pd.DataFrame(x1_val, columns = x1_train.columns)
```

Task 03 (of 07): Predict democratic party voters in each county using on variable

In [5]:
```
# Task3- democratic party in each county
#build a model using MULTIPLE variable
model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[['Total Population']],y_train)
predicted = fitted_model.predict(x_val_scaled['Total Population'].values.resha
pe(-1,1))
corr_coef = numpy.corrcoef(y_val,predicted)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-1-1)
print("One variable r2="+str(R_squared)+"  adj. r2="+str(adj_r_sq))

#Lasso
model1 = linear_model.LinearRegression()
model1 = linear_model.Lasso(alpha = 1).fit(x_train_scaled[['Total Population'
]],y_train)
predicted = model1.predict(x_val_scaled['Total Population'].values.reshape(-1,
1))
corr_coef = numpy.corrcoef(y_val,predicted)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-1-1)
print("lasso-one variable r2="+str(R_squared)+"  adj.r2= "+str(adj_r_sq))


#Ridge
model2 = linear_model.LinearRegression()
model2 = linear_model.Ridge(alpha = 1).fit(x_train_scaled[['Total Population'
]],y_train)
predicted = model2.predict(x_val_scaled['Total Population'].values.reshape(-1,
1))
corr_coef = numpy.corrcoef(y_val,predicted)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-1-1)
print("ridge one variable ="+str(R_squared)+"  adj.r2= "+str(adj_r_sq))
print("=================================================================
===")
```

```
One variable r2=0.9436415220931658  adj. r2=0.9713158723670933
lasso-one variable r2=0.9436415220931673  adj.r2= 0.971315872367094
ridge one variable =0.9436415220931669  adj.r2= 0.9713158723670938
================================================================================
```

Task 03 (of 07): Predict democratic party voters in each county using multpile variable

In [6]:
```
# Task3- democratic party in each county
#build a model using multiple variable

model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[['Total Population', 'Percent White, n
ot Hispanic or Latino', 'Percent Black, not Hispanic or Latino', 'Percent Hisp
anic or Latino',
                                         'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']],y_train)
predicted = fitted_model.predict(x_val_scaled[['Total Population', 'Percent Wh
ite, not Hispanic or Latino', 'Percent Black, not Hispanic or Latino', 'Percen
t Hispanic or Latino',
                                         'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']])
corr_coef = numpy.corrcoef(y_val,predicted)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-8-1)
print("multiple variable = "+str(R_squared)+"  adj.r2= "+str(adj_r_sq))

#Lasso
model = linear_model.Lasso(alpha = 1).fit(x_train_scaled[['Total Population',
'Percent White, not Hispanic or Latino', 'Percent Black, not Hispanic or Latin
o', 'Percent Hispanic or Latino',
                                         'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']],y_train)
predictedl = model.predict(x_val_scaled[['Total Population', 'Percent White, n
ot Hispanic or Latino', 'Percent Black, not Hispanic or Latino', 'Percent Hisp
anic or Latino',
                                         'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']])
corr_coef = numpy.corrcoef(y_val,predictedl)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-8-1)
print("lasso multiple variable = "+str(R_squared)+"  adj.r2= "+str(adj_r_sq))


#Ridge
FINAL_REG_model = linear_model.Ridge(alpha = 1).fit(x_train_scaled[['Total Pop
ulation', 'Percent White, not Hispanic or Latino', 'Percent Black, not Hispani
c or Latino', 'Percent Hispanic or Latino',
                                         'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']],y_train)
predictedr = FINAL_REG_model.predict(x_val_scaled[['Total Population', 'Percen
t White, not Hispanic or Latino', 'Percent Black, not Hispanic or Latino', 'Pe
rcent Hispanic or Latino',
                                         'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']])
corr_coef = numpy.corrcoef(y_val,predictedr)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-8-1)
print("ridge multiple variable =" +str(R_squared)+"  adj.r2= "+str(adj_r_sq))
print("==========================================================================
==============")
print("comparing all the models and their r2 and adj r2 values, lasso is best
 for multpile variable")
```

```
multiple variable = 0.9278780606334682  adj.r2= 0.9622508741653365
lasso multiple variable = 0.9278901279994416  adj.r2= 0.9622573107263362
ridge multiple variable =0.9277408329226396  adj.r2= 0.9621776759159704
================================================================================
=========
comparing all the models and their r2 and adj r2 values, lasso is best for mu
ltpile variable
```

Task 03 (of 07): Predict republic party voters in each county using one variable

In [7]:
```
# Task3- republican party in each county
#build a model using one variable
model = linear_model.LinearRegression()
fitted_model = model.fit(x1_train_scaled[['Total Population']],y1_train)
predicted = fitted_model.predict(x1_val_scaled['Total Population'].values.resh
ape(-1,1))
corr_coef = numpy.corrcoef(y_val,predicted)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-1-1)
print("One variable r2="+str(R_squared)+"  adj. r2="+str(adj_r_sq))

#Lasso
model = linear_model.Lasso(alpha = 1).fit(x_train_scaled[['Total Population'
]],y_train)
predictedl = fitted_model.predict(x_val_scaled['Total Population'].values.resh
ape(-1,1))
corr_coef = numpy.corrcoef(y_val,predictedl)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-1-1)
print("lasso-one variable r2="+str(R_squared)+"  adj.r2= "+str(adj_r_sq))

#Ridge
model = linear_model.Ridge(alpha = 1).fit(x_train_scaled[['Total Population'
]],y_train)
predictedr = fitted_model.predict(x_val_scaled['Total Population'].values.resh
ape(-1,1))
corr_coef = numpy.corrcoef(y_val,predictedr)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-1-1)
print("ridge-one variable r2="+str(R_squared)+"  adj.r2= "+str(adj_r_sq))
print("================================================================================
===")
```

```
One variable r2=0.9436415220931658  adj. r2=0.9713158723670933
lasso-one variable r2=0.9436415220931962  adj.r2= 0.971315872367109
ridge-one variable r2=0.9436415220931962  adj.r2= 0.971315872367109
================================================================================
```

Task 03 (of 07): Predict republic party voters in each county using multiplt variable

In [8]:
```
#multiple variables

#lasso
FINAL_REG_repub_model = linear_model.Lasso(alpha = 1).fit(x_train_scaled[['Tot
al Population', 'Percent White, not Hispanic or Latino', 'Percent Black, not H
ispanic or Latino', 'Percent Hispanic or Latino',
                                    'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']],y_train)
predictedr = FINAL_REG_repub_model.predict(x_val_scaled[['Total Population',
'Percent White, not Hispanic or Latino', 'Percent Black, not Hispanic or Latin
o', 'Percent Hispanic or Latino',
                                    'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']])
corr_coef = numpy.corrcoef(y_val,predictedr)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-8-1)
print("lasso-MULTIPLE variable r2="+str(R_squared)+"  adj.r2= "+str(adj_r_sq))

#Ridge
model = linear_model.Ridge(alpha = 1).fit(x_train_scaled[['Total Population',
'Percent White, not Hispanic or Latino', 'Percent Black, not Hispanic or Latin
o', 'Percent Hispanic or Latino',
                                    'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']],y_train)
predictedr = model.predict(x_val_scaled[['Total Population', 'Percent White, n
ot Hispanic or Latino', 'Percent Black, not Hispanic or Latino', 'Percent Hisp
anic or Latino',
                                    'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']])
corr_coef = numpy.corrcoef(y_val,predictedr)[1, 0]
R_squared = corr_coef**2
adj_r_sq = 1- ((1-corr_coef)*(len(y_val)-1))/(len(y_val)-8-1)
print("Ridge-one variable r2="+str(R_squared)+"  adj.r2= "+str(adj_r_sq))
print("comparing all the models and their r2 and adj r2 values, lasso is best
 for MULIPLE variable")

print("==========================================================================
===")
```

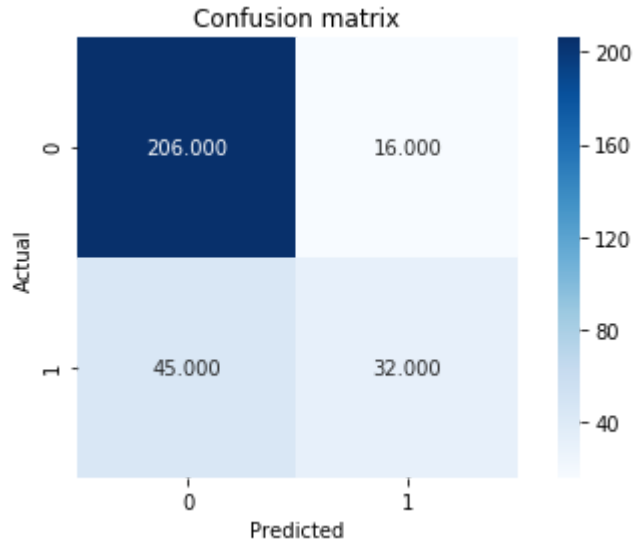
```
lasso-MULTIPLE variable r2=0.9278901279994416  adj.r2= 0.9622573107263362
Ridge-one variable r2=0.9277408329226396  adj.r2= 0.9621776759159704
comparing all the models and their r2 and adj r2 values, lasso is best for MU
LIPLE variable
================================================================================
```

Task 04 (of 07): Build a classification model to classify each county for each party variable commbination 1

In [9]:
```
#task 4 classification model to classify each county
#multiple variable combination 1

x_train, x_val, y_train, y_val = train_test_split(data[['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']],
                                                    data['Party'], test_size=0.25, random_state = 0)
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_val_scaled = scaler.transform(x_val)
```

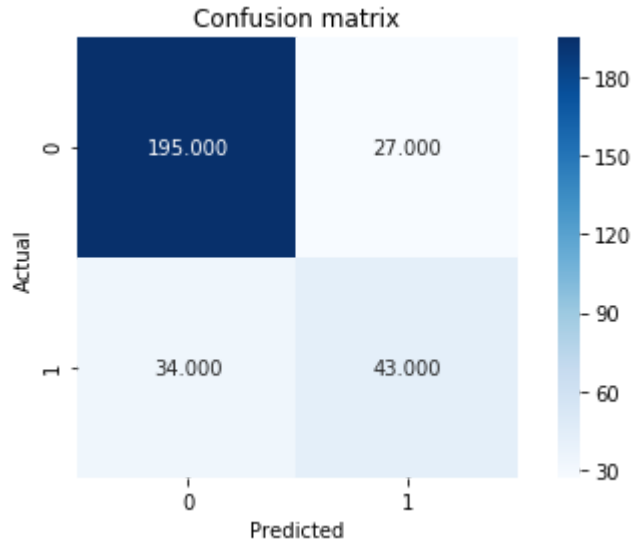
In [10]:
```
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train_scaled, y_train)
y_pred = classifier.predict(x_val_scaled)
accuracy = metrics.accuracy_score(y_val, y_pred)

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, error, precision, recall, F1_score])

conf_matrix = metrics.confusion_matrix(y_val, 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()
```

```
[0.7959866220735786, 0.20401337792642138, array([0.82071713, 0.66666667]), ar
ray([0.92792793, 0.41558442]), array([0.87103594, 0.512     ])]
```



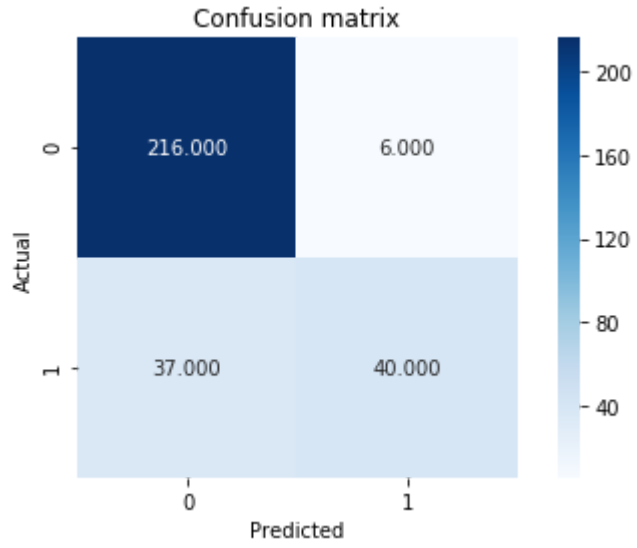
In [11]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled, y_train)
y_pred = classifier.predict(x_val_scaled)
conf_matrix = metrics.confusion_matrix(y_val, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()

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, error, precision, recall, F1_score])
```

```
[0.7959866220735786, 0.20401337792642138, array([0.85152838, 0.61428571]), ar
ray([0.87837838, 0.55844156]), array([0.86474501, 0.58503401])]
```



In [12]:
```
FINAL_classifier = SVC(kernel = 'rbf')
FINAL_classifier.fit(x_train_scaled, y_train)
y_pred = FINAL_classifier.predict(x_val_scaled)
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, error, precision, recall, F1_score])

conf_matrix = metrics.confusion_matrix(y_val, 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()
```

```
[0.8561872909698997, 0.14381270903010035, array([0.85375494, 0.86956522]), ar
ray([0.97297297, 0.51948052]), array([0.90947368, 0.6504065 ])]
```

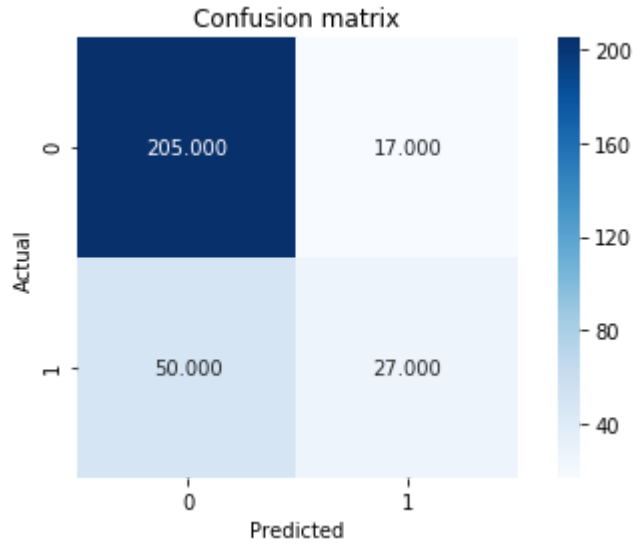


Task 04 (of 07): Build a classification model to classify each county for each party variable commbination 2

In [13]:
```
#multiple combination 2
x_train, x_val, y_train, y_val = train_test_split(data[['Total Population','Percent White, not Hispanic or Latino',
                                                        'Percent Unemployed',
                                                        'Percent Less than High School Degree',"Percent Less than Bachelor's Degree",
                                                        'Percent Rural']],
                                                  data['Party'], test_size=0.25, random_state = 0)
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_val_scaled = scaler.transform(x_val)
```

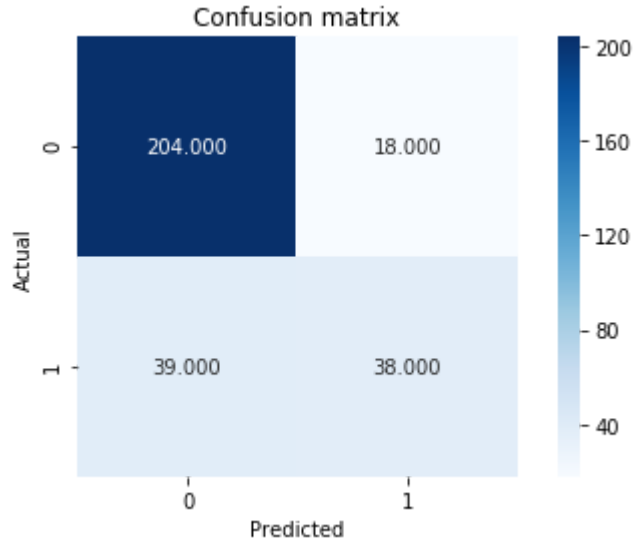
In [14]:
```
classifier = KNeighborsClassifier(n_neighbors = 18)
classifier.fit(x_train_scaled, y_train)
y_pred = classifier.predict(x_val_scaled)
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred, average = None)
recall = metrics.recall_score(y_val, y_pred, average = None)
F1_score = metrics.f1_score(y_val, y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])

conf_matrix = metrics.confusion_matrix(y_val, 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()
```

```
[0.7759197324414716, 0.2240802675585284, array([0.80392157, 0.61363636]), array([0.92342342, 0.35064935]), array([0.85953878, 0.44628099])]
```



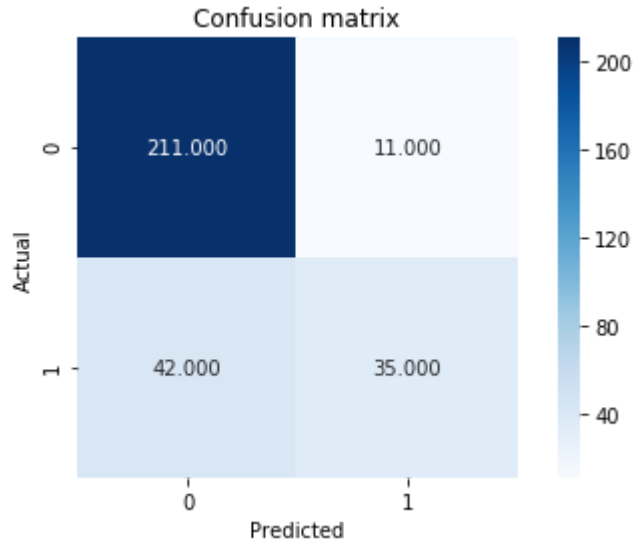
In [15]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled, y_train)
y_pred = classifier.predict(x_val_scaled)
conf_matrix = metrics.confusion_matrix(y_val, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()

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, error, precision, recall, F1_score])
```

```
[0.8093645484949833, 0.1906354515050167, array([0.83950617, 0.67857143]), arr
ay([0.91891892, 0.49350649]), array([0.87741935, 0.57142857])]
```



In [16]:
```
classifier = SVC(kernel = 'rbf')
classifier.fit(x_train_scaled, y_train)
y_pred = classifier.predict(x_val_scaled)
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, error, precision, recall, F1_score])

conf_matrix = metrics.confusion_matrix(y_val, 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()
```

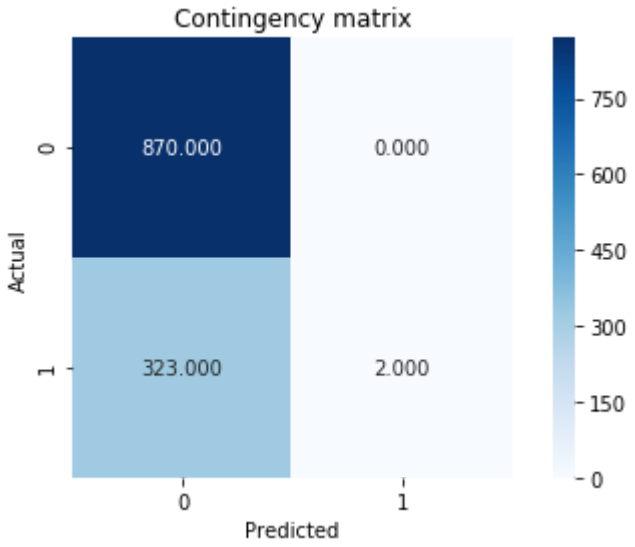
```
[0.822742474916388, 0.17725752508361203, array([0.83399209, 0.76086957]), arr
ay([0.95045045, 0.45454545]), array([0.88842105, 0.56910569])]
```

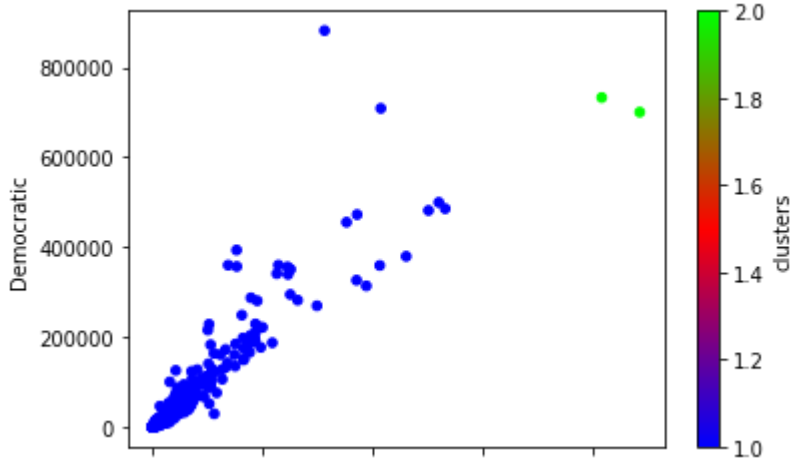


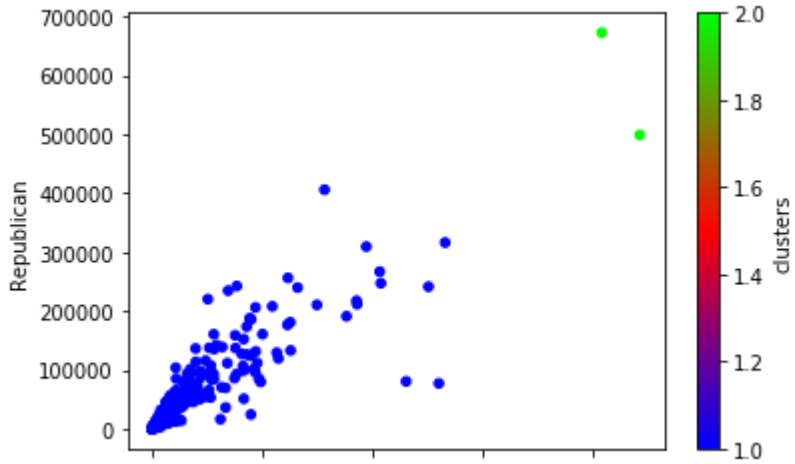
Task 05 (of 07): Build a Clustering model with variable combination 1

In [17]:
```
#task 5- build the clustering model
X = data[['Total Population']]
Y = data['Party']

scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
clustering = linkage(X_scaled, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters.shape

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()
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])
data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.005608925119335567, 0.9531008389502824]
```

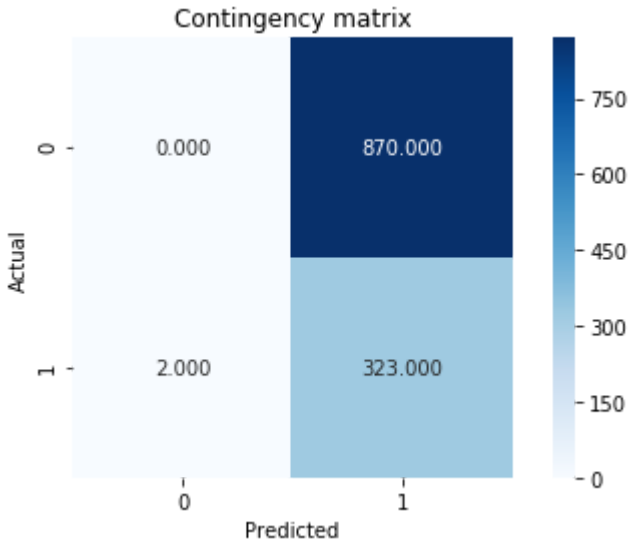


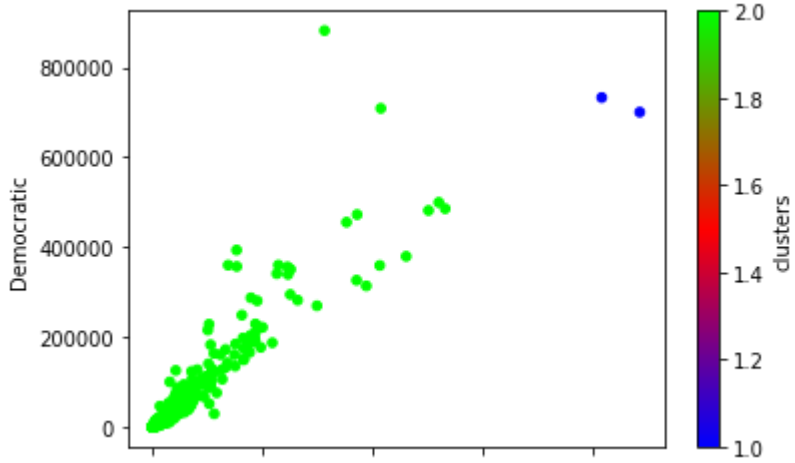


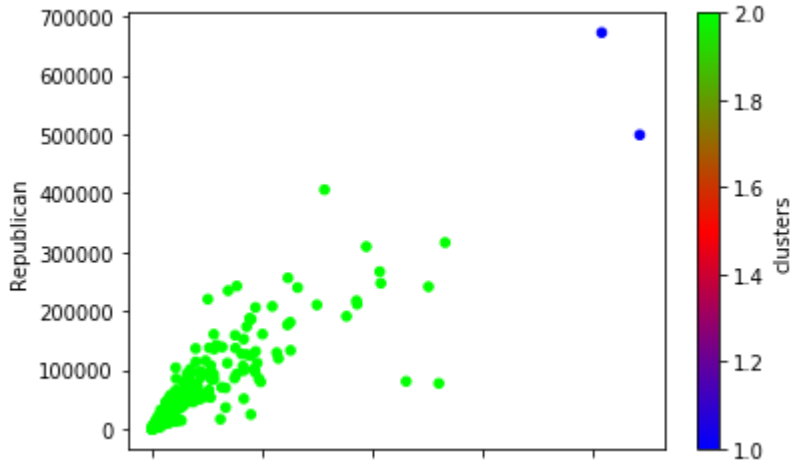


In [18]:
```
clustering = linkage(X_scaled, method = "complete", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters.shape


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()

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])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.005608925119335567, 0.9531008389502824]
```

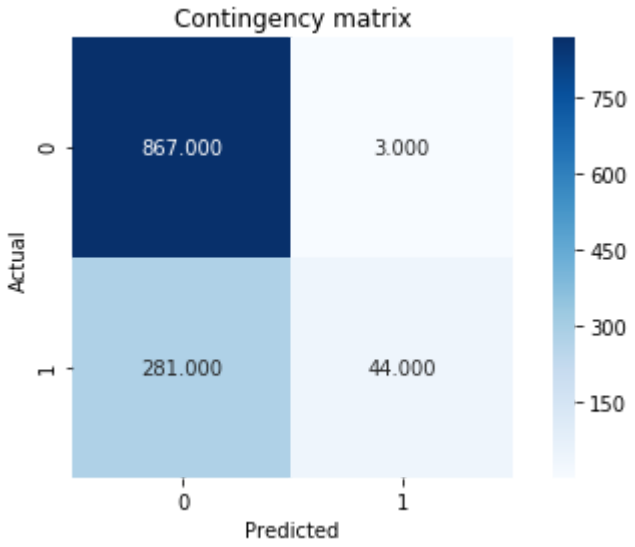


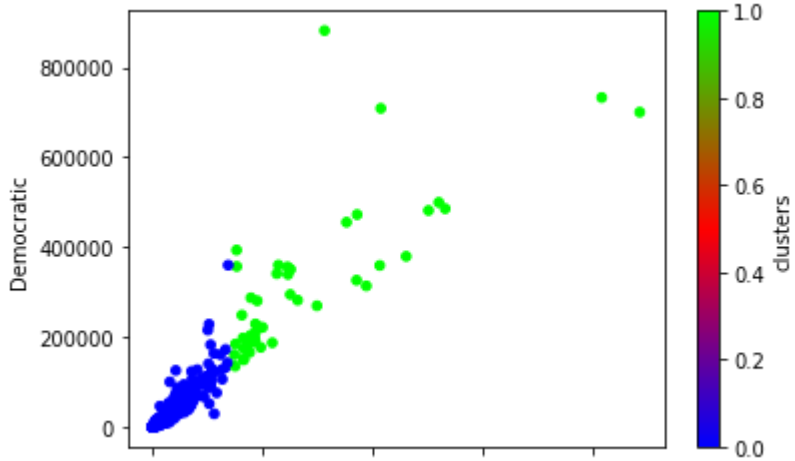


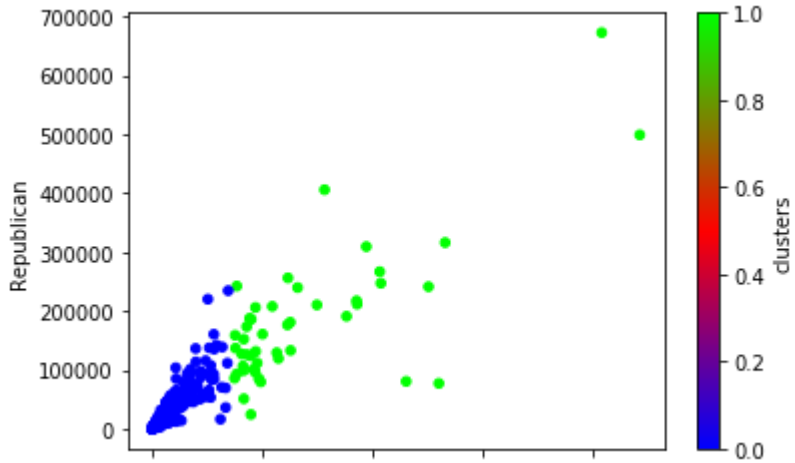


In [19]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state
= 0).fit(X_scaled)
clusters = clustering.labels_

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()


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])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.11979747814620154, 0.902954408093495]
```





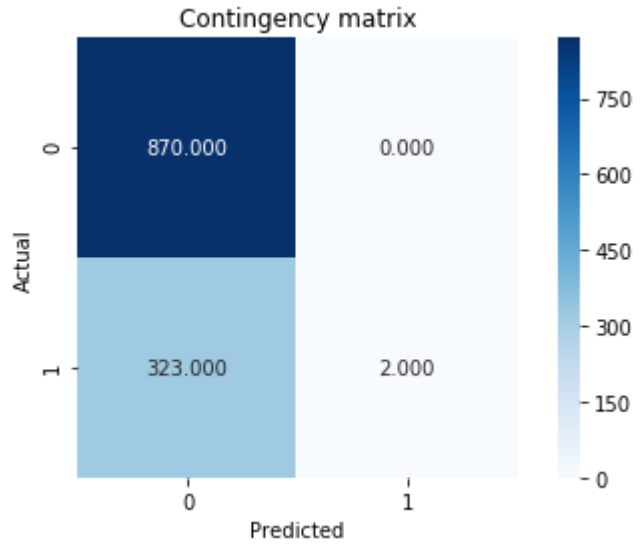


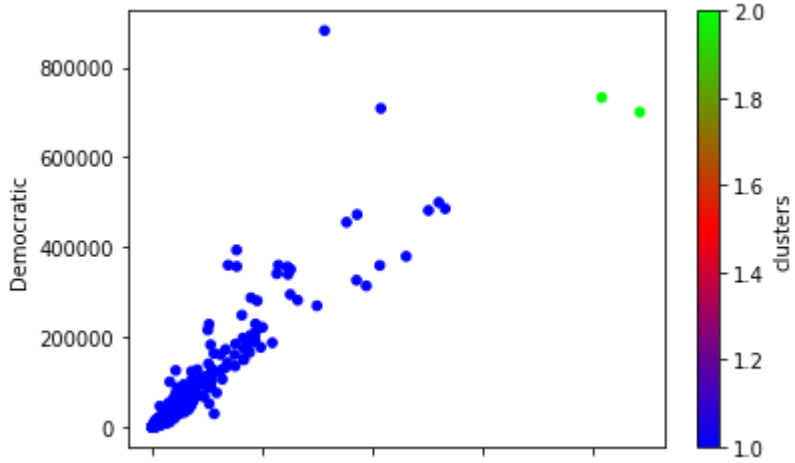
Task 05 (of 07): Build a Clustering model with variable commbination 2

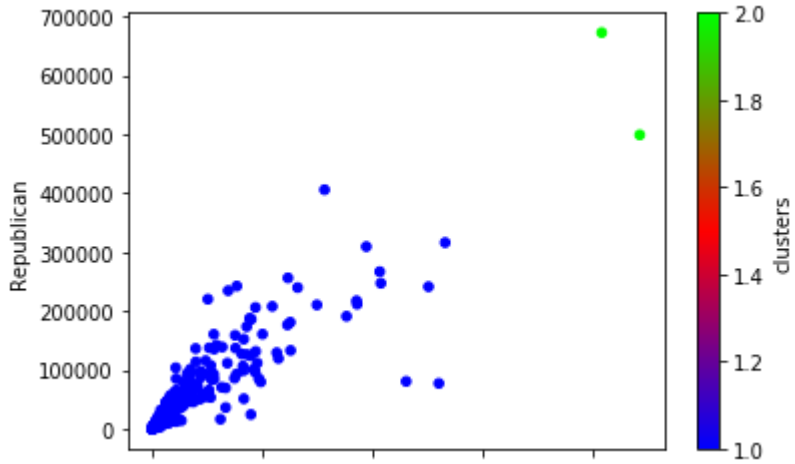
In [21]:
```
#multiple parameter combination 2
x_test=data[['Total Population','Percent White, not Hispanic or Latino',
                                                    'Median Household In
come','Percent Unemployed',
                                                    'Percent Less than H
igh School Degree',"Percent Less than Bachelor's Degree",
                                                    'Percent Rural']]
y_test =  data['Party']
```

In [22]:
```
clustering = linkage(x_test, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters.shape

cont_matrix = metrics.cluster.contingency_matrix(y_test,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()

adjusted_rand_index = metrics.adjusted_rand_score(y_test, clusters)
silhouette_coefficient = metrics.silhouette_score(x_test, clusters, metric =
"euclidean")
print([adjusted_rand_index, silhouette_coefficient])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.005608925119335567, 0.9523736778861548]
```

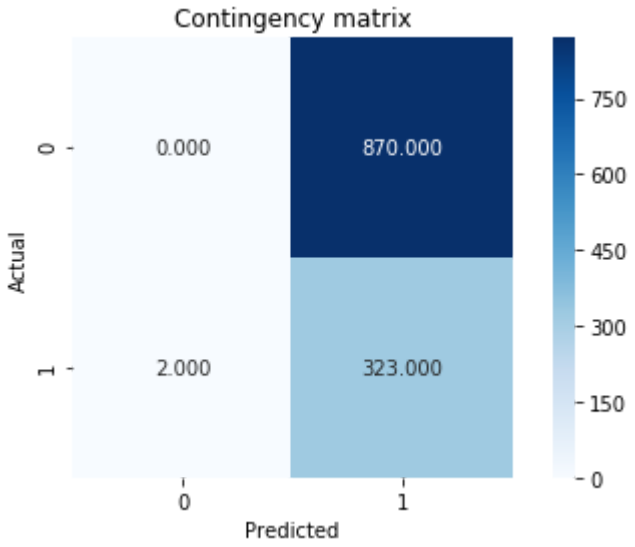


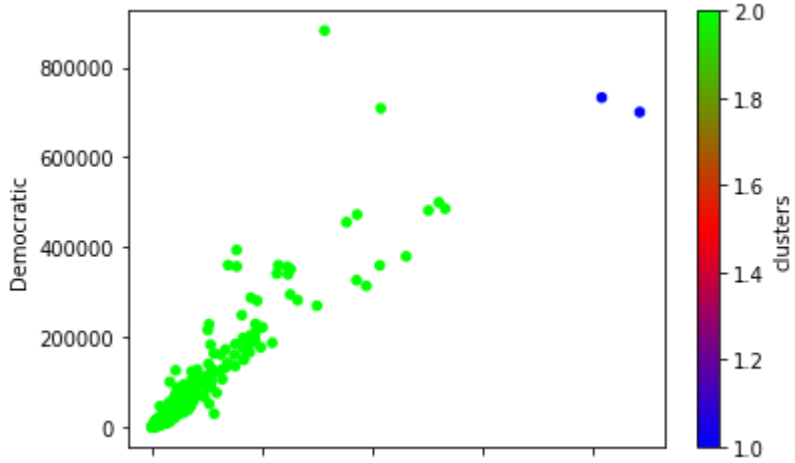


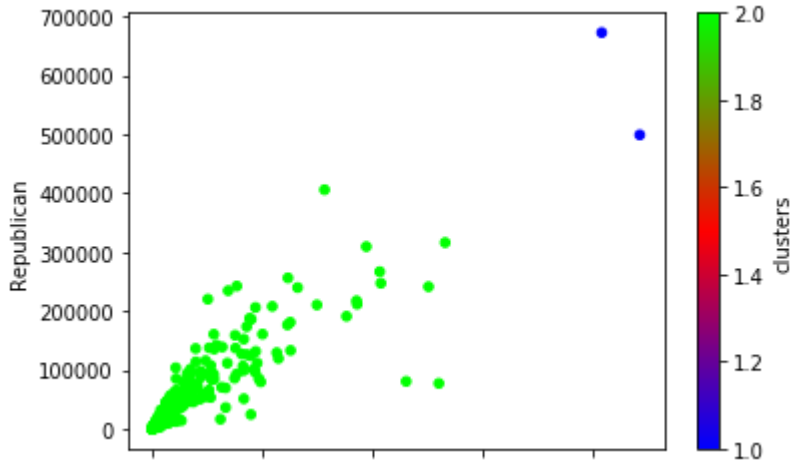


In [23]:
```
clustering = linkage(x_test, method = "complete", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters.shape

cont_matrix = metrics.cluster.contingency_matrix(y_test,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()

adjusted_rand_index = metrics.adjusted_rand_score(y_test, clusters)
silhouette_coefficient = metrics.silhouette_score(x_test, clusters, metric =
"euclidean")
print([adjusted_rand_index, silhouette_coefficient])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.005608925119335567, 0.9523736778861548]
```

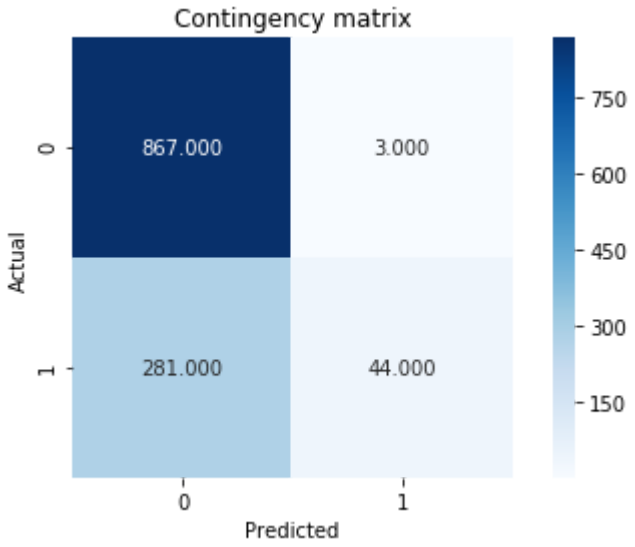


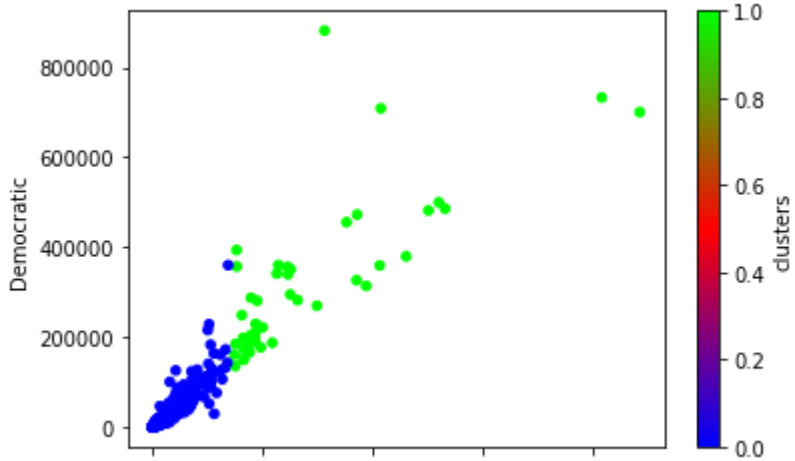


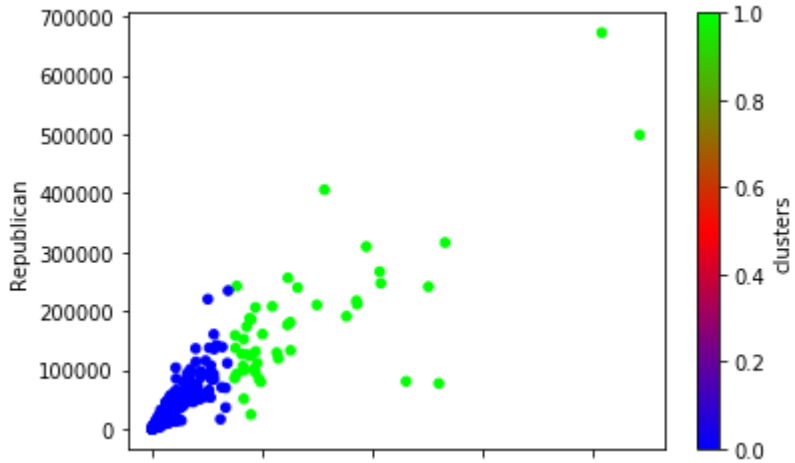


In [24]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state
= 0).fit(X_scaled)
clusters = clustering.labels_

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()


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])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.11979747814620154, 0.902954408093495]
```





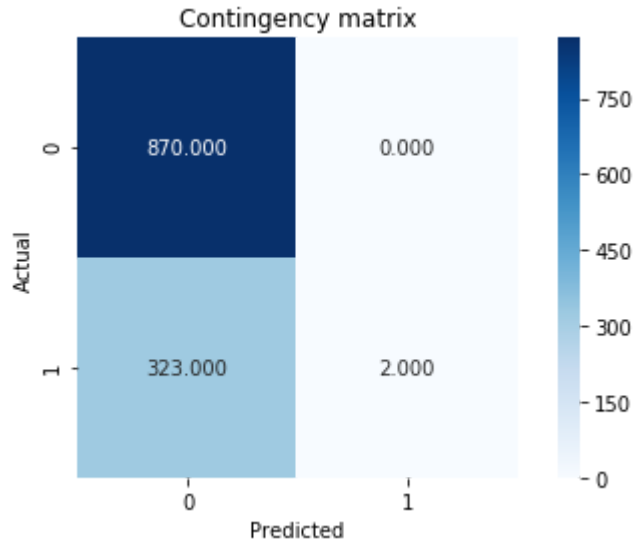


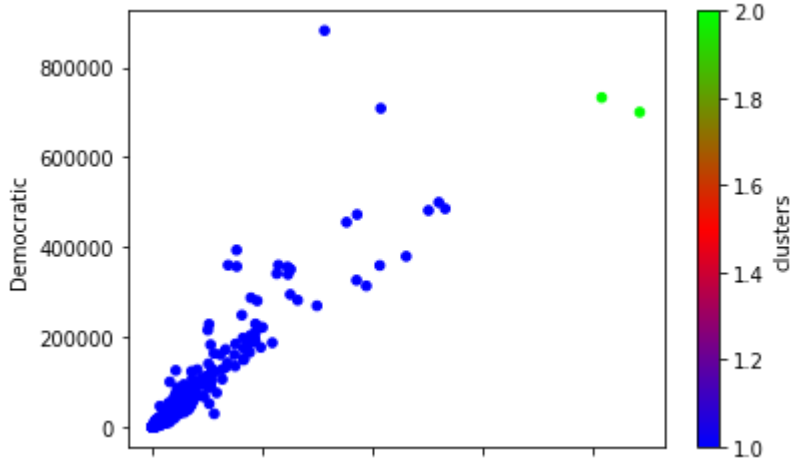
Task 05 (of 07): Build a Clustering model with variable commbination 3

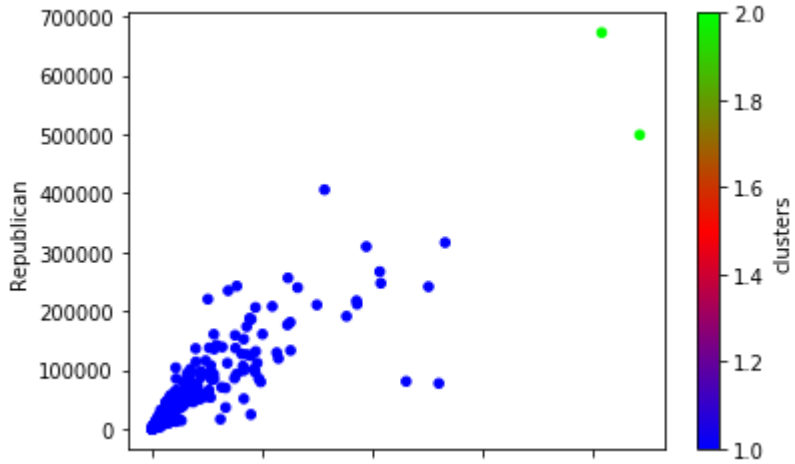
In [26]:
```
#multpile combination 3

x_test=data[['Total Population','Percent White, not Hispanic or Latino',
                                                    'Percent Black, not
 Hispanic or Latino','Percent Hispanic or Latino',
                                                    'Percent Foreign Bor
n','Percent Female','Percent Age 29 and Under',
                                                    'Percent Age 65 and
 Older','Median Household Income','Percent Unemployed',
                                                    'Percent Less than H
igh School Degree',"Percent Less than Bachelor's Degree",
                                                    'Percent Rural']]
y_test =  data['Party']
```

In [27]:
```
clustering = linkage(x_test, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters.shape

cont_matrix = metrics.cluster.contingency_matrix(y_test,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()

adjusted_rand_index = metrics.adjusted_rand_score(y_test, clusters)
silhouette_coefficient = metrics.silhouette_score(x_test, clusters, metric =
"euclidean")
print([adjusted_rand_index, silhouette_coefficient])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.005608925119335567, 0.9523736738921391]
```

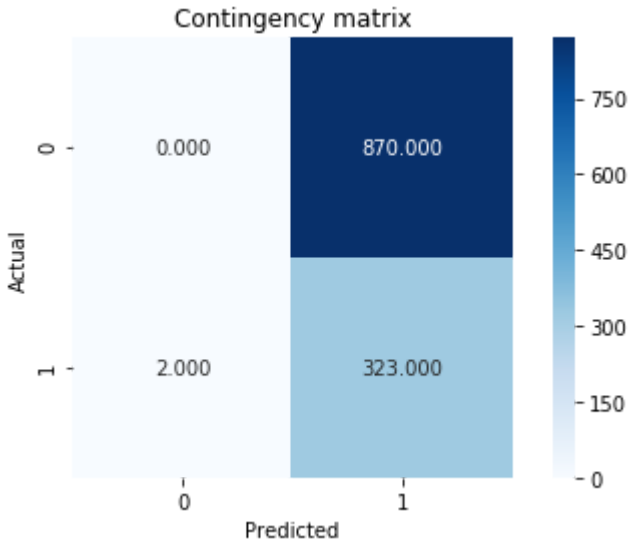


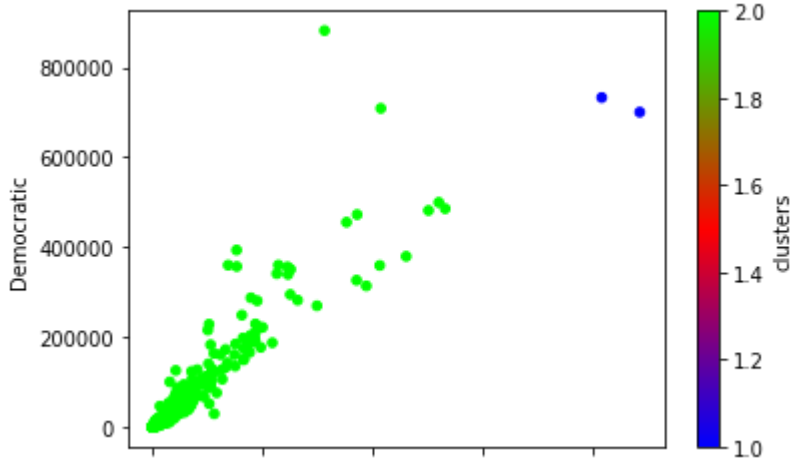


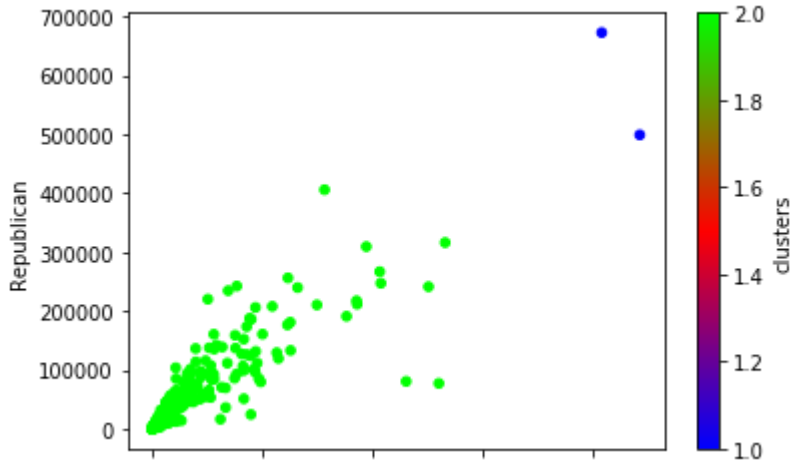


In [28]:
```
clustering = linkage(x_test, method = "complete", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
clusters.shape

cont_matrix = metrics.cluster.contingency_matrix(y_test,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()

adjusted_rand_index = metrics.adjusted_rand_score(y_test, clusters)
silhouette_coefficient = metrics.silhouette_score(x_test, clusters, metric =
"euclidean")
print([adjusted_rand_index, silhouette_coefficient])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.005608925119335567, 0.9523736738921391]
```

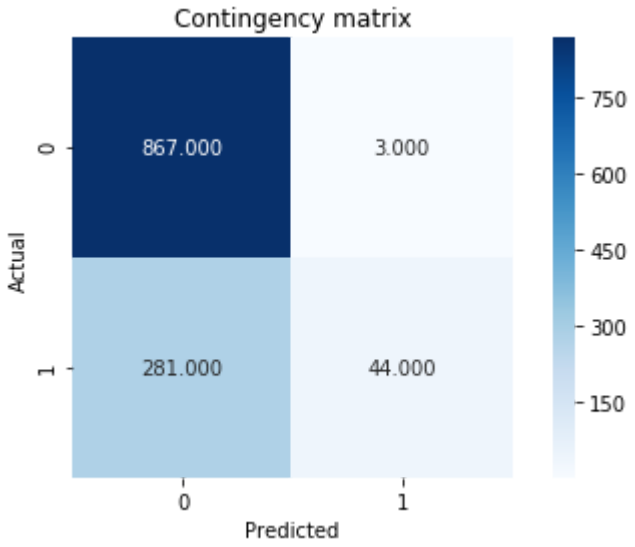


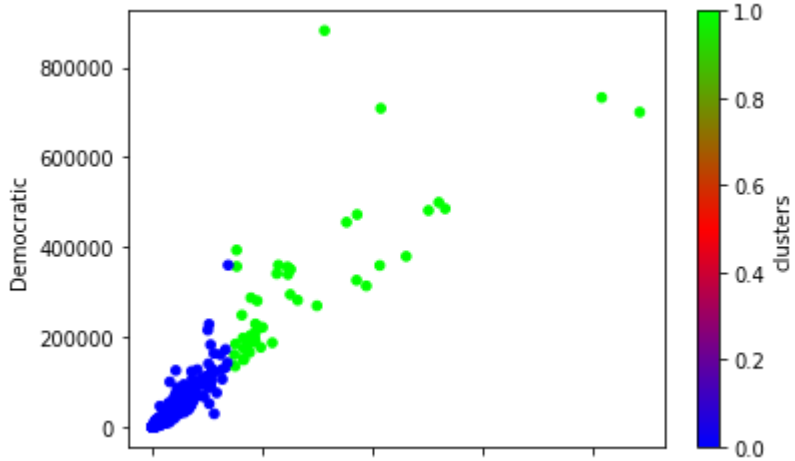


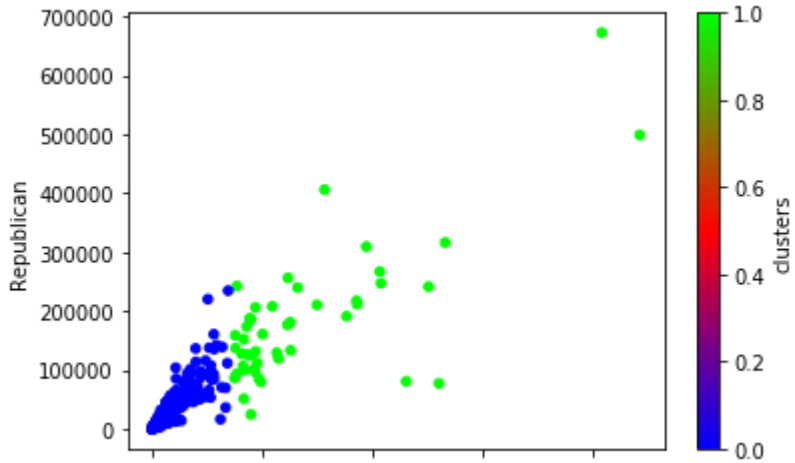


In [29]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state
= 0).fit(X_scaled)
clusters = clustering.labels_

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()


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])


data['clusters'] = clusters
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Democratic', c =
'clusters', colormap = plt.cm.brg)
ax = data.plot(kind = 'scatter', x = 'Total Population', y = 'Republican', c =
'clusters', colormap = plt.cm.brg)
```

```
[0.11979747814620154, 0.902954408093495]
```





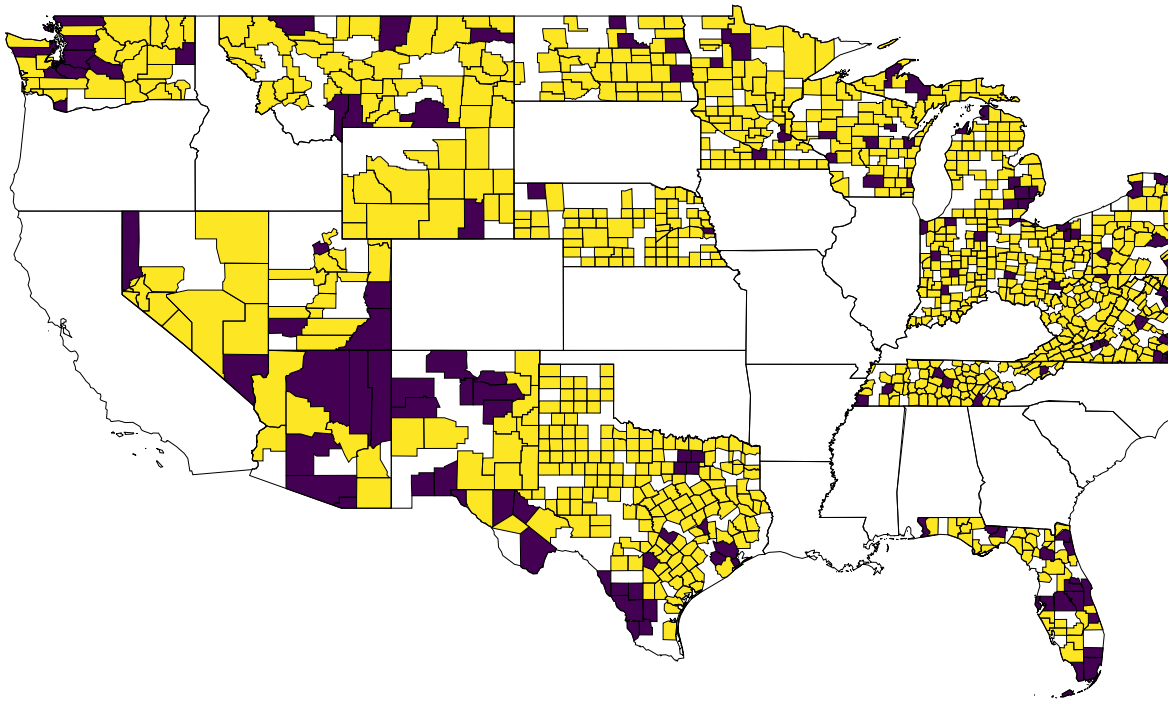


Task 06 (of 07): Map to represent Democratic counties and Republican counties

In [31]:
```
import plotly.figure_factory as ff

x_train = data[['Total Population','Percent White, not Hispanic or Latino',
                'Percent Black, not Hispanic or Latino','Percent Hispanic or L
atino',
                'Percent Foreign Born','Percent Female','Percent Age 29 and Un
der',
                'Percent Age 65 and Older','Median Household Income','Percent
 Unemployed',
                'Percent Less than High School Degree',"Percent Less than Bach
elor's Degree",
                'Percent Rural']]
y_train= data['Party']

x_test = data[['Total Population','Percent White, not Hispanic or Latino',
                'Percent Black, not Hispanic or Latino','Percent Hispanic or L
atino',
                'Percent Foreign Born','Percent Female','Percent Age 29 and Un
der',
                'Percent Age 65 and Older','Median Household Income','Percent
 Unemployed',
                'Percent Less than High School Degree',"Percent Less than Bach
elor's Degree",
                'Percent Rural']]
scaler = StandardScaler()
scaler.fit(x_train)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)

classifier = SVC(kernel = 'rbf')
classifier.fit(x_train_scaled, y_train)
y_pred = classifier.predict(x_test_scaled)

dem_fips = data['FIPS']
y_pred=pd.DataFrame(data=y_pred[0:])
values= y_pred[0].apply(lambda x : "Democrat" if x == 1  else "Republic")
fig = ff.create_choropleth(fips=dem_fips ,values=values, title_text = 'Map of
 Democratic counties and Republican counties',
                           legend_title='Party',state_outline={'color': 'rgb
(0,0,0)', 'width': 0.1},
                           county_outline={'color': 'rgb(0,0,0)', 'width': 0.1
})
fig.layout.template = None
fig.show()
```

```
C:\Users\Preethi\Anaconda3\lib\site-packages\pandas\core\frame.py:6692: Futur
eWarning:

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'.
```

Map of Democratic counties and Republican



Task 07 (of 07): Predict Total votes for democratic and republican in each county and party classification

In [32]:
```
# TASK 7

predicteddemocratic = FINAL_REG_model.predict(test[['Total Population', 'Perce
nt White, not Hispanic or Latino', 'Percent Black, not Hispanic or Latino', 'P
ercent Hispanic or Latino',
                                    'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']])
predictedrepublic = FINAL_REG_repub_model.predict(test[['Total Population', 'P
ercent White, not Hispanic or Latino', 'Percent Black, not Hispanic or Latino'
, 'Percent Hispanic or Latino',
                                    'Percent Foreign Born','Percent Female','Per
cent Age 29 and Under', 'Median Household Income']])
x_test=test[['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 H
igh School Degree',"Percent Less than Bachelor's Degree",
                                                        'Percent Rural']]
scaler = StandardScaler()
scaler.fit(x_test)
x_test= scaler.transform(x_test)

y_pred = FINAL_classifier.predict(x_test)
# y_pred
```

In [33]:
```
import csv

with open('output.csv', mode='w') as csv_file:
    fieldnames = ['State', 'County', 'Democratic', 'Republican', 'Party']
    writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
    writer.writeheader()

    with open('demographics_test.csv', 'r') as readFile:
        reader = csv.reader(readFile)
        next(reader)
        for i,row in enumerate(reader):
            writer.writerow({'State': row[0], 'County': row[1], 'Democratic':
int(round(predicteddemocratic[i])),'Republican': int(round(predictedrepublic[i
])),'Party':int(round(y_pred[i]))})
```