In [1]:
```
# Load libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
from sklearn import linear_model
from sklearn.model_selection import train_test_split, KFold, cross_val_s
core
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.cluster import KMeans, DBSCAN
from sklearn import metrics
```

In [2]:
```
# Load dataset
data = pd.read_csv('merged_train.csv')
data.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
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]:
```
#1. Partition the merged dataset into a training set and a validation se
t using the holdout method or the cross-validation method.
# Democratic
X_train, X_test, Y_train, Y_test = train_test_split(data[['FIPS', 'Total
Population', 'Percent White, not Hispanic or Latino', 'Percent Black, no
t Hispanic or Latino', 'Percent Hispanic or Latino', 'Percent Foreign Bo
rn', 'Percent Female', 'Percent Age 29 and Under', 'Percent Age 65 and O
lder', 'Median Household Income', 'Percent Unemployed', 'Percent Less th
an High School Degree', "Percent Less than Bachelor's Degree", 'Percent
 Rural', 'Party']], data['Democratic'], test_size = 0.25,train_size = 0.
75,random_state = 0)
X_train, X_vals, Y_train, Y_vals = train_test_split(X_train, Y_train, te
st_size = 0.25, train_size = 0.75,  random_state = 0)

# Republican
X_train2, X_test2, Y_train2, Y_test2 = train_test_split(data[['FIPS', 'T
otal Population', 'Percent White, not Hispanic or Latino', 'Percent Blac
k, not Hispanic or Latino', 'Percent Hispanic or Latino', 'Percent Forei
gn Born', 'Percent Female', 'Percent Age 29 and Under', 'Percent Age 65
 and Older', 'Median Household Income', 'Percent Unemployed', 'Percent L
ess than High School Degree', "Percent Less than Bachelor's Degree", 'Pe
rcent Rural', 'Party']], data['Republican'], test_size = 0.25,train_size
= 0.75,random_state = 0)
X_train2, X_vals2, Y_train2, Y_vals2 = train_test_split(X_train2, Y_trai
n2, test_size = 0.25, train_size = 0.75,  random_state = 0)
```

In [4]:
```
#2. Standardize the training set and the validation set.
# Democratic
scaler = StandardScaler()
scaler.fit(X_train)
x_train_scaled = scaler.transform(X_train)
x_vals_scaled = scaler.transform(X_vals)

# Republican
scaler2 = StandardScaler()
scaler2.fit(X_train2)
x_train_scaled2 = scaler2.transform(X_train2)
x_vals_scaled2 = scaler2.transform(X_vals2)
```

In [5]:
```
#3. Build a linear regression model to predict the number of votes cast
 for the Democratic party in each county.
model = linear_model.LinearRegression().fit(X = x_train_scaled[:, [1, 7,
10]], y = Y_train)

# Compute evaluation metrics for the validation set and report your resu
lts.
Rsqr_val = model.score(X = x_vals_scaled[:, [1, 7, 10]], y = Y_vals)
print(Rsqr_val)
```

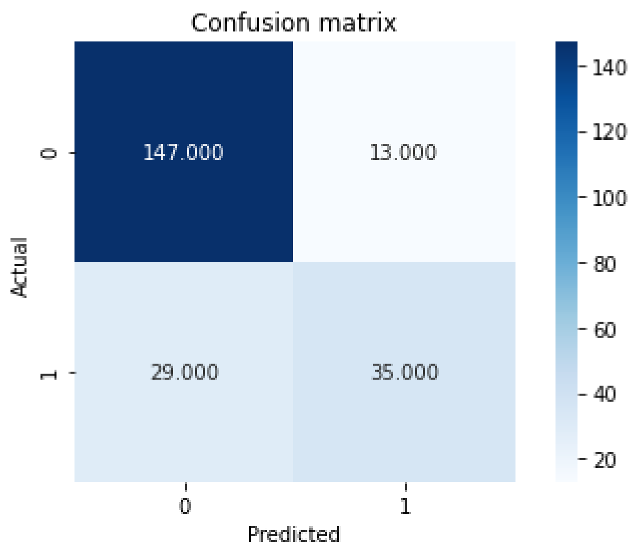
```
0.9513600795496906
```

In [6]:
```
#3. Build a linear regression model to predict the number of votes cast
 for the Republican party in each county.
model2 = linear_model.LinearRegression().fit(X = x_train_scaled2[:, [1,
2, 6, 7, 9, 10, 12, 13]], y = Y_train2)

# Compute evaluation metrics for the validation set and report your resu
lts.
Rsqr_val2 = model2.score(X = x_vals_scaled2[:, [1, 2, 6, 7, 9, 10, 12, 1
3]], y = Y_vals2)
print(Rsqr_val2)
```

```
0.5156182047170289
```

In [7]:
```
#4. Party (Partition)
X_train3, X_test3, Y_train3, Y_test3 = train_test_split(data[['Total Pop
ulation', 'Percent White, not Hispanic or Latino', 'Percent Black, not H
ispanic or Latino', 'Percent Hispanic or Latino', 'Percent Unemployed',
'Percent Less than High School Degree', 'Percent Rural']], data['Party'
], test_size = 0.25,train_size = 0.75,random_state = 0)
X_train3, X_vals3, Y_train3, Y_vals3 = train_test_split(X_train3, Y_trai
n3, test_size = 0.25, train_size = 0.75,  random_state = 0)

#4. Party (Standardize)
scaler3 = StandardScaler()
scaler3.fit(X_train3)
x_train_scaled3 = scaler3.transform(X_train3)
x_vals_scaled3 = scaler3.transform(X_vals3)
```

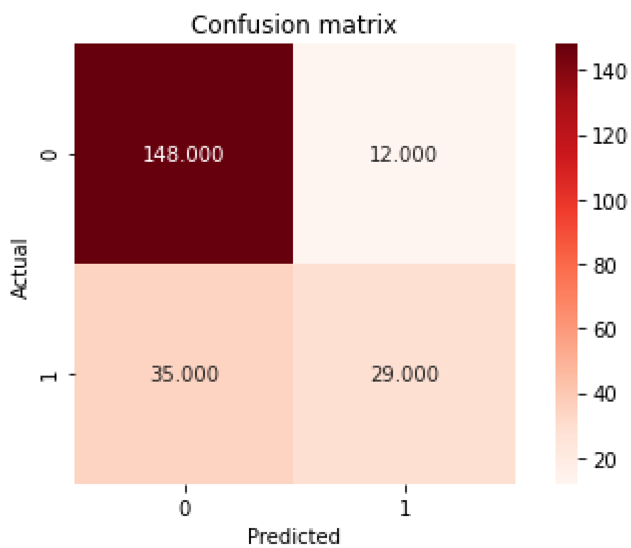
In [8]:
```
#4. Build a classification model (Decision Trees) to classify each count
y as Democratic or Republican.
classifier = DecisionTreeClassifier(criterion = "entropy", random_state
= 0)
classifier.fit(x_train_scaled3, Y_train3)

y_pred = classifier.predict(x_vals_scaled3)

conf_matrix = metrics.confusion_matrix(Y_vals3, 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 [9]:
```
#4. Compute evaluation metrics for the validation set and report your re
sults.
accuracy = metrics.accuracy_score(Y_vals3, y_pred)
error = 1 - metrics.accuracy_score(Y_vals3, y_pred)
precision = metrics.precision_score(Y_vals3, y_pred, average = None)
recall = metrics.recall_score(Y_vals3, y_pred, average = None)
F1_score = metrics.f1_score(Y_vals3, y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8125, 0.1875, array([0.83522727, 0.72916667]), array([0.91875 , 0.54
6875]), array([0.875, 0.625])]
```

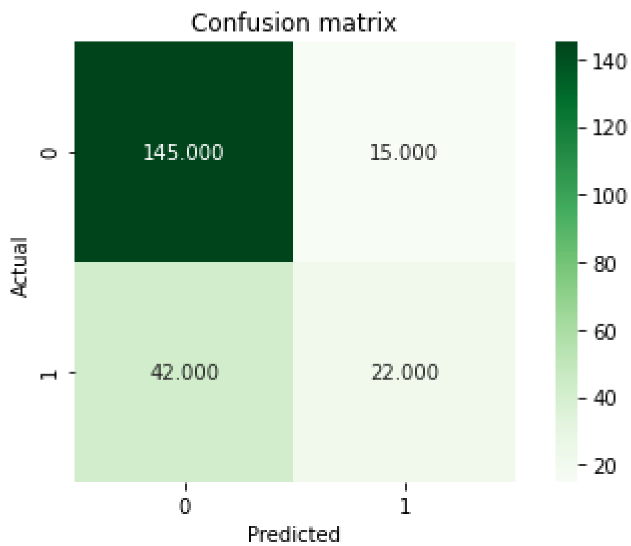
In [10]:
```
#4. Build a classification model (k-Nearest Neighbors) to classify each
 county as Democratic or Republican.
classifier = KNeighborsClassifier(n_neighbors = 3)
classifier.fit(x_train_scaled3, Y_train3)

y_pred = classifier.predict(x_vals_scaled3)

conf_matrix = metrics.confusion_matrix(Y_vals3, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap
= plt.cm.Reds)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



In [11]:
```
#4. Compute evaluation metrics for the validation set and report your re
sults.
accuracy = metrics.accuracy_score(Y_vals3, y_pred)
error = 1 - metrics.accuracy_score(Y_vals3, y_pred)
precision = metrics.precision_score(Y_vals3, y_pred, average = None)
recall = metrics.recall_score(Y_vals3, y_pred, average = None)
F1_score = metrics.f1_score(Y_vals3, y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7901785714285714, 0.2098214285714286, array([0.80874317, 0.7073170
7]), array([0.925    , 0.453125]), array([0.86297376, 0.55238095])]
```

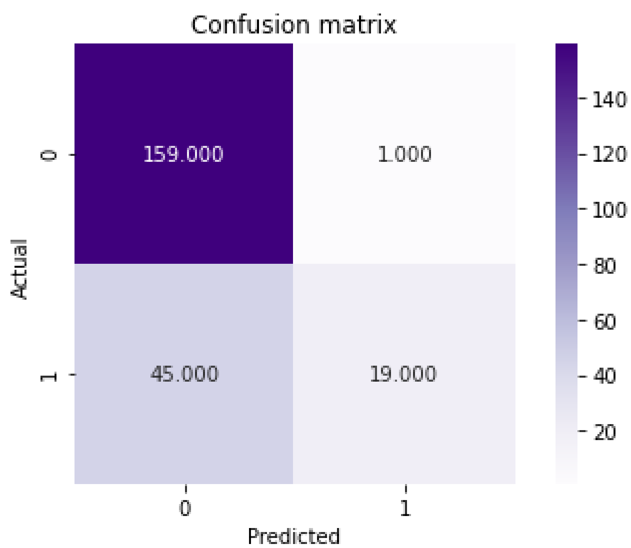
In [12]:
```
#4. Build a classification model (Naive Bayes) to classify each county a
s Democratic or Republican.
classifier = GaussianNB()
classifier.fit(x_train_scaled3, Y_train3)

y_pred = classifier.predict(x_vals_scaled3)

conf_matrix = metrics.confusion_matrix(Y_vals3, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap
= plt.cm.Greens)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```



In [13]:
```
#4. Compute evaluation metrics for the validation set and report your re
sults.
accuracy = metrics.accuracy_score(Y_vals3, y_pred)
error = 1 - metrics.accuracy_score(Y_vals3, y_pred)
precision = metrics.precision_score(Y_vals3, y_pred, average = None)
recall = metrics.recall_score(Y_vals3, y_pred, average = None)
F1_score = metrics.f1_score(Y_vals3, y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7455357142857143, 0.2544642857142857, array([0.77540107, 0.5945945
9]), array([0.90625, 0.34375]), array([0.83573487, 0.43564356])]
```

In [14]:
```
#4. Build a classification model (Support Vector Machines) to classify e
ach county as Democratic or Republican.
classifier = SVC(kernel = "rbf")
classifier.fit(x_train_scaled3, Y_train3)

y_pred = classifier.predict(x_vals_scaled3)

conf_matrix = metrics.confusion_matrix(Y_vals3, y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap
= plt.cm.Purples)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
```

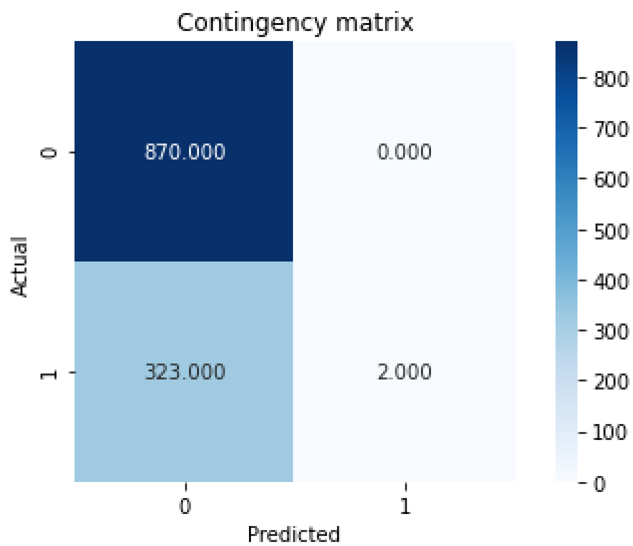


In [15]:
```
#4. Compute evaluation metrics for the validation set and report your re
sults.
accuracy = metrics.accuracy_score(Y_vals3, y_pred)
error = 1 - metrics.accuracy_score(Y_vals3, y_pred)
precision = metrics.precision_score(Y_vals3, y_pred, average = None)
recall = metrics.recall_score(Y_vals3, y_pred, average = None)
F1_score = metrics.f1_score(Y_vals3, y_pred, average = None)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7946428571428571, 0.2053571428571429, array([0.77941176, 0.95
]), array([0.99375 , 0.296875]), array([0.87362637, 0.45238095])]
```

In [16]:
```
#5. Party (Partition)
X = data[['Total Population']]
Y = data['Party']

#5. Party (Standardize)
scaler5 = StandardScaler()
scaler5.fit(X)
X_scaled = scaler5.transform(X)
```

In [17]:
```
#5. Build a clustering model to cluster the counties (Hierarchical Clust
ering)
clustering = linkage(X_scaled, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = "maxclust")

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



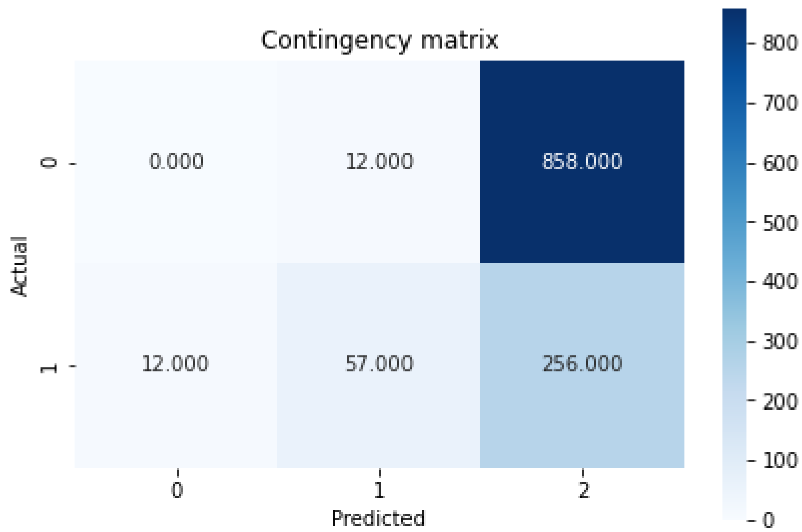
In [18]:
```
#5. Compute unsupervised and supervised evaluation metrics
#   for the validation set with the party of the counties (Democratic or
Republican) as the
#   true cluster and report your results.
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X_scaled, clusters, me
tric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.005608925119335567, 0.9531008389502824]
```

In [19]:
```
#5. Compute evaluation metrics for the true clusters of the data (democr
at/republican).
silhouette_coefficient = metrics.silhouette_score(X_scaled, Y, metric =
"euclidean")
print(silhouette_coefficient)
```

```
0.4204042955856235
```

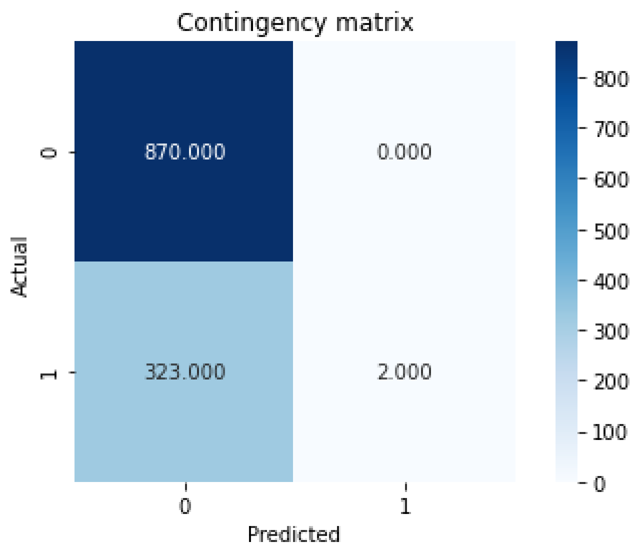
In [20]:
```
#5 K-Means Clustering
clustering2 = KMeans(n_clusters = 3, init = 'random', n_init = 1, random
_state = 2).fit(X_scaled)
clusters2 = clustering2.labels_

# Plot contingency matrix
cont_matrix2 = metrics.cluster.contingency_matrix(Y, clusters2)
sns.heatmap(cont_matrix2, annot = True, fmt = ".3f", square = True, cmap
= plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
```



In [21]:
```
#5. Compute unsupervised and supervised evaluation metrics
#   for the validation set with the party of the counties (Democratic or
Republican) as the
#   true cluster and report your results.
adjusted_rand_index2 = metrics.adjusted_rand_score(Y, clusters2)
silhouette_coefficient2 = metrics.silhouette_score(X_scaled, clusters2,
metric = "euclidean")
print([adjusted_rand_index2, silhouette_coefficient2])
```

```
[0.1745654800126557, 0.8700304559768228]
```

In [22]:
```
#5 DBSCAN Clustering
clustering3 = DBSCAN(eps = 3, min_samples = 5, metric = "euclidean").fit
(X_scaled)
clusters3 = clustering3.labels_

# Plot contingency matrix
cont_matrix3 = metrics.cluster.contingency_matrix(Y, clusters3)
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()
```

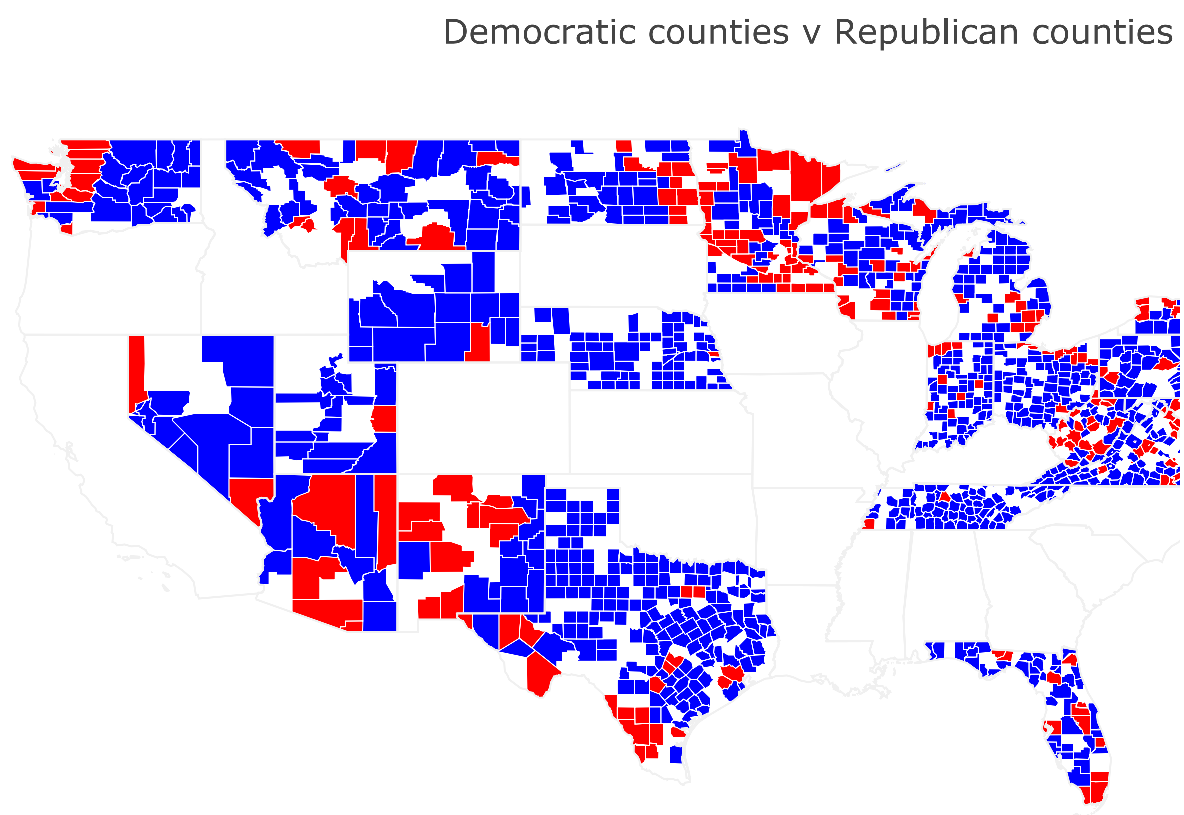


In [23]:
```
#5. Compute unsupervised and supervised evaluation metrics
#   for the validation set with the party of the counties (Democratic or
Republican) as the
#   true cluster and report your results.
adjusted_rand_index3 = metrics.adjusted_rand_score(Y, clusters3)
silhouette_coefficient3 = metrics.silhouette_score(X_scaled, clusters3,
metric = "euclidean")
print([adjusted_rand_index3, silhouette_coefficient3])
```

```
[0.005608925119335567, 0.9531008389502824]
```

In [24]:
```
#6. Create a map of Democratic counties and Republican counties using th
e counties' FIPS codes and Python's Plotly library (plot.ly/python/count
y-choropleth/).
# [OLD]
import plotly.figure_factory as ff
fips = data['FIPS'].tolist()
values = data['Party'].tolist()
colorscale = [
    'rgb(0, 0, 255)',
    'rgb(255, 0, 0)',
]
fig = ff.create_choropleth(fips=fips, values=values, colorscale=colorsca
le,
      county_outline={'color': 'rgb(255,255,255)', 'width': 0.5}, legend
_title='Party by County',
      title='Democratic counties v Republican counties [OLD]')
fig.layout.template = None
```

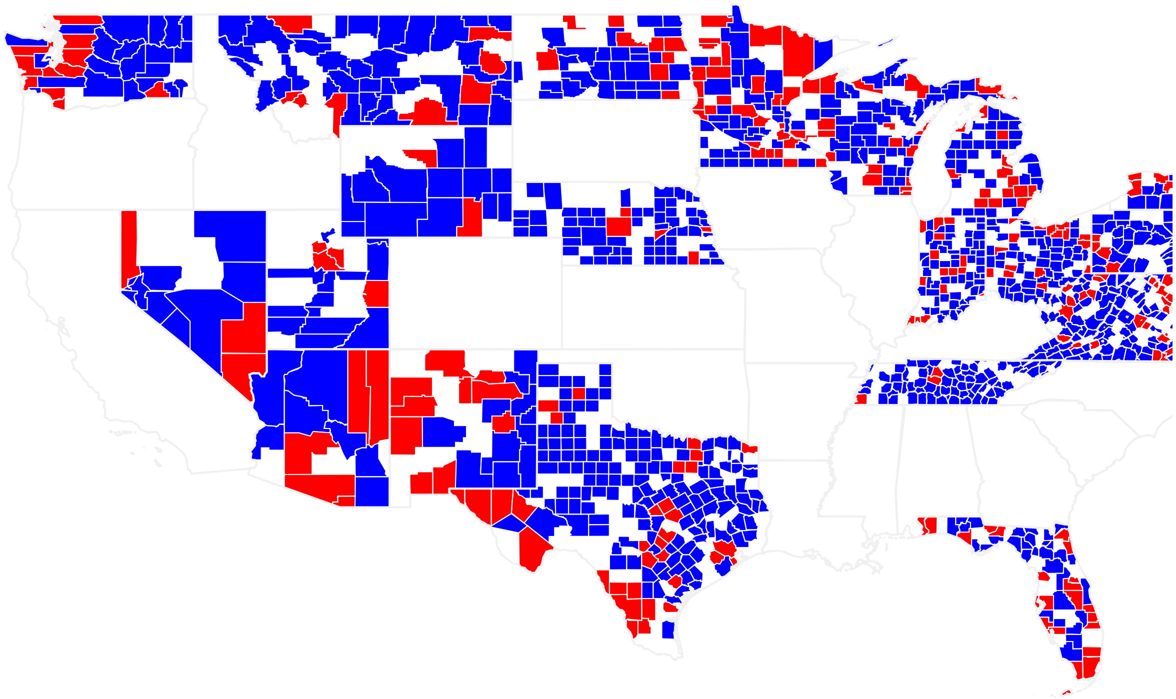
In [25]:
```
#6. Show map [OLD]
fig.show()
```



In [26]:
```
#6. Create a map of Democratic counties and Republican counties using th
e counties' FIPS codes and Python's Plotly library (plot.ly/python/count
y-choropleth/).
# [NEW]
classifier = DecisionTreeClassifier(criterion = "entropy", random_state
= 0)
classifier.fit(x_train_scaled3, Y_train3)

#6. Standardize
X = data[['Total Population', 'Percent White, not Hispanic or Latino',
'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino',
'Percent Unemployed', 'Percent Less than High School Degree', 'Percent R
ural']]
scaler = StandardScaler()
scaler.fit(X)
x_scaled = scaler.transform(X)

party_pred = classifier.predict(x_scaled)

# import plotly.figure_factory as ff
fips = data['FIPS'].tolist()
values = party_pred.tolist()
colorscale = [
    'rgb(0, 0, 255)',
    'rgb(255, 0, 0)',
]
fig = ff.create_choropleth(fips=fips, values=values, colorscale=colorsca
le,
      county_outline={'color': 'rgb(255,255,255)', 'width': 0.5}, legend
_title='Party by County',
      title='Democratic counties v Republican counties [NEW]')
fig.layout.template = None
```

In [27]:
```
#6. Show map [NEW]
fig.show()
```

Democratic counties v Republican counties



In [28]:
```
#7. Load dataset
data2 = pd.read_csv('demographics_test.csv')
data2.head()
```

Out[28]:

	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 [29]:
```
#7. Standardize
# Democratic
X = data2[['Total Population', 'Percent Age 29 and Under', 'Percent Unem
ployed']]
scaler = StandardScaler()
scaler.fit(X)
x_scaled = scaler.transform(X)

# Republican
Y = data2[['Total Population', 'Percent White, not Hispanic or Latino',
'Percent Female', 'Percent Age 29 and Under', 'Median Household Income',
'Percent Unemployed', "Percent Less than Bachelor's Degree", 'Percent Ru
ral']]
scaler = StandardScaler()
scaler.fit(Y)
y_scaled = scaler.transform(Y)

# Party
Z = data2[['Total Population', 'Percent White, not Hispanic or Latino',
'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino',
'Percent Unemployed', 'Percent Less than High School Degree', 'Percent R
ural']]
scaler = StandardScaler()
scaler.fit(Z)
z_scaled = scaler.transform(Z)
```

In [30]:
```
#7. Use your best performing regression and classification models to pre
dict the
#   number of votes cast for the Democratic party in each county for the
test dataset (demographics_test.csv).
import numpy as np
def convert(x):
    y = int(round(x))
    if (y > 0):
        return y
    else:
        return 0

democratic = model.predict(X = x_scaled)
d = np.array(democratic)
data2['Democratic'] = np.array([convert(xi) for xi in d])

#7. Use your best performing regression and classification models to pre
dict the number of votes cast
#   for the Republican party in each county for the test dataset (demogr
aphics_test.csv).
republican = model2.predict(X = y_scaled)
r = np.array(republican)
data2['Republican'] = np.array([convert(xi) for xi in r])

#7. Use your best performing regression and classification models to pre
dict the party (Democratic or Republican) of
#   each county for the test dataset (demographics_test.csv).
classifier = DecisionTreeClassifier(criterion = "entropy", random_state
= 0)
classifier.fit(x_train_scaled3, Y_train3)
party = classifier.predict(z_scaled)
data2['Party'] = party
data2.head(10)
```

Out[30]:

	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	I
0	NV	eureka	32011	1730	98.265896	0.057803	0.462428	0.346821	51.156069	27.
1	TX	zavala	48507	12107	5.798299	0.594697	93.326175	9.193029	49.723301	49.
2	VA	king george	51099	25260	73.804434	16.722090	4.441805	2.505938	50.166271	40.
3	OH	hamilton	39061	805965	66.354867	25.654340	2.890944	5.086945	51.870615	40.
4	TX	austin	48015	29107	63.809393	8.479060	25.502456	9.946061	50.671660	37.
5	MI	barry	26015	59316	94.832760	0.475420	2.576033	1.459977	49.686425	35.
6	NM	valencia	35061	75993	34.159725	1.027726	59.655495	8.202071	49.761162	39.
7	TX	ellis	48139	160225	63.367140	9.053518	25.048526	8.468716	50.679357	42.
8	NJ	mercer	34021	371101	51.655749	19.702992	16.432723	21.813738	51.053217	39.
9	PA	cambria	42021	137762	93.053963	3.179396	1.525820	1.169408	50.736052	33.

In [31]:

```
#7. Q.E.D.
# data2[['State', 'County', 'Democratic', 'Republican', 'Party']].to_csv
(r'/Users/Polvyer/Desktop/CS-418-Project2/project_02/output.csv', index
 = False, header=True)
```

In [ ]:

```
```