Project 02: Regression, Classification, and Clustering

Libraries

In [1]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn import linear_model
from sklearn import metrics
from sklearn.metrics import mean_squared_error
import math


import seaborn as sns
from matplotlib import pyplot as plt
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
```

Dataset

In [2]:
```
#dataset
election_data = pd.read_csv('merged_train.csv')
election_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	Percent Age 29 and Under	Percent Age 65 and Older	Median Household Income	Percent Unemployed	Pe Less S D
0	AZ	apache	4001	72346	18.571863	0.486551	5.947806	1.719515	50.598513	45.854643	13.322091	32460	15.807433	21.7
1	AZ	cochise	4003	128177	56.299492	3.714395	34.403208	11.458374	49.069646	37.902276	19.756275	45383	8.567108	13.4
2	AZ	coconino	4005	138064	54.619597	1.342855	13.711033	4.825298	50.581614	48.946141	10.873943	51106	8.238305	11.0
3	AZ	gila	4007	53179	63.222325	0.552850	18.548675	4.249798	50.296170	32.238290	26.397638	40593	12.129932	15.7
4	AZ	graham	4009	37529	51.461536	1.811932	32.097844	4.385942	46.313518	46.393456	12.315809	47422	14.424104	14.5

Task 1

Partition the merged dataset into a training set and a validation set using the holdout method or the cross-validation method. How did you partition the dataset?

Answer: We are partitioning the dataset such that 75% of the observation are held out for training and the rest for validation.

In [3]:
```
X_train, X_val, y_train, y_val = train_test_split(election_data.iloc[:,3:18], election_data['Party'], tes
t_size = 0.25, random_state = 0)
# print(X_train)
```

Task 2

Standardize the training set and the validation set

In [4]:
```
# variables required for training
# x_train = X_train.select_dtypes(include=[np.int64,np.float64])
x_train = X_train.iloc[:,:-2]

# variables required for validation
# x_val = X_val.select_dtypes(include=[np.int64,np.float64])
x_val = X_val.iloc[:,:-2]

# Standardize the training 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_df = pd.DataFrame(x_train_scaled,index = x_train.index,columns=x_train.columns)
x_val_scaled_df = pd.DataFrame(x_val_scaled,index = x_val.index,columns=x_val.columns)
```

Task 3

Build a classification model to classify each county as Democratic or Republican. Consider at least two different classification techniques with multiple combinations of parameters and multiple combinations of variables. Compute evaluation metrics for the validation set and report your results.

3.1: Regression model for Democratic values

3.1.1: Using all predictor variables

In [5]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled_df, y = X_train['Democratic'])
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 [6]:
```
#predictor variable
y_pred = fitted_model.predict(x_val_scaled)
# print(y_pred)
```

In [7]:
```
# Evaluation Metrics
n, p = x_train_scaled.shape

# Generatiion of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Democratic'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Democratic']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.9338361960241587
Adjusted R squared value: 0.9328609925641973
RMSE value: 14771.994793075706
```

3.1.2: Using the following predictor variables:

- Total Population
- Percent White, not Hispanic or Latino
- Percent Black, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Foreign Born'

In [8]:
```
selected_predictor_variables = ['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born']
```

In [9]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled_df[selected_predictor_variables], y = X_train['Democratic'])
print(fitted_model.coef_)
```

```
[ 70705.8786866   -2212.85847901   -131.80192434 -10178.54695173
   9916.88242758]
```

In [10]:
```
#predictor variable
y_pred = fitted_model.predict(x_val_scaled_df[selected_predictor_variables])
# y_pred
```

In [11]:
```
# Evaluation Metrics
n = x_train_scaled.shape[0]
p = len(selected_predictor_variables)
# print(ind-col-1)

# Generatiion of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Democratic'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Democratic']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.9272983198666902
Adjusted R squared value: 0.9268898834614469
RMSE value: 14592.862156527459
```

3.1.3: Using the following predictor variables:

- Total Population
- Percent Black, not Hispanic or Latino
- Percent Less than Bachelor's Degree

In [12]:
```
selected_predictor_variables = ['Total Population',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Less than Bachelor\'s Degree']
```

In [13]:
```
model = linear_model.LinearRegression()
fitted_model_democratic = model.fit(X = x_train_scaled_df[selected_predictor_variables], y = X_train['Dem
ocratic'])
print(fitted_model_democratic.coef_)
```

```
[70692.75301251  1827.68857508 -9335.76053975]
```

In [14]:
```
#predictor variable
y_pred = fitted_model_democratic.predict(x_val_scaled_df[selected_predictor_variables])
# y_pred
```

In [15]:
```
# Evaluation Metrics
n = x_train_scaled.shape[0]
p = len(selected_predictor_variables)
# print(ind-col-1)

# Generatiion of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Democratic'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Democratic']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.950506110643013
Adjusted R squared value: 0.9503396513738752
RMSE value: 12456.892528655851
```

3.1.4: Using LASSO Regression with all variables

In [16]:
```
model = linear_model.Lasso(alpha = 1)
fitted_model = model.fit(X = x_train_scaled_df, y = X_train['Democratic'])
print(fitted_model.coef_)
```

```
[ 69224.71479124  -3195.33996565  -1013.63916087  -6917.77376216
   3975.00309549    192.59502461  -5290.27001162  -1846.83971098
   1471.58775101   1467.72300999   4030.09531822 -10515.05282676
   -155.56176752]
```

In [17]:
```
#predictor variable
y_pred = fitted_model.predict(x_val_scaled_df)
# y_pred
```

In [18]:
```
# Evaluation Metrics
n, p = x_train_scaled.shape

# Generation of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Democratic'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Democratic']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.9338579590814098
Adjusted R squared value: 0.9328830763921335
RMSE value: 14768.885350551016
```

3.2 Regression model for Republican values

3.2.1: Using all predictor variables

In [19]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled_df, y = X_train['Republican'])
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 [20]:
```
#predictor variable
y_pred = fitted_model.predict(x_val_scaled_df)
# y_pred
```

In [21]:
```
# Evaluation Metrics
n, p = x_train_scaled.shape
# print(ind-col-1)

# Generatiion of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Republican'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Republican']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.7239014362949736
Adjusted R squared value: 0.7198319563310673
RMSE value: 15962.431310602105
```

3.2.2: Using the following predictor variables

- Total Population
- Percent White, not Hispanic or Latino
- Percent Black, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Foreign Born

In [22]:
```
selected_predictor_variables = ['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born']
```

In [23]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_scaled_df[selected_predictor_variables], y = X_train['Republican'])
print(fitted_model.coef_)
```

```
[46801.58031155  2411.56062758 -1926.15808714    98.71008908
  -478.25725257]
```

In [24]:
```
#predictor variable
y_pred = fitted_model.predict(x_val_scaled_df[selected_predictor_variables])
# y_pred
```

In [25]:
```
# Evaluation Metrics
n = x_train_scaled.shape[0]
p = len(selected_predictor_variables)

# Generatiion of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Republican'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Republican']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.67042381870624 98
Adjusted R squared value: 0.6685722671259478
RMSE value: 17111.71419341798
```

3.2.3: Using the following predictor variables:

- Total Population
- Percent White, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Foreign Born
- Percent Age 65 and Older
- Percent Unemployed
- Median Household Income
- Percent Rural

In [26]:
```
selected_predictor_variables = ['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born',
                                'Percent Age 65 and Older',
                                'Percent Unemployed',
                                'Median Household Income',
                                'Percent Rural']
```

In [27]:
```
model = linear_model.LinearRegression()
fitted_model_republican = model.fit(X = x_train_scaled_df[selected_predictor_variables], y = X_train['Rep
ublican'])
print(fitted_model_republican.coef_)
```

```
[45133.5738712   4612.72460625  3998.62967731 -4790.68208843
  2692.84982155  2174.86528205  6130.35899569 -5297.8335129 ]
```

In [28]:
```
#predictor variable
y_pred = fitted_model_republican.predict(x_val_scaled_df[selected_predictor_variables])
# y_pred
```

In [29]:
```
# Evaluation Metrics
n = x_train_scaled.shape[0]
p = len(selected_predictor_variables)
# Generatiion of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Republican'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Republican']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.73020806715309998
Adjusted R squared value: 0.7277747689989
RMSE value: 15749.245925443487
```

3.2.4 Using LASSO Regression with all variables

In [30]:
```
# Model-4 LASSO Regression (with variables)
model = linear_model.Lasso(alpha = 1)
fitted_model = model.fit(X = x_train_scaled_df, y = X_train['Republican'])
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 [31]:
```
#predictor variable
y_pred = fitted_model.predict(x_val_scaled_df)
# y_pred
```

In [32]:
```
# Evaluation Metrics
n, p = x_train_scaled.shape
# Generatiion of Evaluation metrics
corr_coef = np.corrcoef(y_pred,X_val['Republican'])[1, 0]

R_sqrd = corr_coef ** 2
print("R squared value:",R_sqrd)

adjusted_r = 1 - (((1-R_sqrd)*(n-1))/(n-p-1))
print("Adjusted R squared value:",adjusted_r)

rmse_val = math.sqrt(mean_squared_error(y_pred, X_val['Republican']))
print('RMSE value:',rmse_val)
```

```
R squared value: 0.7238886663016905
Adjusted R squared value: 0.7198189981179286
RMSE value: 15962.567869419843
```

Best performing model for Democratic values: Linear Regression with the predictor varibales as 'Total Population', 'Percent Black, not Hispanic or Latino', 'Percent Less than Bachelor's Degree' as it has the highest r squared value (0.950506110643013)

Best performing model for Republican values: Linear Regression with the predictor varibales as 'Total Population', 'Percent White, not Hispanic or Latino', 'Percent Hispanic or Latino', 'Percent Foreign Born', 'Percent Age 65 and Older', 'Percent Unemployed', 'Median Household Income', 'Percent Rural' as it has the highest r squared value (0.7302080671530998)

Task 4

Build a classification model to classify each county as Democratic or Republican. Consider at least two different classification techniques with multiple combinations of parameters and multiple combinations of variables. Compute evaluation metrics for the validation set and report your results.

4.1: Decision Tree

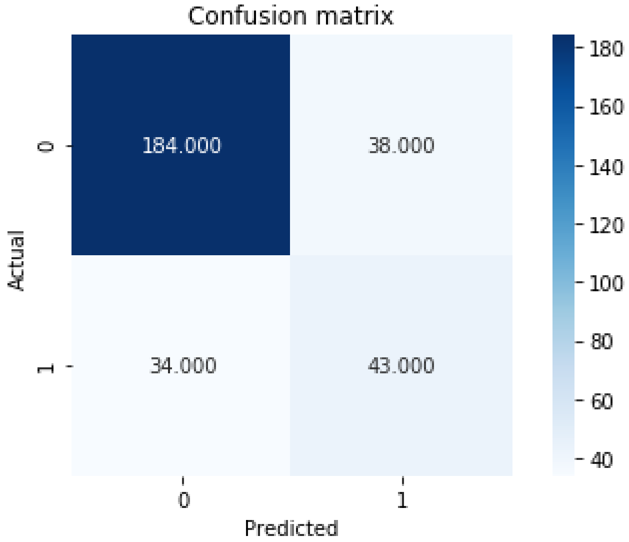
4.1.1: Using all predictor variables

In [33]:
```
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train_scaled, y_train)
```

Out[33]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, 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='deprecated',
                       random_state=0, splitter='best')
```

In [34]:
```
y_pred = classifier.predict(x_val_scaled)
```

In [35]:
```
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()
```



In [36]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7591973244147158, 0.24080267558528423, 0.5308641975308642, 0.5584415584415584, 0.5443037974683544]
```

4.1.2: Using Race Variables

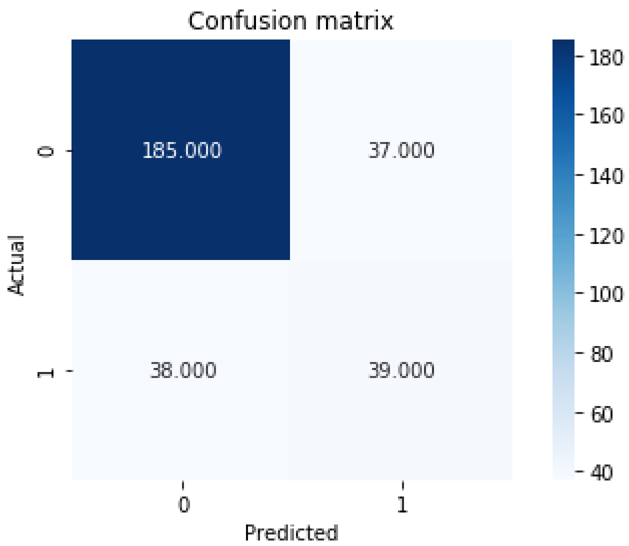
In [37]:
```
selected_predictor_variables = ['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born']
```

In [38]:
```
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[38]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, 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='deprecated',
                       random_state=0, splitter='best')
```

In [39]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [40]:
```
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()
```



In [41]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7491638795986622, 0.25083612040133785, 0.5131578947368421, 0.5064935064935064, 0.5098039215686275]
```

4.1.3: Using Age & Gender Variables

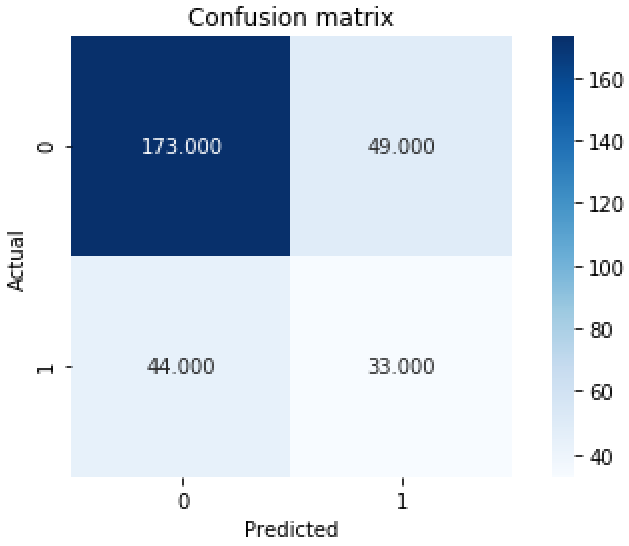
In [42]:
```
selected_predictor_variables = ['Percent Female',
                                'Percent Age 29 and Under',
                                'Percent Age 65 and Older']
```

In [43]:
```
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[43]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, 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='deprecated',
                       random_state=0, splitter='best')
```

In [44]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [45]:
```
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()
```



In [46]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.6889632107023411, 0.3110367892976589, 0.4024390243902439, 0.42857142857142855, 0.41509433962264153]
```

4.1.4: Using Employement, Education and Rural Variables

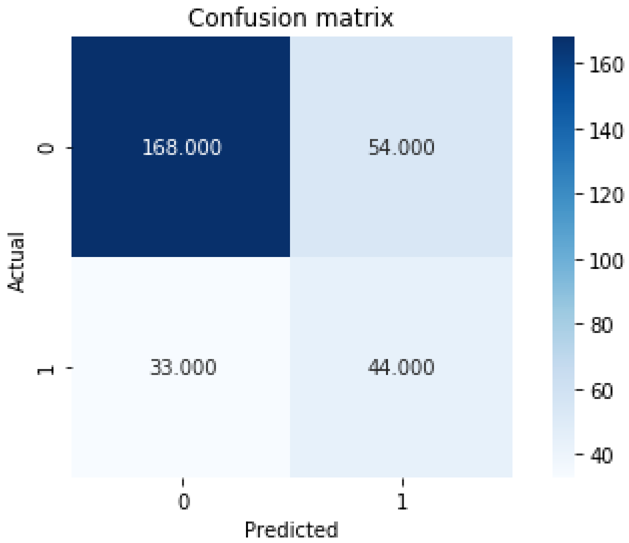
In [47]:
```
selected_predictor_variables = ['Median Household Income',
                                'Percent Unemployed',
                                'Percent Less than High School Degree',
                                'Percent Less than Bachelor\'s Degree',
                                'Percent Rural']
```

In [48]:
```
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[48]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, 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='deprecated',
                       random_state=0, splitter='best')
```

In [49]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [50]:
```
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()
```



In [51]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7090301003344481, 0.2909698996655187, 0.4489795918367347, 0.5714285714285714, 0.5028571428571429]
```

4.1.5: Using the following predictor variables

- Percent White, not Hispanic or Latino
- Percent Black, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Age 29 and Under
- Percent Less than High School Degree
- Percent Less than Bachelor's Degree

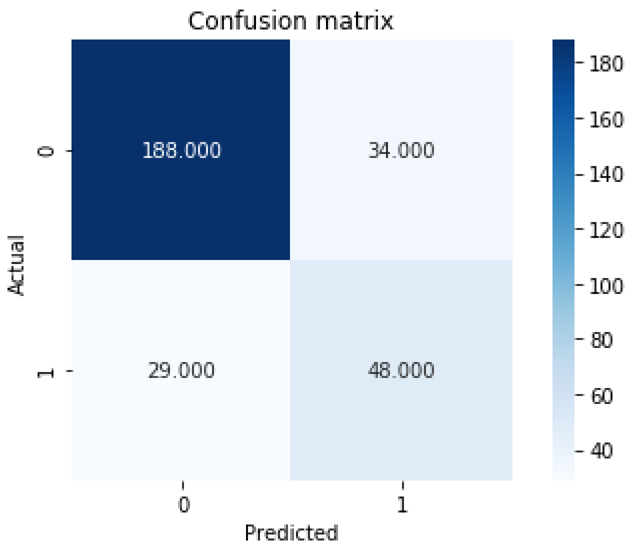
In [52]:
```
selected_predictor_variables = ['Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Age 29 and Under',
                                'Percent Less than High School Degree',
                                "Percent Less than Bachelor's Degree"]
```

In [53]:
```
classifier = DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[53]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, 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='deprecated',
                       random_state=0, splitter='best')
```

In [54]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [55]:
```
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()
```



In [56]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7892976588628763, 0.21070234113712372, 0.5853658536585366, 0.6233766233766234, 0.6037735849056604]
```

4.2: K - Nearest neighbors

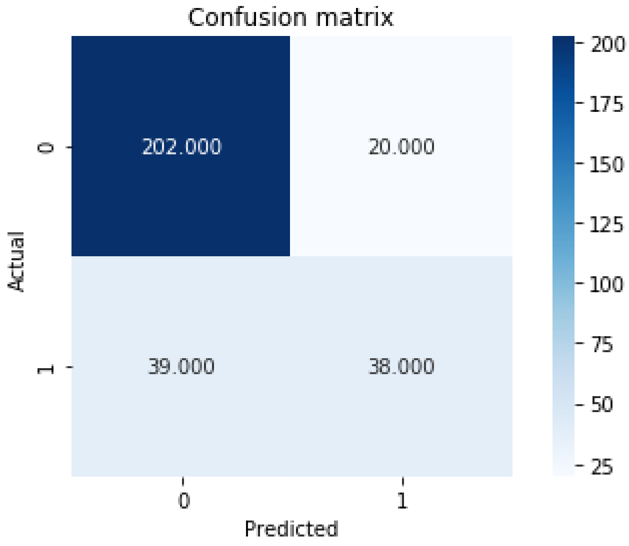
4.2.1: Using all predictor variables

In [57]:
```
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train_scaled, y_train)
```

Out[57]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [58]:
```
y_pred = classifier.predict(x_val_scaled)
```

In [59]:
```
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()
```



In [60]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.802675585284281, 0.19732441471571904, 0.6551724137931034, 0.4935064935064935, 0.562962962962963]
```

4.2.2: Using Race Variables

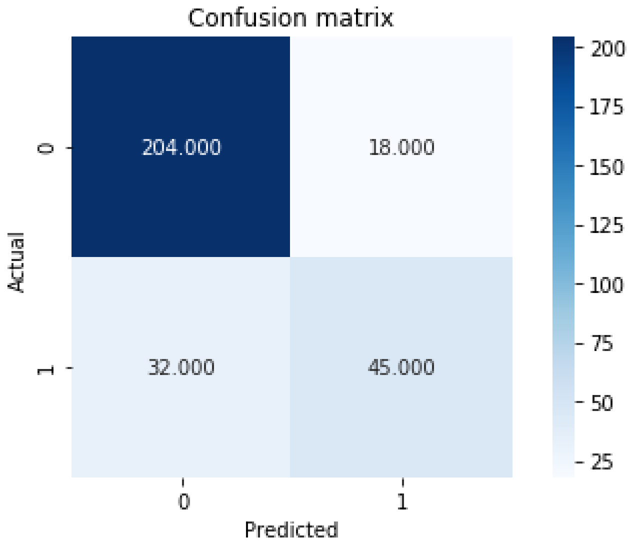
In [61]:
```
selected_predictor_variables = ['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born']
```

In [62]:
```
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[62]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [63]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [64]:
```
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()
```



In [65]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8327759197324415, 0.16722408026755853, 0.7142857142857143, 0.5844155844155844, 0.6428571428571429]
```

4.2.3: Using Age & Gender Variables

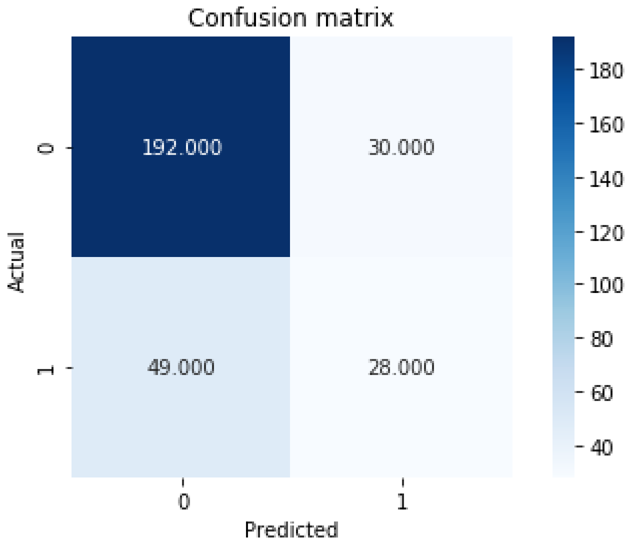
In [66]:
```
selected_predictor_variables = ['Percent Female',
                                'Percent Age 29 and Under',
                                'Percent Age 65 and Older']
```

In [67]:
```
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[67]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [68]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [69]:
```
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()
```



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

```
[0.7357859531772575, 0.2642140468227425, 0.4827586206896552, 0.36363636363636365, 0.4148148148148148]
```

4.2.4: Using Employement, Education and Rural Variables

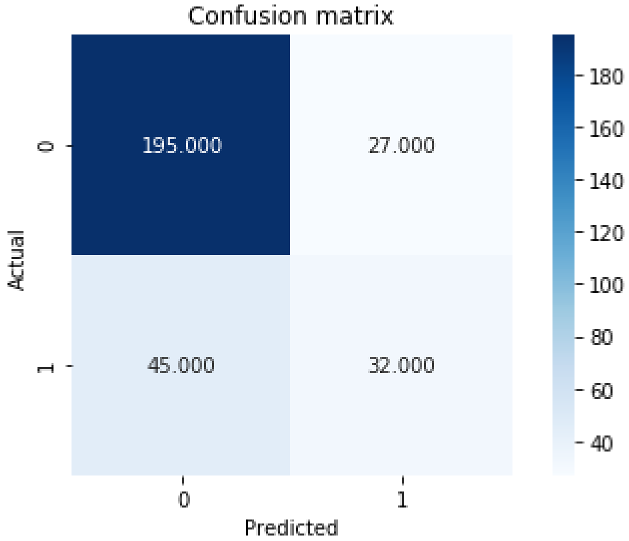
In [71]:
```
selected_predictor_variables = ['Median Household Income',
                                'Percent Unemployed',
                                'Percent Less than High School Degree',
                                'Percent Less than Bachelor\'s Degree',
                                'Percent Rural']
```

In [72]:
```
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[72]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [73]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [74]:
```
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()
```



In [75]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7591973244147158, 0.24080267558528423, 0.5423728813559322, 0.4155844155844156, 0.47058823529411764]
```

4.2.5: Using the following predictor variables

- Percent White, not Hispanic or Latino
- Percent Black, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Age 29 and Under
- Percent Less than High School Degree
- Percent Less than Bachelor's Degree

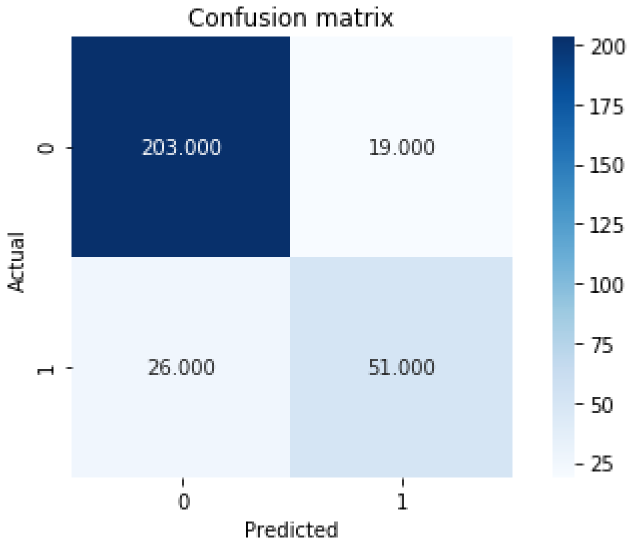
In [76]:
```
selected_predictor_variables = ['Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Age 29 and Under',
                                'Percent Less than High School Degree',
                                "Percent Less than Bachelor's Degree"]
```

In [77]:
```
classifier_k_nearest = KNeighborsClassifier(n_neighbors=3)
classifier_k_nearest.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[77]:
```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

In [78]:
```
y_pred = classifier_k_nearest.predict(x_val_scaled_df[selected_predictor_variables])
```

In [79]:
```
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()
```



In [80]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8494983277591973, 0.1505016722408027, 0.7285714285714285, 0.6623376623376623, 0.6938775510204082]
```

4.3: Naive Bayes

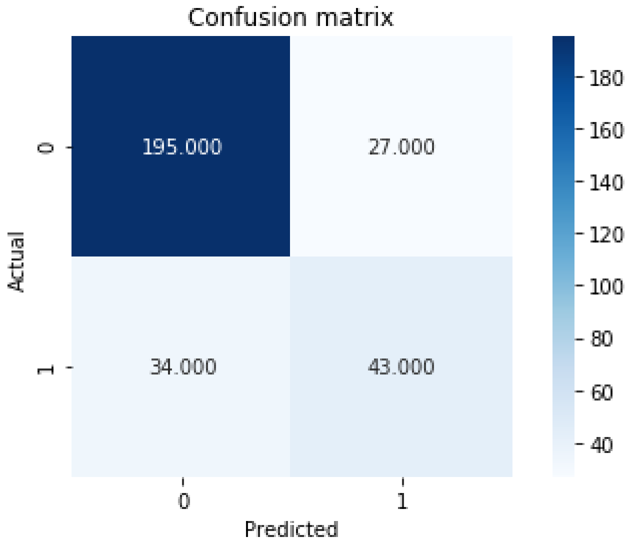
4.3.1: Using all predictor variables

In [81]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled, y_train)
```

Out[81]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [82]:
```
y_pred = classifier.predict(x_val_scaled)
```

In [83]:
```
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()
```



In [84]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7959866220735786, 0.20401337792642138, 0.6142857142857143, 0.5584415584415584, 0.5850340136054422]
```

4.3.2: Using Race Variables

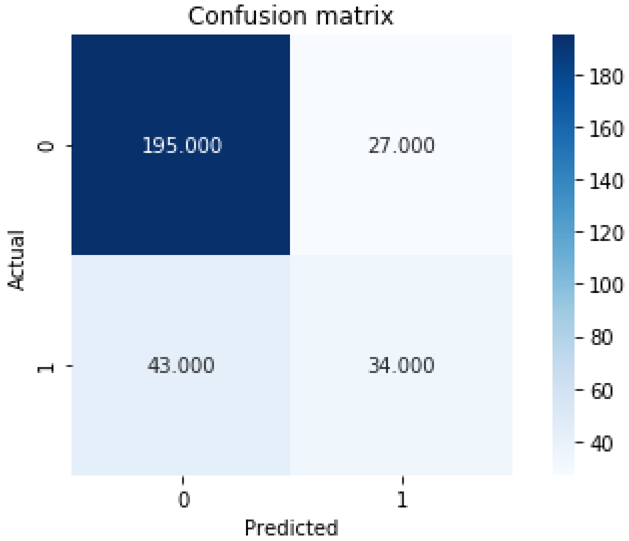
In [85]:
```
selected_predictor_variables = ['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born']
```

In [86]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[86]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [87]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [88]:
```
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()
```



In [89]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7658862876254181, 0.2341137123745819, 0.5573770491803278, 0.44155844155844154, 0.49275362318840582]
```

4.3.3: Using Age & Gender Variables

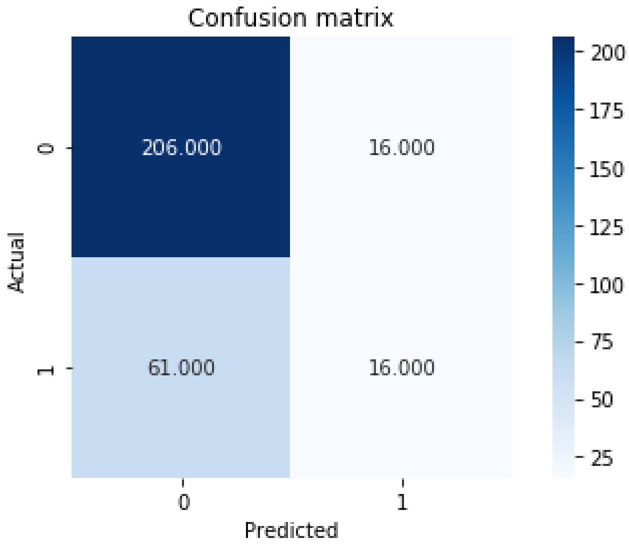
In [90]:
```
selected_predictor_variables = ['Percent Female',
                                'Percent Age 29 and Under',
                                'Percent Age 65 and Older']
```

In [91]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[91]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [92]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [93]:
```
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()
```



In [94]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7424749163879598, 0.2575250836120402, 0.5, 0.2077922077922078, 0.29357798165137616]
```

4.3.4: Using Employement, Education and Rural Variables

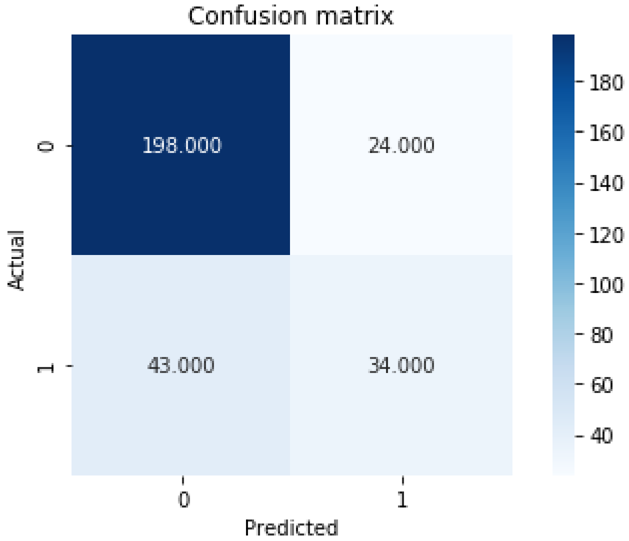
In [95]:
```
selected_predictor_variables = ['Median Household Income',
                                'Percent Unemployed',
                                'Percent Less than High School Degree',
                                'Percent Less than Bachelor\'s Degree',
                                'Percent Rural']
```

In [96]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[96]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [97]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [98]:
```
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()
```



In [99]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7759197324414716, 0.22408026755852842, 0.5862068965517241, 0.44155844155844154, 0.5037037037037037]
```

4.3.5: Using the following predictor variables

- Percent White, not Hispanic or Latino
- Percent Black, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Age 29 and Under
- Percent Less than High School Degree
- Percent Less than Bachelor's Degree

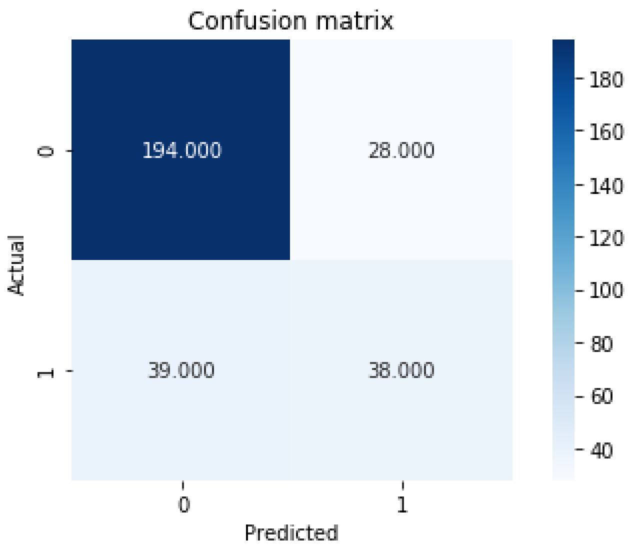
In [100]:
```
selected_predictor_variables = ['Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Age 29 and Under',
                                'Percent Less than High School Degree',
                                "Percent Less than Bachelor's Degree"]
```

In [101]:
```
classifier = GaussianNB()
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[101]:
```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [102]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [103]:
```
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()
```



In [104]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7759197324414716, 0.2240802675585284, 0.5757575757575758, 0.4935064935064935, 0.5314685314685315]
```

4.4: Support Vector Machines

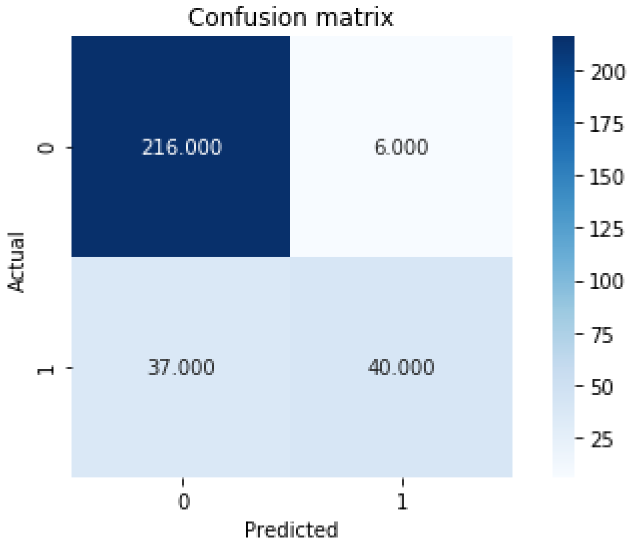
4.4.1: Using all predictor variables

In [105]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled, y_train)
```

Out[105]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [106]:
```
y_pred = classifier.predict(x_val_scaled)
```

In [107]:
```
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()
```



In [108]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8561872909698997, 0.14381270903010035, 0.8695652173913043, 0.5194805194805194, 0.6504065040650406]
```

4.4.2: Using Race Variables

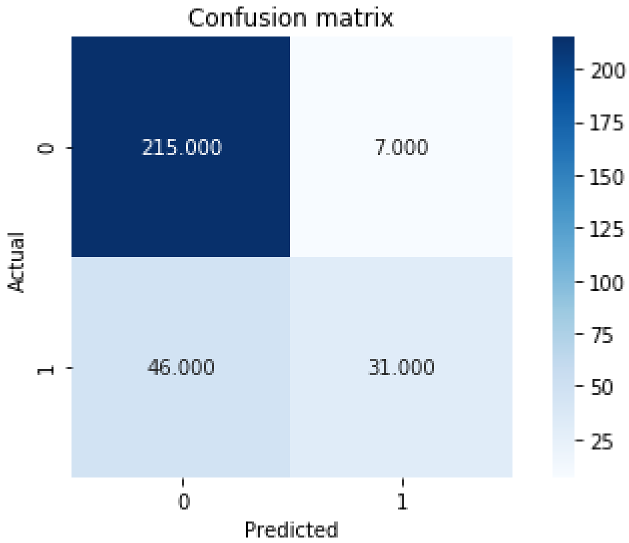
In [109]:
```
selected_predictor_variables = ['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born']
```

In [110]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

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

In [111]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [112]:
```
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()
```



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

```
[0.822742474916388, 0.17725752508361203, 0.8157894736842105, 0.4025974025974026, 0.5391304347826087]
```

4.4.3: Using Age & Gender Variables

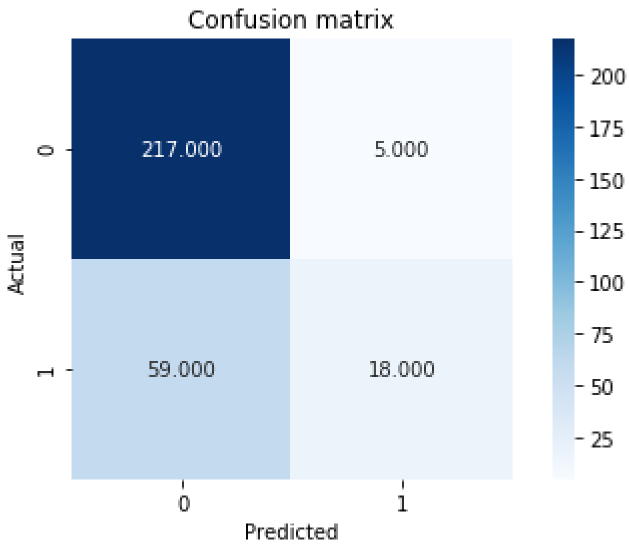
In [114]:
```
selected_predictor_variables = ['Percent Female',
                                'Percent Age 29 and Under',
                                'Percent Age 65 and Older']
```

In [115]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[115]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [116]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [117]:
```
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()
```



In [118]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7859531772575251, 0.21404682274247488, 0.782608695652174, 0.23376623376623376, 0.36]
```

4.4.4: Using Employement, Education and Rural Variables

In [119]:
```
selected_predictor_variables = ['Median Household Income',
                                'Percent Unemployed',
                                'Percent Less than High School Degree',
                                'Percent Less than Bachelor\'s Degree',
                                'Percent Rural']
```

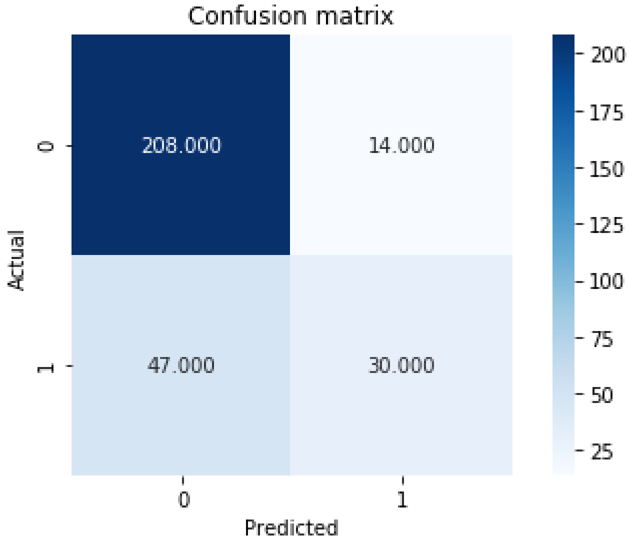
In [120]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[120]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [121]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [122]:

```
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()
```



In [123]:

```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7959866220735786, 0.20401337792642138, 0.6818181818181818, 0.38961038961038963, 0.49586776859504134]
```

4.4.5: Using the following predictor variables

- Percent White, not Hispanic or Latino
- Percent Black, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Age 29 and Under
- Percent Less than High School Degree
- Percent Less than Bachelor's Degree

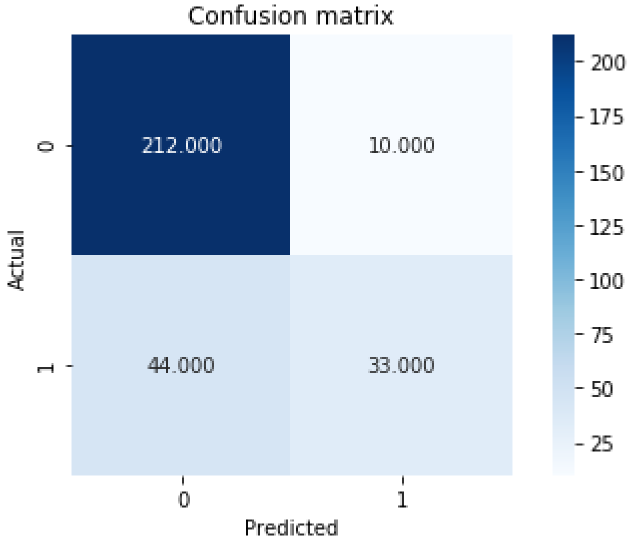
In [124]:
```
selected_predictor_variables = ['Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Age 29 and Under',
                                'Percent Less than High School Degree',
                                "Percent Less than Bachelor's Degree"]
```

In [125]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled_df[selected_predictor_variables], y_train)
```

Out[125]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [126]:
```
y_pred = classifier.predict(x_val_scaled_df[selected_predictor_variables])
```

In [127]:
```
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()
```



In [128]:
```
accuracy = metrics.accuracy_score(y_val, y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_val, y_pred)
recall = metrics.recall_score(y_val, y_pred)
F1_score = metrics.f1_score(y_val, y_pred)
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8193979933110368, 0.18060200668896320, 0.7674418604651163, 0.42857142857142855, 0.5499999999999999]
```

4.4: Model Performances

- **Decision Tree**

Variables	Accuracy	Precision	Recall	F1 Score
All	0.76	0.53	0.55	0.54
Race	0.75	0.51	0.51	0.51
Age & Gender	0.69	0.4	0.43	0.42
Employement, Education and Rural Variables	0.71	0.45	0.57	0.5
Optimum	0.79	0.59	0.62	0.6

- **K Nearest Neighbors**

Variables	Accuracy	Precision	Recall	F1 Score
All	0.8	0.66	0.49	0.56
Race	0.83	0.71	0.58	0.64
Age & Gender	0.74	0.48	0.36	0.42
Employement, Education and Rural Variables	0.76	0.54	0.42	0.47
Optimum	0.85	0.72	0.66	0.69

- **Naive Bayes**

Variables	Accuracy	Precision	Recall	F1 Score
All	0.8	0.61	0.56	0.58
Race	0.77	0.56	0.44	0.49
Age & Gender	0.74	0.5	0.2	0.29
Employement, Education and Rural Variables	0.78	0.57	0.44	0.5
Optimum	0.78	0.58	0.49	0.53

- **Support Vector Machines**

Variables	Accuracy	Precision	Recall	F1 Score
All	0.86	0.87	0.52	0.65
Race	0.82	0.82	0.41	0.53
Age & Gender	0.79	0.78	0.23	0.36
Employement, Education and Rural Variables	0.8	0.68	0.4	0.5
Optimum	0.82	0.77	0.43	0.55

What is the best performing classification model?

Answer: K-Nearest Neighbors with k = 3 using the following variables gives the best F1 Score

- Percent White, not Hispanic or Latino
- Percent Black, not Hispanic or Latino
- Percent Hispanic or Latino
- Percent Age 29 and Under
- Percent Less than High School Degree
- Percent Less than Bachelor's Degree

What is the performance of the model?

- Accuracy: 0.85
- Precision: 0.72
- Recall: 0.66
- F1 Score: 0.69

How did you select the parameters of the model?

Answer: We have one parameters for K-Nearest Neighbors model, the number of nearest neighbors to consider. Among k = 1 to 5, k = 3 gave best performance across all variable combinations.

How did you select the variables of the model?

Answer:

- From the types of variables: Race, Age, Gender, Employement, Education & Rural, we choose a combination of types of variables.
- Upon trial of many different combinations we understand that any single group of variables do not contribute to the best performance of a model. But a combination of different variables. That is Race, Age & Gender, Employment & Education on itself do not work best.
- Race, Age and Education are the best predictor variables.

Task 5

Build a clustering model to cluster the counties. Consider at least two different clustering techniques with multiple combinations of parameters and multiple combinations of variables. 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.

In [129]:
```
all_x = x_val_scaled_df.to_numpy()
best_classification_x = x_val_scaled_df[['Percent White, not Hispanic or Latino',
                                         'Percent Black, not Hispanic or Latino',
                                         'Percent Hispanic or Latino',
                                         'Percent Age 29 and Under',
                                         'Percent Less than High School Degree',
                                         "Percent Less than Bachelor's Degree"]].to_numpy()
best_classification_x_by_clustering_algorithm = x_val_scaled_df[['Total Population',
                                                 'Percent Foreign Born',
                                                 'Percent Less than High School Degree',
                                                 "Percent Less than Bachelor's Degree"]].to_numpy()
best_clustering_x = x_val_scaled_df[['Total Population']].to_numpy()
y = y_val.to_numpy()
```

5.1: Hierarchical Clustering - Single Linkage

5.1.1: Using all predictor variables

In [130]:
```
clustering = linkage(all_x, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [131]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(all_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.5761121869509526]
```

5.1.2: Using best classification predictor variables

In [132]:
```
clustering = linkage(best_classification_x, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [133]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.39604171483766787]
```

5.1.3: Using best classification predictor variables by clustering algorithm

In [134]:
```
clustering = linkage(best_classification_x_by_clustering_algorithm, method = "single", metric = "euclidea
n")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [135]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x_by_clustering_algorithm,
                                                  clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.7789158543666239]
```

5.1.4: Using best clustering predictor variables

In [136]:
```
clustering = linkage(best_clustering_x, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [137]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_clustering_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.9335336089390852]
```

5.2: Hierarchical Clustering - Complete Linkage

5.2.1: Using all predictor variables

In [138]:
```
clustering = linkage(all_x, method = "complete", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [139]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(all_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.24558873323923197, 0.2550107182979104]
```

5.2.2: Using best classification predictor variables

In [140]:
```
clustering = linkage(best_classification_x, method = "complete", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [141]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.05050820092852576, 0.55498710702161]
```

5.2.3: Using best classification predictor variables by clustering algorithm

In [142]:
```
clustering = linkage(best_classification_x_by_clustering_algorithm, method = "complete", metric = "euclid
ean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [143]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x_by_clustering_algorithm,
                                                  clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.7789158543666239]
```

5.2.4: Using best clustering predictor variables

In [144]:
```
clustering = linkage(best_clustering_x, method = "complete", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [145]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_clustering_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.9335336089390852]
```

5.3: KMeans Clustering

5.3.1: Using all predictor variables

In [146]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state=0).fit(all_x)
clusters = clustering.labels_
```

In [147]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(all_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.2157355337732104, 0.3275869370675781]
```

5.3.2: Using best classification predictor variables

In [148]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state=0).fit(best_classification
_x)
clusters = clustering.labels_
```

In [149]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.04510099084307282, 0.47971948560601607]
```

5.3.3 Using best classification predictor variables by clustering algorithm

In [150]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state=0).fit(best_classification
_x_by_clustering_algorithm)
clusters = clustering.labels_
```

In [151]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x_by_clustering_algorithm,
                                                  clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.3297416730135844, 0.44016317374179337]
```

5.3.4: Using best clustering predictor variables

In [152]:
```
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state=0).fit(best_clustering_x)
clusters = clustering.labels_
```

In [153]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_clustering_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.131388585061464, 0.8592746979179324]
```

5.4: DBSCAN Clustering - Eps: 1.5, MinPts: 5

5.4.1: Using all predictor variables

In [154]:
```
clustering = DBSCAN(eps = 1.5, min_samples = 5, metric = "euclidean").fit(all_x)
clusters = clustering.labels_
```

In [155]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(all_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.031145536387172504, 0.17571212619876678]
```

5.4.2: Using best classification predictor variables

In [156]:
```
clustering = DBSCAN(eps = 1.5, min_samples = 5, metric = "euclidean").fit(best_classification_x)
clusters = clustering.labels_
```

In [157]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.20189915786722126, 0.487127764571997]
```

5.4.3: Using best classification predictor variables by clustering algorithm

```
In [158]: clustering = DBSCAN(eps = 1.5, min_samples = 5, metric = "euclidean").fit(best_classification_x_by_cluste
ring_algorithm)
clusters = clustering.labels_
```

```
In [159]: adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x_by_clustering_algorithm,
                                                  clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.06262562590983287, 0.6884441860526223]
```

5.4.3: Using best clustering predictor variables

```
In [160]: clustering = DBSCAN(eps = 1.5, min_samples = 5, metric = "euclidean").fit(best_clustering_x)
clusters = clustering.labels_
```

```
In [161]: adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_clustering_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.9335336089390852]
```

5.5: DBSCAN Clustering - Eps: 1.0, MinPts: 3

5.5.1: Using all predictor variables

```
In [162]: clustering = DBSCAN(eps = 1.0, min_samples = 5, metric = "euclidean").fit(all_x)
clusters = clustering.labels_
```

```
In [163]: adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(all_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[-0.06112315653389419, -0.24675582876304566]
```

5.5.2: Using best classification predictor variables

In [164]:
```
clustering = DBSCAN(eps = 1.0, min_samples = 3, metric = "euclidean").fit(best_classification_x)
clusters = clustering.labels_
```

In [165]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.11763258868329998, 0.2222266780524665]
```

5.5.3: Using best classification predictor variables by clustering algorithm

In [166]:
```
clustering = DBSCAN(eps = 1.0, min_samples = 3, metric = "euclidean").fit(best_classification_x_by_cluste
ring_algorithm)
clusters = clustering.labels_
```

In [167]:
```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_classification_x_by_clustering_algorithm, clusters
, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.1678995613772082, 0.5169657867513632]
```

5.5.4: Using best clustering predictor variables

In [168]:
```
clustering = DBSCAN(eps = 1.0, min_samples = 3, metric = "euclidean").fit(best_clustering_x)
clusters = clustering.labels_
```

In [169]:

```
adjusted_rand_index = metrics.adjusted_rand_score(y, clusters)
silhouette_coefficient = metrics.silhouette_score(best_clustering_x, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.01254522751329356, 0.9335336089390852]
```

5.6: Model Performances

- **Hierarchical Clustering - Single Linkage**

Variables	Classification	Clustering
All	0.0125	0.5761
Best Classification	0.0125	0.396
Best Classification(by clustering)	0.0125	0.7789
Best Clustering	0.0125	0.9335

- **Hierarchical Clustering - Complete Linkage**

Variables	Classification	Clustering
All	0.2455	0.255
Best Classification	0.0505	0.5549
Best Classification(by clustering)	0.0125	0.7789
Best Clustering	0.0125	0.9335

- **KMeans Clustering**

Variables	Classification	Clustering
All	0.2157	0.3275
Best Classification	0.0451	0.4797
Best Classification(by clustering)	0.3297	0.4401
Best Clustering	0.1313	0.8593

- **DBSCAN Clustering - Eps: 1.5, MinPts: 5**

Variables	Classification	Clustering
All	0.0311	0.1757
Best Classification	0.201	0.4871

Variables	Classification	Clustering
Best Classification(by clustering)	0.0626	0.6884
Best Clustering	0.0125	0.9335

- **DBSCAN Clustering - Eps: 1.0, MinPts: 3**

Variables	Classification	Clustering
All	-0.0611	-0.2467
Best Classification	0.1176	0.2222
Best Classification(by clustering)	0.1678	0.5169
Best Clustering	0.0125	0.9335

What is the best performing clustering model?

Answer: All clustering models besides K-Means perform identically.

What is the performance of the model?

Answer: Silhouette Coefficient: 0.9335

How did you select the parameters of model?

Answer: Parameters

- Hierarchical Methods have no parameters to fine tune
- KMeans parameters may perform better if we increase the number of iterations.
- DBSCAN parameters, eps was approximated by obtaining the modes
  - All: 2.7061026407808066
  - Best Classification: 1.022669045646768
  - Best Classification(by clustering): 1.063542197328302
  - Best Clustering: 0.6652472079887632 (2nd most frequent distance)

How did you select the variables of the model?

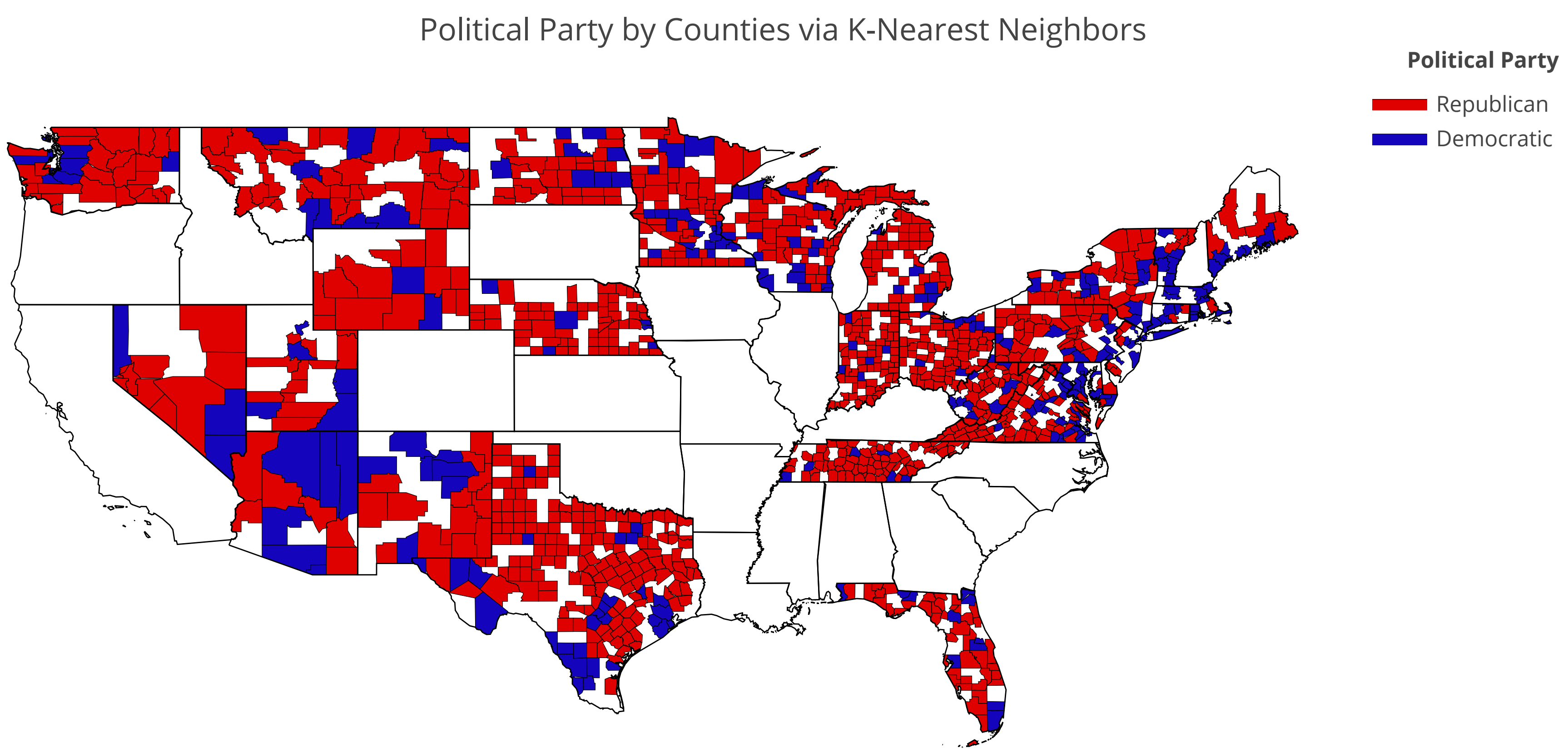
Answer: Through trial of many different variable combinations we find the best variable for clustering to be *Total Population*.

Task 6

Create a map of Democratic counties and Republican counties using the counties' FIPS codes and Python's Plotly library (plot.ly/python/county-choropleth/). Compare with the map of Democratic counties and Republican counties created in Project 01. What conclusions do you make from the plots?

In [170]:

```
comp_data = election_data.iloc[:,:18]
_data = comp_data.select_dtypes(include=[np.int64,np.float64])
_data = _data.iloc[:,1:14]

# Standardizztion of data
scaler = StandardScaler()
scaler.fit(x_train)
_data_scaled = scaler.transform(_data)
_data_scaled_df = pd.DataFrame(_data_scaled,index = _data.index,columns=_data.columns)

# classification Model K-nearest neighbors
prime_pred = classifier_k_nearest.predict(_data_scaled_df[['Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Age 29 and Under',
                                'Percent Less than High School Degree',
                                "Percent Less than Bachelor's Degree"]])

# Merging for fips
prime_data = pd.DataFrame({'Party': prime_pred, 'FIPS': comp_data['FIPS']})

# Map of Democratic and Republic counties with FIPS codes
import plotly.figure_factory as f
from plotly.offline import init_notebook_mode, iplot
init_notebook_mode(connected=True)

fips = prime_data['FIPS'].tolist()
_values = prime_data['Party'].map({0: 'Republican', 1: 'Democratic'})
colors = ["#1405BD", "#DE0100"]
figure = f.create_choropleth(colorscale=colors,
                             fips=fips,
                             county_outline={'color': '#000000', 'width': 0.3},
                              state_outline={'color': '#000000', 'width': 0.7},
                             legend_title='Political Party',
                             values=_values,
                             title='Political Party by Counties via K-Nearest Neighbors')
figure.layout.template = None
iplot(figure, validate=False)
```



Task 7

Use your best performing regression and classification models to predict the number of votes cast for the Democratic party in each county, the number of votes cast for the Republican party in each county, and the party (Democratic or Republican) of each county for the test dataset (demographics_test.csv). Save the output in a single CSV file. For the expected format of the output, see sample_output.csv

In [171]:
```python
election_demographic = pd.read_csv('demographics_test.csv')
election_demographic.head()
```

Out[171]:

	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	Percent Age 29 and Under	Percent Age 65 and Older	Median Household Income	Percent Unemployed	P Les $ I
0	NV	eureka	32011	1730	98.265896	0.057803	0.462428	0.346821	51.156069	27.109827	15.606936	70000	3.755365	8.4
1	TX	zavala	48507	12107	5.798299	0.594697	93.326175	9.193029	49.723301	49.302057	12.480383	26639	11.955168	40.8
2	VA	king george	51099	25260	73.804434	16.722090	4.441805	2.505938	50.166271	40.186065	11.868567	84342	6.479939	7.1
3	OH	hamilton	39061	805965	66.354867	25.654340	2.890944	5.086945	51.870615	40.779686	14.161657	50399	7.864630	9.8
4	TX	austin	48015	29107	63.809393	8.479060	25.502456	9.946061	50.671660	37.351840	17.799842	56681	5.782337	17.5

In [172]:
```python
x_test = election_demographic.select_dtypes(include=[np.int64,np.float64])
x_test = x_test.iloc[:,1:14]
x_test_scaled = scaler.transform(x_test)
x_test_scaled_df = pd.DataFrame(x_test_scaled,index = x_test.index,columns=x_test.columns)
```

In [173]:
```python
y_pred_democratic = fitted_model_democratic.predict(x_test_scaled_df[['Total Population',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Less than Bachelor\'s Degree']])
election_demographic['Democratic'] = y_pred_democratic
```

In [174]:
```python
y_pred_republican = fitted_model_republican.predict(x_test_scaled_df[['Total Population',
                                'Percent White, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Foreign Born',
                                'Percent Age 65 and Older',
                                'Percent Unemployed',
                                'Median Household Income',
                                'Percent Rural']])
election_demographic['Republican'] = y_pred_republican
```

In [175]:
```python
y_pred_party = classifier_k_nearest.predict(x_test_scaled_df[['Percent White, not Hispanic or Latino',
                                'Percent Black, not Hispanic or Latino',
                                'Percent Hispanic or Latino',
                                'Percent Age 29 and Under',
                                'Percent Less than High School Degree',
                                "Percent Less than Bachelor's Degree"]])
election_demographic['Party'] = y_pred_party
election_demographic.head()
```

Out[175]:

	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	Percent Age 29 and Under	Percent Age 65 and Older	Median Household Income	Percent Unemployed	P Les
0	NV	eureka	32011	1730	98.265896	0.057803	0.462428	0.346821	51.156069	27.109827	15.606936	70000	3.755365	8.4
1	TX	zavala	48507	12107	5.798299	0.594697	93.326175	9.193029	49.723301	49.302057	12.480383	26639	11.955168	40.8
2	VA	king george	51099	25260	73.804434	16.722090	4.441805	2.505938	50.166271	40.186065	11.868567	84342	6.479939	7.1
3	OH	hamilton	39061	805965	66.354867	25.654340	2.890944	5.086945	51.870615	40.779686	14.161657	50399	7.864630	9.8
4	TX	austin	48015	29107	63.809393	8.479060	25.502456	9.946061	50.671660	37.351840	17.799842	56681	5.782337	17.5

In [176]:
```python
sample_output = election_demographic[['State','County', 'Democratic', 'Republican', 'Party']]
sample_output.head()
```

Out[176]:

	State	County	Democratic	Republican	Party
0	NV	eureka	-4368.133477	10279.986522	1
1	TX	zavala	-9771.647091	-87.022736	1
2	VA	king george	21823.049764	18795.181860	1
3	OH	hamilton	183669.476767	112375.441324	1
4	TX	austin	7294.738614	6193.106586	0

In [177]:

```
numeric_data = sample_output._get_numeric_data()
numeric_data[numeric_data < 0] = 0
sample_output.head()
```

Out[177]:

	State	County	Democratic	Republican	Party
0	NV	eureka	0.000000	10279.986522	1
1	TX	zavala	0.000000	0.000000	1
2	VA	king george	21823.049764	18795.181860	1
3	OH	hamilton	183669.476767	112375.441324	1
4	TX	austin	7294.738614	6193.106586	0

In [178]:

```
sample_output.to_csv("output.csv")
```