Type Markdown and LaTeX: α2

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier #KNN classifier for queston
4
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.svm import SVC
from sklearn import metrics
import seaborn as sns
from sklearn import linear_model
import statsmodels.formula.api as smf
from scipy.cluster.hierarchy import linkage, fcluster
from sklearn.cluster import KMeans, DBSCAN
from sklearn import metrics
```

In [2]:
```
merged_train = pd.read_csv("merged_train.csv")
```

Question 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?

In [3]:
```
# Partition dataset into training, validation, and test sets using holdout met
hod
x_train_dem, x_test_dem, y_train_dem, y_test_dem = train_test_split(merged_tra
in[['Percent White, not Hispanic or Latino',
                                                           'Percent Black, not
 Hispanic or Latino',
                                                           'Percent Less than H
igh School Degree',
                                                           'Percent Less than B
achelor\'s Degree', 'Total Population']],
                                                           merged_train['Democr
atic'], test_size = 0.25, random_state = 1)
x_train_rep, x_test_rep, y_train_rep, y_test_rep = train_test_split(merged_tra
in[['Percent White, not Hispanic or Latino',
                                                           'Percent Black, not
 Hispanic or Latino',
                                                           'Percent Less than H
igh School Degree',
                                                           'Percent Less than B
achelor\'s Degree', 'Total Population']],
                                                           merged_train['Republ
ican'], test_size = 0.25, random_state = 1)
```

Question 2

Standardize the training set and the validation set.

In [4]:
```
scaler = StandardScaler()
scaler.fit(x_train_dem)
x_train_dem_scaled = scaler.transform(x_train_dem)
x_test_dem_scaled = scaler.transform(x_test_dem)

scaler.fit(x_train_rep)
x_train_rep_scaled = scaler.transform(x_train_rep)
x_test_rep_scaled = scaler.transform(x_test_rep)
```

Question 3

Build a linear regression model to predict the number of votes cast for the Democratic party in each county. Consider multiple combinations of predictor variables. Compute evaluation metrics for the validation set and report your results. What is the best performing linear regression model? What is the performance of the model? How did you select the variables of the model?

- `Repeat this task for the number of votes cast for the Republican party in each county.`

Democratic Regression Models

Each model using Total Population, 'Percent_Age_65_and_Older', 'Percent_White_not_Hispanic_or_Latino', 'Percent_Black_not_Hispanic_or_Latino', 'Percent_Less_than_High_School_Degree', and 'Percent_Less_than_Bachelor's_Degree'

Regular Linear Regression

In [5]:

```
#Using Total Population, Total age 29 and under, Percent Less than Bachelor's
 Degree
#
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_dem_scaled[:,[0,1,2,3,4]], y = y_train_de
m)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_dem.shape)

predicted = fitted_model.predict(x_test_dem_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_dem.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
print("Model score: ", model.score(X = x_train_dem_scaled[:,[0,1,2,3,4]], y =
y_train_dem))
display(predicted.shape)
display(y_test_dem.values.shape)
```

```
coef :  [ 2458.44005276  2245.10130322  2237.36927403 -9007.58810534
 66311.60973183]
Shape:  (896,)
Correlation coefficient:  0.9592659235828307
R Squared:  0.9201911121472212
Model score:  0.8715087853655452

/usr/local/Cellar/python/3.7.4_1/Frameworks/Python.framework/Versions/3.7/li
b/python3.7/site-packages/sklearn/linear_model/base.py:509: RuntimeWarning: i
nternal gelsd driver lwork query error, required iwork dimension not returne
d. This is likely the result of LAPACK bug 0038, fixed in LAPACK 3.2.2 (relea
sed July 21, 2010). Falling back to 'gelss' driver.
  linalg.lstsq(X, y)

(299,)

(299,)
```

Ridge Regression

In [6]:
```
model = linear_model.Ridge(alpha=1)
fitted_model = model.fit(X = x_train_dem_scaled[:,[0,1,2,3,4]], y = y_train_de
m)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_dem.shape)

predicted = fitted_model.predict(x_test_dem_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_dem.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
print("Model score: ", model.score(X = x_train_dem_scaled[:,[0,1,2,3,4]], y =
y_train_dem))
display(predicted.shape)
display(y_test_dem.values.shape)
```

```
coef :  [ 2414.61899566  2252.64034025  2209.11417731 -9015.00953377
 66216.7067524 ]
Shape:  (896,)
Correlation coefficient:  0.9592784149806972
R Squared:  0.9202150774478788
Model score:  0.8715075156666074
```

```
(299,)
```

```
(299,)
```

Lasso Regression

In [7]:
```
model = linear_model.Lasso(alpha=1)
fitted_model = model.fit(X = x_train_dem_scaled[:,[0,1,2,3,4]], y = y_train_de
m)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_dem.shape)

predicted = fitted_model.predict(x_test_dem_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_dem.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
print("Model score: ", model.score(X = x_train_dem_scaled[:,[0,1,2,3,4]], y =
y_train_dem))
display(predicted.shape)
display(y_test_dem.values.shape)
```

```
coef :  [ 2453.50437599  2243.38894281  2232.16485444 -9004.01541752
 66310.50429239]
Shape:  (896,)
Correlation coefficient:  0.9592667853300673
R Squared:  0.9201927654374813
Model score:  0.8715087824797675
```

```
(299,)
```

```
(299,)
```

ElasticNet Regression

In [8]:
```
model = linear_model.ElasticNet()
fitted_model = model.fit(X = x_train_dem_scaled[:,[0,1,2,3,4]], y = y_train_de
m)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_dem.shape)

predicted = fitted_model.predict(x_test_dem_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_dem.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
print("Model score: ", model.score(X = x_train_dem_scaled[:,[0,1,2,3,4]], y =
y_train_dem))
display(predicted.shape)
display(y_test_dem.values.shape)
```

```
coef :  [ -4246.5971995    4264.53414885  -1571.21026746 -10455.91667274
  41848.78605533]
Shape:  (896,)
Correlation coefficient:  0.9480397904254727
R Squared:  0.8987794442299742
Model score:  0.7852684922998014
```

```
(299,)
```

```
(299,)
```

Republican Regression Models

Regular Linear Regression

In [9]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(X = x_train_rep_scaled[:,[0,1,2,3,4]], y = y_train_re
p)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_rep.shape)

predicted = fitted_model.predict(x_test_rep_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_rep.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
display(predicted.shape)
display(y_test_rep.values.shape)
```

```
coef :  [ 1887.05920541 -1675.58278788   116.55188856 -4812.34094727
 38660.85257537]
Shape:  (896,)
Correlation coefficient:  0.9104778214953163
R Squared:  0.828969863434857
```

```
(299,)
```

```
(299,)
```

LASSO Regression

In [10]:
```
model = linear_model.Lasso()
fitted_model = model.fit(X = x_train_rep_scaled[:,[0,1,2,3,4]], y = y_train_re
p)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_rep.shape)

predicted = fitted_model.predict(x_test_rep_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_rep.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
display(predicted.shape)
display(y_test_rep.values.shape)
```

```
coef :  [ 1883.67216574 -1674.66912294   111.74943691 -4808.87356321
 38659.51085346]
Shape:  (896,)
Correlation coefficient:  0.9104799867108763
R Squared:  0.8289738062010374
```

```
(299,)
```

```
(299,)
```

ElasticNet Regression

In [11]:
```
model = linear_model.ElasticNet()
fitted_model = model.fit(X = x_train_rep_scaled[:,[0,1,2,3,4]], y = y_train_re
p)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_rep.shape)

predicted = fitted_model.predict(x_test_rep_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_rep.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
display(predicted.shape)
display(y_test_rep.values.shape)
```

```
coef :  [-1710.5004798    547.32186453 -1629.47163822 -5966.33232549
 24215.36677679]
Shape:  (896,)
Correlation coefficient:  0.8913889890811346
R Squared:  0.7945743298550871
```

```
(299,)
```

```
(299,)
```

Ridge Regression

In [12]:
```
model = linear_model.Ridge()
fitted_model = model.fit(X = x_train_rep_scaled[:,[0,1,2,3,4]], y = y_train_re
p)
print ("coef : ", fitted_model.coef_)
print ("Shape: ", y_train_rep.shape)

predicted = fitted_model.predict(x_test_rep_scaled[:,[0,1,2,3,4]])
corr_coef = np.corrcoef(x=predicted,y=y_test_rep.values)[1, 0]
print("Correlation coefficient: ", corr_coef)
R_squared = corr_coef**2
print("R Squared: ", R_squared)
display(predicted.shape)
display(y_test_rep.values.shape)
```

```
coef :  [ 1864.47723056 -1667.09414962   103.4059047  -4819.08415068
 38604.65120939]
Shape:  (896,)
Correlation coefficient:  0.9104557907471618
R Squared:  0.8289297469050396
```

```
(299,)
```

```
(299,)
```

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

Features 'Percent_Age_65_and_Older', 'Percent_White_not_Hispanic_or_Latino', 'Percent_Black_not_Hispanic_or_Latino', 'Percent_Less_than_High_School_Degree' and 'Percent_Less_than_Bachelor's_Degree' seem to be more important to determine whether a county is labeled as Democratic or Republican.

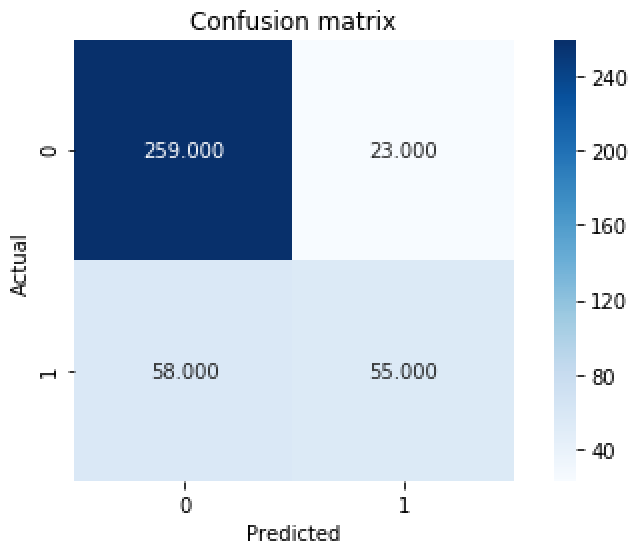
K-nearest Neighbors (KNN)

KNN Variation No. 1

In [13]:
```
X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent Whi
te, not Hispanic or Latino',
                                                          'Percent Black, not
 Hispanic or Latino',
                                                          'Percent Less than H
igh School Degree',
                                                          'Percent Less than B
achelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state =
1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [14]:
```
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train[:,[0,1,2,3]], Y_train)
Y_pred = classifier.predict(X_test[:,[0,1,2,3]])
```

In [15]:
```
conf_matrix = metrics.confusion_matrix(Y_test, Y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.
cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
# print(classification_report(Y_test, Y_pred))
```

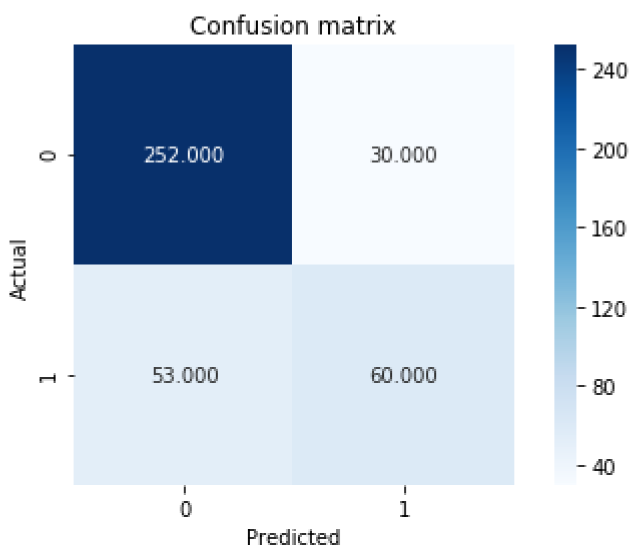


In [16]:
```
print("Accuracy: ", end=""); print(metrics.accuracy_score(Y_test, Y_pred)) # a
ccuracy
print("Error: ", end=""); print(1 - metrics.accuracy_score(Y_test, Y_pred)) #
 error
print("Precision: ", end=""); print(metrics.precision_score(Y_test, Y_pred, av
erage = None)) # precision
print("Recall: ", end=""); print(metrics.recall_score(Y_test, Y_pred, average
= None)) # recall
print("F1 Score: ", end=""); print(metrics.f1_score(Y_test, Y_pred, average =
None)) # F1 score
```

```
Accuracy: 0.7949367088607595
Error: 0.2050632911392405
Precision: [0.8170347  0.70512821]
Recall: [0.91843972 0.48672566]
F1 Score: [0.86477462 0.57591623]
```

KNN Variation No. 2

In [17]:
```
X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent White, not Hispanic or Latino',
                                                                  'Percent Black, not Hispanic or Latino',
                                                                  'Percent Age 65 and Older',
                                                                  'Percent Less than Bachelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state = 1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(X_train[:,[0,1,2,3]], Y_train)
Y_pred = classifier.predict(X_test[:,[0,1,2,3]])

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



In [18]:
```
print("Accuracy: ", end=""); print(metrics.accuracy_score(Y_test, Y_pred)) # a
ccuracy
print("Error: ", end=""); print(1 - metrics.accuracy_score(Y_test, Y_pred)) #
 error
print("Precision: ", end=""); print(metrics.precision_score(Y_test, Y_pred, av
erage = None)) # precision
print("Recall: ", end=""); print(metrics.recall_score(Y_test, Y_pred, average
= None)) # recall
print("F1 Score: ", end=""); print(metrics.f1_score(Y_test, Y_pred, average =
None)) # F1 score
```

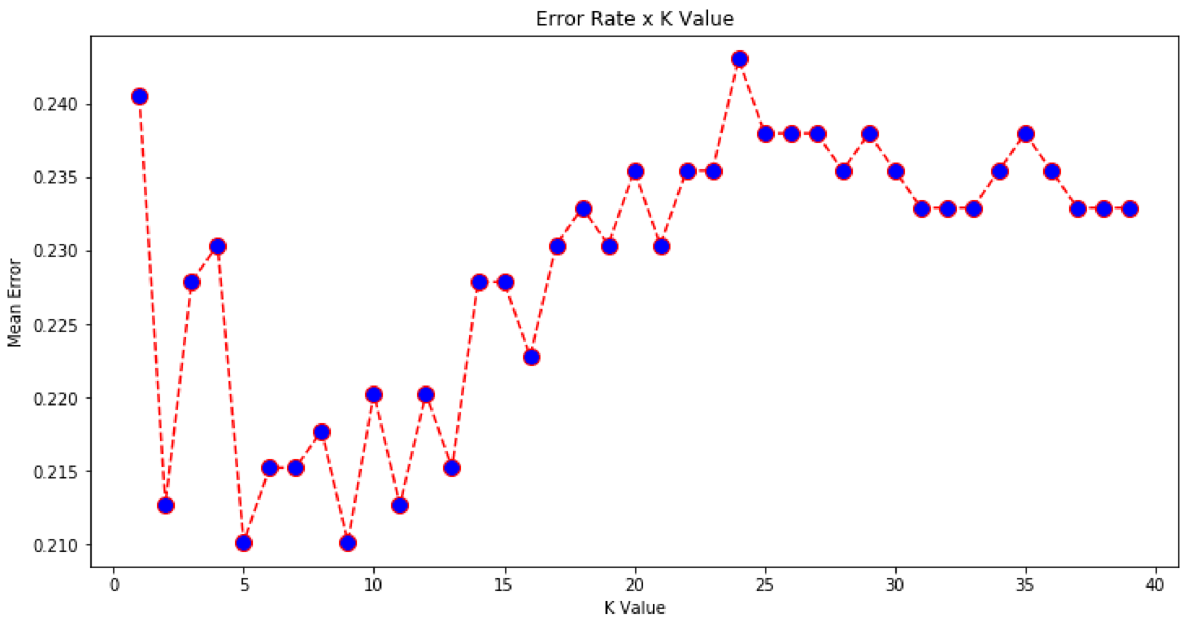
```
Accuracy: 0.789873417721519
Error: 0.21012658227848102
Precision: [0.82622951 0.66666667]
Recall: [0.89361702 0.53097345]
F1 Score: [0.85860307 0.591133  ]
```

Finding the best k-value

In [19]:
```
error = []

# Calculating error for K values between 1 and 40
for i in range(1, 40):
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train[:,[0,1,2,3]], Y_train)
    pred_i = knn.predict(X_test[:,[0,1,2,3]])
    error.append(np.mean(pred_i != Y_test))
```

In [20]:
```
plt.figure(figsize=(12, 6))
plt.plot(range(1, 40), error, color='red', linestyle='dashed', marker='o',
         markerfacecolor='blue', markersize=10)
plt.title('Error Rate x K Value')
plt.xlabel('K Value')
plt.ylabel('Mean Error')
```

Out[20]:
```
Text(0,0.5,'Mean Error')
```



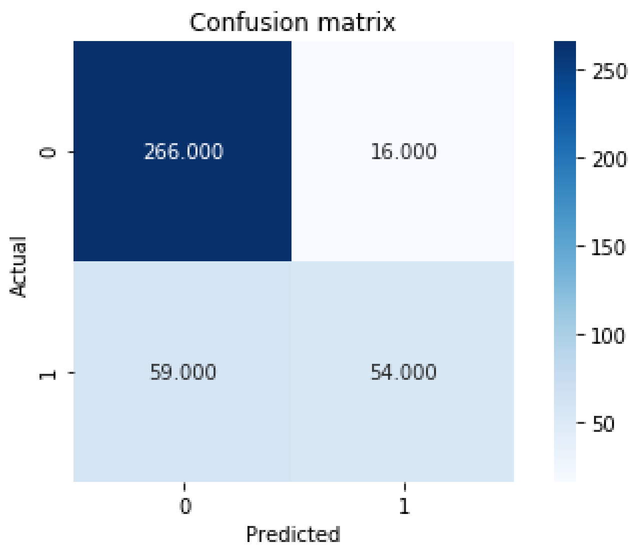
It appears from the result above that a k-value between 5 and 10 is most optimal. Based on this, we will select 5 as our k-value

Linear Support Vector Machine (SVM)

SVM Variation No. 1 (rbf)

In [21]:
```
X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent Whi
te, not Hispanic or Latino',
                                                          'Percent Black, not
 Hispanic or Latino',
                                                          'Percent Less than H
igh School Degree',
                                                          'Percent Less than B
achelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state =
1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [22]:
```
classifier = SVC(kernel = 'rbf')
classifier.fit(X_train[:,[0,1,2,3]],Y_train)

Y_pred = classifier.predict(X_test[:,[0,1,2,3]])

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



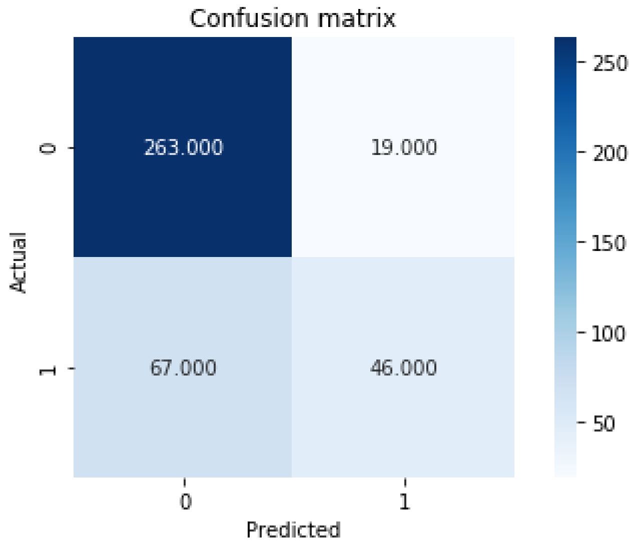
In [23]:
```
print("Accuracy: ", end=""); print(metrics.accuracy_score(Y_test, Y_pred)) # a
ccuracy
print("Error: ", end=""); print(1 - metrics.accuracy_score(Y_test, Y_pred)) #
 error
print("Precision: ", end=""); print(metrics.precision_score(Y_test, Y_pred, av
erage = None)) # precision
print("Recall: ", end=""); print(metrics.recall_score(Y_test, Y_pred, average
= None)) # recall
print("F1 Score: ", end=""); print(metrics.f1_score(Y_test, Y_pred, average =
None)) # F1 score
```

```
Accuracy: 0.810126582278481
Error: 0.189873417721519
Precision: [0.81846154 0.77142857]
Recall: [0.94326241 0.47787611]
F1 Score: [0.87644152 0.59016393]
```

SVM Variation No. 1.2 (linear)

In [24]:
```
X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent White, not Hispanic or Latino',
                                                                   'Percent Black, not Hispanic or Latino',
                                                                   'Percent Less than High School Degree',
                                                                   'Percent Less than Bachelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state = 1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = SVC(kernel = 'linear')
classifier.fit(X_train[:,[0,1,2,3]],Y_train)

Y_pred = classifier.predict(X_test[:,[0,1,2,3]])

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

print("Accuracy: ", end=""); print(metrics.accuracy_score(Y_test, Y_pred)) # accuracy
print("Error: ", end=""); print(1 - metrics.accuracy_score(Y_test, Y_pred)) # error
print("Precision: ", end=""); print(metrics.precision_score(Y_test, Y_pred, average = None)) # precision
print("Recall: ", end=""); print(metrics.recall_score(Y_test, Y_pred, average = None)) # recall
print("F1 Score: ", end=""); print(metrics.f1_score(Y_test, Y_pred, average = None)) # F1 score
```

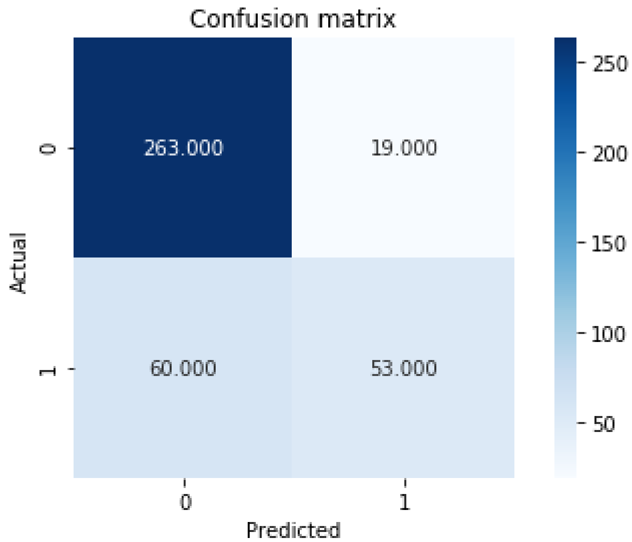
```
Accuracy: 0.7822784810126582
Error: 0.21772151898734182
Precision: [0.7969697  0.70769231]
Recall: [0.93262411 0.40707965]
F1 Score: [0.85947712 0.51685393]
```



SVM Variation No. 2 (rbf)

In [25]:
```
X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent White, not Hispanic or Latino',
                                                              'Percent Black, not Hispanic or Latino',
                                                              'Percent Age 65 and Older',
                                                              'Percent Less than Bachelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state = 1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [26]:
```
classifier = SVC(kernel = 'rbf')
classifier.fit(X_train[:,[0,1,2,3]],Y_train)

Y_pred = classifier.predict(X_test[:,[0,1,2,3]])

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



In [27]:
```
print("Accuracy: ", end=""); print(metrics.accuracy_score(Y_test, Y_pred)) # a
ccuracy
print("Error: ", end=""); print(1 - metrics.accuracy_score(Y_test, Y_pred)) #
 error
print("Precision: ", end=""); print(metrics.precision_score(Y_test, Y_pred, av
erage = None)) # precision
print("Recall: ", end=""); print(metrics.recall_score(Y_test, Y_pred, average
= None)) # recall
print("F1 Score: ", end=""); print(metrics.f1_score(Y_test, Y_pred, average =
None)) # F1 score
```

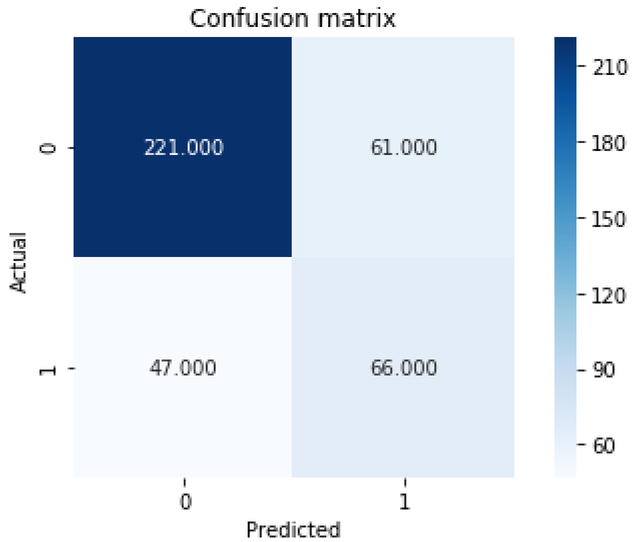
```
Accuracy: 0.8
Error: 0.19999999999999996
Precision: [0.81424149 0.73611111]
Recall: [0.93262411 0.46902655]
F1 Score: [0.86942149 0.57297297]
```

Decision Tree

Decision Tree Variation No.1

In [28]:
```
X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent Whi
te, not Hispanic or Latino',
                                                           'Percent Black, not
 Hispanic or Latino',
                                                           'Percent Less than H
igh School Degree',
                                                           'Percent Less than B
achelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state =
1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 1)
classifier.fit(X_train[:,[0,1,2,3]],Y_train)

Y_pred = classifier.predict(X_test[:,[0,1,2,3]])

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

print("Accuracy: ", end=""); print(metrics.accuracy_score(Y_test, Y_pred)) # a
ccuracy
print("Error: ", end=""); print(1 - metrics.accuracy_score(Y_test, Y_pred)) #
 error
print("Precision: ", end=""); print(metrics.precision_score(Y_test, Y_pred, av
erage = None)) # precision
print("Recall: ", end=""); print(metrics.recall_score(Y_test, Y_pred, average
= None)) # recall
print("F1 Score: ", end=""); print(metrics.f1_score(Y_test, Y_pred, average =
None)) # F1 score
```

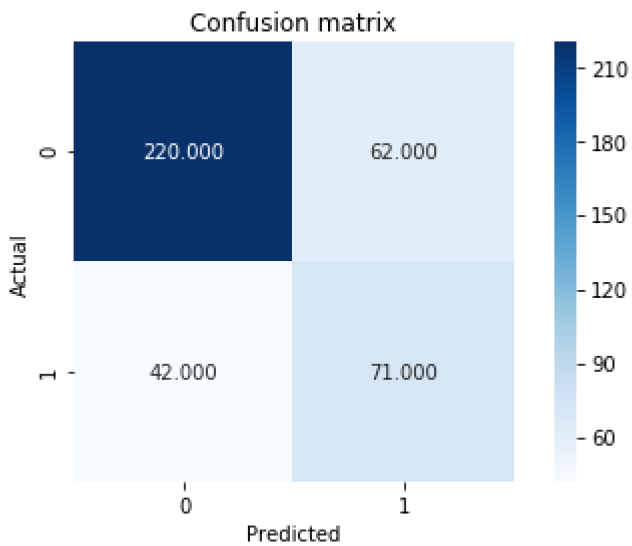
```
Accuracy: 0.7265822784810126
Error: 0.27341772151898736
Precision: [0.82462687 0.51968504]
Recall: [0.78368794 0.5840708 ]
F1 Score: [0.80363636 0.55      ]
```



In [29]:
```
### Decision Tree Variation No.2
```

In [30]:
```
X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent Whi
te, not Hispanic or Latino',
                                                          'Percent Black, not
 Hispanic or Latino',
                                                          'Percent Age 65 and
 Older',
                                                          'Percent Less than B
achelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state =
1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)

classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 1)
classifier.fit(X_train[:,[0,1,2,3]],Y_train)

Y_pred = classifier.predict(X_test[:,[0,1,2,3]])

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

print("Accuracy: ", end=""); print(metrics.accuracy_score(Y_test, Y_pred)) # a
ccuracy
print("Error: ", end=""); print(1 - metrics.accuracy_score(Y_test, Y_pred)) #
 error
print("Precision: ", end=""); print(metrics.precision_score(Y_test, Y_pred, av
erage = None)) # precision
print("Recall: ", end=""); print(metrics.recall_score(Y_test, Y_pred, average
= None)) # recall
print("F1 Score: ", end=""); print(metrics.f1_score(Y_test, Y_pred, average =
None)) # F1 score
```

```
Accuracy: 0.7367088607594937
Error: 0.2632911392405063
Precision: [0.83969466 0.53383459]
Recall: [0.78014184 0.62831858]
F1 Score: [0.80882353 0.57723577]
```



Written Responses

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

The best performing model is SVM Variation No. 1 (rbf) which takes into account having a high school degree instead of age. The performance statistics are as follows:

```
Accuracy: 0.810126582278481
Error: 0.189873417721519
Precision: [0.81846154 0.77142857]
Recall: [0.94326241 0.47787611]
F1 Score: [0.87644152 0.59016393]
```

We selected the variables based on project 1 where the following five variables had the biggest impact on being able to predict party:

```
'Percent_Age_65_and_Older'
'Percent_White_not_Hispanic_or_Latino'
'Percent_Black_not_Hispanic_or_Latino'
'Percent_Less_than_High_School_Degree'
'Percent_Less_than_Bachelor's_Degree'
```

We did some variations on these variables and selected 4 maximum to avoid overfitting or having to deal with more noise

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

Answers:

1. The best cluster model is K-Means clustering (k-means++ initialization).
2. The adjusted rand index is 0.10310966091094773 and the silhouette coefficient is 0.5514817206556075.
3. We selected the method for initialization as k-means++ to avoid poor results.
4. Besed on the variables selected from project 1, we selected 4 maximum to avoid overfitting or having to deal with more noise.

Hierarchical Clustering

In [31]:
```
merged_train.head()
```

Out[31]:

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

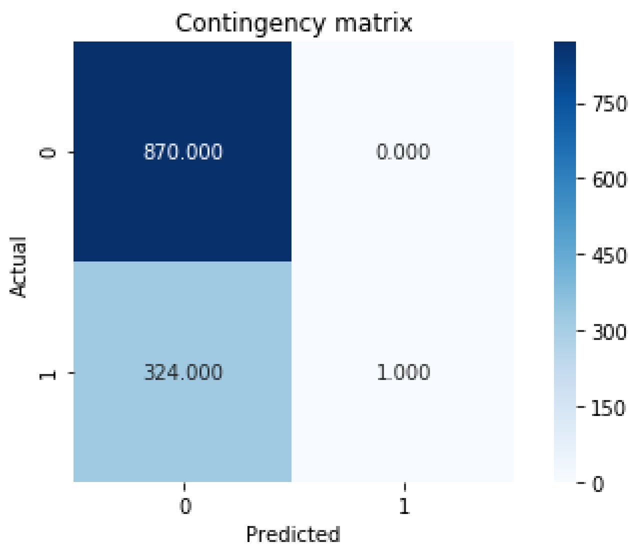
In [32]:
```
# Partition the dataset into attributes and true clusters (Democratic, Republi
can)
# Consider only the following attributes: 'Percent_Age_65_and_Older', 'Percent
_White_not_Hispanic_or_Latino', 'Percent_Black_not_Hispanic_or_Latino',
# 'Percent_Less_than_High_School_Degree', 'Percent_Less_than_Bachelor's_Degre
e'
X = merged_train[['Percent White, not Hispanic or Latino', 'Percent Black, not
Hispanic or Latino',
                  'Percent Less than High School Degree','Percent Less than Ba
chelor\'s Degree']]
Y = merged_train['Party']
```

In [33]:
```
# Standardize the attributes
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

Cluster the dataset using hierarchical clustering with single linkage method.

In [34]:
```
clustering = linkage(X, method = 'single', metric = 'euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [35]:
```
# Plot contingency matrix and compute evaluation metrics for hierarchical clus
tering with single linkage method.
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 [36]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X, clusters, metric = 'eucli
dean')
print([adjusted_rand_index, silhouette_coefficient])
```

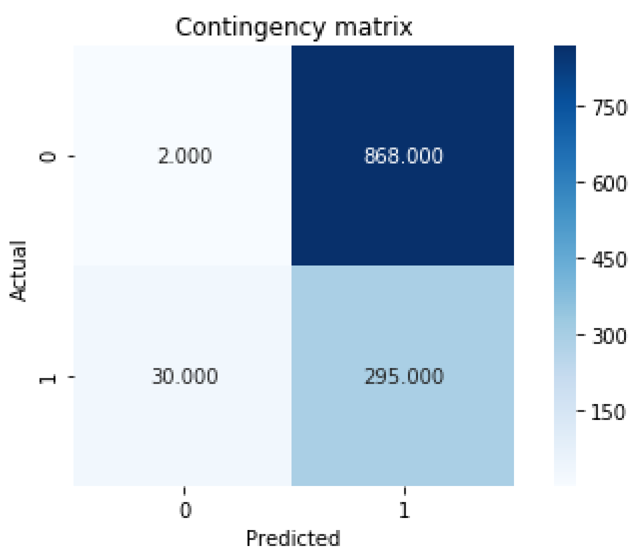
```
[0.0028041107323011935, 0.503160760061915]
```

In [37]:
```
# Plot clusters found using hierarchical clustering with single linkage method
# ax =merged_train.plot(kind = 'scatter', x = 'Percent Age 65 and Older', y =
 'Percent White, not Hispanic or Latino', c = 'clusters', colormap = plt.cm.br
g)
# ax = merged_train.plot(kind = 'scatter', x = 'Percent Age 65 and Older', y =
'Percent Black, not Hispanic or Latino', c = 'clusters', colormap = plt.cm.br
g)
# ax = merged_train.plot(kind = 'scatter', x = 'Percent Less than High School
 Degree', y = 'Percent Less than Bachelor\'s Degree', c = 'clusters', colormap
= plt.cm.brg)
```

Cluster the dataset using hierarchical clustering with complete linkage method

In [38]:
```
clustering = linkage(X, method = 'complete', metric = 'euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [39]:
```
# Plot contingency matrix and compute evaluation metrics for hierarchical clus
tering with single linkage method.
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()
```



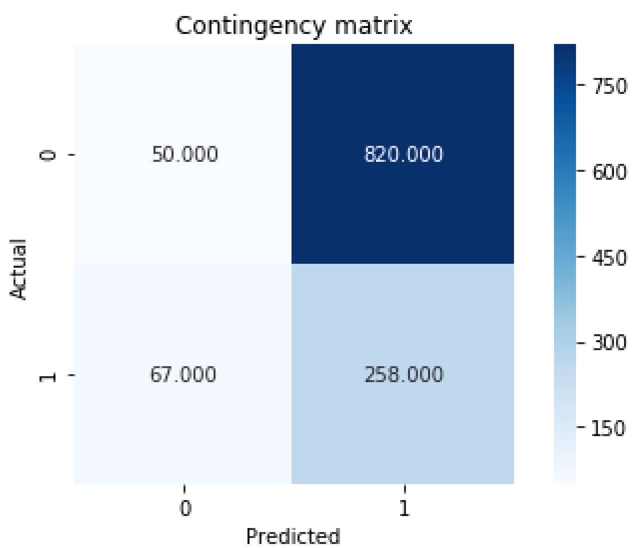
In [40]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X, clusters, metric = 'eucli
dean')
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.08180000972961025, 0.572650129712792]
```

Cluster the dataset using hierarchical clustering with average linkage method

In [41]:
```
clustering = linkage(X, method = 'average', metric = 'euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [42]:
```
# Plot contingency matrix and compute evaluation metrics for hierarchical clus
tering with single linkage method.
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 [43]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X, clusters, metric = 'eucli
dean')
print([adjusted_rand_index, silhouette_coefficient])
```

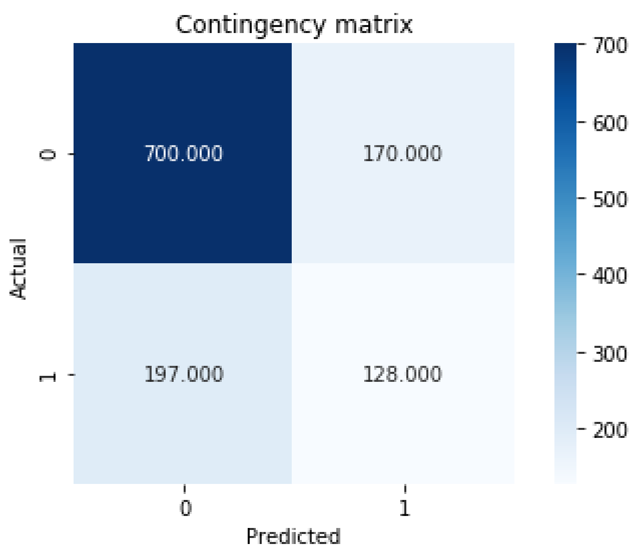
```
[0.11558483331544178, 0.568995736583541]
```

K-Means Clustering

Cluster the dataset using K-Means clustering (random initialization)

In [44]:
```
# Use random initialization of centroids, 10 iterations, and set parameter ran
dom_state to 0.
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state
=0).fit(X)
clusters = clustering.labels_
```

In [45]:
```
# Plot contingency matrix and compute evaluation metrics for K-Means clusterin
g
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()
```



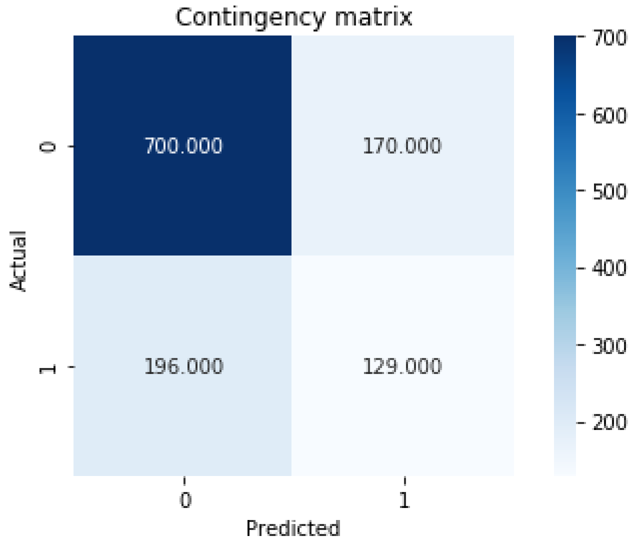
In [46]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X, clusters, metric = 'eucli
dean')
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.10141465888854553, 0.5517282800536569]
```

Cluster the dataset using K-Means clustering (k-means++ initialization)

In [47]:
```
# Use random initialization of centroids, 10 iterations, and set parameter ran
dom_state to 0.
clustering = KMeans(n_clusters = 2, init = 'k-means++', n_init = 10, random_st
ate=0).fit(X)
clusters = clustering.labels_
```

In [48]:
```
# Plot contingency matrix and compute evaluation metrics for K-Means clusterin
g
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 [49]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X, clusters, metric = 'eucli
dean')
print([adjusted_rand_index, silhouette_coefficient])
```

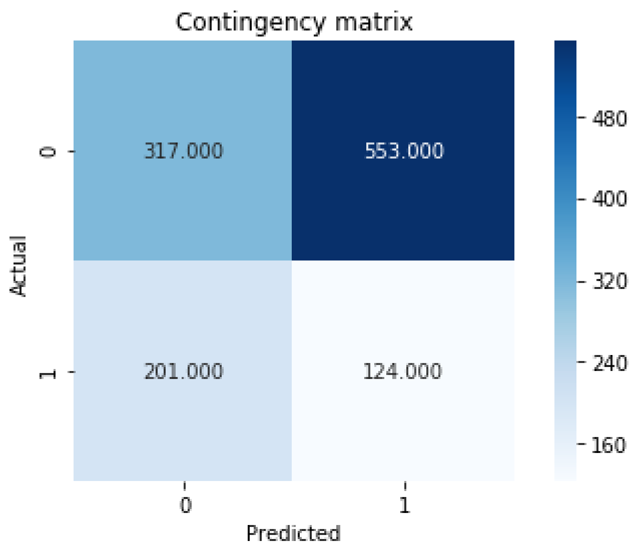
```
[0.10310966091094773, 0.5514817206556075]
```

DBSCAN

Cluster the dataset using DBSCAN

In [50]:
```
clustering = DBSCAN(eps = 4, min_samples = 12, metric = "euclidean").fit(X)
clusters = clustering.labels_
```

In [51]:
```
# Plot contingency matrix and compute evaluation metrics for DBSCAN
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 [52]:
```
adjusted_rand_index = metrics.adjusted_rand_score(Y, clusters)
silhouette_coefficient = metrics.silhouette_score(X, clusters, metric = 'eucli
dean')
print([adjusted_rand_index, silhouette_coefficient])
```

```
[0.06455044691788912, 0.4154637081222089]
```

Question 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?

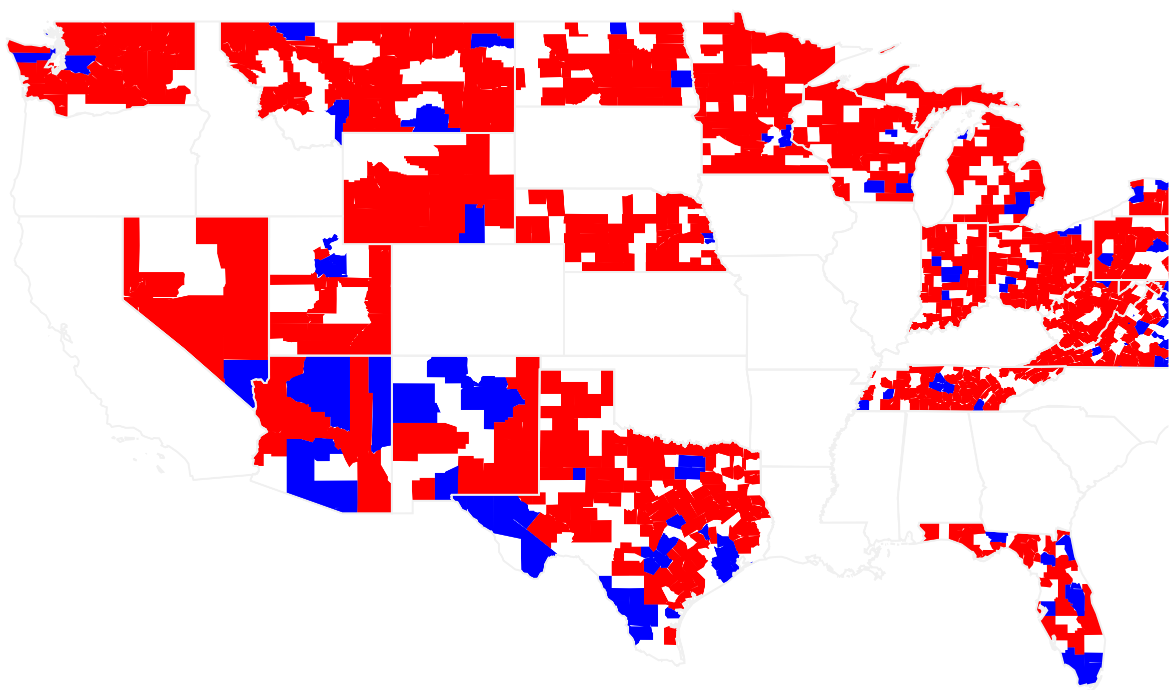
In [53]:
```
# Use SVM (brf) on mergerd_train
x_data = merged_train[['Percent White, not Hispanic or Latino', 'Percent Blac
k, not Hispanic or Latino',
              'Percent Less than High School Degree', 'Percent Less than Bache
lor\'s Degree']]
y_data = merged_train['Party']
```

In [54]:
```
# Standardize
scaler = StandardScaler()
scaler.fit(x_data)
x_data_scaled = scaler.transform(x_data)
```

In [55]:
```
best_classifier = SVC(kernel = 'rbf')
best_classifier.fit(x_data_scaled, y_data)
y_pred_new = best_classifier.predict(x_data_scaled)
y_pred_new = pd.Series(y_pred_new)
```

In [56]:
```
import plotly.figure_factory as ff
fig = ff.create_choropleth(fips=merged_train['FIPS'], values=y_pred_new, color
scale=['#ff0000', '#0000FF'])
fig.layout.template = None
fig.show()
```

```
/usr/local/Cellar/python/3.7.4_1/Frameworks/Python.framework/Versions/3.7/li
b/python3.7/site-packages/pandas/core/frame.py:6211: FutureWarning:

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

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

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



Conclusion:

1. Most Republican (red) counties are classified correctly.
2. Most north Democratic (blue) counties are miss-classified.

Question 7

(5 pts.) 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 [57]:
```
input_predict_train = pd.read_csv("demographics_test.csv")
input_predict_train = input_predict_train[['Percent White, not Hispanic or Lat
ino',
                                                          'Percent Black, not
 Hispanic or Latino',
                                                          'Percent Less than H
igh School Degree',
                                                          'Percent Less than B
achelor\'s Degree', 'Total Population']]
# display(input_predict_train.head())
```

In [58]:
```
# Partition dataset into training, validation, and test sets using holdout met
hod
x_train_dem, x_test_dem, y_train_dem, y_test_dem = train_test_split(merged_tra
in[['Percent White, not Hispanic or Latino',
                                                          'Percent Black, not
 Hispanic or Latino',
                                                          'Percent Less than H
igh School Degree',
                                                          'Percent Less than B
achelor\'s Degree', 'Total Population']],
                                                          merged_train['Democr
atic'], test_size = 0.25, random_state = 1)
x_train_rep, x_test_rep, y_train_rep, y_test_rep = train_test_split(merged_tra
in[['Percent White, not Hispanic or Latino',
                                                          'Percent Black, not
 Hispanic or Latino',
                                                          'Percent Less than H
igh School Degree',
                                                          'Percent Less than B
achelor\'s Degree', 'Total Population']],
                                                          merged_train['Republ
ican'], test_size = 0.25, random_state = 1)

scaler = StandardScaler()
scaler.fit(x_train_dem)
x_train_dem_scaled = scaler.transform(x_train_dem)
x_test_dem_scaled = scaler.transform(x_test_dem)

scaler.fit(x_train_rep)
x_train_rep_scaled = scaler.transform(x_train_rep)
x_test_rep_scaled = scaler.transform(x_test_rep)

scaler.fit(input_predict_train)
input_scaled = scaler.transform(input_predict_train)
```

In [59]:
```
#DEM Regression
model = linear_model.Lasso(alpha=1)
fitted_model = model.fit(X = x_train_dem_scaled[:,[0,1,2,3,4]], y = y_train_de
m)
predicted_dem = fitted_model.predict(input_scaled[:,[0,1,2,3,4]])
```

In [60]:
```
#Republican Regression
model = linear_model.Lasso()
fitted_model = model.fit(X = x_train_rep_scaled[:,[0,1,2,3,4]], y = y_train_re
p)
predicted_rep = fitted_model.predict(input_scaled[:,[0,1,2,3,4]])
```

In [61]:
```
output = pd.read_csv("demographics_test.csv")
output = output[['State','County']]
```

In [62]:
```
input_predict_train = pd.read_csv("demographics_test.csv")
input_predict_train = input_predict_train[['Percent White, not Hispanic or Lat
ino',
                                                           'Percent Black, not
 Hispanic or Latino',
                                                           'Percent Less than H
igh School Degree',
                                                           'Percent Less than B
achelor\'s Degree']]

X_train, X_test, Y_train, Y_test = train_test_split(merged_train[['Percent Whi
te, not Hispanic or Latino',
                                                           'Percent Black, not
 Hispanic or Latino',
                                                           'Percent Less than H
igh School Degree',
                                                           'Percent Less than B
achelor\'s Degree']], merged_train['Party'], test_size = 0.33, random_state =
1)
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
input_scaled = scaler.transform(input_predict_train)

classifier = SVC(kernel = 'rbf')
classifier.fit(X_train[:,[0,1,2,3]],Y_train)

Y_pred = classifier.predict(input_scaled[:,[0,1,2,3]])
# display(Y_pred)

output['Republican'] = predicted_rep
output['Democrat'] = predicted_dem
output['Republican'] = output['Republican'].round(0).astype(int)
output['Democrat'] = output['Democrat'].round(0).astype(int)
output['Party'] = Y_pred
num = output._get_numeric_data()
num[num < 0] = 0
display(output.head())
```

	State	County	Republican	Democrat	Party
0	NV	eureka	2006	0	0
1	TX	zavala	0	0	1
2	VA	king george	9050	12590	1
3	OH	hamilton	145169	252760	1
4	TX	austin	4181	1641	0

In [63]:
```
output.to_csv('./project2_output.csv', index = None, header=True)
```