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

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

In [3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1195 entries, 0 to 1194
Data columns (total 19 columns):
 #   Column                                Non-Null Count  Dtype  
---  ------                                --------------  -----  
 0   State                                 1195 non-null   object 
 1   County                                1195 non-null   object 
 2   FIPS                                  1195 non-null   int64  
 3   Total Population                      1195 non-null   int64  
 4   Percent White, not Hispanic or Latino 1195 non-null   float64
 5   Percent Black, not Hispanic or Latino 1195 non-null   float64
 6   Percent Hispanic or Latino            1195 non-null   float64
 7   Percent Foreign Born                  1195 non-null   float64
 8   Percent Female                        1195 non-null   float64
 9   Percent Age 29 and Under              1195 non-null   float64
 10  Percent Age 65 and Older              1195 non-null   float64
 11  Median Household Income               1195 non-null   int64  
 12  Percent Unemployed                    1195 non-null   float64
 13  Percent Less than High School Degree  1195 non-null   float64
 14  Percent Less than Bachelor's Degree   1195 non-null   float64
 15  Percent Rural                         1195 non-null   float64
 16  Democratic                            1195 non-null   int64  
 17  Republican                            1195 non-null   int64  
 18  Party                                 1195 non-null   int64  
dtypes: float64(11), int64(6), object(2)
memory usage: 177.5+ KB
```

In [4]:
```python
Y = pd.DataFrame(data['Democratic'])
Y['Republican'] = data['Republican']
Y['Party']=data['Party']
Y['Total Population']=data['Total Population']
Y['FIPS']= data['FIPS']
State = pd.DataFrame(data['State'])
State['County'] = data['County']
data_x = data.drop(['Democratic','Republican','State','County','Party'], axis=1)
data_x.head()
```

Out[4]:

	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	Percent Less than High School Degree
0	4001	72346	18.571863	0.486551	5.947806	1.719515	50.598513	45.854643	13.322091	32460	15.807433	21.758252
1	4003	128177	56.299492	3.714395	34.403208	11.458374	49.069646	37.902276	19.756275	45383	8.567108	13.409171
2	4005	138064	54.619597	1.342855	13.711033	4.825298	50.581614	48.946141	10.873943	51106	8.238305	11.085381
3	4007	53179	63.222325	0.552850	18.548675	4.249798	50.296170	32.238290	26.397638	40593	12.129932	15.729958
4	4009	37529	51.461536	1.811932	32.097844	4.385942	46.313518	46.393456	12.315809	47422	14.424104	14.580797

1. (5 pts.) 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 [5]:
```python
x_train, x_test, y_train, y_test = train_test_split(data_x, Y, test_size=0.2, random_state=0)
```

2. (5 pts.) Standardize the training set and the validation set.

In [6]:
```
scaler = StandardScaler()
scaler.fit(x_train)
data_x_scaled = scaler.transform(data_x)
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)
```

3. (25 pts.) 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.

1: use "Total Population" to predict Democratic vote

In [7]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[:,[1]],y_train['Democratic'])
predicted = fitted_model.predict(x_test_scaled[:,[1]])
print(fitted_model.coef_)
corr_coef = np.corrcoef(predicted,y_test['Democratic'])[0,1]
R_squared = corr_coef*corr_coef
print('Model built by "Total Population" R_squared:', R_squared)
```

```
[72934.95317077]
Model built by "Total Population" R_squared: 0.9384743656014513
```

2: use "Total Population","Percent White","Percent Female", to predict Democratic vote

In [8]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[:,[1,2,6]],y_train['Democratic'])
predicted = fitted_model.predict(x_test_scaled[:,[1,2,6]])
corr_coef = np.corrcoef(predicted,y_test['Democratic'])[0,1]
R_squared = corr_coef*corr_coef
print('Model built by "Total Population","Percent White","Percent Female" R_squared:', R_squared)
```

```
Model built by "Total Population","Percent White","Percent Female" R_squared: 0.9378707752394356
```

3: use "Total Population","Median Household Income","Percent Unemployed" to predict Democratic vote

In [9]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[:,[1,9,10]],y_train['Democratic'])
predicted = fitted_model.predict(x_test_scaled[:,[1,9,10]])
corr_coef = np.corrcoef(predicted,y_test['Democratic'])[0,1]
R_squared = corr_coef*corr_coef
print('Model built by "Total Population","Median Household Income" R_squared:', R_squared)
```

```
Model built by "Total Population","Median Household Income" R_squared: 0.9353517973406513
```

4: use LASSO to predict Democratic vote

In [10]:
```
model = linear_model.Lasso(alpha = 3000)
fitted_model = model.fit(x_train_scaled,y_train['Democratic'])
predicted = fitted_model.predict(x_test_scaled)
corr_coef = np.corrcoef(predicted,y_test['Democratic'])[0,1]
R_squared = corr_coef*corr_coef
print('Model built by LASSO, R_squared:', R_squared)
```

```
Model built by LASSO, R_squared: 0.9510326635205784
```

result:

```
R_squared: 0.9384, model built by "Total Population"
R_squared: 0.9378, model built by "Total Population","Percent White","Percent Female"
R_squared: 0.9353, model built by "Total Population","Median Household Income","Percent Unemployed"
R_squared: 0.9510, model built by LASSO based on "Total Population", "Percent Foreign Born", "Percent Less than
Bachelor's Degree"
```

LASSO has the best perfermance, 95.1% of the variability in the number of votes cast for the Democratic party can be explained by "Total Population", "Percent Foreign Born", "Percent Less than Bachelor's Degree"

1: use "Total Population" to predict Republican vote

In [11]:

```
model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[:,[1]],y_train['Republican'])
predicted = fitted_model.predict(x_test_scaled[:,[1]])
print(fitted_model.coef_)
corr_coef = np.corrcoef(predicted,y_test['Republican'])[0,1]
R_squared = corr_coef*corr_coef
adjusted_r_squared = 1 - (1-R_squared)*((np.shape(x_train_scaled)[0]-1)/(np.shape(x_train_scaled)[0]-np.shape
(x_train_scaled)[1]-1))
print('Model built by "Total Population" R_squared:', R_squared,' adjusted_R_squared:', adjusted_r_squared)
```

```
[44250.84661286]
Model built by "Total Population" R_squared: 0.6349943253258931  adjusted_R_squared: 0.629563847700561
```

2: use "Total Population","Percent White","Percent Female", to predict Republican vote

In [12]:

```
model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[:,[1,2,6]],y_train['Republican'])
predicted = fitted_model.predict(x_test_scaled[:,[1,2,6]])
print(fitted_model.coef_)
corr_coef = np.corrcoef(predicted,y_test['Republican'])[0,1]
R_squared = corr_coef*corr_coef
adjusted_r_squared = 1 - (1-R_squared)*((np.shape(x_train_scaled)[0]-1)/(np.shape(x_train_scaled)[0]-np.shape
(x_train_scaled)[1]-1))
print('Model built by "Total Population","Percent White","Percent Female" R_squared:', R_squared, ' adjusted_
R_squared:', adjusted_r_squared)
```

```
[45169.2736523   3218.95229927   768.06744922]
Model built by "Total Population","Percent White","Percent Female" R_squared: 0.6386161888738883  adjusted_R_
squared: 0.6332395965723309
```

3: use "Total Population","Median Household Income",Percent Unemployed to predict Republican vote

In [13]:
```
model = linear_model.LinearRegression()
fitted_model = model.fit(x_train_scaled[:,[1,9,10]],y_train['Republican'])
predicted = fitted_model.predict(x_test_scaled[:,[1,9,10]])
print(fitted_model.coef_)
corr_coef = np.corrcoef(predicted,y_test['Republican'])[0,1]
R_squared = corr_coef*corr_coef
adjusted_r_squared = 1 - (1-R_squared)*((np.shape(x_train_scaled)[0]-1)/(np.shape(x_train_scaled)[0]-np.shape
(x_train_scaled)[1]-1))
print('Model built by "Total Population","Median Household Income" R_squared:', R_squared, ' adjusted_R_squar
ed:', adjusted_r_squared)
```

```
[42748.35927231  5348.60430634  1397.73223414]
Model built by "Total Population","Median Household Income" R_squared: 0.6507388828990277  adjusted_R_square
d: 0.6455426494883862
```

4: use LASSO to predict Republican vote with alpha = 1000

In [14]:
```
model = linear_model.Lasso(alpha = 1000)
fitted_model = model.fit(x_train_scaled,y_train['Republican'])
predicted = fitted_model.predict(x_test_scaled)
print(fitted_model.coef_)
corr_coef = np.corrcoef(predicted,y_test['Republican'])[0,1]
R_squared = corr_coef*corr_coef
adjusted_r_squared = 1 - (1-R_squared)*((np.shape(x_train_scaled)[0]-1)/(np.shape(x_train_scaled)[0]-np.shape
(x_train_scaled)[1]-1))
print('Model built by LASSO, R_squared:', R_squared, ' adjusted_R_squared:', adjusted_r_squared)
```

```
[ -786.34645669 41939.86078642  1096.16644865  -995.95196875
    -0.          -793.69923536     0.          -544.81042106
     0.          2763.46840938     0.            -0.
  -896.33313183 -2808.73152291]
Model built by LASSO, R_squared: 0.6745864784750144  adjusted_R_squared: 0.6697450445734737
```

5: use LASSO to predict Republican vote with alpha = 100

In [15]:
```
model = linear_model.Lasso(alpha = 100)
fitted_model = model.fit(x_train_scaled,y_train['Republican'])
predicted = fitted_model.predict(x_test_scaled)
print(fitted_model.coef_)
corr_coef = np.corrcoef(predicted,y_test['Republican'])[0,1]
R_squared = corr_coef*corr_coef
adjusted_r_squared = 1 - (1-R_squared)*((np.shape(x_train_scaled)[0]-1)/(np.shape(x_train_scaled)[0]-np.shape
(x_train_scaled)[1]-1))
print('Model built by LASSO, R_squared:', R_squared, ' adjusted_R_squared:', adjusted_r_squared)
```

```
[-1528.57203543 43919.2438147   1331.75509196 -2705.44303994
   885.31160473 -5718.69434818  -675.76725505  -952.86386395
  1971.73733439  5507.99285348  1617.69036087  3289.04347235
 -2793.16229115 -5229.19260998]
Model built by LASSO, R_squared: 0.7004672813642709  adjusted_R_squared: 0.6960108966024217
```

result:

```
adjusted_R_squared: 0.6295, model built by "Total Population"
adjusted_R_squared: 0.6386, model built by "Total Population","Percent White","Percent Female"
adjusted_R_squared: 0.6455, model built by "Total Population","Median Household Income","Percent Unemployed"
adjusted_R_squared: 0.6697, model built by LASSO based on 9 features
adjusted_R_squared: 0.6960, model built by LASSO based on all features
```

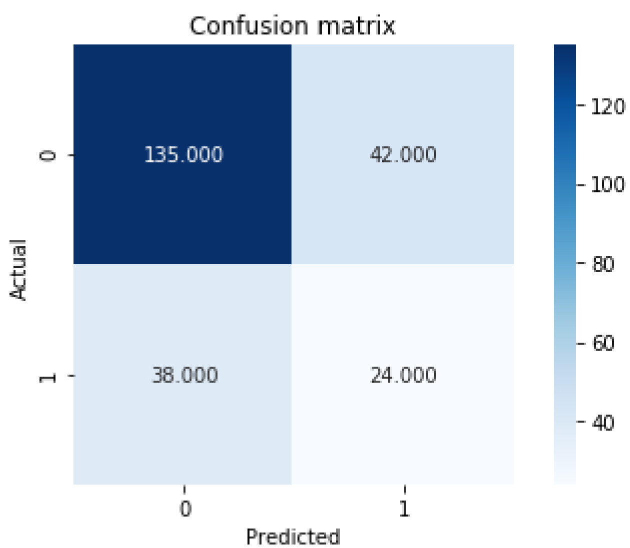
LASSO has the best perfermance, 69.6% of the variability in the number of votes cast for the Republican party can be explained by all features.

4. (25 pts.) 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?

1: use "Total Population" to predict Party with DecisionTree

In [16]:
```python
classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
classifier.fit(x_train_scaled[:,[1]],y_train['Party'])
y_pred = classifier.predict(x_test_scaled[:,[1]])
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(conf_matrix)
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

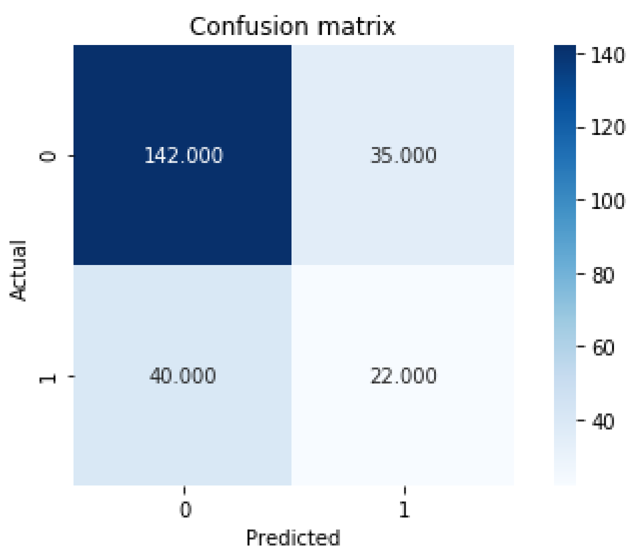
```
[[135  42]
 [ 38  24]]
accuracy is  0.6652719665271967
error is  0.33472803347280333
precision is  [0.78034682 0.36363636]
recall is  [0.76271186 0.38709677]
F1_score is  [0.77142857 0.375     ]
```



2: use "Total Population","Median Household Income" to predict Party with DecisionTree

In [17]:
```python
classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
classifier.fit(x_train_scaled[:,[1,9]],y_train['Party'])
y_pred = classifier.predict(x_test_scaled[:,[1,9]])
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(conf_matrix)
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

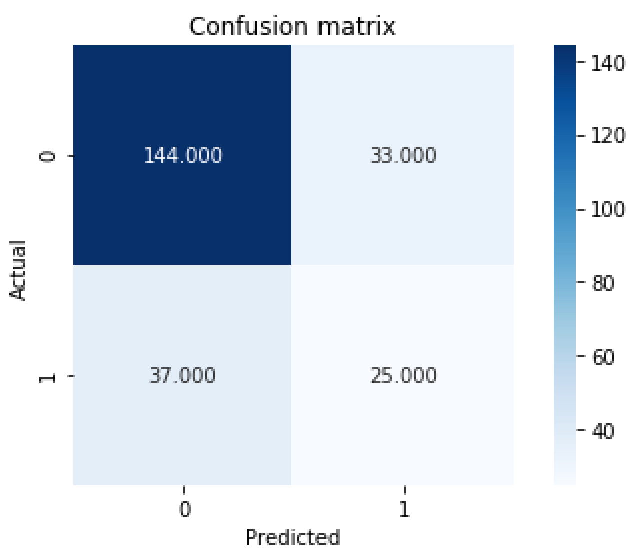
```
[[142  35]
 [ 40  22]]
accuracy is  0.6861924686192469
error is  0.3138075313807531
precision is  [0.78021978 0.38596491]
recall is  [0.80225989 0.35483871]
F1_score is  [0.79108635 0.3697479 ]
```



3: use "Total Population","Median Household Income","Percent White, not Hispanic or Latino" to predict Party with DecisionTree

In [18]:
```
classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
classifier.fit(x_train_scaled[:,[1,2,9]],y_train['Party'])
y_pred = classifier.predict(x_test_scaled[:,[1,2,9]])
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(conf_matrix)
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

```
[[144  33]
 [ 37  25]]
accuracy is  0.7071129707112971
error is  0.2928870292887029
precision is  [0.79558011 0.43103448]
recall is  [0.81355932 0.40322581]
F1_score is  [0.80446927 0.41666667]
```

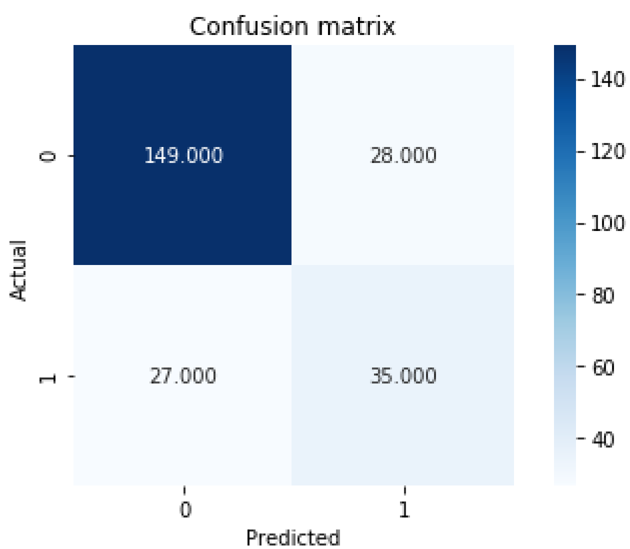


4: use all features to predict Party with DecisionTree

In [19]:

```
classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
classifier.fit(x_train_scaled,y_train['Party'])
y_pred = classifier.predict(x_test_scaled)
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(conf_matrix)
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

```
[[149  28]
 [ 27  35]]
accuracy is  0.7698744769874477
error is  0.2301255230125523
precision is  [0.84659091 0.55555556]
recall is  [0.84180791 0.56451613]
F1_score is  [0.84419263 0.56      ]
```

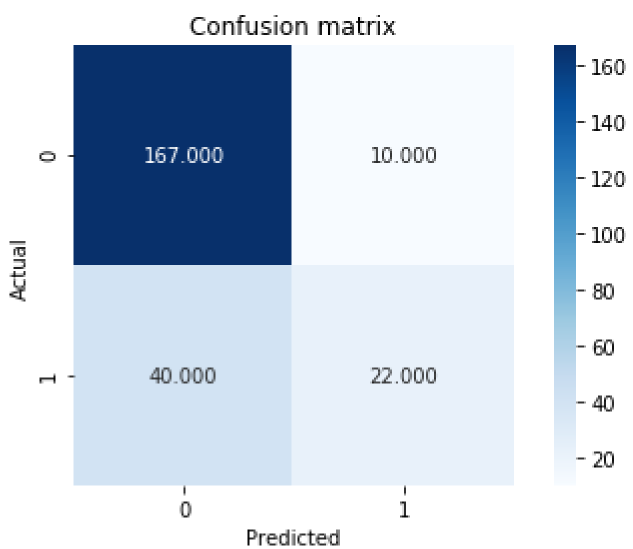


5: use "Total Population" to predict Party with SVC

In [20]:

```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled[:,[1,2,6]],y_train['Party'])
y_pred = classifier.predict(x_test_scaled[:,[1,2,6]])
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
print(conf_matrix)
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

```
[[167  10]
 [ 40  22]]
accuracy is  0.7907949790794979
error is  0.20920502092050208
precision is  [0.80676329 0.6875    ]
recall is  [0.94350282 0.35483871]
F1_score is  [0.86979167 0.46808511]
```

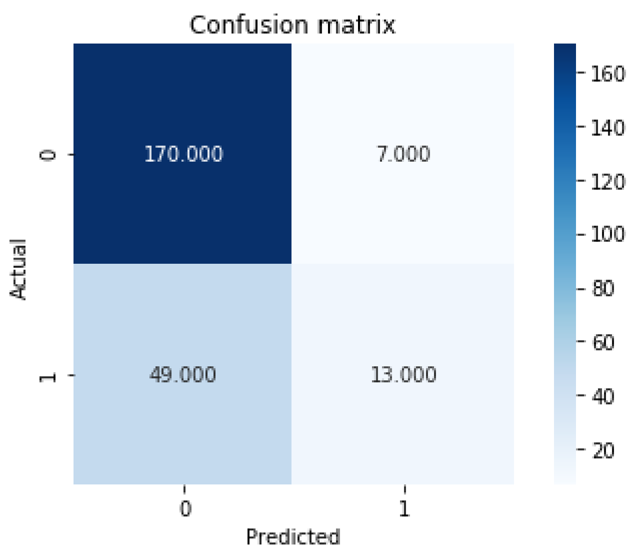


6: use "Total Population","Median Household Income" to predict party with SVC

In [21]:

```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled[:,[1,9]],y_train['Party'])
y_pred = classifier.predict(x_test_scaled[:,[1,9]])
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

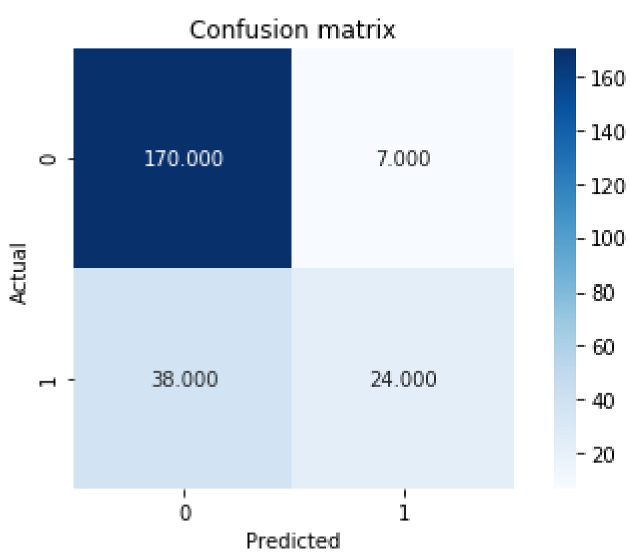
```
accuracy is  0.7656903765690377
error is  0.2343096234309623
precision is  [0.77625571 0.65      ]
recall is  [0.96045198 0.20967742]
F1_score is  [0.85858586 0.31707317]
```



7: use "Total Population","Median Household Income","Percent White, not Hispanic or Latino" to predict Party with SVC

In [22]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled[:,[1,2,9]],y_train['Party'])
y_pred = classifier.predict(x_test_scaled[:,[1,2,9]])
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

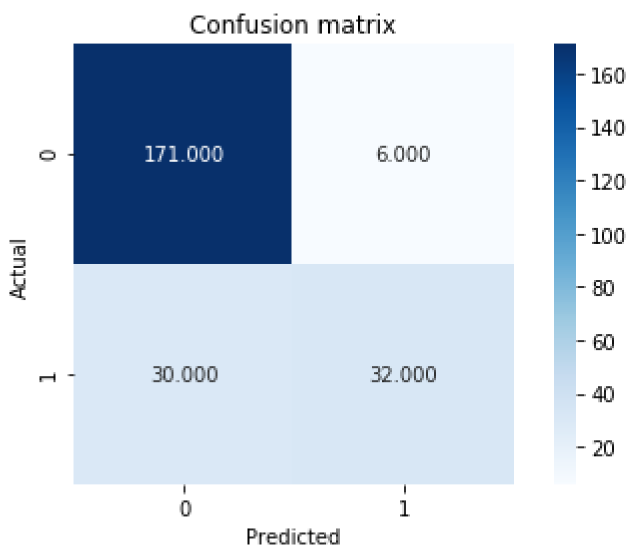
```
accuracy is  0.8117154811715481
error is  0.18828451882845187
precision is  [0.81730769 0.77419355]
recall is  [0.96045198 0.38709677]
F1_score is  [0.88311688 0.51612903]
```



8: use all features to predict Party with SVC

In [23]:
```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled,y_train['Party'])
y_pred = classifier.predict(x_test_scaled)
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

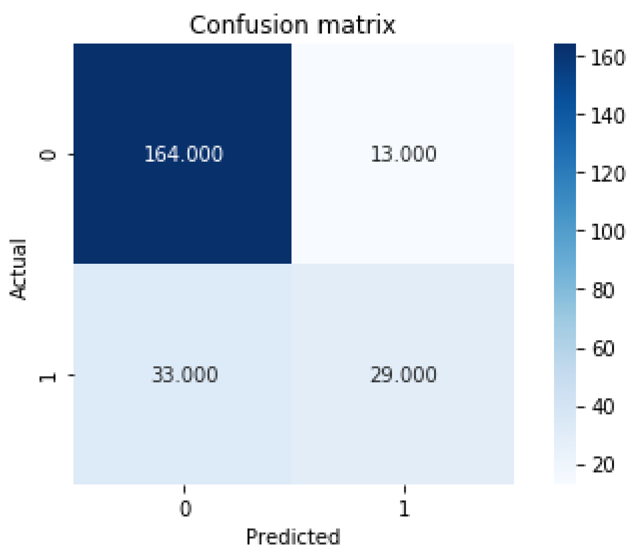
```
accuracy is  0.8493723849372385
error is  0.15062761506276146
precision is  [0.85074627 0.84210526]
recall is  [0.96610169 0.51612903]
F1_score is  [0.9047619 0.64      ]
```



9: use all features to predict Democratic vote with KNeighbors and k=3

In [24]:
```
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train_scaled,y_train['Party'])
y_pred = classifier.predict(x_test_scaled)
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

```
accuracy is  0.8075313807531381
error is  0.19246861924686187
precision is  [0.83248731 0.69047619]
recall is  [0.92655367 0.46774194]
F1_score is  [0.87700535 0.55769231]
```

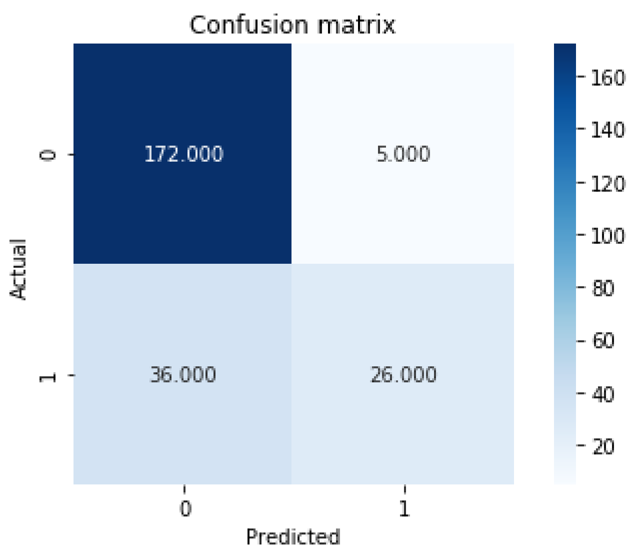


10: use all features to predict Democratic vote with KNeighbors and k=6

In [25]:

```
classifier = KNeighborsClassifier(n_neighbors=6)
classifier.fit(x_train_scaled,y_train['Party'])
y_pred = classifier.predict(x_test_scaled)
conf_matrix = metrics.confusion_matrix(y_test['Party'],y_pred)
sns.heatmap(conf_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Confusion matrix')
plt.tight_layout()
accuracy = metrics.accuracy_score(y_test['Party'], y_pred)
error = 1 - accuracy
precision = metrics.precision_score(y_test['Party'], y_pred, average = None)
recall = metrics.recall_score(y_test['Party'], y_pred, average = None)
F1_score = metrics.f1_score(y_test['Party'], y_pred, average = None)
print('accuracy is ',accuracy,'\nerror is ' ,error,'\nprecision is ',precision,'\nrecall is ',recall,'\nF1_sc
ore is ',F1_score)
```

```
accuracy is  0.8284518828451883
error is  0.17154811715481166
precision is  [0.82692308 0.83870968]
recall is  [0.97175141 0.41935484]
F1_score is  [0.89350649 0.55913978]
```



result

```
accuracy: 0.6652, F1_score: [0.771 0.375] use decision tree with "Total Population" to predict Party
accuracy: 0.6861, F1_score: [0.791 0.369] use decision tree with "Total Population","Median Household Income" to
predict Party
accuracy: 0.7071, F1_score: [0.804 0.416] use decision tree with "Total Population","Median Household Income","Per
cent                                       White" to predict Party
accuracy: 0.7698, F1_score: [0.844 0.560] use decision tree with all features to predict Party
accuracy: 0.7907, F1_score: [0.869 0.468] use SVC with "Total Population" to predict Party
accuracy: 0.7656, F1_score: [0.858 0.317] use SVC with "Total Population","Median Household Income" to predict Par
ty
accuracy: 0.8117, F1_score: [0.883 0.516] use SVC with "Total Population","Median Household Income","Percent
White" to predict Party
accuracy: 0.8493, F1_score: [0.904 0.640] use SVC with all features to predict Party
accuracy: 0.8075, F1_score: [0.877 0.557] use KNeighbors k=3 with all features to predict Party
accuracy: 0.8284, F1_score: [0.893 0.559] use KNeighbors k=6 with all features to predict Party

model built by SVC with all features has the best perfermance,use radial basis function kernel as parameter
```

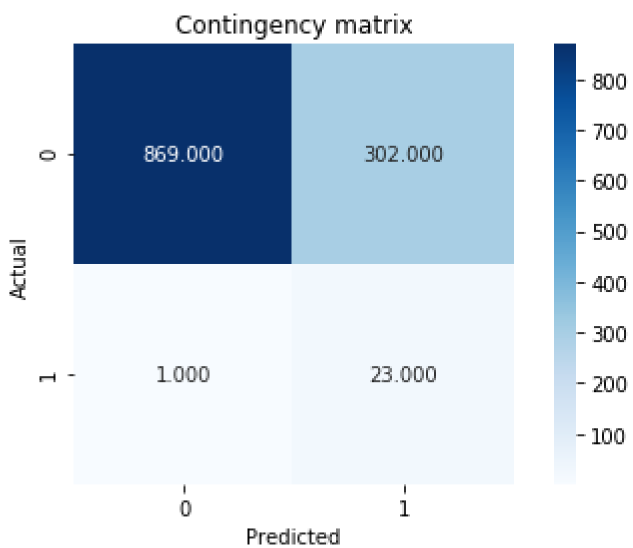
5. (25 pts.) Build a clustering model to cluster the counties. Consider at least two different clustering techniques with multiple combinations of parameters and multiple combinations of variables. Compute unsupervised and supervised evaluation metrics for the validation set with the party of the counties (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?

In [26]:
```
data_x_scaled2 = np.delete(data_x_scaled, [0,3,6], 1)
```

1.1 : use "Total Population", to cluster Party based on single linkage method

In [27]:
```
clustering = linkage(data_x_scaled2[:,[1]], method='single', metric='euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

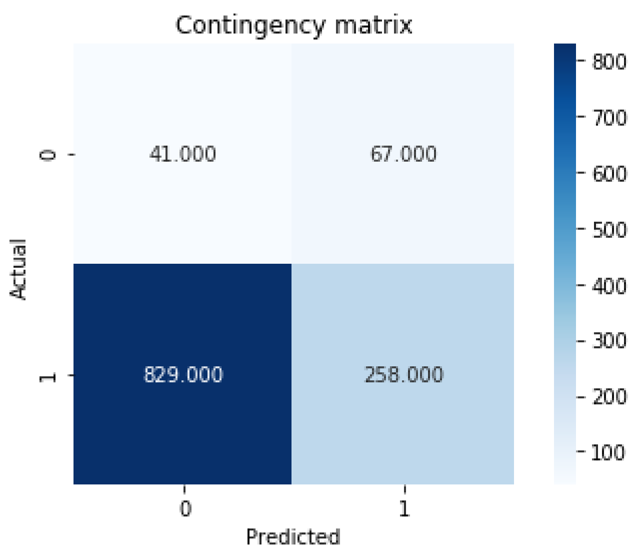
```
[0.06341778206546299, 0.6589687284593577]
```



1.2 : use "Total Population" to cluster Party based on complete linkage method

In [28]:
```
clustering = linkage(data_x_scaled2[:,[1]], method='complete', metric='euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

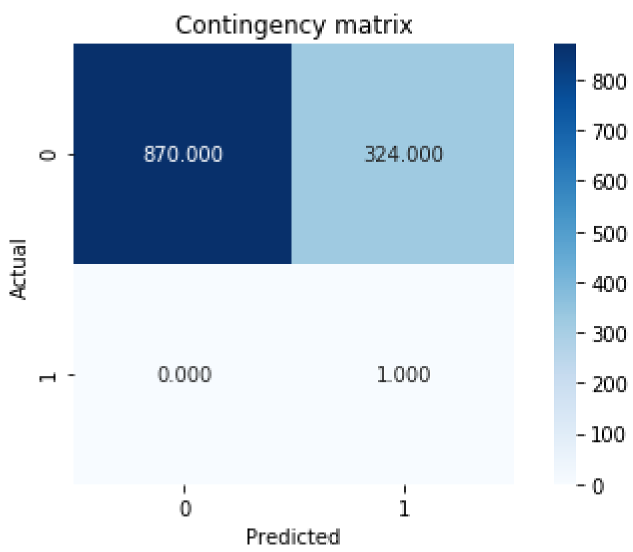
```
[0.1276047164044394, 0.6688371001371756]
```



1.3 : use "Total Population","Median Household Income" to cluster Party based on single linkage method

In [29]:
```
clustering = linkage(data_x_scaled2[:,[1,9]], method='single', metric='euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,9]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

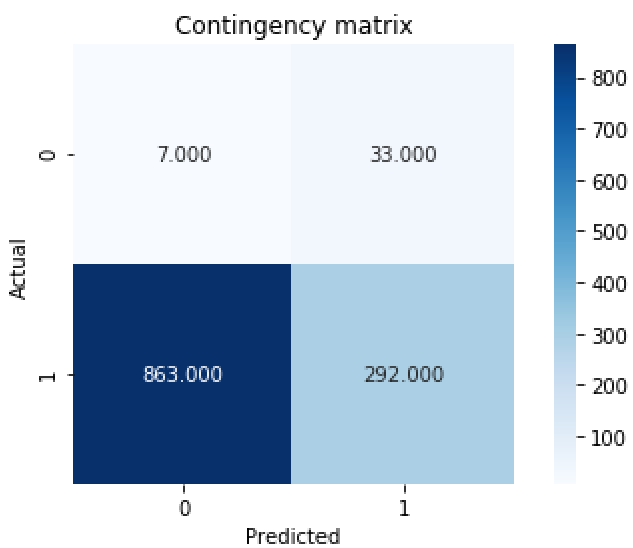
```
[0.0028041107323011935, 0.6910517473842601]
```



1.4 : use "Total Population","Median Household Income" to cluster Party based on complete linkage method

In [30]:
```
clustering = linkage(data_x_scaled2[:,[1,9]], method='complete', metric='euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,9]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

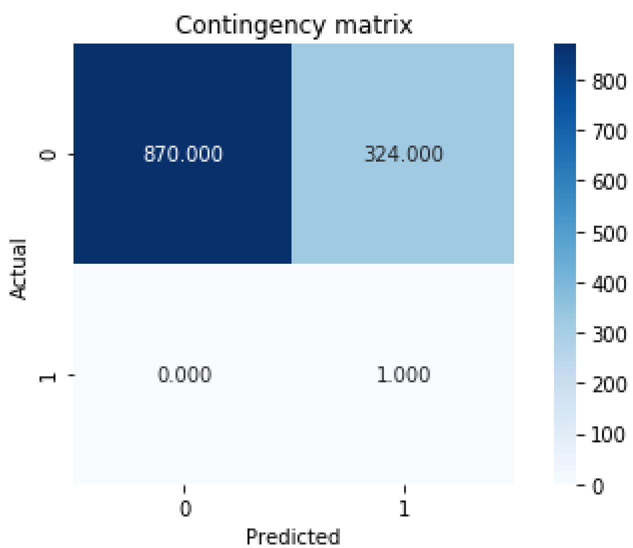
```
[0.08355255182167579, 0.5250596376301193]
```



1.5 : use "Total Population","Median Household Income", "percent of white" to cluster Party based on single linkage method

```
In [31]: clustering = linkage(data_x_scaled2[:,[1,2,9]], method='single', metric='euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,2,9]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

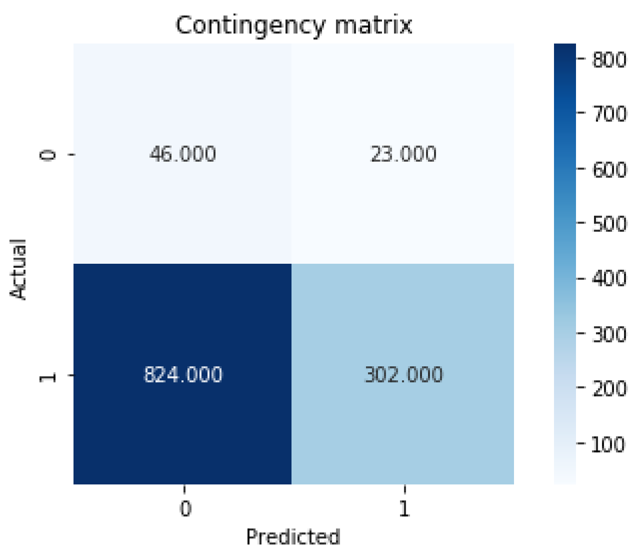
```
[0.0028041107323011935, 0.6408749430786518]
```



1.6 : use "Total Population","Median Household Income", "percent of white" to cluster Party based on complete linkage method

In [32]:
```
clustering = linkage(data_x_scaled2[:,[1,2,9]], method='complete', metric='euclidean')
clusters = fcluster(clustering, 2, criterion = 'maxclust')
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,2,9]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
```

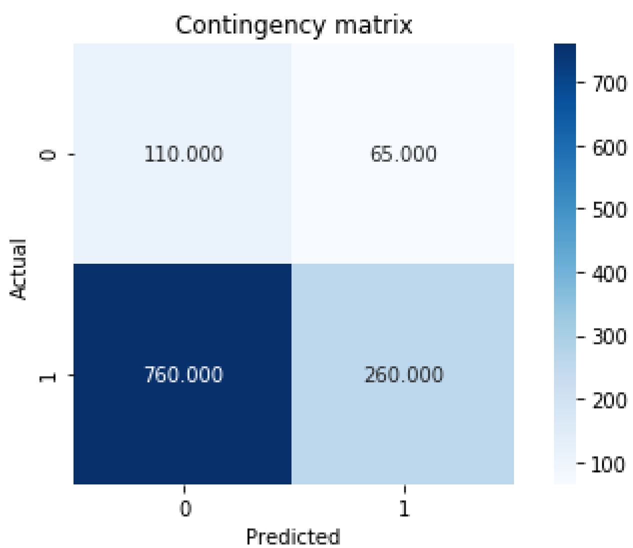
```
[0.013727521151231936, 0.6121123278017568]
```



2.1 : use "Total Population","Median Household Income", "percent of white" to cluster Party based on kmeans method iteration = 10

In [33]:
```
clustering = KMeans(n_clusters=2,init='random',max_iter=10,random_state=0).fit(data_x_scaled2[:,[1,2,9]])
clusters = clustering.labels_
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,2,9]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
```

```
[0.04537501451738291, 0.5521120315698114]
```

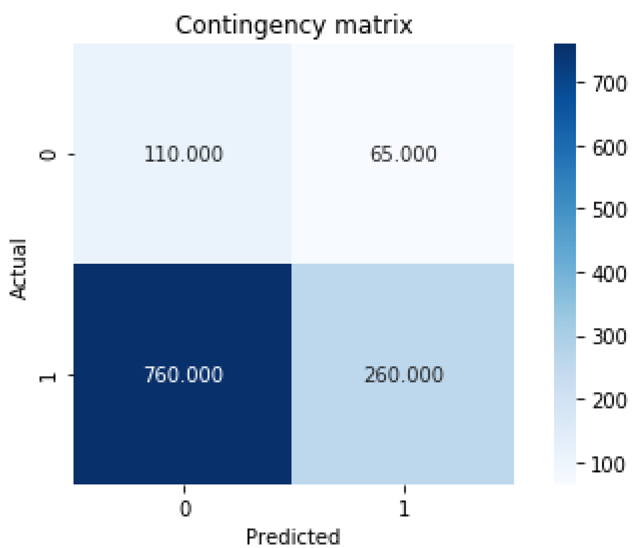


2.2 : use "Total Population","Median Household Income","percent of white" to cluster Party based on kmeans method iteration = 100

In [34]:

```
clustering = KMeans(n_clusters=2,init='random',max_iter=100,random_state=0).fit(data_x_scaled2[:,[1,9,2]])
clusters = clustering.labels_
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,9,2]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
```

```
[0.04537501451738291, 0.5521120315698114]
```

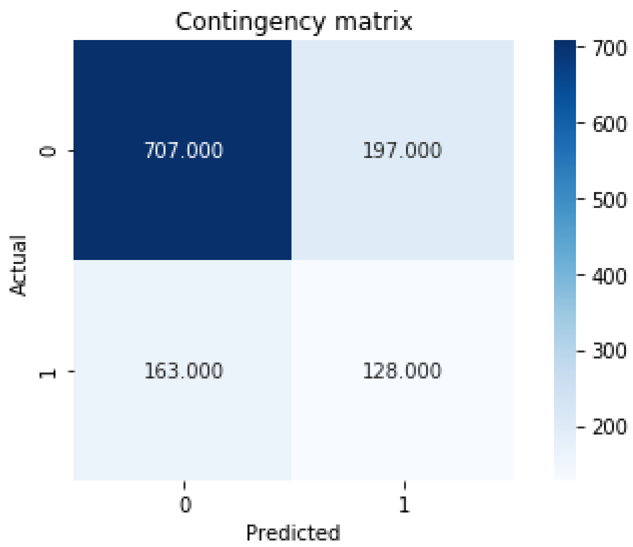


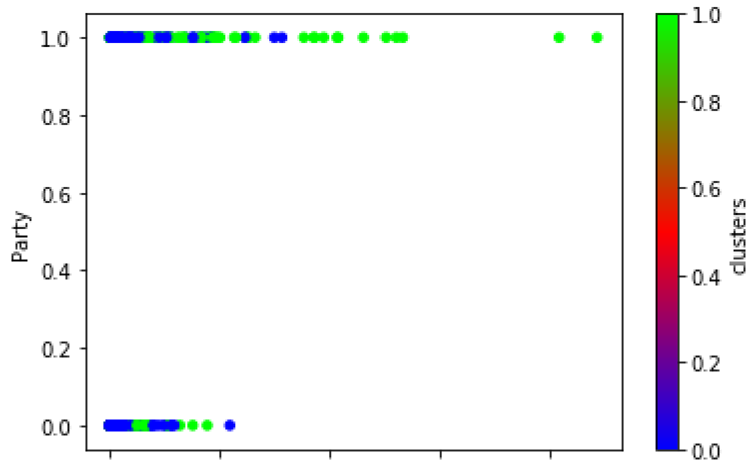
2.3 : use all features to cluster Party based on kmeans method iteration = 10

In [35]:

```
clustering = KMeans(n_clusters=2,init='random',max_iter=10,random_state=0).fit(data_x_scaled2[:,[1]])
clusters = clustering.labels_
cont_matrix = metrics.cluster.contingency_matrix(clusters,Y['Party'])
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
# Plot clusters found using K-Means clustering
Y['clusters'] = clusters
ax = Y.plot(kind = 'scatter', x = 'Total Population', y = 'Party', c = 'clusters', colormap = plt.cm.brg)
# ax.set(title = 'iris data', xlabel = 'petal width', ylabel = 'petal length')
```

```
[0.10878419370681697, 0.7053265714990801]
```





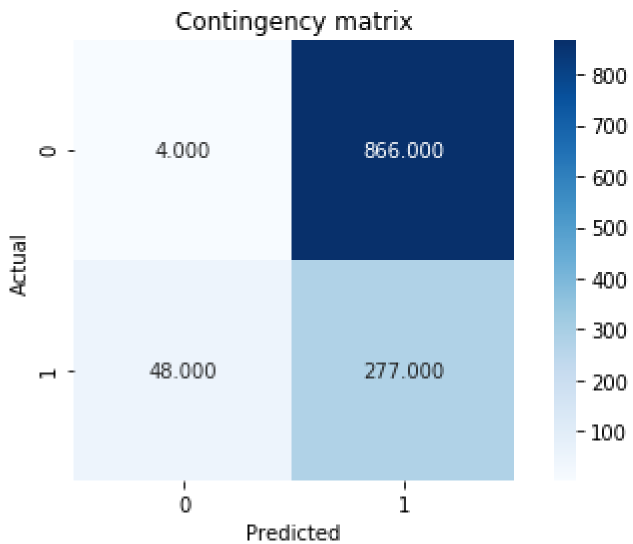
DBSCAN

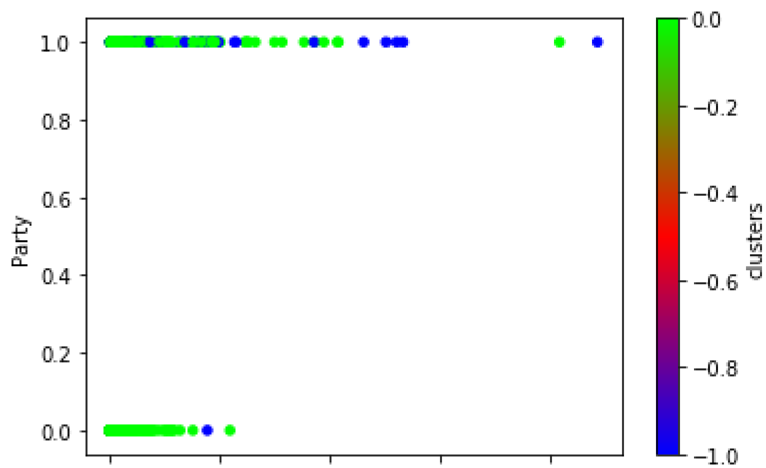
3.1 : use "Total Population","Median Household Income","percent of white" to cluster Party based on kmeans method iteration = 100

In [36]:

```python
clustering = DBSCAN(eps=0.7, min_samples=10).fit(data_x_scaled2[:,[1,2,9]])
clusters = clustering.labels_
cont_matrix = metrics.cluster.contingency_matrix(Y['Party'],clusters)
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,2,9]], clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
# Plot clusters found using K-Means clustering
Y['clusters'] = clusters
ax = Y.plot(kind = 'scatter', x = 'Total Population', y = 'Party', c = 'clusters', colormap = plt.cm.brg)
# ax.set(title = 'iris data', xlabel = 'petal width', ylabel = 'petal length')
```

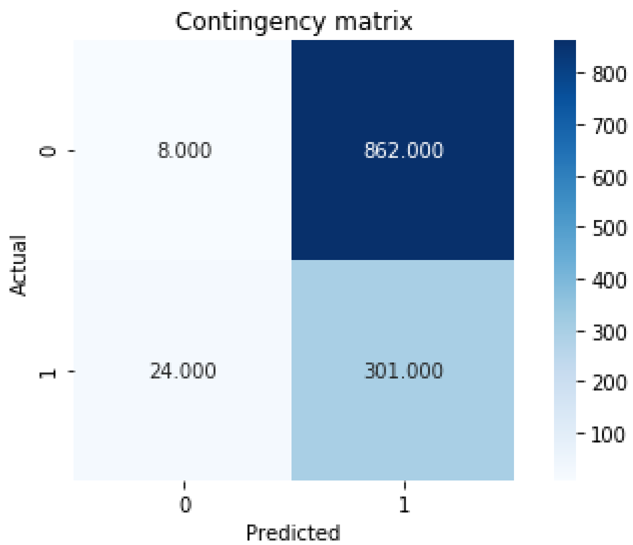
[0.12959192042505843, 0.5525039512529853]

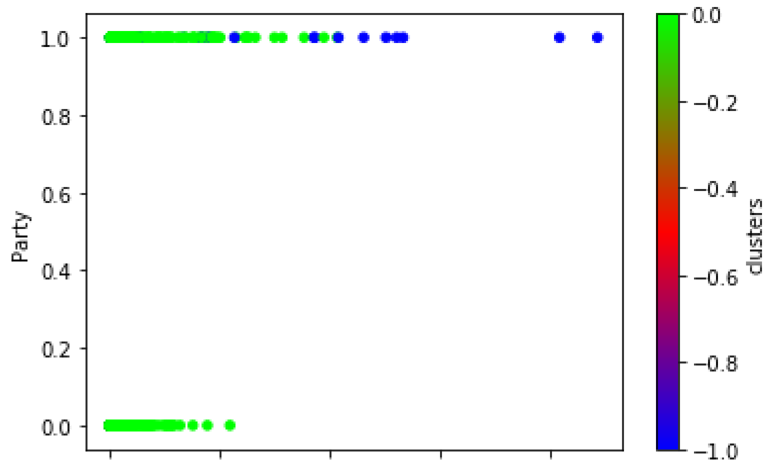




In [37]:
```
clustering = DBSCAN(eps=2.9, min_samples=10).fit(data_x_scaled2)
clusters = clustering.labels_
cont_matrix = metrics.cluster.contingency_matrix(Y['Party'],clusters)
sns.heatmap(cont_matrix, annot = True, fmt = ".3f", square = True, cmap = plt.cm.Blues)
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title('Contingency matrix')
plt.tight_layout()
adjusted_rand_index = metrics.adjusted_rand_score(Y['Party'],clusters)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled, clusters, metric = "euclidean")
print([adjusted_rand_index, silhouette_coefficient])
# Plot clusters found using K-Means clustering
Y['clusters'] = clusters
ax = Y.plot(kind = 'scatter', x = 'Total Population', y = 'Party', c = 'clusters', colormap = plt.cm.brg)
# ax.set(title = 'iris data', xlabel = 'petal width', ylabel = 'petal length')
```

[0.057466708149476645, 0.4809698957529441]





Compute evaluation metrics for the true clusters of the data.

In [38]:
```python
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2,Y['Party'], metric = "euclidean")
print('all features silhouette_coefficient: ',silhouette_coefficient)
silhouette_coefficient = metrics.silhouette_score(data_x_scaled2[:,[1,2,9]],Y['Party'], metric = "euclidean")
print('three features silhouette_coefficient: ',silhouette_coefficient)
```

```
all features silhouette_coefficient:  0.15065111524779617
three features silhouette_coefficient:  0.21892425129127688
```

result

```
    adjusted_rand_index and silhouette_coefficient: [0.0634, 0.6589] single linkage, one variables
    adjusted_rand_index and silhouette_coefficient: [0.1276, 0.6688] complete linkage, one variables
    adjusted_rand_index and silhouette_coefficient: [0.0028, 0.6910] single linkag,e two variables
    adjusted_rand_index and silhouette_coefficient: [0.0835, 0.5250] complete linkage, two variables
    adjusted_rand_index and silhouette_coefficient: [0.0028, 0.6408] single linkag,e three variables
    adjusted_rand_index and silhouette_coefficient: [0.0137, 0.6121] complete linkage, three variables
    adjusted_rand_index and silhouette_coefficient: [0.0453, 0.5521] kmeans method iteration = 10, three variables
    adjusted_rand_index and silhouette_coefficient: [0.0453, 0.5521] kmeans method iteration = 100, three variable
s
    adjusted_rand_index and silhouette_coefficient: [0.1087, 0.7053] kmeans method iteration = 10, all variables
    adjusted_rand_index and silhouette_coefficient: [0.1295, 0.5525] DBSCAN, three features,eps=0.7, min_samples=1
0
    adjusted_rand_index and silhouette_coefficient: [0.0574, 0.4809] DBSCAN, all features,eps=2.9, min_samples=10
Use DBSCAN(eps=0.7, min_samples=10) with "Total Population","Median Household Income","percent of white" feature t
o cluster party has the best perfermance.
The adjusted_rand_index is about 0.25,which means that model is not that good for clustering 'party' based on thre
e features,also means using this model, different party can not be clearly seperated based on these three feature
s. silhouette_coefficient is about 0.4364, diffetent clusters' centroids seperate in a distance, from this aspect,
the clustering is good.
The true cluster's silhouette_coefficient is  0.2189 based on these three features,the true cluster's silhouette_c
oefficient is 0.1506 based on all features
```

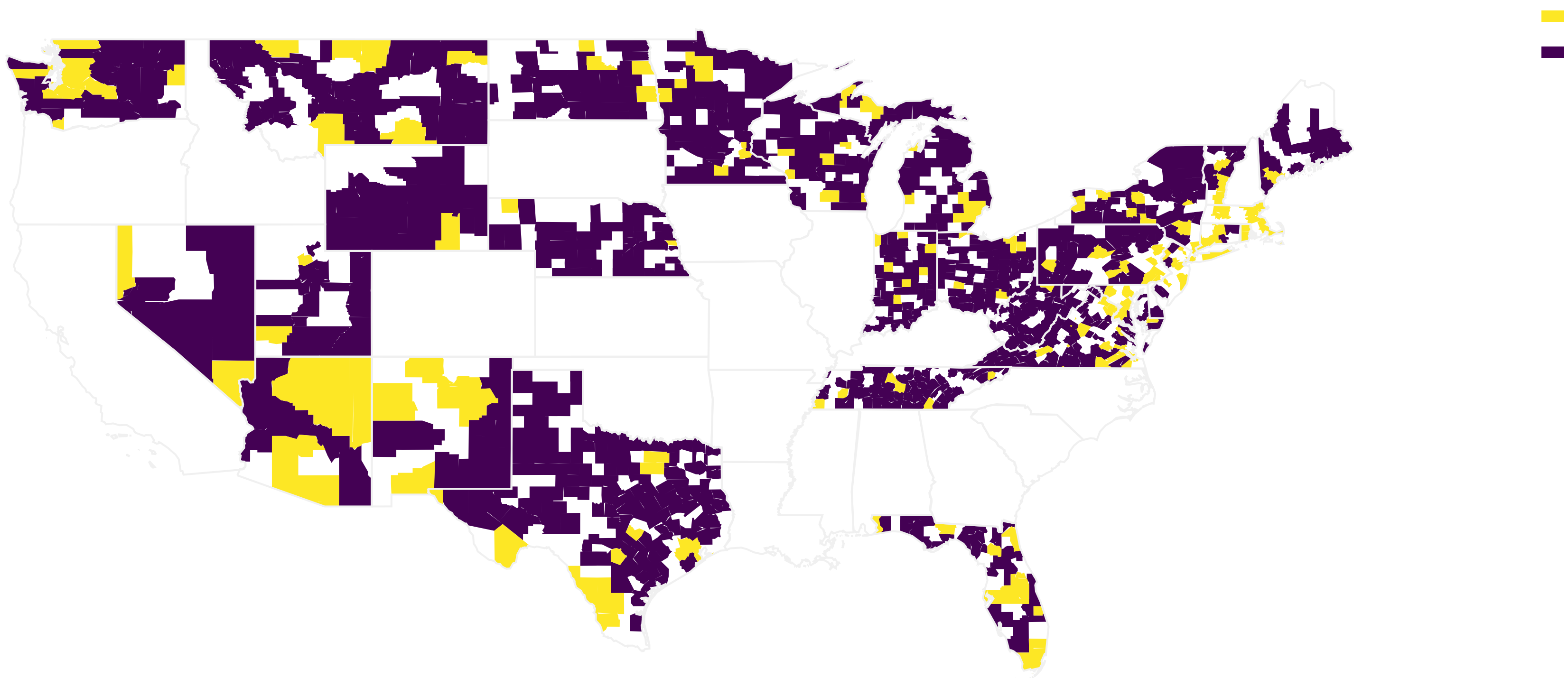
6. (10 pts.) Create a map of Democratic counties and Republican counties using the counties' FIPS codes and Python's Plotly library (plot.ly/python/county-choropleth/). Compare with the map of Democratic counties and Republican counties created in Project 01. What conclusions do you make from the plots?

In [39]:

```
classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled,y_train['Party'])
classify = classifier.predict(data_x_scaled)


Y['classify'] = classify
import plotly.figure_factory as ff
fips = Y['FIPS'].tolist()
values =Y['classify'].tolist()

fig = ff.create_choropleth(fips=fips, values=values)
fig.layout.template = None
fig.show()
```



This map is similar to the previous one, the reason why some counties are classified with different color in this project we analyze are these they are belong to test sample that were not be trained by model so there exist some error, or these counties are located around the classification boundary, the model in order to avoid overfit, just mislabel these counties.

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 [41]:

```
test = pd.read_csv("demographics_test.csv")
new = pd.DataFrame(test['State'])
new['County']= test['County']
test = test.drop(['State','County'], axis=1)
test_scaled = scaler.transform(test)

model = linear_model.Lasso(alpha = 3000)
fitted_model = model.fit(x_train_scaled,y_train['Democratic'])
predicted_Democratic = fitted_model.predict(test_scaled)

model = linear_model.Lasso(alpha = 100)
fitted_model = model.fit(x_train_scaled,y_train['Republican'])
predicted_Republican = fitted_model.predict(test_scaled)

classifier = SVC(kernel='rbf')
classifier.fit(x_train_scaled,y_train['Party'])
classify = classifier.predict(test_scaled)


new['Democratic'] = predicted_Democratic
new['Republican'] = predicted_Republican
new['party'] = classify

new.to_csv(r'E:\predict.csv', index = False, header=True)
```

In [ ]: