Project 02 (Template)

In [206]:
```
# load libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.formula.api as smf
from sklearn import linear_model
from sklearn.model_selection import train_test_split, KFold, cross_val_score,cross_val_predict
from sklearn.preprocessing import scale, StandardScaler
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.model_selection import GridSearchCV

#Libraries for map
import plotly.figure_factory as ff
import chart_studio.plotly as py
import scipy.stats as st
```

In [207]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [208]:
```
# read data
data_train = pd.read_csv('merged_train.csv')
data_test = pd.read_csv('demographics_test.csv')
```

Task1

In [209]:
```
#Partitioning the merged dataset into a training set and a validation set using the cross-valid
folds = KFold(n_splits = 10, shuffle = False)
for train_index, test_index in folds.split(data_train):
    print([train_index.shape[0], test_index.shape[0]])
```

```
[1075, 120]
[1075, 120]
[1075, 120]
[1075, 120]
[1075, 120]
[1076, 119]
[1076, 119]
[1076, 119]
[1076, 119]
[1076, 119]
```

For Democractic Party

In [210]:
```
#First - to predict the number of votes cast for the Democratic party in each county
x = data_train[data_train.columns[0:16]]
y = data_train['Democratic']
```

Task 2

In [211]:

```
#Standardizing the training set and the validation set.
category = x.select_dtypes(include=['object'])
x= x.drop(['County','State'], axis=1)
x.columns
x1=x
cols = x.columns
```

In [212]:

```
scaler = StandardScaler()
scaler.fit(x)
x = scaler.transform(x)
print(x)
```

```
[[-2.63625046e+00 -1.52852858e-01 -3.06640757e+00 ...  1.29637277e+00
   1.12527371e+00  5.66216259e-01]
 [-2.63609667e+00  2.22712198e-02 -1.15599154e+00 ... -3.41198657e-03
  -1.93511628e-01 -6.10414363e-01]
 [-2.63594287e+00  5.32835921e-02 -1.24105649e+00 ... -3.65179484e-01
  -1.39698044e+00 -7.61076704e-01]
 ...
 [ 1.36520628e+00 -2.39218252e-01  3.47986766e-02 ... -6.40853821e-01
   1.67498771e-03 -1.40142266e+00]
 [ 1.36551387e+00 -3.14244542e-01  4.34968142e-01 ... -4.77916280e-01
   3.46469490e-01 -3.98681046e-01]
 [ 1.36566767e+00 -3.53584805e-01  1.65525491e-01 ... -1.32947570e-01
   3.38615907e-02 -6.21212750e-01]]
```

Task 3

In [213]:

```
#Correlation matrix to identify best predictor for simple linear regression model
corr_dict = {}
for c in cols:
    corr_dict[c] = abs(x1[c].corr(data_train['Democratic']))
print (corr_dict)
```

```
{'FIPS': 0.1260748152748367, 'Total Population': 0.9320272614655976, 'Percent White, not Hispanic
or Latino': 0.2740100804425584, 'Percent Black, not Hispanic or Latino': 0.24858235131536197, 'Per
cent Hispanic or Latino': 0.11300070340498887, 'Percent Foreign Born': 0.5045411949757784, 'Percen
t Female': 0.1607853551263807, 'Percent Age 29 and Under': 0.15817705411845867, 'Percent Age 65 an
d Older': 0.25416681887206405, 'Median Household Income': 0.30084737000126827, 'Percent Unemploye
d': 0.0619686979289019, 'Percent Less than High School Degree': 0.10623828078595536, "Percent Less
than Bachelor's Degree": 0.44148289609417396, 'Percent Rural': 0.45025581453551317}
```

In [214]:

```
import operator
sortedcorr_dict = sorted(corr_dict.items(), key= operator.itemgetter(1))
sortedcorr_dict
```

Out[214]:

```
[('Percent Unemployed', 0.0619686979289019),
 ('Percent Less than High School Degree', 0.10623828078595536),
 ('Percent Hispanic or Latino', 0.11300070340498887),
 ('FIPS', 0.1260748152748367),
 ('Percent Age 29 and Under', 0.15817705411845867),
 ('Percent Female', 0.1607853551263807),
 ('Percent Black, not Hispanic or Latino', 0.24858235131536197),
 ('Percent Age 65 and Older', 0.25416681887206405),
 ('Percent White, not Hispanic or Latino', 0.2740100804425584),
 ('Median Household Income', 0.30084737000126827),
 ("Percent Less than Bachelor's Degree", 0.44148289609417396),
 ('Percent Rural', 0.45025581453551317),
 ('Percent Foreign Born', 0.5045411949757784),
 ('Total Population', 0.9320272614655976)]
```

In [215]:
```
#Evaluating with predictor 'Total Population'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,1].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.96233051 0.62586705 0.9337941  0.93559838 0.94522807 0.91697893
 0.86543006 0.8268442  0.85791933 0.8712226 ]
[0.8741213225980164, 0.09285786260366721]
```

In [216]:
```
#Evaluating with predictor 'Percent Foreign Born'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,5].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.31087899  0.24425382  0.19855329 -0.69705329  0.48744005  0.23511014
  0.05188815 -0.94743013  0.20259771  0.28824196]
[0.03744806923789053, 0.44552316064172726]
```

In [217]:
```
#Evaluating with predictor 'Percent Rural'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,13].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.18936159  0.18359809  0.14553604 -0.65801258  0.25985529  0.37555797
  0.17768333 -0.15375597  0.01927199  0.18934806]
[0.07284438184146769, 0.2776093036160327]
```

In [218]:
```
#Evaluating with predictor 'Percent Less than Bachelor's Degree'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,12].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.09663     0.24626903 -0.03025768 -0.36060717  0.17566985  0.4017534
  0.1231017   0.09206524  0.09316693  0.19579432]
[0.10335856117669621, 0.1889497672198973]
```

In [219]:
```
#Evaluating with predictor 'Percent Age 65 and Older'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,8].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[-0.06166975  0.02890744  0.07870157 -0.19498606  0.05384276  0.06644622
  0.09608061 -0.08813394  0.0670198   0.06004256]
[0.010625122668996189, 0.0895172332440331]
```

As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for simple linear regression model is highest for the model with 'Total Population' as the predictor. Hence,

For simple linear regression, to predict number of votes cast for Democratic party, the best performance is given by the model with predictor variable as 'Total Population'

In [220]:

```
#OLS summary to identify multiple combinations of predictor variables at a significance level c
import statsmodels.api as sm
X1 = sm.add_constant(data_train[['FIPS', 'Total Population', 'Percent White, not Hispanic or La
                                 'Percent Black, not Hispanic or Latino', 'Percent Hispanic or
                                'Percent Foreign Born','Percent Female','Percent Age 29 and Und
                                 'Median Household Income','Percent Unemployed','Percent Less t
                                 "Percent Less than Bachelor's Degree",'Percent Rural']].to_num
result = sm.OLS(y,X1).fit()
result.summary()
```

Out[220]:

OLS Regression Results

Dep. Variable:	Democratic	**R-squared:**	0.882
Model:	OLS	**Adj. R-squared:**	0.881
Method:	Least Squares	**F-statistic:**	632.6
Date:	Mon, 18 Nov 2019	**Prob (F-statistic):**	0.00
Time:	19:18:11	**Log-Likelihood:**	-13792.
No. Observations:	1195	**AIC:**	2.761e+04
Df Residuals:	1180	**BIC:**	2.769e+04
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.291e+05	3e+04	4.310	0.000	7.03e+04	1.88e+05
x1	-0.1008	0.060	-1.681	0.093	-0.218	0.017
x2	0.1997	0.003	67.548	0.000	0.194	0.205
x3	-150.1748	101.584	-1.478	0.140	-349.480	49.131
x4	-56.8146	128.446	-0.442	0.658	-308.823	195.194
x5	-278.3753	119.942	-2.321	0.020	-513.699	-43.052
x6	209.1469	227.259	0.920	0.358	-236.731	655.025
x7	48.1115	343.012	0.140	0.888	-624.871	721.094
x8	-801.1537	274.399	-2.920	0.004	-1339.519	-262.788
x9	-431.4604	336.507	-1.282	0.200	-1091.678	228.757
x10	0.0146	0.100	0.147	0.884	-0.181	0.211
x11	251.9702	325.243	0.775	0.439	-386.149	890.090
x12	546.6648	222.636	2.455	0.014	109.858	983.471
x13	-1093.6428	148.150	-7.382	0.000	-1384.309	-802.976
x14	15.5389	34.568	0.450	0.653	-52.283	83.360

Omnibus:	1947.190	**Durbin-Watson:**	1.762
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	1895571.880
Skew:	9.947	**Prob(JB):**	0.00
Kurtosis:	197.099	**Cond. No.**	1.41e+07

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.41e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

In [221]:
```
#Evaluating with all the variables as predictors
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x, y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.95489832 0.65306214 0.93563438 0.77681912 0.93897993 0.93140643
 0.88926476 0.84389616 0.85641832 0.86207317]
[0.8642452709590376, 0.08761161909102733]
```

In [222]:
```
#Evaluating with two predictors 'Total Population', 'Percent Less than Bachelor's Degree'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,12]].reshape(-1,2), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.96941275 0.64871925 0.938147   0.86189062 0.95096555 0.92827846
 0.88674872 0.82800008 0.86411608 0.88298051]
[0.8759259008434993, 0.08683668784594073]
```

In [223]:
```
#Evaluating with the predictors 'Total Population', 'Percent Less than Bachelor's Degree','Perc
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,12,5,13,8]].reshape(-1,5), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.96788683 0.64957303 0.93493892 0.8459717  0.94525318 0.92823639
 0.88205675 0.83289243 0.86435311 0.88021977]
[0.8731382104156695, 0.08577820537660646]
```

In [224]:
```
#Evaluating with the predictors 'Total Population', 'Percent Less than Bachelor's Degree','Perc
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,12,13,7]].reshape(-1,4), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.96763299 0.6507847  0.93372953 0.8583636  0.95003429 0.92928354
 0.88379879 0.83679636 0.86685732 0.88389277]
[0.8761173874700908, 0.08533814034753137]
```

In [225]:
```
#Evaluating with the predictors 'Total Population', 'Percent Less than Bachelor's Degree','Perc
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,12,7]].reshape(-1,3), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.96768806 0.65080786 0.93369853 0.85900243 0.95002191 0.93012499
 0.88725169 0.83674902 0.86704169 0.88410078]
[0.8766486980131882, 0.08541305723112652]
```

As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for multiple linear regression model is highest for the model with 'Total Population',"Percent Less than Bachelor's Degree", 'Percent Age 29 and Under' as the predictors. Hence,

For multiple linear regression, to predict number of votes cast for Democratic party, the best performance is given by the model with predictor variables as 'Total Population',"Percent Less than Bachelor's Degree",'Percent Age 29 and Under'*

In [226]:
```
model=linear_model.LinearRegression().fit(X = x[:,[1,12,7]].reshape(-1,3), y = y)
best_dems=model
```

For Republican Party

In [227]:

```
#Second - to predict the number of votes cast for the Republican party in each county
x = data_train[data_train.columns[0:16]]
y = data_train['Republican']
```

In [228]:

```
#Standardizing the training set and the validation set.
category = x.select_dtypes(include=['object'])
x= x.drop(['County','State'], axis=1)
x.columns
x1=x
cols = x.columns
```

In [229]:

```
scaler = StandardScaler()
scaler.fit(x)
x = scaler.transform(x)
print(x)
```

```
[[-2.63625046e+00 -1.52852858e-01 -3.06640757e+00 ...  1.29637277e+00
   1.12527371e+00  5.66216259e-01]
 [-2.63609667e+00  2.22712198e-02 -1.15599154e+00 ... -3.41198657e-03
  -1.93511628e-01 -6.10414363e-01]
 [-2.63594287e+00  5.32835921e-02 -1.24105649e+00 ... -3.65179484e-01
  -1.39698044e+00 -7.61076704e-01]
 ...
 [ 1.36520628e+00 -2.39218252e-01  3.47986766e-02 ... -6.40853821e-01
   1.67498771e-03 -1.40142266e+00]
 [ 1.36551387e+00 -3.14244542e-01  4.34968142e-01 ... -4.77916280e-01
   3.46469490e-01 -3.98681046e-01]
 [ 1.36566767e+00 -3.53584805e-01  1.65525491e-01 ... -1.32947570e-01
   3.38615907e-02 -6.21212750e-01]]
```

In [230]:

```
#Correlation matrix to identify best predictor for simple linear regression model
corr_dict = {}
for c in cols:
    corr_dict[c] = abs(x1[c].corr(data_train['Republican']))
print (corr_dict)
```

```
{'FIPS': 0.15542188568079557, 'Total Population': 0.9067499465108058, 'Percent White, not Hispanic
or Latino': 0.21027219180127515, 'Percent Black, not Hispanic or Latino': 0.1729493804049177, 'Per
cent Hispanic or Latino': 0.10418643852769169, 'Percent Foreign Born': 0.42654484513833774, 'Perce
nt Female': 0.16525358836178244, 'Percent Age 29 and Under': 0.15477975485350148, 'Percent Age 65
and Older': 0.24006526156825958, 'Median Household Income': 0.320041504027138, 'Percent Unemploye
d': 0.056638987520030035, 'Percent Less than High School Degree': 0.13717801107200953, "Percent Le
ss than Bachelor's Degree": 0.4171438300507714, 'Percent Rural': 0.4813754764602791}
```

In [231]:

```
import operator
sortedcorr_dict = sorted(corr_dict.items(), key= operator.itemgetter(1))
sortedcorr_dict
```

Out[231]:

```
[('Percent Unemployed', 0.056638987520030035),
 ('Percent Hispanic or Latino', 0.10418643852769169),
 ('Percent Less than High School Degree', 0.13717801107200953),
 ('Percent Age 29 and Under', 0.15477975485350148),
 ('FIPS', 0.15542188568079557),
 ('Percent Female', 0.16525358836178244),
 ('Percent Black, not Hispanic or Latino', 0.1729493804049177),
 ('Percent White, not Hispanic or Latino', 0.21027219180127515),
 ('Percent Age 65 and Older', 0.24006526156825958),
 ('Median Household Income', 0.320041504027138),
 ("Percent Less than Bachelor's Degree", 0.4171438300507714),
 ('Percent Foreign Born', 0.42654484513833774),
 ('Percent Rural', 0.4813754764602791),
 ('Total Population', 0.9067499465108058)]
```

In [232]:

```
#Evaluating with predictor 'Total Population'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,1].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.87526991  0.645852    0.90664131  0.78602157 -0.01733154  0.91508408
  0.90662245  0.88928016  0.93398776  0.80359756]
[0.7645025255819977, 0.27330831959314233]
```

In [233]:

```
#Evaluating with predictor 'Percent Foreign Born'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,5].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.11704595  0.19502745  0.15696205 -0.29242465  0.27162975  0.15866059
  0.10125629 -0.36296574  0.09097201  0.32467688]
[0.07608406006882837, 0.21415899095959257]
```

In [234]:

```
#Evaluating with predictor 'Percent Rural'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,13].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.1269351   0.24020195 -0.06808634 -0.11741692  0.34090628  0.49123699
  0.20456855  0.06705776 -0.29663733  0.28927695]
[0.12780430007923932, 0.2245981649253793]
```

In [235]:

```
#Evaluating with predictor 'Percent Less than Bachelor's Degree'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,12].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[ 0.01552391  0.26940246 -0.46219797 -0.11658124  0.17486325  0.40237576
  0.17091039  0.10434618 -0.32939875  0.24437474]
[0.047361873119147614, 0.2604630108650094]
```

In [236]:

```
#Evaluating with predictor 'Percent Age 65 and Older'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,8].reshape(-1,1), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[-0.14492081  0.03397132 -0.08002485 -0.09997718  0.07631973  0.06983268
  0.09987132  0.063497    0.03119151  0.06523328]
[0.011499400673974 88, 0.08195662611048625]
```

As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for simple linear regression model is highest for the model with 'Total Population' as the predictor. Hence,

For simple linear regression, to predict number of votes cast for Republican party, the best performance is given by the model with predictor variable as 'Total Population'

In [237]:

```
#OLS summary to identify multiple combinations of predictor variables at a significance level o
import statsmodels.api as sm
X1 = sm.add_constant(data_train[['FIPS', 'Total Population', 'Percent White, not Hispanic or La
                                  'Percent Black, not Hispanic or Latino', 'Percent Hispanic or
                                 'Percent Foreign Born','Percent Female','Percent Age 29 and Und
                                  'Median Household Income','Percent Unemployed','Percent Less t
                                  "Percent Less than Bachelor's Degree",'Percent Rural']].to_num
result = sm.OLS(y,X1).fit()
result.summary()
```

Out[237]:

OLS Regression Results

Dep. Variable:	Republican	**R-squared:**	0.853
Model:	OLS	**Adj. R-squared:**	0.852
Method:	Least Squares	**F-statistic:**	490.4
Date:	Mon, 18 Nov 2019	**Prob (F-statistic):**	0.00
Time:	19:18:12	**Log-Likelihood:**	-13359.
No. Observations:	1195	**AIC:**	2.675e+04
Df Residuals:	1180	**BIC:**	2.682e+04
Df Model:	14		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	1.077e+04	2.08e+04	0.517	0.606	-3.01e+04	5.17e+04
x1	-0.1627	0.042	-3.901	0.000	-0.245	-0.081
x2	0.1298	0.002	63.094	0.000	0.126	0.134
x3	161.5069	70.680	2.285	0.022	22.834	300.180
x4	-173.5004	89.371	-1.941	0.052	-348.844	1.843
x5	218.7927	83.454	2.622	0.009	55.058	382.527
x6	-1300.8924	158.123	-8.227	0.000	-1611.127	-990.658
x7	-290.6631	238.662	-1.218	0.224	-758.913	177.587
x8	-20.3825	190.923	-0.107	0.915	-394.968	354.203
x9	596.7814	234.136	2.549	0.011	137.413	1056.150
x10	0.4672	0.069	6.724	0.000	0.331	0.603
x11	650.2209	226.299	2.873	0.004	206.228	1094.214
x12	634.4736	154.906	4.096	0.000	330.551	938.396
x13	-371.6360	103.080	-3.605	0.000	-573.877	-169.395
x14	-176.4973	24.052	-7.338	0.000	-223.686	-129.308

Omnibus:	445.225	**Durbin-Watson:**	1.728
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	168842.295
Skew:	-0.199	**Prob(JB):**	0.00
Kurtosis:	61.231	**Cond. No.**	1.41e+07

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.41e+07. This might indicate that there are
strong multicollinearity or other numerical problems.

In [238]:

```
#Evaluating with all the variables as predictors
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x, y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.86874246 0.69668621 0.84368189 0.85077882 0.12795994 0.94399503
 0.88990667 0.86179582 0.87705097 0.83112294]
[0.7791720752455482, 0.22510612424341983]
```

In [239]:

```
#Evaluating with two predictors 'Total Population', 'Percent Less than Bachelor's Degree'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,12]].reshape(-1,2), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.86783969 0.66597527 0.90473258 0.78493934 0.01765712 0.92299889
 0.92584987 0.87996954 0.87223655 0.83149035]
[0.7673689193574214, 0.26052733049478494]
```

In [240]:

```
#Evaluating with the predictors 'Total Population', 'Percent Less than Bachelor's Degree','Perc
#'Percent Foreign Born','Percent Age 65 and Older','Percent Rural','Percent Hispanic or Latino'
#'Median Household Income','Percent Less than High School Degree'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,2,4,5,8,9,10,11,12,13]].reshape(-1,10), y = y, cv =
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.8696952  0.6928362  0.86549936 0.85554606 0.11206841 0.9453725
 0.88774067 0.87494517 0.89192827 0.83217169]
[0.7827803523488464, 0.23193445966502016]
```

In [241]:

```
#Evaluating with the predictors 'Total Population', 'Percent Less than Bachelor's Degree',
#'Percent Age 29 and Under','Percent Rural','Percent Foreign Born','Median Household Income'
model = linear_model.LinearRegression()
scores = cross_val_score(model, X = x[:,[1,5,9,11,12,13]].reshape(-1,6), y = y, cv = folds)
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.85968135 0.67605586 0.88445037 0.86155231 0.11733464 0.94996324
 0.90961736 0.89190874 0.87810602 0.84333285]
[0.7872002729610357, 0.23348847469633535]
```

As we observe the evaluation metric - the average 'R-squared' value returned using cross-validation method for multiple linear regression model is highest for the model with 'Total Population',"Percent Less than Bachelor's Degree", 'Percent Age 29 and Under' as the predictors. Hence,

For multiple linear regression, to predict number of votes cast for Republican party, the best performance is given by the model with predictor variables as 'Total Population',"Percent Less than Bachelor's Degree",'Percent Age 29 and Under','Percent Rural','Percent Foreign Born','Median Household Income'*

In [242]:

```
model=linear_model.LinearRegression().fit(X = x[:,[1,5,9,11,12,13]].reshape(-1,6), y = y)
best_repb=model
```

Task 4

In [243]:

```
# read data
data_train = pd.read_csv('merged_train.csv')
data_test = pd.read_csv('demographics_test.csv')
```

In [244]:
```
#To classify each county as Democratic or Republican using 'Party' variable
x = data_train[data_train.columns[0:16]]
y = data_train['Party']
```

In [245]:
```
#Standardizing the training set and the validation set.
category = x.select_dtypes(include=['object'])
x= x.drop(['County','State'], axis=1)
x.columns
x1=x
cols = x.columns
```

In [246]:
```
scaler = StandardScaler()
scaler.fit(x)
x = scaler.transform(x)
print(x)
```

```
[[-2.63625046e+00 -1.52852858e-01 -3.06640757e+00 ...  1.29637277e+00
   1.12527371e+00  5.66216259e-01]
 [-2.63609667e+00  2.22712198e-02 -1.15599154e+00 ... -3.41198657e-03
  -1.93511628e-01 -6.10414363e-01]
 [-2.63594287e+00  5.32835921e-02 -1.24105649e+00 ... -3.65179484e-01
  -1.39698044e+00 -7.61076704e-01]
 ...
 [ 1.36520628e+00 -2.39218252e-01  3.47986766e-02 ... -6.40853821e-01
   1.67498771e-03 -1.40142266e+00]
 [ 1.36551387e+00 -3.14244542e-01  4.34968142e-01 ... -4.77916280e-01
   3.46469490e-01 -3.98681046e-01]
 [ 1.36566767e+00 -3.53584805e-01  1.65525491e-01 ... -1.32947570e-01
   3.38615907e-02 -6.21212750e-01]]
```

In [247]:
```
#Correlation matrix for each variable with 'Party' variable
corr_dict = {}
for c in cols:
    corr_dict[c] = abs(x1[c].corr(data_train['Party']))
print (corr_dict)
```

```
{'FIPS': 0.05417309415940023, 'Total Population': 0.3449337810794726, 'Percent White, not Hispanic
or Latino': 0.29230633837436154, 'Percent Black, not Hispanic or Latino': 0.24213396439811313, 'Pe
rcent Hispanic or Latino': 0.08048074514191253, 'Percent Foreign Born': 0.2933074062816451, 'Perce
nt Female': 0.1411775697109578, 'Percent Age 29 and Under': 0.21543763546930342, 'Percent Age 65 a
nd Older': 0.24629236356612966, 'Median Household Income': 0.18279521010454283, 'Percent Unemploye
d': 0.08132214051303853, 'Percent Less than High School Degree': 0.14723025534824363, "Percent Les
s than Bachelor's Degree": 0.4425027623847222, 'Percent Rural': 0.37647015842285164}
```

In [248]:
```
import operator
sortedcorr_dict = sorted(corr_dict.items(), key= operator.itemgetter(1))
sortedcorr_dict
```

Out[248]:
```
[('FIPS', 0.05417309415940023),
 ('Percent Hispanic or Latino', 0.08048074514191253),
 ('Percent Unemployed', 0.08132214051303853),
 ('Percent Female', 0.1411775697109578),
 ('Percent Less than High School Degree', 0.14723025534824363),
 ('Median Household Income', 0.18279521010454283),
 ('Percent Age 29 and Under', 0.21543763546930342),
 ('Percent Black, not Hispanic or Latino', 0.24213396439811313),
 ('Percent Age 65 and Older', 0.24629236356612966),
 ('Percent White, not Hispanic or Latino', 0.29230633837436154),
 ('Percent Foreign Born', 0.2933074062816451),
 ('Total Population', 0.3449337810794726),
 ('Percent Rural', 0.37647015842285164),
 ("Percent Less than Bachelor's Degree", 0.4425027623847222)]
```

Decision Tree Classifier

In [249]:

```
#Using Decision-tree classfier using all vairables as predictors
classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
cv_scores = cross_val_score(classifier, X=x, y=y, cv = folds)
#print each cv score (accuracy) and average them
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.85968135 0.67605586 0.88445037 0.86155231 0.11733464 0.94996324
 0.90961736 0.89190874 0.87810602 0.84333285]
[0.7872002729610357, 0.23348847469633535]
```

In [250]:

```
#Using Decision-tree classfier using most 5 correlated variables
classifier = DecisionTreeClassifier(criterion = "entropy", random_state = 0)
cv_scores = cross_val_score(classifier, X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y, cv = folds)
#print each cv score (accuracy) and average them
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.85968135 0.67605586 0.88445037 0.86155231 0.11733464 0.94996324
 0.90961736 0.89190874 0.87810602 0.84333285]
[0.7872002729610357, 0.23348847469633535]
```

In [251]:

```
#For identifying best performing parameters using GridSearchCV
parameters={'min_samples_split' : range(10,500,20),'max_depth': range(1,20,2), 'criterion':['er
clf_tree=DecisionTreeClassifier()
gridsearch = GridSearchCV(estimator=clf_tree, param_grid = parameters, scoring='accuracy', cv=f
gridSearch = gridsearch.fit(X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y)
print(gridSearch.best_score_,gridSearch.best_params_)
```

```
0.802510460251046 {'criterion': 'gini', 'max_depth': 3, 'min_samples_split': 10}
```

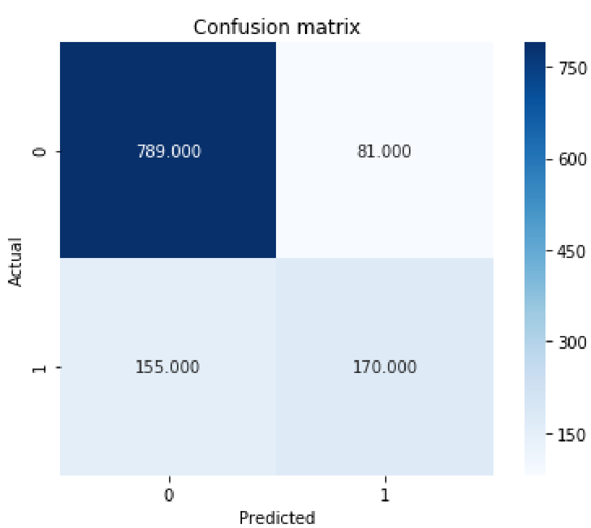
In [252]:

```
final_DT=DecisionTreeClassifier(criterion = "gini", max_depth  = 3, min_samples_split =10,randc
y_pred_DT=cross_val_predict(final_DT,X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y, cv = folds)
```

In [253]:

```
conf_matrix = metrics.confusion_matrix(y, y_pred_DT)
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[253]: (2.0, 0.0)



In [254]:
```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_DT)                     #Accuracy
error = 1 - accuracy                                               #Error
precision = metrics.precision_score(y, y_pred_DT, average = None)  #Precision
recall = metrics.recall_score(y, y_pred_DT, average = None)        #Recall
F1_score = metrics.f1_score(y, y_pred_DT, average = None)          #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.802510460251046, 0.197489539748954, array([0.83580508, 0.67729084]), array([0.90689655, 0.52307
692]), array([0.86990077, 0.59027778])]
```

SVM Classifier

In [255]:
```
#Using SVM classfier using all vairables as predictors
classifier = SVC(kernel='rbf', random_state=0)
cv_scores = cross_val_score(classifier, X=x, y=y, cv = folds)
#print each cv score (accuracy) and average them
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.85968135 0.67605586 0.88445037 0.86155231 0.11733464 0.94996324
 0.90961736 0.89190874 0.87810602 0.84333285]
[0.7872002729610357, 0.23348847469633535]
```

In [256]:
```
#Using SVM classfier using most 5 correlated variables
classifier = SVC(kernel='linear', random_state=0)
cv_scores = cross_val_score(classifier, X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y, cv = folds)
#print each cv score (accuracy) and average them
print(scores)
print([scores.mean(), scores.std()])
```

```
[0.85968135 0.67605586 0.88445037 0.86155231 0.11733464 0.94996324
 0.90961736 0.89190874 0.87810602 0.84333285]
[0.7872002729610357, 0.23348847469633535]
```

In [257]:
```
Svm = SVC(random_state=0)
#Find out the best model for SVM
parameters = [{'C':[1,10,100], 'kernel': ['rbf', 'linear']}]
gridsearch = GridSearchCV(estimator = Svm, param_grid = parameters, scoring = 'accuracy', cv =
gridSearch = gridsearch.fit(X=x[:,[1,13,12,5,2]].reshape(-1,5), y=y)
print(gridSearch.best_score_,gridSearch.best_params_)
```

```
0.8100418410041841 {'C': 10, 'kernel': 'rbf'}
```

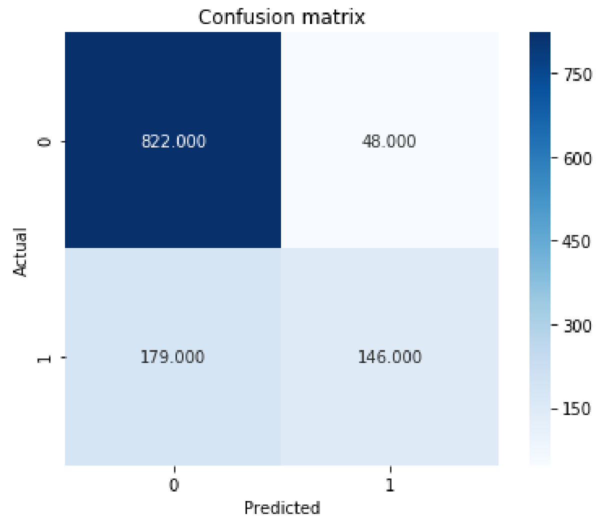
In [258]:
```
final_SVM=SVC(kernel='rbf', C=10, random_state = 0)
y_pred_SVM=cross_val_predict(final_SVM,X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y, cv = folds)
bestModel = final_SVM.fit(X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y)
```

In [259]:

```
conf_matrix = metrics.confusion_matrix(y, y_pred_SVM)
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[259]:

```
(2.0, 0.0)
```



In [260]:

```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_SVM)                    #Accuracy
error = 1 - accuracy                                            #Error
precision = metrics.precision_score(y, y_pred_SVM, average = None)  #Precision
recall = metrics.recall_score(y, y_pred_SVM, average = None)        #Recall
F1_score = metrics.f1_score(y, y_pred_SVM, average = None)          #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.8100418410041841, 0.18995815899581592, array([0.82117882, 0.75257732]), array([0.94482759, 0.44
923077]), array([0.87867451, 0.56262042])]
```

K-Nearest Neighbors

In [261]:

```
#Using KNN and use all variables as predictors
knn_cv = KNeighborsClassifier(n_neighbors = 3)
#train model with cv of 10
cv_scores = cross_val_score(knn_cv, X = x, y = y, cv = folds)
#print each cv score (accuracy) and average them
print(cv_scores)
print([scores.mean(), scores.std()])
```

```
[0.85833333 0.79166667 0.60833333 0.66666667 0.75833333 0.85714286
 0.88235294 0.8487395  0.66386555 0.66386555]
[0.7872002729610357, 0.23348847469633535]
```

In [262]:

```
#Find out the best model for KNN
parameters = [{'n_neighbors':range(10,500,20)}]
gridsearch = GridSearchCV(estimator = knn_cv, param_grid = parameters, scoring = 'accuracy', cv
gridSearch = gridsearch.fit(X=x[:,[1,13,12,5,2]].reshape(-1,5), y=y)
print(gridSearch.best_score_,gridSearch.best_params_)
```

```
0.796652719665272 {'n_neighbors': 10}
```

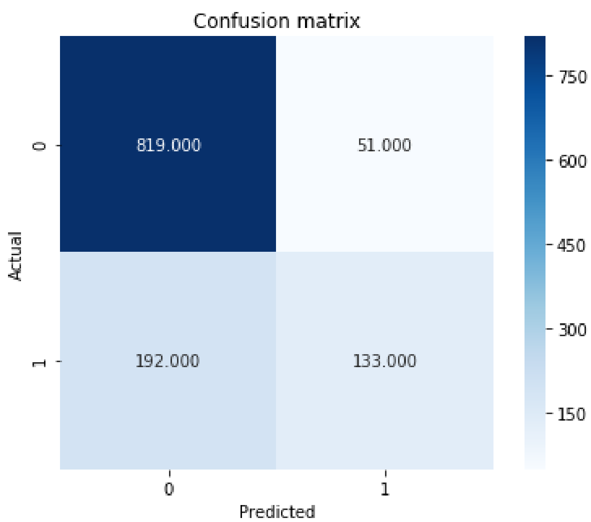
In [263]:

```
final_knn=KNeighborsClassifier(n_neighbors = 10)
y_pred_KNN=cross_val_predict(final_knn,X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y, cv = folds)
```

In [264]:

```
conf_matrix = metrics.confusion_matrix(y, y_pred_KNN)
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[264]: (2.0, 0.0)



In [265]:

```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_KNN)                        #Accuracy
error = 1 - accuracy                                                  #Error
precision = metrics.precision_score(y, y_pred_KNN, average = None)    #Precision
recall = metrics.recall_score(y, y_pred_KNN, average = None)          #Recall
F1_score = metrics.f1_score(y, y_pred_KNN, average = None)            #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.796652719665272, 0.20334728033472804, array([0.81008902, 0.72282609]), array([0.94137931, 0.409
23077]), array([0.8708134 , 0.52259332])]
```

Naive Bayes

In [266]:

```python
#Using all variables as predictors
NB_cv = GaussianNB()
#train model with cv of 10
cv_scores = cross_val_score(NB_cv, X = x, y = y, cv = folds)
#print each cv score (accuracy) and average them
print(cv_scores)
print([scores.mean(), scores.std()])
```

```
[0.83333333 0.79166667 0.625      0.88333333 0.78333333 0.88235294
 0.69747899 0.78151261 0.72268908 0.65546218]
[0.7872002729610357, 0.23348847469633535]
```

In [267]:

```python
#Using 5 most correlated variables as predictors
NB_cv = GaussianNB()
#train model with cv of 10
cv_scores = cross_val_score(NB_cv, X=x[:,[1,12,13,5,2]].reshape(-1,5), y = y, cv = folds)
#print each cv score (accuracy) and average them
print(cv_scores)
print([scores.mean(), scores.std()])
```

```
[0.84166667 0.80833333 0.61666667 0.875      0.8        0.89915966
 0.77310924 0.84033613 0.72268908 0.65546218]
[0.7872002729610357, 0.23348847469633535]
```

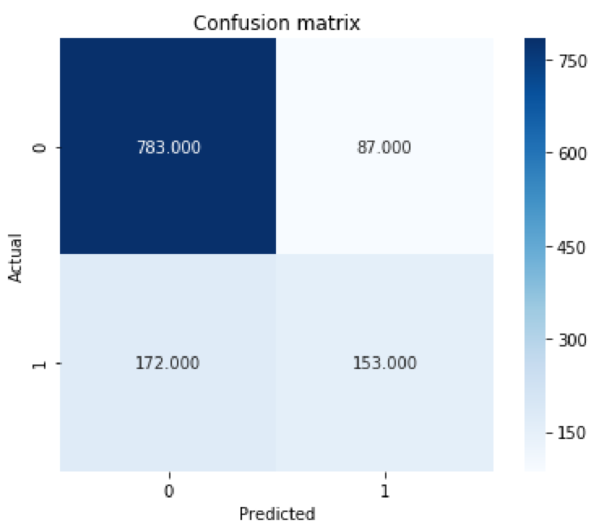
In [268]:

```python
y_pred_NB = cross_val_predict(NB_cv,X=x[:,[1,12,13,5,2]].reshape(-1,5), y=y, cv = folds)
```

In [269]:

```python
conf_matrix = metrics.confusion_matrix(y, y_pred_NB)
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[269]: (2.0, 0.0)



In [270]:
```
#Evaluation metrics
accuracy = metrics.accuracy_score(y, y_pred_NB)                     #Accuracy
error = 1 - accuracy                                             #Error
precision = metrics.precision_score(y, y_pred_NB, average = None)  #Precision
recall = metrics.recall_score(y, y_pred_NB, average = None)        #Recall
F1_score = metrics.f1_score(y, y_pred_NB, average = None)          #F1-Score
print([accuracy, error, precision, recall, F1_score])
```

```
[0.7832635983263598, 0.21673640167364017, array([0.81989529, 0.6375    ]), array([0.9       , 0.47
076923]), array([0.85808219, 0.54159292])]
```

Task 5

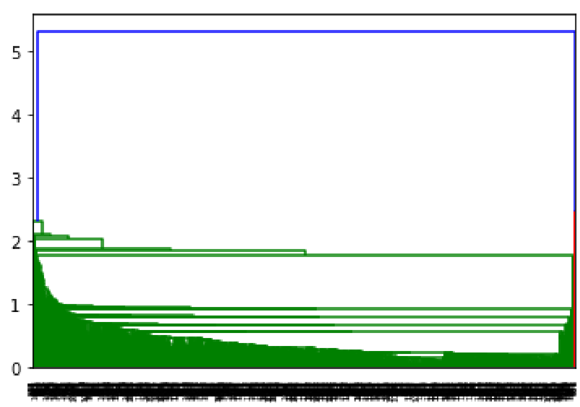
In [271]:
```
# read data
data_train = pd.read_csv('merged_train.csv')
data_test = pd.read_csv('demographics_test.csv')
```

In [272]:
```
#To classify each county as Democratic or Republican using 'Party' variable
X = data_train[['Total Population','Percent White, not Hispanic or Latino','Percent Rural','Per
Y = data_train['Party']
```

In [273]:
```
# Standardize the data
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

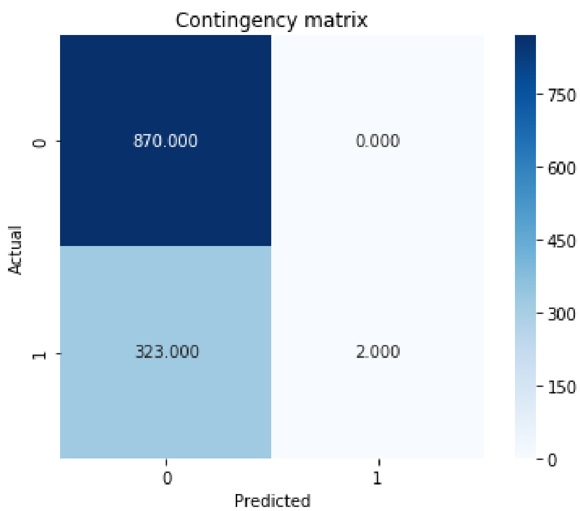
In [274]:
```
#Cluster the dataset using hierarchical clustering with single linkage
from scipy.cluster.hierarchy import linkage, fcluster

clustering = linkage(X_scaled, method = "single", metric = "euclidean")
clusters = fcluster(clustering, 2, criterion = 'maxclust')
```

In [275]:
```
# plot dendrogram
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
plt.figure()
dendrogram(clustering)
plt.show()
```



In [276]:
```
# Plot contingency matrix
from sklearn import metrics
data_train['clusters'] = clusters - 1
cont_matrix = metrics.cluster.contingency_matrix(data_train['Party'], data_train['clusters'])
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[276]: (2.0, 0.0)



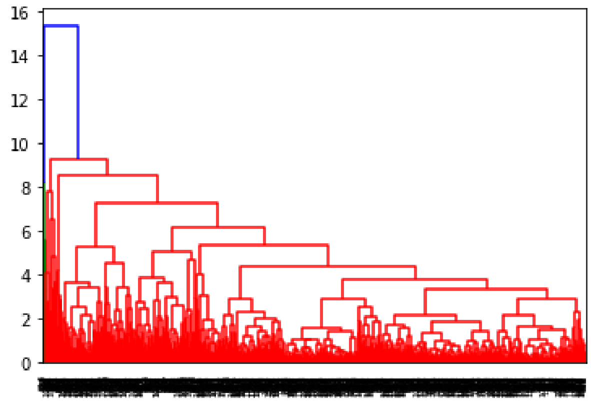
In [277]:
```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.005608925119335567
0.8040606718628662
```

In [278]:
```
# CLUSTERING: Agglomerative clustering with complete (max) linkage
clustering = linkage(X_scaled, method = "complete", metric = "euclidean")
```

In [279]:

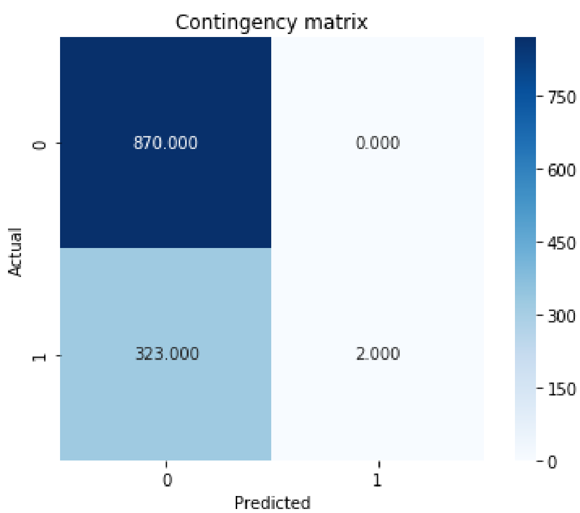
```
plt.figure()
dendrogram(clustering)
plt.show()
```



In [280]:

```
data_train['clusters'] = clusters-1
cont_matrix = metrics.cluster.contingency_matrix(data_train['Party'], data_train['clusters'])
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[280]: (2.0, 0.0)



In [281]:
```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.005608925119335567
0.8040606718628662
```

In [282]:
```
from sklearn.cluster import KMeans, DBSCAN

# K-Means Clustering (random initialization, multiple iterations)
clustering = KMeans(n_clusters = 2, init = 'random', n_init = 10, random_state = 0).fit(X_scale
#Show centroids
clustering.cluster_centers_
```

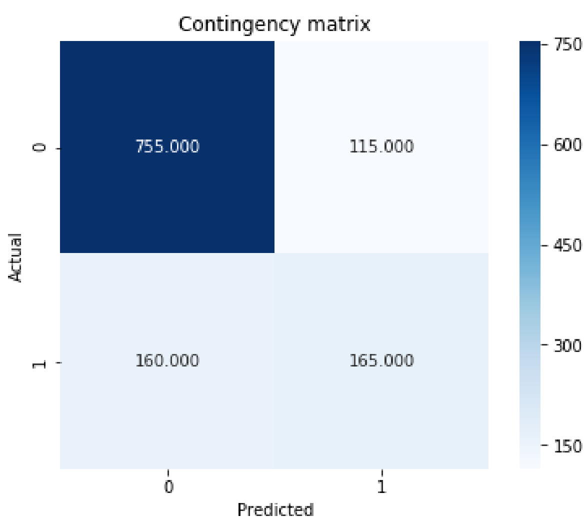
Out[282]:
```
array([[-0.25516482,  0.30447984,  0.36796776,  0.2917043 , -0.37351112],
       [ 0.83384217, -0.99499661, -1.20246608, -0.95324797,  1.22058098]])
```

In [283]:
```
clusters = clustering.labels_
print(clusters)
```

```
[0 1 1 ... 0 0 0]
```

In [284]:
```
data_train['clusters'] = clusters
cont_matrix = metrics.cluster.contingency_matrix(data_train['Party'], data_train['clusters'])
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[284]:
```
(2.0, 0.0)
```



In [285]:
```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.24672990896646144
0.45221355231267646
```

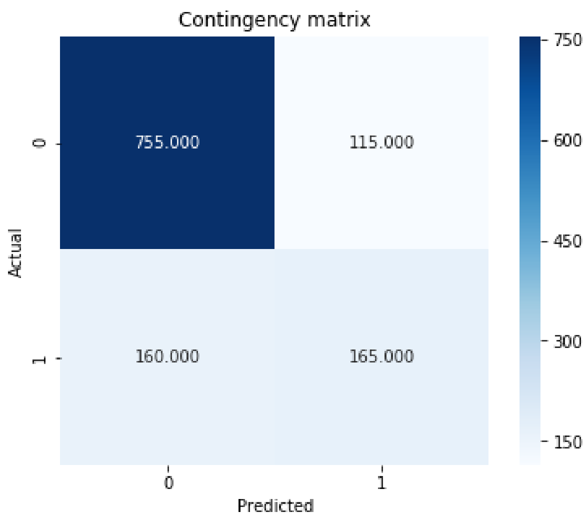
In [286]:
```
#  K-Means clustering (k-means++ initialization, multiple iterations)
clustering = KMeans(n_clusters = 2, init = 'k-means++', n_init = 10).fit(X_scaled)
```

In [287]:
```
clusters = clustering.labels_
print(clusters)
```

```
[0 1 1 ... 0 0 0]
```

In [288]:
```
data_train['clusters'] = clusters
cont_matrix = metrics.cluster.contingency_matrix(data_train['Party'], data_train['clusters'])
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[288]:
```
(2.0, 0.0)
```



In [289]:
```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.24672990896646144
0.45221355231267646
```

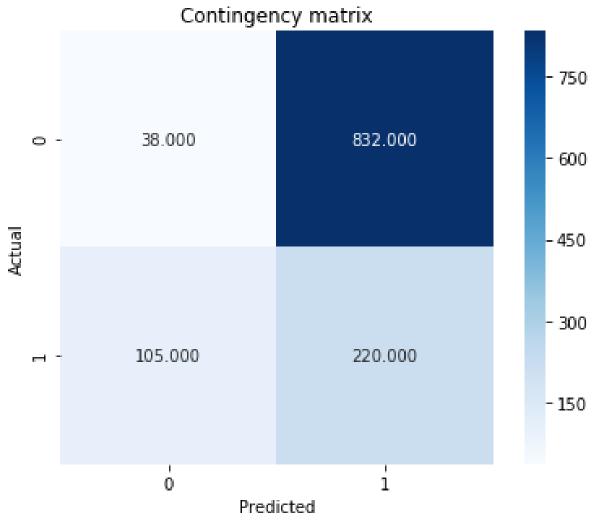
In [290]:
```
# CLUSTERING: DBSCAN
clustering = DBSCAN(eps = 1, min_samples = 20, metric = "euclidean").fit(X_scaled)
```

In [291]:
```
# Show clusters
clusters = clustering.labels_
```

In [292]:

```
data_train['clusters'] = clusters + 1
cont_matrix = metrics.cluster.contingency_matrix(data_train['Party'], data_train['clusters'])
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[292]: (2.0, 0.0)



In [293]:

```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.2299622636512366
0.5150075044243099
```

In [294]:

```
# Changing variables
X = data_train[['Percent White, not Hispanic or Latino','Percent Black, not Hispanic or Latino'
Y = data_train['Party']
```

In [295]:

```
# Standardize the attributes
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

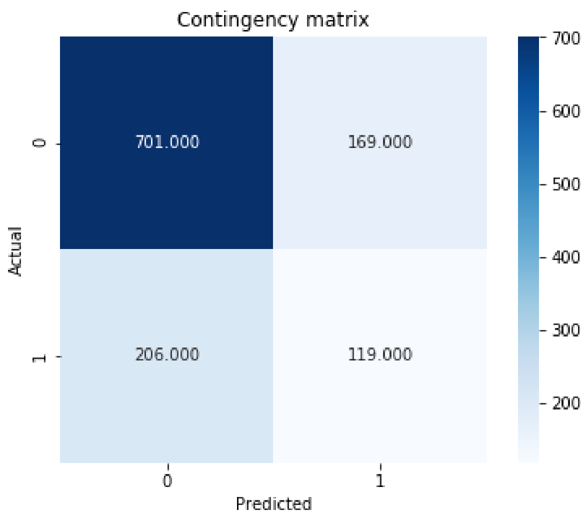
In [296]:
```
#  K-Means clustering (k-means++ initialization, single iteration)
clustering = KMeans(n_clusters = 2, init = 'k-means++', n_init = 1).fit(X_scaled)
```

In [297]:
```
clusters = clustering.labels_
print(clusters[:30])
```

```
[1 1 0 1 1 1 1 0 1 1 1 0 1 1 0 1 0 0 0 1 0 0 1 0 1 0 0 1 1 0]
```

In [298]:
```
data_train['clusters'] = clusters
cont_matrix = metrics.cluster.contingency_matrix(data_train['Party'], data_train['clusters'])
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[298]:
```
(2.0, 0.0)
```



In [299]:
```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.0872903358743292
0.4108600798092358
```

In [300]:
```
# K-means++, multiple iteration with different variables
X = data_train[['Percent White, not Hispanic or Latino','Percent Foreign Born','Percent Less th
# Standardize the attributes
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)
```

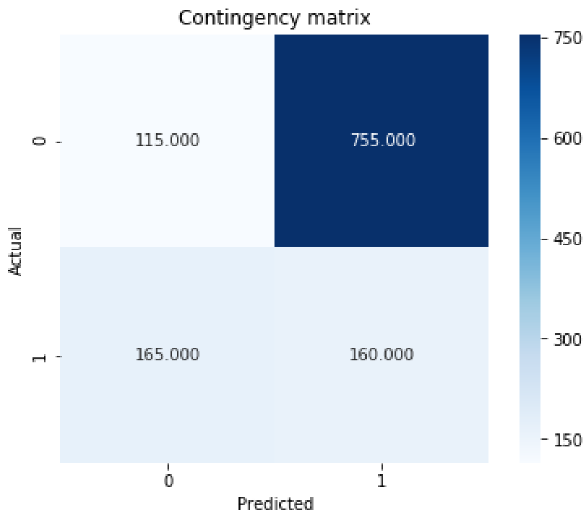
In [301]:
```
clustering = KMeans(n_clusters = 2, init = 'k-means++', n_init = 10).fit(X_scaled)
clusters = clustering.labels_
print(clusters[:30])
```

```
[1 0 0 1 1 1 0 1 1 0 0 1 0 0 0 0 0 1 1 0 1 0 1 0 0 0 1 0 0 1]
```

In [302]:

```
data_train['clusters'] = clusters
cont_matrix = metrics.cluster.contingency_matrix(data_train['Party'], data_train['clusters'])
ax=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()
bottom, top = ax.get_ylim()
ax.set_ylim(bottom + 0.5, top - 0.5)
```

Out[302]: (2.0, 0.0)



In [303]:

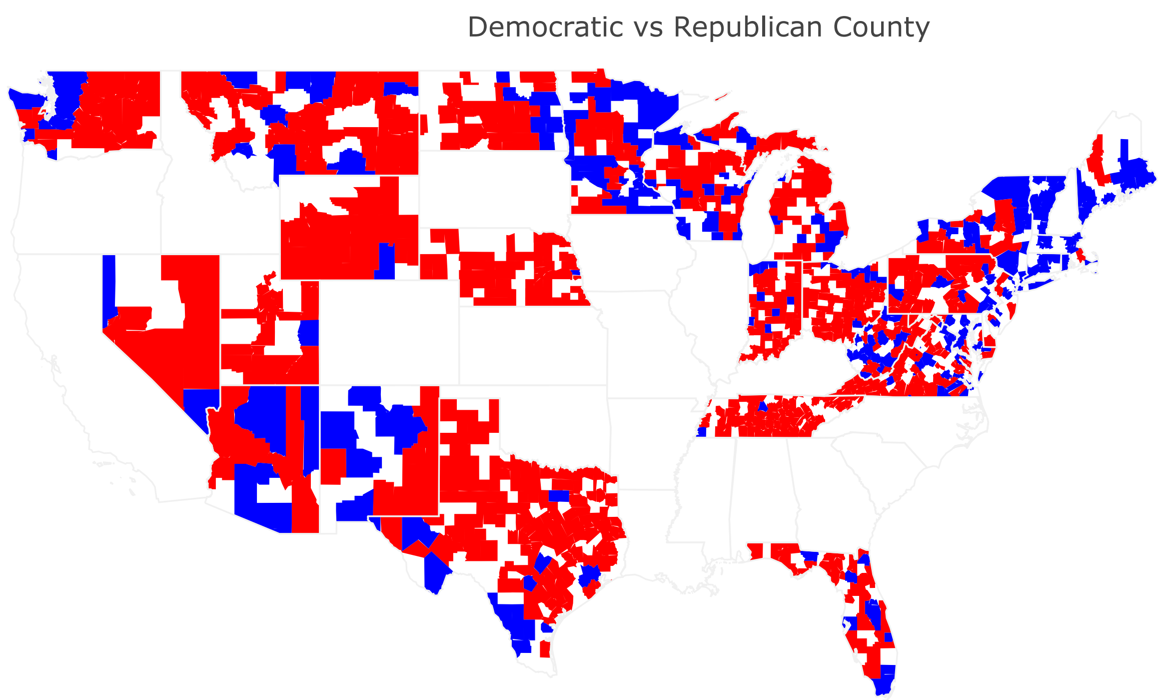
```
# Compute adjusted Rand index and silhouette coefficient
print(metrics.adjusted_rand_score(data_train['Party'], data_train['clusters']))
print(metrics.silhouette_score(X_scaled, data_train['clusters'], metric = "euclidean"))
```

```
0.24672990896646144
0.45221355231267657
```

Task 6

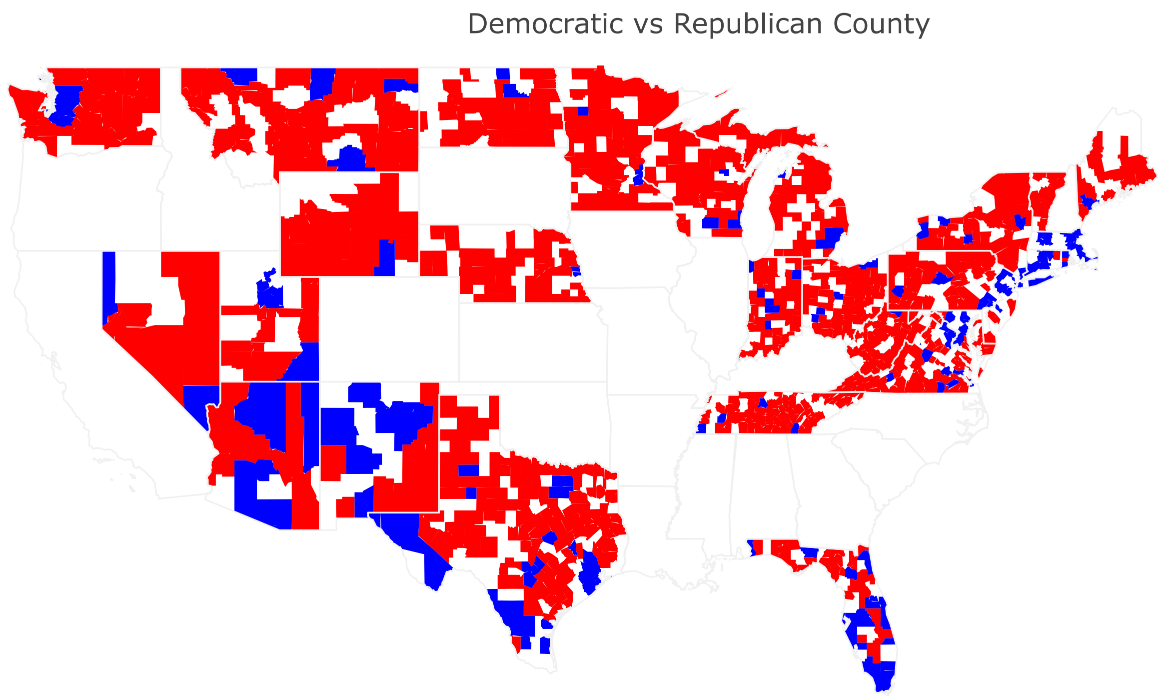
In [304]:

```
#Map from Project 1
#Creating a map of Democratic counties and Republican counties using the counties’ FIPS codes
data_train['FIPS'] = data_train['FIPS'].apply(lambda x: str(x).zfill(4))

colorscale = ["red","blue"]

fips = data_train['FIPS'].tolist()
values1 = data_train['Party'].tolist()


fig = ff.create_choropleth(
    fips = fips, values = values1,
    colorscale =  colorscale,
    show_state_data = True,
    show_hover = True, centroid_marker = {'opacity': 0},
    asp = 2.9, title = 'Democratic vs Republican County',
    legend_title = 'D(1) or R(0)'
)

fig.layout.template = None
fig.show()
```



In [305]:

```python
#Map for Project 2
#Creating a map of Democratic counties and Republican counties using the counties’ FIPS codes
#Using SVM classifier
data_train['FIPS'] = data_train['FIPS'].apply(lambda x: str(x).zfill(4))

colorscale = ["red","blue"]

fips = data_train['FIPS'].tolist()
values1 = y_pred_SVM.tolist()


fig = ff.create_choropleth(
    fips = fips, values = values1,
    colorscale =  colorscale,
    show_state_data = True,
    show_hover = True, centroid_marker = {'opacity': 0},
    asp = 2.9, title = 'Democratic vs Republican County',
    legend_title = 'D(1) or R(0)'
)

fig.layout.template = None
fig.show()
```



Task 7

In [306]:

```
data_test_X = data_test[['Total Population', "Percent Less than Bachelor's Degree", 'Percent Wh

data_test_Xdem = data_test[['Total Population', "Percent Less than Bachelor's Degree", 'Percent

data_test_Xrep = data_test[['Total Population', "Percent Less than Bachelor's Degree", 'Percent

scaler = StandardScaler()
scaler.fit(data_test_X)
data_test_X = scaler.transform(data_test_X)

scaler.fit(data_test_Xdem)
data_test_Xdem = scaler.transform(data_test_Xdem)

scaler.fit(data_test_Xrep)
data_test_Xrep = scaler.transform(data_test_Xrep)

y_pred_dem = best_dems.predict(data_test_Xdem)

y_pred_rep = best_repb.predict(data_test_Xrep)

y_pred = bestModel.predict(data_test_X)

data_test['Party'] = y_pred
data_test['Democratic'] = y_pred_dem
data_test['Republican'] = y_pred_rep
output = data_test[['State','County','Democratic','Republican','Party']]
output[['Democratic', 'Republican']] = output[['Democratic','Republican']].clip(lower = 0)

output.to_csv('Project2_output.csv',index=False,sep=',',encoding='utf-8')

output.head(1000)
```

Out[306]:

	State	County	Democratic	Republican	Party
0	NV	eureka	0.000000	0.000000	0
1	TX	zavala	0.000000	0.000000	0
2	VA	king george	9180.291751	4111.663448	0
3	OH	hamilton	242020.204400	153849.117662	1
4	TX	austin	336.646009	0.000000	0
...	...	...	...	...	...
395	VT	chittenden	61017.458161	42756.939923	1
396	OH	butler	107678.994980	70178.618865	0
397	NE	franklin	0.000000	7598.262280	0
398	MD	cecil	22984.988641	16177.600272	0
399	NY	yates	0.000000	11165.006159	0