In [2]:

```
# Import necessary libraries
import numpy as np
import pandas as pd
import math
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [3]:

```
# Read dataset
# file_merged_train = 'merged_train.csv'
merged_train = pd.read_csv("merged_train.csv")
merged_train.head()
```

	State	County	FIPS	Total Population	Percent White, not Hispanic or Latino	Percent Black, not Hispanic or Latino	Percent Hispanic or Latino	Percent Foreign Born	Percent Female	Percent Age 29 and Under	Percent Age 65 and Older	Median Household Income	Pe Unemp
0	AZ	apache	4001	72346	18.571863	0.486551	5.947806	1.719515	50.598513	45.854643	13.322091	32460	15.8074
1	AZ	cochise	4003	128177	56.299492	3.714395	34.403208	11.458374	49.069646	37.902276	19.756275	45383	8.56710
2	AZ	coconino	4005	138064	54.619597	1.342855	13.711033	4.825298	50.581614	48.946141	10.873943	51106	8.23830
3	AZ	gila	4007	53179	63.222325	0.552850	18.548675	4.249798	50.296170	32.238290	26.397638	40593	12.1299
4	AZ	graham	4009	37529	51.461536	1.811932	32.097844	4.385942	46.313518	46.393456	12.315809	47422	14.4241

In [4]:

```
# drop string columns for scaling and modeling.
nostr_merged_train = merged_train.drop(['State', 'County', 'FIPS'], axis=1)
nostr_merged_train.head()
```

	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	Pe Less Bach D
0	72346	18.571863	0.486551	5.947806	1.719515	50.598513	45.854643	13.322091	32460	15.807433	21.758252	88.94
1	128177	56.299492	3.714395	34.403208	11.458374	49.069646	37.902276	19.756275	45383	8.567108	13.409171	76.83
2	138064	54.619597	1.342855	13.711033	4.825298	50.581614	48.946141	10.873943	51106	8.238305	11.085381	65.79
3	53179	63.222325	0.552850	18.548675	4.249798	50.296170	32.238290	26.397638	40593	12.129932	15.729958	82.26
4	37529	51.461536	1.811932	32.097844	4.385942	46.313518	46.393456	12.315809	47422	14.424104	14.580797	86.67

In [5]:

```
# Separate nostr_merged_train data to X and Y
# split X, Y respectively to training and validation set
dataX = nostr_merged_train.iloc[:, 0: 13]
dataY = nostr_merged_train.iloc[:, 13:]
print(dataX.shape, dataY.shape)

train_X, val_X, train_Y, val_Y = train_test_split(dataX, dataY, test_size = 0.2, random_state = 0)
print(train_X.shape, train_Y.shape, val_X.shape, val_Y.shape)
```

```
(1195, 13) (1195, 3)
(956, 13) (956, 3) (239, 13) (239, 3)
```

In [6]:

```
# 2. (5 pts.) Standardize the training set and the validation set.
# Standardize using dataX and dataY
scaler = StandardScaler()
train_x_scaled = scaler.fit_transform(train_X)
val_x_scaled = scaler.transform(val_X)
print(train_x_scaled.shape, val_x_scaled.shape)
print("==================================================")
print(train_x_scaled[:3])
print("==================================================")
print(val_x_scaled[:3])
```

```
(956, 13) (239, 13)
==================================================
[[-0.18701568  0.74708765 -0.45480038 -0.56159755 -0.66018177 -0.77161007
  -0.11000247 -0.40238207  0.22866307 -0.87384834 -0.85252392  0.23449507
  -0.30737994]
 [-0.27513052  0.63120166 -0.50146982 -0.27150596 -0.42859999  0.29315204
   0.3705039  -0.51827927  0.47127639 -0.07120191 -0.05080592  0.53717504
   0.36963293]
 [-0.3527402  -2.26126882 -0.36741393  3.43238102  2.61845307 -1.24603239
   1.5861013  -0.79129639 -0.80664724 -1.6330594   2.98325257  1.17802891
  -0.84512886]]
==================================================
[[-0.35651604  0.8548356  -0.5241496  -0.59421429 -0.66990291 -0.25449504
  -2.16701127  2.17530202 -0.74822444  0.21611659 -1.05801318  0.19254557
   0.36222698]
 [-0.33481009 -2.64542893 -0.58495405 -0.31637384 -0.60228329  0.34481143
   2.29024997 -1.4695984  -0.50235111  3.74440657  0.05948146  0.6263627
   0.48927813]
 [ 0.56627276  0.27400887 -0.24022101 -0.04335719  0.10666134  0.38166967
  -0.28634091  0.17493655 -0.29293841  0.36478087 -0.3788276  -0.07234263
  -1.11617002]]
```

In [7]:

```
# 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. (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.

In [8]:

```
## Feature-selection
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.linear_model import LogisticRegression
scores = []
for i in range(2, 14):
  selector = SelectKBest(chi2, k=i)
  selector.fit(train_X, train_Y.iloc[:, 2])
  indices = selector.get_support(indices=True)
  model = LogisticRegression().fit(train_X.iloc[:, indices], train_Y.iloc[:, 2])
  acc = accuracy_score(val_Y.iloc[:, 2], model.predict(val_X.iloc[:, indices]))
  scores.append({'feature_num': i, 'indices': dataX.columns[selector.get_support(indices=True)], 'accuracy': acc})
```

In [9]:

```
## Print Features-selection Result
for i in range(12):
  print(scores[i])
```

```
{'feature_num': 2, 'indices': Index(['Total Population', 'Median Household Income'], dtype='object'), 'accuracy': 0.7531380753138075}
{'feature_num': 3, 'indices': Index(['Total Population', 'Median Household Income', 'Percent Rural'], dtype='object'), 'accuracy': 0.7615062761
506276}
{'feature_num': 4, 'indices': Index(['Total Population', 'Percent Black, not Hispanic or Latino',
       'Median Household Income', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7824267782426778}
{'feature_num': 5, 'indices': Index(['Total Population', 'Percent Black, not Hispanic or Latino',
       'Percent Foreign Born', 'Median Household Income', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7824267782426778}
{'feature_num': 6, 'indices': Index(['Total Population', 'Percent White, not Hispanic or Latino',
       'Percent Black, not Hispanic or Latino', 'Percent Foreign Born',
       'Median Household Income', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7740585774058577}
{'feature_num': 7, 'indices': Index(['Total Population', 'Percent White, not Hispanic or Latino',
       'Percent Black, not Hispanic or Latino', 'Percent Foreign Born',
       'Median Household Income', 'Percent Less than Bachelor's Degree',
       'Percent Rural'],
      dtype='object'), 'accuracy': 0.7656903765690377}
{'feature_num': 8, 'indices': Index(['Total Population', 'Percent White, not Hispanic or Latino',
       'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino',
       'Percent Foreign Born', 'Median Household Income',
       'Percent Less than Bachelor's Degree', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7698744769874477}
{'feature_num': 9, 'indices': Index(['Total Population', 'Percent White, not Hispanic or Latino',
       'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino',
       'Percent Foreign Born', 'Median Household Income',
       'Percent Less than High School Degree',
       'Percent Less than Bachelor's Degree', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7782426778242678}
{'feature_num': 10, 'indices': Index(['Total Population', 'Percent White, not Hispanic or Latino',
       'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino',
       'Percent Foreign Born', 'Percent Age 65 and Older',
       'Median Household Income', 'Percent Less than High School Degree',
       'Percent Less than Bachelor's Degree', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7782426778242678}
{'feature_num': 11, 'indices': Index(['Total Population', 'Percent White, not Hispanic or Latino',
       'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino',
       'Percent Foreign Born', 'Percent Age 29 and Under',
       'Percent Age 65 and Older', 'Median Household Income',
       'Percent Less than High School Degree',
       'Percent Less than Bachelor's Degree', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7782426778242678}
{'feature_num': 12, 'indices': Index(['Total Population', 'Percent White, not Hispanic or Latino',
       'Percent Black, not Hispanic or Latino', 'Percent Hispanic or Latino',
       'Percent Foreign Born', 'Percent Age 29 and Under',
       'Percent Age 65 and Older', 'Median Household Income',
       'Percent Unemployed', 'Percent Less than High School Degree',
       'Percent Less than Bachelor's Degree', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7782426778242678}
{'feature_num': 13, 'indices': Index(['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',
       'Percent Less than Bachelor's Degree', 'Percent Rural'],
      dtype='object'), 'accuracy': 0.7782426778242678}
```

In [10]:
```
# Train the models with these three combinations
# Select all features
combine1_X = train_x_scaled
val1_X = val_x_scaled

# Select features from the result of SelectKBest
# ['Total Population', 'Percent Black, not Hispanic or Latino', 'Median Household Income', 'Percent Rural']
combine2_X = train_x_scaled[:, [0, 2, 3, 8, 12]]
val2_X = val_x_scaled[:, [0, 2, 3, 8, 12]]

# Select features project 01's conclusion
# ['Total Population', 'Percent White', 'Percent Black', 'hispanic or latino', 'bachelor's Degree', 'Percent Rural']
combine3_X = train_x_scaled[:, [0, 1, 2, 3, 11, 12]]
val3_X = val_x_scaled[:, [0, 1, 2, 3, 11, 12]]

print(combine1_X.shape, combine2_X.shape, combine3_X.shape)
print(val1_X.shape, val2_X.shape, val3_X.shape)
```

```
(956, 13) (956, 5) (956, 6)
(239, 13) (239, 5) (239, 6)
```

In [11]:
```
# import regression models
from sklearn.linear_model import Lasso
from sklearn.linear_model import Ridge
from sklearn.linear_model import ElasticNet
from sklearn.metrics import r2_score
```

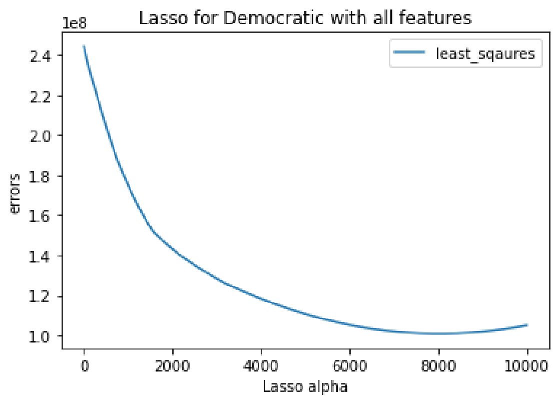
In [12]:
```
def least_sqaures(Y, Y_pred):
  diff = Y - Y_pred
  return np.sum(diff.T.dot(diff)) / len(Y)
```

In [13]:
```
def adj_R_squares(Y,Y_pred, num_features=13):
  return 1 - (1 - r2_score(Y, Y_pred)) * ((len(Y) - 1) / (len(Y) - num_features - 1))
```

In [14]:

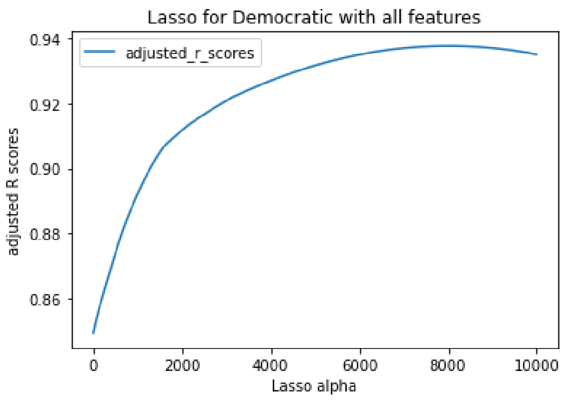
```
# Lasso, Democratic
# All features
alphas = np.arange(0, 10000, 10)
errors = []
adj_scores = []
for alpha in alphas:
  model = Lasso(alpha=alpha)
  model.fit(combine1_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val1_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val1_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('Lasso alpha')
plt.ylabel('errors')
plt.title('Lasso for Democratic with all features')
plt.legend()
plt.show()
```

```
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 347444562241.33057, tolerance: 587697861.197598
  positive)
```



In [15]:

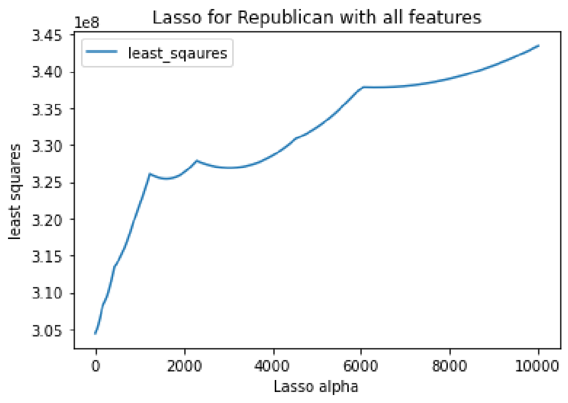
```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Lasso alpha')
plt.ylabel('adjusted R scores')
plt.title('Lasso for Democratic with all features')
plt.legend()
plt.show()
```



In [16]:

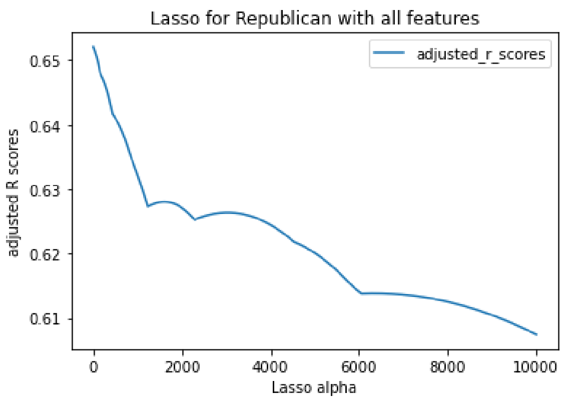
```
# Lasso, Republican
# All features
alphas = np.arange(0, 10000, 1)
errors = []
adj_scores = []
for alpha in alphas:
  model = Lasso(alpha=alpha)
  model.fit(combine1_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val1_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val1_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('Lasso alpha')
plt.ylabel('least squares')
plt.title('Lasso for Republican with all features')
plt.legend()
plt.show()
```

```
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 146635229211.19183, tolerance: 222252571.2967234
  positive)
```



In [17]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Lasso alpha')
plt.ylabel('adjusted R scores')
plt.title('Lasso for Republican with all features')
plt.legend()
plt.show()
```



In [18]:

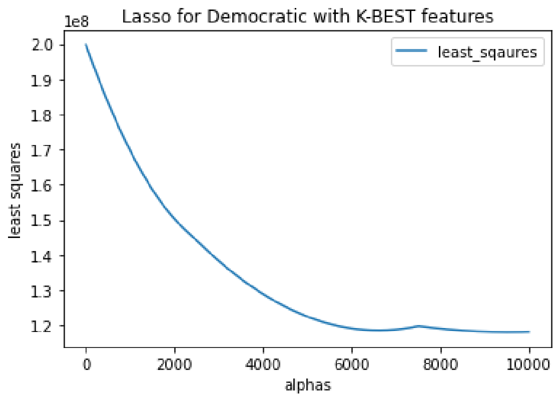
```
# BEST LASSO RESULT FOR DEMOCRATIC with ALL FEATURES -> alpha: 8000
model = Lasso(alpha=8000).fit(combine1_X, train_Y.iloc[:, 0])
print("BEST LASSO RESULT FOR DEMOCRATIC with ALL FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(val1_X
)))
# BEST LASSO RESULT FOR REPUBLICAN with ALL FEATURES -> alpha: 1500
model = Lasso(alpha=0).fit(combine1_X, train_Y.iloc[:, 1])
print("BEST LASSO RESULT FOR REPUBLICAN with ALL FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(val1_X
)))
```

```
BEST LASSO RESULT FOR DEMOCRATIC with ALL FEATURES:  100865087.30148067
BEST LASSO RESULT FOR REPUBLICAN with ALL FEATURES:  304387874.10731435


C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:5: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator
  """
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 146635229211.19183, tolerance: 222252571.2967234
  positive)
```

In [19]:

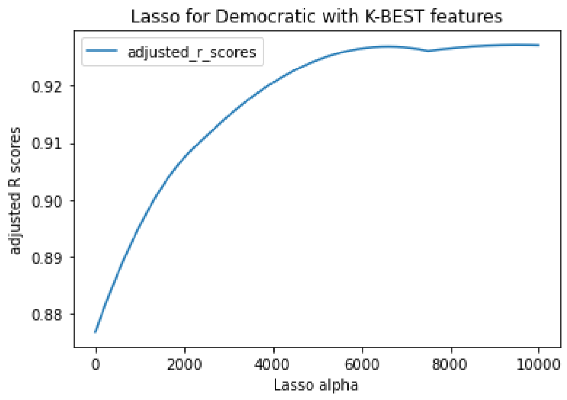
```
# Lasso, Democratic
# K-BEST FEATURES
alphas = np.arange(0, 10000, 10)
errors = []
adj_scores = []
for alpha in alphas:
  model = Lasso(alpha=alpha)
  model.fit(combine2_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val2_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val2_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alphas')
plt.ylabel('least squares')
plt.title('Lasso for Democratic with K-BEST features')
plt.legend()
plt.show()
```

```
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩ipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 374455061537.33545, tolerance: 587697861.197598
  positive)
```



In [20]:

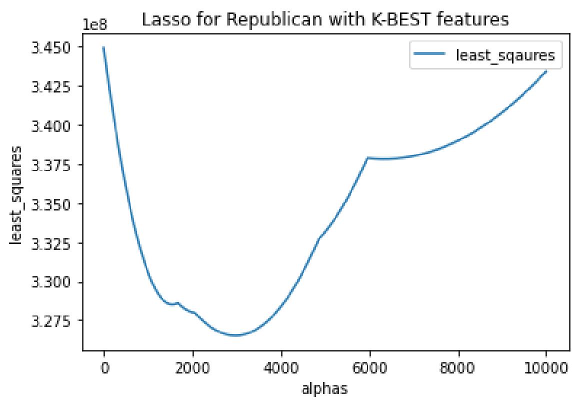
```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Lasso alpha')
plt.ylabel('adjusted R scores')
plt.title('Lasso for Democratic with K-BEST features')
plt.legend()
plt.show()
```



In [21]:

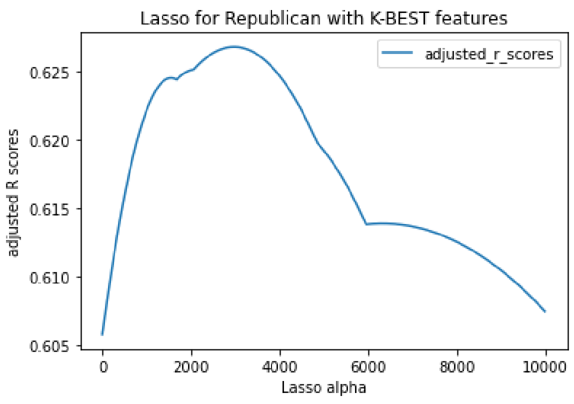
```python
# Lasso, Republican
# All features
alphas = np.arange(0, 10000, 1)
errors = []
adj_scores = []
for alpha in alphas:
  model = Lasso(alpha=alpha)
  model.fit(combine2_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val2_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val2_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('Lasso for Republican with K-BEST features')
plt.legend()
plt.show()
```

```
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩ipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 156813659970.35706, tolerance: 222252571.2967234
  positive)
```



In [22]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Lasso alpha')
plt.ylabel('adjusted R scores')
plt.title('Lasso for Republican with K-BEST features')
plt.legend()
plt.show()
```



In [23]:

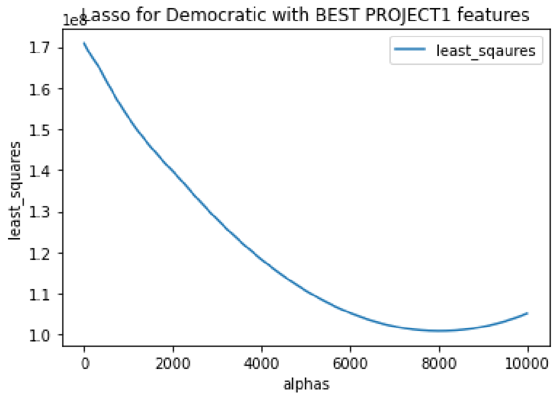
```
# BEST LASSO RESULT FOR DEMOCRATIC with K-BEST FEATURES -> alpha: 8000
model = Lasso(alpha=6500).fit(combine2_X, train_Y.iloc[:, 0])
print("BEST LASSO RESULT FOR DEMOCRATIC with K-BEST FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(val2_
X)))
# BEST LASSO RESULT FOR REPUBLICAN with K-BEST FEATURES -> alpha: 3000
model = Lasso(alpha=3000).fit(combine2_X, train_Y.iloc[:, 1])
print("BEST LASSO RESULT FOR REPUBLICAN with K-BEST FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(val2_
X)))
```

```
BEST LASSO RESULT FOR DEMOCRATIC with K-BEST FEATURES:  118593552.22677654
BEST LASSO RESULT FOR REPUBLICAN with K-BEST FEATURES:  326531458.16985345
```

In [24]:

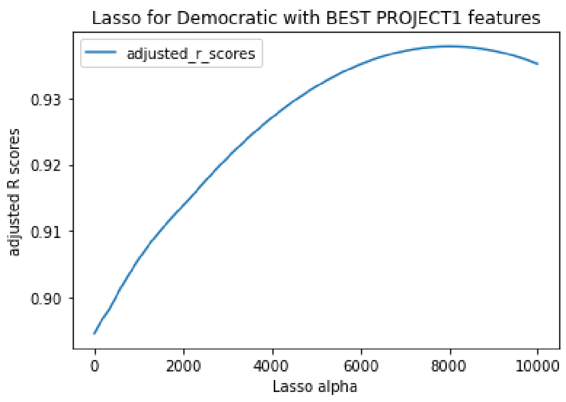
```
# Lasso, Democratic
# BEST PRJECT1 FEATURES
alphas = np.arange(0, 10000, 10)
errors = []
adj_scores = []
for alpha in alphas:
  model = Lasso(alpha=alpha)
  model.fit(combine3_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val3_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val3_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('Lasso for Democratic with BEST PROJECT1 features')
plt.legend()
plt.show()
```

```
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩ipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 360013413141.62463, tolerance: 587697861.197598
  positive)
```



In [25]:

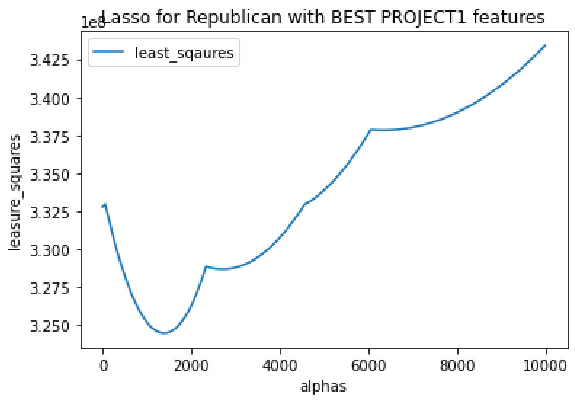
```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Lasso alpha')
plt.ylabel('adjusted R scores')
plt.title('Lasso for Democratic with BEST PROJECT1 features')
plt.legend()
plt.show()
```



In [26]:

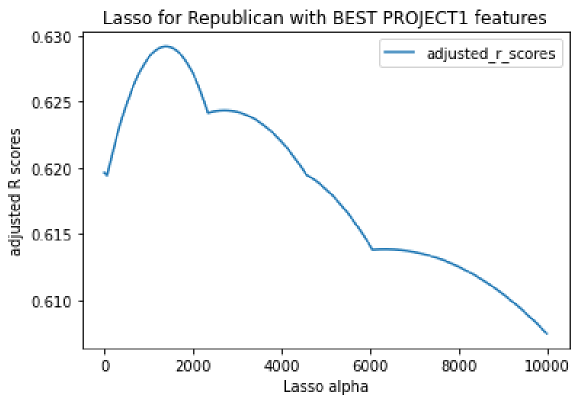
```
# Lasso, Republican
# BEST PRJECT1 FEATURES
alphas = np.arange(0, 10000, 10)
errors = []
adj_scores = []
for alpha in alphas:
  model = Lasso(alpha=alpha)
  model.fit(combine3_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val3_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val3_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alphas')
plt.ylabel('leasure_squares')
plt.title('Lasso for Republican with BEST PROJECT1 features')
plt.legend()
plt.show()
```

```
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩ipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 158156281687.72296, tolerance: 222252571.2967234
  positive)
```



In [27]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Lasso alpha')
plt.ylabel('adjusted R scores')
plt.title('Lasso for Republican with BEST PROJECT1 features')
plt.legend()
plt.show()
```



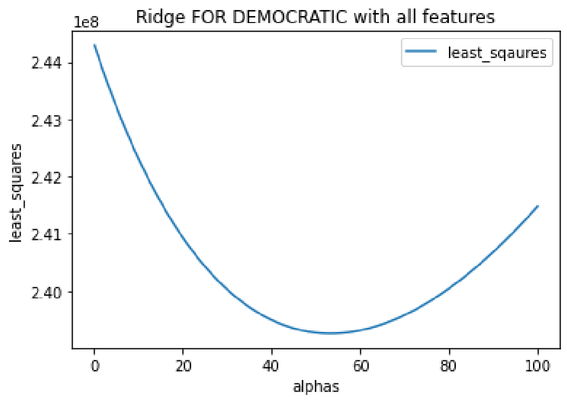
In [28]:

```
# BEST LASSO RESULT FOR DEMOCRATIC with PROJECT1 FEATURES -> alpha: 8000
model = Lasso(alpha=8000).fit(combine3_X, train_Y.iloc[:, 0])
print("BEST LASSO RESULT FOR DEMOCRATIC with PROJECT1 FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(val
3_X)))
# BEST LASSO RESULT FOR REPUBLICAN with PROJECT1 FEATURES -> alpha: 0
model = Lasso(alpha=1800).fit(combine3_X, train_Y.iloc[:, 1])
print("BEST LASSO RESULT FOR REPUBLICAN with PROJECT1 FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(val
3_X)))
```

```
BEST LASSO RESULT FOR DEMOCRATIC with PROJECT1 FEATURES:  100865087.30148067
BEST LASSO RESULT FOR REPUBLICAN with PROJECT1 FEATURES:  325228770.8586026
```

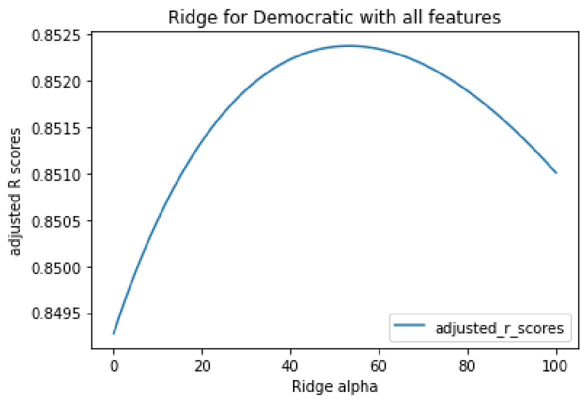
In [29]:

```
# Ridge, DEMOCRATIC
# ALL FEATURES
alphas = np.arange(0.1, 100, 0.01)
errors = []
adj_scores = []
for alpha in alphas:
  model = Ridge(alpha=alpha)
  model.fit(combine1_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val1_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val1_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('Ridge FOR DEMOCRATIC with all features')
plt.legend()
plt.show()
```



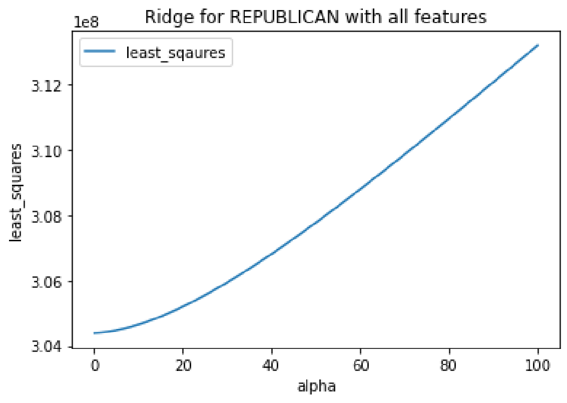
In [30]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Ridge alpha')
plt.ylabel('adjusted R scores')
plt.title('Ridge for Democratic with all features')
plt.legend()
plt.show()
```



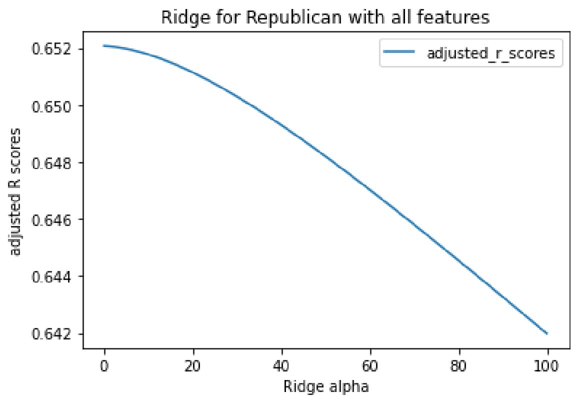
In [31]:

```
# Ridge, REPUBLICAN
# ALL FEATURES
alphas = np.arange(0.1, 100, 0.01)
errors = []
adj_scores = []
for alpha in alphas:
  model = Ridge(alpha=alpha)
  model.fit(combine1_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val1_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val1_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alpha')
plt.ylabel('least_squares')
plt.title('Ridge for REPUBLICAN with all features')
plt.legend()
plt.show()
```



In [32]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Ridge alpha')
plt.ylabel('adjusted R scores')
plt.title('Ridge for Republican with all features')
plt.legend()
plt.show()
```



In [33]:

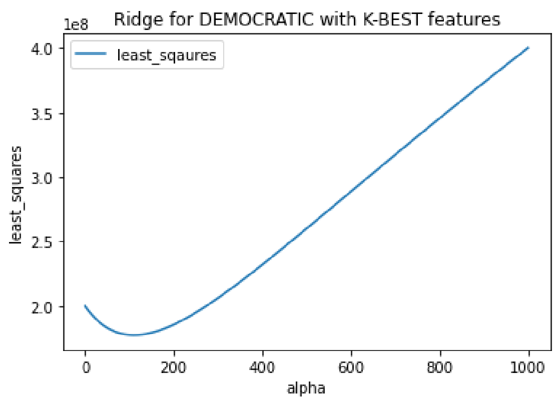
```
# BEST RIDGE RESULT FOR DEMOCRATIC with ALL FEATURES -> alpha: 50
model = Lasso(alpha=50).fit(combine1_X, train_Y.iloc[:, 0])
print("BEST RIDGE RESULT FOR DEMOCRATIC with ALL FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(val1_X
)))
# BEST RIDGE RESULT FOR REPUBLICAN with ALL FEATURES -> alpha: 0
model = Lasso(alpha=0).fit(combine1_X, train_Y.iloc[:, 1])
print("BEST RIDGE RESULT FOR REPUBLICAN with ALL FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(val1_X
)))
```

```
BEST RIDGE RESULT FOR DEMOCRATIC with ALL FEATURES:  239231766.62366772
BEST RIDGE RESULT FOR REPUBLICAN with ALL FEATURES:  304387874.10731435
```

```
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩ipykernel_launcher.py:5: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator
  """
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 146635229211.19183, tolerance: 222252571.2967234
  positive)
```

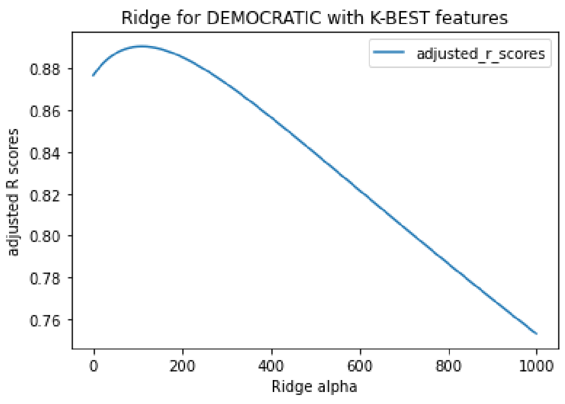
In [34]:

```
# Ridge, DEMOCRATIC
# K-BEST FEATURES
alphas = np.arange(0, 1000, 1)
errors = []
adj_scores = []
for alpha in alphas:
  model = Ridge(alpha=alpha)
  model.fit(combine2_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val2_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val2_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alpha')
plt.ylabel('least_squares')
plt.title('Ridge for DEMOCRATIC with K-BEST features')
plt.legend()
plt.show()
```



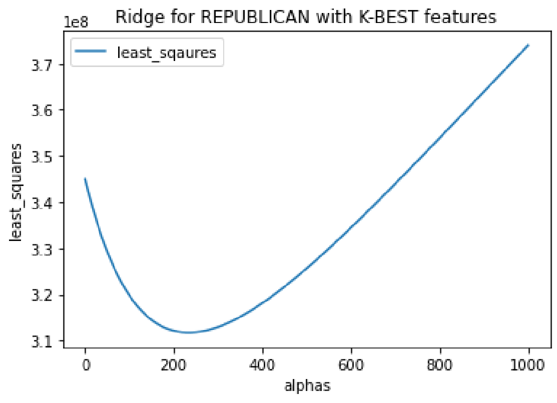
In [35]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Ridge alpha')
plt.ylabel('adjusted R scores')
plt.title('Ridge for DEMOCRATIC with K-BEST features')
plt.legend()
plt.show()
```



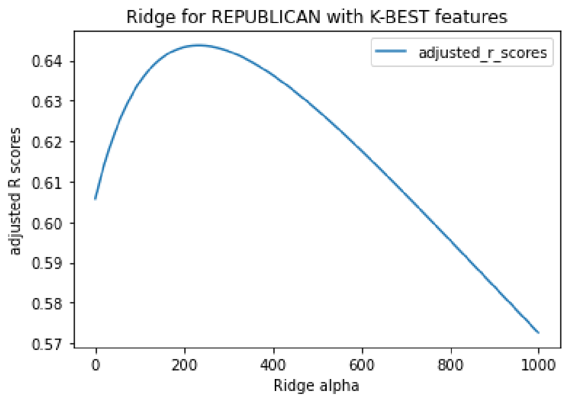
In [36]:

```
# Ridge, REPUBLICAN
# K-BEST FEATURES
alphas = np.arange(0, 1000, 1)
errors = []
adj_scores = []
for alpha in alphas:
  model = Ridge(alpha=alpha)
  model.fit(combine2_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val2_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val2_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('Ridge for REPUBLICAN with K-BEST features')
plt.legend()
plt.show()
```



In [37]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Ridge alpha')
plt.ylabel('adjusted R scores')
plt.title('Ridge for REPUBLICAN with K-BEST features')
plt.legend()
plt.show()
```



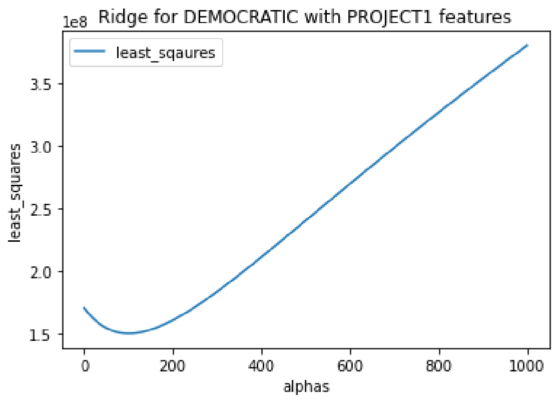
In [38]:

```
# BEST RIDGE RESULT FOR DEMOCRATIC with K-BEST FEATURES -> alpha: 100
model = Ridge(alpha=100).fit(combine2_X, train_Y.iloc[:, 0])
print("BEST RIDGE RESULT FOR DEMOCRATIC with K-BEST FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(val2_
X)))
# BEST RIDGE RESULT FOR REPUBLICAN with K-BEST FEATURES -> alpha: 250
model = Ridge(alpha=250).fit(combine2_X, train_Y.iloc[:, 1])
print("BEST RIDGE RESULT FOR REPUBLICAN with K-BEST FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(val2_
X)))
```

```
BEST RIDGE RESULT FOR DEMOCRATIC with K-BEST FEATURES:  177411709.05204254
BEST RIDGE RESULT FOR REPUBLICAN with K-BEST FEATURES:  311782825.54062074
```

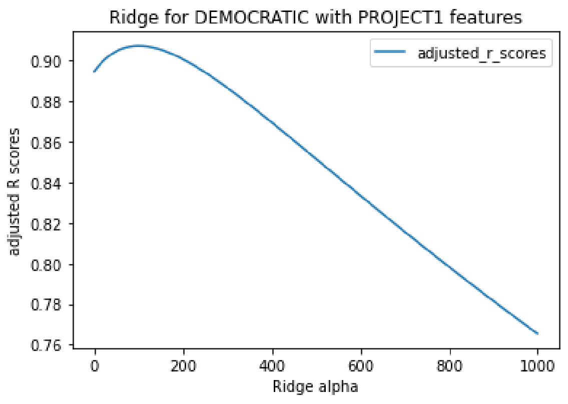
In [39]:

```
# Ridge, DEMOCRATIC
# PROJECT1 FEATURES
alphas = np.arange(0, 1000, 1)
errors = []
adj_scores = []
for alpha in alphas:
  model = Ridge(alpha=alpha)
  model.fit(combine3_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val3_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val3_X)))
plt.plot(alphas, errors, label='least_sqaures')
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('Ridge for DEMOCRATIC with PROJECT1 features')
plt.legend()
plt.show()
```



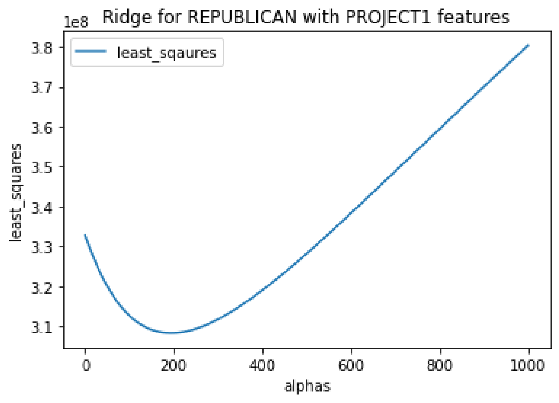
In [40]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Ridge alpha')
plt.ylabel('adjusted R scores')
plt.title('Ridge for DEMOCRATIC with PROJECT1 features')
plt.legend()
plt.show()
```



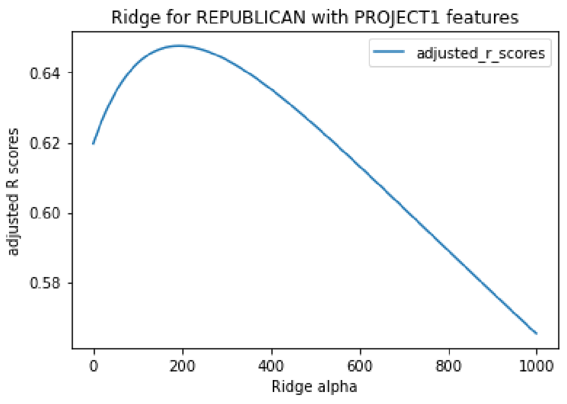
In [41]:

```
# Ridge, REPUBLICAN
# PROJECT1 FEATURES
alphas = np.arange(0, 1000, 1)
errors = []
adj_scores = []
for alpha in alphas:
  model = Ridge(alpha=alpha)
  model.fit(combine3_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val3_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val3_X)))
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('Ridge for REPUBLICAN with PROJECT1 features')
plt.plot(alphas, errors, label='least_sqaures')
plt.legend()
plt.show()
```



In [42]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('Ridge alpha')
plt.ylabel('adjusted R scores')
plt.title('Ridge for REPUBLICAN with PROJECT1 features')
plt.legend()
plt.show()
```



In [43]:

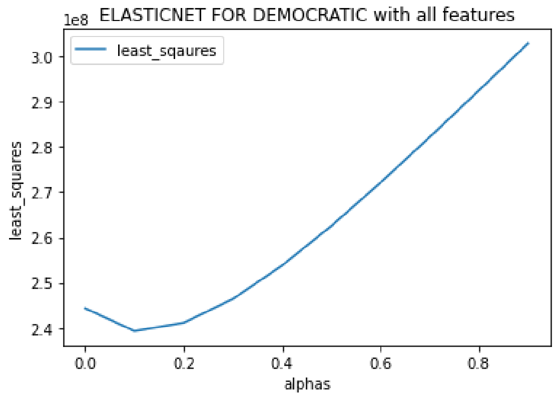
```
# BEST RIDGE RESULT FOR DEMOCRATIC with PROJECT1 FEATURES -> alpha: 100
model = Ridge(alpha=100).fit(combine3_X, train_Y.iloc[:, 0])
print("BEST RIDGE RESULT FOR DEMOCRATIC with PROJECT1 FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(val
3_X)))
# BEST RIDGE RESULT FOR REPUBLICAN with PROJECT1 FEATURES -> alpha: 200
model = Ridge(alpha=200).fit(combine3_X, train_Y.iloc[:, 1])
print("BEST RIDGE RESULT FOR REPUBLICAN with PROJECT1 FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(val
3_X)))
```

```
BEST RIDGE RESULT FOR DEMOCRATIC with PROJECT1 FEATURES:  150430879.68429667
BEST RIDGE RESULT FOR REPUBLICAN with PROJECT1 FEATURES:  308311779.2562058
```

In [44]:

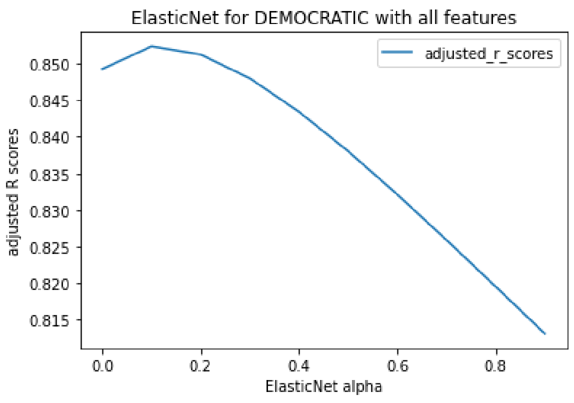
```
# ELASTICNET, DEMOCRATIC
# ALL FEATURES
alphas = np.arange(0, 1, 0.1)
errors = []
adj_scores = []
for alpha in alphas:
  model = ElasticNet(alpha=alpha)
  model.fit(combine1_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val1_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val1_X)))
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('ELASTICNET FOR DEMOCRATIC with all features')
plt.plot(alphas, errors, label='least_sqaures')
plt.legend()
plt.show()
```

```
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 347444562241.33057, tolerance: 587697861.197598
  positive)
```



In [45]:

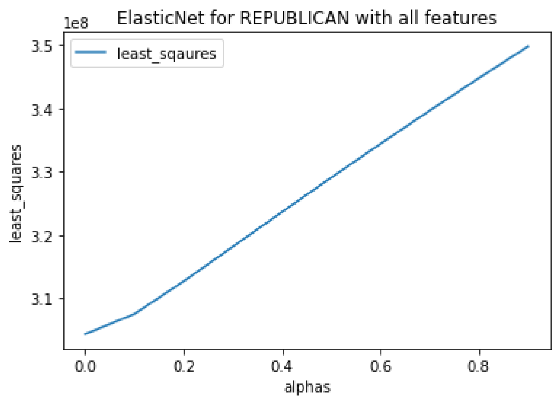
```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('ElasticNet alpha')
plt.ylabel('adjusted R scores')
plt.title('ElasticNet for DEMOCRATIC with all features')
plt.legend()
plt.show()
```



In [46]:

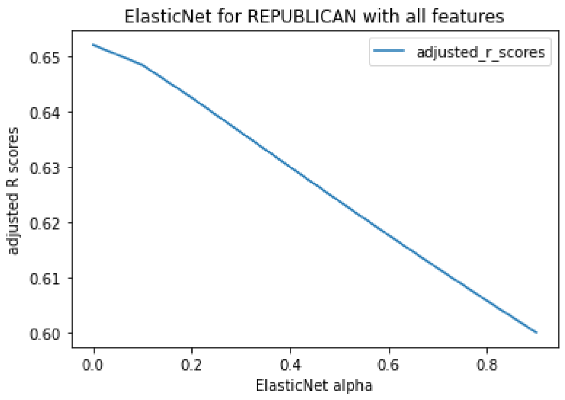
```
# ElasticNEt, REPUBLICAN
# ALL FEATURES
alphas = np.arange(0, 1, 0.1)
errors = []
adj_scores = []
for alpha in alphas:
  model = ElasticNet(alpha=alpha)
  model.fit(combine1_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val1_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val1_X)))
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('ElasticNet for REPUBLICAN with all features')
plt.plot(alphas, errors, label='least_sqaures')
plt.legend()
plt.show()
```

```
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 146635229211.19183, tolerance: 222252571.2967234
  positive)
```



In [47]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('ElasticNet alpha')
plt.ylabel('adjusted R scores')
plt.title('ElasticNet for REPUBLICAN with all features')
plt.legend()
plt.show()
```



In [48]:

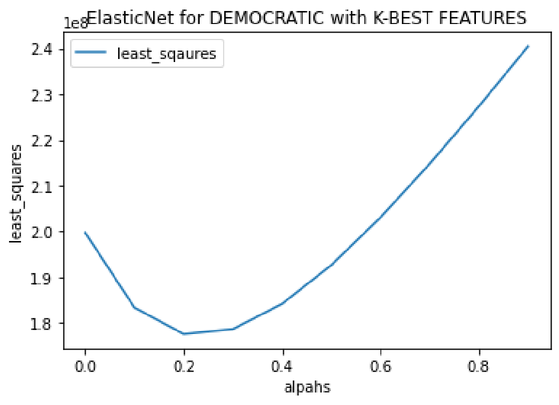
```
# BEST ELASTICNET RESULT FOR DEMOCRATIC with ALL FEATURES -> alpha: 0.001
model = ElasticNet(alpha=0.001).fit(combine1_X, train_Y.iloc[:, 0])
print("BEST ELASTICNET RESULT FOR DEMOCRATIC with ALL FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(val
1_X)))
# BEST ELASTICNET RESULT FOR REPUBLICAN with ALL FEATURES -> alpha: 0.001
model = ElasticNet(alpha=0.001).fit(combine1_X, train_Y.iloc[:, 1])
print("BEST ELASTICNET RESULT FOR REPUBLICAN with ALL FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(val
1_X)))
```

```
BEST ELASTICNET RESULT FOR DEMOCRATIC with ALL FEATURES:  244197956.62761435
BEST ELASTICNET RESULT FOR REPUBLICAN with ALL FEATURES:  304391890.3567718
```

In [49]:

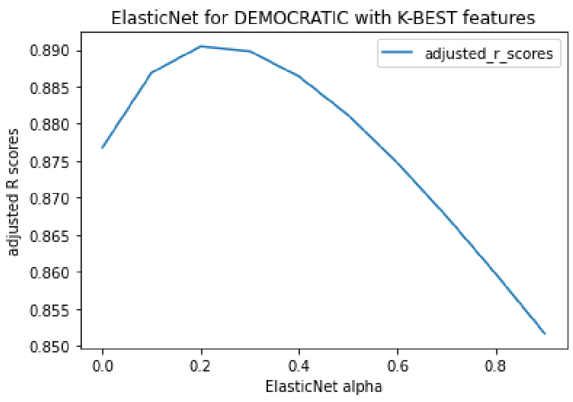
```
# ElasticNet, DEMOCRATIC
# K-BEST FEATURES
alphas = np.arange(0, 1, 0.1)
errors = []
adj_scores = []
for alpha in alphas:
  model = ElasticNet(alpha=alpha)
  model.fit(combine2_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val2_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val2_X)))
plt.xlabel('alpahs')
plt.ylabel('least_squares')
plt.title('ElasticNet for DEMOCRATIC with K-BEST FEATURES')
plt.plot(alphas, errors, label='least_sqaures')
plt.legend()
plt.show()
```

```
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩ipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 374455061537.33545, tolerance: 587697861.197598
  positive)
```



In [50]:

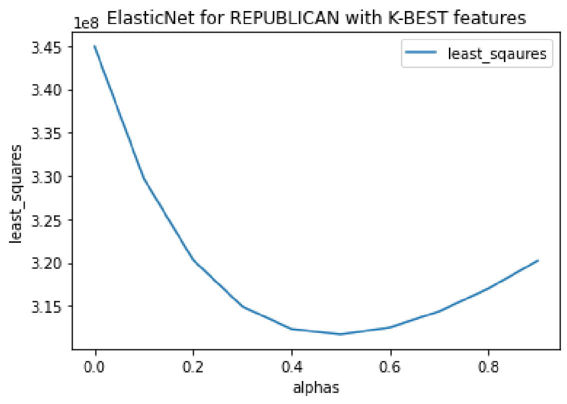
```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('ElasticNet alpha')
plt.ylabel('adjusted R scores')
plt.title('ElasticNet for DEMOCRATIC with K-BEST features')
plt.legend()
plt.show()
```



In [51]:

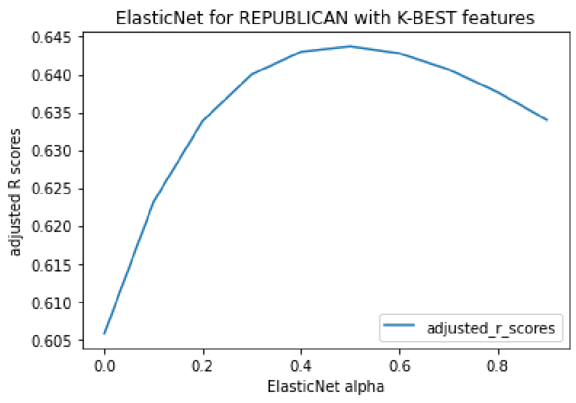
```
# ELASTICNET, REPUBLICAN
# K-BEST FEATURES
alphas = np.arange(0, 1, 0.1)
errors = []
adj_scores = []
for alpha in alphas:
  model = ElasticNet(alpha=alpha)
  model.fit(combine2_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val2_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val2_X)))
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('ElasticNet for REPUBLICAN with K-BEST features')
plt.plot(alphas, errors, label='least_sqaures')
plt.legend()
plt.show()
```

```
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 156813659970.35706, tolerance: 222252571.2967234
  positive)
```



In [52]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('ElasticNet alpha')
plt.ylabel('adjusted R scores')
plt.title('ElasticNet for REPUBLICAN with K-BEST features')
plt.legend()
plt.show()
```



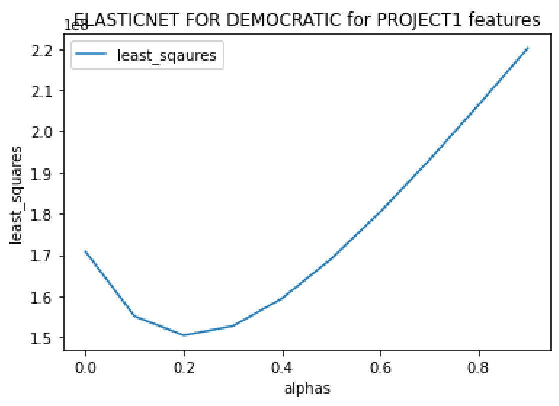
In [53]:

```
# BEST ELASTICNET RESULT FOR DEMOCRATIC with K-BEST FEATURES -> alpha: 0.2
model = ElasticNet(alpha=0.1).fit(combine2_X, train_Y.iloc[:, 0])
print("BEST ELASTICNET RESULT FOR DEMOCRATIC with K-BEST FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predict(
val2_X)))
# BEST ELASTICNET RESULT FOR REPUBLICAN with K-BEST FEATURES -> alpha: 0.5
model = ElasticNet(alpha=0.5).fit(combine2_X, train_Y.iloc[:, 1])
print("BEST ELASTICNET RESULT FOR REPUBLICAN with K-BEST FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predict(
val2_X)))
```

```
BEST ELASTICNET RESULT FOR DEMOCRATIC with K-BEST FEATURES:  183325934.72063282
BEST ELASTICNET RESULT FOR REPUBLICAN with K-BEST FEATURES:  311706923.61274904
```

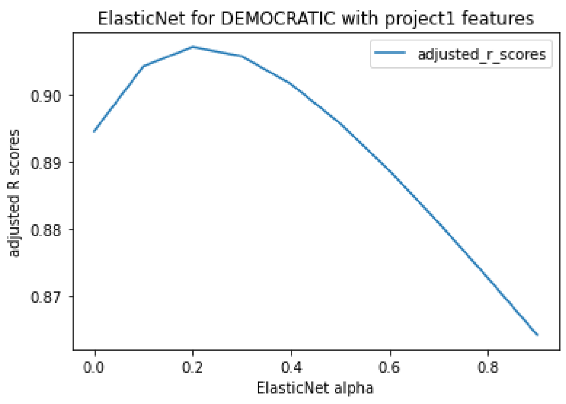
```
In [54]:
# ELASTICNET, DEMOCRATIC
# PROJECT1 FEATURES
alphas = np.arange(0, 1, 0.1)
errors = []
adj_scores = []
for alpha in alphas:
  model = ElasticNet(alpha=alpha)
  model.fit(combine3_X, train_Y.iloc[:, 0])
  errors.append(least_sqaures(val_Y.iloc[:, 0], model.predict(val3_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 0], model.predict(val3_X)))
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('ELASTICNET FOR DEMOCRATIC for PROJECT1 features')
plt.plot(alphas, errors, label='least_sqaures')
plt.legend()
plt.show()
```

```
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩ipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:₩Users₩khhh9₩Anaconda3₩envs₩ml_project₩lib₩site-packages₩sklearn₩linear_model₩_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 360013413141.62463, tolerance: 587697861.197598
  positive)
```



In [55]:

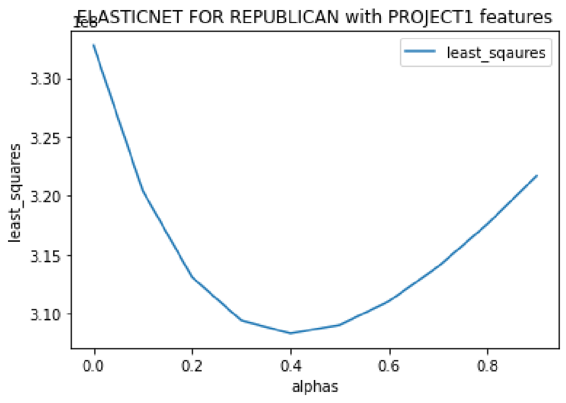
```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('ElasticNet alpha')
plt.ylabel('adjusted R scores')
plt.title('ElasticNet for DEMOCRATIC with project1 features')
plt.legend()
plt.show()
```



In [56]:

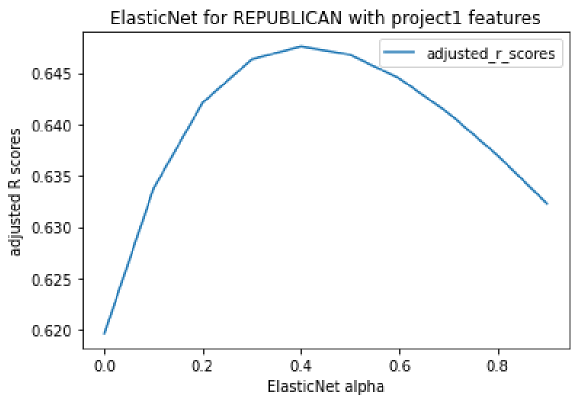
```
# ELASTICNET, REPUBLICAN
# PROJECT1 FEATURES
alphas = np.arange(0, 1, 0.1)
errors = []
adj_scores = []
for alpha in alphas:
  model = ElasticNet(alpha=alpha)
  model.fit(combine3_X, train_Y.iloc[:, 1])
  errors.append(least_sqaures(val_Y.iloc[:, 1], model.predict(val3_X)))
  adj_scores.append(adj_R_squares(val_Y.iloc[:, 1], model.predict(val3_X)))
plt.xlabel('alphas')
plt.ylabel('least_squares')
plt.title('ELASTICNET FOR REPUBLICAN with PROJECT1 features')
plt.plot(alphas, errors, label='least_sqaures')
plt.legend()
plt.show()
```

```
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:8: UserWarning: With alpha=0, this algorithm does not converge
well. You are advised to use the LinearRegression estimator

C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning: Coordinate descent wit
h no regularization may lead to unexpected results and is discouraged.
  positive)
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: ConvergenceWarning: Objective did n
ot converge. You might want to increase the number of iterations. Duality gap: 158156281687.72296, tolerance: 222252571.2967234
  positive)
```



In [57]:

```
plt.plot(alphas, adj_scores, label='adjusted_r_scores')
plt.xlabel('ElasticNet alpha')
plt.ylabel('adjusted R scores')
plt.title('ElasticNet for REPUBLICAN with project1 features')
plt.legend()
plt.show()
```



In [58]:

```
# BEST ELASTICNET RESULT FOR DEMOCRATIC with PROJECT1 FEATURES -> alpha: 0.2
model = ElasticNet(alpha=0.2).fit(combine3_X, train_Y.iloc[:, 0])
print("BEST ELASTICNET RESULT FOR DEMOCRATIC with PROJECT1 FEATURES: ", least_sqaures(val_Y.iloc[:, 0], model.predic
t(val3_X)))
# BEST ELASTICNET RESULT FOR REPUBLICAN with PROJECT1 FEATURES -> alpha: 0.4
model = ElasticNet(alpha=0.4).fit(combine3_X, train_Y.iloc[:, 1])
print("BEST ELASTICNET RESULT FOR REPUBLICAN with PROJECT1 FEATURES: ", least_sqaures(val_Y.iloc[:, 1], model.predic
t(val3_X)))
```

```
BEST ELASTICNET RESULT FOR DEMOCRATIC with PROJECT1 FEATURES:  150462802.58968845
BEST ELASTICNET RESULT FOR REPUBLICAN with PROJECT1 FEATURES:  308297882.7715332
```

In [59]:

```
# BEST RESULT FOR LINEAR REGRESSION
# FOR DEMOCRATIC : LASSO, PROJECT1 FEATRUES, alpha: 8000
# FOR REPUBLICAN : ELASTICNET, ALL FEATURES, alpha: 0
```

1. (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?

In [60]:

```
# Import classification models
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)
```

In [61]:

```
# Label data for classification
y_train = train_Y.iloc[:, 2]
y_test = val_Y.iloc[:, 2]

print(y_train.shape, y_test.shape)
```

```
(956,) (239,)
```

In [62]:

```
# Parameter Tuning to find relative good results
def model_SVC(kernel, degree, n_iter):
  #kernel: 'linear',  ‘poly’ ,  ‘rbf’ ; poly-degree: 3, 5; max_iter: default(-1), 100, 200, 300
  if kernel == 'poly':
    model = SVC(kernel = kernel, degree = degree, max_iter = n_iter)
  else:
    model = SVC(kernel = kernel, max_iter = n_iter)
  return model

def model_KNN(n_neighbors, weights, p):
  #n_neighbors: 5, 10, 15; weights{ ‘uniform’ ,  ‘distance’ }; p: 1, 2 (When p = 1 -> manhattan_distance (l1), and p
= 2 -> euclidean_distance (l2))
  model = KNeighborsClassifier(n_neighbors = n_neighbors, weights = weights, p = p)
  return model

def model_dtree(criterion, splitter):
  #criterion{ “gini” ,  “entropy” }; splitter{ “best” ,  “random” }
  model = DecisionTreeClassifier(criterion = criterion, splitter = splitter)
  return model
```

In [63]:

```
# SVC_model training
# kernel: 'linear',  ‘poly’ ,  ‘rbf’ ; poly-degree: 3, 5; max_iter: default(-1), 100, 200, 300
def train_SVC(x_train, x_val):
  SVC_kernel = ['linear', 'poly', 'rbf']
  poly_degree = [3, 5]
  n_iter = [-1, 100, 200, 300]
  best_SVCacc = 0

  for kernel in SVC_kernel:
    for n in n_iter:
      if kernel == 'poly':
        for d in poly_degree:
          model = model_SVC(kernel, d, n)
      else:
        model = model_SVC(kernel, -1, n)
      #print(model)
      model.fit(x_train, y_train)
      SVC_pred = model.predict(x_val)
      accuracy = accuracy_score(y_test, SVC_pred)
      conf_matrix = confusion_matrix(y_test, SVC_pred)
      #print(accuracy)
      #print(conf_matrix)
      if (accuracy > best_SVCacc):
        best_SVCacc = accuracy
        best_SVCmodel = model
        best_SVCcm = conf_matrix
  return best_SVCmodel, best_SVCacc, best_SVCcm
```

In [64]:

```
# KNN_Model training
#n_neighbors: 5, 10, 15; weights{ ‘uniform’ , ‘distance’ }; p: 1, 2 (When p = 1 -> manhattan_distance (l1), and p =
2 -> euclidean_distance (l2))
def train_KNN(x_train, x_val):
  n_neighbors = [5, 10, 15]
  weights = ['uniform', 'distance']
  p_val = [1, 2]
  best_KNNacc = 0

  for n in n_neighbors:
    for weight in weights:
      for p in p_val:
        model = model_KNN(n, weight, p)
        #print(model)
        model.fit(x_train, y_train)
        KNN_pred = model.predict(x_val)
        accuracy = accuracy_score(y_test, KNN_pred)
        conf_matrix = confusion_matrix(y_test, KNN_pred)
        #print(accuracy)
        #print(conf_matrix)
        if (accuracy > best_KNNacc):
          best_KNNacc = accuracy
          best_KNNmodel = model
          best_KNNcm = conf_matrix
  return best_KNNmodel, best_KNNacc, best_KNNcm
```

In [65]:

```
# dtree_Model training
# criterion{ “gini” , “entropy” }; splitter{ “best” , “random” }
def train_dtree(x_train, x_val):
  criteria = ['gini', 'entropy']
  splitters = ['best', 'random']

  best_DTREEacc = 0

  for criterion in criteria:
    for splitter in splitters:
        model = model_dtree(criterion, splitter)
        #print(model)
        model.fit(x_train, y_train)
        dtree_pred = model.predict(x_val)
        accuracy = accuracy_score(y_test, dtree_pred)
        conf_matrix = confusion_matrix(y_test, dtree_pred)
        #print(accuracy)
        #print(conf_matrix)
        if (accuracy > best_DTREEacc):
          best_DTREEacc = accuracy
          best_DTREEmodel = model
          best_DTREEcm = conf_matrix
  return best_DTREEmodel, best_DTREEacc, best_DTREEcm
```

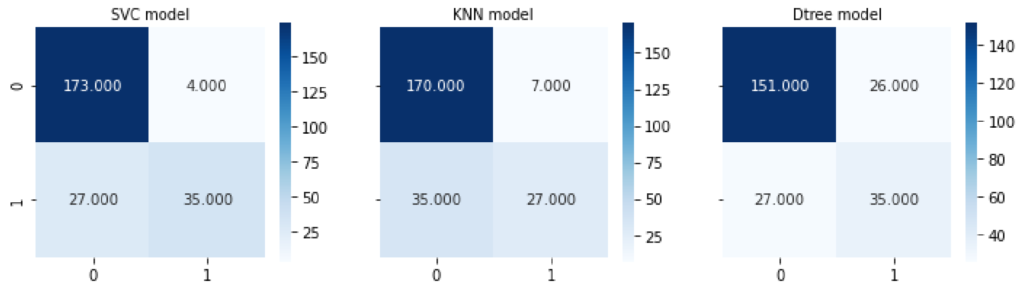
In [66]:

```python
def plot_results(SVC_model, SVC_acc, SVC_cm, KNN_model, KNN_acc, KNN_cm, Dtree_model, Dtree_acc, Dtree_cm):
  # plot best model and acc of each classification
  print('SVC best model:', SVC_model)
  print('SVC best acc:', SVC_acc)
  print('KNN best model:', KNN_model)
  print('KNN best acc:', KNN_acc)
  print('Dtree best model:', Dtree_model)
  print('Dtree best acc:', Dtree_acc)

  # plot confusion matrix
  cf_matrix = {}
  cf_matrix['SVC model'] = SVC_cm
  cf_matrix['KNN model'] = KNN_cm
  cf_matrix['Dtree model'] = Dtree_cm

  fig, axn = plt.subplots(1, 3, sharex=True, sharey=True,figsize=(12, 3))

  for i, ax in enumerate(axn.flat):
    k = list(cf_matrix)[i]
    sns.heatmap(cf_matrix[k], annot = True, fmt = ".3f", square = True,  ax=ax, cmap = plt.cm.Blues)
    ax.set_title(k, fontsize=10)
```

In [67]:

```python
#Train models using combination1 : all features
best_SVCmodel1, best_SVCacc1, best_SVCcm1 = train_SVC(combine1_X, val1_X)
best_KNNmodel1, best_KNNacc1, best_KNNcm1 = train_KNN(combine1_X, val1_X)
best_DTREEmodel1, best_DTREEacc1, best_DTREEcm1 = train_dtree(combine1_X, val1_X)

plot_results(best_SVCmodel1, best_SVCacc1, best_SVCcm1, best_KNNmodel1, best_KNNacc1, best_KNNcm1, best_DTREEmodel1,
best_DTREEacc1, best_DTREEcm1)
```

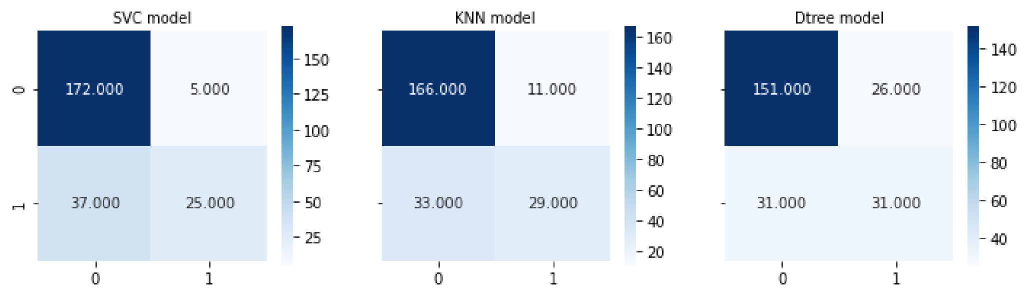
```
SVC best model: SVC()
SVC best acc: 0.8702928870292888
KNN best model: KNeighborsClassifier(n_neighbors=15)
KNN best acc: 0.8242677824267782
Dtree best model: DecisionTreeClassifier(criterion='entropy')
Dtree best acc: 0.7782426778242678
```



In [68]:

```
# Train models using combination2 : feature-selection
best_SVCmodel2, best_SVCacc2, best_SVCcm2 = train_SVC(combine2_X, val2_X)
best_KNNmodel2, best_KNNacc2, best_KNNcm2 = train_KNN(combine2_X, val2_X)
best_DTREEmodel2, best_DTREEacc2, best_DTREEcm2 = train_dtree(combine2_X, val2_X)

plot_results(best_SVCmodel2, best_SVCacc2, best_SVCcm2, best_KNNmodel2, best_KNNacc2, best_KNNcm2, best_DTREEmodel2,
best_DTREEacc2, best_DTREEcm2)
```

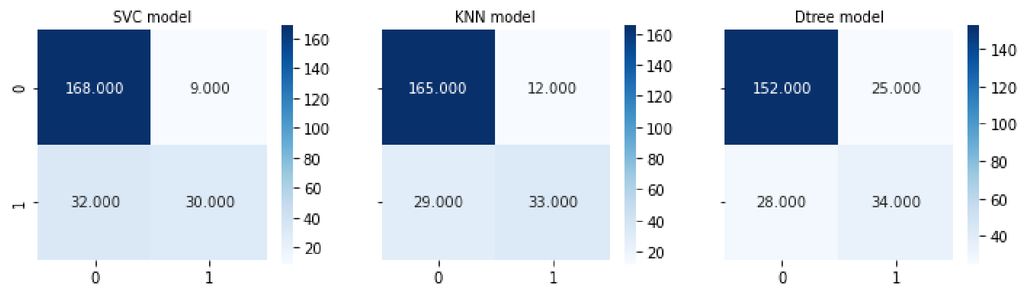
```
SVC best model: SVC()
SVC best acc: 0.8242677824267782
KNN best model: KNeighborsClassifier(n_neighbors=15, p=1)
KNN best acc: 0.8158995815899581
Dtree best model: DecisionTreeClassifier()
Dtree best acc: 0.7615062761506276
```



In [69]:

```
# Train models using combination3 : conclusion from project 1
best_SVCmodel3, best_SVCacc3, best_SVCcm3 = train_SVC(combine3_X, val3_X)
best_KNNmodel3, best_KNNacc3, best_KNNcm3 = train_KNN(combine3_X, val3_X)
best_DTREEmodel3, best_DTREEacc3, best_DTREEcm3 = train_dtree(combine3_X, val3_X)

plot_results(best_SVCmodel3, best_SVCacc3, best_SVCcm3, best_KNNmodel3, best_KNNacc3, best_KNNcm3, best_DTREEmodel3,
best_DTREEacc3, best_DTREEcm3)
```

```
SVC best model: SVC()
SVC best acc: 0.8284518828451883
KNN best model: KNeighborsClassifier(n_neighbors=10, p=1, weights='distance')
KNN best acc: 0.8284518828451883
Dtree best model: DecisionTreeClassifier(criterion='entropy')
Dtree best acc: 0.7782426778242678
```



1. (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 [70]:

```
# Import Clustering models
from sklearn.cluster import KMeans, DBSCAN
from scipy.cluster.hierarchy import linkage, dendrogram, fcluster
from sklearn import metrics
```

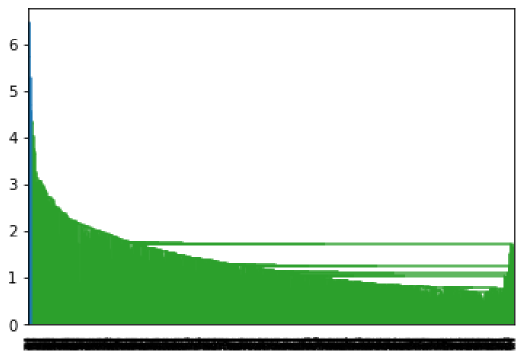
In [71]:

```
y_train=train_Y.iloc[:,2]
y_test=val_Y.iloc[:,2]
```

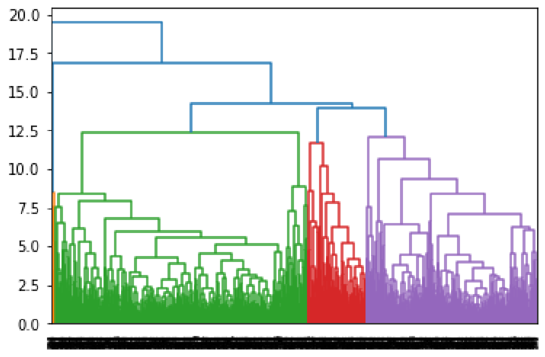
In [72]:

```
##Hierarchical finding the parameter
method=["single","complete","average","ward"]
comb=[combine1_X, combine2_X, combine3_X]
for j,com in enumerate(comb):
  print("comb:",j+1)
  best1=0
  best2=0
  for k,i in enumerate(method):
    clustering = linkage(com, method = i, metric = "euclidean")
    print(i)
    # Plot dendrogram
    plt.figure()
    dendrogram(clustering)
    plt.show()

    # Form clusters
    clusters = fcluster(clustering, 2, criterion = 'maxclust')
    print(np.unique(clusters,return_counts=True))

    if(metrics.adjusted_rand_score(y_train, clusters-1))>best1:
      best1=metrics.adjusted_rand_score(y_train, clusters-1)
      para1=i
    if(metrics.silhouette_score(com, clusters-1, metric = "euclidean"))>best2:
      best2=(metrics.silhouette_score(com, clusters-1, metric = "euclidean"))
      para2=i
  print("comb:",j+1," best method1:",para1,"bestmethod2:",para2)
```

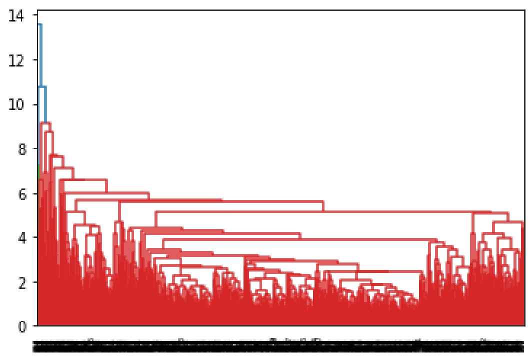
```
comb: 1
single
```



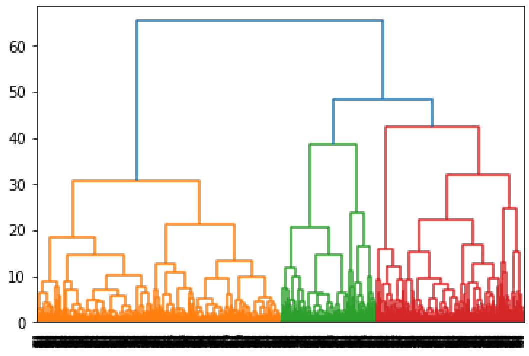
```
(array([1, 2], dtype=int32), array([955,   1], dtype=int64))
complete
```



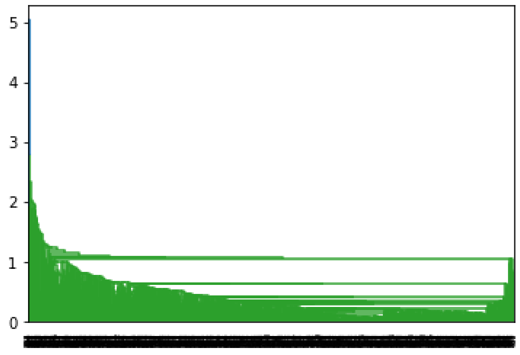
```
(array([1, 2], dtype=int32), array([955,   1], dtype=int64))
average
```



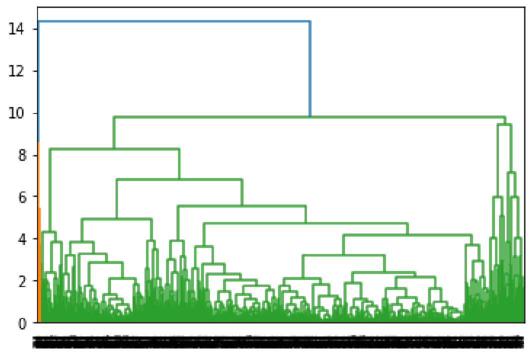
```
(array([1, 2], dtype=int32), array([  2, 954], dtype=int64))
ward
```



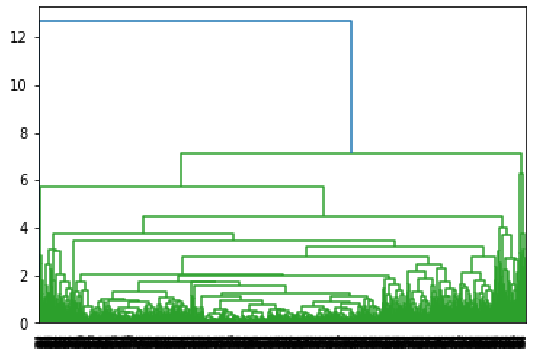
```
(array([1, 2], dtype=int32), array([481, 475], dtype=int64))
comb: 1  best method1: ward bestmethod2: average
comb: 2
single
```



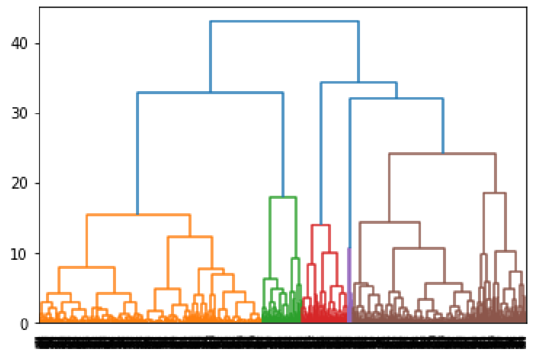
```
(array([1, 2], dtype=int32), array([  2, 954], dtype=int64))
complete
```



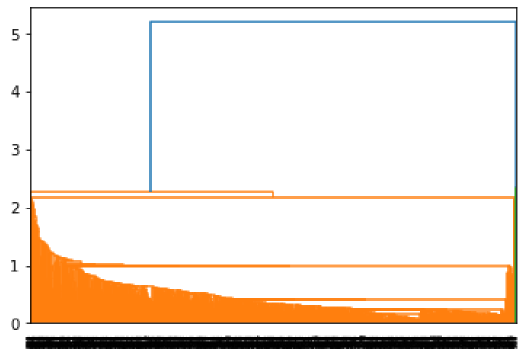
```
(array([1, 2], dtype=int32), array([ 10, 946], dtype=int64))
average
```



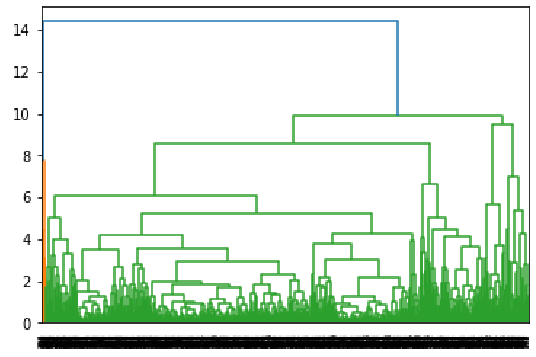
```
(array([1, 2], dtype=int32), array([  2, 954], dtype=int64))
ward
```



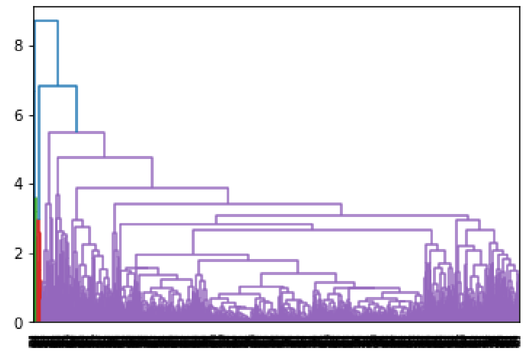
```
(array([1, 2], dtype=int32), array([517, 439], dtype=int64))
comb: 2  best method1: ward bestmethod2: single
comb: 3
single
```



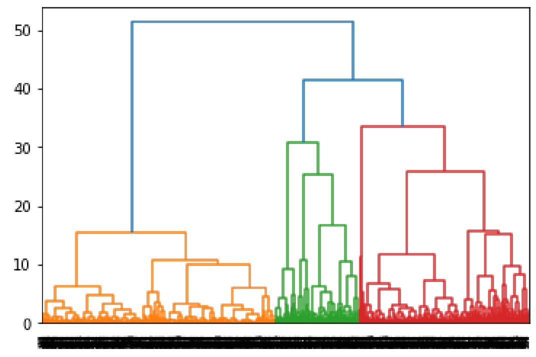
```
(array([1, 2], dtype=int32), array([954,   2], dtype=int64))
complete
```



```
(array([1, 2], dtype=int32), array([  9, 947], dtype=int64))
average
```



```
(array([1, 2], dtype=int32), array([ 10, 946], dtype=int64))
ward
```

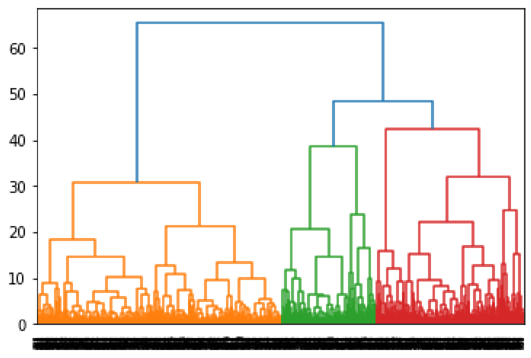


```
(array([1, 2], dtype=int32), array([459, 497], dtype=int64))
comb: 3  best method1: ward bestmethod2: single
```

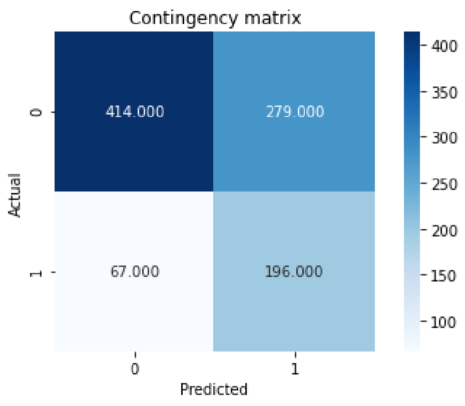
In [73]:

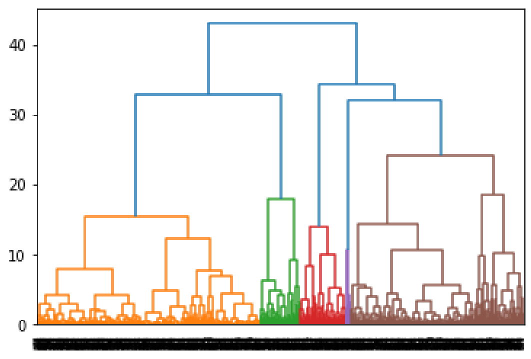
```
##Hierarchical ward
k=1
for i in comb:
  print("ward method with comb",k,"features :")
  k+=1
  clustering = linkage(i, method = "ward", metric = "euclidean")

  # Plot dendrogram
  plt.figure()
  dendrogram(clustering)
  plt.show()

  # Form clusters
  clusters = fcluster(clustering, 2, criterion = 'maxclust')
  print(np.unique(clusters,return_counts=True))

  # Plot contingency matrix
  cont_matrix = metrics.cluster.contingency_matrix(y_train, clusters-1)
  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()

  # Compute adjusted Rand index and silhouette coefficient
  print("adjusted rand score:",metrics.adjusted_rand_score(y_train, clusters-1))
  print("silhouette score:",metrics.silhouette_score(i, clusters-1, metric = "euclidean"))
```

```
ward method with comb 1 features :
```

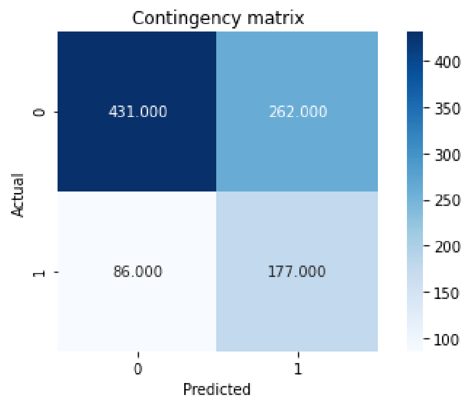


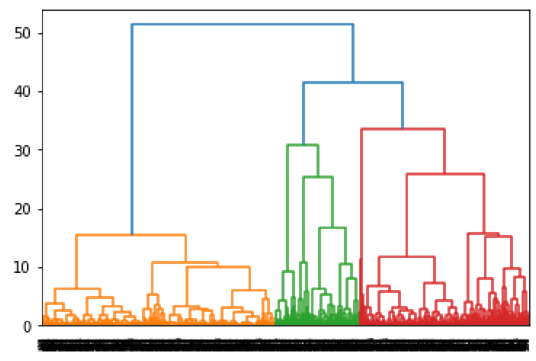
```
(array([1, 2], dtype=int32), array([481, 475], dtype=int64))
adjusted rand score: 0.07547960260815256
silhouette score: 0.17928795608281797
ward method with comb 2 features :
```



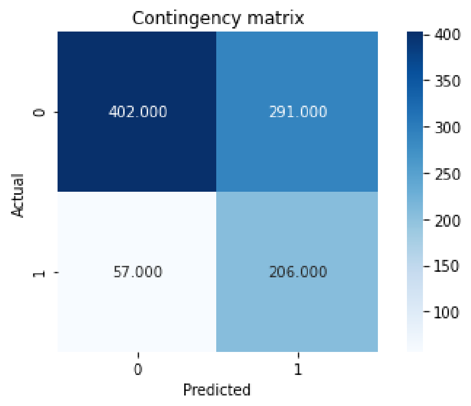


```
(array([1, 2], dtype=int32), array([517, 439], dtype=int64))
adjusted rand score: 0.07194589260579701
silhouette score: 0.24443291211568716
ward method with comb 3 features :
```





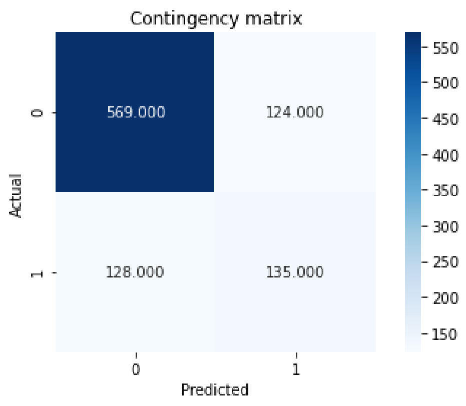
```
(array([1, 2], dtype=int32), array([459, 497], dtype=int64))
adjusted rand score: 0.07289628105385934
silhouette score: 0.26269486484999194
```



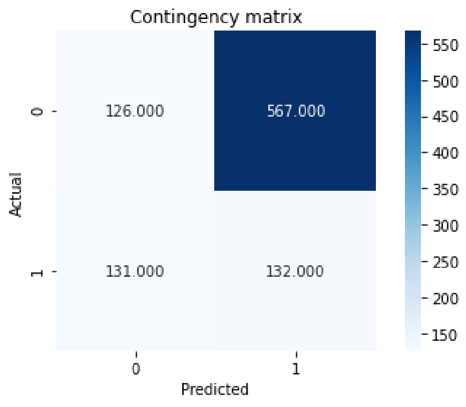
In [76]:

```
##K-means
for j,i in enumerate(comb):
  print("combination",j+1,":")
  clustering = KMeans(n_clusters = 2, init = 'k-means++', n_init = 10,max_iter=300).fit(i)
  clusters=clustering.labels_
  print(np.unique(clusters,return_counts=True))

  # Plot contingency matrix
  cont_matrix = metrics.cluster.contingency_matrix(y_train, 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()
  plt.show()

  # Compute adjusted Rand index and silhouette coefficient
  print("adjusted rand score:",metrics.adjusted_rand_score(y_train, clusters))
  print("silhouette score",metrics.silhouette_score(i, clusters, metric = "euclidean"))

print("best combination1 is having the highest adjusted rand score 0.18919892289605386")
```

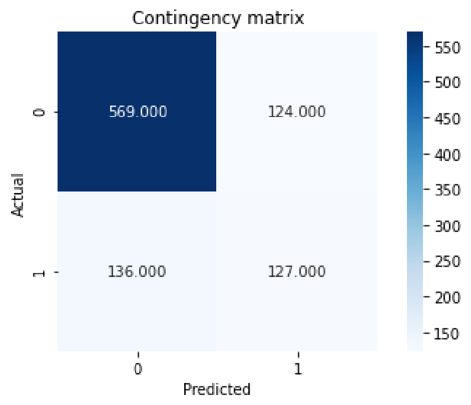
```
combination 1 :
(array([0, 1]), array([697, 259], dtype=int64))
```



```
adjusted rand score: 0.1885473768209509
silhouette score 0.31019293845309287
combination 2 :
(array([0, 1]), array([257, 699], dtype=int64))
```



```
adjusted rand score: 0.17563900727392587
silhouette score 0.36860260126685873
combination 3 :
(array([0, 1]), array([705, 251], dtype=int64))
```



```
adjusted rand score: 0.16957026114331927
silhouette score 0.4221743498492149
best combination1 is having the highest adjusted rand score 0.18919892289605386
```

In [77]:

```
##DBSCAN - finding best parameter for feature combination 1(eps 0.9, min sample=4)
for i in np.arange(0.7,1.3,0.1):
  for j in range(3,10):
    clustering=DBSCAN(eps=i,min_samples=j, metric='euclidean')
    clustering.fit(combine1_X)
    clusters = clustering.labels_
    print("eps=",i,"min_sam=",j,np.unique(clusters,return_counts=True))
```

```
eps= 0.7 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
       16, 17], dtype=int64), array([843,   3,   5,   9,   3,  10,  30,   6,   4,   5,   9,   6,   4,
         3,   3,   3,   4,   3,   3], dtype=int64))
eps= 0.7 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11], dtype=int64), array([881,   8,  22,   4,   4,   5,   4,   3,
7,   5,   5,   4,   4],
      dtype=int64))
eps= 0.7 min_sam= 5 (array([-1,  0,  1,  2], dtype=int64), array([923,  22,   5,   6], dtype=int64))
eps= 0.7 min_sam= 6 (array([-1,  0], dtype=int64), array([934,  22], dtype=int64))
eps= 0.7 min_sam= 7 (array([-1,  0], dtype=int64), array([938,  18], dtype=int64))
eps= 0.7 min_sam= 8 (array([-1,  0], dtype=int64), array([945,  11], dtype=int64))
eps= 0.7 min_sam= 9 (array([-1], dtype=int64), array([956], dtype=int64))
eps= 0.7999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
      dtype=int64), array([756, 117,   3,  37,   8,  10,   4,   3,   3,   3,   3,   3,   3,
         3], dtype=int64))
eps= 0.7999999999999999 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([795, 108,  30,   7,  10,   2,   4], dtype=int64))
eps= 0.7999999999999999 min_sam= 5 (array([-1,  0,  1,  2,  3,  4,  5,  6], dtype=int64), array([816,  79,   7,   9,   8,  22,  10,   5], dtype
=int64))
eps= 0.7999999999999999 min_sam= 6 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([848,   9,  69,   8,  11,   6,   5], dtype=int64))
eps= 0.7999999999999999 min_sam= 7 (array([-1,  0,  1,  2,  3], dtype=int64), array([876,  59,   7,   7,   7], dtype=int64))
eps= 0.7999999999999999 min_sam= 8 (array([-1,  0], dtype=int64), array([904,  52], dtype=int64))
eps= 0.7999999999999999 min_sam= 9 (array([-1,  0], dtype=int64), array([905,  51], dtype=int64))
eps= 0.8999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
      dtype=int64), array([638, 282,   3,   3,   3,   3,   3,   3,   3,   3,   3,   3,   3,
         3], dtype=int64))
eps= 0.8999999999999999 min_sam= 4 (array([-1,  0,  1], dtype=int64), array([689, 223,  44], dtype=int64))
eps= 0.8999999999999999 min_sam= 5 (array([-1,  0,  1,  2], dtype=int64), array([712, 200,  36,   8], dtype=int64))
eps= 0.8999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([736, 184,  36], dtype=int64))
eps= 0.8999999999999999 min_sam= 7 (array([-1,  0,  1], dtype=int64), array([753, 168,  35], dtype=int64))
eps= 0.8999999999999999 min_sam= 8 (array([-1,  0,  1,  2,  3], dtype=int64), array([783, 135,  27,   3,   8], dtype=int64))
eps= 0.8999999999999999 min_sam= 9 (array([-1,  0,  1,  2,  3,  4], dtype=int64), array([800, 109,   7,  26,   6,   8], dtype=int64))
eps= 0.9999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6], dtype=int64), array([566, 370,   3,   3,   3,   5,   3,   3], dtype
=int64))
eps= 0.9999999999999999 min_sam= 4 (array([-1,  0,  1], dtype=int64), array([595, 356,   5], dtype=int64))
eps= 0.9999999999999999 min_sam= 5 (array([-1,  0,  1], dtype=int64), array([621, 330,   5], dtype=int64))
eps= 0.9999999999999999 min_sam= 6 (array([-1,  0], dtype=int64), array([642, 314], dtype=int64))
eps= 0.9999999999999999 min_sam= 7 (array([-1,  0], dtype=int64), array([653, 303], dtype=int64))
eps= 0.9999999999999999 min_sam= 8 (array([-1,  0], dtype=int64), array([668, 288], dtype=int64))
eps= 0.9999999999999999 min_sam= 9 (array([-1,  0], dtype=int64), array([678, 278], dtype=int64))
eps= 1.0999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7], dtype=int64), array([497, 436,   3,   3,   3,   4,   4,   3,
3], dtype=int64))
eps= 1.0999999999999999 min_sam= 4 (array([-1,  0], dtype=int64), array([530, 426], dtype=int64))
eps= 1.0999999999999999 min_sam= 5 (array([-1,  0,  1], dtype=int64), array([543, 410,   3], dtype=int64))
eps= 1.0999999999999999 min_sam= 6 (array([-1,  0], dtype=int64), array([562, 394], dtype=int64))
eps= 1.0999999999999999 min_sam= 7 (array([-1,  0,  1], dtype=int64), array([576, 374,   6], dtype=int64))
eps= 1.0999999999999999 min_sam= 8 (array([-1,  0], dtype=int64), array([594, 362], dtype=int64))
eps= 1.0999999999999999 min_sam= 9 (array([-1,  0], dtype=int64), array([600, 356], dtype=int64))
eps= 1.1999999999999997 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([440, 497,   3,   3,   6,   4,   3], dtype=int64))
eps= 1.1999999999999997 min_sam= 4 (array([-1,  0,  1,  2,  3], dtype=int64), array([461, 483,   5,   4,   3], dtype=int64))
eps= 1.1999999999999997 min_sam= 5 (array([-1,  0,  1], dtype=int64), array([477, 474,   5], dtype=int64))
eps= 1.1999999999999997 min_sam= 6 (array([-1,  0], dtype=int64), array([492, 464], dtype=int64))
eps= 1.1999999999999997 min_sam= 7 (array([-1,  0], dtype=int64), array([504, 452], dtype=int64))
eps= 1.1999999999999997 min_sam= 8 (array([-1,  0], dtype=int64), array([514, 442], dtype=int64))
eps= 1.1999999999999997 min_sam= 9 (array([-1,  0], dtype=int64), array([530, 426], dtype=int64))
eps= 1.2999999999999998 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7], dtype=int64), array([377, 555,   3,   3,   4,   3,   4,   4,
3], dtype=int64))
eps= 1.2999999999999998 min_sam= 4 (array([-1,  0,  1,  2], dtype=int64), array([408, 540,   4,   4], dtype=int64))
eps= 1.2999999999999998 min_sam= 5 (array([-1,  0,  1,  2], dtype=int64), array([425, 522,   5,   4], dtype=int64))
eps= 1.2999999999999998 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([435, 516,   5], dtype=int64))
eps= 1.2999999999999998 min_sam= 7 (array([-1,  0], dtype=int64), array([452, 504], dtype=int64))
eps= 1.2999999999999998 min_sam= 8 (array([-1,  0], dtype=int64), array([462, 494], dtype=int64))
eps= 1.2999999999999998 min_sam= 9 (array([-1,  0], dtype=int64), array([470, 486], dtype=int64))
```

In [78]:

```
##DBSCAN - finding best parameter for feature combination 2(eps=0.8, min sample=7)
for i in np.arange(0.5,1.2,0.1):
  for j in range(3,10):
    clustering=DBSCAN(eps=i,min_samples=j, metric='euclidean')
    clustering.fit(combine2_X)
    clusters = clustering.labels_
    print("eps=",i,"min_sam=",j,np.unique(clusters,return_counts=True))
```

```
eps= 0.5 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
       16], dtype=int64), array([251, 634,   5,   5,   3,   8,   4,   5,   5,   3,   3,   3,   6,
         3,   5,   6,   4,   3], dtype=int64))
eps= 0.5 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5,  6], dtype=int64), array([302, 627,   5,   5,   5,   4,   4,   4], dtype=int64))
eps= 0.5 min_sam= 5 (array([-1,  0,  1,  2], dtype=int64), array([335, 612,   5,   4], dtype=int64))
eps= 0.5 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([351, 598,   7], dtype=int64))
eps= 0.5 min_sam= 7 (array([-1,  0,  1], dtype=int64), array([359, 591,   6], dtype=int64))
eps= 0.5 min_sam= 8 (array([-1,  0], dtype=int64), array([381, 575], dtype=int64))
eps= 0.5 min_sam= 9 (array([-1,  0], dtype=int64), array([382, 574], dtype=int64))
eps= 0.6 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11], dtype=int64), array([180, 713,   5,  11,   3,  13,   4,   3,
10,   3,   3,   5,   3],
      dtype=int64))
eps= 0.6 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5,  6], dtype=int64), array([211, 702,   5,   4,  10,   9,  11,   4], dtype=int64))
eps= 0.6 min_sam= 5 (array([-1,  0,  1,  2,  3,  4], dtype=int64), array([238, 685,   9,  10,   8,   6], dtype=int64))
eps= 0.6 min_sam= 6 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([258, 664,   5,   9,   6,   8,   6], dtype=int64))
eps= 0.6 min_sam= 7 (array([-1,  0,  1,  2], dtype=int64), array([286, 659,   5,   6], dtype=int64))
eps= 0.6 min_sam= 8 (array([-1,  0], dtype=int64), array([306, 650], dtype=int64))
eps= 0.6 min_sam= 9 (array([-1,  0], dtype=int64), array([312, 644], dtype=int64))
eps= 0.7 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
      dtype=int64), array([118, 795,   4,   3,   3,   4,   4,   3,   4,   5,   3,   3,   4,
         3], dtype=int64))
eps= 0.7 min_sam= 4 (array([-1,  0,  1,  2,  3,  4], dtype=int64), array([151, 788,   4,   5,   4,   4], dtype=int64))
eps= 0.7 min_sam= 5 (array([-1,  0,  1], dtype=int64), array([177, 756,  23], dtype=int64))
eps= 0.7 min_sam= 6 (array([-1,  0,  1,  2,  3], dtype=int64), array([189, 731,   6,  16,  14], dtype=int64))
eps= 0.7 min_sam= 7 (array([-1,  0,  1,  2], dtype=int64), array([208, 723,  11,  14], dtype=int64))
eps= 0.7 min_sam= 8 (array([-1,  0,  1,  2,  3], dtype=int64), array([230, 700,  11,   7,   8], dtype=int64))
eps= 0.7 min_sam= 9 (array([-1,  0,  1,  2], dtype=int64), array([252, 689,   9,   6], dtype=int64))
eps= 0.7999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11], dtype=int64), array([ 84, 828,   3,   3,   3,
4,   7,   5,   4,   3,   5,   4,   3],
      dtype=int64))
eps= 0.7999999999999999 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([106, 825,   4,   7,   5,   5,   4], dtype=int64))
eps= 0.7999999999999999 min_sam= 5 (array([-1,  0], dtype=int64), array([142, 814], dtype=int64))
eps= 0.7999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([150, 800,   6], dtype=int64))
eps= 0.7999999999999999 min_sam= 7 (array([-1,  0,  1], dtype=int64), array([158, 772,  26], dtype=int64))
eps= 0.7999999999999999 min_sam= 8 (array([-1,  0,  1], dtype=int64), array([177, 763,  16], dtype=int64))
eps= 0.7999999999999999 min_sam= 9 (array([-1,  0,  1], dtype=int64), array([181, 759,  16], dtype=int64))
eps= 0.8999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7], dtype=int64), array([ 58, 858,   6,   4,  10,   8,   6,   3,
3], dtype=int64))
eps= 0.8999999999999999 min_sam= 4 (array([-1,  0,  1,  2,  3,  4], dtype=int64), array([ 81, 853,   4,   7,   5,   6], dtype=int64))
eps= 0.8999999999999999 min_sam= 5 (array([-1,  0,  1,  2,  3,  4], dtype=int64), array([ 90, 842,   5,   5,   8,   6], dtype=int64))
eps= 0.8999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([110, 838,   8], dtype=int64))
eps= 0.8999999999999999 min_sam= 7 (array([-1,  0], dtype=int64), array([121, 835], dtype=int64))
eps= 0.8999999999999999 min_sam= 8 (array([-1,  0], dtype=int64), array([131, 825], dtype=int64))
eps= 0.8999999999999999 min_sam= 9 (array([-1,  0], dtype=int64), array([138, 818], dtype=int64))
eps= 0.9999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([ 49, 869,  17,   4,   8,   6,   3], dtype=int64))
eps= 0.9999999999999999 min_sam= 4 (array([-1,  0,  1,  2,  3,  4], dtype=int64), array([ 54, 869,  16,   4,   7,   6], dtype=int64))
eps= 0.9999999999999999 min_sam= 5 (array([-1,  0,  1,  2,  3], dtype=int64), array([ 77, 860,   5,   5,   9], dtype=int64))
eps= 0.9999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([ 88, 859,   9], dtype=int64))
eps= 0.9999999999999999 min_sam= 7 (array([-1,  0], dtype=int64), array([100, 856], dtype=int64))
eps= 0.9999999999999999 min_sam= 8 (array([-1,  0,  1], dtype=int64), array([104, 846,   6], dtype=int64))
eps= 0.9999999999999999 min_sam= 9 (array([-1,  0,  1], dtype=int64), array([109, 841,   6], dtype=int64))
eps= 1.0999999999999999 min_sam= 3 (array([-1,  0,  1,  2], dtype=int64), array([ 38, 910,   5,   3], dtype=int64))
eps= 1.0999999999999999 min_sam= 4 (array([-1,  0,  1], dtype=int64), array([ 42, 909,   5], dtype=int64))
eps= 1.0999999999999999 min_sam= 5 (array([-1,  0,  1], dtype=int64), array([ 46, 905,   5], dtype=int64))
eps= 1.0999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([ 60, 891,   5], dtype=int64))
eps= 1.0999999999999999 min_sam= 7 (array([-1,  0], dtype=int64), array([ 73, 883], dtype=int64))
eps= 1.0999999999999999 min_sam= 8 (array([-1,  0,  1], dtype=int64), array([ 75, 874,   7], dtype=int64))
eps= 1.0999999999999999 min_sam= 9 (array([-1,  0,  1], dtype=int64), array([ 78, 871,   7], dtype=int64))
```

In [79]:

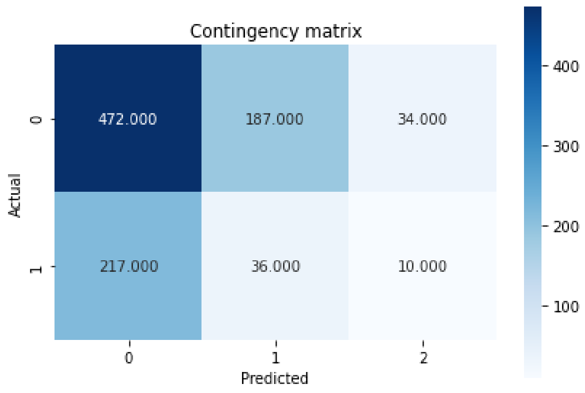
```
##DBSCAN - finding best parameter for feature combination 3(eps=0.4, min sample=6)
for i in np.arange(0.4,1.1,0.1):
  for j in range(3,10):
    clustering=DBSCAN(eps=i,min_samples=j, metric='euclidean')
    clustering.fit(combine3_X)
    clusters = clustering.labels_
    print("eps=",i,"min_sam=",j,np.unique(clusters,return_counts=True))
```

```
eps= 0.4 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15],
      dtype=int64), array([361, 542,   3,   5,   5,   4,   3,   3,   4,   5,   3,   3,   3,
         3,   3,   3,   3], dtype=int64))
eps= 0.4 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5,  6], dtype=int64), array([409, 518,   5,   7,   5,   4,   4,   4], dtype=int64))
eps= 0.4 min_sam= 5 (array([-1,  0,  1,  2,  3], dtype=int64), array([431, 511,   5,   4,   5], dtype=int64))
eps= 0.4 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([459, 479,  18], dtype=int64))
eps= 0.4 min_sam= 7 (array([-1,  0,  1,  2], dtype=int64), array([476, 466,   8,   6], dtype=int64))
eps= 0.4 min_sam= 8 (array([-1,  0,  1], dtype=int64), array([488, 460,   8], dtype=int64))
eps= 0.4 min_sam= 9 (array([-1,  0,  1], dtype=int64), array([505, 440,  11], dtype=int64))
eps= 0.5 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
       16, 17, 18, 19], dtype=int64), array([276, 594,  10,   7,   4,   9,   3,   4,   3,   6,   5,   6,   3,
         3,   4,   3,   3,   4,   3,   3,   3], dtype=int64))
eps= 0.5 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7], dtype=int64), array([326, 591,   4,   4,   6,   5,   4,  10,   6], dtype=int6
4))
eps= 0.5 min_sam= 5 (array([-1,  0,  1,  2,  3], dtype=int64), array([349, 588,   5,   5,   9], dtype=int64))
eps= 0.5 min_sam= 6 (array([-1,  0,  1,  2], dtype=int64), array([367, 569,  14,   6], dtype=int64))
eps= 0.5 min_sam= 7 (array([-1,  0,  1], dtype=int64), array([392, 554,  10], dtype=int64))
eps= 0.5 min_sam= 8 (array([-1,  0], dtype=int64), array([407, 549], dtype=int64))
eps= 0.5 min_sam= 9 (array([-1,  0], dtype=int64), array([417, 539], dtype=int64))
eps= 0.6 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15,
       16], dtype=int64), array([194, 686,  18,   5,   3,   3,   4,   3,  11,   3,   3,   3,   3,
         4,   3,   3,   4,   3], dtype=int64))
eps= 0.6 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([241, 673,  18,   6,  10,   4,   4], dtype=int64))
eps= 0.6 min_sam= 5 (array([-1,  0,  1,  2,  3], dtype=int64), array([271, 653,  15,   9,   8], dtype=int64))
eps= 0.6 min_sam= 6 (array([-1,  0,  1,  2,  3], dtype=int64), array([288, 642,  12,   8,   6], dtype=int64))
eps= 0.6 min_sam= 7 (array([-1,  0,  1,  2,  3], dtype=int64), array([295, 634,  12,   8,   7], dtype=int64))
eps= 0.6 min_sam= 8 (array([-1,  0,  1], dtype=int64), array([322, 624,  10], dtype=int64))
eps= 0.6 min_sam= 9 (array([-1,  0,  1], dtype=int64), array([331, 616,   9], dtype=int64))
eps= 0.7 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
      dtype=int64), array([142, 773,   4,   5,   3,   3,   3,   3,   3,   4,   3,   3,   4,
         3], dtype=int64))
eps= 0.7 min_sam= 4 (array([-1,  0,  1,  2], dtype=int64), array([192, 756,   4,   4], dtype=int64))
eps= 0.7 min_sam= 5 (array([-1,  0], dtype=int64), array([211, 745], dtype=int64))
eps= 0.7 min_sam= 6 (array([-1,  0,  1,  2], dtype=int64), array([218, 708,  26,   4], dtype=int64))
eps= 0.7 min_sam= 7 (array([-1,  0,  1,  2], dtype=int64), array([236, 683,  21,  16], dtype=int64))
eps= 0.7 min_sam= 8 (array([-1,  0,  1,  2], dtype=int64), array([247, 674,  21,  14], dtype=int64))
eps= 0.7 min_sam= 9 (array([-1,  0,  1,  2], dtype=int64), array([259, 669,  14,  14], dtype=int64))
eps= 0.7999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13],
      dtype=int64), array([ 97, 810,   4,   6,   3,   3,   6,   4,   4,   3,   3,   3,   3,
         4,   3], dtype=int64))
eps= 0.7999999999999999 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([137, 798,   5,   4,   4,   4,   4], dtype=int64))
eps= 0.7999999999999999 min_sam= 5 (array([-1,  0,  1], dtype=int64), array([158, 793,   5], dtype=int64))
eps= 0.7999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([166, 786,   4], dtype=int64))
eps= 0.7999999999999999 min_sam= 7 (array([-1,  0], dtype=int64), array([178, 778], dtype=int64))
eps= 0.7999999999999999 min_sam= 8 (array([-1,  0], dtype=int64), array([185, 771], dtype=int64))
eps= 0.7999999999999999 min_sam= 9 (array([-1,  0,  1], dtype=int64), array([194, 746,  16], dtype=int64))
eps= 0.8999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12],
      dtype=int64), array([ 70, 837,   4,   4,   4,   3,   6,   4,   7,   3,   3,   3,   5,
         3], dtype=int64))
eps= 0.8999999999999999 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([104, 829,   4,   5,   6,   4,   4], dtype=int64))
eps= 0.8999999999999999 min_sam= 5 (array([-1,  0,  1,  2], dtype=int64), array([131, 814,   4,   7], dtype=int64))
eps= 0.8999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([140, 810,   6], dtype=int64))
eps= 0.8999999999999999 min_sam= 7 (array([-1,  0], dtype=int64), array([150, 806], dtype=int64))
eps= 0.8999999999999999 min_sam= 8 (array([-1,  0], dtype=int64), array([152, 804], dtype=int64))
eps= 0.8999999999999999 min_sam= 9 (array([-1,  0], dtype=int64), array([157, 799], dtype=int64))
eps= 0.9999999999999999 min_sam= 3 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([ 60, 873,   5,   8,   3,   3,   4], dtype=int64))
eps= 0.9999999999999999 min_sam= 4 (array([-1,  0,  1,  2,  3,  4,  5], dtype=int64), array([ 70, 857,   5,   6,   6,   8,   4], dtype=int64))
eps= 0.9999999999999999 min_sam= 5 (array([-1,  0,  1], dtype=int64), array([ 96, 856,   4], dtype=int64))
eps= 0.9999999999999999 min_sam= 6 (array([-1,  0,  1], dtype=int64), array([111, 837,   8], dtype=int64))
eps= 0.9999999999999999 min_sam= 7 (array([-1,  0,  1], dtype=int64), array([117, 830,   9], dtype=int64))
eps= 0.9999999999999999 min_sam= 8 (array([-1,  0,  1], dtype=int64), array([126, 822,   8], dtype=int64))
eps= 0.9999999999999999 min_sam= 9 (array([-1,  0], dtype=int64), array([137, 819], dtype=int64))
```

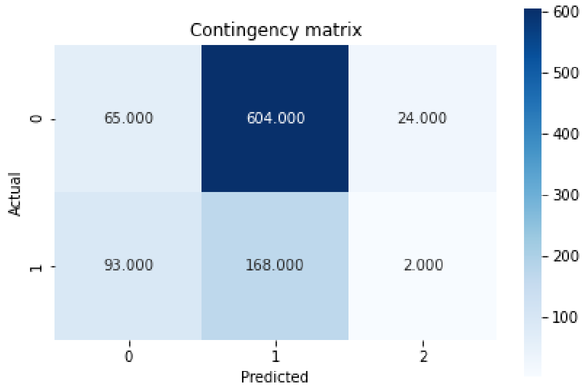
In [80]:

```
eps=[0.9,0.8,0.4]
minsam=[4,7,6]
for i in range(3):
  ##DBSCAN - finding best parameter for feature combination 3(eps=0.4, min sample=6)
  print("Combination:",i+1)
  clustering=DBSCAN(eps=eps[i],min_samples=minsam[i], metric='euclidean')
  clustering.fit(comb[i])
  clusters = clustering.labels_
  print("eps=",eps[i],"min_sample=",minsam[i],np.unique(clusters,return_counts=True))

  clusters = clustering.labels_

  # Plot contingency matrix
  cont_matrix = metrics.cluster.contingency_matrix(y_train, 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()
  plt.show()

  # Compute adjusted Rand index and silhouette coefficient
  print("adjusted random score:",metrics.adjusted_rand_score(y_train, clusters))
  print("silhouette score:",metrics.silhouette_score(comb[i], clusters, metric = "euclidean"),"₩n")
```

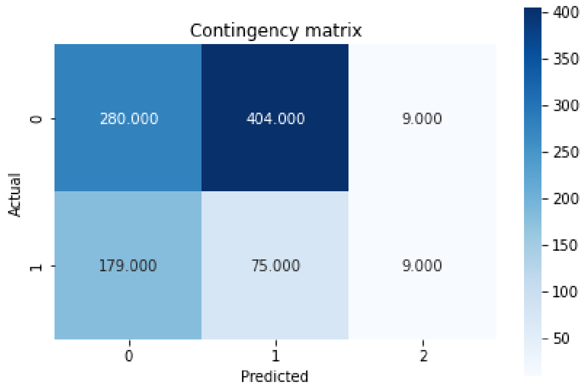
```
Combination: 1
eps= 0.9 min_sample= 4 (array([-1,  0,  1], dtype=int64), array([689, 223,  44], dtype=int64))
```



```
adjusted random score: -0.04160817892251404
silhouette score: -0.09764259930665097

Combination: 2
eps= 0.8 min_sample= 7 (array([-1,  0,  1], dtype=int64), array([158, 772,  26], dtype=int64))
```



```
adjusted random score: 0.15436027987033674
silhouette score: 0.3921971713068565

Combination: 3
eps= 0.4 min_sample= 6 (array([-1,  0,  1], dtype=int64), array([459, 479,  18], dtype=int64))
```



```
adjusted random score: 0.06292270074918523
silhouette score: -0.004071176362910133
```

1. (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 [81]:

```
# !pip install geopandas==0.3.0
# !pip install pyshp==1.2.10
# !pip install shapely==1.6.3
# !pip install plotly

# !pip install plotly-geo
# !pip install plotly
# !pip install geopandas
# !pip install pyshp
# !pip install shapely
```

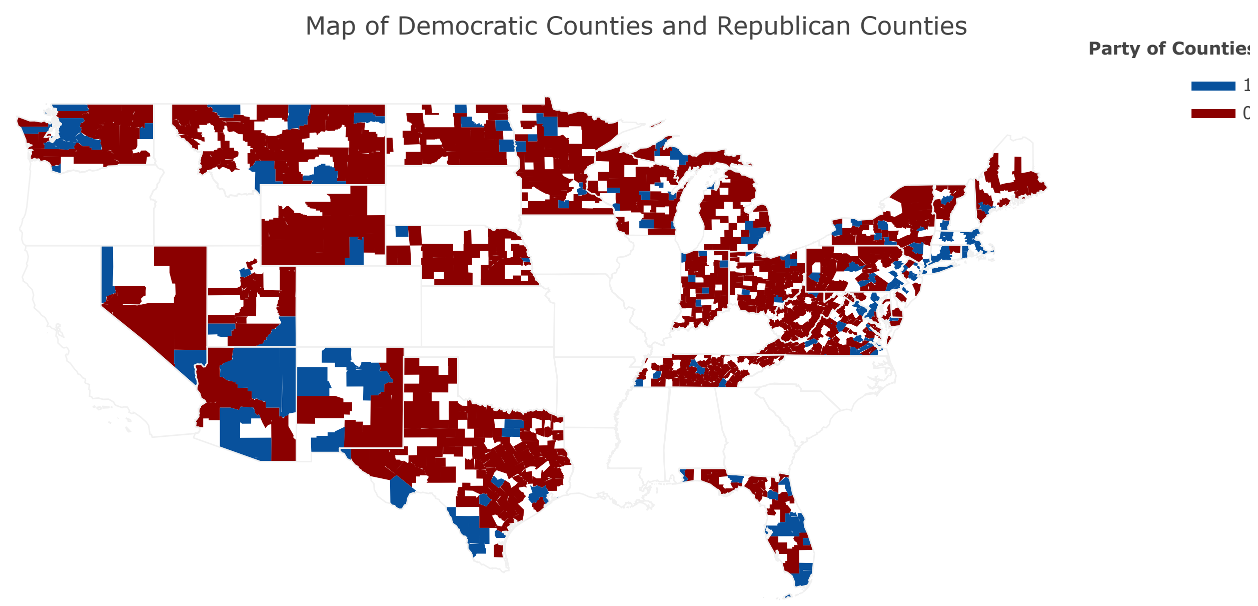
In [82]:

```
#Plot the map with the classified result using SVC model with combine1_X
import plotly
import plotly.figure_factory as ff

#test_set = pd.read_csv("demographics_test.csv")
```

In [83]:

```
data_x_scaled = scaler.transform(dataX)

model_party = SVC(kernel='rbf').fit(combine1_X, train_Y.iloc[:, 2])
party_pred = model_party.predict(data_x_scaled)

#Plot the map
fips = merged_train.loc[:, 'FIPS'].tolist()
values = party_pred.tolist()

print(len(fips), len(values))

colorscale = [
    "#8B0000",
    "#08519c",
]

fig = ff.create_choropleth(
    fips=fips, values=values,
    colorscale=colorscale,
    title= 'Map of Democratic Counties and Republican Counties',
    legend_title= 'Party of Counties'

)

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

```
1195 1195
```



1. (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 [84]:

```
# Read demographics_test dataset
# uploaded_test = files.upload()
#test_file = 'demographics_test.csv'
# test_file = io.StringIO(uploaded_test['demographics_test.csv'].decode('utf-8'))
test_data = pd.read_csv("demographics_test.csv")
test_data.head()
```

	State	County	FIPS	Total Population	Percent White, not Hispanic or Latino	Percent Black, not Hispanic or Latino	Percent Hispanic or Latino	Percent Foreign Born	Percent Female	Percent Age 29 and Under	Percent Age 65 and Older	Median Household Income	Pe Unemp
0	NV	eureka	32011	1730	98.265896	0.057803	0.462428	0.346821	51.156069	27.109827	15.606936	70000	3.755365
1	TX	zavala	48507	12107	5.798299	0.594697	93.326175	9.193029	49.723301	49.302057	12.480383	26639	11.95516
2	VA	king george	51099	25260	73.804434	16.722090	4.441805	2.505938	50.166271	40.186065	11.868567	84342	6.479939
3	OH	hamilton	39061	805965	66.354867	25.654340	2.890944	5.086945	51.870615	40.779686	14.161657	50399	7.864630
4	TX	austin	48015	29107	63.809393	8.479060	25.502456	9.946061	50.671660	37.351840	17.799842	56681	5.782337

In [85]:

```
# BEST RESULT FOR LINEAR REGRESSION
# FOR DEMOCRATIC : LASSO, PROJECT1 FEATRUES, alpha: 8000
# FOR REPUBLICAN : ELASTICNET, ALL FEATURES, alpha: 0

# BEST RESULT FOR CLASSIFICATION
# SVC, rbf kernel, no max-iter
```

In [86]:

```
test_data = pd.read_csv('demographics_test.csv', sep=',')
test_data_dropped = test_data.drop(['State', 'County', 'FIPS'], axis=1)
test_data_scaled = scaler.transform(test_data_dropped)

combine1_test = test_data_scaled
combine2_test = test_data_scaled[:, [0, 2, 3, 8, 12]]
combine3_test = test_data_scaled[:, [0, 1, 2, 3, 11, 12]]

output = {}
output['State'] = test_data['State']
output['County'] = test_data['County']

# Democratic
model_demo = Lasso(alpha=8000).fit(combine3_X, train_Y.iloc[:, 0])
output['Democratic'] = model_demo.predict(combine3_test)
# Republican
model_repu = ElasticNet(alpha=0).fit(combine1_X, train_Y.iloc[:, 1])
output['Republican'] = model_repu.predict(combine1_test)
# Party
model_party = SVC(kernel='rbf').fit(combine1_X, train_Y.iloc[:, 2])
output['Party'] = model_party.predict(combine1_test)

df = pd.DataFrame(output, columns=['State', 'County', 'Democratic', 'Republican', 'Party'])
df.to_csv('prediction_output.csv', sep=',')
```

```
C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWipykernel_launcher.py:17: UserWarning:

With alpha=0, this algorithm does not converge well. You are advised to use the LinearRegression estimator

C:WUsersWkhhh9WAnaconda3WenvsWml_projectWlibWsite-packagesWsklearnWlinear_modelW_coordinate_descent.py:531: UserWarning:

Coordinate descent with no regularization may lead to unexpected results and is discouraged.
```

In [ ]: