Student name (Student ID): Lai Siu Kwok (12354991)

Problem Definition

Heart failure is a serious disease that is difficult to cope with due to its sudden onset and there are many people who die of heart failure every year. In view of this, the aim of this project is to use machine learning techniques such as tests with different algorithms, to find out which algorithms perform well which is the highest accuracy, in this case. Therefore, the best algorithms can be used to predict the survival of patients with heart failure in order to help people who need the medication the most for reducing the number of deaths from heart failure.

Introduction

We will use K-fold cross-validation to compare 4 algorithms, respectively are logistic regression (LR), K Nearest Neighbors (KNN) Naive Bayes (NB), and K-Mean (KM). First, we will divide the dataset into training and testing set as a ratio of 7:3. Second, the normalization techniques will be deployed because of the different types of features. Third, the feature selection will be performed for finding the most significant two features in order to suit the algorithms. Forth, the visualization techniques of the output will give a clearer image for the explanation. Finally, The cross-validation (CV) scores obtained can then be used as criteria for choosing the best parameter of the models, also the model selection for future work in the end.

Library

In [1]:
```
from pandas import read_csv
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from scipy.stats import skew
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
%matplotlib inline
```

Dataset import

In [2]:
```
df = pd.read_csv('heart_failure_clinical_records_dataset.csv')
del df['time'] #The column 'time' is day/time property, not a useful fe
ature
print("Dimention: ", df.shape)
```

```
Dimention:  (299, 12)
```

Find if missing values is present

In [3]:
```
print(df.isnull().sum())
print("\nNo missing value in dataset")
```

```
age                         0
anaemia                     0
creatinine_phosphokinase    0
diabetes                    0
ejection_fraction           0
high_blood_pressure         0
platelets                   0
serum_creatinine            0
serum_sodium                0
sex                         0
smoking                     0
DEATH_EVENT                 0
dtype: int64

No missing value in dataset
```

Data Description

In [4]:
```
df.head()
```

Out[4]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_blood_p
0	75.0	0	582	0	20	1
1	55.0	0	7861	0	38	0
2	65.0	0	146	0	20	0
3	50.0	1	111	0	20	0
4	65.0	1	160	1	20	0

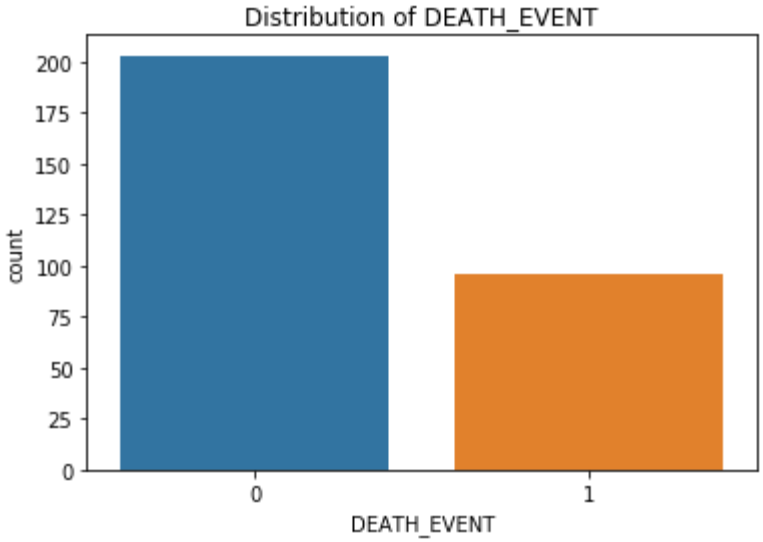
In [5]:
```
df.describe()
```

Out[5]:

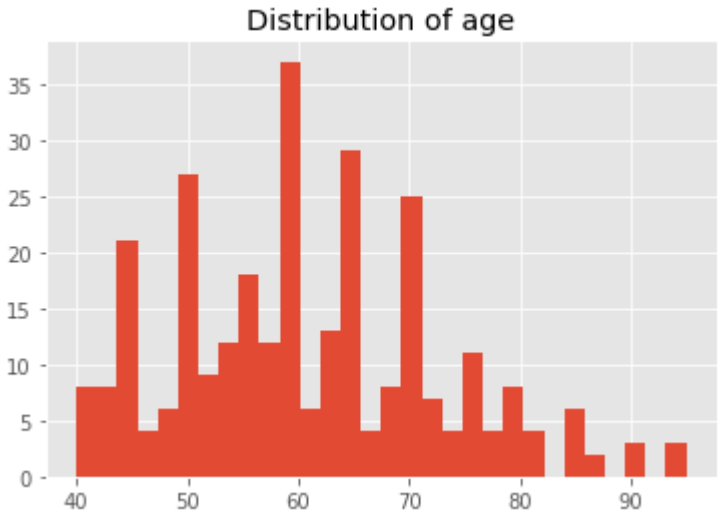
	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fractio
count	299.000000	299.000000	299.000000	299.000000	299.000000
mean	60.833893	0.431438	581.839465	0.418060	38.083612
std	11.894809	0.496107	970.287881	0.494067	11.834841
min	40.000000	0.000000	23.000000	0.000000	14.000000
25%	51.000000	0.000000	116.500000	0.000000	30.000000
50%	60.000000	0.000000	250.000000	0.000000	38.000000
75%	70.000000	1.000000	582.000000	1.000000	45.000000
max	95.000000	1.000000	7861.000000	1.000000	80.000000

In [6]:
```
sns.countplot(x="DEATH_EVENT", data = df)
pyplot.title("Distribution of DEATH_EVENT")
```

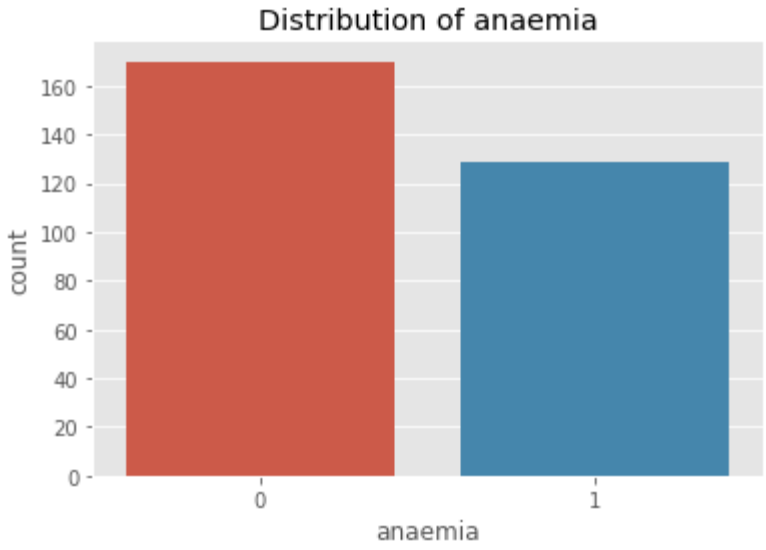
Out[6]:
```
Text(0.5,1,'Distribution of DEATH_EVENT')
```



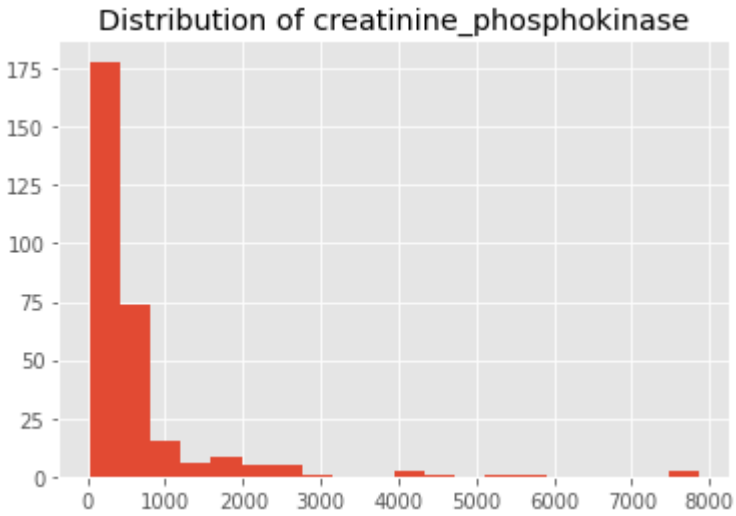
In [7]:
```
pyplot.style.use('ggplot')
pyplot.title("Distribution of age")
pyplot.hist(df['age'], bins=30)
print()
```



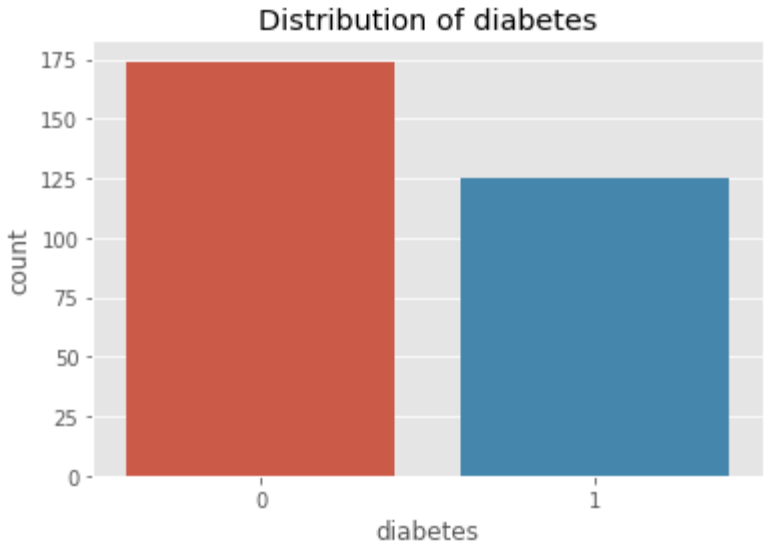
In [8]:
```
sns.countplot(x="anaemia", data = df)
pyplot.title("Distribution of anaemia")
print()
```



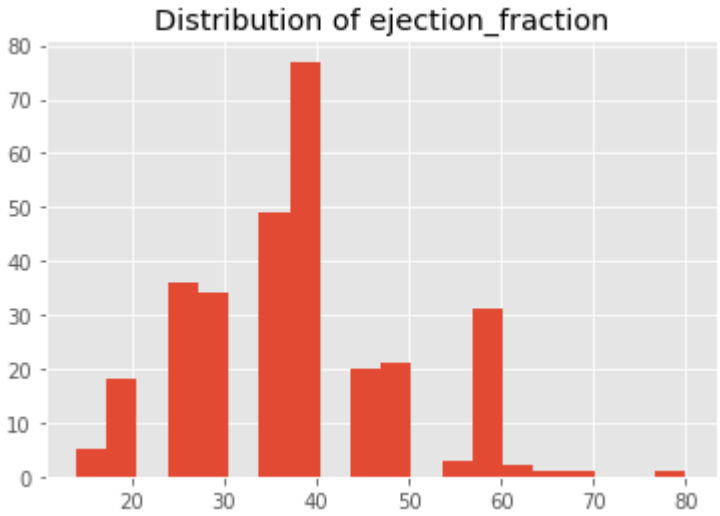
In [9]:
```
pyplot.style.use('ggplot')
pyplot.hist(df['creatinine_phosphokinase'], bins=20)
pyplot.title("Distribution of creatinine_phosphokinase")
print()
```



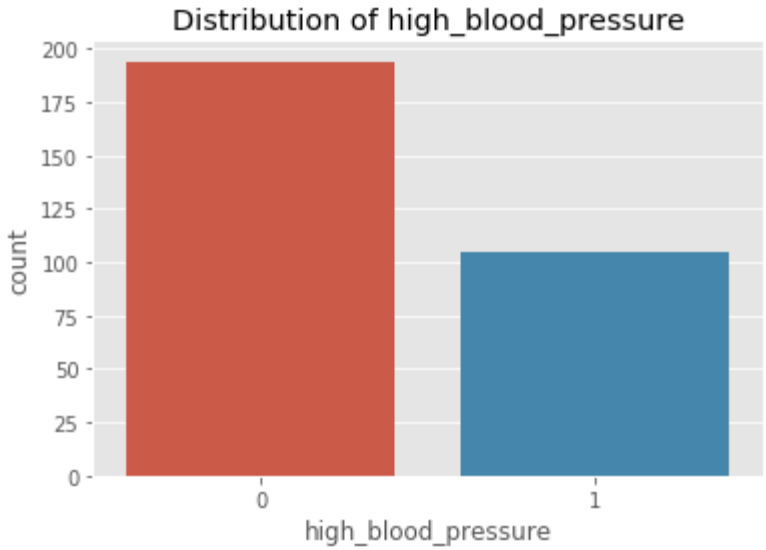
In [10]:
```
sns.countplot(x="diabetes", data = df)
pyplot.title("Distribution of diabetes")
print()
```



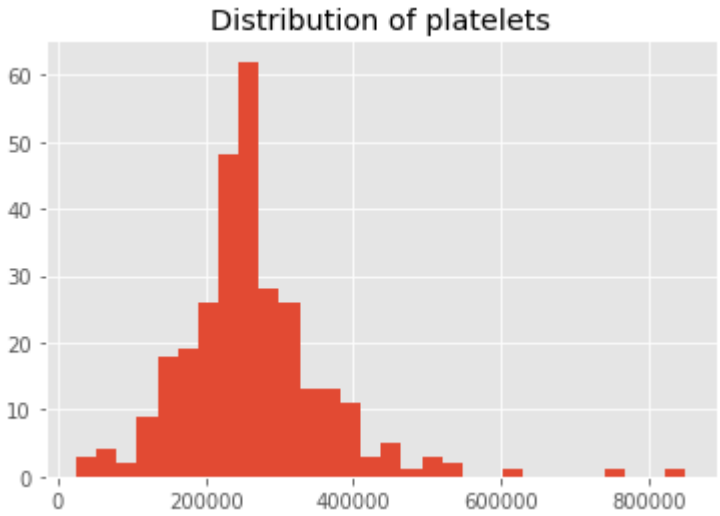
In [11]:
```
pyplot.style.use('ggplot')
pyplot.hist(df['ejection_fraction'], bins=20)
pyplot.title("Distribution of ejection_fraction")
print()
```



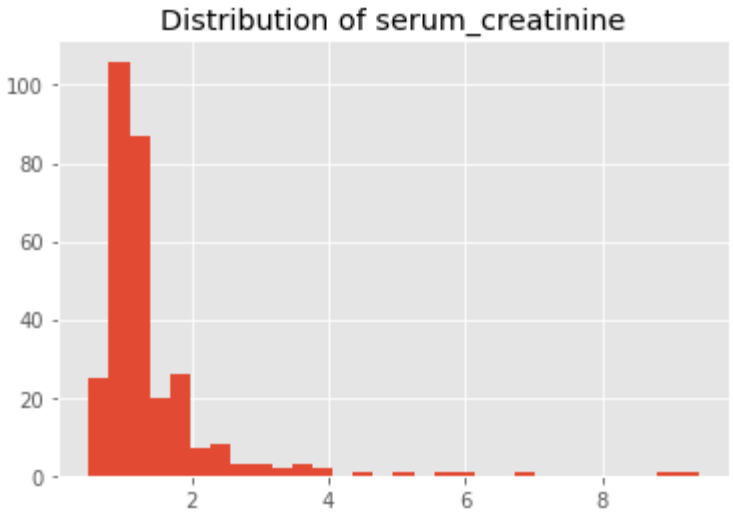
In [12]:
```
sns.countplot(x="high_blood_pressure", data = df)
pyplot.title("Distribution of high_blood_pressure")
print()
```



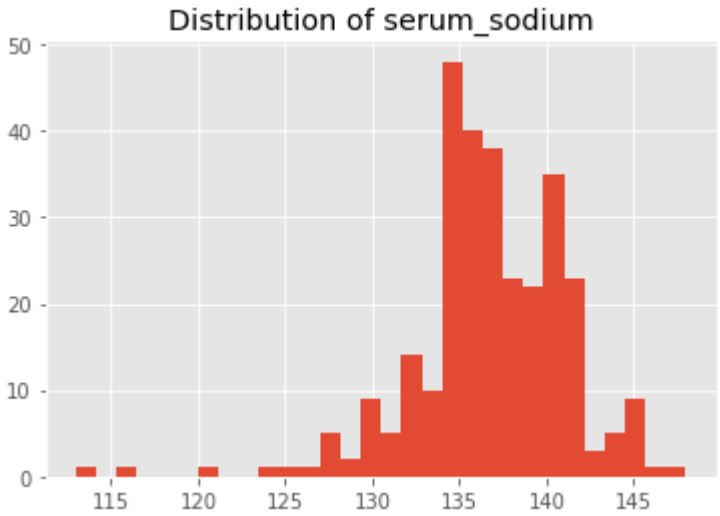
In [13]:
```
pyplot.style.use('ggplot')
pyplot.hist(df['platelets'], bins=30)
pyplot.title("Distribution of platelets")
print()
```



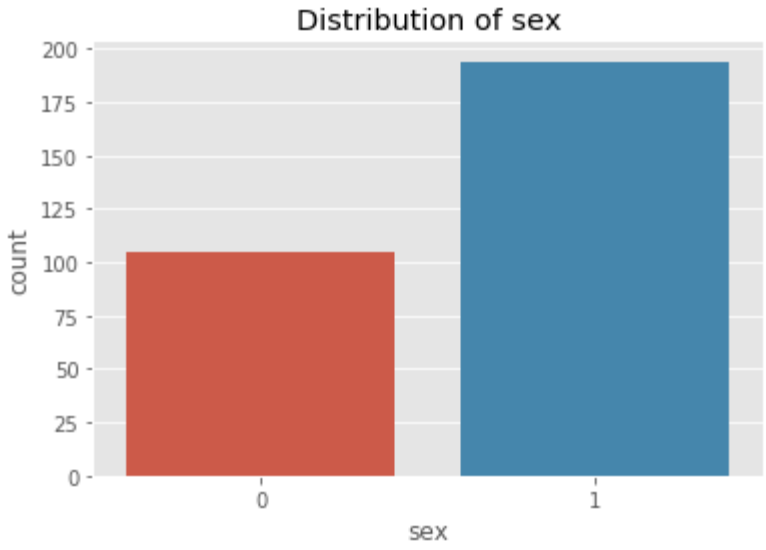
In [14]:
```
pyplot.style.use('ggplot')
pyplot.hist(df['serum_creatinine'], bins=30)
pyplot.title("Distribution of serum_creatinine")
print()
```



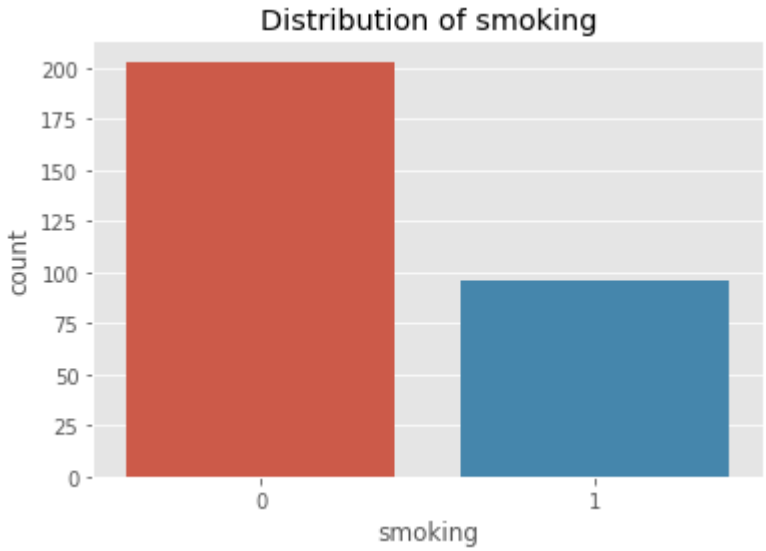
In [15]:
```
pyplot.style.use('ggplot')
pyplot.hist(df['serum_sodium'], bins=30)
pyplot.title("Distribution of serum_sodium")
print()
```



In [16]:
```
sns.countplot(x="sex", data = df)
pyplot.title("Distribution of sex")
print()
```



In [17]:
```
sns.countplot(x="smoking", data = df)
pyplot.title("Distribution of smoking")
print()
```



Correlation matrix

In [18]:
```
corr = df.corr()
corr.style.background_gradient(cmap='coolwarm').set_precision(2)
```

Out[18]:

	age	anaemia	creatinine_phosphokinase	diabetes	eject
age	1	0.088	-0.082	-0.1	0.06
anaemia	0.088	1	-0.19	-0.013	0.032
creatinine_phosphokinase	-0.082	-0.19	1	-0.0096	-0.04
diabetes	-0.1	-0.013	-0.0096	1	-0.00
ejection_fraction	0.06	0.032	-0.044	-0.0049	1
high_blood_pressure	0.093	0.038	-0.071	-0.013	0.024
platelets	-0.052	-0.044	0.024	0.092	0.072
serum_creatinine	0.16	0.052	-0.016	-0.047	-0.01
serum_sodium	-0.046	0.042	0.06	-0.09	0.18
sex	0.065	-0.095	0.08	-0.16	-0.15
smoking	0.019	-0.11	0.0024	-0.15	-0.06
DEATH_EVENT	0.25	0.066	0.063	-0.0019	-0.27

Interpretation

From the above plots, we can see there are no any abnormal phenomenon and distribution about the features. On the other hand, the correlation plot indicates that most of the features have a correlation relationship to the target. Therefore, we can go for conducting our proposed algorithms.

Preprocessing

Normalization

In [19]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
scaler.fit(df.drop('DEATH_EVENT',axis=1))
scaled_features = scaler.transform(df.drop('DEATH_EVENT',axis=1))
df_feat = pd.DataFrame(scaled_features,columns=df.columns[:-1])
df_feat.head()
```

Out[19]:

	age	anaemia	creatinine_phosphokinase	diabetes	ejection_fraction	high_bloo
0	0.636364	0.0	0.071319	0.0	0.090909	1.0
1	0.272727	0.0	1.000000	0.0	0.363636	0.0
2	0.454545	0.0	0.015693	0.0	0.090909	0.0
3	0.181818	1.0	0.011227	0.0	0.090909	0.0
4	0.454545	1.0	0.017479	1.0	0.090909	0.0

Split data into training / testing set

In [20]:
```
array = df_feat.values
X = array[:,:]
Y = df['DEATH_EVENT']

X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.
3, random_state=0)

print('Training set shape: ', X_train.shape, y_train.shape)
print('Testing set shape: ', X_test.shape, y_test.shape)
```

```
Training set shape:  (209, 11) (209,)
Testing set shape:  (90, 11) (90,)
```

Feature selection

In [21]:
```
from sklearn.feature_selection import chi2, SelectKBest, f_classif
ft = SelectKBest(chi2, k = 2).fit(X_train, y_train)
print('Score: ', ft.scores_)
print(df_feat.columns)
```

```
Score:  [1.9986432  0.81907185 0.2091677  0.0970547  1.94938899 0.410
02503
 0.01542149 2.17788074 0.27067962 0.12485051 0.09242322]
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking'],
      dtype='object')
```

In [22]:
```
ft = SelectKBest(f_classif, k= 2).fit(X_train, y_train)
print('Score: ', ft.scores_)
print(df_feat.columns)
```

```
Score:  [18.15532972  1.45934341  0.93239365  0.17039712 27.1092569
0.63534095
  0.29372328 19.790196   10.75667355  0.34983422  0.13294371]
Index(['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
       'ejection_fraction', 'high_blood_pressure', 'platelets',
       'serum_creatinine', 'serum_sodium', 'sex', 'smoking'],
      dtype='object')
```

Interpretation

From the result of feature selection, in order to fit the algorithms, we have to get two most important features. Then, we can notice that the features of 'ejection_fraction' and 'serum_creatinine' having the highest mark from the two criteria. Thus, 'ejection_fraction' and 'serum_creatinine' will be selected.

Remap the X set with correlated features

In [23]:
```
X_train = X_train[:,(4,7)]
X_test = X_test[:,(4,7)]
print("X_train:\n",X_train.shape)
print("X_test:\n",X_test.shape)

print("y_train:\n",y_train.shape)
print("y_test:\n",y_test.shape)
```

```
X_train:
 (209, 2)
X_test:
 (90, 2)
y_train:
 (209,)
y_test:
 (90,)
```

Algorithm implementation

Logistic Regression

Logistic Regression with Cross-Validation

In [24]:
```
from sklearn.linear_model import LogisticRegressionCV
clf = LogisticRegressionCV(Cs=[0.001, 0.01, 0.1, 1, 10, 100, 1000, 1000
0],
                           cv=10,
                           max_iter=1000,
                           random_state=0,
                           penalty='l2').fit(X_train, y_train)

print("Training Accuracy:", clf.score(X_train, y_train))
print("Coeff:\n", clf.coef_)
#print("The detail of iter across every class:\n", clf.n_iter_)
print("The best C across every class:", clf.C_)
```

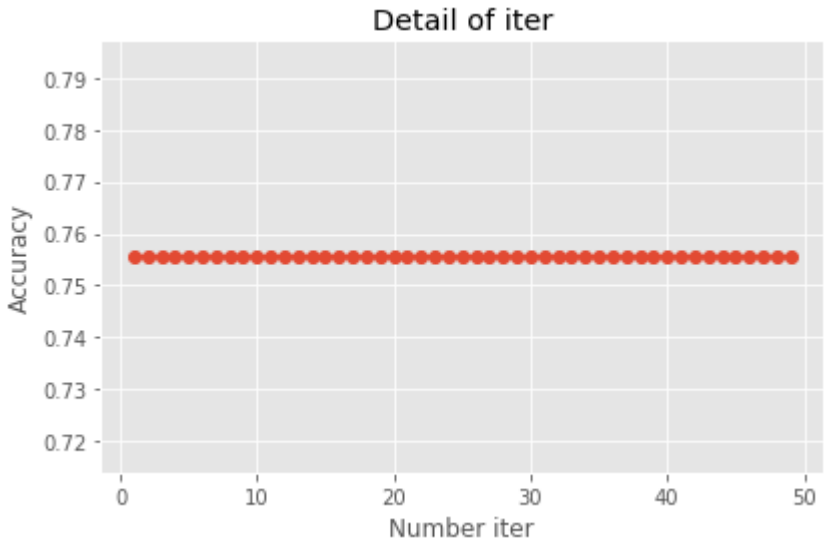
```
Training Accuracy: 0.7416267942583732
Coeff:
 [[-4.85696861  6.09768946]]
The best C across every class: [100.]
```

Logistic Regression with best C

In [25]:
```
from sklearn import metrics
import seaborn as sn

logreg_fit = LogisticRegression(C=float(clf.C_), max_iter=1000)
logreg_fit = logreg_fit.fit(X_train, y_train)
y_pred = logreg_fit.predict(X_test)
```

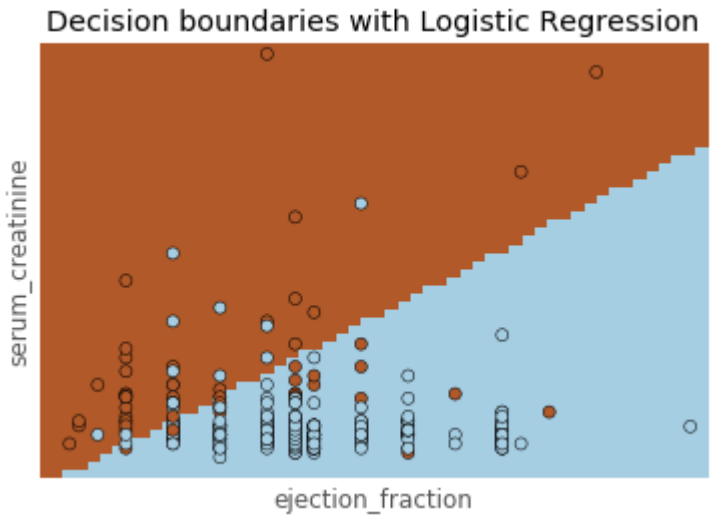
Model Evaluation

In [26]:
```
scores = []
for i in range(1, 1000):
    logreg = LogisticRegression(C=float(clf.C_), max_iter=i)
    logreg_trained = logreg.fit(X_train, y_train)
    scores.append(logreg_trained.score(X_test, y_test))

pyplot.plot(range(1, 50), scores[1:50], marker='o')
pyplot.title("Detail of iter")
pyplot.xlabel('Number iter')
pyplot.ylabel('Accuracy')
pyplot.tight_layout()
pyplot.show()
print("The graph indicates that the # of iter is within the range")
```



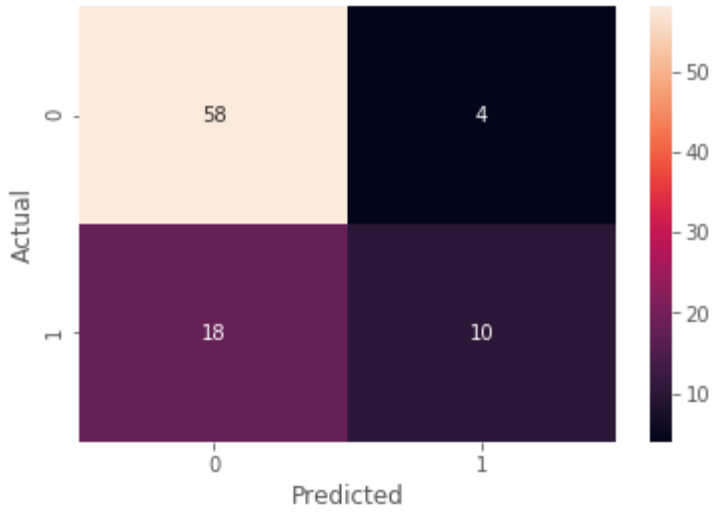

```
The graph indicates that the # of iter is within the range
```

In [27]:
```
import matplotlib.pyplot as plt
logreg_best = LogisticRegression(C=float(clf.C_), max_iter=1000)
logreg_fit = logreg_best.fit(X_train, y_train)

h = .02
X_min, X_max = X[:, 4].min() - .05, X[:, 4].max() + .05
y_min, y_max = X[:, 7].min() - .05, X[:, 7].max() + .05
xx, yy = np.meshgrid(np.arange(X_min, X_max, h), np.arange(y_min, y_ma
x, h))
Z = logreg.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(6, 4))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X[:, 4], X[:, 7], c=Y, edgecolors='k', cmap=plt.cm.Paired)
plt.xlabel('ejection_fraction')
plt.ylabel('serum_creatinine')
plt.title('Decision boundaries with Logistic Regression')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()
```



In [28]:
```
print(classification_report(y_test, y_pred))

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], col
names=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
pyplot.show()
LR_acc = metrics.accuracy_score(y_test, y_pred)
print('Testing set accuracy: ',LR_acc)
```

```
              precision    recall  f1-score   support

           0       0.76      0.94      0.84        62
           1       0.71      0.36      0.48        28

avg / total       0.75      0.76      0.73        90
```



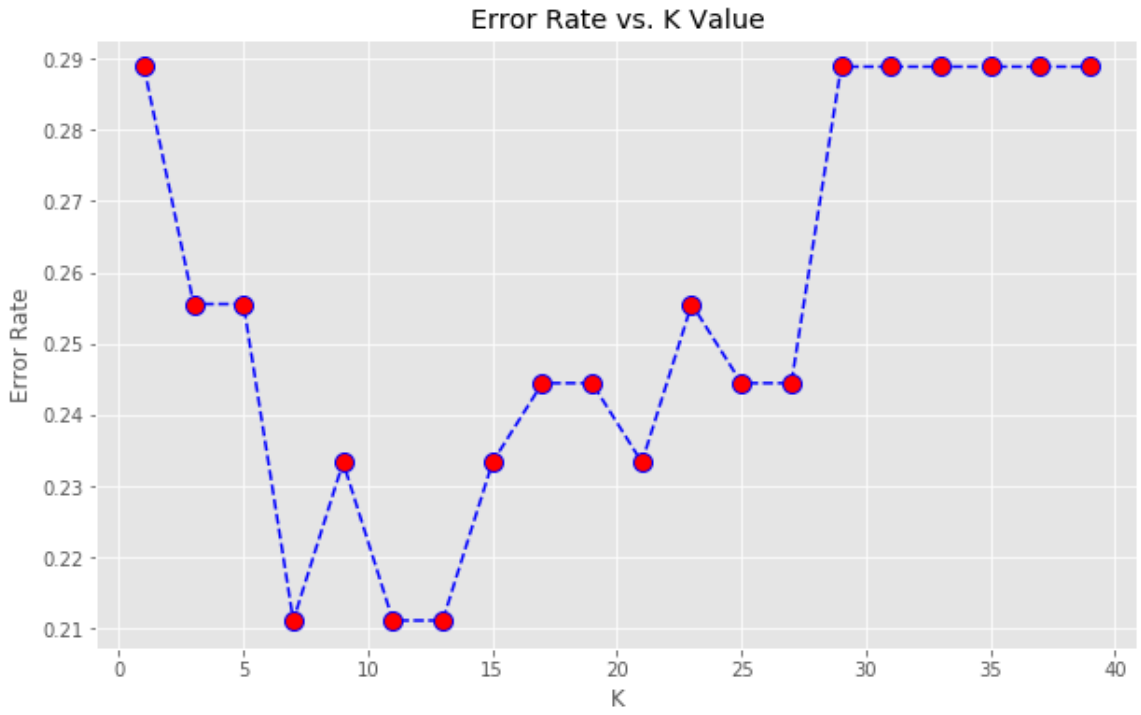

```
Testing set accuracy:  0.7555555555555555
```

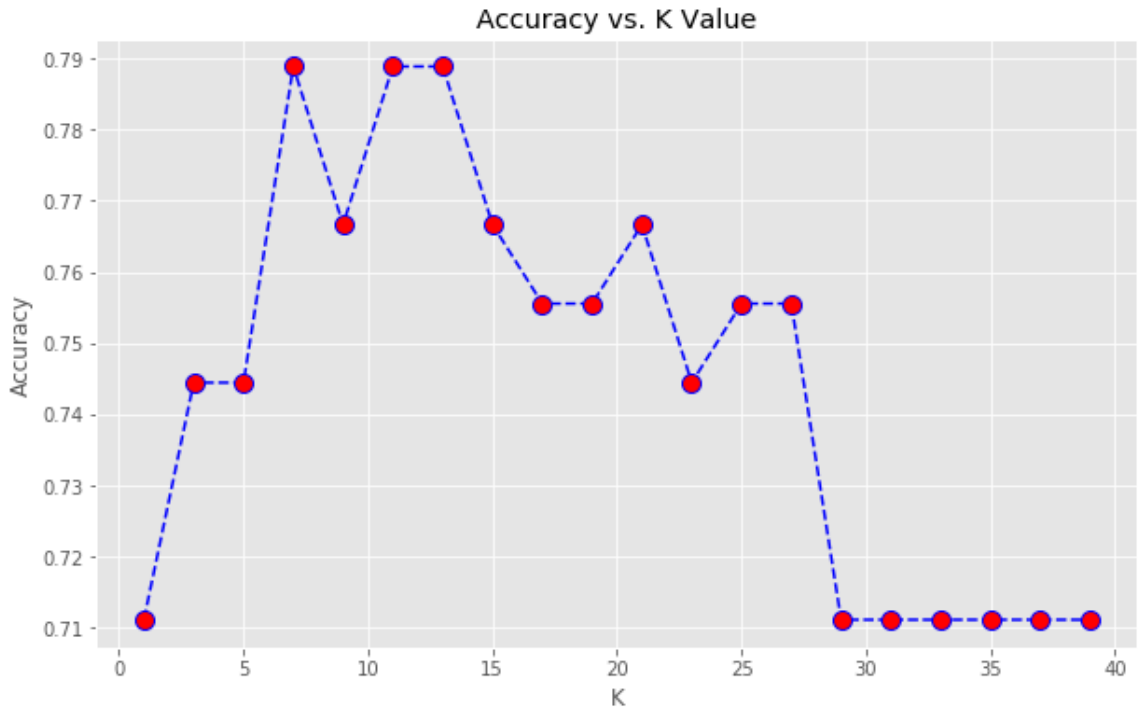
KNN

KNN with Cross-Validation

In [29]:
```
error_rate = []
acc = []
scoresCV = []
for i in range(1,40,2): # 1,3,5,7,9 ..... Law of thumb
    knn = KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train,y_train)
    pred_i = knn.predict(X_test)
    error_rate.append(np.mean(pred_i != y_test))

    #cross_val_score(knn, X_train, y_train, cv=10, scoring='accuracy')
    scores = metrics.accuracy_score(y_test, pred_i)
    acc.append(scores.mean())


pyplot.figure(figsize=(10,6))
pyplot.plot(range(1,40,2),error_rate,color='blue', linestyle='dashed',
marker='o',
         markerfacecolor='red', markersize=10)
pyplot.title('Error Rate vs. K Value')
pyplot.xlabel('K')
pyplot.ylabel('Error Rate')

pyplot.figure(figsize=(10,6))
pyplot.plot(range(1,40,2),acc,color='blue', linestyle='dashed', marker=
'o',
         markerfacecolor='red', markersize=10)
pyplot.title('Accuracy vs. K Value')
pyplot.xlabel('K')
pyplot.ylabel('Accuracy')

print("Error Rate suggests : 7, 11, 13")
print("Accuracy Rate suggests : 7, 11, 13")
```

```
Error Rate suggests : 7, 11, 13
Accuracy Rate suggests : 7, 11, 13
```





Hyperparameter Tuning

In [30]:
```
n_neighbors = [7,11,13]
grid_params = { 'n_neighbors' : n_neighbors,
                'metric' : ['minkowski','euclidean','manhattan']}
gs = GridSearchCV(KNeighborsClassifier(), grid_params, verbose = 1, cv=
10, n_jobs = -1)
g_res = gs.fit(X_train, y_train)
print(g_res.best_score_)
print(g_res.best_params_)
#print(g_res.best_index_)
# use the best hyperparameters
knn_best = KNeighborsClassifier(n_neighbors=n_neighbors[int(g_res.best_
index_)], metric = 'minkowski')
knn_best.fit(X_train, y_train)

print("The best K is", n_neighbors[int(g_res.best_index_)])
```

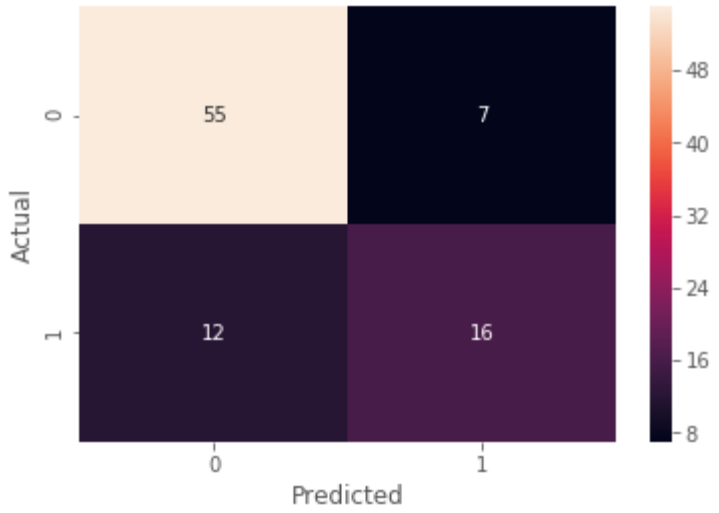
```
Fitting 10 folds for each of 9 candidates, totalling 90 fits
0.7751196172248804
{'metric': 'minkowski', 'n_neighbors': 7}
The best K is 7
[Parallel(n_jobs=-1)]: Done  90 out of  90 | elapsed:    3.1s finishe
d
```

Model Evaluation

KNN with best K

In [31]:
```
y_pred = knn_best.predict(X_test)

print(classification_report(y_test, y_pred))

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], col
names=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
pyplot.show()

KNN_acc = metrics.accuracy_score(y_test, y_pred)
print('Testing set accuracy: ', KNN_acc)
```

```
             precision    recall  f1-score   support

          0       0.82      0.89      0.85        62
          1       0.70      0.57      0.63        28

avg / total       0.78      0.79      0.78        90
```




```
Testing set accuracy:  0.7888888888888889
```

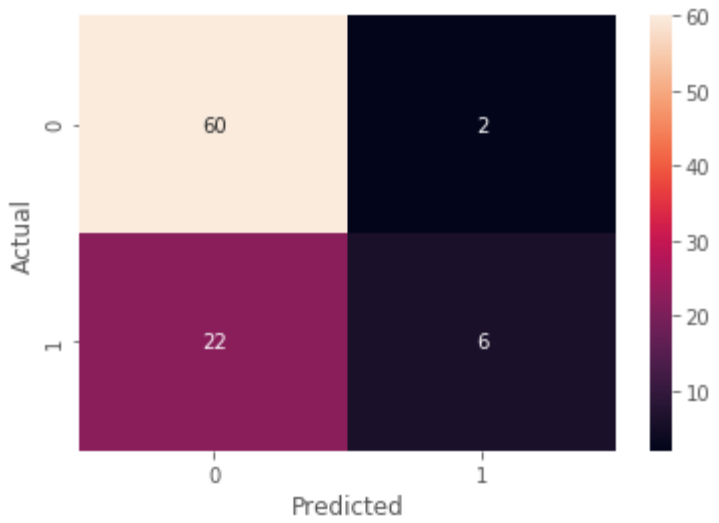
Naïve Bayes Classifier

In [32]:
```
model = GaussianNB()
trained_model = model.fit(X_train, y_train)
y_pred = trained_model.predict(X_test)
```

Model Evaluation

In [33]:
```
print(classification_report(y_test, y_pred))

confusion_matrix = pd.crosstab(y_test, y_pred, rownames=['Actual'], col
names=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
pyplot.show()

NB_acc = metrics.accuracy_score(y_test, y_pred)
print('Testing set accuracy: ', NB_acc)
```

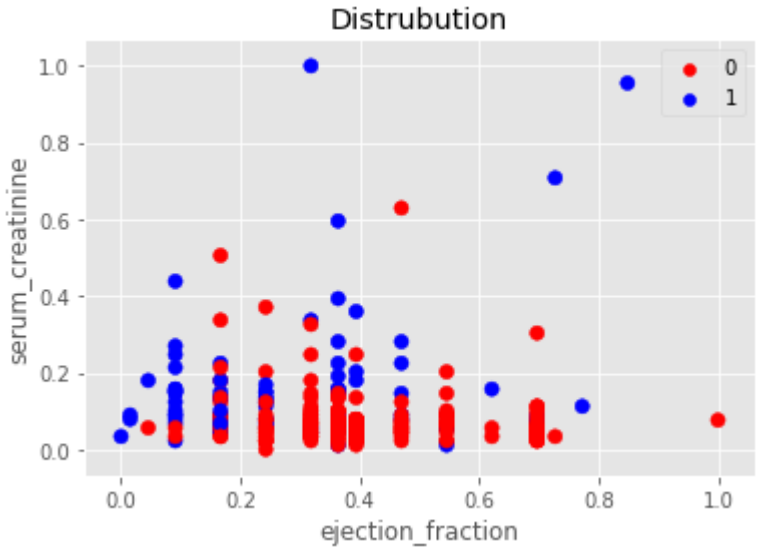
```
              precision    recall  f1-score   support

           0       0.73      0.97      0.83        62
           1       0.75      0.21      0.33        28

avg / total       0.74      0.73      0.68        90
```




```
Testing set accuracy:  0.7333333333333333
```

K-means Clustering

In [34]:
```
colors=np.array(["red", "blue"])
plt.xlabel('ejection_fraction')
plt.ylabel('serum_creatinine')
plt.title('Distrubution')

plt.scatter(X[:, 4], X[:, 7], c=colors[Y], s=50)
for label, c in enumerate(colors):
    plt.scatter([], [], c=c, label=str(label))
plt.legend();
```

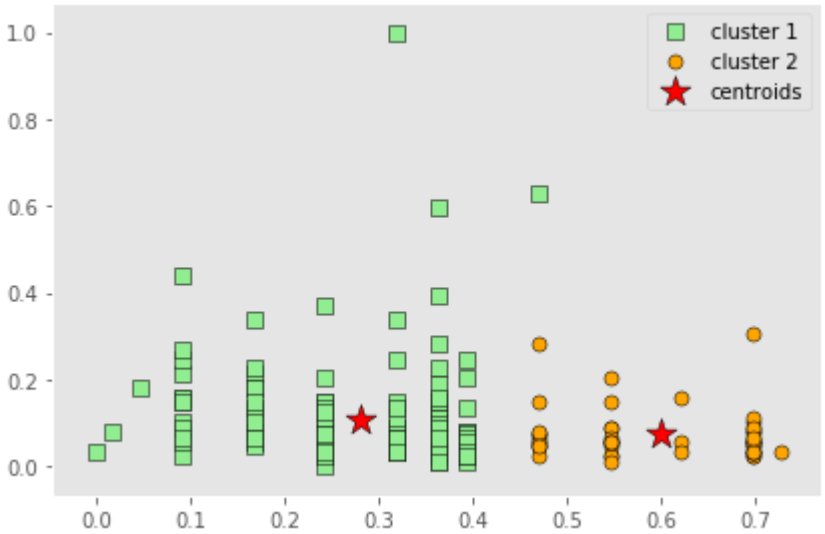


Model fitting

In [35]:
```
from sklearn.cluster import KMeans

km = KMeans(n_clusters=2,
            init='random',
            n_init=100,
            max_iter=300,
            tol=1e-04, # stop criteria
            random_state=0)

km_fit = km.fit(X_train,y_train)
km_pred = km_fit.predict(X_test)

#print(help(km_pred))
#print(km_fit.inertia_)
print("Centroid 1:{}\nCentroid 2:{}".format(km_fit.cluster_centers_[0],
km_fit.cluster_centers_[1]))
print("# of Iter:",km_fit.n_iter_)
```

```
Centroid 1:[0.27977892 0.10716557]
Centroid 2:[0.59909091 0.07361798]
# of Iter: 12
```

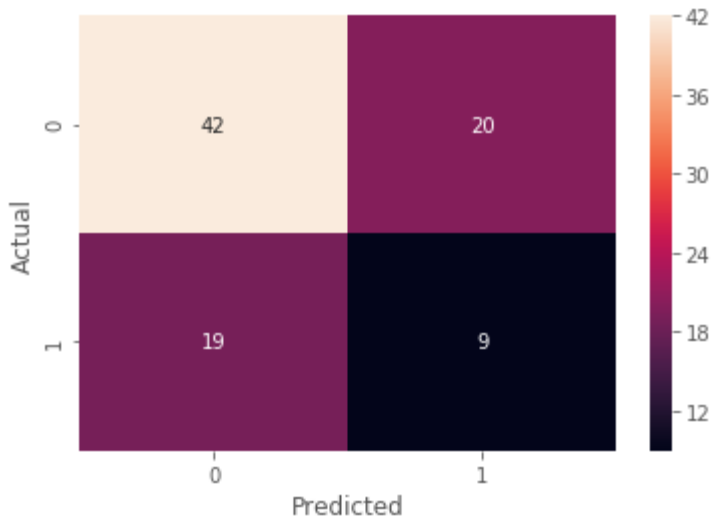
In [36]:
```
y_km = km.fit_predict(X_train)

plt.scatter(X_train[y_km == 0, 0],
            X_train[y_km == 0, 1],
            s=50, c='lightgreen',
            marker='s', edgecolor='black',
            label='cluster 1')
plt.scatter(X_train[y_km == 1, 0],
            X_train[y_km == 1, 1],
            s=50, c='orange',
            marker='o', edgecolor='black',
            label='cluster 2')
plt.scatter(km.cluster_centers_[:, 0],
            km.cluster_centers_[:, 1],
            s=250, marker='*',
            c='red', edgecolor='black',
            label='centroids')
plt.legend(scatterpoints=1)
plt.grid()
plt.tight_layout()

plt.show()
```



Model Evaluation

In [37]:
```
print(classification_report(y_test, km_pred))

confusion_matrix = pd.crosstab(y_test, km_pred, rownames=['Actual'], co
lnames=['Predicted'])
sn.heatmap(confusion_matrix, annot=True)
pyplot.show()

KM_acc = metrics.accuracy_score(y_test, km_pred)
print('Testing set accuracy: ',KM_acc)
```

```
              precision    recall  f1-score   support

           0       0.69      0.68      0.68        62
           1       0.31      0.32      0.32        28

avg / total       0.57      0.57      0.57        90
```

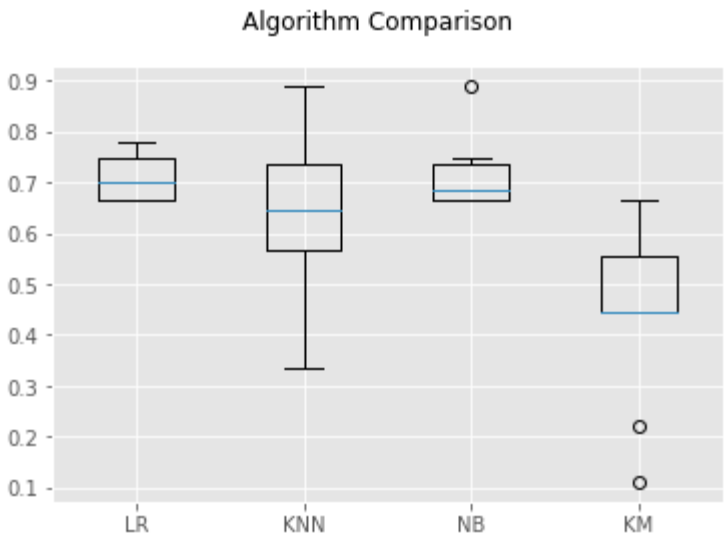



```
Testing set accuracy:  0.5666666666666667
```

Algorithm Comparison

In [38]:
```
names = ['LR','KNN','NB','KM']

models = []
models.append(('LR', logreg_best))
models.append(('KNN', knn_best))
models.append(('NB', GaussianNB()))
models.append(('KM', km))

results = []
scoring = 'accuracy'

for name, model in models:

    model_fit = model.fit(X_train,y_train)
    cv_results = cross_val_score(model_fit, X_test, y_test, cv=10, scor
ing=scoring)
    results.append(cv_results)
    names.append(name)

    pt = "%s:\tMean:%f\tStd:%f" % (name, cv_results.mean(), cv_results.
std())
    print(pt)

fig = pyplot.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
pyplot.boxplot(results)
ax.set_xticklabels(names)
pyplot.show()
```

```
LR:     Mean:0.712222   Std:0.044777
KNN:    Mean:0.645278   Std:0.143942
NB:     Mean:0.712222   Std:0.066889
KM:     Mean:0.444444   Std:0.157135
```



Reference

1. Kevin(2016). A Complete Guide to K-Nearest-Neighbors with Applications in Python and R. Retrieved from https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/ (https://kevinzakka.github.io/2016/07/13/k-nearest-neighbor/)

2. Arvai(2020). K-Means Clustering in Python: A Practical Guide. Retrieved from https://realpython.com/k-means-clustering-python/ (https://realpython.com/k-means-clustering-python/)

3. scikit-learn.org(2020). Confusion matrix. Retrieved from https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html (https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html)

4. Tutorial 4, 5, 6, 7

5. UCL dataset: https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records (https://archive.ics.uci.edu/ml/datasets/Heart+failure+clinical+records)