In [1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
import xgboost as xgb

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.metrics import recall_score, precision_score, classification_report,accuracy_sc
ore,confusion_matrix, roc_curve, auc, roc_curve,accuracy_score,plot_confusion_matrix
from sklearn.preprocessing import StandardScaler, normalize
from sklearn.decomposition import PCA
from scipy import ndimage
import seaborn as sns
```

In [2]:
```python
#Load dataset
Train_Data = pd.read_csv("data_set_ALL_AML_train.csv")
Test_Data = pd.read_csv("data_set_ALL_AML_independent.csv")
labels = pd.read_csv("actual.csv", index_col = 'patient')
```

In [3]:
```python
Train_Data.head()
```

Out[3]:

	Gene Description	Gene Accession Number	1	call	2	call.1	3	call.2	4	call.3	...	29	call.33	30	call.34	31	call.35	3
0	AFFX-BioB-5_at (endogenous control)	AFFX-BioB-5_at	-214	A	-139	A	-76	A	-135	A	...	15	A	-318	A	-32	A	-12
1	AFFX-BioB-M_at (endogenous control)	AFFX-BioB-M_at	-153	A	-73	A	-49	A	-114	A	...	-114	A	-192	A	-49	A	-7
2	AFFX-BioB-3_at (endogenous control)	AFFX-BioB-3_at	-58	A	-1	A	-307	A	265	A	...	2	A	-95	A	49	A	-3
3	AFFX-BioC-5_at (endogenous control)	AFFX-BioC-5_at	88	A	283	A	309	A	12	A	...	193	A	312	A	230	P	33
4	AFFX-BioC-3_at (endogenous control)	AFFX-BioC-3_at	-295	A	-264	A	-376	A	-419	A	...	-51	A	-139	A	-367	A	-18

5 rows × 78 columns

In [4]:
```python
#check for nulls
print(Train_Data.isna().sum().max())
print(Test_Data.isna().sum().max())
```

```
0
0
```

In [5]:
```python
#drop 'call' columns
cols = [col for col in Test_Data.columns if 'call' in col]
test = Test_Data.drop(cols, 1)
cols = [col for col in Train_Data.columns if 'call' in col]
train = Train_Data.drop(cols, 1)
```

In [6]:
```python
#Join all the data
patients = [str(i) for i in range(1, 73, 1)]
df_all = pd.concat([train, test], axis = 1)[patients]
```

In [7]:
```python
#transpose rows and columns
df_all = df_all.T
```

In [8]:
```python
#Encode the categorical columns
df_all["patient"] = pd.to_numeric(patients)
labels["cancer"]= pd.get_dummies(labels.cancer, drop_first=True)
# add the cancer column to train data

Data = pd.merge(df_all, labels, on="patient")
```

In [9]:
```python
Data.head()
```

Out[9]:

	0	1	2	3	4	5	6	7	8	9	...	7121	7122	7123	7124	7125	7126	7127	7128	patient	ca
0	-214	-153	-58	88	-295	-558	199	-176	252	206	...	-125	389	-37	793	329	36	191	-37	1	
1	-139	-73	-1	283	-264	-400	-330	-168	101	74	...	-36	442	-17	782	295	11	76	-14	2	
2	-76	-49	-307	309	-376	-650	33	-367	206	-215	...	33	168	52	1138	777	41	228	-41	3	
3	-135	-114	265	12	-419	-585	158	-253	49	31	...	218	174	-110	627	170	-50	126	-91	4	
4	-106	-125	-76	168	-230	-284	4	-122	70	252	...	57	504	-26	250	314	14	56	-25	5	

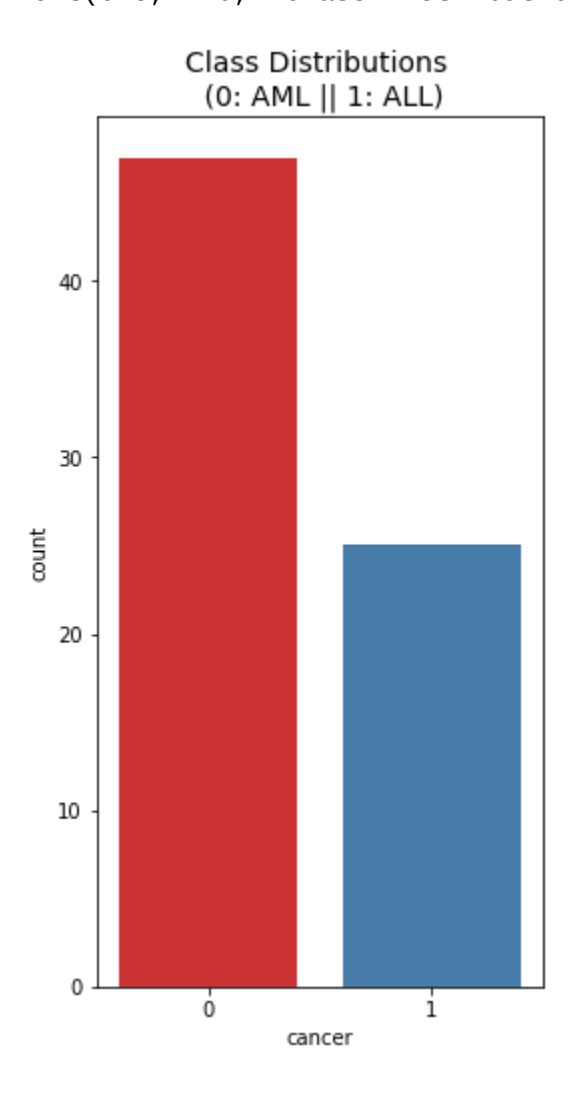
5 rows × 7131 columns

In [10]:
```python
Data['cancer'].value_counts()
```

Out[10]:
```
0    47
1    25
Name: cancer, dtype: int64
```

In [11]:
```python
plt.figure(figsize=(4,8))
colors = ["AML", "ALL"]
sns.countplot('cancer', data=Data, palette = "Set1")
plt.title('Class Distributions \n (0: AML || 1: ALL)', fontsize=14)
```

Out[11]: `Text(0.5, 1.0, 'Class Distributions \n (0: AML || 1: ALL)')`



In [12]:
```python
#X -> matrix of independent variable
#y -> vector of dependent variable
X, y = Data.drop(columns=["cancer"]), Data["cancer"]
```

In [13]:
```python
X
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9	...	7120	7121	7122	7123	7124	7125	7126	7127	7128	pati
0	-214	-153	-58	88	-295	-558	199	-176	252	206	...	511	-125	389	-37	793	329	36	191	-37	
1	-139	-73	-1	283	-264	-400	-330	-168	101	74	...	837	-36	442	-17	782	295	11	76	-14	
2	-76	-49	-307	309	-376	-650	33	-367	206	-215	...	1199	33	168	52	1138	777	41	228	-41	
3	-135	-114	265	12	-419	-585	158	-253	49	31	...	835	218	174	-110	627	170	-50	126	-91	
4	-106	-125	-76	168	-230	-284	4	-122	70	252	...	649	57	504	-26	250	314	14	56	-25	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
67	-154	-136	49	180	-257	-273	141	-123	52	878	...	540	13	1075	-45	524	249	40	-68	-1	
68	-79	-118	-30	68	-110	-264	-28	-61	40	-217	...	617	-34	738	11	742	234	72	109	-30	
69	-55	-44	12	129	-108	-301	-222	-133	136	320	...	318	35	241	-66	320	174	-4	176	40	
70	-59	-114	23	146	-171	-227	-73	-126	-6	149	...	760	-38	201	-55	348	208	0	74	-12	
71	-131	-126	-50	211	-206	-287	-34	-114	62	341	...	697	3	1046	27	874	393	34	237	-2	

72 rows × 7130 columns

In [14]:
```python
y
```

Out[14]:
```
0     0
1     0
2     0
3     0
4     0
     ..
67    0
68    0
69    0
70    0
71    0
Name: cancer, Length: 72, dtype: uint8
```

In [15]:
```python
#split the dataset
X_train, X_test, y_train, y_test =  train_test_split(X,y,test_size = 0.25, random_state= 0)
```

In [16]:
```python
#feature scaling
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

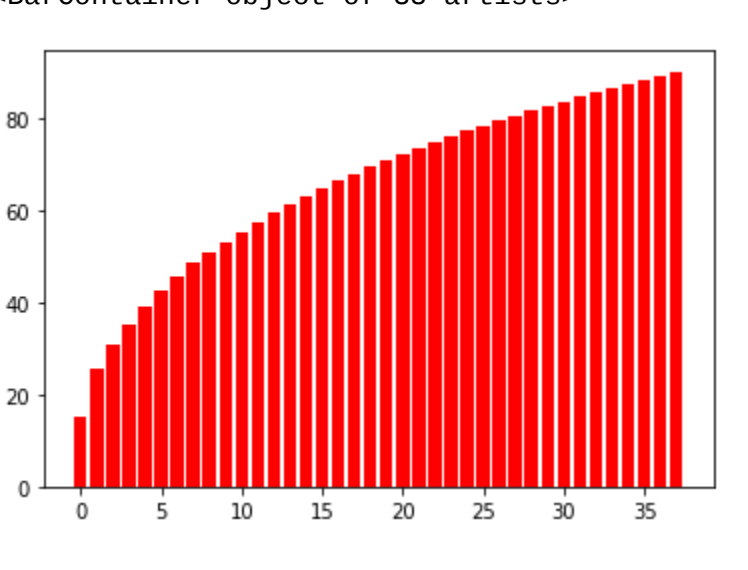
In [17]:
```python
X_train.shape
```

Out[17]: `(54, 7130)`

In [18]:
```python
#Dimentionality reduction using Principal Component Analysis(PCA)
pca = PCA()
pca.fit_transform(X_train)

total = sum(pca.explained_variance_)
k = 0
current_variance = 0
while current_variance/total < 0.90:
    current_variance += pca.explained_variance_[k]
    k = k + 1

print(k, " features explain around 90% of the variance. From 7129 features to ", k, ", not t
oo bad.", sep='')

pca = PCA(n_components=k)
X_train_pca = pca.fit(X_train)
X_train_pca = pca.transform(X_train)
X_test_pca = pca.transform(X_test)

var_exp = pca.explained_variance_ratio_.cumsum()
var_exp = var_exp*100
plt.bar(range(k), var_exp,color = 'r')
```

```
38 features explain around 90% of the variance. From 7129 features to 38, not too bad.
```

Out[18]: `<BarContainer object of 38 artists>`

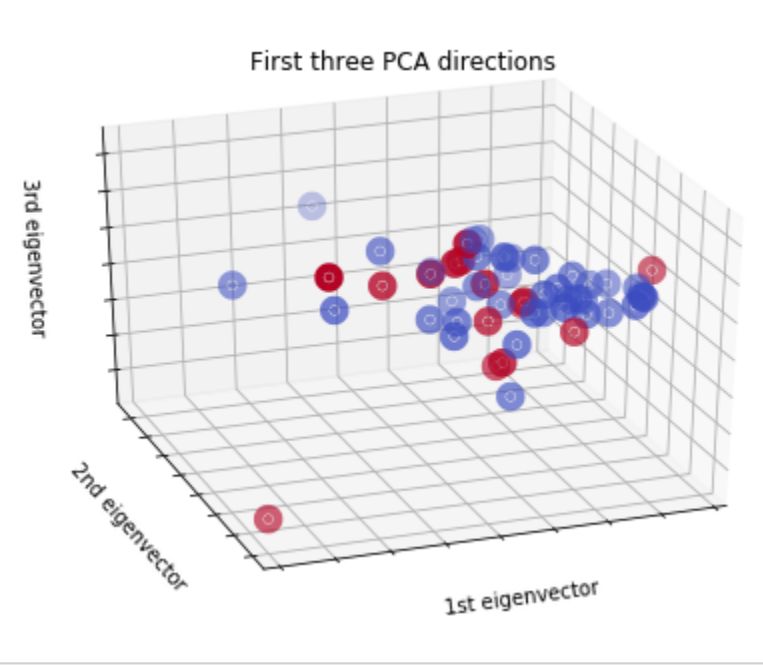


In [19]:
```python
pca.n_components_
```

Out[19]: `38`

In [20]:
```python
#3D plot of first 3 principal components
from mpl_toolkits.mplot3d import Axes3D

pca3 = PCA(n_components=3).fit(X_train)
X_train_reduced = pca3.transform(X_train)

plt.clf()
fig = plt.figure(1, figsize=(10,6))
ax = Axes3D(fig, elev=-150, azim=110,)
ax.scatter(X_train_reduced[:, 0], X_train_reduced[:, 1], X_train_reduced[:, 2], c = y_train,
cmap='coolwarm', linewidths=10)
ax.set_title("First three PCA directions")
ax.set_xlabel("1st eigenvector")
ax.w_xaxis.set_ticklabels([])
ax.set_ylabel("2nd eigenvector")
ax.w_yaxis.set_ticklabels([])
ax.set_zlabel("3rd eigenvector")
ax.w_zaxis.set_ticklabels([])
```

Out[20]: `[]`



In [21]:
```python
from sklearn.utils import resample
from collections import Counter

print("Before Upsampling:-")
print(Counter(y_train))

# Let's use SMOTE to oversample
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X_train_ov, y_train_ov = oversample.fit_resample(X_train_pca,y_train)


print("After Upsampling:-")
print(Counter(y_train_ov))
```

```
Before Upsampling:-
Counter({0: 39, 1: 15})
After Upsampling:-
Counter({0: 39, 1: 39})
```

In [22]:
```python
pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\windows\anaconda3\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\windows\anaconda3\lib\site-packag
es (from imblearn) (0.8.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\windows\anaconda3\lib\site-packages
(from imbalanced-learn->imblearn) (1.19.5)
Requirement already satisfied: joblib>=0.11 in c:\users\windows\anaconda3\lib\site-packages
(from imbalanced-learn->imblearn) (0.14.1)
Requirement already satisfied: scipy>=0.19.1 in c:\users\windows\anaconda3\lib\site-packages
(from imbalanced-learn->imblearn) (1.4.1)
Requirement already satisfied: scikit-learn>=0.24 in c:\users\windows\anaconda3\lib\site-pack
ages (from imbalanced-learn->imblearn) (0.24.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\windows\anaconda3\lib\site-pa
ckages (from scikit-learn>=0.24->imbalanced-learn->imblearn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [23]:
```python
from sklearn.utils import resample
from collections import Counter

print("Before Upsampling:-")
print(Counter(y_train))

# Let's use SMOTE to oversample
from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X_train_ov, y_train_ov = oversample.fit_resample(X_train_pca,y_train)


print("After Upsampling:-")
print(Counter(y_train_ov))
```

```
Before Upsampling:-
Counter({0: 39, 1: 15})
After Upsampling:-
Counter({0: 39, 1: 39})
```

In [24]:
```python
#Hyperprameter optimization for SVM
# do a grid search
svc_params = [{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
              {'C': [1, 10, 100, 1000], 'kernel': ['rbf'], 'gamma': [0.1, 0.2, 0.3, 0.4, 0.5
, 0.6, 0.7, 0.8, 0.9]}]

search = GridSearchCV(SVC(), svc_params, n_jobs=-1, verbose=1)
search.fit(X_train_ov, y_train_ov)

best_accuracy = search.best_score_ #to get best score
best_parameters = search.best_params_ #to get best parameters
# select best svc
best_svc = search.best_estimator_
best_svc
```

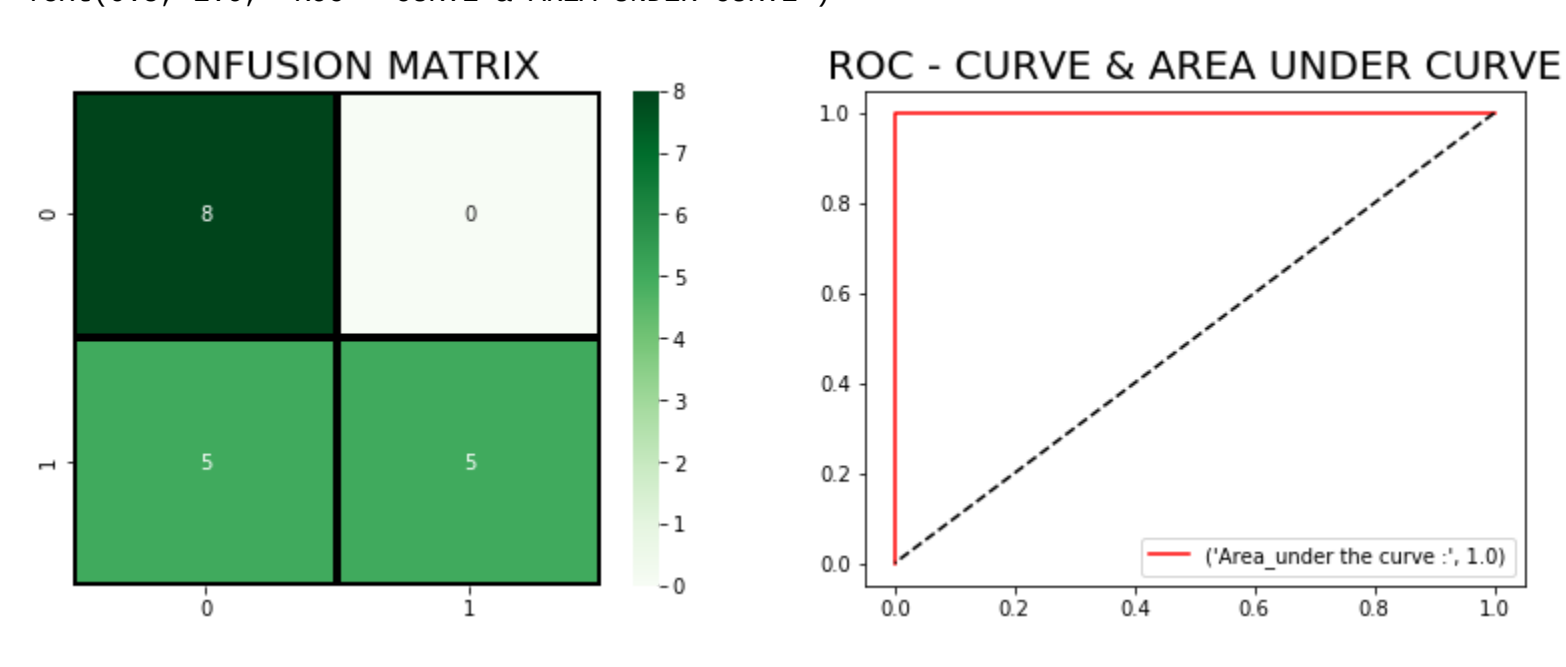
```
Fitting 5 folds for each of 40 candidates, totalling 200 fits
```

Out[24]: `SVC(C=1, kernel='linear')`

In [25]:
```python
#build SVM model with best parameters
svc_model = SVC(C=1, kernel='linear',probability=True)

svc_model.fit(X_train_ov, y_train_ov)

prediction=svc_model.predict(X_test_pca)

acc_svc = accuracy_score(prediction,y_test)
print('The accuracy of SVM is', acc_svc)
print ("\nClassification report :\n",(classification_report(y_test,prediction)))

#Confusion matrix
plt.figure(figsize=(13,10))
plt.subplot(221)
sns.heatmap(confusion_matrix(y_test,prediction),annot=True, cmap='Greens', fmt = "d",linecol
or="k",linewidths=3)
plt.title("CONFUSION MATRIX",fontsize=20)

#ROC curve and Area under the curve plotting
predicting_probabilites = svc_model.predict_proba(X_test_pca)[:,1]
fpr,tpr,thresholds = roc_curve(y_test,predicting_probabilites)
plt.subplot(222)
plt.plot(fpr,tpr,label = ("Area_under the curve :",auc(fpr,tpr)),color = "r")
plt.plot([1,0],[1,0],linestyle = "dashed",color ="k")
plt.legend(loc = "best")
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
```

```
The accuracy of SVM is 0.7222222222222222

Classification report :
               precision    recall  f1-score   support

           0       0.62      1.00      0.76         8
           1       1.00      0.50      0.67        10

    accuracy                           0.72        18
   macro avg       0.81      0.75      0.71        18
weighted avg       0.83      0.72      0.71        18
```

Out[25]: `Text(0.5, 1.0, 'ROC - CURVE & AREA UNDER CURVE')`



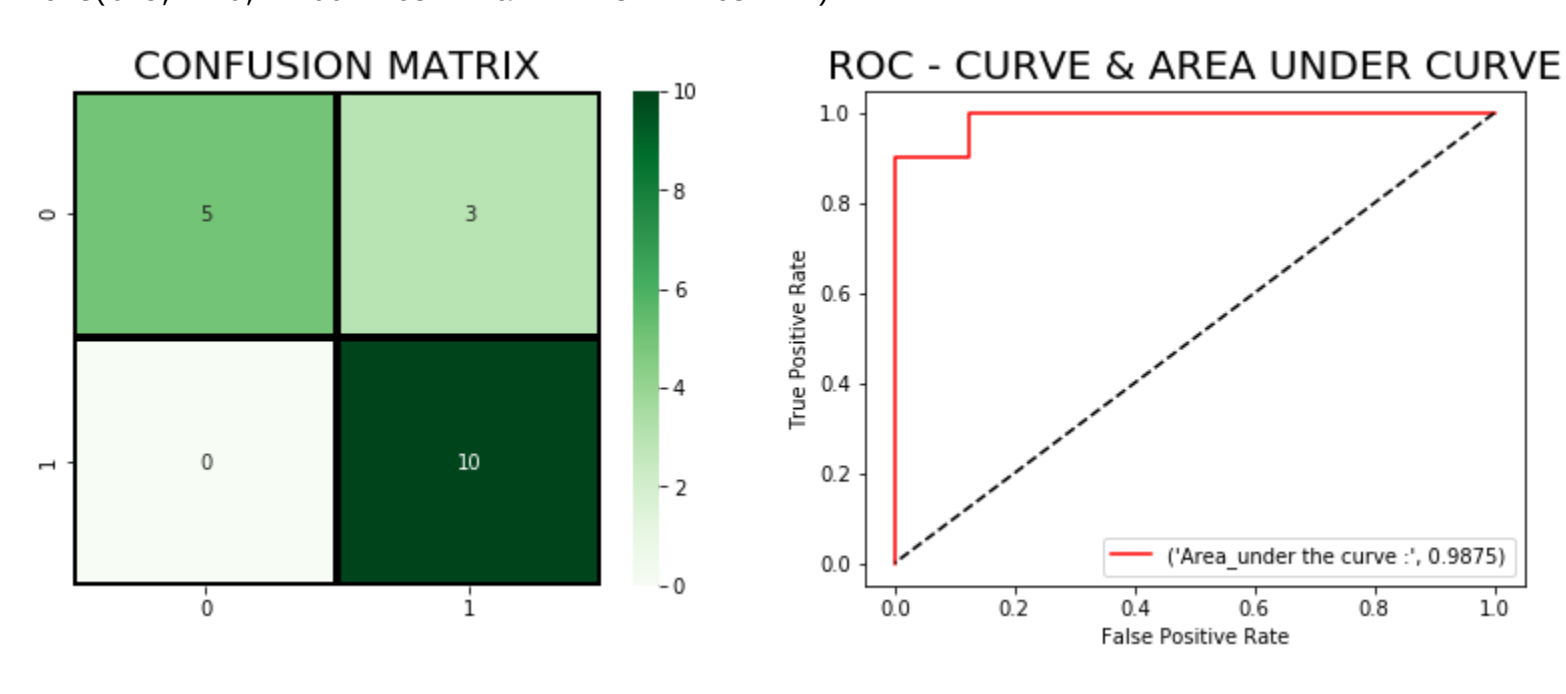
In [26]:
```python
#Hyperprameter optimization for Logistic regression
log_grid = {'C': [1e-03, 1e-2, 1e-1, 1, 10],
            'penalty': ['l1', 'l2']}

log_model = GridSearchCV(estimator=LogisticRegression(solver='liblinear'),
                  param_grid=log_grid,
                  cv=3,
                  scoring='accuracy')
log_model.fit(X_train_ov, y_train_ov)


best_accuracy = log_model.best_score_ #to get best score
best_parameters = log_model.best_params_ #to get best parameters
# select best svc
best_lr = log_model.best_estimator_
best_lr
```

Out[26]: `LogisticRegression(C=0.001, solver='liblinear')`

In [27]:
```python
#Logistic Regression
lr_model = LogisticRegression(C=0.001, solver='liblinear')

lr_model.fit(X_train_ov,y_train_ov)

prediction=lr_model.predict(X_test_pca)

acc_log = accuracy_score(prediction,y_test)
print('Validation accuracy of Logistic Regression is', acc_log)
print ("\nClassification report :\n",(classification_report(y_test,prediction)))

#Confusion matrix
plt.figure(figsize=(13,10))
plt.subplot(221)
sns.heatmap(confusion_matrix(y_test,prediction),annot=True,cmap="Greens",fmt = "d",linecolor
="k",linewidths=3)
plt.title("CONFUSION MATRIX",fontsize=20)

#ROC curve and Area under the curve plotting
predicting_probabilites = lr_model.predict_proba(X_test_pca)[:,1]
fpr,tpr,thresholds = roc_curve(y_test,predicting_probabilites)
plt.subplot(222)
plt.plot(fpr,tpr,label = ("Area_under the curve :",auc(fpr,tpr)),color = "r")
plt.plot([1,0],[1,0],linestyle = "dashed",color ="k")
plt.legend(loc = "best")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
```

```
Validation accuracy of Logistic Regression is 0.8333333333333334

Classification report :
               precision    recall  f1-score   support

           0       1.00      0.62      0.77         8
           1       0.77      1.00      0.87        10

    accuracy                           0.83        18
   macro avg       0.88      0.81      0.82        18
weighted avg       0.87      0.83      0.82        18
```

Out[27]: `Text(0.5, 1.0, 'ROC - CURVE & AREA UNDER CURVE')`



In [28]:
```python
#Model evaluation
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'Logistic Regression'],

    'Score': [acc_svc,  acc_log]})
models.sort_values(by='Score', ascending=False)
```

Out[28]:

	Model	Score
1	Logistic Regression	0.833333
0	Support Vector Machines	0.722222

In [29]:
```python
knn_param = {
    "n_neighbors": [i for i in range(1,30,5)],
    "weights": ["uniform", "distance"],
    "algorithm": ["ball_tree", "kd_tree", "brute"],
    "leaf_size": [1, 10, 30],
    "p": [1,2]
}
search = GridSearchCV(KNeighborsClassifier(), knn_param, n_jobs=-1, verbose=1)
search.fit(X_train_ov, y_train_ov)

best_accuracy = search.best_score_ #to get best score
best_parameters = search.best_params_ #to get best parameters
# select best svc
best_knn = search.best_estimator_
best_knn
```

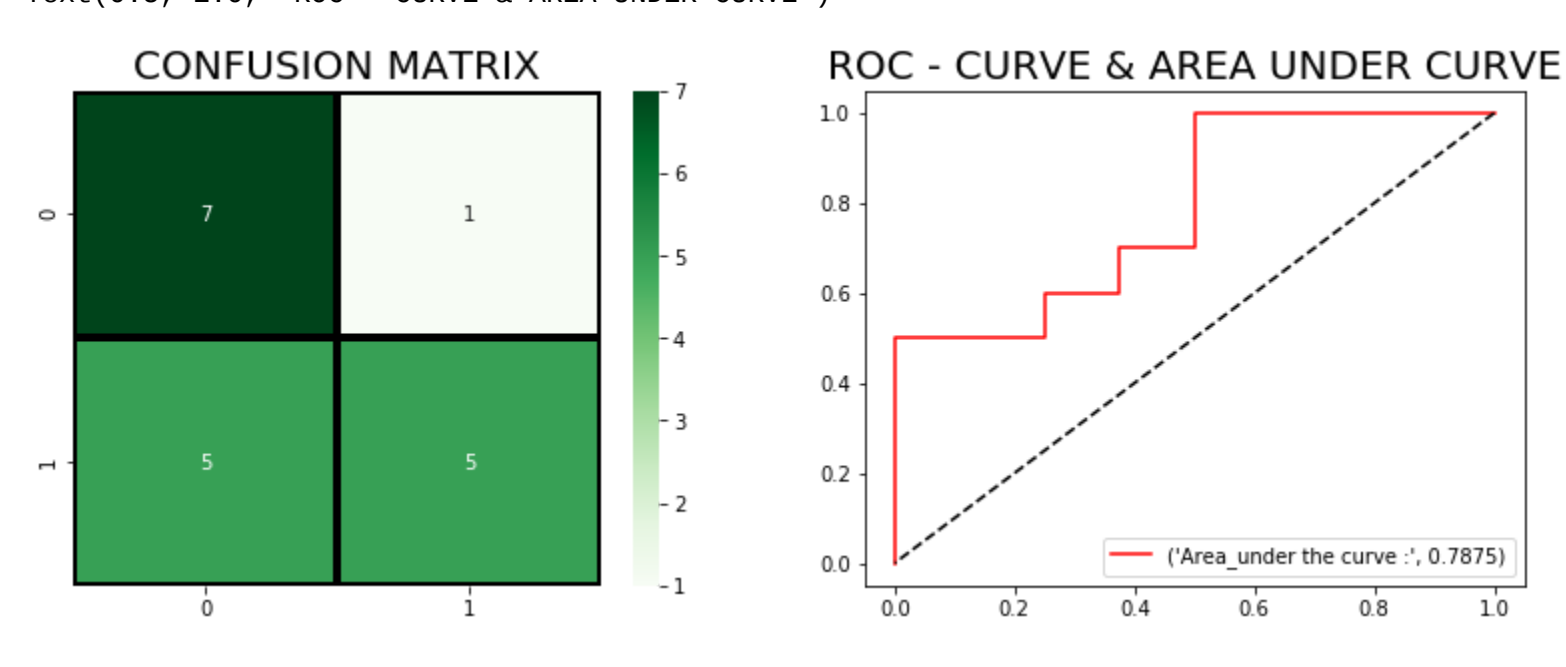
```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
```

Out[29]: `KNeighborsClassifier(algorithm='ball_tree', leaf_size=1, n_neighbors=1)`

In [30]:
```python
#Hyperprameter optimization for KNN
knn_model = KNeighborsClassifier(algorithm='ball_tree', leaf_size=1, n_neighbors=6,
                     weights='distance')

knn_model.fit(X_train_ov,y_train_ov)
prediction=knn_model.predict(X_test_pca)

acc_knn = accuracy_score(prediction,y_test)
print('The accuracy of K-NN is', acc_knn)
print ("\nClassification report :\n",(classification_report(y_test,prediction)))

#Confusion matrix
plt.figure(figsize=(13,10))
plt.subplot(221)
sns.heatmap(confusion_matrix(y_test,prediction),annot=True, cmap='Greens', fmt = "d",linecol
or="k",linewidths=3)
plt.title("CONFUSION MATRIX",fontsize=20)

#ROC curve and Area under the curve plotting
predicting_probabilites = knn_model.predict_proba(X_test_pca)[:,1]
fpr,tpr,thresholds = roc_curve(y_test,predicting_probabilites)
plt.subplot(222)
plt.plot(fpr,tpr,label = ("Area_under the curve :",auc(fpr,tpr)),color = "r")
plt.plot([1,0],[1,0],linestyle = "dashed",color ="k")
plt.legend(loc = "best")
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
```

```
The accuracy of K-NN is 0.6666666666666666

Classification report :
               precision    recall  f1-score   support

           0       0.58      0.88      0.70         8
           1       0.83      0.50      0.62        10

    accuracy                           0.67        18
   macro avg       0.71      0.69      0.66        18
weighted avg       0.72      0.67      0.66        18
```

Out[30]: `Text(0.5, 1.0, 'ROC - CURVE & AREA UNDER CURVE')`



In [31]:
```python
#Hyperprameter optimization for Decision trees
params = {'max_leaf_nodes': list(range(2, 100)), 'min_samples_split': [2, 3, 4, 5, 6], 'max_
depth':[3,4,5,6,7,8]}
decision_search = GridSearchCV(DecisionTreeClassifier(random_state=42), params, verbose=1, c
v=3)

decision_search.fit(X_train_ov, y_train_ov)


best_accuracy = decision_search.best_score_ #to get best score
best_parameters = decision_search.best_params_ #to get best parameters
# select best svc
best_ds = decision_search.best_estimator_
best_ds
```

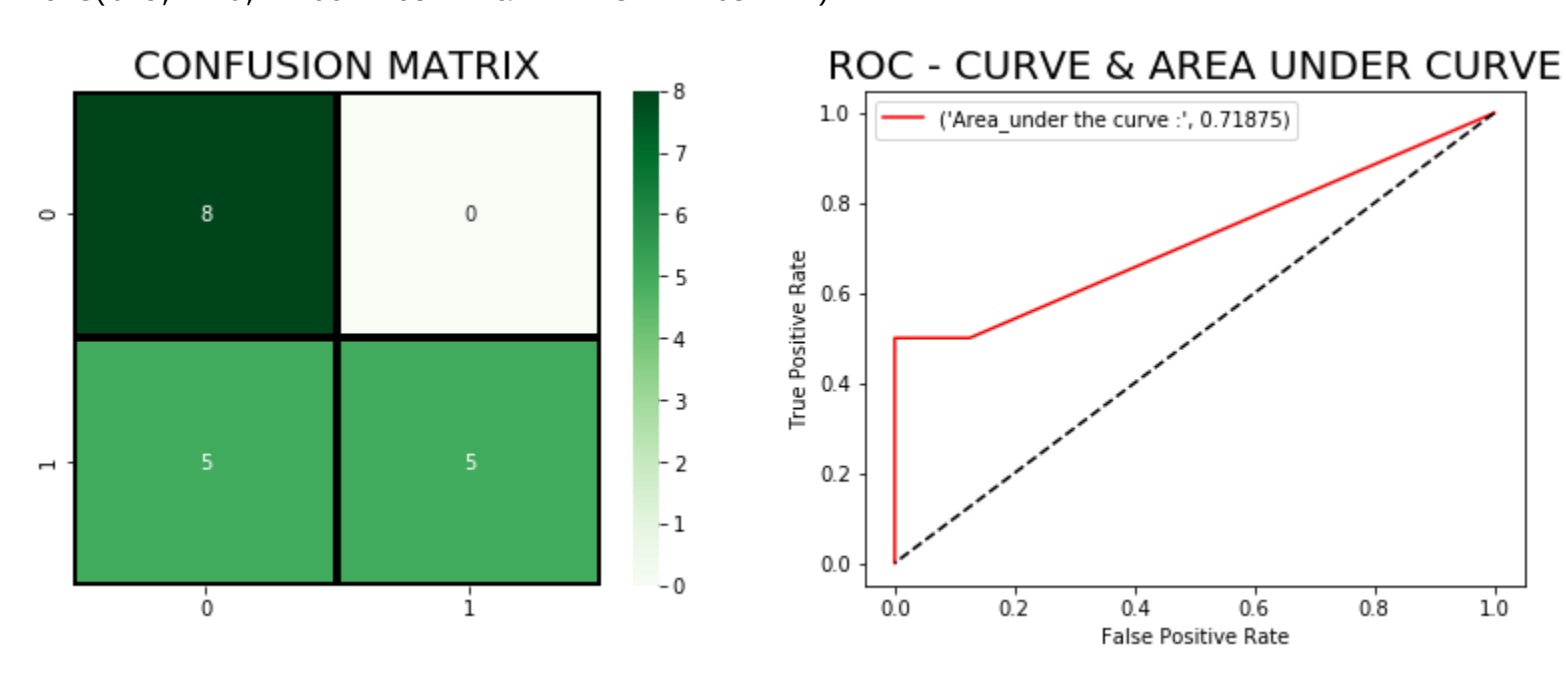
```
Fitting 3 folds for each of 2940 candidates, totalling 8820 fits
```

Out[31]: `DecisionTreeClassifier(max_depth=3, max_leaf_nodes=5, random_state=42)`

In [32]:
```python
#Decision Tree
ds_model = DecisionTreeClassifier(max_depth=3, max_leaf_nodes=3, random_state=42)

ds_model.fit(X_train_ov,y_train_ov)

prediction=ds_model.predict(X_test_pca)

acc_decision_tree = accuracy_score(prediction,y_test)
print('Validation accuracy of Decision Tree is', acc_decision_tree)
print ("\nClassification report :\n",(classification_report(y_test,prediction)))

#Confusion matrix
plt.figure(figsize=(13,10))
plt.subplot(221)
sns.heatmap(confusion_matrix(y_test,prediction),annot=True,cmap="Greens",fmt = "d",linecolor
="k",linewidths=3)
plt.title("CONFUSION MATRIX",fontsize=20)

#ROC curve and Area under the curve plotting
predicting_probabilites = ds_model.predict_proba(X_test_pca)[:,1]
fpr,tpr,thresholds = roc_curve(y_test,predicting_probabilites)
plt.subplot(222)
plt.plot(fpr,tpr,label = ("Area_under the curve :",auc(fpr,tpr)),color = "r")
plt.plot([1,0],[1,0],linestyle = "dashed",color ="k")
plt.legend(loc = "best")
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("ROC - CURVE & AREA UNDER CURVE",fontsize=20)
```

```
Validation accuracy of Decision Tree is 0.7222222222222222

Classification report :
               precision    recall  f1-score   support

           0       0.62      1.00      0.76         8
           1       1.00      0.50      0.67        10

    accuracy                           0.72        18
   macro avg       0.81      0.75      0.71        18
weighted avg       0.83      0.72      0.71        18
```

Out[32]: `Text(0.5, 1.0, 'ROC - CURVE & AREA UNDER CURVE')`



In [35]:
```python
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression', 'Decision Tree'],

    'Score': [acc_svc, acc_knn, acc_log, acc_decision_tree]})
models.sort_values(by='Score', ascending=False)
```

Out[35]:

	Model	Score
2	Logistic Regression	0.833333
0	Support Vector Machines	0.722222
3	Decision Tree	0.722222
1	KNN	0.666667

In [ ]: