# ML Lab 10: 26th March, 2025

## Name: Soumyadeep Ganguly

## Reg No: 24MDT0082

In [62]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [63]:
```
df = pd.read_csv('lab10data.csv')
df.head()
```

Out[63]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal | target |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 48 | 1 | 2 | 124 | 255 | 1 | 1 | 175 | 0 | 0.0 | 2 | 2 | 2 | 1 |
| **1** | 68 | 0 | 2 | 120 | 211 | 0 | 0 | 115 | 0 | 1.5 | 1 | 0 | 2 | 1 |
| **2** | 46 | 1 | 0 | 120 | 249 | 0 | 0 | 144 | 0 | 0.8 | 2 | 0 | 3 | 0 |
| **3** | 60 | 1 | 0 | 130 | 253 | 0 | 1 | 144 | 1 | 1.4 | 2 | 1 | 3 | 0 |
| **4** | 43 | 1 | 0 | 115 | 303 | 0 | 1 | 181 | 0 | 1.2 | 1 | 0 | 2 | 1 |

In [64]:
```
df.dtypes
```

Out[64]:
```
age         int64
sex         int64
cp          int64
trestbps    int64
chol        int64
fbs         int64
restecg     int64
thalach     int64
exang       int64
oldpeak   float64
slope       int64
ca          int64
thal        int64
target      int64
dtype: object
```

# Preprocessing

In [65]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [66]:
```
scaler = MinMaxScaler()
X = df.drop(['target'], axis=1)
y = df['target']

scaled_X = scaler.fit_transform(X)
X = pd.DataFrame(scaled_X, columns = X.columns)
X.head()
```

Out[66]:

| | age | sex | cp | trestbps | chol | fbs | restecg | thalach | exang | oldpeak | slope | ca | thal |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.395833 | 1.0 | 0.666667 | 0.306122 | 0.294521 | 1.0 | 0.5 | 0.793893 | 0.0 | 0.000000 | 1.0 | 0.50 | 0.666667 |
| **1** | 0.812500 | 0.0 | 0.666667 | 0.265306 | 0.194064 | 0.0 | 0.0 | 0.335878 | 0.0 | 0.267857 | 0.5 | 0.00 | 0.666667 |
| **2** | 0.354167 | 1.0 | 0.000000 | 0.265306 | 0.280822 | 0.0 | 0.0 | 0.557252 | 0.0 | 0.142857 | 1.0 | 0.00 | 1.000000 |
| **3** | 0.645833 | 1.0 | 0.000000 | 0.367347 | 0.289954 | 0.0 | 0.5 | 0.557252 | 1.0 | 0.250000 | 1.0 | 0.25 | 1.000000 |
| **4** | 0.291667 | 1.0 | 0.000000 | 0.214286 | 0.404110 | 0.0 | 0.5 | 0.839695 | 0.0 | 0.214286 | 0.5 | 0.00 | 0.666667 |

# PCA

In [67]:
```
from sklearn.decomposition import PCA
```

In [68]:
```
pca = PCA(n_components = 2)
pc_X = pca.fit_transform(X)
pc_X
```

Out[68]:
```
array([[-0.23522174, -0.7111645 ],
       [-0.61442471,  0.48110119],
       [-0.06443809, -0.35336159],
       [ 0.69975976,  0.08597526],
       [-0.05093584, -0.30573675],
       [ 0.1003509 , -0.2138113 ],
       [ 0.74877193,  0.09960479],
       [-0.21849872, -0.68918824],
       [ 0.46663659, -0.013086  ],
       [-0.62516878,  0.48582954],
       [-0.624672  ,  0.44984363],
       [-0.36929451, -0.53567358],
       [ 0.78364549, -0.12752742],
       [-0.81717358,  0.34101822],
       [ 0.235398  ,  1.00066192],
       [-0.29051287,  0.59591091],
       [ 0.71093233,  0.12279361],
       [ 0.86473216,  0.17145438],
       [-0.20406023, -0.43625508],
       [ 0.61705649, -0.14238798],
       [-0.62827583,  0.45517818],
       [ 0.83791947,  0.22383137],
       [ 0.69987153,  0.07697176],
       [-0.66440001,  0.43927683],
       [-0.47643074,  0.61143065],
       [-0.1259226 , -0.43429024],
       [ 0.07064487, -0.549728  ],
       [-0.83387913,  0.34080572],
       [-0.03930769, -0.36759135],
       [ 0.26501976, -0.41687917],
       [-0.23167894, -0.68999279],
       [ 0.8776946 ,  0.18705249],
       [ 0.11930381, -0.23791664],
       [ 0.31762362, -0.21318989],
       [ 0.59730846,  0.03852486],
       [-0.08579792, -0.36699754],
       [-0.30030033, -0.51949687],
       [-0.85538856,  0.39287624],
       [-0.7251578 ,  0.38146   ],
       [-0.10311769, -0.36264012],
       [ 0.96964597, -0.07519052],
       [ 0.71831183,  0.18420095],
       [ 0.27177819,  0.98294635],
       [ 0.00273046, -0.28721551],
       [-0.78589445,  0.39571354],
```

```
       [-0.27394347, -0.44914303],
       [ 0.52823321, -0.0065165 ],
       [-0.12888377, -0.34046886],
       [ 0.70976969,  0.10056697],
       [ 0.71949439,  0.18620373],
       [ 0.85154601,  0.10942926],
       [ 0.84396617,  0.19642803],
       [ 0.91642401,  0.21855696],
       [ 0.0298716 , -0.30259733],
       [ 0.63508579,  0.03703488],
       [ 0.17108728, -0.21097597],
       [ 0.42856007,  0.8406353 ],
       [-0.07481201, -0.26284782],
       [ 0.35098562, -0.12584041],
       [ 0.01194653, -0.32054236],
       [-0.82226992,  0.31321272],
       [-0.7013165 ,  0.21640359],
       [-0.67365892,  0.47763115],
       [-0.07954792, -0.31912431],
       [-0.20317366, -0.37465955],
       [-0.39219682, -0.55098082],
       [ 0.8348998 ,  0.1511027 ],
       [-0.27350548, -0.47464119],
       [-0.81183902,  0.29909735],
       [-0.13529798, -0.39928764],
       [-0.63536939,  0.50345193],
       [ 0.88389682,  0.20636924],
       [ 0.10874391, -0.27944135],
       [-0.05540845, -0.35300344],
       [ 0.82227196,  0.18383936],
       [-0.24595949,  0.45774802],
       [ 0.85361623,  0.18058447],
       [-0.19231146, -0.4148893 ],
       [ 0.83233772,  0.15275175],
       [-0.04685948, -0.59919274],
       [-0.69689453,  0.40800494],
       [-0.82288329,  0.34240302],
       [-0.49051069,  0.57321495],
       [-0.78690786,  0.33419871],
       [-0.1699777 , -0.38990308],
       [-0.2496717 , -0.5849888 ],
       [ 0.91334767,  0.1698216 ],
       [-0.69708937,  0.38014988],
       [ 0.48964409, -0.11973564],
       [-0.72799235,  0.42992157],
```

```
 [-0.4500464 ,  0.61183695],
 [-0.37848341, -0.53904371],
 [ 0.62010297,  0.02533211],
 [-0.15672572, -0.35441692],
 [ 0.69133623, -0.13072716],
 [ 0.30166293,  1.05769685],
 [-0.03121024, -0.36680337],
 [ 0.24468684, -0.13949864],
 [ 0.77177038,  0.20591734],
 [-0.14727578, -0.49224378],
 [ 0.89374386,  0.19242046],
 [-0.17158696, -0.38913822],
 [-0.88820128,  0.26698651],
 [-0.22046594, -0.4320584 ],
 [ 0.38173298,  1.00707193],
 [-0.7009914 ,  0.38214912],
 [-0.73316371,  0.36828529],
 [-0.5704898 ,  0.25681803],
 [ 0.11323205,  0.65773631],
 [ 0.90668187,  0.19946893],
 [-0.2530694 , -0.50313929],
 [-0.93432507,  0.26490167],
 [ 0.87200298,  0.15864359],
 [ 0.26951735,  0.98982208],
 [ 0.00190079, -0.39316829],
 [ 0.08328001, -0.17026781],
 [ 0.94134428,  0.15982886],
 [-0.12712544, -0.38748441],
 [-0.38690841, -0.53663255],
 [-0.17486367, -0.50418058],
 [ 0.19363302, -0.36918903],
 [-0.36682692, -0.52585768],
 [-0.13137183, -0.37031337],
 [-0.80370545,  0.34505265],
 [ 0.66865293,  0.10271856],
 [-0.2920301 , -0.49137884],
 [ 0.40654496, -0.12848371],
 [-0.25579969, -0.42174571],
 [-0.71229089,  0.3821404 ],
 [-0.03629649, -0.42102783],
 [-0.80931678,  0.37575319],
 [ 1.03983806,  0.10360788],
 [-0.08074347, -0.54354588],
 [-0.25795541, -0.43661242],
 [ 0.05576172, -0.37762175],
```

```
       [-0.63326199,  0.47582315],
       [-0.08702181, -0.53985418],
       [-0.35850425, -0.53209181],
       [ 0.70922636, -0.01000493],
       [-0.77310781,  0.37297937],
       [-0.59725764,  0.46102883],
       [-0.14773805, -0.36158498],
       [ 0.56966999, -0.01330883],
       [ 0.68424026,  0.09443224],
       [-0.47121715,  0.57057377],
       [-0.28327865, -0.50005278],
       [-0.32721012, -0.50450635],
       [-0.08179176, -0.36726268],
       [-0.59316511,  0.49405509],
       [-0.25153947, -0.48766431],
       [-0.8788213 ,  0.27719382],
       [-0.14809461, -0.63563678],
       [-0.25469626, -0.497448  ],
       [ 0.83762707,  0.1150745 ],
       [-0.10650819, -0.40057984],
       [-0.19884895, -0.46223222],
       [ 1.02358416,  0.05598585],
       [-0.16826454, -0.35682576],
       [-0.49441659,  0.57858832],
       [-0.74128377,  0.3517146 ],
       [-0.21490447, -0.71525618],
       [ 0.26127303, -0.40031193],
       [-0.23975655, -0.47321503],
       [-0.18483861, -0.4454762 ],
       [ 0.14439287, -0.4547836 ],
       [ 0.29677479,  1.00489791],
       [-0.66428874,  0.43257768],
       [-0.08249778, -0.38468823],
       [-0.14012566, -0.40383759],
       [-0.25156355,  0.62408792],
       [ 0.20641428,  0.99096865],
       [-0.08351051, -0.33645949],
       [-0.16785057, -0.69981446],
       [ 0.83747295,  0.14523762],
       [ 1.03933157,  0.29595457],
       [-0.2496717 , -0.5849888 ],
       [-0.25333013, -0.44098314],
       [ 0.801883  ,  0.14942923],
       [ 0.97604494,  0.18866931],
       [-0.27176885, -0.43117751],
```

```
[-0.76954875,  0.38330075],
[ 0.89997024, -0.05208366],
[ 0.08661972,  0.92021675],
[ 0.80555159,  0.1680903 ],
[-0.12288882,  0.71289617],
[-0.6557288 ,  0.42744003],
[-0.37212588,  0.62258391],
[ 0.6183415 , -0.14845285],
[-0.1579606 , -0.41669491],
[ 0.79424775, -0.10816758],
[-0.26497277, -0.45423412],
[ 0.69625181,  0.11059579],
[-0.76758338,  0.35077322],
[-0.33058104,  0.57816422],
[ 0.90997533,  0.19102861],
[ 0.21534599,  0.99049656],
[-0.63087125,  0.48060299],
[ 0.98804415,  0.30064055],
[-0.20202485, -0.42794201],
[-0.1404868 , -0.36582407],
[-0.27134433, -0.49572141],
[-0.67002997,  0.22076496],
[-0.16163726, -0.62289309],
[-0.56186612,  0.46691356],
[-0.12862635, -0.56504381],
[-0.26096797, -0.44002333],
[-0.25734822, -0.66957447],
[ 0.92717704,  0.19158558],
[-0.81805936,  0.11660116],
[ 0.91401722,  0.18041303],
[-0.65465385,  0.48295743],
[ 0.41717656,  0.85663035]])
```

# (a) Logistic Regression

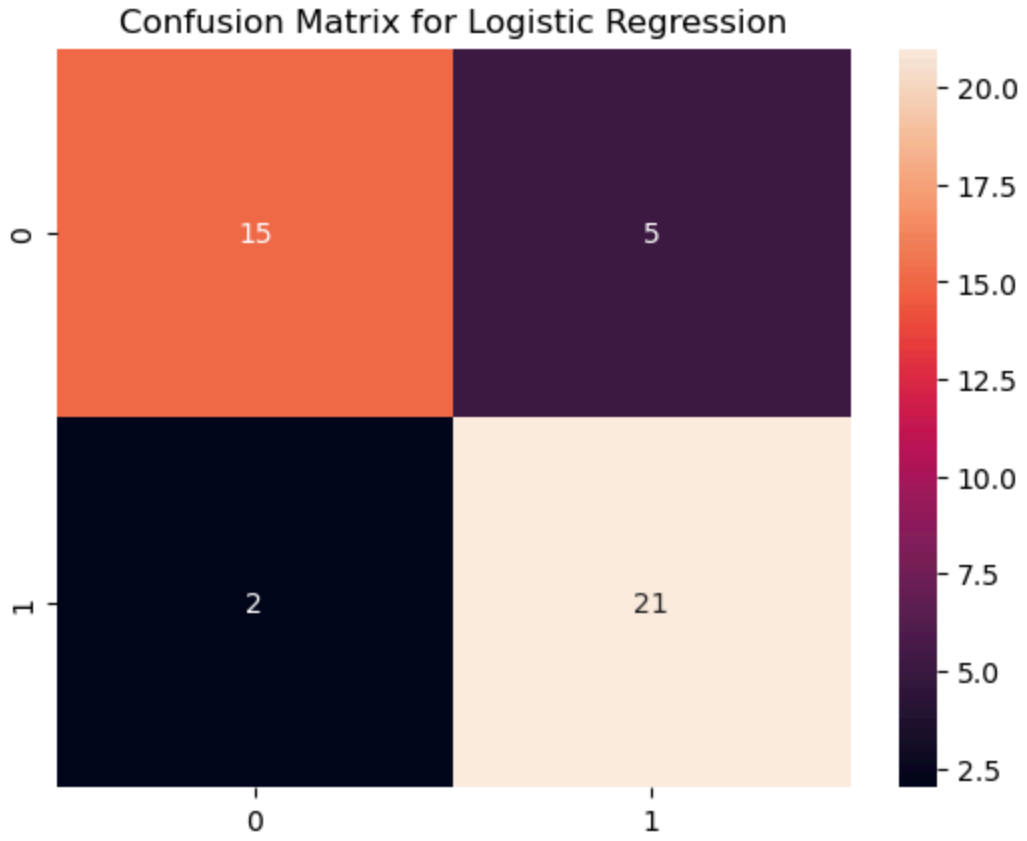
In [69]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
```

In [70]:
```
X_train, X_test, y_train, y_test = train_test_split(pc_X, y, test_size = 0.2, random_state = 42)
```

In [71]:
```
LR = LogisticRegression()
LR.fit(X_train, y_train)
lr_y_pred = LR.predict(X_test)
lr_cfm = confusion_matrix(y_test, lr_y_pred)
print(f"Accuracy of Logistic Regression is: {accuracy_score(y_test, lr_y_pred)}")
```

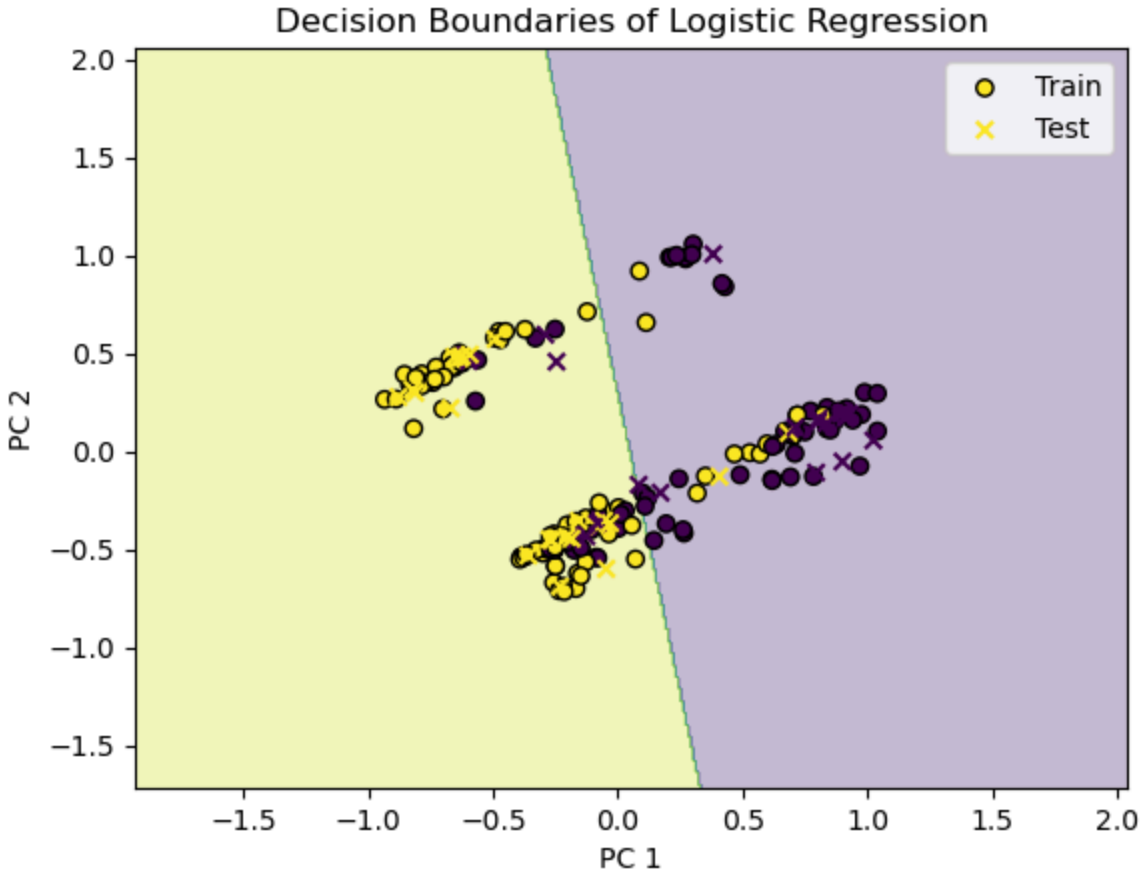
```
Accuracy of Logistic Regression is: 0.8372093023255814
```

In [72]:
```
sns.heatmap(lr_cfm, annot=True)
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```




In [73]:
```
x_min, x_max = pc_X[:, 0].min() -1, pc_X[:, 0].max() +1
y_min, y_max = pc_X[:, 1].min() -1, pc_X[:, 1].max() +1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 400), np.linspace(y_min, y_max, 400))
Z = LR.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)
```

```
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X_train[:, 0], X_train[:, 1], c=y_train, edgecolors='k', label='Train')
plt.scatter(X_test[:, 0], X_test[:, 1], c=y_test, marker='x', label='Test')
plt.xlabel("PC 1")
plt.ylabel("PC 2")
plt.title("Decision Boundaries of Logistic Regression")
plt.legend()
plt.show()
```



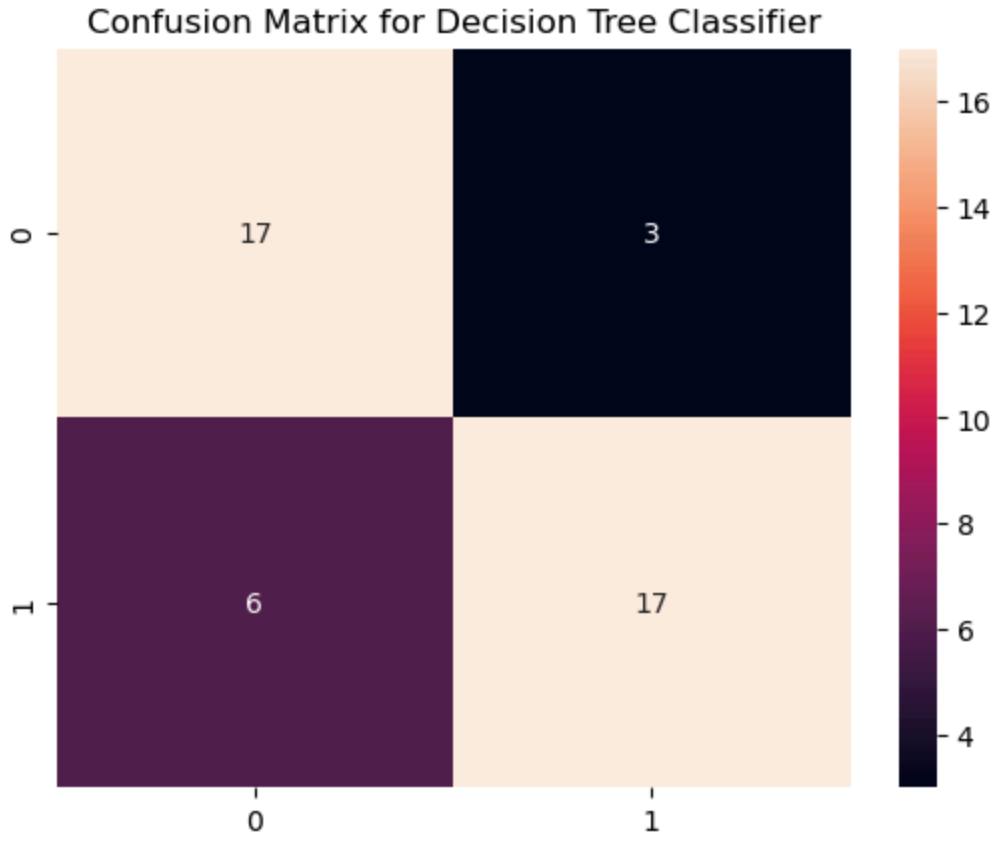

# (b) Decision tree

In [74]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

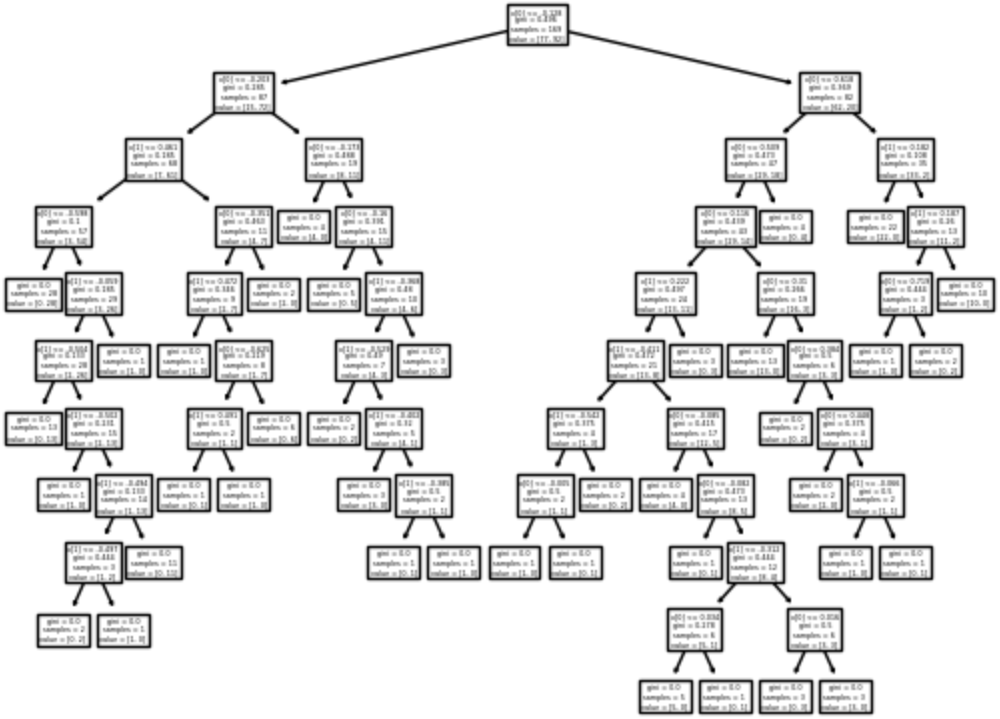
In [75]:
```
DT = DecisionTreeClassifier()
DT.fit(X_train, y_train)
dt_y_pred = DT.predict(X_test)
dt_cfm = confusion_matrix(y_test, dt_y_pred)
print(f"Accuracy of Decision Tree Classifier is: {accuracy_score(y_test, dt_y_pred)}")
```

Accuracy of Decision Tree Classifier is: 0.7906976744186046

In [76]:
```
sns.heatmap(dt_cfm, annot=True)
plt.title('Confusion Matrix for Decision Tree Classifier')
plt.show()
```




In [77]:
```
plot_tree(DT)
plt.show()
```



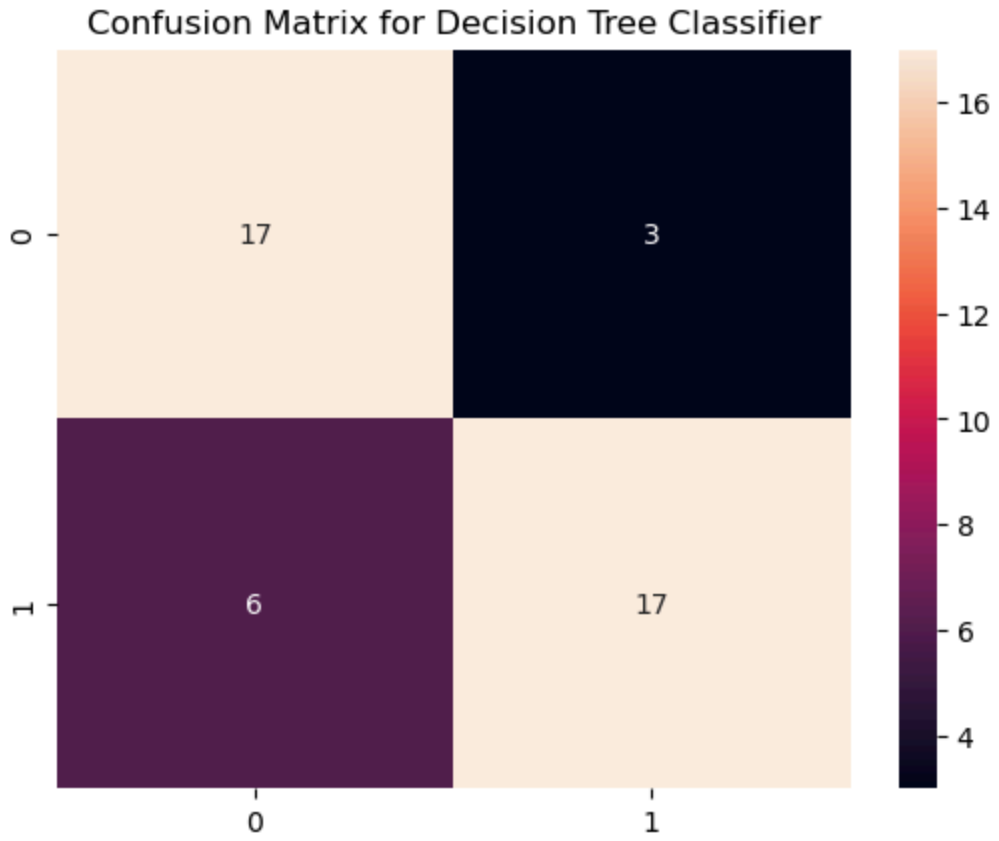
# (c) Bagging

In [78]:
```
from sklearn.ensemble import BaggingClassifier
BC = BaggingClassifier(n_estimators = 100, random_state = 0)
BC.fit(X_train,y_train)
bc_y_pred = BC.predict(X_test)

dt_cfm = confusion_matrix(y_test, bc_y_pred)
print(f"Accuracy of Decision Tree Classifier is: {accuracy_score(y_test, bc_y_pred)}")

sns.heatmap(dt_cfm, annot=True)
plt.title('Confusion Matrix for Decision Tree Classifier')
plt.show()
```

Accuracy of Decision Tree Classifier is: 0.7906976744186046




# Hyperparameter Tuning on Bagging Classifier

In [79]:
```
from sklearn.model_selection import GridSearchCV
bc_param_grid = {
'n_estimators':[ 10, 50, 80 ,100, 150, 200, 250, 300]
}
bc_base_model = BaggingClassifier(random_state=0)

bc_gs = GridSearchCV(estimator = bc_base_model, param_grid = bc_param_grid,cv=5, n_jobs = -1)
bc_gs.fit(X_train, y_train)

print(f'Best parameters for "Bagging Classifier": {bc_gs.best_params_}')
print(f'Best Accuracy: {bc_gs.best_score_}')
```

```
Best parameters for "Bagging Classifier": {'n_estimators': 50}
Best Accuracy: 0.7392156862745097
```

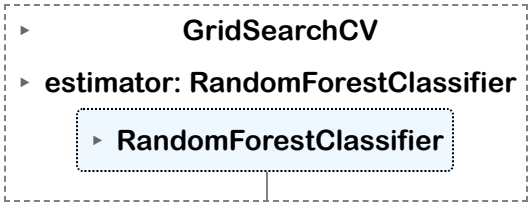
# (d) RandomForest with hyperparamter tuning

In [80]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [81]:
```
rfc_base_model = RandomForestClassifier(max_features='sqrt')
rfc_param_grid = {
'n_estimators':[ 10, 50, 80 ,100, 150, 200, 250, 300],
'max_depth': [1,2,3,4,10,15,20]
}

rfc_gs = GridSearchCV(estimator = rfc_base_model, param_grid = rfc_param_grid,cv=5, n_jobs = -1)
rfc_gs.fit(X_train, y_train)
```

Out[81]:




In [82]:
```
print(f'Best parameters for "Random Forest Classifier": {rfc_gs.best_params_}')
print(f'Best Accuracy: {rfc_gs.best_score_}')
```

```
Best parameters for "Random Forest Classifier": {'max_depth': 2, 'n_estimators': 50}
Best Accuracy: 0.7811051693404634
```

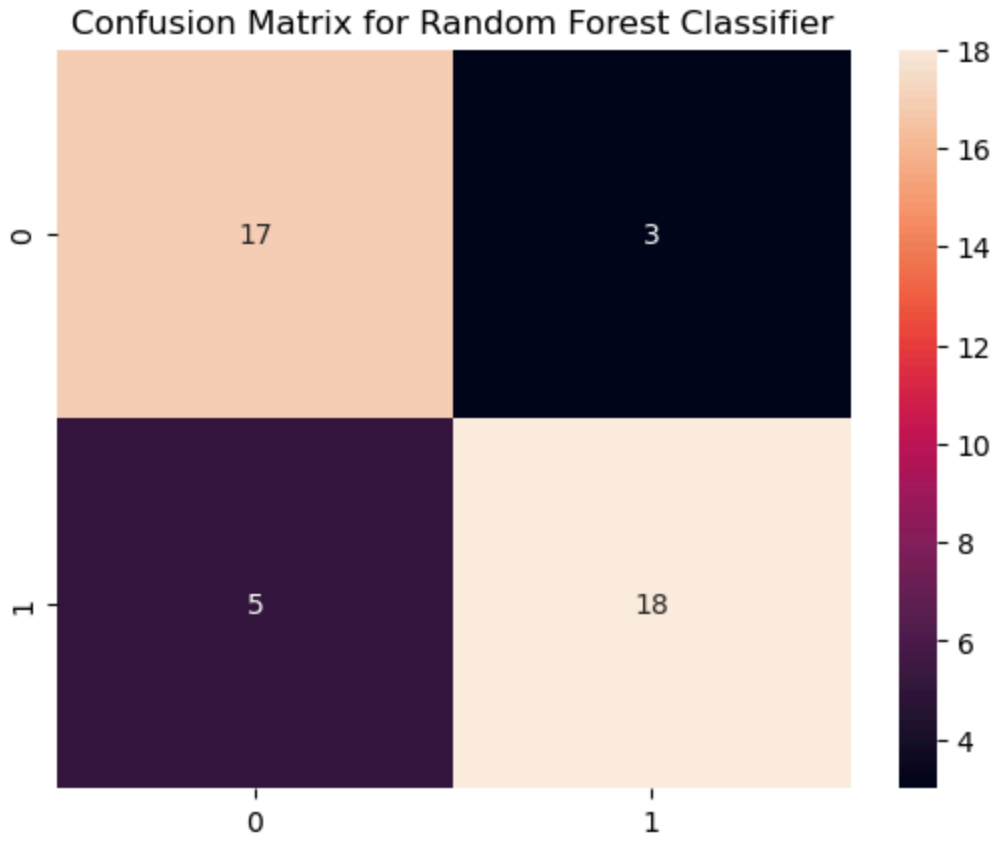
In [83]:
```
RFC =  RandomForestClassifier(max_depth= 2, n_estimators= 50, max_features='sqrt')

RFC.fit(X_train,y_train)
rfc_y_pred = RFC.predict(X_test)

rfc_cfm = confusion_matrix(y_test, rfc_y_pred)
print(f"Accuracy of RandomForest is: {accuracy_score(y_test, rfc_y_pred)}")

sns.heatmap(rfc_cfm, annot=True)
plt.title('Confusion Matrix for Random Forest Classifier')
plt.show()
```

```
Accuracy of RandomForest is: 0.813953488372093
```




# (e) Adaboost with hyperparameter tuning

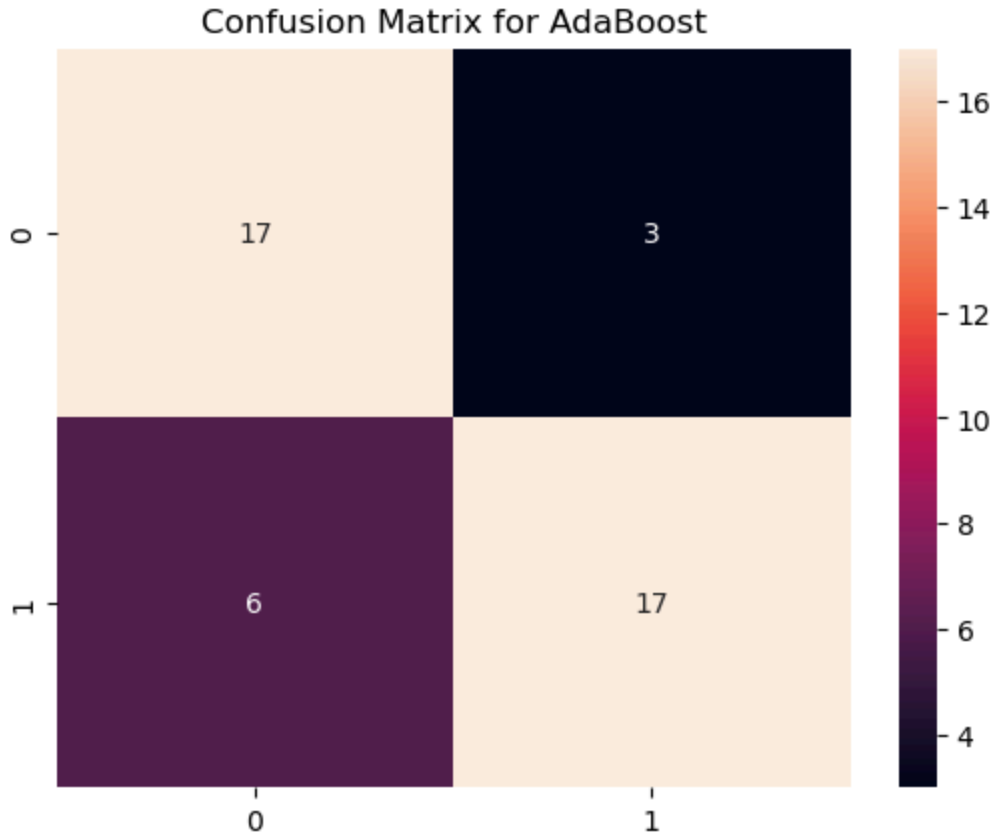
In [97]:
```
from sklearn.ensemble import AdaBoostClassifier
```

In [98]:
```
ADB = AdaBoostClassifier(n_estimators = 50, learning_rate=0.01, random_state=0)
ADB.fit(X_train, y_train)
adb_y_pred = ADB.predict(X_test)

adb_cfm = confusion_matrix(y_test, adb_y_pred)
print(f"Accuracy of AdaBoost is: {accuracy_score(y_test, adb_y_pred)}")

sns.heatmap(adb_cfm, annot=True)
plt.title('Confusion Matrix for AdaBoost')
plt.show()
```

Accuracy of AdaBoost is: 0.7906976744186046




# (f) Gradientboosting with hyperparameter tuning

In [92]:
```
from sklearn.ensemble import GradientBoostingClassifier
gbr_base_model =  GradientBoostingClassifier(random_state=0)

gbr_param_grid = {
'n_estimators':[ 10, 50, 80 ,100, 150, 200, 250, 300],
'max_depth': [1,2,3,4,10,15,20],
'learning_rate': [0.01,0.02,0.1,0.05,0.5,0.07,0.9]
}

gbr_gs = GridSearchCV(estimator = gbr_base_model, param_grid = gbr_param_grid,cv=5, n_jobs = -1)
gbr_gs.fit(X_train, y_train)
```

Out[92]:

GridSearchCV

estimator: GradientBoostingClassifier

GradientBoostingClassifier

In [93]:
```
print(f'Best parameters for "Gradient Boosting Classifier": {gbr_gs.best_params_}')
print(f'Best Accuracy: {gbr_gs.best_score_}')
```

```
Best parameters for "Gradient Boosting Classifier": {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 300}
Best Accuracy: 0.7752228163992869
```

In [94]:
```
GBR = GradientBoostingClassifier(learning_rate=0.01, n_estimators=300, max_depth = 1, random_state=0)
GBR.fit(X_train, y_train)
```

Out[94]:

GradientBoostingClassifier

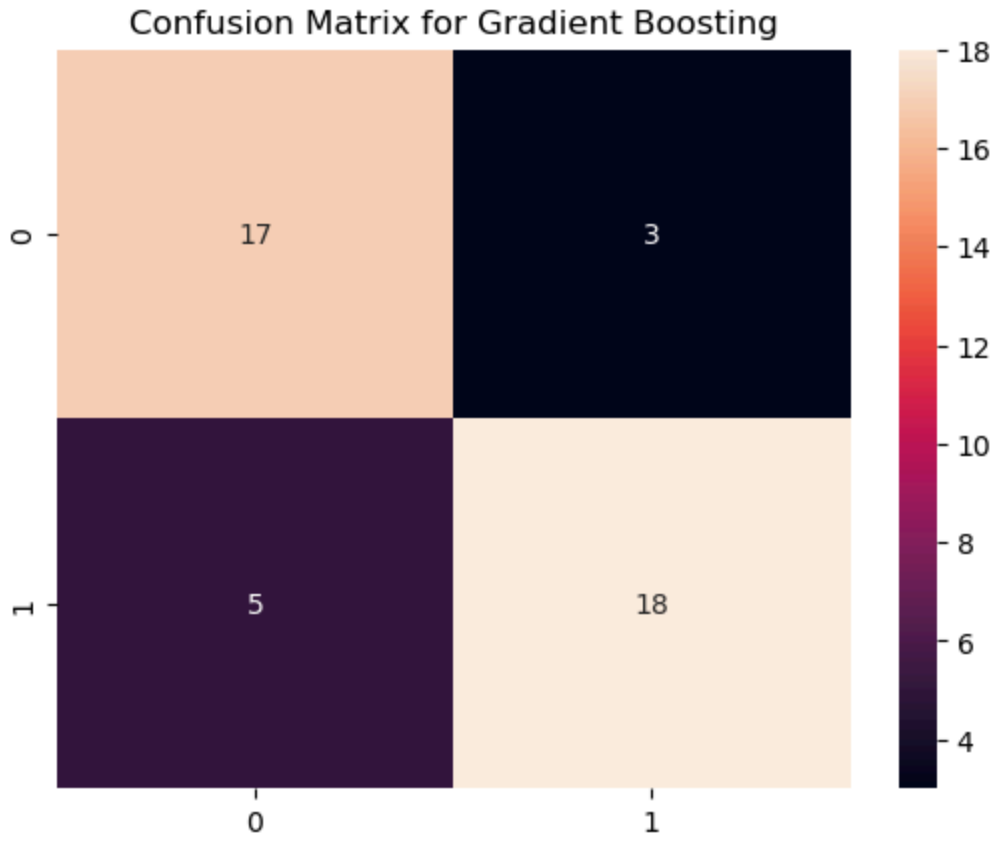
```
GradientBoostingClassifier(learning_rate=0.01, max_depth=1, n_estimators=300,
                           random_state=0)
```

In [95]:
```
gbr_y_pred = GBR.predict(X_test)

gbr_cfm = confusion_matrix(y_test, gbr_y_pred)
print(f"Accuracy of Gradient Boosting is: {accuracy_score(y_test, gbr_y_pred)}")

sns.heatmap(gbr_cfm, annot=True)
plt.title('Confusion Matrix for Gradient Boosting')
plt.show()
```

```
Accuracy of Gradient Boosting is: 0.813953488372093
```


Confusion Matrix for Gradient Boosting
17
3
5
18
0
1
0
1
18
16
14
12
10
8
6
4

In [ ]: