In [1]:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:

```
dataset = pd.read_csv('C:/Users/ISHITA SWAMI/Desktop/Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

In [3]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =
```

In [4]:

```
print(X_train)
```

```
[[    44  39000]
 [    32 120000]
 [    38  50000]
 [    32 135000]
 [    52  21000]
 [    53 104000]
 [    39  42000]
 [    38  61000]
 [    36  50000]
 [    36  63000]
 [    35  25000]
 [    35  50000]
 [    42  73000]
 [    47  49000]
 [    59  29000]
 [    49  65000]
 [    45 131000]
 [    31  89000]
 [    46  82000]
```

In [5]:

```
print(y_train)
```

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

In [6]:

```
print(X_test)
```

```
[[    30  87000]
 [    38  50000]
 [    35  75000]
 [    30  79000]
 [    35  50000]
 [    27  20000]
 [    31  15000]
 [    36 144000]
 [    18  68000]
 [    47  43000]
 [    30  49000]
 [    28  55000]
 [    37  55000]
 [    39  77000]
 [    20  86000]
 [    32 117000]
 [    37  77000]
 [    19  85000]
 [    55 130000]
 [    35  22000]
 [    35  47000]
 [    47 144000]
 [    41  51000]
 [    47 105000]
 [    23  28000]
 [    49 141000]
 [    28  87000]
 [    29  80000]
 [    37  62000]
 [    32  86000]
 [    21  88000]
 [    37  79000]
 [    57  60000]
 [    37  53000]
 [    24  58000]
 [    18  52000]
 [    22  81000]
 [    34  43000]
 [    31  34000]
 [    49  36000]
 [    27  88000]
 [    41  52000]
 [    27  84000]
 [    35  20000]
 [    43 112000]
 [    27  58000]
 [    37  80000]
 [    52  90000]
 [    26  30000]
 [    49  86000]
 [    57 122000]
 [    34  25000]
 [    35  57000]
 [    34 115000]
 [    59  88000]
 [    45  32000]
 [    29  83000]
```

```
 [    26  80000]
 [    49  28000]
 [    23  20000]
 [    32  18000]
 [    60  42000]
 [    19  76000]
 [    36  99000]
 [    19  26000]
 [    60  83000]
 [    24  89000]
 [    27  58000]
 [    40  47000]
 [    42  70000]
 [    32 150000]
 [    35  77000]
 [    22  63000]
 [    45  22000]
 [    27  89000]
 [    18  82000]
 [    42  79000]
 [    40  60000]
 [    53  34000]
 [    47 107000]
 [    58 144000]
 [    59  83000]
 [    24  55000]
 [    26  35000]
 [    58  38000]
 [    42  80000]
 [    40  75000]
 [    59 130000]
 [    46  41000]
 [    41  60000]
 [    42  64000]
 [    37 146000]
 [    23  48000]
 [    25  33000]
 [    24  84000]
 [    27  96000]
 [    23  63000]
 [    48  33000]
 [    48  90000]
 [    42 104000]]
```

In [7]:

```
print(y_test)
```

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

In [8]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

In [9]:

```
print(X_train)
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]
 [-0.30964085 -1.29261101]
 [-0.30964085 -0.5677824 ]
 [ 0.38358493  0.09905991]
 [ 0.8787462  -0.59677555]
 [ 2.06713324 -1.17663843]
 [ 1.07681071 -0.13288524]
 [ 0.68068169  1.78066227]
 [-0.70576986  0.56295021]
 [ 0.77971394  0.35999821]
```

In [10]:

```
print(X_test)
```

```
[[-0.80480212  0.50496393]
 [-0.01254409 -0.5677824 ]
 [-0.30964085  0.1570462 ]
 [-0.80480212  0.27301877]
 [-0.30964085 -0.5677824 ]
 [-1.10189888 -1.43757673]
 [-0.70576986 -1.58254245]
 [-0.21060859  2.15757314]
 [-1.99318916 -0.04590581]
 [ 0.8787462  -0.77073441]
 [-0.80480212 -0.59677555]
 [-1.00286662 -0.42281668]
 [-0.11157634 -0.42281668]
 [ 0.08648817  0.21503249]
 [-1.79512465  0.47597078]
 [-0.60673761  1.37475825]
 [-0.11157634  0.21503249]
 [-1.89415691  0.44697764]
 [ 1.67100423  1.75166912]
 [-0.30964085 -1.37959044]
 [-0.30964085 -0.65476184]
 [ 0.8787462   2.15757314]
 [ 0.28455268 -0.53878926]
 [ 0.8787462   1.02684052]
 [-1.49802789 -1.20563157]
 [ 1.07681071  2.07059371]
 [-1.00286662  0.50496393]
 [-0.90383437  0.30201192]
 [-0.11157634 -0.21986468]
 [-0.60673761  0.47597078]
 [-1.6960924   0.53395707]
 [-0.11157634  0.27301877]
 [ 1.86906873 -0.27785096]
 [-0.11157634 -0.48080297]
 [-1.39899564 -0.33583725]
 [-1.99318916 -0.50979612]
 [-1.59706014  0.33100506]
 [-0.4086731  -0.77073441]
 [-0.70576986 -1.03167271]
 [ 1.07681071 -0.97368642]
 [-1.10189888  0.53395707]
 [ 0.28455268 -0.50979612]
 [-1.10189888  0.41798449]
 [-0.30964085 -1.43757673]
 [ 0.48261718  1.22979253]
 [-1.10189888 -0.33583725]
 [-0.11157634  0.30201192]
 [ 1.37390747  0.59194336]
 [-1.20093113 -1.14764529]
 [ 1.07681071  0.47597078]
 [ 1.86906873  1.51972397]
 [-0.4086731  -1.29261101]
 [-0.30964085 -0.3648304 ]
 [-0.4086731   1.31677196]
 [ 2.06713324  0.53395707]
 [ 0.68068169 -1.089659  ]
 [-0.90383437  0.38899135]
```

```
 [-1.20093113  0.30201192]
 [ 1.07681071 -1.20563157]
 [-1.49802789 -1.43757673]
 [-0.60673761 -1.49556302]
 [ 2.1661655  -0.79972756]
 [-1.89415691  0.18603934]
 [-0.21060859  0.85288166]
 [-1.89415691 -1.26361786]
 [ 2.1661655   0.38899135]
 [-1.39899564  0.56295021]
 [-1.10189888 -0.33583725]
 [ 0.18552042 -0.65476184]
 [ 0.38358493  0.01208048]
 [-0.60673761  2.331532  ]
 [-0.30964085  0.21503249]
 [-1.59706014 -0.19087153]
 [ 0.68068169 -1.37959044]
 [-1.10189888  0.56295021]
 [-1.99318916  0.35999821]
 [ 0.38358493  0.27301877]
 [ 0.18552042 -0.27785096]
 [ 1.47293972 -1.03167271]
 [ 0.8787462   1.08482681]
 [ 1.96810099  2.15757314]
 [ 2.06713324  0.38899135]
 [-1.39899564 -0.42281668]
 [-1.20093113 -1.00267957]
 [ 1.96810099 -0.91570013]
 [ 0.38358493  0.30201192]
 [ 0.18552042  0.1570462 ]
 [ 2.06713324  1.75166912]
 [ 0.77971394 -0.8287207 ]
 [ 0.28455268 -0.27785096]
 [ 0.38358493 -0.16187839]
 [-0.11157634  2.21555943]
 [-1.49802789 -0.62576869]
 [-1.29996338 -1.06066585]
 [-1.39899564  0.41798449]
 [-1.10189888  0.76590222]
 [-1.49802789 -0.19087153]
 [ 0.97777845 -1.06066585]
 [ 0.97777845  0.59194336]
 [ 0.38358493  0.99784738]]
```

In [11]:

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[11]:

```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

In [12]:

```
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

In [13]:

```
y_pred = classifier.predict(X_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

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

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

In [14]:

```
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

```
[[62  6]
 [ 3 29]]
```

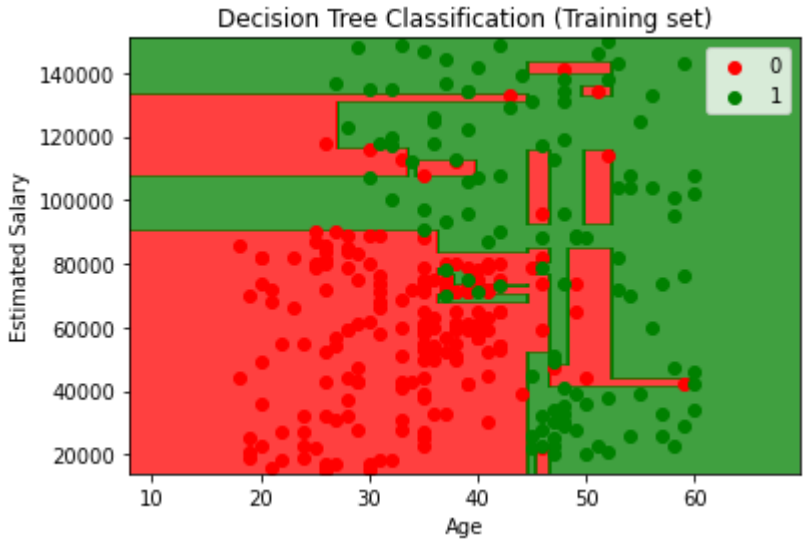
Out[14]:

```
0.91
```

In [17]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_train), y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 1
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T))
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'gre
plt.title('Decision Tree Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D ar
ray with a single row if you intend to specify the same RGB or RGBA value fo
r all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D ar
ray with a single row if you intend to specify the same RGB or RGBA value fo
r all points.
```



In [18]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = sc.inverse_transform(X_test), y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 10, stop = X_set[:, 0].max() + 1
                     np.arange(start = X_set[:, 1].min() - 1000, stop = X_set[:, 1].max() +
plt.contourf(X1, X2, classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T))
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1], c = ListedColormap(('red', 'gre
plt.title('Decision Tree Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D ar
ray with a single row if you intend to specify the same RGB or RGBA value fo
r all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D ar
ray with a single row if you intend to specify the same RGB or RGBA value fo
r all points.
```

In [ ]:

```

```