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 =
print(X_train)
print(y_train)
print(X_test)
print(y_test)
```

```
[    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]
```

In [4]:

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

```
[ 2.06713524  2.12857999]
 [ 1.86906873 -1.26361786]
 [ 1.37390747 -0.91570013]
 [ 0.8787462   1.25878567]
 [ 1.47293972  2.12857999]
 [-0.30964085 -1.23462472]
 [ 1.96810099  0.91086794]
 [ 0.68068169 -0.71274813]
 [-1.49802789  0.35999821]
 [ 0.77971394 -1.3505973 ]
 [ 0.38358493 -0.13288524]]
[[-1.00286662  0.41798449]
 [-0.01254409 -0.30684411]
 [-1.20093113  0.41798449]
 [-0.90383437 -1.20563157]
 [-0.11157634  0.04107362]
 [-1.59706014 -0.42281668]
 [ 0.97777845 -1.00267957]
 [ 1.07681071 -1.20563157]
 [-0.01254409 -0.13288524]]
```

In [5]:

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, y_train)
```

Out[5]:

```
KNeighborsClassifier()
```

In [6]:

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

```
[0]
```

In [7]:

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

In [8]:

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

```
[[64  4]
 [ 3 29]]
```

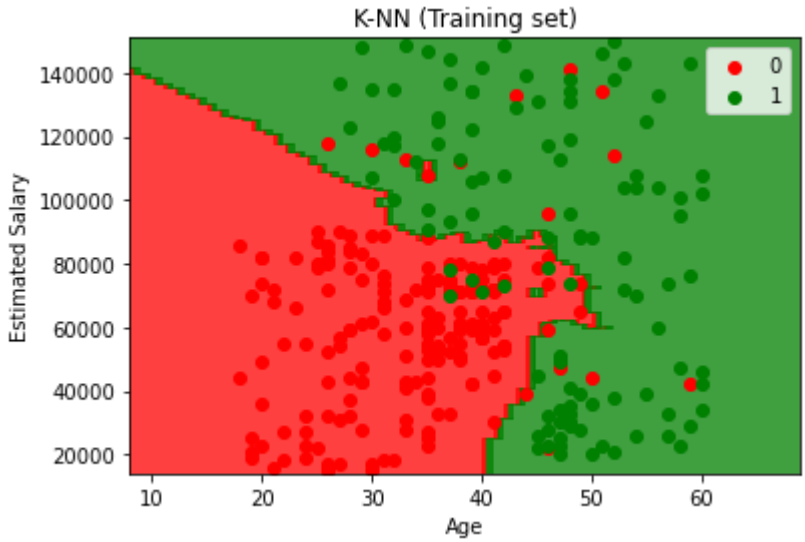
Out[8]:

```
0.93
```

In [9]:

```
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('K-NN (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

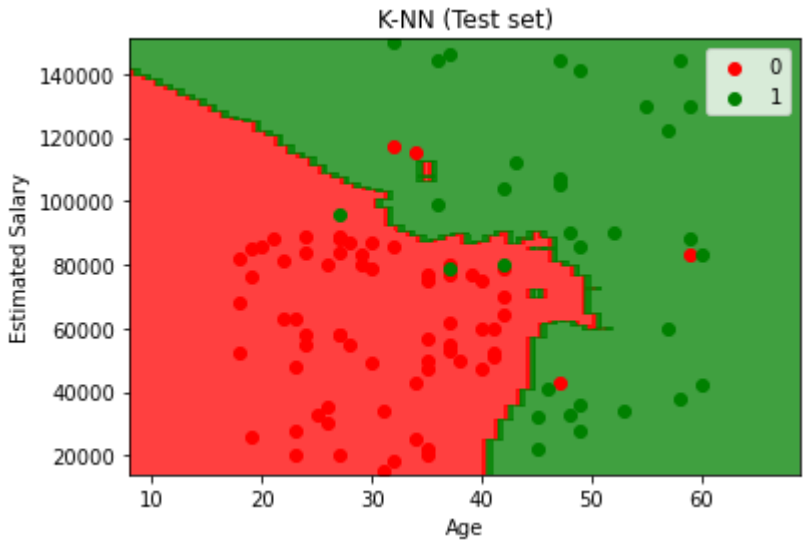
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should
be avoided as value-mapping will have precedence in case its length matches
with *x* & *y*.  Please use the *color* keyword-argument or provide a 2-D ar
ray with a single row if you intend to specify the same RGB or RGBA 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 [14]:

```
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('K-NN (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 [ ]: