Simple Linear Regression

```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\anaconda\lib\site-packages
(1.21.5)
Note: you may need to restart the kernel to use updated packages.
```

```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\anaconda\lib\site-packages
(1.4.2)Note: you may need to restart the kernel to use updated
packages.

Requirement already satisfied: python-dateutil>=2.8.1 in c:\anaconda\
lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\anaconda\lib\site-
packages (from pandas) (2021.3)
Requirement already satisfied: numpy>=1.18.5 in c:\anaconda\lib\site-
packages (from pandas) (1.21.5)
Requirement already satisfied: six>=1.5 in c:\anaconda\lib\site-
packages (from python-dateutil>=2.8.1->pandas) (1.16.0)
```

```
pip install -U scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\anaconda\lib\site-
packages (1.0.2)
Collecting scikit-learn
  Downloading scikit_learn-1.3.0-cp39-cp39-win_amd64.whl (9.3 MB)
Requirement already satisfied: numpy>=1.17.3 in c:\anaconda\lib\site-
packages (from scikit-learn) (1.21.5)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\anaconda\
lib\site-packages (from scikit-learn) (2.2.0)
Collecting joblib>=1.1.1
  Downloading joblib-1.3.1-py3-none-any.whl (301 kB)
Requirement already satisfied: scipy>=1.5.0 in c:\anaconda\lib\site-
packages (from scikit-learn) (1.7.3)
Installing collected packages: joblib, scikit-learn
  Attempting uninstall: joblib
    Found existing installation: joblib 1.1.0
    Uninstalling joblib-1.1.0:
      Successfully uninstalled joblib-1.1.0
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.0.2
    Uninstalling scikit-learn-1.0.2:
      Successfully uninstalled scikit-learn-1.0.2
Successfully installed joblib-1.3.1 scikit-learn-1.3.0
Note: you may need to restart the kernel to use updated packages.
```

```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\anaconda\lib\site-packages (3.5.1)
Requirement already satisfied: cycler>=0.10 in c:\anaconda\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\anaconda\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: packaging>=20.0 in c:\anaconda\lib\site-packages (from matplotlib) (21.3)
Requirement already satisfied: numpy>=1.17 in c:\anaconda\lib\site-packages (from matplotlib) (1.21.5)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\anaconda\lib\site-packages (from matplotlib) (1.3.2)
Requirement already satisfied: pyparsing>=2.2.1 in c:\anaconda\lib\site-packages (from matplotlib) (3.0.4)
Requirement already satisfied: pillow>=6.2.0 in c:\anaconda\lib\site-packages (from matplotlib) (9.0.1)
Requirement already satisfied: python-dateutil>=2.7 in c:\anaconda\lib\site-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.5 in c:\anaconda\lib\site-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

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

```
data = pd.read_csv(r'''C:\Users\Sanika\Downloads\Salary_Data.csv''')
```

```
data
```

```
    YearsExperience    Salary
0               1.1   39343.0
1               1.3   46205.0
2               1.5   37731.0
3               2.0   43525.0
4               2.2   39891.0
5               2.9   56642.0
6               3.0   60150.0
7               3.2   54445.0
8               3.2   64445.0
9               3.7   57189.0
10              3.9   63218.0
11              4.0   55794.0
12              4.0   56957.0
13              4.1   57081.0
14              4.5   61111.0
15              4.9   67938.0
16              5.1   66029.0
```

```
17                5.3   83088.0
18                5.9   81363.0
19                6.0   93940.0
20                6.8   91738.0
21                7.1   98273.0
22                7.9  101302.0
23                8.2  113812.0
24                8.7  109431.0
25                9.0  105582.0
26                9.5  116969.0
27                9.6  112635.0
28               10.3  122391.0
29               10.5  121872.0
```

```
data.head()
```

```
   YearsExperience   Salary
0              1.1  39343.0
1              1.3  46205.0
2              1.5  37731.0
3              2.0  43525.0
4              2.2  39891.0
```

```
data.tail()
```

```
    YearsExperience    Salary
25              9.0  105582.0
26              9.5  116969.0
27              9.6  112635.0
28             10.3  122391.0
29             10.5  121872.0
```

```
data[13:21]
```

```
    YearsExperience   Salary
13              4.1  57081.0
14              4.5  61111.0
15              4.9  67938.0
16              5.1  66029.0
17              5.3  83088.0
18              5.9  81363.0
19              6.0  93940.0
20              6.8  91738.0
```

```
x = data.iloc[:,:-1].values
y = data.iloc[:,1:].values
```

```
x
```

```
array([[ 1.1],
       [ 1.3],
       [ 1.5],
       [ 2. ],
       [ 2.2],
       [ 2.9],
       [ 3. ],
       [ 3.2],
       [ 3.2],
       [ 3.7],
       [ 3.9],
       [ 4. ],
       [ 4. ],
       [ 4.1],
       [ 4.5],
       [ 4.9],
       [ 5.1],
       [ 5.3],
       [ 5.9],
       [ 6. ],
       [ 6.8],
       [ 7.1],
       [ 7.9],
       [ 8.2],
       [ 8.7],
       [ 9. ],
       [ 9.5],
       [ 9.6],
       [10.3],
       [10.5]])
```

```
y
```

```
array([[ 39343.],
       [ 46205.],
       [ 37731.],
       [ 43525.],
       [ 39891.],
       [ 56642.],
       [ 60150.],
       [ 54445.],
       [ 64445.],
       [ 57189.],
       [ 63218.],
       [ 55794.],
       [ 56957.],
       [ 57081.],
       [ 61111.],
       [ 67938.],
       [ 66029.],
       [ 83088.],
       [ 81363.],
       [ 93940.],
       [ 91738.],
       [ 98273.],
       [101302.],
       [113812.],
       [109431.],
       [105582.],
       [116969.],
       [112635.],
       [122391.],
       [121872.]])
```

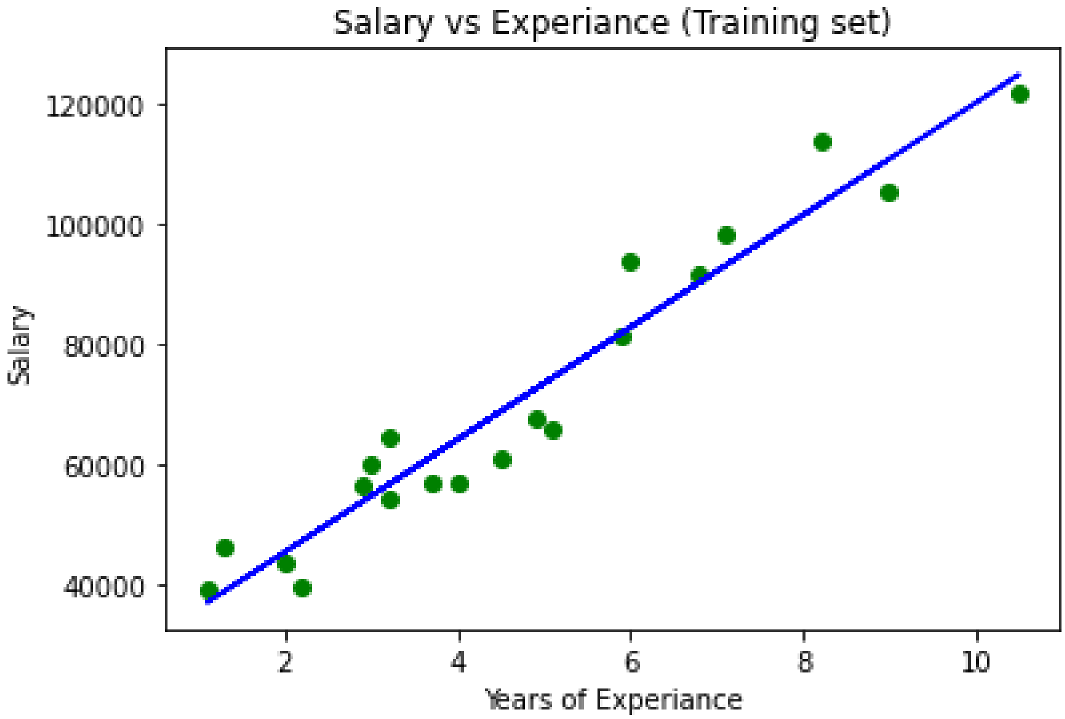
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size =
1/3, random_state = 0)
```

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(x_train, y_train)
```

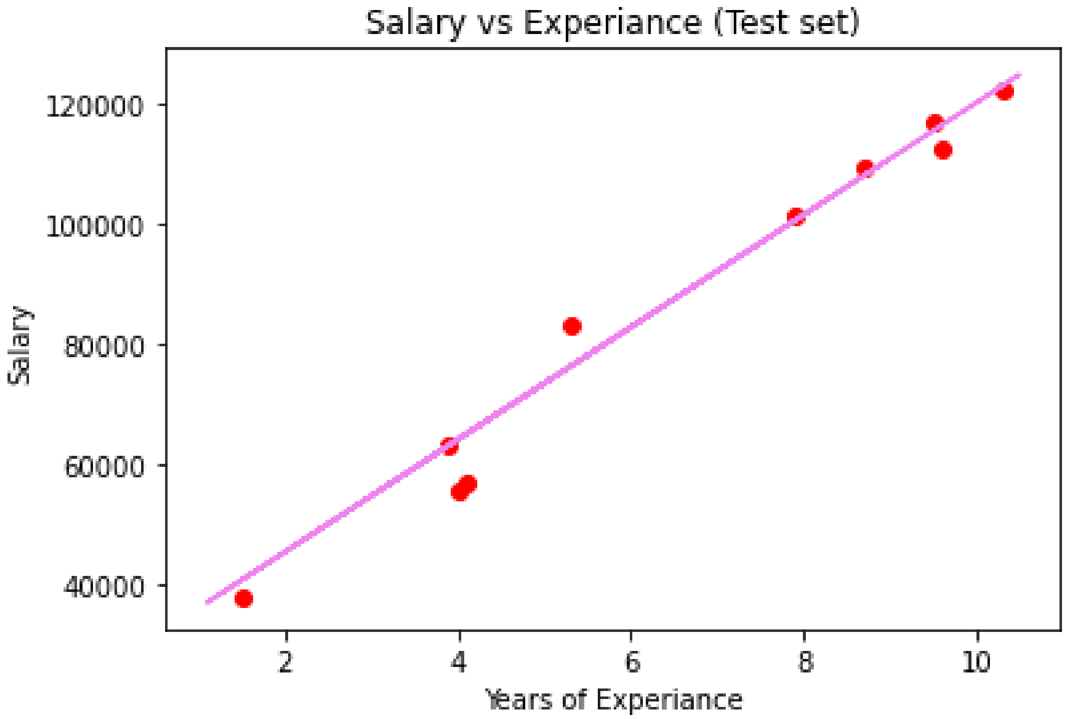
```
LinearRegression()
```

```
y_pred = regressor.predict(x_test)
```

```
plt.scatter(x_train, y_train, color = 'green')
plt.plot(x_train, regressor.predict(x_train), color = 'blue')
plt.title("Salary vs Experiance (Training set)")
plt.xlabel("Years of Experiance")
plt.ylabel("Salary")
plt.show()
```



```
plt.scatter(x_test, y_test, color = 'red')
plt.plot(x_train, regressor.predict(x_train), color = 'violet')
plt.title("Salary vs Experiance (Test set)")
plt.xlabel("Years of Experiance")
plt.ylabel("Salary")
plt.show()
```


Salary vs Experiance (Test set)
Salary
120000
100000
80000
60000
40000
2
4
6
8
10
Years of Experiance

Multiple linear regression

import numpy as np

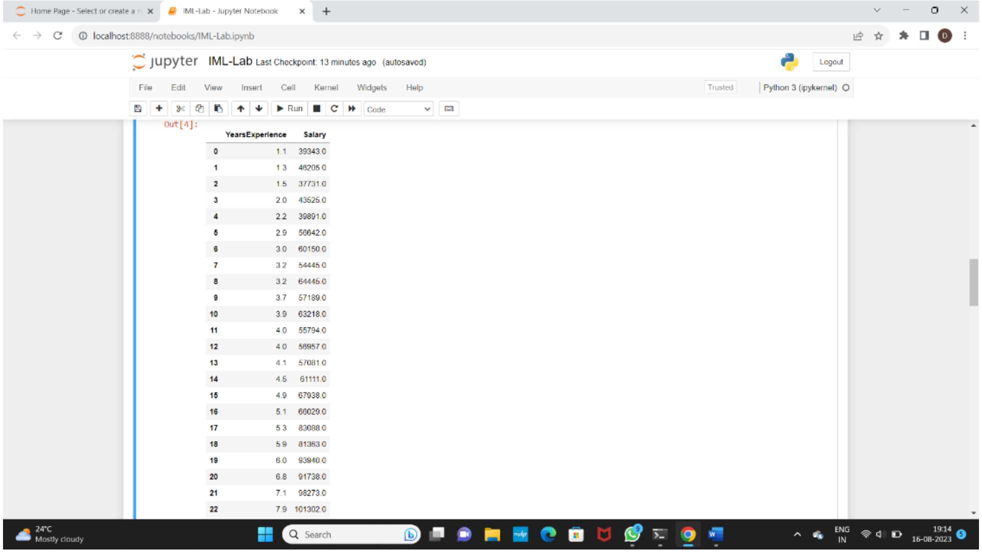
import matplotlib.pyplot as plt

import pandas as pd

dataset = pd.read_csv(r"'C:\Users\dhana\Downloads\Salary_Data.csv"')

dataset

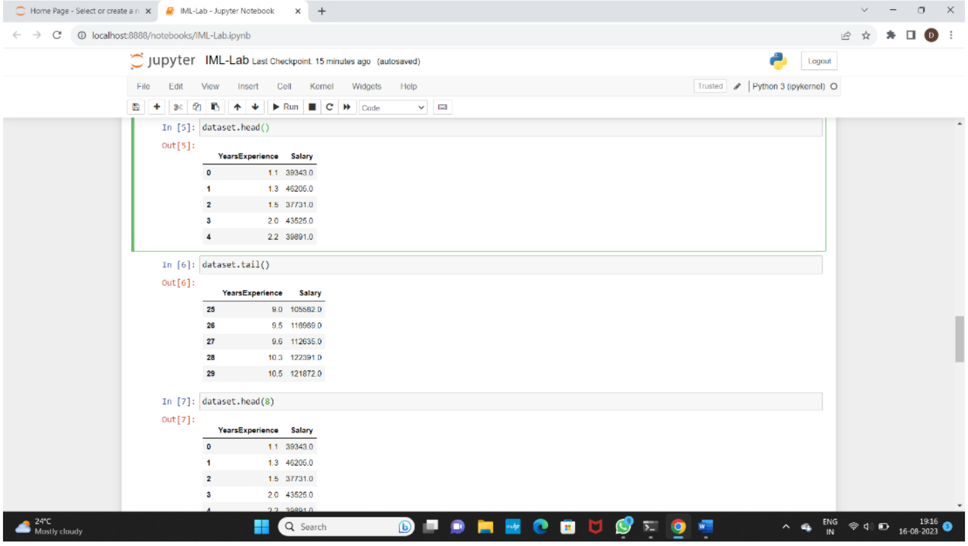
Output:



Out[4]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0
8	3.2	64445.0
9	3.7	57189.0
10	3.9	63218.0
11	4.0	55794.0
12	4.0	56957.0
13	4.1	57081.0
14	4.5	61111.0
15	4.9	67938.0
16	5.1	66029.0
17	5.3	83088.0
18	5.9	81363.0
19	6.0	93940.0
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0

IML-Lab - Jupyter Notebook
localhost:8888/notebooks/IML-Lab.ipynb
jupyter IML-Lab Last Checkpoint: 13 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (ipykernel)
Run
Code
24°C
Mostly cloudy
Search
ENG
IN
19:14
16-08-2023



```
In [5]: dataset.head()
```

Out[5]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

```
In [6]: dataset.tail()
```

Out[6]:

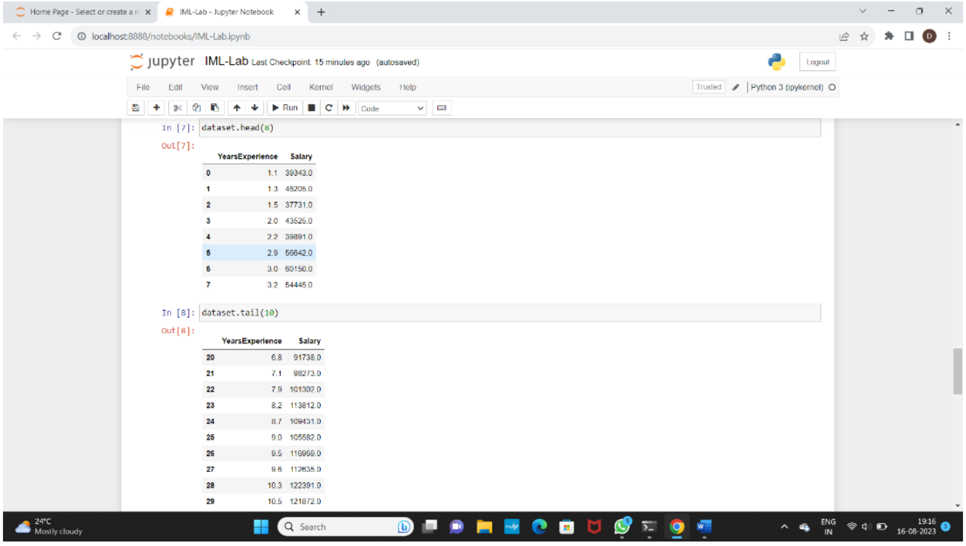
	YearsExperience	Salary
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

```
In [7]: dataset.head(8)
```

Out[7]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

IML-Lab - Jupyter Notebook
localhost:8888/notebooks/IML-Lab.ipynb
jupyter IML-Lab Last Checkpoint: 15 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (ipykernel)
Run
Code
24°C
Mostly cloudy
Search
ENG
IN
19:16
16-08-2023



IML-Lab - Jupyter Notebook
localhost:8888/notebooks/IML-Lab.ipynb
jupyter IML-Lab Last Checkpoint: 15 minutes ago (autosaved)
File Edit View Insert Cell Kernel Widgets Help
Run
Code
Trusted
Python 3 (ipykernel)
Logout

```
In [7]: dataset.head(8)
Out[7]:
```

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0
5	2.9	56642.0
6	3.0	60150.0
7	3.2	54445.0

```
In [8]: dataset.tail(10)
Out[8]:
```

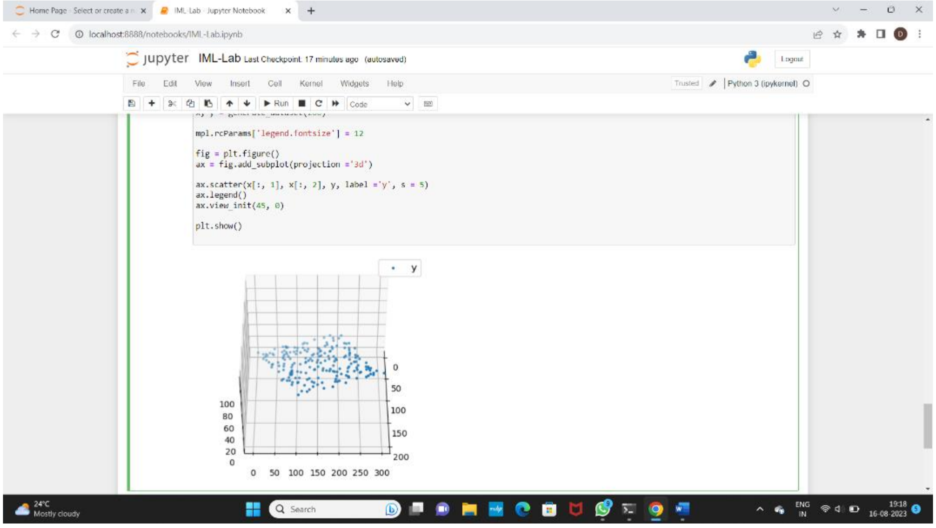
	YearsExperience	Salary
20	6.8	91738.0
21	7.1	98273.0
22	7.9	101302.0
23	8.2	113812.0
24	8.7	109431.0
25	9.0	105582.0
26	9.5	116969.0
27	9.6	112635.0
28	10.3	122391.0
29	10.5	121872.0

24°C
Mostly cloudy
Search
ENG
IN
19:16
16-08-2023

```
import numpy as np

import matplotlib as mpl

from mpl_toolkits.mplot3d import Axes3D

import matplotlib.pyplot as plt


def generate_dataset(n):
    x = []
    y = []
    random_x1 = np.random.rand()
    random_x2 = np.random.rand()
    for i in range(n):
        x1 = i
        x2 = i/2 + np.random.rand()*n
        x.append([1, x1, x2])
        y.append(random_x1 * x1 + random_x2 * x2 + 1)
    return np.array(x), np.array(y)
x, y = generate_dataset(200)

mpl.rcParams['legend.fontsize'] = 12
```

```
fig = plt.figure()

ax = fig.add_subplot(projection ='3d')


ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)

ax.legend()

ax.view_init(45, 0)


plt.show()
```

Output:



Home Page - Select or create a n
IML-Lab - Jupyter Notebook
localhost:8888/notebooks/IML-Lab.ipynb
jupyter IML-Lab Last Checkpoint: 17 minutes ago (autosaved)
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3 (ipykernel)
Run
Code

```
mpl.rcParams['legend.fontsize'] = 12

fig = plt.figure()
ax = fig.add_subplot(projection ='3d')

ax.scatter(x[:, 1], x[:, 2], y, label ='y', s = 5)
ax.legend()
ax.view_init(45, 0)

plt.show()
```

```
from sklearn.linear_model import LinearRegression

import numpy as np

X = np.array([[1, 2, 3], [2, 3, 4], [3, 4, 5], [4, 5, 6]])

y = np.array([1, 2, 3, 4])

reg = LinearRegression()

reg.fit(X, y)

print(reg.coef_)
```

Output:

```
[0.33333333 0.33333333 0.33333333]
```

Logistic Regression

In [1]:

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

Importing the dataset

In [4]:

```
dataset = pd.read_csv(r'''C:\Users\dhana\OneDrive\Documents\Social_Network_Ads (
x = dataset.iloc[:,[2, 3]].values
y = dataset.iloc[:, -1].values
```

In [5]:

```
dataset.head()
```

Out[5]:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0

In [6]:

```
dataset.tail()
```

Out[6]:

	User ID	Gender	Age	EstimatedSalary	Purchased
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

Splitting the dataset into the Training set and Test set

In [7]:

```
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_st
```

Feature Scaling

In [8]:
```
pip install sklearn
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: sklearn in c:\users\dhana\appdata\roaming\python\p
ython310\site-packages (0.0.post7)
Note: you may need to restart the kernel to use updated packages.
```

In [9]:
```
pip install StandardScaler
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: StandardScaler in c:\users\dhana\appdata\roaming\p
ython\python310\site-packages (0.5)
Requirement already satisfied: scikit-elm in c:\users\dhana\appdata\roaming\pytho
n\python310\site-packages (from StandardScaler) (0.21a0)
Requirement already satisfied: pandas in c:\programdata\anaconda3\lib\site-packag
es (from StandardScaler) (1.5.3)
Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\lib\site-
packages (from StandardScaler) (1.2.1)
Requirement already satisfied: numpy in c:\programdata\anaconda3\lib\site-package
s (from StandardScaler) (1.23.5)
Requirement already satisfied: dask in c:\programdata\anaconda3\lib\site-packages
(from StandardScaler) (2022.7.0)
Requirement already satisfied: partd>=0.3.10 in c:\programdata\anaconda3\lib\site
-packages (from dask->StandardScaler) (1.2.0)
Requirement already satisfied: packaging>=20.0 in c:\programdata\anaconda3\lib\si
te-packages (from dask->StandardScaler) (22.0)
Requirement already satisfied: fsspec>=0.6.0 in c:\programdata\anaconda3\lib\site
-packages (from dask->StandardScaler) (2022.11.0)
Requirement already satisfied: cloudpickle>=1.1.1 in c:\programdata\anaconda3\lib
\site-packages (from dask->StandardScaler) (2.0.0)
Requirement already satisfied: toolz>=0.8.2 in c:\programdata\anaconda3\lib\site-
packages (from dask->StandardScaler) (0.12.0)
Requirement already satisfied: pyyaml>=5.3.1 in c:\programdata\anaconda3\lib\site
-packages (from dask->StandardScaler) (6.0)
Requirement already satisfied: pytz>=2020.1 in c:\programdata\anaconda3\lib\site-
packages (from pandas->StandardScaler) (2022.7)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\programdata\anaconda3
\lib\site-packages (from pandas->StandardScaler) (2.8.2)
Requirement already satisfied: scipy in c:\programdata\anaconda3\lib\site-package
s (from scikit-elm->StandardScaler) (1.10.0)
Requirement already satisfied: joblib>=1.1.1 in c:\programdata\anaconda3\lib\site
-packages (from scikit-learn->StandardScaler) (1.1.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\programdata\anaconda3\l
ib\site-packages (from scikit-learn->StandardScaler) (2.2.0)
Requirement already satisfied: locket in c:\programdata\anaconda3\lib\site-packag
es (from partd>=0.3.10->dask->StandardScaler) (1.0.0)
Requirement already satisfied: six>=1.5 in c:\programdata\anaconda3\lib\site-pack
ages (from python-dateutil>=2.8.1->pandas->StandardScaler) (1.16.0)
Note: you may need to restart the kernel to use updated packages.
```

In [10]:
```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Training the Logistic Regression model on training set

In [11]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train,y_train)
```

Out[11]:

```
▾        LogisticRegression

LogisticRegression(random_state=0)
```

Predicting the Test set results

In [12]:

```
y_pred = classifier.predict(X_test)
```

Making the confusion Matrix

In [13]:

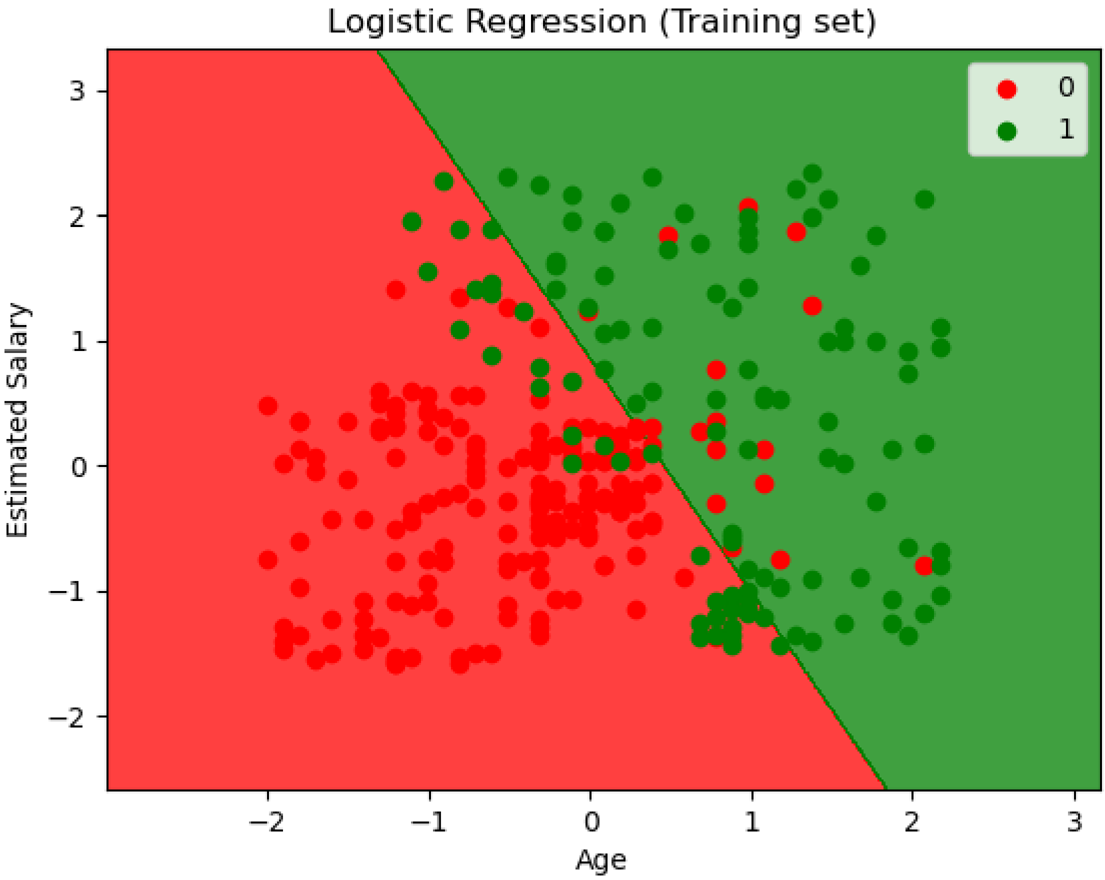
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test,y_pred)
print(cm)
```

```
[[65  3]
 [ 8 24]]
```

In [17]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1,X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() -1, stop = X_set[:, 0].m
                   np.arange(start = X_set[:, 1].min() -1, stop = X_set[:, 1].ma
plt.contourf(X1,X2,classifier.predict(np.array([X1.ravel(),X2.ravel()]).T).resha
             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','green'))(i),label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\dhana\AppData\Local\Temp\ipykernel_21176\3711601667.py:10: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be av
oided as value-mapping will have precedence in case its length matches with *x* &
*y*.  Please use the *color* keyword-argument or provide a 2D array with a single
row if you intend to specify the same RGB or RGBA value for all points.
  plt.scatter(X_set[y_set == j,0],X_set[y_set == j,1],
```


Logistic Regression (Training set)
0
1
3
2
1
0
−1
−2
Estimated Salary
−2
−1
0
1
2
3
Age

```
In [ ]:
```

Multiclass Classification using Decision Trees

Importing Required Libraries

In [1]:

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
```

Dataset

In [2]:

```
Iris_dataset = load_iris()
```

Training and Testing

In [3]:

```
X_train, X_test, y_train, y_test = train_test_split(Iris_dataset.data, Iris_dataset.target, rand
```

Training using Decision Tree Classifier

In [4]:

```
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)
```

Out[4]:

```
DecisionTreeClassifier()
```

Prediction

In [5]:

```
y_pred = clf.predict(X_test)
```

Accuracy

In [6]:

```
accuracy = np.mean(y_pred == y_test)
print("Accuracy is {:.2f} %".format(accuracy*100))
```

```
Accuracy is 97.37 %
```

K-Nearest Neighbor(KNN)

importing libraries and dataset

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd

data_set= pd.read_csv(r'''C:\Users\katka\Downloads\
Social_Network_Ads.csv''')
```

Extracting Independent and dependent Variable

```
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
```

Splitting the dataset into training and test set.

```
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=0)
```

feature Scaling

```
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

Fitting K-NN classifier to the training set

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

```
KNeighborsClassifier()
```

Predicting the test set result

```
y_pred= classifier.predict(x_test)
```

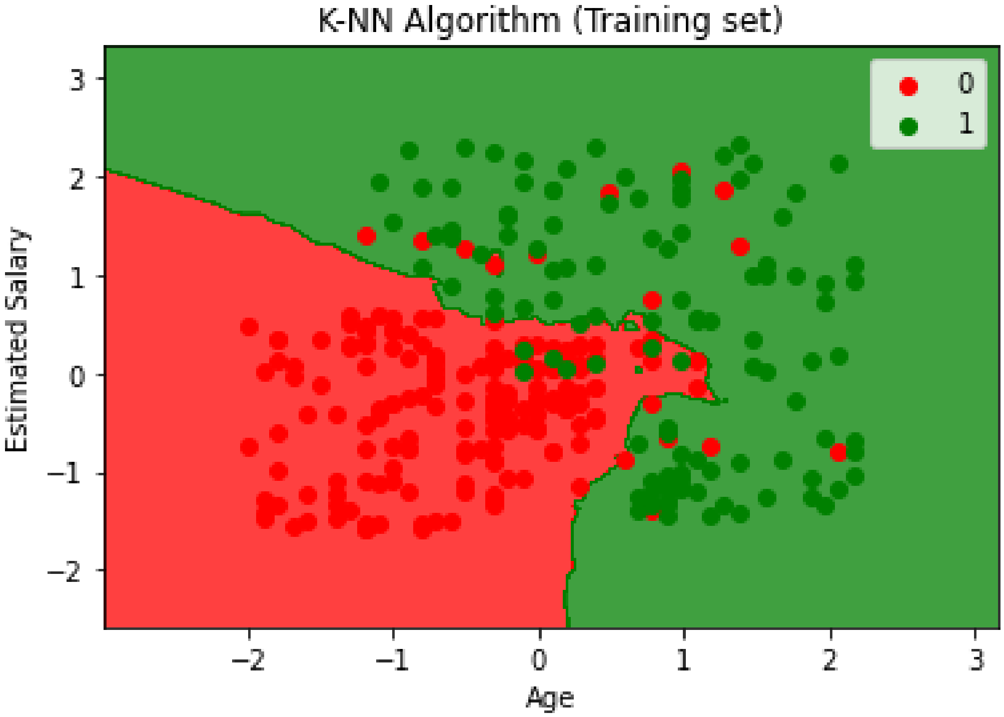
Creating the Confusion matrix

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

Visulaizing the trianing set result

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop =
x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1,
step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('red','green' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('red', 'green'))(i), label = j)
mtp.title('K-NN Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.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 array with a single row if you intend to
specify the same RGB or RGBA value for 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 array with a single row if you intend to
specify the same RGB or RGBA value for all points.
```

K-NN Algorithm (Training set)
Estimated Salary
3
2
1
0
−1
−2
Age
−2
−1
0
1
2
3
● 0
● 1

Naive Bayes

Importing the libraries

In [1]:

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

Importing the dataset

In [2]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

In [12]:

```
print(dataset)
```

```
      User ID  Gender  Age  EstimatedSalary  Purchased
0    15624510    Male   19            19000          0
1    15810944    Male   35            20000          0
2    15668575  Female   26            43000          0
3    15603246  Female   27            57000          0
4    15804002    Male   19            76000          0
..        ...     ...  ...              ...        ...
395  15691863  Female   46            41000          1
396  15706071    Male   51            23000          1
397  15654296  Female   50            20000          1
398  15755018    Male   36            33000          0
399  15594041  Female   49            36000          1

[400 rows x 5 columns]

---------------------------------------------------------------------------
-
AttributeError                            Traceback (most recent call las
t)
Cell In[12], line 1
----> 1 print(dataset).head(50)

AttributeError: 'NoneType' object has no attribute 'head'
```

Splitting the dataset into the Training set and Test set

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,
```

Feature Scaling

In [4]:

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

Training the Naive Bayes model on the Training set

In [5]:

```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

Out[5]:

```
▾ GaussianNB

GaussianNB()
```

Predicting the Test set results

In [6]:

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:

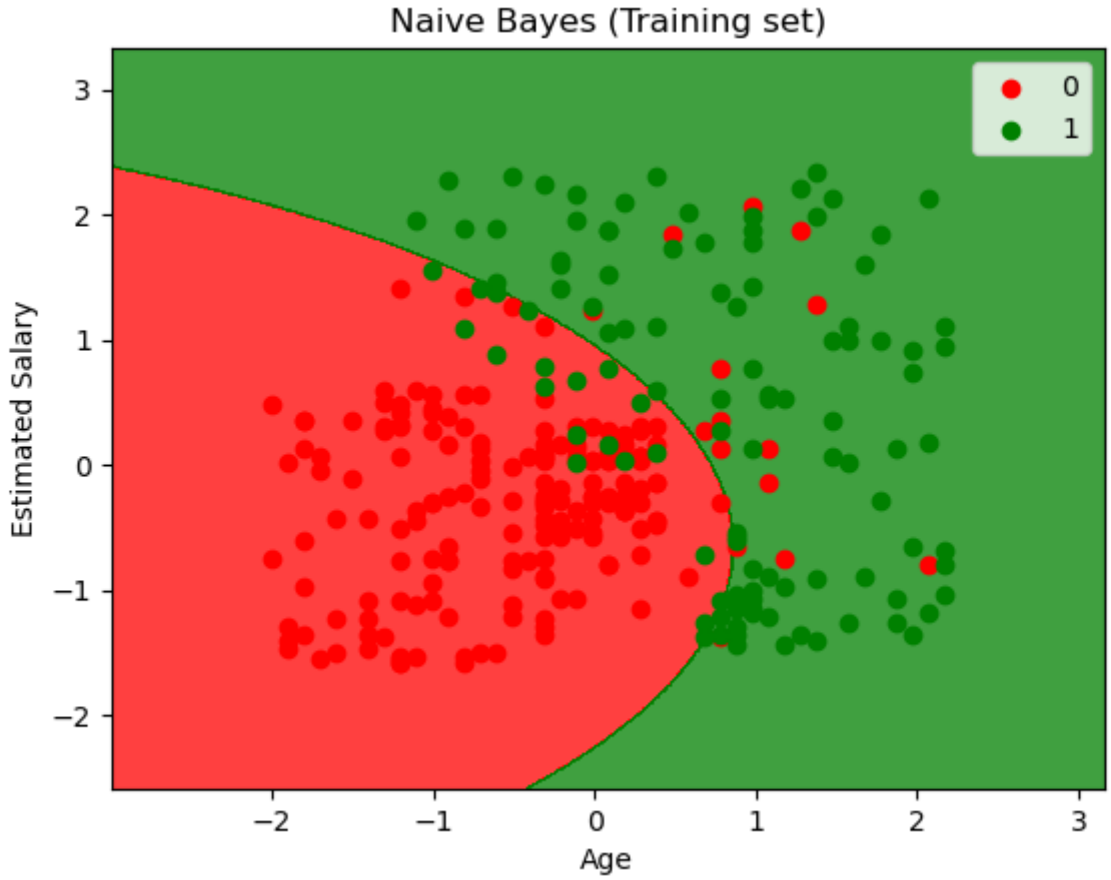
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[65  3]
 [ 7 25]]
```

Visualising the Training set results

```
In [8]: from matplotlib.colors import ListedColormap
        X_set, y_set = X_train, y_train
        X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[
                             np.arange(start = X_set[:, 1].min() - 1, stop = X_set[
        plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).
                     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', 'green'))(i), label = j)
        plt.title('Naive Bayes (Training set)')
        plt.xlabel('Age')
        plt.ylabel('Estimated Salary')
        plt.legend()
        plt.show()
```

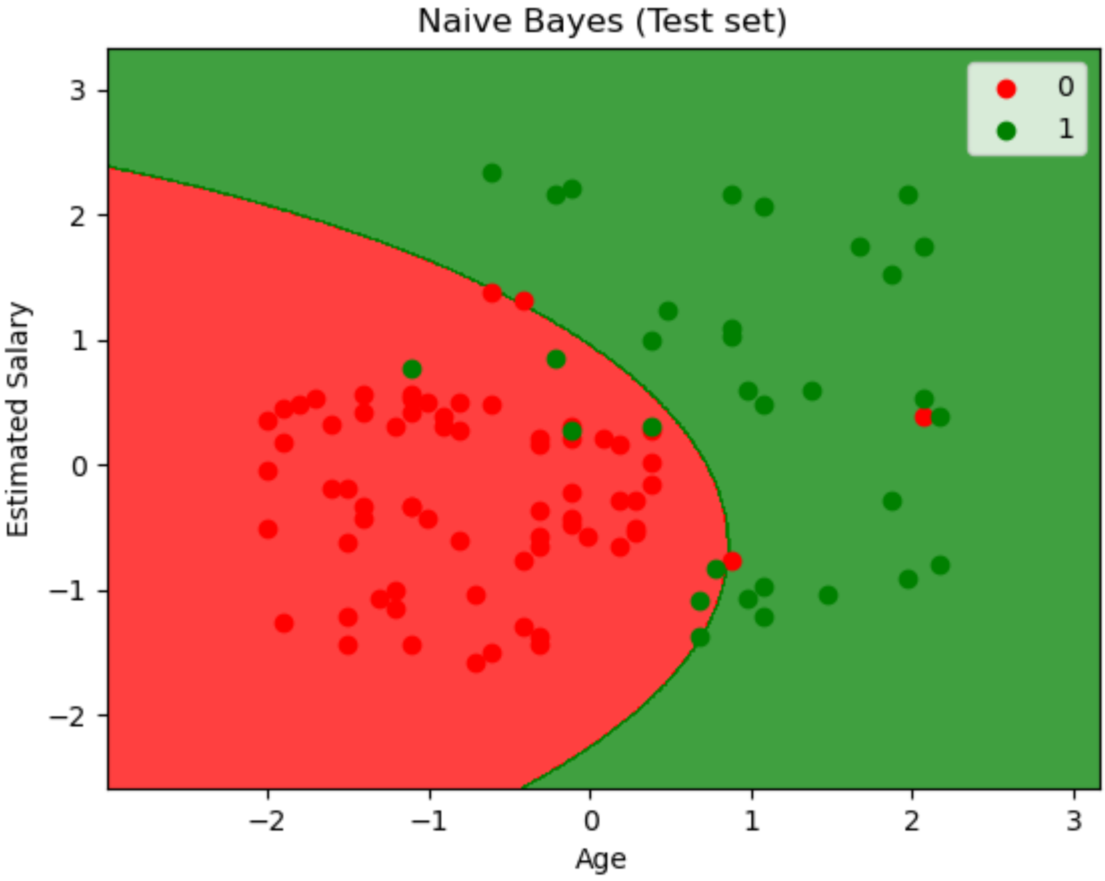
```
C:\Users\dyp\AppData\Local\Temp\ipykernel_4772\2643083737.py:10: UserWarni
ng: *c* argument looks like a single numeric RGB or RGBA sequence, which s
hould be avoided as value-mapping will have precedence in case its length
matches with *x* & *y*.  Please use the *color* keyword-argument or provid
e a 2D array with a single row if you intend to specify the same RGB or RG
BA value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



Visualising the Test set results

```
In [9]: from matplotlib.colors import ListedColormap
        X_set, y_set = X_test, y_test
        X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[
                             np.arange(start = X_set[:, 1].min() - 1, stop = X_set[
        plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).
                     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', 'green'))(i), label = j)
        plt.title('Naive Bayes (Test set)')
        plt.xlabel('Age')
        plt.ylabel('Estimated Salary')
        plt.legend()
        plt.show()
```

```
C:\Users\dyp\AppData\Local\Temp\ipykernel_4772\664088336.py:10: UserWarnin
g: *c* argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its length m
atches with *x* & *y*.  Please use the *color* keyword-argument or provide
a 2D array with a single row if you intend to specify the same RGB or RGBA
value for all points.
  plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
```



Support Vector Machine (SVM)

Importing the libraries

In [0]:

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

Importing the dataset

In [0]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
```

Feature Scaling

In [0]:

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

Training the SVM model on the Training set

In [5]:

```
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, y_train)
```

Out[5]:

```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=
0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='lin
ear',
    max_iter=-1, probability=False, random_state=0, shrinking=True, tol
=0.001,
    verbose=False)
```

Predicting the Test set results

In [0]:

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:

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

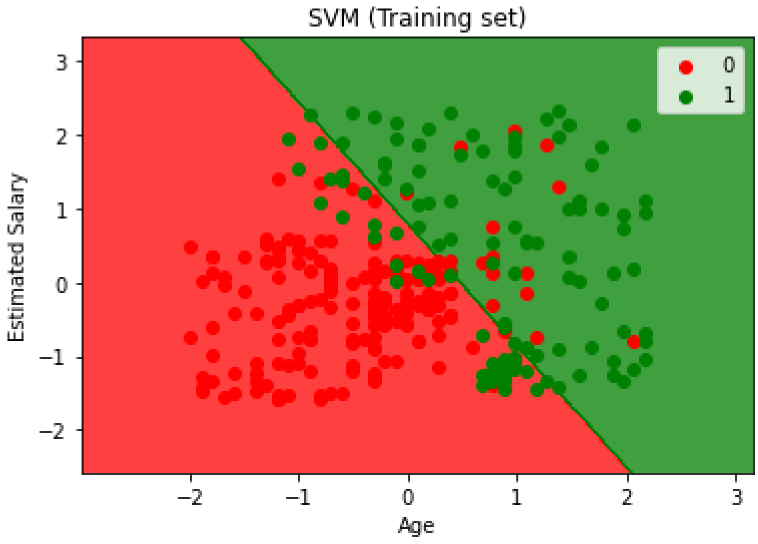
```
[[66  2]
 [ 8 24]]
```

Visualising the Training set results

In [8]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_s
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_s
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()
             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', 'green'))(i), label = j)
plt.title('SVM (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 sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
```

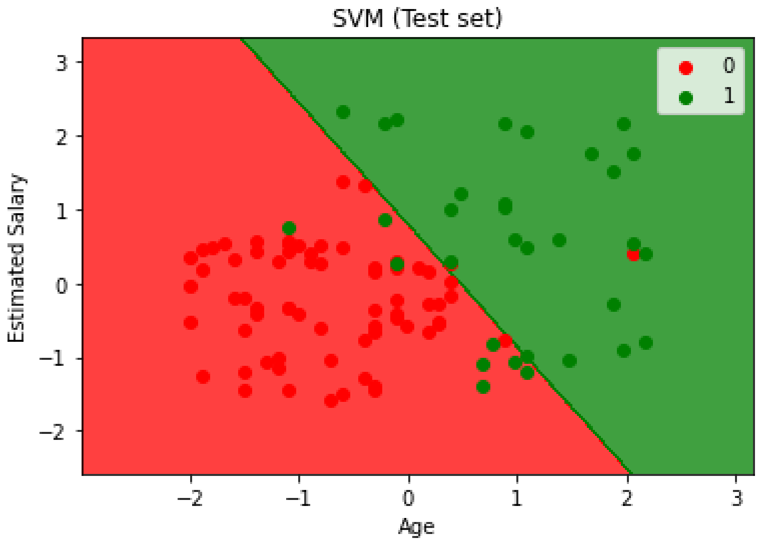


Visualising the Test set results

In [9]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_s
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_s
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()
             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', 'green'))(i), label = j)
plt.title('SVM (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 sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
```



Decision Tree Classification

Importing the libraries ¶

In [0]:

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

Importing the dataset

In [0]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [0]:

```
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,
```

Feature Scaling

In [0]:

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

Training the Decision Tree Classification model on the Training set

In [5]:

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

Out[5]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entrop
y',
                       max_depth=None, max_features=None, max_leaf_nodes=N
one,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=0, splitter='best')
```

Predicting the Test set results

In [0]:

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:

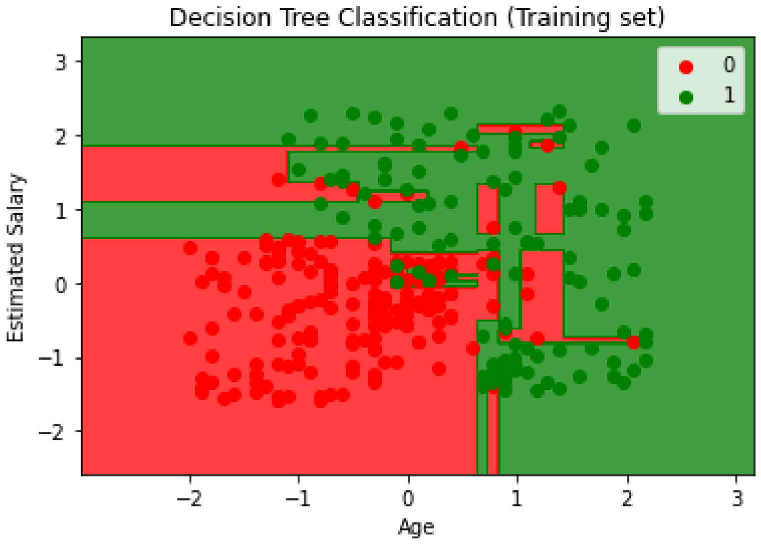
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

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

Visualising the Training set results

```
In [8]:
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).
             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', 'green'))(i), label = j)
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 shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with 'x' & 'y'.  Please use a 2-D array with a single row if you reall
y want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with 'x' & 'y'.  Please use a 2-D array with a single row if you reall
y want to specify the same RGB or RGBA value for all points.
```

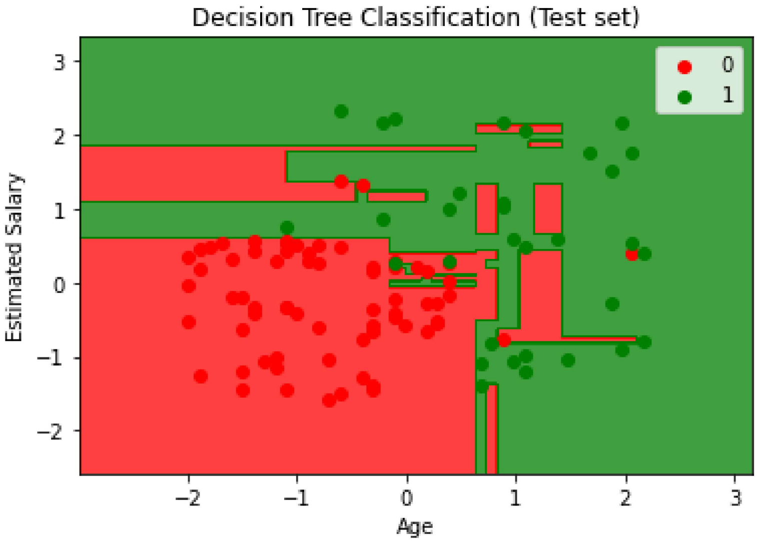


Visualising the Test set results

In [9]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).
             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', 'green'))(i), label = j)
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 shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with 'x' & 'y'.  Please use a 2-D array with a single row if you reall
y want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which shoul
d be avoided as value-mapping will have precedence in case its length matc
hes with 'x' & 'y'.  Please use a 2-D array with a single row if you reall
y want to specify the same RGB or RGBA value for all points.
```



Random Forest Classification

Importing the libraries

In [0]:

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

Importing the dataset

In [0]:

```
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, -1].values
```

Splitting the dataset into the Training set and Test set

In [0]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.
```

Feature Scaling

In [0]:

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

Training the Random Forest Classification model on the Training set

In [5]:

```
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entr
classifier.fit(X_train, y_train)
```

Out[5]:

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=Non
e,
                       criterion='entropy', max_depth=None, max_feature
s='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=No
ne,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10,
                       n_jobs=None, oob_score=False, random_state=0, ve
rbose=0,
                       warm_start=False)
```

Predicting the Test set results

In [0]:

```
y_pred = classifier.predict(X_test)
```

Making the Confusion Matrix

In [7]:

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

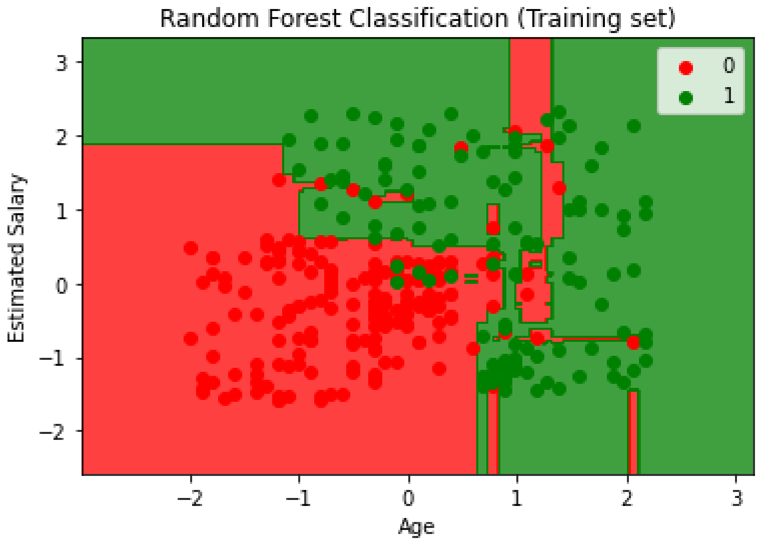
```
[[63  5]
 [ 4 28]]
```

Visualising the Training set results

In [8]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_s
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_s
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()
             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', 'green'))(i), label = j)
plt.title('Random Forest 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 sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
```

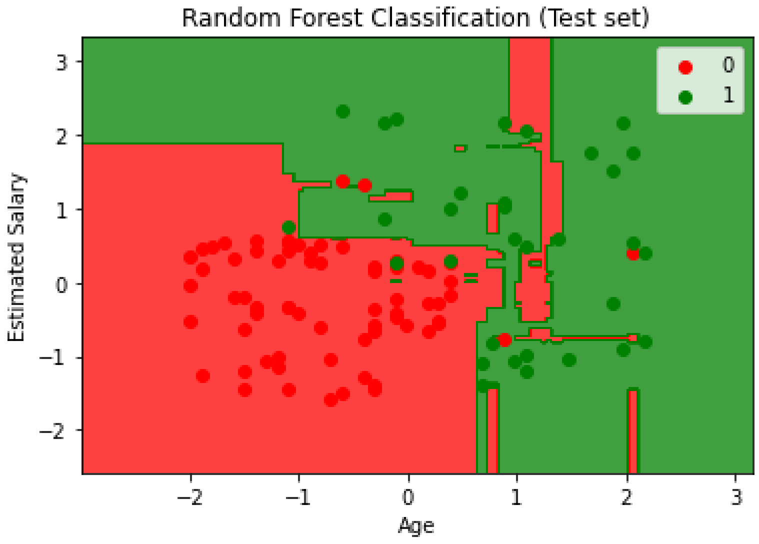


Visualising the Test set results

In [9]:

```
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_s
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_s
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()
             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', 'green'))(i), label = j)
plt.title('Random Forest 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 sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
'c' argument looks like a single numeric RGB or RGBA sequence, which sh
ould be avoided as value-mapping will have precedence in case its lengt
h matches with 'x' & 'y'.  Please use a 2-D array with a single row if
you really want to specify the same RGB or RGBA value for all points.
```


Random Forest Classification (Test set)
● 0
● 1
Estimated Salary
Age