Logistic Regression

October 15, 2024

1 Logistic Regression

[13]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import os
```

[14]:
```
# Importing the dataset
os.chdir("C:\\Users\ddaya\OneDrive\Documents\Python_programming")
dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

[15]:
```
# Splitting the dataset into the Training set 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)
print(X_train)
print(y_train)
print(X_test)
print(y_test)
```

```
[[    37  71000]
 [    36  50000]
 [    48  29000]
 [    30  87000]
 [    32  18000]
 [    32 100000]
 [    47  25000]
 [    40  75000]
 [    29  47000]
 [    27  17000]
 [    31  76000]
 [    48  41000]
 [    26  35000]
 [    31  15000]
 [    41  51000]
```

```
[     24   27000]
[     29   75000]
[     38   61000]
[     47   30000]
[     20   74000]
[     33   31000]
[     35   25000]
[     35   75000]
[     24   55000]
[     23   66000]
[     26   43000]
[     32  117000]
[     29   83000]
[     35   44000]
[     30   15000]
[     26   80000]
[     35   88000]
[     33  113000]
[     18   86000]
[     26   86000]
[     28   59000]
[     22   81000]
[     20   82000]
[     33   69000]
[     24   32000]
[     37   55000]
[     46   59000]
[     59   83000]
[     41   59000]
[     37   72000]
[     31   68000]
[     31   89000]
[     30  135000]
[     31  118000]
[     32   18000]
[     31   58000]
[     45   22000]
[     18   68000]
[     45   22000]
[     29   80000]
[     28   87000]
[     29   83000]
[     18   52000]
[     27   57000]
[     21   88000]
[     26   84000]
[     27   84000]
[     22   63000]
```

```
[    34 112000]
[    30  62000]
[    28  55000]
[    20  86000]
[    21  68000]
[    27  96000]
[    18  82000]
[    19  19000]
[    30 116000]
[    41  30000]
[    25  79000]
[    26  52000]
[    32  86000]
[    35  27000]
[    28  79000]
[    33  51000]
[    35  71000]
[    24  58000]
[    35  20000]
[    36  75000]
[    19  25000]
[    32 135000]
[    35 108000]
[    21  72000]
[    26  15000]
[    45  26000]
[    26  72000]
[    35  23000]
[    23  63000]
[    30  17000]
[    27  90000]
[    29  43000]
[    42  80000]
[    27 137000]
[    30  89000]
[    26  32000]
[    41  45000]
[    21  16000]
[    35  59000]
[    31  18000]
[    28  37000]
[    20  49000]
[    26  17000]
[    40  57000]
[    29  61000]
[    22  55000]
[    20  23000]
[    22  27000]
```

```
 [     47  20000]
 [     39  42000]
 [     25  90000]
 [     35  38000]
 [     27  31000]
 [     22  18000]
 [     19  85000]
 [     26  81000]
 [     25  80000]
 [     28  85000]
 [     33  28000]
 [     47  49000]
 [     35  65000]
 [     33 149000]
 [     23  82000]
 [     36  52000]
 [     27  54000]]
[0 0 1 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 1 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0]
[[     27  58000]
 [     24  89000]
 [     32 120000]
 [     24  55000]
 [     39  61000]
 [     27  88000]
 [     32 150000]
 [     40  47000]
 [     19  21000]
 [     35  50000]
 [     30 107000]
 [     34  25000]
 [     27  58000]
 [     28 123000]
 [     37  33000]
 [     29  28000]
 [     28  44000]
 [     46  28000]
 [     25  87000]
 [     18  44000]
 [     24  19000]
 [     30  49000]
 [     19  76000]
 [     40  59000]
 [     27  20000]
 [     38  80000]
 [     35  53000]
```

```
 [     28  59000]
 [     39  71000]
 [     26  30000]
 [     49  28000]
 [     23  48000]
 [     42  65000]
 [     23  20000]
 [     25  33000]
 [     28  84000]
 [     30  80000]
 [     27  89000]
 [     35  73000]
 [     29 148000]
 [     46  23000]
 [     31  74000]
 [     26  15000]]
[0 0 1 0 0 0 1 0 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
 0 0 1 1 0 0]
```

[16]:

```python
# Feature Scaling
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)
```

```
[[ 8.18877097e-01  3.33231348e-01]
 [ 6.91518326e-01 -3.63289541e-01]
 [ 2.21982358e+00 -1.05981043e+00]
 [-7.26342990e-02  8.63913930e-01]
 [ 1.82083243e-01 -1.42465470e+00]
 [ 1.82083243e-01  1.29509353e+00]
 [ 2.09246481e+00 -1.19248108e+00]
 [ 1.20095341e+00  4.65901993e-01]
 [-1.99993070e-01 -4.62792525e-01]
 [-4.54710612e-01 -1.45782237e+00]
 [ 5.47244718e-02  4.99069655e-01]
 [ 2.21982358e+00 -6.61798493e-01]
 [-5.82069382e-01 -8.60804461e-01]
 [ 5.47244718e-02 -1.52415769e+00]
 [ 1.32831218e+00 -3.30121880e-01]
 [-8.36786924e-01 -1.12614575e+00]
 [-1.99993070e-01  4.65901993e-01]
 [ 9.46235868e-01  1.55473413e-03]
 [ 2.09246481e+00 -1.02664277e+00]
 [-1.34622201e+00  4.32734332e-01]
 [ 3.09442014e-01 -9.93475107e-01]
```

```
[ 5.64159555e-01 -1.19248108e+00]
[ 5.64159555e-01  4.65901993e-01]
[-8.36786924e-01 -1.97451234e-01]
[-9.64145695e-01  1.67393041e-01]
[-5.82069382e-01 -5.95463170e-01]
[ 1.82083243e-01  1.85894377e+00]
[-1.99993070e-01  7.31243284e-01]
[ 5.64159555e-01 -5.62295509e-01]
[-7.26342990e-02 -1.52415769e+00]
[-5.82069382e-01  6.31740300e-01]
[ 5.64159555e-01  8.97081591e-01]
[ 3.09442014e-01  1.72627313e+00]
[-1.60093955e+00  8.30746268e-01]
[-5.82069382e-01  8.30746268e-01]
[-3.27351841e-01 -6.47805886e-02]
[-1.09150447e+00  6.64907961e-01]
[-1.34622201e+00  6.98075623e-01]
[ 3.09442014e-01  2.66896025e-01]
[-8.36786924e-01 -9.60307445e-01]
[ 8.18877097e-01 -1.97451234e-01]
[ 1.96510603e+00 -6.47805886e-02]
[ 3.62077006e+00  7.31243284e-01]
[ 1.32831218e+00 -6.47805886e-02]
[ 8.18877097e-01  3.66399009e-01]
[ 5.47244718e-02  2.33728364e-01]
[ 5.47244718e-02  9.30249252e-01]
[-7.26342990e-02  2.45596168e+00]
[ 5.47244718e-02  1.89211143e+00]
[ 1.82083243e-01 -1.42465470e+00]
[ 5.47244718e-02 -9.79482500e-02]
[ 1.83774726e+00 -1.29198406e+00]
[-1.60093955e+00  2.33728364e-01]
[ 1.83774726e+00 -1.29198406e+00]
[-1.99993070e-01  6.31740300e-01]
[-3.27351841e-01  8.63913930e-01]
[-1.99993070e-01  7.31243284e-01]
[-1.60093955e+00 -2.96954218e-01]
[-4.54710612e-01 -1.31115911e-01]
[-1.21886324e+00  8.97081591e-01]
[-5.82069382e-01  7.64410946e-01]
[-4.54710612e-01  7.64410946e-01]
[-1.09150447e+00  6.78900569e-02]
[ 4.36800784e-01  1.69310546e+00]
[-7.26342990e-02  3.47223955e-02]
[-3.27351841e-01 -1.97451234e-01]
[-1.34622201e+00  8.30746268e-01]
[-1.21886324e+00  2.33728364e-01]
[-4.54710612e-01  1.16242288e+00]
```

```
[-1.60093955e+00  6.98075623e-01]
[-1.47358078e+00 -1.39148704e+00]
[-7.26342990e-02  1.82577611e+00]
[ 1.32831218e+00 -1.02664277e+00]
[-7.09428153e-01  5.98572639e-01]
[-5.82069382e-01 -2.96954218e-01]
[ 1.82083243e-01  8.30746268e-01]
[ 5.64159555e-01 -1.12614575e+00]
[-3.27351841e-01  5.98572639e-01]
[ 3.09442014e-01 -3.30121880e-01]
[ 5.64159555e-01  3.33231348e-01]
[-8.36786924e-01 -9.79482500e-02]
[ 5.64159555e-01 -1.35831938e+00]
[ 6.91518326e-01  4.65901993e-01]
[-1.47358078e+00 -1.19248108e+00]
[ 1.82083243e-01  2.45596168e+00]
[ 5.64159555e-01  1.56043482e+00]
[-1.21886324e+00  3.66399009e-01]
[-5.82069382e-01 -1.52415769e+00]
[ 1.83774726e+00 -1.15931341e+00]
[-5.82069382e-01  3.66399009e-01]
[ 5.64159555e-01 -1.25881640e+00]
[-9.64145695e-01  6.78900569e-02]
[-7.26342990e-02 -1.45782237e+00]
[-4.54710612e-01  9.63416914e-01]
[-1.99993070e-01 -5.95463170e-01]
[ 1.45567095e+00  6.31740300e-01]
[-4.54710612e-01  2.52229700e+00]
[-7.26342990e-02  9.30249252e-01]
[-5.82069382e-01 -9.60307445e-01]
[ 1.32831218e+00 -5.29127848e-01]
[-1.21886324e+00 -1.49099003e+00]
[ 5.64159555e-01 -6.47805886e-02]
[ 5.47244718e-02 -1.42465470e+00]
[-3.27351841e-01 -7.94469139e-01]
[-1.34622201e+00 -3.96457202e-01]
[-5.82069382e-01 -1.45782237e+00]
[ 1.20095341e+00 -1.31115911e-01]
[-1.99993070e-01  1.55473413e-03]
[-1.09150447e+00 -1.97451234e-01]
[-1.34622201e+00 -1.25881640e+00]
[-1.09150447e+00 -1.12614575e+00]
[ 2.09246481e+00 -1.35831938e+00]
[ 1.07359464e+00 -6.28630832e-01]
[-7.09428153e-01  9.63416914e-01]
[ 5.64159555e-01 -7.61301477e-01]
[-4.54710612e-01 -9.93475107e-01]
[-1.09150447e+00 -1.42465470e+00]
```

```
 [-1.47358078e+00  7.97578607e-01]
 [-5.82069382e-01  6.64907961e-01]
 [-7.09428153e-01  6.31740300e-01]
 [-3.27351841e-01  7.97578607e-01]
 [ 3.09442014e-01 -1.09297809e+00]
 [ 2.09246481e+00 -3.96457202e-01]
 [ 5.64159555e-01  1.34225380e-01]
 [ 3.09442014e-01  2.92030893e+00]
 [-9.64145695e-01  6.98075623e-01]
 [ 6.91518326e-01 -2.96954218e-01]
 [-4.54710612e-01 -2.30618895e-01]]
[[-4.54710612e-01 -9.79482500e-02]
 [-8.36786924e-01  9.30249252e-01]
 [ 1.82083243e-01  1.95844675e+00]
 [-8.36786924e-01 -1.97451234e-01]
 [ 1.07359464e+00  1.55473413e-03]
 [-4.54710612e-01  8.97081591e-01]
 [ 1.82083243e-01  2.95347660e+00]
 [ 1.20095341e+00 -4.62792525e-01]
 [-1.47358078e+00 -1.32515172e+00]
 [ 5.64159555e-01 -3.63289541e-01]
 [-7.26342990e-02  1.52726716e+00]
 [ 4.36800784e-01 -1.19248108e+00]
 [-4.54710612e-01 -9.79482500e-02]
 [-3.27351841e-01  2.05794974e+00]
 [ 8.18877097e-01 -9.27139784e-01]
 [-1.99993070e-01 -1.09297809e+00]
 [-3.27351841e-01 -5.62295509e-01]
 [ 1.96510603e+00 -1.09297809e+00]
 [-7.09428153e-01  8.63913930e-01]
 [-1.60093955e+00 -5.62295509e-01]
 [-8.36786924e-01 -1.39148704e+00]
 [-7.26342990e-02 -3.96457202e-01]
 [-1.47358078e+00  4.99069655e-01]
 [ 1.20095341e+00 -6.47805886e-02]
 [-4.54710612e-01 -1.35831938e+00]
 [ 9.46235868e-01  6.31740300e-01]
 [ 5.64159555e-01 -2.63786557e-01]
 [-3.27351841e-01 -6.47805886e-02]
 [ 1.07359464e+00  3.33231348e-01]
 [-5.82069382e-01 -1.02664277e+00]
 [ 2.34718235e+00 -1.09297809e+00]
 [-9.64145695e-01 -4.29624864e-01]
 [ 1.45567095e+00  1.34225380e-01]
 [-9.64145695e-01 -1.35831938e+00]
 [-7.09428153e-01 -9.27139784e-01]
 [-3.27351841e-01  7.64410946e-01]
 [-7.26342990e-02  6.31740300e-01]
```

```
 [-4.54710612e-01  9.30249252e-01]
 [ 5.64159555e-01  3.99566671e-01]
 [-1.99993070e-01  2.88714127e+00]
 [ 1.96510603e+00 -1.25881640e+00]
 [ 5.47244718e-02  4.32734332e-01]
 [-5.82069382e-01 -1.52415769e+00]]
```

[17]:
```
# Training the Logistic Regression model on the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

[17]:
```
LogisticRegression(random_state=0)
```

[18]:
```
# Predicting a new result
print(classifier.predict(sc.transform([[30,87000]])))
```

```
[0]
```

[19]:
```
# Predicting the Test set results
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 1]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [0 0]
 [0 0]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [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]
 [0 0]
 [0 0]
 [0 0]
 [0 1]
 [0 1]
 [0 0]
 [0 0]]
```

[20]:
```python
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix, accuracy_score
cm = confusion_matrix(y_test, y_pred)
print(cm)
accuracy_score(y_test, y_pred)
```

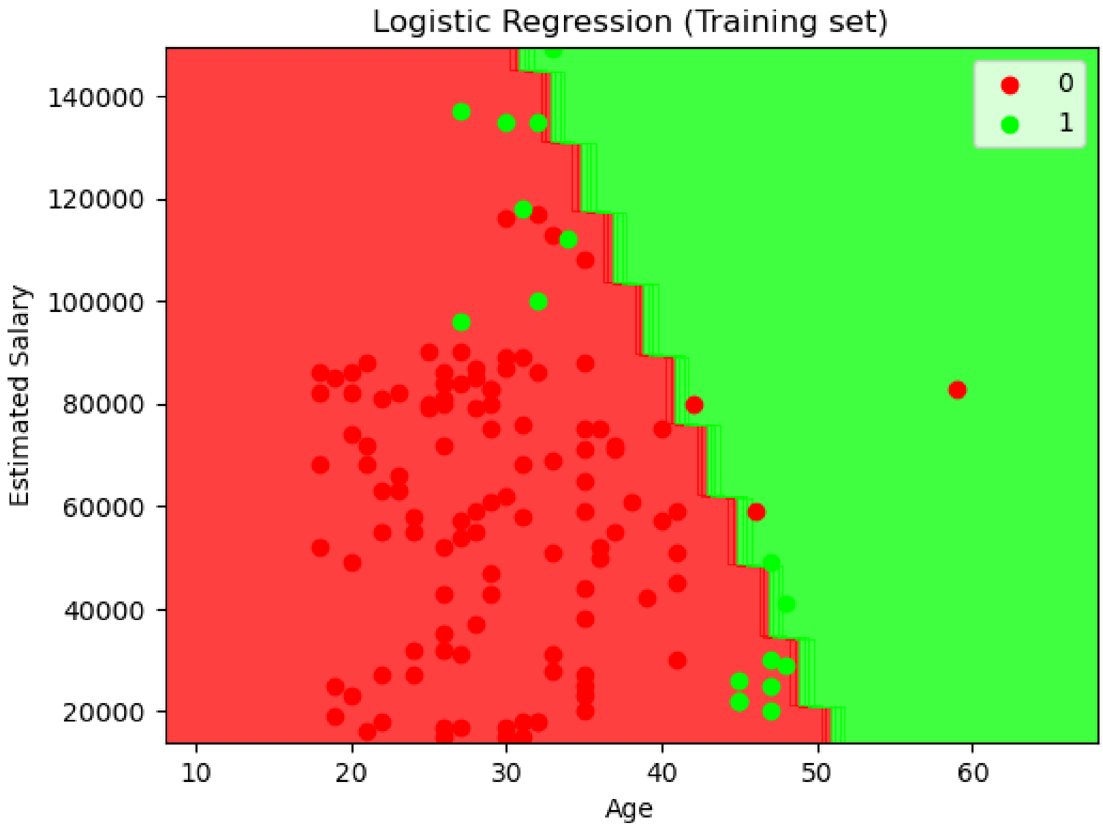
```
[[35  0]
 [ 6  2]]
```

[20]: 0.8604651162790697

[42]:
```python
# Visualising the Training set results
from matplotlib.colors import ListedColormap
import numpy as np
import matplotlib.pyplot as plt

# Inverse transform to get the original scale
X_set, y_set = sc.inverse_transform(X_train), y_train

# Create the meshgrid with a larger step size
step_size_age = 2  # Increased step size for age
step_size_salary = 500  # Increased step size for estimated salary

# Generate the meshgrid for plotting
X1, X2 = np.meshgrid(
    np.arange(start=X_set[:, 0].min() - 10, stop=X_set[:, 0].max() + 10,␣
 ↪step=step_size_age),
    np.arange(start=X_set[:, 1].min() - 1000, stop=X_set[:, 1].max() + 1000,␣
 ↪step=step_size_salary)
```

```
)

# Predict the classifier output and reshape
Z = classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).
↪reshape(X1.shape)

# Plotting
plt.contourf(X1, X2, Z, alpha=0.75, cmap=ListedColormap([(1, 0, 0), (0, 1,␣
↪0)]))  # Using RGB tuples for colors
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# Scatter plot the training points
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([(1, 0, 0), (0, 1, 0)])(i), label=j)  # Same␣
↪color scheme
                # Alternatively, you can use:
                # c=['red', 'green'][i]

plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

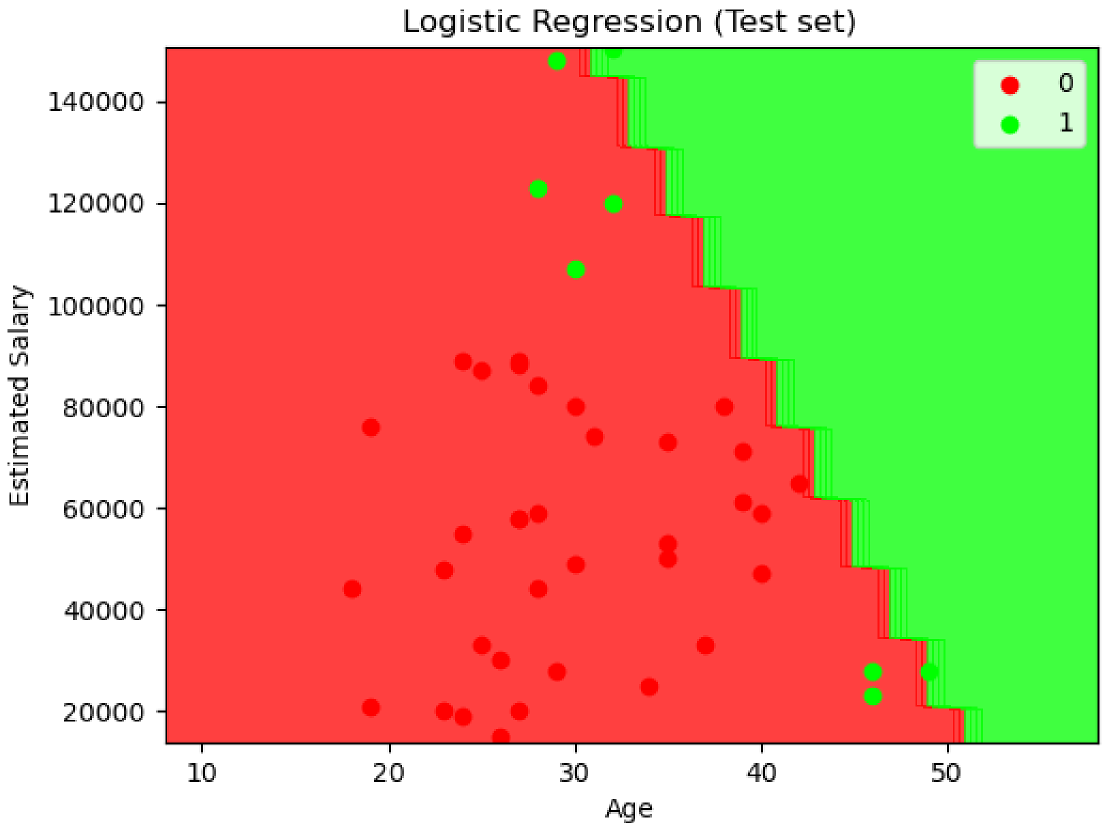
```
C:\Users\ddaya\AppData\Local\Temp\ipykernel_15000\4181641556.py:29: UserWarning:
*c* argument looks like a single numeric RGB or RGBA sequence, which should be
avoided as value-mapping will have precedence in case its length matches with
*x* & *y*.  Please use the *color* keyword-argument or provide a 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],
```



[44]:
```
# Visualising the Training set results
from matplotlib.colors import ListedColormap
import numpy as np
import matplotlib.pyplot as plt

# Inverse transform to get the original scale
X_set, y_set = sc.inverse_transform(X_test), y_test

# Create the meshgrid with a larger step size
step_size_age = 2  # Increased step size for age
step_size_salary = 500   # Increased step size for estimated salary

# Generate the meshgrid for plotting
X1, X2 = np.meshgrid(
    np.arange(start=X_set[:, 0].min() - 10, stop=X_set[:, 0].max() + 10,␣
 ↪step=step_size_age),
    np.arange(start=X_set[:, 1].min() - 1000, stop=X_set[:, 1].max() + 1000,␣
 ↪step=step_size_salary)
)
```

```python
# Predict the classifier output and reshape
Z = classifier.predict(sc.transform(np.array([X1.ravel(), X2.ravel()]).T)).
↪reshape(X1.shape)

# Plotting
plt.contourf(X1, X2, Z, alpha=0.75, cmap=ListedColormap([(1, 0, 0), (0, 1,␣
↪0)]))  # Using RGB tuples for colors
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())

# Scatter plot the training points
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([(1, 0, 0), (0, 1, 0)])(i), label=j)  # Same␣
↪color scheme
                # Alternatively, you can use:
                # c=['red', 'green'][i]

plt.title('Logistic Regression (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
C:\Users\ddaya\AppData\Local\Temp\ipykernel_15000\2172703052.py:29: UserWarning:
*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 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 (Test set)
0
1
140000
120000
100000
80000
60000
40000
20000
Estimated Salary
10
20
30
40
50
Age

[ ]: