Page No.
Date

Name:- Rohini J. Devkar
Roll no:- 23272
PRN no:- 72030818G
TE-2
DSBDA Practical

Practical No. 5

Aim:- 1] Implement logistic regression using Python/R to perform classification on Social_Network_ADS.csv dataset
2] Compute confusion matrix to find TP, FP, TN, FN, Accuracy, Error Rate, Precision, Recall on the given dataset.

Theory:-

* Logistic Regression:-

Logistic Regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated, and it's convienient, ~~and it's con~~ for you to interpret the results. Although, it's essentially a method for binary classification, it can also be applied to multiclass problems.

o Logistic regression ~~is one of the~~ predicts the output of a categorical dependent variable. Therefore the outcome must be a categorical or discrete value. It can be either yes or no, 0 or 1, true or false, etc. but instead of giving the exact value as 0 and 1, it gives the probabilistic values lie between 0 and 1.

o In logistic regression, instead of fitting a regression line, we fit an "S" shaped logistic function, which predicts two maximum values (0 or 1).

* Type of Logistic Regression :-

1) Binomial :- In binomial logistic regression, there can be only two possible types of the dependent variables, such as 0 or 1, Pass or fail, etc.

2) Multinomial :- In multinomial logistic regression, there can be 3 or more possible unordered types of the dependent variable, such as "cat", "dogs", or "sheep".

3) Ordinal :- In ordinal logistic regression, there can be 3 or more possible ordered types of dependent variables, such as "low", "medium", or "high".

* Steps in Logistic Regression :-

1) Data Pre-processing step
2) Fitting logistic regression to the training set.
3) Predicting the test result.
4) Test accuracy of the result (Creation of confusion matrix)
5) Visualizing the test set result.

* Confusion Matrix of Logistic regression :-

The confusion matrix is a two by two table that contains four outcomes produced by a binary classifier.

Confusion matrix		Predicted	
		Positive	Negative
Observed	Positive	TP	FN
	Negative	FP	TN

- Also, it is useful to calculate accuracy, error rate, precision and recall.

Page No.
Date

- The above table has given following cases:-
  - True Negative:- model has given prediction No, the real or actual value was also No.
  - True positive:- The model has predicted yes, and the actual value was also true.
  - False Negative:- The model has predicted ~~yes~~ no and but the actual value was Yes, it is also called as Type-II error.
  - False Positive:- The model has predicted yes but the actual value was ~~Yes~~ no, it is called Type-I error.

- We can perform various calculations for the model, such as model's accuracy, using this matrix. These calculations are given below:-

① Accuracy:- It is calculated as the number of all correct predictions divided by the total number of the dataset. The best accuracy is 1.0, whereas the worst is 0.0.

$$ACC = \frac{TP+TN}{TP+TN+FN+FP} = \frac{TP+TN}{P+N}$$

② Error Rate:- Error Rate is calculated as the number of all incorrect predictions divided by the total number of the dataset. The best error rate is 0.0, whereas the worst is 1.0.

$$ERR = \frac{FP+FN}{TP+TN+FN+FP} = \frac{FP+FN}{P+N}$$

③ Precision:- Precision is calculated as the number of correct positive predictions divided by the total number of positive predictions. It is also called positive predictive value (PPV). The best precision is 1.0, whereas the worst is 0.0.

$$PREC = \frac{TP}{TP+FP}$$

Scanned with CamScanner

Page No.
Date

(4) Recall :- It is calculated as the number of correct positive predictions divided by the total number of positives or positive observations. It is also called as Sensitivity or True positive rate. The best recall is 1.0, whereas the worst is 0.0.

$$SN = \frac{TP}{TP + FN} = \frac{TP}{P}$$

* In this practical, we use Social_Network_Ads.csv dataset for logistic regression.

Now, we find confusion matrix using this dataset.

We know,

~~P~~ Confusion matrix is,

		Predicted	
		+ve	-ve
Observed	+ve	74	5
	-ve	11	30

TP = 74, FN = 5, FP = 11, TN = 30

then,

1) $Acc = \frac{TP + TN}{P + N} = \frac{74 + 30}{120} = 0.875$

2) $Err\ Rate = \frac{FP + FN}{P + N} = \frac{5 + 11}{120} = 0.458$

3) $Prec = \frac{TP}{TP + FP} = \frac{74}{74 + 11} = 0.870$

4) $Recall = \frac{TP}{TP + FN} = \frac{74}{74 + 5} = 0.936$

* Conclusion :- Logistic regression works fine only when the target variable is discrete in nature. They do not have the flexibility to act as regression analysis.

Data Science And Big Data Analytics

Name:- Rohini Janardan Devkar

PRN NO:- 72030818G

Roll no:- 23272

Class :- TE 2(COMP)

Problem statement:-

Implement logistic regression using Python/R to perform classification on Social_Network_Ads.csv dataset

Compute Confusion matrix to find TP, FP, TN, FN, Accuracy, Error rate, Precision, Recall on the given dataset.

In [21]:
```
#importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In [22]:
```
#importing datasets
data_set= pd.read_csv('Social_Network_Ads.csv')
```

In [24]:
```
#Checking the dataset
dataset.head()
```

Out[24]:

	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 [25]:
```
#Check the metadata
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
```

```
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

In [26]:

```
#Chceking the null values
dataset.isnull().sum()
```

Out[26]:

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

In [27]:

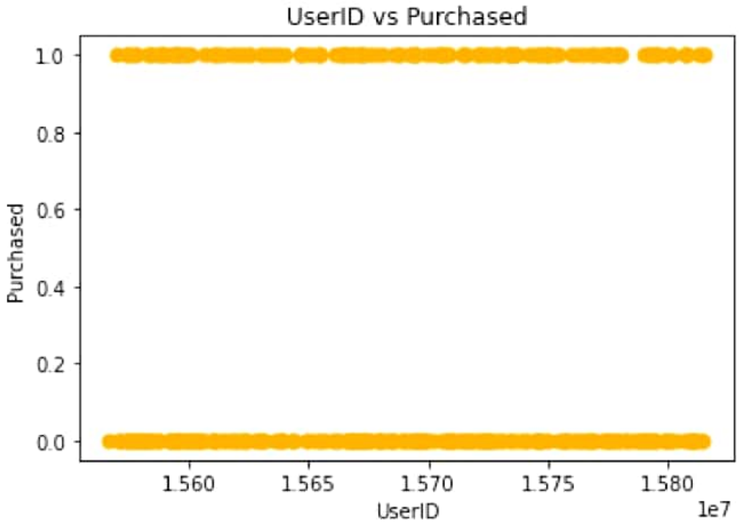
```
#Check its dimensions
dataset.shape
```

Out[27]:

```
(400, 5)
```

In [28]:

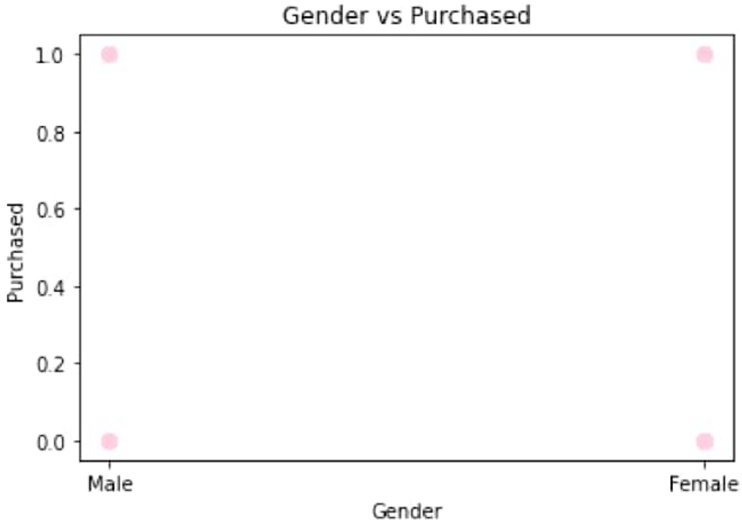
```
#Plot UserID vs Purchased...........
x1 = dataset.iloc[:, 0].values
y1 = dataset.iloc[:, 4].values
plt.scatter(x1,y1,color='Orange',s=50)
plt.xlabel('UserID')
plt.ylabel('Purchased')
plt.title('UserID vs Purchased')
plt.show()
```



In [29]:

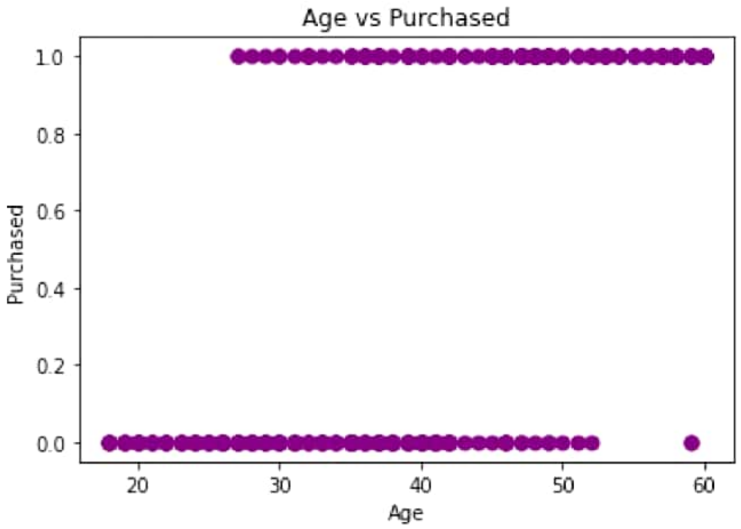
```
#Plot Gender vs Purchased...........
x1 = dataset.iloc[:, 1].values
y1 = dataset.iloc[:, 4].values
plt.scatter(x1,y1,color='pink',s=50)
plt.xlabel('Gender')
plt.ylabel('Purchased')
```

```
plt.title('Gender vs Purchased')
plt.show()
```



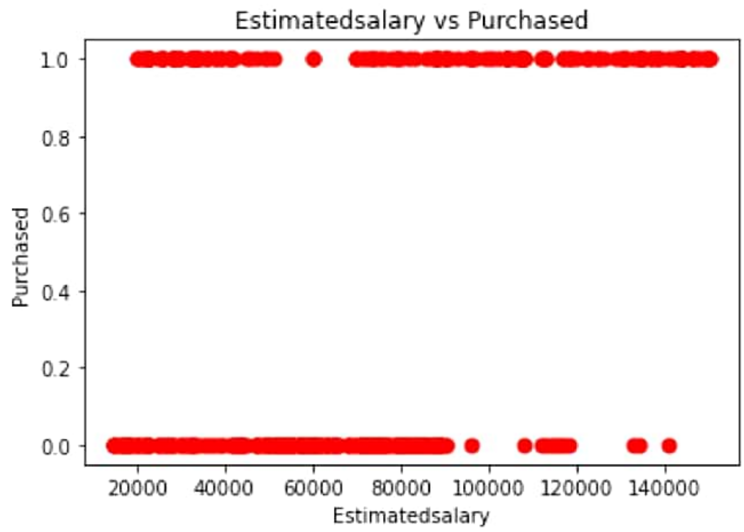
In [30]:

```
#Plot Age vs Purchased............
x1 = dataset.iloc[:, 2].values
y1 = dataset.iloc[:, 4].values
plt.scatter(x1,y1,color='purple',s=50)
plt.xlabel('Age')
plt.ylabel('Purchased')
plt.title('Age vs Purchased')
plt.show()
```



In [31]:

```
#Plot Estimatedsalary vs Purchased............
x1 = dataset.iloc[:, 3].values
y1 = dataset.iloc[:, 4].values
plt.scatter(x1,y1,color='red',s=50)
plt.xlabel('Estimatedsalary')
plt.ylabel('Purchased')
plt.title('Estimatedsalary vs Purchased')
plt.show()
```



In [32]:
```
#Headmap:-To see the correlation between them!
import seaborn as sns
plt.figure(figsize=(7,4)) #7 is the size of the width and 4 is parts....
sns.heatmap(dataset.corr(),annot=True,cmap='cubehelix_r')
```

Out[32]: <AxesSubplot:>

In [33]:
```
#Seperating dependent and indepndent values
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

In [34]:
```
print(X)
```

```
[[    19  19000]
 [    35  20000]
 [    26  43000]
 [    27  57000]
 [    19  76000]
 [    27  58000]
 [    27  84000]
 [    32 150000]
 [    25  33000]
```

```
 [    35  65000]
 [    26  80000]
 [    26  52000]
 [    20  86000]
 [    32  18000]
 [    18  82000]
 [    29  80000]
 [    47  25000]
 [    45  26000]
 [    46  28000]
 [    48  29000]
 [    45  22000]
 [    47  49000]
 [    48  41000]
 [    45  22000]
 [    46  23000]
 [    47  20000]
 [    49  28000]
 [    47  30000]
 [    29  43000]
 [    31  18000]
 [    31  74000]
 [    27 137000]
 [    21  16000]
 [    28  44000]
 [    27  90000]
 [    35  27000]
 [    33  28000]
 [    30  49000]
 [    26  72000]
 [    27  31000]
 [    27  17000]
 [    33  51000]
 [    35 108000]
 [    30  15000]
 [    28  84000]
 [    23  20000]
 [    25  79000]
 [    27  54000]
 [    30 135000]
 [    31  89000]
 [    24  32000]
 [    18  44000]
 [    29  83000]
 [    35  23000]
 [    27  58000]
 [    24  55000]
 [    23  48000]
 [    28  79000]
 [    22  18000]
 [    32 117000]
 [    27  20000]
 [    25  87000]
 [    23  66000]
 [    32 120000]
 [    59  83000]
 [    24  58000]
 [    24  19000]
 [    23  82000]
 [    22  63000]
```

```
 [    31  68000]
 [    25  80000]
 [    24  27000]
 [    20  23000]
 [    33 113000]
 [    32  18000]
 [    34 112000]
 [    18  52000]
 [    22  27000]
 [    28  87000]
 [    26  17000]
 [    30  80000]
 [    39  42000]
 [    20  49000]
 [    35  88000]
 [    30  62000]
 [    31 118000]
 [    24  55000]
 [    28  85000]
 [    26  81000]
 [    35  50000]
 [    22  81000]
 [    30 116000]
 [    26  15000]
 [    29  28000]
 [    29  83000]
 [    35  44000]
 [    35  25000]
 [    28 123000]
 [    35  73000]
 [    28  37000]
 [    27  88000]
 [    28  59000]
 [    32  86000]
 [    33 149000]
 [    19  21000]
 [    21  72000]
 [    26  35000]
 [    27  89000]
 [    26  86000]
 [    38  80000]
 [    39  71000]
 [    37  71000]
 [    38  61000]
 [    37  55000]
 [    42  80000]
 [    40  57000]
 [    35  75000]
 [    36  52000]
 [    40  59000]
 [    41  59000]
 [    36  75000]
 [    37  72000]
 [    40  75000]
 [    35  53000]
 [    41  51000]
 [    39  61000]
 [    42  65000]
 [    26  32000]
 [    30  17000]
```

```
 [    26  84000]
 [    31  58000]
 [    33  31000]
 [    30  87000]
 [    21  68000]
 [    28  55000]
 [    23  63000]
 [    20  82000]
 [    30 107000]
 [    28  59000]
 [    19  25000]
 [    19  85000]
 [    18  68000]
 [    35  59000]
 [    30  89000]
 [    34  25000]
 [    24  89000]
 [    27  96000]
 [    41  30000]
 [    29  61000]
 [    20  74000]
 [    26  15000]
 [    41  45000]
 [    31  76000]
 [    36  50000]
 [    40  47000]
 [    31  15000]
 [    46  59000]
 [    29  75000]
 [    26  30000]
 [    32 135000]
 [    32 100000]
 [    25  90000]
 [    37  33000]
 [    35  38000]
 [    33  69000]
 [    18  86000]
 [    22  55000]
 [    35  71000]
 [    29 148000]
 [    29  47000]
 [    21  88000]
 [    34 115000]
 [    26 118000]
 [    34  43000]
 [    34  72000]
 [    23  28000]
 [    35  47000]
 [    25  22000]
 [    24  23000]
 [    31  34000]
 [    26  16000]
 [    31  71000]
 [    32 117000]
 [    33  43000]
 [    33  60000]
 [    31  66000]
 [    20  82000]
 [    33  41000]
 [    35  72000]
```

```
 [    28  32000]
 [    24  84000]
 [    19  26000]
 [    29  43000]
 [    19  70000]
 [    28  89000]
 [    34  43000]
 [    30  79000]
 [    20  36000]
 [    26  80000]
 [    35  22000]
 [    35  39000]
 [    49  74000]
 [    39 134000]
 [    41  71000]
 [    58 101000]
 [    47  47000]
 [    55 130000]
 [    52 114000]
 [    40 142000]
 [    46  22000]
 [    48  96000]
 [    52 150000]
 [    59  42000]
 [    35  58000]
 [    47  43000]
 [    60 108000]
 [    49  65000]
 [    40  78000]
 [    46  96000]
 [    59 143000]
 [    41  80000]
 [    35  91000]
 [    37 144000]
 [    60 102000]
 [    35  60000]
 [    37  53000]
 [    36 126000]
 [    56 133000]
 [    40  72000]
 [    42  80000]
 [    35 147000]
 [    39  42000]
 [    40 107000]
 [    49  86000]
 [    38 112000]
 [    46  79000]
 [    40  57000]
 [    37  80000]
 [    46  82000]
 [    53 143000]
 [    42 149000]
 [    38  59000]
 [    50  88000]
 [    56 104000]
 [    41  72000]
 [    51 146000]
 [    35  50000]
 [    57 122000]
 [    41  52000]
```

```
 [    35  97000]
 [    44  39000]
 [    37  52000]
 [    48 134000]
 [    37 146000]
 [    50  44000]
 [    52  90000]
 [    41  72000]
 [    40  57000]
 [    58  95000]
 [    45 131000]
 [    35  77000]
 [    36 144000]
 [    55 125000]
 [    35  72000]
 [    48  90000]
 [    42 108000]
 [    40  75000]
 [    37  74000]
 [    47 144000]
 [    40  61000]
 [    43 133000]
 [    59  76000]
 [    60  42000]
 [    39 106000]
 [    57  26000]
 [    57  74000]
 [    38  71000]
 [    49  88000]
 [    52  38000]
 [    50  36000]
 [    59  88000]
 [    35  61000]
 [    37  70000]
 [    52  21000]
 [    48 141000]
 [    37  93000]
 [    37  62000]
 [    48 138000]
 [    41  79000]
 [    37  78000]
 [    39 134000]
 [    49  89000]
 [    55  39000]
 [    37  77000]
 [    35  57000]
 [    36  63000]
 [    42  73000]
 [    43 112000]
 [    45  79000]
 [    46 117000]
 [    58  38000]
 [    48  74000]
 [    37 137000]
 [    37  79000]
 [    40  60000]
 [    42  54000]
 [    51 134000]
 [    47 113000]
 [    36 125000]
```

```
 [     38  50000]
 [     42  70000]
 [     39  96000]
 [     38  50000]
 [     49 141000]
 [     39  79000]
 [     39  75000]
 [     54 104000]
 [     35  55000]
 [     45  32000]
 [     36  60000]
 [     52 138000]
 [     53  82000]
 [     41  52000]
 [     48  30000]
 [     48 131000]
 [     41  60000]
 [     41  72000]
 [     42  75000]
 [     36 118000]
 [     47 107000]
 [     38  51000]
 [     48 119000]
 [     42  65000]
 [     40  65000]
 [     57  60000]
 [     36  54000]
 [     58 144000]
 [     35  79000]
 [     38  55000]
 [     39 122000]
 [     53 104000]
 [     35  75000]
 [     38  65000]
 [     47  51000]
 [     47 105000]
 [     41  63000]
 [     53  72000]
 [     54 108000]
 [     39  77000]
 [     38  61000]
 [     38 113000]
 [     37  75000]
 [     42  90000]
 [     37  57000]
 [     36  99000]
 [     60  34000]
 [     54  70000]
 [     41  72000]
 [     40  71000]
 [     42  54000]
 [     43 129000]
 [     53  34000]
 [     47  50000]
 [     42  79000]
 [     42 104000]
 [     59  29000]
 [     58  47000]
 [     46  88000]
 [     38  71000]
```

```
 [     54  26000]
 [     60  46000]
 [     60  83000]
 [     39  73000]
 [     59 130000]
 [     37  80000]
 [     46  32000]
 [     46  74000]
 [     42  53000]
 [     41  87000]
 [     58  23000]
 [     42  64000]
 [     48  33000]
 [     44 139000]
 [     49  28000]
 [     57  33000]
 [     56  60000]
 [     49  39000]
 [     39  71000]
 [     47  34000]
 [     48  35000]
 [     48  33000]
 [     47  23000]
 [     45  45000]
 [     60  42000]
 [     39  59000]
 [     46  41000]
 [     51  23000]
 [     50  20000]
 [     36  33000]
 [     49  36000]]
```

In [35]:

```
# 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 = 0.30, random_stat
```

In [37]:

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

In [38]:

```
# Fitting Logistic Regression to the Training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, y_train)
```

Out[38]:

```
LogisticRegression(random_state=0)
```

In [45]:

```
# Predicting the Test set results
y_pred = classifier.predict(X_test)
print(y_pred)
```

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

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

In [40]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
[[74  5]
 [11 30]]
```

In [41]:

```
#Accuray=(TN+TP)/Total+
Accuracy=(74+31)/120
Accuracy
```

Out[41]:

```
0.875
```

In [42]:

```
#Error_rate=(FN+FP)/Total
Error_rate=(5+10)/120
Error_rate
```

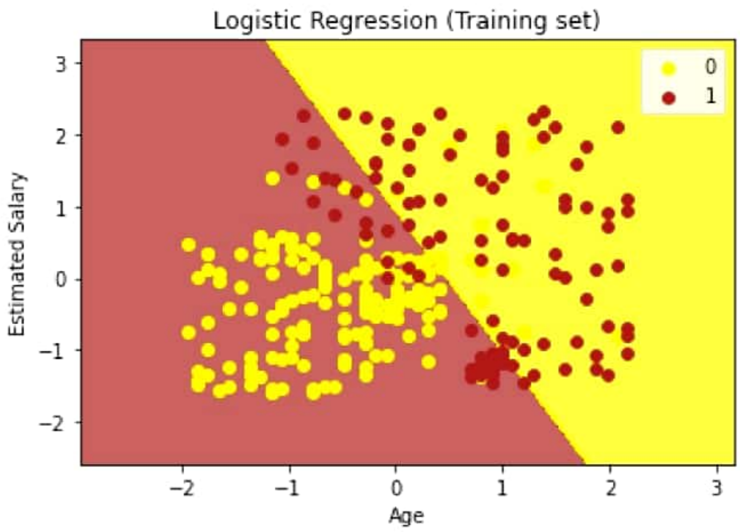
Out[42]:

```
0.125
```

In [43]:

```
# Visualising the Training set results
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].max()
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
             alpha = 0.75, cmap = ListedColormap(('brown', 'yellow')))
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(('yellow', 'brown'))(i), label = j)
plt.title('Logistic Regression (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

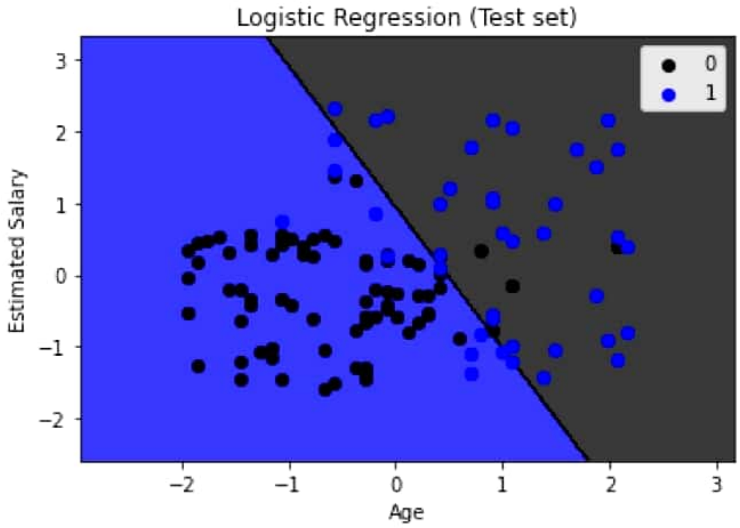
```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
s value-mapping will have precedence in case its length matches with *x* & *y*.  Please
use the *color* keyword-argument or provide a 2D array with a single row if you intend t
o specify the same RGB or RGBA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



In [44]:

```
# Visualising the Test set results
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[:, 0].max()
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max()
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X
             alpha = 0.75, cmap = ListedColormap(('blue', 'black')))
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(('black', 'blue'))(i), label = j)
plt.title('Logistic Regression (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 a
s 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 t
o 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 a
s 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 t
o specify the same RGB or RGBA value for all points.
```


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