# Assignment – Lead scoring case study

Mayank Kumar

Lakshana Govindarajan Vijaya

Amit Kumar

**PPT Index**

# Goals of the Case Study

1. Build a logistic regression model to assign a lead score between 0 and 100 to each of the leads which can be used by the company to target potential leads. A higher score would mean that the lead is hot, i.e. is most likely to convert whereas a lower score would mean that the lead is cold and will mostly not get converted.

2. There are some more problems presented by the company which your model should be able to adjust to if the company's requirement changes in the future so you will need to handle these as well. These problems are provided in a separate doc file. Please fill it based on the logistic regression model you got in the first step. Also, make sure you include this in your final PPT where you'll make recommendations.

# Demo Cell

Label Encoding:# Label encoding converts categorical data into numerical form by assigning a unique number (integer) to each category.
# For example:# Male → 0# Female → 1# One-Hot Encoding (OHE):# One-hot encoding creates binary columns (0/1) for each category and marks a 1 for the column corresponding to the category.
# For example:# Gender: Male, Female# Male → [1, 0]# Female → [0, 1]

```python
# Sample DataFrame
data = {
    "Emp_ID": [101, 102, 103, 104, 105],
    "Gender": ["Male", "Female", "Female", "Male", "Female"],
    "Department": ["IT", "HR", "Finance", "Marketing", "IT"]
}
df = pd.DataFrame(data)
df
```

4]:

| | Emp_ID | Gender | Department |
|---|---|---|---|
| **0** | 101 | Male | IT |
| **1** | 102 | Female | HR |
| **2** | 103 | Female | Finance |
| **3** | 104 | Male | Marketing |
| **4** | 105 | Female | IT |

```python
print("Original DataFrame:")
print(df)
print("\nDataFrame with Label Encoding:")
print(df[['Emp_ID', 'Gender', 'Gender_LabelEncoded', 'Department_LabelEncoded']])
print("\nDataFrame with One-Hot Encoding:")
print(df_ohe)
```

```
Original DataFrame:
   Emp_ID  Gender Department  Gender_LabelEncoded  Department_LabelEncoded
0     101    Male         IT                    1                        2
1     102  Female         HR                    0                        1
2     103  Female    Finance                    0                        0
3     104    Male  Marketing                    1                        3
4     105  Female         IT                    0                        2

DataFrame with Label Encoding:
   Emp_ID  Gender  Gender_LabelEncoded  Department_LabelEncoded
0     101    Male                    1                        2
1     102  Female                    0                        1
2     103  Female                    0                        0
3     104    Male                    1                        3
4     105  Female                    0                        2

DataFrame with One-Hot Encoding:
   Emp_ID  Gender Department  Gender_LabelEncoded  Department_LabelEncoded  \
0     101    Male         IT                    1                        2
1     102  Female         HR                    0                        1
2     103  Female    Finance                    0                        0
3     104    Male  Marketing                    1                        3
4     105  Female         IT                    0                        2

   Gender_Female  Gender_Male  Dept_Finance  Dept_HR  Dept_IT  Dept_Marketing
0              0            1             0        0        1               0
1              1            0             0        1        0               0
2              1            0             1        0        0               0
3              0            1             0        0        0               1
4              1            0             0        0        1               0
```

# Dummy variable creation

[37]:
```python
# Check the columns which are of type 'object'

temp = leads.loc[:, leads.dtypes == 'object']
temp.columns
```

[37]:
```
Index(['Lead Origin', 'Lead Source', 'Do Not Email', 'Last Activity',
       'Specialization', 'What is your current occupation',
       'A free copy of Mastering The Interview', 'Last Notable Activity'],
      dtype='object')
```

[17]:
```python
# Demo Cell
df = pd.DataFrame({'What is your area of interest?': ['Select', 'Python', 'Select']})
df
```

[17]:

| | What is your area of interest? |
|---|---|
| **0** | Select |
| **1** | Python |
| **2** | Select |

[ ]:
```python
# What is your area of interest?
# Unique Values: ["Python", "Select"]
```

[18]:
```python
df = pd.get_dummies(df, dtype=int)
df
```

[18]:

| | What is your area of interest?_Python | What is your area of interest?_Select |
|---|---|---|
| **0** | 0 | 1 |
| **1** | 1 | 0 |
| **2** | 0 | 1 |

[19]:
```python
# Drop column "What is your area of interest?_Select" in df
df.drop('What is your area of interest?_Select', axis=1, inplace=True)
df
```

# Dummy variable creation

[41]:
```
# Create dummy variables using the 'get_dummies' command
dummy = pd.get_dummies(leads[['Lead Origin', 'Lead Source', 'Do Not Email', 'Last Activity',
                              'What is your current occupation','A free copy of Mastering The Interview',
                              'Last Notable Activity']], drop_first=True)

# Add the results to the master dataframe
leads = pd.concat([leads, dummy], axis=1)
```

[42]:
```
# Creating dummy variable separately for the variable 'Specialization' since it has the level 'Select'
# which is useless so we
# drop that level by specifying it explicitly

dummy_spl = pd.get_dummies(leads['Specialization'], prefix = 'Specialization')
dummy_spl = dummy_spl.drop(['Specialization_Select'], 1)
leads = pd.concat([leads, dummy_spl], axis = 1)
```

[43]:
```
# Drop the variables for which the dummy variables have been created

leads = leads.drop(['Lead Origin', 'Lead Source', 'Do Not Email', 'Last Activity',
                   'Specialization', 'What is your current occupation',
                   'A free copy of Mastering The Interview', 'Last Notable Activity'], 1)
```

[44]:
```
# Let's take a look at the dataset again

leads.head()
```

[44]:

| | Converted | TotalVisits | Total Time Spent on Website | Page Views Per Visit | Lead Origin_Landing Page Submission | Lead Origin_Lead Add Form | Lead Origin_Lead Import | Lead Source_Direct Traffic | Lead Source_Facebook | Lead Source_Google | ... | Specialization_IT Projects Management | Specializatio |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.0 | 0 | 0.0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| **1** | 0 | 5.0 | 674 | 2.5 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| **2** | 1 | 2.0 | 1532 | 2.0 | 1 | 0 | 0 | 1 | 0 | 0 | ... | 0 | |
| **3** | 0 | 1.0 | 305 | 1.0 | 1 | 0 | 0 | 1 | 0 | 0 | ... | 0 | |
| **4** | 1 | 2.0 | 1428 | 1.0 | 1 | 0 | 0 | 0 | 0 | 1 | ... | 0 | |

5 rows × 75 columns

# Test-Train Split

[45]:
```
# Import the required library

from sklearn.model_selection import train_test_split
```

[46]:
```
# Put all the feature variables in X

X = leads.drop('Converted', axis=1)
```

[46]:

| | TotalVisits | Total Time Spent on Website | Page Views Per Visit | Lead Origin_Landing Page Submission | Lead Origin_Lead Add Form | Lead Origin_Lead Import | Lead Source_Direct Traffic | Lead Source_Facebook | Lead Source_Google | Lead Source_Live Chat | ... | Specialization_IT Projects Management | Specializati |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0.0 | 0 | 0.0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| **1** | 5.0 | 674 | 2.5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| **2** | 2.0 | 1532 | 2.0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... | 0 | |
| **3** | 1.0 | 305 | 1.0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... | 0 | |
| **4** | 2.0 | 1428 | 1.0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | ... | 0 | |

5 rows × 74 columns

[47]:
```
# Put the target variable in y

y = leads['Converted']
```

[47]:
```
0    0
1    0
2    1
3    0
4    1
Name: Converted, dtype: int64
```

[48]:
```
# Split the dataset into 70% train and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=100)
```

## Scaling

Now there are a few numeric variables present in the dataset which have different scales. So let's go ahead and scale these variables.

[49]:

```
# Import MinMax scaler

from sklearn.preprocessing import MinMaxScaler
```

[50]:

```
# Scale the three numeric features present in the dataset

scaler = MinMaxScaler()
# TotalVisits, Total Time Spent on Website, Page Views Per Visit
X_train[['TotalVisits', 'Total Time Spent on Website', 'Page Views Per Visit']] = scaler.fit_transform(X_train[['TotalVisits', 'Total Time Spent on Websi

X_train.head()
```

[50]:

| | TotalVisits | Total Time Spent on Website | Page Views Per Visit | Lead Origin_Landing Page Submission | Lead Origin_Lead Add Form | Lead Origin_Lead Import | Lead Source_Direct Traffic | Lead Source_Facebook | Lead Source_Google | Lead Source_Live Chat | ... | Specialization_IT Projects Management | Special |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **8003** | 0.015936 | 0.029489 | 0.125 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... | 1 | |
| **218** | 0.015936 | 0.082306 | 0.250 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... | 0 | |
| **4171** | 0.023904 | 0.034331 | 0.375 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | ... | 0 | |
| **4037** | 0.000000 | 0.000000 | 0.000 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |
| **3660** | 0.000000 | 0.000000 | 0.000 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | ... | 0 | |

5 rows × 74 columns

# Correlation

## Looking at the correlations

Let's now look at the correlations. Since the number of variables are pretty high, it's better that we look at the table instead of plotting a heatmap

[51]:
```
# Looking at the correlation table

leads.corr()
```

[51]:

| | Converted | TotalVisits | Total Time Spent on Website | Page Views Per Visit | Lead Origin_Landing Page Submission | Lead Origin_Lead Add Form | Lead Origin_Lead Import | Lead Source_Direct Traffic | Lead Source_Facebook | Lead Source_Google | ... | Speci M |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Converted** | 1.000000 | 0.005651 | 0.313338 | -0.063362 | -0.117563 | 0.288666 | -0.019269 | -0.133600 | -0.021207 | 0.020205 | ... | |
| **TotalVisits** | 0.005651 | 1.000000 | 0.202551 | 0.489039 | 0.267954 | -0.208375 | -0.043000 | 0.075252 | -0.042052 | 0.085306 | ... | |
| **Total Time Spent on Website** | 0.313338 | 0.202551 | 1.000000 | 0.303870 | 0.275606 | -0.249493 | -0.061429 | 0.114088 | -0.060945 | 0.227496 | ... | |
| **Page Views Per Visit** | -0.063362 | 0.489039 | 0.303870 | 1.000000 | 0.458168 | -0.340185 | -0.065739 | 0.109785 | -0.062896 | 0.183735 | ... | |
| **Lead Origin_Landing Page Submission** | -0.117563 | 0.267954 | 0.275606 | 0.458168 | 1.000000 | -0.363764 | -0.074917 | 0.508857 | -0.071507 | 0.067225 | ... | |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | |
| **Specialization_Retail Management** | -0.018603 | 0.014223 | 0.024919 | 0.026099 | 0.070983 | -0.025339 | -0.007261 | 0.022168 | -0.007395 | 0.021190 | ... | |
| **Specialization_Rural and Agribusiness** | 0.006964 | 0.068015 | 0.018767 | 0.027465 | 0.050077 | -0.018872 | -0.006251 | 0.021596 | -0.006366 | -0.037642 | ... | |
| **Specialization_Services Excellence** | -0.005142 | 0.015114 | 0.003203 | 0.015230 | 0.039433 | -0.011155 | -0.004093 | 0.053189 | -0.004169 | -0.027058 | ... | |
| **Specialization_Supply Chain Management** | 0.005785 | 0.063383 | 0.045386 | 0.052972 | 0.111610 | -0.035065 | -0.001963 | 0.093536 | -0.002431 | -0.027074 | ... | |
| **Specialization_Travel and Tourism** | -0.011762 | 0.064384 | 0.037867 | 0.111284 | 0.094875 | -0.045397 | -0.010092 | 0.002757 | -0.010278 | -0.053104 | ... | |

75 rows × 75 columns

# Model Building

## Step 2: Model Building

Let's now move to model building. As you can see that there are a lot of variables present in the dataset which we cannot deal with. So the best way to approach this is to select a small set of features from this pool of variables using RFE.

[52]:
```
# Import 'LogisticRegression' and create a LogisticRegression object
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
```

[53]:
```
# Import RFE and select 15 variables
from sklearn.feature_selection import RFE
# running RFE with 15 variables as output
rfe = RFE(estimator=logreg, n_features_to_select=15)
rfe = rfe.fit(X_train, y_train)
```

[54]:
```
# Let's take a look at which features have been selected by RFE

list(zip(X_train.columns, rfe.support_, rfe.ranking_))
```

[54]:
```
[('TotalVisits', True, 1),
 ('Total Time Spent on Website', True, 1),
 ('Page Views Per Visit', False, 23),
 ('Lead Origin_Landing Page Submission', False, 8),
 ('Lead Origin_Lead Add Form', True, 1),
 ('Lead Origin_Lead Import', False, 52),
 ('Lead Source_Direct Traffic', False, 24),
 ('Lead Source_Facebook', False, 51),
 ('Lead Source_Google', False, 36),
 ('Lead Source_Live Chat', False, 44),
 ('Lead Source_Olark Chat', True, 1),
 ('Lead Source_Organic Search', False, 35),
 ('Lead Source_Pay per Click Ads', False, 43),
 ('Lead Source_Press_Release', False, 53),
 ('Lead Source_Reference', True, 1),
 ('Lead Source_Referral Sites', False, 37),
 ('Lead Source_Social Media', False, 58),
 ('Lead Source_WeLearn', False, 42),
```

[55]:
```
# Put all the columns selected by RFE in the variable 'col'
col = X_train.columns[rfe.support_]
```

Now you have all the variables selected by RFE and since we care about the statistics part, i.e. the p-values and the VIFs, let's use these variables to create a logistic regression model using statsmodels.

# Model Building

[56]:
```
# Select only the columns selected by RFE
X_train = X_train[col]
```

[57]:
```
# Import statsmodels

import statsmodels.api as sm
```

[58]:
```
# Fit a logistic Regression model on X_train after adding a constant and output the summary

X_train_sm = sm.add_constant(X_train)
logm2 = sm.GLM(y_train, X_train_sm, family = sm.families.Binomial())
res = logm2.fit()
res.summary()
```

[58]:

Generalized Linear Model Regression Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | Converted | **No. Observations:** | 4461 |
| **Model:** | GLM | **Df Residuals:** | 4445 |
| **Model Family:** | Binomial | **Df Model:** | 15 |
| **Link Function:** | logit | **Scale:** | 1.0000 |
| **Method:** | IRLS | **Log-Likelihood:** | -2072.8 |
| **Date:** | Fri, 03 Dec 2021 | **Deviance:** | 4145.5 |
| **Time:** | 12:04:45 | **Pearson chi2:** | 4.84e+03 |
| **No. Iterations:** | 22 | | |
| **Covariance Type:** | nonrobust | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **const** | -1.0061 | 0.600 | -1.677 | 0.094 | -2.182 | 0.170 |
| **TotalVisits** | 11.3439 | 2.682 | 4.230 | 0.000 | 6.088 | 16.600 |
| **Total Time Spent on Website** | 4.4312 | 0.185 | 23.924 | 0.000 | 4.068 | 4.794 |
| **Lead Origin_Lead Add Form** | 2.9483 | 1.191 | 2.475 | 0.013 | 0.614 | 5.283 |
| **Lead Source_Olark Chat** | 1.4584 | 0.122 | 11.962 | 0.000 | 1.219 | 1.697 |
| **Lead Source_Reference** | 1.2994 | 1.214 | 1.070 | 0.285 | -1.080 | 3.679 |
| **Lead Source_Welingak Website** | 3.4159 | 1.558 | 2.192 | 0.028 | 0.362 | 6.470 |

# Model Building

[59]:
```python
# Import 'variance_inflation_factor'

from statsmodels.stats.outliers_influence import variance_inflation_factor
```

[60]:
```python
# Make a VIF dataframe for all the variables present

vif = pd.DataFrame()
vif['Features'] = X_train.columns
vif['VIF'] = [variance_inflation_factor(X_train.values, i) for i in range(X_train.shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

[60]:

| | Features | VIF |
|---|---|---|
| 2 | Lead Origin_Lead Add Form | 84.19 |
| 4 | Lead Source_Reference | 65.18 |
| 5 | Lead Source_Welingak Website | 20.03 |
| 11 | What is your current occupation_Unemployed | 3.65 |
| 7 | Last Activity_Had a Phone Conversation | 2.44 |
| 13 | Last Notable Activity_Had a Phone Conversation | 2.43 |
| 1 | Total Time Spent on Website | 2.38 |
| 0 | TotalVisits | 1.62 |
| 8 | Last Activity_SMS Sent | 1.59 |

VIFs seem to be in a decent range except for three variables.

Let's first drop the variable `Lead Source_Reference` since it has a high p-value as well as a high VIF.

[61]:
```python
# Let's first drop the variable `Lead Source_Reference` since it has a high p-value as well as a high VIF.
X_train.drop('Lead Source_Reference', axis=1, inplace=True)
```

[62]:
```python
# Refit the model with the new set of features

logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomial())
logm1.fit().summary()
```

# Model Evaluation

Now, both the p-values and VIFs seem decent enough for all the variables. So let's go ahead and make predictions using this final set of features.

[71]:
```
# Use 'predict' to predict the probabilities on the train set

y_train_pred = res.predict(sm.add_constant(X_train))
y_train_pred[:10]
```

[71]:
```
8003    0.300117
218     0.142002
4171    0.127629
4037    0.291558
3660    0.954795
207     0.194426
2044    0.178073
6411    0.949460
6498    0.075995
2085    0.982316
dtype: float64
```

[72]:
```
# Reshaping it into an array

y_train_pred = y_train_pred.values.reshape(-1)
y_train_pred[:10]
```

[72]:
```
array([0.30011695, 0.14200165, 0.12762885, 0.29155814, 0.95479546,
       0.19442563, 0.17807328, 0.94946006, 0.07599465, 0.98231619])
```

### Creating a dataframe with the actual conversion flag and the predicted probabilities

[73]:
```
# Create a new dataframe containing the actual conversion flag and the probabilities predicted by the model

y_train_pred_final = pd.DataFrame({'Converted':y_train.values, 'Conversion_Prob':y_train_pred})
y_train_pred_final.head()
```

[73]:

| | Converted | Conversion_Prob |
|---|---|---|
| **0** | 0 | 0.300117 |
| **1** | 0 | 0.142002 |
| **2** | 1 | 0.127629 |
| **3** | 1 | 0.291558 |
| **4** | 1 | 0.954795 |

## **Model Evaluation-** Creating a dataframe with the actual conversion flag and the predicted probabilities

[73]:
```
# Create a new dataframe containing the actual conversion flag and the probabilities predicted by the model

y_train_pred_final = pd.DataFrame({'Converted':y_train.values, 'Conversion_Prob':y_train_pred})
y_train_pred_final.head()
```

[73]:

| | Converted | Conversion_Prob |
|---|---|---|
| **0** | 0 | 0.300117 |
| **1** | 0 | 0.142002 |
| **2** | 1 | 0.127629 |
| **3** | 1 | 0.291558 |
| **4** | 1 | 0.954795 |

### Creating new column 'Predicted' with 1 if Paid_Prob > 0.5 else 0

[74]:
```
# Create a new column 'Predicted' with 1 if Conversion_Prob > 0.5 else 0
y_train_pred_final['Predicted'] = y_train_pred_final.Conversion_Prob.map(lambda x: 1 if x > 0.5 else 0)

# Let's take a look at the dataframe
y_train_pred_final.head()
```

[74]:

| | Converted | Conversion_Prob | Predicted |
|---|---|---|---|
| **0** | 0 | 0.300117 | 0 |
| **1** | 0 | 0.142002 | 0 |
| **2** | 1 | 0.127629 | 0 |
| **3** | 1 | 0.291558 | 0 |
| **4** | 1 | 0.954795 | 1 |

Now that you have the probabilities and have also made conversion predictions using them, it's time to evaluate the model.

[75]:
```
# Import metrics from sklearn for evaluation

from sklearn import metrics
```

[76]:
```
# Create confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.Converted, y_train_pred_final.Predicted)
```

# **Model Evaluation-** Creating a dataframe with the actual conversion flag and the predicted probabilities

```
0.7806124187401926
```

[79]:
```
# Let's evaluate the other metrics as well

TP = confusion[1,1] # true positive
TN = confusion[0,0] # true negatives
FP = confusion[0,1] # false positives
FN = confusion[1,0] # false negatives
```

[80]:
```
# Calculate the sensitivity

TP/(TP+FN)
```

[80]: 0.739413680781759

[81]:
```
# Calculate the specificity

TN/(TN+FP)
```

[81]: 0.8343425605536332

```
### Finding the Optimal Cutoff

Now 0.5 was just arbitrary to loosely check the model performace. But in order to get good results, you need to optimise the threshold. So first let's
plot an ROC curve to see what AUC we get.
```

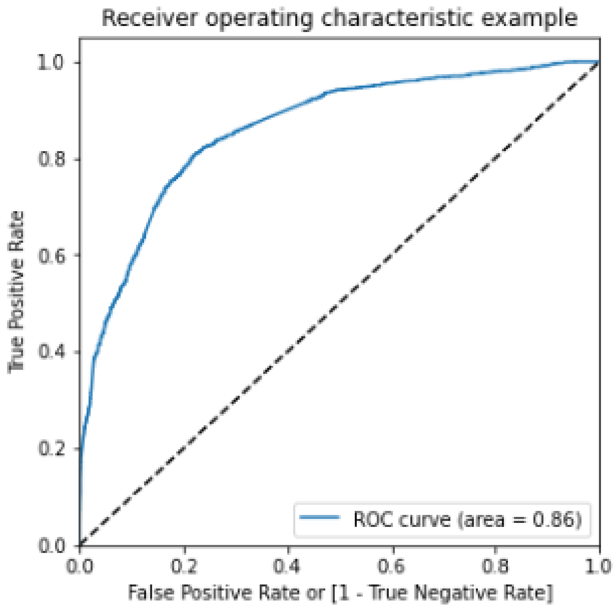
[82]:
```
# ROC function

def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.ro   probs )
    plt.figure(figsize=(5, 5))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return None
```

## Model Evaluation

```
[83]: fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Converted,
                                          y_train_pred_final.Conversion_Prob, drop_intermediate = False )
```

```
[84]: # Import matplotlib to plot the ROC curve

import matplotlib.pyplot as plt
```

```
[85]: # Call the ROC function

draw_roc(y_train_pred_final.Converted, y_train_pred_final.Conversion_Prob)
```



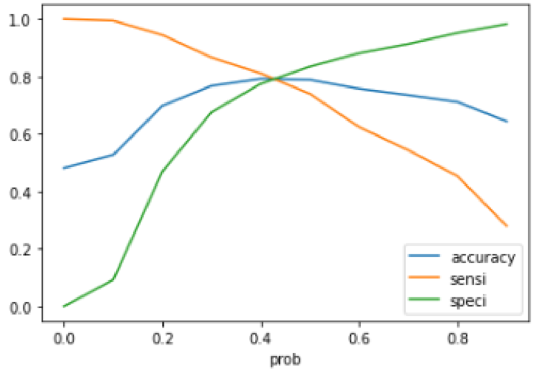

The area under the curve of the ROC is 0.86 which is quite good. So we seem to have a good model. Let's also check the sensitivity and specificity tradeoff to find the optimal cutoff point.

# Model Evaluation

```
from sklearn.metrics import confusion_matrix

# TP = confusion[1,1] # true positive
# TN = confusion[0,0] # true negatives
# FP = confusion[0,1] # false positives
# FN = confusion[1,0] # false negatives

num = [0.0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
for i in num:
    cm1 = metrics.confusion_matrix(y_train_pred_final.Converted, y_train_pred_final[i] )
    total1=sum(sum(cm1))
    accuracy = (cm1[0,0]+cm1[1,1])/total1

    speci = cm1[0,0]/(cm1[0,0]+cm1[0,1])
    sensi = cm1[1,1]/(cm1[1,0]+cm1[1,1])
    cutoff_df.loc[i] =[ i ,accuracy,sensi,speci]
print(cutoff_df)
```

```
     prob  accuracy     sensi     speci
0.0   0.0  0.481731  1.000000  0.000000
0.1   0.1  0.527012  0.994416  0.092561
0.2   0.2  0.698274  0.944160  0.469723
0.3   0.3  0.767541  0.865984  0.676038
0.4   0.4  0.791975  0.810610  0.774654
0.5   0.5  0.788612  0.739414  0.834343
0.6   0.6  0.757229  0.624011  0.881055
0.7   0.7  0.735037  0.543509  0.913062
0.8   0.8  0.711500  0.453234  0.951557
0.9   0.9  0.644026  0.279665  0.982699
```

[88]:
```
# Let's plot it as well

cutoff_df.plot.line(x='prob', y=['accuracy','sensi','speci'])
plt.show()
```



# Making Predictions on the Test Set

[95]:
```
# Scale the test set as well using just 'transform'

X_test[['TotalVisits', 'Page Views Per Visit', 'Total Time Spent on Website']] =
        scaler.transform(X_test[['TotalVisits', 'Page Views Per Visit', 'Total Time Spent on Website']])
```

[96]:
```
# Select the columns in X_train for X_test as well

X_test = X_test[col]
X_test.head()
```

[96]:

| | TotalVisits | Total Time Spent on Website | Lead Origin_Lead Add Form | Lead Source_Olark Chat | Lead Source_Reference | Lead Source_Welingak Website | Do Not Email_Yes | Last Activity_Had a Phone Conversation | Last Activity_SMS Sent | What is your current occupation_Housewife | Wha occupation_ |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 4771 | 0.000000 | 0.000000 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | |
| 6122 | 0.027888 | 0.029049 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 9202 | 0.015936 | 0.416813 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 6570 | 0.011952 | 0.378961 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | |
| 2668 | 0.031873 | 0.395246 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |

[97]:
```
# Add a constant to X_test

X_test_sm = sm.add_constant(X_test[col])
```

[98]:
```
# Check X_test_sm

X_test_sm
```

[98]:

| | const | TotalVisits | Total Time Spent on Website | Lead Origin_Lead Add Form | Lead Source_Olark Chat | Lead Source_Reference | Lead Source_Welingak Website | Do Not Email_Yes | Last Activity_Had a Phone Conversation | Last Activity_SMS Sent | What is your current occupation_Housewife | occ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 4771 | 1.0 | 0.000000 | 0.000000 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | |
| 6122 | 1.0 | 0.027888 | 0.029049 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| 9202 | 1.0 | 0.015936 | 0.416813 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |
| 6570 | 1.0 | 0.011952 | 0.378961 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | |
| 2668 | 1.0 | 0.031873 | 0.395246 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | |

# Making Predictions on the Test Set

[110]:
```python
# Make predictions on the test set using 0.45 as the cutoff

y_pred_final['final_predicted'] = y_pred_final.Conversion_Prob.map(lambda x: 1 if x > 0.42 else 0)
```

[111]:
```python
# Check y_pred_final

y_pred_final.head()
```

[111]:

| | Converted | Conversion_Prob | final_predicted |
|---|---|---|---|
| **0** | 1 | 0.996296 | 1 |
| **1** | 0 | 0.129992 | 0 |
| **2** | 0 | 0.703937 | 1 |
| **3** | 1 | 0.299564 | 0 |
| **4** | 1 | 0.720796 | 1 |

[112]:
```python
# Let's check the overall accuracy

metrics.accuracy_score(y_pred_final['Converted'], y_pred_final.final_predicted)
```

[112]: 0.7845188284518828

[113]:
```python
confusion2 = metrics.confusion_matrix(y_pred_final['Converted'], y_pred_final.final_predicted )
confusion2
```

[113]:
```
array([[786, 210],
       [202, 714]])
```

[114]:
```python
TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

[115]:
```python
# Calculate sensitivity
TP / float(TP+FN)
```

[115]: 0.7794759825327511

[116]:
```python
# Calculate specificity
TN / float(TN+FP)
```

[116]: 0.7891566265060241

# Precision-Recall View

[117]:
```
#Looking at the confusion matrix again
```

[118]:
```
confusion = metrics.confusion_matrix(y_train_pred_final.Converted, y_train_pred_final.Predicted )
confusion
```

[118]:
```
array([[1929,  383],
       [ 560, 1589]])
```

```
##### Precision
TP / TP + FP
```

[119]:
```
confusion[1,1]/(confusion[0,1]+confusion[1,1])
```

[119]:
```
0.8057809330628803
```

```
##### Recall
TP / TP + FN
```

[120]:
```
confusion[1,1]/(confusion[1,0]+confusion[1,1])
```

[120]:
```
0.739413680781759
```

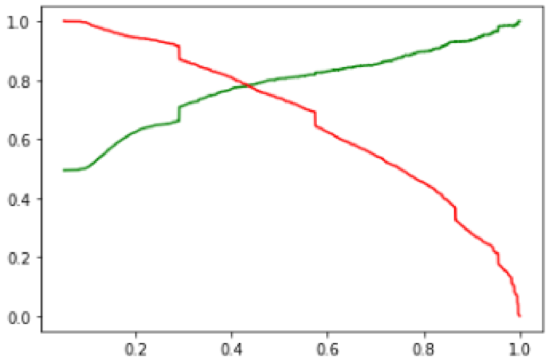
```
### Precision and recall tradeoff
```

[121]:
```
from sklearn.metrics import precision_recall_curve
```

[122]:
```
y_train_pred_final.Converted, y_train_pred_final.Predicted
```

[122]:
```
(0       0
 1       0
 2       1
 3       1
 4       1
        ..
 4456    1
 4457    0
 4458    0
 4459    0
 4460    0
 Name: Converted, Length: 4461, dtype: int64,
 0       0
 1       0
 2       0
 3       0
 4       1
        ..
 4456    1
 4457    1
 4458    1
 4459    0
```

# Precision-Recall View




```
[125]: y_train_pred_final['final_predicted'] = y_train_pred_final.Conversion_Prob.map(lambda x: 1 if x > 0.44 else 0)

y_train_pred_final.head()
```

[125]:

| | Converted | Conversion_Prob | Predicted | 0.0 | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | final_predicted |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 0 | 0.300117 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **1** | 0 | 0.142002 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **2** | 1 | 0.127629 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **3** | 1 | 0.291558 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| **4** | 1 | 0.954795 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

```
[126]: # Let's check the accuracy now

metrics.accuracy_score(y_train_pred_final.Converted, y_train_pred_final.final_predicted)

[126]: 0.7895090786819099
```

```
[127]: # Let's create the confusion matrix once again

confusion2 = metrics.confusion_matrix(y_train_pred_final.Converted, y_train_pred_final.final_predicted )
confusion2

[127]: array([[1852,  460],
       [ 479, 1670]])
```

## Predictions on the Test Set

```
[142]: # Check y_pred_final

y_pred_final.head()
```

[142]:

| | Converted | Conversion_Prob | final_predicted |
|---|---|---|---|
| **0** | 1 | 0.996296 | 1 |
| **1** | 0 | 0.129992 | 0 |
| **2** | 0 | 0.703937 | 1 |
| **3** | 1 | 0.299564 | 0 |
| **4** | 1 | 0.720796 | 1 |

```
[143]: # Let's check the overall accuracy

metrics.accuracy_score(y_pred_final['Converted'], y_pred_final.final_predicted)
```

```
[143]: 0.7866108786610879
```

```
[144]: confusion2 = metrics.confusion_matrix(y_pred_final['Converted'], y_pred_final.final_predicted )
confusion2
```

```
[144]: array([[801, 195],
       [213, 703]])
```

```
[145]: TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

```
[146]: # Calculate Precision

TP/(TP+FP)
```

```
[146]: 0.7828507795100222
```

```
[147]: # Calculate Recall

TP/(TP+FN)
```

```
[147]: 0.767467248908297
```

## Conclusion Summary

By implementing the structured approach outlined, X Education can significantly enhance its lead conversion process. Here’s a recap of how each step contributes to achieving the company's goals:

**Key Benefits of the Structured Approach**

1. **Data-Driven Insights**: By thoroughly understanding and preprocessing the data, the company can ensure that the model is built on a solid foundation. This leads to more accurate predictions and better identification of potential leads.
2. **Targeted Marketing Efforts**: With the logistic regression model assigning lead scores, the sales team can focus their efforts on leads that are statistically more likely to convert. This targeted approach increases efficiency and optimizes resource allocation.
3. **Improved Conversion Rates**: By concentrating on "Hot Leads," the company can aim for a higher conversion rate, potentially reaching the target of 80%. This not only boosts revenue but also enhances the overall effectiveness of the sales process.
4. **Adaptability to Change**: The model's design allows for adjustments based on changing business needs or market conditions. This flexibility ensures that X Education can remain competitive and responsive to new challenges.
5. **Continuous Improvement**: By establishing a feedback loop and regularly updating the model with new data, X Education can refine its lead scoring process over time, leading to sustained improvements in conversion rates.
6. **Strategic Recommendations**: The insights gained from the model can inform broader marketing and

## Conclusion Summary

1. **Strategic Recommendations**: The insights gained from the model can inform broader marketing and sales strategies, helping the company to better understand its customer base and tailor its offerings accordingly.

To ensure successful implementation, X Education should consider the following next steps:

- **Pilot Testing**: Before a full rollout, conduct a pilot test of the lead scoring model with a small segment of leads to evaluate its effectiveness in real-world scenarios.
- **Training for Sales Team**: Provide training for the sales team on how to interpret lead scores and adjust their outreach strategies accordingly.
- **Monitoring and Evaluation**: Set up a system for ongoing monitoring of the model's performance and the actual conversion rates to ensure that the desired outcomes are being achieved.
- **Feedback Mechanism**: Create a feedback mechanism for the sales team to report back on lead quality and conversion outcomes, which can be used to further refine the model.

By following these steps and leveraging the insights gained from the model, X Education can not only improve its lead conversion rates but also build a more robust and data-driven sales strategy that aligns with its business objectives.