

# **Lead scoring case study**

**PPT/PDF describing the analysis:**

Group members: Sushmita, Vaibhavi, Taha

# Problem statement

- An X Education need help to select the most promising leads, i.e. the leads that are most likely to convert into paying customers. The company requires us to build a model wherein you need to assign a lead score to each of the leads such that the customers with higher lead score have a higher conversion chance and the customers with lower lead score have a lower conversion chance. The CEO, in particular, has given a ballpark of the target lead conversion rate to be around 80%.

# Goals and Objectives

- There are quite a few goals for this case study.

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.

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.

# Solution steps:

- Data importing, cleaning and processing.
- EDA: Univariate, Bi-variate, multivariate analyses.
- Dummy variable and splitting data into train-test.
- Building the model (logistic regression)
- Manual feature selection using VIF
- Creating predictions after fine tuning
- Evaluating model, optimising cut-off, getting results (metrics) on test set and CONCLUSIONS.

# Data processing

- Numpy, pandas, matplotlib, seaborn were used.
- Data cleaning -> replacing empty choices with nan, consistent casing, dropping columns with high null %, imputing null values.

```
[27] # Replace null values in Country by 'unknown'  
df2['Country'] = df2['Country'].fillna('unknown')  
# see unique value of the column after replacing value  
df2['Country'].value_counts()  
  
... Country  
india 6491  
unknown 2301  
united states 69  
united arab emirates 53  
singapore 24  
saudi arabia 21  
united kingdom 15  
australia 13  
qatar 10
```

```
[32] ... # Rechecking the percentage of missing values  
round(100*(df2.isnull().sum()/len(df2.index)), 2)  
  
Prospect ID 0.00  
Lead Origin 0.00  
Lead Source 0.39  
Do Not Email 0.00  
Do Not Call 0.00  
Converted 0.00  
TotalVisits 1.48  
Total Time Spent on Website 0.00  
Page Views Per Visit 1.48  
Last Activity 1.11  
Country 26.63  
Specialization 36.58  
What is your current occupation 29.11  
What matters most to you in choosing a course 29.32
```

```
[3] # Group values with a frequency of less than 10 into "others" to simplify the values in the Country column and avoid unnecessary complexity when build  
country_count = df2['Country'].value_counts()  
top_country_count = country_count[country_count >= 10].index  
# replace rare values by 'others'  
df2['Country'] = df2['Country'].apply(lambda x: x if x in top_country_count else 'Other')  
# see unique value in Tags column after replacing rare values by 'others' value  
df2['Country'].value_counts()  
  
Country  
india 6491  
unknown 2301  
Other 77  
united states 69
```

Python

# EDA

## Univariate analysis

- Analysing counts of variables.
- Viewing histplots of variables.

```
plt.figure(figsize = (20,40))

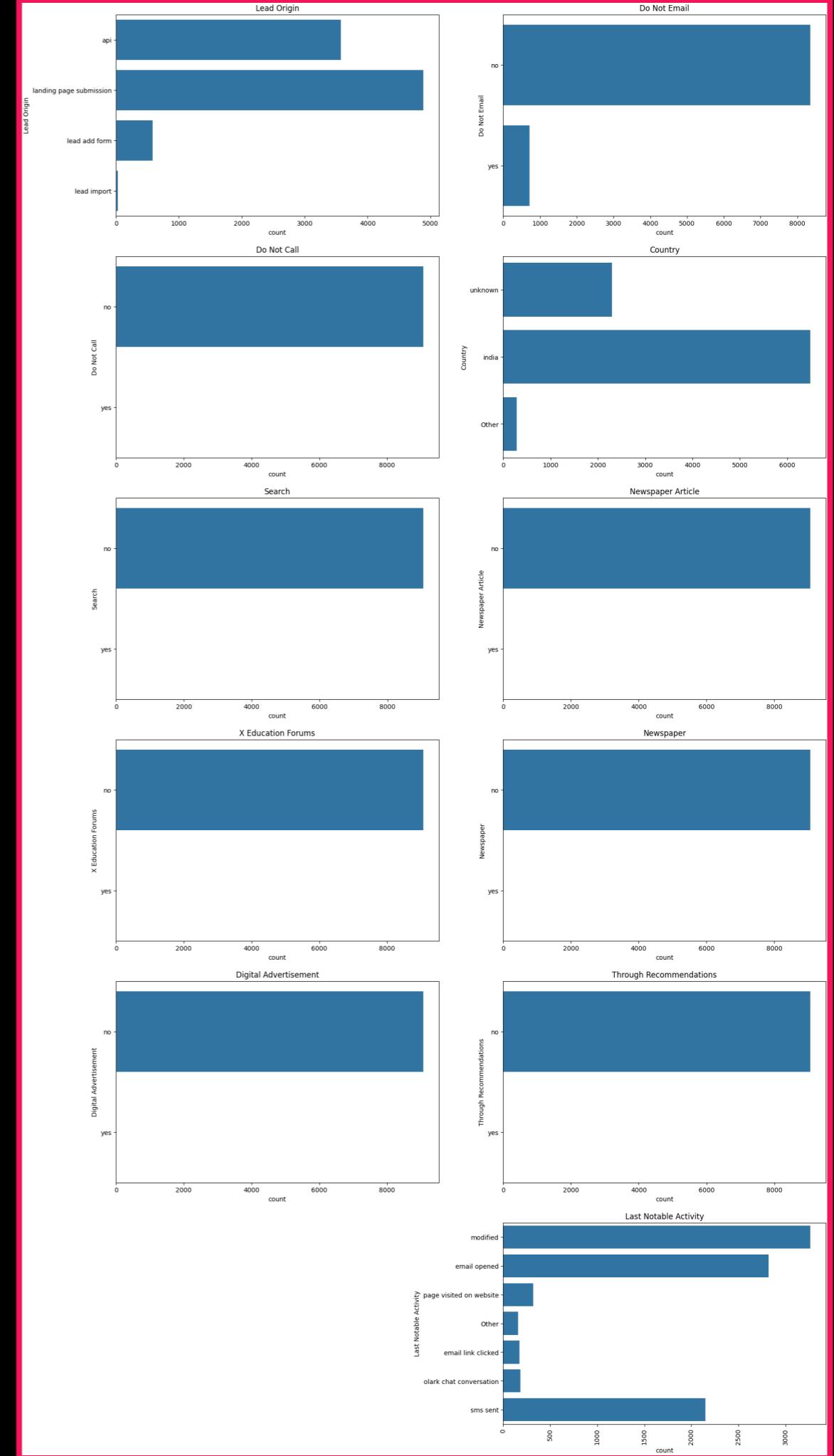
plt.subplot(6,2,1)
sns.countplot(df_final['Lead Origin'])
plt.title('Lead Origin')

plt.subplot(6,2,2)
sns.countplot(df_final['Do Not Email'])
plt.title('Do Not Email')

plt.subplot(6,2,3)
sns.countplot(df_final['Do Not Call'])
plt.title('Do Not Call')

plt.subplot(6,2,4)
sns.countplot(df_final['Country'])
plt.title('Country')

plt.subplot(6,2,5)
sns.countplot(df_final['Search'])
plt.title('Search')
```



```

import matplotlib.pyplot as plt
import seaborn as sns

categorical_columns = df_final.select_dtypes(include=['object', 'category']).columns.tolist()

plt.figure(figsize=(28, 30))

# Loop through categorical columns and plot
for idx, col in enumerate(categorical_columns, start=1):
    plt.subplot((len(categorical_columns) + 1) // 2, 2, idx) # Adjust rows dynamically
    sns.countplot(x=col, hue='Converted', data=df_final)
    plt.yscale('log')
    plt.xticks(rotation=90)

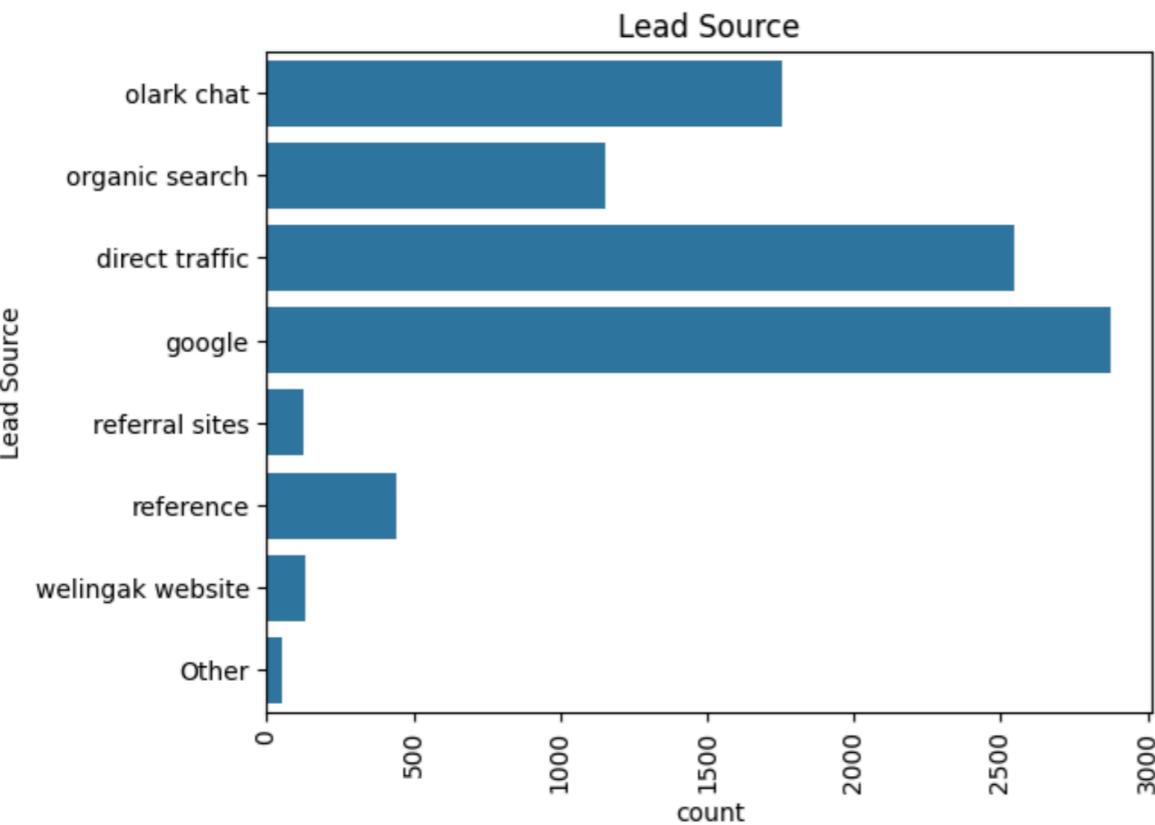
plt.subplots_adjust(hspace=0.5)
plt.tight_layout() # Prevent overlapping
plt.show()

```

```

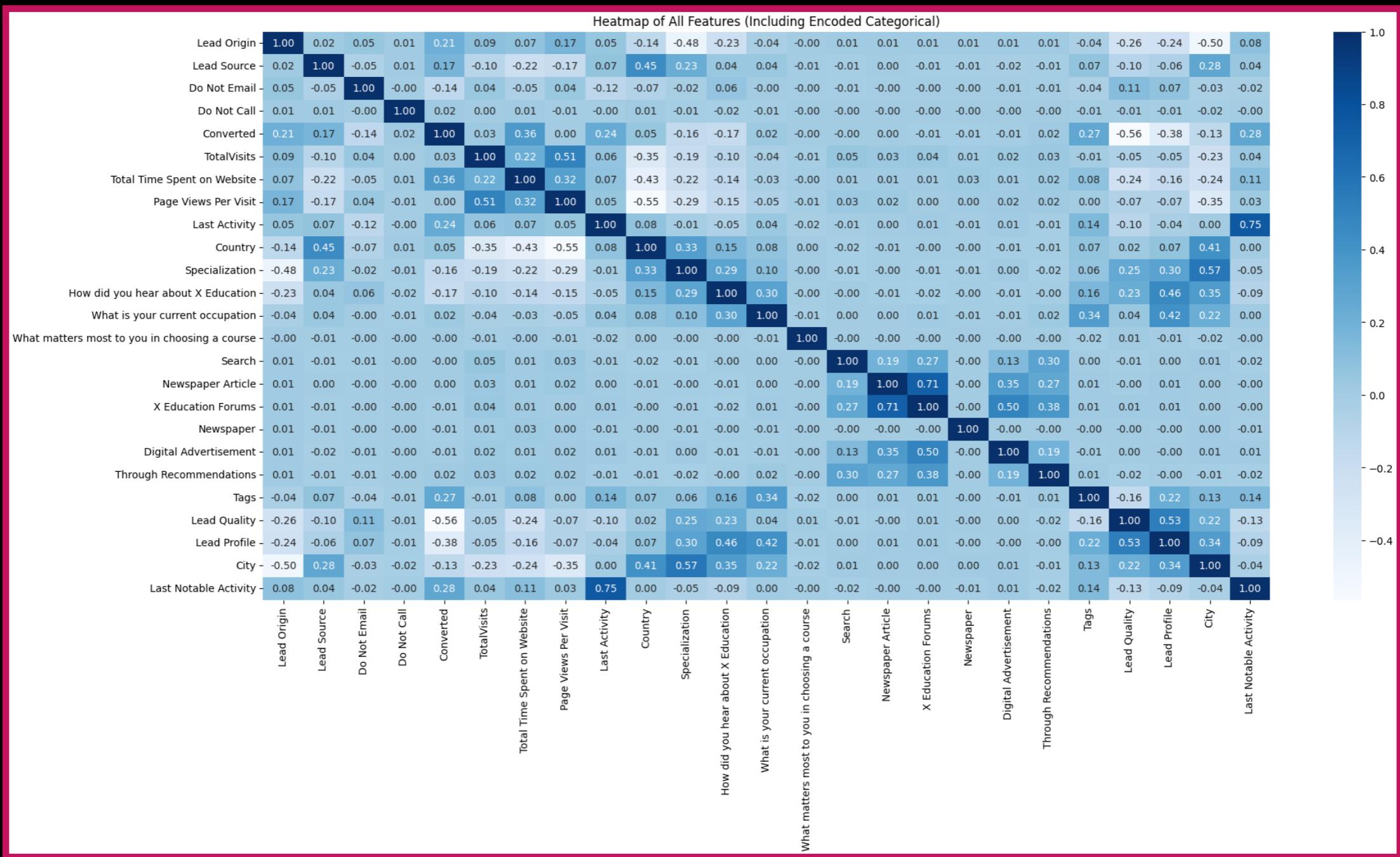
sns.countplot(df_final['Lead Source'].tick_params(axis='x', rotation = 90)
plt.title('Lead Source')
plt.show()

```



# Multivariate analysis

- Checking the count of categorical columns.
- Using heatmap to study relationships between multiple variables.



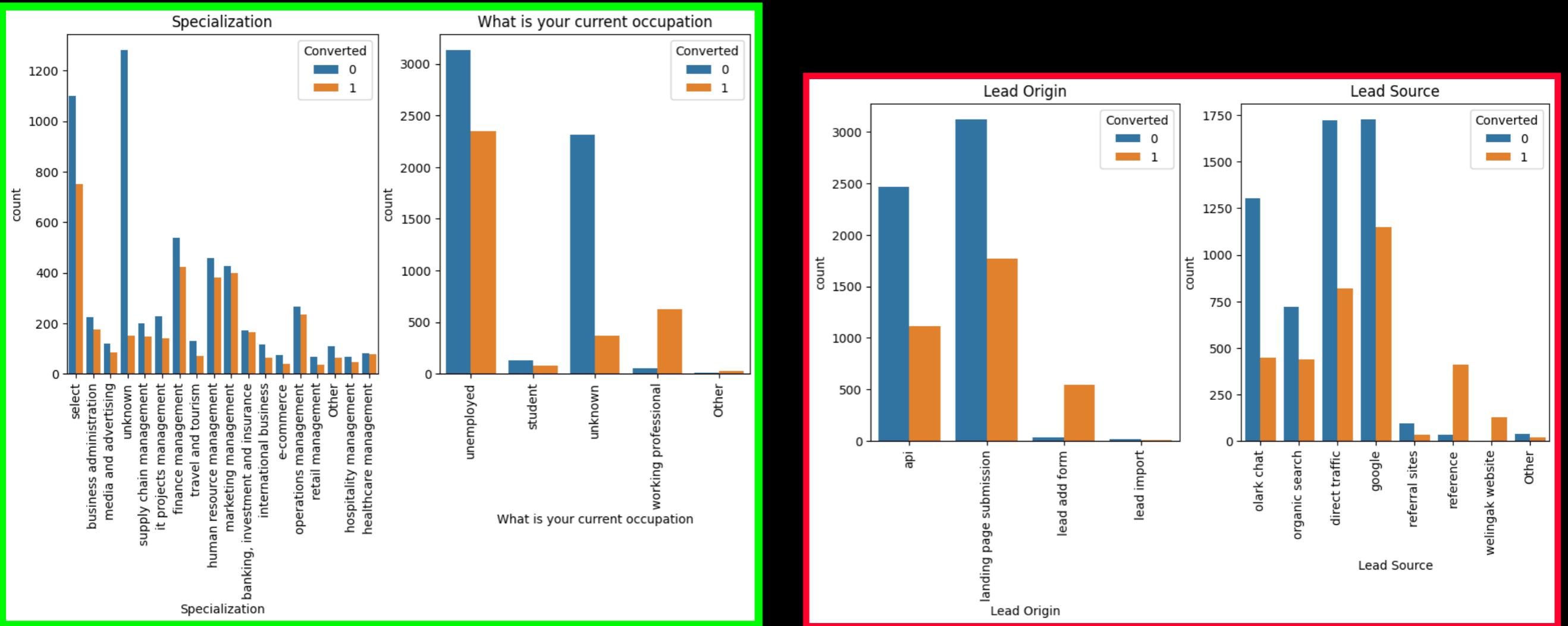
```

plt.figure(figsize = (10,5))

plt.subplot(1,2,1)
sns.countplot(x='Specialization', hue='Converted', data= df_final).tick_params(axis='x', rotation = 90)
plt.title('Specialization')

plt.subplot(1,2,2)
sns.countplot(x='What is your current occupation', hue='Converted', data= df_final).tick_params(axis='x', rotation = 90)
plt.title('What is your current occupation')
plt.show()

```



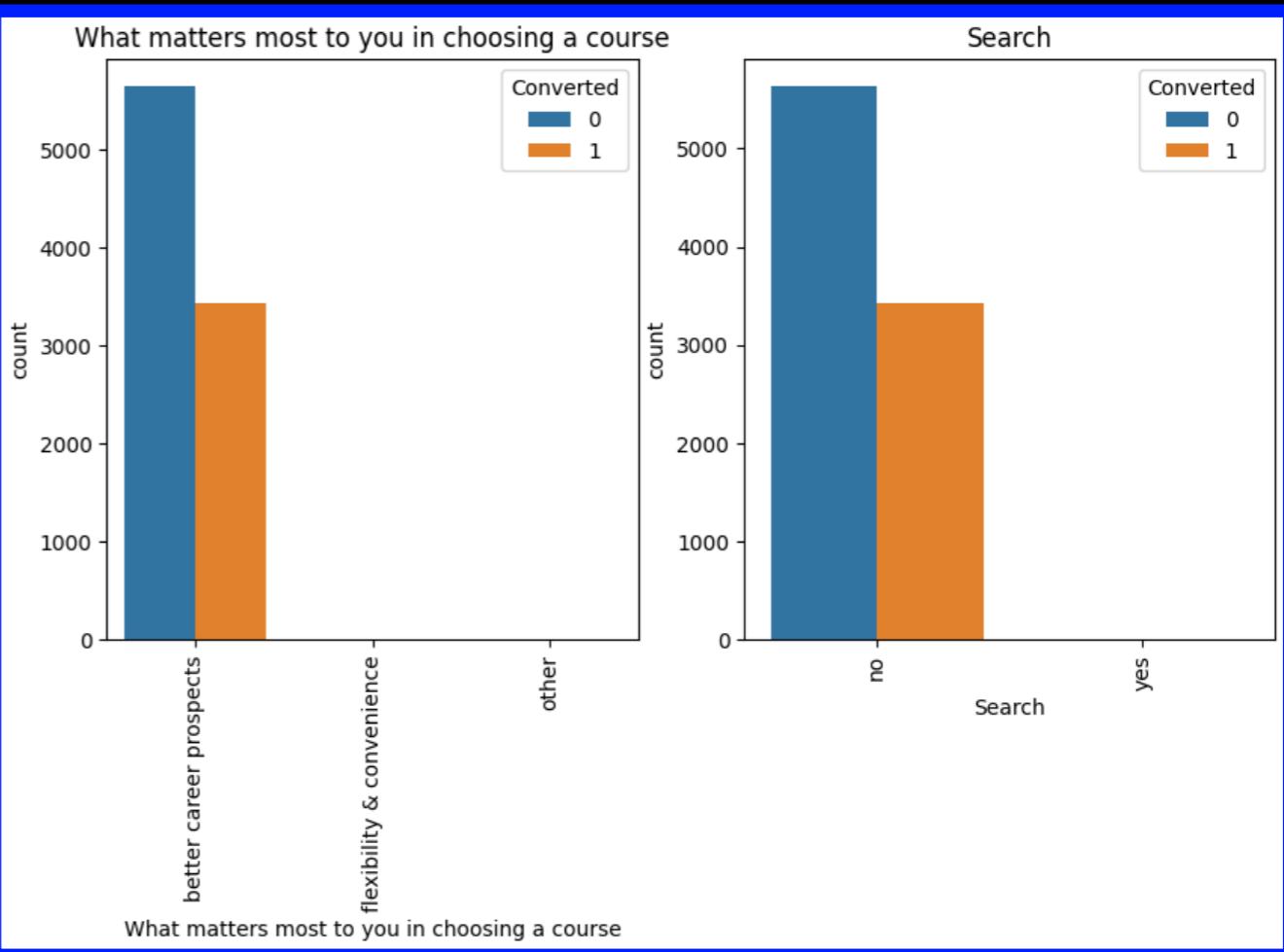
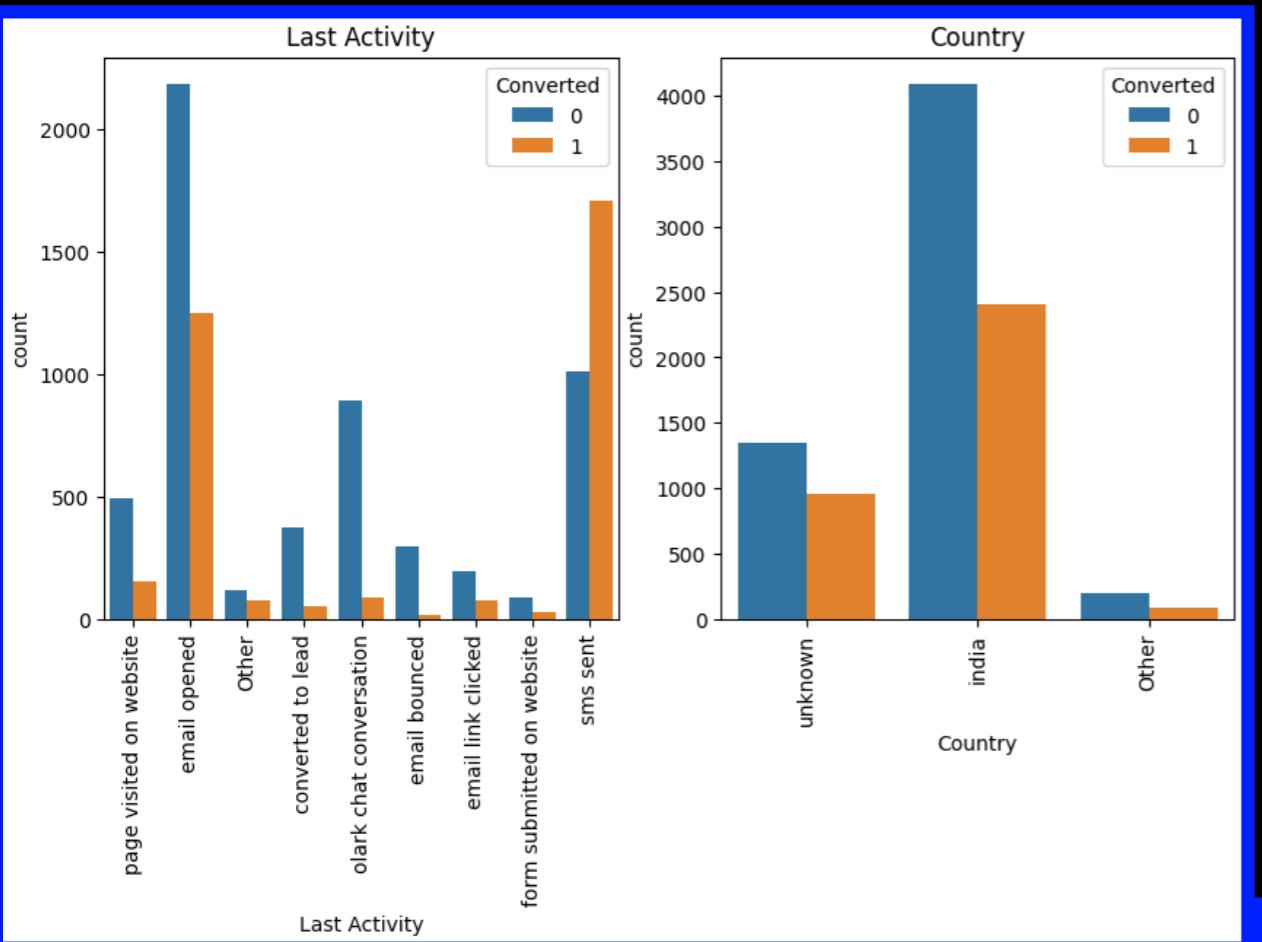
```

plt.figure(figsize = (10,5))

plt.subplot(1,2,1)
sns.countplot(x='Lead Origin', hue='Converted', data= df_final).tick_params(axis='x', rotation = 90)
plt.title('Lead Origin')

plt.subplot(1,2,2)
sns.countplot(x='Lead Source', hue='Converted', data= df_final).tick_params(axis='x', rotation = 90)
plt.title('Lead Source')
plt.show()

```



# Creating Dummy vars, splitting into Train-Test data:

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

Python

```
Index(['Lead Origin', 'Lead Source', 'Do Not Email', 'Do Not Call',  
       'Last Activity', 'Country', 'Specialization',  
       'What is your current occupation',  
       'What matters most to you in choosing a course', 'Search',  
       'Newspaper Article', 'X Education Forums', 'Newspaper',  
       'Digital Advertisement', 'Through Recommendations',  
       'A free copy of Mastering The Interview', 'Last Notable Activity'],  
      dtype='object')
```

```
# Create dummy variables using the 'get_dummies'  
dummy = pd.get_dummies(df_final[['Lead Origin','Specialization','Lead Source', 'Do Not Email', 'Last Activity', 'What is your current occupation','A  
# Add the results to the master dataframe  
df_final_dum = pd.concat([df_final, dummy], axis=1)  
df_final_dum
```

Python

	Lead Origin	Lead Source	Do Not Email	Do Not Call	Converted	TotalVisits	Total Time Spent on Website	Page Views Per Visit	Last Activity	Country	...	Last Notable Activity_form submitted on website	Last Notable Activity_had a phone conversation	Last Notable Activity_modified	Last Notable Activity_o conversation
0	api	olark chat	no	no	0	0.0	0	0.00	page visited on website	not provided	...	False	False	True	F
1	api	organic search	no	no	0	5.0	674	2.50	email opened	india	...	False	False	False	F
2	landing page submission	direct traffic	no	no	1	2.0	1532	2.00	email opened	india	...	False	False	False	F

```
# Split the dataset into 70% and 30% for train and test respectively  
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=10)
```

Python

```
# Import MinMax scaler  
from sklearn.preprocessing import MinMaxScaler  
# Scale the three numeric features  
scaler = MinMaxScaler()  
X_train[['TotalVisits', 'Page Views Per Visit', 'Total Time Spent on Website']] = scaler.fit_transform(X_train[['TotalVisits', 'Page Views Per Visit',  
X_train.head()
```

Python

	TotalVisits	Total Time Spent on Website	Page Views Per Visit	Lead Origin_landing page submission	Lead Origin_lead add form	Lead Origin_lead import	Specialization_business administration	Specialization_e-business	Specialization_e-commerce	Specialization_e-finance management
1289	0.014184	0.612676	0.083333	True	False	False	False	False	False	True
3604	0.000000	0.000000	0.000000	False	False	False	False	False	False	False
5584	0.042553	0.751761	0.250000	True	False	False	False	False	False	False
7679	0.000000	0.000000	0.000000	False	False	False	False	False	False	False
7563	0.014184	0.787852	0.083333	True	False	False	False	False	False	False

5 rows x 80 columns

# Building the model

- Feature selection using RFE.
  - Using statsmodels, view summary of logistic model.
  - Examine p-values, accuracy.
  - Calculate vif's to eliminate some features.

## 5. Model Building

```
# Import 'LogisticRegression'  
from sklearn.linear_model import LogisticRegression  
logreg = LogisticRegression()  
  
# Import RFE  
from sklearn.feature_selection import RFE  
  
from sklearn.linear_model import LogisticRegression  
  
logreg = LogisticRegression() # Ensure you define logreg first  
rfe = RFE(estimator=logreg, n_features_to_select=15) # Corrected syntax  
rfe = rfe.fit(X_train, y_train)  
  
# Features that have been selected by RFE  
list(zip(X_train.columns, rfe.support_, rfe.ranking_))  
  
'TotalVisits', True, 1),  
'Total Time Spent on Website', True, 1),  
'Lead Origin_lead add form', True, 1),  
'Lead Source_direct traffic', True, 1),  
'Lead Source_google', True, 1),  
'Lead Source_organic search', True, 1),  
'Lead Source_welingak website', True, 1),  
'Do Not Email_yes', True, 1),  
'Last Activity_olark chat conversation', True, 1),
```

```

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

```

Generalized Linear Model Regression Results								
Dep. Variable:	Converted	No. Observations:	6351					
Model:	GLM	Df Residuals:	6335					
Model Family:	Binomial	Df Model:	15					
Link Function:	Logit	Scale:	1.0000					
Method:	IRLS	Log-Likelihood:	-3239.5					
Date:	Wed, 12 Mar 2025	Deviance:	6478.9					
Time:	15:49:53	Pearson chi2:	6.66e+03					
No. Iterations:	21	Pseudo R-squ. (CS):	0.2697					
Covariance Type:	nonrobust							
		coef	std err	z	P> z	[0.025	0.975]	
	const	-1.0055	0.077	-13.139	0.000	-1.156	-0.856	
	TotalVisits	-19.8464	4.82e+04	-0.000	1.000	-9.45e+04	9.44e+04	
	Total Time Spent on Website	-22.6948	4.82e+04	-0.000	1.000	-9.45e+04	9.44e+04	
	Lead Origin_lead add form	2.5558	0.222	11.491	0.000	2.120	2.992	
	Lead Source_direct traffic	-0.2037	0.093	-2.201	0.028	-0.385	-0.022	
	Lead Source_google	0.1280	0.088	1.450	0.147	-0.045	0.301	
	Lead Source_organic search	0.0061	0.109	0.056	0.956	-0.207	0.219	
	Lead Source_welingak website	2.5529	1.033	2.472	0.013	0.529	4.577	
	Do Not Email_yes	-1.5104	0.154	-9.782	0.000	-1.813	-1.208	
	Last Activity_olark chat conversation	-1.4719	0.156	-9.461	0.000	-1.777	-1.167	
	Last Activity_sms sent	1.3379	0.065	20.623	0.000	1.211	1.465	
	What is your current occupation_housewife	23.2589	2.02e+04	0.001	0.999	-3.95e+04	3.96e+04	
	What is your current occupation_other	2.0102	0.725	2.773	0.006	0.589	3.431	
	What is your current occupation_working professional	2.8842	0.184	15.662	0.000	2.523	3.245	
	Last Notable Activity_had a phone conversation	23.2666	1.63e+04	0.001	0.999	-3.19e+04	3.19e+04	
	Last Notable Activity_unreachable	1.6314	0.547	2.981	0.003	0.559	2.704	

```

# 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

```

	Features	VIF
2	Lead Origin_lead add form	1.47
9	Last Activity_sms sent	1.46
6	Lead Source_welingak website	1.31
3	Lead Source_direct traffic	1.21
4	Lead Source_google	1.19
12	What is your current occupation_working profes...	1.16
5	Lead Source_organic search	1.10
7	Do Not Email_yes	1.10
8	Last Activity_olark chat conversation	1.02
13	Last Notable Activity_unreachable	1.01
0	TotalVisits	1.00
1	Total Time Spent on Website	1.00
10	What is your current occupation_housewife	1.00
11	What is your current occupation_other	1.00

```
X_train.drop('What is your current occupation_housewife', axis = 1, inplace = True)
```

Python

```
# Refit the model with the new set of features  
X_train_sm = sm.add_constant(X_train)  
logm3 = sm.GLM(y_train, X_train_sm, family = sm.families.Binomial())  
res = logm3.fit()  
res.summary()
```

Python

## Making predictions on the dataset:

```
> # Predicting the probabilities on the train set  
y_train_pred = res.predict(X_train_sm)  
y_train_pred[:10]
```

```
232] .. 1289  0.298083  
3604  0.269836  
5584  0.076423  
7679  0.269836  
7563  0.076423  
7978  0.586384  
7780  0.274650  
7863  0.983881  
838   0.870389  
708   0.298083  
dtype: float64
```

Python

```
> # Reshaping to an array  
y_train_pred = y_train_pred.values.reshape(-1)  
y_train_pred[:10]
```

```
234] .. array([0.2980832 , 0.26983639, 0.07642343, 0.26983639, 0.07642343,  
0.58638417, 0.2746502 , 0.9838813 , 0.87038897, 0.2980832 ])
```

Python

```
> # Data frame with given conversion rate and probability of predicted ones  
y_train_pred_final = pd.DataFrame({'Converted':y_train.values, 'Conversion_Prob':y_train_pred})  
y_train_pred_final.head()
```

```
236] ..
```

	Converted	Conversion_Prob
0	1	0.298083
1	0	0.269836
2	0	0.076423
3	0	0.269836
4	0	0.076423

Python

# Evaluating the model and Optimising the cutoff

## 7. Model Evaluation

```
[90] # Importing metrics from sklearn for evaluation
from sklearn import metrics
Python

[91] # Creating confusion matrix
confusion = metrics.confusion_matrix(y_train_pred_final.Converted, y_train_pred_final.Predicted )
confusion
Python

... array([[3680,  215],
       [ 213, 2243]])

[92] # Predicted      not_churn     churn
# Actual
# not_churn      3403        492
# churn          729         1727
Python

[93] # Check the overall accuracy
metrics.accuracy_score(y_train_pred_final.Converted, y_train_pred_final.Predicted)
Python

... 0.9326090379467801
```

That's around 93% accuracy with is a very good value

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

```
# Calculating the sensitivity
TP/(TP+FN)
```

```
0.9132736156351792
```

```
# Calculating the specificity
TN/(TN+FP)
```

```
0.944801026957638
```

# ROC curve

The previous cut off was randomly selected. Now to find the optimum one

```
# ROC function
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, 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
```

259]

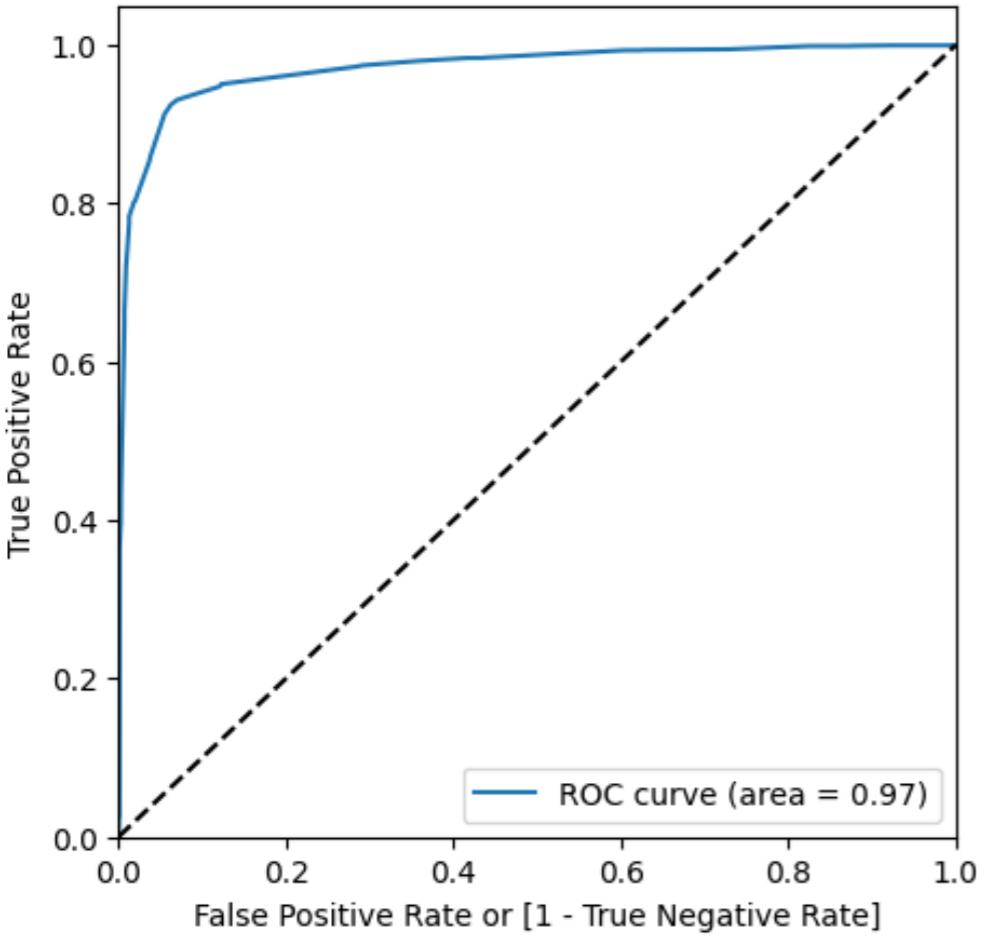
```
fpr, tpr, thresholds = metrics.roc_curve( y_train_pred_final.Converted, y_train_pred_final.Conversion_Prob, drop_intermediate = False )
```

261]

```
# Call the ROC function
draw_roc(y_train_pred_final.Converted, y_train_pred_final.Conversion_Prob)
```

263]

Receiver operating characteristic example



```

# Check the overall accuracy
metrics.accuracy_score(y_train_pred_final.Converted, y_train_pred_final.final_predicted)
104] ... 0.9324515824279641

# Creating confusion matrix
confusion2 = metrics.confusion_matrix(y_train_pred_final.Converted, y_train_pred_final.final_predicted )
confusion2
105] ... array([[3660, 235],
   [ 194, 2262]])

# Substituting the value of true positive
TP = confusion2[1,1]
# Substituting the value of true negatives
TN = confusion2[0,0]
# Substituting the value of false positives
FP = confusion2[0,1]
# Substituting the value of false negatives
FN = confusion2[1,0]
106] ... Python

# Calculating the sensitivity
TP/(TP+FN)
107] ... 0.9210097719869706

# Calculating the specificity
TN/(TN+FP)
108] ... 0.9396662387676509

```

```

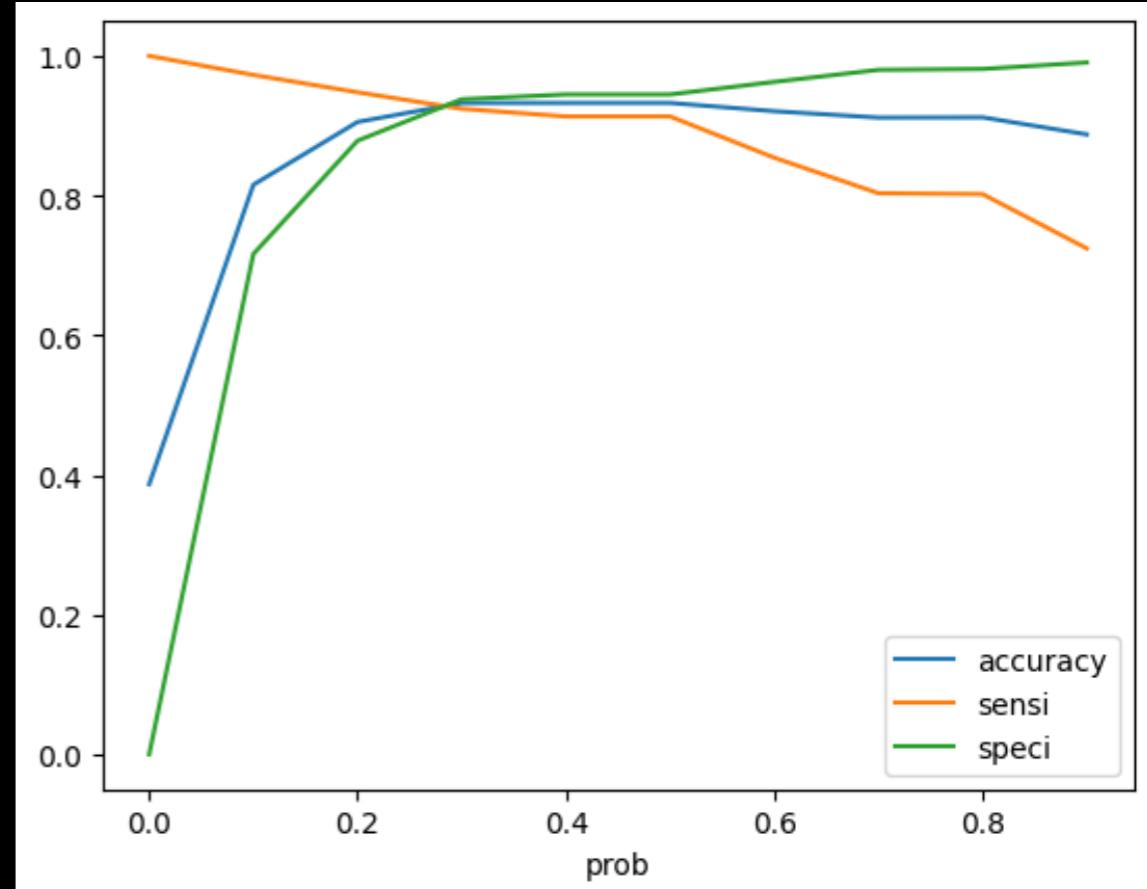
# Creating a dataframe to see the values of accuracy, sensitivity, and specificity at different values of probability cutoffs
cutoff_df = pd.DataFrame( columns = ['prob','accuracy','sensi','speci'])
# Making confusing matrix to find values of sensitivity, accurate and specificity for each level of probability
from sklearn.metrics import confusion_matrix
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]
cutoff_df
68] ... Python

   prob  accuracy     sensi     speci
0.0  0.0  0.386711  1.000000  0.000000
0.1  0.1  0.518973  0.971498  0.233633
0.2  0.2  0.525114  0.970277  0.244416
0.3  0.3  0.750276  0.626221  0.828498
0.4  0.4  0.750276  0.626221  0.828498
0.5  0.5  0.750276  0.626221  0.828498
0.6  0.6  0.731696  0.416938  0.930167
0.7  0.7  0.714218  0.282980  0.986136
0.8  0.8  0.713274  0.279723  0.986650
0.9  0.9  0.679421  0.178339  0.995379

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

```



# Results and metrics on test set

```
[138] # Check the overall accuracy  
metrics.accuracy_score(y_pred_final['Converted'], y_pred_final.final_predicted)
```

```
... 0.9302240176276166
```

Python

```
[139] # Creating confusion matrix  
confusion2 = metrics.confusion_matrix(y_pred_final['Converted'], y_pred_final.final_predicted )  
confusion2
```

```
... array([[1635, 109],  
       [ 81, 898]])
```

Python

```
[140] # Substituting the value of true positive  
TP = confusion2[1,1]  
# Substituting the value of true negatives  
TN = confusion2[0,0]  
# Substituting the value of false positives  
FP = confusion2[0,1]  
# Substituting the value of false negatives  
FN = confusion2[1,0]
```

Python

```
[141] # Precision = TP / TP + FP  
TP / (TP + FP)
```

```
... 0.8917576961271102
```

Python

```
[142] #Recall = TP / TP + FN  
TP / (TP + FN)
```

```
... 0.9172625127681308
```

Python

# Conclusion

The key factors influencing potential buyers, ranked from most to least important, are:

1. The total duration spent on the website.
2. The overall number of visits.
3. The lead source, particularly when it comes from:
  - Google
  - Direct traffic
  - Organic search
  - Welingak website
4. The last recorded activity, specifically:
  - SMS interactions
  - Olark chat conversations
5. The lead origin being from a Lead Ad format.
6. The individual's current occupation being a working professional.

Considering these factors, X Education has a strong opportunity to convert nearly all potential buyers into actual customers, significantly boosting their course enrollments.