LEAS SCORING ASSIGNMENT – GROUP ASSIGNMENT

Rishabh Gupta
Mandar darekar

Problem Statement

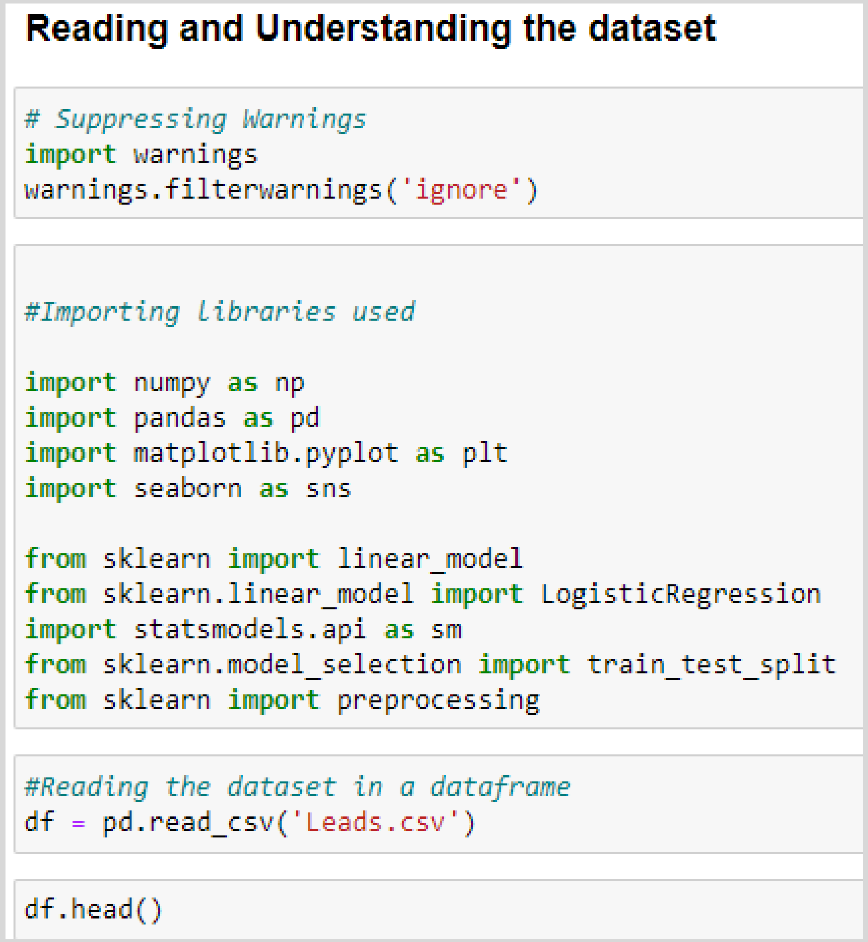
- X Education is an online education provider to industry professional. They market their products through various digital media and capture the leads(people who click ads/ watch videos/ enter contact info) and have sales team try and convert those leads to actual customers. The conversion rate though is not very great and hence, X education has appointed us to help them 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 we 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%. They want to solve this problem using data and analytics.

Approach

To approach this problem we are given a dataset with >9000 data points and we need to utilize them to work on a target variable 'Converted' so that we increase the overall lead conversion ratio. We need to identify the potential 'Hot Leads' so that sales can specifically focus on those leads and try and convert almost all of them. We chose to do the following steps to go forward with this case study:

1. Read and understand the data
2. Clean the data - Data Preparation (This includes handling nulls and treating outliers and making the dataset to a most useful form so that we can drive best results)
3. Performing univariate and bivariate analysis to get a glimpse and summary deductions from the dataset
4. Creating Dummies for categorical Variables - Data Preparation
5. Performing train-test split and scaling - Data Preparation
6. Modelling - RFE, GLM, optimal cutoffs
7. Check the validity of the model by using cunfusion matrix and metrics such as Accuracy, Sensitivity, Speficity, Recall and Precision
8. Test the data on the test set to check how the model is performing
9. Drive insights on the outcome and suggest recommendations based on optimal threshold

Reading Data and Cleaning Data



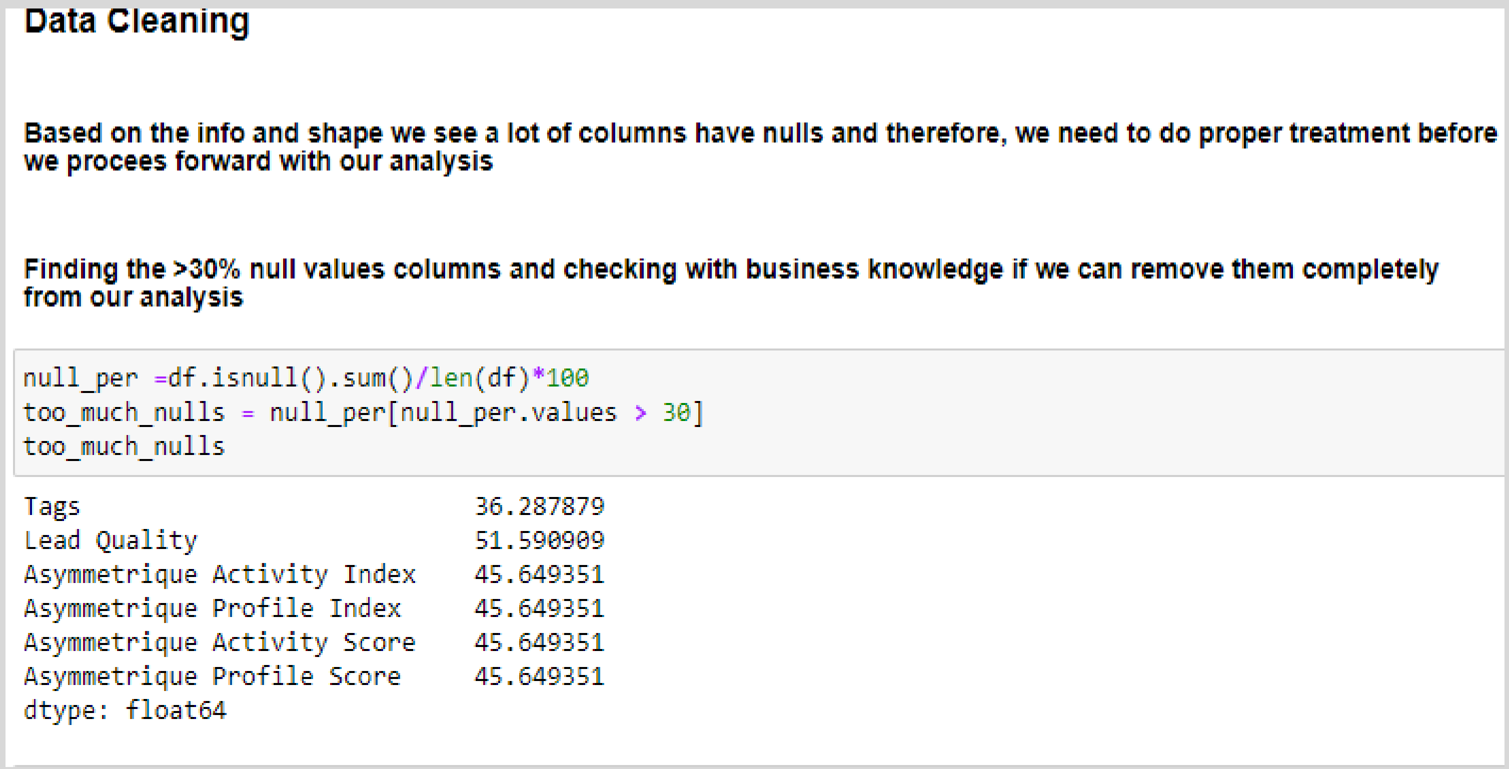
Reading and Understanding the dataset

```python
# Suppressing Warnings
import warnings
warnings.filterwarnings('ignore')
```

```python
#Importing libraries used

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
```

```python
#Reading the dataset in a dataframe
df = pd.read_csv('Leads.csv')
```

```python
df.head()
```



Data Cleaning

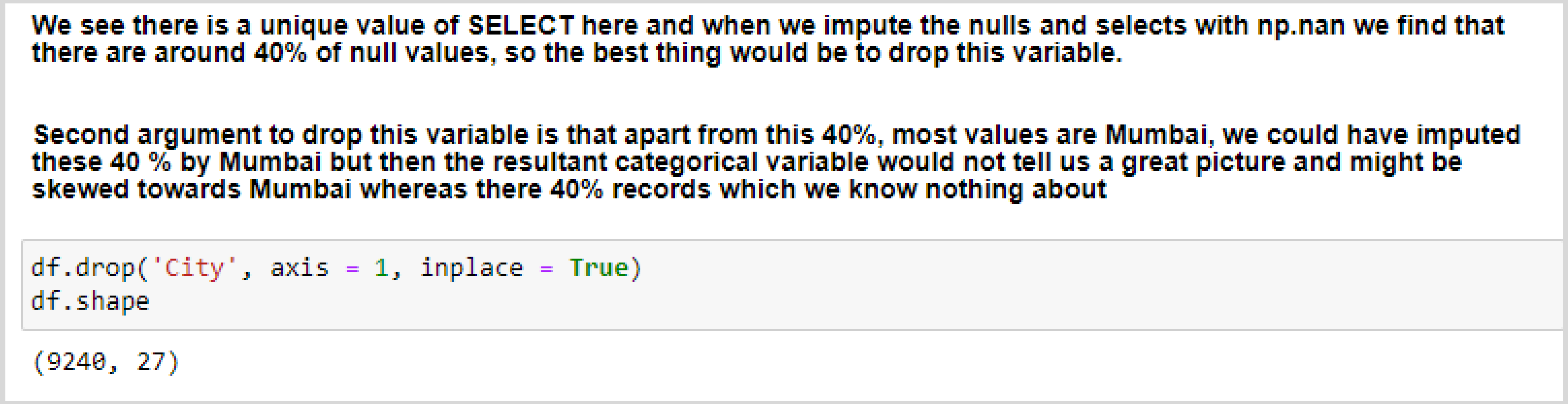
Based on the info and shape we see a lot of columns have nulls and therefore, we need to do proper treatment before we procees forward with our analysis

Finding the >30% null values columns and checking with business knowledge if we can remove them completely from our analysis

```python
null_per =df.isnull().sum()/len(df)*100
too_much_nulls = null_per[null_per.values > 30]
too_much_nulls
```

```
Tags                           36.287879
Lead Quality                   51.590909
Asymmetrique Activity Index    45.649351
Asymmetrique Profile Index     45.649351
Asymmetrique Activity Score    45.649351
Asymmetrique Profile Score     45.649351
dtype: float64
```

Data Cleaning

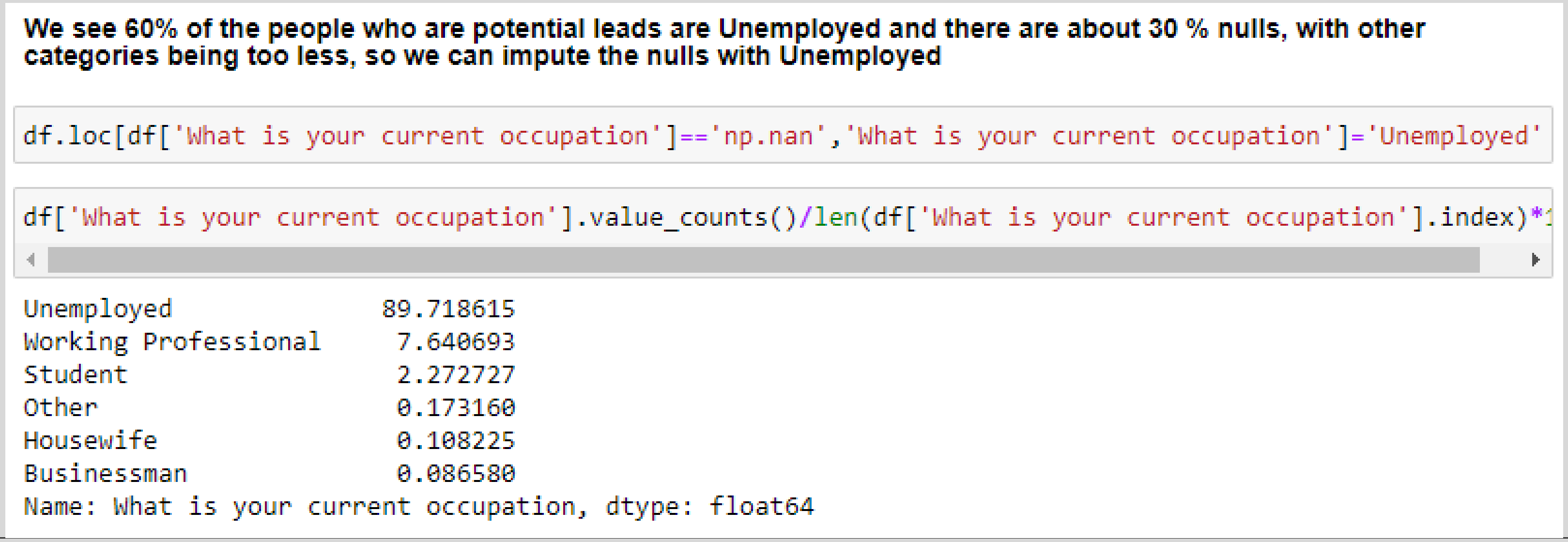


We see there is a unique value of SELECT here and when we impute the nulls and selects with np.nan we find that there are around 40% of null values, so the best thing would be to drop this variable.

Second argument to drop this variable is that apart from this 40%, most values are Mumbai, we could have imputed these 40 % by Mumbai but then the resultant categorical variable would not tell us a great picture and might be skewed towards Mumbai whereas there 40% records which we know nothing about

```
df.drop('City', axis = 1, inplace = True)
df.shape
```

```
(9240, 27)
```



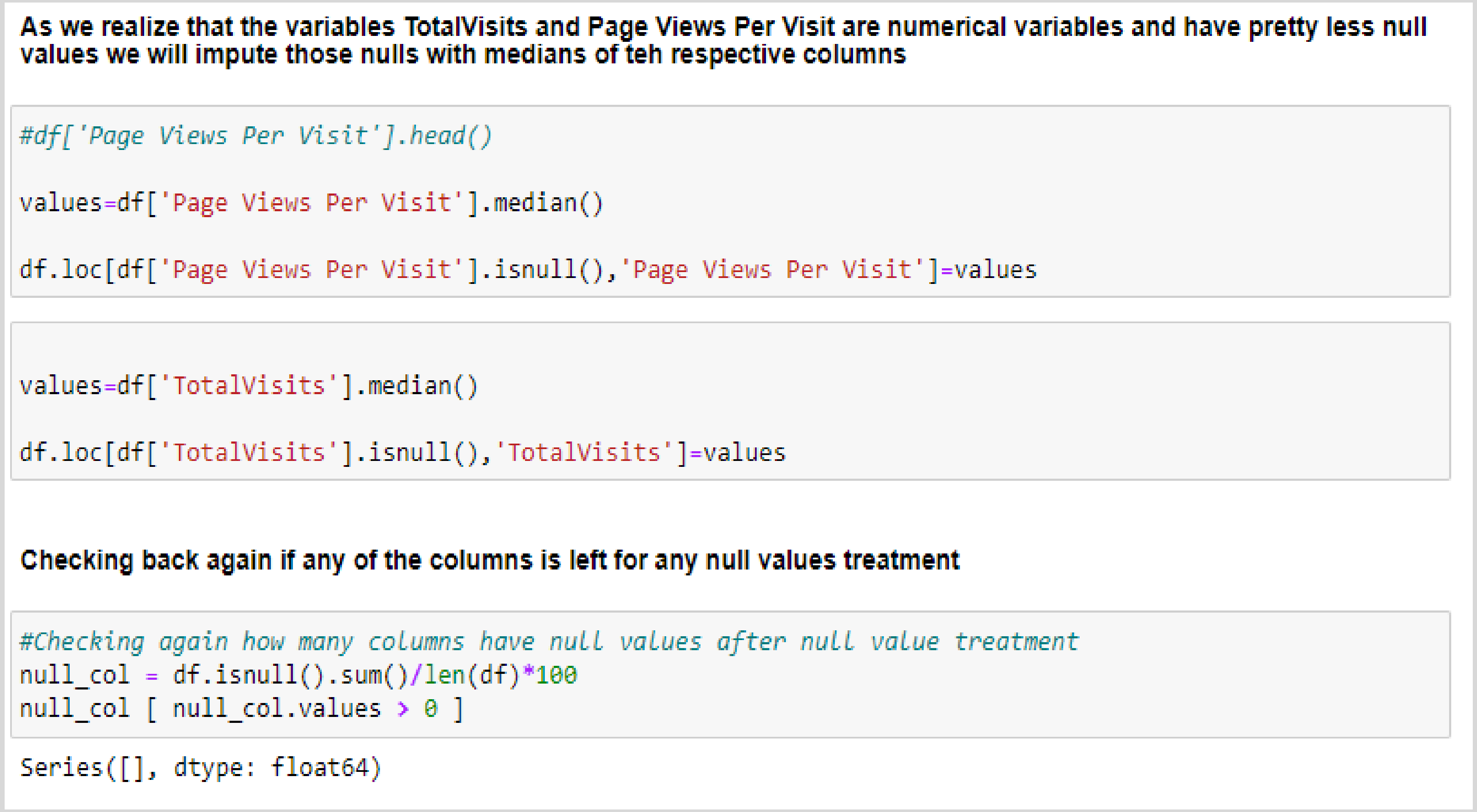
We see 60% of the people who are potential leads are Unemployed and there are about 30 % nulls, with other categories being too less, so we can impute the nulls with Unemployed

```
df.loc[df['What is your current occupation']=='np.nan','What is your current occupation']='Unemployed'
```

```
df['What is your current occupation'].value_counts()/len(df['What is your current occupation'].index)*1
```

```
Unemployed              89.718615
Working Professional     7.640693
Student                  2.272727
Other                    0.173160
Housewife                0.108225
Businessman              0.086580
Name: What is your current occupation, dtype: float64
```

Data Cleaning – Numerical data



As we realize that the variables TotalVisits and Page Views Per Visit are numerical variables and have pretty less null values we will impute those nulls with medians of teh respective columns

```
#df['Page Views Per Visit'].head()

values=df['Page Views Per Visit'].median()

df.loc[df['Page Views Per Visit'].isnull(),'Page Views Per Visit']=values
```

```


values=df['TotalVisits'].median()

df.loc[df['TotalVisits'].isnull(),'TotalVisits']=values
```

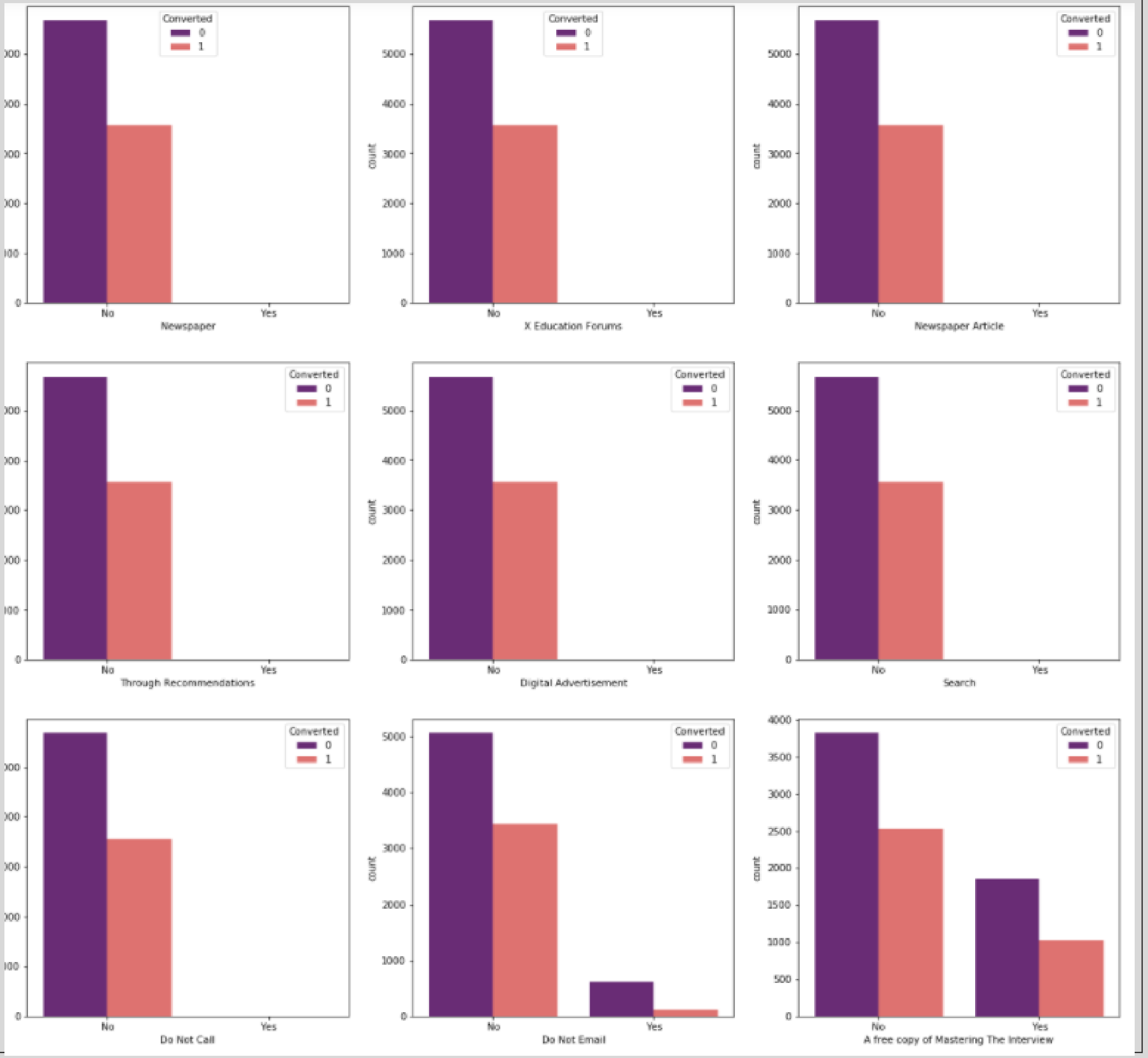
Checking back again if any of the columns is left for any null values treatment

```
#Checking again how many columns have null values after null value treatment
null_col = df.isnull().sum()/len(df)*100
null_col [ null_col.values > 0 ]
```

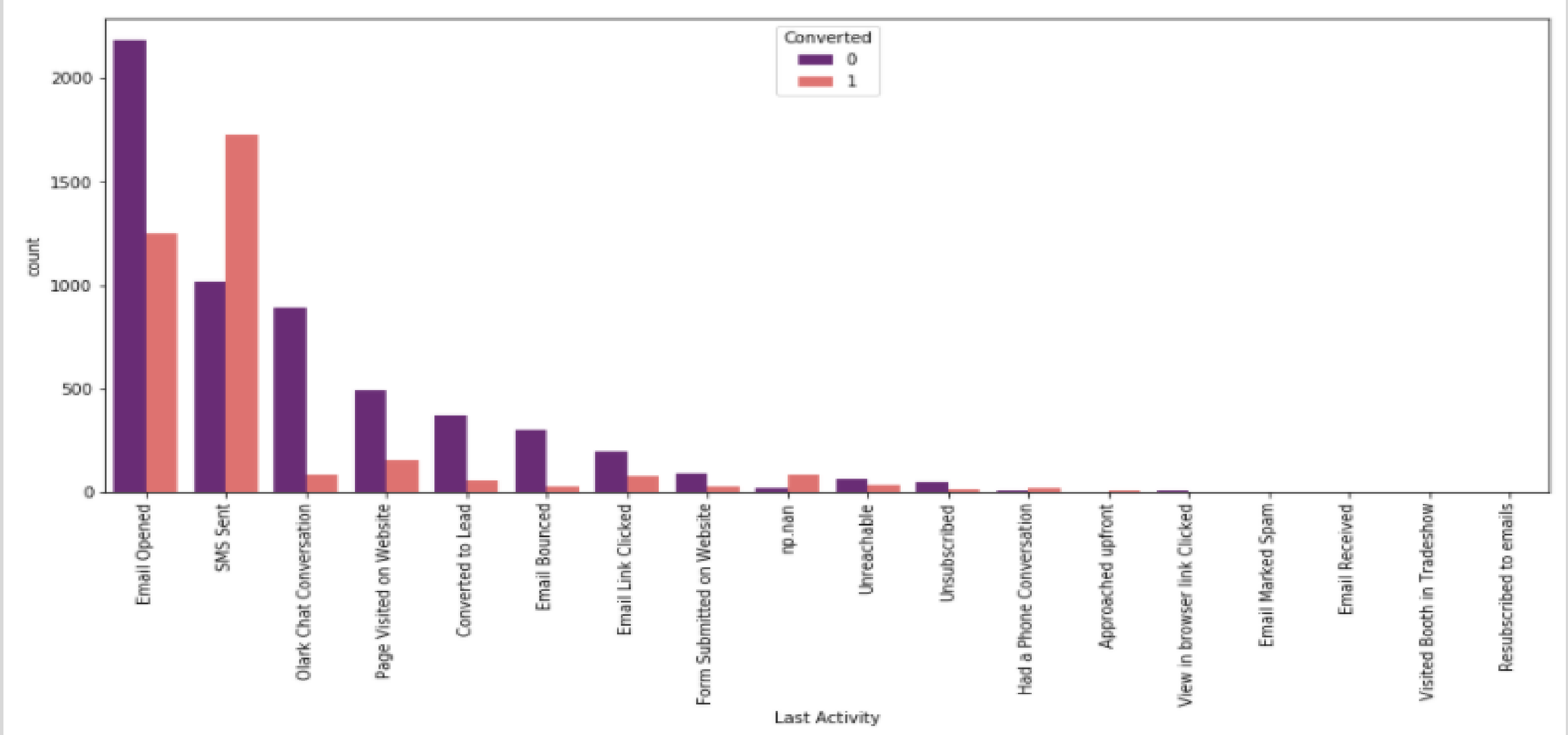
```
Series([], dtype: float64)
```

Univariate Analysis

- **Newspaper , X Education Forums , Newspaper Articles , Through Recommendations , Digital Advertisements :**
- for all these variables above mentioned , all the values are no hence it does not have any significant role in lead score, drop this column
- **Search** :
- 99% values are no except a few yes and missing, hence it does not have any significant role in lead score, drop this column
- **Do Not Call :**
- All the values are no except 2 values, hence there is no variance, doesnt indicate anything about leads and can easily be dropped

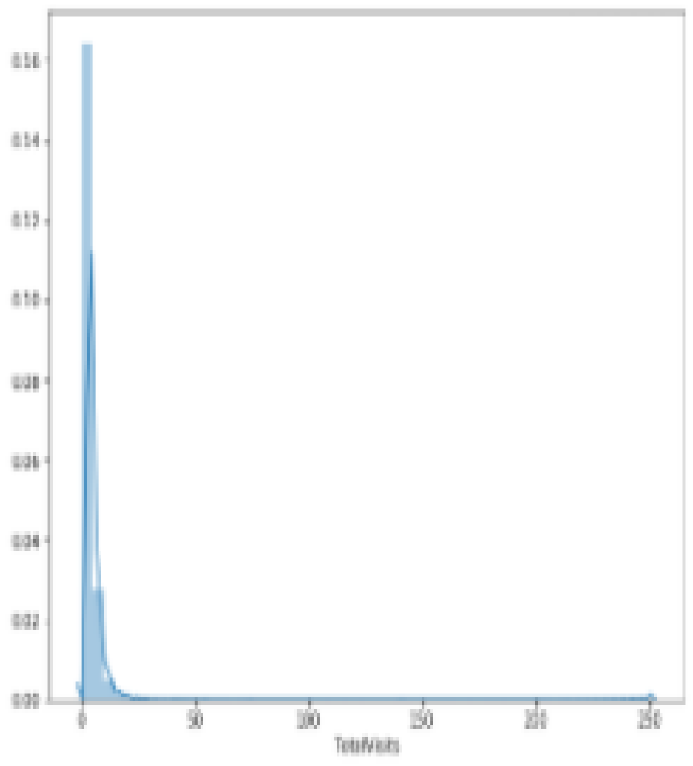


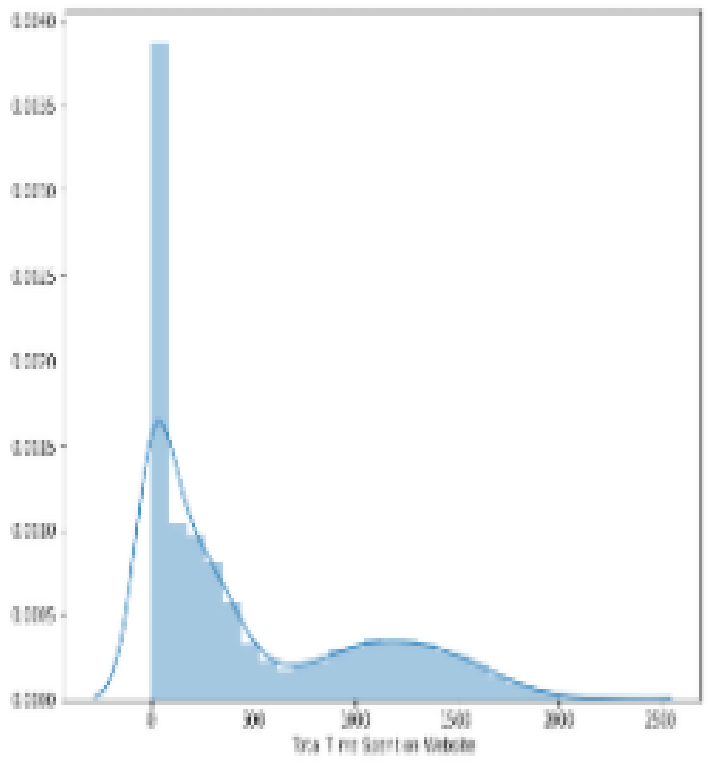
Univariate Analysis

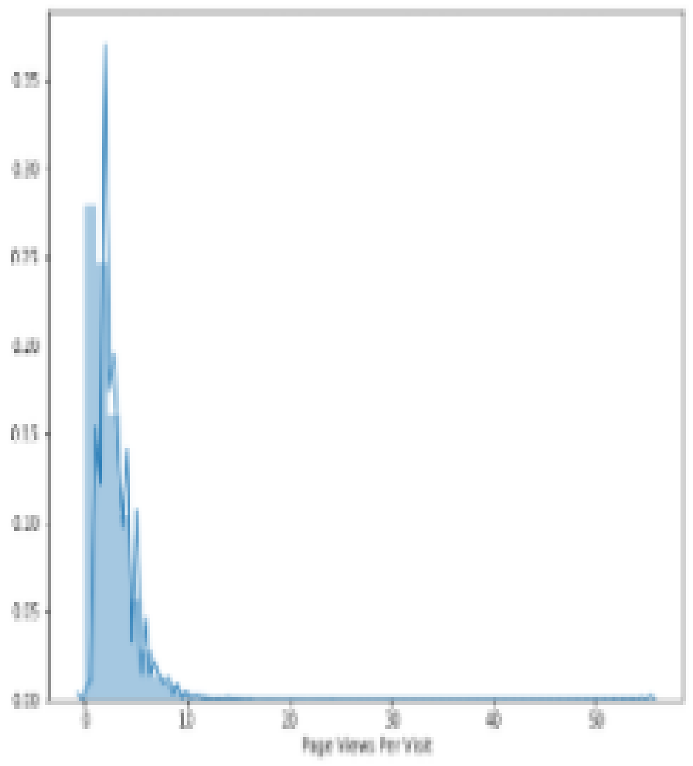


We see that the SMS Sent Activity people have relatively very high conversion rate than others and they even convert more often than not, so Sales team can predominantly work on converting more of these as this is the strength.

Univariate Analysis Numerical Fields

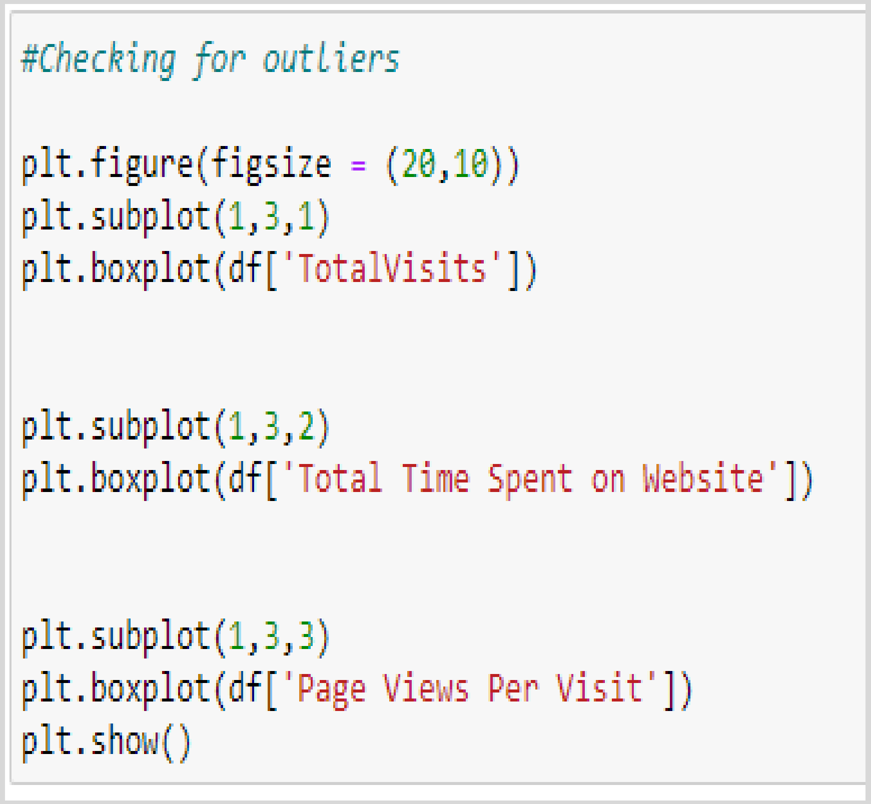




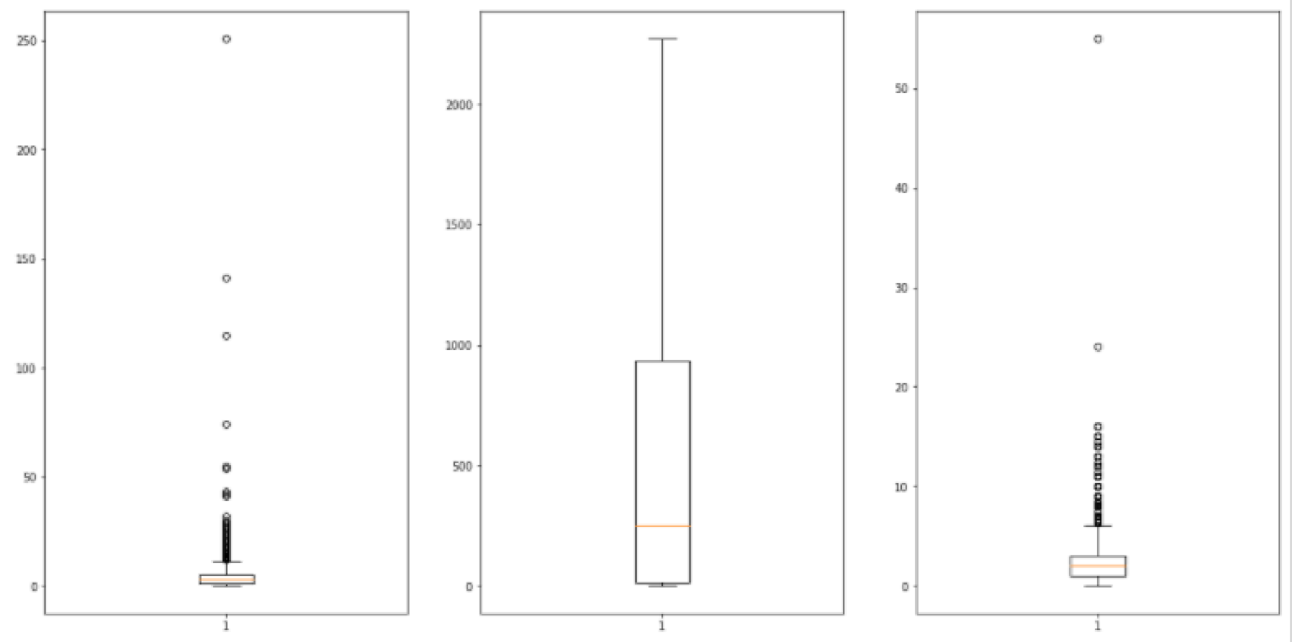


The numeric fields seem to be following the normal curve

Outlier Analysis

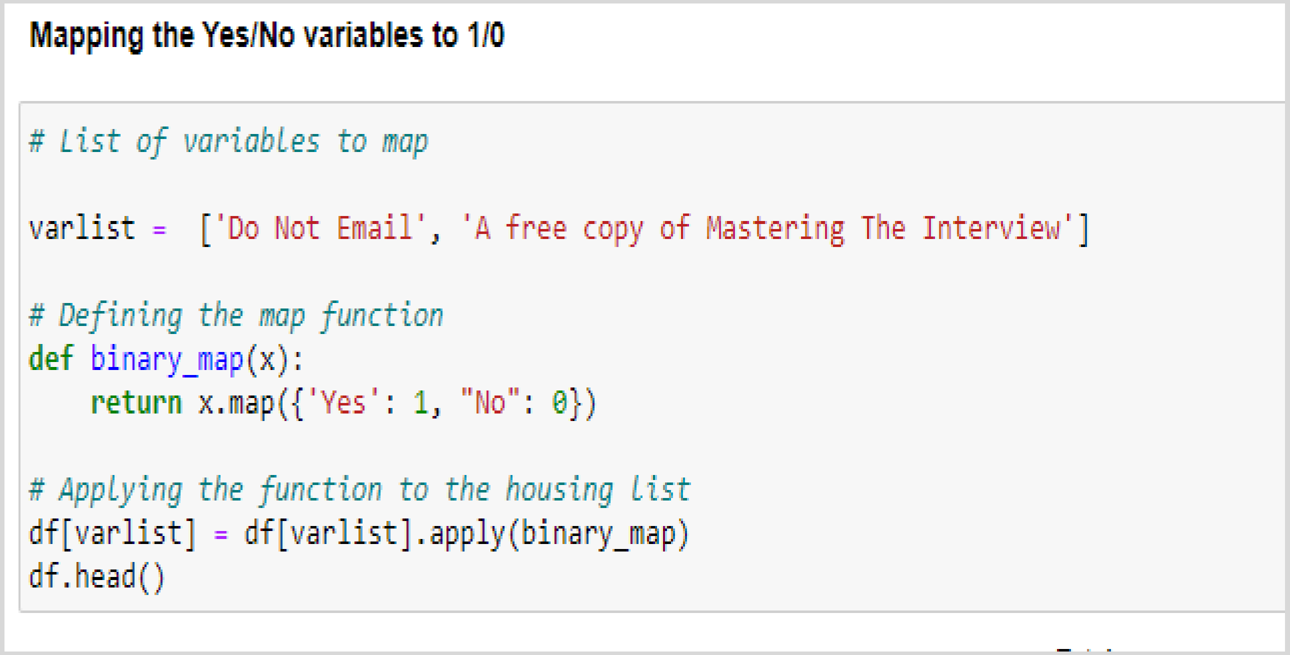


```
#Checking for outliers

plt.figure(figsize = (20,10))
plt.subplot(1,3,1)
plt.boxplot(df['TotalVisits'])


plt.subplot(1,3,2)
plt.boxplot(df['Total Time Spent on Website'])


plt.subplot(1,3,3)
plt.boxplot(df['Page Views Per Visit'])
plt.show()
```



We see Total Visits and Pages View per Visit have a few outliers but we are not very sure if we want treat the, as there can be cases where people are spending more time than most and then finally converting the leads to actual customers, so we'll leave it for business to decide

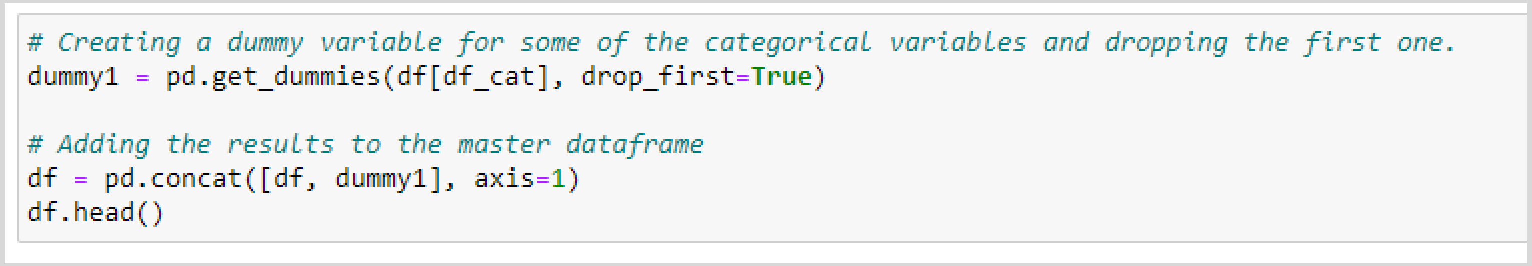
Mapping Yes/No flags to 1/0



Mapping the Yes/No variables to 1/0

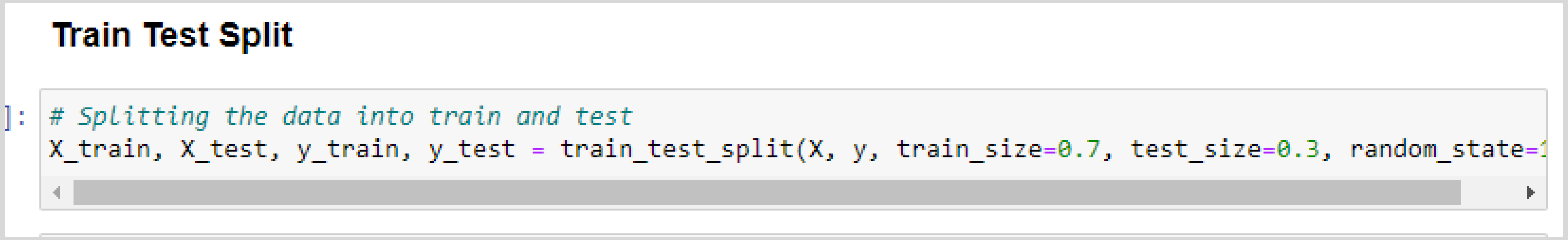
```
# List of variables to map

varlist =  ['Do Not Email', 'A free copy of Mastering The Interview']

# Defining the map function
def binary_map(x):
    return x.map({'Yes': 1, "No": 0})

# Applying the function to the housing list
df[varlist] = df[varlist].apply(binary_map)
df.head()
```

Dummy Variables



```
# Creating a dummy variable for some of the categorical variables and dropping the first one.
dummy1 = pd.get_dummies(df[df_cat], drop_first=True)

# Adding the results to the master dataframe
df = pd.concat([df, dummy1], axis=1)
df.head()
```

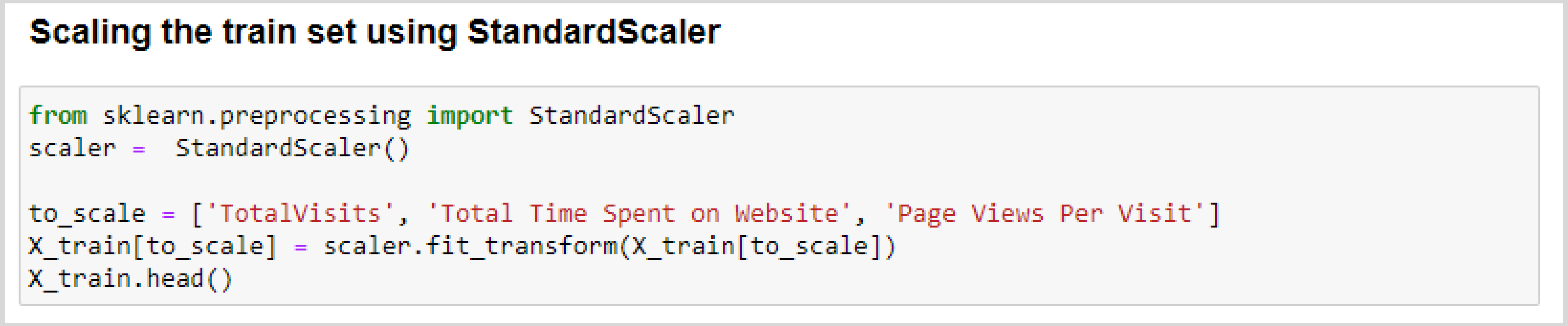
Train Test Split



Train Test Split

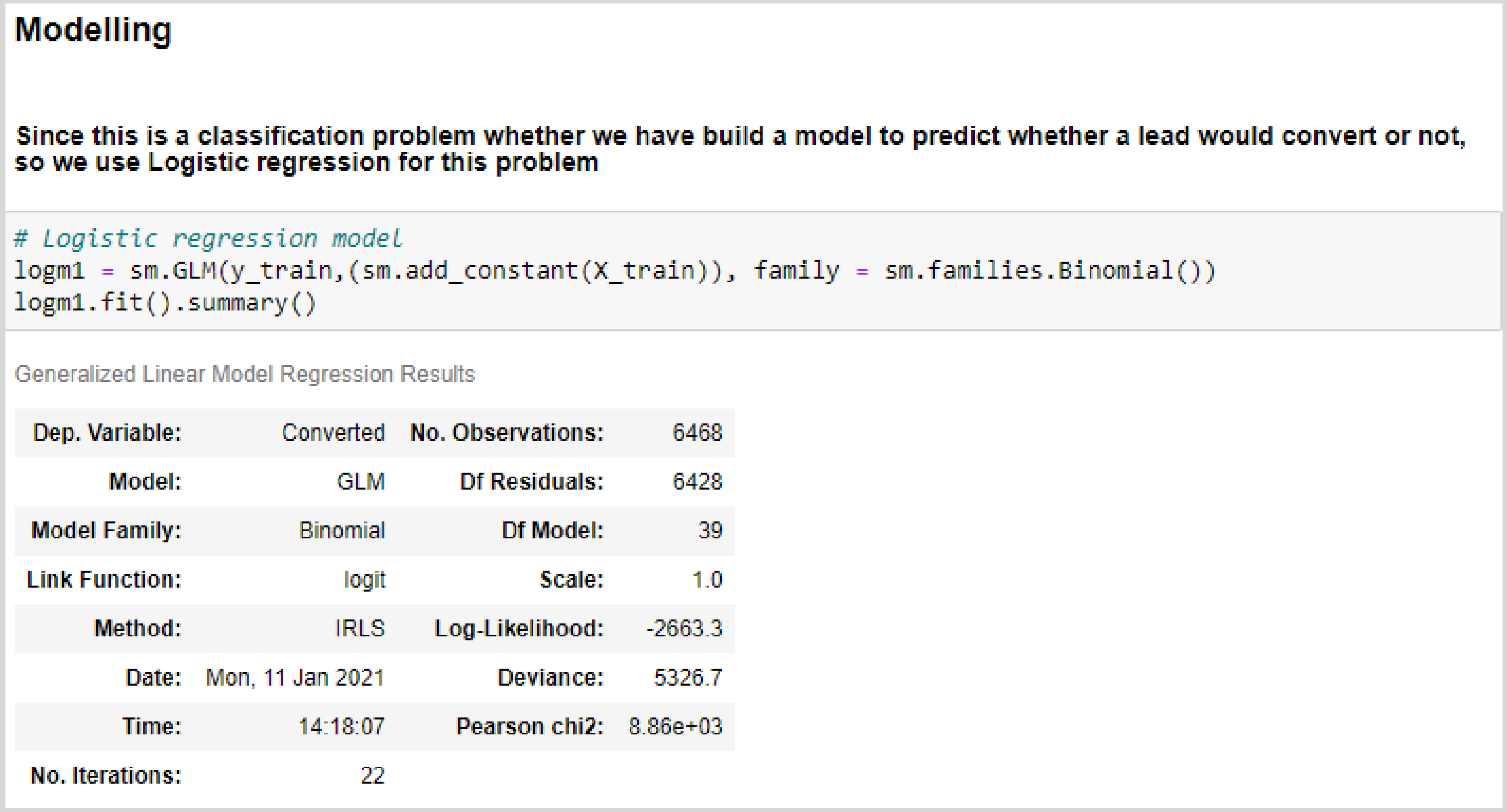
```
]: # Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7, test_size=0.3, random_state=
```

Scaling the train dataset


Scaling the train set using StandardScaler

```
from sklearn.preprocessing import StandardScaler
scaler =  StandardScaler()

to_scale = ['TotalVisits', 'Total Time Spent on Website', 'Page Views Per Visit']
X_train[to_scale] = scaler.fit_transform(X_train[to_scale])
X_train.head()
```

Modelling



Modelling

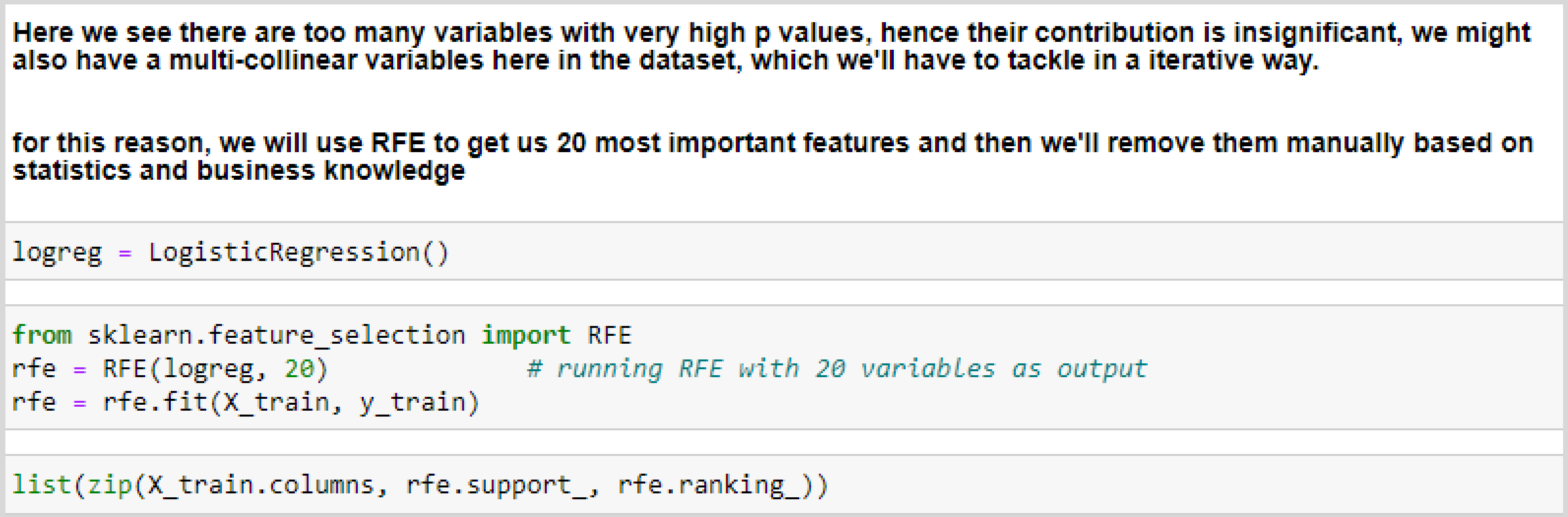
Since this is a classification problem whether we have build a model to predict whether a lead would convert or not, so we use Logistic regression for this problem

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

Generalized Linear Model Regression Results

Dep. Variable:	Converted	**No. Observations:**	6468
Model:	GLM	**Df Residuals:**	6428
Model Family:	Binomial	**Df Model:**	39
Link Function:	logit	**Scale:**	1.0
Method:	IRLS	**Log-Likelihood:**	-2663.3
Date:	Mon, 11 Jan 2021	**Deviance:**	5326.7
Time:	14:18:07	**Pearson chi2:**	8.86e+03
No. Iterations:	22		

RFE



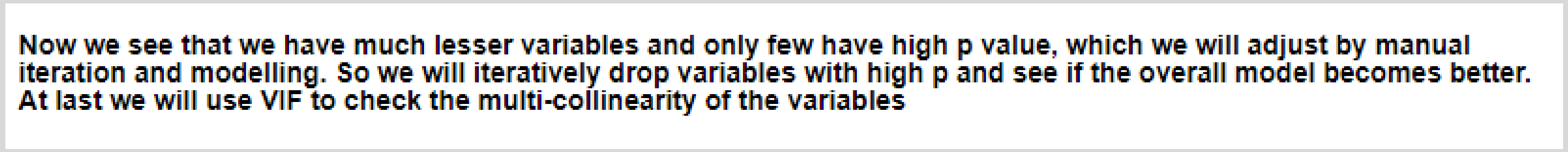
Here we see there are too many variables with very high p values, hence their contribution is insignificant, we might also have a multi-collinear variables here in the dataset, which we'll have to tackle in a iterative way.

for this reason, we will use RFE to get us 20 most important features and then we'll remove them manually based on statistics and business knowledge

```
logreg = LogisticRegression()
```

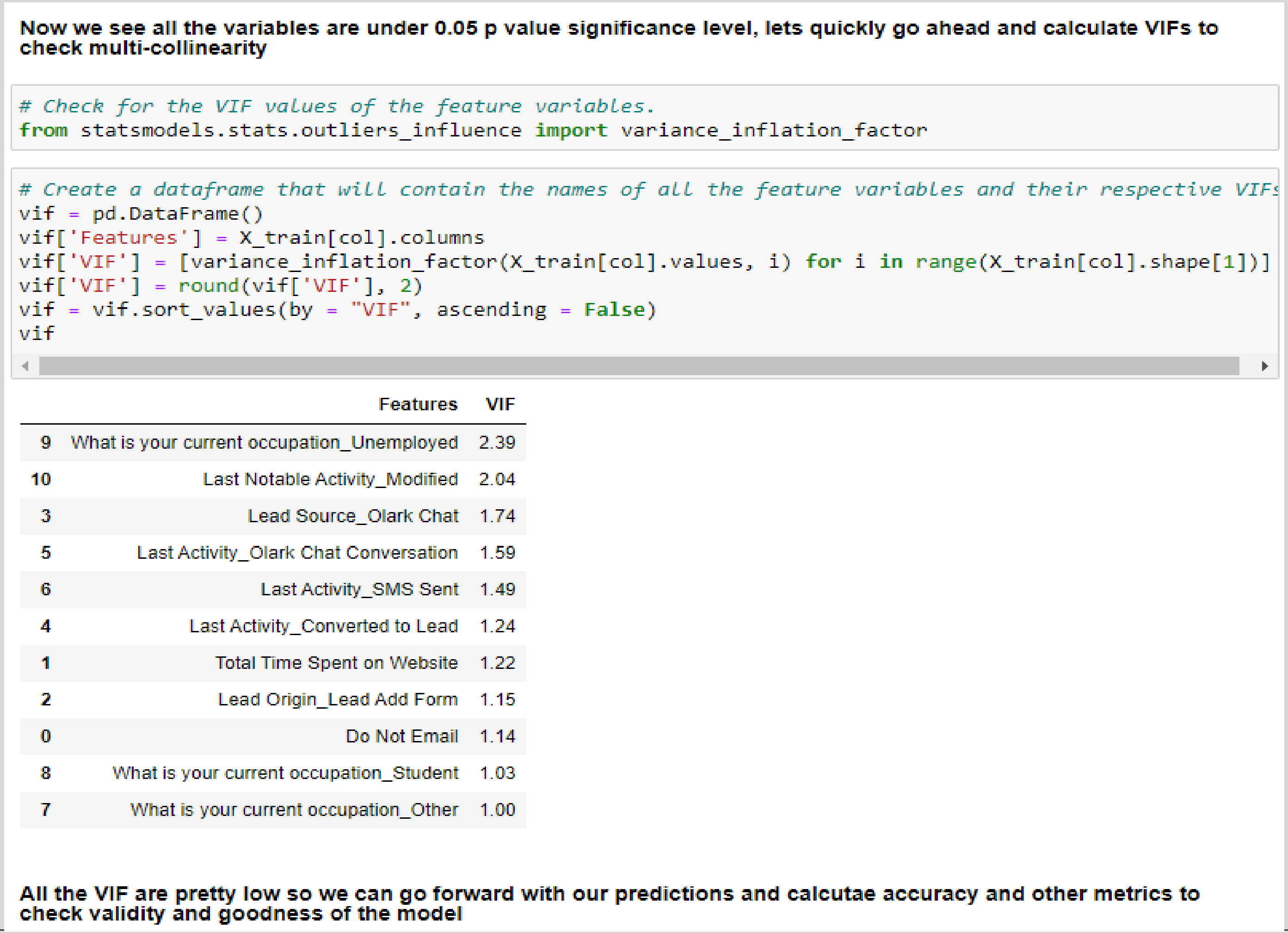
```
from sklearn.feature_selection import RFE
rfe = RFE(logreg, 20)             # running RFE with 20 variables as output
rfe = rfe.fit(X_train, y_train)
```

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



Now we see that we have much lesser variables and only few have high p value, which we will adjust by manual iteration and modelling. So we will iteratively drop variables with high p and see if the overall model becomes better. At last we will use VIF to check the multi-collinearity of the variables

VIF



Now we see all the variables are under 0.05 p value significance level, lets quickly go ahead and calculate VIFs to check multi-collinearity

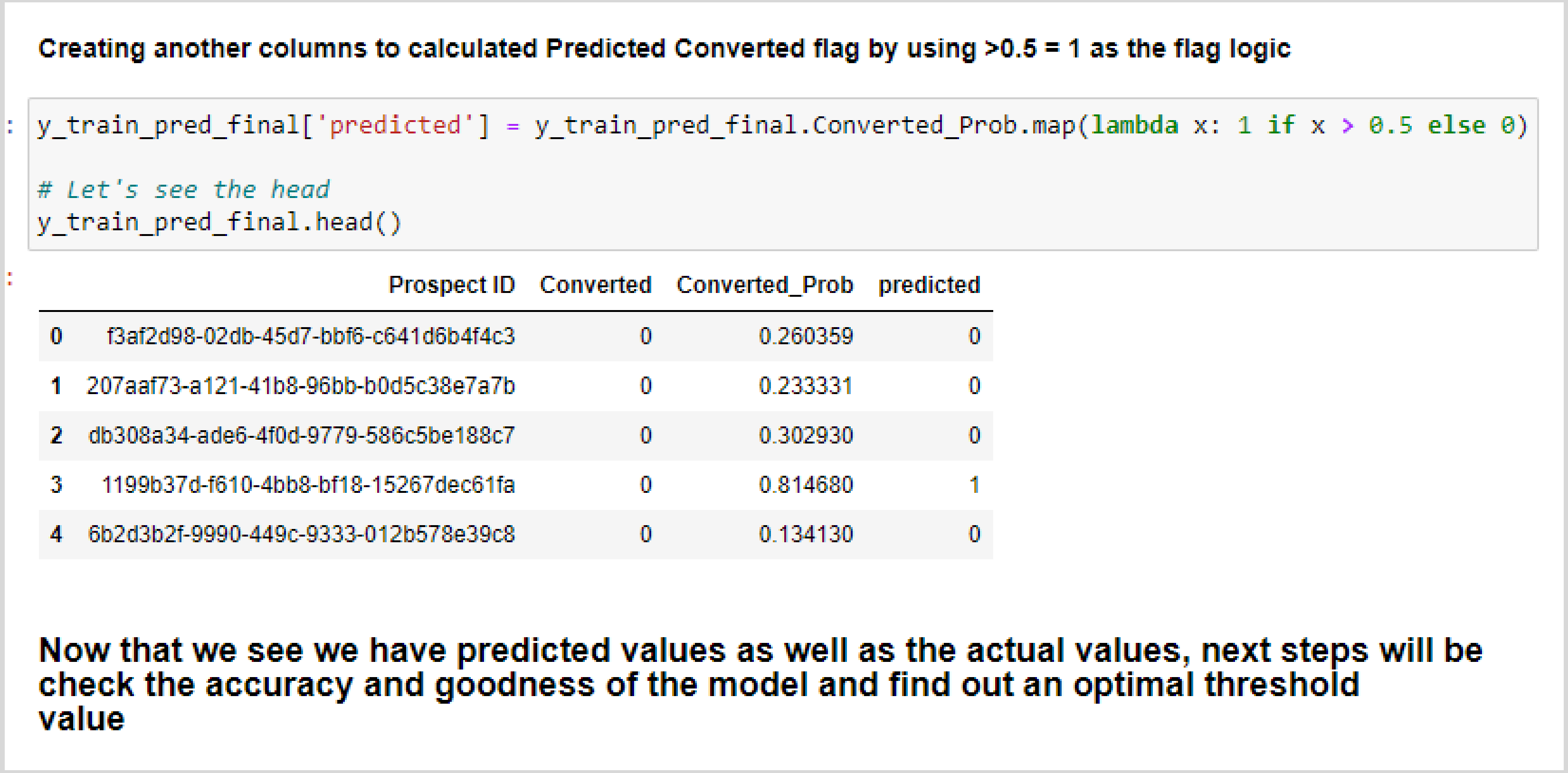
```
# Check for the VIF values of the feature variables.
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
# Create a dataframe that will contain the names of all the feature variables and their respective VIFs
vif = pd.DataFrame()
vif['Features'] = X_train[col].columns
vif['VIF'] = [variance_inflation_factor(X_train[col].values, i) for i in range(X_train[col].shape[1])]
vif['VIF'] = round(vif['VIF'], 2)
vif = vif.sort_values(by = "VIF", ascending = False)
vif
```

	Features	VIF
9	What is your current occupation_Unemployed	2.39
10	Last Notable Activity_Modified	2.04
3	Lead Source_Olark Chat	1.74
5	Last Activity_Olark Chat Conversation	1.59
6	Last Activity_SMS Sent	1.49
4	Last Activity_Converted to Lead	1.24
1	Total Time Spent on Website	1.22
2	Lead Origin_Lead Add Form	1.15
0	Do Not Email	1.14
8	What is your current occupation_Student	1.03
7	What is your current occupation_Other	1.00

All the VIF are pretty low so we can go forward with our predictions and calcutae accuracy and other metrics to check validity and goodness of the model

Threshold – Iteration 1



Creating another columns to calculated Predicted Converted flag by using >0.5 = 1 as the flag logic

```
y_train_pred_final['predicted'] = y_train_pred_final.Converted_Prob.map(lambda x: 1 if x > 0.5 else 0)

# Let's see the head
y_train_pred_final.head()
```

	Prospect ID	Converted	Converted_Prob	predicted
0	f3af2d98-02db-45d7-bbf6-c641d6b4f4c3	0	0.260359	0
1	207aaf73-a121-41b8-96bb-b0d5c38e7a7b	0	0.233331	0
2	db308a34-ade6-4f0d-9779-586c5be188c7	0	0.302930	0
3	1199b37d-f610-4bb8-bf18-15267dec61fa	0	0.814680	1
4	6b2d3b2f-9990-449c-9333-012b578e39c8	0	0.134130	0

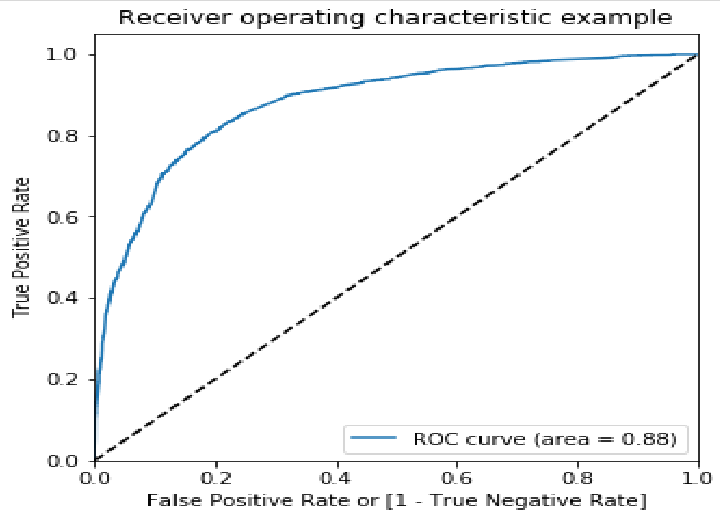
Now that we see we have predicted values as well as the actual values, next steps will be check the accuracy and goodness of the model and find out an optimal threshold value

ROC

ROC Curve

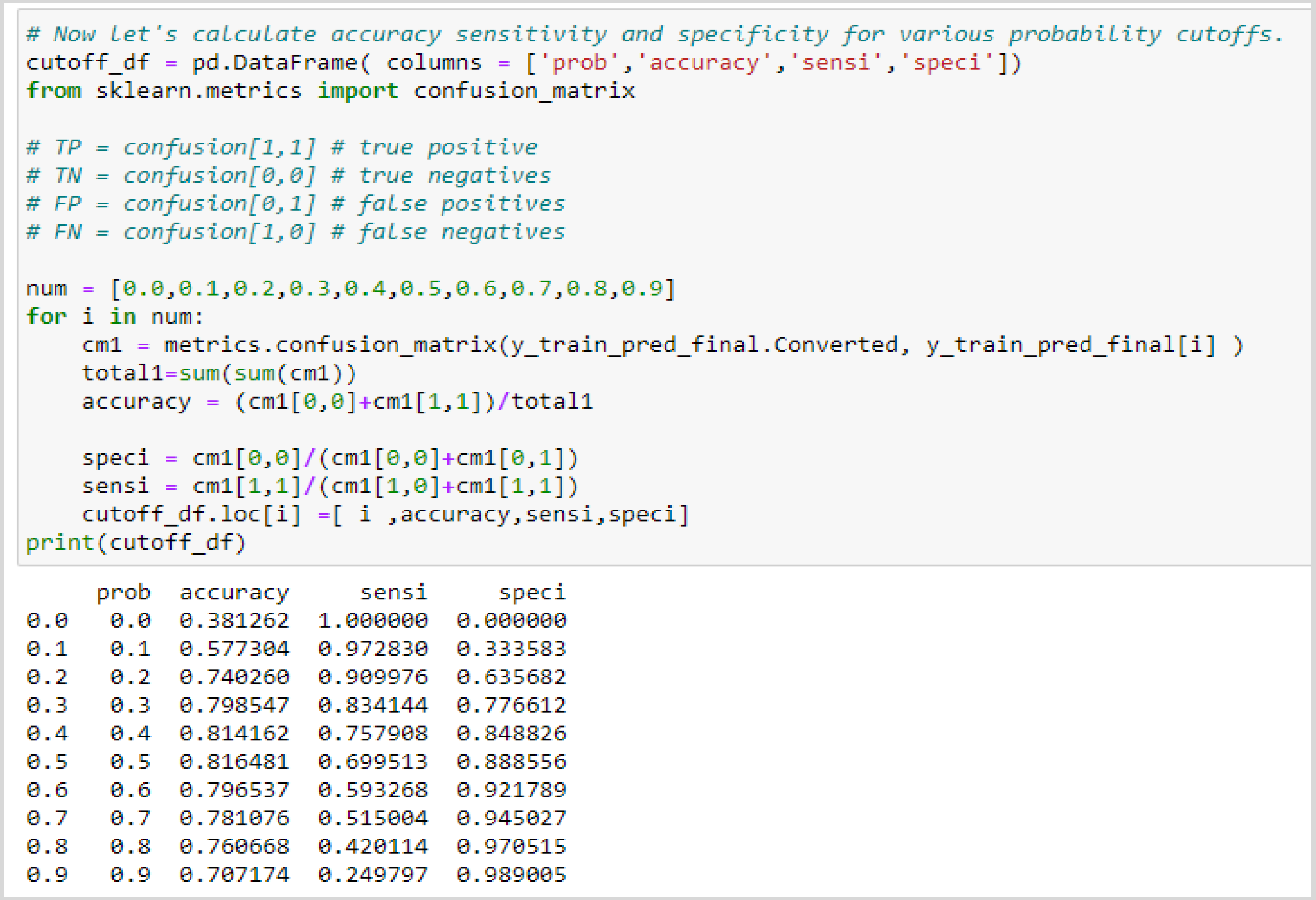
An ROC curve demonstrates several things:

It shows the tradeoff between sensitivity and specificity (any increase in sensitivity will be accompanied by a decrease in specificity). The closer the curve follows the left-hand border and then the top border of the ROC space, the more accurate the test. The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.



The ROC graph looks good and the graph is not close to 45 degree line and shows closeness to true positive rate

Optimal value of threshold

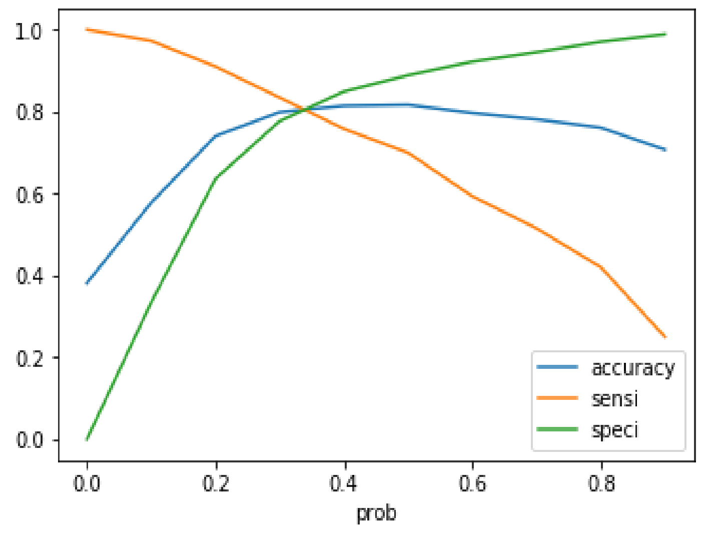


```
# Now let's calculate accuracy sensitivity and specificity for various probability cutoffs.
cutoff_df = pd.DataFrame( columns = ['prob','accuracy','sensi','speci'])
from sklearn.metrics import confusion_matrix

# 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  0.381262  1.000000  0.000000
0.1   0.1  0.577304  0.972830  0.333583
0.2   0.2  0.740260  0.909976  0.635682
0.3   0.3  0.798547  0.834144  0.776612
0.4   0.4  0.814162  0.757908  0.848826
0.5   0.5  0.816481  0.699513  0.888556
0.6   0.6  0.796537  0.593268  0.921789
0.7   0.7  0.781076  0.515004  0.945027
0.8   0.8  0.760668  0.420114  0.970515
0.9   0.9  0.707174  0.249797  0.989005
```

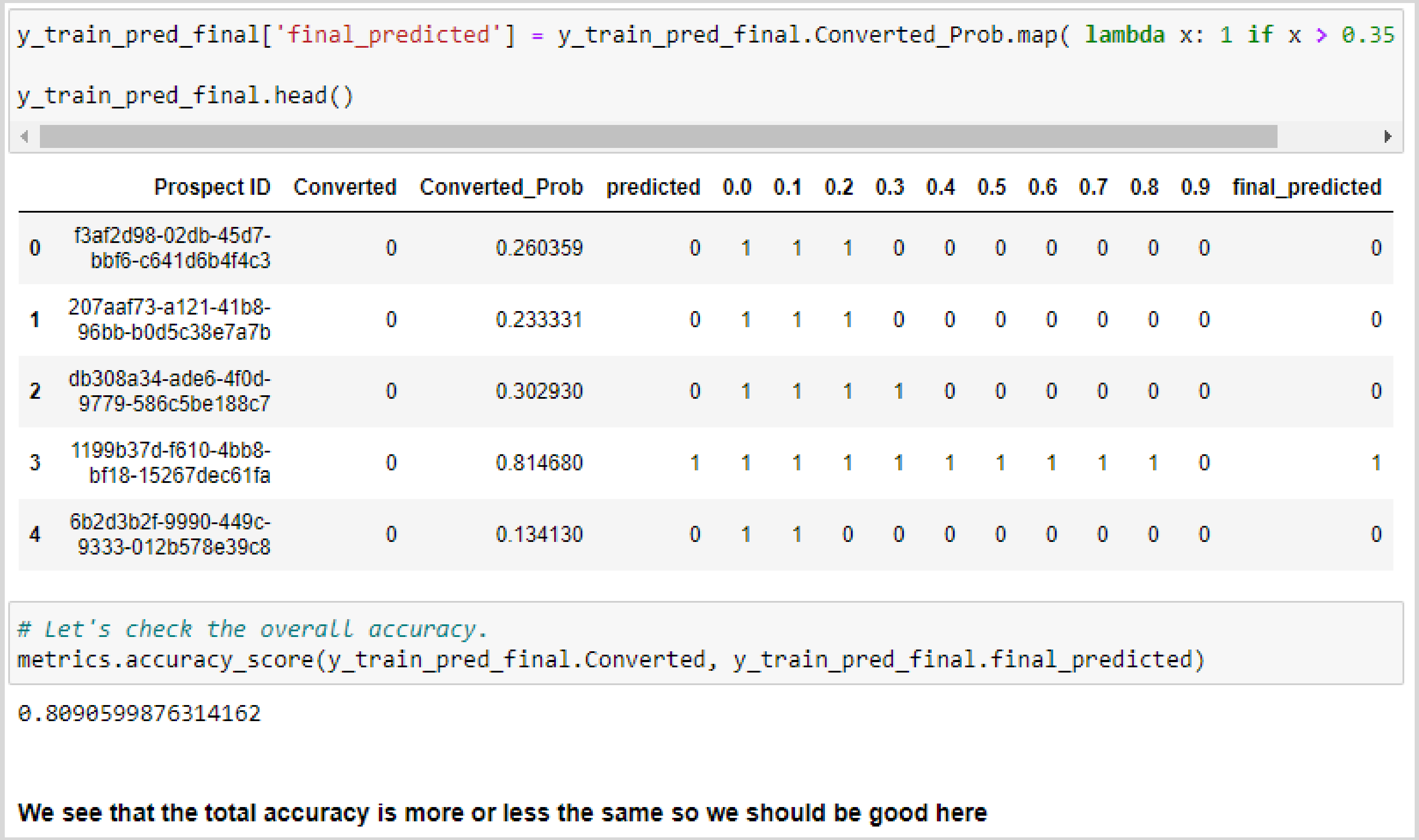
Optimal value of threshold

```
# Let's plot accuracy sensitivity and specificity for various probabilities.
cutoff_df.plot.line(x='prob', y=['accuracy','sensi','speci'])
plt.show()
```



From the above graph and the numbers of specificity and sensitivity we see at 0.35 are they mosyt balanced, so we should use that as our measure and recalculate our metrics and then go to the test data set for goodness of the model

Accuracy with new threshold of 0.35



```
y_train_pred_final['final_predicted'] = y_train_pred_final.Converted_Prob.map( lambda x: 1 if x > 0.35

y_train_pred_final.head()
```

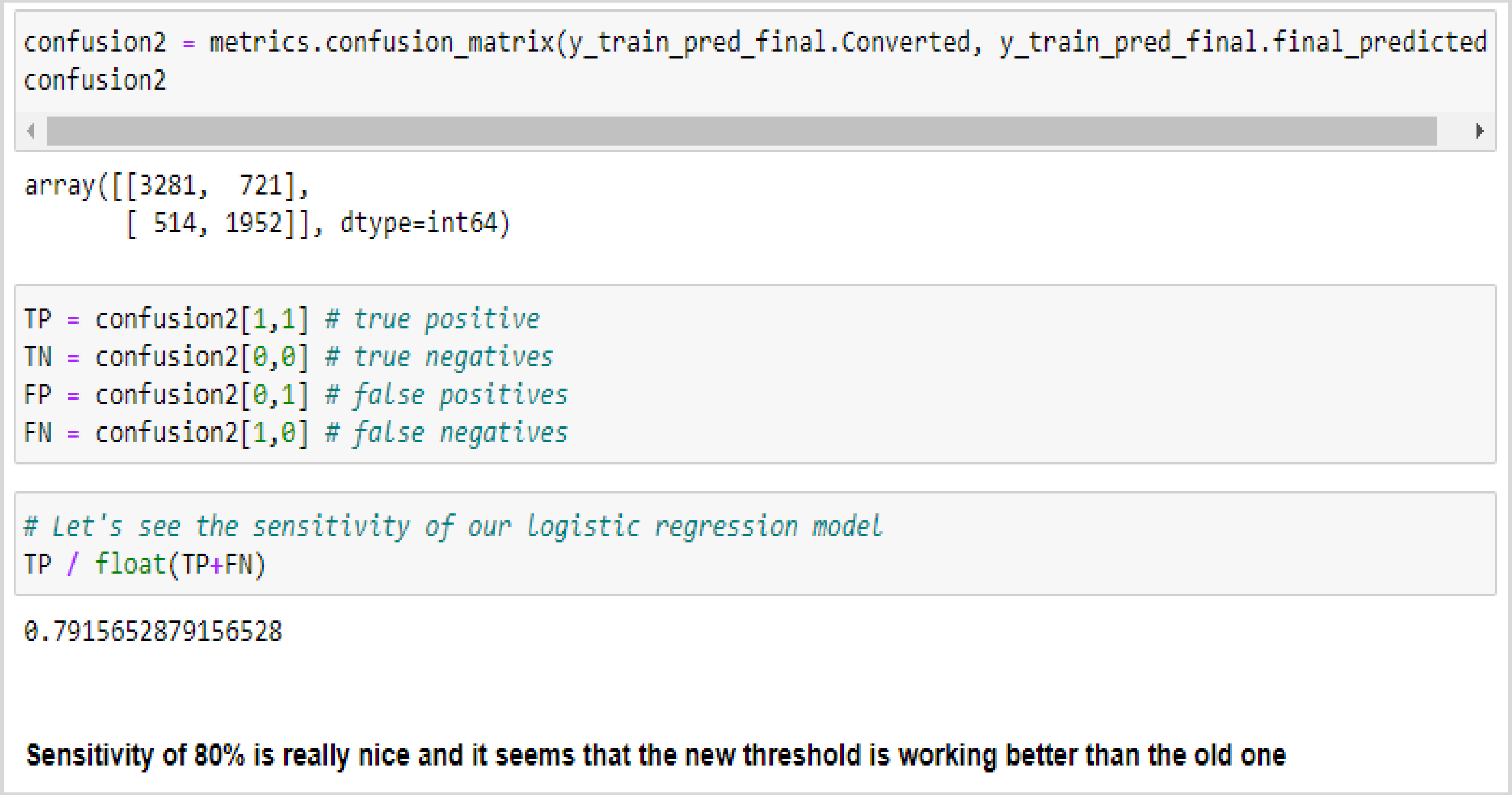
	Prospect ID	Converted	Converted_Prob	predicted	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	final_predicted
0	f3af2d98-02db-45d7-bbf6-c641d6b4f4c3	0	0.260359	0	1	1	1	0	0	0	0	0	0	0	0
1	207aaf73-a121-41b8-96bb-b0d5c38e7a7b	0	0.233331	0	1	1	1	0	0	0	0	0	0	0	0
2	db308a34-ade6-4f0d-9779-586c5be188c7	0	0.302930	0	1	1	1	1	0	0	0	0	0	0	0
3	1199b37d-f610-4bb8-bf18-15267dec61fa	0	0.814680	1	1	1	1	1	1	1	1	1	1	0	1
4	6b2d3b2f-9990-449c-9333-012b578e39c8	0	0.134130	0	1	1	0	0	0	0	0	0	0	0	0

```
# Let's check the overall accuracy.
metrics.accuracy_score(y_train_pred_final.Converted, y_train_pred_final.final_predicted)
```

```
0.8090599876314162
```

We see that the total accuracy is more or less the same so we should be good here

Confusion Matrix



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

```
array([[3281,  721],
       [ 514, 1952]], dtype=int64)
```

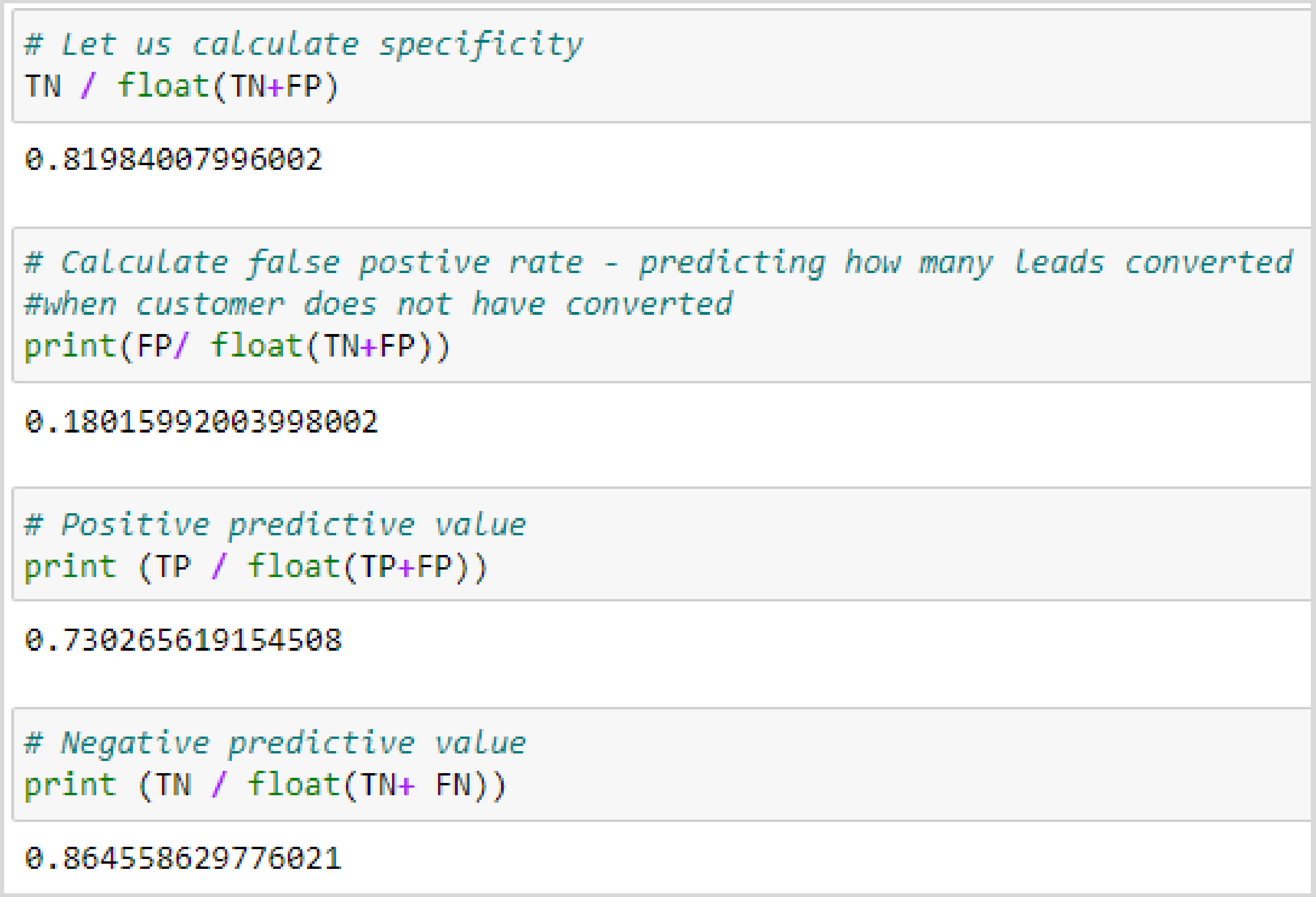
```
TP = confusion2[1,1] # true positive
TN = confusion2[0,0] # true negatives
FP = confusion2[0,1] # false positives
FN = confusion2[1,0] # false negatives
```

```
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

```
0.7915652879156528
```

Sensitivity of 80% is really nice and it seems that the new threshold is working better than the old one

Other metrics



```
# Let us calculate specificity
TN / float(TN+FP)
```

```
0.81984007996002
```

```
# Calculate false postive rate - predicting how many leads converted
#when customer does not have converted
print(FP/ float(TN+FP))
```

```
0.18015992003998002
```

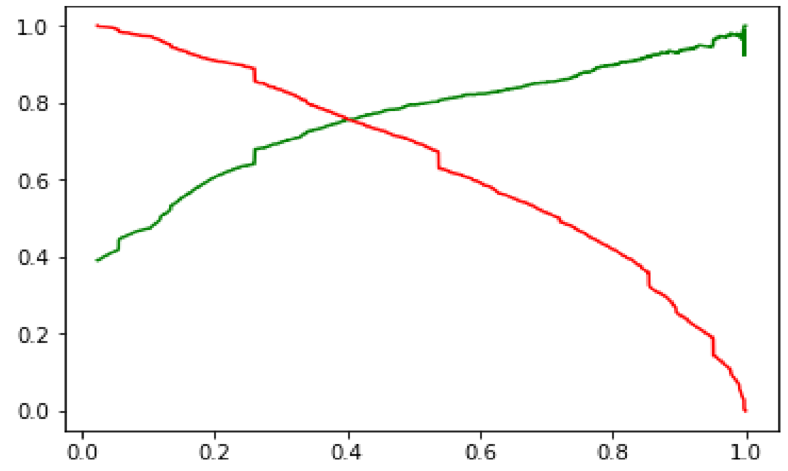
```
# Positive predictive value
print (TP / float(TP+FP))
```

```
0.730265619154508
```

```
# Negative predictive value
print (TN / float(TN+ FN))
```

```
0.864558629776021
```

Recall Precision



```
Recall = TP / float(TP+FN)
Recall
```

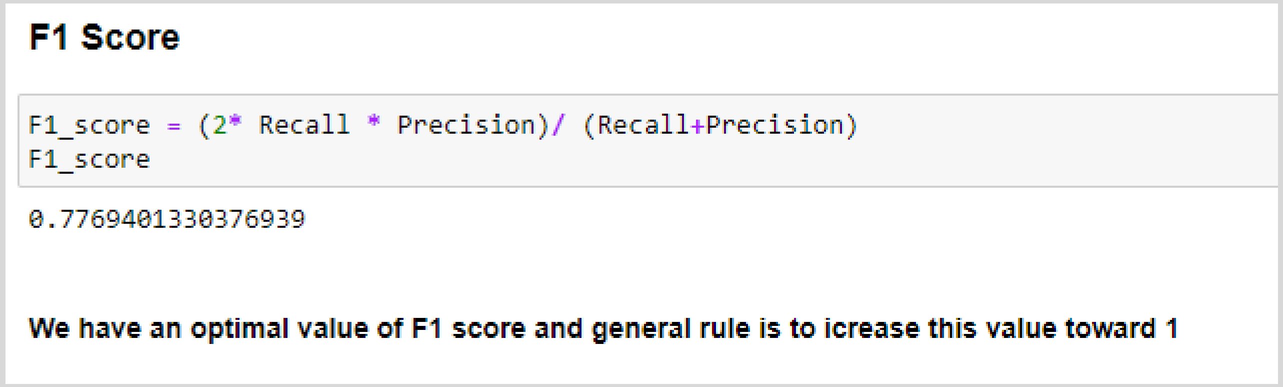
```
0.8
```

Recall is 0.8 which is exactly what the CEO would have wanted as he wanted to convert 80% of his leads

```
Precision = TP / float(TP+FP)
Precision
```

```
0.7551724137931034
```

F1 score



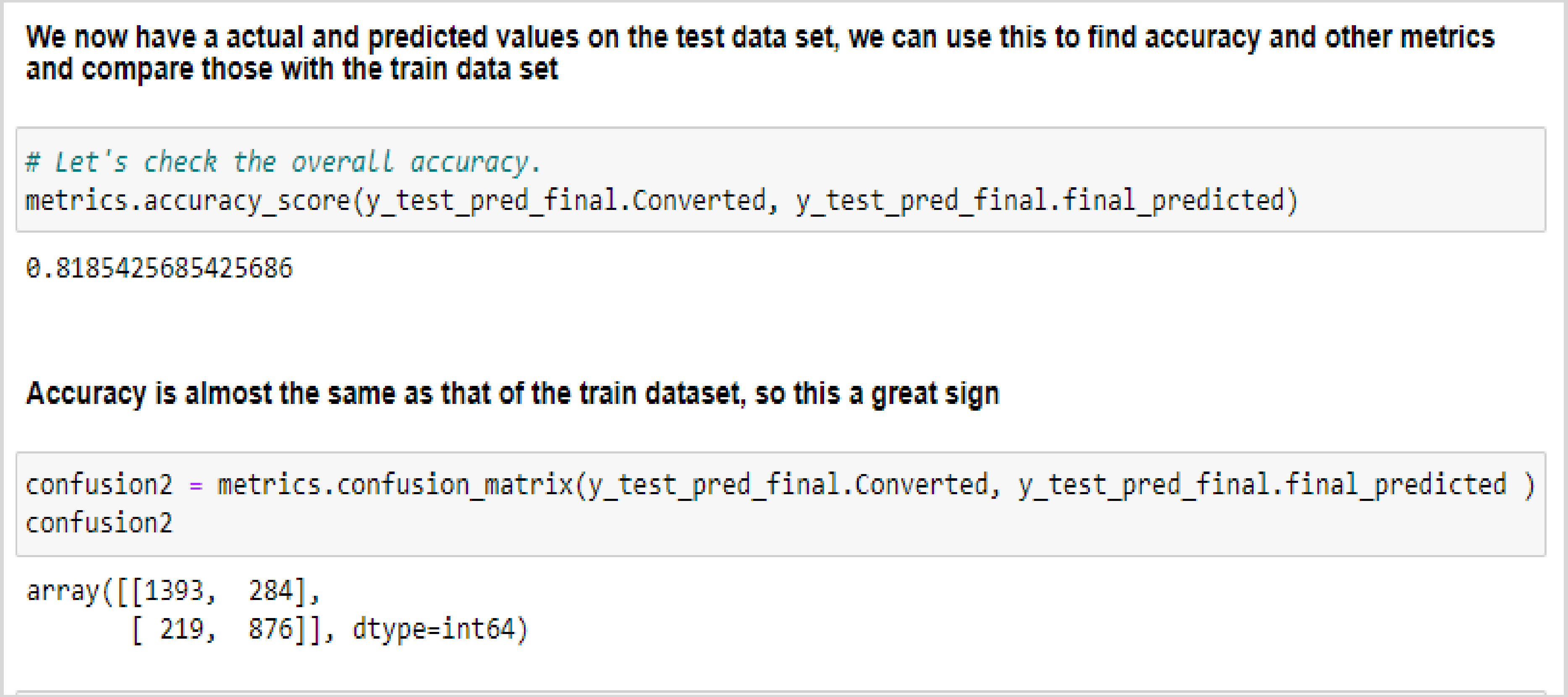
F1 Score

```
F1_score = (2* Recall * Precision)/ (Recall+Precision)
F1_score
```

```
0.7769401330376939
```

We have an optimal value of F1 score and general rule is to icrease this value toward 1

Test Dataset



We now have a actual and predicted values on the test data set, we can use this to find accuracy and other metrics and compare those with the train data set

```
# Let's check the overall accuracy.
metrics.accuracy_score(y_test_pred_final.Converted, y_test_pred_final.final_predicted)
```

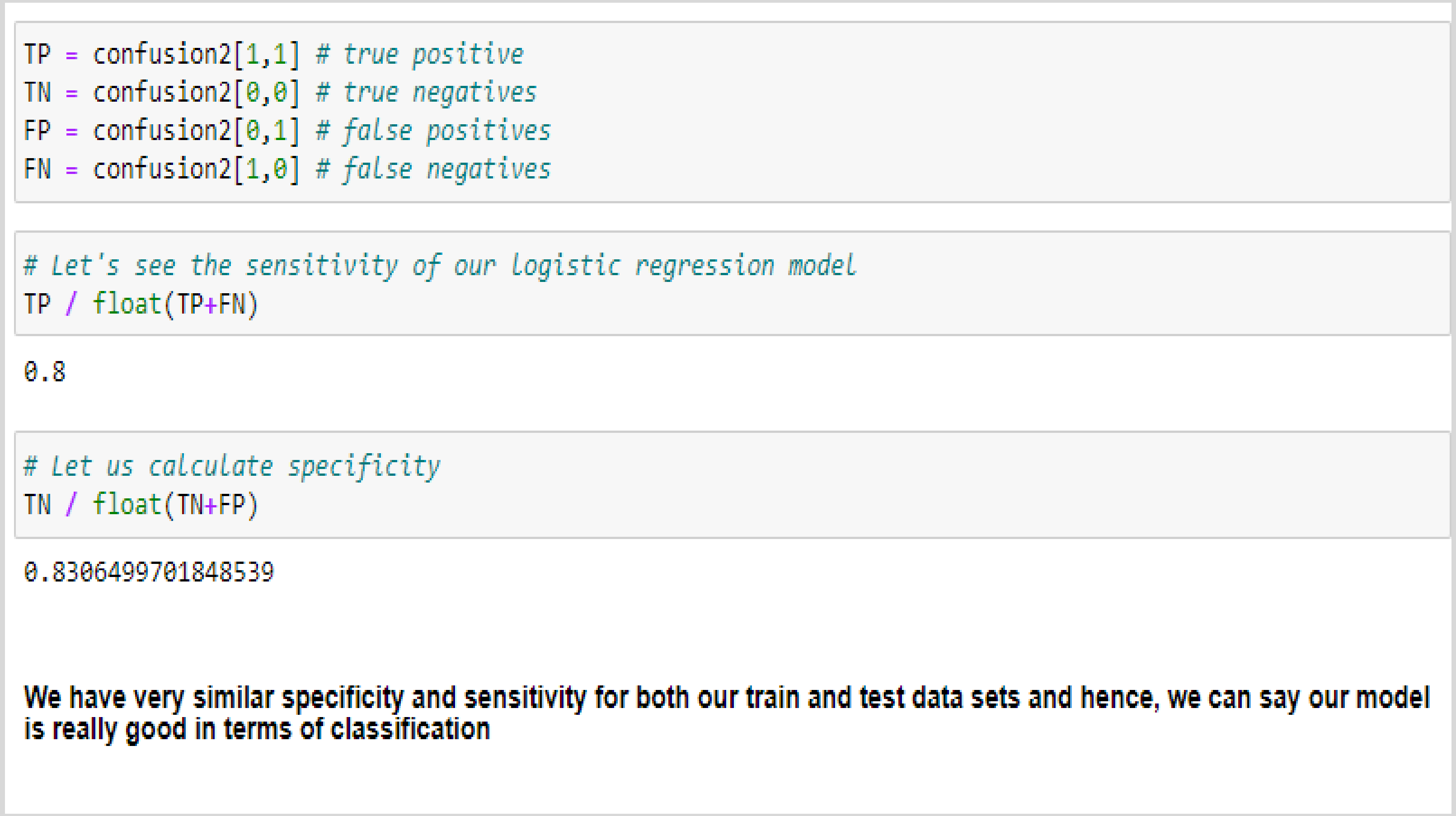
```
0.8185425685425686
```

Accuracy is almost the same as that of the train dataset, so this a great sign

```
confusion2 = metrics.confusion_matrix(y_test_pred_final.Converted, y_test_pred_final.final_predicted )
confusion2
```

```
array([[1393,  284],
       [ 219,  876]], dtype=int64)
```

Test Dataset



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

```
# Let's see the sensitivity of our logistic regression model
TP / float(TP+FN)
```

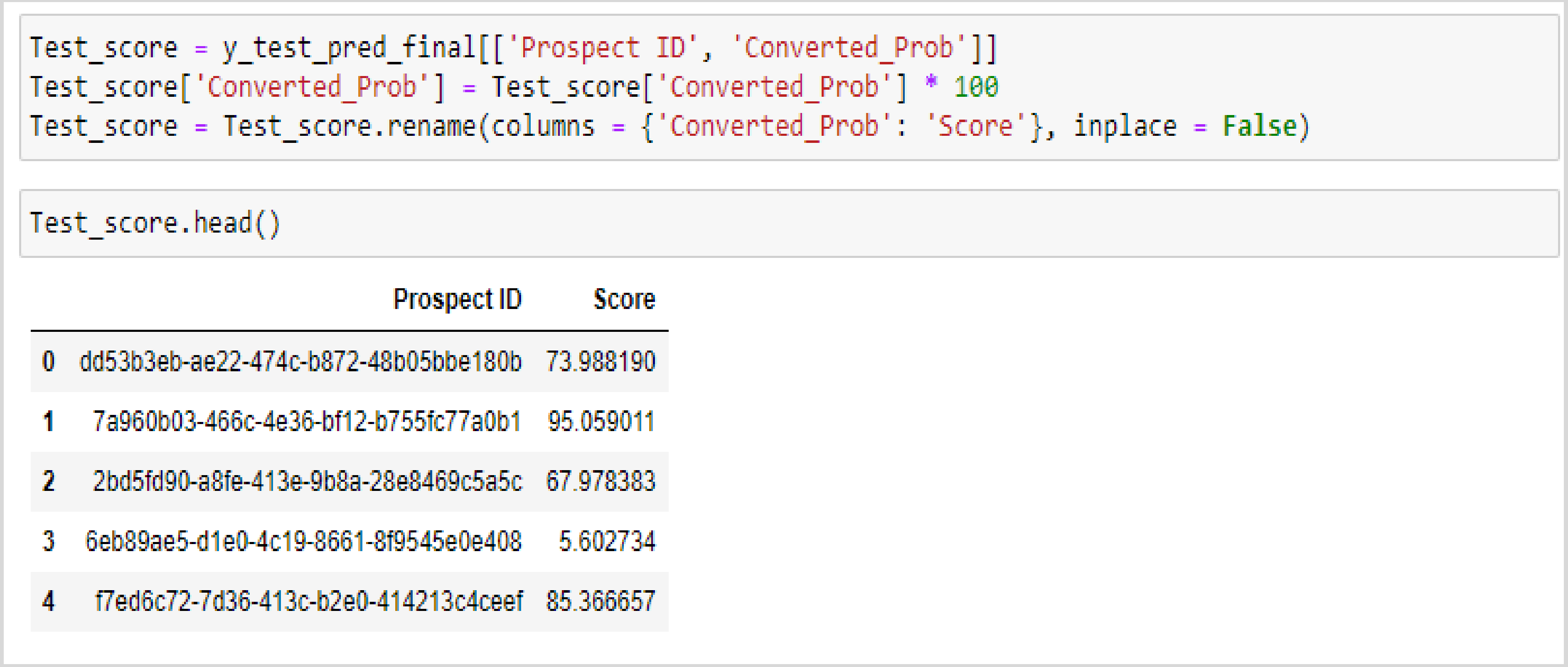
```
0.8
```

```
# Let us calculate specificity
TN / float(TN+FP)
```

```
0.8306499701848539
```

We have very similar specificity and sensitivity for both our train and test data sets and hence, we can say our model is really good in terms of classification

Assigning scores

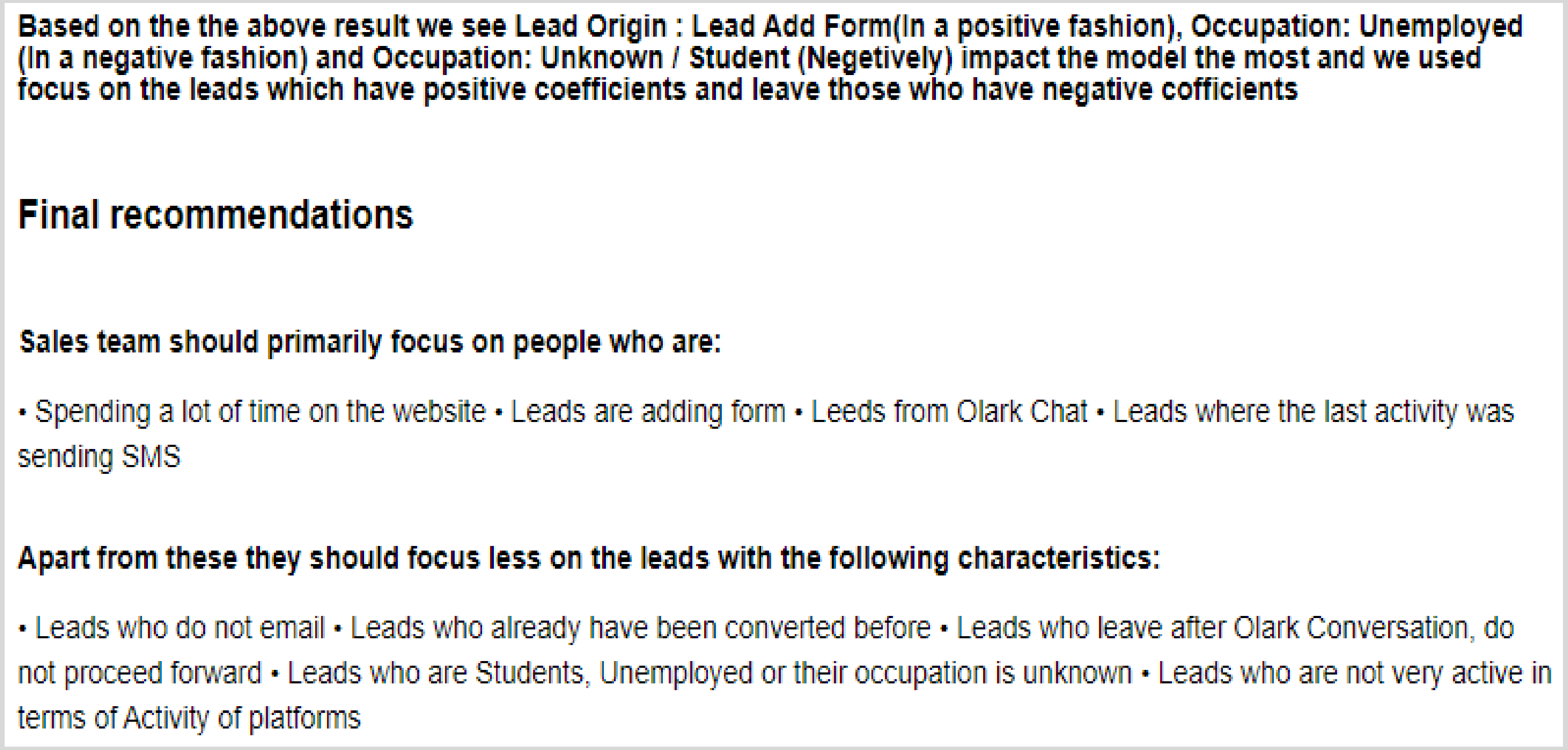


```
Test_score = y_test_pred_final[['Prospect ID', 'Converted_Prob']]
Test_score['Converted_Prob'] = Test_score['Converted_Prob'] * 100
Test_score = Test_score.rename(columns = {'Converted_Prob': 'Score'}, inplace = False)
```

```
Test_score.head()
```

```
                            Prospect ID      Score
0  dd53b3eb-ae22-474c-b872-48b05bbe180b  73.988190
1  7a960b03-466c-4e36-bf12-b755fc77a0b1  95.059011
2  2bd5fd90-a8fe-413e-9b8a-28e8469c5a5c  67.978383
3  6eb89ae5-d1e0-4c19-8661-8f9545e0e408   5.602734
4  f7ed6c72-7d36-413c-b2e0-414213c4ceef  85.366657
```

Highest impacting Features and Recommendations



Based on the the above result we see Lead Origin : Lead Add Form(In a positive fashion), Occupation: Unemployed (In a negative fashion) and Occupation: Unknown / Student (Negetively) impact the model the most and we used focus on the leads which have positive coefficients and leave those who have negative cofficients

Final recommendations

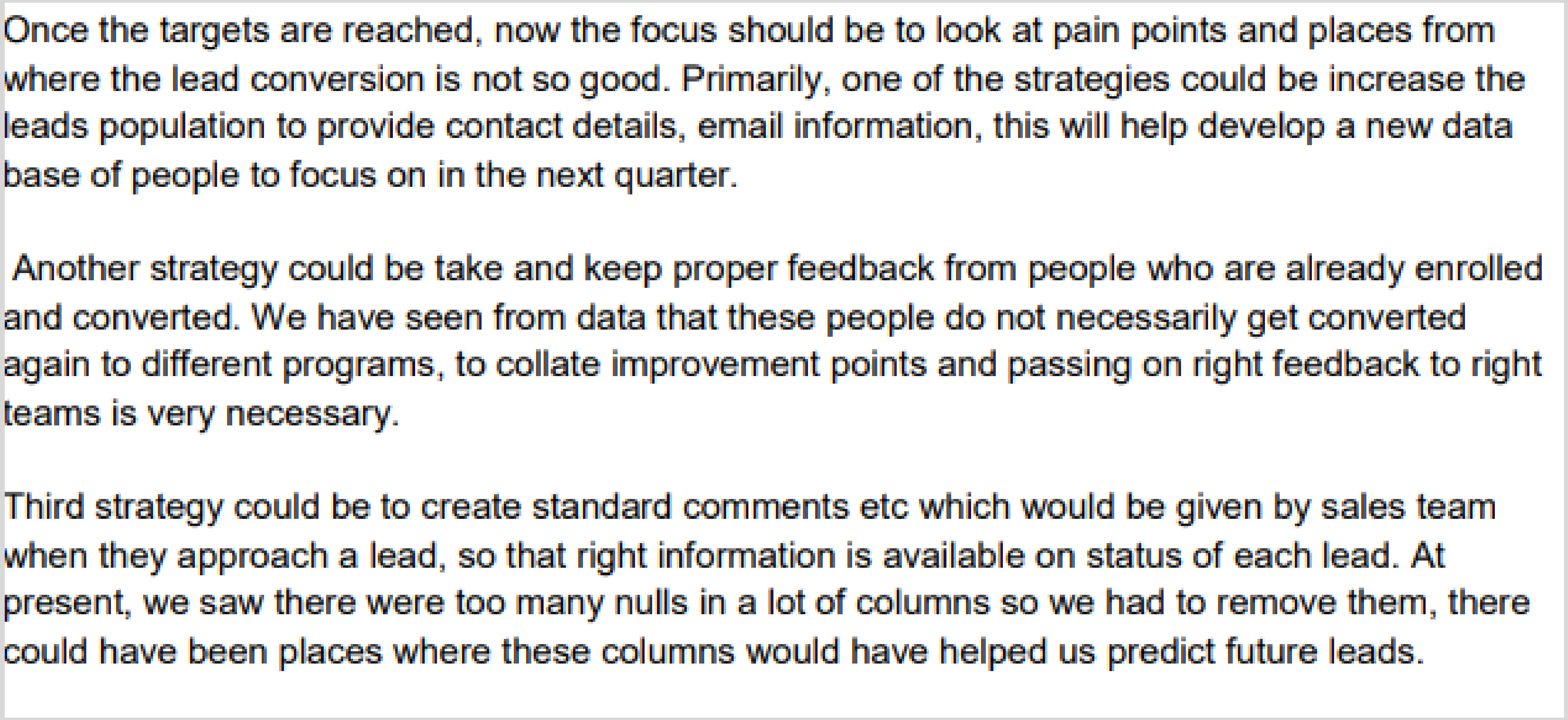
Sales team should primarily focus on people who are:

• Spending a lot of time on the website • Leads are adding form • Leeds from Olark Chat • Leads where the last activity was sending SMS

Apart from these they should focus less on the leads with the following characteristics:

• Leads who do not email • Leads who already have been converted before • Leads who leave after Olark Conversation, do not proceed forward • Leads who are Students, Unemployed or their occupation is unknown • Leads who are not very active in terms of Activity of platforms

Strategy to CEO



Once the targets are reached, now the focus should be to look at pain points and places from where the lead conversion is not so good. Primarily, one of the strategies could be increase the leads population to provide contact details, email information, this will help develop a new data base of people to focus on in the next quarter.

Another strategy could be take and keep proper feedback from people who are already enrolled and converted. We have seen from data that these people do not necessarily get converted again to different programs, to collate improvement points and passing on right feedback to right teams is very necessary.

Third strategy could be to create standard comments etc which would be given by sales team when they approach a lead, so that right information is available on status of each lead. At present, we saw there were too many nulls in a lot of columns so we had to remove them, there could have been places where these columns would have helped us predict future leads.