Summary: This Model is for an Insurance company sales team. The team is given leads from their existing customer base, predominantly health insurance customers. With out leads prioritization, they have to contact all the leads which is in-effcient. With this model, we were able to reduce the contactable leads by 50% with out leaving any potential conversions.

```
We developed a lead prioritization model.

In test data, we have total 83990 leads. With out the help of prioritization
model, the sales team will have to
contact all these leads with a conversion rate of 13.6 % conversion rate.

With the new prioritization model the conversion rate incresed by 100% to 26%.
This will weed away 50% of leads form the sales leads.
```

In [106…
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import QuantileTransformer
from sklearn.metrics import roc_auc_score, confusion_matrix, precision_recall_curve, (
precision_score,recall_score, roc_curve, RocCurveDisplay
from sklearn.linear_model import LogisticRegression
```

In [107…
```python
df = pd.read_csv('./train.csv')
```

In [108…
```python
df.head(10)
```

Out[108…

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_P
0	1	Male	44	1	28.0	0	> 2 Years	Yes	
1	2	Male	76	1	3.0	0	1-2 Year	No	
2	3	Male	47	1	28.0	0	> 2 Years	Yes	
3	4	Male	21	1	11.0	1	< 1 Year	No	
4	5	Female	29	1	41.0	1	< 1 Year	No	
5	6	Female	24	1	33.0	0	< 1 Year	Yes	
6	7	Male	23	1	11.0	0	< 1 Year	Yes	
7	8	Female	56	1	28.0	0	1-2 Year	Yes	
8	9	Female	24	1	3.0	1	< 1 Year	No	
9	10	Female	32	1	6.0	1	< 1 Year	No	

In [109…
```python
df.drop(columns=['id'],inplace=True)
```

In [110…
```python
list_predictors = list(df.columns)
list_predictors.remove('Response')
```

Data Processing

In [111…
```python
def extract_numeric_ordinals(data_frame,columns):
    numeric_columns = []
    ordinal_columns = []
    for column in columns:
        if df[column].dtype !='O':
            if len(list(df[column].unique()))/df.shape[0] < 0.01:
                ordinal_columns.append(column)
            else:
                numeric_columns.append(column)
    return(numeric_columns, ordinal_columns)
```

In [112…
```python
def extract_categoricals(data_frame,columns):
    categorical_columns = []
    for column in columns:
        if df[column].dtype == 'O':
            categorical_columns.append(column)
    return(categorical_columns)
```

In [113…
```python
def categorical_ordinal_convert_to_ranked_numerical(dataframe,ordinal_columns, catego
    complete_list = ordinal_columns+categorical_columns
    for column in complete_list:
        S = pd.DataFrame(dataframe.groupby(column)[target].mean().sort_values(ascendi
        S.reset_index(inplace=True)
        S['rank'] = S[target].rank(pct=True)
        map_dict = dict(zip(list(S[column]), list(S['rank'])))
        df[column+'ranked_ordinal'] = df[column].map(map_dict)
        df.drop(columns=[column],inplace=True)
    return(dataframe)
```

In [114…
```python
numeric_columns, ordinal_columns = extract_numeric_ordinals(df,list_predictors)
```

In [115…
```python
categorical_columns = extract_categoricals(df,list(df.columns))
```

In [116…
```python
dataframe_processed = categorical_ordinal_convert_to_ranked_numerical(df, ordinal_colu
```

In [117…
```python
ordinal_columns
```

Out[117…
```
['Age',
 'Driving_License',
 'Region_Code',
 'Previously_Insured',
 'Policy_Sales_Channel',
 'Vintage']
```

In [118…
```python
dataframe_processed
```

Out[118…

	Annual_Premium	Response	Ageranked_ordinal	Driving_Licenseranked_ordinal	Region_Coderanked_ordi
0	40454.0	1	0.909091	1.0	0.981
1	33536.0	0	0.303030	1.0	0.830
2	38294.0	1	0.939394	1.0	0.981
3	28619.0	0	0.075758	1.0	0.660
4	27496.0	0	0.439394	1.0	0.735
...	...	...	...	...	
381104	30170.0	0	0.318182	1.0	0.226
381105	40016.0	0	0.515152	1.0	0.339
381106	35118.0	0	0.075758	1.0	0.245
381107	44617.0	0	0.393939	1.0	0.471
381108	41777.0	0	0.893939	1.0	0.754

381109 rows × 11 columns

Statistical Preprocessing

In [119…
```python
list_predictors = list(dataframe_processed.columns)
list_predictors.remove('Response')
```

In [120…
```python
X_predictors = df[list_predictors].to_numpy()
Y_response = df['Response'].to_numpy()
```

In [121…
```python
quantile_transformer = QuantileTransformer()
quantile_transformer.fit(X_predictors)
X_predictors_transform = quantile_transformer.transform(X_predictors)
```

In [122…
```python
X_train, X_test, Y_train , Y_test = train_test_split(X_predictors_transform, Y_respons
                                                     stratify=Y_response, test_size=0.2
                                                     random_state = 42)
```

Logistic Regression Model

In [123…
```python
LogisticModel = LogisticRegression()
C_grid=[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
GridSearchModel = GridSearchCV(estimator=LogisticModel, param_grid={'C':C_grid},
                               scoring='roc_auc',cv =5 )
GridSearchModel.fit(X_train, Y_train)
```

Out[123…
```
GridSearchCV(cv=5, estimator=LogisticRegression(),
             param_grid={'C': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]},
             scoring='roc_auc')
```

In [124…
```python
print(" The best training score:{:.5f} for the model is available at param value:{:.2
                                                     format(GridSearchModel.best_sco
```

```
 The best training score:0.84661 for the model is available at param value:1.00
```

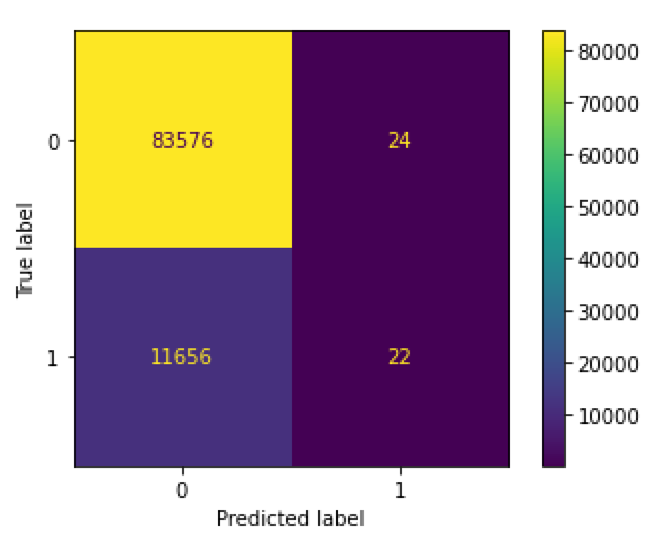
In [125…
```python
print(" The Testscore:{:.5f} ".\
      format(roc_auc_score(Y_test,  GridSearchModel.predict_proba(X_test)[:,
```

```
 The Testscore:0.84888
```

The model has comparable scores between training and test so there is no signs of over fitting.

In [126…
```python
# Testing on validationSet
Y_test_predicted = GridSearchModel.predict(X_test)
cm = confusion_matrix(Y_test, Y_test_predicted)
ConfusionMatrixDisplay(cm).plot()
```

Out[126…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f44a3bbbe0>
```



In [127…
```python
print("Precision:{:.2f} and Recall{:.5f} for this Model".format(precision_score(Y_test
                                                                 recall_score(Y_test, Y_test_p
```

```
Precision:0.48 and Recall0.00188 for this Model
```

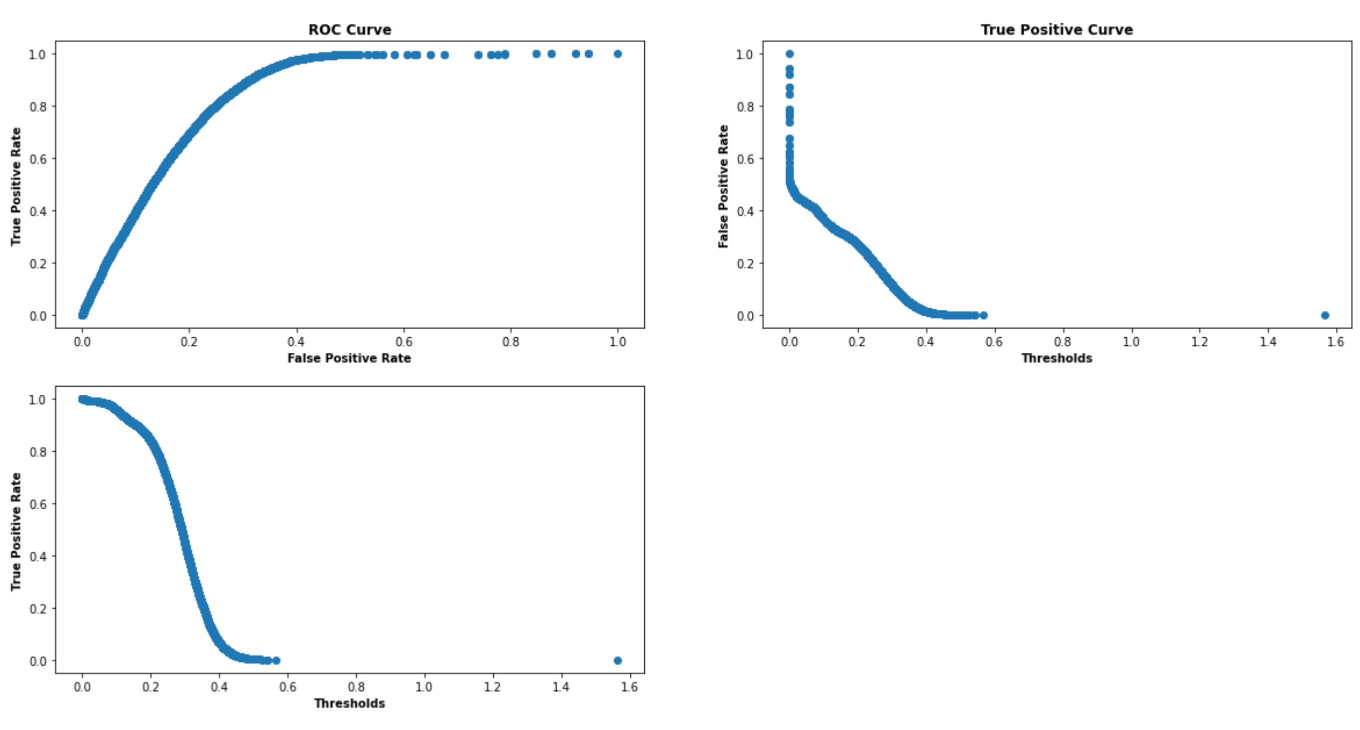
The Model has AUC score of 0.85 for both test and training sets How ever, at the default probability threshold of 0.5, the precision and re-call score are not sasticfactory.

To modify the probability threshold, we will plot the ROC curve and obtain the threshold that gives best true positive rate with least possibile true negative rate.

Model Diagnostics

In [128…
```python
fpr, tpr, thresholds = roc_curve(Y_test,  GridSearchModel.predict_proba(X_test)[:, 1])
fig1 = plt.figure(figsize=(20,10))
ax1 = fig1.add_subplot(2,2,1)
ax2 = fig1.add_subplot(2,2,2)
ax3 = fig1.add_subplot(2,2,3)
ax1.scatter(x = fpr, y = tpr)
ax1.set_xlabel("False Positive Rate", weight='bold')
ax1.set_ylabel("True Positive Rate", weight='bold')
ax1.set_title("ROC Curve", weight='bold')
ax2.scatter(x = thresholds, y = fpr)
ax2.set_xlabel("Thresholds", weight='bold')
ax2.set_ylabel("False Positive Rate", weight='bold')
ax2.set_title("True Positive Curve", weight='bold')
ax3.scatter(x = thresholds, y = tpr)
ax3.set_xlabel("Thresholds", weight='bold')
ax3.set_ylabel("True Positive Rate", weight='bold')
ax2.set_title("True Positive Curve", weight='bold')
```

Out[128…
```
Text(0.5, 1.0, 'True Positive Curve')
```



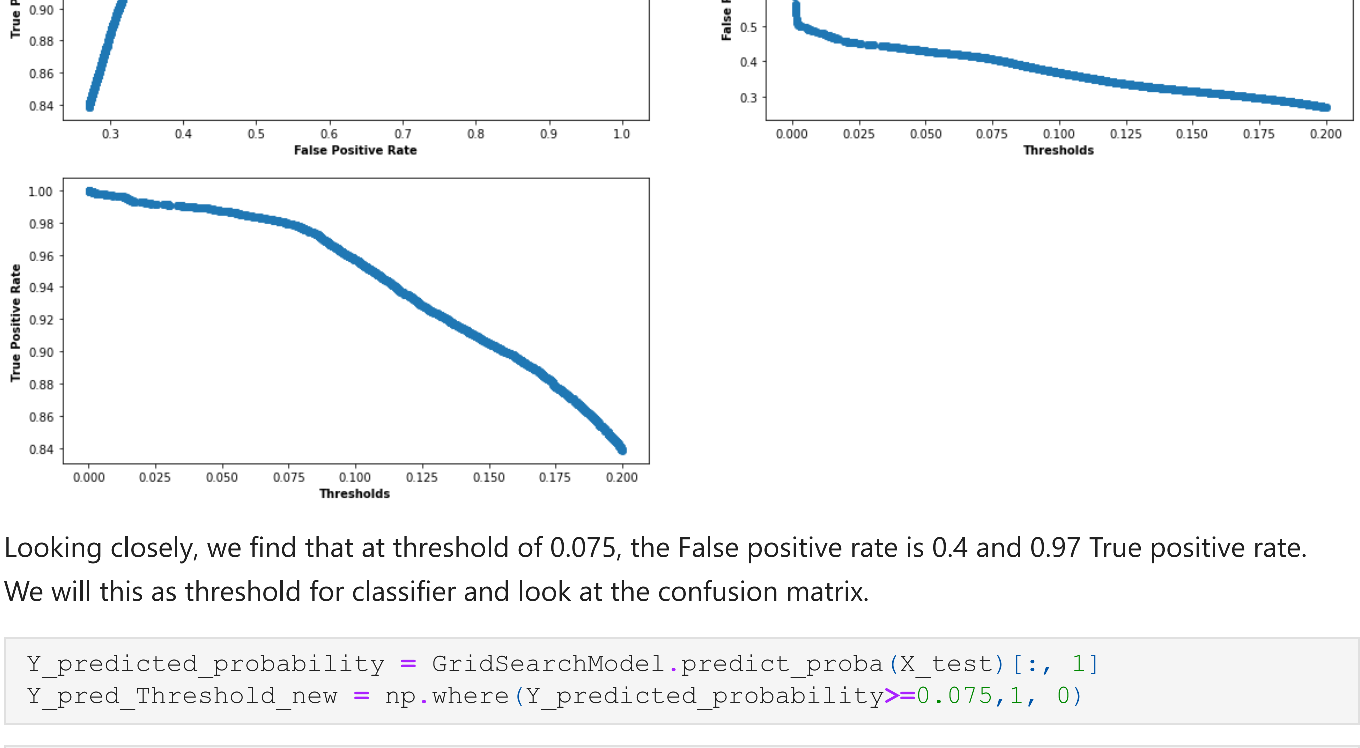
The goal of machine learning model is to balance between True positive rate and false positive rate. As true positive rate increases,which is desired, the false positive rate increases too. So we should determine the threshold that maximizes True positive rate with the least false positive rate.

The ROC surve is flattening at FPR of 0.4. This means 40% of label zero will be labelled as positive. Given that this is a lead priorotisation model, this threshold eliminaated 60% of the leads while ensuring that all the potential conversions are captured.

In [129…
```python
# Exploring Thresholds below 0.2 for more focused TPR and FPR.

fpr, tpr, thresholds = roc_curve(Y_test,  GridSearchModel.predict_proba(X_test)[:, 1])
df_performance  = pd.DataFrame({'FalsePositiveRate':fpr, 'TruePositiveRate':tpr, 'Thre
df_performance = df_performance[df_performance["Thresholds"].between(0,0.2)]
fig1 = plt.figure(figsize=(20,10))
ax1 = fig1.add_subplot(2,2,1)
ax2 = fig1.add_subplot(2,2,2)
ax3 = fig1.add_subplot(2,2,3)
ax1.scatter(x = df_performance["FalsePositiveRate"], y = df_performance["TruePositiveR
ax1.set_xlabel("False Positive Rate", weight='bold')
ax1.set_ylabel("True Positive Rate", weight='bold')
ax1.set_title("ROC Curve", weight='bold')
ax2.scatter(x = df_performance["Thresholds"], y = df_performance["FalsePositiveRate"],
ax2.set_xlabel("Thresholds", weight='bold')
ax2.set_ylabel("False Positive Rate", weight='bold')
ax2.set_title("True Positive Curve", weight='bold')
ax3.scatter(x = df_performance["Thresholds"], y = df_performance["TruePositiveRate"],
ax3.set_xlabel("Thresholds", weight='bold')
ax3.set_ylabel("True Positive Rate", weight='bold')
ax2.set_title("True Positive Curve", weight='bold')
```

Out[129…
```
Text(0.5, 1.0, 'True Positive Curve')
```



Looking closely, we find that at threshold of 0.075, the False positive rate is 0.4 and 0.97 True positive rate. We will this as threshold for classifier and look at the confusion matrix.

In [130…
```python
Y_predicted_probability = GridSearchModel.predict_proba(X_test)[:, 1]
Y_pred_Threshold_new = np.where(Y_predicted_probability>=0.075,1, 0)
```

In [131…
```python
cm = confusion_matrix(Y_test, Y_pred_Threshold_new)
ConfusionMatrixDisplay(cm).plot()
```

Out[131…
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1f4446fc5e0>
```