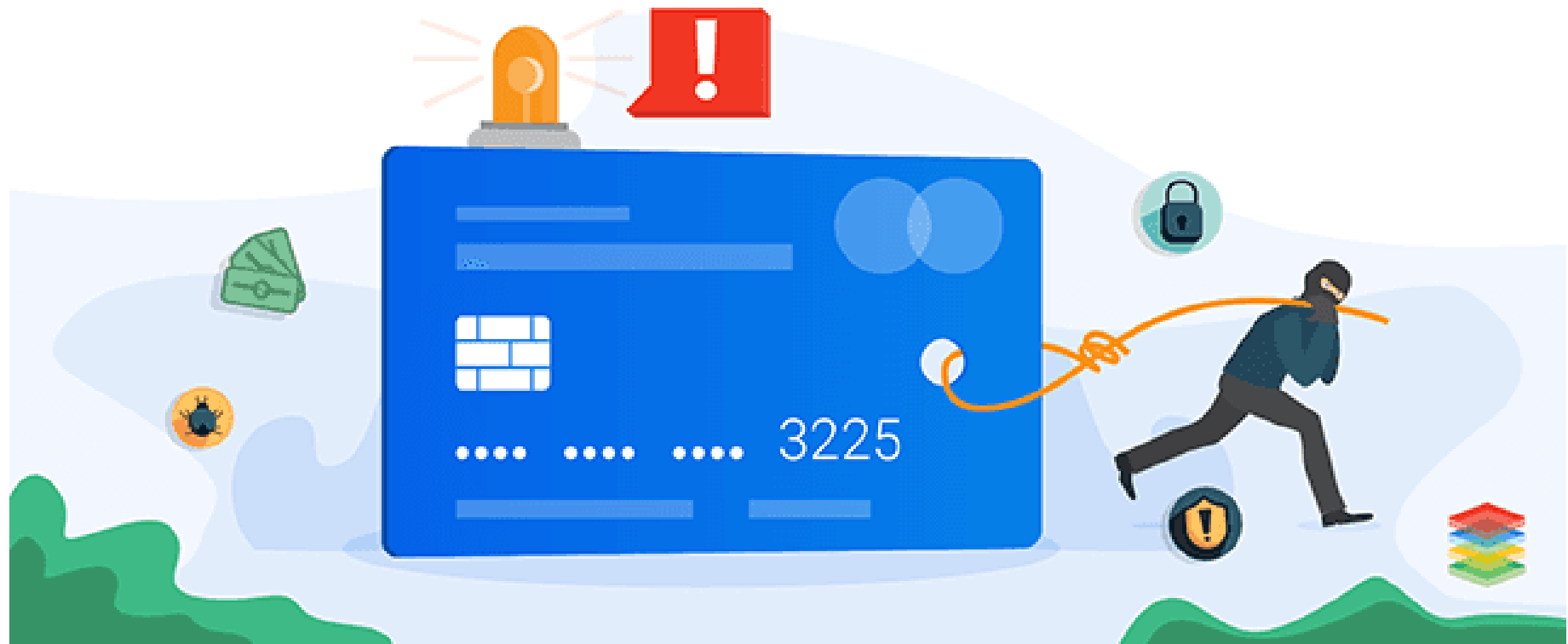


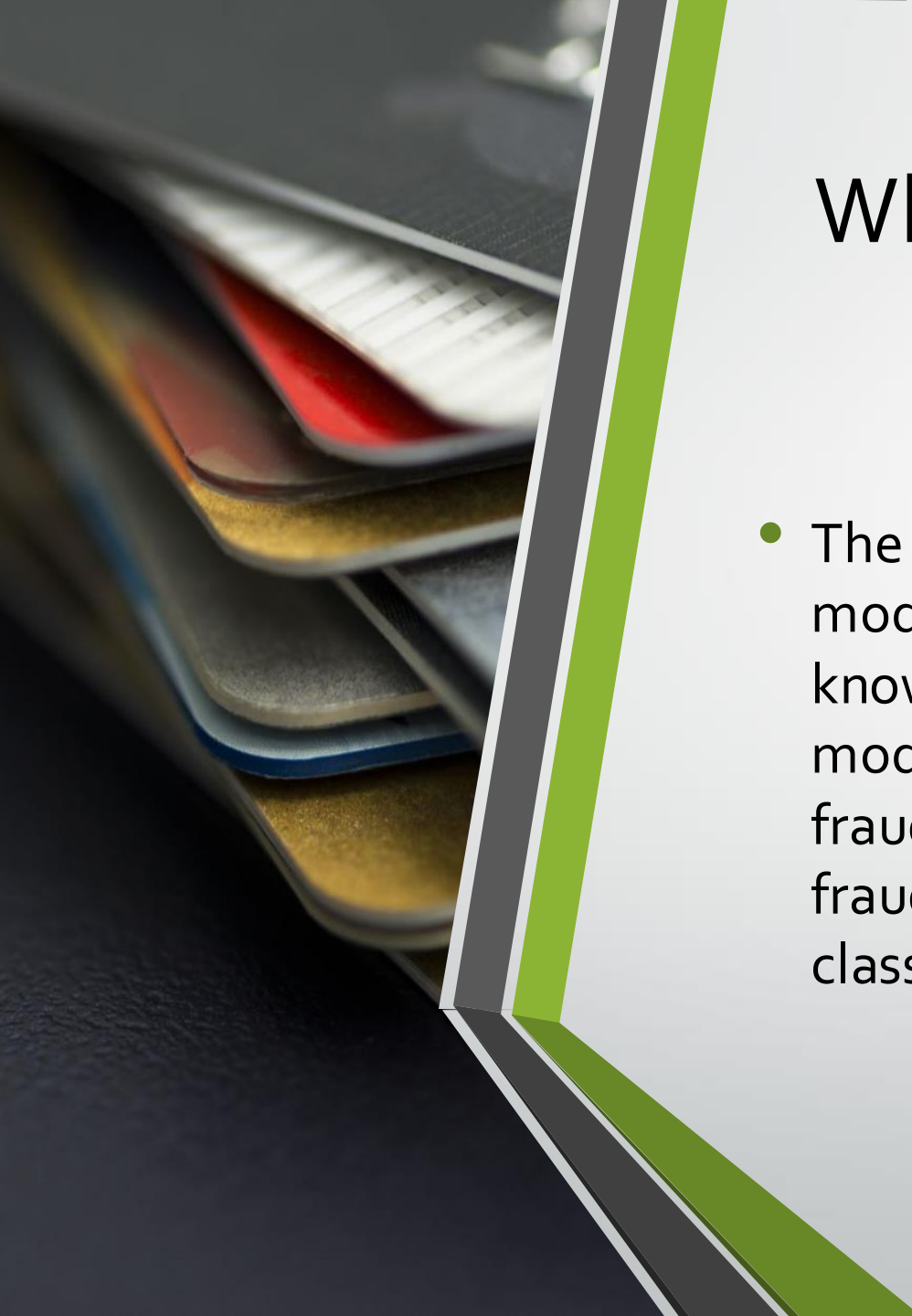
Credit Card Fraud Detection





What is credit card fraud?

- Credit card fraud is when someone uses your credit card or credit account to make purchase you didn't authorize. This activity can happen in different way: If you have lose your credit card or have it stolen, it can be used to make purchase or other transaction, either in person or online



What credit card fraud detection does?

- The credit card fraud detection problem include modeling past credit card transactions with the knowledge of ones that turned out to be fraud. This models then used to identify whether new transaction is fraudulent or not. Our aim here to detect maximum fraudulent transactions while minimizing incorrect fraud classifications

Reading the dataset

```
# Reading the dataset  
df = pd.read_csv('creditcard.csv')  
df.head()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698	0.363787	0.090794	-0.551600
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102	-0.255425	-0.166974	1.612720
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676	-1.514654	0.207643	0.624500
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436	-1.387024	-0.054952	-0.226480
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533	0.817739	0.753074	-0.822840

- Shape = (284807, 31)

Distribution of Class

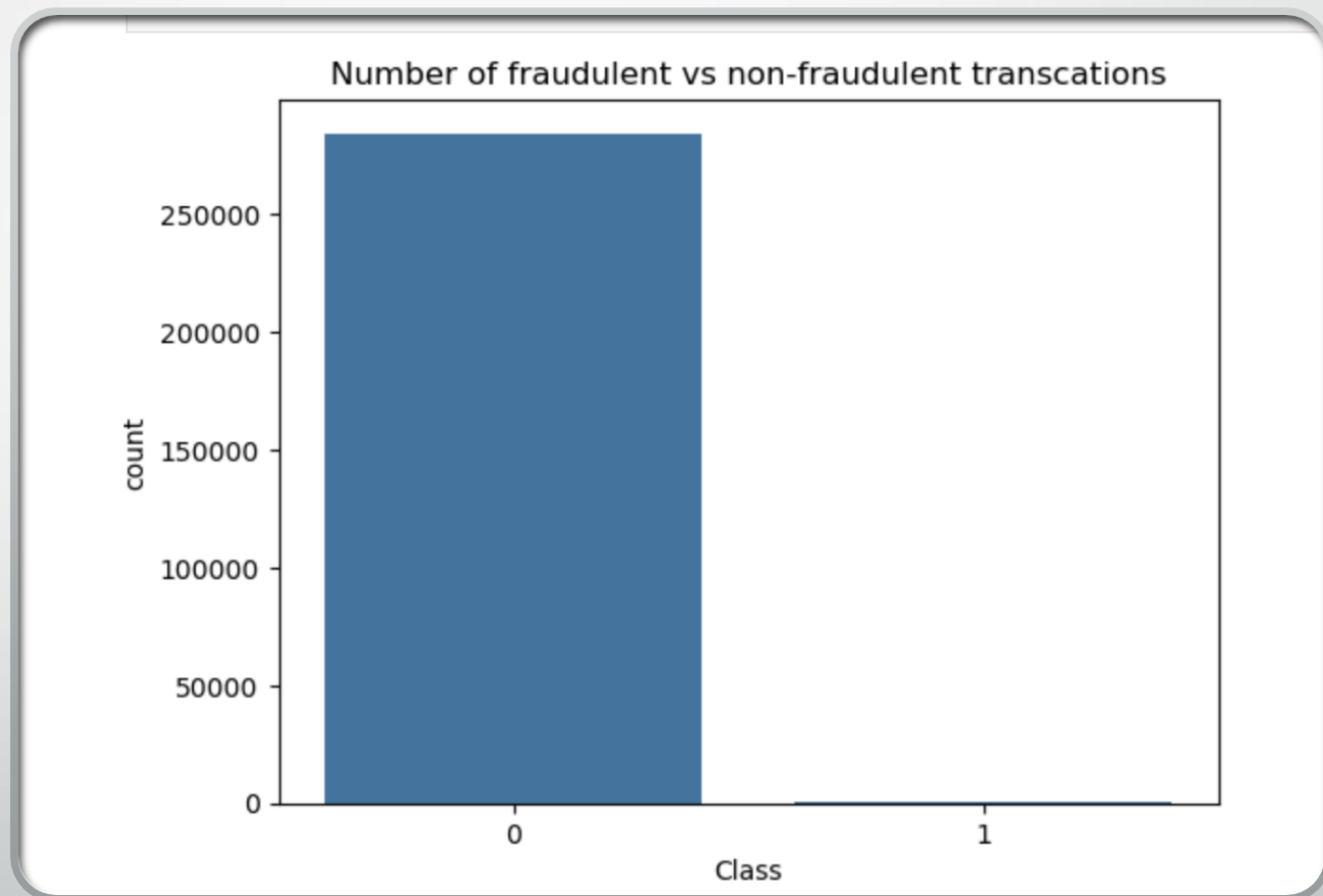
Class

0 – 284315

1 – 492

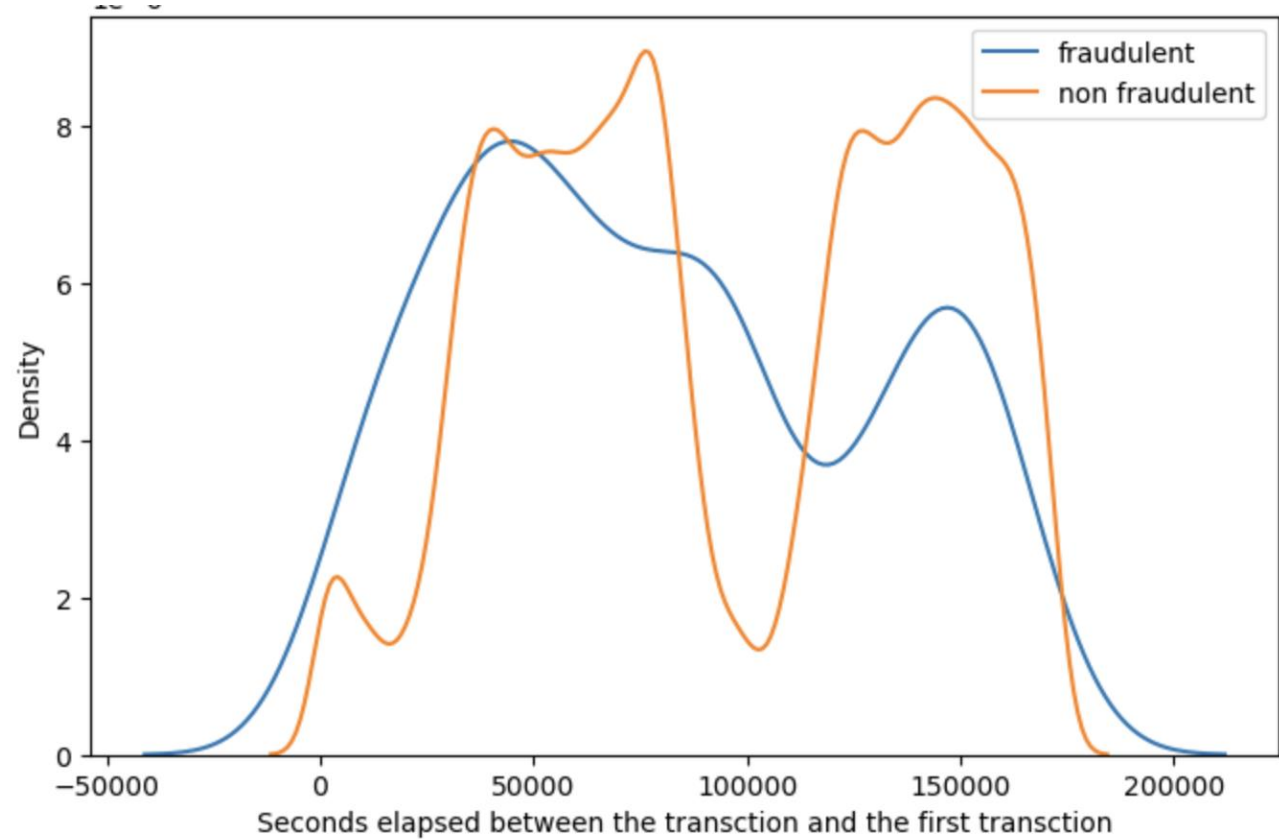
Normal share – 99.83%

Fraud share – 0.17%



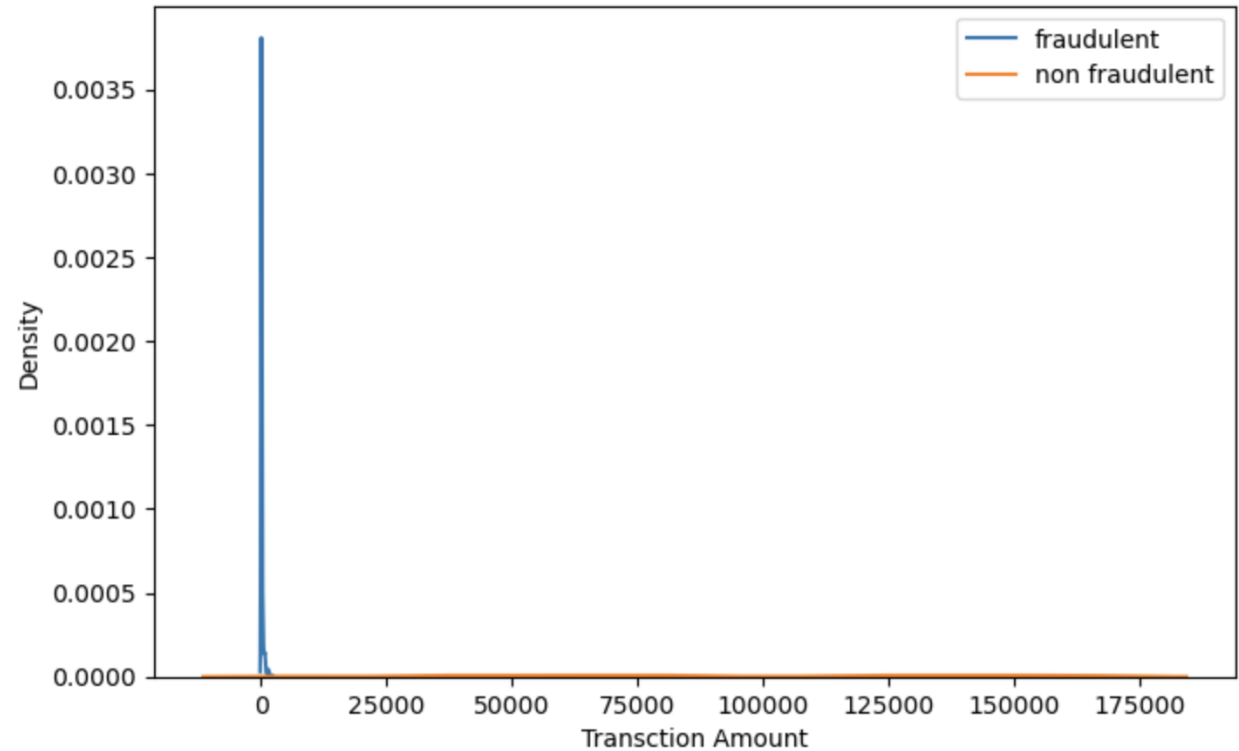
Distribution of classes with Time

- We do not see any specific pattern for the fraudulent and non-fraudulent transactions with respect to Time. Hence, we can drop the Time column.



Distribution of classes with Amount

- We can see that the fraudulent transactions are mostly dense in the lower range of amount, whereas the non-fraudulent transactions are spread throughout low to high range of amount.



Handling the Imbalance data

As we see that the data is heavily imbalanced, We will try several approaches for handling data imbalance:

- Undersampling :- Here for balancing the class distribution, the non-fraudulent transactions count will be reduced to 396 (similar count of fraudulent transactions)
- Oversampling :- Here we will make the same count of non-fraudulent transactions as fraudulent transactions.
- SMOTE :- Synthetic minority oversampling technique. It is another oversampling technique, which uses nearest neighbor algorithm to create synthetic data.
- Adasyn:- This is similar to SMOTE with minor changes that the new synthetic data is generated on the region of low density of imbalanced data points.

Model building on balanced data with Undersampling

Logistic regression :

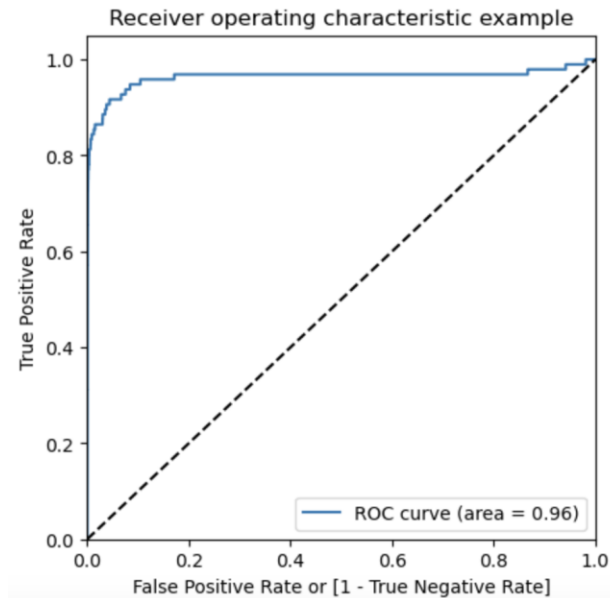
Train set

- Accuracy = 0.96
- Sensitivity = 0.92
- Specificity = 0.99
- ROC = 0.99

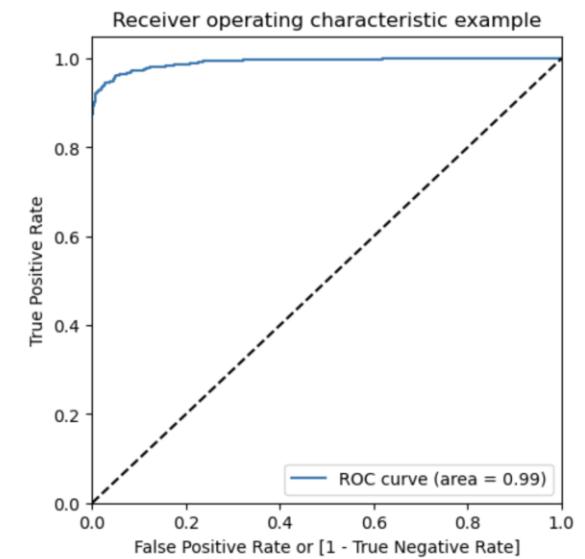
Test set

- Accuracy = 0.97
- Sensitivity = 0.86
- Specificity = 0.97
- ROC = 0.96

```
# Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



```
# Plot the ROC curve  
draw_roc(y_train_rus, y_train_pred_proba)
```



Model building on balanced data with Undersampling

XGBoost:

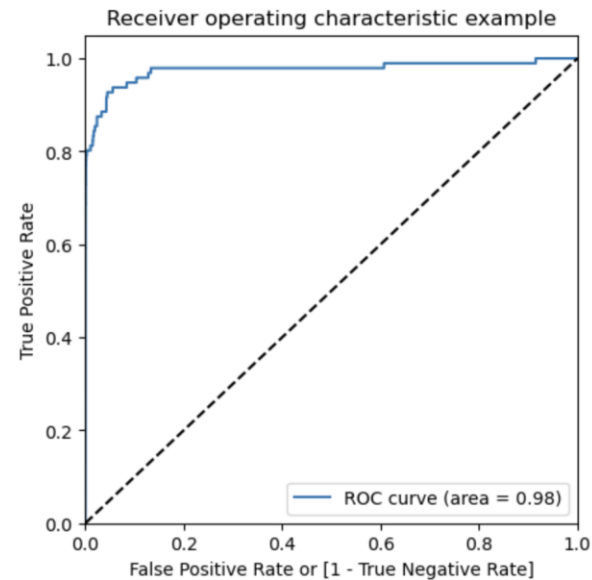
Train set

- Accuracy = 1.0
- Sensitivity = 1.0
- Specificity = 1.0
- ROC-AUC = 1.0

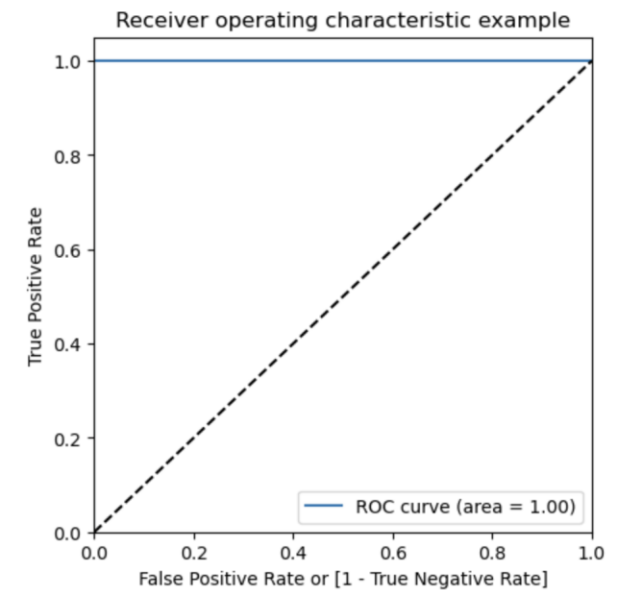
Test set

- Accuracy = 0.95
- Sensitivity = 0.93
- Specificity = 0.95
- ROC-AUC = 0.98

```
# Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



```
# Plot the ROC curve  
draw_roc(y_train_rus, y_train_pred_proba)
```



Model building on balanced data with Oversampling

Logistic regression:

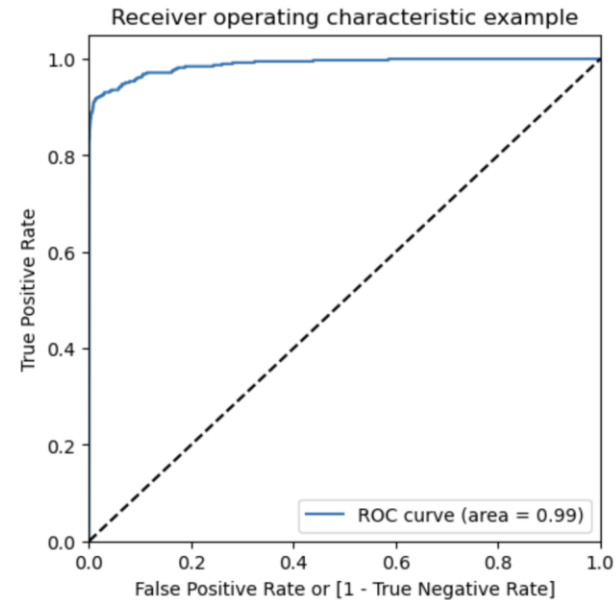
Train set

- Accuracy = 0.95
- Sensitivity = 0.92
- Specificity = 0.98
- ROC = 0.99

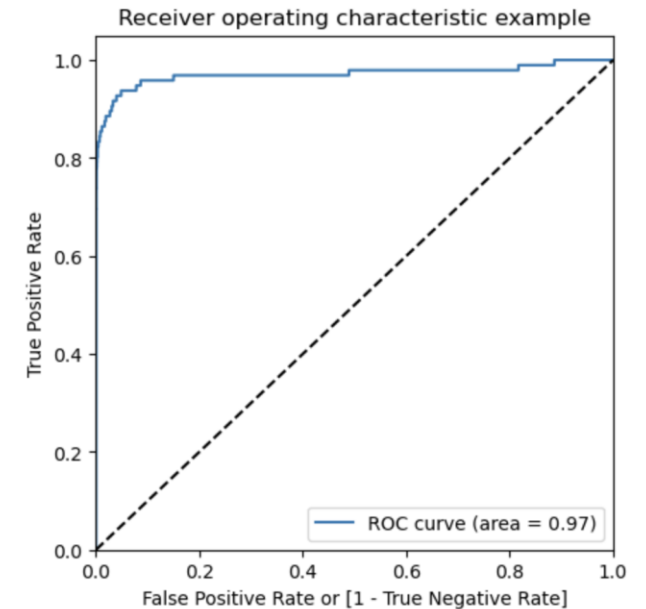
Test set

- Accuracy = 0.98
- Sensitivity = 0.89
- Specificity = 0.98
- ROC = 0.97

```
# Plot the ROC curve  
draw_roc(y_train_ros, y_train_pred_proba)
```



```
# Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



Model building on balanced data with Oversampling

XGboost:

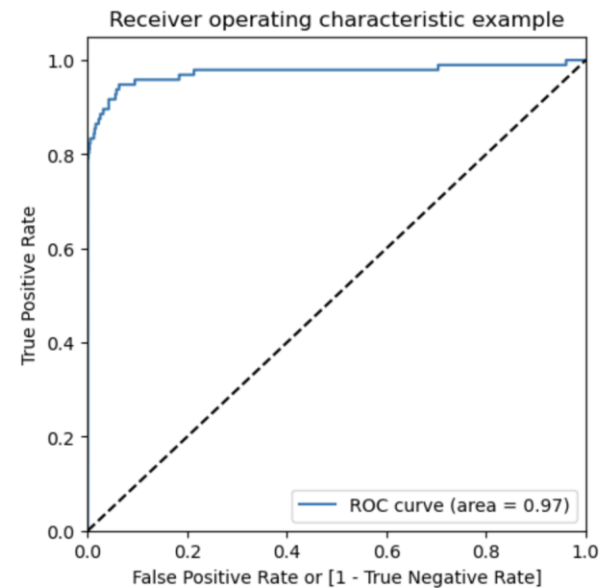
Train set

- Accuracy = 1.0
- Sensitivity = 1.0
- Specificity = 1.0
- ROC-AUC = 1.0

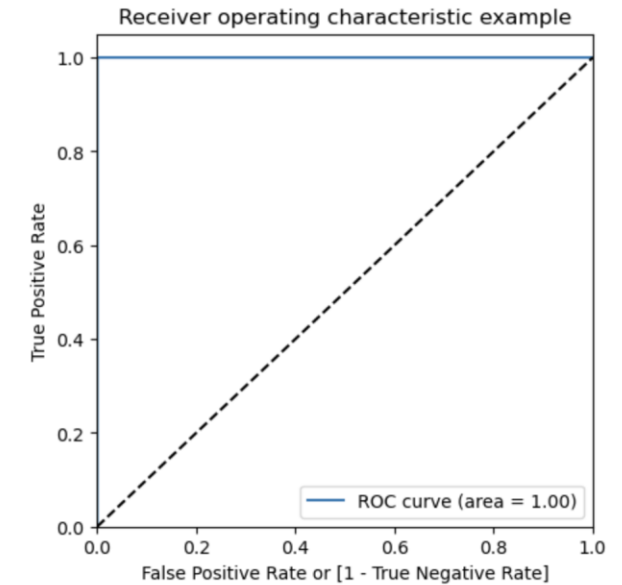
Test set

- Accuracy = 0.99
- Sensitivity = 0.78
- Specificity = 0.99
- ROC-AUC = 0.97

```
# Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



```
62... # Plot the ROC curve  
draw_roc(y_train_ros, y_train_pred_proba)
```



Model building on balanced data with SMOTE

Logistic regression

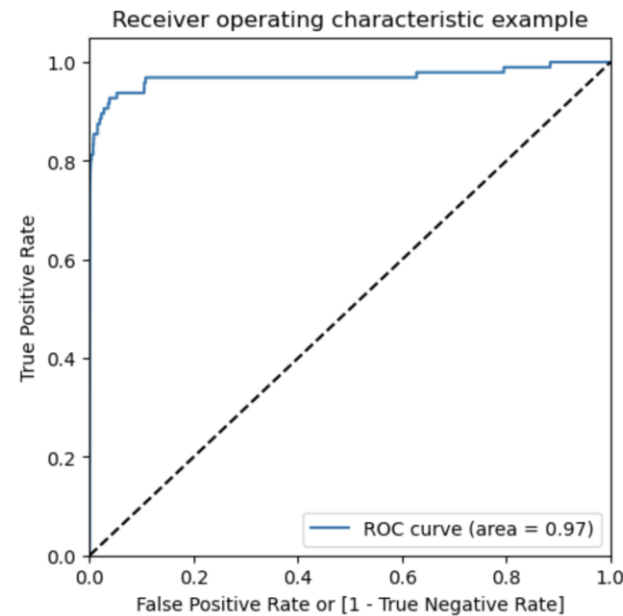
Train set

- Accuracy = 0.95
- Sensitivity = 0.92
- Specificity = 0.98
- ROC = 0.99

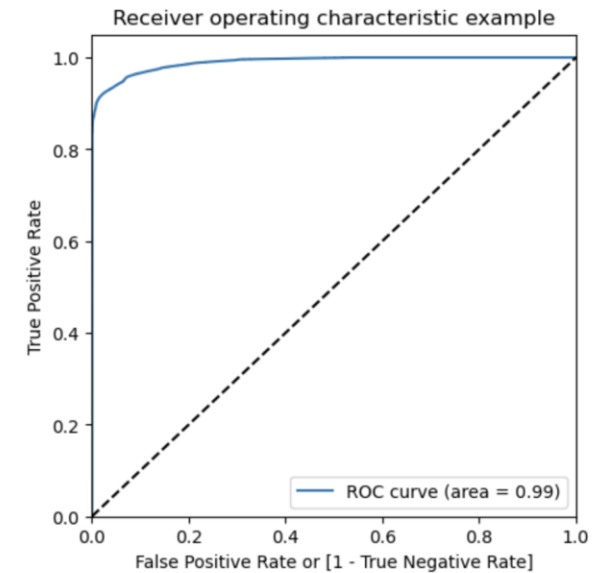
Test set

- Accuracy = 0.97
- Sensitivity = 0.90
- Specificity = 0.97
- ROC = 0.97

```
... # Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



```
# Plot the ROC curve  
draw_roc(y_train_smote, y_train_pred_proba_log_bal_smote)
```



Model building on balanced data with SMOTE

XGboost:

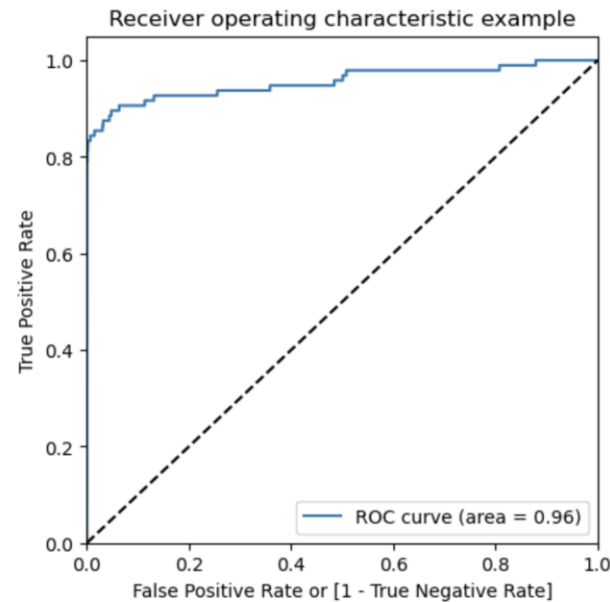
Train set

- Accuracy = 0.99
- Sensitivity = 1.0
- Specificity = 0.99
- ROC-AUC = 1.0

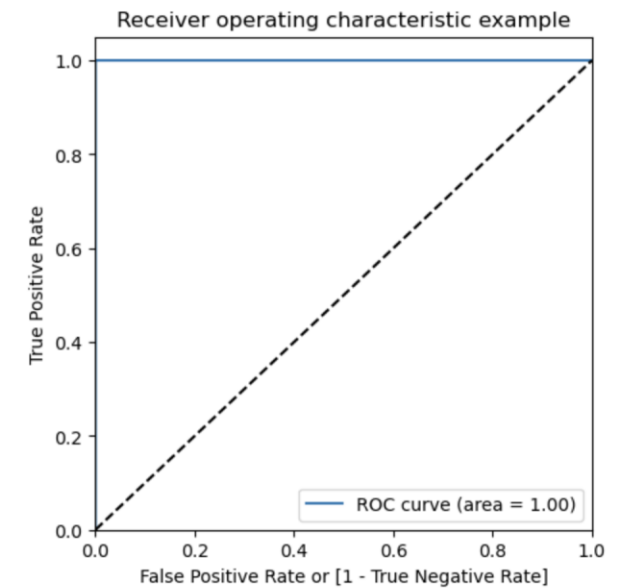
Test set

- Accuracy = 0.99
- Sensitivity = 0.79
- Specificity = 0.99
- ROC-AUC = 0.96

```
1... # Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



```
6... # Plot the ROC curve  
draw_roc(y_train_smote, y_train_pred_proba)
```



Model building on balanced data with AdaSyn

Logistic regression:

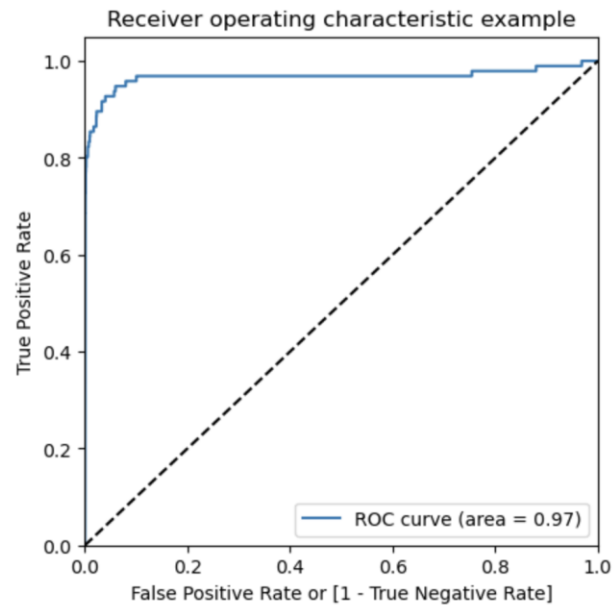
Train set

- Accuracy = 0.89
- Sensitivity = 0.86
- Specificity = 0.91
- ROC = 0.96

Test set

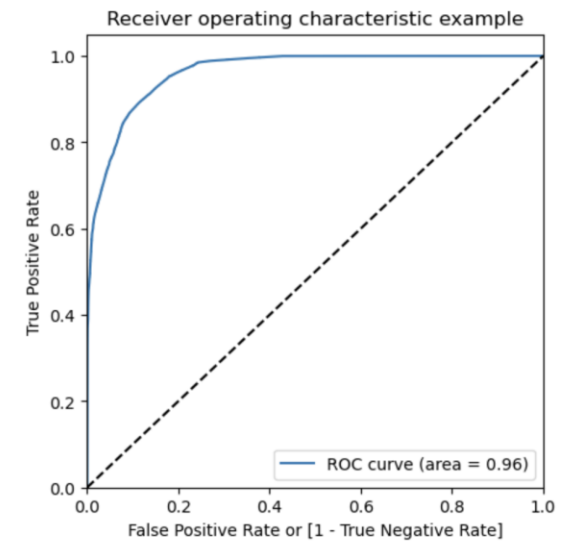
- Accuracy = 0.91
- Sensitivity = 0.96
- Specificity = 0.91
- ROC = 0.97

```
# Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



[270...

```
# Plot the ROC curve  
draw_roc(y_train_adasyn, y_train_pred_proba)
```



Model building on balanced data with AdaSyn

XGboost:

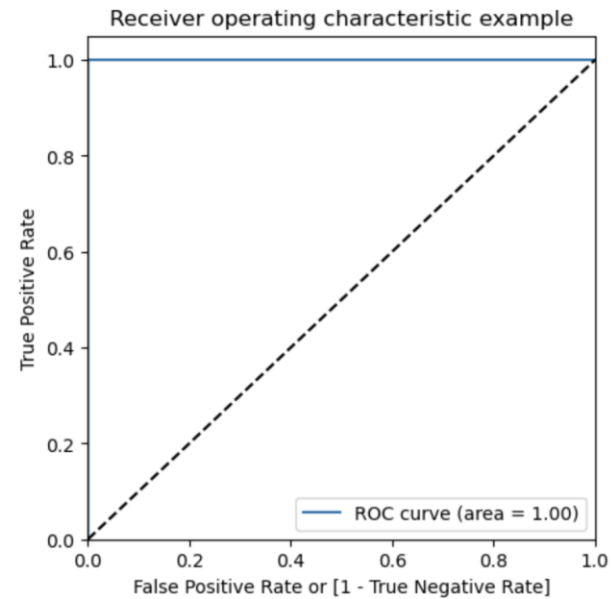
Train set

- Accuracy = 0.99
- Sensitivity = 1.0
- Specificity = 1.0
- ROC-AUC = 1.0

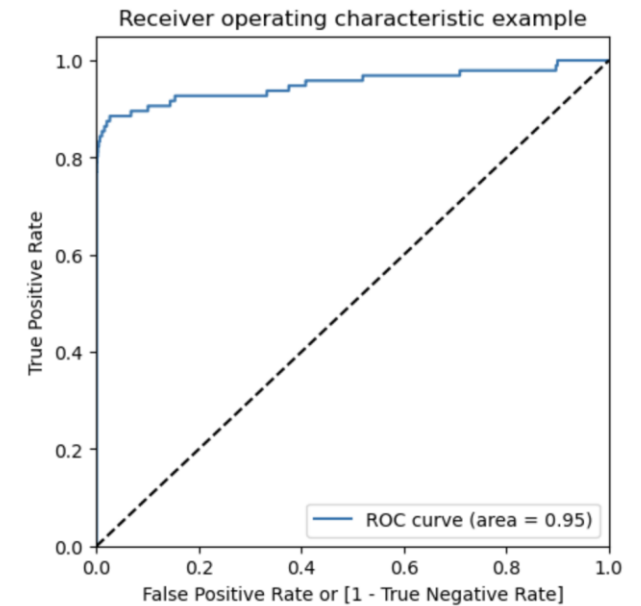
Test set

- Accuracy = 0.99
- Sensitivity = 0.77
- Specificity = 0.99
- ROC-AUC = 0.95

```
.... # Plot the ROC curve  
draw_roc(y_train_adasyn, y_train_pred_proba)
```



```
. # Plot the ROC curve  
draw_roc(y_test, y_test_pred_proba)
```



Choosing best model on the balanced data

- Here we balanced the data with various approach such as Undersampling, Oversampling, SMOTE and Adasyn. With every data balancing technique we built several models such as Logistic, XGBoost, Decision Tree, and Random Forest.
- We can see that almost all the models performed more or less good. But we should be interested in the best model.
- Though the Undersampling technique models performed well, we should keep mind that by doing the undersampling some information were lost. Hence, it is better not to consider the undersampling models.
- Whereas the SMOTE and Adasyn models performed well. Among those models the simplest model Logistic regression has ROC score 0.99 in the train set and 0.97 on the test set. We can consider the Logistic model as the best model to choose because of the easy interpretation of the models and also the resource requirements to build the model is lesser than the other heavy models such as Random forest or XGBoost.
- Hence, we can conclude that the Logistic regression model with SMOTE is the best model for its simplicity and less resource requirement.

Print the FPR,TPR & select the best threshold from the roc curve for the best model

We can see that the threshold is 0.53, for which the TPR is the highest and FPR is the lowest and we got the best ROC score.

Print the FPR,TPR & select the best threshold from the roc curve for the best model

```
1... print('Train auc =', metrics.roc_auc_score(y_train_smote, y_train_pred_proba_log_bal_smote))  
    fpr, tpr, thresholds = metrics.roc_curve(y_train_smote, y_train_pred_proba_log_bal_smote)  
    threshold = thresholds[np.argmax(tpr-fpr)]  
    print("Threshold=",threshold)
```

Train auc = 0.9897681302852576

Threshold= 0.5322737615586992

Model deployment plan

Model deployment plan

Deployment Overview:

- Once the model is trained, it can be deployed as:
 - A REST API using Flask or FastAPI
 - Cloud Deployment: Deploy models on cloud platforms like AWS, Azure, or GCP to scale up as needed.
 - Saved and loaded via pickle or joblib
 - Monitoring: Continuously monitor the model's performance in production to ensure it remains effective over time.

```
3... import joblib  
    joblib.dump(rf, 'model.pkl')
```

```
3... ['model.pkl']
```



“Thank You”

