Credit card fraud detection

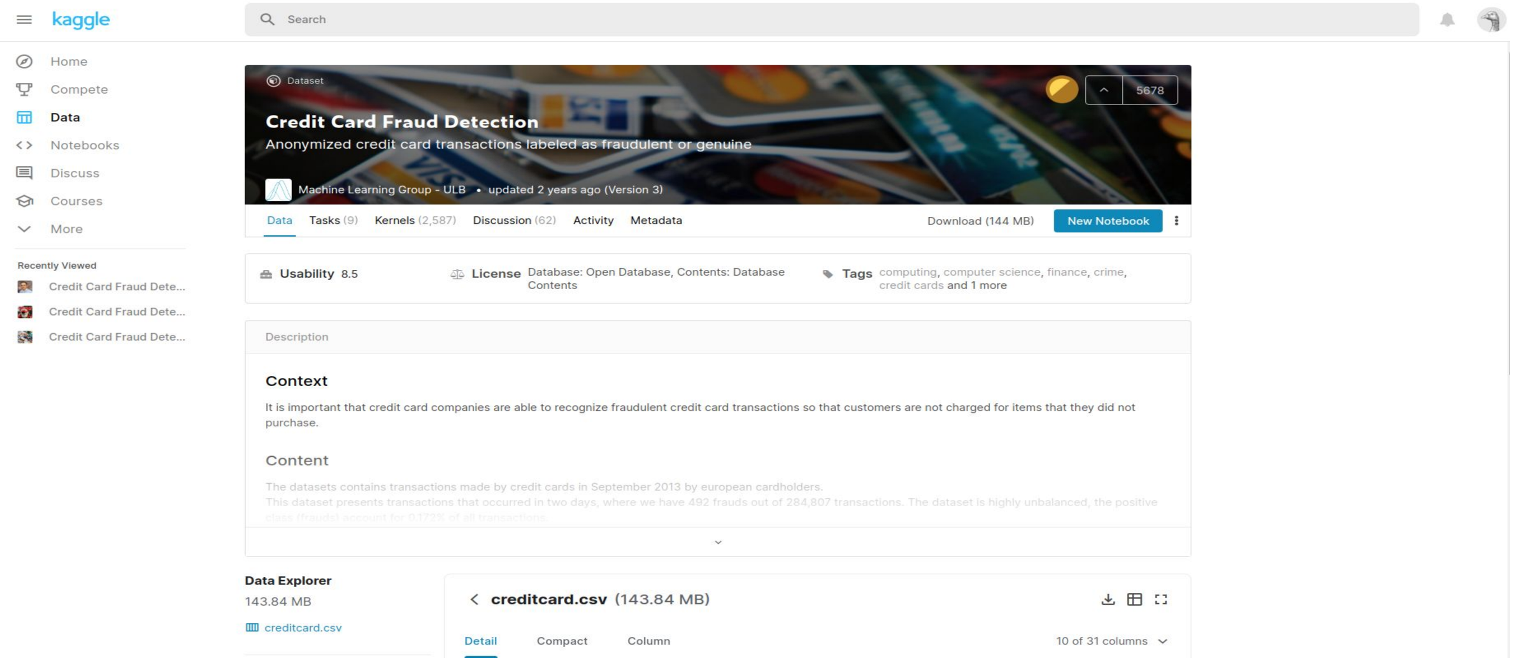
Sajjad Haider, Hein Lu and
Atul Kumar Yadav
Kiel, 29.06.2020

Machine Learning and Deep Learning project
opencampus.sh course 2020

Goal of the project

- Detect the fraud in credit card transaction records
- Develop ML models to identify the frauds
- Identify the best ML model for the task
- Improve the model

Dataset : description



Dataset : description

```
[ ] # Explore the features available in our dataframe
    print(df.info())
    print()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
```

Dataset : description

```
[ ] # Explore the features available in our dataframe
    print(df.info())
    print()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
```

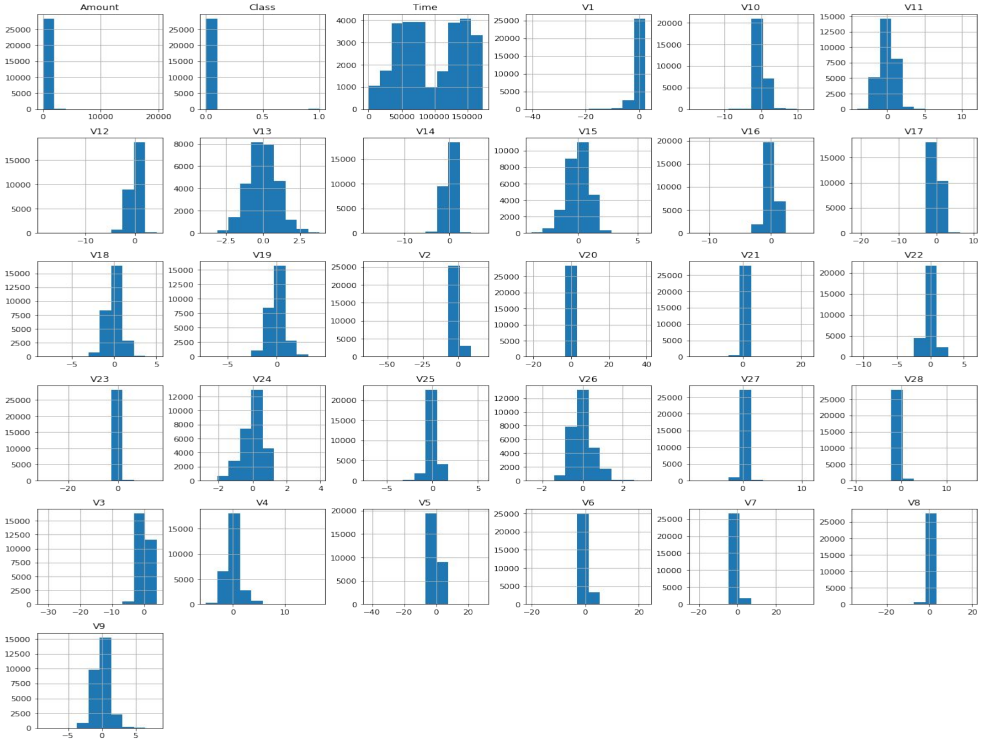
The datasets contains:

- Transactions made by credit cards in September 2013 by european cardholders.
- Transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions.

Dataset : description

```
[ ] # Explore the features available in our dataframe
    print(df.info())
    print()
```

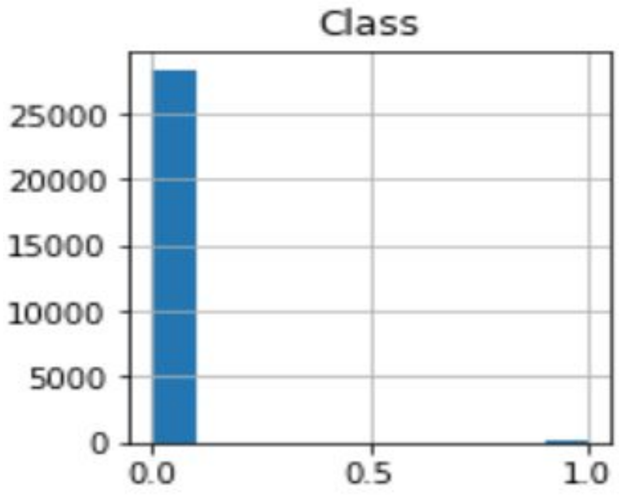
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   Time     284807 non-null  float64
 1   V1       284807 non-null  float64
 2   V2       284807 non-null  float64
 3   V3       284807 non-null  float64
 4   V4       284807 non-null  float64
 5   V5       284807 non-null  float64
 6   V6       284807 non-null  float64
 7   V7       284807 non-null  float64
 8   V8       284807 non-null  float64
 9   V9       284807 non-null  float64
 10  V10      284807 non-null  float64
 11  V11      284807 non-null  float64
 12  V12      284807 non-null  float64
 13  V13      284807 non-null  float64
 14  V14      284807 non-null  float64
 15  V15      284807 non-null  float64
 16  V16      284807 non-null  float64
 17  V17      284807 non-null  float64
 18  V18      284807 non-null  float64
 19  V19      284807 non-null  float64
 20  V20      284807 non-null  float64
 21  V21      284807 non-null  float64
 22  V22      284807 non-null  float64
 23  V23      284807 non-null  float64
 24  V24      284807 non-null  float64
 25  V25      284807 non-null  float64
 26  V26      284807 non-null  float64
 27  V27      284807 non-null  float64
 28  V28      284807 non-null  float64
 29  Amount   284807 non-null  float64
 30  Class    284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
```



Dataset : unbalanced data challenges

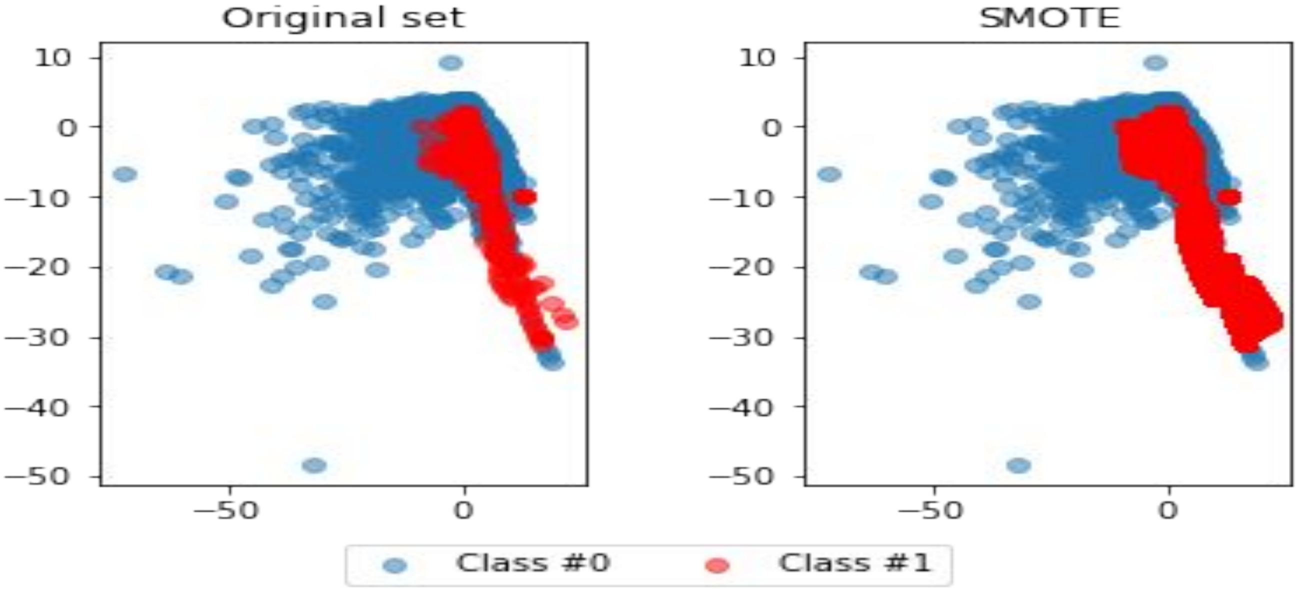
```
[ ] # Explore the features available in our dataframe
    print(df.info())
    print()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column   Non-Null Count   Dtype
---  ------   --------------   -----
 0   Time     284807 non-null  float64
 1   V1       284807 non-null  float64
 2   V2       284807 non-null  float64
 3   V3       284807 non-null  float64
 4   V4       284807 non-null  float64
 5   V5       284807 non-null  float64
 6   V6       284807 non-null  float64
 7   V7       284807 non-null  float64
 8   V8       284807 non-null  float64
 9   V9       284807 non-null  float64
 10  V10      284807 non-null  float64
 11  V11      284807 non-null  float64
 12  V12      284807 non-null  float64
 13  V13      284807 non-null  float64
 14  V14      284807 non-null  float64
 15  V15      284807 non-null  float64
 16  V16      284807 non-null  float64
 17  V17      284807 non-null  float64
 18  V18      284807 non-null  float64
 19  V19      284807 non-null  float64
 20  V20      284807 non-null  float64
 21  V21      284807 non-null  float64
 22  V22      284807 non-null  float64
 23  V23      284807 non-null  float64
 24  V24      284807 non-null  float64
 25  V25      284807 non-null  float64
 26  V26      284807 non-null  float64
 27  V27      284807 non-null  float64
 28  V28      284807 non-null  float64
 29  Amount   284807 non-null  float64
 30  Class    284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
None
```



The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.

Dataset : unbalanced data solution



Re-balancing our data using the Synthetic Minority Over-sampling Technique (SMOTE) and comparing to original data

Terminology

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

$$recall = \frac{true\ positives}{true\ positives\ + false\ negatives}$$

$$precision = \frac{true\ positives}{true\ positives + false\ positives}$$

True Positives (TP): These are cases in which we predicted yes (they have the fraud), and they do have the fraud.

True Negatives (TN): We predicted no, and they don't have the fraud.

False Positives (FP): We predicted yes, but they don't actually have the fraud. (Also known as a "Type I error.")

False Negatives (FN): We predicted no, but they actually do have the fraud. (Also known as a "Type II error.")

Fraud detection: traditional way

```
[ ] # Implement a rule for stating which cases are flagged as fraud
    df['flag_as_fraud'] = np.where(np.logical_and(df['V1'] < -3, df['V3'] < -5), 1, 0)

    # Create a crosstab of flagged fraud cases versus the actual fraud cases
    pd.crosstab(df.Class, df.flag_as_fraud, rownames=['Actual Fraud'], colnames=['Flagged Fraud'])
```

Flagged Fraud	0	1
Actual Fraud		
0	283089	1226
1	322	170

Not bad, with this rule, we detect 170 (TP) out of 492 fraud cases, but can't detect the other 322 (TN), and get 1226 FP and 283089 FN.

Fraud detection: Logistic regression

```
Classification report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85296
         1.0       0.89      0.63      0.74       147

    accuracy                           1.00     85443
   macro avg       0.95      0.81      0.87     85443
weighted avg       1.00      1.00      1.00     85443

Confusion matrix:
 [[85285    11]
 [   55    92]]
```

- We are getting much less false positives, so that's an improvement. Also, we're catching a higher percentage of fraud cases, so that is also better than before.
- we are using only our test data to calculate the model results.
- We're comparing the crosstab on the full dataset from the last exercise, with a confusion matrix of only 30% of the total dataset, so that's where that difference comes from.

*combined with SMOTE and using pipeline

Fraud detection: Logistic regression*

```
Classifcation report:
              precision    recall  f1-score   support

         0.0       1.00      0.98      0.99     85296
         1.0       0.07      0.88      0.13       147

    accuracy                           0.98     85443
   macro avg       0.53      0.93      0.56     85443
weighted avg       1.00      0.98      0.99     85443

Confusion matrix:
 [[83522  1774]
 [   18   129]]
```

- As we can see, the SMOTE slightly improves our results. We now manage to find all cases of fraud, but we have a slightly higher number of false positives.
- Remember, not in all cases does resampling necessarily lead to better results. When the fraud cases are very spread and scattered over the data, using SMOTE can introduce a bit of bias.
- Nearest neighbors aren't necessarily also fraud cases, so the synthetic samples might 'confuse' the model slightly.

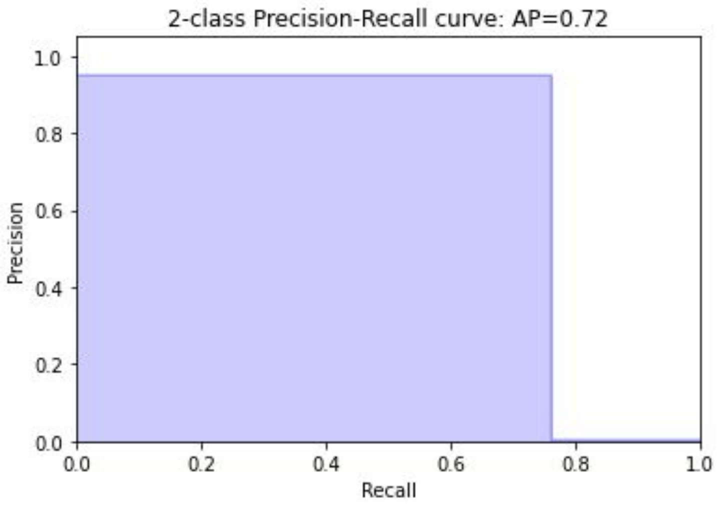
Fraud detection: using Random Forest Classifier

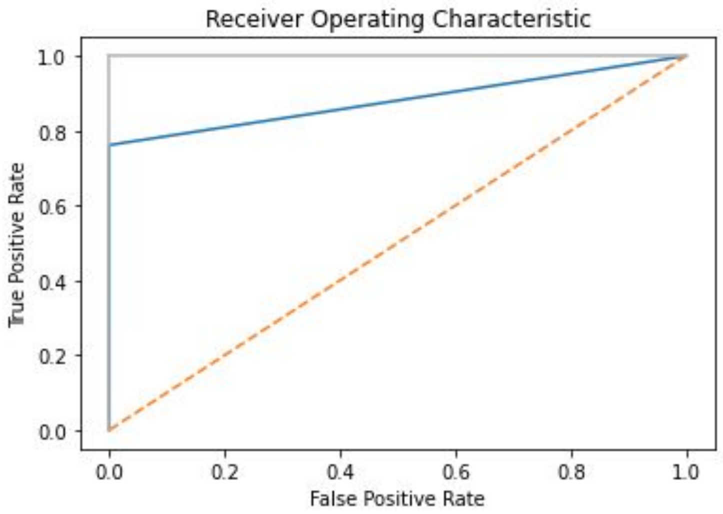
```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85296
         1.0       0.95      0.76      0.85       147

    accuracy                           1.00     85443
   macro avg       0.97      0.88      0.92     85443
weighted avg       1.00      1.00      1.00     85443

[[85290     6]
 [   35   112]]
AUC ROC score:  0.9338071375614587
```

- Given the highly imbalanced data that we're working with, we have now obtained better performance
- The model predicts 118 cases of fraud, out of which 112 are actual fraud. You have only 6 false positives.
- This is really good, and as a result you have a very high precision score.
- You do however, don't catch 35 cases of actual fraud (false negative).

- AUC ROC : Area under the ROC (Receiver Operating Characteristic) curve is a performance measurement for classification problem at various thresholds settings.
- ROC is a probability curve and AUC represents degree or measure of separability. It tells how much model is capable of distinguishing between classes.
- **Higher** the AUC, **better** the model is at predicting 0s as 0s and 1s as 1s. By analogy, Higher the AUC, better the model is at distinguishing between transaction with fraud and no fraud.

Fraud detection:using Random Forest Classifier





The ROC curve plots the true positives vs. false positives , for a classifier, as its discrimination threshold is varied. Since, a random method describes a horizontal curve through the unit interval, it has an AUC of 0.5. Minimally, classifiers should perform better than this, and the extent to which they score higher than one another (meaning the area under the ROC curve is larger), they have better expected performance.

Fraud detection: Adjusting Random Forest Classifier (RFC_ad1)

```python
# Define the model with balanced subsample
model = RandomForestClassifier(class_weight='balanced_subsample', n_estimators=100, random_state=5)

# Fit your training model to your training set
model.fit(X_train, y_train)

# Obtain the predicted values and probabilities from the model
predicted = model.predict(X_test)
probs = model.predict_proba(X_test)

# Print the roc_auc_score, the classification report and confusion matrix
print(roc_auc_score(y_test, probs[:,1]))
print(classification_report(y_test, predicted))
print(confusion_matrix(y_test, predicted))
```

```
0.9373615465694811
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85296
         1.0       0.97      0.75      0.85       147

    accuracy                           1.00     85443
   macro avg       0.99      0.87      0.92     85443
weighted avg       1.00      1.00      1.00     85443

[[85293     3]
 [   37   110]]
```

```python
# Define the model as the random forest
model = RandomForestClassifier(random_state=5)
```

- We can see that the model results don't improve drastically.
- If we mostly care about catching fraud, and not so much about the false positives, this does actually not improve our model at all.

Fraud detection: Adjusting Random Forest Classifier (RFC_ad2)

```
# Change the model options
model = RandomForestClassifier(bootstrap=True,
                               class_weight={0:1, 1:12}, # 0: non-fraud , 1:fraud
                               criterion='entropy',

                               # Change depth of model
                               max_depth=10,

                               # Change the number of samples in leaf nodes
                               min_samples_leaf=10,

                               # Change the number of trees to use
                               n_estimators=20,

                               n_jobs=-1, random_state=5)

# Run the function get_model_results
get_model_results(X_train, y_train, X_test, y_test, model)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85296
         1.0       0.86      0.80      0.83       147

    accuracy                           1.00     85443
   macro avg       0.93      0.90      0.91     85443
weighted avg       1.00      1.00      1.00     85443

[[85277    19]
 [   30   117]]
```

- Defining more options in the model improved the prediction.
- The number of false negatives has been reduced (more fraud cases are identified) while keeping the number of false positives low.

Fraud detection : using GridSearchCV

```python
# Input the optimal parameters in the model
model = RandomForestClassifier(class_weight={0:1,1:12},
                               criterion='gini',
                               n_estimators=30,
                               max_features='log2',
                               min_samples_leaf=10,
                               max_depth=8,
                               n_jobs=-1, random_state=5)

# Get results from your model
get_model_results(X_train, y_train, X_test, y_test, model)
```

```
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00     85296
         1.0       0.83      0.80      0.82       147

    accuracy                           1.00     85443
   macro avg       0.92      0.90      0.91     85443
weighted avg       1.00      1.00      1.00     85443

[[85272    24]
 [   29   118]]
```

- Model has been improved even further.
- The number of false positives has now been slightly reduced even further, which means we are catching more cases of fraud.
- However, we see that the number of false negatives is still the same, which is due to Precision-Recall trade-off.
- To determine which model is best, we need to take into account how bad it is not to catch fraudsters, versus how many false positives the fraud analytics team can deal with.

Fraud detection：using Anomaly Detection technique

Local Outlier Factor (LOF)

- The anomaly score of each sample is called Local Outlier Factor.
- It measures the local deviation of density of a given sample with respect to its neighbors. It is local in that the anomaly score depends on how isolated the object is with respect to the surrounding neighborhood.

Isolation Forest Algorithm

- The Isolation Forest 'isolates' observations by randomly selecting a feature and then randomly selecting a split value between the maximum and minimum values of the selected feature.
- Since recursive partitioning can be represented by a tree structure, the number of splittings required to isolate a sample is equivalent to the path length from the root node to the terminating node.
- This path length, averaged over a forest of such random trees, is a measure of normality and our decision function.
- Random partitioning produces noticeably shorter paths for anomalies. Hence, when a forest of random trees collectively produce shorter path lengths for particular samples, they are highly likely to be anomalies.

Fraud detection: using Anomaly Detection technique

```
Isolation Forest: 71
0.99750711000316
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.28      0.29      0.28        49

    accuracy                           1.00     28481
   macro avg       0.64      0.64      0.64     28481
weighted avg       1.00      1.00      1.00     28481

Local Outlier Factor: 97
0.9965942207085425
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     28432
           1       0.02      0.02      0.02        49

    accuracy                           1.00     28481
   macro avg       0.51      0.51      0.51     28481
weighted avg       1.00      1.00      1.00     28481
```

- Isolation Forest detected 71 errors versus Local Outlier Factor detecting 97 errors
- Isolation Forest has a 99.75% more accurate than LOF of 99.65%
- When comparing error precision & recall for 2 models , the Isolation Forest performed much better than the LOF as we can see that the detection of fraud cases is around 29 % versus LOF detection rate of just 2 %
- So overall Isolation Forest Method performed much better in determining the fraud cases which is around 30%.
- We can also improve on this accuracy by increasing the sample size or use deep learning algorithms however at the cost of computational expense.We can also use complex anomaly detection models to get better accuracy in determining more fraudulent cases

Fraud detection: using TensorFlow

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 256)               7936
_________________________________________________________________
dense_1 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 256)               65792
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 257
=================================================================
Total params: 139,777
Trainable params: 139,777
Non-trainable params: 0
```

At the end of training, out of 56255 validation transactions, we are:

- Correctly identifying 66 of them as fraudulent
- Missing 09 fraudulent transactions
- At the cost of incorrectly flagging 631 legitimate transactions

```
Epoch 50/50
112/112 - 5s - loss: 4.8024e-07 - fn: 3.0000 - fp: 3581.0000 - tn: 223848.0000 - tp: 414.0000 - precision: 0.1036 - recall: 0.9928
        val_loss: 0.1029 - val_fn: 9.0000 - val_fp: 631.0000 - val_tn: 56255.0000 - val_tp: 66.0000 - val_precision: 0.0947 - val_recall: 0.8800
```

Model Comparison

Model	True Positive	Precision	Recall
Traditional crosstab	170		
Logistic regression	92	0.89	0.63
Logistic regression with SMOTe	129	0.07	0.88
Random Forest Classifier (RFC)	**112**	**0.95**	**0.76**
RFC adjustment 1	**110**	**0.97**	**0.75**
RFC adjustment 2	117	0.86	0.80
GridSearchCV	118	0.83	0.80
Anomaly detection with Isolation Forest	71	0.28	0.59
Anomaly detection with Local Outlier Factor	97	0.02	0.02

Thank you for your attention!!!