# Credit Card Fraud Detection using XGBoost

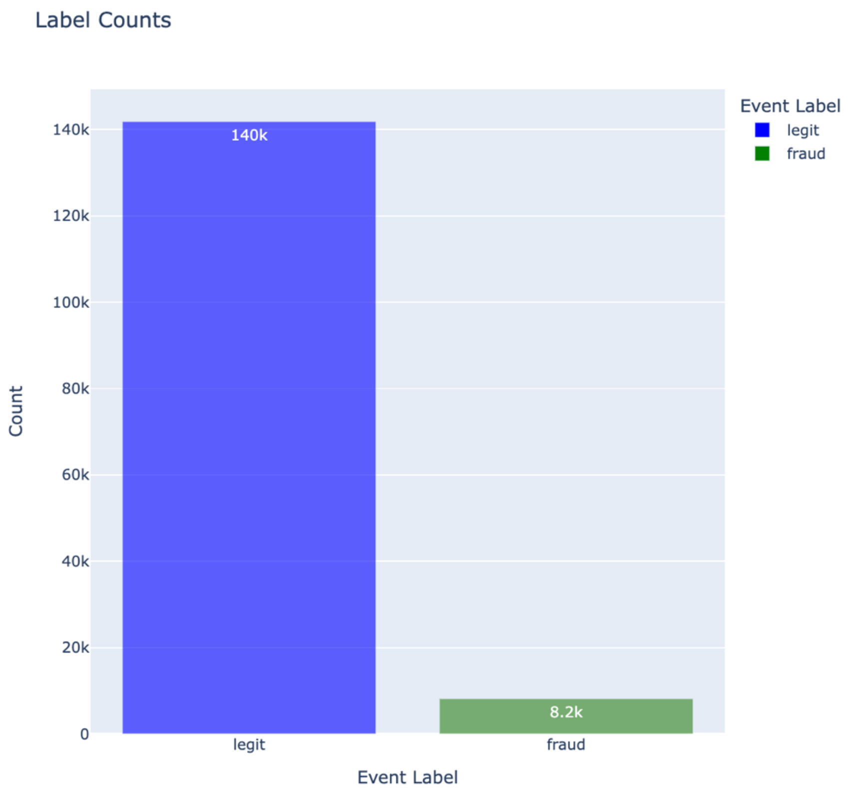
**Dataset:** The dataset has 150000 rows and 25 columns. Data types and column names are below.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 26 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   account_age_days      149892 non-null  float64
 1   transaction_amt       149870 non-null  float64
 2   transaction_adj_amt   149886 non-null  float64
 3   historic_velocity     149885 non-null  float64
 4   ip_address            149873 non-null  object 
 5   user_agent            149887 non-null  object 
 6   email_domain          149910 non-null  object 
 7   phone_number          149873 non-null  object 
 8   billing_city          149884 non-null  object 
 9   billing_postal        149876 non-null  float64
 10  billing_state         149887 non-null  object 
 11  card_bin              149872 non-null  float64
 12  currency              149892 non-null  object 
 13  cvv                   149877 non-null  object 
 14  signature_image       149895 non-null  object 
 15  transaction_type      149884 non-null  object 
 16  transaction_env       149877 non-null  object 
 17  EVENT_TIMESTAMP       149888 non-null  object 
 18  applicant_name        149857 non-null  object 
 19  billing_address       149866 non-null  object 
 20  merchant_id           149893 non-null  object 
 21  locale                149866 non-null  object 
 22  tranaction_initiate   149874 non-null  object 
 23  days_since_last_logon 149864 non-null  float64
 24  inital_amount         149872 non-null  float64
 25  EVENT_LABEL           150000 non-null  object 
dtypes: float64(8), object(18)
memory usage: 29.8+ MB
```

The data set has few null values in each of the columns. Here is the count of null values for each column in the dataset.

```
account_age_days         108
transaction_amt          130
transaction_adj_amt      114
historic_velocity        115
ip_address               127
user_agent               113
email_domain              90
phone_number             127
billing_city             116
billing_postal           124
billing_state            113
card_bin                 128
currency                 108
cvv                      123
signature_image          105
transaction_type         116
transaction_env          123
EVENT_TIMESTAMP          112
applicant_name           143
billing_address          134
merchant_id              107
locale                   134
tranaction_initiate      126
days_since_last_logon    136
inital_amount            128
EVENT_LABEL                0
dtype: int64
```

**Target Column** is EVENT_LABEL that has two labels which are legit and fraud. The dataset consists of 140k legit cases and 8.2k fraud cases indicating the dataset is imbalanced.




## Data Processing & Data Selection

**Missing Values:** XGBoost supports missing values by default. In tree algorithms, branch directions for missing values are learned during training. Therefore, no approach has been taken to handle the missing values.

**Dropping Columns:** Following columns has been dropped from the dataset as they are not important for the EVENT_LABEL prediction.

```
#data selection
columns_to_drop = ['ip_address', 'user_agent', 'email_domain',
                   'phone_number', 'billing_postal',
                   'card_bin',
                   'EVENT_TIMESTAMP',
                   'applicant_name', 'billing_address', 'merchant_id',
                   'locale', 'tranaction_initiate',
                   'EVENT_TIME', 'EVENT_DATE']

p_df = p_df.drop(columns=columns_to_drop, axis = 1)
```

**Label Encoding:** As there are few categorical features with highly variable categories. So, label encoding has been performed to transform them to numerical datapoints.

For the EVENT_LABEL, label encoding has done manually, and the labels were considered as 'legit' = 1, 'fraud'= 0.

```
#label encoding
columns_to_label_encode = ['billing_city',
                           'cvv', 'currency',
                           'signature_image', 'transaction_type', 'transaction_env',
                           'user_agent_os']

label_encoder = LabelEncoder()

for column in columns_to_label_encode:
    p_df[column] = label_encoder.fit_transform(p_df[column])


p_df['EVENT_LABEL'] = p_df['EVENT_LABEL'].replace({'legit': 1, 'fraud': 0})
```

**One-Hot-Encoding:** For the billing_state column one-hot-encoding has performed.

```
#one-hot-encoding
billing_state_ = pd.get_dummies(df['billing_state']).astype('int')

p_df = pd.concat([p_df, billing_state_], axis=1)
p_df = p_df.drop('billing_state', axis = 1)
```

**Train Test Split**

```
#train test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
```

**Building XGB classifier with hyperparameter and GridSearch**

```
xgbc = xgb.XGBClassifier(tree_method = 'hist',n_jobs = 6, random_state = 10)

param_grid = {

    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'n_estimators': [50, 100, 200],
    'subsample': [0.8, 0.9, 1.0],
}

cv = RepeatedStratifiedKFold(n_splits=3, n_repeats=3, random_state=1)

grid_search = GridSearchCV(estimator=xgbc, param_grid=param_grid, scoring='average_precision', cv=cv,verbose = 3)
grid_search.fit(X_train, y_train)
```

**Tree_method = 'hist'**: signifies the adoption of a histogram-based approach for constructing decision trees during the boosting process. Instead of individually evaluating data points, this method groups them into bins, significantly reducing computational complexity.

**Learning_rate:** This parameter controls the step size or shrinkage applied to the contribution of each tree during the boosting process. Essentially, it scales the contribution of each tree's predictions to the final prediction. A lower learning rate makes the model more robust by reducing the impact of each individual tree. A higher learning rate accelerates the learning process but increases the risk of overfitting.

**Max_depth:** it sets the maximum depth of each tree in the ensemble, controlling the number of nodes from the root to the farthest leaf. This parameter is crucial for balancing model complexity and overfitting: setting it too high risks overfitting, while setting it too low may lead to underfitting. Tuning max_depth helps find the right balance between capturing intricate patterns in the data and preventing excessive memorization.

**Subsample**: it refers to the fraction of the training dataset that is randomly sampled to train each individual tree. It essentially controls the randomness introduced during the construction of each tree. A value of 1.0 means the entire dataset is used to train each tree, while values less than 1.0 indicate that only a fraction of the dataset is sampled. This parameter helps prevent overfitting by introducing diversity among the trees in an ensemble, making the model more robust and less prone to memorizing the training data.

**Best parameters:**

```
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'learning_rate': 0.2, 'max_depth': 5, 'n_estimators': 200, 'subsample': 1.0}
```

## Model Performance

**Classification Report of each class:** 0 is fraud class and 1is legit class.

Classification Report:

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.92 | 0.73 | 0.82 | 1633 |
| 1 | 0.98 | 1.00 | 0.99 | 28367 |
| accuracy | | | 0.98 | 30000 |
| macro avg | 0.95 | 0.87 | 0.90 | 30000 |
| weighted avg | 0.98 | 0.98 | 0.98 | 30000 |

**Precision, Recall, F1-Score:**

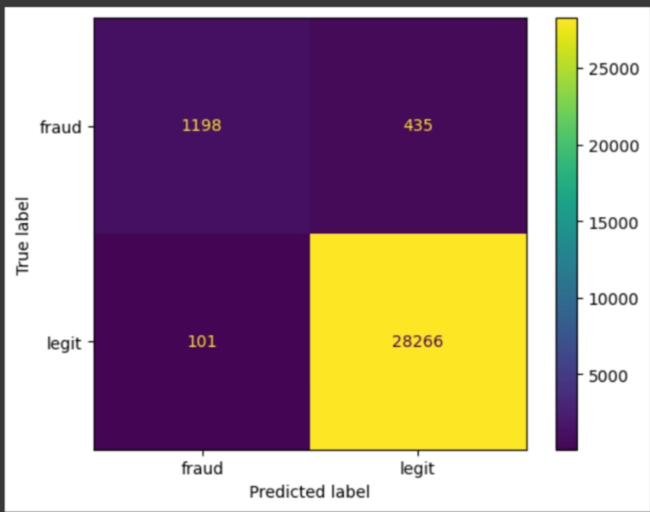
```
_precision = metrics.precision_score(y_test,_y_pred)
_recall = metrics.recall_score(y_test,_y_pred)
_f1_score = metrics.f1_score(y_test,_y_pred)

print('Precision on testing data is:', np.round(_precision,3))
print('Recall on testing data is:', np.round(_recall,3))
print('F1 score on testing data is:', np.round(_f1_score,3))
```

```
Precision on testing data is: 0.985
Recall on testing data is: 0.996
F1 score on testing data is: 0.991
```

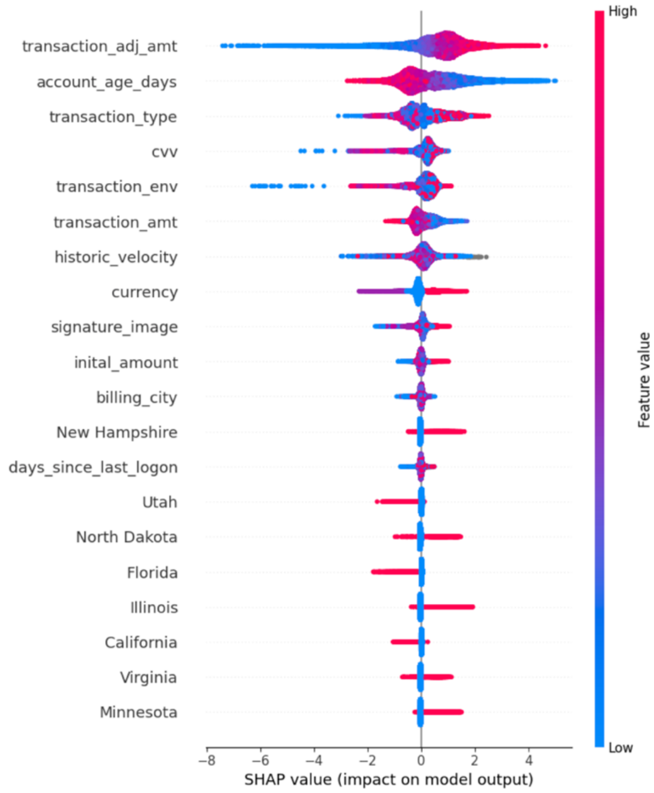
**Confusion Matrix:** From the confusion matrix we can see that model predicts 28266 cases correctly as legit and 1198 cases as fraud. However, it miss labeled 435 fraud cases as legit and 101 legit cases as fraud cases.

The current performance of the model, with a 73% accuracy rate in detecting fraud cases, demonstrates promising capability. However, to enhance its effectiveness, we could explore various avenues for optimization. This might involve refining the algorithm, gathering more diverse training data, or fine-tuning the model parameters. Additionally, conducting a thorough analysis of misclassified cases can provide valuable insights into areas for improvement.

```
#confusion matrix plot
confusion_matrix = metrics.confusion_matrix(y_test,_y_pred)
cm = metrics.ConfusionMatrixDisplay(confusion_matrix=confusion_matrix
                                    ,display_labels=['fraud','legit']
                                    )
cm.plot()
plt.show()
```



## SHAP Analysis




From this bee swarm plot, we can observe that higher transaction_adj_amt values are associated with the positive class, indicating legitimate transactions. Conversely, older accounts seem to assist the model in identifying the negative class, which represents fraudulent activities.

## Important Features