MEMORIAL UNIVERSITY

ENGI-981B Final Project

A Fine-Tuned Supervised Extra Tree Model for Credit Card Fraud Detection

Xin Guan | Student No.: 202195156

Supervisor: Jonathan Anderson

Instructor: Cheng Li

Abstract

The Credit card fraud issue has existed for decades and remains unsolved for customers, businesses, and financial institutions, which causes huge losses every year. This issue destroys personal financial and reputational trust and leaves a long-term restoration process for victims. Machine learning as a tool could tackle this issue from many aspects. Many industries apply this state-of-the-art technology to their businesses, and it is approved that machine learning guarantees better outcomes and less labor cost at the same time. This project intends to adopt supervised machine learning techniques to address the credit card fraud issue and try to find an optimal model along the way. A fine-tuned extra tree classifier was proposed in this project through a few robust analysis steps, such as correlation analysis, k-fold cross-validation, hyperparameter tuning, etc. The dataset used in this project has an inherent flaw: the severely imbalanced ratio between normal and fraudulent transactions. The resampling technique will be examined through data visualization in this project. Due to the imbalanced property of the dataset, the precision and recall curve and average precision are adopted into this project. Finally, a fine-tuned model is proposed, which has improved the recall and f1-score at 10.14% and 4.3%, respectively. Two goals are achieved through the project: conducting an efficient feature selection function and examining all resampling techniques. A relative suggestion is that researchers must pay attention to data bias issues when resampling techniques are applied to their projects.

Keywords: Credit card fraud, feature selection, extra tree classifier, k-fold cross-validation, hyperparameter tuning, average precision

1. Background

1.1 Introduction

The Credit card fraud (CCF) issue is a common and longstanding worldwide security issue that yearly distresses customers, businesses, and financial institutions. Criminals are trying to take over the usage right from the victim. According to the latest Nilson report, global losses from card fraud rose to $32.34 billion in 2021, a 14% increase from the $28.43 billion losses reported in 2020 (Mullen, 2023). The impact of CCF extends beyond financial damages as it may also result in reputational harm and diminished customer trust. Furthermore, individuals subject to CCF may encounter a complicated and exasperating procedure of disputing charges and rebuilding their credit (Lee & Scott, 2017).

In many cases, the ruined personal credit frustrates the victims, which could take years to rebuild. This situation could lead the victims to receive loans hardly and be unlikely to obtain a higher mortgage rate. On the other hand, small businesses like convenience stores or boutiques could also suffer property losses unwillingly. CCF could happen even if no card is present at the store.

There are several types of CCF. The first one is skimming. This fraudulent method involves making illegal copies of credit cards using equipment that can read and copy the original card information. Fraudsters use small electric devices called "skimmers" to extract and store card numbers and other credit card information. A tiny keypad to record the card security code and the skimmer could be attached to any typical cash desk, such as restaurants, bars, and convenience stores. The second fraudulent method is phishing. This one is the most popular fraudulent form to steal customer information. The attacker usually masquerades as a credible bank manager or after-sales service supervisor to send a request to the victims. The fraud action will succeed once the victims open the email and message or click on the link unconsciously (Barker et al., 2008).

1.2 Objective

CCF detection is complex, tedious, and error-prone. It involves multiple vital steps, such as data collection, processing, analysis, and development of decisions based on current and previous transactions. In traditional fraud detection processes, data collection and processing account for most of the time, requiring specialized personnel to verify the data precisely. Secondly, a technical team is essential to develop decision rules for fraud detection schemes. When false positives occur, it is time-consuming and requires human resources to propose revised rules.

In addition, the traditional fraud transaction statistical process is non-dynamic, which means that only obvious fraudulent behaviors can be spotted based on historical fraud transaction data. Some hidden and advanced fraudulent behaviors will be identified as regular transactions. Finally, verification methods of fraudulent identity may cause problems for customers and banks, such as some traditional verification methods include SMS, phone, video, and password question verification. Therefore, traditional fraud detection systems consume a massive amount of time and labor, which are inefficient and may not achieve high accuracy. CCF could lead to significant financial losses for individuals, banks, and companies.

This project proposes a supervised machine learning (ML) model that adopts state-of-the-art artificial intelligence technology to mitigate the CCF issue since this technology has been successfully applied in many industries in recent years. The primary purpose of this project is to enhance the efficiency and accuracy of the traditional fraud detection system. Artificial intelligence technology could improve the efficiency and accuracy of the overall fraud detection process. The model could identify CCF by training a fine-tuned extra tree model and improving the efficiency and accuracy of fraud detection. The system adopts Python as its primary programming language since it is the second most popular programming language in 2023 (Veeraraghavan, 2023). Besides that, it has rich ML and statistical-related libraries, making it more efficient to design a fraud detection system and perform real-time statistics and data processing. The core part of the system is establishing a predictive model, which applies supervised ML algorithms, and train the model with labeled transactions dataset.

1.3 Related work

In this section, several related research papers are summarized in terms of tackling the CCF issue by adopting ML techniques.

Kultur & Calayan (2017) proposed an ensembled model which consists of multiple well-known supervised ML models to tackle the CCF issue. They proposed three ensembled strategies for particular purposes to pick up the optimal models from decision trees, random forest, Bayesian network, Naïve Bayes, support vector machine, and k nearest neighbors. Three strategies are optimistic voting, pessimistic voting, and weighted voting. As a result, optimistic voting gives a 31.59% detection rate of fraudulent transactions, and pessimistic voting and weighted voting give 93.92% and 64.02% detection rate of fraudulent transactions, respectively. Furthermore, their false alarm rate is 0.10%, 13.72%, and 0.75%, respectively. Sivanantham et al. (2021) proposed an ensembled classifier by voting to also solve the CCF issue. To build a voting classifier, the paper uses three supervised ML models, logistic

regression, random forest, and gradient boosting. The hybrid classifier is a wrapper of all classifiers to improve accuracy, recall, and precision. The result is decent, with 99.98% accuracy and a 1.910 false alarm rate. Alam et al. (2021) investigated the imbalance of the dataset and proposed a feature selection method based on the area under the receiver characteristics curve (AUC-ROC). Three different classifiers are adopted in the paper, which are random forest, AdaBoost, and CatBoost. The feature selection step that makes different datasets for building models is based on the performance of AUC-ROC from each classifier. The train test ratio is 9:1 in this paper. The overall accuracy is decent, around 99.90%. Plakandaras et al. (2022) presented an AutoML SaaS platform named JAD to address the CFF issue. The JAD can conduct implementation, feature selection, model training, and building, hyperparameter tuning, etc. Furthermore, it can sort out the prediction and output the result. This tool is user-friendly and easy to retain and update the particular model. Carcillo et al. (2021) proposed a hybrid model involving unsupervised and supervised ML techniques. The unsupervised outlier scores are adapted to their model for extending the feature set, but the outcomes are unexpected. However, the cluster approach, which could cause overfitting and variance issues, is that using the area under the precision and recall curve (AUC-PR) curve is better for the result interpretation. The future work is divided into several directions, all focusing on the accuracy matric. Dal Pozzolo et al. (2018) investigated the downside of the CCF issue and analyzed the alert–feedback interaction of supervised ML models. They pointed out that precision is the most critical alert since the researchers can only review a few alerts in practice. Based on their experiments, they found that feedback is essential during the learning process. Their experiment results also supported this claim since they received less precise alerts when set feedback importance was lower. Keswani et al. (2020) investigated random under-sampling and SMOTE over-sampling techniques that come from the Imblearn library. The paper found that accuracy, sensitivity, precision, and F1-score performance are decent from random under-sampling without misclassification. However, the performance of precision from SMOTE is pretty low compared to under-sampling. The random under-sampling technique dramatically reduces the dataset size and generates data bias. On the other hand, SMOTE's performance depends on the dataset's feature dimension. Alfaiz and Fati (2022) proposed a model trying to improve the current CCF situation. The approach is divided into two parts. At first, three optimal ML algorithms are picked up. After that, combined these three algorithms along with nineteen resampling techniques. Evaluation metrics in their paper are precision, recall, F1-score, accuracy, and the area under the receiver operation characteristic curve(AUC). The alternative algorithms are decision tree, random forest, logistic regression, K-nearest neighbors, naïve Bayes,

gradient boosting machines, light gradient boosting machines, extreme gradient boosting, and category boosting. The resampling techniques are 11 under-sampling, six over-sampling, and two under and over-sampling combinations. As a result, the best model is K-nearest neighbors along with cat boost, and it received 97.94%, 95.91%, and 87.4% for the AUC, recall, and F1-score, respectively.

1.4 Paper structure

The paper will be divided into five sections. Section two will present algorithms and techniques adopted into this project. Furthermore, the data visualization and analysis outcomes will be shown along with algorithms and functions. ML model selection and construction are included in this section as well. All ML results, such as classification reports, confusion matrix, and the precision-recall curve, will be presented in section three. This project's limitations and future work are delivered right after the result section in section four. In section five, a conclusion will be given based on the achievements of this project.

1.5 Datasets clarification

To understand this project, datasets and split data like train-test sets are presented briefly in the list below,

1. df, which stores unmodified original data that has been read from a CSV file.
2. df_obvious_feats, used for data visualization to compare the 'Time' and 'Amount' features.
3. df_v_feats, used for data visualization between each numerical feature.
4. df_corr_feats, used for data visualization between each strongly correlated feature.
5. df_w, used to store the weighted-sampling dataset, in which the fraud transactions account for 10% of the overall dataset.
6. df_under is applied RandomUnderSampler function from Imblearn library to under-sample df_corr_feats dataset.
7. df_over is applied SMOTE function from Imblearn library to over-sample df_corr_feats dataset.

The name of the train-test sets will be listed below,

1. X_train_v, X_test_v, y_train_v, y_test_v are split from df_v_feats and used for final model construction and final model result comparison.
2. X_train_corr, X_test_corr, y_train_corr, y_test_corr are split from df_corr_feats and used for all resampling techniques, final model construction, and final model result comparison.

2. Methodology

This section will describe the project from core concept to implementation. This section will present all progress, such as Python libraries selection, data exploration with data visualization, dataset resampling techniques, model selection and pre-construction, hyperparameter tuning, and final model construction.

2.1 Python libraries selection

Appropriate tools must be applied as a core part of this project in grasping a comprehensive understanding of the dataset. Many popular Python libraries have the capability and will be introduced into this project to keep the process of data manipulation and reshaping straightforward. The first two famous libraries are Pandas and Numpy. The data science and ML communities use these two libraries as a default setup, making data manipulation and transformation smooth throughout their project. The Second part is data visualization libraries, which are Matplotlib and Seaborn. Data visualization is an indispensable portion of an ML project due to the capability to transform numerical values into visible figures. The hidden patterns and relationships will be shown through this process. The next one is Scikit-learn. This library is crucial for data science and ML projects since it provides many tools, such as scalers, classifiers, and assessment metrics. It mainly scales the data, fits them into unique learning models, and outputs the result.

After these standard libraries were imported, two more parts were applied to this project. The first part is the data resampling library. Due to the imbalanced property of the dataset used in this project, the resampling library Imblearn was introduced to tackle the imbalance issue. The outcomes will be compared with other datasets afterward to make the final result more reasonable and reliable.

Last, Jupyter Notebook, Colab Notebook, and Pycaret are also applied in this project. Jupyter Notebook and Colab are two interactive development environments for data science and ML. It makes running code, editing comments, and interacting with outcomes simple and smooth. Pycaret is a less code ML library used to compare the original dataset's models and pick the suitable one for the project. All applied libraries are shown in Figures 1 and 2.

2.2 Data exploration

The initial procedure of every data science and ML project is to get statistical insight from the dataset. This step is a crucial milestone in every ML project. A comprehensive data exploration will lead to success. This section

will present the dataset through statistical analysis and correlation analysis. Data visualization, another essential tool for ML projects, is applied to characterize the dataset clearly and comprehensively. The final step of this section is to run a feature selection based on the analyses above to pick up strongly correlated features for further progress. The CCF dataset consists of hundreds of thousands of numbers, and we could barely grasp the crucial information from them explicitly without data processing. Data processing, such as statistical and correlation analysis, can provide a thorough understanding of the dataset. On the other hand, data visualization as a tool could offer a general idea of the dataset from a different perspective and assist us in drawing a relatively comprehensive understanding of the dataset.

Figure 1

Python Libraries Used in This Project

```
# import basic libraries
import numpy as np
import pandas as pd

# import plot libraries
import matplotlib.pyplot as plt
import seaborn as sns

# dataset split for train and test set
from sklearn.model_selection import train_test_split

# import imblearn library for resampling
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE

# model related libraries
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import GridSearchCV, StratifiedKFold, cross_val_score
from imblearn import FunctionSampler
from imblearn.pipeline import Pipeline
```

Figure 2

Python Libraries Used in This Project Continued

```
# Assessment metrics
from sklearn.metrics import (
    classification_report,
    ConfusionMatrixDisplay,
    average_precision_score,
    precision_recall_curve,
    PrecisionRecallDisplay,
)

# several config
%config InlineBackend.figure_format = 'png'
%matplotlib inline
%load_ext autoreload
%autoreload 2
```

2.2.1 Statistical analysis

The dataset was picked up from the website Kaggle, which is widely recognized by machine learning and data science communities. The dataset consists of real transactions which European credit card holders made in September 2013 (MACHINE LEARNING GROUP – ULB, 2021). The dataset was read into Jupyter Notebook as a Pandas data frame. This data structure makes it simple to access and manipulate the whole dataset. Figures 3 and 4 show that the original dataset consists of 31 features consisting of 28 numerical features, 'Time', 'Amount', and 'Class' features after calling head() function. The 28 numerical features are transformed by the PCA technique, which aims to protect the confidential information of cardholders. The 'Amount' feature is the expenditure from one transaction. The 'Class' feature marks whether the transaction is fraudulent or normal. The 'Time' feature contains the seconds elapsed between each transaction and the first transaction in the dataset.

Figure 3

Data Blocks from the Original Dataset

```
1 df.head()
```

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

5 rows × 31 columns

Figure 4

Data Blocks from the Original Dataset Continued

V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
-0.018307	0.277838	-0.110474	0.066928	0.128539	-0.189115	0.133558	-0.021053	149.62	0
-0.225775	-0.638672	0.101288	-0.339846	0.167170	0.125895	-0.008983	0.014724	2.69	0
0.247998	0.771679	0.909412	-0.689281	-0.327642	-0.139097	-0.055353	-0.059752	378.66	0
-0.108300	0.005274	-0.190321	-1.175575	0.647376	-0.221929	0.062723	0.061458	123.50	0
-0.009431	0.798278	-0.137458	0.141267	-0.206010	0.502292	0.219422	0.215153	69.99	0

The next step is to check the statistical description of each feature. Figures 5 and 6 show a general statistical description of the original dataset after calling describe() function. After the statistical description, we need to

ensure that the value of each data block is not null. Figure 7 shows the information in the dataset after calling info() function, and there are no null values.

Figure 5

The Statistical Description of the Original Dataset

```
1 df.describe()
```

	Time	V1	V2	V3	V4	V5	V6	V7	V8	V9
count	284807.000000	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05
mean	94813.859575	1.168375e-15	3.416908e-16	-1.379537e-15	2.074095e-15	9.604066e-16	1.487313e-15	-5.556467e-16	1.213481e-16	-2.406331e-15
std	47488.145955	1.958696e+00	1.651309e+00	1.516255e+00	1.415869e+00	1.380247e+00	1.332271e+00	1.237094e+00	1.194353e+00	1.098632e+00
min	0.000000	-5.640751e+01	-7.271573e+01	-4.832559e+01	-5.683171e+00	-1.137433e+02	-2.616051e+01	-4.355724e+01	-7.321672e+01	-1.343407e+01
25%	54201.500000	-9.203734e-01	-5.985499e-01	-8.903648e-01	-8.486401e-01	-6.915971e-01	-7.682956e-01	-5.540759e-01	-2.086297e-01	-6.430976e-01
50%	84692.000000	1.810880e-02	6.548556e-02	1.798463e-01	-1.984653e-02	-5.433583e-02	-2.741871e-01	4.010308e-02	2.235804e-02	-5.142873e-02
75%	139320.500000	1.315642e+00	8.037239e-01	1.027196e+00	7.433413e-01	6.119264e-01	3.985649e-01	5.704361e-01	3.273459e-01	5.971390e-01
max	172792.000000	2.454930e+00	2.205773e+01	9.382558e+00	1.687534e+01	3.480167e+01	7.330163e+01	1.205895e+02	2.000721e+01	1.559499e+01

8 rows × 31 columns

Figure 6

The Statistical Description of the Original Dataset Continued

V21	V22	V23	V24	V25	V26	V27	V28	Amount	Class
2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	2.848070e+05	284807.000000	284807.000000
1.654067e-16	-3.568593e-16	2.578648e-16	4.473266e-15	5.340915e-16	1.683437e-15	-3.660091e-16	-1.227390e-16	88.349619	0.001727
7.345240e-01	7.257016e-01	6.244603e-01	6.056471e-01	5.212781e-01	4.822270e-01	4.036325e-01	3.300833e-01	250.120109	0.041527
-3.483038e+01	-1.093314e+01	-4.480774e+01	-2.836627e+00	-1.029540e+01	-2.604551e+00	-2.256568e+01	-1.543008e+01	0.000000	0.000000
-2.283949e-01	-5.423504e-01	-1.618463e-01	-3.545861e-01	-3.171451e-01	-3.269839e-01	-7.083953e-02	-5.295979e-02	5.600000	0.000000
-2.945017e-02	6.781943e-03	-1.119293e-02	4.097606e-02	1.659350e-02	-5.213911e-02	1.342146e-03	1.124383e-02	22.000000	0.000000
1.863772e-01	5.285536e-01	1.476421e-01	4.395266e-01	3.507156e-01	2.409522e-01	9.104512e-02	7.827995e-02	77.165000	0.000000
2.720284e+01	1.050309e+01	2.252841e+01	4.584549e+00	7.519589e+00	3.517346e+00	3.161220e+01	3.384781e+01	25691.160000	1.000000

Figure 7

Null Values Checking for the Original Dataset

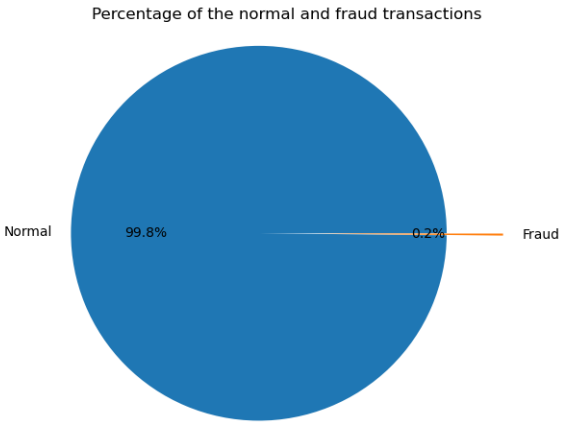
```
1 df.info()
```

```
<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
```

A pie chart from Figure 8 shows that the normal and fraudulent transactions are severely imbalanced. Fraud transactions only account for 0.2% of the total transactions, which can tell from the records that there only have 492 fraud transactions out of 284,807.

Figure 8

The Percentage of Normal and Fraud Transactions



The crucial step in statistical analysis is to get a brief grasp of the distribution of each feature. Figure 9 shows the distribution of each feature. As the figure shows, all features from V1 to V28 are a sort of standard distribution. Most of them are severely skewed. Distribution of the 'Amount' feature is gathered around 0 since most of the transactions are small-scale consumption. The 'Time' feature has two peaks, which means people tend to purchase items at a certain amount of time since the 'Time' feature contains the seconds elapsed between each transaction and the first transaction in the dataset.

2.2.2 Correlation analysis

After statistical analysis, we need to dive deeper into our dataset. As shown in the previous section, the relationship between each feature is now unclear. This section introduces correlation analysis in terms of tackling feature selection issues.

The first step is to check the correlation coefficient between each feature. Figure 10 shows the correlation coefficient matrix. The correlation method is Pearson, which is the standard linear correlation coefficient set to the default of corr() function. The correlation coefficient matrix will be analyzed into two parts: the first focuses on the correlations between every 28 numerical features, and the second focuses on the correlations between the ‘Class’ and each numerical feature. As Figure 10 shows, the correlations between each numerical feature are

relatively weak, which means they are independent. Figure 11 shows features that correlate stronger than threshold 0.12, which I set up as an intermediate level of correlation based on the scale shown in the figure.

Figure 9

The Distribution of Each Numerical Feature

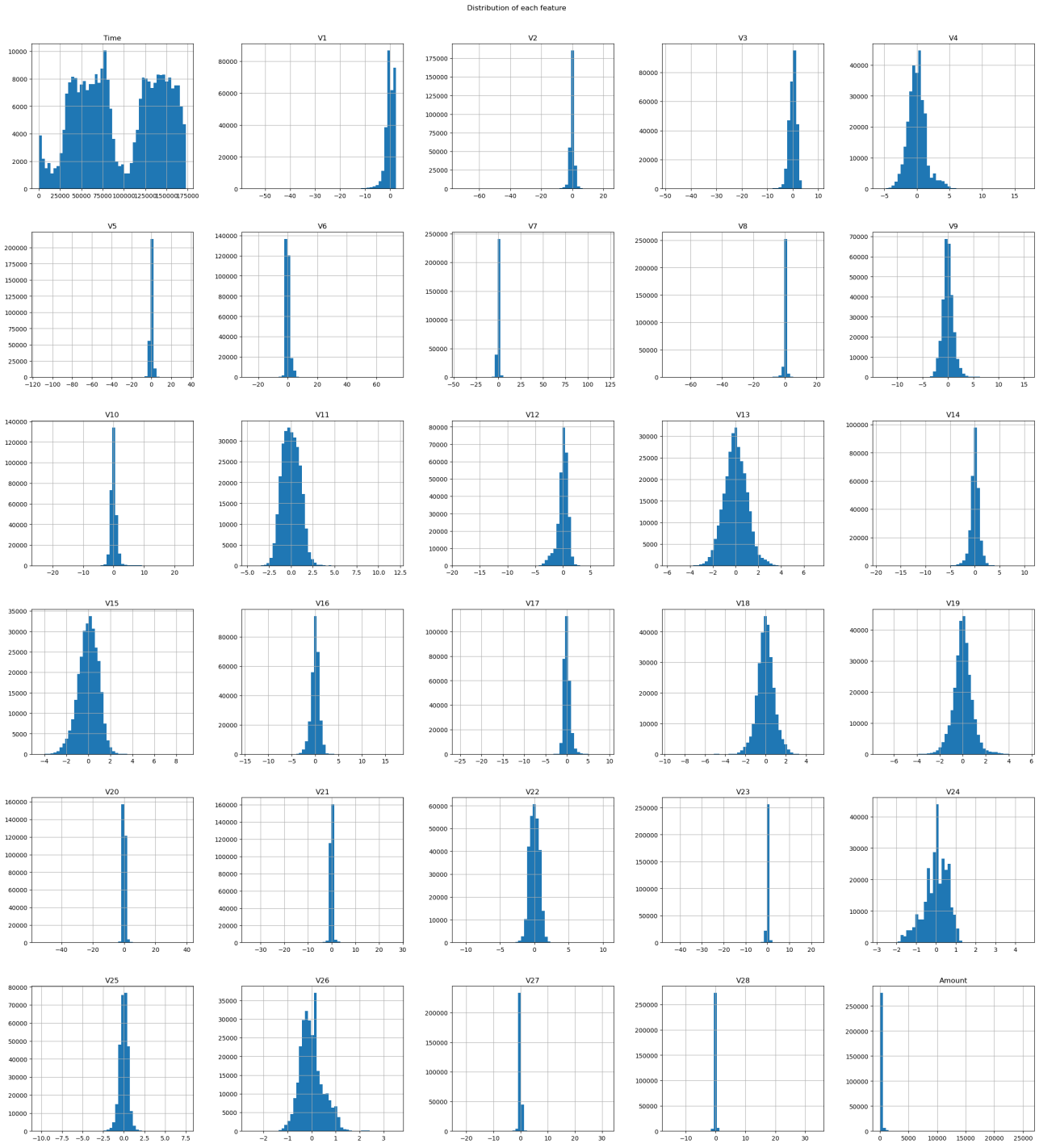


Figure 10

The Correlation Coefficient Matrix from the Original Dataset

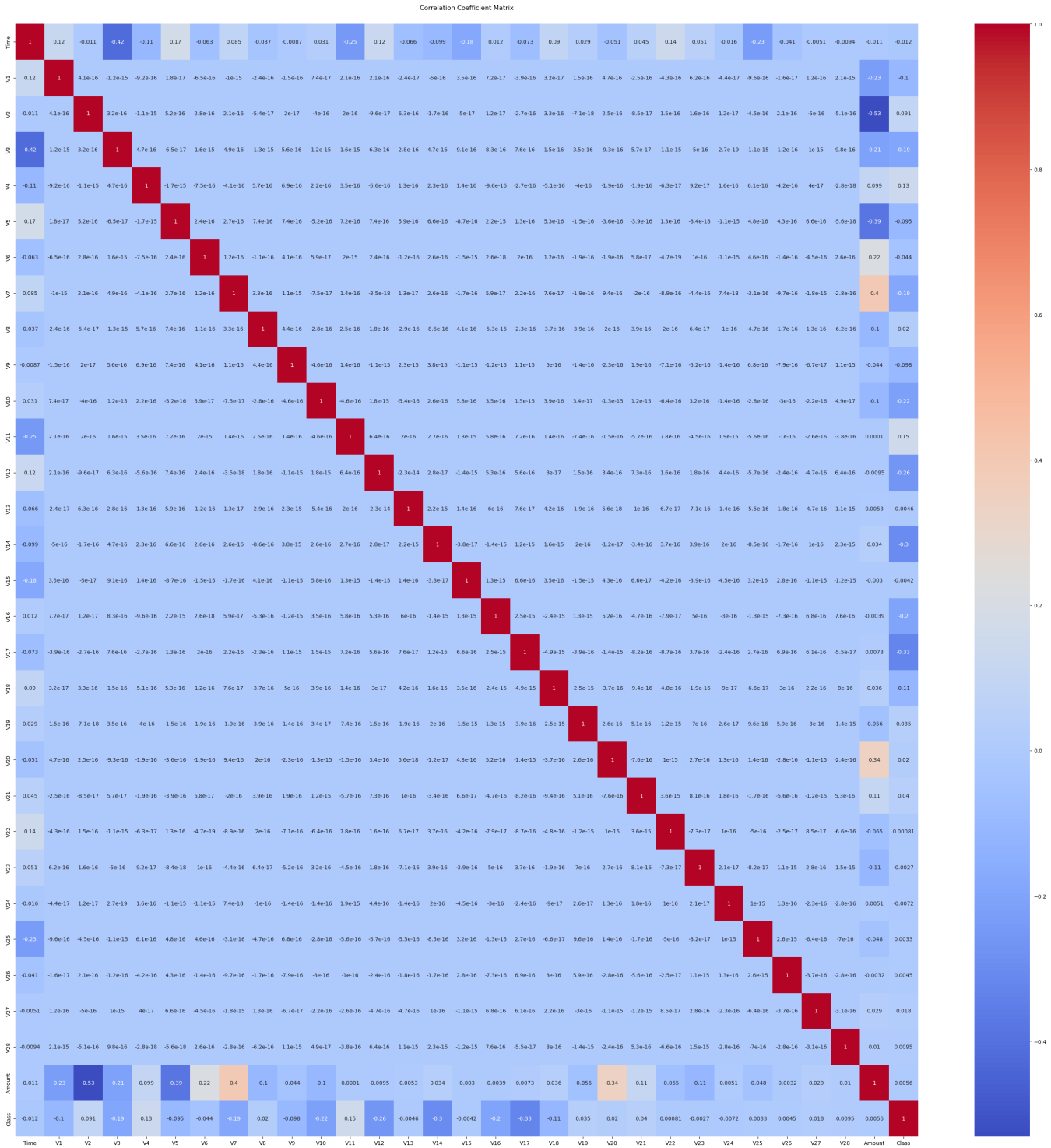


Figure 11

Correlation Scores between Each Numerical Feature and the 'Class' Feature

```python
def corr_list(dataset):

    # correlations between nemerical features and the Class feature
    corr = dataset.corr()['Class'].to_dict()
    output_dict = {}

    for key, value in corr.items():
        if abs(value) > 0.12:
            output_dict[key] = round(value, 3)

    return output_dict
```

```python
fea_corr = corr_list(df)
fea_corr
```

```
{'V3': -0.193,
 'V4': 0.133,
 'V7': -0.187,
 'V10': -0.217,
 'V11': 0.155,
 'V12': -0.261,
 'V14': -0.303,
 'V16': -0.197,
 'V17': -0.326,
 'Class': 1.0}
```

2.2.3 Feature selection

This section will select the specific features using statistical and correlation analysis from the previous part and the data visualization techniques. The first step is checking the relationship between the 'Time' and 'Amount' features. Figure 12 shows that they have no linear relationship and no correlation between fraud transactions and the 'Time' or 'Amount', respectively. This phenomenon also complies with reality since people can purchase goods anytime. Another summary from this figure is that the 'Time' feature is unrelated to the transactions, whether it is fraud or normal. However, an exciting discovery is that fraud transactions seem more likely to happen around small-value transactions. Still, the result is not robust due to the lack of fraudulent transactions and research samples.

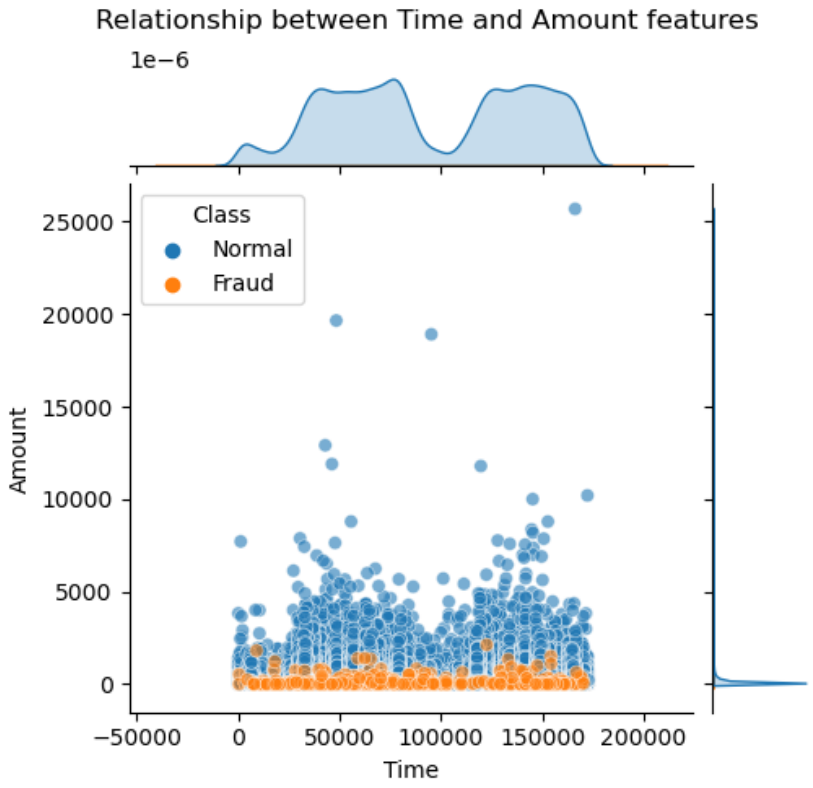
After comparing the 'Time' and 'Amount' features, hidden relationships between numerical features will be investigated through data visualization. Figure 13 shows Seaborn pair plots of weakly correlated pairs between each numerical feature and the 'Class' feature. The numerical features that have a weak correlation with the 'Class' feature are V1, V2, V5, V6, V8, V9, V13, V15, V18, V19, V20, V21, V22, V23, V24, V25, V26, V27, and V28. The diagonal plots show the distribution of each feature. As the figure shows, each pair of numerical features has no

linear relationship, and there is no specific cluster relationship that contributes to normal or fraud transactions. Hence, these features will be dropped afterward and not used for further dataset split and model construction.

Figure 14 shows the Seaborn pair plots of strongly correlated pairs between each numerical feature and the 'Class' feature. The numerical features strongly correlated with the 'Class' feature are V3, V4, V7, V10, V11, V12, V14, V16, and V17. The diagonal plots show the distribution of each feature. As the figure shows, V3, V4, V7, and V10 have no linear relationship with other numerical features. However, V11, V12, V14, V16, and V17 have a strong linear relationship to some extent. So, the numerical features with linear relationship will be removed based on the correlation coefficient score shown in Figure 11, which are V11, V12, and V16. Figure 15 shows the pair plots after these three numerical features were removed. These features will be kept and used for the final dataset split and model construction.

Figure 12

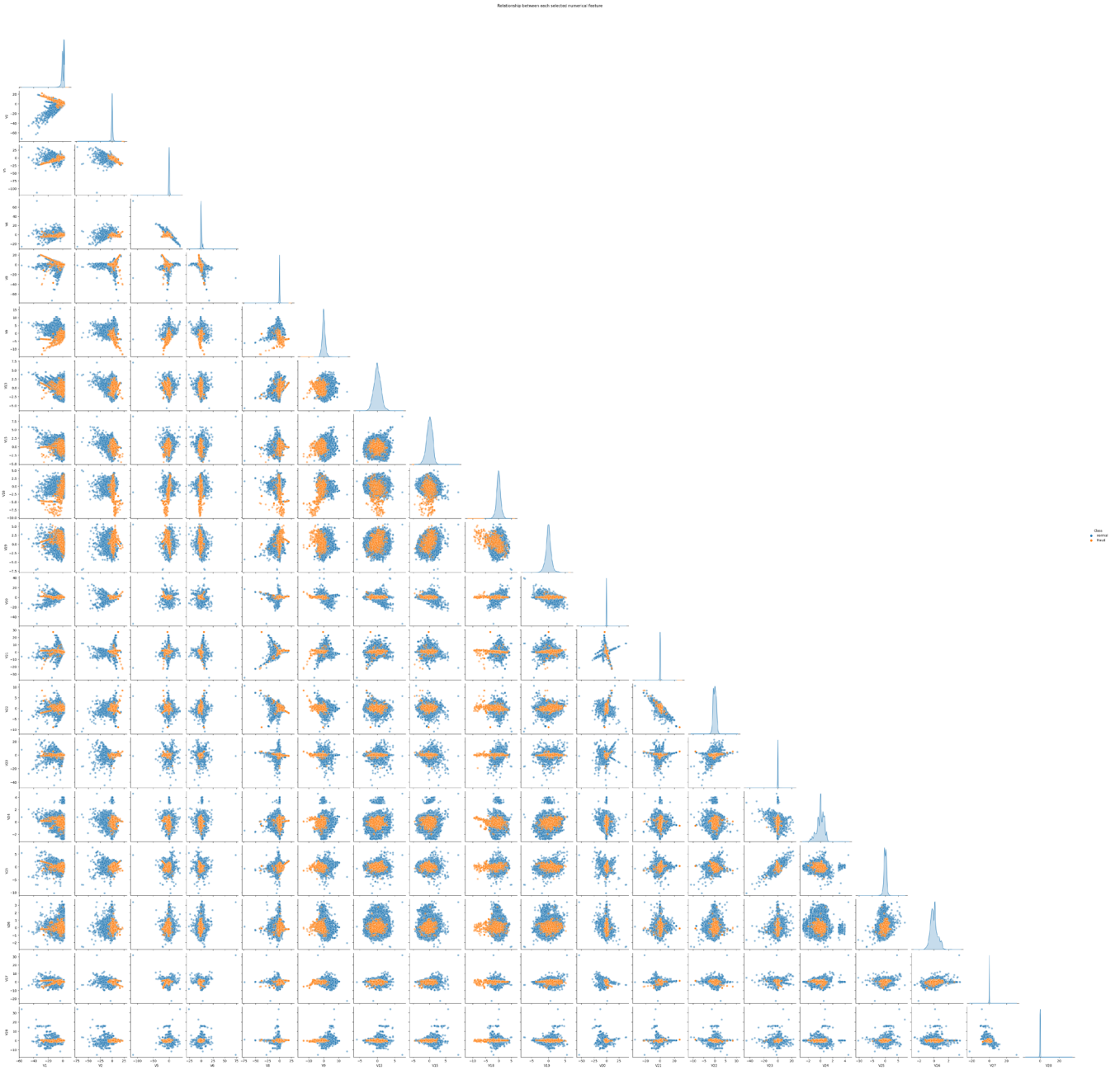
The Scatter Plot between the 'Time' and 'Amount' Features



Note. This scatter plot shows the cluster relationship between the ‘Time’ and ‘Amount’ feature that correspond with the label ‘Class’. The figure clearly indicates no cluster and linear relationship between them.

Figure 13

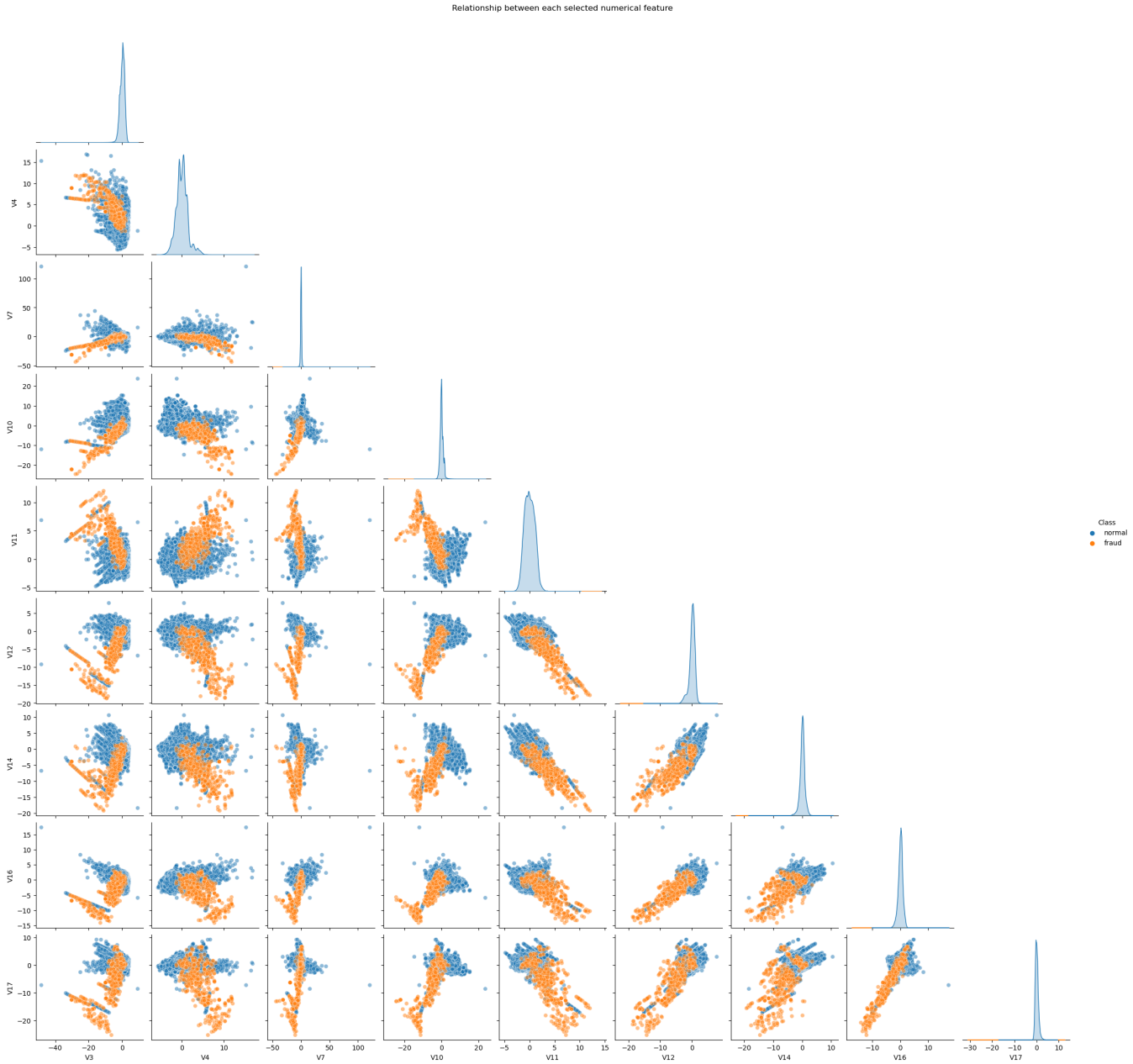
The Pair Plots Generated from Weak Correlated Features Corresponding with the 'Class' Feature



Note. Each scatter plot is between two weakly correlated features. The blue dots represent normal transactions. The orange dots represent fraudulent transactions. The Line plots sit along the diagonal are the distribution of each feature.

Figure 14

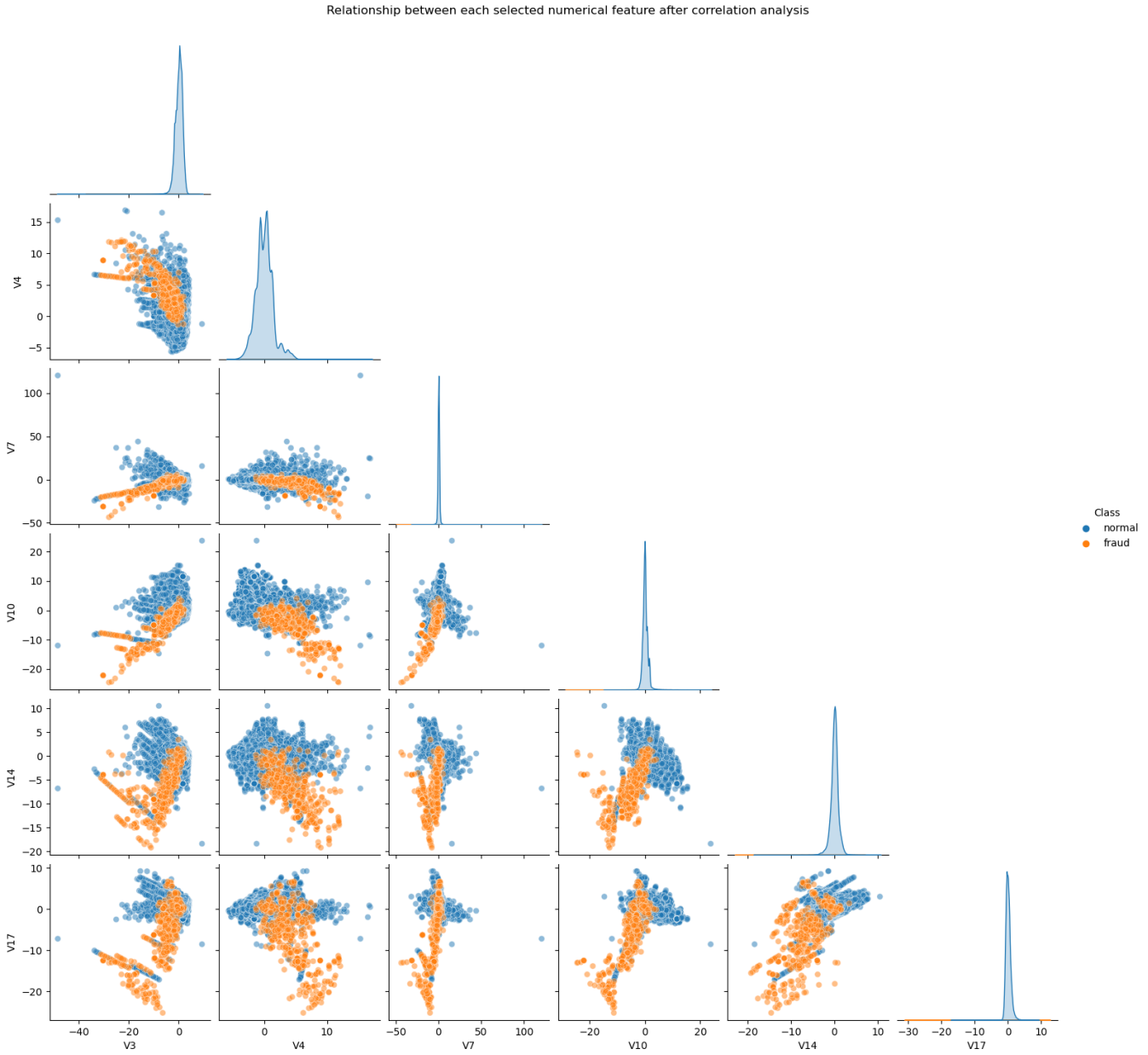
The Pair plots Generated from Strongly Correlated Features Corresponding with the 'Class' Feature



Note. Each scatter plot is between two strongly correlated features. The blue dots represent the normal transactions. The orange dots represent fraudulent transactions. The Line plots sit along the diagonal are the distribution of each feature. V11, V12, V14, and V16 have a strong linear relationship with V17. Some of them will be dropped further based on correlation scores shown in Figure 11.

Figure 15

Final Pair Plots Generated from Strongly Correlated Features Corresponding with the 'Class' Feature



2.3 Dataset split

In this section, the original dataset will be split into train-test sets in general for further parts based on the results from the statistical analysis, the correlation analysis, and the data visualization in the previous section. The code of the dataset split is shown in Figure 16. The test set size is set to 20% of the total dataset since five-fold stratified cross-validation is applied to this project, which means each fold includes 79 fraud transactions, and the

test set has 98 fraud transactions. From this point, the original dataset will be split into two groups of train-test datasets in general.

There have two groups of train-test sets briefly. The first train-test set consists of all numerical features from V1 to V28, but the 'Time' and 'Amount' features are excluded based on feature selection from Figure 12. The train-test set are called X_train_v, X_test_v, y_train_v, y_test_v, respectively. The purpose of keeping this group is to compare results with the strongly correlated train-test set. The second train-test set consists of six strongly correlated numerical features selected based on correlation analysis and data visualization from Figure 15. The train-test sets are X_train_corr, X_test_corr, y_train_corr, y_test_corr, respectively. This group will be resampled into three different datasets through the pipeline and used for final model building and result afterward.

Figure 16

Custom Train Test Split Function

```
# customed train test split function
def data_split(dataset, sp_size=0.2):

    #columns should be a list or single str
    input = dataset.drop('Class', axis=1)
    target = dataset.Class
    X_train, X_test, y_train, y_test = train_test_split(input, target, test_size=sp_size,
                                                        random_state=1, stratify=target)

    return X_train, X_test, y_train, y_test
```

```
X_train_v, X_test_v, y_train_v, y_test_v = data_split(df_v_feats)
X_train_corr, X_test_corr, y_train_corr, y_test_corr = data_split(df_corr_feats)
```

Let's take a glance at the train-test sets. The details of X_train_v and X_train_corr are shown in Figures 17 to 20, respectively. A few things need to be clear at this point. The index from the original dataset is the leftmost column with no label at the top row. The random state variable of train_test_split is set to one for all train-test sets to ensure data consistency through every split manipulation. The 'Class' feature is separated as a single Pandas Series, which is filled into the target sets y_train_v and y_train_corr, when using Scikit-learn train_test_split function.

2.4 Dataset resampling

The resampling technique could be an optional solution for this issue because the dataset presented in this project is severely imbalanced. In this section, df_corr is transformed into three different resampled datasets through three distinct resampling techniques, which are weighted-resampling by using Pandas sample function,

over-sampling by SMOTE from Imblearn library, and under-sampling by RandomUnderSampler from Imblearn library. In the weighted-sampling dataset, the fraud transactions account for 10% of the total resampled dataset. By contrast, the under-sampling and over-sampling datasets are set to a balanced ratio between normal and fraud transactions, meaning they both account for 50% of the total resampled dataset. Finally, three resampled datasets are presented through scatter plots for comparison.

Figure 17

The First Five Rows of the X_Train_v Dataset

```
1 X_train_v.head()
```

	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10
9341	1.148607	-0.004154	-0.231106	1.124256	0.696077	1.177523	-0.160142	0.201478	1.568269	-0.320290
210529	-0.910538	-0.971254	1.033829	-1.514111	-1.068542	0.148626	-0.660712	0.694582	-0.563151	-0.396853
51525	1.222501	0.491904	-0.082522	0.987913	0.164676	-0.845453	0.515114	-0.258261	-0.667442	0.186485
128333	1.184303	-0.066290	0.400333	-0.048278	-0.447460	-0.521259	-0.103204	-0.044640	-0.032670	-0.048690
252025	-0.484059	0.439377	-2.033102	-3.398765	2.209264	3.175789	-0.566306	0.423994	-1.281704	0.137172

5 rows × 28 columns

Figure 18

Checking the Shape and Values of All Numerical Features Train-test Set

```
1 X_train_v.shape
```

```
(227845, 28)
```

```
1 y_train_v.value_counts()
```

```
0    227451
1       394
Name: Class, dtype: int64
```

```
1 y_test_v.value_counts()
```

```
0    56864
1       98
Name: Class, dtype: int64
```

Figure 19

The First Five Rows of the X_Train_corr Dataset

```
1 X_train_corr.head()
```

	V3	V4	V7	V10	V14	V17
9341	-0.231106	1.124256	-0.160142	-0.320290	1.788162	0.170873
210529	1.033829	-1.514111	-0.660712	-0.396853	-0.561552	0.210419
51525	-0.082522	0.987913	0.515114	0.186485	0.593085	-0.612390
128333	0.400333	-0.048278	-0.103204	-0.048690	0.334798	-0.747442
252025	-2.033102	-3.398765	-0.566306	0.137172	0.442717	-0.073262

Figure 20

Checking the Shape and Value of Strongly Correlated Features Train-test Set

```
X_train_corr.shape
```

```
(227845, 6)
```

```
y_train_corr.value_counts()
```

```
0    227451
1       394
Name: Class, dtype: int64
```

```
y_test_corr.value_counts()
```

```
0    56864
1       98
Name: Class, dtype: int64
```

The first resampling step is to write a custom function that transforms the df_corr into three distinct resampled datasets for data visualization. The code is shown in Figures 21 and 22. The weighted-resampling function uses a simple sample() function from Pandas, and the over-sampling and under-sampling functions use simple SMOTE and RandomUnderSampler from Imblearn library. The random state variable of resampling functions is set to one for all to ensure data consistency through every dataset manipulation. The resampling results are shown in Figures 23, 24, and 25 by calling head() and value_counts() functions.

Figure 21

The Custom Weighted-sampling Function by Using Pandas sample() Function

```
# weighted resampling function for data visualization by using pandas
# input is X_train set, target is y_train set
# fraud transactions account for 10% of total resampled dataset
def res_weighted(input, target):

    # concatenate input and target into a new dataframe
    df_ = pd.concat([input, target], axis=1)

    # split normal and fraud transactions
    normal = df_.query('Class==0')
    fraud = df_.query('Class==1')

    # concatenate resampled normal transactions with fraud transactions
    normal_res = normal.sample(n=(len(fraud) * 9), random_state=1)
    df_res = pd.concat([normal_res, fraud])

    return df_res
```

Figure 22

The Custom Over and Under Sampling Function by Using Imbearn Library

```
# over and under resampling function for data visualization by using imblearn
# input is X_train set, target is y_train set
# method is either oversampling or undersampling
# normal and fraud transactions both account for 50% of total resampled dataset
def resample(input, target, method):

    # choose resampling method
    if method == 'oversample':
        model = SMOTE(random_state=1)
    elif method == 'undersample':
        model = RandomUnderSampler(random_state=1)
    else:
        raise ValueError("Method must be provided!")

    # resampling input and target and return a pandas dataframe
    X_res, y_res = model.fit_resample(input, target)
    X_res['Class'] = y_res

    return X_res
```

Figure 23

The Custom Weighted-sampling Dataset Structure

```
df_w = res_weighted(X_train_corr, y_train_corr)
```

```
df_w.head()
```

	V3	V4	V7	V10	V14	V17	Class
200885	1.870001	-3.215858	-0.003371	-1.972292	-0.599401	-0.984886	0
168773	-0.451852	-0.796593	1.488813	-0.498832	0.857930	-0.335348	0
234668	-1.666090	0.456784	0.517232	-0.421143	0.164872	-0.104433	0
105882	2.122859	1.676907	-0.533538	-0.299473	-0.065641	0.948603	0
243972	-0.274683	-3.795781	1.281695	-2.687676	1.095618	-0.487153	0

```
df_w.Class.value_counts()
```

```
0    3546
1     394
Name: Class, dtype: int64
```

```
len(df_w)
```

```
3940
```

Figure 24

The Under-sampling Dataset Structure by Using RandomUnderSampler

```
1 df_under = resample(X_train_corr, y_train_corr, 'undersample')
```

```
1 df_under.head()
```

	V3	V4	V7	V10	V14	V17	Class
0	1.870001	-3.215858	-0.003371	-1.972292	-0.599401	-0.984886	0
1	-0.451852	-0.796593	1.488813	-0.498832	0.857930	-0.335348	0
2	-1.666090	0.456784	0.517232	-0.421143	0.164872	-0.104433	0
3	2.122859	1.676907	-0.533538	-0.299473	-0.065641	0.948603	0
4	-0.274683	-3.795781	1.281695	-2.687676	1.095618	-0.487153	0

```
1 df_under.Class.value_counts()
```

```
0    394
1    394
Name: Class, dtype: int64
```

```
1 len(df_under)
```

```
788
```

Figure 25

The Over-sampling Dataset Structure by Using SMOTE

```
1 df_over = resample(X_train_corr, y_train_corr, 'oversample')
```

```
1 df_over.head()
```

	V3	V4	V7	V10	V14	V17	Class
0	-0.231106	1.124256	-0.160142	-0.320290	1.788162	0.170873	0
1	1.033829	-1.514111	-0.660712	-0.396853	-0.561552	0.210419	0
2	-0.082522	0.987913	0.515114	0.186485	0.593085	-0.612390	0
3	0.400333	-0.048278	-0.103204	-0.048690	0.334798	-0.747442	0
4	-2.033102	-3.398765	-0.566306	0.137172	0.442717	-0.073262	0

```
1 df_over.Class.value_counts()
```

```
0    227451
1    227451
Name: Class, dtype: int64
```

```
1 len(df_over)
```

```
454902
```

2.4.1 Data visualization after dataset resampling

After dataset reconstruction, the same feature should be compared between three resampled datasets. Four data visualization groups come from three different resampled datasets and the unsampled from strongly correlated dataset. One thing must be clear that the amount of fraud transactions remains the same throughout all four groups. However, the only difference is the normal transactions, which were resampled through different functions. The topmost and rightmost line plots from Figures 26 to 33 show the distribution of each feature. As the figures show, the distribution of each feature from the weighted-sampling dataset did not change too much. By contrast, the distribution of each feature from the other two resampled datasets changed a lot since the ratio between normal and fraud is balanced to one because more synthetic data points are added and real values are removed from the majority part, respectively. This type of data manipulation reinforces the data flaw. This phenomenon is shown in the scatter plot as well. So, resampling for a severely imbalanced dataset might not be a practical solution since it will introduce data bias eventually. However, concluding this outcome is too early and naive since we do not even fit the dataset into a robust model. Hence, these resampled datasets will be kept and fit into the final model for further comparison.

Figure 26

The Scatter Plot between Feature 'V4' and 'V17' from the Unsampled Dataset

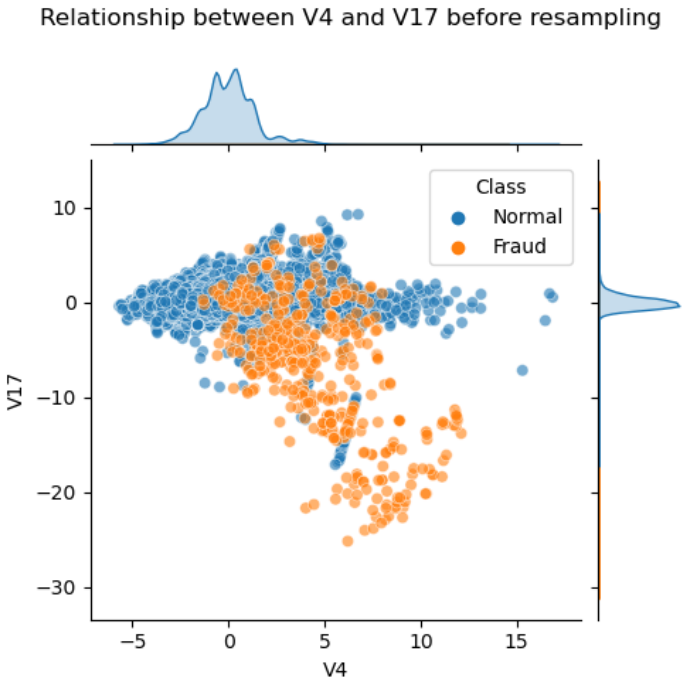


Figure 27

The Scatter Plot between Feature 'V3' and 'V14' from the Unsampled Dataset

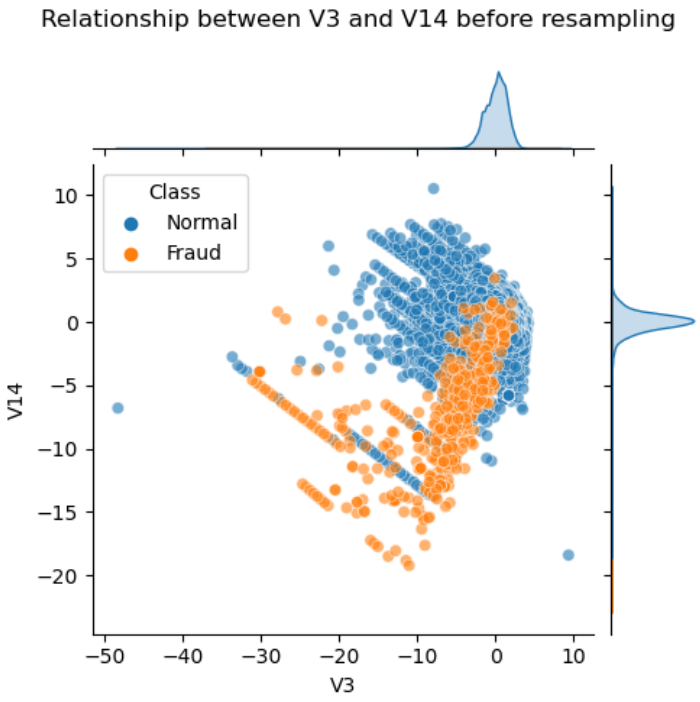


Figure 28

The Scatter Plot between Feature 'V4' and 'V17' from the Weighted-sampling Dataset

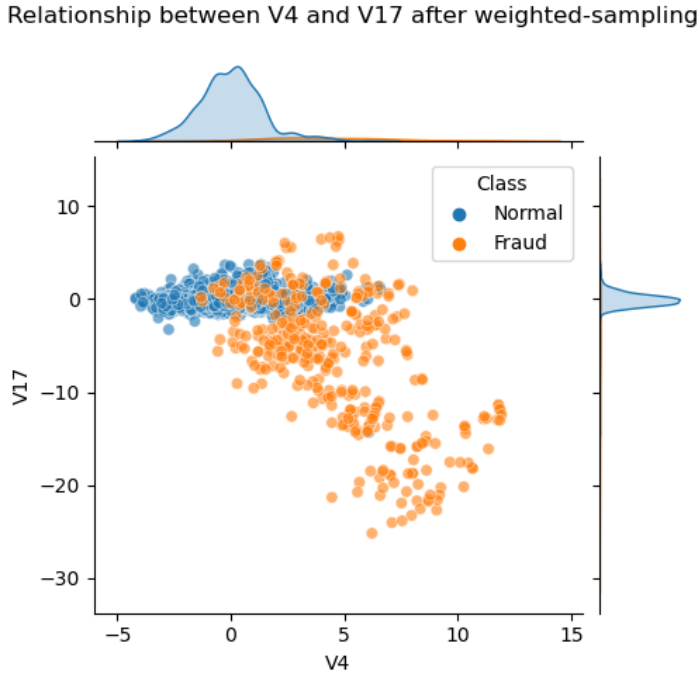


Figure 29

The Scatter Plot between Feature 'V3' and 'V14' from the Weighted-sampling Dataset

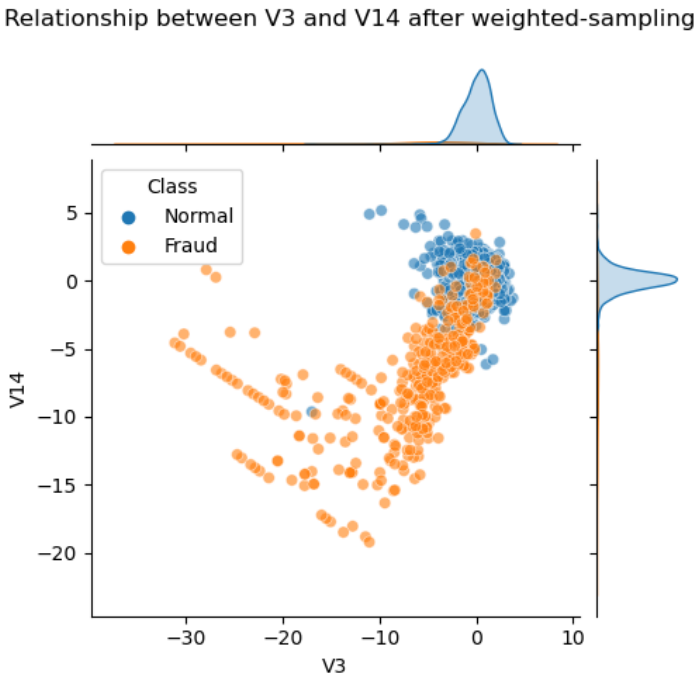


Figure 30

The Scatter Plot between Feature 'V4' and 'V17' from the Under-sampling Dataset

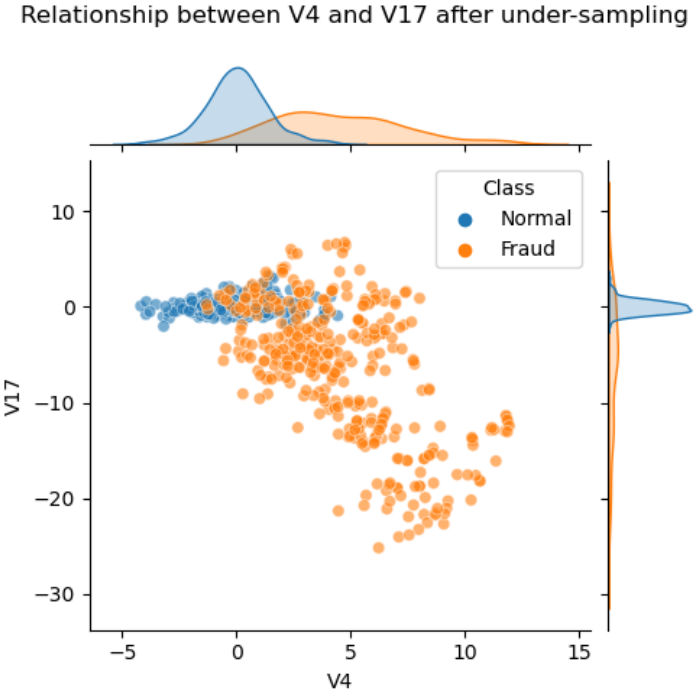


Figure 31

The Scatter Plot between Feature 'V3' and 'V14' from the Under-sampling Dataset

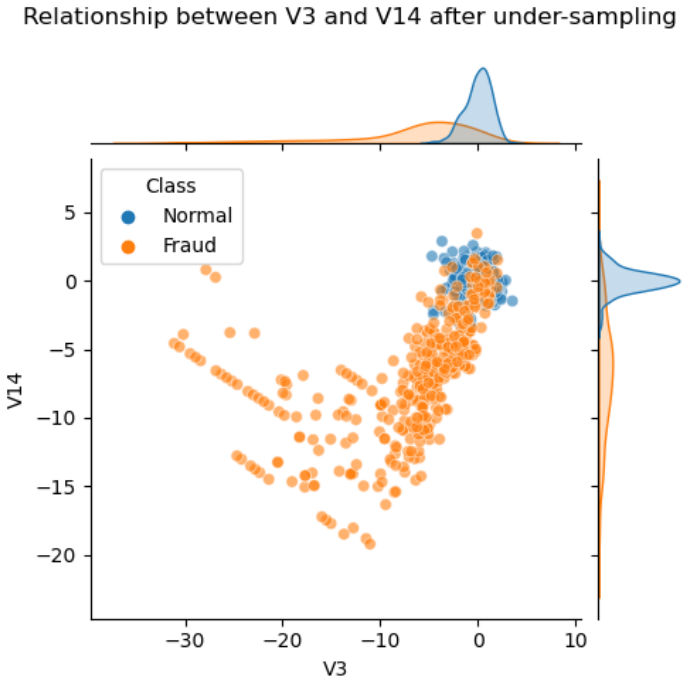


Figure 32

The Scatter Plot between Feature 'V4' and 'V17' from the Over-sampling Dataset

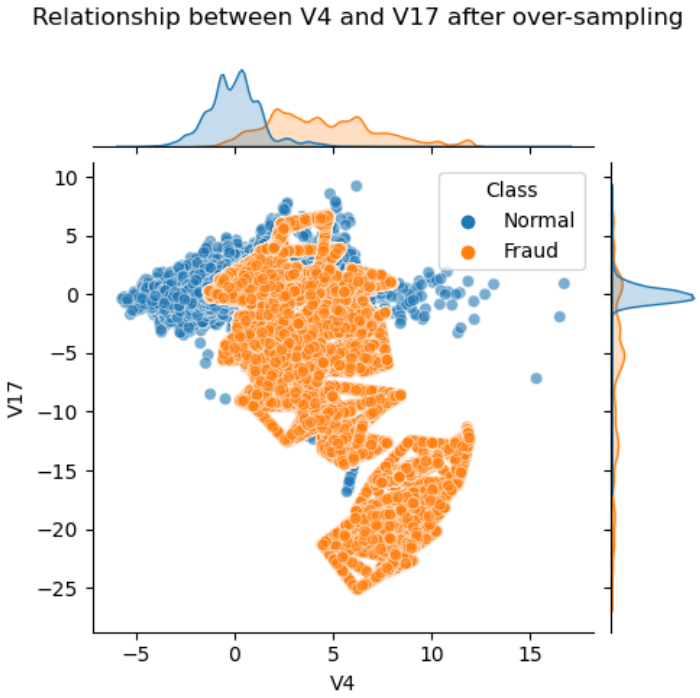
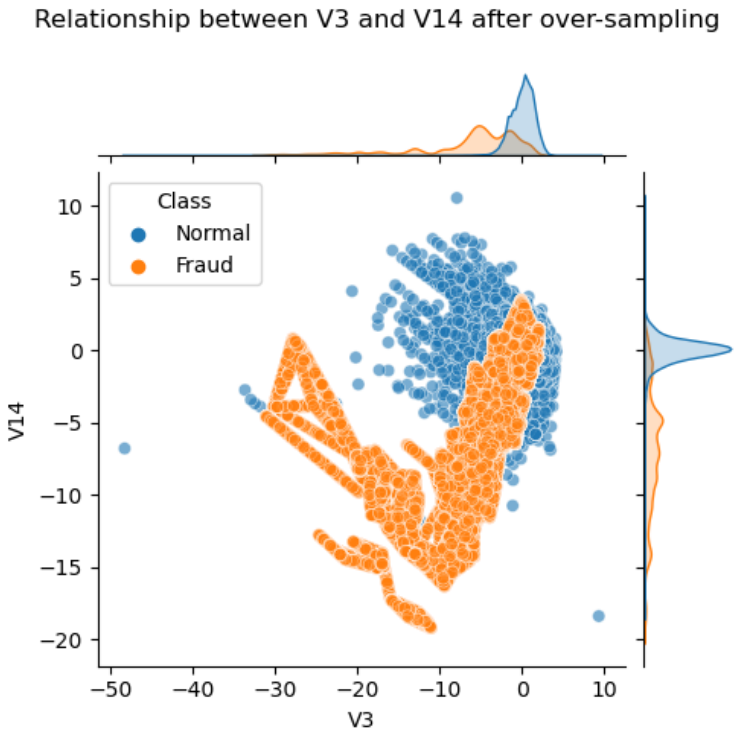


Figure 33

The Scatter Plot between Feature 'V3' and 'V14' from the Over-sampling Dataset



2.5 Model evaluation through cross-validation

Cross-validation is an essential tool in machine learning to evaluate the performance of a model on unseen data. In case to utilize the dataset efficiently, it splits the train set into several parts according to k-fold. It iteratively fits k-1 folds data into the model and 1-fold data into validation (Cross-validation: evaluating estimator performance, n.d.). Five-fold stratified cross-validations are applied in this project, and the stratified target is set to the 'Class' feature. This cross-validation keeps every fold the same ratio between normal and fraudulent transactions. In general, cross-validation could be used to detect model overfitting. The average and range of cross-validation scores from all five datasets are shown in Figure 34. The range shown in the figure is to evaluate whether the model is overfitting, and it seems inconsistent in five-fold cross-validation from all numerical features dataset to under-sampling dataset. The reason is the number of normal transactions is enormous but small for fraud transactions. This is an inherent flaw that cannot be eliminated at any point. This phenomenon can be seen in Figure 35, which changes the assessment metric to recall. However, the score corresponding to the over-sampling dataset seems the best. The reason is that the over-sampling technique introduces data bias, and reinforces the

pattern of the fraud transactions by making huge synthetic data points.

Figure 34

The Cross-validation Score Generated from Average Precision

	Avg. average precision	Range	dataset
0	0.8454	0.0720	All numerical features
1	0.8323	0.0851	Strong correlated features
2	0.7170	0.1111	Weighted-resampling
3	0.6946	0.1021	Under-resmapling
4	0.8240	0.0609	Over-resampling

Figure 35

The Cross-validation Score Generated from Recall

	Avg. recall	Range	dataset
0	0.7690	0.1392	All numerical features
1	0.7766	0.1139	Strong correlated features
2	0.8604	0.0633	Weighted-resampling
3	0.8959	0.0759	Under-resmapling
4	0.8274	0.0759	Over-resampling

2.6 Model construction

In this section, a fine-tuned supervised ML model for this particular project will be constructed through several stages. Several datasets are prepared through correlation analysis, data visualization, and feature selection. Due to the complexity of the datasets, cross-validation, hyperparameter tuning, and pipeline structure should be applied in this section. Many online ML articles suggested that, resampling should be finished before cross-validation, and the correct way to process that is by putting the resampling function and prepared model in the same pipeline (Martin, 2019). Hyperparameter tuning is the final step in this section. This whole procedure ensures the model is optimal after each step.

2.6.1 Model selection by Pycaret

The model selection is a time-consuming and tedious process since many supervised ML models are waiting to be chosen. However, a tool called Pycaret saves time and code for this process simultaneously, which is an open-source and low-code ML library that automates ML workflows. For the model selection phase, I used the dataset that dropped the 'Time' and 'Amount' features to fit every supervised model. The result, along with assessment metrics, is shown in Figure 36. The extra tree classifier stands out of several supervised ML models. Hence, it will be picked up as a final model used for further hyperparameter tuning and cross-validation phases. After model selection, a custom function for weighted resampling is shown in Figure 37. The code is modified from Figure 21 but changes the input variables from the Pandas data frame to the Numpy array since FunctionSampler from Imblearn has specific requirements.

Figure 36

The Model Selection Result through Pycaret

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	PRC_AUC
et	Extra Trees Classifier	0.9996	0.9513	0.7690	0.9645	0.8553	0.8551	0.8608	0.7426
knn	K Neighbors Classifier	0.9995	0.9211	0.7537	0.9302	0.8324	0.8322	0.8369	0.7027
rf	Random Forest Classifier	0.9995	0.9437	0.7614	0.9612	0.8495	0.8492	0.8551	0.7327
xgboost	Extreme Gradient Boosting	0.9995	0.9772	0.7715	0.9500	0.8511	0.8509	0.8557	0.7345
lda	Linear Discriminant Analysis	0.9994	0.8919	0.7537	0.8581	0.8007	0.8003	0.8030	0.6469
lr	Logistic Regression	0.9992	0.9724	0.6114	0.8739	0.7160	0.7156	0.7288	0.5347
ada	Ada Boost Classifier	0.9992	0.9654	0.6801	0.8087	0.7380	0.7376	0.7408	0.5529
dt	Decision Tree Classifier	0.9991	0.8691	0.7386	0.7291	0.7328	0.7323	0.7328	0.5392
svm	SVM - Linear Kernel	0.9990	0.0000	0.5352	0.8403	0.6479	0.6475	0.6669	0.4494
gbc	Gradient Boosting Classifier	0.9990	0.7013	0.5451	0.7898	0.6418	0.6413	0.6540	0.4399
ridge	Ridge Classifier	0.9988	0.0000	0.4009	0.8256	0.5372	0.5367	0.5733	0.3327
dummy	Dummy Classifier	0.9983	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0017
lightgbm	Light Gradient Boosting Machine	0.9942	0.6985	0.5355	0.2149	0.2954	0.2936	0.3279	0.1156
nb	Naive Bayes	0.9777	0.9581	0.8146	0.0603	0.1123	0.1094	0.2180	0.0496
qda	Quadratic Discriminant Analysis	0.9736	0.9702	0.8629	0.0545	0.1024	0.0995	0.2128	0.0474

2.6.2 Model building through the pipeline

After the preparation stage, a custom pipeline function should be built for resampling. The code is shown in Figure 38, and the classifier only sets the random state variable. The random state variable is set to one to ensure data consistency through data manipulation.

Figure 37

The Custom Weighted-sampling Function for Building the Pipeline

```
# this function is used for building a pipeline through cross validation
# fraud transactions account for 10% of total transactions
# df_corr will be fit into this function
def res_weighted_pl(input, target):

    # input and target should be numpy array that defined by FunctionSampler
    array_ = np.concatenate((input, target.reshape(-1, 1)), axis=1)

    # transfer numpy array into pandas dataframe with column name
    df_ = pd.DataFrame(
        array_, columns=['V3', 'V4', 'V7', 'V10', 'V14', 'V17', 'Class']
    )

    # split normal and fraud transactions
    normal = df_.query('Class==0')
    fraud = df_.query('Class==1')

    # concatenate resampled normal transactions with fraud transactions
    normal_res = normal.sample(n=(len(fraud) * 9), random_state=1)
    df_res = pd.concat([normal_res, fraud])

    # input target split
    X_res = df_res.drop('Class', axis=1)
    y_res = df_res.Class

    return X_res, y_res
```

Figure 38

Construction Pipeline that Combines All Resampling Methods and the Classifier

```
# making a function to create particular resampled pipeline
# model is the final classifier after hyperparameter tuning
def res_pl(method):

    # choose resampling method
    if method == 'weight':
        # custom sampler
        sampler = FunctionSampler(func=res_weighted_pl)
    elif method == 'under':
        sampler = RandomUnderSampler(random_state=1)
    elif method == 'over':
        sampler = SMOTE(random_state=1)
    else:
        raise ValueError("Method must be provided!")

    # default classifier for further hyperparameter tuning
    clf = ExtraTreesClassifier(random_state=1)
    # make a pipeline for resamplers and classifiers
    pipeline = Pipeline([('Resampler', sampler), ('Classifier', clf)])

    return pipeline
```

2.6.3 Hyperparameter tuning

A model with default parameters might not be the best option in many ML cases since the dataset has hidden patterns, so the model might be inefficient with default settings. So hyperparameter tuning is introduced into this

phase to construct a robust model. In this project, hyperparameter refers to the parameters defined within the extra tree classifier mentioned above. The grid search technique is adopted since it is a powerful tool from Scikit-learn to tuning the model. It could assist us in trying every possible combination of the provided parameters and searching for the optimal combination through the tuning process with the train set. However, hyperparameter tuning is time-consuming since the fundamental method is an exhaustive search from every combination of parameters (Kasture, 2020). All hyperparameter tuning code is shown in Figure 39 and 40. After hyperparameter tuning, two single classifiers and three resampling pipelines, which are shown in Figures 41 and 42, and 43, should be constructed for final testing and result. The figure does not show some optimal parameters since they are set to default values.

Figure 39

Parameters that Need to be Tuned

```
# define parameters for further tuning
# for single classifier
# default n_estimators=100, criterion=gini, max_features=sqrt
param = {
    'n_estimators': [25, 50, 100],
    'criterion': ['gini', 'entropy', 'log_loss'],
    'max_features': ['sqrt', 'log2', None]
}

# for pipeline
param_res = {
    'Classifier__n_estimators': [25, 50, 100],
    'Classifier__criterion': ['gini', 'entropy', 'log_loss'],
    'Classifier__max_features': ['sqrt', 'log2', None]
}
```

Figure 40

The Hyperparameter Tuning Code

```
# build hyperparameter tuning function by using grid search
# par should be a dict, input and target are train set and labeled target
# search.best_params_
def hyper_tune(input, target, par, pl_grid=None, score='average_precision'):

    fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=1)
    clf_grid = ExtraTreesClassifier(random_state=1)

    if pl_grid is None:
        search = GridSearchCV(clf_grid, par, cv=fold, scoring=score)
    else:
        search = GridSearchCV(pl_grid, par, cv=fold, scoring=score)

    # fit the grid search
    search.fit(input, target)

    return search.best_estimator_
```

Figure 41

The Final Model for All Numerical Features and Strongly Correlated Datasets after Hyperparameter Tuning

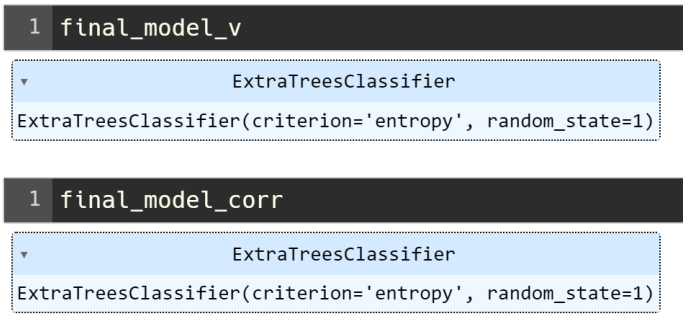


Figure 42

The Final Model for Weighted-sampling Dataset after Hyperparameter Tuning

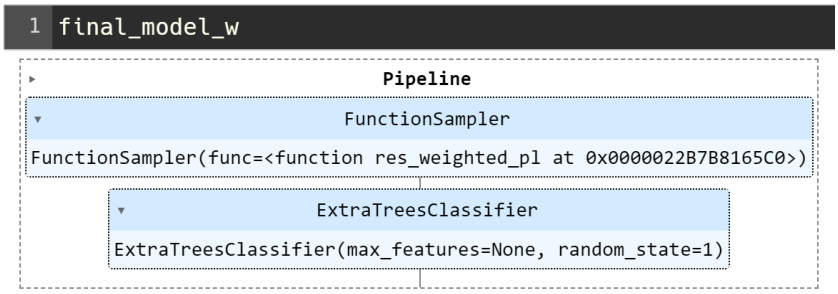
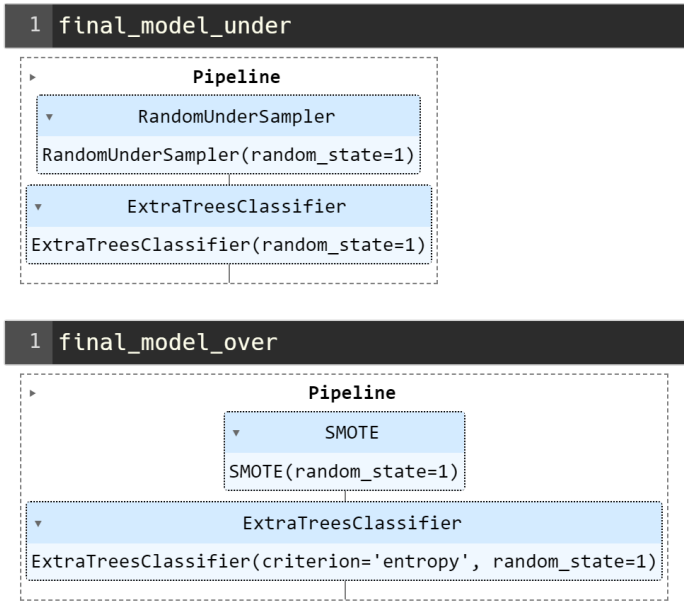


Figure 43

The Final Model for Under and Over Sampling datasets after Hyperparameter Tuning



3. Result

This section will present the final result from three perspectives: classification report, confusion matrix, and precision-recall curve. Each of them uses the same assessment metrics concept: recall and precision. These two-assessment metrics are pretty crucial when encountering the classification issue. The equations of them are defined in (1) and (2), respectively ("Precision and recall," 2023). Recall refers to how many true positives could be detected through the model. Precision refers to how many false alarms could be arisen. There is always a trade-off between recall and precision. These two metrics cannot be improved at the same time. If the recall score is relatively worse, the finance institutions will take a loss since the model does not raise the alarm efficiently when encountering genuine fraud transactions. However, suppose the precision score decreases, which means the false alarms will increase. In that case, the bank will occasionally double-check customer transactions since the model picks up normal transactions as fraud. The precision-recall curve will be shown afterward.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{1}$$

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{2}$$

3.1 Classification report

This section presents the five classification reports summarizing different models shown in Figures 44 to 48. The first comparison is between the single classifiers from all numerical and strongly correlated feature datasets. The strongly correlated feature dataset outperformed the all numerical feature dataset from both recall and precision. It is a big success because the train set size has shrunk, but the result is even better. The same effect could be seen in the confusion matrix part. Secondly, higher recall scores have been seen on weighted and under-sampling datasets, but a massive fallback in precision. The main reason is the sample size decreased tremendously from the original dataset. The total length of weighted and under-sampling datasets is 3940 and 788, and they only account for 1.73% and 0.35% of the strongly correlated train set, respectively. The over-sampling dataset results worse than all numerical features and strongly correlated features datasets since it reinforces the pattern for fraud transactions, so the outcome is distracted by enormous synthetic data points.

Figure 44

The Classification Report of All Numerical Features Dataset

```
1 print(report_v)
```

	precision	recall	f1-score	support
Normal	1.000	1.000	1.000	56864
Fraud	0.921	0.837	0.877	98
accuracy			1.000	56962
macro avg	0.961	0.918	0.938	56962
weighted avg	1.000	1.000	1.000	56962

Figure 45

The Classification Report of Strongly Correlated Features Dataset

```
1 print(report_corr)
```

	precision	recall	f1-score	support
Normal	1.000	1.000	1.000	56864
Fraud	0.943	0.847	0.892	98
accuracy			1.000	56962
macro avg	0.971	0.923	0.946	56962
weighted avg	1.000	1.000	1.000	56962

Figure 46

The Classification Report of Weighted-sampling Dataset

```
1 print(report_w)
```

	precision	recall	f1-score	support
Normal	1.000	0.998	0.999	56864
Fraud	0.390	0.908	0.546	98
accuracy			0.997	56962
macro avg	0.695	0.953	0.772	56962
weighted avg	0.999	0.997	0.998	56962

Figure 47

The Classification Report of Under-sampling Dataset

```
1 print(report_under)
```

	precision	recall	f1-score	support
Normal	1.000	0.972	0.986	56864
Fraud	0.054	0.918	0.102	98
accuracy			0.972	56962
macro avg	0.527	0.945	0.544	56962
weighted avg	0.998	0.972	0.984	56962

Figure 48

The Classification Report of Over-sampling Dataset

```
1 print(report_over)
```

```
              precision    recall  f1-score   support

      Normal       1.000     0.999     1.000     56864
       Fraud       0.737     0.857     0.792        98

    accuracy                          0.999     56962
   macro avg       0.868     0.928     0.896     56962
weighted avg       0.999     0.999     0.999     56962
```

3.2 Confusion matrix

The confusion matrix is usually applied as an essential model assessment tool for classification issues. It is a specific two rows and two columns table layout, usually with the genuine values sitting along the y-axis and predicted value sitting along the x-axis, or vice versa (“Confusion matrix,” 2023). The matrix could be used as a visualization tool to clarify precision and recall. There are four slots in the confusion matrix: true positive, true negative, false positive, and false negative. Positives and negatives mean fraud and normal transactions. By contrast, true and false mean whether the model predicted the right and wrong values, respectively. True positive and true negative describes the model that predicted the values as correctly as the genuine values. The false positive means the model failed to classify the normal transactions correctly but raised false alarms. The false negative means the model failed to detect actual fraud transactions. The main focus is on false positives and false negatives due to the severely imbalanced property of the dataset generating enormous true negatives. False positives and false negatives can also be called false alarms when describing the performance of the predicted model.

The first comparison is between datasets with all numerical and strongly correlated features. Figures 49 and 50 show that the strongly correlated features dataset outperforms all numerical features dataset. It received better recall and precision. It is a big success in this project since the overall dataset size is reduced by 78.6% based on statistical and correlation analysis. The second comparison is between down-sampling datasets, weighted and under-sampling datasets, and the all numerical features dataset shown in Figures 51, 52, and 49. The recall has improved, but the precision fell significantly since the overall sample size is considerably less than the all numerical features dataset. The last comparison is between all numerical features and over-sampling datasets shown in Figures 49 and 53. The recall and precision are both worse than the all numerical features dataset.

Figure 49

The Confusion Matrix Generated from All Numerical Features Dataset

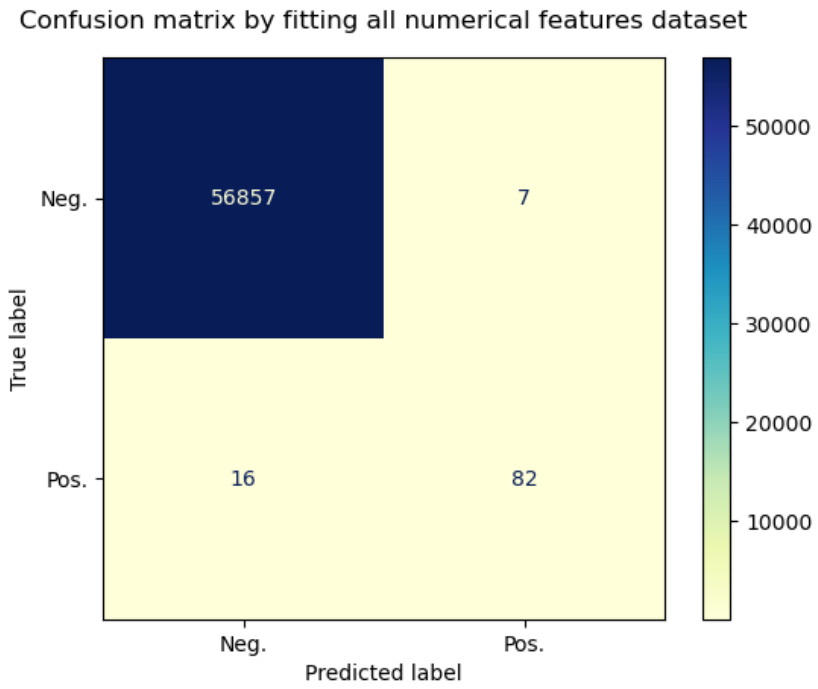


Figure 50

The Confusion Matrix Generated from Strongly Correlated Features Dataset

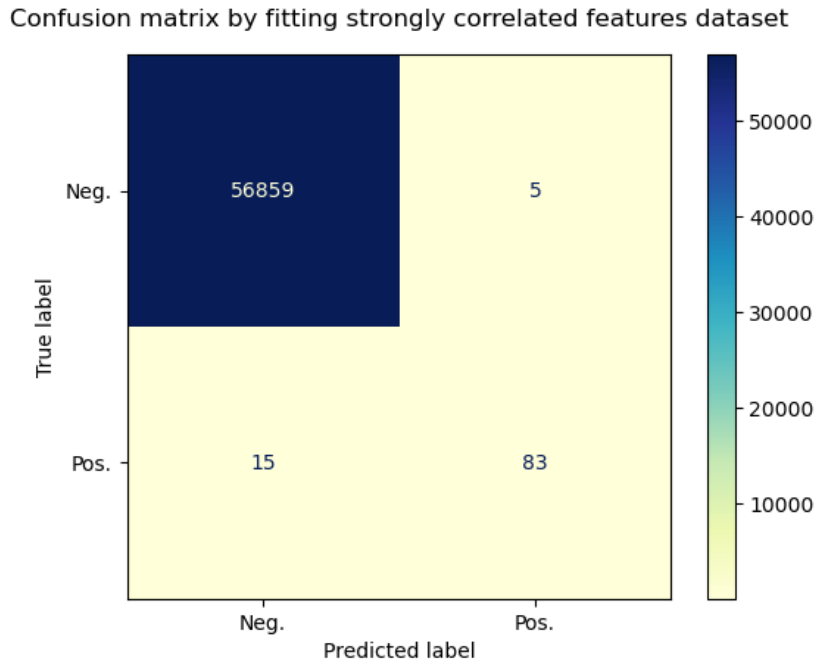


Figure 51

The Confusion Matrix Generated from Weighted-sampling Dataset

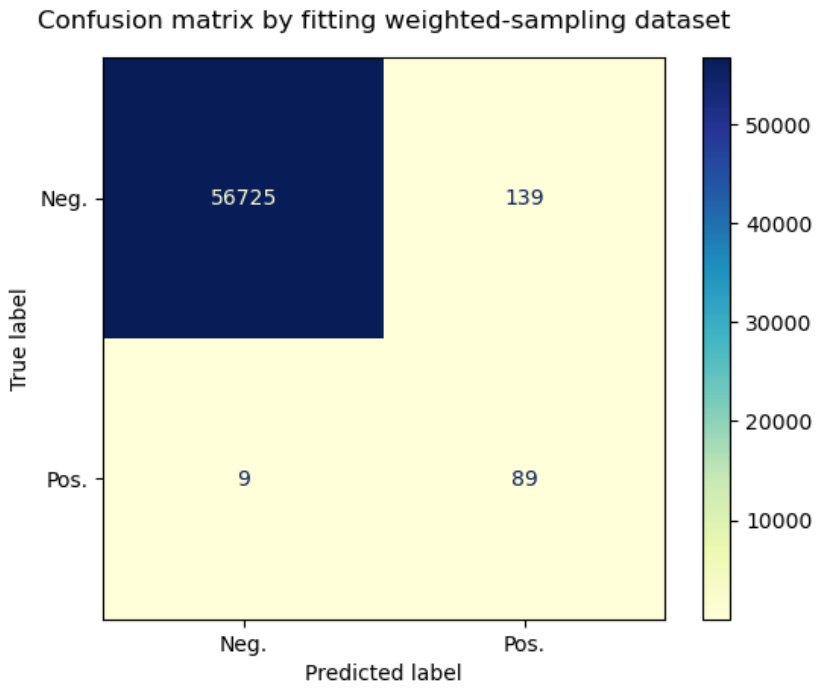


Figure 52

The Confusion Matrix Generated from Under-sampling Dataset

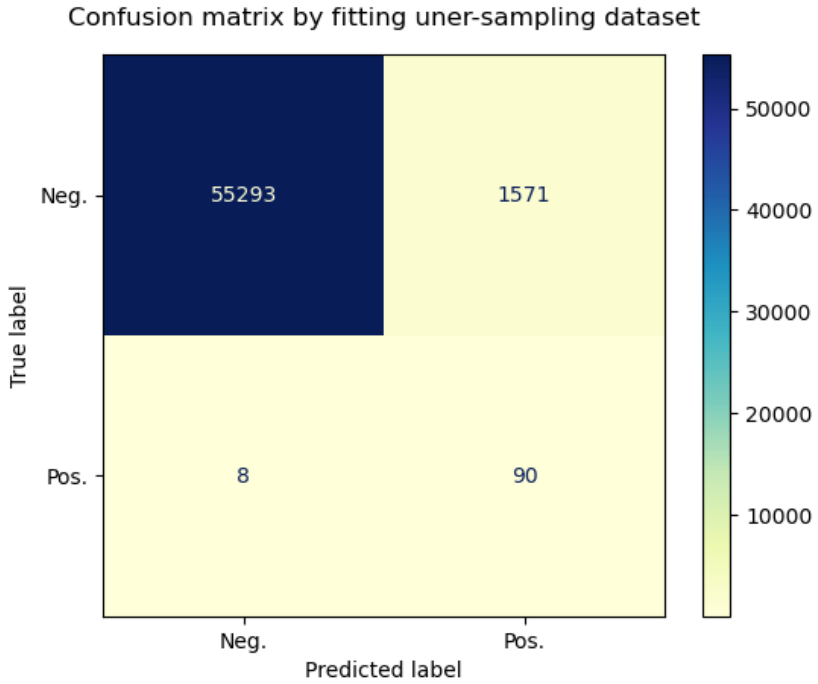
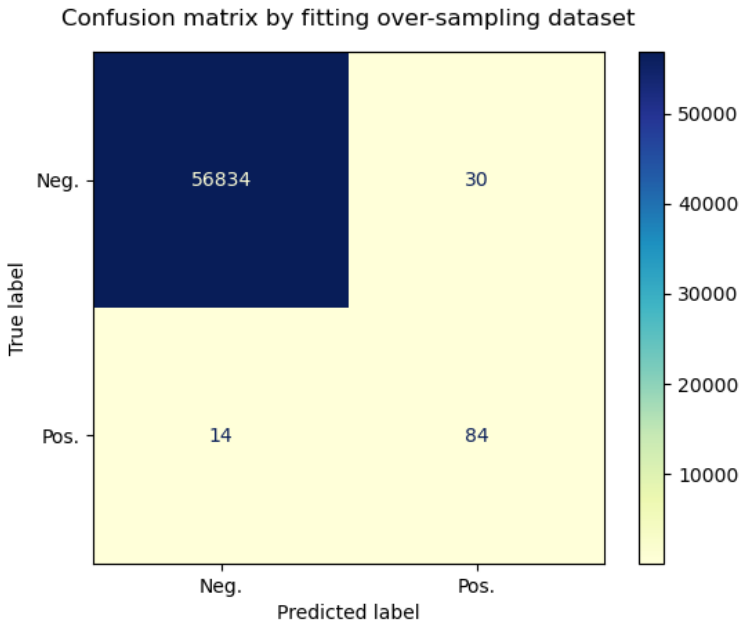


Figure 53

The Confusion Matrix Generated from Over-sampling Dataset



3.3 Precision recall curve

This section will analyze the precision-recall curve since it is a crucial metric when dealing with imbalanced datasets. The average precision and the optimal points corresponding to the pair of recall and precision with it are shown in the label of Figure 55 and Figure 56. A custom equation (3) is presented below to demonstrate what I want to address,

$$False\ positive = \frac{Recall \times (1 - Precision)}{Precision} \times All\ positives(TP + FN) \quad (3)$$

All positives are the total fraud records in the test set, which is a fixed number. Suppose the primary goal is to balance the recall and precision while getting a decent amount of false positives, which could benefit both customer satisfaction and the financial institutions. An optimal point on the precision-recall curve based on (3) should be the particular pair of precision and recall, which keeps recall at a relatively high score and simultaneously has the most significant product of recall and precision. The core algorithm to pick up the optimal point is shown in Figure 54. The optimal pair is slightly different from the final model's classification report since the final model's assessment scores are the average precision from the train set. Figure 57 and 58 show the confusion matrix generated from the final fine-tuned model that applies the optimal precision and recall pair.

The performance is slightly better on recall but worse on precision. This final result means the financial institutions could save 2% more on their fraud issue cost, but customer satisfaction will drop 2%.

Figure 54

A Simple Algorithm to Find the Optimal Pair of Recall and Precision

```
# Find the optimal pair of recall and precision
l = []
for p, r in zip(precision, recall):
    temp = r * p
    l.append(temp)

index_ = l.index(max(l))
r_ = recall[index_]
p_ = precision[index_]
```

Figure 55

The Precision-Recall Curve of All Numerical Features Dataset

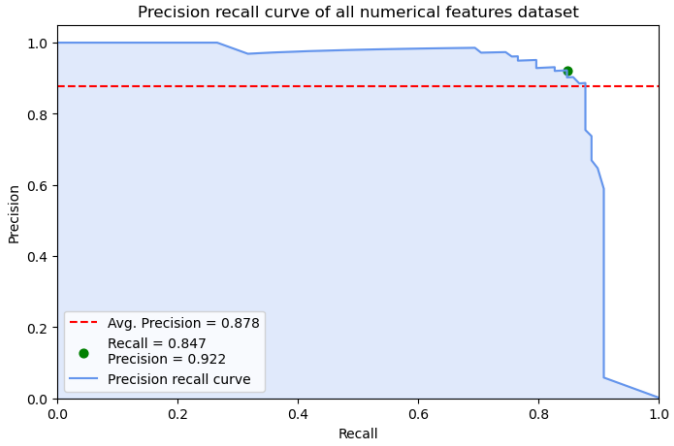


Figure 56

The Precision-Recall Curve of Strongly Correlated Features Dataset

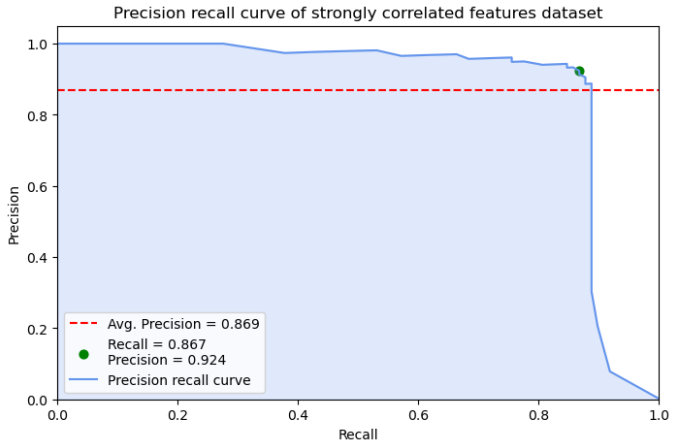


Figure 57

The Confusion Matrix of All Numerical Features Dataset by Applying the Optimal Pair of Precision and Recall

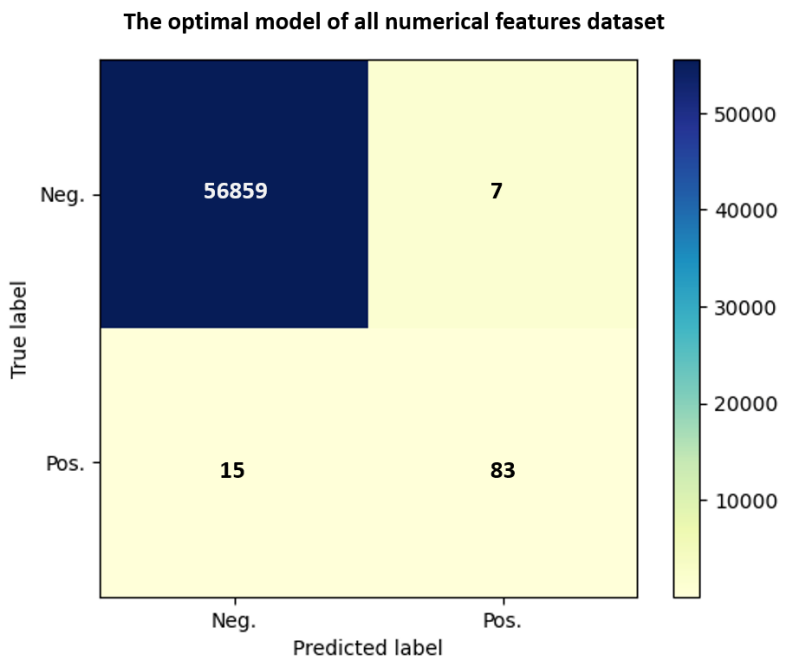
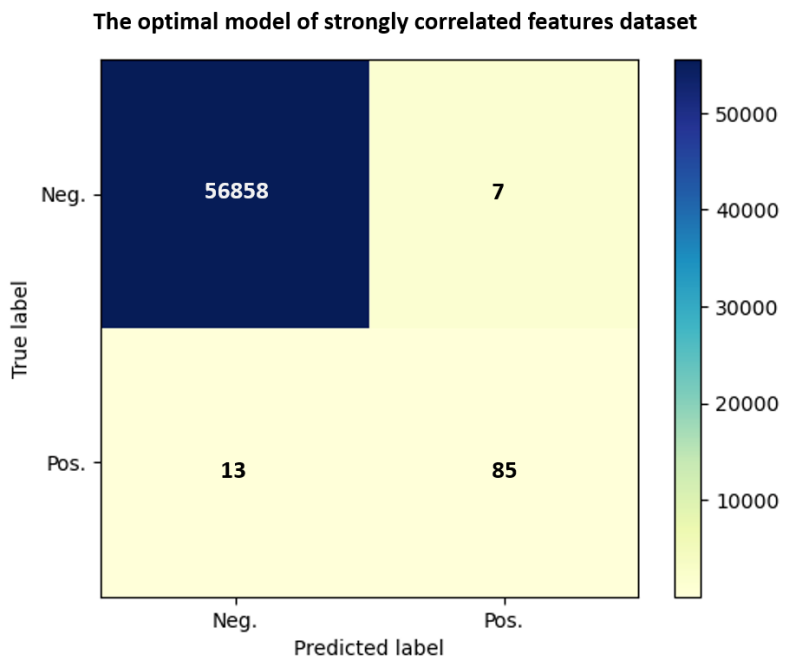


Figure 58

The Confusion Matrix of Strongly Correlated Dataset by Applying the Optimal Pair of Precision and Recall



4. Limitation and future work

4.1 Limitation

The whole project is built along with a particular relevant dataset with a few inherent flaws. On the other hand, the techniques used in this project could be less efficient when applied to different topics and datasets from several aspects. All limitations are listed below.

- **Dataset limitation**

  The dataset applied in this project regards financial issues and is processed through PCA to keep confidentiality. Furthermore, the dataset is severely imbalanced, meaning the fraud transactions account for only 0.2% of total records. Hence, these inherent flaws could affect the data manipulation process.

- **Problem definition**

  Several other researchers defined this issue as an anomaly detection problem, which could affect the selection of the core ML techniques switching from supervised classifiers to unsupervised classifiers.

- **Functions and models selection**

  All functions and models proposed in this paper might not be optimal due to the time duration and industry insight. Hence, the result could differ when sample quantity, feature quantity, and fraud-normal ratio are varied. Other functions and models could be applied to the same issue based on researchers understanding and their perspective.

4.2 Future work

This project expands my insight into the ML industry to a certain extent. The future work will develop from aspects as follows,

- Gaining an understanding of the fundamental mathematics of particular algorithms.
- Focusing on the neural network and autoencoder and solving this project as an anomaly detection issue.
- Trying more supervised and unsupervised ML projects on Kaggle to gain practical experience.

5. Conclusion

This project proposes a fine-tuned supervised extra tree classifier to tackle the CCF issue. The model was evaluated through five-fold stratified cross-validation, average precision score, and hyperparameters tuning using grid search. The recall and f1-score of the strongly correlated features model improved by 10.14% and 4.3% from

the training scores generated from the model selection phase of all numerical features datasets, respectively. However, the precision score of the strongly correlated features model fell by 2.2% from all numerical features model. By contrast, the pipeline models of weighted-sampling and under-sampling have better recall than the single classifier model but worse precision. The over-sampling pipeline model has slightly better recall but worse precision than all numerical features model. Due to the over-sampling model reinforcing the patterns of fraudulent transactions by making enormous synthetic data points, it passed the overfitting test through cross-validation. However, the other four models suffered from overfitting to a certain degree due to the severely imbalanced inherent flaw.

Overall, this project achieved two goals. The first goal is successfully conducting an efficient feature selection function with statistical and correlation analysis. This function dramatically reduces the input train set size but maintains the recall and precision performance. The second goal is to examine the disadvantage of the resampling techniques. Researchers should be cautious when applying resampling techniques to their projects since it could reinforce data bias.

Reference

Mullen, C. (2023, January 5). *Card industry's fraud-fighting efforts pay off: Nilson Report.* Payments Dive. https://www.paymentsdive.com/news/card-industry-fraud-fighting-efforts-pay-off-nilson-report-credit-debit/639675/

Lee, J. G., & Scott, G. G. (2017). *Preventing credit card fraud: a complete guide for everyone from merchants to consumers*. Rowman & Littlefield.

Barker, K. J., D'Amato, J., & Sheridon, P. (2008). Credit card fraud: awareness and prevention. *Journal of Financial Crime*, *15*(4), 398–410. https://doi.org/10.1108/13590790810907236

Veeraraghavan, S. (2023, July 30). *Top 20 Best Programming Languages to Learn in 2023*. Simplilearn. https://www.simplilearn.com/best-programming-languages-start-learning-today-article

Kultur, Y., & Calayan, M. U. (2017). Hybrid approaches for detecting credit card fraud. *Expert Systems*, *34*(2), e12191–n/a. https://doi.org/10.1111/exsy.12191

Sivanantham, S., Dhinagar, S.R., Kawin, P., & Amarnath, J. (2021). Hybrid Approach Using ML Techniques in Credit Card Fraud Detection. *Advances in Smart System Technologies, 1163*, 243–251. https://doi.org/10.1007/978-981-15-5029-4_19

Alam, M., Podder, P., Bharati, S., & Mondal, M.R.H. (2021). Effective ML Approaches for Credit Card Fraud Detection. *Innovations in Bio-Inspired Computing and Applications*, *1372*, 154-163. https://doi.org/10.1007/978-3-030-73603-3_14

Plakandaras, V., Gogas, P., Papadimitriou, T., & Tsamardinos, I. (2022). Credit Card Fraud Detection with Automated ML Systems. *Applied Artificial Intelligence*, *36*(1). https://doi.org/10.1080/08839514.2022.2086354

Carcillo, F., Le Borgne, Y.-A., Caelen, O., Kessaci, Y., Oblé, F., & Bontempi, G. (2021). Combining unsupervised and supervised learning in credit card fraud detection. *Information Sciences*, *557*, 317–331. https://doi.org/10.1016/j.ins.2019.05.042

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2018). Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy. *IEEE Transaction on Neural Networks and Learning Systems*, *29*(8), 3784–3797. https://doi.org/10.1109/TNNLS.2017.2736643

Keswani, B., Vijay, P., Nayak, N, Keswani, P., Dash, S., Sahoo, L., Mishra, T. C., & Mohapatra, A. G. (2020). Adapting ML Techniques for Credit Card Fraud Detection. *International Conference on Innovative Computing and Communications*, *1087*, 443–455. https://doi.org/10.1007/978-981-15-1286-5_38

Alfaiz, N. S., & Fati, S. M. (2022). Enhanced Credit Card Fraud Detection Model Using ML. *Electronics (Basel)*, *11*(4), 662–. https://doi.org/10.3390/electronics11040662

MACHINE LEARNING GROUP – ULB. (2021, March 5). *Credit Card Fraud Detection*. Kaggle. https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud

Dal Pozzolo, A., Caelen, O., Johnson, R. A., & Bontempi, G. (2015). Calibrating Probability with Undersampling for Unbalanced Classification. *2015 IEEE Symposium Series on Computational Intelligence*, 159-166. https://doi.org/10.1109/ssci.2015.33

Dal Pozzolo, A., Caelen, O., Le Borgne, Y.-A., Waterschoot, S., & Bontempi, G. (2014). Learned lessons in credit card fraud detection from a practitioner perspective. *Expert Systems with Applications*, *41*(10), 4915–4928. https://doi.org/10.1016/j.eswa.2014.02.026

Dal Pozzolo, A., Boracchi, G., Caelen, O., Alippi, C., & Bontempi, G. (2018). Credit Card Fraud Detection: A Realistic Modeling and a Novel Learning Strategy. (2018). *IEEE Transactions on Neural Networks and Learning Systems*, *29*(8), 3784–3797. https://doi.org/10.1109/tnnls.2017.2736643

Dal Pozzolo, A. (2015). *Adaptive Machine learning for credit card fraud detection* (Unpublished PhD's dissertation). Université Libre de Bruxelles, Brussels, Belgium.

Carcillo, F., Dal Pozzolo, A., Le Borgne, Y.-A., Caelen, O., Mazzer, Y., & Bontempi G. (2018). SCARFF: A Scalable Framework for Streaming Credit Card Fraud Detection with Spark. *Information Fusion*, *41,* 182–194. https://doi.org/10.1016/j.inffus.2017.09.005

Lebichot, B., Le Borgne, Y.-A., He-Guelton, L., Oblé, F., & Bontempi G. (2019). Deep-Learning Domain Adaptation Techniques for Credit Cards Fraud Detection. *Recent Advances in Big Data and Deep Learning*, *1*, 78–88, https://doi.org/10.1007/978-3-030-16841-4_8

Carcillo, F., Le Borgne, Y.-A., Caelen, O., Kessaci, Y., Oblé, F., & Bontempi, G. (2021). Combining unsupervised and supervised learning in credit card fraud detection. *Information Sciences*, *557*, 317–331. https://doi.org/10.1016/j.ins.2019.05.042

Le Borgne, Y.-A., & Bontempi, G. (2021). *Reproducible Machine Learning for Credit Card Fraud Detection -*

Practical Handbook.

Lebichot, B., Paldino, G.M., Siblini, W., He-Guelton, L., Oblé, F., & Bontempi, G. (2021) Incremental learning strategies for credit cards fraud detection. *International Journal of Data Science and Analytics*, *12*, 165–174. https://doi.org/10.1007/s41060-021-00258-0

Cross-validation: evaluating estimator performance. (n.d.). Scikit-learn. https://scikit-learn.org/stable/modules/cross_validation.html

Martin, D. (2019, May 20). *How to do cross-validation when upsampling data*. Stacked Turtles.https://kiwidamien.github.io/how-to-do-cross-validation-when-upsampling-data.html

Kasture, N. (2020, Nov 16). *Why Hyper parameter tuning is important for your model?* Medium. https://medium.com/analytics-vidhya/why-hyper-parameter-tuning-is-important-for-your-model-1ff4c8f145d3

Precision and recall. (2023, July 5). *In Wikipedia*. https://en.wikipedia.org/w/index.php?title=Precision_and_recall&oldid=1163655774

Confusion matrix. (2023, April 7). *In Wikipedia*. https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=1148699071