Identifying Fraudulent Transactions using Machine Learning: A Study of Credit Card Fraud

MOON KARMAKAR

ID- 40389123

Word Count- 2742

Technical Report submitted in part
fulfilment of the degree of Master of Science in
Business Analytics

Year of Submission: 2023

Queen's Management School

DECLARATION

This is to certify that:

i. The portfolio comprises only my original work;

ii. AI technologies (e.g. chat GTP) have not been used in the writing of the portfolio dissertation.

iii. No portion of the work referred to in the dissertation has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning.

Moon Karmakar.

Moon Karmakar

-- ------------------------------

[Candidate's Signature] Printed Name

07 September 2023

Date

Contents

Table of Figures

1. Introduction

Within the domain of contemporary financial systems, the identification and prevention of fraudulent transactions are of the utmost importance. The ability to detect credit card fraud reliably has significant implications for financial institutions and cardholders. This configuration manual serves as a comprehensive guide, outlining the system configuration, software dependencies, essential tools, and environmental prerequisites required to conduct a complex research project titled **"Identifying Fraudulent Transactions Using Machine Learning: A Study of Credit Card Fraud."**

The technical book provides a clear and complete reference for recreating project technical features. It creates the research environment, enabling others to examine, verify, or build on the project's results.The project's primary objective was to create a credit card fraud-detecting ensemble classifier. A well-planned and thorough process was used to accomplish this. We shall explain this technique, its technical components, and how we achieved the project's goals in the following parts.

2. System Setup

All the research and assignments are carried out in a device named Lenovo Yoga 7 series.

Device name: LAPTOP-O6TOTQDK

Processor: Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz 1.19 GHz

Installed RAM: 8.00 GB (7.79 GB usable)

Device ID: 5DAB1CA8-BF9D-41DA-9585-CAE4C611C7AE

Product ID: 00327-35887-60238-AAOEM

System type: 64-bit operating system, x64-based processor

Pen and touch: No pen or touch input is available for this display

3. Software Used

Python Programming Language: Version 3.11.4

Integrated Development Environment (IDE): Jupyter Notebook Version 6.0.1

Anaconda Platform Version: 2023.07.2

Additional Tools: Microsoft Excel, Lucid, Microsoft Word, Google Sheets

Web Browser: Google Chrome

4. Packages Installed

```python
# Install and import essential libraries
# Data processing
import pandas as pd
import numpy as np
import os
import random

# Mathematical and statistical functions
import math
import scipy.stats as stats
from scipy.stats import pointbiserialr, chi2_contingency, randint as sp_randint

# Data analysis and visualization
import researchpy as rp
import ppscore as pps
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns

# Machine Learning with scikit-learn
import sklearn

# Feature selection and data splitting
from sklearn.feature_selection import f_classif, SelectKBest
from sklearn.model_selection import train_test_split

# Categorical encoding
import category_encoders as ce

# Class imbalance handling
import imblearn
from imblearn.pipeline import Pipeline
from imblearn.under_sampling import RandomUnderSampler
from imblearn.over_sampling import SMOTE, BorderlineSMOTE
from collections import Counter

# Classification algorithms
from sklearn.ensemble import RandomForestClassifier, BaggingClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

# Model evaluation
from sklearn import metrics
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import auc
from imblearn.metrics import geometric_mean_score
from sklearn.metrics import make_scorer

# Hyperparameter tuning
from sklearn.model_selection import RandomizedSearchCV

# Cross-validation
from sklearn.model_selection import cross_val_score, cross_validate, StratifiedKFold

# Datetime library
import datetime
```

Figure 1 All the packages that are installed and imported

These Python packages and libraries serve various essential purposes in data analysis and machine learning. Pandas and NumPy facilitate data manipulation and numerical operations, while the os module interacts with the operating system for file management. Random and Math provide randomization and mathematical functions. Scipy.stats handles statistical functions, and Researchpy simplifies statistical analyses. Ppscore calculates predictive power between variables. Matplotlib and Seaborn aid in data visualization. Scikit-learn offers machine learning tools. Feature selection and data splitting are supported by SelectKBest and train_test_split. Category Encoders encode categorical data. Imbalanced-learn tackles class imbalance. Various classification algorithms like Random Forest, XGBoost, LGBM, Adaboost and CatBoost are implemented. Model evaluation metrics, imbalanced learning metrics, confusion matrix visualization, and precision-recall metrics are used for model assessment. The geometric mean score, AUC, and custom scorers enhance evaluation. Hyperparameter tuning leverages RandomizedSearchCV, and cross-validation is conducted using StratifiedKFold. These libraries collectively empower data analysis and model development in Python.

5. Differences from Existing Literature:

In the realm of credit card fraud detection, machine learning has emerged as a vital ally in the ongoing battle against fraudulent activities. This literature review underscores the critical role of key evaluation metrics, including recall, precision, F1-Score, MCC, G-Mean, and AUC-PR, in assessing the performance of machine learning models dedicated to this task (Dietterich, 1998). High recall, as demonstrated by models such as AdaBoost, XGBoost, LightGBM, and Bagging, is indispensable in ensuring the identification of genuine fraudulent transactions (Raschka, 2018). Precision, exemplified by Random Forest and Bagging, is equally pivotal as it minimizes the unsettling occurrence of false positives, which can inconvenience customers. The F1-Score, a harmonious blend of precision and recall, is instrumental in dealing with the imbalanced datasets characteristic of credit card fraud detection (Fernández-Delgado et al., 2014). Moreover, metrics like MCC and G-Mean provide comprehensive insights into overall model performance, with models like XGBoost, LightGBM, and Bagging consistently exhibiting robustness in binary classification tasks (Svetnik et al., 2004). AUC-PR sheds light on a model's ability to make well-informed decisions by optimizing the precision-recall trade-off, a domain where models like XGBoost and LightGBM excel (Chen et al., 2016; Ke et al., 2017). In conclusion, these metrics collectively offer a holistic view of machine learning model performance in credit card fraud detection, with models like XGBoost, LightGBM, and Bagging showing significant promise for further enhancing the efficacy of fraud detection systems in the future.

6. Detailed Data Processing Steps:

6.1 Data Preprocessing

6.1.1 Examine the Pandas DataFrame Structure.

- The first stage in data preprocessing is to import the data into a Pandas DataFrame to determine its structure. This typically entails verifying the number of rows and columns, the data type, and a cursory examination of the first few rows to determine how the data is formatted.

```
In [4]: fraud_df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 1852394 entries, 0 to 555718
Data columns (total 23 columns):
 #   Column                 Dtype  
---  ------                 -----  
 0   Unnamed: 0             int64  
 1   trans_date_trans_time  object 
 2   cc_num                 int64  
 3   merchant               object 
 4   category               object 
 5   amt                    float64
 6   first                  object 
 7   last                   object 
 8   gender                 object 
 9   street                 object 
 10  city                   object 
 11  state                  object 
 12  zip                    int64  
 13  lat                    float64
 14  long                   float64
 15  city_pop               int64  
 16  job                    object 
 17  dob                    object 
 18  trans_num              object 
 19  unix_time              int64  
 20  merch_lat              float64
 21  merch_long             float64
 22  is_fraud               int64  
dtypes: float64(5), int64(6), object(12)
memory usage: 339.2+ MB
```

Figure 2: Data Structure

6.1.2 Change Datatypes

- Converted the 'is_fraud' column to the category data type as it represents a binary classification label.
- Changed the data types of the 'gender' and 'category' columns to category as these are categorical variables.
- Converted the 'dob' and 'trans-date-trans-time' columns to datetime data types to facilitate date-based calculations.
- Converted 'cc-num' and 'zip' to strings as they are nominal variables represented by numbers, not integers.

```
In [5]: #Alter the data type of the binary target variable 'is_fraud' to 'category'.
        fraud_df["is_fraud"] = fraud_df["is_fraud"].astype('category')
        fraud_df["is_fraud"].dtypes

Out[5]: CategoricalDtype(categories=[0, 1], ordered=False)

In [6]: # Change the data type of 'gender' and 'category' variables to 'category'.
        fraud_df["gender"] = fraud_df["gender"].astype('category')
        fraud_df["category"] = fraud_df["category"].astype('category')

In [7]: # Update the data type of 'dob' and 'trans_day_trans_time' variables to datetime.
        fraud_df['dob'] = pd.to_datetime(fraud_df['dob'])
        fraud_df['trans_date_trans_time'] = pd.to_datetime(fraud_df['trans_date_trans_time'])

In [8]: #Transform the 'cc_num' and 'zip' variables into strings
        #as they are not integers but rather nominal variables with numeric representations.
        fraud_df["zip"] = fraud_df["zip"].astype('str')
        fraud_df["cc_num"] = fraud_df["cc_num"].astype('str')
```

Figure 3:Data modification

6.1.3 Remove Irrelevant Columns

- Removed the 'Unnamed: 0' column, which is deemed irrelevant for analysis.
- Also, considered removing variables with almost unique values to reduce dimensionality and noise.

```
In [9]: # remove irrelevant column
        fraud_df = fraud_df.drop(columns="Unnamed: 0")
        fraud_df = fraud_df.drop(columns="unix_time")
        fraud_df = fraud_df.drop(columns="merch_lat")
        fraud_df = fraud_df.drop(columns="merch_long")
```

Figure 4: Data Cleaning

6.1.4 Check for Missing Values and Duplicates

- Performed data quality checks to identify and handle any missing values. This ensures that your data is complete.
- Checked for and remove any duplicate rows to prevent data redundancy.

```
In [10]: fraud_df.isnull().sum().sum()

Out[10]: 0

In [11]: fraud_df.duplicated().sum()

Out[11]: 0
```

Figure 5: Handling Missing Values

6.1.5 Check Cardinality for Categorical Variables

- Calculated the number of unique values (cardinality) for each categorical variable. High cardinality can impact modeling, and you may need to consider strategies like one-hot encoding or feature engineering.

```
In [12]: fraud_df.nunique()

Out[12]: trans_date_trans_time    1819551
         cc_num                       999
         merchant                     693
         category                      14
         amt                        60616
         first                        355
         last                         486
         gender                         2
         street                       999
         city                         906
         state                         51
         zip                          985
         lat                          983
         long                         983
         city_pop                     891
         job                          497
         dob                          984
         trans_num                1852394
         is_fraud                       2
         dtype: int64
```

Figure 6: Nunique values

6.2 Exploratory Data Analysis (EDA)

6.2.1 EDA for 'is_fraud'

- Visualized the class distribution of the target variable 'is_fraud' using a bar plot. Note the class imbalance in the dataset.

```
In [13]: # Instances of legitimate and fraudulent transactions.
         occurences = fraud_df['is_fraud'].value_counts()
         occurences
         #The following code retrieves the counts of legitimate ('0') and fraudulent ('1') transactions:
         # '0' denotes legitimate transactions.
         #'1' denotes fraudulent transactions.

Out[13]: 0    1842743
         1       9651
         Name: is_fraud, dtype: int64
```

Figure 7: Number of Occurences

6.2.2 Bar Plots for Categorical Variables

- Created bar plots for other categorical variables like 'category' to visualize their distributions. This helps understand the categorical data's characteristics.

```
In [16]: # Plot the number of legitimate and fraudulent transactions by category
         colors = ['blue', 'red']
         plt.figure(figsize=(20,8))
         plt.title('Volume of Legitimate and Fraudulent transactions by Category')
         sns.barplot(x="category", y='trans_num', hue="is_fraud", palette=colors,
                     data=fraud_df.groupby(['category', 'is_fraud']).agg({'trans_num' : 'count'}).reset_index())
```

```
Out[16]: <AxesSubplot:title={'center':'Volume of Legitimate and Fraudulent transactions by Category'}, xlabel='category', ylabel='trans_num'>
```

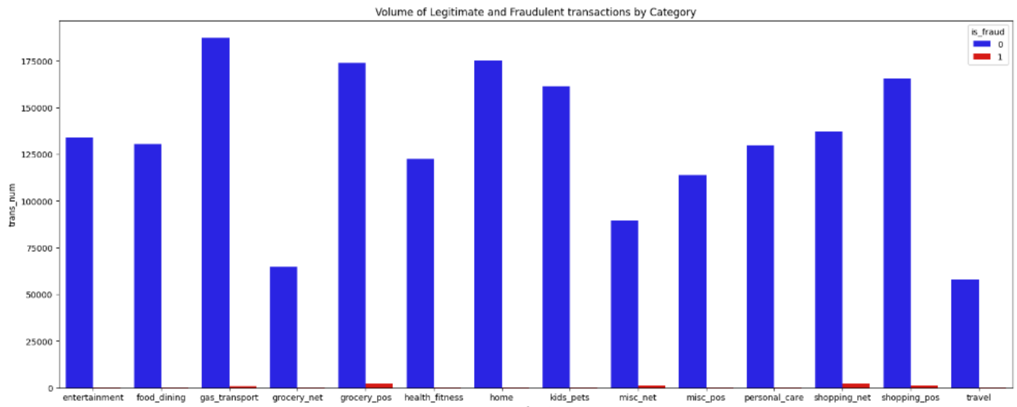

Figure 8: Quantity of Fraud and Non-fraud Transactions Based on Category

6.2.3 Descriptive Statistics and Histograms for Continuous Variables

- Calculated descriptive statistics (mean, median, etc.) for continuous variables like 'amount' and create histograms to understand their distributions.
- Analyzed the differences in these statistics between legitimate and fraudulent transactions.

```
In [14]: #visual representation of the data using histograms
         fraud_df.hist(figsize = (15, 15))
         plt.show()
```

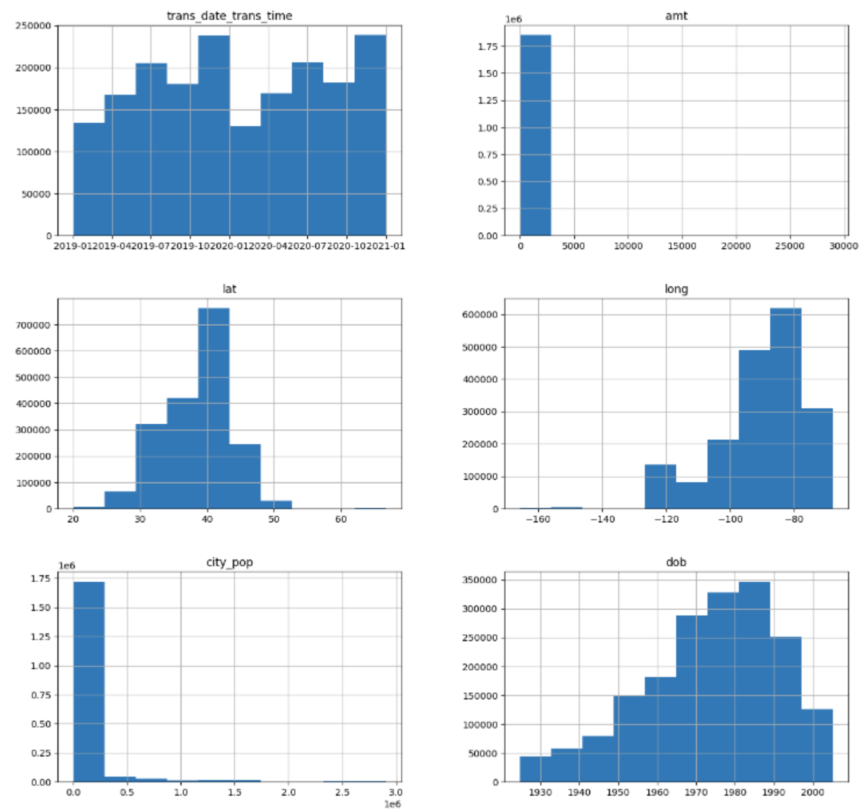

Figure 9: Descriptive Statistics and Histograms were Generated for Continuous Variables

6.3 Feature Engineering

6.3.1 Create the Variable 'age' at the Time of the Transaction

- Calculated the 'age' of individuals at the time of each transaction by subtracting their date of birth ('dob') from the transaction date ('trans-date-trans-time').

```
18]: import numpy as np

     # Create the variable 'age' at the time of the transaction
     fraud_df['age'] = np.round((fraud_df['trans_date_trans_time'] - fraud_df['dob']) / np.timedelta64(1, 'Y'))

     # Convert 'age' to integer
     fraud_df['age'] = fraud_df['age'].astype('int')

     # Display the data type of the 'age' column
     fraud_df.dtypes[['age']]

18]: age     int32
     dtype: object
```

Figure 10: Age created

6.3.2 Create the Variable 'transaction-hour'

- Extracted the transaction hour from the 'trans-date-trans-time' variable to create a new feature, 'transaction-hour.'
- Explored the distribution of fraudulent transactions by hour.

```
In [19]: # Create Variable transaction hour
         # Derive the feature 'transaction hour. Transaction hour for all transactions
         #Extract the hour of the transaction from the variable 'trans_day_trans_time'
         fraud_df[ 'transaction_hour'] = fraud_df['trans_date_trans_time'].dt.hour
         fraud_df['transaction_hour']

Out[19]: 0         0
         1         0
         2         0
         3         0
         4         0
                  ..
         555714    23
         555715    23
         555716    23
         555717    23
         555718    23
         Name: transaction_hour, Length: 1852394, dtype: int64
```

Figure 11: Transaction Hour Created

6.3.3 Create the Variable 'day-of-week'

- Extracted the 'day of the week' from 'trans-date-trans-time' to create 'day-of-week.'

```
In [23]: # Generate the 'day-of-week' variable
         # Extract the day of the week for each transaction
         fraud_df['day_of_week'] =fraud_df['trans_date_trans_time'].dt.day_name()
         fraud_df['day_of_week']

Out[23]: 0          Tuesday
         1          Tuesday
         2          Tuesday
         3          Tuesday
         4          Tuesday
                     ...
         555714    Thursday
         555715    Thursday
         555716    Thursday
         555717    Thursday
         555718    Thursday
         Name: day_of_week, Length: 1852394, dtype: object
```

Figure 12: 'Day of week' created

6.3.4 Create the Variable 'month of transaction'

- Extracted the month from 'trans-date-trans-time' to create 'month of transaction.'

```
In [24]: #Create the variable 'month of transaction'
         #Extract the year_month for all transactions
         fraud_df['year_month']=fraud_df['trans_date_trans_time'].dt.to_period('M')
         fraud_df['year_month']

         #Extract the Month of transaction
         fraud_df['month_of_trans']=fraud_df['year_month'].dt.month
         fraud_df['month_of_trans']

Out[24]: 0          1
         1          1
         2          1
         3          1
         4          1
                   ..
         555714    12
         555715    12
         555716    12
         555717    12
         555718    12
         Name: month_of_trans, Length: 1852394, dtype: int64
```

Figure 13: 'month of transaction' Created

6.3.5 Create the Variable 'time since last transaction'

- Calculated the time elapsed in seconds since the last transaction using 'trans-date-trans-time.'

```
[25]: # Generate the 'time since last transaction' variable
      # Time since last transaction = 'time_since_last_trans' and is measured in 'seconds'
      # I've developed a new function named 'timeDifference' to calculate the time elapsed
      #since the cardholder's previous credit card transaction.
      def timeDifference(x):
        x['time_since_last_trans']= x.trans_date_trans_time-x.trans_date_trans_time.shift()
        return x

[26]: #cc-num identifies a card holder
      fraud_df = fraud_df.groupby('cc_num').apply(timeDifference)

[27]: fraud_df['time_since_last_trans']= fraud_df['time_since_last_trans'].dt.seconds

[28]: # Examine null values for this newly created feature.
      # Given that it calculates the time since the last transaction,
      # it's expected to have some null values, particularly for customers making their first transaction!

[29]: fraud_df['time_since_last_trans'].isnull().sum().sum()

[29]: 999

[30]: # Replace the null values by 0. It means'0' seconds from last transaction

[31]: fraud_df['time_since_last_trans']= fraud_df['time_since_last_trans'].replace(np.nan, 0)

[32]: fraud_df['time_since_last_trans'].isnull().sum().sum()

[32]: 0
```

Figure 14: 'time since last transaction' Created

6.3.6 Generate Frequencies of Transactions in the Last n Days

- Computed the frequency of transactions made over specific time periods (e.g., last 1, 7, 14, 30, 60 days). This can help capture patterns related to transaction frequency.

```
[34]: import pandas as pd
#Volume of Transactions made in a day
def last1DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_1_days').sort_index()
    count_1_days = temp.rolling(window=2, min_periods=0).count()
    x['last_1_days_trans_count'] = count_1_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last1DaysTransCount)
```

```
[35]: #Volume of Transactions made in the last 7 days
def last7DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_7_days').sort_index()
    count_7_days = temp.rolling(window=7,min_periods=1).count()
    x['last_7_days_trans_count'] = count_7_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last7DaysTransCount)
```

```
[36]: #Volume of Transactions made in the last 14 days
def last14DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_14_days').sort_index()
    count_14_days = temp.rolling(window=14,min_periods=1).count()
    x['last_14_days_trans_count'] = count_14_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last14DaysTransCount)
```

```
[37]: #Volume of Transactions made in the last 30 days
def last30DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_30_days').sort_index()
    count_30_days = temp.rolling(window=30,min_periods=1).count()
    x['last_30_days_trans_count'] = count_30_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last30DaysTransCount)
```

```
[38]: #Volume of Transactions made in the last 60 days
def last60DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_60_days').sort_index()
    count_60_days = temp.rolling(window=60,min_periods=1).count()
    x['last_60_days_trans_count'] = count_60_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last60DaysTransCount)
```

Figure 15: Transaction Frequency

6.4 Feature Selection

6.4.1 Feature Selection for Continuous Variables

- Computed a correlation matrix to understand the relationship between continuous predictors and the target variable.
- Used the SelectKBest method with ANOVA F-values to select the most important continuous features.

```
[45]: #Creating new dataset
fraud_df2 = fraud_df[['amt', 'age', 'transaction_hour',
                      'hourEncoded', 'time_since_last_trans',
                      'last_7_days_trans_count', 'last_14_days_trans_count',
                      'last_30_days_trans_count', 'last_60_days_trans_count', 'is_fraud']]
```

```
[46]: # Separate the target variable 'is_fraud' from the dataframe
X = fraud_df2.loc[:, fraud_df2.columns != 'is_fraud']
y = fraud_df2['is_fraud']
```

```
[47]: #Conduct ANOVA F-test for feature selection

fs = SelectKBest(score_func=f_classif, k=9)
```

```
[48]: #The fit_transform method in feature selection is used to both fit the feature selection algorithm to
#the training data (X and y) and transform the data to keep only the selected features.
fit = fs.fit_transform(X, y)
```

Figure 16: ANOVA F Test

6.4.2 Feature Selection for Categorical Variables

- Performed a Chi-Square test to assess the association between categorical features and the target variable ('is_fraud').
- Computed Cramer's V to quantify the strength of the association.
- Applied Bonferroni Correction Post-Hoc Test to select only those categorical predictors where all their categories have a significant relationship with 'is_fraud'.

```
[51]: #Test the association between the categorical Independent variables and the target variable 'is_fraud'

from scipy.stats import chi2_contingency

# Define the list of categorical column names
categorical_columns = ['category', 'street', 'zip', 'city', 'state',
                       'first', 'last', 'cc_num', 'job', 'merchant', 'day_of_week', 'month_of_trans']

chi2_check = []

for i in categorical_columns:
    if chi2_contingency(pd.crosstab(fraud_df['is_fraud'], fraud_df[i]))[1] < 0.05:
        chi2_check.append('Reject Null Hypothesis')
    else:
        chi2_check.append('Fail to Reject Null Hypothesis')

res = pd.DataFrame(data=[categorical_columns, chi2_check]).T
res.columns = ['Column', 'Hypothesis']

print(res)

# The Null Hypothesis states that there is no association between the categorical predictor and the target
#variable 'is_fraud'
```

```
            Column              Hypothesis
0         category  Reject Null Hypothesis
1           street  Reject Null Hypothesis
2              zip  Reject Null Hypothesis
3             city  Reject Null Hypothesis
4            state  Reject Null Hypothesis
5            first  Reject Null Hypothesis
6             last  Reject Null Hypothesis
7           cc_num  Reject Null Hypothesis
8              job  Reject Null Hypothesis
9         merchant  Reject Null Hypothesis
10     day_of_week  Reject Null Hypothesis
11  month_of_trans  Reject Null Hypothesis
```

Figure 17: Bonferroni Correction Post Hoc Test

```
[52]: import researchpy as rp

crosstab, test_results_category, expected = rp.crosstab(fraud_df["is_fraud"], fraud_df["category"],
                                                        test="chi-square",
                                                        expected_freqs=True,
                                                        prop="cell")

test_results_category
```

[52]:

	Chi-square test	results
0	Pearson Chi-square (13.0) =	8329.1399
1	p-value =	0.0000
2	Cramer's V =	0.0671

Figure 18: Chi-Square Test

6.5 Data Preparation for Modeling

6.5.1 Split the Dataset into Train and Test Set

- The dataset is split into a training set (70%) and a test set (30%), using stratified splitting to maintain class proportions due to class imbalance.

```
[58]: # Split the Dataset into Train and Test Set

[59]: X = fraud_data.drop(columns=['is_fraud'])  # Exclude the target variable
      y = fraud_data['is_fraud']

[60]: #Split the dataset into train and test sets
      X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1, stratify=y)


      #Using a fixed "random state" value of 1 ensures consistent outcomes across multiple runs or executions.
      # 70% of the data will be used to train the models
      #30% of the data will be used to test the models

[61]: training_set_size = len(X_train)
      test_set_size = len(X_test)
      print(training_set_size)
      print(test_set_size)

      1296675
      555719
```

Figure 19: Train Test Dataset Size

6.5.2 Categorical Encoding

- Encoded categorical variables ('category' and 'day-of-week') using CatBoost encoding separately for the train and test sets.

```
[62]: # Categorical Encoding
```

```
[63]: # Using CATBOOST ENCODING
#define cateboost encoder
cbe_encoder= ce.cat_boost.CatBoostEncoder()
```

```
[64]: feature_list=['category', 'day_of_week'] # the categorical variables i want to encode
```

```
[65]: from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
# Fit the encoder and transform the feature
# Fit the encoder and transform the feature
cbe_encoder = ColumnTransformer([
    ('categorical', OneHotEncoder(handle_unknown='ignore'), feature_list)
])
```

```
[66]: train_cbe= cbe_encoder.fit_transform(X_train[feature_list],y_train)
test_cbe= cbe_encoder.transform(X_test[feature_list])
```

Figure 20: 'category' and 'day-of-week' Created

6.5.3 Dealing with Class Imbalance: Balance the Train Set

- Addressed class imbalance by balancing the training data using a hybrid sampling approach of Random Undersampling (RUS) and Borderline-SMOTE. Adjust sampling parameters to achieve the desired class proportions.

```
70]: # Adjust dataset sampling
# Modify the dataset accordingly
# Apply fitting and transformation to the dataset
X_train_sampled, y_train_sampled = pipeline.fit_resample(X_train, y_train)
```

```
54]: counter=Counter(y_train_sampled)
print(counter)

Counter({0: 1289919, 1: 6756})
```

Figure 21: Applying RUS and Borderline SMOTE

7. Areas for Improvement: (Models)

In the context of Precision-Recall curves, "plot smoothness" refers to the degree of continuity and gradual change in the curves when they are graphed. A smooth curve signifies a continuous transition of precision and recall values as the discrimination threshold for a classifier varies (Uematsu et al., 2012).

Observations demonstrated that the Precision-Recall curves for the Random Forest (RF) and Light Gradient Boosting Machine (LGBM) models lacked the desired degree of smoothness. This was attributed to the small number of points used for plotting, which led to less uniform contours for RF and interruptions in the case of LGBM (Yin et al., 2021). By generating a more closely spaced set of nodes along the Precision-Recall curve, it is possible to improve the curves' regularity. Instead of relying on a few discrete points, a greater number of threshold values should be considered. This method enables the construction of a curve that appears smoother and provides a more accurate representation of how precision and recall evolve with varying thresholds. Implementation in practice requires calculating precision and recall for a broader range of threshold values and then plotting these values accordingly (Svetnik et al., 2004). To accomplish this, a finer increment in threshold values or interpolation methods can be utilized to estimate precision and recall values at intervals between the existing discrete points. For the RF and LGBM variants, this allows for the generation of a visually appealing and instructive Precision-Recall curve.

While AdaBoost's recall rate of 0.98 demonstrates its exceptional ability to identify the vast majority of actual fraudulent transactions, its precision score of 0.19 is cause for concern. This precision value indicates that AdaBoost's accuracy in identifying fraudulent transactions is only 19%, resulting in a significant number of false alarms. In operational terms, this could result in difficulties and inconvenience for consumers. Hence, the model is not appropriate to use.

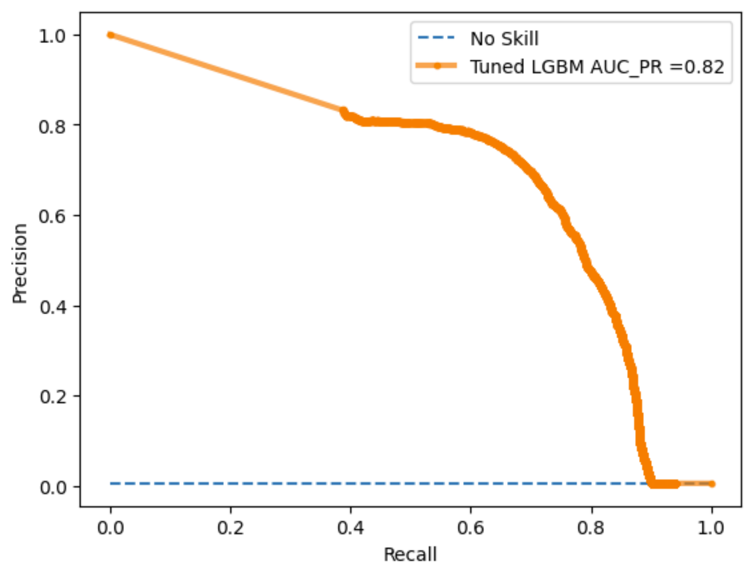

Figure 22: Plot of LGBM

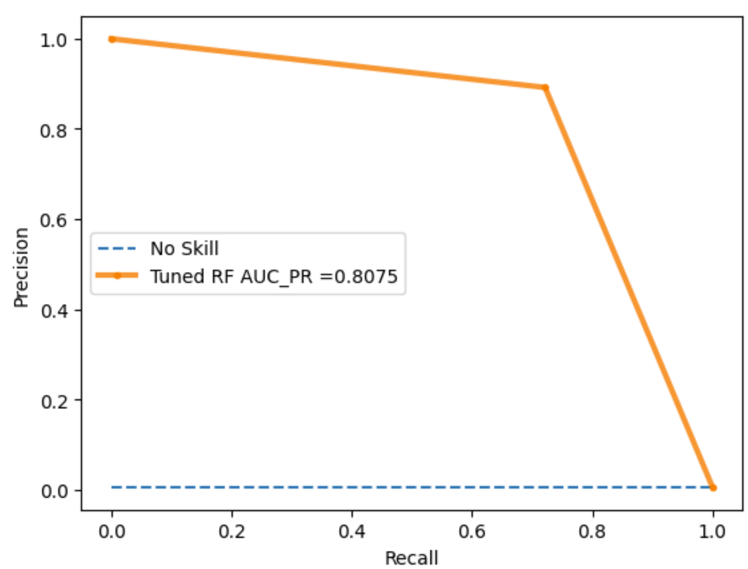

Figure 23: Plot of Random Forest

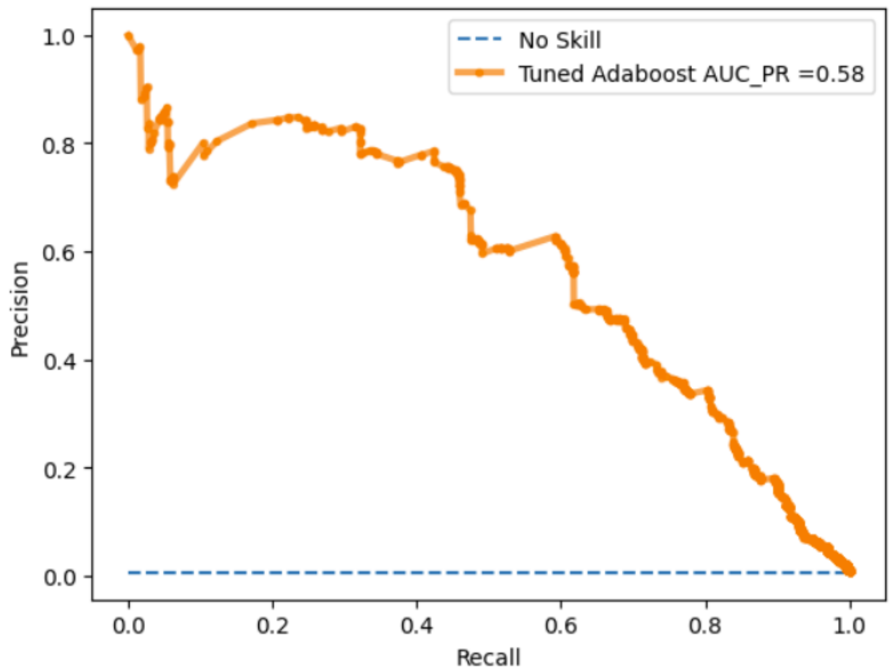

Figure 24: Plot for AdaBoost

8. Technical Choices and Considerations:

Python is favored over many other programming languages for dealing with large datasets and classification due to its unique combination of flexibility, efficacy, and a vast library ecosystem (Windows & 2021, 2006). Python's versatility is demonstrated by the fact that it is not limited to data analysis; it is a general-purpose programming language applicable to a vast array of applications. This versatility is matched by an extensive library landscape, with NumPy for numerical operations, pandas for data manipulation, and Matplotlib, Seaborn, and scikit-learn for data visualization and machine learning, collectively offering a robust toolkit for data professionals.

Despite Python's reputation for being slow at low-level numerical computations, its performance is enhanced by libraries such as NumPy, which are frequently implemented in speedier programming languages such as C and C++ (Rand et al., n.d.). Python's user-friendly, legible syntax appeals to both novice and experienced programmers, encouraging collaboration and seamless transitions between project duties. Python's active community ensures ample support, while its seamless integration with diverse data sources, cross-platform compatibility, and scalability via frameworks such as Apache Spark establish it as the language of choice for data analysis and processing.

9. Limitations

Python's prominence in data analysis and processing is well-deserved due to the language's adaptability, extensive library ecosystem, and user-friendly, legible syntax. Python does, however, have its limitations. Its interpretive nature can result in performance bottlenecks, especially when dealing with computationally intensive duties and large datasets. The Global Interpreter Lock (GIL) limits multithreading capabilities, thereby restricting the use of multicore processors to their fullest extent (Atlanta et al., 2010). Python's memory usage can be substantial, posing difficulties with very large datasets and resource-intensive programs (Whitehorn et al., n.d.). In addition, the global state management and dependency management features of Python can make it difficult to manage large-scale initiatives. Python's capabilities in data analysis, machine learning, and scientific computing continue to make it a favorite among data professionals, despite the fact that it may not be the optimal choice for mobile or real-time web development (Engelhardt et al., 2022). Mitigating its limitations often involves cautious design choices and leveraging specialized libraries or even integrating Python with other languages as required.

10. Conclusion

In summary, this study highlights four key principles in machine learning. First, model selection must correspond to the problem and objectives. Second, evaluation metrics must correspond to particular objectives, such as precision or recall. Third, model choices have real-world consequences, like transaction declines. Lastly, meticulous model comparisons enable informed decisions, highlighting the fact that success extends beyond algorithms to intelligent model selection and evaluation for effective real-world applications.

11. Recommendations

Integration of Interpretable Models: To increase model transparency and confidence, future research should concentrate on integrating interpretable machine learning models with complex ones. Models such as logistic regression, decision trees, and rule-based models can offer valuable insights into the factors that influence fraud prediction. This hybrid approach can aid in conveying model decisions to stakeholders and regulatory bodies, thereby addressing rising concerns regarding model interpretability (Caruana et al., 2015).

Real-Time Detection: Investigating real-time fraud detection systems that can analyze transactions as they occur is crucial. This involves the use of techniques such as online learning and streaming data processing. This research can lead to the creation of systems that rapidly adapt to evolving fraud schemes, thereby enhancing the security of financial institutions and consumers (Wang et al., n.d.).

12. Reflective Summary

Engaging in this research project as a student has been an invaluable experience. It focused on the hands-on application of data science and machine learning to transform theoretical knowledge into practical skills. I acquired expertise in Python libraries, model selection, data preprocessing, and management of imbalanced data. The importance of data visualization and collaboration could not be overstated, whereas obstacles presented learning opportunities. In this dynamic field, continuous learning is essential, and our novel model selection strategy contributes to the broader research landscape. Overall, this journey has bridged theory and practice, fostering curiosity and skill development.

LOGBOOK

Logbook Entry 1 - Project Kick-off (Week 1-2)

Activities
- Read 7 to 8 journals/articles to get an initial idea.
- Started with the abstract and introduction.
- Defined project objectives and scope.
- Conducted initial data collection and exploration.
- Set up the project environment and tools.
Technical Decisions
- Chose Python as the primary programming language due to its extensive libraries for data analysis.
- Selected Jupyter Notebook for code development and documentation for its interactivity.
- Jupyter Notebook enables easy sharing of project progress and results.

Logbook Entry 2 - Data Preprocessing (Week 3-4)

Activities
- Cleaned and preprocessed the raw dataset.
- Handled missing values and duplicates.
- Transformed categorical variables into numerical representations.
Technical Decisions
- Used Pandas and NumPy for data cleaning and transformation.
- Employed one-hot encoding for categorical variables.

Logbook Entry 3 - Model Selection (Week 5-6)

Activities
- Explored various machine learning algorithms: Random Forest, XGBoost, LightGBM, CatBoost, and Bagging.
- Conducted hyperparameter tuning for each algorithm.
- Compared model performance using cross-validation.
Technical Decisions
- Utilized Scikit-Learn for implementing and evaluating models.
- Employed Randomized Search for hyperparameter tuning.

Logbook Entry 4 - Model Evaluation (Week 6)

Activities
- Evaluated model performance using various metrics: Recall, Precision, F1-Score, MCC, G-Mean, AUC-PR.
- Analyzed confusion matrices and precision-recall curves.
- Identified key predictors for credit card fraud.
Technical Decisions
- Utilized Scikit-Learn and Imbalanced-Learn for performance metrics calculation.
- Visualized results using Matplotlib and Seaborn.

Logbook Entry 5 - Final Model Selection (Week 7)

Activities
- Selected XGBoost as the final model due to its superior performance.
- Documented model parameters and configuration.
- Prepared the final model for deployment.
Technical Decisions
- XGBoost was chosen based on its high predictive accuracy.
- Used Jupyter Notebook for model documentation and deployment preparation.

Logbook Entry 6 - Project Conclusion (Week 8)

Activities
- Compiled project documentation and findings.
- Conducted a final review of code and results.
- Prepared a summary presentation for stakeholders.
Technical Decisions
- Utilized Markdown for creating project reports and presentations.
- Ensured all code written with appropriate comments.

Gantt Chart

Identifying Fraudulent Transactions Using Machine Learning: A Study of Credit Card Fraud

MOON KARMAKAR 40389123

Project start date: 7/7/2023

Milestone Description	Progress	Start	Days
Introduction	5%	7/7/2023	2
Literature Review	5%	14/07/2023	3
Data Collection and Preprocessing	10%	19/07/2023	10
Feature Selection and Dimensionality Reduction	10%	29/07/2023	5
Model Development and Evaluation	15%	5/8/2023	7
Comparative Analysis of Algorithms	15%	12/8/2023	10
Interpretability and Explainability of Models	25%	22/08/2023	5
Findings	5%	27/08/2023	2
Discussions of Findings	5%	29/08/2023	2
Conclusion	2%	31/08/2023	2
Recommendations	1%	31/08/2023	Sameday
References	2%	31/08/2023	Sameday
Appendices	10%	31/08/2023	Sameday
Technical Report	15%	1/9/2023	1
LogBook	10%	3/9/2023	2
References	5%	4/9/2023	1
Appendices(Technical Report)	5%	5/9/2023	1
REVISING THE WHOLE DISSERTATION	80%	7/9/2023	2
FINAL SUBMISSION	100%	7/9/2023	Sameday

References

Atlanta, D. B.-P. P. Conference., Georgia, undefined, & 2010, undefined. (2010). Understanding the python gil. *Rrroger.Github.Io*. https://rrroger.github.io/static/pdf/UnderstandingGIL.pdf

Caruana, R., Lou, Y., Gehrke, J., Koch, P., Sturm, M., & Elhadad, N. (2015). Intelligible models for healthcare: Predicting pneumonia risk and hospital 30-day readmission. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, *2015-August*, 1721–1730. https://doi.org/10.1145/2783258.2788613

Chen, T., … C. G. sigkdd international conference on knowledge, & 2016, undefined. (2016). Xgboost: A scalable tree boosting system. *Dl.Acm.OrgT Chen, C GuestrinProceedings of the 22nd Acm Sigkdd International Conference on Knowledge, 2016•dl.Acm.Org*, *13-17-August-2016*, 785–794. https://doi.org/10.1145/2939672.2939785

Dietterich, T. G. (1998). Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. *Neural Computation*, *10*(7), 1895–1923. https://doi.org/10.1162/089976698300017197

Engelhardt, N., Penington, G., & Shahbazi-Moghaddam, A. (2022). Finding pythons in unexpected places. *Classical and Quantum Gravity*, *39*(9), 094002. https://doi.org/10.1088/1361-6382/AC3E75

Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D., & Fernández-Delgado, A. (2014). Do we need hundreds of classifiers to solve real world classification problems? *Jmlr.OrgM Fernández-Delgado, E Cernadas, S Barro, D AmorimThe Journal of Machine Learning Research, 2014•jmlr.Org*, *15*, 3133–3181. https://www.jmlr.org/papers/volume15/delgado14a/delgado14a.pdf?source=post_page---------------------------

Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., & Liu, T.-Y. (2017). LightGBM: A Highly Efficient Gradient Boosting Decision Tree. *Advances in Neural Information Processing Systems*, *30*. https://github.com/Microsoft/LightGBM.

Rand, K., Grytten, I., Pavlovic, M., Kanduri, C., bioRxiv, G. S.-, & 2022, undefined. (n.d.). BioNumPy: Fast and easy analysis of biological data with Python. *Biorxiv.Org*. https://doi.org/10.1101/2022.12.21.521373

Raschka, S. (2018). *Model Evaluation, Model Selection, and Algorithm Selection in Machine Learning*. http://arxiv.org/abs/1811.12808

Svetnik, V., Liaw, A., Tong, C., & Wang, T. (2004). Application of Breiman's Random Forest to modeling structure-activity relationships of pharmaceutical molecules. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3077*, 334–343. https://doi.org/10.1007/978-3-540-25966-4_33

Uematsu, A., Matsui, M., Tanaka, C., Takahashi, T., Noguchi, K., Suzuki, M., & Nishijo, H. (2012). Developmental Trajectories of Amygdala and Hippocampus from Infancy to Early Adulthood in Healthy Individuals. *PLOS ONE*, *7*(10), e46970. https://doi.org/10.1371/JOURNAL.PONE.0046970

Wang, T., Xu, J., Zhang, W., Gu, Z., Systems, H. Z.-G. C., & 2018, undefined. (n.d.). Self-adaptive cloud monitoring with online anomaly detection. *Elsevier*. Retrieved September 7, 2023, from https://www.sciencedirect.com/science/article/pii/S0167739X1730376X?casa_token=7k__YHQnbD wAAAAA:wOGwC_Ol8KFt8wzS8frDkxePFoWE0YkjnswrCHTQAEDbt1P_MzdtMZoiNUko3175y64lAM2IYg

Whitehorn, N., Santen, J. van, Communications, S. L.-C. P., & 2013, undefined. (n.d.). Penalized splines for smooth representation of high-dimensional Monte Carlo datasets. *Elsevier*. Retrieved September 7, 2023, from https://www.sciencedirect.com/science/article/pii/S0010465513001434?casa_token=tHB_NGTyCwQAAAAA:QobskGTRqkEzPlNkv_s8-i_usTgiESbS2fw6Z12XWzWh6-sjiitkdhvUZxFBhACnmL3VcgYKDg

Windows, W. P.-P. R. for, & 2021, undefined. (2006). Python. *Citeseer*. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=1f2ee3831eebfc97bfafd514ca2abb7e2c5c86bb

Yin, L., Li, B., Li, P., Intelligence, R. Z.-C. T. on, & 2023, undefined. (2021). Research on stock trend prediction method based on optimized random forest. *Wiley Online Library*, *8*(1), 274–284. https://doi.org/10.1049/cit2.12067

Appendix 1- Code

```
#Import the essential python libraries for data manipulation,data processing and data analysis
!pip install pandas
import pandas as pd
!pip install numpy
import numpy as np
!pip install os_sys
import os
!pip install random
import random
## Import math library for mathematical computations and also import scipy.stats for statistical functions.
!pip install Mathematics-Module
import math
!pip install scipy
import scipy.stats
from scipy import stats
from scipy.stats import pointbiserialr
from scipy.stats import chi2_contingency
from scipy.stats import randint as sp_randint
!pip install researchpy
import researchpy as rp
!pip install ppscore
import ppscore as pps
## Import ploting libraries
!pip install matplotlib
import matplotlib.pyplot as plt
from matplotlib import pyplot
#To enable plotting graphs in notebook
%matplotlib inline
#Import seaborn
!pip install seaborn
import seaborn as sns


## Import Python scikit-learn library for Machine Learning
!pip install scikit-learn
import sklearn
# Import libraries for ANOVA feature selection
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import SelectKBest
# Import libraries to split the dataset into train and test set
from sklearn.model_selection import train_test_split
## Encoding. Import the library for categorical encoding
!pip install category_encoders
import category_encoders as ce
## Resampling
# To deal with the class imbalance problem: imbalance learn
!pip install imbalanced-learn
import imblearn
from imblearn.pipeline import Pipeline
# RUS = Random Undersampling
```

```
from imblearn.under_sampling import RandomUnderSampler
# SMOTE = Synthetic Minority Oversampling Technique
!pip install smote-variants
from imblearn.over_sampling import SMOTE
# Borderline SMOTE
from imblearn.over_sampling import BorderlineSMOTE


!pip install collection
from collections import Counter

## Classification Algorithms
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import BaggingClassifier
!pip install xgboost
from xgboost import XGBClassifier
!pip install lightgbm
from lightgbm import LGBMClassifier
!pip install catboost
from catboost import CatBoostClassifier
import catboost as ctb

# Import libraries for model evaluation
from sklearn import metrics
from sklearn.metrics import classification_report
from imblearn.metrics import classification_report_imbalanced
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import average_precision_score
from sklearn.metrics import precision_recall_curve, plot_precision_recall_curve
from sklearn.metrics import PrecisionRecallDisplay
from sklearn.metrics import auc
from imblearn.metrics import geometric_mean_score
from sklearn.metrics import make_scorer

# For Hyperparameter Tuning
from sklearn.model_selection import RandomizedSearchCV

#Cross Validation libraries
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import cross_validate
from sklearn.model_selection import StratifiedKFold

## Other libraries
!pip install DateTime
import datetime

import pandas as pd
```

```
# Concatenate fraudTrain and fraudTest, and create a new dataframe: fraud_df
fraud_train = pd.read_csv('C:/Users/moon/OneDrive/Desktop/fraudTrain.csv')
fraud_test = pd.read_csv('C:/Users/moon/OneDrive/Desktop/fraudTest.csv')
fraud_df = pd.concat([fraud_train, fraud_test])

fraud_df.info()

#Alter the data type of the binary target variable 'is_fraud' to 'category'.
fraud_df["is_fraud"] = fraud_df["is_fraud"].astype('category')
fraud_df["is_fraud"].dtypes

# Change the data type of 'gender' and 'category' variables to 'category'.
fraud_df["gender"] = fraud_df["gender"].astype('category')
fraud_df["category"] = fraud_df["category"].astype('category')


# Update the data type of 'dob' and 'trans_day_trans_time' variables to datetime.
fraud_df['dob'] = pd.to_datetime(fraud_df['dob'])
fraud_df['trans_date_trans_time'] = pd.to_datetime(fraud_df['trans_date_trans_time'])

#Transform the 'cc_num' and 'zip' variables into strings
#as they are not integers but rather nominal variables with numeric representations.
fraud_df["zip"] = fraud_df["zip"].astype('str')
fraud_df["cc_num"] = fraud_df["cc_num"].astype('str')

# remove irrelevant column
fraud_df = fraud_df.drop(columns="Unnamed: 0")
fraud_df = fraud_df.drop(columns="unix_time")
fraud_df = fraud_df.drop(columns="merch_lat")
fraud_df = fraud_df.drop(columns="merch_long")

fraud_df.isnull().sum().sum()

fraud_df.duplicated().sum()

fraud_df.nunique()

# Instances of legitimate and fraudulent transactions.
occurences = fraud_df['is_fraud'].value_counts()
occurences
#The following code retrieves the counts of legitimate ('0') and fraudulent ('1') transactions:
# '0' denotes legitimate transactions.
#'1' denotes fraudulent transactions.

#visual representation of the data using histograms
fraud_df.hist(figsize = (15, 15))
plt.show()

# Plot the number of legitimate and fraudulent transactions
import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt
colors = ['blue', 'yellow']
sns.countplot(data=fraud_df, x='is_fraud', palette=colors).set(title='Volume of Legitimate and Fraudulent
Transactions')
plt.show()
# Blue = Legitimate transaction "0"
# Yellow = Fraudulent transaction "1"
# THIS DATASET IS HIGHLY IMBALANCED

# Plot the number of legitimate and fraudulent transactions by category
colors = ['blue', 'red']
plt.figure(figsize=(20,8))
plt.title('Volume of Legitimate and Fraudulent transactions by Category')
sns.barplot(x="category", y='trans_num', hue="is_fraud", palette=colors,
        data=fraud_df.groupby(['category', 'is_fraud']).agg({'trans_num' : 'count'}).reset_index())


# Compute some relevant summary statistics for the variable 'Amount'('amt')
# Separate legitimate and fraudulent transactions
legitimate = fraud_df[fraud_df['is_fraud'] == 0]
fraudulent = fraud_df[fraud_df['is_fraud'] == 1]
max_legitimate_amt=legitimate['amt'].max()
max_fraudulent_amt=fraudulent['amt'].max()
import statistics
mean_legitimate_amt=legitimate['amt'].mean()
mean_fraudulent_amt=fraudulent['amt'].mean()
median_legitimate_amt=legitimate['amt'].median()
median_fraudulent_amt=fraudulent['amt'].median()

print('Maximun Legitimate Transactions Amount: {}'.format(max_legitimate_amt))
print('Maximun Fraudulent Tragactions Amount: {}'.format(max_fraudulent_amt))

print('Average Legitimate Transactions Amount: {}'.format (mean_legitimate_amt))
print('Average Fraudulent Transactions Amount: {}'.format(mean_fraudulent_amt))

print('Median of Legitimate Transactions Amount: {}'.format (median_legitimate_amt))
print('Median of Fraudulent Transactions Amount: {}'.format(median_fraudulent_amt))

import pandas as pd
import matplotlib.pyplot as plt

# Separate legitimate and fraudulent transactions
legitimate = fraud_df[fraud_df['is_fraud'] == 0]
fraudulent = fraud_df[fraud_df['is_fraud'] == 1]

# Compute maximum amounts
max_legitimate_amt = legitimate['amt'].max()
max_fraudulent_amt = fraudulent['amt'].max()

# Create a DataFrame for plotting
data = pd.DataFrame({
  'Transaction Type': ['Legitimate', 'Fraudulent'],
```

```
    'Maximum Amount': [max_legitimate_amt, max_fraudulent_amt]
})

# Create a bar chart
plt.figure(figsize=(8, 6))
plt.bar(data['Transaction Type'], data['Maximum Amount'])
plt.xlabel('Transaction Type')
plt.ylabel('Maximum Amount')
plt.title('Maximum Transaction Amount by Transaction Type')
plt.show()


import numpy as np

# Create the variable 'age' at the time of the transaction
fraud_df['age'] = np.round((fraud_df['trans_date_trans_time'] - fraud_df['dob']) / np.timedelta64(1, 'Y'))

# Convert 'age' to integer
fraud_df['age'] = fraud_df['age'].astype('int')

# Display the data type of the 'age' column
fraud_df.dtypes[['age']]

# Create Variable transaction hour
# Derive the feature 'transaction hour. Transaction hour for all transactions
#Extract the hour of the transaction from the variable 'trans_day_trans_time'
fraud_df[ 'transaction_hour'] = fraud_df['trans_date_trans_time'].dt.hour
fraud_df['transaction_hour']

#Transaction Hour for Fraudulent Transactions
#Extract the hour of fraudulent transactions from the variable 'trans_day_trans_time'
fraudulent=fraud_df[fraud_df['is_fraud']==1]

fraudulent['transaction_hour']= fraudulent['trans_date_trans_time'].dt.hour
fraudulent['transaction_hour']

sns.countplot(data=fraudulent, x='transaction_hour').set(title='Volume of Fraudulent Transactions by Hour')
plt.show()
#Plot the Number of Fraudulent Transactions per hour

# Encode the 'transaction_hour' variable

# 'risky-transactions': Occurring from 9:00 PM to 4:00 AM

fraud_df['hourEncoded'] =0
fraud_df.loc[fraud_df.transaction_hour< 4, 'hourEncoded']= 1

fraud_df.loc[fraud_df.transaction_hour> 21, 'hourEncoded'] = 1
# Utilizing the time frame between 21:00 and 4:00 yields a significantly higher Cramer's V coefficient
#compared to using the interval from 22:00 to 4:00.
#Cramer's V is a statistic used to measure the strength of association between categorical variables in a contingency
table.
```

```
# Generate the 'day-of-week' variable
# Extract the day of the week for each transaction
fraud_df['day_of_week'] =fraud_df['trans_date_trans_time'].dt.day_name()
fraud_df['day_of_week']

#Create the variable 'month of transaction'
#Extract the year_month for all transactions
fraud_df['year_month']=fraud_df['trans_date_trans_time'].dt.to_period('M')
fraud_df['year_month']

#Extract the Month of transaction
fraud_df['month_of_trans']=fraud_df['year_month'].dt.month
fraud_df['month_of_trans']

# Generate the 'time since last transaction' variable
# Time since last transaction = 'time_since_last_trans' and is measured in 'seconds'
# I've developed a new function named 'timeDifference' to calculate the time elapsed
#since the cardholder's previous credit card transaction.
def timeDifference(x):
  x['time_since_last_trans']= x.trans_date_trans_time-x.trans_date_trans_time.shift()
  return x


#cc-num identifies a card holder
fraud_df = fraud_df.groupby('cc_num').apply(timeDifference)

fraud_df['time_since_last_trans']= fraud_df['time_since_last_trans'].dt.seconds

# Examine null values for this newly created feature.
# Given that it calculates the time since the last transaction,
# it's expected to have some null values, particularly for customers making their first transaction!

fraud_df['time_since_last_trans'].isnull().sum().sum()

# Replace the null values by 0. It means'0' seconds from last transaction

fraud_df['time_since_last_trans']= fraud_df['time_since_last_trans'].replace(np.nan, 0)

fraud_df['time_since_last_trans'].isnull().sum().sum()

# Generate Frequencies of Transactions made in the last 1/ 7/ 14 / 30 / 60 days

import pandas as pd
#Volume of Transactions made in a day
def last1DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_1_days').sort_index()
    count_1_days = temp.rolling(window=2, min_periods=0).count()
    x['last_1_days_trans_count'] = count_1_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last1DaysTransCount)
```

```
#Volume of Transactions made in the last 7 days
def last7DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_7_days').sort_index()
    count_7_days = temp.rolling(window=7,min_periods=1).count()
    x['last_7_days_trans_count'] = count_7_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last7DaysTransCount)

#Volume of Transactions made in the last 14 days
def last14DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_14_days').sort_index()
    count_14_days = temp.rolling(window=14,min_periods=1).count()
    x['last_14_days_trans_count'] = count_14_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last14DaysTransCount)

#Volume of Transactions made in the last 30 days
def last30DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_30_days').sort_index()
    count_30_days = temp.rolling(window=30,min_periods=1).count()
    x['last_30_days_trans_count'] = count_30_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last30DaysTransCount)

#Volume of Transactions made in the last 60 days
def last60DaysTransCount(x):
    temp = pd.Series(x.index, index=x.index, name='count_60_days').sort_index()
    count_60_days = temp.rolling(window=60,min_periods=1).count()
    x['last_60_days_trans_count'] = count_60_days.values
    return x
fraud_df = fraud_df.groupby('cc_num').apply(last60DaysTransCount)

#I'm removing the DOB column because I've already calculated the age of the individuals
#in the dataset using their date of birth (DOB).
#Since I now have the age information, keeping the DOB column would be redundant and unnecessary for
#the analysis and modeling tasks.

fraud_df = fraud_df.drop(columns="dob")

# Remove the "trans_date_trans_time" variable as it is redundant variable
fraud_df= fraud_df.drop(columns="trans_date_trans_time")

#Remove the variable trans_num (transaction number) as it is unique and irrelevant for modelling purposes.
fraud_df = fraud_df.drop(columns="trans_num")

# Feature Selection

fig, ax = plt.subplots(figsize=(20,10))
sns.heatmap(fraud_df.corr(),annot=True).set_title('Correlation heatmap')

# Anova F- test for feature selection
#The ANOVA F-test is a statistical method used for feature selection.
```

```
#It assesses if there are significant differences in the means of a
#continuous feature across different categories of a categorical target variable.
#Features with high F-values and low p-values are considered important for prediction.

#Creating new dataset
fraud_df2 = fraud_df[['amt', 'age', 'transaction_hour',
                     'hourEncoded', 'time_since_last_trans',
                     'last_7_days_trans_count', 'last_14_days_trans_count',
                     'last_30_days_trans_count', 'last_60_days_trans_count', 'is_fraud']]

# Separate the target variable 'is_fraud' from the dataframe
X = fraud_df2.loc[:, fraud_df2.columns != 'is_fraud']
y = fraud_df2['is_fraud']

#Conduct ANOVA F-test for feature selection

fs = SelectKBest(score_func=f_classif, k=9)

#The fit_transform method in feature selection is used to both fit the feature selection algorithm to
#the training data (X and y) and transform the data to keep only the selected features.
fit = fs.fit_transform(X, y)

#print(X_selected_fs.shape)

# apply feature selection
X_selected= fs.fit_transform(X,y)
print(X_selected.shape)

#see the columns that have been selected

X.columns[fs.get_support(indices=True)].tolist()

#Test the association between the categorical Independent variables and the target variable 'is_fraud'

from scipy.stats import chi2_contingency

# Define the list of categorical column names
categorical_columns = ['category', 'street', 'zip', 'city', 'state',
                      'first', 'last', 'cc_num', 'job', 'merchant', 'day_of_week', 'month_of_trans']

chi2_check = []

for i in categorical_columns:
    if chi2_contingency(pd.crosstab(fraud_df['is_fraud'], fraud_df[i]))[1] < 0.05:
        chi2_check.append('Reject Null Hypothesis')
    else:
        chi2_check.append('Fail to Reject Null Hypothesis')

res = pd.DataFrame(data=[categorical_columns, chi2_check]).T
res.columns = ['Column', 'Hypothesis']

print(res)
```

```
# The Null Hypothesis states that there is no association between the categorical predictor and the target
#variable 'is_fraud'

import researchpy as rp

crosstab, test_results_category, expected = rp.crosstab(fraud_df["is_fraud"], fraud_df["category"],
                                            test="chi-square",
                                            expected_freqs=True,
                                            prop="cell")
test_results_category

#Bonferroni Correction Post hoc test
#The Bonferroni Correction Post hoc test is a statistical method used in hypothesis testing and statistical analysis.

categorical_columns=['category']

chi2_check = []
for i in categorical_columns:
    contingency_table = pd.crosstab(fraud_df['is_fraud'], fraud_df[i])
    chi2_stat, p_value, dof, expected = chi2_contingency(contingency_table)

    if p_value < 0.05:
        chi2_check.append('Reject Null Hypothesis')
    else:
        chi2_check.append('Fail to Reject Null Hypothesis')

res = pd.DataFrame(data=[categorical_columns, chi2_check]).T
res.columns = ['Columns', 'Hypothesis']
print(res)

check = {}

for i in res[res['Hypothesis'] == 'Reject Null Hypothesis']['Columns']:
    dummies = pd.get_dummies(fraud_df[i])
    bon_p_value = 0.05 / fraud_df[i].nunique()

    for series in dummies:
        if chi2_contingency(pd.crosstab(fraud_df['is_fraud'], dummies[series]))[1] < bon_p_value:
            check['{}-{}'.format(i, series)] = 'Reject Null Hypothesis'
        else:
            check['{}-{}'.format(i, series)] = 'Fail to Reject Null Hypothesis'

res_chi_ph = pd.DataFrame(data=[check.keys(), check.values()]).T
res_chi_ph.columns = ['Pair', 'Hypothesis']
print(res_chi_ph)

import pandas as pd

columns_to_keep = ['amt', 'age', 'hourEncoded', 'time_since_last_trans',
                  'last_7_days_trans_count', 'last_14_days_trans_count',
                  'last_30_days_trans_count', 'last_60_days_trans_count',
```

```
                 'category', 'day_of_week', 'is_fraud']

# Create a new DataFrame with the selected columns
fraud_data = fraud_df[columns_to_keep].copy()

# Convert the "day_of_week" column to the "category" data type
fraud_data['day_of_week'] = fraud_data['day_of_week'].astype('category')

# Reset the index to start from 0
fraud_data.reset_index(drop=True, inplace=True)

# Print the shape of the new DataFrame
print(fraud_data.shape)

# Display data type information
fraud_data.info()



# Split the Dataset into Train and Test Set

X = fraud_data.drop(columns=['is_fraud'])  # Exclude the target variable
y = fraud_data['is_fraud']

#Split the dataset into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=1, stratify=y)


#Using a fixed "random state" value of 1 ensures consistent outcomes across multiple runs or executions.
# 70% of the data will be used to train the models
#30% of the data will be used to test the models

training_set_size = len(X_train)
test_set_size = len(X_test)
print(training_set_size)
print(test_set_size)

# Categorical Encoding

# Using CATBOOST ENCODING
#define cateboost encoder
cbe_encoder= ce.cat_boost.CatBoostEncoder()

feature_list=['category', 'day_of_week'] # the categorical variables i want to encode

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
# Fit the encoder and transform the feature
# Fit the encoder and transform the feature
cbe_encoder = ColumnTransformer([
   ('categorical', OneHotEncoder(handle_unknown='ignore'), feature_list)
])
```

```
train_cbe= cbe_encoder.fit_transform(X_train[feature_list],y_train)
test_cbe= cbe_encoder.transform(X_test[feature_list])

# Dealing with the Class Imbalance : Balance the Train set

#define Resampling pipeline

from imblearn.over_sampling import BorderlineSMOTE
from imblearn.under_sampling import RandomUnderSampler

# Define Resampling pipeline
under = RandomUnderSampler(sampling_strategy=0.05, random_state=42)
over = BorderlineSMOTE(sampling_strategy=0.9, random_state=42)
steps = [('u', under), ('o', over)]
pipeline = Pipeline(steps=steps)


#random state = 42 is to produce the same results across differents runs
# sampling strategy parameters have been manually tuned

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler
from imblearn.pipeline import Pipeline as ImblearnPipeline

# Define your categorical columns
categorical_cols = ['category', 'day_of_week']

# Define the column transformer to apply encoding
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_cols)
    ],
    remainder='passthrough'  # Pass through any remaining columns as-is
)

# Create a pipeline with preprocessing and resampling
pipeline = ImblearnPipeline([
    ('preprocessor', preprocessor),
    ('sampler', RandomOverSampler())  # Use the desired resampling technique
])


# Adjust dataset sampling
# Modify the dataset accordingly
# Apply fitting and transformation to the dataset
X_train_sampled, y_train_sampled = pipeline.fit_resample(X_train, y_train)

counter=Counter(y_train_sampled)
```

```
print(counter)
# summarize class distribution before and after applying RUS and Borderline-SMOTE
counter= Counter(y_train)
print('Before RUS + BorderlineSMOTE', counter)
counter= Counter(y_train_sampled)
print('After RUS + BorderlineSMOTE', counter)

# MODEL IMPLEMENTATION

import pandas as pd

# Assuming X_train_sampled and y_train_sampled are NumPy arrays
# Convert them to Pandas DataFrames
X_train_sampled = pd.DataFrame(X_train_sampled)
y_train_sampled = pd.DataFrame(y_train_sampled)

# Concatenate them along the columns (axis=1)
balanced_train = pd.concat([X_train_sampled_df, y_train_sampled_df], axis=1)


# seperate the target variable 'is_fraud' from the sampled_train dataframe
X_balanced_train= balanced_train.loc[:, balanced_train.columns != 'is_fraud']
y_balanced_train= balanced_train.is_fraud

# ADABOOST

# Import AdaBoostClassifier
from sklearn.ensemble import AdaBoostClassifier

# Define the AdaBoost model
AdaBoost = AdaBoostClassifier(random_state=42)

# Fit the model on the train set
AdaBoost.fit(X_balanced_train_encoded, y_balanced_train)

from sklearn.tree import DecisionTreeClassifier
params_adaboost = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.1, 0.2, 0.3],
    'base_estimator': [None, DecisionTreeClassifier(max_depth=1), DecisionTreeClassifier(max_depth=2)],
    'algorithm': ['SAMME', 'SAMME.R'],
}

# Due to the large size of the training set, use 3-fold cross-validation
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

# Define the AdaBoost base model
adaboost_base = AdaBoostClassifier(random_state=42)


# Instantiate RandomizedSearchCV and pass in the hyperparameters
rand_AdaBoost = RandomizedSearchCV(adaboost_base, params_adaboost, scoring='f1',
```

```
                              cv=cv, n_jobs=1, verbose=1, random_state=42, n_iter=10)

rand_AdaBoost.fit(X_balanced_train_encoded, y_balanced_train)

print(rand_AdaBoost.best_params_)

AdaBoost_best = rand_AdaBoost.best_estimator_

print(AdaBoost_best)  # best-tuned model

# RANDOM FOREST

# Import Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier

# Define the model
rf= RandomForestClassifier(random_state = random.seed(42), n_jobs =-1, verbose = 1)

from sklearn.preprocessing import LabelEncoder

# List of column names representing categorical features
categorical_features = ['category', 'day_of_week']

# Apply label encoding to categorical features
label_encoder = LabelEncoder()
X_balanced_train_encoded = X_balanced_train.copy()

for col in categorical_features:
    X_balanced_train_encoded[col] = label_encoder.fit_transform(X_balanced_train_encoded[col])

# Fit the model on training set
rf.fit(X_balanced_train_encoded, y_balanced_train)

# RF Hyperparameter Optimisation using RandomizedSearchCV

#HYPERPARAMETER SEARCH


from sklearn.model_selection import StratifiedKFold, RandomizedSearchCV
from sklearn.ensemble import RandomForestClassifier

#create a dictionary with the list of hyperparameters
params_rf = {
    'n_estimators': [100, 300, 500],
    'max_depth': [None, 4, 81],
    'min_samples_split': [2, 20, 200, 50, 100, 200],
    'min_samples_leaf': [1, 10, 100, 25, 50, 100]
}


#Due to the large size of the training set I will use 3-fold cross validations (3-fold CV)
```

```
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

#Define the base RF model (fl)

rf1 = RandomForestClassifier(random_state=42, n_jobs=-1, verbose=1)

#APPLY RANDOMIZED SEARCH

#Instantiate RandomizedSearchCV and pass in the hyperparameters
#rand_rf= RandomizedSearchCV(rf1, params_rf, scoring = 'f1', cv= cv, n_jobs=-1, verbose=1, random_state=42)

rand_rf = RandomizedSearchCV(
    RandomForestClassifier(), param_distributions=params_rf, scoring='f1', cv=cv,
    n_jobs=1, verbose=1, random_state=42, n_iter=10
)

# Fit the model on training set
rand_rf.fit(X_balanced_train_encoded, y_balanced_train)

#Optimised Hyperparameters for RF

# print the best hyperparameters that maximize scoring
best_params = rand_rf.best_params_
print(best_params)

# BAGGING

#Import Bagging Classifier
from sklearn.ensemble import BaggingClassifier

# Define the bagging Model
bagging = BaggingClassifier(random_state= random.seed(42), n_jobs=1, verbose=1)

# Fit the model on training set
bagging.fit(X_balanced_train_encoded, y_balanced_train)

#Bagging Hyperparameter Optimisation using RandomizedSearchCV

#HYPERPARAMETER SEARCH

#create a dictionary with the list of hyperparameters

params_bagging= {'n_estimators': [10, 50, 100, 300, 500],
                'max_samples' : [0.3, 0.5, 1.0],
               'max_features': [0.3, 0.5, 1.0]
                }

#Due to the large size of the training set I will use 3-fold cross validationa (3-fold CV)
cv= StratifiedKFold(n_splits =3, shuffle=True, random_state=42)

#Define the base Bagging base model (bagging1)
```

```
bagging1 = BaggingClassifier(random_state = 42, n_jobs = -1, verbose=1)

#APPLY RANDOMIZED SEARCH

#Instantiate RandomizedSearchCV and pass in the hyperparameters

#rand_bagging = RandomizedSearchCV(bagging1, params_bagging, scoring= 'f1', cv=cv, n_jobs=-1, verbose=1,
random_state=42)
rand_bagging = RandomizedSearchCV(
    BaggingClassifier(), param_distributions=params_bagging, scoring='f1', cv=cv,
    n_jobs=1, verbose=1, random_state=42, n_iter=10
)

#Fit the model on training set
rand_bagging.fit(X_balanced_train_encoded, y_balanced_train)

#Optimised Hyperparameters for Bagging

#Print the best hyperparameters that maximize scoring
print(rand_bagging.best_params_)

Bagging_best = rand_bagging.best_estimator_

print(Bagging_best) #best tuned model

# XGBOOST

#Define the XGBoost model
from xgboost import XGBClassifier

XGBoost=XGBClassifier(random_state = 42, n_jobs=-1, verbosity = 1)

#Fit the model on the train set
XGBoost.fit(X_balanced_train_encoded, y_balanced_train)

#XGBoost Hyperparameter Optimisation using RandomizedSearchCV

#HYPERPARAMETER SEARCH

#create a dictionary with the list of hyperparameters
params_xgboost={
    'n_estimators':[100, 300, 500],
    'learning_rate': [0.1, 0.2, 0.3],
    'max_depths': [4, 6],
    'gamma':[0, 0.1, 0.2],
    'subsample': [0.5, 0.75, 1],
    'colsample_bytree': [0.5, 0.75, 1],
    'min_child_weight': [1, 5, 25],
    }
```

```
#Due to the large size of the training set will use 3-fold cross validations (3-fold cv)
cv =StratifiedKFold(n_splits=3, shuffle= True, random_state=42)

#Define the XGBoost base mode

xgboost1 = XGBClassifier(random_state=42, n_jobs=1, verbosity=1) #random state is to get same results in every

#APPLY RANDOMIZED SEARCH

#Instantiate RandomizedSearchcv and pass in the hyperparameters

#Fit the model on the train set
rand_XGBoost = RandomizedSearchCV(xgboost1, params_xgboost, scoring='f1',
                        cv=cv, n_jobs=1, verbose=1, random_state=42, n_iter=10)

rand_XGBoost.fit(X_balanced_train_encoded, y_balanced_train)

print(rand_XGBoost.best_params_)

XGBoost_best = rand_XGBoost.best_estimator_

print(XGBoost_best) #best tuned model

#LIGHTGBM

from lightgbm import LGBMClassifier
Lgbm= LGBMClassifier(random_state=42, n_jobs= -1, verbosity=1)

# Fit the model on the train set
Lgbm.fit(X_balanced_train, y_balanced_train)

#HYPERPARAMETER SEARCH
# create a dictionary with the list of hyperpara

params_lgbm ={ 'n_estimators': [100, 300, 500, 1000],
        'learning_rate': [0.05, 0.08, 0.1, 0.21],
        'max_depth': [4, 5, 6, 7],
        'num_leaves': sp_randint(500, 5000),
        'min_data_in_leaf': sp_randint(500, 3500),
        'colsample_bytree': [0.5, 0.75, 1],
        'max_bin': sp_randint (50, 2000),
        'subsample': [0.5, 0.75, 11],
        }

#Due to the large size of the training set will use 3-fold cross validations (3-fold cv)
cv =StratifiedKFold(n_splits=3, shuffle= True, random_state=42)

#Define the XGBoost base mode

Lgbm1 = LGBMClassifier(random_state=42, n_jobs=1, verbosity=1) #random state is to get same results in every
```

```
#APPLY RANDOMIZED SEARCH

# Instantiate RandomizedSearchCV and pass in the hyperparameters
rand_lgbm = RandomizedSearchCV(Lgbm1, params_lgbm, scoring='f1', cv=cv, n_jobs=1, verbose=1,
random_state=42, n_iter=10)

# Fit the model on the train set
rand_lgbm.fit(X_balanced_train_encoded, y_balanced_train)


print(rand_lgbm.best_params_)

LGBM_best = rand_lgbm.best_estimator_

print(LGBM_best) #best tuned model

# CATBOOST

from catboost import CatBoostClassifier

CatBoost = CatBoostClassifier(random_state=42, thread_count=-1, verbose=1)
# Fit the model on the train set
CatBoost.fit(X_balanced_train_encoded, y_balanced_train)

# CatBoost Hyperparameter Optimisation using RandomizedSearchCV

#HYPERPARAMETER SEARCH

# create a dictionary with the list of hyperparameters

params_catboost = {'iterations': [500, 850, 1000, 1500, 2000],
        'learning_rate': [0.03, 0.05, 0.08, 0.1, 0.3],
        'depth': [4, 5, 6, 71],
        'l2_leaf_reg': [1.0, 3.0, 5.0, 8.0],
        }

#Due to the large size of the training set will use 3-fold cross validations (3-fold cv)
cv = StratifiedKFold(n_splits=3, shuffle=True, random_state=42)

#Define the XGBoost base mode

CatBoost1 = CatBoostClassifier(random_state=42, verbose=1) #random state is to get same results in every

#APPLY RANDOMIZED SEARCH

# Instantiate RandomizedSearchCV with adjusted parameters
rand_catboost = RandomizedSearchCV(
  CatBoost1, params_catboost, scoring='f1', cv=cv, n_jobs=1,
  verbose=1, random_state=42, n_iter=10
)
```

```
# Fit the model on the train set
rand_catboost.fit(X_balanced_train_encoded, y_balanced_train)

print(rand_catboost.best_params_)

# Access the best estimator
CatBoost_best = rand_catboost.best_estimator_

print(CatBoost_best) #best tuned model

# Model Evaluation

feature_list=['category', 'day_of_week']

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
# Fit the encoder and transform the feature
# Fit the encoder and transform the feature
cbe_encoder = ColumnTransformer([
    ('categorical', OneHotEncoder(handle_unknown='ignore'), feature_list)
])

train_cbe= cbe_encoder.fit_transform(X_train[feature_list],y_train)
test_cbe= cbe_encoder.fit_transform(X_test[feature_list],y_test)

from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
import pandas as pd

# List of column names representing categorical features
cat_features = ['category', 'day_of_week']

# Assuming you have already defined X_test_sampled, y_test_sampled, and balanced_test as you mentioned

# Apply label encoding to categorical features
label_encoder = LabelEncoder()
X_test_encoded = X_test.copy()

for col in cat_features:
    X_test_encoded[col] = label_encoder.fit_transform(X_test_encoded[col])

# Define a OneHotEncoder for all categorical features
onehot_encoder = ColumnTransformer(
    transformers=[('cat', OneHotEncoder(), cat_features)],
    remainder='passthrough'  # Pass through non-categorical features
)

# Fit the onehot_encoder on the test data
onehot_encoder.fit(X_test_encoded)

# Transform the test data using the trained OneHotEncoder
X_balanced_test_encoded = onehot_encoder.transform(X_test_encoded)
```

```
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import BorderlineSMOTE
from imblearn.under_sampling import RandomUnderSampler

# Define the resampling strategies
over = BorderlineSMOTE(sampling_strategy=0.9)
under = RandomUnderSampler(sampling_strategy=0.05)

# Create the imblearn pipeline with resampling steps
steps = [('u', under), ('o', over)]
pipeline = Pipeline(steps=steps)

# Fit and transform your encoded test data using the pipeline
X_balanced_test, y_balanced_test = pipeline.fit_resample(X_test_encoded, y_test)

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler
from imblearn.pipeline import Pipeline as ImblearnPipeline

# Define your categorical columns
categorical_cols = ['category', 'day_of_week']

# Define the column transformer to apply encoding
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(), categorical_cols)
    ],
    remainder='passthrough'  # Pass through any remaining columns as-is
)

# Create a pipeline with preprocessing and resampling
pipeline = ImblearnPipeline([
    ('preprocessor', preprocessor),
    ('sampler', RandomOverSampler())  # Use the desired resampling technique
])


# resample dataset
# transform the dataset
# Fit and transform the dataset
X_test_sampled, y_test_sampled = pipeline.fit_resample(X_test, y_test)

counter=Counter(y_test_sampled)
print(counter)

from imblearn.over_sampling import RandomOverSampler
from collections import Counter
```

```
# Calculate the class proportions based on the original class distribution
class_distribution = Counter(y_test)
total_samples = len(y_train)
desired_proportion_0 = class_distribution[0] / total_samples
desired_proportion_1 = class_distribution[1] / total_samples

# Calculate the desired class counts based on the desired proportions
desired_count_0 = round(desired_proportion_0 * total_samples)
desired_count_1 = round(desired_proportion_1 * total_samples)

# Define the desired class distribution
desired_distribution = {0: desired_count_0, 1: desired_count_1}

# Create the RandomOverSampler with the desired distribution
ros = RandomOverSampler(sampling_strategy=desired_distribution, random_state=1)

# Resample the dataset
X_test_sampled, y_test_sampled = ros.fit_resample(X_test, y_test)

# Summarize the new class distribution
counter = Counter(y_test_sampled)
print(counter)
# summarize class distribution before and after applying RUS and Borderline-SMOTE
counter= Counter(y_test)
print('Before RUS + BorderlineSMOTE', counter)
counter= Counter(y_test_sampled)
print('After RUS + BorderlineSMOTE', counter)

from lightgbm import LGBMClassifier

# Assuming X_test_encoded has been properly encoded and preprocessed

# Make predictions using the trained LGBM_best model and the encoded test dataset
y_LGBM_best_pred = LGBM_best.predict(X_test_encoded)

# Round the predictions to get binary values
LGBM_best_predictions = [round(value) for value in y_LGBM_best_pred]


from sklearn.ensemble import AdaBoostClassifier
# Assuming you have trained and tuned the AdaBoost model and stored it as adaboost_best
y_AdaBoost_best_pred =AdaBoost_best.predict(X_balanced_test_encoded)
Adaboost_best_predictions = [round(value) for value in y_AdaBoost_best_pred]


from sklearn.ensemble import RandomForestClassifier
# Assuming you have already trained and selected the best Random Forest model
# Let's call it rand_rf_best
best_rand_rf_model = rand_rf.best_estimator_

# Make predictions using the best Random Forest model and the encoded test dataset
y_rand_rf_pred = best_rand_rf_model.predict(X_test_encoded)
```

```
# Round the predictions to get binary values
rand_rf_predictions = [round(value) for value in y_rand_rf_pred]

from sklearn.ensemble import BaggingClassifier

# Assuming you have already trained and selected the best BaggingClassifier model
# Let's call it bagging_best
y_Bagging_best_pred = Bagging_best.predict(X_test_encoded)
Bagging_best_predictions = [round(value) for value in y_Bagging_best_pred]


from xgboost import XGBClassifier

# Let's call it xgb_best
y_XGBoost_best_pred = XGBoost_best.predict(X_test_encoded)
XGBoost_best_predictions = [round(value) for value in y_XGBoost_best_pred]


from catboost import CatBoostClassifier
# Assuming you have already trained and selected the best CatBoost model
# Let's call it catboost_best
y_CatBoost_best_pred = CatBoost_best.predict(X_test_encoded)
CatBoost_best_predictions = [round(value) for value in y_CatBoost_best_pred]


from sklearn.metrics import classification_report
# Evaluate the prediction of the 'LightGBM_best' classifier
print(" Classification Report of the Tuned Adaboost classifier: \n ", classification_report(y_test,
y_AdaBoost_best_pred))

from sklearn.metrics import classification_report
# Evaluate the prediction of the 'LightGBM_best' classifier
print(" Classification Report of the Tuned LightGBM classifier: \n ", classification_report(y_test,
y_LGBM_best_pred))


# Evaluate the prediction of the 'LightGBM_best' classifier
print(" Classification Report of the Tuned XGBoost classifier: \n ", classification_report(y_test,
y_XGBoost_best_pred))

# Evaluate the prediction of the 'LightGBM_best' classifier
print(" Classification Report of the Tuned Bagging classifier: \n ", classification_report(y_test,
y_Bagging_best_pred))

# Evaluate the prediction of the 'LightGBM_best' classifier
print(" Classification Report of the Tuned Random Forest classifier: \n ", classification_report(y_test,
y_rand_rf_pred))

# Evaluate the prediction of the 'LightGBM_best' classifier
print(" Classification Report of the Tuned CatBoost classifier: \n ", classification_report(y_test,
y_CatBoost_best_pred))
```

```
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(AdaBoost_best, X_balanced_test_encoded, y_test)
from sklearn.metrics import plot_confusion_matrix
plot_confusion_matrix(LGBM_best, X_test_encoded, y_test)
plot_confusion_matrix(best_rand_rf_model, X_test_encoded, y_test)
plot_confusion_matrix(Bagging_best, X_test_encoded, y_test)
plot_confusion_matrix(XGBoost_best, X_test_encoded, y_test)
plot_confusion_matrix(CatBoost_best, X_test_encoded, y_test)

from sklearn.metrics import recall_score, precision_score, f1_score, matthews_corrcoef, precision_recall_curve, auc
from imblearn.metrics import geometric_mean_score

# Assuming y_Adaboost_best_pred contains your model predictions

# Calculate Recall
recall = recall_score(y_test, y_AdaBoost_best_pred)
print('Recall: %.2f' % recall)

# Calculate Precision
precision = precision_score(y_test, y_AdaBoost_best_pred)
print('Precision: %.2f' % precision)

# Calculate F1-Score
F_Score = f1_score(y_test, y_AdaBoost_best_pred)
print('F1-Score: %.2f' % F_Score)

# Calculate Matthews Correlation Coefficient (MCC)
MCC = matthews_corrcoef(y_test, y_AdaBoost_best_pred)
print('MCC: %.2f' % MCC)

# Calculate G-Mean
G_Mean = geometric_mean_score(y_test, y_AdaBoost_best_pred, average='weighted')
print('G-Mean: %.2f' % G_Mean)

# Calculate precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_AdaBoost_best_pred)

# Calculate AUC-PR (Area Under the Precision-Recall Curve)
auc_score = auc(recall, precision)
print('AUC-PR: %.2f' % auc_score)


# calculate the precision-recall AUC

precision, recall, thresholds = precision_recall_curve(y_test, y_AdaBoost_best_pred)

auc_score = auc(recall, precision)

print('AUC-PR:', auc_score)

#Plot the Precision-Recall curve
```

```
yhat= AdaBoost.predict_proba (X_balanced_test_encoded)

yhat= yhat[:,1]

precision, recall, thresholds = precision_recall_curve (y_test, yhat)

#Plot the Precision-Recall (PR) Curve for the model

no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0,1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(recall, precision, marker='.', label="Tuned Adaboost AUC_PR ={:0.2f}".format(auc_score), lw = 3,
alpha=0.7)

#axis Labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()

#show the plot
pyplot.show()

importance_Adaboost = AdaBoost_best.feature_importances_

# Sort the feature importance in descending order
sorted_indices= np.argsort(importance_Adaboost)[::-1]

plt.title('Tuned Adaboost Feature Importance')
plt.bar(range(X_train_sampled.shape[1]), importance_Adaboost[sorted_indices], align='center')
plt.xticks(range(X_train_sampled.shape[1]), X_train_sampled.columns[sorted_indices], rotation=90)
plt.tight_layout()
plt.show()

!pip install imbalanced-learn
from sklearn.metrics import recall_score, precision_score, f1_score, matthews_corrcoef, precision_recall_curve, auc
from imblearn.metrics import geometric_mean_score

# Assuming y_LGBM_best_pred contains your model predictions

# Calculate Recall
recall = recall_score(y_test, y_LGBM_best_pred)
print('Recall: %.2f' % recall)

# Calculate Precision
precision = precision_score(y_test, y_LGBM_best_pred)
print('Precision: %.2f' % precision)

# Calculate F1-Score
F_Score = f1_score(y_test, y_LGBM_best_pred)
print('F1-Score: %.2f' % F_Score)

# Calculate Matthews Correlation Coefficient (MCC)
```

```
MCC = matthews_corrcoef(y_test, y_LGBM_best_pred)
print('MCC: %.2f' % MCC)

# Calculate G-Mean
G_Mean = geometric_mean_score(y_test, y_LGBM_best_pred, average='weighted')
print('G-Mean: %.2f' % G_Mean)

# Calculate precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_LGBM_best_pred)

# Calculate AUC-PR (Area Under the Precision-Recall Curve)
auc_score = auc(recall, precision)
print('AUC-PR: %.2f' % auc_score)

# calculate the precision-recall AUC

precision, recall, thresholds = precision_recall_curve(y_test, y_LGBM_best_pred)

auc_score = auc(recall, precision)

print('AUC-PR:', auc_score)

#Plot the Precision-Recall curve

yhat= Lgbm.predict_proba (X_balanced_test_encoded)

yhat= yhat[:,1]

precision, recall, thresholds = precision_recall_curve (y_test, yhat)

#Plot the Precision-Recall (PR) Curve for the model

no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0,1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(recall, precision, marker='.', label="Tuned LGBM AUC_PR ={:0.2f}".format(auc_score), lw = 3,
alpha=0.7)

#axis Labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()

#show the plot
pyplot.show()

importance_lgbm = LGBM_best.feature_importances_


# Sort the feature importance in descending order
sorted_indices= np.argsort(importance_lgbm)[::-1]

plt.title('Tuned LightGBM Feature Importance')
```

```
plt.bar(range(X_train_sampled.shape[1]), importance_lgbm[sorted_indices], align='center')
plt.xticks(range(X_train_sampled.shape[1]), X_train_sampled.columns[sorted_indices], rotation=90)
plt.tight_layout()
plt.show()

# RANDOM FOREST

from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
import pandas as pd

# Assuming X_balanced_train_encoded and y_train are your training data
# Assuming X_balanced_test_encoded is your test data
# Assuming you've already defined and fitted the RandomForestClassifier (rf) on your training data

# Fit the RandomForestClassifier on your training data
rf.fit(X_test_encoded, y_test)

# calculate the precision-recall AUC

precision, recall, thresholds = precision_recall_curve(y_test, y_rand_rf_pred)

auc_score = auc(recall, precision)

print('AUC-PR:', auc_score)

#Plot the Precision-Recall curve

yhat= rf.predict_proba (X_test_encoded)

yhat= yhat[:,1]

precision, recall, threshold = precision_recall_curve (y_test, yhat)
threshold = 0.6

from sklearn.metrics import recall_score, precision_score, f1_score, matthews_corrcoef, precision_recall_curve, auc
from imblearn.metrics import geometric_mean_score

# Assuming y_rand_rf_pred contains your Random Forest model predictions

# Calculate Recall
recall = recall_score(y_test, y_rand_rf_pred)
print('Recall: %.2f' % recall)

# Calculate Precision
precision = precision_score(y_test, y_rand_rf_pred)
print('Precision: %.2f' % precision)

# Calculate F1-Score
F_Score = f1_score(y_test, y_rand_rf_pred)
print('F1-Score: %.2f' % F_Score)
```

```
# Calculate Matthews Correlation Coefficient (MCC)
MCC = matthews_corrcoef(y_test, y_rand_rf_pred)
print('MCC: %.2f' % MCC)

# Calculate G-Mean
G_Mean = geometric_mean_score(y_test, y_rand_rf_pred, average='weighted')
print('G-Mean: %.2f' % G_Mean)

# Calculate precision-recall curve
precision, recall, _ = precision_recall_curve(y_test, y_rand_rf_pred)

# Calculate AUC-PR (Area Under the Precision-Recall Curve)
auc_score = auc(recall, precision)
print('AUC-PR: %.2f' % auc_score)

#Plot the Precision-Recall (PR) Curve for the model

no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0,1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(recall, precision, marker='.', label="Tuned RF AUC_PR ={:0.4f}".format(auc_score), lw = 3, alpha=0.8)

#axis Labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()

#show the plot
pyplot.show()

importance_rf =rf.feature_importances_

# Sort the feature importance in descending order
sorted_indices= np.argsort(importance_rf)[::-1]

plt.title('Tuned Random Forest Feature Importance')
plt.bar(range(X_train_sampled.shape[1]), importance_rf[sorted_indices], align='center')
plt.xticks(range(X_train_sampled.shape[1]), X_train_sampled.columns[sorted_indices], rotation=90)
plt.tight_layout()
plt.show()
from sklearn.metrics import recall_score, precision_score, f1_score, matthews_corrcoef, precision_recall_curve, auc
from imblearn.metrics import geometric_mean_score

# Assuming y_XGBoost_best_pred contains your XGBoost model predictions

# Calculate Recall
recall_xgb = recall_score(y_test, y_XGBoost_best_pred)
print('Recall: %.2f' % recall_xgb)

# Calculate Precision
precision_xgb = precision_score(y_test, y_XGBoost_best_pred)
print('Precision: %.2f' % precision_xgb)
```

```
# Calculate F1-Score
f1_xgb = f1_score(y_test, y_XGBoost_best_pred)
print('F1-Score: %.2f' % f1_xgb)

# Calculate Matthews Correlation Coefficient (MCC)
mcc_xgb = matthews_corrcoef(y_test, y_XGBoost_best_pred)
print('MCC: %.2f' % mcc_xgb)

# Calculate G-Mean
g_mean_xgb = geometric_mean_score(y_test, y_XGBoost_best_pred, average='weighted')
print('G-Mean: %.2f' % g_mean_xgb)

# Calculate precision-recall curve
precision_xgb, recall_xgb, _ = precision_recall_curve(y_test, y_XGBoost_best_pred)

# Calculate AUC-PR (Area Under the Precision-Recall Curve)
auc_score_xgb = auc(recall_xgb, precision_xgb)
print('AUC-PR: %.2f' % auc_score_xgb)

# calculate the precision-recall AUC

precision, recall, thresholds = precision_recall_curve(y_test, y_XGBoost_best_pred)
auc_score = auc(recall, precision)
print('AUC-PR:', auc_score)

#Plot the Precision-Recall curve
yhat= XGBoost.predict_proba (X_test_encoded)
yhat= yhat[:,1]
precision, recall, threshold = precision_recall_curve (y_test, yhat)
threshold = 0.6

#Plot the Precision-Recall (PR) Curve for the model

no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0,1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(recall, precision, marker='.', label="Tuned XGBoost AUC_PR ={:0.4f}".format(auc_score), lw = 3,
alpha=0.8)

#axis Labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()

#show the plot
pyplot.show()

importance_XGBoost =XGBoost_best.feature_importances_

# Sort the feature importance in descending order
sorted_indices= np.argsort(importance_XGBoost)[::-1]
```

```
plt.title('Tuned XGBoost Feature Importance')
plt.bar(range(X_train_sampled.shape[1]), importance_rf[sorted_indices], align='center')
plt.xticks(range(X_train_sampled.shape[1]), X_train_sampled.columns[sorted_indices], rotation=90)
plt.tight_layout()
plt.show()

from sklearn.metrics import recall_score, precision_score, f1_score, matthews_corrcoef, precision_recall_curve, auc
from imblearn.metrics import geometric_mean_score

# Assuming y_Bagging_best_pred contains your Bagging model predictions

# Calculate Recall
recall_bagging = recall_score(y_test, y_Bagging_best_pred)
print('Recall: %.2f' % recall_bagging)

# Calculate Precision
precision_bagging = precision_score(y_test, y_Bagging_best_pred)
print('Precision: %.2f' % precision_bagging)

# Calculate F1-Score
f1_bagging = f1_score(y_test, y_Bagging_best_pred)
print('F1-Score: %.2f' % f1_bagging)

# Calculate Matthews Correlation Coefficient (MCC)
mcc_bagging = matthews_corrcoef(y_test, y_Bagging_best_pred)
print('MCC: %.2f' % mcc_bagging)

# Calculate G-Mean
g_mean_bagging = geometric_mean_score(y_test, y_Bagging_best_pred, average='weighted')
print('G-Mean: %.2f' % g_mean_bagging)

# Calculate precision-recall curve
precision_bagging, recall_bagging, _ = precision_recall_curve(y_test, y_Bagging_best_pred)

# Calculate AUC-PR (Area Under the Precision-Recall Curve)
auc_score_bagging = auc(recall_bagging, precision_bagging)
print('AUC-PR: %.2f' % auc_score_bagging)

# calculate the precision-recall AUC
precision, recall, thresholds = precision_recall_curve(y_test, y_Bagging_best_pred)
auc_score = auc(recall, precision)
print('AUC-PR:', auc_score)

#Plot the Precision-Recall curve
yhat= bagging.predict_proba (X_test_encoded)
yhat= yhat[:,1]
precision, recall, threshold = precision_recall_curve (y_test, yhat)
threshold = 0.6

#Plot the Precision-Recall (PR) Curve for the model

no_skill = len(y_test[y_test==1]) / len(y_test)
```

```
pyplot.plot([0,1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(recall, precision, marker='.', label="Tuned Bagging AUC_PR ={:0.4f}".format(auc_score), lw = 3,
alpha=0.8)

#axis Labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()

#show the plot
pyplot.show()

from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
importance_bagging =Bagging_best.feature_importances_

# Sort the feature importance in descending order
sorted_indices= np.argsort(importance_bagging)[::-1]

plt.title('Tuned Bagging Feature Importance')
plt.bar(range(X_train_sampled.shape[1]), importance_rf[sorted_indices], align='center')
plt.xticks(range(X_train_sampled.shape[1]), X_train_sampled.columns[sorted_indices], rotation=90)
plt.tight_layout()
plt.show()

from sklearn.metrics import recall_score, precision_score, f1_score, matthews_corrcoef, precision_recall_curve, auc
from imblearn.metrics import geometric_mean_score

# Assuming y_CatBoost_best_pred contains your CatBoost model predictions

# Calculate Recall
recall_catboost = recall_score(y_test, y_CatBoost_best_pred)
print('Recall: %.2f' % recall_catboost)

# Calculate Precision
precision_catboost = precision_score(y_test, y_CatBoost_best_pred)
print('Precision: %.2f' % precision_catboost)

# Calculate F1-Score
f1_catboost = f1_score(y_test, y_CatBoost_best_pred)
print('F1-Score: %.2f' % f1_catboost)

# Calculate Matthews Correlation Coefficient (MCC)
mcc_catboost = matthews_corrcoef(y_test, y_CatBoost_best_pred)
print('MCC: %.2f' % mcc_catboost)

# Calculate G-Mean
g_mean_catboost = geometric_mean_score(y_test, y_CatBoost_best_pred, average='weighted')
print('G-Mean: %.2f' % g_mean_catboost)

# Calculate precision-recall curve
precision_catboost, recall_catboost, _ = precision_recall_curve(y_test, y_CatBoost_best_pred)
```

```
# Calculate AUC-PR (Area Under the Precision-Recall Curve)
auc_score_catboost = auc(recall_catboost, precision_catboost)
print('AUC-PR: %.2f' % auc_score_catboost)


# calculate the precision-recall AUC
precision, recall, thresholds = precision_recall_curve(y_test, y_CatBoost_best_pred)
auc_score = auc(recall, precision)
print('AUC-PR:', auc_score)

#Plot the Precision-Recall curve
yhat= CatBoost.predict_proba (X_test_encoded)
yhat= yhat[:,1]
precision, recall, threshold = precision_recall_curve (y_test, yhat)
threshold = 0.6

#Plot the Precision-Recall (PR) Curve for the model

no_skill = len(y_test[y_test==1]) / len(y_test)
pyplot.plot([0,1], [no_skill, no_skill], linestyle='--', label='No Skill')
pyplot.plot(recall, precision, marker='.', label="Tuned CatBoost AUC_PR ={:0.4f}".format(auc_score), lw = 3,
alpha=0.8)

#axis Labels
pyplot.xlabel('Recall')
pyplot.ylabel('Precision')
pyplot.legend()

#show the plot
pyplot.show()

importance_CatBoost =CatBoost_best.feature_importances_

# Sort the feature importance in descending order
sorted_indices= np.argsort(importance_CatBoost)[::-1]

plt.title('Tuned CatBoost Feature Importance')
plt.bar(range(X_train_sampled.shape[1]), importance_rf[sorted_indices], align='center')
plt.xticks(range(X_train_sampled.shape[1]), X_train_sampled.columns[sorted_indices], rotation=90)
plt.tight_layout()
plt.show()
```

Appendix 2- Visualization

21]:
```
sns.countplot(data=fraudulent, x='transaction_hour').set(title='Volume of Fraudulent Transactions by Hour')
plt.show()
#Plot the Number of Fraudulent Transactions per hour
```

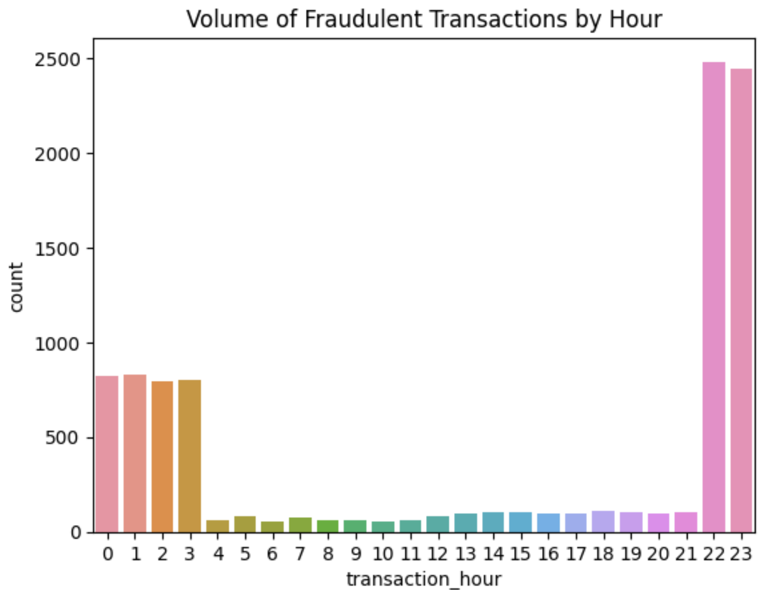

Figure 25: Number of Fraud Transaction Based on Hour

Appendix 3- Dissertation Checklist Sheet

Name: Moon Karmakar
Date Submitted: 7th September 2023
Signature (Digital): Moon Karmakar
I confirm that my dissertation contains the following prescribed elements:
✓ My dissertation portfolio meets the style requirements set out in the MSc Business Analytics Portfolio Dissertation Handbook including a word count on the front page of each element.
✓ I have reviewed the Turnitin similarity report prior to submission.
✓My dissertation title captures succinctly the focus of my dissertation
✓ My title page is formatted as prescribed in the MSc Business Analytics Portfolio Dissertation Handbook
✓ The abstract provides a clear and succinct overview of my study
✓ Each element contains a Table of Contents, and List of Figures and Tables (where appropriate)
✓ My dissertation contains a statement of acknowledgement (optional)
✓ The Introduction section of the research report, at a minimum, covers each of the following issues:
- Background to/context of the project
- Research question(s), aim(s) and objectives
- Why the project is necessary/important
- A summary of the Methodology
- Outline of the key findings
- Overview of chapter structure of the remainder of dissertation
✓ The Background section of the research report, at a minimum, covers each of the following issues:
- Synthesizes the key technical literature relating to the topic
- Synthesizes the key theoretical literature relating to the topic

✓The methodology section of the research report:
- Contains justification for the tools and method(s) selected
- Details the procedures adopted (e.g. the data source/acquisition, data processing, procedures for maximizing rigor and robustness, methods of data analysis etc.) - Contains ethical considerations and decisions
✓ The findings section of the research report reports the results in detail and provides possible explanations for the various findings
✓ The discussion section of the research report makes appropriate linkages between the findings and the literature reviewed
✓ The conclusions section of the research report includes:
- Conclusions about each research question and/or hypothesis
- General conclusions about the research problem
- Implications for theory, for policy and/or management practice
- Limitations of the research
- Suggestions for practice and future research
✓ The technical report, log book, and reflective discussion have each been included.

✓ The reference list is in alphabetical order and follows the Harvard system
✓I have signed and dated the Candidate Declaration