Credit Card Fraud Detection: A Hands-On Project

Introduction

🏫 *Author: Nhi Yen*

💡 *I write about Machine Learning on Medium || Github || Kaggle || Linkedin. If you found this article interesting, your support by giving me ☆ will help me spread the knowledge to others.*

Credit card fraud is a major concern for banks and financial institutions. Fraudsters use various techniques to steal credit card information and make unauthorized transactions. In this project, we will explore a dataset containing credit card transactions and build models to predict fraudulent transactions.

We will use the Kaggle dataset Credit Card Fraud Detection which contains credit card transactions made by European cardholders. The dataset consists of 284,807 transactions, out of which 492 are fraudulent. The data contains only numerical input variables which are a result of Principal Component Analysis (PCA) transformations due to confidentiality issues. The features include 'Time', 'Amount', and 'V1' through 'V28', as well as the 'Class' variable, which is the target variable indicating whether the transaction is fraudulent (1) or not (0).

In this project, we will start with exploratory data analysis (EDA) to get a better understanding of the data. Next, we will perform data processing and modeling, where we will build several classification models to predict fraudulent transactions. We will also address the issue of imbalanced classes by using undersampling. Finally, we will evaluate the performance of the models and choose the best one based on various evaluation metrics such as precision, recall, F1-score, and accuracy.

READ MORE

```python
# Import necessary libraries
%matplotlib inline
import scipy.stats as stats
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from imblearn.under_sampling import RandomUnderSampler
from sklearn import linear_model
from sklearn.model_selection import GridSearchCV
from sklearn import svm
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.dummy import DummyClassifier
import sklearn.metrics as metrics
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score, recall_score,
precision_recall_curve, f1_score, fbeta_score, accuracy_score

# Set plot style
plt.style.use('ggplot')

# Turn off warnings
import warnings
warnings.filterwarnings('ignore')

# Set font size for all plots
plt.rcParams['font.size'] = 12
plt.rcParams['axes.titlesize'] = 18
plt.rcParams['axes.labelsize'] = 12
plt.rcParams['xtick.labelsize'] = 10
plt.rcParams['ytick.labelsize'] = 10
plt.rcParams['legend.fontsize'] = 10
```

```
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
```

```
# Loading data
df = pd.read_csv('/kaggle/input/creditcardfraud/creditcard.csv')
```

1. Exploratory Data Analysis

```
# Printing random sample of 10 rows to check data loading
df.sample(10)
```

```
            Time        V1        V2        V3        V4        V5
V6  \
104291   69003.0 -1.769825  1.229880 -0.359125  0.502417  0.914113
4.756624
47928    43420.0 -0.335464  0.944520  1.444409  0.509949  0.225335 -
0.664476
117808   74819.0 -0.825715  0.459793  1.729183 -1.336597 -0.096040 -
0.437804
221169  142462.0  2.220891 -1.735027 -0.451585 -1.621137 -1.729430 -
0.047030
139603   83247.0 -1.165577  1.275190  1.240048  2.433737  0.703401
2.249204
106025   69786.0 -0.535842  0.590314  0.929668 -1.628139 -0.376264 -
1.389383
267167  162653.0 -2.029764  0.068646  0.424219 -0.696047  0.504933 -
0.249883
37175    38836.0 -0.879685  1.420351  1.189178 -0.014796 -0.015929 -
```

```
0.729290
245900  152960.0  2.176772 -0.524938 -1.415405 -0.494516  0.098441 -
0.001469
260785  159724.0  2.081039 -1.065925 -2.139389 -1.208732  1.575862
3.672579

              V7        V8        V9  ...       V21       V22
V23  \
104291 -0.863296  1.896219  0.602818  ... -0.384586 -0.799696 -
0.006240
47928   0.542422 -0.037535 -0.696113  ... -0.094761 -0.242987
0.132006
117808  0.401778  0.274934 -0.165275  ...  0.024469 -0.002620
0.060874
221169 -1.827377  0.215458 -0.775930  ...  0.127920  0.775034
0.197296
139603 -0.330115  1.250532 -1.613043  ... -0.010312 -0.028097
0.286860
106025  0.549499 -0.305641  0.858874  ... -0.028886 -0.147118
0.058098
267167 -0.809185  1.078749 -0.110982  ...  0.020477 -0.416974 -
0.434087
37175   0.645677  0.049051 -0.074700  ... -0.273513 -0.540389
0.078612
245900 -0.330780 -0.105914 -0.887651  ... -0.630516 -1.246660
0.348347
260785 -1.238063  0.892891 -0.370079  ... -0.345271 -0.516392
0.310172

             V24       V25       V26       V27       V28  Amount
Class
104291  1.007295 -0.129352 -0.531629 -0.703931 -0.466469   83.83
0
47928   0.377992 -0.828644  0.095587  0.154216  0.177857    1.29
0
117808  0.243652 -0.486581  0.720185  0.252660  0.165237   28.62
0
221169  0.723855 -0.288544 -0.009355  0.022548 -0.048482   16.88
0
139603 -1.430137 -0.706012  0.091018  0.201047  0.089597    9.82
0
106025  0.388111 -0.566760 -0.162962 -0.267293  0.173787    1.22
0
267167 -1.062439 -0.286624 -0.585759 -0.102585 -0.253339    1.00
0
37175   0.348010 -0.163703  0.089859  0.472949  0.232589    8.94
0
245900  0.058178 -0.268778  0.001470 -0.035416 -0.060095    2.78
0
260785  0.685259 -0.279842  0.282346  0.007168 -0.050476   26.00
```

```
0

[10 rows x 31 columns]
```

✅ We can only work with three non-transformed variables which are **Time, Amount, and Class** *(where Class takes values of 1 for fraud and 0 for not fraud)*.

```python
# Printing data overview
df.info()
```

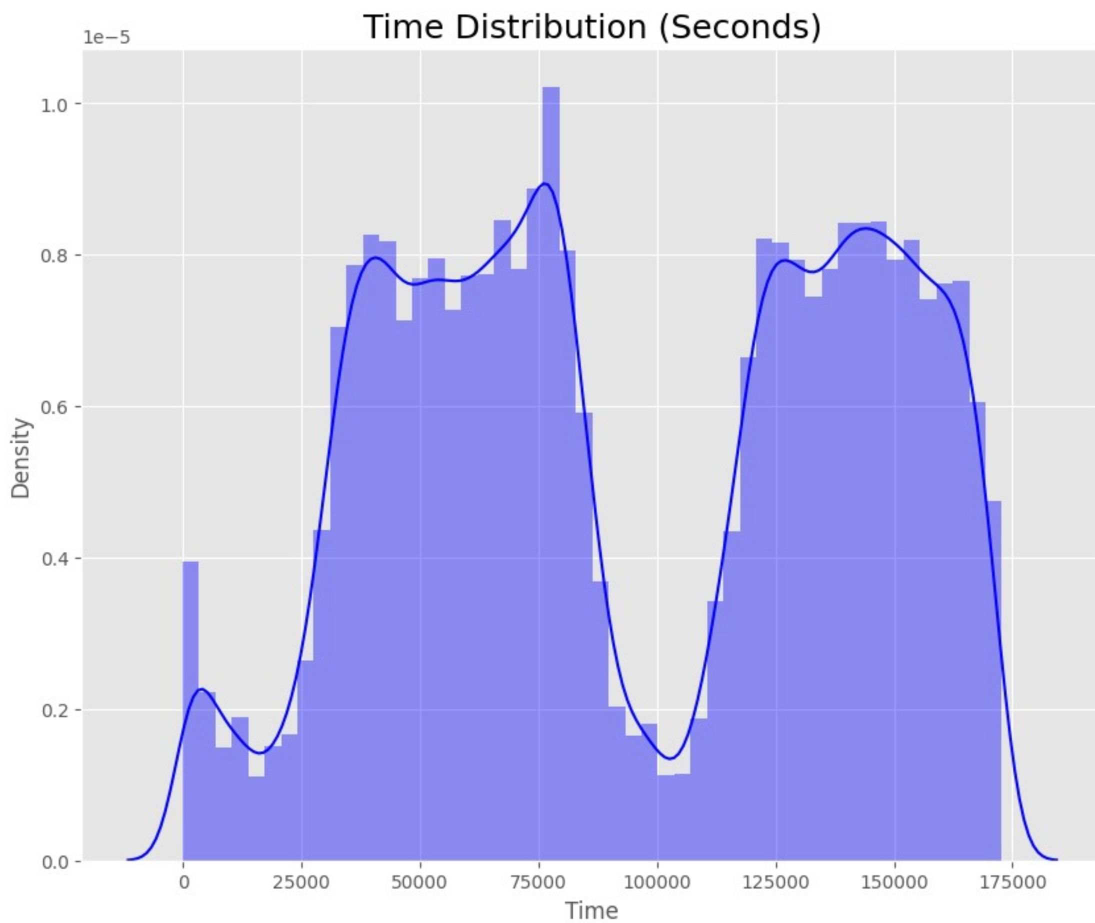
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype  
---  ------  --------------   -----  
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64  
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

```python
# Printing numerical summary for Time and Amount columns
df.loc[:, ['Time', 'Amount']].describe()
```

```
                Time         Amount
count  284807.000000  284807.000000
mean    94813.859575      88.349619
std     47488.145955     250.120109
min         0.000000       0.000000
25%     54201.500000       5.600000
50%     84692.000000      22.000000
75%    139320.500000      77.165000
max    172792.000000   25691.160000
```

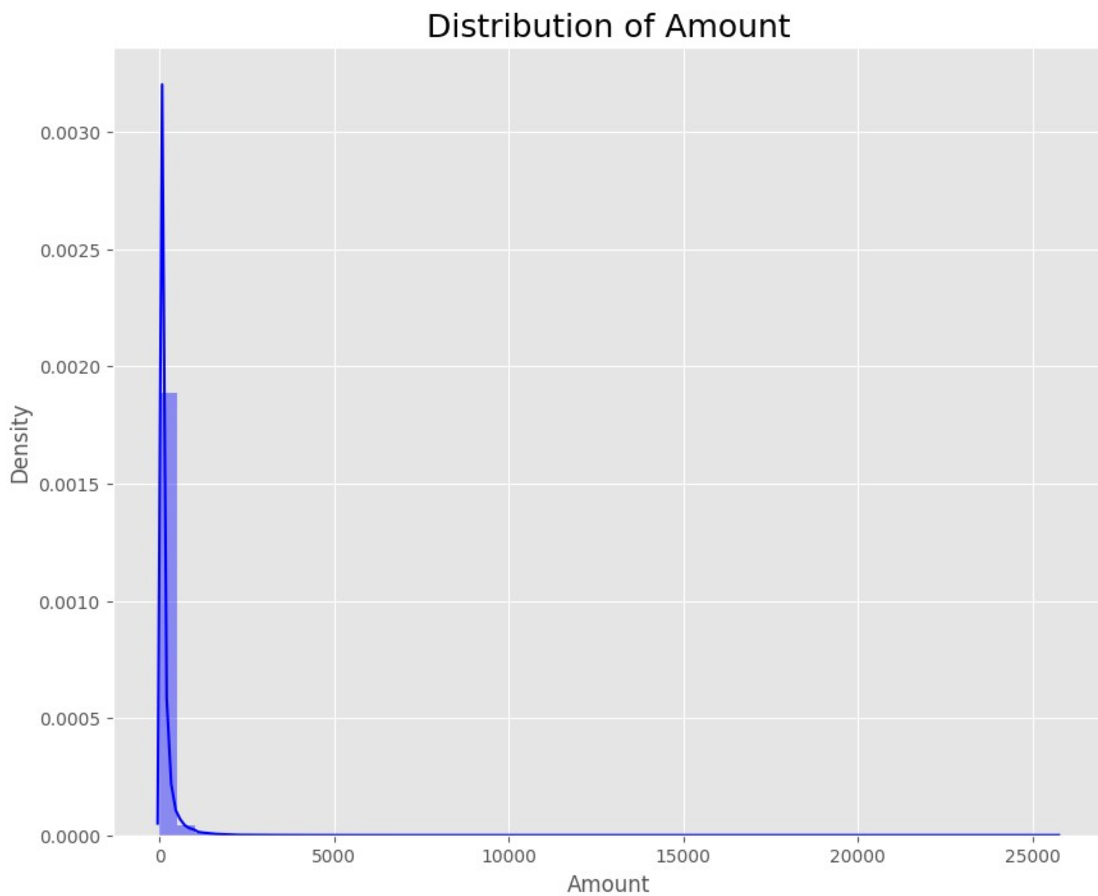
✅ From the plot, we can observe that the `Time` feature has a bimodal distribution with two peaks, indicating that there are two periods during the day when credit card transactions are more frequent. The first peak occurs at around 50,000 seconds (approximately 14 hours), while the second peak occurs at around 120,000 seconds (approximately 33 hours). This suggests that there may be a pattern in the timing of credit card transactions that could be useful for fraud detection.

```
# Plotting distribution of Time feature
plt.figure(figsize=(10,8), )
plt.title('Time Distribution (Seconds)')
sns.distplot(df['Time'], color='blue')

# Save the plot as PNG file
plt.savefig('time_distribution.png');
```



✅ From the plot, we can observe that the distribution of the Amount feature is highly skewed to the right, with a long tail to the right. This indicates that the majority of the transactions have low amounts, while a few transactions have extremely high amounts. As a result, this suggests that the dataset contains some outliers in terms of transaction amounts. Therefore, when building a model for fraud detection, it may be necessary to handle outliers in the `Amount` feature, for instance, by using a log transformation or robust statistical methods.

```
# Plotting distribution of Amount feature
plt.figure(figsize=(10,8))
plt.title('Distribution of Amount')
sns.distplot(df['Amount'], color='blue')

# Save the plot as PNG file
plt.savefig('amount_distribution.png');
```



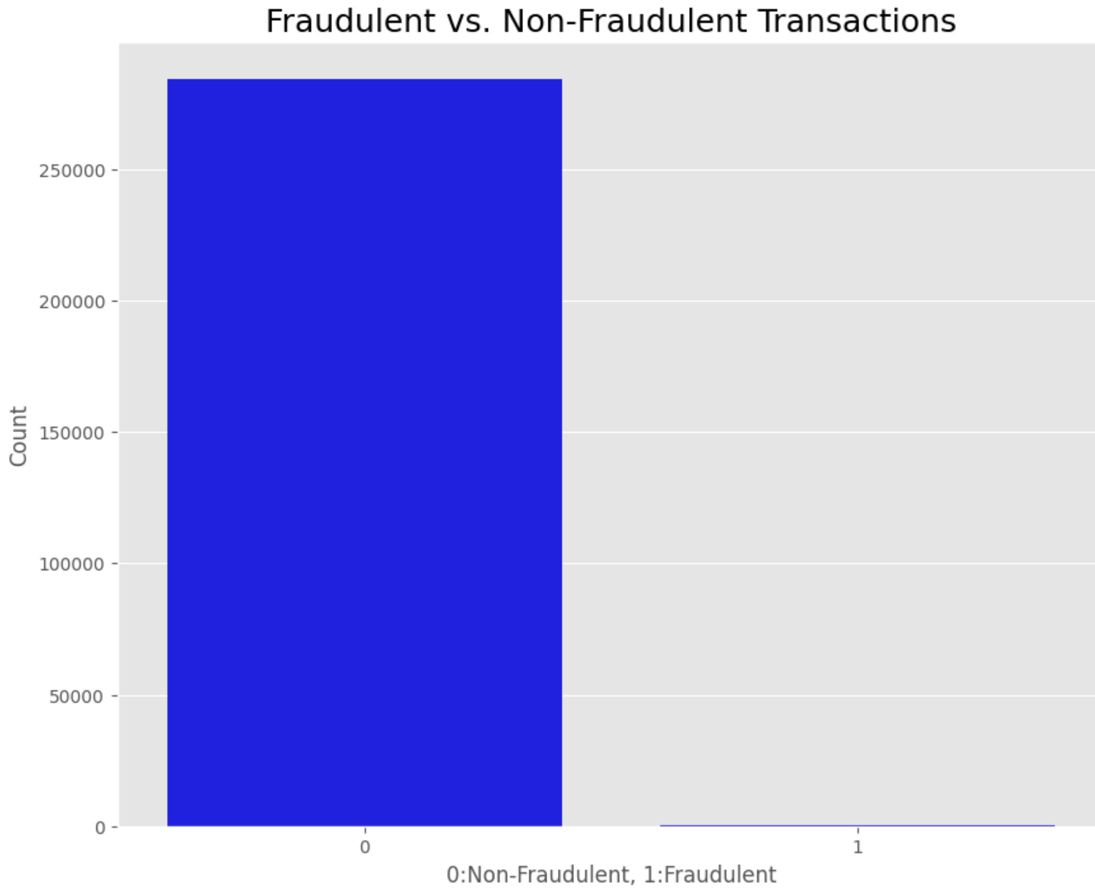

```
# Counting number of fraud vs non-fraud transactions and displaying
them with their ratio
fraud = df['Class'].value_counts()[1]
nonfraud = df['Class'].value_counts()[0]
print(f'Fraudulent: {fraud}, Non-fraudulent: {nonfraud}')
print(f'Ratio of fraud to non-fraud: {fraud}/{nonfraud}
({fraud/nonfraud*100:.3f}%)')
```

```
Fraudulent: 492, Non-fraudulent: 284315
Ratio of fraud to non-fraud: 492/284315 (0.173%)
```

✅ From the plot, we can observe that the dataset is highly imbalanced, with a vast majority of transactions being non-fraudulent (class 0) and a relatively small number of transactions being fraudulent (class 1). This indicates that the dataset has a class imbalance problem, which may affect the performance of a model trained on this dataset. It may be necessary to use techniques such as oversampling, undersampling, or class weighting to handle the class imbalance problem when building a model for fraud detection.

```
# Plotting count of fraud vs non-fraud transactions in a bar chart
plt.figure(figsize=(10,8))
sns.barplot(x=df['Class'].value_counts().index,
y=df['Class'].value_counts(), color='blue')
plt.title('Fraudulent vs. Non-Fraudulent Transactions')
```

```
plt.ylabel('Count')
plt.xlabel('0:Non-Fraudulent, 1:Fraudulent')

# Save the plot as PNG file
plt.savefig('fraud_vs_nonfraud_transactions.png');
```



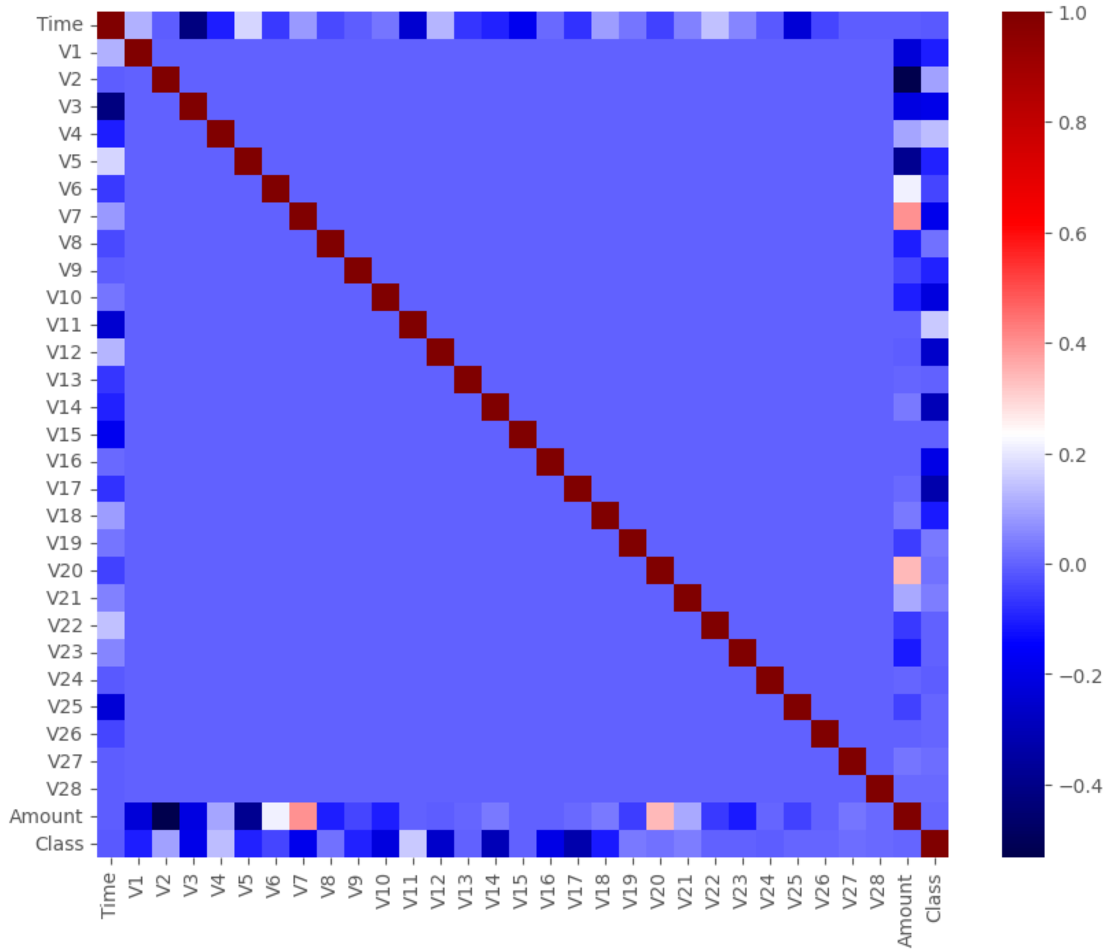
2. Data Processing

✅ From the heatmap, it can be observed that there are no strong positive or negative correlations between any pairs of variables in the dataset. The strongest correlations are found:

- Time and V3, with a correlation coefficient of -0.42
- Amount and V2, with a correlation coefficient of -0.53
- Amount and V4, with a correlation coefficient of 0.4.

Although these correlations are relatively high, the risk of multicollinearity is not expected to be significant. Overall, the heatmap suggests that there are no highly correlated variables that need to be removed before building a machine learning model.

```
# Plotting heatmap to find any high correlations between variables
plt.figure(figsize=(10,8))
sns.heatmap(data=df.corr(), cmap="seismic", annot=False)
# plt.show()

# Save the plot as PNG file
plt.savefig('corr_heatmap.png');
```



3. Modeling

✅ The "Credit Card Fraud Detection" dataset has credit card transactions labeled as fraudulent or not. The dataset is imbalanced, so it needs a model that can accurately detect fraudulent transactions without wrongly flagging non-fraudulent transactions.

✅ To help with classification problems, **StandardScaler** standardizes data by giving it a mean of 0 and a standard deviation of 1, which results in a normal distribution. This technique works well when dealing with a wide range of amounts and time. To scale the data, the training set is used to initialize the fit, and the train, validation, and test sets are then scaled before running them into the models.

✅ The dataset was divided into 60% for training, 20% for validation, and 20% for testing. To balance the imbalanced dataset, **Random Undersampling** was used to match the number of fraudulent transactions. Logistic Regression and Random Forest models were used, and good results were produced.

✅ The commonly used models for the "Credit Card Fraud Detection" dataset are Logistic Regression, Naive Bayes, Random Forest, and Dummy Classifier.

- **Logistic Regression** is widely used for fraud detection because of its interpretability and ability to handle large datasets.
- **Naive Bayes** is commonly used for fraud detection because it can handle datasets with a large number of features and can provide fast predictions.
- **Random Forest** is commonly used for fraud detection because it can handle complex datasets and is less prone to overfitting.
- The **Dummy Classifier** is a simple algorithm used as a benchmark to compare the performance of other models.

```
# Drop the 'Class' column to prepare data for splitting
data = df.drop(columns=['Class'])

# Get the target variable
answer = df['Class']

# Split data into training, validation and test sets, ensuring the class distribution is maintained
X_trainval, X_test, y_trainval, y_test = train_test_split(data, answer
                                                                                                    ,
test_size=0.2
                                                                                                    ,
stratify=df['Class']
                                                                                                    ,
random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_trainval,
y_trainval
                                                        , test_size=0.25
                                                        ,
stratify=y_trainval
                                                        , random_state=42)

# Initialize the StandardScaler object and fit it to the training data
scaler = StandardScaler()
scaler.fit(X_train)

# Scale the training, validation, and test sets using the scaler
X_train_std = scaler.transform(X_train)
X_val_std = scaler.transform(X_val)
X_test_std = scaler.transform(X_test)
```

✅ Undersampling will be utilized to address the issue of imbalanced classes.

```
# Undersampling will be utilized to address the issue of imbalanced
classes.

# Instantiate RandomUnderSampler
rus = RandomUnderSampler(random_state=42)

# Undersample the training set
X_train_under, y_train_under = rus.fit_resample(X_train_std, y_train)

# Undersample the validation set
X_val_under, y_val_under = rus.fit_resample(X_val_std, y_val)
```

3.1. Logistic Regression

```
# Logistic Regression
# Run CV with 5 folds (logit)
penalty = ['l2']
C = np.logspace(0, 4, 10, 100, 1000)
param_grid = dict(C=C, penalty=penalty)

logistic = linear_model.LogisticRegression(solver='lbfgs',
max_iter=10000)
logistic_grid = GridSearchCV(logistic, param_grid, cv=5,
scoring='roc_auc', verbose=10, n_jobs=-1)
logistic_grid.fit(X_train_under, y_train_under)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits

/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"
/opt/conda/lib/python3.10/site-packages/scipy/__init__.py:146:
UserWarning: A NumPy version >=1.16.5 and <1.23.0 is required for this
version of SciPy (detected version 1.23.5
  warnings.warn(f"A NumPy version >={np_minversion} and
<{np_maxversion}"

[CV 1/5; 1/10] START C=1.0,
penalty=l2..........................................
```

```
[CV 1/5; 1/10] END ..........C=1.0, penalty=l2;, score=0.994 total
time=   0.0s
[CV 2/5; 1/10] START C=1.0,
penalty=l2..........................................
[CV 2/5; 1/10] END ..........C=1.0, penalty=l2;, score=0.983 total
time=   0.0s
[CV 3/5; 1/10] START C=1.0,
penalty=l2..........................................
[CV 5/5; 1/10] START C=1.0,
penalty=l2..........................................
[CV 4/5; 1/10] START C=1.0,
penalty=l2..........................................
[CV 3/5; 1/10] END ..........C=1.0, penalty=l2;, score=1.000 total
time=   0.0s
[CV 5/5; 1/10] END ..........C=1.0, penalty=l2;, score=0.988 total
time=   0.0s
[CV 4/5; 1/10] END ..........C=1.0, penalty=l2;, score=0.948 total
time=   0.0s
[CV 1/5; 2/10] START C=21.544346900318832,
penalty=l2..........................
[CV 2/5; 2/10] START C=21.544346900318832,
penalty=l2..........................
[CV 1/5; 2/10] END C=21.544346900318832, penalty=l2;, score=0.996
total time=   0.0s
[CV 3/5; 2/10] START C=21.544346900318832,
penalty=l2..........................
[CV 2/5; 2/10] END C=21.544346900318832, penalty=l2;, score=0.982
total time=   0.0s
[CV 4/5; 2/10] START C=21.544346900318832,
penalty=l2..........................
[CV 5/5; 2/10] START C=21.544346900318832,
penalty=l2..........................
[CV 1/5; 3/10] START C=464.15888336127773,
penalty=l2..........................
[CV 3/5; 2/10] END C=21.544346900318832, penalty=l2;, score=0.999
total time=   0.0s
[CV 2/5; 3/10] START C=464.15888336127773,
penalty=l2..........................
[CV 4/5; 2/10] END C=21.544346900318832, penalty=l2;, score=0.949
total time=   0.0s
[CV 5/5; 2/10] END C=21.544346900318832, penalty=l2;, score=0.989
total time=   0.0s
[CV 3/5; 3/10] START C=464.15888336127773,
penalty=l2..........................
[CV 4/5; 3/10] START C=464.15888336127773,
penalty=l2..........................
[CV 1/5; 3/10] END C=464.15888336127773, penalty=l2;, score=0.997
total time=   0.1s
[CV 5/5; 3/10] START C=464.15888336127773,
penalty=l2..........................
```

```
[CV 4/5; 3/10] END C=464.15888336127773, penalty=l2;, score=0.955
total time=   0.1s
[CV 1/5; 4/10] START C=9999.999999999995,
penalty=l2...........................
[CV 2/5; 3/10] END C=464.15888336127773, penalty=l2;, score=0.977
total time=   0.1s
[CV 2/5; 4/10] START C=9999.999999999995,
penalty=l2...........................
[CV 3/5; 3/10] END C=464.15888336127773, penalty=l2;, score=0.997
total time=   0.1s
[CV 3/5; 4/10] START C=9999.999999999995,
penalty=l2...........................
[CV 5/5; 3/10] END C=464.15888336127773, penalty=l2;, score=0.988
total time=   0.1s
[CV 4/5; 4/10] START C=9999.999999999995,
penalty=l2...........................
[CV 1/5; 4/10] END C=9999.999999999995, penalty=l2;, score=0.999 total
time=   0.2s
[CV 5/5; 4/10] START C=9999.999999999995,
penalty=l2...........................
[CV 3/5; 4/10] END C=9999.999999999995, penalty=l2;, score=0.998 total
time=   0.2s
[CV 1/5; 5/10] START C=215443.46900318822,
penalty=l2..........................
[CV 2/5; 4/10] END C=9999.999999999995, penalty=l2;, score=0.976 total
time=   0.3s
[CV 2/5; 5/10] START C=215443.46900318822,
penalty=l2..........................
[CV 4/5; 4/10] END C=9999.999999999995, penalty=l2;, score=0.958 total
time=   0.3s
[CV 5/5; 4/10] END C=9999.999999999995, penalty=l2;, score=0.981 total
time=   0.2s
[CV 3/5; 5/10] START C=215443.46900318822,
penalty=l2..........................
[CV 4/5; 5/10] START C=215443.46900318822,
penalty=l2..........................
[CV 1/5; 5/10] END C=215443.46900318822, penalty=l2;, score=0.999
total time=   0.2s
[CV 5/5; 5/10] START C=215443.46900318822,
penalty=l2..........................
[CV 2/5; 5/10] END C=215443.46900318822, penalty=l2;, score=0.976
total time=   0.1s
[CV 1/5; 6/10] START C=4641588.833612782,
penalty=l2...........................
[CV 3/5; 5/10] END C=215443.46900318822, penalty=l2;, score=0.986
total time=   0.2s
[CV 2/5; 6/10] START C=4641588.833612782,
penalty=l2...........................
[CV 1/5; 6/10] END C=4641588.833612782, penalty=l2;, score=0.999 total
time=   0.2s
```

```
[CV 3/5; 6/10] START C=4641588.833612782,
penalty=l2............................
[CV 5/5; 5/10] END C=215443.46900318822, penalty=l2;, score=0.983
total time=   0.2s
[CV 4/5; 6/10] START C=4641588.833612782,
penalty=l2............................
[CV 2/5; 6/10] END C=4641588.833612782, penalty=l2;, score=0.977 total
time=   0.1s
[CV 5/5; 6/10] START C=4641588.833612782,
penalty=l2............................
[CV 3/5; 6/10] END C=4641588.833612782, penalty=l2;, score=0.986 total
time=   0.1s
[CV 1/5; 7/10] START C=99999999.9999999,
penalty=l2.............................
[CV 4/5; 5/10] END C=215443.46900318822, penalty=l2;, score=0.956
total time=   0.5s
[CV 2/5; 7/10] START C=99999999.9999999,
penalty=l2.............................
[CV 1/5; 7/10] END C=99999999.9999999, penalty=l2;, score=0.999 total
time=   0.1s
[CV 3/5; 7/10] START C=99999999.9999999,
penalty=l2.............................
[CV 5/5; 6/10] END C=4641588.833612782, penalty=l2;, score=0.986 total
time=   0.4s
[CV 4/5; 7/10] START C=99999999.9999999,
penalty=l2.............................
[CV 2/5; 7/10] END C=99999999.9999999, penalty=l2;, score=0.977 total
time=   0.3s
[CV 5/5; 7/10] START C=99999999.9999999,
penalty=l2.............................
[CV 3/5; 7/10] END C=99999999.9999999, penalty=l2;, score=0.985 total
time=   0.3s
[CV 1/5; 8/10] START C=2154434690.031878,
penalty=l2............................
[CV 5/5; 7/10] END C=99999999.9999999, penalty=l2;, score=0.984 total
time=   0.1s
[CV 2/5; 8/10] START C=2154434690.031878,
penalty=l2............................
[CV 1/5; 8/10] END C=2154434690.031878, penalty=l2;, score=0.999 total
time=   0.2s
[CV 3/5; 8/10] START C=2154434690.031878,
penalty=l2............................
[CV 2/5; 8/10] END C=2154434690.031878, penalty=l2;, score=0.976 total
time=   0.2s
[CV 4/5; 8/10] START C=2154434690.031878,
penalty=l2............................
[CV 3/5; 8/10] END C=2154434690.031878, penalty=l2;, score=0.986 total
time=   0.2s
[CV 5/5; 8/10] START C=2154434690.031878,
penalty=l2............................
```

```
[CV 5/5; 8/10] END C=2154434690.031878, penalty=l2;, score=0.986 total
time=   0.3s
[CV 1/5; 9/10] START C=46415888336.12772,
penalty=l2............................
[CV 1/5; 9/10] END C=46415888336.12772, penalty=l2;, score=0.999 total
time=   0.1s
[CV 2/5; 9/10] START C=46415888336.12772,
penalty=l2............................
[CV 2/5; 9/10] END C=46415888336.12772, penalty=l2;, score=0.976 total
time=   0.2s
[CV 3/5; 9/10] START C=46415888336.12772,
penalty=l2............................
[CV 4/5; 7/10] END C=99999999.9999999, penalty=l2;, score=0.942 total
time=   1.1s
[CV 4/5; 9/10] START C=46415888336.12772,
penalty=l2............................
[CV 3/5; 9/10] END C=46415888336.12772, penalty=l2;, score=0.981 total
time=   0.2s
[CV 5/5; 9/10] START C=46415888336.12772,
penalty=l2............................
[CV 4/5; 6/10] END C=4641588.833612782, penalty=l2;, score=0.955 total
time=   2.1s
[CV 1/5; 10/10] START C=1000000000000.0,
penalty=l2............................
[CV 5/5; 9/10] END C=46415888336.12772, penalty=l2;, score=0.986 total
time=   0.3s
[CV 2/5; 10/10] START C=1000000000000.0,
penalty=l2............................
[CV 1/5; 10/10] END C=1000000000000.0, penalty=l2;, score=0.999 total
time=   0.2s
[CV 3/5; 10/10] START C=1000000000000.0,
penalty=l2............................
[CV 2/5; 10/10] END C=1000000000000.0, penalty=l2;, score=0.977 total
time=   0.2s
[CV 4/5; 10/10] START C=1000000000000.0,
penalty=l2............................
[CV 4/5; 9/10] END C=46415888336.12772, penalty=l2;, score=0.946 total
time=   0.7s
[CV 5/5; 10/10] START C=1000000000000.0,
penalty=l2............................
[CV 3/5; 10/10] END C=1000000000000.0, penalty=l2;, score=0.986 total
time=   0.2s
[CV 5/5; 10/10] END C=1000000000000.0, penalty=l2;, score=0.986 total
time=   0.1s
[CV 4/5; 8/10] END C=2154434690.031878, penalty=l2;, score=0.953 total
time=   1.8s
[CV 4/5; 10/10] END C=1000000000000.0, penalty=l2;, score=0.941 total
time=   1.5s
```

```
GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=10000),
n_jobs=-1,
             param_grid={'C': array([1.00000000e+00, 2.15443469e+01,
4.64158883e+02, 1.00000000e+04,
       2.15443469e+05, 4.64158883e+06, 1.00000000e+08, 2.15443469e+09,
       4.64158883e+10, 1.00000000e+12]),
                         'penalty': ['l2']},
             scoring='roc_auc', verbose=10)
```

Support Vector Machine (SVM)

```
# # Support Vector Machine (SVM)
# # Run CV with 5 folds (SVM)
# C = [1]
# gammas = [0.001, 0.1]
# param_grid = dict(C=C, gamma=gammas)

# svm1 = svm.SVC(kernel='rbf', probability=True)
# svm_grid = GridSearchCV(svm1, param_grid, cv=5, scoring='roc_auc',
verbose=10, n_jobs=-1)
# svm_grid.fit(X_train_under, y_train_under)
```

3.2. Naive Bayes

```
# Naive Bayes
# Fit a Naive Bayes Model
gnb = GaussianNB()
gnb_best = gnb.fit(X_train_under, y_train_under)
```

3.3. Random Forest

```
# Random Forest
# Run CV with 5 folds (Random Forest)
# Create the parameter grid based on the results of random search
param_grid = {
    'max_depth': [5, 10, 15],
    'max_features': ['sqrt'],
    'min_samples_leaf': [10, 20],
    'min_samples_split': [2, 5],
    'n_estimators': [500, 700]
}

rf = RandomForestClassifier()
rf_grid = GridSearchCV(rf, param_grid, cv=5, scoring='roc_auc',
verbose=10, n_jobs=-1)
rf_grid.fit(X_train_under,y_train_under)
```

```
Fitting 5 folds for each of 24 candidates, totalling 120 fits
[CV 1/5; 1/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 3/5; 1/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 4/5; 1/24] START max_depth=5, max_features=sqrt,
```

```
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 2/5; 1/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 1/5; 1/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.987 total time=   1.9s
[CV 5/5; 1/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 3/5; 1/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.999 total time=   1.9s
[CV 1/5; 2/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 4/5; 1/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.957 total time=   1.9s
[CV 2/5; 2/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 2/5; 1/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.976 total time=   1.9s
[CV 3/5; 2/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 5/5; 1/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.989 total time=   1.8s
[CV 4/5; 2/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 1/5; 2/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.988 total time=   2.5s
[CV 5/5; 2/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 2/5; 2/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.976 total time=   2.6s
[CV 1/5; 3/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 3/5; 2/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.999 total time=   2.6s
[CV 2/5; 3/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 4/5; 2/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.953 total time=   2.7s
[CV 3/5; 3/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 1/5; 3/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
```

```
score=0.989 total time=   2.2s
[CV 2/5; 3/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.978 total time=   2.2s
[CV 4/5; 3/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 5/5; 3/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 5/5; 2/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.987 total time=   2.9s
[CV 1/5; 4/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 4/5; 3/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.956 total time=   2.0s
[CV 5/5; 3/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.987 total time=   2.0s
[CV 2/5; 4/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 3/5; 4/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 3/5; 3/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.998 total time=   2.5s
[CV 4/5; 4/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 1/5; 4/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.988 total time=   3.2s
[CV 5/5; 4/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 3/5; 4/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.999 total time=   2.8s
[CV 1/5; 5/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 2/5; 4/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.978 total time=   3.2s
[CV 2/5; 5/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 4/5; 4/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.955 total time=   3.3s
[CV 3/5; 5/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 5/5; 4/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
```

```
score=0.988 total time=   3.0s
[CV 4/5; 5/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 1/5; 5/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.988 total time=   2.2s
[CV 5/5; 5/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 2/5; 5/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.980 total time=   2.0s
[CV 1/5; 6/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 3/5; 5/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.998 total time=   2.7s
[CV 2/5; 6/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 5/5; 5/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.986 total time=   2.4s
[CV 3/5; 6/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 4/5; 5/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.953 total time=   2.8s
[CV 4/5; 6/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 1/5; 6/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.988 total time=   3.2s
[CV 5/5; 6/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 2/5; 6/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.980 total time=   3.0s
[CV 1/5; 7/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 3/5; 6/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.998 total time=   2.5s
[CV 2/5; 7/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 4/5; 6/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.950 total time=   2.4s
[CV 3/5; 7/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 5/5; 6/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
```

```
score=0.986 total time=   2.4s
[CV 4/5; 7/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 1/5; 7/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.988 total time=   1.7s
[CV 5/5; 7/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 2/5; 7/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.981 total time=   1.7s
[CV 1/5; 8/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 3/5; 7/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.998 total time=   1.7s
[CV 2/5; 8/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 4/5; 7/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.950 total time=   1.7s
[CV 3/5; 8/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 5/5; 7/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.986 total time=   1.8s
[CV 4/5; 8/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 1/5; 8/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.989 total time=   2.4s
[CV 5/5; 8/24] START max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 2/5; 8/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.979 total time=   2.4s
[CV 1/5; 9/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 3/5; 8/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.998 total time=   2.4s
[CV 2/5; 9/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 4/5; 8/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.951 total time=   2.4s
[CV 3/5; 9/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 1/5; 9/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
```

```
score=0.990 total time=   1.9s
[CV 4/5; 9/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 5/5; 8/24] END max_depth=5, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.986 total time=   2.4s
[CV 5/5; 9/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 2/5; 9/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.980 total time=   1.9s
[CV 1/5; 10/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 3/5; 9/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.999 total time=   1.9s
[CV 2/5; 10/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 4/5; 9/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.959 total time=   1.8s
[CV 3/5; 10/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 5/5; 9/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.989 total time=   1.8s
[CV 4/5; 10/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 1/5; 10/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.989 total time=   2.5s
[CV 5/5; 10/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 2/5; 10/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.978 total time=   2.5s
[CV 1/5; 11/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 3/5; 10/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.999 total time=   2.5s
[CV 2/5; 11/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 4/5; 10/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.957 total time=   2.5s
[CV 3/5; 11/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 1/5; 11/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
```

```
score=0.989 total time=   1.8s
[CV 4/5; 11/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 5/5; 10/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.988 total time=   2.5s
[CV 5/5; 11/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 2/5; 11/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.978 total time=   1.8s
[CV 1/5; 12/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 3/5; 11/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.999 total time=   1.8s
[CV 2/5; 12/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 4/5; 11/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.956 total time=   1.8s
[CV 3/5; 12/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 5/5; 11/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.988 total time=   1.8s
[CV 4/5; 12/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 1/5; 12/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.989 total time=   2.6s
[CV 5/5; 12/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 2/5; 12/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.978 total time=   2.6s
[CV 1/5; 13/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 3/5; 12/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.999 total time=   2.6s
[CV 2/5; 13/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 4/5; 12/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.956 total time=   2.6s
[CV 3/5; 13/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 1/5; 13/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
```

```
score=0.988 total time=   1.7s
[CV 4/5; 13/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 5/5; 12/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.988 total time=   2.6s
[CV 5/5; 13/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 2/5; 13/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.977 total time=   1.7s
[CV 1/5; 14/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 3/5; 13/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.998 total time=   1.7s
[CV 2/5; 14/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 4/5; 13/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.950 total time=   1.7s
[CV 3/5; 14/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 5/5; 13/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.986 total time=   1.7s
[CV 4/5; 14/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 1/5; 14/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.987 total time=   2.4s
[CV 5/5; 14/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 2/5; 14/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.978 total time=   2.4s
[CV 1/5; 15/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 3/5; 14/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.999 total time=   2.4s
[CV 2/5; 15/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 4/5; 14/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.951 total time=   2.4s
[CV 3/5; 15/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 1/5; 15/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
```

```
score=0.988 total time=   1.7s
[CV 4/5; 15/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 5/5; 14/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.987 total time=   2.4s
[CV 5/5; 15/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 2/5; 15/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.981 total time=   1.7s
[CV 1/5; 16/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 3/5; 15/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.998 total time=   1.7s
[CV 2/5; 16/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 4/5; 15/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.955 total time=   1.7s
[CV 3/5; 16/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 5/5; 15/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.986 total time=   1.7s
[CV 4/5; 16/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 1/5; 16/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.988 total time=   2.4s
[CV 5/5; 16/24] START max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 2/5; 16/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.980 total time=   2.4s
[CV 1/5; 17/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 3/5; 16/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.998 total time=   2.4s
[CV 2/5; 17/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 4/5; 16/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.950 total time=   2.4s
[CV 3/5; 17/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 5/5; 16/24] END max_depth=10, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
```

```
score=0.985 total time=   2.4s
[CV 4/5; 17/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 1/5; 17/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.990 total time=   2.0s
[CV 5/5; 17/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500
[CV 2/5; 17/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.980 total time=   2.1s
[CV 1/5; 18/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 3/5; 17/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.999 total time=   2.4s
[CV 2/5; 18/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 4/5; 17/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.958 total time=   2.0s
[CV 3/5; 18/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 5/5; 17/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=500;,
score=0.988 total time=   2.1s
[CV 4/5; 18/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 1/5; 18/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.989 total time=   2.8s
[CV 5/5; 18/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700
[CV 2/5; 18/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.978 total time=   2.6s
[CV 1/5; 19/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 3/5; 18/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.999 total time=   2.5s
[CV 2/5; 19/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 4/5; 18/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.957 total time=   2.5s
[CV 3/5; 19/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 1/5; 19/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
```

```
score=0.989 total time=   1.8s
[CV 4/5; 19/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 5/5; 18/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=2, n_estimators=700;,
score=0.989 total time=   2.5s
[CV 5/5; 19/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500
[CV 2/5; 19/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.981 total time=   1.8s
[CV 1/5; 20/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 3/5; 19/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.999 total time=   2.0s
[CV 2/5; 20/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 4/5; 19/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.956 total time=   2.4s
[CV 3/5; 20/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 5/5; 19/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=500;,
score=0.988 total time=   2.2s
[CV 4/5; 20/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 1/5; 20/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.989 total time=   2.6s
[CV 5/5; 20/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700
[CV 2/5; 20/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.978 total time=   2.7s
[CV 1/5; 21/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 3/5; 20/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.998 total time=   2.5s
[CV 2/5; 21/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 4/5; 20/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.959 total time=   2.6s
[CV 3/5; 21/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 1/5; 21/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
```

```
score=0.987 total time=   1.7s
[CV 4/5; 21/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 5/5; 20/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=10, min_samples_split=5, n_estimators=700;,
score=0.989 total time=   2.6s
[CV 5/5; 21/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500
[CV 2/5; 21/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.980 total time=   1.8s
[CV 1/5; 22/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 3/5; 21/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.999 total time=   1.7s
[CV 2/5; 22/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 4/5; 21/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.951 total time=   1.7s
[CV 3/5; 22/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 5/5; 21/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=500;,
score=0.986 total time=   1.7s
[CV 4/5; 22/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 1/5; 22/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.989 total time=   2.4s
[CV 5/5; 22/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700
[CV 2/5; 22/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.978 total time=   2.4s
[CV 1/5; 23/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 3/5; 22/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.998 total time=   2.4s
[CV 2/5; 23/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 4/5; 22/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.950 total time=   2.4s
[CV 3/5; 23/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 1/5; 23/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
```

```
score=0.988 total time=   1.7s
[CV 4/5; 23/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 2/5; 23/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.979 total time=   1.7s
[CV 5/5; 23/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500
[CV 5/5; 22/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=2, n_estimators=700;,
score=0.986 total time=   2.4s
[CV 1/5; 24/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 3/5; 23/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.999 total time=   1.7s
[CV 2/5; 24/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 4/5; 23/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.951 total time=   1.7s
[CV 3/5; 24/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 5/5; 23/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=500;,
score=0.987 total time=   1.7s
[CV 4/5; 24/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 1/5; 24/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.988 total time=   2.4s
[CV 5/5; 24/24] START max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700
[CV 2/5; 24/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.978 total time=   2.4s
[CV 4/5; 24/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.952 total time=   2.0s
[CV 3/5; 24/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.998 total time=   2.4s
[CV 5/5; 24/24] END max_depth=15, max_features=sqrt,
min_samples_leaf=20, min_samples_split=5, n_estimators=700;,
score=0.986 total time=   2.0s

GridSearchCV(cv=5, estimator=RandomForestClassifier(), n_jobs=-1,
             param_grid={'max_depth': [5, 10, 15], 'max_features':
['sqrt'],
                         'min_samples_leaf': [10, 20],
```

```
                        'min_samples_split': [2, 5],
                        'n_estimators': [500, 700]},
            scoring='roc_auc', verbose=10)
```

3.4. Dummy Classifier

```
# Dummy Classifier
dummy = DummyClassifier()
dummy.fit(X_train_under, y_train_under)
```

```
DummyClassifier()
```

4. Model Evaluation

4.1. Find ROC scores for all models

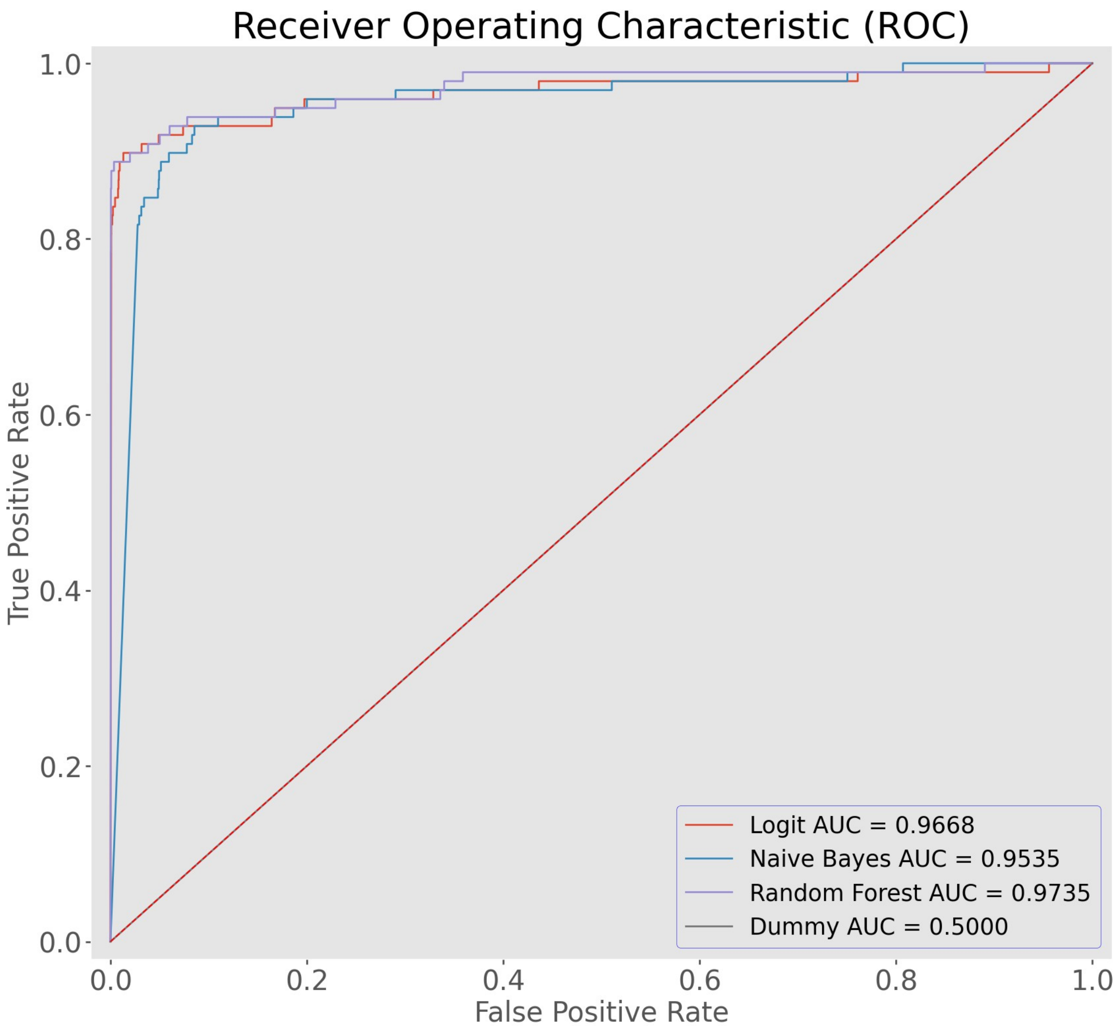
$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

$$
\begin{aligned}
precision &= \frac{TP}{TP + FP} \\
recall &= \frac{TP}{TP + FN} \\
F1 &= \frac{2 \times precision \times recall}{precision + recall} \\
accuracy &= \frac{TP + TN}{TP + FN + TN + FP} \\
specificity &= \frac{TN}{TN + FP}
\end{aligned}
$$

```
def plot_roc_curves(X, y, models, model_names, figsize=(20,18)):
    """
```

```
    Plots ROC curves for a list of models.

    Parameters:
    X (numpy.ndarray or pandas.DataFrame): input features for the
models
    y (numpy.ndarray or pandas.DataFrame): target variable
    models (list): list of models to compare
    model_names (list): list of model names to display on the plot
    figsize (tuple): size of the figure to display the plot

    Returns:
    None
    """
    fig, ax = plt.subplots(figsize=figsize)

    # Loop over models and plot ROC curve
    for i, model in enumerate(models):
        y_pred = list(model.predict_proba(X)[:, 1])
        fpr, tpr, threshold = metrics.roc_curve(y, y_pred)
        roc_auc = metrics.auc(fpr, tpr)
        plt.plot(fpr, tpr, label=(model_names[i] + ' AUC = %0.4f' %
roc_auc), linewidth=2.0)

    ax.grid(False)
    ax.tick_params(length=6, width=2, labelsize=30, grid_color='r',
grid_alpha=0.5)
    leg = plt.legend(loc='lower right', prop={'size': 25})
    leg.get_frame().set_edgecolor('b')
    plt.title('Receiver Operating Characteristic (ROC)', fontsize=40)
    plt.plot([0, 1], [0, 1], 'r--')
    plt.xlim([-.02, 1.02])
    plt.ylim([-.02, 1.02])
    plt.ylabel('True Positive Rate', fontsize=30)
    plt.xlabel('False Positive Rate', fontsize=30)
#     plt.show()

# Define the list of models to compare
models = [logistic_grid.best_estimator_, gnb_best,
rf_grid.best_estimator_, dummy]
model_names = ['Logit', 'Naive Bayes', 'Random Forest', 'Dummy']

# Plot ROC curves for in-sample data
plot_roc_curves(X_val_under, y_val_under, models, model_names)

# Save the plot as PNG file
plt.savefig('roc_insample.png');

# Plot ROC curves for out-of-sample data
plot_roc_curves(X_test_std, y_test, models, model_names)
```

```python
# Save the plot as PNG file
plt.savefig('roc_outsample.png');
```



- Recall (True Positive Rate): This metric measures the percentage of all fraudulent transactions that the model correctly identifies as fraudulent.
- Precision: This metric indicates the percentage of items that the model labels as fraud that are actually fraudulent.
- False Positive Rate: This metric measures the percentage of non-fraudulent transactions that the model incorrectly labels as fraudulent.
- Accuracy: This metric reflects how often the model is correct in its predictions overall. However, it can be misleading in the case of imbalanced data or fraud detection.
- F1 score: This metric is a combination of precision and recall, taking both false positives and false negatives into account. It's a weighted average of precision and recall and is usually more useful than accuracy, especially when dealing with uneven classes.

4.2. Determine the optimal threshold for each model.

✅ The function find_best_threshold() can be used to determine the optimal threshold for a given model. The optimal threshold is the value that maximizes the F1 score, a measure that combines precision and recall, for a binary classification problem.

✅ The function takes two arguments: model is the trained model, and num_steps is the number of steps in the threshold range to iterate over.

✅ The function first initializes variables for the highest F1 score, the best threshold, and the best accuracy, recall, and precision scores. It then iterates over a range of thresholds from 0 to 1, with num_steps steps. For each threshold, it predicts the target variable using the given threshold and calculates the F1 score, accuracy, recall, and precision scores. If the F1 score is higher than the current highest F1 score, it updates the best threshold and evaluation metrics.

✅ After iterating over all the thresholds, the function returns the best threshold and the corresponding F1 score, accuracy, recall, and precision scores.

✅ The math equation to find the F1 score is:

`F1 = 2 * (precision * recall) / (precision + recall)`

where

- precision = TP / (TP + FP)
- recall = TP / (TP + FN)
- TP: True Positive (model predicts positive and it is positive)
- FP: False Positive (model predicts positive but it is negative)
- FN: False Negative (model predicts negative but it is positive)

```
# Define a function to find the best threshold for a given model
def find_best_threshold(model, num_steps):
    highest_f1 = 0
    best_threshold = 0
    best_acc = 0
    best_rec = 0
    best_pre = 0
    # Iterate over a range of thresholds
    for threshold in np.linspace(0, 1, num_steps):
        # Predict the target variable using the given threshold
        y_predict = (model.predict_proba(X_val_under)[:, 1] >=
threshold)
        # Calculate various evaluation metrics
        f1 = f1_score(y_val_under, y_predict)
        acc = accuracy_score(y_val_under, y_predict)
        rec = recall_score(y_val_under, y_predict)
        pre = precision_score(y_val_under, y_predict)
        # Update the best threshold and metrics if F1 score improves
        if f1 > highest_f1:
            best_threshold, highest_f1, best_acc, best_rec, best_pre =
```

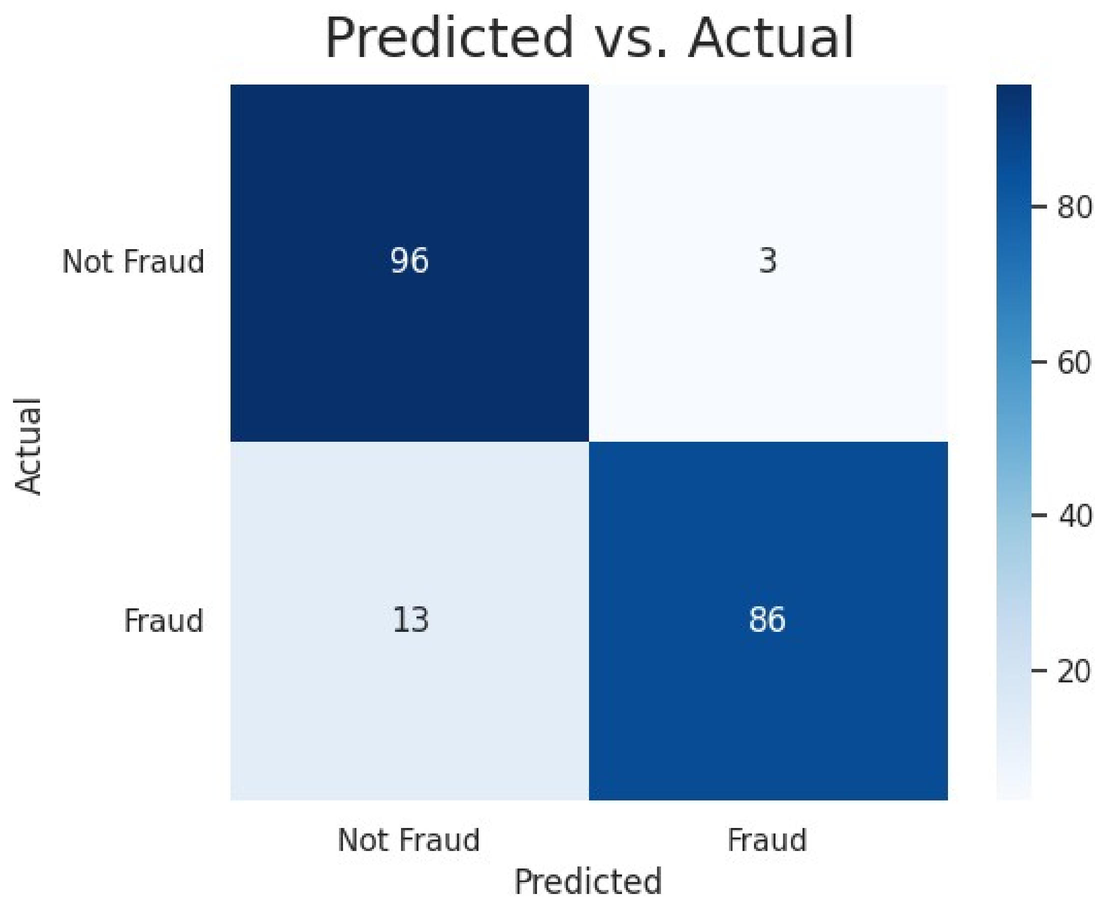
```
\
                threshold, f1, acc, rec, pre
    # Return the best threshold and evaluation metrics
    return best_threshold, highest_f1, best_acc, best_rec, best_pre

# Define a list of models and their names
models = [logistic_grid, gnb_best, rf_grid]
model_names = ["Logistic Regression", "Naive-Bayes", "Random Forest"]

# Create an empty list to store the results
chart = list()

# Iterate over the models and find the best threshold for each one
for item, name in zip(models, model_names):
    best_thresh, high_f1, high_acc, high_rec, high_pre =
find_best_threshold(item, 20)
    # Append the results to the chart list
    chart.append([name, best_thresh, high_f1, high_acc, high_rec,
high_pre])

# Create a pandas dataframe from the chart list and display it
chart = pd.DataFrame(chart, columns=['Model', 'Best Threshold', 'F1
Score', 'Accuracy', 'Recall', 'Precision'])
chart.to_csv('model_evaluation_scores.csv')
chart
```

```
                 Model  Best Threshold  F1 Score  Accuracy
Recall  \
0  Logistic Regression        0.842105  0.916667  0.919192  0.888889

1          Naive-Bayes        0.052632  0.870466  0.873737  0.848485

2        Random Forest        0.473684  0.918919  0.924242  0.858586


   Precision
0   0.946237
1   0.893617
2   0.988372
```

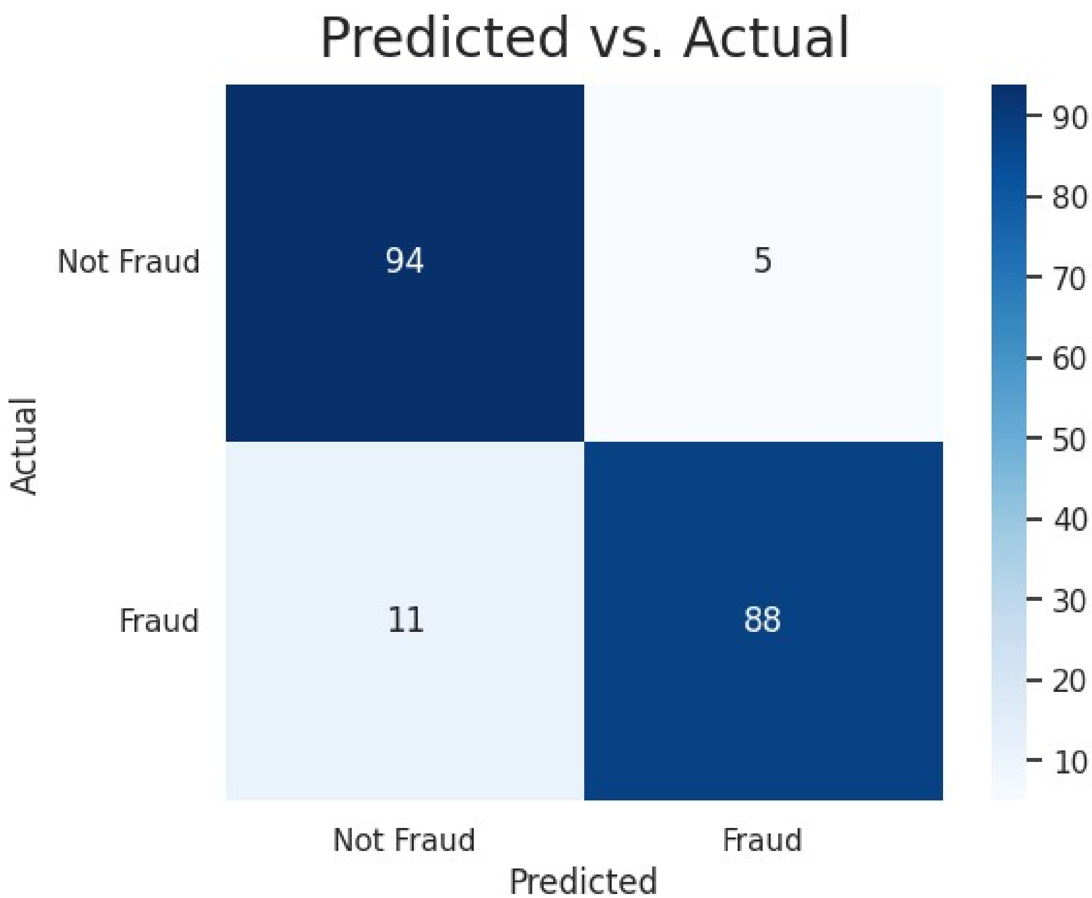
4.3. Confusion Matrix

```
def make_confusion_matrix_val(model, threshold=0.5):
    """
    Create a confusion matrix plot for the given model and threshold.

    Parameters:
    -----------
    model : sklearn classifier
        The classification model to evaluate.
    threshold : float, default=0.5
```

```
        Probability threshold for binary classification.

    Returns:
    --------
    None

    """
    # Predict class 1 if probability of being in class 1 is greater
than threshold
    # (model.predict(X_test) does this automatically with a threshold
of 0.5)
    y_predict = (model.predict_proba(X_val_under)[:, 1] >= threshold)

    # calculate the confusion matrix
    fraud_confusion = confusion_matrix(y_val_under, y_predict)

    # plot the confusion matrix as heatmap
    plt.figure(dpi=100)
    sns.set(font_scale=1)
    sns.heatmap(fraud_confusion, cmap=plt.cm.Blues, annot=True,
square=True, fmt='d',
           xticklabels=['Not Fraud', 'Fraud'],
           yticklabels=['Not Fraud', 'Fraud']);

    # calculate TP, FP, FN, and TN values from the confusion matrix
    TP = fraud_confusion[0][0]
    FP = fraud_confusion[0][1]
    FN = fraud_confusion[1][0]
    TN = fraud_confusion[1][1]

    # rotate y-axis ticks
    plt.yticks(rotation = 0)

    # set plot title, x and y labels
    plt.title('Predicted vs. Actual',fontname = '.SF Compact
Display',fontsize = 20,pad = 10);
    plt.xlabel('Predicted')
    plt.ylabel('Actual')

# Create a confusion matrix for the Random Forest model with a
threshold of 0.421 on the validation data
make_confusion_matrix_val(rf_grid, threshold=0.421)

# Save the plot as PNG file
plt.savefig('confusion_matrix_val_random_forest.png');
```

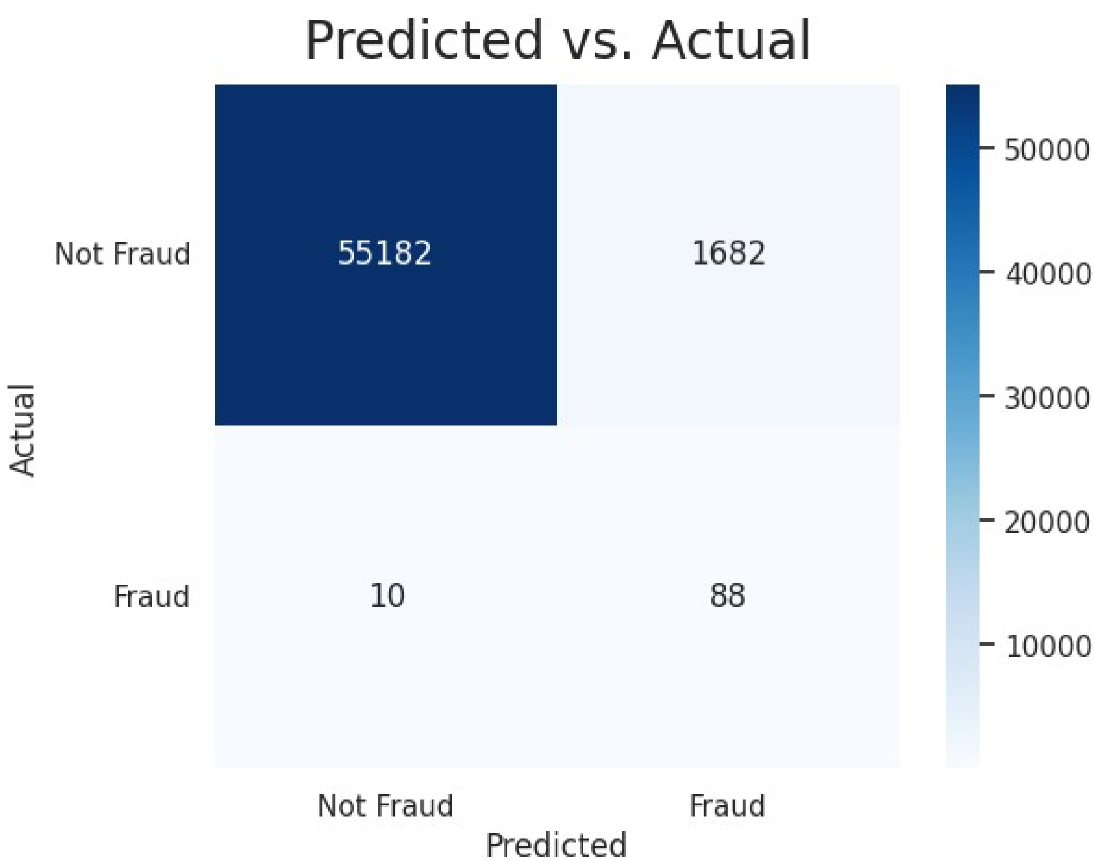



```python
# Create a confusion matrix for the Logistic Regression model with a
threshold of 0.842 on the validation data
make_confusion_matrix_val(logistic_grid, threshold=0.842)

# Save the plot as PNG file
plt.savefig('confusion_matrix_val_logistic_regression.png');
```

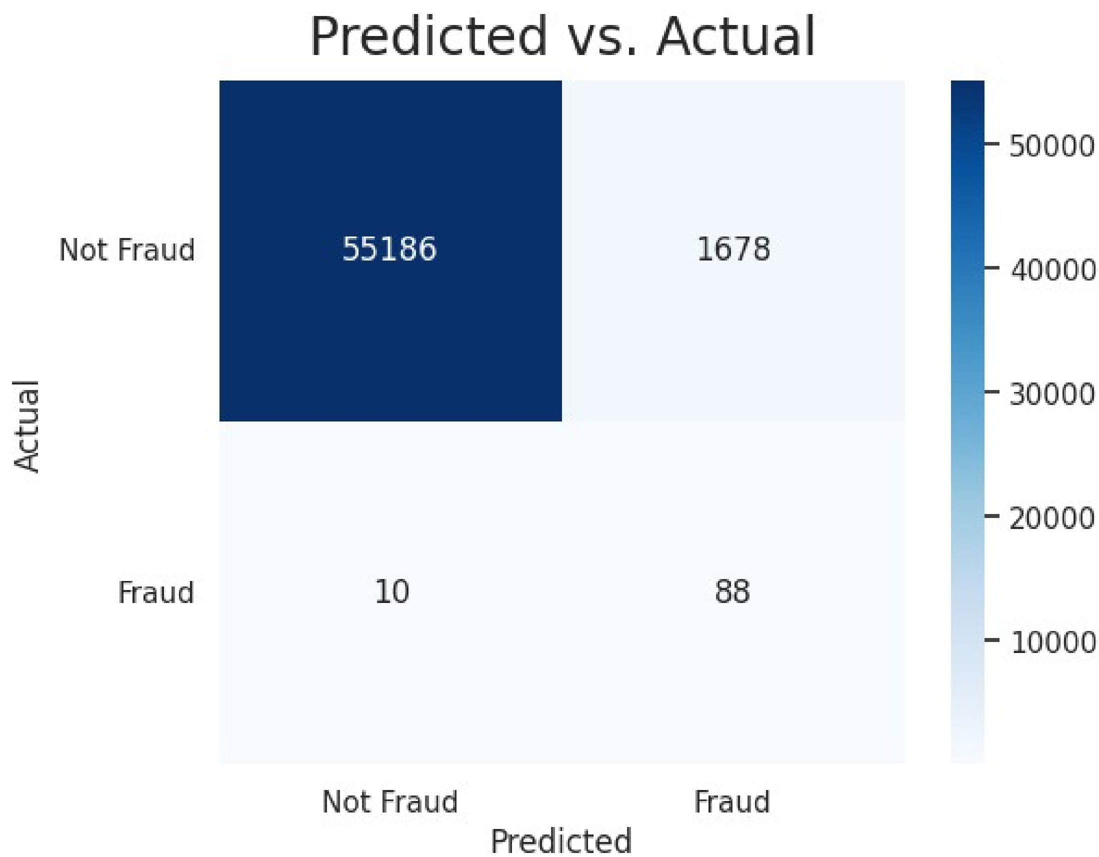



```
def make_confusion_matrix_test(model, threshold=0.5):
    """
    Generates a confusion matrix for a given model on the test
dataset, given a threshold.

    Args:
    - model: a trained machine learning model
    - threshold: threshold for binary classification

    Returns: None
    """

    # Predict class 1 if probability of being in class 1 is greater
than threshold
    y_predict = (model.predict_proba(X_test_std)[:, 1] >= threshold)

    # Generate confusion matrix
    fraud_confusion = confusion_matrix(y_test, y_predict)

    # Plot heatmap of confusion matrix
    plt.figure(dpi=100)
    sns.set(font_scale=1)
```

```
    sns.heatmap(fraud_confusion, cmap=plt.cm.Blues, annot=True,
square=True, fmt='d',
                xticklabels=['Not Fraud', 'Fraud'],
                yticklabels=['Not Fraud', 'Fraud'])

    # Calculate TP, FP, FN, TN
    TP = fraud_confusion[0][0]
    FP = fraud_confusion[0][1]
    FN = fraud_confusion[1][0]
    TN = fraud_confusion[1][1]

    # Add title, labels and rotate y-tick labels
    plt.yticks(rotation=0)
    plt.title('Predicted vs. Actual', fontname='.SF Compact Display',
fontsize=20, pad=10)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')

# Generate confusion matrix for random forest model on test dataset
make_confusion_matrix_test(rf_grid, threshold=0.421)

# Save the plot as PNG file
plt.savefig('confusion_matrix_test_random_forest.png');
```



```
# Generate confusion matrix for logistic regression model on test
dataset
make_confusion_matrix_test(logistic_grid, threshold=0.842)

# Save the plot as PNG file
plt.savefig('confusion_matrix_test_logistic_regression.png');
```



References

- Kaggle Dataset: Credit Card Fraud Detection
- Github Repo - HERE
- Kaggle Project - HERE
- Detail Explanation about the code on MEDIUM