UTM UNIVERSITI TEKNOLOGI MALAYSIA

FACULTY OF COMPUTING
UTM Johor Bahru

MCSD 2213 Advanced Analytics for Data Science

Mini Lab Project – Supervised Learning Model

Random Forest & Logistic Regression

Lecturer	Prof. Dr. Roliana Ibrahim
Date of Submission	30th June 2023

No.	Group Members	Matric No
1	Syeinrita Devi A/P Anbealagan	MCS221022
2	Siti Nurul Arisya Binti Mohd Hanifah	MCS221020

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

Supervised learning models play a crucial role in machine learning, enabling us to develop prediction models from labelled data. In this research, our focus is on fraud detection, and we will compare and analyse two commonly used supervised learning algorithms logistic regression and random forest. The dataset we will be using is the "Fraud Detection" dataset from Kaggle, which provides a rich set of transactional data for our analysis. The "Fraud Detection" dataset offers valuable insights into fraudulent activity by capturing various transaction parameters such as transaction amount, merchant type, transaction time, and anonymized card details. The business insights/goals to achieve are to determine whether a transaction is fraudulent or not, making it a binary classification problem and to find the best model for predicting fraud in credit card transactions among Random Forest and Logistic Regression.

Logistic regression is a widely used statistical modelling technique that is well-suited for binary classification tasks. It calculates the likelihood of an event occurring based on a set of independent variables, making it an excellent choice for fraud detection. Logistic regression provides interpretable results, allowing us to understand the relationship between input features and the probability of fraud. On the other hand, random forest is an ensemble learning method that combines multiple decision trees to make predictions. It is known for its ability to handle complex relationships and discover patterns in data. Random forest builds an ensemble of decision trees by introducing randomness into the training process, resulting in robust predictions. This makes it particularly useful for fraud detection tasks where the presence of intricate fraud patterns requires a flexible and powerful modelling approach.

Fraud detection is a critical task in various industries, such as finance and e-commerce, as it helps mitigate financial losses and protects customers. However, fraud datasets often suffer from severe class imbalance, where many transactions are non-fraudulent and only a small fraction represents fraudulent cases. This data imbalance poses a challenge for machine learning models, as they tend to perform poorly on minority class prediction.

REPORT OVERVIEW

In this report, we will explore how the "Fraud Detection" dataset is modelled using logistic regression and random forest. To ensure the integrity and quality of the dataset, we will pre-process it, which involves activities such as data cleaning, feature engineering, and handling missing values. The models will then be trained, tested, and evaluated using appropriate performance metrics.

We will utilize several key evaluation metrics to assess the effectiveness of logistic regression and random forest, including accuracy, precision, recall, F1-score, and AUC-ROC (Area Under the Receiver Operating Characteristic Curve). These metrics provide insights into various aspects of the model's performance, such as overall accuracy, the ability to correctly identify fraud cases, and the trade-off between precision and recall. To address the data imbalance in the fraud detection task, we will explore two approaches SMOTE (Synthetic Minority Over-sampling Technique) and under-sampling. SMOTE is an oversampling technique that generates synthetic samples for the minority class, effectively balancing the dataset. Under-sampling involves randomly selecting a subset of the majority class samples to create a balanced dataset. Both techniques aim to improve the model's ability to capture patterns and make accurate predictions for the minority class.

By comparing the performance of logistic regression and random forest with and without the use of SMOTE or under-sampling, we can determine the most effective approach for addressing the data imbalance in the fraud detection task. These findings and insights will contribute to a better understanding of the efficiency of logistic regression and random forest in real-world fraud detection applications. In summary, this research aims to evaluate the performance of logistic regression and random forest models on the "Fraud Detection" dataset, taking into account the data imbalance challenge. By analysing the results and comparing different approaches, we can gain valuable insights into the strengths, limitations, and suitability of these models for fraud detection tasks.

DATA OVERVIEW

A publicly available dataset on Kaggle is the Credit Card Transactions Fraud Detection Dataset. It consists of transaction information that a European credit card company gathered over the course of two days in September 2013. There are 1048 575 credit card transactions in the dataset. We can learn more about fraudulent activities thanks to the fraud detection dataset's useful information about credit card transactions. Credit card fraud has become a significant concern for both financial institutions and customers due to the rise in popularity of online transactions. Identifying fraudulent transactions is essential to reducing financial losses and safeguarding sensitive customer data.

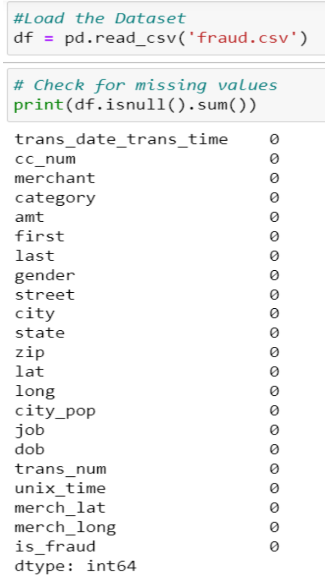
This dataset includes a number of elements related to credit card transactions, such as the date and time of the transaction, the credit card number, information about the merchant, the transaction amount, and the name, gender, and address of the cardholder. We can identify patterns and trends in this dataset that can help us spot fraudulent transactions. Our main goal is to create a fraud detection model using supervised learning methods like logistic and regression analysis. On labelled data, where each transaction is classified as either fraudulent or not, supervised learning models are trained. We can create predictive models that accurately classify and identify fraudulent transactions by utilising this labelled data.

We seek to identify the underlying patterns and relationships between the transaction features and transaction fraud by applying regression and logistic regression algorithms to this dataset. These models will gain knowledge from past data, allowing them to predict future, unforeseen transactions. The ultimate objective is to create a strong fraud detection system that can quickly recognise and flag suspicious transactions, reducing the financial impact of fraudulent activities. We can gain useful insights from this dataset and support ongoing efforts to stop credit card fraud by using supervised learning techniques. We can give financial institutions and customers a higher level of security and assurance during credit card transactions by creating precise and dependable fraud detection models.

DATA PREPROCESSING

Data pre-processing involves transforming raw data to make it suitable for analysis or machine learning tasks. It includes steps like cleaning data by handling missing values and outliers, integrating data from multiple sources, transforming data through normalization or scaling, selecting relevant features, encoding categorical variables, discretizing data if needed, and splitting the dataset into training, validation, and testing sets. These steps ensure data quality, compatibility, and enhance the performance of analysis or machine learning models.

HANDLING MISSING VALUES



```
#Load the Dataset
df = pd.read_csv('fraud.csv')
```

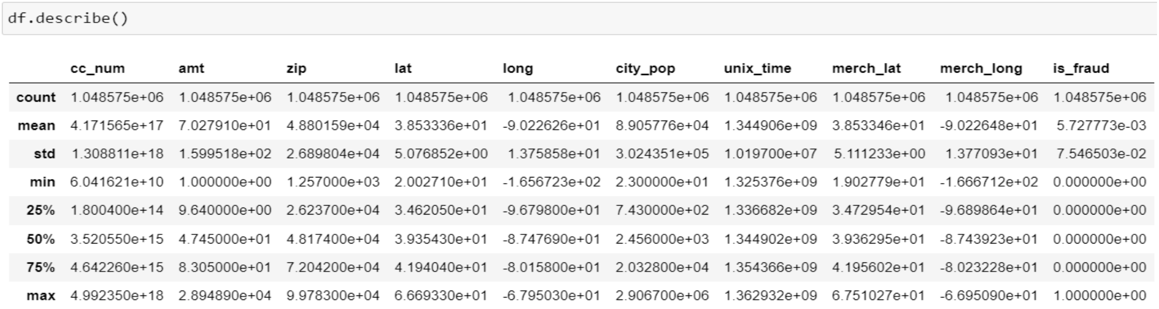
```
# Check for missing values
print(df.isnull().sum())
```

```
trans_date_trans_time    0
cc_num                   0
merchant                 0
category                 0
amt                      0
first                    0
last                     0
gender                   0
street                   0
city                     0
state                    0
zip                      0
lat                      0
long                     0
city_pop                 0
job                      0
dob                      0
trans_num                0
unix_time                0
merch_lat                0
merch_long               0
is_fraud                 0
dtype: int64
```

Figure 1 Missing values

The train data downloaded from Kaggle is being used for the modelling and is renamed as fraud.csv. Checking a dataset for missing values is an important step in data pre-processing and analysis. Missing values can have a major impact on model correctness and dependability by introducing bias, affecting statistical measurements, and leading to inaccurate conclusions.

The presence of missing values might be especially problematic in the context of fraud detection. Any missing or inadequate information or records may impair the capacity to effectively identify fraudulent transactions and detect patterns indicative of fraudulent conduct. As a result, it is critical to guarantee that the dataset is complete and does not contain any missing information. Fortunately, there are no missing values across the columns in the "Fraud Detection" dataset. This is beneficial since it streamlines data pre-processing and allows you to focus on other key components of the analysis.

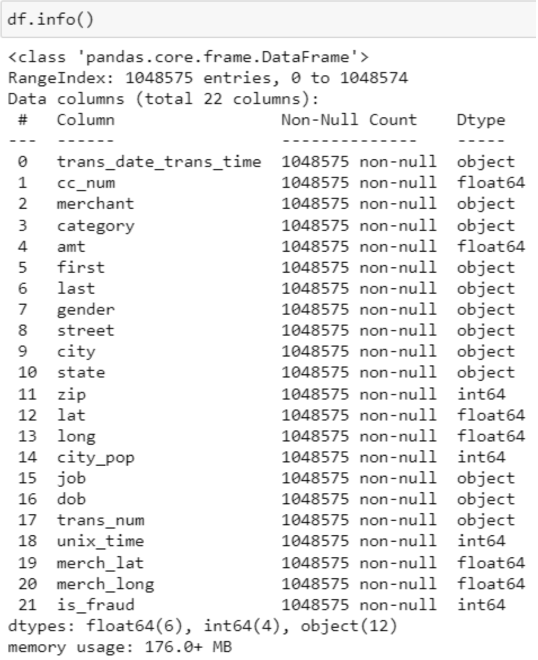


```
df.describe()
```

	cc_num	amt	zip	lat	long	city_pop	unix_time	merch_lat	merch_long	is_fraud
count	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06	1.048575e+06
mean	4.171565e+17	7.027910e+01	4.880159e+04	3.853336e+01	-9.022626e+01	8.905776e+04	1.344906e+09	3.853346e+01	-9.022648e+01	5.727773e-03
std	1.308811e+18	1.599518e+02	2.689804e+04	5.076852e+00	1.375858e+01	3.024351e+05	1.019700e+07	5.111233e+00	1.377093e+01	7.546503e-02
min	6.041621e+10	1.000000e+00	1.257000e+03	2.002710e+01	-1.656723e+02	2.300000e+01	1.325376e+09	1.902779e+01	-1.666712e+02	0.000000e+00
25%	1.800400e+14	9.640000e+00	2.623700e+04	3.462050e+01	-9.679800e+01	7.430000e+02	1.336682e+09	3.472954e+01	-9.689864e+01	0.000000e+00
50%	3.520550e+15	4.745000e+01	4.817400e+04	3.935430e+01	-8.747690e+01	2.456000e+03	1.344902e+09	3.936295e+01	-8.743923e+01	0.000000e+00
75%	4.642260e+15	8.305000e+01	7.204200e+04	4.194040e+01	-8.015800e+01	2.032800e+04	1.354366e+09	4.195602e+01	-8.023228e+01	0.000000e+00
max	4.992350e+18	2.894890e+04	9.978300e+04	6.669330e+01	-6.795030e+01	2.906700e+06	1.362932e+09	6.751027e+01	-6.695090e+01	1.000000e+00

Figure 2 Data describe

The describe() function provides a summary of the statistical measures for each numerical column in the data.



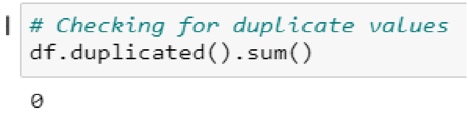
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1048575 entries, 0 to 1048574
Data columns (total 22 columns):
 #   Column                 Non-Null Count    Dtype  
---  ------                 --------------    -----  
 0   trans_date_trans_time  1048575 non-null  object 
 1   cc_num                 1048575 non-null  float64
 2   merchant               1048575 non-null  object 
 3   category               1048575 non-null  object 
 4   amt                    1048575 non-null  float64
 5   first                  1048575 non-null  object 
 6   last                   1048575 non-null  object 
 7   gender                 1048575 non-null  object 
 8   street                 1048575 non-null  object 
 9   city                   1048575 non-null  object 
 10  state                  1048575 non-null  object 
 11  zip                    1048575 non-null  int64  
 12  lat                    1048575 non-null  float64
 13  long                   1048575 non-null  float64
 14  city_pop               1048575 non-null  int64  
 15  job                    1048575 non-null  object 
 16  dob                    1048575 non-null  object 
 17  trans_num              1048575 non-null  object 
 18  unix_time              1048575 non-null  int64  
 19  merch_lat              1048575 non-null  float64
 20  merch_long             1048575 non-null  float64
 21  is_fraud               1048575 non-null  int64  
dtypes: float64(6), int64(4), object(12)
memory usage: 176.0+ MB
```

Figure 3 Data summary

The info() function provides a concise summary of the data, including the column names, data types, and the number of non-null values in each column.

HANDLING DUPLICATED VALUES



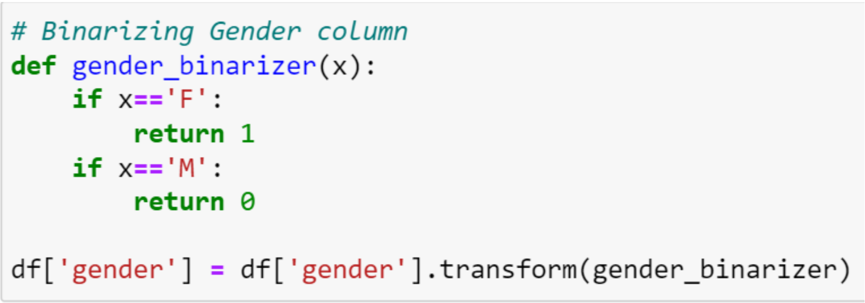
```
# Checking for duplicate values
df.duplicated().sum()

0
```

Figure 4 Duplicated values

In order to ensure data integrity and correctness, it is critical to check for duplicate values in data analysis. Duplicate values can add bias and skew statistical analysis or machine learning models' outcomes. Duplicate detection and handling are very important when working with datasets that should have unique observations, such as transactional data. As stated below, this dataset has no duplicate values.

HANDLING CATEGORICAL VARIABLES

DUMMY VARIABLES

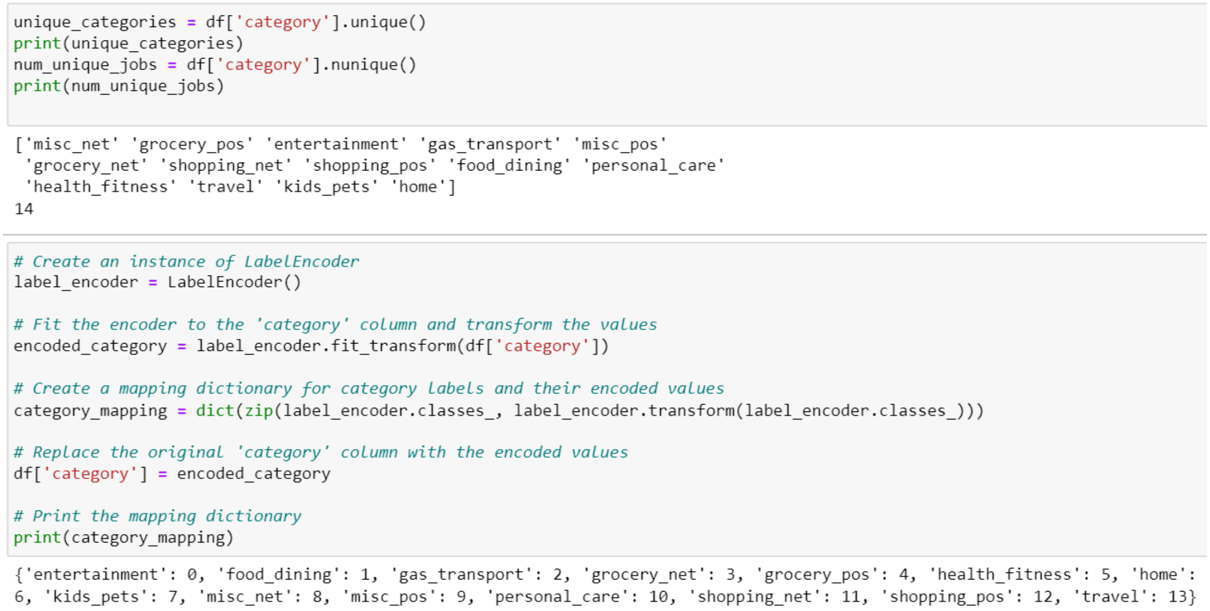


```
# Binarizing Gender column
def gender_binarizer(x):
    if x=='F':
        return 1
    if x=='M':
        return 0

df['gender'] = df['gender'].transform(gender_binarizer)
```

Figure 5 Converting gender variable

Converting categorical data into numerical representations is critical for effective analysis in supervised modelling techniques such as logistic regression and random forest. Binarizing the "gender" column in this code is required to convert the category gender variable to a binary feature. The models can understand and use this information in the prediction process by assigning 0 for male and 1 for female. This conversion allows the algorithms to incorporate any potential connections between gender and fraud detection, resulting in increased prediction accuracy and relevance.

LABEL ENCODER



```
unique_categories = df['category'].unique()
print(unique_categories)
num_unique_jobs = df['category'].nunique()
print(num_unique_jobs)
```

```
['misc_net' 'grocery_pos' 'entertainment' 'gas_transport' 'misc_pos'
 'grocery_net' 'shopping_net' 'shopping_pos' 'food_dining' 'personal_care'
 'health_fitness' 'travel' 'kids_pets' 'home']
14
```

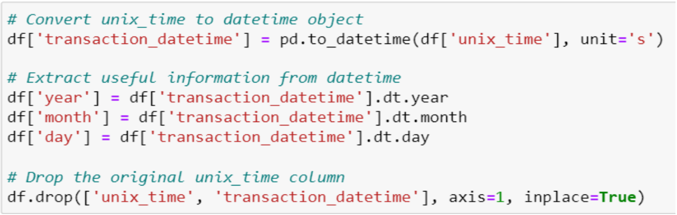
```
# Create an instance of LabelEncoder
label_encoder = LabelEncoder()

# Fit the encoder to the 'category' column and transform the values
encoded_category = label_encoder.fit_transform(df['category'])

# Create a mapping dictionary for category labels and their encoded values
category_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))

# Replace the original 'category' column with the encoded values
df['category'] = encoded_category

# Print the mapping dictionary
print(category_mapping)
```

```
{'entertainment': 0, 'food_dining': 1, 'gas_transport': 2, 'grocery_net': 3, 'grocery_pos': 4, 'health_fitness': 5, 'home':
6, 'kids_pets': 7, 'misc_net': 8, 'misc_pos': 9, 'personal_care': 10, 'shopping_net': 11, 'shopping_pos': 12, 'travel': 13}
```

Figure 6 LabelEncoder

Many machine learning techniques, such as logistic regression and random forest, necessitate the encoding of categorical variables as numerical values. We can effectively process the data by finding the unique categories and accurately encoding them into numeric representations. While the labels supplied by LabelEncoder have no inherent value, they do provide a consistent and interpretable representation of the categories. This enables us to comprehend the relationship between the encoded values and the original categories, which aids in the interpretation of the model's output. Thus, the important categorical variables needed for the analysis have been converted using LabelEncoder to make an effective analysis on the dataset.

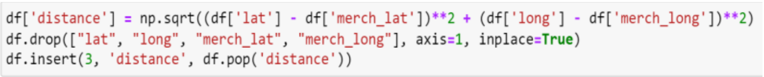
TRANFORMING & CREATING NEW VARIABLES



```
# Convert unix_time to datetime object
df['transaction_datetime'] = pd.to_datetime(df['unix_time'], unit='s')

# Extract useful information from datetime
df['year'] = df['transaction_datetime'].dt.year
df['month'] = df['transaction_datetime'].dt.month
df['day'] = df['transaction_datetime'].dt.day

# Drop the original unix_time column
df.drop(['unix_time', 'transaction_datetime'], axis=1, inplace=True)
```

Figure 7 New date time variable

The "unix_time" column contains Unix time stamps that denote the number of seconds since January 1, 1970. We can simply interact with these numbers and extract specific information such as year, month, and day by converting them to a datetime object with pd.to_datetime. These extracted features provide insights into fraud detection patterns and trends over time. The removal of the original "unix_time" column reduces redundancy and maintains the dataset's cleanliness. Overall, transforming "unix_time" to a datetime object and extracting temporal information enhances the dataset and improves the performance of logistic regression and random forest models in fraud detection by capturing time-related trends.

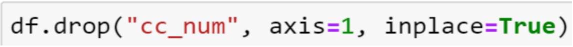


```
df['distance'] = np.sqrt((df['lat'] - df['merch_lat'])**2 + (df['long'] - df['merch_long'])**2)
df.drop(["lat", "long", "merch_lat", "merch_long"], axis=1, inplace=True)
df.insert(3, 'distance', df.pop('distance'))
```

Figure 8 New distance variable

Using the latitude and longitude coordinates, the code snippet computes the Euclidean distance between the transaction and merchant locations. We capture the spatial proximity between the two places by incorporating this distance as a new column in the DataFrame, which can be useful for fraud detection. The original latitude and longitude columns are eliminated after computing the distance to streamline the dataset and remove unnecessary features. This phase contributes to the improved efficiency and relevance of logistic regression and random forest models in detecting fraudulent activity based on spatial patterns.

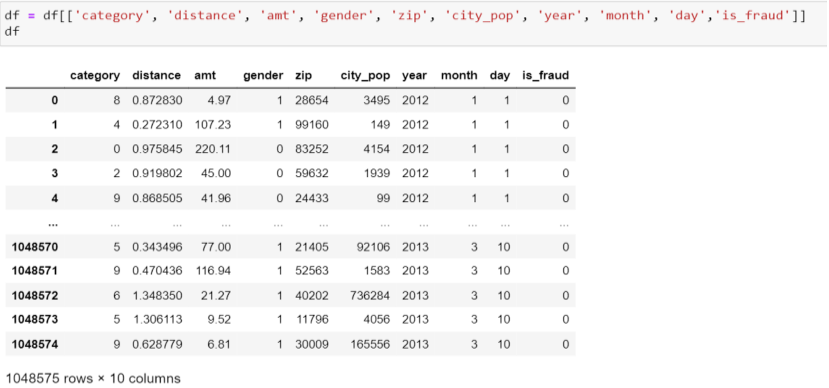
REMOVING UNWANTED COLUMNS



```
df.drop("cc_num", axis=1, inplace=True)
```

Figure 9 Removal of variables

Then, the not useful attributes for the analysis like cc_num variable is removed from the data. Removing irrelevant variables enhances the model's performance by providing a cleaner signal and allowing it to focus on the most relevant features. It also helps reduce the dimensionality of the dataset, making the model more efficient and mitigating the risk of overfitting.



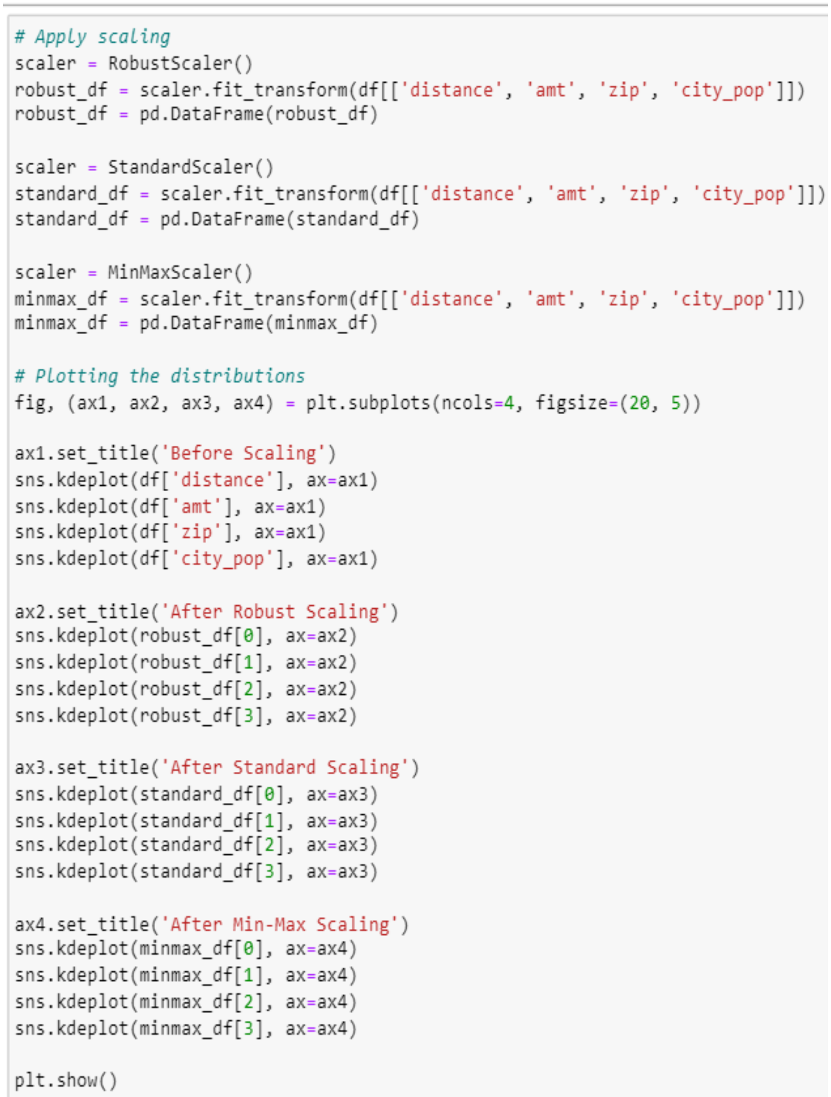
```
df = df[['category', 'distance', 'amt', 'gender', 'zip', 'city_pop', 'year', 'month', 'day','is_fraud']]
df
```

```
         category  distance     amt  gender    zip  city_pop  year  month  day  is_fraud
      0         8  0.872830    4.97       1  28654      3495  2012      1    1         0
      1         4  0.272310  107.23       1  99160       149  2012      1    1         0
      2         0  0.975845  220.11       0  83252      4154  2012      1    1         0
      3         2  0.919802   45.00       0  59632      1939  2012      1    1         0
      4         9  0.868505   41.96       0  24433        99  2012      1    1         0
    ...       ...       ...     ...     ...    ...       ...   ...    ...  ...       ...
1048570         5  0.343496   77.00       1  21405     92106  2013      3   10         0
1048571         9  0.470436  116.94       1  52563      1583  2013      3   10         0
1048572         6  1.348350   21.27       1  40202    736284  2013      3   10         0
1048573         5  1.306113    9.52       1  11796      4056  2013      3   10         0
1048574         9  0.628779    6.81       1  30009    165556  2013      3   10         0

1048575 rows × 10 columns
```

Figure 10 Rearrange columns

The columns then rearrange so that the target variable, is_fraud be the last column of the data.

SCALING



```
# Apply scaling
scaler = RobustScaler()
robust_df = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
robust_df = pd.DataFrame(robust_df)

scaler = StandardScaler()
standard_df = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
standard_df = pd.DataFrame(standard_df)

scaler = MinMaxScaler()
minmax_df = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
minmax_df = pd.DataFrame(minmax_df)

# Plotting the distributions
fig, (ax1, ax2, ax3, ax4) = plt.subplots(ncols=4, figsize=(20, 5))

ax1.set_title('Before Scaling')
sns.kdeplot(df['distance'], ax=ax1)
sns.kdeplot(df['amt'], ax=ax1)
sns.kdeplot(df['zip'], ax=ax1)
sns.kdeplot(df['city_pop'], ax=ax1)

ax2.set_title('After Robust Scaling')
sns.kdeplot(robust_df[0], ax=ax2)
sns.kdeplot(robust_df[1], ax=ax2)
sns.kdeplot(robust_df[2], ax=ax2)
sns.kdeplot(robust_df[3], ax=ax2)

ax3.set_title('After Standard Scaling')
sns.kdeplot(standard_df[0], ax=ax3)
sns.kdeplot(standard_df[1], ax=ax3)
sns.kdeplot(standard_df[2], ax=ax3)
sns.kdeplot(standard_df[3], ax=ax3)

ax4.set_title('After Min-Max Scaling')
sns.kdeplot(minmax_df[0], ax=ax4)
sns.kdeplot(minmax_df[1], ax=ax4)
sns.kdeplot(minmax_df[2], ax=ax4)
sns.kdeplot(minmax_df[3], ax=ax4)

plt.show()
```

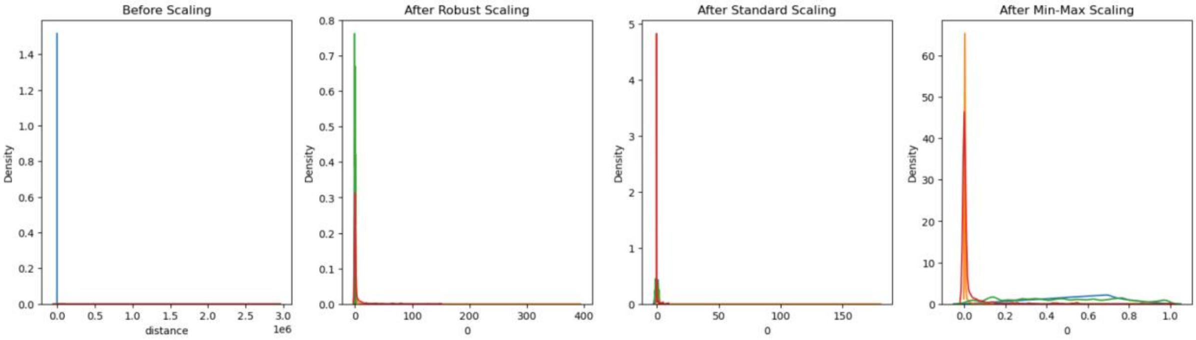
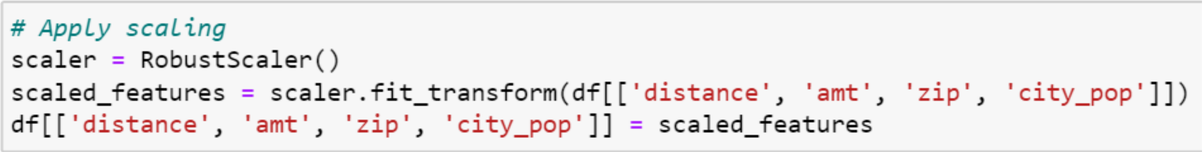


Figure 11 Scaling

Scaling is necessary to ensure that features with varying scales and ranges do not dominate the learning process or introduce biases into the models. Distance calculations or numerical comparisons between features are frequently used in supervised learning algorithms, and having features on different scales can result in inaccurate or skewed predictions. The models benefit from scaling the features to a standardised range because it allows them to weigh each feature equally and make fair comparisons based on their contributions. To achieve feature scaling, techniques such as Robust Scaling, Standard Scaling, and Min-Max Scaling are commonly used. Robust Scaling is less sensitive to outliers, making it appropriate for datasets with extreme values. When the features are normally distributed, Standard Scaling transforms them to have a zero mean and unit variance. The features are scaled to a specific range, typically between 0 and 1, while retaining the original distribution.

After scaling, the plotted distributions show that the feature distributions are more comparable and aligned, which can improve the model's ability to capture meaningful patterns and make accurate predictions. After scaling the features, we can observe that there is not much visible difference in the Figure 11. This can happen if the features in the dataset have similar scales already or if the scaling techniques used do not significantly alter the distributions. Besides, scaling may not be visible in the graphs in some cases, but it can still have a significant impact on the performance of supervised learning models. The primary goal of scaling is to bring features to a similar scale and range, which allows the models to accurately interpret their relative importance. It can improve model convergence, stability, and generalisation to previously unseen data.

ROBUST SCALING

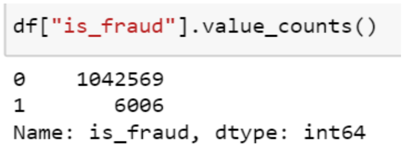


```
# Apply scaling
scaler = RobustScaler()
scaled_features = scaler.fit_transform(df[['distance', 'amt', 'zip', 'city_pop']])
df[['distance', 'amt', 'zip', 'city_pop']] = scaled_features
```

Figure 12 Robust scaling

In this case, RobustScaler was chosen because it is less sensitive to outliers than other scaling methods. Outliers are common in fraud detection datasets and can have a significant impact on the scaling process. RobustScaler scales the features using robust statistics such as median and interquartile range, making it more resistant to outliers. We ensure that the scaling is not skewed by extreme values by using RobustScaler, resulting in more reliable and consistent results. This is especially important in supervised learning models such as logistic regression and random forest because it helps capture meaningful patterns while reducing the impact of outliers.

HANDLING IMBALANCE DATA



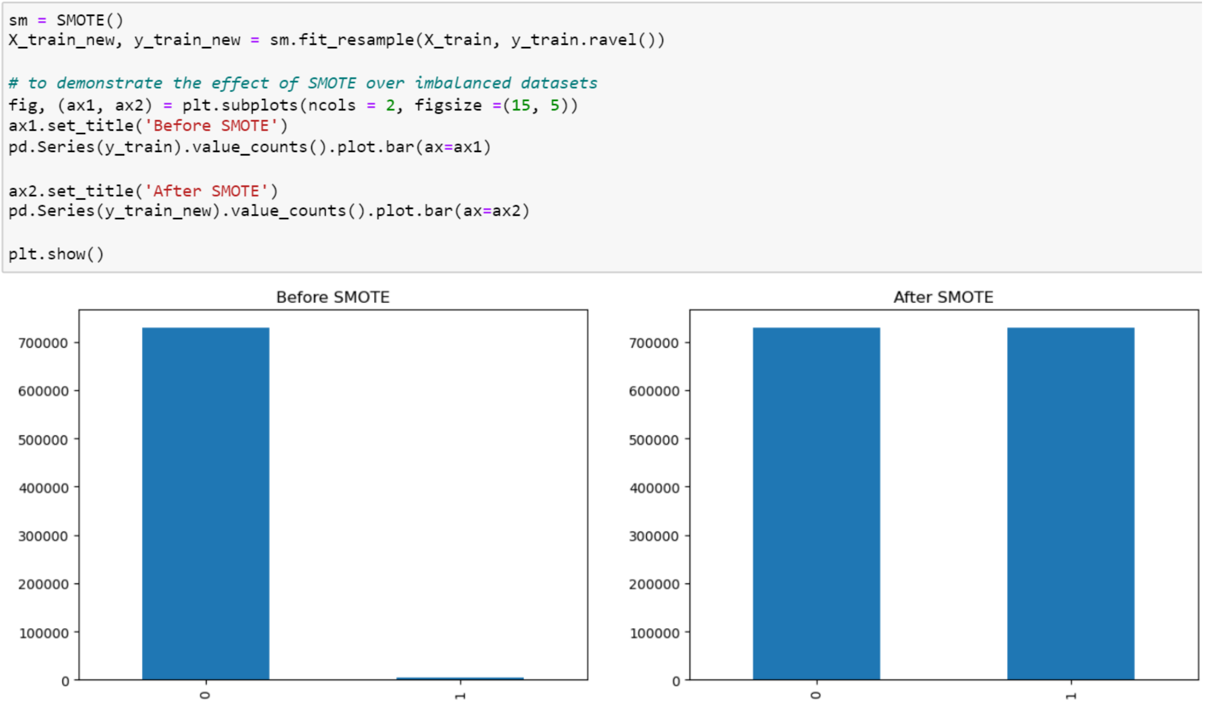
```
df["is_fraud"].value_counts()
```

```
0    1042569
1       6006
Name: is_fraud, dtype: int64
```

Figure 13 Imbalance data

To avoid biases in prediction, data must be balanced. There are 1,042,569 instances labelled as 0 (indicating non-fraudulent transactions) and 6,006 instances labelled as 1 (indicating fraudulent transactions) in this data. This indicates that the dataset is skewed, with a significantly higher proportion of non-fraudulent transactions than fraudulent ones. Imbalanced datasets can make training machine learning models difficult because they can lead to biassed predictions. To ensure fair and accurate predictions for both classes, it is critical to handle the class imbalance issue appropriately before model training.

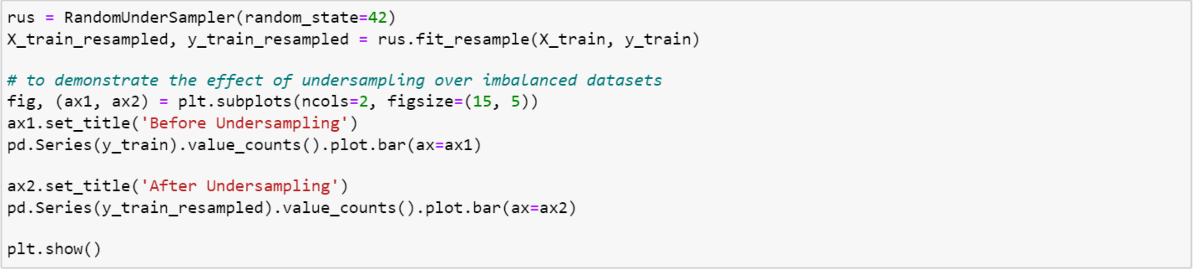
SYNTHETIC MINORITY OVER-SAMPLING TECHNIQUE (SMOTE)



```
sm = SMOTE()
X_train_new, y_train_new = sm.fit_resample(X_train, y_train.ravel())

# to demonstrate the effect of SMOTE over imbalanced datasets
fig, (ax1, ax2) = plt.subplots(ncols = 2, figsize =(15, 5))
ax1.set_title('Before SMOTE')
pd.Series(y_train).value_counts().plot.bar(ax=ax1)

ax2.set_title('After SMOTE')
pd.Series(y_train_new).value_counts().plot.bar(ax=ax2)

plt.show()
```

Figure 14 SMOTE

UNDERSAMPLING



```
rus = RandomUnderSampler(random_state=42)
X_train_resampled, y_train_resampled = rus.fit_resample(X_train, y_train)

# to demonstrate the effect of undersampling over imbalanced datasets
fig, (ax1, ax2) = plt.subplots(ncols=2, figsize=(15, 5))
ax1.set_title('Before Undersampling')
pd.Series(y_train).value_counts().plot.bar(ax=ax1)

ax2.set_title('After Undersampling')
pd.Series(y_train_resampled).value_counts().plot.bar(ax=ax2)

plt.show()
```

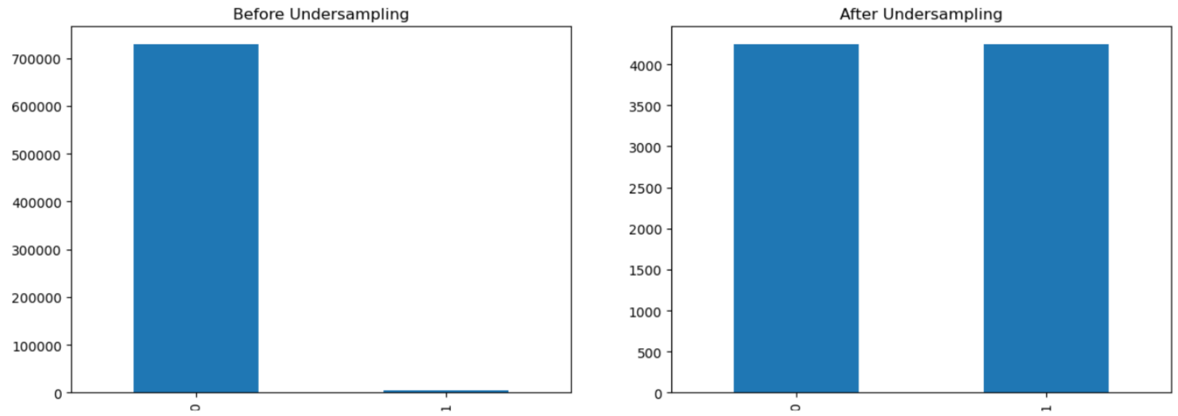


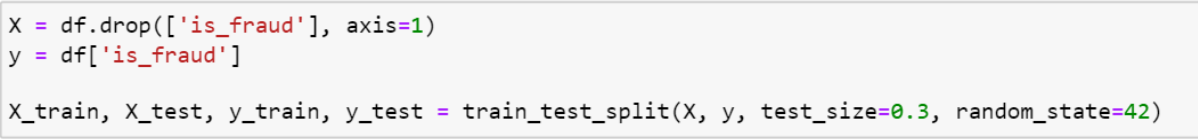
Figure 15 Under-sampling

Synthetic Minority Over-sampling Technique (SMOTE) and Random Under-sampling are the two methods used to deal with the imbalanced data. SMOTE is an oversampling technique used to balance a dataset by creating synthetic samples of the minority class (fraudulent transactions). It creates new synthetic instances by interpolating between existing instances of the minority class. This helps to increase minority representation and address the issue of class imbalance. We can observe that after applying SMOTE to the dataset both fraudulent and non-fraudulent transactions became same as shown in the Figure 14.

Random under-sampling, on the other hand, reduces the majority class (non-fraudulent transactions) by selecting a subset of instances at random from it. This contributes to the dataset's balance by matching the number of instances in the majority and minority classes. We can observe that after applying under-sampling to the dataset both fraudulent and non-fraudulent transactions became same as shown in the Figure 15.

Both methods aim to address the issue of class imbalance by improving the model's ability to learn from the minority class. The decision between SMOTE and Random Under-sampling is influenced by the dataset's characteristics and the problem at hand. SMOTE and Random Under-sampling are used in the analysis to compare their effectiveness in dealing with the imbalance. We can determine which method produces a more balanced dataset by visualising the class distribution before and after each method. This evaluation will aid in determining which method is best suited for improving the analysis's performance.

DATA SPLITTING (TRAIN & TEST)



```
X = df.drop(['is_fraud'], axis=1)
y = df['is_fraud']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

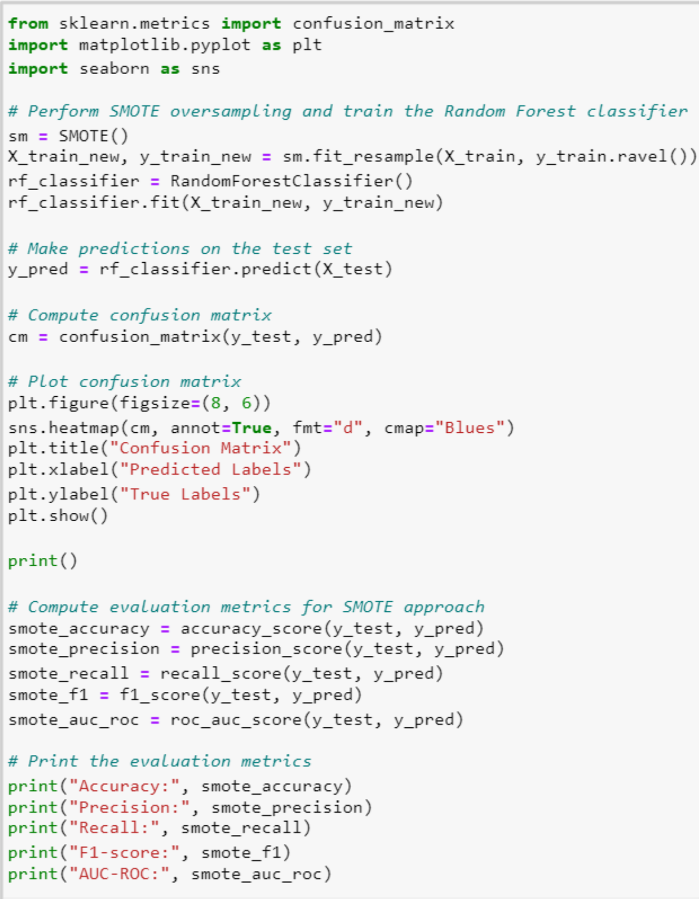
Figure 16 Data splitting

The code divides the dataset into separate training and testing sets after performing a train-test split on the features (X). This division is critical for evaluating the performance of the supervised learning models. 70% of the dataset is used for training (X_train, y_train) and 30% for testing (X_test, y_test). The goal is to train the models on the training data and then evaluate their performance on the unknown testing data, allowing us to assess their ability to generalise to new data.

MODELLING – SUPERVISED LEARNING

RANDOM FOREST CLASSIFIER (DECISION TREE)

SMOTE



```
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

# Perform SMOTE oversampling and train the Random Forest classifier
sm = SMOTE()
X_train_new, y_train_new = sm.fit_resample(X_train, y_train.ravel())
rf_classifier = RandomForestClassifier()
rf_classifier.fit(X_train_new, y_train_new)

# Make predictions on the test set
y_pred = rf_classifier.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

print()

# Compute evaluation metrics for SMOTE approach
smote_accuracy = accuracy_score(y_test, y_pred)
smote_precision = precision_score(y_test, y_pred)
smote_recall = recall_score(y_test, y_pred)
smote_f1 = f1_score(y_test, y_pred)
smote_auc_roc = roc_auc_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", smote_accuracy)
print("Precision:", smote_precision)
print("Recall:", smote_recall)
print("F1-score:", smote_f1)
print("AUC-ROC:", smote_auc_roc)
```

Figure 17 Train & Test for Random Forest (SMOTE)

Using the SMOTE (Synthetic Minority Over-sampling Technique), a Random Forest classifier is trained on the oversampled data. SMOTE is used to address the dataset's class imbalance problem by generating synthetic samples of the minority class (fraudulent transactions) to balance the dataset.

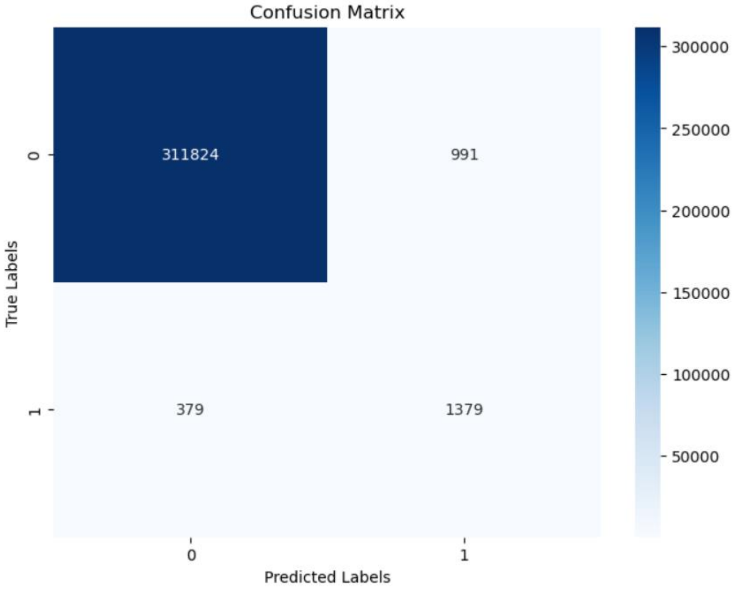
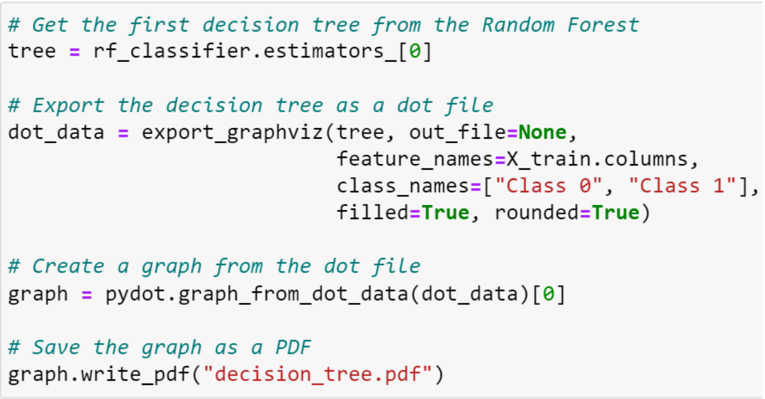


Figure 18Confusion Matrix for SMOTE Technique

Predictions are made on the test set (X_test) after the Random Forest classifier has been trained. The confusion matrix is then printed to assess the model's performance. The number of true negatives (311,824), false positives (991), false negatives (379), and true positives (1,379) are all displayed in the confusion matrix.

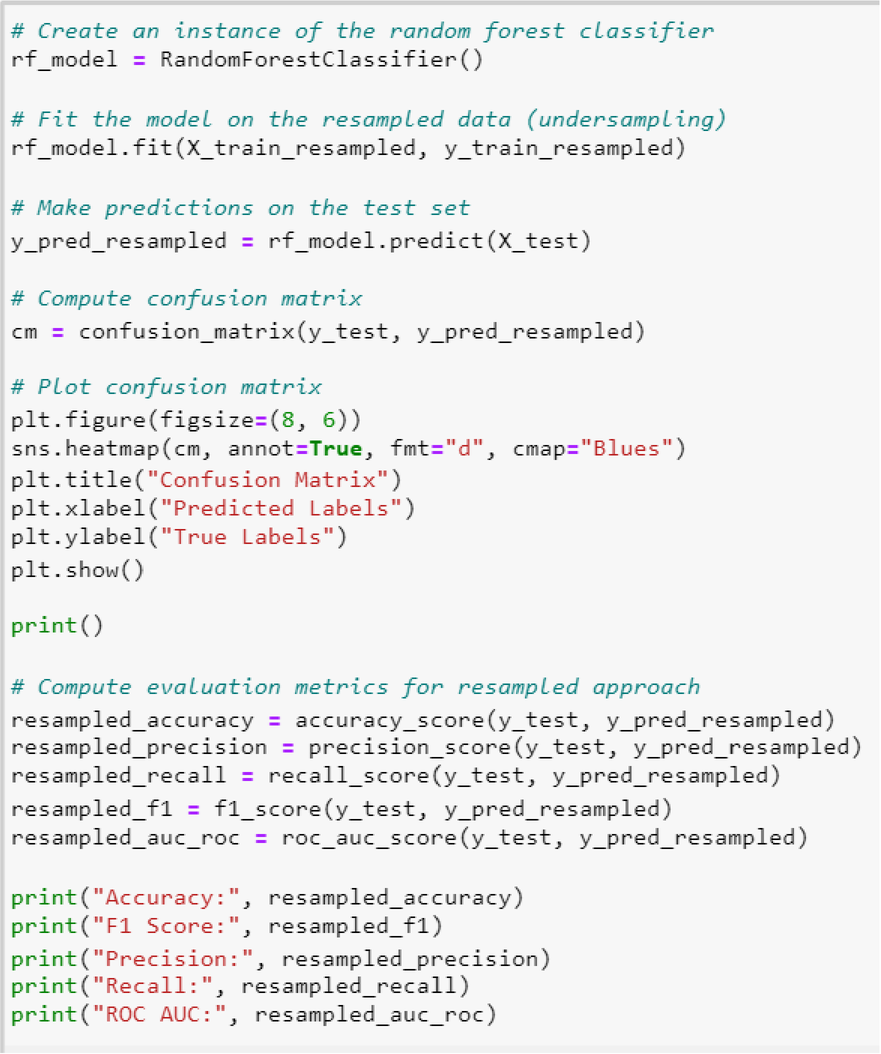
DECISION TREE



```python
# Get the first decision tree from the Random Forest
tree = rf_classifier.estimators_[0]

# Export the decision tree as a dot file
dot_data = export_graphviz(tree, out_file=None,
                           feature_names=X_train.columns,
                           class_names=["Class 0", "Class 1"],
                           filled=True, rounded=True)

# Create a graph from the dot file
graph = pydot.graph_from_dot_data(dot_data)[0]

# Save the graph as a PDF
graph.write_pdf("decision_tree.pdf")
```

The code exports the first decision tree from the SMOTE-trained Random Forest classifier. The exported decision tree depicts the structure of the tree and aids in understanding its decision-making process. For easy sharing and reference, the tree is saved as a PDF file. This analysis aids in interpreting feature importance and comprehending the Random Forest model with SMOTE's underlying logic. View the pdf by clicking here.

UNDERSAMPLING



```python
# Create an instance of the random forest classifier
rf_model = RandomForestClassifier()

# Fit the model on the resampled data (undersampling)
rf_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_resampled = rf_model.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_resampled)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

print()

# Compute evaluation metrics for resampled approach
resampled_accuracy = accuracy_score(y_test, y_pred_resampled)
resampled_precision = precision_score(y_test, y_pred_resampled)
resampled_recall = recall_score(y_test, y_pred_resampled)
resampled_f1 = f1_score(y_test, y_pred_resampled)
resampled_auc_roc = roc_auc_score(y_test, y_pred_resampled)

print("Accuracy:", resampled_accuracy)
print("F1 Score:", resampled_f1)
print("Precision:", resampled_precision)
print("Recall:", resampled_recall)
print("ROC AUC:", resampled_auc_roc)
```

Figure 19 Train & Test for Random Forest (Under-sampling)

Under-sampling is another approach used to address the class imbalance problem in a dataset. Unlike oversampling techniques like SMOTE, under-sampling aims to reduce the number of instances in the majority class (non-fraudulent transactions) to achieve a more balanced distribution between classes.

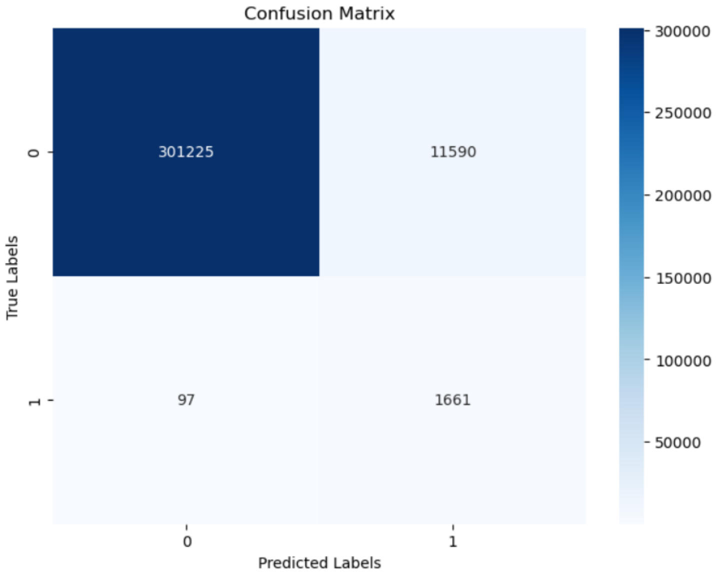
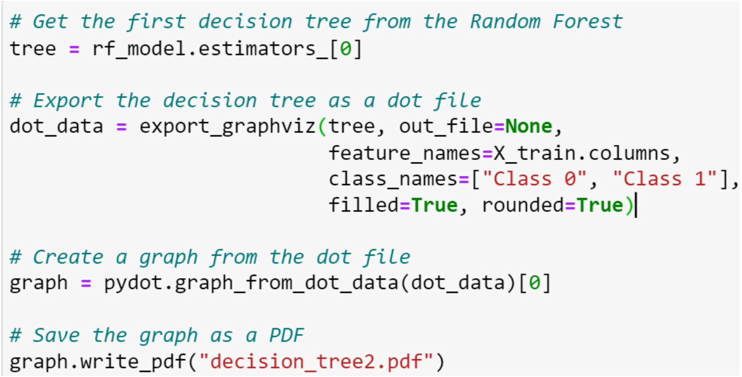


Figure 20 Confusion Matrix for Under-sampling Technique (RF)

The Random Forest classifier is trained on resampled data before making predictions on the test set (X_test). The confusion matrix is then printed to assess the performance of the model. The number of true negatives (301,225), false positives (11,590), false negatives (97), and true positives (1,661) are all displayed in the confusion matrix.

DECISION TREE



```
# Get the first decision tree from the Random Forest
tree = rf_model.estimators_[0]

# Export the decision tree as a dot file
dot_data = export_graphviz(tree, out_file=None,
                           feature_names=X_train.columns,
                           class_names=["Class 0", "Class 1"],
                           filled=True, rounded=True)

# Create a graph from the dot file
graph = pydot.graph_from_dot_data(dot_data)[0]

# Save the graph as a PDF
graph.write_pdf("decision_tree2.pdf")
```

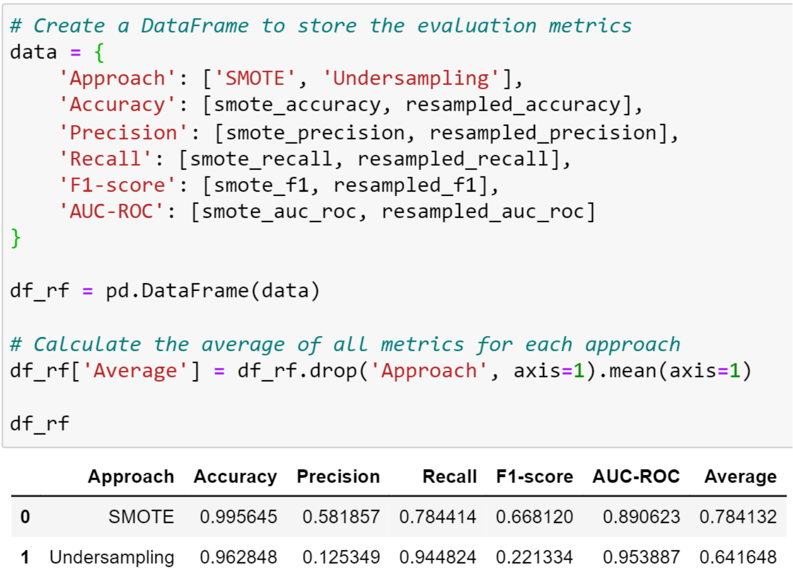
The code exports the first decision tree from the undersampling-trained Random Forest classifier. The exported decision tree depicts the structure of the tree and aids in understanding its decision-making process. For easy sharing and reference, the tree is saved as a PDF file. This analysis aids in interpreting feature importance and comprehending the Random Forest model with undersampling's underlying logic. View the pdf by clicking here.

COMPARISON (SMOTE & UNDERSAMPLING)

CONFUSION MATRIX

When comparing the two sets of results, it is evident that the Random Forest classifier performs differently in each scenario when evaluated on the test set (X_test). With the SMOTE technique, results demonstrate a higher number of true negatives and lower false positives, indicating a better ability to correctly predict the negative class. However, it also has more false negatives and lower true positives, suggesting a higher number of instances incorrectly predicted as the negative class and a lower number of instances correctly predicted as the positive class. In contrast, with under-sampling technique, results showcases a higher number of false positives but lower number of false negatives, along with higher counts of true positives. Although the first set has more false negatives compared to the second set, the higher true negative count and lower false positive count outweigh this drawback.

PERFORMANCE MEASURES



```
# Create a DataFrame to store the evaluation metrics
data = {
    'Approach': ['SMOTE', 'Undersampling'],
    'Accuracy': [smote_accuracy, resampled_accuracy],
    'Precision': [smote_precision, resampled_precision],
    'Recall': [smote_recall, resampled_recall],
    'F1-score': [smote_f1, resampled_f1],
    'AUC-ROC': [smote_auc_roc, resampled_auc_roc]
}

df_rf = pd.DataFrame(data)

# Calculate the average of all metrics for each approach
df_rf['Average'] = df_rf.drop('Approach', axis=1).mean(axis=1)

df_rf
```

	Approach	Accuracy	Precision	Recall	F1-score	AUC-ROC	Average
0	SMOTE	0.995645	0.581857	0.784414	0.668120	0.890623	0.784132
1	Undersampling	0.962848	0.125349	0.944824	0.221334	0.953887	0.641648

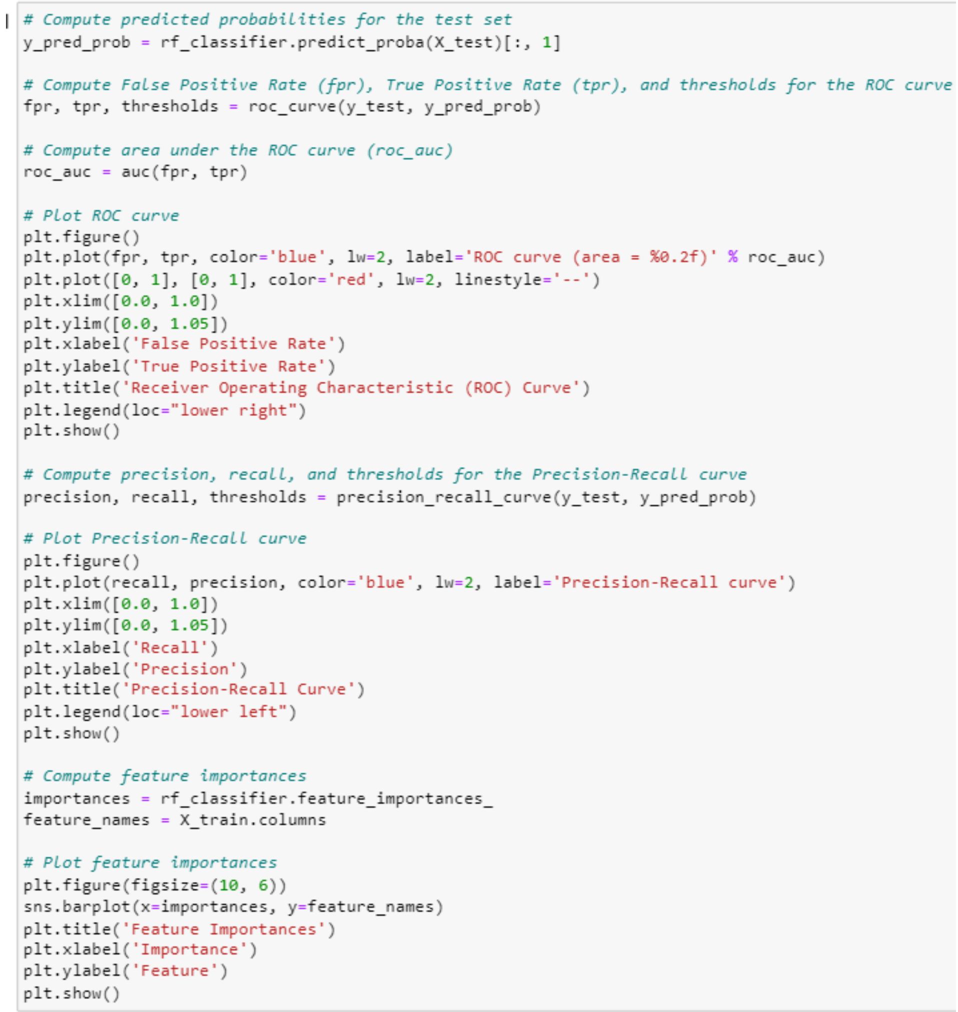
Figure 21Comparison between SMOTE and Under-sampling Techniques Result (RF)

To further assess the model's performance, the evaluation metrics (accuracy, precision, recall, F1-score, and AUC-ROC) are computed. Accuracy quantifies the proportion of correctly predicted positive cases among all predicted positive cases, recall (also known as sensitivity) quantifies the proportion of actual positive cases correctly identified, F1-score combines precision and recall into a single metric, and AUC-ROC (Area Under the Receiver Operating Characteristic Curve) represents the trade-off between true positive rate and false positive rate. The average of all metrics for each approach is calculated to assess and compare the performance of the SMOTE and under-sampling approaches using Random Forest. The average is calculated to provide a summary measure that takes multiple metrics into account at the same time and allows for a more comprehensive comparison.

The Random Forest classifier achieved an accuracy of 99.56%, precision of 58.19%, recall of 78.44%, F1-score of 66.81%, and AUC-ROC of 88.06% in the provided results. These metrics show that the model detects fraudulent transactions reasonably well, with a high overall accuracy but some room for improvement in precision and F1-score. The AUC-ROC value indicates a good balance of true and false positive rates. The Random Forest classifier trained on undersampled data achieved an accuracy of 96.28%, precision of 12.53 %, recall of 94.48%, F1 score of 12.53%, and ROC AUC of 95.39% in the provided results.

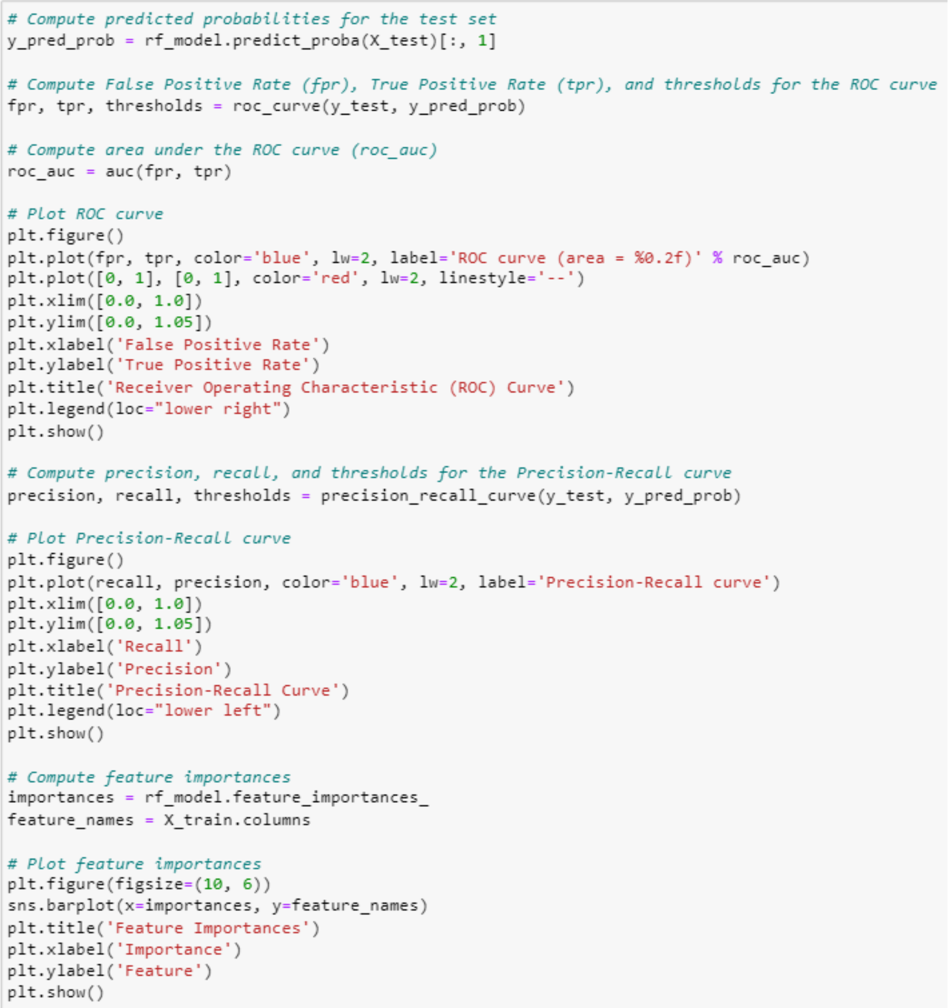
These metrics show that the model detects fraudulent transactions reasonably well, with a high recall (ability to identify most actual fraud cases) but a low precision (potentially high false positive rate). The F1 score is low, indicating a compromise between precision and recall. The average of all performance measure shows that for SMOTE is 78.41% while for under-sampling is 64.16%, indicating that SMOTE technique to handle the imbalance fraud dataset is more suitable for the dataset compared to under-sampling in the Random Forest modelling.

GRAPHS



```python
# Compute predicted probabilities for the test set
y_pred_prob = rf_classifier.predict_proba(X_test)[:, 1]

# Compute False Positive Rate (fpr), True Positive Rate (tpr), and thresholds for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute area under the ROC curve (roc_auc)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Compute precision, recall, and thresholds for the Precision-Recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)

# Plot Precision-Recall curve
plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()

# Compute feature importances
importances = rf_classifier.feature_importances_
feature_names = X_train.columns

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=feature_names)
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

Figure 22 Graphs for SMOTE (RF)



```
# Compute predicted probabilities for the test set
y_pred_prob = rf_model.predict_proba(X_test)[:, 1]

# Compute False Positive Rate (fpr), True Positive Rate (tpr), and thresholds for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute area under the ROC curve (roc_auc)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Compute precision, recall, and thresholds for the Precision-Recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)

# Plot Precision-Recall curve
plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()

# Compute feature importances
importances = rf_model.feature_importances_
feature_names = X_train.columns

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=feature_names)
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

Figure 23 Graphs for Under-sampling (RF)

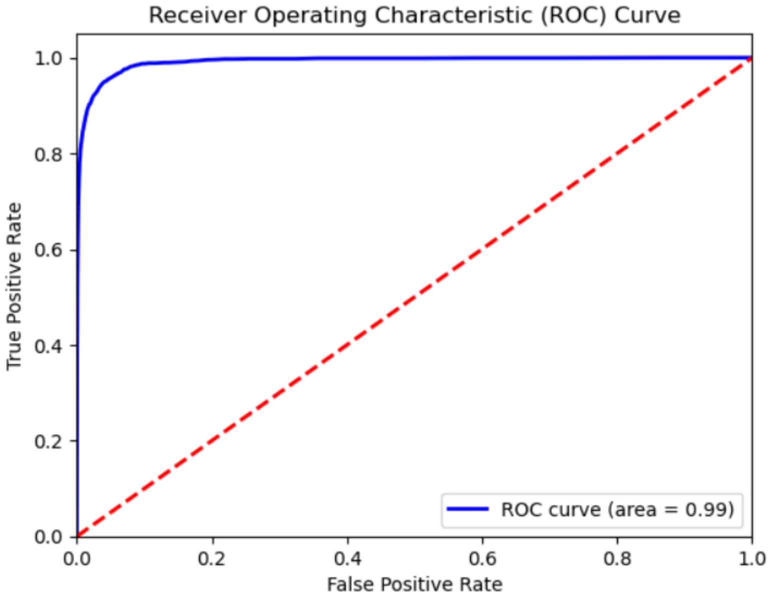


Figure 24 ROC Curve for SMOTE Technique (RF)

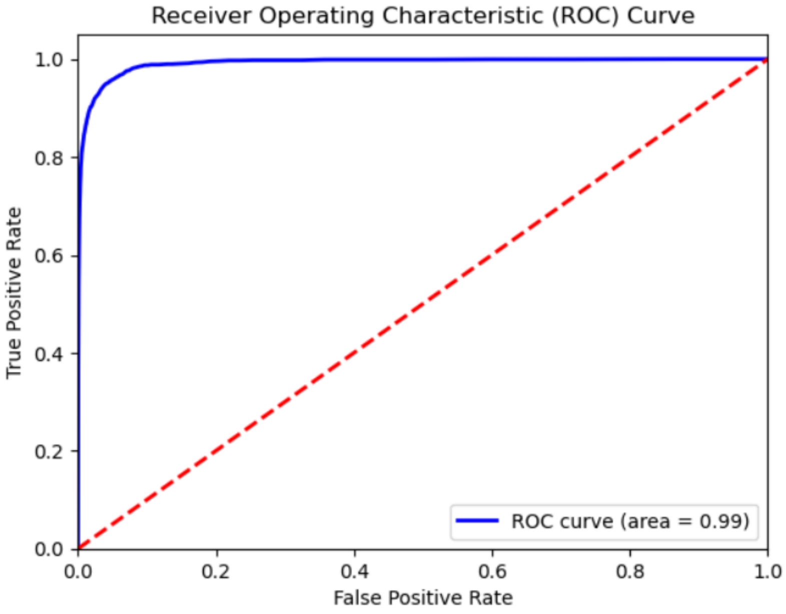


Figure 25 ROC Curve for Under-sampling Technique (RF)

In the context of a Random Forest classifier, the Receiver Operating Characteristic (ROC) curve is a graphical representation that illustrates the performance of the classifier. It shows the trade-off between the True Positive Rate (TPR) and the False Positive Rate (FPR) for different classification thresholds. In the ROC curve, the ideal performance is when the curve is closer to the top-left corner, where the True Positive Rate (TPR) is high, and the False Positive Rate (FPR) is low.

Both of the ROC curve above indicates that the classifier has a high ability to correctly classify positive instances while minimizing the misclassification of negative instances. A higher area under the ROC curve (AUC-ROC) that is closer to 1 represents a better classifier performance in terms of its ability to distinguish between the classes.

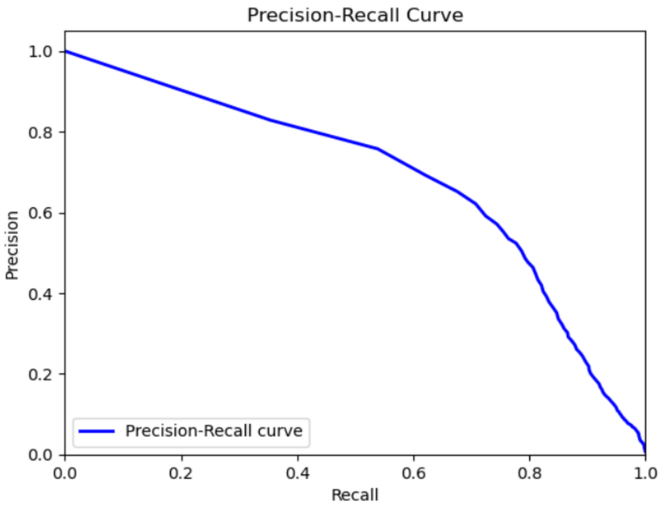


Figure 26 PRC Curve for SMOTE Technique (RF)

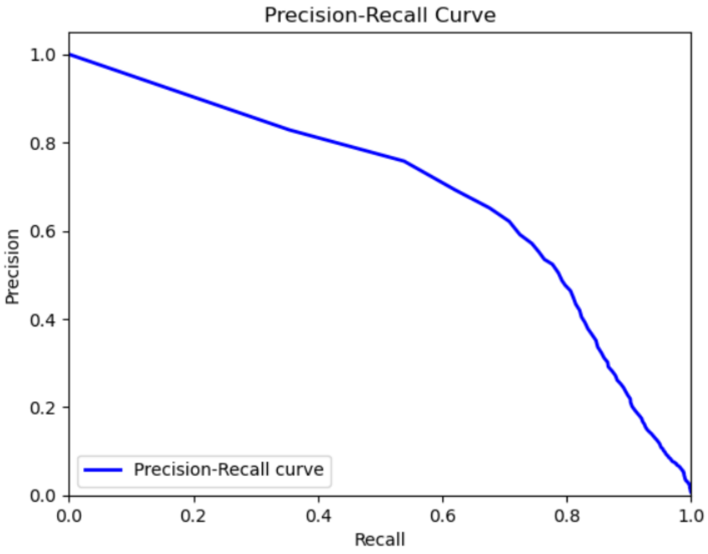


Figure 27 PRC Curve for Under-sampling Technique (RF)

The Precision-Recall Curve (PRC) is another evaluation metric that assesses the performance of the classifier, particularly in scenarios with imbalanced class distributions. The PRC plots the precision (positive predictive value) on the y-axis against the recall (sensitivity or true positive rate) on the x-axis for different classification thresholds. A good PRC exhibits a desirable balance between precision and recall. Higher precision indicates that the positive predictions are more reliable, while higher recall suggests that the classifier can identify a larger portion of positive instances. Overall, the curve for SMOTE technique and under-sampling technique did not differ much. Both of the curve shown above evident that it has a higher precision and higher recall sensitivity.

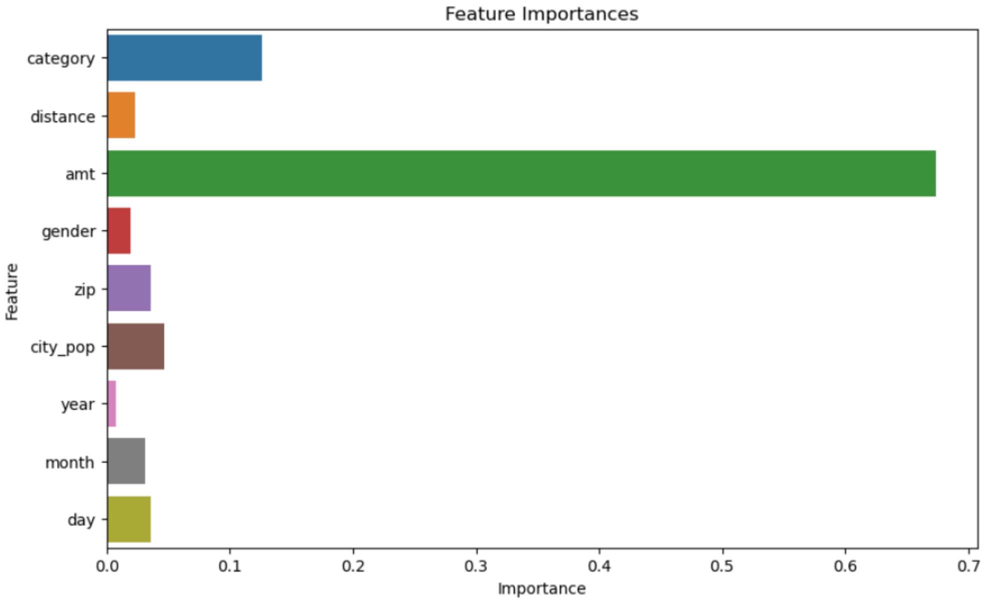


Figure 28Feature Importance for SMOTE Technique (RF)

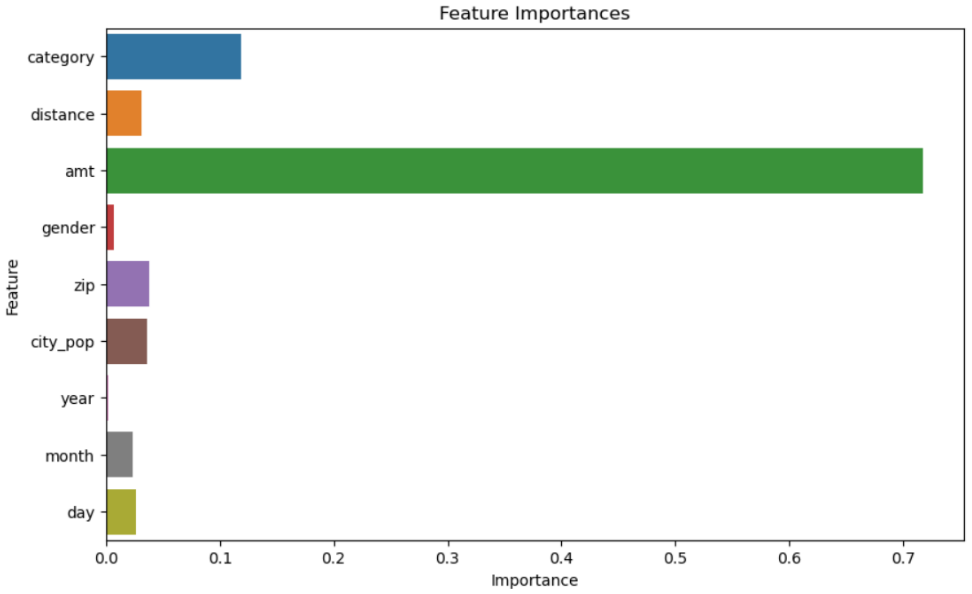


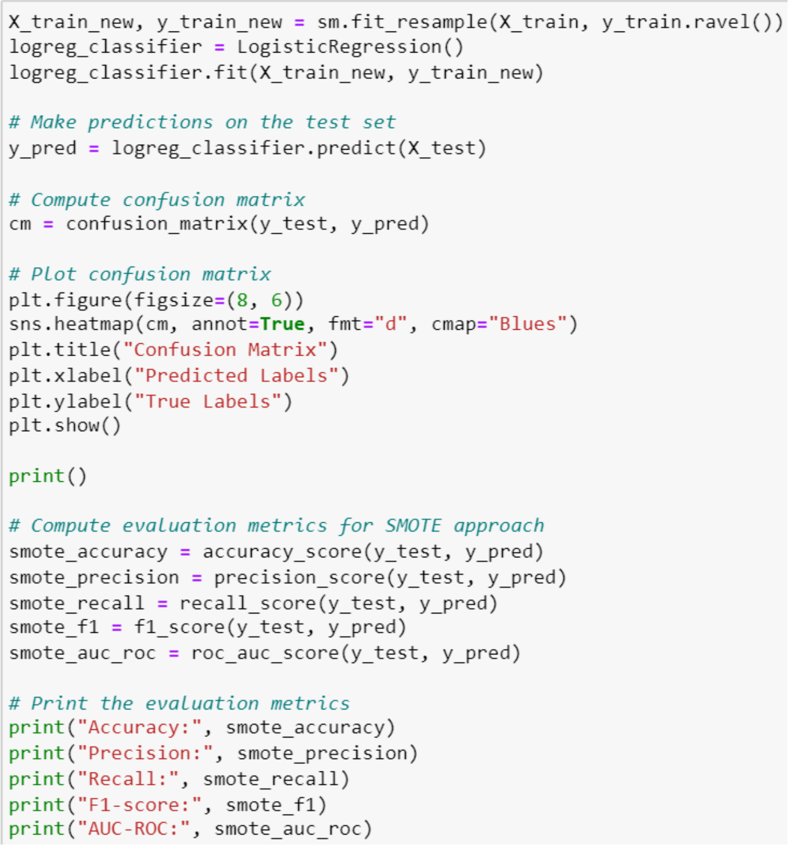
Figure 29 Feature Importance for Under-sampling Technique (RF)

Feature importance refers to a measure of the relative significance or contribution of each feature in the classification task. It provides insights into which features have the most influence on the model's predictions. Random Forest calculates feature importance based on the following principle when a feature is used to split a tree node, it measures how much the impurity (e.g., Gini impurity or entropy) of the resulting child nodes decreases compared to the parent node. The more the impurity decreases, the higher the importance of that feature. In both of the techniques, the most influential feature in the random forest classifier is amount (amt) while the year feature does not affect the analysis much. Moreover, in SMOTE, city_pop variable contributes a bit more to the analysis compared to it is in under-sampling.

LOGISTIC REGRESSION

After modelling using Random Forest, the Logistic Regression model was modelled using the same two techniques to handle the imbalance fraud detection dataset, SMOTE (Synthetic Minority Over-sampling Technique) and under-sampling. Both techniques were applied repeatedly to assess their impact on the model's performance. The purpose of this section is to compare the results obtained from the application of these two techniques. By examining and contrasting the outcomes, we aim to gain insights into the effectiveness and suitability of each technique in addressing the imbalanced class distribution present in the dataset.

SMOTE



```
X_train_new, y_train_new = sm.fit_resample(X_train, y_train.ravel())
logreg_classifier = LogisticRegression()
logreg_classifier.fit(X_train_new, y_train_new)

# Make predictions on the test set
y_pred = logreg_classifier.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

print()

# Compute evaluation metrics for SMOTE approach
smote_accuracy = accuracy_score(y_test, y_pred)
smote_precision = precision_score(y_test, y_pred)
smote_recall = recall_score(y_test, y_pred)
smote_f1 = f1_score(y_test, y_pred)
smote_auc_roc = roc_auc_score(y_test, y_pred)

# Print the evaluation metrics
print("Accuracy:", smote_accuracy)
print("Precision:", smote_precision)
print("Recall:", smote_recall)
print("F1-score:", smote_f1)
print("AUC-ROC:", smote_auc_roc)
```

Figure 30 Train & Test for Logistic Regression (SMOTE)

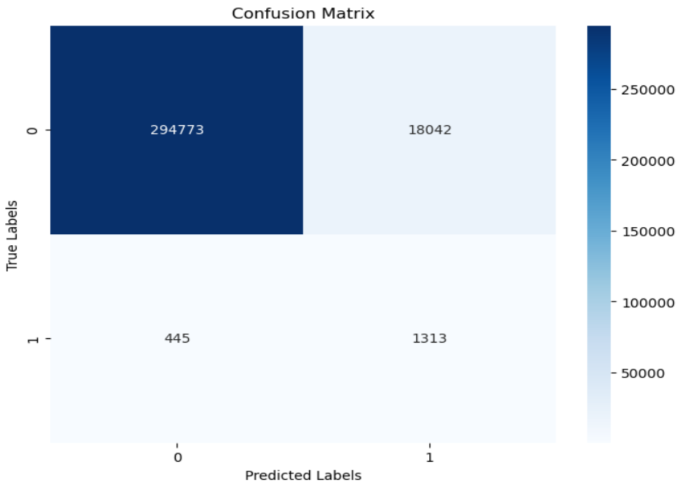
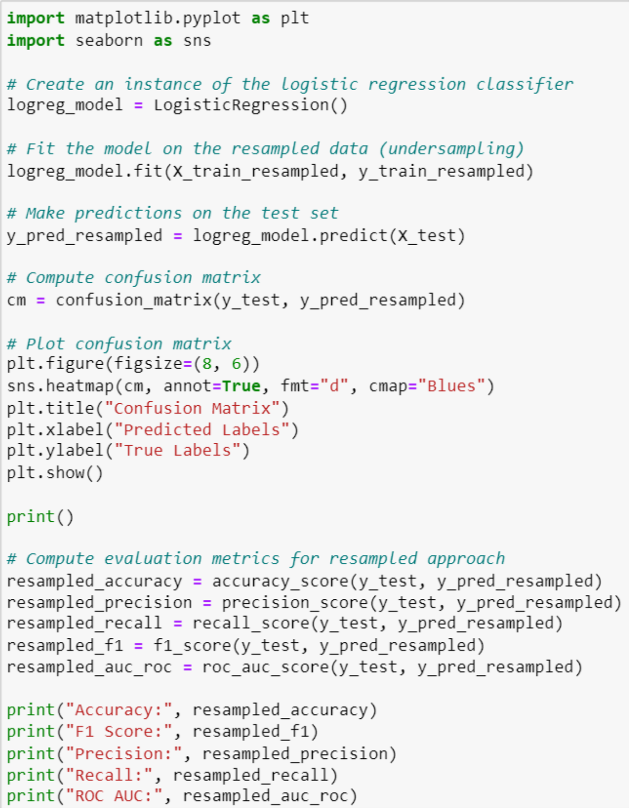


Figure 31 Confusion Matrix for SMOTE Technique (LR)

Using the logistic regression model, predictions are made on the test set (X_test) after the dataset has been trained. The confusion matrix is then printed to assess the model's performance. The number of true negatives (294773), false positives (18042), false negatives (445), and true positives (1313) are all displayed in the confusion matrix.

UNDERSAMPLING



```python
import matplotlib.pyplot as plt
import seaborn as sns

# Create an instance of the logistic regression classifier
logreg_model = LogisticRegression()

# Fit the model on the resampled data (undersampling)
logreg_model.fit(X_train_resampled, y_train_resampled)

# Make predictions on the test set
y_pred_resampled = logreg_model.predict(X_test)

# Compute confusion matrix
cm = confusion_matrix(y_test, y_pred_resampled)

# Plot confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()

print()

# Compute evaluation metrics for resampled approach
resampled_accuracy = accuracy_score(y_test, y_pred_resampled)
resampled_precision = precision_score(y_test, y_pred_resampled)
resampled_recall = recall_score(y_test, y_pred_resampled)
resampled_f1 = f1_score(y_test, y_pred_resampled)
resampled_auc_roc = roc_auc_score(y_test, y_pred_resampled)

print("Accuracy:", resampled_accuracy)
print("F1 Score:", resampled_f1)
print("Precision:", resampled_precision)
print("Recall:", resampled_recall)
print("ROC AUC:", resampled_auc_roc)
```

Figure 32 Train & Test for Logistic Regression (Under-sampling)

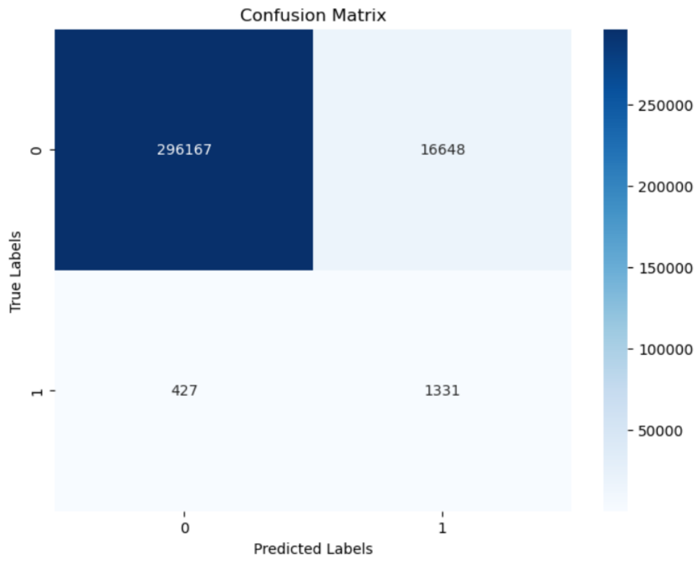


Figure 33 Confusion Matrix for Under-sampling Technique (LR)

The same steps were repeated using under-sampling dataset. The confusion matrix is then printed to assess the model's performance. The number of true negatives (296167), false positives (16648), false negatives (427), and true positives (1331) are all displayed in the confusion matrix.

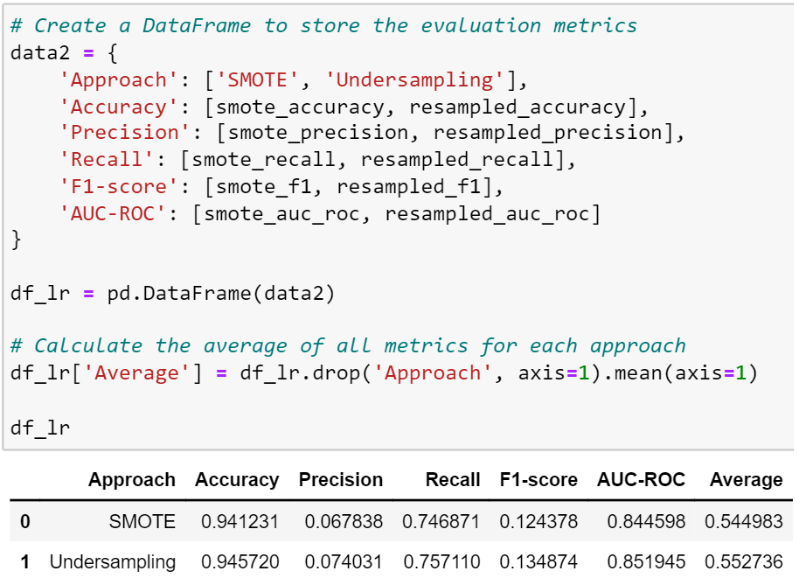
COMPARISON (SMOTE & UNDERSAMPLING)

CONFUSION MATRIX

Upon comparison, the second set of results demonstrates a slightly better performance in terms of the confusion matrix metrics. It has a higher count of true negatives (296,167) compared to the first set (294,773), indicating a better ability to correctly predict the negative class. Additionally, the second set has a lower count of false positives (16,648) compared to the first set (18,042), suggesting a lower false positive rate. Regarding false negatives, the second set (427) has a lower count compared to the first set (445), indicating a better ability to correctly predict the positive class. Both sets have a similar count of true positives, with the second set (1,331) slightly higher than the first set (1,313).

Based on the provided information, the second set of results appears to demonstrate a slightly better overall performance in terms of correctly predicting the negative class, reducing false positives, and achieving a slightly lower false negative count. However, a comprehensive evaluation using multiple metrics is recommended to make a more conclusive assessment.

PERFORMANCE MEASURES



```python
# Create a DataFrame to store the evaluation metrics
data2 = {
    'Approach': ['SMOTE', 'Undersampling'],
    'Accuracy': [smote_accuracy, resampled_accuracy],
    'Precision': [smote_precision, resampled_precision],
    'Recall': [smote_recall, resampled_recall],
    'F1-score': [smote_f1, resampled_f1],
    'AUC-ROC': [smote_auc_roc, resampled_auc_roc]
}

df_lr = pd.DataFrame(data2)

# Calculate the average of all metrics for each approach
df_lr['Average'] = df_lr.drop('Approach', axis=1).mean(axis=1)

df_lr
```

	Approach	Accuracy	Precision	Recall	F1-score	AUC-ROC	Average
0	SMOTE	0.941231	0.067838	0.746871	0.124378	0.844598	0.544983
1	Undersampling	0.945720	0.074031	0.757110	0.134874	0.851945	0.552736

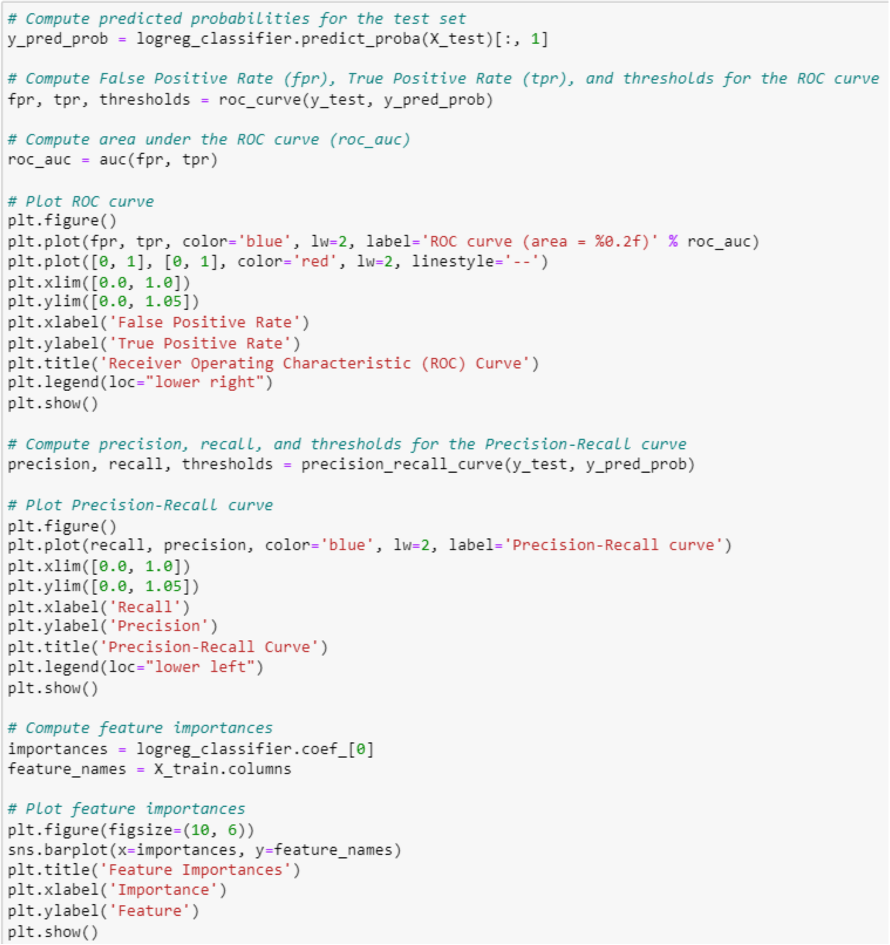
Figure 34 Comparison between SMOTE and Undersampling Techniques Result (LR)

Additional evaluation metrics such as accuracy, precision, recall, and F1-score should be considered to obtain a more comprehensive understanding. Figure 34 shows the comparison between the SMOTE and Under-sampling Techniques results in context of Accuracy, Precision, Recall, F1-score, AUC-ROC and also their average. Upon comparison, the Under-sampling approach yields slightly better results across several metrics. It achieves a higher accuracy (94.57%) compared to SMOTE (94.12%), indicating a better overall correct classification rate. Additionally, Under-sampling exhibits a higher precision (7.40%) compared to SMOTE (6.78%), suggesting a lower false positive rate.

Regarding recall, Under-sampling (75.71%) outperforms SMOTE (74.68%) by capturing a higher proportion of actual positive cases. The F1-Score for Under-sampling (13.48%) is also higher than SMOTE (12.43%), indicating a better balance between precision and recall. Both approaches demonstrate relatively high AUC-ROC values, with Under-sampling (85.19%) slightly surpassing SMOTE (84.50%). This suggests a good trade-off between true positive and false positive rates for both scenarios.

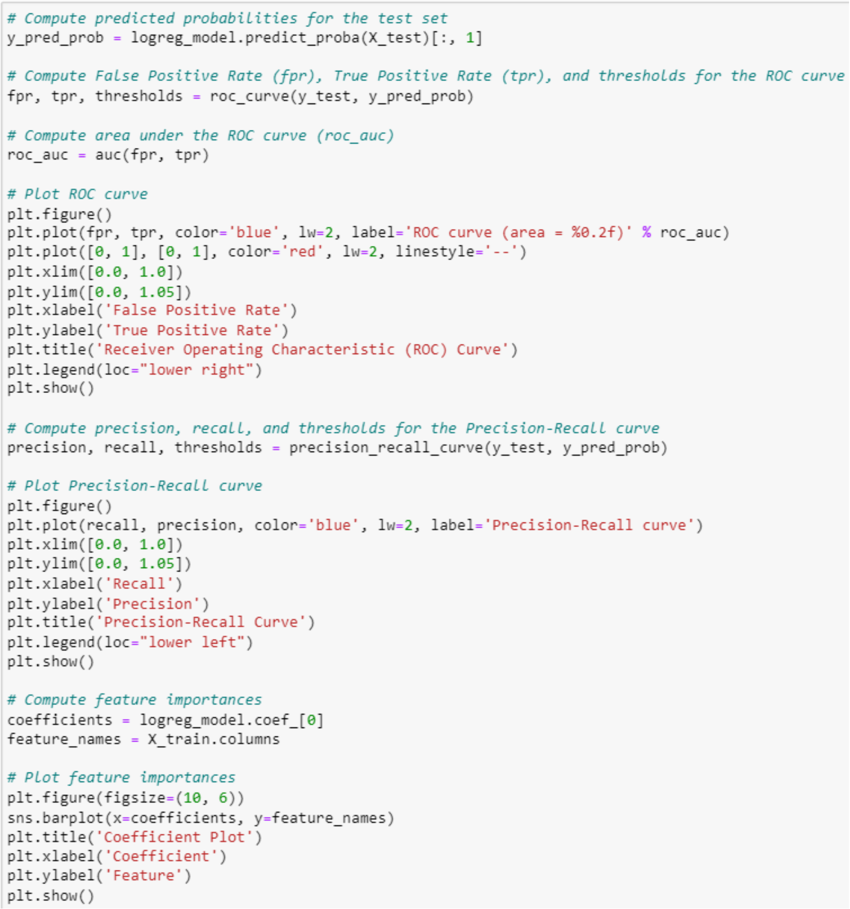
Based on these comparisons, the Under-sampling approach generally exhibits better overall performance in terms of accuracy, precision, recall, F1-Score, and AUC-ROC. The average of all performance measure shows that for SMOTE is 54.50% while for under-sampling is 55.27 %, indicating that under-sampling technique to handle the imbalance fraud dataset is more suitable for the dataset compared to under-sampling in the Logistic Regression modelling, which contradicts the results we obtained for Random Forest modelling.

GRAPHS



```
# Compute predicted probabilities for the test set
y_pred_prob = logreg_classifier.predict_proba(X_test)[:, 1]

# Compute False Positive Rate (fpr), True Positive Rate (tpr), and thresholds for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute area under the ROC curve (roc_auc)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Compute precision, recall, and thresholds for the Precision-Recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)

# Plot Precision-Recall curve
plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()

# Compute feature importances
importances = logreg_classifier.coef_[0]
feature_names = X_train.columns

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=importances, y=feature_names)
plt.title('Feature Importances')
plt.xlabel('Importance')
plt.ylabel('Feature')
plt.show()
```

Figure 35 Graphs for SMOTE (LR)



```
# Compute predicted probabilities for the test set
y_pred_prob = logreg_model.predict_proba(X_test)[:, 1]

# Compute False Positive Rate (fpr), True Positive Rate (tpr), and thresholds for the ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Compute area under the ROC curve (roc_auc)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure()
plt.plot(fpr, tpr, color='blue', lw=2, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], color='red', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc="lower right")
plt.show()

# Compute precision, recall, and thresholds for the Precision-Recall curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_prob)

# Plot Precision-Recall curve
plt.figure()
plt.plot(recall, precision, color='blue', lw=2, label='Precision-Recall curve')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()

# Compute feature importances
coefficients = logreg_model.coef_[0]
feature_names = X_train.columns

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=coefficients, y=feature_names)
plt.title('Coefficient Plot')
plt.xlabel('Coefficient')
plt.ylabel('Feature')
plt.show()
```

Figure 36 Graphs for Under-sampling (LR)

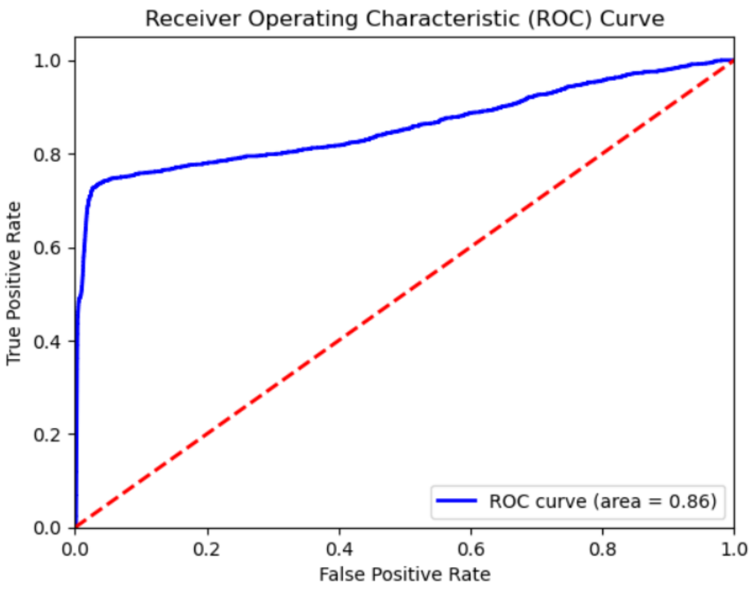


Figure 37 ROC Curve for SMOTE Technique (LR)

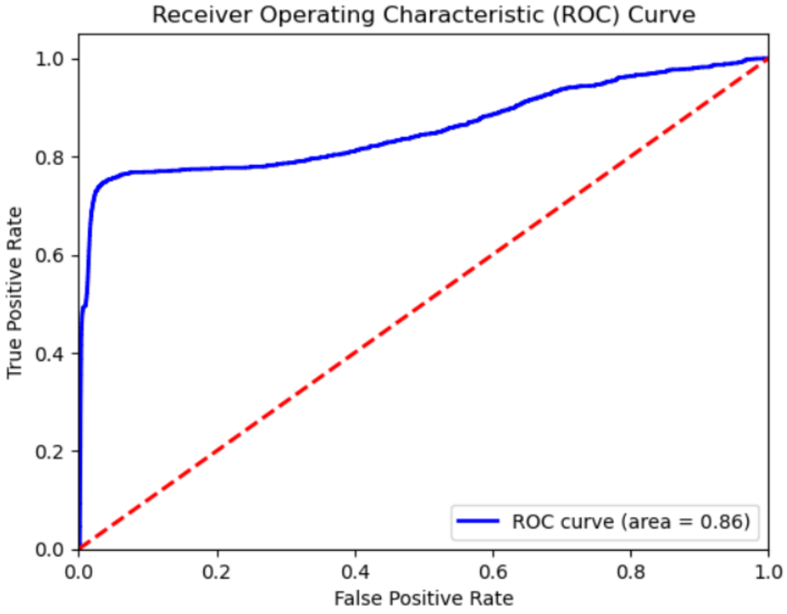


Figure 38 ROC Curve for Under-sampling Technique (LR)

Both of the ROC curve for SMOTE and under-sampling technique are not much differ. Although the ROC Curve slightly below than 1.0, it still indicates that the classifier has a high ability to correctly classify positive instances while minimizing the misclassification of negative instances.

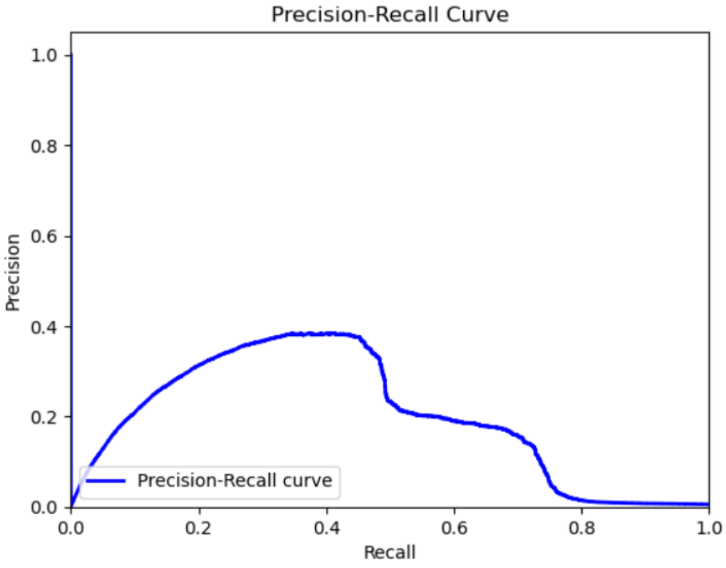


Figure 39 PRC Curve for SMOTE Technique (LR)

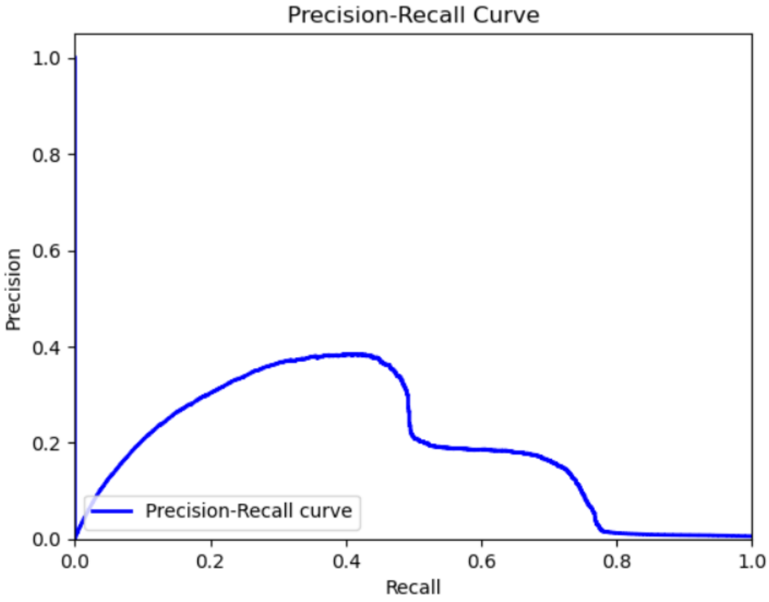


Figure 40 PRC Curve for Under-sampling Technique (LR)

Both of the PRC Curve for the techniques records a low precision and low recall threshold. The combination of low precision and low recall indicates that the model struggles to correctly identify and classify positive instances. It is likely that the model is biased towards the majority class or faces challenges in distinguishing the minority class due to imbalanced data or other factors. It is crucial to carefully analyse the Precision-Recall curve, evaluate other evaluation metrics, and consider the specific requirements and constraints of the problem domain to devise appropriate strategies for improving the precision and recall of the model.

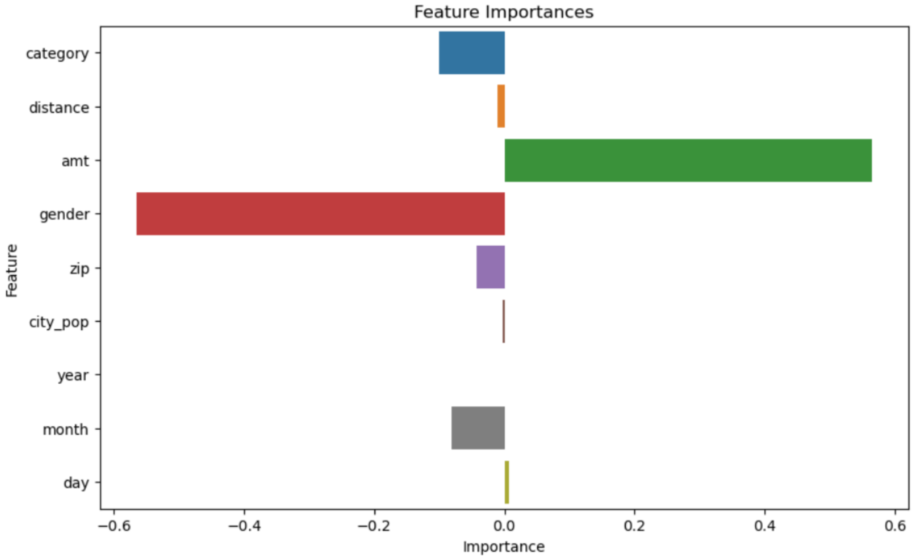


Figure 41 Feature Importance for SMOTE Technique (LR)

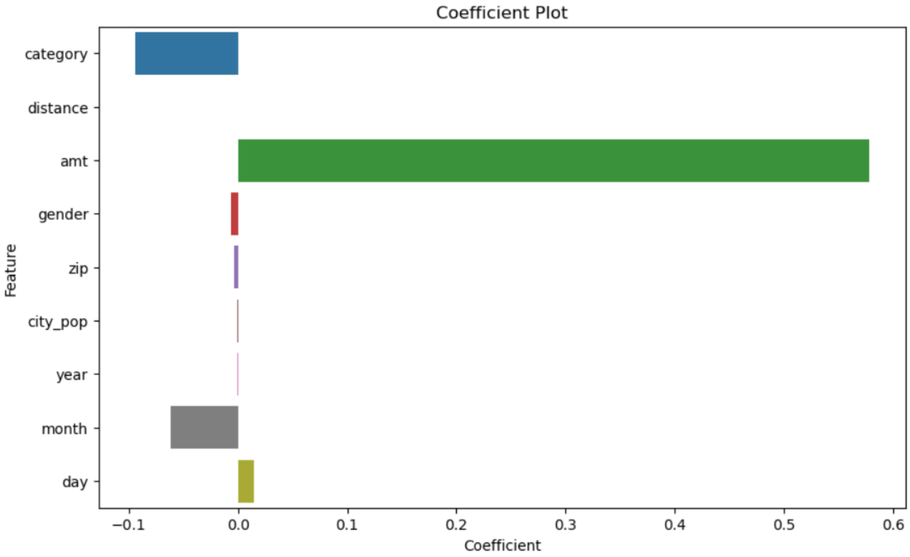


Figure 42 Feature Importance for Under-sampling Technique (LR)

In logistic regression, the coefficients associated with each feature can be interpreted as a measure of the feature's importance. However, the coefficients themselves may not directly indicate the importance of a feature in a positive or negative sense. The magnitude and direction of the coefficients provide information about the strength and direction of the relationship between each feature and the target variable. In logistic regression, a positive coefficient means that an increase in the feature's value leads to an increase in the predicted probability of the positive class. Conversely, a negative coefficient means that an increase in the feature's value leads to a decrease in the predicted probability of the positive class.

In the Figure 42 shown above, feature importance for SMOTE technique indicates that for the "amount" feature, the positive coefficient suggests that an increase in the value of "amount" leads to an increase in the predicted probability of the positive class. This means that higher values of "amount" are associated with a higher likelihood of the positive outcome or event being predicted by the logistic regression model. On the other hand, the negative coefficient for the "gender" feature indicates that an increase in the value of "gender" leads to a decrease in the predicted probability of the positive class. This means that certain categories in the "gender" feature are associated with a lower likelihood of the positive outcome being predicted. However, for under-sampling technique, “category” variable recorded the higher negative coefficient while “amount” remains the highest positive coefficient. These results highlight the significance of both the "amount" and "category" features in the logistic regression model, albeit with different effects on the predicted probabilities. Further analysis and consideration of other factors such as statistical significance and model performance metrics would provide a more comprehensive understanding of the feature importance and their implications for the model.

COMPARISON - RANDOM FOREST AND LOGISTIC REGRESSION MODEL

SMOTE

CONFUSION MATRIX

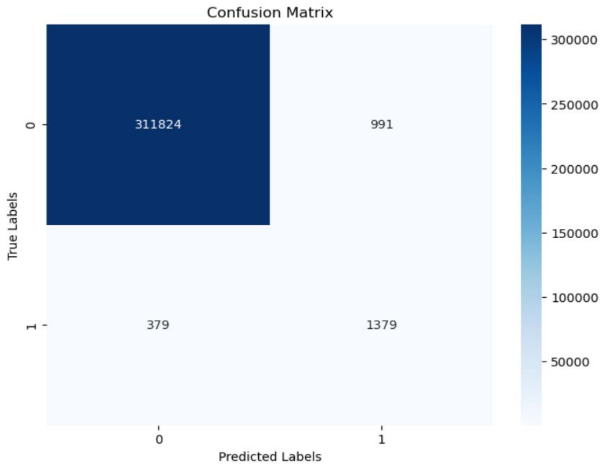


Figure 43 Confusion Matrix for SMOTE Technique (RF)

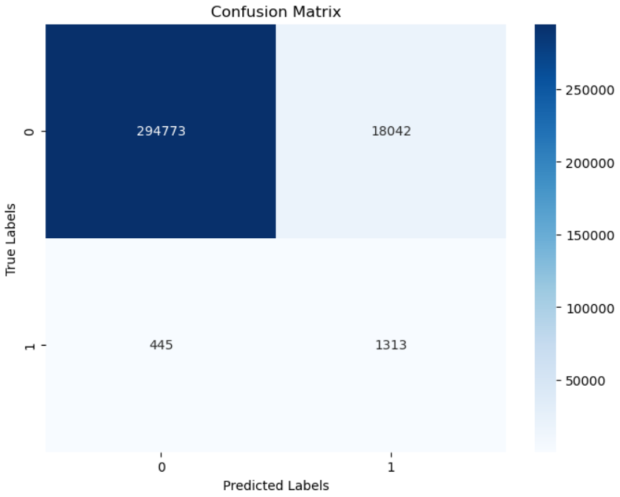
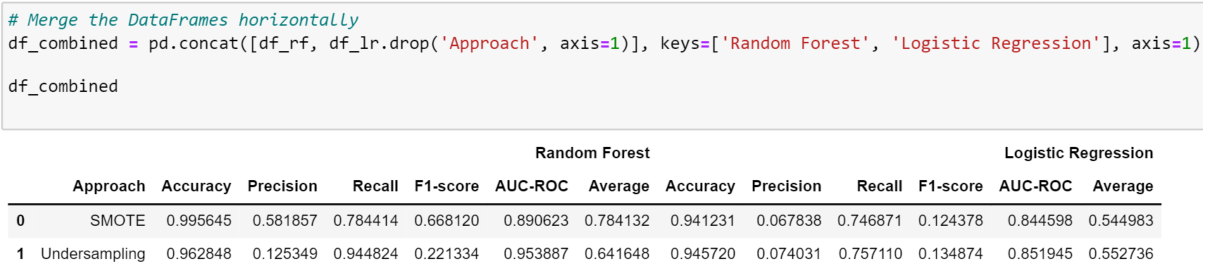


Figure 44 Confusion Matrix for SMOTE Technique (LR)

In the Random Forest classifier, we observe a higher number of true negatives (311,824) compared to the logistic regression model (294,773). On the other hand, the logistic regression model has a higher number of false positives (18,042) than the Random Forest classifier (991). This indicates that the Random Forest classifier performed better in correctly identifying the negative class instances, while the logistic regression model had a higher number of false positive errors. The Random Forest classifier had a smaller number of false negatives (379) compared to the logistic regression model (445), indicating that the Random Forest model had a better performance in correctly identifying positive class instances. However, the logistic regression model had a higher number of true positives (1,313) compared to the Random Forest classifier (1,379).

In summary, the Random Forest classifier showed better performance in terms of true negatives and false negatives, while the logistic regression model had higher true positives but a higher number of false positives. Further analysis using different evaluation metrics can provide a more comprehensive understanding of the models' performance and help in selecting the most suitable model for the given problem. To further assess the overall performance of these models, additional metrics such as accuracy, precision, recall, and F1-score should be considered.

PERFORMANCE MEASURES



```
# Merge the DataFrames horizontally
df_combined = pd.concat([df_rf, df_lr.drop('Approach', axis=1)], keys=['Random Forest', 'Logistic Regression'], axis=1)

df_combined
```

		Random Forest						Logistic Regression					
	Approach	Accuracy	Precision	Recall	F1-score	AUC-ROC	Average	Accuracy	Precision	Recall	F1-score	AUC-ROC	Average
0	SMOTE	0.995645	0.581857	0.784414	0.668120	0.890623	0.784132	0.941231	0.067838	0.746871	0.124378	0.844598	0.544983
1	Undersampling	0.962848	0.125349	0.944824	0.221334	0.953887	0.641648	0.945720	0.074031	0.757110	0.134874	0.851945	0.552736

Figure 45 Performance Measure summary

Figure 45 above shows the summary of accuracy, precision, recall, F1-score, AUC-ROC, and average for both Random Forest and Logistic Regression model. When comparing the performance of the Random Forest classifier and the Logistic Regression model, notable differences can be observed across multiple metrics. The Random Forest classifier achieves impressive results, with an accuracy of 99.56%. This indicates that the model makes highly accurate predictions on the test set. In contrast, the Logistic Regression model achieves an accuracy of 94.12%, which is considerably lower than the Random Forest classifier. Looking at precision, the Random Forest classifier outperforms the Logistic Regression model significantly, with a precision of 58.18% compared to a mere 6.78%. The Random Forest classifier demonstrates a stronger ability to correctly identify positive class instances, as reflected by its higher precision score. Similarly, the Random Forest classifier shows higher recall (78.44%) compared to the Logistic Regression model (74.68%). This means that the Random Forest model is better at identifying true positive instances from the positive class, capturing a larger proportion of the positive cases.

Considering the F1-score, which balances precision and recall, the Random Forest classifier again outperforms the Logistic Regression model. The Random Forest classifier achieves an F1-score of 66.81%, indicating a better overall trade-off between precision and recall. In contrast, the Logistic Regression model lags behind with an F1-score of 12.43%. Furthermore, when assessing the models' discrimination ability, the Random Forest classifier shows a higher

AUC-ROC score of 89.06%, while the Logistic Regression model achieves a score of 84.46%. This suggests that the Random Forest classifier exhibits better discriminative power in distinguishing between positive and negative instances. Overall, the Random Forest classifier surpasses the Logistic Regression model in all evaluated metrics, including accuracy, precision, recall, F1-score, and AUC-ROC. These results highlight the superior performance and predictive capability of the Random Forest model in comparison to the Logistic Regression model for the given problem and dataset.

UNDERSAMPLING

CONFUSION MATRIX

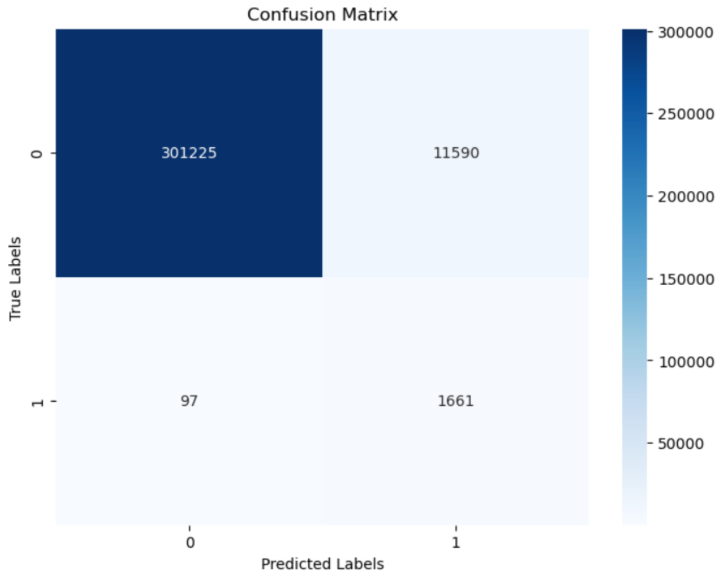


Figure 46 Confusion Matrix for Under-sampling Technique (RF)

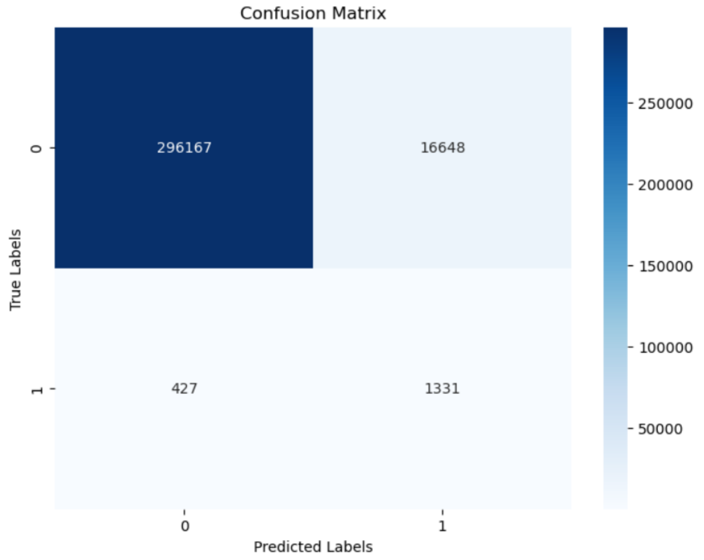
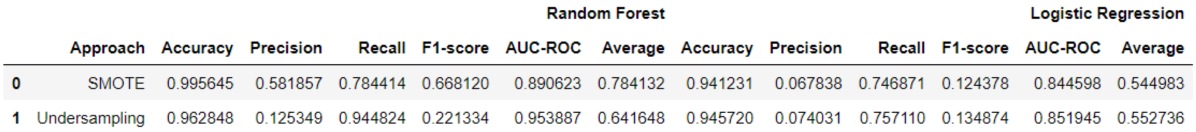


Figure 47 Confusion Matrix for Under-sampling Technique (LR)

Examining the true negatives (TN) and false positives (FP), we observe that the Random Forest classifier achieves a higher number of true negatives (301,225) compared to the Logistic Regression model (296,167). Conversely, the Logistic Regression model yields a greater number of false positives (16,648) than the Random Forest classifier (11,590). This indicates that the Random Forest classifier performs better in correctly identifying negative class instances, while the Logistic Regression model has a higher rate of false positive errors.

In terms of false negatives (FN) and true positives (TP), the Random Forest classifier demonstrates a lower number of false negatives (97) compared to the Logistic Regression model (427). This suggests that the Random Forest model excels in correctly identifying positive class instances. However, the Logistic Regression model exhibits a higher number of true positives (1,331) in comparison to the Random Forest classifier (1,661). In summary, the Random Forest classifier showcases better performance in terms of true negatives and false negatives, while the Logistic Regression model yields higher true positives but a larger number of false positives. To gain a comprehensive understanding of the models' performance, we will further compare the accuracy, precision, recall, F1-score, AUC-ROC, and average for both Random Forest and Logistic Regression model.

PERFORMANCE MEASURES



		Random Forest						Logistic Regression					
	Approach	Accuracy	Precision	Recall	F1-score	AUC-ROC	Average	Accuracy	Precision	Recall	F1-score	AUC-ROC	Average
0	SMOTE	0.995645	0.581857	0.784414	0.668120	0.890623	0.784132	0.941231	0.067838	0.746871	0.124378	0.844598	0.544983
1	Undersampling	0.962848	0.125349	0.944824	0.221334	0.953887	0.641648	0.945720	0.074031	0.757110	0.134874	0.851945	0.552736

Figure 48 Performance measure summary

The Random Forest classifier demonstrates superior performance with an accuracy of 96.28%, outperforming the Logistic Regression model, which achieves an accuracy of 94.57%. This indicates that the Random Forest model makes more accurate overall predictions on the test set. Examining precision, the Random Forest classifier achieves a higher precision rate of 12.53% compared to the Logistic Regression model's precision of 7.40%. The Random Forest model displays a better ability to correctly identify positive class instances, as evidenced by its higher precision score. Furthermore, the Random Forest classifier significantly outperforms the Logistic Regression model in terms of recall. The Random Forest model achieves an impressive recall rate of 94.48%, while the Logistic Regression model achieves a lower recall rate of 75.71%. This indicates that the Random Forest model is more effective at capturing true positive instances from the positive class.

Considering the F1-score, which balances precision and recall, the Random Forest classifier again outperforms the Logistic Regression model. The Random Forest model achieves an F1-score of 22.13%, highlighting its ability to strike a better balance between precision and recall. In contrast, the Logistic Regression model achieves an F1-score of 13.49%. In terms of discrimination ability, the Random Forest classifier showcases a higher AUC-ROC score of

95.39%, while the Logistic Regression model achieves a score of 85.19%. This signifies that the Random Forest classifier excels in distinguishing between positive and negative instances, exhibiting stronger discriminative power. In brief, the Random Forest classifier outperforms the Logistic Regression model across all evaluated metrics, including accuracy, precision, recall, F1-score, and AUC-ROC. These results indicate that the Random Forest model is a more effective classifier for the given problem and dataset, offering improved accuracy, precision, recall, and discrimination ability compared to the Logistic Regression model.

CONCLUSION

After conducting a thorough comparison between the Random Forest classifier and the Logistic Regression model, using both the SMOTE and under-sampling techniques, it becomes evident that the Random Forest classifier consistently outperforms the Logistic Regression model across multiple evaluation metrics. Let's delve into the details.

When applying the SMOTE technique, the Random Forest classifier showcases superior performance in various aspects. It achieves higher accuracy, precision, and recall rates compared to the Logistic Regression model. The Random Forest classifier also demonstrates a better balance between precision and recall, as indicated by its higher F1-score. Moreover, it exhibits better discriminative power in distinguishing between positive and negative instances, as reflected by its higher AUC-ROC score.

Similarly, when employing the under-sampling technique, the Random Forest classifier outperforms the Logistic Regression model in terms of true negatives and false negatives. It excels in correctly identifying negative class instances and has a lower number of false negatives. Although the Logistic Regression model performs better in true positives, the Random Forest classifier surpasses it when evaluating accuracy, precision, recall, F1-score, and AUC-ROC.

Based on these comparisons, it can be concluded that the Random Forest classifier consistently demonstrates superior performance and predictive capability compared to the Logistic Regression model. It achieves higher accuracy, precision, recall, and F1-score, while also exhibiting better discriminative power.

Moreover, the specific evaluation metrics chosen for assessing fraud detection performance may vary depending on the task requirements. The F1-score is a widely used metric in fraud detection as it provides a balanced measure, considering both precision and recall. It combines the model's ability to correctly identify positive cases (fraud) while minimizing false positives.

In the case of the SMOTE approach, the average F1-score is 0.783, while for the under-sampling approach, it is 0.641. This indicates that the SMOTE approach with the Random

Forest classifier outperforms in terms of F1-score. Additionally, the SMOTE approach yields a higher average of 78.30% compared to the under-sampling approach.

However, it is crucial to consider other factors and specific requirements of the fraud detection task before determining the best approach for handling imbalance in the data. The choice of the ideal model depends on the acceptable trade-off between precision and recall within the application context. For instance, under-sampling may be preferred if the aim is to minimize false positives (precision), while the SMOTE approach may be more suitable for capturing as many fraud cases as possible (recall).

In conclusion, the Random Forest classifier is recommended as the preferred model for the given problem and dataset, regardless of whether the SMOTE or under-sampling technique is employed. It consistently outperforms the Logistic Regression model in terms of accuracy, precision, recall, F1-score, and AUC-ROC, making it a reliable choice for fraud detection tasks.