CSE422 Lab Project Report

Section: 5

Project Name: Online Payments Fraud Detection

Submitted By:

Name: Junaid Ahmed Sifat	Student Id: 19201066
Name: Md.Waseq Alauddin Alvi	Student Id: 20101153
Name: Farhan Ishraq Fagun	Student Id: 20101295
Name: Md. Irtiza Hossain	Student Id: 20101481

Submitted To:

Syed Zamil Hasan (SZH)

Md. Nowroz Junaed Rahman

April 28, 2023

TABLE OF CONTENTS

List of Tables

List of Figures

1. INTRODUCTION

Online payment fraud is a growing problem that affects individuals, businesses, and financial institutions worldwide. Fraudulent transactions can result in significant financial losses, damage to reputation, and can cause disruptions to operations. Therefore, the ability to accurately detect and prevent fraudulent transactions is of utmost importance.

The dataset available on Kaggle provides an excellent opportunity to address this issue, as it contains a diverse range of transactional data that can be used to build a robust fraud detection model. By leveraging machine learning algorithms and techniques, we aim to build a model that can accurately identify fraudulent transactions in real-time, providing valuable insights that can be used to prevent further fraudulent activity.

Our project is not only focused on developing a model that can accurately detect fraudulent transactions but also on contributing to the larger effort to combat online payment fraud. By sharing our findings and insights, we hope to raise awareness about the issue and inspire others to develop similar solutions. Ultimately, our goal is to help create a safer and more secure online payment ecosystem for all.

2. Dataset Description

The Kaggle-available Fraud Detection dataset is a collection of transactional data connected to online payments. There are 129,667 rows of data in the collection, with each row representing a unique transaction. The dataset comprises information such as the transaction amount, payment method, client details, and whether or not the transaction was fraudulent. To protect the privacy and security of the persons and enterprises concerned, the data has been anonymized and processed. This dataset is a significant resource for developing machine learning models capable of properly recognizing fraudulent transactions, helping to contribute to the larger effort to prevent online payment fraud.

2.1. Source

- Source: Kaggle (https://www.kaggle.com)
- Link: https://www.kaggle.com/datasets/kartik2112/fraud-detection
- Reference: Kartik Ahuja (2020). Fraud Detection, Kaggle dataset.

2.2. Dataset Description

In this dataset, we have a total of 23 features. Among them some of the most important features are

- Transaction ID: A unique identifier for each transaction.
- Transaction Date and Time: The date and time when the transaction occurred.
- Transaction Amount: The amount of the transaction in US dollars.
- Card Type: The type of card used for the transaction (Visa, Mastercard, etc.).
- Card Number: The first six and last four digits of the card used for the transaction.
- Shop Name: Name of the online store where the transaction occurred.
- Purchase Type: Type of purchase made (online, phone, etc.)
- Fraudulent: Binary value indicating whether the transaction was fraudulent (1) or not (0).

There are a total of 129,667 rows of data (data points) in our dataset. Here the Fraudulent (isfraud) feature is the deciding value or target value for our machine learning models. As we are trying to find if a transaction is a fraud or not so we say we are solving a classification problem.

Among the 23 features in our dataset, 12 features have categorical values and the rest are quantitative values.

```
In [3]:  1 dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1296675 entries, 0 to 1296674
Data columns (total 23 columns):
 #   Column                 Non-Null Count    Dtype
---  ------                 --------------    -----
 0   Unnamed: 0             1296675 non-null  int64
 1   trans_date_trans_time  1296675 non-null  object
 2   cc_num                 1296675 non-null  int64
 3   merchant               1296675 non-null  object
 4   category               1296675 non-null  object
 5   amt                    1295371 non-null  float64
 6   first                  1296675 non-null  object
 7   last                   1296675 non-null  object
 8   gender                 1296675 non-null  object
 9   street                 1296675 non-null  object
 10  city                   1296675 non-null  object
 11  state                  1296675 non-null  object
 12  zip                    1296675 non-null  int64
 13  lat                    1296675 non-null  float64
 14  long                   1296675 non-null  float64
 15  city_pop               1296675 non-null  int64
 16  job                    1296675 non-null  object
 17  dob                    1296675 non-null  object
 18  trans_num              1296675 non-null  object
 19  unix_time              1296675 non-null  int64
 20  merch_lat              1296675 non-null  float64
 21  merch_long             1296675 non-null  float64
 22  is_fraud               1296675 non-null  int64
dtypes: float64(5), int64(6), object(12)
memory usage: 227.5+ MB
```

Figure 2.1: Dataset Features

We tried to find the correlation between all the features by applying a heatmap using the Seaborn library.

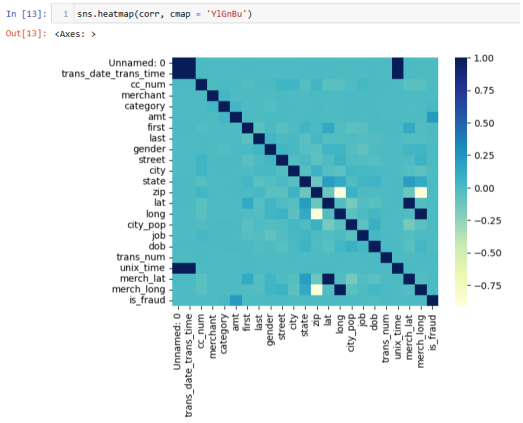

Figure 2.2: Heatmap of the Imbalanced Dataset

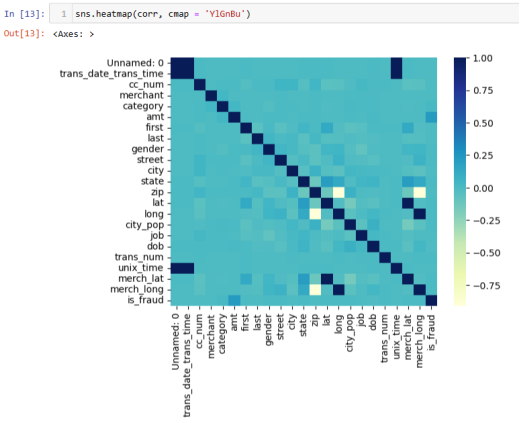

Figure 2.3: Heatmap of the Under Samples Dataset

2.3. Imbalanced Dataset

We are solving a classification problem and among the total of 129,667 rows of data (data points) in our dataset only 7506 rows are marked as fraud (class 1) and 1289189 rows are marked as genuine (class 0). To overcome this imbalance in the data we used two techniques. The first one is under sampling and another one is over-sampling. We have trained and tested our models using both the imbalanced dataset and also with the under-sampled dataset. In the under-sampled dataset, we kept a total of 15012 rows of data where both the fraud and genuine class has 7506 rows. We did not use the oversampling dataset to train and test our model because of our limitation of computational power.

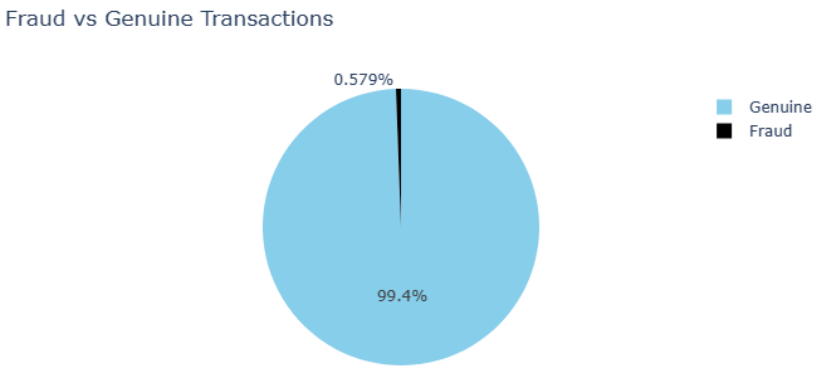

Figure 2.4: Imbalanced Dataset

3. Dataset Pre-Processing

3.1. Handling Null Values

In the dataset we are working with, there are 1281 null values in the 'amt' feature, meaning that there is no value recorded for these transactions. To handle these null values, we can use an imputation strategy, which involves replacing the missing values with other values. While training our data using under-sampled data it was very necessary to impute these null values. We could just drop the null-valued columns but for the undersampled data, it would not be very suitable as we have very few data points there.

In this case, we are using the 'SimpleImputer' class from the 'sklearn.impute' module in Python to impute the missing values. We are using the "median" strategy, which means that the missing values will be replaced with the median value of the 'amt' column. We used the median strategy because 52.8% values of the amt columns are closest to the median. The median is a statistical measure that represents the middle value in a dataset when it is ordered from smallest to largest. It is another common strategy for imputing missing values in numerical data. The advantage of using the median over the mean is that it is less sensitive to extreme values or outliers in the data, as it only depends on the order of values rather than their actual values. However, using the median may not be appropriate if the data is heavily skewed, as it may not accurately represent the central tendency of the data. In such cases, other imputation strategies such as the mode or regression-based imputation may be more appropriate

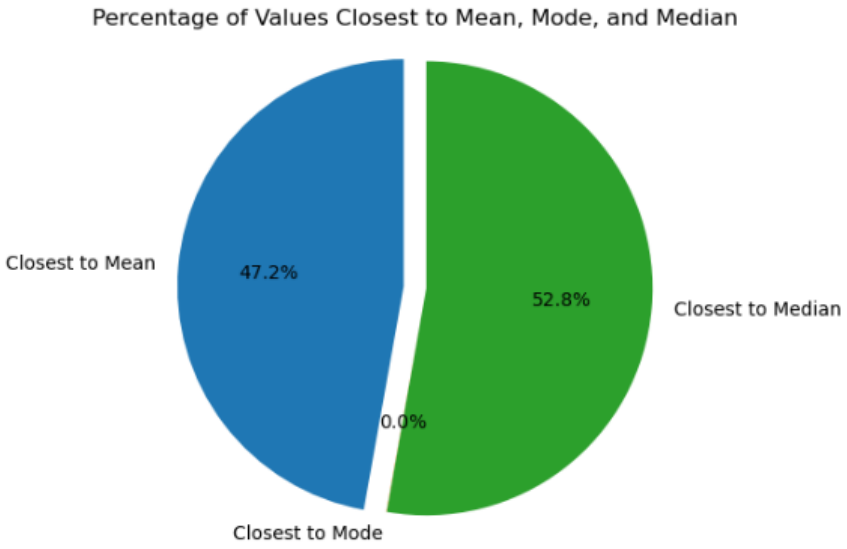

Figure 3.1: Percentage of Values Closest to Mean, Mode, and Median of amt column

In another column named ‘unix_time‘ we also encounter the null value problem. But this time the rows that are null for this column are 151. It is really small in terms of a total of 129,667 data points. So we can easily ignore these data points so we drop the rows which have a null value in the ‘unix_time‘ feature.

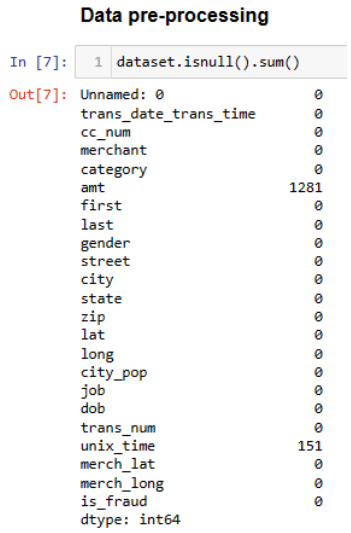

Figure 3.2: Before pre-processing

Figure 3.3: After pre-processing

3.2. Handling Categorical Values

Some of the characteristics in the dataset we’re dealing with are categorical variables, which means they only have a limited number of distinct values. Many machine learning methods, however, demand numerical input, therefore we must encode these categorical variables into numbers before using them in our models. For example for our training ”job” is a very important feature which is a categorical value. So we encode these types of necessary features using Label encoder.

Label encoding, which assigns a unique integer value to each category in the variable, is a typical approach to encoding categorical variables. For example, if we have a categorical variable called color with three possible values: ”red”, ”green”, and ”blue”, we may use label encoding to assign the numbers 0, 1, and 2 to these categories, respectively.

Label encoding is a quick and easy approach to convert category variables into numerical data that may be used in machine learning models. It is also relatively efficient because it does not necessitate the creation of additional columns or the execution of sophisticated data transformations.

It is important to note that label encoding presupposes an ordinal relationship between the categories, which means that the numerals assigned to the categories indicate some sort of order or ranking. If this assumption is not true, or if the categories do not have a natural order, then other encoding methods (such as one-hot encoding) may be more suited.

Handling with Catagorical values

In [10]:
```
object_cols = dataset.select_dtypes(include=['object']).columns
for col in object_cols:
    le = LabelEncoder()
    dataset[col] = le.fit_transform(dataset[col])
```

In [11]:
```
dataset.head()
```

Out[11]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	long	city_pop	job	dob	trans
0	0	0	2703186189652095	514	8	4.97	162	18	0	568	...	36.0788	-81.1781	3495	370	779	
1	1	1	630423337322	241	4	107.23	309	157	0	435	...	48.8878	-118.2105	149	428	607	
2	2	2	38859492057661	390	0	220.11	115	381	1	602	...	42.1808	-112.2620	4154	307	302	
3	3	3	3534093764340240	360	2	45.00	163	463	1	930	...	46.2306	-112.1138	1939	328	397	
4	4	4	375534208663984	297	9	41.96	336	149	1	418	...	38.4207	-79.4629	99	116	734	

5 rows × 23 columns

Figure 3.4: Handling Categorical Values

4. Feature Scaling

Feature scaling is a machine learning approach for normalizing or standardizing the range of independent variables or features in a dataset. The goal is for each characteristic to contribute equally to the model's predictions. One of the most prominent feature scaling strategies is the scikit-learn library's "StandardScaler" class. It scales the features so that they have a zero mean and a unit variance.

The StandardScaler is used to scale the training and testing data in the following code sample. To begin, the StandardScaler is fitted to the training data using the fit() method, which calculates the mean and standard deviation of each feature in the training set. The mean and standard deviation determined during the fit phase is then utilized to standardize the training and testing data using the transform() method.

It's worth noting that feature scaling is required for some machine learning algorithms, such as gradient descent-based techniques, which can converge quicker when the features are on a similar scale. It can also improve the accuracy of other algorithms by preventing some features from being regarded as more relevant merely because they have greater values than others.

Feature scaling

In [104]:
```
scaler= StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

In [105]:
```
dataset.head()
```

Out[105]:

	Unnamed: 0	trans_date_trans_time	cc_num	merchant	category	amt	first	last	gender	street	...	lat	long	city_pop	job	dob	trans
0	0	0	2703186189652095	514	8	4.97	162	18	0	568	...	36.0788	-81.1781	3495	370	779	
1	1	1	630423337322	241	4	107.23	309	157	0	435	...	48.8878	-118.2105	149	428	607	1
2	2	2	38859492057661	390	0	220.11	115	381	1	602	...	42.1808	-112.2620	4154	307	302	8
3	3	3	3534093764340240	360	2	45.00	163	463	1	930	...	46.2306	-112.1138	1939	328	397	5
4	4	4	375534208663984	297	9	41.96	336	149	1	418	...	38.4207	-79.4629	99	116	734	8

5 rows × 23 columns

Figure 4.1: Feature Scaling

5. Dataset Splitting

Splitting the dataset into training and testing sets is a popular strategy in machine learning since it allows you to evaluate the model's performance on unseen data. When dealing with imbalanced datasets, where the amount of samples in each class is not equal, stratified splitting is extremely crucial.

We are splitting the dataset into training and testing sets using the "train_test_split" function from the "sklearn.model_selection" module in Python.

The X variable includes all of the features in the dataset except the target variable (is_fraud), but the Y variable only includes the target variable. To delete the is_fraud column from X, we utilize the drop function.

The train_test_split function is used to randomly split the dataset into training and testing sets, with the test_size parameter specifying a 70:30 ratio (30% of the data is utilized for testing). The stratify parameter is set to y, indicating that the splitting is stratified. This ensures that the proportion of fraudulent transactions in the training and testing sets is roughly the same as it is in the original dataset. Because the random_state argument is set to a constant value (2), the same split will be created each time the code is performed. This contributes to reproducible and reliable results.

Spliting the dataset

In [20]:
```
X = train_df.drop("is_fraud",axis=1).values
y = train_df["is_fraud"].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,stratify=y,random_state=2)
```

In [21]:
```
print("X_train shape:", X_train.shape)
print("y_train shape:", y_train.shape)
```

```
X_train shape: (907566, 15)
y_train shape: (907566,)
```

In [22]:
```
print("X_test shape:", X_test.shape)
print("y_test shape:", y_test.shape)
```

```
X_test shape: (388958, 15)
y_test shape: (388958,)
```

Figure 5.1: Splitting and Scaling the dataset

6. Model Training & Testing

6.1. KNN

The K-Nearest Neighbors (KNN) classification technique assigns a label to a query point based on the majority class of its K nearest neighbors after locating the K closest data points to it in the training set.

Deciding how many K neighbors I want. Then measure the distance between every point in the training set and the query point. Then determine the K data points that are the closest to the query point based on the calculated distance. Based on the majority class of the query point's K closest neighbors, give it a class. The K neighbors are asked to vote on this. The algorithm may produce overfitting if the K value is too low, while underfitting if the K value is excessively high.

Here's K-Nearest Neighbors (KNN) for this dataset:

Model training: K-Nearest Neighbors (KNN) classifier is trained on the preprocessed dataset. importing data, checking data, splitting data, calculating distance, predicting, and calculating label works here.

Model evaluation: train the model based on the dataset and test the model on the same dataset by evaluating with comparing those predicted responses.

Comparison with other models: The performance of the K-Nearest Neighbors (KNN) is compared with other models like the Naive Bayes model and Logistic Regression to see which model performs better on this dataset.

KNN	Training Accuracy	Testing Accuracy	Precision	Recall
Imbalanced Dataset	0.99	0.99	0.6504	0.4323
Under Sampled-Dataset	0.9065	0.8126	0.8232	0.7961

Table 6.1: Scores of KNN Model

In this specific dataset, KNN doesn't perform relatively well compared to naive bayes in terms of accuracy, precision, and recall. But in comparison with it Logistic Regression, it is a little bit behind in terms of precision.

6.2. Logistic Regression

Logistic regression is a statistical method used for binary classification problems, where the goal is to predict a binary outcome (e.g.1/0, yes/no, true/false) based on a set of input features. It models the relationship between the input features and the probability of the binary outcome, using a logistic function. In our dataset, the Logistic regression model is used to detect if a transaction is fraudulent or genuine by the class label. To use logistic regression, we first collect data and identify the input features and the binary outcome we want to predict. Then, we fit a logistic regression model to the data using an optimization algorithm that finds the best set of weights for each input feature that maximizes the likelihood of the observed outcomes. Once the model is trained, we can use it to make predictions on new data by feeding in the input features and obtaining the predicted probability of the binary outcome. Finally, we apply a threshold to the predicted probabilities to obtain the binary prediction. When dealing with the original imbalanced dataset where We are solving a classification problem and among the total of 129,667 rows of data (datapoints) in our dataset only 7506 rows are marked as fraud (class 1) and 1289189 rows are marked as genuine (class 0), the logistic regression model demonstrated Training and Testing Accuracy Accuracy very high and Precision and Recall to be 0. But after applying Under Sampled-Dataset on our dataset Precision and Recall improved.

In this dataset, we had to use Dataset Pre-Processing for handling Null Values Categorical Values, and feature scaling. After that, we did the dataset splitting into training and testing sets for model training and testing.

Model training: Logistic regression is trained on the preprocessed dataset. Logistic regression is used in this dataset because it is a binary classification problem. The logistic function, also known as the sigmoid function, takes any input value and maps it to a value between 0 and 1, which represents the predicted probability of a positive outcome.

The logistic function is defined as:

$f(x) = \frac{1}{1+e^{-z}}$

where x is the input feature vector, and z is the dot product of the input feature vector and a weight vector, plus a bias term:

$z = w^T x + b$

Here, w is the weight vector and b is the bias term.

To predict the probability of the positive outcome for a new input feature vector, we plug it into the logistic function, which outputs a value between 0 and 1. If the predicted probability is greater than or equal to a threshold, typically 0.5, the model predicts a positive outcome; otherwise, it predicts a negative outcome.

Model evaluation: To assess the effectiveness of the Logistic Regression model in distinguishing between fraudulent and genuine transactions, it can be evaluated on the test dataset using performance metrics like accuracy, precision, and recall.

Comparison with other models: The performance of the Logistic Regression model is compared with other models like K-Nearest Neighbors (KNN) Naive Bayes to see which model performs better on this dataset. After using logistic regression on the balanced dataset after sampling it shows that the logistic regression is better than the other 2 models in terms of accuracy precision and recall.

Logistic Regression	Training Accuracy	Testing Accuracy	Precision	Recall
Imbalanced Dataset	0.99	0.99	0	0
Under Sampled-Dataset	0.8549	0.8565	0.9431	0.7587

Table 6.2: Scores of Logistic Regression Model

6.3. Naive Bayes

In this dataset, Naive Bayes is used as a classification algorithm to predict whether a transaction is fraudulent or genuine. Naive Bayes is a probabilistic machine learning algorithm based on Bayes' Theorem, which works on the assumption that the features are conditionally independent given the class label. This assumption is called "naive" because it might not always be true, but it still performs well in many cases, especially when dealing with categorical data or text data.

The Gaussian Naive Bayes model is used because it assumes that the likelihood of the features follows a Gaussian distribution. It works by calculating the mean and variance of each feature for each class and then uses these parameters to estimate the probabilities of a new data point belonging to each class. The class with the highest probability is chosen as the final prediction.

When dealing with the original imbalanced dataset, Naive Bayes demonstrated relatively low precision and recall for detecting fraudulent transactions. However, after applying undersampling to balance the dataset, the model's performance improved significantly. The comparison of model performance before and after balancing the dataset shows the impor-

tance of addressing class imbalance when working with fraud detection problems.

Here's how Naive Bayes works for this dataset:

Preprocessing: The dataset is preprocessed to handle missing values, categorical variables, and to scale the features. This ensures that the algorithm can process the data efficiently.

Model training: The Gaussian Naive Bayes classifier is trained on the preprocessed dataset. Gaussian Naive Bayes is used because it assumes that the continuous features follow a Gaussian (normal) distribution, which is a reasonable assumption for many real-world datasets.

Model evaluation: The performance of the Naive Bayes model is evaluated on the test dataset using metrics such as accuracy, precision, recall, etc. This gives an idea of how well the model is able to classify fraudulent and genuine transactions.

Naive Bayes	**Training Accuracy**	**Testing Accuracy**	**Precision**	**Recall**
Imbalanced Dataset	0.99	0.99	0.2157	0.4822
Under Sampled-Dataset	0.76	0.76	0.9514	0.9514

Table 6.3: Scores of Naive Bayes Model

Comparison with other models: The performance of the Naive Bayes model is compared with other models like K-Nearest Neighbors (KNN) and Logistic Regression to see which model performs better on this dataset.

Handling class imbalance dataset: The dataset is undersampled to create a balanced dataset with an equal number of fraudulent and genuine transactions. This helps address the class imbalance issue, which can cause biased predictions toward the majority class.

Model training and evaluation on balanced data: The Naive Bayes model is trained and evaluated again on the balanced dataset to see if the performance improves when class imbalance is addressed In this specific dataset, Naive Bayes performed relatively poor compared to KNN and Logistic Regression in terms of accuracy, precision, and recall. However, it's important to note that the performance of the algorithm may vary depending on the dataset and the specific problem being solved.

7. Model Selection/Comparison Analysis

7.1. Confusion Matrix

As we trained our models in two different approaches we are trying to show and compare the confusion matrices for each model, each method. In this subsection, the first figure is the confusion matrix of the model trained on the imbalanced dataset and the second figure is the confusion matrix of the model trained on the under-sampled dataset.

KNN

The two confusion matrices show the performance of a model in classifying transactions as real or fraudulent. In the first scenario(Figure 7.1), out of a total of 386,751 transactions, 386,184 were classified as real (true negatives) and 1278 as fraudulent (false negatives). In the second scenario(Figure 7.2), out of a total of 4,504 transactions, 1867 were classified as real (true negatives) and 1792 as fraudulent (true positives). Comparing the two matrices, we can see that the model performed better in the second scenario in terms of detecting fraudulent transactions. The number of true positives increased from 973 to 1792, while the number of false negatives decreased from 1333 to 459. This indicates a higher true positive rate (TPR) and a lower false negative rate (FNR) in the second scenario. Overall, the second model seems to have a better performance in detecting fraudulent transactions.

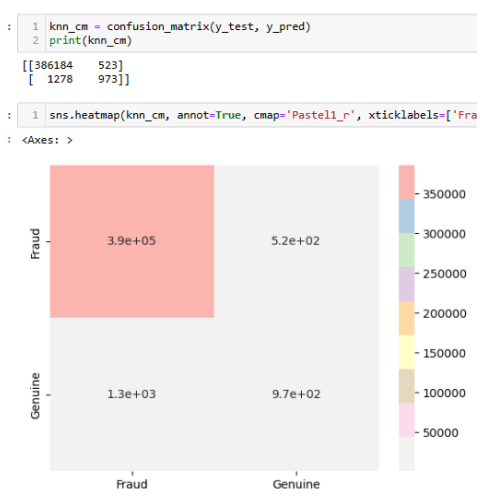

Figure 7.1: Confusion Matrix of KNN on Imbalanced Data

Figure 7.2: Confusion Matrix of KNN on Under Sampled Data

Logistic Regression

The first confusion matrix (Figure 7.3) shows a total of 386,771 transactions, out of which 386,503 were categorized as genuine (true negatives), 2251 as fraudulent (false negatives), and none were categorized as genuine (true positives). In contrast, the second confusion matrix (Figure 7.4) has a total of 3,500 transactions, out of which 2149 were categorized as genuine (true negatives), 1708 as fraudulent (true positives), 109 as genuine (false positives), and 542 as fraudulent (false negatives).

Comparing the two matrices, we can observe that the second one performs better in identifying both genuine and fraudulent transactions. Specifically, the second matrix has a higher number of true positives and true negatives, indicating better overall performance. Additionally, the second matrix has a higher recall for fraudulent transactions, meaning more of them are correctly identified, and higher specificity for genuine transactions, meaning fewer of them are incorrectly classified as fraudulent. Overall, the second model appears to be more accurate and reliable in detecting fraudulent transactions than the first one.

Figure 7.3: Confusion Matrix of Logistic Regression on Imbalanced Data

Figure 7.4: Confusion Matrix of Logistic Regression on Under Sampled Data

Naive Bayes

In the first confusion matrix (Figure 7.5), there are 382,932 true negatives (TN), 1,167 false positives (FP), 3,942 false negatives (FN), and 1,084 true positives (TP).

In the second confusion matrix (Figure 7.6), there are 2,190 true negatives (TN), 631 false positives (FP), 54 false negatives (FN), and 1,621 true positives (TP).

Comparing the two matrices, we can see that the second one has higher values for TP and TN, indicating better performance in identifying both genuine and fraudulent transactions. Specifically, the second matrix has a higher recall for genuine transactions, meaning more of them are correctly identified, and higher specificity for fraudulent transactions, meaning fewer of them are incorrectly classified as genuine.

In other words, the second model has a better overall performance in correctly classifying both genuine and fraudulent transactions compared to the first model. The higher values of TP and TN in the second model indicate that it has a higher accuracy in identifying both types of transactions. Additionally, the higher recall for genuine transactions in the second model indicates that it is better at correctly identifying genuine transactions, and the higher specificity for fraudulent transactions indicates that it is better at correctly identifying fraudulent transactions.

Figure 7.5: Confusion Matrix of Naive Bayes on Imbalanced Data

Figure 7.6: Confusion Matrix of Naive Bayes on Under Sampled Data

7.2. Accuracy Comparison

If we compare our models based on the accuracy we can see all the models outperformed when we tested them on our imbalanced dataset. Around 99% testing accuracy we got for each model compared with the under-sampled dataset which is really very good. The under-sampled dataset also has a handsome accuracy rate where k-NN has 0.8126, logistic regression has 0.8565, and naive bayes has 0.76. So in terms of accuracy under-sampled dataset is not good as an imbalanced dataset. But in the confusion matrix subsection, we saw how an imbalanced dataset is bad in terms of precision. We will look into it again in the precision,

recall comparison subsection. So only using the accuracy scores we cannot determine which model is best.

Model	Imbalance Dataset	Under-Sampled Dataset
k-NN	0.9953	0.8126
Logistic Regression	0.9937	0.8565
Naive Bayes	0.99	0.76

Table 7.1: Testing accuracy scores for each model on Imbalance Dataset and Under-Sampled Dataset

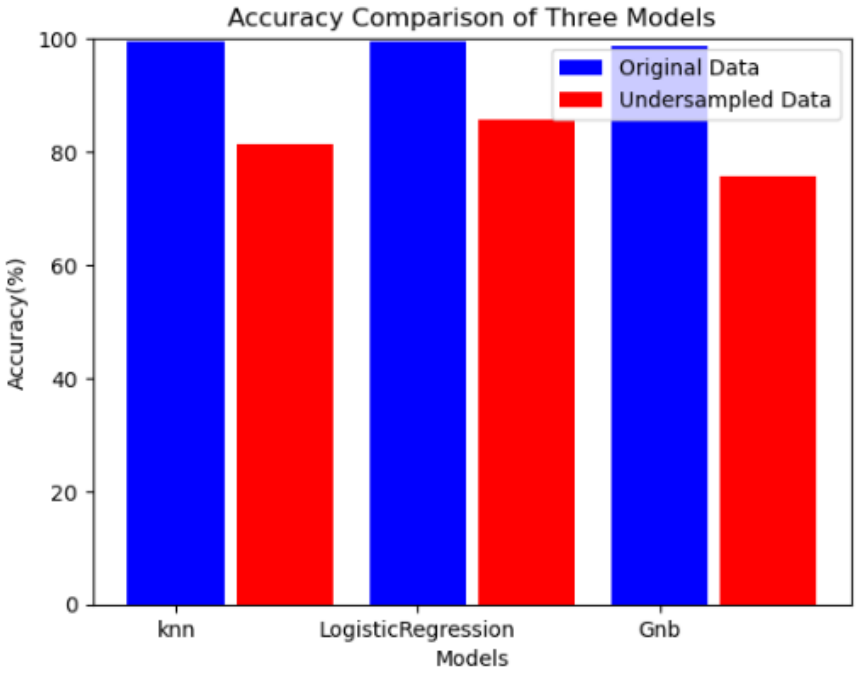

Figure 7.7: Testing accuracy scores for each model on Imbalance Dataset and Under-Sampled Dataset

7.3. Precision Comparison

Precision is a measure of the accuracy of a classification model. It is defined as the ratio of true positive predictions to the total number of positive predictions made by the model. In other words, it measures the proportion of correctly classified positive instances out of all the instances classified as positive by the model. Precision is often used in conjunction with recall (also known as sensitivity) to evaluate the overall performance of a classification model.

The imbalance dataset has very poor precision compared with the under-sampled dataset.

Among the under-sampled dataset, Naive Bayes has the highest precision. So in terms of precision Naive Bayes(on the under-sampled dataset) is a very good model.

Model	Imbalance Dataset	Under-Sampled Dataset
k-NN	0.6504	0.8232
Logistic Regression	0.0	0.9431
Naive Bayes	0.2157	0.9514

Table 7.2: Precision scores for each model on Imbalance Dataset and Under-Sampled Dataset

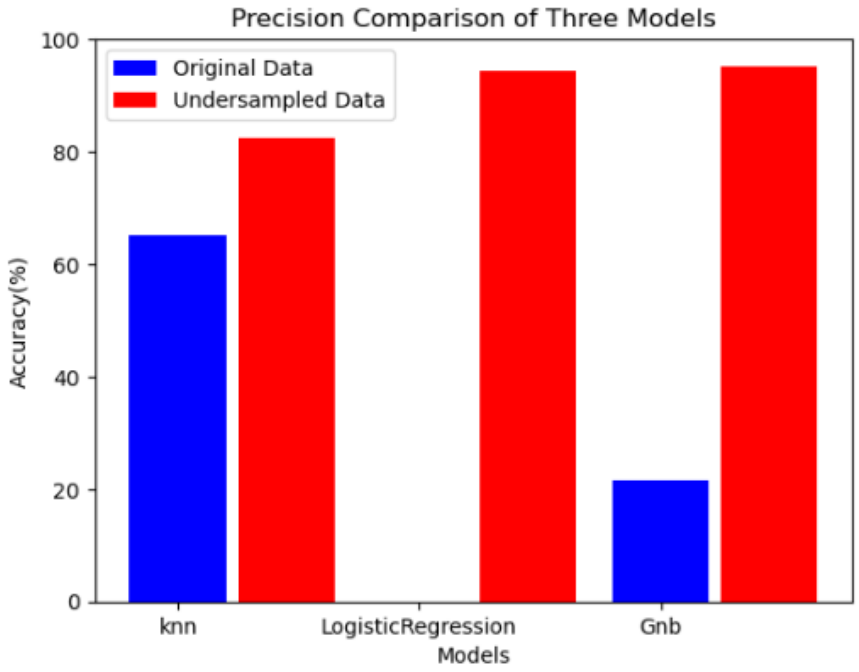

Figure 7.8: Precision scores for each model on Imbalance Dataset and Under-Sampled Dataset

7.4. Recall Comparison

Recall is a performance metric used to evaluate the ability of a classification model to correctly identify all positive instances (i.e., true positives) out of all actual positive instances (i.e., true positives and false negatives) in a dataset. In other words, recall measures the proportion of actual positives that are correctly identified by the model. It is also known as sensitivity or true positive rate. The formula for recall is:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{7.1}$$

where TP is the number of true positives and FN is the number of false negatives.

The imbalance dataset has very poor recall compared with the under-sampled dataset. Among the under-sampled dataset, Naive Bayes has the highest recall. So in terms of recall score Naive Bayes(on the under-sampled dataset) is a very good model.

Model	**Imbalance Dataset**	**Under-Sampled Dataset**
k-NN	0.4323	0.7961
Logistic Regression	0.0	0.7587
Naive Bayes	0.4816	0.9514

Table 7.3: Recall scores for each model on Imbalance Dataset and Under-Sampled Dataset

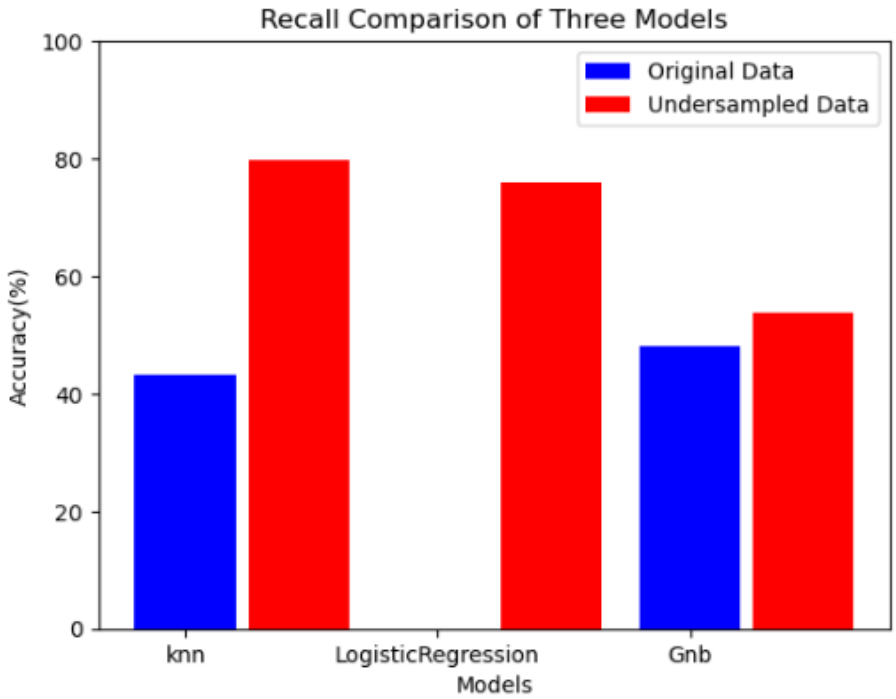

Figure 7.9: Recall scores for each model on Imbalance Dataset and Under-Sampled Dataset

8. Conclusion

We may conclude that the Logistic Regression model performs the best on the under-sampled dataset based on the testing accuracy, precision, and recall scores of three classification models - k-NN, Logistic Regression, and Naive Bayes - on two different datasets (imbalance and under-sampled).

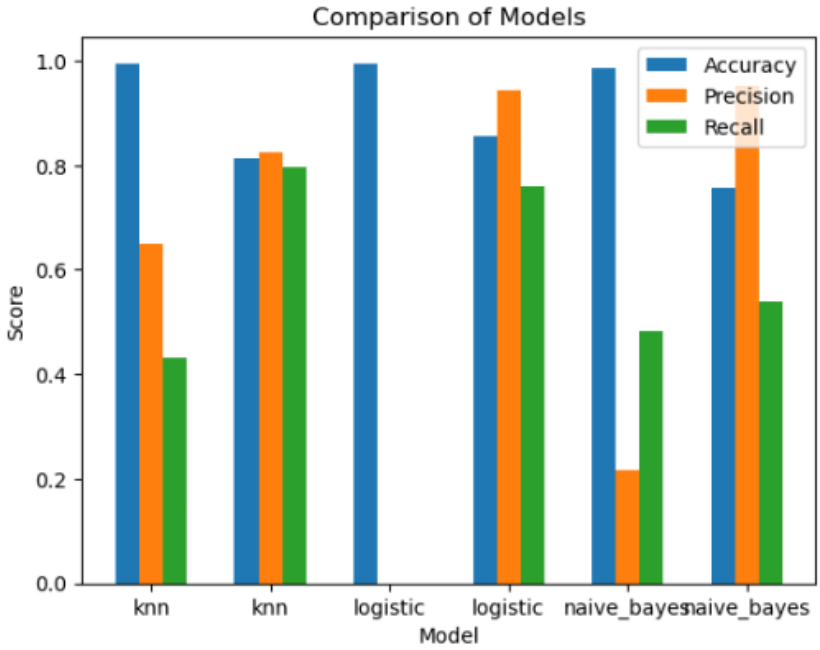

Figure 8.1: Accuracy, Precision, Recall scores for each model on Imbalance Dataset and Under-Sampled Dataset

However, it should be noted that the under-sampled dataset is not ideal because it requires the removal of a considerable quantity of data, which might result in the loss of vital information. This demonstrates the issue of imbalanced datasets, in which the class distribution is highly skewed toward one class, making it difficult for classification algorithms to correctly identify the minority class.

This project could be developed in the future to include other strategies for dealing with imbalanced datasets, such as oversampling, data augmentation, and the use of ensemble models. Furthermore, extra characteristics could be added to the dataset to improve the classification models' performance. Finally, the models might be tested on a bigger, more diverse dataset to see how robust they are and how well they generalize.