**Improving Classification Accuracy with Imbalanced Data in Python**

Let’s get started with the task of performing classification on imbalanced data by importing the necessary Python libraries and the **dataset.**

**import pandas as pd**

**# load the dataset**

**data = pd.read\_csv("Insurance claims data.csv")**

**print(data.head())**

**Output:**

**policy\_id subscription\_length vehicle\_age customer\_age region\_code \  
0 POL045360 9.3 1.2 41 C8   
1 POL016745 8.2 1.8 35 C2   
2 POL007194 9.5 0.2 44 C8   
3 POL018146 5.2 0.4 44 C10   
4 POL049011 10.1 1.0 56 C13   
  
 region\_density segment model fuel\_type max\_torque ... is\_brake\_assist \  
0 8794 C2 M4 Diesel 250Nm@2750rpm ... Yes   
1 27003 C1 M9 Diesel 200Nm@1750rpm ... No   
2 8794 C2 M4 Diesel 250Nm@2750rpm ... Yes   
3 73430 A M1 CNG 60Nm@3500rpm ... No   
4 5410 B2 M5 Diesel 200Nm@3000rpm ... No   
  
 is\_power\_door\_locks is\_central\_locking is\_power\_steering \  
0 Yes Yes Yes   
1 Yes Yes Yes   
2 Yes Yes Yes   
3 No No Yes   
4 Yes Yes Yes   
  
 is\_driver\_seat\_height\_adjustable is\_day\_night\_rear\_view\_mirror is\_ecw \  
0 Yes No Yes   
1 Yes Yes Yes   
2 Yes No Yes   
3 No No No   
4 No No Yes   
  
 is\_speed\_alert ncap\_rating claim\_status   
0 Yes 3 0   
1 Yes 4 0   
2 Yes 3 0   
3 Yes 0 0   
4 Yes 5 0   
  
[5 rows x 41 columns]**

Let’s have a quick look at the column information and whether the data contains any null values or not:

data.info()

**Output:**

**<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 58592 entries, 0 to 58591  
Data columns (total 41 columns):  
 # Column Non-Null Count Dtype   
--- ------ -------------- -----   
 0 policy\_id 58592 non-null object   
 1 subscription\_length 58592 non-null float64  
 2 vehicle\_age 58592 non-null float64  
 3 customer\_age 58592 non-null int64   
 4 region\_code 58592 non-null object   
 5 region\_density 58592 non-null int64   
 6 segment 58592 non-null object   
 7 model 58592 non-null object   
 8 fuel\_type 58592 non-null object   
 9 max\_torque 58592 non-null object   
 10 max\_power 58592 non-null object   
 11 engine\_type 58592 non-null object   
 12 airbags 58592 non-null int64   
 13 is\_esc 58592 non-null object   
 14 is\_adjustable\_steering 58592 non-null object   
 15 is\_tpms 58592 non-null object   
 16 is\_parking\_sensors 58592 non-null object   
 17 is\_parking\_camera 58592 non-null object   
 18 rear\_brakes\_type 58592 non-null object   
 19 displacement 58592 non-null int64   
 20 cylinder 58592 non-null int64   
 21 transmission\_type 58592 non-null object   
 22 steering\_type 58592 non-null object   
 23 turning\_radius 58592 non-null float64  
 24 length 58592 non-null int64   
 25 width 58592 non-null int64   
 26 gross\_weight 58592 non-null int64   
 27 is\_front\_fog\_lights 58592 non-null object   
 28 is\_rear\_window\_wiper 58592 non-null object   
 29 is\_rear\_window\_washer 58592 non-null object   
 30 is\_rear\_window\_defogger 58592 non-null object   
 31 is\_brake\_assist 58592 non-null object   
 32 is\_power\_door\_locks 58592 non-null object   
 33 is\_central\_locking 58592 non-null object   
 34 is\_power\_steering 58592 non-null object   
 35 is\_driver\_seat\_height\_adjustable 58592 non-null object   
 36 is\_day\_night\_rear\_view\_mirror 58592 non-null object   
 37 is\_ecw 58592 non-null object   
 38 is\_speed\_alert 58592 non-null object   
 39 ncap\_rating 58592 non-null int64   
 40 claim\_status 58592 non-null int64   
dtypes: float64(3), int64(10), object(28)  
memory usage: 18.3+ MB**

**data.isnull().sum()**

**Output :**

**policy\_id 0  
subscription\_length 0  
vehicle\_age 0  
customer\_age 0  
region\_code 0  
region\_density 0  
segment 0  
model 0  
fuel\_type 0  
max\_torque 0  
max\_power 0  
engine\_type 0  
airbags 0  
is\_esc 0  
is\_adjustable\_steering 0  
is\_tpms 0  
is\_parking\_sensors 0  
is\_parking\_camera 0  
rear\_brakes\_type 0  
displacement 0  
cylinder 0  
transmission\_type 0  
steering\_type 0  
turning\_radius 0  
length 0  
width 0  
gross\_weight 0  
is\_front\_fog\_lights 0  
is\_rear\_window\_wiper 0  
is\_rear\_window\_washer 0  
is\_rear\_window\_defogger 0  
is\_brake\_assist 0  
is\_power\_door\_locks 0  
is\_central\_locking 0  
is\_power\_steering 0  
is\_driver\_seat\_height\_adjustable 0  
is\_day\_night\_rear\_view\_mirror 0  
is\_ecw 0  
is\_speed\_alert 0  
ncap\_rating 0  
claim\_status 0  
dtype: int64**

The dataset contains 58,592 entries and 41 columns, including the target variable claim\_status. It is based on the problem of insurance claim frequency prediction. Here’s a brief overview of some of the features:

* policy\_id: Unique identifier for the insurance policy
* subscription\_length, vehicle\_age, customer\_age: Numeric attributes related to the policy, vehicle, and customer
* region\_code, segment, model, fuel\_type: Categorical attributes representing the region, vehicle segment, model, and fuel type
* max\_torque, max\_power, engine\_type: Specifications of the vehicle’s engine
* airbags, is\_esc, is\_adjustable\_steering: Features related to the vehicle’s safety and convenience
* claim\_status: Target variable indicating whether a claim was made (1) or not (0)

Next, I will perform exploratory data analysis to visualize and understand the distributions, relationships, and patterns in the data. It will include examining the distribution of the target variable and key features. Let’s start with visualizing the distribution of the claim\_status to understand the class balance:

**import matplotlib.pyplot as plt**

**import seaborn as sns**

**sns.set\_style("whitegrid")**

**# plot the distribution of the target variable 'claim\_status'**

**plt.figure(figsize=(8, 5))**

**sns.countplot(x='claim\_status', data=data)**

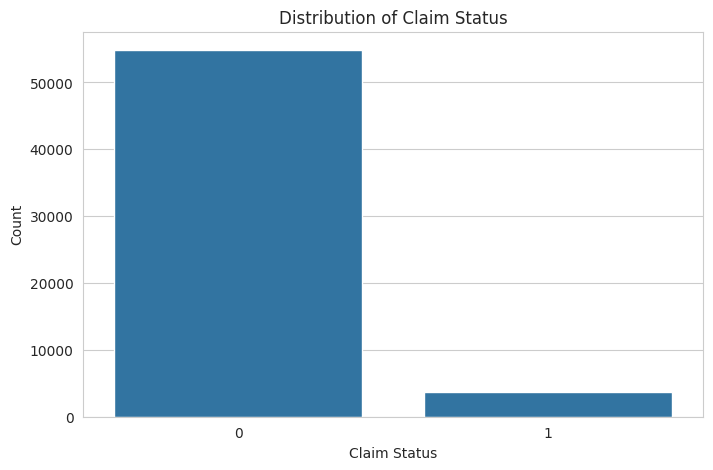
**plt.title('Distribution of Claim Status')**

**plt.xlabel('Claim Status')**

**plt.ylabel('Count')**

**plt.show()**

**Output:**



The distribution of the claim\_status shows a significant imbalance between the classes, with much fewer claims (1) compared to no claims (0). This imbalance will be a challenge to address during the model training phase to ensure our model does not become biased toward predicting the majority class.

Next, I will perform an analysis of both numerical and categorical features to understand their distributions and relationships with the claim\_status. Let’s start by examining the distributions of some key numerical features such as subscription\_length, vehicle\_age, and customer\_age:



1

# selecting numerical columns for analysis

2

numerical\_columns = ['subscription\_length', 'vehicle\_age', 'customer\_age']

3

​

4

# plotting distributions of numerical features

5

plt.figure(figsize=(15, 5))

6

for i, column in enumerate(numerical\_columns, 1):

7

plt.subplot(1, 3, i)

8

sns.histplot(data[column], bins=30, kde=True)

9

plt.title(f'Distribution of {column}')

10

​

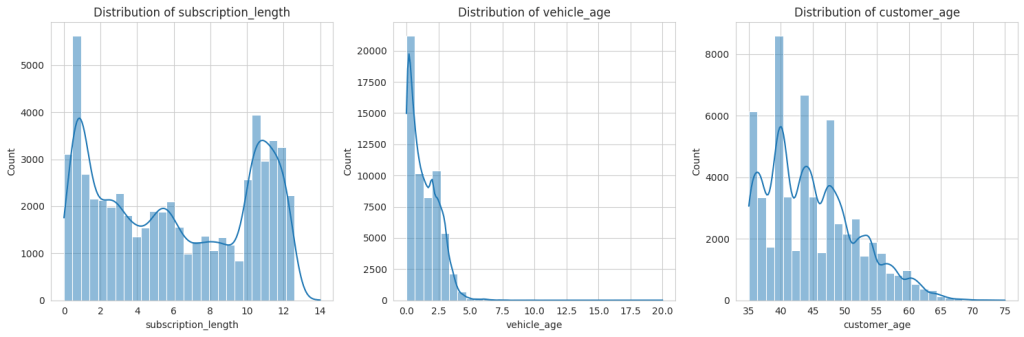
11

plt.tight\_layout()

12

plt.show()

**Output:**



The distributions of the numerical features subscription\_length, vehicle\_age, and customer\_age show the following characteristics:

* subscription\_length: Most values are clustered around lower numbers, indicating that many policies have shorter subscription lengths.
* vehicle\_age: This distribution is somewhat uniform but with spikes at specific ages, possibly representing common vehicle age intervals in the dataset.
* customer\_age: This shows a fairly normal distribution, with the majority of customers falling within a middle-age range.

Next, we will analyze relevant categorical features to understand their variation and relationship with the claim\_status. I’ll focus on features like region\_code, segment, and fuel\_type:



1

# selecting some relevant categorical columns for analysis

2

categorical\_columns = ['region\_code', 'segment', 'fuel\_type']

3

​

4

# plotting distributions of categorical features

5

plt.figure(figsize=(15, 10))

6

for i, column in enumerate(categorical\_columns, 1):

7

plt.subplot(3, 1, i)

8

sns.countplot(y=column, data=data, order = data[column].value\_counts().index)

9

plt.title(f'Distribution of {column}')

10

plt.xlabel('Count')

11

plt.ylabel(column)

12

​

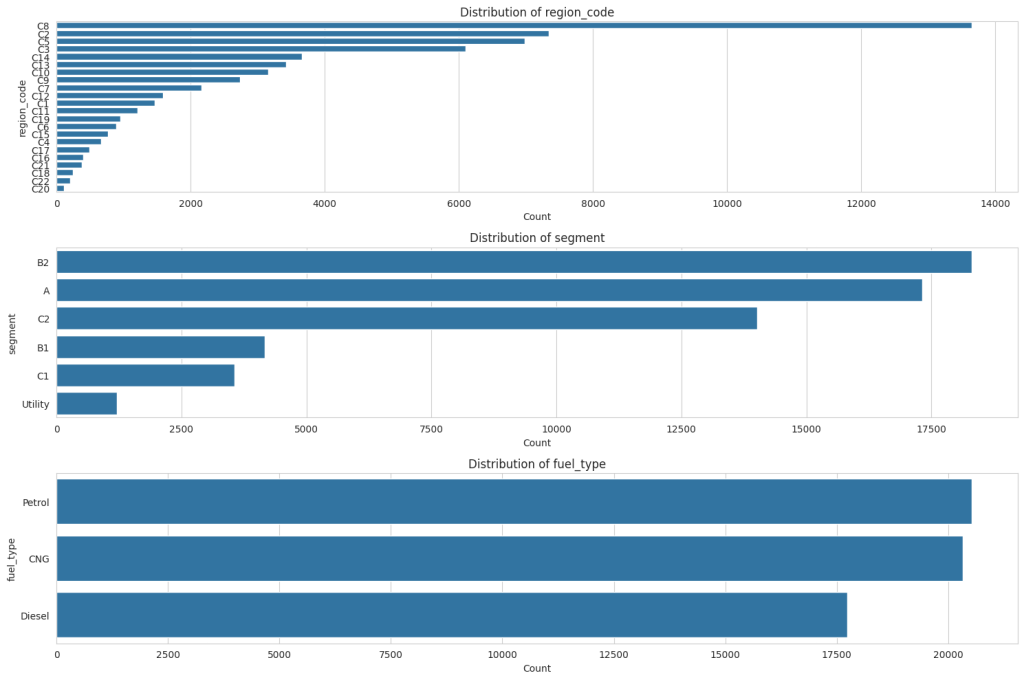
13

plt.tight\_layout()

14

plt.show()

**Output:**



For ‘region\_code,’ there is a wide variety of codes, each with varying counts, but a few specific codes dominate with much higher counts than others. In the ‘segment’ distribution, there are fewer categories, with the ‘B2’ segment being the most common, followed by ‘A’ and ‘C2,’ and the ‘Utility’ segment being the least common. Lastly, ‘fuel\_type’ shows three categories: ‘Petrol’ has the highest count than CNG and Diesel.

**Handling Class Imbalance**

The next step is to balance the dataset using oversampling to handle the class imbalance observed in the claim\_status. Let’s proceed with balancing the classes:

**from sklearn.utils import resample**

**# separate majority and minority classes**

**majority = data[data.claim\_status == 0]**

**minority = data[data.claim\_status == 1]**

**# oversample the minority class**

**minority\_oversampled = resample(minority,**

**replace=True,**

**n\_samples=len(majority),**

**random\_state=42)**

**# combine majority class with oversampled minority class**

**oversampled\_data = pd.concat([majority, minority\_oversampled])**

**# check the distribution of undersampled and oversampled datasets**

**oversampled\_distribution = oversampled\_data.claim\_status.value\_counts()**

**oversampled\_distribution**

**Output:**

**0 54844  
1 54844  
Name: claim\_status, dtype: int64**

After performing oversampling on the minority class, both classes are balanced with 54,844 entries each. Now, let’s have a look at some key variables to see what the balanced data looks like:

**# plotting the distribution of 'customer\_age', 'vehicle\_age', and 'subscription\_length' with respect to 'claim\_status'**

**plt.figure(figsize=(15, 5))**

**# 'customer\_age' distribution**

**plt.subplot(1, 3, 1)**

**sns.histplot(data=oversampled\_data, x='customer\_age', hue='claim\_status', element='step', bins=30)**

**plt.title('Customer Age Distribution')**

**# 'vehicle\_age' distribution**

**plt.subplot(1, 3, 2)**

**sns.histplot(data=oversampled\_data, x='vehicle\_age', hue='claim\_status', element='step', bins=30)**

**plt.title('Vehicle Age Distribution')**

**# 'subscription\_length' distribution**

**plt.subplot(1, 3, 3)**

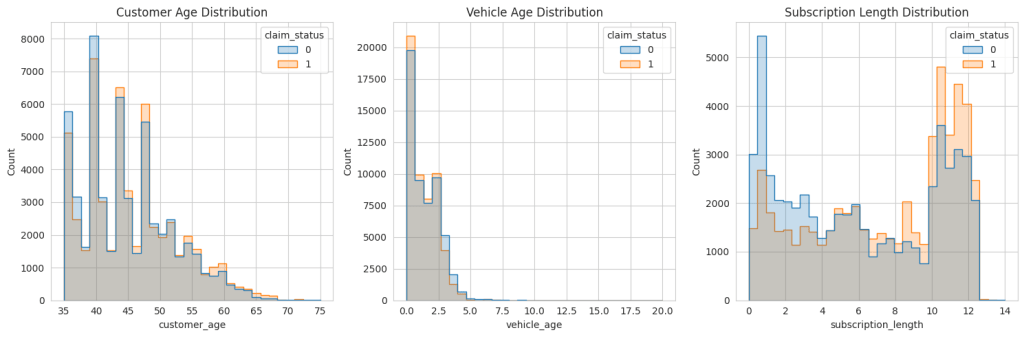
**sns.histplot(data=oversampled\_data, x='subscription\_length', hue='claim\_status', element='step', bins=30)**

**plt.title('Subscription Length Distribution')**

**plt.tight\_layout()**

**plt.show()**

**Output:**



The oversampled data does look like the original data. So, let’s move forward.

#### Feature Selection

Now, we will identify the most important variables for predicting insurance frequency claims. It involves analyzing both categorical and numerical features to determine their impact on the target variable. We will use feature importance techniques suitable for both types of variables.

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

# encode categorical variables

le = LabelEncoder()

encoded\_data = data.apply(lambda col: le.fit\_transform(col) if col.dtype == 'object' else col)

# separate features and target variable

X = encoded\_data.drop('claim\_status', axis=1)

y = encoded\_data['claim\_status']

# create a random forest classifier model

rf\_model = RandomForestClassifier(random\_state=42)

rf\_model.fit(X, y)

# get feature importance

feature\_importance = rf\_model.feature\_importances\_

# create a dataframe for visualization of feature importance

features\_df = pd.DataFrame({'Feature': X.columns, 'Importance': feature\_importance})

features\_df = features\_df.sort\_values(by='Importance', ascending=False)

print(features\_df.head(10)) # displaying the top 10 important features

Output:

**Feature Importance  
0 policy\_id 0.321072  
1 subscription\_length 0.248309  
3 customer\_age 0.176639  
2 vehicle\_age 0.135190  
5 region\_density 0.053838  
4 region\_code 0.052649  
7 model 0.000957  
24 length 0.000846  
26 gross\_weight 0.000834  
11 engine\_type 0.000791**

The top 10 most important variables for predicting insurance frequency claims, according to the Random Forest model, are:

1. policy\_id: Unique identifier for the insurance policy
2. subscription\_length: Length of the insurance subscription
3. customer\_age: Age of the customer
4. vehicle\_age: Age of the vehicle
5. region\_density: Population density of the region
6. region\_code: Code representing the region
7. model: Model of the vehicle
8. engine\_type: Type of engine in the vehicle
9. gross\_weight: Gross weight of the vehicle
10. length: Length of the vehicle

These variables appear to have the most influence on the likelihood of an insurance claim being made. However, it’s notable that policy\_id has a very high importance, which might not be intuitively relevant for prediction. So, we need to make sure to drop the policy\_id column while model training.

#### Model Training

The next step is to build a predictive model using the oversampled data. Given the nature of the task (binary classification), a suitable algorithm could be logistic regression, random forest, or gradient boosting. Considering the effectiveness of random forests in handling both numerical and categorical data and their ability to model complex interactions, we’ll proceed with a Random Forest classifier:

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, accuracy\_score

from sklearn.ensemble import RandomForestClassifier

# drop 'Policy\_id' column from the data

oversampled\_data = oversampled\_data.drop('policy\_id', axis=1)

# prepare the oversampled data

X\_oversampled = oversampled\_data.drop('claim\_status', axis=1)

y\_oversampled = oversampled\_data['claim\_status']

# encoding categorical columns

X\_oversampled\_encoded = X\_oversampled.apply(lambda col: LabelEncoder().fit\_transform(col) if col.dtype == 'object' else col)

# splitting the dataset into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X\_oversampled\_encoded, y\_oversampled, test\_size=0.3, random\_state=42)

# create and train the Random Forest model

rf\_model\_oversampled = RandomForestClassifier(random\_state=42)

rf\_model\_oversampled.fit(X\_train, y\_train)

# predictions

y\_pred = rf\_model\_oversampled.predict(X\_test)

print(classification\_report(y\_test, y\_pred))

Output:

The classification report above provides various metrics to evaluate the performance of the predictive model on the test data. Here’s an interpretation of the results:

1. For class 0 (no claim), precision is 1.00, meaning that when the model predicts no claim, it is correct 100% of the time. For class 1 (claim), precision is 0.98, indicating that when the model predicts a claim, it is correct 98% of the time.
2. For class 0, recall is 0.98, signifying that the model correctly identifies 98% of all actual no-claim instances. For class 1, recall is 1.00, showing that the model correctly identifies 100% of all actual claim instances.
3. The F1-score for both classes is 0.99, indicating a high balance between precision and recall. It means the model is both accurate and reliable in its predictions across both classes.
4. The overall accuracy of the model is 99%, which means that it correctly predicts the claim status for 99% of the cases in the test dataset.
5. The macro average for precision, recall and F1-score is 0.99, reflecting the average performance of the model across both classes without considering the imbalance in class distribution. This high value suggests that the model performs well across both classes. The weighted average for precision, recall, and F1-score is also 0.99, taking into account the imbalance in class distribution. It indicates that, on average, the model performs consistently well across the different classes when considering their distribution in the dataset.

These results indicate a highly effective model for predicting insurance claims, with strong performance metrics across both classes of outcomes. The high recall for claims (class 1) is particularly notable as it implies that the model is very effective at identifying the instances where claims occur, which is often the primary concern in imbalanced datasets.

Now, let’s label the original imbalanced data using our model to see how many instances are correctly classified from our model:

original\_encoded = data.drop('policy\_id', axis=1).copy()

encoders = {col: LabelEncoder().fit(X\_oversampled[col]) for col in X\_oversampled.select\_dtypes(include=['object']).columns}

for col in original\_encoded.select\_dtypes(include=['object']).columns:

if col in encoders:

original\_encoded[col] = encoders[col].transform(original\_encoded[col])

original\_encoded\_predictions = rf\_model\_oversampled.predict(original\_encoded.drop('claim\_status', axis=1))

comparison\_df = pd.DataFrame({

'Actual': original\_encoded['claim\_status'],

'Predicted': original\_encoded\_predictions

})

print(comparison\_df.head(10))

Output:

**Actual Predicted  
0 0 0  
1 0 0  
2 0 0  
3 0 0  
4 0 0  
5 0 0  
6 0 0  
7 0 0  
8 0 0  
9 0 0**

Let’s visualize the percentage of correctly classified and misclassified samples:



1

correctly\_classified = (comparison\_df['Actual'] == comparison\_df['Predicted']).sum()

2

incorrectly\_classified = (comparison\_df['Actual'] != comparison\_df['Predicted']).sum()

3

​

4

classification\_counts = [correctly\_classified, incorrectly\_classified]

5

labels = ['Correctly Classified', 'Misclassified']

6

​

7

# create a pie chart

8

plt.figure(figsize=(8, 8))

9

plt.pie(classification\_counts, labels=labels, autopct='%1.1f%%', startangle=140, colors=['#4CAF50', '#FF5733'])

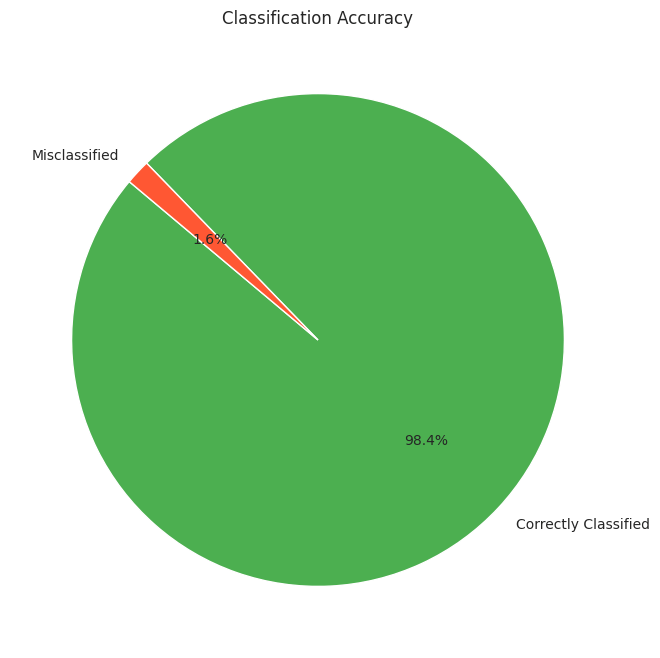
10

plt.title('Classification Accuracy')

11

plt.show()

Output:



So, we can see that our model performs well on the original imbalanced data as well.

### Summary

So, this is how to handle class imbalance and perform classification on imbalanced data. Imbalanced data refers to a situation in classification problems where the number of observations in each class significantly differs. In such datasets, one class (the majority class) vastly outnumbers the other class (the minority class). This imbalance can lead to biased models that favour the majority class, resulting in poor predictive performance on the minority class, which is often the class of greater interest.