ASSIGNMENT 4 REPORT

(Arun Jiju Joseph Id:1002140653)

Introduction:

We are using the same data as for the previous homework. Using the df2 table created from last homework where it has already cleaned, dropped some of the variables and but DID NOT CREATE dummy variables

Part 1: Data from our lives:

Described situation or problem from your job, everyday life, current events, etc., for which a classification would be appropriate. The example which I used was based on Email Spam Detection. In this scenario, a machine learning classification system can be employed to automatically categorize incoming emails as either "spam" or "not spam." Other scenarios are also discussed such as in Medical Diagnosis, Sentiment Analysis in Customer Reviews and Image Recognition for Autonomous Vehicles.

Part 2: Data Preprocessing:

The main objective of this step is to clean the data in order to get it ready for analysis. The dataset we're using for Part two is taken from the 1985 Auto Imports Dataset. This data must be processed because it is raw before the analysis can be done. The null values, duplicate entries, etc. in the data may affect the final results. We clean and organise the data in order to make it understandable so that we can draw any inferences from it. Data cleaning is the process of removing unnecessary or undesirable data from a dataset. Data wrangling is the process of getting this clean data into a format that can be read and used for analysis.

We are using the Auto Imports Dataset, which is a CSV file. We'll outline the procedure we'll use to clean this in the manner that follows. The first step is to read the csv file and import all the relevant libraries, such as NumPy, pandas, and sklearn. We will examine the variable data types. The method is demonstrated in the following:

```
data_types = df.dtypes
print(data_types)
```

```
fuel_type         object
body              object
wheel_base       float64
length           float64
width            float64
heights          float64
curb_weight        int64
engine_type       object
cylinders         object
engine_size        int64
bore              object
stroke            object
compression      float64
horse_power       object
peak_rpm          object
city_mpg           int64
highway_mpg        int64
price              int64
dtype: object
```

The three main data types are float64, int64, and object. There are 18 different types of this variable data. Next task is to Replace '?' with None and to Change the variables: bore, stroke, horse_power, peak_rpm to float64.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 201 entries, 0 to 200
Data columns (total 18 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   fuel_type     201 non-null    object 
 1   body          201 non-null    object 
 2   wheel_base    201 non-null    float64
 3   length        201 non-null    float64
 4   width         201 non-null    float64
 5   heights       201 non-null    float64
 6   curb_weight   201 non-null    int64  
 7   engine_type   201 non-null    object 
 8   cylinders     201 non-null    object 
 9   engine_size   201 non-null    int64  
 10  bore          197 non-null    float64
 11  stroke        197 non-null    float64
 12  compression   201 non-null    float64
 13  horse_power   199 non-null    float64
 14  peak_rpm      199 non-null    float64
 15  city_mpg      201 non-null    int64  
 16  highway_mpg   201 non-null    int64  
 17  price         201 non-null    int64  
dtypes: float64(9), int64(5), object(4)
memory usage: 28.4+ KB
```

The datatypes of the variables bore, stroke, horse_power, peak_rpm is changed to float64. The columns that have just null values must also be removed, which is done as shown below.

```
df2.isnull().sum()

fuel_type       0
wheel_base      0
length          0
width           0
heights         0
curb_weight     0
engine_size     0
bore            0
stroke          0
compression     0
horse_power     0
peak_rpm        0
city_mpg        0
highway_mpg     0
price           0
dtype: int64
```

We will not create dummy variables. Instead of creating dummy variables, we will recode the column as bellow.

2.1 We had replaced gas and diesel string values in fuel_type to 0 and 1.

```
df2['fuel_type'] = df2['fuel_type'].replace({'gas': 0, 'diesel': 1})

print(df2['fuel_type'].tail(10))

191    0
192    0
193    0
194    0
195    0
196    0
197    0
198    0
199    1
200    0
Name: fuel_type, dtype: int64
```

2.2: We Defined X and y: where dependent variable is fuel_type, the rest of the variables are our independent variables.

```
#your code
# Define X (independent variables)
X = df2.drop('fuel_type', axis=1)  # Assuming 'fuel_type' is the column you want to predict

# Define y (dependent variable)
y = df2['fuel_type']
```

2.3 Splited the data into training and testing set. Using test_size=0.3, random_state=746 !

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=746)
```

Performing EDA:

Step 1: Descriptions and features

The data set that we are using has in total 15 variables with float64, int64 and object as their data type. To obtain the mean, maximum, minimum, and other statistical figures for each column, we use the describe () function. We describe the data in order to highlight central tendencies and the distributional structure of the dataset.

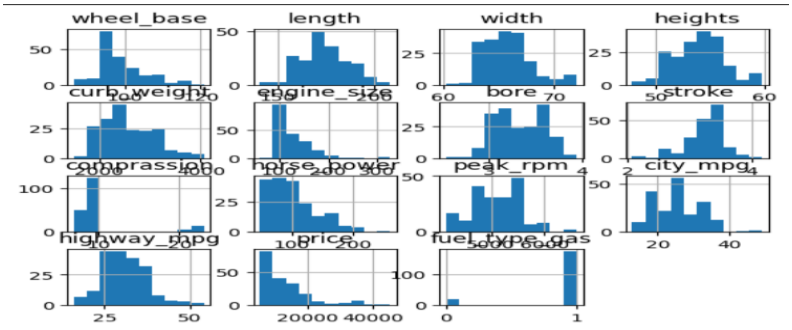
	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_power	peak_rpm	city_mpg	highway_mpg	
count	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.000000	195.(
mean	98.896410	174.256923	65.886154	53.861538	2559.000000	127.938462	3.329385	3.250308	10.194974	103.271795	5099.487179	25.374359	30.841026	13248.(
std	6.132038	12.476443	2.132484	2.396778	524.715799	41.433916	0.271866	0.314115	4.062109	37.869730	468.271381	6.401382	6.829315	8056.3
min	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	2.540000	2.070000	7.000000	48.000000	4150.000000	13.000000	16.000000	5118.(
25%	94.500000	166.300000	64.050000	52.000000	2145.000000	98.000000	3.150000	3.110000	8.500000	70.000000	4800.000000	19.500000	25.000000	7756.5
50%	97.000000	173.200000	65.400000	54.100000	2414.000000	120.000000	3.310000	3.290000	9.000000	95.000000	5100.000000	25.000000	30.000000	10245.(
75%	102.400000	184.050000	66.900000	55.650000	2943.500000	145.500000	3.590000	3.410000	9.400000	116.000000	5500.000000	30.000000	35.000000	16509.(
max	120.900000	208.100000	72.000000	59.800000	4066.000000	326.000000	3.940000	4.170000	23.000000	262.000000	6600.000000	49.000000	54.000000	45400.(

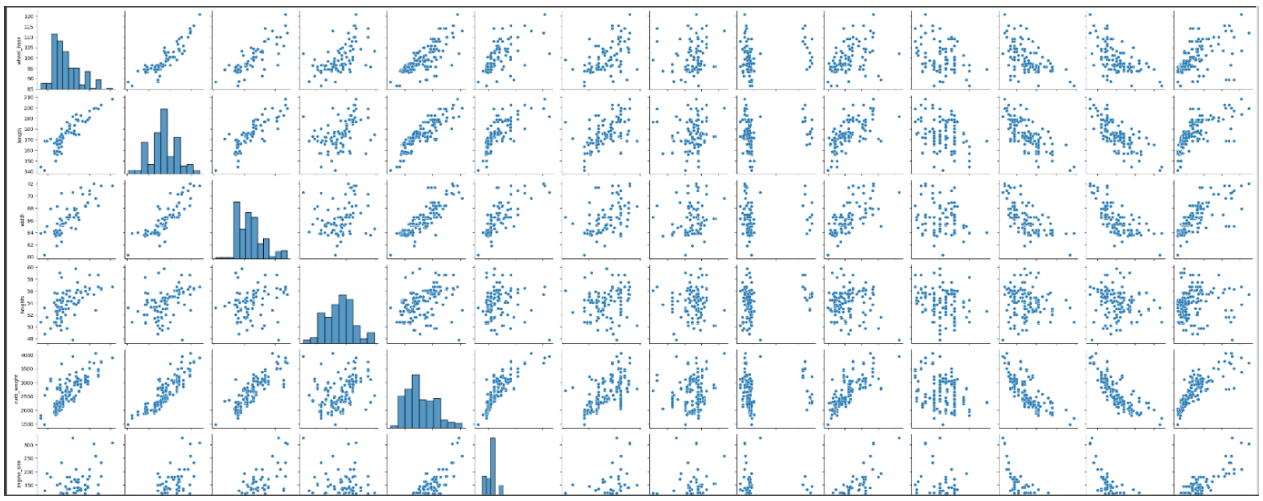
Step 2: Checking Missing value

Looking for null values, duplicate values, and the number of unique values in the columns. The count is used to determine whether the data is balanced.

Step 3: Checking the shape of the data

To determine the shape of the data, we produced historgram and pairplot.



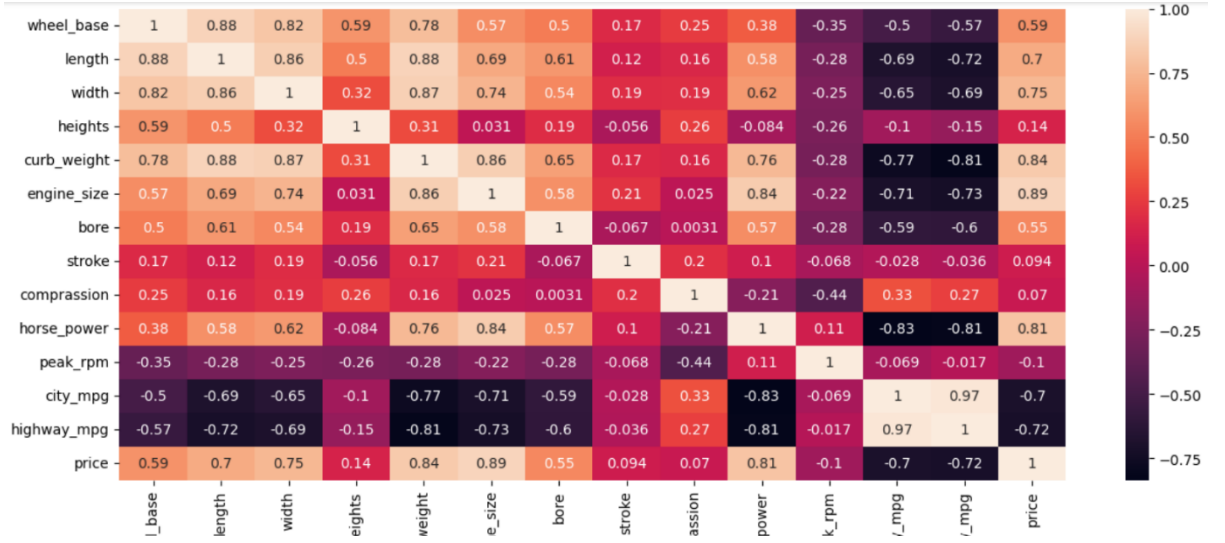


Step 4: Identifying significant correlations

The heatmap, a coloured matrix, displays the correlation between the variables in the data set. All correlations are demonstrated to be positive, and the grid shows how each connection is connected to the others. The variables in the dataset are either positively or directly connected.

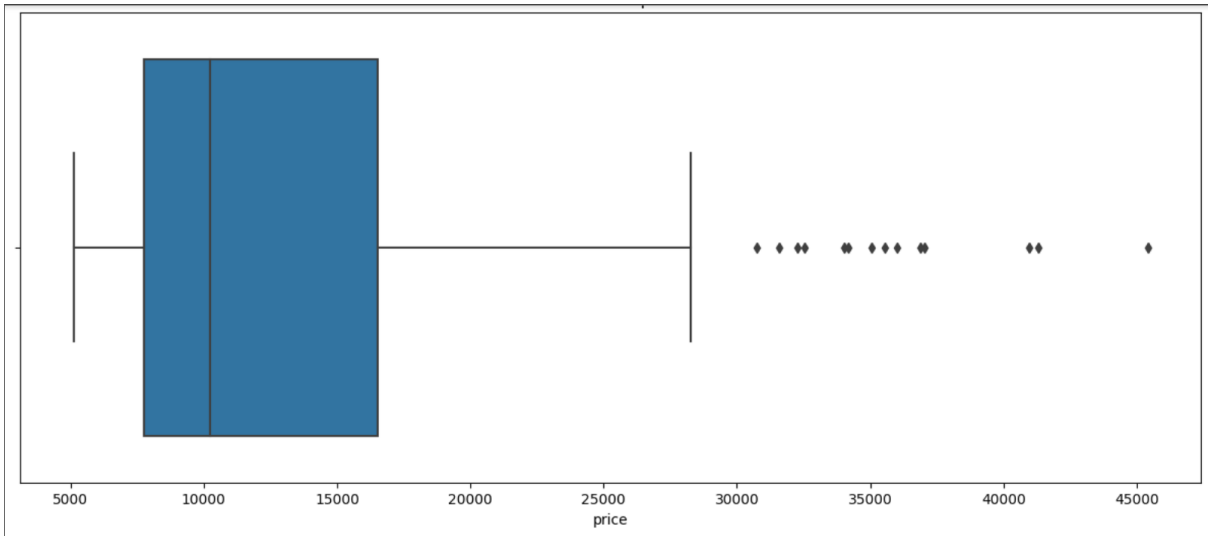
Correlation is used to look at how the variables are related. Based on a scale of -1 to 1, correlation is defined as a negative or indirect connection, +1 as a positive or direct association, and 0 as no correlation. We used Pearson correlation to determine how much the variables were linearly correlated. We created a heatmap to provide as proof of this.

	wheel_base	length	width	heights	curb_weight	engine_size	bore	stroke	compression	horse_power	peak_rpm	city_mpg	highway_mpg	price
wheel_base	1.000000	0.879222	0.819009	0.592500	0.782720	0.569704	0.498228	0.171722	0.247730	0.375541	-0.352331	-0.499126	-0.566355	0.585793
length	0.879222	1.000000	0.858084	0.496218	0.881665	0.687479	0.609437	0.118664	0.160172	0.583813	-0.280986	-0.689660	-0.719324	0.695331
width	0.819009	0.858084	1.000000	0.315834	0.867315	0.740320	0.544311	0.186432	0.190997	0.616779	-0.251627	-0.647099	-0.692220	0.754273
heights	0.592500	0.496218	0.315834	1.000000	0.307732	0.031286	0.189283	-0.055525	0.261160	-0.084412	-0.264078	-0.102367	-0.151188	0.138291
curb_weight	0.782720	0.881665	0.867315	0.307732	1.000000	0.857573	0.645806	0.172785	0.155382	0.760285	-0.278944	-0.772171	-0.812710	0.835729
engine_size	0.569704	0.687479	0.740320	0.031286	0.857573	1.000000	0.583091	0.211989	0.024617	0.842691	-0.219008	-0.710624	-0.732138	0.888942
bore	0.498228	0.609437	0.544311	0.189283	0.645806	0.583091	1.000000	-0.066793	0.003057	0.568527	-0.277662	-0.591950	-0.600040	0.546873
stroke	0.171722	0.118664	0.186432	-0.055525	0.172785	0.211989	-0.066793	1.000000	0.199882	0.100040	-0.068300	-0.027641	-0.036453	0.093746
compression	0.247730	0.160172	0.190997	0.261160	0.155382	0.024617	0.003057	0.199882	1.000000	-0.214401	-0.444582	0.331413	0.267941	0.069500
horse_power	0.375541	0.583813	0.616779	-0.084412	0.760285	0.842691	0.568527	0.100040	-0.214401	1.000000	0.105654	-0.834117	-0.812917	0.811027
peak_rpm	-0.352331	-0.280986	-0.251627	-0.264078	-0.278944	-0.219008	-0.277662	-0.068300	-0.444582	0.105654	1.000000	-0.069493	-0.016950	-0.104333
city_mpg	-0.499126	-0.689660	-0.647099	-0.102367	-0.772171	-0.710624	-0.591950	-0.027641	0.331413	-0.834117	-0.069493	1.000000	0.972350	-0.702685
highway_mpg	-0.566355	-0.719324	-0.692220	-0.151188	-0.812710	-0.732138	-0.600040	-0.036453	0.267941	-0.812917	-0.016950	0.972350	1.000000	-0.715590
price	0.585793	0.695331	0.754273	0.138291	0.835729	0.888942	0.546873	0.093746	0.069500	0.811027	-0.104333	-0.702685	-0.715590	1.000000

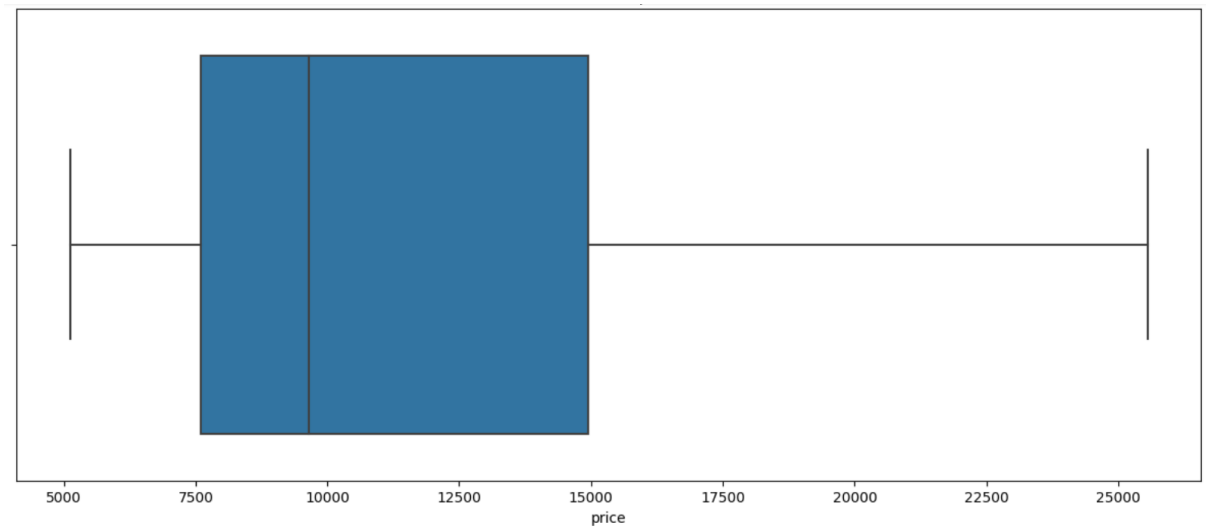


Step 5: Detecting and Handling outliers

Since outliers are outside the box plots, they are easy to identify. Because outliers might make it harder to understand the data, we need to be aware of them. We ultimately plotted a box plot by removing the outliers. We are removing outliers for price variable by plotting a boxplot and the removing from it.



After identifying which rows the outliers are present, we will be removing it from the data, which we will get as:



Part 2: Classification:

3.1 Using Logistic regression we had classified the data and printed the confusion matrix, classification report and AUC.

The model achieved a perfect classification on both classes, with precision, recall, and F1-score all equal to 1.0. The AUC score of 1.0 further supports the model's excellent discrimination ability. The overall accuracy is 100%, indicating that the model correctly classified all classes in the dataset.

```
Confusion Matrix:
 [[50  0]
 [ 0  9]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59


AUC Score: 1.0
```

3.2 Using Naive Bayes we had classified the data and printed the confusion matrix, classification report and AUC.

```
Naive bayes
Confusion Matrix:
[[50  0]
 [ 0  9]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        50
           1       1.00      1.00      1.00         9

    accuracy                           1.00        59
   macro avg       1.00      1.00      1.00        59
weighted avg       1.00      1.00      1.00        59


AUC (Area Under the Curve): 1.0
```

Similar to Logistic Regression, the Naive Bayes model achieved perfect classification on both classes. The AUC score of 1.0 suggests that the model has a perfect true positive rate and a zero false positive rate. The overall accuracy is 100%, indicating that the model correctly classified all instances in the dataset. The consistent high performance across multiple metrics suggests that both Logistic Regression and Naive Bayes models fit the data exceptionally well.

3.3 Using KNN to classify the data. Found the optimal k and then printed the confusion matrix, classification report and AUC.

The model performs well in predicting class 0, as indicated by high precision, recall, and F1-score for class 0. However, the model struggles with class 1, as reflected in low recall and F1-score for class 1.The AUC score of 0.56 indicates there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This score suggests that the model's performance is not significantly better. These results indicate that the KNN model with k=2 has limitations in effectively capturing the characteristics of the dataset, particularly in handling imbalanced classes.

```
Optimal k value: 2
Confusion Matrix:
 [[50  0]
 [ 8  1]]

Classification Report:
              precision    recall  f1-score   support

           0       0.86      1.00      0.93        50
           1       1.00      0.11      0.20         9

    accuracy                           0.86        59
   macro avg       0.93      0.56      0.56        59
weighted avg       0.88      0.86      0.82        59


AUC Score: 0.5555555555555556
```

3.4 Choosing SVM to classify the data. Printed the confusion matrix, classification report and AUC.

```
Confusion Matrix:
 [[50  0]
 [ 9  0]]

Classification Report:
              precision    recall  f1-score   support

           0       0.85      1.00      0.92        50
           1       0.00      0.00      0.00         9

    accuracy                           0.85        59
   macro avg       0.42      0.50      0.46        59
weighted avg       0.72      0.85      0.78        59


AUC Score: 0.8533333333333333
```

Similar to the KNN model, the SVM model performs well in predicting class 0, with high precision, recall, and F1-score for class 0. However, the model struggles with class 1, as reflected in low precision, recall, and F1-score for class 1. The AUC score of 0.85 indicates good overall discrimination ability, but the model's performance on class 1 needs improvement. Similar to the KNN model, the SVM model did not correctly predict any instances of class 1, which may be indicative of a class imbalance issue or challenges in capturing the characteristics of class 1.

3.5 Comparing the results and reporting the findings.

Logistic Regression: Logistic Regression performed exceptionally well on the given dataset, suggesting that the linear decision boundary fits the data well. However, achieving perfect results might also raise concerns about overfitting, especially if the dataset is relatively small.

Naive Bayes: Similar to Logistic Regression, Naive Bayes performed exceptionally well. The Naive Bayes algorithm assumes independence between features, and its simplicity might make it well-suited for certain types of datasets.

KNN (k=2):KNN with k=2 faced challenges, especially in correctly predicting instances of class 1. The low recall for class 1 suggests that the model missed many positive instances. This could be due to the choice of k or the sensitivity of KNN to imbalanced classes.

SVM: SVM faced challenges similar to KNN in correctly predicting instances of class 1. The choice of kernel and hyperparameters could impact SVM performance.

Comparisons:

- Logistic Regression and Naive Bayes performed exceptionally well, achieving perfect results. However, their simplicity might be a limitation, and overfitting could be a concern.
- KNN and SVM struggled with correctly predicting instances of class 1. The choice of k in KNN and the choice of kernel and hyperparameters in SVM could impact their performance.
- The dataset might have class imbalance issues, affecting the performance of KNN and SVM, which struggle with imbalanced data.

If **random forest** was used it could do the best job as it is suitable for very larger datasets and complex relationships.

4. fixing the imbalanced nature of the data with a tool from the lecture. Run one of the classification methods.

```
Confusion Matrix:
[[37 13]
 [ 5  4]]

Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.74      0.80        50
           1       0.24      0.44      0.31         9

    accuracy                           0.69        59
   macro avg       0.56      0.59      0.56        59
weighted avg       0.78      0.69      0.73        59


AUC (Area Under the Curve): 0.7
```

The AUC score is 0.7, indicating moderate discrimination ability. This score suggests that the model has a reasonable ability to distinguish between the two classes. SMOTE has helped address the class imbalance issue, leading to improved performance compared to the previous SVM model without oversampling. The precision, recall, and F1-score for both classes have improved, indicating a better balance between correctly predicting positive instances and avoiding false positives/negatives. The results indicate progress in handling class imbalance using SMOTE, but further refinement and exploration may still be beneficial for optimizing model performance.