Implementation Report for Question 13

By: Katayoon Kobraei

Instructor: Professor Farahani

Abstract:

In this section of the second series of exercises, efforts have been made to achieve the best machine learning models for the dataset "Detection Fraud Claim Insurance Vehicle" using various methods. Initially, preprocessing is performed on the model to make the data more usable. In this context, the data is visualized with relevant charts to derive the best conclusions. Then, the most well-known classification models are tested on this dataset, and attempts are made to identify the best one using methods such as ROC curves, etc. To improve the results for the best model, cross-validation methods will also be employed. The process will further examine imbalanced data. To handle this data, techniques like OverSampling and UnderSampling are utilized. Finally, to ensure that the selected top models yield the best results, boosting techniques are applied to achieve the highest accuracy. Details of all these processes will be discussed further.

Introduction:

The dataset worked on in this exercise is "Detection Fraud Claim Insurance Vehicle." This dataset consists of a list of all accidents that occurred within a specific timeframe, with the ultimate goal of training a model to determine whether a person applying for insurance has committed fraud or not. Since this is a binary classification problem, it is necessary to test the most famous models for such issues, like linear regression, random forest, etc. Therefore, after preparing the data for processing, a comparison of these models is conducted, and their performance is enhanced to find the best possible model.

Part One: Analyzing the Entire Dataset and Data Preparation (Exploratory Data Analysis)

First, the overall structure of the dataset is examined, including all features, total sample count, structure of each feature, and a few random samples. To do this, the file is loaded using the `csv_read()` function and stored in a DataFrame. Then, using the `columns` feature, all dataset features are printed, which include Month, DayOfWeek, WeekOfMonth, Make, AccidentArea, DayOfWeekClaimed, MonthClaimed, WeekOfMonthClaimed, Sex, MaritalStatus, Age, Fault, PolicyType, VehicleCategory, VehiclePrice, FraudFound_P, PolicyNumber, RepNumber, Deductible, DriverRating, Days_Policy_Accident, Days_Policy_Claim, PastNumberOfClaims, AgeOfVehicle, AgeOfPolicyHolder, PoliceReportFiled, WitnessPresent, AgentType, BasePolicy, Year, NumberOfCars, Claim_AddressChange, NumberOfSuppliments. Then, using the `info()` function, the type of each of these features is obtained.

Column Non-Null Count Dtype

--- ------ -------------- -----

0 Month 15420 non-null object

1 WeekOfMonth 15420 non-null int64

2 DayOfWeek 15420 non-null object

3 Make 15420 non-null object

4 AccidentArea 15420 non-null object

5 DayOfWeekClaimed 15420 non-null object

6 MonthClaimed 15420 non-null object

7 WeekOfMonthClaimed 15420 non-null int64

8 Sex 15420 non-null object

9 MaritalStatus 15420 non-null object

10 Age 15420 non-null int64

11 Fault 15420 non-null object

12 PolicyType 15420 non-null object

13 VehicleCategory 15420 non-null object

14 VehiclePrice 15420 non-null object

15 FraudFound_P 15420 non-null int64

16 PolicyNumber 15420 non-null int64

17 RepNumber 15420 non-null int64

18 Deductible 15420 non-null int64

19 DriverRating 15420 non-null int64

20 Days_Policy_Accident 15420 non-null object

21 Days_Policy_Claim 15420 non-null object

22 PastNumberOfClaims 15420 non-null object

23 AgeOfVehicle 15420 non-null object

24 AgeOfPolicyHolder 15420 non-null object

25 PoliceReportFiled 15420 non-null object

26 WitnessPresent 15420 non-null object

27 AgentType 15420 non-null object

28 NumberOfSuppliments 15420 non-null object

29 AddressChange_Claim 15420 non-null object

30 NumberOfCars 15420 non-null object

31 Year 15420 non-null int64

32 BasePolicy 15420 non-null object

Next, the first five samples of this dataset are checked using the `head()` function, and additional descriptions such as the mean of each feature can be obtained using the `describe()` function.

In the next step, which is crucial in data preprocessing, we need to find and remove any missing values (NaN). This is done using the `dropna()` function, which can simply be called on the DataFrame to remove these values based on preference. To check the removal of these values from all features, the `sum()` and `isnull()` functions are called on the DataFrame to print the count of missing or undefined values for each feature. This dataset was already pre-prepared and had no undefined values, so in the end, we have:

Month 0

WeekOfMonth 0

DayOfWeek 0

Make 0

AccidentArea 0

DayOfWeekClaimed 0

MonthClaimed 0

WeekOfMonthClaimed 0

Sex 0

MaritalStatus 0

Age 0

Fault 0

PolicyType 0

VehicleCategory 0

VehiclePrice 0

FraudFound_P 0

PolicyNumber 0

RepNumber 0

Deductible 0

DriverRating 0

Days_Policy_Accident 0

Days_Policy_Claim 0

PastNumberOfClaims 0

AgeOfVehicle 0

AgeOfPolicyHolder 0

PoliceReportFiled 0

WitnessPresent 0

AgentType 0

NumberOfSuppliments 0

AddressChange_Claim 0

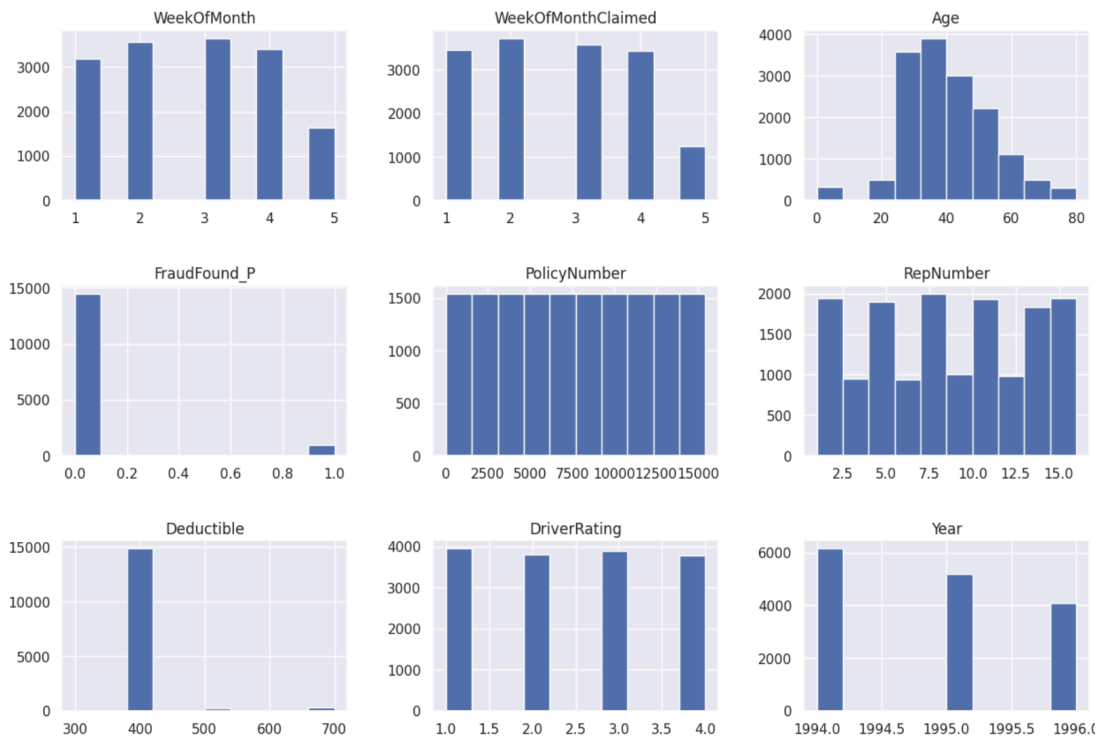
NumberOfCars 0

Year 0

BasePolicy 0

Now that the data is ready for processing, we try to understand the relationships between the data and the value of each feature, as well as the overall dispersion of each feature, by drawing shapes. This is important because, due to the multitude of features, we need to identify the most relevant ones that have the highest correlation with what needs to be predicted.

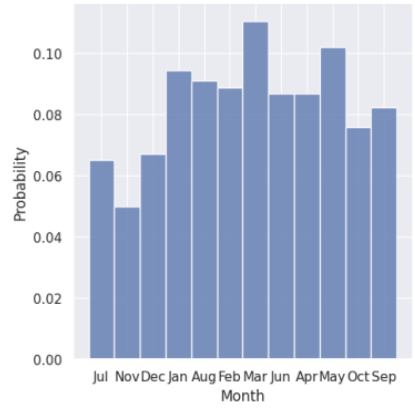
By calling the `hist()` function on the DataFrame, we can obtain a histogram scatter plot for all numerical features to see which ones have practical dispersion. For example, we see that the "number of policies" feature does not have a specific dispersion, so using this feature for the desired prediction would not be appropriate. Conversely, the "age" feature shows appropriate dispersion, which could be useful for the model.

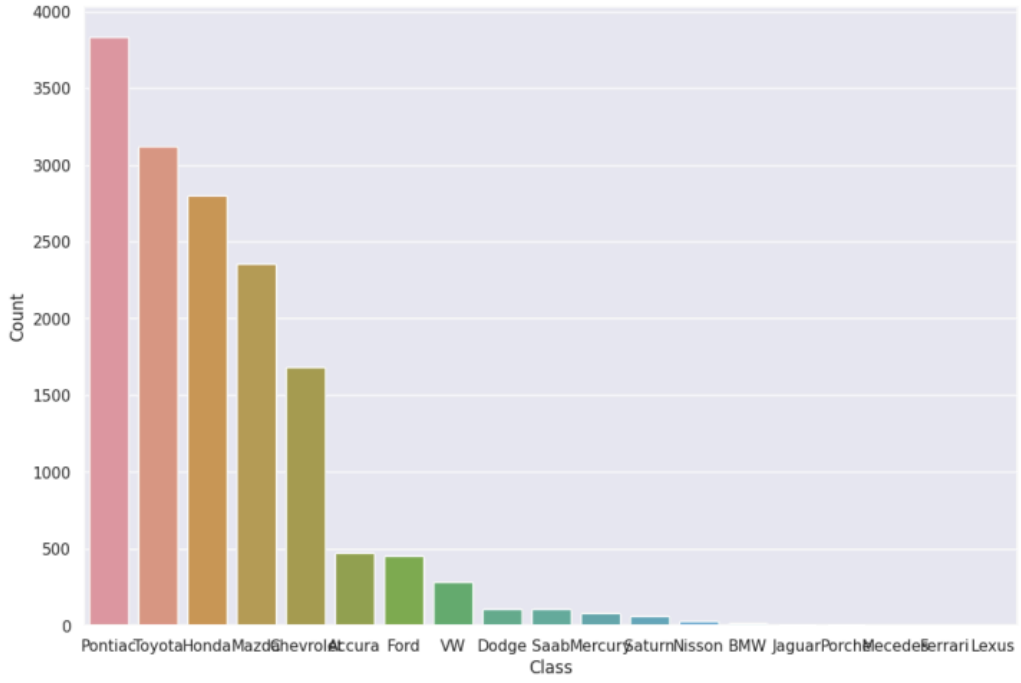


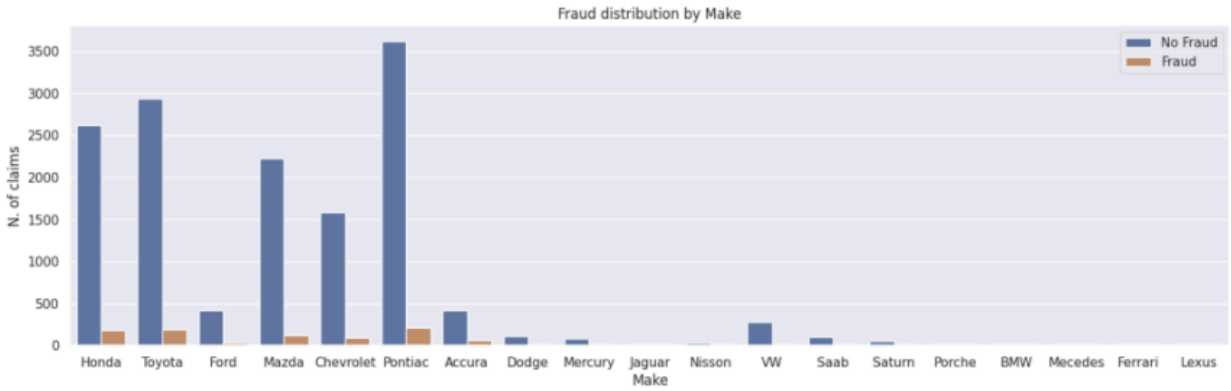
For example, we see that the "number of policies" feature does not have a specific dispersion, so using this feature for the desired prediction would not be appropriate. Conversely, the "age" feature shows appropriate dispersion, which could be useful for the model.

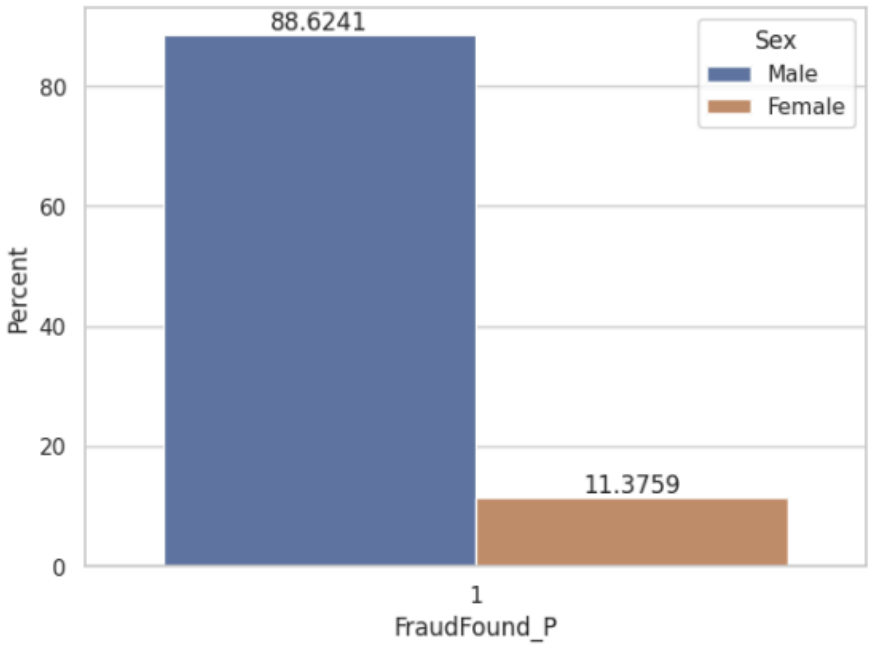
Next, we can plot separate charts for features that appear to have an impact on predictions and examine them individually.

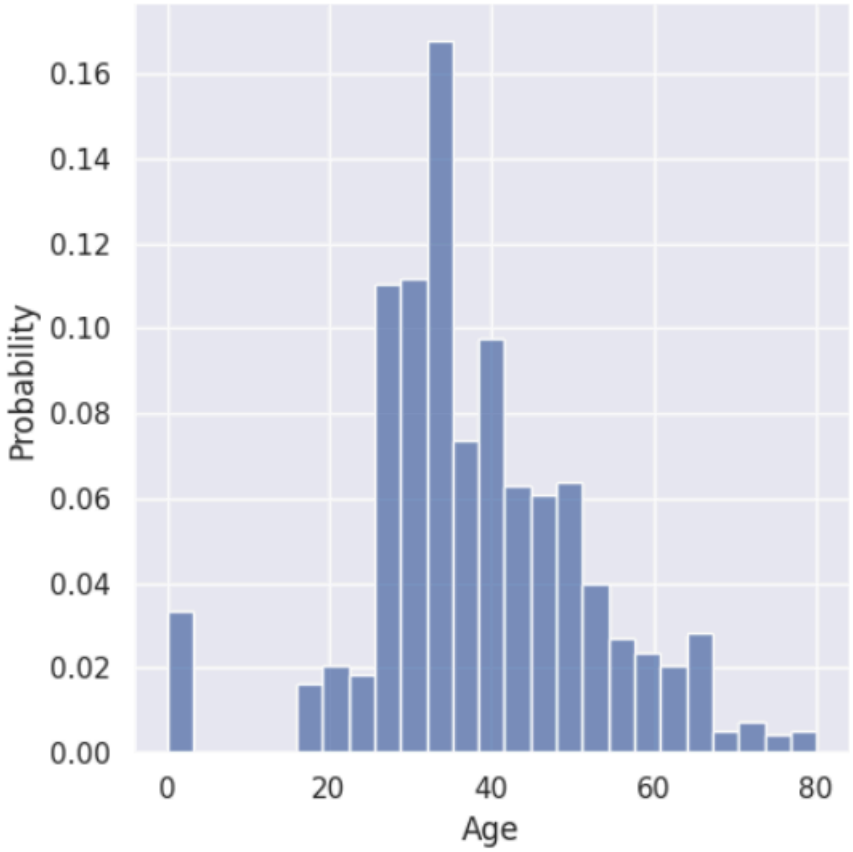
In these charts, we explore the probability of a fraudulent accident being recorded in each month, the number of accidents recorded for each vehicle sample (both fraudulent and real), the registration of real and fraudulent accidents for each car model, and the number of fraudulent accidents recorded for each gender.

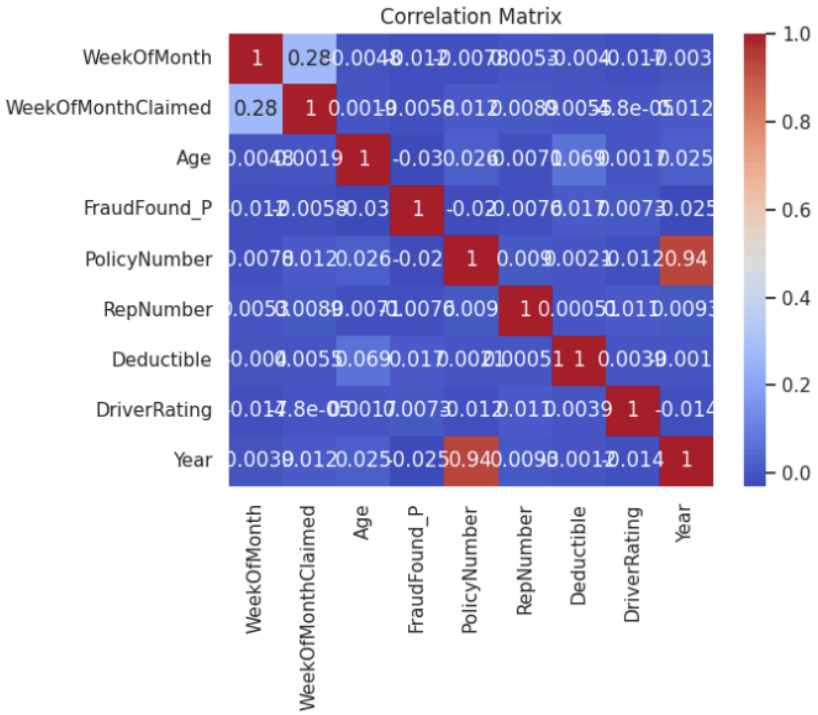





Fraud distribution by Make
No Fraud
Fraud
N. of claims
0
500
1000
1500
2000
2500
3000
3500
Honda
Toyota
Ford
Mazda
Chevrolet
Pontiac
Accura
Dodge
Mercury
Jaguar
Nisson
VW
Saab
Saturn
Porche
BMW
Mecedes
Ferrari
Lexus
Make


88.6241
11.3759
Sex
Male
Female
Percent
0
20
40
60
80
1
FraudFound_P


Probability
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
20
40
60
80
Age


Correlation Matrix
WeekOfMonth
WeekOfMonthClaimed
Age
FraudFound_P
PolicyNumber
RepNumber
Deductible
DriverRating
Year
1.0
0.8
0.6
0.4
0.2
0.0

More examples are provided in the coding section. We can then plot the Correlation matrix among the numerical data to examine the relationship between each. Since there are many features with non-numerical values, it is necessary to encode them, either all or select the ones we need. For this purpose, we use the `LabelEncoder` from the `preprocessing` library. We then apply it to our DataFrame using the `apply()` function. We will see that all non-numeric labels will be converted into integer values, which the model will be able to work with.

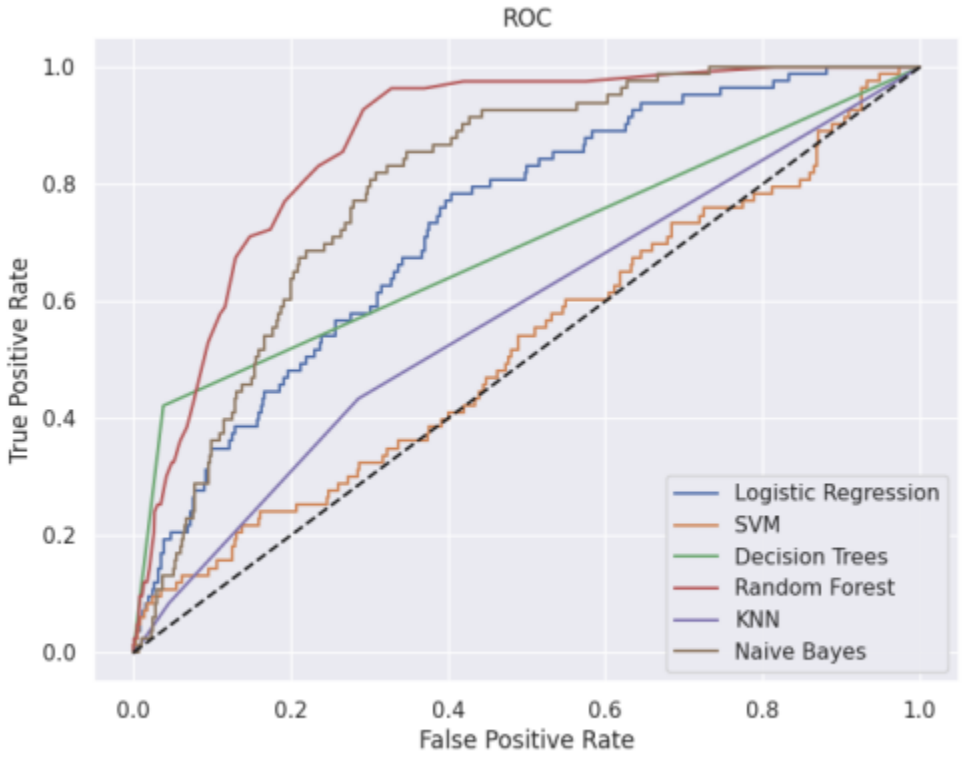
At this stage, the data is ready for modeling, and we can split it into test and train datasets. We explore the age range of individuals who have recorded fraudulent accidents. For example, we see that "FraudFound" has a strong correlation with the "Age" feature, which we can utilize later.

Part Two: Testing Different Models on the Dataset (Different Models Testing)

- **Logistic Regression**: This model is implemented using the ready-to-use library `LogisticRegression`, and the training and testing data are fitted to it to make predictions. We will see that in this model, we achieve an accuracy of approximately 69.10%.
- **SVM**: This model is implemented using the ready-to-use library `SVC`, and the training and testing data are fitted to it to make predictions. We will see that in this model, we achieve an accuracy of approximately 69.13%.
- **Decision Tree**: This model is implemented using the ready-to-use library `DecisionTreeClassifier`, and the training and testing data are fitted to it to make predictions. We will see that in this model, we achieve an accuracy of approximately 69.23%.
- **Random Forest**: This model is implemented using the ready-to-use library `RandomForestClassifier`, and the training and testing data are fitted to it to make predictions. We will see that in this model, we achieve an accuracy of approximately 69.19%.
- **KNN**: This model is implemented using the ready-to-use library `KNeighborsClassifier`, and the training and testing data are fitted to it to make predictions. We will see that in this model, we achieve an accuracy of approximately 69.10%.
- **Naïve Bayes**: This model is implemented using the ready-to-use library `GaussianNB`, and the training and testing data are fitted to it to make

predictions. We will see that in this model, we achieve an accuracy of approximately 63.41%.

For better comparison of these models, we utilize the ROC curve to determine the best-performing models. As seen, the Random Forest model demonstrates better performance on this dataset since the area under the curve (AUC) is greater.



Part Three: Using Cross-Validation (Different Models Testing)

We know that there are different techniques for cross-validation, such as Hold-Out or K-Fold, etc. In this exercise, we used the `StratifiedKFold` library to find the best performance for our models. To do this, we listed our models and calculated the accuracy of each model in separate data batches. Finally, we printed the average accuracy obtained from all batches for each model. This was done to understand the actual performance and check for overfitting.

```
Model: Logistic Regression

Cross-Validation Accuracy: [0.9342252, 0.93526786,
0.93415179, 0.93526786, 0.93415179]

Average Accuracy: 0.9346128961618092

----------------------------------------

Model: Decision Tree

Cross-Validation Accuracy: [0.91415831, 0.94196429,
0.91517857, 0.92745536, 0.89955357]

Average Accuracy: 0.9196620182353878
```

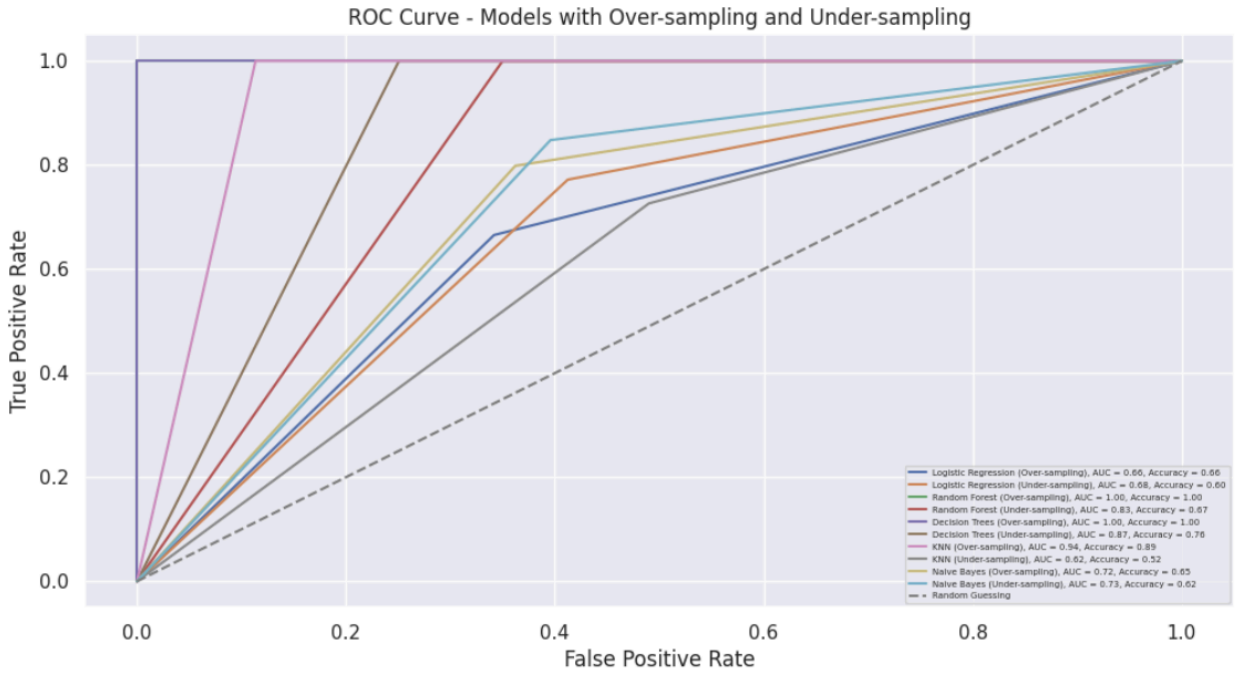
```
----------------------------------------

Model: Random Forest

Cross-Validation Accuracy: [0.93088071, 0.93638393,
0.93973214, 0.93861607, 0.93526786]

Average Accuracy: 0.9361761426978819

----------------------------------------

Model: K-Nearest Neighbors

Cross-Validation Accuracy: [0.92976589, 0.93415179,
0.93191964, 0.93415179, 0.93415179]

Average Accuracy: 0.9328281772575251

----------------------------------------

Model: Support Vector Machines

Cross-Validation Accuracy: [0.9342252, 0.93526786,
0.93526786, 0.93526786, 0.93415179]

Average Accuracy: 0.9348361104475235
```

```
----------------------------------------

Model: Naive Bayes

Cross-Validation Accuracy: [0.80602007, 0.8046875,
0.76227679, 0.77232143, 0.828125]

Average Accuracy: 0.7946861562350692
```

As seen, the SVM and Random Forest models again showed satisfactory results and appear to be good candidates for this dataset.

Part Four: Checking Imbalanced Data

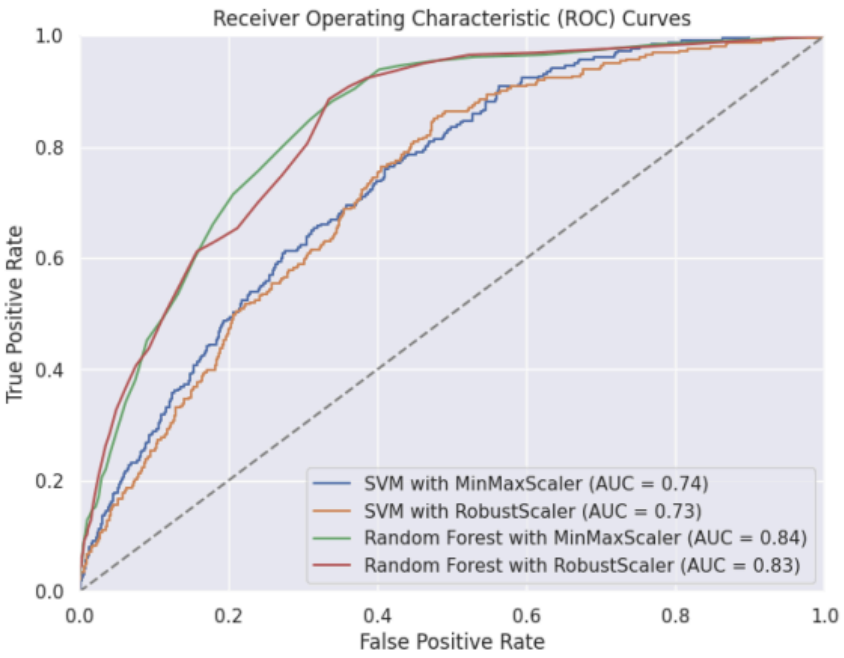
We utilized techniques such as OverSampling and UnderSampling to manage imbalanced data. Essentially, we plotted the model's performance once using these techniques and again without them. Then we plotted the ROC curve for when we used these techniques. We can see that the performance of all models has significantly improved, indicating that there were substantial imbalances in the dataset.



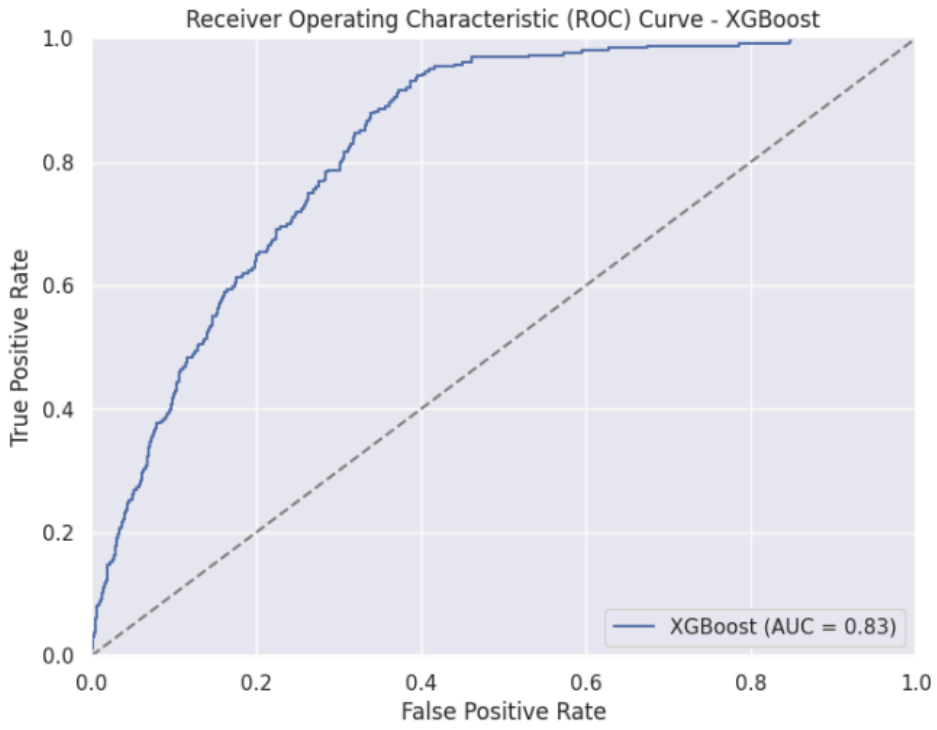
Part Five: Boosting the Performance of SVM and Random Forest Models

For this, we can utilize various techniques. For instance, we decided to perform additional preprocessing on the dataset to further enhance performance. For this, we used functions like MinMaxScaler and RobustScaler.

If we plot the ROC curve, we will see that when using these functions, the area under the curve for the models increases, indicating improved performance and accuracy.



To boost the model, we can also use the XGBoost library. In this case, the model's performance reaches its peak, yielding the best accuracy.



In conclusion, this exercise taught us how to identify the best classification models and work on improving their performance using the mentioned methods. We could have also employed feature engineering techniques to enhance performance even further. Additionally, there were other normalization techniques available for features that we could have examined to determine the best ones to use. Overall, it can be said that we now understand how to solve a classification problem.