Faculty of Engineering & Technology Electrical & Computer Engineering Department

ENCS5141
Intelligent Systems Lab
First Semester 2023/2024

Assignment #2

Name and ID : **Raghad Afaghani - 1192423**

Section : **1**

Instructor : **Yazan Abu Farha**

Teacher Assistant : **Eng. Hanan Awawdeh**

Date : **25.12.2023**

Table Of Content

List Of Images

List Of Tables

1. Literature Review

1.1 Bagging

Bagging, also known as Bootstrap Aggregation, is an easy yet powerful way to combine multiple models in machine learning. like taking small samples (bags) from a big dataset and using them to understand the whole dataset better. like having a puzzle, and each small sample is a piece of that puzzle.looking at these pieces helps us guess what the whole puzzle looks like.Bagging is often used with decision trees, a type of model in machine learning. The main goal here is to take the guesses from each small sample and put them together to make a better overall guess. This way, the final result is more reliable and gives a good overview of what the entire dataset is saying.

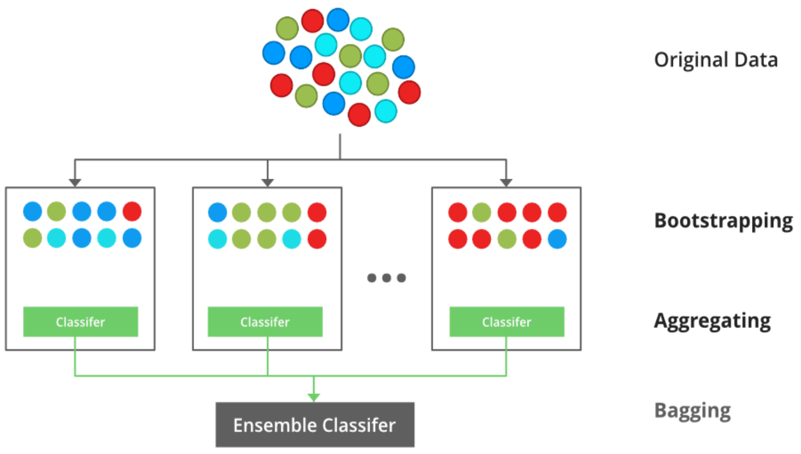


Figure 1. Bagging Technique

Bagging works as follows :

- Creating Subsets: Splitting the main data into several equal smaller sets, randomly choosing data points (with repeats allowed).
- Building Models: For every small set, developing a basic model (weak model).
- Training Independently: The models run in parallel and training done in each model separately using its own data set, all at the same time.
- Combining Results: Finally, merging the outcomes from all models to form the final prediction.

1.2 Boosting

Boosting is a method where models are built in a sequence, each new one improving on the errors of the previous. It starts with simple models, each analyzing the data for mistakes. If a model misjudges a piece of data, that data's importance is increased, making the next model focus more on it. This way, each subsequent model specifically targets what was missed before. By sequentially correcting errors and combining these models, boosting turns weaker individual models into a strong, collective one.

Boosting works as follows :

- Initial Setup: Initializing the dataset with equal weight assigned to each data point.
- Model Training & Error Identification: Input the dataset into the model and identify the wrongly classified data points.
- Weight Adjustment: Increasing the weights for wrongly classified points, decreasing for correct ones, and then normalizing all weights.
- Result Evaluation: Checking if the desired results are achieved. If not, return to step 2.
- Completion: Ending the process once the required results are obtained.

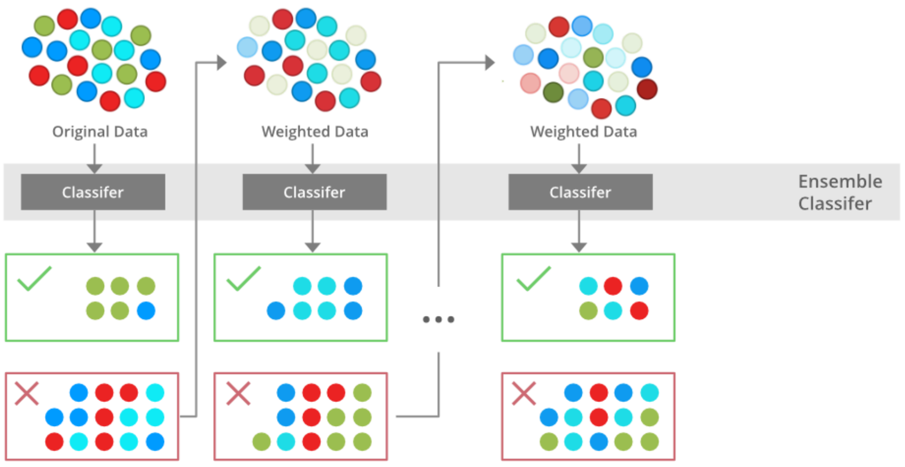


Figure 2. Boosting Technique

Bagging	**Boosting**
Focuses on minimizing overfitting and inconsistency in the model's predictions.	Aims at lowering the error due to oversimplification and improving the model's accuracy.
Equal Weight to Each Model	Performance-Based Weighting
Each Model is created separately	New models learn from previous ones
Classifiers are trained at the same time(in parallel)	Classifiers are trained one after the other.(in sequence)
Use bagging for unstable classifiers with high variance	Choose boosting for simple, stable classifiers with high bias.
Example: The Random forest	Example: XGBoost

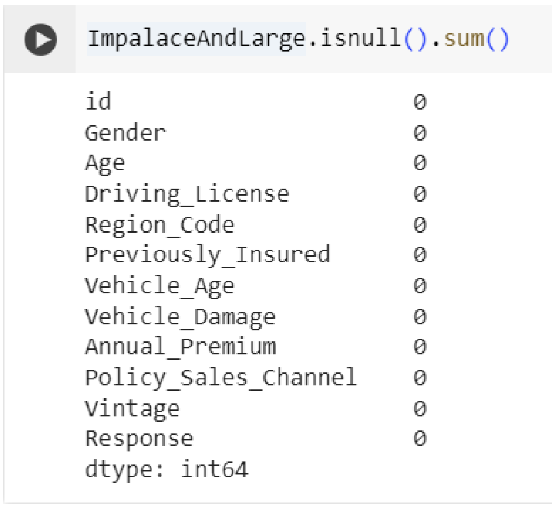
Table 1. Bagging vs Boosting

2. Scenarios designed and analysis

2.1 Imbalance and Large Dataset

This dataset contains information from an insurance company aiming to predict customer interest in vehicle insurance based on their past year's health insurance policies. The dataset includes demographic details (gender, age, region code type), vehicle information (age, damage status), and policy data (premium amount, sourcing channel). By building a predictive model, the company can optimize its communication strategy, effectively reaching out to potential vehicle insurance customers among its existing policyholders, ultimately enhancing its business model and revenue.

- The dataset is large, containing 382,154 rows and 10 columns.
- There is no missing data as shown in Figure[3].



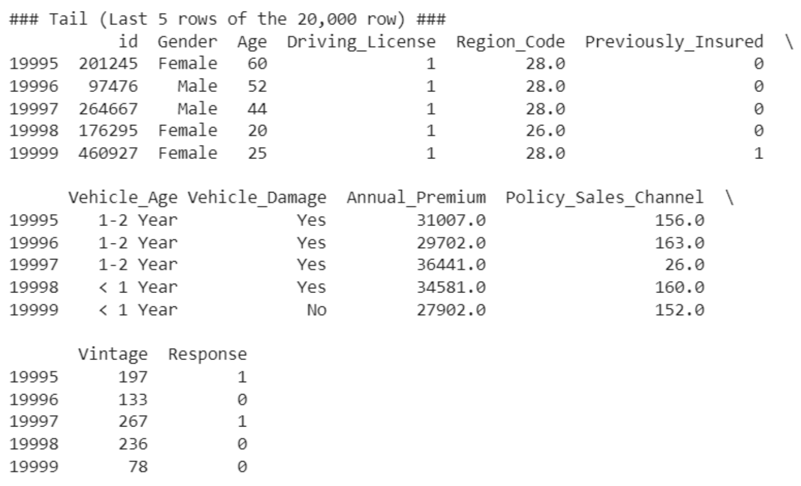
```
ImpalaceAndLarge.isnull().sum()
```

```
id                      0
Gender                  0
Age                     0
Driving_License         0
Region_Code             0
Previously_Insured      0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Policy_Sales_Channel    0
Vintage                 0
Response                0
dtype: int64
```

Figure 3. Checking for missing values

2.1.1 Imbalance Scenario

- The focus here on the first 20,000 rows as shown in Figure[4], to observe just the effects of the imbalance in the dataset.



```
### Tail (Last 5 rows of the 20,000 row) ###
           id  Gender  Age  Driving_License  Region_Code  Previously_Insured  \
19995  201245  Female   60                1         28.0                   0
19996   97476    Male   52                1         28.0                   0
19997  264667    Male   44                1         28.0                   0
19998  176295  Female   20                1         26.0                   0
19999  460927  Female   25                1         28.0                   1

      Vehicle_Age Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  \
19995    1-2 Year            Yes         31007.0                 156.0
19996    1-2 Year            Yes         29702.0                 163.0
19997    1-2 Year            Yes         36441.0                  26.0
19998    < 1 Year            Yes         34581.0                 160.0
19999    < 1 Year             No         27902.0                 152.0

       Vintage  Response
19995      197         1
19996      133         0
19997      267         1
19998      236         0
19999       78         0
```

Figure 4. Imbalance dataset

- The data set is imbalanced as appears in Figure[5]. This means that the dataset has more information about people who are not interested in vehicle insurance compared to those who are interested. which means having lots of [no] responses and only a few [yes] responses, which can make it tricky to predict who might want vehicle insurance.

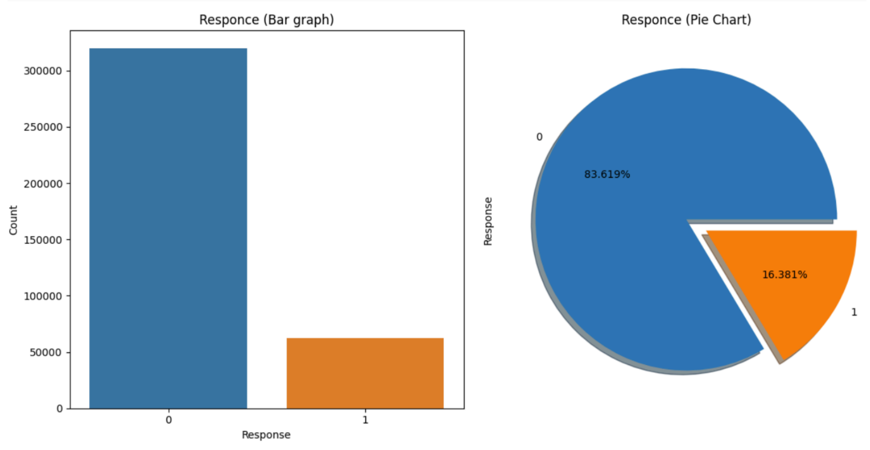
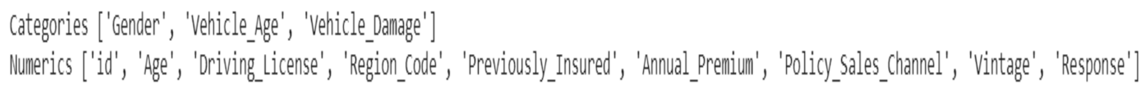


Figure 5. Pie chart for showing the imbalance

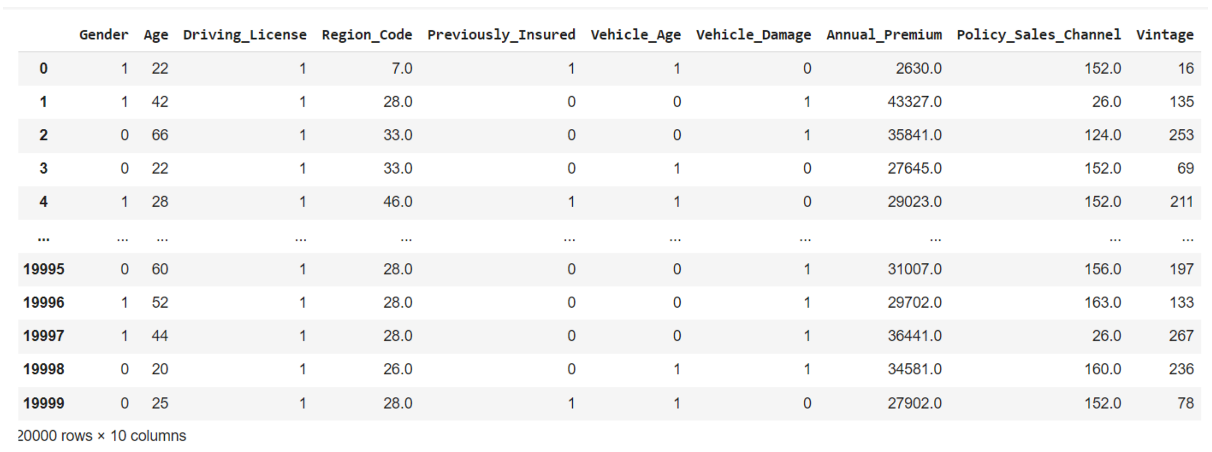
- The dataset contains a mix of categorical and numeric columns as shown in Figure[6]. I've identified the categorical columns by checking their data types and stored them separately. To prepare the categorical data for machine learning, I've applied label encoding, converting the string values in these columns into numerical representations. This preprocessing step ensures that machine learning models can effectively use categorical information for analysis and predictions.



```
Categories ['Gender', 'Vehicle_Age', 'Vehicle_Damage']
Numerics ['id', 'Age', 'Driving_License', 'Region_Code', 'Previously_Insured', 'Annual_Premium', 'Policy_Sales_Channel', 'Vintage', 'Response']
```

Figure 6. Categorical and numerical features

- The dataset after data preparing and setting the Response as target as shown in Figure[7].



	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	1	22	1	7.0	1	1	0	2630.0	152.0	16
1	1	42	1	28.0	0	0	1	43327.0	26.0	135
2	0	66	1	33.0	0	0	1	35841.0	124.0	253
3	0	22	1	33.0	0	1	0	27645.0	152.0	69
4	1	28	1	46.0	1	1	0	29023.0	152.0	211
...	...	...	...	...	...	...	...	...	...	...
19995	0	60	1	28.0	0	0	1	31007.0	156.0	197
19996	1	52	1	28.0	0	0	1	29702.0	163.0	133
19997	1	44	1	28.0	0	0	1	36441.0	26.0	267
19998	0	20	1	26.0	0	1	1	34581.0	160.0	236
19999	0	25	1	28.0	1	1	0	27902.0	152.0	78

20000 rows × 10 columns

Figure 7. Data after processing

The training was done by using two different techniques , Random Forest and XGboost.Both algorithms are tuned using a grid search method to find the best combination of hyperparameters. The Random Forest algorithm operates by constructing multiple decision trees and outputting the mode of their predictions, thus enhancing accuracy and controlling overfitting. XGBoost, on the other hand, is an optimized gradient boosting algorithm known for its speed and performance, which builds sequential trees where each tree corrects the errors of the previous ones.

When assessing the performance of the Random Forest and XGBoost algorithms, it's important to consider two main aspects. Computational Efficiency and Predictive Performance. Computational efficiency encompasses training time and memory usage, with training time reflecting the algorithm's ability to learn from data promptly and memory usage indicating the resources required, both crucial for handling datasets and ensuring adaptability to various computational environments. On the other hand, predictive performance is measured by accuracy, precision, recall, and F1 score, determining how well the algorithm can identify and classify the data points correctly.

In the comparison of Random Forest and XGBoost for predicting customer interest in vehicle insurance, Random Forest achieved a slightly higher accuracy (about 84.8%) and excelled in precision (approximately 57.9%), effectively identifying potential customers. It also used less memory (around 27.63 MB) compared to XGBoost's 44.09 MB. However, XGBoost performed better in terms of recall (about 27.5% against Random Forest's 19.9%) and had a higher F1 Score (approximately 36.2% versus Random Forest's 29.7%). These results suggest that while Random Forest is more precise and memory-efficient, XGBoost offers a more balanced performance, excelling in correctly identifying a higher proportion of actual positives (recall) and achieving a better balance between precision and recall (F1 Score).

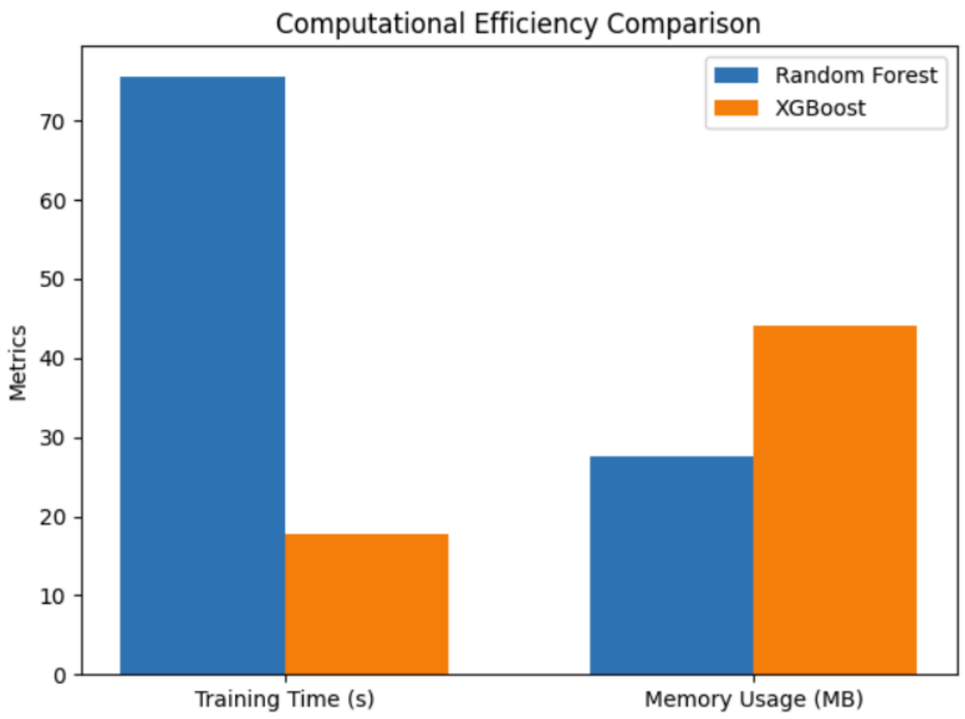


Figure 8. Computational Efficiency for Scenario I

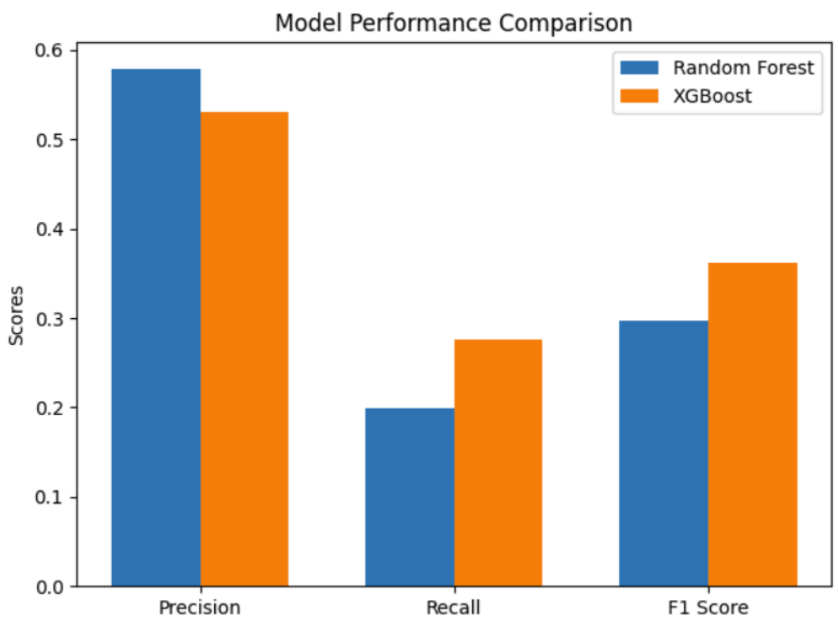
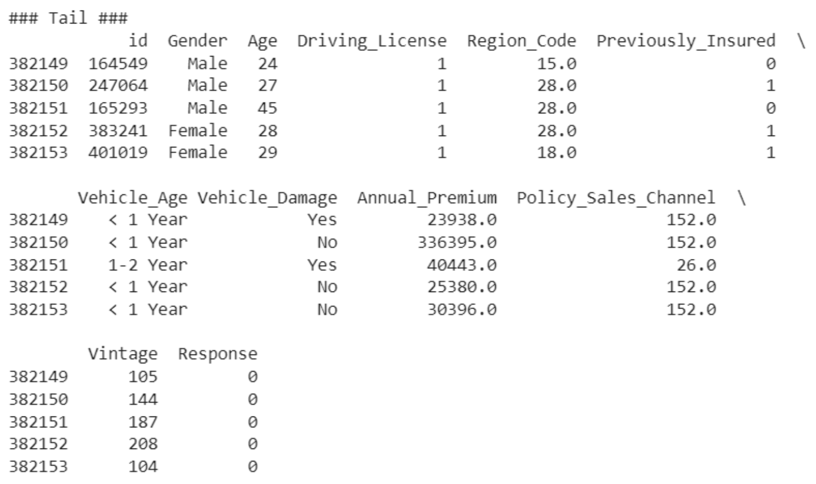


Figure 9. Model performance for scenario I

2.1.2 Large Dataset Scenario

Using the same dataset in its entirety, without any reduction of rows as shown in Figure[10], and with the implementation of techniques to address the issue of data imbalance.



```
### Tail ###
            id  Gender  Age  Driving_License  Region_Code  Previously_Insured  \
382149  164549    Male   24                1         15.0                   0
382150  247064    Male   27                1         28.0                   1
382151  165293    Male   45                1         28.0                   0
382152  383241  Female   28                1         28.0                   1
382153  401019  Female   29                1         18.0                   1

       Vehicle_Age Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  \
382149    < 1 Year            Yes         23938.0                 152.0
382150    < 1 Year             No        336395.0                 152.0
382151    1-2 Year            Yes         40443.0                  26.0
382152    < 1 Year             No         25380.0                 152.0
382153    < 1 Year             No         30396.0                 152.0

        Vintage  Response
382149      105         0
382150      144         0
382151      187         0
382152      208         0
382153      104         0
```

Figure 10. Large dataset scenario

- The dataset contains a mix of categorical and numeric columns as in the previous scenario. So the encoding was applied.
- To address the issue of imbalance, the Random Oversampling approach was taken. This technique involves generating synthetic samples for the minority class in the dataset, effectively increasing its representation. By doing so, the class distribution

becomes more balanced as shown in Figure[11], which can lead to improved model performance and fairness, particularly in scenarios with imbalanced data.

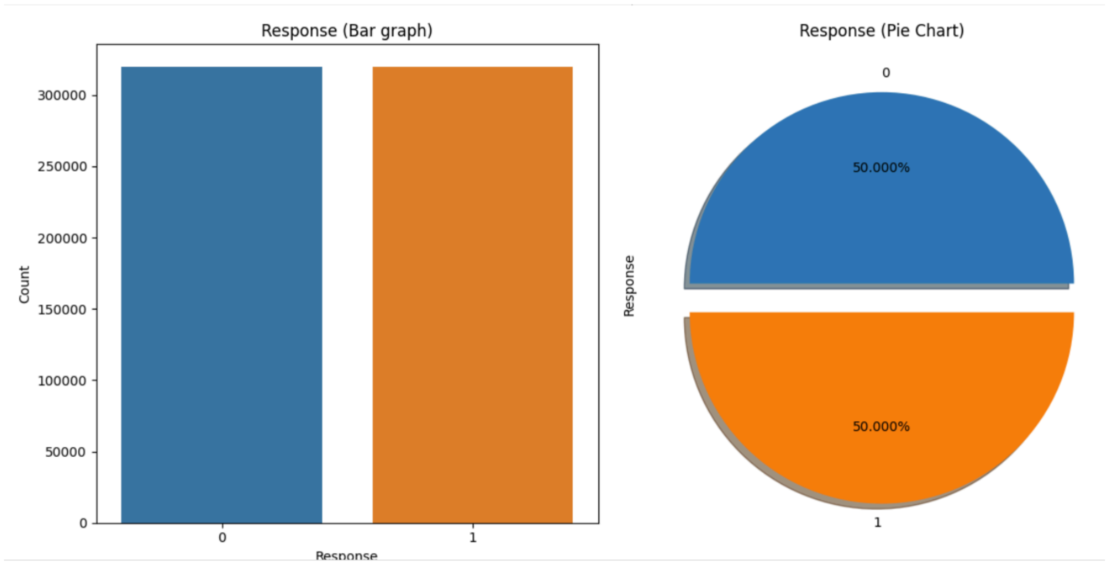
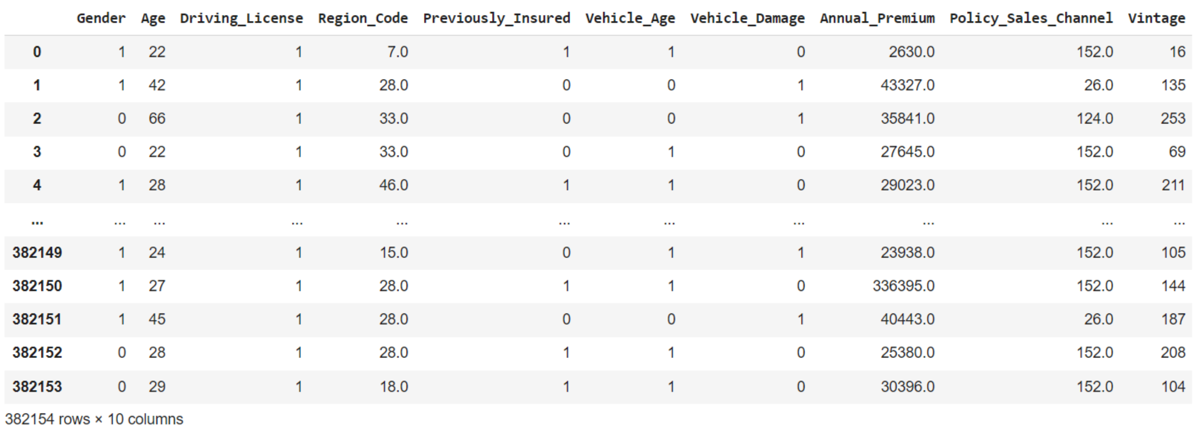


Figure 11. Target after make it balance

- The dataset after data preparing and setting the Response as target as appears in Figure[12].



	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	1	22	1	7.0	1	1	0	2630.0	152.0	16
1	1	42	1	28.0	0	0	1	43327.0	26.0	135
2	0	66	1	33.0	0	0	1	35841.0	124.0	253
3	0	22	1	33.0	0	1	0	27645.0	152.0	69
4	1	28	1	46.0	1	1	0	29023.0	152.0	211
...	...	...	...	...	...	...	...	...	...	...
382149	1	24	1	15.0	0	1	1	23938.0	152.0	105
382150	1	27	1	28.0	1	1	0	336395.0	152.0	144
382151	1	45	1	28.0	0	0	1	40443.0	26.0	187
382152	0	28	1	28.0	1	1	0	25380.0	152.0	208
382153	0	29	1	18.0	1	1	0	30396.0	152.0	104

382154 rows × 10 columns

Figure 12. Dataset after data processing

In comparing the two approaches in Figure[13] and Figure[14] for addressing the issue of data imbalance to see the impact of the two models in the large data , Random Oversampling significantly improved the performance of both the Random Forest and XGBoost models. The Random Forest model achieved an impressive accuracy of 92.08%, with high precision, recall, and F1 score values. However, this improvement came at the cost of significantly higher memory usage, consuming approximately 844.37 MB. On the other hand, the XGBoost model, while still benefiting from the oversampling technique, had a lower accuracy of 84.40%. It exhibited slightly lower precision, recall, and F1 score compared to Random Forest. However, XGBoost demonstrated superior memory efficiency, utilizing only 477.14 MB.

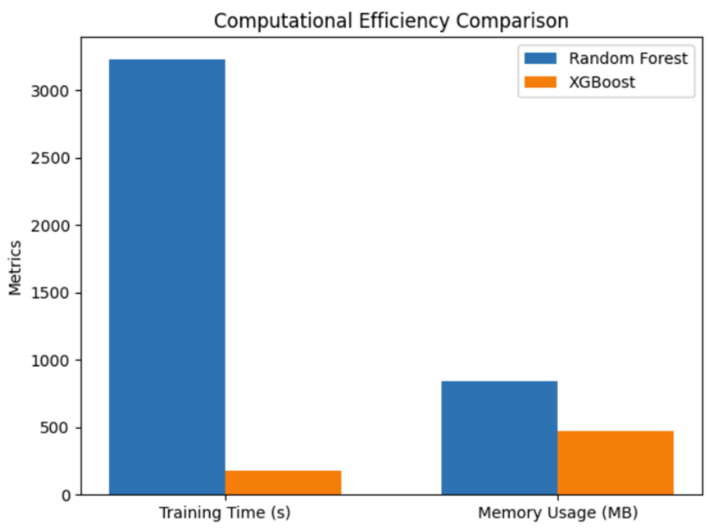


Figure 13. Computational Efficiency for Scenario II

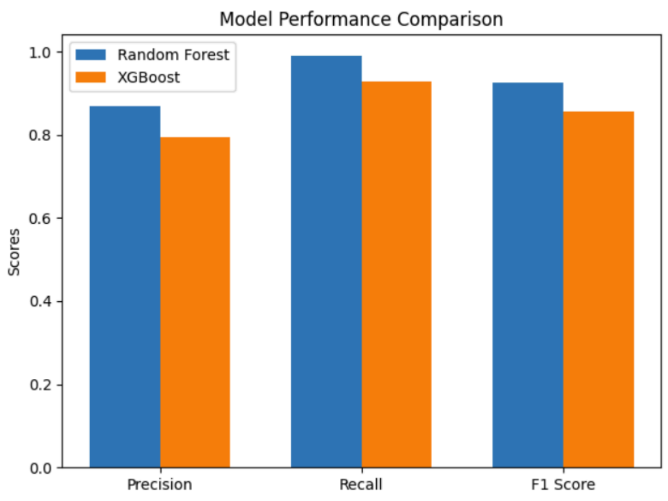
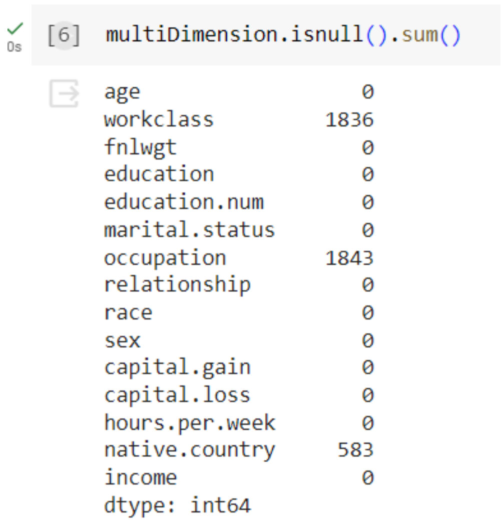


Figure 14. Model performance for scenario II

2.2 Multi Dimensional Dataset

This dataset, created by Ronny Kohavi and Barry Becker from the 1994 Census Bureau data, is used to figure out if someone makes more than $50,000 a year. It includes simple yet important details about people, like their age, the kind of job they have, how much school they've finished, if they're married, what kind of job they do, their family relationships, race, gender, money they earn or lose from investments, how many hours they work each week, and where they're originally from. All these pieces of information help to understand how different things in people's lives can affect how much money they make.

- The dataset contains 32561 rows and 15 columns.
- The dataset, which was initially thought to have no missing data, actually contains missing entries in 'workclass' (1,836), 'occupation' (1,843), and 'native.country' (583) after converting “?” to NaN as appears in Figure[15], while columns like 'age' and 'education' are fully populated. This highlights the need for data cleaning and preprocessing before analysis.



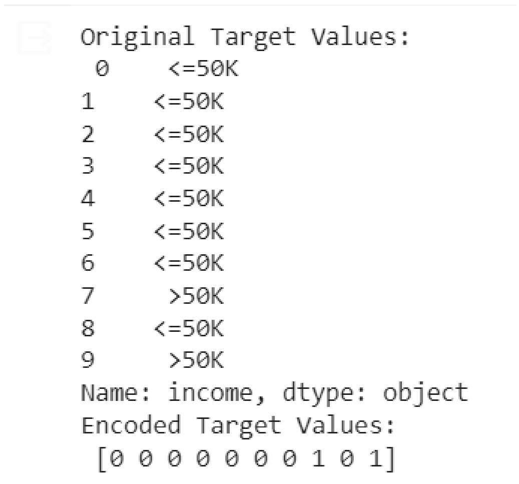
```
[6] multiDimension.isnull().sum()

age                  0
workclass         1836
fnlwgt               0
education            0
education.num        0
marital.status       0
occupation        1843
relationship         0
race                 0
sex                  0
capital.gain         0
capital.loss         0
hours.per.week       0
native.country     583
income               0
dtype: int64
```

Figure 15. Checks for nulls

- To address the missing data in the 'workclass', 'occupation', and 'native.country' columns of the dataset, an imputation technique has been applied. For each of these columns, the missing values are replaced with the most frequently occurring value (mode) in that column.

- In the dataset, the target variable 'income' has been encoded for use in machine learning models. The original categorical labels in the 'income' column, '<=50K' and '>50K', are transformed into numerical values for efficient processing. Specifically, the label '<=50K' is encoded as 0, and the label '>50K' is encoded as 1 using Label Encoding as appears in Figure[16].



```
Original Target Values:
 0    <=50K
1    <=50K
2    <=50K
3    <=50K
4    <=50K
5    <=50K
6    <=50K
7     >50K
8    <=50K
9     >50K
Name: income, dtype: object
Encoded Target Values:
 [0 0 0 0 0 0 0 1 0 1]
```

Figure 16. Target Encoding

- The dataset contains several categorical features such as 'workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', and 'native.country', which are initially in text format. To prepare this data for machine learning, a Label Encoding is applied. This technique transforms each unique text category into a unique numerical identifier, making the data understandable for machine learning algorithms, which typically require numerical input.
- The dataset contains features with varying scales, necessitating the use of the Standard Scaler technique for normalization. This process adjusts each feature to have a mean of zero and a standard deviation of one, ensuring no single feature disproportionately influences the model's performance

The dataset undergoes training and evaluation using RandomForest and XGBoost classifiers within a Python environment. The process involves splitting the data into training and testing sets, then applying GridSearchCV to both models to optimize their hyperparameters. The training effectiveness is assessed through metrics like accuracy, precision, recall, and F1 score, alongside measuring the training time and memory usage. Finally, the results are visualized using bar graphs, comparing the computational efficiency and performance metrics (precision, recall, F1 score) of both models, providing a clear and comprehensive overview of their performance.

In the comparison shown in Figure[17] and Figure[18], XGBoost showed a higher accuracy, at about 87.1%, in predicting whether someone earns more than $50,000 a year, compared to the Random Forest's accuracy of around 86.6%. XGBoost was also more memory-friendly, using just over 18 MB, while Random Forest used nearly 183 MB. For being sure about its predictions, XGBoost scored around 77.6% in precision, almost the same as Random Forest's 77.8%. However, XGBoost was better at correctly identifying the higher earners (known as recall) with a score of about 64.6%, compared to Random Forest's 61.2%. Lastly, XGBoost's F1 score, which combines precision and recall, was higher at approximately 70.5% against Random Forest's 68.5%, showing it balanced accuracy and reliability a bit better.

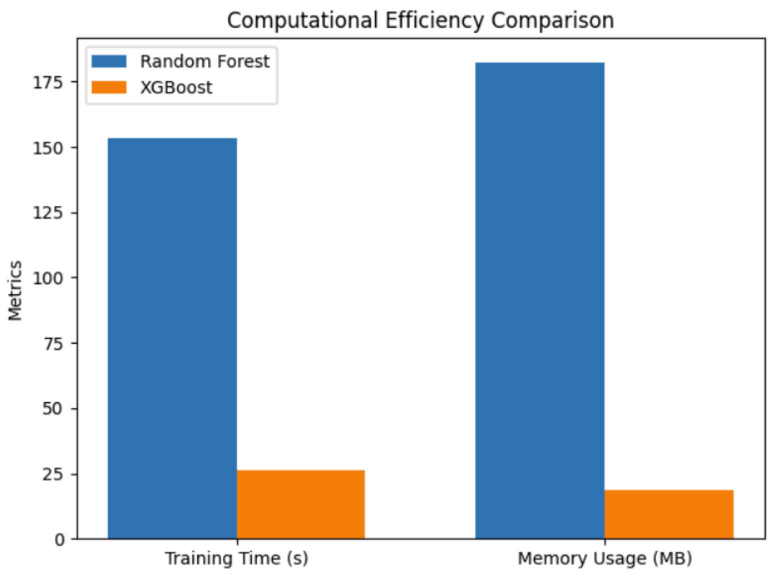


Figure 17. Computational Efficiency for Scenario III

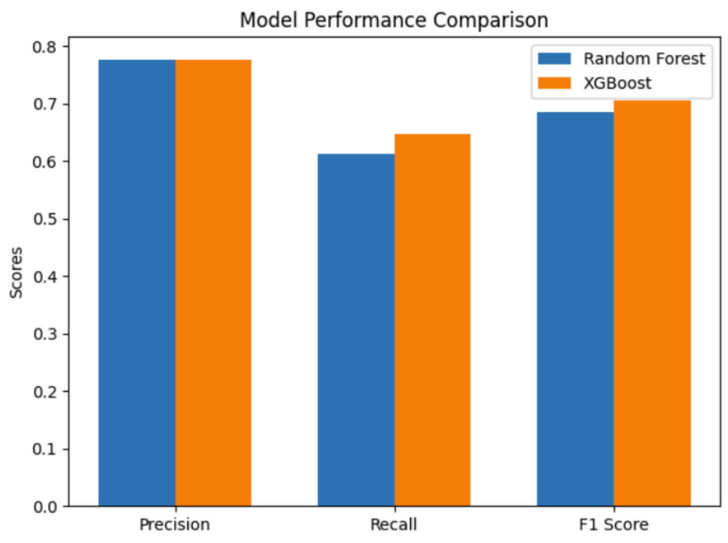


Figure 18. Model performance for scenario III

3. Conclusion and recommendations

- Imbalance Scenario

In the context of uneven datasets for predicting interest in vehicle insurance, XGBoost typically stands out as the better choice. Its advantages in speed and precision, particularly with advanced graphics card technology, make it highly effective for quickly and accurately identifying potential customers. While Random Forest has the benefit of being easier to understand in how it makes decisions, XGBoost's superior performance in terms of accuracy and training efficiency often makes it the more suitable option in such scenarios.

- Large Dataset Scenario

In the large dataset scenario for predicting interest in vehicle insurance, XGBoost is generally the better choice, particularly when GPU acceleration is available. Its ability to utilize GPU resources makes it more adept at handling large datasets, leading to faster processing and lower memory usage, while still maintaining a commendable level of accuracy. This efficiency makes XGBoost well-suited for environments where quick model training and computational resource management are important. On the other hand, Random Forest, despite its slightly higher accuracy, does not benefit from GPU acceleration, resulting in higher memory usage and longer training times. Therefore, in situations where computational efficiency and speed are key considerations, XGBoost has a clear advantage.

- Multi-Dimensional Scenario

In the multi-dimensional dataset scenario for predicting if someone makes more than $50,000 a year, the comparison between Random Forest and XGBoost shows a close match in performance, with XGBoost slightly edging out. XGBoost offers a bit higher accuracy and is significantly more efficient in terms of memory usage, using substantially less memory compared to Random Forest. It also demonstrates slightly better recall and a higher F1 score, indicating a more balanced performance in terms of precision and recall. While Random Forest does provide decent accuracy and precision, its higher memory consumption and slightly lower recall and F1 score make it less optimal in this context. Therefore, considering the efficiency in memory usage and the slightly better overall performance, XGBoost stands out as the more suitable choice for handling multi-dimensional datasets, especially in environments where memory efficiency is a crucial factor.

4. References

- https://www.geeksforgeeks.org/bagging-vs-boosting-in-machine-learning/
- https://www.kaggle.com/datasets/arashnic/imbalanced-data-practice/data
- https://www.kaggle.com/datasets/uciml/adult-census-income/data