Leveraging Machine Learning to Detect

Auto Insurance Fraud

Introduction

Insurance fraud poses a significant challenge to the auto insurance industry, leading to substantial financial losses and operational inefficiencies. Traditional methods of detecting fraudulent claims often fall short due to the complexity and volume of data. Machine learning offers a powerful solution to identify and mitigate fraudulent claims effectively. In this project, we develop a predictive model using auto insurance data to determine whether a claim is fraudulent.

Data Overview

The dataset for this project includes detailed information about insurance policies, customers, and accidents. Below is a description of the independent variables used in the model:

1. **months\_as\_customer**: Number of months the customer has been with the insurance company.
2. **age**: The customer's age.
3. **policy\_number**: Unique ID for tracking the customer’s subscription status and other details.
4. **policy\_bind\_date**: The date the insurance policy was issued.
5. **policy\_state**: The state where the policy was issued.
6. **policy\_csl**: Combined Single Limit of the policy.
7. **policy\_deductible**: The amount the customer must pay out-of-pocket before the insurance kicks in.
8. **policy\_annual\_premium**: The annual premium amount.
9. **umbrella\_limit**: Additional coverage beyond the existing policy limits.
10. **insured\_zip**: Zip code of the insured individual.
11. **insured\_sex**: Gender of the insured individual.
12. **insured\_education\_level**: Education level of the insured individual.
13. **insured\_occupation**: Occupation of the insured individual.
14. **insured\_hobbies**: Hobbies of the insured individual.
15. **insured\_relationship**: Marital status of the insured individual.
16. **capital\_gains**: Profits accrued from the insurance policy.
17. **capital\_loss**: Losses incurred from insurance claims.
18. **incident\_date**: Date of the incident.
19. **incident\_type**: Type of incident.
20. **collision\_type**: Area of damage on the vehicle.
21. **incident\_severity**: Severity of the incident.
22. **authorities\_contacted**: Authorities contacted post-incident.
23. **incident\_state**: State where the incident occurred.
24. **incident\_city**: City where the incident occurred.
25. **incident\_location**: Location of the incident.
26. **incident\_hour\_of\_the\_day**: Hour of the day when the incident occurred.
27. **number\_of\_vehicles\_involved**: Number of vehicles involved in the incident.
28. **property\_damage**: Whether property was damaged.
29. **bodily\_injuries**: Number of bodily injuries.
30. **witnesses**: Number of witnesses.
31. **police\_report\_available**: Availability of a police report.
32. **total\_claim\_amount**: Total amount claimed.
33. **injury\_claim**: Amount claimed for injuries.
34. **property\_claim**: Amount claimed for property damage.
35. **vehicle\_claim**: Amount claimed for vehicle damage.
36. **auto\_make**: Make of the vehicle.
37. **auto\_model**: Model of the vehicle.
38. **auto\_year**: Year of the vehicle.
39. **fraud\_reported**
40. **Data Preparation**:

### **Understanding the Dataset**

The dataset for this project includes detailed information about insurance policies, customers, and accidents.

The target variable, **fraud\_reported** , indicates whether the claim is fraudulent.

* 1. **Data Collection**
  + First Step, we are going to start with **Data loading.**
  + To load the dataset using URL from GitHub, any other repository or we upload the data sheet (.csv, xls ..etc)
  + Here we are going to use python Panda’s library.
  + Use: import pandas as pd

url = ‘<url details >’

df\_data = pd.read\_csv(url)



Once the data is uploaded and we view the data using ‘df.head’, shape, info ..etc and also we can analyse and get some understanding towards the data frame

* 1. **Data Cleaning**
  + **Handling the missing values**

Missing values can distort the model's performance. Therefore, we need to handle them appropriately.

#Find missing values

Here I used ‘ isnull().sum()’ to get null data and I found one of the column got missing details .

|  |  |
| --- | --- |
|  |  |

* + **To handle missing values:**

we used mode method to manage the missing values in '**authorities\_contacted**' To encode categorical variables.

### Question:

### Why We Selected the Mode Method to Manage Missing Values in 'authorities\_contacted' ?

Handling missing values is a critical step in the data preparation process for building a machine learning model. The choice of method for managing missing values can significantly impact the model's performance. In this case, we selected the mode method to handle missing values in the 'authorities\_contacted' column for several reasons:

#### **Categorical Nature of the Variable**

The 'authorities\_contacted' column is a categorical variable that represents whether and which authorities were contacted after an incident. Since categorical variables have discrete values, using statistical measures like the mean or median, which are suitable for continuous variables, would not make sense. The mode, which is the most frequently occurring value in the column, is a suitable measure for categorical data.

* + **Handling unique values:**

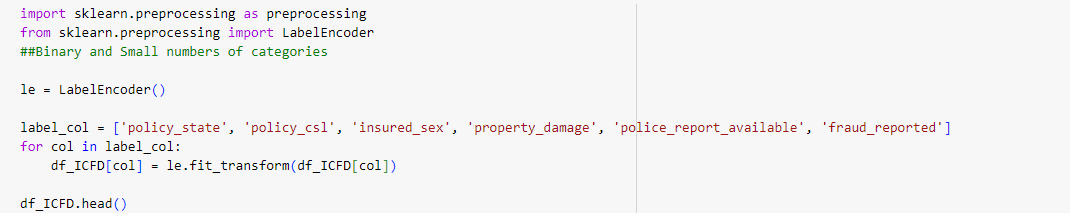
When building a machine learning model, it's crucial to appropriately handle unique values in the dataset. Unique values can be challenging, especially if they are categorical. Below, we discuss how to handle the unique values found in your dataset.

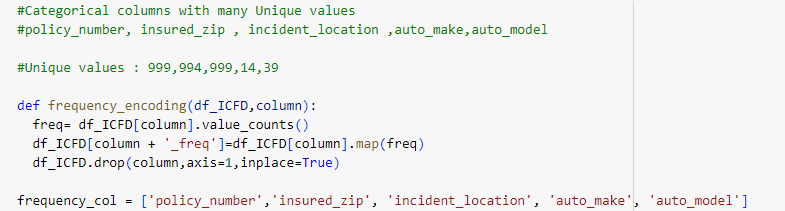
As we found lot’s of unique values and we worked for categorical encoding for categorical variables:

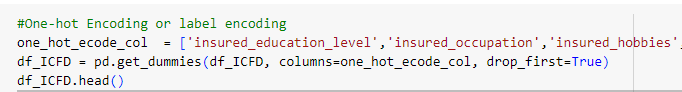
We used one-hot encoding, some Binary and small numbers of categories we used Label Encoder .

1. **Exploratory data Analysis (EDA)**
   * Descriptive Statistics
   * Data Visualization
   * Identifying patterns and relationships

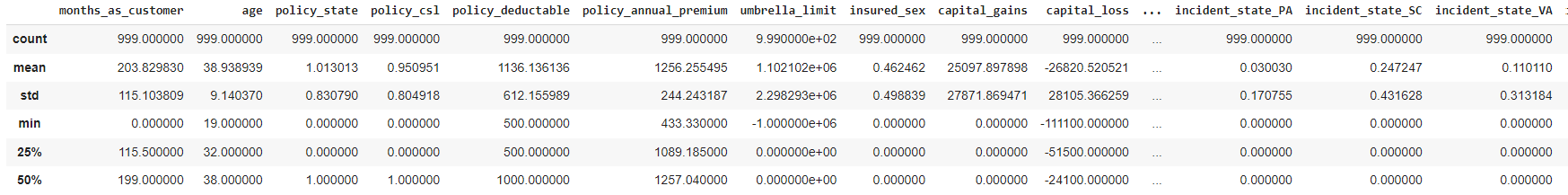
Here we can understand the data and also, we can visualised data

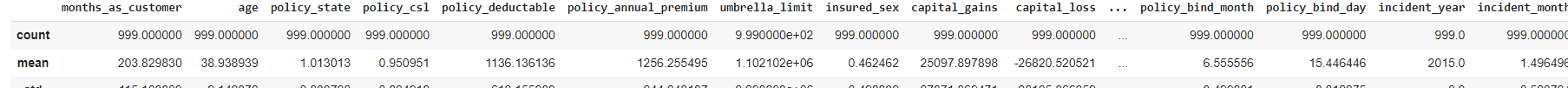






After completing the encoding the data we can see all data in describe:





Now I can see there is no missing values in data set.

Once data cleaning is done, I have checked the correlation matrix.

* + **Key Observations from the Correlation Data**

**Months as Customer and Age**

Here first correlation between Age and months\_as\_customers is having high correlation: 0.92

as well as:

total\_claim\_amount vs injury\_clime (0.81) total\_claim\_amount vs property\_clime (0.81) total\_claim\_amount vs Vehicle\_clime(0.98) injury\_clime vs Vehicle\_clime (0.72) property\_clime vs Vehicle\_clime(0.73) between this four have high positive correlation .

property\_clime vs injury\_clime (0.56)

Interpretation: This high positive correlation indicates that as the number of months as a customer increases the age of the customer also tends to be higher. This is expected as older customers are likely to have longer tenure with the insurance company.

* + **Policy Number and Incident Location**

**Correlation:** The values for these features are not directly correlated as they are categorical and unique identifiers. They have been hashed or encoded for the purpose of machine learning.

**Incident Severity and Total Claim Amount**

Correlation: Likely to be high (not explicitly mentioned in the snippet)

Interpretation: More severe incidents generally result in higher claim amounts. This makes sense as severe accidents usually require more extensive repairs and medical costs.

**Insured Zip Code and Incident Location**

Correlation: Not explicitly high, but there may be some correlation as incidents could occur near the insured's home location.

Interpretation: While the insured zip code and incident location might not be highly correlated, geographical proximity can still be a factor in determining risk and frequency of incidents.

**Capital Gains and Capital Loss**

Correlation: Likely to be negatively correlated

Interpretation: If a policyholder has high capital gains from premiums, they are less likely to have high capital losses from claims. This negative correlation helps in identifying potential fraudulent activities, as disproportionate gains and losses can be a red flag.

**Handling High Cardinality Features**

For features with high cardinality such as policy\_number, incident\_location, and insured\_zip, the following methods are used:

**Hashing Trick**: Converts high cardinality features into a lower dimensional space. This method is memory efficient and reduces the risk of overfitting.

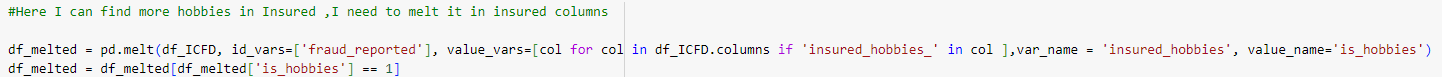
**Frequency Encoding**: Encodes categories based on their frequency, reducing the dimensionality without losing significant information.

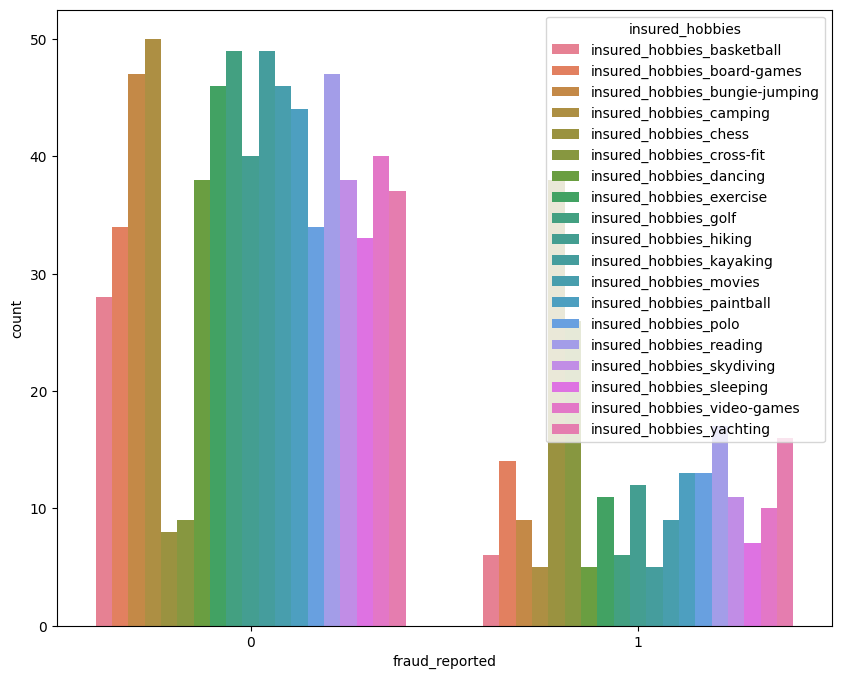
Feature Engineering for DateTime Variables

For policy\_bind\_date and incident\_date, new features such as year, month, and day are extracted. This allows the model to capture seasonal trends and other time-related patterns.

And, we fixed the skewness.

Here we can find the more hobbies In Insured

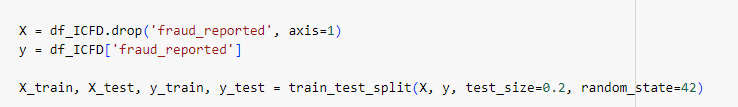




1. Feature Engineering:
   1. Feature Creation

* + Splitting the dataset:

We split the data set into training and testing sets to evaluate our model’s performance.



**Explanation**:

The process of preparing the data for machine learning model training and testing.

Here ‘s a details explanation of each step :

* 1. Feature Transformation and Feature Selection
  + Splitting the features and targets :

**X**: This variable contains all the features (independent variables) from the dataset except the target variable fraud\_reported. The drop method is used to remove the fraud\_reported column from the dataset. This is because fraud\_reported is the column we are trying to predict.

**y**: This variable contains only the target variable fraud\_reported, which indicates whether a claim is fraudulent or not.

* + Splitting the data into training and testing :

**X\_train - Training set of features**

**X\_test – testing set of features**

**y\_train – training set of target variables**

**y\_test – testing set of target variable**

**And splitting the data test size = 20 % reserved for testing and remaining for train 80% : training the model**

**Note :**

**train\_test\_split**: This function from the **sklearn.model\_selection** module is used to split the dataset into training and testing sets.

Here, we can raise one question :

why we need to split the data in to train and test?

**Ans:**

1. **Model training and Model Evaluation**

**Model Training**: The training set is used to fit the machine learning model. The model learns the relationships between the features and the target variable using this data.

**Model Evaluation**: The testing set is used to evaluate the performance of the trained model. It allows us to assess how well the model generalizes to unseen data.

By splitting the data into training and testing sets, we can evaluate the model's performance more accurately and avoid overfitting. Overfitting occurs when the model learns the training data too well, including its noise and outliers, leading to poor performance on new, unseen data.

**In model building we are going to do most important part i.e**

**Model training with different models like Logistic regression and** LGBMClassifier other classifier models :

And I’m using all below models to check the best model.

from sklearn.svm import SVC

from sklearn.ensemble import RandomForestClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.ensemble import BaggingClassifier

import xgboost as xgb

Here I build and tested the model : LGBM Classifier and

**Mode LGBM classification result:**

Best iteration is:

[200] training's f1: 0.785146 valid\_1's f1: 0.58427

F1 Score: 0.5842696629213483

LightGBM Model Performance

Accuracy Score: 0.815

**explanation**:

**Best Iteration:** This refers to the number of boosting iterations (trees) that resulted in the best performance on the validation set.

In this case, the best iteration is 200.

**Training's f1**: 0.785146: This is the F1 score on the training set at the best iteration.

An F1 score of 0.7851 indicates a good balance between precision and recall on the training data.

**valid\_1's f1:** **0.58427**: This is the F1 score on the validation set at the best iteration.

An F1 score of **0.5843** on the validation set is lower than that on the training set, suggesting that the model performs reasonably well but not perfectly on unseen data.

**F1 Score**: The F1 score is a harmonic mean of precision and recall. It is a useful metric for classification problems where true negatives are less important and there is an uneven class distribution.

**F1 Score = 2 X (Precision x Recall)/Precision + Recall**

**Accuracy Score**:

The accuracy score is the ratio of correctly predicted instances to the total instances in the dataset.

**It is calculated as:**

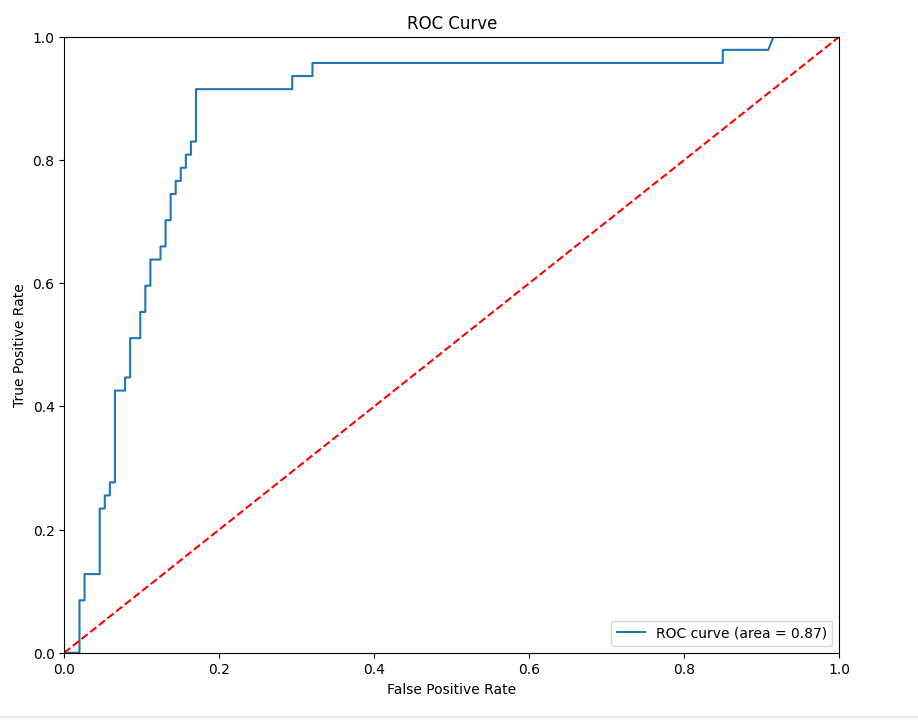
**Accuracy=Number of Correct Predictions/Total Number of Predictions**

​An accuracy score of 0.815 means that the model correctly predicts whether a claim is fraudulent or not 81.5% of the time.

And I have calculated the ROC\_Cyrve : roc\_curve(y)test,pred\_test) calculates the false positive rate (FPR),True positive rate(TPR) and thresholds used to plot the ROC curve.

We can use to plot the Roc Curve i.e plt.plot(fpr,tpr)

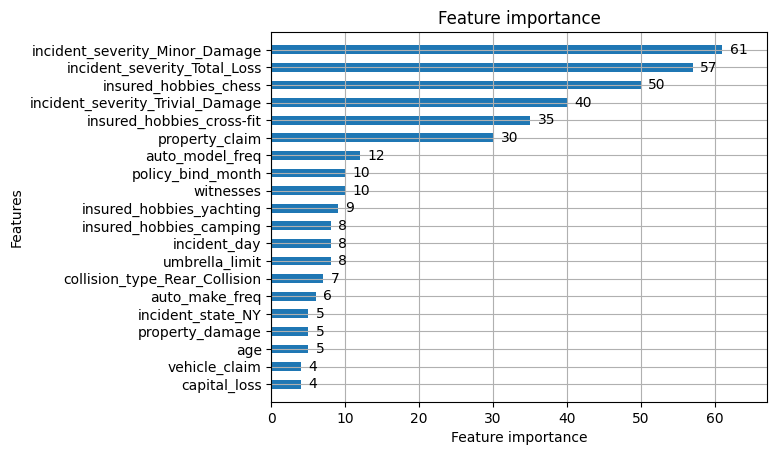
It Provided a detailed ROC curve plot with the AUC value, a reference diagonal line, and a legend, offering more insights into the model's performance.



The true positive rate is high for low values of the false positive rate, indicating that the classifier is effective at distinguishing between the positive and negative classes.

The curve rises sharply towards the top left corner, suggesting that the classifier has a high level of accuracy.

The AUC of 0.87 confirms that the classifier has good discriminative ability.



**Interpretation**

**Higher Importance Scores**: Features with higher scores have a greater impact on the model's decision-making process. For instance, the incident severity and insured hobbies have high importance, suggesting these are key factors in predicting the target variable.

**Lower Importance Scores**: Features with lower scores have less impact. For example, capital\_loss and vehicle\_claim have lower scores, indicating they are less influential.

1. Model selection:

Here we are :



### **Explanation of selected Classifiers**

* + **SVC (Support Vector Classifier)**

'svc': SVC()

**Library**: sklearn.svm

**Description**: A classifier that uses hyperplanes to separate data into classes. It is effective in high-dimensional spaces and suitable for both linear and non-linear classification.

* + **RandomForestClassifier**

'rfc': RandomForestClassifier()

**Library**: sklearn.ensemble

**Description**: An ensemble method that fits multiple decision trees on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

* + **KNeighborsClassifier**

'knc': KNeighborsClassifier()

**Library**: sklearn.neighbors

**Description**: A non-parametric method used for classification and regression. It predicts the class of a data point based on the majority class among its k-nearest neighbors.

* + **GaussianNB (Gaussian Naive Bayes)**

'gau': GaussianNB()

**Library**: sklearn.naive\_bayes

**Description**: A probabilistic classifier based on Bayes' theorem with the assumption of independence between every pair of features. It is particularly suited when the assumption of feature independence holds true.

* + **DecisionTreeClassifier**

'dtc': DecisionTreeClassifier()

**Library**: sklearn.tree

**Description**: A classifier that uses a tree structure to make decisions based on the features of the input data. It splits the data into subsets based on the value of the most significant attribute at each node.

**- AdaBoostClassifier**

'abc': AdaBoostClassifier()

**Library**: sklearn.ensemble

**Description**: An ensemble learning method that combines multiple weak classifiers to form a strong classifier. It works by fitting a sequence of weak learners, each focusing more on the errors of the previous ones.

* + **GradientBoostingClassifier**

'grd': GradientBoostingClassifier()

**Library**: sklearn.ensemble

**Description**: Another ensemble technique that builds trees sequentially, with each new tree attempting to correct errors made by the previous one. It combines the strengths of multiple weak learners to produce a powerful ensemble.

* + **BaggingClassifier**

'bagg': BaggingClassifier()

**Library**: sklearn.ensemble

**Description**: An ensemble method that creates multiple versions of a predictor by training each version on a different random subset of the data. The final prediction is made by averaging the predictions of all the versions.

* + **XGBClassifier (Extreme Gradient Boosting)**

'xgb': xgb.XGBClassifier()

Library: xgboost

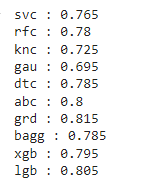
Description: An efficient and scalable implementation of gradient boosting framework. It includes several advanced features for model tuning and performance.

* + **LGBMClassifier (LightGBM)**

'lgb': lgb.LGBMClassifier()

**Library**: lightgbm

**Description**: A gradient boosting framework that uses tree-based learning algorithms. It is designed to be distributed and efficient with the capability to handle large datasets.



After executing we can see above result from all selected models and from here **GradientBoostingClassifier** providing highest Accuracy :0.815(81.5%)

we can consider **GradientBoostingClassifier** is the best model when I compare with other models in above list.