**Problem Statement**

Global Insure, a leading insurance company, processes thousands of claims annually. However, a significant percentage of these claims turn out to be fraudulent, resulting in considerable financial losses. The company’s current process for identifying fraudulent claims involves manual inspections, which is time-consuming and inefficient. Fraudulent claims are often detected too late in the process, after the company has already paid out significant amounts. Global Insure wants to improve its fraud detection process using data-driven insights to classify claims as fraudulent or legitimate early in the approval process. This would minimize financial losses and optimize the overall claims handling process.

**Business Objective**

Global Insure wants to build a model to classify insurance claims as either fraudulent or legitimate based on historical claim details and customer profiles. By using features like claim amounts, customer profiles and claim types, the company aims to predict which claims are likely to be fraudulent before they are approved.

Based on this assignment, you have to answer the following questions:

* How can we analyze historical claim data to detect patterns that indicate fraudulent claims?
* Which features are most predictive of fraudulent behavior?
* Can we predict the likelihood of fraud for an incoming claim, based on past data?
* What insights can be drawn from the model that can help in improving the fraud detection process?

**Data Dictionary**

The insurance claims data has 40 Columns and 1000 Rows. The following data dictionary provides the description for each column present in the dataset:

|  |  |
| --- | --- |
| **Column Name** | **Description** |
| months\_as\_customer | Represents the duration in months that a customer has been associated with the insurance company. |
| age | Represents the age of the insured person. |
| policy\_number | Represents a unique identifier for each insurance policy. |
| policy\_bind\_date | Represents the date when the insurance policy was initiated. |
| policy\_state | Represents the state where the insurance policy is applicable. |
| policy\_csl | Represents the combined single limit for the insurance policy. |
| policy\_deductable | Represents the amount that the insured person needs to pay before the insurance coverage kicks in. |
| policy\_annual\_premium | Represents the yearly cost of the insurance policy. |
| umbrella\_limit | Represents an additional layer of liability coverage provided beyond the limits of the primary insurance policy. |
| insured\_zip | Represents the zip code of the insured person. |
| insured\_sex | Represents the gender of the insured person. |
| insured\_education\_level | Represents the highest educational qualification of the insured person. |
| insured\_occupation | Represents the profession or job of the insured person. |
| insured\_hobbies | Represents the hobbies or leisure activities of the insured person. |
| insured\_relationship | Represents the relationship of the insured person to the policyholder. |
| capital-gains | Represents the profit earned from the sale of assets such as stocks, bonds, or real estate. |
| capital-loss | Represents the loss incurred from the sale of assets such as stocks, bonds, or real estate. |
| incident\_date | Represents the date when the incident or accident occurred. |
| incident\_type | Represents the category or type of incident that led to the claim. |
| collision\_type | Represents the type of collision that occurred in an accident. |
| incident\_severity | Represents the extent of damage or injury caused by the incident. |
| authorities\_contacted | Represents the authorities or agencies that were contacted after the incident. |
| incident\_state | Represents the state where the incident occurred. |
| incident\_city | Represents the city where the incident occurred. |
| incident\_location | Represents the specific location or address where the incident occurred. |
| incident\_hour\_of\_the\_day | Represents the hour of the day when the incident occurred. |
| number\_of\_vehicles\_involved | Represents the total number of vehicles involved in the incident. |
| property\_damage | Represents whether there was any damage to property in the incident. |
| bodily\_injuries | Represents the number of bodily injuries resulting from the incident. |
| witnesses | Represents the number of witnesses present at the scene of the incident. |
| police\_report\_available | Represents whether a police report is available for the incident. |
| total\_claim\_amount | Represents the total amount claimed by the insured person for the incident. |
| injury\_claim | Represents the amount claimed for injuries sustained in the incident. |
| property\_claim | Represents the amount claimed for property damage in the incident. |
| vehicle\_claim | Represents the amount claimed for vehicle damage in the incident. |
| auto\_make | Represents the manufacturer of the insured vehicle. |
| auto\_model | Represents the specific model of the insured vehicle. |
| auto\_year | Represents the year of manufacture of the insured vehicle. |
| fraud\_reported | Represents whether the claim was reported as fraudulent or not. |
| \_c39 | Represents an unknown or unspecified variable. |

**Tasks to be Achieved:**

Here is a summary of the tasks performed to address the fraudulent claim detection problem:

* **1. Data Preparation**
  + Imported necessary libraries such as pandas, numpy, seaborn, and matplotlib.
  + Loaded the dataset from the provided CSV file.
  + Inspected the first few rows, the shape, and the column information of the dataset.
* **2. Data Cleaning**
  + **2.1 Handle null values**
    - Checked for the number of missing values in each column.
    - Handled rows containing null values.
  + **2.2 Identify and handle redundant values and columns**
    - Examined columns for redundant or unnecessary values.
    - Identified and dropped completely empty columns.
    - Identified and dropped rows with illogical or invalid values.
    - Removed columns with a high number of unique values that have limited predictive power.
  + **2.3 Fix Data Types**
    - Converted columns containing date information to the correct datetime data type.
* **3. Train-Validation Split**
  + Imported the train\_test\_split function from scikit-learn.
  + Defined the feature (X) and target (y) variables.
  + Split the dataset into a 70% training set and a 30% validation set, using stratification on the target variable.
* **4. EDA on Training Data**
  + **4.1 Perform univariate analysis**
    - Selected numerical columns for analysis.
    - Visualized the distribution of numerical features using plots.
  + **4.2 Perform correlation analysis**
    - Created and visualized a correlation matrix (heatmap) for numerical features to check for multicollinearity.
  + **4.3 Check class balance**
    - Examined and visualized the distribution of the target variable to check for class imbalance.
  + **4.4 Perform bivariate analysis**
    - Analyzed the relationship between categorical features and the target variable using target likelihood.
    - Explored and visualized the relationships between numerical features and the target variable.
* **5. EDA on Validation Data (Optional)**
  + Conducted similar univariate, correlation, class balance, and bivariate analyses on the validation dataset to ensure consistency with the training data.
* **6. Feature Engineering**
  + **6.1 Perform resampling**
    - Used RandomOverSampler to address class imbalance in the training data.
  + **6.2 Feature Creation**
    - Created new features from existing ones (e.g., from date/time columns) for both training and validation sets.
  + **6.3 Handle redundant columns**
    - Dropped redundant or low-value columns identified during EDA and feature creation.
  + **6.4 Combine values in Categorical Columns**
    - Grouped infrequent categories in categorical columns to reduce sparsity.
  + **6.5 Dummy variable creation**
    - Identified categorical columns for encoding.
    - Created dummy variables for categorical features in both training and validation sets.
    - Encoded the dependent feature into a numerical format.
  + **6.6 Feature scaling**
    - Imported a scaling tool from scikit-learn.
    - Scaled the numerical features in both training and validation data.
* **7. Model Building**
  + **7.1 Feature selection (for Logistic Regression)**
    - Imported RFECV (Recursive Feature Elimination with Cross-Validation).
    - Applied RFECV to select the most relevant features for the logistic regression model.
    - Retained the selected features for model training.
  + **7.2 Build Logistic Regression Model**
    - Added a constant to the training data for the statsmodels API.
    - Fitted a logistic regression model and examined the summary, including p-values.
    - Calculated the Variance Inflation Factor (VIF) for the selected features to check for multicollinearity.
    - Made initial predictions on the training data using a 0.5 cutoff.
    - Evaluated the model's performance using accuracy, confusion matrix, sensitivity, specificity, precision, recall, and F1-score.
  + **7.3 Find the Optimal Cutoff**
    - Plotted the ROC curve to visualize the true positive rate vs. false positive rate.
    - Analyzed the trade-off between sensitivity and specificity at various probability cutoffs.
    - Selected an optimal cutoff and made final predictions on the training data.
    - Re-evaluated the model's performance with the new cutoff.
    - Plotted the precision-recall curve to analyze the trade-off between precision and recall.
  + **7.4 Build Random Forest Model**
    - Imported necessary libraries for Random Forest.
    - Built a baseline Random Forest classifier.
    - Extracted and visualized feature importance scores to select the most important features.
    - Trained the model using only the selected important features.
    - Generated predictions and evaluated the model on the training data using accuracy, confusion matrix, and other relevant metrics.
    - Used cross-validation to check for model overfitting.
  + **7.5 Hyperparameter Tuning**
    - Used GridSearchCV to find the optimal hyperparameters for the Random Forest model.
    - Built the final Random Forest model using the best hyperparameters found.
    - Evaluated the final tuned model on the training data.
* **8. Predicting and Model Evaluation**
  + **8.1 Make predictions over validation data using logistic regression model**
    - Prepared the validation data by selecting the relevant features and adding a constant.
    - Made predictions on the validation data using the trained logistic regression model.
    - Applied the optimal cutoff to the predicted probabilities to get the final predictions.
    - Evaluated the logistic regression model's performance on the validation data using accuracy, confusion matrix, sensitivity, specificity, precision, recall, and F1-score.
  + **8.2 Make predictions over validation data using random forest model**
    - Prepared the validation data by selecting the important features identified earlier.
    - Made predictions on the validation data using the final tuned Random Forest model.
    - Evaluated the Random Forest model's performance on the validation data using accuracy, confusion matrix, and other relevant metrics.
* **Evaluation and Conclusion**
  + Summarized the findings, compared the performance of the two models, and provided concluding remarks on the effectiveness of the developed solution for fraudulent claim detection.