Fraudulent Claim Detection Case Study

Batch : DS_C72

By - Nidhi Kumari

Summary

- Problem Statement
- Business Objective
- Approach
- Results

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 Objectives

- 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?

Approach

- We have followed the below mentioned steps for completing this assignment:
  1. Data Preparation
  2. Data Cleaning
  3. Train Validation Split 70-30
  4. EDA on Training Data
  5. EDA on Validation Data (optional)
  6. Feature Engineering
  7. Model Building
  8. Predicting and Model Evaluation

Results – Data Preparation

- 1.1 Load the Data

```
[ ]: # Check at the first few entries
df.head()
```

[4]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	...	police_
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0	466132	...	
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000	468176	...	
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	5000000	430632	...	
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000	608117	...	
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000	610706	...	

5 rows × 40 columns

```
[ ]: # Inspect the shape of the dataset
df.shape
```

```
[5]: (1000, 40)
```

Results – Data Cleaning

- 2.1.1 Handle null values

authorities_contacted	91
incident_state	0
incident_city	0
incident_location	0
incident_hour_of_the_day	0
number_of_vehicles_involved	0
property_damage	0
bodily_injuries	0
witnesses	0
police_report_available	0
total_claim_amount	0
injury_claim	0
property_claim	0
vehicle_claim	0
auto_make	0
auto_model	0
auto_year	0
fraud_reported	0
_c39	1000

- From above we can see entire column with name "_c39" has null values while column named "authorities_contacted" have 91 null values.

Results – Data Cleaning

- 2.1.2 Handle rows containing null values

```python
# Handle the rows containing null values

#In addition to above identified columns having null values, we also found some columns having entires "?" which are actually missing data.
# we will replace thise entries with na and then again find the missing values before treatment.
df = df.replace('?', np.nan)
print("Final counts for missing values after replacing ? by na")
print(df.isnull().sum())
print("############")
missing_percentage = df.isnull().sum() / len(df) * 100
print("Missing % for each column")
print(missing_percentage)
columns_to_drop = missing_percentage[missing_percentage > 40].index
print("############")
print(" Columns supposed to be droped")
print(columns_to_drop)
columns_to_impute = missing_percentage[(missing_percentage > 0) & (missing_percentage < 40)].index
print("############")
print(" Columns supposed to be imputed")
print(columns_to_impute)
# Impute missing values in 'collision_type', 'property_damage', 'police_report_available','authorities_contacted' as value is less than 40 %
df['collision_type'].fillna(df['collision_type'].mode()[0], inplace=True)
df['property_damage'].fillna(df['property_damage'].mode()[0], inplace=True)
df['police_report_available'].fillna(df['police_report_available'].mode()[0], inplace=True)
df['authorities_contacted'].fillna(df['authorities_contacted'].mode()[0], inplace=True)
```

- Thus after analysis it is found that 5 columns have missing data among which "_c39" is completely missing while rest four columns have missing data lesser than 35-36%.
- Thus we will drop _c39 while retain other columns with proper imputation (by mode as these are categorical).

Results – Data Cleaning

- 2.2.4 Identify and handle redundant values and columns

```
# Write code to display all the columns with their unique values and counts and check for redundant values
for col in df.columns:
    print(f"\nColumn: {col}")
    print(df[col].value_counts())
```

1. Checking for Redundant Columns:
2. Column 'policy_number' is having >90% unique counts. High variability. Thus is a redundant feature.
3. Column 'policy_bind_date' is having >90% unique counts. High variability. Thus is a redundant feature.
4. Column 'policy_annual_premium' is having >90% unique counts. High variability. Thus is a redundant feature.
5. Column 'insured_zip' is having >90% unique counts. High variability. Thus is a redundant feature.
6. Column 'incident_location' is having >90% unique counts. High variability. Thus is a redundant feature.
7. Column '_c39' is completely empty. Should be dropped

Results – Train-Validation Split

- 3.1,3.2 & 3.2

3. Train-Validation Split [5 marks]

3.1 Import required libraries

```
# Import train-test-split
from sklearn.model_selection import train_test_split
```

3.2 Define feature and target variables [2 Marks]

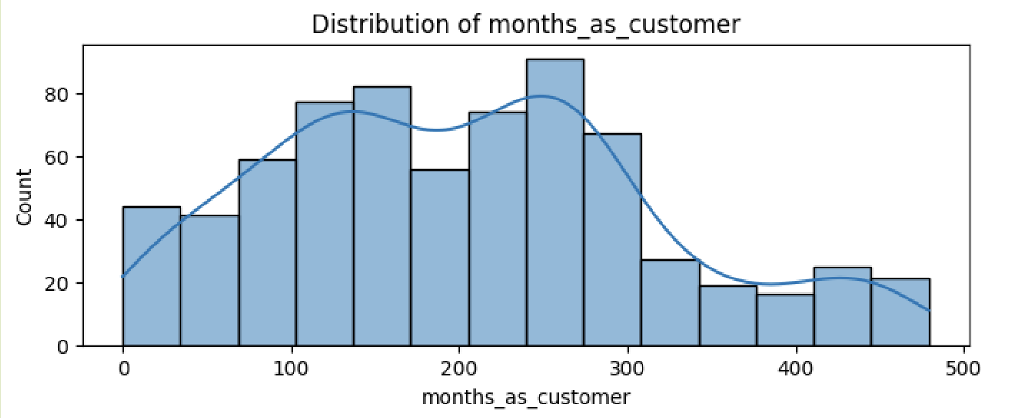
```
# Put all the feature variables in X
X = df.drop('fraud_reported', axis=1)
# Put the target variable in y
y = df['fraud_reported']
```

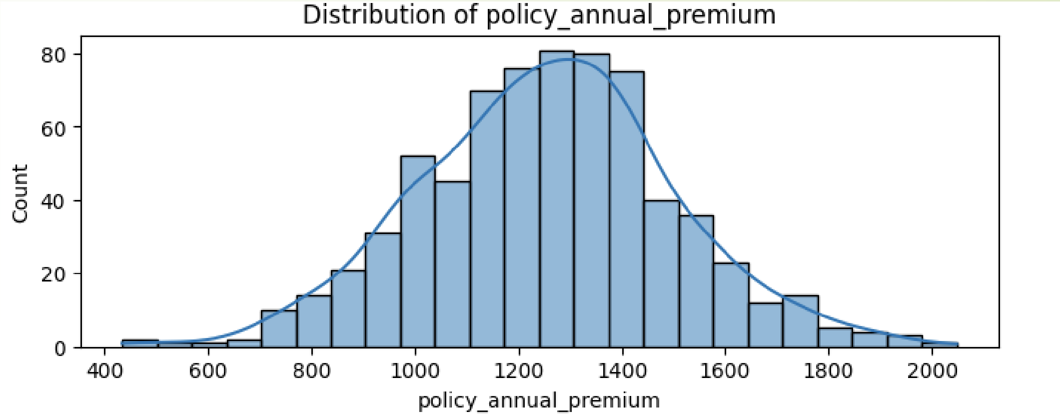
3.3 Split the data [3 Marks]

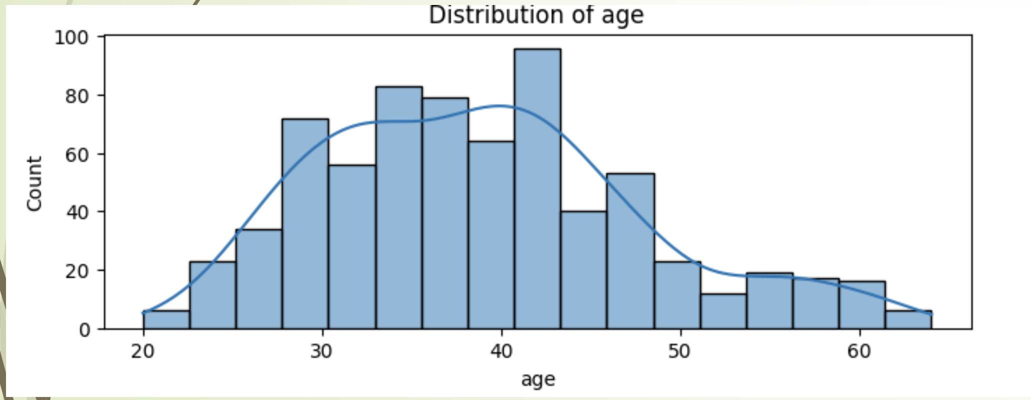
```
# Split the dataset into 70% train and 30% validation and use stratification on the target variable
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.3, stratify=y, random_state=42)

# Reset index for all train and test sets
X_train = X_train.reset_index(drop=True)
X_val = X_val.reset_index(drop=True)
y_train = y_train.reset_index(drop=True)
y_val = y_val.reset_index(drop=True)
```

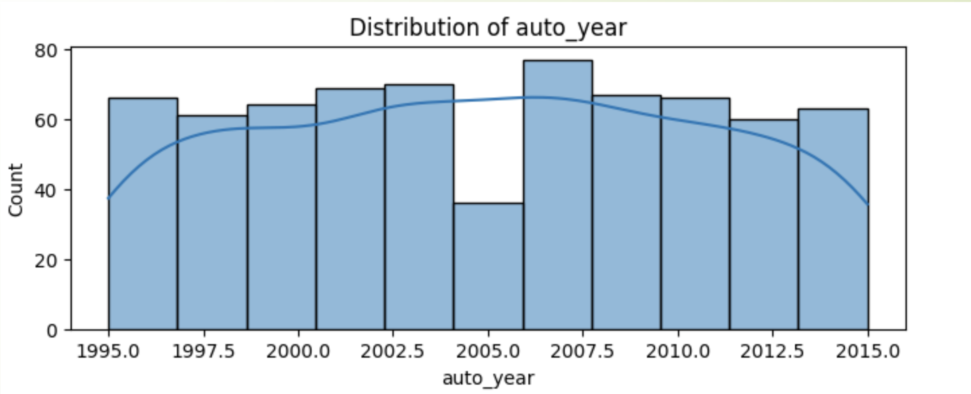
- Train-Validation Split helps to define the target variable & Independent Variables
- 70% of data is used as test and 30% of data is used as validation purpose

Results – EDA on Training Data









Results – EDA on Training Data

4.1 Analysis of Histogram

1. months_as_customer Distribution: near normal
2. age Distribution: Near-normal.
3. policy_deductable Distribution: Discrete with only a few levels (500, 1000, 2000).
4. policy_annual_premium Distribution: Approximately normal.
5. umbrella_limit Distribution: Highly skewed with most values at zero and a few very large outliers.
6. capital-gains Distribution: Right-skewed with many zeros and few large values.
7. capital-loss Distribution: Left-skewed with many zeros, negative values (illogical).
8. incident_hour_of_the_day Distribution: Fairly uniform.
9. number_of_vehicles_involved Distribution: Mostly 1 vehicle.

Results – EDA on Training Data

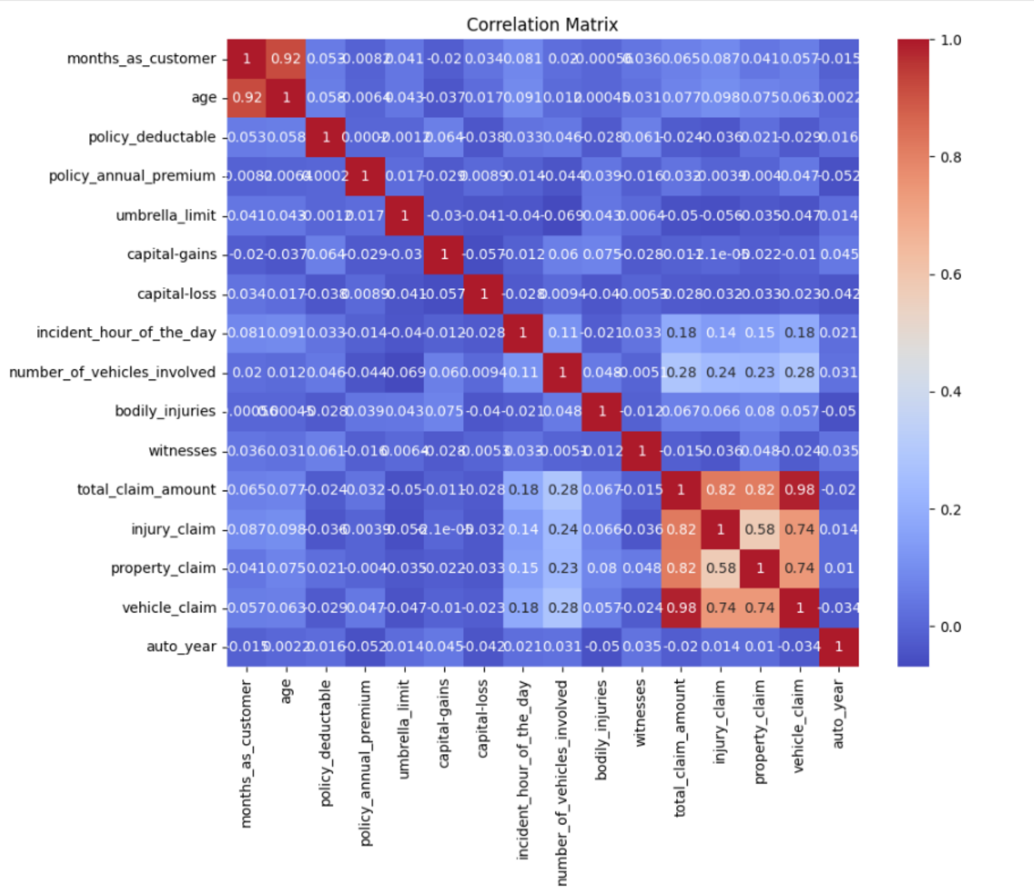
1. bodily_injuries Distribution: Only 0, 1, or 2.
2. witnesses Distribution: 0–3, discrete.
3. total_claim_amount Distribution: Right-skewed.
4. injury_claim Distribution: Right-skewed.
5. property_claim Same as above.
6. vehicle_claim Same as above.
7. auto_year Distribution: Uniform across recent years.

Capital loss Contains invalid negative values we can drop column umbrella_limit dominated by zeros injury_claim/property_claim/vehicle_claim Consider dropping two out of three due to redundancy with total_claim_amount(we will check multicollinearity among these and than decide

Results – EDA on Training Data

4.2 Correlation Analysis

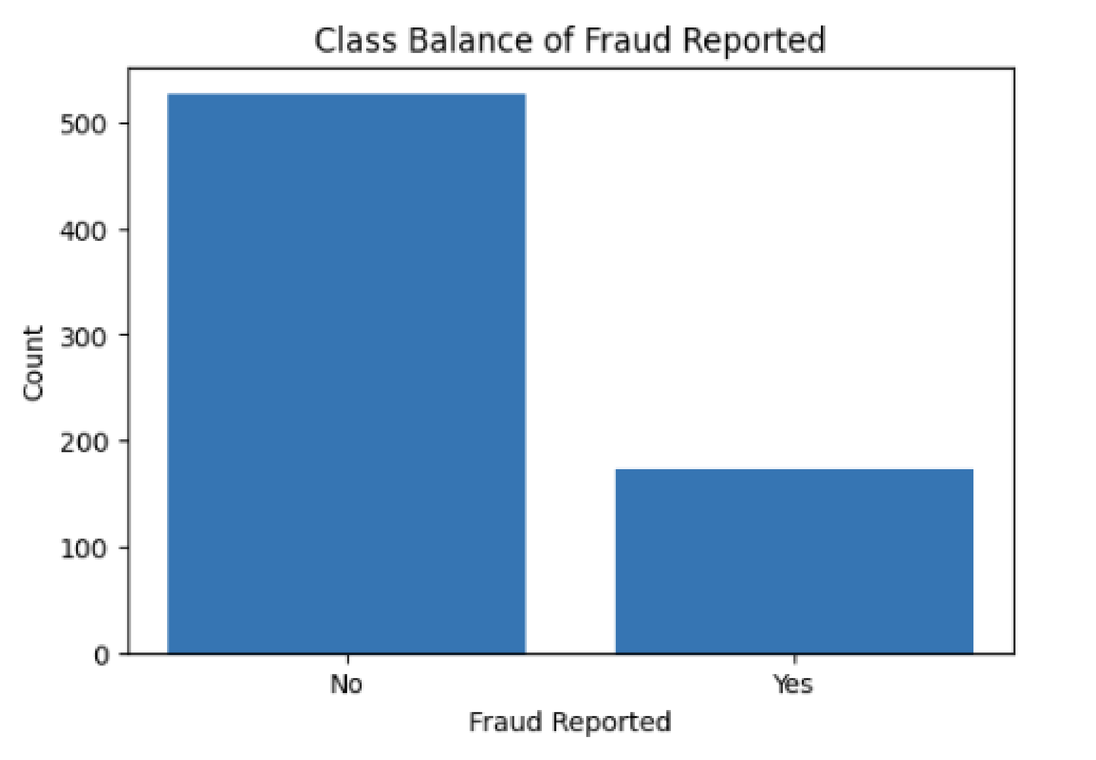
- We can see that there is high correlation between age and months_as_customer.We will drop the "Age" column. Also there is high correlation between total_clam_amount, injury_claim, property_claim, vehicle_claim as total claim is the sum of all others. So we will drop the total claim column.
- **Age and the total claim column shall be removed as are highly correlated with other.**



Results – EDA on Training Data

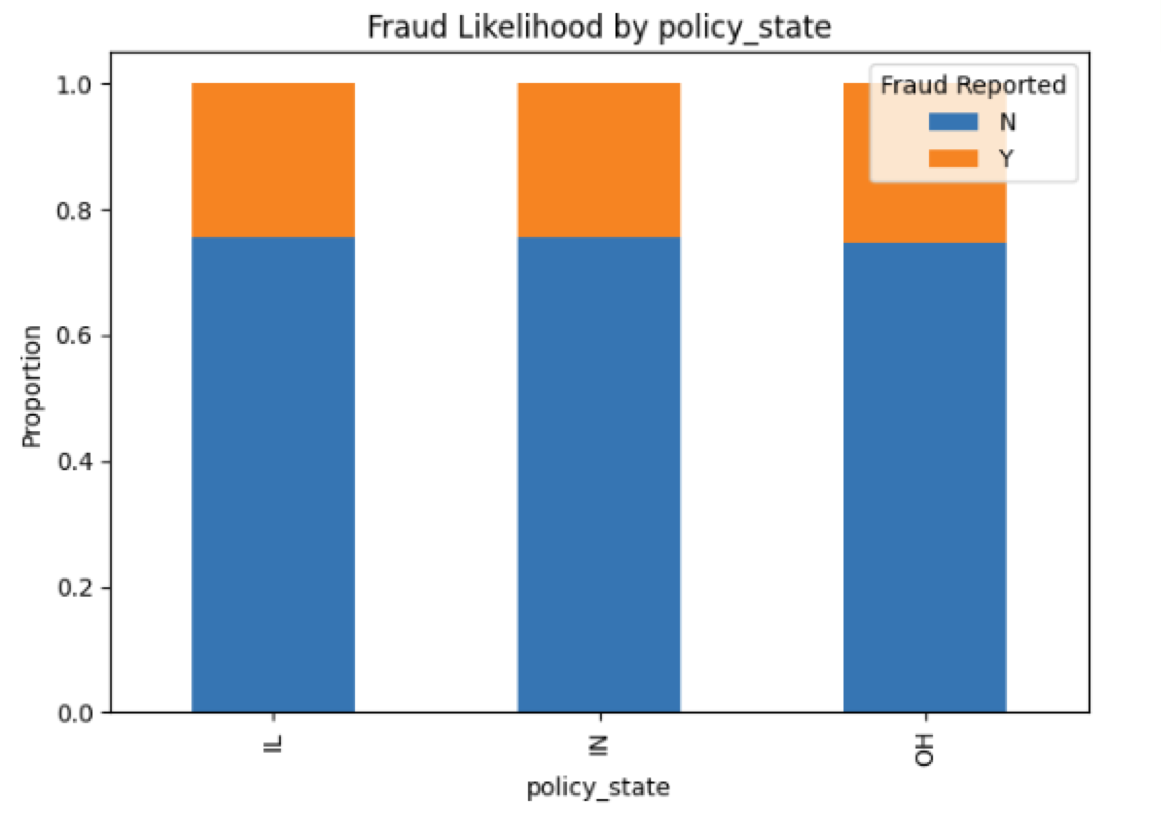
4.3 Check Class Balance

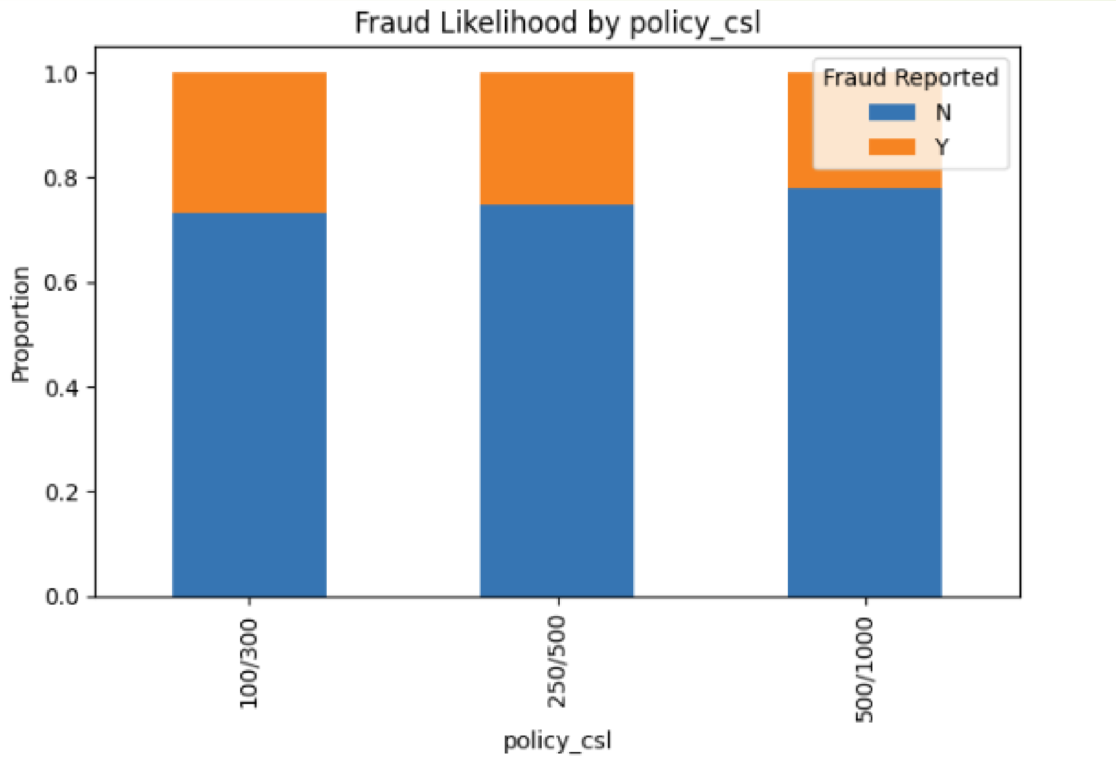
- **Target variable is highly imbalanced.**



Results – EDA on Training Data

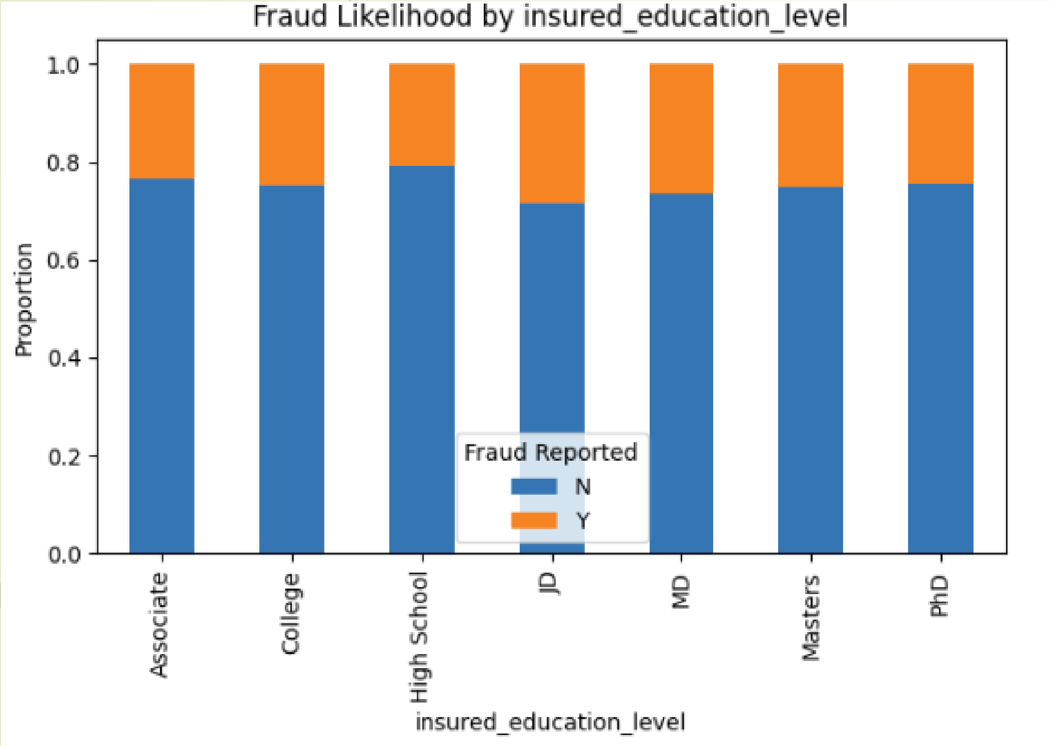
Bivariate Analysis :Target Likelihood

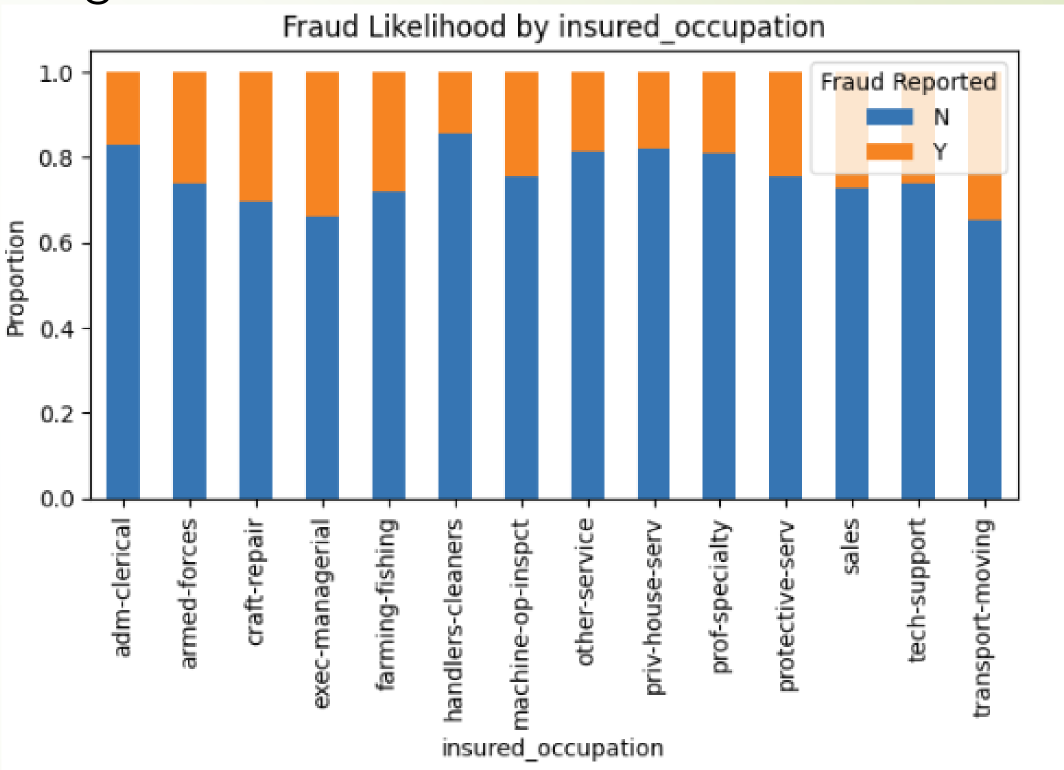




Results – EDA on Training Data

Bivariate Analysis :Target Likelihood





Results – EDA on Training Data

Summary of Target Likelihood Analysis:

- policy_state: Not significant (p-value = 0.9684). Categories with higher-than-baseline fraud rate: [2].
- policy_csl: Not significant (p-value = 0.4788). Categories with higher-than-baseline fraud rate: [0, 1].
- insured_sex: Not significant (p-value = 0.9742). Categories with higher-than-baseline fraud rate: [0].
- insured_education_level: Not significant (p-value = 0.9294). Categories with higher-than-baseline fraud rate: [3, 4, 5].
- insured_occupation: Not significant (p-value = 0.4016). Categories with higher-than-baseline fraud rate: [1, 2, 3, 4, 11, 12, 13].
- insured_hobbies: Significant (p-value = 0.0000). Categories with higher-than-baseline fraud rate: [5, 6, 10, 13, 14, 19].
- insured_relationship: Not significant (p-value = 0.1671). Categories with higher-than-baseline fraud rate: [2, 4, 5].
- incident_type: Significant (p-value = 0.0001). Categories with higher-than-baseline fraud rate: [0, 2].
- collision_type: Not significant (p-value = 0.1963). Categories with higher-than-baseline fraud rate: [0, 2].
- incident_severity: Significant (p-value = 0.0000). Categories with higher-than-baseline fraud rate: [0]. - authorities_contacted: Significant (p-value = 0.0039).

Re_{sults} – EDA on Training Data

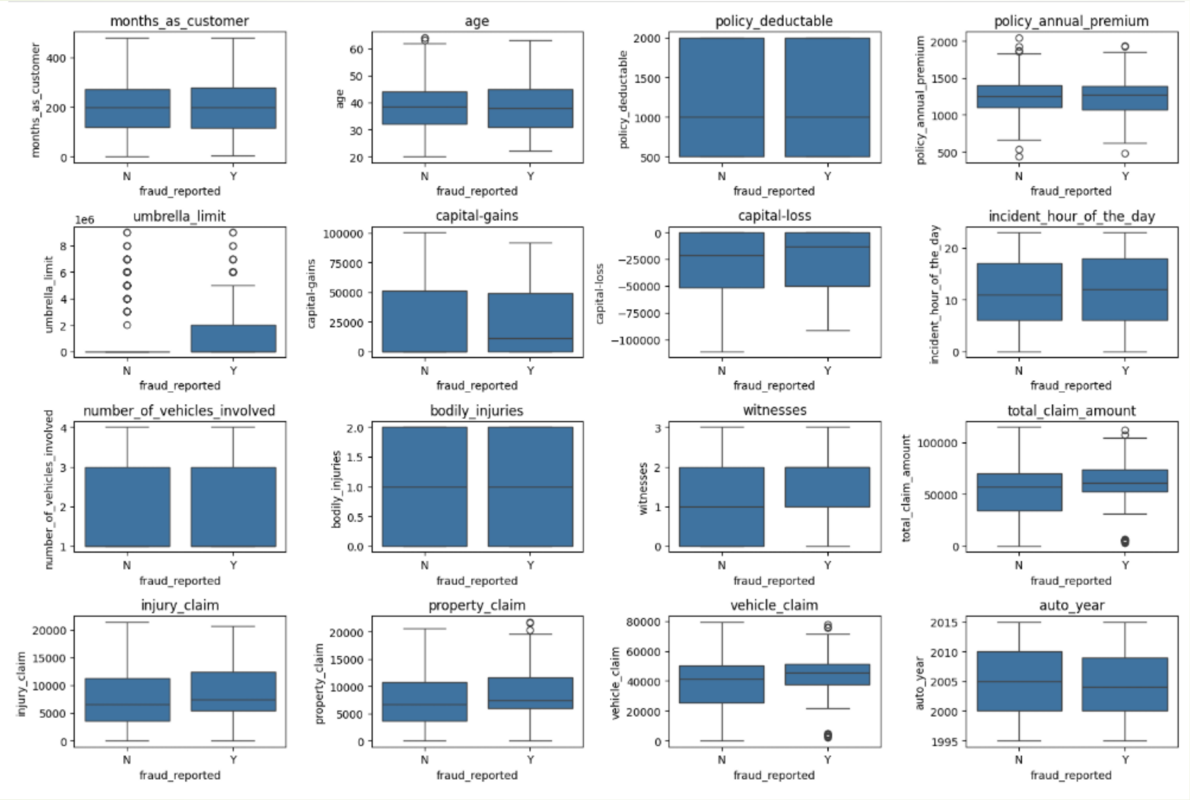
Summary of Target Likelihood Analysis:

- ❖ uthorities_contacted: Significant (p-value = 0.0039). Categories with higher-than-baseline fraud rate: [0, 1, 2].
- ❖ - incident_state: Significant (p-value = 0.0098). Categories with higher-than-baseline fraud rate: [0, 2, 3, 4, 5].
- ❖ - incident_city: Not significant (p-value = 0.5827). Categories with higher-than-baseline fraud rate: [0, 1, 2, 6].
- ❖ - property_damage: Not significant (p-value = 0.2289). Categories with higher-than-baseline fraud rate: [1].
- ❖ - police_report_available: Not significant (p-value = 0.5368). Categories with higher-than-baseline fraud rate: [0].
- ❖ - auto_make: Not significant (p-value = 0.6983). Categories with higher-than-baseline fraud rate: [1, 2, 3, 5, 6, 7, 8, 13].
- ❖ - auto_model: Not significant (p-value = 0.1969). Categories with higher-than-baseline fraud rate: [1, 2, 4, 5, 7, 10, 12, 14, 15, 16, 17, 18, 20, 22, 24, 26, 28, 32, 34, 37, 38].

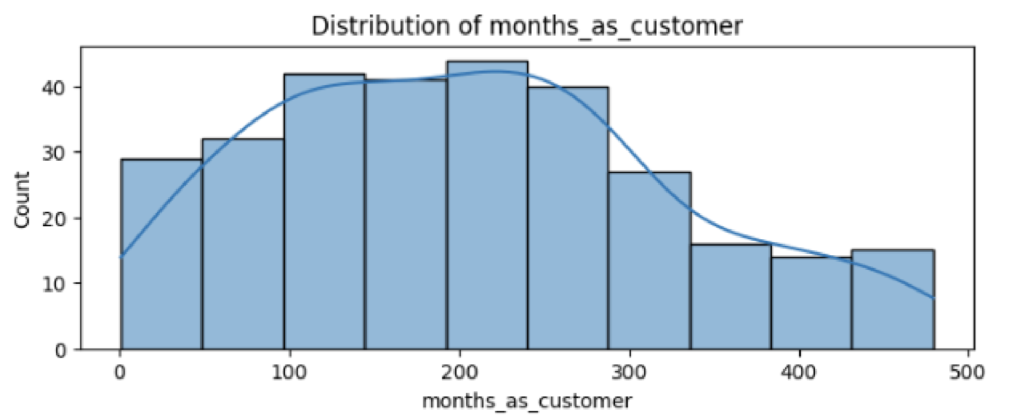
** Target liklihood analysis**
Insured_hobbies,incident_type,incident_severity,authorities_contacted,incident_state found to be significant as per target liklihood analysis.

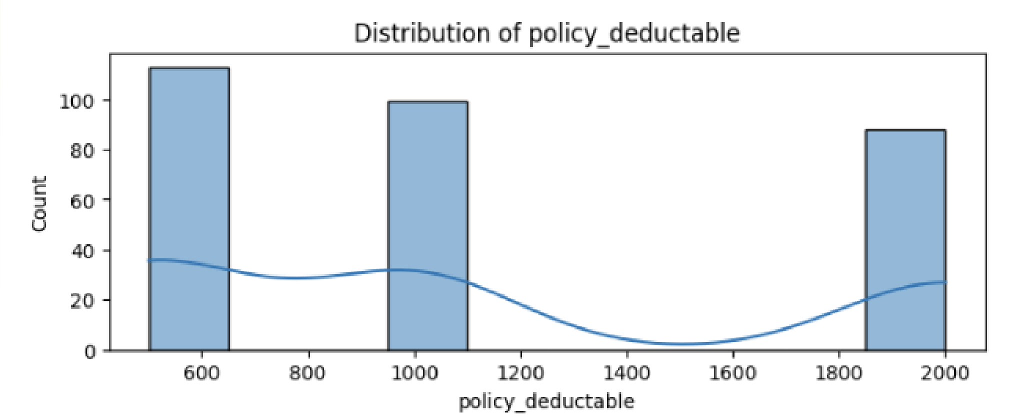
Results – EDA on Training Data

- 4.4.2 Relationship between Numerical Features and the target variable

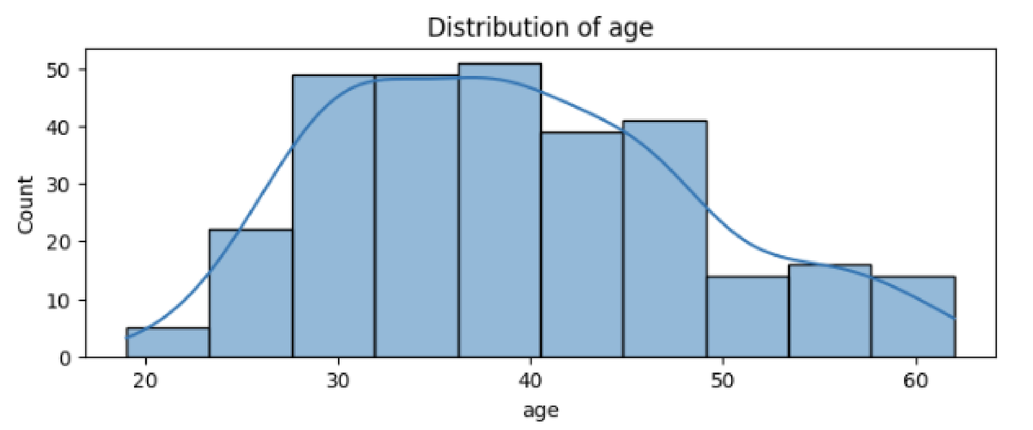


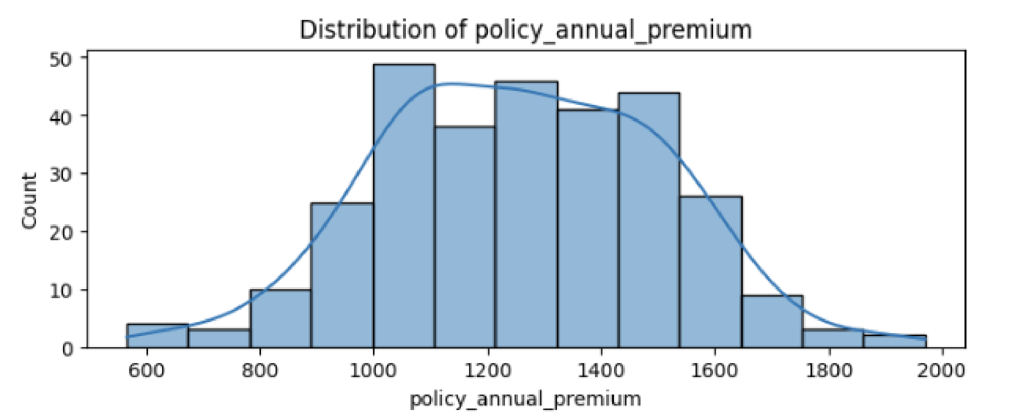
Results – EDA on Validation Data



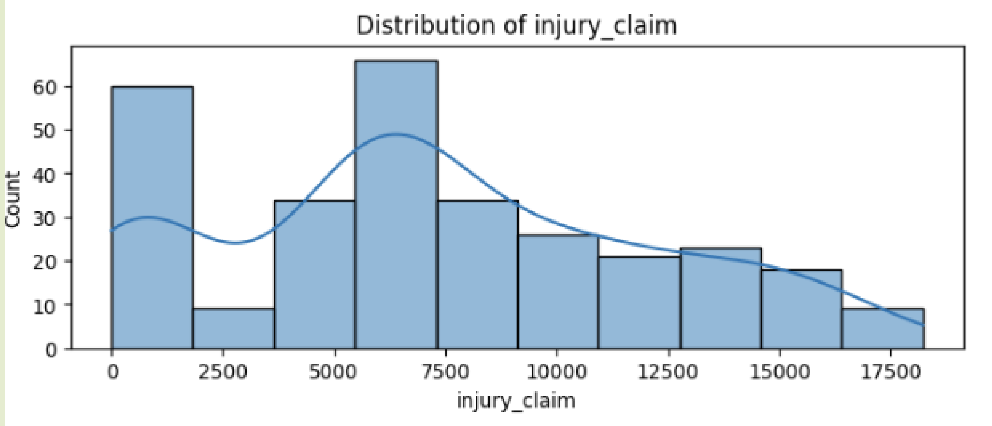


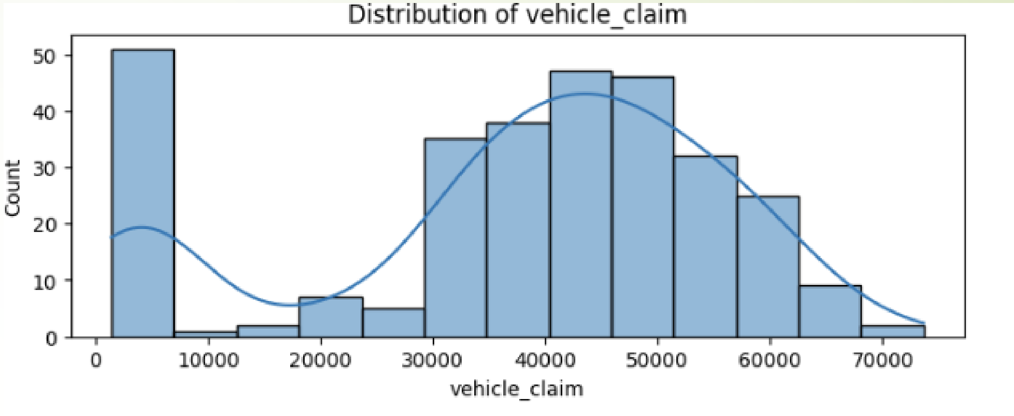
Univariate Analysis



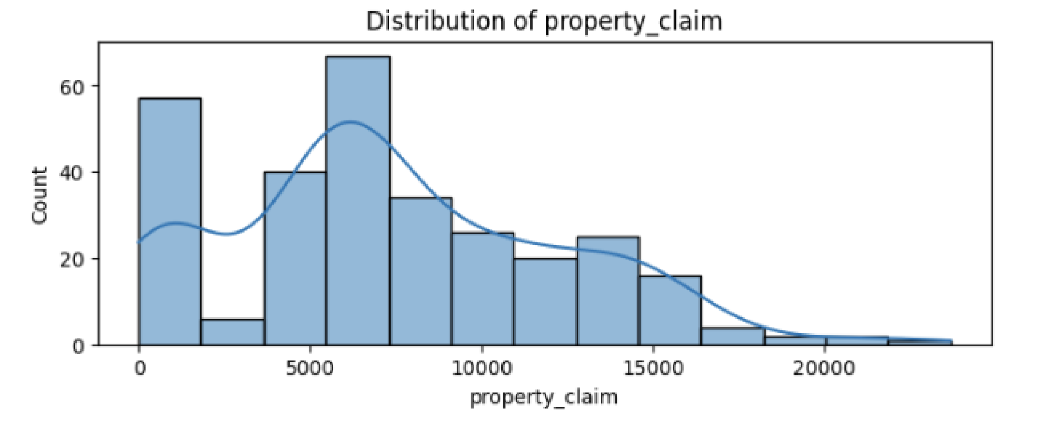


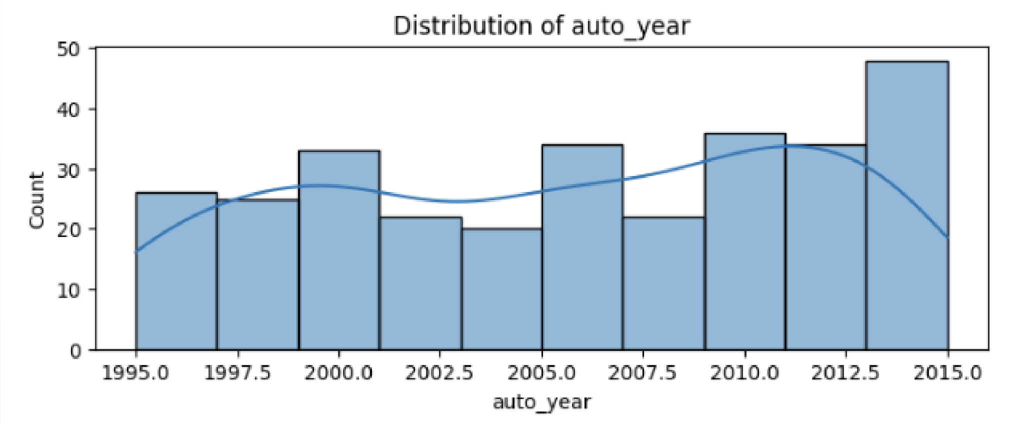
Results – EDA on Validation Data





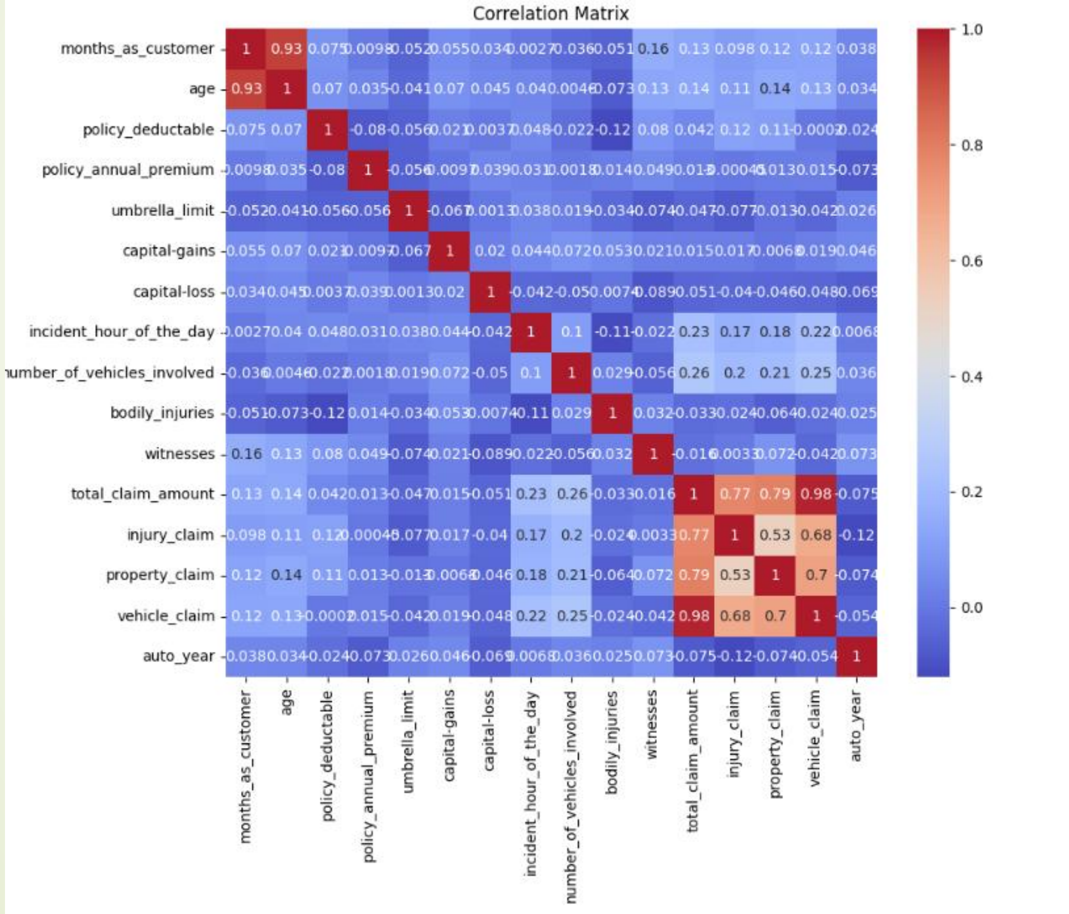
Univariate Analysis





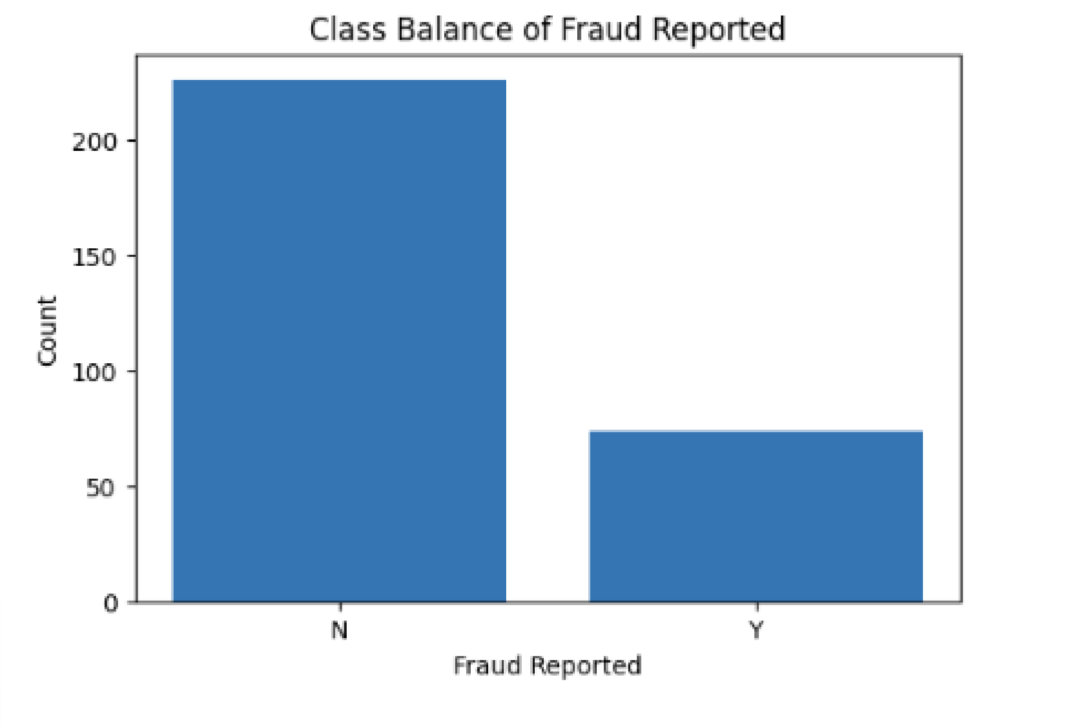
Results – EDA on Validation Data

Correlation Analysis



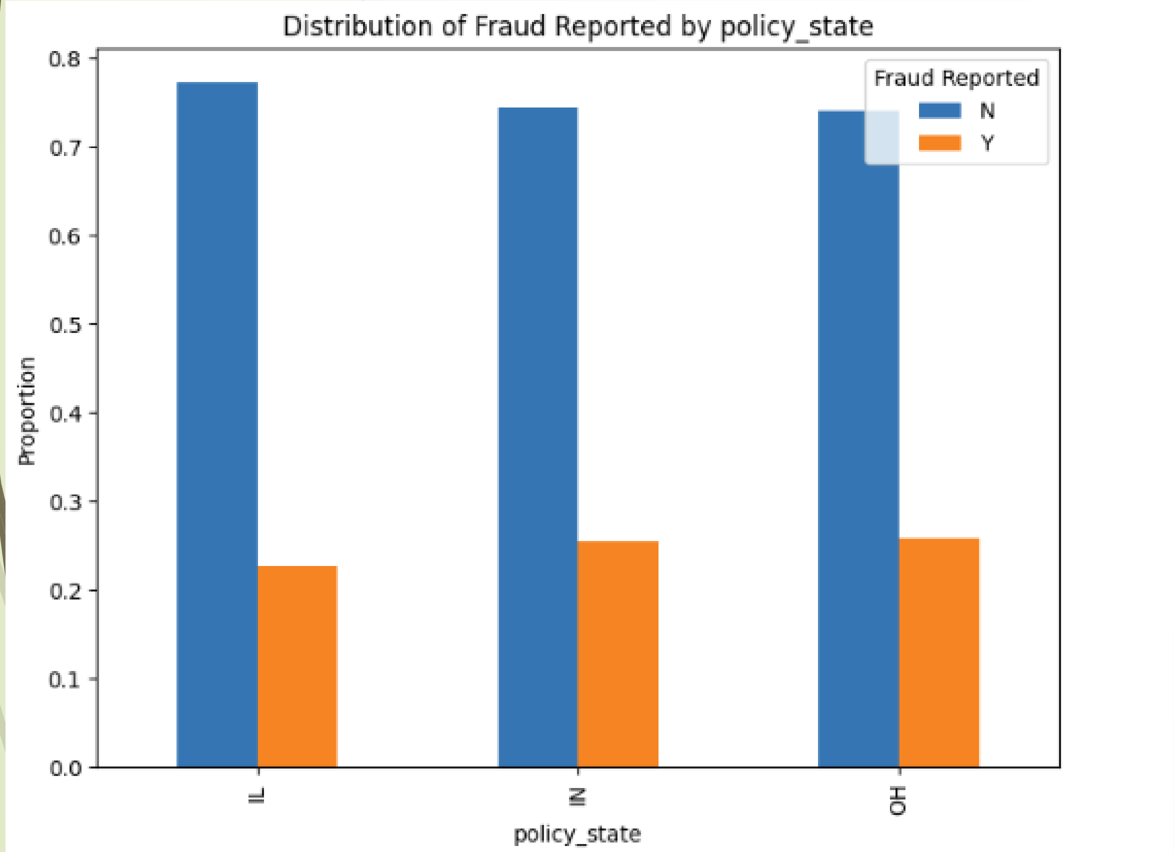
Results – EDA on Validation Data

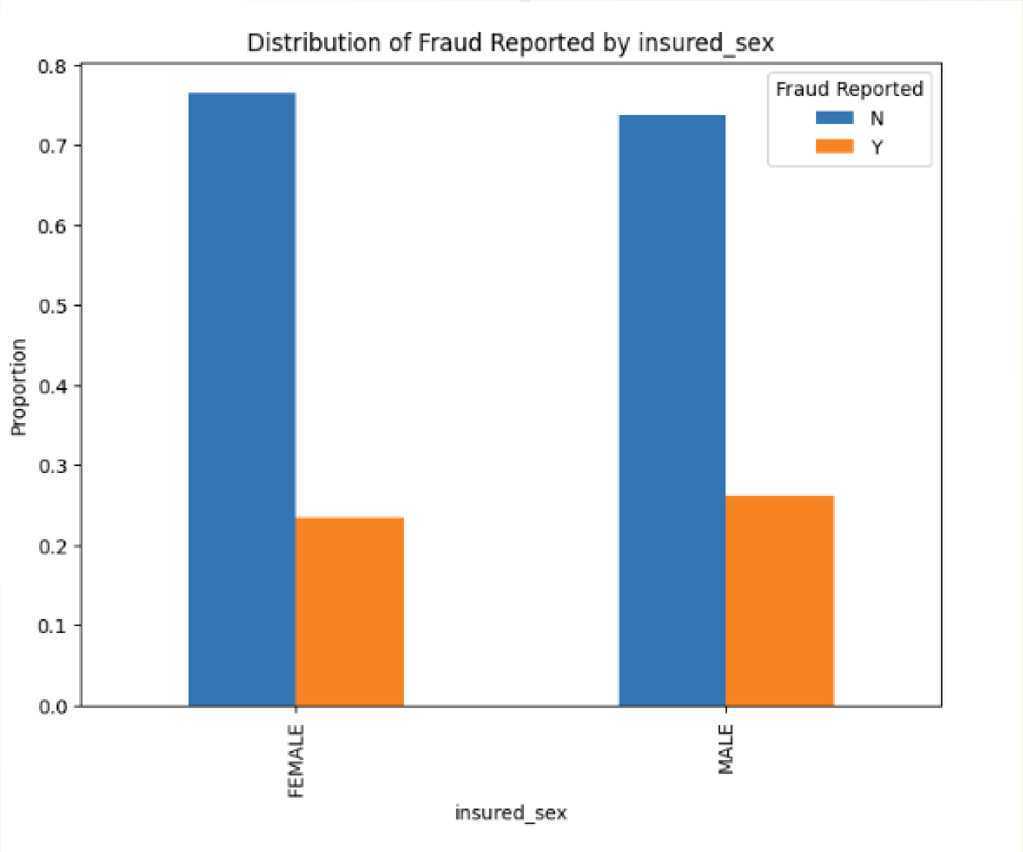
5.3 Check Class Balance



Results – EDA on Validation Data

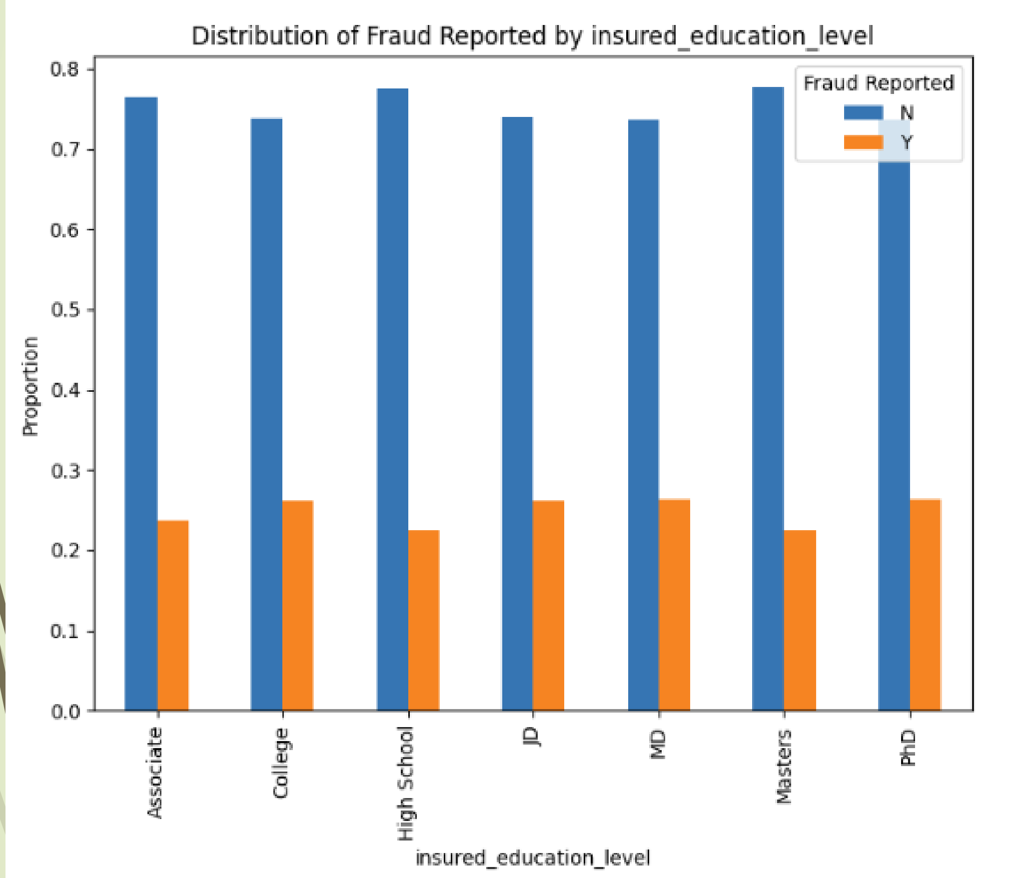
Bivariate Analysis :Target Likelihood





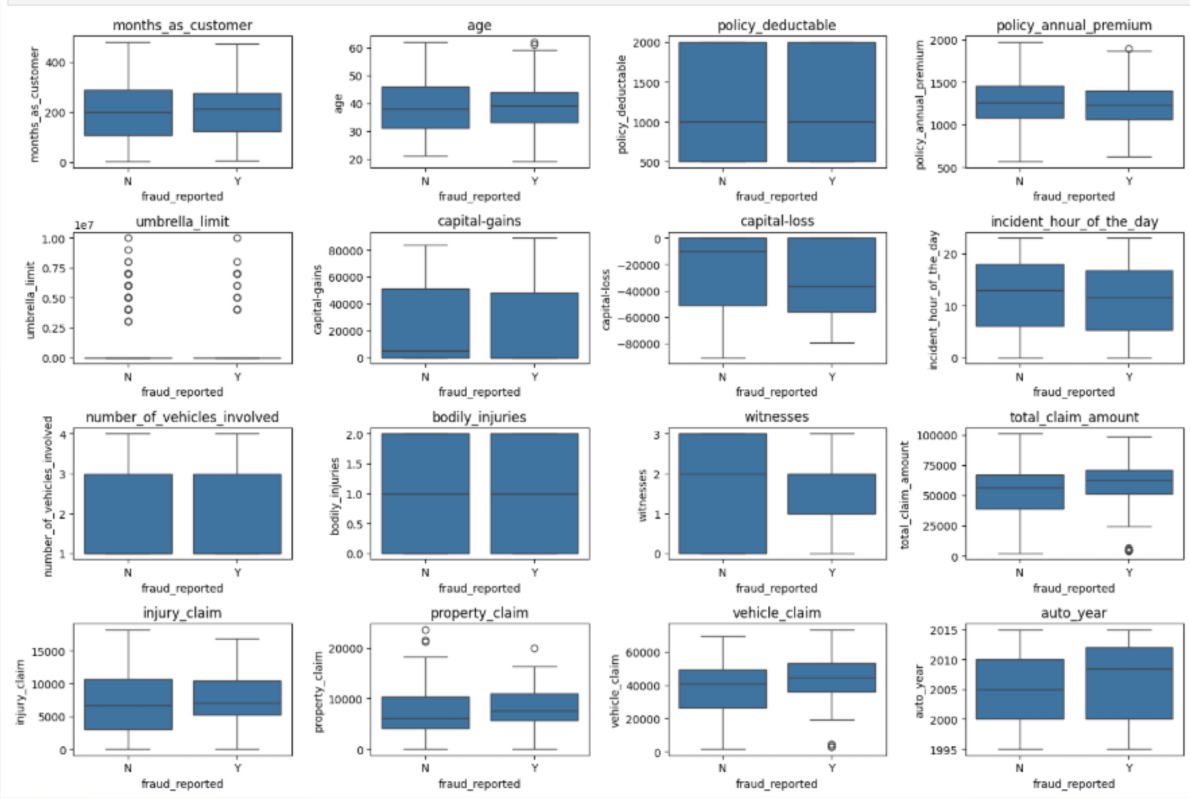
Results – EDA on Validation Data

Bivariate Analysis :Target Likelihood



Results – EDA on Validation Data

❖ 5.4.2 Relationship between Numerical Features and the target variable



Results – Feature Engineering

- 6.1 Resampling : RandomOverSampler

```
[ ]: X_train_resampled.columns

[35]: Index(['months_as_customer', 'age', 'policy_bind_date', 'policy_state',
             'policy_csl', 'policy_deductable', 'policy_annual_premium',
             'umbrella_limit', 'insured_sex', 'insured_education_level',
             'insured_occupation', 'insured_hobbies', 'insured_relationship',
             'capital-gains', 'capital-loss', 'incident_date', 'incident_type',
             'collision_type', 'incident_severity', 'authorities_contacted',
             'incident_state', 'incident_city', 'incident_hour_of_the_day',
             'number_of_vehicles_involved', 'property_damage', 'bodily_injuries',
             'witnesses', 'police_report_available', 'total_claim_amount',
             'injury_claim', 'property_claim', 'vehicle_claim', 'auto_make',
             'auto_model', 'auto_year'],
            dtype='object')
```

Results – Feature Engineering

- 6.3 Handle redundant columns

```
# We had identified 5 redundant columns namely 'policy_number' 'policy_bind_date' 'policy_annual_premium'  'insured_zip'  'incident_location'
#'incident_date' though was not found redundant but after feature enginering 'days_since_policy' it is to be dropped.
#similarly 'Age' is also used to generate 'age_group' thus can be removed.
# Correlation alalysis shows high correlated feature as Age and the total claim column to be dropped.
red_cols=['policy_bind_date', 'policy_annual_premium','incident_date']  #['policy_number',  'insured_zip', 'incident_location'] were already removed
high_cor_col=['age', 'total_claim_amount']
drop_cols =red_cols + high_cor_col
X_train_resampled = X_train_resampled.drop(columns=drop_cols)
X_val = X_val.drop(columns=drop_cols)
```

Results – Feature Engineering

- 6.3 Handle redundant columns

```
Data columns (total 33 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           300 non-null    int64
 1   policy_state                 300 non-null    object
 2   policy_csl                   300 non-null    object
 3   policy_deductable            300 non-null    int64
 4   umbrella_limit               300 non-null    int64
 5   insured_sex                  300 non-null    object
 6   insured_education_level      300 non-null    object
 7   insured_occupation           300 non-null    object
 8   insured_hobbies              300 non-null    object
 9   insured_relationship         300 non-null    object
 10  capital-gains                300 non-null    int64
 11  capital-loss                 300 non-null    int64
 12  incident_type                300 non-null    object
 13  collision_type               300 non-null    object
 14  incident_severity            300 non-null    object
 15  authorities_contacted        300 non-null    object
 16  incident_state               300 non-null    object
 17  incident_city                300 non-null    object
 18  incident_hour_of_the_day     300 non-null    int64
 19  number_of_vehicles_involved  300 non-null    int64
 20  property_damage              300 non-null    object
 21  bodily_injuries              300 non-null    int64
 22  witnesses                    300 non-null    int64
 23  police_report_available      300 non-null    object
 24  injury_claim                 300 non-null    int64
 25  property_claim               300 non-null    int64
 26  vehicle_claim                300 non-null    int64
 27  auto_make                    300 non-null    object
 28  auto_model                   300 non-null    object
 29  auto_year                    300 non-null    int64
 30  days_since_policy            300 non-null    int64
 31  claim_to_premium_ratio       300 non-null    float64
 32  age_group                    300 non-null    category
dtypes: category(1), float64(1), int64(14), object(17)
```

Results – Feature Engineering

❖ 6.6 Feature Scaling

6.6 Feature scaling [3 marks]

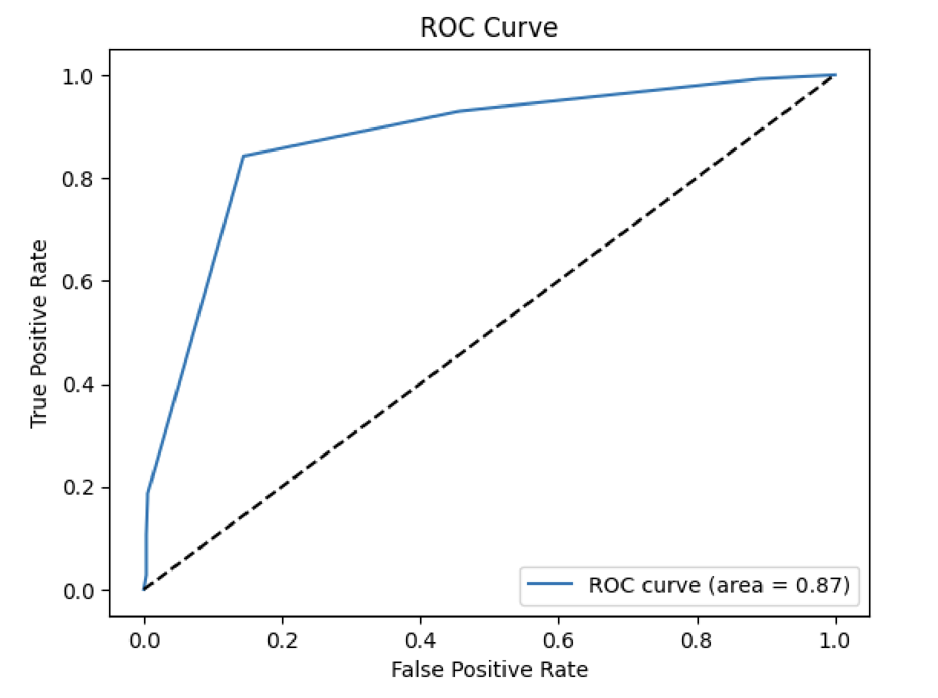
Scale numerical features to a common range to prevent features with larger values from dominating the model. Choose a scaling method appropriate for the data and the chosen model. Apply the same scaling to both training and validation data.

[ ]:

```
# Import the necessary scaling tool from scikit-learn
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

#find numerical cols
numerical_cols = X_train_dummies.select_dtypes(include=['int64', 'float64']).columns

# Scale the numeric features present in the training data
X_train_dummies[numerical_cols] = scaler.fit_transform(X_train_dummies[numerical_cols])

# Scale the numeric features present in the validation data
X_val_dummies[numerical_cols] = scaler.transform(X_val_dummies[numerical_cols])
```

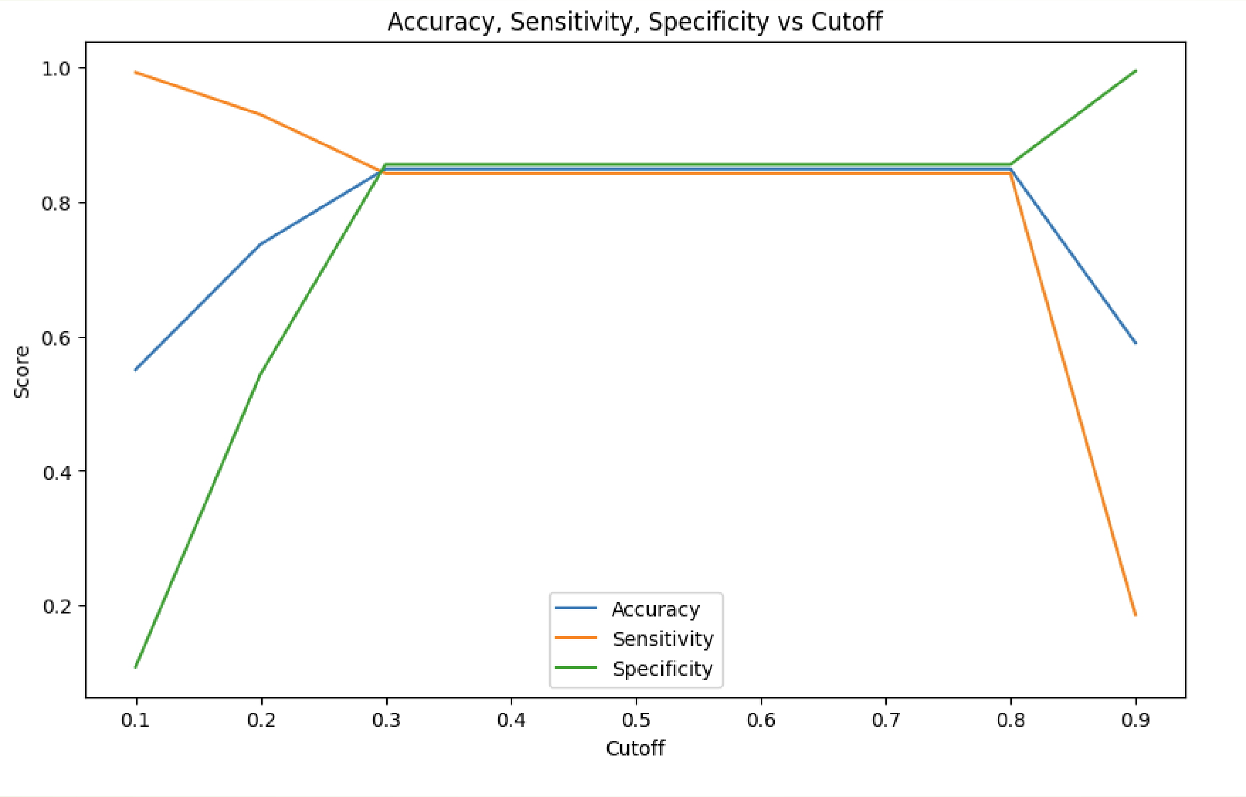
Results – Model Building

- 7.3 Optimal Cut-off and ROC Curve



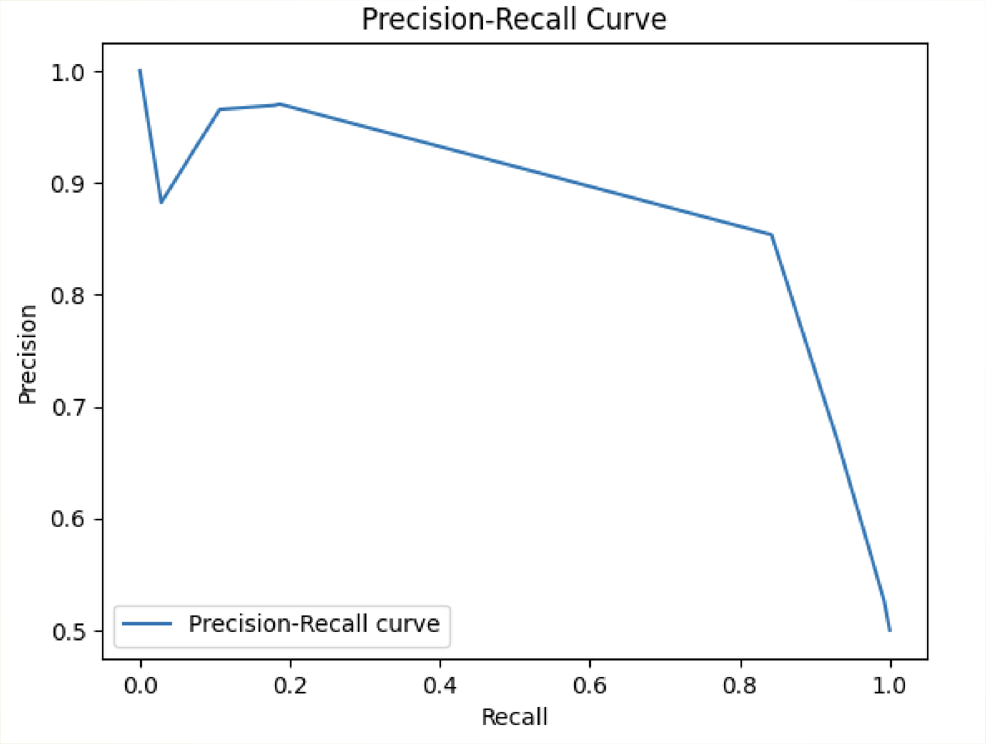
Results – Model Building

- 7.3.3 Accuracy, Sensitivity, Specificity at different values of cutoffs



Results – Model Building

- 7.3.9 Precision-Recall Curve



Results – Hyperparameter Tuning

❖ 7.5.1 Grid Search

7.5.1 Use grid search to find the best hyperparameter values [2 Marks]

```
[ ]: # Use grid search to find the best hyperparamter values
from sklearn.model_selection import GridSearchCV
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [10, 20, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}
grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(X_train_rf, y_train_resampled)


# Best Hyperparameters
print(f"Best Hyperparameters: {grid_search.best_params_}")
```

```
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
```

Results – Predictions and Model Evaluation

- 8.1 Predictions over validation data using logistic regression model

8.1.8 Calculate sensitivity, specificity, precision, recall and f1 score of the model [2 Marks]

```
]: # Calculate the sensitivity

val_sensitivity = TP / (TP + FN)
# Calculate the specificity

val_specificity = TN / (TN + FP)

# Calculate Precision

val_precision = TP / (TP + FP)

# Calculate Recall

val_recall = val_sensitivity

# Calculate F1 Score

val_f1_score = 2 * (val_precision * val_recall) / (val_precision + val_recall)
print(f"Validation Sensitivity: {val_sensitivity}, Specificity: {val_specificity}, Precision: {val_precision}, Recall: {val_recall}, F1-Score: {val_f1_sc
```

```
Validation Sensitivity: 0.6756756756756757, Specificity: 0.8539823008849557, Precision: 0.6024096385542169, Recall: 0.6756756756756757, F1-Score: 0.63694
26751592356
```

Results – Predictions and Model Evaluation

- 8.2 Predictions over validation data using random forest model

8.2.5 Calculate sensitivity, specificity, precision, recall and F1-score of the model [5 Marks]

```
[ ]: # Calculate Sensitivity
val_rf_sensitivity = TP / (TP + FN)

# Calculate Specificity

val_rf_specificity = TN / (TN + FP)

# Calculate Precision


val_rf_precision = TP / (TP + FP)
# Calculate Recall

val_rf_recall = val_rf_sensitivity

# Calculate F1-score

val_rf_f1_score = 2 * (val_rf_precision * val_rf_recall) / (val_rf_precision + val_rf_recall)
print(f"Validation RF Sensitivity: {val_rf_sensitivity}, Specificity: {val_rf_specificity}, Precision: {val_rf_precision}, Recall: {val_rf_recall}, F1-Sc
```

```
Validation RF Sensitivity: 0.527027027027027, Specificity: 0.8893805309734514, Precision: 0.609375, Recall: 0.527027027027027, F1-Score: 0.56521739130434
78
```

Results – Evaluation and Conclusion

- The fraud detection models developed using Logistic Regression and Random Forest provide effective tools for identifying fraudulent insurance claims.
- The Logistic Regression model, after feature selection with RFECV and optimal cutoff tuning, achieved balanced performance with good sensitivity and specificity, making it interpretable and suitable for scenarios where understanding feature impacts is crucial.
- The Random Forest model demonstrated strong predictive power, particularly in handling complex interactions, though it showed signs of overfitting. Both models were evaluated on validation data, with the Logistic Regression generally outperforming Random Forest in accuracy and F1-score, suggesting it may be preferred for deployment.
- Key features like claim amounts, incident severity, and customer tenure were highly predictive of fraud. To improve the fraud detection process, Global Insure should prioritize these features in their screening process, implement the Logistic Regression model for initial claim screening, and continuously monitor model performance to adapt to evolving fraud patterns.

Thanks !