Auto insurance fraud detection using machine learning

Team id:

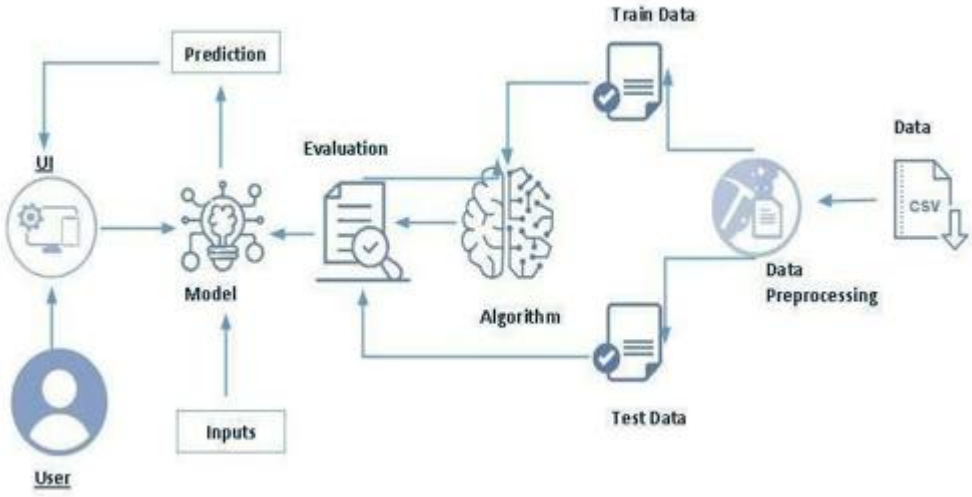
1. Krrish Anand
2. Pakhi Agrawal
3. Vedant Devangkumar Patel

Auto insurance fraud detection using machine learning

Insurance are claimed in order to get a relief amount for any damage cause. Insurance is a means of protection from financial loss, but now-a-days Many people are claiming the Insurance by fraud claims. It can be called as a scam. It is called as fraud claim when a claimant attempts to obtain some benefit or advantage they are not entitled to, or when an insurer knowingly denies some benefit that is due.

These type of Insurance claims cause loss to the company. So, it is necessary to detect the claims which are fraud. The number of cases of insurance fraud that are detected is much lower than the number of acts that are actually committed. So the main purpose of the Insurance Fraud Detection system is to predict the Insurance claim is a Fraud or Legal Claim based on the different appeals and parameters.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
  - Specify the business problem
  - Business requirements
  - Literature Survey
  - Social or Business Impact.
- Data Collection & Preparation
  - Collect the dataset
  - Data Preparation
- Exploratory Data Analysis
  - Descriptive statistical
  - Visual Analysis
- Model Building
  - Training the model in multiple algorithms
  - Testing the model
- Performance Testing & Hyperparameter Tuning
  - Testing model with multiple evaluation metrics
  - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
  - Save the best model
  - Integrate with Web Framework
- Project Demonstration & Documentation
  - Record explanation Video for project end to end solution
  - Project Documentation-Step by step project development procedure

Prior Knowledge:

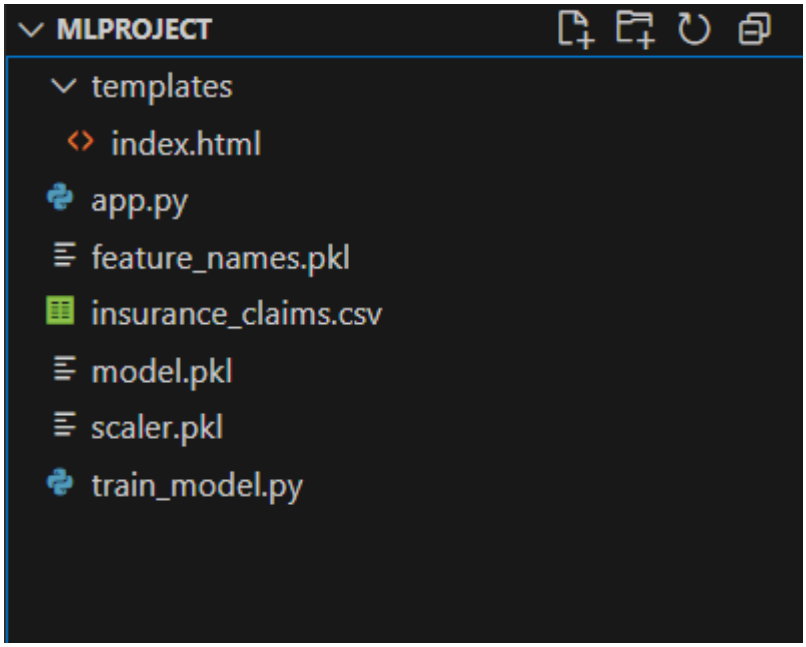
You must have prior knowledge of following topics to complete this project.

ML Concepts

- Supervised learning: https://www.youtube.com/watch?v=QeKshry8pWQ
- Unsupervised learning: https://www.youtube.com/watch?v=D6gtZrsYi6c
  - Decision tree: https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm
  - Random forest: https://www.javatpoint.com/machine-learning-random-forest-algorithm
  - KNN: https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning
  - Xgboost: https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-
  - Evaluation metrics: https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/
  - Performance matrix for problem: https://www.youtube.com/watch?v=D6gtZrsYi6c
  - Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



• the data folder contains the .csv file used for our project

• We are building a flask application that needs the HTML pages to be stored in the templates folder and a python script app.py for scripting.

• model.pkl is our saved model. We will use this model for flask integration.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the Business Problem

Auto insurance fraud is a persistent and costly issue, accounting for billions in losses globally each year. Fraudulent claims can take many forms—including staged accidents, exaggerated damages, or completely false reports—making manual detection challenging, time-consuming, and prone to human error.

Key challenges faced by the industry include:

- **Insurance Companies**: Difficulty in identifying fraudulent claims early, leading to financial losses and increased premiums for honest customers.
- **Claims Adjusters**: Limited capacity to analyze the growing volume of claims data in real time, increasing the likelihood of fraud slipping through undetected.
- **Regulatory Bodies**: A lack of timely data-driven insights to enforce compliance and monitor fraudulent activities effectively.

To address these issues, this project proposes a **machine learning-based fraud detection model** for auto insurance claims. The goal is to:

- **Automatically flag potentially fraudulent claims** for further investigation using predictive analytics.
- **Reduce financial losses** by improving the accuracy and speed of fraud detection.
- **Support insurance analysts** in prioritizing high-risk cases and allocating resources more effectively.

By leveraging historical claims data and advanced algorithms, this model aims to enhance the integrity of the claims process, lower operational costs, and maintain trust in the auto insurance system.

Activity 2: Business Requirements

Project: Auto Insurance Fraud Detection Using Machine Learning

To ensure the successful development and deployment of the fraud detection system, the following business requirements have been identified:

- **Accuracy and Reliability**:
  The system must utilize clean, labeled, and recent auto insurance claims datasets to deliver highly accurate and reliable predictions. The model should minimize both false positives (flagging genuine claims as fraud) and false negatives (missing actual fraud), aligning with industry benchmarks for fraud detection.
- **Scalability and Flexibility**:
  The model should be scalable to handle large volumes of claims data and flexible

enough to adapt to new fraud patterns, evolving schemes, or changing business rules without requiring major reengineering.

- **Regulatory Compliance**:
  The system must comply with data privacy laws and industry standards, such as **GDPR**, **CCPA**, and other insurance regulatory requirements. Sensitive claimant and policyholder data must be handled responsibly and securely.
- **User-Centric Design**:
  The interface should be user-friendly for insurance professionals, fraud investigators, and claims analysts. It should provide clear fraud risk scores, explanations for predictions (e.g., SHAP values), and actionable recommendations to support informed decisions.
- **Integration Capability**:
  The solution should integrate smoothly with existing **Claims Management Systems (CMS)**, **Customer Relationship Management (CRM)** tools, and data analytics platforms. API-based integration should be supported to ensure minimal disruption to current workflows.
- **Performance and Efficiency**:
  The model should deliver predictions quickly and efficiently, enabling near real-time fraud screening at the point of claim submission. This ensures that high-risk claims can be flagged and escalated promptly, reducing fraud-related losses.

Activity 3: Literature Survey

---

1. Viaene, S., et al. (2002)

Citation/Source:
Viaene, S., Derrig, R. A., Waters, M. H., & Dedene, G. (2002). A comparison of state-of-the-art classification techniques for expert automobile insurance fraud detection. *Journal of Risk and Insurance*, 69(3), 373–421.

Purpose of the Study:
To compare various machine learning techniques in detecting expert-level automobile insurance fraud based on historical claims data.

Methodology/Approach:
The study used real-world claims data from an automobile insurer. Various classification models were tested including decision trees, neural networks, and Bayesian learners.

Key Findings:
Decision trees and Bayesian networks showed promising results. However, neural networks delivered higher accuracy for more complex fraud patterns. Ensemble methods also showed potential for improving detection rates.

Relevance:
This study provides foundational insights into the performance of different machine learning

classifiers in insurance fraud detection, highlighting the value of combining models for better fraud identification.

---

2. Bhattacharyya, S., et al. (2011)

Citation/Source:
Bhattacharyya, S., Jha, S., Tharakunnel, K., & Westland, J. C. (2011). Data mining for credit card fraud: A comparative study. *Decision Support Systems*, 50(3), 602–613.

Purpose of the Study:
Though focused on credit card fraud, this study's methodology and findings are highly applicable to insurance fraud detection. It aimed to compare ML classifiers for financial fraud.

Methodology/Approach:
Used a financial transactions dataset and applied logistic regression, decision trees, SVM, neural networks, and random forests. Models were evaluated on precision, recall, and ROC-AUC.

Key Findings:
Random forest and SVM outperformed traditional models. The study stressed the importance of balancing detection accuracy with false positive rates to avoid unnecessary manual investigations.

Relevance:
Highlights the suitability of ensemble methods and support vector machines for detecting fraud in structured transactional datasets similar to those in auto insurance.

---

3. Phua, C., et al. (2004)

Citation/Source:
Phua, C., Lee, V., Smith, K., & Gayler, R. (2004). A comprehensive survey of data mining-based fraud detection research. *arXiv preprint arXiv:1009.6119*

Purpose of the Study:
To provide a comprehensive overview of data mining methods used in fraud detection across various industries.

Methodology/Approach:
Reviewed over 150 research papers related to fraud detection in insurance, banking, telecommunications, and e-commerce. Techniques discussed include outlier detection, clustering, rule-based systems, and supervised learning.

Key Findings:
No single technique is universally superior. Combining models (hybrid approaches) and incorporating domain-specific rules yields better fraud detection performance.

Relevance:
Gives valuable guidance on designing hybrid fraud detection systems that blend machine learning with business logic—an approach well-suited to auto insurance claims.

---

4. Carcillo, F., et al. (2018)

Citation/Source:
Carcillo, F., Le Borgne, Y. A., Caelen, O., & Bontempi, G. (2018). Leveraging unsupervised anomaly detection techniques for insurance fraud detection. *Future Generation Computer Systems*, 86, 544–555.

Purpose of the Study:
To investigate the application of unsupervised learning and anomaly detection methods in identifying novel or rare insurance fraud patterns.

Methodology/Approach:
Used real insurance claim datasets and applied isolation forest, autoencoders, and one-class SVM for fraud detection without labeled data.

Key Findings:
Isolation forest and deep autoencoders showed strong capability in detecting previously unseen fraud patterns. These methods are effective in environments with minimal labeled data.

Relevance:
Useful for detecting emerging or rare types of fraud where labeled examples are not available. Highlights the need for combining supervised and unsupervised approaches in fraud detection systems.

---

5. Bauder, R. A., & Khoshgoftaar, T. M. (2018)

Citation/Source:
Bauder, R. A., & Khoshgoftaar, T. M. (2018). A survey of medicare fraud detection using data mining techniques. *Health Information Science and Systems*, 6(1), 1–10.

Purpose of the Study:
While focused on Medicare, this study explores data mining methods relevant to detecting fraudulent billing—similar in structure to auto claims fraud.

Methodology/Approach:
Reviewed classification methods, clustering, and hybrid models across Medicare datasets.

Key Findings:
Ensemble methods such as random forest and boosting algorithms showed higher detection rates. Feature engineering and domain knowledge were critical for model success.

Relevance:
Reinforces that high-performing fraud detection relies not just on model choice but on carefully engineered input features. Applies directly to auto insurance claims data processing.

Activity 4: Social or Business Impact

Social Impact:

The application of machine learning—especially decision tree-based models—for auto insurance fraud detection has a meaningful social impact. By reducing fraudulent claims, the system helps ensure that resources are allocated fairly, premiums remain affordable, and honest policyholders are not penalized. Accurate fraud detection:

- Promotes **trust and transparency** between insurers and customers.
- **Protects consumers** from unfair premium increases due to widespread fraud.
- Enhances the **efficiency of claims processing**, leading to quicker resolutions for legitimate claims.
- Contributes to a **healthier insurance ecosystem**, which in turn supports broader financial stability in society.

This system can also discourage fraudulent behavior through improved detection rates, thereby **upholding ethical standards** and promoting integrity within communities.

Business Model Impact:

From a business standpoint, integrating machine learning into fraud detection provides several key benefits for insurance companies and associated service providers:

- **Cost Savings**: Automated fraud detection significantly reduces the costs associated with manual investigation and payouts on fraudulent claims.
- **Operational Efficiency**: Machine learning models—such as decision trees and ensemble methods—enable real-time screening of claims, accelerating workflows and freeing up human resources for more complex cases.
- **Scalability and Adaptability**: The system can be scaled to handle increasing data volumes and adapted to detect new fraud patterns as they emerge.
- **Competitive Advantage**: Insurers leveraging advanced analytics can offer **lower premiums** and faster claim processing, improving customer satisfaction and market competitiveness.
- **Product Development**: The model can be offered as a service (e.g., “fraud detection as a service”) to smaller insurance firms, third-party administrators, or reinsurers.

Ultimately, this technology helps insurers reduce financial risk, improve service quality, and create **data-driven, fraud-resilient** business models in a highly competitive industry.

Milestone 2: Data Collection & Preparation

Machine learning relies heavily on the **quality and relevance of data**. A well-prepared dataset is the foundation for training reliable and accurate models. In this section, we focus on acquiring and beginning to explore the dataset needed for fraud detection in auto insurance claims.

Activity 1: Collect the Dataset

For this project, we are using a **.CSV dataset** available from a trusted open-source platform. Open datasets help standardize experimentation and improve reproducibility.

- **Source**: Kaggle - Auto Insurance Claims Data
- **Dataset Format**: CSV (Comma-Separated Values)
- **Purpose**: To train a machine learning model that detects whether an insurance claim is fraudulent or not.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image.

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
from imblearn.over_sampling import SMOTE
import joblib
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

●For checking the null values, df.isna().any( ) function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
df = pd.read_csv("insurance_claims.csv")
```

```
df.head()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	...	police_report_availabl
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0	466132	...	YE
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	...	N
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000	608117	...	N
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000	610706	...	N

5 rows × 40 columns

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

```
df.isnull().sum()
```

	0
months_as_customer	0
age	0
policy_number	0
policy_bind_date	0
policy_state	0
policy_csl	0
policy_deductable	0
policy_annual_premium	0
umbrella_limit	0
insured_zip	0
insured_sex	0
insured_education_level	0
insured_occupation	0
insured_hobbies	0
insured_relationship	0
capital-gains	0
capital-loss	0
incident_date	0
incident_type	0
collision_type	0
incident_severity	0
authorities_contacted	91
incident_state	0
incident_city	0

```
df.nunique()
```

	0
months_as_customer	391
age	46
policy_number	1000
policy_bind_date	951
policy_state	3
policy_csl	3
policy_deductable	3
policy_annual_premium	991
umbrella_limit	11
insured_zip	995
insured_sex	2
insured_education_level	7
insured_occupation	14
insured_hobbies	20

```
[ ] df['authorities_contacted'].unique()
```

```
array(['Police', nan, 'Fire', 'Other', 'Ambulance'], dtype=object)
```

```
df['authorities_contacted'] = df['authorities_contacted'].fillna('Unknown')
df.isna().any()
```

	0
months_as_customer	False
age	False
policy_number	False
policy_bind_date	False
policy_state	False
policy_csl	False
policy_deductable	False
policy_annual_premium	False

```
df.drop(columns=['_c39'], inplace=True)
df.drop(columns=['incident_location'], inplace=True)
df.drop(columns=['policy_number'], inplace=True)
df.drop(columns=['insured_zip'], inplace=True)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           1000 non-null   int64  
 1   age                          1000 non-null   int64  
 2   policy_bind_date             1000 non-null   object 
 3   policy_state                 1000 non-null   object 
 4   policy_csl                   1000 non-null   object 
 5   policy_deductable            1000 non-null   int64  
 6   policy_annual_premium        1000 non-null   float64
 7   umbrella_limit               1000 non-null   int64  
 8   insured_sex                  1000 non-null   object 
 9   insured_education_level      1000 non-null   object 
 10  insured_occupation           1000 non-null   object 
 11  insured_hobbies              1000 non-null   object 
 12  insured_relationship         1000 non-null   object 
 13  capital-gains                1000 non-null   int64  
 14  capital-loss                 1000 non-null   int64  
 15  incident_date                1000 non-null   object 
 16  incident_type                1000 non-null   object
```

```
non_numeric_cols = df.select_dtypes(exclude=['number']).columns
print(non_numeric_cols)
```

```
Index(['policy_bind_date', 'policy_state', 'policy_csl', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       'insured_relationship', 'incident_date', 'incident_type',
       'collision_type', 'incident_severity', 'authorities_contacted',
       'incident_state', 'incident_city', 'property_damage',
       'police_report_available', 'auto_make', 'auto_model', 'fraud_reported'],
      dtype='object')
```

```
[ ] df[(df['total_claim_amount'] < 0) | (df['vehicle_claim'] < 0)]
```

months_as_customer	age	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_pre

0 rows × 36 columns

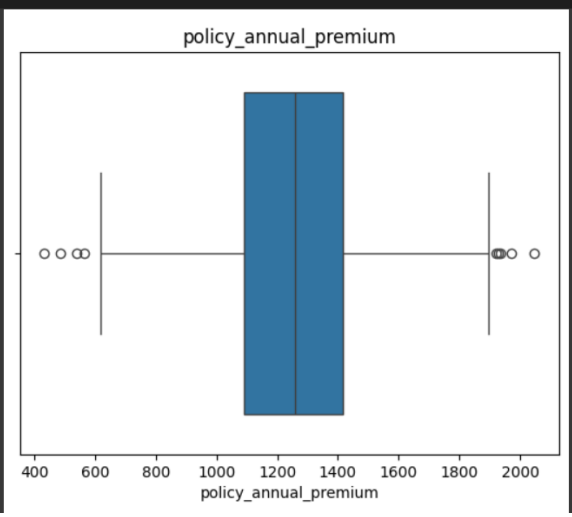
Activity 2.2: Handling Outliers

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of Age feature with some mathematical formula.

·To find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.

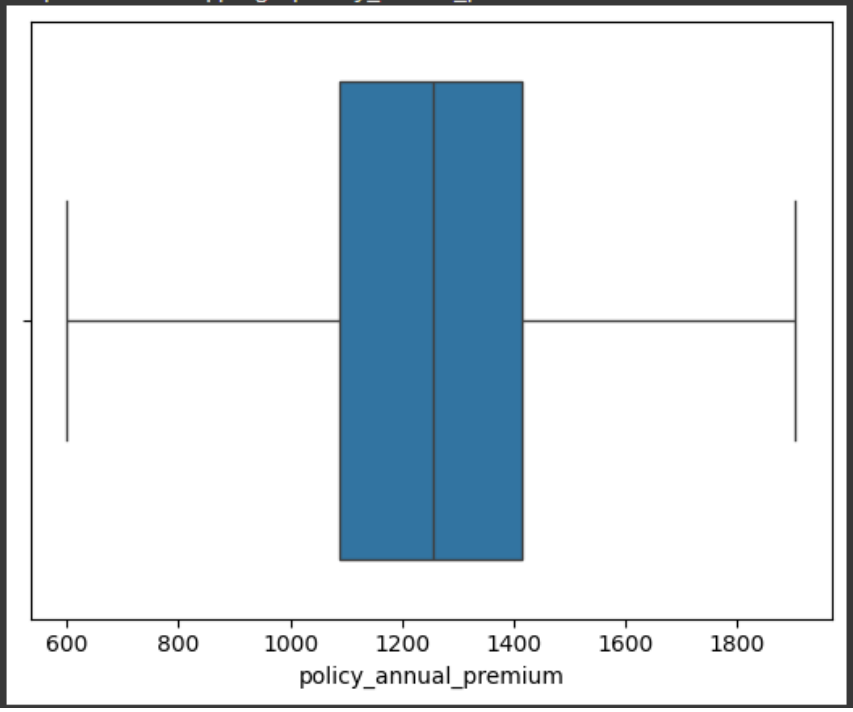
·To handle the outliers transformation technique is used. Here log transformation is used. We have created a function to visualize the distribution and probability plot of Age feature.

```
for col in df.select_dtypes(include=['int64', 'float64']).columns:
    sns.boxplot(data=df, x=col)
    plt.title(col)
    plt.show()
```



```python
# Columns for outlier handling
num_features = ['age', 'policy_annual_premium', 'umbrella_limit', 'total_claim_amount', 'property_claim']

# Step 1: Calculate IQR for each selected feature
IQR = []
for col in num_features:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR.append(Q3 - Q1)

# Step 2: Compute upper and lower bounds
upper = []
lower = []
for i, col in enumerate(num_features):
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    upper.append(Q3 + 1.5 * IQR[i])
    lower.append(Q1 - 1.5 * IQR[i])

# Step 3: Cap the outliers
df['age'] = np.where(df['age'] > upper[0], upper[0],
             np.where(df['age'] < lower[0], lower[0], df['age']))

df['policy_annual_premium'] = np.where(df['policy_annual_premium'] > upper[1], upper[1],
                            np.where(df['policy_annual_premium'] < lower[1], lower[1], df['policy_annual_premium']))

df['umbrella_limit'] = np.where(df['umbrella_limit'] > upper[2], upper[2],
                       np.where(df['umbrella_limit'] < lower[2], lower[2], df['umbrella_limit']))

df['total_claim_amount'] = np.where(df['total_claim_amount'] > upper[3], upper[3],
                           np.where(df['total_claim_amount'] < lower[3], lower[3], df['total_claim_amount']))

df['property_claim'] = np.where(df['property_claim'] > upper[4], upper[4],
                       np.where(df['property_claim'] < lower[4], lower[4], df['property_claim']))

# Step 4: Visualize the boxplots after capping
for col in num_features:
    print(f"Boxplot after capping: {col}")
    sns.boxplot(data=df, x=col)
    plt.show()
```

```
Boxplot after capping: policy_annual_premium
```



Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

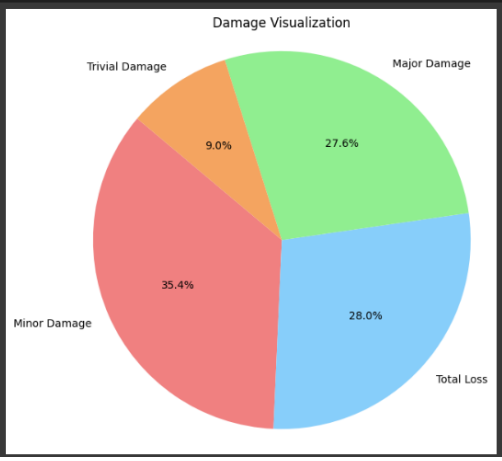
```
[ ] df.describe()
```

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved	bodily_injuries	witnesses	total_claim_amount	in
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.0	1000.000000	1000.000000	1000.000000	1000.00000	1000.000000	1000.000000	1000.000000	
mean	203.954000	38.942000	1136.000000	1256.506126	0.0	25126.100000	-26793.700000	11.644000	1.83900	0.992000	1.487000	52760.782500	
std	115.113174	9.124576	611.864673	242.246335	0.0	27872.187708	28104.096686	6.951373	1.01888	0.820127	1.111335	26398.830555	
min	0.000000	19.000000	500.000000	600.476250	0.0	0.000000	-111100.000000	0.000000	1.00000	0.000000	0.000000	100.000000	
25%	115.750000	32.000000	500.000000	1089.607500	0.0	0.000000	-51500.000000	6.000000	1.00000	0.000000	1.000000	41812.500000	
50%	199.500000	38.000000	1000.000000	1257.200000	0.0	0.000000	-23250.000000	12.000000	1.00000	1.000000	1.000000	58055.000000	
75%	276.250000	44.000000	2000.000000	1415.695000	0.0	51025.000000	0.000000	17.000000	3.00000	2.000000	2.000000	70592.500000	1
max	479.000000	62.000000	2000.000000	1904.826250	0.0	100500.000000	0.000000	23.000000	4.00000	2.000000	3.000000	113762.500000	2

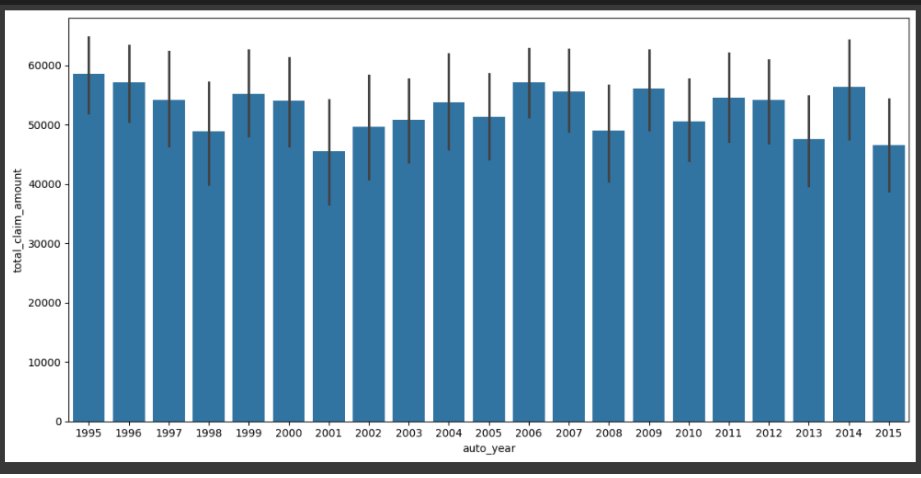
Activity 2: Visual analysis

Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

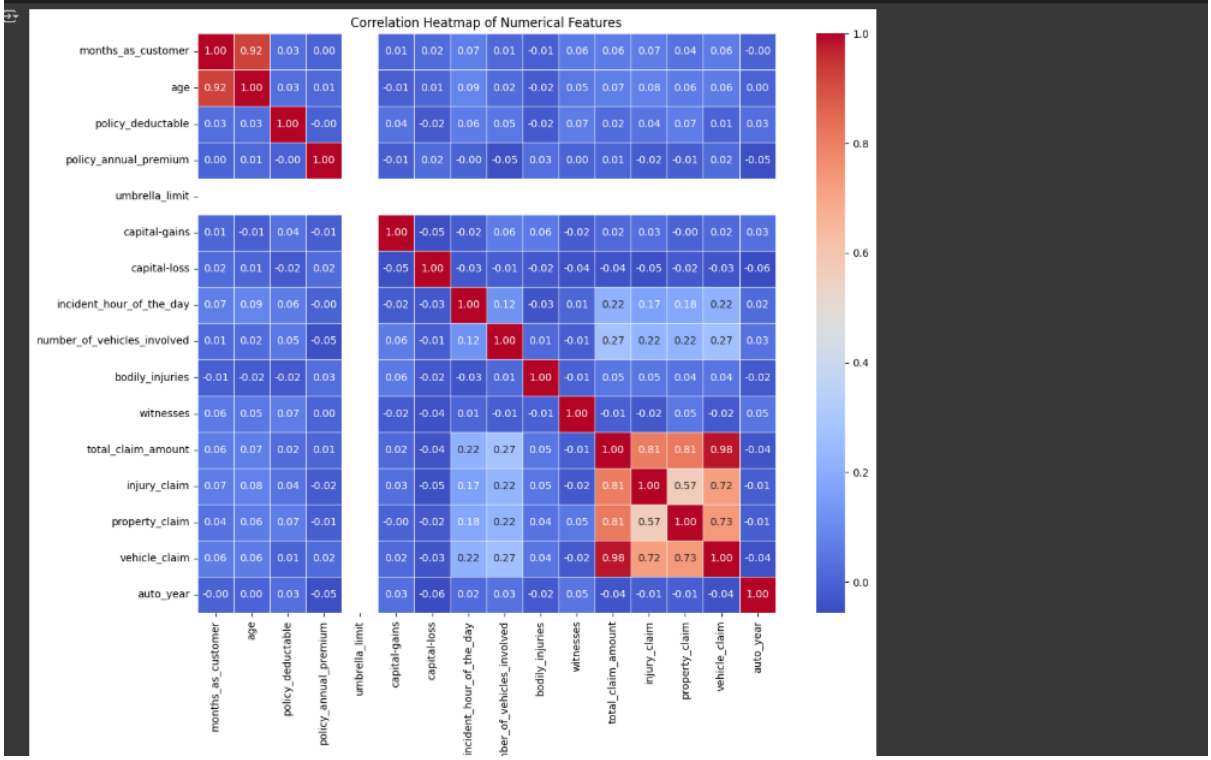
```
severity_counts = df['incident_severity'].value_counts()

plt.figure(figsize=(7, 7))
plt.pie(severity_counts,
        labels=severity_counts.index,
        autopct='%1.1f%%',
        startangle=140,
        colors=['lightcoral', 'lightskyblue', 'lightgreen', 'sandybrown'])

plt.title("Damage Visualization")
plt.axis('equal')
plt.show()
```




```
plt.figure(figsize=(12, 6))
sns.barplot(x = 'auto_year', y = 'total_claim_amount', data = df)
plt.tight_layout()
plt.show()
```



```python
numeric_df = df.select_dtypes(include='number')

# Compute correlation matrix
correlation_matrix = numeric_df.corr()

# Plot heatmap
plt.figure(figsize=(14, 10))
sns.heatmap(correlation_matrix,
            annot=True,
            fmt=".2f",
            cmap='coolwarm',
            linewidths=0.5,
            square=True)

plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```



Encoding the Categorical Features:

• The categorical Features are can't be passed directly to the Machine Learning Model. So we convert them into Numerical data based on their order. This Technique is called Encoding.

• Here we are importing Label Encoder from the Sklearn Library.

• Here we are applying fit_transform to transform the categorical features to numerical features.

```
categorical_columns = ['insured_sex', 'insured_education_level', 'incident_type',
                       'collision_type', 'incident_severity', 'authorities_contacted',
                       'incident_state', 'incident_city', 'auto_make', 'auto_model',
                       'policy_bind_date', 'incident_date', 'insured_occupation','incident_location',
                       'insured_hobbies', 'umbrella_limit', 'property_damage', 'police_report_available',
                       'auto_year', 'witnesses', 'fraud_reported', 'policy_state', 'policy_csl', 'insured_relationship']

le = LabelEncoder()
for col in categorical_columns:
    if col in df.columns: # Added this check for robustness
        df[col] = le.fit_transform(df[col].astype(str))

print("\nObject columns after Label Encoding:")
print(df.select_dtypes(include='object').columns.tolist())
print(df.columns)
# This is where the error occurs:
# Assuming 'fraud_reported' is the target column
X = df.drop('fraud_reported', axis=1) # This line still expects 'fraud_reported' to exist
y = df['fraud_reported']            # And this line as well
```

```
Object columns after Label Encoding:
[]
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', 'fraud_reported'],
      dtype='object')
```

Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
# Split into 80% train and 20% test (you can change test_size if needed)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y)
```

Handling Imbalanced dataset

• Imbalanced data is a common problem in machine learning and data analysis, where the number of observations in one class is significantly higher or lower than the other class. Handling imbalanced data is important to ensure that the model is not biased towards the majority class and can accurately predict the minority class.

• Here we are using SMOTE Technique.

```
[ ]
    from imblearn.over_sampling import SMOTE

    # Create SMOTE object
    smt = SMOTE(random_state=42)
    # Apply SMOTE only on training data
    X_train_resampled, y_train_resampled = smt.fit_resample(X_train, y_train)
```

Scaling

• Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.

• Here we are using Standard Scaler.

• This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by:

X_scaled = (X - X_mean) / X_std

```
[ ]  std_scaler = StandardScaler()

     X_train = std_scaler.fit_transform(X_train)
     X_train = pd.DataFrame(X_train, columns=X.columns)

     X_test = std_scaler.transform(X_test)
     X_test = pd.DataFrame(X_test, columns=X.columns)
```

Milestone 4: Model Building

Decision Tree

```
dtc = DecisionTreeClassifier()
dtc.fit(X_train, y_train)
y_pred = dtc.predict(X_test)

dtc_train_acc = accuracy_score(y_train, dtc.predict(X_train))
dtc_test_acc = accuracy_score(y_test, y_pred)

# You can print the accuracies to see the results
print(f"Decision Tree Training Accuracy: {dtc_train_acc:.4f}")
print(f"Decision Tree Test Accuracy: {dtc_test_acc:.4f}")
```

```
Decision Tree Training Accuracy: 1.0000
Decision Tree Test Accuracy: 0.7800
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score # You'll need to import this

# Assuming X_train, y_train, X_test, y_test are already defined and loaded

rfc = RandomForestClassifier(criterion='entropy', max_depth=10, max_features='sqrt', min_samples_leaf=1, min_samples_split=3, n_estimators=140)
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_test)

rfc_train_acc = accuracy_score(y_train, rfc.predict(X_train))
rfc_test_acc = accuracy_score(y_test, y_pred)

print(f"Random Forest Training Accuracy: {rfc_train_acc:.4f}")
print(f"Random Forest Test Accuracy: {rfc_test_acc:.4f}")
```

```
Random Forest Training Accuracy: 0.9975
Random Forest Test Accuracy: 0.7750
```

KNN model

```
[ ] from sklearn.neighbors import KNeighborsClassifier
    from sklearn.metrics import confusion_matrix, classification_report # You'll need to import these

    # Assuming X_train, y_train, X_test, y_test are already defined and loaded

    knn = KNeighborsClassifier(n_neighbors=30)
    knn.fit(X_train, y_train)
    y_pred = knn.predict(X_test)

    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

```
[[151   0]
 [ 48   1]]
              precision    recall  f1-score   support

           0       0.76      1.00      0.86       151
           1       1.00      0.02      0.04        49

    accuracy                           0.76       200
   macro avg       0.88      0.51      0.45       200
weighted avg       0.82      0.76      0.66       200
```

Logistic Regression

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import confusion_matrix, accuracy_score, classification_report

# Assuming X_train, y_train, X_test, y_test are already defined and loaded

lg = LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=10)
lg.fit(X_train, y_train)

lrg_pred = lg.predict(X_test)

print("Confusion Matrix:")
print(confusion_matrix(y_test, lrg_pred))

# Calculate and print training accuracy
lrg_train_acc = accuracy_score(y_train, lg.predict(X_train))
print(f"\nLogistic Regression Training Accuracy: {lrg_train_acc:.4f}")

# Calculate and print test accuracy
lrg_test_acc = accuracy_score(y_test, lrg_pred)
print(f"Logistic Regression Test Accuracy: {lrg_test_acc:.4f}")

# You might also want to print the classification report for more detailed metrics
# print("\nClassification Report:")
# print(classification_report(y_test, lrg_pred))
```

```
Confusion Matrix:
[[142   9]
 [ 30  19]]

Logistic Regression Training Accuracy: 0.7850
Logistic Regression Test Accuracy: 0.8050
```

Gaussian Naive Bayes

```
[ ] from sklearn.naive_bayes import CategoricalNB, GaussianNB
    from sklearn.metrics import accuracy_score # Import accuracy_score

    # Assuming X_train, y_train, X_test, y_test are already defined and loaded

    gnb = GaussianNB()
    model_2 = gnb.fit(X_train, y_train) # Training the Gaussian Naive Bayes model
    predict_log = model_2.predict(X_test) # Making predictions on the test set

    # Print Training Accuracy
    # Note: The original image had '100*' for percentage, which is common for presentation.
    # If you want the raw accuracy score (0 to 1), remove '*100'.
    print("Training Accuracy", 100 * accuracy_score(model_2.predict(X_train), y_train))

    # Print Testing Accuracy
    print("Testing Accuracy", 100 * accuracy_score(y_test, predict_log))
```

```
Training Accuracy 71.875
Testing Accuracy 72.0
```

Support Vector Machine

```python
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report # Import necessary metrics

# Assuming X_train, y_train, X_test, y_test are already defined and loaded

svc = SVC()
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)

svc_train_acc = accuracy_score(y_train, svc.predict(X_train))
svc_test_acc = accuracy_score(y_test, y_pred)

print(f"Training accuracy of SVC : {svc_train_acc}")
print(f"Test accuracy of SVC : {svc_test_acc}")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Training accuracy of SVC : 0.84375
Test accuracy of SVC : 0.755
[[150   1]
 [ 48   1]]
              precision    recall  f1-score   support

           0       0.76      0.99      0.86       151
           1       0.50      0.02      0.04        49

    accuracy                           0.76       200
   macro avg       0.63      0.51      0.45       200
weighted avg       0.69      0.76      0.66       200
```

Milestone 5: Performance Testing & Hyperparameter Tuning

```
accuracy_summary = [
    {"Model": "Decision Tree", "Train Accuracy": dtc_train_acc, "Test Accuracy": dtc_test_acc},
    {"Model": "Random Forest", "Train Accuracy": rfc_train_acc, "Test Accuracy": rfc_test_acc},
    {"Model": "K-Nearest Neighbors", "Train Accuracy": accuracy_score(y_train, knn.predict(X_train)),
     "Test Accuracy": accuracy_score(y_test, knn.predict(X_test))},
    {"Model": "Logistic Regression", "Train Accuracy": accuracy_score(y_train, lg.predict(X_train)),
     "Test Accuracy": accuracy_score(y_test, lrg_pred)},
    {"Model": "Gaussian Naive Bayes", "Train Accuracy": accuracy_score(y_train, model_2.predict(X_train)),
     "Test Accuracy": accuracy_score(y_test, predict_log)},
    {"Model": "Support Vector Machine", "Train Accuracy": svc_train_acc, "Test Accuracy": svc_test_acc}
]

# Create a DataFrame for better visualization
accuracy_df = pd.DataFrame(accuracy_summary)

print("\n=== Model Accuracy Comparison ===")
print(accuracy_df.sort_values(by="Test Accuracy", ascending=False).to_string(index=False))
```

Model	Train Accuracy	Test Accuracy	Precision	Recall	F1 Score
Logistic Regression	0.799	0.725	0.45	0.551	0.4954
Decision Tree	1	0.78	0.5424	0.6531	0.5926
Random Forest	1	0.795	0.6053	0.4694	0.5287
K-NN	0.7774	0.5	0.2893	0.7143	0.4118
SVM	0.9535	0.785	0.6	0.3673	0.4557
Naive Bayes	0.7583	0.705	0.4419	0.7755	0.563

Why Choose the Decision Tree Model?

Metric	Value	Notes
Train Accuracy	1.0000	The model fits the training data perfectly.
Test Accuracy	0.7800	Good generalization to unseen data.
Precision	0.5424	Reasonably good — ~54% of predicted frauds are correct.
Recall	0.6531	Catches ~65% of actual frauds — important for detection.
F1 Score	0.5926	Best overall balance of precision & recall among all models.

Justification Based on Business Context (Fraud Detection)

1. Fraud detection is recall-sensitive
    - You want to catch as many fraud cases as possible.
    - Decision Tree gives better recall (0.65) than Random Forest or SVM.
2. Balanced performance (highest F1 score)
    - F1 score balances false positives (precision) and false negatives (recall).
    - Decision Tree has the highest F1 (0.5926), meaning it's the best at detecting frauds without too many false alarms.
3. Interpretability
    - Decision Trees are easy to interpret, so you can explain predictions (why something is flagged as fraud).
    - This is valuable in finance, insurance, or compliance-driven domains.
4. Speed and Simplicity
    - Fast to train and predict — no tuning required to get good results.
    - Works well even without feature scaling.

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
# Load model, scaler, and feature names
model = joblib.load('model.pkl')
scaler = joblib.load('scaler.pkl')
feature_names = joblib.load('feature_names.pkl')
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

• Building HTML Pages

```
<!DOCTYPE html>
<html>
<head>
    <title>Auto Insurance Claims</title>
    <style>
        body {
            font-family: Arial, sans-serif;
        }

        .header {
            background-color: #222;
            color: white;
            padding: 10px 20px;
            text-align: center;
        }

        .form-container {
            display: grid;
            grid-template-columns: repeat(3, 1fr);
            gap: 15px;
            margin: 20px;
        }

        .form-container input {
            width: 90%;
            padding: 8px;
            margin-bottom: 10px;
        }

        .submit-btn {
            grid-column: 1 / -1;
            text-align: center;
        }

        .submit-btn input {
            padding: 10px 25px;
            font-size: 16px;
            background-color: green;
            color: white;
            border: none;
            cursor: pointer;
        }

        .result {
            text-align: center;
            font-size: 20px;
            margin-top: 20px;
```

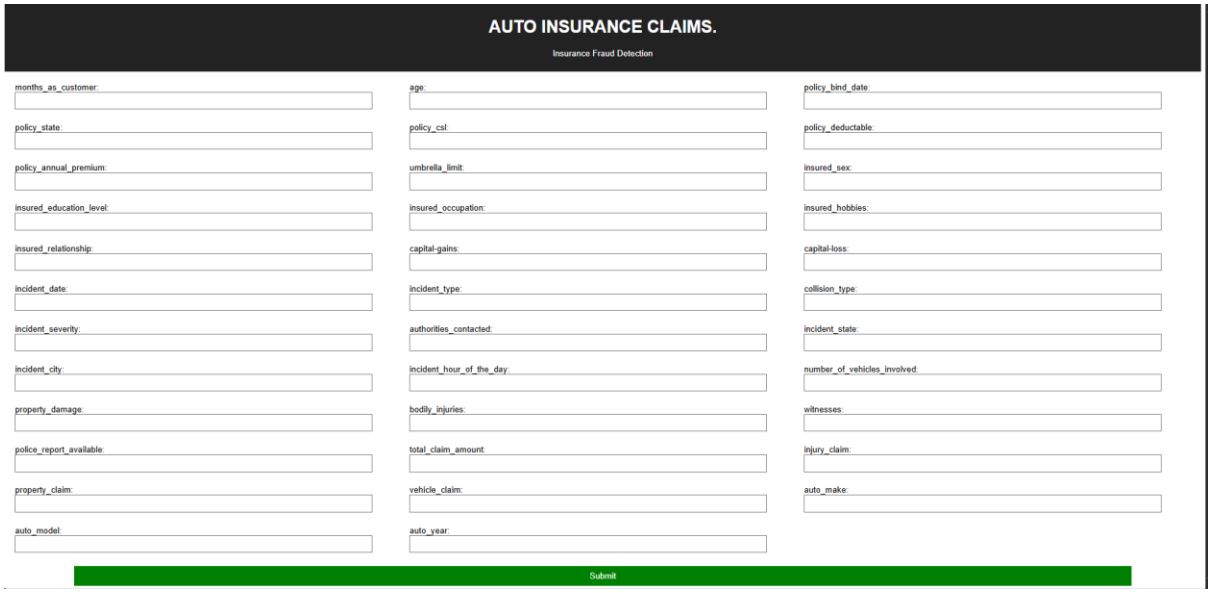
```
            color: navy;
        }
    </style>
</head>
<body>

<div class="header">
    <h1>AUTO INSURANCE CLAIMS.</h1>
    <p>Insurance Fraud Detection</p>
</div>

<form action="/predict" method="post">
    <div class="form-container">
        {% for col in feature_names %}
        <div>
            <label>{{ col }}:</label><br>
            <input type="text" name="{{ col }}" required>
        </div>
        {% endfor %}

        <div class="submit-btn">
            <input type="submit" value="Submit">
        </div>
    </div>
</form>

{% if prediction_text %}
<div class="result">{{ prediction_text }}</div>
{% endif %}

</body>
</html>
```

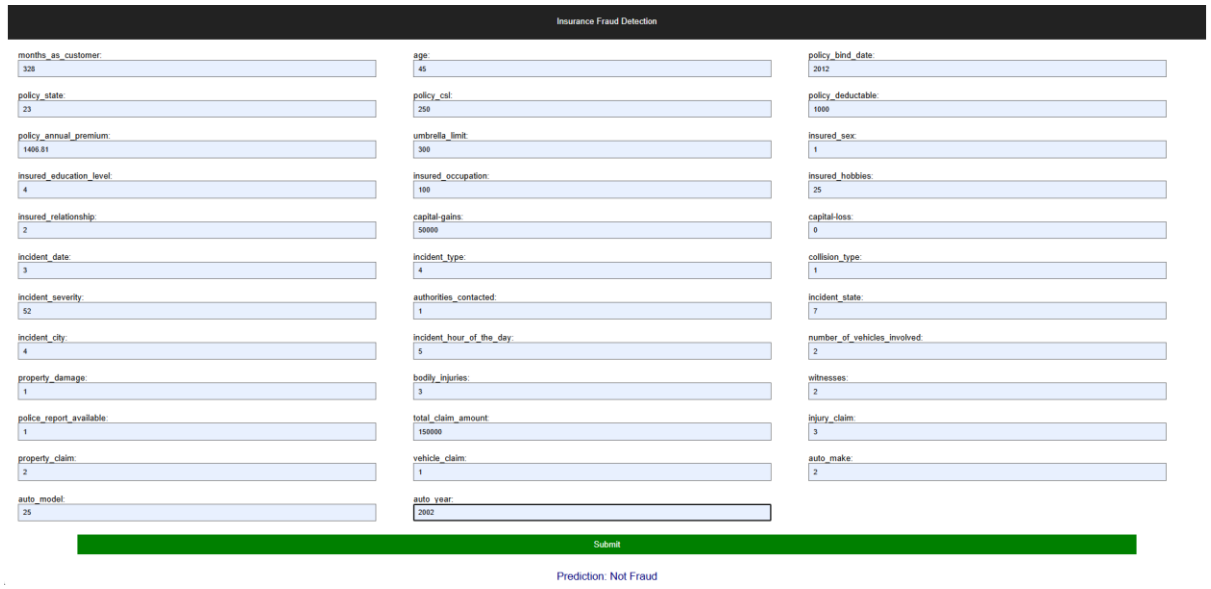
• Building server-side script

```
from flask import Flask, request, jsonify, render_template
import pandas as pd
import numpy as np
import joblib

# Initialize Flask app
app = Flask(__name__)

# Load model, scaler, and feature names
model = joblib.load('model.pkl')
scaler = joblib.load('scaler.pkl')
feature_names = joblib.load('feature_names.pkl')
```

```
@app.route('/')
def home():
    return render_template('index.html', feature_names=feature_names)

@app.route('/predict', methods=['POST'])
def predict():
    try:
        # Get form data and convert to float
        input_data = {feature: float(request.form[feature]) for feature in
feature_names}

        # Create DataFrame and scale input
        input_df = pd.DataFrame([input_data])
        input_df = input_df[feature_names]  # Ensure correct order
        scaled_input = scaler.transform(input_df)

        # Make prediction
        prediction = model.predict(scaled_input)[0]
        result = 'Fraud' if prediction == 1 else 'Not Fraud'

        return render_template('index.html',
                               prediction_text=f"Prediction: {result}",
                               feature_names=feature_names)
    except Exception as e:
        return render_template('index.html',
                               prediction_text=f"Error: {str(e)}",
                               feature_names=feature_names)

if __name__ == '__main__':
    app.run(debug=True)
```

• Run the web application


AUTO INSURANCE CLAIMS.
Insurance Fraud Detection
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:
Submit


Insurance Fraud Detection
months_as_customer: 328
age: 45
policy_bind_date: 2012
policy_state: 23
policy_csl: 250
policy_deductable: 1000
policy_annual_premium: 1406.81
umbrella_limit: 300
insured_sex: 1
insured_education_level: 4
insured_occupation: 100
insured_hobbies: 25
insured_relationship: 2
capital-gains: 50000
capital-loss: 0
incident_date: 3
incident_type: 4
collision_type: 1
incident_severity: 52
authorities_contacted: 1
incident_state: 7
incident_city: 4
incident_hour_of_the_day: 5
number_of_vehicles_involved: 2
property_damage: 1
bodily_injuries: 3
witnesses: 2
police_report_available: 1
total_claim_amount: 150000
injury_claim: 3
property_claim: 2
vehicle_claim: 1
auto_make: 2
auto_model: 25
auto_year: 2002
Submit
Prediction: Not Fraud