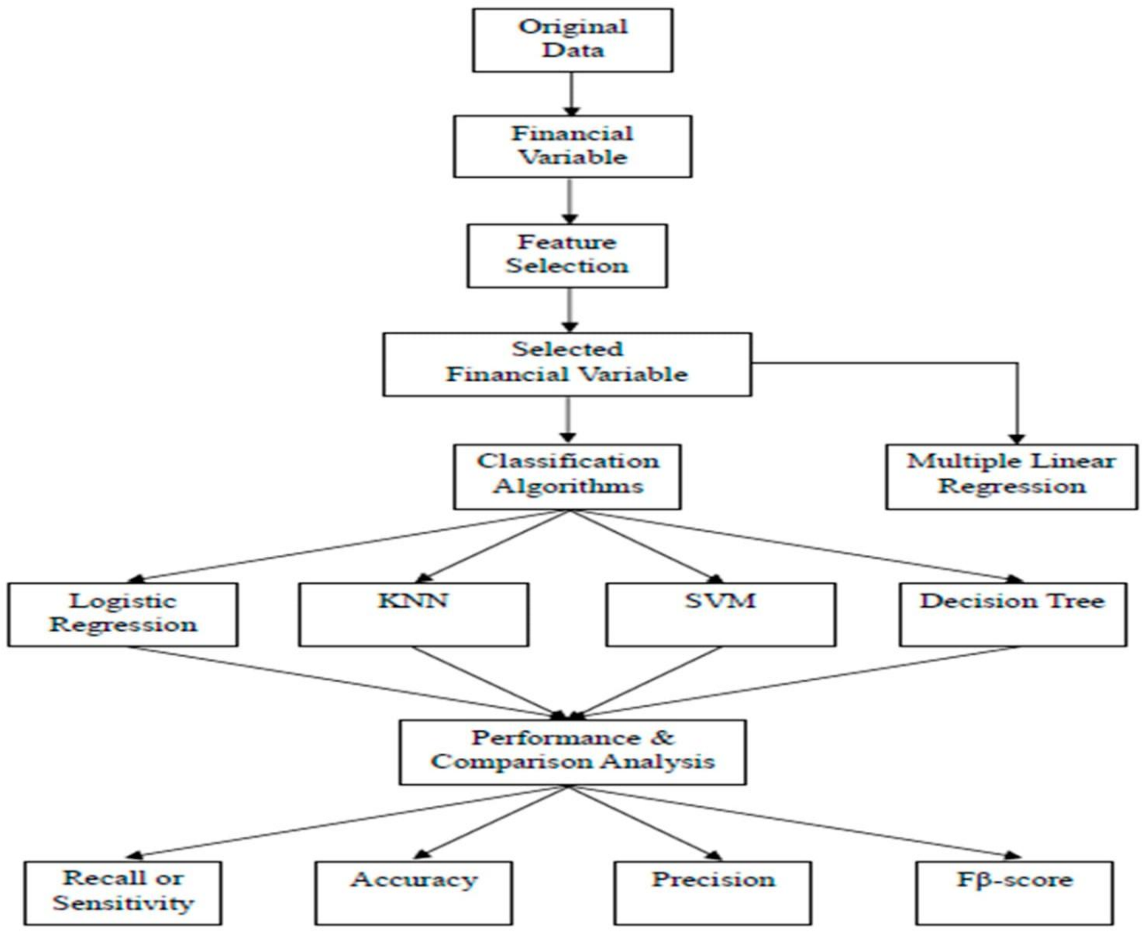
Auto Insurance Fraud Detection Using Machine Learning

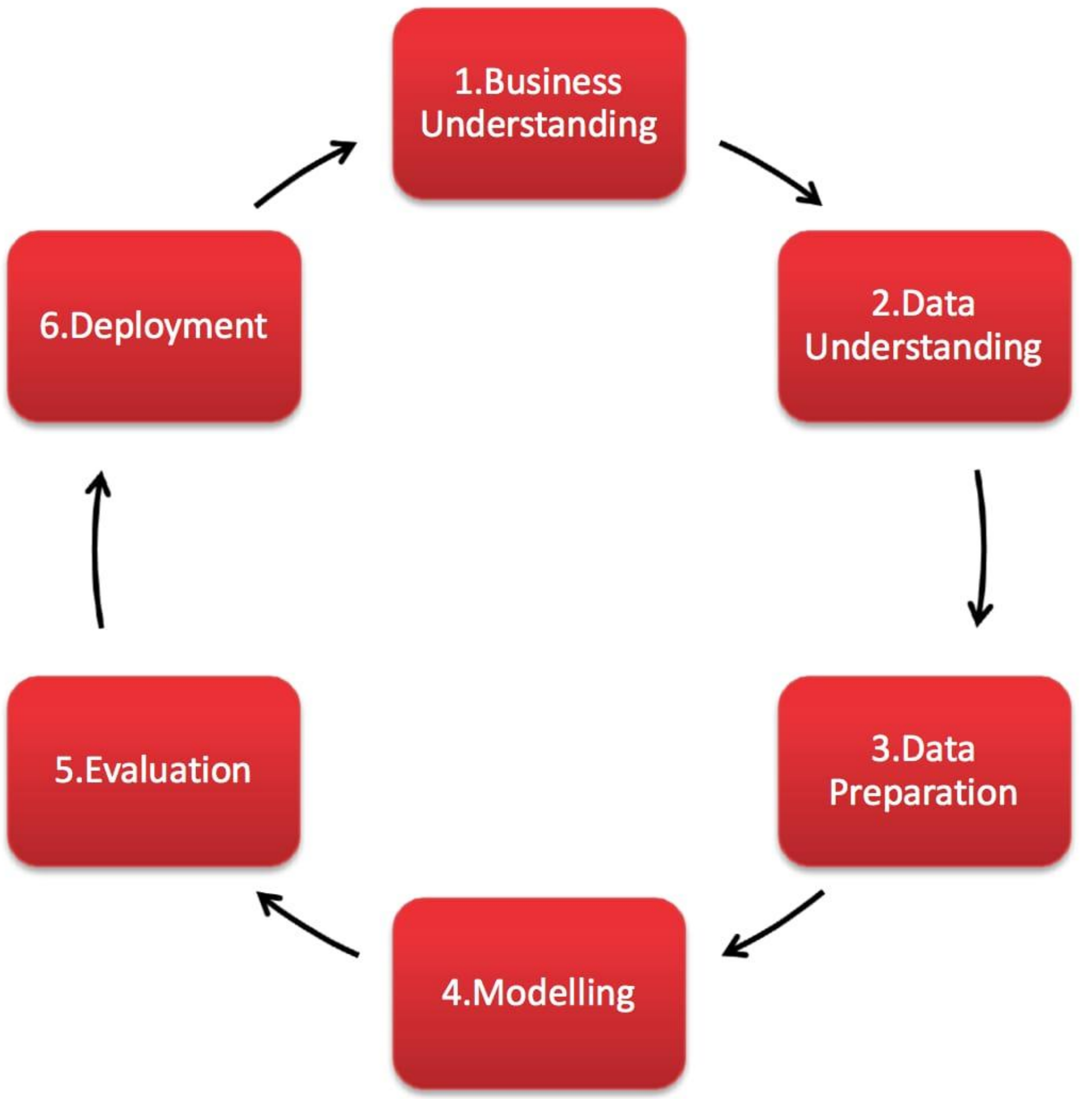
Insurance provides financial protection against unexpected losses or damages.However, in recent times, many individuals have started exploiting this system by making fraudulent claims-requests for compensation to which they are not legitimately entitled.This practice,often referred to as insurance fraud, can also occur when an insurer deliberately denies benefits that are rightfully due to claimant.

Such fraudulent activities result in significant financial losses for insurance companies .Despite the rising number of frauds , only a small fraction are actually detected.Therefore,it is crucial to develop an Insurance fraud detection System that can accurately identify whether an insurance claim is legitimate or fraudulent , based on various features and parameters.

Technical Architecture:

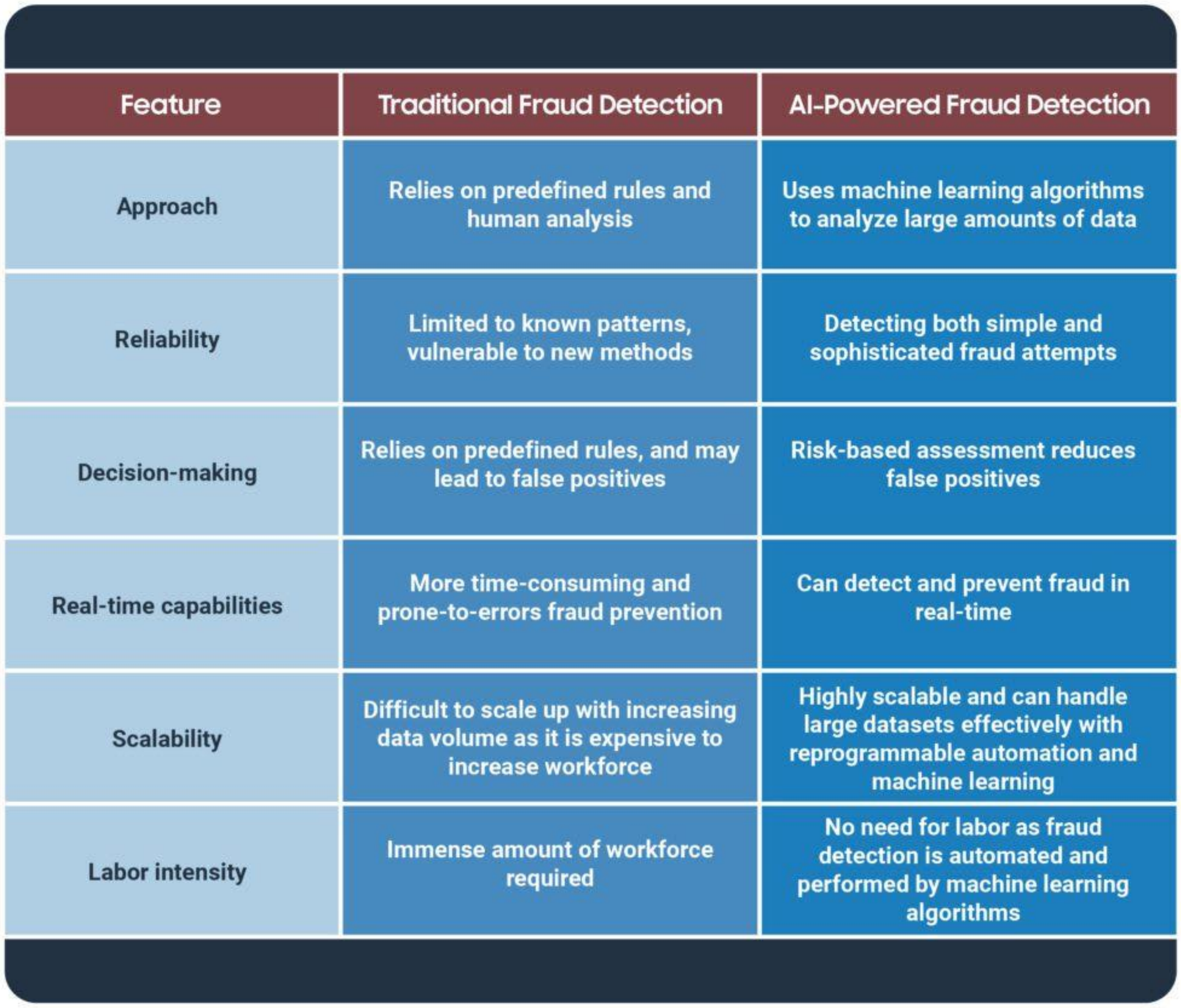


Business Understanding



A circular diagram illustrating the six phases of the CRISP-DM methodology, starting with business understanding.
Business understanding encompasses multiple dimensions including problem scope definition, stakeholder identification, resource assessment, and success criteria establishment [12][13]. This phase requires collaboration between domain experts, technical teams, and business stakeholders to ensure alignment between technical capabilities and business objectives [2][15]. The iterative nature of CRISP-DM allows for continuous refinement of problem understanding as project insights emerge [14][16].

Comparison between Traditional Fraud detection and AI powered fraud detection



Feature	Traditional Fraud Detection	AI-Powered Fraud Detection
Approach	Relies on predefined rules and human analysis	Uses machine learning algorithms to analyze large amounts of data
Reliability	Limited to known patterns, vulnerable to new methods	Detecting both simple and sophisticated fraud attempts
Decision-making	Relies on predefined rules, and may lead to false positives	Risk-based assessment reduces false positives
Real-time capabilities	More time-consuming and prone-to-errors fraud prevention	Can detect and prevent fraud in real-time
Scalability	Difficult to scale up with increasing data volume as it is expensive to increase workforce	Highly scalable and can handle large datasets effectively with reprogrammable automation and machine learning
Labor intensity	Immense amount of workforce required	No need for labor as fraud detection is automated and performed by machine learning algorithms

Comparison of traditional and AI-powered fraud detection methods highlighting differences in approach, reliability, decision-making, real-time capabilities, scalability, and labor intensity.

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

o Supervised learning: https://www\.tutorialspoint\.com

o Unsupervised learning:https://www\.tutorialspoint\.com

· Decision tree: https://www\.tutorialspoint\.com

· Random forest: https://www\.tutorialspoint\.com

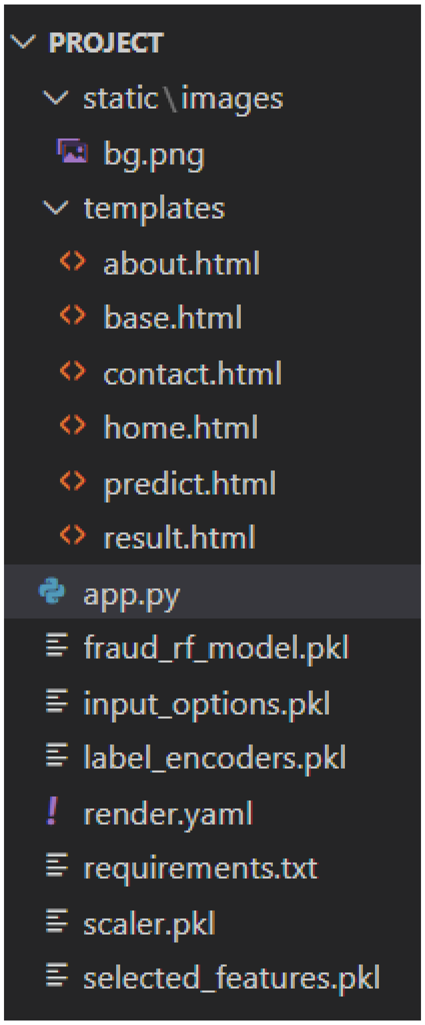
· KNN: https://www\.tutorialspoint\.com

·Xgboost:https://www\.geeksforgeeks\.org

· Evaluation metrics: https://www\.geeksforgeeks\.org

Flask Basics : https://www.youtube.com/watch?v=lj4I_CvBnt0

Project Structure:

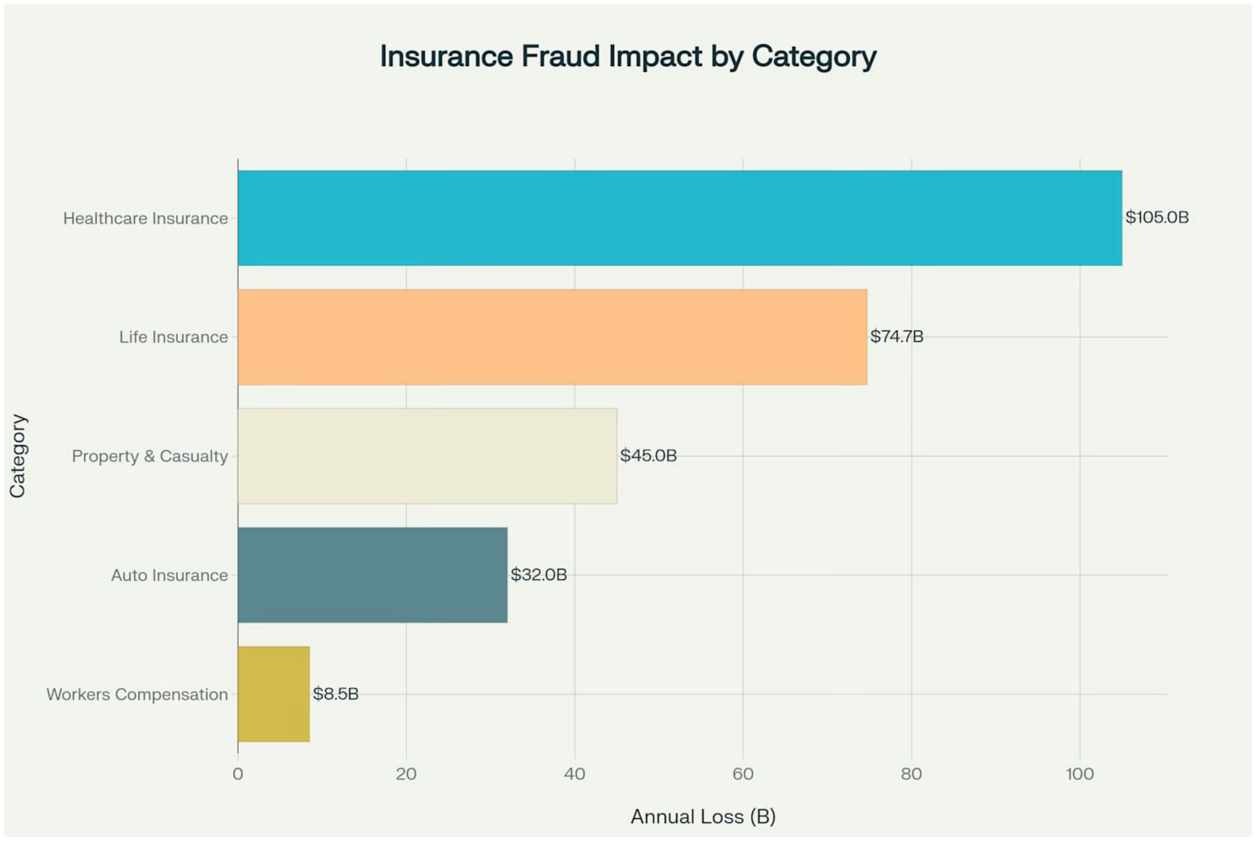


```
PROJECT
  static\images
    bg.png
  templates
    about.html
    base.html
    contact.html
    home.html
    predict.html
    result.html
  app.py
  fraud_rf_model.pkl
  input_options.pkl
  label_encoders.pkl
  render.yaml
  requirements.txt
  scaler.pkl
  selected_features.pkl
```

Milestone 1:Define Problem / Problem Understanding

Activity 1: Specify the business problem

Auto insurance fraud constitutes a significant portion of the broader insurance fraud landscape, contributing approximately $32 billion annually to total fraud losses in the United States . The Coalition Against Insurance Fraud estimates that fraudulent claims overall cost American consumers more than $80 billion annually, with property and casualty insurance fraud representing $45 billion of this total. These substantial financial losses directly impact honest policyholders through increased premiums, with the average American family paying between $400 and $700 annually due to fraud-related costs



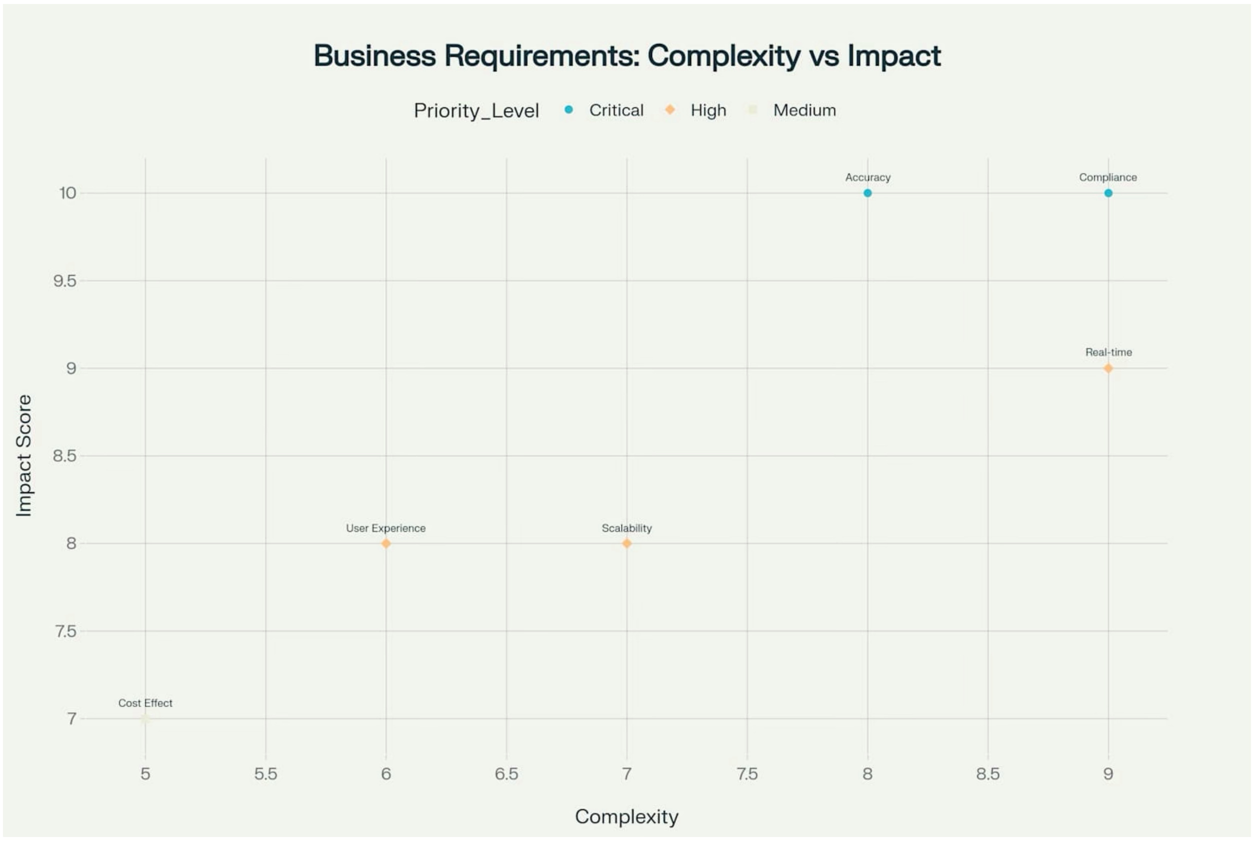
Annual financial losses from different types of insurance fraud, highlighting auto insurance as a significant contributor

The insurance industry faces multiple categories of fraudulent activities, ranging from hard fraud involving deliberately staged accidents to soft fraud encompassing claim exaggeration and premium manipulation [5][7]. Traditional rule-based detection systems struggle with evolving fraud patterns, generating high false positive rates that burden investigation resources and delay legitimate claim processing [8][9]. Modern fraud detection requires sophisticated approaches capable of identifying complex patterns while minimizing disruption to genuine customers

Activity 2: Business requirements

Auto insurance fraud detection systems must satisfy demanding functional requirements to operate effectively in production environments[]. Accuracy and reliability represent critical priorities, with industry standards requiring minimum 95% accuracy, 95% precision, and 90% recall rates to balance fraud detection effectiveness with customer experience. Real-time processing capabilities are essential, with systems needing to evaluate claims within 100 milliseconds to avoid disrupting customer interactions.

Scalability requirements reflect the high-volume nature of insurance operations, with systems handling 10,000+ claims daily while maintaining consistent performance. Regulatory compliance adds complexity through requirements for model explainability, audit trails, and adherence to data protection regulations such as GDPR. These compliance requirements necessitate transparent decision-making processes and comprehensive documentation throughout the detection workflow.



Business requirements analysis showing the relationship between implementation complexity and business impact by priority level

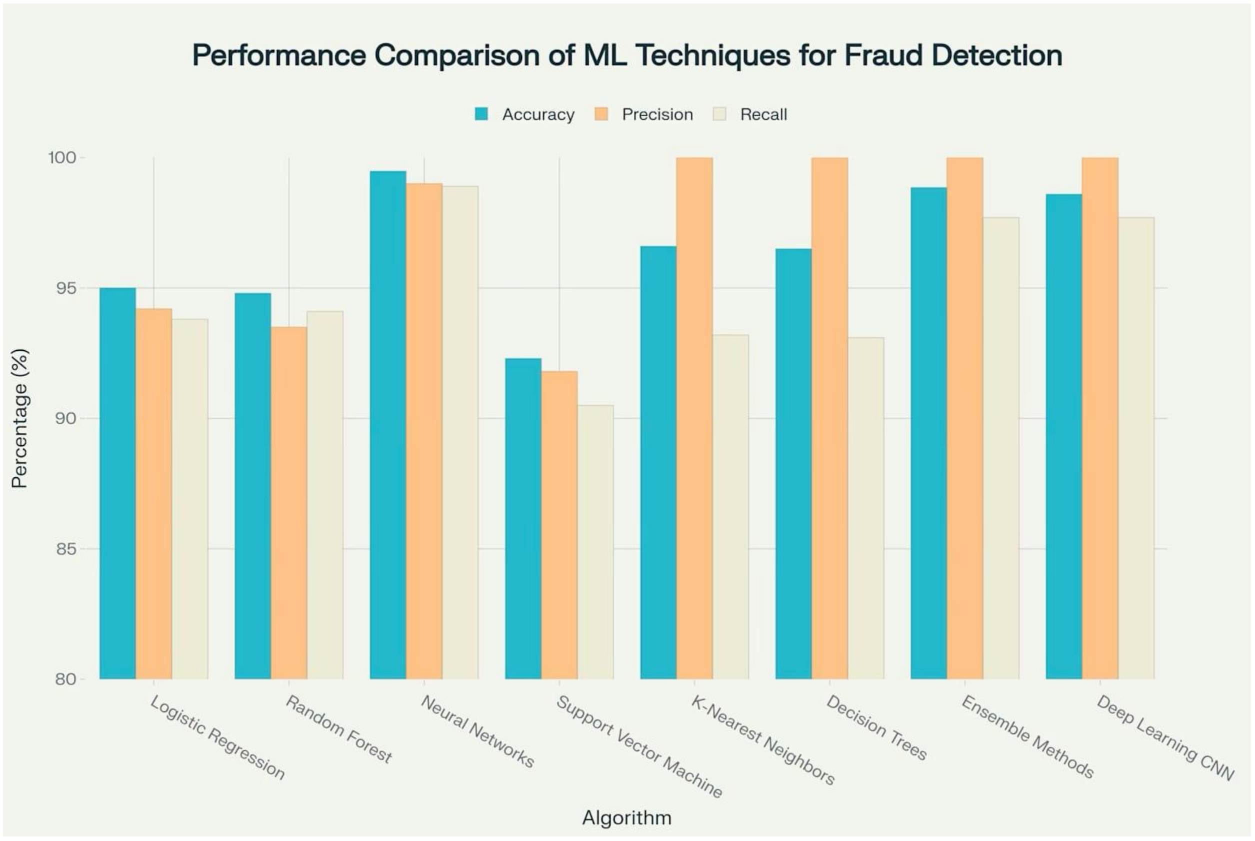
Cost effectiveness considerations balance implementation investments against operational savings and fraud prevention benefits. Organizations typically target positive return on investment within 12 months, supported by reduced investigation costs and decreased false positive rates. User experience requirements ensure seamless integration with existing claims processing workflows while providing intuitive interfaces for fraud investigator.

Activity 3: Literature Survey

Contemporary research demonstrates significant advancement in machine learning applications for fraud detection, with multiple algorithmic approaches showing promising results [14][15].

Supervised learning techniques dominate the field, utilizing labeled historical data to train models capable of distinguishing fraudulent from legitimate transactions . The prevalence of logistic regression, neural networks, and ensemble methods reflects their proven effectiveness in handling complex fraud patterns.

Recent systematic literature reviews indicate growing adoption of deep learning techniques, with convolutional neural networks achieving accuracy rates exceeding 98% in vehicle insurance fraud detection scenarios. Ensemble methods consistently demonstrate superior performance, combining multiple base learners to achieve accuracy rates of 98.85% with precision and recall scores above 97%[1]. These performance improvements represent substantial advancement over traditional rule-based approaches.



Performance metrics comparison of machine learning algorithms for auto insurance fraud detection

Research gaps persist in several areas, including limited real-world deployment studies, insufficient focus on adversarial fraud patterns, and inadequate handling of imbalanced datasets. The majority of published studies utilize synthetic or heavily processed datasets, limiting the generalizability of findings to operational environments. Additionally, few studies provide comprehensive cost-benefit analysis or examine long-term model performance under evolving fraud patterns

Activity 4: Social or Business Impact.

Auto insurance fraud detection systems provide substantial societal benefits through consumer protection and cost reduction mechanisms. Effective fraud prevention directly reduces premium costs for honest policyholders, addressing the $400-700 annual burden imposed by fraudulent activities [5][6]. Enhanced detection capabilities improve industry integrity and public confidence in insurance systems, supporting broader economic stability.

Resource optimization represents another significant social benefit, freeing investigative personnel from routine false positive reviews to focus on complex fraud schemes requiring human expertise. Faster processing of legitimate claims improves customer experience during stressful periods following accidents or losses. The technology contributes to broader efforts combating organized fraud networks that often engage in additional criminal activities.

Business impact-Organizations implementing machine learning fraud detection typically achieve substantial return on investment through multiple value streams. Direct fraud prevention reduces claim payouts by 70-80%, while operational efficiency improvements decrease investigation costs by approximately 40%. Processing time reductions of 30% for legitimate claims enhance customer satisfaction and competitive positioning. AI fraud detection use cases across auto, health, and property & casualty insurance lines with specific fraud types and AI techniques.The business model extends beyond immediate cost savings to encompass strategic advantages including improved risk assessment, enhanced underwriting capabilities, and data-driven product development [1][23]. Organizations report 200-1000% return on investment within 7 months of implementation, justifying substantial initial investments in technology and personnel [17][23]. Competitive differentiation emerges through superior customer experience and more accurate pricing models.

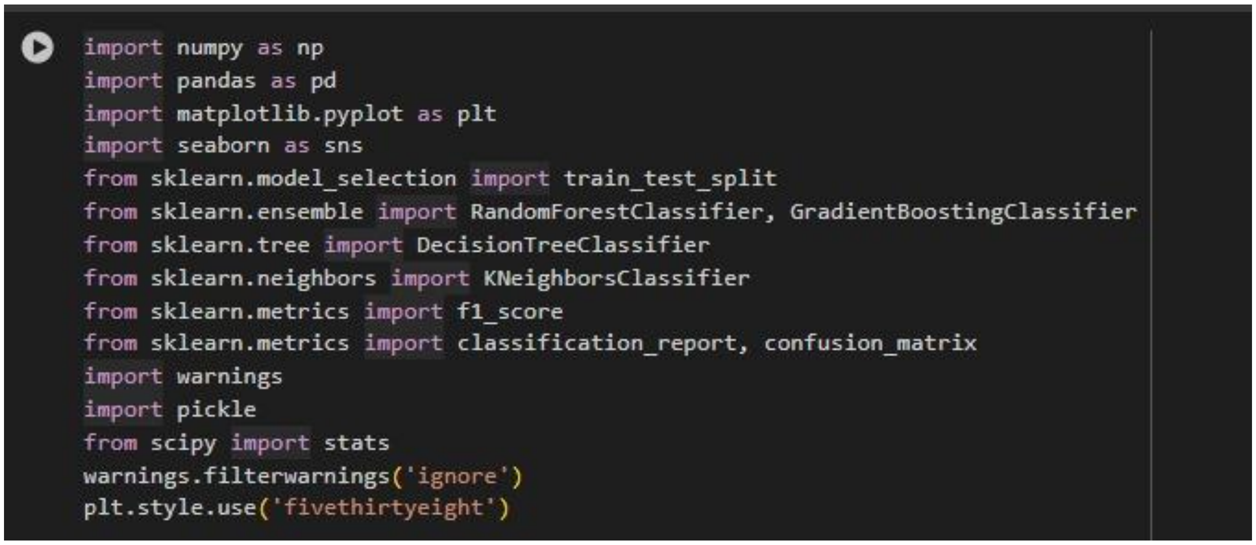
Milestone 2: Data Collection & Preparation

Collect the dataset

We have collected the dataset from Kaggle. The link is given below:

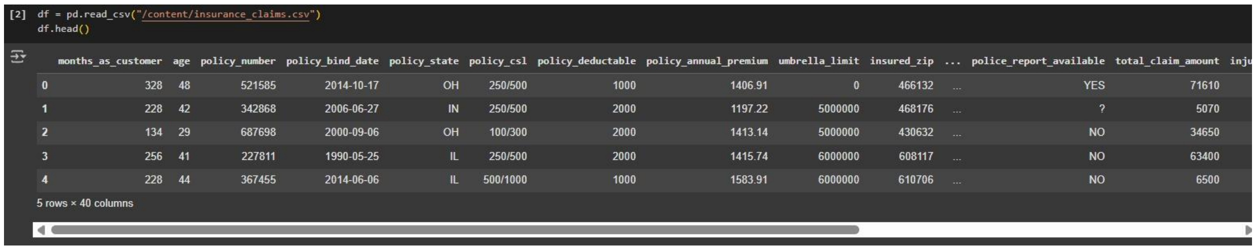
https://www.kaggle.com/datasets/buntyshah/auto-insurance-claims-data

Importing the libraries



```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
from scipy import stats
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
```

Read the Dataset



```
[2] df = pd.read_csv("/content/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_available  total_claim_amount  inju
0                 328   48         521585        2014-10-17            OH     250/500               1000                1406.91               0       466132  ...                      YES               71610
1                 228   42         342868        2006-06-27            IN     250/500               2000                1197.22         5000000       468176  ...                        ?                5070
2                 134   29         687698        2000-09-06            OH     100/300               2000                1413.14         5000000       430632  ...                       NO               34650
3                 256   41         227811        1990-05-25            IL     250/500               2000                1415.74         6000000       608117  ...                       NO               63400
4                 228   44         367455        2014-06-06            IL    500/1000               1000                1583.91         6000000       610706  ...                       NO                6500
5 rows × 40 columns
```

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.

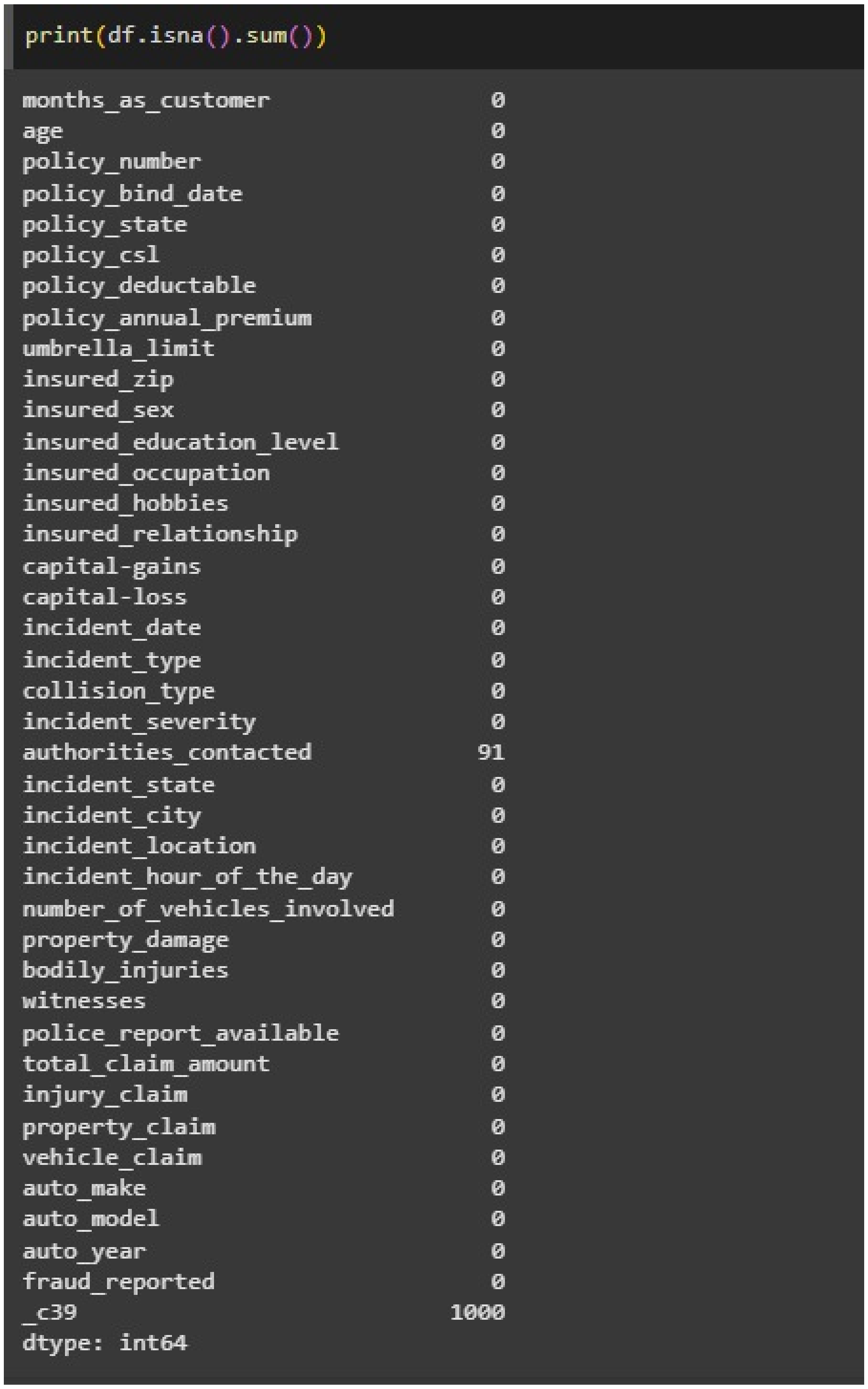
● Handling missing values

● 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.

Handling missing values

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.

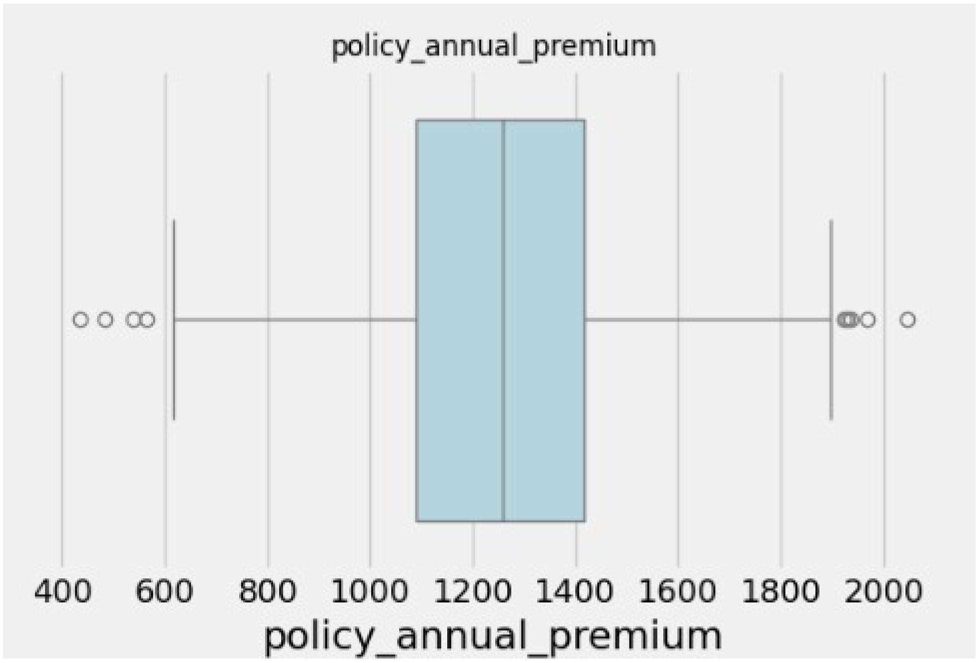


```
print(df.isna().sum())
```

```
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
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
dtype: int64
```

Handling Outliers

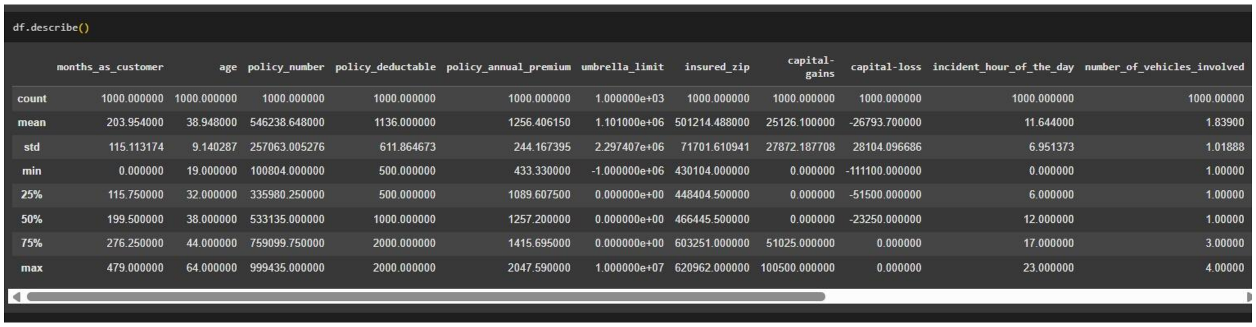
With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of policy_annual_premium feature with some mathematical formula. · From the below diagram, we could visualize that policy_annual_premium feature has outliers. Boxplot from seaborn library is used here



Milestone 3: Exploratory Data Analysis

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()
```

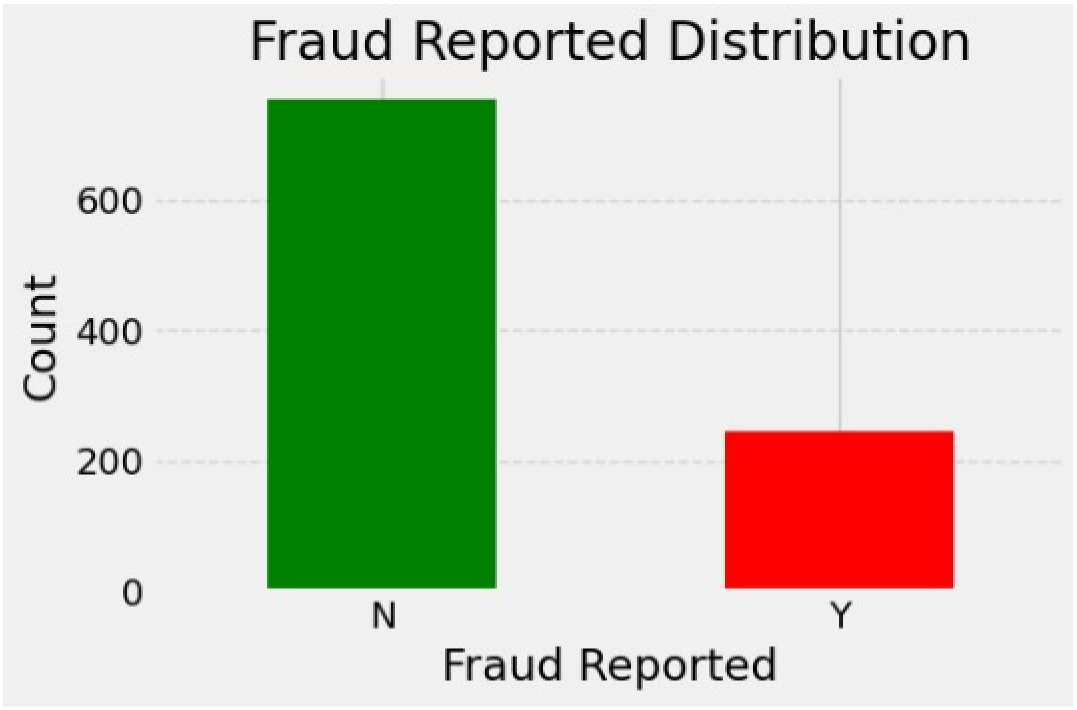
```
       months_as_customer          age  policy_number  policy_deductable  policy_annual_premium  umbrella_limit    insured_zip        capital-  capital-loss  incident_hour_of_the_day  number_of_vehicles_involved
                                                                                                                                       gains
count         1000.000000  1000.000000    1000.000000        1000.000000            1000.000000    1.000000e+03    1000.000000     1000.000000   1000.000000               1000.000000                   1000.00000
mean           203.954000    38.948000  546238.648000        1136.000000            1256.406150    1.101000e+06  501214.488000    25126.100000 -26793.700000                 11.644000                      1.83900
std            115.113174     9.140287  257063.005276         611.864673             244.167395    2.297407e+06   71701.610941    27872.187708  28104.096686                  6.951373                      1.01888
min              0.000000    19.000000  100804.000000         500.000000             433.330000   -1.000000e+06  430104.000000        0.000000 -111100.000000                 0.000000                      1.00000
25%            115.750000    32.000000  335980.250000         500.000000            1089.607500    0.000000e+00  448404.500000        0.000000 -51500.000000                  6.000000                      1.00000
50%            199.500000    38.000000  533135.000000        1000.000000            1257.200000    0.000000e+00  466445.500000        0.000000 -23250.000000                 12.000000                      1.00000
75%            276.250000    44.000000  759099.750000        2000.000000            1415.695000    0.000000e+00  603251.000000    51025.000000      0.000000                 17.000000                      3.00000
max            479.000000    64.000000  999435.000000        2000.000000            2047.590000    1.000000e+07  620962.000000   100500.000000      0.000000                 23.000000                      4.00000
```

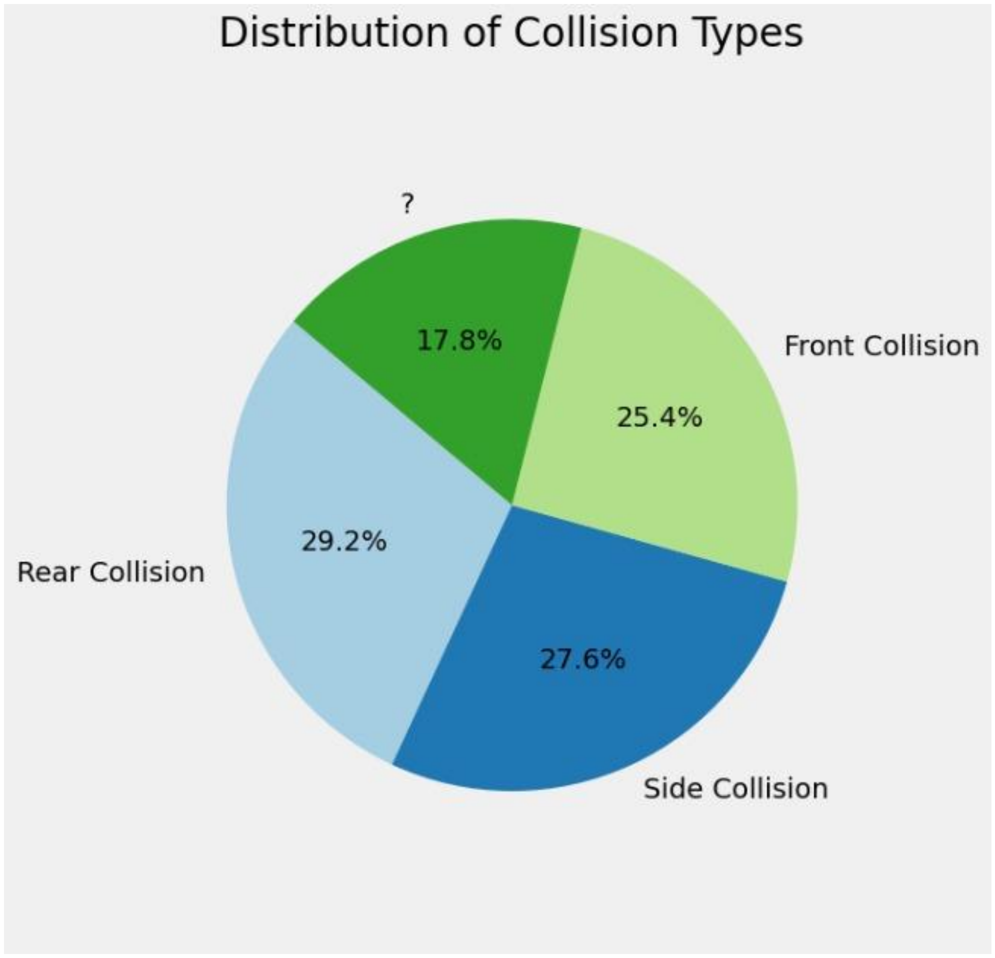
Visual analysis

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.

Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as Piechart and countplot. Seaborn package provides a wonderful function countplot. It is more useful for categorical features. With the help of countplot, we can Number of unique values in the feature. From the countplot we can say that there are only 247 fraud cases reported in 1000 insurance claims.

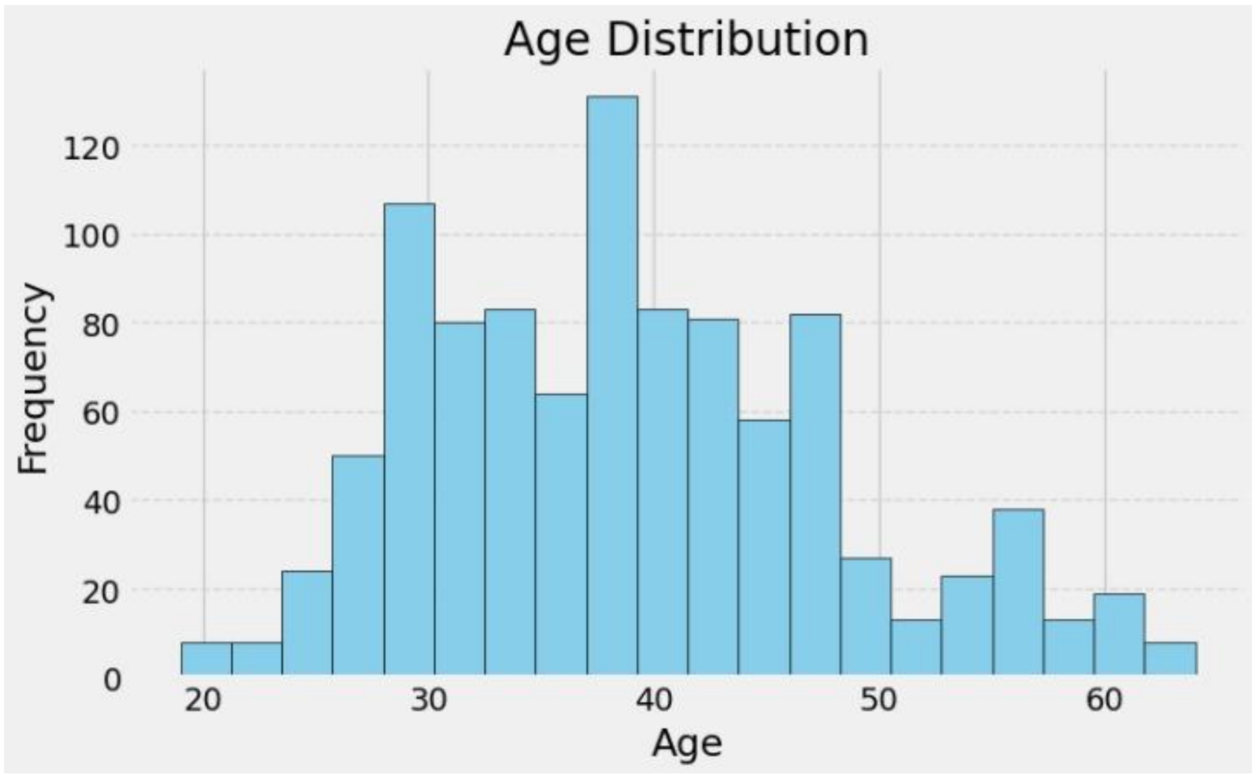




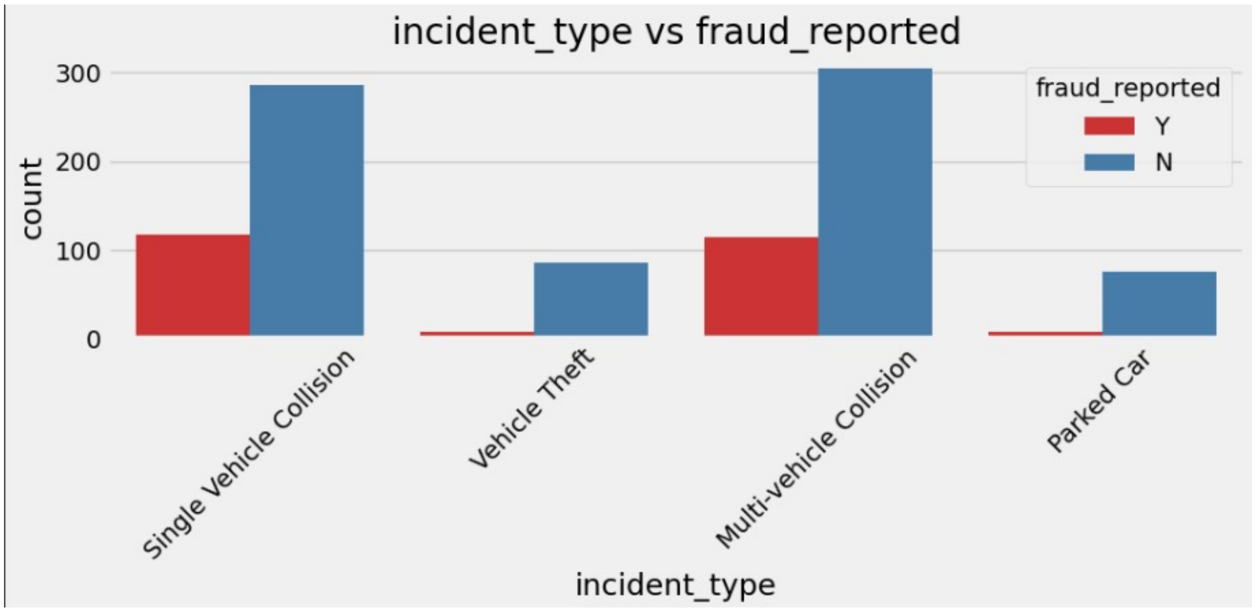
Collision Types and Their Proportions:

1. **Rear Collision – 29.2%**
    - This is the **most common** type of collision in the chart.
    - It occurs when one vehicle hits the back of another.
    - Example: Sudden braking leading to a car hitting from behind.
2. **Side Collision – 27.6%**
    - The second most frequent.
    - This happens when a vehicle is struck on its side, often at intersections or lane changes.

3. **Front Collision** – **25.4%**
    - This involves head-on impacts.
    - Can be due to wrong-lane driving, or hitting a stationary object from the front.
4. **? (Unknown or Unclassified)** – **17.8%**
    - This portion represents collisions that **haven't been clearly categorized**.
    - The label is just a "?", which may indicate **missing data** or **miscellaneous types**.



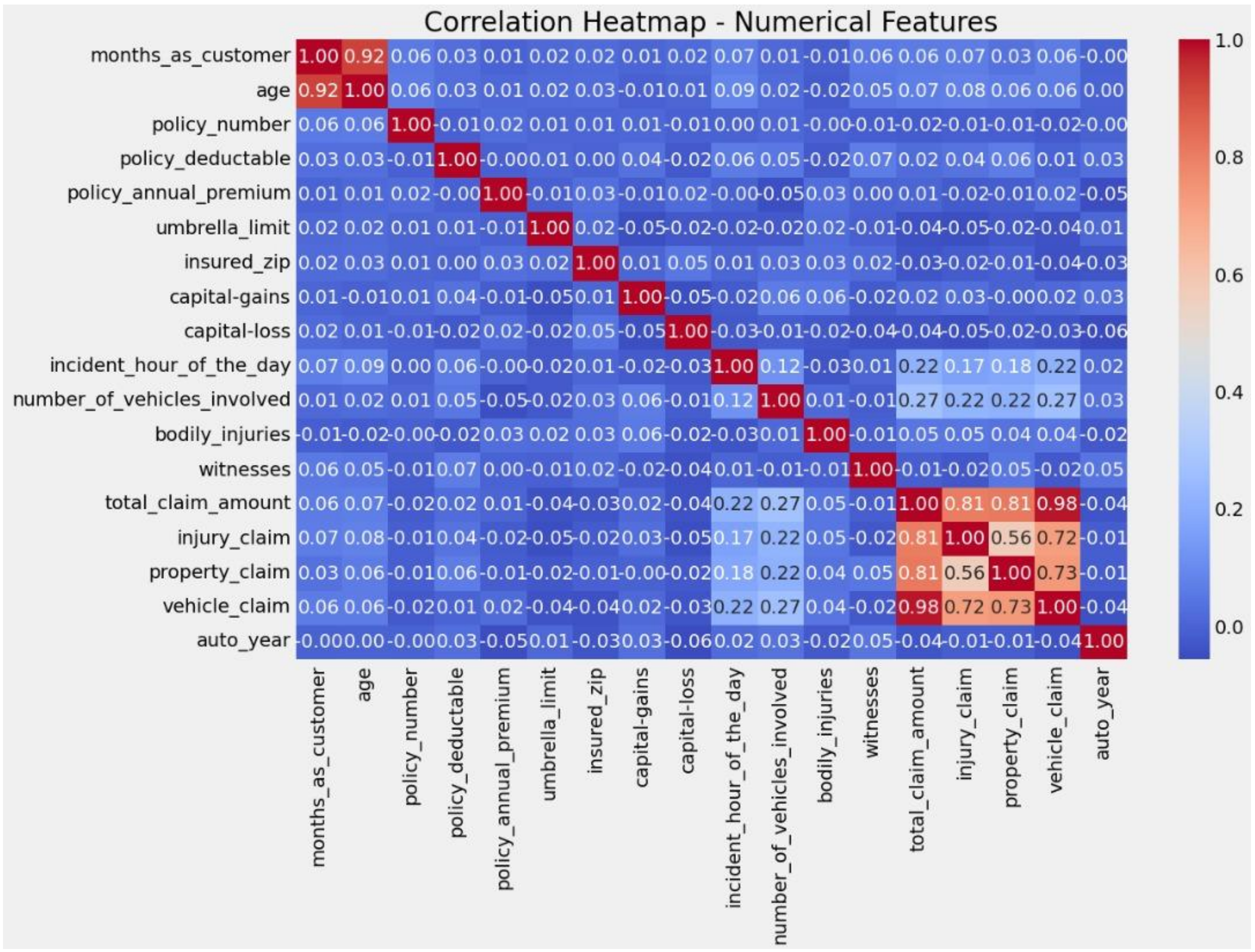
Bivariate analysis



1. **Single Vehicle Collision:**
    - Large number of total incidents.
    - A significant amount of fraud reported (red bar is large).
    - Many non-fraud cases too (even larger blue bar).
2. **Vehicle Theft:**
    - Total incidents are low compared to others.
    - Very few fraud cases (small red bar).
    - Most are non-fraud.
3. **Multi-vehicle Collision:**
    - Highest number of total incidents.
    - Very similar fraud pattern to Single Vehicle Collision.
    - Many cases involve fraud reports (red bar is high).

4. **Parked Car:**
    - Few total incidents.
    - Almost no fraud reported (red bar barely visible).
    - Most are genuine claims.

Multivariate analysis



A **correlation heatmap** shows how strongly two variables are related to each other:

- **Values range from -1 to 1**:
    - **1.00** = perfect positive correlation (they increase together).
    - **-1.00** = perfect negative correlation (one increases, the other decreases).

    - **0.00** = no correlation (completely unrelated).

- **Color Scale**:
    - **Red** = strong positive correlation.
    - **Blue** = weak or no correlation.
    - **White/Pale** = moderate correlation.

Each square shows the **correlation value between two variables**. Diagonal values are all 1.00 (a feature with itself).

Here are key insights:

1. **months_as_customer** ↔ **age** = **0.92**
    - Older customers have typically been with the company longer.
    - These two variables are **highly correlated**, possibly redundant.
2. **total_claim_amount** ↔ **vehicle_claim (0.98)**, **injury_claim (0.81)**, **property_claim (0.72)**
    - These three types of claims **heavily contribute** to the total claim amount.
    - This is expected, as total = injury + property + vehicle.
3. **injury_claim** ↔ **property_claim (0.56)** and **vehicle_claim (0.73)**
    - These different claims often occur together — e.g., an injury often comes with vehicle damage.

Most other variables (like `insured_zip`, `policy_number`, `capital-gains`, etc.) show **very low correlation values** (mostly under 0.1) with others, which means they behave **independently**.

`policy_number` has almost no correlation with anything — expected, since it's likely a random ID.

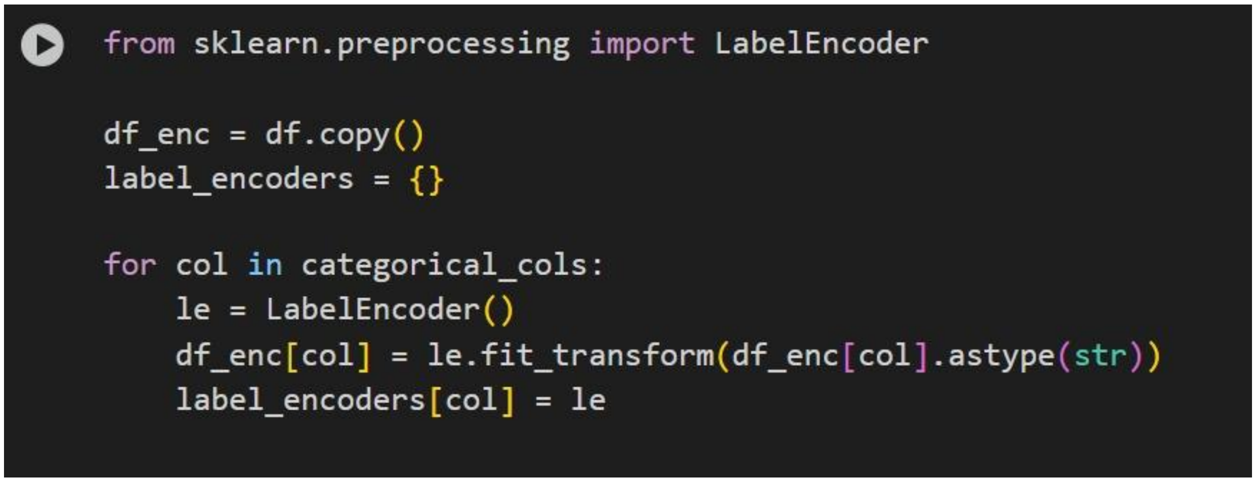
`incident_hour_of_the_day`, `number_of_vehicles_involved`, and `witnesses` also have low correlations with other variables — may need further analysis to assess their influence on fraud detection or claim amount.

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.

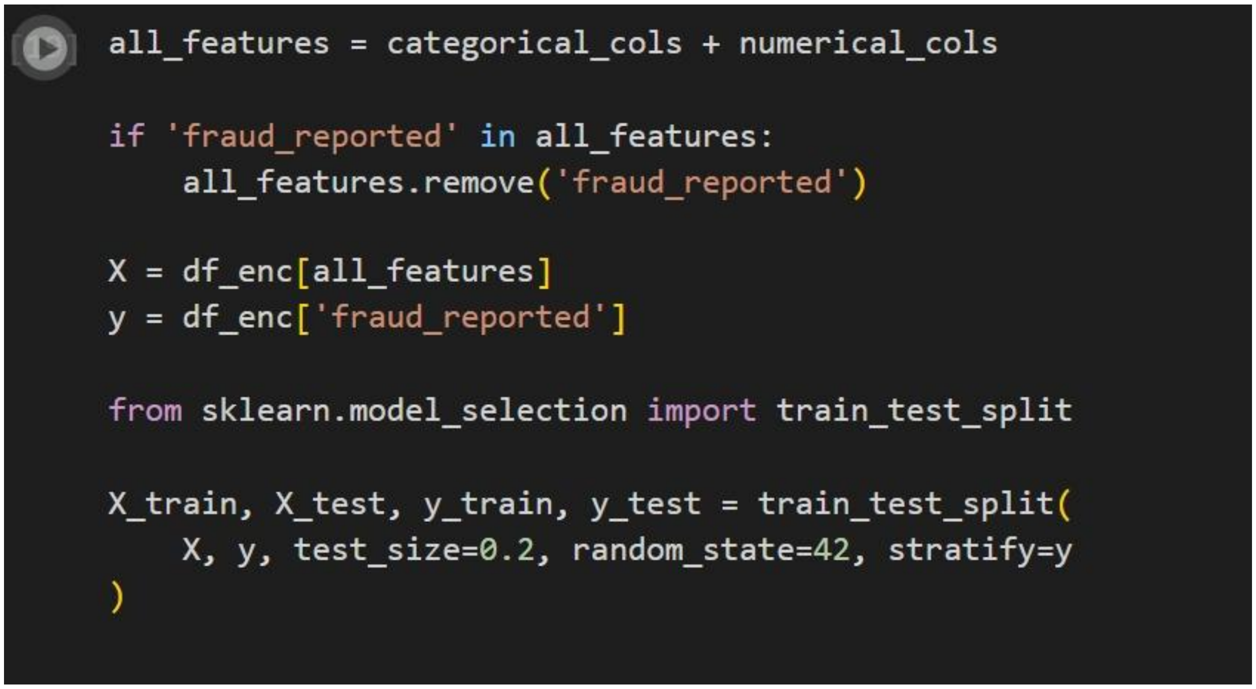
• Here we are applying fit_transform to transform the categorical features to numerical features.



```python
from sklearn.preprocessing import LabelEncoder

df_enc = df.copy()
label_encoders = {}

for col in categorical_cols:
    le = LabelEncoder()
    df_enc[col] = le.fit_transform(df_enc[col].astype(str))
    label_encoders[col] = le
```

Splitting data into train and test

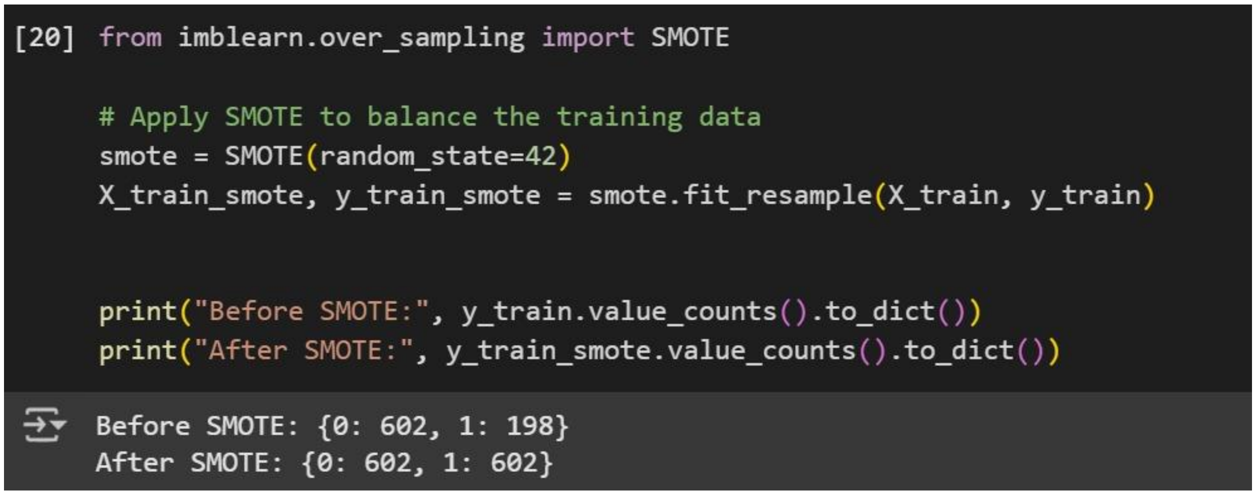
Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.


```
all_features = categorical_cols + numerical_cols

if 'fraud_reported' in all_features:
    all_features.remove('fraud_reported')

X = df_enc[all_features]
y = df_enc['fraud_reported']

from sklearn.model_selection import train_test_split

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.


```
[20] from imblearn.over_sampling import SMOTE

     # Apply SMOTE to balance the training data
     smote = SMOTE(random_state=42)
     X_train_smote, y_train_smote = smote.fit_resample(X_train, y_train)


     print("Before SMOTE:", y_train.value_counts().to_dict())
     print("After SMOTE:", y_train_smote.value_counts().to_dict())
```

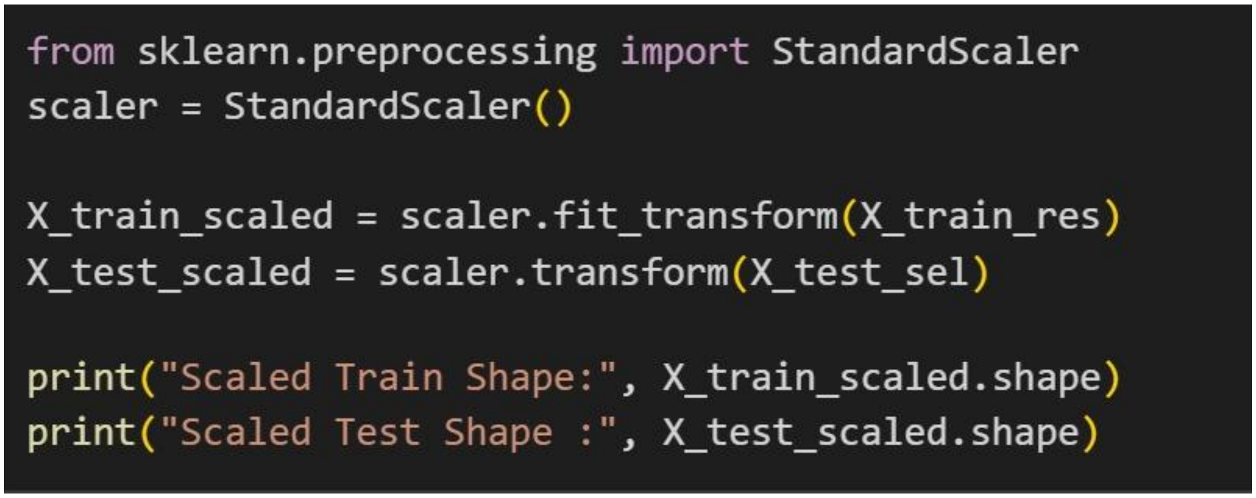
```
Before SMOTE: {0: 602, 1: 198}
After SMOTE: {0: 602, 1: 602}
```

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



```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train_res)
X_test_scaled = scaler.transform(X_test_sel)

print("Scaled Train Shape:", X_train_scaled.shape)
print("Scaled Test Shape :", X_test_scaled.shape)
```

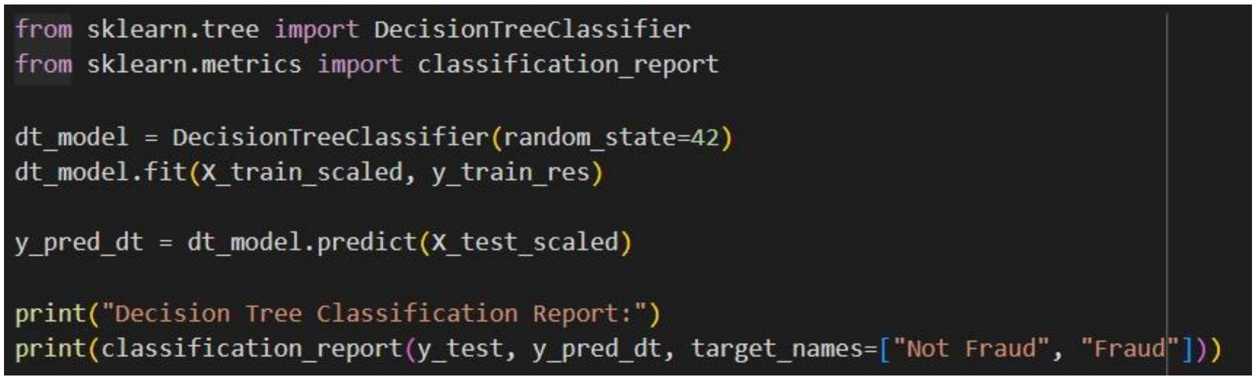
Milestone 4: Model Building

Training the model in multiple algorithms

Now our data is cleaned and it’s time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance.

Decision tree model

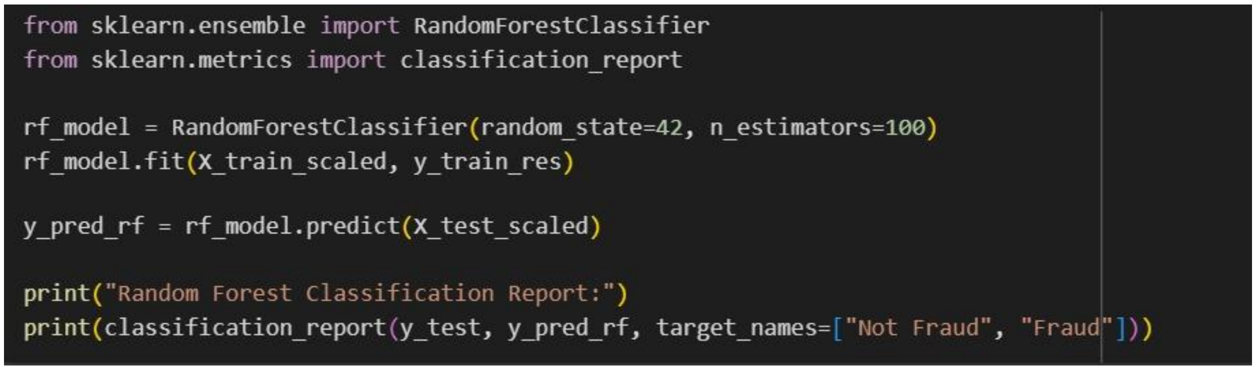
First Decision Tree is imported from sklearn Library then DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.



```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report

dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train_scaled, y_train_res)

y_pred_dt = dt_model.predict(X_test_scaled)

print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt, target_names=["Not Fraud", "Fraud"]))
```

Activity 1.2: Random forest model

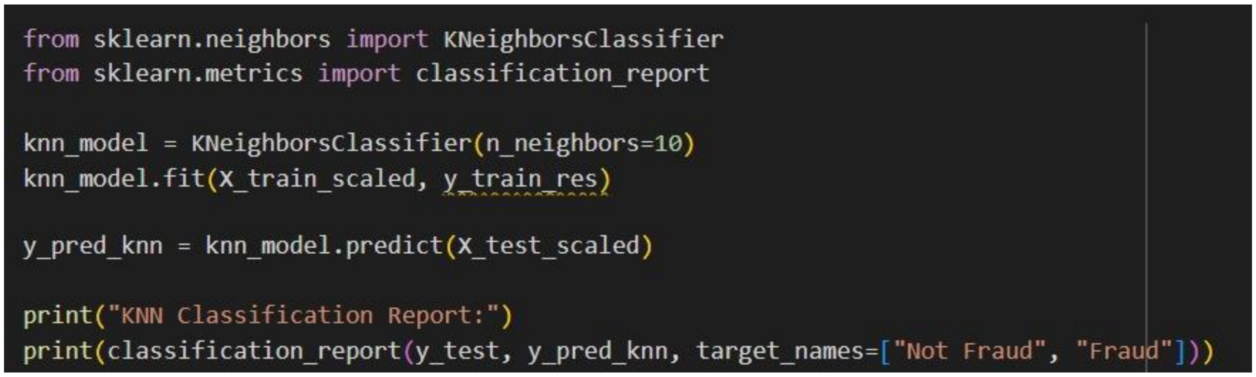
First Random Forest Model is imported from sklearn Library then RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.



```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

rf_model = RandomForestClassifier(random_state=42, n_estimators=100)
rf_model.fit(X_train_scaled, y_train_res)

y_pred_rf = rf_model.predict(X_test_scaled)

print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf, target_names=["Not Fraud", "Fraud"]))
```

KNN model

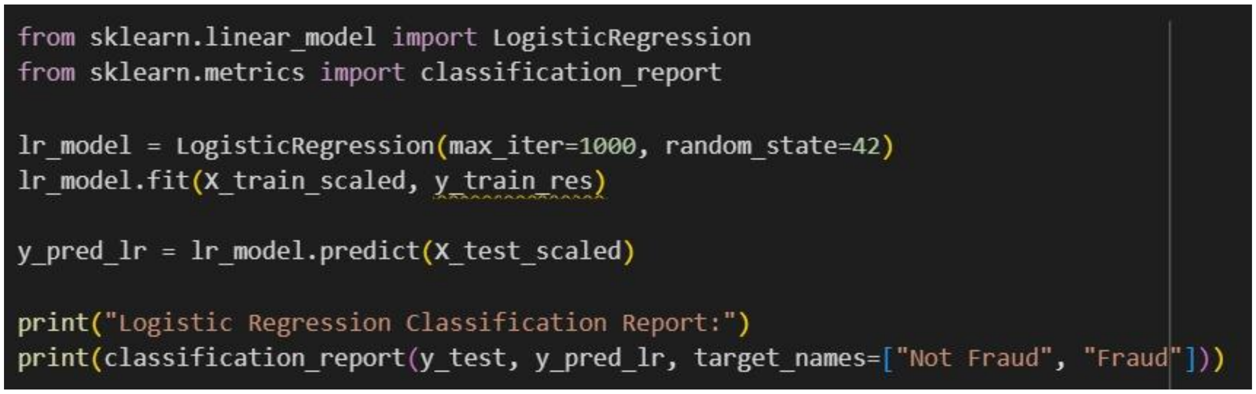
KNN Model is imported from sklearn Library then KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.



```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report

knn_model = KNeighborsClassifier(n_neighbors=10)
knn_model.fit(X_train_scaled, y_train_res)

y_pred_knn = knn_model.predict(X_test_scaled)

print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn, target_names=["Not Fraud", "Fraud"]))
```

Activity 1.4: Logistic Regression model

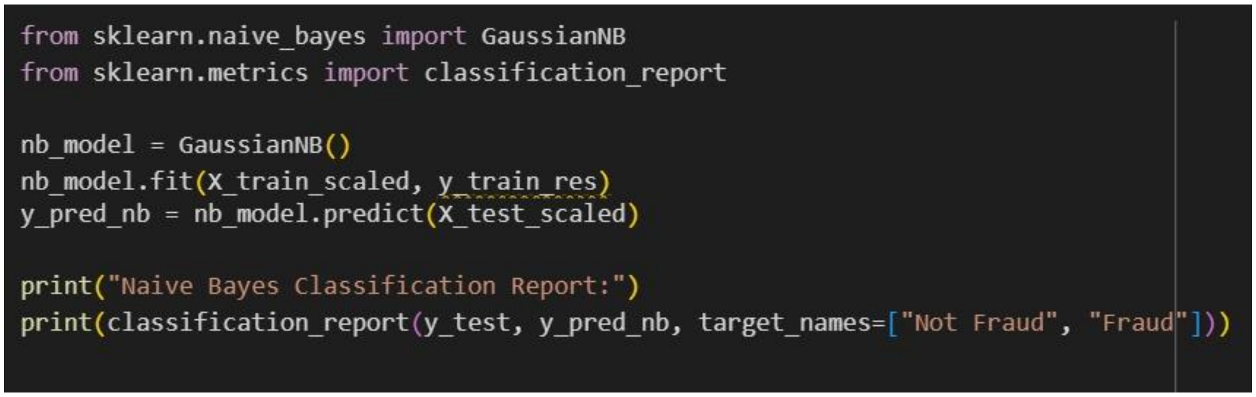
Logistic Regression Model is imported from sklearn Library then Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix is done.



```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

lr_model = LogisticRegression(max_iter=1000, random_state=42)
lr_model.fit(X_train_scaled, y_train_res)

y_pred_lr = lr_model.predict(X_test_scaled)

print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=["Not Fraud", "Fraud"]))
```

Activity 1.5: Naïve Bayes model

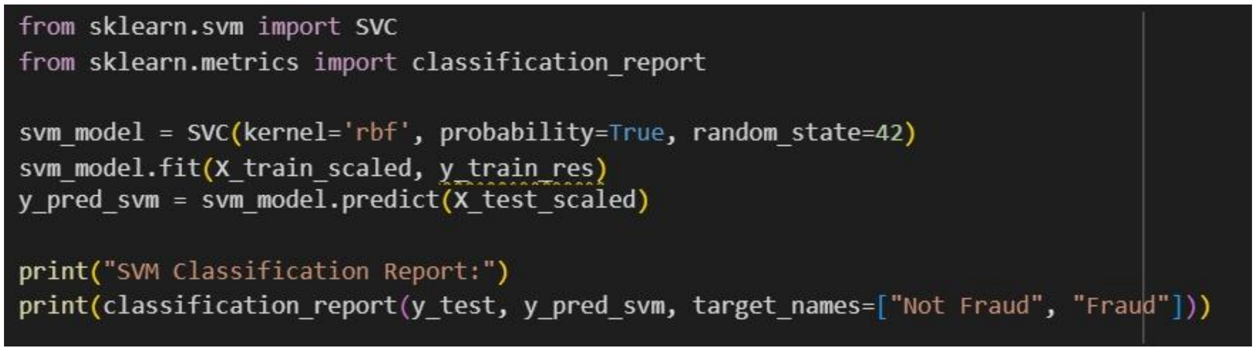
Naïve Bayes Model is imported from sklearn Library then Naïve Bayes algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. We can find the Train and Test accuracy by X_train and X_test.



```python
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report

nb_model = GaussianNB()
nb_model.fit(X_train_scaled, y_train_res)
y_pred_nb = nb_model.predict(X_test_scaled)

print("Naive Bayes Classification Report:")
print(classification_report(y_test, y_pred_nb, target_names=["Not Fraud", "Fraud"]))
```

Activity 1.6: SVM model

SVM Model is imported from sklearn Library then SVM algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.



```python
from sklearn.svm import SVC
from sklearn.metrics import classification_report

svm_model = SVC(kernel='rbf', probability=True, random_state=42)
svm_model.fit(X_train_scaled, y_train_res)
y_pred_svm = svm_model.predict(X_test_scaled)

print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=["Not Fraud", "Fraud"]))
```

Testing the model

Here we have tested with Decision Tree algorithm. You can test with all algorithm. With the help of predict() function.

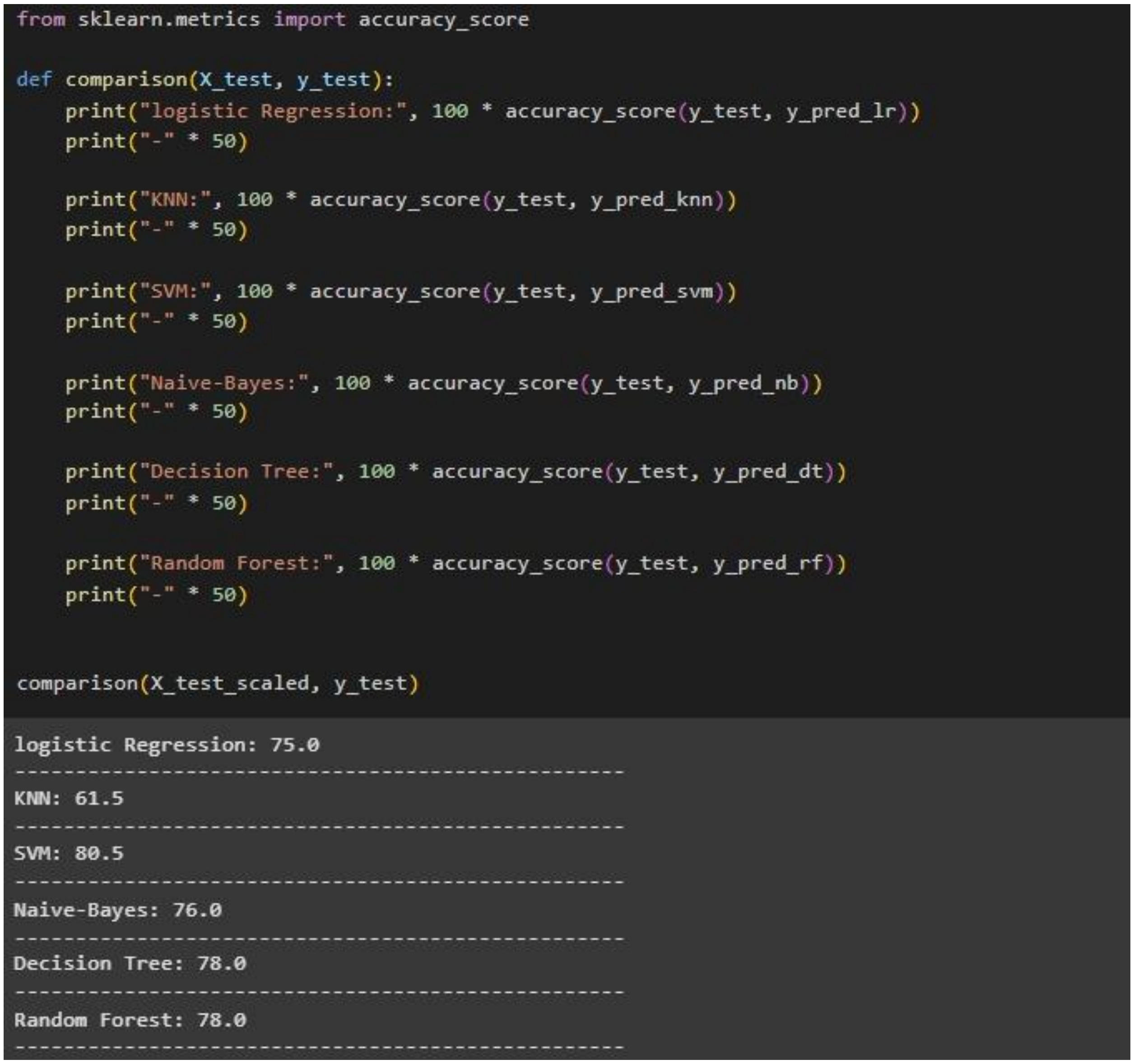
Milestone 5: Performance Testing & Hyperparameter Tuning

Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support .

Compare the model

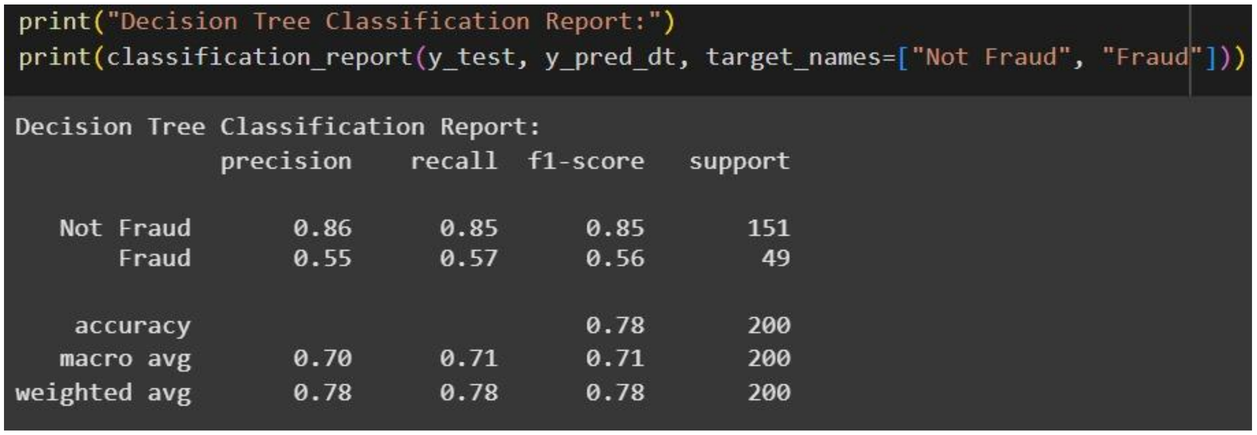
For comparing the above four models, the compareModel function is defined.



```python
from sklearn.metrics import accuracy_score

def comparison(X_test, y_test):
    print("logistic Regression:", 100 * accuracy_score(y_test, y_pred_lr))
    print("-" * 50)

    print("KNN:", 100 * accuracy_score(y_test, y_pred_knn))
    print("-" * 50)

    print("SVM:", 100 * accuracy_score(y_test, y_pred_svm))
    print("-" * 50)

    print("Naive-Bayes:", 100 * accuracy_score(y_test, y_pred_nb))
    print("-" * 50)

    print("Decision Tree:", 100 * accuracy_score(y_test, y_pred_dt))
    print("-" * 50)

    print("Random Forest:", 100 * accuracy_score(y_test, y_pred_rf))
    print("-" * 50)


comparison(X_test_scaled, y_test)
```

```
logistic Regression: 75.0
--------------------------------------------------
KNN: 61.5
--------------------------------------------------
SVM: 80.5
--------------------------------------------------
Naive-Bayes: 76.0
--------------------------------------------------
Decision Tree: 78.0
--------------------------------------------------
Random Forest: 78.0
--------------------------------------------------
```

1. **Decision Tree Classification Report**



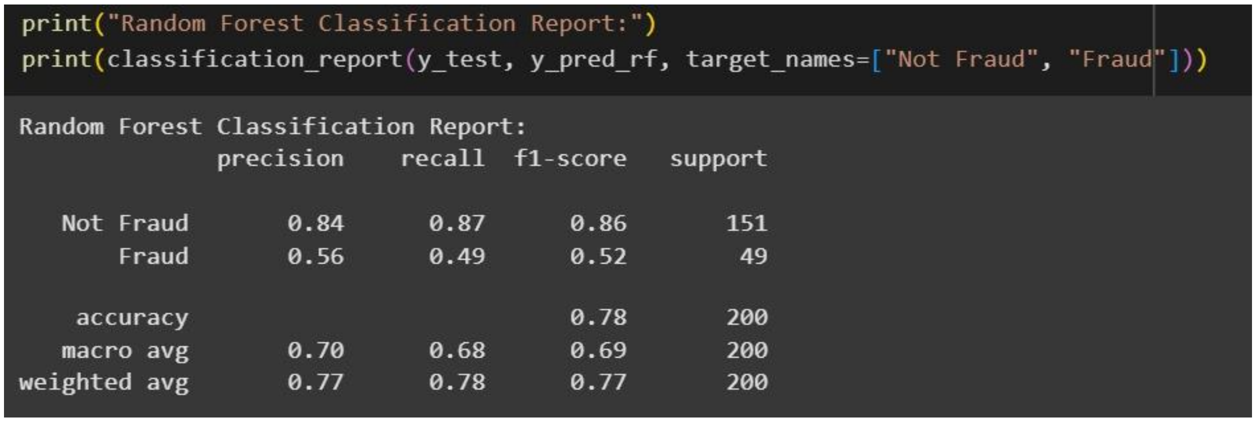
```
print("Decision Tree Classification Report:")
print(classification_report(y_test, y_pred_dt, target_names=["Not Fraud", "Fraud"]))
```

```
Decision Tree Classification Report:
              precision    recall  f1-score   support

   Not Fraud       0.86      0.85      0.85       151
       Fraud       0.55      0.57      0.56        49

    accuracy                           0.78       200
   macro avg       0.70      0.71      0.71       200
weighted avg       0.78      0.78      0.78       200
```

Explanation:

- Fraud class performance:
    - Precision: 0.55
    - Recall: 0.57
    - F1-score: 0.56
- Accuracy = 0.78
- **Conclusion:** Decent individual classifier, performs comparably to Random Forest but not better overall.

2. **Random Forest Classification Report**



```
print("Random Forest Classification Report:")
print(classification_report(y_test, y_pred_rf, target_names=["Not Fraud", "Fraud"]))
```

```
Random Forest Classification Report:
              precision    recall  f1-score   support

   Not Fraud       0.84      0.87      0.86       151
       Fraud       0.56      0.49      0.52        49

    accuracy                           0.78       200
   macro avg       0.70      0.68      0.69       200
weighted avg       0.77      0.78      0.77       200
```

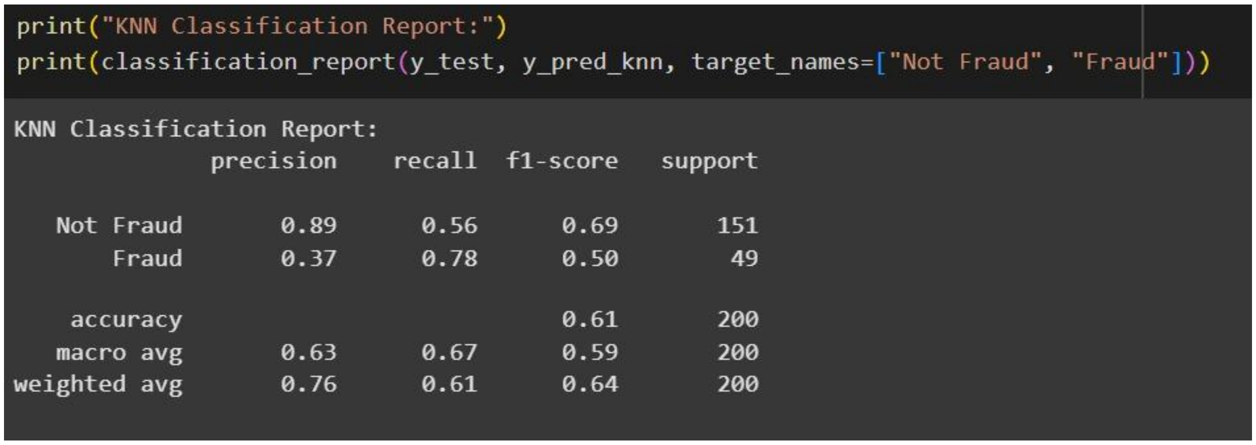
Explanation:

- **Fraud Class:**

- Precision: 0.56
- Recall: 0.49
- F1-score: 0.52

- Accuracy = 0.78
- **Conclusion:** Balanced but not exceptional. However, in **ROC/AUC**, Random Forest shines (best overall), suggesting it ranks frauds well even if threshold-based performance is average.

3. KNN Classification Report



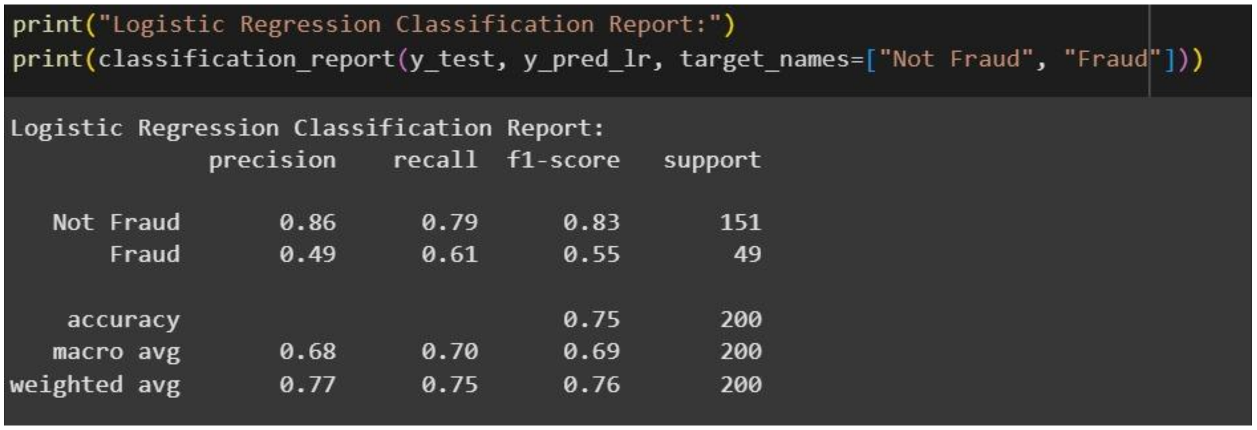
```
print("KNN Classification Report:")
print(classification_report(y_test, y_pred_knn, target_names=["Not Fraud", "Fraud"]))
```

```
KNN Classification Report:
              precision    recall  f1-score   support

   Not Fraud       0.89      0.56      0.69       151
       Fraud       0.37      0.78      0.50        49

    accuracy                           0.61       200
   macro avg       0.63      0.67      0.59       200
weighted avg       0.76      0.61      0.64       200
```

Explanation:

- Performs **poorly**:
  - Accuracy = **0.61**
  - Fraud precision = **0.37**, recall = **0.78**
- High recall but very low precision: it flags lots of frauds, most of which are incorrect.
- **Conclusion:** High false positive rate makes KNN impractical for fraud detection.

4. Logistic Regression Classification Report

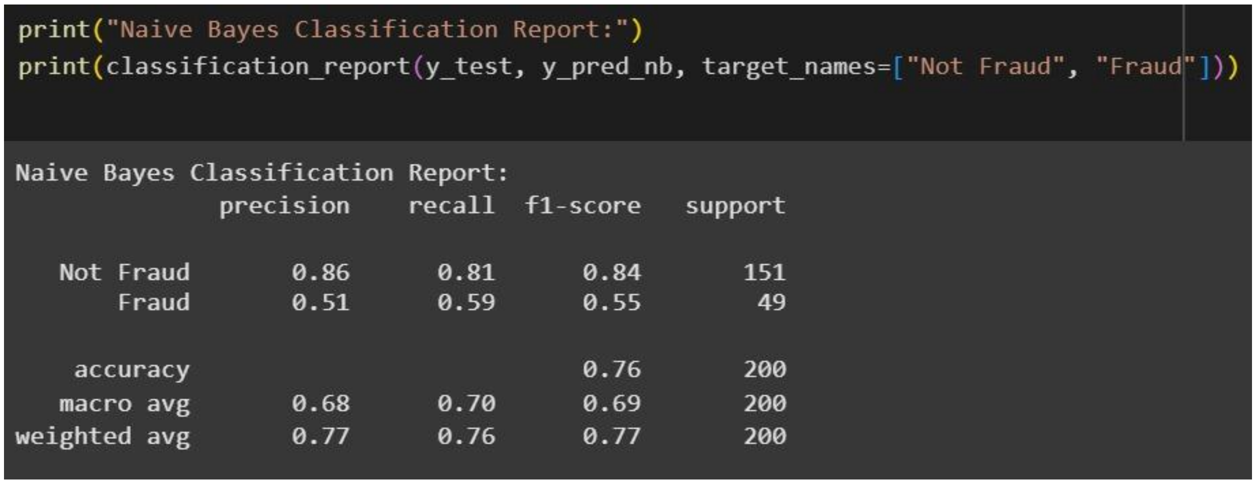

```
print("Logistic Regression Classification Report:")
print(classification_report(y_test, y_pred_lr, target_names=["Not Fraud", "Fraud"]))
```

```
Logistic Regression Classification Report:
              precision    recall  f1-score   support

   Not Fraud       0.86      0.79      0.83       151
       Fraud       0.49      0.61      0.55        49

    accuracy                           0.75       200
   macro avg       0.68      0.70      0.69       200
weighted avg       0.77      0.75      0.76       200
```

Explanation:

- **Fraud class recall is 0.61** (better than SVM, NB)
- But **precision is only 0.49**
- F1-score = 0.55
- **Conclusion:** Logistic Regression finds more frauds but also has more false positives, hence low precision.

5. **Naive Bayes Classification Report**


```
print("Naive Bayes Classification Report:")
print(classification_report(y_test, y_pred_nb, target_names=["Not Fraud", "Fraud"]))
```

```
Naive Bayes Classification Report:
              precision    recall  f1-score   support

   Not Fraud       0.86      0.81      0.84       151
       Fraud       0.51      0.59      0.55        49

    accuracy                           0.76       200
   macro avg       0.68      0.70      0.69       200
weighted avg       0.77      0.76      0.77       200
```

Explanation:

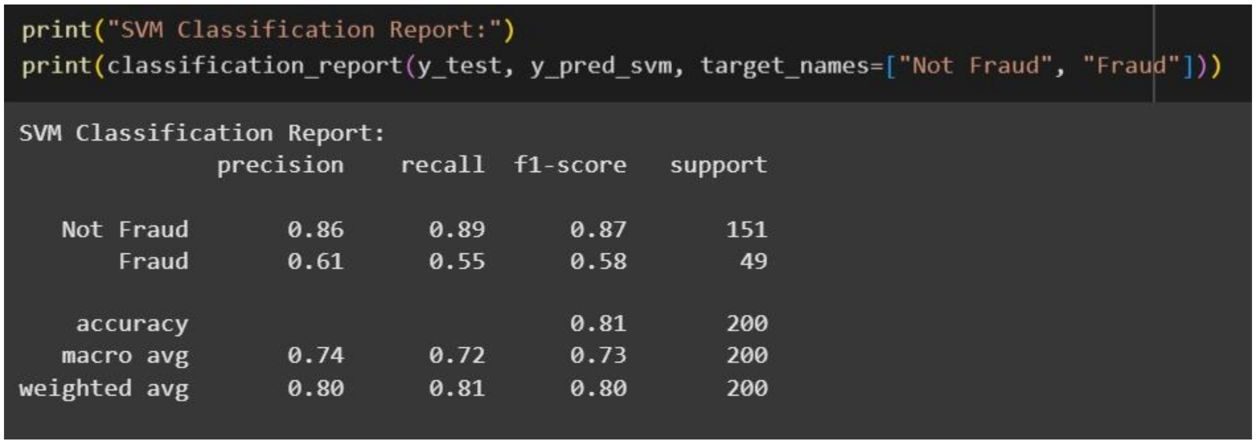
- **Fraud Class:**
  - Precision: 0.51
  - Recall: 0.59

    - F1-score: 0.55

- Accuracy = 0.76 (lower than SVM)

- **Conclusion:** Slightly weaker than SVM on fraud detection, but still decent. Works fast but lacks the complexity to capture relationships.

6. SVM Classification Report



```
print("SVM Classification Report:")
print(classification_report(y_test, y_pred_svm, target_names=["Not Fraud", "Fraud"]))
```

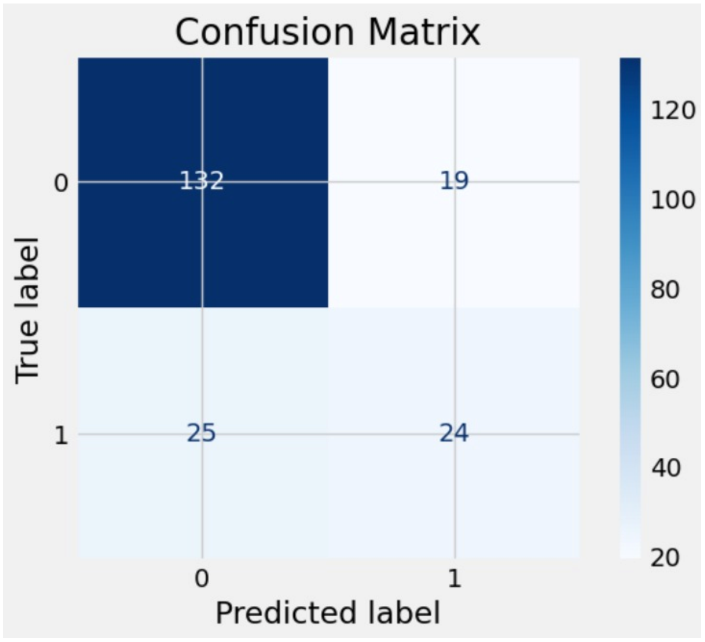
```
SVM Classification Report:
              precision    recall  f1-score   support

   Not Fraud       0.86      0.89      0.87       151
       Fraud       0.61      0.55      0.58        49

    accuracy                           0.81       200
   macro avg       0.74      0.72      0.73       200
weighted avg       0.80      0.81      0.80       200
```

Explanation:

- Performance of **SVM** on test set:
    - **Not Fraud class:**
        - Precision: 0.86
        - Recall: 0.89
        - F1-score: 0.87
    - **Fraud class:**
        - Precision: 0.61
        - Recall: 0.55
        - F1-score: 0.58
- **Overall accuracy:** 0.81
- **Macro average F1-score:** 0.73

- **Conclusion:** SVM has strong performance on the majority class, and moderate results on the fraud class.

Confusion Matrix

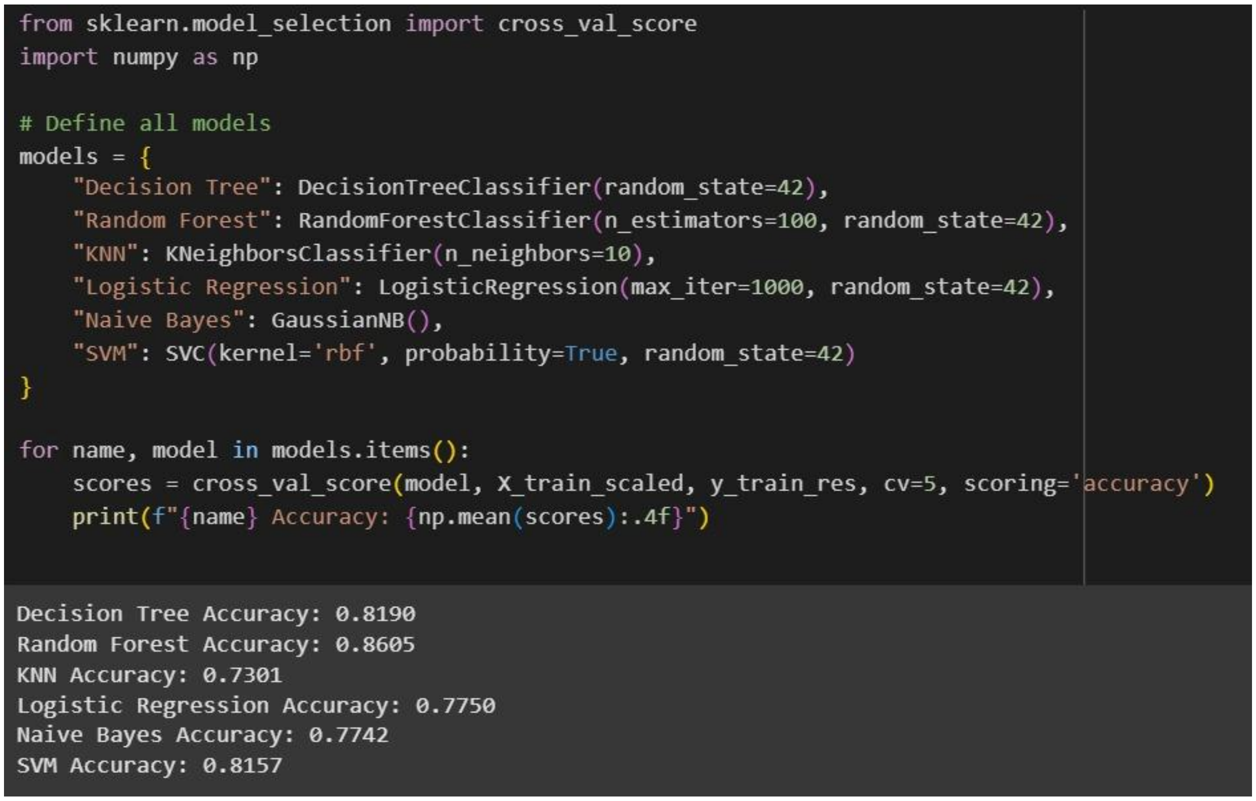


Explanation:

- Based on predictions made by **SVM model** (from the next screenshot).
- Matrix:
  - **True Negatives (TN):** 132
  - **False Positives (FP):** 19
  - **False Negatives (FN):** 25
  - **True Positives (TP):** 24
- **Interpretation:**
  - **Fraud Detection Recall:** 24 / (24 + 25) = **~0.49**
  - Indicates SVM missed 25 frauds but caught 24 — decent balance.

- Majority class is "Not Fraud", which is typical of real-world fraud data (class imbalance).

Accuracy Comparison via Cross-Validation



```
from sklearn.model_selection import cross_val_score
import numpy as np

# Define all models
models = {
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=10),
    "Logistic Regression": LogisticRegression(max_iter=1000, random_state=42),
    "Naive Bayes": GaussianNB(),
    "SVM": SVC(kernel='rbf', probability=True, random_state=42)
}

for name, model in models.items():
    scores = cross_val_score(model, X_train_scaled, y_train_res, cv=5, scoring='accuracy')
    print(f"{name} Accuracy: {np.mean(scores):.4f}")
```

```
Decision Tree Accuracy: 0.8190
Random Forest Accuracy: 0.8605
KNN Accuracy: 0.7301
Logistic Regression Accuracy: 0.7750
Naive Bayes Accuracy: 0.7742
SVM Accuracy: 0.8157
```

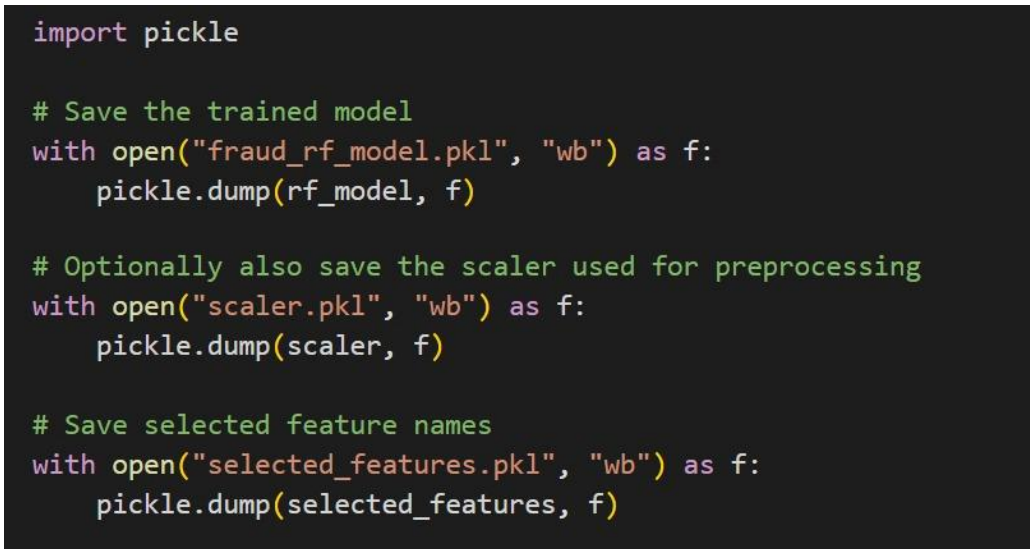
Explanation:

- Code shows 5-fold cross-validation using cross_val_score on six models.
- Model accuracies:
    - **Random Forest:** 0.8605 Highest
    - **Decision Tree:** 0.8190
    - **SVM:** 0.8157
    - **Logistic Regression:** 0.7750
    - **Naive Bayes:** 0.7742
    - **KNN:** 0.7301 Lowest

- **Conclusion:** Random Forest had the best average accuracy, reinforcing what was seen in the ROC curve.

Milestone 6: Model Deployment

Activity 1: Save the best model



```python
import pickle

# Save the trained model
with open("fraud_rf_model.pkl", "wb") as f:
    pickle.dump(rf_model, f)

# Optionally also save the scaler used for preprocessing
with open("scaler.pkl", "wb") as f:
    pickle.dump(scaler, f)

# Save selected feature names
with open("selected_features.pkl", "wb") as f:
    pickle.dump(selected_features, f)
```

Integrate with Web Framework

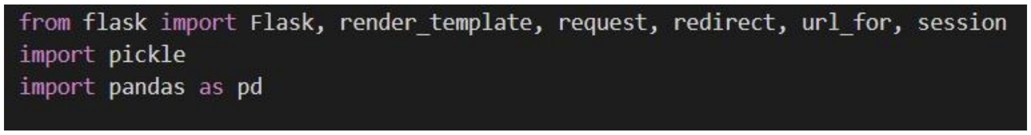
In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI. This section has the following tasks

● Building HTML Pages

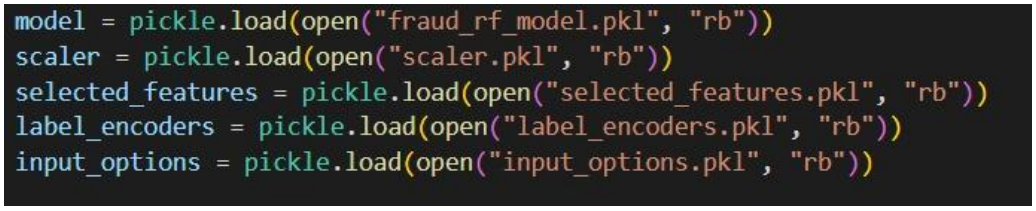
● Building server-side script

● Run the web application

Build python code



```python
from flask import Flask, render_template, request, redirect, url_for, session
import pickle
import pandas as pd
```

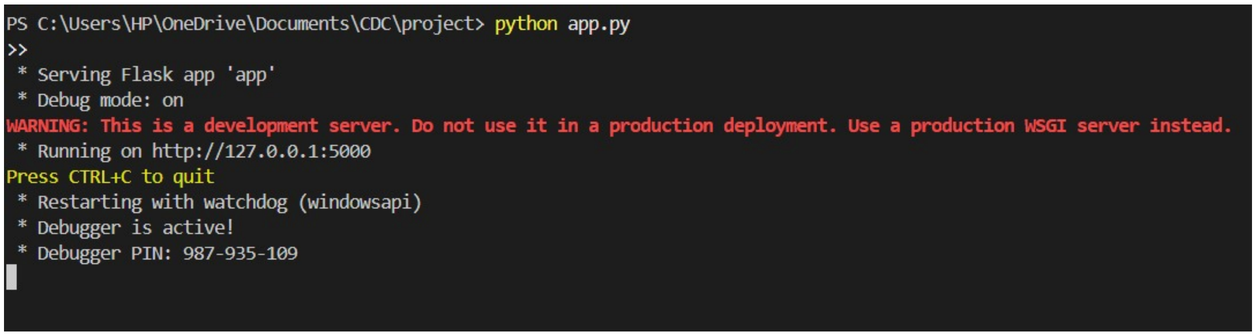

```python
model = pickle.load(open("fraud_rf_model.pkl", "rb"))
scaler = pickle.load(open("scaler.pkl", "rb"))
selected_features = pickle.load(open("selected_features.pkl", "rb"))
label_encoders = pickle.load(open("label_encoders.pkl", "rb"))
input_options = pickle.load(open("input_options.pkl", "rb"))
```

This code saves important components of a machine learning pipeline using
Python's pickle module:

- Trained model: Stores the trained Random Forest model for future predictions.
- Scaler: Saves the preprocessing scaler to ensure new data is transformed the same way as training data.
- Selected features: Keeps the list of feature names to maintain consistency in input data structure.

This allows you to reuse the model and preprocessing steps without retraining.

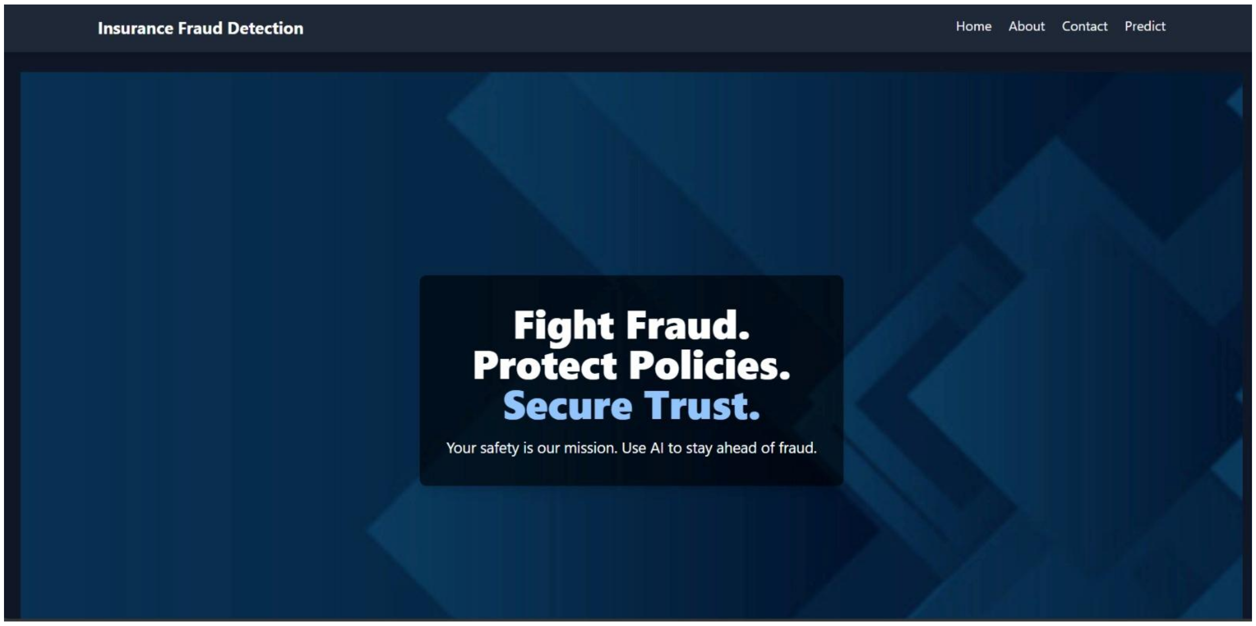
Flask Development Server Startup


```
PS C:\Users\HP\OneDrive\Documents\CDC\project> python app.py
>>
 * Serving Flask app 'app'
 * Debug mode: on
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
 * Restarting with watchdog (windowsapi)
 * Debugger is active!
 * Debugger PIN: 987-935-109
```

- This message shows that your Flask web application is running in development mode.
- The server is accessible at http://127.0.0.1:5000 on your local machine.
- The warning in red reminds you not to use this server for production; it is only for testing and development.
- Debug mode is enabled, allowing automatic reloading and debugging features for easier development.

Prediction Output and Explanation Panel

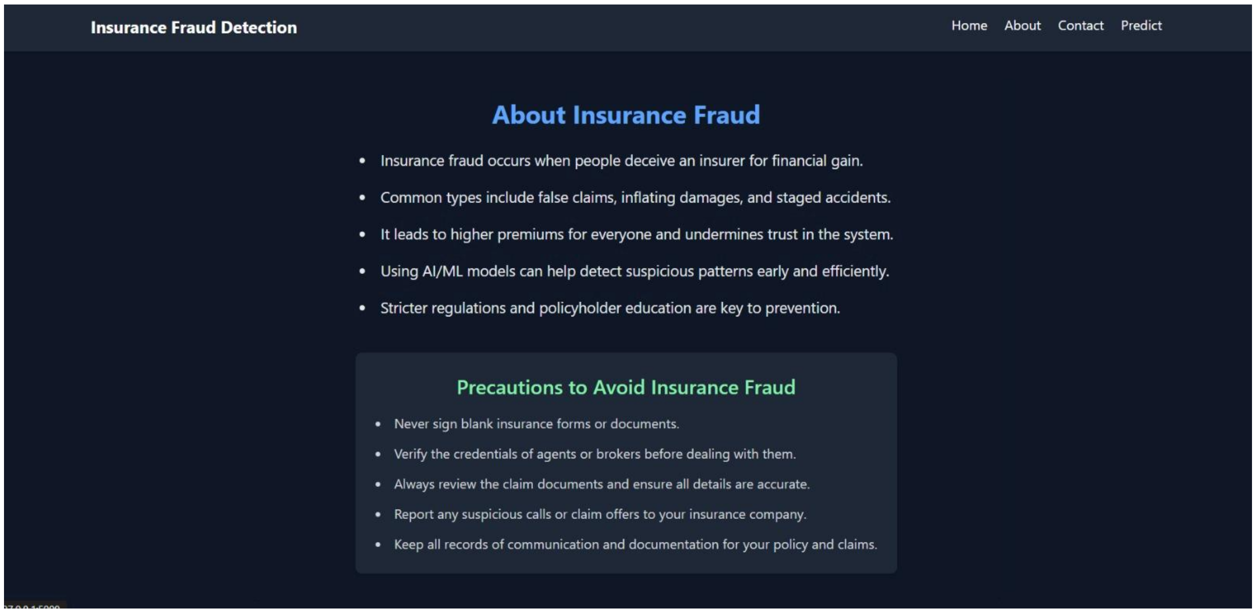
Home Page



Fight Fraud.
Protect Policies.
Secure Trust.

Your safety is our mission. Use AI to stay ahead of fraud.

About Page



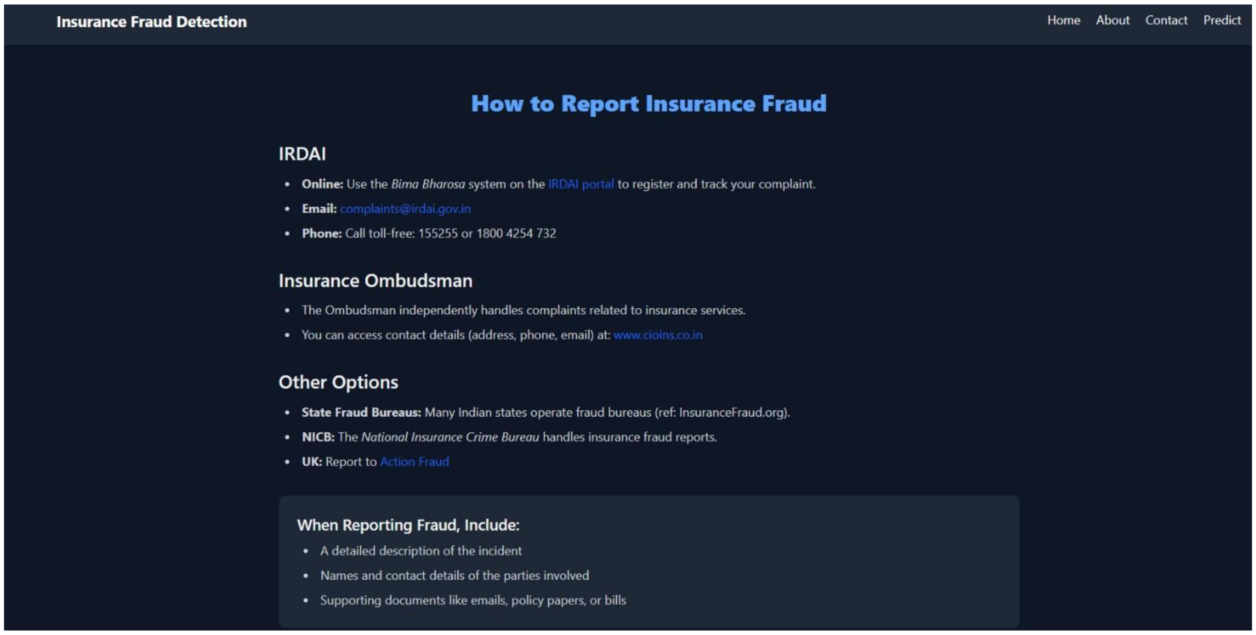
About Insurance Fraud

- Insurance fraud occurs when people deceive an insurer for financial gain.
- Common types include false claims, inflating damages, and staged accidents.
- It leads to higher premiums for everyone and undermines trust in the system.
- Using AI/ML models can help detect suspicious patterns early and efficiently.
- Stricter regulations and policyholder education are key to prevention.

Precautions to Avoid Insurance Fraud

- Never sign blank insurance forms or documents.
- Verify the credentials of agents or brokers before dealing with them.
- Always review the claim documents and ensure all details are accurate.
- Report any suspicious calls or claim offers to your insurance company.
- Keep all records of communication and documentation for your policy and claims.

Contact Page


Insurance Fraud Detection

Home About Contact Predict

How to Report Insurance Fraud

IRDAI

- **Online:** Use the *Bima Bharosa* system on the IRDAI portal to register and track your complaint.
- **Email:** complaints@irdai.gov.in
- **Phone:** Call toll-free: 155255 or 1800 4254 732

Insurance Ombudsman

- The Ombudsman independently handles complaints related to insurance services.
- You can access contact details (address, phone, email) at: www.cioins.co.in

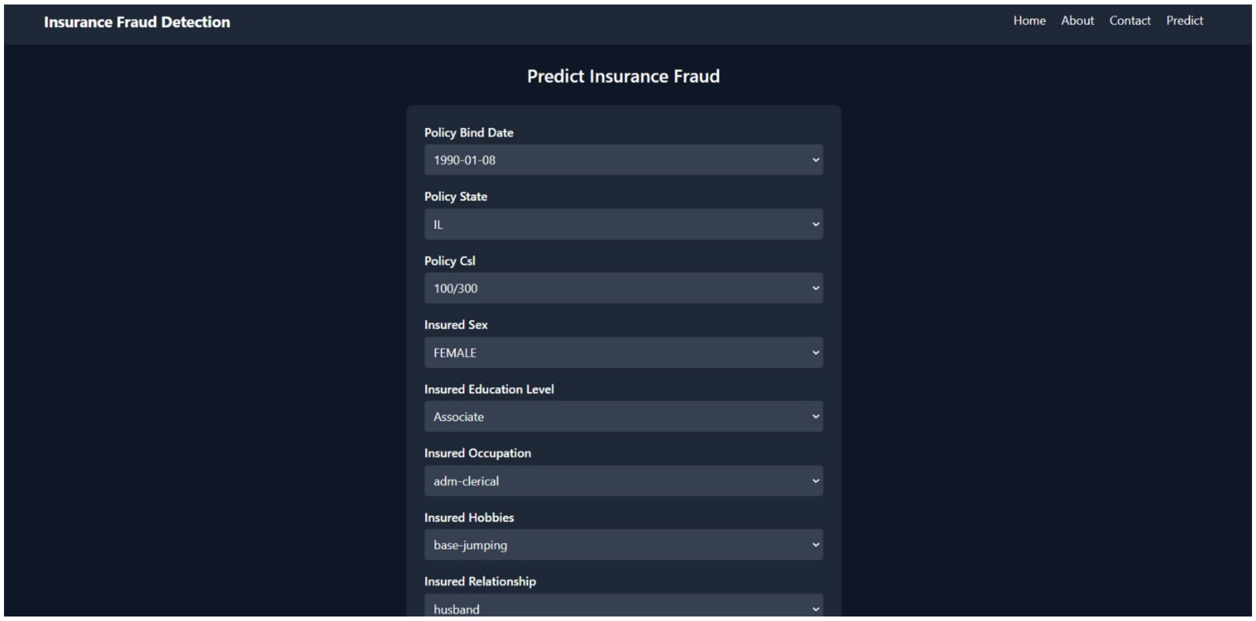
Other Options

- **State Fraud Bureaus:** Many Indian states operate fraud bureaus (ref: InsuranceFraud.org).
- **NICB:** The *National Insurance Crime Bureau* handles insurance fraud reports.
- **UK:** Report to Action Fraud

When Reporting Fraud, Include:

- A detailed description of the incident
- Names and contact details of the parties involved
- Supporting documents like emails, policy papers, or bills

Predict Page

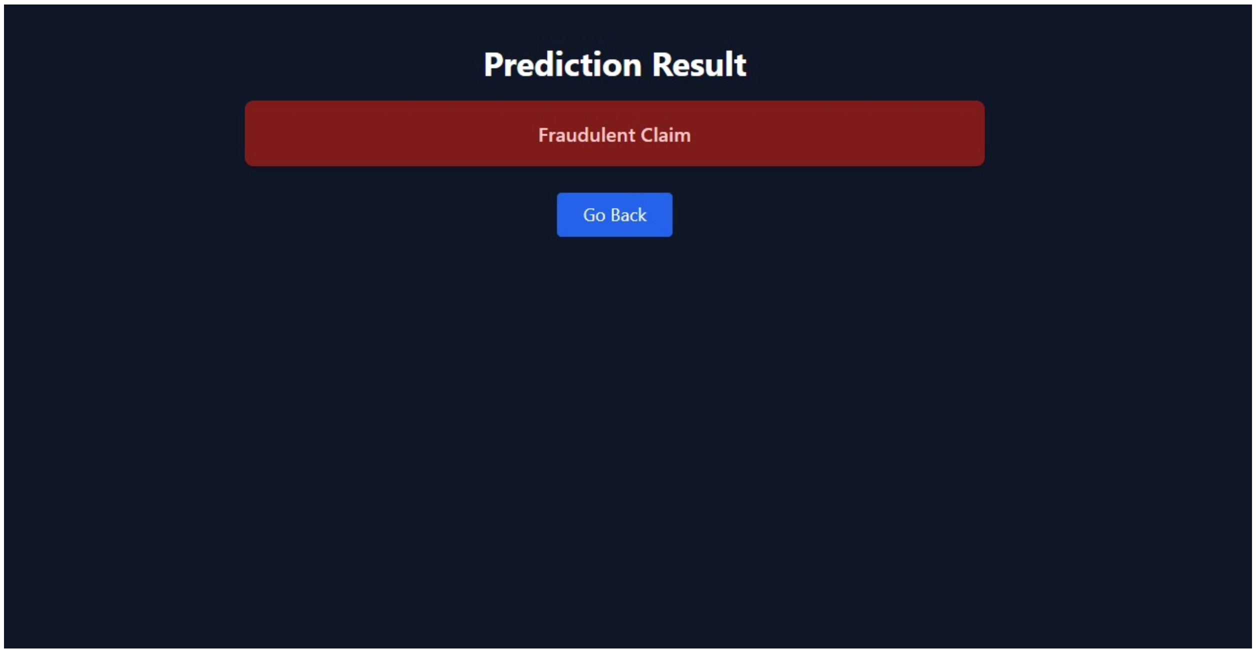

Insurance Fraud Detection

Home About Contact Predict

Predict Insurance Fraud

Policy Bind Date	1990-01-08
Policy State	IL
Policy Csl	100/300
Insured Sex	FEMALE
Insured Education Level	Associate
Insured Occupation	adm-clerical
Insured Hobbies	base-jumping
Insured Relationship	husband

Result Page


Prediction Result
Fraudulent Claim
Go Back