# Fraud Detection Analysis in Insurance

This project aims to analyze insurance claim data to identify potential fraud patterns and build a machine learning model capable of detecting fraudulent claims.

Insurance fraud represents a significant financial risk for insurance companies. Detecting suspicious claims early can help reduce financial losses and improve operational efficiency.

The objectives of this project are:

```
- Explore the dataset and identify fraud patterns

- Perform feature engineering to extract useful signals

- Train several machine learning models

- Optimize the best-performing model

- Identify the most important fraud indicators
```

# Dataset Overview

Dataset Description

The dataset contains 1000 insurance claims with 40 features, including:

```
- Policy information

- Customer demographic attributes

- Accident details

- Claim amounts

- Fraud indicator (fraud_reported)
```

The target variable is:

fraud_reported ( 0 → No Fraud, 1 → Fraud)

Data Types

The dataset contains a mix of:

```
- Numerical variables

- Categorical variables

- Date features
```

# import libraries

In [3]:
```
import pandas as pd
import numpy as np


import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression, LinearRegression
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from xgboost import XGBClassifier, XGBRegressor

from sklearn.metrics import classification_report, roc_auc_score, confusion_matrix,
from sklearn.model_selection import GridSearchCV
import time
```

In [4]:
```
df = pd.read_csv(r"C:\Users\Imene MESSAADI\Documents\Projets_Portfolio\Assurances\c
```

In [5]:
```
df.head()
```

Out[5]:

| | months_as_customer | age | policy_number | policy_bind_date | policy_state | policy_csl | policy_dedu |
|---|---|---|---|---|---|---|---|
| **0** | 328 | 48 | 521585 | 2014-10-17 | OH | 250/500 | |
| **1** | 228 | 42 | 342868 | 2006-06-27 | IN | 250/500 | |
| **2** | 134 | 29 | 687698 | 2000-09-06 | OH | 100/300 | |
| **3** | 256 | 41 | 227811 | 1990-05-25 | IL | 250/500 | |
| **4** | 228 | 44 | 367455 | 2014-06-06 | IL | 500/1000 | |

5 rows × 40 columns

# Explore the data

In [6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           1000 non-null   int64  
 1   age                          1000 non-null   int64  
 2   policy_number                1000 non-null   int64  
 3   policy_bind_date             1000 non-null   object 
 4   policy_state                 1000 non-null   object 
 5   policy_csl                   1000 non-null   object 
 6   policy_deductable            1000 non-null   int64  
 7   policy_annual_premium        1000 non-null   float64
 8   umbrella_limit               1000 non-null   int64  
 9   insured_zip                  1000 non-null   int64  
 10  insured_sex                  1000 non-null   object 
 11  insured_education_level      1000 non-null   object 
 12  insured_occupation           1000 non-null   object 
 13  insured_hobbies              1000 non-null   object 
 14  insured_relationship         1000 non-null   object 
 15  capital-gains                1000 non-null   int64  
 16  capital-loss                 1000 non-null   int64  
 17  incident_date                1000 non-null   object 
 18  incident_type                1000 non-null   object 
 19  collision_type               1000 non-null   object 
 20  incident_severity            1000 non-null   object 
 21  authorities_contacted        909 non-null    object 
 22  incident_state               1000 non-null   object 
 23  incident_city                1000 non-null   object 
 24  incident_location            1000 non-null   object 
 25  incident_hour_of_the_day     1000 non-null   int64  
 26  number_of_vehicles_involved  1000 non-null   int64  
 27  property_damage              1000 non-null   object 
 28  bodily_injuries              1000 non-null   int64  
 29  witnesses                    1000 non-null   int64  
 30  police_report_available      1000 non-null   object 
 31  total_claim_amount           1000 non-null   int64  
 32  injury_claim                 1000 non-null   int64  
 33  property_claim               1000 non-null   int64  
 34  vehicle_claim                1000 non-null   int64  
 35  auto_make                    1000 non-null   object 
 36  auto_model                   1000 non-null   object 
 37  auto_year                    1000 non-null   int64  
 38  fraud_reported               1000 non-null   object 
 39  _c39                         0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 312.6+ KB
```

In [7]:

```
df.describe()
```

Out[7]:

| | months_as_customer | age | policy_number | policy_deductable | policy_annual_premium |
|---|---|---|---|---|---|
| **count** | 1000.000000 | 1000.000000 | 1000.000000 | 1000.000000 | 1000.00000( |
| **mean** | 203.954000 | 38.948000 | 546238.648000 | 1136.000000 | 1256.40615( |
| **std** | 115.113174 | 9.140287 | 257063.005276 | 611.864673 | 244.16739! |
| **min** | 0.000000 | 19.000000 | 100804.000000 | 500.000000 | 433.33000( |
| **25%** | 115.750000 | 32.000000 | 335980.250000 | 500.000000 | 1089.60750( |
| **50%** | 199.500000 | 38.000000 | 533135.000000 | 1000.000000 | 1257.20000( |
| **75%** | 276.250000 | 44.000000 | 759099.750000 | 2000.000000 | 1415.69500( |
| **max** | 479.000000 | 64.000000 | 999435.000000 | 2000.000000 | 2047.59000( |

In [8]:

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

Out[8]:

```
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
```

In [9]:

```
df.shape
```

Out[9]:
```
(1000, 40)
```

# Data Cleaning

The column _c39 contained only missing values so i can drop it

In [10]:
```
df = df.drop(columns=['_c39'])
```

In [11]:
```
#df.columns
```

### Handling missing values

Some rows contained missing values in the 'authorities_contacted' column. This represents less than 5% of the dataset, those rows can be removed.

In [12]:
```
df.dropna(inplace= True)
```

### Handdling Inconsistent values

Some categorical colums contained "?" representing missing information. so i decided to replace it with 'Unknown' (in the next steps )

# Exploratory Data Analysis (EDA)

In [13]:
```
df['fraud_reported'].value_counts()
```

Out[13]:
```
fraud_reported
N    668
Y    241
Name: count, dtype: int64
```

In [14]:
```
df.duplicated().sum()
```

Out[14]:
```
0
```

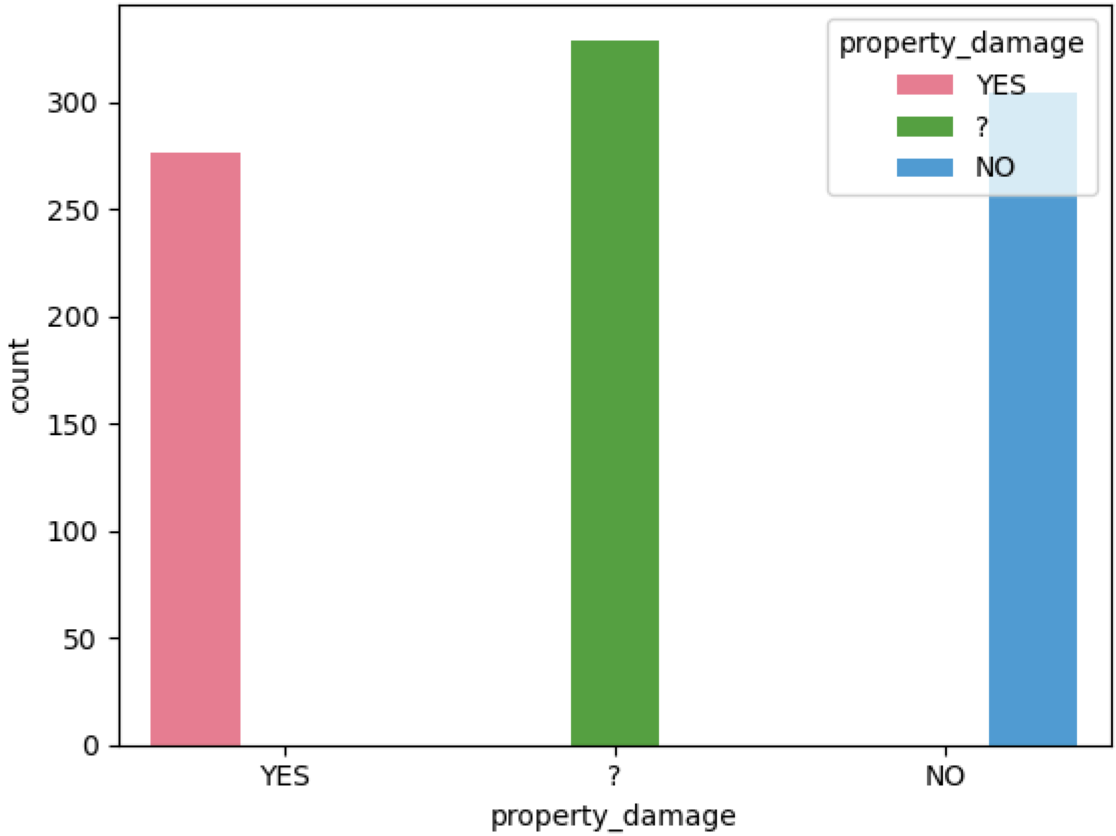
In [15]:
```
df.isnull().sum()
```

Out[15]:
```
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          0
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
dtype: int64
```

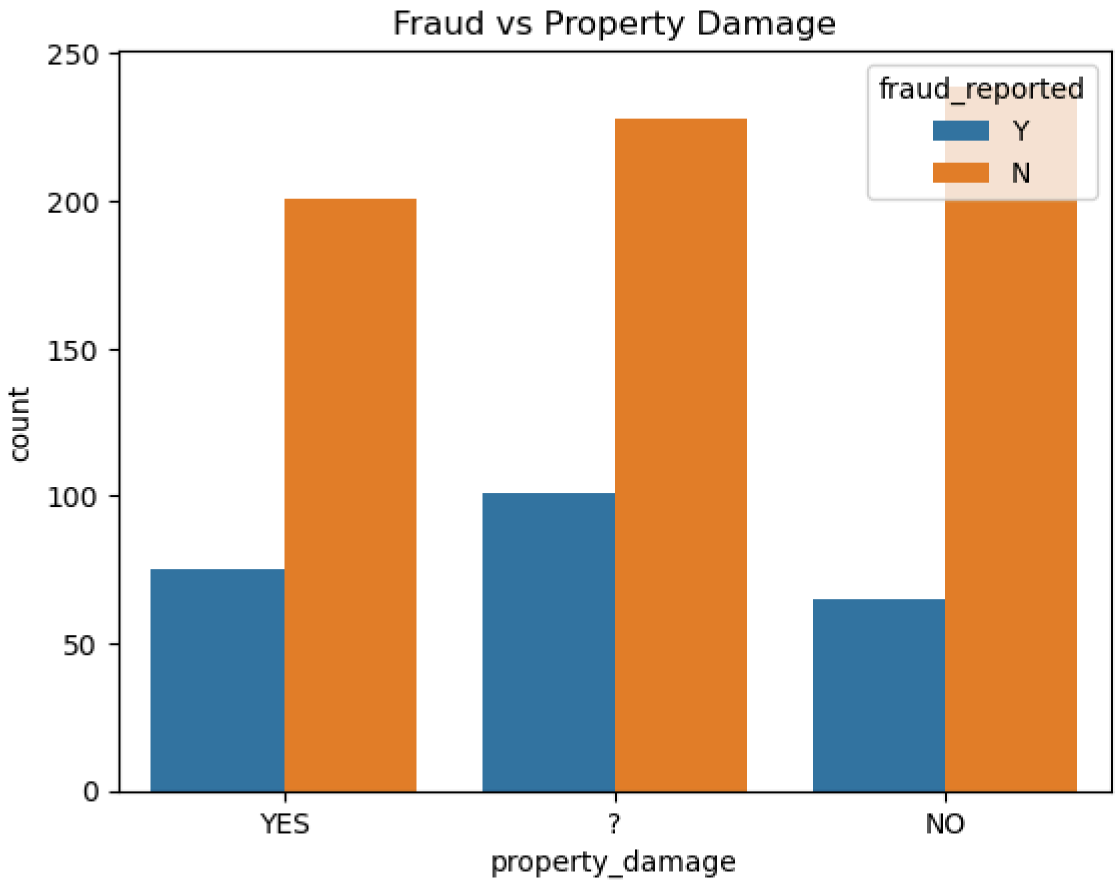
In [16]:
```
df['property_damage'].value_counts()
```

Out[16]:
```
property_damage
?      329
NO     304
YES    276
Name: count, dtype: int64
```

In [17]:
```
sns.countplot(x='property_damage', data =df, hue='property_damage', palette='husl')
plt.show()
```




In [18]:
```
sns.countplot(x='property_damage', data = df, hue = 'fraud_reported')
plt.title("Fraud vs Property Damage")
plt.show()
```

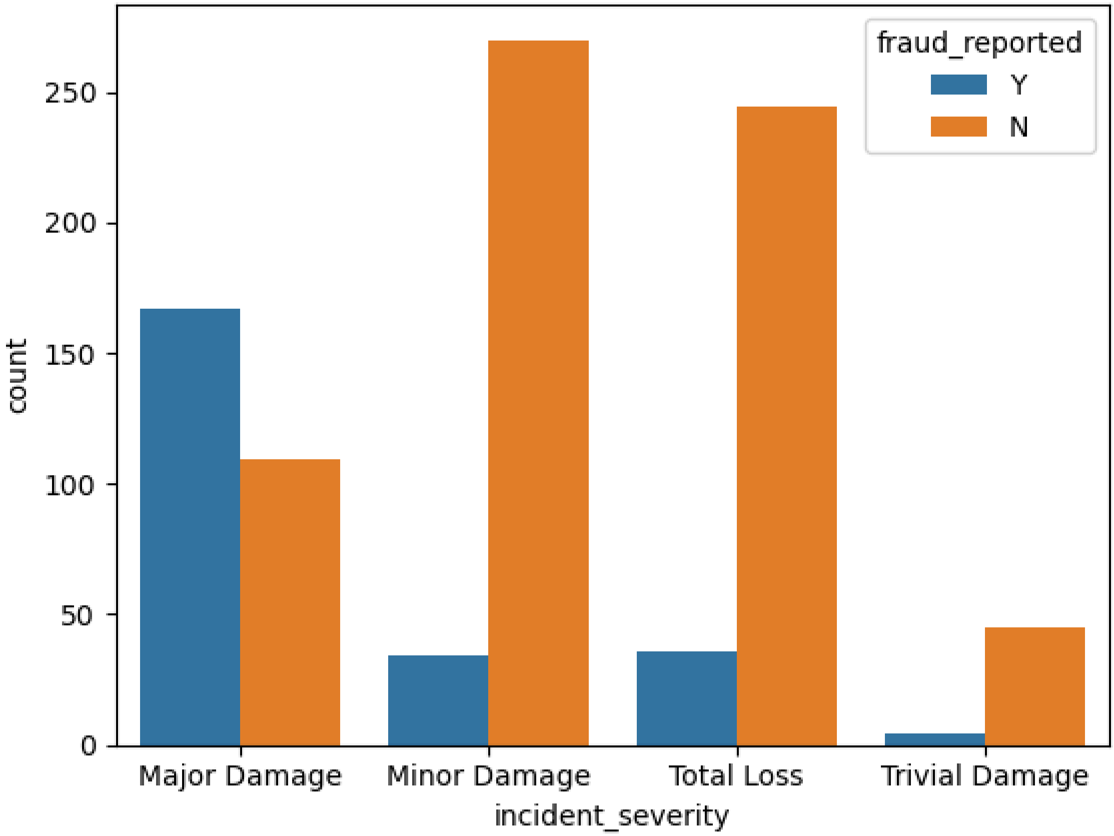



In [19]:
```
fraud_rate = pd.crosstab(df['property_damage'], df['fraud_reported'], normalize='ir
print(fraud_rate)
```

```
fraud_reported          N         Y
property_damage
?                0.693009  0.306991
NO               0.786184  0.213816
YES              0.728261  0.271739
```

In [20]:
```
df['property_damage'] = df['property_damage'].replace('?', 'Unknown')
```

In [21]:
```
sns.countplot(x='incident_severity', hue='fraud_reported', data=df)
plt.show()
```




Incidents categorized as Major Damage have the highest fraud rate. so this suggests that some claims may exagerate the severity of the incident.

In [22]:
```
pd.crosstab(df['incident_severity'], df['fraud_reported'], normalize='index')
```

Out[22]:

| **fraud_reported** | **N** | **Y** |
|---|---|---|
| **incident_severity** | | |
| **Major Damage** | 0.394928 | 0.605072 |
| **Minor Damage** | 0.888158 | 0.111842 |
| **Total Loss** | 0.871429 | 0.128571 |
| **Trivial Damage** | 0.918367 | 0.081633 |

i'm seeing that the major damage incidents have very high fraud probability(61%) whiwh is very intersting, cause claims with major damage severity show significantly higher fraud compared to minor or trivial incidents.

fraudulent claims are more frequently associated with major damga incidents, so i suggest possible exageration of claim severity.

So i want to create a fraud risk score feature to transform the catogorical business information into risk number score.

In [23]:
```
pd.crosstab(df['collision_type'], df['fraud_reported'], normalize='index')
```

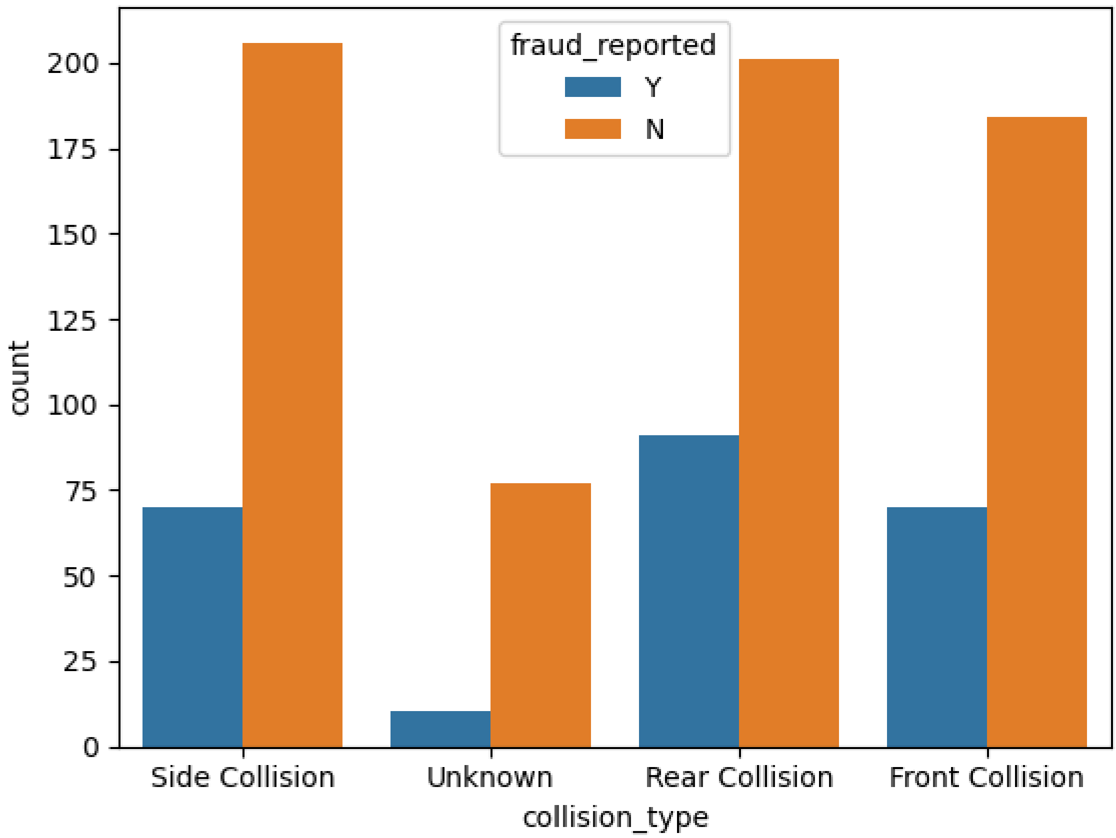
Out[23]:

| fraud_reported / collision_type | N | Y |
|---|---|---|
| ? | 0.885057 | 0.114943 |
| Front Collision | 0.724409 | 0.275591 |
| Rear Collision | 0.688356 | 0.311644 |
| Side Collision | 0.746377 | 0.253623 |

i can see that the rear collision has the highest fraud probability which is interesting in insurance for several reasons

In [24]:
```
df['collision_type'] = df['collision_type'].replace('?', 'Unknown')
```

In [25]:
```
sns.countplot(x='collision_type', hue='fraud_reported', data=df)

plt.show()
```




Rear collision shows the highest fraud probability

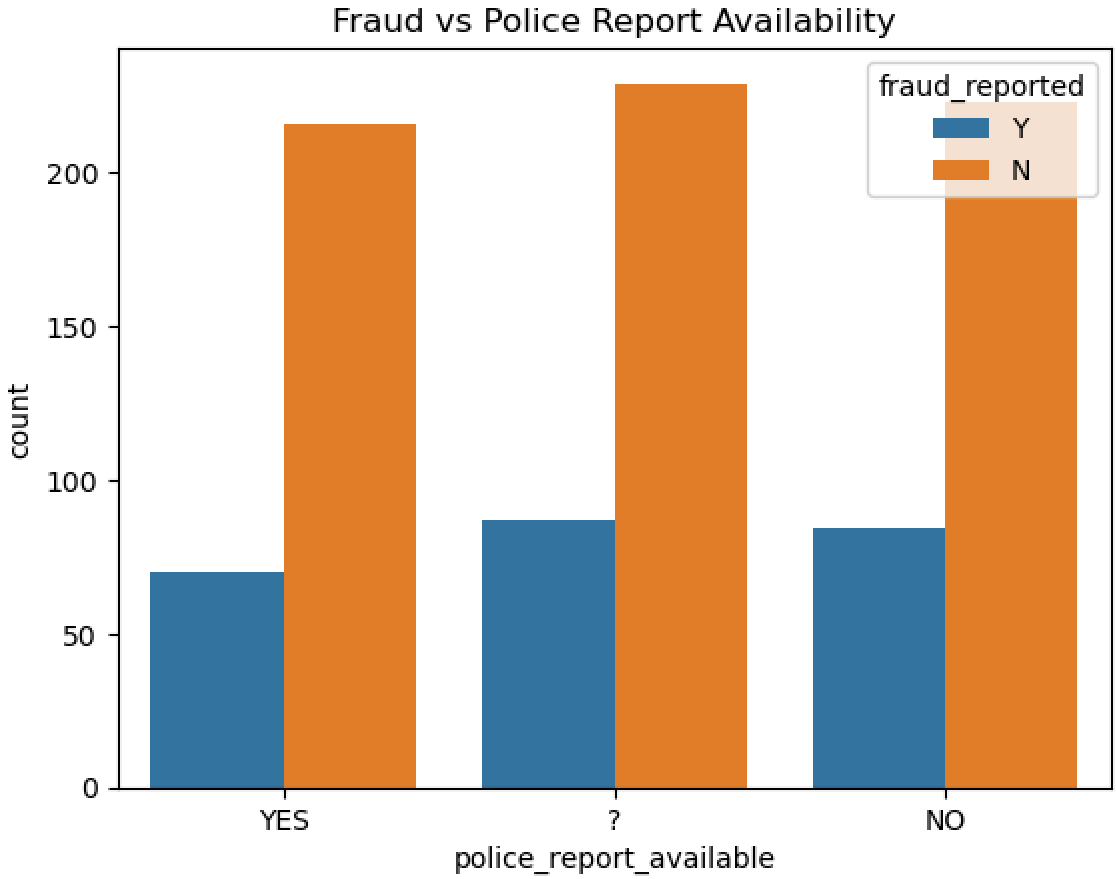
this can be explained by the fact that this kind of collisions are often difficult to verify. making them easier to manipulate.

In [26]:
```
pd.crosstab(df['police_report_available'], df['fraud_reported'], normalize='index')
```

Out[26]:

| fraud_reported | N | Y |
|---|---|---|
| **police_report_available** | | |
| **?** | 0.724684 | 0.275316 |
| **NO** | 0.726384 | 0.273616 |
| **YES** | 0.755245 | 0.244755 |

In [27]:
```
sns.countplot(x='police_report_available', hue='fraud_reported', data=df)
plt.title("Fraud vs Police Report Availability")
plt.show()
```




fraudrate is slightly higher when police report is missing which is a classic fraud pattern.

when the report is available, the fraud rate decrease.

In [28]:
```
df['police_report_available'] = df['police_report_available'].replace('?', 'Unknowr
```

Now i'm creating some features for my project :

In [29]:
```
severity_map = {
    "Trivial Damage" : 1,
    "Minor Damage" : 2,
    "Major Damage" : 3,
    "Total Loss" :4
}
df['severity_score'] = df['incident_severity'].map(severity_map)
```

In [30]:
```
df['claim_ratio'] = df['total_claim_amount'] / df['policy_annual_premium']
df['night_incident'] = df['incident_hour_of_the_day'].apply ( lambda x:1 if x<=6 el
#accidents occuring late at night may have less witness, potentially increasing fra
```

```
df['high_severity']= df['severity_score'].apply( lambda x:1 if x in [3,4] else 0)
df['vehicle_age']= 2026 - df['auto_year']
```

# Correlation

In [31]:
```
corr = df.corr(numeric_only=True)
corr
```

Out[31]:

| | months_as_customer | age | policy_number | policy_deductable | po |
|---|---|---|---|---|---|
| **months_as_customer** | 1.000000 | 0.922209 | 0.054543 | 0.031039 | |
| **age** | 0.922209 | 1.000000 | 0.048465 | 0.032037 | |
| **policy_number** | 0.054543 | 0.048465 | 1.000000 | -0.018363 | |
| **policy_deductable** | 0.031039 | 0.032037 | -0.018363 | 1.000000 | |
| **policy_annual_premium** | 0.003161 | 0.006252 | 0.016241 | -0.010063 | |
| **umbrella_limit** | 0.014856 | 0.008767 | -0.006924 | -0.006726 | |
| **insured_zip** | 0.018316 | 0.024681 | 0.014248 | -0.007881 | |
| **capital-gains** | 0.011235 | -0.001925 | -0.010942 | 0.024653 | |
| **capital-loss** | 0.012649 | -0.005044 | -0.007304 | -0.018671 | |
| **incident_hour_of_the_day** | 0.078276 | 0.092387 | 0.004124 | 0.057806 | |
| **number_of_vehicles_involved** | 0.007600 | 0.014846 | 0.015190 | 0.062274 | |
| **bodily_injuries** | -0.004542 | -0.009031 | -0.006029 | -0.019147 | |
| **witnesses** | 0.047183 | 0.042181 | -0.004871 | 0.078426 | |
| **total_claim_amount** | 0.057304 | 0.065127 | -0.021827 | 0.045397 | |
| **injury_claim** | 0.060052 | 0.070513 | -0.009260 | 0.056627 | |
| **property_claim** | 0.025795 | 0.054234 | -0.011157 | 0.086800 | |
| **vehicle_claim** | 0.056035 | 0.056061 | -0.024836 | 0.023231 | |
| **auto_year** | -0.003662 | -0.003420 | 0.013944 | 0.020655 | |
| **severity_score** | -0.021664 | -0.014637 | -0.033594 | 0.024606 | |
| **claim_ratio** | 0.034825 | 0.035071 | -0.031419 | 0.038919 | |
| **night_incident** | -0.100398 | -0.102788 | -0.028303 | -0.046517 | |
| **high_severity** | -0.014101 | -0.004823 | -0.024752 | 0.018108 | |
| **vehicle_age** | 0.003662 | 0.003420 | -0.013944 | -0.020655 | |

23 rows × 23 columns

In [32]:
```
upper = corr.where(np.triu(np.ones(corr.shape), k=1).astype(bool))
```

In [33]:
```
upper
```

Out[33]:

| | months_as_customer | age | policy_number | policy_deductable | pol |
|---|---|---|---|---|---|
| **months_as_customer** | NaN | 0.922209 | 0.054543 | 0.031039 | |
| **age** | NaN | NaN | 0.048465 | 0.032037 | |
| **policy_number** | NaN | NaN | NaN | -0.018363 | |
| **policy_deductable** | NaN | NaN | NaN | NaN | |
| **policy_annual_premium** | NaN | NaN | NaN | NaN | |
| **umbrella_limit** | NaN | NaN | NaN | NaN | |
| **insured_zip** | NaN | NaN | NaN | NaN | |
| **capital-gains** | NaN | NaN | NaN | NaN | |
| **capital-loss** | NaN | NaN | NaN | NaN | |
| **incident_hour_of_the_day** | NaN | NaN | NaN | NaN | |
| **number_of_vehicles_involved** | NaN | NaN | NaN | NaN | |
| **bodily_injuries** | NaN | NaN | NaN | NaN | |
| **witnesses** | NaN | NaN | NaN | NaN | |
| **total_claim_amount** | NaN | NaN | NaN | NaN | |
| **injury_claim** | NaN | NaN | NaN | NaN | |
| **property_claim** | NaN | NaN | NaN | NaN | |
| **vehicle_claim** | NaN | NaN | NaN | NaN | |
| **auto_year** | NaN | NaN | NaN | NaN | |
| **severity_score** | NaN | NaN | NaN | NaN | |
| **claim_ratio** | NaN | NaN | NaN | NaN | |
| **night_incident** | NaN | NaN | NaN | NaN | |
| **high_severity** | NaN | NaN | NaN | NaN | |
| **vehicle_age** | NaN | NaN | NaN | NaN | |

23 rows × 23 columns

i can see a strong correlation between some columns. For certain variables, the correlation is strong due to redundancy, such as 'age' and 'months_as_customer', which is consistent since the older an individual is, the more likely they are to have been a customer for a longer period.

also a strong correlation between 'vehicle_age' and 'auto_year', which is normal since the vehicle age is derived from the auto_year column. The same applies to the two variables 'incident_hour_of_the_day' and 'night_incident'.

next, there is also a strong correlation between 'total_claim_amount' and 'claim_ratio', because the calculation of one is derived from the other. Additionally, 'total_claim_amount' is highly correlated with 'vehicle_claim', 'injury_claim', and 'property_claim', which makes sense since total_claim_amount is supposed to be the sum of these three columns ===> indicating redundancy.

same case for 'high_severity' and 'severity_score' there is a redundancy

In such situations, it is preferable to remove or filter redundant variables to avoid multicollinearity, especially for linear models such as logistic regression.

However, the choice of which variables to remove should be guided by business understanding (it is essential to grasp the company's needs), the logic behind how the variables were constructed, and the type of model being used. For these aspects, human intelligence is particularly crucial.like tree-based models such as Random Forest or XGBoost, multicollinearity is generally less of an issue.

to start, i decided to remove the 'total_claim_amount' column, which is essentially just the sum of the three other claim columns. This is particularly useful because a fraud can affect only one of the three claims, not all three simultaneously. I also remove the 'age' column, since 'months_as_customer' seems more relevant for fraud detection, reflecting customer loyalty and tenure.

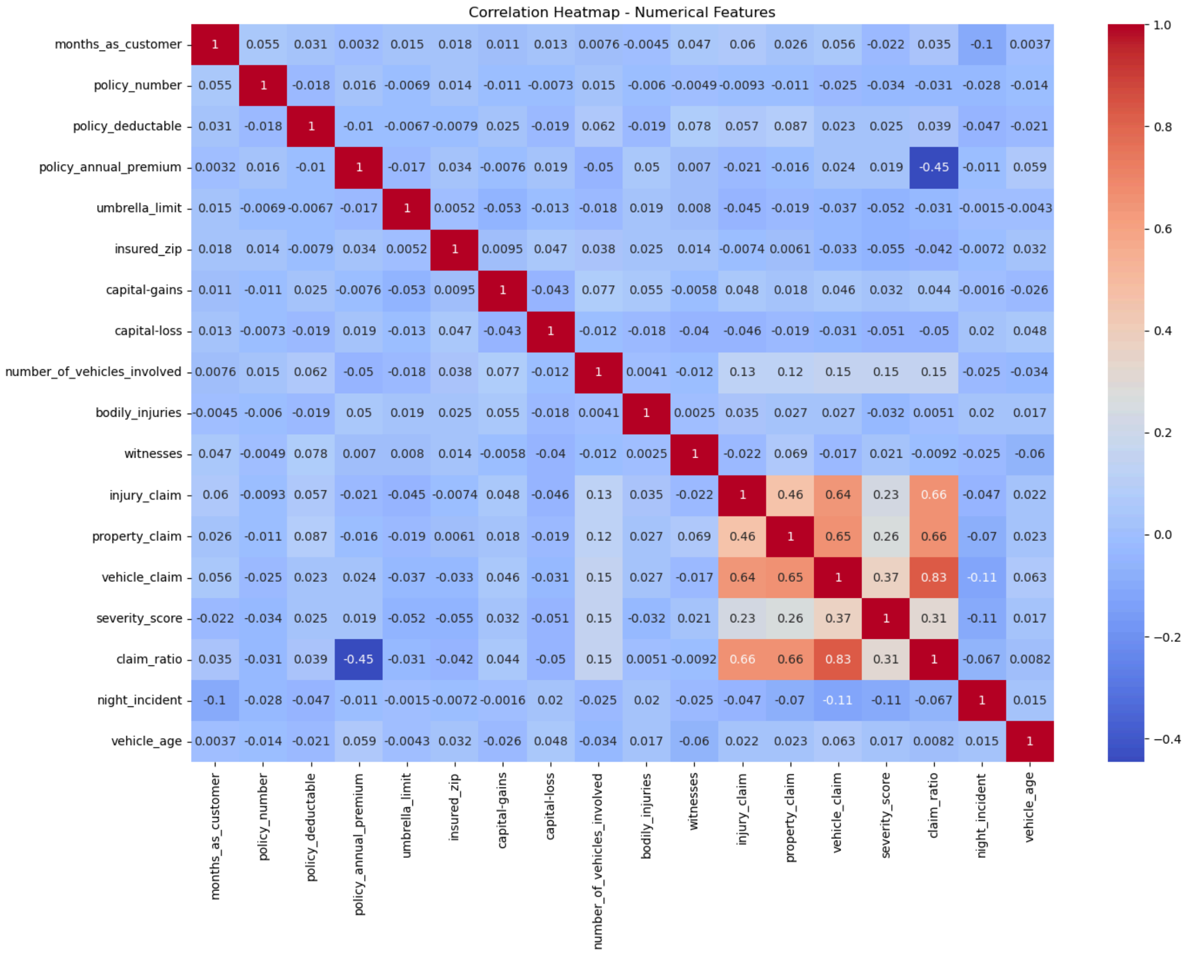
to delete too 'auto_year', 'incident_hour_of_the_day' et 'high_severity'

In [34]:
```
df = df.drop(columns = ['total_claim_amount', 'age','auto_year', 'high_severity', '
```

In [35]:
```
#df.columns
```

i decided to create a heatmap for better visualization. It is perfectly normal to observe a strong correlation between 'claim_ratio', 'vehicle_claim', 'injury_claim', and 'property_claim', as previously explained. However, I cannot remove any of these three variables because they are important for this project.

In [36]:
```
plt.figure(figsize=(16,11))
sns.heatmap(df.corr(numeric_only=True), cmap='coolwarm', annot= True)
plt.title("Correlation Heatmap - Numerical Features")
plt.show()
```

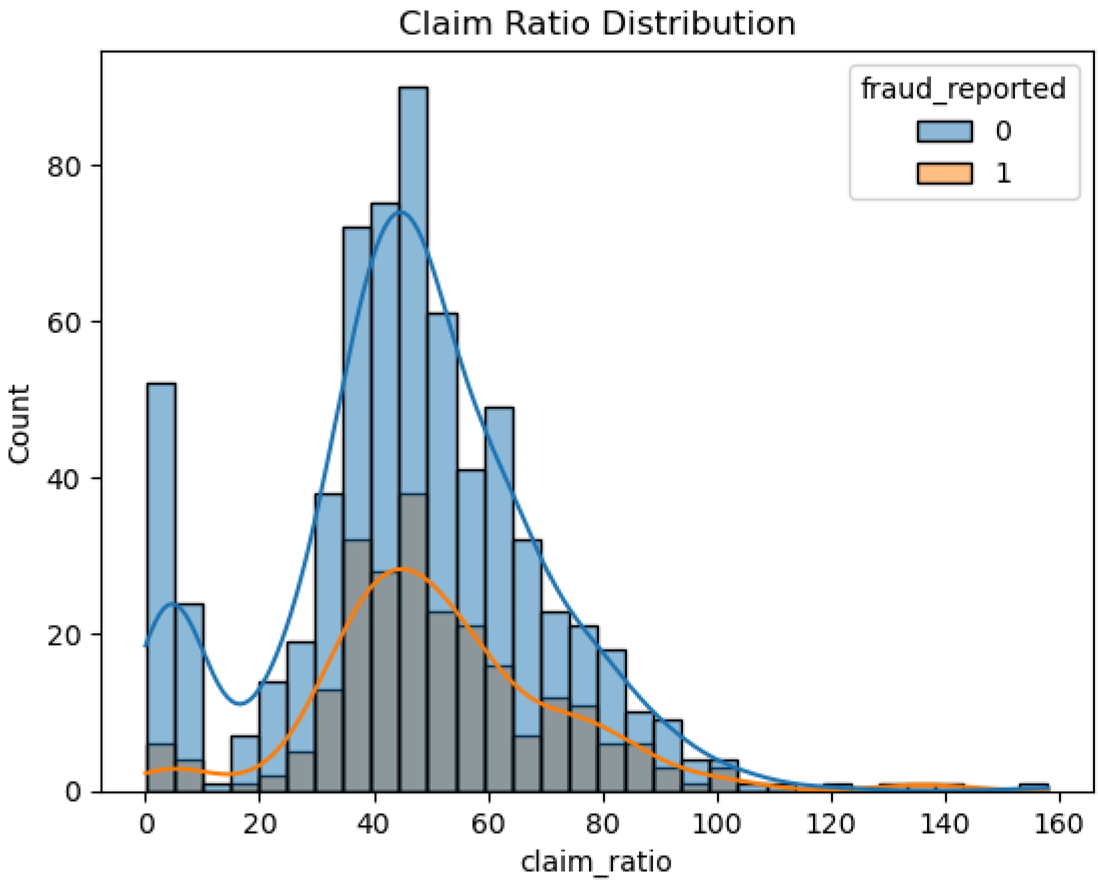



'severity_score' et 'claim_rati'o à 0.31 : plus la sévérité est élevée, plus le ratio grimpe

In [37]:
```
df['fraud_reported'] = df['fraud_reported'].map({'N':0,  'Y':1})
```

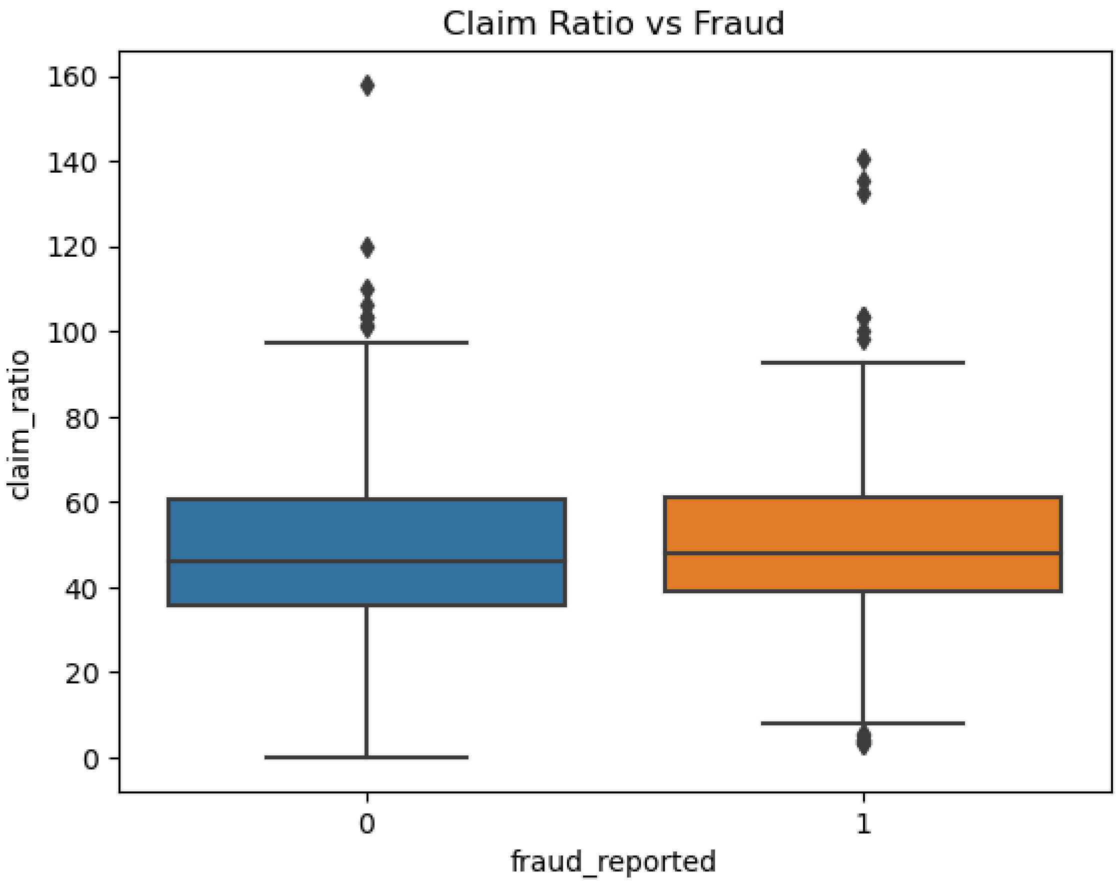
In [38]:
```
sns.histplot(data =df , x='claim_ratio',hue='fraud_reported', kde=True)
plt.title("Claim Ratio Distribution")
plt.show()
```

```
C:\Apps\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na o
ption is deprecated and will be removed in a future version. Convert inf values to
NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Apps\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping w
ith a length-1 list-like, you will need to pass a length-1 tuple to get_group in a
future version of pandas. Pass `(name,)` instead of `name` to silence this warnin
g.
  data_subset = grouped_data.get_group(pd_key)
C:\Apps\Lib\site-packages\seaborn\_oldcore.py:1075: FutureWarning: When grouping w
ith a length-1 list-like, you will need to pass a length-1 tuple to get_group in a
future version of pandas. Pass `(name,)` instead of `name` to silence this warnin
g.
  data_subset = grouped_data.get_group(pd_key)
```




The claim_ratio of fraudulent claims shows a wider distribution shifted toward higher values, suggesting claim inflation behavior. The near absence of fraud for ratios below 20 confirms that fraud cases systematically target significant amounts. However, the strong overlap between the two distributions indicates that this variable alone is insufficient to discriminate fraud, so it must be combined with other variables in the model.

In [39]:
```
sns.boxplot(
    x='fraud_reported',
    y='claim_ratio',
    data=df
)
plt.title("Claim Ratio vs Fraud")
plt.show()
```

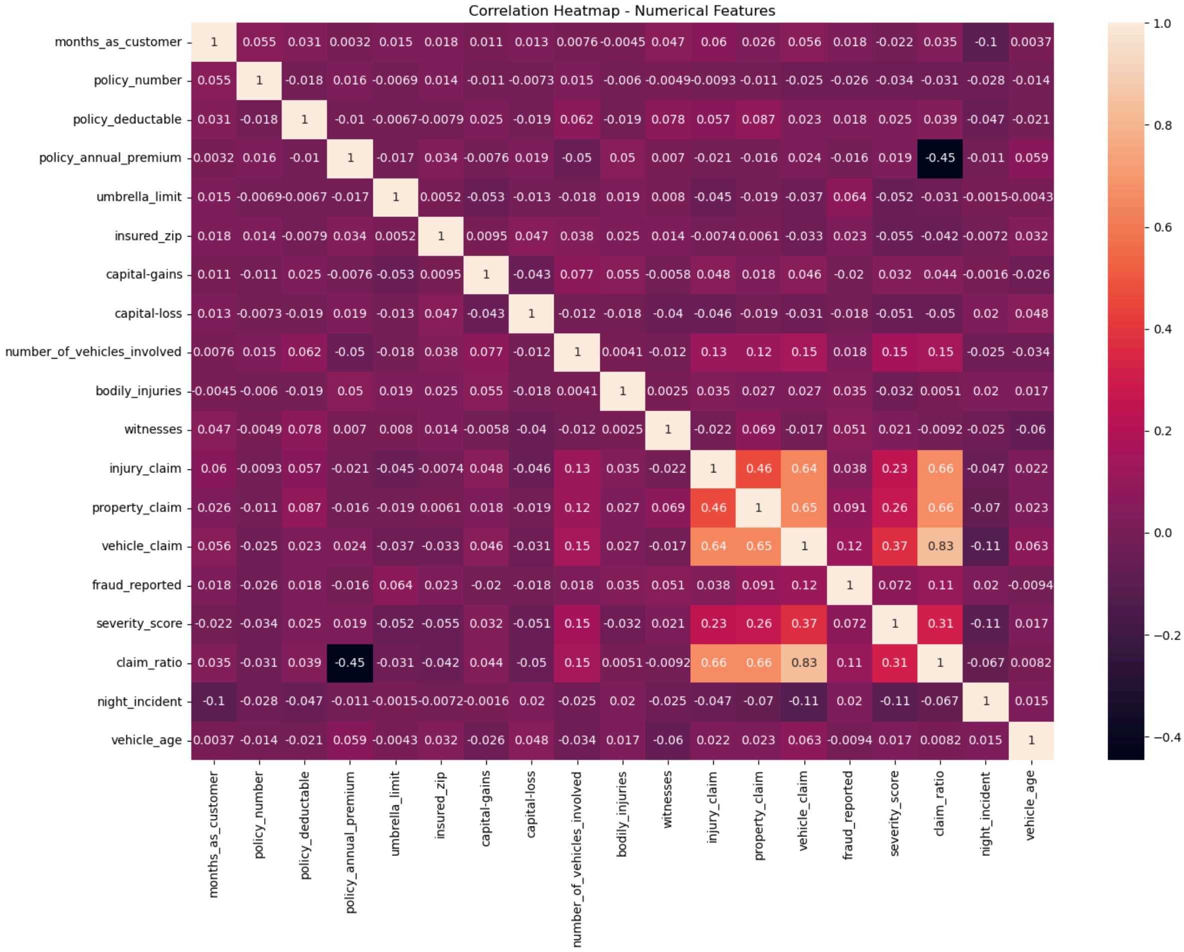



Here is a confirmation of the previous visual. The distributions are very similar between fraudulent and non-fraudulent claims, the medians are very close, and the outliers beyond 100 are comparable. This result indicates that the claim_ratio has low individual discriminative power for fraud detection. Combining it with other variables is essential to improve detection.

Regarding outliers, we cannot simply remove them, because in this case they may reflect instances of fraud, which is exactly what we want to detect. Therefore, they might not be noise but a signal of fraud. (Besides, if I use a tree-based model, it is robust to outliers anyway.)

The best approach is to start by plotting a heatmap to examine the relationship between the variable 'fraud_reported' and the other numerical variables. This allows us to get an initial overview of potential correlations before building the model:

In [40]:
```
plt.figure(figsize=(16,11))
sns.heatmap(df.corr(numeric_only=True), annot= True)
plt.title("Correlation Heatmap - Numerical Features")
plt.show()
```




all correlations with the fraud_reported variable are extremely weak. This indicates that no single numerical variable is able to predict fraud on its own. The predictive signal is more likely to be found within the categorical variables

In [41]:

```
#df
```

In [42]:

```
cat_cols = df.select_dtypes(include='object').columns.tolist()

for col in cat_cols:
    print(f"\n=== {col} ===")
    ct = pd.crosstab(df[col], df['fraud_reported'], normalize='index')
    ct.columns = ['Non Fraud', 'Fraud']
    ct = ct.sort_values('Fraud', ascending=False)
    print(ct.round(2))
```

```
=== policy_bind_date ===
                  Non Fraud  Fraud
policy_bind_date
1996-09-21              0.0    1.0
1995-07-25              0.0    1.0
2001-11-26              0.0    1.0
2011-01-22              0.0    1.0
2006-04-13              0.0    1.0
...                     ...    ...
1999-07-05              1.0    0.0
1999-07-24              1.0    0.0
1999-07-31              1.0    0.0
1999-08-11              1.0    0.0
2015-02-22              1.0    0.0

[865 rows x 2 columns]

=== policy_state ===
              Non Fraud  Fraud
policy_state
OH                 0.72   0.28
IN                 0.72   0.28
IL                 0.76   0.24

=== policy_csl ===
            Non Fraud  Fraud
policy_csl
250/500          0.72   0.28
100/300          0.74   0.26
500/1000         0.76   0.24

=== insured_sex ===
             Non Fraud  Fraud
insured_sex
MALE              0.71   0.29
FEMALE            0.75   0.25

=== insured_education_level ===
                         Non Fraud  Fraud
insured_education_level
MD                            0.71   0.29
College                       0.71   0.29
PhD                           0.72   0.28
JD                            0.72   0.28
Associate                     0.74   0.26
Masters                       0.76   0.24
High School                   0.77   0.23

=== insured_occupation ===
                    Non Fraud  Fraud
insured_occupation
exec-managerial          0.60   0.40
farming-fishing          0.67   0.33
transport-moving         0.69   0.31
tech-support             0.69   0.31
craft-repair             0.70   0.30
sales                    0.70   0.30
armed-forces             0.73   0.27
machine-op-inspct        0.76   0.24
protective-serv          0.76   0.24
prof-specialty           0.77   0.23
handlers-cleaners        0.79   0.21
other-service            0.80   0.20
priv-house-serv          0.82   0.18
```

```
adm-clerical             0.83   0.17

=== insured_hobbies ===
                 Non Fraud  Fraud
insured_hobbies
chess                 0.19   0.81
cross-fit             0.24   0.76
yachting              0.67   0.33
board-games           0.67   0.33
polo                  0.69   0.31
reading               0.71   0.29
base-jumping          0.72   0.28
paintball             0.76   0.24
hiking                0.76   0.24
skydiving             0.77   0.23
video-games           0.77   0.23
sleeping              0.79   0.21
exercise              0.80   0.20
basketball            0.81   0.19
movies                0.82   0.18
bungie-jumping        0.83   0.17
dancing               0.88   0.12
kayaking              0.89   0.11
golf                  0.89   0.11
camping               0.90   0.10

=== insured_relationship ===
                      Non Fraud  Fraud
insured_relationship
other-relative             0.69   0.31
not-in-family              0.72   0.28
wife                       0.72   0.28
unmarried                  0.74   0.26
own-child                  0.77   0.23
husband                    0.78   0.22

=== incident_date ===
               Non Fraud  Fraud
incident_date
2015-01-05          0.40   0.60
2015-02-08          0.41   0.59
2015-01-26          0.50   0.50
2015-01-19          0.55   0.45
2015-02-14          0.56   0.44
2015-02-07          0.56   0.44
2015-01-02          0.56   0.44
2015-01-20          0.56   0.44
2015-01-29          0.60   0.40
2015-01-10          0.61   0.39
2015-01-15          0.62   0.38
2015-02-19          0.62   0.38
2015-02-02          0.64   0.36
2015-02-01          0.65   0.35
2015-01-21          0.65   0.35
2015-01-09          0.65   0.35
2015-02-04          0.65   0.35
2015-01-13          0.67   0.33
2015-03-01          0.67   0.33
2015-01-14          0.68   0.32
2015-01-01          0.69   0.31
2015-02-20          0.69   0.31
2015-01-24          0.71   0.29
2015-01-08          0.71   0.29
2015-01-12          0.72   0.28
```

```
2015-01-31          0.72   0.28
2015-02-12          0.74   0.26
2015-02-06          0.74   0.26
2015-01-27          0.75   0.25
2015-02-21          0.75   0.25
2015-01-03          0.75   0.25
2015-01-30          0.75   0.25
2015-01-11          0.78   0.22
2015-02-16          0.79   0.21
2015-02-13          0.80   0.20
2015-02-24          0.80   0.20
2015-01-16          0.80   0.20
2015-01-25          0.80   0.20
2015-02-15          0.81   0.19
2015-02-28          0.81   0.19
2015-02-09          0.82   0.18
2015-02-03          0.82   0.18
2015-02-22          0.82   0.18
2015-01-07          0.83   0.17
2015-02-23          0.83   0.17
2015-02-27          0.83   0.17
2015-01-22          0.85   0.15
2015-01-23          0.85   0.15
2015-01-17          0.85   0.15
2015-01-28          0.85   0.15
2015-02-05          0.86   0.14
2015-02-26          0.86   0.14
2015-02-10          0.86   0.14
2015-01-18          0.87   0.13
2015-02-17          0.87   0.13
2015-02-25          0.88   0.12
2015-02-11          0.89   0.11
2015-01-04          0.91   0.09
2015-02-18          0.93   0.07
2015-01-06          0.94   0.06

=== incident_type ===
                          Non Fraud  Fraud
incident_type
Single Vehicle Collision       0.71   0.29
Multi-vehicle Collision        0.73   0.27
Parked Car                     0.88   0.12
Vehicle Theft                  0.89   0.11

=== collision_type ===
                 Non Fraud  Fraud
collision_type
Rear Collision        0.69   0.31
Front Collision       0.72   0.28
Side Collision        0.75   0.25
Unknown               0.89   0.11

=== incident_severity ===
                   Non Fraud  Fraud
incident_severity
Major Damage            0.39   0.61
Total Loss              0.87   0.13
Minor Damage            0.89   0.11
Trivial Damage          0.92   0.08

=== authorities_contacted ===
                         Non Fraud  Fraud
authorities_contacted
Other                         0.68   0.32
```

```
Ambulance                   0.71   0.29
Fire                        0.73   0.27
Police                      0.79   0.21

=== incident_state ===
                Non Fraud  Fraud
incident_state
OH                   0.50   0.50
NC                   0.67   0.33
SC                   0.68   0.32
PA                   0.72   0.28
VA                   0.76   0.24
NY                   0.76   0.24
WV                   0.81   0.19

=== incident_city ===
               Non Fraud  Fraud
incident_city
Arlington           0.68   0.32
Hillsdale           0.73   0.27
Columbus            0.73   0.27
Springfield         0.74   0.26
Northbend           0.75   0.25
Riverwood           0.75   0.25
Northbrook          0.77   0.23

=== incident_location ===
                      Non Fraud  Fraud
incident_location
1012 5th Lane               0.0    1.0
5093 Flute Lane             0.0    1.0
5431 3rd Ridge              0.0    1.0
5352 Lincoln Drive          0.0    1.0
5280 Pine Ave               0.0    1.0
...                         ...    ...
5380 Pine St                1.0    0.0
5383 Maple Drive            1.0    0.0
2204 Washington Lane        1.0    0.0
5445 Tree Hwy               1.0    0.0
6467 Best Ave               1.0    0.0

[909 rows x 2 columns]

=== property_damage ===
                 Non Fraud  Fraud
property_damage
Unknown               0.69   0.31
YES                   0.73   0.27
NO                    0.79   0.21

=== police_report_available ===
                         Non Fraud  Fraud
police_report_available
Unknown                       0.72   0.28
NO                            0.73   0.27
YES                           0.76   0.24

=== auto_make ===
            Non Fraud  Fraud
auto_make
Mercedes         0.63   0.37
Ford             0.67   0.33
Audi             0.68   0.32
Volkswagen       0.69   0.31
```

```
Chevrolet        0.70   0.30
BMW              0.72   0.28
Dodge            0.74   0.26
Honda            0.74   0.26
Saab             0.75   0.25
Suburu           0.76   0.24
Toyota           0.79   0.21
Accura           0.79   0.21
Nissan           0.80   0.20
Jeep             0.82   0.18

=== auto_model ===
                Non Fraud  Fraud
auto_model
Silverado            0.53   0.47
ML350                0.56   0.44
X6                   0.56   0.44
F150                 0.60   0.40
Civic                0.61   0.39
C300                 0.61   0.39
M5                   0.62   0.38
Tahoe                0.64   0.36
RAM                  0.65   0.35
92x                  0.67   0.33
Impreza              0.67   0.33
A5                   0.67   0.33
Fusion               0.68   0.32
Passat               0.69   0.31
Maxima               0.70   0.30
A3                   0.70   0.30
Jetta                0.70   0.30
X5                   0.70   0.30
Highlander           0.70   0.30
E400                 0.71   0.29
Forrestor            0.73   0.27
Escape               0.74   0.26
Grand Cherokee       0.75   0.25
MDX                  0.76   0.24
Accord               0.77   0.23
93                   0.78   0.22
TL                   0.79   0.21
95                   0.80   0.20
Corolla              0.82   0.18
Legacy               0.83   0.17
Camry                0.84   0.16
CRV                  0.84   0.16
Ultima               0.84   0.16
Neon                 0.84   0.16
Pathfinder           0.85   0.15
Wrangler             0.87   0.13
Malibu               0.88   0.12
RSX                  0.91   0.09
3 Series             0.94   0.06
```

In [43]:

```
#à supprimer car le signal est trop faible :
df = df.drop(columns=['policy_bind_date', 'incident_date', 'incident_location', 'po
```

# Encodage:

In [44]:

```
df = df.drop(columns=['incident_severity'])#cause i already create a column
```

In [45]:
```
df = pd.get_dummies(df, drop_first=True).astype(int)
```

In [46]:
```
df
```

Out[46]:

| | months_as_customer | policy_number | policy_deductable | policy_annual_premium | umbrella_limi |
|---|---|---|---|---|---|
| **0** | 328 | 521585 | 1000 | 1406 | |
| **1** | 228 | 342868 | 2000 | 1197 | 500000 |
| **2** | 134 | 687698 | 2000 | 1413 | 500000 |
| **3** | 256 | 227811 | 2000 | 1415 | 600000 |
| **5** | 256 | 104594 | 1000 | 1351 | |
| **...** | ... | ... | ... | ... | . |
| **995** | 3 | 941851 | 1000 | 1310 | |
| **996** | 285 | 186934 | 1000 | 1436 | |
| **997** | 130 | 918516 | 500 | 1383 | 300000 |
| **998** | 458 | 533940 | 2000 | 1356 | 500000 |
| **999** | 456 | 556080 | 1000 | 766 | |

909 rows × 86 columns

In [47]:
```
#df.shape
```

In [48]:
```
X= df.drop(columns=['fraud_reported'])
y = df['fraud_reported']
```

since Random Forest and XGBoost do not require feature scaling, unlike Logistic Regression, I first perform the train-test split and then apply scaling only for the Logistic Regression model.

## Train-test split

In [49]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

# Model Training

In [50]:
```
#scalling pour logestic regression uniquement:
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

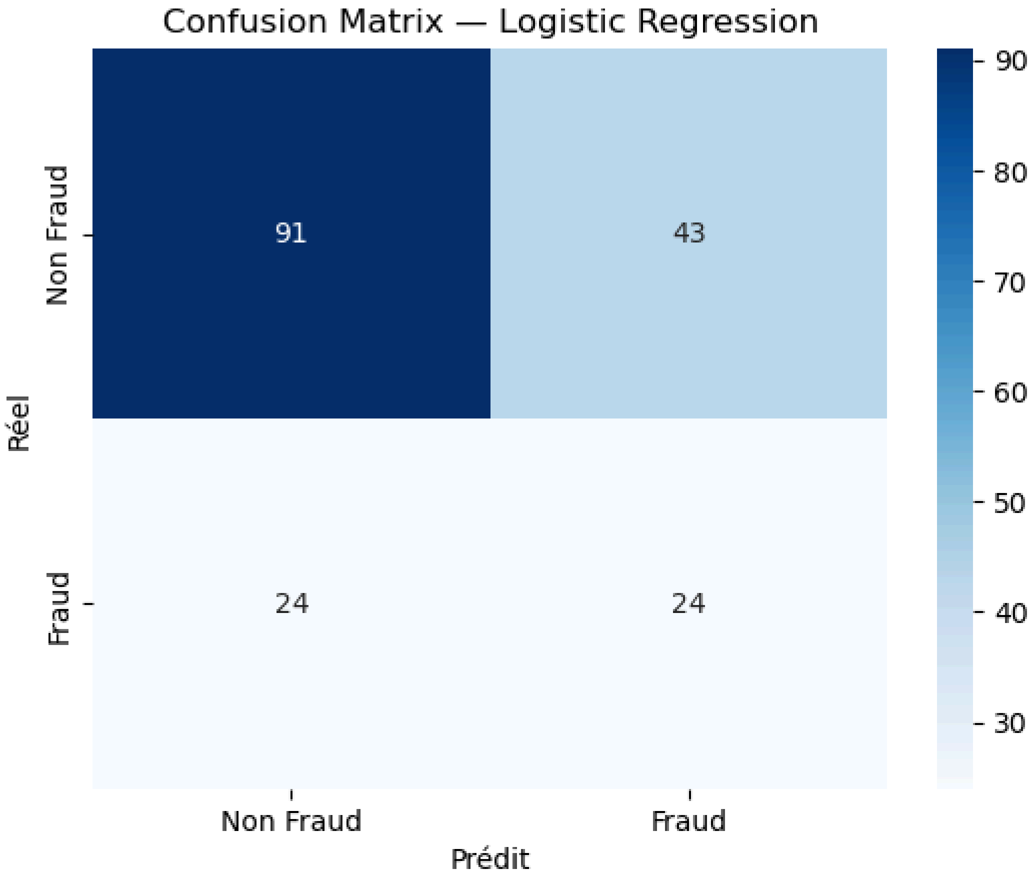
In [51]:
```
#LR
lr = LogisticRegression(max_iter=1000, class_weight='balanced', random_state=42)
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)
```

In [52]:
```
#random forest
rf = RandomForestClassifier(n_estimators=100, class_weight='balanced', random_state
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)
```

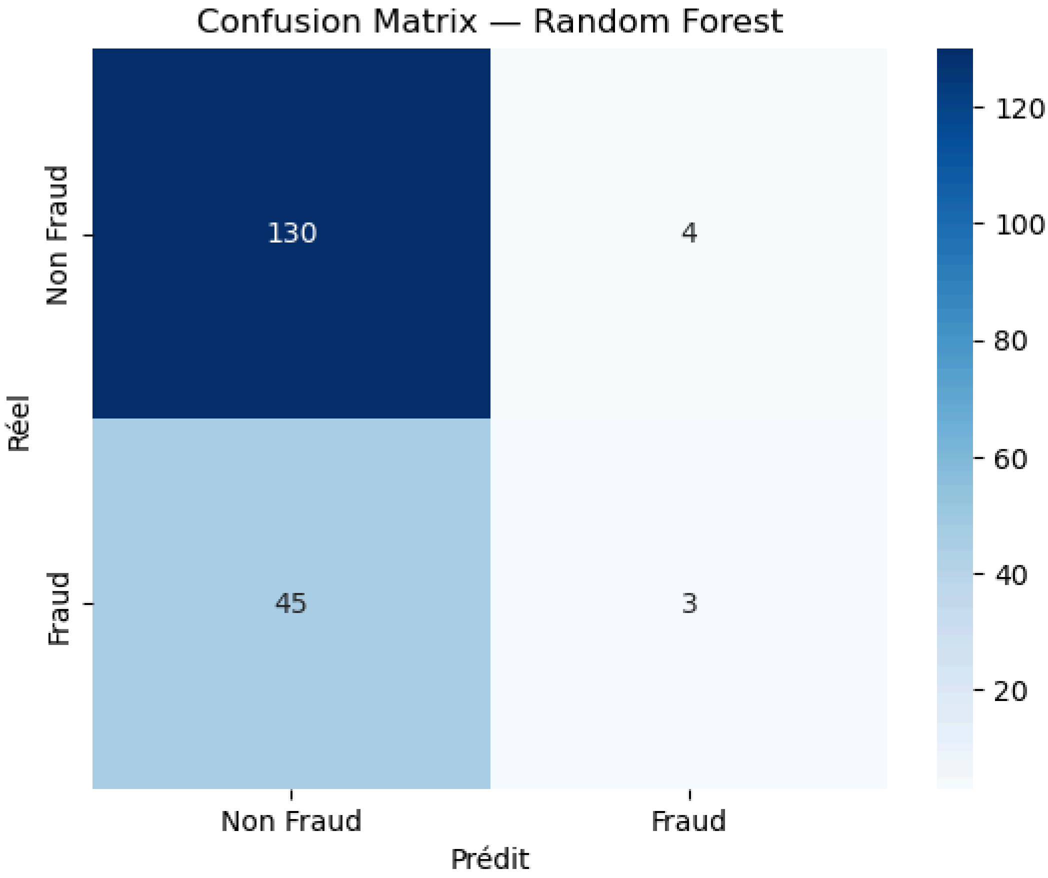
In [53]:
```
xgb = XGBClassifier(n_estimators=100, random_state=42, eval_metric='logloss', scale
xgb.fit(X_train, y_train)
y_pred_xgb = xgb.predict(X_test)
```

In [54]:
```
models = {
    'Logistic Regression': (y_pred_lr, lr.predict_proba(X_test_scaled)[:,1]),
    'Random Forest': (y_pred_rf, rf.predict_proba(X_test)[:,1]),
    'XGBoost': (y_pred_xgb, xgb.predict_proba(X_test)[:,1])
}
for name, (y_pred, y_proba) in models.items():
    print(f"\n{'='*40}")
    print(f"  {name}")
    print(f"{'='*40}")
    print(classification_report(y_test, y_pred))
    print(f"AUC-ROC : {roc_auc_score(y_test, y_proba):.3f}")

    # Matrice de confusion
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['Non Fraud', 'Fraud'],
                yticklabels=['Non Fraud', 'Fraud'])
    plt.title(f"Confusion Matrix — {name}")
    plt.ylabel('Réel')
    plt.xlabel('Prédit')
    plt.show()
```

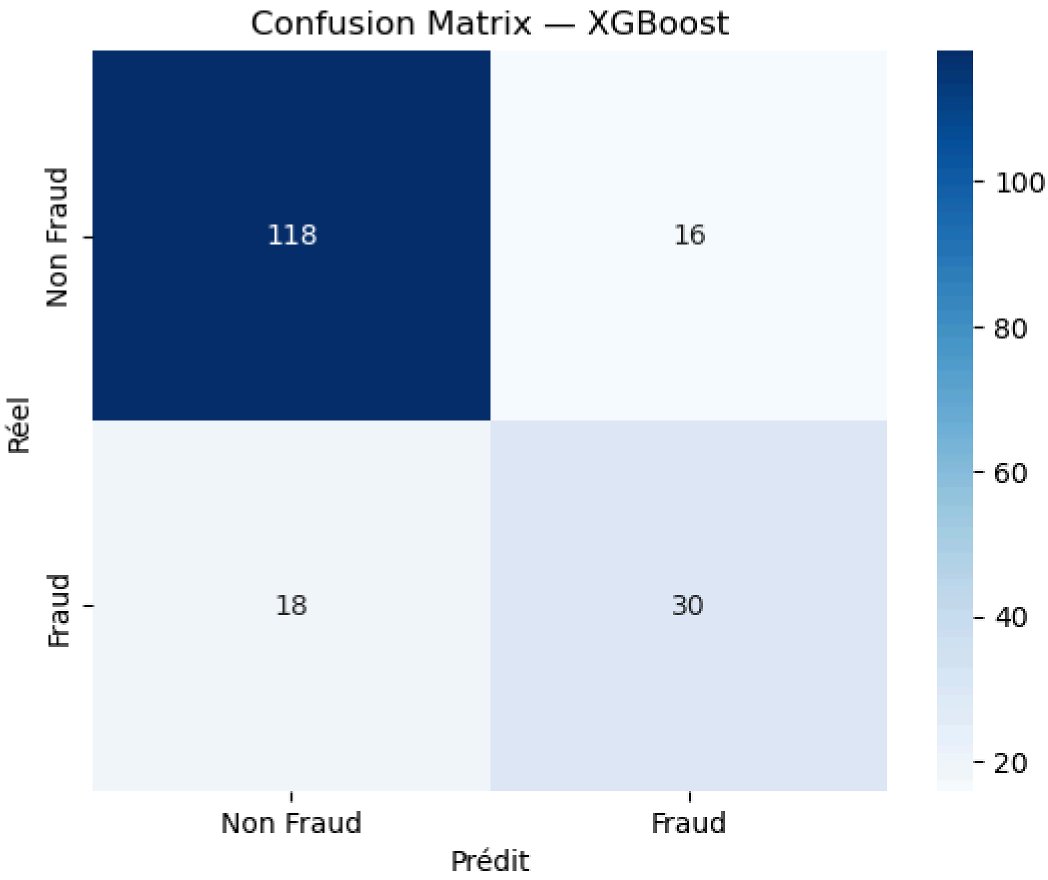
```
========================================
  Logistic Regression
========================================
              precision    recall  f1-score   support

           0       0.79      0.68      0.73       134
           1       0.36      0.50      0.42        48

    accuracy                           0.63       182
   macro avg       0.57      0.59      0.57       182
weighted avg       0.68      0.63      0.65       182

AUC-ROC : 0.648
```




```
========================================
  Random Forest
========================================
              precision    recall  f1-score   support

           0       0.74      0.97      0.84       134
           1       0.43      0.06      0.11        48

    accuracy                           0.73       182
   macro avg       0.59      0.52      0.48       182
weighted avg       0.66      0.73      0.65       182

AUC-ROC : 0.819
```




```
========================================
  XGBoost
========================================
              precision    recall  f1-score   support

           0       0.87      0.88      0.87       134
           1       0.65      0.62      0.64        48

    accuracy                           0.81       182
   macro avg       0.76      0.75      0.76       182
weighted avg       0.81      0.81      0.81       182

AUC-ROC : 0.834
```




The XGBoost model shows better performance, with an AUC-ROC of 0.834 and an F1-score of 0.64 for the fraud class. However, despite an accuracy of 73%, it only detects 3 fraud cases out of 48. Therefore, the model needs to be optimized using hyperparameter tuning to further improve its performance.

# Hyperparameter Tuning

In [55]:

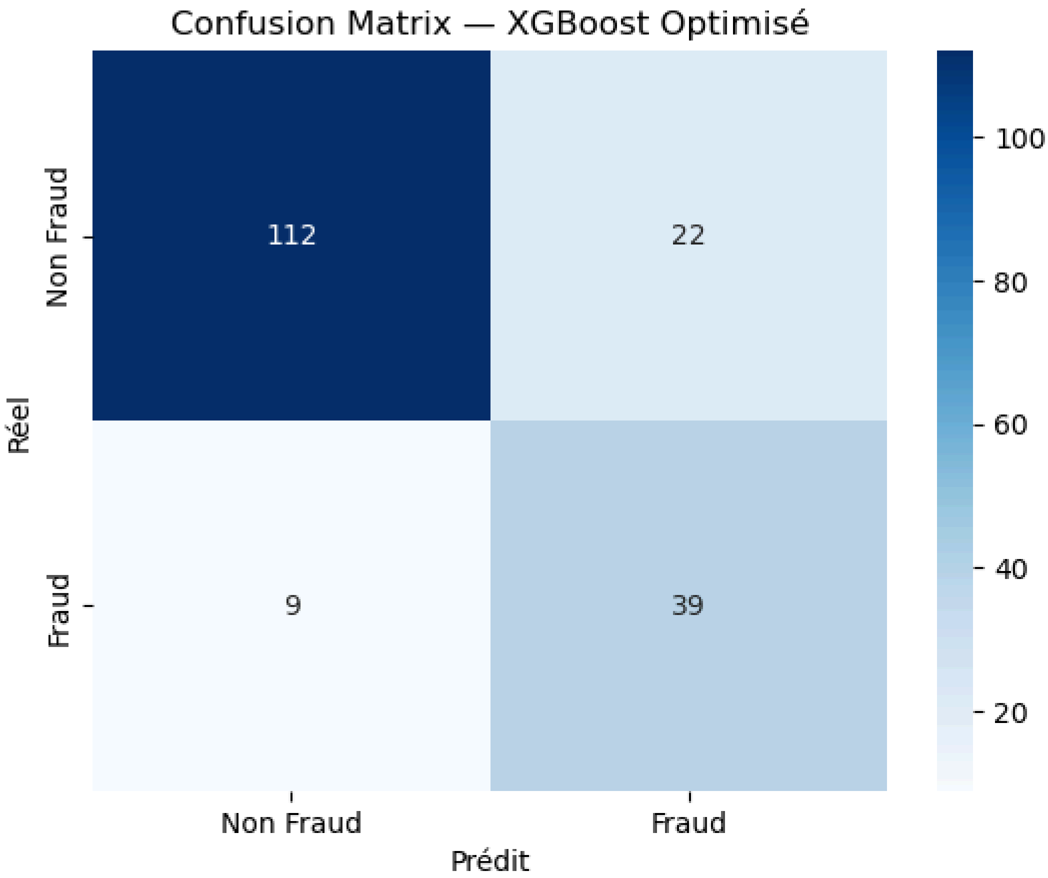
```
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [3, 4, 5],
    'learning_rate': [0.01, 0.1, 0.2],
    'scale_pos_weight': [2, 3, 4]
}

grid_search = GridSearchCV(
    XGBClassifier(random_state=42, eval_metric='logloss'),
    param_grid,
    cv=5,
    scoring='roc_auc',
    n_jobs=-1,
    verbose=1
)

start = time.time()
grid_search.fit(X_train, y_train)
end = time.time()
print(f"Temps d'entraînement : {end - start:.2f} secondes")

print(f"Meilleurs paramètres : {grid_search.best_params_}")
print(f"Meilleur AUC-ROC (cross-validation) : {grid_search.best_score_:.3f}")
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
Temps d'entraînement : 62.33 secondes
Meilleurs paramètres : {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 10
0, 'scale_pos_weight': 2}
Meilleur AUC-ROC (cross-validation) : 0.871
```

In [56]:

```
best_xgb = XGBClassifier(
    learning_rate=0.01,
    max_depth=3,
    n_estimators=100,
    scale_pos_weight=2,
    random_state=42,
    eval_metric='logloss'
)

best_xgb.fit(X_train, y_train)
y_pred_best = best_xgb.predict(X_test)
y_proba_best = best_xgb.predict_proba(X_test)[:,1]

print(classification_report(y_test, y_pred_best))
print(f"AUC-ROC : {roc_auc_score(y_test, y_proba_best):.3f}")

cm = confusion_matrix(y_test, y_pred_best)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Non Fraud', 'Fraud'],
            yticklabels=['Non Fraud', 'Fraud'])
plt.title("Confusion Matrix – XGBoost Optimisé")
plt.ylabel('Réel')
plt.xlabel('Prédit')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.93      0.84      0.88       134
           1       0.64      0.81      0.72        48

    accuracy                           0.83       182
   macro avg       0.78      0.82      0.80       182
weighted avg       0.85      0.83      0.84       182

AUC-ROC : 0.852
```




In [57]:
```
y_proba_best = best_xgb.predict_proba(X_test)[:,1]

#test different
for threshold in [0.3, 0.4, 0.5]:
    y_pred_threshold = (y_proba_best >= threshold).astype(int)
    print(f"\nSeuil : {threshold}")
    print(classification_report(y_test, y_pred_threshold))
```

```
Seuil : 0.3
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       134
           1       0.64      0.88      0.74        48

    accuracy                           0.84       182
   macro avg       0.79      0.85      0.81       182
weighted avg       0.87      0.84      0.84       182


Seuil : 0.4
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       134
           1       0.64      0.88      0.74        48

    accuracy                           0.84       182
   macro avg       0.79      0.85      0.81       182
weighted avg       0.87      0.84      0.84       182


Seuil : 0.5
              precision    recall  f1-score   support

           0       0.93      0.84      0.88       134
           1       0.64      0.81      0.72        48

    accuracy                           0.83       182
   macro avg       0.78      0.82      0.80       182
weighted avg       0.85      0.83      0.84       182
```

# Threshold Optimization

In [58]:

```
for threshold in [0.35, 0.40, 0.45, 0.50]:
    y_pred_threshold = (y_proba_best >= threshold).astype(int)
    recall_fraud = recall_score(y_test, y_pred_threshold)
    precision_fraud = precision_score(y_test, y_pred_threshold)
    f1_fraud = f1_score(y_test, y_pred_threshold)
    print(f"Seuil {threshold} | Recall: {recall_fraud:.2f} | Precision: {precision_
```

```
Seuil 0.35 | Recall: 0.88 | Precision: 0.64 | F1: 0.74
Seuil 0.4 | Recall: 0.88 | Precision: 0.64 | F1: 0.74
Seuil 0.45 | Recall: 0.81 | Precision: 0.64 | F1: 0.72
Seuil 0.5 | Recall: 0.81 | Precision: 0.64 | F1: 0.72
```

Instead of using the default 0.5 probability threshold, several thresholds were tested.

Best threshold: 0.4

Results: (Recall: 88%, Precision: 64%, F1 Score: 0.74)

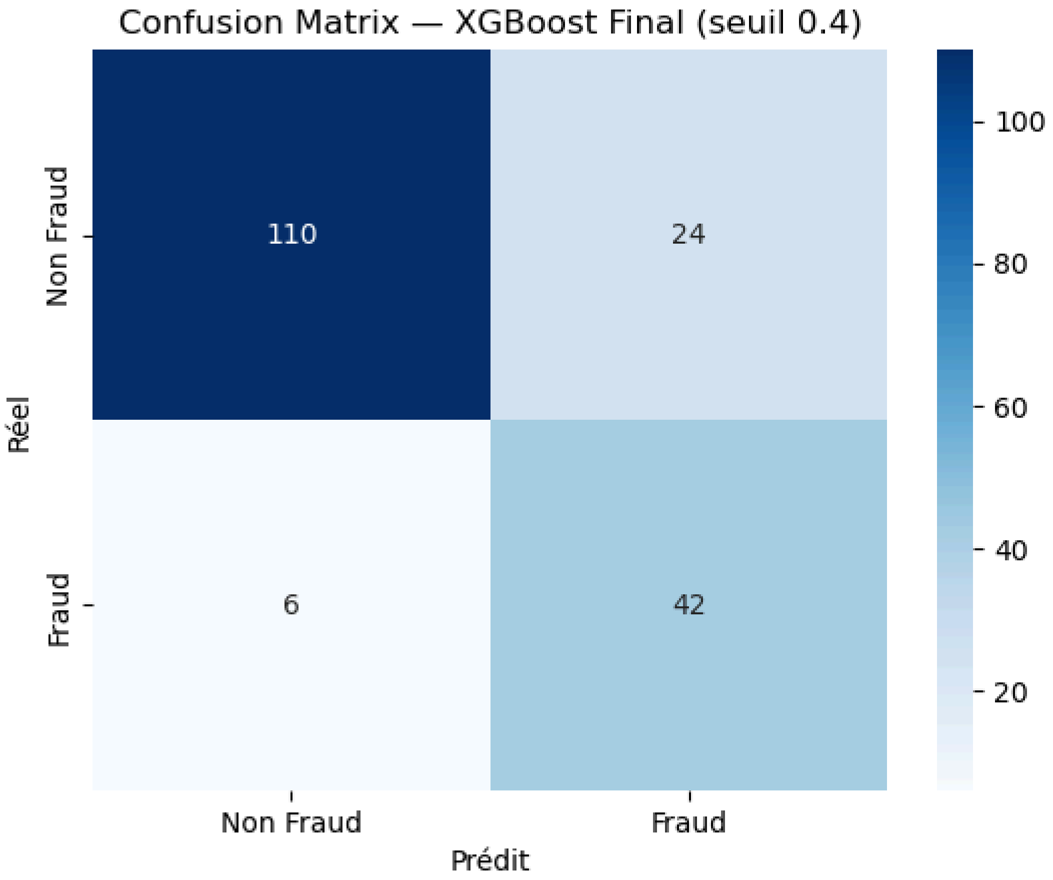
Lowering the threshold increases fraud detection while keeping a reasonable precision.

# Final Mdel Performance

In [59]:

```
y_pred_final = (y_proba_best >= 0.4).astype(int)
```

```
cm = confusion_matrix(y_test, y_pred_final)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Non Fraud', 'Fraud'],
            yticklabels=['Non Fraud', 'Fraud'])
plt.title("Confusion Matrix — XGBoost Final (seuil 0.4)")
plt.ylabel('Réel')
plt.xlabel('Prédit')
plt.show()
```

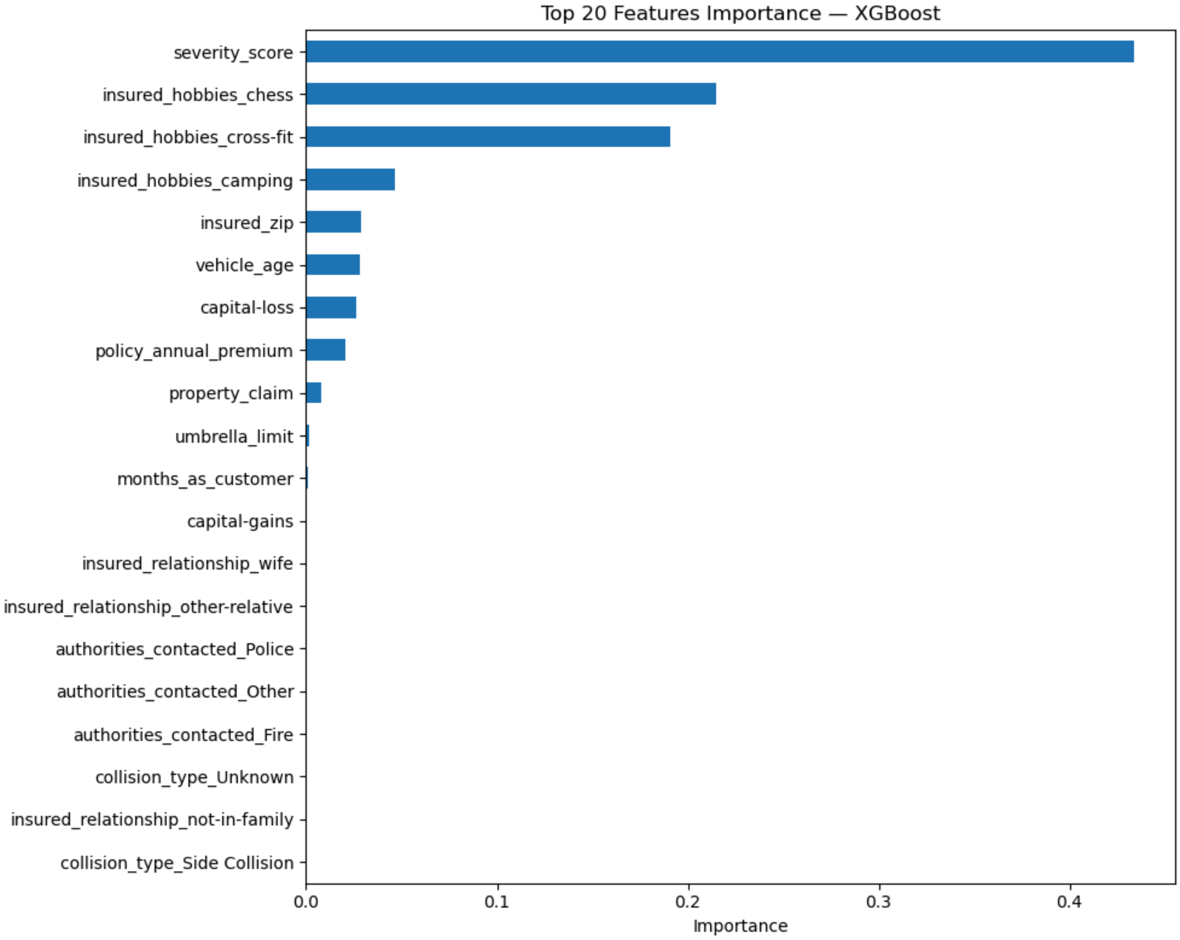



Final model results: Accuracy: 83.5%, Fraud recall: 88%, AUC-ROC: 0.852

The model successfully detects 42 out of 48 fraud cases, significantly reducing missed fraud.

# Feature Importance

In [60]:
```
feat_importance = pd.Series(
    best_xgb.feature_importances_,
    index=X.columns
).sort_values(ascending=False)

plt.figure(figsize=(10, 8))
feat_importance.head(20).plot(kind='barh')
plt.gca().invert_yaxis()
plt.title("Top 20 Features Importance — XGBoost")
plt.xlabel("Importance")
plt.tight_layout()
plt.show()
```




In [61]:
```
#garder uniquement les features avec importance > 0.01
important_features = feat_importance[feat_importance > 0.01].index.tolist()
print(f"Nombre de features retenues : {len(important_features)}")
print(important_features)
```

```
Nombre de features retenues : 8
['severity_score', 'insured_hobbies_chess', 'insured_hobbies_cross-fit', 'insured_
hobbies_camping', 'insured_zip', 'vehicle_age', 'capital-loss', 'policy_annual_pre
mium']
```

The most important variables identified by the model include:

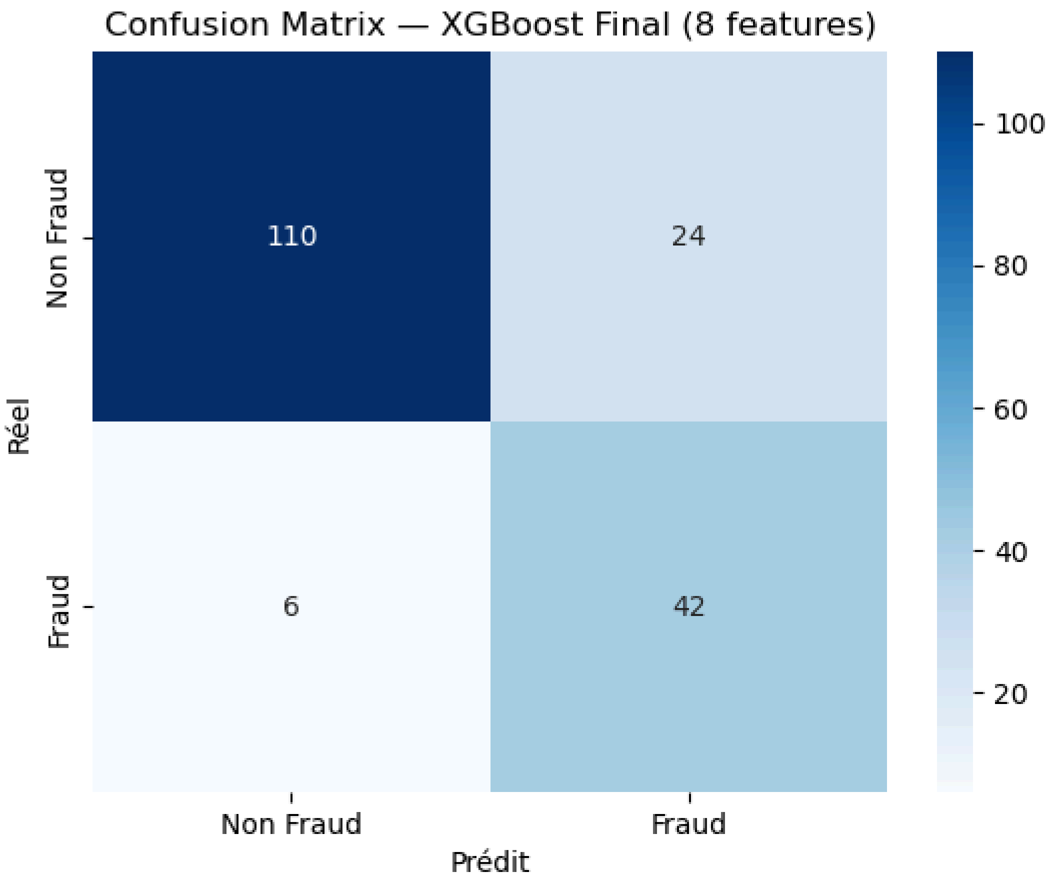
severity_score, insured_hobbies_chess, insured_hobbies_cross-fit, insured_zip, vehicle_age, capital_loss, policy_annual_premium

This confirms the insights observed during the exploratory analysis.

In [62]:
```
#new dataset with importance feature
X_train_imp = X_train[important_features]
X_test_imp = X_test[important_features]

#réentraîner XGBoost
best_xgb_imp = XGBClassifier(
    learning_rate=0.01,
    max_depth=3,
    n_estimators=100,
    scale_pos_weight=2,
    random_state=42,
    eval_metric='logloss'
)

best_xgb_imp.fit(X_train_imp, y_train)
```

```python
y_proba_imp = best_xgb_imp.predict_proba(X_test_imp)[:,1]
y_pred_imp = (y_proba_imp >= 0.4).astype(int)

print(classification_report(y_test, y_pred_imp))
print(f"AUC-ROC : {roc_auc_score(y_test, y_proba_imp):.3f}")
print(f"Taux d'erreur : {(1 - accuracy_score(y_test, y_pred_imp))*100:.1f}%")

#conf matrix
cm = confusion_matrix(y_test, y_pred_imp)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Non Fraud', 'Fraud'],
            yticklabels=['Non Fraud', 'Fraud'])
plt.title("Confusion Matrix — XGBoost Final (8 features)")
plt.ylabel('Réel')
plt.xlabel('Prédit')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.95      0.82      0.88       134
           1       0.64      0.88      0.74        48

    accuracy                           0.84       182
   macro avg       0.79      0.85      0.81       182
weighted avg       0.87      0.84      0.84       182

AUC-ROC : 0.852
Taux d'erreur : 16.5%
```




Using only 8 features, I obtained exactly the same performance as with the initial 86 features. This is a positive result, as it indicates that the model's performance remains unchanged while being significantly simpler. It also confirms that fraud detection in this dataset relies primarily on these key features

This project demonstrates how machine learning can be applied to insurance data to detect fraudulent claims.

Using advanced models such as XGBoost, it is possible to detect a large proportion of fraud cases while maintaining good overall performance.