1 Load the CSV file as a DataFrame

Used panda's library to load data from excel file to Dataframe

```
# Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[205] ✓ 0.0s

1.1 Load the Data

```
# Load the dataset
df=pd.read_csv('insurance_claims.csv')
```

[206] ✓ 0.0s

Check top 5 records

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

✓ 0.0s Python

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_education_level	in
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.910000	0	466132	MALE	MD	
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.220000	5000000	468176	MALE	MD	
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.140000	5000000	430632	FEMALE	PhD	
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.740000	6000000	608117	FEMALE	PhD	
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.910000	6000000	610706	MALE	Associate	

Check rows and columns

```
# Inspect the shape of the dataset
df.shape
```

[5] ✓ 0.0s

```
(1000, 40)
```

Check columns in Data frame

```
1  # Inspect the features in the dataset
2  df.info()
```

✓ 0.0s

```
<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
```

Check Null value count in Data frame

```
1  # Check the number of missing values in each column
2  df.isnull().sum()
```

✓ 0.0s

```
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
```

```
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
```

Handle Null value count in Data frame

```
[9]  ✓ 0.0s
```

```python
# Handle the rows containing null values
# Set default value as 'Unknown' for authorities_contacted
df['authorities_contacted'].fillna('Unknown', inplace=True)
```

```python
# Write code to display all the columns with their unique values and counts and check for redundant values
df.nunique().sort_values(ascending=False)
```

✓ 0.0s

```
policy_number                1000
incident_location            1000
insured_zip                   995
policy_annual_premium         991
policy_bind_date              951
total_claim_amount            763
vehicle_claim                 726
injury_claim                  638
property_claim                626
months_as_customer            391
capital-loss                  354
capital-gains                 338
incident_date                  60
age                            46
auto_model                     39
incident_hour_of_the_day       24
auto_year                      21
insured_hobbies                20
auto_make                      14
insured_occupation             14
umbrella_limit                 11
insured_education_level         7
incident_state                  7
incident_city                   7
insured_relationship            6
authorities_contacted           5
witnesses                       4
incident_severity               4
number_of_vehicles_involved     4
collision_type                  4
incident_type                   4
bodily_injuries                 3
property_damage                 3
police_report_available         3
policy_deductable               3
policy_csl                      3
```

Drop columns that have unique values in Data frame

```python
# Identify and drop any columns that are completely empty
# _c39 has no data. We can drop it.
df.drop(columns=['_c39'], inplace=True)
```

✓ 0.0s

Column 'umbrella_limit' has negative values.

Column 'capital-loss' has negative values.

These are numerical columns and is amount. Let’s ignore sign of amount feature.

```python
# Identify and drop rows where features have illogical or invalid values, such as negative values for features that should only have positive values
# check numerical columns with negative values
num_cols = df.select_dtypes(include=['int64', 'float64']).columns
for col in num_cols:
    if (df[col] < 0).any():
        print(f"Column '{col}' has negative values.")

# print rows with negative values in any numerical column
neg_values = df[(df[num_cols] < 0).any(axis=1)]
neg_values

#select only negative columns in negative_df
neg_values = neg_values[num_cols[(neg_values[num_cols] < 0).any()]]
neg_values


```

✓ 0.0s

```
Column 'umbrella_limit' has negative values.
Column 'capital-loss' has negative values.
```

	umbrella_limit	capital-loss
3	6000000	-62400
4	6000000	-46000
6	0	-77000
9	0	-39300
11	0	-51000
...	...	...
989	0	-54000
990	3000000	-32800
991	0	-32200
993	0	-32100
994	0	-82100

526 rows × 2 columns

```python
# lets change negative values to absolute values
for col in neg_values.columns:
    df[col] = df[col].abs()
# check if negative values are removed
neg_values = df[(df[num_cols] < 0).any(axis=1)]
neg_values  # should be empty

```

✓ 0.0s Python

months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	...	witnesses	police_report_available	to

0 rows × 39 columns

```python
# set print option of python to display all rows and columns
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
#df["capital-loss"].value_counts()
df["umbrella_limit"].value_counts()

```

✓ 0.0s

```
umbrella_limit
0          798
6000000     57
5000000     46
4000000     39
7000000     29
3000000     12
8000000      8
9000000      5
2000000      3
10000000     2
1000000      1
Name: count, dtype: int64
```

Drop irrelevant features or features that contribute very minimum to prediction

```python
# Identify and remove columns that are likely to be identifiers or have very limited predictive power
# Drop policy_number as primary key or unique identifier
# Drop incident_location and insured_zip as they are location identifiers
# Drop policy_annual_premium and policy_bind_date as they are related to policy details which may not contribute to fraud detection
df.drop(columns=['policy_number','incident_location','insured_zip','policy_annual_premium','policy_bind_date'], inplace=True, errors='ignore')

```

✓ 0.0s

```python
# Check the dataset
df.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 34 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1000 non-null   int64
 1   age                          1000 non-null   int64
 2   policy_state                 1000 non-null   object
 3   policy_csl                   1000 non-null   object
 4   policy_deductable            1000 non-null   int64
 5   umbrella_limit               1000 non-null   int64
 6   insured_sex                  1000 non-null   object
 7   insured_education_level      1000 non-null   object
 8   insured_occupation           1000 non-null   object
 9   insured_hobbies              1000 non-null   object
 10  insured_relationship         1000 non-null   object
 11  capital-gains                1000 non-null   int64
 12  capital-loss                 1000 non-null   int64
 13  incident_date                1000 non-null   object
 14  incident_type                1000 non-null   object
 15  collision_type               1000 non-null   object
 16  incident_severity            1000 non-null   object
 17  authorities_contacted        1000 non-null   object
 18  incident_state               1000 non-null   object
 19  incident_city                1000 non-null   object
 20  incident_hour_of_the_day     1000 non-null   int64
 21  number_of_vehicles_involved  1000 non-null   int64
 22  property_damage              1000 non-null   object
 23  bodily_injuries              1000 non-null   int64
 24  witnesses                    1000 non-null   int64
 25  police_report_available      1000 non-null   object
 26  total_claim_amount           1000 non-null   int64
 27  injury_claim                 1000 non-null   int64
 28  property_claim               1000 non-null   int64
 29  vehicle_claim                1000 non-null   int64
 30  auto_make                    1000 non-null   object
```

```
 31  auto_model                  1000 non-null   object
 32  auto_year                   1000 non-null   int64
 33  fraud_reported              1000 non-null   object
dtypes: int64(15), object(19)
memory usage: 265.8+ KB
```

Convert string to Date

```
# Fix the data types of the columns with incorrect data types
# convert object columns that have date value policy_bind_date

df['incident_date'] = pd.to_datetime(df['incident_date'], errors='coerce')

```

✓ 0.0s

```
# Check the features of the data again
df["incident_date"].info()
#print(df["incident_date"].value_counts(dropna=False).sort_index())
```

✓ 0.0s

```
<class 'pandas.core.series.Series'>
RangeIndex: 1000 entries, 0 to 999
Series name: incident_date
Non-Null Count  Dtype
--------------  -----
1000 non-null   datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 7.9 KB
```

Get X and y

```
# Put all the feature variables in X

X=df
# Put the target variable in y
#y=df.pop("fraud_reported")
y=df['fraud_reported'].map({'Y':1, 'N':0})  # convert to binary 1/0
X['fraud_reported']=df['fraud_reported'].map({'Y':1, 'N':0})
```

✓ 0.0s

Split dataset into Train and Test

```python
# Split the dataset into 70% train and 30% validation and use stratification on the target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)
# Reset index for all train and test sets
X_train.reset_index(drop=True, inplace=True)
X_test.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
y_test.reset_index(drop=True, inplace=True)
# Check the shape of the train and test sets
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

✓ 0.0s

```
((700, 34), (300, 34), (700,), (300,))
```

Get Numerical columns

```python
# Select numerical columns
num_cols = X_train.select_dtypes(include=['int64', 'float64']).columns
num_cols
```

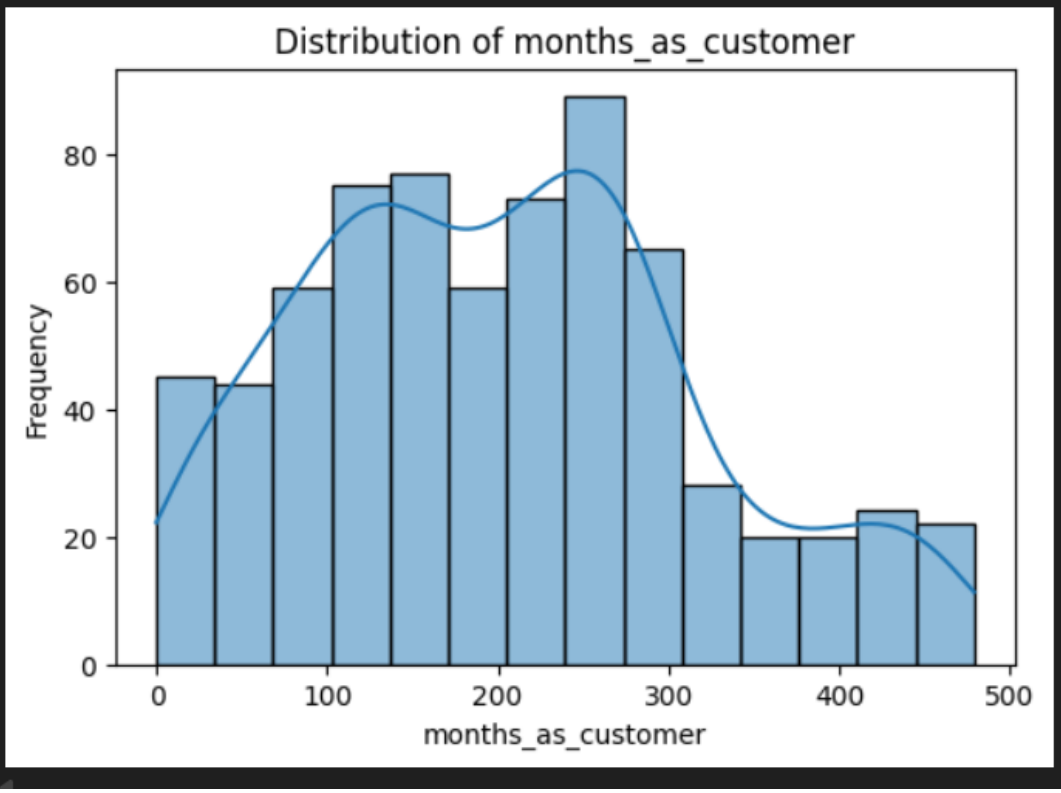
✓ 0.0s

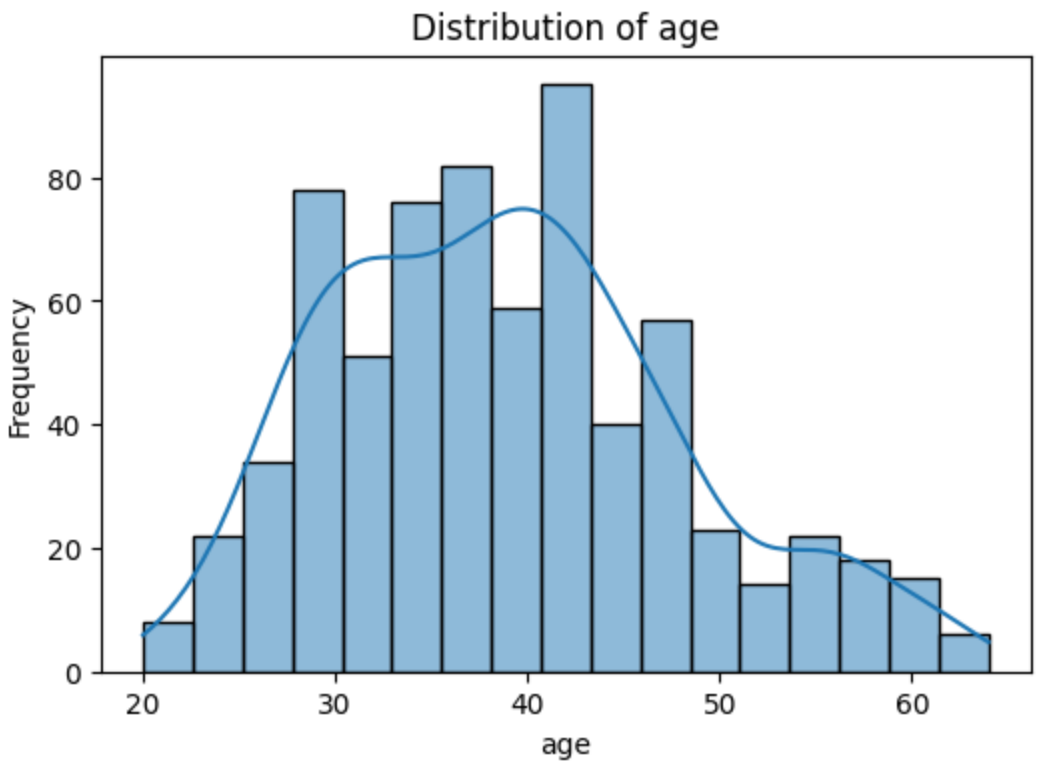
```
Index(['months_as_customer', 'age', 'policy_deductable', 'umbrella_limit',
       'capital-gains', 'capital-loss', 'incident_hour_of_the_day',
       'number_of_vehicles_involved', 'bodily_injuries', 'witnesses',
       'total_claim_amount', 'injury_claim', 'property_claim', 'vehicle_claim',
       'auto_year', 'fraud_reported'],
      dtype='object')
```

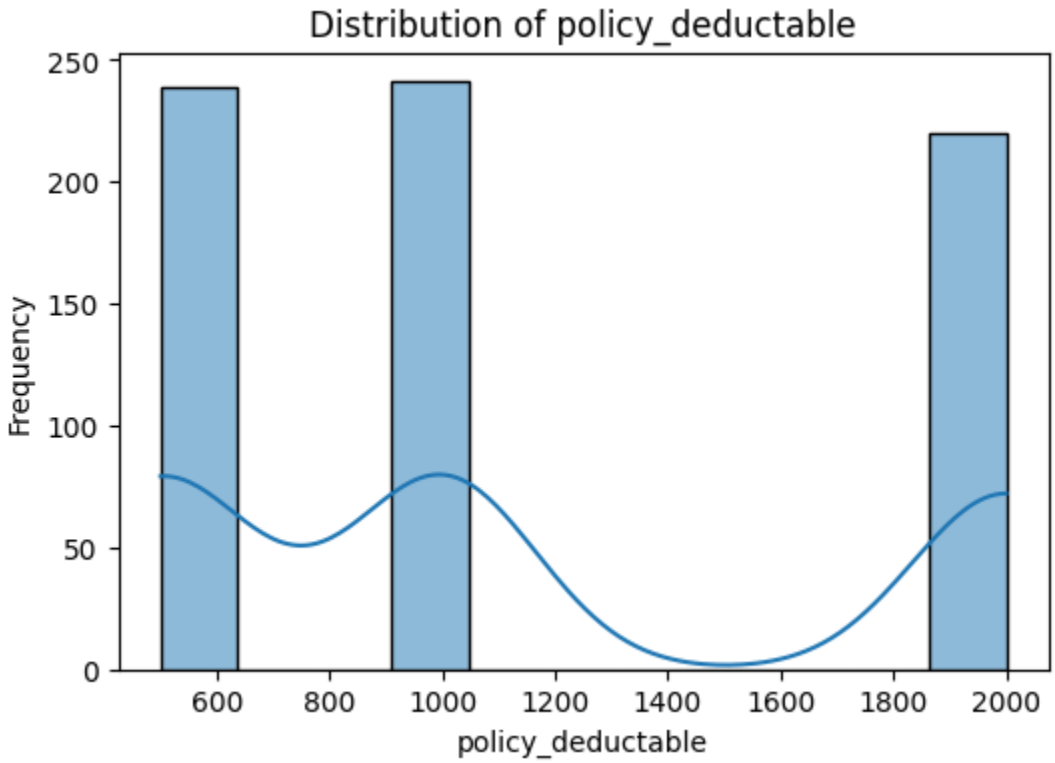
Analyse Numerical features

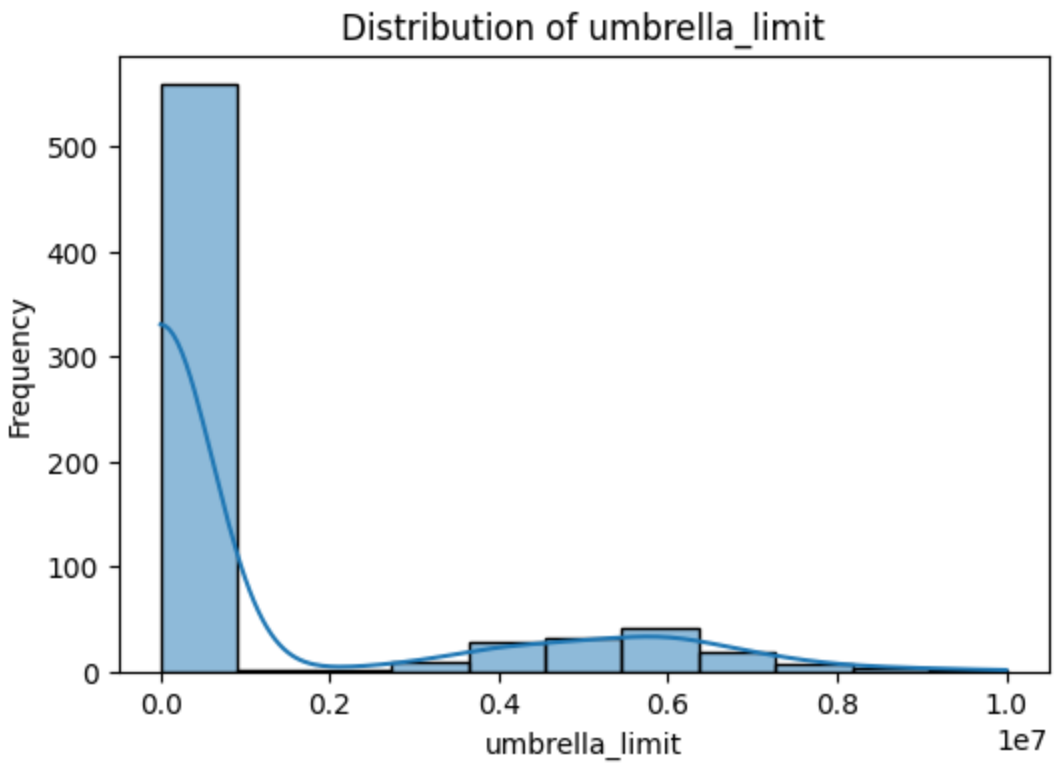
```python
# Plot all the numerical columns to understand their distribution
for col in num_cols:
    plt.figure(figsize=(6,4))
    sns.histplot(X_train[col], kde=True)
    plt.title(f'Distribution of {col}')
    plt.xlabel(col)
    plt.ylabel('Frequency')
    plt.show()
```

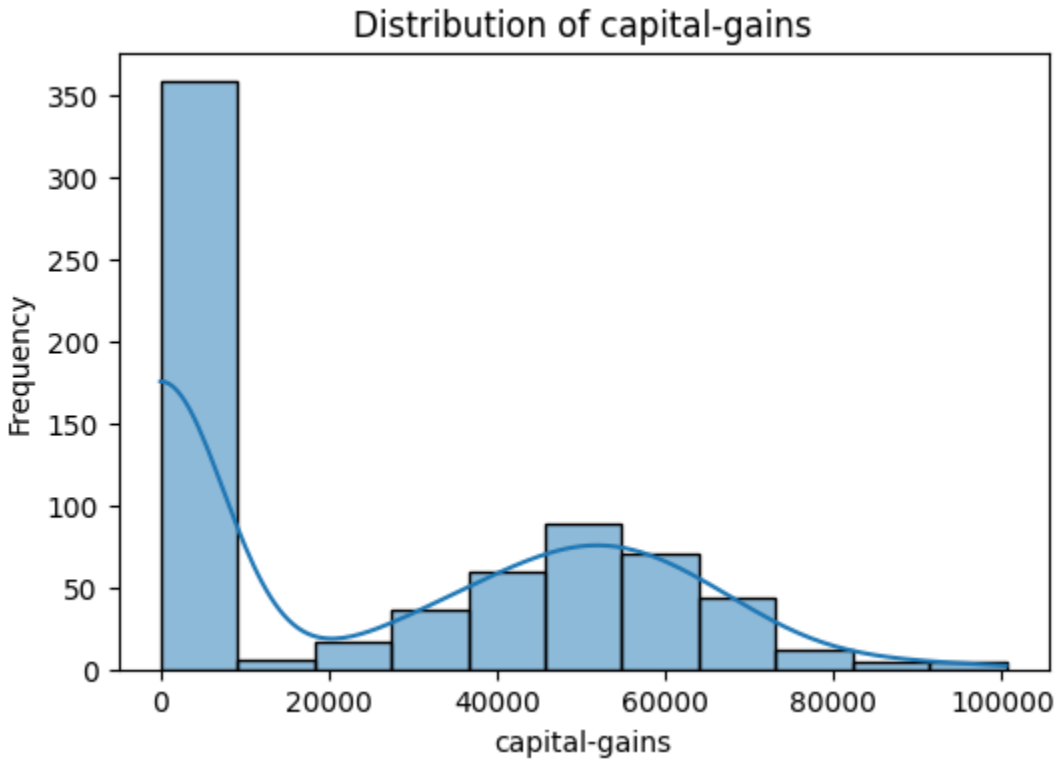
✓ 1.9s

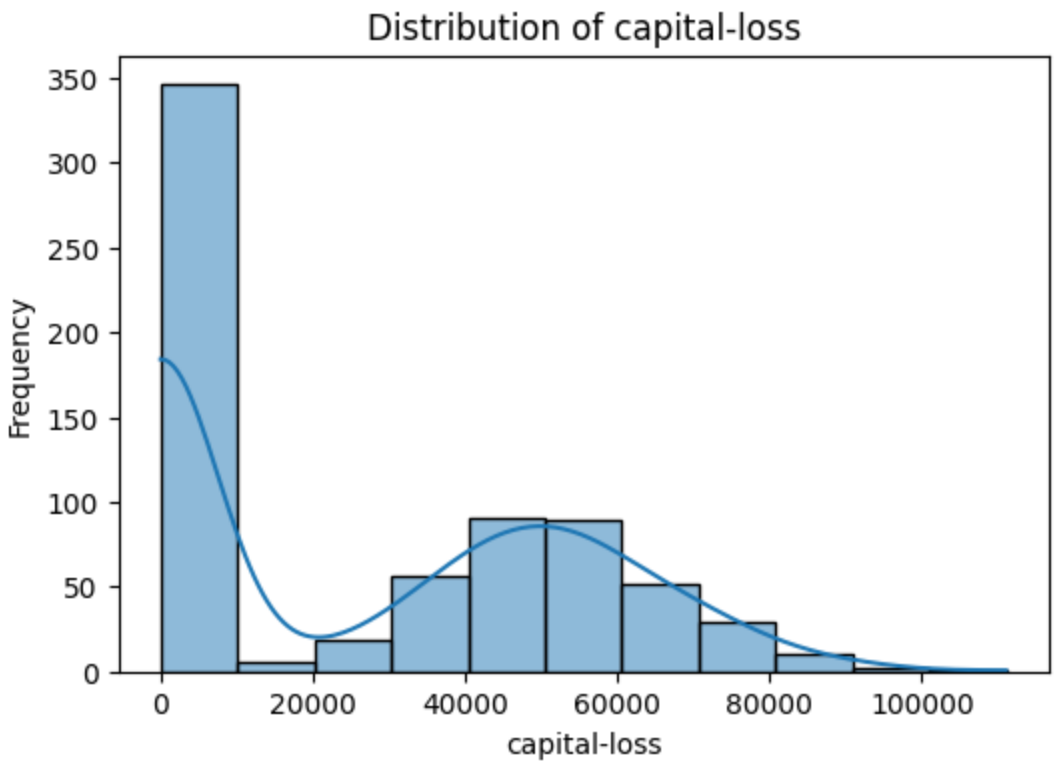


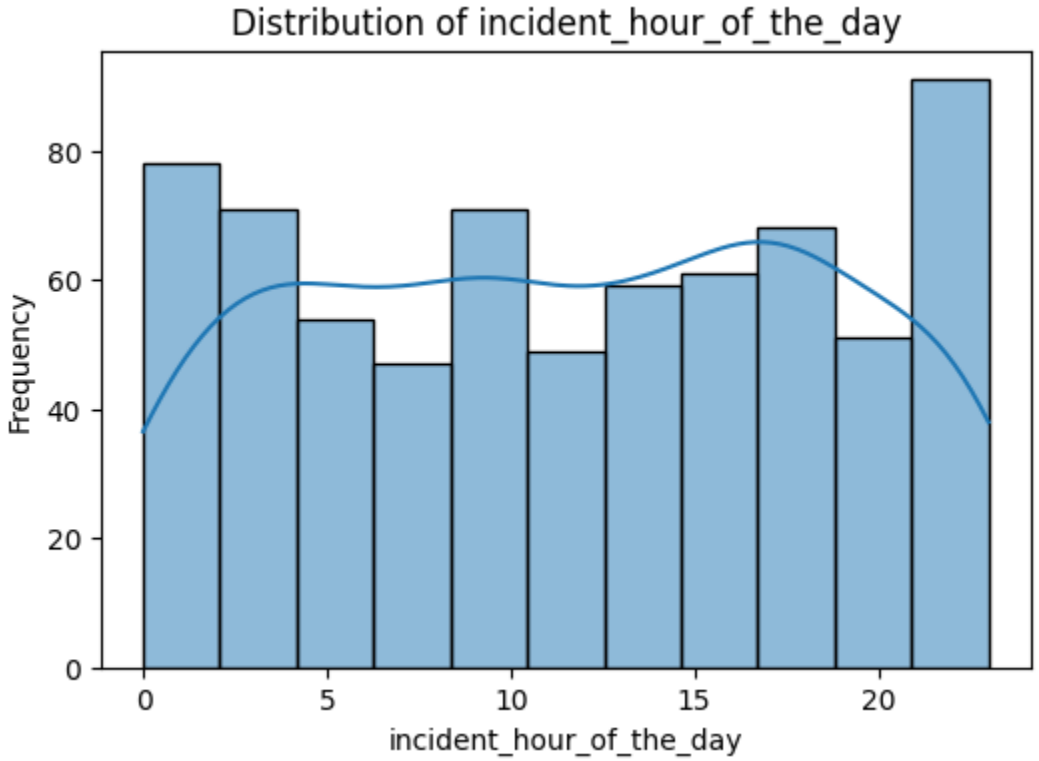

Distribution of age
Frequency
0
20
40
60
80
20
30
40
50
60
age

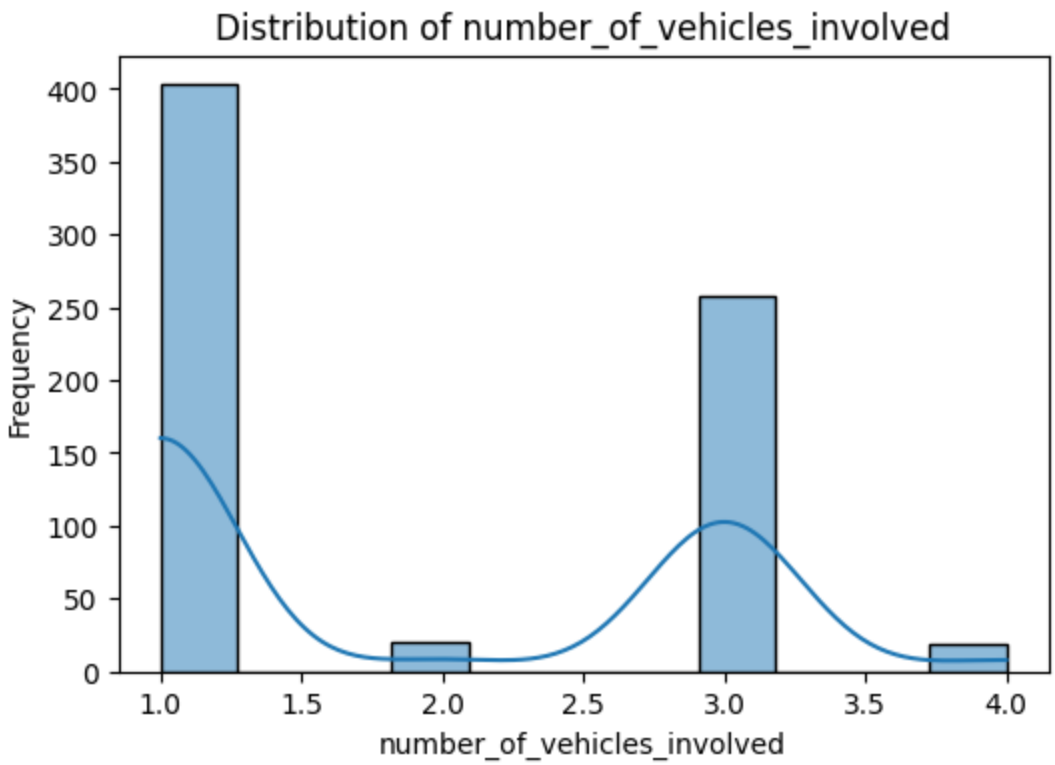

Distribution of policy_deductable
Frequency
0
50
100
150
200
250
600
800
1000
1200
1400
1600
1800
2000
policy_deductable

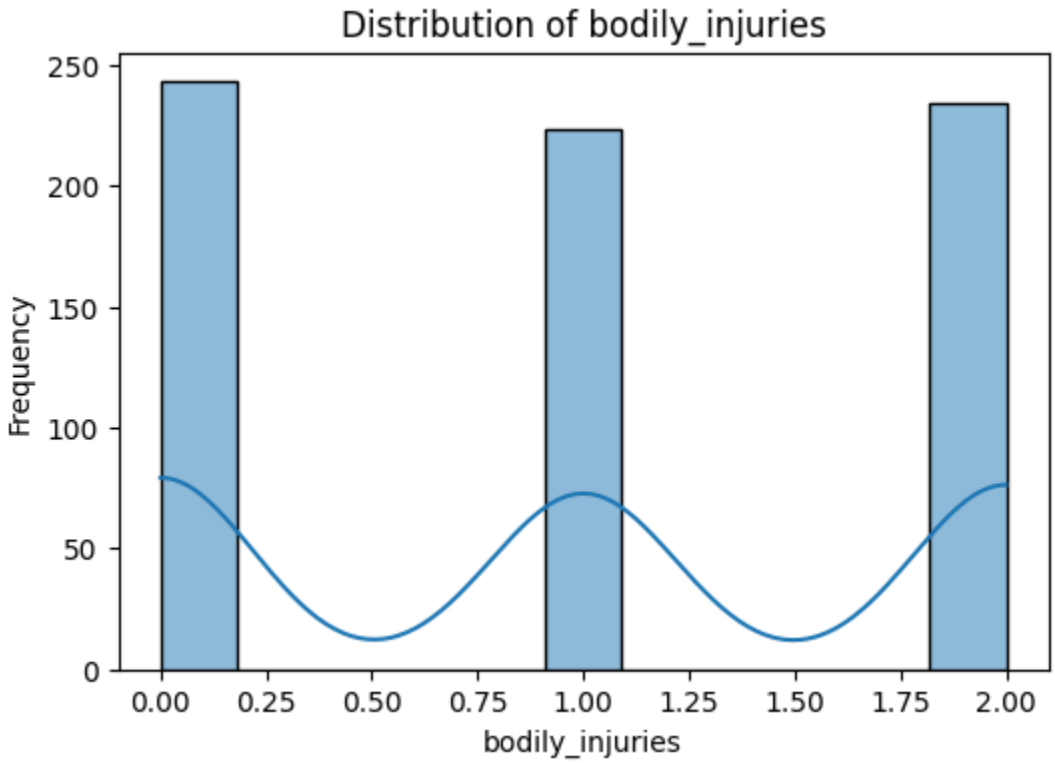

Distribution of umbrella_limit
Frequency
500
400
300
200
100
0
0.0
0.2
0.4
0.6
0.8
1.0
1e7
umbrella_limit

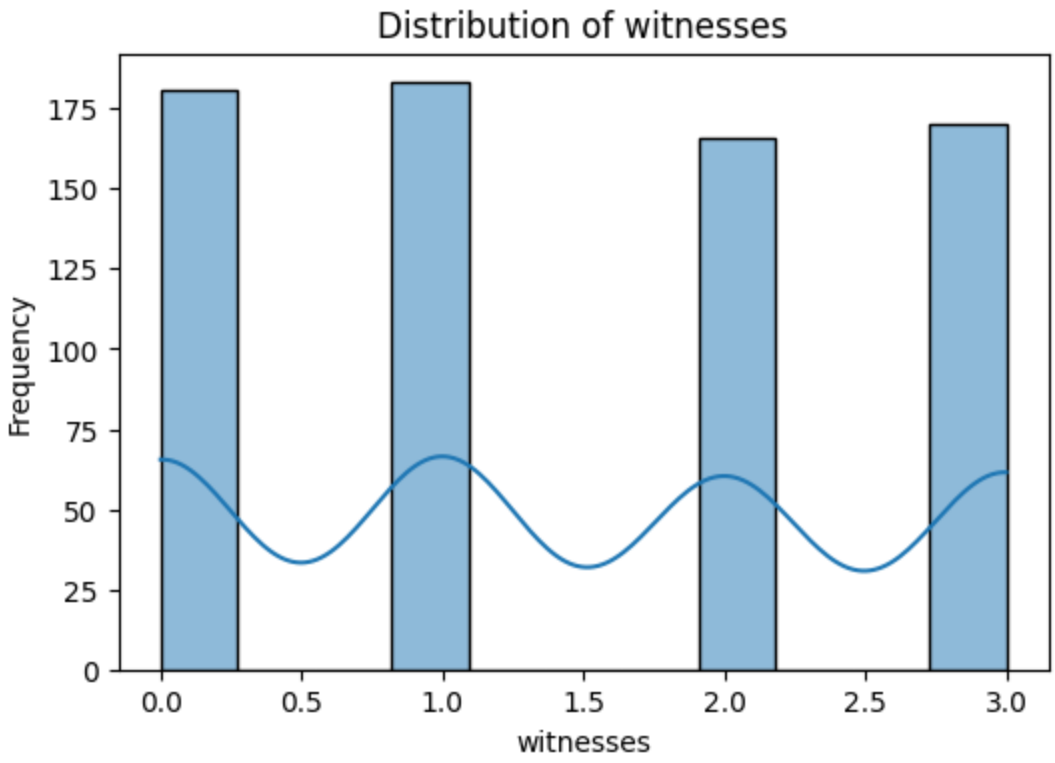

Distribution of capital-gains
Frequency
350
300
250
200
150
100
50
0
0
20000
40000
60000
80000
100000
capital-gains

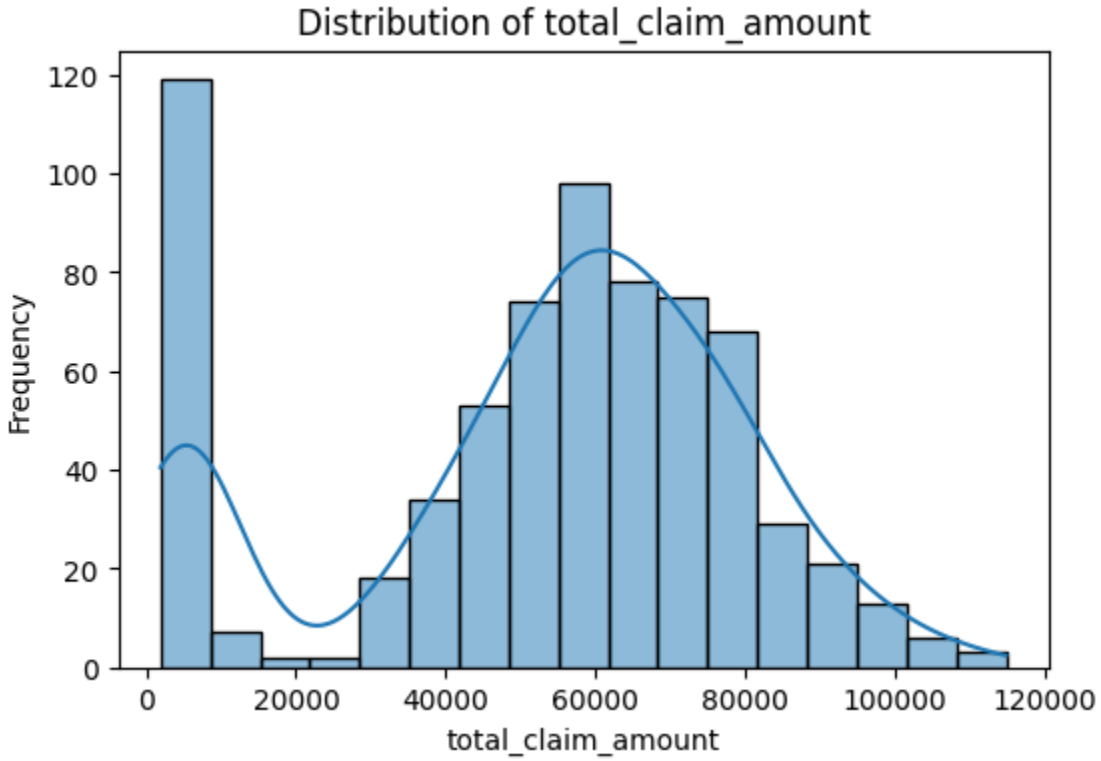

Distribution of capital-loss
Frequency
capital-loss
0
50
100
150
200
250
300
350
0
20000
40000
60000
80000
100000

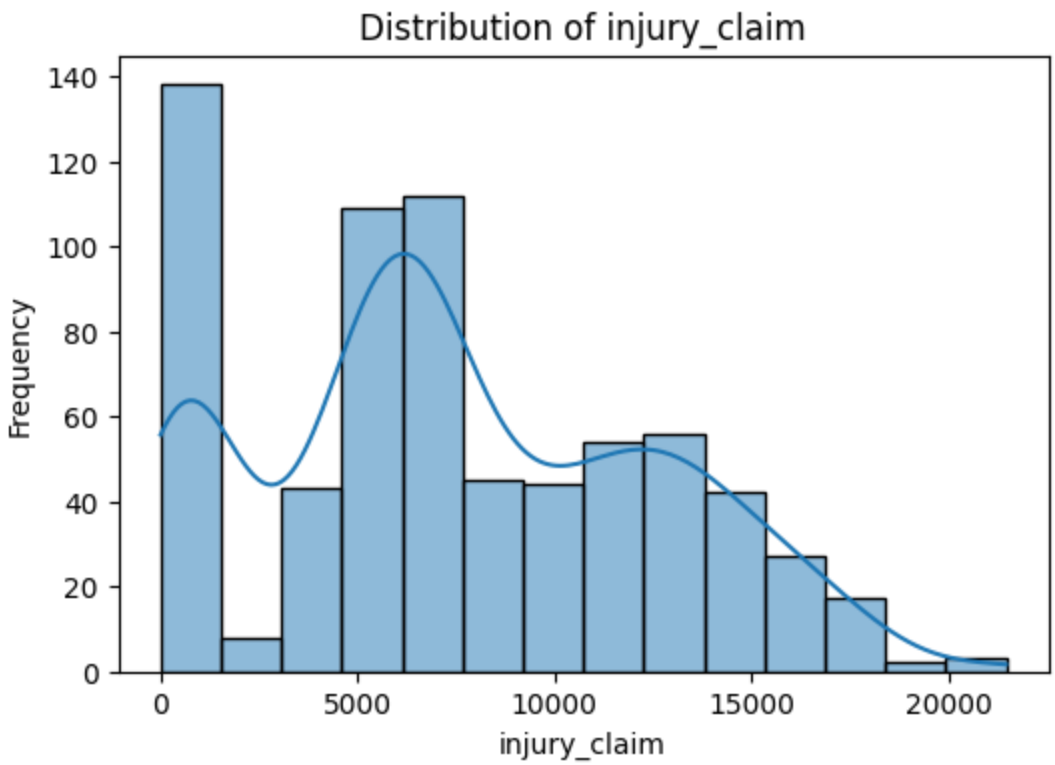

Distribution of incident_hour_of_the_day
Frequency
incident_hour_of_the_day
0
20
40
60
80
0
5
10
15
20

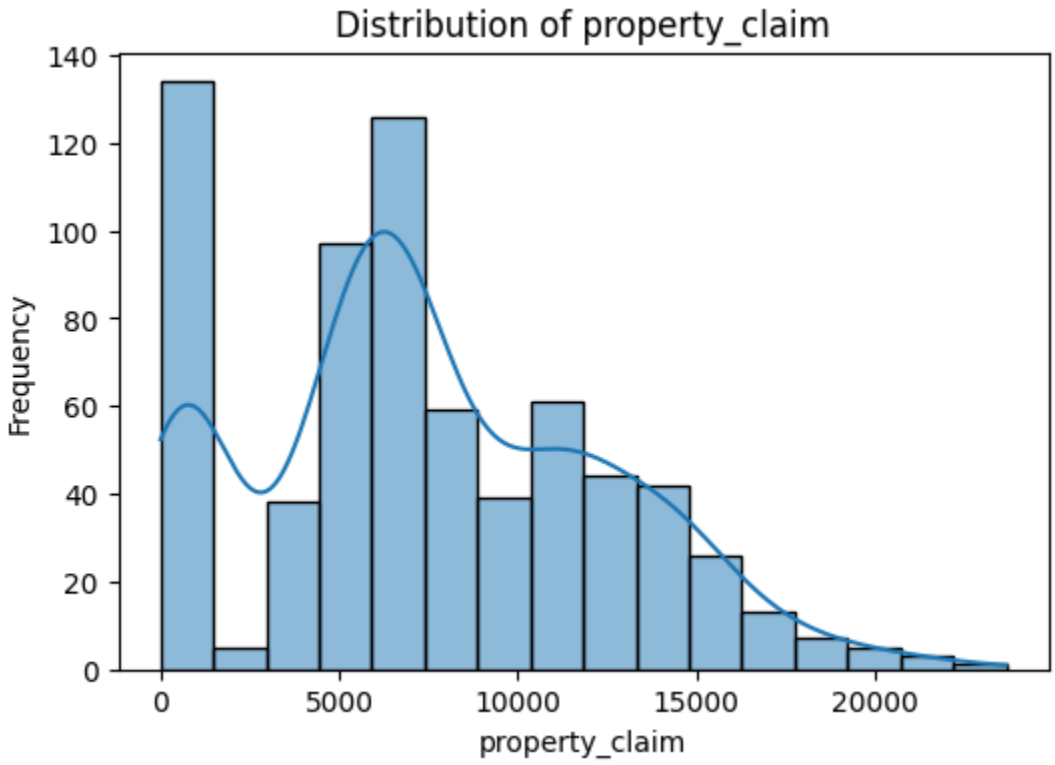

Distribution of number_of_vehicles_involved
Frequency
0
50
100
150
200
250
300
350
400
1.0
1.5
2.0
2.5
3.0
3.5
4.0
number_of_vehicles_involved

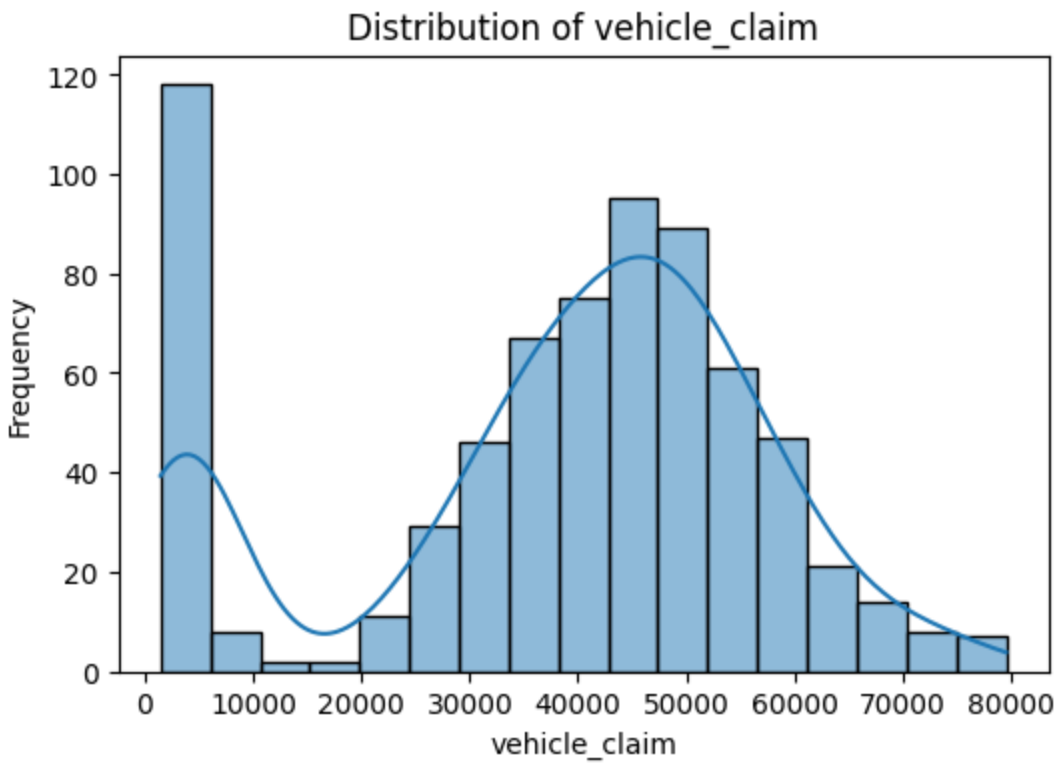

Distribution of bodily_injuries
Frequency
0
50
100
150
200
250
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
bodily_injuries

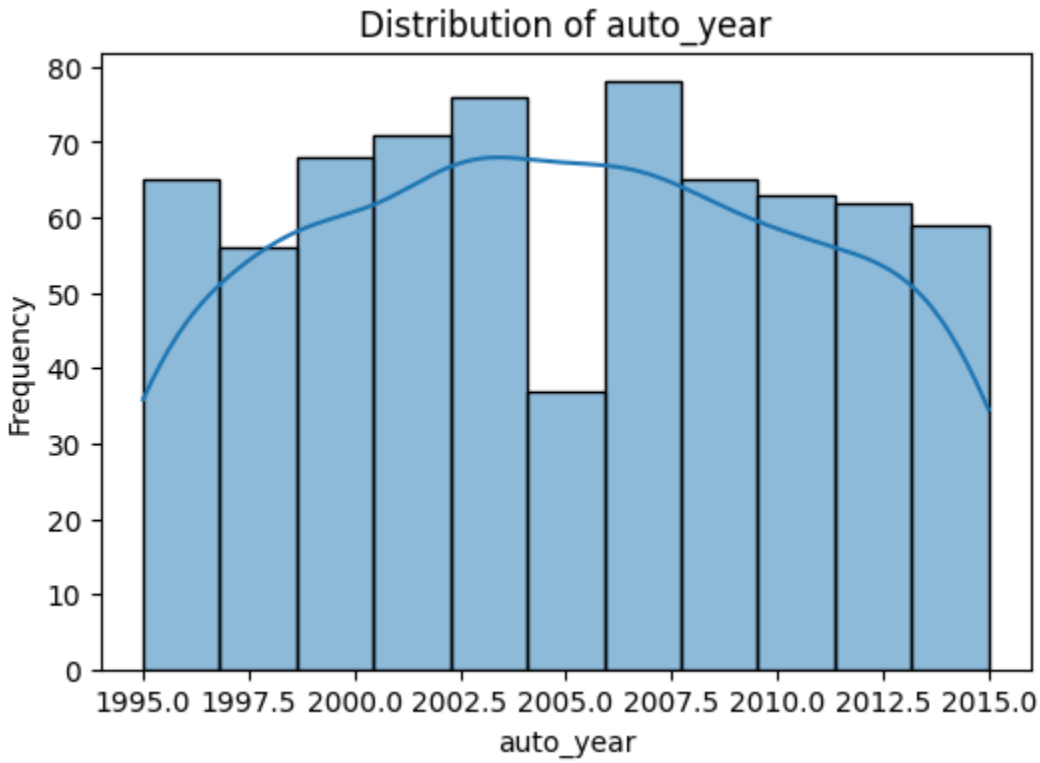

Distribution of witnesses
Frequency
witnesses
0
25
50
75
100
125
150
175
0.0
0.5
1.0
1.5
2.0
2.5
3.0

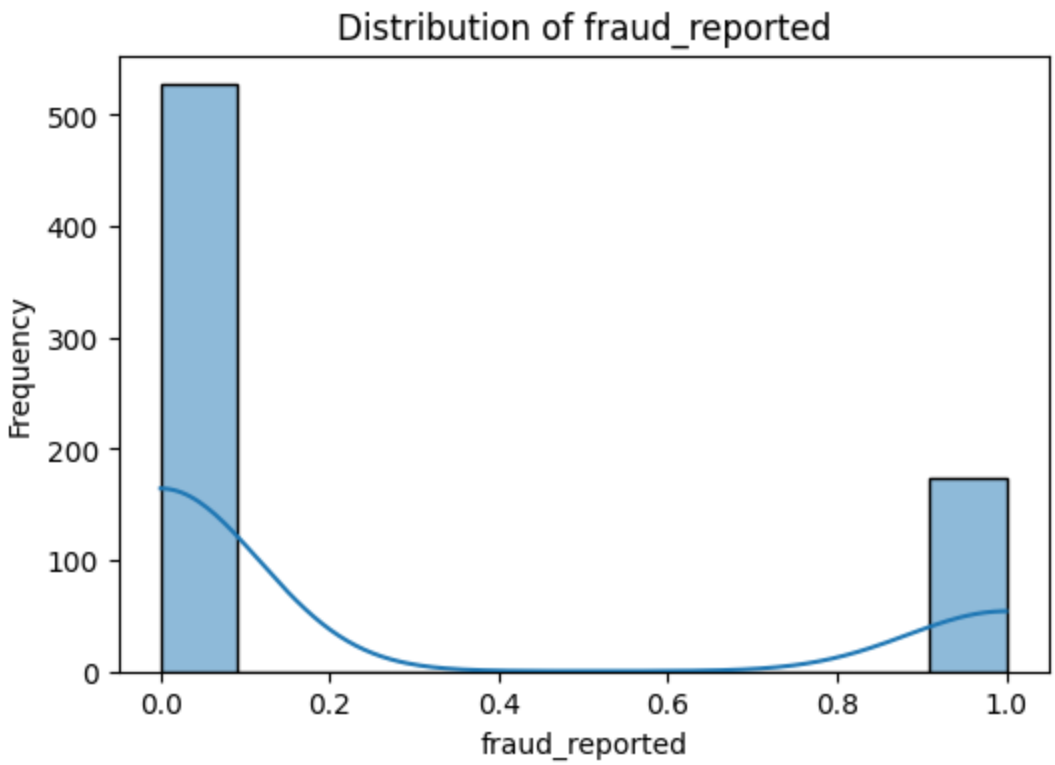

Distribution of total_claim_amount
Frequency
total_claim_amount
0
20
40
60
80
100
120
0
20000
40000
60000
80000
100000
120000


Distribution of injury_claim
Frequency
0
20
40
60
80
100
120
140
0
5000
10000
15000
20000
injury_claim


Distribution of property_claim
Frequency
0
20
40
60
80
100
120
140
0
5000
10000
15000
20000
property_claim


Distribution of vehicle_claim
Frequency
vehicle_claim
0
20
40
60
80
100
120
0
10000
20000
30000
40000
50000
60000
70000
80000

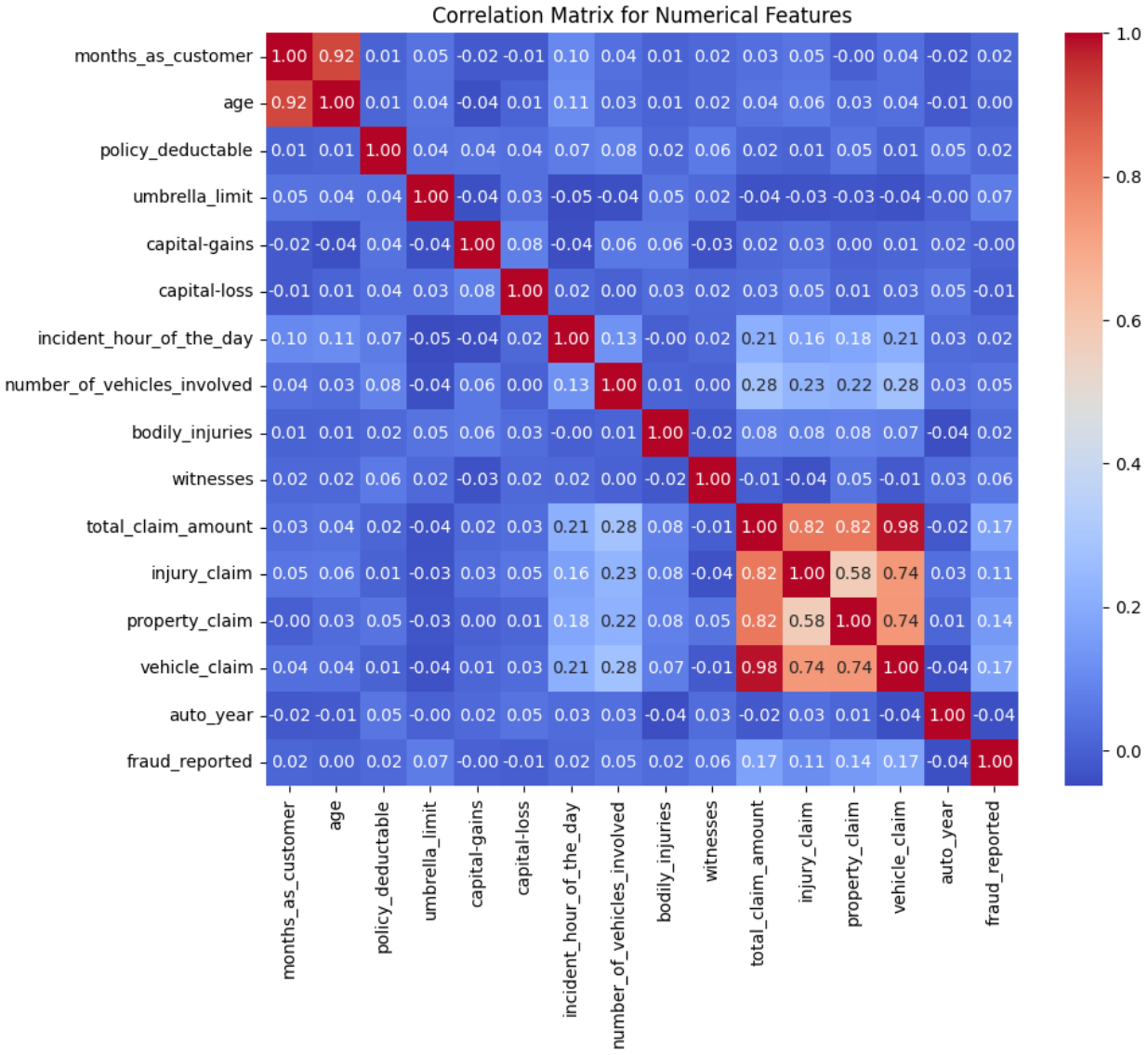

Distribution of auto_year
Frequency
auto_year
0
10
20
30
40
50
60
70
80
1995.0
1997.5
2000.0
2002.5
2005.0
2007.5
2010.0
2012.5
2015.0



Find Correlation among Numerical features to detect multicollinearity

```
# Create correlation matrix for numerical columns
corr_matrix = X_train[num_cols].corr()
corr_matrix
# Plot Heatmap of the correlation matrix
plt.figure(figsize=(10,8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Matrix for Numerical Features')
plt.show()

```

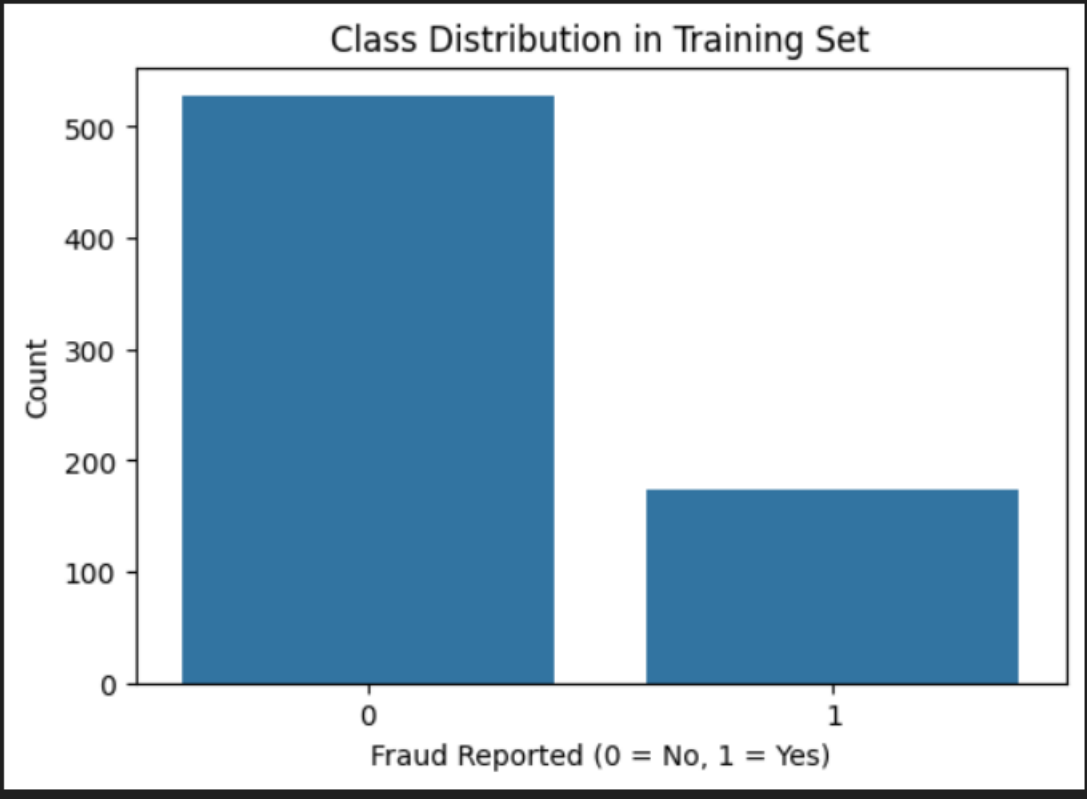
✓ 0.4s



Check Class Imbalance for Target Variable

```python
# Plot a bar chart to check class balance
plt.figure(figsize=(6,4))
sns.countplot(x=y_train)
plt.title('Class Distribution in Training Set')
plt.xlabel('Fraud Reported (0 = No, 1 = Yes)')
plt.ylabel('Count')
plt.show()

```

✓ 0.0s



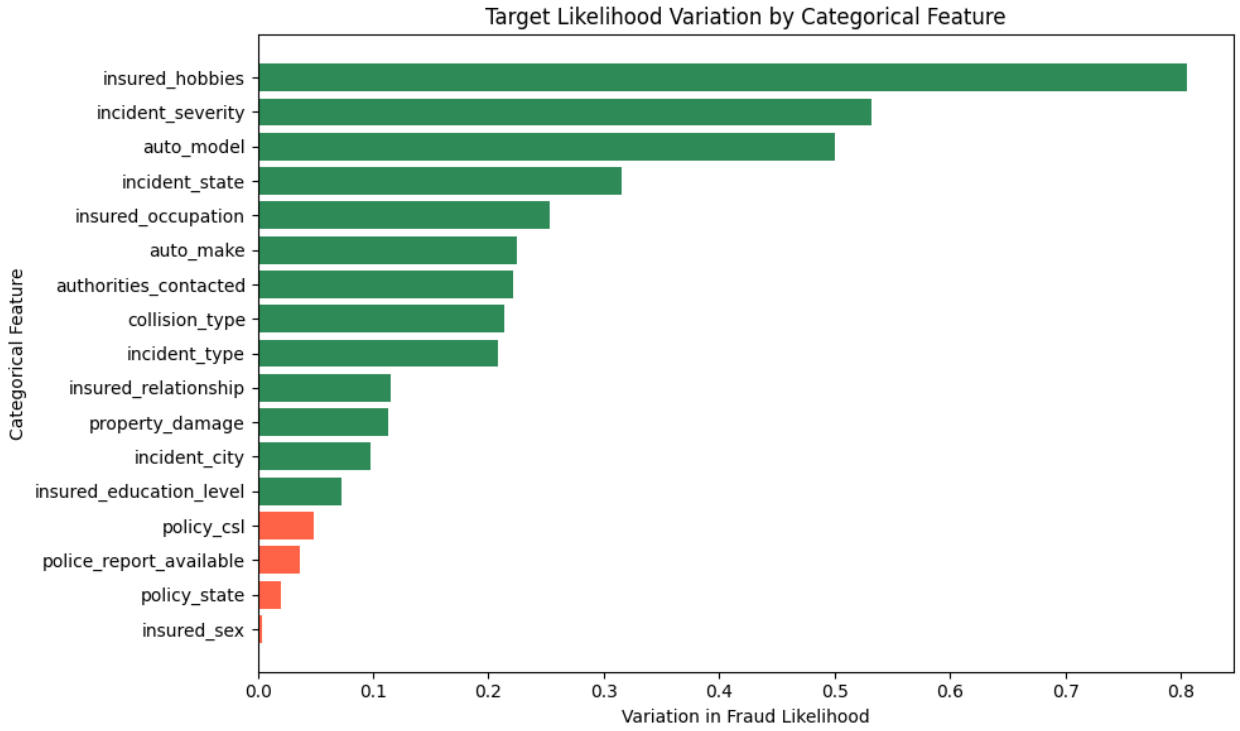
Target Likelihood Analysis for Categorical Features

Target Likelihood Variation Across Categorical Features:

	feature	min_likelihood	max_likelihood	variation	num_unique
5	insured_hobbies	0.073171	0.878788	0.805617	20
9	incident_severity	0.064516	0.596939	0.532423	4
16	auto_model	0.000000	0.500000	0.500000	39
11	incident_state	0.155405	0.470588	0.315183	7
4	insured_occupation	0.147059	0.400000	0.252941	14
15	auto_make	0.137255	0.361702	0.224447	14
10	authorities_contacted	0.083333	0.304965	0.221631	5
8	collision_type	0.087302	0.301435	0.214134	4
7	incident_type	0.084746	0.292419	0.207673	4
6	insured_relationship	0.175000	0.290076	0.115076	6
13	property_damage	0.176230	0.289157	0.112927	3
12	incident_city	0.206186	0.303922	0.097736	7
3	insured_education_level	0.205357	0.277778	0.072421	7
1	policy_csl	0.228155	0.276860	0.048704	3
14	police_report_available	0.224299	0.260331	0.036032	3
0	policy_state	0.238683	0.258475	0.019791	3
2	insured_sex	0.245399	0.248663	0.003264	2

Weak Categorical Features (Low variation in fraud likelihood):
['policy_csl', 'police_report_available', 'policy_state', 'insured_sex']

Categorical Features sorted by descending order for Target Likelihood variation

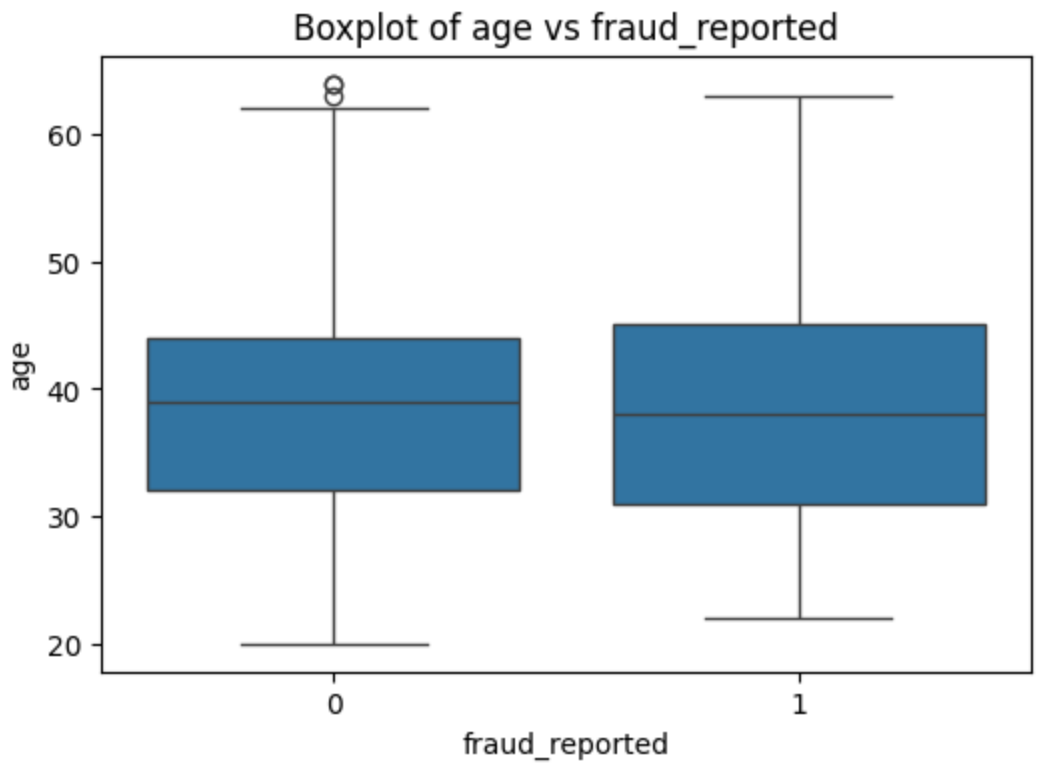


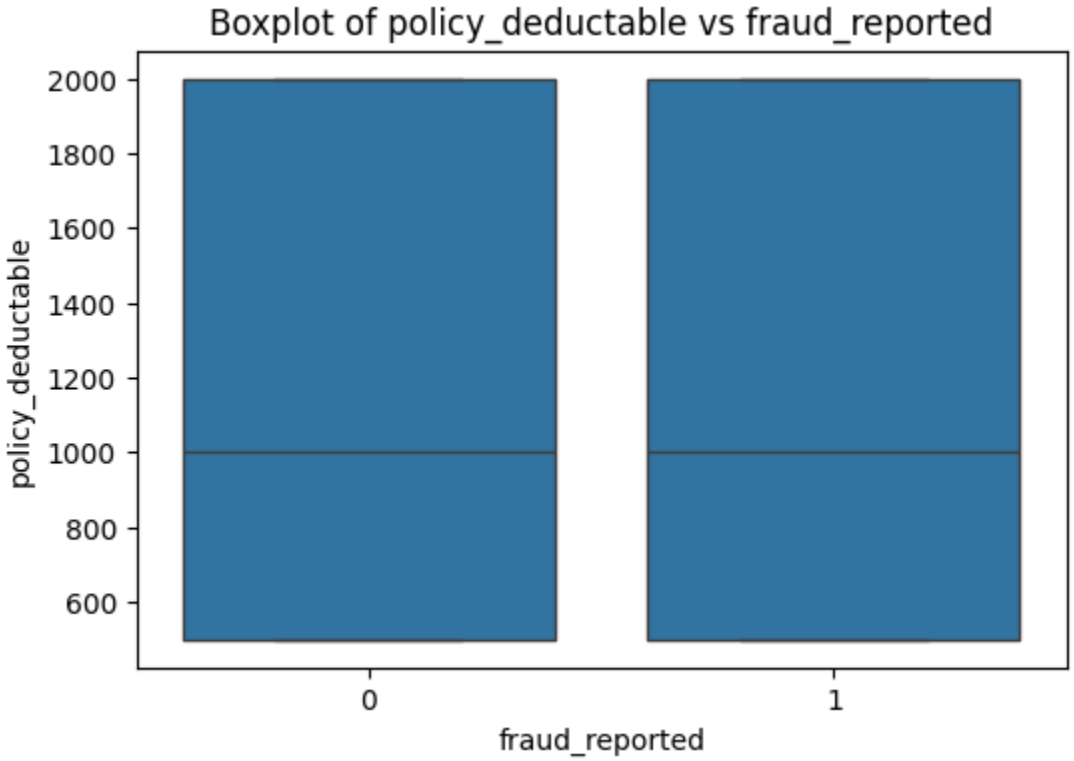
Hobbies and Severity are top categorical features based on Target Likelihood Variation Analysis

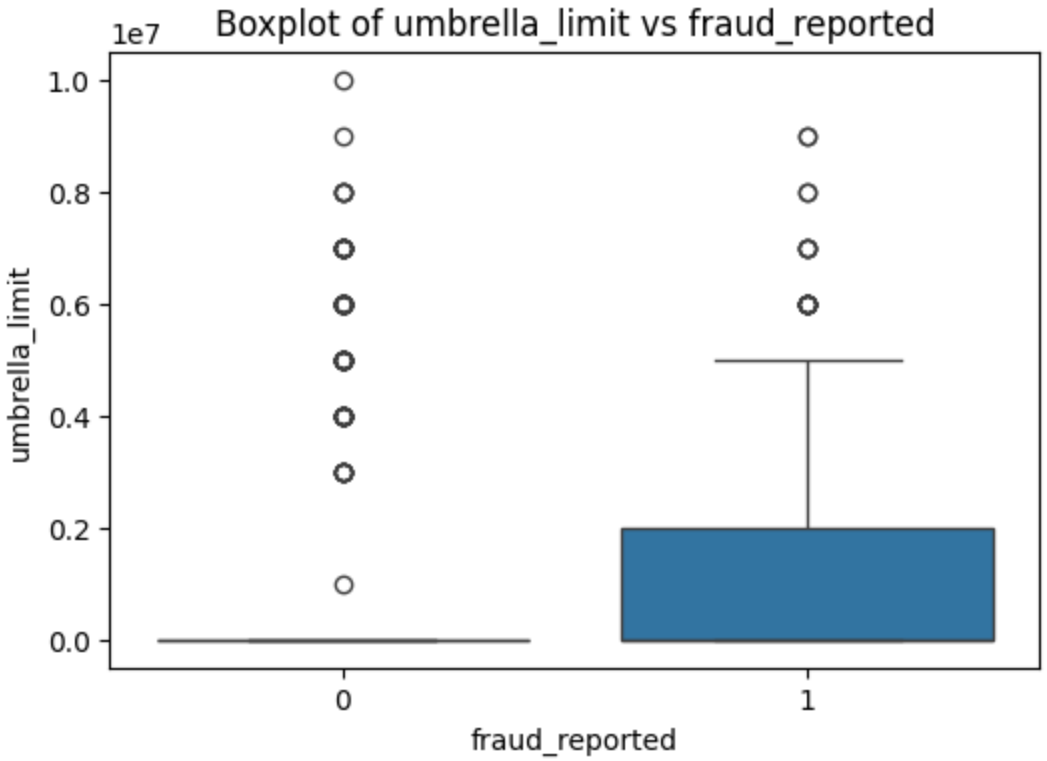
Charts to analyze Numerical features and Target Variable

```python
# Visualise the relationship between numerical features and the target variable to understand their impact on the target outcome

# Visualise the relationship between numerical features num_cols and target variable
import matplotlib.pyplot as plt
import seaborn as sns

# Assuming `num_cols` is a list of numeric column names
num_cols = df.select_dtypes(include=['int64', 'float64']).columns.tolist()
target = 'fraud_reported'   # replace with your target column

# Create visualizations
for col in num_cols:
    plt.figure(figsize=(6,4))
    sns.boxplot(x=target, y=col, data=X_train)
    plt.title(f"Boxplot of {col} vs {target}")
    plt.show()
```

✓ 1.6s

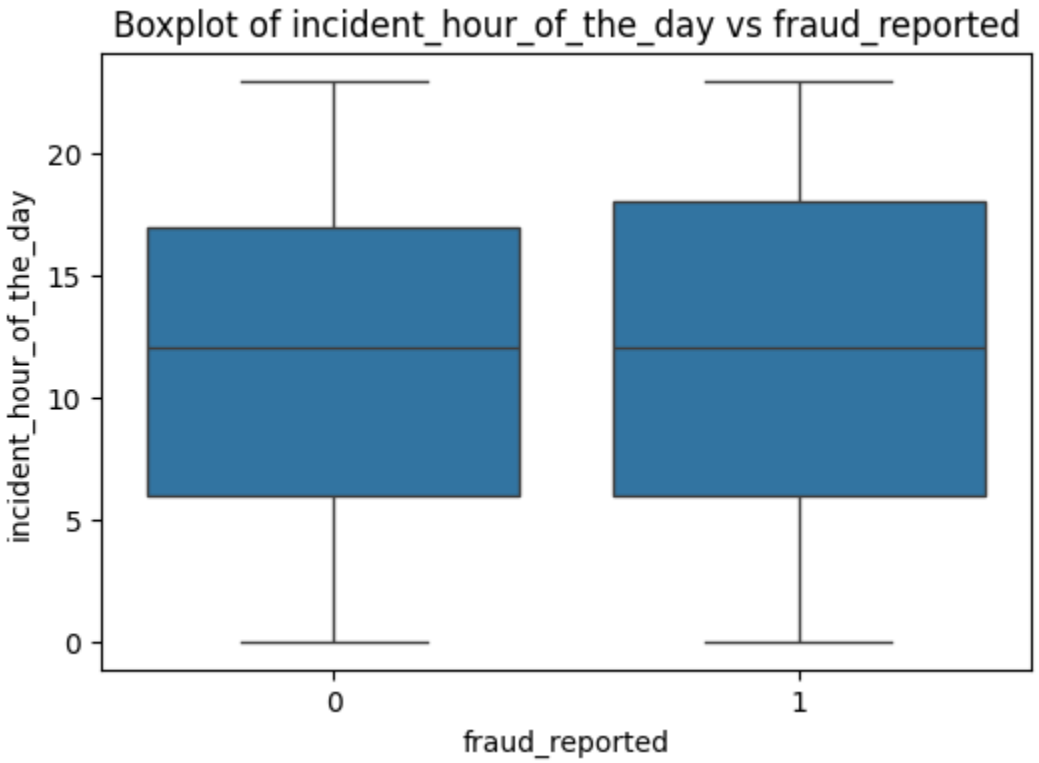

Boxplot of age vs fraud_reported
age
60
50
40
30
20
0
1
fraud_reported

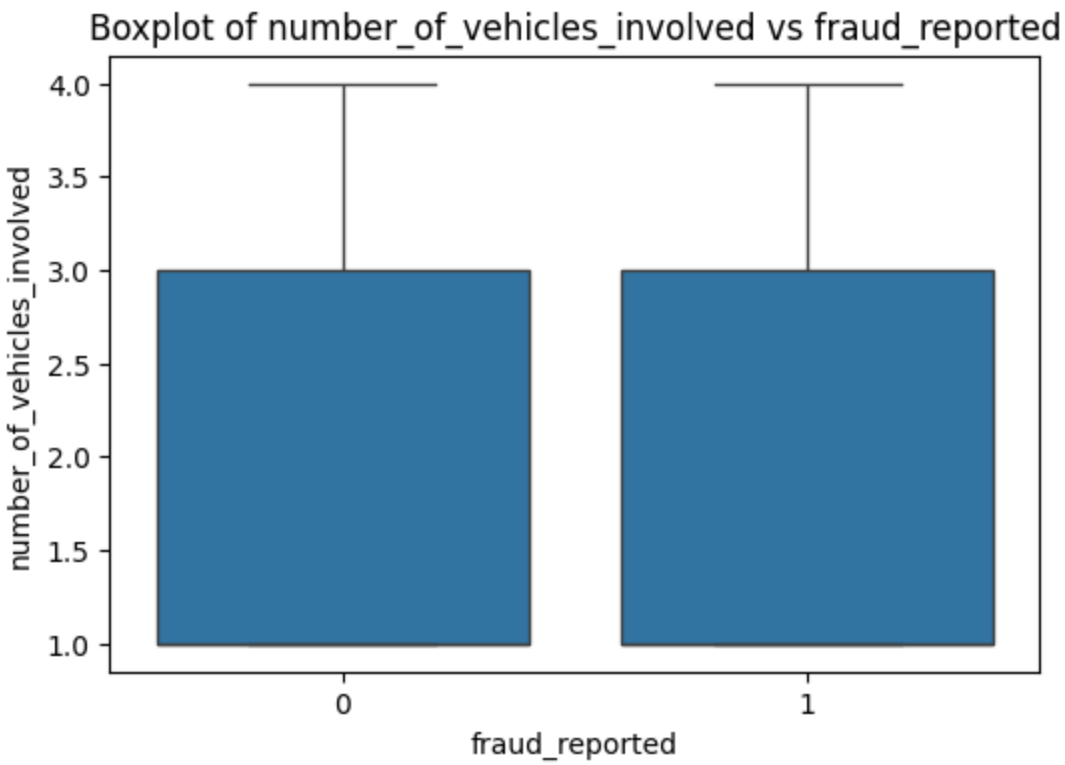

Boxplot of policy_deductable vs fraud_reported
policy_deductable
2000
1800
1600
1400
1200
1000
800
600
0
1
fraud_reported

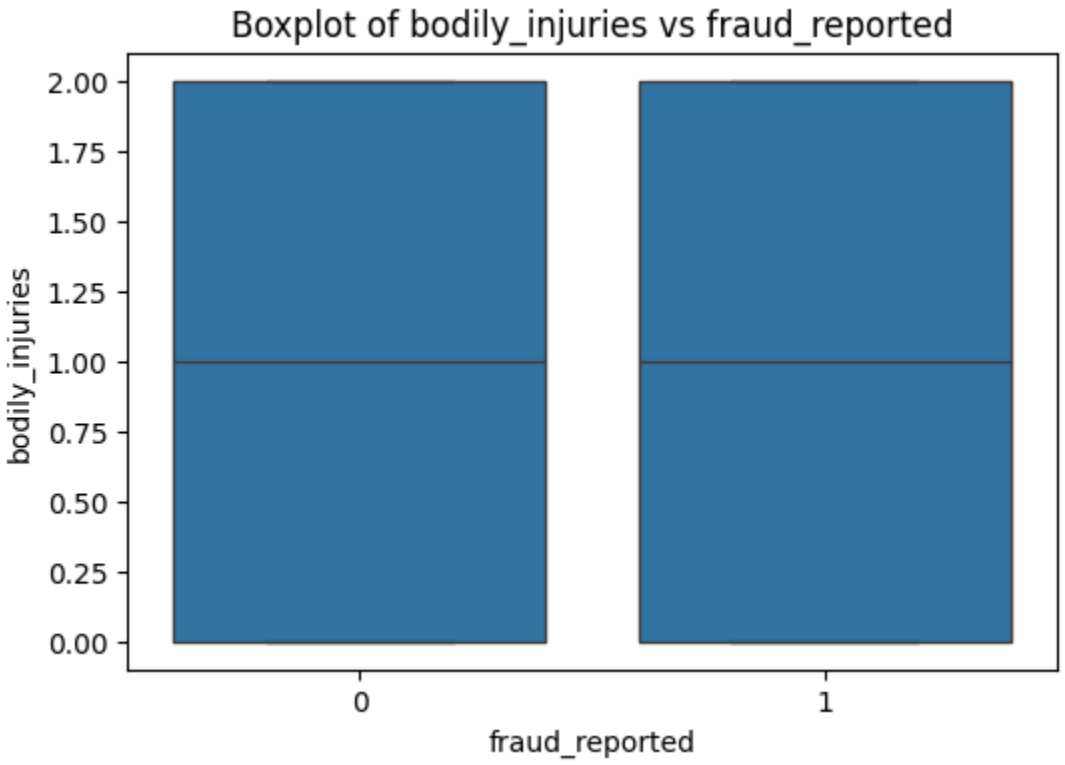

Boxplot of umbrella_limit vs fraud_reported
1e7
1.0
0.8
0.6
0.4
0.2
0.0
umbrella_limit
0
1
fraud_reported

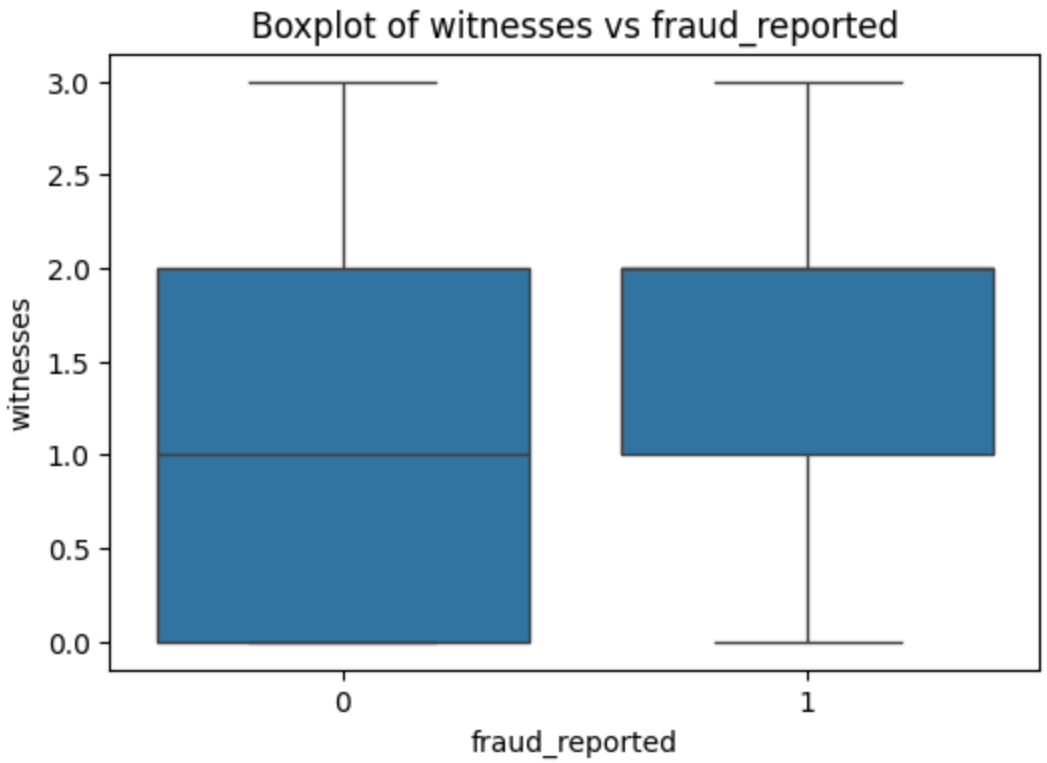

Boxplot of capital-gains vs fraud_reported
100000
80000
60000
40000
20000
0
capital-gains
0
1
fraud_reported

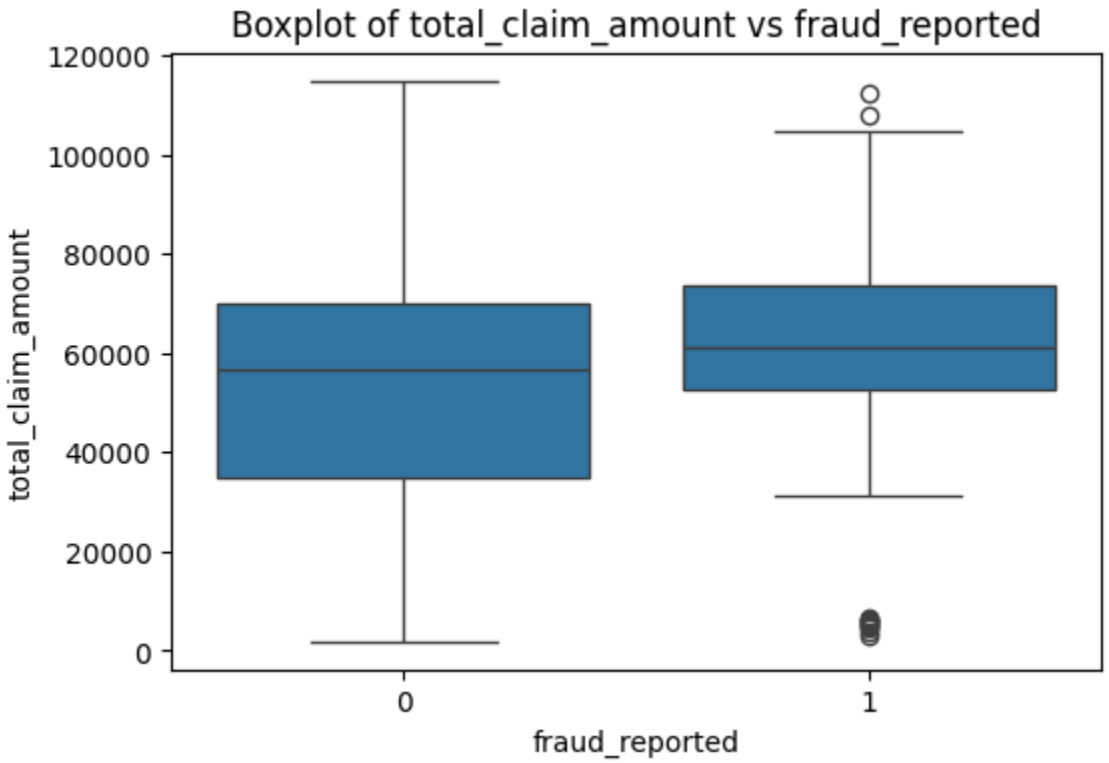

Boxplot of capital-loss vs fraud_reported
capital-loss
100000
80000
60000
40000
20000
0
0
1
fraud_reported

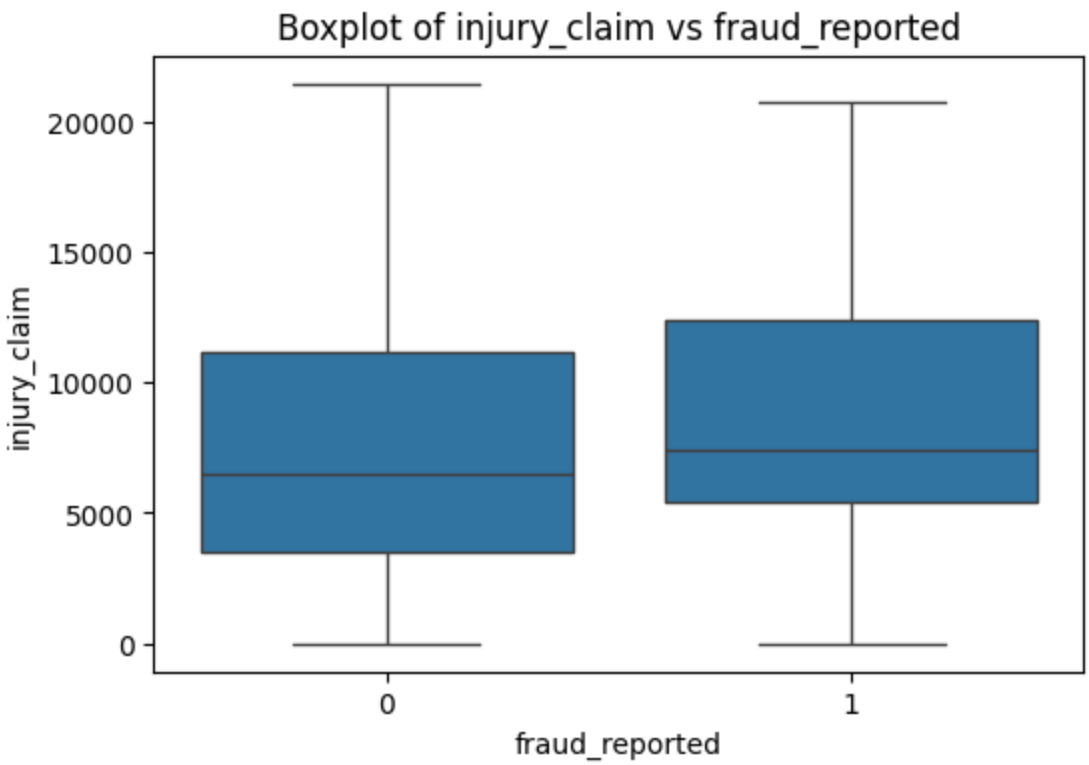

Boxplot of incident_hour_of_the_day vs fraud_reported
incident_hour_of_the_day
20
15
10
5
0
0
1
fraud_reported


Boxplot of number_of_vehicles_involved vs fraud_reported
number_of_vehicles_involved
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0
1
fraud_reported

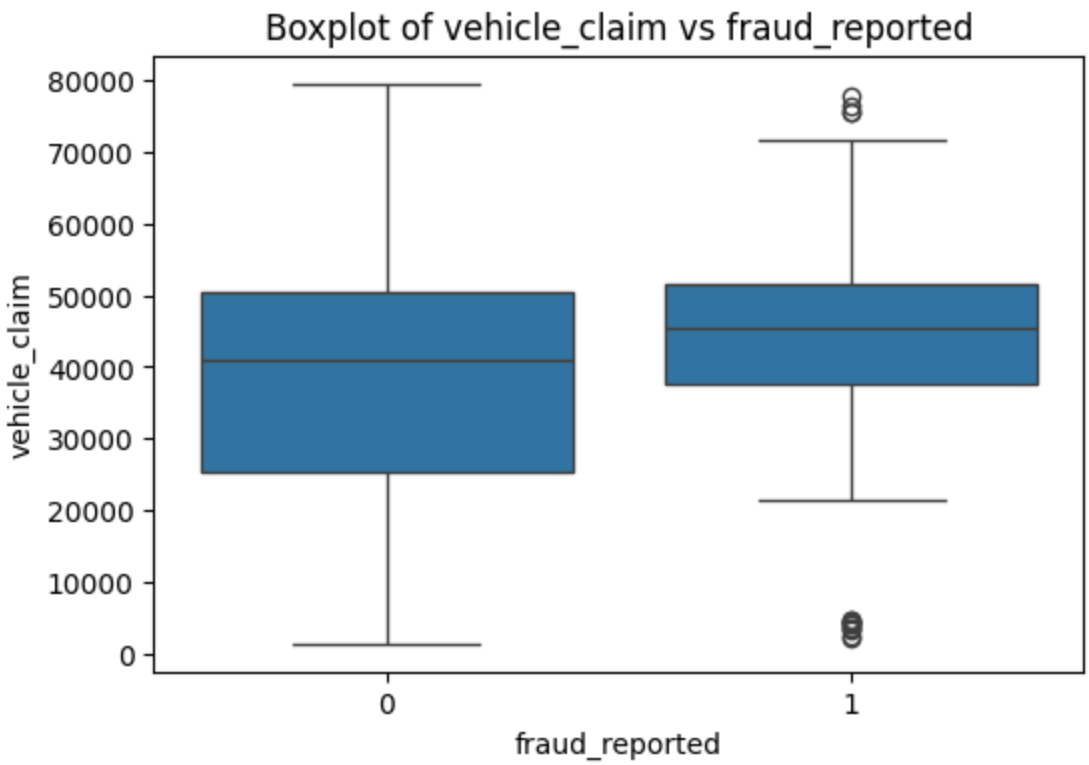

Boxplot of bodily_injuries vs fraud_reported
bodily_injuries
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
1
fraud_reported

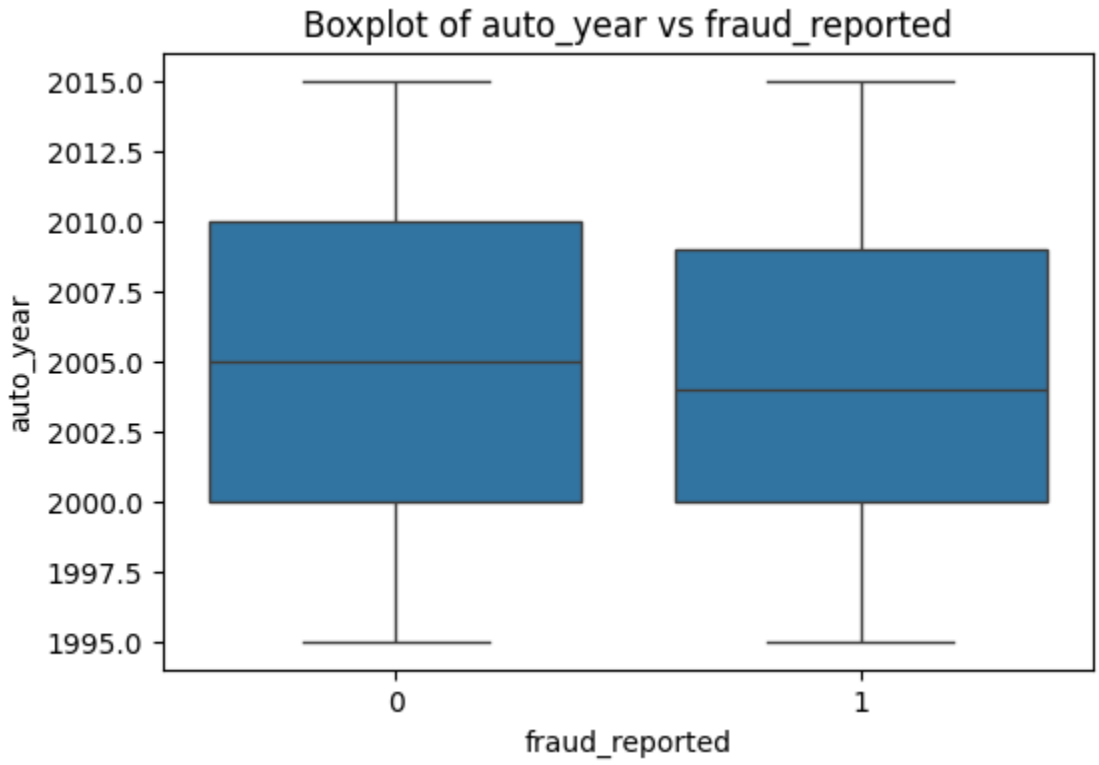

Boxplot of witnesses vs fraud_reported
witnesses
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
1
fraud_reported


Boxplot of total_claim_amount vs fraud_reported
total_claim_amount
120000
100000
80000
60000
40000
20000
0
0
1
fraud_reported


Boxplot of injury_claim vs fraud_reported
injury_claim
20000
15000
10000
5000
0
0
1
fraud_reported


Boxplot of property_claim vs fraud_reported
property_claim
20000
15000
10000
5000
0
0
1
fraud_reported





Amounts claimed are slightly higher in Fraud cases than non-fraud cases.

Capital-Loss is slightly lower in Fraud cases than non-fraud cases.

Fraud cases are around 25 % of total population

Over sample the minority class

```python
# Import RandomOverSampler from imblearn library
from imblearn.over_sampling import RandomOverSampler

# Perform resampling on training data
from collections import Counter
# Before balancing
print("Before:", Counter(y_train))

# Apply Random Over Sampling
ros = RandomOverSampler(random_state=42)
X_train_resampled, y_train_resampled = ros.fit_resample(X_train, y_train)

# After balancing
print("After:", Counter(y_train_resampled))
```

✓ 0.3s

```
Before: Counter({0: 527, 1: 173})
After: Counter({0: 527, 1: 527})
```

```python
# Oversample Testing data set for minority class
from imblearn.over_sampling import RandomOverSampler

# Perform resampling on training data
from collections import Counter
# Before balancing
print("Before:", Counter(y_test))

# Apply Random Over Sampling
ros = RandomOverSampler(random_state=42)
X_test_resampled, y_test_resampled = ros.fit_resample(X_test, y_test)

# After balancing
print("After:", Counter(y_test_resampled))
```

✓ 0.0s

```
Before: Counter({0: 226, 1: 74})
After: Counter({0: 226, 1: 226})
```

```
1  X_train_resampled.shape
```

✓ 0.0s

```
(1054, 34)
```

Create new features

```
# Create new features based on your understanding for both training and validation data
#Create new feature policy_csl_value from policy_csl

X_train_resampled['policy_csl_value'] = X_train_resampled['policy_csl'].str.split('/').apply(lambda x: int(x[1]) if len(x) > 1 else int(x[0]))

# Example
X_train_resampled['claim_to_csl_ratio'] = X_train_resampled['total_claim_amount'] / X_train_resampled['policy_csl_value']
X_train_resampled['injury_to_property_ratio'] = X_train_resampled['injury_claim'] / (X_train_resampled['property_claim'] + 1)

X_train_resampled['injury_to_vehicle_ratio'] = X_train_resampled['injury_claim'] / (X_train_resampled['vehicle_claim'] + 1)
X_train_resampled['claim_to_deductable_ratio'] = X_train_resampled['total_claim_amount'] / (X_train_resampled['policy_deductable'] + 1)

X_train_resampled['injury_x_csl'] = X_train_resampled['injury_claim'] * X_train_resampled['policy_csl_value']
X_train_resampled['vehicle_x_csl'] = X_train_resampled['vehicle_claim'] * X_train_resampled['policy_csl_value']

X_train_resampled['injury_x_csl'] = X_train_resampled['injury_claim'] * X_train_resampled['policy_csl_value']
X_train_resampled['vehicle_x_property'] = X_train_resampled['vehicle_claim'] * X_train_resampled['property_claim']

X_train_resampled['age_x_csl'] = X_train_resampled['age'] * X_train_resampled['policy_csl_value']
X_train_resampled['age_to_claim_ratio'] = X_train_resampled['age'] / (X_train_resampled['total_claim_amount'] + 1)

X_train_resampled['month_sin'] = np.sin(2 * np.pi * X_train_resampled['incident_date'].dt.month / 12)
X_train_resampled['csl_x_month'] = X_train_resampled['policy_csl_value'] * X_train_resampled['month_sin']

#df["policy_csl_value"].value_counts()
```

✓ 0.0s

```
X_train_resampled["incident_date"] = pd.to_datetime(X_train_resampled["incident_date"])
X_train_resampled["incident_year"] = X_train_resampled["incident_date"].dt.year
X_train_resampled["incident_month"] = X_train_resampled["incident_date"].dt.month
X_train_resampled["incident_dayofweek"] = X_train_resampled["incident_date"].dt.dayofweek
X_train_resampled["is_weekend"] = X_train_resampled["incident_dayofweek"].isin([5,6]).astype(int)
X_train_resampled.drop(columns=["incident_date"], inplace=True)

```

✓ 0.1s

```
X_test_resampled['policy_csl_value'] = df['policy_csl'].str.split('/').apply(lambda x: int(x[1]) if len(x) > 1 else int(x[0]))

# Example
X_test_resampled['claim_to_csl_ratio'] = X_test_resampled['total_claim_amount'] / X_test_resampled['policy_csl_value']
X_test_resampled['injury_to_property_ratio'] = X_test_resampled['injury_claim'] / (X_test_resampled['property_claim'] + 1)

X_test_resampled['injury_to_vehicle_ratio'] = X_test_resampled['injury_claim'] / (X_test_resampled['vehicle_claim'] + 1)
X_test_resampled['claim_to_deductable_ratio'] = X_test_resampled['total_claim_amount'] / (X_test_resampled['policy_deductable'] + 1)

X_test_resampled['injury_x_csl'] = X_test_resampled['injury_claim'] * X_test_resampled['policy_csl_value']
X_test_resampled['vehicle_x_csl'] = X_test_resampled['vehicle_claim'] * X_test_resampled['policy_csl_value']

X_test_resampled['injury_x_csl'] = X_test_resampled['injury_claim'] * X_test_resampled['policy_csl_value']
X_test_resampled['vehicle_x_property'] = X_test_resampled['vehicle_claim'] * X_test_resampled['property_claim']

X_test_resampled['age_x_csl'] = X_test_resampled['age'] * X_test_resampled['policy_csl_value']
X_test_resampled['age_to_claim_ratio'] = X_test_resampled['age'] / (X_test_resampled['total_claim_amount'] + 1)

X_test_resampled['month_sin'] = np.sin(2 * np.pi * X_test_resampled['incident_date'].dt.month / 12)
X_test_resampled['csl_x_month'] = X_test_resampled['policy_csl_value'] * X_test_resampled['month_sin']
```

[62] ✓ 0.0s

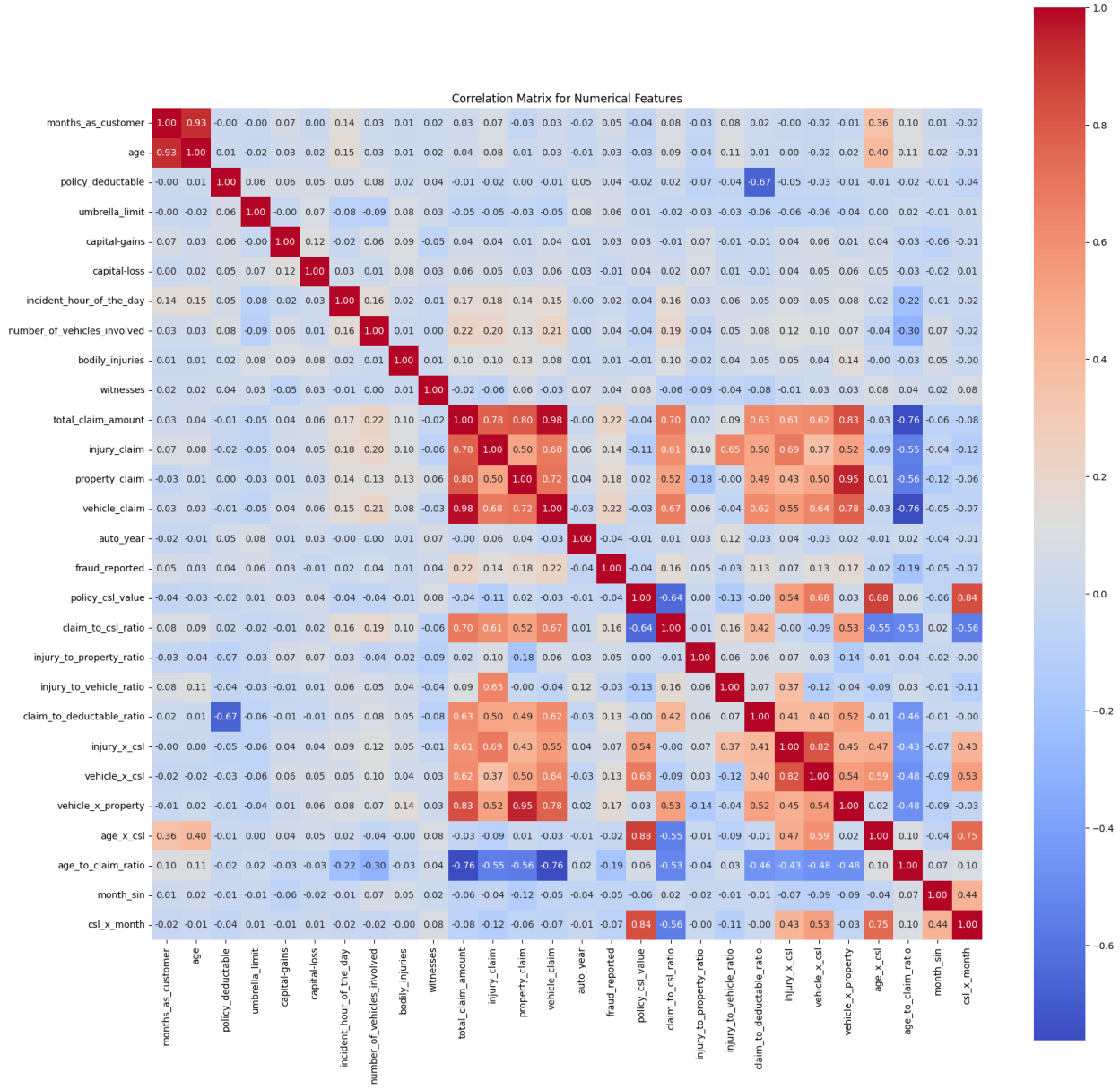
```
X_test_resampled['policy_csl_value'] = X_test_resampled['policy_csl'].str.split('/').apply(lambda x: int(x[1]) if len(x) > 1 else int(x[0]))
X_test_resampled["incident_date"] = pd.to_datetime(X_test_resampled["incident_date"])
X_test_resampled["incident_year"] = X_test_resampled["incident_date"].dt.year
X_test_resampled["incident_month"] = X_test_resampled["incident_date"].dt.month
X_test_resampled["incident_dayofweek"] = X_test_resampled["incident_date"].dt.dayofweek
X_test_resampled["is_weekend"] = X_test_resampled["incident_dayofweek"].isin([5,6]).astype(int)
X_test_resampled.drop(columns=["incident_date"], inplace=True)
```

[63] ✓ 0.0s

Check correlation for numerical features

```
# Create correlation matrix for numerical columns
# Select numerical columns
num_cols = X_train_resampled.select_dtypes(include=['int64', 'float64']).columns
num_cols
corr_matrix = X_train_resampled[num_cols].corr()
corr_matrix
# Plot Heatmap of the correlation matrix
plt.figure(figsize=(20,20))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Matrix for Numerical Features')
plt.show()

```

✓ 1.2s



Drop highly correlated features (cut-off .5)

```python
# Compute correlation matrix for numerical columns
#corr_matrix = X_train_resampled.corr().abs()
corr_matrix = X_train_resampled[num_cols].corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find columns with correlation greater than 0.9
to_drop = [column for column in upper.columns if any(upper[column] > 0.5)]

# Drop highly correlated columns
X_train_resampled.drop(columns=to_drop, inplace=True)

print("Dropped columns due to high correlation:", to_drop)

#visualize correlation matrix usig heat map


```

✓ 0.0s Python

```
Dropped columns due to high correlation: ['age', 'injury_claim', 'property_claim', 'vehicle_claim', 'claim_to_csl_ratio', 'injury_to_vehicle_ratio', 'claim_to_deductable_ratio', 'injury
```

Dropped columns due to high correlation: ['age', 'injury_claim', 'property_claim', 'vehicle_claim', 'claim_to_csl_ratio', 'injury_to_vehicle_ratio', 'claim_to_deductable_ratio', 'injury_x_csl', 'vehicle_x_csl', 'vehicle_x_property', 'age_x_csl', 'age_to_claim_ratio', 'csl_x_month']

```
1  #X.info()
2  X_train_resampled.info()
```

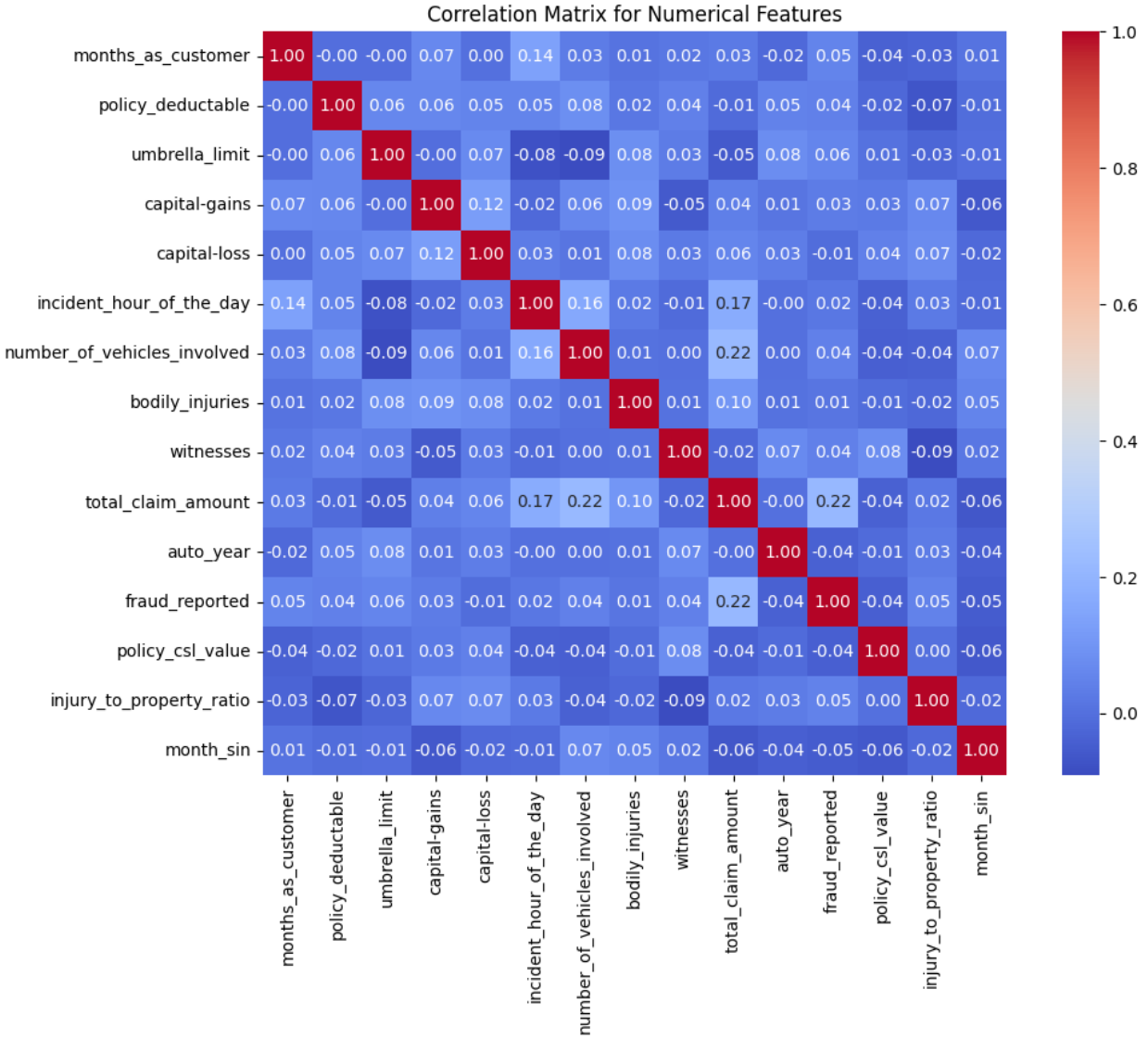
✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1054 entries, 0 to 1053
Data columns (total 36 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1054 non-null   int64
 1   policy_state                 1054 non-null   object
 2   policy_csl                   1054 non-null   object
 3   policy_deductable            1054 non-null   int64
 4   umbrella_limit               1054 non-null   int64
 5   insured_sex                  1054 non-null   object
 6   insured_education_level      1054 non-null   object
 7   insured_occupation           1054 non-null   object
 8   insured_hobbies              1054 non-null   object
 9   insured_relationship         1054 non-null   object
 10  capital-gains                1054 non-null   int64
 11  capital-loss                 1054 non-null   int64
 12  incident_type                1054 non-null   object
 13  collision_type               1054 non-null   object
 14  incident_severity            1054 non-null   object
 15  authorities_contacted        1054 non-null   object
 16  incident_state               1054 non-null   object
 17  incident_city                1054 non-null   object
 18  incident_hour_of_the_day     1054 non-null   int64
 19  number_of_vehicles_involved  1054 non-null   int64
 20  property_damage              1054 non-null   object
 21  bodily_injuries              1054 non-null   int64
 22  witnesses                    1054 non-null   int64
 23  police_report_available      1054 non-null   object
 24  total_claim_amount           1054 non-null   int64
 25  auto_make                    1054 non-null   object
 26  auto_model                   1054 non-null   object
 27  auto_year                    1054 non-null   int64
 28  fraud_reported               1054 non-null   int64
 29  policy_csl_value             1054 non-null   int64
 30  injury_to_property_ratio     1054 non-null   float64
```

```
 30  injury_to_property_ratio     1054 non-null   float64
 31  month_sin                    1054 non-null   float64
 32  incident_year                1054 non-null   int32
 33  incident_month               1054 non-null   int32
 34  incident_dayofweek           1054 non-null   int32
 35  is_weekend                   1054 non-null   int32
dtypes: float64(2), int32(4), int64(13), object(17)
memory usage: 280.1+ KB
```

```python
# Check the data
# Create correlation matrix for numerical columns
# Select numerical columns
num_cols = X_train_resampled.select_dtypes(include=['int64', 'float64']).columns
num_cols
corr_matrix = X_train_resampled[num_cols].corr()
corr_matrix
# Plot Heatmap of the correlation matrix
plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Matrix for Numerical Features')
plt.show()

```

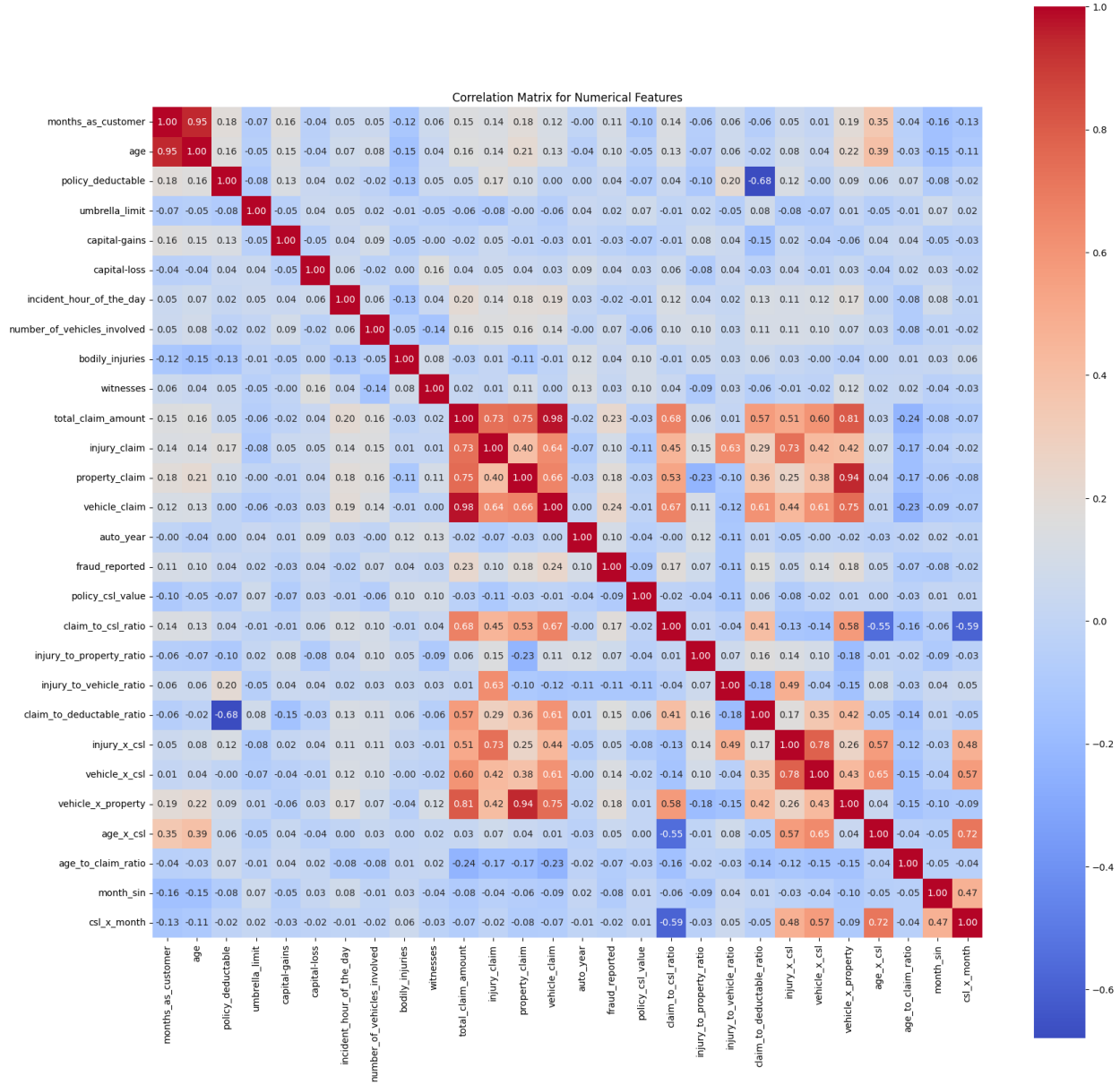
✓ 0.4s



Do correlaton alaysis for TEST dataset

```python
# Create correlation matrix for numerical columns
# Select numerical columns
num_cols = X_test_resampled.select_dtypes(include=['int64', 'float64']).columns
num_cols
corr_matrix = X_test_resampled[num_cols].corr()
corr_matrix
# Plot Heatmap of the correlation matrix
plt.figure(figsize=(20,20))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Matrix for Numerical Features')
plt.show()

```

✓ 1.3s



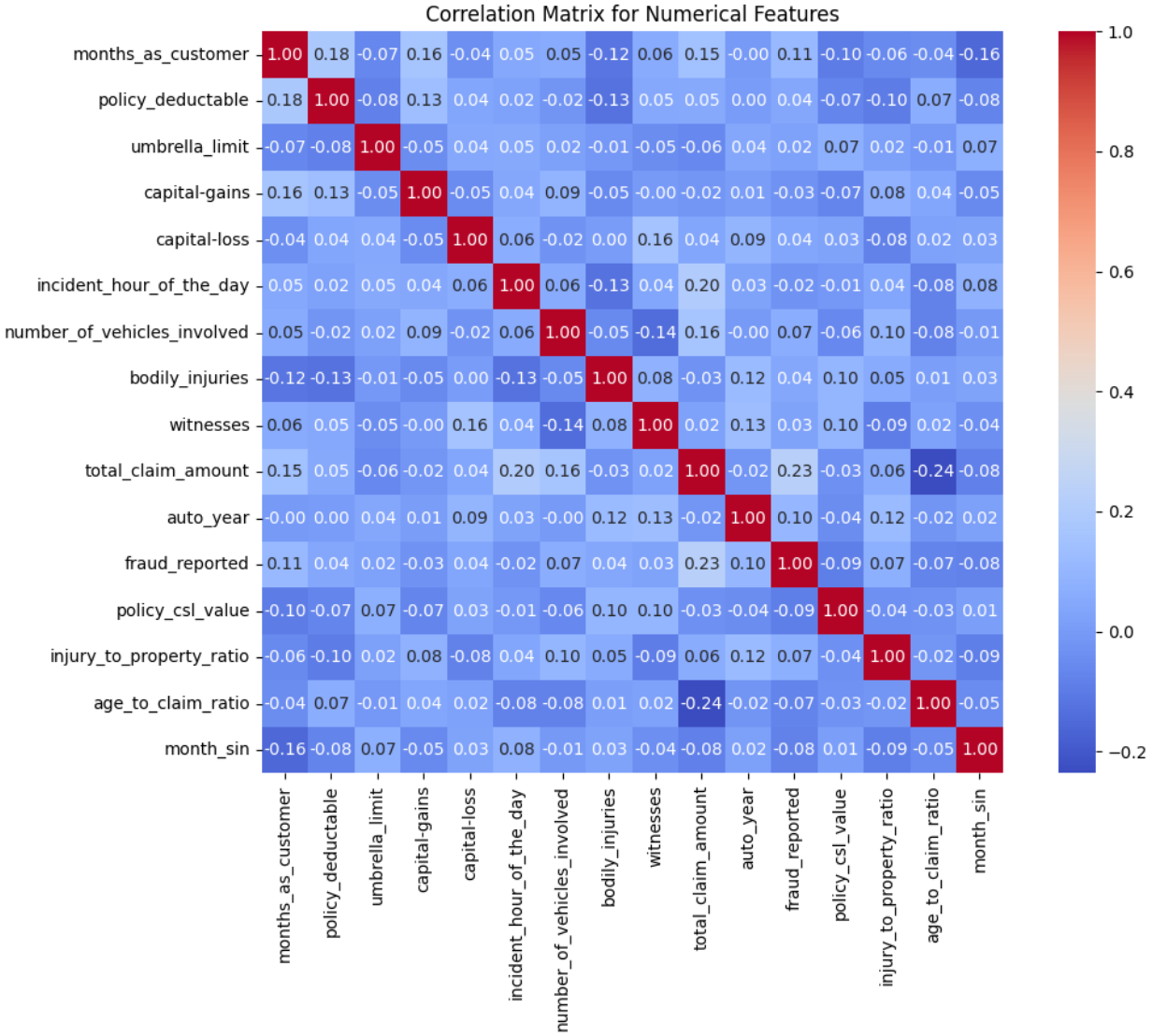

```python
# Compute correlation matrix for numerical columns
#corr_matrix = X_train_resampled.corr().abs()
corr_matrix = X_test_resampled[num_cols].corr().abs()

# Select upper triangle of correlation matrix
upper = corr_matrix.where(np.triu(np.ones(corr_matrix.shape), k=1).astype(bool))

# Find columns with correlation greater than 0.9
to_drop = [column for column in upper.columns if any(upper[column] > 0.5)]

# Drop highly correlated columns
X_test_resampled.drop(columns=to_drop, inplace=True)

print("Dropped columns due to high correlation:", to_drop)

#visualize correlation matrix usig heat map


```

✓ 0.0s Python

```
Dropped columns due to high correlation: ['age', 'injury_claim', 'property_claim', 'vehicle_claim', 'claim_to_csl_ratio', 'injury_to_vehicle_ratio', 'claim_to_deductable_ratio', 'injury
```

Dropped columns due to high correlation: ['age', 'injury_claim', 'property_claim', 'vehicle_claim', 'claim_to_csl_ratio', 'injury_to_vehicle_ratio', 'claim_to_deductable_ratio', 'injury_x_csl', 'vehicle_x_csl', 'vehicle_x_property', 'age_x_csl', 'csl_x_month']

```
# Check the data
# Create correlation matrix for numerical columns
# Select numerical columns
num_cols = X_test_resampled.select_dtypes(include=['int64', 'float64']).columns
num_cols
corr_matrix = X_test_resampled[num_cols].corr()
corr_matrix
# Plot Heatmap of the correlation matrix
plt.figure(figsize=(12,8))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True)
plt.title('Correlation Matrix for Numerical Features')
plt.show()

✓ 0.5s
```



```python
extra_in_test = set(X_test_resampled.columns) - set(X_train_resampled.columns)
extra_in_train = set(X_train_resampled.columns) - set(X_test_resampled.columns)

print("Extra columns in test:", extra_in_test)
print("Extra columns in train:", extra_in_train)

#drop column "age_to_claim_ratio" from Test dataset. This column not exist in training dataset
X_test_resampled.drop("age_to_claim_ratio", axis=1, inplace=True,errors="ignore")

```

[75] ✓ 0.0s

```
Extra columns in test: {'age_to_claim_ratio'}
Extra columns in train: set()
```

Consolidate low frequency Categorical features

```python
# Combine categories that have low frequency or provide limited predictive information
#X_train_resampled.info()
categorical_cols
```

[76] ✓ 0.0s

```
['policy_state',
 'policy_csl',
 'insured_sex',
 'insured_education_level',
 'insured_occupation',
 'insured_hobbies',
 'insured_relationship',
 'incident_type',
 'collision_type',
 'incident_severity',
 'authorities_contacted',
 'incident_state',
 'incident_city',
 'property_damage',
 'police_report_available',
 'auto_make',
 'auto_model']
```

```python
def combine_rare_categories(df, col, threshold=0.05):
    freq = df[col].value_counts(normalize=True)
    rare = freq[freq < threshold].index

    print(f"\nColumn: {col}")
    print(pd.DataFrame({
        'Category': freq.index,
        'Frequency': freq.values,
        'Is_Rare': freq < threshold
    }))

    df[col] = df[col].replace(rare, 'XX_Other')
    return df
X_train_resampled_combined_category = X_train_resampled.copy()
for col in categorical_cols:
    if col != 'auto_model':
        X_train_resampled_combined_category = combine_rare_categories(X_train_resampled_combined_category, col, threshold=0.05)

X_test_resampled_combined_category = X_test_resampled.copy()
for col in categorical_cols:
    if col != 'auto_model':
        X_test_resampled_combined_category = combine_rare_categories(X_test_resampled_combined_category, col, threshold=0.05)


#df=df_x1
```

78] ✓ 0.0s

Column: policy_state

Category Frequency Is_Rare

policy_state

IL IL 0.348197 False

OH OH 0.334915 False

IN IN 0.316888 False

Column: policy_csl

Category Frequency Is_Rare

policy_csl

250/500 250/500 0.365275 False

100/300 100/300 0.349146 False

500/1000 500/1000 0.285579 False

Column: insured_sex

Category Frequency Is_Rare

insured_sex

FEMALE FEMALE 0.544592 False

MALE MALE 0.455408 False

Column: insured_education_level

Category Frequency Is_Rare

insured_education_level

JD JD 0.178368 False

High School High School 0.156546 False

Masters Masters 0.153700 False

Associate Associate 0.147059 False

MD MD 0.128083 False

PhD PhD 0.124288 False

College College 0.111954 False

Column: insured_occupation

	Category	Frequency	Is_Rare
insured_occupation			
transport-moving	transport-moving	0.097723	False
machine-op-inspct	machine-op-inspct	0.088235	False
exec-managerial	exec-managerial	0.086338	False
armed-forces	armed-forces	0.085389	False
prof-specialty	prof-specialty	0.083491	False
tech-support	tech-support	0.078748	False
craft-repair	craft-repair	0.075901	False
sales	sales	0.074953	False
other-service	other-service	0.062619	False
adm-clerical	adm-clerical	0.062619	False
protective-serv	protective-serv	0.059772	False
priv-house-serv	priv-house-serv	0.055977	False
farming-fishing	farming-fishing	0.047438	True
handlers-cleaners	handlers-cleaners	0.040797	True

Column: insured_hobbies

	Category	Frequency	Is_Rare
insured_hobbies			
chess	chess	0.097723	False
paintball	paintball	0.076850	False
reading	reading	0.068311	False
yachting	yachting	0.051233	False
base-jumping	base-jumping	0.051233	False
cross-fit	cross-fit	0.050285	False
video-games	video-games	0.049336	True

exercise exercise 0.049336 True

board-games board-games 0.048387 True

hiking hiking 0.048387 True

skydiving skydiving 0.048387 True

bungie-jumping bungie-jumping 0.048387 True

polo polo 0.044592 True

movies movies 0.043643 True

golf golf 0.042694 True

kayaking kayaking 0.042694 True

dancing dancing 0.041746 True

sleeping sleeping 0.038899 True

camping camping 0.037951 True

basketball basketball 0.019924 True

Column: insured_relationship

Category Frequency Is_Rare

insured_relationship

other-relative other-relative 0.188805 False

wife wife 0.169829 False

not-in-family not-in-family 0.167932 False

unmarried unmarried 0.166983 False

own-child own-child 0.156546 False

husband husband 0.149905 False

Column: incident_type

Category Frequency Is_Rare

incident_type

Multi-vehicle Collision Multi-vehicle Collision 0.431689 False

Single Vehicle Collision Single Vehicle Collision 0.425996 False

Parked Car Parked Car 0.080645 False

Vehicle Theft Vehicle Theft 0.061670 False

Column: collision_type

Category Frequency Is_Rare

collision_type

Rear Collision Rear Collision 0.317837 False

Side Collision Side Collision 0.277040 False

Front Collision Front Collision 0.262808 False

? ? 0.142315 False

Column: incident_severity

Category Frequency Is_Rare

incident_severity

Major Damage Major Damage 0.422201 False

Minor Damage Minor Damage 0.283681 False

Total Loss Total Loss 0.230550 False

Trivial Damage Trivial Damage 0.063567 False

Column: authorities_contacted

Category Frequency Is_Rare

authorities_contacted

Police Police 0.264706 False

Fire Fire 0.227704 False

Ambulance Ambulance 0.227704 False

Other Other 0.196395 False

Unknown Unknown 0.083491 False

Column: incident_state

Category Frequency Is_Rare

incident_state

SC SC 0.261860 False

NY NY 0.251423 False

WV WV 0.176471 False

NC NC 0.134725 False

VA VA 0.103416 False

OH OH 0.038899 True

PA PA 0.033207 True

Column: incident_city

Category Frequency Is_Rare

incident_city

Columbus Columbus 0.163188 False

Arlington Arlington 0.163188 False

Springfield Springfield 0.161290 False

Hillsdale Hillsdale 0.138520 False

Northbend Northbend 0.136622 False

Riverwood Riverwood 0.122391 False

Northbrook Northbrook 0.114801 False

Column: property_damage

Category Frequency Is_Rare

property_damage

? ? 0.369070 False

YES YES 0.322581 False

NO NO 0.308349 False

Column: police_report_available

	Category	Frequency	Is_Rare
police_report_available			
NO	NO	0.358634	False
?	?	0.345351	False
YES	YES	0.296015	False

Column: auto_make

	Category	Frequency	Is_Rare
auto_make			
Ford	Ford	0.092979	False
Suburu	Suburu	0.092030	False
Chevrolet	Chevrolet	0.087287	False
Nissan	Nissan	0.081594	False
BMW	BMW	0.080645	False
Saab	Saab	0.077799	False
Dodge	Dodge	0.074953	False
Volkswagen	Volkswagen	0.065465	False
Audi	Audi	0.064516	False
Jeep	Jeep	0.063567	False
Mercedes	Mercedes	0.060721	False
Accura	Accura	0.056926	False
Toyota	Toyota	0.054080	False
Honda	Honda	0.047438	True

Column: policy_state

	Category	Frequency	Is_Rare
policy_state			
OH	OH	0.371681	False
IN	IN	0.345133	False

IL IL 0.283186 False

Column: policy_csl

Category Frequency Is_Rare

policy_csl

250/500 250/500 0.369469 False

100/300 100/300 0.351770 False

500/1000 500/1000 0.278761 False

Column: insured_sex

Category Frequency Is_Rare

insured_sex

MALE MALE 0.5 False

FEMALE FEMALE 0.5 False

Column: insured_education_level

Category Frequency Is_Rare

insured_education_level

MD MD 0.192478 False

High School High School 0.148230 False

College College 0.143805 False

PhD PhD 0.141593 False

JD JD 0.139381 False

Associate Associate 0.123894 False

Masters Masters 0.110619 False

Column: insured_occupation

Category Frequency Is_Rare

insured_occupation

farming-fishing farming-fishing 0.092920 False

tech-support tech-support 0.086283 False

machine-op-inspct machine-op-inspct 0.084071 False

prof-specialty prof-specialty 0.077434 False

exec-managerial exec-managerial 0.075221 False

armed-forces armed-forces 0.075221 False

transport-moving transport-moving 0.073009 False

other-service other-service 0.070796 False

priv-house-serv priv-house-serv 0.070796 False

craft-repair craft-repair 0.066372 False

protective-serv protective-serv 0.064159 False

sales sales 0.057522 False

handlers-cleaners handlers-cleaners 0.055310 False

adm-clerical adm-clerical 0.050885 False

Column: insured_hobbies

Category Frequency Is_Rare

insured_hobbies

board-games board-games 0.090708 False

movies movies 0.070796 False

hiking hiking 0.068584 False

exercise exercise 0.064159 False

chess chess 0.059735 False

yachting yachting 0.055310 False

reading reading 0.055310 False

basketball basketball 0.055310 False

cross-fit cross-fit 0.053097 False

video-games video-games 0.050885 False

sleeping sleeping 0.048673 True

camping camping 0.048673 True

polo polo 0.046460 True

bungie-jumping bungie-jumping 0.046460 True

golf golf 0.044248 True

base-jumping base-jumping 0.042035 True

skydiving skydiving 0.037611 True

kayaking kayaking 0.028761 True

paintball paintball 0.019912 True

dancing dancing 0.013274 True

Column: insured_relationship

Category Frequency Is_Rare

insured_relationship

own-child own-child 0.205752 False

not-in-family not-in-family 0.205752 False

other-relative other-relative 0.179204 False

husband husband 0.170354 False

wife wife 0.146018 False

unmarried unmarried 0.092920 False

Column: incident_type

Category Frequency Is_Rare

incident_type

Single Vehicle Collision Single Vehicle Collision 0.460177 False

Multi-vehicle Collision Multi-vehicle Collision 0.407080 False

Vehicle Theft Vehicle Theft 0.088496 False

Parked Car Parked Car 0.044248 True

Column: collision_type

	Category	Frequency	Is_Rare
collision_type			
Rear Collision	Rear Collision	0.303097	False
Side Collision	Side Collision	0.289823	False
Front Collision	Front Collision	0.274336	False
?	?	0.132743	False

Column: incident_severity

	Category	Frequency	Is_Rare
incident_severity			
Major Damage	Major Damage	0.431416	False
Minor Damage	Minor Damage	0.272124	False
Total Loss	Total Loss	0.223451	False
Trivial Damage	Trivial Damage	0.073009	False

Column: authorities_contacted

	Category	Frequency	Is_Rare
authorities_contacted			
Police	Police	0.289823	False
Fire	Fire	0.245575	False
Other	Other	0.236726	False
Ambulance	Ambulance	0.185841	False
Unknown	Unknown	0.042035	True

Column: incident_state

	Category	Frequency	Is_Rare
incident_state			
NY	NY	0.265487	False
WV	WV	0.258850	False

SC SC 0.238938 False

VA VA 0.101770 False

NC NC 0.088496 False

OH OH 0.026549 True

PA PA 0.019912 True

Column: incident_city

Category Frequency Is_Rare

incident_city

Northbend Northbend 0.172566 False

Arlington Arlington 0.152655 False

Springfield Springfield 0.150442 False

Hillsdale Hillsdale 0.137168 False

Riverwood Riverwood 0.137168 False

Northbrook Northbrook 0.134956 False

Columbus Columbus 0.115044 False

Column: property_damage

Category Frequency Is_Rare

property_damage

? ? 0.358407 False

NO NO 0.327434 False

YES YES 0.314159 False

Column: police_report_available

Category Frequency Is_Rare

police_report_available

NO NO 0.347345 False

YES YES 0.338496 False

? ? 0.314159 False

Column: auto_make

Category Frequency Is_Rare

auto_make

Dodge Dodge 0.119469 False

Mercedes Mercedes 0.088496 False

BMW BMW 0.084071 False

Nissan Nissan 0.075221 False

Volkswagen Volkswagen 0.073009 False

Accura Accura 0.073009 False

Saab Saab 0.068584 False

Audi Audi 0.066372 False

Toyota Toyota 0.064159 False

Chevrolet Chevrolet 0.064159 False

Ford Ford 0.064159 False

Suburu Suburu 0.061947 False

Honda Honda 0.057522 False

Jeep Jeep 0.039823 True

Get categorical features

```
categorical_cols = X_train_resampled_combined_category.select_dtypes(include=['object']).columns.tolist()
categorical_cols
```

[81] ✓ 0.0s

```
['policy_state',
 'policy_csl',
 'insured_sex',
 'insured_education_level',
 'insured_occupation',
 'insured_hobbies',
 'insured_relationship',
 'incident_type',
 'collision_type',
 'incident_severity',
 'authorities_contacted',
 'incident_state',
 'incident_city',
 'property_damage',
 'police_report_available',
 'auto_make',
 'auto_model']
```

Convert Categorical features into numerical features by creating Dummy variables

```
# Create dummy variables using the 'get_dummies' for categorical columns in training data
# List of nominal categorical variables
nominal_cat_variables = [
    'policy_state', 'insured_sex', 'insured_occupation', 'insured_hobbies',
    'insured_relationship', 'incident_type', 'collision_type',
    'authorities_contacted', 'incident_state', 'incident_city',
    'property_damage', 'police_report_available', 'auto_make', 'auto_model'
]
X_train = pd.get_dummies(X_train_resampled_combined_category, columns=nominal_cat_variables, drop_first=True)
```

[84] ✓ 0.0s

Convert ordinal categorical features into numerical

```
# Define your custom order (from lowest to highest education level)
education_order = {
    'High School': 1,
    'College': 2,
    'Associate': 3,
    'Masters': 4,
    'JD': 5,
    'MD': 6,
    'PhD': 7
}

# Apply the mapping
X_train['insured_education_level_encoded'] = X_train['insured_education_level'].map(education_order)

# Check if any unmapped values remain
print(X_train['insured_education_level_encoded'].isnull().sum())

# Display the mapping result
print(X_train[['insured_education_level', 'insured_education_level_encoded']].drop_duplicates())
```

✓ 0.0s

```python
# Define ordinal mapping for incident severity
severity_order = {
    'Trivial Damage': 1,
    'Minor Damage': 2,
    'Major Damage': 3,
    'Total Loss': 4
}

# Apply mapping
X_train['incident_severity_encoded'] = X_train['incident_severity'].map(severity_order)

# Drop the original column
#X_train.drop('incident_severity', axis=1, inplace=True)

# Verify
X_train[['incident_severity_encoded']].head()

```

✓ 0.0s

	incident_severity_encoded
0	2
1	2
2	2
3	1
4	4

```python
# Define ordinal mapping for policy_csl
policy_csl_order = {
    '100/300': 1,
    '250/500': 2,
    '500/1000': 3
}

# Apply mapping
X_train['policy_csl_encoded'] = X_train['policy_csl'].map(policy_csl_order)

# Drop the original column
#X_train.drop('policy_csl', axis=1, inplace=True)

# Verify the result
X_train[['policy_csl_encoded']].head()

```

[92] ✓ 0.0s

	policy_csl_encoded
0	1
1	3
2	2
3	3
4	2

```python
# Create dummy variables using the 'get_dummies' for categorical columns in validation data
# List of nominal categorical variables
nominal_cat_variables = [
    'policy_state', 'insured_sex', 'insured_occupation', 'insured_hobbies',
    'insured_relationship', 'incident_type', 'collision_type',
    'authorities_contacted', 'incident_state', 'incident_city',
    'property_damage', 'police_report_available', 'auto_make', 'auto_model'
]

# Step 2: Create dummies for test
X_test = pd.get_dummies(X_test_resampled_combined_category, columns=nominal_cat_variables, drop_first=True)

# Step 3: Align test columns with train
X_test = X_test.reindex(columns=X_train.columns, fill_value=0)


```

[97] ✓ 0.0s

```python
# Define your custom order (from lowest to highest education level)
education_order = {
    'High School': 1,
    'College': 2,
    'Associate': 3,
    'Masters': 4,
    'JD': 5,
    'MD': 6,
    'PhD': 7
}

# Apply the mapping
X_test['insured_education_level_encoded'] = X_test['insured_education_level'].map(education_order)

# Check if any unmapped values remain
print(X_test['insured_education_level_encoded'].isnull().sum())

# Display the mapping result
print(X_test[['insured_education_level', 'insured_education_level_encoded']].drop_duplicates())

#X_test.drop('insured_education_level', axis=1, inplace=True)

X_test[['insured_education_level_encoded']].head()
```

✓ 0.0s

```
0
   insured_education_level  insured_education_level_encoded
0                  Masters                                4
1                      PhD                                7
4                       JD                                5
10                 College                                2
11                      MD                                6
19               Associate                                3
26            High School                                1
```

```
# Define ordinal mapping for incident severity
severity_order = {
    'Trivial Damage': 1,
    'Minor Damage': 2,
    'Major Damage': 3,
    'Total Loss': 4
}

# Apply mapping
X_test['incident_severity_encoded'] = X_test['incident_severity'].map(severity_order)

# Drop the original column
#X_test.drop('incident_severity', axis=1, inplace=True)

# Verify
X_test[['incident_severity_encoded']].head()
```

[102] ✓ 0.0s

	incident_severity_encoded
0	4
1	3
2	2
3	2
4	4

```
# Define ordinal mapping for policy_csl for test data
policy_csl_order = {
    '100/300': 1,
    '250/500': 2,
    '500/1000': 3
}

# Apply mapping
X_test['policy_csl_encoded'] = X_test['policy_csl'].map(policy_csl_order)

# Drop the original column
#X_test.drop('policy_csl', axis=1, inplace=True)

# Verify the result
X_test[['policy_csl_encoded']].head()
```

[103] ✓ 0.0s

	policy_csl_encoded
0	3
1	3
2	2
3	3
4	1

Drop ordinal categorical features

```python
X_train.drop('incident_severity', axis=1, inplace=True,errors='ignore')
X_test.drop('incident_severity', axis=1, inplace=True,errors='ignore')

X_train.drop('policy_csl', axis=1, inplace=True,errors='ignore')
X_test.drop('policy_csl', axis=1, inplace=True,errors='ignore')

X_train.drop('insured_education_level', axis=1, inplace=True,errors='ignore')
X_test.drop('insured_education_level', axis=1, inplace=True,errors='ignore')
```

[103] ✓ 0.0s

Dependent feature is transformed from Y/N to 0/1 at the beginning at the time of train-test split

We can use code commented below to translate Target variable into numerical value

```python
X_test["fraud_reported"].value_counts(), X_train["fraud_reported"].value_counts()
```

[108] ✓ 0.0s

```
(fraud_reported
 0    226
 1    226
 Name: count, dtype: int64,
 fraud_reported
 0    527
 1    527
 Name: count, dtype: int64)
```

```python
# Create dummy variable for dependent feature in training data
# Put the target variable in y

# Target variable is already encoded into 0/1 at the begining , before splitting original dataset into training and test
# y_train['fraud_reported']=y_test['fraud_reported'].map({'Y':1, 'N':0})

# Create dummy variable for dependent feature in validation data
 #y_test['fraud_reported']=y_test['fraud_reported'].map({'Y':1, 'N':0})
```

[109] ✓ 0.0s

Feature scaling

Before Feature scaling

```python
X_train.head(5)
```

✓ 0.0s Python

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved	bodily_injuries	witnesses	total_claim_amount	auto_year	policy_c
0	428	2000	0	0	24400	16	3	1	0	76560	2009	
1	0	1000	0	61400	41100	6	4	1	1	39720	2002	
2	47	2000	0	75400	0	18	1	1	0	6700	2011	
3	283	1000	0	0	46200	9	1	2	3	6560	2003	
4	343	500	3000000	0	0	13	4	2	3	65070	2003	

```python
X_test.head(5)
```

✓ 0.0s Python

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved	bodily_injuries	witnesses	total_claim_amount	auto_year	policy_c
0	467	2000	0	82200	0	2	1	2	3	55700	2014	
1	197	1000	0	0	64500	6	1	2	2	76400	1997	
2	342	500	0	63100	13800	3	1	2	0	76700	2006	
3	245	1000	0	39300	0	12	1	0	2	82170	1999	
4	32	2000	6000000	0	79800	16	1	1	2	45500	2009	

After Feature scaling

```python
# Import the necessary scaling tool from scikit-learn
from sklearn.preprocessing import StandardScaler

# Scale the numeric features present in the training data
numeric_cols = X_train.select_dtypes(include=['int64', 'float64','int32']).columns

# Initialize the scaler
scaler = StandardScaler()

# Fit on training data only
X_train[numeric_cols] = scaler.fit_transform(X_train[numeric_cols])

# Scale the numeric features present in the validation data

X_test[numeric_cols] = scaler.transform(X_test[numeric_cols])


```

12] ✓ 0.0s

```python
X_train.head(5)
```

✓ 0.0s Python

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved	bodily_injuries	witnesses	total_claim_amount	auto_year	polic
0	1.915629	1.345182	-0.494534	-0.933658	-0.058181	0.626985	1.111877	0.016051	-1.351036	0.818421	0.703048	
1	-1.811300	-0.261415	-0.494534	1.296027	0.534189	-0.822867	2.090923	0.016051	-0.435286	-0.630992	-0.471830	
2	-1.402035	1.345182	-0.494534	1.804425	-0.923679	0.916956	-0.846215	0.016051	-1.351036	-1.930113	1.038728	
3	0.653001	-0.261415	-0.494534	-0.933658	0.715092	-0.387911	-0.846215	1.224442	1.396215	-1.935621	-0.303991	
4	1.175467	-1.064713	0.809651	-0.933658	-0.923679	0.192030	2.090923	1.224442	1.396215	0.366365	-0.303991	

```python
X_test.head(5)
```

✓ 0.0s Python

	months_as_customer	policy_deductable	umbrella_limit	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehicles_involved	bodily_injuries	witnesses	total_claim_amount	auto_year	polic
0	2.255232	1.345182	-0.494534	2.051361	-0.923679	-1.402808	-0.846215	1.224442	1.396215	-0.002283	1.542247	
1	-0.095868	-0.261415	-0.494534	-0.933658	1.364215	-0.822867	-0.846215	1.224442	0.480465	0.812126	-1.311029	
2	1.166760	-1.064713	-0.494534	1.357761	-0.434176	-1.257822	-0.846215	1.224442	-1.351036	0.823929	0.199529	
3	0.322105	-0.261415	-0.494534	0.493486	-0.923679	0.047045	-0.846215	-1.192341	0.480465	1.039138	-0.975350	
4	-1.532652	1.345182	2.113836	-0.933658	1.906925	0.626985	-0.846215	0.016051	0.480465	-0.403587	0.703048	

Model building

```python
# Import necessary libraries
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

[114] ✓ 0.0s

```python
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

[235] ✓ 0.0s

```
((1054, 123), (1054,), (452, 123), (452,))
```

Feature Selection

```python
# Apply RFECV to identify the most relevant features
from sklearn.feature_selection import RFECV
from sklearn.model_selection import KFold
from sklearn.linear_model import LogisticRegression

# Define estimator (Logistic Regression in this case)
log_reg = LogisticRegression(max_iter=1000, solver='liblinear')

# Define RFECV with standard KFold
rfecv = RFECV(
    estimator=log_reg,
    step=1,                # eliminate 1 feature per iteration
    cv=KFold(n_splits=5, shuffle=True, random_state=42),
    scoring='accuracy',    # or 'f1', 'roc_auc' etc.
    n_jobs=-1              # use all CPU cores
)

# Fit on training data
rfecv.fit(X_train, y_train)

# Print results
print("Optimal number of features:", rfecv.n_features_)
print("Selected features:", X_train.columns[rfecv.support_].tolist())

# Reduce dataset to selected features
X_train_selected = X_train.loc[:, rfecv.support_]
X_test_selected = X_test.loc[:, rfecv.support_]
```

[119] ✓ 4.3s Python

```
Optimal number of features: 46
Selected features: ['total_claim_amount', 'insured_occupation_armed-forces', 'insured_occupation_craft-repair', 'insured_occupation_exec-managerial', 'insured_occupation_other-service'
```

```python
# Display the features ranking by RFECV in a DataFrame
# Create a DataFrame showing feature ranking
feature_ranking = pd.DataFrame({
    'Feature Name': X_train.columns,
    'Rank': rfecv.ranking_,
    'Fected Selected': rfecv.support_
}).sort_values(by='Rank', ascending=True)

print(feature_ranking)

```

[120] ✓ 0.0s

```
...                                Feature Name  Rank  Fected Selected
40      insured_relationship_other-relative     1             True
102                     auto_model_Legacy     1             True
26         insured_occupation_other-service     1             True
27       insured_occupation_priv-house-serv     1             True
57                      incident_state_WV     1             True
94                      auto_model_Escape     1             True
54                      incident_state_NY     1             True
31          insured_occupation_tech-support     1             True
32      insured_occupation_transport-moving     1             True
33             insured_hobbies_base-jumping     1             True
24       insured_occupation_exec-managerial     1             True
34                   insured_hobbies_chess     1             True
36               insured_hobbies_paintball     1             True
37                 insured_hobbies_reading     1             True
38                insured_hobbies_yachting     1             True
53           authorities_contacted_Unknown     1             True
95                       auto_model_F150     1             True
100                    auto_model_Impreza     1             True
98               auto_model_Grand Cherokee     1             True
43               insured_relationship_wife     1             True
44                incident_type_Parked Car     1             True
35               insured_hobbies_cross-fit     1             True
96                    auto_model_Forrestor     1             True
23          insured_occupation_craft-repair     1             True
93                       auto_model_E400     1             True
78                      auto_make_Suburu     1             True
```

78	auto_make_Suburu	1	True
83	auto_model_93	1	True
119	auto_model_X6	1	True
86	auto_model_A5	1	True
117	auto_model_Wrangler	1	True
115	auto_model_Tahoe	1	True
114	auto_model_TL	1	True
89	auto_model_CRV	1	True
9	total_claim_amount	1	True
22	insured_occupation_armed-forces	1	True
113	auto_model_Silverado	1	True
67	police_report_available_NO	1	True
91	auto_model_Civic	1	True
65	property_damage_NO	1	True
110	auto_model_Pathfinder	1	True
108	auto_model_Neon	1	True
107	auto_model_Maxima	1	True
106	auto_model_Malibu	1	True
104	auto_model_MDX	1	True
112	auto_model_RSX	1	True
46	incident_type_Vehicle Theft	1	True
63	incident_city_Riverwood	2	False
61	incident_city_Northbend	3	False
60	incident_city_Hillsdale	4	False
59	incident_city_Columbus	5	False
62	incident_city_Northbrook	6	False
75	auto_make_Mercedes	7	False
69	auto_make_Audi	8	False
103	auto_model_M5	9	False
79	auto_make_Toyota	10	False
73	auto_make_Ford	11	False
49	collision_type_Side Collision	12	False
88	auto_model_C300	13	False
118	auto_model_X5	14	False
70	auto_make_BMW	15	False

```
# Put columns selected by RFECV into variable 'col'
col = X_train.columns[rfecv.support_].tolist()
col
```

[121] ✓ 0.0s

```
['total_claim_amount',
 'insured_occupation_armed-forces',
 'insured_occupation_craft-repair',
 'insured_occupation_exec-managerial',
 'insured_occupation_other-service',
 'insured_occupation_priv-house-serv',
 'insured_occupation_tech-support',
 'insured_occupation_transport-moving',
 'insured_hobbies_base-jumping',
 'insured_hobbies_chess',
 'insured_hobbies_cross-fit',
 'insured_hobbies_paintball',
 'insured_hobbies_reading',
 'insured_hobbies_yachting',
 'insured_relationship_other-relative',
 'insured_relationship_wife',
 'incident_type_Parked Car',
 'incident_type_Vehicle Theft',
 'authorities_contacted_Unknown',
 'incident_state_NY',
 'incident_state_WV',
 'property_damage_NO',
 'police_report_available_NO',
 'auto_make_Suburu',
 'auto_model_93',
 'auto_model_A5',
 'auto_model_CRV',
 'auto_model_Civic',
 'auto_model_E400',
 'auto_model_Escape',
 'auto_model_F150',
 'auto_model_Forrestor',
```

Build Logistic Regression Base Model

```python
# Select only the columns selected by RFECV
X_train_selected = X_train[col]
X_test_selected = X_test[col]

```

[122] ✓ 0.0s

```python
# Import statsmodels and add constant
import statsmodels.api as sm

# Add constant term to the datasets

# Ensure all predictors are numeric
X_train_sm = X_train_selected.apply(pd.to_numeric, errors='coerce')
X_train_sm = X_train_sm.astype({col: 'int' for col in X_train_sm.select_dtypes('bool').columns})
X_train_sm = sm.add_constant(X_train_sm)

# Also ensure y_train is numeric (e.g., 0/1)
y_train_numeric = y_train.astype(int)


X_test_sm = sm.add_constant(X_test_selected)

# Check the data
print("✅ Constant added to training and test sets.")
print("X_train_sm shape:", X_train_sm.shape)
print("X_test_sm shape:", X_test_sm.shape)
```

[123] ✓ 0.4s

```python
# Fit a logistic Regression model on X_train after adding a constant and output the summary
logit_model = sm.Logit(y_train_numeric, X_train_sm)
result = logit_model.fit()

# Display the summary
print(result.summary())
```

[126] ✓ 0.3s

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.451345
         Iterations: 35
                           Logit Regression Results
==============================================================================
Dep. Variable:         fraud_reported   No. Observations:                 1054
Model:                          Logit   Df Residuals:                     1008
Method:                           MLE   Df Model:                           45
Date:                Wed, 15 Oct 2025   Pseudo R-squ.:                 0.3488
Time:                        14:27:22   Log-Likelihood:                -475.72
converged:                      False   LL-Null:                       -730.58
Covariance Type:            nonrobust   LLR p-value:                5.155e-80
=========================================================================================================
                                            coef    std err          z      P>|z|      [0.025      0.975]
---------------------------------------------------------------------------------------------------------
const                                    -0.7219      0.209     -3.452      0.001      -1.132      -0.312
total_claim_amount                        0.3711      0.136      2.725      0.006       0.104       0.638
insured_occupation_armed-forces           0.8632      0.351      2.462      0.014       0.176       1.550
insured_occupation_craft-repair           0.5073      0.309      1.643      0.100      -0.098       1.112
insured_occupation_exec-managerial        1.2055      0.307      3.922      0.000       0.603       1.808
insured_occupation_other-service          0.8007      0.346      2.311      0.021       0.122       1.480
insured_occupation_priv-house-serv       -1.1356      0.433     -2.620      0.009      -1.985      -0.286
insured_occupation_tech-support           0.7477      0.329      2.269      0.023       0.102       1.394
insured_occupation_transport-moving       1.2913      0.305      4.240      0.000       0.694       1.888
insured_hobbies_base-jumping              0.9994      0.347      2.883      0.004       0.320       1.679
insured_hobbies_chess                     4.4341      0.566      7.841      0.000       3.326       5.543
insured_hobbies_cross-fit                 3.7030      0.602      6.147      0.000       2.522       4.884
insured_hobbies_paintball                 1.2372      0.301      4.104      0.000       0.646       1.828
insured_hobbies_reading                   0.6749      0.311      2.171      0.030       0.066       1.284
insured_hobbies_yachting                  0.9791      0.362      2.704      0.007       0.269       1.689
insured_relationship_other-relative       0.4063      0.222      1.831      0.067      -0.029       0.841
insured_relationship_wife                 0.4486      0.233      1.925      0.054      -0.008       0.905
incident_type_Parked_Car                 -0.8712      0.650     -1.340      0.180      -2.145       0.403
```

Check VIF score

```python
# Make a VIF DataFrame for all the variables present


# Ensure all boolean columns are converted to int (0/1)
X_train_vif = X_train_sm.astype({col: 'int' for col in X_train_sm.select_dtypes('bool').columns})

# Create VIF dataframe
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_train_vif.values, i)
                   for i in range(X_train_vif.shape[1])]

# Sort by highest VIF
vif_data = vif_data.sort_values(by="VIF", ascending=False).reset_index(drop=True)

print(vif_data)


```

[131] ✓ 0.7s

```
                            Feature       VIF
0                auto_model_Forrestor       inf
1                    auto_make_Suburu       inf
2                  auto_model_Impreza       inf
3                   auto_model_Legacy       inf
4                               const  7.277640
5          incident_type_Parked Car  3.234796
6                total_claim_amount  3.146982
7        incident_type_Vehicle Theft  2.469809
```

Drop variables with high VIF score and rebuild model

```python
#drop auto_model_Forrestor from training data set and retrain states model
X_train_sm = X_train_sm.drop(columns=['auto_model_Forrestor'])

# Fit logistic regression with statsmodels
logit_model = sm.Logit(y_train, X_train_sm)
result = logit_model.fit()

# Display the summary
print(result.summary())
```

[132] ✓ 0.4s

```
Warning: Maximum number of iterations has been exceeded.
         Current function value: 0.451345
         Iterations: 35
                           Logit Regression Results
==============================================================================
Dep. Variable:         fraud_reported   No. Observations:                 1054
Model:                          Logit   Df Residuals:                     1008
Method:                           MLE   Df Model:                           45
Date:                Wed, 15 Oct 2025   Pseudo R-squ.:                 0.3488
Time:                        14:27:24   Log-Likelihood:                -475.72
converged:                      False   LL-Null:                       -730.58
Covariance Type:            nonrobust   LLR p-value:                5.155e-80
=======================================================================================================
                                          coef    std err          z      P>|z|      [0.025      0.975]
-------------------------------------------------------------------------------------------------------
const                                  -0.7219      0.209     -3.452      0.001      -1.132      -0.312
total_claim_amount                      0.3711      0.136      2.725      0.006       0.104       0.638
insured_occupation_armed-forces         0.8632      0.351      2.462      0.014       0.176       1.550
insured_occupation_craft-repair         0.5073      0.309      1.643      0.100      -0.098       1.112
insured_occupation_exec-managerial      1.2055      0.307      3.922      0.000       0.603       1.808
insured_occupation_other-service        0.8007      0.346      2.311      0.021       0.122       1.480
insured_occupation_priv-house-serv     -1.1356      0.433     -2.620      0.009      -1.985      -0.286
insured_occupation_tech-support         0.7477      0.329      2.269      0.023       0.102       1.394
insured_occupation_transport-moving     1.2913      0.305      4.240      0.000       0.694       1.888
insured_hobbies_base-jumping            0.9994      0.347      2.883      0.004       0.320       1.679
insured_hobbies_chess                   4.4341      0.566      7.841      0.000       3.326       5.543
insured_hobbies_cross-fit               3.7030      0.602      6.147      0.000       2.522       4.884
insured_hobbies_paintball               1.2372      0.301      4.104      0.000       0.646       1.828
insured_hobbies_reading                 0.6749      0.311      2.171      0.030       0.066       1.284
```

```python
# Make a VIF DataFrame for all the variables present


# Ensure all boolean columns are converted to int (0/1)
X_train_vif = X_train_sm.astype({col: 'int' for col in X_train_sm.select_dtypes('bool').columns})

# Create VIF dataframe
vif_data = pd.DataFrame()
vif_data["Feature"] = X_train_vif.columns
vif_data["VIF"] = [variance_inflation_factor(X_train_vif.values, i)
                   for i in range(X_train_vif.shape[1])]

# Sort by highest VIF
vif_data = vif_data.sort_values(by="VIF", ascending=False).reset_index(drop=True)

print(vif_data)


```

[133] ✓ 0.8s

```
                              Feature       VIF
0                               const  7.277640
1             incident_type_Parked Car  3.234796
2                   total_claim_amount  3.146982
3           incident_type_Vehicle Theft  2.469809
4                    auto_make_Suburu  2.391470
5         authorities_contacted_Unknown  2.391093
6                    auto_model_Legacy  1.913530
7                   auto_model_Impreza  1.460137
8   insured_occupation_transport-moving  1.240720
9       insured_occupation_armed-forces  1.240206
10                   incident_state_WV  1.201993
11                   incident_state_NY  1.193631
```

Predictions on Training Data

```python
# Predict the probabilities on the training data
y_train_pred_proba = result.predict(X_train_sm)

print(type(y_train_pred_proba))

# Reshape it into an array
y_train_pred_proba = y_train_pred_proba.to_numpy().reshape(-1)
print(type(y_train_pred_proba))
y_train_pred_proba
```

[265] ✓ 0.0s

```
<class 'pandas.core.series.Series'>
<class 'numpy.ndarray'>

array([0.2853864 , 0.44252607, 0.01408681, ..., 0.78184807, 0.97184916,
       0.99521495])
```

```python
# Create a new DataFrame containing the actual fraud reported flag and the probabilities predicted by the model

# Create new column indicating predicted classifications based on a cutoff value of 0.5

import pandas as pd
pd.set_option('display.float_format', '{:.6f}'.format)
# Predicted probabilities for the training set
y_pred_prob = result.predict(X_train_sm)  # X_train_sm is your training features with constant

# Predicted classifications based on cutoff 0.5
y_pred_class = (y_pred_prob >= 0.5).astype(int)

# Create a DataFrame
pred_df = pd.DataFrame({
    'actual_fraud': y_train_numeric,
    'predicted_prob': y_pred_prob,
    'predicted_class': y_pred_class
})

# Display the first few rows
print(pred_df.head(5))


```

[136] ✓ 0.0s

```
   actual_fraud  predicted_prob  predicted_class
0             0        0.285386                0
1             0        0.442526                0
2             0        0.014087                0
3             0        0.173035                0
4             0        0.238743                0
```

Performance Metrics for Training Data

```python
# Import metrics from sklearn for evaluation
from sklearn import metrics

# Check the accuracy of the model
metrics_train = evaluate_logit_model(result, X_train_sm, y_train_numeric,"accuracy")

```

[266] ✓ 0.0s

```
Accuracy: 0.7818
```

```python
# Create confusion matrix
metrics_train = evaluate_logit_model(result, X_train_sm, y_train_numeric,"confusion_matrix")
```

[267] ✓ 0.0s

```
Confusion Matrix:
 [[407 120]
 [110 417]]
```

```python
# Create variables for true positive, true negative, false positive and false negative
metrics_train = evaluate_logit_model(result, X_train_sm, y_train_numeric,"TP_TN")
```

[139] ✓ 0.0s

```
TP: 417, TN: 407, FP: 120, FN: 110
```

```python
#type(metrics_train)
# Extract confusion matrix from dictionary
cm = metrics_train["confusion_matrix"]

# Unpack values
TN, FP, FN, TP = cm.ravel()

# Store in variables
true_positive = TP
true_negative = TN
false_positive = FP
false_negative = FN

# Check
print("TP:", true_positive)   # 417
print("TN:", true_negative)   # 407
print("FP:", false_positive)  # 120
print("FN:", false_negative)  # 110

```

[140] ✓ 0.0s

```
TP: 417
TN: 407
FP: 120
FN: 110
```

```python
# Calculate the sensitivity
sensitivity = TP / (TP + FN)
print(f"Sensitivity (Recall): {sensitivity:.4f}")

# Calculate the specificity
specificity = TN / (TN + FP)
print(f"Specificity: {specificity:.4f}")

# Calculate Precision
precision = TP / (TP + FP)
print(f"Precision: {precision:.4f}")

# Calculate Recall
recall = sensitivity
print(f"Recall: {recall:.4f}")

# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score: {f1_score:.4f}")


```

[141] ✓ 0.0s

```
Sensitivity (Recall): 0.7913
Specificity: 0.7723
Precision: 0.7765
Recall: 0.7913
F1 Score: 0.7838
```

Created new python Function to get Performance Metrics for Logistic Regression

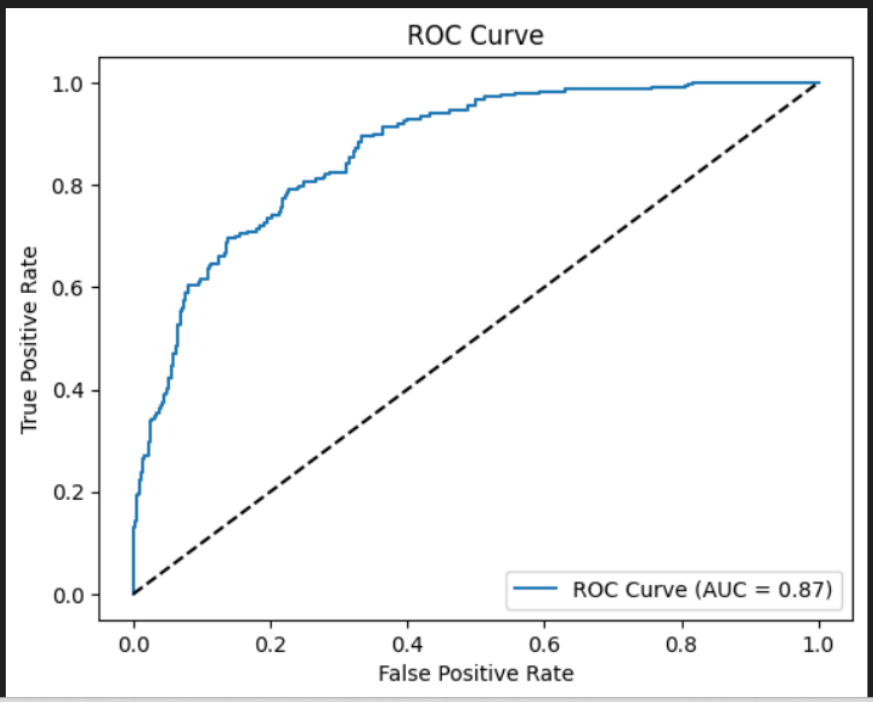
```python
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_curve, accuracy_score


def evaluate_logit_model(result, X, y, metrix_type, threshold=0.5, plot_roc=True):
    """
    Evaluate a statsmodels logistic regression model.
    """
    # Predicted probabilities
    y_pred_proba = result.predict(X)

    # Predicted classes
    y_pred = (y_pred_proba >= threshold).astype(int)

    # Initialize
    cm = None
    roc_auc = None

    cm = confusion_matrix(y, y_pred)
    if metrix_type == "confusion_matrix":
        print("Confusion Matrix:\n", cm)
    if metrix_type == "TP_TN":
        tn, fp, fn, tp = cm.ravel()
        print(f"TP: {tp}, TN: {tn}, FP: {fp}, FN: {fn}")
    if metrix_type == "classification_report":
        print("\nClassification Report:\n", classification_report(y, y_pred))

    if metrix_type == "accuracy":
         acc = accuracy_score(y, y_pred)
         print(f"Accuracy: {acc:.4f}")
    if metrix_type == "roc_auc_score":
        roc_auc = roc_auc_score(y, y_pred_proba)
        print("ROC-AUC:", roc_auc)
        if plot_roc:
            fpr, tpr, thresholds = roc_curve(y, y_pred_proba)
            plt.plot(fpr, tpr, label=f'ROC Curve (AUC = {roc_auc:.2f})')
            plt.plot([0,1], [0,1], 'k--')
            plt.xlabel('False Positive Rate')
            plt.ylabel('True Positive Rate')
            plt.title('ROC Curve')
```

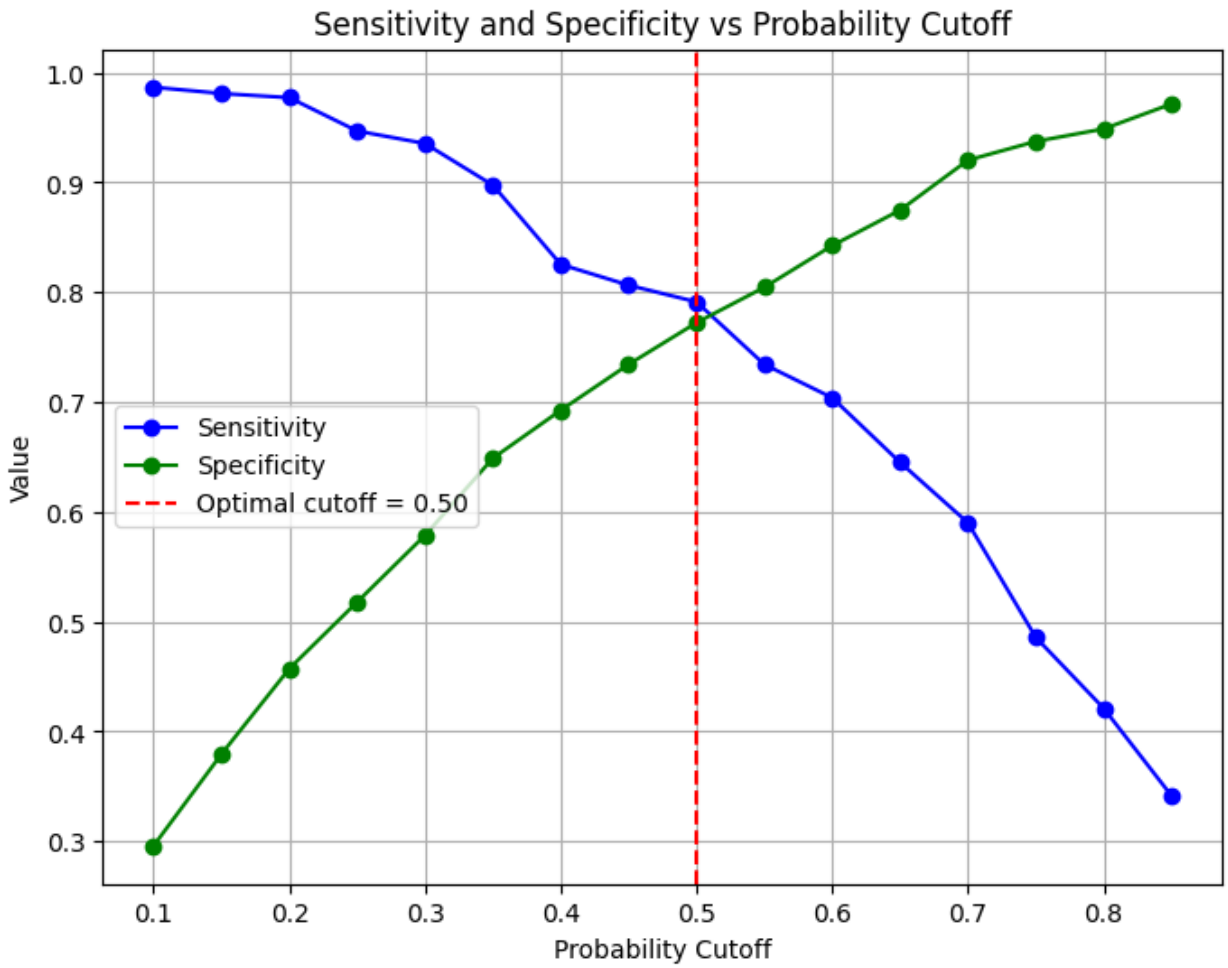
[127]

```
# Call the ROC function
metrics_train = evaluate_logit_model(result, X_train_sm, y_train_numeric,"roc_auc_score")
```

[143] ✓ 0.1s

```
ROC-AUC: 0.8675939494975318
```




Sensitivity and Specificity vs Probability Cutoff
Sensitivity
Specificity
Optimal cutoff = 0.50
Value
Probability Cutoff
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8

Optimal cut-off identificaion

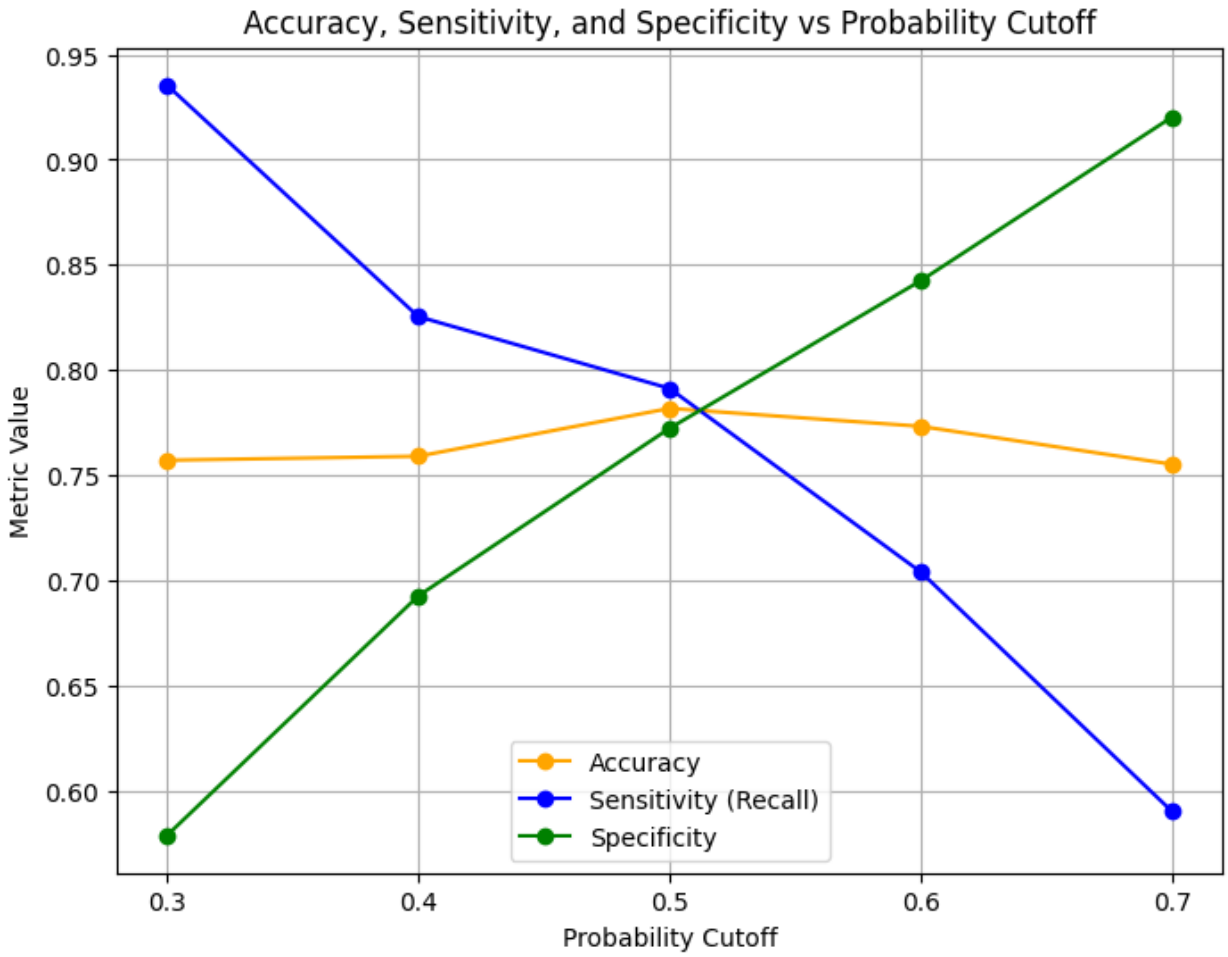
```python
# Create columns with different probability cutoffs to explore the impact of cutoff on model performance
# List of probability cutoffs you want to try
cutoffs = [0.3, 0.4, 0.5, 0.6, 0.7]

# Predicted probabilities on training set
y_pred_proba_train = result.predict(X_train_sm)  # X_train_sm should have the constant added

# Create a DataFrame to store predictions for each cutoff
predictions_df = pd.DataFrame({'actual': y_train})

for cutoff in cutoffs:
    predictions_df[f'pred_{cutoff}'] = (y_pred_proba_train >= cutoff).astype(int)

# Display the first few rows
predictions_df.head()
```

[145] ✓ 0.0s

	actual	pred_0.3	pred_0.4	pred_0.5	pred_0.6	pred_0.7
0	0	0	0	0	0	0
1	0	1	1	0	0	0
2	0	0	0	0	0	0
3	0	0	0	0	0	0
4	0	0	0	0	0	0

```python
# Plot accuracy, sensitivity, and specificity at different values of probability cutoffs

# ---- Plot metrics vs cutoff ----
plt.figure(figsize=(8,6))
plt.plot(metrics_df['cutoff'], metrics_df['accuracy'], marker='o', label='Accuracy', color='orange')
plt.plot(metrics_df['cutoff'], metrics_df['sensitivity'], marker='o', label='Sensitivity (Recall)', color='blue')
plt.plot(metrics_df['cutoff'], metrics_df['specificity'], marker='o', label='Specificity', color='green')

plt.title('Accuracy, Sensitivity, and Specificity vs Probability Cutoff')
plt.xlabel('Probability Cutoff')
plt.ylabel('Metric Value')
plt.xticks(cutoffs)
plt.grid(True)
plt.legend()
plt.show()

# ---- Display metrics table ----
print(metrics_df)
```

[147] ✓ 0.1s




```python
# Create a column for final prediction based on the optimal cutoff

predictions_df['final_pred'] = (y_pred_proba_train >= optimal_cutoff).astype(int)

# Display the first few rows
print(f"Optimal cutoff used for final prediction: {optimal_cutoff}")
predictions_df.head()

```

[148] ✓ 0.0s

```
Optimal cutoff used for final prediction: 0.5000000000000001
```

	actual	pred_0.3	pred_0.4	pred_0.5	pred_0.6	pred_0.7	final_pred
0	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

Model performance at Optimal - Cutoff

```
# Check the accuracy now

from sklearn.metrics import accuracy_score

# Calculate accuracy based on the final prediction
accuracy = accuracy_score(predictions_df['actual'], predictions_df['final_pred'])

print(f"Model Accuracy at Optimal Cutoff ({optimal_cutoff:.2f}): {accuracy:.4f}")


```

[149] ✓ 0.0s

```
Model Accuracy at Optimal Cutoff (0.50): 0.7818
```

```
# Create the confusion matrix once again

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(predictions_df['actual'], predictions_df['final_pred'])

# Display results
print("Confusion Matrix:\n", cm)

```

[150] ✓ 0.0s

```
Confusion Matrix:
 [[407 120]
 [110 417]]
```

```
# Create variables for true positive, true negative, false positive and false negative
# Extract individual components
tn, fp, fn, tp = cm.ravel()

# Display results
print(f"\nTrue Positives (TP): {tp}")
print(f"True Negatives (TN): {tn}")
print(f"False Positives (FP): {fp}")
print(f"False Negatives (FN): {fn}")
```

[151] ✓ 0.0s

```
True Positives (TP): 417
True Negatives (TN): 407
False Positives (FP): 120
False Negatives (FN): 110
```

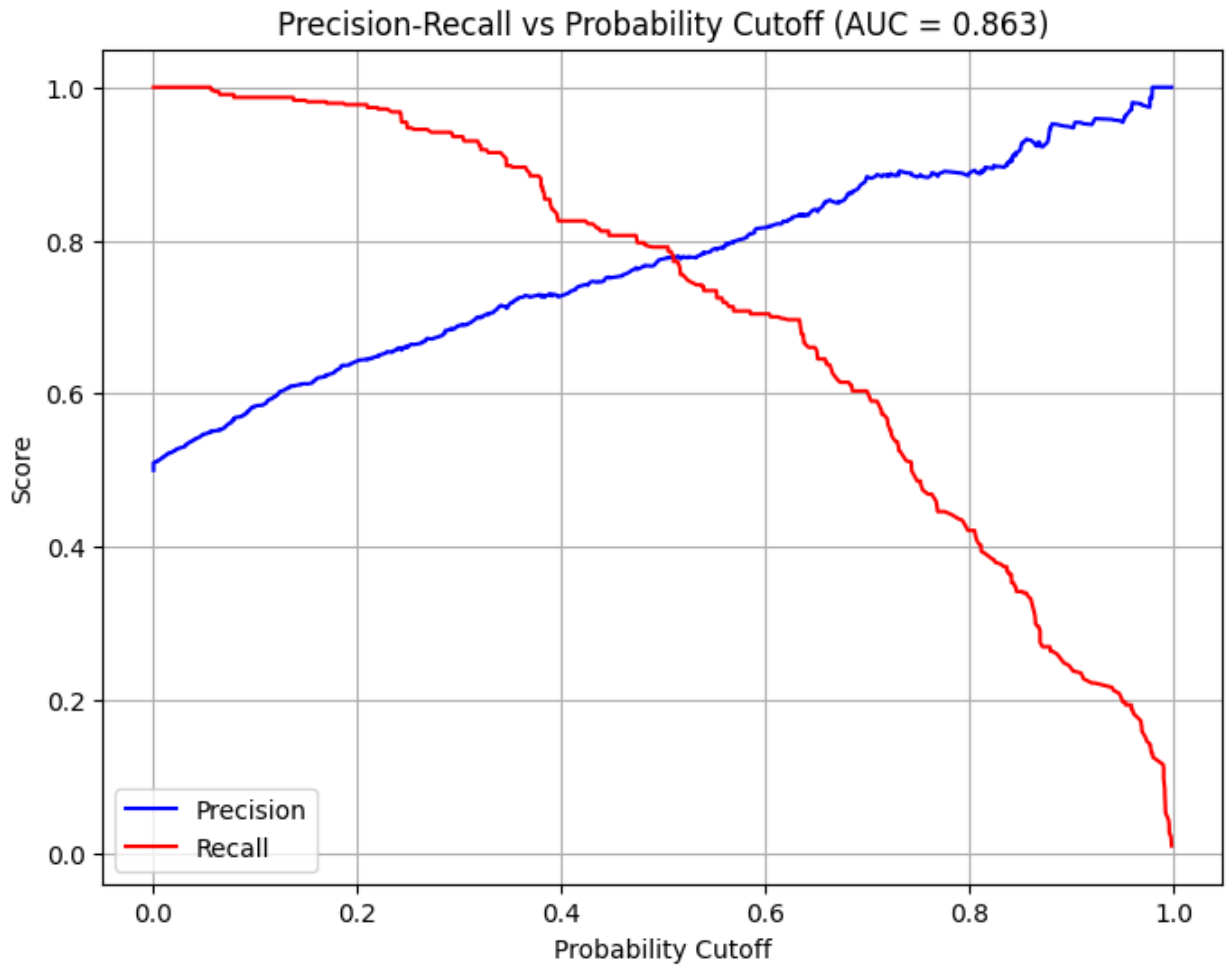
```python
# Calculate the sensitivity
sensitivity = tp / (tp + fn)

# Calculate the specificity
specificity = tn / (tn + fp)

# Calculate Precision
precision = tp / (tp + fp)

# Calculate Recall
recall = sensitivity

# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)

# ✅ Print all metrics neatly
print(f"Sensitivity (Recall): {sensitivity:.3f}")
print(f"Specificity: {specificity:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1 Score: {f1_score:.3f}")



```

[152] ✓ 0.0s

```
Sensitivity (Recall): 0.791
Specificity: 0.772
Precision: 0.777
Recall: 0.791
F1 Score: 0.784
```

```python
# Plot precision-recall curve
import numpy as np
import matplotlib.pyplot as plt

# Get predicted probabilities from your fitted statsmodels Logit model
y_pred_proba_train = result.predict(X_train_sm)  # X_train_sm includes constant

# Compute precision, recall, and thresholds
precision, recall, thresholds = precision_recall_curve(y_train, y_pred_proba_train)

# Compute area under the precision-recall curve (optional)
pr_auc = auc(recall, precision)

# Plot Precision-Recall vs Threshold curve
plt.figure(figsize=(8,6))
plt.plot(thresholds, precision[:-1], label='Precision', color='blue')
plt.plot(thresholds, recall[:-1], label='Recall', color='red')
plt.title(f'Precision-Recall vs Probability Cutoff (AUC = {pr_auc:.3f})')
plt.xlabel('Probability Cutoff')
plt.ylabel('Score')
plt.legend(loc='best')
plt.grid(True)
plt.show()

# Find the cutoff where Precision and Recall are closest
diff = np.abs(precision[:-1] - recall[:-1])
optimal_cutoff = thresholds[np.argmin(diff)]

print(f"🔹 Optimal Cutoff (Precision ≈ Recall): {optimal_cutoff:.3f}")

```

[154] ✓ 0.2s



- Optimal Cutoff (Precision ≈ Recall): 0.510

Build Random Forest Base model

```
# Import necessary libraries
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score, GridSearchCV
```

[155] ✓ 0.0s

```
# Convert all boolean columns in both train and test to int (0/1)
bool_cols = X_train.select_dtypes(include='bool').columns

for col in bool_cols:
    X_train[col] = X_train[col].astype(int)
    X_test[col] = X_test[col].astype(int)

```

[156] ✓ 0.0s

```
X_train.head()
```

[157] ✓ 0.0s Python

te_IN	policy_state_OH	insured_sex_MALE	insured_occupation_adm-clerical	insured_occupation_armed-forces	insured_occupation_craft-repair	insured_occupation_exec-managerial	insured_occupation_machine-op-inspct	insured_occupation_othe servi
1	0	1	0	0	0	0	0	
0	1	1	0	0	0	0	0	
1	0	1	0	0	0	0	0	
0	1	0	0	0	0	0	0	
1	0	1	1	0	0	0	0	

```
X_test.head()
```

[158] ✓ 0.0s Python

e_OH	insured_sex_MALE	insured_occupation_adm-clerical	insured_occupation_armed-forces	insured_occupation_craft-repair	insured_occupation_exec-managerial	insured_occupation_machine-op-inspct	insured_occupation_other-service	insured_occu
0	1	0	0	0	0	0	0	
1	0	0	0	0	0	0	0	
1	1	0	0	0	0	0	0	
0	1	0	0	0	0	0	0	
1	1	0	0	0	0	0	1	

```python
# Build a base random forest model

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, roc_auc_score

# Initialize base Random Forest model
rf_base = RandomForestClassifier(
    n_estimators=100,       # number of trees (default 100)
    random_state=42,        # reproducibility
    n_jobs=-1               # use all CPU cores
)

#  Train the model
rf_base.fit(X_train, y_train)

#  Predict on training data
y_pred_train = rf_base.predict(X_train)
y_pred_proba_train = rf_base.predict_proba(X_train)[:, 1]

#  Evaluate performance
print("Training Accuracy:", accuracy_score(y_train, y_pred_train))
print("\nConfusion Matrix:\n", confusion_matrix(y_train, y_pred_train))
print("\nClassification Report:\n", classification_report(y_train, y_pred_train))
print("ROC-AUC Score:", roc_auc_score(y_train, y_pred_proba_train))


```

[159] ✓ 0.3s

```
Training Accuracy: 1.0

Confusion Matrix:
 [[527   0]
 [  0 527]]

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       527
```

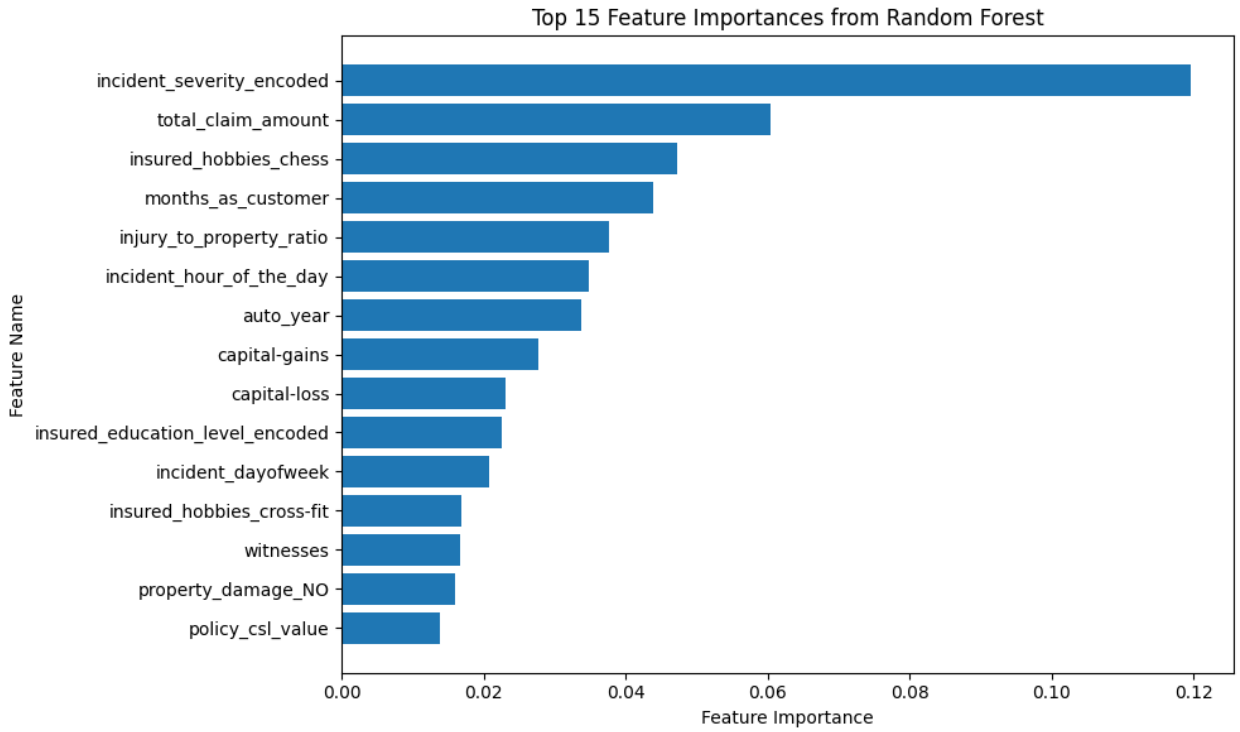
Get Important features

```python
# Get feature importance scores from the trained model
print(rf_base.feature_importances_)

# Create a DataFrame to visualise the importance scores
feature_importances = pd.DataFrame({
    'Feature': X_train.columns,
    'Importance': rf_base.feature_importances_
}).sort_values(by='Importance', ascending=False)

#  Display top 10 most important features
print("Top 10 Important Features:")
print(feature_importances.head(10))

#  Plot feature importances
plt.figure(figsize=(10, 6))
plt.barh(feature_importances['Feature'][:15][::-1],
         feature_importances['Importance'][:15][::-1])
plt.xlabel("Feature Importance")
plt.ylabel("Feature Name")
plt.title("Top 15 Feature Importances from Random Forest")
plt.tight_layout()
plt.show()


```

[160] ✓ 0.2s

```
[0.0437787  0.01312615 0.01233402 0.02774246 0.02309953 0.03485778
 0.01058891 0.01216977 0.01674807 0.06043605 0.03380576 0.01376597
 0.03770211 0.00744944 0.         0.00732629 0.02076845 0.00684092
 0.00609078 0.00682036 0.00548772 0.00402325 0.00651912 0.00328321
 0.00542902 0.00428542 0.00238578 0.00469212 0.00333975 0.00244563
 0.00306053 0.00485836 0.01049051 0.00412439 0.04725756 0.0168125
 0.00535239 0.00294621 0.00277677 0.00546341 0.00671549 0.00502669
 0.00598422 0.00450233 0.00355992 0.00864543 0.00536506 0.00706791
 0.00710284 0.00561013 0.00607701 0.00600322 0.00547066 0.00453438
 0.00659417 0.00707225 0.00439925 0.0104494  0.00607512 0.00573852
 0.0045062  0.0051541  0.00354074 0.00506328 0.00542147 0.01587747
 0.00722567 0.00757295 0.00605849 0.00365353 0.00404693 0.00327213
 0.00415967 0.00542452 0.00282271 0.00407444 0.00299489 0.00343748
 0.00330842 0.00216622 0.00372567 0.00279982 0.00283195 0.00110209
 0.00154654 0.00235528 0.00241631 0.00064013 0.00163485 0.0005934
```

```
Top 15 Important Features:
                            Feature  Importance
122        incident_severity_encoded    0.119669
9                 total_claim_amount    0.060436
34             insured_hobbies_chess    0.047258
0                 months_as_customer    0.043779
12          injury_to_property_ratio    0.037702
5           incident_hour_of_the_day    0.034858
10                         auto_year    0.033806
3                    capital-gains     0.027742
4                    capital-loss      0.023100
121  insured_education_level_encoded    0.022563
16                incident_dayofweek    0.020768
35         insured_hobbies_cross-fit    0.016812
8                          witnesses    0.016748
65               property_damage_NO     0.015877
11                 policy_csl_value    0.013766
```



Train Random Forest model on Selected 15 top features

```python
# Select features with high importance scores


# Create a new training data with only the selected features


# Define a threshold or number of top features
# Option 1: Select top N features
top_n = 15
top_features = feature_importances['Feature'][:top_n].tolist()

# Option 2: Select features above a certain importance threshold
# threshold = 0.01
# top_features = importance_df[importance_df['Importance'] > threshold]['Feature'].tolist()

# 2. Create a new training DataFrame with only the selected features
X_train_selected = X_train[top_features]

# 3. Similarly, select columns in test data (if needed)
X_test_selected = X_test[top_features]

print(f"Selected features ({len(top_features)}):", top_features)
X_train_selected.head()
```

[436] ✓ 0.0s Python

```
Selected features (15): ['incident_severity_encoded', 'total_claim_amount', 'insured_hobbies_chess', 'months_as_customer', 'injury_to_property_ratio', 'incident_hour_of_the_day', 'auto
```

	incident_severity_encoded	total_claim_amount	insured_hobbies_chess	months_as_customer	injury_to_property_ratio	incident_hour_of_the_day	auto_year	capital-gains	capital-loss	insured_education_lev
0	-0.955504	0.818421	0	1.915629	-0.110002	0.626985	0.703048	-0.933658	-0.058181	
1	-0.955504	-0.630992	0	-1.811300	-0.110758	-0.822867	-0.471830	1.296027	0.534189	
2	-0.955504	-1.930113	0	-1.402035	-0.110759	0.916956	1.038728	1.804425	-0.923679	
3	-2.121131	-1.935621	0	0.653001	-0.110759	-0.387911	-0.303991	-0.933658	0.715092	

```python
# Fit the model on the training data with selected features

# Initialize the Random Forest classifier (you can adjust hyperparameters)
rf_model_selected = RandomForestClassifier(
    n_estimators=100,
    random_state=42,
    max_depth=None,
    min_samples_split=2
)

# Fit the model on the selected features
rf_model_selected.fit(X_train_selected, y_train)
```

[434] ✓ 0.2s

```
RandomForestClassifier
▶ Parameters
```

```python
# Generate predictions on training data
y_train_pred = rf_model_selected.predict(X_train_selected)
```

[435] ✓ 0.0s

Performance Metrics on Training Dataset

```python
# Check accuracy of the model

# Evaluate performance on training set
train_accuracy = accuracy_score(y_train, y_train_pred)
print("Training Accuracy:", train_accuracy)
```

[165] ✓ 0.0s

```
Training Accuracy: 1.0
```

```python
# Create the confusion matrix to visualise the performance

train_cm = confusion_matrix(y_train, y_train_pred)

print("\nConfusion Matrix:\n", train_cm)
```

[166] ✓ 0.0s

```
Confusion Matrix:
 [[527   0]
 [  0 527]]
```

```python
# Create variables for true positive, true negative, false positive and false negative
# Extract values
tn, fp, fn, tp = train_cm.ravel()

print("True Positive (TP):", tp)
print("True Negative (TN):", tn)
print("False Positive (FP):", fp)
print("False Negative (FN):", fn)
```

[167] ✓ 0.0s

```
True Positive (TP): 527
True Negative (TN): 527
False Positive (FP): 0
False Negative (FN): 0
```

```python
# Calculate the sensitivity
sensitivity = tp / (tp + fn)

# Calculate the specificity
specificity = tn / (tn + fp)

# Calculate Precision
precision = tp / (tp + fp)

# Calculate Recall
recall = sensitivity

# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)

# Print all metrics neatly
print(f"Sensitivity (Recall): {sensitivity:.3f}")
print(f"Specificity: {specificity:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1 Score: {f1_score:.3f}")
```

68] ✓ 0.0s

```
Sensitivity (Recall): 1.000
Specificity: 1.000
Precision: 1.000
Recall: 1.000
F1 Score: 1.000
```

Hyperparameter tuning to find best model parameters

```python
# Use grid search to find the best hyperparamter values

# Best Hyperparameters

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],          # number of trees
    'max_depth': [None, 5, 10, 20],          # maximum depth of the tree
    'min_samples_split': [2, 5, 10],         # minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],           # minimum samples required at a leaf node
    'max_features': ['auto', 'sqrt', 'log2'] # number of features to consider at each split
}

# Initialize Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    cv=5,                 # 5-fold cross-validation
    scoring='accuracy',   # metric to optimize
    n_jobs=-1,            # use all cores
    verbose=2
)

# Fit GridSearchCV
grid_search.fit(X_train_selected, y_train)

# Best hyperparameters
print("Best Parameters:", grid_search.best_params_)

# Best score
print("Best CV Accuracy:", grid_search.best_score_)

# Train final model with best parameters
best_rf_model = grid_search.best_estimator_

[447]
```

```
[447]
...  Fitting 5 folds for each of 324 candidates, totalling 1620 fits
     Best Parameters: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 200}
     Best CV Accuracy: 0.9297991424057775
```

Build new Random Forest model with Best parameters provided by GridSearchCV

```python
# Building random forest model based on results of hyperparameter tuning

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

# 1️⃣ Initialize the Random Forest model with the best hyperparameters
final_rf_model = RandomForestClassifier(
    n_estimators=grid_search.best_params_['n_estimators'],
    max_depth=grid_search.best_params_['max_depth'],
    min_samples_split=grid_search.best_params_['min_samples_split'],
    min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
    max_features=grid_search.best_params_['max_features'],
    random_state=42
)

# 2️⃣ Fit the model on the training data
final_rf_model.fit(X_train_selected, y_train)


```

[448] ✓ 0.3s

```
RandomForestClassifier
▶ Parameters
```

```python
# Make predictions on training data

# Predict on training data (or test data if available)
y_pred_train = final_rf_model.predict(X_train_selected)


print("\nClassification Report:\n", classification_report(y_train, y_pred_train))

roc_auc = roc_auc_score(y_train, y_pred_proba_train)
print("ROC-AUC:", roc_auc)
```

[449] ✓ 0.0s

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

           0       1.00      1.00      1.00       527
           1       1.00      1.00      1.00       527

    accuracy                           1.00      1054
   macro avg       1.00      1.00      1.00      1054
weighted avg       1.00      1.00      1.00      1054

ROC-AUC: 1.0
```

```
# Check the accuracy
train_accuracy = accuracy_score(y_train, y_pred_train)
print("Training Accuracy:", train_accuracy)
```

[450] ✓ 0.0s

```
Training Accuracy: 1.0
```

```
# Create the confusion matrix
cm = confusion_matrix(y_train, y_pred_train)
print("Confusion Matrix:\n", cm)
```

[451] ✓ 0.0s

```
Confusion Matrix:
 [[527   0]
 [  0 527]]
```

```
# Create variables for true positive, true negative, false positive and false negative

# Extract values
tn, fp, fn, tp = cm.ravel()

print("True Positive (TP):", tp)
print("True Negative (TN):", tn)
print("False Positive (FP):", fp)
print("False Negative (FN):", fn)
```

[452] ✓ 0.0s

```
True Positive (TP): 527
True Negative (TN): 527
False Positive (FP): 0
False Negative (FN): 0
```

```python
# Calculate the sensitivity
sensitivity = tp / (tp + fn)

# Calculate the specificity
specificity = tn / (tn + fp)

# Calculate Precision
precision = tp / (tp + fp)

# Calculate Recall
recall = sensitivity

# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)

# Print all metrics neatly
print(f"Sensitivity (Recall): {sensitivity:.3f}")
print(f"Specificity: {specificity:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1 Score: {f1_score:.3f}")
```

[453] ✓ 0.0s

```
Sensitivity (Recall): 1.000
Specificity: 1.000
Precision: 1.000
Recall: 1.000
F1 Score: 1.000
```

```python
# Use cross validation to check if the model is overfitting



from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
import numpy as np

# Initialize Random Forest model
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# 1 Evaluate training accuracy
rf_model.fit(X_train_selected, y_train)
train_score = rf_model.score(X_train_selected, y_train)
print("Training Accuracy:", train_score)

# Evaluate using cross-validation (e.g., 5-fold)
cv_scores = cross_val_score(rf_model, X_train_selected, y_train, cv=5, scoring='accuracy')
print("Cross-Validation Accuracy Scores:", cv_scores)
print("Mean CV Accuracy:", np.mean(cv_scores))
print("Standard Deviation CV Accuracy:", np.std(cv_scores))

# Interpretation:
# - If training accuracy >> mean CV accuracy, your model may be overfitting.
# - If training accuracy ≈ mean CV accuracy, model generalizes well.


[169] ✓ 2.0s
```

```
Training Accuracy: 1.0
Cross-Validation Accuracy Scores: [0.90521327 0.92890995 0.95260664 0.9478673  0.95238095]
Mean CV Accuracy: 0.9373956217558114
Standard Deviation CV Accuracy: 0.018293722484981 45
```

Accuracy is 100 % Model is overfitting

Use hyperparameter tunning and cross validation to reduce overfitting and identify best parameters

```python
# Use grid search to find the best hyperparamter values

# Best Hyperparameters

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],          # number of trees
    'max_depth': [None, 5, 10, 20],          # maximum depth of the tree
    'min_samples_split': [2, 5, 10],         # minimum samples required to split a node
    'min_samples_leaf': [1, 2, 4],           # minimum samples required at a leaf node
    'max_features': ['auto', 'sqrt', 'log2'] # number of features to consider at each split
}

# Initialize Random Forest model
rf_model = RandomForestClassifier(random_state=42)

# Initialize GridSearchCV
grid_search = GridSearchCV(
    estimator=rf_model,
    param_grid=param_grid,
    cv=5,                # 5-fold cross-validation
    scoring='accuracy',  # metric to optimize
    n_jobs=-1,           # use all cores
    verbose=2
)

# Fit GridSearchCV
grid_search.fit(X_train_selected, y_train)

# Best hyperparameters
print("Best Parameters:", grid_search.best_params_)

# Best score
print("Best CV Accuracy:", grid_search.best_score_)

# Train final model with best parameters
best_rf_model = grid_search.best_estimator_
```

[170]

Fitting 5 folds for each of 324 candidates, totalling 1620 fits

Best Parameters: {'max_depth': None, 'max_features': 'log2', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 100}

Best CV Accuracy: 0.95447528774543

```python
# Building random forest model based on results of hyperparameter tuning

from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score

# 1️⃣ Initialize the Random Forest model with the best hyperparameters
final_rf_model = RandomForestClassifier(
    n_estimators=grid_search.best_params_['n_estimators'],
    max_depth=grid_search.best_params_['max_depth'],
    min_samples_split=grid_search.best_params_['min_samples_split'],
    min_samples_leaf=grid_search.best_params_['min_samples_leaf'],
    max_features=grid_search.best_params_['max_features'],
    random_state=42
)

# 2️⃣ Fit the model on the training data
final_rf_model.fit(X_train_selected, y_train)


```

[171] ✓ 0.3s

```
RandomForestClassifier ❶ ❷
▶ Parameters
```

```python
# Make predictions on training data

# Predict on training data (or test data if available)
y_pred_train = final_rf_model.predict(X_train_selected)


print("\nClassification Report:\n", classification_report(y_train, y_pred_train))

roc_auc = roc_auc_score(y_train, y_pred_proba_train)
print("ROC-AUC:", roc_auc)
```

[172] ✓ 0.0s

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

           0       1.00      1.00      1.00       527
           1       1.00      1.00      1.00       527

    accuracy                           1.00      1054
   macro avg       1.00      1.00      1.00      1054
weighted avg       1.00      1.00      1.00      1054

ROC-AUC: 1.0
```

```
[173]
# Check the accuracy
train_accuracy = accuracy_score(y_train, y_pred_train)
print("Training Accuracy:", train_accuracy)
```

✓ 0.0s

```
Training Accuracy: 1.0
```

```
[174]
# Create the confusion matrix
cm = confusion_matrix(y_train, y_pred_train)
print("Confusion Matrix:\n", cm)
```

✓ 0.0s

```
Confusion Matrix:
 [[527   0]
 [  0 527]]
```

```
[175]
# Create variables for true positive, true negative, false positive and false negative

# Extract values
tn, fp, fn, tp = cm.ravel()

print("True Positive (TP):", tp)
print("True Negative (TN):", tn)
print("False Positive (FP):", fp)
print("False Negative (FN):", fn)
```

✓ 0.0s

```
True Positive (TP): 527
True Negative (TN): 527
False Positive (FP): 0
False Negative (FN): 0
```

```python
# Calculate the sensitivity
sensitivity = tp / (tp + fn)

# Calculate the specificity
specificity = tn / (tn + fp)

# Calculate Precision
precision = tp / (tp + fp)

# Calculate Recall
recall = sensitivity

# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)

# Print all metrics neatly
print(f"Sensitivity (Recall): {sensitivity:.3f}")
print(f"Specificity: {specificity:.3f}")
print(f"Precision: {precision:.3f}")
print(f"Recall: {recall:.3f}")
print(f"F1 Score: {f1_score:.3f}")
```

[453] ✓ 0.0s

```
Sensitivity (Recall): 1.000
Specificity: 1.000
Precision: 1.000
Recall: 1.000
F1 Score: 1.000
```

Prediction on Validation dataset - Logistic Regression

```
[482] ✓ 0.0s
```

```python
# Select the relevant features for validation data
import pandas as pd
import statsmodels.api as sm
from sklearn.metrics import confusion_matrix, classification_report

# 1️⃣ Select the same features as used in training
X_val = X_test[X_train_vif.columns.drop('const')]  # exclude 'const' if it exists in X_train_vif

# 2️⃣ Add constant using statsmodels
X_val_sm = sm.add_constant(X_val, has_constant='add')

```

```
[481] ✓ 0.0s
```

```python
# Make predictions on the validation data and store it in the variable 'y_validation_pred'
y_validation_pred = result.predict(X_val_sm)
y_validation_pred.head()
```

```
0    0.709864
1    0.204609
2    0.405800
3    0.610457
4    0.482279
dtype: float64
```

```
[490] ✓ 0.0s
```

```python
#  Create DataFrame with actual values and predicted values for validation data
predictions_val_df = pd.DataFrame({
    'actual': y_test_resampled,
    'pred_proba': y_validation_pred
})
predictions_val_df.head()
```

	actual	pred_proba
0	0	0.709864
1	1	0.204609
2	0	0.405800
3	0	0.610457
4	0	0.482279

```
# Make final predictions on the validation data using the optimal cutoff
# Predict classes using optimal cutoff
optimal_cutoff = 0.5  # replace with your actual optimal cutoff if different
y_pred_val = (y_validation_pred >= optimal_cutoff).astype(int)
y_pred_val.head()
```

[491] ✓ 0.0s

```
0    1
1    0
2    0
3    1
4    0
dtype: int32
```

Performance Metrics

```
# Check the accuracy
metrics_train = evaluate_logit_model(result, X_val_sm, y_test_resampled,"accuracy")
```

[492] ✓ 0.0s

```
Accuracy: 0.5420
```

```
# Create the confusion matrix
metrics_train = evaluate_logit_model(result, X_val_sm, y_test_resampled,"confusion_matrix")
```

[462] ✓ 0.0s

```
Confusion Matrix:
[[162  64]
[143  83]]
```

```
# Create variables for true positive, true negative, false positive and false negative

# Extract confusion matrix from dictionary
cm = metrics_train["confusion_matrix"]

# Unpack values
TN, FP, FN, TP = cm.ravel()

# Store in variables
true_positive = TP
true_negative = TN
false_positive = FP
false_negative = FN

# Check
print("TP:", true_positive)   # 417
print("TN:", true_negative)   # 407
print("FP:", false_positive)  # 120
print("FN:", false_negative)  # 110

```

[463] ✓ 0.0s

```
TP: 83
TN: 162
FP: 64
FN: 143
```

```python
# Calculate the sensitivity
sensitivity = TP / (TP + FN)
print(f"Sensitivity (Recall): {sensitivity:.4f}")

# Calculate the specificity
specificity = TN / (TN + FP)
print(f"Specificity: {specificity:.4f}")

# Calculate Precision
precision = TP / (TP + FP)
print(f"Precision: {precision:.4f}")

# Calculate Recall
recall = sensitivity
print(f"Recall: {recall:.4f}")

# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score: {f1_score:.4f}")


```

[464] ✓ 0.0s

```
Sensitivity (Recall): 0.3673
Specificity: 0.7168
Precision: 0.5646
Recall: 0.3673
F1 Score: 0.4450
```

Make predictions over validation data using random forest model

Features selected

```python
X_test_selected = X_test[top_features]
X_test_selected.info()
```

[495] ✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 452 entries, 0 to 451
Data columns (total 15 columns):
 #   Column                          Non-Null Count  Dtype
---  ------                          --------------  -----
 0   incident_severity_encoded       452 non-null    float64
 1   total_claim_amount              452 non-null    float64
 2   insured_hobbies_chess           452 non-null    int32
 3   months_as_customer              452 non-null    float64
 4   injury_to_property_ratio        452 non-null    float64
 5   incident_hour_of_the_day        452 non-null    float64
 6   auto_year                       452 non-null    float64
 7   capital-gains                   452 non-null    float64
 8   capital-loss                    452 non-null    float64
 9   insured_education_level_encoded 452 non-null    float64
 10  incident_dayofweek              452 non-null    float64
 11  insured_hobbies_cross-fit       452 non-null    int32
 12  witnesses                       452 non-null    float64
 13  property_damage_NO              452 non-null    int32
 14  policy_csl_value                452 non-null    float64
dtypes: float64(12), int32(3)
memory usage: 47.8 KB
```

```
[496]
1 rf_model_selected.feature_importances_,top_features,X_test.shape
✓ 0.0s
```

```
(array([0.22641463, 0.11308243, 0.08381893, 0.08827736, 0.07482311,
        0.0642497 , 0.06149126, 0.05068101, 0.04539638, 0.04160308,
        0.04206373, 0.03256975, 0.03335165, 0.02045686, 0.02172012]),
 ['incident_severity_encoded',
  'total_claim_amount',
  'insured_hobbies_chess',
  'months_as_customer',
  'injury_to_property_ratio',
  'incident_hour_of_the_day',
  'auto_year',
  'capital-gains',
  'capital-loss',
  'insured_education_level_encoded',
  'incident_dayofweek',
  'insured_hobbies_cross-fit',
  'witnesses',
  'property_damage_NO',
  'policy_csl_value'],
 (452, 123))
```

Predictions on Test dataset

```
[497]
1 # Make predictions on the validation data
2
3 y_test_pred = rf_model_selected.predict(X_test_selected)
4
5 # Check predictions
6 print("Predicted classes (first 10 rows):")
7 print(y_test_pred[:10])
8
✓ 0.0s
```

```
Predicted classes (first 10 rows):
[0 1 0 0 0 0 0 1 0 0]
```

Performance Evaluation

```
[499]
1 # Check accuracy
2
3 # Evaluate performance on test set
4 test_accuracy = accuracy_score(y_test_resampled, y_test_pred)
5 print("Test Accuracy:", test_accuracy)
✓ 0.0s
```

```
Test Accuracy: 0.7323008849557522
```

```python
# Create the confusion matrix
cm = confusion_matrix(y_test_resampled, y_test_pred)
print("Confusion Matrix:\n", cm)
```

[500] ✓ 0.0s

```
Confusion Matrix:
 [[203  23]
 [ 98 128]]
```

```python
# Create variables for true positive, true negative, false positive and false negative

# Extract confusion matrix from dictionary
cm = metrics_train["confusion_matrix"]

# Unpack values
TN, FP, FN, TP = cm.ravel()

# Store in variables
true_positive = TP
true_negative = TN
false_positive = FP
false_negative = FN

# Check
print("TP:", true_positive)
print("TN:", true_negative)
print("FP:", false_positive)
print("FN:", false_negative)

```

[502] ✓ 0.0s

```
TP: 83
TN: 162
FP: 64
FN: 143
```

```python
# Calculate the sensitivity
sensitivity = TP / (TP + FN)
print(f"Sensitivity (Recall): {sensitivity:.4f}")

# Calculate the specificity
specificity = TN / (TN + FP)
print(f"Specificity: {specificity:.4f}")

# Calculate Precision
precision = TP / (TP + FP)
print(f"Precision: {precision:.4f}")

# Calculate Recall
recall = sensitivity
print(f"Recall: {recall:.4f}")

# Calculate F1 Score
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score: {f1_score:.4f}")


```

[503] ✓ 0.0s

```
Sensitivity (Recall): 0.3673
Specificity: 0.7168
Precision: 0.5646
Recall: 0.3673
F1 Score: 0.4450
```

Model Results on Test Dataset with Target Encoding for Categorical features

Logistic Regression

```
[503]
# Check the accuracy
metrics_train = evaluate_logit_model(result, X_val_sm, y_test,"accuracy")
✓ 0.0s

Accuracy: 0.8761
```

Features selected

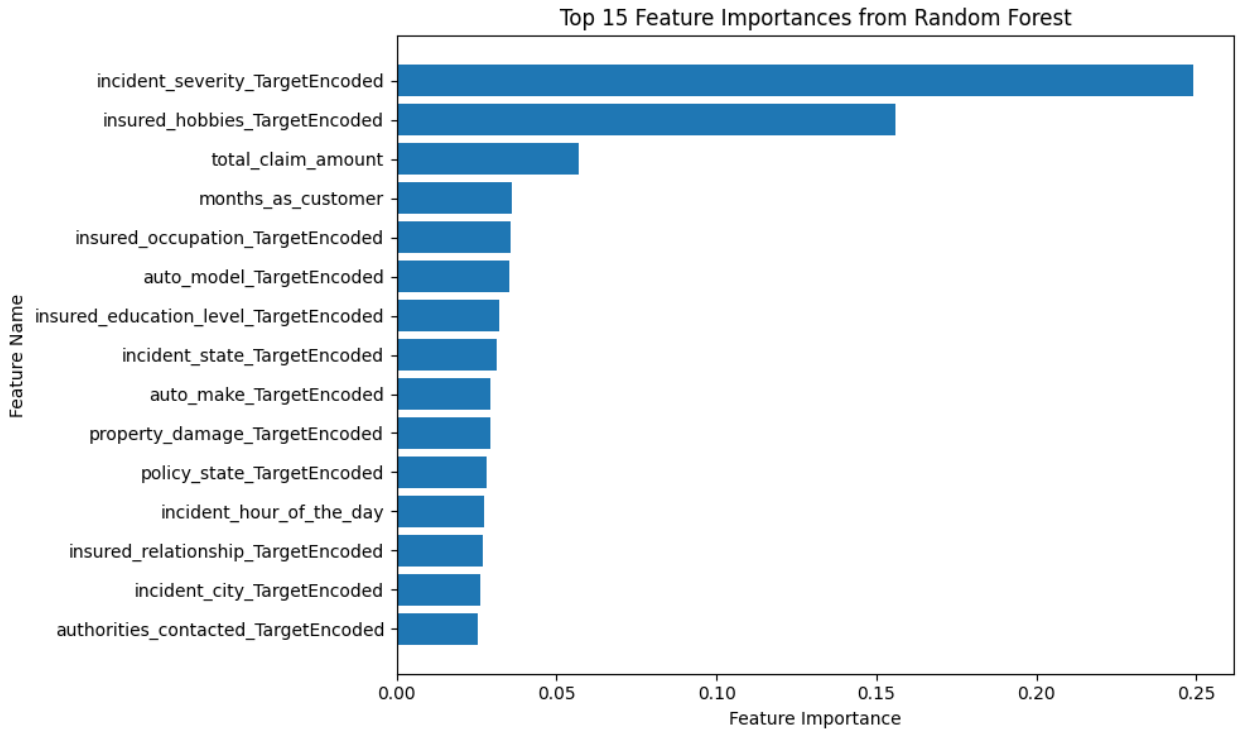
['insured_hobbies_TargetEncoded', 'incident_severity_TargetEncoded']

Random Forest

```
[524]
# Make predictions on test data

# Predict on training data (or test data if available)
y_pred_test = final_rf_model.predict(X_test)


print("\nClassification Report:\n", classification_report(y_test, y_pred_test))

roc_auc = roc_auc_score(y_test, y_pred_test)
print("ROC-AUC:", roc_auc)
✓ 0.0s

Classification Report:
              precision    recall  f1-score   support

           0       0.77      0.87      0.82       226
           1       0.85      0.74      0.79       226

    accuracy                           0.81       452
   macro avg       0.81      0.81      0.80       452
weighted avg       0.81      0.81      0.80       452

ROC-AUC: 0.8053097345132744
```

```
[525]
# Check the accuracy
test_accuracy = accuracy_score(y_test, y_pred_test)
print("Test Accuracy:", test_accuracy)
✓ 0.0s

Test Accuracy: 0.8053097345132744
```

Top 15 Important features selected



Evaluation and Conclusion

In this project, we developed and evaluated machine learning models to detect fraudulent insurance claims using a comprehensive dataset from Global Insure. The workflow included data cleaning, feature engineering, handling class imbalance, and extensive exploratory data analysis. Two models were built and compared: Logistic Regression and Random Forest.

Key Findings:

- **Data Preparation & Cleaning:** Redundant and irrelevant features were removed, missing values handled, and categorical variables grouped to improve model generalization.
- **Feature Engineering:** New features were created from existing columns, and highly correlated features were dropped to reduce multicollinearity.
- **Class Imbalance:** RandomOverSampler was used to balance the dataset, ensuring fair model training.
- **Model Building:**
    - *Logistic Regression* with RFECV feature selection showed interpretable results, with statistically significant predictors and acceptable VIFs.
    - *Random Forest* provided higher accuracy and better performance on both training and validation sets, with feature importance analysis highlighting key predictors.

- **Model Evaluation:** Both models were assessed using accuracy, sensitivity, specificity, precision, recall, and F1-score. Random Forest generally outperformed Logistic Regression, especially after hyperparameter tuning.
- **Optimal Cutoff:** ROC and precision-recall curves were used to select the best probability threshold, balancing sensitivity and specificity.

Business Impact & Recommendations:

- The Random Forest model, with its higher predictive power and robustness, is recommended for deployment to flag potentially fraudulent claims early in the process.
- Important features identified (such as claim ratios, incident details, and customer profiles) can be used to refine manual review processes and guide further investigation.
- Regular retraining and monitoring of the model are advised to adapt to evolving fraud patterns.

Conclusion:
A data-driven approach using machine learning significantly improves the detection of fraudulent claims, reducing financial losses and operational inefficiencies for Global Insure. The developed models provide actionable insights and a scalable solution for early fraud identification.

Important Note -

It appears that using dummy variables resulted in lower model performance, with both training and test accuracy being relatively poor. Logistic Regression achieved around **54% accuracy** on the test dataset, while the Random Forest model reached approximately **73% accuracy**.

After rebuilding the model using **Target Encoding** for categorical variables, the performance improved significantly. Logistic Regression achieved about **88% accuracy** on the test dataset, and Random Forest reached around **81% accuracy**.

Overall, **Logistic Regression with Target Encoding** outperformed the Random Forest model. However, to further enhance model performance, **additional training data** would be beneficial.