

Insurance Fraudulent Claim Detection

Problem Statement

Global Insure, a leading insurance company, processes thousands of claims annually. However, a significant percentage of these claims turn out to be fraudulent, resulting in considerable financial losses. The company's current process to identify fraudulent claims involves manual inspections, which is time-consuming and inefficient. Fraudulent claims are often detected too late in the process, after the company has already paid out significant amounts. Global Insure wants to improve its fraud detection process using data-driven insights to classify claims as fraudulent or legitimate early in the approval process. This would minimize financial losses and optimize the overall claims handling process.

Business Objective

Global Insure wants to build a model to classify insurance claims as either fraudulent or legitimate based on historical claim details and customer profiles. By using features like claim amounts, customer profiles and claim types, the company aims to predict which claims are likely to be fraudulent before they are approved.

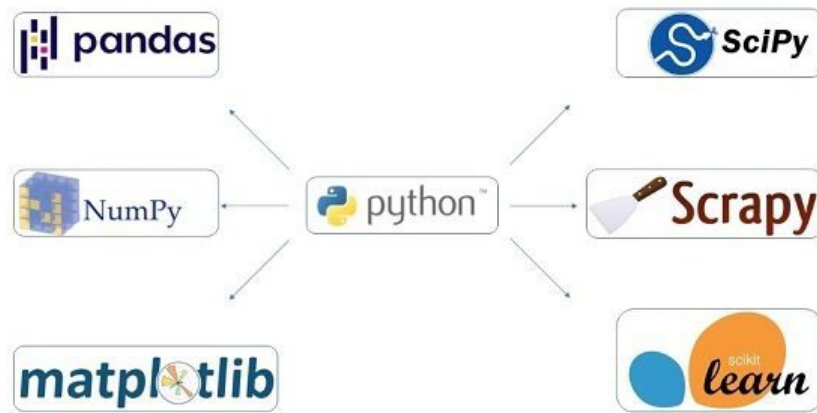
This project utilizes Machine Learning models to assist the Auto Insurance sector in addressing this issue.

Hardware & Software Requirements

- OS: Windows 11 Enterprise, 64 bit
- Processor: 11th Gen Intel(R) Core(TM) i7-1185G7 @ 3.00GHz 3.00 GHz
- RAM: 16 GB
- Anaconda Navigator - Jupyter Notebook

Libraries :

- Numpy
- Pandas
- Matplotlib
- Seaborn
- Scipy
- Date Time
- Scikit Learn



Exploratory Data Analysis (EDA)

Import the necessary Libraries:

```
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from scipy.stats import zscore

from sklearn.preprocessing import PowerTransformer

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

from statsmodels.stats.outliers_influence import variance_inflation_factor

from sklearn import datasets

from sklearn import metrics

from sklearn import model_selection

#from sklearn.metrics import plot_roc_curve

from sklearn.metrics import roc_curve

import warnings

warnings.filterwarnings('ignore')
```

Data Overview

The data used in this case study comprises insurance claims with various attributes such as claim amount, claimant details, type of insurance, and claim status. The dataset includes both genuine and fraudulent claims, labeled accordingly.

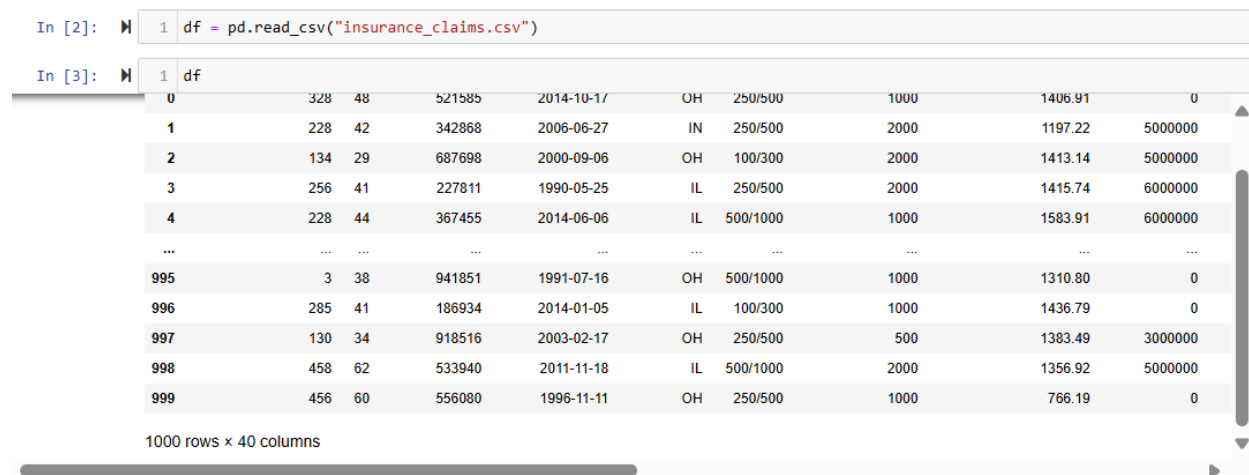
Preprocessing the Data

The first step in detecting fraudulent claims is to preprocess the data. This involves:

- Handling missing values
- Encoding categorical variables
- Normalizing numerical features
- Splitting the data into training and testing sets

Import the Dataset

Importing DataSet



```
In [2]: df = pd.read_csv("insurance_claims.csv")
```

```
In [3]: df
```

0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	5000000
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000
...
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.80	0
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.79	0
997	130	34	918516	2003-02-17	OH	250/500	500	1383.49	3000000
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.92	5000000
999	456	60	556080	1996-11-11	OH	250/500	1000	766.19	0

1000 rows x 40 columns

By examining the dataset, we can determine that it includes both categorical and numerical columns. The "fraud_reported" column serves as our target variable. Since it contains two categories, this problem is classified as a "Classification Problem," where our goal is to predict whether an insurance claim is fraudulent. Given that it is a classification problem, we will employ various classification algorithms during the model-building process, which will be discussed as we proceed with the data analysis.

Data Preparation

In this part we will firstly be exploring the data with some basic steps and then further proceed with some crucial analysis, like feature extraction, imputing and encoding.

Let's start with checking shape, unique values, value counts, info etc.

After doing the analysis if we find any unnecessary columns in the dataset, we can drop those columns.

By using 'df.shape', we determined the number of rows and columns in the dataset, which revealed 1000 rows and 40 columns. Although PCA (Principal Component Analysis) could be applied, we chose not to reduce the data at this stage. Given the relatively small size of the dataset and the fundamental principle for data scientists to retain as much data as possible, we decided to proceed with the entire dataset.

```
# Checking dimension of dataset
df.shape
```

```
(1000, 40)
```

Out of 40 columns 39 are independent columns and remaining one is our target variable "fraud_reported".

```
df.columns
```

```
Index(['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', 'insured_occupation', 'insured_hobbies',
      'insured_relationship', 'capital-gains', 'capital-loss',
      'incident_date', 'incident_type', 'collision_type', 'incident_severity',
      'authorities_contacted', 'incident_state', 'incident_city',
      'incident_location', 'incident_hour_of_the_day',
      'number_of_vehicles_involved', 'property_damage', 'bodily_injuries',
      'witnesses', 'police_report_available', 'total_claim_amount',
      'injury_claim', 'property_claim', 'vehicle_claim', 'auto_make',
      'auto_model', 'auto_year', 'fraud_reported', '_c39'],
      dtype='object')
```

Company's data for insurance claim policy : months_as_customer, age, policy_number(dropped), policy_bind_date, policy_csl, policy_deductable, policy_annual_premium, umbrella_limit

Customer Personal details : insured_zip, insured_sex, insured_education_level, insured_occupation, insured_hobbies, insured_relationship, capital-gains, capital-loss.

Details of the incident : incident_date, incident_type, collision_type, incident_severity, authorities_contacted, incident_state, incident_city, incident_location(dropped), incident_hour_of_the_day, number_of_vehicles_involved, property_damage, bodily_injuries, witnesses, police_report_available, total_amount_claimed, injury_claim, property_claim, vehicle_claim, auto_make, auto_model, auto_year

Target Variable:

fraud_reported : Y-YES / N-NO

```
# To get good overview of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 columns):
#   Column                Non-Null Count  Dtype
---  -
0   months_as_customer    1000 non-null  int64
1   age                   1000 non-null  int64
```

This gives the information about the dataset which includes indexing type, column Name, non-null values, dtypes and memory usage of the dataset.

Here the column _c39 has 0 non null values which means it has NaN throughout the data so we can drop this column.

```
# Dropping _c39 column
df.drop("_c39",axis=1,inplace=True)|
```

Checking the data types using the “df.dtypes” below.

```
# Checking the type of dataset
df.dtypes

months_as_customer    int64
age                   int64
policy_number         int64
policy_bind_date      object
...                   ...
```

Checking the number of unique values of each column in the data set using the code below

```
# Checking number of unique values in each column
df.nunique().to_frame("No of Unique Values")
```

No of Unique Values	
months_as_customer	391
age	46

Checking the value counts of each columns using the code below.

```
# Checking the value counts of each columns
for i in df.columns:
    print(df[i].value_counts())
    print('*'*60)

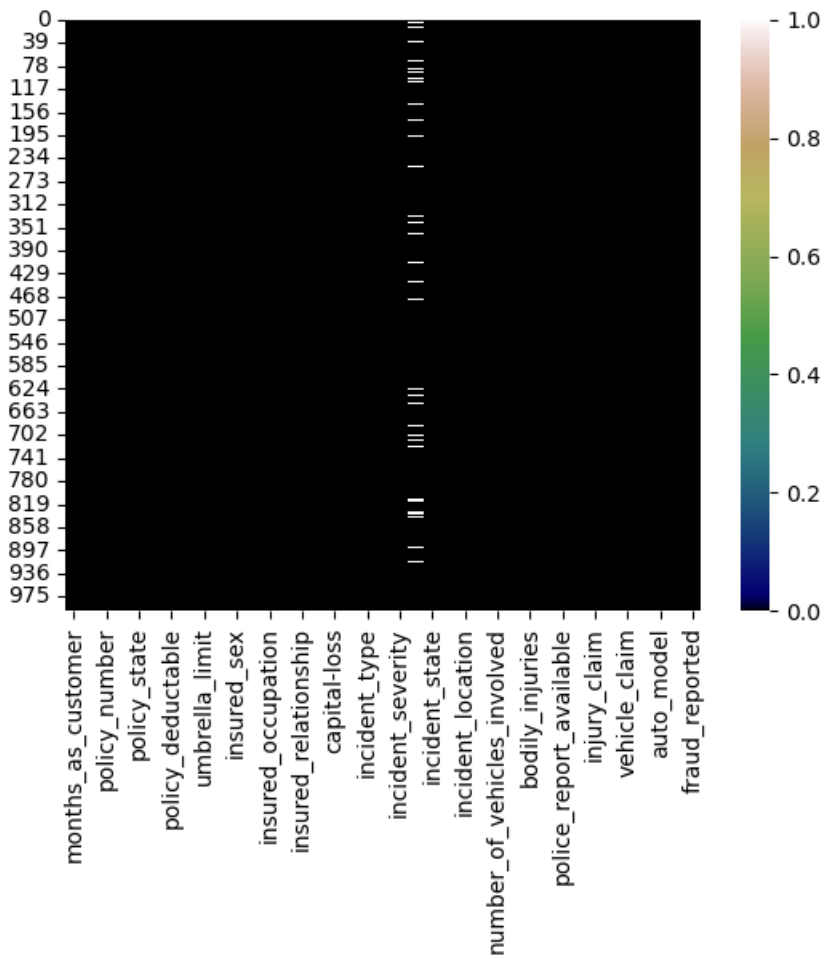
194    8
128    7
254    7
140    7
210    7
...
390    1
411    1
453    1
448    1
17     1
Name: months_as_customer, Length: 391, dtype: int64
*****
43     49
```

Check Null values

```

1 # Visualize the null values through heatmap
2 sns.heatmap(df.isnull(), cmap="gist_earth")
3 plt.show()

```



Observations

- We can see that there are null values in the dataset.
- The dataset contains 3 different types of data namely integer data type, float data type and object data type.
- After analyzing it is seen that c39 column has only entries those are all NaN. Keeping all entries NaN is useless hence dropping that column.

```

# Dropping _c39 column
df.drop("_c39",axis=1,inplace=True)|

```

- We can observe the columns "policy_number" and "incident_location" have 1000 unique values. So, it not required for the prediction so we can drop it.

```

# Dropping policy_number and incident_location columns
df.drop("policy_number",axis=1,inplace=True)
df.drop("incident_location",axis=1,inplace=True)

```

- These are the list of value counts in each columns. By looking at the value counts of each column we can realize that the columns umbrella_limit, capital-gains and capital-loss contains more zero values around 79.8%, 50.8% and 47.5%. I am keeping the zero values in capital_gains and capital_loss columns as it is. Since the umbrella_limit columns has more than 70% of zero values, let's drop that column. Also the collision_type, police_report_available and property_damage have ? in them

```
# Dropping umbrella_limit column
df.drop("umbrella_limit",axis=1,inplace=True)
```

- The column insured_zip is the zip code given to each person. If we take a look at the value count and unique values of the column insured_zip, it contains 995 unique values that means the 5 entries are repeating.

```
# Dropping insured_zip column as it is not important for the prediction
df.drop('insured_zip',axis=1,inplace=True)
```

Feature Engineering

Feature engineering is a crucial step in improving the accuracy of the model. In this case study, we derived new features based on the existing data, such as:

- Claim frequency by the same claimant
- Time taken to file the claim after the incident
- Comparison of claim amount with average claims for similar incidents

Feature Extraction

The policy_bind_date and incident_date have object data type which should be in datetime data type that means the python is not able to understand the type of these columns and giving default data type. We will convert this object data type to datetime data type and we will extract the values from these columns.

```
# Converting Date columns from object type into datetime data type
df['policy_bind_date']=pd.to_datetime(df['policy_bind_date'])
df['incident_date']=pd.to_datetime(df['incident_date'])
```

After converting object data type into datetime data type, we will extract Day, Month and Year from both the columns.

```
# Extracting Day, Month and Year column from policy_bind_date
df["policy_bind_Day"] = df['policy_bind_date'].dt.day
df["policy_bind_Month"] = df['policy_bind_date'].dt.month
df["policy_bind_Year"] = df['policy_bind_date'].dt.year

# Extracting Day, Month and Year column from incident_date
df["incident_Day"] = df['incident_date'].dt.day
df["incident_Month"] = df['incident_date'].dt.month
df["incident_Year"] = df['incident_date'].dt.year
```

After we have extracted Day, Month and Year columns, from both policy_bind_date and incident_date columns. So, we can drop these columns.

```
# Dropping policy_bind_date and incident_date columns
df.drop(["policy_bind_date", "incident_date"], axis=1, inplace=True)
```

Also, we have observed that the feature 'incident-year' has one unique value throughout the column also it is not important for our prediction so we can drop this column.

```
# Dropping incident_Year column
df.drop("incident_Year", axis=1, inplace=True)
```

Here we have extracted age of the vehicle based on auto year by assuming the data is collected in the year 2018 as below and dropped "auto year" column after extraction.

```
# Let's extract age of the vehicle from auto_year by subtracting it from the year 2018
df["Vehicle_Age"] = 2018 - df["auto_year"]
df.drop("auto_year", axis=1, inplace = True)
```

Imputation:

Imputation is a technique to fill null values in the dataset using mean, median or mode. We did not get any null values while checking for the null values, however from the value counts of the columns we have observed that some columns have "?" values, they are not NAN values but we need to fill them.

```
# Checking which columns contains "?" sign
df[df.columns[(df == '?').any()]].nunique()

collision_type      4
property_damage     3
police_report_available  3
dtype: int64
```

These are the columns which contains "?" sign. Since these columns seems to be categorical so we will replace "?" values with most frequently occurring values of the respective columns that is their mode values.

```
# Checking mode of the above columns
print("The mode of collision_type is:", df["collision_type"].mode())
print("The mode of property_damage is:", df["property_damage"].mode())
print("The mode of police_report_available is:", df["police_report_available"].mode())
```

The mode of property damage and police_report_available is "?", which means the data is almost covered by "?" sign. So, we will fill them by the second highest count of the respective column.


```
# Checking value count of property_damage column and police_report_available
print("The value count of property_damage:\n", df["property_damage"].value_counts())
print("\n")
print("The value count of police_report_available:\n", df["police_report_available"].value_counts())
```

The value count of property_damage:

```
?      360
NO     338
YES    302
```

Name: property_damage, dtype: int64

The value count of police_report_available:

```
?      343
NO     343
YES    314
```

Name: police_report_available, dtype: int64

```
1 # Replacing "?" by their mode values
2 df['collision_type'] = df.collision_type.str.replace('?', df['collision_type'].mode()[0])
3 df['property_damage'] = df.property_damage.str.replace('?', "NO")
4 df['police_report_available'] = df.police_report_available.str.replace('?', "NO")
```

```
1 # Checking value count again
2 print("The value count of collision_type:\n", df["collision_type"].value_counts())
3 print("\n")
4 print("The value count of property_damage:\n", df["property_damage"].value_counts())
5 print("\n")
6 print("The value count of police_report_available:\n", df["police_report_available"].value_counts())
```

The value count of collision_type:

```
collision_type
Rear Collision    470
Side Collision    276
Front Collision   254
Name: count, dtype: int64
```

The value count of property_damage:

```
property_damage
NO      698
YES     302
Name: count, dtype: int64
```

The value count of police_report_available:

```
police_report_available
NO      686
YES     314
Name: count, dtype: int64
```

Here we have replaced the "?" sign by mode and "NO" values.

```

1 # Extracting csl_per_person and csl_per_accident from policy_csl column
2 df['csl_per_person'] = df.policy_csl.str.split('/', expand=True)[0]
3 df['csl_per_accident'] = df.policy_csl.str.split('/', expand=True)[1]

```

```

1 # Converting object data type into integer data type
2 df['csl_per_person'] = df['csl_per_person'].astype('int64')
3 df['csl_per_accident'] = df['csl_per_accident'].astype('int64')

```

```

1 # Since we have extracted the data from policy_csl, let's drop that column
2 df.drop("policy_csl",axis=1,inplace=True)

```

```

1 # Let's extract age of the vehicle from auto_year by subtracting it from the year 2018
2 df["Vehicle_Age"] = 2018 - df["auto_year"]
3 df.drop("auto_year",axis=1, inplace = True)

```

Here we have extracted age of the vehicle on the basis of auto year by assuming the data is collected in the year 2018.

```

1 # Let's check the unique values again
2 df.nunique().to_frame("No of Unique Values")

```

Now after all the data cleaning until now, the dataset looks like below

```

# Let's take a look at the dataframe after preprocessing
df.head()

```

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	insured_sex	insured_education_level	insured_occupation	insured_hobbie
0	328	48	OH	1000	1406.91	MALE	MD	craft-repair	sleepin
1	228	42	IN	2000	1197.22	MALE	MD	machine-op-inspct	readin
2	134	29	OH	2000	1413.14	FEMALE	PhD	sales	board-game
3	256	41	IL	2000	1415.74	FEMALE	PhD	armed-forces	board-game
4	228	44	IL	1000	1583.91	MALE	Associate	sales	board-game

Statistical Summary of Dataset

```

# Statistical summary of numerical columns
df.describe(include="all").T

```

Here we use "describe" method along with its parameter "all" to include each column present in our dataset irrespective of them being numeric or text data. We have used transpose method to see the column information properly without scrolling multiple times.

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	NaN	NaN	NaN	203.954	115.113174	0.0	115.75	199.5	276.25	479.0
age	1000.0	NaN	NaN	NaN	38.948	9.140287	19.0	32.0	38.0	44.0	64.0
policy_state	1000	3	OH	352	NaN	NaN	NaN	NaN	NaN	NaN	NaN
policy_deductable	1000.0	NaN	NaN	NaN	1136.0	611.864673	500.0	500.0	1000.0	2000.0	2000.0
policy_annual_premium	1000.0	NaN	NaN	NaN	1256.40615	244.167395	433.33	1089.6075	1257.2	1415.695	2047.59
insured_sex	1000	2	FEMALE	537	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_education_level	1000	7	JD	161	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_occupation	1000	14	machine-op-inspct	93	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_hobbies	1000	20	reading	64	NaN	NaN	NaN	NaN	NaN	NaN	NaN
insured_relationship	1000	6	own-child	183	NaN	NaN	NaN	NaN	NaN	NaN	NaN
capital-gains	1000.0	NaN	NaN	NaN	25126.1	27872.187708	0.0	0.0	0.0	51025.0	100500.0
capital-loss	1000.0	NaN	NaN	NaN	-26793.7	28104.096686	-111100.0	-51500.0	-23250.0	0.0	0.0
incident_type	1000	4	Multi-vehicle Collision	419	NaN	NaN	NaN	NaN	NaN	NaN	NaN
collision_type	1000	3	Rear Collision	470	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_severity	1000	4	Minor Damage	354	NaN	NaN	NaN	NaN	NaN	NaN	NaN
authorities_contacted	1000	5	Police	292	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_state	1000	7	NY	262	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_city	1000	7	Springfield	157	NaN	NaN	NaN	NaN	NaN	NaN	NaN
incident_hour_of_the_day	1000.0	NaN	NaN	NaN	11.644	6.951373	0.0	6.0	12.0	17.0	23.0
number_of_vehicles_involved	1000.0	NaN	NaN	NaN	1.839	1.01888	1.0	1.0	1.0	3.0	4.0
property_damage	1000.0	2	NO	698	NaN	NaN	NaN	NaN	NaN	NaN	NaN

From the above dataset, we can infer below details:

- The counts of all the columns are equal => no missing values in the dataset.
- In some columns like policy_deductable, capital-gains, injury_claim etc, mean value is greater than the median(50%), => data in these columns are skewed to right.
- Columns like total_claim_amount, vehicle_claim etc, the mean is less than the median => data in these columns are skewed to left.
- Columns like policy_annual_premium have equal mean and median that means the data is symmetric and is normally distributed and no skewness present.

Data Visualization

Preparing for Visualization

We will look into the categorical and numerical columns so that we can create two different lists and visualize the features accordingly.

```
# Separating numerical and categorical columns

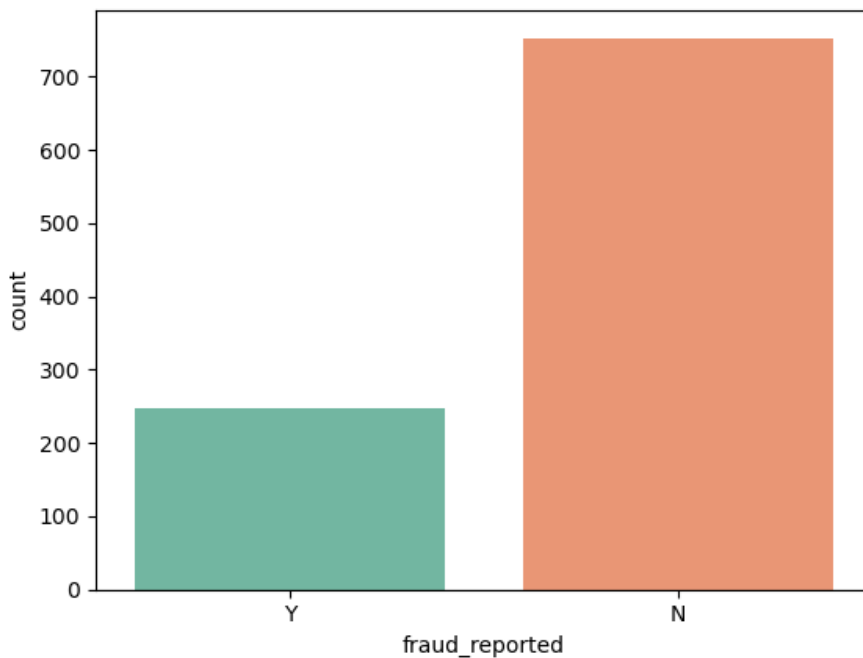
# Checking for categorical columns
categorical_col=[]
for i in df.dtypes.index:
    if df.dtypes[i]=='object':
        categorical_col.append(i)
print("Categorical columns are:\n",categorical_col)
```

```
numerical_col=[]
for i in df.dtypes.index:
    if df.dtypes[i]!='object':
        numerical_col.append(i)
print("Numerical columns are:\n",numerical_col)
```

Univariate Analysis (Categorical Columns)

```
1 #Visualize how many insurance claims are fraudulent
2 print(df["fraud_reported"].value_counts())
3 sns.countplot(df["fraud_reported"], x= df["fraud_reported"], palette="Set2")
4 plt.show()
```

```
fraud_reported
N    753
Y    247
Name: count, dtype: int64
```



From the plot we can observe that the count of "N" is high compared to "Y".

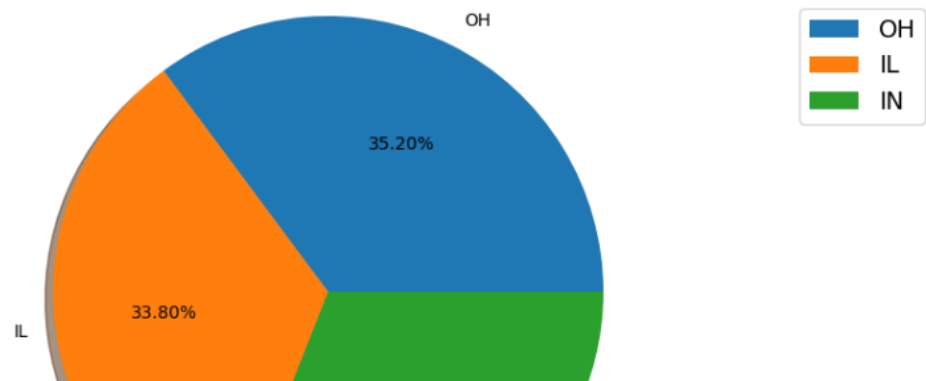
We will balance the data using the oversampling method in later part.

```
def generate_pie(i):
    plt.figure(figsize=(10,5))
    plt.pie(i.value_counts(), labels=i.value_counts().index, autopct='%1.2f%%', shadow=True,)
    plt.legend(prop={'size':14})
    plt.axis('equal')
    plt.tight_layout()
    return plt.show()
```

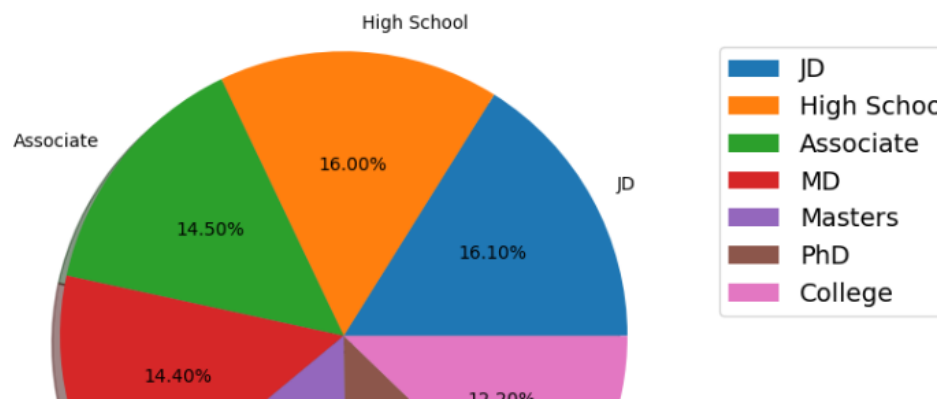
```
cols1 = ['policy_state', 'insured_sex', 'insured_education_level', 'insured_relationship', 'incident_type', 'collision_type', 'proprietary']
```

```
plotnumber=1
for j in df[cols1]:
    print(f"Pie plot for the column:", j)
    generate_pie(df[j])
```

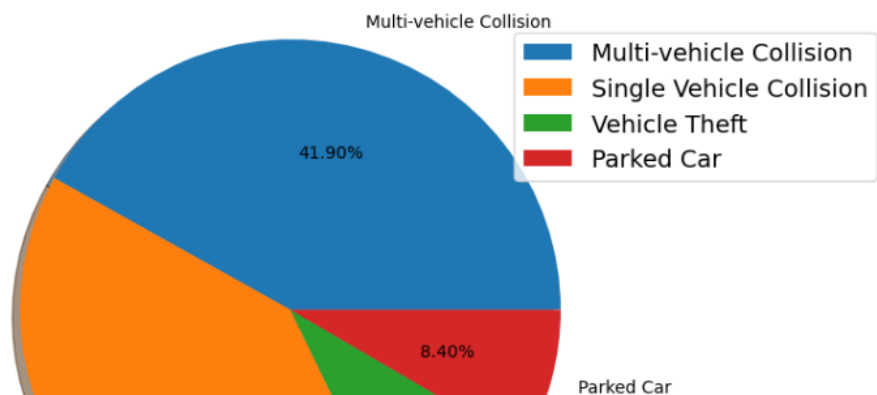
Pie plot for the column: policy_state



Pie plot for the column: insured_education_level



Pie plot for the column: incident_type



Above are the pie plots for some of the categorical columns and below are few observations

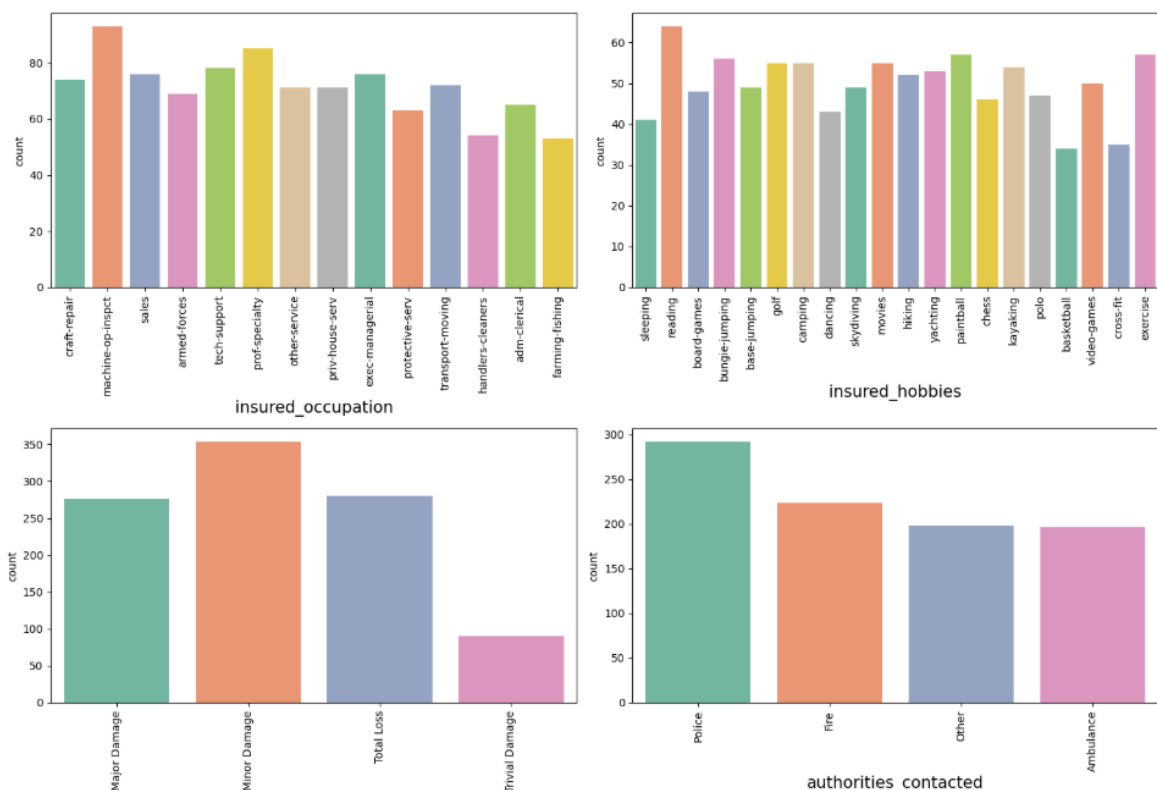
- The types of the policies claimed by the customers are almost same but still the policy state type OH has bit high counts and the type IL has bit less count.
- Both male and female have insurance but the count for Female is higher than Male counts.
- The count is pretty much same for all the education level but still the people who have completed their college and PhD have less count compared to others.
- Similar to insured education, insured relationship is also almost equally distributed.

- Under the incident type, Multi-vehicle collision and Single Vehicle Collision have almost similar counts of around 41.9% and 40.3%. But the count is very less in Parked car and Vehicle Theft.
- The collision type has 3 different types . In this the count is high in Rear collision and the other two types have almost equal counts. As we observe the propert damage plot, around 69.8% of the people did not face any property damage while 30.2% of the people faced the property damage. About 68.6% of the people produced the police reports while 31.4% of the people didn't submit any police reports.

```

1 cols2 = ['insured_occupation', 'insured_hobbies', 'incident_severity', 'authorities_contacted', 'incident_state',
2
3 plt.figure(figsize=(15,25),facecolor='white')
4 plotnumber=1
5 for column in cols2:
6     if plotnumber:
7         ax=plt.subplot(5,2,plotnumber)
8         sns.countplot(x= df[column],palette="Set2")
9         plt.xticks(rotation=90)
10        plt.xlabel(column,fontsize=15)
11    plotnumber+=1
12 plt.tight_layout()
13 plt.show()

```

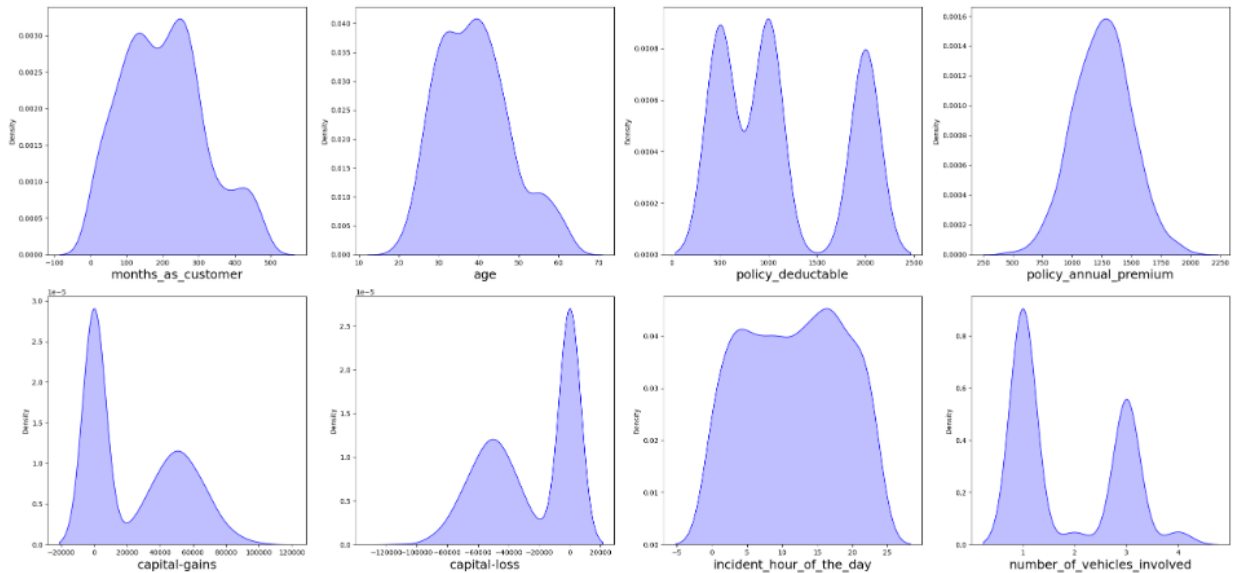


- Above are the count plots for the remaining categorical columns and below are the observations
- In the insured occupation category, the majority of the data is represented by machine operation inspectors, followed by professional specialties. The other insured occupations have nearly equal counts.
- Regarding insured hobbies, reading has the highest representation, followed by exercise. The other categories have average counts. Incident severity shows a high count for minor damages, while trivial damage has a significantly lower count compared to others.
- When accidents occur, authorities most frequently contact the police, with this category having the highest count. Fire has the second highest count, while Ambulance and Others have nearly equal counts. The None category has a much lower count compared to all others.
- For the incident state, New York, South Carolina, and West Virginia have the highest counts. In terms of

incident city, almost all columns have equal counts

Checking the Distribution of the dataset (numerical columns)

```
1 # Check how the data is distributed in each column
2 plt.figure(figsize=(25,35),facecolor='white')
3 plotnumber=1
4 for column in numerical_col:
5     if plotnumber<=23:
6         ax=plt.subplot(6,4,plotnumber)
7         sns.distplot(df[column],color="blue",hist=False,kde_kws={"shade": True})
8         plt.xlabel(column,fontsize=18)
9         plotnumber+=1
10 plt.tight_layout()
```



- Above are the distribution plots for all the numerical columns. From these plots, we can observe the following:
- Most of the columns exhibit a normal distribution. However, some columns, such as capital gains and incident months, have a mean value greater than the median, indicating a left skew.
- Conversely, the capital loss column is right-skewed, as the median is greater than the mean. We will address and correct these skews using appropriate methods later on

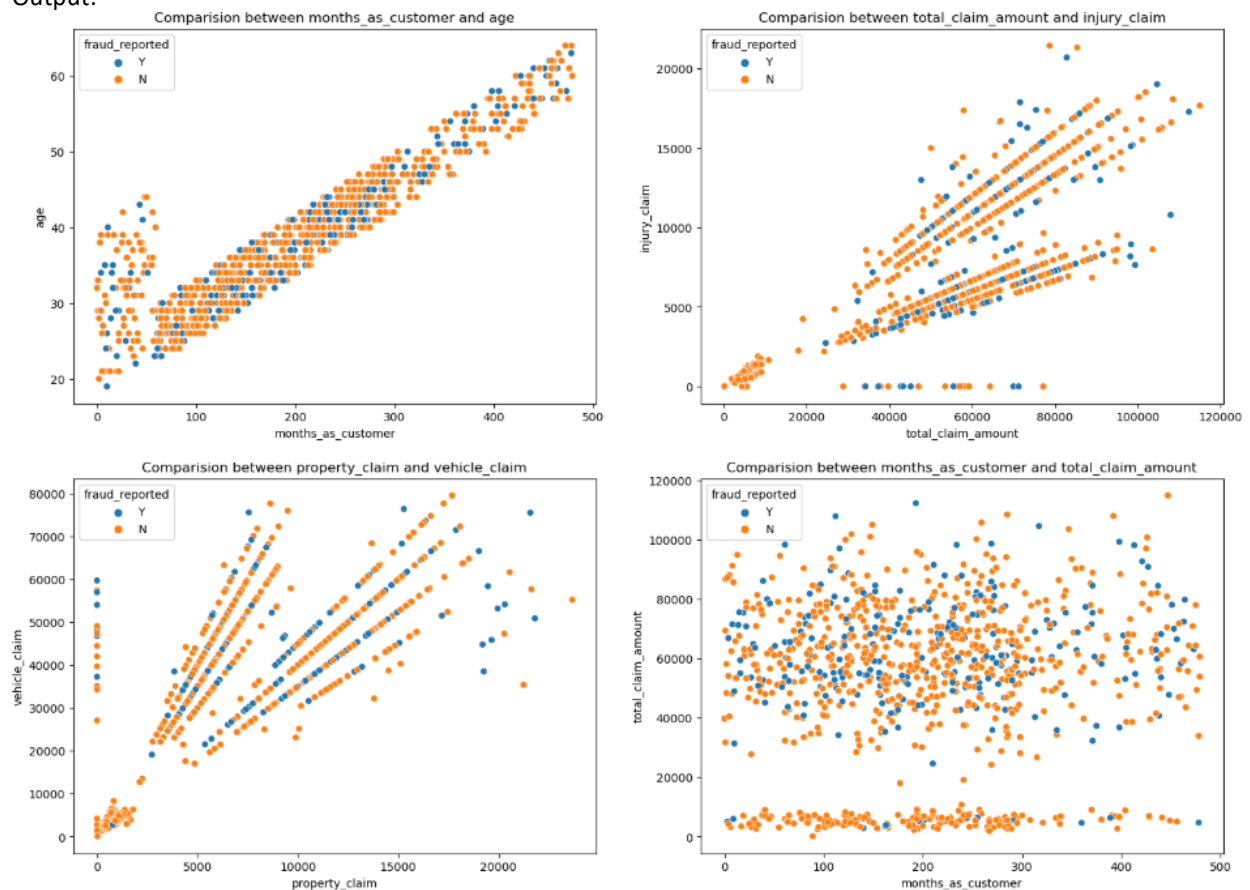
Bivariate Analysis

```

1 # Comparison between two variables
2 plt.figure(figsize=[18,13])
3
4 plt.subplot(2,2,1)
5 plt.title('Comparison between months_as_customer and age')
6 sns.scatterplot(x= df['months_as_customer'],y= df['age'],hue=df['fraud_reported']);
7
8 plt.subplot(2,2,2)
9 plt.title('Comparison between total_claim_amount and injury_claim')
10 sns.scatterplot(x= df['total_claim_amount'],y= df['injury_claim'],hue=df['fraud_reported']);
11
12 plt.subplot(2,2,3)
13 plt.title('Comparison between property_claim and vehicle_claim')
14 sns.scatterplot(x= df['property_claim'],y= df['vehicle_claim'],hue=df['fraud_reported']);
15
16 plt.subplot(2,2,4)
17 plt.title('Comparison between months_as_customer and total_claim_amount')
18 sns.scatterplot(x= df['months_as_customer'],y= df['total_claim_amount'],hue=df['fraud_reported']);

```

Output:



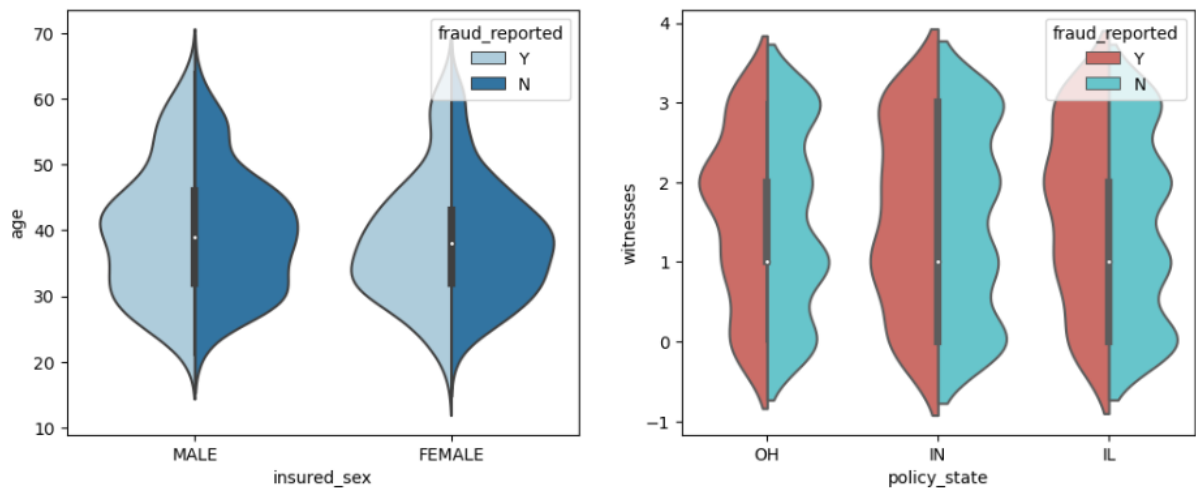
- From the above scatter plot, we can observe the following:
- There is a positive linear relationship between the age and month_as_customer columns.
- As age increases, the month_as_customer value also increases, with very few fraud cases reported in this scenario.
- In the second graph, we see a positive linear relationship where, as the total claim amount increases, the injury claim amount also increases.

- The third plot shows a similar trend: as the property claim amount increases, the vehicle claim amount also increases.
- In the fourth plot, the data is scattered, indicating no significant relationship between the features

```

1 fig, axes = plt.subplots(2, 2, figsize=(12, 10))
2
3 # Comparing insured_sex and age
4 sns.violinplot(x='insured_sex', y='age', ax=axes[0, 0], data=df, palette="Paired", hue="fraud_reported", split=True)
5
6 # Comparing policy_state and witnesses
7 sns.violinplot(x='policy_state', y='witnesses', ax=axes[0, 1], data=df, hue="fraud_reported", split=True, palette="hls")
8
9 # Comparing csl_per_accident and property_claim
10 sns.violinplot(x='csl_per_accident', y='property_claim', ax=axes[1, 0], data=df, hue="fraud_reported", split=True, palette="Set")
11
12 # Comparing csl_per_person and age
13 sns.violinplot(x='csl_per_person', y='age', ax=axes[1, 1], data=df, hue="fraud_reported", split=True, palette="hus1")
14 plt.show()

```

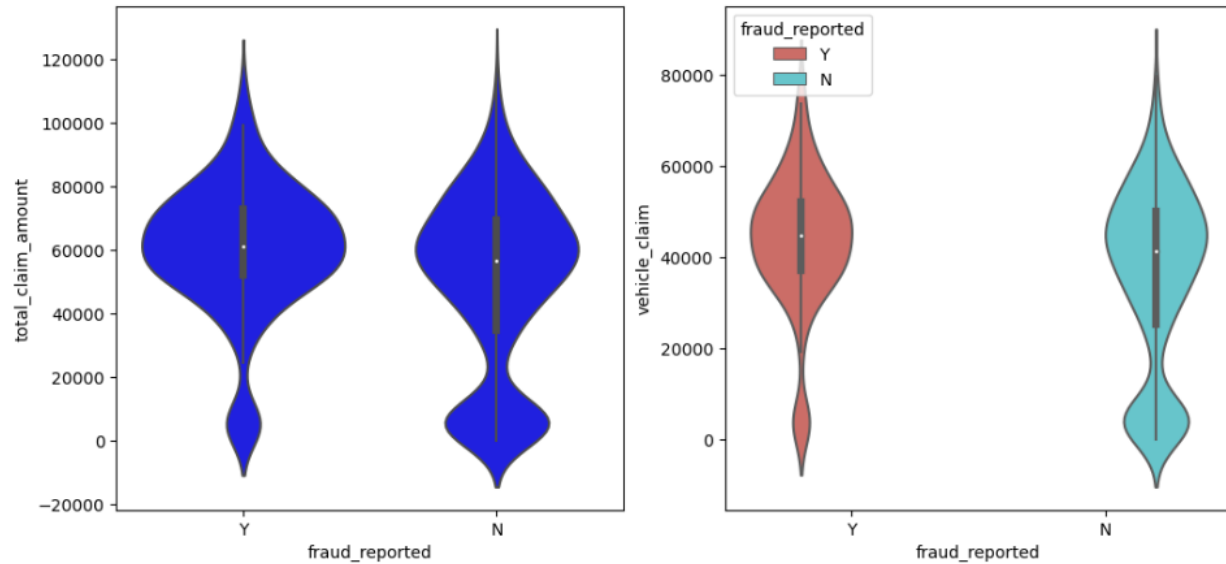


Fraud reports are high for both males and females aged between 30-45. Individuals who own policies in the state of "IN" have a high incidence of fraud reports. Those with CSL per accident insurance and property claims ranging from 5,000 to 15,000 also report high fraud. Additionally, individuals aged 30-45 with CSL per person insurance are facing fraudulent reports

```

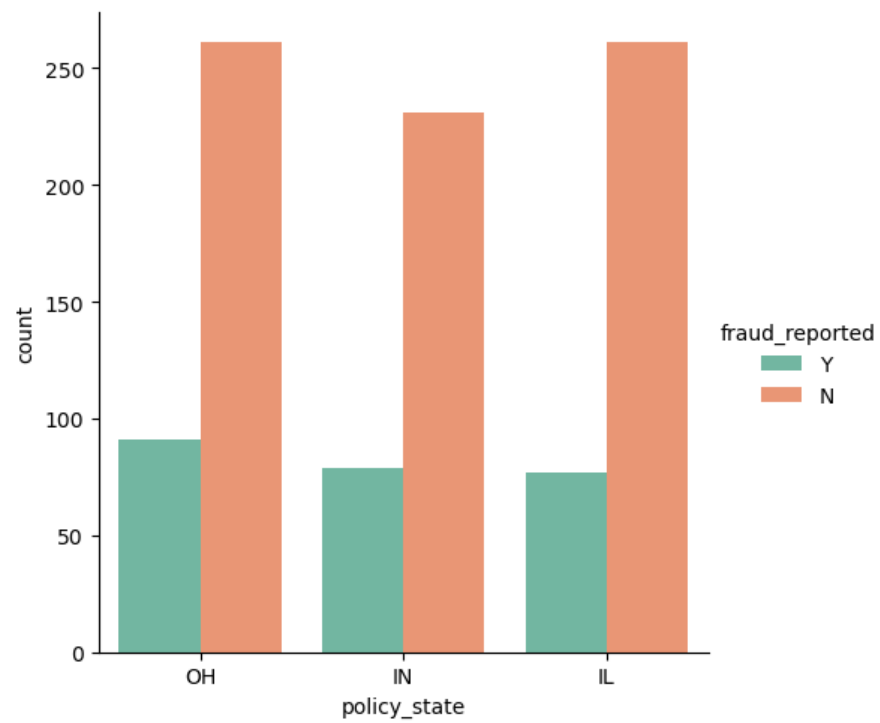
1 fig, axes = plt.subplots(2, 2, figsize=(12, 12))
2
3 # Comparing insured_sex and age
4 sns.violinplot(x='fraud_reported', y='total_claim_amount', ax=axes[0, 0], data=df, color="b")
5
6 # Comparing policy_state and witnesses
7 sns.violinplot(x='fraud_reported', y='vehicle_claim', ax=axes[0, 1], data=df, hue="fraud_reported", palette="hls")
8
9 # Comparing csl_per_accident and property_claim
10 sns.violinplot(x='fraud_reported', y='property_claim', ax=axes[1, 0], data=df, hue="fraud_reported", palette="hus1")
11
12 # Comparing csl_per_person and age
13 sns.violinplot(x='fraud_reported', y='injury_claim', ax=axes[1, 1], data=df, hue="fraud_reported", palette="Paired")
14 plt.show()

```

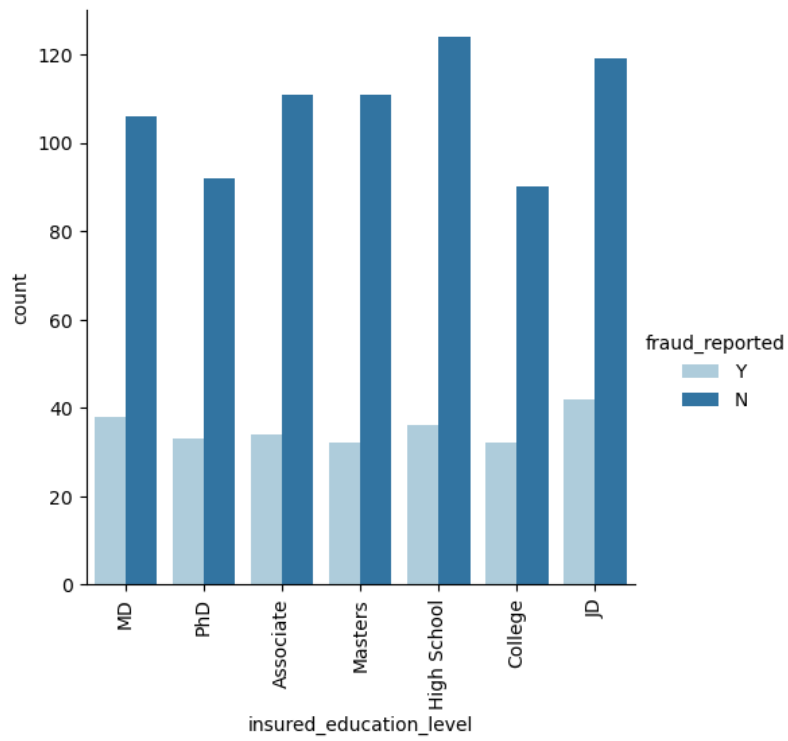


Most fraud reports are found when the total claimed amount is between 50,000 and 70,000. Fraud reports are also high when the claimed vehicle value is between 37,000 and 57,000. Additionally, fraud reports are frequent when the property claimed is between 5,200 and 8,500. Most fraud is reported when injury claims are between 5,000 and 8,000

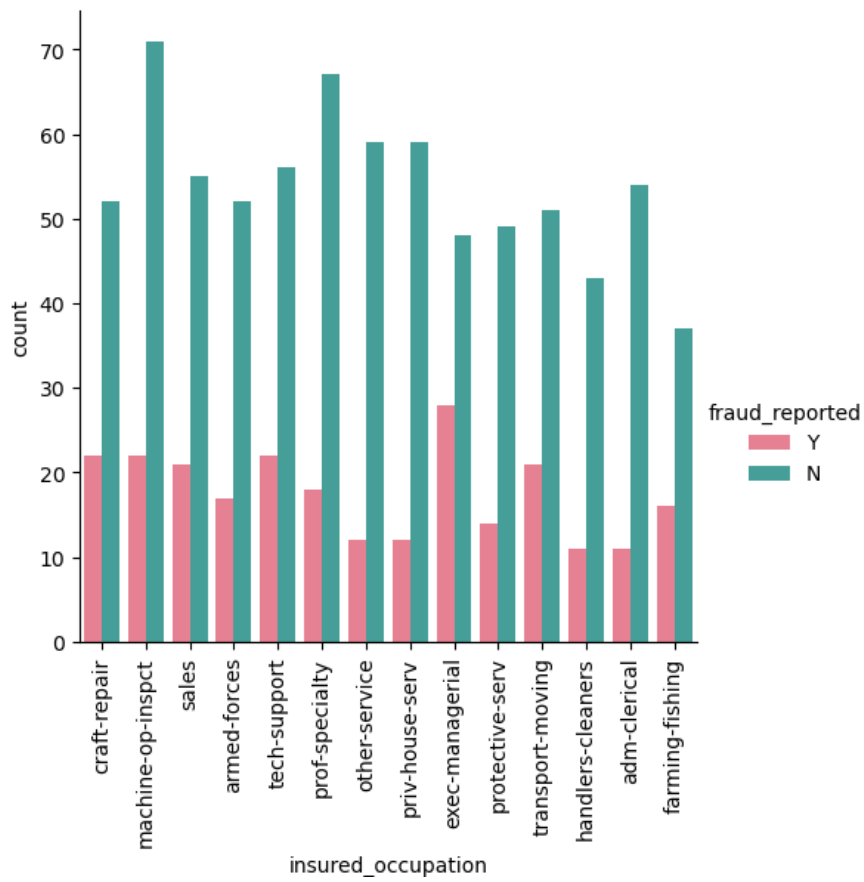
```
1 # Comparing policy_state and fraud_reported
2 sns.catplot(x = 'policy_state', kind='count', data = df, hue='fraud_reported', palette="Set2")
3 plt.show()
```



```
1 # Comparing insured_education_level and fraud_reported
2
3 sns.catplot(x='insured_education_level',kind='count',data=df,hue='fraud_reported',palette="Paired")
4 plt.xticks(rotation=90)
5 plt.show()
```



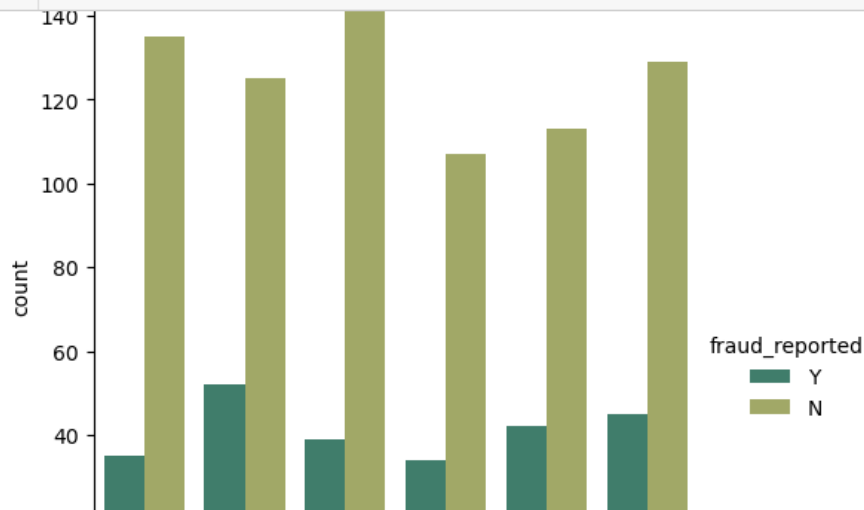
```
1 # Comparing insured_occupation and fraud_reported
2 sns.catplot(x= 'insured_occupation',kind='count',data=df,hue='fraud_reported',palette="husl")
3 plt.xticks(rotation=90)
4 plt.show()
```



```

1 # Comparing insured_relationship and fraud_reported
2 sns.catplot(x='insured_relationship',kind='count',data=df,hue='fraud_reported',palette="gist_earth")
3 plt.xticks(rotation=90)
4 plt.show()

```



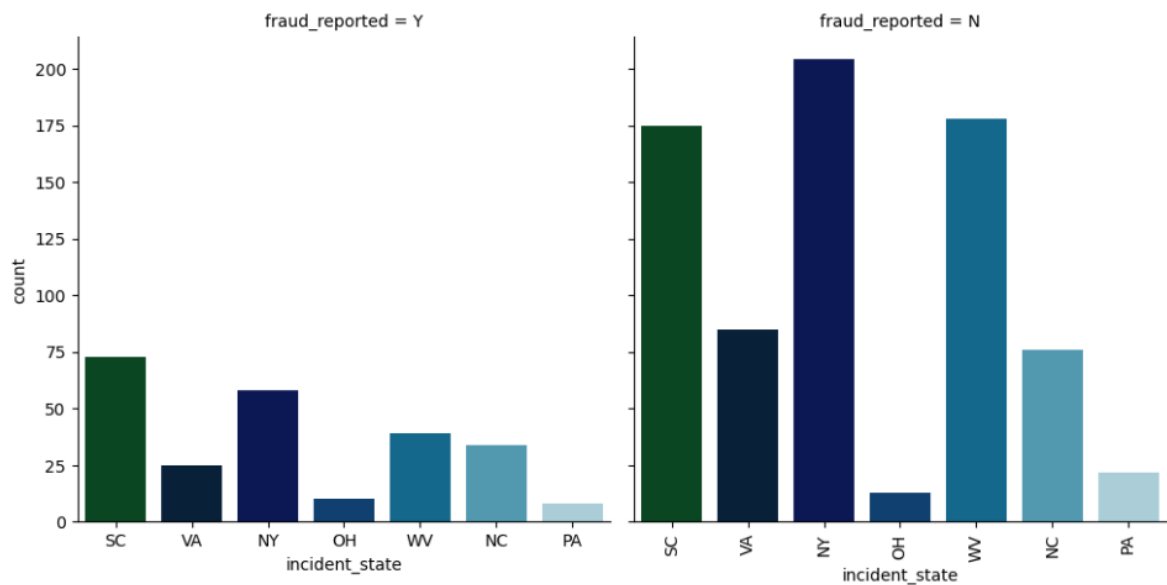
The fraud report is high for the customers who have other relative and it is very less for unmarried people.

The police contacted cases are very high and the fraud report is in equal for all the authorities except None.

```

8]: 1 # Comparing incident_state and fraud_reported
    2 sns.catplot(x='incident_state',kind='count',data=df,col='fraud_reported',palette="ocean")
    3 plt.xticks(rotation=90)
    4 plt.show()

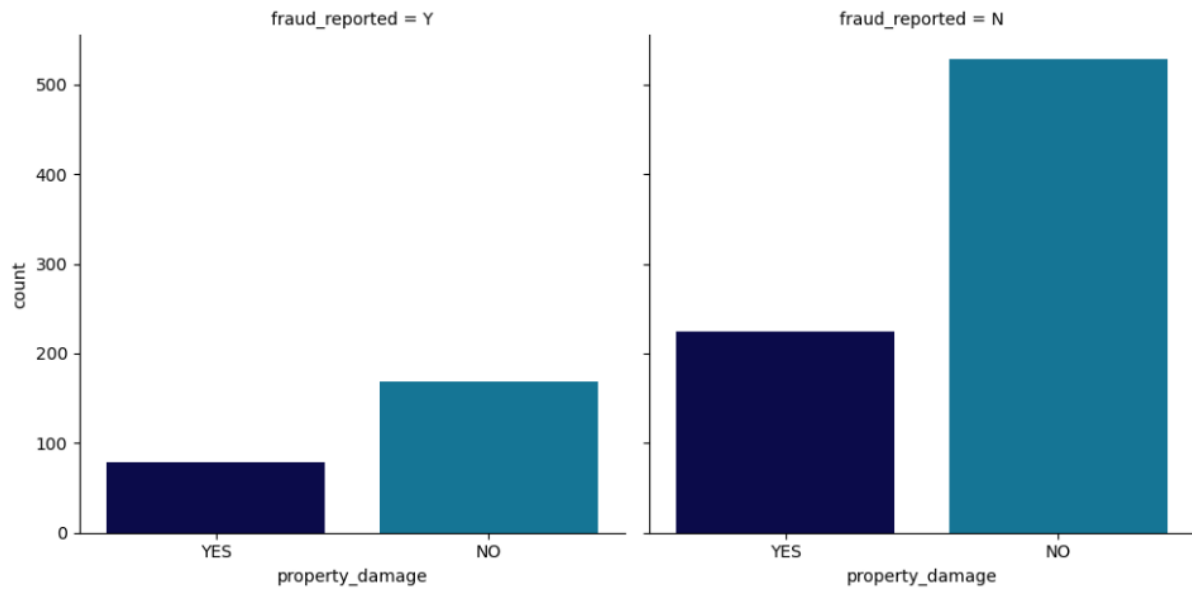
```



The state SC has high fraud reports compared to other states.

Cities Riverwood and Northbrook have very less fraud reports compared to others.

```
1 # Comparing property_damage and fraud_reported
2 sns.catplot(x='property_damage',kind='count',data=df,col='fraud_reported',palette="ocean")
3 plt.show()
```

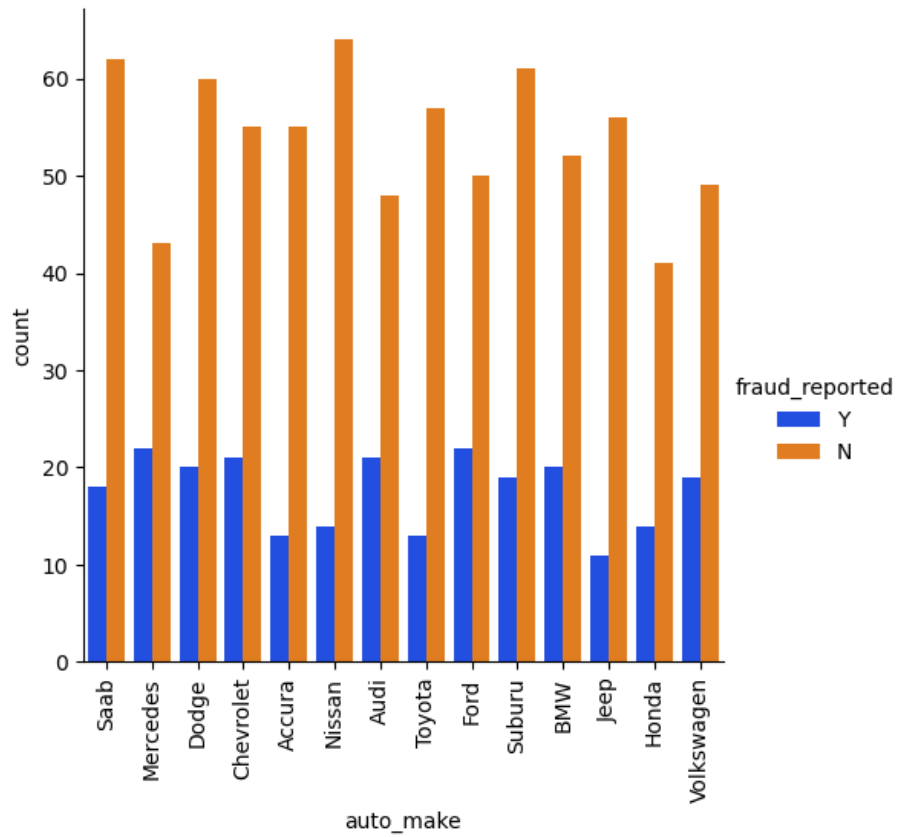


Customers who do not have any property damage case they have high fraud reports.

```

1 # Compare auto_make and fraud_reported
2 sns.catplot(x='auto_make',kind='count',data=df,hue='fraud_reported',palette="bright")
3 plt.xticks(rotation=90)
4 plt.show()

```

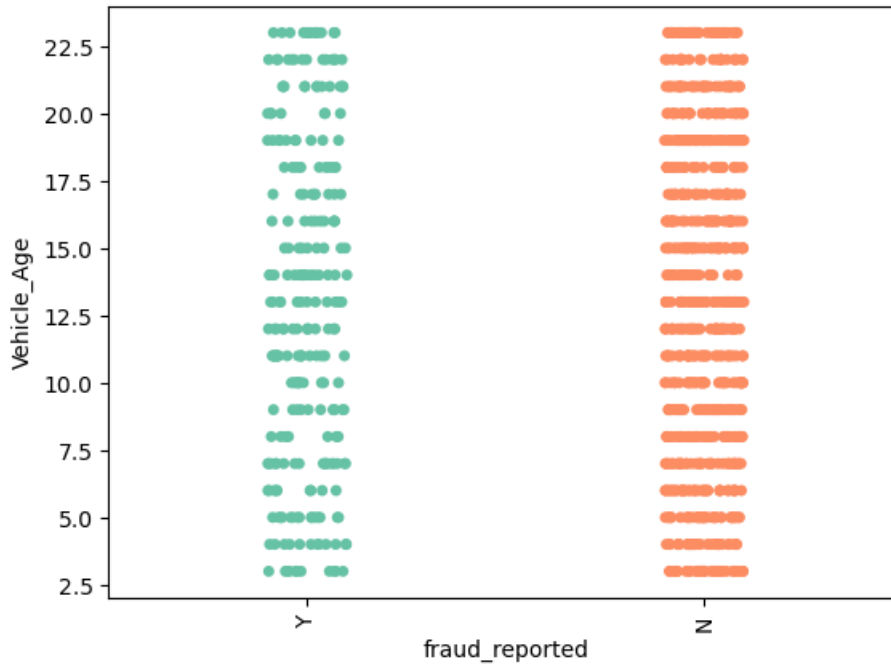


all the auto make cases the fraud report is almost same.


```

1 # Compare Vehicle_Age and fraud_reported
2 sns.stripplot(y='Vehicle_Age',x='fraud_reported',data=df,palette="Set2")
3 plt.xticks(rotation=90)
4 plt.show()

```



re is no significant difference between the features.

```

sns.pairplot(df,hue="fraud_reported",palette="husl")
plt.show()

```



In the pair plot we can see the relation between each variable with respect to other variables.

Observations

- Fraud report is relatively high in the "OH" policy state.
- Individuals in executive-managerial positions have higher fraud reports compared to others.
- Fraud reports are high for customers with other relatives and very low for unmarried individuals
- Fraud reports are high among people whose hobbies include playing chess and cross fit.
- The fraudulent level is very low for individuals with high school education, whereas those with "JD" education have a high fraud report. People with higher insured education levels face more insurance fraud compared to those with lower education levels.
- Fraud reports are very high in multi-vehicle and single-vehicle collisions compared to other types.
- The fraud level is high in rear collisions, while the other two collision types have average reports.
- Fraud reports are high in incidents with major damage severity, whereas trivial damage incidents have lower fraud reports.
- If no police report is available, the fraud report is very high.
- Fraud reports are almost the same across all auto makes.
- There is no significant difference between the features in Vehicle_Age vs fraud_reported.
- Fraud reports are high for customers with other relatives and very low for unmarried individuals

Now we will Data Pre-processing by identifying the outliers and remove them. We will check the skewness of the dataset and remove the skewness.

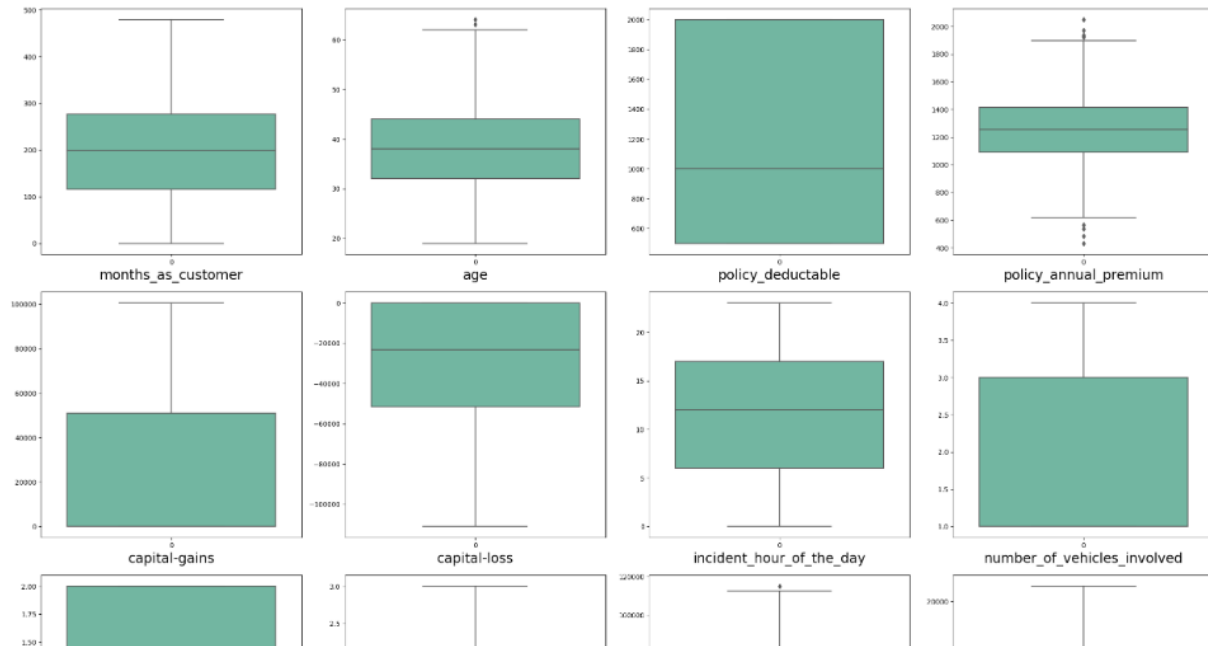
Pre-Processing Pipeline

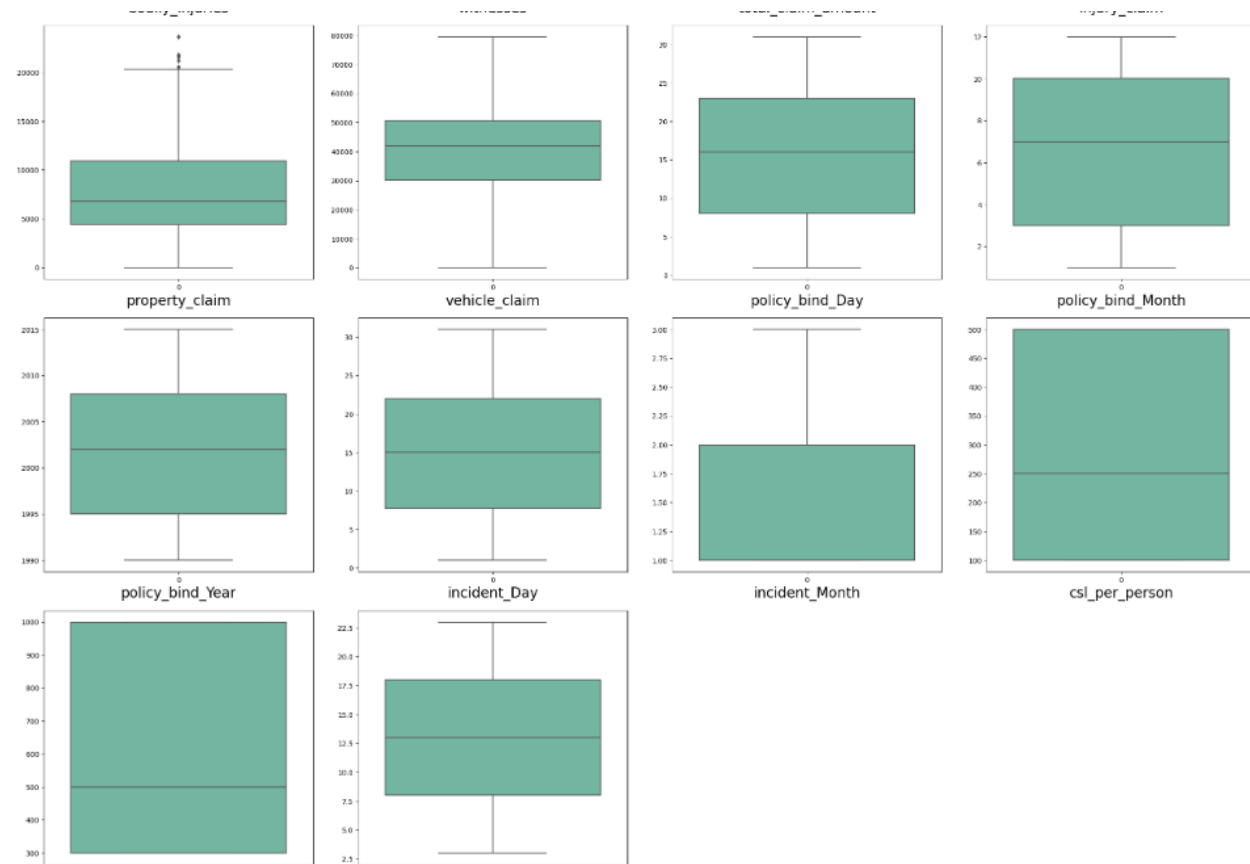
Identifying the outliers

```

1 # Check the outliers by plotting box plot
2
3 plt.figure(figsize=(25,35),facecolor='white')
4 plotnumber=1
5 for column in numerical_col:
6     if plotnumber<=23:
7         ax=plt.subplot(6,4,plotnumber)
8         sns.boxplot(df[column],palette="Set2")
9         plt.xlabel(column,fontsize=20)
10        plotnumber+=1
11 plt.tight_layout()

```





We can find the outliers in the following columns:

age policy_annual_premium , total_claim_amount , property_claim and incident_month.

These are the numerical columns which contain outliers. Remove the outliers in these columns using Zscore method.

```
# Feature containing outliers
features = df[['age', 'policy_annual_premium', 'total_claim_amount', 'property_claim', 'incident_Month']]

z=np.abs(zscore(features))

z
```

```
# Creating new dataframe
new_df = df[(z<3).all(axis=1)]
new_df
```

This is the new dataframe after removing the outliers. Here we have removed the outliers whose Zscore is less than 3.

Percentage data loss:

```
loss_percent=(1000-996)/1000*100
print(loss_percent,'%')
```

0.4 %

After removing the outliers, we are checking the data loss percentage by comparing the rows in our original data set and the new data set and 0.4% data loss is in the acceptable range.

Check skewness in the data:

```
# Checking the skewness
new_df.skew().sort_values()

vehicle_claim          -0.619755
total_claim_amount     -0.593473
capital-loss           -0.393015
incident_hour_of_the_day -0.039123
policy_bind_Month      -0.029722
bodily_injuries         0.011117
witnesses              0.025758
policy_bind_Day         0.028923
policy_annual_premium   0.032042
Vehicle_Age            0.049276
incident_Day           0.055659
policy_bind_Year        0.058499
injury_claim           0.267970
property_claim          0.357130
months_as_customer      0.359605
csl_per_person          0.413713
policy_deductable       0.473229
age                    0.474526
capital-gains           0.478850
number_of_vehicles_involved 0.500364
csl_per_accident        0.609316
incident_Month          1.377097
dtype: float64
```

As we can see that skewness is present in the dataset, we will use yeo-johnson method to remove the skewness.

```
# Removing skewness using yeo-johnson method to get better prediction

skew = ["total_claim_amount", "vehicle_claim", "incident_Month", "csl_per_accident"]

transf = PowerTransformer(method='yeo-johnson')
new_df[skew] = transf.fit_transform(new_df[skew].values)
new_df[skew].head()
```

	total_claim_amount	vehicle_claim	incident_Month	csl_per_accident
0	0.717556	0.754553	-1.101370	0.052612
1	-1.777785	-1.787353	-1.101370	0.052612
2	-0.716483	-0.820820	-0.026479	-1.174021
3	0.392931	0.678427	1.553647	0.052612
4	-1.730555	-1.740710	-0.026479	1.313327

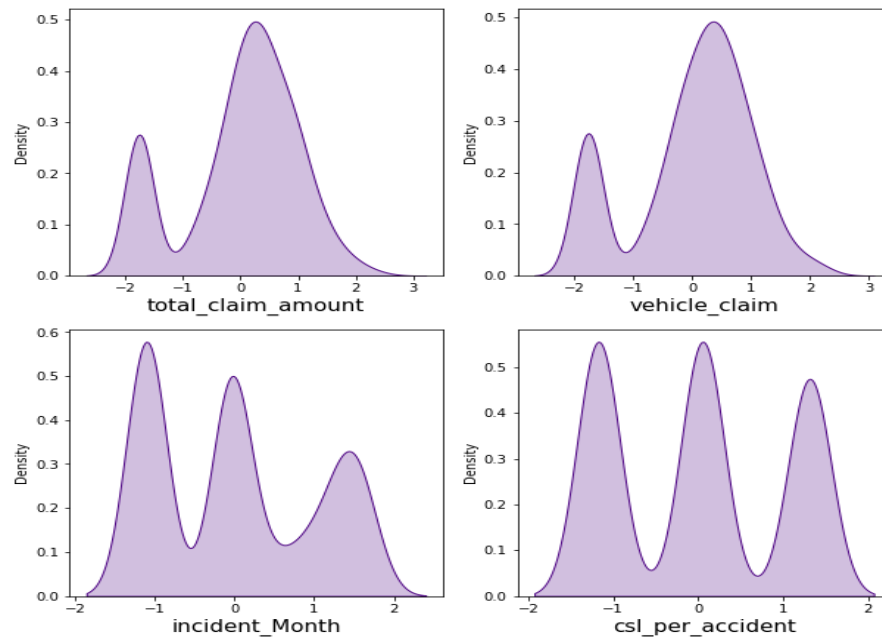
```
# Checking skewness after using yeo-johnson method
new_df[skew].skew().sort_values()
```

```
vehicle_claim          -0.521354
total_claim_amount     -0.508953
csl_per_accident        0.110964
incident_Month          0.305741
dtype: float64
```

After removing skewness Let's check how the data has been distributed in each column.

```
plt.figure(figsize=(10,10), facecolor='white')
plotnumber = 1

for column in new_df[skew]:
    if plotnumber<=4:
        ax = plt.subplot(2,2,plotnumber)
        sns.distplot(new_df[column],color='indigo',kde_kws={"shade": True},hist=False)
        plt.xlabel(column,fontsize=15)
        plotnumber+=1
plt.show()
```



Encoding the categorical columns using Label Encoding

```
LE=LabelEncoder()
new_df[categorical_col]= new_df[categorical_col].apply(LE.fit_transform)
new_df[categorical_col].head()
```

	policy_state	insured_sex	insured_education_level	insured_occupation	insured_hobbies	insured_relationship	incident_type	collision_type	incident_severity
0	2	1	4	2	17	0	2	2	0
1	1	1	4	6	15	2	3	1	1
2	2	0	6	11	2	3	0	1	1
3	0	0	6	1	2	4	2	0	0
4	0	1	0	11	2	4	3	1	1

Now we have encoded the dataset using label encoder and the dataset looks like above.

Moving forward, to check the correlation between the feature and target and also the relation between the features using the heatmap.

```
# Checking the correlation between features and the target
cor = new_df.corr()
cor
```

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	insured_sex	insured_education_level	ins
months_as_customer	1.000000	0.922092	0.000118	0.023512	-0.003378	0.059002	-0.000848	
age	0.922092	1.000000	-0.015182	0.026772	0.005890	0.072900	0.001485	
policy_state	0.000118	-0.015182	1.000000	0.010740	0.014475	-0.019294	-0.032551	
policy_deductable	0.023512	0.026772	0.010740	1.000000	-0.008445	-0.011671	0.014571	
policy_annual_premium	-0.003378	0.005890	0.014475	-0.008445	1.000000	0.041830	-0.021475	
insured_sex	0.059002	0.072900	-0.019294	-0.011671	0.041830	1.000000	0.007473	
insured_education_level	-0.000848	0.001485	-0.032551	0.014571	-0.021475	0.007473	1.000000	
insured_occupation	0.005365	0.014030	-0.031016	-0.050024	0.030963	-0.009257	-0.021502	

```
# Visualizing the correlation matrix by plotting heat map.
```

```
plt.style.use('seaborn-pastel')
```

```
upper_triangle = np.triu(df.corr())
```

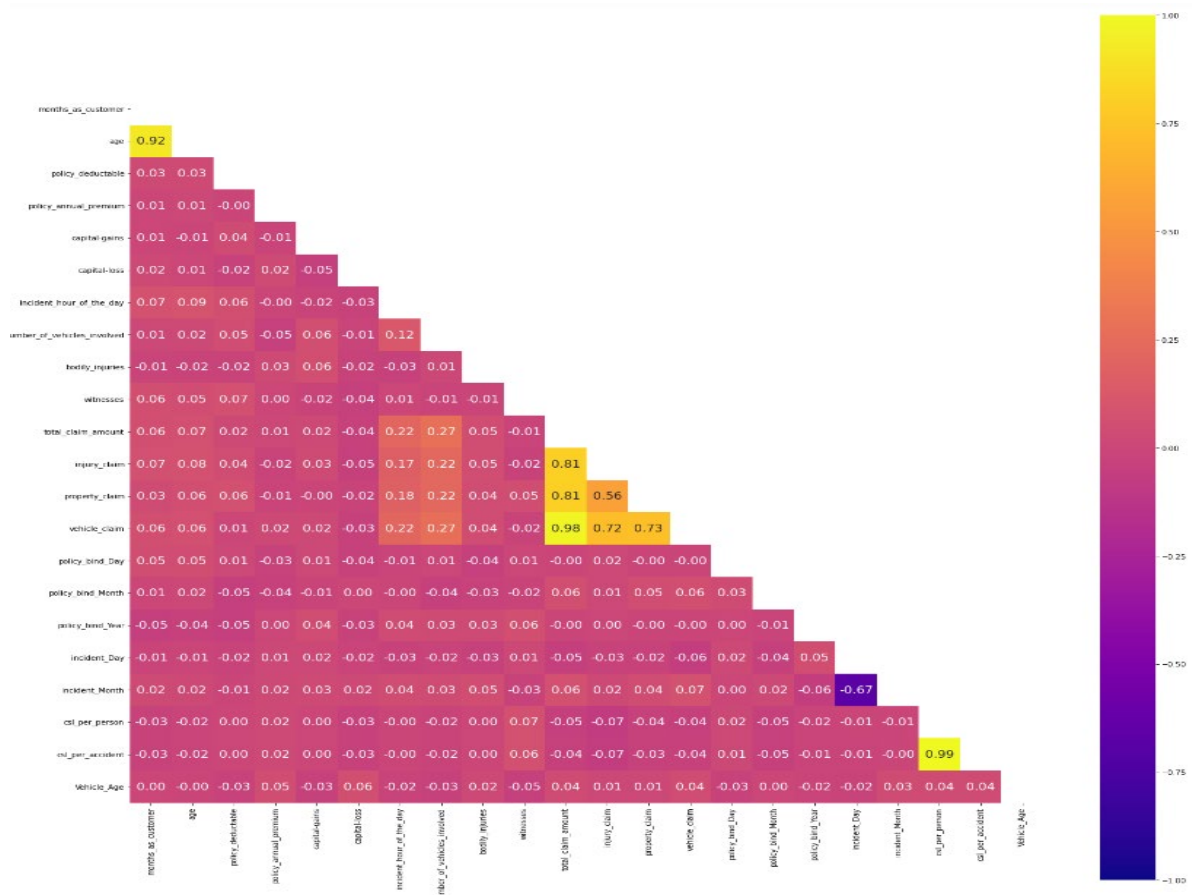
```
plt.figure(figsize=(25,25))
```

```
sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, square=True, fmt='0.2f',
             annot_kws={'size':16}, cmap="plasma", mask=upper_triangle)
```

```
plt.xticks(fontsize=10)
```

```
plt.yticks(fontsize=10)
```

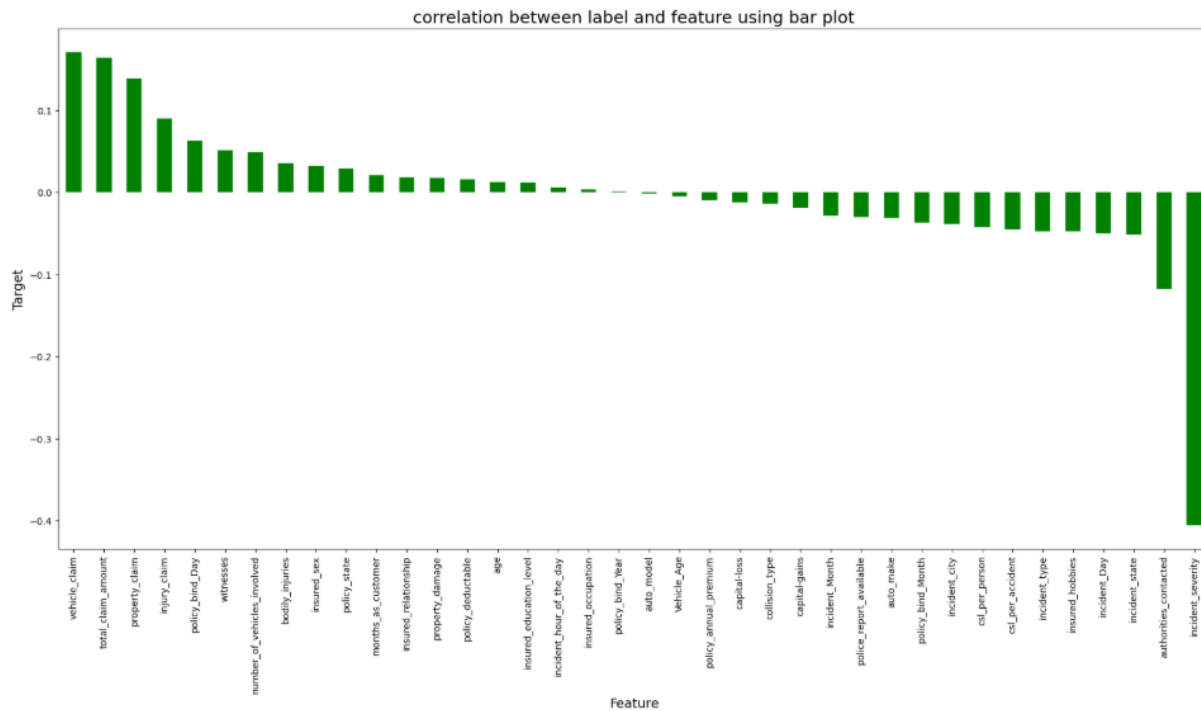
```
plt.show()
```



```

1 plt.figure(figsize=(22,10))
2 new_df.corr()[['fraud_reported']].sort_values(ascending=False).drop(['fraud_reported']).plot(kind='bar',color='g')
3 plt.xlabel('Feature',fontsize=14)
4 plt.ylabel('Target',fontsize=14)
5 plt.title('correlation between label and feature using bar plot',fontsize=18)
6 plt.show()

```



This heatmap shows the correlation matrix by visualizing the data. we can observe the relation between the features

The heatmap contains both positive and negative correlations.

There is very little correlation between the target and the label. We can observe that most of the columns are highly correlated with each other that leads to multicollinearity problems. We will check the VIF value to overcome this multicollinearity problem.

Splitting the dataset into Features and Target

```

x = new_df.drop("fraud_reported", axis=1)
y = new_df["fraud_reported"]

```

```

# Dimension of x
x.shape

```

```
(996, 38)
```

```

# Dimension of y
y.shape

```

```
(996,)
```


Feature Scaling using Standard Scaler

```
scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
x.head()
```

The data has now been scaled.

In the heat map we have found some features having high correlation between each other which means multicollinearity exists. So, let's check the VIF value to solve multicollinearity problem.

Checking Multicollinearity

```
# Finding variance inflation factor in each scaled column
def calc_vif(x):
    vif = pd.DataFrame()
    vif["Features"] = x.columns
    vif["VIF values"] = [variance_inflation_factor(x.values,i) for i in range(x.shape[1])]
    return(vif)
```

```
calc_vif(x)
```

	Features	VIF values
0	months_as_customer	6.851319
1	age	6.855557
2	policy_state	1.039410
3	policy_deductable	1.045656
4	policy_annual_premium	1.037450
5	insured_sex	1.037410
6	insured_education_level	1.047043

We can observe that some columns have VIF above 10 that mean they are causing multicollinearity problem. Let's drop the feature having high VIF value amongst all the columns.

```
# Dropping total_claim_amount column as it contains high VIF value
x.drop(["total_claim_amount"],axis=1,inplace=True)

# Dropping csl_per_accident column
x.drop(["csl_per_accident"],axis=1,inplace=True)
```

As there is a huge difference between the count of 0 & 1, the data is not balanced. As this is a classification problem we will balance using oversampling method.

```
y.value_counts()
```

```
0    750
1    246
Name: fraud_reported, dtype: int64
```

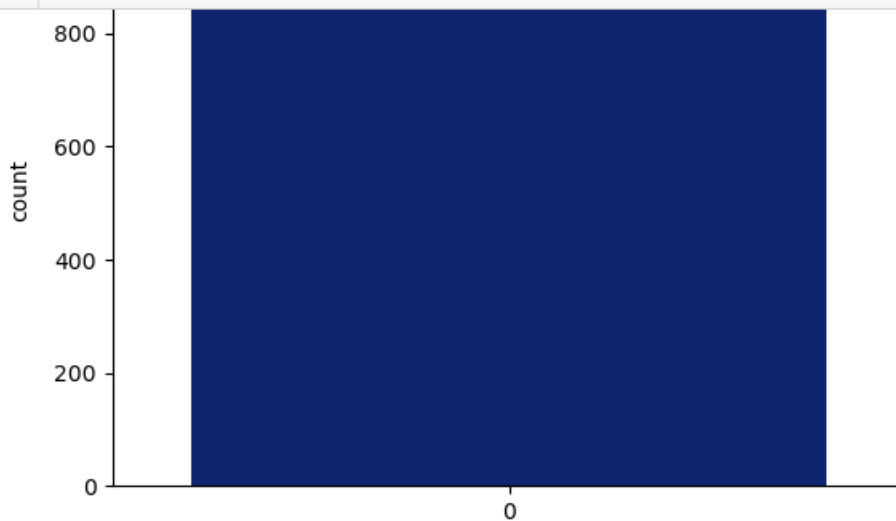
```
# Oversampling the data
from imblearn.over_sampling import SMOTE
SM = SMOTE()
x, y = SM.fit_resample(x,y)
```

```
# Checking value count of target column
y.value_counts()
```

```
1    750
0    750
Name: fraud_reported, dtype: int64
```

As we have treated the oversampling issue using SMOTE, now we can proceed with modelling.

```
1 # Visualizing the data after oversampling
2 sns.countplot(y,palette="dark")
```



Model Selection

Several machine learning models were evaluated for detecting fraudulent claims. The models considered include:

- Logistic Regression
- Naïve Bayes
- Support Vector Machines (SVM)
- Decision Tree Classifier
- K Neighbors Classifier
- SGD Classifier
- Random Forest Classifier
- Extra Tree Classifier

Each model was trained on the training dataset and evaluated using the testing dataset.

Machine Learning

Finding the best random state

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split
rf = RandomForestClassifier()
```

For Test size of 0.30

```
maxAccu=0
maxRS=0

for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.30, random_state =i)
    rf.fit(x_train, y_train)
    pred = rf.predict(x_test)
    acc=accuracy_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

Best accuracy is 0.9177777777777778 on Random_state 78

For Test size of 0.20

```
maxAccu=0
maxRS=0

for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.20, random_state =i)
    rf.fit(x_train, y_train)
    pred = rf.predict(x_test)
    acc=accuracy_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

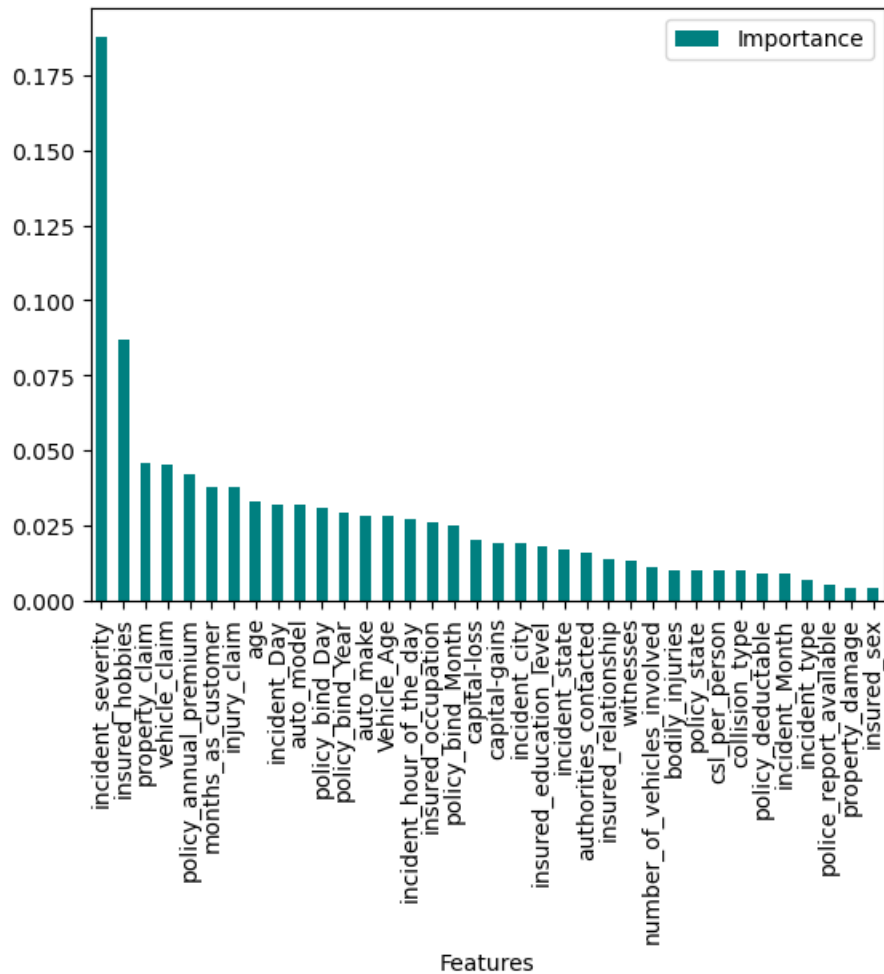
Best accuracy is 0.93 on Random_state 80

We have got the best random state as 80 for test size 0.2.

Here we have used the RandomForestClassifier to find the best random state and got an accuracy score of 93% at the random state of 80. Let's use this random state to build our models.

Feature importance bar graph

```
rf.fit(x_train, y_train)
importances = pd.DataFrame({'Features':x.columns, 'Importance':np.round(rf.feature_importances_,3)})
importances = importances.sort_values('Importance', ascending=False).set_index('Features')
importances.plot.bar(color='teal')
importances
```



This bar plot shows us the importance of the features using random forest algorithm on predicting our Target variable.

We will import all the required libraries as shown below.

```
# Importing required libraries
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier

from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Results

The results of the models were compared, and the best-performing model was selected based on the evaluation metrics. The extra tree classifier emerged as the top performer with high accuracy and a good balance between precision and recall.

Run Various Machine Learning Algorithms

We will run various algorithms and check which has the highest Accuracy score, Cross Validation Score and (Accuracy Score - Cross Validation Score) for comparison between all the algorithms.

```
model = LogisticRegression()  
classifier(model, x, y)
```

Accuracy Score: 77.33333333333333

Classification Report:

	precision	recall	f1-score	support
0	0.75	0.82	0.78	147
1	0.81	0.73	0.77	153
accuracy			0.77	300
macro avg	0.78	0.77	0.77	300
weighted avg	0.78	0.77	0.77	300

Cross Validation Score: 75.13333333333334

Accuracy Score - Cross Validation Score is 2.1999999999999886

```
model = GaussianNB()  
classifier(model, x, y)
```

Accuracy Score: 74.66666666666667

Classification Report:

	precision	recall	f1-score	support
0	0.80	0.65	0.71	147
1	0.71	0.84	0.77	153
accuracy			0.75	300
macro avg	0.76	0.74	0.74	300
weighted avg	0.75	0.75	0.74	300

Cross Validation Score: 74.06666666666666

Accuracy Score - Cross Validation Score is 0.6000000000000085

```
model = SVC(kernel='rbf')  
classifier(model, x, y)
```

Accuracy Score: 89.33333333333333

Classification Report:

	precision	recall	f1-score	support
0	0.90	0.88	0.89	147
1	0.89	0.91	0.90	153
accuracy			0.89	300
macro avg	0.89	0.89	0.89	300
weighted avg	0.89	0.89	0.89	300

Cross Validation Score: 86.86666666666667

Accuracy Score - Cross Validation Score is 2.4666666666666544

```
model = SVC(kernel='linear')  
classifier(model, x, y)
```

Accuracy Score: 77.66666666666666

Classification Report:

	precision	recall	f1-score	support
0	0.75	0.82	0.78	147
1	0.81	0.73	0.77	153
accuracy			0.78	300
macro avg	0.78	0.78	0.78	300
weighted avg	0.78	0.78	0.78	300

Cross Validation Score: 75.6

Accuracy Score - Cross Validation Score is 2.066666666666663

```
model = SVC(kernel='poly')
classifier(model, x, y)
```

Accuracy Score: 90.0

Classification Report:				
	precision	recall	f1-score	support
0	0.96	0.83	0.89	147
1	0.86	0.97	0.91	153
accuracy			0.90	300
macro avg	0.91	0.90	0.90	300
weighted avg	0.91	0.90	0.90	300

Cross Validation Score: 87.53333333333335

Accuracy Score - Cross Validation Score is 2.466666666666544

```
model = KNeighborsClassifier()
classifier(model, x, y)
```

Accuracy Score: 64.66666666666666

Classification Report:				
	precision	recall	f1-score	support
0	0.90	0.31	0.46	147
1	0.59	0.97	0.74	153
accuracy			0.65	300
macro avg	0.75	0.64	0.60	300
weighted avg	0.75	0.65	0.60	300

Cross Validation Score: 65.53333333333333

Accuracy Score - Cross Validation Score is -0.866666666666742

```
model = RandomForestClassifier()
classifier(model, x, y)
```

Accuracy Score: 91.0

Classification Report:				
	precision	recall	f1-score	support
0	0.88	0.94	0.91	147
1	0.94	0.88	0.91	153
accuracy			0.91	300
macro avg	0.91	0.91	0.91	300
weighted avg	0.91	0.91	0.91	300

Cross Validation Score: 87.13333333333333

Accuracy Score - Cross Validation Score is 3.866666666666742

```
model = DecisionTreeClassifier()
classifier(model, x, y)
```

Accuracy Score: 84.33333333333334

Classification Report:				
	precision	recall	f1-score	support
0	0.83	0.86	0.84	147
1	0.86	0.83	0.84	153
accuracy			0.84	300
macro avg	0.84	0.84	0.84	300
weighted avg	0.84	0.84	0.84	300

Cross Validation Score: 82.86666666666666

Accuracy Score - Cross Validation Score is 1.466666666666828

```
model = SGDClassifier()
classifier(model, x, y)
```

Accuracy Score: 73.33333333333333

Classification Report:				
	precision	recall	f1-score	support
0	0.74	0.69	0.72	147
1	0.72	0.77	0.75	153
accuracy			0.73	300
macro avg	0.73	0.73	0.73	300
weighted avg	0.73	0.73	0.73	300

Cross Validation Score: 66.33333333333333

Accuracy Score - Cross Validation Score is 7.0

```
model = ExtraTreesClassifier()
classifier(model, x, y)
```

Accuracy Score: 96.0

Classification Report:				
	precision	recall	f1-score	support
0	0.96	0.96	0.96	147
1	0.96	0.96	0.96	153
accuracy			0.96	300
macro avg	0.96	0.96	0.96	300
weighted avg	0.96	0.96	0.96	300

Cross Validation Score: 91.06666666666665

Accuracy Score - Cross Validation Score is 4.93333333333351

<pre>model = AdaBoostClassifier() classifier(model, x, y)</pre>					<pre>model = GradientBoostingClassifier() classifier(model, x, y)</pre>				
Accuracy Score: 88.0					Accuracy Score: 91.66666666666666				
Classification Report: precision recall f1-score support					Classification Report: precision recall f1-score support				
0 0.85 0.92 0.88 147					0 0.90 0.93 0.92 147				
1 0.91 0.84 0.88 153					1 0.93 0.90 0.92 153				
accuracy 0.88 300					accuracy 0.92 300				
macro avg 0.88 0.88 0.88 300					macro avg 0.92 0.92 0.92 300				
weighted avg 0.88 0.88 0.88 300					weighted avg 0.92 0.92 0.92 300				
Cross Validation Score: 84.2					Cross Validation Score: 87.33333333333333				
Accuracy Score - Cross Validation Score is 3.799999999999997					Accuracy Score - Cross Validation Score is 4.333333333333329				

By referring all the above algorithms, we can see that ExtraTreesClassifier gives the best results since the (Accuracy Score - Cross Validation Score) is the least comparing others while having higher Cross Validation Score and the highest Accuracy Score comparing all the other models.

As we have found the best fit model, let's perform Hyper Parameter Tuning to improve the performance of the model.

Hyper parameter tuning

Creating train_test_split and checking the shape of the subsets.

```
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.20,random_state=80)
```

```
x_test.shape
```

```
(300, 36)
```

```
y_test.shape
```

```
(300,)
```

```
x_train.shape
```

```
(1200, 36)
```

```
y_train.shape
```

```
(1200,)
```

Creating a list of parameters to pass into the Grid Search CV.

```
# creating parameters list to pass into GridSearchCV
```

```
parameters = {'criterion' : ['gini', 'entropy'],
              'max_features' : ['auto', 'sqrt', 'log2'],
              'n_jobs' : [-2, -1, 1],
              'random_state' : [None, 80],
              'n_estimators' : [100, 200, 300]}
```

Running Grid Search CV for ExtraTreesClassifier at cv = 5

```
GCV = GridSearchCV(ExtraTreesClassifier(), parameters, cv=5)
GCV.fit(x_train,y_train)

GridSearchCV(cv=5, estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                          'max_features': ['auto', 'sqrt', 'log2'],
                          'n_estimators': [100, 200, 300], 'n_jobs': [-2, -1, 1],
                          'random_state': [None, 80]})
```

Get the list of the best parameters from Grid Search CV.

```
GCV.best_params_      # printing best parameters found by GridSearchCV

{'criterion': 'gini',
 'max_features': 'log2',
 'n_estimators': 200,
 'n_jobs': -2,
 'random_state': None}
```

Here we got the best parameters, we will build our final model using these parameters.

```
final_modelc = ExtraTreesClassifier(criterion = 'gini', max_features = 'log2', n_estimators = 200, n_jobs = -2 , random_state = 1)

final_fitc = final_modelc.fit(x_train,y_train)  # final fit

final_predc = final_modelc.predict(x_test)  # predicting with best parameters
```

The Final Accuracy Score of the final Model.

```
best_acc_score = (accuracy_score(y_test, final_predc))*100  # checking accuracy score
print("The Accuracy Score for the Best Model is ", best_acc_score)
```

The Accuracy Score for the Best Model is 95.0

We successfully performed the Hyper Parameter Tuning on the Final Model.

The Final Cross Validation Score of the final Model.

```
# Final Cross Validation Score
final_cv_score = (cross_val_score(final_modelc, x, y, cv=5).mean())*100
print("Cross Validation Score:", final_cv_score)
```

Cross Validation Score: 91.33333333333333

We got final accuracy score of 95% and Cross Validation Score of 91.3333% which is good.

The Final Classification Report of the final Model.


```
# Final Classification Report
final_class_report = classification_report(y_test, final_predc)
print("\nClassification Report:\n", final_class_report)
```

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

     0       0.96       0.94       0.95        147
     1       0.94       0.96       0.95        153

 accuracy          0.95
 macro avg         0.95
weighted avg         0.95
```

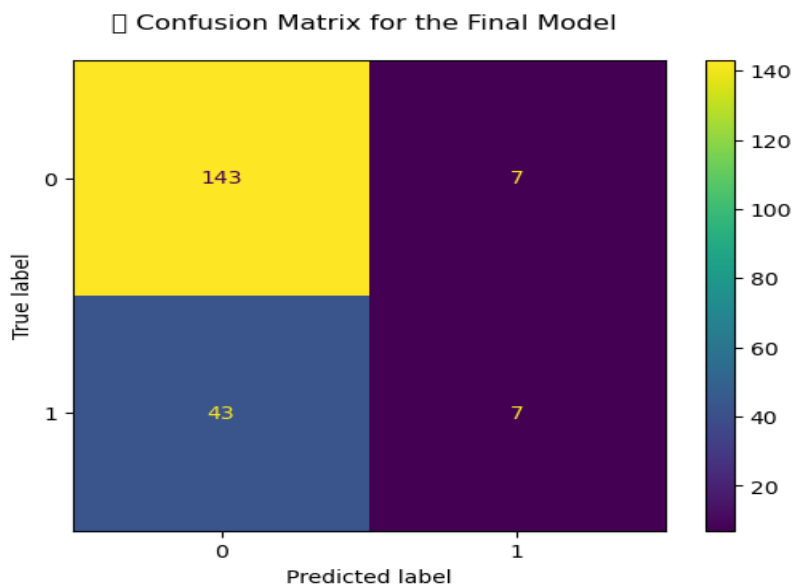
Evaluation Metrics

The performance of the models was evaluated using the following metrics:

- Accuracy
- Precision
- Recall
- F1-Score
- Area Under the Receiver Operating Characteristic Curve (AUC-ROC)

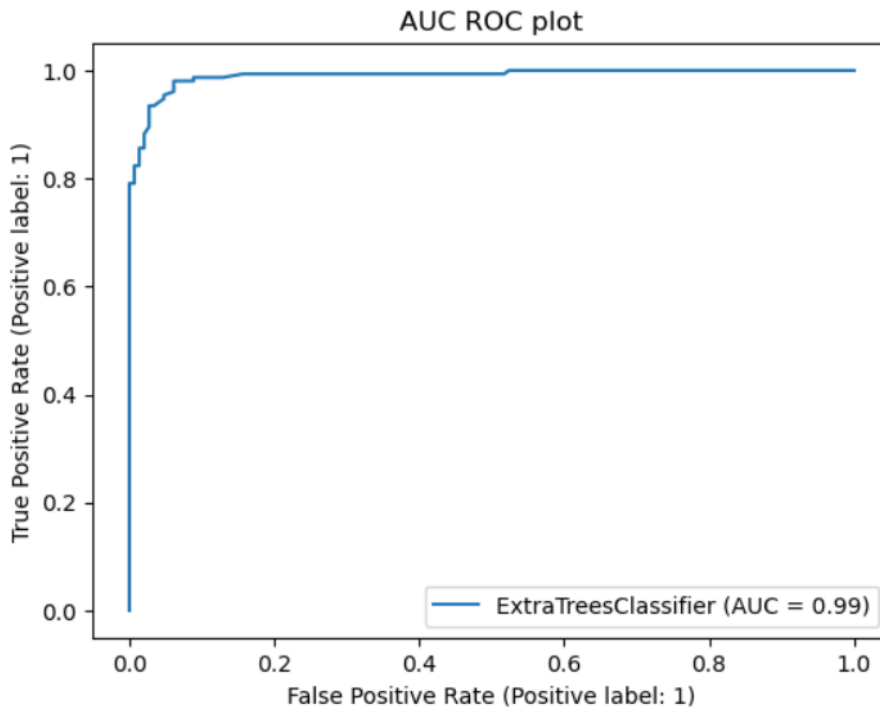
Print Confusion matrix

```
1 from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
2
3 plt.style.use('default')
4 class_names = df.columns
5 #metrics.plot_confusion_matrix(final_modelc, x_test, y_test, cmap='mako')
6 ConfusionMatrixDisplay.from_estimator(final_modelc, x_test, y_test)
7 plt.title('\t Confusion Matrix for the Final Model \n')
8 plt.show()
```



AUC ROC curve

```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(final_modelc, x_test, y_test)
plt.title("AUC ROC plot")
plt.show()
```



We got a final accuracy score of 95% , Cross Validation Score of 91.3% and AUC score is 0.99 which is good.

Save the model in pickle Format

```
1 # pickeling or serialization of a file
2 import pickle
3 filename = 'Insurance_Claims_Fraud_Detection_Final_Model.pkl'
4 pickle.dump(final_modelc, open(filename, 'wb'))
```

Conclusion

Detecting fraudulent insurance claims is a complex yet essential task for the insurance industry. Through this case study, we demonstrated the use of various machine learning models and techniques to identify fraudulent claims effectively. The ExtraTreeClassifier model, with its robustness and high performance, proved to be an excellent choice for this purpose.

Prediction Conclusion

We will predict the "fraud_reported" target column using the final model sending the "x_test" set for predicting the "y_test" and then compare the original "y_test" and the predicted "y_test" in a dataframe.

```
import numpy as np
ac=np.array(y_test)
predictedc=np.array(final_modelc.predict(x_test))
df_comparisonc = pd.DataFrame({"original":ac,"predicted":predictedc},index= range(len(ac)))
df_comparisonc
```

	original	predicted
0	0	0
1	1	1
2	0	1
3	0	0
4	0	0
...
295	0	0
296	0	0
297	1	1
298	1	1
299	0	0

300 rows × 2 columns

Hence predicted the "fraud_reported" using the final Model and presented it as a data frame to compare the predictions.

Save the comparison file as a csv file.

```
1 df_comparisonc.to_csv('Insurance_Claims_Fraud_Detection_Prediction_Results.csv')
```