BLOG

INSURANCE CLAIMS- FRAUD DETECTION

Author: Shashi Sahu

Batch: 1843

DataTrained

INTRODUCTION

One of the biggest preventable losses that hurts insurers worldwide is fraudulent insurance claims.

Insurance fraud is a deliberate deception perpetrated against or by an insurance company or by an agent for financial profit. Fraud may be committed at different points in the transaction by applicants, policyholders, third-party claimants, or professionals who provide services to claimants. Insurance agents and company employees may also commit insurance fraud. Common frauds include "padding," or inflating claims; misrepresenting facts on an insurance application; submitting claims for injuries or damage that never occurred; and staging accidents.

Furthermore, we use machine learning to predict which claims are likely to be fraudulent. This information can narrow down the list of claims that need a further check. It enables an insurer to detect more fraudulent claims.

What is Insurance Fraud?

When an insurance firm, agent, adjuster, or customer intentionally lies to get an unfair advantage, then insurance fraud occurs. It can happen when purchasing, utilizing, reselling, or underwriting insurance. Insurance fraud can be classified into a variety of subcategories, including fraud committed by both consumers and insurance firms. Fraud affects consumers and businesses financially, in addition to adding to insurance companies' costs.

Data Science and Machine Learning, which has been very useful in many industries that have always managed to bring accuracy or detect negative incidents.

In this blog, I have created a Machine Learning model to detect if the claim is fraudulent or not. Here various features have been used like, insured personal information, insured persons, personal details, and incident information. In total the dataset has 40 different features. So using all these previously acquired information and analysis done with the data I have achieved a good model that has more than 93% accuracy rate. So let's see what steps are involved to attain this accuracy. Various visualization techniques have also been used to understand the co-linearity and importance of the features.

Hardware & Software Requirements & Tools Used:

Hardware required:
•Processor: Intel(R) Core(TM) i3-7020U CPU @ 2.30GHz 2.30 GHz
•RAM: 8 GB or above
•ROM/SSD: 250 GB or above

Software requirement:
• Jupiter Notebook

Libraries Used:
• Python
• NumPy
• Pandas
• Matplotlib
• Seaborn
• Date Time
• Scikit Learn

1.PROBLEM DEFINITION

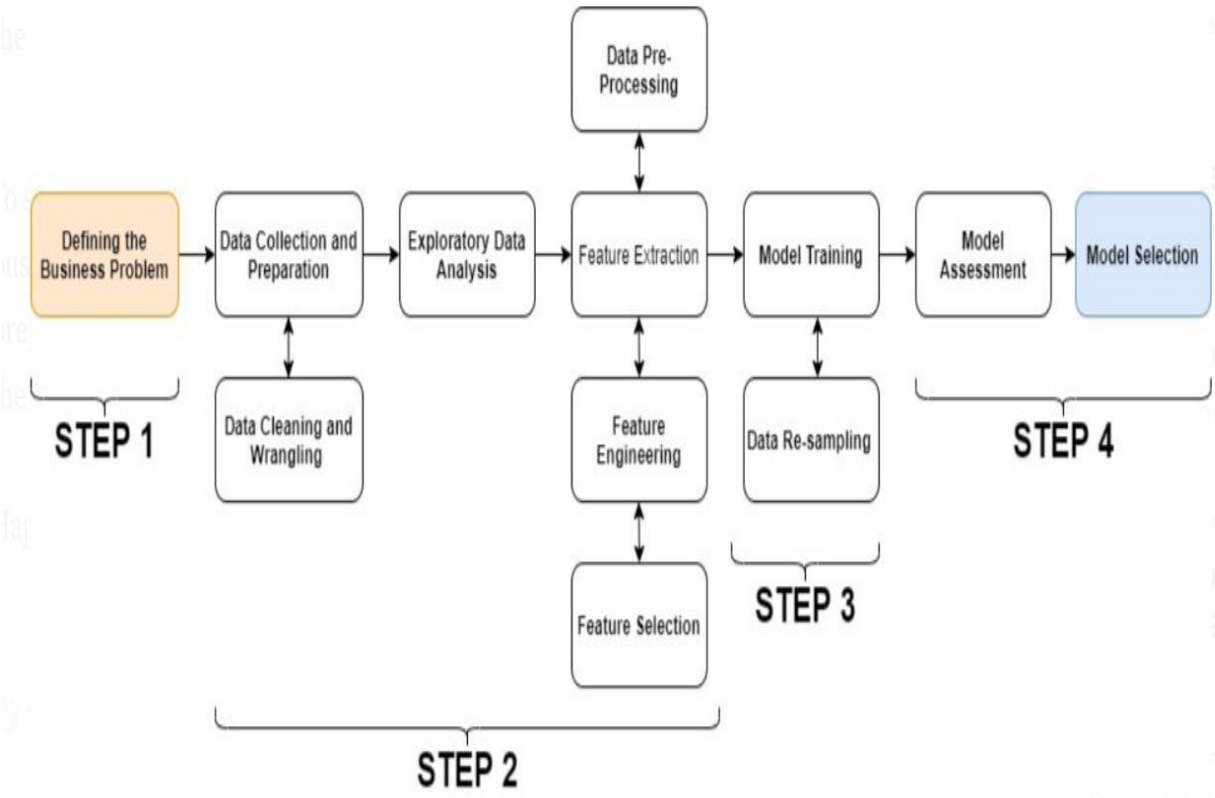
Insurance fraud is a large problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

In this project, we have provided a dataset which has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

In this example, I will be working with some auto insurance data to demonstrate how we can create a predictive model that predicts if an insurance claim is fraudulent or not.

Workflow of a Machine Learning Project:

To build a Machine Learning Model, we have a Machine Learning Workflow that every Machine Learning algorithm has to touch upon in the life of the model. Let's a sneak peek into the model workflow and then we will look into the actual machine learning model and understand it better along with the workflow.



Now that we understand the lifecycle of a Machine Learning Model, let's import the necessary libraries and proceed further.

Importing the necessary Libraries:

To analyze the dataset, we have imported all the necessary libraries as shown below.

- Pandas have been used to import the dataset and also in creating data frames.
- Seaborn and Matplotlib have been used for visualization
- Numpy have been used to perform numerical operations.
- Sklearn has been used in the model building

```
In [1]: import numpy as np
        import pandas as pd
        import seaborn as sns
        import matplotlib.pyplot as plt
        import warnings
        import os
        import scipy as stats
        %matplotlib inline
        warnings.filterwarnings('ignore')
```

Importing the Dataset:

Let’s import the dataset from the source.

```
# Reading the csv file from dataset
df = pd.read_csv("https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fr
pd.set_option("display.max_columns",None)
df
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	17-10-2014	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	27-06-2006	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	06-09-2000	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	25-05-1990	IL	250/500	2000	1415.74	6000000	608117
4	228	44	367455	06-06-2014	IL	500/1000	1000	1583.91	6000000	610706
...	...	...	...	...	...	...	...	...	...	..
995	3	38	941851	16-07-1991	OH	500/1000	1000	1310.80	0	431286
996	285	41	186934	05-01-2014	IL	100/300	1000	1436.79	0	608177
997	130	34	918516	17-02-2003	OH	250/500	500	1383.49	3000000	442797
998	458	62	533940	18-11-2011	IL	500/1000	2000	1356.92	5000000	441714
999	456	60	556080	11-11-1996	OH	250/500	1000	766.19	0	612260

1000 rows × 40 columns

Here we have imported the data using pd.read_csv. The dataset contains the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

The dataset contains both **categorical** and **numerical** columns. Here **"fraud_reported"** is our **target column**, since it has two categories so it termed to be **"Classification Problem"** where we need to predict if an insurance claim is fraudulent or not.

By using 'df. shape' we can figure out how many rows and columns we have in our dataset. We have got the output that we have 1000 rows and 40 columns. PCA can be done, however, I decided not to lose any data at this time as the dataset is comparatively small in size, and the first lesson of a data scientist is 'Data is Crucial' hence proceeded will all the datasets.

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

```
(1000, 40)
```

As per the workflow of the machine learning model, we have defined our problem and we have selected the data from the source. Now we will perform EDA, data pre-processing, and transformation, which is the most important part of any machine learning model, further data will be analyzed and cleaned the data for

better model accuracy which we will get, or the model can remain overfitting or underfitting. We will discuss further where all the steps are used.

2. EXPLORATORY DATA ANALYSIS

Data Preparation:

Now, we will explore 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 head, tail, sample, unique values, value counts, info, etc.
- After analyzing if we find any unnecessary columns in the datasets, we can use the drop command to drop those columns.

```
In [3]: df.head()  # shows top 5 columns of the dataset
```

Out[3]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	17-10-2014	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	27-06-2006	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	06-09-2000	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	25-05-1990	IL	250/500	2000	1415.74	6000000	608117
4	228	44	367455	06-06-2014	IL	500/1000	1000	1583.91	6000000	610706

```
In [4]: df.tail() # shows bottom 5 columns of the dataset
```

Out[4]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
995	3	38	941851	16-07-1991	OH	500/1000	1000	1310.80	0	431289
996	285	41	186934	05-01-2014	IL	100/300	1000	1436.79	0	608177
997	130	34	918516	17-02-2003	OH	250/500	500	1383.49	3000000	442797
998	458	62	533940	18-11-2011	IL	500/1000	2000	1356.92	5000000	441714
999	456	60	556080	11-11-1996	OH	250/500	1000	766.19	0	612260

```
In [5]: df.sample() # shows any one column from the dataset
```

Out[5]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
357	110	28	435784	13-07-2013	OH	250/500	1000	1573.93	0	461919

```
In [6]: df.sample(3) # shows any 3 sample columns of the dataset
```

Out[6]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
612	56	42	655356	07-07-1996	IL	250/500	500	1339.39	0	471786
341	299	42	337677	20-07-2008	OH	100/300	2000	1437.33	0	450339
184	296	46	922167	23-02-1993	OH	100/300	1000	1141.35	7000000	476456

```
In [8]: # To get good overview of the dataset
        df.info()
```

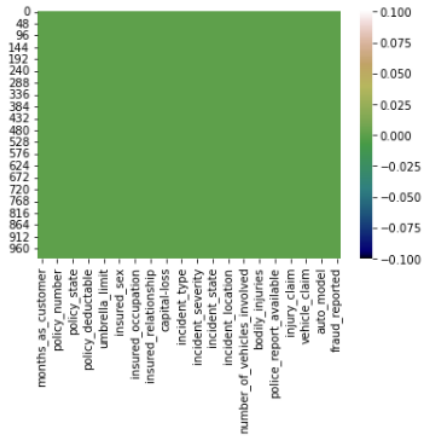
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           1000 non-null   int64  
 1   age                          1000 non-null   int64  
 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        1000 non-null   object 
 22  incident_state               1000 non-null   object 
 23  incident_city                1000 non-null   object 
 24  incident_location            1000 non-null   object 
 25  incident_hour_of_the_day     1000 non-null   int64  
 26  number_of_vehicles_involved  1000 non-null   int64  
 27  property_damage              1000 non-null   object 
 28  bodily_injuries              1000 non-null   int64  
 29  witnesses                    1000 non-null   int64  
 30  police_report_available      1000 non-null   object 
 31  total_claim_amount           1000 non-null   int64  
 32  injury_claim                 1000 non-null   int64  
 33  property_claim               1000 non-null   int64  
 34  vehicle_claim                1000 non-null   int64  
 35  auto_make                    1000 non-null   object 
 36  auto_model                   1000 non-null   object 
 37  auto_year                    1000 non-null   int64  
 38  fraud_reported               1000 non-null   object 
 39  _c39                         0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 312.6+ KB
```

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

After doing this basic analysis, now we are checking for the null values and further will mention all the observations.

```
In [10]: # Checking null values
         df.isnull().sum()
```

```
In [11]: # Let's visualize the null values clearly
         sns.heatmap(df.isnull(), cmap="gist_earth")
         plt.show()
```



In [13]:
```
# Checking number of unique values in each column
df.nunique().to_frame("No of Unique Values")
```

Out[13]:

In [11]:
```
# Checking the value counts of each columns
for i in df.columns:
        print(df[i].value_counts())
        print('*'*100)
```

3. EDA CONCLUDING REMARKS

Observations:

- As we can see the dataset does not have any null values in it.
- Dataset contains 3 different types of data namely integer data type, float data type, and object data type.
- After analyzing we can observe that the c_39 column has only NaN entries. Keeping all entries NaN is useless for evaluation, hence we are dropping that column.

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

- It can observe that column policy number and incident location have 1000 unique values which mean they have only one value count. So it is not required for our analysis 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)
```

- 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.

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

- The column insured zip is the zip code assigned 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 which means the 5 entries are repeating. Since it is giving some information about the person, either we can drop this or we can convert its data type from integer to object for better processing.
-
```
# Dropping insured_zip column as it is not important for the prediction
df.drop('insured_zip',axis=1,inplace=True)
```

Proceeding to Feature Extraction:

policy_bind_date and incident_date have object data types that should be in the DateTime data type, which means python is not able to understand the type of

this column and give the default data type. We will convert this object data type to the Date Time data type and we will extract the values from these columns.

```
In [14]: # 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'])
```

As we have converted the object data type into a DateTime data type. Let's extract Day, Month, and Year from both columns.

```
In [16]: # 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
```

We have extracted Day, Month, &Year columns, from the policy_bind_date and incident_date columns. So we can drop these columns now.

```
In [17]: # Dropping policy_bind_date and incident_date columns
         df.drop(["policy_bind_date","incident_date"],axis=1,inplace=True)
```

From the features we can see that the policy_csl column is showing as an object data type but it contains numerical data, maybe it is because of the presence of "/" in that column. So first we will extract two columns csl_per_person and csl_per_accident from policy_csl columns and then will convert their object data type into an integer data type.

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

```
: # Converting object data type into integer data type
  df['csl_per_person'] = df['csl_per_person'].astype('int64')
  df['csl_per_accident'] = df['csl_per_accident'].astype('int64')
```

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

- After extracting we have dropped the policy_csl feature.
- 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 also.

Moving on to Imputation:

Imputation is a technique to fill null values in the dataset using mean, median, or mode. you might be thinking that we did not get any null values while checking for the null values in the dataset, 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()
```

These columns contain the "?" sign. Since this column seems to be categorical so we will replace "?" values with the most frequently occurring values of the respective columns i.e. 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 collision_type is: 0    Rear Collision
Name: collision_type, dtype: object
The mode of property_damage is: 0    ?
Name: property_damage, dtype: object
The mode of police_report_available is: 0     ?
1    NO
Name: police_report_available, dtype: object
```

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

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

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

Unique values present in each column after feature extraction and selection. Here incident_Year column 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)
```

After cleaning the dataset until now, the dataset looks like this!

In [29]:
```
# Let's take a look at the dataframe after preprocessing
df.head()
```

Out[29]:

	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

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)
print("\n")

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

These are the categorical and numerical columns present in the dataset.

Categorical columns are:
['policy_state', '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', 'fraud_reported']

Numerical columns are:
['months_as_customer', 'age', 'policy_deductable', 'policy_annual_premium', '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', 'policy_bind_Day', 'policy_bind_Month', 'policy_bind_Year', 'incident_Day', 'incident_Month', 'csl_per_person', 'csl_per_accident', 'Vehicle_Age']

```
# Checking the list of counts of target
df['fraud_reported'].unique()
```

```
array(['Y', 'N'], dtype=object)
```

There are two unique categories in fraud_reported column these values tells about if an insurance claim is fraudulent or not. Here we can assume that "Y" stands for "Yes" that is the insurance is fraudulent and "N" stands for "No" means the insurance claim is not fraudulent.

```
# Checking the unique values in target column
df['fraud_reported'].value_counts()
```

```
N    753
Y    247
```

From the value count of the target column fraud_reported we can notice the count of the categories are different which means the data is not balanced. We will use oversampling method to balance the data before building the models.

Statistical summary of numerical columns:

This gives the statistical information of the numerical columns present in the dataframe.

```
df.describe()
```

	months_as_customer	age	policy_deductable	policy_annual_premium	capital-gains	capital-loss	incident_hour_of_the_day	number_of_vehic
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	
mean	203.954000	38.948000	1136.000000	1256.406150	25126.100000	-26793.700000	11.644000	
std	115.113174	9.140287	611.864673	244.167395	27872.187708	28104.096686	6.951373	
min	0.000000	19.000000	500.000000	433.330000	0.000000	-111100.000000	0.000000	
25%	115.750000	32.000000	500.000000	1089.607500	0.000000	-51500.000000	6.000000	
50%	199.500000	38.000000	1000.000000	1257.200000	0.000000	-23250.000000	12.000000	
75%	276.250000	44.000000	2000.000000	1415.695000	51025.000000	0.000000	17.000000	
max	479.000000	64.000000	2000.000000	2047.590000	100500.000000	0.000000	23.000000	

The summary of this dataset looks perfect since there is no negative/ invalid values present.

From the above description we can observe the following things:

- Here the counts of all the columns are equal which means there are no missing values in the dataset.
- In some of the columns like policy_deductable, capital-gains, injury_claim etc we can observe the mean value is greater than the median (50%) which means the data in those columns are skewed to right.
- And in some of the columns like total_claim_amount, vehicle_claim...etc we can observe the median is greater than the mean which means the data in the columns are skewed to left.
- And some of the columns have equal mean and median that means the data symmetric and is normally distributed and no skewness preaent.
- There is a huge difference in 75% and max it shows that huge outliers present in the columns.

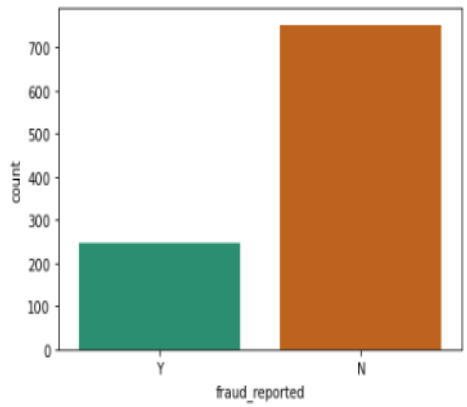
DATA VISUALIZATION

❖ Univariate Analysis

➢ Plotting categorical columns:

```
#Visualizing how many insurance claims is fraudulent
print(df["fraud_reported"].value_counts())
sns.countplot(df["fraud_reported"],palette="Dark2")
plt.show()
```

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



From the plot we can observe that the count of "N" is high compared to "N". Which means here we can assume that "Y" stands for "Yes" that is the insurance is fraudulent and "N" stands for "No" means the insurance claim is not fraudulent. Here most of the insurance claims have not reported as fraudulent.

Since it is our target column, it indicates the class imbalance issue. We will balance the data using oversampling method in later part.

➢ Visualize data using Pie chart:

```
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','prope

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

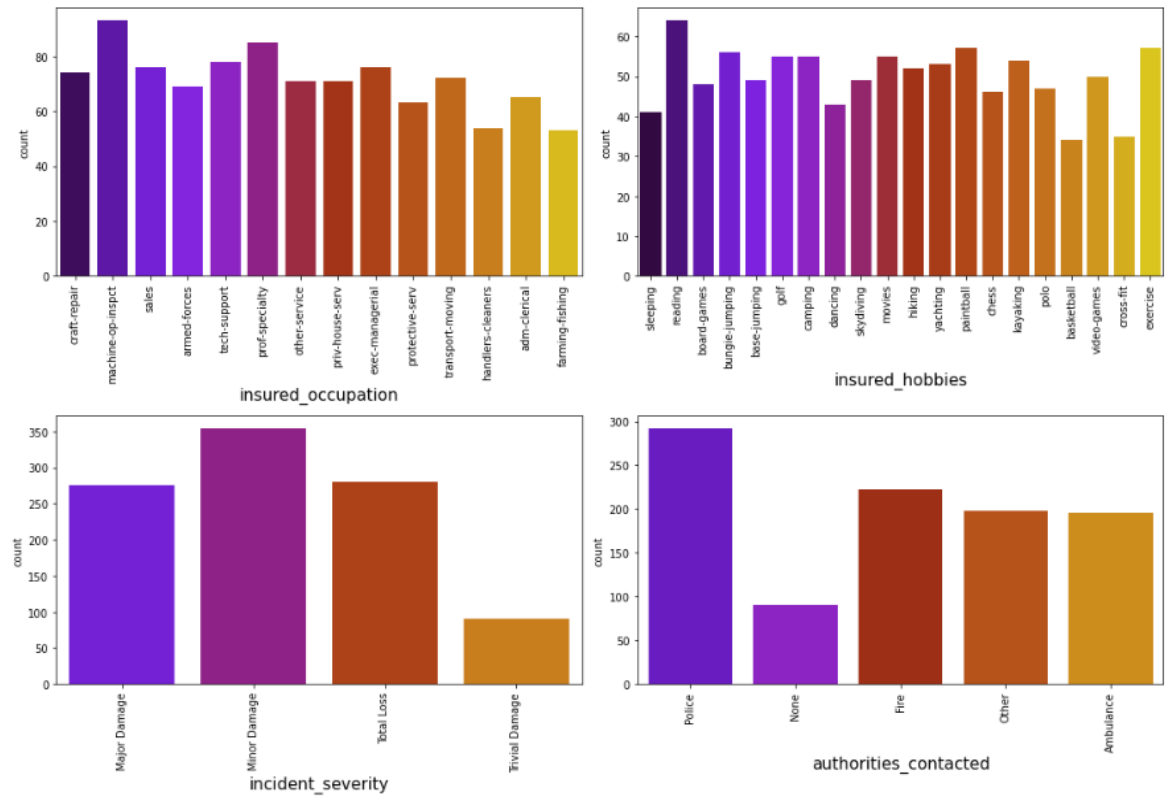
Above are the pie plots for some of the categorical columns. From the above pie plots we can observe the following things:

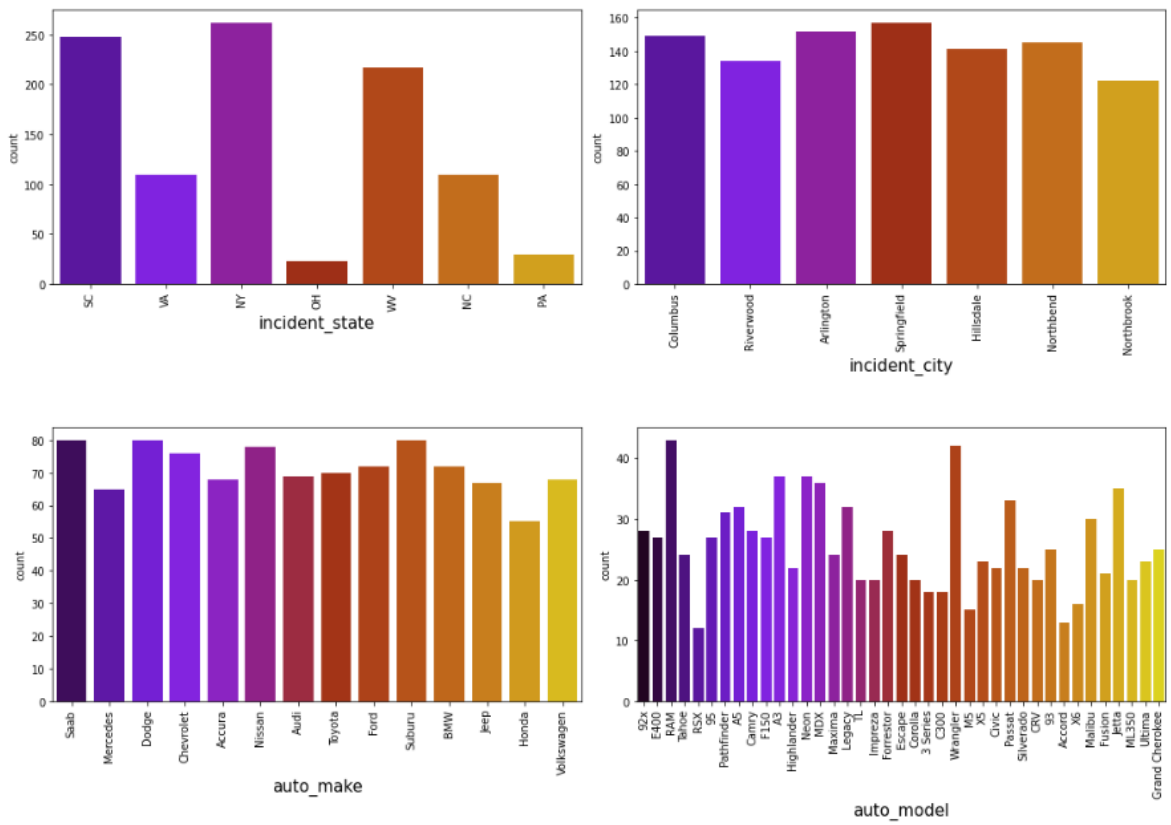
- The types of the policies claimed by the customers are almost same but still the policy state type HO has bit high counts and the type IL has bit less count.
- Both male and female have insurance but the count for Female is bit higher than Male counts.
- The count is pretty much same for all the education 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.

- While looking at the incident type, Multi-vehicle collision and Single Vehicle Collision have pretty much similar counts of around 41% and 400%. 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% of the people did not face any property damage while 30% of the people faced the property damage.
- Over 68% of the people were produced the police reports while 31% of the people didn't submit any police reports.

➢ Count plots:

```
cols2 = ['insured_occupation', 'insured_hobbies', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_city'

plt.figure(figsize=(15,25),facecolor='white')
plotnumber=1
for column in cols2:
    if plotnumber:
        ax=plt.subplot(5,2,plotnumber)
        sns.countplot(df[column],palette="gnuplot")
        plt.xticks(rotation=90)
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.tight_layout()
plt.show()
```





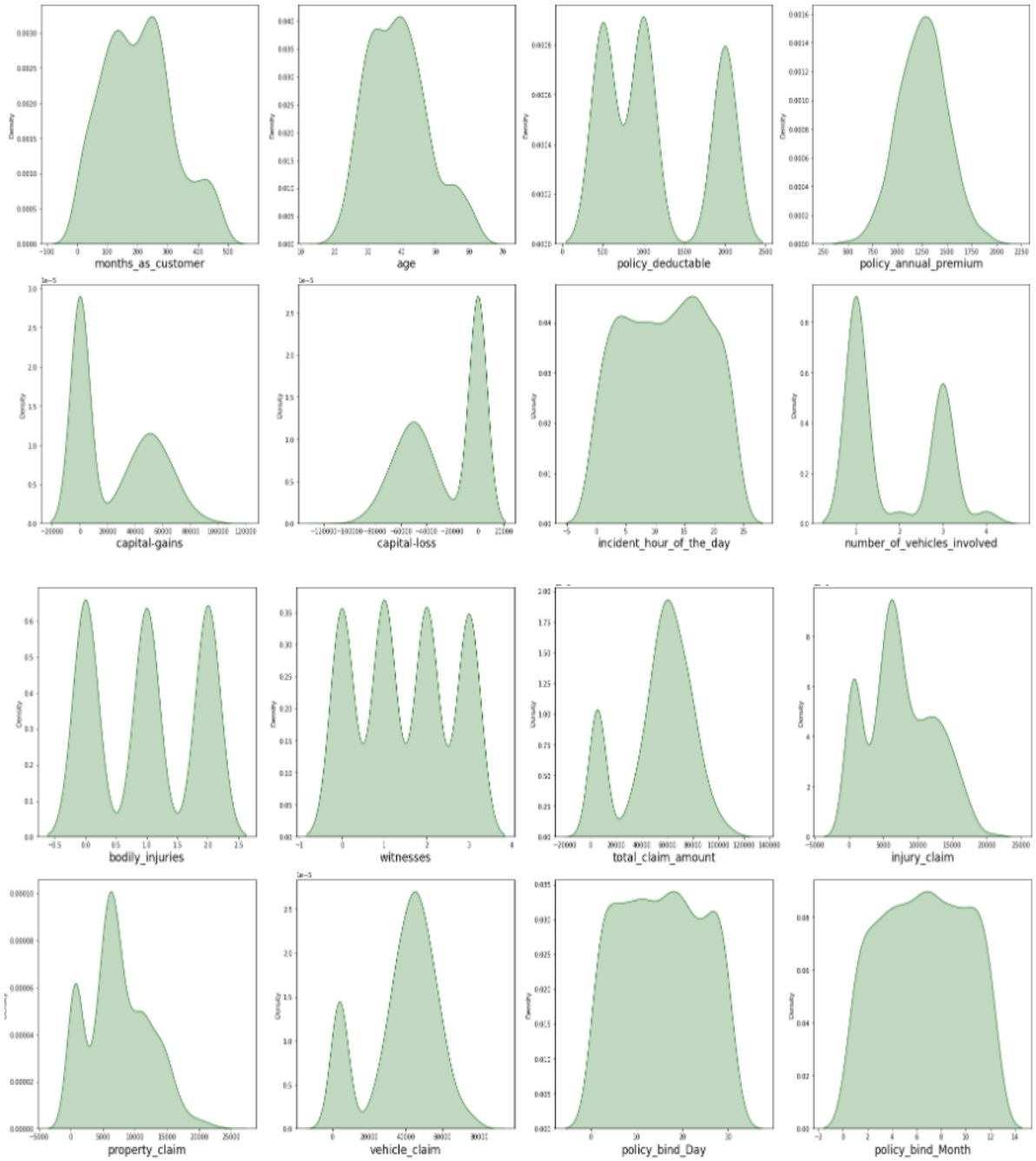
Above are the count plots for the remaining categorical columns. From the count plots we can observe the following things:

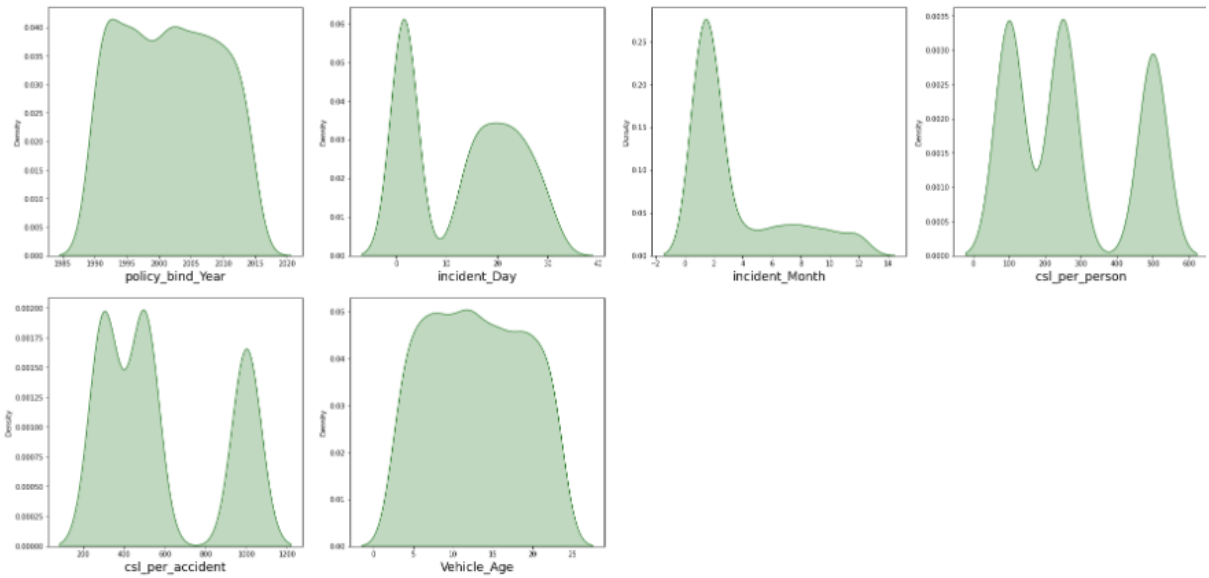
- In the insured occupation we can observe most of the data is covered by machine operation inspector followed by professional speciality. Apart from this all the other insured occupations have almost same couns.
- With respect to insured hobbies, we can notice reading covered the highest data followed by exercise. And other categories have the average counts.
- The incident severity count is high for Minor damages and trivial damage data has very less count compared to others.
- When the accidents occurs then most of the authorities contacts the police, here the category police covers highest data and Fire having the second highest count. But Ambulance and Others have almost same counts and the count is very less for None compared to all.
- With respect to the incident state, New York, South Carolina and West Virginia states have highest counts.
- In incident city, almost all the columns have equal counts.
- When we look at the vehicle manufactured companies, the categories Saab, Suburu, Dodge, Nissan and Volkswagen have highest counts.
- When we take a look at the vehicle models then RAM and Wrangler automobile models have highest counts and also RSX and Accord have very less count.

➢ Distribution Plot:

a) Plotting numerical columns

```
# Checking how the data has been distriubted in each column

plt.figure(figsize=(25,35),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=23:
        ax=plt.subplot(6,4,plotnumber)
        sns.distplot(df[column],color="darkgreen",hist=False,kde_kws={"shade": True})
        plt.xlabel(column,fontsize=18)
    plotnumber+=1
plt.tight_layout()
```





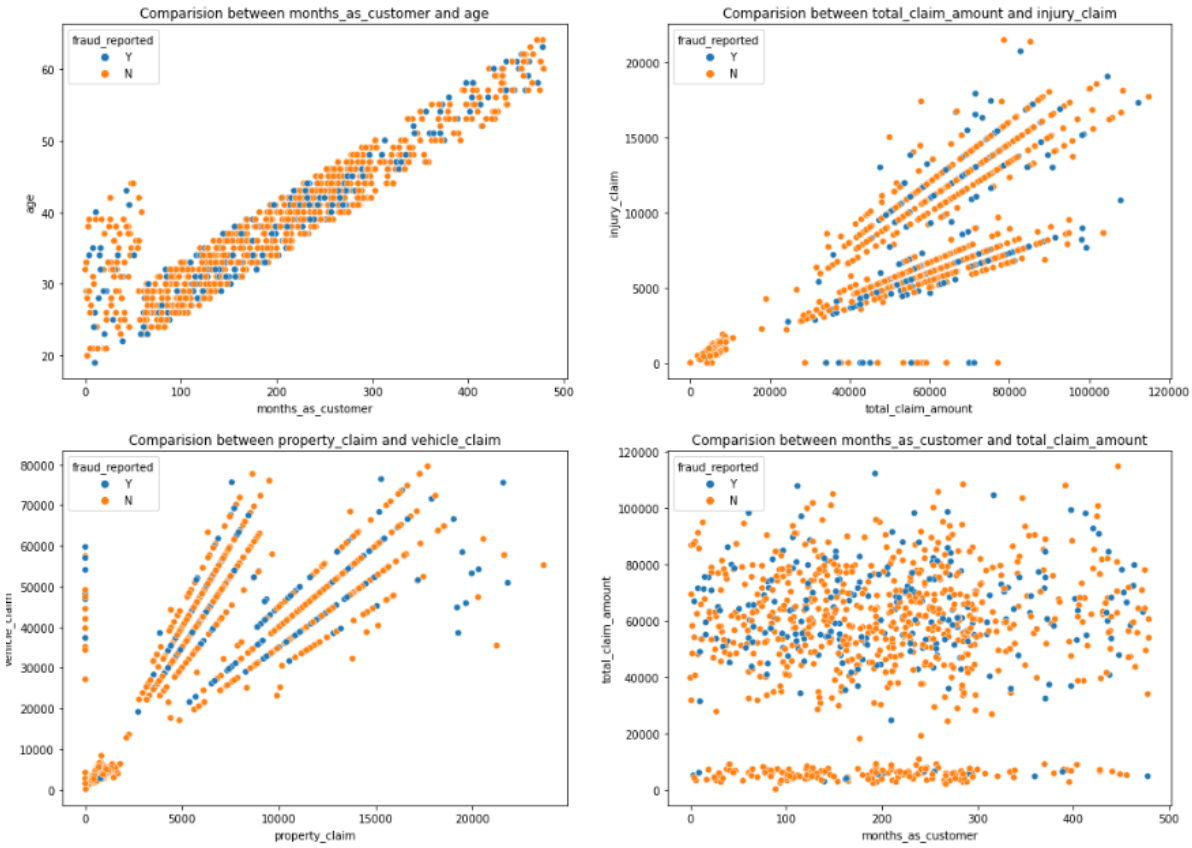
Above are the distribution plots for all the numerical columns. From the distplots we can observe the following things:

- The data is normally distributed in most of the columns.
- Some of the columns like capital gains and incident months have mean value greater than the median, hence they are skewed to right.
- The data in the column capital loss is skewed to left since the median is greater than the mean.
- We will remove the skewness using appropriate methods in the later part.

❖ Bivariate Analysis:

➢ Scatter Plot:

```
# Comparision between two variables
plt.figure(figsize=[18,13])

plt.subplot(2,2,1)
plt.title('Comparision between months_as_customer and age')
sns.scatterplot(df['months_as_customer'],df['age'],hue=df['fraud_reported']);

plt.subplot(2,2,2)
plt.title('Comparision between total_claim_amount and injury_claim')
sns.scatterplot(df['total_claim_amount'],df['injury_claim'],hue=df['fraud_reported']);

plt.subplot(2,2,3)
plt.title('Comparision between property_claim and vehicle_claim')
sns.scatterplot(df['property_claim'],df['vehicle_claim'],hue=df['fraud_reported']);

plt.subplot(2,2,4)
plt.title('Comparision between months_as_customer and total_claim_amount')
sns.scatterplot(df['months_as_customer'],df['total_claim_amount'],hue=df['fraud_reported']);
```

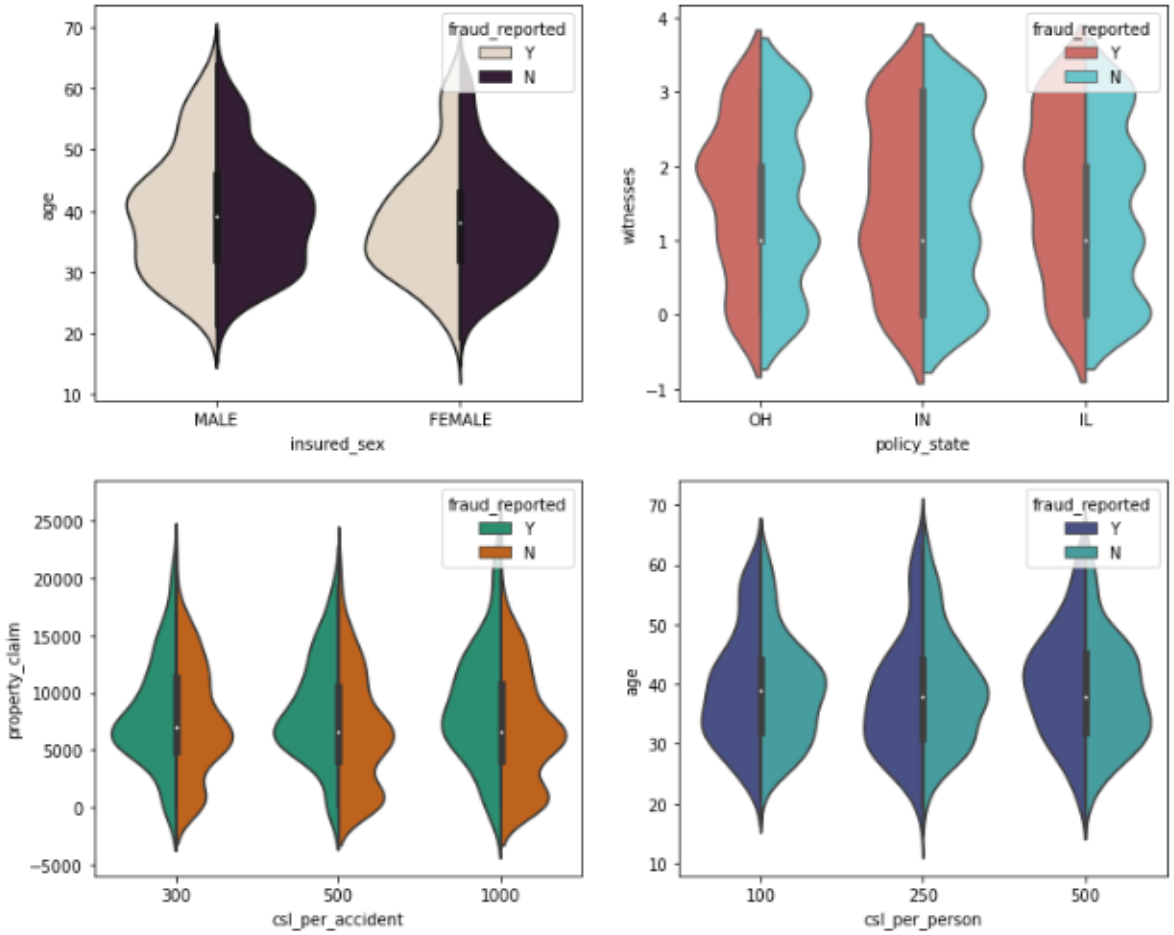


From the above scatter plot we can observe the following things:

- There is a positive linear relation between age and month_as_customer column. As age increases the month_as customers also increases, also the fraud reported is very less in this case.
- In the second graph we can observe the positive linear relation, as total claim amount increases, injury claim is also increases.
- Third plot is also same as second one that is as the property claim increases, vehicle claim is also increases.
- In the fourth plot we can observe the data is scattered and there is no much relation between the features.

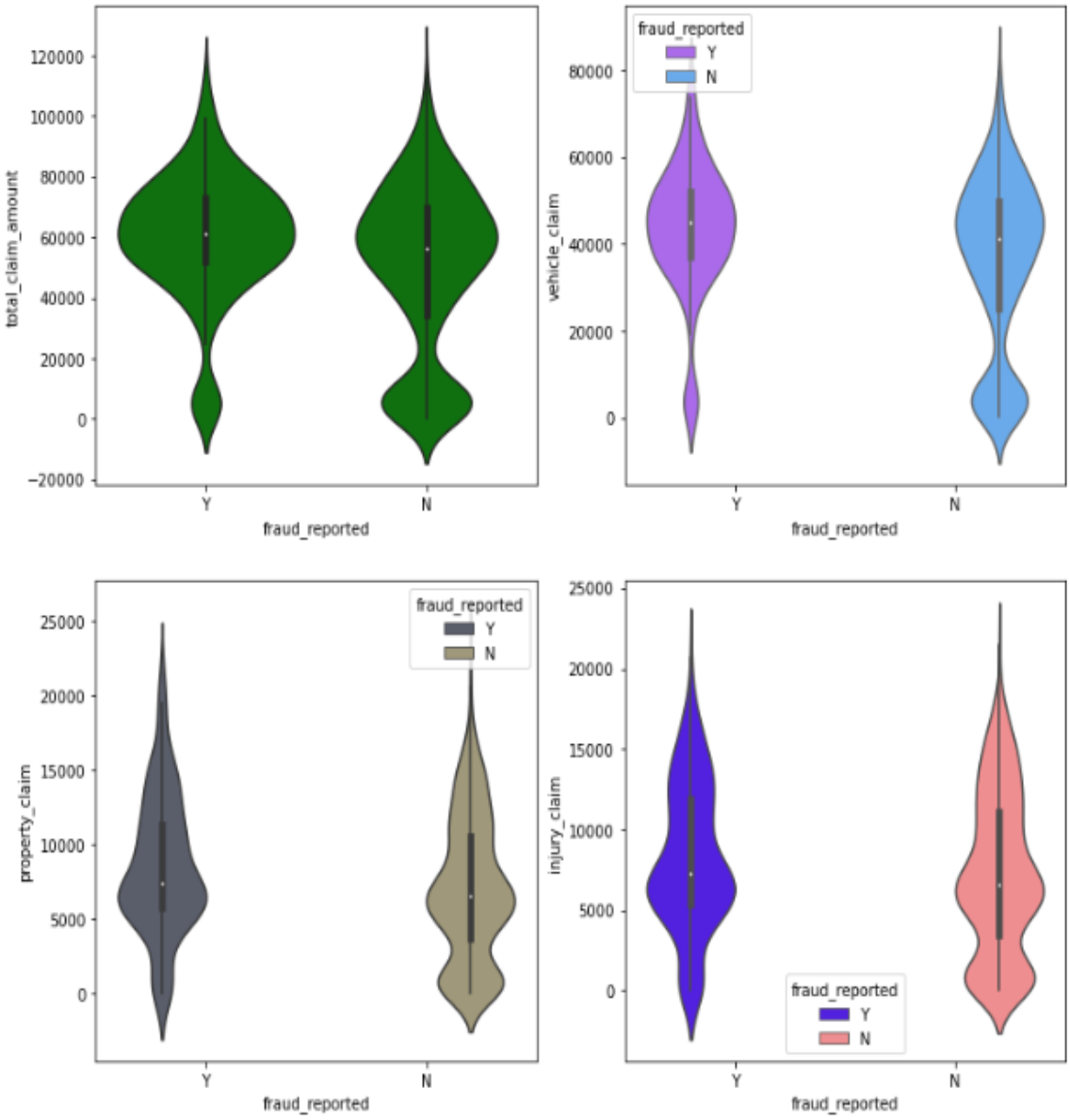
➢ Violin Plot:

```
fig,axes=plt.subplots(2,2,figsize=(12,10))

# Comparing insured_sex and age
sns.violinplot(x='insured_sex',y='age',ax=axes[0,0],data=df,palette="ch:.25",hue="fraud_reported",split=True)

# Comparing policy_state and witnesses
sns.violinplot(x='policy_state',y='witnesses',ax=axes[0,1],data=df,hue="fraud_reported",split=True,palette="hls")

# Comparing csl_per_accident and property_claim
sns.violinplot(x='csl_per_accident',y='property_claim',ax=axes[1,0],data=df,hue="fraud_reported",split=True,palette="Dark2")

# Comparing csl_per_person and age
sns.violinplot(x='csl_per_person',y='age',ax=axes[1,1],data=df,hue="fraud_reported",split=True,palette="mako")
plt.show()
```



- The fraud report is high for both the males, females having age between 30-45.
- The people who own the policy state "IN" have high fraud report.
- The person who has csl per accident insurance by claiming property in the range 5000-15000 have the fraud report.
- The csl_per_person with age 30-45 are facing the fraudulent reports.

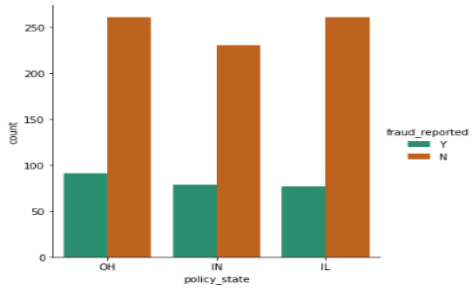
```
fig,axes=plt.subplots(2,2,figsize=(12,12))

# Comparing insured_sex and age
sns.violinplot(x='fraud_reported',y='total_claim_amount',ax=axes[0,0],data=df,color="g")

# Comparing policy_state and witnesses
sns.violinplot(x='fraud_reported',y='vehicle_claim',ax=axes[0,1],data=df,hue="fraud_reported",palette="cool_r")

# Comparing csl_per_accident and property_claim
sns.violinplot(x='fraud_reported',y='property_claim',ax=axes[1,0],data=df,hue="fraud_reported",palette="cividis")

# Comparing csl_per_person and age
sns.violinplot(x='fraud_reported',y='injury_claim',ax=axes[1,1],data=df,hue="fraud_reported",palette="gnuplot2")
plt.show()
```



- Most of the fraud reports found when the total claimed amount is 5000-70000.
- The fraud report is high when the claimed vehicle is between 3700-5900.
- The fraud reports is high when the property claimed is between 5200-8500.
- Most fraud reported when injury claim are between 5000 to 8000.

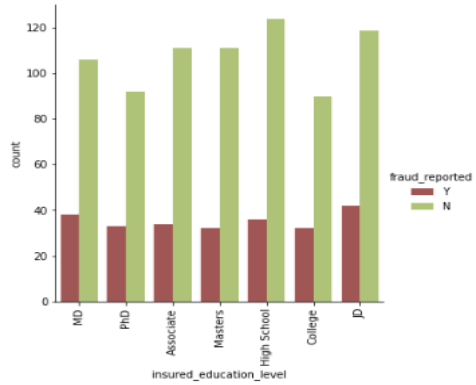
❖ Factor Plot:

```
# Comparing policy_state and fraud_reported
sns.factorplot('policy_state',kind='count',data=df,hue='fraud_reported',palette="Dark2")
plt.show()
```



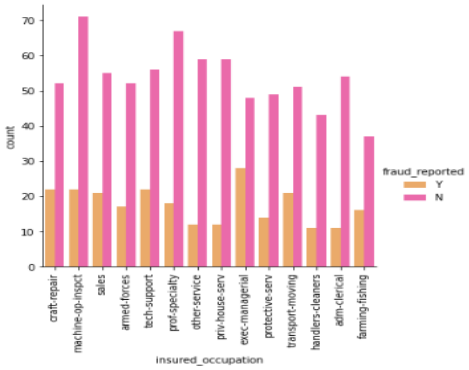
Fraud report is bit high in the "OH" policy state.

```
# Comparing insured_education_level and fraud_reported

sns.factorplot('insured_education_level',kind='count',data=df,hue='fraud_reported',palette="tab20b_r")
plt.xticks(rotation=90)
plt.show()
```



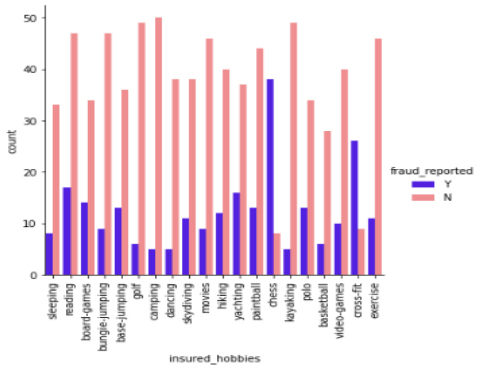
The fraudulent level is very less for the people who have high school education and the people who have completed their "JD" education have high fraud report. The people who have high insured education are facing insurance fraudulent compared to the people with less insured education level.

```
# Comparing insured_occupation and fraud_reported
sns.factorplot('insured_occupation',kind='count',data=df,hue='fraud_reported',palette="spring_r")
plt.xticks(rotation=90)
plt.show()
```



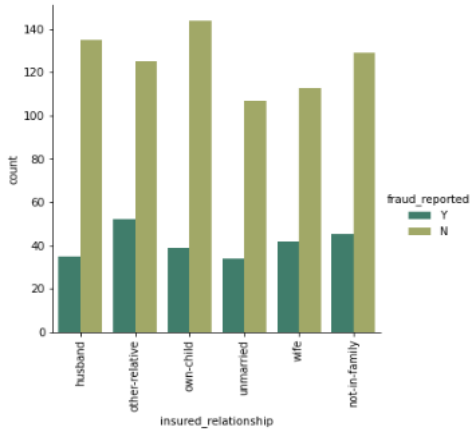
The people who are in the position exec-managerial have high fraud reports compared to others.

```
# Comparing insured_hobbies and fraud_reported
sns.factorplot('insured_hobbies',kind='count',data=df,hue='fraud_reported',palette="gnuplot2")
plt.xticks(rotation=90)
plt.show()
```



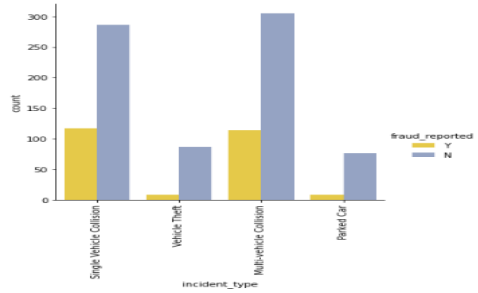
The fraud report is high for the people who have the hobby of playing chess and cross fit.

```
# Comparing insured_relationship and fraud_reported
sns.factorplot('insured_relationship',kind='count',data=df,hue='fraud_reported',palette="gist_earth")
plt.xticks(rotation=90)
plt.show()
```



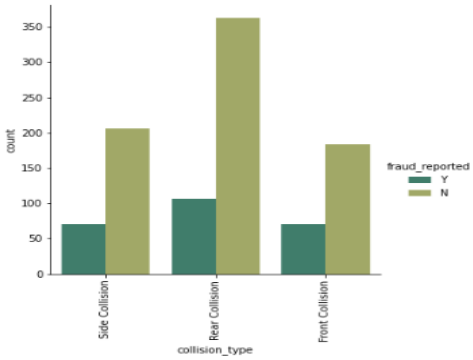
The fraud report is high for the customers who have other relative and it is very less for unmarried people.

```
# Comparing incident_type and fraud_reported
sns.factorplot('incident_type',kind='count',data=df,hue='fraud_reported',palette="Set2_r")
plt.xticks(rotation=90)
plt.show()
```



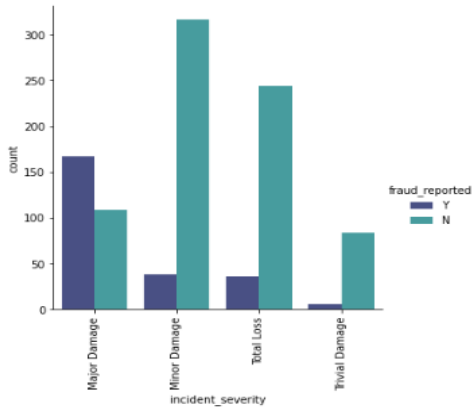
In Multivehicle collision and single vehicle collision, the fraud report is very high compared to others.

```
# Comparing collision_type and fraud_reported
sns.factorplot('collision_type',kind='count',data=df,hue='fraud_reported',palette="gist_earth")
plt.xticks(rotation=90)
plt.show()
```



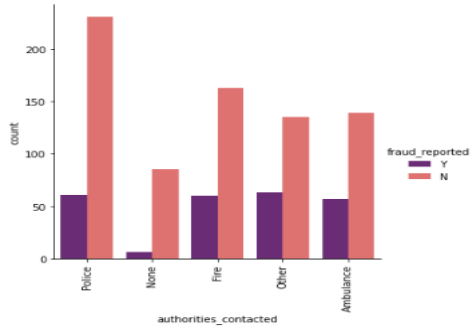
The fraud level is high in the collision type Rear Collision and other two collision type have average reports.

```
# Comparing incident_severity and fraud_reported
sns.factorplot('incident_severity',kind='count',data=df,hue='fraud_reported',palette="mako")
plt.xticks(rotation=90)
plt.show()
```



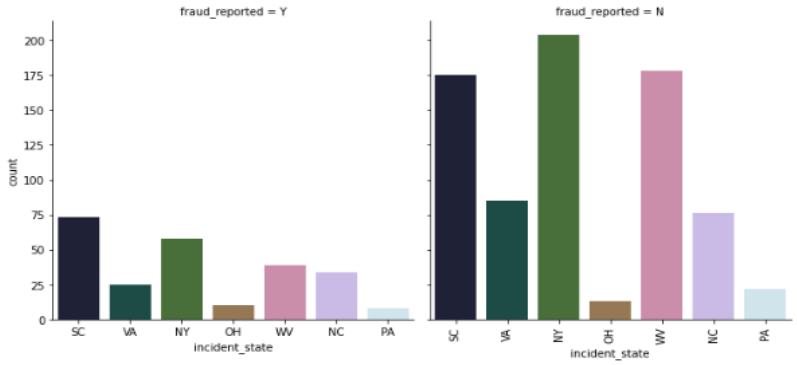
The fraud report is high in Major damage incident severity and for Trivial Damage the report is less compared to others.

```
# Comparing authorities_contacted and fraud_reported
sns.factorplot('authorities_contacted',kind='count',data=df,hue='fraud_reported',palette="magma")
plt.xticks(rotation=90)
plt.show()
```



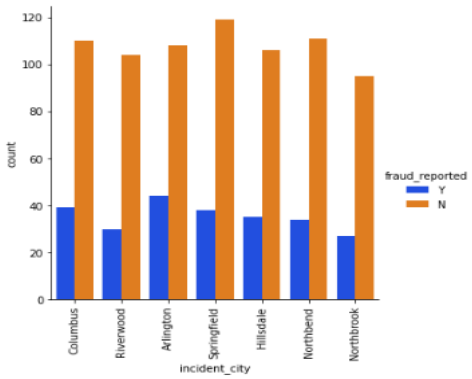
The police contacted cases are very high and the fraud report is in equal for all the authorities except None.

```
# Comparing incident_state and fraud_reported
sns.factorplot('incident_state',kind='count',data=df,col='fraud_reported',palette="cubehelix")
plt.xticks(rotation=90)
plt.show()
```



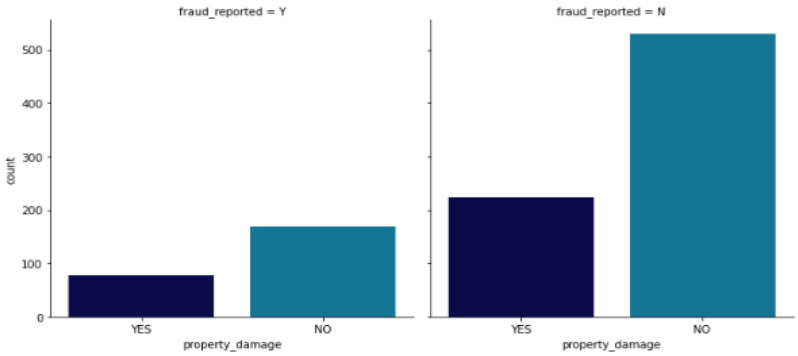
The state SC has high fraud reports compared to other states.

```
# Comparing incident_city and fraud_reported
sns.catplot('incident_city',kind='count',data=df,hue='fraud_reported',palette="bright")
plt.xticks(rotation=90)
plt.show()
```



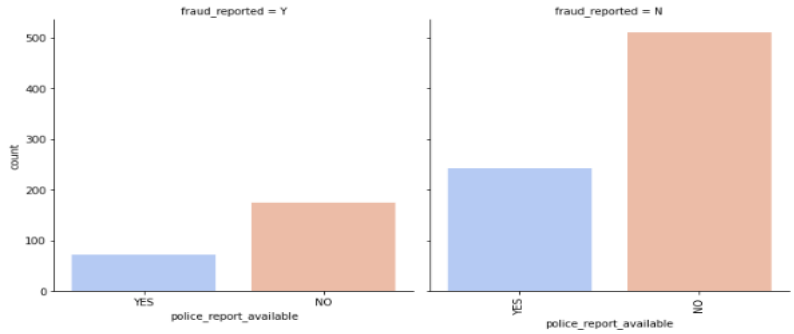
The cities Riverwood and Northbrook have very less fraud reports compared to others.

```
# Comparing property_damage and fraud_reported
sns.factorplot('property_damage',kind='count',data=df,col='fraud_reported',palette="ocean")
plt.show()
```



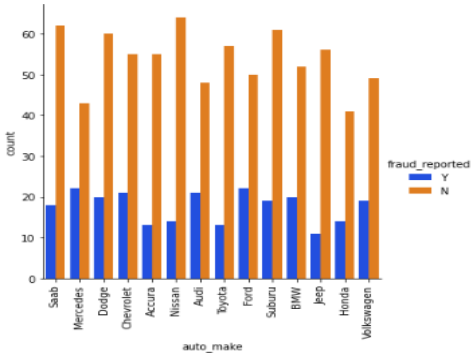
The customers who do not have any property damage case they have high fraud reports.

```
# Comparing police_report_available and fraud_reported
sns.factorplot('police_report_available',kind='count',data=df,col='fraud_reported',palette="coolwarm")
plt.xticks(rotation=90)
plt.show()
```



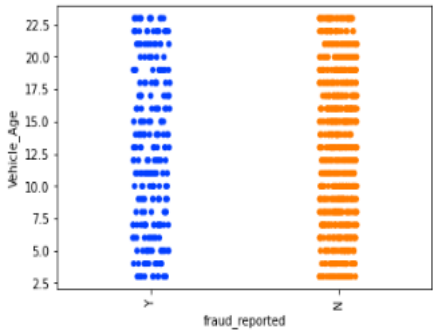
If there is no police report available then the fraud report is very high.

```
# Comparing auto_make and fraud_reported
sns.catplot('auto_make',kind='count',data=df,hue='fraud_reported',palette="bright")
plt.xticks(rotation=90)
plt.show()
```



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

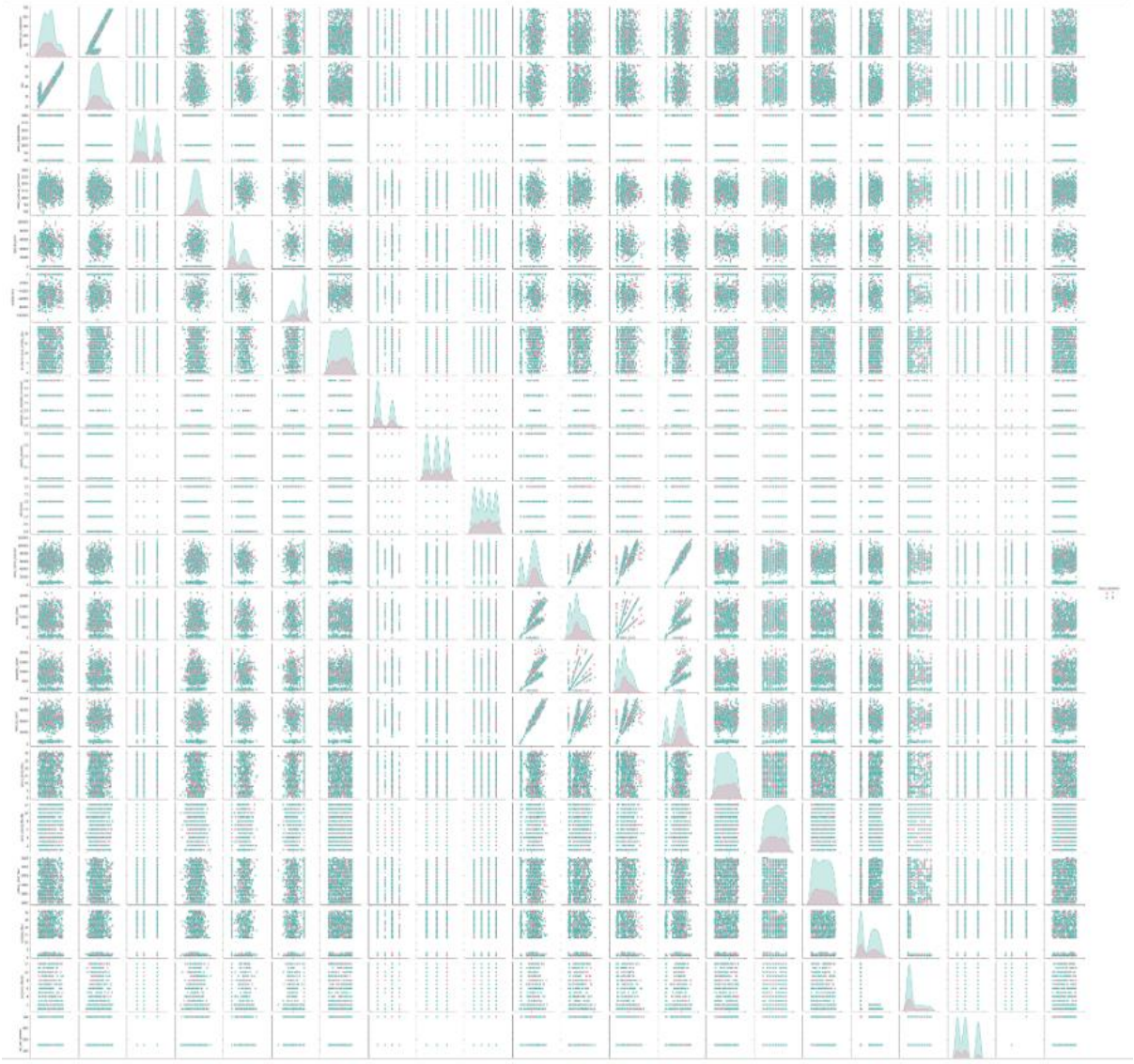
```
# Comparing  Vehicle_Age and fraud_reported
sns.stripplot(y='Vehicle_Age',x='fraud_reported',data=df,palette="bright")
plt.xticks(rotation=90)
plt.show()
```



There is no significant difference between the features.

❖ Multivariate Analysis:

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



Observations:

- The pairplot gives the pairwise relation between the features on the basis of the target "fraud_reported". On the diagonal we can notice the distribution plots.
- From the pair plot we can observe some of the features have strong correlation with each other.
- We can also find some outliers presentin the data, we will remove them using either Zscore or IQR method.

Identifying the outliers:

```
# Let's check the outliers by ploting box plot

plt.figure(figsize=(25,35),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=23:
        ax=plt.subplot(6,4,plotnumber)
        sns.boxplot(df[column],palette="Set2_r")
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```

From the above box plot we can see the outliers in the following columns:

1. age
2. policy_annual_premium
3. total_claim_amount
4. property_claim
5. incident_month

These are the numerical columns which contains outliers.

Let's remove outliers in these columns using either Zscore method or IQR method.

Removing outliers:

1. Zscore method-

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

# Using zscore to remove outliers
from scipy.stats import zscore

z=np.abs(zscore(features))

z
```

Now we have removed the outliers.

```
# 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.

```
# Shape of original dataset
df.shape
```

```
(1000, 39)
```

Before removing outliers we had 1000 rows and 39 columns.

```
# Shape of new dataframe
new_df.shape
```

```
(996, 39)
```

After removing outliers there are 996 rows and 39 columns.

```
# Checking the the data loss
data_loss = (1000-996)/1000*100
data_loss
```

```
0.4
```

Here we are losing very less data hence removing outliers. Let's remove the outliers and check data loss using IQR method.

2. IQR (Inter Quartile Range) method-

```
# 1st quantile
Q1=features.quantile(0.25)

# 3rd quantile
Q3=features.quantile(0.75)

# IQR
IQR=Q3 - Q1

df1=df[~((df < (Q1 - 1.5 * IQR)) |(df > (Q3 + 1.5 * IQR))).any(axis=1)]
```

```
df1.shape
```

```
(942, 39)
```

Using IQR method the dataframe has 942 rows and 39 columns.

```
# Checking the the data loss
data_loss = (1000-942)/1000*100
data_loss
```

```
5.800000000000001
```

Using IQR method also the data loss is very less. But compared to Zscore the data loss is high in IQR, so let's consider Zscore method only.

Checking skewness in the data:

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

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

The following features contains the skewness:

1. total_claim_amount
2. vehicle_claim
3. incident_Month
4. csl_per_accident

Removing Skewness using yeo-johnson method:

```
# Removing skewness using yeo-johnson  method to get better prediction
skew = ["total_claim_amount","vehicle_claim","incident_Month","csl_per_accident"]

from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
'''
parameters:
method = 'box-cox' or 'yeo-johnson'
'''
```

```
"\nparameters:\nmethod = 'box-cox' or 'yeo-johnson'\n"
```

```
new_df[skew] = scaler.fit_transform(new_df[skew].values)
new_df[skew].head()
```

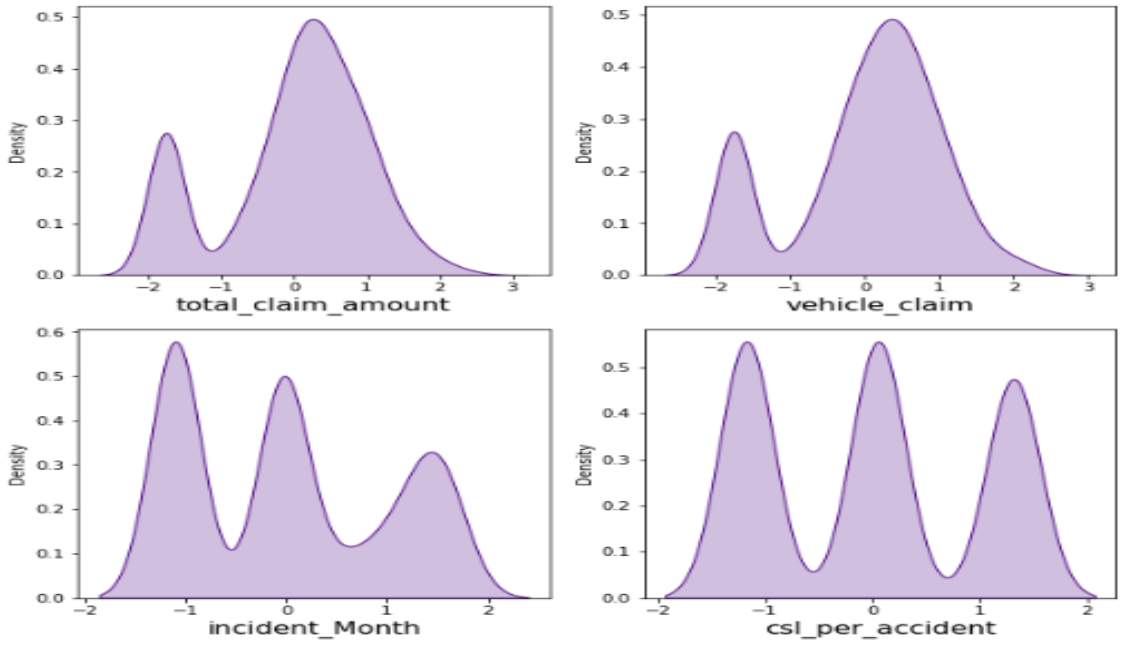
	total_claim_amount	vehicle_claim	incident_Month	csl_per_accident
0	0.717556	0.754553	-1.101370	0.052612
1	-1.777785	-1.787353	-1.101370	0.052612
2	-0.716483	-0.820820	-0.026479	-1.174021
3	0.392931	0.678427	1.553647	0.052612
4	-1.730555	-1.740710	-0.026479	1.313327

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

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

The skewness is almost reduced in all the columns.

```
# 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()
```



The data looks almost normal after removing the skewness compared to the previous data.

Encoding the categorical columns using Label Encoding:

```
from sklearn.preprocessing import LabelEncoder

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

Encoded the categorical columns using label encoder.

```
new_df[categorical_col].head()
```

	policy_state	insured_sex	insured_education_level	insured_occupation	insured_hobbies	insured_relationship	incident_type	collision_type	incident_severity	authorities_conta
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	

The categorical columns have been converted into numerical columns by using label encoding.

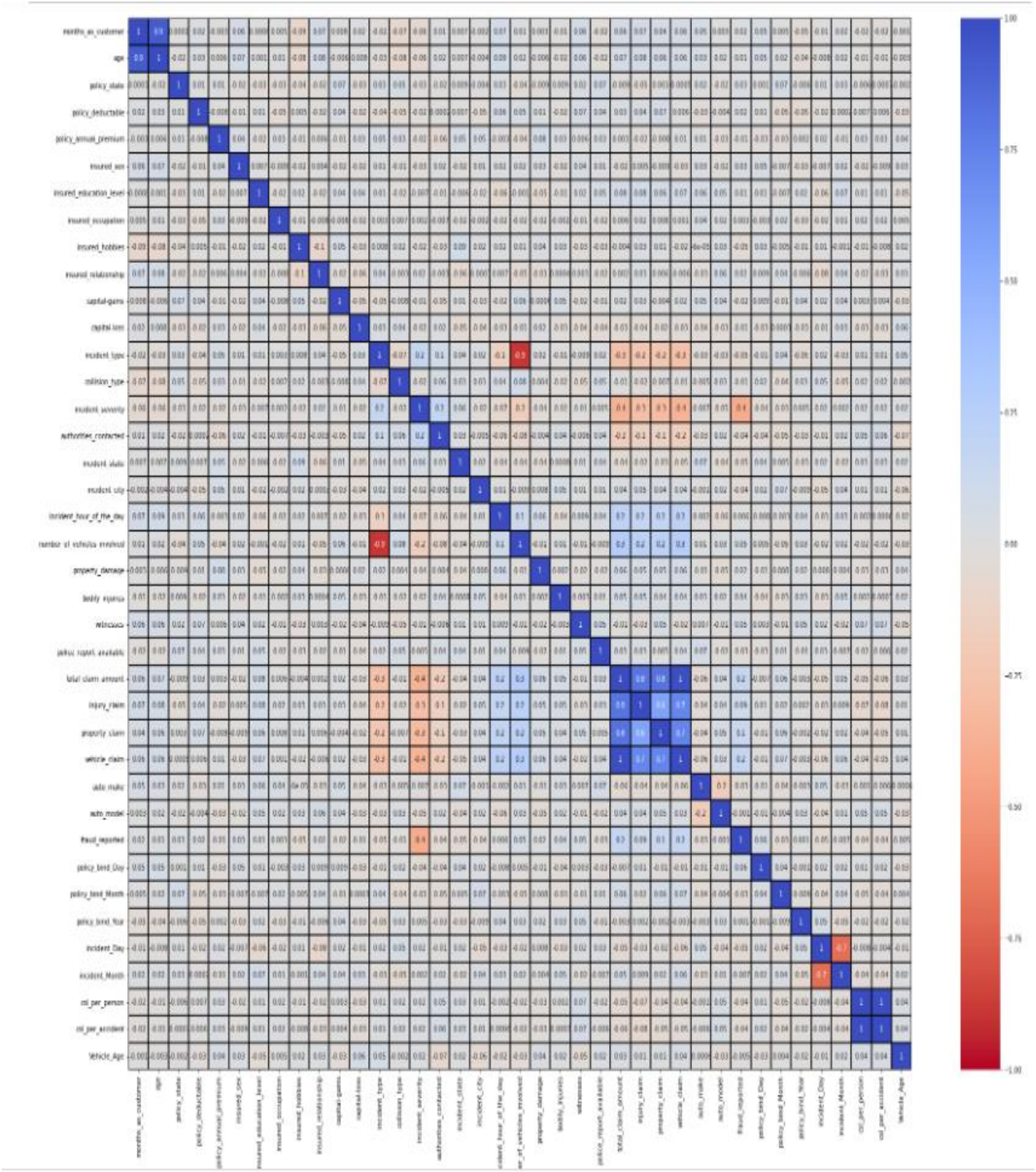
Correlation between the target variable and independent variables using HEAT map:

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

This gives the correlation between the dependent and independent variables. We can visualize this by plotting heat map.

```
# Visualizing the correlation matrix by plotting heat map.
plt.figure(figsize=(30,25))
sns.heatmap(new_df.corr(),linewidths=.1,vmin=-1, vmax=1, fmt='.1g',linecolor="black", annot = True, annot_kws={'size':10},cmap="coolwarm_r")
plt.yticks(rotation=0);
```

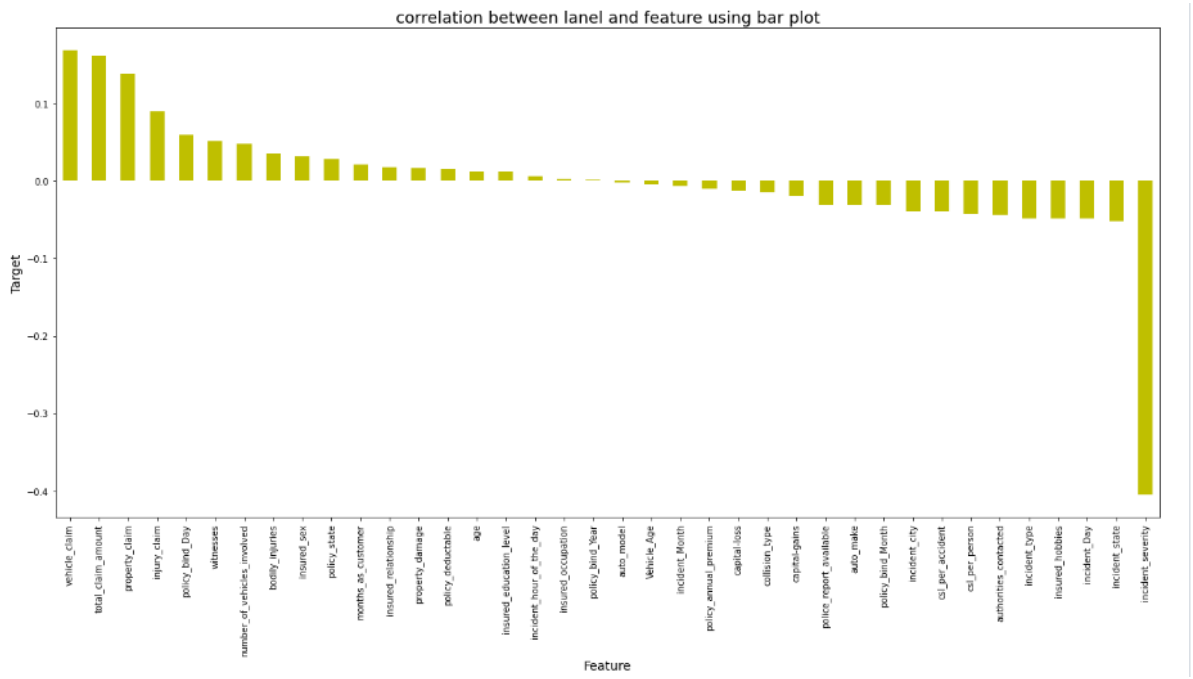
This heat map shows the correlation matrix by visualizing the data. We can observe the relation between one features to other.



This heat map contains both positive and negative correlation.

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

Visualizing the correlation between label and features using bar plot:



From the bar plot we can observe that the columns policy_bind_Year, insured_occupation and auto_model age are very less correlated with the target. We can drop this columns if necessary.

4. PREPROCESSING PIPELINES

Separating the features and label variables into x and y

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

We have separated both dependent and independent variables.

```
# Dimension of x
x.shape
```

```
(996, 38)
```

```
# Dimension of y
y.shape
```

```
(996,)
```

Feature Scaling using Standard Scalarization:

```
from sklearn.preprocessing import StandardScaler

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

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	insured_sex	insured_education_level	insured_occupation	insured_hobbies	insured_relatic
0	1.074671	0.987190	1.186130	-0.224722	0.621371	1.075102	0.531088	-1.162296	1.280299	-1.4
1	0.204846	0.330455	-0.018137	1.409024	-0.251375	1.075102	0.531088	-0.166257	0.928186	-0.2
2	-0.612790	-1.092470	1.186130	1.409024	0.647301	-0.930144	1.557206	1.078792	-1.360550	0.3
3	0.448397	0.221000	-1.222403	1.409024	0.658123	-0.930144	1.557206	-1.411305	-1.360550	0.9
4	0.204846	0.549367	-1.222403	-0.224722	1.358059	1.075102	-1.521148	1.078792	-1.360550	0.9

I have scaled the data using standard scalarizaion method to overcome with the issue of data biasness.

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

Checking Variance Inflation Factor (VIF):

```
# Finding varience inflation factor in each scaled column i.e, x.shape[1] (1/(1-R2))

from statsmodels.stats.outliers_influence import variance_inflation_factor
vif = pd.DataFrame()
vif["VIF values"] = [variance_inflation_factor(x.values,i)
                     for i in range(len(x.columns))]
vif["Features"] = x.columns

# Let's check the values
vif
```

We can observe some columns have VIF above 10 that means they are causing multicollinearity problem.

1. total_claim amount
2. injury_claim
3. property_claim
4. vehicle_claim
5. csl_per_person
6. csl_per_accident

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)
```

Again checking VIF value to confirm whether the multicollinearity still exists or not

```
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(x.values, i)
                     for i in range(x.shape[1])]
vif["Features"]=x.columns
vif
```

Still we have high VIF value in csl_per_person and csl_per_accident. Let's remove csl_per_accident column as it has VIF bit greater than csl_per_person.

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

```
# Again checking VIF value to confirm whether the multicollinearity still exists or not
vif=pd.DataFrame()
vif["vif_Features"]=[variance_inflation_factor(x.values, i)
                     for i in range(x.shape[1])]
vif["Features"]=x.columns
vif
```

Now we are overcome with the multicollinearity issue as the VIF values are less than 10 in all the columns.

```
y.value_counts()
```

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

Here we can observe the data is not balanced, since it is classification problem we will balance the data using oversampling method.

Oversampling:

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

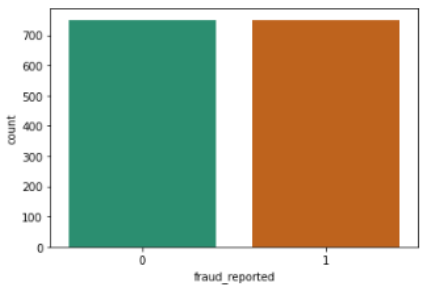
```
# Checking value count of target column
y.value_counts()
```

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

Now our data is balanced, so we can build our models.

```
# Visualizing the data after oversampling
sns.countplot(y,palette="Dark2")
```

```
<AxesSubplot:xlabel='fraud_reported', ylabel='count'>
```



We can clearly observe the balanced data now.

5. BUILDING MACHINE LEARNING MODELS

Finding best random state:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
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)
    DTC = RandomForestClassifier()
    DTC.fit(x_train, y_train)
    pred = DTC.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  0.9155555555555556  on Random_state  9
```

We have got the best random state as 9 and maximum accuracy as 91.55%.

Creating train_test _split:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=maxRS)
```

Classification Algorithms:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier
from xgboost import XGBClassifier as xgb
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, accuracy_score
from sklearn.model_selection import cross_val_score
```

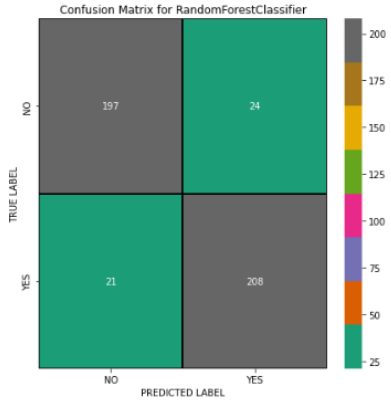
Random Forest Classifier

```
# Checking accuracy for Random Forest Classifier
RFC = RandomForestClassifier()
RFC.fit(x_train,y_train)

# Prediction
predRFC = RFC.predict(x_test)
rfc=accuracy_score(y_test, predRFC)
print(rfc)
print(confusion_matrix(y_test, predRFC))
print(classification_report(y_test,predRFC))
```

```
0.9
[[197  24]
 [ 21 208]]
              precision    recall  f1-score   support

           0       0.90      0.89      0.90       221
           1       0.90      0.91      0.90       229

    accuracy                           0.90       450
   macro avg       0.90      0.90      0.90       450
weighted avg       0.90      0.90      0.90       450
```

The accuracy using Random Forest Classifier is 90%.

```
# Lets plot confusion matrix for RandomForestClassifier
cm = confusion_matrix(y_test,predRFC)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Dark2",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for RandomForestClassifier')
plt.show()
```



With the help of confusion matrix we can able to observe the true positive rate, false positive rate, true negative rate and false negative rate. And is plotted predicted value against True values.

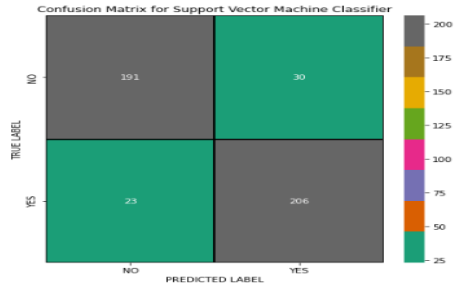
Support Vector Machine Classifier

```
# Checking accuracy for Support Vector Machine Classifier
svc = SVC()
svc.fit(x_train,y_train)

# Prediction
predsvc = svc.predict(x_test)
sv=accuracy_score(y_test, predsvc)
print(sv)
print(confusion_matrix(y_test, predsvc))
print(classification_report(y_test,predsvc))
```

```
0.8822222222222222
[[191  30]
 [ 23 206]]
              precision    recall  f1-score   support

           0       0.89      0.86      0.88       221
           1       0.87      0.90      0.89       229

    accuracy                           0.88       450
   macro avg       0.88      0.88      0.88       450
weighted avg       0.88      0.88      0.88       450
```

The accuracy score using Support Vector Machine classifier is 88.22%.

```
# Lets plot confusion matrix for Support Vector Machine Classifier
cm = confusion_matrix(y_test,predsvc)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Dark2",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for Support Vector Machine Classifier')
plt.show()
```



With the help of confusion matrix we can able to observe the true positive rate, false positive rate, true negative rate and false negative rate. And is plotted predicted value against True values.

Gradient Boosting Classifier

```
# Checking accuracy for Gradient Boosting Classifier
GB = GradientBoostingClassifier()
GB.fit(x_train,y_train)

# Prediction
predGB = GB.predict(x_test)

gb=accuracy_score(y_test, predGB)
print(gb)
print(confusion_matrix(y_test, predGB))
print(classification_report(y_test,predGB))
```

```
0.9111111111111111
[[196  25]
 [ 15 214]]
              precision    recall  f1-score   support

           0       0.93      0.89      0.91       221
           1       0.90      0.93      0.91       229

    accuracy                           0.91       450
   macro avg       0.91      0.91      0.91       450
weighted avg       0.91      0.91      0.91       450
```

The accuracy using Gradient Boosting Classifier is 91.11%.

```
# Lets plot confusion matrix for Gradient Boosting Classifier
cm = confusion_matrix(y_test,predGB)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Dark2",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for Gradient Boosting Classifier')
plt.show()
```

With the help of confusion matrix we can able to observe the true positive rate, false positive rate, true negative rate and false negative rate. And is plotted predicted value against True values.

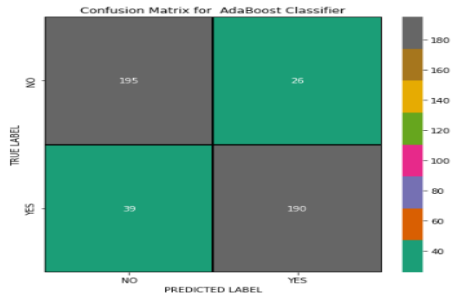
AdaBoost Classifier

```
# Checking accuracy for AdaBoost Classifier
ABC = AdaBoostClassifier()
ABC.fit(x_train,y_train)

# Prediction
predABC = ABC.predict(x_test)

abc=accuracy_score(y_test, predABC)
print(abc)
print(confusion_matrix(y_test, predABC))
print(classification_report(y_test,predABC))
```

```
0.8555555555555555
[[195  26]
 [ 39 190]]
              precision    recall  f1-score   support

           0       0.83      0.88      0.86       221
           1       0.88      0.83      0.85       229

    accuracy                           0.86       450
   macro avg       0.86      0.86      0.86       450
weighted avg       0.86      0.86      0.86       450
```

The accuracy using AdaBoost Classifier is 85.55%.

```
# Lets plot confusion matrix for  AdaBoost Classifier
cm = confusion_matrix(y_test,predABC)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Dark2",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for  AdaBoost Classifier')
plt.show()
```



With the help of confusion matrix we can able to observe the true positive rate, false positive rate, true negative rate and false negative rate. And is plotted predicted value against True values.

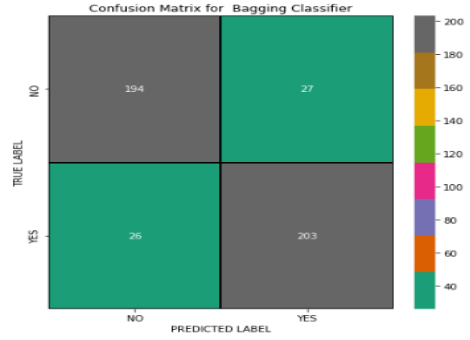
Bagging Classifier

```
# Checking accuracy for BaggingClassifier
BC = BaggingClassifier()
BC.fit(x_train,y_train)

# Prediction
predBC = BC.predict(x_test)
bc=accuracy_score(y_test, predBC)
print(bc)
print(confusion_matrix(y_test, predBC))
print(classification_report(y_test,predBC))
```

```
0.8822222222222222
[[194  27]
 [ 26 203]]
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       221
           1       0.88      0.89      0.88       229

    accuracy                           0.88       450
   macro avg       0.88      0.88      0.88       450
weighted avg       0.88      0.88      0.88       450
```

The accuracy using Bagging classifier is 88.22%.

```
# Lets plot confusion matrix for  Bagging Classifier
cm = confusion_matrix(y_test,predBC)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Dark2",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for  Bagging Classifier')
plt.show()
```



With the help of confusion matrix we can able to observe the true positive rate, false positive rate, true negative rate and false negative rate. And is plotted predicted value against True values.

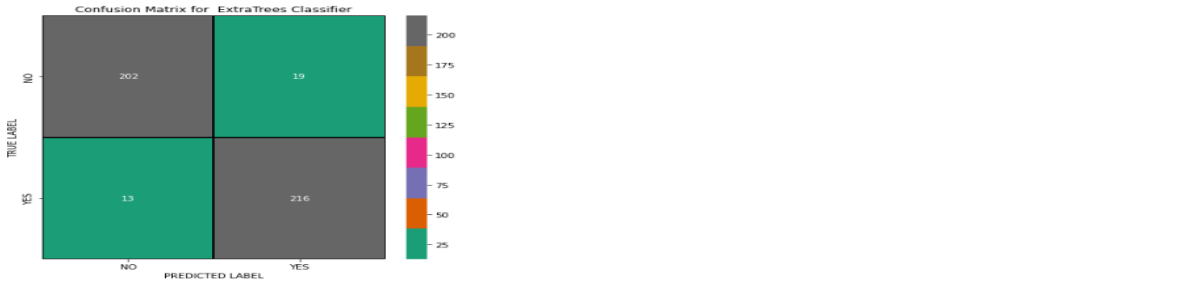
Extra Trees Classifier

```
# Checking accuracy for ExtraTreesClassifier
XT = ExtraTreesClassifier()
XT.fit(x_train,y_train)

# Prediction
predXT = XT.predict(x_test)

etc=accuracy_score(y_test, predXT)
print(etc)
print(confusion_matrix(y_test, predXT))
print(classification_report(y_test,predXT))
```

```
0.9288888888888889
[[202  19]
 [ 13 216]]
              precision    recall  f1-score   support

           0       0.94      0.91      0.93       221
           1       0.92      0.94      0.93       229

    accuracy                           0.93       450
   macro avg       0.93      0.93      0.93       450
weighted avg       0.93      0.93      0.93       450
```

The accuracy using Extra Trees Classifier is 92.88%.

```
# Lets plot confusion matrix for  ExtraTreesClassifier
cm = confusion_matrix(y_test,predXT)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Dark2",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for  ExtraTrees Classifier')
plt.show()
```



With the help of confusion matrix we can able to observe the true positive rate, false positive rate, true negative rate and false negative rate. And is plotted predicted value against True values.

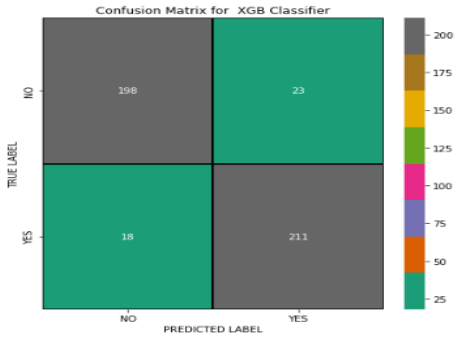
XGB Classifier

```
# Checking accuracy for XGBClassifier
XGB = xgb(verbosity=0)
XGB.fit(x_train,y_train)

# Prediction
predXGB = XGB.predict(x_test)
xgb1=accuracy_score(y_test, predXGB)
print(xgb1)
print(confusion_matrix(y_test, predXGB))
print(classification_report(y_test,predXGB))
```

```
0.9088888888888889
[[198  23]
 [ 18 211]]
              precision    recall  f1-score   support

           0       0.92      0.90      0.91       221
           1       0.90      0.92      0.91       229

    accuracy                           0.91       450
   macro avg       0.91      0.91      0.91       450
weighted avg       0.91      0.91      0.91       450
```

The accuracy using XGB classifier is 90.88%.

```
# Lets plot confusion matrix for  XGBClassifier
cm = confusion_matrix(y_test,predXGB)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="Dark2",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for  XGB Classifier')
plt.show()
```



With the help of confusion matrix we can able to observe the true positive rate, false positive rate, true negative rate and false negative rate. And is plotted predicted value against True values.

Checking Cross Validation Score

```
# cv score for Random Forest Classifier
rf=cross_val_score(RFC,x,y,cv=5).mean()
print(rf)
```

```
0.876
```

```
# cv score for Support Vector Machine Classifier
sv_cv=cross_val_score(svc,x,y,cv=5).mean()
print(sv_cv)
```

```
0.858
```

```
# cv score for Gradient Boosting Classifier
gb_cv=cross_val_score(GB,x,y,cv=5).mean()
print(gb_cv)
```

```
0.8713333333333333
```

```
# cv score for AdaBoosting Classifier
ada_cv=cross_val_score(ABC,x,y,cv=5).mean()
print(ada_cv)
```

```
0.8386666666666667
```

```
# cv score for Bagging Classifier
bc_cv=cross_val_score(BC,x,y,cv=5).mean()
print(bc_cv)
```

```
0.876
```

```
# cv score for Extra Trees Classifier
ex_cv=cross_val_score(XT,x,y,cv=5).mean()
print(ex_cv)
```

```
0.9146666666666668
```

```
# cv score for XGB Classifier
xgb_cv=cross_val_score(XGB,x,y,cv=5).mean()
print(xgb_cv)
```

```
0.8766666666666667
```

Making DataFrame of Classifiers:

```
model_list=['Random Forest Classifier','Support Vector Machine Classifier','Gradient Boosting Classifier','AdaBoosting Classifier','Bagging Classifier
accuracyscore=[rfc,sv,gb,abc,bc,etc,xgb1]
```

```
crossval=[rf,sv_cv,gb_cv,ada_cv,bc_cv,ex_cv,xgb_cv]
```

```
models=pd.DataFrame({})
models["Classifier"]=model_list
models["Accuracy_score"]=accuracyscore
models["Cross Validation_Score"]=crossval
```

```
models
```

	Classifier	Accuracy_score	Cross Validation_Score
0	Random Forest Classifier	0.900000	0.876000
1	Support Vector Machine Classifier	0.882222	0.858000
2	Gradient Boosting Classifier	0.911111	0.871333
3	AdaBoosting Classifier	0.855556	0.838667
4	Bagging Classifier	0.882222	0.876000
5	Extra Trees Classifier	0.928889	0.914667
6	XGB Classifier	0.908889	0.876667

Above are the cross validation score for the models.

From the difference between the accuracy score and the cross validation score we can conclude that **ExtraTrees** Classifier as our best fitting model which is giving very less difference compare to other models.

Hyper Parameter Tuning:

```
# ExtraTrees Classifier
from sklearn.model_selection import GridSearchCV

parameters = {'criterion' : ['gini','entropy'],
              'max_features':['aoto','sqrt','log2'],
              'max_depth' : [0, 10, 20],
              'n_jobs' : [-2, -1, 1],
              'n_estimators' : [50,100, 200, 300]}
```

These are the parameters for ExtraTrees classifier.

```
GCV=GridSearchCV(ExtraTreesClassifier(),parameters,cv=5)
```

Running GridSearchCV for ExtraTrees Classifier.

```
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [0, 10, 20],
                         'max_features': ['aoto', 'sqrt', 'log2'],
                         'n_estimators': [50, 100, 200, 300],
                         'n_jobs': [-2, -1, 1]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
GCV.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 20,
 'max_features': 'sqrt',
 'n_estimators': 300,
 'n_jobs': 1}
```

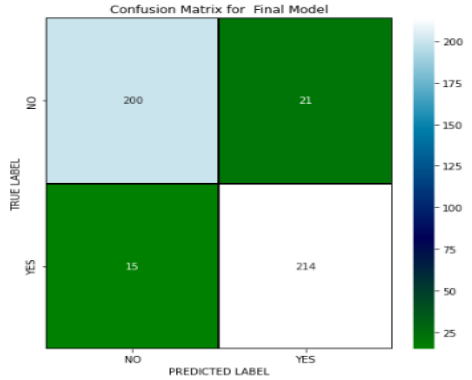
These are the best parameters values that we have got for ExtraTrees classifier.

```
FinalModel = ExtraTreesClassifier(criterion='gini', max_depth=20, max_features='log2', n_estimators=200, n_jobs=-2)
FinalModel.fit(x_train, y_train)
pred = FinalModel.predict(x_test)
acc=accuracy_score(y_test,pred)
print(acc*100)
```

```
93.11111111111111
```

After tuning the model we are getting 93.11% accuracy.

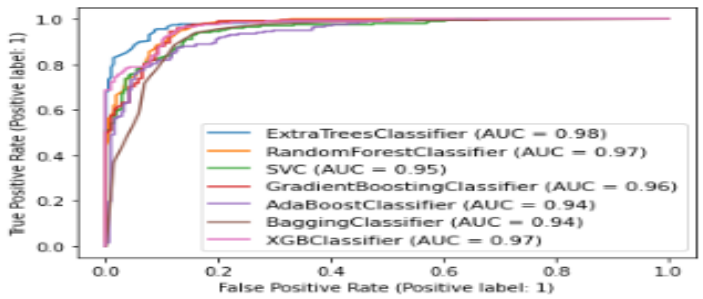
```
# Lets plot confusion matrix for  FinalModel
cm = confusion_matrix(y_test,pred)

x_axis_labels = ["NO","YES"]
y_axis_labels = ["NO","YES"]

f , ax = plt.subplots(figsize=(7,7))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax, cmap="ocean",xticklabels=x_axis_labels,yticklabels=y_axis_labels)

plt.xlabel("PREDICTED LABEL")
plt.ylabel("TRUE LABEL")
plt.title('Confusion Matrix for  Final Model')
plt.show()
```



With the help of confusion matrix we can able to see actual and predicted values.

Plotting ROC and compare AUC for all the models used:

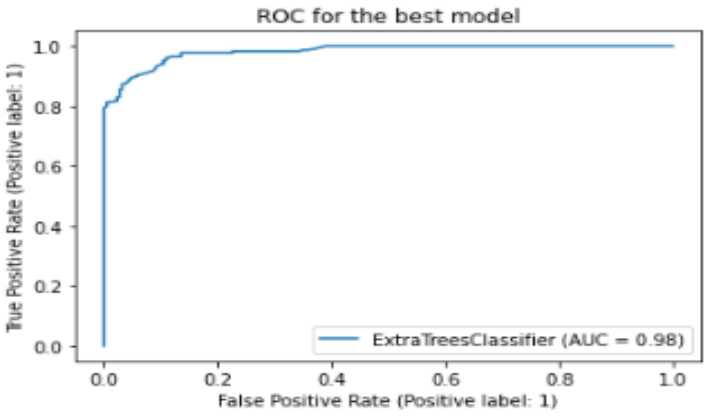
```
# Plotting for all the models used here
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn.metrics import plot_roc_curve


disp = plot_roc_curve(XT,x_test,y_test)      # ax_=Axes with confusion matrix
plot_roc_curve(RFC, x_test, y_test, ax=disp.ax_)
plot_roc_curve(svc, x_test, y_test, ax=disp.ax_)
plot_roc_curve(GB, x_test, y_test, ax=disp.ax_)
plot_roc_curve(ABC, x_test, y_test, ax=disp.ax_)
plot_roc_curve(BC, x_test, y_test, ax=disp.ax_)
plot_roc_curve(XGB, x_test, y_test, ax=disp.ax_)

plt.legend(prop={'size':11}, loc='lower right')
plt.show()
```



Here we can see the Area under curve for each model used here.

Plotting ROC and Compare AUC for the best model:

```
# Let's check the Auc for the best model after hyper parameter tuning
plot_roc_curve(FinalModel, x_test, y_test)
plt.title("ROC for the best model")
plt.show()
```



Here we have plotted the ROC curve for the final model and the AUC value for the best model is 98%.

Saving the Model:

```
# Saving the model using .pkl
import joblib
joblib.dump(FinalModel,"InsuranceclaimFraudDetection.pkl")
```

```
['InsuranceclaimFraudDetection.pkl']
```

We have saved our model using joblib library.

Predicting the saved model:

```
# Let's load the saved model and get the prediction

# Loading the saved model
model=joblib.load("InsuranceclaimFraudDetection.pkl")

#Prediction
prediction = model.predict(x_test)
prediction
```

```
array([1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1,
       1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1,
       1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0,
       0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0,
       1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 1, 1, 1, 0, 1, 0, 0, 1])
```

These are the predicted fraud_reported of the customers.

```
pd.DataFrame([model.predict(x_test)[:],y_test[:]],index=["Predicted","Original"]).T
```

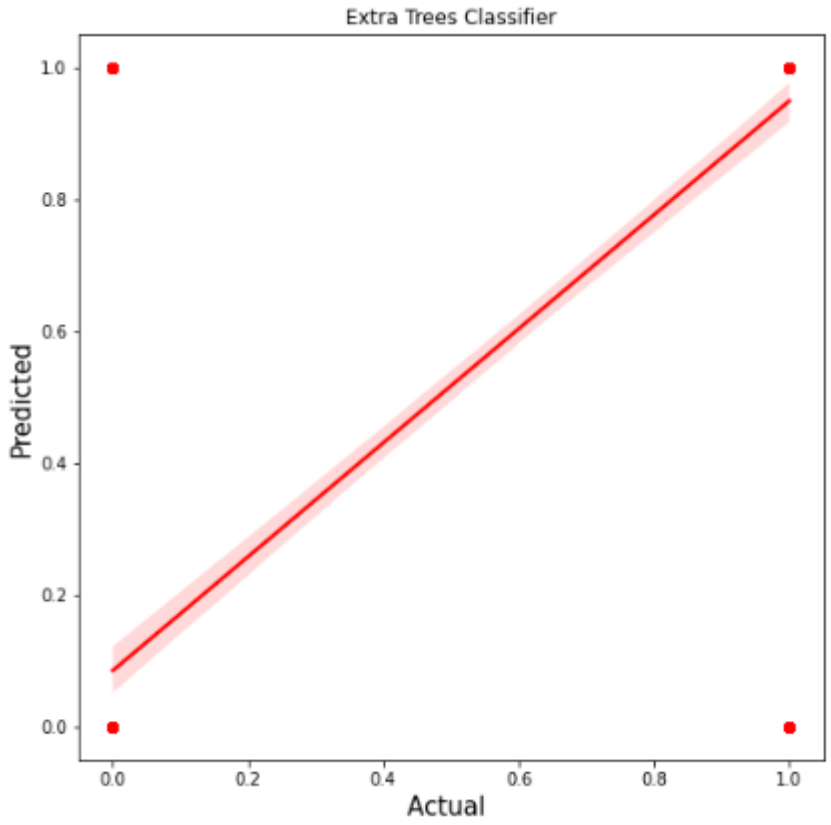
	Predicted	Original
0	0	0
1	1	1
2	0	0
3	1	1
4	1	1
...	...	...
445	1	1
446	0	0
447	0	0
448	0	0
449	0	0

450 rows × 2 columns

Here we can observe that actual and predicted fraud reports are almost same.

Plotting for prediction:

```
plt.figure(figsize=(8,8))
sns.regplot(y_test,prediction,color='r')
plt.xlabel('Actual ',fontsize=15)
plt.ylabel('Predicted',fontsize=15)
plt.title("Extra Trees Classifier")
plt.show()
```



6. CONCLUDING REMARKS

- At the beginning of the blog we have discussed the workflow of a Machine Learning Model, you can see how we have touched base on each point and finally reached up to the model building and made the model ready for deployment.
- This industry area needs a good vision of data, and in every model building problem, Data Analysis and Feature Engineering is the most crucial part.
- You can see how we have handled numerical and categorical data and also how we build different machine learning models on the same dataset.
- Using hyperparameter tuning we have improved our model accuracy, for instance in this model the accuracy remains the same.
- Using this machine Learning Model we humans can easily predict whether the insurance claim is fraudulent or not and we could reject those applications which will be considered fraud claims.

References:

- analyticsjobs.in
- stackoverflow.com
- www.iii.org
- towardsdatascience.com
- google.com