DataTrained
Keep Skilling, Keep Growing

Insurance Claim Fraud Detection

Subham Kundu
BATCH-DS2311

Problem Definition

- **The objective of this project is to build a predictive model that can detect fraud in insurance. The challenge behind machine learning fraud detection is that frauds are much less common compared to legitimate insurance claims. This type of problem is known as Imbalanced class classification.**

- **Dataset:** https://github.com/dsrscientist/Data-Science-ML-Capstone-Projects/blob/master/Automobile_insurance_fraud.csv

INTRODUCTION

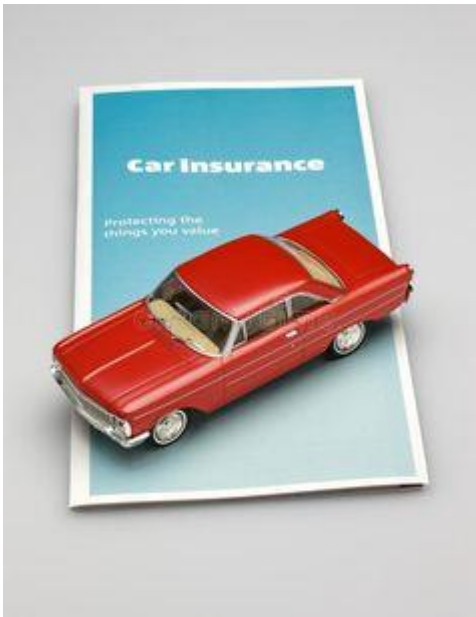
According to the Insurance Information Institute, "Insurance fraud is a deliberate deception perpetrated against or by an insurance company or agent for the purpose of financial gain." Fraud may be committed at different points by applicants, policyholders, third-party claimants, or professionals who 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.

People who commit insurance fraud include:

- Organized criminals who steal large sums through fraudulent business activities,
- Professionals and technicians who inflate service costs or charge for services not rendered, and
- Ordinary people who want to cover their deductible or view filing a claim as an opportunity to make a little money.

Some insurance lines are more vulnerable to fraud than others. Healthcare, workers' compensation, and auto insurance are generally considered to be the sectors most affected.



The auto insurance industry is complicated and involves millions of dollars changing hands every day. And whenever there is a large amount of money running through complex systems, there is opportunity for fraud. This fraud can be committed by professionals and companies working in the industry. But it can also be committed against them.

Insurance fraud can be broadly classified into 2 types.

- Soft Insurance Fraud
- Hard Insurance Fraud

Soft Insurance fraud: An example for this is, if the accident has taken place, but the amount of damage that has happened to the vehicle is very less. In such cases, the individual claims to the insurance company that a huge amount of damage has occurred to the vehicle with the goal of charging the insurance company a higher bill.

Hard Insurance fraud: An example for this is, an individual intentionally plans and invests the loss so that he can claim for the insurance from the company. A common example for this type of fraud is staging a car wreck with the goal of benefitting from the resulting claim.

In the project, we focus on the insurance claim data of an Automobile insurance company. Because of fraudulent claims, insurance companies lose large amounts of money, which indirectly affects the public. Therefore, it is important to know which claims are genuine and which claims are fraud.

In this article, we'll check how to spot insurance fraud and the consequences of engaging in insurance fraud by building machine learning models and getting predictions of which claims are likely to be fraudulent.

PROBLEM DEFINITION

Insurance fraud is a huge 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 are provided with 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, we 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.

The problem statement explains that the target variable "fraud reported" contains the categories, so it is a "Classification Problem", we need to predict whether an insurance claim is fraudulent or not.

DATA ANALYSIS

Data Analysis refers to the process of cleaning, transforming and extracting data to discover useful information for business decision making.

IMPORTING NECESSARY LIBRARIES

We import the libraries necessary for data analysis

```
import pandas as pd # for data wrangling purpose
import numpy as np # Basic computation library
import seaborn as sns # For Visualization
import matplotlib.pyplot as plt # ploting package
%matplotlib inline
import warnings # Filtering warnings
warnings.filterwarnings('ignore')
```

IMPORTING THE DATASET

We import the Dataset

```
# Importing Insurance Claims dataset Csv file using pandas
df=pd.read_csv('Automobile_insurance_fraud.csv')
```

```
print('No of Rows:',df.shape[0])
print('No of Columns:',df.shape[1])
pd.set_option('display.max_columns', None) # This will enable us to see truncated columns
df.head()
```

```
No of Rows: 1000
No of Columns: 40
```


The dataset contains **1000 rows and 40 columns** of numerical & categorical data. Next, we check the **HEAD( ), TAIL( ) & SAMPLE( )** of the dataset. After this we do some Exploratory Data Analysis (EDA) of the given dataset.

DATA PREPARATION & CLEANING

We check the columns present and sort by data types in the dataset:

```
# Sort columns by datatypes
df.columns.to_series().groupby(df.dtypes).groups
```

```
{int64: ['months_as_customer', 'age', 'policy_number', 'policy_deductable', 'umbrella_limit', 'insured_zip', 'capital-gains', 'capital-loss', 'incident_hour_of_the_day', 'number_of_vehicles_involved', 'bodily_injuries', 'witnesses', 'total_claim_amount', 'injury_claim', 'property_claim', 'vehicle_claim', 'auto_year'], float64: ['policy_annual_premium', '_c39'], object: ['policy_bind_date', 'policy_state', 'policy_csl', 'insured_sex', 'insured_education_level', 'insured_occupation', 'insured_hobbies', 'insured_relationship', 'incident_date', 'incident_type', 'collision_type', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_city', 'incident_location', 'property_damage', 'police_report_available', 'auto_make', 'auto_model', 'fraud_reported']}
```

- We will check for null values and the missing values present in the dataset, sum of such null values (if present) in the dataset & a visual heat map of the null values.

Since dataset is large, Let check for any entry which is repeated or duplicated in dataset.

```
df.duplicated('policy_number').sum() # This will check if any duplicate entry or duplicate row with same policy_number
```

```
0
```

Let check if any whitespace, 'NA' or '-' exist in dataset.

```
df.isin([' ','NA','-']).sum().any()
```

```
False
```

```
df.isin(['?']).sum().any()
```

```
True
```

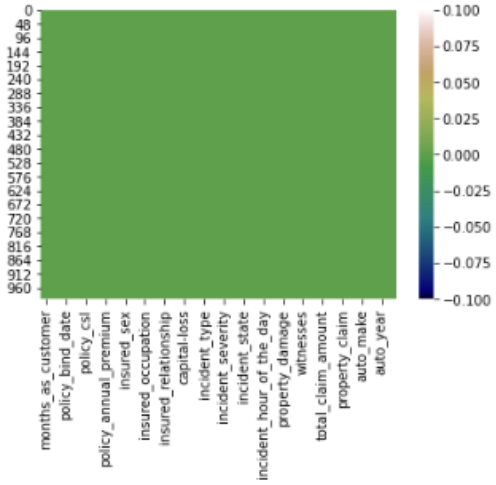
```
#Finding what percentage of data is missing from the dataset
missing_values = df.isnull().sum().sort_values(ascending = False)
percentage_missing_values =(missing_values/len(df))*100
print(pd.concat([missing_values, percentage_missing_values], axis =1, keys =['Missing Values', '% Missing data']))
```

```
                             Missing Values  % Missing data
_c39                                   1000          100.0
property_damage                         360           36.0
police_report_available                 343           34.3
collision_type                          178           17.8
auto_make                                 0            0.0
vehicle_claim                             0            0.0
capital-gains                             0            0.0
insured_relationship                      0            0.0
insured_hobbies                           0            0.0
insured_occupation                        0            0.0
insured_education_level                   0            0.0
insured_sex                               0            0.0
insured_zip                               0            0.0
umbrella_limit                            0            0.0
policy_annual_premium                     0            0.0
policy_deductable                         0            0.0
policy_csl                                0            0.0
policy_state                              0            0.0
policy_bind_date                          0            0.0
policy_number                             0            0.0
age                                       0            0.0
capital-loss                              0            0.0
incident_date                             0            0.0
incident_type                             0            0.0
auto_year                                 0            0.0
property_claim                            0            0.0
injury_claim                              0            0.0
total_claim_amount                        0            0.0
auto_model                                0            0.0
witnesses                                 0            0.0
bodily_injuries                           0            0.0
number_of_vehicles_involved               0            0.0
fraud_reported                            0            0.0
incident_hour_of_the_day                  0            0.0
incident_location                         0            0.0
incident_city                             0            0.0
incident_state                            0            0.0
authorities_contacted                     0            0.0
incident_severity                         0            0.0
months_as_customer                        0            0.0
```

Now we will check the heatmap of the missing values:

Missing value check after imputation

```
# Heatmap of missing value
sns.heatmap(df.isnull(),cmap="gist_earth")
plt.show()
```



Next, we convert date columns from object type into data type:

```
# 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'])
```

```
# 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
```

As incident year is 2015, we will use 2015 as base year for new column creation.

```
# Lets extract age of the vehicle from auto_year by subtracting it from the year 2018
df['Automobile_Age']=2015 - df['auto_year']
# Droping auto year column
df.drop("auto_year",axis=1,inplace=True)
```

Next, we will split feature in categorical and numerical variable.

```
Category = ['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 = ['months_as_customer','CSL_Personal','CSL_Accidental', 'age', 'policy_deductable', 'umbrella_limit', 'capit
             'incident_hour_of_the_day', 'number_of_vehicles_involved', 'bodily_injuries', 'witnesses',
             'total_claim_amount','injury_claim', 'property_claim', 'vehicle_claim', 'Automobile_Age','policy_annual_pr
```

```
df.columns.to_series().groupby(df.dtypes).groups
```

```
{int64: ['months_as_customer', 'age', 'policy_deductable', 'umbrella_limit', 'capital-gains', 'capital-loss', 'incident_
hour_of_the_day', 'number_of_vehicles_involved', 'bodily_injuries', 'witnesses', 'total_claim_amount', 'injury_claim',
'property_claim', 'vehicle_claim', 'policy_bind_day', 'policy_bind_month', 'policy_bind_year', 'incident_day', 'incident
_month', 'Automobile_Age'], float64: ['policy_annual_premium'], object: ['policy_state', 'insured_sex', 'insured_educati
on_level', 'insured_occupation', 'insured_hobbies', 'insured_relationship', 'incident_type', 'collision_type', 'incident
_severity', 'authorities_contacted', 'incident_state', 'incident_city', 'property_damage', 'police_report_available', 'a
uto_make', 'auto_model', 'fraud_reported', 'CSL_Personal', 'CSL_Accidental']}
```

Next, we check the different statistical measurements of all the numerical columns, then specifically our target variable column.

```
df.describe().T.style.background_gradient(subset=['mean','std','50%','count'], cmap='RdPu')
```

	count	mean	std	min	25%	50%	7
months_as_customer	1000.000000	203.954000	115.113174	0.000000	115.750000	199.500000	276.250
age	1000.000000	38.948000	9.140287	19.000000	32.000000	38.000000	44.000
policy_deductable	1000.000000	1136.000000	611.864673	500.000000	500.000000	1000.000000	2000.000
policy_annual_premium	1000.000000	1256.406150	244.167395	433.330000	1089.607500	1257.200000	1415.695
umbrella_limit	1000.000000	1101000.000000	2297406.598118	-1000000.000000	0.000000	0.000000	0.000
capital-gains	1000.000000	25126.100000	27872.187708	0.000000	0.000000	0.000000	51025.000
capital-loss	1000.000000	-26793.700000	28104.096686	-111100.000000	-51500.000000	-23250.000000	0.000
incident_hour_of_the_day	1000.000000	11.644000	6.951373	0.000000	6.000000	12.000000	17.000
number_of_vehicles_involved	1000.000000	1.839000	1.018880	1.000000	1.000000	1.000000	3.000
bodily_injuries	1000.000000	0.992000	0.820127	0.000000	0.000000	1.000000	2.000
witnesses	1000.000000	1.487000	1.111335	0.000000	1.000000	1.000000	2.000
total_claim_amount	1000.000000	52761.940000	26401.533190	100.000000	41812.500000	58055.000000	70592.500
injury_claim	1000.000000	7433.420000	4880.951853	0.000000	4295.000000	6775.000000	11305.000
property_claim	1000.000000	7399.570000	4824.726179	0.000000	4445.000000	6750.000000	10885.000
vehicle_claim	1000.000000	37928.950000	18886.252893	70.000000	30292.500000	42100.000000	50822.500
policy_bind_day	1000.000000	15.448000	8.850176	1.000000	8.000000	16.000000	23.000
policy_bind_month	1000.000000	6.559000	3.391758	1.000000	4.000000	7.000000	9.000
policy_bind_year	1000.000000	2001.604000	7.360391	1990.000000	1995.000000	2002.000000	2008.000
incident_day	1000.000000	13.084000	10.443180	1.000000	2.000000	15.000000	22.000
incident_month	1000.000000	3.407000	3.276291	1.000000	1.000000	2.000000	5.000
Automobile_Age	1000.000000	9.897000	6.015861	0.000000	5.000000	10.000000	15.000

```
df[Category].describe().T.style.background_gradient(cmap='summer_r')
```

	count	unique	top	freq
policy_state	1000	3	OH	352
insured_sex	1000	2	FEMALE	537
insured_education_level	1000	7	JD	161
insured_occupation	1000	14	machine-op-inspct	93
insured_hobbies	1000	20	reading	64
insured_relationship	1000	6	own-child	183
incident_type	1000	4	Multi-vehicle Collision	419
collision_type	1000	3	Rear Collision	470
incident_severity	1000	4	Minor Damage	354
authorities_contacted	1000	5	Police	292
incident_state	1000	7	NY	262
incident_city	1000	7	Springfield	157
property_damage	1000	2	NO	698
police_report_available	1000	2	NO	686
auto_make	1000	14	Suburu	80
auto_model	1000	39	RAM	43
fraud_reported	1000	2	N	753

From the above output we find the following observations:

- Here the counts of all the columns are equal which means there are no missing values in the dataset.
- In the columns “policy deductible”, “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 the right.
- And in the columns “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 the left.
- And in some of the columns the mean and median are equal, which means the data is symmetric and is normally distributed and no skew- ness present.

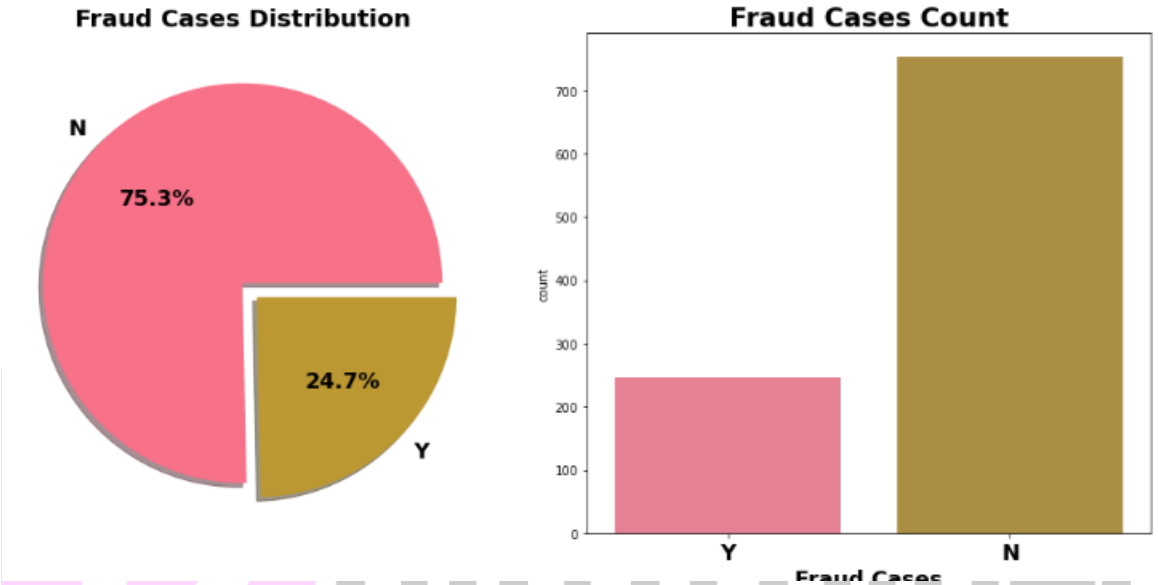
After this, we will start with Enlisting Value counts & Sub-categories of different categorial features available:

```
for i in Category:
    print(i)
    print(df[i].value_counts())
    print("="*100)
```

DATA VISUALISATION

Now we visualize our data. For visualization we will target the variable:

```
plt.rcParams["figure.autolayout"] = True
sns.set_palette('husl')
f,ax=plt.subplots(1,2,figsize=(14,7))
df['fraud_reported'].value_counts().plot.pie(explode=[0,0.1],autopct='%3.1f%%',
                                             textprops ={ 'fontweight': 'bold','fontsize':18}, ax=ax[0],shadow=True)
ax[0].set_title('Fraud Cases Distribution', fontsize=20,fontweight ='bold')
ax[0].set_ylabel('')
sns.countplot('fraud_reported',data=df,ax=ax[1])
ax[1].set_title('Fraud Cases Count',fontsize=22,fontweight ='bold')
ax[1].set_xlabel("Fraud Cases",fontsize=18,fontweight ='bold')
plt.xticks(fontsize=18,fontweight ='bold')
plt.show()
```



Observation:

- Out of all cases around 24.7 % cases are Fraud.
- 'fraud reported' is our target variable to be predicted. From count plot we can say dataset is imbalanced in nature.

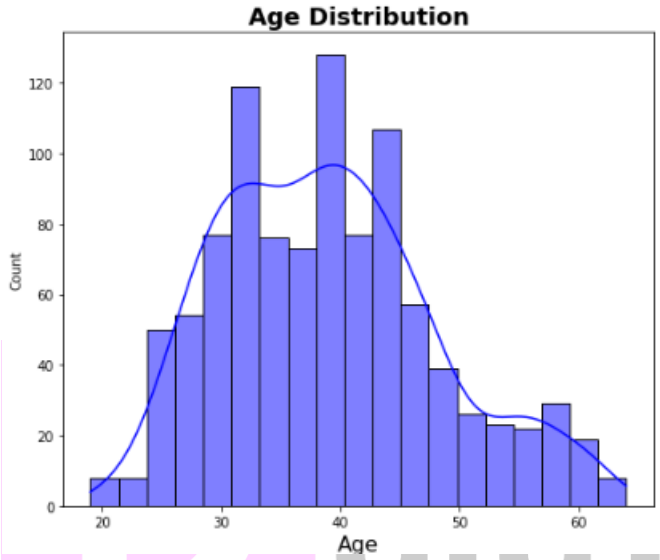
Now, we will start exploring target variable against independent variable to gain more insights.

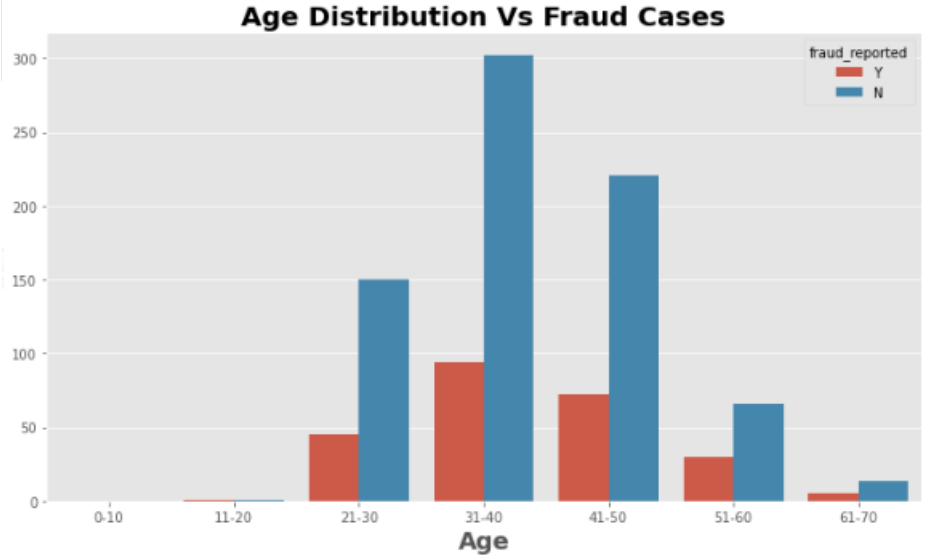
Analysing Age vs Fraud

```
print('Minimum Age :',df.age.min(),'Years')
print('Maximum Age :',df.age.max(),'Years')
print('Average Age :',df.age.mean(),'Years')
```

```
Minimum Age : 19 Years
Maximum Age : 64 Years
Average Age : 38.948 Years
```

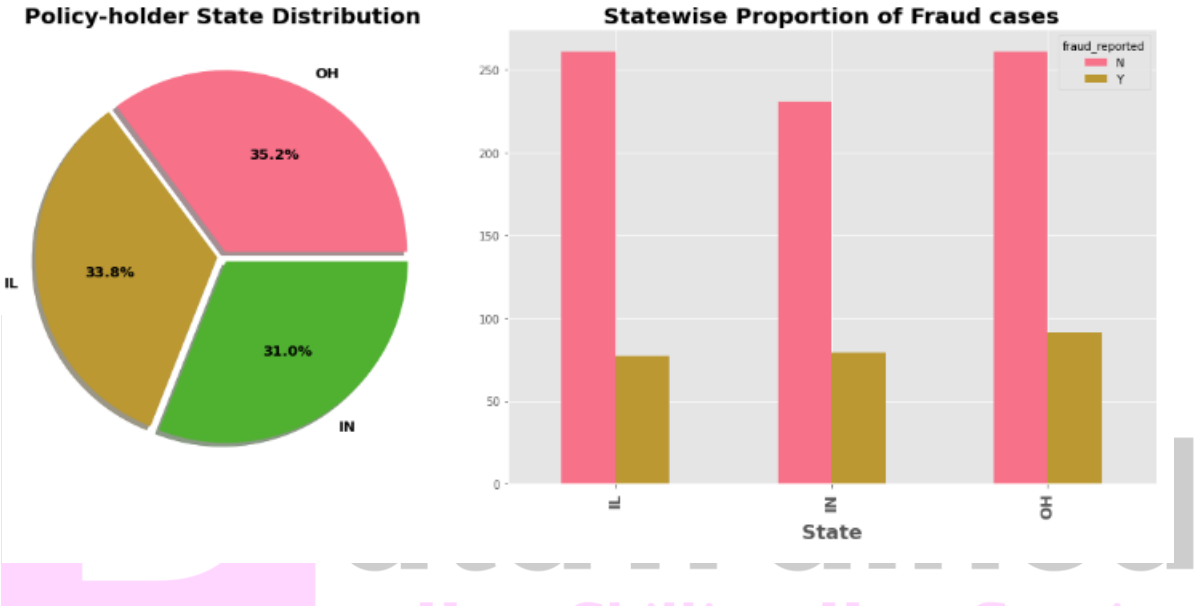
```
plt.figure(figsize=(7,6))
sns.histplot(df.age, kde=True, color='b')
plt.xlabel('Age',fontsize=16)
plt.title('Age Distribution', fontsize=18,fontweight ='bold')
plt.show()
```





Policy State Vs Fraud cases

```
plt.rcParams["figure.autolayout"] = True
sns.set_palette('husl')
fig,ax=plt.subplots(1,2,figsize=(16,7))
df['policy_state'].value_counts().plot.pie(explode=[0.03,0.03,0.03],autopct='%2.1f%%',
                                           textprops ={ 'fontweight': 'bold','fontsize':13}, ax=ax[0],shadow=True)
ax[0].set_title('Policy-holder State Distribution', fontsize=20,fontweight ='bold')
ax[0].set_ylabel('')

table = pd.crosstab(df['policy_state'], df['fraud_reported'])
table.plot(kind = 'bar', ax=ax[1])
ax[1].set_title('Statewise Proportion of Fraud cases',fontsize=20,fontweight ='bold')
ax[1].set_xlabel(" State ",fontsize=18,fontweight ='bold')
plt.xticks(fontsize=14,fontweight ='bold')
plt.tight_layout()
plt.show()
```

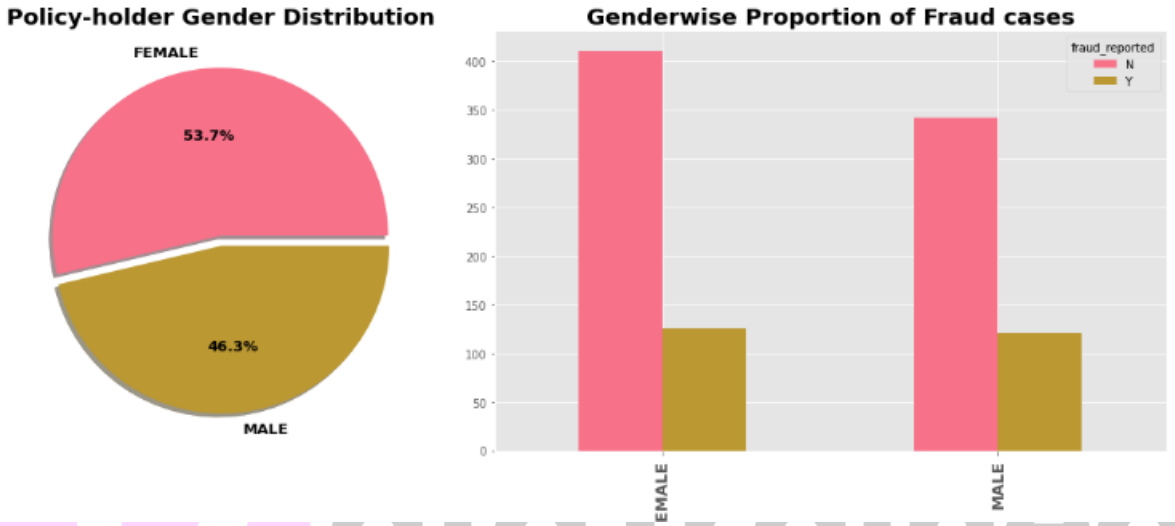


Observation:

- Almost same amount of cases come from each state.
- Maximum fraud cases come from state of Ohio.

Insured Gender VS Fraud cases

```
plt.rcParams["figure.autolayout"] = True
sns.set_palette('husl')
fig,ax=plt.subplots(1,2,figsize=(16,7))
df['insured_sex'].value_counts().plot.pie(explode=[0.03,0.03],autopct='%2.1f%%',
                                          textprops ={ 'fontweight': 'bold','fontsize':13}, ax=ax[0],shadow=True)
ax[0].set_title('Policy-holder Gender Distribution', fontsize=20,fontweight ='bold')
ax[0].set_ylabel('')

table = pd.crosstab(df['insured_sex'], df['fraud_reported'])
table.plot(kind = 'bar', ax=ax[1])
ax[1].set_title('Genderwise Proportion of Fraud cases',fontsize=20,fontweight ='bold')
ax[1].set_xlabel(" Gender ",fontsize=18,fontweight ='bold')
plt.xticks(fontsize=14,fontweight ='bold')
plt.tight_layout()
plt.show()
```

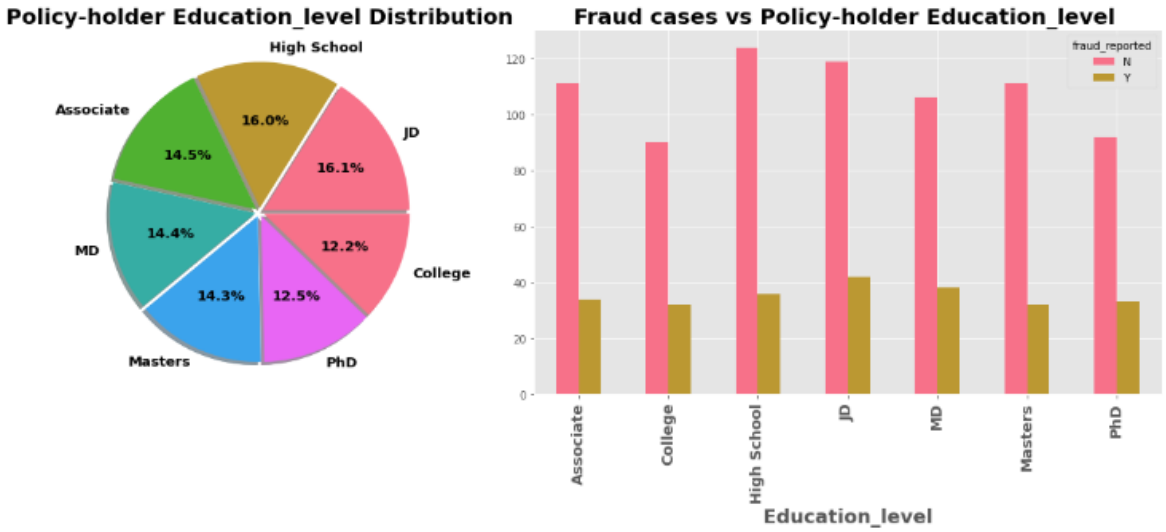


Comment:

- Number of claims come from female is higher than which reported by male insured.
- Almost same amount of fraud cases comes from same gender.

Education_level vs Fraud cases

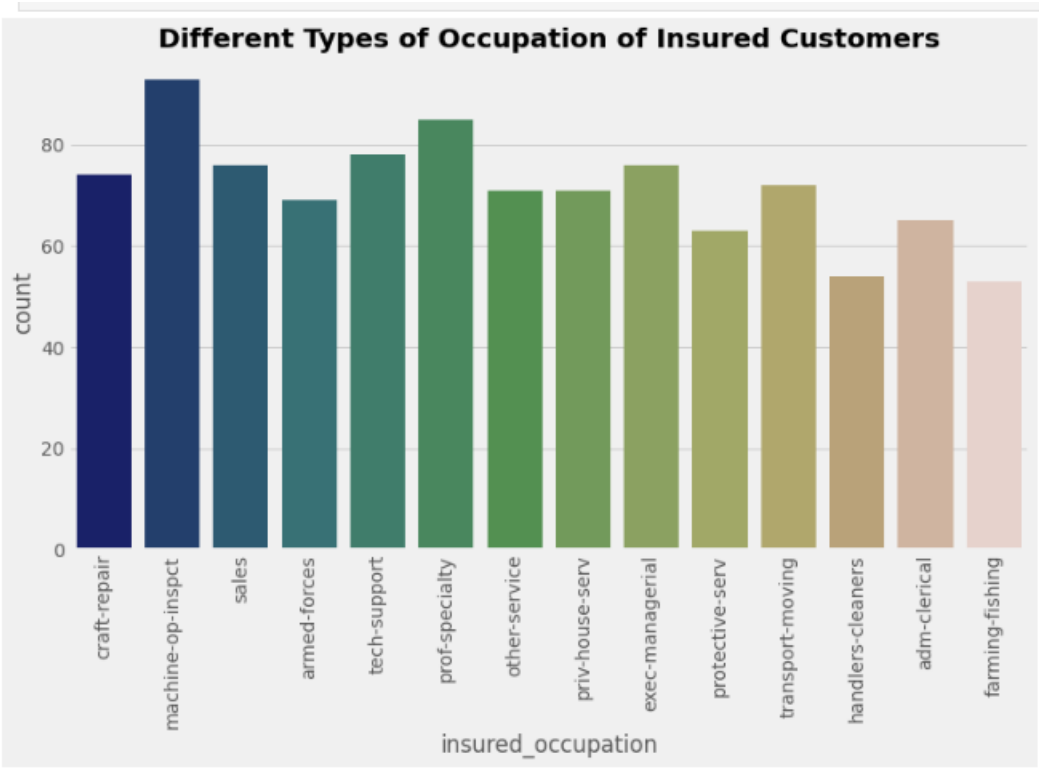
```
plt.rcParams["figure.autolayout"] = True
sns.set_palette('husl')
fig,ax=plt.subplots(1,2,figsize=(16,7))
df['insured_education_level'].value_counts().plot.pie(explode=[0.03,0.03,0.03,0.03,0.03,0.03,0.03,.03],autopct='%2.1f%%',
                                                      textprops ={ 'fontweight': 'bold','fontsize':13}, ax=ax[0],shadow=True)
ax[0].set_title('Policy-holder Education_level Distribution', fontsize=20,fontweight ='bold')
ax[0].set_ylabel('')

table = pd.crosstab(df['insured_education_level'], df['fraud_reported'])
table.plot(kind = 'bar', ax=ax[1])
ax[1].set_title('Fraud cases vs Policy-holder Education_level ',fontsize=20,fontweight ='bold')
ax[1].set_xlabel(" Education_level ",fontsize=18,fontweight ='bold')
plt.xticks(fontsize=14,fontweight ='bold')
plt.tight_layout()
plt.show()
```



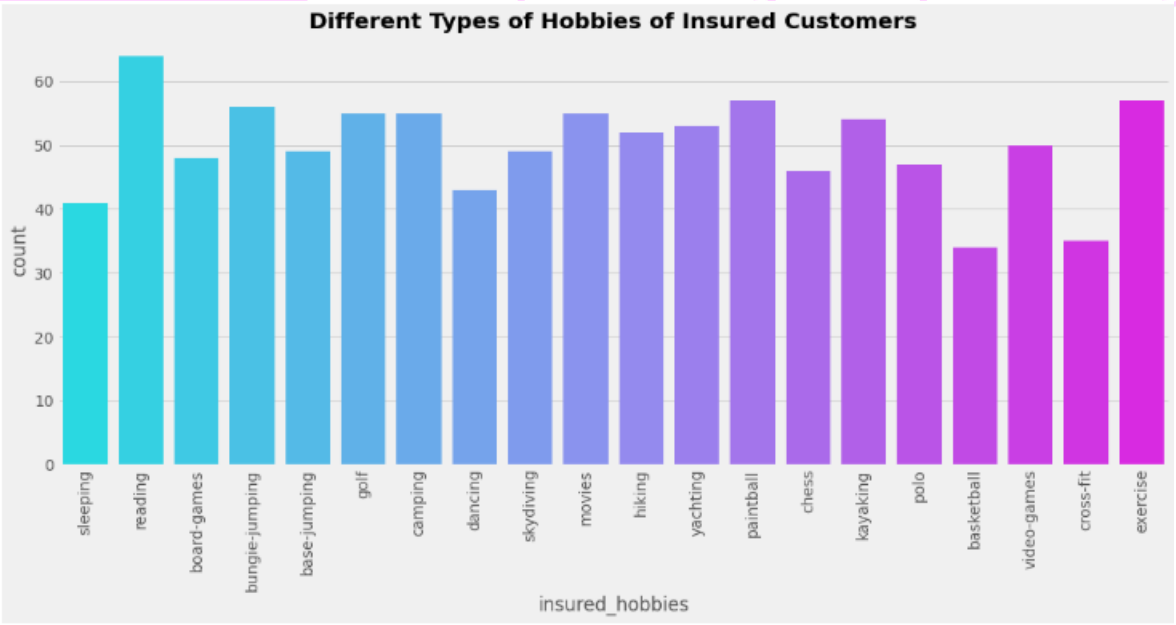
Comment:

We can see tendency to make fraud claims has across every education backgroud, even in Masters,PhD. Education Level is not much important variable for us!!!

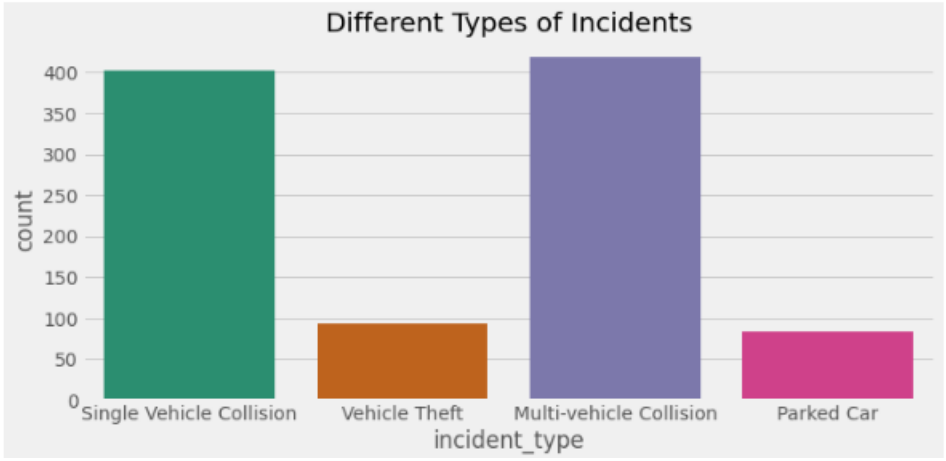
Occupation of Insured Customers:

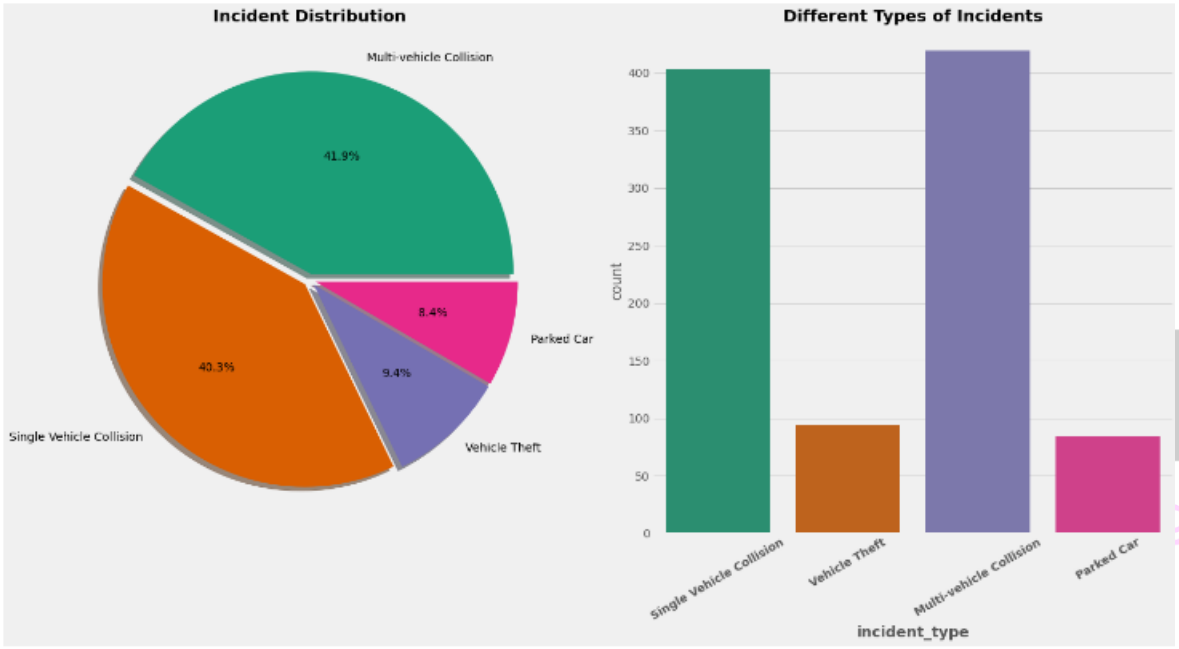


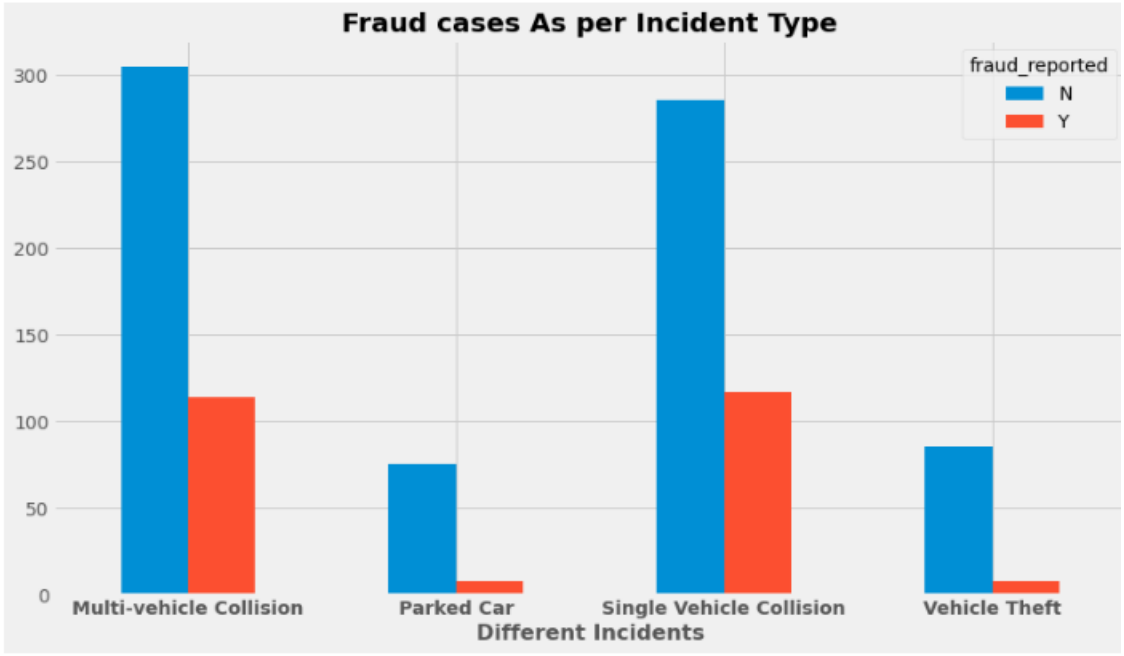
Hobbies of Insured Customers:



Different Types of Incidents Vs Fraud cases:



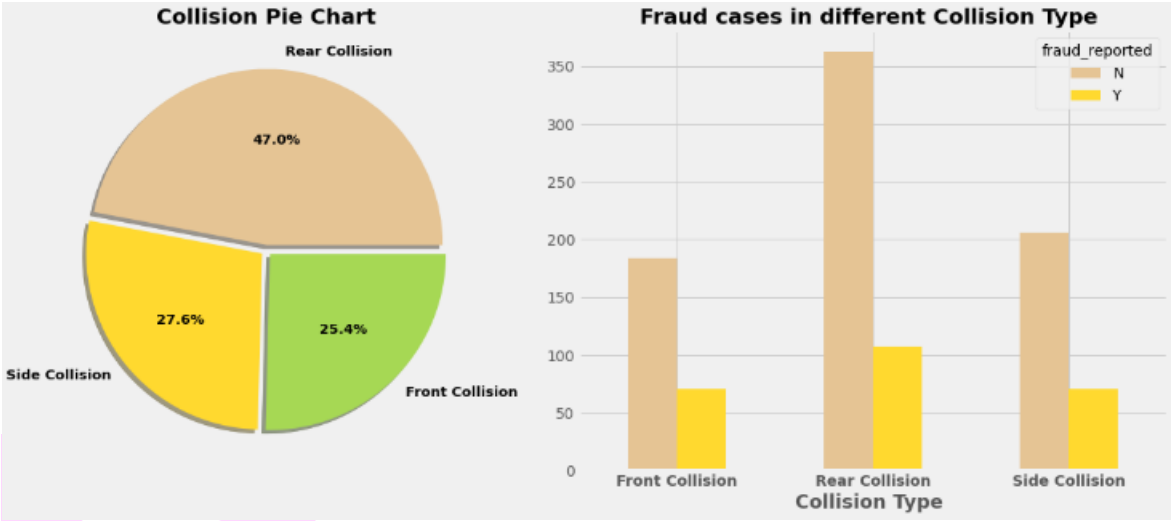


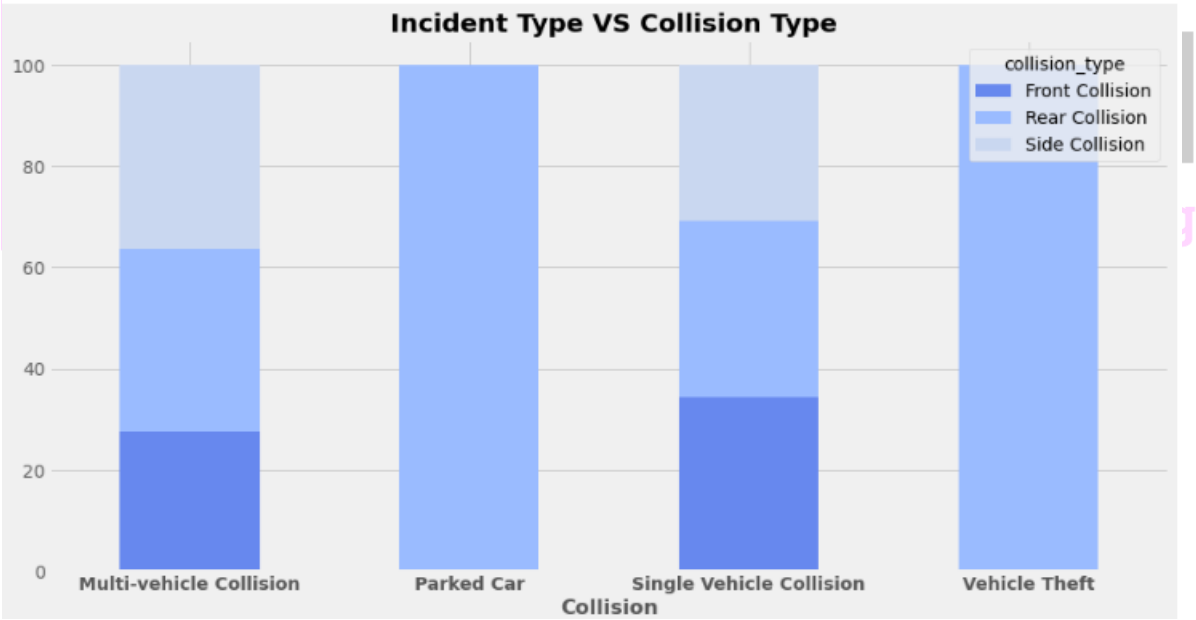


Observation:

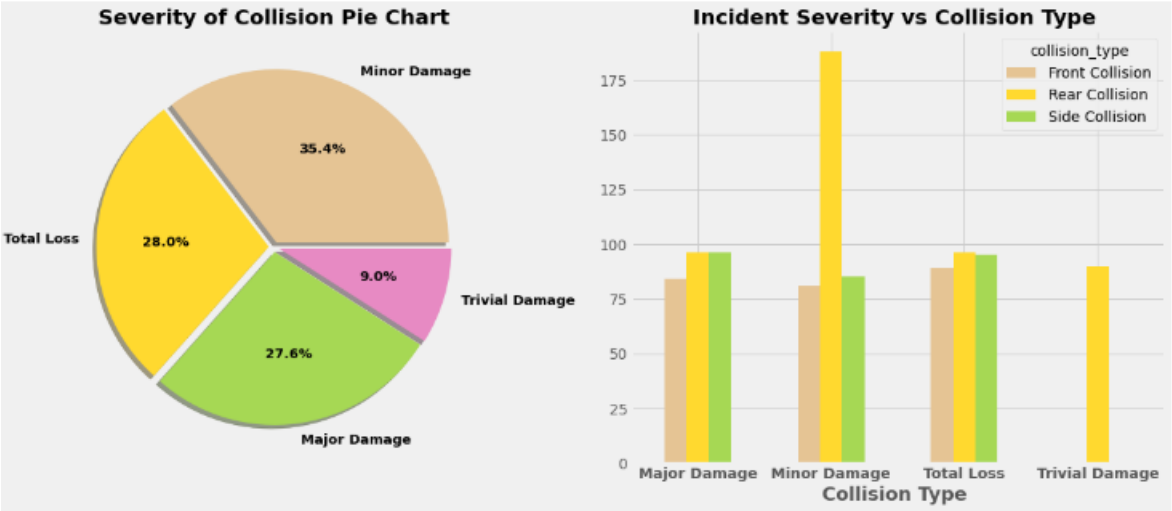
- Most of case comes from Multi-vehicle and single vehicle collision.
- Some claims are due to automobile robbery.
- **One claim out of three claim is fraud in multi or single vehicle collision incident.**

Exploration of different Collision:





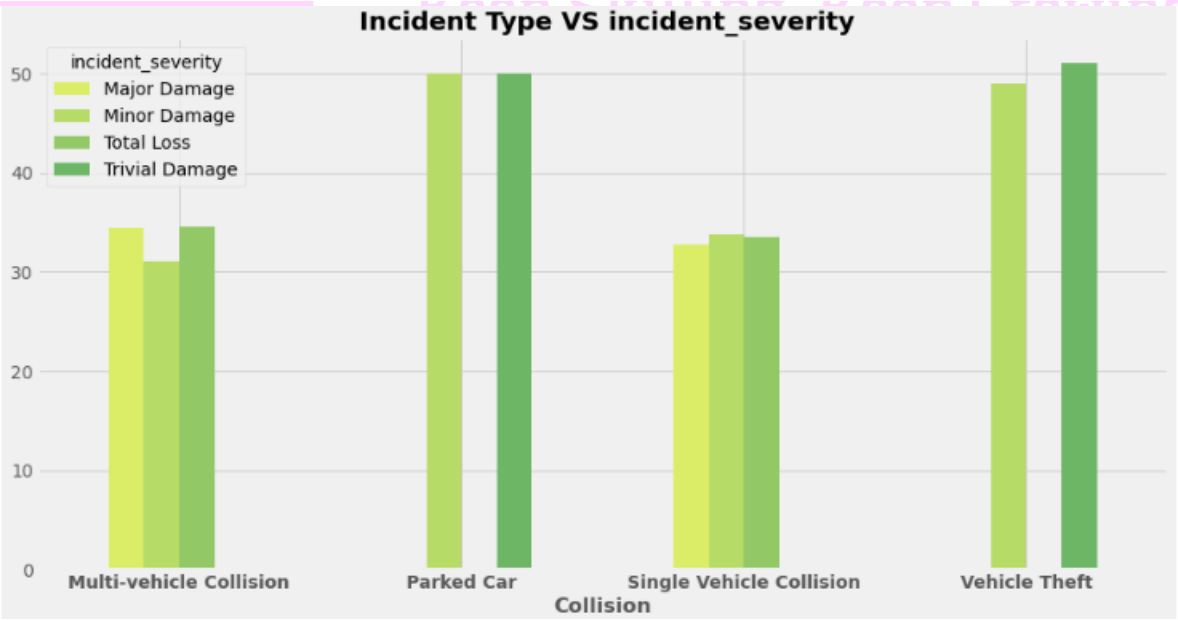
Collision VS Incident Severity:



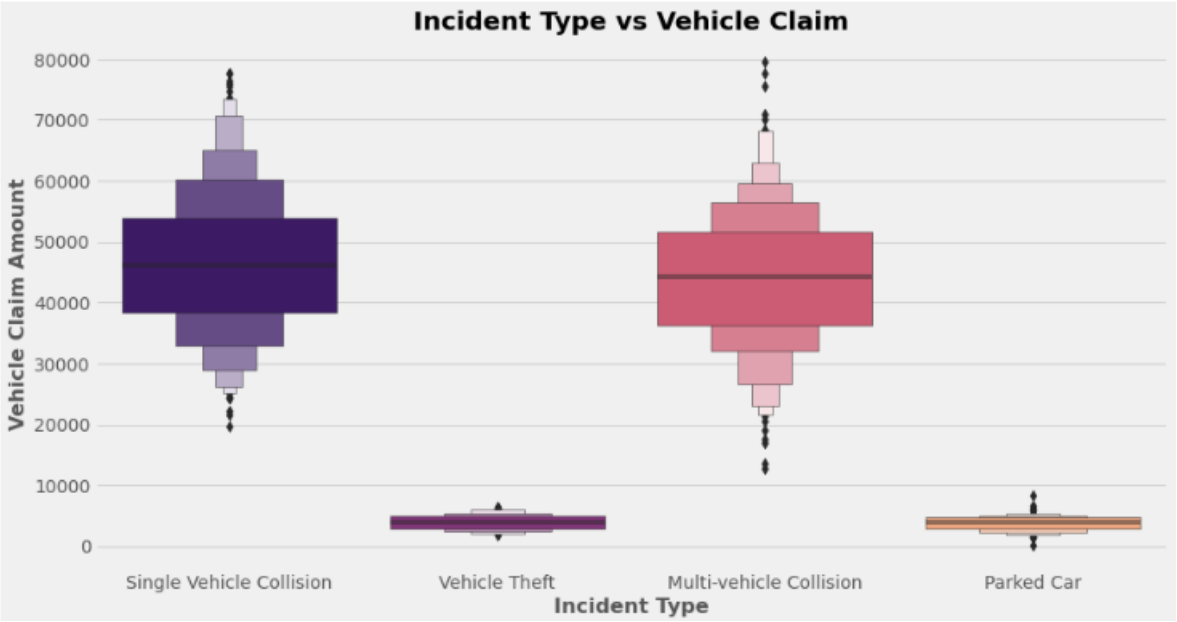
Incident Type VS Severity of collision to gain more insight:

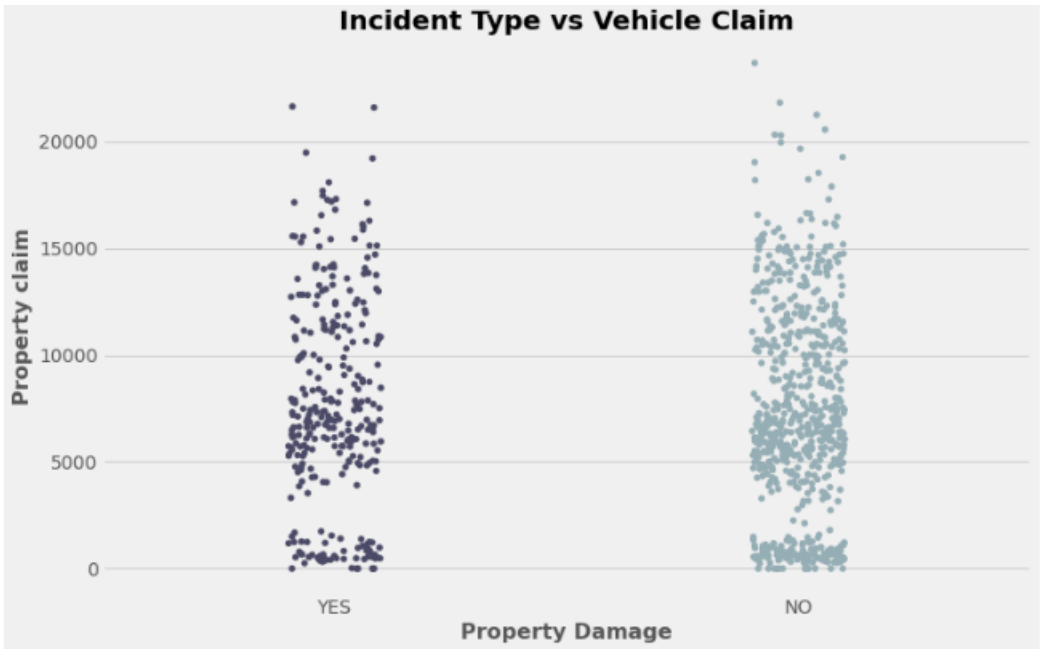
```
pd.crosstab(df['incident_type'], df['incident_severity']).style.background_gradient(cmap='summer_r')
```

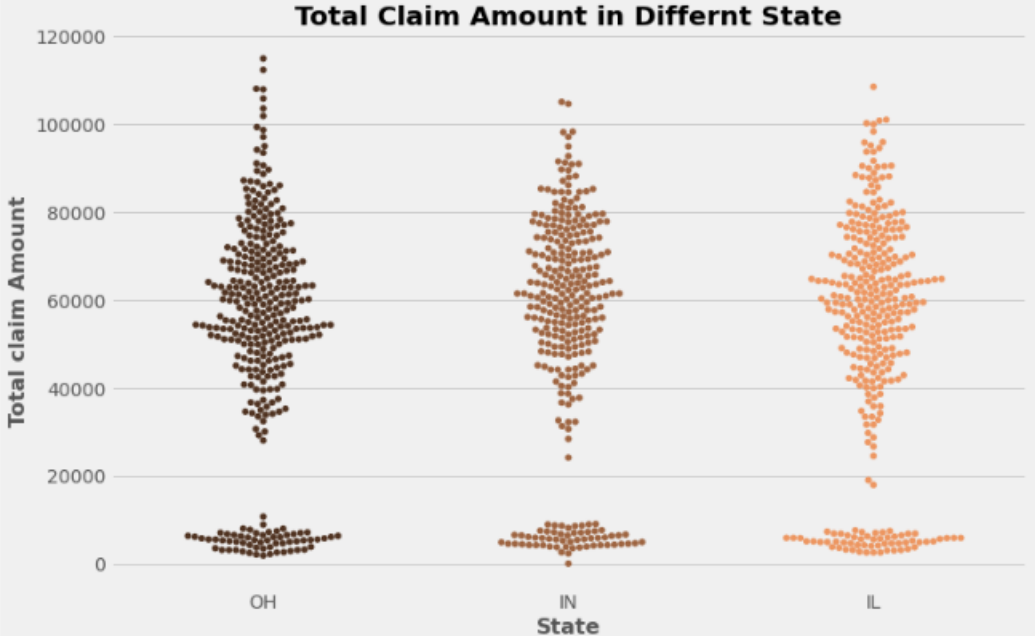
incident_type	Major Damage	Minor Damage	Total Loss	Trivial Damage
Multi-vehicle Collision	144	130	145	0
Parked Car	0	42	0	42
Single Vehicle Collision	132	136	135	0
Vehicle Theft	0	46	0	48

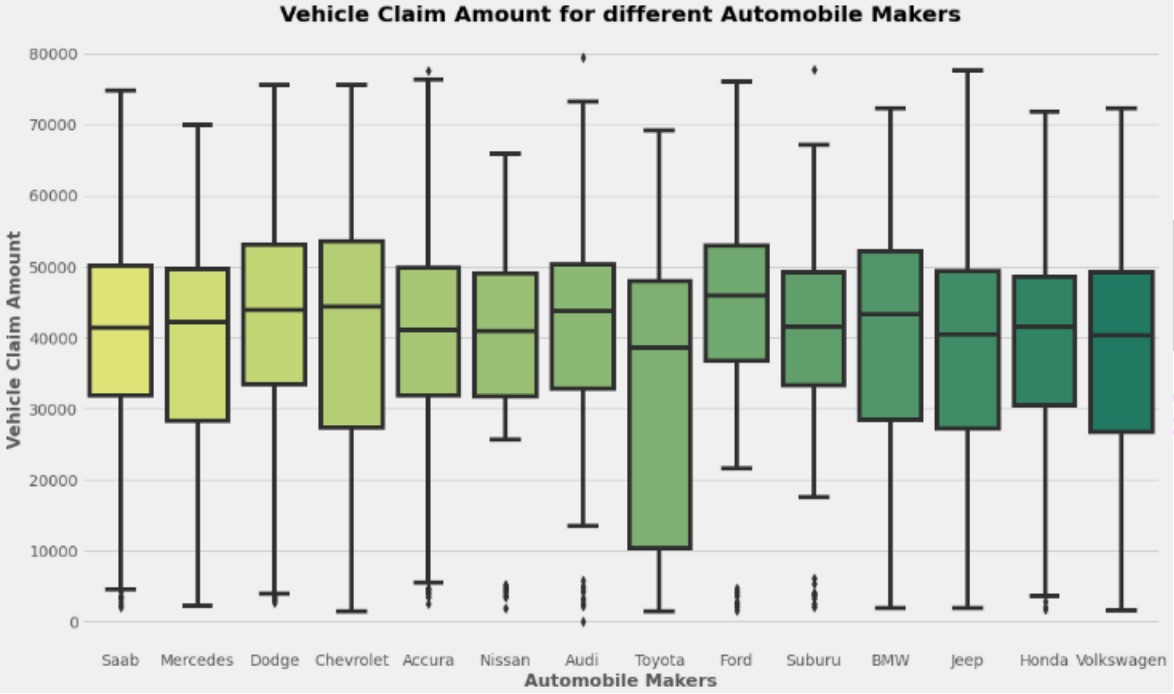


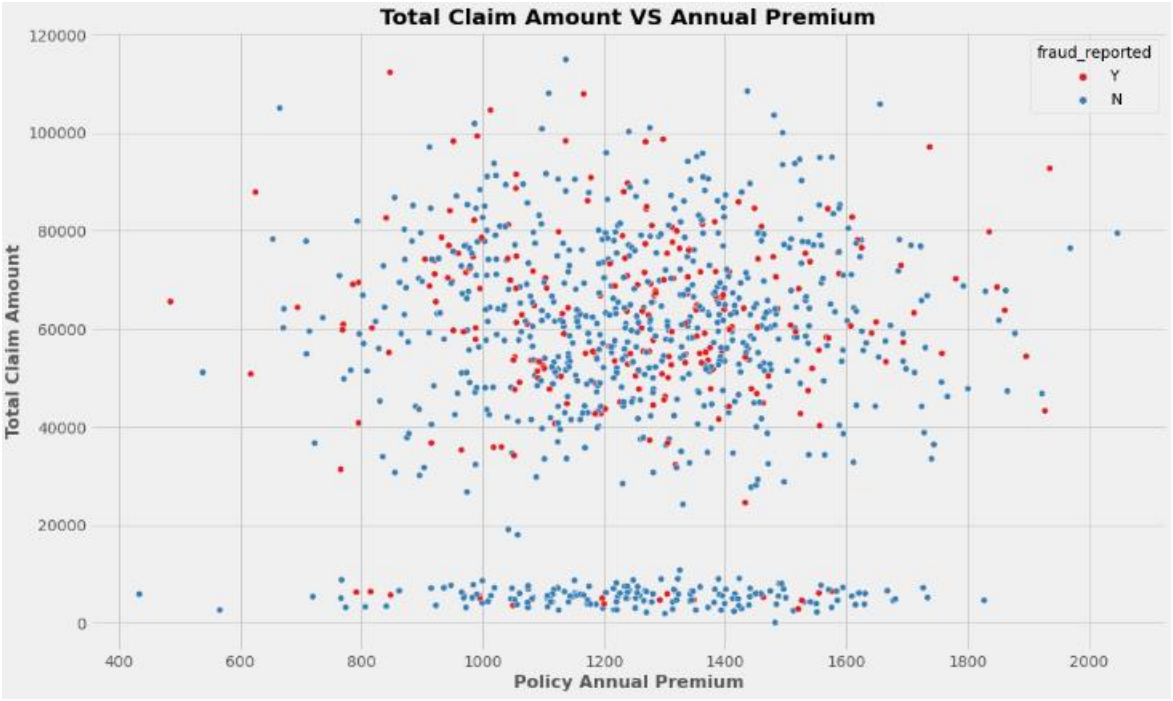
Incident Type VS Vehicle Claim:

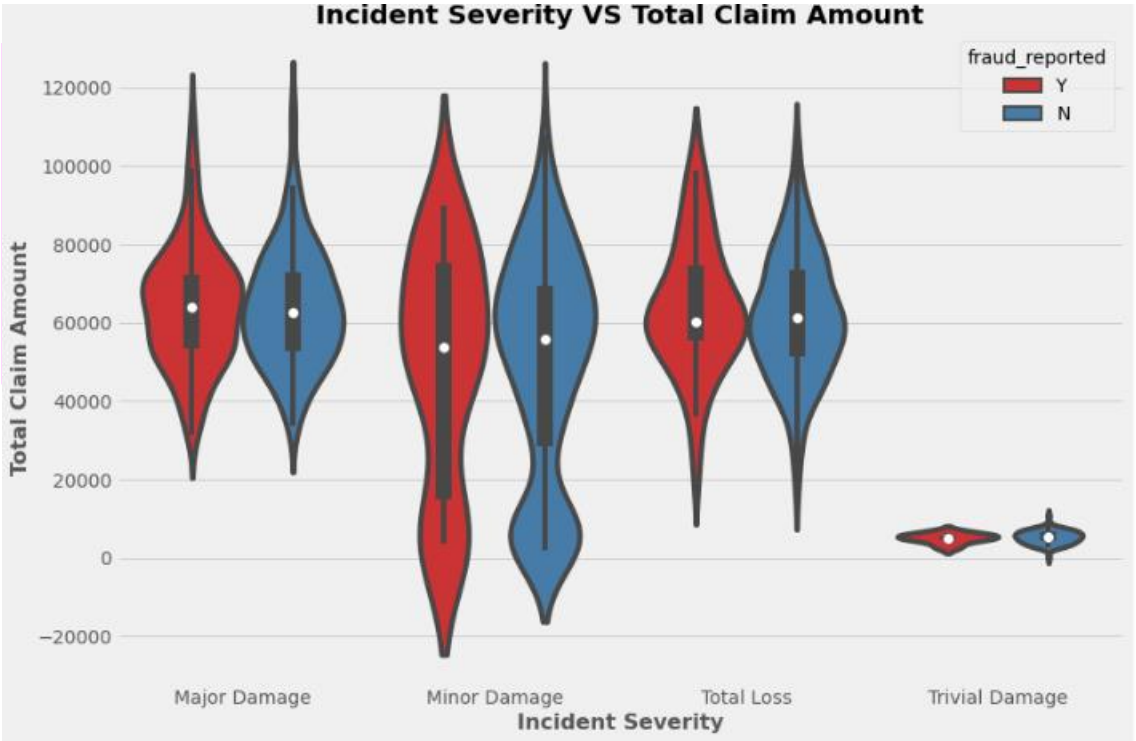





Total Claim Amount in Differnt State
120000
100000
80000
60000
40000
20000
0
Total claim Amount
OH
IN
IL
State


Vehicle Claim Amount for different Automobile Makers
80000
70000
60000
50000
40000
30000
20000
10000
0
Vehicle Claim Amount
Saab
Mercedes
Dodge
Chevrolet
Accura
Nissan
Audi
Toyota
Ford
Suburu
BMW
Jeep
Honda
Volkswagen
Automobile Makers





EDA CONCLUSION

- We have checked the null values in the dataset and there was no missing values found. (One column "_c39" with only "NaN" values was dropped)
- Extracted some new features from the existing features to get better results without any hindrance. And dropped the old columns, if I keep them as it is they will act as duplicates and that leads to a multi collinearity problem.

- Coming to the visualization part, we have found when and where the fraud reports are high in number.
- To get the better insights about the features, I have used count plots, box plots, pair plots, pie charts, scatter plots and swarm plots.

ENCODING THE DATA FRAME

Since our dataset contains many columns with object data type, we need to encode them using any of the encoding methods. Here we apply the label encoding method.

```
Category = ['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', 'CSL_Personal', 'CSL_Accidental']
```

```
# Using Label Encoder on categorical variable
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
for i in Category:
    df[i] = le.fit_transform(df[i])
df.head()
```

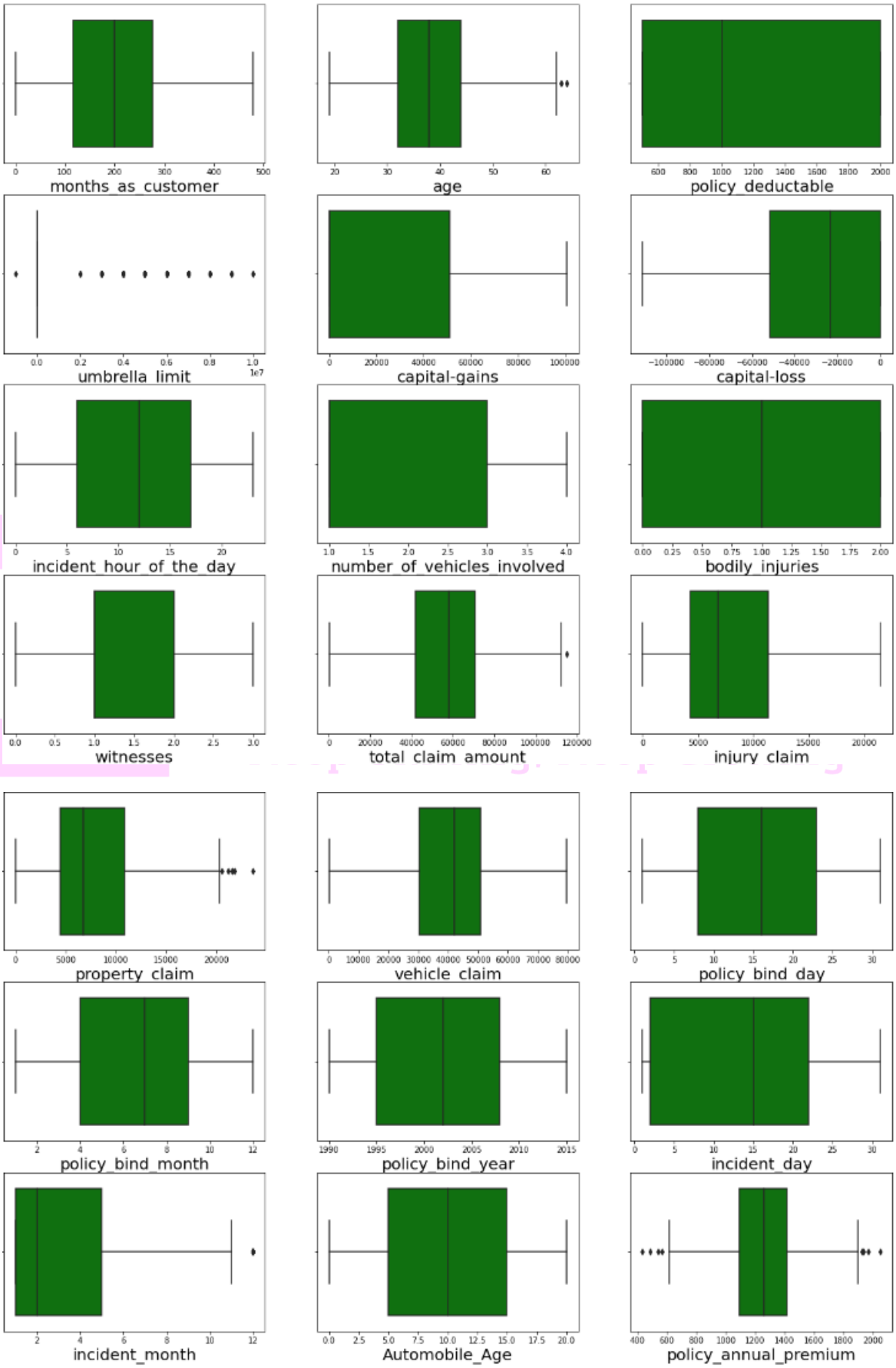
	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education
0	328	48	2	1000	1406.91	0	1	
1	228	42	1	2000	1197.22	5000000	1	
2	134	29	2	2000	1413.14	5000000	0	
3	256	41	0	2000	1415.74	6000000	0	
4	228	44	0	1000	1583.91	6000000	1	

CHECKING OUTLIERS & SKEWNESS

We checked the data for outlier, box plots were used to check each column to find outliers.

```
plt.figure(figsize=(20,30),facecolor='white')
plotnumber=1

for column in Numerical:
    if plotnumber<=21:
        ax=plt.subplot(7,3,plotnumber)
        sns.boxplot(df[column],color='g')
        plt.xlabel(column,fontsize=20)
    plotnumber+=1
plt.show()
```

Outliers were found in the following columns:


months_as_customer
age
policy_deductable
umbrella_limit
1e7
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
Automobile_Age
policy_annual_premium

Outliers were found in “age”, “policy _annual_ premium”, “total _claim _amount”, “property _claim”, “fraud _reported” and “incident _Month”.

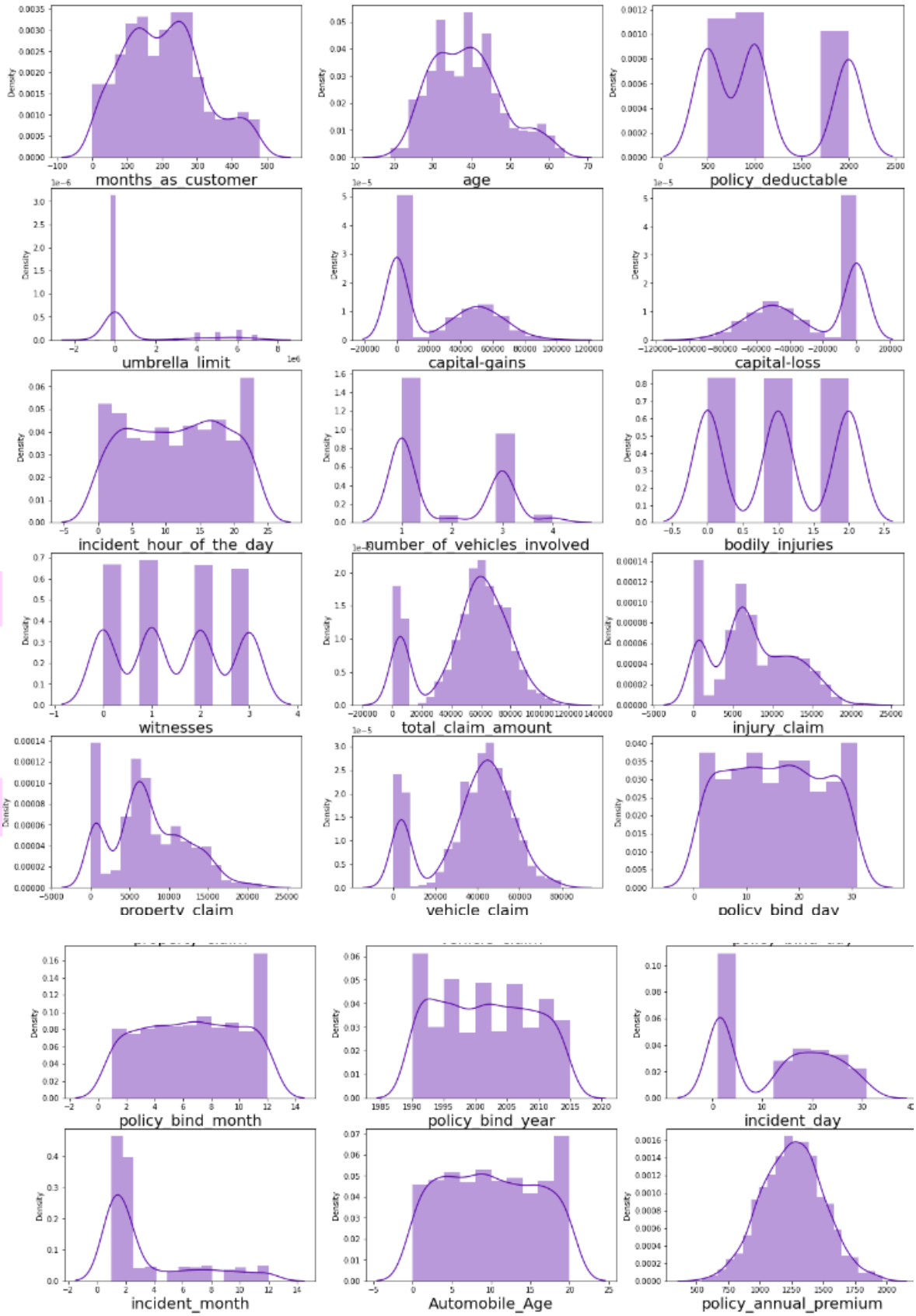
So, we removed the outliers using the Z-Score Method.

```
from scipy.stats import zscore
z = np.abs(zscore(df))
threshold = 3
df1 = df[(z<3).all(axis = 1)]

print ("Shape of the dataframe before removing outliers: ", df.shape)
print ("Shape of the dataframe after removing outliers: ", df1.shape)
print ("Percentage of data loss post outlier removal: ", (df.shape[0]-df1.shape[0])/df.shape[0]*100)

df=df1.copy() # reassigning the changed dataframe name to our original dataframe name
```

```
Shape of the dataframe before removing outliers:  (1000, 40)
Shape of the dataframe after removing outliers:  (980, 40)
Percentage of data loss post outlier removal:  2.0
```

After removing the outliers our data loss was 2 %. Which was affordable.

Then we checked for the skew ness of all the columns of the dataset.


months_as_customer
age
policy_deductable
umbrella_limit
capital-gains
capital-loss
incident_hour_of_the_day
number_of_vehicles_involved
bodily_injuries
witnesses
total_claim_amount
injury_claim
property_claim
vehicle_claim
policy_bind_day
policy_bind_month
policy_bind_year
incident_day
incident_month
Automobile_Age
policy_annual_premium

```
df[Numerical].skew()
```

```
months_as_customer             0.362608
age                            0.475385
policy_deductable              0.476090
umbrella_limit                 1.801424
capital-gains                  0.466619
capital-loss                  -0.376884
incident_hour_of_the_day      -0.039280
number_of_vehicles_involved    0.509725
bodily_injuries                0.003757
witnesses                      0.026211
total_claim_amount            -0.593593
injury_claim                   0.271759
property_claim                 0.361356
vehicle_claim                 -0.620936
policy_bind_day                0.028152
policy_bind_month             -0.024643
policy_bind_year               0.065022
incident_day                   0.055443
incident_month                 1.388336
Automobile_Age                 0.054522
policy_annual_premium          0.035964
dtype: float64
```

- **Out above features 'umbrella_limit','total_claim_amount' and 'vehicle_claim' are continous variable with skew data. The variable 'incident_month' is skewed but it is discrete in nature.So ignore it.**

```
# Making the skew less than or equal to +0.5 and -0.5 for better prediction using yeo-johnson method
skew=['total_claim_amount','vehicle_claim']

# Importing Powertransformer
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')

# Transfroming skew data
df[skew] = scaler.fit_transform(df[skew].values)
```

Checking Skewness after transformation

```
df[skew].skew()
```

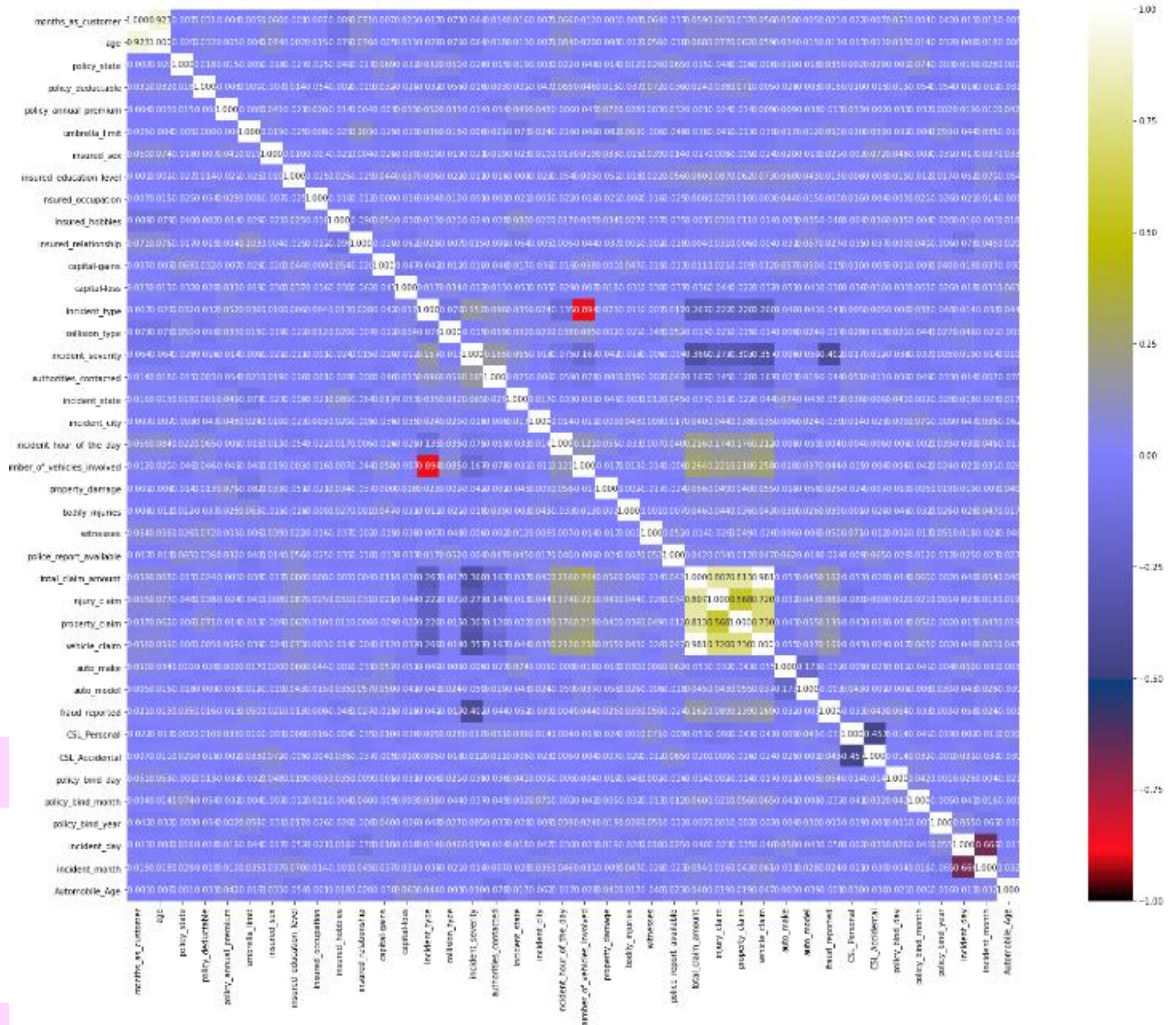
```
total_claim_amount   -0.508540
vehicle_claim        -0.521805
dtype: float64
```

For 'total_claim_amount','vehicle_claim' skewness has not been removed but it got reduced

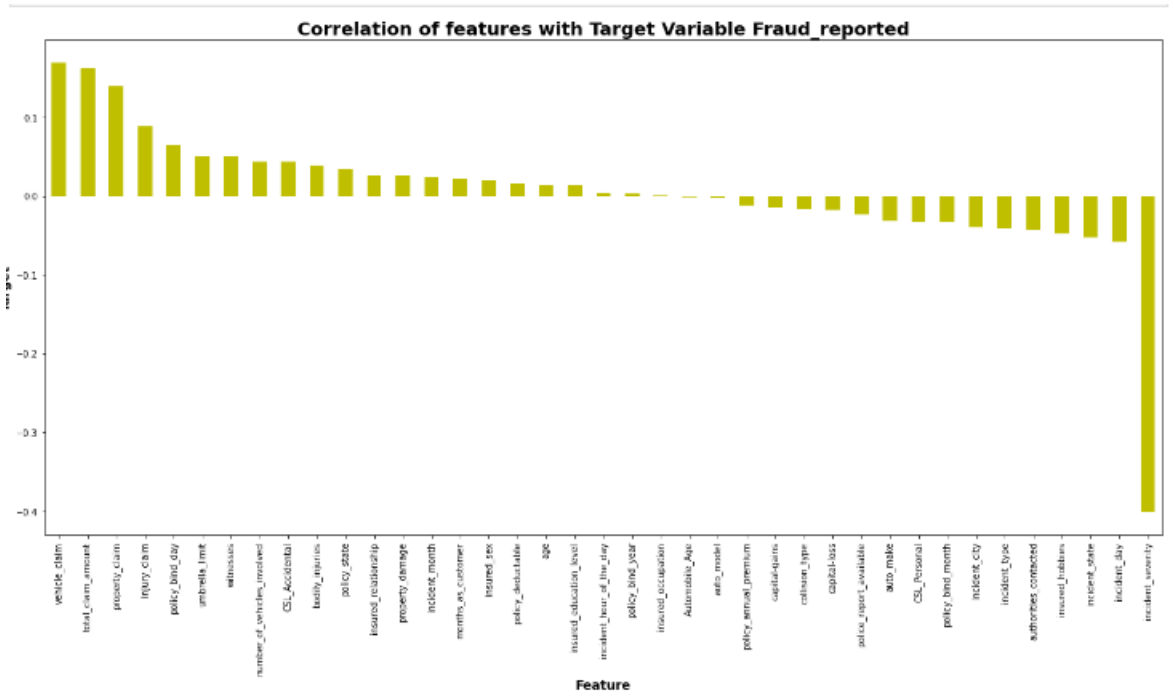
CORRELATION MAP

This Heatmap shows the correlation matrix by visualizing the data. we can observe the relation between one feature to another. This heat map contains both positive and negative correlation.

```
plt.figure(figsize=(30,20))
sns.heatmap(df.corr(), vmin=-1, vmax=1, annot=True, square=True, fmt='0.3f', cmap="gist_stern")
plt.show()
```



From the above correlation map, we see that there is very less correlation between the target variables and the other variables. We can observe that most of the columns are highly correlated with each other which result to the multicollinearity problem. We will check the VIF value to overcome this multicollinearity issue.



Observation:

- incident_severity is correlated with target variable with correlation of 0.4. Other variable are poorly correlated with target variable.
- Other variable are poorly correlated with target variable.
- injury_claim,property_claim,vehicle_claim are highly correlated with each other.
- incident_hour_of_the_day is highly negative correlated with incident type.

PRE-PROCESSING PIPELINE

First of all, we have to balance imbalanced target Variable it using SMOTE.

```
df.fraud_reported.value_counts()
```

```
0    740
1    240
Name: fraud_reported, dtype: int64
```

As Target variable data is Imbalanced in nature we will need to balance target variable.

Balancing using SMOTE:

```
from imblearn.over_sampling import SMOTE
```

```
# Splitting data in target and dependent feature
X = df.drop(['fraud_reported'], axis =1)
Y = df['fraud_reported']
```

```
# Oversampleing using SMOTE Techniques
oversample = SMOTE()
X, Y = oversample.fit_resample(X, Y)
```

```
Y.value_counts()
```

```
1    740
0    740
Name: fraud_reported, dtype: int64
```

We have successfully resolved the class imbalanced problem and now all the categories have same data ensuring that the ML model does not get biased towards one category.

Now, we have to scale the data containing independent variables (x) in order to overcome the data biasness. Since I have removed the skewness and outliers and my data is also normal so I can use the Standard Scaler method to scale the data. If it is not the case then we could apply Min Max Scaler.

```
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
X_scale = scaler.fit_transform(X)
```

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

In the heat map we found some features having high correlation with each other which means that there is a multicollinearity problem, so let's check the VIF values to solve the multicollinearity problem.

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

We got the VIF values as:

	VIF values	Features
0	7.532092	months_as_customer
1	7.495173	age
2	1.071769	policy_state
3	1.058563	policy_deductable
4	1.049535	policy_annual_premium
5	1.071370	umbrella_limit
6	1.087630	insured_sex
7	1.059495	insured_education_level
8	1.028722	insured_occupation
9	1.073167	insured_hobbies
10	1.074099	insured_relationship
11	1.083222	capital-gains
12	1.073871	capital-loss
13	3.808525	incident_type
14	1.107347	collision_type
15	1.365146	incident_severity
16	1.120086	authorities_contacted
17	1.071639	incident_state
18	1.064319	incident_city
19	1.110808	incident_hour_of_the_day
20	3.643035	number_of_vehicles_involved
21	1.065856	property_damage
22	1.052270	bodily_injuries
23	1.067249	witnesses
24	1.104328	police_report_available
25	43792.489034	total_claim_amount
26	1691.268293	injury_claim
27	1694.643275	property_claim
28	21839.207437	vehicle_claim
29	1.091412	auto_make
30	1.088643	auto_model
31	1.235768	CSL_Personal
32	1.198840	CSL_Accidental
33	1.036216	policy_bind_day
34	1.058074	policy_bind_month
35	1.043120	policy_bind_year
36	1.864309	incident_day
37	1.866335	incident_month
38	1.068227	Automobile_Age

We see very high VIF values in “total _claim _amount”, “injury _claim”, “property _claim”, “vehicle _claim”, “policy _bind _year”.

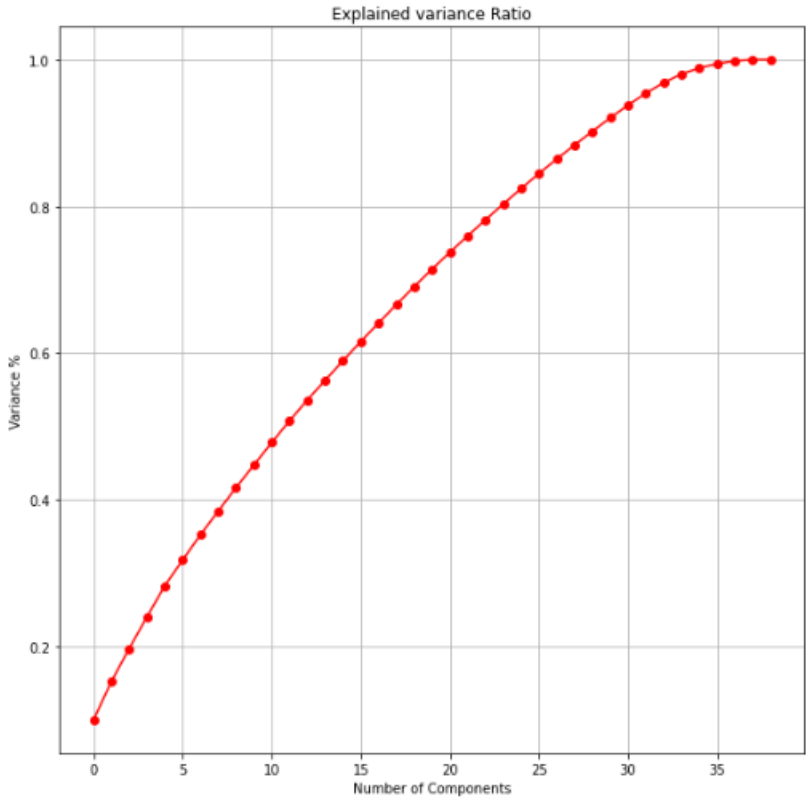
Strategy to Address Multicollinearity:

1. Removing Some of highly correlated features. But this will not work here as most of input features are correlated with each other either moderated or poorly.
2. Another way to address Multicollinearity is to Scaled Data and then apply PCA.

PCA :

Now, we have come across the PCA.

```
from sklearn.decomposition import PCA
pca = PCA()
#plot the graph to find the principal components
x_pca = pca.fit_transform(X_scale)
plt.figure(figsize=(10,10))
plt.plot(np.cumsum(pca.explained_variance_ratio_), 'ro-')
plt.xlabel('Number of Components')
plt.ylabel('Variance %')
plt.title('Explained variance Ratio')
plt.grid()
```



AS per the graph, we can see that 28 principal components attribute for 90% of variation in the data. We shall pick the first 28 components for our prediction.

BUILDING MACHINE LEARNING MODELS

Since all the pre-processing and data cleaning is done, now our data is ready for model building process. Let's get the predictions by creating some classification algorithms.

Before building the models, we first need to find the best random state and accuracy using any one of the classification models.

FINDING THE BEST RANDOM STATE & ACCURACY

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,classification_report,f1_score
maxAccu=0
maxRS=0
for i in range(1,250):
    X_train,X_test,Y_train,Y_test = train_test_split(principle_x,Y,test_size = 0.3, random_state=i)
    log_reg=LogisticRegression()
    log_reg.fit(X_train,Y_train)
    y_pred=log_reg.predict(X_test)
    acc=accuracy_score(Y_test,y_pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print('Best accuracy is', maxAccu ,'on Random_state', maxRS)
```

```
Best accuracy is 0.8108108108108109 on Random_state 9
```

- ❖ We have got the best random state as 9 and best accuracy as 81.08% using the Logistic Regression model. Now let's create new train sets and test sets and fit them into the models to find our ideal model.

```
X_train, X_test, Y_train, Y_test = train_test_split(principle_x, Y, random_state=9, test_size=.3)
log_reg=LogisticRegression()
log_reg.fit(X_train,Y_train)
y_pred=log_reg.predict(X_test)
print('\033[1m'+'Logistics Regression Evaluation'+'\033[0m')
print('\n')
print('\033[1m'+'Accuracy Score of Logistics Regression :'+'\033[0m', accuracy_score(Y_test, y_pred))
print('\n')
print('\033[1m'+'Confusion matrix of Logistics Regression :'+'\033[0m \n',confusion_matrix(Y_test, y_pred))
print('\n')
print('\033[1m'+'classification Report of Logistics Regression'+'\033[0m \n',classification_report(Y_test, y_pred))
```

CLASSIFICATION ALGORITHMS :

We have used 7 different classification algorithms for our predictions, they are: Logistic Regression Model, Decision Tree Classifier, Gaussian NB Classifier, K-Nearest Neighbors Classifier, SVC Model, Random Forest Classifier and Extra Trees Classifier

We have used evaluation metrics like classification report, confusion matrix, roc score and accuracy score. And we also used a cross validation score (cvs) to get the difference from the model accuracy for better result.

```
model=[ LogisticRegression(),
        SVC(),
        GaussianNB(),
        DecisionTreeClassifier(),
        KNeighborsClassifier(n_neighbors = 3),
        RandomForestClassifier(),
        ExtraTreesClassifier()]

for m in model:
    m.fit(X_train,Y_train)
    y_pred=m.predict(X_test)
    print('\033[1m'+'Classification ML Algorithm Evaluation Matrix',m,'is' +'\033[0m')
    print('\n')
    print('\033[1m'+'Accuracy Score :'+'\033[0m\n', accuracy_score(Y_test, y_pred))
    print('\n')
    print('\033[1m'+'Confusion matrix :'+'\033[0m \n',confusion_matrix(Y_test, y_pred))
    print('\n')
    print('\033[1m'+'Classification Report :'+'\033[0m \n',classification_report(Y_test, y_pred))
    print('\n')
    print('=========================================================================================================='
```

- The Logistic Regression Model gave us an accuracy score of **81.08 %.**

```
Classification ML Algorithm Evaluation Matrix LogisticRegression() is

Accuracy Score :
 0.8108108108108109

Confusion matrix :
 [[182  45]
 [ 39 178]]

Classification Report :
              precision    recall  f1-score   support

           0       0.82      0.80      0.81       227
           1       0.80      0.82      0.81       217

    accuracy                           0.81       444
   macro avg       0.81      0.81      0.81       444
weighted avg       0.81      0.81      0.81       444
```

- The SVC Model gave us an accuracy score of **83.33 %.**

```
Classification ML Algorithm Evaluation Matrix SVC() is

Accuracy Score :
 0.8333333333333334

Confusion matrix :
 [[193  34]
 [ 40 177]]

Classification Report :
              precision    recall  f1-score   support

           0       0.83      0.85      0.84       227
           1       0.84      0.82      0.83       217

    accuracy                           0.83       444
   macro avg       0.83      0.83      0.83       444
weighted avg       0.83      0.83      0.83       444
```

- The Gaussian Model gave us an accuracy score of **78.82** %.

```
Classification ML Algorithm Evaluation Matrix GaussianNB() is

Accuracy Score :
 0.7882882882882883

Confusion matrix :
 [[182  45]
 [ 49 168]]

Classification Report :
              precision    recall  f1-score   support

           0       0.79      0.80      0.79       227
           1       0.79      0.77      0.78       217

    accuracy                           0.79       444
   macro avg       0.79      0.79      0.79       444
weighted avg       0.79      0.79      0.79       444
```

❖ The Decision Tree Model gave us an accuracy score of **68.01 %.**

```
--------------------------------------------------------------------------------
Classification ML Algorithm Evaluation Matrix DecisionTreeClassifier() is


Accuracy Score :
 0.6801801801801802


Confusion matrix :
 [[152  75]
 [ 67 150]]


Classification Report :
              precision    recall  f1-score   support

           0       0.69      0.67      0.68       227
           1       0.67      0.69      0.68       217

    accuracy                           0.68       444
   macro avg       0.68      0.68      0.68       444
weighted avg       0.68      0.68      0.68       444
```

❖ The K Neighbor Classifier Model gave us an accuracy score of **72.29 %.**

```
--------------------------------------------------------------------------------
Classification ML Algorithm Evaluation Matrix KNeighborsClassifier(n_neighbors=3) is


Accuracy Score :
 0.722972972972973


Confusion matrix :
 [[115 112]
 [ 11 206]]


Classification Report :
              precision    recall  f1-score   support

           0       0.91      0.51      0.65       227
           1       0.65      0.95      0.77       217

    accuracy                           0.72       444
   macro avg       0.78      0.73      0.71       444
weighted avg       0.78      0.72      0.71       444
```

❖ The Random Forest Classifier Model gave us an accuracy score of **80.63 %.**

```
Classification ML Algorithm Evaluation Matrix RandomForestClassifier() is


Accuracy Score :
 0.8063063063063063


Confusion matrix :
 [[189  38]
 [ 48 169]]


Classification Report :
              precision    recall  f1-score   support

           0       0.80      0.83      0.81       227
           1       0.82      0.78      0.80       217

    accuracy                           0.81       444
   macro avg       0.81      0.81      0.81       444
weighted avg       0.81      0.81      0.81       444
```

❖ The Extra Trees Classifier Model gave us an accuracy score of **82.20 %.**

```
Classification ML Algorithm Evaluation Matrix ExtraTreesClassifier() is


Accuracy Score :
 0.8220720720720721


Confusion matrix :
 [[186  41]
 [ 38 179]]


Classification Report :
              precision    recall  f1-score   support

           0       0.83      0.82      0.82       227
           1       0.81      0.82      0.82       217

    accuracy                           0.82       444
   macro avg       0.82      0.82      0.82       444
weighted avg       0.82      0.82      0.82       444
```

- From the above Classification Models, the highest accuracy score belongs to **SVC followed by Extra Trees Classifier, then by Logistic Regression model & Random Forest Classifier.**

The lowest Accuracy score belongs to **Decision Tree Model**.

CROSS VALIDATION SCORES:

❖ We now checked the cross-validation score of each of the models mentioned above.

```
from sklearn.model_selection import cross_val_score
model=[LogisticRegression(),
        SVC(),
        GaussianNB(),
        DecisionTreeClassifier(),
        KNeighborsClassifier(n_neighbors = 3),
        RandomForestClassifier(),
        ExtraTreesClassifier()]

for m in model:
    score = cross_val_score(m, principle_x, Y, cv =5)
    print('\n')
    print('\033[1m'+'Cross Validation Score', m, ':'+'\033[0m\n')
    print("Score :" ,score)
    print("Mean Score :",score.mean())
    print("Std deviation :",score.std())
    print('\n')
    print('============================================================================================================')
```

```
Cross Validation Score LogisticRegression() :

Score : [0.59459459 0.68918919 0.81756757 0.81081081 0.87837838]
Mean Score : 0.7581081081081081
Std deviation : 0.1022393530777175
```

```
--------------------------------------------------------------------------------------------------------------


Cross Validation Score SVC() :

Score : [0.58783784 0.72297297 0.89189189 0.89864865 0.92905405]
Mean Score : 0.806081081081081
Std deviation : 0.13083705343199603


==============================================================================================================


Cross Validation Score GaussianNB() :

Score : [0.57432432 0.70608108 0.86148649 0.875      0.89527027]
Mean Score : 0.7824324324324323
Std deviation : 0.12384602471552263


==============================================================================================================


Cross Validation Score DecisionTreeClassifier() :

Score : [0.58783784 0.65540541 0.76689189 0.77027027 0.76351351]
Mean Score : 0.7087837837837838
Std deviation : 0.07433660832148685


==============================================================================================================


Cross Validation Score KNeighborsClassifier(n_neighbors=3) :

Score : [0.68918919 0.71283784 0.75675676 0.79054054 0.78040541]
Mean Score : 0.745945945945946
Std deviation : 0.03901405414392448


==============================================================================================================


Cross Validation Score RandomForestClassifier() :

Score : [0.60472973 0.70945946 0.86148649 0.89189189 0.92905405]
Mean Score : 0.7993243243243243
Std deviation : 0.12264211505711642


 Cross Validation Score ExtraTreesClassifier() :

 Score : [0.66216216 0.73986486 0.91216216 0.90540541 0.92567568]
 Mean Score : 0.8290540540540541
 Std deviation : 0.10759167188844888
```

❖ HYPER PARAMETER TUNING :

Since the Cross Validation Score and the Accuracy Score are both high, we shall consider this model for hyper parameter tuning.

- We will use Grid Search CV for hyper parameter tuning.

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

```
parameter= {'criterion' : ['gini', 'entropy'],
            'max_features':['auto','sqrt','log2'],
             'min_samples_split':[3,5,8,11],
             'max_depth' : [10,20,30],
             'n_estimators' : [100,200,300,400]
            }
```

```
GCV = GridSearchCV(ExtraTreesClassifier(),parameter,verbose=10)
GCV.fit(X_train,Y_train)
```

```
GCV.best_params_
```

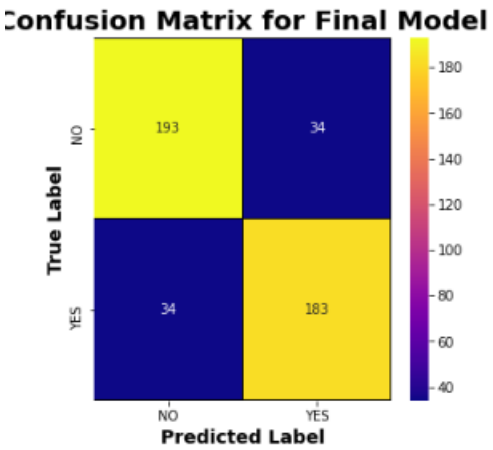
```
{'criterion': 'gini',
 'max_depth': 30,
 'max_features': 'log2',
 'min_samples_split': 3,
 'n_estimators': 200}
```

- These were found to be the best parameters after tuning, now let us use these parameters to improve our model.

```
Final_mod = ExtraTreesClassifier(criterion='gini',n_estimators= 200, max_depth=30 ,
                                 min_samples_split= 3, max_features= 'log2')
Final_mod.fit(X_train,Y_train)
y_pred=Final_mod.predict(X_test)
print('\033[1m'+'Accuracy Score :'+'\033[0m\n', accuracy_score(Y_test, y_pred))
```

```
Accuracy Score :
 0.8468468468468469
```

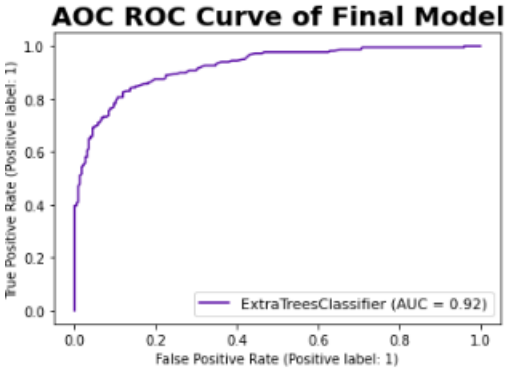
```
# Lets plot confusion matrix for  FinalModel
Matrix = confusion_matrix(Y_test, y_pred)

x_labels = ["NO","YES"]
y_labels = ["NO","YES"]

fig , ax = plt.subplots(figsize=(5,5))
sns.heatmap(cm, annot = True,linewidths=.2, linecolor="black", fmt = ".0f", ax=ax,
            cmap="plasma", xticklabels = x_labels, yticklabels = y_labels)

plt.xlabel("Predicted Label",fontsize=14,fontweight='bold')
plt.ylabel("True Label",fontsize=14,fontweight='bold')
plt.title('Confusion Matrix for Final Model',fontsize=20,fontweight='bold')
plt.show()
```



The model after hyper parameter tuning has an improved accuracy score of 84.68 %.

Now we will plot the ROC curve and compare the AUC for the best model.

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.metrics import plot_roc_curve
disp = plot_roc_curve(Final_mod,X_test,Y_test)
plt.legend(prop={'size':11}, loc='lower right')
plt.title('AOC ROC Curve of Final Model',fontsize=20,fontweight='bold')
plt.show()
auc_score = roc_auc_score(Y_test, Final_mod.predict(X_test))
print('\033[1m'+'Auc Score :'+'\033[0m\n',auc_score)
```




```
Auc Score :
 0.8467691183337055
```

We have plotted the ROC-AUC curve, AUC score is **84.67%.**

❖ SAVING THE MODEL :

Finally, we saved the model by using library **"joblib".**

```
import joblib
joblib.dump(Final_mod,'Insurance_claims_Final.pkl')
```

```
['Insurance_claims_Final.pkl']
```

❖ PREDICTION:

By loading the saved model, we can now check predict value whether the insurance claim is fraudulent or not.

```
# Prediction
prediction = Final_mod.predict(X_test)
```

```
Actual = np.array(Y_test)
df_Pred = pd.DataFrame()
df_Pred["Predicted Values"] = prediction
df_Pred["Actual Values"] = Actual
df_Pred.head()
```

	Predicted Values	Actual Values
0	1	1
1	0	0
2	0	0
3	1	1
4	1	1

- The above shows the predicted values and the actual values. The values are almost similar.

CONCLUDING REMARKS

In this project we went through the different processes involved in building a machine learning model. We started with Exploratory Data Analysis, did some Data Cleaning, conducted some Feature Extraction & Feature Engineering which were crucial in making our data ready for visualization and model building.

We did some Data Visualization using count plots, scatter plots, bar plots & swarm plots. After visualization we encoded the data frame using Label Encoder. Next, we checked for outliers present in the data and removed them using the z-score method. We checked the skewness of our data and reduced it for better model building.

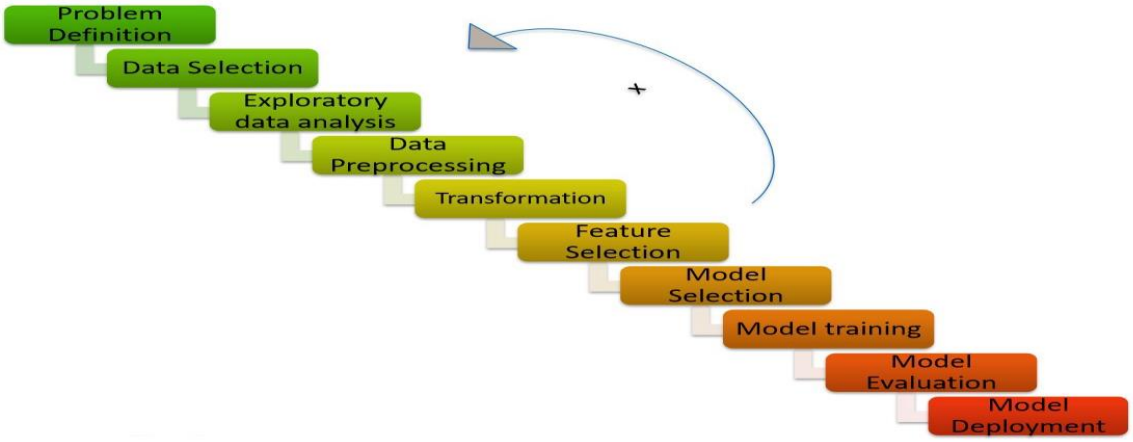
And lastly, we built different classification models to predict whether the insurance claim is fraudulent or not and performed the hyper tuning to improve the best model by using different parameters.

With the help of above techniques, our model is able to predict the fraudulent report with the accuracy of 84.68%. Also, we have seen that the actual and predicted values are almost the same, which means our model worked correctly.

Insurance fraud is a huge 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.

Building machine learning models for such problems can help the insurance companies to choose the correct insurer. So, Machine learning techniques are very useful to solve these kinds of problems. This project has built a model that can

detect auto insurance fraud. In doing so, the model can reduce loses for insurance companies. The challenge behind fraud detection in machine learning is that frauds are far less common as compared to legit insurance claims.



Hardware & Software Requirements & Tools Used:

Hardware required:

- Processor: core i5 or above
- RAM: 8 GB or above
- ROM/SSD: 250 GB or above

Software requirement:

- Jupiter Notebook

Libraries Used:

- Python
- NumPy
- Date Time
- Scikit Learn
- Seaborn
- Pandas
- Matplotlib

Thank You!