Blog on: Insurance Claim Fraud Detection Using Machine Learning

Author: Suraj Kumar Soni

Batch: 1842

DataTrained

Introduction:

Scam or Fraud is one of the largest and most well-known problems that insurers face in the insurance industry.

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.

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 a 92% 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: core i5 or above
- 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

Problem Definition:

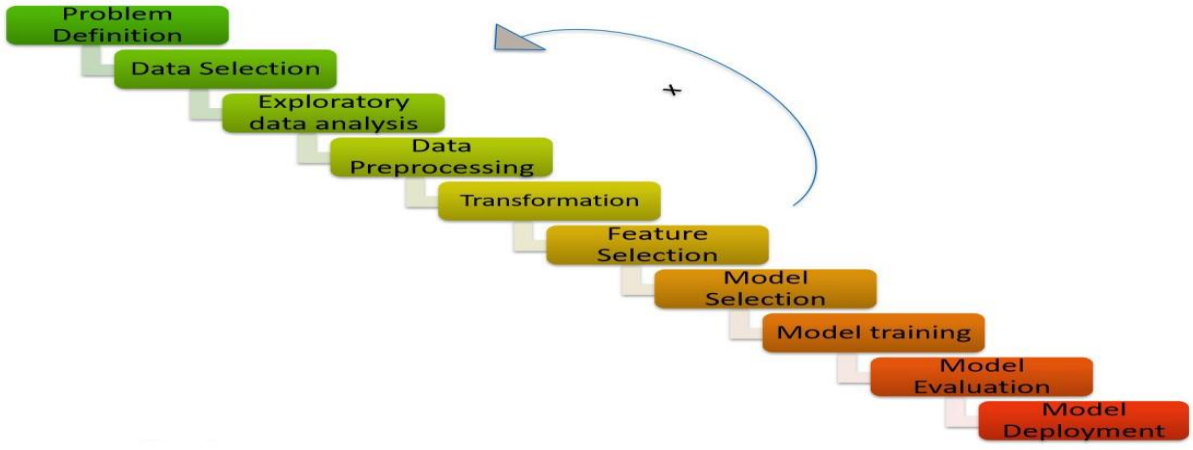
Insurance fraud is a big problem in this industry. It's difficult to identify fraud claims. Machine Learning algorithms are in a unique position to help the Auto Insurance industry with this huge problem. In this project, we are provided a dataset that has the details of the insurance policy along with the customer's details. It also has the details of the accident based on which the claims have been made.

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

In this problem, we will be looking into the insured person's details and the incidents and we will be analyzing the sample to understand if the claim is genuine or not.

Let's deep dive step by step into the data analysis process.

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



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
- Date Time has been used to extract day/month/date separately
- Sklearn has been used in the model building

```
import warnings
warnings.simplefilter("ignore")
warnings.filterwarnings("iganore")
import joblib

import pandas as pd
import numpy as np
import seaborn as sns
import missingno
import matplotlib.pyplot as plt
%matplotlib inline

from scipy.stats import zscore
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OrdinalEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.neighbors import KNeighborsClassifier
import xgboost as xgb
import lightgbm as lgb

from sklearn import metrics
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
```

Importing the Dataset

- Let's import the dataset from the source.

```
df = pd.read_csv("https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fraud.csv")
```

```
df # checking the first 5 and last 5 rows of our dataset
```

- imported the dataset which was in "CSV" format as "df". Below is how the dataset looks.

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

By observing the dataset we could understand that the dataset contains both categorical and numerical columns. Here "fraud_reported" is our target column since it has two categories so it is called to be a "Classification Problem" where we need to predict if an insurance claim is fraudulent or not. As it is a classification problem hence we will use all the classification algorithms while building the model.

by using 'df. shape' we can find figure out how many rows and columns we have in the 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 lifecycle 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 preprocessing, and transformation, which is the most important part of any machine learning model, further data will be analyzed and cleaned for better model accuracy which we will get, or the model can remain overfitting or underfitting. We will discuss further why all the steps are used.

Exploratory Data Analysis and 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 shape, 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.

```
# 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):
```

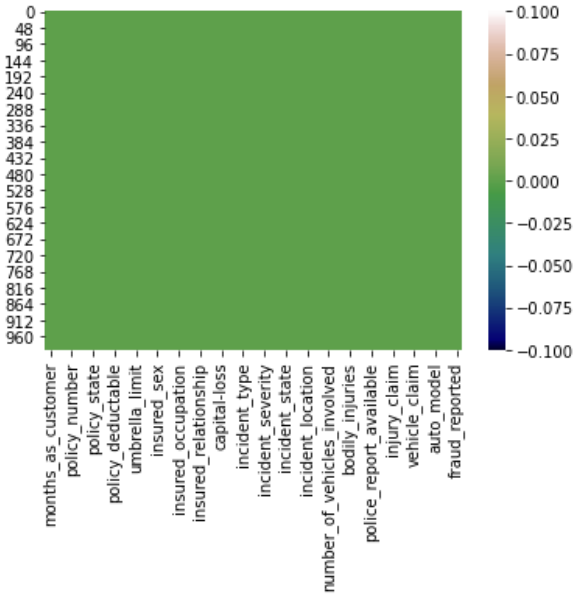
```
# Checking the type of dataset
df.dtypes
```

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

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

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

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



Observations :

1. As we can see the dataset does not have any null values in it.
2. dataset contains 3 different types of data namely integer data type, float data type, and object data type.
3. 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)
```

4. 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)
```

5. by looking at the value counts of the particular column we can realize that the columns **umbrella limit, capital-gains**, and **capital-loss** contain more zero values around 79.8%, 50.8%, and 47.5%. we will be keeping the zero values in the capital gains and capital loss columns as it is. Since the umbrella limit columns have more than 70% of zero values, let's drop them as it will create no impact on the analysis.

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

6. 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.

```
# 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.

```
# 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.

```
# 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.

```python
# 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]
```

```python
# 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')
```

```python
# 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.

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

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.

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

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

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 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")
```

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

	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

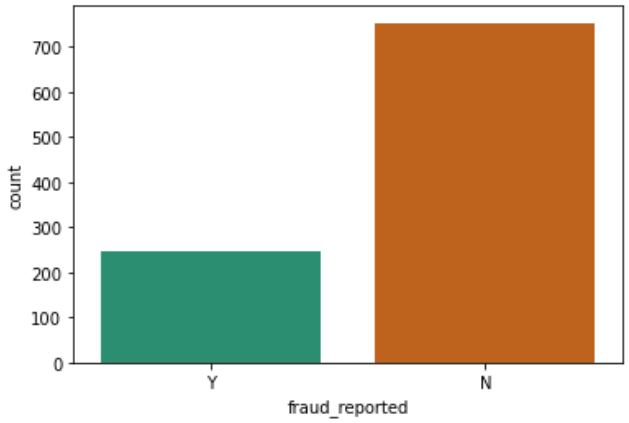
Preparing for Visualization

we will look into the categorical and numerical columns so that we can visualize the features accordingly.

```
# Separating numerical and categorcal 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)
```

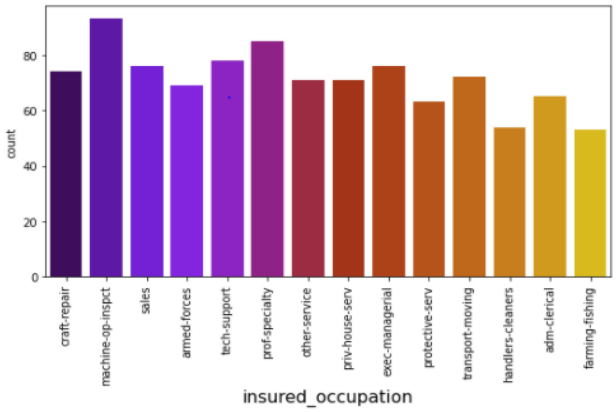
Visualization

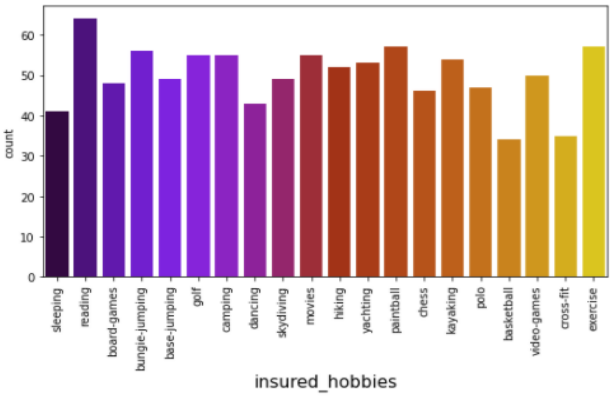
```
print(df["fraud_reported"].value_counts())
sns.countplot(df["fraud_reported"],palette="Dark2")
plt.show()
```

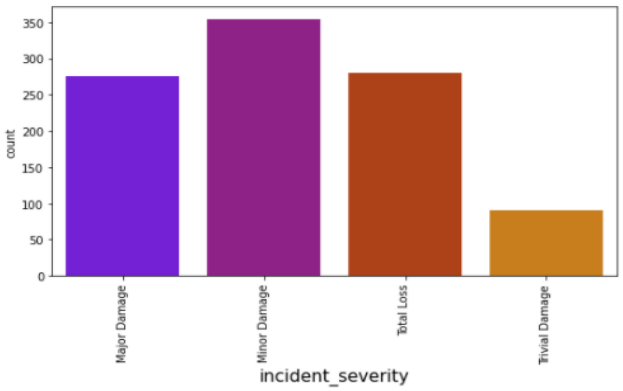


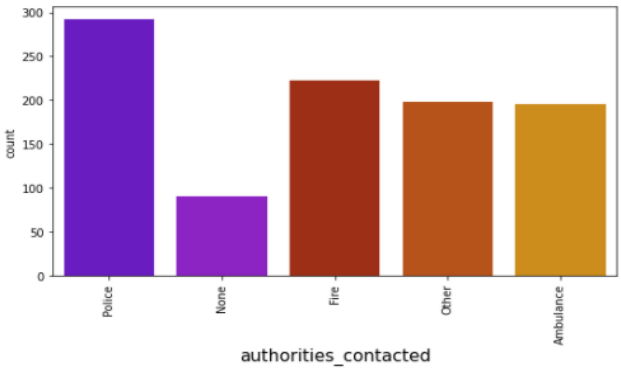
After looking into the plot, we can observe that the count of "N" is high compared to "Y". This means here we can assume that "Y" stands for "Yes" and that the insurance is fraudulent and "N" stands for "No" which means the insurance claim is not fraudulent. most of the insurance claims have not been reported as fraudulent. Since it is our target column, it indicates the class is imbalanced. We will balance the data using an oversampling method further.

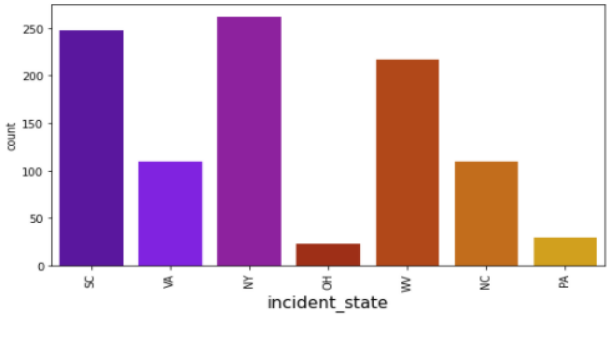
```
cols2 = ['insured_occupation', 'insured_hobbies', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_

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()
```

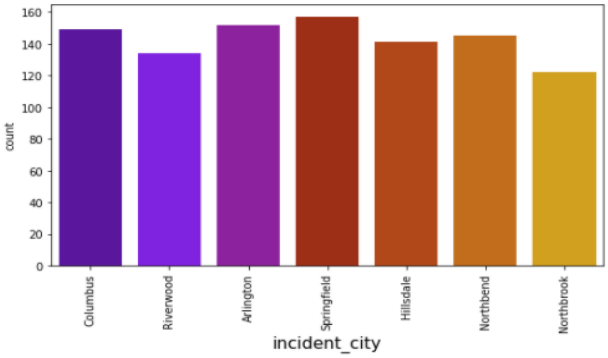

count
craft-repair
machine-op-inspct
sales
armed-forces
tech-support
prof-specialty
other-service
priv-house-serv
exec-managerial
protective-serv
transport-moving
handlers-cleaners
adm-clerical
farming-fishing
insured_occupation

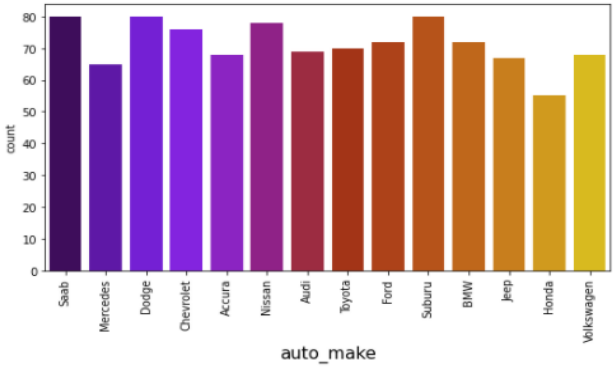

count
sleeping
reading
board-games
bungie-jumping
base-jumping
golf
camping
dancing
skydiving
movies
hiking
yachting
paintball
chess
kayaking
polo
basketball
video-games
cross-fit
exercise
insured_hobbies

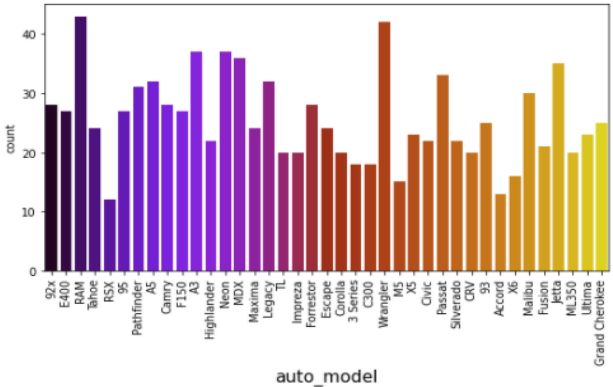

count
Major Damage
Minor Damage
Total Loss
Trivial Damage
incident_severity


count
Police
None
Fire
Other
Ambulance
authorities_contacted


count
SC
VA
NY
OH
WV
NC
PA
incident_state


count
Columbus
Riverwood
Arlington
Springfield
Hillsdale
Northbend
Northbrook
incident_city

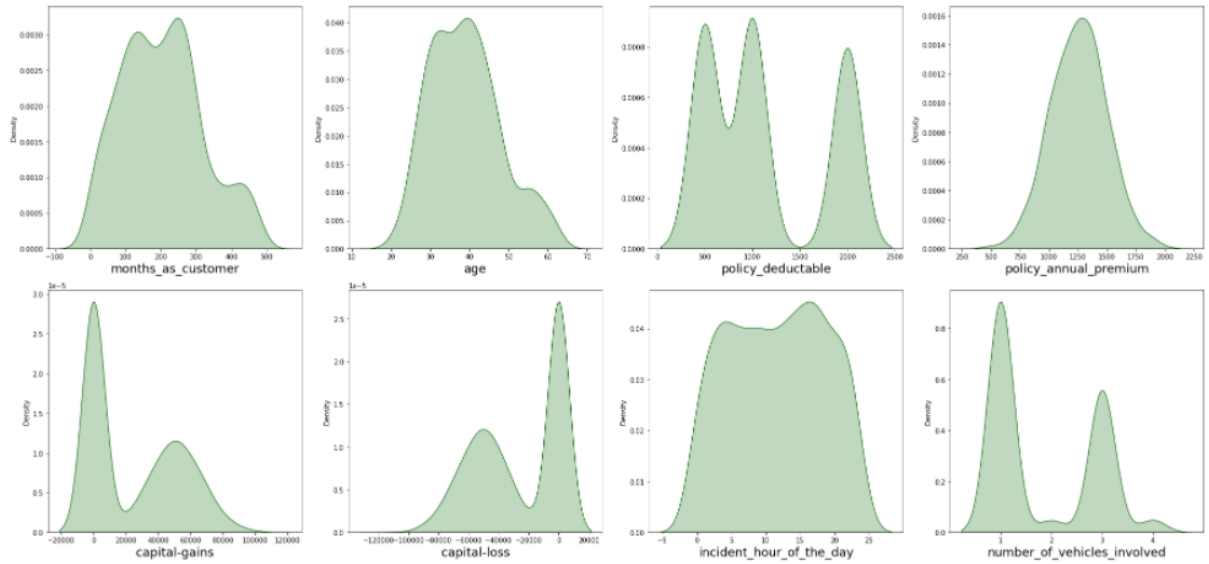

count
Saab
Mercedes
Dodge
Chevrolet
Accura
Nissan
Audi
Toyota
Ford
Suburu
BMW
Jeep
Honda
Volkswagen
auto_make

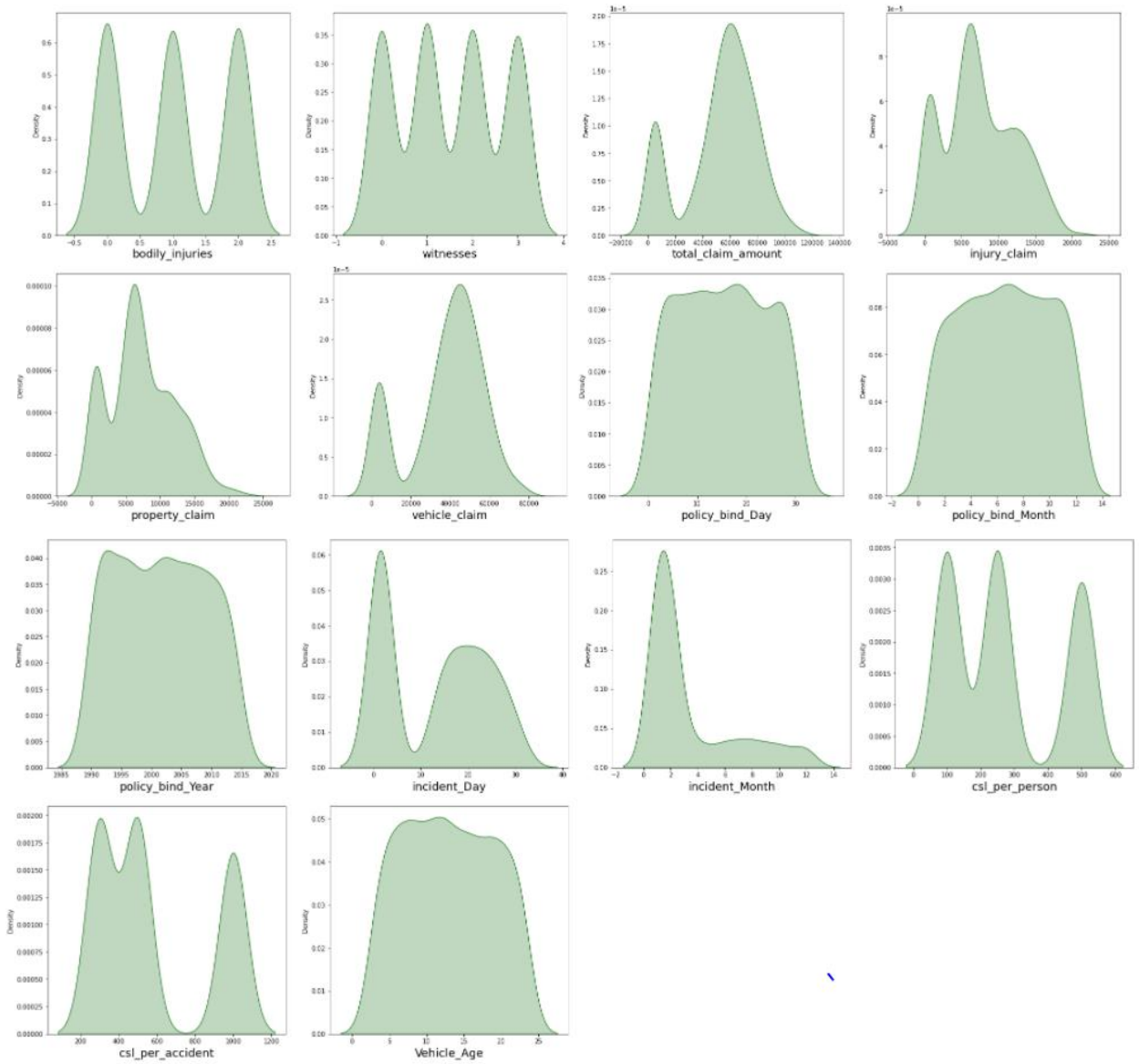

count
92x
E400
RAM
Tahoe
RSX
95
Pathfinder
A5
Camry
F150
A3
Highlander
Neon
MDX
Maxima
Legacy
TL
Impreza
Forrestor
Escape
Corolla
3 Series
C300
Wrangler
M5
X5
Civic
Passat
Silverado
CRV
93
Accord
X6
Malibu
Fusion
Jetta
ML350
Ultima
Grand Cherokee
auto_model

By looking into the count plots we can observe the following things:

- In the insured occupation we can observe most of the data is covered by a machine operation inspector followed by a professional.
- Concerning 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 a very less count compared to others.
- When accidents occur most of the authorities contact the police, here the category police cover the highest data, and Fire has the second highest count. But Ambulance and Others have almost the same counts and the count is very less for none compared to all.
- Concerning to the incident state, New York, South Carolina, and West Virginia states have the highest counts. In the 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 the highest counts.
- When we take a look at the vehicle models the RAM and Wrangler automobile models have the highest counts and also RSX and Accord have very less counts.

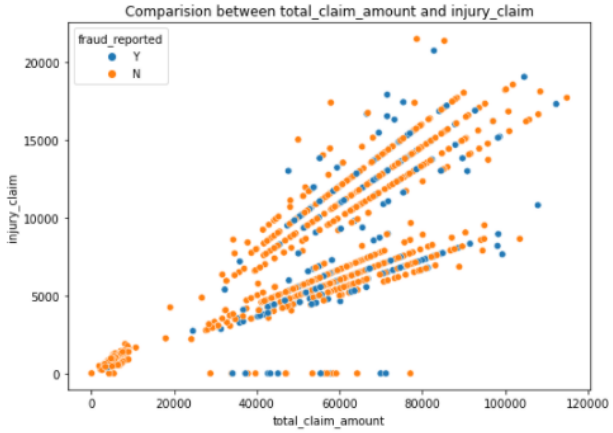
```
plt.figure(figsize=(25,35),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=23:
        ax=plt.subplot(6,4,plotnumber)
        sns.distplot(df[column],color="darkgreen",hist=False,kde_kws={"shade": True})
        plt.xlabel(column,fontsize=18)
    plotnumber+=1
plt.tight_layout()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

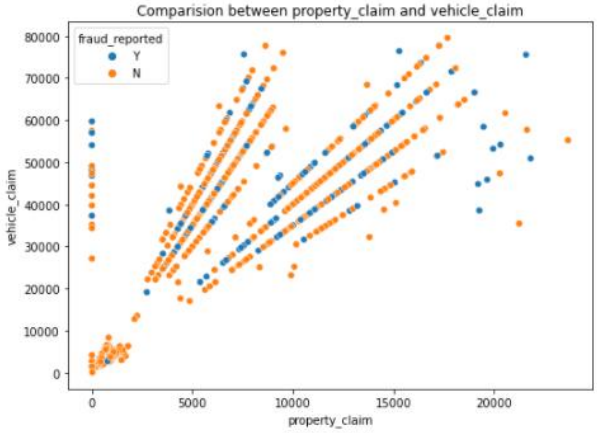




The data is normally distributed in most of the columns. Some of the columns like capital gains and incident months have a mean value greater than the median, hence they are skewed to right. The data in the column capital loss is skewed to the left since the median is greater than the mean. We will remove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

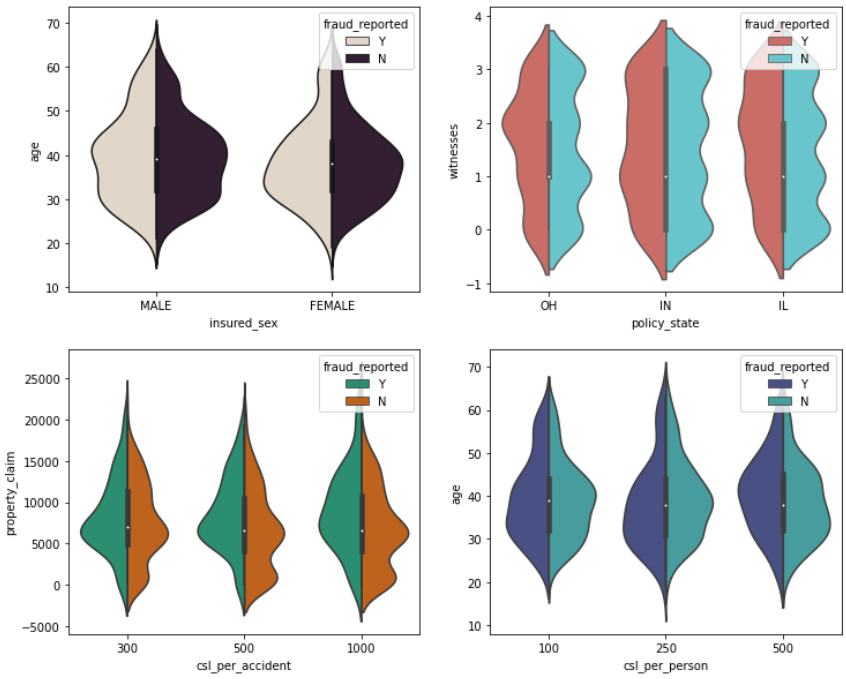
```
# Comparision between two variables
plt.figure(figsize=[18,13])

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']);
```



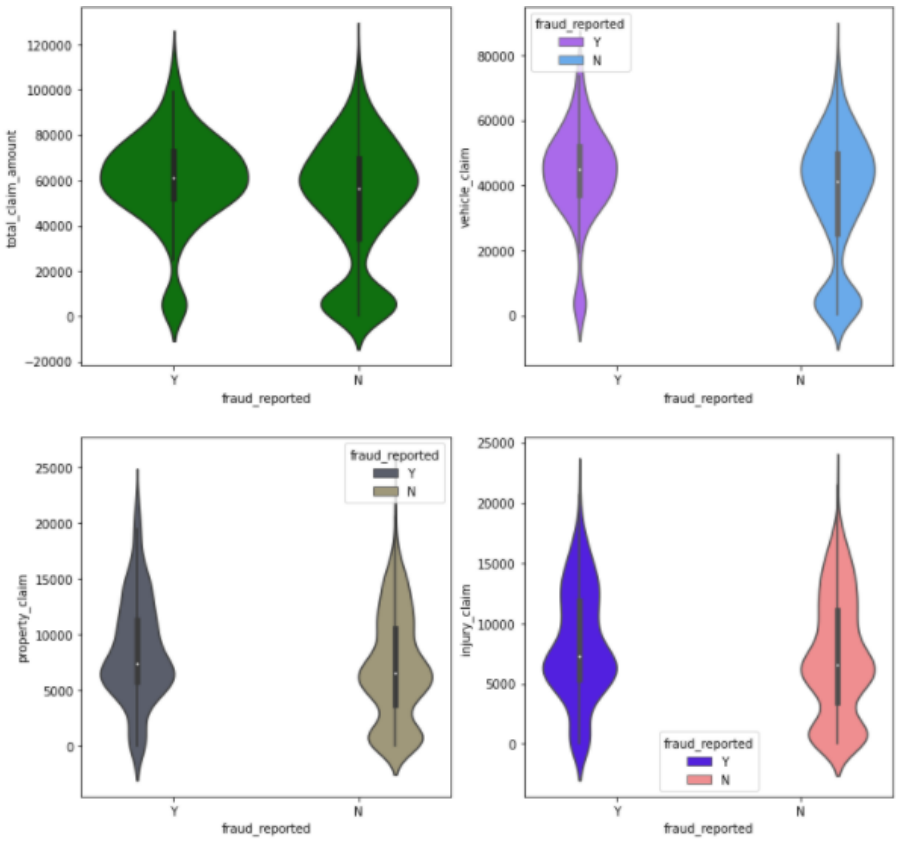


- There is a positive linear relationship between the age and month_as_customer column. As age increases the month_as customers also increases, also the fraud reported is very less in this case.
- In the second graph, we can observe the positive linear relation, as the total claim amount increases, an injury claim also increases.
- The third plot is also the same as the second one as the property claim increases, vehicle claim is also increased.
- In the fourth plot, we can observe the data is scattered and there is not much relation between the features.

```
# 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()
```

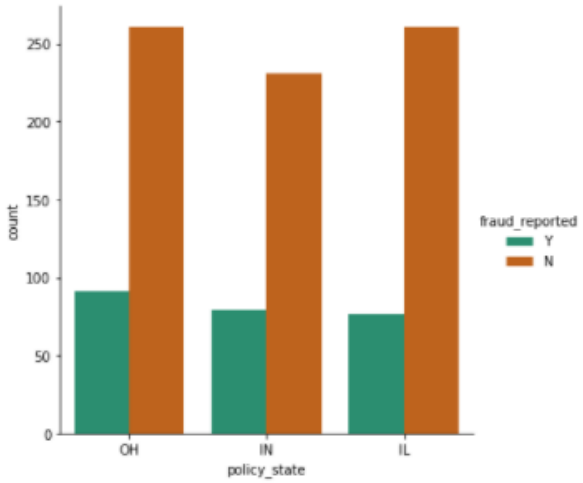



```
# Comparing insured_sex and age
sns.violinplot(x='fraud_reported',y='total_claim_amount',ax=axes[0,0],data=df,color="g")

# 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()
```

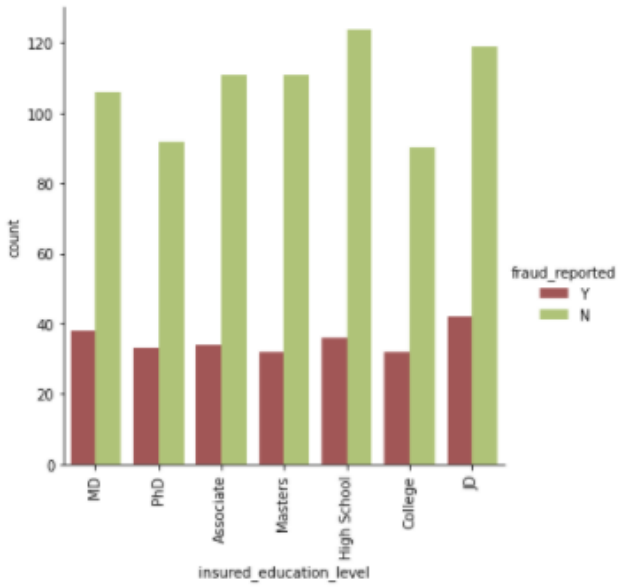


Visualization is a technique whereby comparison and plotting of the data become self-explanatory, which we have seen until now. Moving ahead with some more visualization plots before we can proceed to model building.

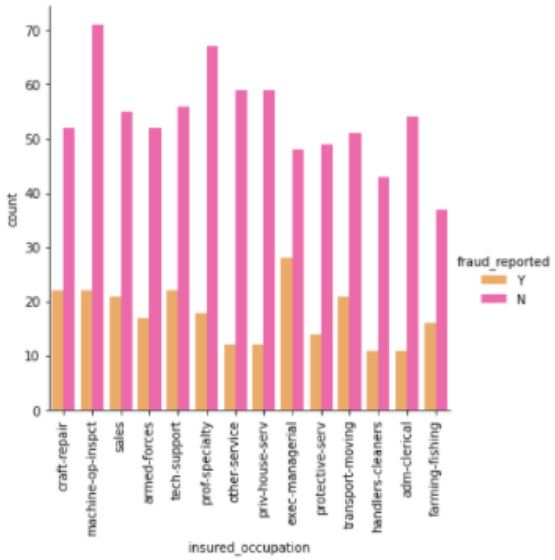
```
# Comparing policy_state and fraud_reported
sns.factorplot('policy_state',kind='count',data=df,hue='fraud_reported',palette="Dark2")
plt.show()
```



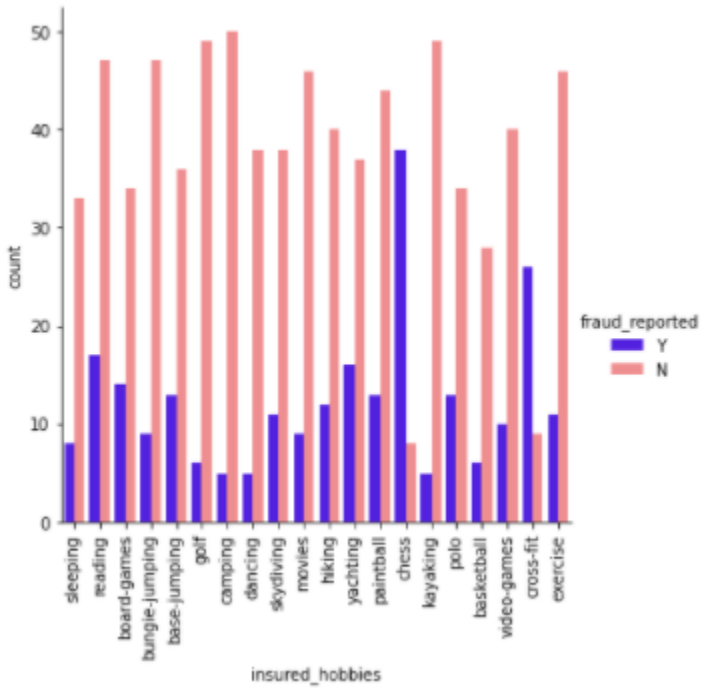

```
# Comparing insured_education_level and fraud_reported

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



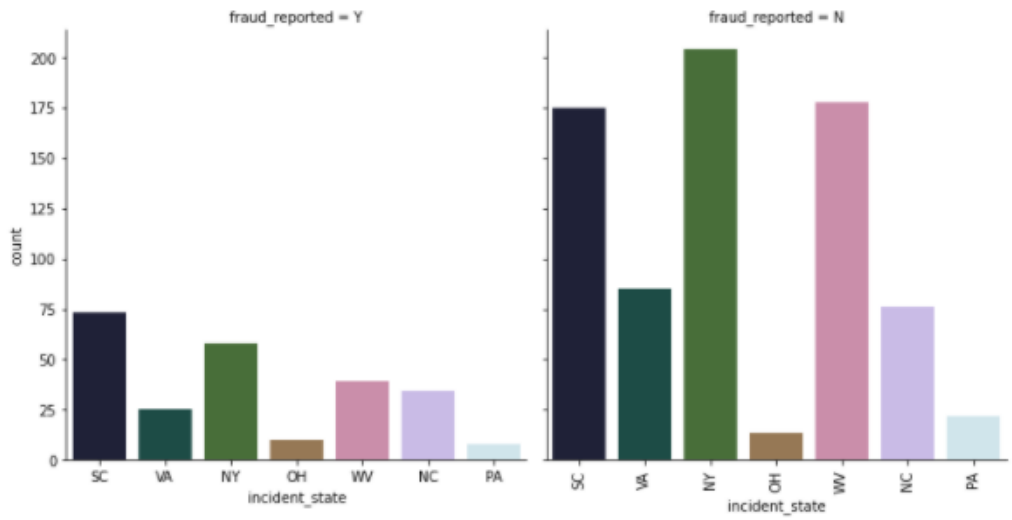
```
# Comparing insured_occupation and fraud_reported
sns.factorplot('insured_occupation',kind='count',data=df,hue='fraud_reported',palette="spring_r")
plt.xticks(rotation=90)
plt.show()
```



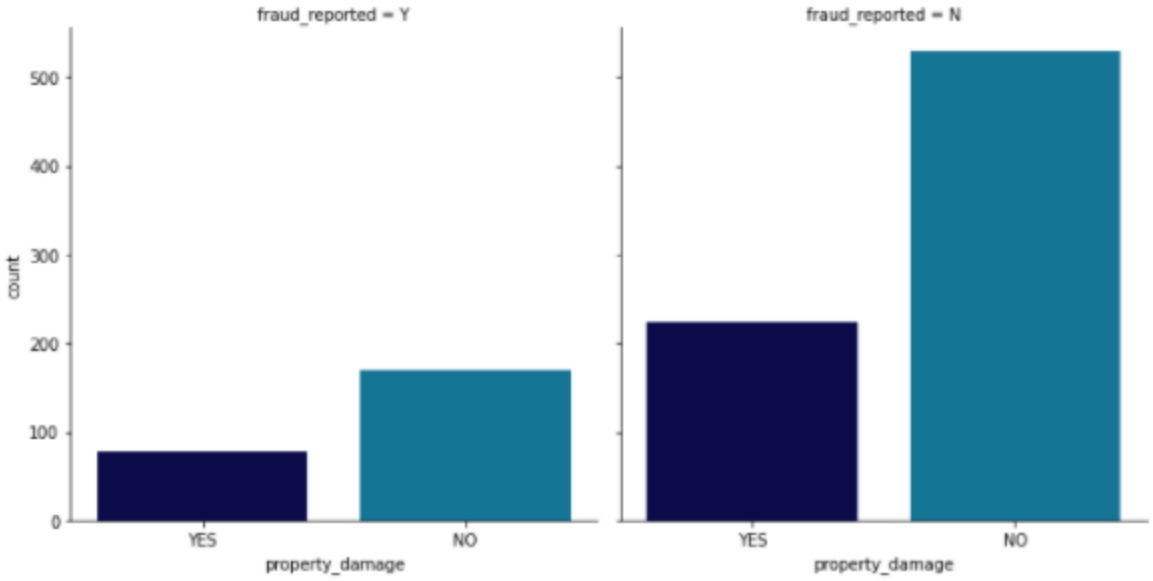

```
# Comparing insured_hobbies and fraud_reported
sns.factorplot('insured_hobbies',kind='count',data=df,hue='fraud_reported',palette="gnuplot2")
plt.xticks(rotation=90)
plt.show()
```




```
# 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()
```



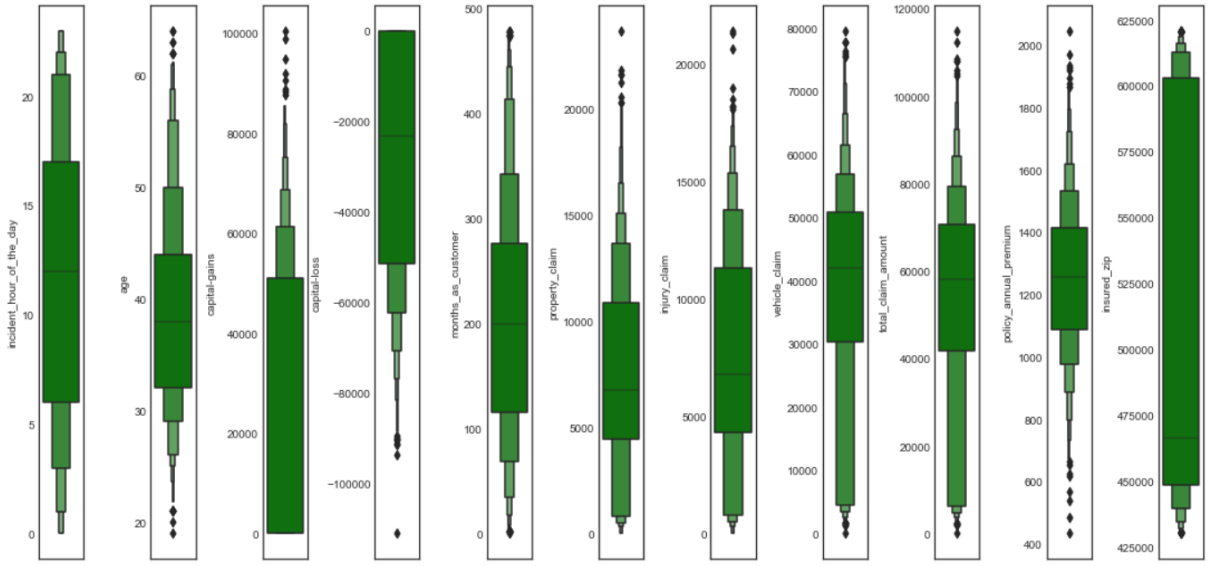

```
# Comparing property_damage and fraud_reported
sns.factorplot('property_damage',kind='count',data=df,col='fraud_reported',palette="ocean")
plt.show()
```



Now we have done the visualization to analyze and understand the data. So in this EDA part, we have looked into various aspects of the dataset, like looking for the null values and imputing, extracting date time, observing the value counts, doing the feature extraction, etc. Now we will be performing another analysis by identifying the outliers and removing them. Along with it we will also look for the skewness of the dataset and remove the skewness.

Identifying the Outliers and Skewness

```
# 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()
```



we have used a box plot to identify the outliers and we can find the outliers in the following columns:

- Age
- policy_annual_premium
- total_claim_amount
- property_claim
- incident_month

These are the numerical columns that contain outliers hence removing the outliers in these columns using the Z-score method.

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

z=np.abs(zscore(features))

z
```

Now that we have removed the outliers, I will proceed to look into the skewness of the data and then remove it.

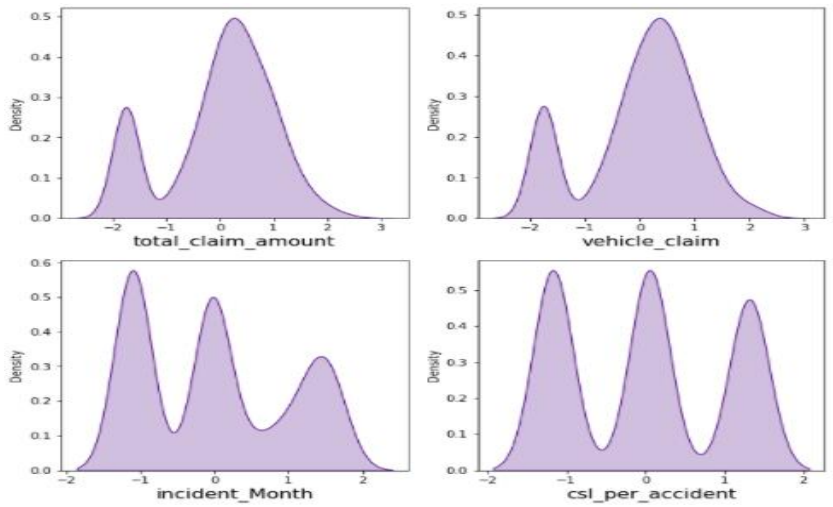
```
# Checking the skewness
new_df.skew().sort_values()
```

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

As we can see that skewness is present in the dataset, hence I am using the yeo-johnson method to remove the skewness.

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


scaler = PowerTransformer(method='yeo-johnson')
'''
parameters:
method = 'box-cox' or 'yeo-johnson'
'''
```

- Now we have removed the skewness and the data looks normally distributed.



Now we have completed our analysis of the dataset and also cleaned the data so that we can build a model.
We have seen above that some problems still exist in the dataset. We have seen that the dataset has both numerical and categorical data. The model only understands numerical data; hence we will encode the data. we have seen that there can be some multi-collinearity, which we will see through a heatmap and also further try to remove it. Again we have also seen that the target variable is imbalanced, hence will fix it by oversampling. And finally, we will scale the data so that it is ready to be trained and tested.

Encoding the data

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

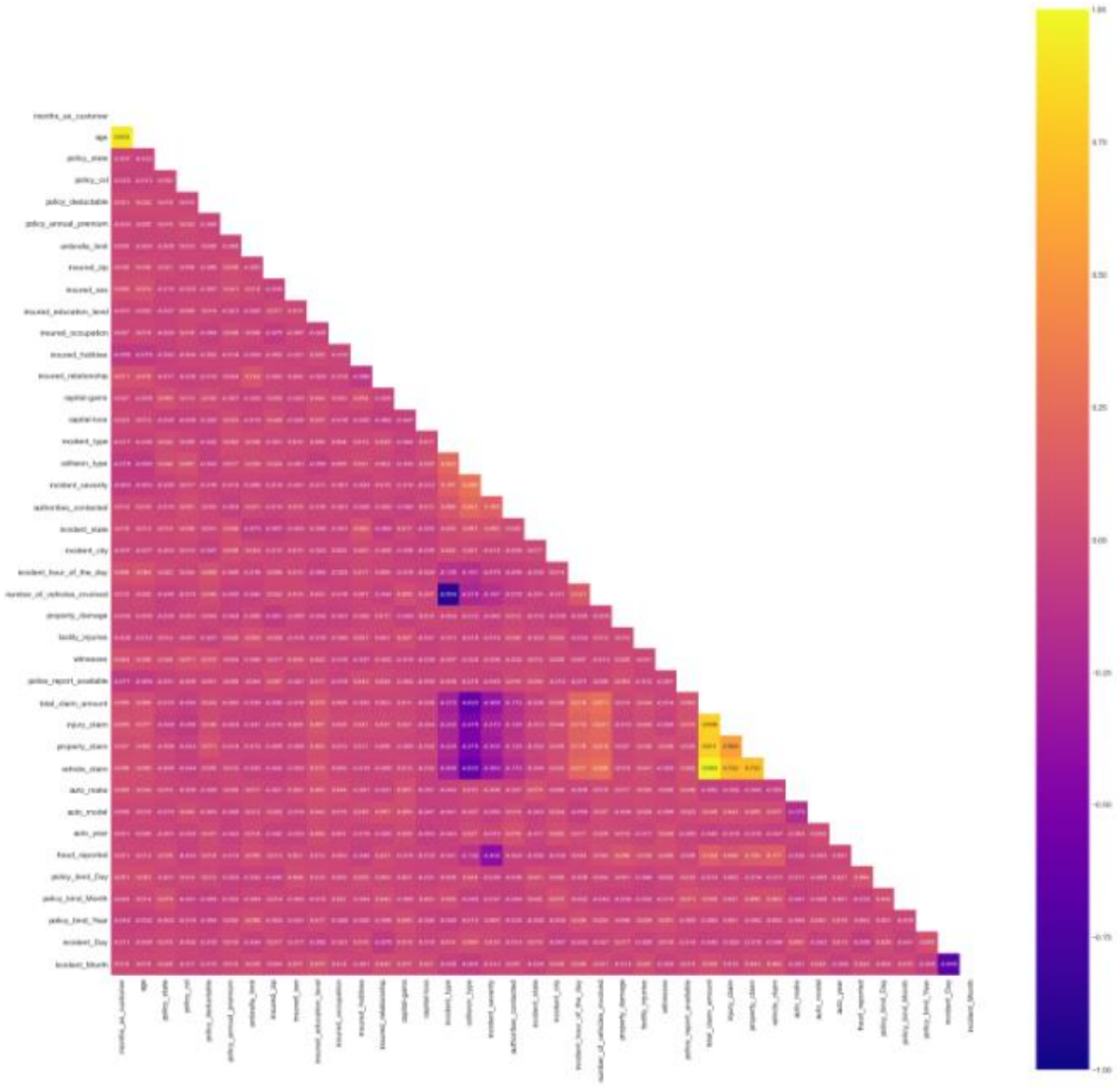
```
new_df[categorical_col].head()
```

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

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

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

```
# 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},cm
plt.yticks(rotation=0);
```



This heatmap shows the correlation matrix by visualizing the data. We can observe the relation between one feature to another.
There is very less correlation between the target and the label. We can observe that most of the columns are highly correlated with each other which leads to the multicollinearity problem. We will check the VIF value to overcome this multicollinearity problem.

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"]
```

Scaling the DataSet

- Feature Scaling using Standard Scalarization

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

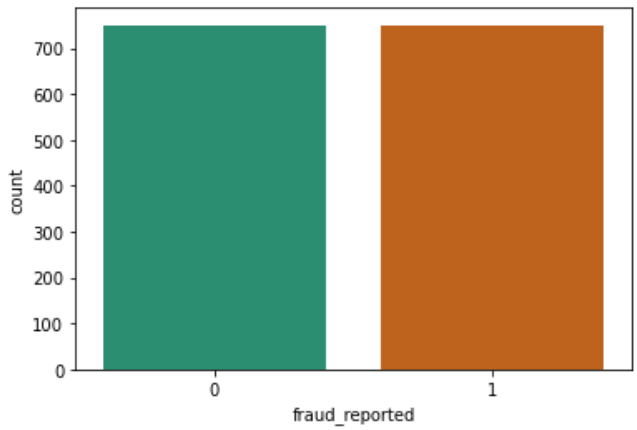
Checking Multi-colinearity using VIF

```
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
```

- It is observed that some columns have VIF above 10 which mean they are causing multicollinearity problem. Let's drop the feature having a high VIF value amongst all the columns.
- I have dropped the total_claim_amount and csl_per_accident features with a colinearity of more than 10, and now we have removed the problem.
- We had earlier identified another problem of imbalanced data in the target variable, let us treat it.

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

- As we have treated the oversampling issue using SMOTE, now the data looks good.



- Finally, we have got into the position where we will start building the model.
- First, let's find the best random state in which we can build the model.

(Random state ensures that the splits that you generate are reproducible. Scikit-learn uses random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.)

```
maxAccu=0
maxRS=0

for i in range(1, 1000):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=i)
    lr=LogisticRegression()
    lr.fit(X_train, Y_train)
    pred = lr.predict(X_test)
    acc_score = (accuracy_score(Y_test, pred))*100

    if acc_score>maxAccu:
        maxAccu=acc_score
        maxRS=i

print("Best accuracy score is", maxAccu,"on Random State", maxRS)
```

```
Best accuracy score is 78.91891891891892 on Random State 49
```

Here we have used the RandomForestClassifier to find the best random state, as we have got an accuracy score of 79% (pretty good), at the random state of 49. Let's use this random state to build our models.
Before doing that, let us split the dataset into train and test using train_test_split.

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

Model Building

```
# Random Forest Classifier

model=RandomForestClassifier(max_depth=15, random_state=111)
classify(model, X, Y)
```

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

           0       0.88      0.94      0.91       170
           1       0.95      0.89      0.92       200

    accuracy                           0.91       370
   macro avg       0.91      0.92      0.91       370
weighted avg       0.92      0.91      0.91       370

Accuracy Score: 91.35135135135135
Cross Validation Score: 86.74965643609711

Accuracy Score - Cross Validation Score is 4.601694915254242
```

Created the Random Forest Classifier Model and checked for it's evaluation metrics.

The first model was built using RandomForestClassifier, which gave an accuracy score of 91%, however, we are hungry data scientists and will not be satisfied with only one model. We will try various models and see what accuracy score we get.

```
# Support Vector Classifier

model=SVC(C=1.0, kernel='rbf', gamma='auto', random_state=42)
classify(model, X, Y)
```

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

           0       0.88      0.92      0.90       170
           1       0.93      0.90      0.91       200

    accuracy                           0.91       370
   macro avg       0.91      0.91      0.91       370
weighted avg       0.91      0.91      0.91       370

Accuracy Score: 90.81081081081082
Cross Validation Score: 84.78721942281264

Accuracy Score - Cross Validation Score is 6.023591387998181
```

Created the Support Vector Classifier Model and checked for it's evaluation metrics.

- With Support Vector Classifier we got an accuracy score of 91%.

```
# Logistic Regression

model=LogisticRegression()
classify(model, X, Y)
```

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

           0       0.75      0.78      0.77       170
           1       0.81      0.79      0.79       200

    accuracy                           0.78       370
   macro avg       0.78      0.78      0.78       370
weighted avg       0.78      0.78      0.78       370

Accuracy Score: 78.10810810810811
Cross Validation Score: 73.61818598259278

Accuracy Score - Cross Validation Score is 4.489922125515335
```

Created the Logistic Regression Model and checked for it's evaluation metrics.

- With logistic Regression, we got an accuracy score of 78%.

```
# LGBM Classifier

model=lgb.LGBMClassifier()
classify(model, X, Y)
```

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

           0       0.89      0.92      0.90       170
           1       0.93      0.90      0.91       200

    accuracy                           0.91       370
   macro avg       0.91      0.91      0.91       370
weighted avg       0.91      0.91      0.91       370

Accuracy Score: 90.81081081081082
Cross Validation Score: 87.28905176362804

Accuracy Score - Cross Validation Score is 3.5217590471827833
```

Created the LGBM Classifier Model and checked for it's evaluation metrics.

- With LGBM we got an accuracy score of 90%.

```
# XGB Classifier

model=xgb.XGBClassifier(verbosity=0)
classify(model, X, Y)
```

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

           0       0.90      0.92      0.91       170
           1       0.93      0.91      0.92       200

    accuracy                           0.92       370
   macro avg       0.92      0.92      0.92       370
weighted avg       0.92      0.92      0.92       370

Accuracy Score: 91.62162162162161
Cross Validation Score: 87.55886394869445

Accuracy Score - Cross Validation Score is 4.062757672927162
```

Created the XGB Classifier Model and checked for it's evaluation metrics.

- With XGB Classifier we got an accuracy score of 88%.

```
# Extra Trees Classifier

model=ExtraTreesClassifier()
classify(model, X, Y)
```

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

           0       0.92      0.95      0.93       170
           1       0.96      0.93      0.94       200

    accuracy                           0.94       370
   macro avg       0.94      0.94      0.94       370
weighted avg       0.94      0.94      0.94       370

Accuracy Score: 93.78378378378378
Cross Validation Score: 89.99404489234998

Accuracy Score - Cross Validation Score is 3.7897388914337995
```

Created the Extra Trees Classifier Model and checked for it's evaluation metrics.

With this model, ExtraTreesClassifier we have got an accuracy score of 93%, which is better than RandomForestClassifier.

Before we can announce the best model, we always have to make sure that the model is not overfitting; hence we will perform cross-validation of all the models built.

```
# cv score for Random Forest Classifier
print('Random Forest Classifier:',cross_val_score(rfc,x,y,cv=5).mean())

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

# cv score for Gradient Boosting Classifier
print('Gradient Boosting Classifier:',cross_val_score(gb,x,y,cv=5).mean())

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

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

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

```
Random Forest Classifier: 0.8673333333333334
Support Vector Machine Classifier: 0.868
Gradient Boosting Classifier: 0.8726666666666667
AdaBoosting Classifier: 0.8433333333333334
Bagging Classifier: 0.8646666666666667
Extra Trees Classifier: 0.9126666666666667
```

After the cross-validation, we can see that ExtraTreesClassification is the best fit model.

Now, that we have found the best fit model, let us perform some HyperParameterTuning to improve the performance of the model.

```
# 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]}
```

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

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

```
GCV.best_params_
```

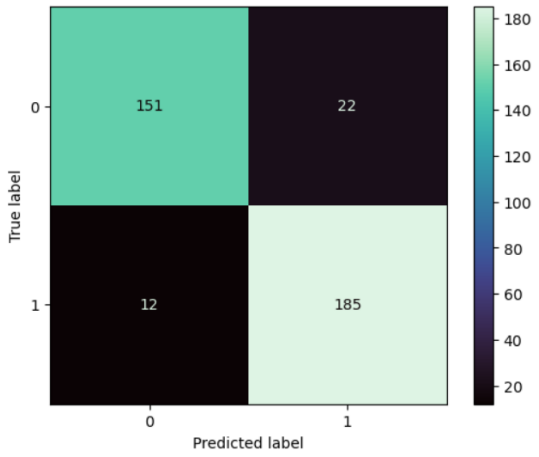
```
{'criterion': 'gini',
 'max_depth': 20,
 'max_features': 'log2',
 'n_estimators': 300,
 'n_jobs': -2}
```

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

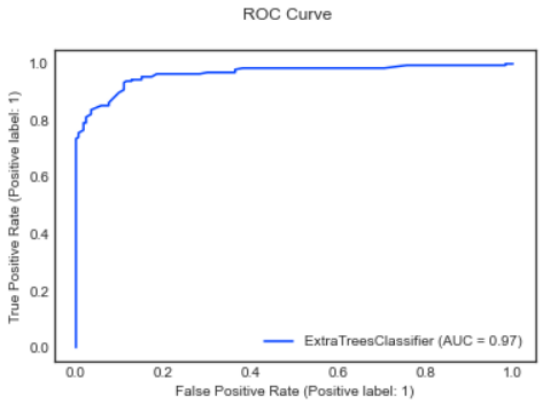
```
Insurance = ExtraTreesClassifier(criterion='gini', max_depth=20, max_features='log2', n_estimators=300, n_jobs=-2)
Insurance.fit(x_train, y_train)
pred = Insurance.predict(x_test)
acc=accuracy_score(y_test,pred)
print(acc*100)
```

```
92.0
```

We have built our final model and we can see that the accuracy scores have increased by 1% from the cross-validation score.



- This is the confusion matrix for the model.
- Plotting and AUCROC curve for the final model.



- So here we can see that the area under the curve is quite good for this model.

Saving the model

```
# Saving the model using .pkl
import joblib
joblib.dump(Insurance,"Insurance_claim_Fraud_Detection.pkl")
```

```
['Insurance_claim_Fraud_Detection.pkl']
```

Predicting the model

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

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

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

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

Conclusion:-

At the beginning of the blog we discussed the lifecycle 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 can improve our model accuracy, for instance in this model the accuracy remained the same.

Using this machine Learning Model we people 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