Diving Into Data Analysis: A blog by Mohd Sharik Hashmi

Insurance Claim Fraud Detection Project

In association with Data Trained Academy: Batch: 1838

Introduction:

Fraud is one of the largest and most well-known problems that insurers face in the insurance industry. This article focuses on claim data of automobile insurance.

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 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.

THANKS … to Data Science and Machine Learning, which has been very useful in many industries that have managed to bring accuracy or detect negative incidents. Here in this blog, I have created a Machine Learning model to detect if the claim is fraudulent or not. Here various features has been used like, insured information, insured persons, personal details and the incident information. In total the dataset have 40 features. So using all these previously accuired information and analysis done with the data I have achieved a good model that has 93.33% accuracy. So let's see what are the steps involved to attain this accuracy.

Various visualization techniques have also been used to understand the co-linearity and importance of the features.

Note: Various jargons are used in the article assuming the fact that the reader is aware of the language used in data science.

Hardware & Software Requirements & Tools Used:

Hardware required:

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

Software requirement:

- Jupiter Notebook

Libraries Used:

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

Problem Definition:

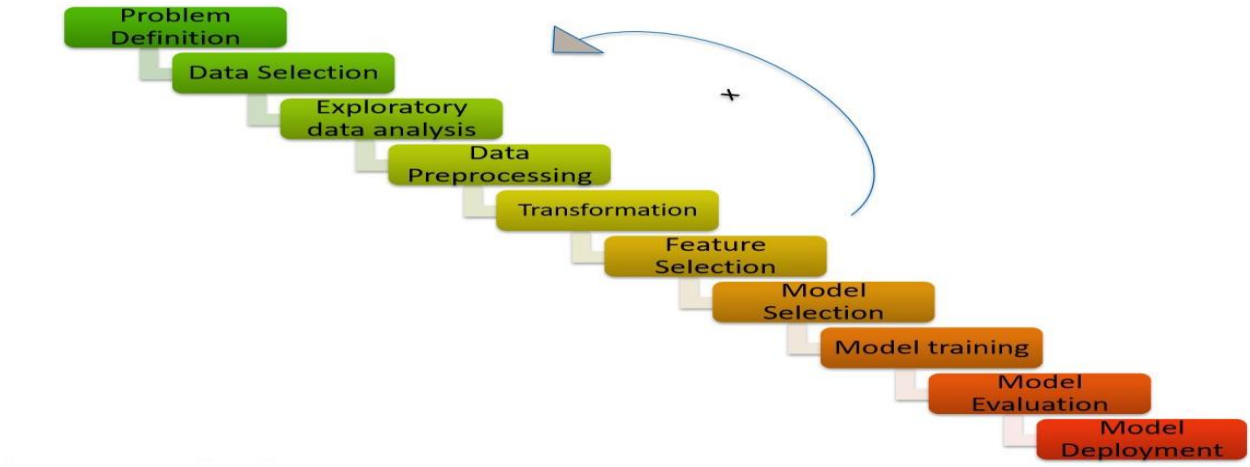
Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem. In this project; we are provided a dataset which has the details of the insurance policy along with the customer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

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

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

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

In order to build a Machine Learning Model, we have a Machine Learning Life Cycle that every Machine Learning Project 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, lets import the necessary libraries and proceed further.

Importing the necessary Libraries:

To analyze the dataset or even to import the dataset, we have imported all the necessary libraries as shows below.

Pandas has been used to import the dataset and also in creating data frames.
Seaborn and Matplotlib has been used for visualization

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

```
#Importing the necessary Libraries

# Linear algebra
import numpy as np

import warnings
warnings.filterwarnings('ignore')

#data processing
import pandas as pd

#data visualization
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib import style

#ALogithms
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,roc_auc_score
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier,ExtraTreesClassifier


import joblib
```

Importing the Dataset

Let's import the dataset first.

```
In [9]: df=pd.read_csv('Automobile_insurance_fraud.csv')
        df
```

I have imported the dataset which was in "csv" format as "df". Below is how the dataset looks.

insured_zip	...	police_report_available	total_claim_amount	injury_claim	property_claim	vehicle_claim	auto_make	auto_model	auto_year	fraud_reported	_c39
466132	...	YES	71610	6510	13020	52080	Saab	92x	2004	Y	NaN
468176	...	?	5070	780	780	3510	Mercedes	E400	2007	Y	NaN
430632	...	NO	34650	7700	3850	23100	Dodge	RAM	2007	N	NaN
608117	...	NO	63400	6340	6340	50720	Chevrolet	Tahoe	2014	Y	NaN
610706	...	NO	6500	1300	650	4550	Accura	RSX	2009	N	NaN

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

Also by doing a simple code 'df.shape' we also figured out how many rows and columns we have. We have got the result 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 and the first lesson of a data scientist is 'Data is Crucial' hence proceeded will all the data.

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

```
(1000, 40)
```

As per the lifecycle of the machine learning model we have already completed point 1 and 2. Now lets move on to the point 3,4, 5 and 6 which is the most crucial part of any machine learning model, as the best way possible data will be analyzed and cleaned the better model accuracy we will get, or the model can remain over fitting or under fitting. We will discuss further why all the steps are used.

Exploratory Data Analysis and Data Preparation:

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

Let's start with checking shape, unique values, value counts, info etc.
After doing the analysis if we find any unnecessary columns in the dataset we can drop those columns.

```
#checking the information of data
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
```

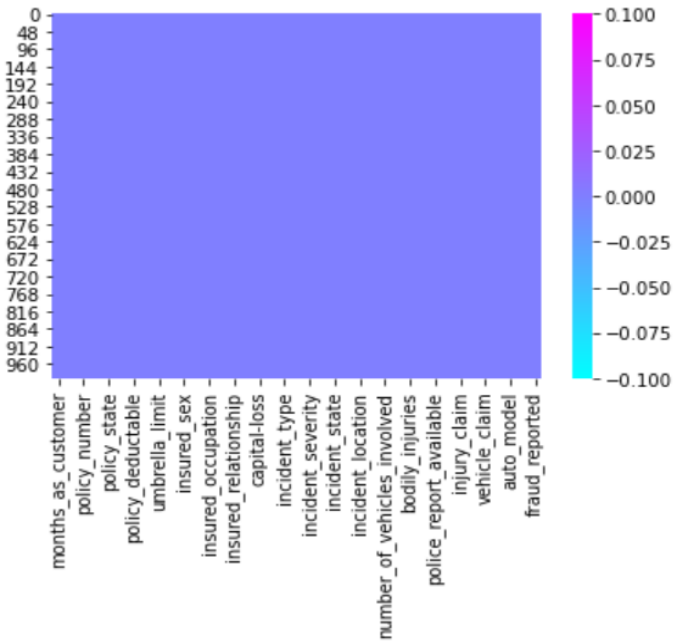
```
#checking the unique values
df.nunique().to_frame('NO. of Unique values')
```

```
#checking the value counts
for i in df.columns:
    print(df[i].value_counts())
    print('\n')
```

we are checking for the null values and further will mention all the observations.

```
: #heatmap for null values
  sns.heatmap(df.isnull(),cmap='cool')
```

```
: <AxesSubplot:>
```



Observations :

- First we can see that we do not have any null values in the dataset.
- Second, the dataset contains 3 different types of data namely integer data type, float data type and object data type.
- Third, after analyzing it is seen that c39 column has only entries those are all NaN. Keeping all entries NaN is useless hence dropping that column

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

- Fourth, we can observe the columns policy_number and incident_location have 1000 unique values which means they have only one value count. So it not required for the prediction so we can drop it.

```
#Dropping The columns
df.drop('policy_number',axis=1,inplace=True)
```

```
df.drop('incident_location',axis=1,inplace=True)
```

- Fifth, by looking at the value counts of each column we can realize that the columns umbrella_limit, capital-gains and capital-loss contains more zero values around 79.8%,

50.8% and 47.5%. I am keeping the zero values in capital_gains and capital_loss columns as it is. Since the umbrella_limit columns has more that 70% of zero values, let's drop that column.

```
#Dropping the columns
df.drop('umbrella_limit',axis=1,inplace=True)
```

- Sixth, the column insured_zip is the zip code given to each person. If we take a look at the value count and unique values of the column insured_zip, it contains 995 unique values that mean 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 the column
df.drop('insured_zip',axis=1,inplace=True)
```

Proceeding to Feature Extraction:

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

```
#Dropping The columns
df.drop('policy_number',axis=1,inplace=True)
```

```
df.drop('incident_location',axis=1,inplace=True)
```

Now that we have converted object data type into datetime data type. Now let's extract Day, Month and Year from both the columns

```
# Extracting Day, Month and Year column from policy_bind_date
df['policy_bind_date']=pd.to_datetime(df['policy_bind_date'])
df['policy_bind_day'] = df['policy_bind_date'].dt.day
df['policy_bind_month'] = df['policy_bind_date'].dt.month
df['policy_bind_year'] = df['policy_bind_date'].dt.year

# Extracting Day, Month and Year column from incident_date
df['incident_day'] = df['incident_date'].dt.day
df['incident_month'] = df['incident_date'].dt.month
df['incident_year'] = df['incident_date'].dt.year
```

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

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

```
df.drop('incident_date',axis=1,inplace=True)
```

Again, from the features we can see that the policy_csl column is showing as 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 colums and then will convert their object data type into integer data type.

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

```
#cinverting the obh=j in to int
df['csl_per_person']=df['csl_per_person'].astype('int64')
df['csl_per_accident']=df['csl_per_accident'].astype('int64')
```

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

After extracting we have dropped the policy_csl feature.
Here we also have extracted age of the vehicle on the basis of auto year by assuming the data is collected in the year 2018 so we can drop this column.

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

Moving on to Imputation:

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

So, let's begin……

```
#checking which column contain'?'
df[df.columns[(df=='?').any()]].nunique()
```

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

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

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

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

```
# 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")
```

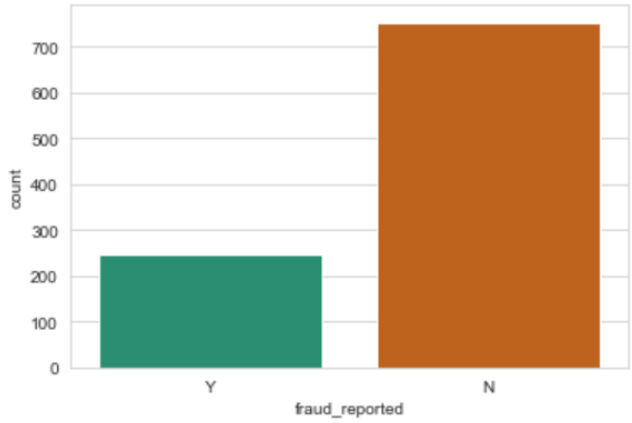
Preparing for Visualization

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

```
# 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")

# 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)
print("\n")
```

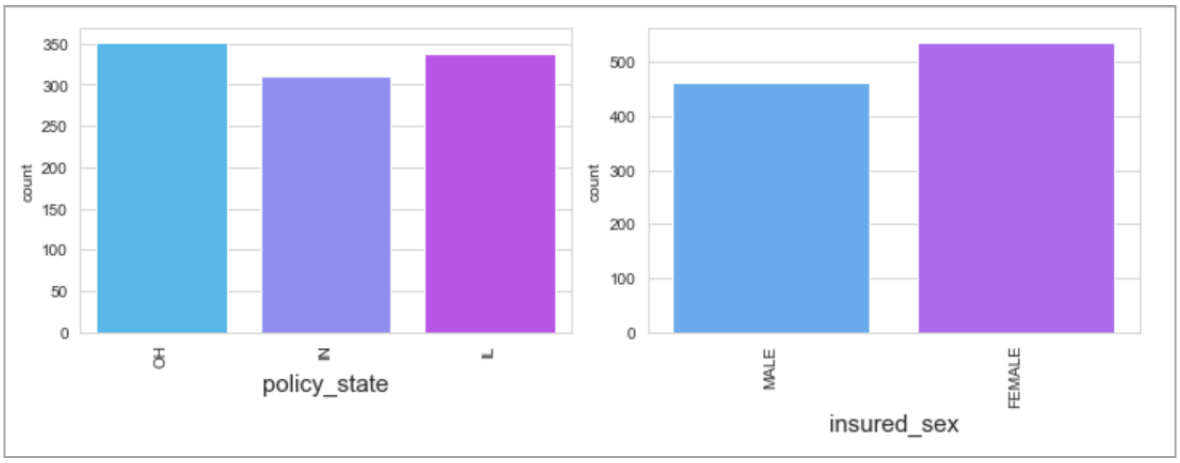
Visualization

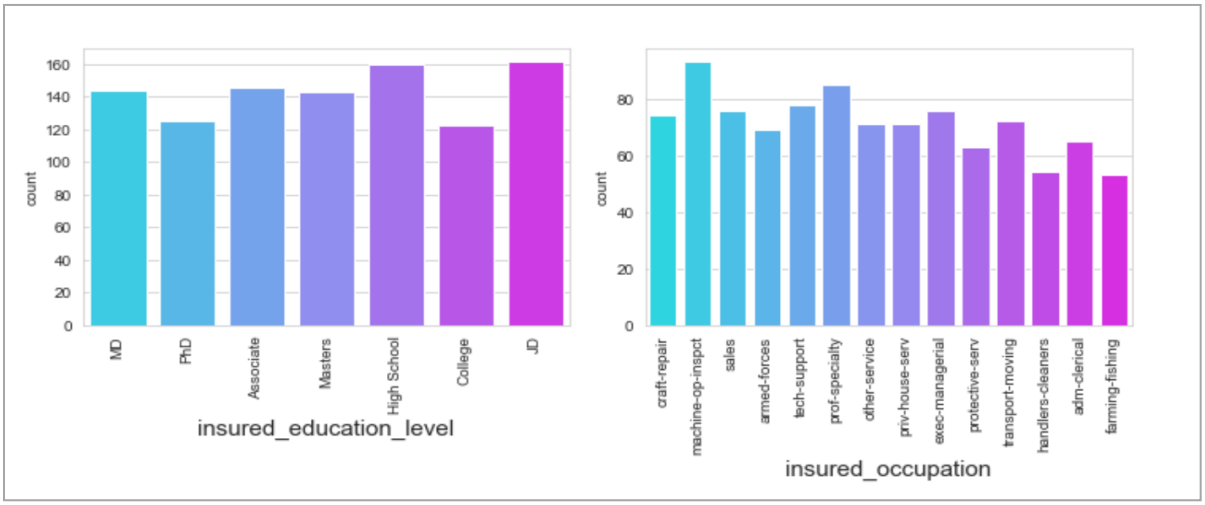
```
print(df["fraud_reported"].value_counts())
sns.countplot(df["fraud_reported"],palette="Dark2")
plt.show()
```

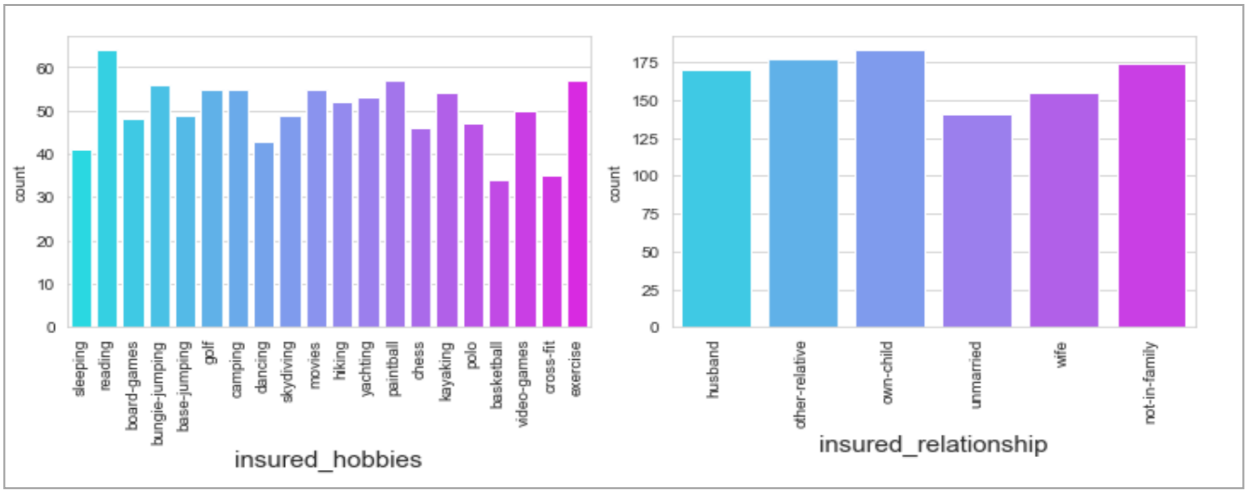


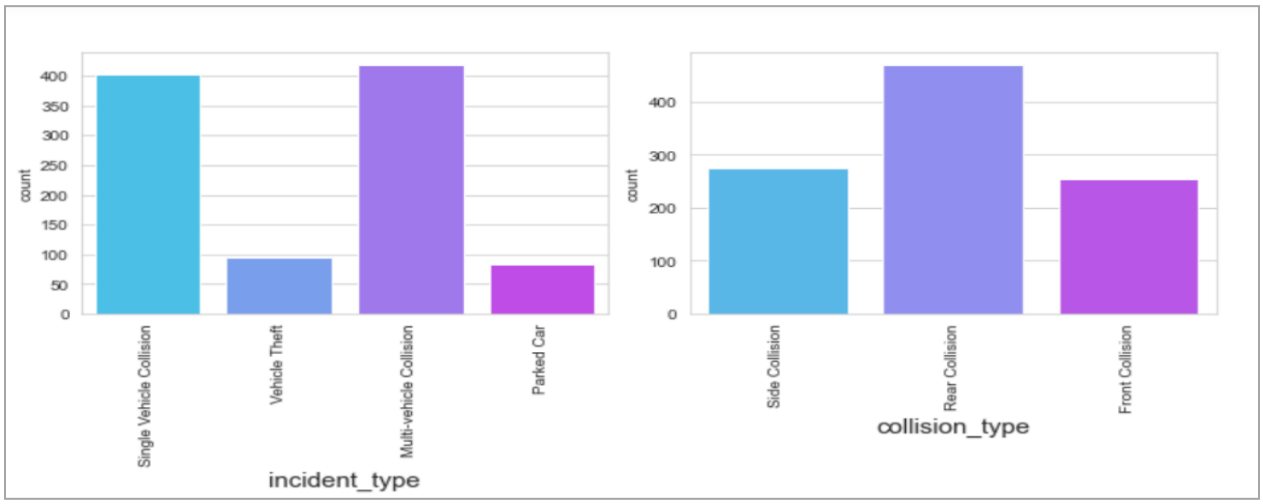
By looking into the plot we can observe that the count of "N" is high compared to "N". Which means here we can assume that "Y" stands for "Yes" that is the insurance is fraudulent and "N" stands for "No" means the insurance claim is not fraudulent. Here most of the insurance claims have not reported as fraudulent. Since it is our target column, it indicates the class imbalance issue. We will balance the data using oversampling method in later part.

```
plt.figure(figsize=(10,35))
plotnumber=1
for col in categorical_col:
    if plotnumber<=8:
        ax = plt.subplot(8,2,plotnumber)
        sns.countplot(df[col],palette="cool")
        plt.xticks(rotation=90)
        plt.xlabel(col,fontsize=15)
    plotnumber+=1
plt.tight_layout()
plt.show()
```





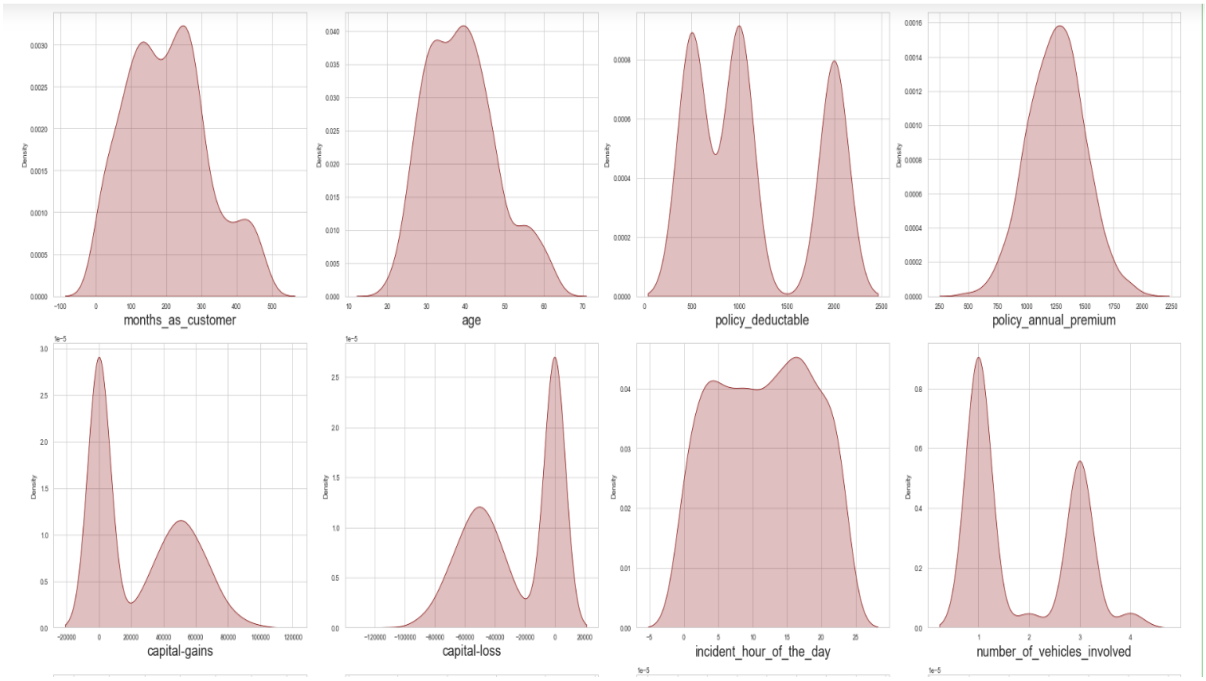


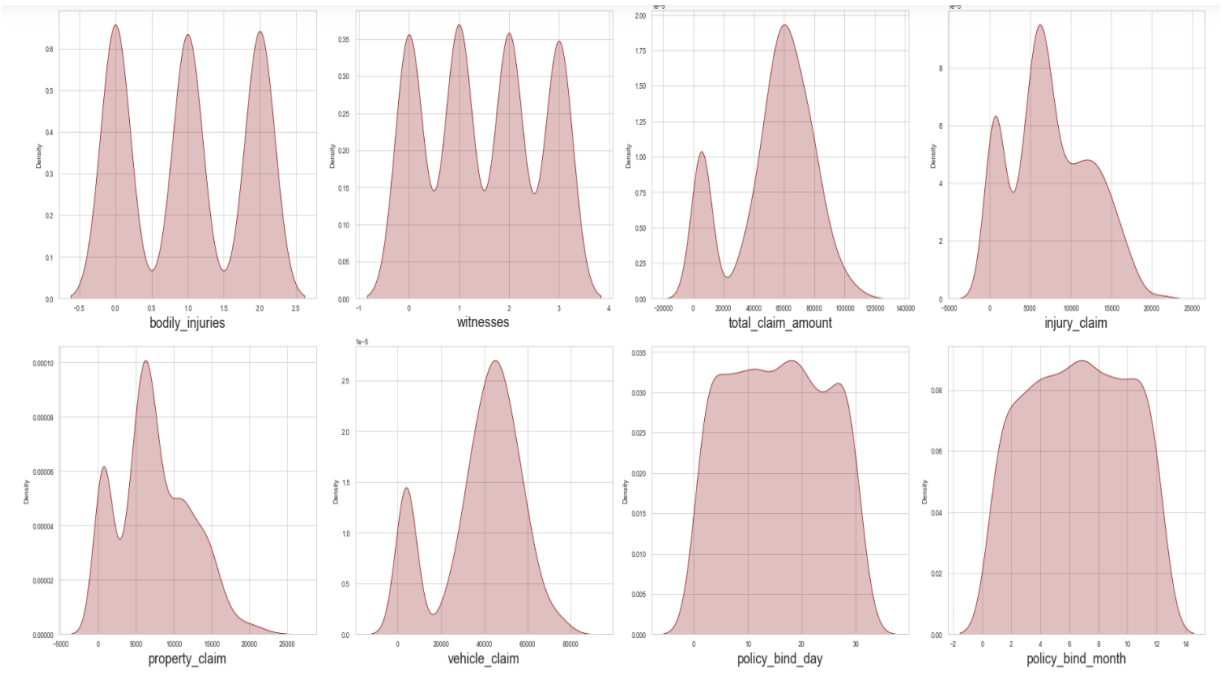


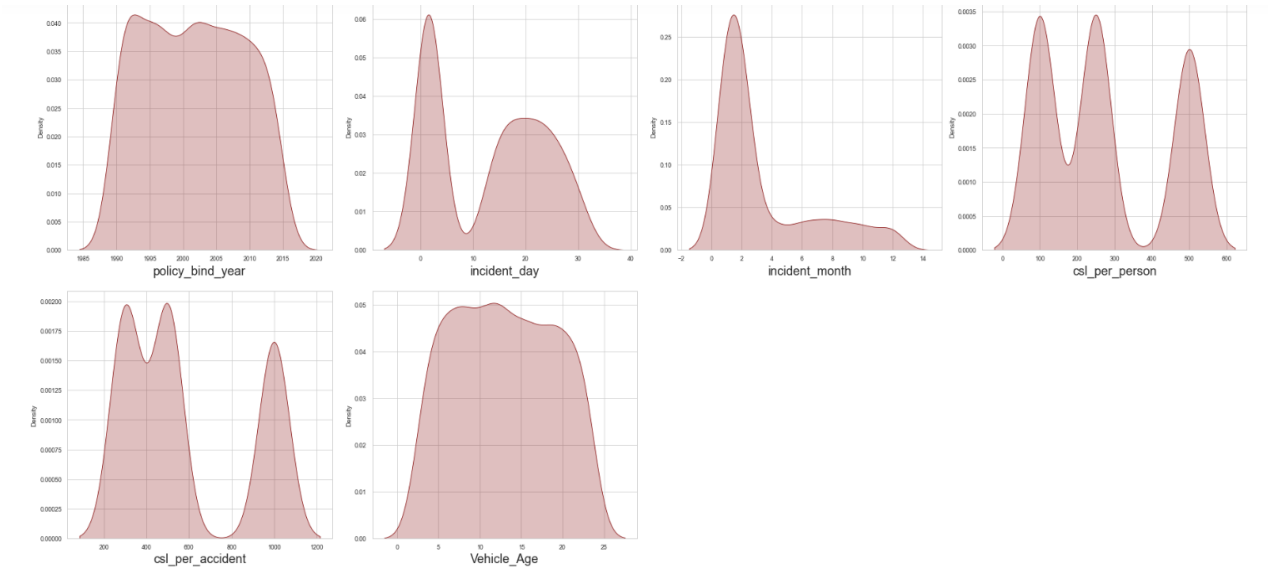
By looking into the count plots we can observe the following things:

- In Policy_state OH and L cover almost same data.
- In Insured_sex Female is covering the large data then Male
- In Insured_education_level High School covering the large data followed by JD
- In the insured occupation we can observe most of the data is covered by machine operation inspector followed by professional speciality.
- With respect to insured hobbies, we can notice reading covered the highest data followed by exercise. And other categories have the average counts.
- The incident_type Multi-vehicle Collision count is high followed by single Vehicle collision.
- In collision_type Rate collision count is high.

```
#checking hoe the data has been distributed


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='maroon',hist=False,kde_kws={'shade':True})
        plt.xlabel(column,fontsize=18)
    plotnumber+=1
plt.tight_layout()
```

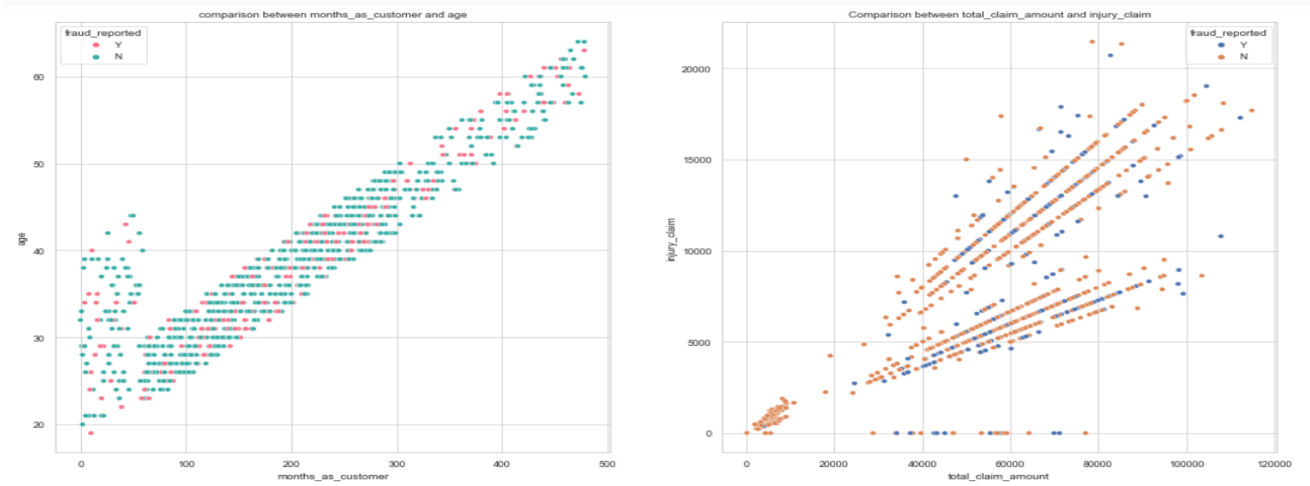

months_as_customer
age
policy_deductable
policy_annual_premium
capital-gains
capital-loss
incident_hour_of_the_day
number_of_vehicles_involved

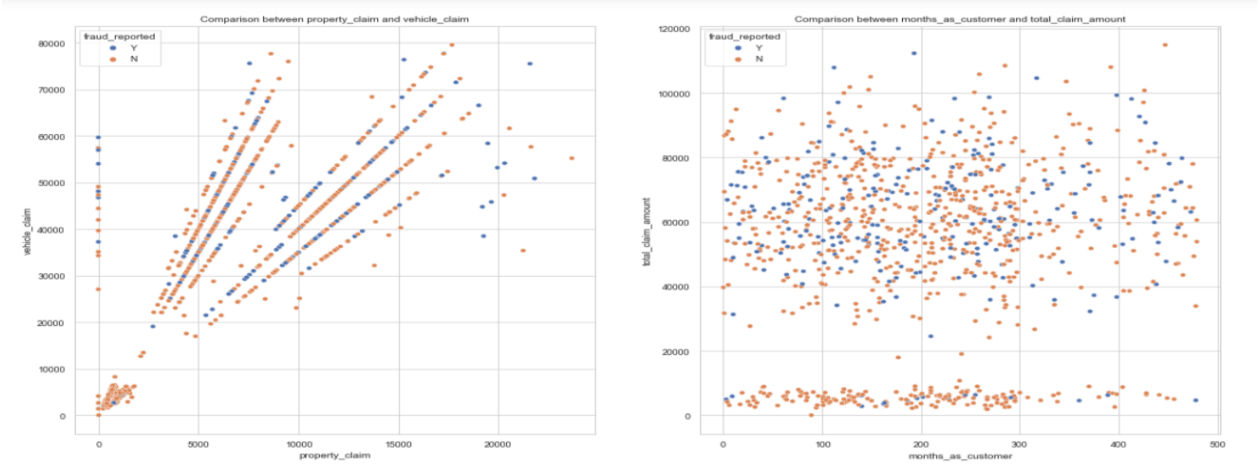

bodily_injuries
witnesses
total_claim_amount
injury_claim
property_claim
vehicle_claim
policy_bind_day
policy_bind_month



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

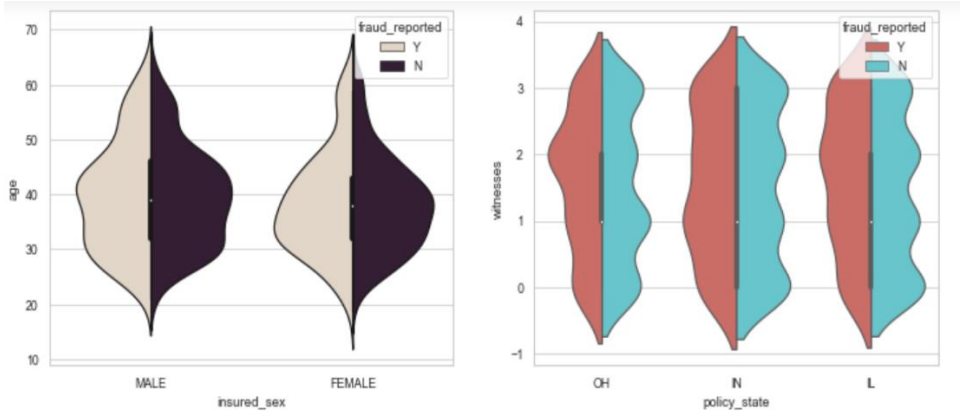
```
#comparison between two variables
plt.figure(figsize=(18,20))
plt.subplot(2,2,1)
plt.title('comparison between months_as_customer and age')
sns.scatterplot(df['months_as_customer'],df['age'],hue=df['fraud_reported'],palette='husl')

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

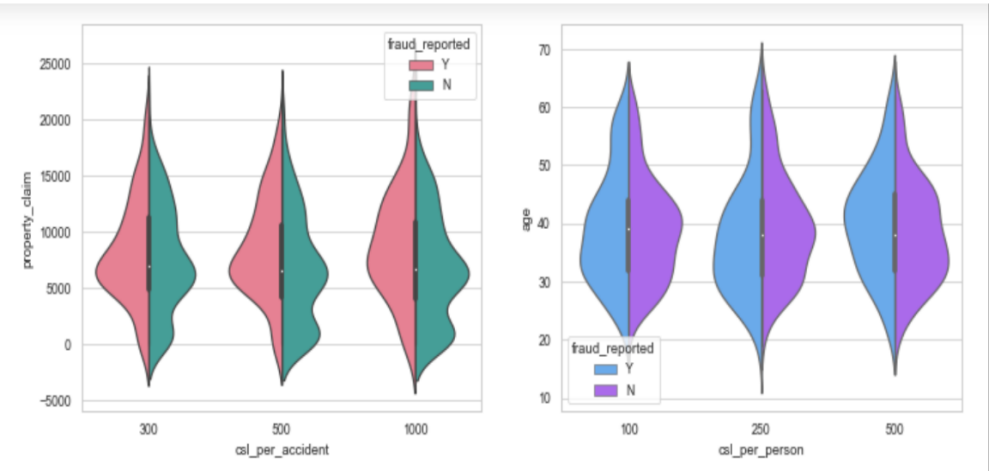




- There is a positive linear relation between age and month_as_customer column. As age increases the month_as customers also increases, also the fraud reported is very less in this case.
- In the second graph we can observe the positive linear relation, as total claim amount increases, injury claim is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

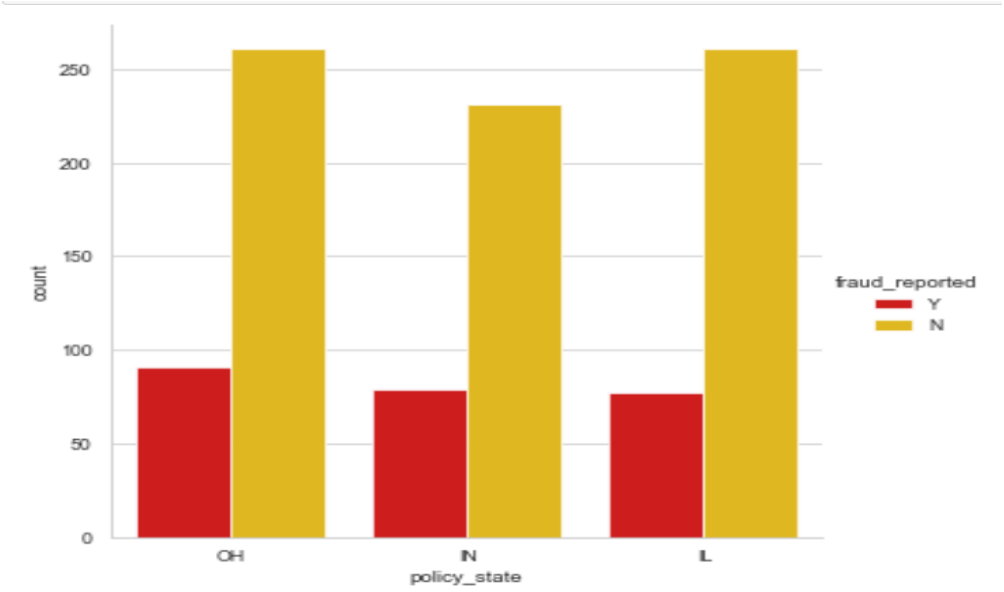
```
fig,axes=plt.subplots(2,2,figsize=(12,10))

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

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

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

# 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="cool")
plt.show()
```





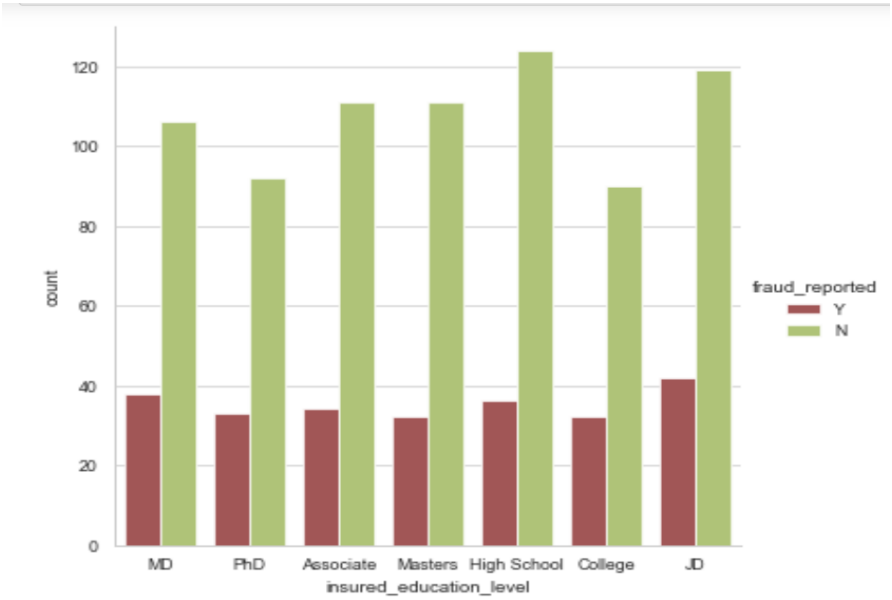
Visualization is a technique where by comparison and plotting the data becomes self explanatory, which we have seen until now. Moving ahead with some more visualization plots before we can proceed to model building.

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



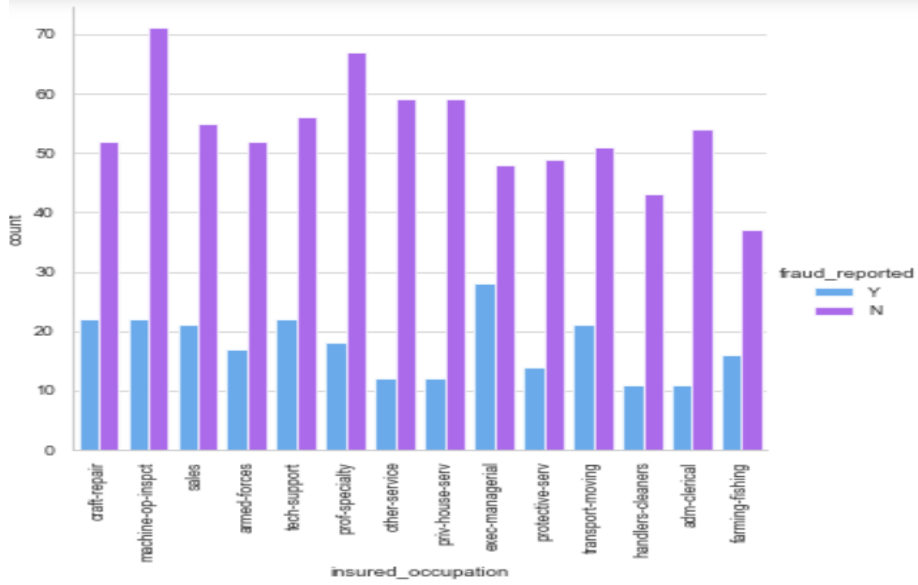
Fraud report is high in "OH" policy_state.

```
# Comparing insured_education_level and fraud_reported
sns.factorplot('insured_education_level',kind='count',data=df,hue='fraud_reported',palette='tab20b_r')
plt.show()
```



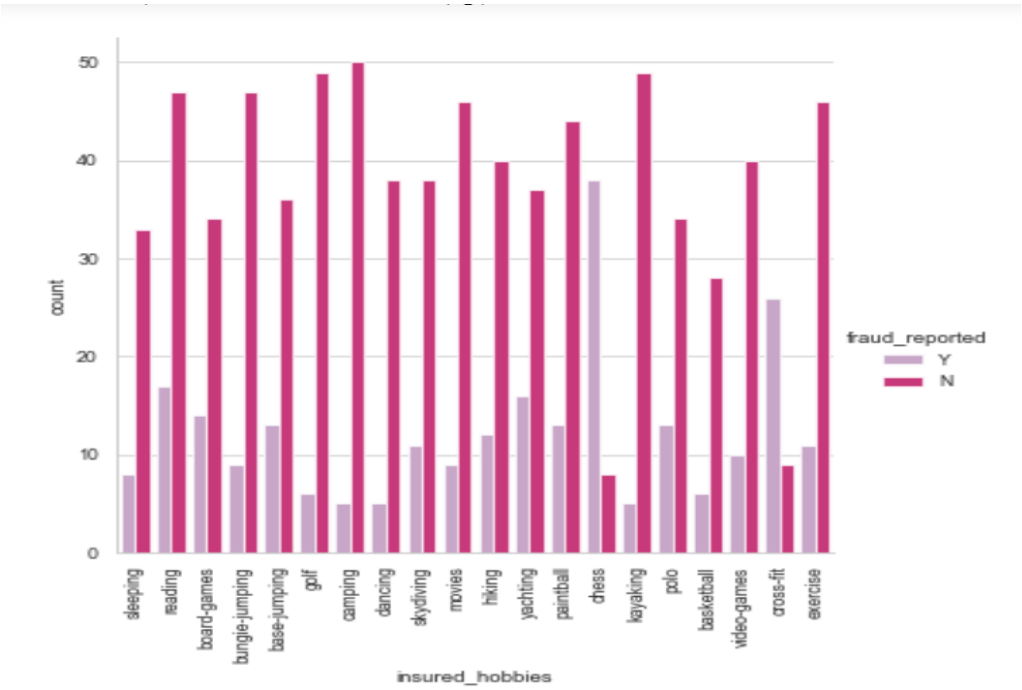
The fraudulent level is very less for the people who have high school education and the people who have completed their "JD" education have high fruad report.The people who have high insured education are facing insurance fraudulent compared to the people with less insured education level.

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



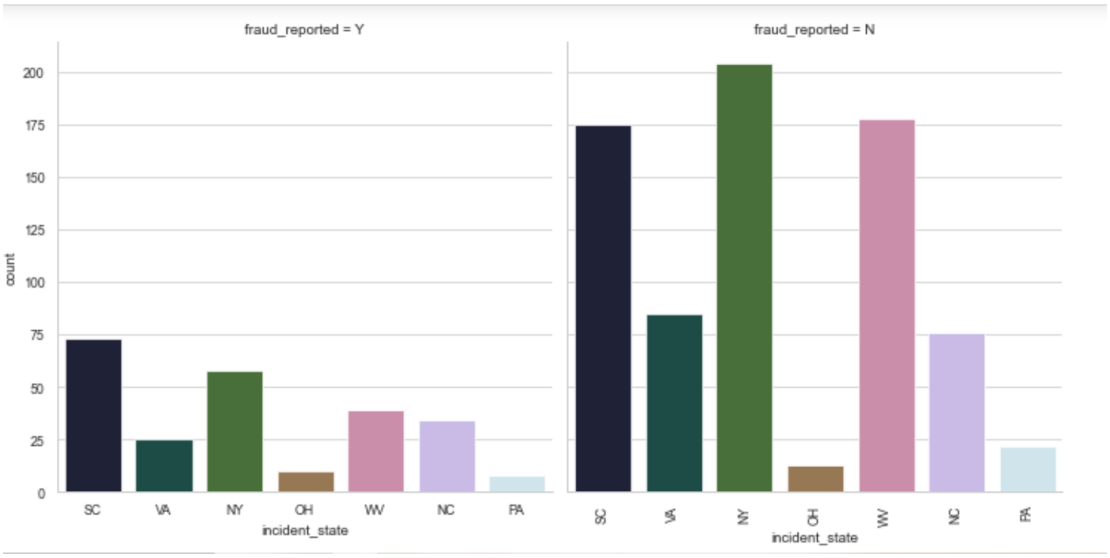
The people who are in the position exec-managerial have high fraud reports compared to others.

```
#comparing the insured_hobbies and fraude_report
sns.factorplot('insured_hobbies',kind='count',data=df,hue='fraud_reported',palette='PuRd')
plt.xticks(rotation=90)
```



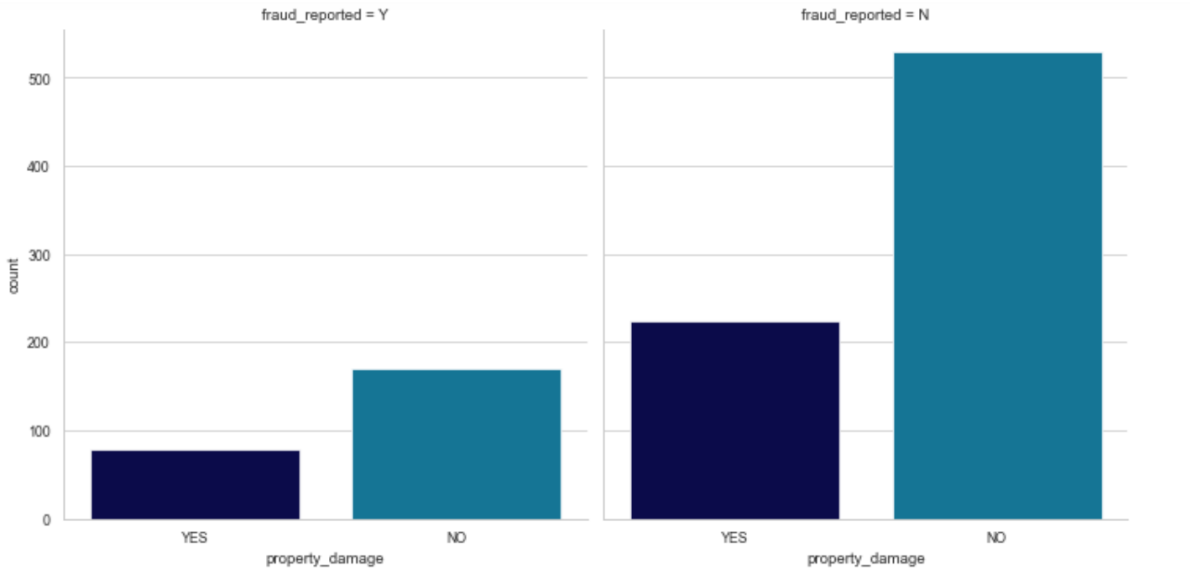
The fraud report is high for the people who have the hobby of playing chess and cross fit.

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



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

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

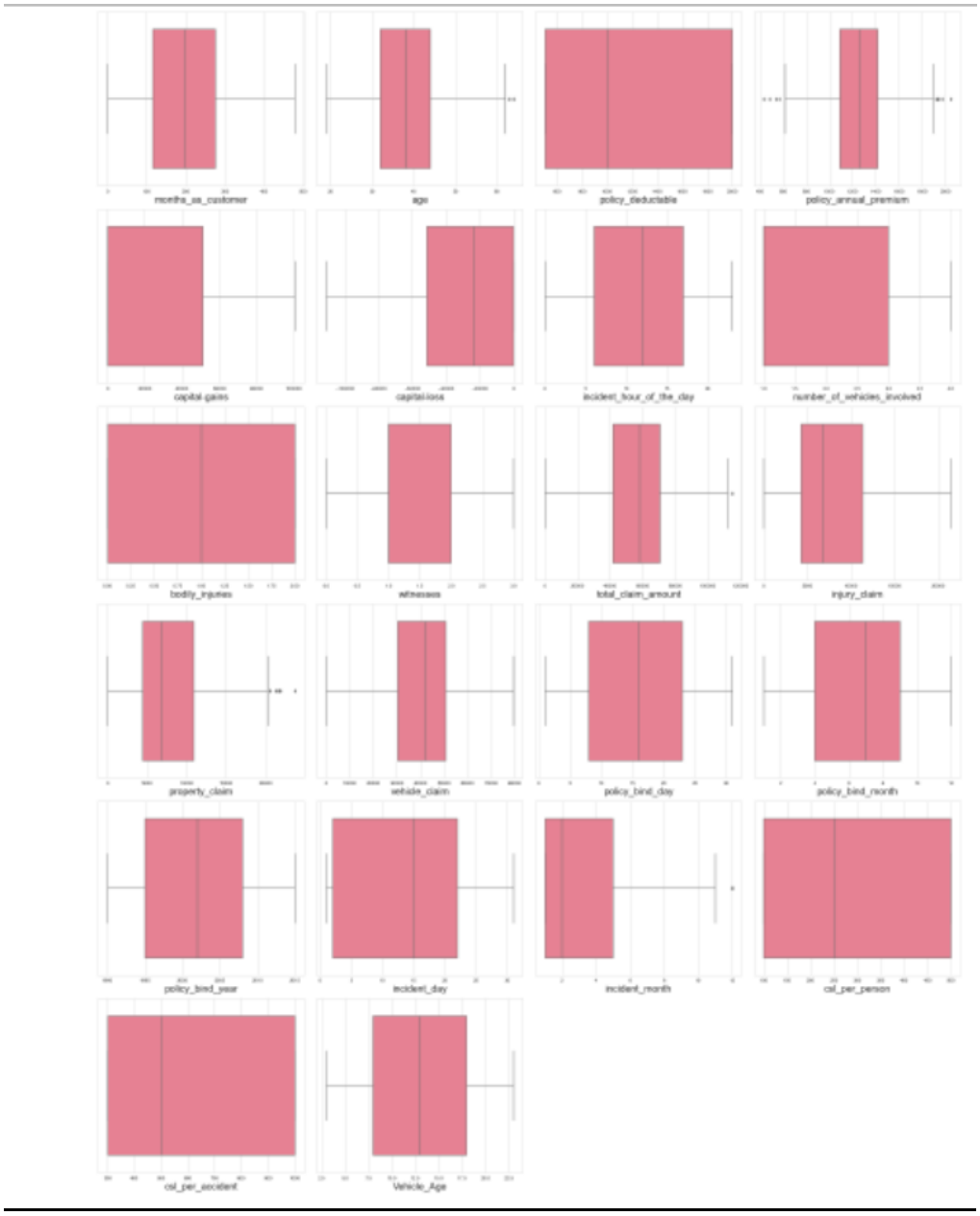


The customers who do not have any property damage case they have high fraud reports.

Now we have done with the visualization in order to analyze and understand the data. So in this EDA part, we have looked into various aspect of the dataset, like looking for the null values and imputing, extracting date time, observing the value counts and 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

```
# Lets check the outliers by plotting boxplot
plt.figure(figsize=(25,35),facecolor="white")
plotnumber=1
for col in numerical_col:
    if plotnumber<=23:
        ax = plt.subplot(6,4,plotnumber)
        sns.boxplot(df[col],palette="husl")
        plt.xlabel(col,fontsize=20)
    plotnumber+=1
plt.tight_layout()
```



As we can see, I have used 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 which contains outliers hence removing the outliers in these columns using Zscore method.

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

z=np.abs(zscore(features))

z
```

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

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

auto_model	fraud_reported	policy_bind_day	policy_bind_month	policy_bind_year	incident_day	incident_month	csl_per_person	csl_per_accident	Vehicle_Age
92x	Y	17	10	2014	25	1	250	500	14
E400	Y	27	6	2006	21	1	250	500	11
RAM	N	9	6	2000	22	2	100	300	11
Tahoe	Y	25	5	1990	1	10	250	500	4
RSX	N	6	6	2014	17	2	500	1000	9
...	...	...	...	...	...	...	...	...	...
Accord	N	16	7	1991	22	2	500	1000	12
Passat	N	1	5	2014	24	1	100	300	3
Impreza	N	17	2	2003	23	1	250	500	22
A5	N	18	11	2011	26	2	500	1000	20
E400	N	11	11	1996	26	2	250	500	11

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"]

from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method='yeo-johnson')
```

Now we have removed the skewness and the data.

Now we have completed our analysis of the dataset and also cleaned the dataset so that we can build a good model....

However, we are not yet ready. We have seen above that there are some problems that still exist in the dataset. We have seen that the dataset has both numerical and categorical data. The model only understand numerical data, hence we will encode the data. Also we have seen that there can be some multi-collinearity, that we will see through a heatmap and also further remove it. Again we have also seen that the target variable is imbalance, hence will fix it by oversampling. And finally we will scale the data so that it is ready to be trained and tested.
Let's begin.

Encoding the data

```
from sklearn.preprocessing import LabelEncoder
lr=LabelEncoder()
new_df[categorical_col]= new_df[categorical_col].apply(lr.fit_transform)
```

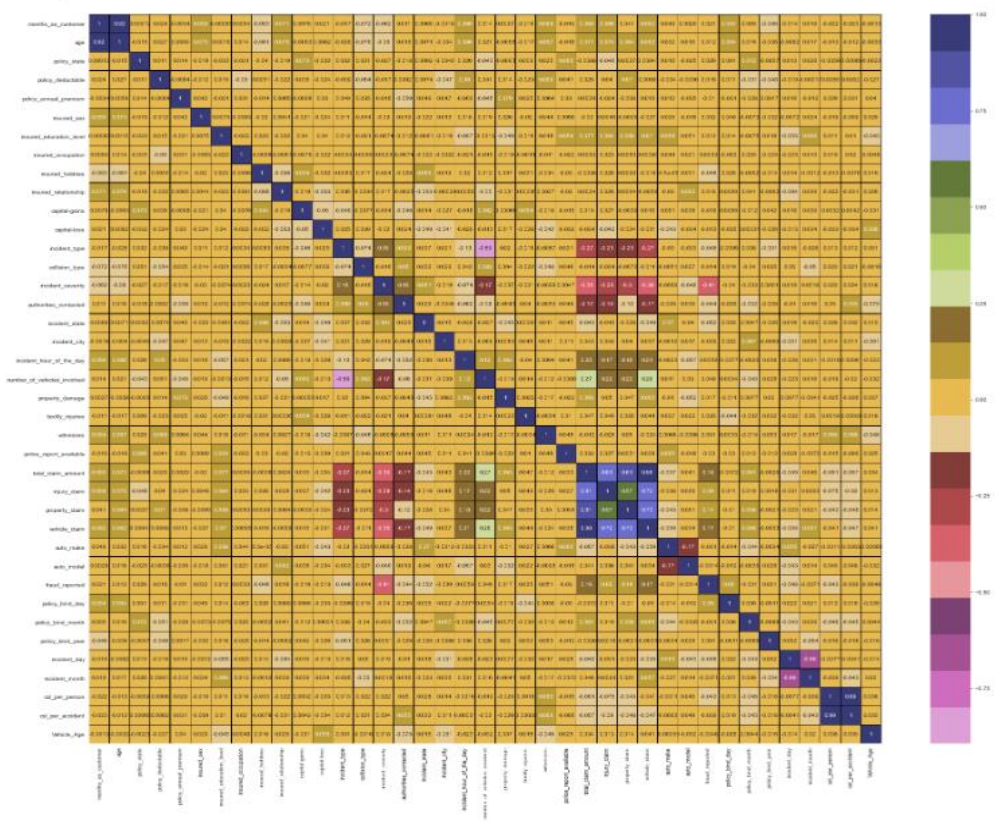
```
new_df[categorical_col].head()
```

ıorities_contacted	incident_state	incident_city	property_damage	police_report_available	auto_make	auto_model	fraud_reported
4	4	1	1	1	10	1	1
4	5	5	0	0	8	12	1
4	1	1	0	0	4	30	0
4	2	0	0	0	3	34	1
2	1	0	0	0	0	31	0

Now we have encoded the dataset using 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.

```
#Heatmap for correlation
plt.figure(figsize=(30,25))
sns.heatmap(new_df.corr(),linecolor='black',linewidths=0.2,annot=True,cmap='tab20b_r')
```



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 the most of the columns are highly correlated with each other which lead to the multicollinearity problem. We will check the VIF value to overcome with 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

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x=pd.DataFrame(ss.fit_transform(x),columns=x.columns)
x.head()
```

Checking Multi-colinearity using VIF

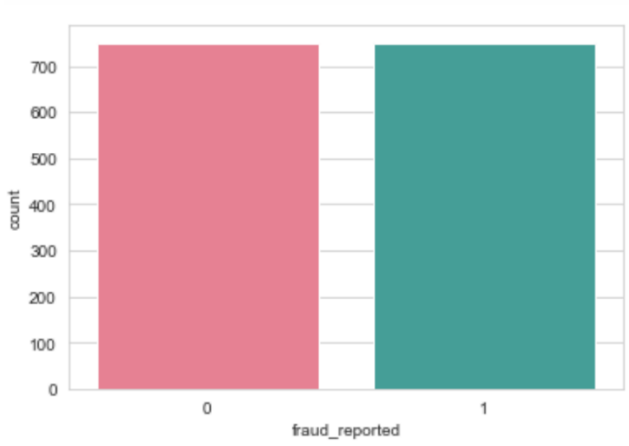
```
#Finding the variance inflation factor

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

It is observed that some columns have VIF above 10 that mean they are causing multicollinearity problem. Let's drop the feature having high VIF value amongst all the columns.
I have dropped the total_claim_amount and csl_per_accident features with colinearity more than 10, and now we have removed the problem.

We had earlier identified another problem of imbalance data in the target variable, lets treat it.

```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
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.
At 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 use 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.)

```
maxAcc=0
maxRs=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=i)
    rf=RandomForestClassifier()
    rf.fit(x_train,y_train)
    pred=rf.predict(x_test)
    acc=accuracy_score(y_test,pred)
    if acc>maxAcc:
        maxAcc=acc
        maxRs=i
print("Best accuracy is ",maxAcc,"at random state",maxRs)
```

```
Best accuracy is  0.92 at random state 177
```

Here we have used the RandomForestClassifier to find the best random state, as we have got an accuracy score of 92% (pretty good), at the random state of 177. 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.

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30,random_state=maxRs)
```

Model Building

RandomForestClassifier

```
# Checking accuracy for RandomForestClassifier
rf= RandomForestClassifier()
rf.fit(x_train,y_train)
pred = rf.predict(x_test)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.9022222222222223
[[210  17]
 [ 27 196]]
              precision    recall  f1-score   support

           0       0.89      0.93      0.91       227
           1       0.92      0.88      0.90       223

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

The first model was built using RandomForestClassifier, which gave an accuracy score of 90%, however we are hungry data scientist and will not be satisfied with only one model. We will try various models and see what accuracy score we get.

Support Vector Machine Classifier

```
# Checking accuracy for SVC
svc= SVC()
svc.fit(x_train,y_train)
pred = svc.predict(x_test)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8844444444444445
[[201  26]
 [ 26 197]]
              precision    recall  f1-score   support

           0       0.89      0.89      0.89       227
           1       0.88      0.88      0.88       223

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

With SupportVectorClassifier we got an accuracy score of 88%.

GradientBoostingClassifier ¶

```
# Checking accuracy for GradientBoosting classifier
gb= GradientBoostingClassifier()
gb.fit(x_train,y_train)
pred = gb.predict(x_test)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.9177777777777778
[[212  15]
 [ 22 201]]
              precision    recall  f1-score   support

           0       0.91      0.93      0.92       227
           1       0.93      0.90      0.92       223

    accuracy                           0.92       450
   macro avg       0.92      0.92      0.92       450
weighted avg       0.92      0.92      0.92       450
```

With GradientBostingClassifier we got an accuracy score of 91%.

DecisionTreeClassifier

```
# Checking accuracy for DecisionTree classifier
dc= DecisionTreeClassifier()
dc.fit(x_train,y_train)
pred = dc.predict(x_test)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.8466666666666667
[[192  35]
 [ 34 189]]
              precision    recall  f1-score   support

           0       0.85      0.85      0.85       227
           1       0.84      0.85      0.85       223

    accuracy                           0.85       450
   macro avg       0.85      0.85      0.85       450
weighted avg       0.85      0.85      0.85       450
```

With DecisionTreeClassifier we got an accuracy score of 84%.

LogisticRegression

```
# Checking accuracy for LogisticRegression
lr= LogisticRegression()
lr.fit(x_train,y_train)
pred =lr.predict(x_test)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.7622222222222222
[[169  58]
 [ 49 174]]
              precision    recall  f1-score   support

           0       0.78      0.74      0.76       227
           1       0.75      0.78      0.76       223

    accuracy                           0.76       450
   macro avg       0.76      0.76      0.76       450
weighted avg       0.76      0.76      0.76       450
```

With LogisticRegression we got an accuracy score of 76%.

ExtraTreesClassifier

```
# Checking accuracy for Extratree classifier
et= ExtraTreesClassifier()
et.fit(x_train,y_train)
pred = et.predict(x_test)
print(accuracy_score(y_test,pred))
print(confusion_matrix(y_test,pred))
print(classification_report(y_test,pred))
```

```
0.9288888888888889
[[211  16]
 [ 16 207]]
              precision    recall  f1-score   support

           0       0.93      0.93      0.93       227
           1       0.93      0.93      0.93       223

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

With this model, ExtraTreesClassifier we have got accuracy score of 92%, which is better than RandomForestClassifier.
Before we can announce the best model, we always have to make sure that the model is not over fitting; hence we will perform cross validation of all the models built.

```
# Checking cross validation score for LogisticRegression
from sklearn.model_selection import cross_val_score
print('cross validation score',cross_val_score(lr,x,y,cv=5).mean())
```
```
cross validation score 0.7266666666666668
```
```
# Checking cross validation score for RandomForest
print('cross validation score',cross_val_score(rf,x,y,cv=5).mean())
```
```
cross validation score 0.8800000000000001
```
```
# Checking cross validation score for DecisionTree
print('cross validation score',cross_val_score(dc,x,y,cv=5).mean())
```
```
cross validation score 0.8306666666666667
```
```
# Checking cross validation score for GradientBoost
print('cross validation score',cross_val_score(gb,x,y,cv=5).mean())
```
```
cross validation score 0.8746666666666666
```
```
# Checking cross validation score for svc
print('cross validation score',cross_val_score(svc,x,y,cv=5).mean())
```
```
cross validation score 0.8786666666666667
```
```
# Checking cross validation score for ExtraTreeclassifier
print('cross validation score',cross_val_score(et,x,y,cv=5).mean())
```
```
cross validation score 0.9146666666666666
```

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

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

```
# Extra Trees Classifier
from sklearn.model_selection import GridSearchCV

parameters = {'criterion':['gini','entrophy'],
              'max_features':['auto','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)
```

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

```
gcv.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 20,
 'max_features': 'log2',
 'n_estimators': 200,
 'n_jobs': -1}
```

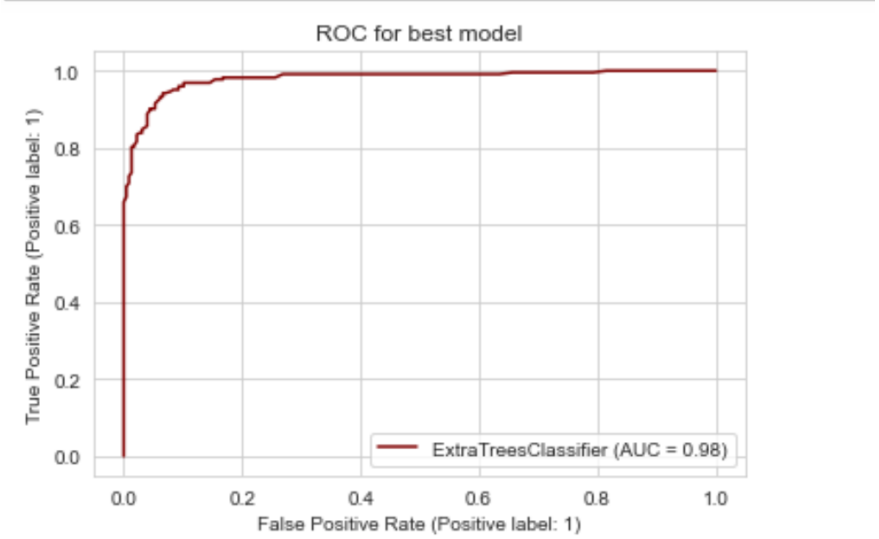
Here we have got the best parameters, and we will build our final model using these parameters.

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

```
93.33333333333333
```

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

Plotting and AUCROC curve for the final model.



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

Saving the model

```
#saving the model
joblib.dump(Final_model,'Insurance_claims_Fraud_Detection.pkl')
```

```
['Insurance_claims_Fraud_Detection.pkl']
```

Predicting the model

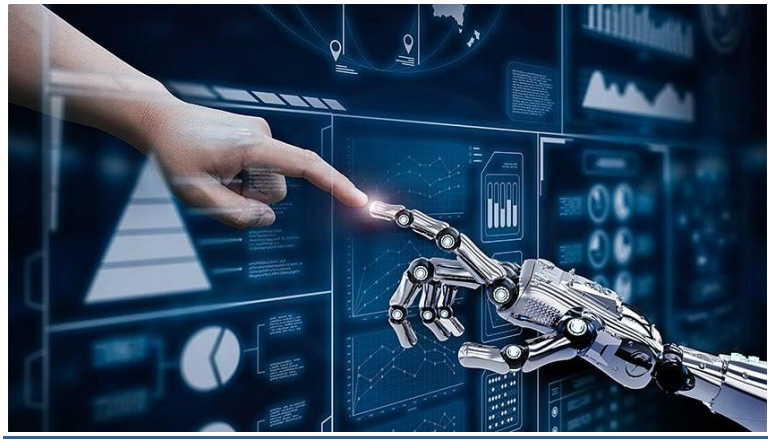
```
# Lets load the saved model and get the predictions
model = joblib.load("Insurance_claims_Fraud_Detection.pkl")

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

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

```
a = np.array(y_test)
df = pd.DataFrame()
df['Predicted'] = prediction
df['Original'] = a
df
```

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

Concluding Remarks



In the beginning of the blog we have discussed about the lifecycle of a Machine Learning Model, you can see how we have touched based 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 on 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 hyper parameter tuning we can improve our model accuracy, for instance in this model the accuracy remained same.

Using this machine Learning Model we people can easily predict the insurance claim is fraudulent or not and we could reject those application which will be considered as fraud claims.


THANK YOU