INSURANCE CLAIMS – FRAUD DETECTION USING MACHINE LEARNING

BY: AKASH KUMAR

Insurance Claim Fraud Detection Project

In association with Data Trained Academy: Batch: 1843

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 have been used like, insured information, insured persons, personal details and the incident information. In total the dataset has 40 features. So, using all these previously acquired information and analysis done with the data I have achieved a good model that has 92% 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 Requirement & Tools Used

Hardware Requirement:

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

Software Requirement:

- Jupyter Notebook

Libraries Used:

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

Heading forward we will try to understand the problem statement and the dataset.

Problem Definition:

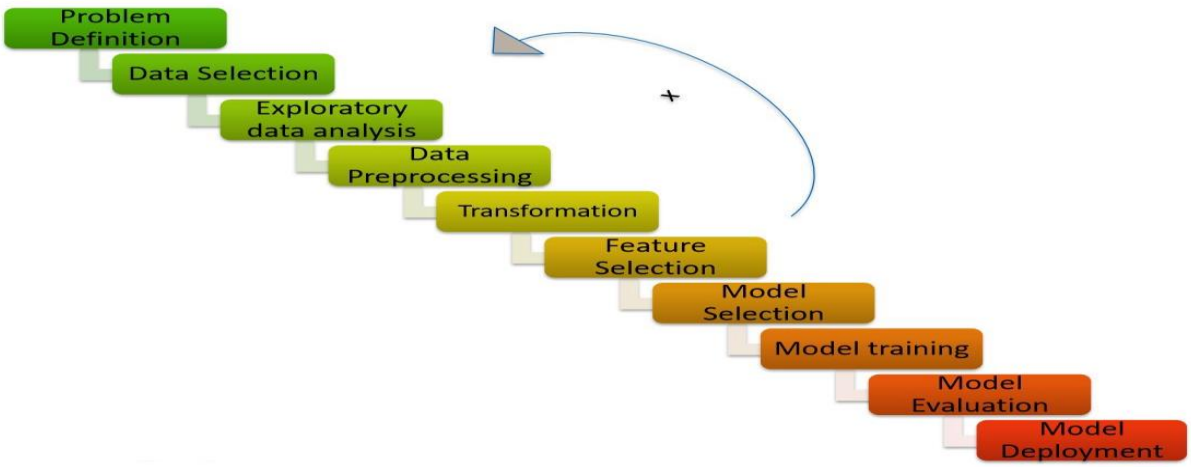
Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem. In this project; we are provided a dataset which has the details of the insurance policy along with the customer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

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import zscore
from sklearn.preprocessing import PowerTransformer
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,ExtraTreesClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import GradientBoostingClassifier, AdaBoostClassifier, BaggingClassifier
from xgboost import XGBClassifier as xgb
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, accuracy_score
from sklearn.model_selection import cross_val_score
from sklearn import datasets
from sklearn import metrics
from sklearn import model_selection
from sklearn.metrics import plot_roc_curve
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

Importing the Dataset

Let’s import the dataset first.

```
df=pd.read_csv('/Users/prita/Downloads/DATASETS/Capstone Project/Insurance Claims- Fraud Detection/Automobile_insurance_fra
```

```
df.head()
```

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 let’s 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.

```
# 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):
```

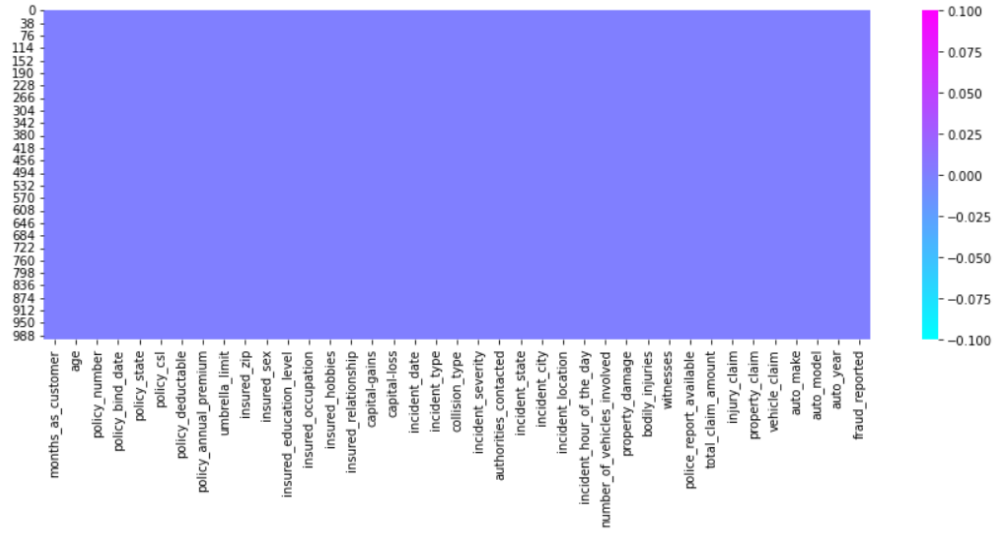
```
# Checking the type of dataset
df.dtypes
```

```
# 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
plt.figure(figsize=(15,5))

sns.heatmap(df.isnull(), cmap="cool")
plt.show()
```



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 _c39 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 policy_number and incident_location 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 have more than 70% of zero values, let's drop that column.

```
# Dropping umbrella_limit column
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 insured_zip column as it is not important for the prediction
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.

```
# 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'])
```

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_Day"] = df['policy_bind_date'].dt.day
df["policy_bind_Month"] = df['policy_bind_date'].dt.month
df["policy_bind_Year"] = df['policy_bind_date'].dt.year

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

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

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

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]
```

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

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

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

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

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

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

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

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

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

```
# 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")
```

Now after all the data cleaning 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

5 rows × 39 columns

Preparing for Visualization

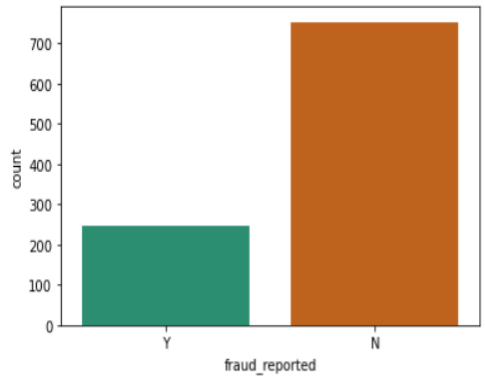
First, we will look into the categorical and numerical columns so that we can visualize the features accordingly.

```
# 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

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

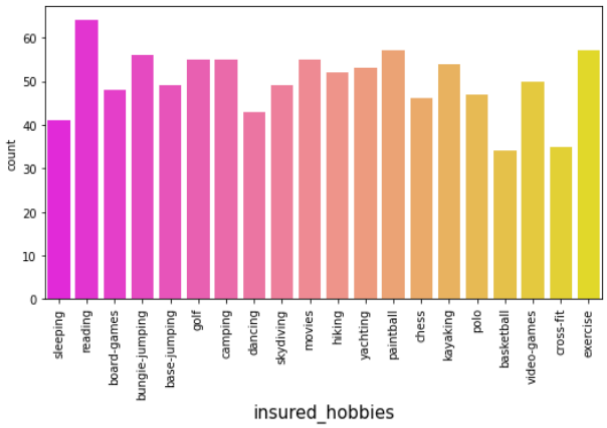
```
N    753
Y    247
Name: fraud_reported, dtype: int64
```

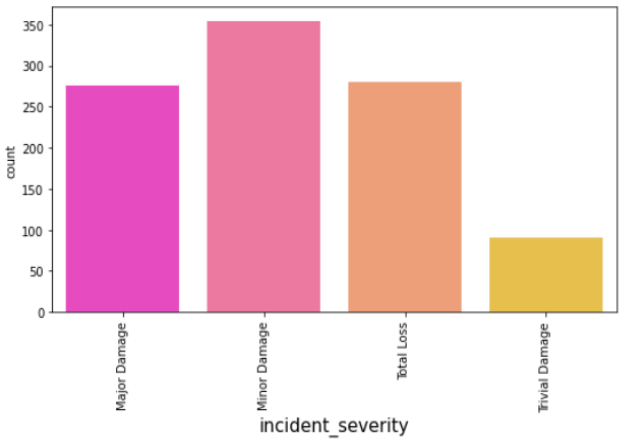


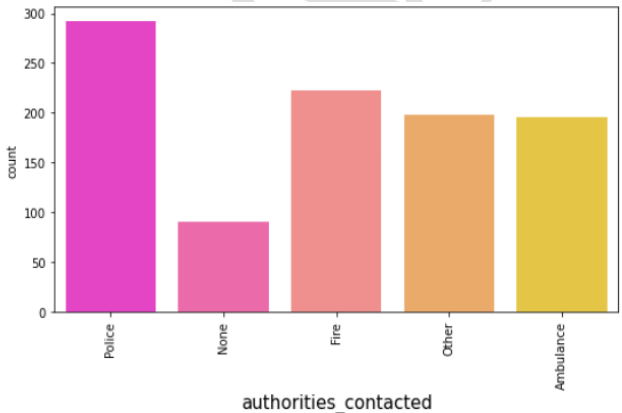
By looking into the plot, we can observe that the count of "N" is high compared to "N". Which means here we can assume that "Y" stands for "Yes" that is the insurance is fraudulent and "N" stands for "No" means the insurance claim is not fraudulent. Here most of the insurance claims have not reported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

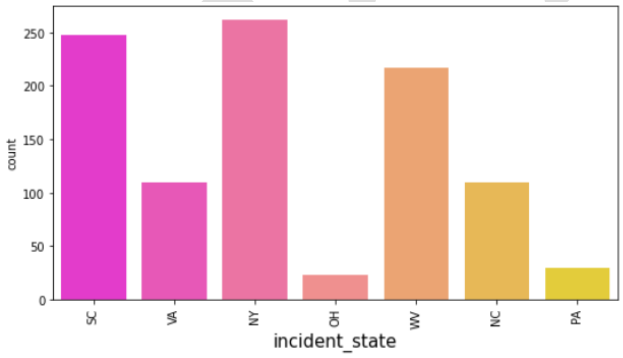
```
cols2 = ['insured_occupation', 'insured_hobbies', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident

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="spring")
        plt.xticks(rotation=90)
        plt.xlabel(column,fontsize=15)
    plotnumber+=1
plt.tight_layout()
plt.show()
```

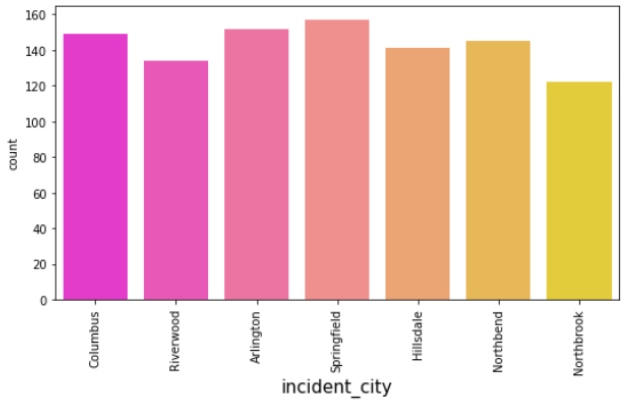

count
craft-repair
machine-op-inspct
sales
armed-forces
tech-support
prof-specialty
other-service
priv-house-serv
exec-managerial
protective-serv
transport-moving
handlers-cleaners
adm-clerical
farming-fishing
insured_occupation

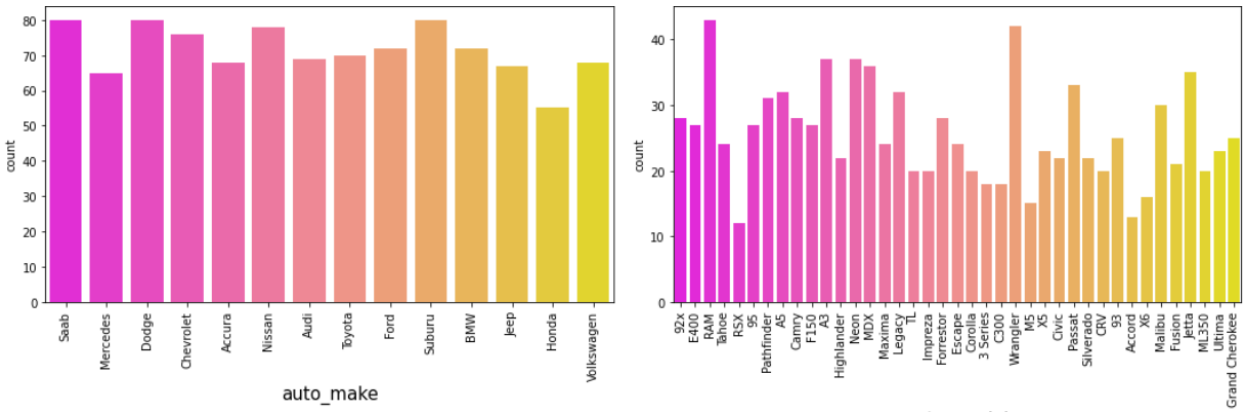

count
sleeping
reading
board-games
bungie-jumping
base-jumping
golf
camping
dancing
skydiving
movies
hiking
yachting
paintball
chess
kayaking
polo
basketball
video-games
cross-fit
exercise
insured_hobbies


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

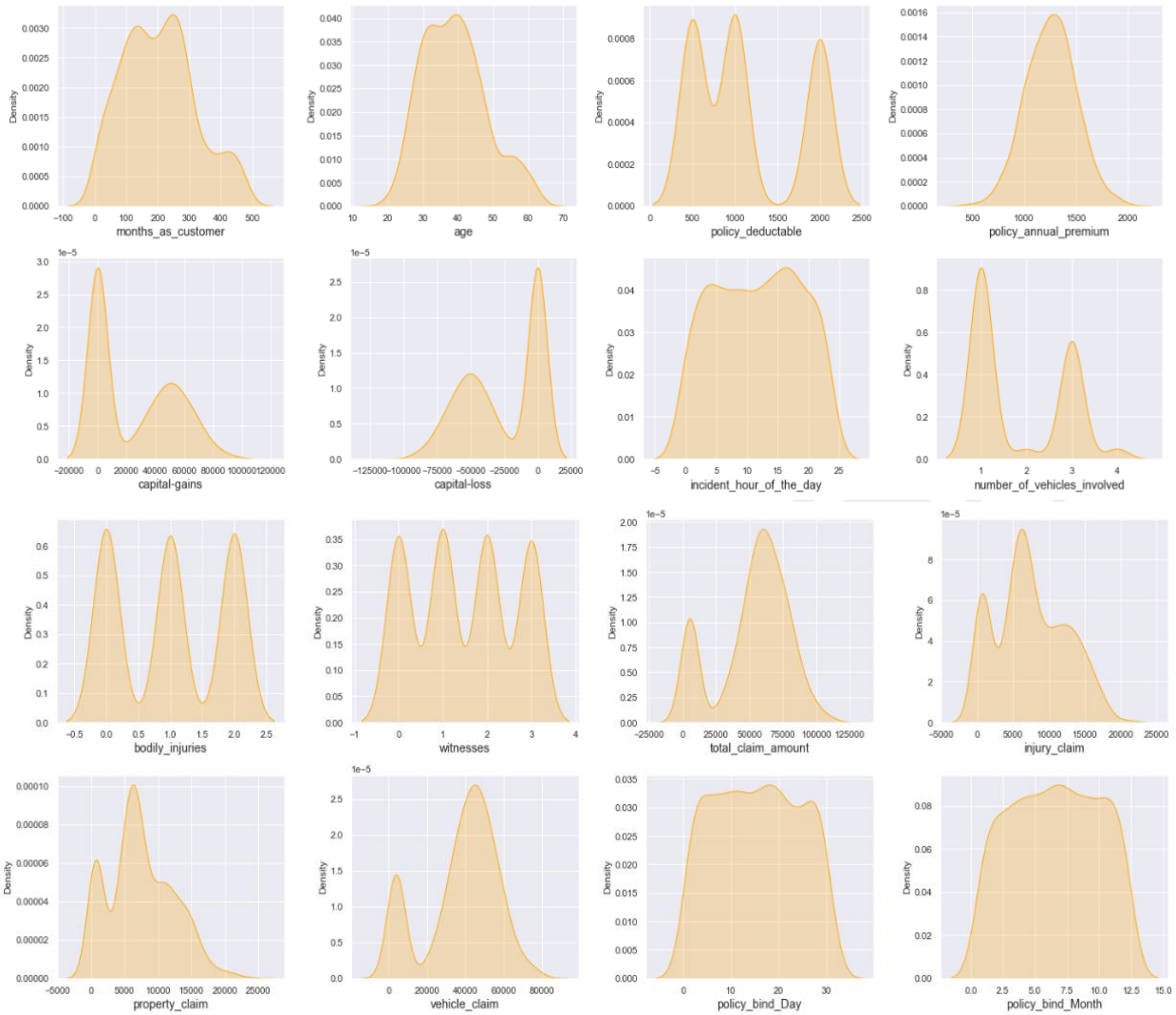

count
Columbus
Riverwood
Arlington
Springfield
Hillsdale
Northbend
Northbrook
incident_city



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

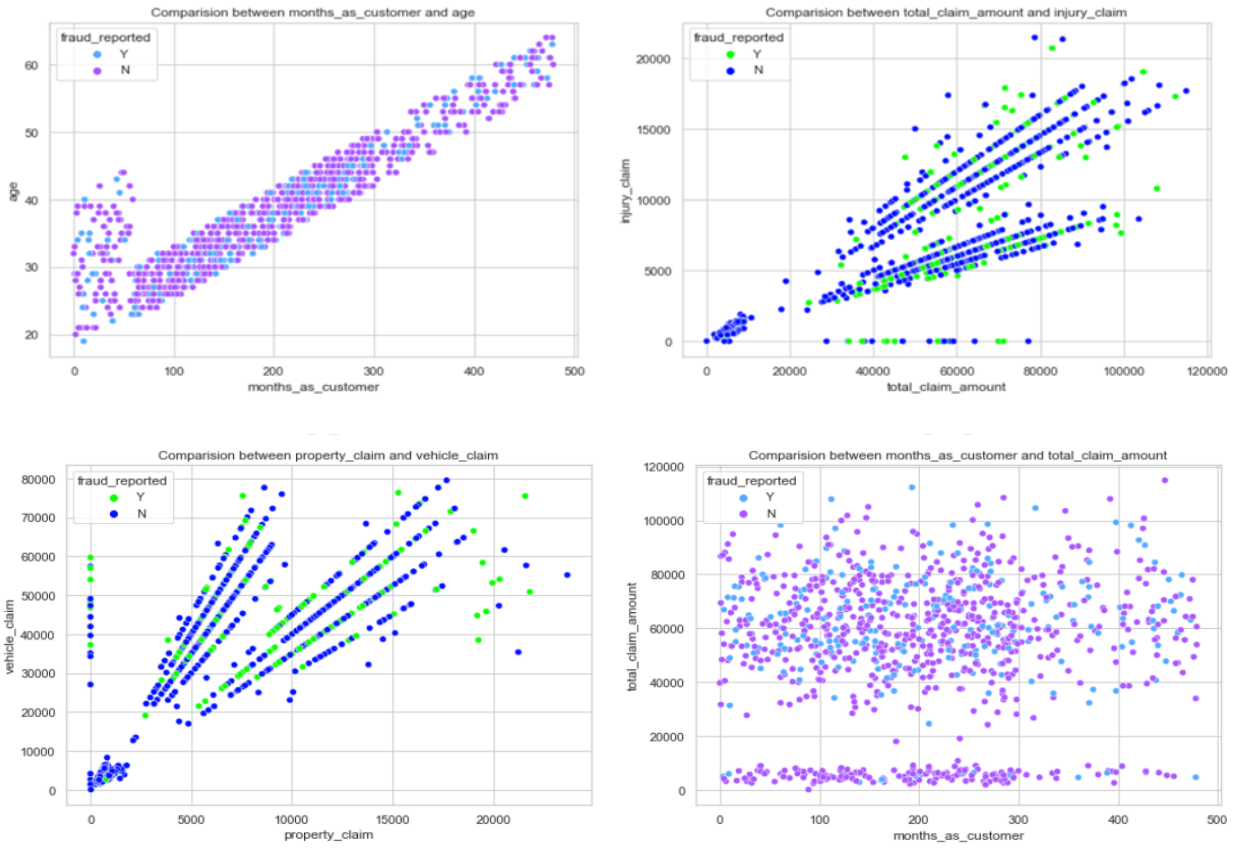
- In the insured occupation we can observe most of the data is covered by machine operation inspector followed by professional speciality.
- With respect to insured hobbies, we can notice reading covered the highest data followed by exercise. And other categories have the average counts.
- The incident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

```
# Checking how the data has been distriubted in each column
sns.set(style="darkgrid")

plt.figure(figsize=(20,25),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=23:
        ax=plt.subplot(6,4,plotnumber)
        sns.distplot(df[column],color="orange",hist=False,kde_kws={"shade": True})
        plt.xlabel(column,fontsize=14)
    plotnumber+=1
plt.tight_layout()
```



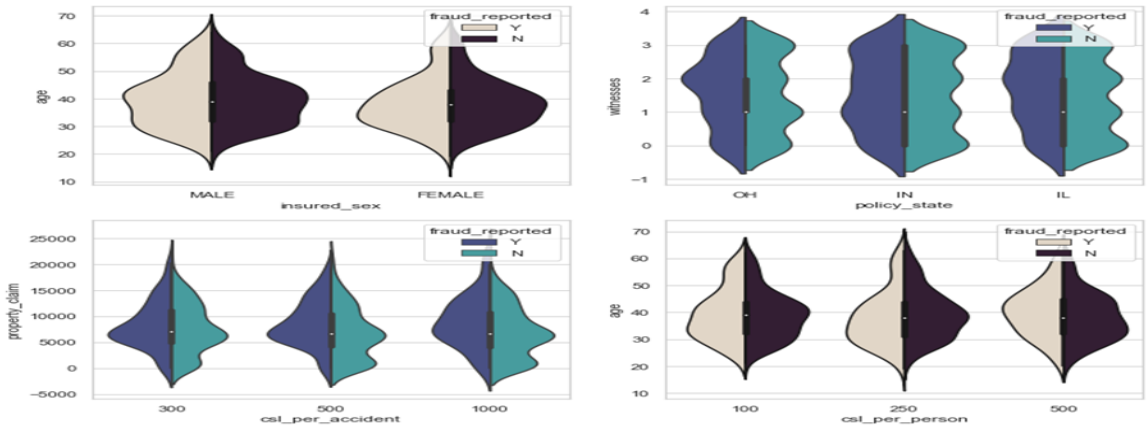
The data is normally distributed in most of the columns. Some of the columns like capital gains and incident months have mean value greater than the median, hence they are skewed to right. The data in the column capital loss is skewed to left since the median is greater than the mean. We will remove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

```python
# Comparision between two variables
sns.set(style="whitegrid")

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'],palette="cool");

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'],palette="hsv");

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'],palette="hsv");

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'],palette="cool");
```

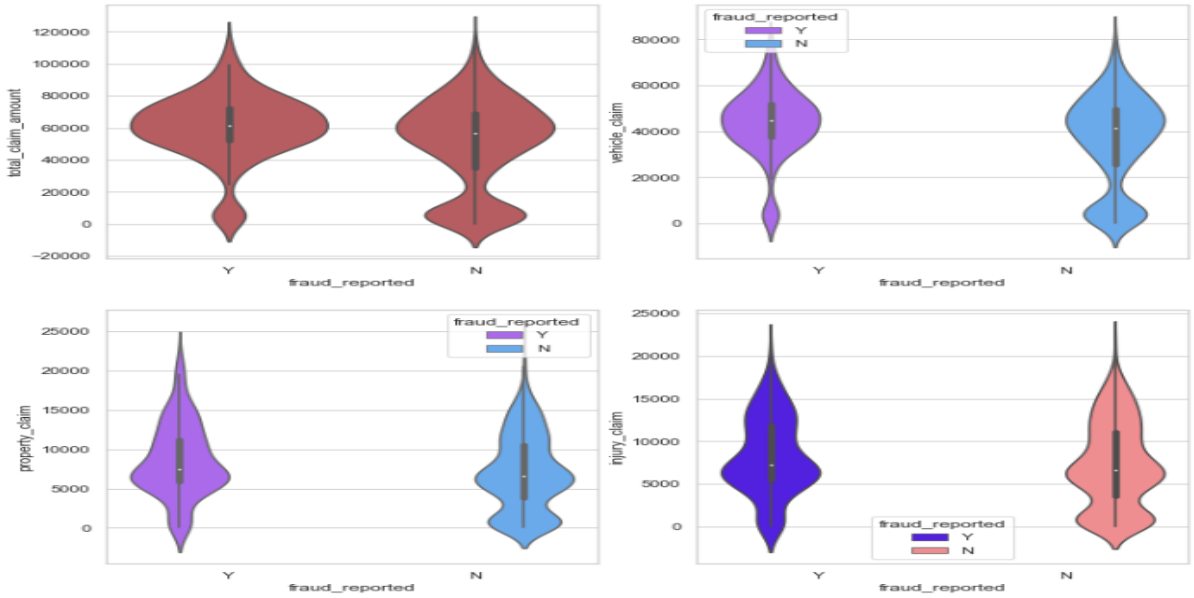


- There is a positive linear relation between age and month_as_customer column. As age increases the month as customers also increases, also the fraud reported is very less in this case.
- In the second graph we can observe the positive linear relation, as total claim amount increases, injury claim is 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="mako")

# 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="mako")

# 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="ch:.25")
plt.show()
```

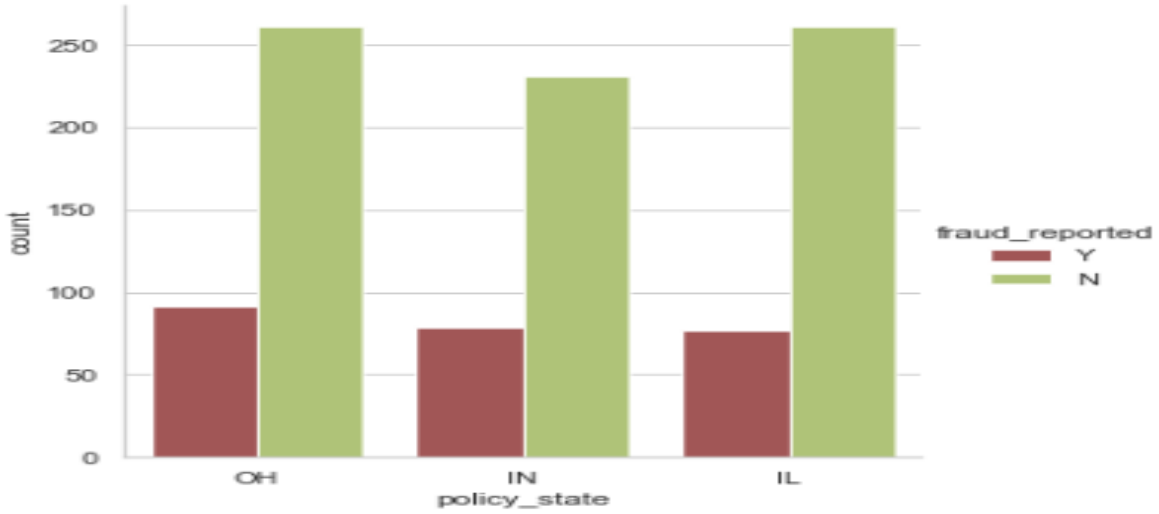



```
fig,axes=plt.subplots(2,2,figsize=(12,12))

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

# 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="cool_r")

# 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()
```

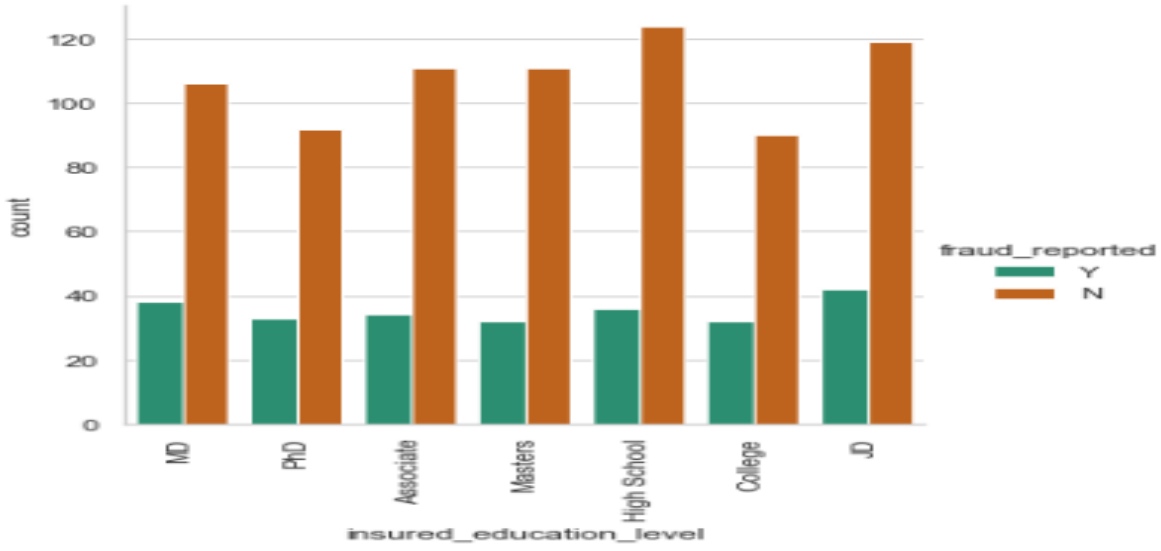


Visualization is a technique where by comparison and plotting the data becomes self-explanatory, which we have seen until now. Moving ahead with some more visualization plots before we can proceed to model building.

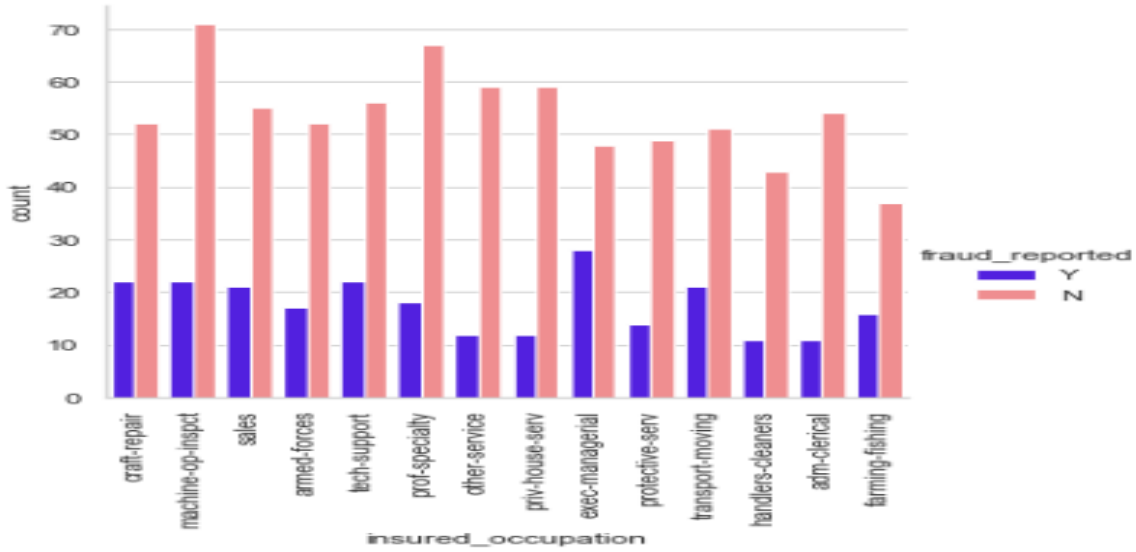
```
# Comparing policy_state and fraud_reported
sns.factorplot('policy_state',kind='count',data=df,hue='fraud_reported',palette="tab20b_r")
plt.show()
```



```
# Comparing insured_education_level and fraud_reported

sns.factorplot('insured_education_level',kind='count',data=df,hue='fraud_reported',palette="Dark2")
plt.xticks(rotation=90)
plt.show()
```

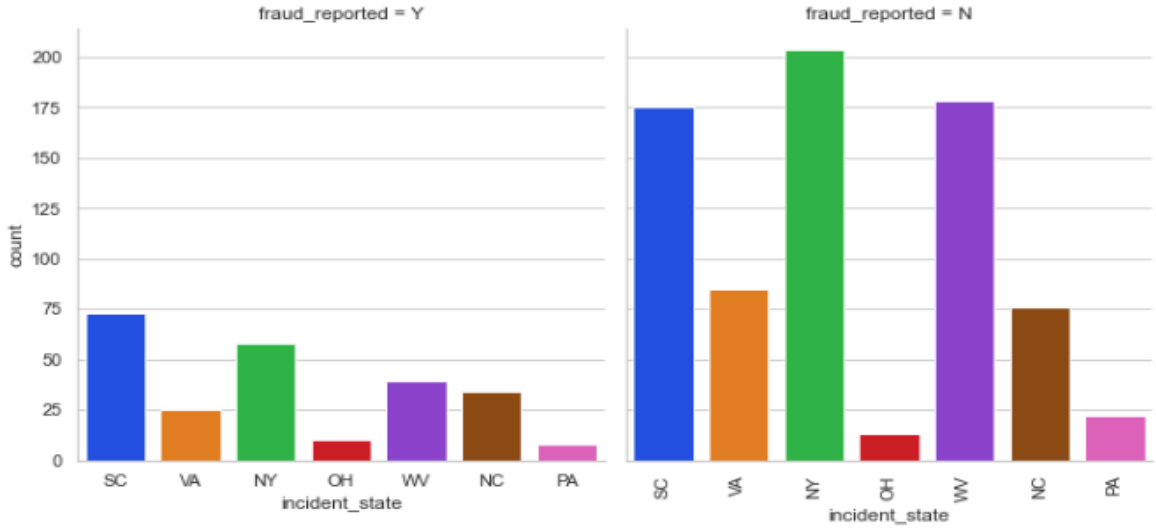



```
# Comparing insured_occupation and fraud_reported
sns.factorplot('insured_occupation',kind='count',data=df,hue='fraud_reported',palette="gnuplot2")
plt.xticks(rotation=90)
plt.show()
```

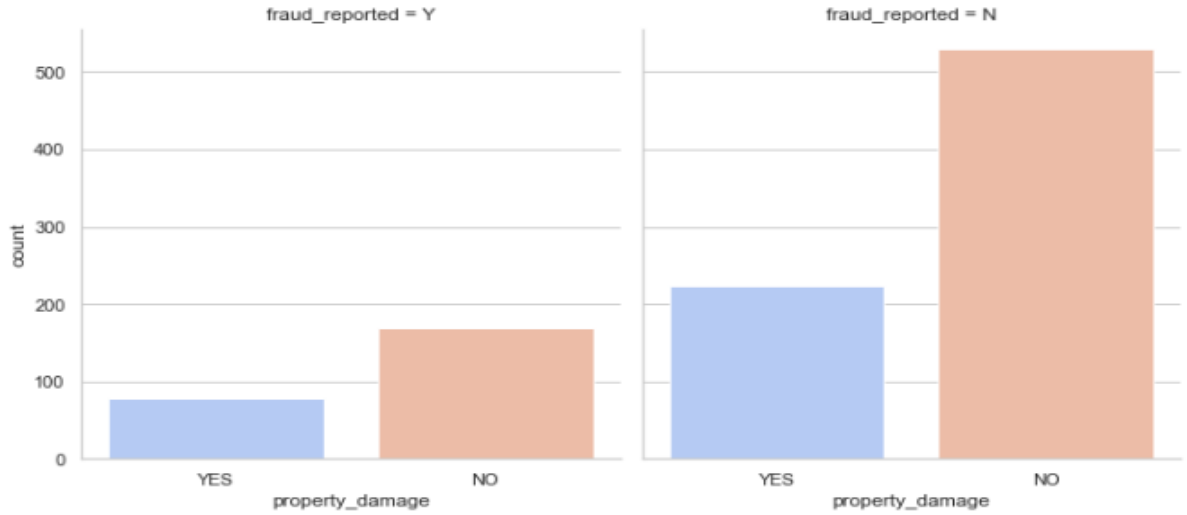


```
# Comparing insured_hobbies and fraud_reported
sns.factorplot('insured_hobbies',kind='count',data=df,hue='fraud_reported',palette="spring_r")
plt.xticks(rotation=90)
plt.show()
```


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



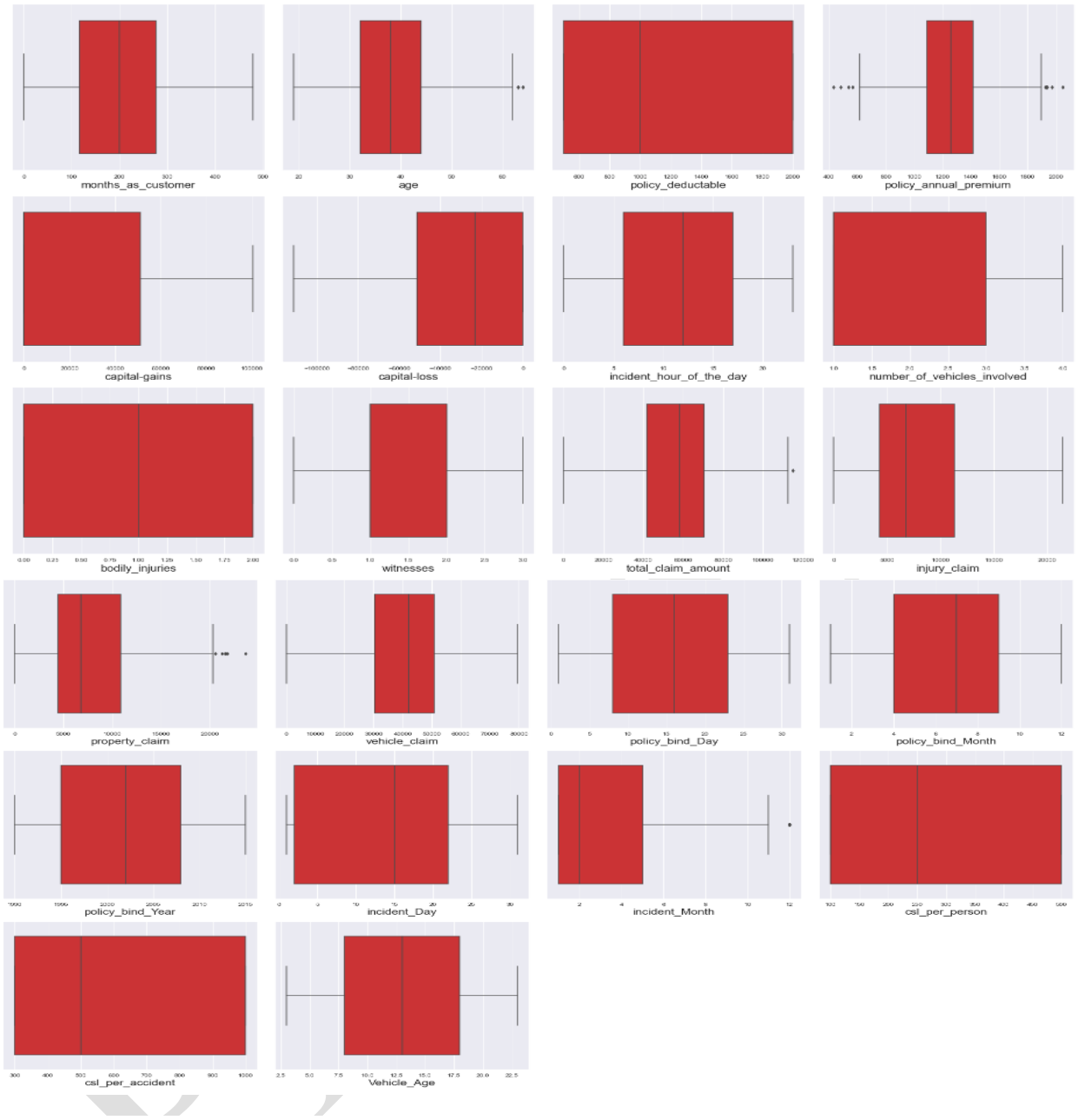
```
# Comparing property_damage and fraud_reported
sns.factorplot('property_damage',kind='count',data=df,col='fraud_reported',palette="coolwarm")
plt.show()
```



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

```
# Let's check the outliers by ploting box plot
sns.set(style="darkgrid")

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="Set1")
        plt.xlabel(column,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.

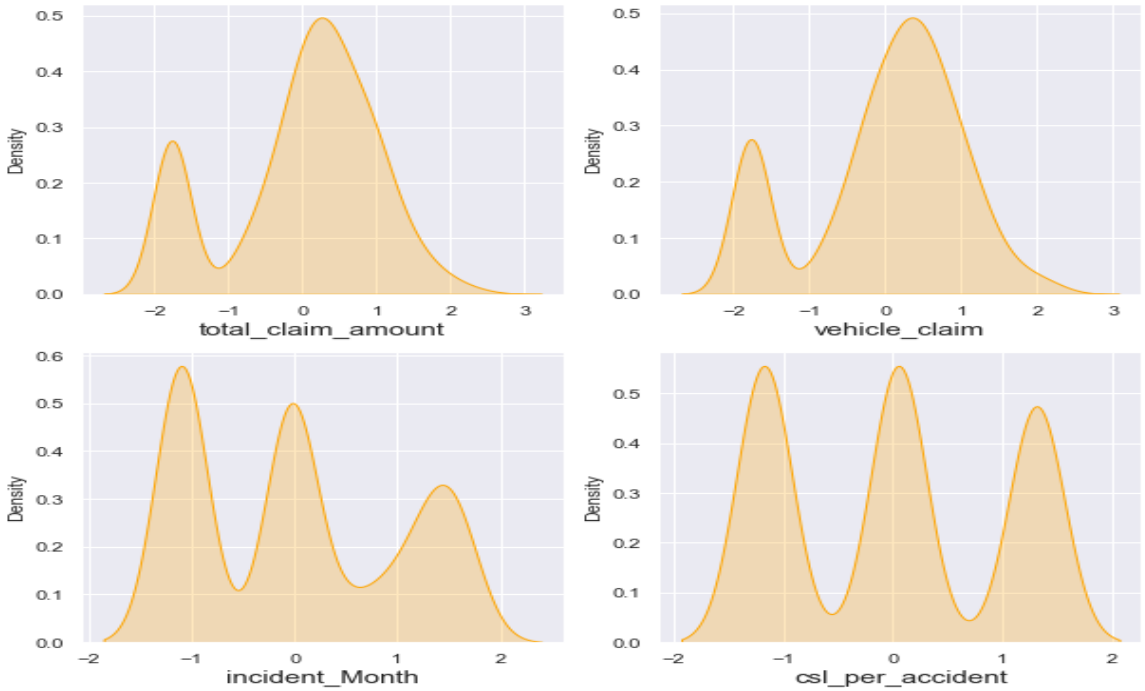
```
# Checking the skewness
new_df.skew().sort_values()
```

```
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 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 understands 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

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

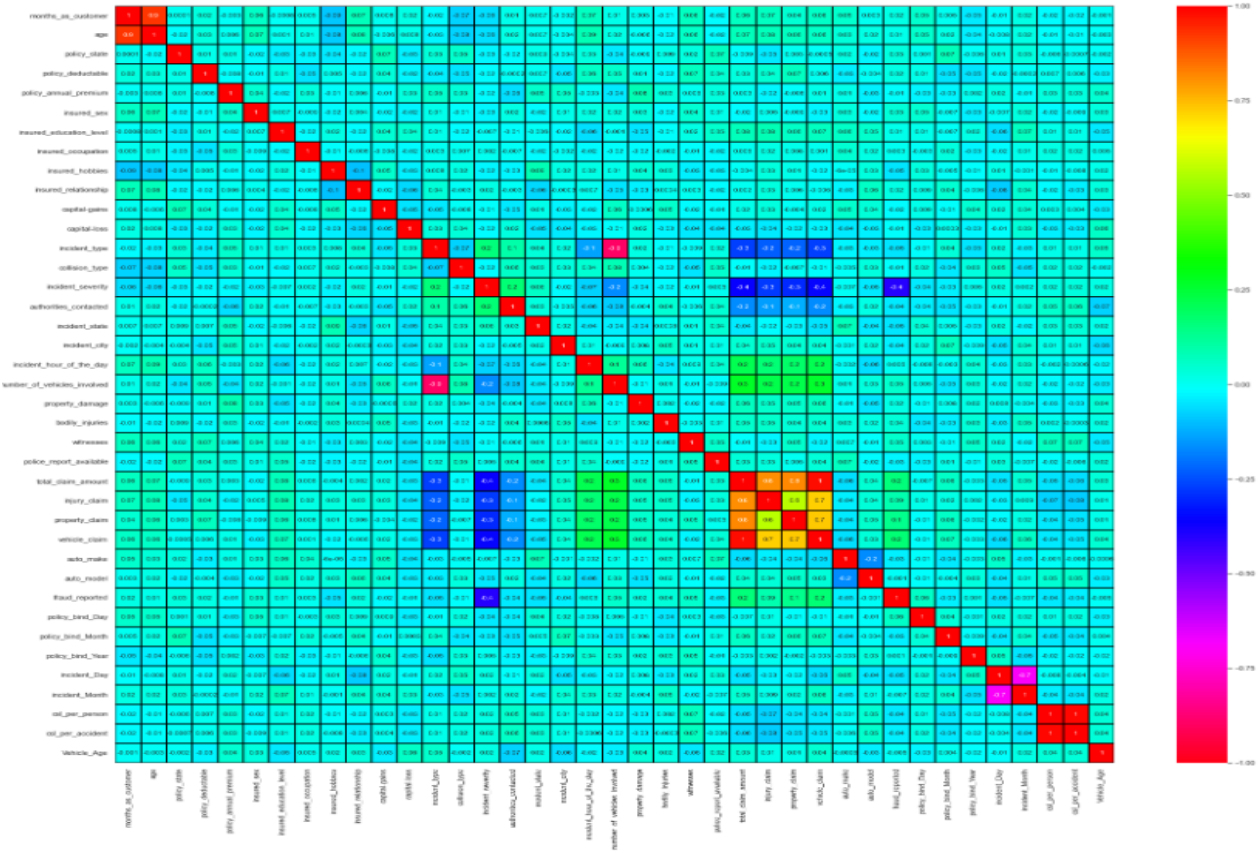
```
new_df[categorical_col].head()
```

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

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

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

Checking Multicollinearity 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 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 collinearity 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.

```
# Oversampling the data
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.)

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.30, random_state =i)
    rfc = RandomForestClassifier()
    rfc.fit(x_train, y_train)
    pred = rfc.predict(x_test)
    acc=accuracy_score(y_test, pred)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is ",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is  0.92  on Random_state  93
```

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 93. 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=.30,random_state=maxRS)
```

Model Building

Random Forest Classifier

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

# Prediction
predrfc = rfc.predict(x_test)

print(accuracy_score(y_test, predrfc))
print(confusion_matrix(y_test, predrfc))
print(classification_report(y_test,predrfc))
```

```
0.9044444444444445
```

The first model was built using RandomForestClassifier, which gave an accuracy score of 90.44%, 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 Support Vector Machine Classifier
svc = SVC()
svc.fit(x_train,y_train)

# Prediction
predsvc = svc.predict(x_test)

print(accuracy_score(y_test, predsvc))
print(confusion_matrix(y_test, predsvc))
print(classification_report(y_test,predsvc))
```

```
0.8822222222222222
```

With Support Vector Classifier we got an accuracy score of 88.22%.

Gradient Boosting Classifier

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

# Prediction
predgb = gb.predict(x_test)

print(accuracy_score(y_test, predgb))
print(confusion_matrix(y_test, predgb))
print(classification_report(y_test,predgb))
```

```
0.9022222222222223
```

With GradientBostingClassifier we got an accuracy score of 90.22%.

AdaBoost Classifier

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

# Prediction
predABC = ABC.predict(x_test)

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

```
0.8755555555555555
```

With AdaBoost Classifier, we got an accuracy score of 87.55%.

Bagging Classifier

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

# Prediction
predBC = BC.predict(x_test)

print(accuracy_score(y_test, predBC))
print(confusion_matrix(y_test, predBC))
print(classification_report(y_test,predBC))
```

```
0.8888888888888888
```

With Bagging Classifier, we got an accuracy score of 88.88%.

Extra Trees Classifier

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

# Prediction
predXT = XT.predict(x_test)

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

```
0.9222222222222223
```

With this model, Extra Trees Classifier we have got accuracy score of 92.22%, 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.

```
# 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: 0.8693333333333333
Support Vector Machine Classifier: 0.86
Gradient Boosting Classifier: 0.874
AdaBoosting Classifier: 0.8406666666666667
Bagging Classifier: 0.8673333333333334
Extra Trees Classifier: 0.9093333333333333
```

After the cross validation we can clearly see that Extra Trees, Classification is the best fit model.

Now, that we have found the best fit model, lets perform some Hyper Parameter Tuning 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_
```

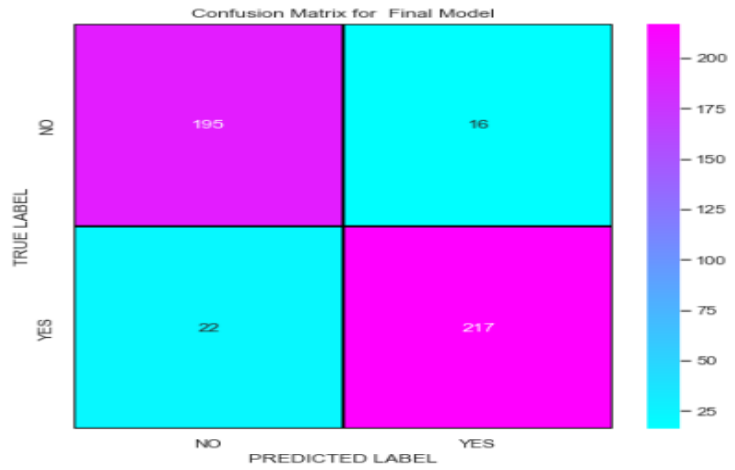
```
{'criterion': 'gini',
 'max_depth': 20,
 'max_features': 'sqrt',
 'n_estimators': 100,
 'n_jobs': -2}
```

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='sqrt', n_estimators=100, n_jobs=-2)
Insurance.fit(x_train, y_train)
pred = Insurance.predict(x_test)
acc=accuracy_score(y_test,pred)
print(acc*100)
```

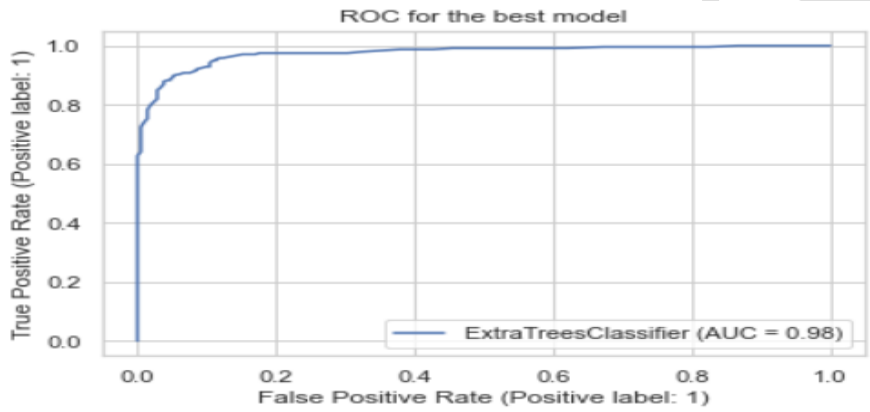
```
91.55555555555556
```

We have built our final model and we can see that the accuracy score has increased than 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 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
```

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

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 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.

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.