Blog on: Insurance Claim Fraud Detection

Author: Jayshree Singh

Batch: 1843

DataTrained

Introduction:

One of the biggest and most well-known issues facing insurers in the insurance sector is scam or fraud.

A purposeful deceit committed against, by, or on behalf of an insurance company or agent is known as insurance fraud. Applicants, policyholders, third-party claimants, or experts who offer services to claimants may all commit fraud at various stages of the transaction. Insurance fraud can also be committed by insurance brokers and corporate personnel. Inflating claims, lying on insurance applications, submitting claims for damage or injuries that never happened, and faking accidents are all examples of common scams.

Data science and machine learning, which have always been able to deliver accuracy or detect unfavorable situations, have been very helpful in many businesses. In this blog, I've developed a machine learning model to determine whether or not the assertion is false. Here, a variety of elements have been utilized, including incident data, insured person personal information, and insured individuals' personal information. The dataset comprises 40 distinct features in total. So I was able to create a solid model with a 92% accuracy rate using all of the previously gathered information and data analysis. So let's examine the procedures needed to achieve this accuracy.

The co-linearity and significance of the features have also been explained using a variety of visualization techniques.

Hardware & Software Requirements & Tools Used:

Hardware required:

- Processor: core i3 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:

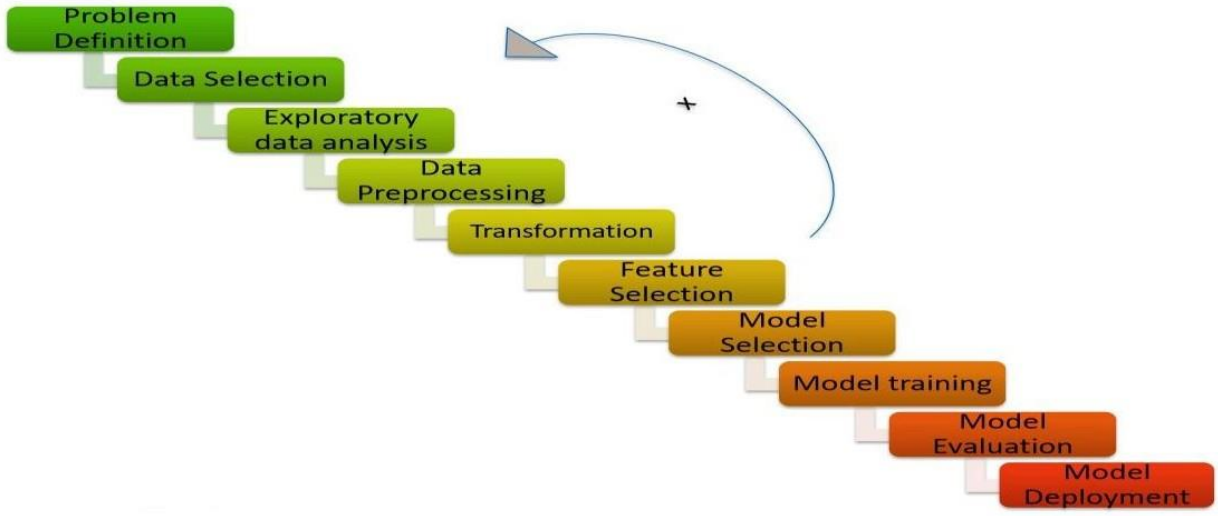
Insurance fraud is a big problem in this industry. It's difficult to identify fraud claims. Machine Learning algorithms are in a unique position to help the Auto Insurance industry with this huge problem. In this project, we are provided a dataset that has the details of the insurance policy along with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

In this example, we'll use data from the vehicle insurance industry to show how to build a predictive model that can determine whether a customer's insurance claim is fraudulent or not.

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.

Every machine learning algorithm must go through the machine learning life cycle in order to construct a machine learning model. Let's take a quick look at the model life cycle before we examine the actual



machine learning model and better understand its lifecycle.

Now that we are aware of the machine learning model lifetime, let's load the required libraries and go on.

Importing the necessary Libraries:

We have imported all necessary libraries as stated below in order to analyse the dataset.

- 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 data from the original 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	
...	...	...	...	...	...	...	...	...	...	...	...	

We could tell that the dataset comprises both categorical and numerical columns just by looking at it. We need to determine whether an insurance claim is fraudulent or not in this "Classification Problem," where "fraud reported" is our target column because it has two categories. We will build the model using all classification algorithms since it is a classification problem.

We can count the number of rows and columns in the dataset with the df. shape function. The output indicates that there are 1000 rows and 40 columns. PCA is an option, but since the dataset is very tiny and the first lesson of a data scientist is that "Data is Crucial," I decided to proceed with all the datasets instead of risking losing any data at this moment.

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

```
(1000, 40)
```

We have specified our problem in accordance with the machine learning model's lifetime, and we have chosen the source's data for analysis. We will now execute EDA, data preprocessing, and transformation, which is the most crucial step in creating a machine learning model. If we don't achieve higher model accuracy, the model will continue to be overfitted or underfitted. We'll go into more detail about the purpose of each step later.

Exploratory Data Analysis and Data Preparation:

We will now investigate the data using some fundamental techniques before moving on to more critical analyses, such as feature extraction, imputing, and encoding.

- Let's start with checking shape, unique values, value counts, info, etc.
- • We can use the drop command to remove any unneeded columns from the datasets after analysis..

```
# 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):
```

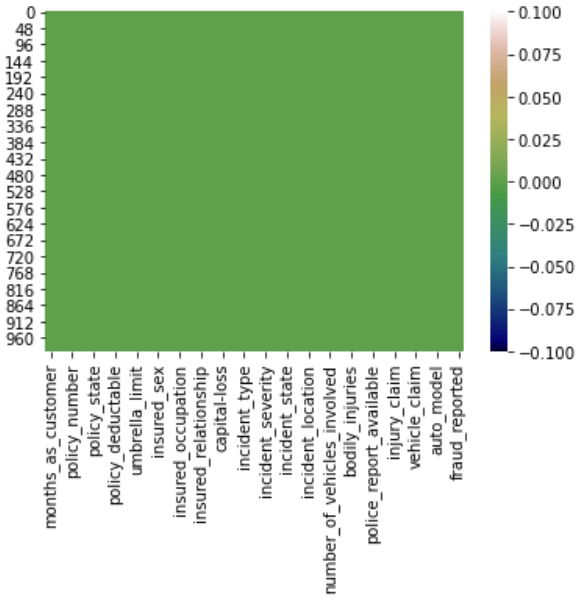
```
# Checking the type of dataset
df.dtypes
```

```
# 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)
```

- Following this basic analysis, we are currently examining the null values and will then list all the data.

```
# 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 examining the value counts for a specific column, we can see that the columns umbrella limit, capital gains, and capital loss have higher percentages of zero values than the other columns, respectively, about 79.8%, 50.8%, and 47.5%. The zero numbers in the columns for capital gains and capital losses will remain the same. Let's remove the umbrella limit columns since they contain more than 70% zero values because doing so will have no effect on the analysis.

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

6. The zip code given to each individual is listed in the column insured zip. If we examine the insured zip column's value count and unique values, we see that it has 995 unique values, indicating that the 5 items are repetitive. Since it provides some personal information, we may either remove it or change the data type from integer to object for easier 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:

Python is unable to recognise the type of this column and provide the default data type since incident date and policy bind date have object data types that belong in the DateTime data type. The values from these columns will be extracted when this object data type is converted to the Date Time data type.

```
# Converting Date columns from object type into datetime data type
df['policy_bind_date']=pd.to_datetime(df['policy_bind_date'])
df['incident_date']=pd.to_datetime(df['incident_date'])
```

Since the object data type was changed to a DateTime data type. Take the Day, Month, and Year values from each column.

```
# 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
```

From the policy bind date and incident date columns, we have retrieved Day, Month, and Year columns. We can now remove these columns.

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

Due to the existence of the character "/," the policy csl column appears in the features as an object data type even though it contains numerical information. Therefore, we will first extract two columns from the policy csl columns, csl per person and csl per accident, and then change their object data type to an integer data type.

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

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

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

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

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

Moving on to Imputation:

Imputation is a method that uses the dataset's mean, median, or mode to fill in any null values. You would suppose that when we checked the dataset for null values, we did not find any; however, we have seen from the value counts of the columns that some columns have "?" values, which are not NAN values but need to be filled in..

```
# 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

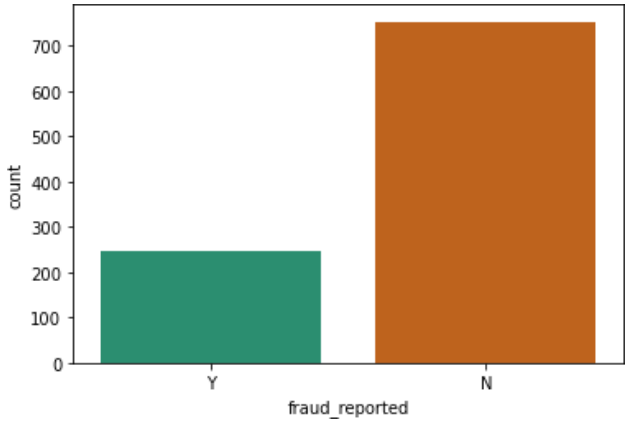
Preparing for Visualization

We will examine the numerical and category columns to visualise the attributes appropriately.

```
# 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)
```

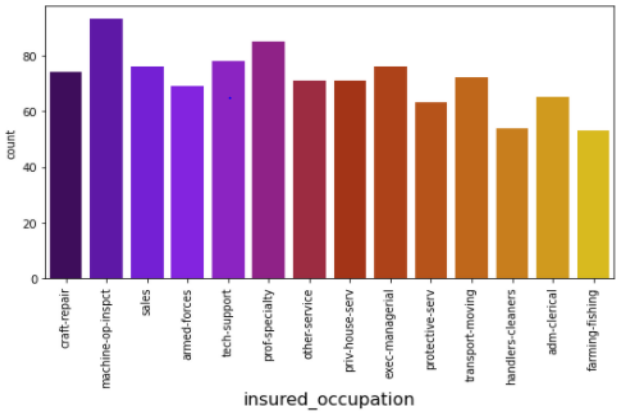
Visualization

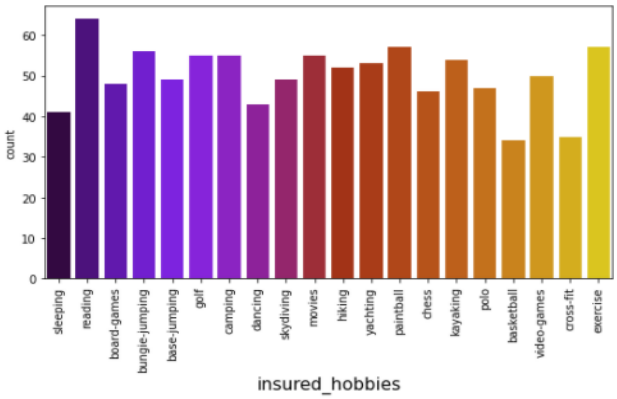
```
print(df["fraud_reported"].value_counts())
sns.countplot(df["fraud_reported"],palette="Dark2")
plt.show()
```

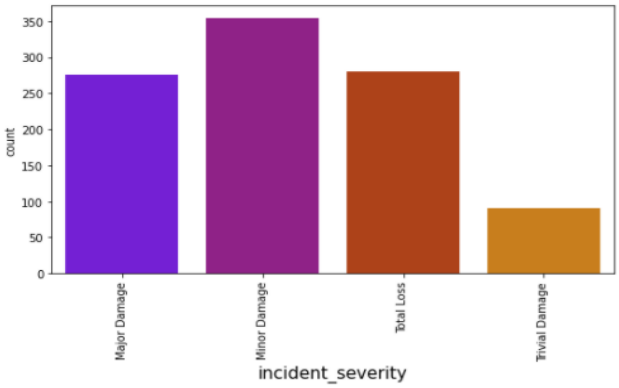


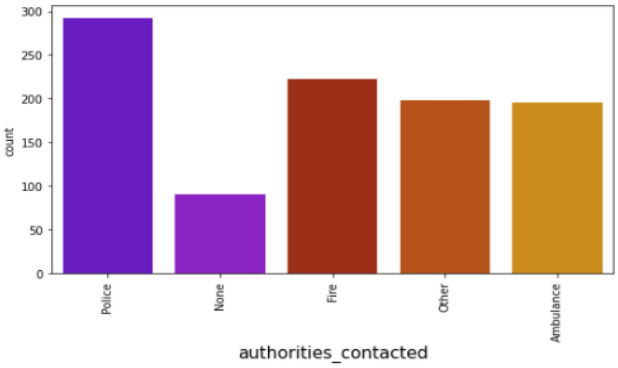
Looking at the plot, we can see that there are more "N"s than "Ys" overall. Assuming that "Y" refers for "Yes" and that the insurance is fake and that "N" stands for "No" and that the insurance claim is not fraudulent, we can conclude that the situation described below is true. The majority of insurance claims were not flagged as fraudulent. It serves as a sign that the class is unbalanced because it is our target column. We will continue to balance the data using an oversampling approach.

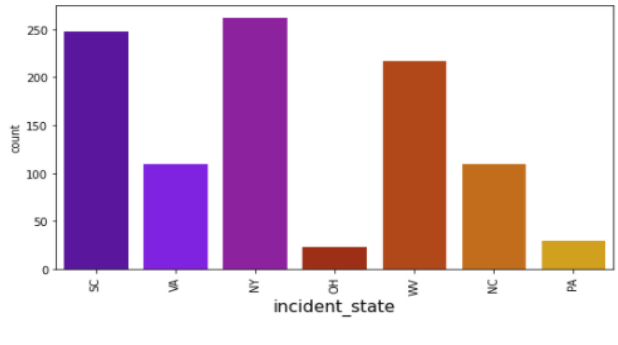
```
cols2 = ['insured_occupation', 'insured_hobbies', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_

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()
```

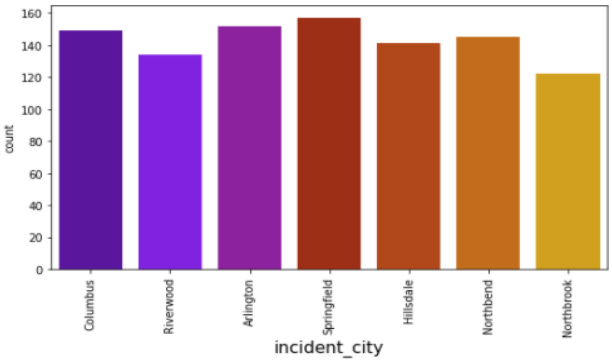

count
craft-repair
machine-op-inspct
sales
armed-forces
tech-support
prof-specialty
other-service
priv-house-serv
exec-managerial
protective-serv
transport-moving
handlers-cleaners
adm-clerical
farming-fishing
insured_occupation

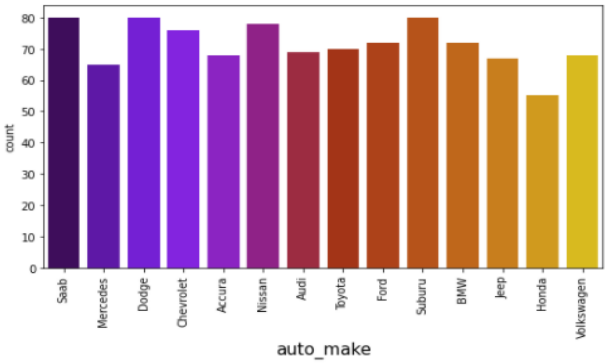

count
sleeping
reading
board-games
bungie-jumping
base-jumping
golf
camping
dancing
skydiving
movies
hiking
yachting
paintball
chess
kayaking
polo
basketball
video-games
cross-fit
exercise
insured_hobbies

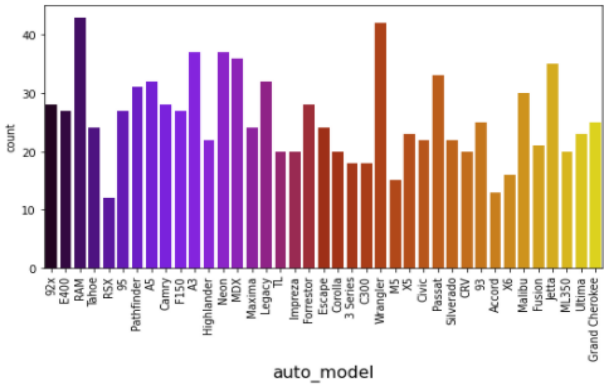

count
Major Damage
Minor Damage
Total Loss
Trivial Damage
incident_severity


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

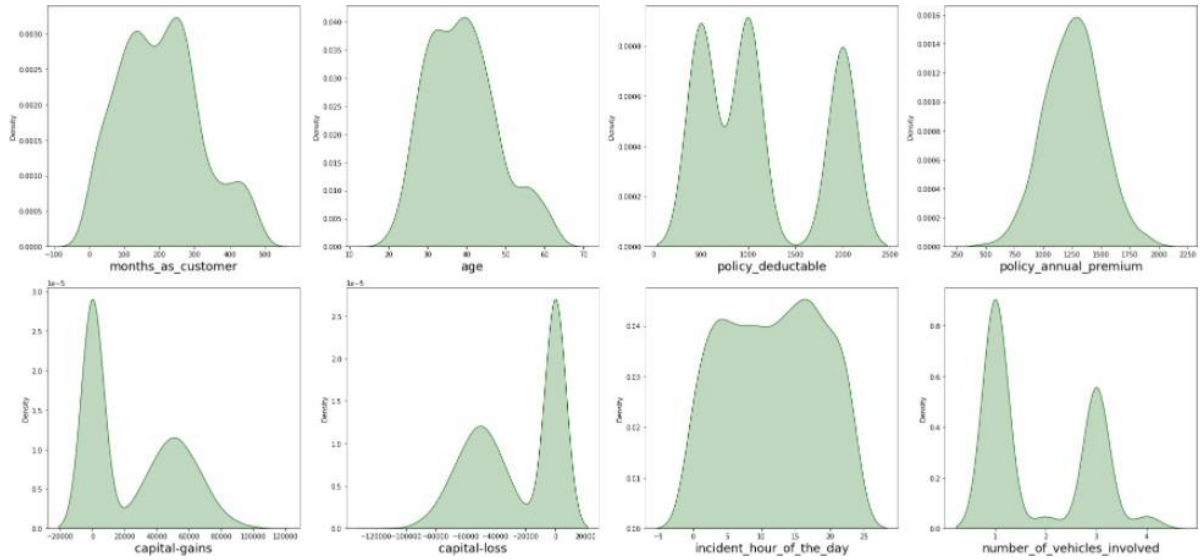

count
Saab
Mercedes
Dodge
Chevrolet
Accura
Nissan
Audi
Toyota
Ford
Suburu
BMW
Jeep
Honda
Volkswagen
auto_make

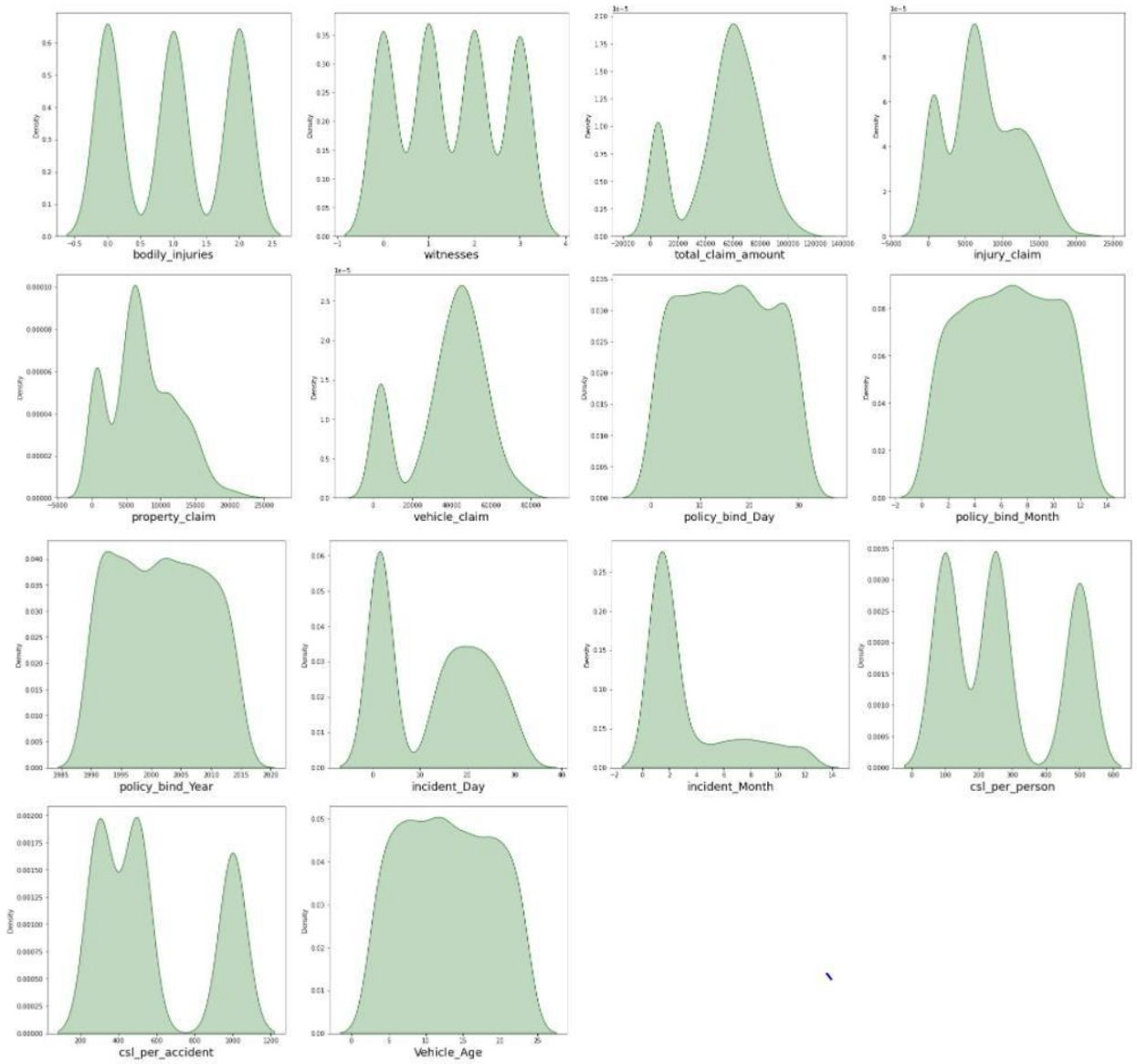

count
92x
E400
RAM
Tahoe
RSX
95
Pathfinder
A5
Camry
F150
A3
Highlander
Neon
MDX
Maxima
Legacy
TL
Impreza
Forrestor
Escape
Corolla
3 Series
C300
Wrangler
M5
X5
Civic
Passat
Silverado
CRV
93
Accord
X6
Malibu
Fusion
Jetta
ML350
Ultima
Grand Cherokee
auto_model

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.

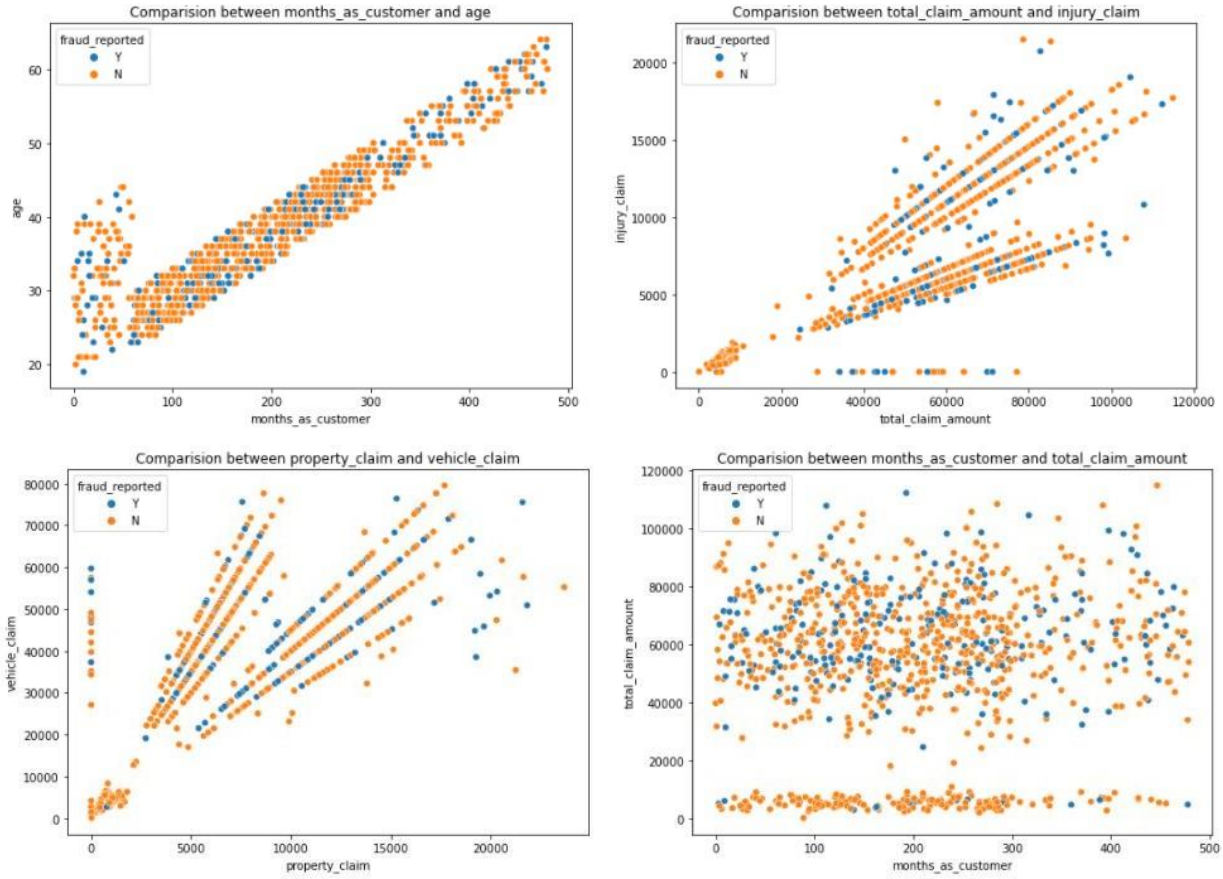
```
plt.figure(figsize=(25,35),facecolor='white')
plotnumber=1
for column in numerical_col:
    if plotnumber<=23:
        ax=plt.subplot(6,4,plotnumber)
        sns.distplot(df[column],color="darkgreen",hist=False,kde_kws={"shade": True})
        plt.xlabel(column,fontsize=18)
    plotnumber+=1
plt.tight_layout()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





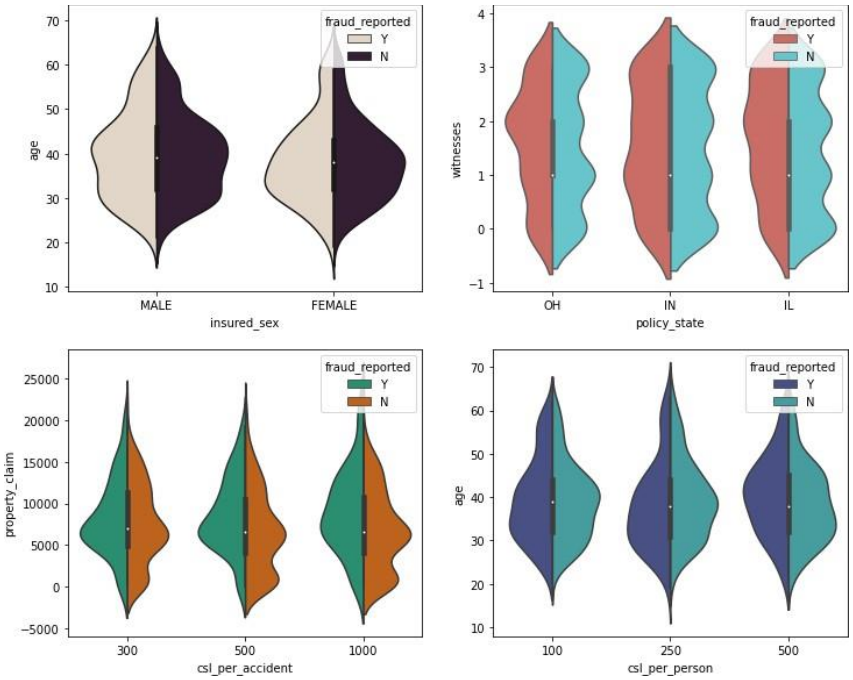
In the majority of the columns, the data is distributed regularly. Since the means of several of the columns, such as capital gains and incident months, are higher than the median, they are skewed to the right. Since the median is higher than the mean, the numbers in the capital loss column are skewed to the left. In the following section, we will use the proper techniques to remove the skewness.

```
# 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']);
```

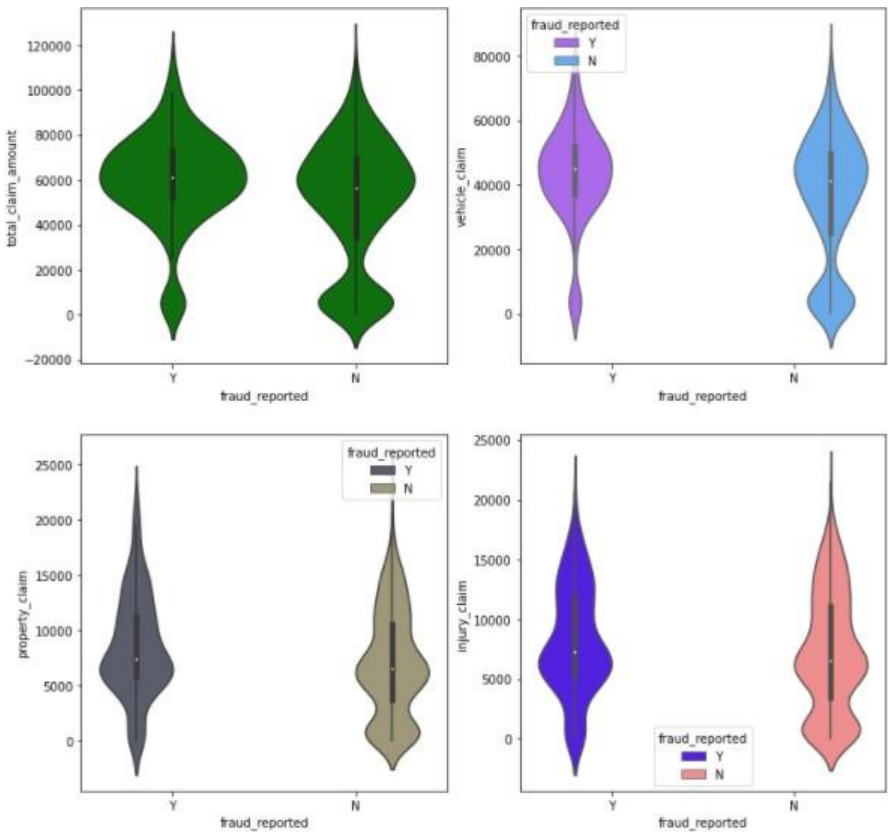


- There is a positive linear relationship between the age and month_as_customer column. As age increases the month_as customers also increases, also the fraud reported is very less in this case.
- In the second graph, we can observe the positive linear relation, as the total claim amount increases, an 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()
```

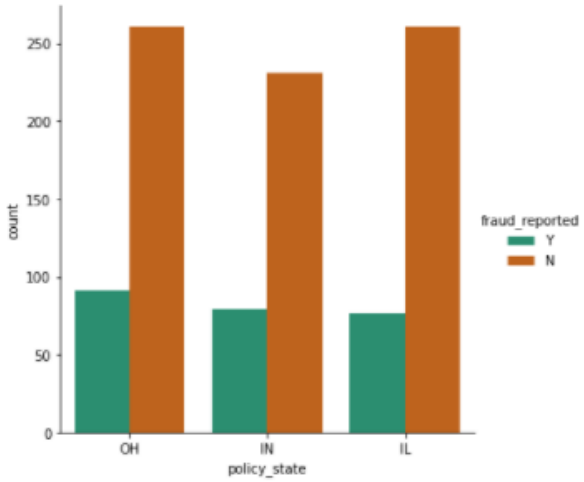



```
# Comparing insured_sex and age
sns.violinplot(x='fraud_reported',y='total_claim_amount',ax=axes[0,0],data=df,color="g")

# 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()
```

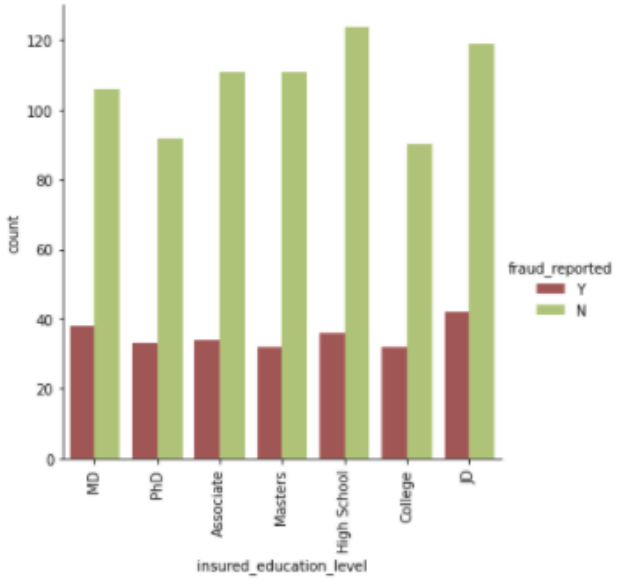


As we have seen up to this point, visualisation is a technique that makes comparison and graphing of the data self-explanatory. Before we can move on to model construction, let's move forward with a few more visualisation plots.

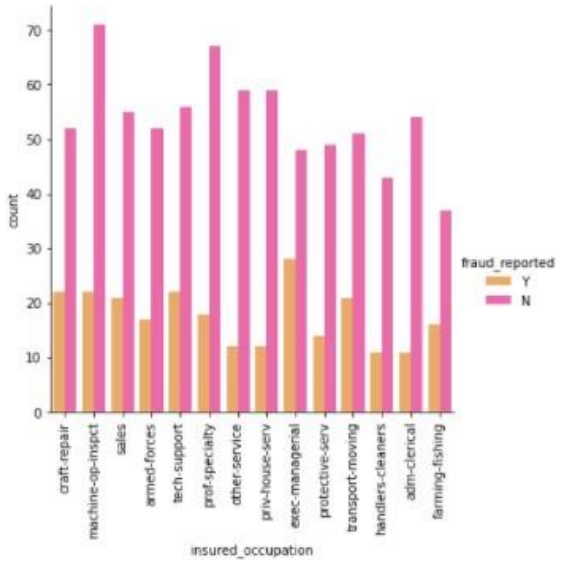
```
# Comparing policy_state and fraud_reported
sns.factorplot('policy_state',kind='count',data=df,hue='fraud_reported',palette="Dark2")
plt.show()
```



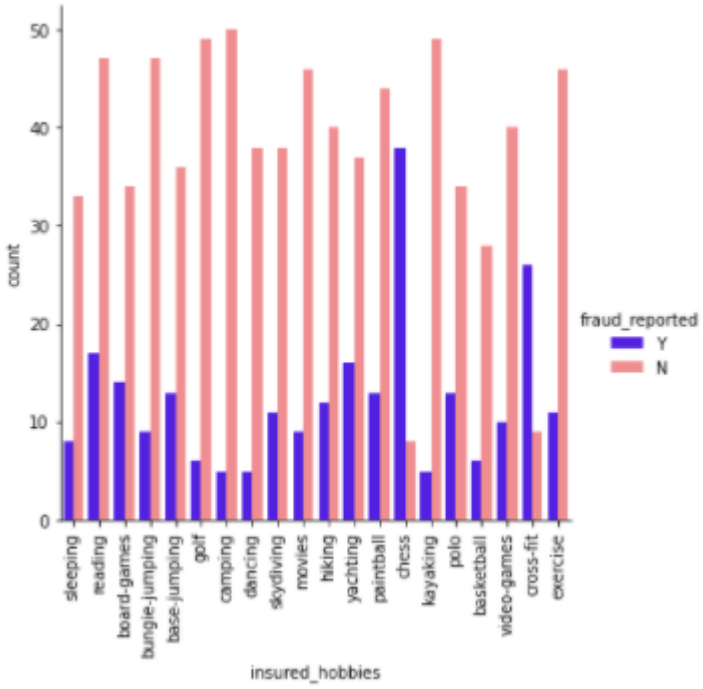

```
# Comparing insured_education_level and fraud_reported

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



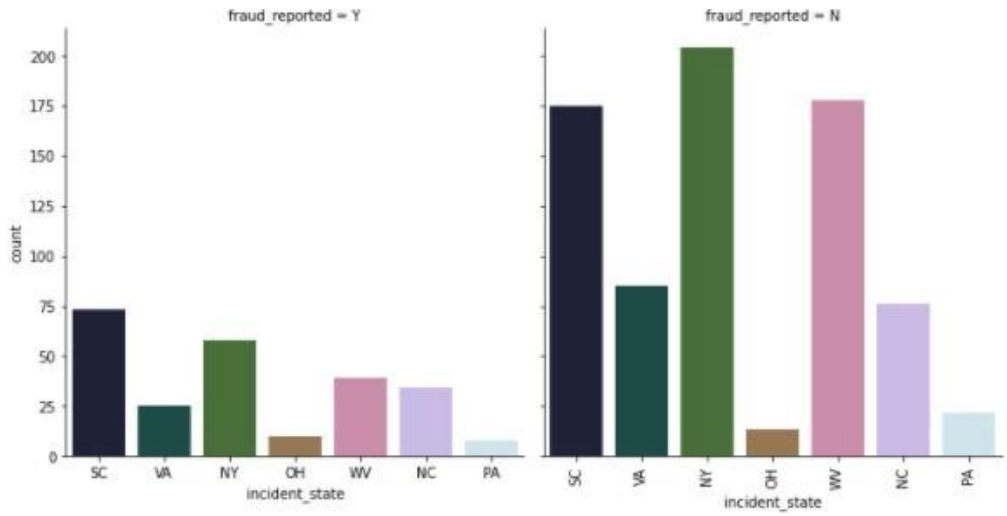
```
# Comparing insured_occupation and fraud_reported
sns.factorplot('insured_occupation',kind='count',data=df,hue='fraud_reported',palette="spring_r")
plt.xticks(rotation=90)
plt.show()
```



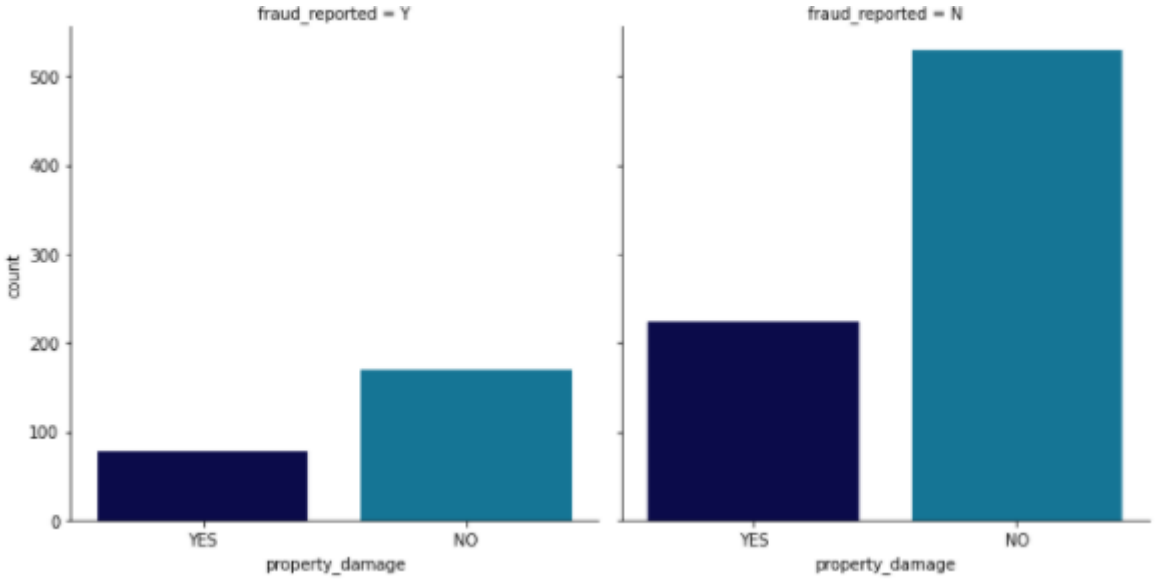

```
# Comparing insured_hobbies and fraud_reported
sns.factorplot('insured_hobbies',kind='count',data=df,hue='fraud_reported',palette="gnuplot2")
plt.xticks(rotation=90)
plt.show()
```




```
# 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()
```



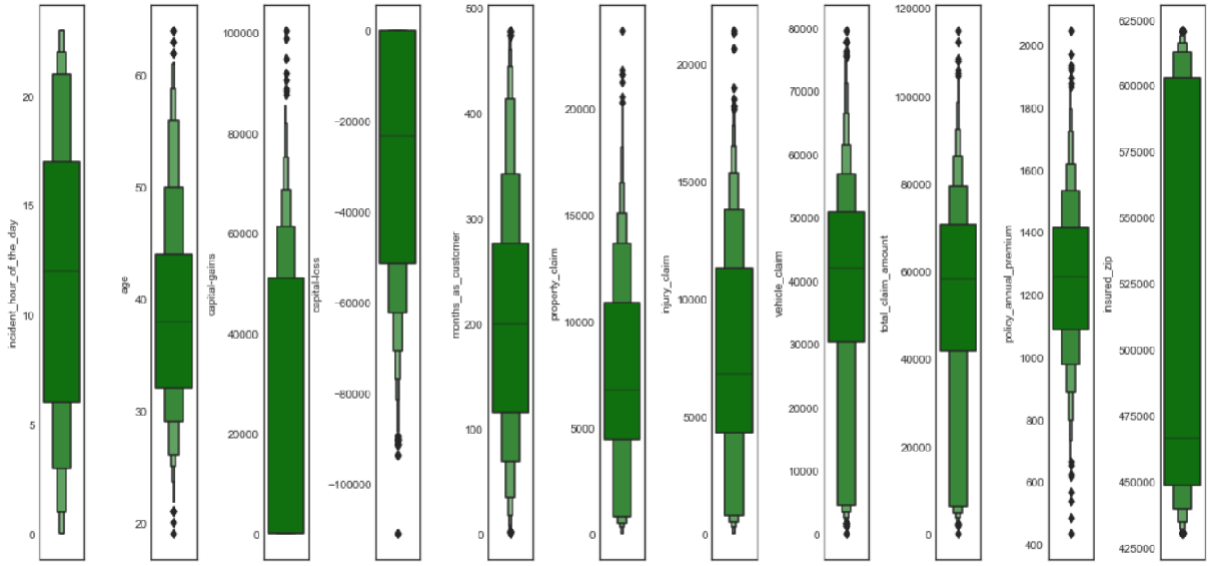

```
# Comparing property_damage and fraud_reported
sns.factorplot('property_damage',kind='count',data=df,col='fraud_reported',palette="ocean")
plt.show()
```



In order to examine and comprehend the data, we have now completed the visualisation. As a result, in this EDA section, we have examined different elements of the dataset, including searching for and imputing null values, extracting date and time, observing value counts, performing feature extraction, etc. We will now do another analysis after locating and eliminating the outliers. Along with that, we'll check the dataset for skewness and take that into account.

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()
```



The outliers can be found in the following columns because we utilised a box plot to discover them.:

- Age
- policy_annual_premium
- total_claim_amount
- property_claim
- incident_month

These are the numerical columns that have outliers, so using the Z-score approach, remove the outliers from these columns.

```
# 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.

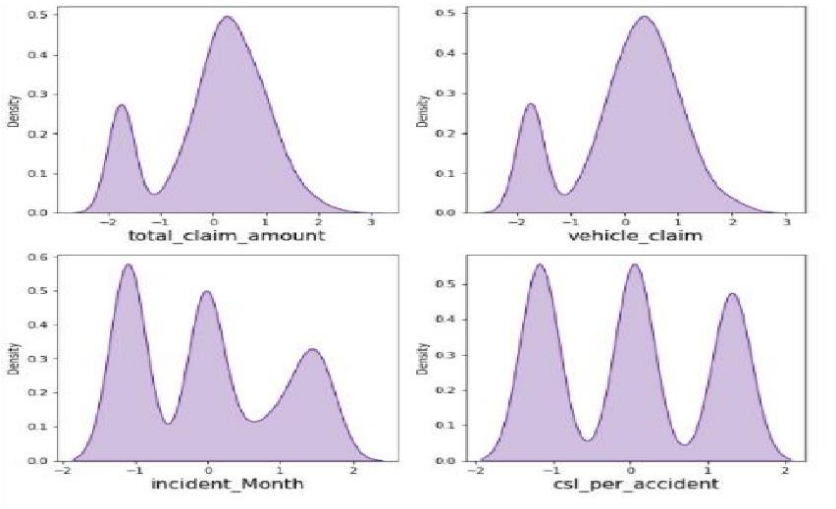
```
# Checking the skewness
new_df.skew().sort_values()
```

```
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.



In order to create a model, we have now finished our study of the dataset and cleaned the data.

As was mentioned above, there are still some issues with the dataset. As we've seen, the dataset contains both categorical and numerical information. We shall encode the data because the model can only understand numerical data. As we will observe through a heatmap and further attempt to eliminate, we have seen that there may be some multi-collinearity. Once more, we can see that the imbalance in the target variable will be corrected by oversampling. Last but not least, we will scale the data to make it suitable for training and testing.

Encoding the data

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

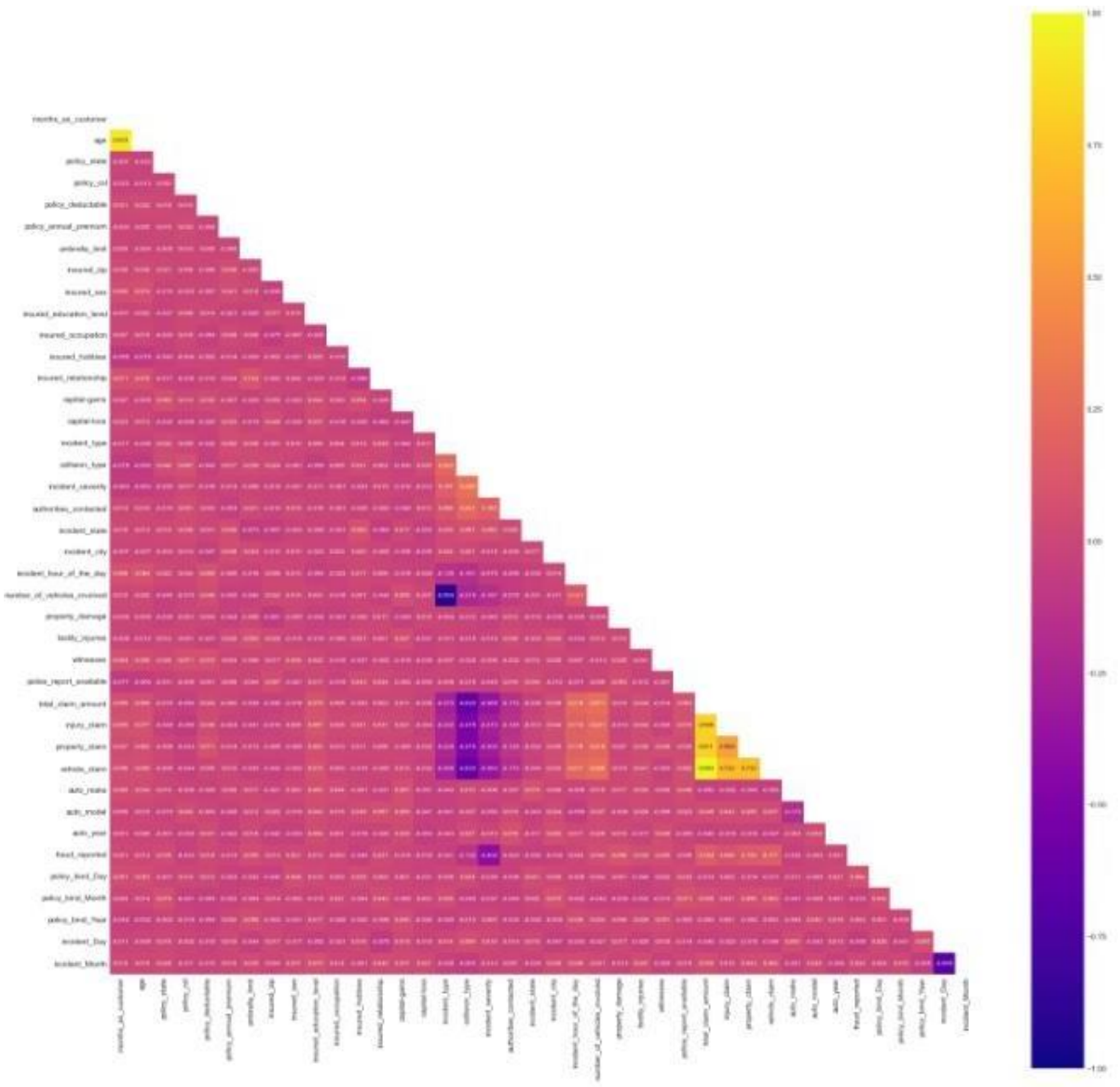
```
new_df[categorical_col].head()
```

	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.

Next, use the heatmap to check the correlation between the feature and the target as well as the relationship between the features.

```
# 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);
```



By putting the data into a heatmap, the correlation matrix is displayed. The relationship between one feature and another can be seen.

The objective and the label have virtually little in common. We can see that the majority of the columns have strong correlations with one another, which contributes to the multicollinearity issue. To solve this multicollinearity issue, we shall examine the VIF value.

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()
```

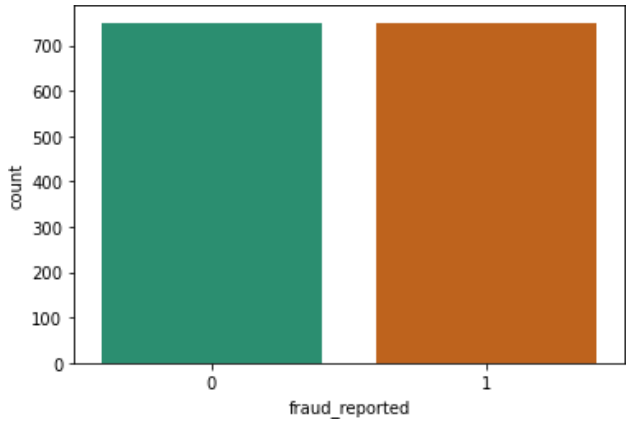
Checking Multi-colinearity using VIF

```
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 has been noted that certain columns have VIF values above 10, which indicates a multicollinearity issue. Drop the feature that has the highest VIF value across all columns.
- With a colinearity of greater than 10, I eliminated the total claim amount and csl per accident features, and the issue is now resolved.
- 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)
```

• The data now appears to be accurate because we used SMOTE to address the oversampling problem.



- 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
```

Given that the accuracy score at the random state of 49 was 79%, we utilised the RandomForestClassifier to determine which random state was the best. Let's construct our models using this random state. 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.

Since we are insatiable data scientists, we won't be content with just one model. The first model was created using RandomForestClassifier, which provided an accuracy score of 91%. We'll test different models to determine how accurate they are.

```
# 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.

This model, ExtraTreesClassifier, outperformed RandomForestClassifier in terms of accuracy with a score of 93%.

Cross-validation of all the created models will be done in order to ensure that the model is not overfitted before we can declare one to be the best.

```
# 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 improvethe 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_
```

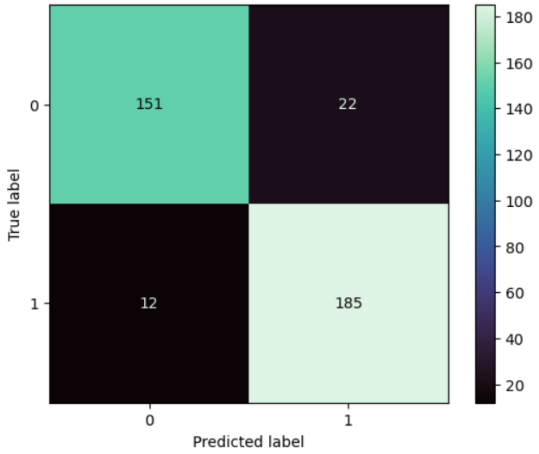
```
{'criterion': 'gini',
 'max_depth': 20,
 'max_features': 'log2',
 'n_estimators': 300,
 'n_jobs': -2}
```

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

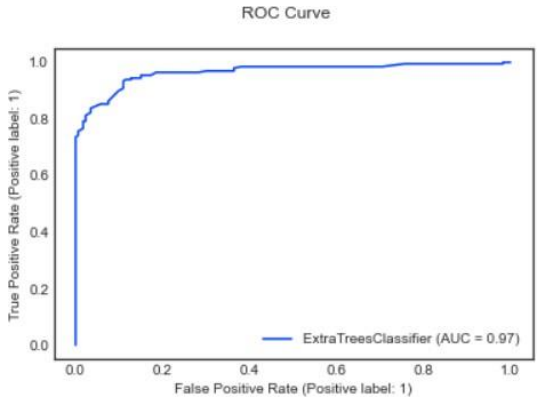
```
Insurance = ExtraTreesClassifier(criterion='gini', max_depth=20, max_features='log2', n_estimators=300, n_jobs=-2)
Insurance.fit(x_train, y_train)
pred = Insurance.predict(x_test)
acc=accuracy_score(y_test,pred)
print(acc*100)
```

```
92.0
```

We have built our final model and we can see that the accuracy scores have increased by 1% fromthe 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, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 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, 1, 0, 0, 0, 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, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0,
   1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 0, 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, 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, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 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:-

- The lifespan of a machine learning model was described at the beginning of this blog. You can see how we briefly touched on each stage before moving on to model building and deployment.

- This industry sector requires a clear understanding of the data, and the most important step in any model construction is Data Analysis and Feature Engineering.

- You can see how we handled category and numerical data as well as how we created various machine learning models using the same dataset.

- With the help of hyper parameter adjustment, we can increase the accuracy of our models; in the case of this model, accuracy remained the same.

- We can predict with ease if an insurance claim is fake or not using this machine learning model, and we can reject applications for those claims.

References:

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