Insurance Claim Fraud Detection

By

NIDHI KATIYAR

BATCH-1843

Problem Definition

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

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

INTRODUCTION

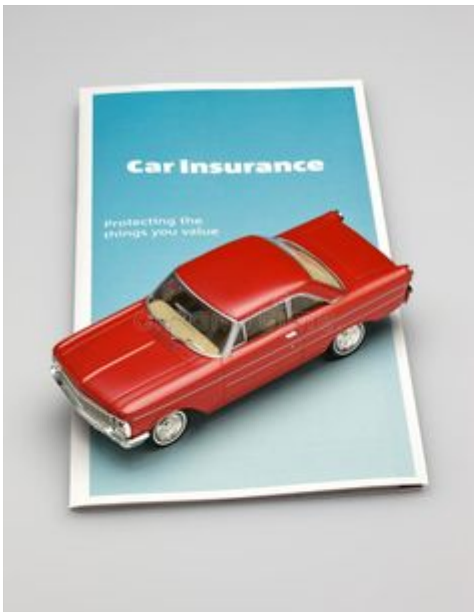
According to the Insurance Information Institute, "Insurance fraud is a deliberate deception perpetrated against or by an insurance company or agent for the purpose of financial gain." Fraud may be committed at different points by applicants, policyholders, third-party claimants, or professionals who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

Common frauds include "padding," or inflating claims; misrepresenting facts on an insurance application; submitting claims for injuries or damage that never occurred; and staging accidents.

People who commit insurance fraud include:

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

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



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

Insurance fraud can be broadly classified into 2 types.

- Soft Insurance Fraud
- Hard Insurance Fraud

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

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

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

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

PROBLEM DEFINITION

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

In this project, we are provided with a dataset which has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

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

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

DATA ANALYSIS

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

IMPORTING NECESSARY LIBRARIES

We import the libraries necessary for data analysis

```
In [1]: # Importing Necessary Libraries
        import pandas as pd
        import numpy as np
        import seaborn as sns
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings('ignore')
```

IMPORTING THE DATASET

We import the Dataset

```
In [2]: # Dataset
        df_icf=pd.read_csv("Automobile_insurance_fraud.csv")
        pd.set_option("display.max_columns",None)
        df_icf
```

Out[2]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl
0	328	48	521585	17-10-2014	OH	250/500
1	228	42	342868	27-06-2006	IN	250/500
2	134	29	687698	06-09-2000	OH	100/300
3	256	41	227811	25-05-1990	IL	250/500
4	228	44	367455	06-06-2014	IL	500/1000
...	...	...	...	...	...	...
995	3	38	941851	16-07-1991	OH	500/1000
996	285	41	186934	05-01-2014	IL	100/300
997	130	34	918516	17-02-2003	OH	250/500
998	458	62	533940	18-11-2011	IL	500/1000
999	456	60	556080	11-11-1996	OH	250/500

1000 rows × 40 columns

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

DATA PREPARATION & CLEANING

We check the columns present in the dataset

```
In [7]: # Column Names
        df_icf.columns

Out[7]: Index(['months_as_customer', 'age', 'policy_number', 'policy_bind_date',
               'policy_state', 'policy_csl', 'policy_deductable',
               'policy_annual_premium', 'umbrella_limit', 'insured_zip', 'insured_sex',
               'insured_education_level', 'insured_occupation', 'insured_hobbies',
               'insured_relationship', 'capital-gains', 'capital-loss',
               'incident_date', 'incident_type', 'collision_type', 'incident_severity',
               'authorities_contacted', 'incident_state', 'incident_city',
               'incident_location', 'incident_hour_of_the_day',
               'number_of_vehicles_involved', 'property_damage', 'bodily_injuries',
               'witnesses', 'police_report_available', 'total_claim_amount',
               'injury_claim', 'property_claim', 'vehicle_claim', 'auto_make',
               'auto_model', 'auto_year', 'fraud_reported', '_c39'],
              dtype='object')
```

Data types of each column

```
In [9]: # Column Data Types
        df_icf.dtypes

Out[9]: months_as_customer               int64
        age                              int64
        policy_number                    int64
        policy_bind_date                object
        policy_state                    object
        policy_csl                      object
        policy_deductable                int64
        policy_annual_premium          float64
        umbrella_limit                   int64
        insured_zip                      int64
        insured_sex                     object
        insured_education_level         object
        insured_occupation              object
        insured_hobbies                 object
        insured_relationship            object
        capital-gains                    int64
        capital-loss                     int64
        incident_date                   object
        incident_type                   object
        collision_type                  object
        incident_severity               object
        authorities_contacted           object
        incident_state                  object
        incident_city                   object
        incident_location               object
        incident_hour_of_the_day         int64
        number_of_vehicles_involved      int64
        property_damage                 object
        bodily_injuries                  int64
        witnesses                        int64
        police_report_available         object
        total_claim_amount               int64
        injury_claim                     int64
        property_claim                   int64
        vehicle_claim                    int64
        auto_make                       object
        auto_model                      object
        auto_year                        int64
        fraud_reported                  object
        _c39                           float64
        dtype: object
```

➢ We Will check for null values present in the dataset, sum of such null values (if present) in the dataset & a visual heat map of the null values.

In [11]:
```
# Sum of null values
df_icf.isnull().sum()
```

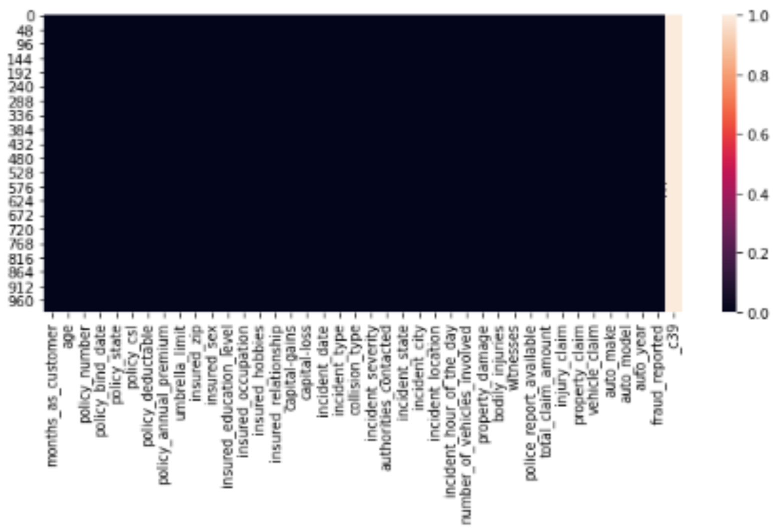
Out[11]:
```
months_as_customer             0
age                            0
policy_number                  0
policy_bind_date               0
policy_state                   0
policy_csl                     0
policy_deductable              0
policy_annual_premium          0
umbrella_limit                 0
insured_zip                    0
insured_sex                    0
insured_education_level        0
insured_occupation             0
insured_hobbies                0
insured_relationship           0
capital-gains                  0
capital-loss                   0
incident_date                  0
incident_type                  0
collision_type                 0
incident_severity              0
authorities_contacted          0
incident_state                 0
incident_city                  0
incident_location              0
incident_hour_of_the_day       0
number_of_vehicles_involved    0
property_damage                0
bodily_injuries                0
witnesses                      0
police_report_available        0
total_claim_amount             0
injury_claim                   0
property_claim                 0
vehicle_claim                  0
```


In [12]:
```
# Visualizing the null values
plt.figure(figsize=[10,4])
sns.heatmap(df_icf.isnull())
```

Out[12]: <AxesSubplot:>

In [12]:
```
# Visualizing the null values
plt.figure(figsize=[10,4])
sns.heatmap(df_icf.isnull())
```

Out[12]: <AxesSubplot:>



Next, we check the statistical information using “df_icf.info( )”.

```
In [13]: df_icf.info()
```

After running df_icf.info( ), I found the column “c_39” having one unique count as NAN throughout the dataset and it is of no use, so I dropped that column.

```
In [14]: # Dropping column
         df_icf = df_icf.drop(["_c39"],axis=1)
```

Next, we see the unique values present in each column of the dataset.

```
In [15]: #Checking unique values of each column
         df_icf.nunique()

Out[15]: months_as_customer              391
         age                              46
         policy_number                  1000
         policy_bind_date                951
         policy_state                      3
         policy_csl                        3
         policy_deductable                 3
         policy_annual_premium           991
         umbrella_limit                   11
         insured_zip                     995
         insured_sex                       2
         insured_education_level           7
         insured_occupation               14
         insured_hobbies                  20
         insured_relationship              6
         capital-gains                   338
         capital-loss                    354
         incident_date                    60
         incident_type                     4
         collision_type                    4
         incident_severity                 4
         authorities_contacted             5
         incident_state                    7
         incident_city                     7
         incident_location              1000
         incident_hour_of_the_day         24
         number_of_vehicles_involved       4
         property_damage                   3
         bodily_injuries                   3
         witnesses                         4
         police_report_available           3
         total_claim_amount              763
         injury_claim                    638
         property_claim                  626
         vehicle_claim                   726
         auto_make                        14
         auto_model                       39
         auto_year                        21
         fraud_reported                    2
         dtype: int64
```


After running df_icf.nunique( ), we see the columns “policy_number” and “incident_location” have 1000 unique counts which means all the values in these categorical columns are unique. We can drop these columns as they would be not of much use in model building.

```
In [16]: #Droping policy_number and incident_location column
         df_icf = df_icf.drop(["policy_number"],axis=1)
         df_icf = df_icf.drop(["incident_location"],axis=1)
```

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

In [17]:
```
df_icf.describe()
```

Out[17]:

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-loss	incident_hou
count	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	1000.000000	1000.000000	1000.000000	
mean	203.954000	38.948000	1136.000000	1256.406150	1.101000e+06	501214.488000	25126.100000	-26793.700000	
std	115.113174	9.140287	611.864673	244.167395	2.297407e+06	71701.610941	27872.187708	28104.096686	
min	0.000000	19.000000	500.000000	433.330000	-1.000000e+06	430104.000000	0.000000	-111100.000000	
25%	115.750000	32.000000	500.000000	1089.607500	0.000000e+00	448404.500000	0.000000	-51500.000000	
50%	199.500000	38.000000	1000.000000	1257.200000	0.000000e+00	466445.500000	0.000000	-23250.000000	
75%	276.250000	44.000000	2000.000000	1415.695000	0.000000e+00	603251.000000	51025.000000	0.000000	
max	479.000000	64.000000	2000.000000	2047.590000	1.000000e+07	620962.000000	100500.000000	0.000000	

In [18]:
```
# Mean of our target variable 'fraud_reported'
df_icf.groupby('fraud_reported').mean()
```

Out[18]:

fraud_reported	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-loss	incident
N	202.600266	38.884462	1130.810093	1258.430000	1.023904e+06	500419.537849	25432.005312	-26554.581673	
Y	208.080972	39.141700	1151.821862	1250.236275	1.336032e+06	503637.959514	24193.522267	-27522.672065	

From the above output we find the following observations:

- Here the counts of all the columns are equal which means there are no missing values in the dataset.
- In the columns "policy deductible", "capital-gains", "injury _claim" etc we can observe the mean value is greater than the median (50%) which means the data in those columns are skewed to the right.
- And in the columns "total_ claim _amount", "vehicle _claim" etc we can observe the median is greater than the mean which means the data in the columns are skewed to the left.
- And in some of the columns the mean and median are equal, which means the data is symmetric and is normally distributed and no skew- ness present.

After this, we check the value counts of every column to see whether there is a need for feature extraction & feature engineering.

In [19]:
```
# Value counts of each column.
for i in df_icf.columns:
        print(df_icf[i].value_counts())
        print('-----------------------------------------------------------')
```

By running the above for loop, you will get the value counts of all the columns present in the dataset.

Looking into the value counts of each column, we see that the column "umbrella _limit" contains about 80% of zero values. It might create a skew -ness problem in the data so it seemed better to drop this column.

```
In [20]: # Droping umbrella_limit column
         df_icf=df_icf.drop(["umbrella_limit"],axis=1)
```

Also, the column "insured _zip", contains the zip code given to each person. If we take a look at the value count and unique values of the column, it contains 995 unique values that mean the 5 entries are repeating. Since it is giving information about the identity of the person, it is not important for the processing so we can drop this column as well.

```
In [21]: # Dropping insured_zip column as it is not important for the prediction
         df_icf.drop('insured_zip',axis=1,inplace=True)
```

By looking at the dataset and value counts of the various columns, we see some columns having "?" signs. These are not to be considered as NAN values but we need to fill them.

The columns, "collision type", "property_damage" & "police_report_available" contain the "?" sign. Since these columns seem to be categorical, we will replace "?" values with most frequently occurring values of the respective columns that are their mode values. In some of these columns the mode and the "?" values are the same so we shall replace the "?" values with the second highest occurring values in the respective columns.

```
In [22]: # Mode of column
         df_icf["collision_type"].mode()

Out[22]: 0    Rear Collision
         dtype: object

In [23]: # Replacing '?' with mode value
         df_icf['collision_type'] = df_icf.collision_type.str.replace('?', 'Rear Collision')

In [24]: #Checking the value counts of property_damage column
         df_icf.property_damage.value_counts()

Out[24]: ?      360
         NO     338
         YES    302
         Name: property_damage, dtype: int64

In [25]: #Replacing '?' with mode value
         df_icf['property_damage'] = df_icf.property_damage.str.replace('?', 'NO')

In [26]: #Checking the value counts of police_report_available column
         df_icf.police_report_available.value_counts()

Out[26]: ?      343
         NO     343
         YES    314
         Name: police_report_available, dtype: int64

In [27]: #Replacing '?' with mode value
         df_icf['police_report_available'] = df_icf.police_report_available.str.replace('?', 'NO')
```

We have now replaced all the "?" values with the respective modes of the respective columns.

Now let us do some feature extraction, we shall first convert the columns, “policy _bind _date” & “incident _date” from object data type to Date Time data type, and extract the year, month and day from these columns. And finally we dropping these columns after extraction.

```
In [28]: import datetime as dt
```

```
In [29]: #Converting object data type to datetime
         df_icf['policy_bind_date'] =  pd.to_datetime(df_icf['policy_bind_date'])
         df_icf['incident_date'] =  pd.to_datetime(df_icf['incident_date'])
```

```
In [31]: # Extracting year
         df_icf["policy_bind_year"]=pd.to_datetime(df_icf.policy_bind_date, format="%d/%m/%Y").dt.year

         # Extracting month
         df_icf["policy_bind_month"]=pd.to_datetime(df_icf.policy_bind_date, format="%d/%m/%Y").dt.month

         # Extracting day
         df_icf["policy_bind_day"]=pd.to_datetime(df_icf.policy_bind_date, format="%d/%m/%Y").dt.day
```

```
In [32]: # Droping policy_bind_date column after extraction
         df_icf = df_icf.drop(["policy_bind_date"],axis=1)
```

```
In [33]: # Extracting year
         df_icf["incident_year"]=pd.to_datetime(df_icf.incident_date, format="%d/%m/%Y").dt.year

         # Extracting month
         df_icf["incident_month"]=pd.to_datetime(df_icf.incident_date, format="%d/%m/%Y").dt.month

         # Extracting day
         df_icf["incident_day"]=pd.to_datetime(df_icf.incident_date, format="%d/%m/%Y").dt.day
```

```
In [34]: # Droping incident_date column after extraction
         df_icf = df_icf.drop(["incident_date"],axis=1)
```

The column “incident _year” has only one unique value, so we shall drop this column as it will not be useful for model building.

```
In [35]: # Droping incident_year column after extraction
         df_icf = df_icf.drop(["incident_year"],axis=1)
```

Next, we shall extract the “ csl _per _person” & “csl _per _accident” from the column “policy _csl”. After extraction we shall change the data type of these columns to “integer“ data type. Finally we shall drop the “policy _csl” column.

```
In [36]: # Extracting columns from policy_csl
         df_icf['csl_per_person'] = df_icf.policy_csl.str.split('/', expand=True)[0]
         df_icf['csl_per_accident'] = df_icf.policy_csl.str.split('/', expand=True)[1]

In [37]: # Changing dtype of extracted column
         df_icf[['csl_per_person']] = df_icf[['csl_per_person']].astype('int64')
         df_icf[['csl_per_accident']] = df_icf[['csl_per_accident']].astype('int64')

In [38]: # Droping policy_csl column after extraction
         df_icf=df_icf.drop(["policy_csl"],axis=1)
```

We then shall extract the “auto _age” from the column “auto _year”. Since the data belongs to the year 2018 we shall subtract the auto year from the year 2018 to get the auto age.

```
In [39]: df_icf['auto_age'] = 2018 - df_icf['auto_year']

In [40]: df_icf['auto_age']

Out[40]: 0      14
         1      11
         2      11
         3       4
         4       9
                ..
         995    12
         996     3
         997    22
         998    20
         999    11
         Name: auto_age, Length: 1000, dtype: int64

In [41]: # Droping auto_year column after extraction
         df_icf = df_icf.drop(["auto_year"],axis=1)
```


Lastly, before Data Visualisation we shall see the unique values present in our target variable column, the value counts of these unique values & lastly, if there are any empty observations in the target variable column.

```
In [43]: df_icf["fraud_reported"].unique()

Out[43]: array(['Y', 'N'], dtype=object)

In [44]: df_icf['fraud_reported'].value_counts()

Out[44]: N    753
         Y    247
         Name: fraud_reported, dtype: int64

In [45]: #Checking for any empty observation in target column
         df_icf.loc[df_icf['fraud_reported'] == " "]

Out[45]:
```

months_as_customer	age	policy_state	policy_deductable	policy_annua

No empty observations in target column.

DATA VISUALISATION

Now we visualise our data. For visualisation we shall divide the columns into categorical columns and numerical columns to make visualisation better.

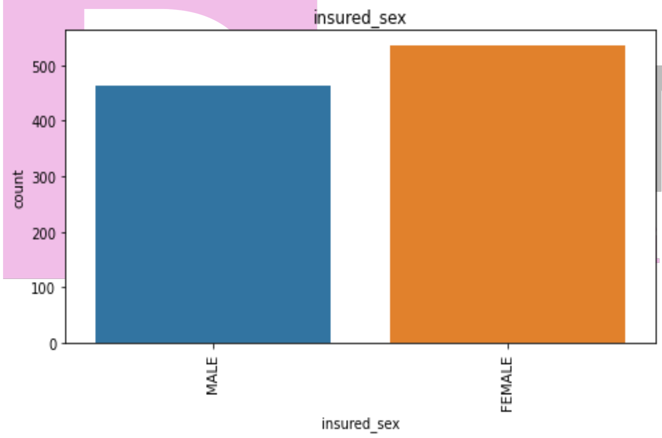
In [47]:
```python
# Categorical Columns
categorical_columns=[]
for i in df_icf.dtypes.index:
    if df_icf.dtypes[i]=='object':
        categorical_columns.append(i)
print(categorical_columns)
```

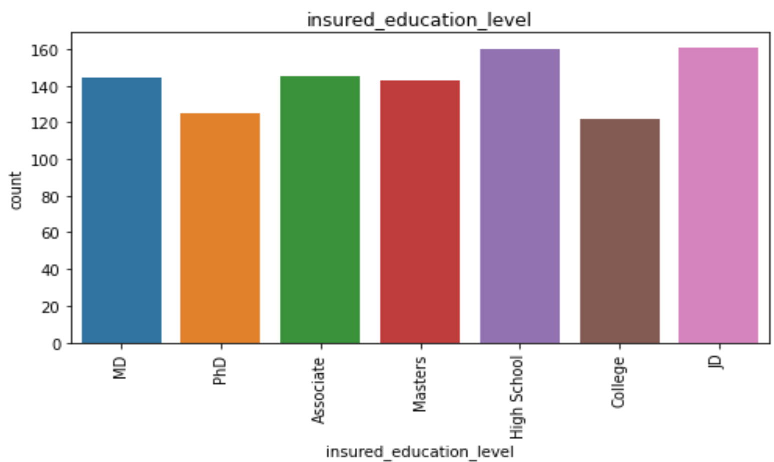
```
['policy_state', 'insured_sex', 'insured_education_level', 'insured_occupation', 'insured_hobbies', 'insured_relationship', 'in
cident_type', 'collision_type', 'incident_severity', 'authorities_contacted', 'incident_state', 'incident_city', 'property_dama
ge', 'police_report_available', 'auto_make', 'auto_model', 'fraud_reported']
```

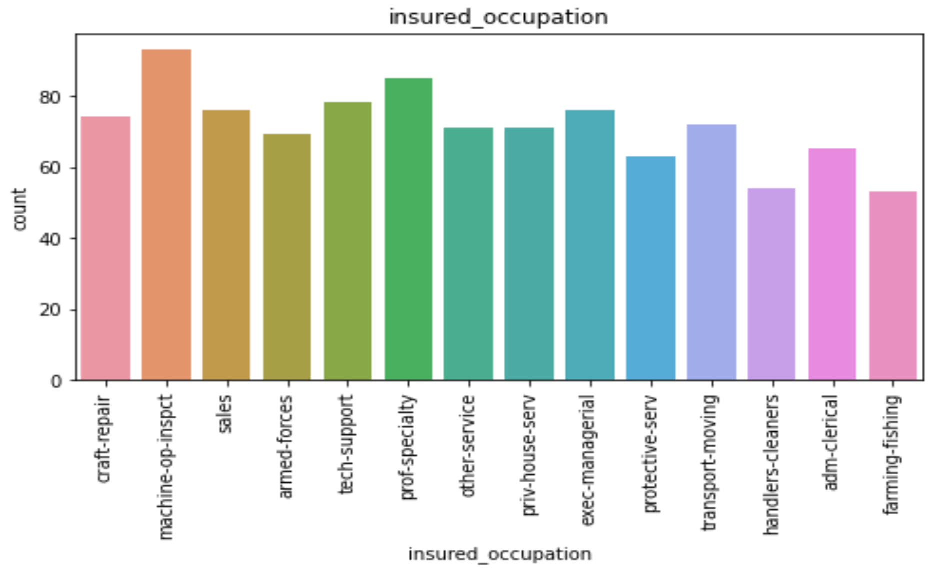
In [48]:
```python
# Numerical Columns
numerical_columns=[]
for i in df_icf.dtypes.index:
    if df_icf.dtypes[i]!='object':
        numerical_columns.append(i)
print(numerical_columns)
```

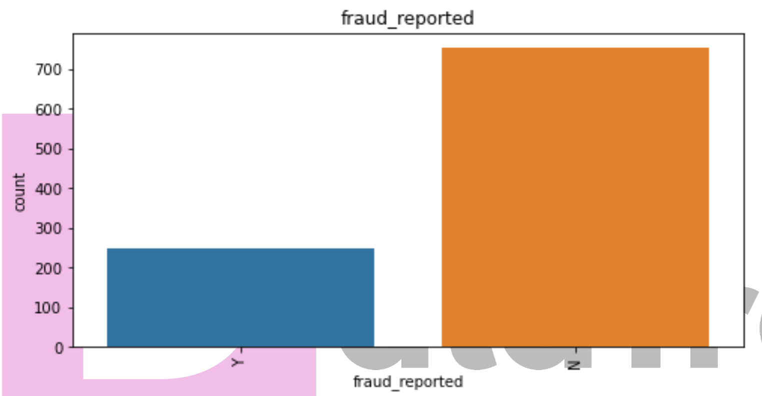
```
['months_as_customer', 'age', 'policy_deductable', 'policy_annual_premium', 'capital-gains', 'capital-loss', 'incident_hour_of_
the_day', 'number_of_vehicles_involved', 'bodily_injuries', 'witnesses', 'total_claim_amount', 'injury_claim', 'property_clai
m', 'vehicle_claim', 'policy_bind_year', 'policy_bind_month', 'policy_bind_day', 'incident_month', 'incident_day', 'csl_per_per
son', 'csl_per_accident', 'auto_age']
```

Now for Univariate analysis, we use count plots for plotting the categorical columns of our dataset. Here are a few plots of categorical data:

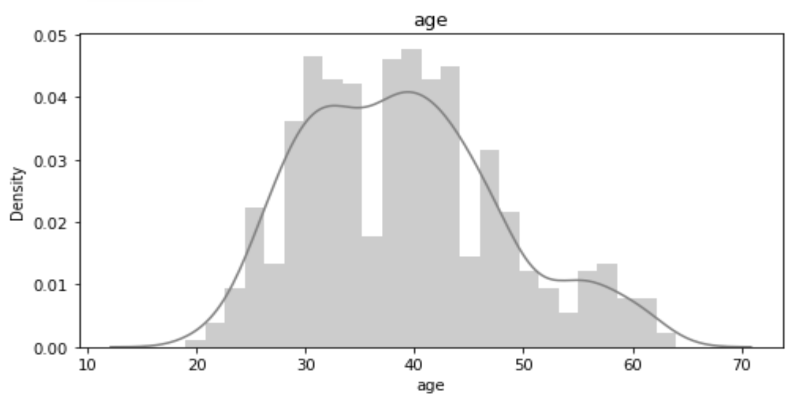


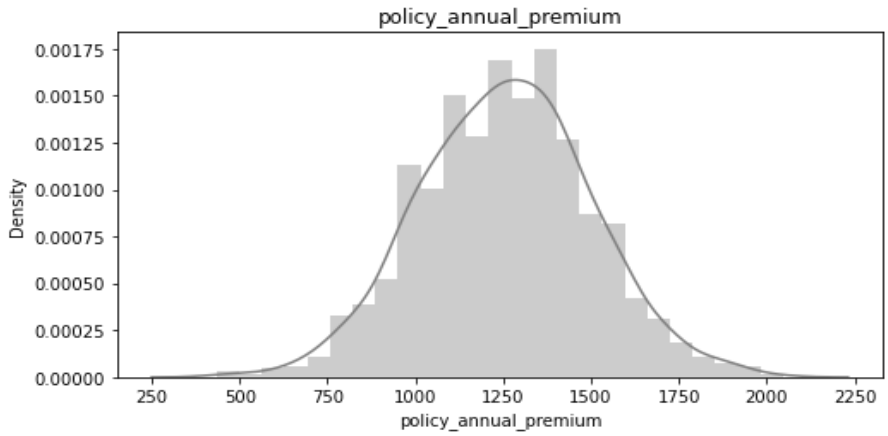


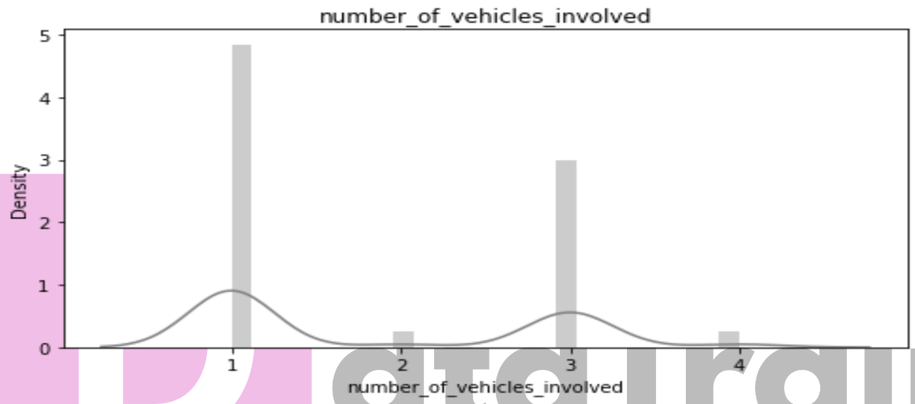


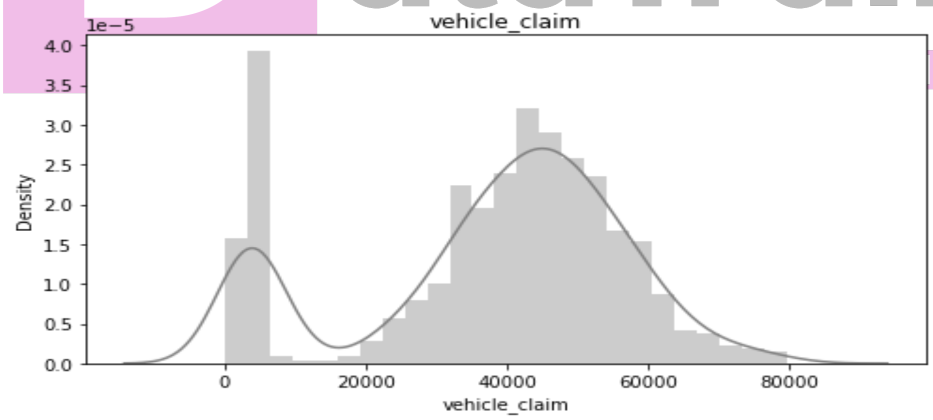


We use dist plots for the numerical columns in our dataset .Here are a few plots of numerical data:

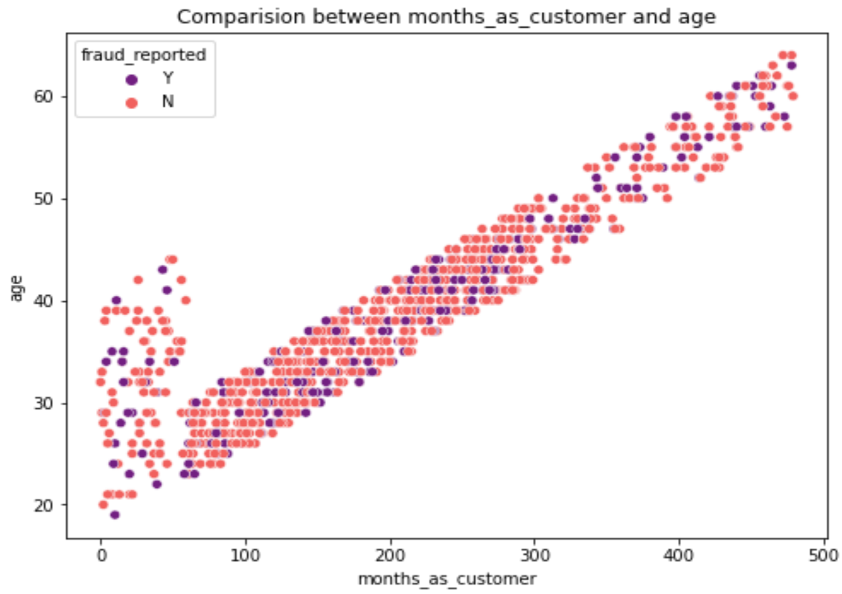




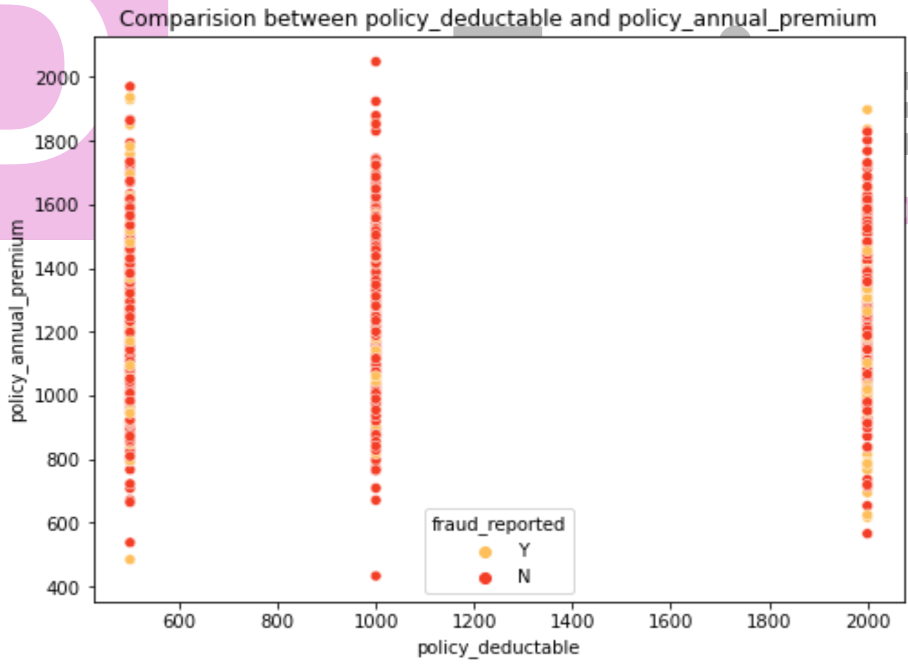




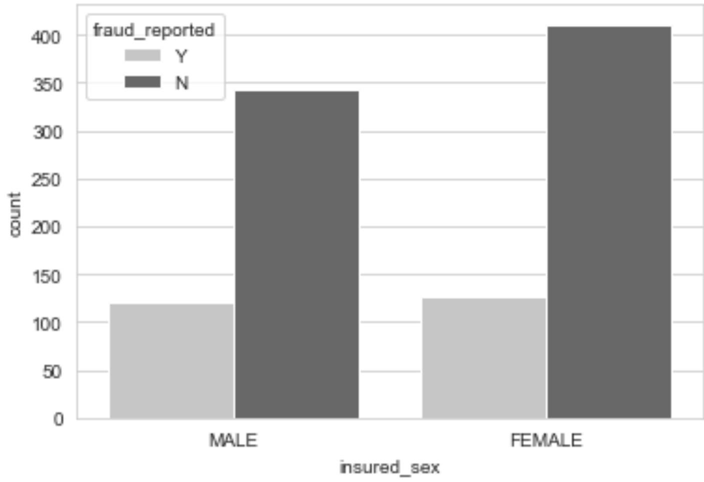
Next we conduct bivariate analysis. For bivariate analysis we used scatter plots & count plots for visualisation.



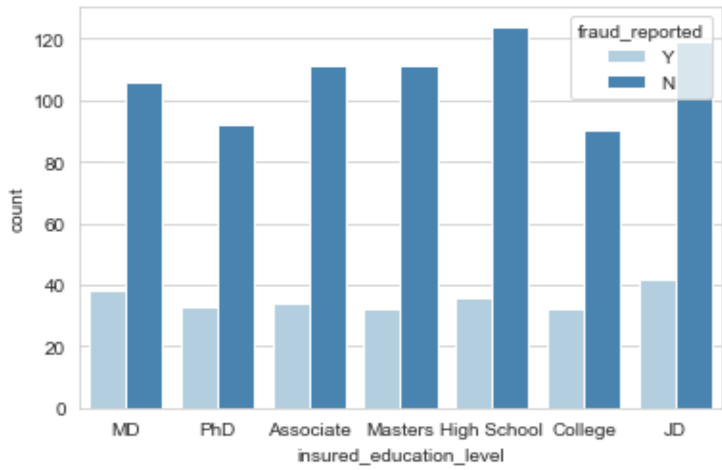
From the above scatter plot, we can observe a strong linear relationship between the “age” and “month _as_ customer”. As, month _as_ customer increases, the age of the person also increases. Also, as the person gets older, the frequency of the both fraud reported classes are vanishing slowly. That means, the people having young age are more likely to have high fraud reports.



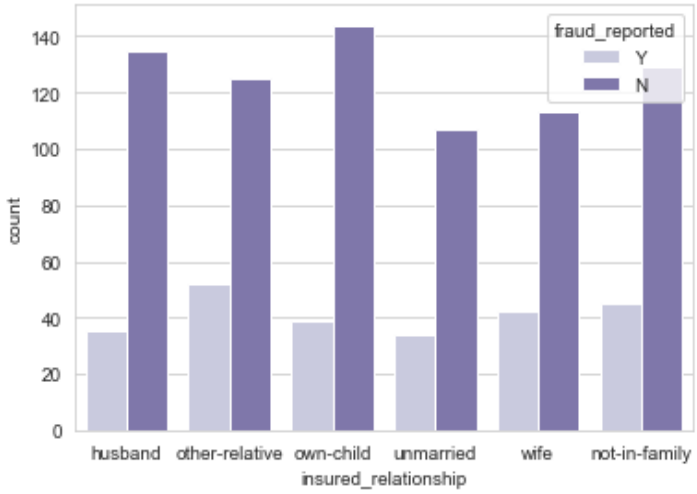
From the above scatter plot, we can observe that in the “policy _ annual _premium” range of 400 to 2000 the “policy _ amount deductible” of 1000, which is the highest count among the policy deductible amounts, has the least amount of “fraud _reported” as Y and a higher number of “fraud _reported” as N compared to other policy deductible amounts.



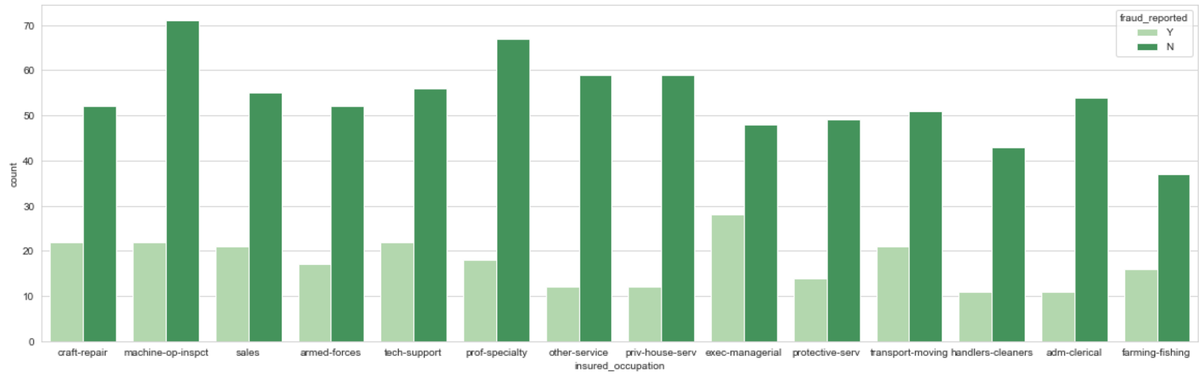
Above is the count plot to compare “insured _sex” and “fraud _reported”. We notice both male and female customers have insurance but the count for females is a bit higher than male counts. The fraud reported data are almost the same in both genders but the non-fraud reports are a bit high in case of female , graph shows that the female customers are more trustworthy than male customers.



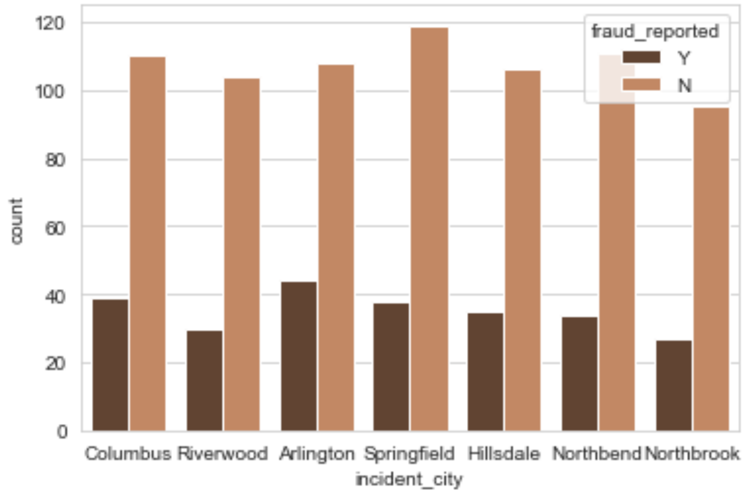
From the above count plot we can observe that the fraud _reported is very less for the people who have high school education and the people who have completed their "JD" education have high fraud _reported among others. That means the people with less education level are more trustworthy.



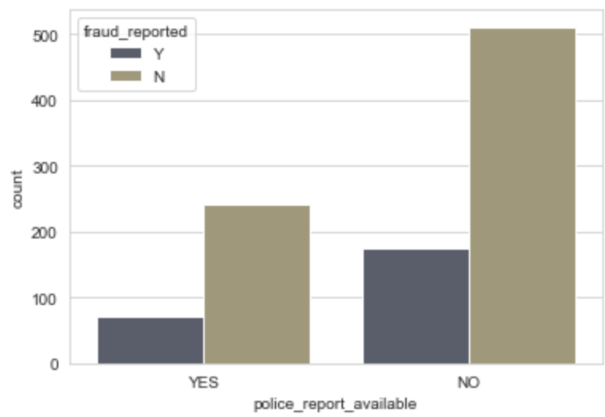
From the above count plot we observe the fraud _reported is very less in people who take insurance for their own-children, followed by people who take insurance for their husband. The fraud _reported is highest in people who take insurance for their ‘other-relative’, followed by insurance taken for ‘not-in-family’. It concludes that insurance taken for own-children, husband & wife are usually trustworthy.



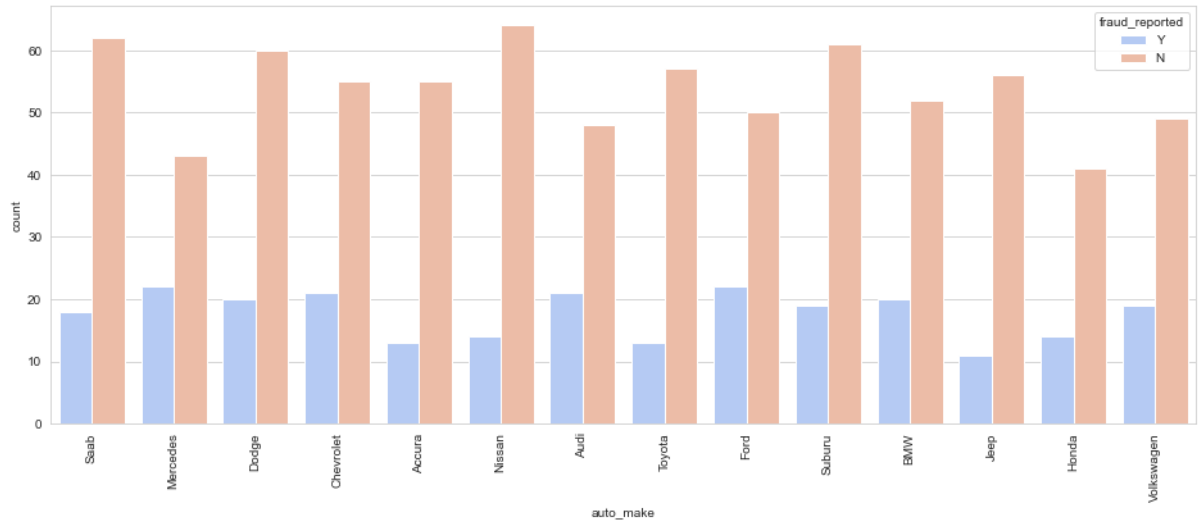
From the above count plot, we observe less fraud _reported if the occupation of the insured is machine operation inspector followed by professional uniqueness. Apart from this all the other insured occupations have almost the same counts. The people whose occupation is exec-managerial have high fraud reports compared to others.



From the above count plot we see the highest number of fraud _reported as no in the city of Springfield, followed by Columbus & North bend. The highest number of fraud _reported as yes is in the city of Arlington.



From the above count plot we see that the police report is not available in most cases when the fraud_ reported is no, compared to when the fraud_ reported is yes. If there are no police reports available then the fraud_ reported is very high.



From the above count plot, the fraud_ reported was the least with the automaker being Nissan, followed by Saab, Subaru & Dodge. The fraud_ reported was high when the automaker was Ford, Mercedes, Audi or BMW.

In a similar manner, we plot the graphs for other columns and make good data visualisation.

EDA CONCLUSION

- We have checked the null values in the dataset and there was no missing values found. (One column "_c39" with only "NaN" values was dropped)
- We have dropped some of the irrelevant columns ("policy_ number", "incident_ location", "umbrella_ limit", "insured_ zip") to overcome the multicollinearity problem.
- Replaced the corrupted entries "?" in the columns with their respective mode values.
- Extracted some new features from the existing features to get better results without any hindrance. And dropped the old columns, if I keep them as it is they will act as duplicates and that leads to a multi collinearity problem.
- Coming to the visualisation part, we have found when and where the fraud reports are high in number.
- To get the better insights about the features, I have used count plots, box plots, pair plots, pie charts, scatter plots and distribution plots.

ENCODING THE DATA FRAME

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

```
In [82]: from sklearn.preprocessing import LabelEncoder
         le=LabelEncoder()
```

```
In [83]: df_icf[categorical_columns]= df_icf[categorical_columns].apply(le.fit_transform)
```

```
In [84]: df_icf
```

Out[84]:

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	insured_sex	insured_education_level	insured_occupation	insured_hobl
0	328	48	2	1000	1406.91	1	4	2	
1	228	42	1	2000	1197.22	1	4	6	
2	134	29	2	2000	1413.14	0	6	11	
3	256	41	0	2000	1415.74	0	6	1	
4	228	44	0	1000	1583.91	1	0	11	
...	...	...	...	...	...	...	...	...	
995	3	38	2	1000	1310.80	0	5	2	
996	285	41	0	1000	1436.79	0	6	9	
997	130	34	2	500	1383.49	0	5	1	
998	458	62	0	2000	1356.92	1	0	5	
999	456	60	2	1000	766.19	0	0	11	

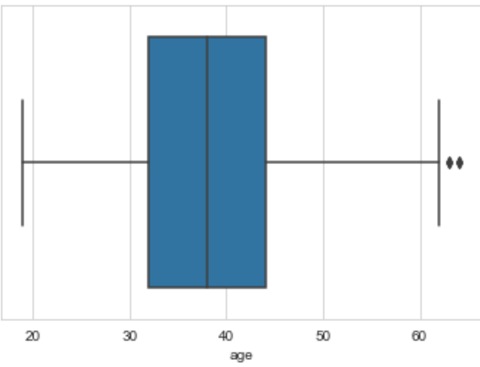
1000 rows × 39 columns

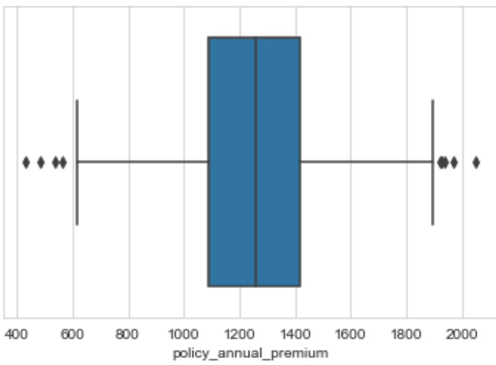
CHECKING OUTLIERS & SKEWNESS

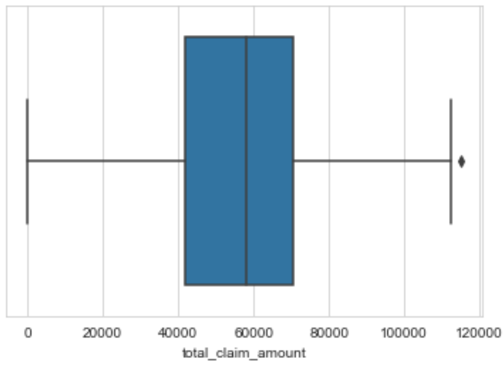
We checked the data for outlier, box plots were used to check each column to find outliers.

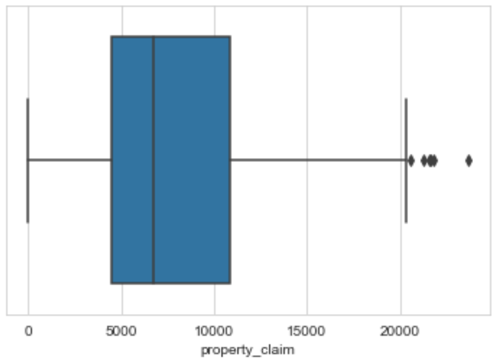
```
In [85]: for i in df_icf.columns:
             sns.boxplot(df_icf[i])
             plt.show()
```

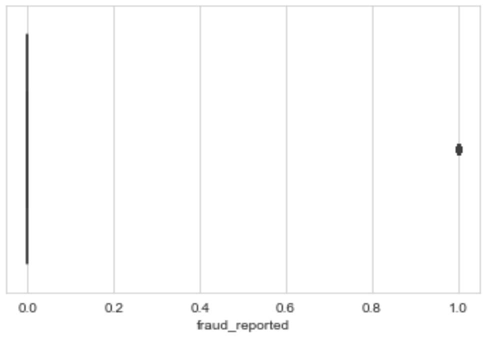
Outliers were found in the following columns:

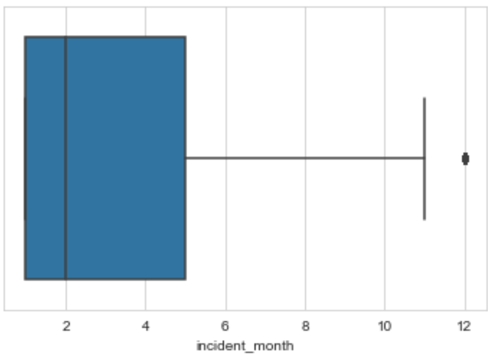












Outliers were found in “age”, “policy _annual_ premium”, “total _claim _amount”, “property _claim”, “fraud _reported” and “incident _Month”.

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

In [87]:
```
df_outliers=df_icf[["age", "policy_annual_premium", "total_claim_amount","property_claim","incident_month","fraud_reported"]]
```

In [88]:
```
from scipy.stats import zscore

z=np.abs(zscore(df_outliers))
df_ICFD=df_icf[(z<3).all(axis=1)]
df_ICFD
```

Out[88]:

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	insured_sex	insured_education_level	insured_occupation	insured_hobl
0	328	48	2	1000	1406.91	1	4	2	
1	228	42	1	2000	1197.22	1	4	6	
2	134	29	2	2000	1413.14	0	6	11	
3	256	41	0	2000	1415.74	0	6	1	
4	228	44	0	1000	1583.91	1	0	11	
...	...	...	...	...	...	...	...	...	
995	3	38	2	1000	1310.80	0	5	2	
996	285	41	0	1000	1436.79	0	6	9	
997	130	34	2	500	1383.49	0	5	1	
998	458	62	0	2000	1356.92	1	0	5	
999	456	60	2	1000	766.19	0	0	11	

996 rows × 39 columns

After removing the outliers our data loss was 0.4 %. Which was affordable.

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

```
In [92]: df_ICFD.skew()

Out[92]: months_as_customer             0.359605
         age                            0.474526
         policy_state                  -0.028155
         policy_deductable              0.473229
         policy_annual_premium          0.032042
         insured_sex                    0.145176
         insured_education_level        0.001349
         insured_occupation            -0.063714
         insured_hobbies               -0.060160
         insured_relationship           0.076423
         capital-gains                  0.478850
         capital-loss                  -0.393015
         incident_type                  0.102917
         collision_type                -0.033826
         incident_severity              0.275635
         authorities_contacted         -0.120741
         incident_state                -0.144616
         incident_city                  0.046459
         incident_hour_of_the_day      -0.039123
         number_of_vehicles_involved    0.500364
         property_damage                0.857547
         bodily_injuries                0.011117
         witnesses                      0.025758
         police_report_available        0.806478
         total_claim_amount            -0.593473
         injury_claim                   0.267970
         property_claim                 0.357130
         vehicle_claim                 -0.619755
         auto_make                     -0.018165
         auto_model                    -0.081747
         fraud_reported                 1.175133
         policy_bind_year               0.058499
         policy_bind_month             -0.029722
         policy_bind_day                0.028923
         incident_month                 1.377097
         incident_day                   0.055659
         csl_per_person                 0.413713
         csl_per_accident               0.609316
         auto_age                       0.049276
         dtype: float64
```


There was skewness in `age, policy deductible ,number _of _vehicles _involved , property _damage, police _report _available, total _ claim _amount , vehicle _claim , csl _per _accident, incident _month.`

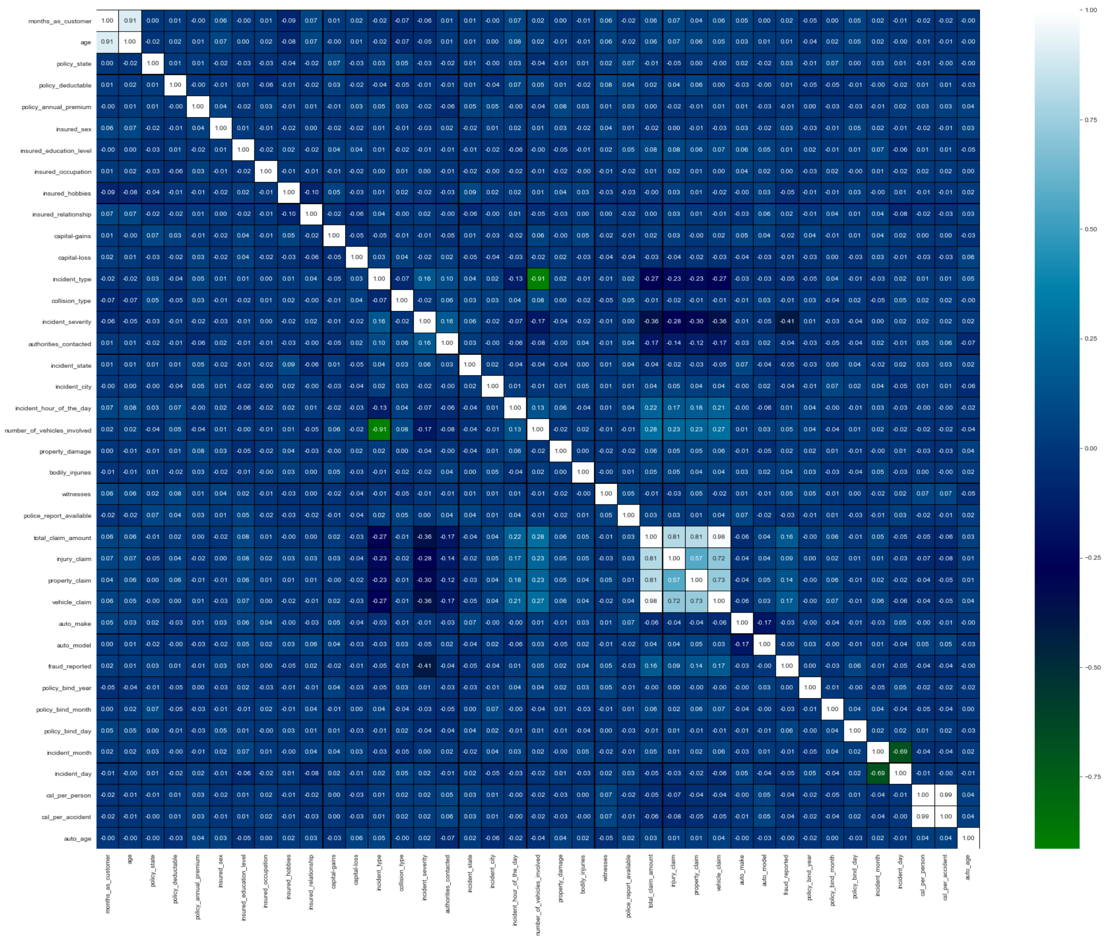
Now,We used the power transformation method (yeo -johnson method) to remove the skewness in the dataset. After using it, the skewness has almost been reduced.

```
In [94]: from sklearn.preprocessing import PowerTransformer
         scal = PowerTransformer(method='yeo-johnson')

In [95]: df_ICFD[df_skew] = scal.fit_transform(df_ICFD[df_skew].values)

In [96]: df_ICFD[df_skew].skew()

Out[96]: age                           -0.002306
         policy_deductable              0.022778
         number_of_vehicles_involved    0.361213
         property_damage                0.857547
         police_report_available        0.806478
         total_claim_amount            -0.508953
         vehicle_claim                 -0.521354
         csl_per_accident               0.110964
         incident_month                 0.305741
         dtype: float64
```

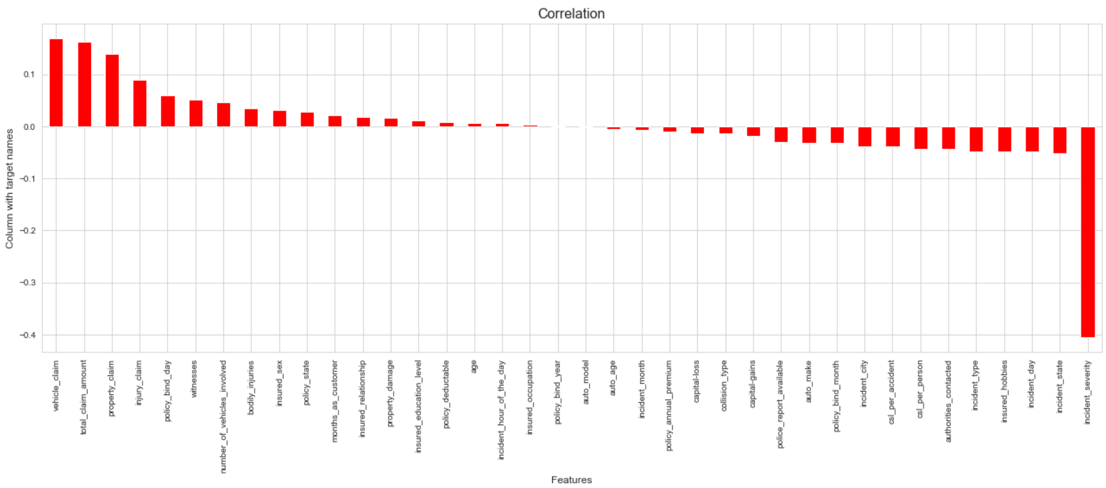
CORRELATION MAP

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

```
In [100]: # Visualizing df_ICFD.corr() using heatmap
          plt.figure(figsize=(30,25))
          sns.heatmap(df_ICFD.corr(),annot=True,linewidths=0.1,linecolor="black",fmt=".2f",cmap="ocean")
```



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



To get the better insights from the heat map we have used bar plots to show the positive and negative correlation between the target variable and other columns.

- ➢ Policy _ bind _year & auto _model are the least correlated with target column.
- ➢ Next, insured _occupation is slightly correlated with the target variable. Auto _age and incident _hour _of _the _day are also less correlated with target.
- ➢ Vehicle _claim, total _claim _amount & property _claim are highly positively correlated with the target.
- ➢ Incident _severity , is highly negatively correlated with the target.

PRE-PROCESSING PIPELINE

First of all , We have to separate the target variable “fraud _reported” and features to process the dataset for model building.

```
In [102]: x = df_ICFD.drop("fraud_reported",axis=1)
          y = df_ICFD["fraud_reported"]

In [103]: x.shape

Out[103]: (996, 38)

In [104]: y.shape

Out[104]: (996,)
```

I have separated independent and dependent features and stored them in x and y respectively.

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

In [105]:
```
from sklearn.preprocessing import StandardScaler

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

Out[105]:

	months_as_customer	age	policy_state	policy_deductable	policy_annua
0	1.074671	1.005252	1.186130	0.064182	
1	0.204846	0.426872	-0.018137	1.269641	
2	-0.612790	-1.143091	1.186130	1.269641	
3	0.448397	0.323178	-1.222403	1.269641	
4	0.204846	0.627644	-1.222403	0.064182	
...	...	...	...	...	
991	-1.752261	-0.002408	1.186130	0.064182	
992	0.700646	0.323178	-1.222403	0.064182	
993	-0.647583	-0.475146	1.186130	-1.212292	
994	2.205443	2.131369	-1.222403	1.269641	
995	2.188047	1.985825	1.186130	0.064182	

996 rows × 38 columns

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

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

In [106]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
vif=pd.DataFrame()
vif["Features"]=x.columns
vif["VIF"]=[variance_inflation_factor(x.values, i) for i in range(x.shape[1])]
vif
```

We got the VIF values as :

	Features	VIF
0	months_as_customer	5.791601
1	age	5.773023
2	policy_state	1.039651
3	policy_deductable	1.044540
4	policy_annual_premium	1.038323
5	insured_sex	1.036090
6	insured_education_level	1.047232
7	insured_occupation	1.018551
8	insured_hobbies	1.054046
9	insured_relationship	1.053933
10	capital-gains	1.038012
11	capital-loss	1.042550
12	incident_type	6.334534
13	collision_type	1.046081
14	incident_severity	1.240243
15	authorities_contacted	1.107264
16	incident_state	1.045420
17	incident_city	1.030590
18	incident_hour_of_the_day	1.103991
19	number_of_vehicles_involved	6.358096
20	property_damage	1.030602
21	bodily_injuries	1.029098
22	witnesses	1.044730
23	police_report_available	1.044838
24	total_claim_amount	43340.904506
25	injury_claim	1597.756113
26	property_claim	1551.116964
27	vehicle_claim	21579.686527
28	auto_make	1.079084
29	auto_model	1.066430
30	policy_bind_year	1.028266
31	policy_bind_month	1.038789
32	policy_bind_day	1.025451
33	incident_month	1.980684
34	incident_day	1.971631
35	csl_per_person	52.817175
36	csl_per_accident	52.954440
37	auto_age	1.040193

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

The acceptable range of VIF is below 10. We observed the highest VIF in total _claim_ amount, so we dropped this column first and again checked the VIF to confirm whether the multicollinearity issue was solved or not. Again, we found a high VIF in the csl _per _accident column. So, we dropped that column too. After removing 2 columns our multicollinearity got solved by giving VIF values below 10 in all the columns.

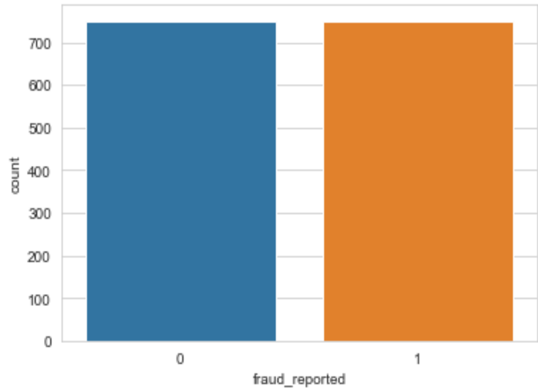
OVER SAMPLING :

Now, since we have come across the data imbalance issue, we need to fix it by either oversampling or under-sampling the data. Oversampling is preferred, because under-sampling causes a huge data loss.

Oversampling was done as follows:

```
In [111]: y.value_counts()

Out[111]: 0    750
          1    246
          Name: fraud_reported, dtype: int64

In [112]: from imblearn.over_sampling import SMOTE
          sm = SMOTE()
          x , y = sm.fit_resample(x,y)

In [113]: y.value_counts()

Out[113]: 0    750
          1    750
          Name: fraud_reported, dtype: int64

In [114]: # Visualizing the data after oversampling
          sns.countplot(y)
```

The data is now balanced that we can observe in the count plot



BUILDING MACHINE LEARNING MODELS

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

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

FINDING THE BEST RANDOM STATE & ACCURACY

```
In [116]: maxAccu=0
maxRS=0
for i in range(200):
    x_train,x_test,y_train,y_test=train_test_split(x, y, test_size = 0.30, random_state = i)
    lr=LogisticRegression()
    lr.fit(x_train,y_train)
    predrs=lr.predict(x_test)
    acc=accuracy_score(y_test,predrs)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is :",maxAccu," on Random State :",maxRS)

Best accuracy is : 0.7977777777777778  on Random State : 93
```

- We have got the best random state as 93 and best accuracy as 79.77% using the Logistic Regression model. Now let's create new train sets and test sets and fit them into the models to find our ideal model.

```
In [117]: # dividing the dataset for training and testing with best random state
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x, y, test_size=.30, random_state=maxRS)
```

CLASSIFICATION ALGORITHMS :

We have used 9 different classification algorithms for our predictions, they are : Logistic Regression Model, Decision Tree Classifier, Gaussian NB Classifier, Gradient Boosting Classifier, K-Nearest Neighbors Classifier, SVC Model, Random Forest Classifier, XG Boost Classifier & Extra Trees Classifier.

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

- LOGISTIC REGRESSION MODEL :

```
In [124]: lg=LogisticRegression()
lg.fit(x_train, y_train)
lg.score(x_train, y_train)
pred_lg=lg.predict(x_test)

print("accuracy score: ",accuracy_score(y_test,pred_lg))
print(confusion_matrix(y_test,pred_lg))
print(classification_report(y_test,pred_lg))

accuracy score:  0.7955555555555556
[[170  41]
 [ 51 188]]
              precision    recall  f1-score   support

           0       0.77      0.81      0.79       211
           1       0.82      0.79      0.80       239

    accuracy                           0.80       450
   macro avg       0.80      0.80      0.80       450
weighted avg       0.80      0.80      0.80       450
```

- The Logistic Regression Model gave us an accuracy score of 79.55 %.

- **DECISION TREE CLASSIFIER :**

```
In [126]: dtc=DecisionTreeClassifier()
          dtc.fit(x_train,y_train)
          dtc.score(x_train,y_train)
          pred_dtc=dtc.predict(x_test)

          print("accuracy score: ",accuracy_score(y_test,pred_dtc))
          print(confusion_matrix(y_test,pred_dtc))
          print(classification_report(y_test,pred_dtc))
```

```
accuracy score:  0.7377777777777778
[[106 105]
 [ 13 226]]
              precision    recall  f1-score   support

           0       0.89      0.50      0.64       211
           1       0.68      0.95      0.79       239

    accuracy                           0.74       450
   macro avg       0.79      0.72      0.72       450
weighted avg       0.78      0.74      0.72       450
```

- The Decision Tree Classifier Model gave us an accuracy score of 73.77 %.

- **GAUSSIAN NB CLASSIFIER :**

```
In [128]: gnb=GaussianNB()
          gnb.fit(x_train,y_train)
          gnb.score(x_train,y_train)
          pred_gnb=gnb.predict(x_test)

          print("accuracy score: ",accuracy_score(y_test,pred_gnb))
          print(confusion_matrix(y_test,pred_gnb))
          print(classification_report(y_test,pred_gnb))
```

```
accuracy score:  0.8066666666666666
[[163  48]
 [ 39 200]]
              precision    recall  f1-score   support

           0       0.81      0.77      0.79       211
           1       0.81      0.84      0.82       239

    accuracy                           0.81       450
   macro avg       0.81      0.80      0.81       450
weighted avg       0.81      0.81      0.81       450
```


- The Gaussian NB Classifier Model gave us an accuracy score of 80.66 %.

- **GRADIENT BOOSTING CLASSIFIER :**

```
In [130]: gbc=GradientBoostingClassifier()
          gbc.fit(x_train,y_train)
          gbc.score(x_train,y_train)
          pred_gcb=gbc.predict(x_test)

          print("accuracy score: ",accuracy_score(y_test,pred_gcb))
          print(confusion_matrix(y_test,pred_gcb))
          print(classification_report(y_test,pred_gcb))
```

```
accuracy score:  0.7555555555555555
[[108 103]
 [  7 232]]
              precision    recall  f1-score   support

           0       0.94      0.51      0.66       211
           1       0.69      0.97      0.81       239

    accuracy                           0.76       450
   macro avg       0.82      0.74      0.74       450
weighted avg       0.81      0.76      0.74       450
```

- The Gradient Boosting Classifier Model gave us an accuracy score of 75.55 %.

- K NEAREST NEIGHBORS CLASSIFIER :

```
In [132]: knn=KNeighborsClassifier()
          knn.fit(x_train,y_train)
          knn.score(x_train,y_train)
          pred_knn=knn.predict(x_test)

          print("accuracy score: ",accuracy_score(y_test,pred_knn))
          print(confusion_matrix(y_test,pred_knn))
          print(classification_report(y_test,pred_knn))
```

```
accuracy score:  0.6911111111111111
[[ 81 130]
 [  9 230]]
              precision    recall  f1-score   support

           0       0.90      0.38      0.54       211
           1       0.64      0.96      0.77       239

    accuracy                           0.69       450
   macro avg       0.77      0.67      0.65       450
weighted avg       0.76      0.69      0.66       450
```

- The K Nearest Neighbors Classifier Model gave us an accuracy score of 69.11 %.

- **SVC(support vector classifier) MODEL:**

```
In [134]: svc = SVC()
svc.fit(x_train, y_train)
svc.score(x_train, y_train)
svc_pred = svc.predict(x_test)
print("accuracy score: ",accuracy_score(y_test,svc_pred))
print(confusion_matrix(y_test,svc_pred))
print(classification_report(y_test,svc_pred))
```

```
accuracy score:  0.8888888888888888
[[189  22]
 [ 28 211]]
              precision    recall  f1-score   support

           0       0.87      0.90      0.88       211
           1       0.91      0.88      0.89       239

    accuracy                           0.89       450
   macro avg       0.89      0.89      0.89       450
weighted avg       0.89      0.89      0.89       450
```

❖ The SVC Model gave us an accuracy score of 88.88 %.

❖ RANDOM FOREST CLASSIFIER :

```
In [136]: rfc=RandomForestClassifier()
rfc.fit(x_train,y_train)
rfc.score(x_train,y_train)
pred_rfc=rfc.predict(x_test)

print("accuracy score: ",accuracy_score(y_test,pred_rfc))
print(confusion_matrix(y_test,pred_rfc))
print(classification_report(y_test,pred_rfc))
```



```
accuracy score:  0.9133333333333333
[[182  29]
 [ 10 229]]
              precision    recall  f1-score   support

           0       0.95      0.86      0.90       211
           1       0.89      0.96      0.92       239

    accuracy                           0.91       450
   macro avg       0.92      0.91      0.91       450
weighted avg       0.92      0.91      0.91       450
```

❖ The Random Forest Classifier Model gave us an accuracy score of 91.33 %.

❖ XGBOOST CLASSIFIER :

```
In [138]: xgb=XGBClassifier()
          xgb.fit(x_train,y_train)
          xgb.score(x_train,y_train)
          pred_xgb=xgb.predict(x_test)

          print("accuracy score: ",accuracy_score(y_test,pred_xgb))
          print(confusion_matrix(y_test,pred_xgb))
          print(classification_report(y_test,pred_xgb))

accuracy score:  0.8622222222222222
[[156  55]
 [  7 232]]
              precision    recall  f1-score   support

           0       0.96      0.74      0.83       211
           1       0.81      0.97      0.88       239

    accuracy                           0.86       450
   macro avg       0.88      0.86      0.86       450
weighted avg       0.88      0.86      0.86       450
```

➢ The XG Boost Classifier Model gave us an accuracy score of 86.22 %.

❖ **EXTRA TREES CLASSIFIER :**

```
In [140]: etc=ExtraTreesClassifier()
          etc.fit(x_train,y_train)
          etc.score(x_train,y_train)

          pred_etc=etc.predict(x_test)
          print("accuracy score: ",accuracy_score(y_test,pred_etc))
          print(confusion_matrix(y_test,pred_etc))
          print(classification_report(y_test,pred_etc))

accuracy score:  0.9177777777777778
[[187  24]
 [ 13 226]]
              precision    recall  f1-score   support

           0       0.94      0.89      0.91       211
           1       0.90      0.95      0.92       239

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


➢ The Extra Trees Classifier gave us an accuracy score of 91.77 %.

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

,And then Gaussian NB Classifier, Logistic Regression Model, Decision Tree Classifier and Gradient Boosting Classifier.

The lowest Accuracy score belongs to K Nearest Neighbors Classifier.

CROSS VALIDATION SCORES:

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

```
In [142]: scr_lg=cross_val_score(lg,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_lg.mean())

          Cross validation score of this model is:  0.738
```

➢ The Cross Validation Score of the Logistic Regression Model is 73.8 %.

```
In [143]: scr_dtc=cross_val_score(dtc,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_dtc.mean())

          Cross validation score of this model is:  0.8373333333333335
```

➢ The Cross Validation Score of the Decision Tree Classifier Model is 83.73 %.

```
In [144]: scr_gnb=cross_val_score(gnb,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_gnb.mean())

          Cross validation score of this model is:  0.736
```

➢ The Cross Validation Score of the GaussianNB Classifier Model is 73.6 %.


```
In [145]: scr_gbc=cross_val_score(gbc,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_gbc.mean())

          Cross validation score of this model is:  0.8800000000000001
```

➢ The Cross Validation Score of the Gradient Boosting Classifier Model is 88.00 %.

```
In [146]: scr_knn=cross_val_score(knn,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_knn.mean())

          Cross validation score of this model is:  0.674
```

➢ The Cross Validation Score of the K Nearest Neighbors Classifier Model is 67.4 %.

```
In [147]: scr_svc=cross_val_score(svc,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_svc.mean())

          Cross validation score of this model is:  0.8620000000000001
```

➢ The Cross Validation Score of the SVC Model is 86.20 %.

```
In [148]: scr_rfc=cross_val_score(rfc,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_rfc.mean())

          Cross validation score of this model is:  0.8826666666666666
```

➢ The Cross Validation Score of the Random Forest Classifier Model is 88.26 %.

```
In [149]: scr_xgb=cross_val_score(xgb,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_xgb.mean())

          Cross validation score of this model is:  0.8886666666666667
```

➢ The Cross Validation Score of the XG Boost Classifier Model is 88.86 %.


```
In [150]: scr_etc=cross_val_score(etc,x,y,cv=5)
          print("Cross validation score of this model is: ",scr_etc.mean())

          Cross validation score of this model is:  0.9106666666666667
```

➢ The Cross Validation Score of the Extra Trees Classifier Model is 91.06 %.

❖ The highest Cross validation Score belongs to Extra Trees Classifier, followed by XG Boost Classifier, Random Forest Classifier, Gradient Boosting Classifier & SVC(support vector classifier) model.

Followed by Decision Tree Classifier, Logistic Regression model, Gaussian NB Classifier . and lastly, K Nearest Neighbors Classifier.

❖ HYPER PARAMETER TUNING :

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

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

```
In [151]: from sklearn.model_selection import GridSearchCV
```

```
In [156]: parameters = {'criterion' : ['gini','entropy'],
                        'random_state' : [10, 50, 1000],
                        'max_depth' : [0, 10, 20],
                        'n_jobs' : [-2, -1, 1],
                        'n_estimators' : [50, 100, 200, 300]}
          grid_etc=GridSearchCV(etc, param_grid = parameters, cv = 8)
```

➢ By using the above parameters, we are tuning the best model (Extra Trees Classifier) and after tuning we have to choose the best parameters from the above list.

```
In [157]: grid_etc.fit(x_train, y_train)

Out[157]: GridSearchCV(cv=8, estimator=ExtraTreesClassifier(),
                       param_grid={'criterion': ['gini', 'entropy'],
                                   'max_depth': [0, 10, 20],
                                   'n_estimators': [50, 100, 200, 300],
                                   'n_jobs': [-2, -1, 1],
                                   'random_state': [10, 50, 1000]})
```

```
In [158]: grid_etc.best_params_

Out[158]: {'criterion': 'entropy',
           'max_depth': 20,
           'n_estimators': 100,
           'n_jobs': -2,
           'random_state': 1000}
```

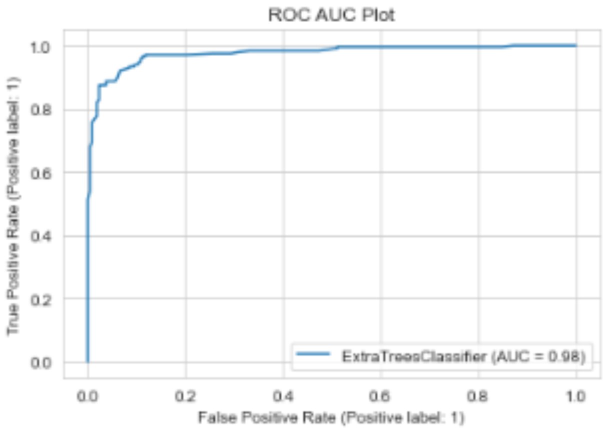

➢ These were found to be the best parameters after tuning, now let us use these parameters to improve our model.

```
In [159]: etc1=ExtraTreesClassifier(criterion='entropy',random_state=1000,max_depth=20,n_jobs=-2,n_estimators=100)

          etc1.fit(x_train,y_train)
          pred=etc1.predict(x_test)
          print("accuracy score: ",accuracy_score(y_test,pred))
          print(confusion_matrix(y_test,pred))
          print(classification_report(y_test,pred))

          accuracy score:  0.9244444444444444
          [[188  23]
           [ 11 228]]
                        precision    recall  f1-score   support

                     0       0.94      0.89      0.92       211
                     1       0.91      0.95      0.93       239

              accuracy                           0.92       450
             macro avg       0.93      0.92      0.92       450
          weighted avg       0.93      0.92      0.92       450
```

The model after hyper parameter tuning has an improved accuracy score of 92.44 %.

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

```
In [161]: from sklearn.metrics import plot_roc_curve
          plot_roc_curve(etc1,x_test,y_test)
          plt.title("ROC AUC Plot")
          plt.show()
```



We have plotted the ROC-AUC curve, AUC score is 98 %.

❖ SAVING THE MODEL :

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

```
In [162]: import joblib
          joblib.dump(etc1,"Insurance_Claim_Fraud_Detection.pkl")

Out[162]: ['Insurance_Claim_Fraud_Detection.pkl']
```

❖ PREDICTION:

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

```
In [163]: # Loading the saved model
          fraud_detection_model=joblib.load("Insurance_Claim_Fraud_Detection.pkl")

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

```
Out[163]: array([1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
                 0, 1, 0, 1, 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, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0,
                 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1,
                 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 1,
                 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1,
                 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 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, 1, 1, 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, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0,
                 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 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, 1, 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, 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, 0, 1, 0, 1, 1, 1, 0,
                 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0,
                 1, 1, 0, 1, 1, 1, 0, 0, 0, 1])
```

❖ Comparing the Predicted Values & the Actual Values.

```
In [164]: pd.DataFrame([fraud_detection_model.predict(x_test)[:],y_test[:]],index=["Predicted Value","Actual Value"])
```

Out[164]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Predicted Value	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1
Actual Value	1	1	1	1	1	1	1	1	1	0	1	1	1	0	1	1	1	1	1	1

	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
Predicted Value	1	1	0	1	0	1	1	0	1	1	0	1	1	0	1	1	0	1	0
Actual Value	1	1	0	1	0	0	1	0	1	1	0	1	1	0	1	1	0	1	0

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

CONCLUDING REMARKS

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

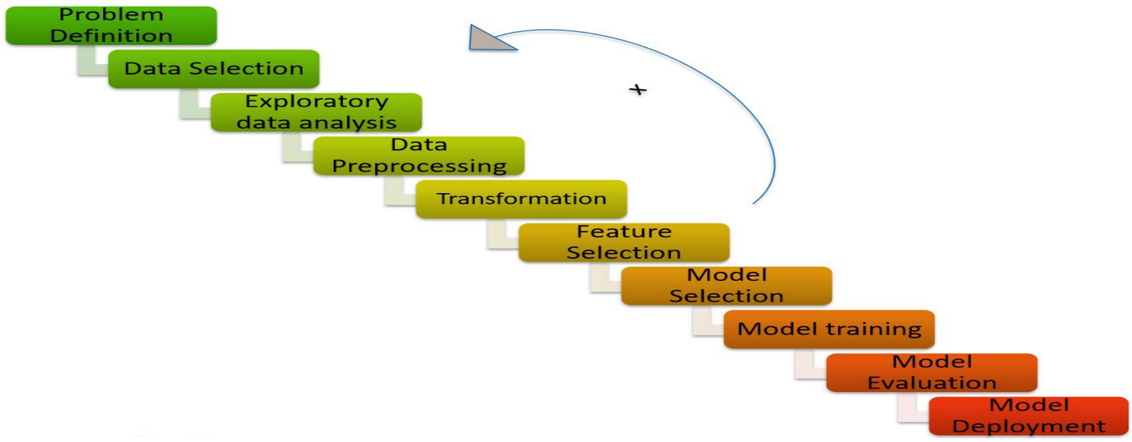
We did some Data Visualisation using count plots, scatter plots, bar plots & dist plots. After visualisation we encoded the data frame using Label Encoder. Next we checked for outliers present in the data and removed them using the z-score method. We checked the skewness of our data and reduced it for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

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

With the help of above techniques, our model is able to predict the fraudulent report with the accuracy of 92.44%. Also, we have seen that the actual and predicted values are almost the same, which means our model worked correctly.

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

Building machine learning models for such problems can help the insurance companies to choose the correct insurer. So, Machine learning techniques are very useful to solve these kinds of problems. This project has built a model that can detect auto insurance fraud. In doing so, the model can reduces loses for insurance companies. The challenge behind fraud detection in machine learning is that frauds are far less common as compared to legit insurance claims.



About the Author:

Pursuing Post Graduation Diploma in Data Science, Machine Learning and Neural Network from Datatrained . I am an aspiring Data Scientist whose purpose is to learn in detail all the concepts needed for Data Science. I am passionate about Data Science and have skills that help me derive valuable insights from data, such as Data Manipulation, Data Visualisation, Data Analysis, EDA, and Machine Learning.

Hardware & Software Requirements & Tools Used:

Hardware required:

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

Software requirement:

- Jupiter Notebook

Libraries Used:

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

DataTrained

Keep Skilling, Keep Growing