Detecting Insurance Fraud with Machine Learning

Insurance Fraud has been around since the beginning of insurance organizations. These are unnoticed crimes which cost the insurance industry billions a year. Insurance fraud is a grave problem with all insurance service providers. Even a small fraud can be extremely expensive. As a result, businesses must be careful while analyzing every claim during the insurance claims settlement process. And while it is not possible to verify manually every single claim, so the claim settlement process in insurance can be made easier through Artificial Intelligence.

AI-based Investigation: We will train the machine to identify whether the case is fraud or not. Many past data have been provided to Machine Learning on which basis AI helps to find the pattern and decide upon the case.

Machine Learning: A Big Step in Fraud Detection

Machine learning is a part of Artificial Intelligence (AI). The idea behind AI is to create a computerized system that can engage in complex analysis and not only replace human input but improve upon it.

Machine learning gives systems the ability to learn and improve from experience, with no extra programming. To do this, the system analyzes large, labelled datasets. Labelled dataset means, with some input, features it also concludes output features. By comparing those inputs to outputs ML finds the pattern and do more complex analysis.

This article contains the following sub-topics

1. Problem Definition.
2. Data Analysis.
3. EDA Concluding Remark.
4. Pre-Processing Pipeline.
5. Building Machine Learning Models.
6. Concluding Remarks.

Problem Definition:

The goal of this project is to build a model that can detect auto insurance fraud. The challenge behind fraud detection in machine learning is that frauds are less common as compared to legit insurance claims. However, frauds are unethical and are a loss to the company.

Building a model that can classify auto insurance fraud which can help to cut losses for the insurance company.

Data Analysis:

About the dataset

- Data Source: https://raw.githubusercontent.com/FlipRoboTechnologies/ML_-Datasets/main/Insurance%20Claim%20Fraud%20Detection/Automobile_insurance_fraud.csv

```
# Import Required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Load the dataset to our environment for analysis. In this case, we are using, Jupyter Notebook.

df = pd.read_csv('https://raw.githubusercontent.com/FlipRoboTechnologies/ML_-Datasets/main/Insurance%20Claim%20Fraud%20Detection/Automobile_insurance_fraud.csv')

The inspiration for this project was to perform classification on imbalance class data sets, in particular fraud. (Imbalanced class datasets where one type of data (target) is much more than the other)

The current data set was labelled with n=1000 records.

```
print(" Dataset have \n Rows= ",df.shape[0],'\n Columns= ',df.shape[1])
```

```
Dataset have
Rows=  1000
Columns=  40
```

And 38 different features on which basis it decided about the 39th feature (Target feature). This dataset was found on kaggle.com, however, it is not mentioned if this data is from multiple insurance companies or just one company.

Explore the features on which basis Machine was able to predict about the label.

We have 17 integer features, 21 Object features, 1 float feature

Integer features are those who has values in integer like 1, 2, 3,….

Float features are those who has values in decimal like 1.2, 2.3,…

Object features are categorical in nature or in string format like Yes, No

df.head(12)

328	48	5215 85	17-10-2014	OH	250/500	1000	1406.91	0	466132	...	2	YES.1	7161 0	6510	13020	52080	Saab	92x	2004	Y
0	228	42	342868	27-06-20	IN	250/500	2000	1197.22	5000000	468176	...	0	?	5070	780	780	3510	Mercedes	E400	2007	Y

328	**48**	**521585**	**17-10-2014**	**OH**	**250/500**	**1000**	**1406.91**	**0**	**4661632**	**...**	**2**	**YES.1**	**71610**	**6510**	**130220**	**520880**	**Saab**	**92x**	**2004**	**Y**	
				06																	
1	134	29	687698	06-09-20000	OH	100/300	2000	1413.14	500000000	4306632	...	3	NO	34650	77000	38500	2310100	Dodge	RAM	2007	N
2	256	41	227811	25-05-1990	IL	250/500	2000	1415.74	600000000	6081117	...	2	NO	63400	63400	63400	507220	Chevrolet	Tahoe	2014	Y
3	228	44	367455	06-06-20014	IL	500/10000	1000	1583.91	600000000	6107706	...	1	NO	6500	13000	650	4550	Acura	RSX	2009	N
4	256	39	1045 94	12-10-20006	OH	250/500	1000	1351.10	0	4784 56	...	2	NO	64100	64100	64100	512800	Saab	95	2003	Y
5	137	34	413978	04-06-	IN	250/500	1000	1333.35	0	4417 16	...	0	?	78650	21450	7150	500500	Nissan	Pathfinder	2012	N

328	**48**	**521585**	**17-10-2014**	**OH**	**250/500**	**1000**	**1406.91**	**0**	**4661132**	**...**	**2**	**YES.1**	**71610**	**6510**	**130220**	**520800**	**Saab**	**92x**	**2004**	**Y**		
				2000																		
6	165	37	429027	03-02-1990	IL	100/300	1000	1137.03	0	603195	...	2	YES	51590	93800	93800	328330	Audi	A5	2015	N	
7	27	33	485665	05-02-1997	IL	100/300	500	1442.99	0	601734	...	1	YES	27700	27700	27700	221160	Toyota	Camry	2012	N	
8	212	42	636550	25-07-2011	IL	100/300	500	1315.68	0	600983	...	1	?	42300	47000	47000	329000	Saab	92x	1996	N	
9	235	42	543610	26-05-2002	OH	100/300	500	1253.12	40000000	462283	...	2	?	87010	7910	158200	632800	Ford	F150	2002	N	
10	447	61	214618	29-0	OH	100/300	2000	1137.16	0	615561	...	2	YES	11492 0	176	176	795	Audi	A3	2006	N	

328	**48**	**521585**	**17-10-2014**	**OH**	**250/500**	**1000**	**1406.91**	**0**	**4661632**	**...**	**2**	**YES.1**	**71610**	**6510**	**13020**	**52080**	**Saab**	**92x**	**2004**	**Y**		
				5-1999											80	80	60					
11	60	23	842643	20-11-1997	OH	500/1000	500	1215.36	30000000	432220	...	0	NO	56520	4710	9420	423900	Saab	95	2000	N	

12 rows × 39 columns

In [5]:

df.tail(13)

Out[5]:

	328	**48**	**521585**	**17-10-2014**	**OH**	**250/500**	**1000**	**1406.91**	**0**	**4661632**	**...**	**2**	**YES.1**	**71610**	**6510**	**13020**	**52080**	**Saab**	**92x**	**2004**	**Y**	
986	235	39	836349	01-05-2013	IL	500/1000	2000	1453.61	4000000	6195770	...	3	?	60320	9280	9280	41760	Chevrolet	Tahoe	2012	Y	
987	295	46	591269	09-01-1999	IN	100/300	500	1672.88	0	4789047	...	1	NO	60700	12140	6070	42490	Honda	Civic	1997	N	

	328	**48**	**521585**	**17-10-2014**	**OH**	**250/500**	**1000**	**1406.91**	**0**	**466132**	**...**	**2**	**YES.1**	**71610**	**6510**	**13020**	**52080**	**Saab**	**92x**	**2004**	**Y**
988	22	21	550127	04-07-2007	IN	250/500	1000	1248.05	0	443550	...	2	?	53280	5920	0	47360	Chevrolet	Malibu	2015	N
989	286	43	663190	05-02-1994	IL	100/300	500	1564.43	3000000	477644	...	2	YES	34290	3810	3810	26670	Jeep	Grand Cherokee	2013	N
990	257	44	109392	12-07-2006	OH	100/300	1000	1280.88	0	433981	...	1	NO	46980	0	5220	41760	Accura	TL	2002	N
991	94	26	215278	24-10-2007	IN	100/300	500	722.66	0	433696	...	2	YES	36700	3670	7340	25690	Nissan	Pathfinder	2010	N
992	124	28	674570	08-12-2001	OH	250/500	1000	1235.14	0	443567	...	1	?	60200	6020	6020	48160	Volkswagen	Passat	2012	N
993	141	30	681486	24-03-20	IN	500/1000	1000	1347.04	0	430665	...	2	YES	6480	540	1080	4860	Honda	Civic	1996	N

	328	**48**	**521585**	**17-10-20147**	**OH**	**250/500**	**1000**	**1406.91**	**0**	**461632**	**...**	**2**	**YES.1**	**71610**	**6510**	**13020**	**52080**	**Saab**	**92x**	**2004**	**Y**
994	3	38	941851	16-07-19911	OH	500/1000	1000	1310.80	0	431289	...	1	?	87200	17440	8720	61040	Honda	Accord	2006	N
995	285	41	186934	05-01-20014	IL	100/300	1000	1436.79	0	608177	...	3	?	108480	18080	18080	72320	Volkswagen	Passat	2015	N
996	130	34	918516	17-02-20003	OH	250/500	500	1383.49	3000000	442797	...	3	YES	67500	7500	7500	52500	Suburu	Impreza	1996	N
997	458	62	533940	18-11-20011	IL	500/1000	2000	1356.92	5000000	441714	...	1	YES	46980	5220	5220	36540	Audi	A5	1998	N
998	456	60	556080	11-11-11996	OH	250/500	1000	766.19	0	612260	...	3	?	5060	460	920	3680	Mercedes	E400	2007	N

13 rows × 39 columns

In our dataset, we have 53.8% Categorical features and 46.2% are continuous features.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 39 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           1000 non-null   int64
 1   age                          1000 non-null   int64
 2   policy_number                1000 non-null   int64
 3   policy_bind_date             1000 non-null   object
 4   policy_state                 1000 non-null   object
 5   policy_csl                   1000 non-null   object
 6   policy_deductable            1000 non-null   int64
 7   policy_annual_premium        1000 non-null   float64
 8   umbrella_limit               1000 non-null   int64
 9   insured_zip                  1000 non-null   int64
 10  insured_sex                  1000 non-null   object
 11  insured_education_level      1000 non-null   object
 12  insured_occupation           1000 non-null   object
 13  insured_hobbies              1000 non-null   object
 14  insured_relationship         1000 non-null   object
 15  capital-gains                1000 non-null   int64
 16  capital-loss                 1000 non-null   int64
 17  incident_date                1000 non-null   object
 18  incident_type                1000 non-null   object
 19  collision_type               1000 non-null   object
 20  incident_severity            1000 non-null   object
 21  authorities_contacted        1000 non-null   object
 22  incident_state               1000 non-null   object
 23  incident_city                1000 non-null   object
 24  incident_location            1000 non-null   object
 25  incident_hour_of_the_day     1000 non-null   int64
 26  number_of_vehicles_involved  1000 non-null   int64
 27  property_damage              1000 non-null   object
 28  bodily_injuries              1000 non-null   int64
 29  witnesses                    1000 non-null   int64
 30  police_report_available      1000 non-null   object
 31  total_claim_amount           1000 non-null   int64
 32  injury_claim                 1000 non-null   int64
 33  property_claim               1000 non-null   int64
 34  vehicle_claim                1000 non-null   int64
 35  auto_make                    1000 non-null   object
 36  auto_model                   1000 non-null   object
 37  auto_year                    1000 non-null   int64
 38  fraud_reported               1000 non-null   object
dtypes: float64(1), int64(17), object(21)
memory usage: 304.8+ KB
```

First of all, to know the authenticity of our dataset, we check for the Null values (empty entry)

Missing Values

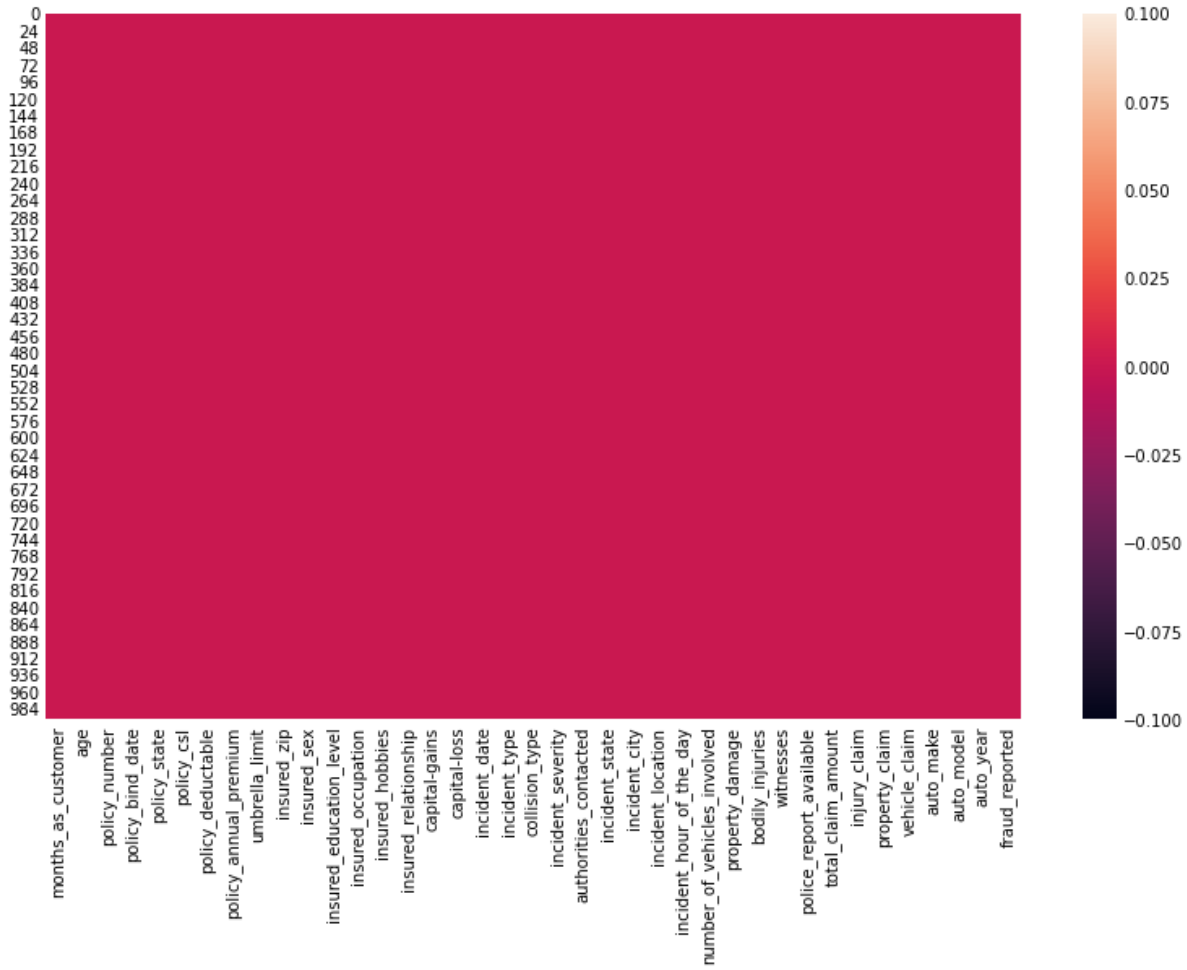
```
df.isnull().sum()
```

```
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
auto_make                     0
auto_model                    0
auto_year                     0
fraud_reported                0
_c39                       1000
dtype: int64
```

We can also visualize any null values

```
plt.figure(figsize=(12,8))
sns.heatmap(df.isnull())
```

```
<AxesSubplot:>
```



We don’t have any Null values in our dataset.Upon further investigation, we come to know that this dataset have ‘?’ instead of NaN values.

```
df['collision_type'].value_counts(dropna=False)
```

```
Rear Collision     292
Side Collision     276
Front Collision    254
?                  178
Name: collision_type, dtype: int64
```

Check every Object type feature and replace '?' into Nan values so that can be handled easily.

```
df['collision_type'].replace('?',np.nan,inplace=True)
```

```
df['property_damage'].unique()
```

```
array(['YES', '?', 'NO'], dtype=object)
```

```
df['property_damage'].replace('?',np.nan, inplace=True)
```

we had to impute those Null values with the meaningful entry by various methods like

Mean, Median, Mode replacement

Random Sample imputation

Capturing NaN values with a new feature

End of distribution imputation

Arbitrary Imputation

Frequent categories imputation

EDA Concluding Remark.

Check my GitHub for more detailed EDA

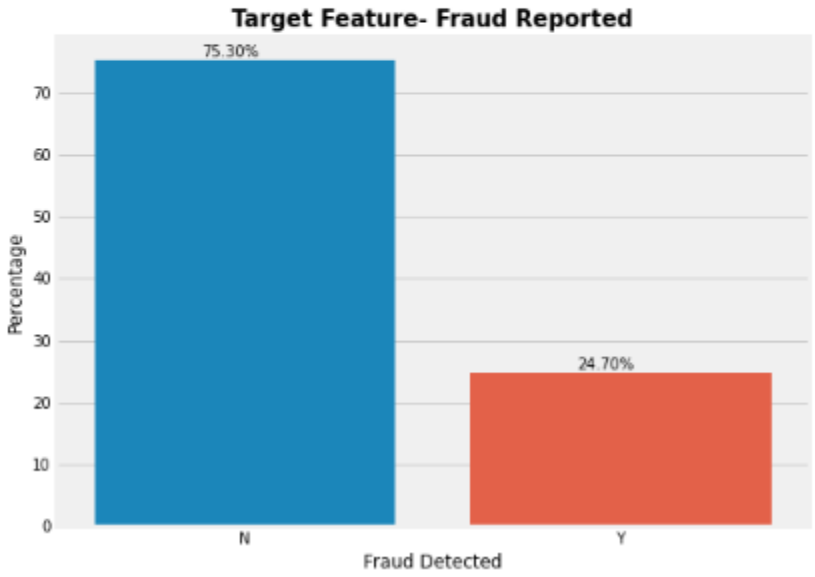
ABHIBRO123/EVALUATION-PR-3: PHASE -3 PROJECT SOLUTION (github.com)

Dependent Variable (Target Feature)

EDA started with the dependent variable, fraud_reported. There were 247 frauds detected and 753 were genuine cases. In percentage, 24.7% of the data were with fraud details while 75.3% were with genuine case

```
plt.figure(figsize=(8,6))
plt.title("Target Feature- Fraud Reported",fontdict={'fontweight':'bold','fontsize':15})
ax=sns.barplot(x=target_df.index,y=target_df.values)
plt.xlabel('Fraud Detected')
plt.ylabel('Percentage')

for p in ax.patches:
    height=p.get_height()
    width=p.get_width()
    x,_=p.get_xy()
    ax.text(x+width/2.8,height+0.5,f'{height:.2f}%')
```



Correlation is the quantitative method to find the relationship between independent features with Target feature as well within an independent feature. If you are a beginner, let me revise that all the input features are also known as independent features and the target feature is also known as dependent features as it depends on the input variables.

1. Month_as_customer and age had.92 correlation
2. Total_claim_amount has good correlation with other claim features

Total_claim_amount – injury_claim =.81

Total_claim_amount – property_claim = .81

Total_claim_amount – vehicle_claim = .98

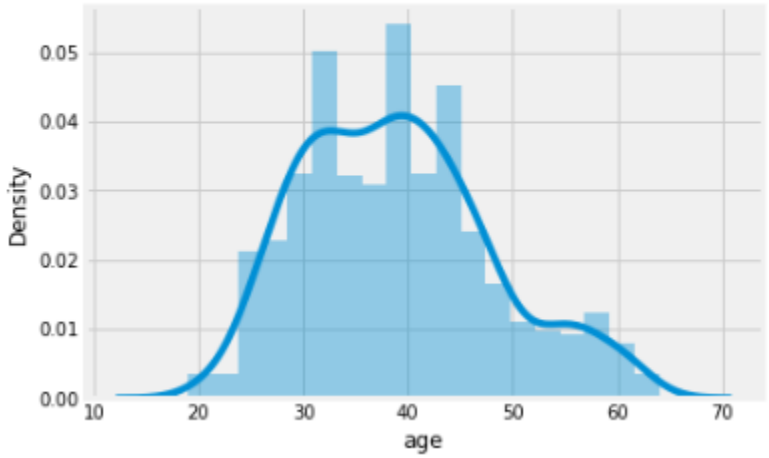
EDA on Independent features:

This will help to understand which features are more responsible for output prediction.

Age

```
sns.distplot(df['age'])
```

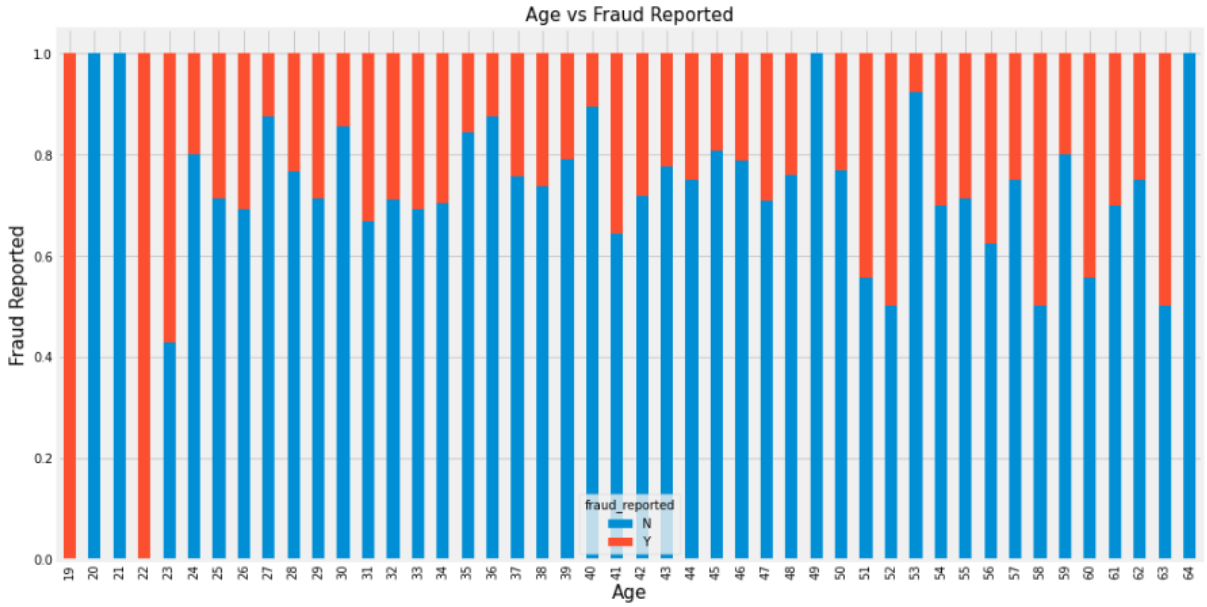
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```



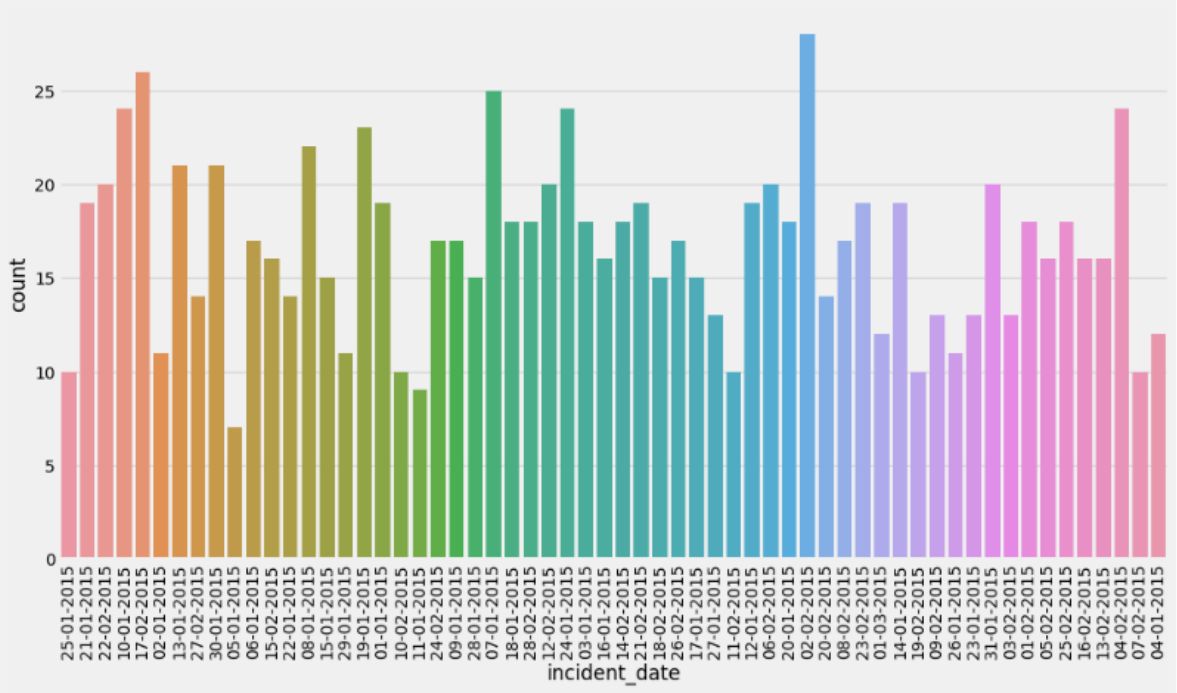
The distribution of Age data is not normally distributed, somehow right-skewed. This is the Age of Insurer.

```
plt.rcParams['figure.figsize'] = [15, 8]
table=pd.crosstab(df['age'],df['fraud_reported'])
table.div(table.sum(1),axis=0).plot(kind='bar',stacked=True)
plt.title('Age vs Fraud Reported',fontsize=15)
plt.xlabel('Age',fontsize=15)
plt.ylabel('Fraud Reported',fontsize=15)
```

```
Text(0, 0.5, 'Fraud Reported')
```



This visualization says that most cases where an insurer is 19 and 22 years old are totally fraud. Age has a very significant impact on target feature.



4 types of Incident file has been came for insurance.

Multi-Vehicle Collision (Most cases)

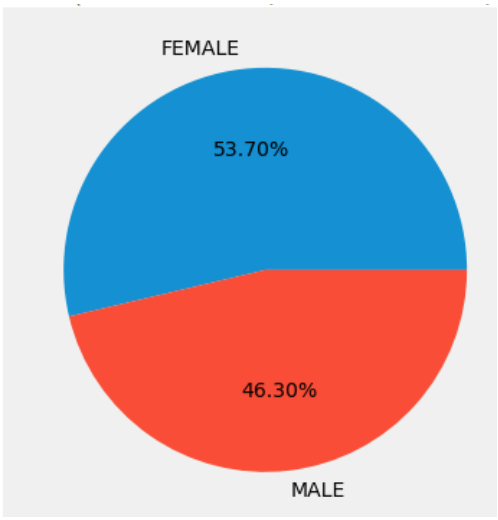
Single-Vehicle Collision

Vehicle-Theft

Parked Car

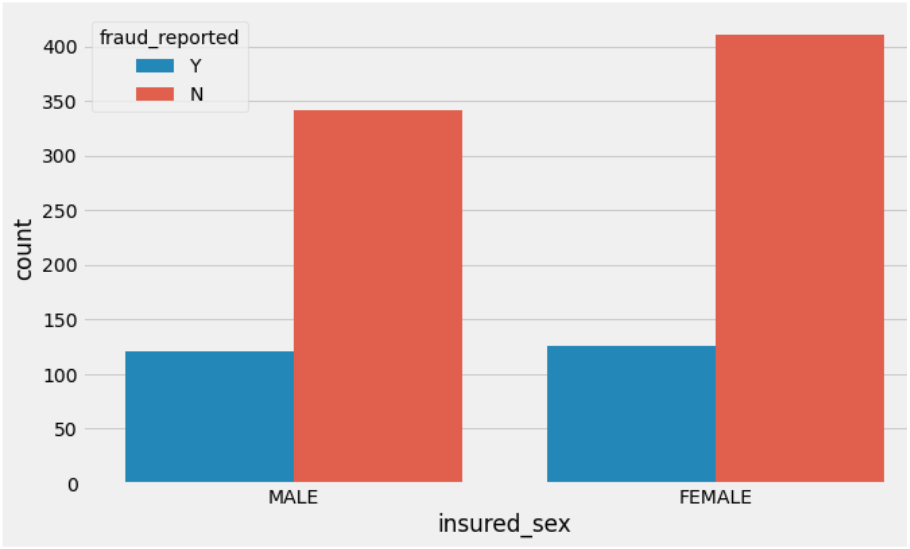
since 1995 to 2015 registered vehicles data we have among which most fraud cases came from 2004, 2007 and 2011 registered vehicles.

Let’s do some EDA on Insured person:



As per our data, we had almost equal cases of Male and female insurer.

53.7% are Male while 46.3% are female insurer.

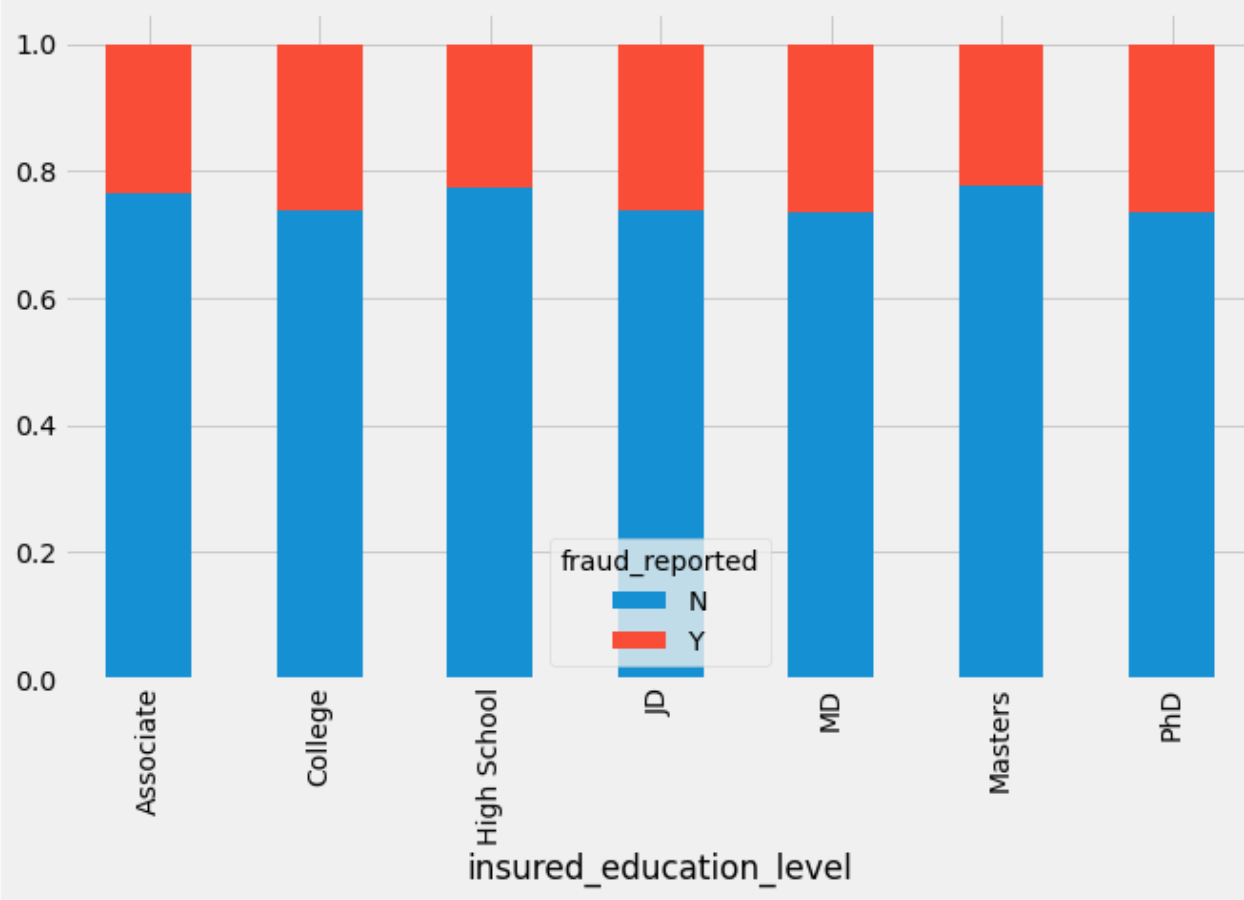


Among which chances are same for fraud case. Means fraud happening doesn't depends on insurer sex.

Insured_education_level

```
table=pd.crosstab(df['insured_education_level'],df['fraud_reported'])
table.div(table.sum(1),axis=0).plot(kind='bar',stacked=True)
```

```
<AxesSubplot:xlabel='insured_education_level'>
```

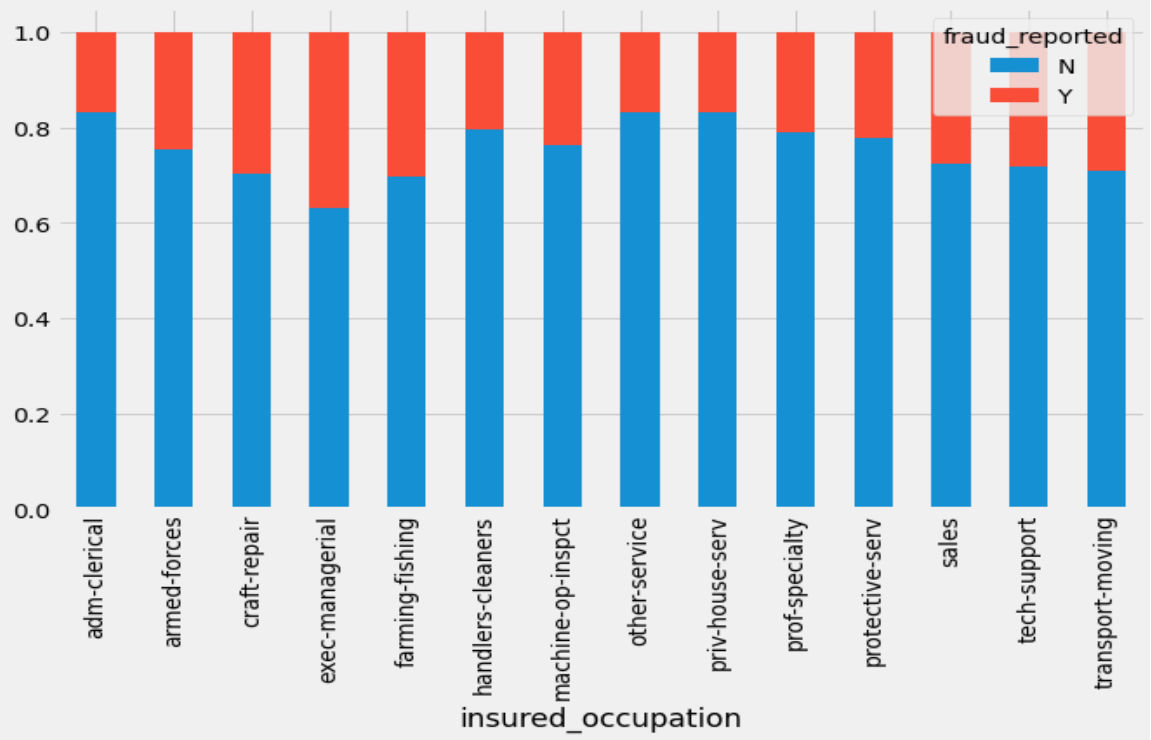


Almost chances are same for all educated insured. But specifically, College, JD, MD, PhD have done more frauds

If we go through, Insured occupation. Executive Manager have done most fraud. Insurer involved in the below occupation has done more frauds Craft-repair , farming-fishing, sales, tech-support, transport-moving

```
table=pd.crosstab(df['insured_occupation'],df['fraud_reported'])
table.div(table.sum(1),axis=0).plot(kind='bar',stacked=True)
```

```
<AxesSubplot:xlabel='insured_occupation'>
```



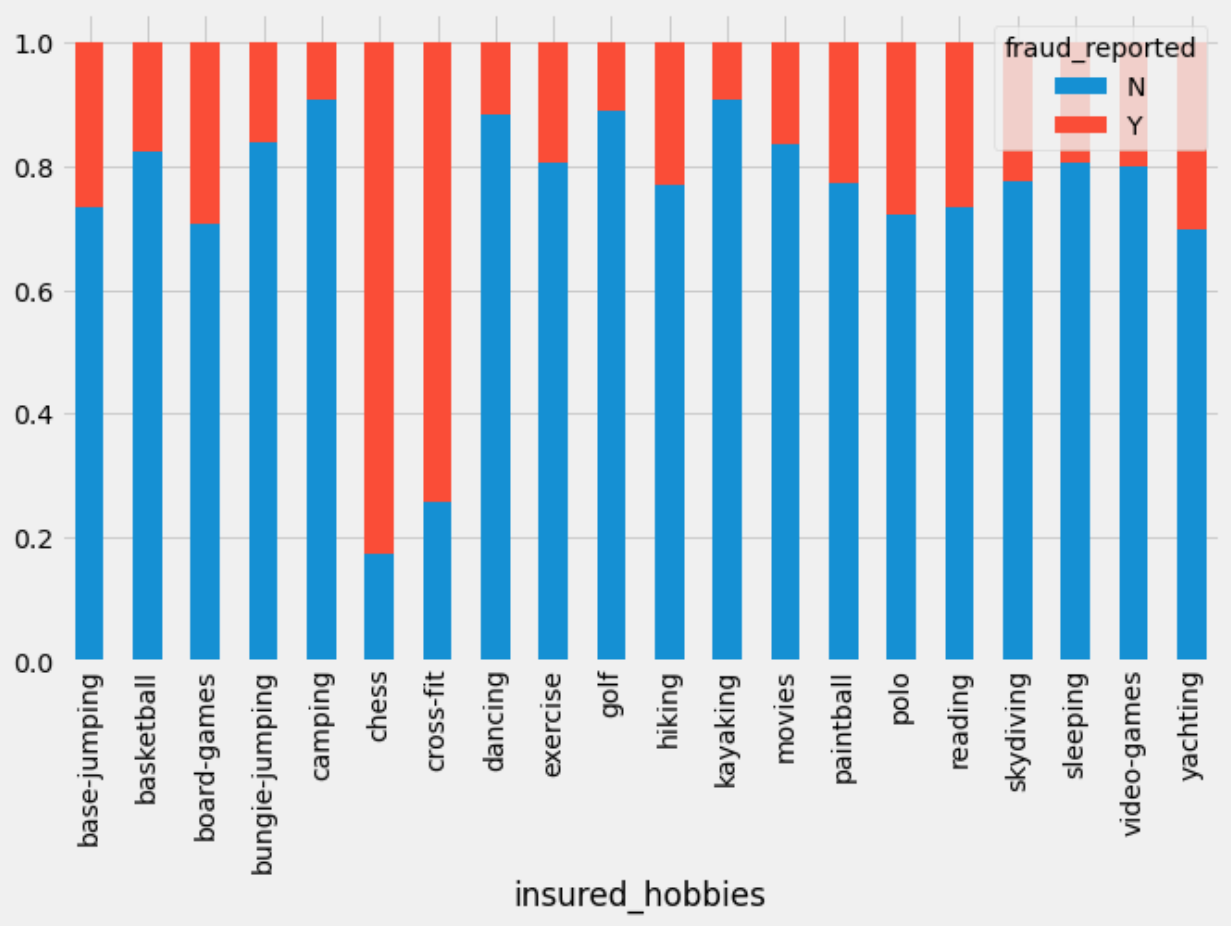
insured_hobbies vs fraud

Peoples having hobbies Chess and cross-fit are more crucial for insurance company because in mostly guilt cases insurer had the hobbies like chess and cross-fit.

Below visualization clears that hobbies can tell the intension and intelligence level of the insurer.

```
table=pd.crosstab(df['insured_hobbies'],df['fraud_reported'])
table.div(table.sum(1),axis=0).plot(kind='bar',stacked=True)
```

```
<AxesSubplot:xlabel='insured_hobbies'>
```

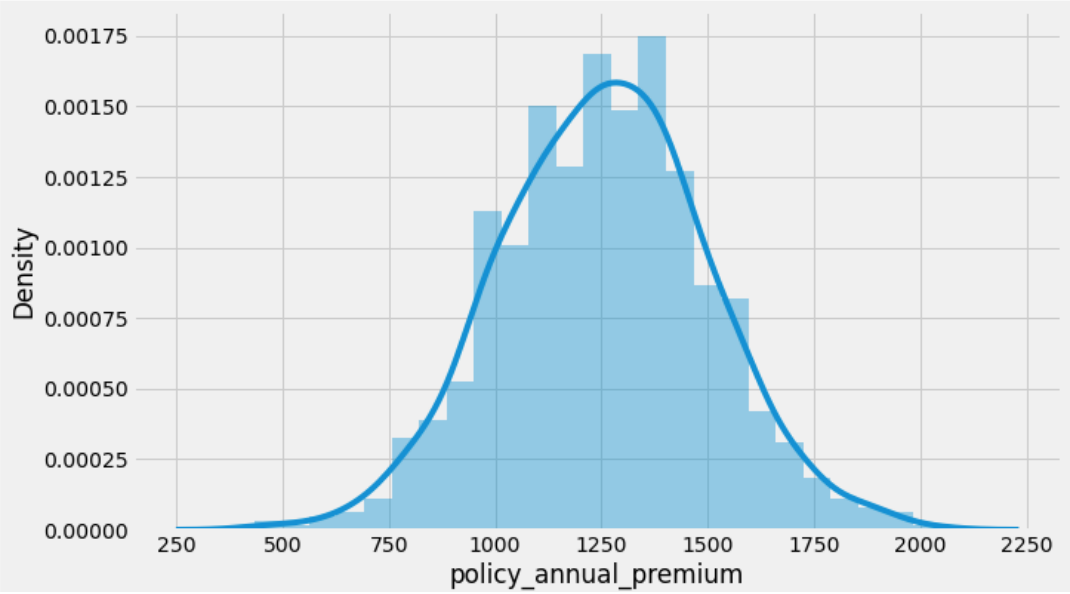


Policy_annual_premium

Policy annual premium seems to be having Normally distributed data. Little bit skewed on both sides.

```
sns.distplot(df['policy_annual_premium'])
```

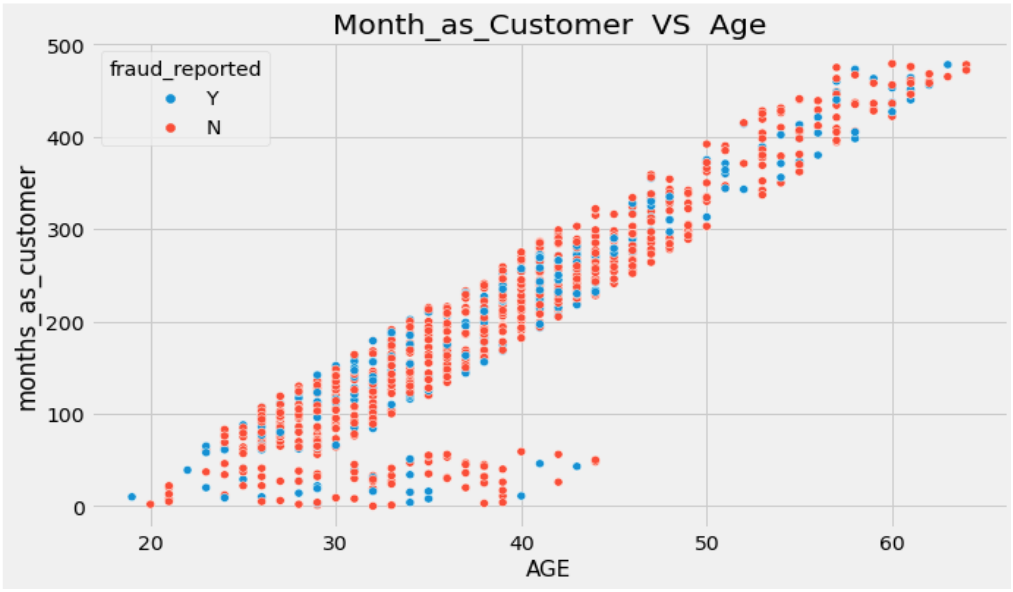
```
<AxesSubplot:xlabel='policy_annual_premium', ylabel='Density'>
```



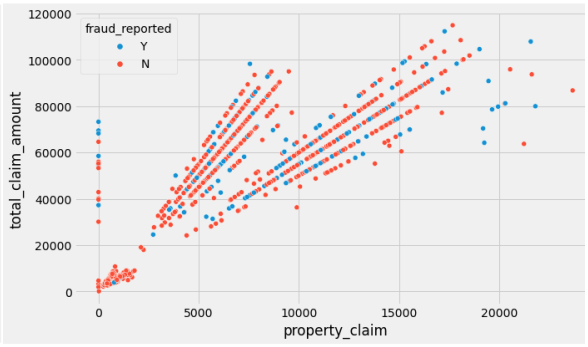
Month_as_customer vs Age

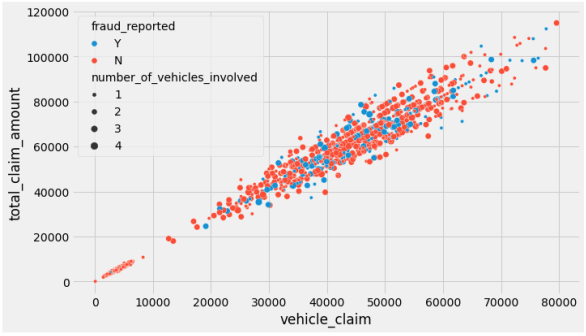
```
sns.scatterplot('age','months_as_customer',hue='fraud_reported',data=df)
plt.title('Month_as_Customer  VS  Age')
plt.xlabel('AGE',fontsize=15)
```

```
Text(0.5, 0, 'AGE')
```



Month_as_customer has very good positive correlation with the age of insurer. As the Age increases, months_as_customer increases with the company.





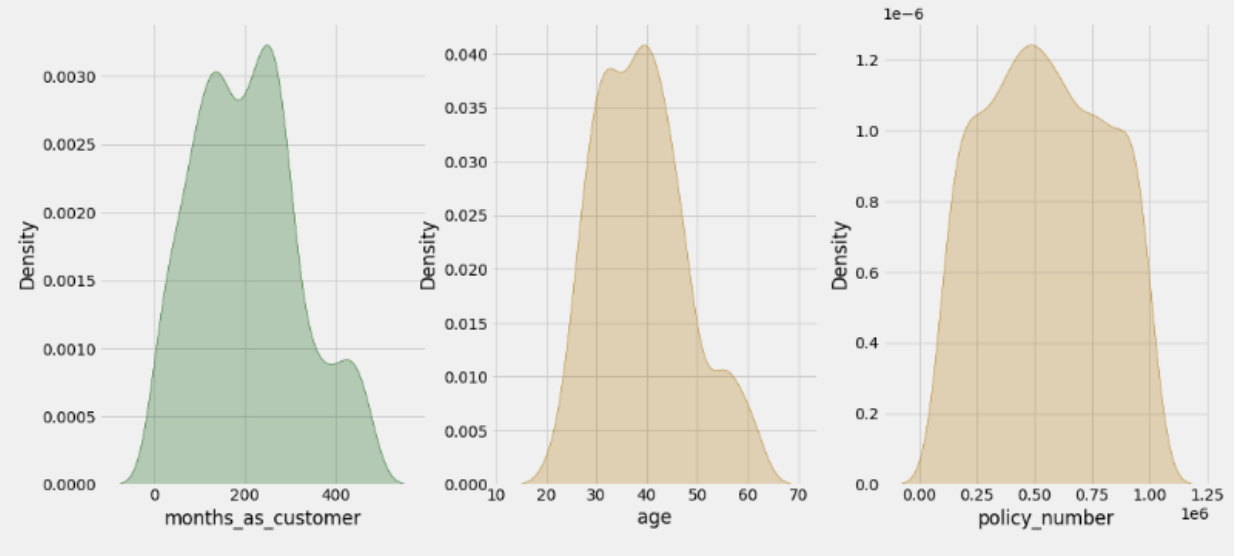
Total_claim_amount is highly good correlated with other claims like injury_claim, property_claim and vehicle_claim.

Upon investigation, we come to know that total_claim is the total of all 3 other claims. So later on we can drop this total_claim feature because its information has been covered by other 3 features.

Distribution of Numerical features

```
plt.figure(figsize=(16,50))
for i,col in enumerate(df[cont_features].columns):
    rand_col=color_[random.sample(range(6),1)[0]]
    plt.subplot(6,3,i+1)

    sns.kdeplot(data=df,x=col,color=rand_col,fill=rand_col,palette=cmap_[random.sample(range(3),1)[0]])
```



Missing Values

After replacing '?' into Nan values now our 3 categorical features have some missing / NaN values

Collision_type – 17.8% missing values

Property_damage – 36% missing values

Police_report_available - available – 34.3% missing values

% of missing values

```
df.isnull().sum()/df.shape[0]*100
```

```
months_as_customer            0.0
age                           0.0
policy_number                 0.0
policy_bind_date              0.0
policy_state                  0.0
policy_csl                    0.0
policy_deductable             0.0
policy_annual_premium         0.0
umbrella_limit                0.0
insured_zip                   0.0
insured_sex                   0.0
insured_education_level       0.0
insured_occupation            0.0
insured_hobbies               0.0
insured_relationship          0.0
capital-gains                 0.0
capital-loss                  0.0
incident_date                 0.0
incident_type                 0.0
collision_type               17.8
incident_severity             0.0
authorities_contacted         0.0
incident_state                0.0
incident_city                 0.0
incident_location             0.0
incident_hour_of_the_day      0.0
number_of_vehicles_involved   0.0
property_damage              36.0
bodily_injuries               0.0
witnesses                     0.0
police_report_available      34.3
total_claim_amount            0.0
injury_claim                  0.0
property_claim                0.0
vehicle_claim                 0.0
auto_make                     0.0
auto_model                    0.0
auto_year                     0.0
fraud_reported                0.0
dtype: float64
```

After analyse, we decided to fill NaN values with most frequent category within feature.

Wheresoever is null values, fill with Mode. This will fill all of your null values

```
Missing_coulmn=[]
for i in df.columns:
    if df[i].isnull().sum() !=0:
        df[i].fillna(df[i].mode()[0],inplace=True)
```

Feature Selection

Irrelevant columns:

- policy_number is not required as it is no help in prediction fraud case
- policy_bind_date is not required as we have months_as_customer, how old is policy.
- insured_zip is not required as we have policy_state and many more details for insured like sex, education, hobby, occupation, relationship

Feature Engineering

Policy_csl

```
CSL is Combined Single Limit:
```

```
df['policy_csl'].unique()
```

```
array(['250/500', '100/300', '500/1000'], dtype=object)
```

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

```
df['csl_per_person'].head()
```

```
0    250
1    250
2    100
3    250
4    500
Name: csl_per_person, dtype: object
```

```
df['csl_per_accident'].head()
```

```
0     500
1     500
2     300
3     500
4    1000
Name: csl_per_accident, dtype: object
```

We have done feature extraction here from policy_csl, we have created 2 new features csl_per_person and csl_per_sccident

Outliers

With the help of box plot, we can visualize if any outliers are present in our feature. Outliers are unusual values in data which are far from reality.

We have separated our available features into category and continuous according to feature types.

```
1 catg_features=[col for col in X.columns if X[col].dtypes=='object']
2 cont_features=[col for col in X.columns if X[col].dtypes!='object']
```

```
1 catg_features
```

```
['policy_state',
 'umbrella_limit',
 'insured_sex',
 'insured_education_level',
 'insured_occupation',
 'insured_hobbies',
 'insured_relationship',
 'incident_type',
 'collision_type',
 'incident_severity',
 'authorities_contacted',
 'incident_state',
 'incident_city',
 'property_damage',
 'police_report_available',
 'auto_make',
 'auto_model',
 'incident_period_of_the_day']
```

```
1 cont_features
```

```
['months_as_customer',
 'age',
 'policy_deductable',
 'policy_annual_premium',
 'capital-gains',
 'capital-loss',
 'number_of_vehicles_involved',
 'bodily_injuries',
 'witnesses',
 'total_claim_amount',
 'injury_claim',
 'property_claim',
 'vehicle_claim',
 'Vehicle_Age']
```

Draw Boxplot with continuous features:

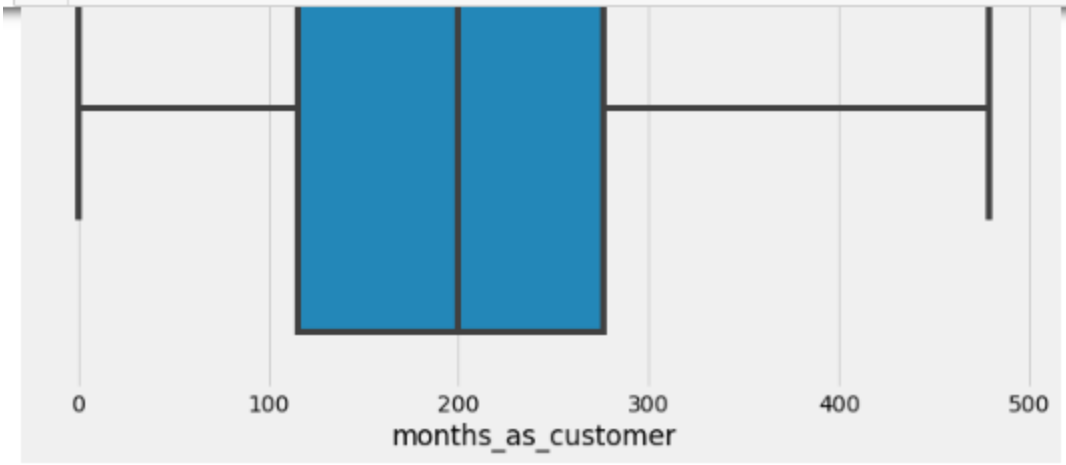
Step 1: Draw box plot for all continuous datatype features available

Step 2: check skewness in the features

Step 3: point out those features having outliers

Step 4: With IQR method, we can handle outliers in dataset

```
for i in cont_features:
    sns.boxplot(X[i])
    plt.show()
```




```
Few features have outliers, Lets handle them with IQR method
```

```
X[cont_features].skew()
```

```
months_as_customer             0.362177
age                            0.478988
policy_deductable              0.477887
policy_annual_premium          0.004402
capital-gains                  0.478850
capital-loss                  -0.391472
number_of_vehicles_involved    0.502664
bodily_injuries                0.014777
witnesses                      0.019636
total_claim_amount            -0.594582
injury_claim                   0.264811
property_claim                 0.378169
vehicle_claim                 -0.621098
Vehicle_Age                    0.048289
dtype: float64
```

```
missing_column=['age','policy_annual_premium','total_claim_amount','property_claim']
```

```
for i in missing_column:
    IQR= X[i].quantile(.75)-X[i].quantile(.25)
    lower=X[i].quantile(.25) - (1.5 * IQR)
    upper=X[i].quantile(.75) + (1.5 * IQR)
    X[i]=np.where(X[i]<lower,lower,X[i])
    X[i]=np.where(X[i]>upper,upper,X[i])
```

Feature Selection – Multi-Collinearity

Multi-Collinearity is unavoidable issue with data. Machine Learning Algorithms assumes that all independent features are correlated with target variable only. There is no relation between independent features but in reality this is not true. Somehow independent features are also correlated within independents features. Various techniques are available to find that correlation within independent features or feature importance accordingly to some extent we can handle multi collinearity.

Techniques like:

VIF

Constant Features

Mutual Info Gain

In this case, we are using VIF (Variance Inflation Factor) to find multicollinearity within our independent features only.

VIF works for continuous features only, also we will not include target variable because we want to find out the multicollinearity within independent features.

So, go ahead with continuous features with target feature.

VIF is 1/(1-R2)

The Variance Inflation Factor is a measure of collinearity among predictors variables within a multiple regression or

In simple words, this matric tells you how other variables are explaining your 1 variable. If VIF is large for 1 features means that can be very well explained by other features in your dataset. We don’t require that VIF with large value.

VIF

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
scaled= sc.fit_transform(X[cont_features])
```

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

```
VIF= pd.DataFrame()
VIF['features']=X[cont_features].columns
```

```
VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(cont_features))]
```

```
VIF
```

	features	vif
0	months_as_customer	6.815060
1	age	6.788114
2	policy_deductable	1.020949
3	policy_annual_premium	1.013444
4	capital-gains	1.014914
5	capital-loss	1.012754
6	number_of_vehicles_involved	1.095850
7	bodily_injuries	1.011043
8	witnesses	1.023162
9	total_claim_amount	47858.381223
10	injury_claim	1632.697036
11	property_claim	1607.393224
12	vehicle_claim	24471.259850
13	Vehicle_Age	1.015279

```
# However Total claim is the total of injury_claim + property_claim + vehicle_claim
# Delete total_claim_amount
```

```
X.drop('total_claim_amount',axis=1,inplace=True)
```

VIF required Scaled data, so we have used StandardScaler to bring all continuous features to scaled then VIF calculated.

VIF of total_claim_amount is very very large 47858, means this particular features can be explained by other variables and we also know that total_claim_amount is the total of all 3 claims. We can drop this features and calculate VIF again for the remaining features.

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
scaled= sc.fit_transform(X[cont_features])

VIF= pd.DataFrame()
VIF['features']=X[cont_features].columns

VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(cont_features))]
VIF
```

	features	vif
0	months_as_customer	6.772147
1	age	6.774011
2	policy_deductable	1.019308
3	policy_annual_premium	1.010403
4	capital-gains	1.013336
5	capital-loss	1.012154
6	number_of_vehicles_involved	1.092676
7	bodily_injuries	1.008444
8	witnesses	1.023126
9	injury_claim	2.128118
10	property_claim	2.242766
11	vehicle_claim	3.214606
12	Vehicle_Age	1.013401

```
month_as_customer and age is also high correalated with each other= .92
Delete Age, how ever we require how old the customer is for company
```

```
X.drop('age',axis=1,inplace=True)
```

Normal accepted values for VIF is -+5, Here, age and months_as_customer is more than 6. As pwe our analysis, we know age and month_as_customer feature are highly correlated. However for insurance purposes we require the details of how old the customer is so we can drop age features here.

Again, calculate VIF for remaining features.

Now, after dropping 2 variables VIF are in range for all features. Along with skewness are also in control.

Skewness accepted range is -+ .5

```
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
scaled= sc.fit_transform(X[cont_features])

VIF= pd.DataFrame()
VIF['features']=X[cont_features].columns

VIF['vif']= [variance_inflation_factor(scaled,i) for i in range(len(cont_features))]
VIF
```

	features	vif
0	months_as_customer	1.010202
1	policy_deductable	1.019296
2	policy_annual_premium	1.009315
3	capital-gains	1.012127
4	capital-loss	1.011092
5	number_of_vehicles_involved	1.092361
6	bodily_injuries	1.008084
7	witnesses	1.022882
8	injury_claim	2.125611
9	property_claim	2.225209
10	vehicle_claim	3.199822
11	Vehicle_Age	1.013396

```
X[cont_features].skew()
```

```
months_as_customer             0.362177
policy_deductable              0.477887
policy_annual_premium          0.016003
capital-gains                  0.478850
capital-loss                  -0.391472
number_of_vehicles_involved    0.502664
bodily_injuries                0.014777
witnesses                      0.019636
injury_claim                   0.264811
property_claim                 0.348531
vehicle_claim                 -0.621098
Vehicle_Age                    0.048289
dtype: float64
```

For complete Python file: Click Here

Transformation and Standardization

Data transformation is the process of converting raw data into a format or structure that would be more suitable for model building and also data discovery in general.

Data standardization is the process of rescaling the attributes so that they have mean as 0 and variance as 1

```
from sklearn.preprocessing import power_transform
from sklearn.preprocessing import StandardScaler
```

```
for i in cont_features:
    pow=power_transform(X[cont_features])
    X[i]=sc.fit_transform(pow)
```

Power_transform will transform the data and StandardScaler will scale down the data

Here, we have fixed the numerical features.

Encoding

Machine Learning Algorithms are trained to understand digits only, ML Algos can't work on strings or categorical data so we have to convert categorical data into numerical.

Categorical data are of 2 types:

1. Nominal 2. Ordinal

Nominal data can be converted into numerical by get_dummy method or one-hot encoding while

Ordinal data have order within feature according to their weightage.

Remaining categorical are Nominal

X= independent features

Y= dependent feature

Imbalance data- SMOTE

Our data is imbalance as our target feature have 1 type of data is more than the other. Here if we process imbalanced data to the machine algorithm, it will be learning more for 1 type of data that will create bias in the target prediction.

```
df['fraud_reported'].value_counts(normalize=True)*100
```

```
N    75.3
Y    24.7
Name: fraud_reported, dtype: float64
```

As of now, our data have 75.3% cases of Genuine and 24.7% cases of fraud so ML will learn more about genuine cases.

Various Techniques can be used to balance the data, here we will use SMOTE or oversampling. It will create more records to balance the target feature as per their classification.

```
from imblearn.over_sampling import SMOTE
sm=SMOTE()
x,y=sm.fit_resample(X,Y)
```

```
x.shape , y.shape
```

```
((1506, 123), (1506,))
```

```
round(y.value_counts(normalize=True) * 100, 2).astype('str') + ' %'
```

```
0    50.0 %
1    50.0 %
Name: Target, dtype: object
```

Data is balanced now, we are ready to feed data into the machine algorithm

Building Machine Learning Models

Import required libraries for machine Learning

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
```

Train_test_split is used to split the complete dataset into the train and test portions. Train data will be used to train the Model then with test data we will compare the accuracy.

We will check the accuracy of the model through metrics like accuracy_score,

confusion_matrix,

classification_report

f1 score

```
x_train,x_test,y_train,y_test= train_test_split(x,y,random_state=8,test_size=.3)
```

Data split into x_train,x_test,y_train ,y_test

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import RidgeClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

```
LR_model= LogisticRegression()
RD_model= RidgeClassifier()
DT_model= DecisionTreeClassifier()
SV_model= SVC()
KNR_model= KNeighborsClassifier()
RFR_model= RandomForestClassifier()
XGB_model= XGBClassifier()
SGH_model= SGDClassifier()
Bag_model=BaggingClassifier()
ADA_model=AdaBoostClassifier()
GB_model= GradientBoostingClassifier()

model=[LR_model,RD_model,DT_model,SV_model,KNR_model,RFR_model,XGB_model,SGH_model,Bag_model,ADA_model,GB_model ]
```

I have taken most of the Algos, so you can find the difference.

Now, though the below code, fit the data with every ML one by one and calculate accuracy of the model and f1 score

```
accuracy=[]
f1=[]

for m in model:
    m.fit(x_train,y_train)
    m.score(x_train,y_train)
    pred= m.predict(x_test)
    accuracy.append(round(accuracy_score(y_test,pred) * 100, 2))
    f1.append(round(f1_score(y_test,pred) * 100, 2))
```

```
pd.DataFrame({'Model':model,'Accuracy':accuracy,'F1 Score':f1})
```

	Model	Accuracy	F1 Score
0	LogisticRegression()	91.37	90.18
1	RidgeClassifier()	92.26	91.40
2	DecisionTreeClassifier()	86.28	84.65
3	SVC()	87.39	84.96
4	KNeighborsClassifier()	49.78	63.80
5	(DecisionTreeClassifier(max_features='auto', r...	90.49	88.89
6	XGBClassifier(base_score=0.5, booster='gbtree'...	91.37	90.32
7	SGDClassifier()	79.65	81.30
8	(DecisionTreeClassifier(random_state=103366839...	90.71	89.71
9	(DecisionTreeClassifier(max_depth=1, random_st...	88.05	86.50
10	([DecisionTreeRegressor(criterion='friedman_ms...	89.16	87.90

Cross Validation

The goal of cross validation is to test the model's ability to predict new data that was not used in estimating it, in order to flag problems like overfitting or section bias and to give an insight on how the model will generalize to an independent dataset.

Like an unknown dataset, for instance from a real problem

```
from sklearn.model_selection import cross_val_score
```

```
acc=[]
cross=[]
diff=[]
for i in model:
    acc.append(accuracy_score(y_test,i.predict(x_test))*100)
    cross.append(cross_val_score(i,x,y,cv=5, scoring='accuracy').mean()*100)
    diff.append((accuracy_score(y_test,i.predict(x_test))*100)- (cross_val_score(i,x,y,cv=5, scoring='accuracy').mean()*100))

pd.DataFrame({'Model':model,'Accuracy':acc,'Cross Validation':cross,'Difference':diff})
```

From this code, we will get the accuracy score of all models with the train dataset along with cross validation with complete dataset.

	Model	Accuracy	Cross Validation	Difference
0	LogisticRegression()	91.371681	85.670502	5.701179
1	RidgeClassifier()	92.256637	84.410244	7.846393
2	DecisionTreeClassifier()	86.283186	83.872742	1.548637
3	SVC()	87.389381	83.351301	4.038079
4	KNeighborsClassifier()	49.778761	54.250292	-4.471530
5	(DecisionTreeClassifier(max_features='auto', r...	90.486726	86.665860	3.091289
6	XGBClassifier(base_score=0.5, booster='gbtree'...	91.371681	87.525687	3.845994
7	SGDClassifier()	79.646018	83.613342	-3.434443
8	(DecisionTreeClassifier(random_state=103366839...	90.707965	86.462344	3.183818
9	(DecisionTreeClassifier(max_depth=1, random_st...	88.053097	85.006711	3.046387
10	([DecisionTreeRegressor(criterion='friedman_ms...	89.159292	86.129458	2.963609

For a generalized model, we select the model with minimum difference between accuracy of train data and the accuracy score of the complete dataset.

As per our requirement and based on analysis, we will decide the model to go with.

Here, we are going further with GradientBoostingClassifier

HyperTuning

Basically, Models work on defaults parameters, so if we can change the parameters upon our requirement we can also improve the pehrformance of the model

For hyper tuning, we can use RandomSearchCV and GridSearchCV

RandomSearchCV will select few parameters combinations from the options while GridSearchCV will try all parameters combinations.

```
from sklearn.model_selection import GridSearchCV
```

```
params= {"learning_rate"    : [0.01,.05,.1,.2,.3,.5 ] ,
         'n_estimators':[5,50,100,200,300,400],
         "max_depth"        : [ 3, 4, 5, 6, 8]
          }
```

```
GCV= GridSearchCV(GB_model,params,cv=5,scoring='accuracy', n_jobs=-1)
GCV.fit(x_train,y_train)
```

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid={'learning_rate': [0.01, 0.05, 0.1, 0.2, 0.3, 0.5],
                         'max_depth': [3, 4, 5, 6, 8],
                         'n_estimators': [5, 50, 100, 200, 300, 400]},
             scoring='accuracy')
```

We have passed 3 parameters options in the dictionary.

CV=5 , it will cross validate 5 times

N_jobs= -1 will use every core for this computation

Fit with train data

```
GCV.best_estimator_
```

```
GradientBoostingClassifier(learning_rate=0.05, max_depth=8, n_estimators=50)
```

```
GCV.best_params_
```

```
{'learning_rate': 0.05, 'max_depth': 8, 'n_estimators': 50}
```

```
pred=GCV.best_estimator_.predict(x_test)
accuracy_score(y_test,pred)
```

```
0.8849557522123894
```

GCV.best_estimator_ and GCV.best_params provides the best estimator for this cross-validation

This is not necessary that hyper tuning will always work better, sometimes default parameters provides the best results.

Default parameters accuracy is 89.15

Hypertuned accuracy is 88.49

Other Metrics to evaluate

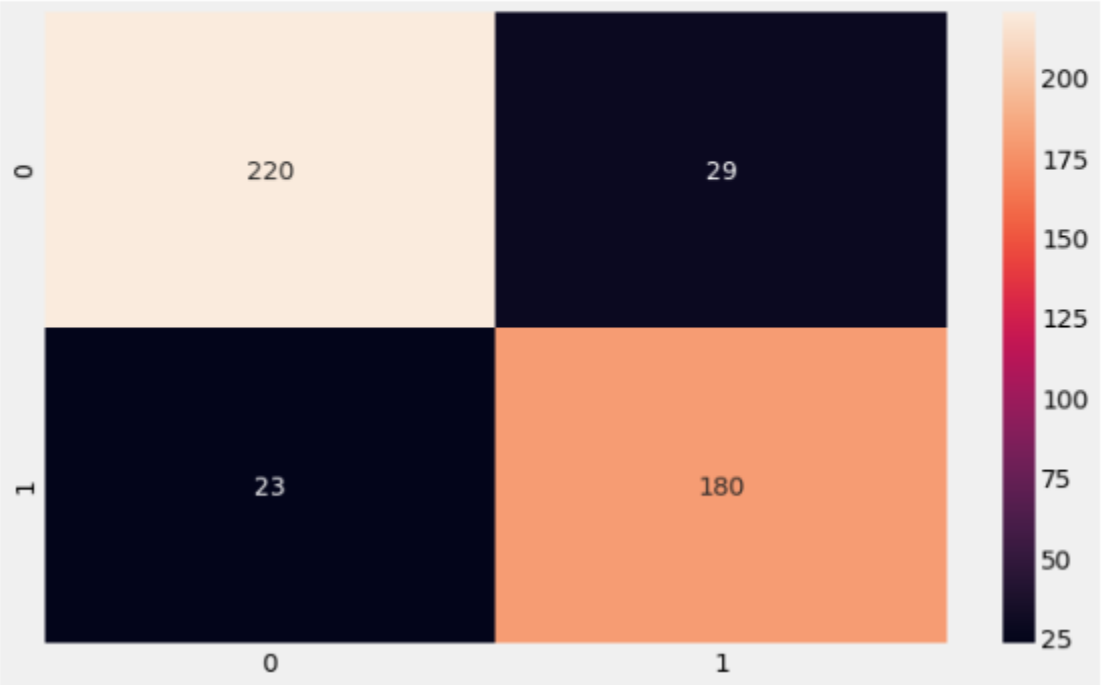
Confusion Matrix

The number of cases for each class of the test set is shown in the confusion matrix below.

The y-axis shows the actual classes while the x-axis shows the predicted classes.
Percentage out of the total sample size of the test set is printed on each quadrant.

```
: from sklearn.metrics import confusion_matrix
  confusion_matrix(y_test,pred)
  sns.heatmap(confusion_matrix(y_test,pred),annot=True, fmt='d')
```

```
: <AxesSubplot:>
```



ROC AUC Curve

The ROC curve below summarizes how well our model is at balancing between the true positive rate (sensitivity) and the false positive rate(1-specificity). Ideally, we want to have a 100% true positive rate of predicting fraud and a 100% true negative rate of predicting non-frauds (or a 0% false-positive which is 100% — 100% true negative rate). This means we have a perfect prediction for both classes. However, in imbalance class problems, this is extremely hard to achieve in the real world. On top of that, there is a trade between the true positive rate and the true negative rate and conversely the false positive rate.

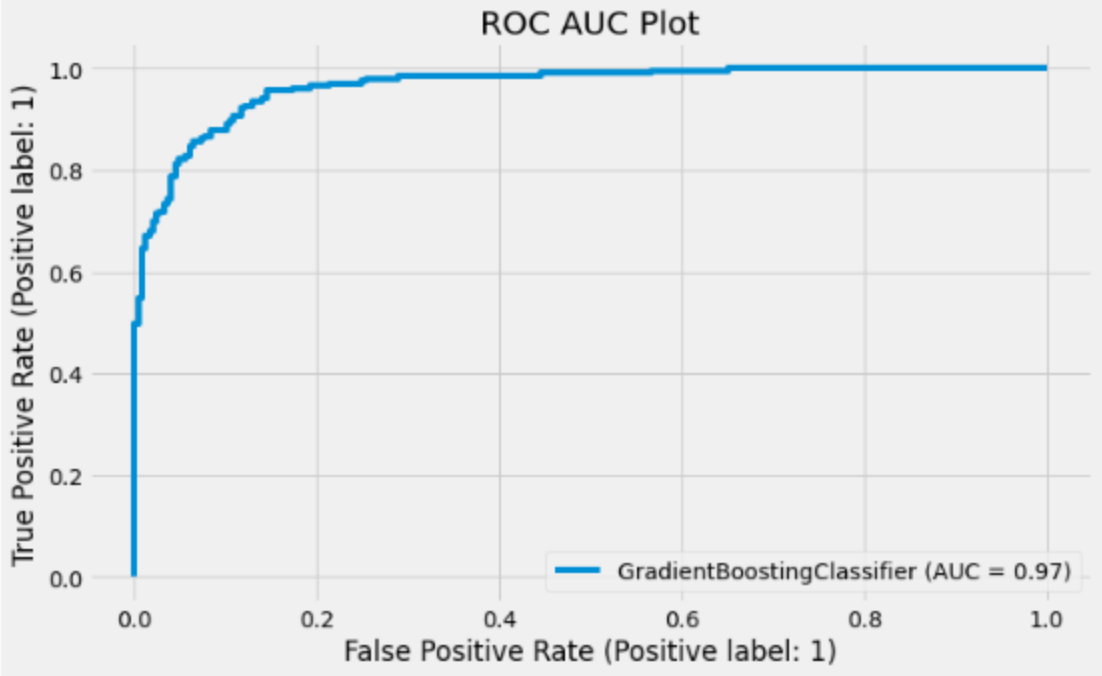
This graph summarizes how well we can distinguish between two classes at each threshold of the true positive and false positive rate. The area under curve is used as a summary percentage of this metric. In sum, the model has outperformed the baseline ROC AUC scores by a huge margin.

Although our model performed better in predicting non-fraud cases, the model has performed very well on fraud cases as well. We have a higher false alarm than frauds escaping the detection. It is better in our case to identify more frauds than to let fraud cases escape detection. Thus, this model has succeeded in its purpose to detect fraud claims. Unlike the baseline model that sacrifices too much resources into investigations and hinder customer experience, we are also able to balance this out in this model. We can detect more fraud and we are able to balance this with correct prediction of non-fraud cases

```
from sklearn.metrics import roc_auc_score,roc_curve,plot_roc_curve
```

```
plot_roc_curve(GB_model,x_test,y_test)
plt.title('ROC AUC Plot')
```

```
Text(0.5, 1.0, 'ROC AUC Plot')
```



ROC AUC= 97%

Concluding Remarks

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.

In conclusion, the model was able to correctly distinguish between fraud claims and legit claims with high accuracy**.**