Insurance fraud Claim- Automobile

Insurance fraud detection is a challenging problem. Machine learning techniques allow us to identify and manage these frauds. By the use of Ml we can improve predictive accuracy and can enable loss control units to achieve higher coverage with low false positive rates.

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

Business case:

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

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

Explanatory Data Analysis & Pre-Processing Pipeline

About the dataset -

- Data Source: https://github.com/dsrscientist/Data-Science-ML-Capstone-Projects/blob/master/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 your environment for analysis.

In this case, we are using, Jupyter Notebook.

```
In [2]: # Load dataset
        df= pd.read_csv('Automobile_insurance_fraud.csv')

In [3]: pd.set_option('display.max_columns',None)

In [4]: df.head()
```

Out[4]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	17-10-2014	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	27-06-2006	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	06-09-2000	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	25-05-1990	IL	250/500	2000	1415.74	6000000	608117
4	228	44	367455	06-06-2014	IL	500/1000	1000	1583.91	6000000	610706

The current data set has 1000 records and has 40 different columns.

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

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

In [7]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 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
```

```
 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
 39  _c39                         0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 312.6+ KB
```

As we can see from **df.info** command the columns types and names are shown.

These are as follows -

We have 17 integer features, 21 Object features, 2 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

```
Out of 40 features:
    17 of Integer Type
    2 of Float Type
    21 of Object types
```

Out of these,

There are 38 different features on which basis it decided about the 39th feature (Target feature).

Now we will see which columns are categorical and which are continuous -

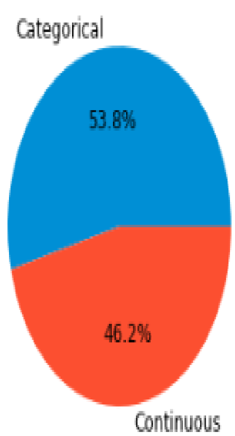
```
catg_features=[col for col in df.columns if df[col].dtypes=='object']
cont_features=[col for col in df.columns if df[col].dtypes!='object']
```

```
print(f'Number of Categorical features: {len(catg_features)}')
print(f'Number of Continuous features: {len(cont_features)}')
```

```
Number of Categorical features: 21
Number of Continuous features: 18
```

```
plt.pie([len(catg_features),len(cont_features)],labels=['Categorical','Continuous'],textprops={'fontsize':12},autopct='%1.1f%%')
```

```
([<matplotlib.patches.Wedge at 0x1fc979de7c0>,
  <matplotlib.patches.Wedge at 0x1fc979e02e0>],
 [Text(-0.13259044265487685, 1.0919797500487745, 'Categorical'),
  Text(0.13259054489339886, -1.0919797376347566, 'Continuous')],
 [Text(-0.07232205962993281, 0.5956253182084223, '53.8%'),
  Text(0.07232211539639936, -0.5956253114371399, '46.2%')])
```



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

Then our next step is to know the authenticity of our dataset, we check for the Null values (empty entry)

Missing Values

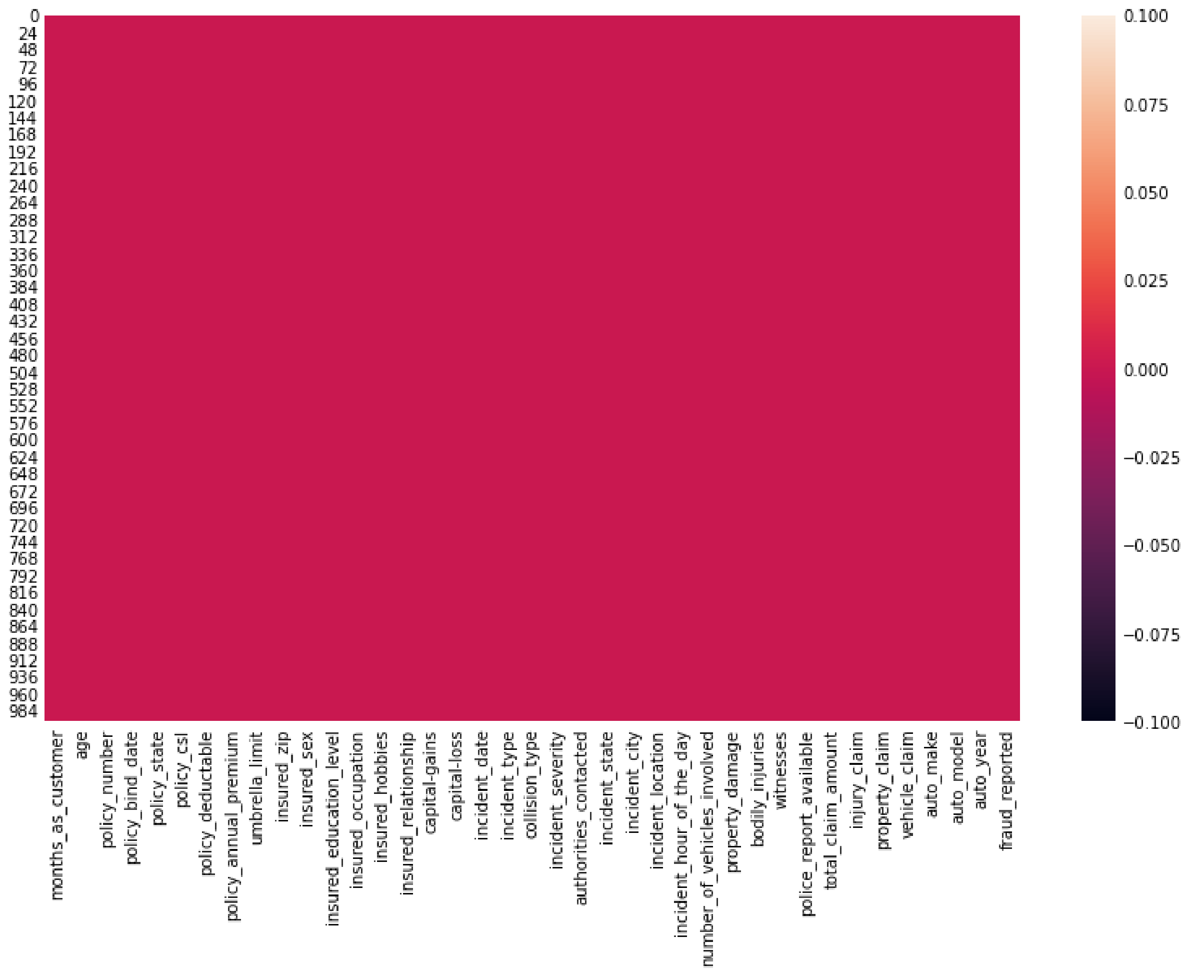
```
df.isnull().sum()
```

```
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:>
```



So, 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
```

Next step, 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

And many more methods.

EDA Concluding Remark

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

Correlation Amount variables

Correlation is the quantitative method to find the relationship between independent features with Target feature as well within an independent feature.

Month_as_customer and age had .92 correlation

1. 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

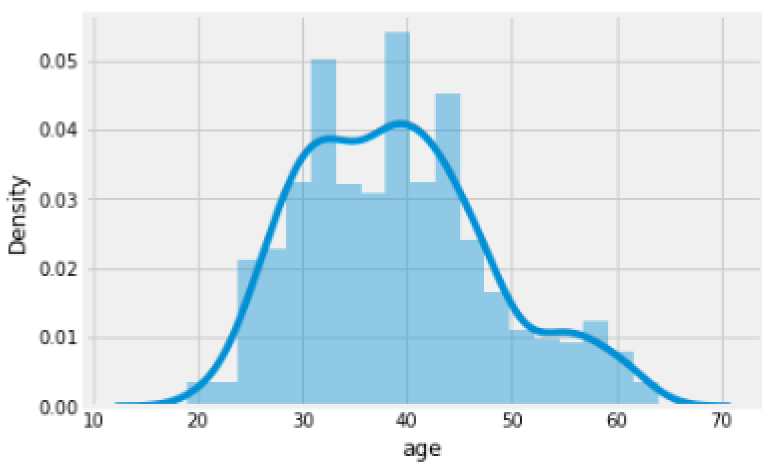
EDA on Independent features

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

Age

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

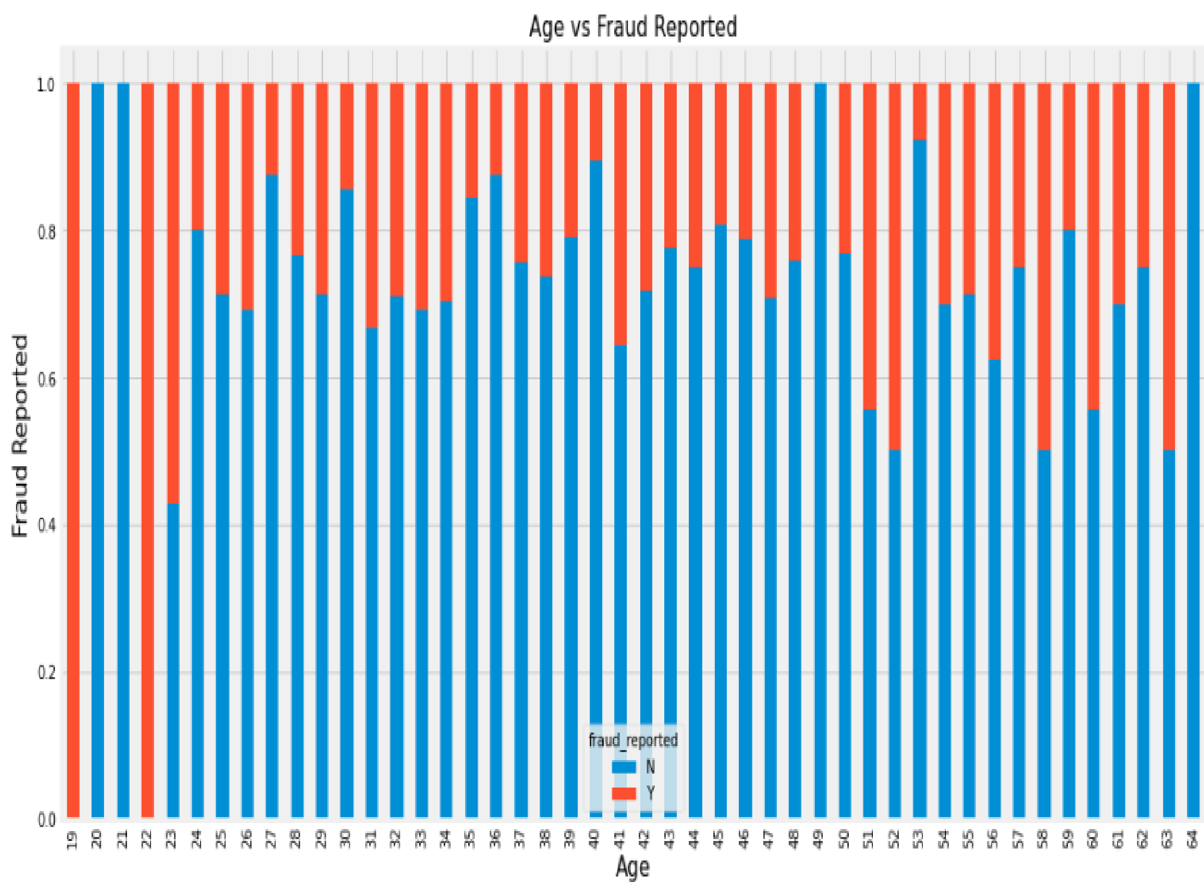
```
<AxesSubplot:xlabel='age', ylabel='Density'>
```



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.

Incident_state

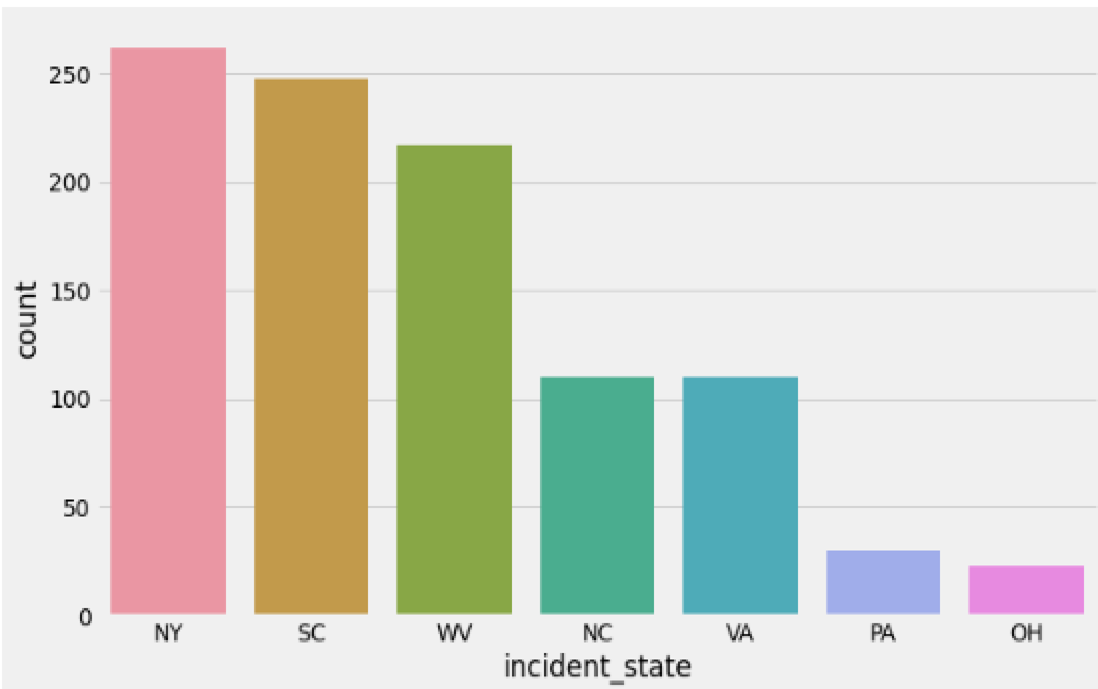
According to this data, Most of the incident happened in NY state, after SC, WV.

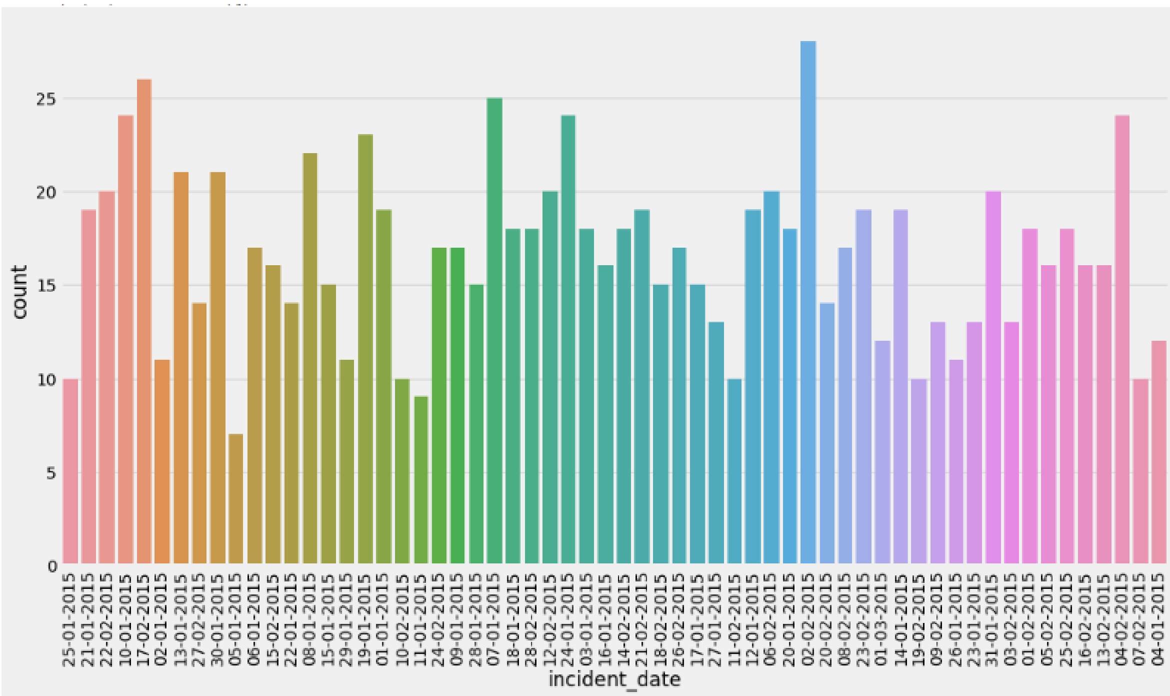
```
df['incident_state'].nunique()
```

7

```
sns.countplot(df['incident_state'],order=df['incident_state'].value_counts().index)
```

<AxesSubplot:xlabel='incident_state', ylabel='count'>





Date : For this data records, all incidents happened in January and February of 2015

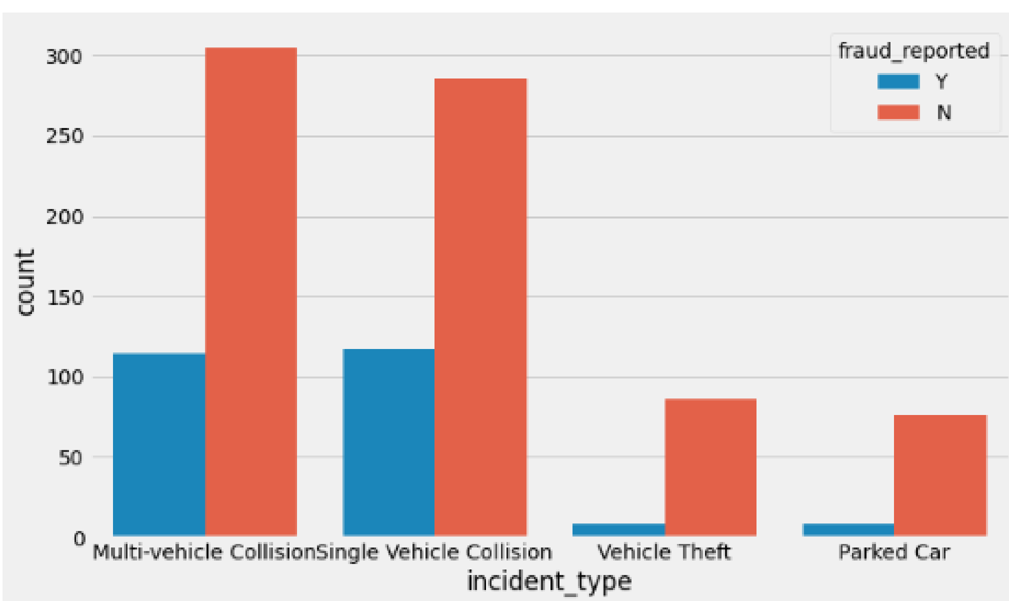
incident_type vs fraud

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

```
array(['Single Vehicle Collision', 'Vehicle Theft',
       'Multi-vehicle Collision', 'Parked Car'], dtype=object)
```

```
sns.countplot(df['incident_type'], order=df['incident_type'].value_counts().index,hue=df['fraud_reported'])
```

```
<AxesSubplot:xlabel='incident_type', ylabel='count'>
```



4 types of Incident filed has been came for insurance-

Multi-Vehicle Collision (Most cases)

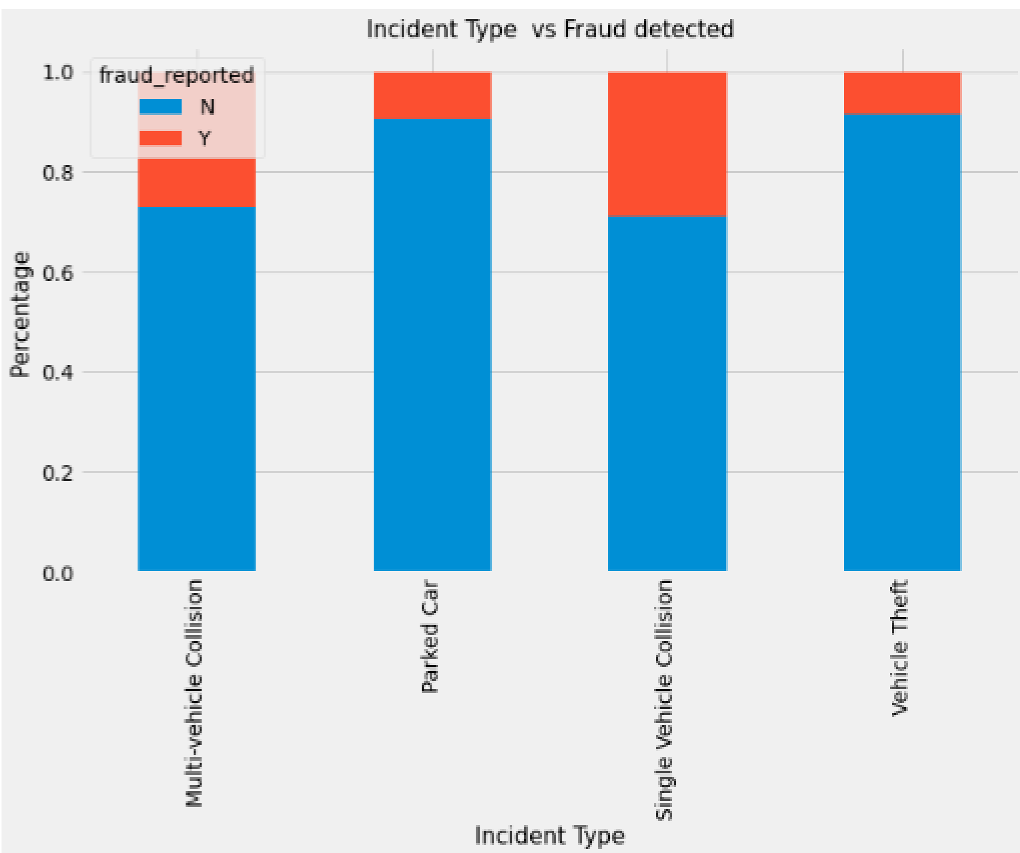
Single-Vehicle Collision

Vehicle-Theft

Parked Car

```
table=pd.crosstab(df['incident_type'],df['fraud_reported'])
table.div(table.sum(1),axis=0).plot(kind='bar',stacked=True)
plt.title("Incident Type  vs Fraud detected",fontsize=15)
plt.xlabel('Incident Type',fontsize=15)
plt.ylabel('Percentage ',fontsize=15)
```

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



One can notice, most fraud cases were found with incident type multi-Vehicle Collision and Single Vehicle collision.

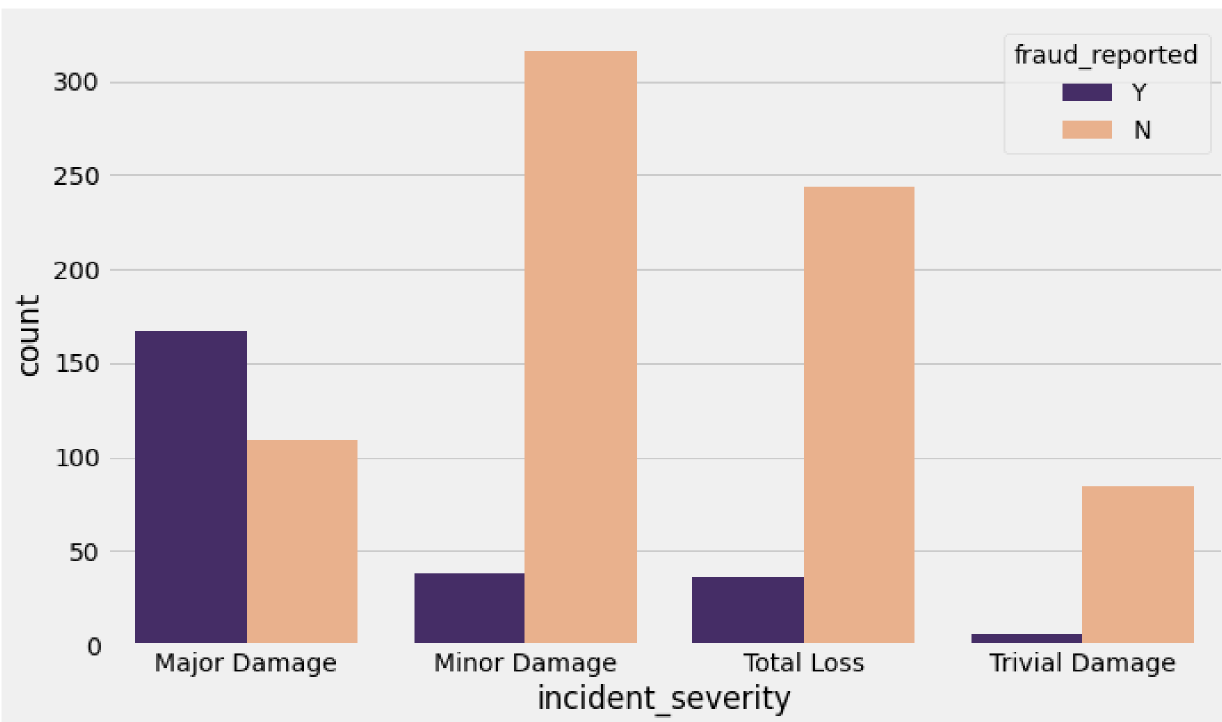
This could be the case in a parked car, parking in charge has to take care of the parked car along with CCTV security. Fraud cases would be difficult to generate from here.

Vehicle theft involves police FIR, which can lead to serious investigation so fraud cases are less in these incidents.

Incident Severity vs fraud_reported:

```
sns.countplot(df['incident_severity'],hue=df['fraud_reported'],palette=['#432371',"#FAAE7B"])
```

```
<AxesSubplot:xlabel='incident_severity', ylabel='count'>
```



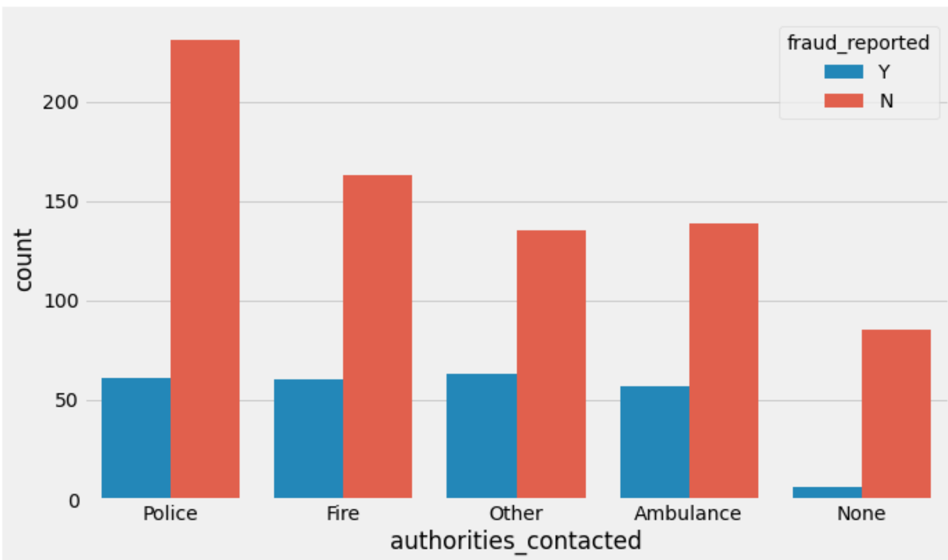
Most of the cases came with Minor damage, then total loss and major damage. This visualization shows major damage have more fraud_cases.

authorities_contacted vs fraud_reported

As per accident severity, Police has been contacted most of the time after accident. There are equal chances of fraud when the authorities contacted after accident are Police, Fire, Other , Ambulance however, no one contacted in most of genuine cases.

```
sns.countplot(df['authorities_contacted'], order=df['authorities_contacted'].value_counts().index, hue=df['fraud_reported'])
```

```
<AxesSubplot:xlabel='authorities_contacted', ylabel='count'>
```



number_of_vehicles_involved

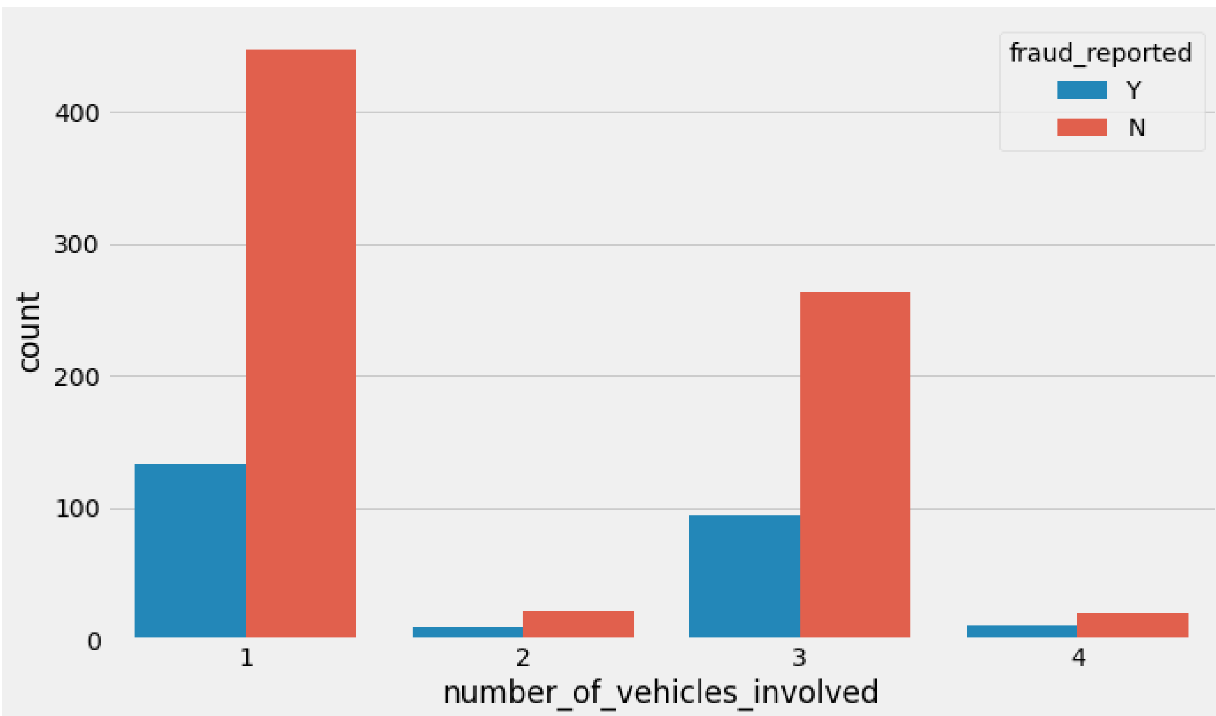
As we have investigated already, most of the reported accidents involved 1 or 3 vehicle. So accordingly, fraud cases are more with accident 1 or 3 vehicle.

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

```
array([1, 3, 4, 2], dtype=int64)
```

```
sns.countplot(df['number_of_vehicles_involved'], hue=df['fraud_reported'])
```

```
<AxesSubplot:xlabel='number_of_vehicles_involved', ylabel='count'>
```

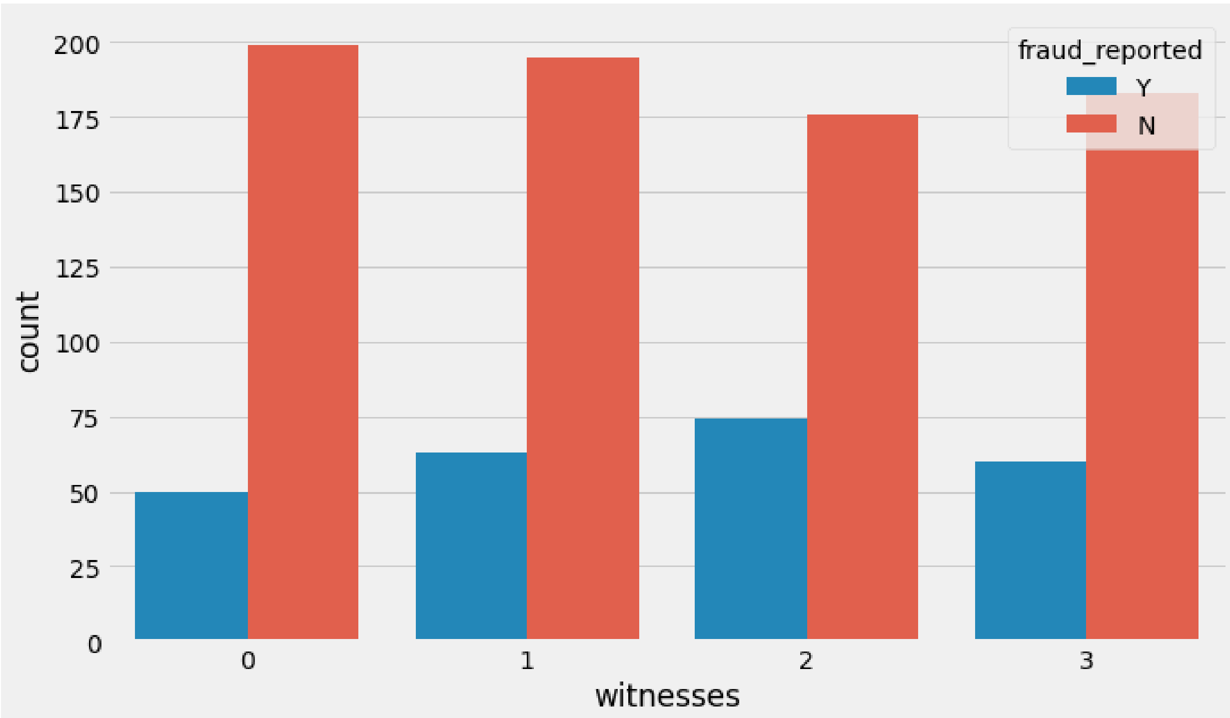


Witnesses

Fraud are more , when witnesses are 1 or 2.

```
sns.countplot(df['witnesses'],hue=df['fraud_reported'])
```

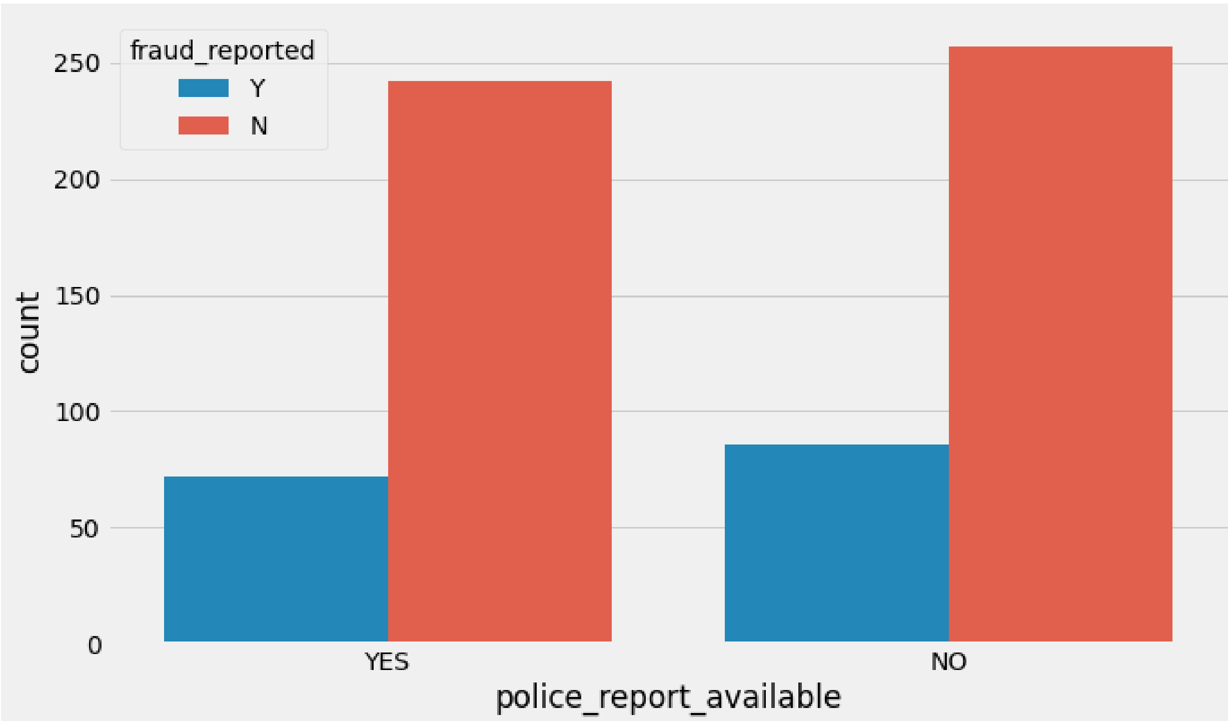
```
<AxesSubplot:xlabel='witnesses', ylabel='count'>
```



police_report_available

```
sns.countplot(df['police_report_available'],hue=df['fraud_reported'])
```

```
<AxesSubplot:xlabel='police_report_available', ylabel='count'>
```

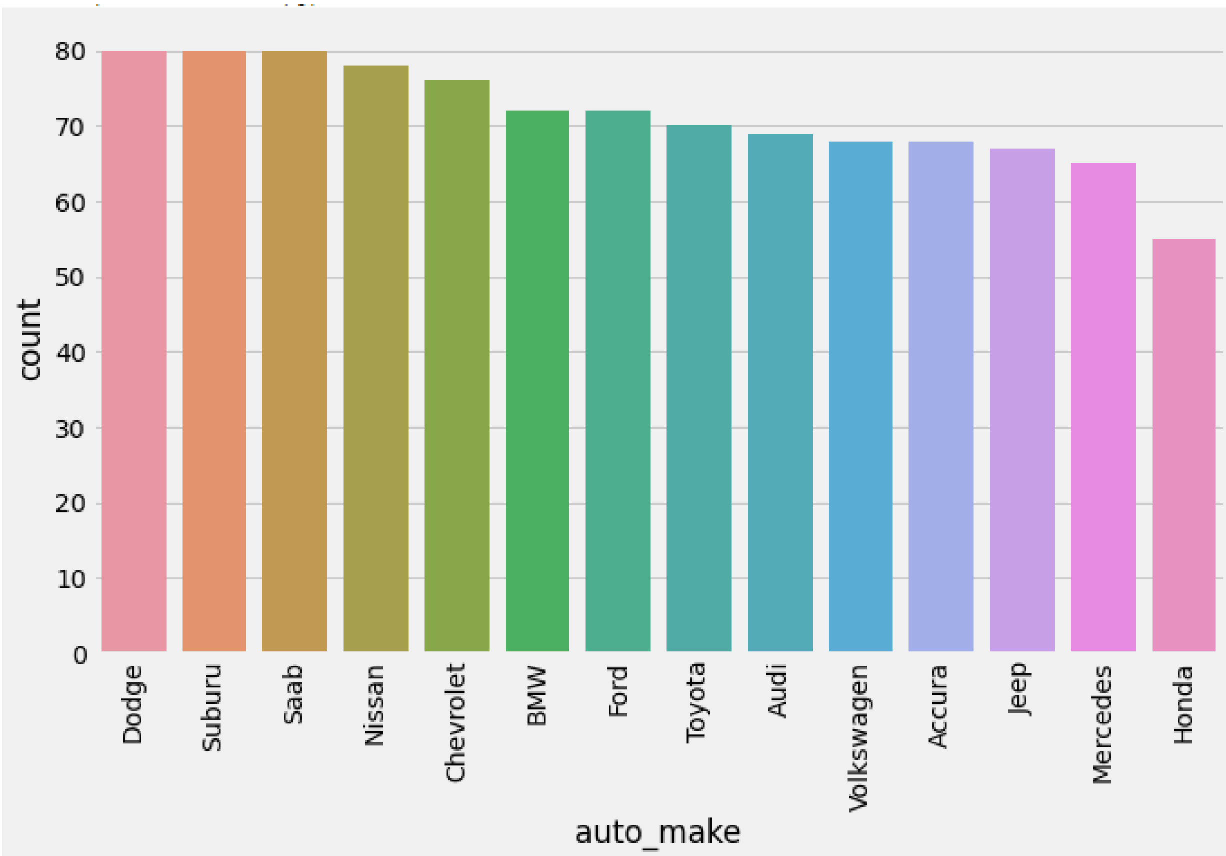


Fraud cases are more when police report is not available. Obviously, guilty person will not contact police for fraud cases.

Vehicle details which was included in accidents.

auto_make

This is the company which make vehicles -



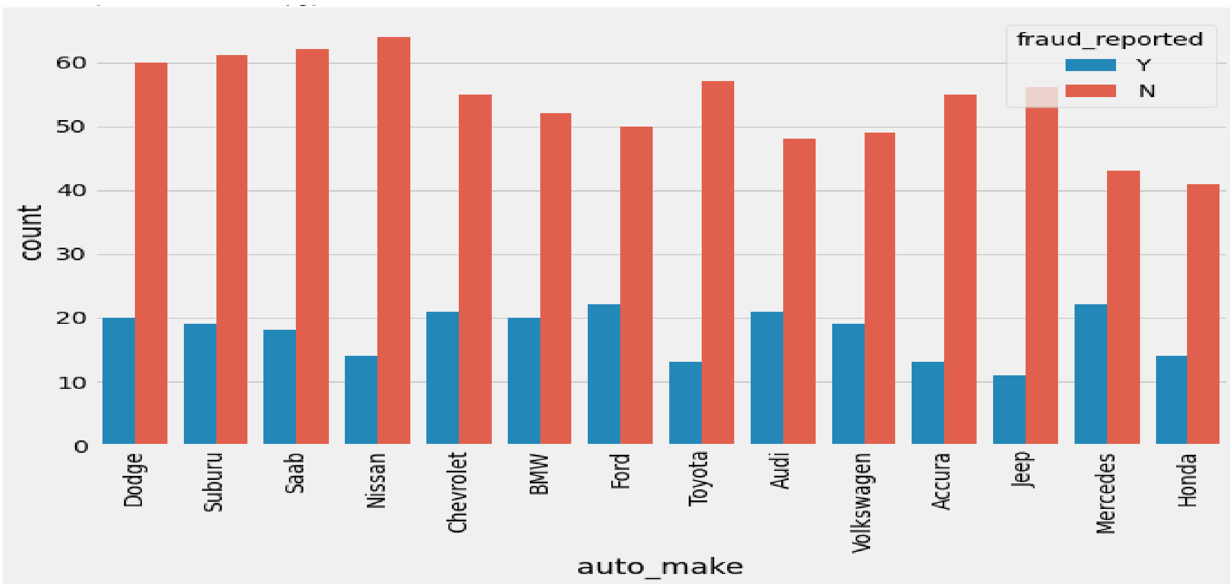
Most accidents happened with Dodge, Suburu and Saab, Nisaan.

Check which vehicle was guide in fraud cases.

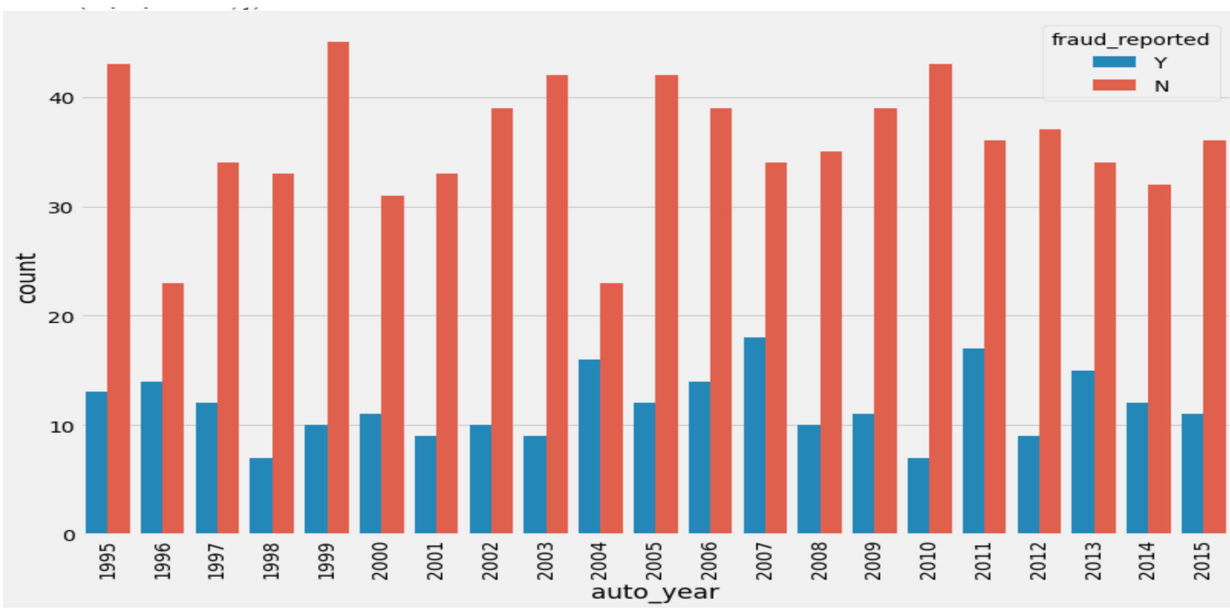
As per below visualization,

Fraud cases were more with Ford car and Mercedes car,

Audi , Chevrolet, BMW, Dodge, Suburu are also had more fraud cases. These are expensive car’s therefore vehicle owner did fraud to claim/bear expenses for their vehicle.

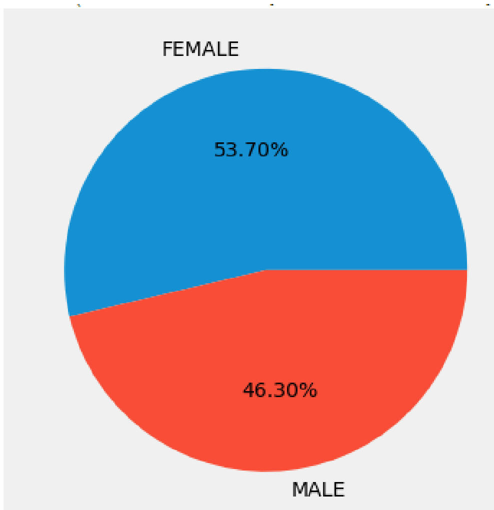


Auto_year vs fraud



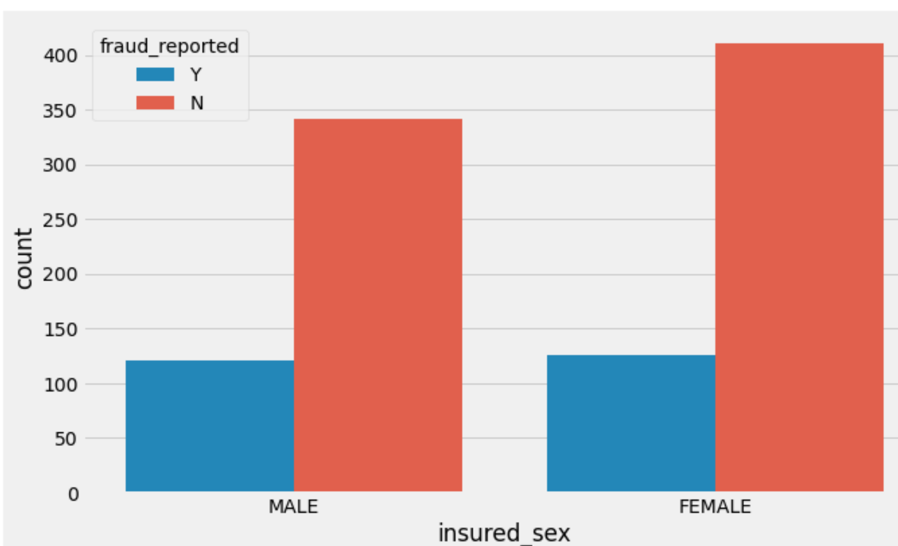
Since 1995 to 2015 registered vehicles data we have among which most fraud cases came from 2004, 2007 and 2011 registered vehicles.

Insured Person data



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

53.7% are Male while 46.3% are female insurer.

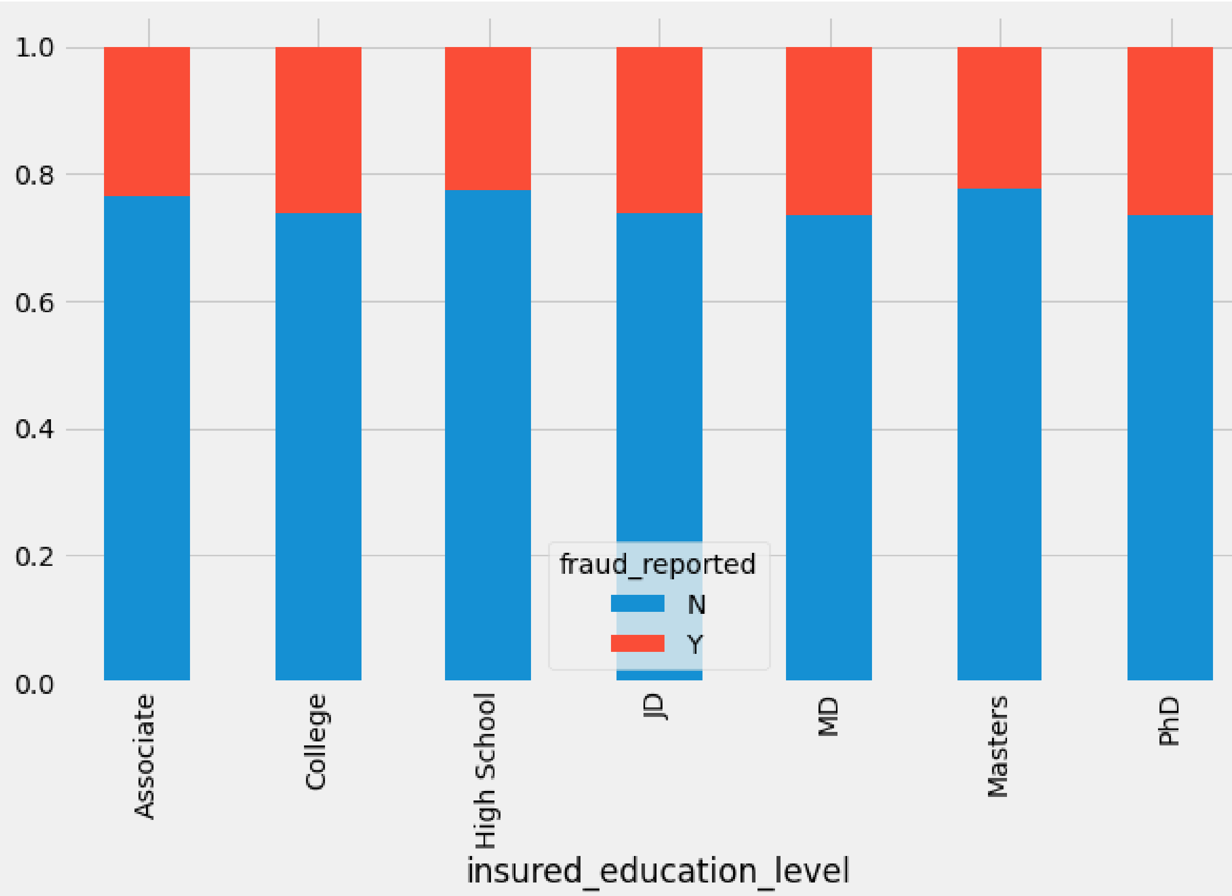


Among which chances are same for fraud case. Means fraud happening doesn't depends on insurer sex.

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'>

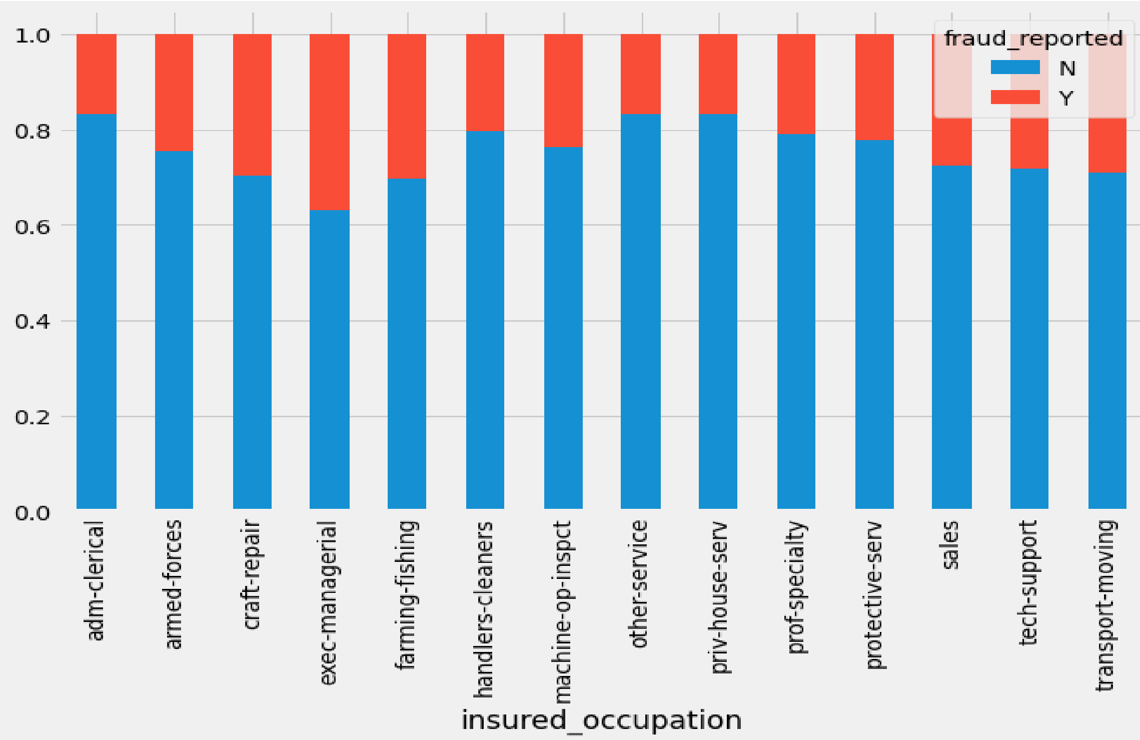


Almost chances are same for all educated insured. But specifically, College, JD, MD, PhD have done more frauds

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'>
```



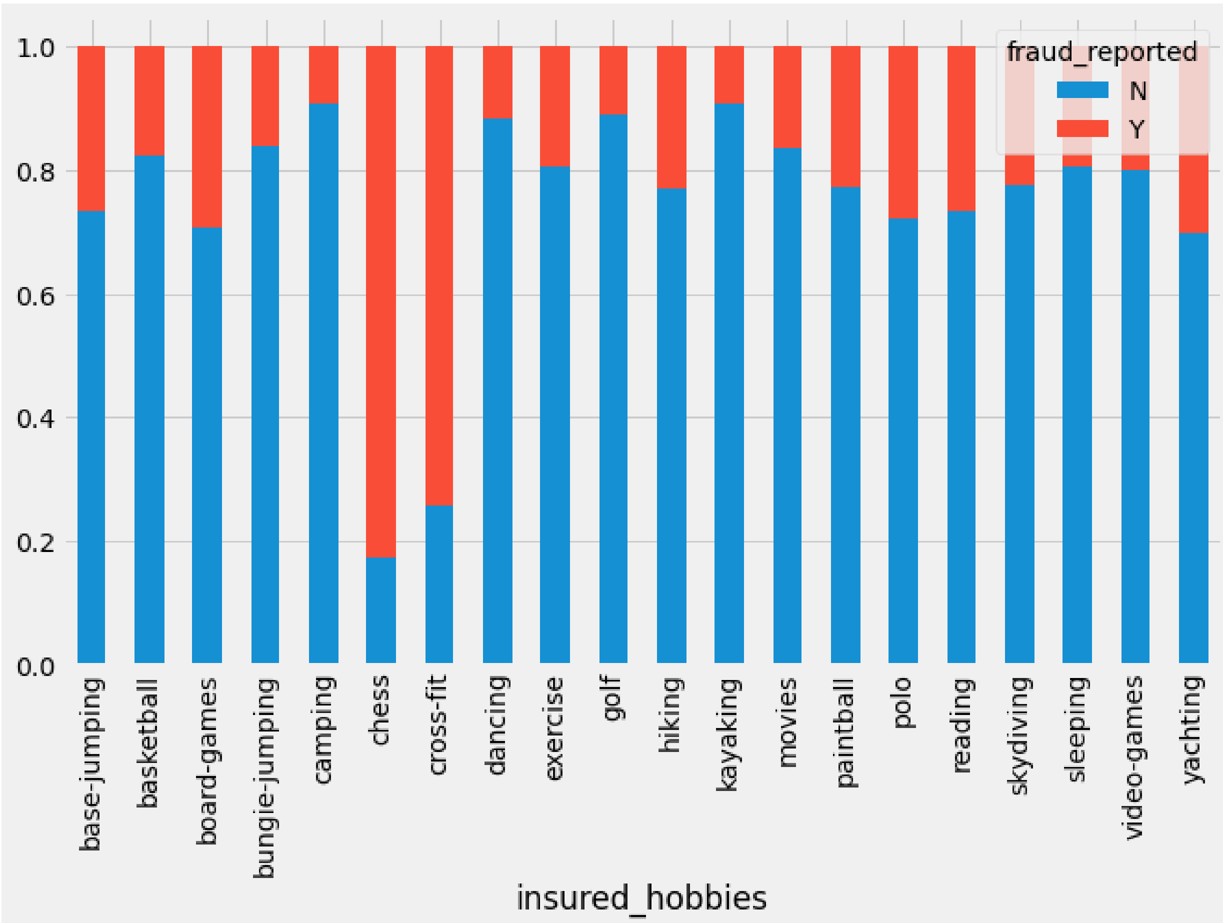
insured_hobbies vs fraud

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'>
```

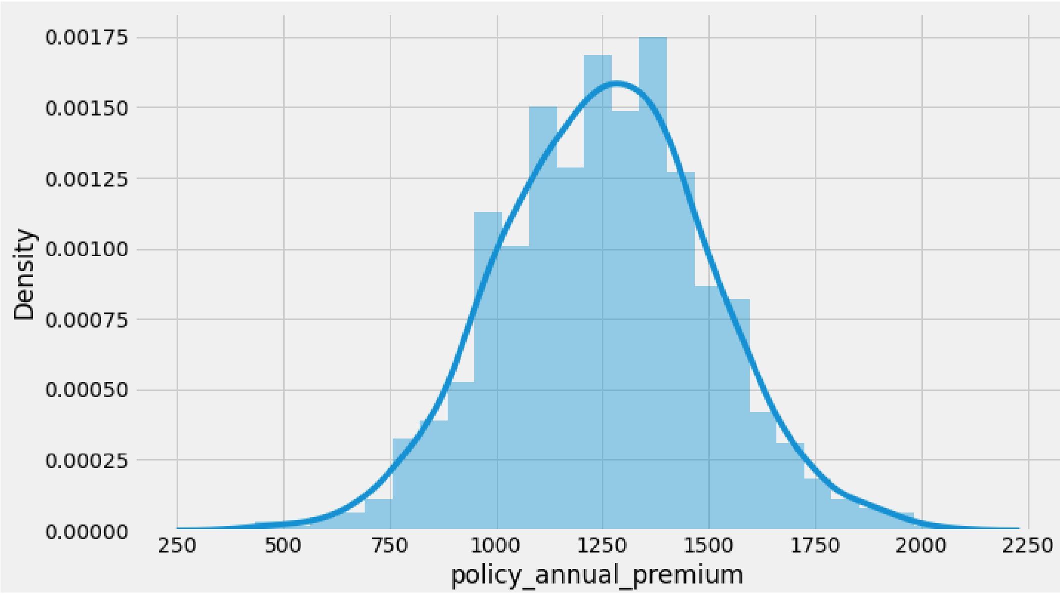


Policy_annual_premium

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

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

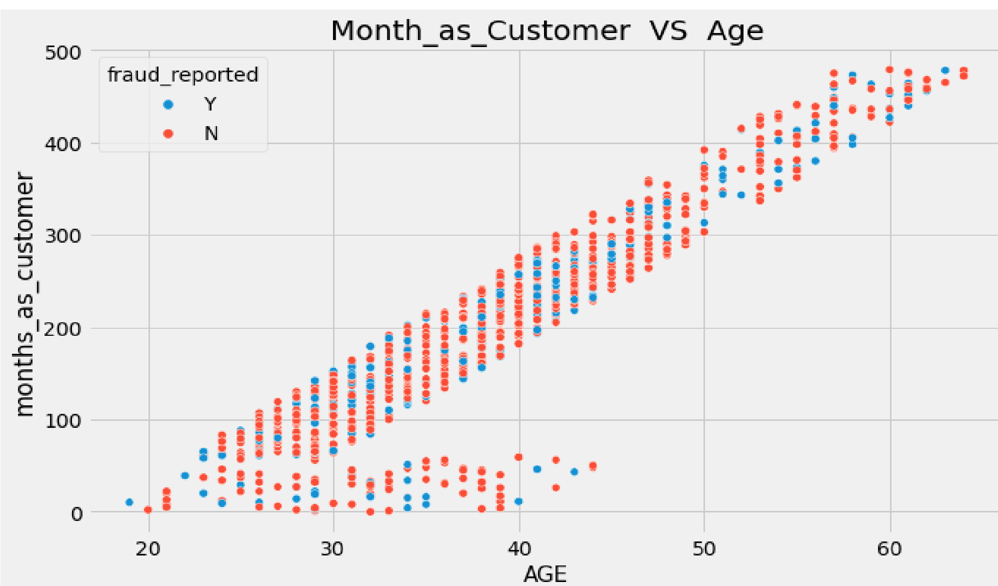
```
<AxesSubplot:xlabel='policy_annual_premium', ylabel='Density'>
```



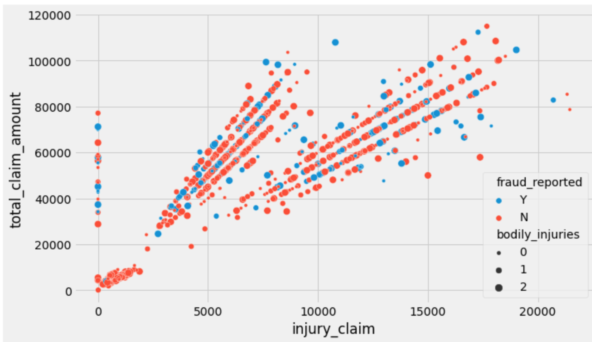
Month_as_customer vs Age

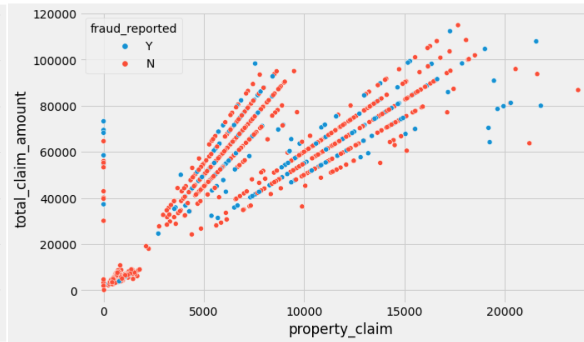
```
sns.scatterplot('age','months_as_customer',hue='fraud_reported',data=df)
plt.title('Month_as_Customer  VS  Age')
plt.xlabel('AGE',fontsize=15)
```

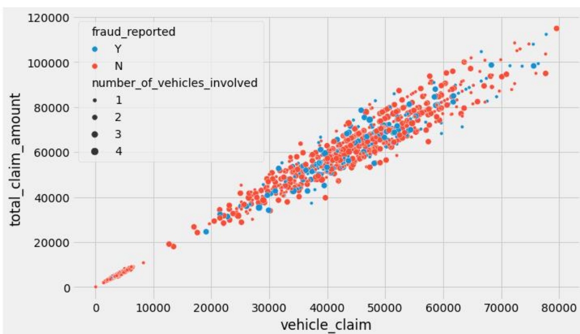
```
Text(0.5, 0, 'AGE')
```



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.

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.

```
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

Incident_hour_of_the_day

```
# This should be treated like categorical column

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

```
array([ 5,  8,  7, 20, 19,  0, 23, 21, 14, 22,  9, 12, 15,  6, 16,  4, 10,
        1, 17,  3, 11, 13, 18,  2], dtype=int64)
```

```
bins=[-1,5,11,16,20,24]
name=['night','Morning','afternoon','evening','midnight']
df['incident_period_of_the_day']= pd.cut(df['incident_hour_of_the_day'],bins,labels=name)
```

```
df[['incident_hour_of_the_day','incident_period_of_the_day']]
```

	incident_hour_of_the_day	incident_period_of_the_day
0	5	night
1	8	Morning
2	7	Morning
3	5	night
4	20	evening
...	...	...
995	20	evening
996	23	midnight
997	4	night
998	2	night
999	6	Morning

1000 rows × 2 columns

We have converted incident hours into time of the day, like in early morning, morning, afternoon, evening, night

Auto_year

```
df.auto_year.value_counts()
```

```
auto_year is the year of vehicle, it is imp factor to tell the age of car which decides the premium, cover amount and fraud case

All Incidents happened in 2015, so calculate Car age at the time of accident
```

```
df['Vehicle_Age']= 2015-df['auto_year']
df['Vehicle_Age']
```

```
0      11
1       8
2       8
3       1
4       6
       ..
995     9
996     0
997    19
998    17
999     8
Name: Vehicle_Age, Length: 1000, dtype: int64
```

Auto_year is a important feature, we have extracted the total_age of vehicle

Drop the features to avoid garbage information to the ML

```
# dropping unimportant columns

df = df.drop(columns = ['policy_number', 'policy_csl','insured_zip','policy_bind_date', 'incident_date', 'incident_location'
                        'auto_year'])

df.head(2)
```

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.

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

```
catg_features
```

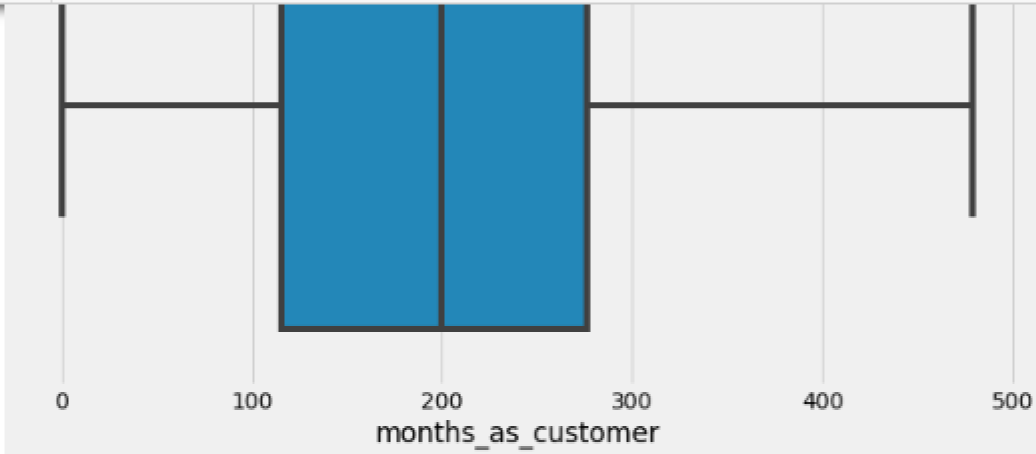
```
['policy_state',
 'umbrella_limit',
 'insured_sex',
 'insured_education_level',
 'insured_occupation',
 'insured_hobbies',
 'insured_relationship',
 'incident_type',
 'collision_type',
 'incident_severity',
 'authorities_contacted',
 'incident_state',
 'incident_city',
 'property_damage',
 'police_report_available',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

```
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']
```

Drawing Boxplot with continuous features-

```
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.

<u>Techniques like:</u>

- 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.

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.

```python
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
```

```python
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, calculating 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

```python
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

```python
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
```

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.

```
ordinal=['umbrella_limit','insured_education_level','insured_occupation']
```

```
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

for i in ordinal:
    X[i]=le.fit_transform(X[i])
```

Remaining categorical are Nominal

```
X=pd.get_dummies(X,drop_first=True)
```

```
X.shape  , Y.shape
```

```
((1000, 123), (1000,))
```

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 so we can proceed to ML 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

Modeling

Five different classifiers were used in this Project:

- ✓ Logistics regression
- ✓ Ridge Classifier
- ✓ Decision Tree Classifier
- ✓ SVC
- ✓ K-nearest neighbors
- ✓ Random Forest
- ✓ XGBoost
- ✓ SGD Classifier
- ✓ BaggingClassifier
- ✓ Adaboost classifier
- ✓ Gradient Boosting

```
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 ]
```

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.

Out[225]:

	Model	Accuracy	Cross Validation	Difference
0	LogisticRegression()	92.699115	85.737828	6.961287
1	RidgeClassifier()	92.477876	84.942025	7.535851
2	DecisionTreeClassifier()	84.955752	83.870102	1.084550
3	SVC()	87.610619	83.484852	4.125768
4	KNeighborsClassifier()	50.884956	55.844536	-4.959580
5	(DecisionTreeClassifier(max_features='auto', r...	90.044248	85.802733	3.843504
6	XGBClassifier(base_score=0.5, booster='gbtree'...	93.584071	86.862335	6.721736
7	SGDClassifier()	91.592920	84.670964	8.711576
8	(DecisionTreeClassifier(random_state=573854055...	90.044248	86.927680	3.249238
9	(DecisionTreeClassifier(max_depth=1, random_st...	88.716814	85.736067	2.980747
10	([DecisionTreeRegressor(criterion='friedman_ms...	91.592920	86.726365	4.933221

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.

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 performance 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

```
In [230]: GCV.best_estimator_
Out[230]: GradientBoostingClassifier(max_depth=4, n_estimators=5)

In [231]: GCV.best_params_
Out[231]: {'learning_rate': 0.1, 'max_depth': 4, 'n_estimators': 5}

In [232]: pred=GCV.best_estimator_.predict(x_test)
          accuracy_score(y_test,pred)
Out[232]: 0.9137168141592921
```

GCV.best_estimator_and

GCV.best_params

provides the best estimator for this cross-validation

Default parameters accuracy is 89.16

Hypertuned accuracy is 91.37

<u>Other Metrics to evaluate</u>

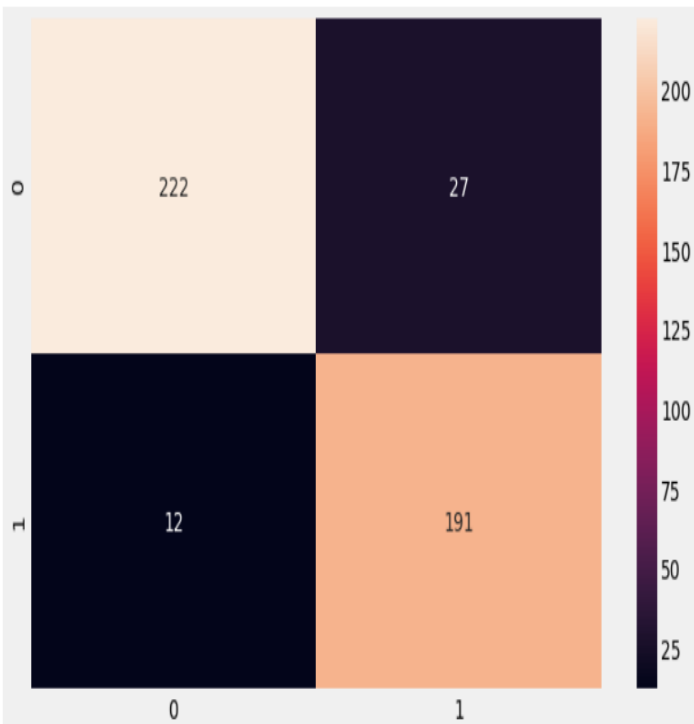
Confusion Matrix

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.

```
In [233]: from sklearn.metrics import confusion_matrix
          confusion_matrix(y_test,pred)
          sns.heatmap(confusion_matrix(y_test,pred),annot=True, fmt='d')
```

```
Out[233]: <AxesSubplot:>
```



ROC AUC Curve

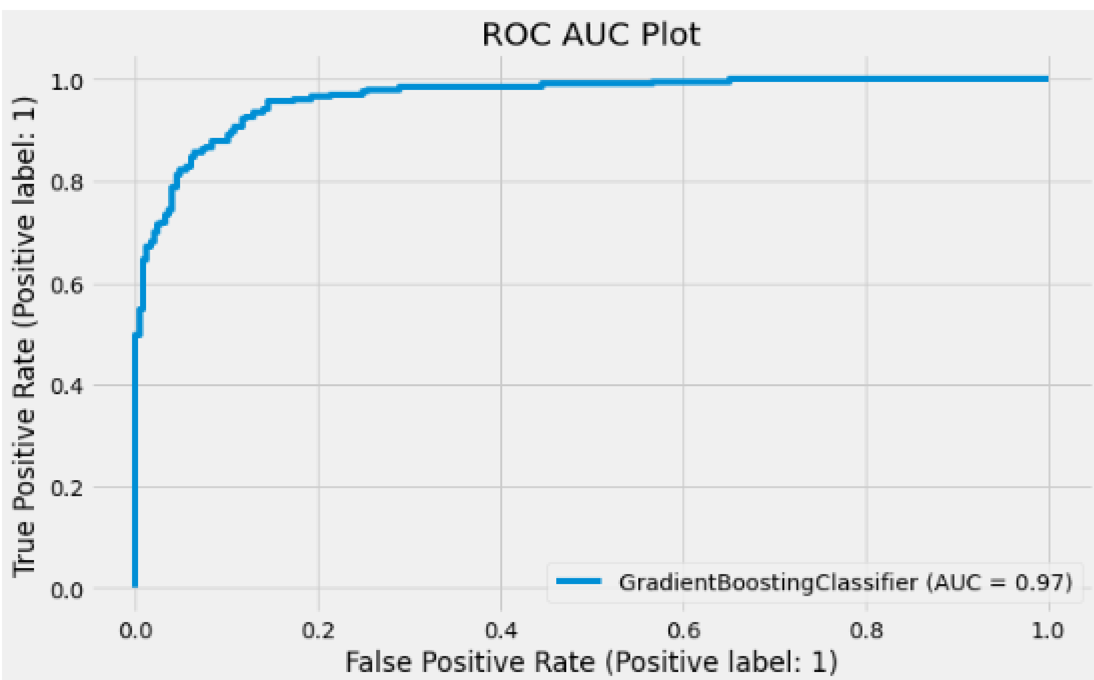
The ROC curve below summarizes how well our model is at balancing between the true positive rate (sensitivity) and the false positive rate(1-specificity). Ideally, we want to have a 100% true positive rate of predicting fraud and a 100% true negative rate of predicting non-frauds (or a 0% false-positive 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.

```
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**.**

The Study is not without limitation -

- Here the sample size is small. Statistical models are more stable when data sets are larger.

- It also generalize better if it takes a bigger portion of the actual population.(here the actual population size is small)

- We are also restricted to incidents between 2 months which may not be an accurate picture of the year.