INSURANCE CLAIM – FRAUD DETECTION BLOG /ARTICLE

Insurance Claim -Fraud Detection Status Prediction

Introduction:

Insurance Claim is made by the requestor to the policy provider. Insurance can be made for Home, Property, Land, Accident, Car, Health, Auto etc.

- For example, if we have Health Insurance, suddenly we get serious health issues, we need not to worry about money, whether we can afford the medical expenses or not. We can claim the health insurance by submitting the insurance claim form. All the medical expenses can be done in the health insurance claim.
- We can claim if our home is damaged from Earth quake or got into an accident. At that moment we can do insurance claim by submitting the form. So, that we don't give money from our pocket.
- In this article we see the Auto insurance claim, to predict where the insurance claim is fraudulent or not.

1)Problem Statement:

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

2)Data Analysis:

Data Analysis is the process of cleaning, transforming, pre-processing, modelling data to get a useful information and make a prediction.

Data Analysis can consist of Text, Statistical analysis, Description, Diagnostic, Predictive Analysis.

Data Analysis plays a very important role, before building the model, we do the analysis of data, we can replace any missing value present in data, we can remove unnecessary column from dataset. We can convert categorical data to numerical data using Label Encoder/Original Encoder. Data Analysis play an important role in Decision making, improve accuracy and help in scientific approach to give a good insight using a visualization technique.

Importing Necessary Libraries:

```
# Import Necessary Libraries

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from scipy.stats import zscore
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

Loading the dataset:

```
df=pd.read_csv('Automobile_insurance_fraud.csv')
```

```
pd.set_option('display.max_columns',None)
```

```
df.head()
```

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

```
df.tail()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insure

```
df.tail()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insure
995	3	38	941851	16-07-1991	OH	500/1000	1000	1310.80	0	431289	FEMALE	
996	285	41	186934	05-01-2014	IL	100/300	1000	1436.79	0	608177	FEMALE	
997	130	34	918516	17-02-2003	OH	250/500	500	1383.49	3000000	442797	FEMALE	
998	458	62	533940	18-11-2011	IL	500/1000	2000	1356.92	5000000	441714	MALE	
999	456	60	556080	11-11-1996	OH	250/500	1000	766.19	0	612260	FEMALE	

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

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

There are 1000 rows and 40 columns present in Insurance claim dataset. It contains both categorical and Numerical data.

Data Pre-processing and Data Cleaning:

In data pre-processing we check the statistical summary, data info, Unique value, column name, shape of data, value count.

In Data cleaning, we drop the unnecessary column, fill the missing value with mean/median if it is numerical data. If it is categorical data, we can replace with mode.

Statistical summary of data:

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00
_c39	0.0	NaN	NaN	NaN	NaN	NaN	NaN	NaN

Using the describe method, we check the statistics of data, their count value, mean, standard deviation, Minimum value, Maximum value, 1st Quartile, 2nd Quartile, 3th Quartile. Here mean value is greater than median.

```
df.dtypes
```

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

The data contains float, integer and Object data type.

```
plt.figure(figsize=(15,8))
sns.countplot(df['incident_date'])
plt.xticks(rotation=90)
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
        34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46,
        51, 52, 53, 54, 55, 56, 57, 58, 59]),
 [Text(0, 0, '25-01-2015'),
  Text(1, 0, '21-01-2015'),
  Text(2, 0, '22-02-2015'),
  Text(3, 0, '10-01-2015'),
  Text(4, 0, '17-02-2015'),
  Text(5, 0, '02-01-2015'),
  Text(6, 0, '13-01-2015'),
  Text(7, 0, '27-02-2015'),
  Text(8, 0, '30-01-2015'),
  Text(9, 0, '05-01-2015'),
  Text(10, 0, '06-01-2015'),
  Text(11, 0, '15-02-2015'),
  Text(12, 0, '22-01-2015'),
  Text(13, 0, '08-01-2015'),
  Text(14, 0, '15-01-2015'),
  Text(15, 0, '29-01-2015'),
  Text(16, 0, '19-01-2015'),
  Text(17, 0, '01-01-2015'),
  Text(18, 0, '10-02-2015'),
  Text(19, 0, '11-01-2015'),
  Text(20, 0, '24-02-2015'),
  Text(21, 0, '09-01-2015'),
  Text(22, 0, '28-01-2015'),
  Text(23, 0, '07-01-2015'),
```

In the Incident Date column, it is considered as Object data type. Which is not correct. So, we have converted date time data type and we have separated into Day, Month and Year.

policy_bind_date

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

```
951
```

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

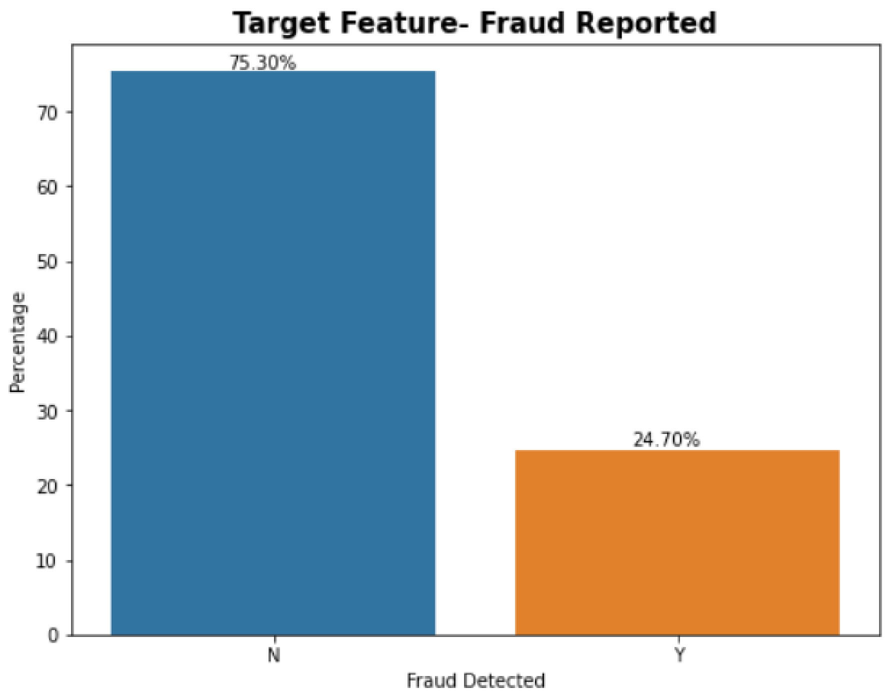
```
01-01-2006    3
28-04-1992    3
05-08-1992    3
14-12-1991    2
09-08-2004    2
             ..
03-06-2014    1
12-12-1998    1
18-02-1999    1
30-10-1997    1
11-11-1996    1
Name: policy_bind_date, Length: 951, dtype: int64
```

In Policy Bind date, the same way we have converted into datetime data type and separated into Day, Month and Year. Then we can drop the Policy bind date column.

```
# 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)
```

We can drop the unnecessary column, policy number, insured zip, incident location, etc.

```
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}%')
```



We can see that a greater number of people reported No for Fraud Claim. No-753. Yes-247. Still few people reported there a fraud claim.

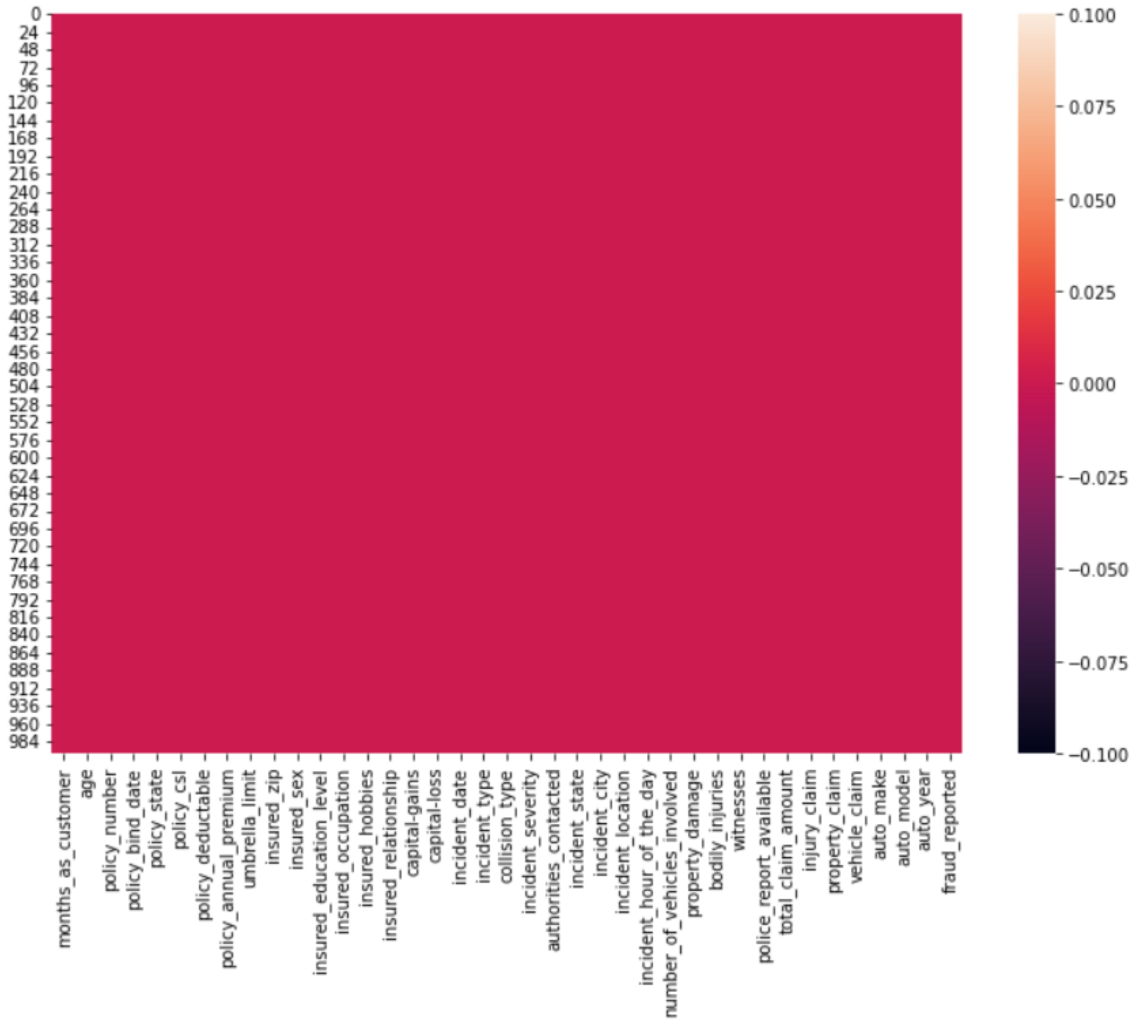
```
df.isnull().sum()
```

```
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
dtype: int64
```

We can see there is no null value present in data.

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

```
<AxesSubplot:>
```

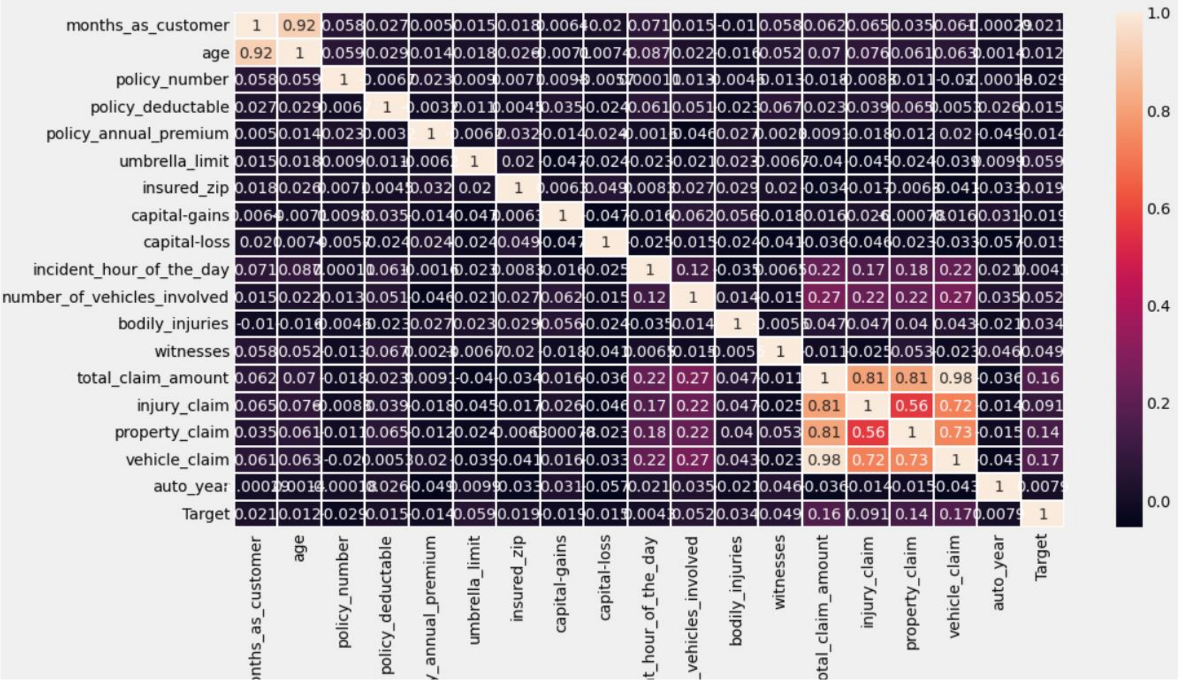


There is no null value present in the data. We have visualized using the heatmap.

Correlation:

```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),annot=True,linecolor='white',linewidths=.25)
```

```
<AxesSubplot:>
```



Visualizing the correlation using the heatmap. In total_claim_amount, injury_claim, property_claim, vehicle_claim there is a high correlation present, other 's columns are less corelated.

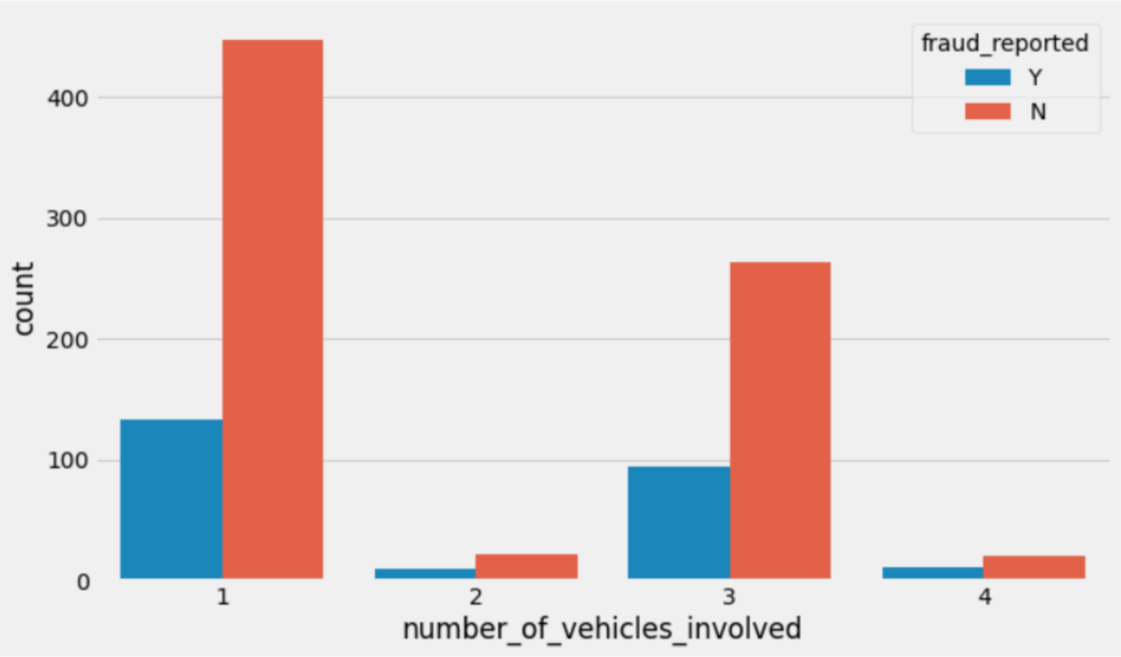
number_of_vehicles_involved

```
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'>
```



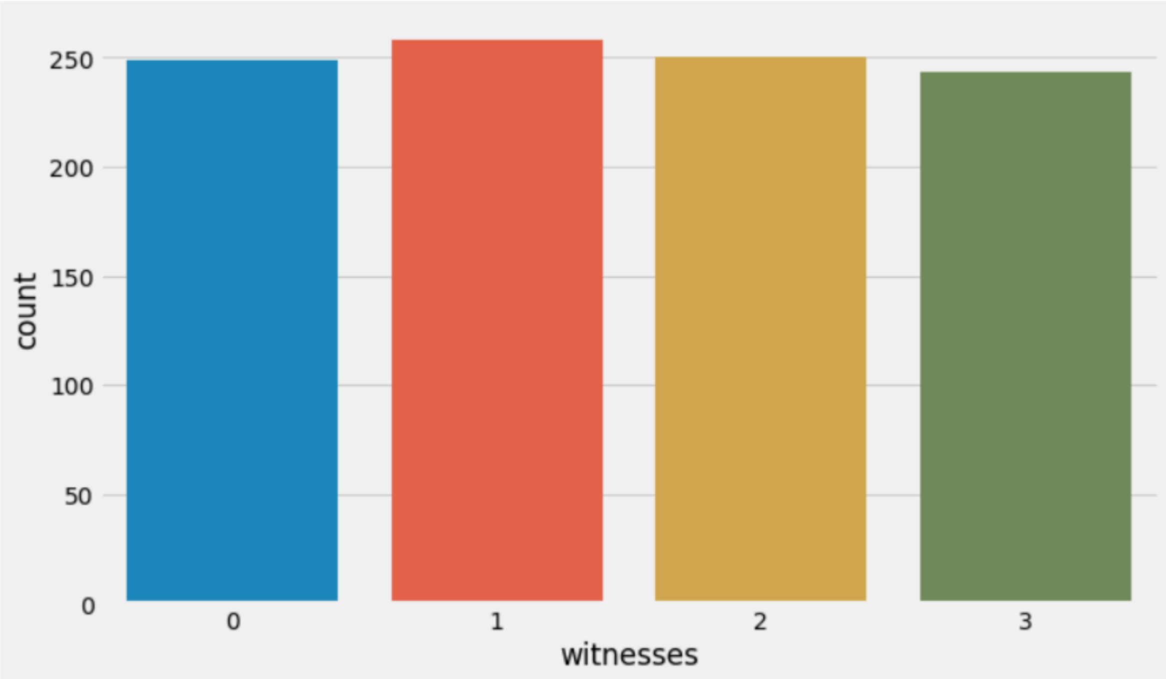
Using the count plot, we have checked the number of vehicles involved with fraud_reported. Those who have only 1 vehicle, the count is high that there is a no fraud claim. Those who are having more than 1 vehicle, there is high change of fraud claim.

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

```
array([2, 0, 3, 1], dtype=int64)
```

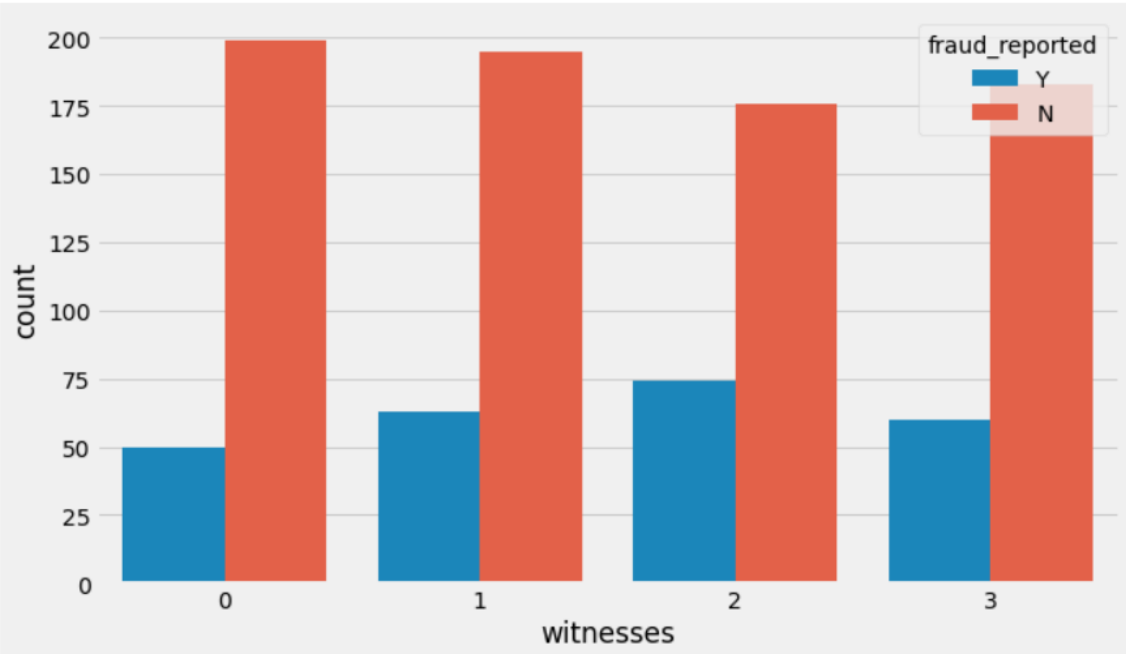
```
sns.countplot(df['witnesses'])
```

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




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

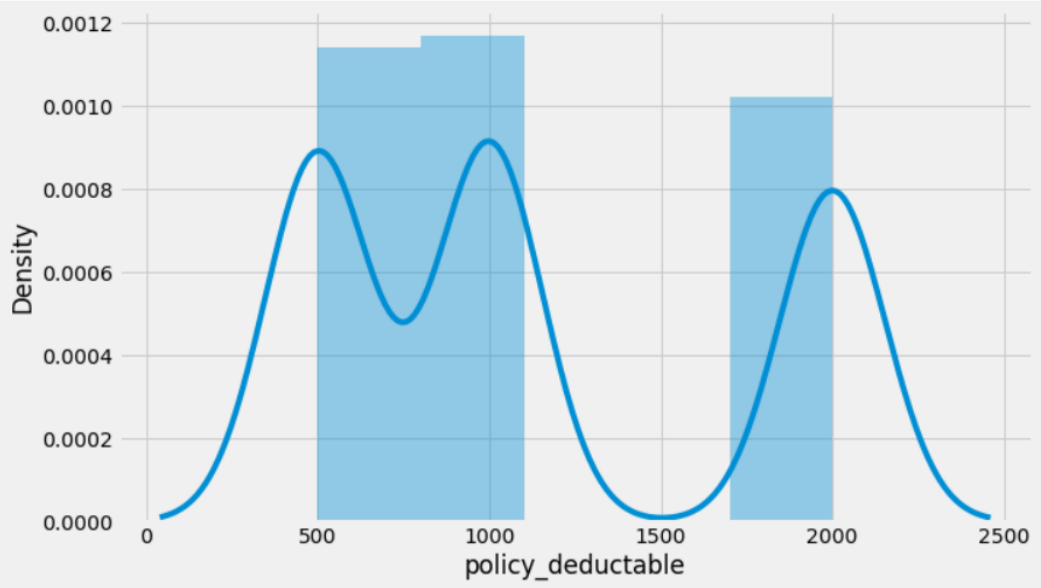
```
<AxesSubplot:xlabel='witnesses', ylabel='count'>
```



We have more than 1 witness reported the fraud claim. High count is, there is no fraud claim, few people has reported the fraud claim.

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

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




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

```
array([1000, 2000,  500], dtype=int64)
```

We have compared the Policy deductable and fraud reported. We see that policy deductable count 1000 has more than others.

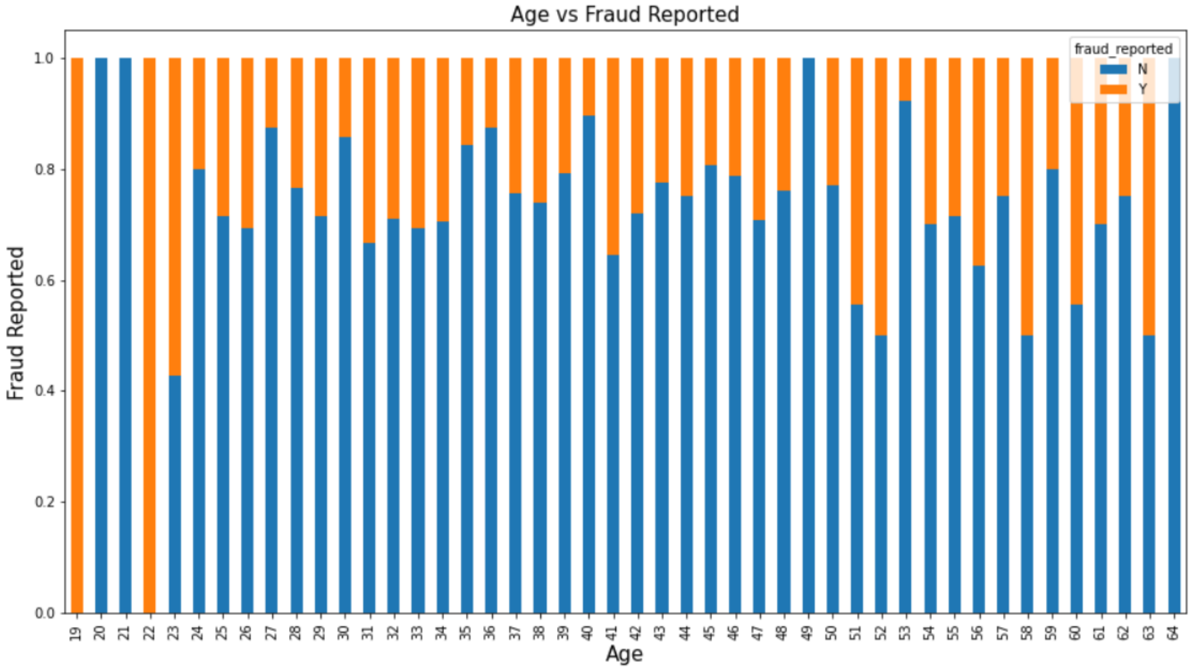
```
df['incident_hour_of_the_day']
```

```
0       5
1       8
2       7
3       5
4      20
       ..
995    20
996    23
997     4
998     2
999     6
Name: incident_hour_of_the_day, Length: 1000, dtype: int64
```

```
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)
```

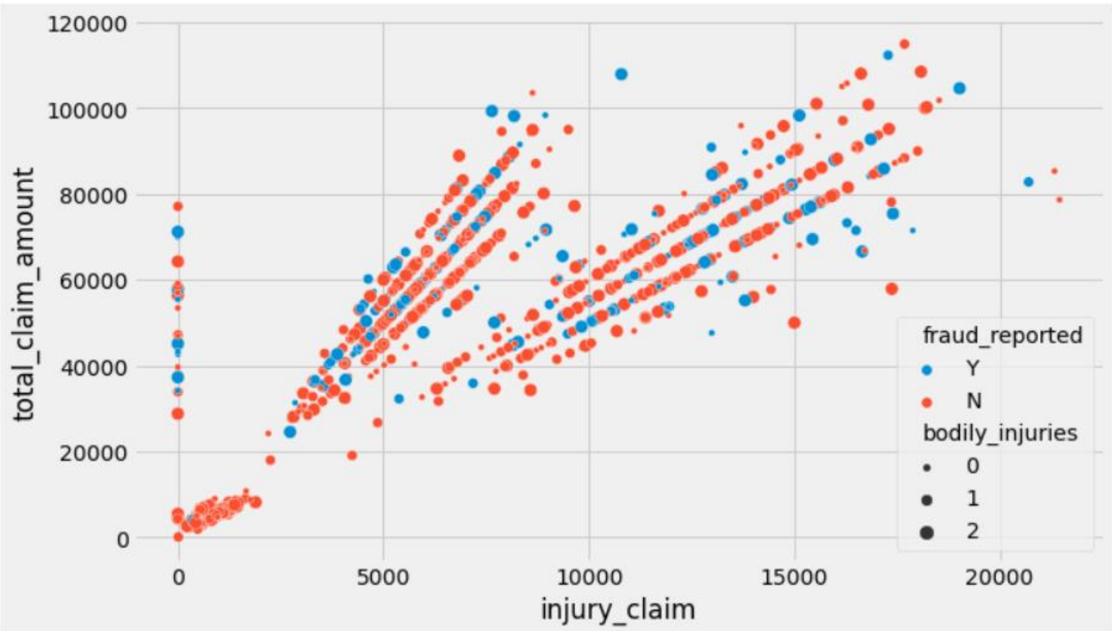
We have checked the incident hour of day; in which hour more fraud claim is happening. In the peak hour there is high fraud claim.



We compare the age with fraud reported, we clearly see that there is high number of frauds reported.

```
sns.scatterplot('injury_claim','total_claim_amount',hue='fraud_reported',size='bodily_injuries',data=df)
```

```
<AxesSubplot:xlabel='injury_claim', ylabel='total_claim_amount'>
```



In the injury claim between 5000-10000 there is high number of frauds reported.

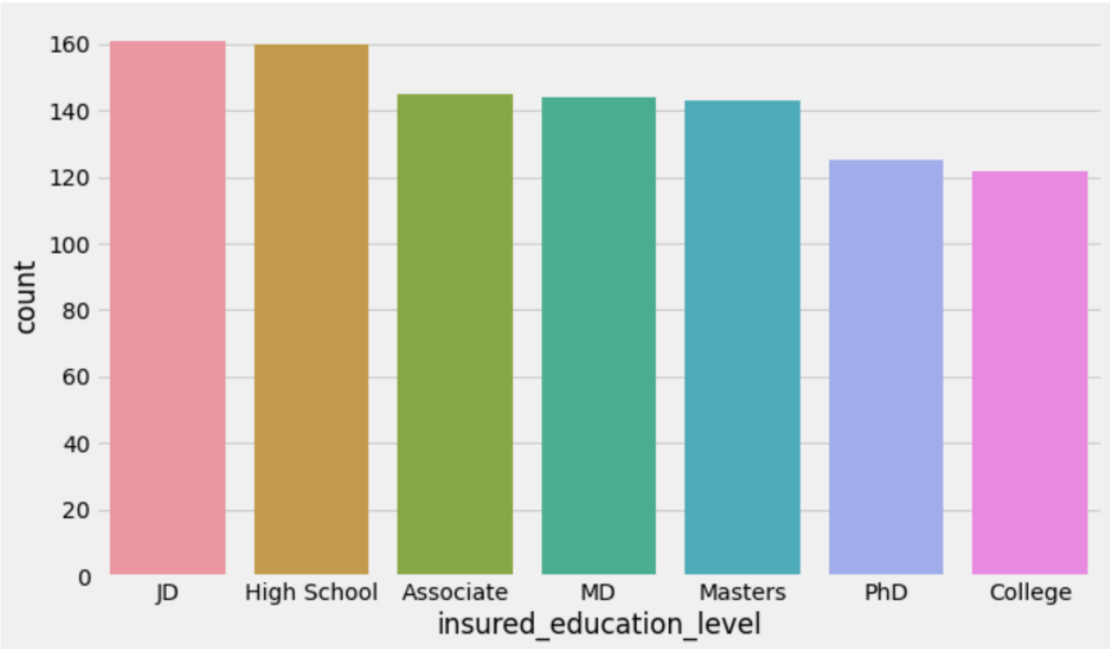
insured_education_level

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

```
array(['MD', 'PhD', 'Associate', 'Masters', 'High School', 'College',
       'JD'], dtype=object)
```

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

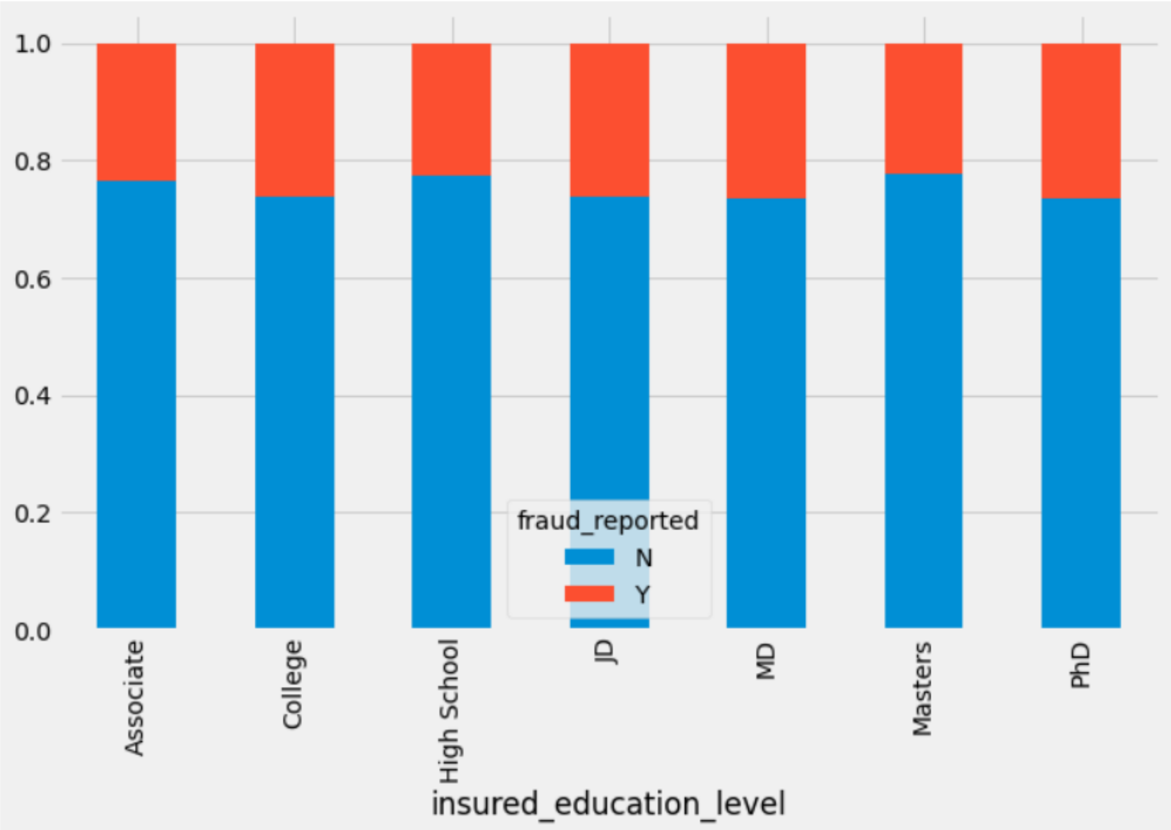
```
<AxesSubplot:xlabel='insured_education_level', ylabel='count'>
```



If we see the education level of insured person who has taken insurance from the company. Most of the insured person has completed JD and High School.

```
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'>
```



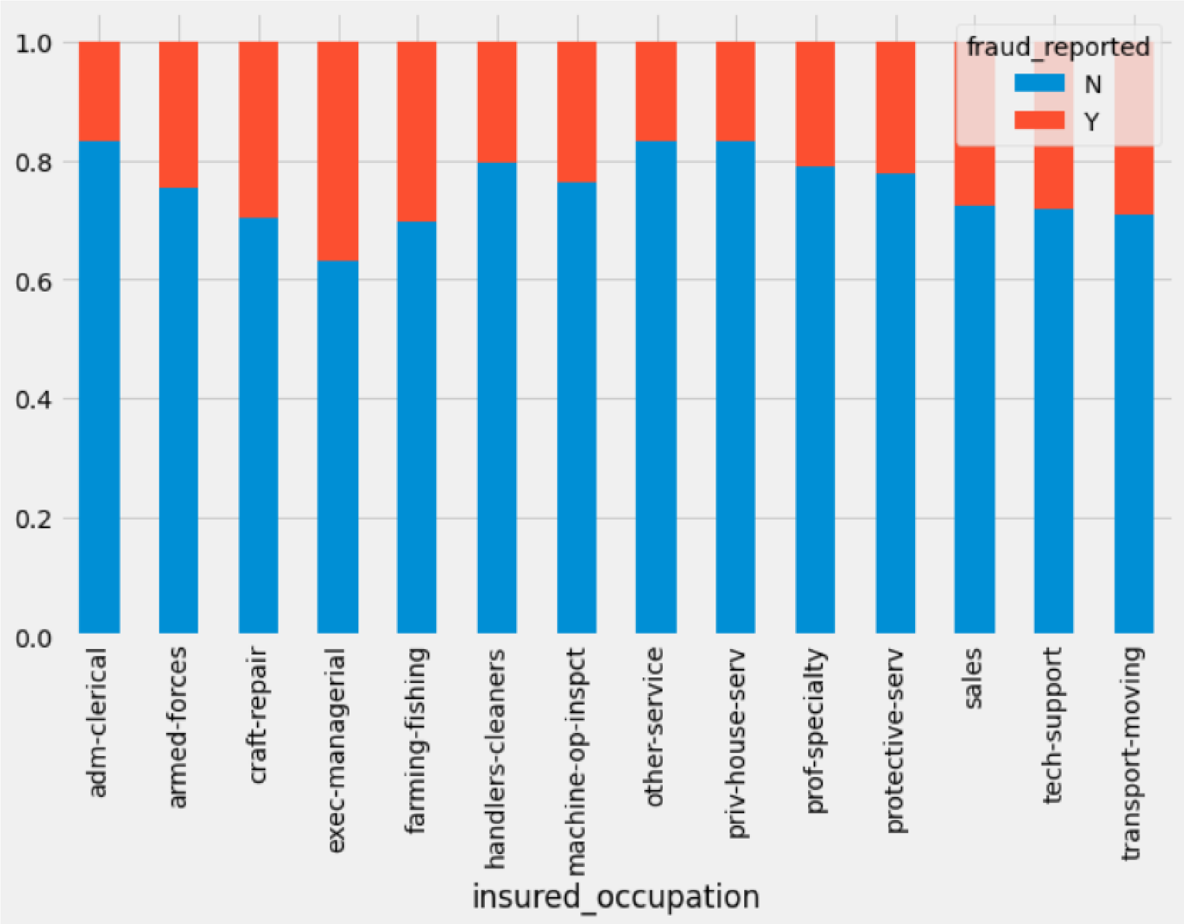
If we compare the education level with fraud reported, those have completed JD has higher chance of fraud claim.

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

```
array(['craft-repair', 'machine-op-inspct', 'sales', 'armed-forces',
       'tech-support', 'prof-specialty', 'other-service',
       'priv-house-serv', 'exec-managerial', 'protective-serv',
       'transport-moving', 'handlers-cleaners', 'adm-clerical',
       'farming-fishing'], dtype=object)
```

```
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'>



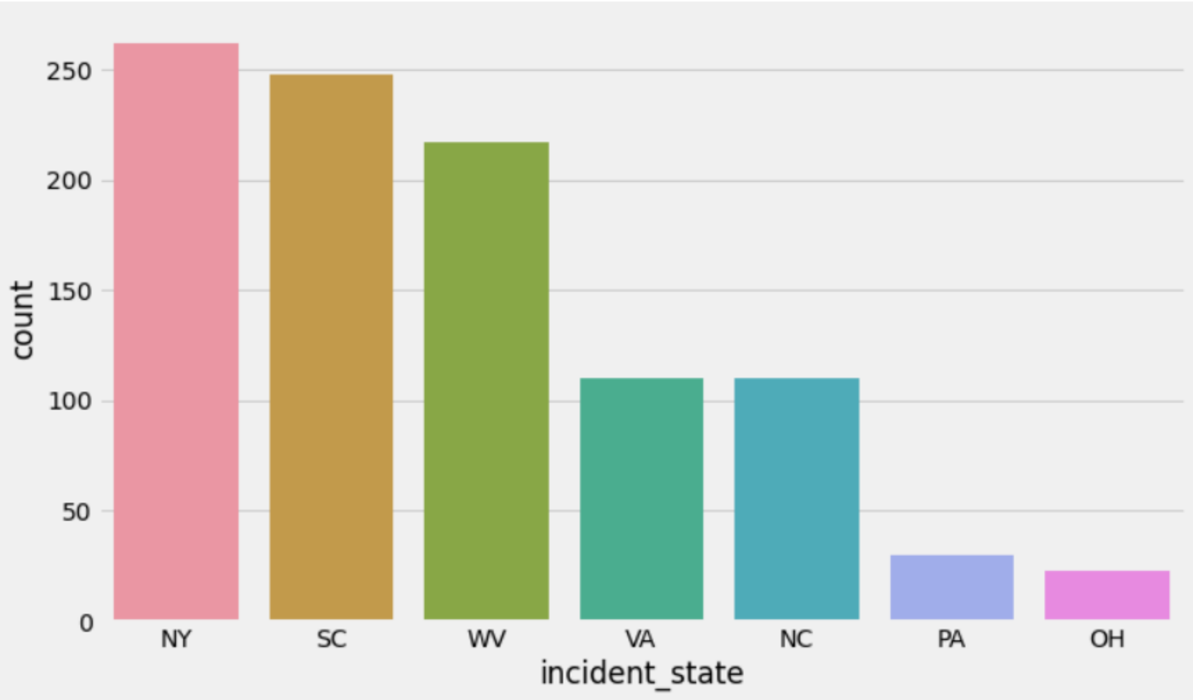
Majority of insured person occupation is Machine-on-inspect, when we checked the insured person occupation and fraud reported. The diagram shows the person who is in executive-managerial has high chance of fraud claim.

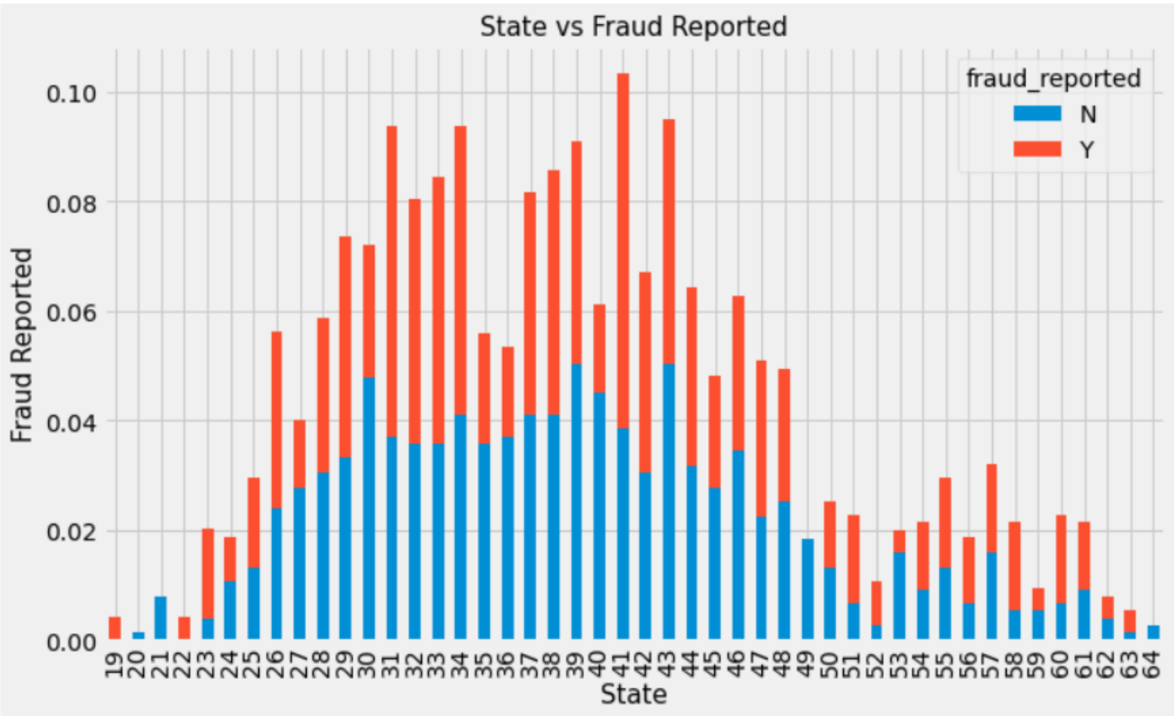
```
df['incident_state'].nunique()
```

```
7
```

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

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





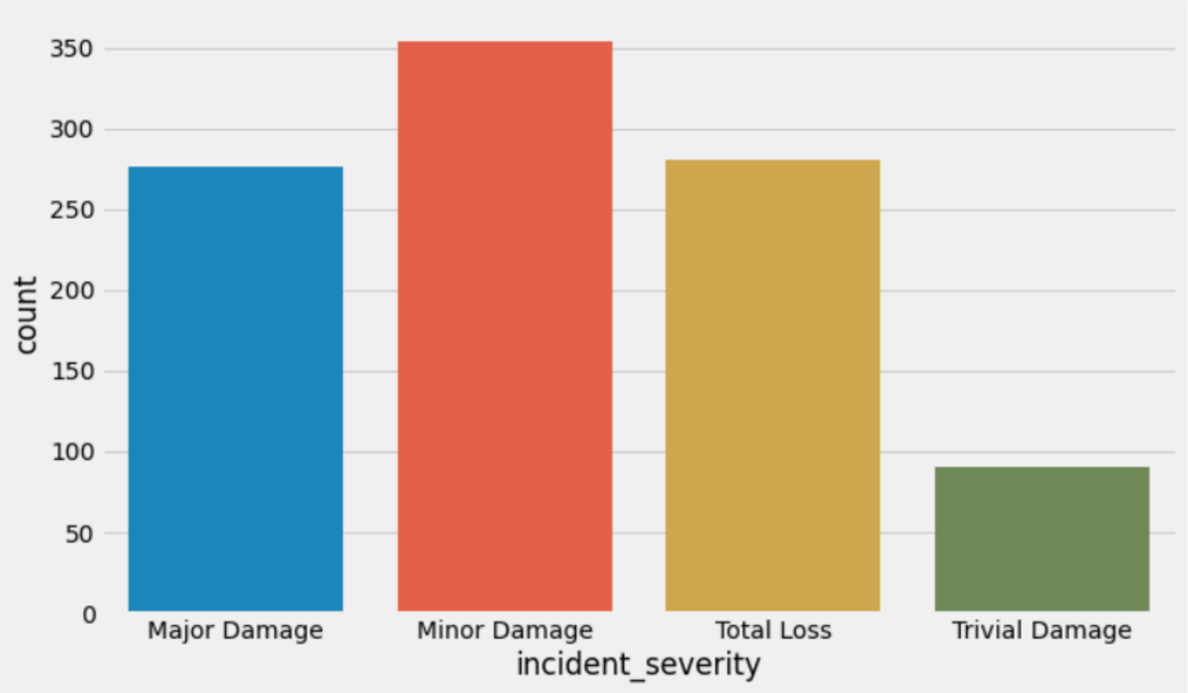
Majority of insured person belongs to NY state. The person who belongs to SC State there is high chance of fraud claim. The PA state have very less fraud claim, that is because there are only few insured people belongs to PA state.

```
df['incident_severity'].value_counts(normalize=True)
```

```
Minor Damage      0.354
Total Loss        0.280
Major Damage      0.276
Trivial Damage    0.090
Name: incident_severity, dtype: float64
```

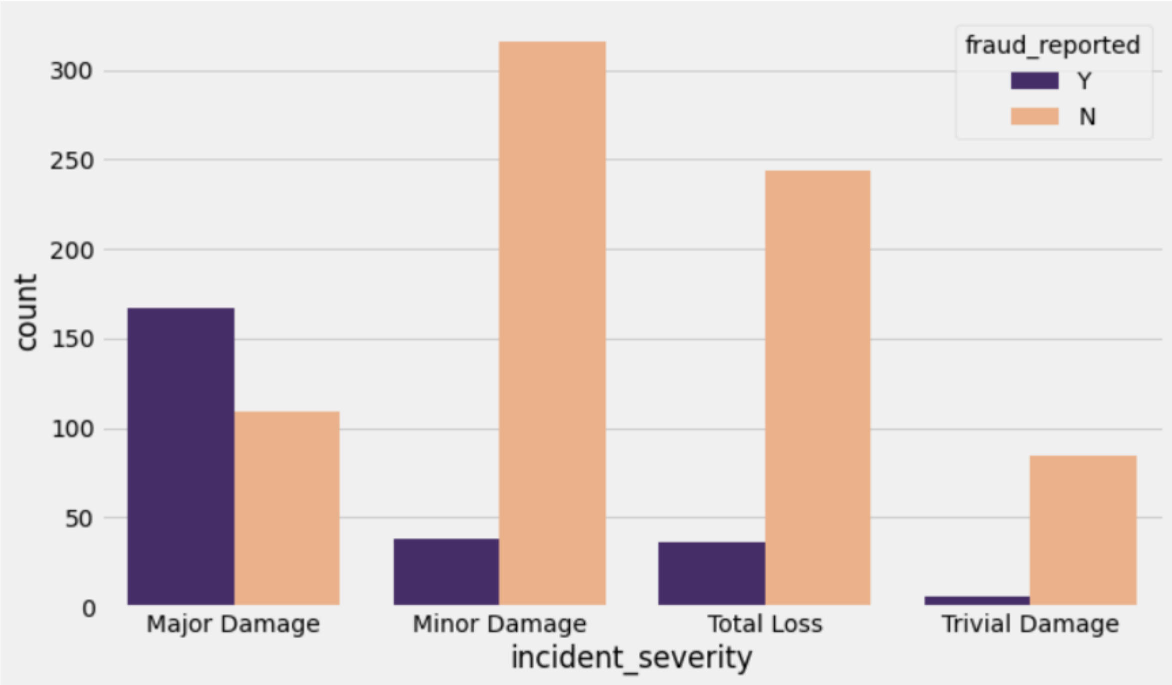
```
sns.countplot(df['incident_severity'])
```

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



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

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

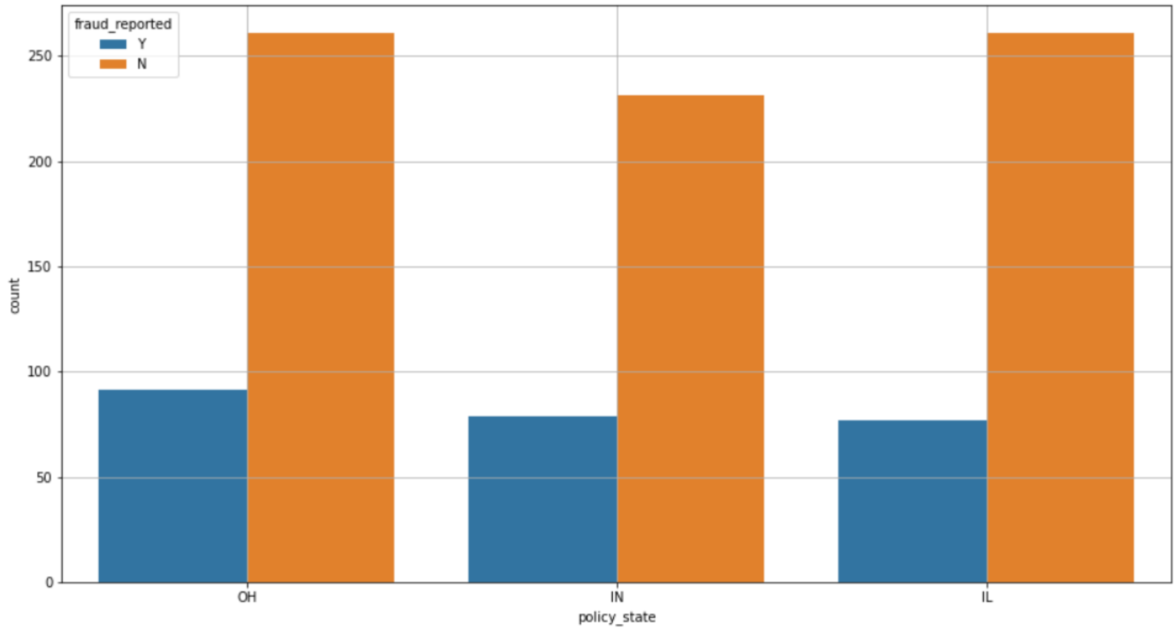


There is a high count in Major Damage and also there is high chance of fraud reported in the Major Damage category. If there is light damage happen to vehicle, they claim huge amount from insurance company for Major Damage. Sometimes they even damage their own vehicle and submitted the insurance claim form. For that purpose, we can build the Machine learning model to predict which is fraudulent claim or not.

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

3

```
sns.countplot(df['policy_state'],hue=df['fraud_reported'])
plt.grid()
```



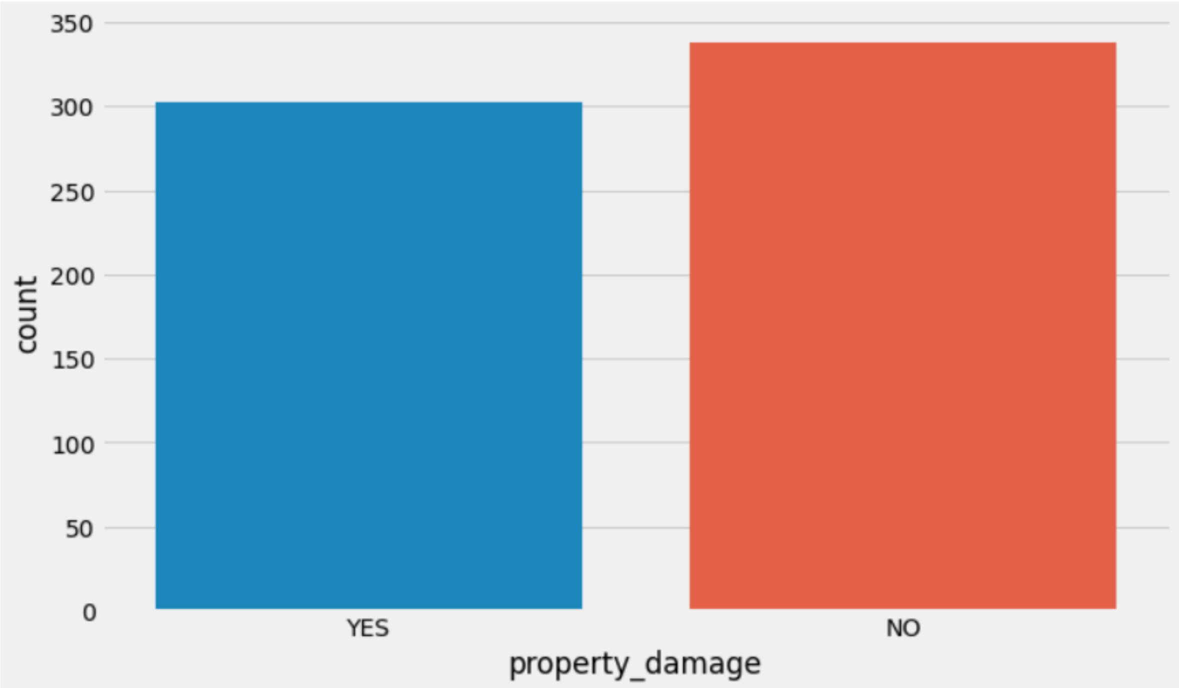
There is almost equal fraud reported in all the policy state. There is a little bit high fraud reported in OH policy state.

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

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

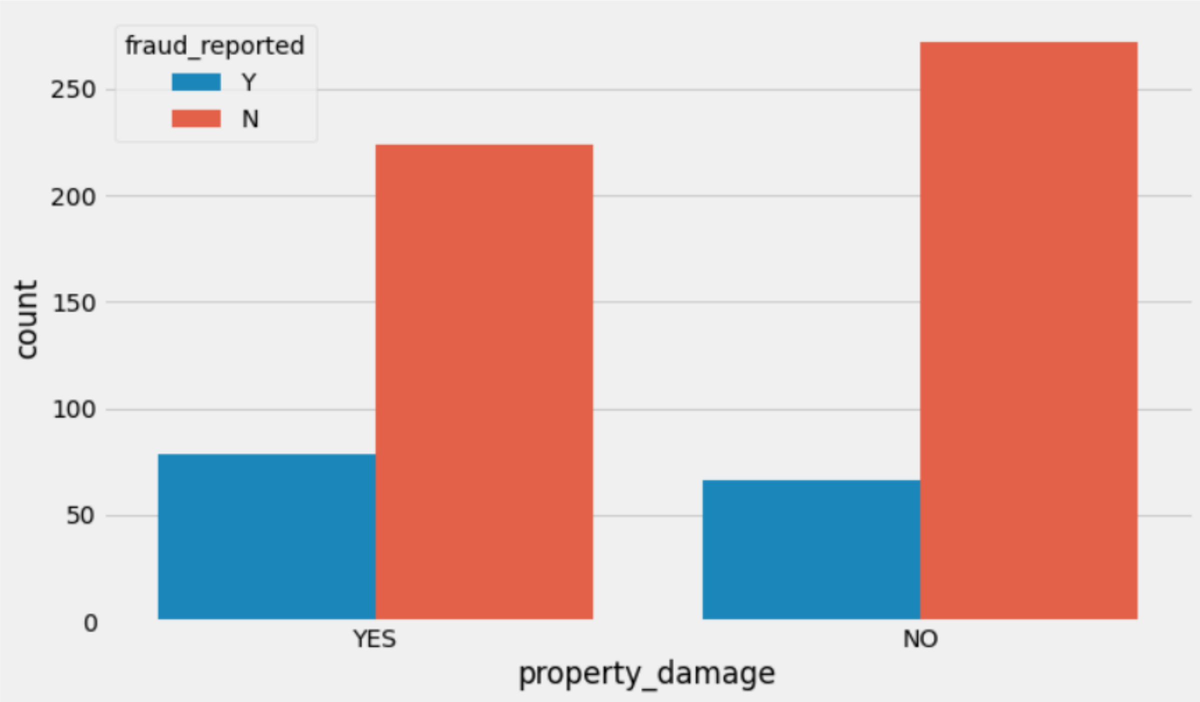
```
sns.countplot(df['property_damage'])
```

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



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

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




```
df.groupby('property_damage')['fraud_reported'].value_counts()
```

```
property_damage  fraud_reported
NO               N                 272
                 Y                  66
YES              N                 224
                 Y                  78
Name: fraud_reported, dtype: int64
```

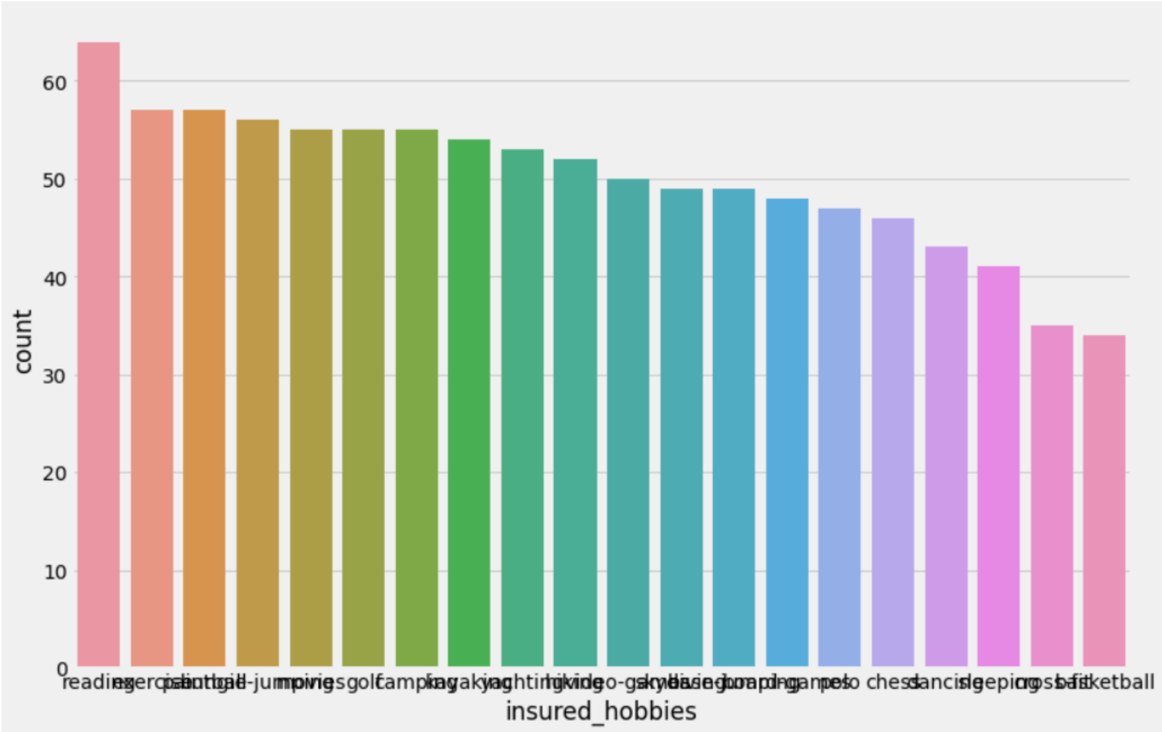
In the Property damage column, there are many rows that don’t have any information. 301 Insured persons has reported the property damage and 338 insured persons has not reported any property damage.

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

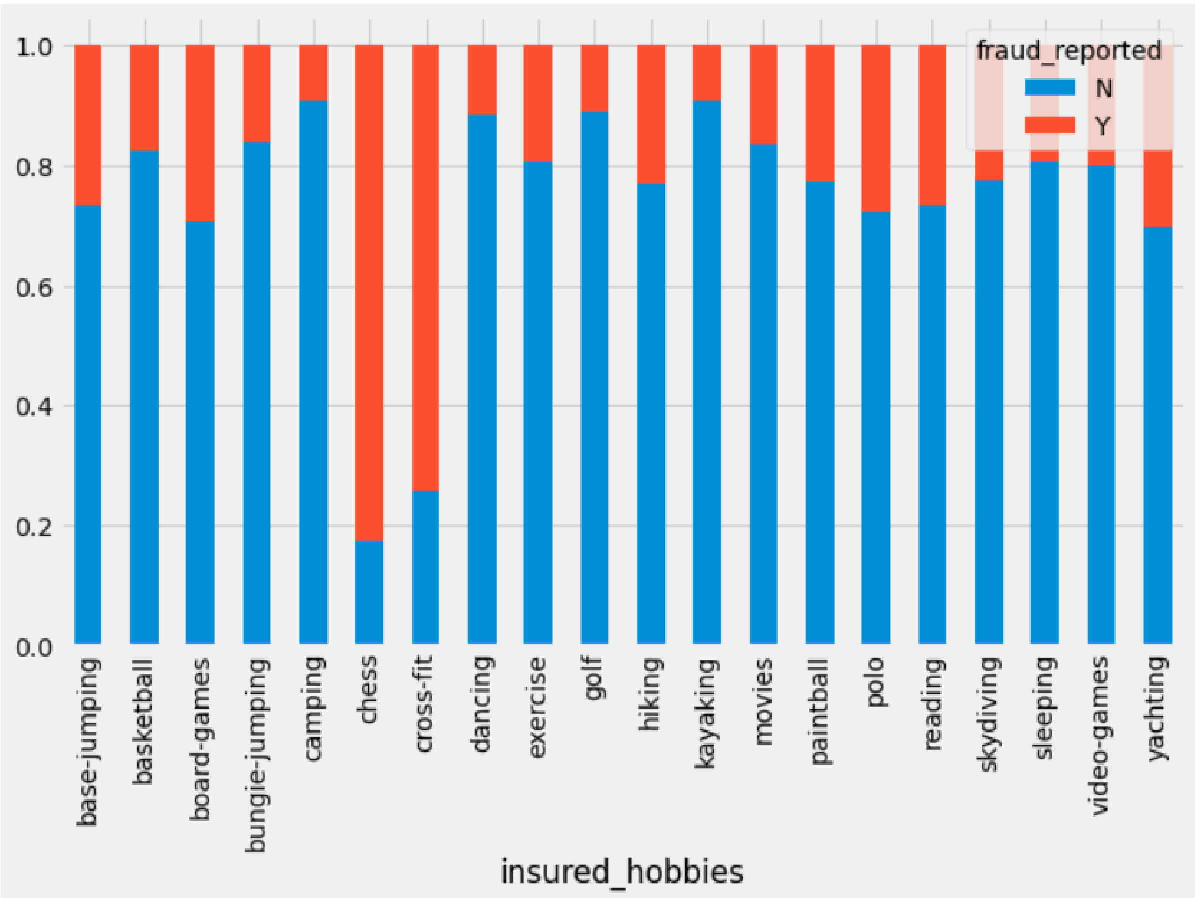
```
array(['sleeping', 'reading', 'board-games', 'bungie-jumping',
       'base-jumping', 'golf', 'camping', 'dancing', 'skydiving',
       'movies', 'hiking', 'yachting', 'paintball', 'chess', 'kayaking',
       'polo', 'basketball', 'video-games', 'cross-fit', 'exercise'],
      dtype=object)
```

```
plt.figure(figsize=(12,8))
sns.countplot(df['insured_hobbies'],order=df['insured_hobbies'].value_counts().index)
```

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



<AxesSubplot:xlabel='insured_hobbies'>



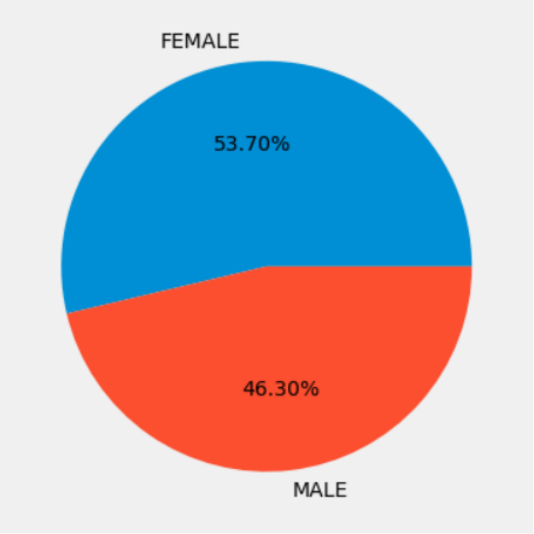
Let’s see what are the hobbies insured person have. Majority of the insured person have the hobbies of reading books. Then comes next Paintball and exercise. Only few insured persons are interested in playing basketball. If we checked the fraud reported with Hobbies. We see in the diagram, those who are interested in playing chess there is a high chance of fraud reported. Obviously, chess is mind game, we have to think each move carefully. So, the fraud claims also need to do carefully, only those people who strategically and logically strong they can do the fraud claim.

```python
df['insured_sex'].unique()
```

```
array(['MALE', 'FEMALE'], dtype=object)
```

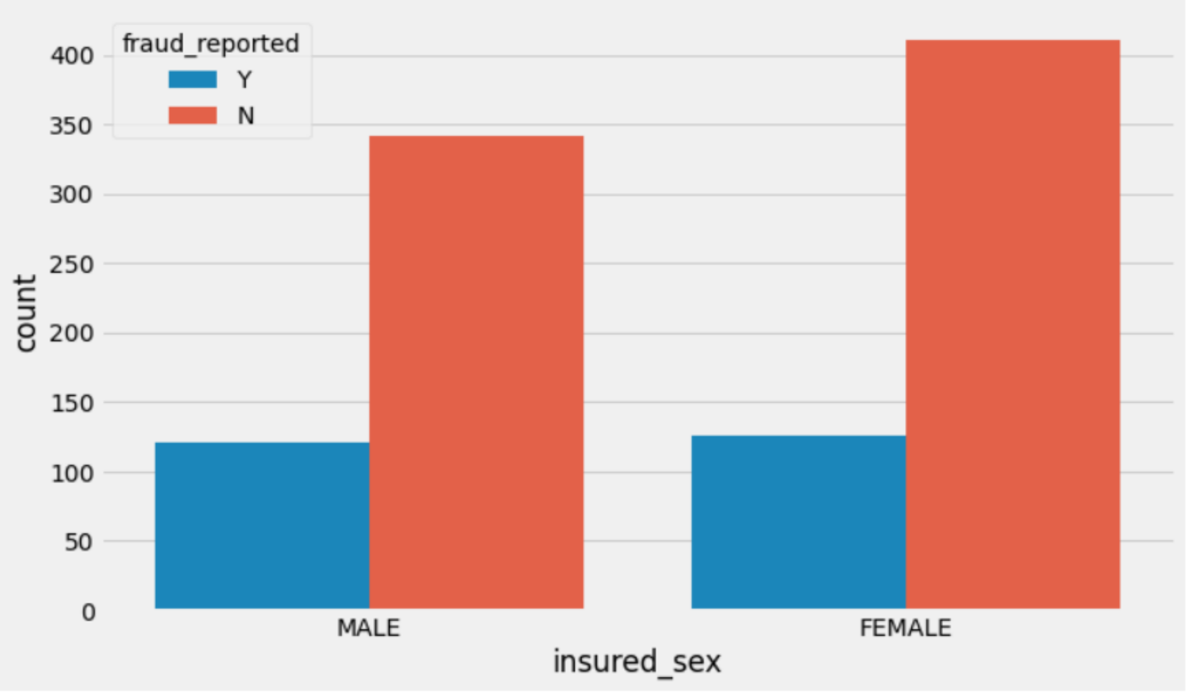
```python
plt.pie(df['insured_sex'].value_counts().values,labels=df['insured_sex'].value_counts().index,autopct='%1.2f%%')
```

```
([<matplotlib.patches.Wedge at 0x20c6218ec10>,
  <matplotlib.patches.Wedge at 0x20c6219b430>],
 [Text(-0.12757508092656847, 1.0925770447554624, 'FEMALE'),
  Text(0.12757508092656858, -1.0925770447554624, 'MALE')],
 [Text(-0.06958640777812825, 0.5959511153211613, '53.70%'),
  Text(0.0695864077781283, -0.5959511153211613, '46.30%')])
```




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

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

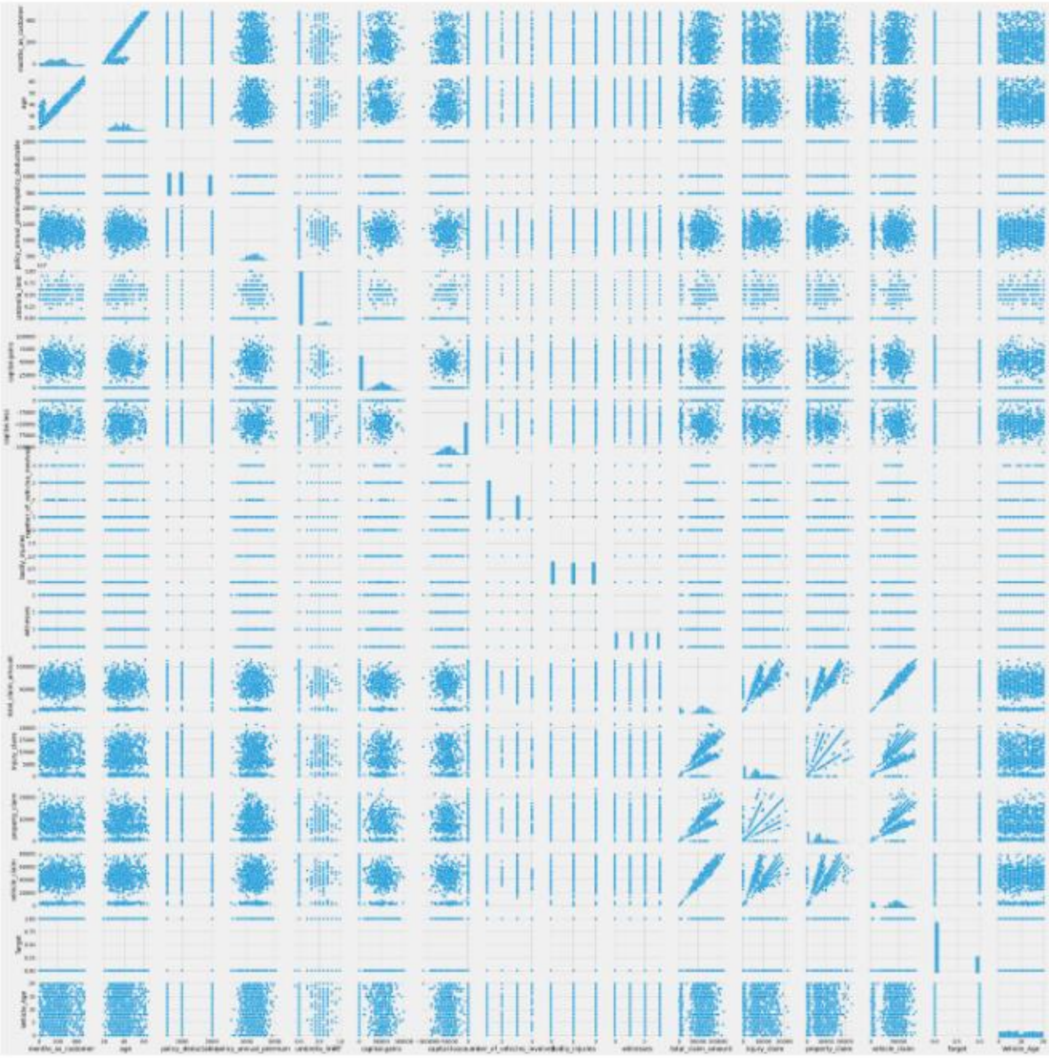


There is equal chance of fraud claim in both the gender.

Checking Pairplot:

```
sns.pairplot(df)
```

<seaborn.axisgrid.PairGrid at 0x20c6a83a880>

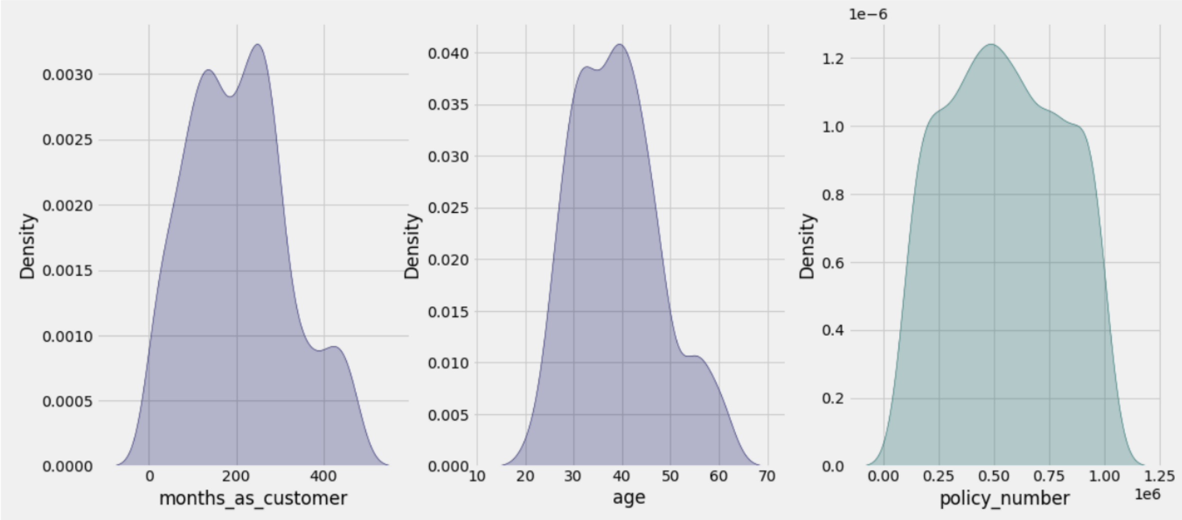


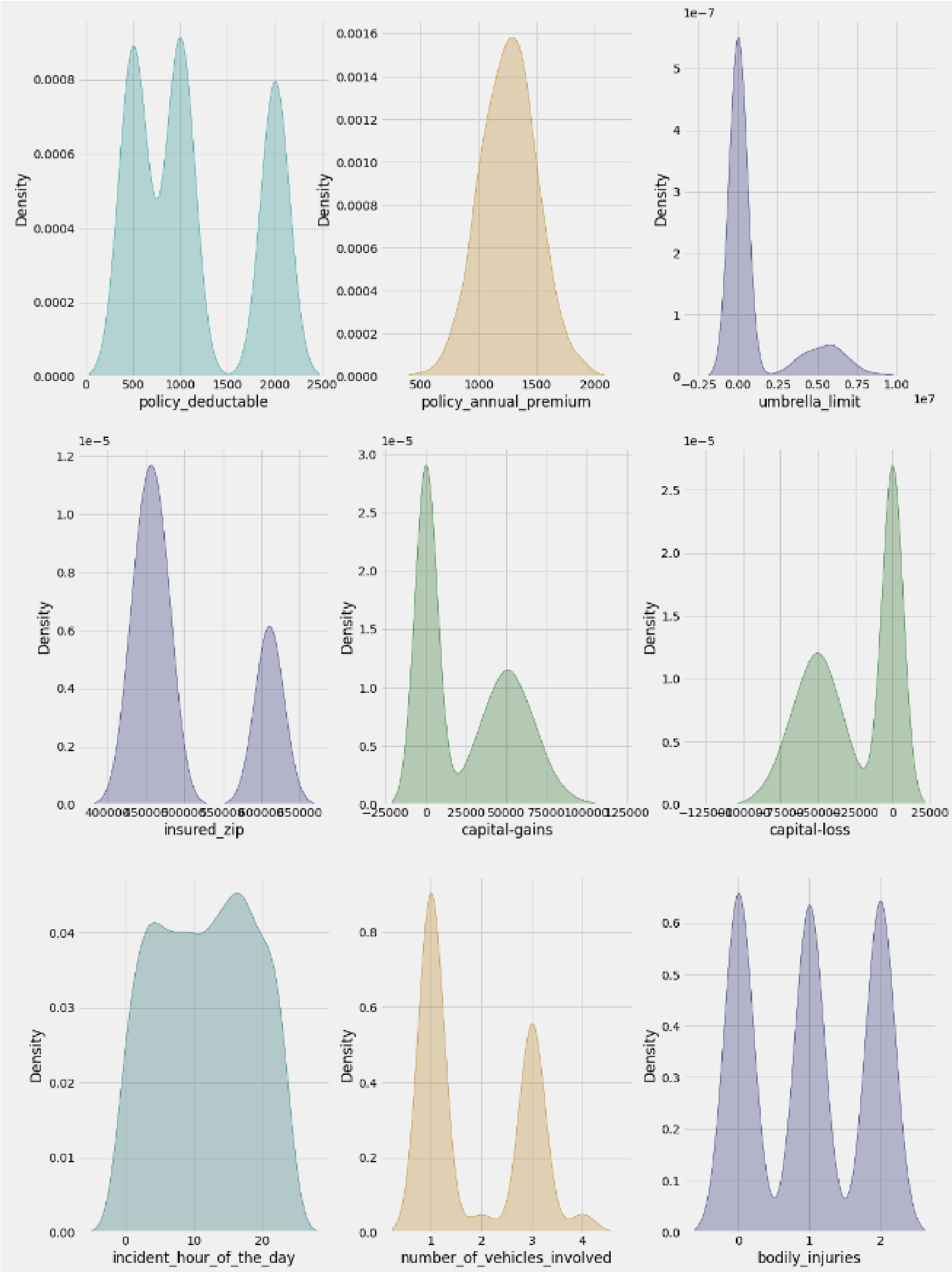
In the pair plot we see each feature with fraud reported. In the diagram the diagonal it is normally distributed. The data are scatter.

Checking Distribution for numerical data:

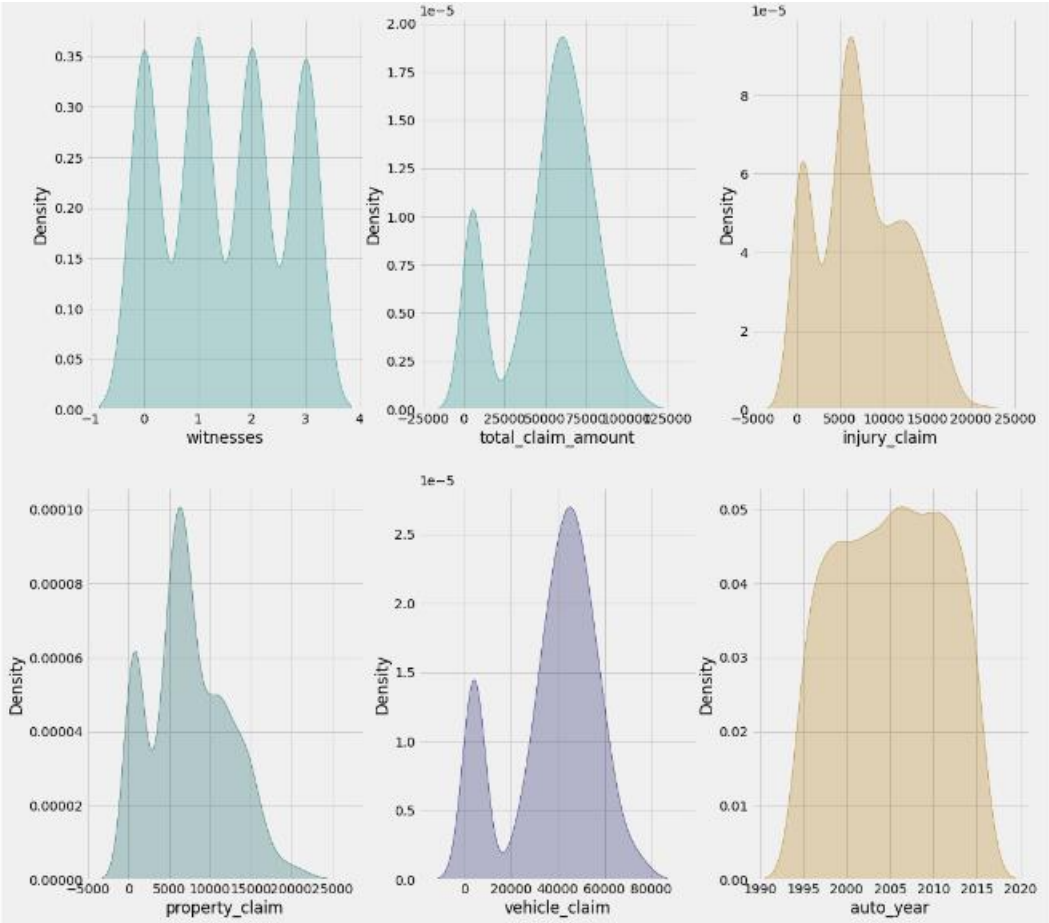
Distribution of Numerical features

```
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]])
```





Density
0.0000
0.0002
0.0004
0.0006
0.0008
0
500
1000
1500
2000
2500
policy_deductable
0.0000
0.0002
0.0004
0.0006
0.0008
0.0010
0.0012
0.0014
0.0016
500
1000
1500
2000
policy_annual_premium
1e−7
0
1
2
3
4
5
−0.25
0.00
0.25
0.50
0.75
1.00
1e7
umbrella_limit
1e−5
0.0
0.2
0.4
0.6
0.8
1.0
1.2
insured_zip
1e−5
0.0
0.5
1.0
1.5
2.0
2.5
3.0
−25000
0
25000
50000
75000
100000
125000
capital-gains
1e−5
0.0
0.5
1.0
1.5
2.0
2.5
0
25000
capital-loss
0.00
0.01
0.02
0.03
0.04
0
10
20
incident_hour_of_the_day
0.0
0.2
0.4
0.6
0.8
1
2
3
4
number_of_vehicles_involved
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0
1
2
bodily_injuries

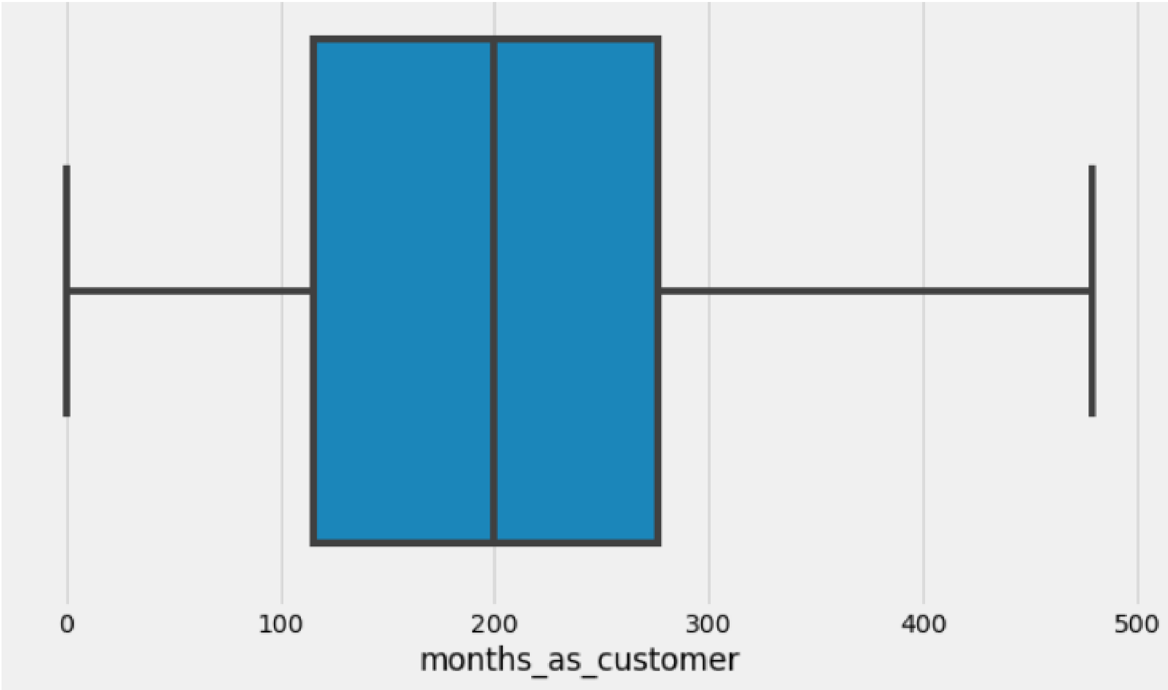


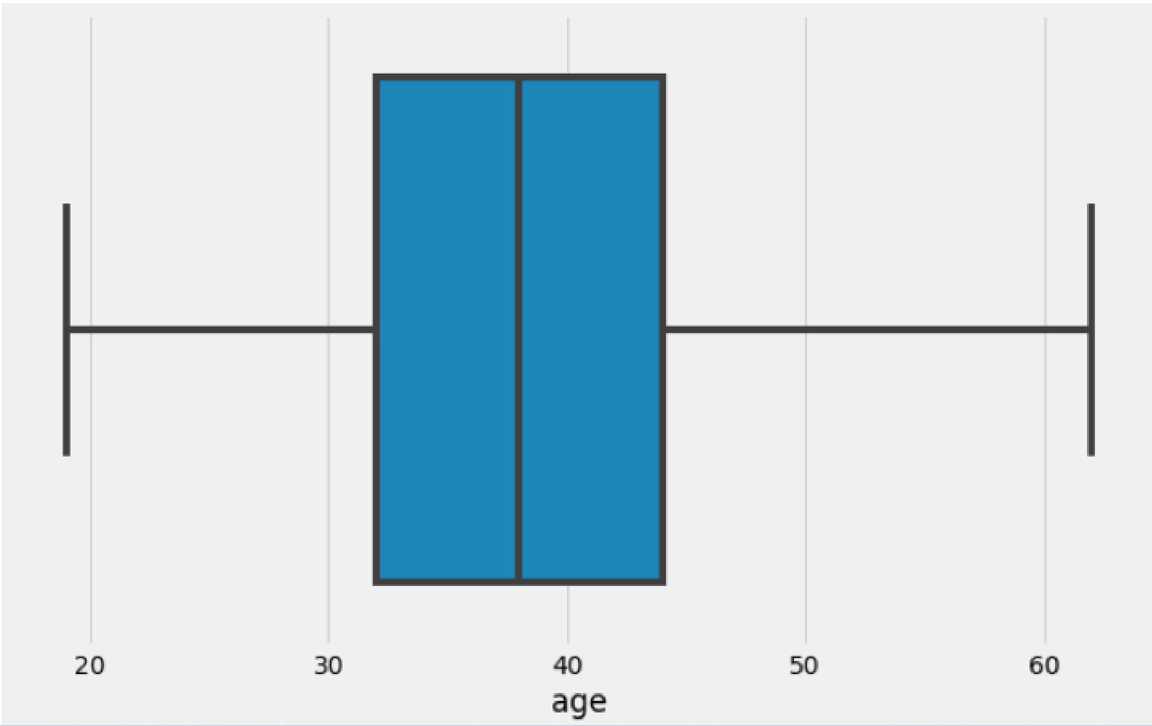
In the distribution plot we see that data are normally distributed and skewness also present. In insurance claim, property claim, injury claim, that data is slightly goes high and the diagram looks like a wave. We can remove the skewness to make the good accuracy.

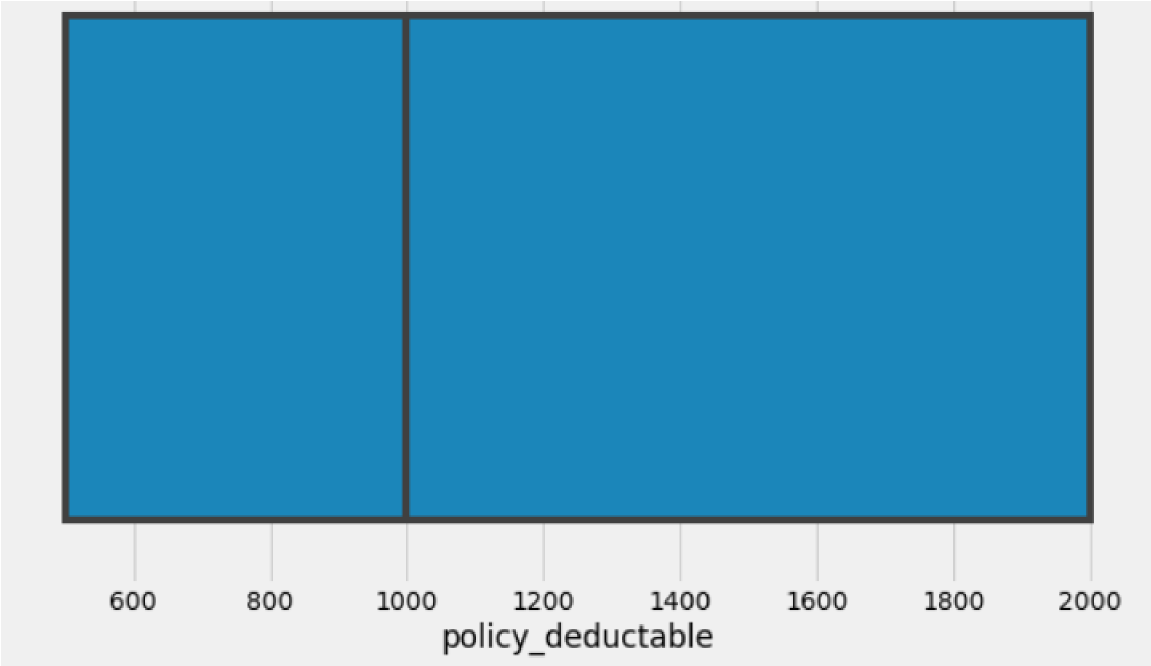
3. EDA Concluding Remark

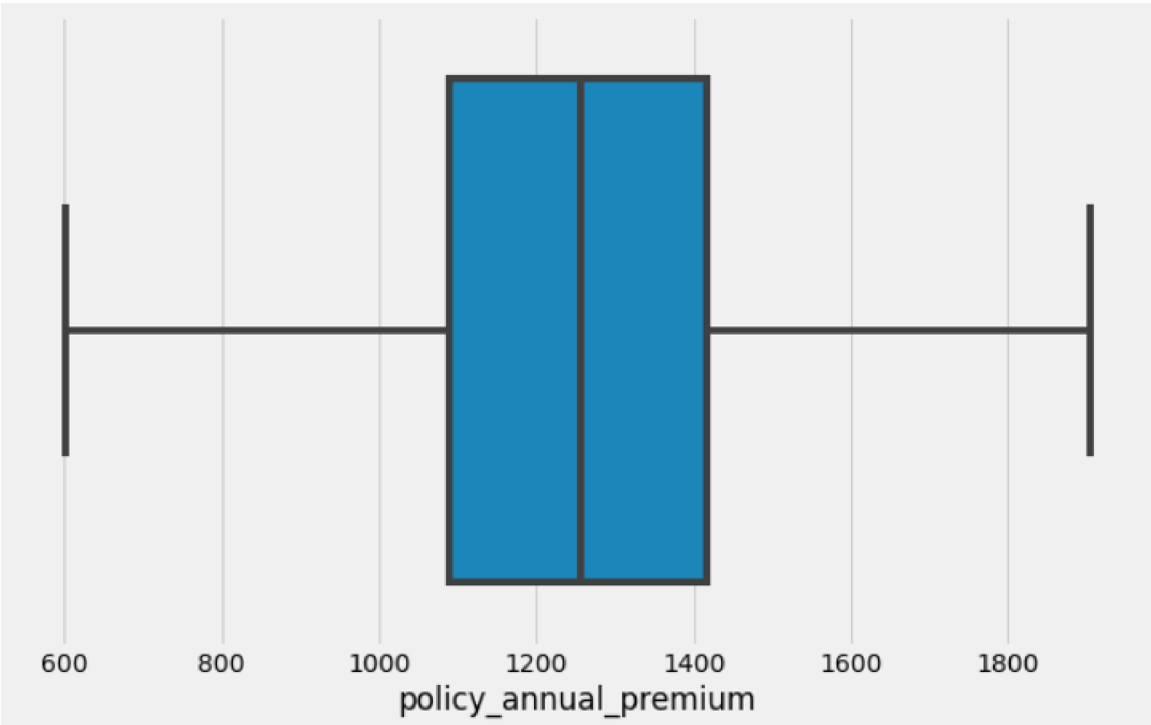
- Checked the null value using the heatmap, we found there is no null value present in data.
- We have converted the date datatype from object to datatime datatype and we have split into day, month and years.
- We have converted the categorical data to numerical data using the Label Encoder.
- We dropped the irrelevant columns from our data set.
- We checked the unique value, data info, shape of data, column name, statistical summary of data using describe method.
- We have checked the count of each feature and visualize them using bar chart, violin plot, count plot.
- We have visualized each feature with fraud reported and analysis in which category there is high chance of fraud reported.
- We have checked the correlation of data.
- In some column there is a '?' present we have replaced it with 'No info'
- Checked the normal distribution of feature to find whether the data contained skewness or not.
- Using the pair plot we have checked the relations of each feature with fraud reported.

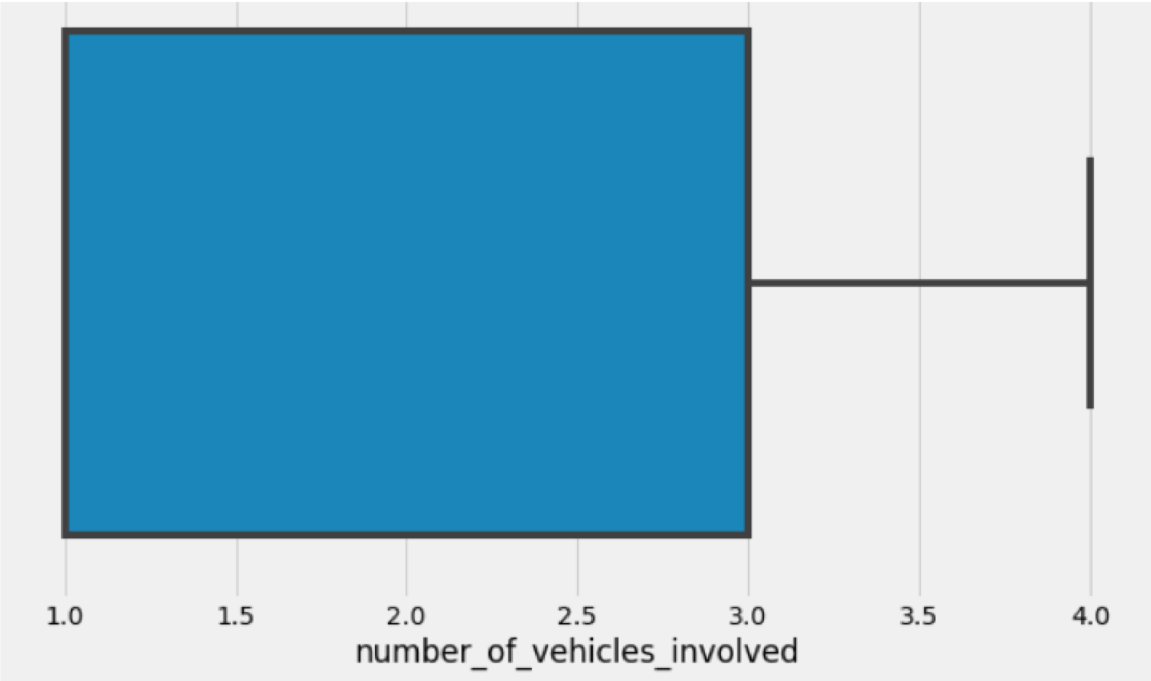
BOX PLOT:

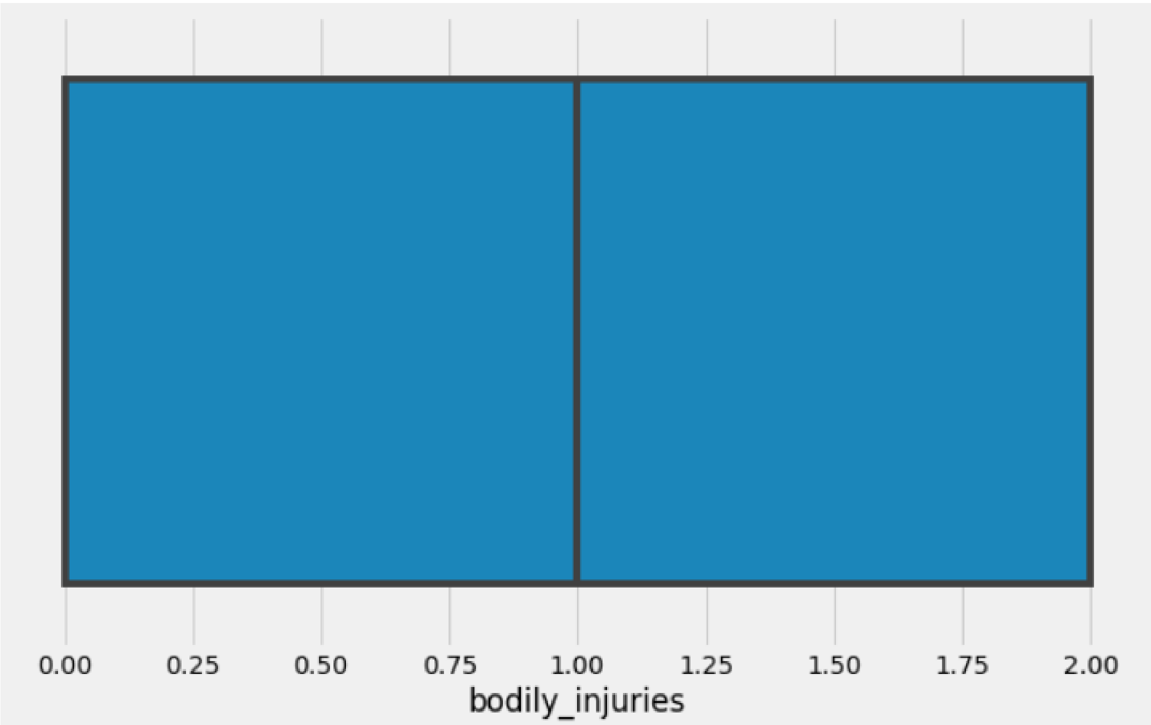

0
100
200
300
400
500
months_as_customer

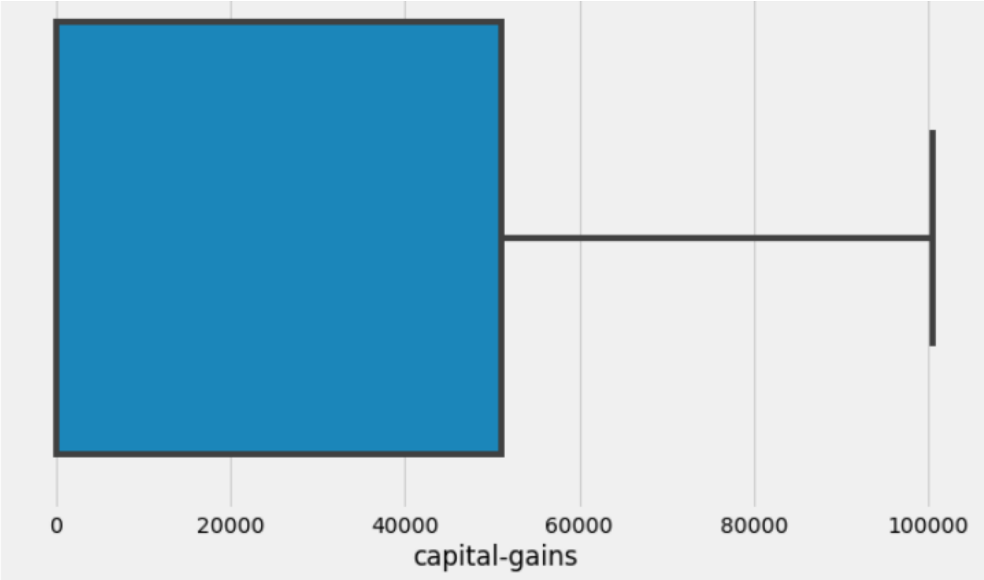

20
30
40
50
60
age

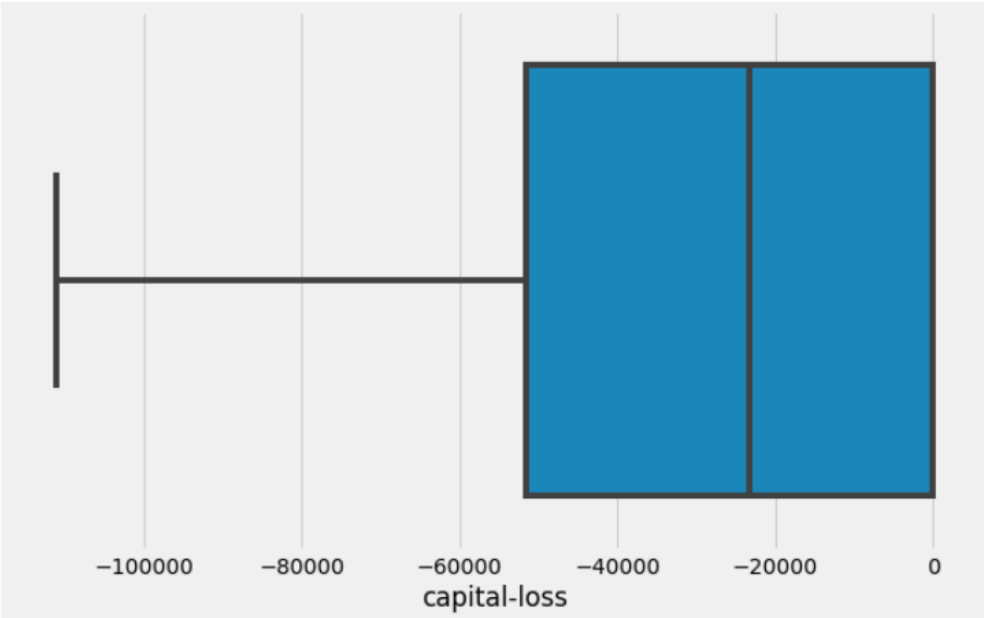

600
800
1000
1200
1400
1600
1800
2000
policy_deductable

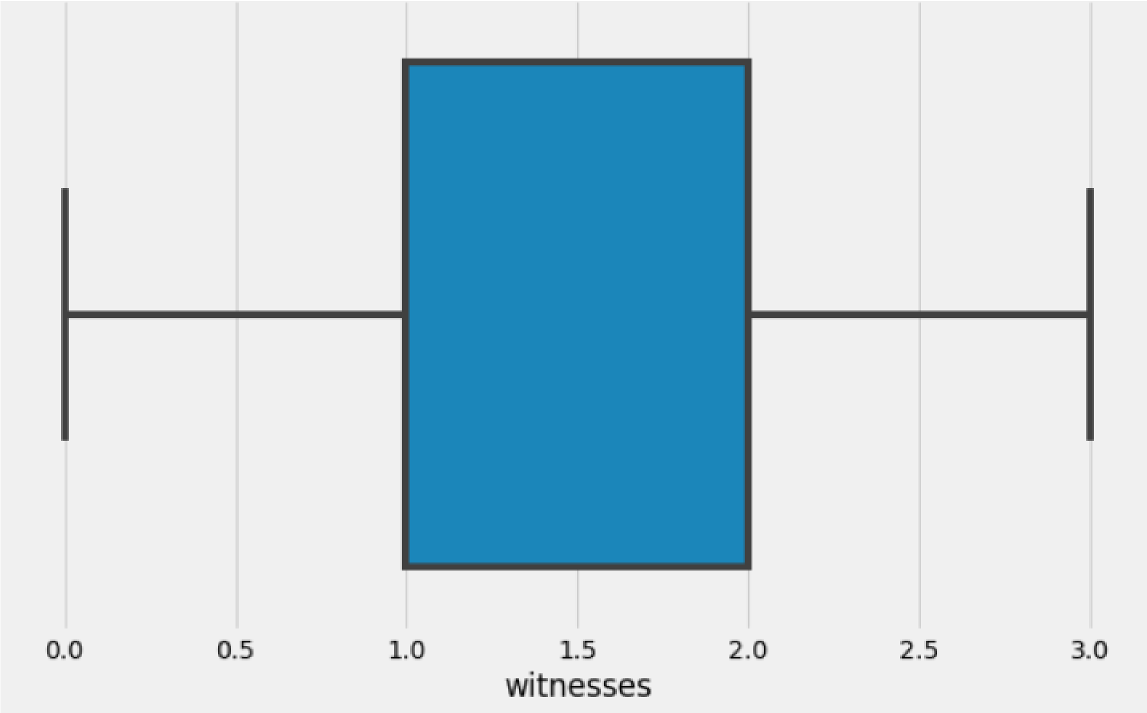

600
800
1000
1200
1400
1600
1800
policy_annual_premium


1.0
1.5
2.0
2.5
3.0
3.5
4.0
number_of_vehicles_involved

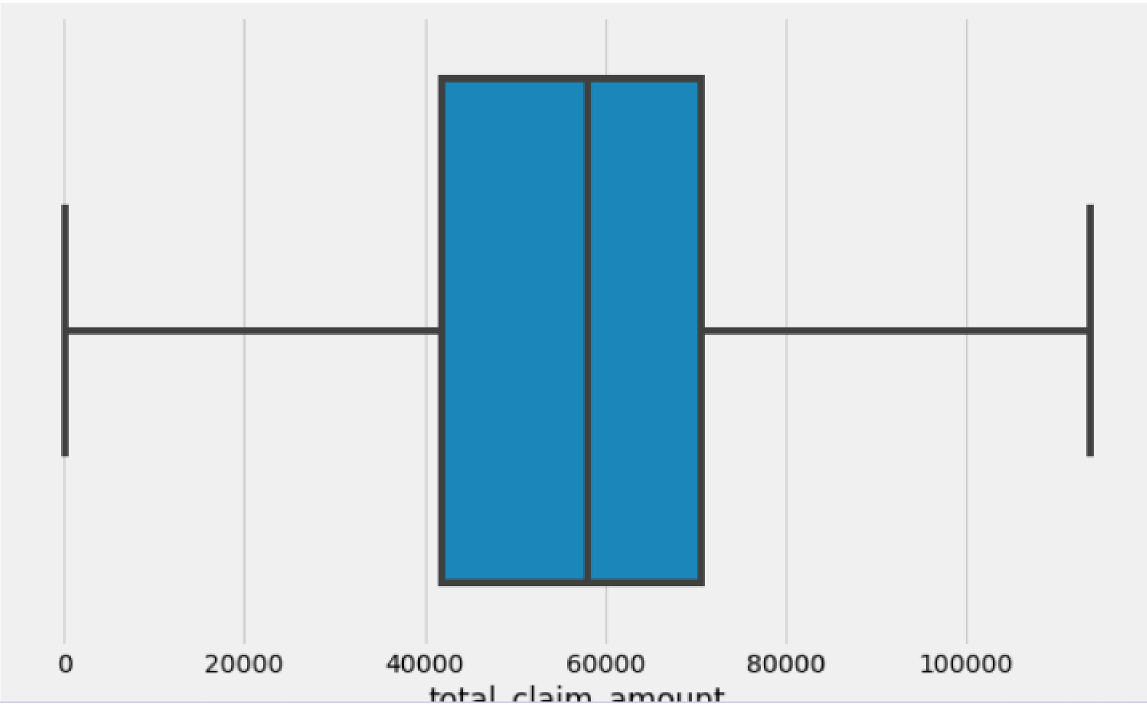

0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00
bodily_injuries


0
20000
40000
60000
80000
100000
capital-gains

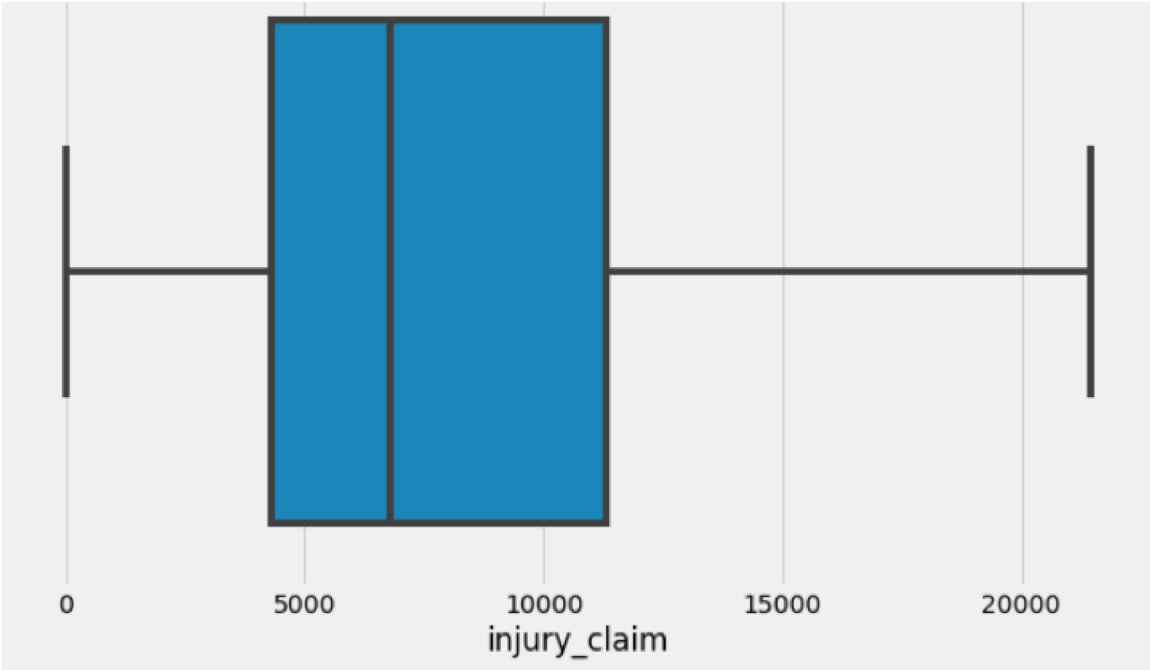

−100000
−80000
−60000
−40000
−20000
0
capital-loss

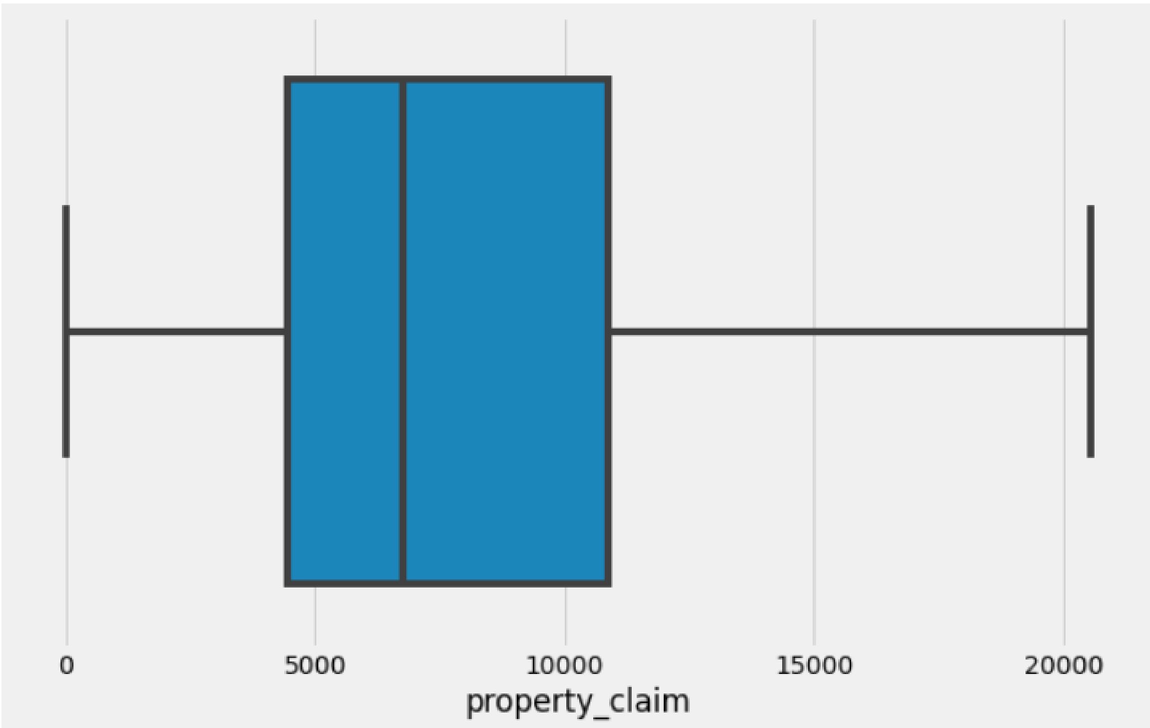


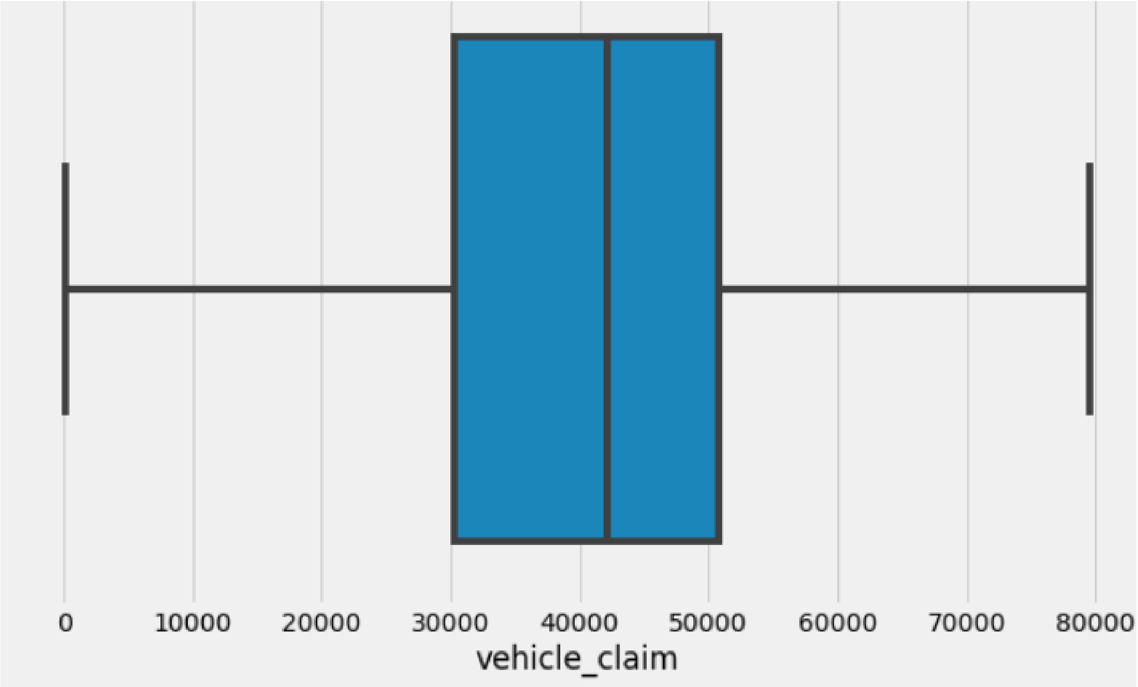
0.0
0.5
1.0
1.5
2.0
2.5
3.0
witnesses

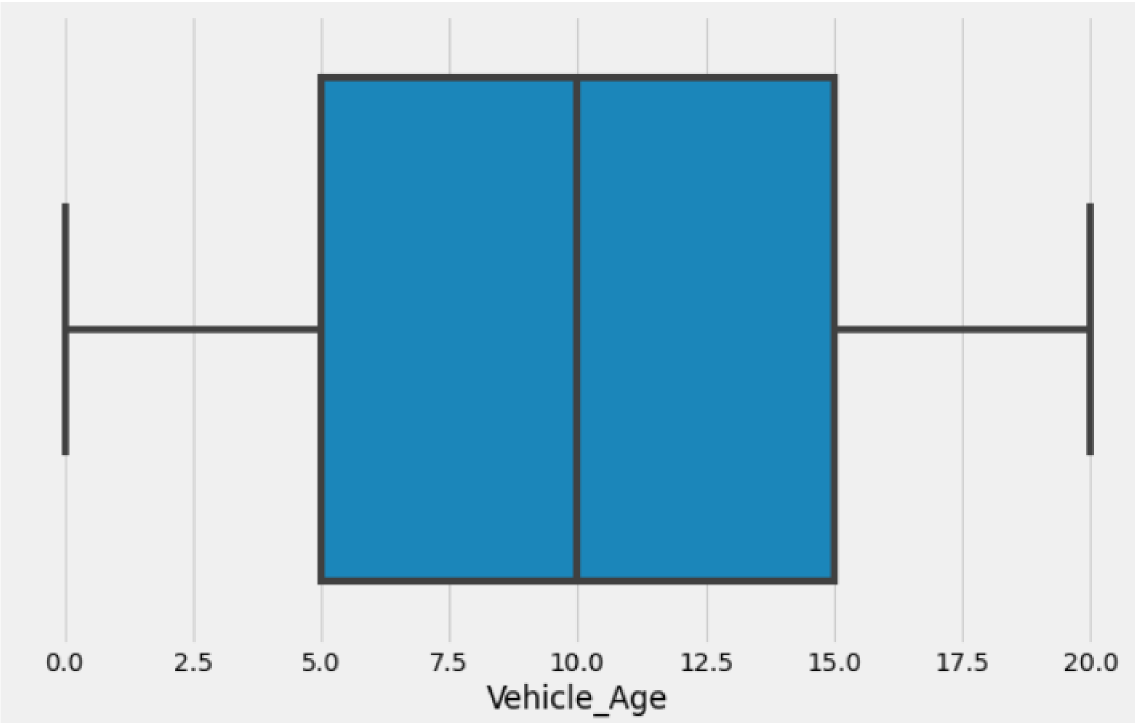


0
20000
40000
60000
80000
100000
total_claim_amount


0
5000
10000
15000
20000
injury_claim


0
5000
10000
15000
20000
property_claim





To check Outliers, I have used a boxplot. Outliers is present in age, policy_annual_premium, umbrella_limit, property_claim, incident_month.

Skewness:

```
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
```

Skewness is the distortion of the curve of normal distribution. Skewness can be of three types, positive, negative or zero skewness. If the tail of the normal curve is on the right side, then it is positive skewness. If the tail of the normal curve is on the left side, then it is a negative skewness.

In our dataset the skewness is present in Umbrella limit, total_claim_amount, vehicle_claim. We can remove the skewness using Power Transformer technique.

Skewess Removal:

```
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)
X
```

	months_as_customer	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occupation	insured_hobbies	insu
0	1.051279	OH	1.060250	1.060250	0	MALE	MD	craft-repair	sleeping	
1	0.304536	IN	0.275268	0.275268	5000000	MALE	MD	machine-op-inspct	reading	
2	-0.511226	OH	-0.535246	-0.535246	5000000	FEMALE	PhD	sales	board-games	
3	0.523344	IL	0.501720	0.501720	6000000	FEMALE	PhD	armed-forces	board-games	
4	0.304536	IL	0.275268	0.275268	6000000	MALE	Associate	sales	board-games	
...	...	...	...	...	...	...	...	...	...	
995	-2.287530	OH	-2.176642	-2.176642	0	FEMALE	Masters	craft-repair	paintball	
996	0.741446	IL	0.730542	0.730542	0	FEMALE	PhD	prof-specialty	sleeping	
997	-0.549706	OH	-0.572261	-0.572261	3000000	FEMALE	Masters	armed-forces	bungie-jumping	
998	1.912654	IL	2.000266	2.000266	5000000	MALE	Associate	handlers-cleaners	base-jumping	
999	1.900100	OH	1.986353	1.986353	0	FEMALE	Associate	sales	kayaking	

1000 rows × 32 columns

```
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])
```

```
X.head()
```

	months_as_customer	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occupation	insured_hobbies	insure
0	1.051279	OH	1.060250	1.060250	1	MALE	4	2	sleeping	
1	0.304536	IN	0.275268	0.275268	5	MALE	4	6	reading	
2	-0.511226	OH	-0.535246	-0.535246	5	FEMALE	6	11	board-games	
3	0.523344	IL	0.501720	0.501720	6	FEMALE	6	1	board-games	
4	0.304536	IL	0.275268	0.275268	6	MALE	0	11	board-games	

Converting Categorical data to Numerical data using Label Encoder:

```
cat=df[catg_features]
```

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

```
for i in catg_features:
    cat[i]=le.fit_transform(cat[i])
```

```
from sklearn.feature_selection import chi2
f_p_values=chi2(cat,Y)
f_p_values
```

```
(array([5.89272453e-01, 6.43859677e+00, 5.11837872e-01, 9.92123137e-02,
        5.93073552e-03, 7.25093249e+00, 5.17412489e-01, 2.53002377e+00,
        1.55344701e-01, 1.22804296e+02, 2.27023703e+00, 3.78510795e+00,
        2.29765975e+00, 2.06533457e-01, 5.28951079e-01, 1.88154130e+00,
        3.32040130e-03, 9.40383398e-01, 1.06123429e+00, 4.17843704e-01]),
 array([4.42700595e-01, 1.11666821e-02, 4.74344336e-01, 7.52776944e-01,
        9.38614580e-01, 7.08642033e-03, 4.71947532e-01, 1.11698533e-01,
        6.93479215e-01, 1.53904737e-28, 1.31879731e-01, 5.17105746e-02,
        1.29569089e-01, 6.49498120e-01, 4.67048182e-01, 1.70159073e-01,
        9.54048990e-01, 3.32179261e-01, 3.02933840e-01, 5.18014960e-01]))
```

```
f_p_values[1]
```

```
array([4.42700595e-01, 1.11666821e-02, 4.74344336e-01, 7.52776944e-01,
       9.38614580e-01, 7.08642033e-03, 4.71947532e-01, 1.11698533e-01,
       6.93479215e-01, 1.53904737e-28, 1.31879731e-01, 5.17105746e-02,
       1.29569089e-01, 6.49498120e-01, 4.67048182e-01, 1.70159073e-01,
       9.54048990e-01, 3.32179261e-01, 3.02933840e-01, 5.18014960e-01])
```

```
p_value=pd.Series(f_p_values[1],index=cat.columns)
p_value
```

```
policy_state                   4.427006e-01
umbrella_limit                 1.116668e-02
insured_sex                    4.743443e-01
insured_education_level        7.527769e-01
insured_occupation             9.386146e-01
insured_hobbies                7.086420e-03
insured_relationship           4.719475e-01
incident_type                  1.116985e-01
collision_type                 6.934792e-01
incident_severity              1.539047e-28
authorities_contacted          1.318797e-01
incident_state                 5.171057e-02
incident_city                  1.295691e-01
property_damage                6.494981e-01
police_report_available        4.670482e-01
auto_make                      1.701591e-01
auto_model                     9.540490e-01
csl_per_person                 3.321793e-01
csl_per_accident               3.029338e-01
incident_period_of_the_day     5.180150e-01
dtype: float64
```

```
p_value.sort_values(ascending=False,inplace=True)
```

We can't make the model using a categorical data. So, we need to convert categorical data to numerical data. I have used **Label Encoder** to convert my categorical data to numerical data.

4. Pre-Processing Pipeline.

Standard Scaler:

```
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

After data splitted into x and y, we can scaled the data. So, our data looks equal in size and store the data in a data frame.

The target variable fraud_reported is not equal. We have balanced the data. So, we have used the over sampling technique.

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

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

```
((1506, 127), (1506,))
```

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

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

Now the data is balanced.

Variance Inflation Factor:

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

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

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

```python
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

Variation Inflation Factor measures the severity of multi-collinearity. If there is high variation inflation present in data then there is result of collinearity.

We have considered the high inflation factor which is more than 10 for our data.

High variance inflation is present in total_claim_amount, injury_claim, property_claim, vehicle_claim.

So, we can drop the total_claim_amount, its VIF is 47858.

Checking the VIF after dropping of the total_claim_amount:

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

```
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']
```

```
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

After dropping the total_claim_amount, we removed all the high inflation factor from our data set. There is no multi-collinearity exist.

5. Building Machine Learning Models

Importing the necessary libraries for our model

```
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 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()
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,SGH_model,Bag_model,ADA_model,GB_model ]
```

Checking the best random state for our model. Using Logistic Regression, Ridge Classifier, Decision Tree Classifier, SVC, KNeighborsClassifier, Random Forest Classifier, SGDCClassifier, BaggingClassifier, AdaBoostClassifier, GradientBoosting Classifier.

Confusion Matrix:

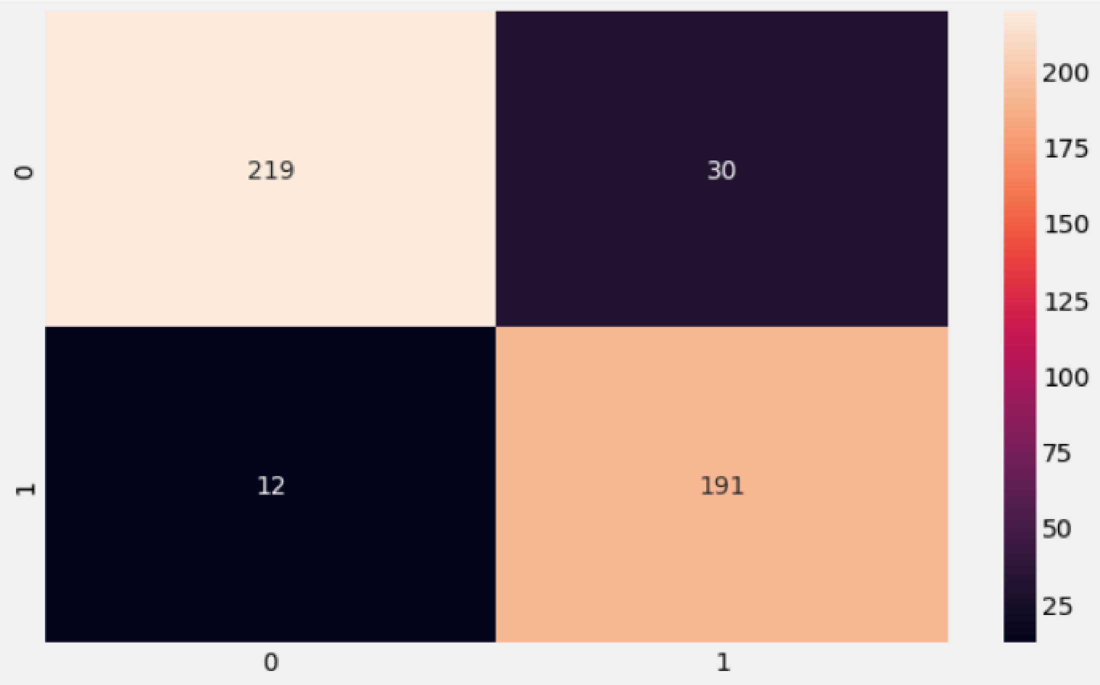
```
Confusion Matrix of  LogisticRegression()  is
 [[235  14]
 [ 18 185]]
Confusion Matrix of  RidgeClassifier()  is
 [[232  17]
 [ 17 186]]
Confusion Matrix of  DecisionTreeClassifier()  is
 [[216  33]
 [ 25 178]]
Confusion Matrix of  SVC()  is
 [[238  11]
 [ 42 161]]
Confusion Matrix of  KNeighborsClassifier()  is
 [[ 26 223]
 [  1 202]]
Confusion Matrix of  RandomForestClassifier()  is
 [[237  12]
 [ 30 173]]
Confusion Matrix of  SGDClassifier()  is
 [[227  22]
 [ 20 183]]
Confusion Matrix of  BaggingClassifier()  is
 [[228  21]
 [ 25 178]]
Confusion Matrix of  AdaBoostClassifier()  is
 [[230  19]
 [ 19 184]]
Confusion Matrix of  GradientBoostingClassifier()  is
 [[228  21]
 [ 14 189]]
```

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

	Model	Accuracy	F1 Score
0	LogisticRegression()	92.92	92.04
1	RidgeClassifier()	92.48	91.63
2	DecisionTreeClassifier()	87.17	85.99
3	SVC()	88.27	85.87
4	KNeighborsClassifier()	50.44	64.33
5	(DecisionTreeClassifier(max_features='sqrt', r...	90.71	89.18
6	SGDClassifier()	90.71	89.71
7	(DecisionTreeClassifier(random_state=53501980)...	89.82	88.56
8	(DecisionTreeClassifier(max_depth=1, random_st...	91.59	90.64
9	([DecisionTreeRegressor(criterion='friedman_ms...	92.26	91.53

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

```
<AxesSubplot:>
```



Cross Validation:

	Model	Accuracy	Cross Validation	Difference
0	LogisticRegression()	92.920354	85.803833	7.116521
1	RidgeClassifier()	92.477876	85.141361	7.336515
2	DecisionTreeClassifier()	87.168142	83.209610	3.626745
3	SVC()	88.274336	83.352401	4.921935
4	KNeighborsClassifier()	50.442478	56.109657	-5.667179
5	(DecisionTreeClassifier(max_features='sqrt', r...	90.707965	85.802513	4.374551
6	SGDClassifier()	90.707965	81.092165	12.610893
7	(DecisionTreeClassifier(random_state=53501980)...	89.823009	86.727905	2.829103
8	(DecisionTreeClassifier(max_depth=1, random_st...	91.592920	85.868958	5.723963
9	([DecisionTreeRegressor(criterion='friedman_ms...	92.256637	86.792590	5.530493

We checked cross validation score of various models, Gradient Boosting Classifier gives good accuracy score and cross validation score. So, we will consider Gradient Boosting Classifier as our final model. Let’s do the Hyper parameter tuning to check if we can increase the accuracy of our model.

Gradient Boosting Classifier() Hypertuning

```
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')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
GCV.best_estimator_
```

```
GradientBoostingClassifier(learning_rate=0.2, max_depth=4, n_estimators=5)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
GCV.best_params_
```

```
{'learning_rate': 0.2, 'max_depth': 4, 'n_estimators': 5}
```

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

```
0.9070796460176991
```

This are the best parameters and best estimators that are present in Gradient Boosting Classifier

We have done the hyper parameter tuning using grid search CV and passed some parameter and we trained the model. In Grid Search CV the best score is 90%. We will pass this estimator in our final model to check the accuracy.

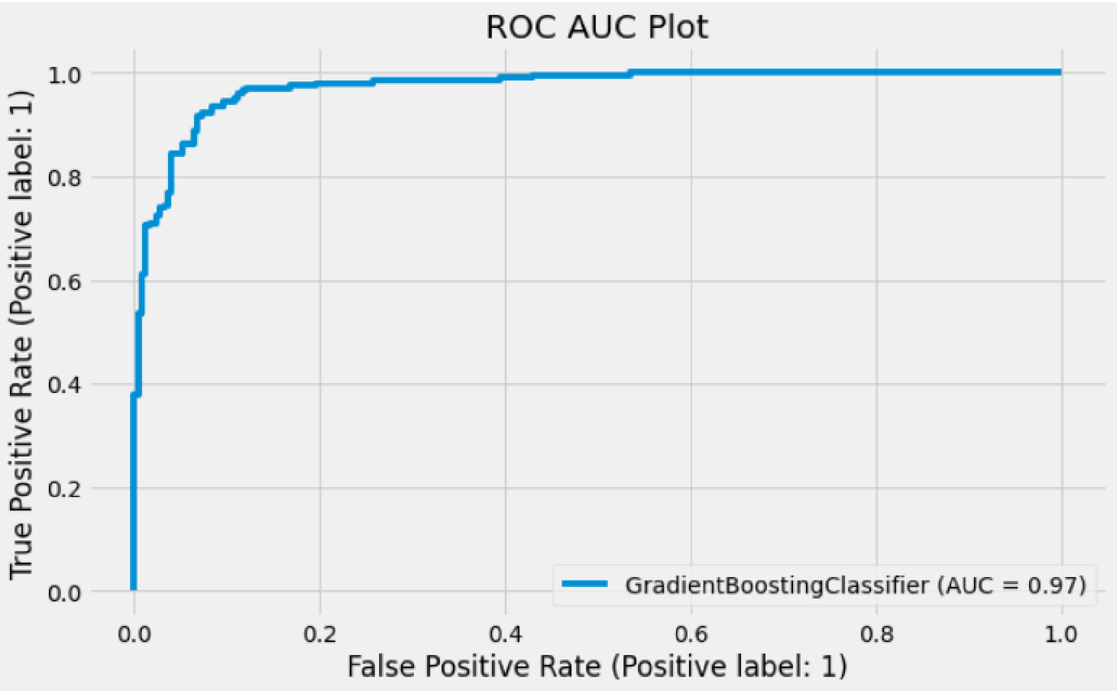
ROC Curve shows the relationship between sensitivity and specificity for every possible cut off.

ROC AUC CURVE:

```
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')
```



This is our false positive rate, true positive rate and thresholds of our model.

We have plotted the ROC Curve of our final model Gradient Boosting Classifier.

Saving The Model:

```
import joblib
joblib.dump(GB_model,"Insurance-claim-fraud.pkl")
```

```
['Insurance-claim-fraud.pkl']
```

We have saved the model using Pickle library. This is the predicted value of our final model.

Feature Selection: Constant Features

```
from sklearn.feature_selection import VarianceThreshold
var_thres=VarianceThreshold(threshold=0)
var_thres.fit(x_train)
```

```
VarianceThreshold(threshold=0)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
sum(var_thres.get_support())
```

```
127
```

```
len(x_train.columns[var_thres.get_support()])
```

```
127
```

```
x_train.shape
```

```
(1054, 127)
```

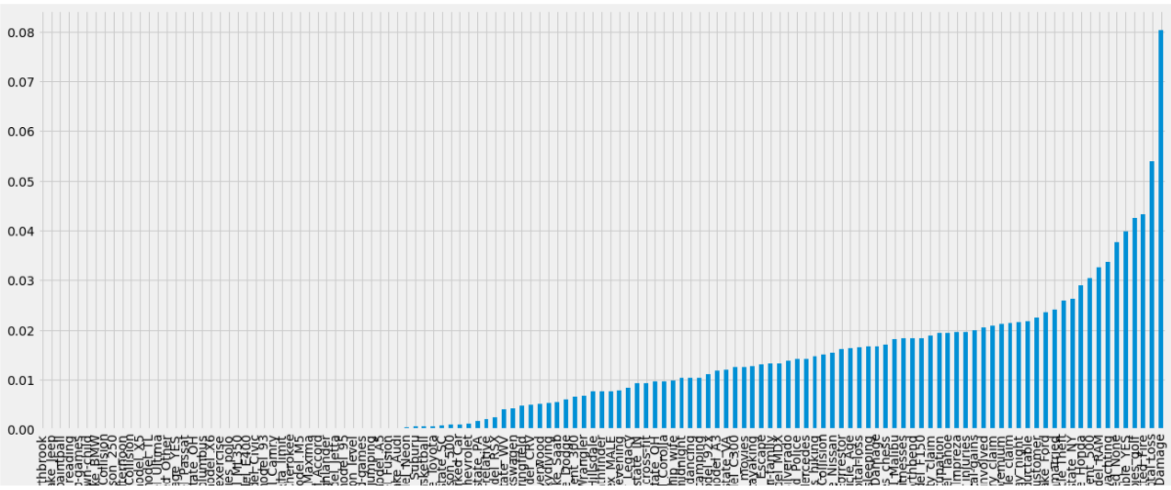
Feature Selection: Mutual Info Gain

```
from sklearn.feature_selection import mutual_info_classif

mutualinfo=mutual_info_classif(x_train,y_train)
```

```
mutual_info=pd.Series(mutualinfo)
mutual_info.index=x_train.columns
```

```
mutual_info.sort_values(ascending=True).plot.bar(figsize=(20,8))
```

<AxesSubplot:>



```
from sklearn.feature_selection import SelectKBest
select= SelectKBest(mutual_info_classif,k=93)
select.fit(x_train,y_train)
```

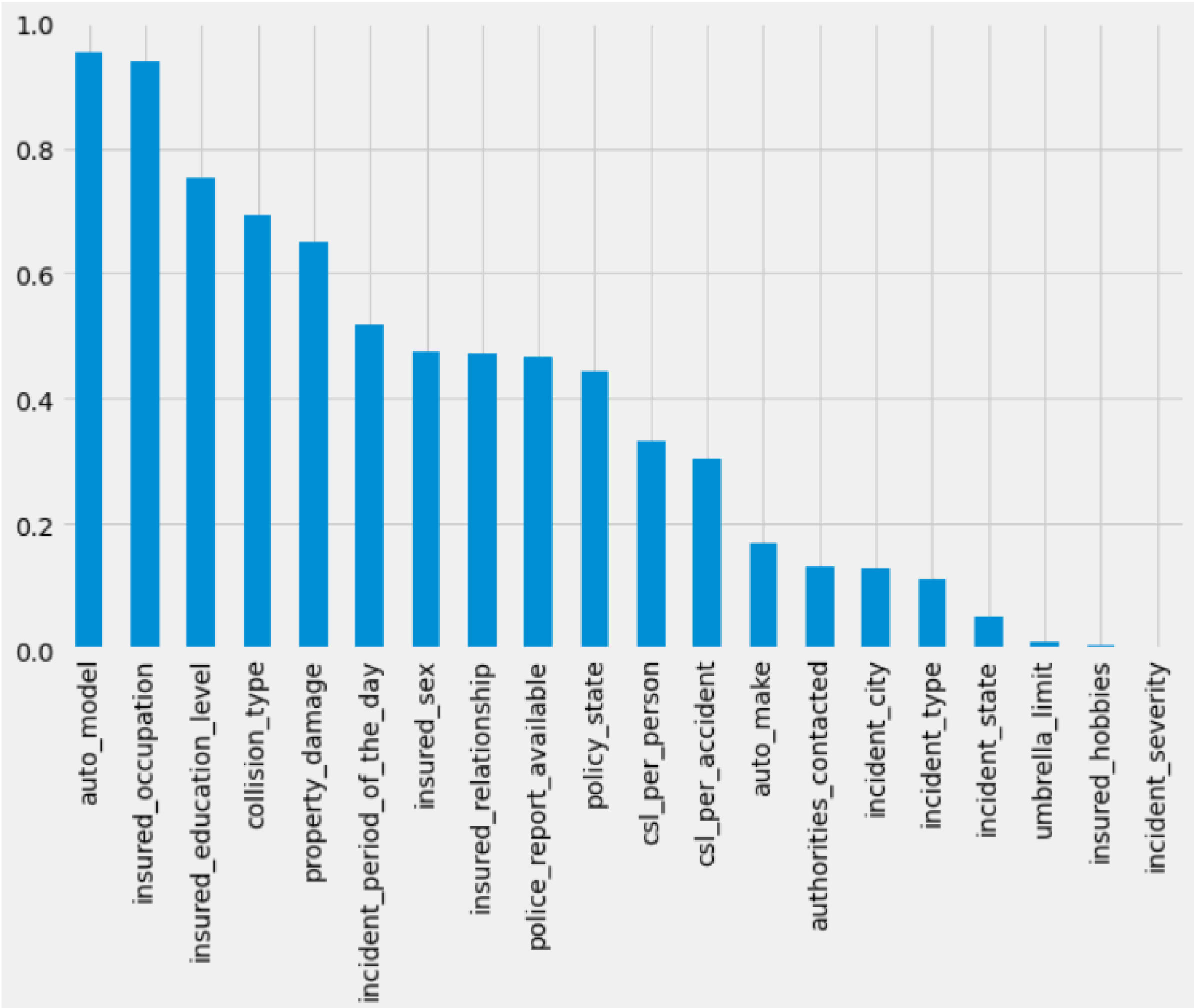
```
SelectKBest(k=93,
            score_func=<function mutual_info_classif at 0x0000020C78219160>)
```

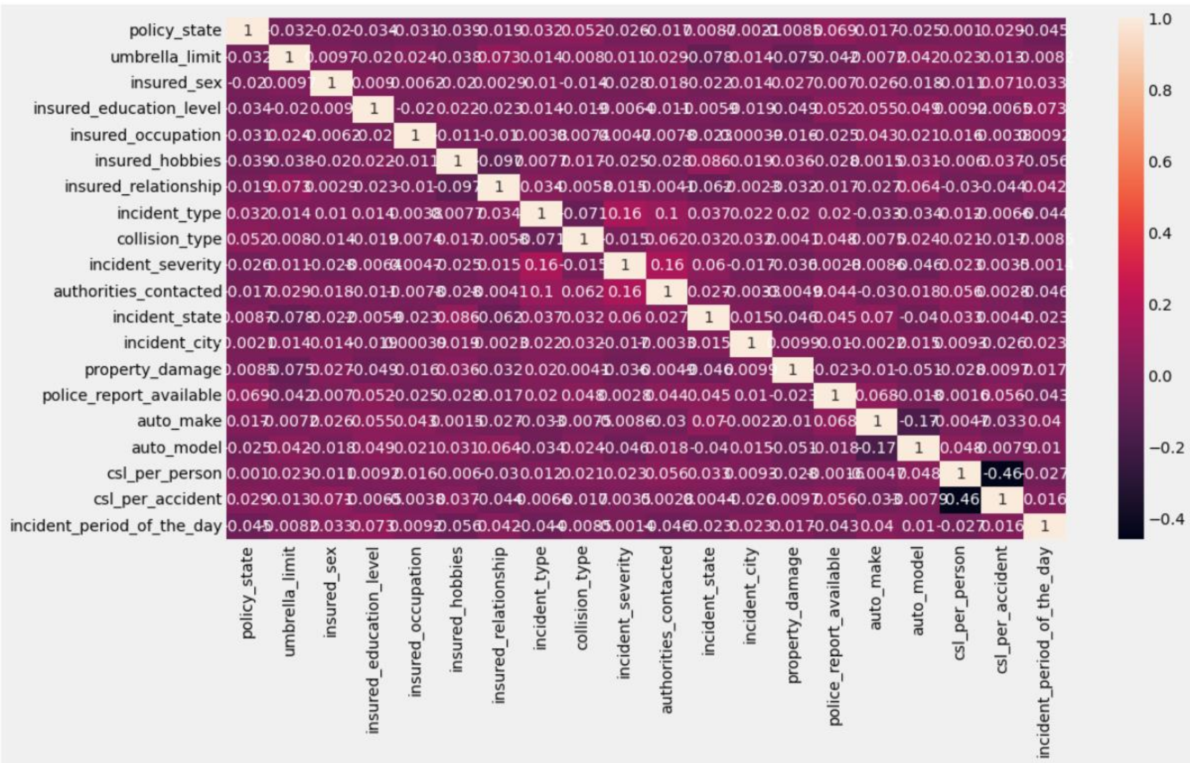
	Model	Accuracy	F1 Score
0	LogisticRegression()	90.93	89.98
1	RidgeClassifier()	91.37	90.51
2	DecisionTreeClassifier()	86.06	84.60
3	SVC()	89.16	87.40
4	KNeighborsClassifier()	51.33	64.52
5	(DecisionTreeClassifier(max_features='sqrt', r...	91.37	90.08
6	SGDClassifier()	89.16	86.86
7	(DecisionTreeClassifier(random_state=855586725...	90.93	89.83
8	(DecisionTreeClassifier(max_depth=1, random_st...	89.38	88.35
9	([DecisionTreeRegressor(criterion='friedman_ms...	91.37	90.51

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

```
for i in catg_features:
    cat[i]=le.fit_transform(cat[i])
```

```
policy_state                4.427006e-01
umbrella_limit              1.116668e-02
insured_sex                 4.743443e-01
insured_education_level     7.527769e-01
insured_occupation          9.386146e-01
insured_hobbies             7.086420e-03
insured_relationship        4.719475e-01
incident_type               1.116985e-01
collision_type              6.934792e-01
incident_severity           1.539047e-28
authorities_contacted       1.318797e-01
incident_state              5.171057e-02
incident_city               1.295691e-01
property_damage             6.494981e-01
police_report_available     4.670482e-01
auto_make                   1.701591e-01
auto_model                  9.540490e-01
csl_per_person              3.321793e-01
csl_per_accident            3.029338e-01
incident_period_of_the_day  5.180150e-01
dtype: float64
```


1.0
0.8
0.6
0.4
0.2
0.0
auto_model
insured_occupation
insured_education_level
collision_type
property_damage
incident_period_of_the_day
insured_sex
insured_relationship
police_report_available
policy_state
csl_per_person
csl_per_accident
auto_make
authorities_contacted
incident_city
incident_type
incident_state
umbrella_limit
insured_hobbies
incident_severity


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
csl_per_person
csl_per_accident
incident_period_of_the_day
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4

6. Concluding Remarks

- In this Insurance Claim – Fraud Detection project we have analysis the data using various plot in Visualization. We have gone through Feature Engineering, Pre-processing the data. We have handled the imbalance data. We have converted the categorical data to numerical data using a Label Encoder.
- We have checked the distribution of data and removed the skewness present. We have checked the correlation of data and the variation inflation factor. Then we have scaled the data.
- We have splitted the data for training and testing.
- We have dropped the column which is having the high variation inflation factor to avoid multi-collinearity issue. We have built various classification model and checked their accuracy score, confusion matrix and classification report.
- We have checked the cross-validation score of each model and compare with other models. Picked the best model which gives a good score.
- In our project Gradient Boosting Classifier gives a good score. We have done the hyper parameter tuning to increase accuracy. In our scenario the accuracy score didn't increase.
- We have seen that predicted and actual value almost similar; this means our model is working well. Based on this prediction we can predict which insurance claim is fraudulent or not.
- The insurance company can check the insured person features and predict the results whether to accept the insurance claim or reject it.
- This will help to decide and avoid the major loss to the company.
- Machine Learning model plays very important role in predicting the fraudulent claim.