INSURANCE CLAIMS FRAUD DETECTION

What are we going to discuss today?

Today we have been offered insurance for almost everything, you know what even for life, e.g. Health Insurance, Property Insurance, Automobile Insurance, even Insurance for mobiles, Life Insurance, Electronic gadgets what-not, the list goes on…, Let us know the challenges faced by the insurance companies.

What even insurance companies have challenges? Why not every business model has its own loop holes and challenges. Let us see the challenges and how can we help them with our knowledge of Machine Learning and Data Science.

Business Problem:

The most common and biggest problem faced by Insurance Companies is FRAUD CLAIMS, which is a huge loss to company. So we will try to find out whether the claim is Genuine or Fraud based on the data from an Auto Insurance Company by building a Machine Learning model.

Image source: Google – Credits to respected owner created

Before getting started with the project in building Machine Learning model, even before viewing the data it is always best practice to know about the problem / business domain and have some idea of it. Which is helpful in our entire project phase. So I recommend to have a quick check with our best buddy, you're right google the best source for basic insights.

Glimpse of Automobile Insurance Domain:

In an automobile insurance the company have some certain terms and conditions, checklists where the insurance will be covered and where it doesn't take responsibility or pay any amount if it doesn't meet their terms like : Not eligible for claim in case of accident when drunk and driven, etc.

The Fraud Claim: It is a false claim by user even if they don't meet the terms of company but will show as the Genuine and make claim to the company that they followed all their terms and is eligible for claim for their own benefits / to not occur loss.

DATA SOURCE:

The source of data to me is from DATA TRAINED Institute.

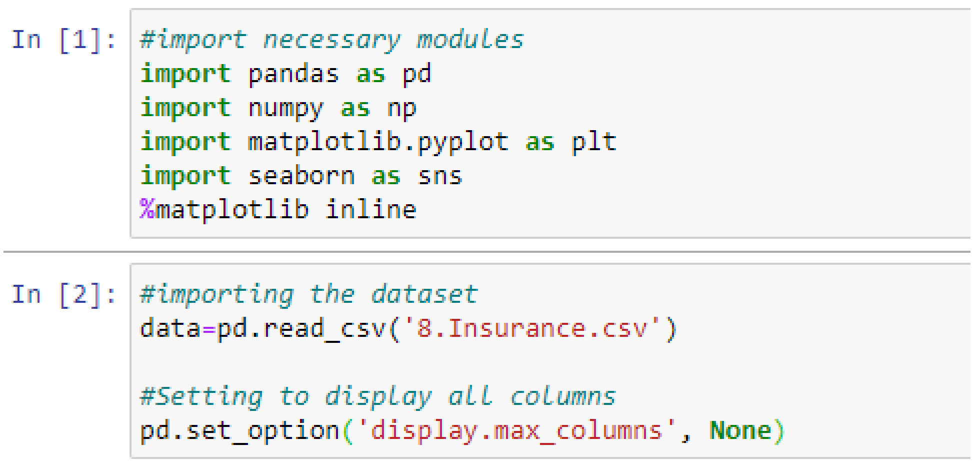
File format: .csv

The data is collection of different parameters and checks with the claim status whether fraud or not collected from different insurance companies and compiled into one file.

Step by Step process:

1. Import the dataset:

Importing the dataset in jupyter notebook and saving to a variable



```
In [1]: #import necessary modules
        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
```

```
In [2]: #importing the dataset
        data=pd.read_csv('8.Insurance.csv')

        #Setting to display all columns
        pd.set_option('display.max_columns', None)
```

Cell [1]: we imported some required modules to get data and create visualizations.

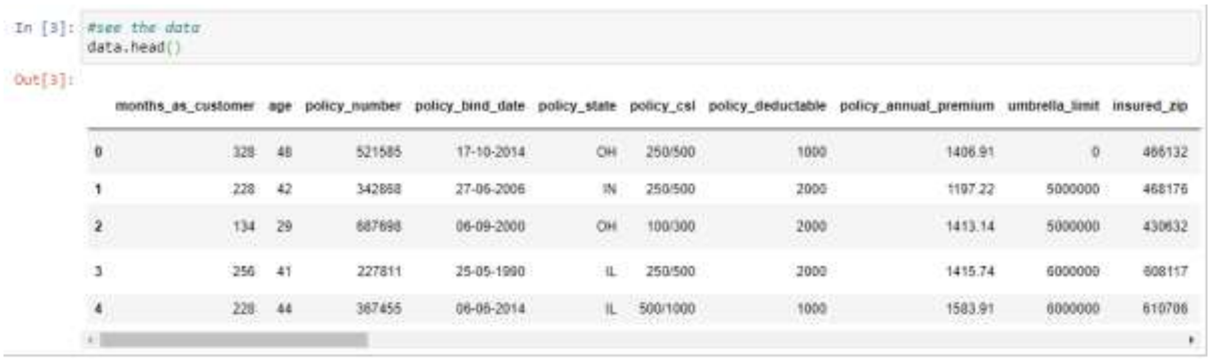
Cell [2]: We imported dataset into variable ‘data’,

8. Insurance.csv is the file name which I stored in my local machine at current working directory.

Then we set an option present, to display all the columns of the dataset when we viewed it.

Data Analysis: Narrow method of checking assumptions for model fitting, hypothesis testing by cleaning, filling missing values and making transformations as required.

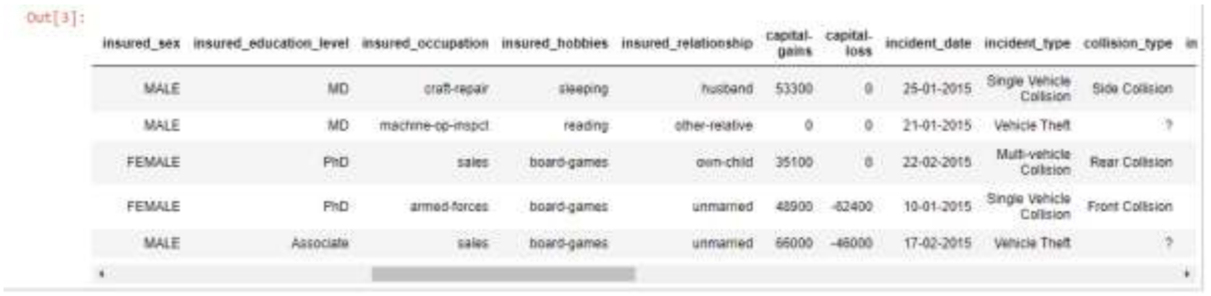
Viewing the data:



```
In [3]: #see the data
        data.head()
```

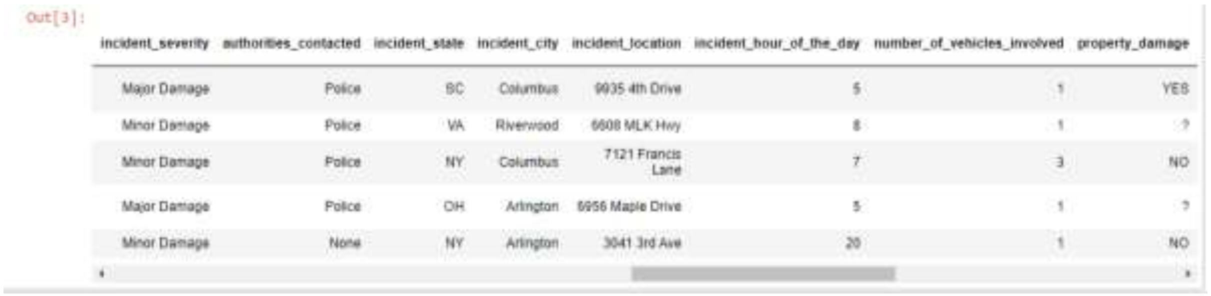
Out[3]:

	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



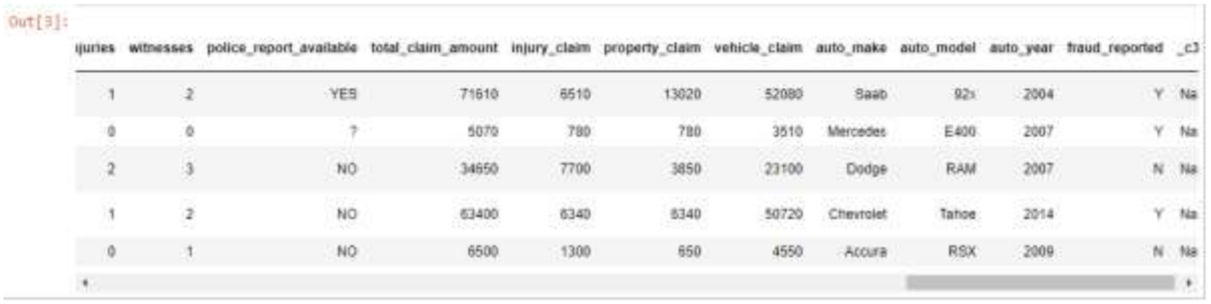
Out[3]:

insured_sex	insured_education_level	insured_occupation	insured_hobbies	insured_relationship	capital-gains	capital-loss	incident_date	incident_type	collision_type	in
MALE	MD	craft-repair	sleeping	husband	53300	0	25-01-2015	Single Vehicle Collision	Side Collision	
MALE	MD	machine-op-inspct	reading	other-relative	0	0	21-01-2015	Vehicle Theft	?	
FEMALE	PhD	sales	board-games	own-child	35100	0	22-02-2015	Multi-vehicle Collision	Rear Collision	
FEMALE	PhD	armed-forces	board-games	unmarried	48900	-62400	10-01-2015	Single Vehicle Collision	Front Collision	
MALE	Associate	sales	board-games	unmarried	66000	-46000	17-02-2015	Vehicle Theft	?	



Out[3]:

incident_severity	authorities_contacted	incident_state	incident_city	incident_location	incident_hour_of_the_day	number_of_vehicles_involved	property_damage
Major Damage	Police	SC	Columbus	9935 4th Drive	5	1	YES
Minor Damage	Police	VA	Riverwood	6608 MLK Hwy	8	1	?
Minor Damage	Police	NY	Columbus	7121 Francis Lane	7	3	NO
Major Damage	Police	OH	Arlington	6956 Maple Drive	5	1	?
Minor Damage	None	NY	Arlington	3041 3rd Ave	20	1	NO

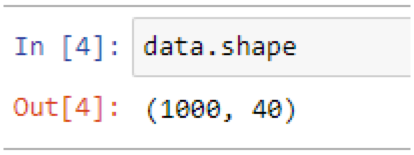


Out[3]:

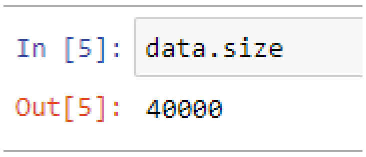
ijuries	witnesses	police_report_available	total_claim_amount	injury_claim	property_claim	vehicle_claim	auto_make	auto_model	auto_year	fraud_reported	_c3
1	2	YES	71610	6510	13020	52080	Saab	92x	2004	Y	Na
0	0	?	5070	780	780	3510	Mercedes	E400	2007	Y	Na
2	3	NO	34650	7700	3850	23100	Dodge	RAM	2007	N	Na
1	2	NO	63400	6340	6340	50720	Chevrolet	Tahoe	2014	Y	Na
0	1	NO	6500	1300	650	4550	Accura	RSX	2009	N	Na

We are now seeing the top 5 rows of our dataset. The data we have consists of columns with values categorical and numerical type also few columns with '?' in them.

Let us see the shape and size of dataset:



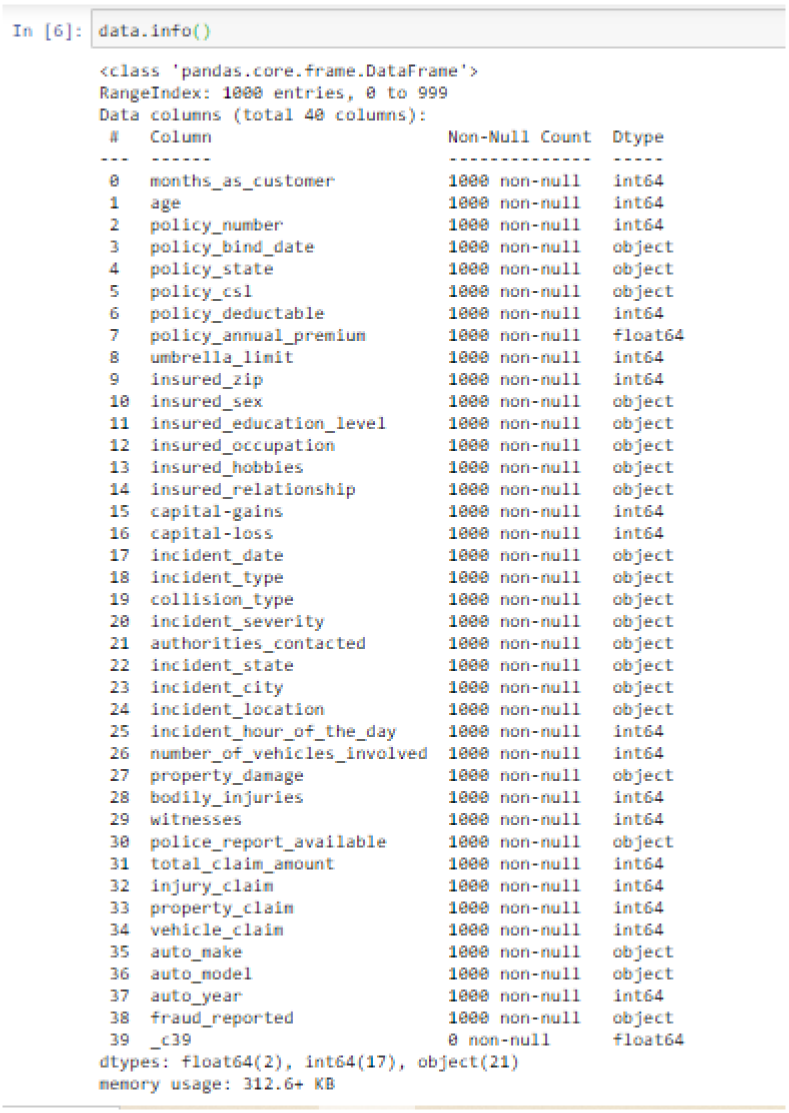
```
In [4]: data.shape

Out[4]: (1000, 40)
```



```
In [5]: data.size

Out[5]: 40000
```

There are 1000 rows and 40 columns, with a size of 40000 values (product of rows and columns)

Column details: Important step in EDA



```
In [6]: data.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
 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
```

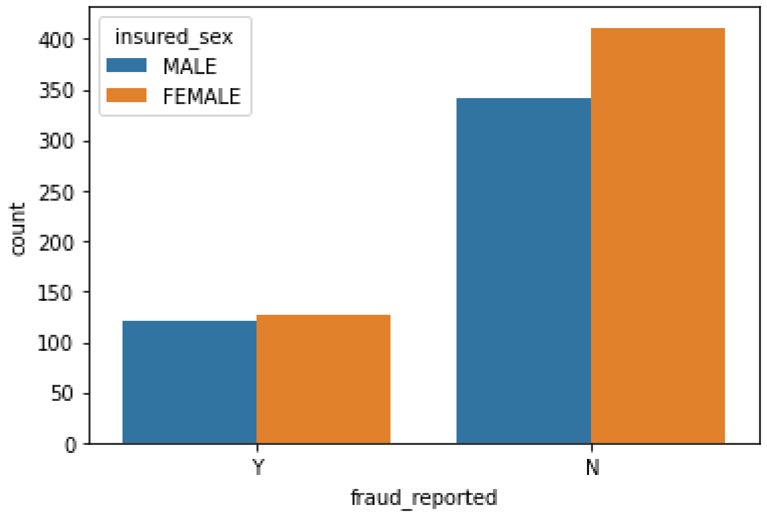
We got the brief of the data columns like – column name, number of non-null values and its data type. This is important step to see whether there are any datatype mismatches and to fix it. So we need to carefully observe the type of data present and verify here.

Observations:

- Our data is mixture of integer, float and object data types.
- Our target variable ‘fraud_reported’ is object type. So we use ‘Classification Model’.
- We see there are no null values in our brief data, but we found ‘?’ in some columns, so we need to replace them in the best possible way.
- Column policy_csl is shown as object but we can see it contains numerical data. There might be chances of ‘?’ or any other special character or text present in that column. So it is shown as object type. We need to find that and convert its type.
- Observe column 39: ‘_c39’- it has 0 non null values, i.e. all the values are null. So we can drop that column straight away.
- **Assumptions:**
    - Assuming that, for making a fraud it is personal intention and that doesn’t depend on what kind of auto make, model and year. So I opt to drop these columns also.
    - Policy_number: It is a unique id given to the customer, to track the subscription status and other details of customer. So we can drop this column, before dropping check if there are any duplicates in that column, if there are duplicates remove those rows by keeping one uniquely then drop the column.
    - Insured_zip: It is the zip code where the insurance was made. It doesn’t give any information or useful to us.so we will drop this column too.
    - Policy_csl is basically Combined Single Limit, i.e. how much credible and what their premium covers, it is also not necessary. So we will drop it.

We have seen that fraud reported by male & female both are almost same count.

```
: sns.countplot(data['fraud_reported'],hue=data['insured_sex'])

C:\Users\Lucky Girish\anaconda3\lib\site-packages\seaborn\_decor
yword arg: x. From version 0.12, the only valid positional argum
icit keyword will result in an error or misinterpretation.
  warnings.warn(

: <matplotlib.axes._subplots.AxesSubplot at 0x22b55456a90>
```



```
In [10]: #Dropping the columns
         dfc=data.drop(['policy_number','insured_zip','auto_make','auto_model','auto_year','_c39','policy_csl'],axis=1)
```

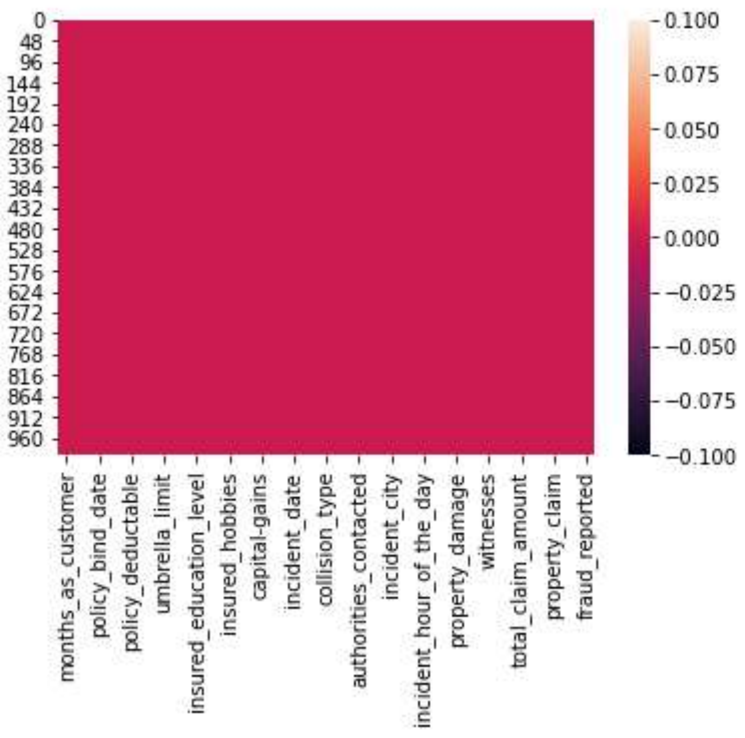
We dropped the columns and copy of it saved into new variable ‘dfc’, the initial variable data will have all columns. Where as in dfc we have columns that aren’t dropped.

```
In [11]: dfc.columns

Out[11]: Index(['months_as_customer', 'age', 'policy_bind_date', 'policy_state',
                'policy_deductable', 'policy_annual_premium', 'umbrella_limit',
                'insured_sex', 'insured_education_level', 'insured_occupation',
                'insured_hobbies', 'insured_relationship', 'capital-gains',
                'capital-loss', 'incident_date', 'incident_type', 'collision_type',
                'incident_severity', 'authorities_contacted', 'incident_state',
                'incident_city', 'incident_location', 'incident_hour_of_the_day',
                'number_of_vehicles_involved', 'property_damage', 'bodily_injuries',
                'witnesses', 'police_report_available', 'total_claim_amount',
                'injury_claim', 'property_claim', 'vehicle_claim', 'fraud_reported'],
               dtype='object')
```

This gives list of columns present in the variable dfc.

Checking for null values:

```
In [12]: #Visualising null values
         sns.heatmap(dfc.isnull())

Out[12]: <matplotlib.axes._subplots.AxesSubplot at 0x23c74090850>
```



There are no null values. We are going to replace the '?' with mean, median or mode or any other based on the column.

We see '?' present in many columns:

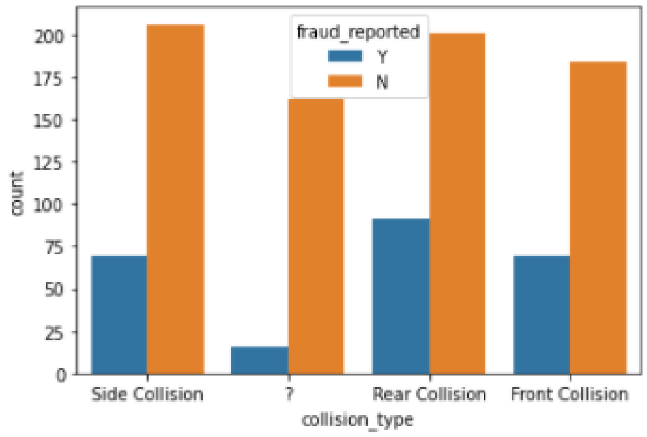
for e.g. we deal with one column collision type-

We can see there are 4 unique values in this column, as shown below. Front, Rear, Side and '?' Collisions. '?' is there in 178 rows. This might not be filled, it is a manual error. Because the collision should be in one of the category, and must note while claimed.

The best, thumb hand rule to deal this kind in case of categorical values is to fill with the mode of column.

So we will replace these '?' with the most repeated type of collision type: Rear Collision

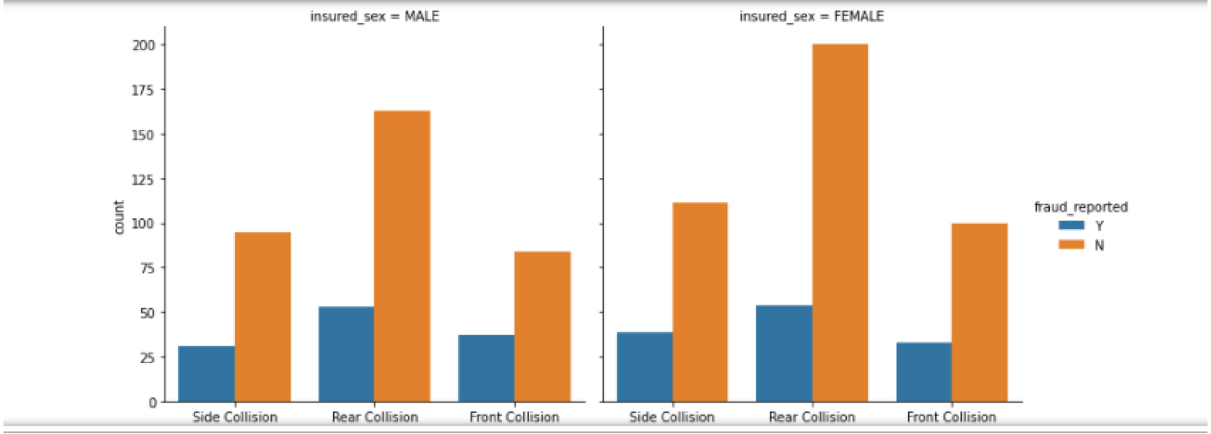
```
In [13]: #Collision_type
         dfc['collision_type'].nunique()

Out[13]: 4

In [15]: dfc['collision_type'].value_counts()

Out[15]: Rear Collision     292
         Side Collision     276
         Front Collision    254
         ?                  178
         Name: collision_type, dtype: int64

In [14]: sns.countplot(dfc['collision_type'],hue=dfc['fraud_reported'])

Out[14]: <matplotlib.axes._subplots.AxesSubplot at 0x23c74805a30>
```



Replacing and counts after replaced wrt to fraud claim or not:

```
In [27]: dfc['collision_type']=dfc['collision_type'].replace('?','Rear Collision')
```

```
In [37]: sns.catplot(x='collision_type', hue='fraud_reported',col='insured_sex',kind="count",data=dfc)
```

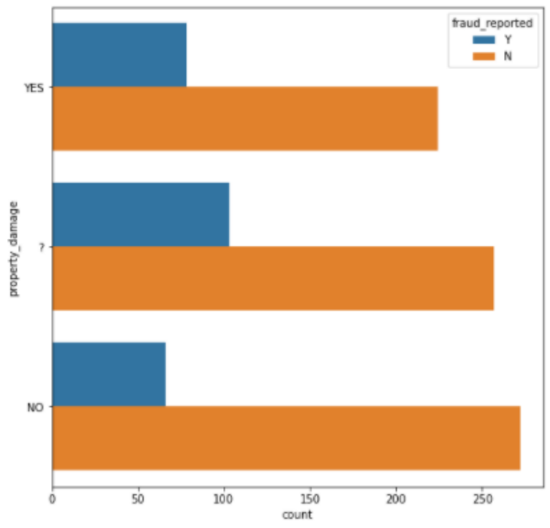


More collisions done by females than males. Rear collisions are max. From above we can also see, what kind of collision made by male or female and whether it is fraud claim or not. Almost count of all the fraud claims made are same irrespective of gender. Side collisions reported by females are fraudulent.

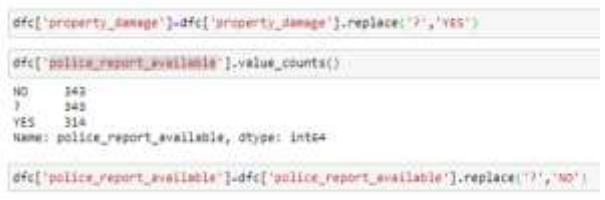
Similarly we do fill the '?' with mode of that column or by analysing the column. I had filled them with mode.

```
check('property_damage')
```

```
no. of unique values in property_damage is: 3
unique values in property_damage is: ?      360
NO     338
YES    302
Name: property_damage, dtype: int64
```



In Property damage '?' is replaced with YES. Since we observed that '?' have more fraud claims. By comparing with other data, claims with yes are more fraud. So we replaced that with YES.



```
dfc['property_damage']=dfc['property_damage'].replace('?','YES')
```

```
dfc['police_report_available'].value_counts()
```

```
NO     343
?      343
YES    314
Name: police_report_available, dtype: int64
```

```
dfc['police_report_available']=dfc['police_report_available'].replace('?','NO')
```

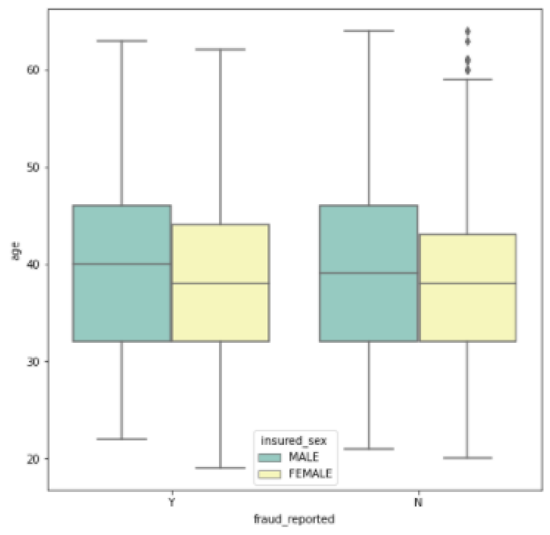
Similarly police report is also filled with NO

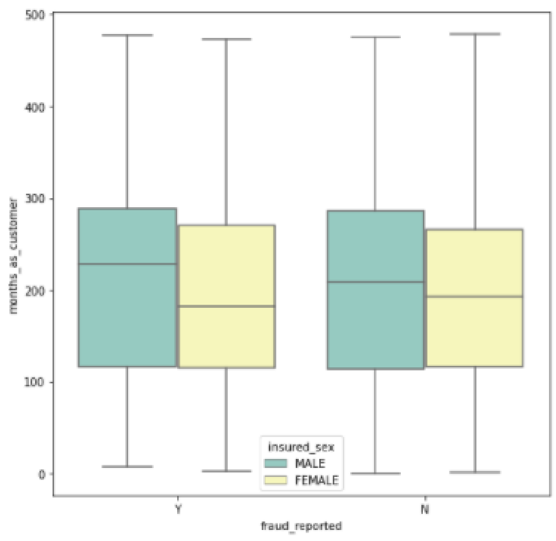
Let us see some important observations:

Average age and months being a customer: 38.9 yrs. old, 203.95 months as customer

```
Brief of age: count    1000.000000
mean       38.948000
std         9.140287
min        19.000000
25%        32.000000
50%        38.000000
75%        44.000000
max        64.000000
Name: age, dtype: float64
skewness: 0.47898804709224163
unique values count in age: 46
```

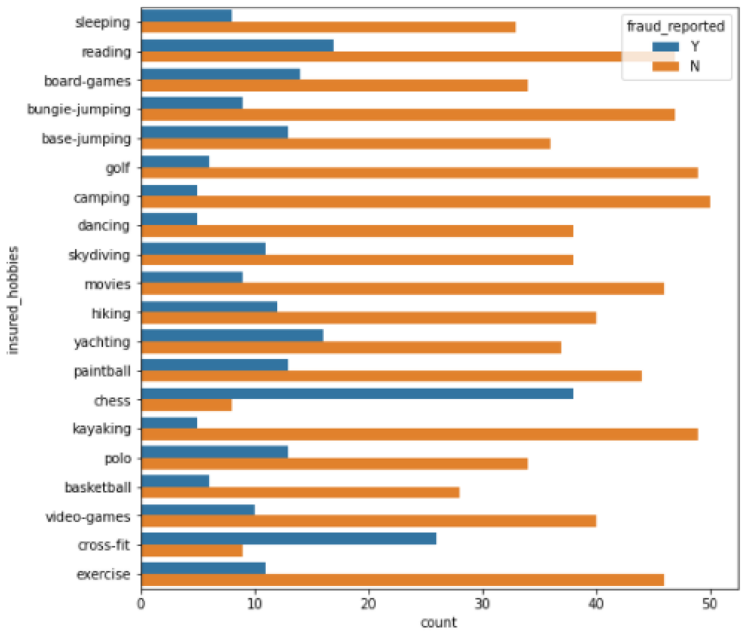
```
Brief of months_as_customer: count    1000.000000
mean      203.954000
std       115.113174
min         0.000000
25%       115.750000
50%       199.500000
75%       276.250000
max       479.000000
Name: months_as_customer, dtype: float64
skewness: 0.3621768477780205
unique values count in months_as_customer: 391
```



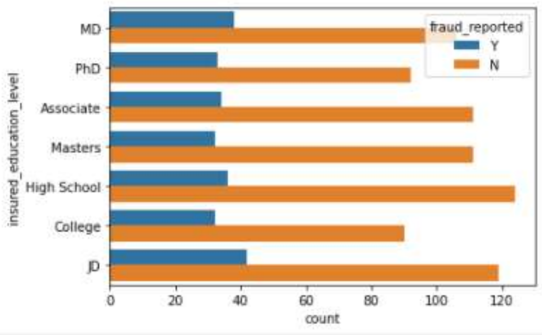


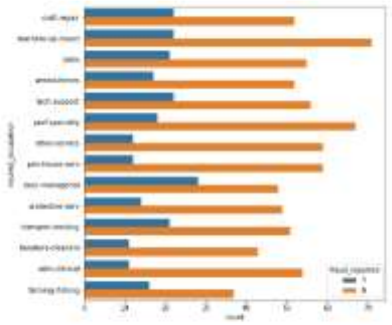
We can see males had done frauds from the age of 22/23 approx., females had done from teens, approx. from 18 yrs. Frauds committed by clients from their 1st month onwards.

Hobbies: People with hobby of chess and followed by cross-fit had done more fraud claims.



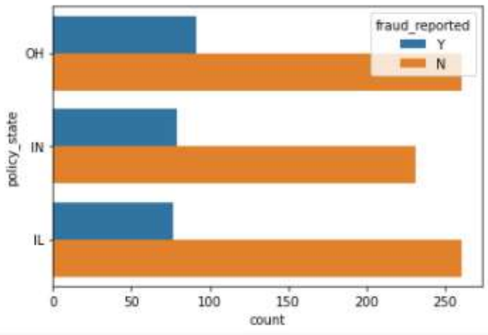
Education level: We can see, those who studied JD, MD has done more frauds comparatively.

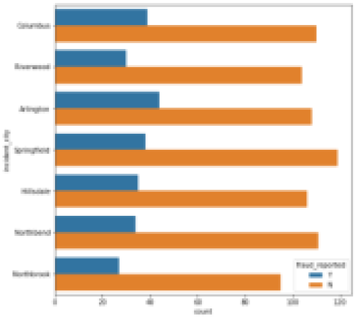




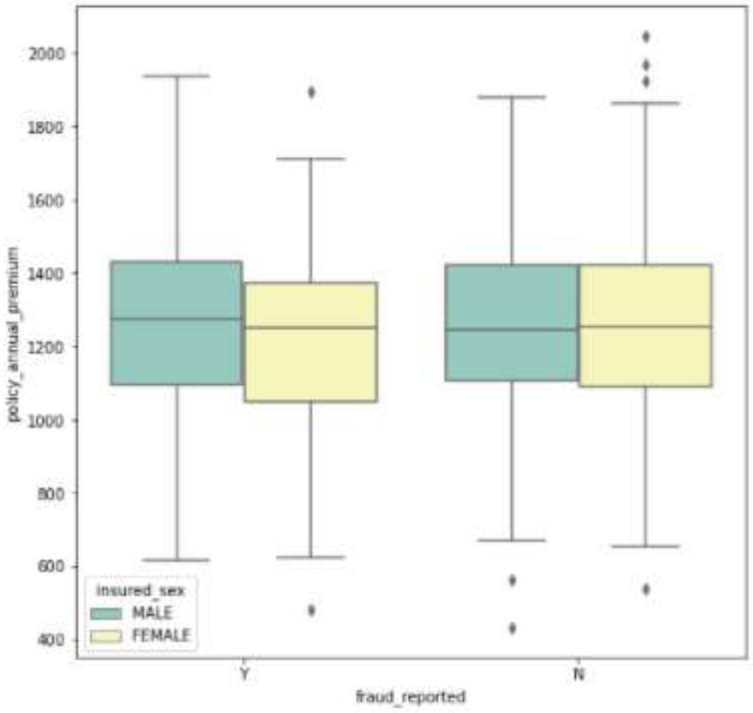
Occupation: People of job exec.manager are doing more frauds.

More frauds happened in state OH and in cities: `Arlington, Columbus`

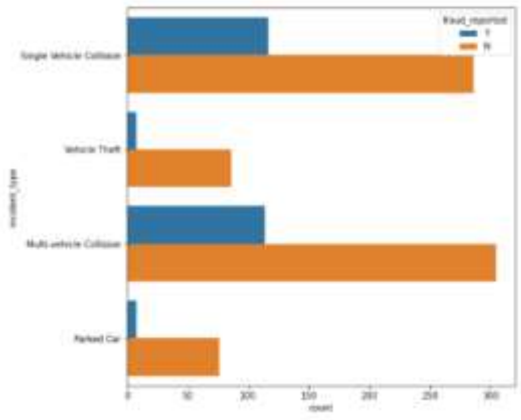


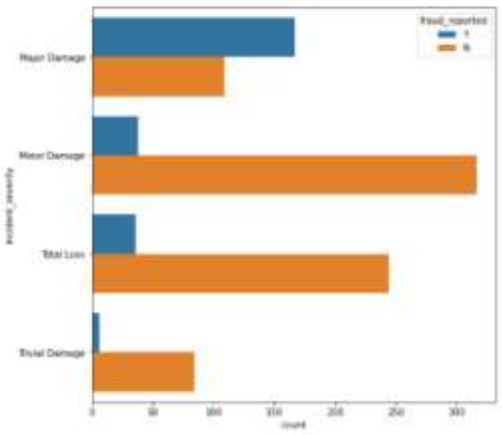


Policy premium: We seen that people with premium of above average are committing frauds

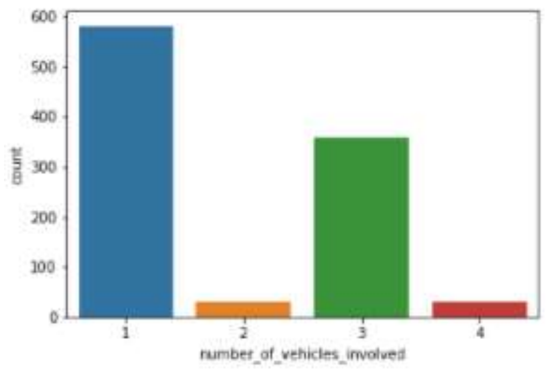


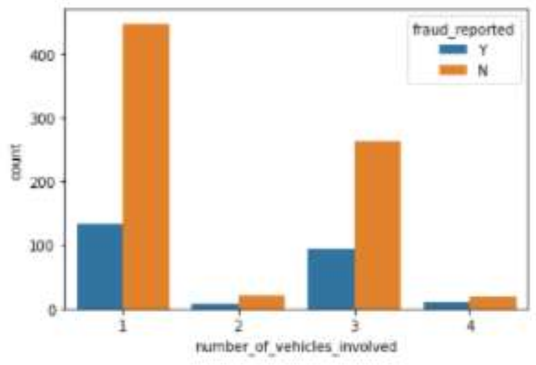
People who claimed incident type as Single vehicle collision or multiple vehicle and major damage collision are more fraudulent claims.



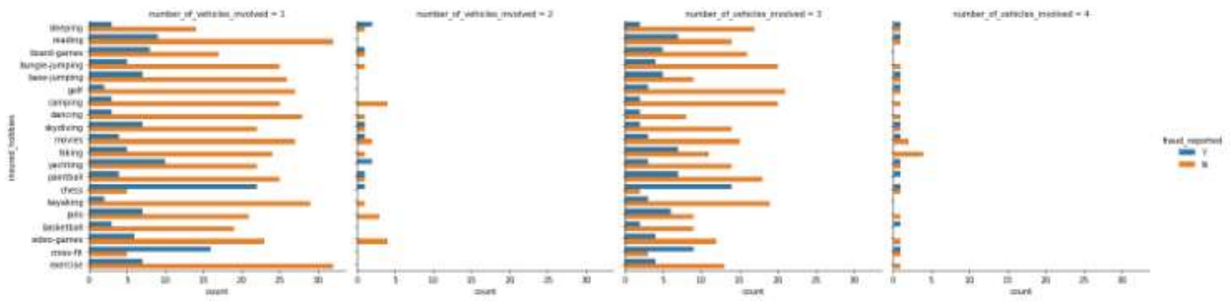


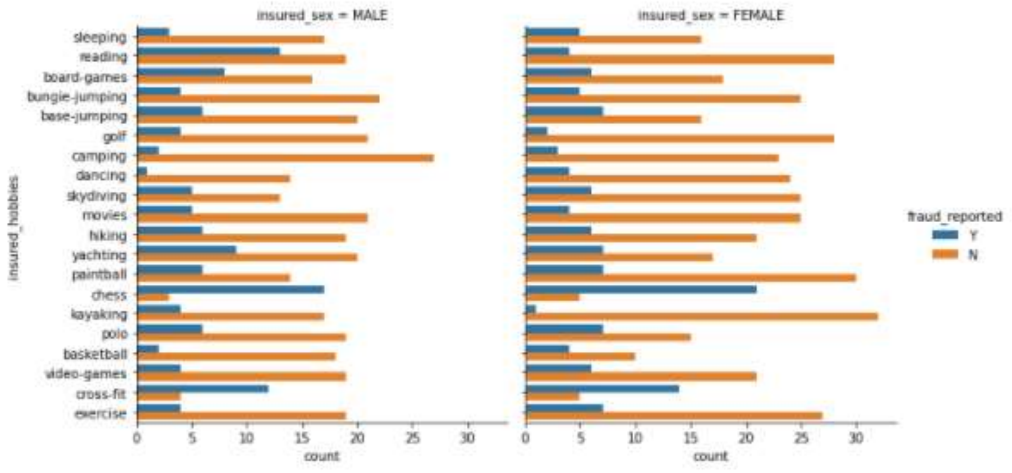
Majority of the incidents are 1 and 3 vehicles, also 1 involved are more fraud claims, but if we see carefully even though 2 and 4 vehicles involved claims are less, 50% of them are fraud.

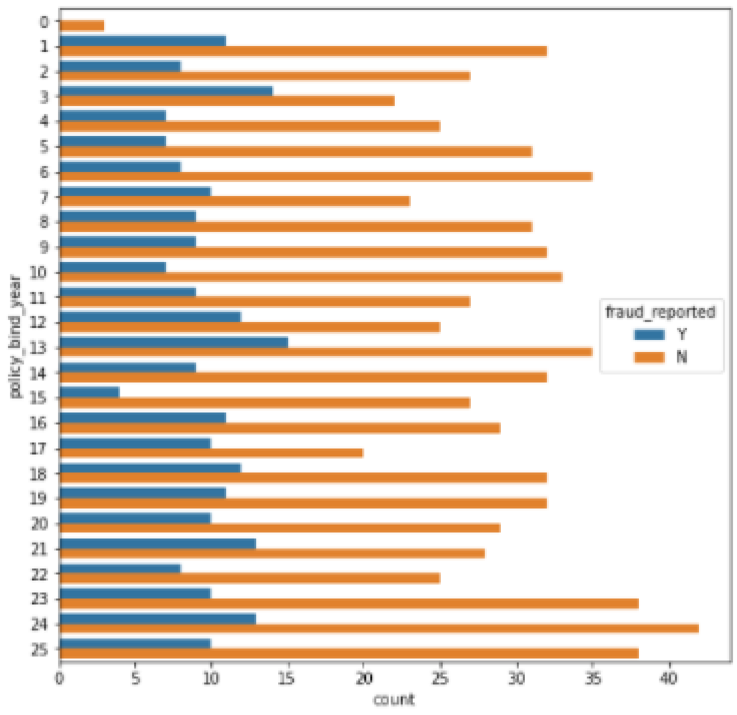




Observing the fraud basis on no. of vehicles and Hobbies , Gender and Hobbies.







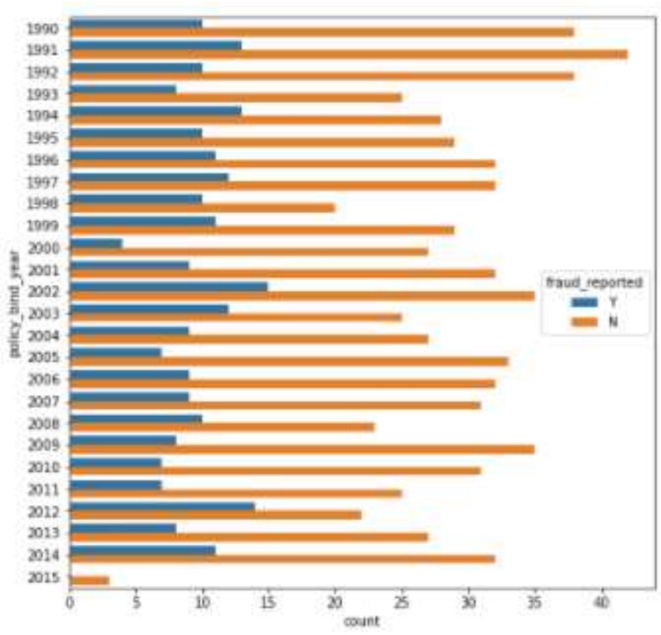
Based on number of years of policy binded

Similarly we observe and get insights from other columns.

We removed two more columns, incident date and incident location, which are not necessary, have almost unique values in each row.

We made a new column policy bind year from policy bind date and dropped the original column.

From that we observed policy been made from the years 1990 to 2015, and more frauds are of those who made insurance in year 1991, 2002, 2012.



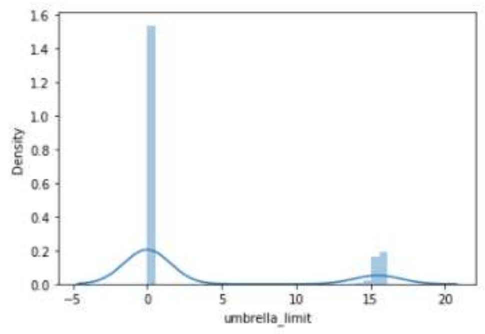
So to make it simple, we found the difference of number of years from 2015.

We made all the columns encoded and find the correlation.

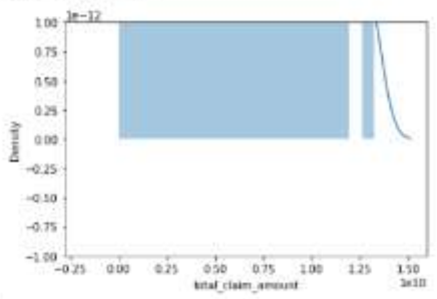
Checking the skewness and reducing skewness in columns umbrella limit, total claim amount

```
dfc.skew()
```

```
months_as_customer             0.362177
age                            0.478988
policy_state                  -0.026177
policy_deductable              0.477887
policy_annual_premium          0.004402
umbrella_limit                 1.806712
insured_sex                    0.148630
insured_education_level       -0.000148
insured_occupation            -0.058881
insured_hobbies               -0.061563
insured_relationship           0.077488
capital-gains                  0.478850
capital-loss                  -0.391472
incident_type                  0.101507
collision_type                -0.033682
incident_severity              0.279016
authorities_contacted         -0.121744
incident_state                -0.148865
incident_city                  0.049531
incident_hour_of_the_day      -0.035584
number_of_vehicles_involved    0.502664
property_damage               -0.685977
bodily_injuries                0.014777
witnesses                      0.019636
police_report_available        0.802728
total_claim_amount            -0.594582
injury_claim                   0.264811
property_claim                 0.378169
vehicle_claim                 -0.621098
fraud_reported                 1.175051
policy_bind_year              -0.052511
dtype: float64
```

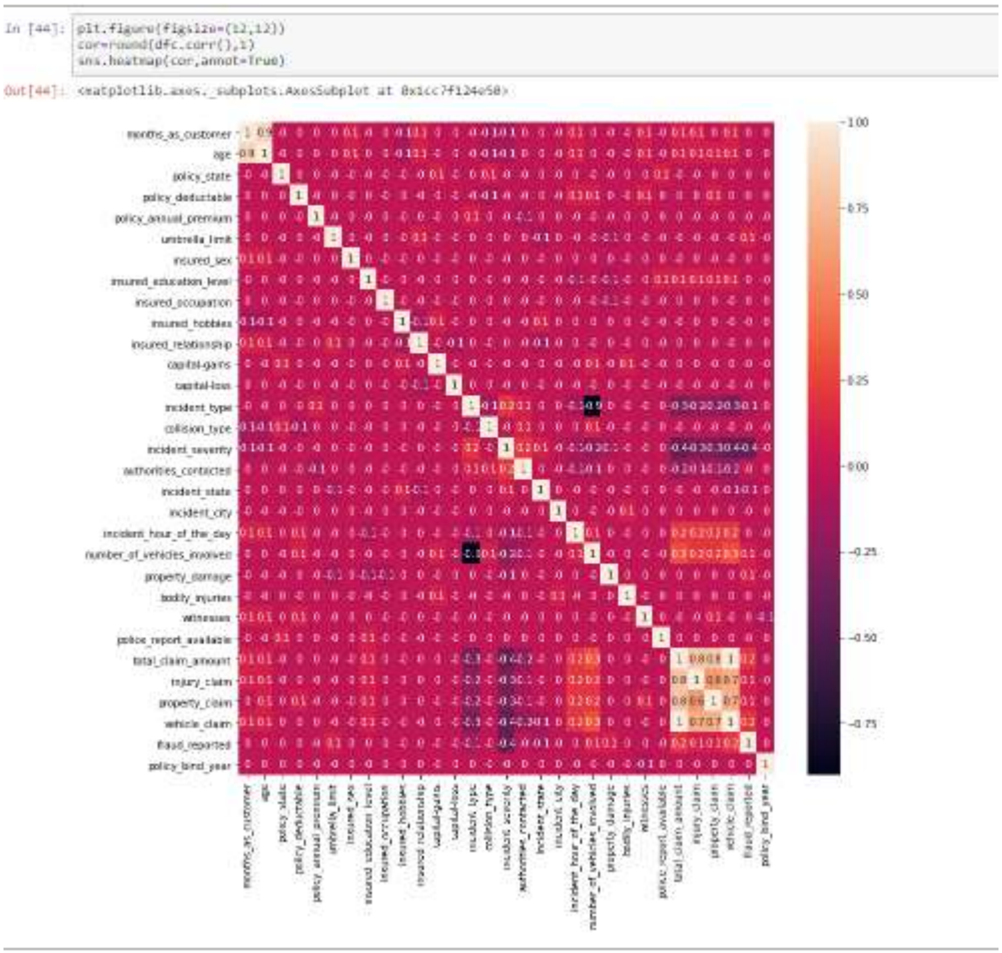


```
0.5330084731063726
```



Skewness reduced to 1.4 from 1.8, Skewness reduced to 0.5 from -0.59

Plot of correlation:

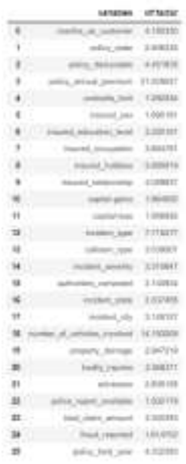


```
In [44]: plt.figure(figsize=(12,12))
         cor=round(dfc.corr(),1)
         sns.heatmap(cor,annot=True)

Out[44]: <matplotlib.axes._subplots.AxesSubplot at 0x1cc7f124e58>
```

Observations:

- Age & months as customer are highly correlated, we will drop the column age.
- Total claim amount, injury claim, property claim, vehicle claim are correlated to each other. So we will drop all those columns except Total claim amount
- Incident hour of the day doesn't have any correlation with fraud. So we will drop it.\
- There are many columns which have no correlation with fraud reported

Let us drop these columns, find the VIF to know the multicollinearity and drop other columns.

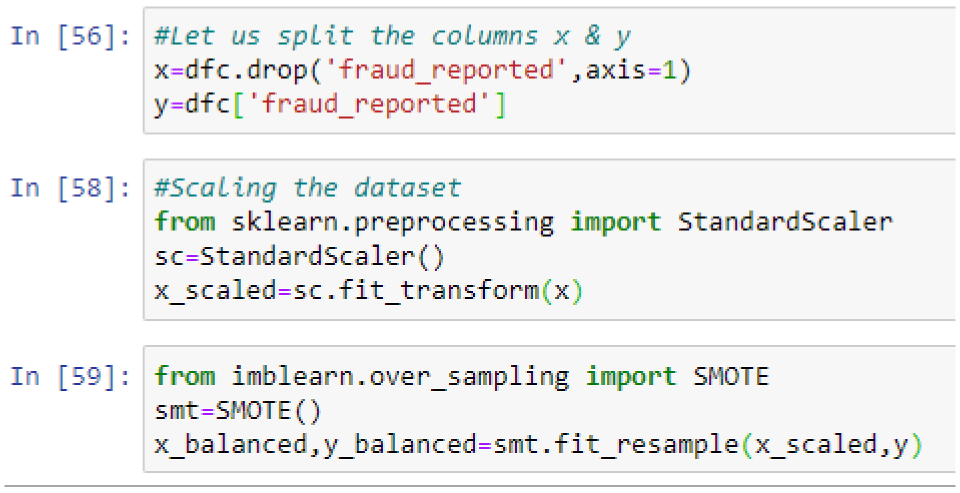


	variables	vif factor
0	months_as_customer	4.180150
1	policy_state	2.808333
2	policy_deductable	4.457838
3	policy_annual_premium	11.308837
4	umbrella_limit	1.292834
5	insured_sex	1.900191
6	insured_education_level	2.220101
7	insured_occupation	3.804781
8	insured_hobbies	3.269416
9	insured_relationship	4.339837
10	capital-gains	1.964832
11	capital-loss	1.958933
12	incident_type	7.778277
13	collision_type	3.518867
14	incident_severity	3.315847
15	authorities_contacted	3.132834
16	incident_state	2.537855
17	incident_city	3.108107
18	number_of_vehicles_involved	14.150308
19	property_damage	2.047219
20	bodily_injuries	2.388271
21	witnesses	2.858108
22	police_report_available	1.932178
23	total_claim_amount	3.330381
24	fraud_reported	1.614752
25	policy_bind_year	4.332381

By viewing VIF I dropped columns

Number of vehicles involved, premium deductables, capital gain and capital loss.

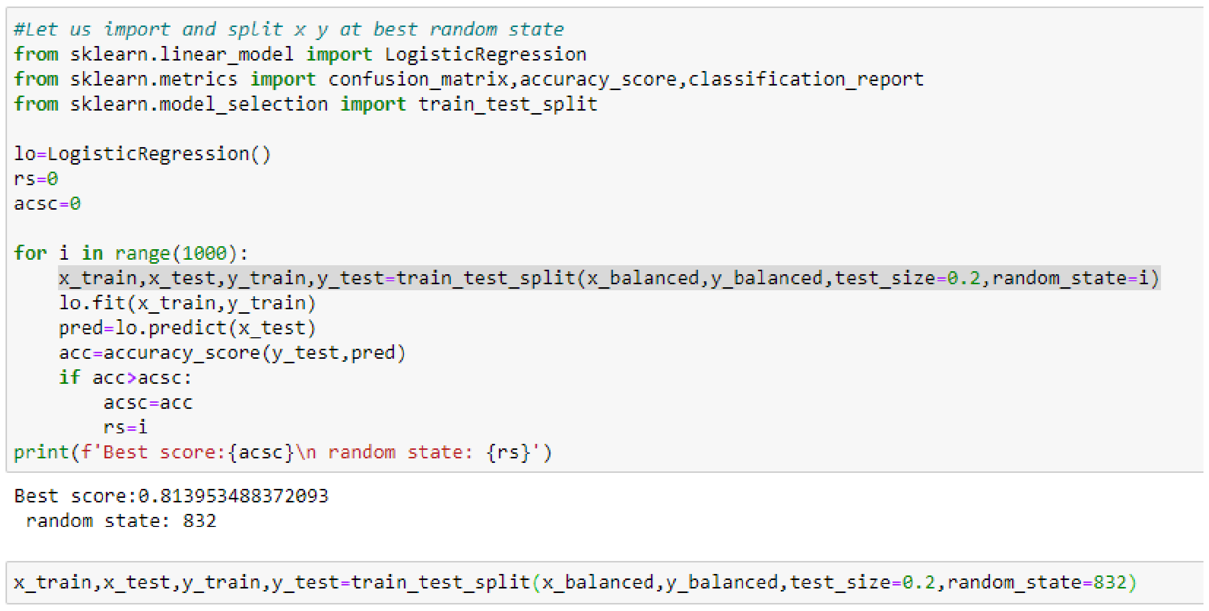
Finally two last steps before making model, Split x & y, then scale the x and finally balance the dataset.



```
In [56]: #Let us split the columns x & y
         x=dfc.drop('fraud_reported',axis=1)
         y=dfc['fraud_reported']

In [58]: #Scaling the dataset
         from sklearn.preprocessing import StandardScaler
         sc=StandardScaler()
         x_scaled=sc.fit_transform(x)

In [59]: from imblearn.over_sampling import SMOTE
         smt=SMOTE()
         x_balanced,y_balanced=smt.fit_resample(x_scaled,y)
```

X is features, y is our target.

Found best random state & split.

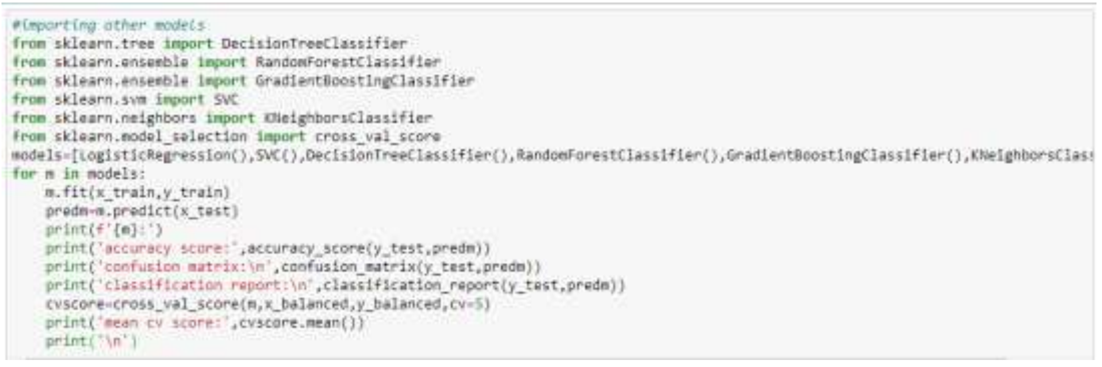


```
#Let us import and split x y at best random state
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix,accuracy_score,classification_report
from sklearn.model_selection import train_test_split

lo=LogisticRegression()
rs=0
acsc=0

for i in range(1000):
    x_train,x_test,y_train,y_test=train_test_split(x_balanced,y_balanced,test_size=0.2,random_state=i)
    lo.fit(x_train,y_train)
    pred=lo.predict(x_test)
    acc=accuracy_score(y_test,pred)
    if acc>acsc:
        acsc=acc
        rs=i
print(f'Best score:{acsc}\n random state: {rs}')
```

```
Best score:0.813953488372093
 random state: 832
```

```
x_train,x_test,y_train,y_test=train_test_split(x_balanced,y_balanced,test_size=0.2,random_state=832)
```

Imported different models to try and select the best one



```
#Importing other models
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
models=[LogisticRegression(),SVC(),DecisionTreeClassifier(),RandomForestClassifier(),GradientBoostingClassifier(),KNeighborsClas
for m in models:
    m.fit(x_train,y_train)
    predm=m.predict(x_test)
    print(f'{m}:')
    print('accuracy score:',accuracy_score(y_test,predm))
    print('confusion matrix:\n',confusion_matrix(y_test,predm))
    print('classification report:\n',classification_report(y_test,predm))
    cvscore=cross_val_score(m,x_balanced,y_balanced,cv=5)
    print('mean cv score:',cvscore.mean())
    print('\n')
```

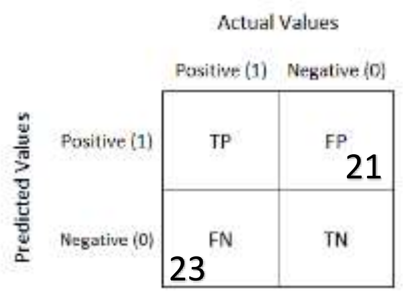
From above we got results of accuracy score, f1 score, evaluation report and cross validation score of different models –

Model	F1 score	Cross validation	Difference
Logistic Regression	81	74.5	6.5
Decision Tree Classifier	85	84.9	0.1
Random Forest Classifier	91	87	4
Gradient Boosting Classifier	91	88	3
K Neighbors Classifier	72	73	1
SVC	89	84.5	4.5

From above we can see, Decision Tree is the best model with least difference between CV score and F1 score. These are the metrics generated:

```
DecisionTreeClassifier():
accuracy score: 0.8538205980066446
confusion matrix:
 [[129  21]
 [ 23 128]]
classification report:
               precision    recall  f1-score   support

           0       0.85      0.86      0.85       150
           1       0.86      0.85      0.85       151

    accuracy                           0.85       301
   macro avg       0.85      0.85      0.85       301
weighted avg       0.85      0.85      0.85       301

mean cv score: 0.8491052048726466
```

From confusion matrix we can see



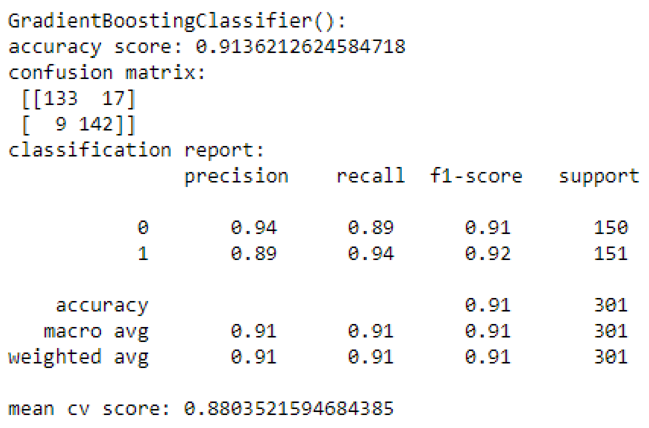
Our model is saying 23 members as not fraud where they are actually frauds. We need to reduce this for better results.

FP is ok because checking whether fraud or not is less cost to company than paying huge amount to the original fraud claim.

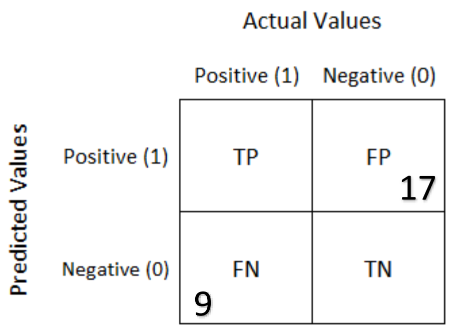
In our case we need least False Negative.

But wait there is almost 6% difference in Decision Tree classifier, Random Forest or Gradient Boosting. Let us observer their metrics once

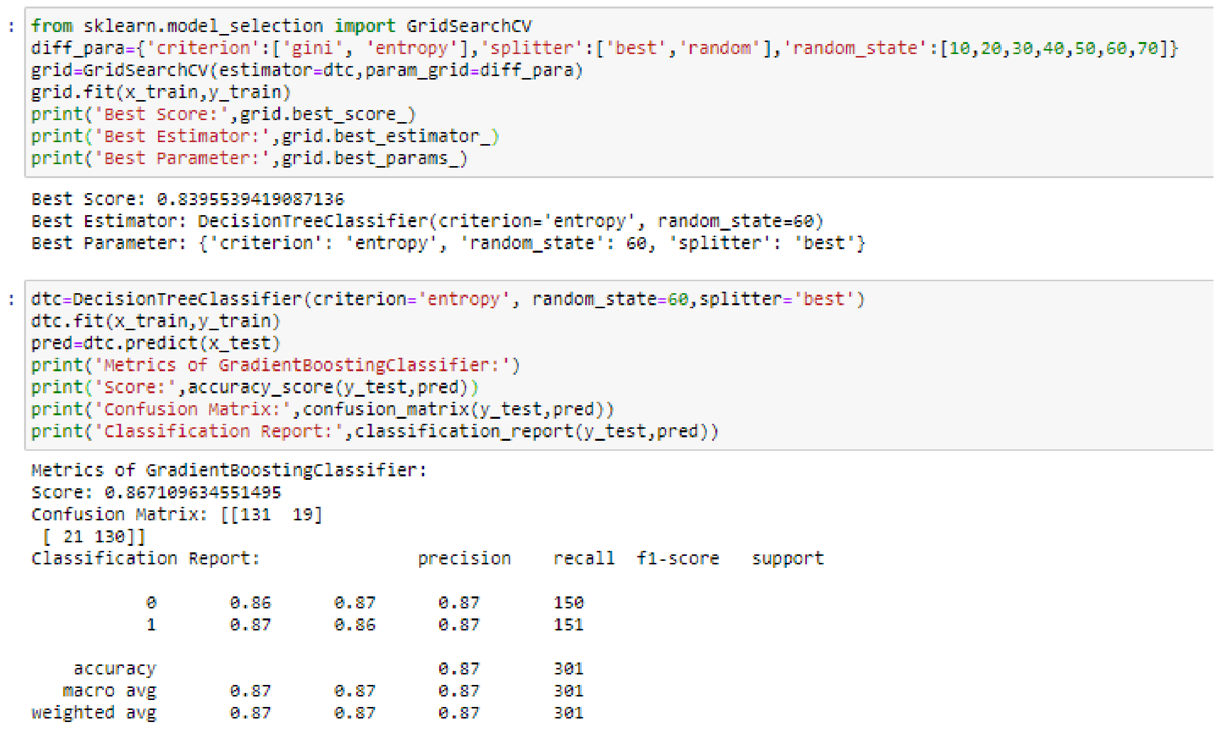
If we see other models: It has 3% difference but giving the least FN



```
GradientBoostingClassifier():
accuracy score: 0.9136212624584718
confusion matrix:
 [[133  17]
 [  9 142]]
classification report:
               precision    recall  f1-score   support

           0       0.94      0.89      0.91       150
           1       0.89      0.94      0.92       151

    accuracy                           0.91       301
   macro avg       0.91      0.91      0.91       301
weighted avg       0.91      0.91      0.91       301

mean cv score: 0.8803521594684385
```



By tuning the decision tree classifier: The model accuracy increased from 85 to 87% only. With FN as 21.



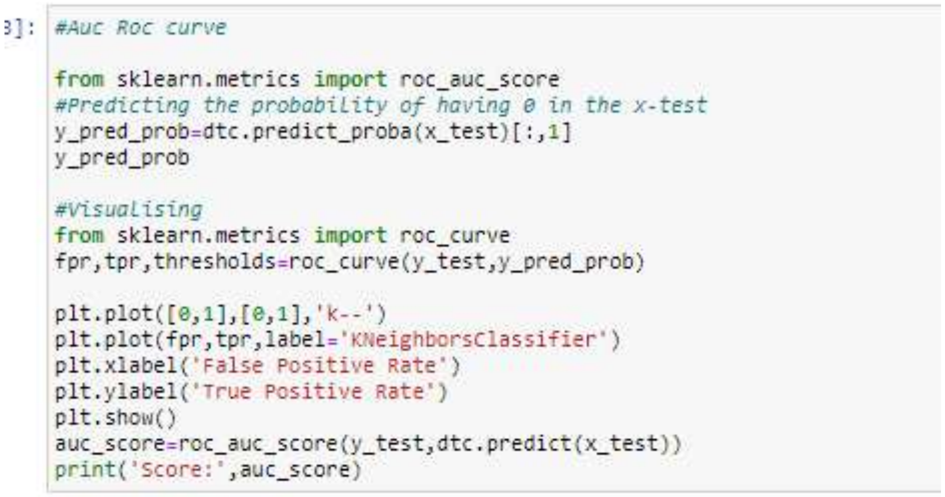
```
from sklearn.model_selection import GridSearchCV
diff_para={'criterion':['gini', 'entropy'],'splitter':['best','random'],'random_state':[10,20,30,40,50,60,70]}
grid=GridSearchCV(estimator=dtc,param_grid=diff_para)
grid.fit(x_train,y_train)
print('Best Score:',grid.best_score_)
print('Best Estimator:',grid.best_estimator_)
print('Best Parameter:',grid.best_params_)
```

```
Best Score: 0.8395539419087136
Best Estimator: DecisionTreeClassifier(criterion='entropy', random_state=60)
Best Parameter: {'criterion': 'entropy', 'random_state': 60, 'splitter': 'best'}
```

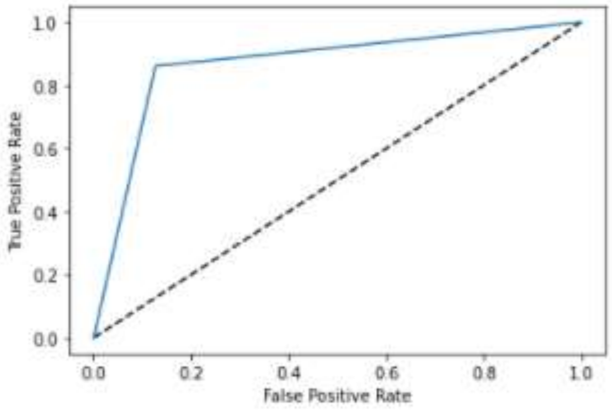
```
dtc=DecisionTreeClassifier(criterion='entropy', random_state=60,splitter='best')
dtc.fit(x_train,y_train)
pred=dtc.predict(x_test)
print('Metrics of GradientBoostingClassifier:')
print('Score:',accuracy_score(y_test,pred))
print('Confusion Matrix:',confusion_matrix(y_test,pred))
print('Classification Report:',classification_report(y_test,pred))
```

```
Metrics of GradientBoostingClassifier:
Score: 0.867109634551495
Confusion Matrix: [[131  19]
 [ 21 130]]
Classification Report:               precision    recall  f1-score   support

           0       0.86      0.87      0.87       150
           1       0.87      0.86      0.87       151

    accuracy                           0.87       301
   macro avg       0.87      0.87      0.87       301
weighted avg       0.87      0.87      0.87       301
```

AUC-ROC:



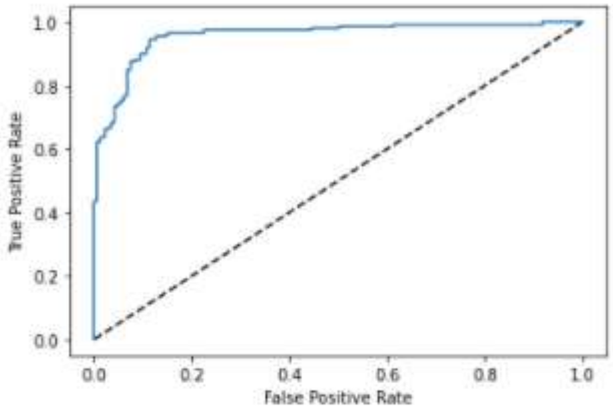
```
3]: #Auc Roc curve

    from sklearn.metrics import roc_auc_score
    #Predicting the probability of having 0 in the x-test
    y_pred_prob=dtc.predict_proba(x_test)[:,1]
    y_pred_prob

    #Visualising
    from sklearn.metrics import roc_curve
    fpr,tpr,thresholds=roc_curve(y_test,y_pred_prob)

    plt.plot([0,1],[0,1],'k--')
    plt.plot(fpr,tpr,label='KNeighborsClassifier')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.show()
    auc_score=roc_auc_score(y_test,dtc.predict(x_test))
    print('Score:',auc_score)
```



```
Score: 0.867130242825607
```

For the Gradient Boosting model, even after tuning there is no improvement in accuracy:

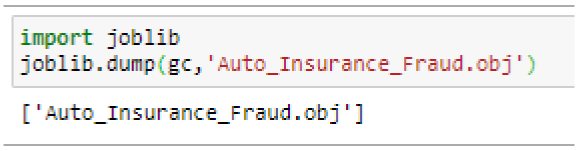
AUC-ROC:



```
Score: 0.9135320088300221
```

I am considering this as my best model and saving it to predict.

Saving the model:



```
import joblib
joblib.dump(gc,'Auto_Insurance_Fraud.obj')
```

```
['Auto_Insurance_Fraud.obj']
```

Conclusion:

Now we made a model, that can predict whether the claim made is Fraud or not with a 91% accuracy.

We have Seen, Gender, number of months as customer, Premium, Hobbies, Education , Type of incident , Incident severity and the policy bind year are the most important columns that help us to predict and understand whether the claim is fraud or not.

- Customers with education of JD are doing more fraud.
- Customers with hobbies of CHESS, CROSS-FIT are doing more fraud.
- Customers who claimed the incident to be single vehicle collision or Multi vehicle collision are more fraud claims.
- Gender is almost same in both the cases of fraud claims
- Premium – Customers who opted for higher premiums done frauds
- Majority of claims made by customers with policy bin year 1991, 2002, and 2012 are frauds.

We encoded the values, categorical columns. Our result is also encoded as 0 and 1 where

0 is NO

1 is YES

Later it will be decoded, when the model is deployed.

Attaching the GitHub link along with this, where you can get dataset, the complete step by step procedure followed jupyternotebook .ipynb file.

https://github.com/Makham-Sai-Girish/Auto_Insurance_Fraud-Claim_detection