INSURANCE CLAIM – FRAUD DETECTION

A MACHINE LEARNING PROJECT

Written By-

Amandeep Singh

https://github.com/AmandeepSinghDhalla/Machine-Learning-Projects/blob/Evaluation/Insurance%20Claim%20Fraud%20Prediction.ipynb

INTRODUCTION TO THE PROBLEM

Insurance fraud can be seen on the rise in the industry. It encompasses the dishonest activities that an individual may perpetrate to achieve the insurance claim from any company. This could range from staging the incident, misrepresenting the situation including the relevant actors and the cause of the incident and finally the extent of damage caused.

According to the FBI, the insurance industry in the USA consists of over 7000 companies that collectively received over $1 trillion annually in premiums. FBI also estimates the total cost of insurance fraud (non-health insurance) to be more than $40 billion annually.

Needless to say, trying to identify such frauds can be quite perplexing. But Machine Learning can aid in making this arduous work undemanding. Using the machine learning models, we can save a ton of time and money in predicting frauds even before they happen. Companies will be safe from being duped by the swindler.

In this project problem, we are provided with a dataset that has the details of the insurance policy along with the customer details. It also has the details of the accident based on which the claims have been made. The dataset can be downloaded from this link.

For this project, we will make the fraud prediction models using machine learning with some auto insurance data that will predict if an insurance claim is fraudulent or not. We will also check the accuracy and the performance of our models to find the best prediction model.

EXPLORATORY DATA ANALYSIS

Before building the any machine learning model, it is paramount that the data used is clean and properly scaled. Exploratory Data Analysis is the very first step of the machine learning project where we will clean and analyze the data using Python libraries like Pandas, NumPy, SciPy etc. It is also imperative that we find out the information from the data regarding the different factors that can result in the fraud claims. The information gain should be unbiased information towards anyone. For this, we will use data visualization techniques and plot the plots, graphs and figures using the visualization libraries like Matplotlib, Seaborn, Plotly etc.

We start the EDA with importing the important libraries onto the Jupyter notebook and then load the dataset in notebook from the provided link.

```
'''Importing Important Libraires'''
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

```
data = pd.read_csv("https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_
data
```

	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
...	...	...	...	...	...	...	...	...	...	...
995	3	38	941851	16-07-1991	OH	500/1000	1000	1310.80	0	431289
996	285	41	186934	05-01-2014	IL	100/300	1000	1436.79	0	608177
997	130	34	918516	17-02-2003	OH	250/500	500	1383.49	3000000	442797
998	458	62	533940	18-11-2011	IL	500/1000	2000	1356.92	5000000	441714
999	456	60	556080	11-11-1996	OH	250/500	1000	766.19	0	612260

1000 rows × 40 columns

The provided data consist of 1000 observations and for each observation there are 40 attributes. We make a copy of the pandas data frame to work on, then change the display options to view all the features of the data frame.

```
pd.set_option('display.max_columns', 100)
ds = data.copy()
ds.head()
```

ries	witnesses	police_report_available	total_claim_amount	injury_claim	property_claim	vehicle_claim	auto_make	auto_model	auto_year	fraud_reported	_c39
1	2	YES	71610	6510	13020	52080	Saab	92x	2004	Y	NaN
0	0	?	5070	780	780	3510	Mercedes	E400	2007	Y	NaN
2	3	NO	34650	7700	3850	23100	Dodge	RAM	2007	N	NaN
1	2	NO	63400	6340	6340	50720	Chevrolet	Tahoe	2014	Y	NaN
0	1	NO	6500	1300	650	4550	Accura	RSX	2009	N	NaN

In this problem data, we have to predict the fraud in the Insurance Claim for all the observations. We are given the target class 'fraud_reported' in the dataset which from the look of it consist of 2 distinct values, 'Y' and 'N'. So, this is a type of binary classification problem. Hence, we will build the classifier ml models.

The features of the data are of both continuous and categorical type. First, we use the descriptive statistics to look statistical functions of the continuous features.

```
ds.describe()
```

	months_as_customer	age	policy_number	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-lo
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	1000.000000	1000.000000	1000.0000
mean	203.954000	38.948000	546238.648000	1136.000000	1256.406150	1.101000e+06	501214.488000	25126.100000	-26793.7000
std	115.113174	9.140287	257063.005276	611.864673	244.167395	2.297407e+06	71701.610941	27872.187708	28104.0966
min	0.000000	19.000000	100804.000000	500.000000	433.330000	-1.000000e+06	430104.000000	0.000000	-111100.0000
25%	115.750000	32.000000	335980.250000	500.000000	1089.607500	0.000000e+00	448404.500000	0.000000	-51500.0000
50%	199.500000	38.000000	533135.000000	1000.000000	1257.200000	0.000000e+00	466445.500000	0.000000	-23250.0000
75%	276.250000	44.000000	759099.750000	2000.000000	1415.695000	0.000000e+00	603251.000000	51025.000000	0.0000
max	479.000000	64.000000	999435.000000	2000.000000	2047.590000	1.000000e+07	620962.000000	100500.000000	0.0000

Some information can be gathered from the descriptive statistics. The minimum age of insurance client is 19 and maximum age is 64. Also, the oldest customer is 479 months old. Similarly other information can be gathered from above descriptive statistics.

We see that for the feature 'umbrella_limit', some negative values are present in the observations. Umbrella limit is often referred to as excess liability insurance. If a policyholder is sued for damages that exceed the liability limits of insurance coverage types, an umbrella policy helps pay what they owe. So, the negative values in feature 'umbrella_limit' are invalid. Observations with these values will be removed from the data frame.

```
ds.loc[ds['umbrella_limit'] < 0]
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
290	284	42	526039	04-05-1995	OH	100/300	500	1338.54	-1000000	438178

```
# only one observation with invalid value present
# removing invalid value
ds.drop([290], inplace = True)
```

Once the invalid values are removed, we have a look at the info of the data frame.

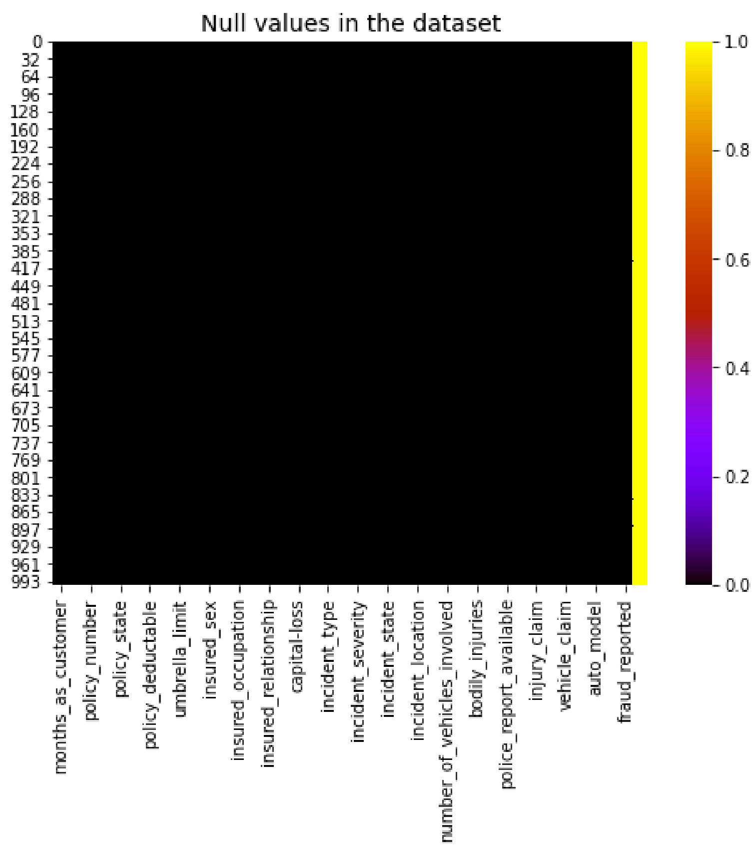
```
ds.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 999 entries, 0 to 999
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   months_as_customer           999 non-null    int64
 1   age                          999 non-null    int64
 2   policy_number                999 non-null    int64
 3   policy_bind_date             999 non-null    object
 4   policy_state                 999 non-null    object
 5   policy_csl                   999 non-null    object
 6   policy_deductable            999 non-null    int64
 7   policy_annual_premium        999 non-null    float64
 8   umbrella_limit               999 non-null    int64
 9   insured_zip                  999 non-null    int64
 10  insured_sex                  999 non-null    object
 11  insured_education_level      999 non-null    object
 12  insured_occupation           999 non-null    object
 13  insured_hobbies              999 non-null    object
 14  insured_relationship         999 non-null    object
 15  capital-gains                999 non-null    int64
 16  capital-loss                 999 non-null    int64
 17  incident_date                999 non-null    object
 18  incident_type                999 non-null    object
 19  collision_type               999 non-null    object
 19  collision_type               999 non-null    object
 20  incident_severity            999 non-null    object
 21  authorities_contacted        999 non-null    object
 22  incident_state               999 non-null    object
 23  incident_city                999 non-null    object
 24  incident_location            999 non-null    object
 25  incident_hour_of_the_day     999 non-null    int64
 26  number_of_vehicles_involved  999 non-null    int64
 27  property_damage              999 non-null    object
 28  bodily_injuries              999 non-null    int64
 29  witnesses                    999 non-null    int64
 30  police_report_available      999 non-null    object
 31  total_claim_amount           999 non-null    int64
 32  injury_claim                 999 non-null    int64
 33  property_claim               999 non-null    int64
 34  vehicle_claim                999 non-null    int64
 35  auto_make                    999 non-null    object
 36  auto_model                   999 non-null    object
 37  auto_year                    999 non-null    int64
 38  fraud_reported               999 non-null    object
 39  _c39                         0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 320.0+ KB
```

There is a total of 40 features in the data frame that includes the target class which is object datatype. Other than that, there are 2 float, 17 int and 21 object type features. The cleaning of numerical features and encoding of categorical features will be done in the coming steps of EDA.

Null Values Handling: If there are any null values present in the data frame before using it to build the ml model, then this will affect the performance of the models. So, we treat the null values present in any dataset. We use the heatmap to see if there are any of these values present in the data.

```
plt.figure(figsize = (8,6))
plt.title("Null values in the dataset", fontsize = 14)
sns.heatmap(ds.isnull(), cmap = 'gnuplot')
plt.show()
```



From the above heatmap, we see that the last column '_c39' comprises of only null values. We will remove it from the data. Also, there are no null values present in the other columns of the datasets.

```
ds.drop(['_c39'], axis = 1, inplace = True)
```

Feature Engineering and Cleaning: Once we have handled the null values, it's time to move ahead to data cleaning and feature engineering. To perform this step, it would be a good idea to look at the number of unique values for each feature. This will help us to identify if any feature is unique or has very low variance or is completely constant throughout the observations.

```
# Number of unique variables in each feature.
ds.nunique()
```

```
months_as_customer             391
age                             46
policy_number                  999
policy_bind_date               950
policy_state                     3
policy_csl                       3
policy_deductable                3
policy_annual_premium          990
umbrella_limit                  10
insured_zip                    994
insured_sex                      2
insured_education_level          7
insured_occupation              14
insured_hobbies                 20
insured_relationship             6
capital-gains                  338
capital-loss                   354
incident_date                   60
incident_type                    4
collision_type                   4
incident_severity                4
authorities_contacted            5
incident_state                   7
incident_city                    7
incident_location              999
incident_hour_of_the_day        24
number_of_vehicles_involved      4
property_damage                  3
bodily_injuries                  3
witnesses                        4
police_report_available          3
total_claim_amount             762
injury_claim                   638
property_claim                 625
vehicle_claim                  725
auto_make                       14
auto_model                      39
auto_year                       21
fraud_reported                   2
dtype: int64
```

'policy_number', 'insured_zip' and 'incident_location' are features that have unique different values for every observation. So, they will be removed from the data. The columns 'policy_bind_date' and 'incident_date' consist of dates which is not useful for this case study. These columns will be removed as well. For the detection of the fraud, the 'auto_make' can be useful, but 'auto_model' which tells the model of the vehicle will increase the cardinality. We will also remove the 'auto_model' from the data frame.

```
ds.drop(['policy_number', 'incident_location', 'insured_zip',
         'policy_bind_date', 'incident_date', 'auto_model'],
        axis = 1, inplace = True)
```

Moving ahead, we have a look at the value counts of all the remaining categorical features. This will help us in identifying the best technique for encoding of the attributes.

```python
categorical_columns = []
for i in ds.columns:
    if ds[i].dtype == 'object':
        categorical_columns.append(i) # appending all the categorical features

for i in categorical_columns:
    print(ds[i].value_counts(),"\n") # printing value counts for categorical features
```

```
OH    351
IL    338
IN    310
Name: policy_state, dtype: int64

250/500     351
100/300     348
500/1000    300
Name: policy_csl, dtype: int64

FEMALE    537
MALE      462
Name: insured_sex, dtype: int64

JD             161
High School    160
Associate      144
MD             144
Masters        143
PhD            125
College        122
Name: insured_education_level, dtype: int64

machine-op-inspct    92
prof-specialty       85
tech-support         78
sales                76
exec-managerial      76
craft-repair         74
transport-moving     72
other-service        71
priv-house-serv      71
armed-forces         69
adm-clerical         65
protective-serv      63
handlers-cleaners    54
farming-fishing      53
Name: insured_occupation, dtype: int64

reading           64
paintball         57
exercise          57
bungie-jumping    56
camping           55
movies            55
golf              55
yachting          53
kayaking          53
hiking            52
video-games       50
base-jumping      49
skydiving         49
board-games       48
polo              47
chess             46
dancing           43
sleeping          41
cross-fit         35
basketball        34
Name: insured_hobbies, dtype: int64

own-child         183
other-relative    177
not-in-family     174
husband           170
wife              154
unmarried         141
Name: insured_relationship, dtype: int64

Multi-vehicle Collision     419
Single Vehicle Collision    402
Vehicle Theft                94
Parked Car                   84
Name: incident_type, dtype: int64

Rear Collision     292
Side Collision     275
Front Collision    254
?                  178
Name: collision_type, dtype: int64

Minor Damage      354
Total Loss        280
Major Damage      275
Trivial Damage     90
Name: incident_severity, dtype: int64

Police       292
Fire         223
Other        198
Ambulance    195
None          91
Name: authorities_contacted, dtype: int64

NY    262
SC    248
WV    217
VA    110
NC    109
PA     30
OH     23
Name: incident_state, dtype: int64

Springfield    157
Arlington      151
Columbus       149
Northbend      145
Hillsdale      141
Riverwood      134
Northbrook     122
Name: incident_city, dtype: int64

?      360
NO     338
YES    301
Name: property_damage, dtype: int64

NO     343
?      342
YES    314
Name: police_report_available, dtype: int64

Suburu        80
Dodge         80
Saab          80
Nissan        78
Chevrolet     75
Ford          72
BMW           72
Toyota        70
Audi          69
Accura        68
Volkswagen    68
Jeep          67
Mercedes      65
Honda         55
Name: auto_make, dtype: int64

N    752
Y    247
Name: fraud_reported, dtype: int64
```

After having a look at the value counts for categorical features, I see that the features ‘police_report_available’, ’property_damage’, ‘collision_type’ have values given as '?'. These values are unknown values and we first replace them with NaN.

```
ds['police_report_available'].replace('?', np.NaN, inplace = True)
ds['property_damage'].replace('?', np.NaN, inplace = True)
ds['collision_type'].replace('?', np.NaN, inplace = True)
```

Now, some new null values are present in these features. To replace the null values in the features, we will use the mode function from SciPy library where the values will be replaced with the mode of the particular column.

But simply replacing the null values with mode will not be the right operation according to me. Let’s look at the features one by one and handle NaN in them.

police_report_available’- We use the pivot table to get the mode of column ‘police_report_available’ after grouping it by ‘authorities_contacted’.

```
from scipy.stats import mode

# geting mode of 'police_report_available' after grouping observations by 'authorities_contacted'
police_report_available_mode = ds.pivot_table(values='police_report_available',
                                              columns='authorities_contacted',
                                              aggfunc=(lambda x:mode(x).mode[0]))
police_report_available_mode
```

authorities_contacted	Ambulance	Fire	None	Other	Police
police_report_available	NO	NO	NO	YES	NO

We use ‘police_report_available_mode’ to fill the NaN values with mode. For that we use the apply function.

```
# replacing the null values with the mode of 'police_report_available'
loc1 = ds['police_report_available'].isnull()
ds.loc[loc1, 'police_report_available'] = ds.loc[loc1, 'authorities_contacted'].apply(lambda x: police_report_available_mode[x])
```

’property_damage’- Similarly we fill the NaN values with mode after grouping this feature by ‘incident_severity’ column.

```python
# geting mode of 'property_damage' after grouping observations by 'incident_severity'
property_damage_mode = ds.pivot_table(values = 'property_damage',
                                      columns = 'incident_severity',
                                      aggfunc = (lambda x:mode(x).mode[0]))
property_damage_mode
```

incident_severity	Major Damage	Minor Damage	Total Loss	Trivial Damage
property_damage	YES	NO	NO	NO

```python
loc2 = ds['property_damage'].isnull()
ds.loc[loc2, 'property_damage'] = ds.loc[loc2, 'incident_severity'].apply(lambda x: property_damage_mode[x])
```

'collision_type'- Getting mode of this column after grouping it by 'incident_type'.

```python
# geting mode of 'collision_type' after grouping observations by 'incident_type'
collision_type_mode = ds.pivot_table(values = 'collision_type',
                                     columns = 'incident_type',
                                     aggfunc = (lambda x: mode(x).mode[0]))
collision_type_mode
```

incident_type	Multi-vehicle Collision	Parked Car	Single Vehicle Collision	Vehicle Theft
collision_type	Rear Collision	0	Rear Collision	0

We see from the mode of 'collision_type' feature that still some places have null values present in the mode table. So, for this column, we cannot replace the values with mode after grouping. Hence, we will create assign new value 'Unknown' to all the null values in this feature.

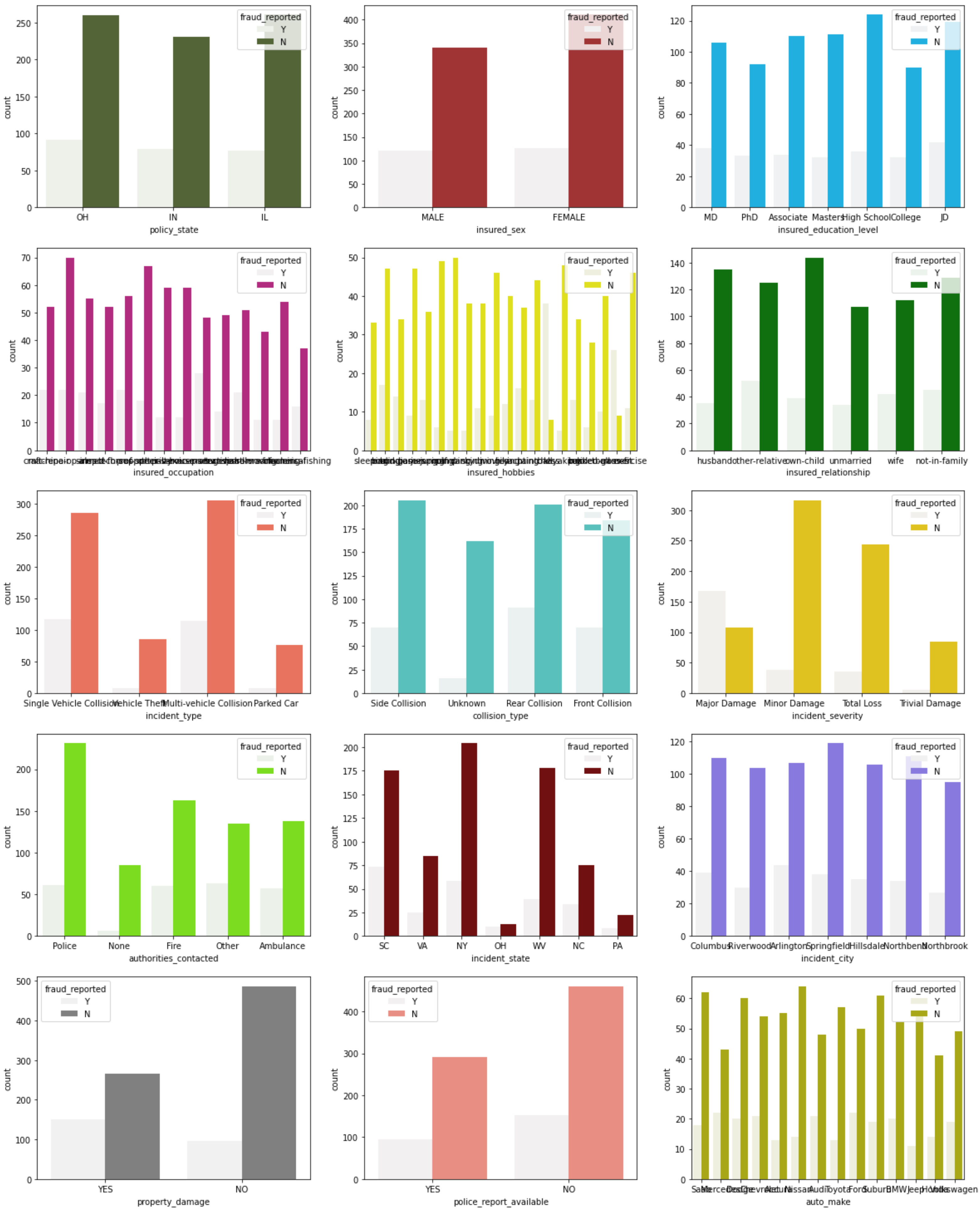
```python
ds['collision_type'].replace(np.NaN, 'Unknown',inplace = True)
```

Relation between categorical features and target:

Data visualization gives us information and facts related to the data and problem at hand that would otherwise have been impossible to get. In this step of EDA, we will first plot the count plots of different categorical features and see how do they affect the insurance fraud claims and to what extent. To plot the graphs, we use the seaborn library.

```python
categorical_list = ['policy_state', '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']

colour_list = ['darkolivegreen', 'firebrick', 'deepskyblue', 'mediumvioletred', 'yellow', 'green', 'tomato', 'mediumturquoise',
               'gold', 'lawngreen', 'maroon', 'mediumslateblue', 'grey', 'salmon', 'y']

plt.figure(figsize = (20,26))
plt.suptitle("Plots for relation between Fraud Claims and different features", fontsize = 24)

for i in range(0, len(categorical_list)):
    plt.subplot(5, 3, i+1)
    sns.countplot(x = ds[categorical_list[i]], hue = ds['fraud_reported'], color = colour_list[i])
```

Plots for relation between Fraud Claims and different features



From the above count plots for the categorical features, various types of insights can be noted for different features and how do they affect the target class. For the majority of claims, a police report is not available and there is no property damage. The highest number of insurance claims are made from New York, South Carolina and WV. Ohio State and PA reported the least number of insurance claims. After the incident occurred, the police were the authorities that were contacted the most followed by the fire department. For a very smaller number of incident cases, no authorities were contacted. The majority of the accidents are single or multi-vehicle collision which resulted in either minor damage or total damage for the greatest number of cases. Also, the highest number of insurance claims were made by women.

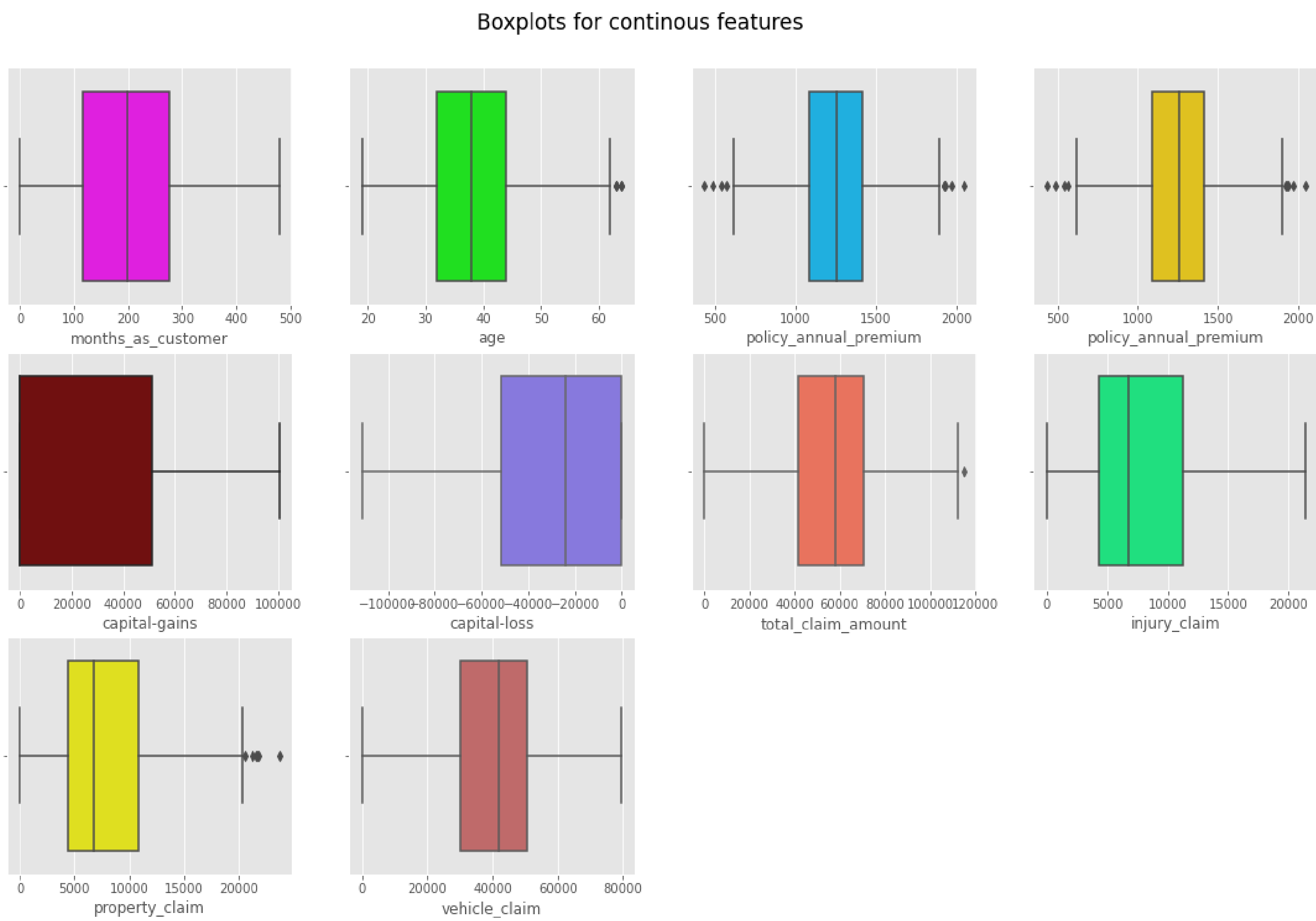
All the above information is gathered with the help of the data visualization techniques, and as a result, we are able to analyze the relationship between the features and the target class.

Outliers detection and handling:

Outliers are the unrealistic or invalid values that are present. Their presence can be a result of various factors like observation fault made by machine or they can be a result of incorrect data gathered by the analysts. Anyhow, it is very important to detect and remove or replace the outliers from the problem data before the ml model building. To detect them, we will first look at the boxplots for the continuous features.

```
plt.style.use('ggplot')
```

```
continous_list = ['months_as_customer', 'age', 'policy_annual_premium', 'policy_annual_premium', 'capital-gains',
                  'capital-loss', 'total_claim_amount', 'injury_claim', 'property_claim', 'vehicle_claim']

colour_list = ['magenta', 'lime', 'deepskyblue', 'gold', 'maroon', 'mediumslateblue', 'tomato', 'springgreen', 'yellow',
               'indianred']

plt.figure(figsize = (19, 12))
plt.suptitle("Boxplots for continous features", fontsize = 17)

for i in range(0, len(continous_list)):
    plt.subplot(3,4,i+1)
    sns.boxplot(ds[continous_list[i]], color = colour_list[i])
```



The box plots for the continuous features in the dataset shows that the outliers are very less or may not be present for the continuous features, so we do not need to treat them.

Feature Encoding: In the previous step, we have cleaned and analyzed both the categorical and continuous columns in the data frame. After looking at the value counts for different columns, we will encode the features in following manner,

1. Frequency Encoding-
    - insured_education_level
    - insured_relationship
    - incident_type
    - authorities_contacted
    - incident_state

- incident_city
- collision_type

2. Ordinal Encoding-

- incident_severity
- insured_sex
- insured_hobbies
- property_damage
- police_report_available

3. One-hot Encoding-

- policy_state

The target variable fraud_reported will be encoded using label encoding.

```
# frequency encoding 'insured_education_level'
insured_education_level_enc = (ds.groupby('insured_education_level').size()) / len(ds)
print(insured_education_level_enc)

ds['insured_education_level'] = ds['insured_education_level'].apply(lambda x: insured_education_level_enc[x])
ds['insured_education_level'].head()
```

```
insured_education_level
Associate      0.144144
College        0.122122
High School    0.160160
JD             0.161161
MD             0.144144
Masters        0.143143
PhD            0.125125
dtype: float64

0    0.144144
1    0.144144
2    0.125125
3    0.125125
4    0.144144
Name: insured_education_level, dtype: float64
```

```
# frequency encoding 'insured_education_level'
insured_relationship_enc = (ds.groupby('insured_relationship').size()) / len(ds)
print(insured_relationship_enc)

ds['insured_relationship'] = ds['insured_relationship'].apply(lambda x: insured_relationship_enc[x])
ds['insured_relationship'].head()
```

```
insured_relationship
husband           0.170170
not-in-family     0.174174
other-relative    0.177177
own-child         0.183183
unmarried         0.141141
wife              0.154154
dtype: float64

0    0.170170
1    0.177177
2    0.183183
3    0.141141
4    0.141141
Name: insured_relationship, dtype: float64
```

```python
# frequency encoding 'incident_type'
incident_type_enc = (ds.groupby('incident_type').size()) / len(ds)
print(incident_type_enc)

ds['incident_type'] = ds['incident_type'].apply(lambda x: incident_type_enc[x])
ds['incident_type'].head()
```

```
incident_type
Multi-vehicle Collision     0.419419
Parked Car                  0.084084
Single Vehicle Collision    0.402402
Vehicle Theft               0.094094
dtype: float64

0    0.402402
1    0.094094
2    0.419419
3    0.402402
4    0.094094
Name: incident_type, dtype: float64
```

```python
# frequency encoding 'authorities_contacted'
authorities_contacted_enc = (ds.groupby('authorities_contacted').size()) / len(ds)
print(authorities_contacted_enc)

ds['authorities_contacted'] = ds['authorities_contacted'].apply(lambda x: authorities_contacted_enc[x])
ds['authorities_contacted'].head()
```

```
authorities_contacted
Ambulance    0.195195
Fire         0.223223
None         0.091091
Other        0.198198
Police       0.292292
dtype: float64

0    0.292292
1    0.292292
2    0.292292
3    0.292292
4    0.091091
Name: authorities_contacted, dtype: float64
```

```python
# frequency encoding 'incident_state'
incident_state_enc = (ds.groupby('incident_state').size()) / len(ds)
print(incident_state_enc)

ds['incident_state'] = ds['incident_state'].apply(lambda x: incident_state_enc[x])
ds['incident_state'].head()
```

```
incident_state
NC    0.109109
NY    0.262262
OH    0.023023
PA    0.030030
SC    0.248248
VA    0.110110
WV    0.217217
dtype: float64

0    0.248248
1    0.110110
2    0.262262
3    0.023023
4    0.262262
Name: incident_state, dtype: float64
```

```python
# frequency encoding 'incident_city'
incident_city_enc = (ds.groupby('incident_city').size()) / len(ds)
print(incident_city_enc)

ds['incident_city'] = ds['incident_city'].apply(lambda x: incident_city_enc[x])
ds['incident_city'].head()
```

```
incident_city
Arlington      0.151151
Columbus       0.149149
Hillsdale      0.141141
Northbend      0.145145
Northbrook     0.122122
Riverwood      0.134134
Springfield    0.157157
dtype: float64

0    0.149149
1    0.134134
2    0.149149
3    0.151151
4    0.151151
Name: incident_city, dtype: float64
```

```python
# frequency encoding 'collision_type'
collision_type_enc = (ds.groupby('collision_type').size()) / len(ds)
print(collision_type_enc)

ds['collision_type'] = ds['collision_type'].apply(lambda x: collision_type_enc[x])
ds['collision_type'].head()
```

```
collision_type
Front Collision    0.254254
Rear Collision     0.292292
Side Collision     0.275275
Unknown            0.178178
dtype: float64

0    0.275275
1    0.178178
2    0.292292
3    0.254254
4    0.178178
Name: collision_type, dtype: float64
```

```python
# encoding 'incident_severity'
severity_map = {'Trivial Damage': 0, 'Minor Damage': 1, 'Major Damage': 2, 'Total Loss': 3}

ds['incident_severity'] = ds['incident_severity'].map(severity_map)
```

```python
# encoding 'insured_sex'
sex_map = {'MALE': 1, 'FEMALE': 0}

ds['insured_sex'] = ds['insured_sex'].map(sex_map)
```

```python
from sklearn.preprocessing import OrdinalEncoder
```

```python
oe = OrdinalEncoder()
# encoding 'insured_hobbies'
ds['insured_hobbies'] = oe.fit_transform(ds[['insured_hobbies']])

# encoding 'auto_make'
ds['auto_make'] = oe.fit_transform(ds[['auto_make']])

# encoding 'insured_occupation'
ds['insured_occupation'] = oe.fit_transform(ds[['insured_occupation']])
```

```
# encoding 'property_damage'
ds['property_damage'] = ds['property_damage'].map({'YES':1,'NO':0})

# encoding 'police_report_available'
ds['police_report_available'] = ds['police_report_available'].map({'YES':1,'NO':0})
```

```
# encoding 'policy_state' using one-hot encoding
ds = pd.get_dummies(ds, columns = ['policy_state'], drop_first = True)
```

```
# encoding target using label encoding
ds['fraud_reported'] = ds['fraud_reported'].map({'Y':1,'N':0})
```

Once we have encoded all the categorical features and the target class in the data frame, all the columns of the data now are continuous or numerical. We can have a look at the dataset.

```
print("Dataset after the encoding")
ds
```

```
Dataset after the encoding
```

	months_as_customer	age	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occupation
0	328	48	250/500	1000	1406.91	0	1	0.144144	2.0
1	228	42	250/500	2000	1197.22	5000000	1	0.144144	6.0
2	134	29	100/300	2000	1413.14	5000000	0	0.125125	11.0
3	256	41	250/500	2000	1415.74	6000000	0	0.125125	1.0
4	228	44	500/1000	1000	1583.91	6000000	1	0.144144	11.0
...	...	...	...	...	...	...	...	...	...
995	3	38	500/1000	1000	1310.80	0	0	0.143143	2.0
996	285	41	100/300	1000	1436.79	0	0	0.125125	9.0
997	130	34	250/500	500	1383.49	3000000	0	0.143143	1.0
998	458	62	500/1000	2000	1356.92	5000000	1	0.144144	5.0
999	456	60	250/500	1000	766.19	0	0	0.144144	11.0

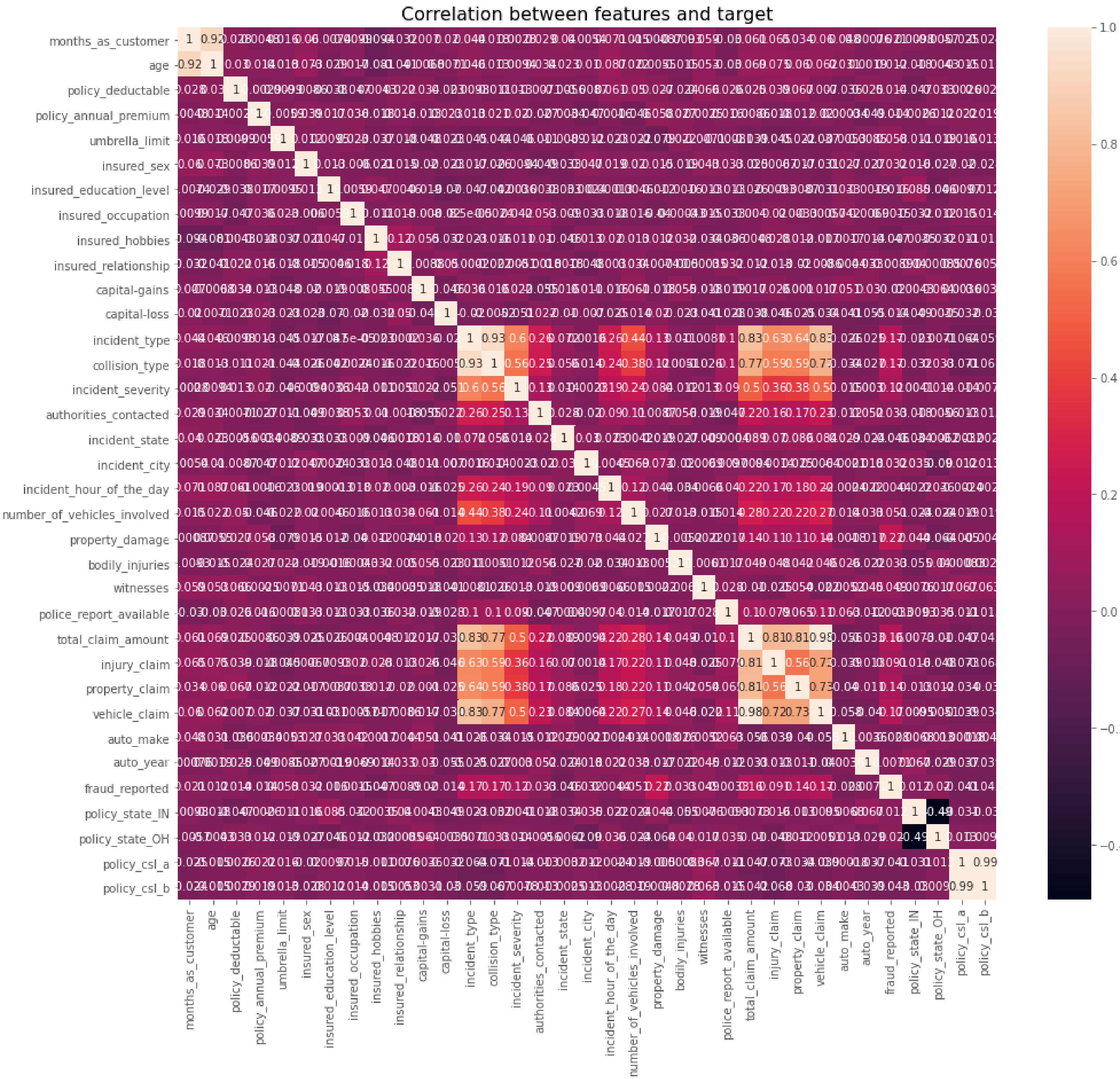
999 rows × 34 columns

As we see that after the cleaning and encoding of the data, we have a total of 999 observations and for each observation we have 34 columns including the target column. There is one feature 'policy_csl' which is continuous with the single limit values, but it is taken as object due to character present in the column. We will seperate the values in this feature using split function to create two features.

```
ds[['policy_csl_a','policy_csl_b']] = ds['policy_csl'].str.split("/",expand=True,)
ds['policy_csl_a'] = ds['policy_csl_a'].astype(np.int64)
ds['policy_csl_b'] = ds['policy_csl_b'].astype(np.int64)

ds.drop(['policy_csl'], axis = 1, inplace = True)
```

Correlation between features and target: Now, we look at the correlation between different features and the target variable. For this, we use the heatmap as a visualization tool.

```
plt.figure(figsize = (16,14))
plt.title("Correlation between features and target", fontsize = 16)
sns.heatmap(ds.corr(), annot =True)
plt.show()
```



From the above heatmap, we see that the feature 'vehicle_claim' has 0.98 correlation with 'total_claim_amount'. Also, 'policy_csl_a' has 0.99 correlation with 'policy_csl_b'. So, both of these features will be removed from the dataset to avoid colinearity.

```
ds.drop(['vehicle_claim', 'policy_csl_a'], axis = 1, inplace = True)
```

```
ds['Fraud_reported'] = ds['fraud_reported']
ds.drop(['fraud_reported'], axis = 1, inplace =True)
```

Scaling the data: For this problem project, we are provided with a data which consist of different types of features both categorical and numerical. After the encoding of the categorical columns, we are left with a data frame which has different scales of measurement for each feature. If we directly use this data to build the models, then the performance of the model will be affected and as a result, the prediction accuracy will dwindle. So, it is important to scale this data before the model building.

Since we have both the encoded categorical features and numerical features, it is important to scale to get the normal distribution of data. Hence, we will use the Standard Scaler from the sklearn library.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
ds.loc[:,'months_as_customer':'policy_csl_b'] = scaler.fit_transform(ds.loc[:,'months_as_customer':'policy_csl_b'])
ds
```

	months_as_customer	age	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occupation	insur
0	1.078558	0.990731	-0.223433	0.616768	-0.480353	1.078118	-0.006863	-1.157915	
1	0.209637	0.334259	1.411801	-0.242077	1.696927	1.078118	-0.006863	-0.162641	
2	-0.607149	-1.088095	1.411801	0.642285	1.696927	-0.927543	-1.378113	1.081451	
3	0.452935	0.224847	1.411801	0.652934	2.132383	-0.927543	-1.378113	-1.406733	
4	0.209637	0.553083	-0.223433	1.341721	2.132383	1.078118	-0.006863	1.081451	
...	...	...	...	...	...	...	...	...	
995	-1.745435	-0.103388	-0.223433	0.223122	-0.480353	-0.927543	-0.079034	-1.157915	
996	0.704922	0.224847	-0.223433	0.739150	-0.480353	-0.927543	-1.378113	0.583814	
997	-0.641905	-0.541036	-1.041050	0.520845	0.826015	-0.927543	-0.079034	-1.406733	
998	2.208155	2.522497	1.411801	0.412020	1.696927	1.078118	-0.006863	-0.411460	
999	2.190776	2.303673	-0.223433	-2.007482	-0.480353	-0.927543	-0.006863	1.081451	

999 rows × 33 columns

Data Imbalance: During the problem introduction, we learned that this is a type of binary classification machine learning problem. So, we have to predict if the values are yes/no or 0/1. In this case, if the data is imbalanced, then this could result in the ml model being overfitted and give prediction equal to the majority target class for all the observations or even it can result in the underfitting of the model. To avoid this problem, we first check the imbalance of the data.

```
x = ds.loc[:,'months_as_customer':'policy_csl_b']
y = ds.loc[:,'Fraud_reported']
```

```
plt.style.use('seaborn')
```

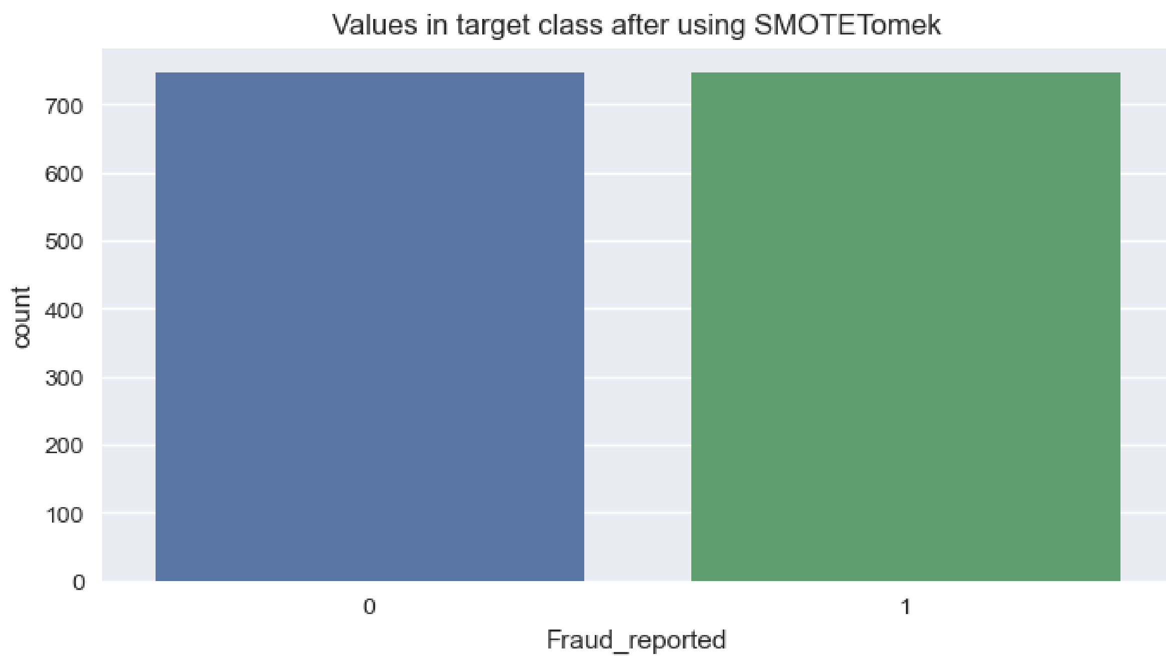
```
plt.figure(figsize = (8, 4))
plt.title("Values distribution in target class")
sns.countplot(data = ds, x = 'Fraud_reported')
plt.show()
```

From the above graph, it is clear that the data is heavily imbalanced with most values of target class present as No or 0. We will balance the imbalance using the SMOTETomek over-sampler, which will create the new synthetic values for target class 1 using Euclidian distance and Tomek method.

```
from imblearn.combine import SMOTETomek
smk = SMOTETomek()
x_new, y_new = smk.fit_resample(x, y)
print(x_new.shape, y_new.shape)
```

```
(1492, 32) (1492,)
```

```
plt.figure(figsize = (8, 4))
plt.title("Values in target class after using SMOTETomek")
sns.countplot(x = y_new)
plt.show()
```



After using the SMOTETomek over-sampler, new values have been created for the minority class and the data is now perfectly balanced. This data can be used for the building of the models.

CONCLUDING EDA

We started the Exploratory Data Analysis by identifying the type of problem that we are trying to solve and found that this project is a type of binary classification problem. After that, we performed various steps and operations like null values handling, feature engineering, data cleaning and feature encoding so that we have a clean useful data to build the model. We also used various types of data visualizations like boxplots, heatmaps and other plots which not only helped us to identify the null values and outliers in the data, but also gave various insights about features and the values present in them. For example, we found out that the greatest number of insurance claims were made by female and the authority which was contacted the most after the accident were the police.

A number of features were found to be not useful for the building of the prediction model like '_c39', 'policy_number', 'incident_location', 'insured_zip', 'policy_bind_date', 'incident_date' and 'auto_model'. The correlation heatmap also gave the information that the features 'vehicle_claim' and 'policy_csl_a' have a very high correlation value (>0.95) with the other features and can be removed from the data.

At the last of the EDA, we scaled the data using Standard Scaler to bring all the columns to a normal distribution and equivalent scales. Finally, the data imbalance was found and the minority class was oversampled using the SMOTETomek over-sampler.

Moving ahead, we will use the clean and scaled data to build the best possible model for the prediction.

MACHINE LEARNING MODEL BUILDING

We are finally ready to build the machine learning models that will help us to predict the frauds in the insurance claims. The final data frame is clean, scaled and balanced, which will be used to build the ml models. We will fit the data into different machine learning models and evaluate their performance and compare it with one another to see which model gives the best possible accuracy.

Best Random State: In order to build a machine learning model, we first need to split the data into two parts, the training phase and testing phase. The training data will be used to train, measure and evaluate the performance of models. For the splitting part, we will use a simple code to decide the best possible random state for data splitting.

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split

max_accuracy = 0
best_rs = 0
for i in range(1, 150):
    x_train, x_test, y_train, y_test = train_test_split(x_new, y_new, test_size = 0.25, random_state = i)
    lg = LogisticRegression()
    lg.fit(x_train, y_train)
    pred = lg.predict(x_test)
    acc = accuracy_score(y_test, pred)
    if acc > max_accuracy: # after each iteration, acc is replace by the best possible accuracy
        max_accuracy = acc
        best_rs = i
print(f"Best Random State is {best_rs}, {max_accuracy*100}")
```

```
Best Random State is 125, 72.11796246648794
```

In this case, we used the Logistic Regression model to find out the best random state. But any model can be used for this purpose. As the output code suggests, the best possible random state is 125 for this dataset. So, it is used to split the data.

```
x_train, x_test, y_train, y_test = train_test_split(x_new, y_new, test_size = 0.25, random_state = 125)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(1119, 32) (373, 32) (1119,) (373,)
```

training data has 1119 observations whereas testing data has 373 observations.

Model Selection: As this is a binary classification problem, we will use the classifier models for the prediction of the fraud. The models that I will use for prediction and then evaluate them are 'Logistic Regression', 'Decision Tree Classifier', 'K-Neighbors Classifier ', 'SVC', 'Random Forest Classifier' and 'ADA Boost Classifier'. Let's start with fitting the data into the models.

```
'''importing the models'''
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
```

```
# For Logistic Regression
lg = LogisticRegression()
lg.fit(x_train, y_train)
pred_lg = lg.predict(x_test)
print("Accuracy Score of Logistic Regression model is", accuracy_score(y_test, pred_lg)*100)

# For Decision Tree Classifier
dtc = DecisionTreeClassifier()
dtc.fit(x_train, y_train)
pred_dtc = dtc.predict(x_test)
print("Accuracy Score of Decision Tree Classifier model is", accuracy_score(y_test, pred_dtc)*100)

# For K-Nearest Neighbour Classifier
knc = KNeighborsClassifier(n_neighbors = 5)
knc.fit(x_train, y_train)
pred_knc = knc.predict(x_test)
print("Accuracy Score of K-Nearest Neighbour Classifier model is", accuracy_score(y_test, pred_knc)*100)

# For Support Vector Classifier
svc = SVC(kernel = 'rbf')
svc.fit(x_train, y_train)
pred_svc = svc.predict(x_test)
print("Accuracy Score of Support Vector Classifier model is", accuracy_score(y_test, pred_svc)*100)

# For Random Forest Classifier
rfc = RandomForestClassifier()
rfc.fit(x_train, y_train)
pred_rfc = rfc.predict(x_test)
print("Accuracy Score of Random Forest model is", accuracy_score(y_test, pred_rfc)*100)

# For ADA Boost Classifier
ada= AdaBoostClassifier()
ada.fit(x_train, y_train) # fitting the model
pred_ada = ada.predict(x_test) # predicting the values
print("Accuracy Score of ADA Boost model is", accuracy_score(y_test, pred_ada)*100)
```

```
Accuracy Score of Logistic Regression model is 72.11796246648794
Accuracy Score of Decision Tree Classifier model is 86.05898123324397
Accuracy Score of K-Nearest Neighbour Classifier model is 70.77747989276139
Accuracy Score of Support Vector Classifier model is 86.05898123324397
Accuracy Score of Random Forest model is 90.61662198391421
Accuracy Score of ADA Boost model is 86.05898123324397
```

Out of all the models, random forest classifier has the highest accuracy score of 90.61%. But this can be due to over fitting or under fitting of the data. In that case, the accuracy scores of the model can be wrong.

Cross Validation of Models: In order to avoid the overfitting or underfitting of the data in the models, we will cross validate all the models for 10 validations. We then compare the mean accuracy scores of the model with the accuracy scores will give us the actual results for the accuracy.

```
from sklearn.model_selection import cross_val_score

lg_scores = cross_val_score(lg, x_new, y_new, cv = 10) # cross validating the model
print(lg_scores) # accuracy scores of each cross validation cycle
print(f"Mean of accuracy scores is for Logistic Regression is {lg_scores.mean()*100}\n")

dtc_scores = cross_val_score(dtc, x_new, y_new, cv = 10)
print(dtc_scores)
print(f"Mean of accuracy scores is for Decision Tree Classifier is {dtc_scores.mean()*100}\n")

knc_scores = cross_val_score(knc, x_new, y_new, cv = 10)
print(knc_scores)
print(f"Mean of accuracy scores is for KNN Classifier is {knc_scores.mean()*100}\n")

svc_scores = cross_val_score(svc, x_new, y_new, cv = 10)
print(svc_scores)
print(f"Mean of accuracy scores is for SVC Classifier is {svc_scores.mean()*100}\n")

rfc_scores = cross_val_score(rfc, x_new, y_new, cv = 10)
print(rfc_scores)
print(f"Mean of accuracy scores is for Random Forest Classifier is {rfc_scores.mean()*100}\n")

ada_scores = cross_val_score(ada, x_new, y_new, cv = 10)
print(ada_scores)
print(f"Mean of accuracy scores is for ADA Boost Classifier is {ada_scores.mean()*100}\n")
```

```
[0.69333333 0.62       0.60402685 0.67114094 0.71812081 0.66442953
 0.67785235 0.68456376 0.74496644 0.65100671]
Mean of accuracy scores is for Logistic Regression is 67.29440715883669

[0.78666667 0.73333333 0.75838926 0.8590604  0.91275168 0.85234899
 0.83892617 0.87248322 0.9261745  0.89261745]
Mean of accuracy scores is for Decision Tree Classifier is 84.32751677852349

[0.78666667 0.73333333 0.75838926 0.8590604  0.91275168 0.85234899
 0.83892617 0.87248322 0.9261745  0.89261745]
Mean of accuracy scores is for Decision Tree Classifier is 84.32751677852349

[0.64666667 0.67333333 0.6442953  0.67114094 0.69127517 0.67785235
 0.71812081 0.69798658 0.72483221 0.61744966]
Mean of accuracy scores is for KNN Classifier is 67.62953020134228

[0.8        0.69333333 0.75167785 0.87248322 0.87248322 0.89261745
 0.86577181 0.9261745  0.90604027 0.84563758]
Mean of accuracy scores is for SVC Classifier is 84.26219239373603

[0.72666667 0.71333333 0.73154362 0.9261745  0.93288591 0.91946309
 0.94630872 0.96644295 0.97315436 0.95302013]
Mean of accuracy scores is for Random Forest Classifier is 87.88993288590603

[0.74666667 0.70666667 0.76510067 0.89932886 0.89932886 0.89932886
 0.91946309 0.95302013 0.93959732 0.93288591]
Mean of accuracy scores is for ADA Boost Classifier is 86.61387024608501
```

```python
# Checking for difference between accuracy and mean accuracies.
lis3 = ['Logistic Regression','Decision Tree Classifier','KNeighbors Classifier',
        'SVC', 'Random Forest Classifier', 'ADA Boost Classifier']

lis1 = [accuracy_score(y_test, pred_lg)*100, accuracy_score(y_test, pred_dtc)*100,
        accuracy_score(y_test, pred_knc)*100, accuracy_score(y_test, pred_svc)*100,
        accuracy_score(y_test, pred_rfc)*100, accuracy_score(y_test, pred_ada)*100]

lis2 = [lg_scores.mean()*100, dtc_scores.mean()*100, knc_scores.mean()*100,
        svc_scores.mean()*100, rfc_scores.mean()*100, ada_scores.mean()*100]

for i in range(0, 6):
    dif = (lis1[i]) - (lis2[i])
    print(lis3[i], dif)
```

```
Logistic Regression 4.823555307651247
Decision Tree Classifier 1.7314644547204807
KNeighbors Classifier 3.1479496914191145
SVC 1.7967888395079399
Random Forest Classifier 2.7266890980081797
ADA Boost Classifier -0.5548890128410449
```

After the cross validation, we see that the least difference between mean accuracies and total accuracy is given by Support Vector Machine Classifier, so we will use it the build the final model.

Hyperparameter Tuning: After the cross validation of the models, we have selected the SVC as the best possible model for this case study. Now tuning the parameters of this model to get the best possible results. To tune the parameters, we use GridSearchCV.

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

```python
svc = SVC()
parameters = { 'kernel' : ['rbf', 'linear'], 'gamma': [1, 0.1, 0.01, 0.001, 0.0001],
              'C': [0.1, 1, 10, 100, 1000]}
gs = GridSearchCV(estimator = svc, param_grid = parameters, scoring = 'f1', cv = 5)
gs.fit(x_train, y_train)
print(gs.best_score_)
print(gs.best_params_)
```

```
0.8927876610611161
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
```

```python
svc = SVC(kernel = 'rbf', C = 10, gamma = 0.1)
svc.fit(x_train, y_train)
print(svc.score(x_train, y_train))
pred_svc = svc.predict(x_test)
```

```
1.0
```

The best parameters given after the tuning are 'C': 10, 'gamma': 0.1, 'kernel': 'rbf'. We have used these parameters to build the model.

Model Evaluation: The support vector classifier is selected after the cross validation and the parameters of this model is then tuned. It is now time to evaluate the model. For the evaluation of SVC, we will use the Classification report to check the Precision, Recall and f1-score for the model. The confusion matrix will give the type 1 and type 2 errors. We will then plot the ROC curve and check the AUC score of the model.

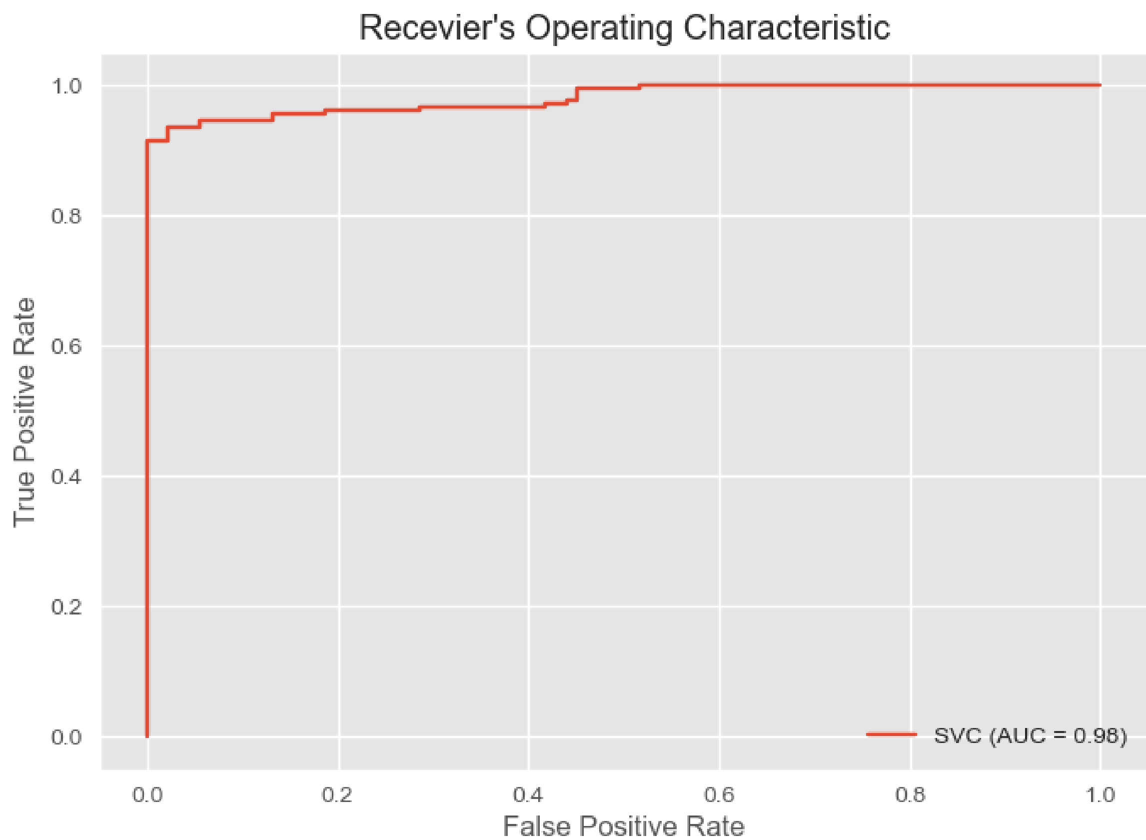
```
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import plot_roc_curve
```

```
plt.style.use('ggplot')
```

```
print("Accuracy Score of SVC model is", accuracy_score(y_test, pred_svc))
print("Confusion matrix for SVC Model is")
print(confusion_matrix(y_test, pred_svc))
print("Classification Report of the SVC Model is")
print(classification_report(y_test, pred_svc))

plot_roc_curve(svc, x_test, y_test) # arg. are model name, feature testing data, label testing data.
plt.title("Recevier's Operating Characteristic")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.show()
```

```
Accuracy Score of SVC model is 0.9463806970509383
Confusion matrix for SVC Model is
[[174   8]
 [ 12 179]]
Classification Report of the SVC Model is
              precision    recall  f1-score   support

           0       0.94      0.96      0.95       182
           1       0.96      0.94      0.95       191

    accuracy                           0.95       373
   macro avg       0.95      0.95      0.95       373
weighted avg       0.95      0.95      0.95       373
```

After the cross validation of Support Vector Classifier, the accuracy given by the model was 84.26%. But, the tuning of the parameters has increased the accuracy of the model to 94.63%. That is a very significant increase in the accuracy. Looking at the evaluation scores given by the classification report, the f1-score for the model is 0.95. Precision for target class 0 is 0.94 and for target class 1 is 0.96. Similarly, the Recall for target 0 class is 0.96 and for target class 1 is 0.94. Now looking at the ROC curve for the SVC model-



The Receiver’s Operating Characteristics Plot show that the AUC score of the Support Vector Classifier is 0.98. This is a very good score. The predictions made by the machine learning model are very accurate.

Serialization: We will save the SVC model as an object using the joblib library so that it can be used for the prediction of this frauds in Insurance Claims.

```
import joblib
joblib.dump(svc, 'Insurance_Claim_Fraud_Prediction.obj') # saving the model as an object
```

```
['Insurance_Claim_Fraud_Prediction.obj']
```

FINAL CONCLUSIONS

Our main aim to take this problem is to build effective prediction models that would help companies prevent scams and frauds in Insurance Claims by deceits. For the building of the fraud prediction model, we performed various operations and steps on the dataset provided. Just like a detective, we moved ahead step by step by solving a problem in each step.

During the EDA, we cleaned the provided data, handled the null values in the data frame, treated the features and then encoded all the categorical columns using proper encoding techniques. After the scaling, we found that the dataset was imbalanced, so we over-sampled the minority target class by creating new synthetic values.

Finally, with the clean processed data, we can build a highly accurate and good performing prediction model after the cross-validation and hyperparameter tuning. Using Support Vector Classifier, we achieved an accuracy of 95%, f1-score of 0.95 and AUC score of 0.98.

Using the built model, the companies will be able to predict the fraud insurance claims on similar datasets and save a ton of time and money. With the help of proper efforts and techniques, we have solved a real-life problem. Viola!!!

The complete solution notebook for this problem can be looked at by clicking on the following link.