Insurance Fraud Detection

Introduction

Problem Definition:

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. Machine Learning is a boon in helping the Auto Insurance industry with this problem.

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

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

Executive Summary:

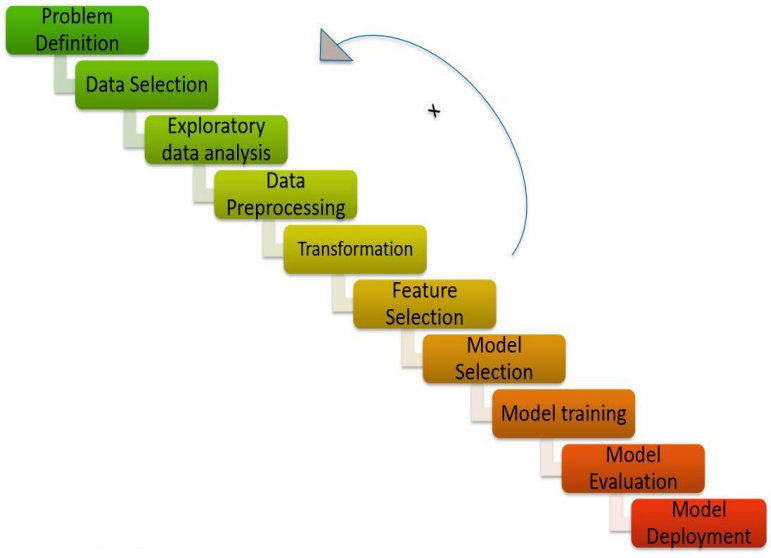
In this project, a dataset was provided with the details of the insurance policy along with the customer details, as well as details of the accident on the basis of which the claims have been made.

The Dataset was first cleaned, the various feature columns were analysed, then with feature engineering and based on strength of correlation and ANOVA f-score values, the feature columns were selected that would best predict the Target variable, to train and test machine learning models.

The auto insurance dataset was worked with to build a predictive model that best predicts if an insurance claim is fraudulent or not. Several models were trained and fitted with a part of the dataset and then tested with a different part of the dataset. The model that performed the best with the best confusion matrix performance, f1 score, ROC-AUC score and cross validation performance was then selected and tuned further with hyper parameter tuning techniques.

Data Analysis

In order to build a Machine Learning Model, we have a Machine Learning Life Cycle that every Machine Learning Project has to touch upon in the life of the model. Let's take a look at the model life cycle and then we will look into the actual machine learning model and understand it better along with the lifecycle as we move forward.



Now that we understand the lifecycle of a Machine Learning Model, lets import the necessary libraries and proceed further.

Importing the necessary Libraries:

To analyze the dataset or even to import the dataset, we have now imported all the important libraries that we will be needing during analysis.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import os
import pickle
import statsmodels.api as sm
import warnings
warnings.filterwarnings('ignore')
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, mean_absolute_error,mean_squared_error,r2_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression, Ridge, Lasso
from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, RandomForestRegressor, AdaBoostRegressor
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.preprocessing import power_transform, LabelEncoder, PowerTransformer, StandardScaler
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.model_selection import cross_val_score
from scipy import stats
from scipy.stats import zscore
import statsmodels.api as sm
```

Importing the Dataset

Let's import the dataset first.

```
df=pd.read_csv('https://raw.githubusercontent.com/dsrscientist/Data-Science-ML-Capstone-Projects/master/Automobile_insurance_fraud.csv
df
```

Copied the raw data and saved it as a csv file on my local computer after which I imported the entire dataset on this Jupyter Notebook with the help of pandas.

About the Dataset:

I have imported the dataset which was in "csv" format as "df". Below is how the dataset looks.

df

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_z
0	328	48	521585	17-10-2014	OH	250/500	1000	1406.91	0	4661
1	228	42	342868	27-06-2006	IN	250/500	2000	1197.22	5000000	4681
2	134	29	687698	06-09-2000	OH	100/300	2000	1413.14	5000000	4306
3	256	41	227811	25-05-1990	IL	250/500	2000	1415.74	6000000	6081
4	228	44	367455	06-06-2014	IL	500/1000	1000	1583.91	6000000	6107
...	...	...	...	...	...	...	...	...	...	
995	3	38	941851	16-07-1991	OH	500/1000	1000	1310.80	0	4312
996	285	41	186934	05-01-2014	IL	100/300	1000	1436.79	0	6081
997	130	34	918516	17-02-2003	OH	250/500	500	1383.49	3000000	4427
998	458	62	533940	18-11-2011	IL	500/1000	2000	1356.92	5000000	4417
999	456	60	556080	11-11-1996	OH	250/500	1000	766.19	0	6122

1000 rows × 40 columns

The given dataset consists of 1000 rows and 40 columns.

This data set contains of Independent and Dependent (target) variables.

Independent variable: They are also known as Input variables. These are the input for a process that is being analyzed.

Dependent variable: They are also known as Output or Target variables. They are dependent on Independent variables for their outcome.

The Independent Feature columns are:

- **months_as_customer:** Number of months for which the person has been a customer
- **age:** Age of Customer
- **policy_number**: Identification number of policy
- **policy_bind_date**: Time period between effective date of coverage and policy issuance.

- **policy_state**: State where policy is active
- **policy_csl:** Policy Combined single limit
- **policy_deductable:** Amount paid before the insurance company starts paying up.
- **Policy_annual_premium**: The total amount of premium paid annually
- **Umbrella_limit:** Provides excess limits and gives additional excess coverage
- **Insured_zip:** Zip Code of the Insured address
- **insured_sex :** Gender
- **Insured_education_level:** Education Background of Insured
- **Insured_occupation:** Occupation of Insured
- **Insured_hobbies:** Hobbies of the Insured
- **Insured_relationship:** Relationship of the Insured
- **Capital-gains:** Capital Gains made from insurance
- **Capital-loss:** Capital Loss incurred
- **Incident_date:** Date on which Incident Occured
- **incident_type:** Type of Incident
- **Collision_type:** Type of collision
- **incident_severity:** Severity of Incident
- **Authorities_contacted:** Whether authorities were contacted
- **Incident_state:** State where incident occurred
- **incident_city:** City where incident occurred
- **incident_location:** Location of incident
- **Incident_hour_of_the_day:** Time of the day when incident occurred
- **number_of_vehicles_involved:** Number of vehicles involved in incident.
- **property_damage:** Whether there was property damage or not
- **Bodily_injuries:** Severity of bodily injuries
- **witnesses:** Number of Witnesses
- **Police_report_available:** Whether police reports are available
- **Total_claim_amount:** Total amount of claim
- **Injury_claim:** Injury Claim amount

- **Property_claim:** Property Claim amount
- **vehicle_claim:** Vehicle Claim amount
- **Auto_make:** Make of Vehicle
- **Auto_model:** Model of Vehicle
- **Auto_year:** Year of Vehicle Manufacture

The Target Variable to predict is given in the column:

Fraud_reported: Whether fraud was reported as Yes or No

Data Analysis (Exploratory Data Analysis)

We cannot have null values in the data as this will affect the data and eventually the predicted result. Therefore, we must check for any null values in the dataset.

Data Cleaning:

Upon inspecting all the columns in the dataframe, it is observed that column _c39 has no usable data present as they were all NaN values. Other columns appear to have no NaN.

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

months_as_customer            0
age                           0
policy_number                 0
policy_bind_date              0
policy_state                  0
policy_csl                    0
policy_deductable             0
policy_annual_premium         0
umbrella_limit                0
insured_zip                   0
insured_sex                   0
insured_education_level       0
insured_occupation            0
insured_hobbies               0
insured_relationship          0
capital-gains                 0
capital-loss                  0
incident_date                 0
incident_type                 0
collision_type                0
incident_severity             0
authorities_contacted         0
incident_state                0
incident_city                 0
incident_location             0
incident_hour_of_the_day      0
number_of_vehicles_involved   0
property_damage               0
bodily_injuries               0
witnesses                     0
police_report_available       0
total_claim_amount            0
injury_claim                  0
property_claim                0
vehicle_claim                 0
auto_make                     0
auto_model                    0
auto_year                     0
fraud_reported                0
_c39                       1000
dtype: int64
```

Checking for blank spaces, random characters in each column

```
# printing all data type and their unique values
for column in df.columns:
    if df[column].dtype==object:
        print(df[column].value_counts())
        print('==========================================================')
```

```
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
==========================================================
OH    352
IL    338
IN    310
Name: policy_state, dtype: int64
==========================================================
250/500    351
```

There are 178, 360 and 343 null values in collision_type, property_damage and police_report_available respectively.

- For collision_type we can replace the null values with the mode of that column. The mode of the column for collision_type is no.
- For property_damage we can replace the null values with 'NO' because it could be that they weren't any property damage in the first place. Also, the mode of the column is 'NO'
- Similarly, for police_report_available we can replace the null values with 'NO' because it could be that they haven't reported. Similarly here also the mode of the column is 'NO'

```
df.isin(['?']).sum() #Checking for ? values in dataset
```

```
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               178
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              360
bodily_injuries                0
witnesses                      0
police_report_available      343
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
```

Therefore, Converting ' ?' to NaN values, which will later be imputed with values using various imputation techniques.

```
# Replacing '?' by the most common collision type using mode
df['collision_type']=df['collision_type'].replace('?', 'Rear Collision')

# If there is no values in property_damage then it means no property damage.
df['property_damage']=df['property_damage'].replace('?', 'NO')

# If there is nothing in police_report_available, that means there is no FIR and nothing.
df['police_report_available']=df['police_report_available'].replace('?', 'NO')
```

_c39 has no usable data present. There is only one unique value, i.e: 'nan' which is null values. Therefore, we can eliminate this column.

```
#drop _c39 columns
df.drop('_c39',axis=1,inplace=True)
```

It is observed that no more null values remain in the dataset.

```
df.isin(['?']).sum() #Checking for ? values in dataset
```

```
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
```

Exploratory Data Analysis

```
# Lets do statical analysis, Get unique and top values for the dataset
df.describe()
```

	months_as_customer	age	policy_number	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	capital-gains	capital-l
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03	1000.000000	1000.000000	1000.000
mean	203.954000	38.948000	546238.648000	1136.000000	1256.406150	1.101000e+06	501214.488000	25126.100000	-26793.700
std	115.113174	9.140287	257063.005276	611.864673	244.167395	2.297407e+06	71701.610941	27872.187708	28104.096
min	0.000000	19.000000	100804.000000	500.000000	433.330000	-1.000000e+06	430104.000000	0.000000	-111100.000
25%	115.750000	32.000000	335980.250000	500.000000	1089.607500	0.000000e+00	448404.500000	0.000000	-51500.000
50%	199.500000	38.000000	533135.000000	1000.000000	1257.200000	0.000000e+00	466445.500000	0.000000	-23250.000
75%	276.250000	44.000000	759099.750000	2000.000000	1415.695000	0.000000e+00	603251.000000	51025.000000	0.000
max	479.000000	64.000000	999435.000000	2000.000000	2047.590000	1.000000e+07	620962.000000	100500.000000	0.000

Difference in mean and 50% and considerable difference in 75% and max of columns months_as_customer,policy_annual_premium,capital-gains,total_claim_amount,injury_claim and property_claim suggests skewness in respective data distributions and presence of outliers.

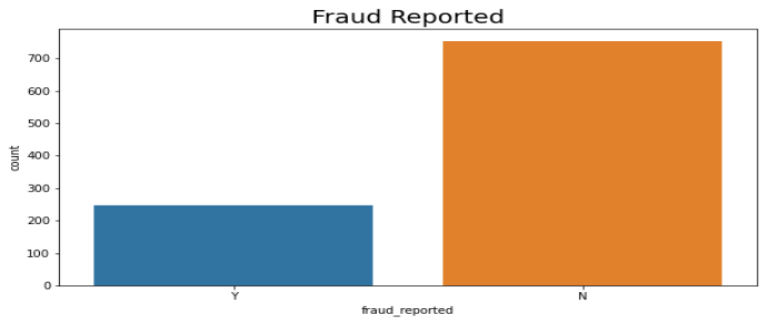
This is a Classification Problem since the Target variable / Label column ("fraud_reported") has Categorical type of Data.

Univariate Analysis

Analyzing the Target Class

There are 2 unique categorical values in the Label column / target variable, viz. 'Y' and 'N'.

```
plt.figure(figsize = (10,5)) #Plotting with size of 10 * 5
sns.countplot(x="fraud_reported", data=df) #Plotting Countplot for fraud_reported
plt.title("Fraud Reported", fontsize = 20) #Setting up the title
plt.show() #plotting the graph
```



Class

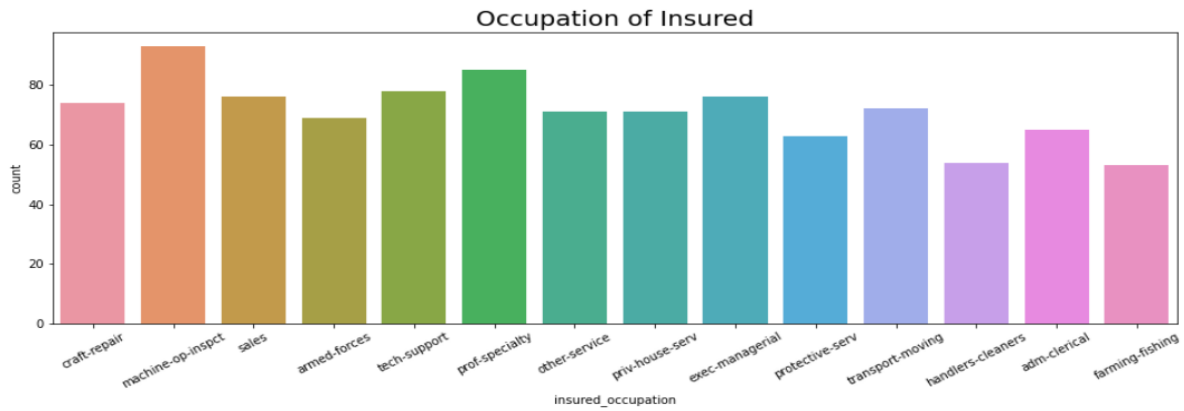
'N' : Has 753 of total values

'Y' : Has 247 of total values

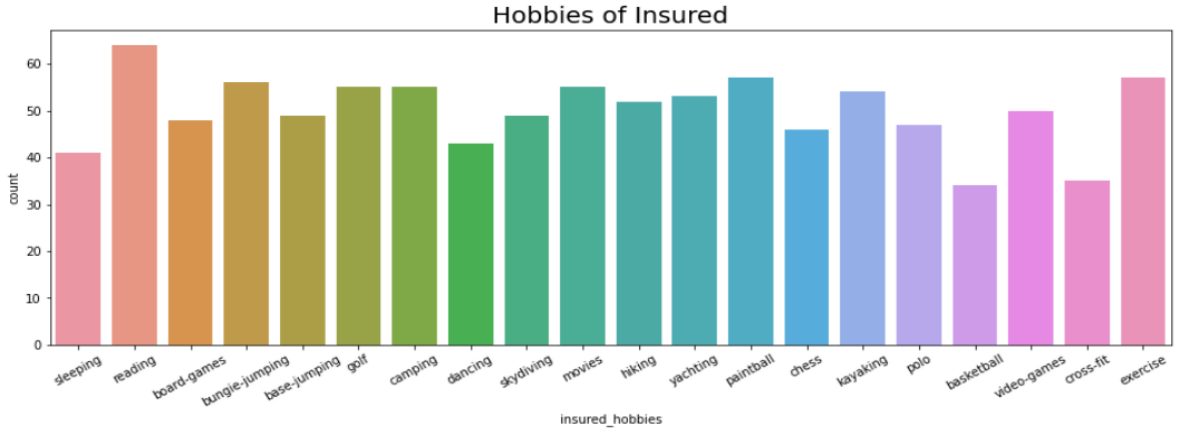
From the plot we can observe that the count of "N" is high compared to "Y". We can assume that "Y" stands for "Yes" that is the insurance is fraudulent and "N" stands for "No" means the insurance claim is not fraudulent. Here most of the insurance claims have not reported as fraudulent. Since it is our target column, it indicates the class imbalance issue. We will balance the data using oversampling method in later part.

Analyzing the rest of the Feature Columns

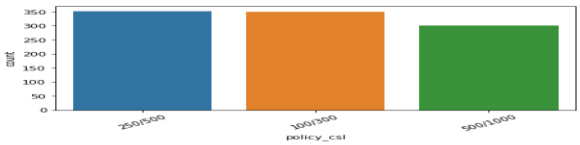
```
plt.figure(figsize = (16,5)) #Plotting with size of 10 * 5
ax=sns.countplot(x="insured_occupation", data=df) #Plotting Countplot for insured_occupation
ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
plt.title("Occupation of Insured", fontsize = 20) #Setting up the title
plt.show() #plotting the graph
```

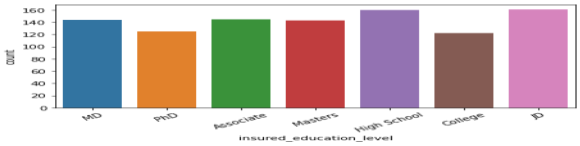


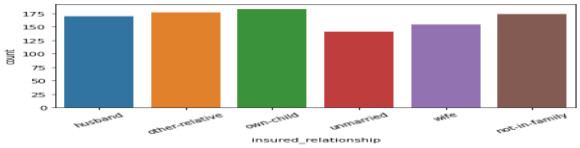

```
plt.figure(figsize = (16,5)) #Plotting with size of 10 * 5
ax=sns.countplot(x="insured_hobbies", data=df) #Plotting Countplot for insured_sex
ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
plt.title("Hobbies of Insured", fontsize = 20) #Setting up the title
plt.show() #plotting the graph
```

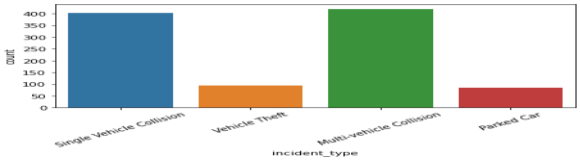


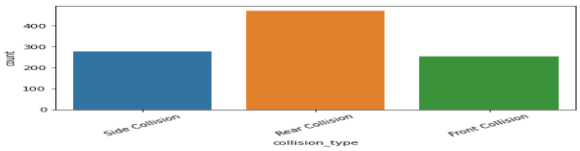
```
label_list = ['policy_state','policy_csl','insured_sex','insured_education_level','insured_relationship','incident_type','collision_type','incident_severit
fig,axes = plt.subplots(7,2,figsize=(16,45))
for i,cat in enumerate(label_list):
    row,col = i//2,i%2 #getting size of plots in row and cols
    ax=sns.countplot(x=cat,data=df,ax=axes[row,col]) #Plotting count plot with hue Loan Status
    ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
plt.subplots_adjust(hspace=1) # Plotting the graphs
```

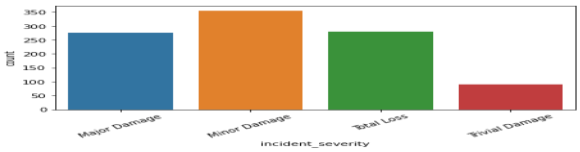


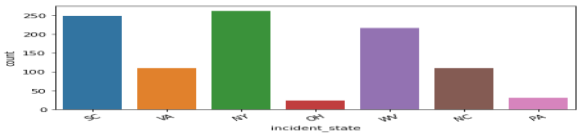


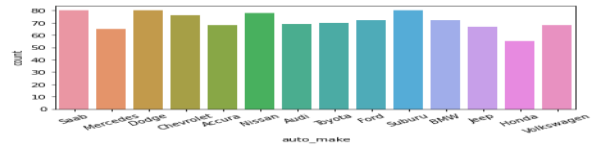












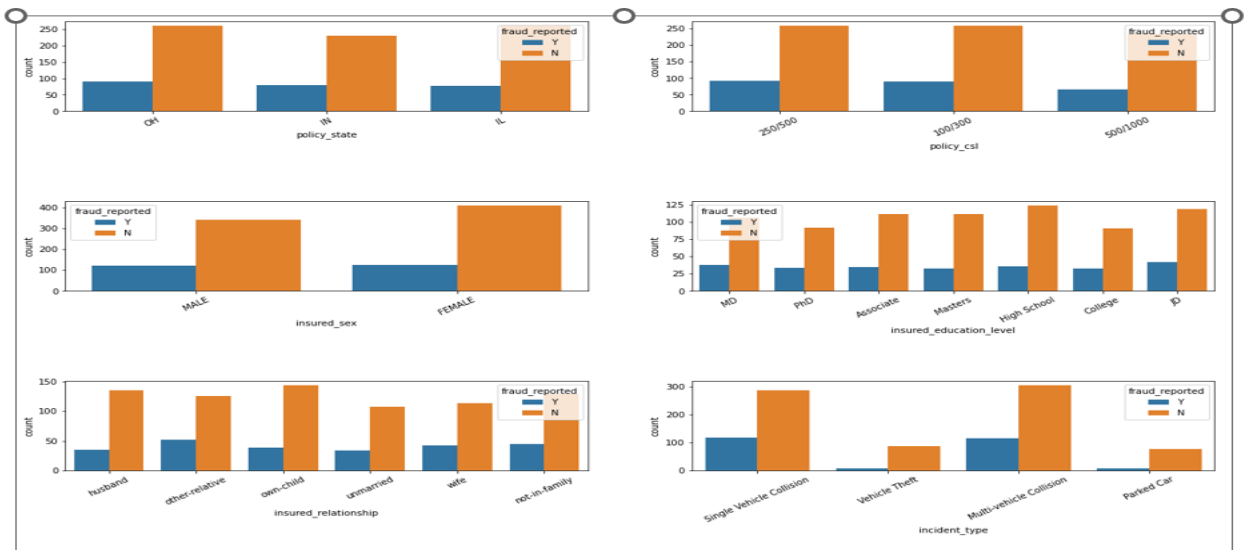
Upon analyzing the rest of the Feature Columns, following observations are made:

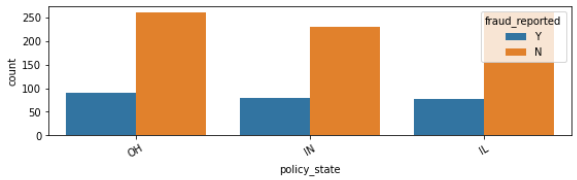
- Policy_state is high in OH and IL where as low in IN.
- Policy_csl is 250/500 and 100/300 is high whereas 500/1000 is less.
- Female applied for Insurance more than Male.
- Mostly insured person have education level High School and JD.
- The people who have child, those peoples did insurances more.
- Single Vehicle Collision and Multi vehicle Collision had more incident whereas Vehicle Theft and Parked car incidents are less.
- Rear Collision is more than 400 whereas Front collision is less.
- Minor Damages are happen more whereas Trival Damages are less.
- People contacted more and in very less conditions people didn't call to anyone.
- More incident happened in NY where as in OH very less incident happen.
- Almost all cities have same number of incidents.
- Property damages are less.
- Police reports are available only in 30% cases.
- All cars are insured equally near 60-80.

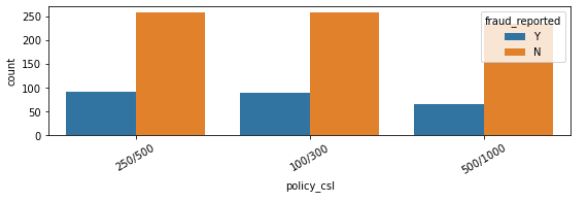
Bivariate Analysis

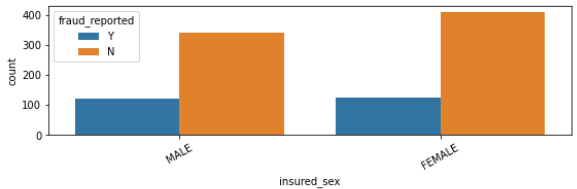
Interpreting Relationship between Dependent Variable and Independent Variables

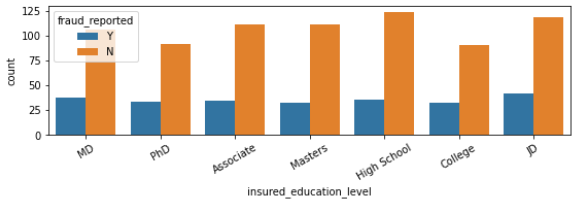
```
fig,axes = plt.subplots(7,2,figsize=(20,30))
for i,cat in enumerate(label_list):
    row,col = i//2,i%2 #getting size of plots in row and cols
    ax=sns.countplot(x=cat,data=df,hue='fraud_reported',ax=axes[row,col]) #Plotting count plot with hue Loan Status
    ax.set_xticklabels(ax.get_xticklabels(), rotation=30)
plt.subplots_adjust(hspace=1) # Plotting the graphs
```

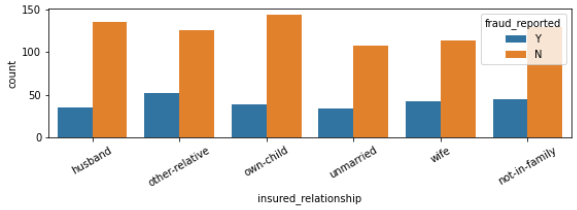


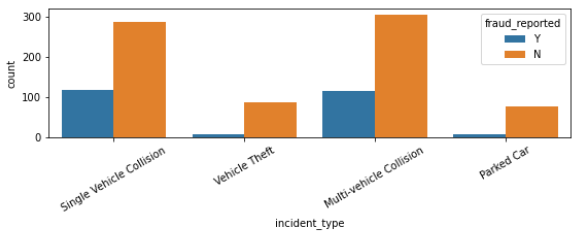

fraud_reported
Y
N
count
OH
IN
IL
policy_state

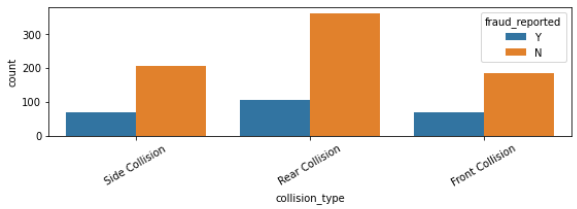

fraud_reported
Y
N
count
250/500
100/300
500/1000
policy_csl

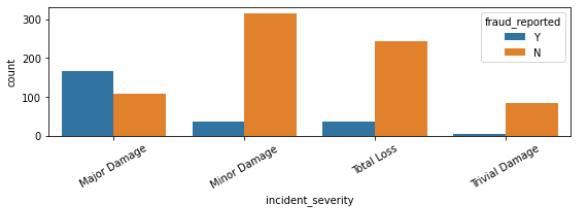

fraud_reported
Y
N
count
MALE
FEMALE
insured_sex

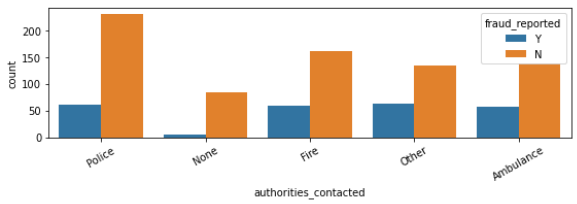

fraud_reported
Y
N
count
MD
PhD
Associate
Masters
High School
College
JD
insured_education_level

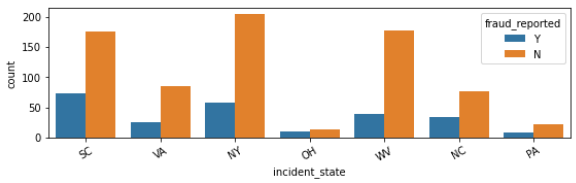

fraud_reported
Y
N
count
husband
other-relative
own-child
unmarried
wife
not-in-family
insured_relationship

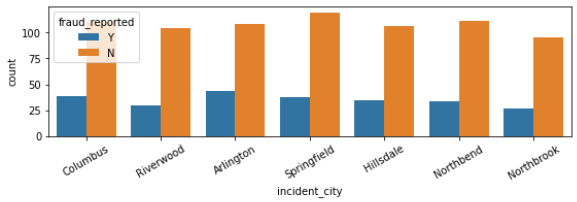

fraud_reported
Y
N
count
Single Vehicle Collision
Vehicle Theft
Multi-vehicle Collision
Parked Car
incident_type

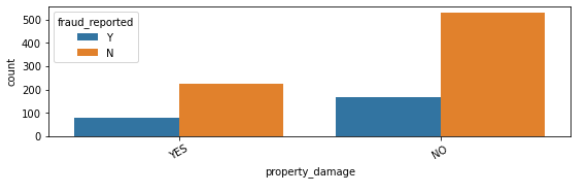

fraud_reported
Y
N
count
Side Collision
Rear Collision
Front Collision
collision_type

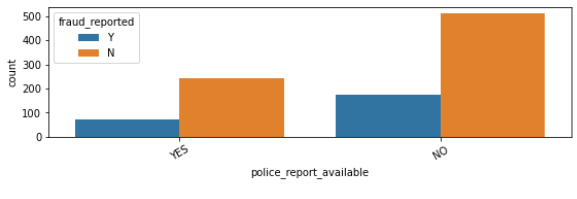

fraud_reported
Y
N
count
Major Damage
Minor Damage
Total Loss
Trivial Damage
incident_severity

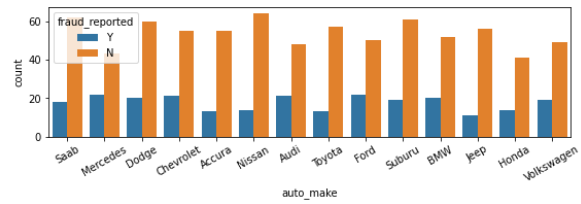

fraud_reported
Y
N
count
Police
None
Fire
Other
Ambulance
authorities_contacted

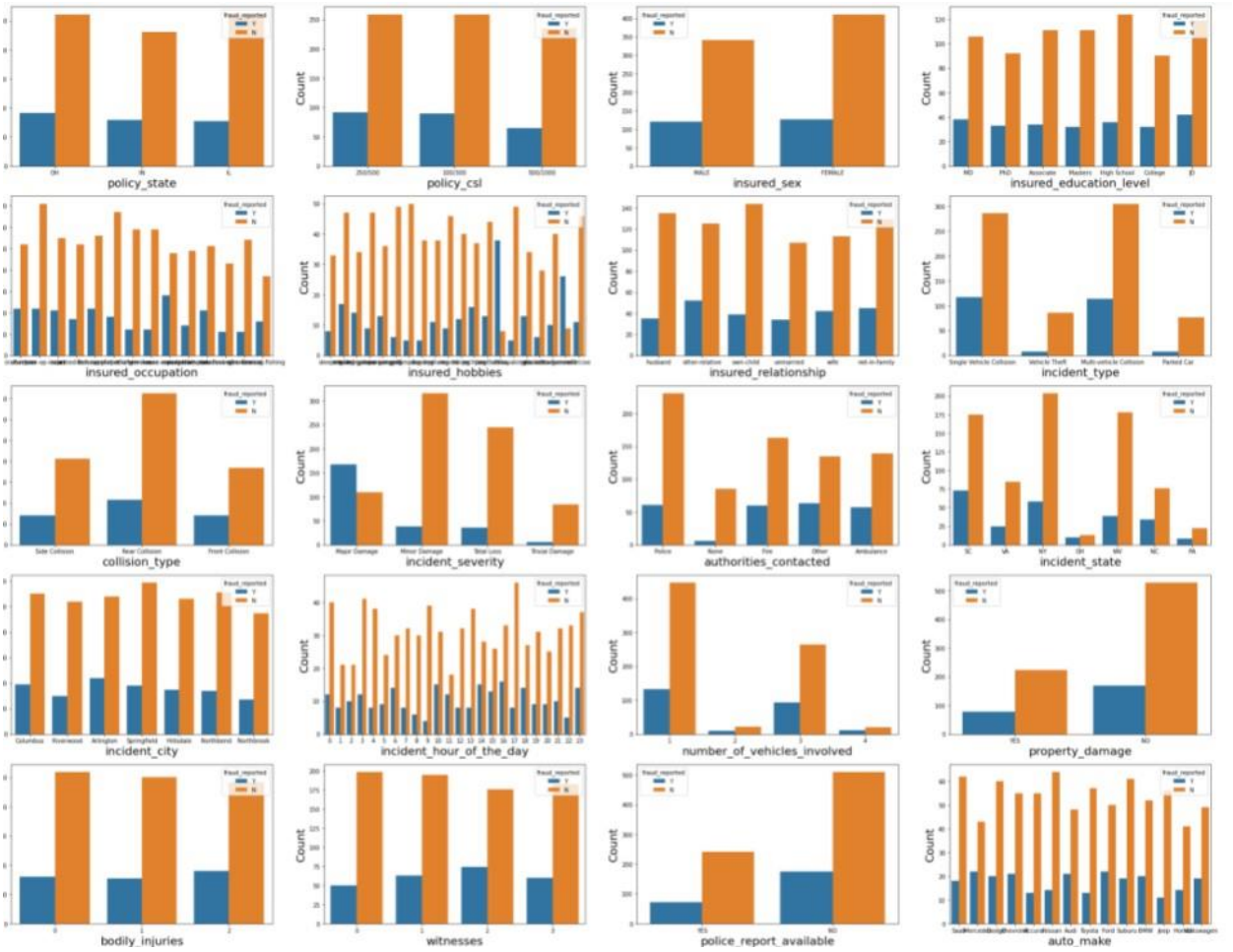

fraud_reported
Y
N
count
SC
VA
NY
OH
WV
NC
PA
incident_state


fraud_reported
Y
N
count
Columbus
Riverwood
Arlington
Springfield
Hillsdale
Northbend
Northbrook
incident_city

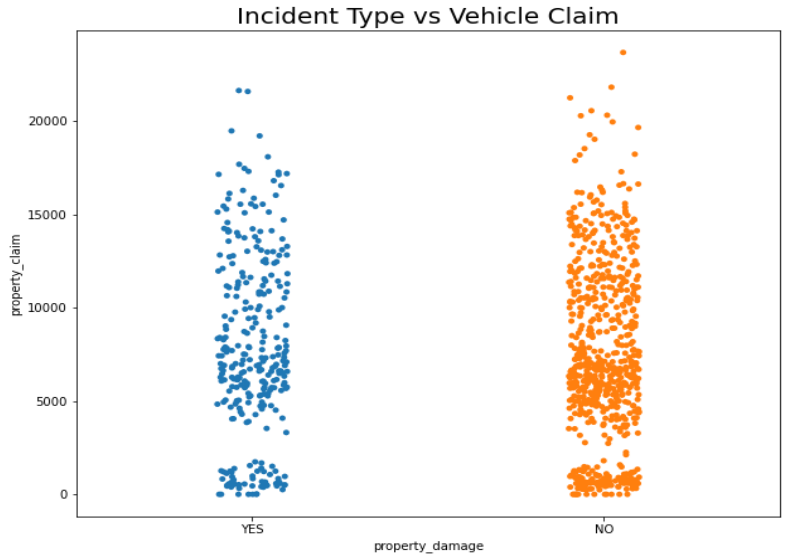

fraud_reported
Y
N
count
YES
NO
property_damage


fraud_reported
Y
N
count
YES
NO
police_report_available

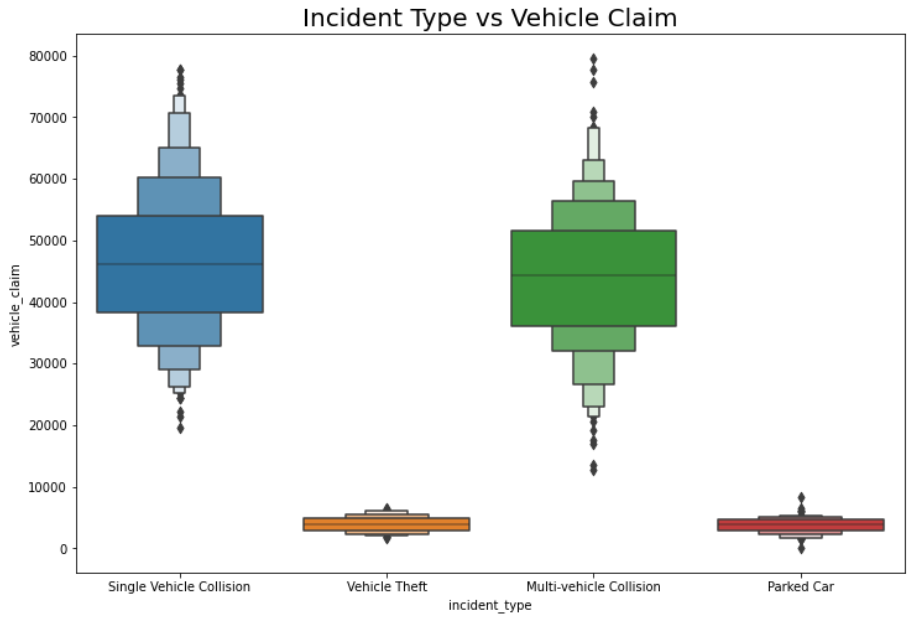

fraud_reported
Y
N
count
Saab
Mercedes
Dodge
Chevrolet
Accura
Nissan
Audi
Toyota
Ford
Suburu
BMW
Jeep
Honda
Volkswagen
auto_make



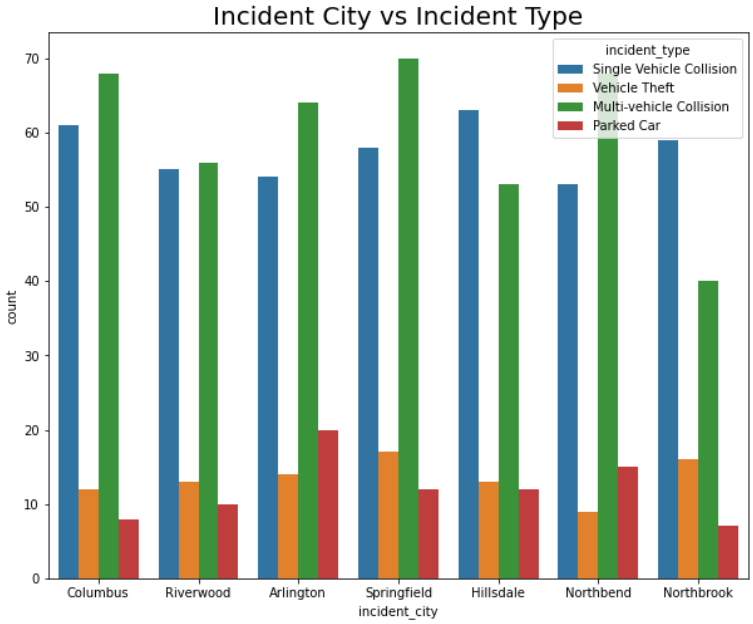

```
plt.figure(figsize = (10,8))
sns.stripplot(df['property_damage'], df['property_claim'])
plt.title('Incident Type vs Vehicle Claim', fontsize = 20)
plt.show()
```



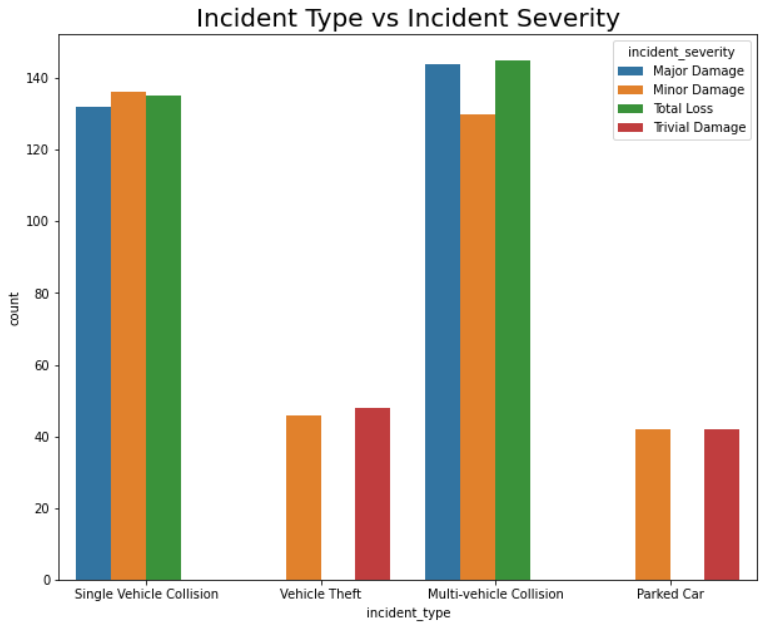
```
plt.figure(figsize = (12,8))
sns.boxenplot(df['incident_type'], df['vehicle_claim'])
plt.title('Incident Type vs Vehicle Claim', fontsize = 20)
plt.show()
```




```
plt.subplots(figsize=(10, 8))
sns.countplot(x='incident_city',hue='incident_type',data=df)
plt.title('Incident City vs Incident Type', fontsize = 20)
plt.show()
```



```
plt.subplots(figsize=(10, 8))
sns.countplot(x='incident_type',hue='incident_severity',data=df)
plt.title('Incident Type vs Incident Severity', fontsize = 20)
plt.show()
```

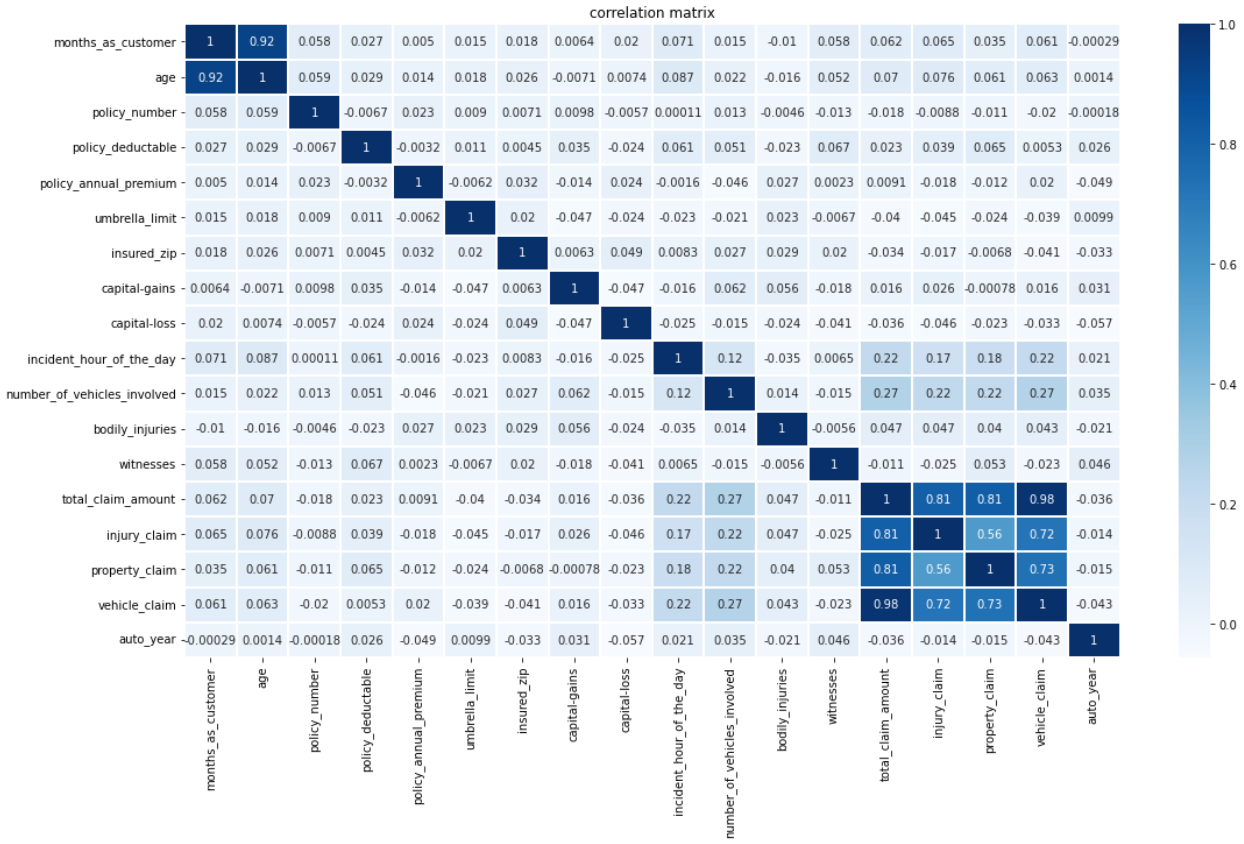


Following observations can be made from above graphs:

- In OH and IL Policy state Fraud didn't happen much when compared with IN. In IN Fraud happen more than OH and IL.
- In Policy_csl 250/500 and 100/300 Fraud reported more than 500/1000.
- More number of Fraud case reported against Female.
- Person who has education level JD did more frauds than others.
- People who have other relatives has reported highest fraud cases.
- In case of Single Vehicle Collision and Multi vehicle Collision maximum frauds happened.
- When Collision type is Rear, Fraud reported more.
- At the time of Major Damages, Fraud is reported too high.
- When People didn't call to anyone that time Fraud happen very less. If person contacted to anyone, Fraud happened.
- In SC state Fraud Reported more than other states.
- Almost all cities have same number of Fraud Reported.
- When Property is not damages that time Fraud happed more.
- When there is no Police report available that time Fraud happen more.
- Mercedes and Ford have a greater number of Fraud Reported.
- We can clearly see that In Single Vehicle collision and Multi-vehicle collision, Major Damage, Minor Damage and Total loss happened, whereas when Vehicle is Theft or car was parked that time Minor damage and Trivial Damage happen.

Multi-Variate Analysis

```
fig=plt.figure(figsize=(18,10))
hc=df.corr(method='pearson')
sns.heatmap(hc,annot=True,cmap="Blues",linewidths=.1, annot_kws={"size": 10})
plt.title("correlation matrix")
plt.show()
```



In the above plot we can see the relation between each variable with respect to other variables.

Now we have done with the visualization in order to analyze and understand the data. So, in this EDA part, we have looked into various aspect of the dataset, like looking for the null values and imputing, extracting date time, observing the value counts and doing the feature extraction etc.

Now we will be performing Data Pre-processing by identifying the outliers and removing them. Along with it we will also look for the skewness of the dataset and remove the skewness.

Data Visualization and EDA Concluding Remarks:

In the given data, 'fraud_reported' feature is the Target feature or variable. The unique values of this feature are only 2 i.e Y and N (Yes and No), which means it has only two classes. So, as there are only two unique values this is a 'Classification Problem.'

The dependent or target variable has 753 nonfraudulent cases and 247 fraudulent cases which can be seen below in the form of value counts and bar chart.

Data Pre-Processing

The data set has variables in both object type and numerical type (int and float)

Therefore, we have to pre-process the data to move forward.

All the float type or int type variables should be converted into the same scale since the range of values of raw data varies widely, in some machine learning algorithms, objective functions do not work correctly without normalization.

Therefore, normalization is to be performed only on the numerical type (int and float type) variables.

Feature Engineering

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

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

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

```
#Converting Auto Year into Vehicle Age by subtracting 2020
df['vehicle_age'] = 2020 - df['auto_year']
```

```
df=df.drop(['policy_csl', 'auto_year', 'policy_number', 'insured_zip', 'policy_bind_date', 'incident_date','incident_location','incident_hour_of_the_da
```

```
df.head()
```

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occupation
0	328	48	OH	1000	1406.91	0	MALE	MD	craft-repai
1	228	42	IN	2000	1197.22	5000000	MALE	MD	machine-op-inspct
2	134	29	OH	2000	1413.14	5000000	FEMALE	PhD	sales
3	256	41	IL	2000	1415.74	6000000	FEMALE	PhD	armed-forces
4	228	44	IL	1000	1583.91	6000000	MALE	Associate	sales

5 rows × 34 columns

```
#Getting all columns i.e. (int64) values in int_label
int_label=list(df.select_dtypes(['int64']).columns)
```

Encoding the categorical columns using Label Encoding

We can see that there are object-type variables too. These variables contain string data that cannot be passed into the machine learning model as it won't be able to recognize string data type. It only recognizes numerical data.

Therefore, we need to convert the string data into numerical data. This can be done by manually encoding or by using an encoder such Label Encoder, one-hot encoder etc. For example: the target variable fraud_reported consists of only two unique values, Y & N. after encoding this will get converted to 0 and 1. Similarly, if there are three unique values then it will be converted to 0,1, and 2.

```
label_list=list(df.select_dtypes(['object']).columns) #Getting the list of object columns in label_list
le=LabelEncoder() #initilizing Label Encoder
for i in label_list:
    df[i] = le.fit_transform(df[i]) #Converting Object coulmns to number's using label encoder
```

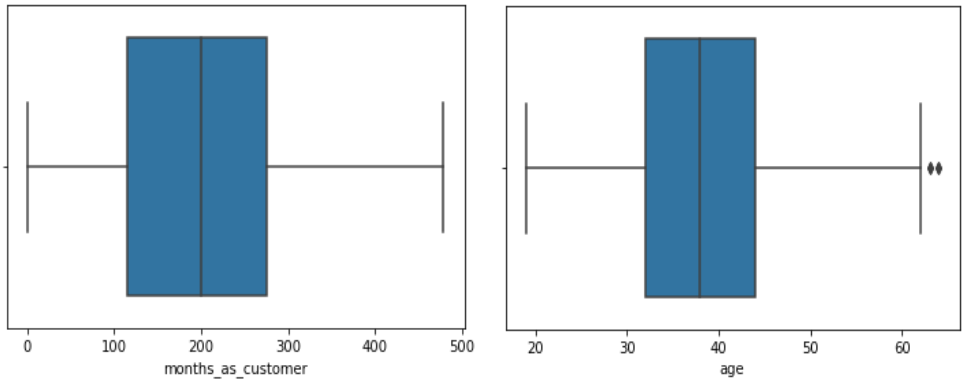
```
df.head()
```

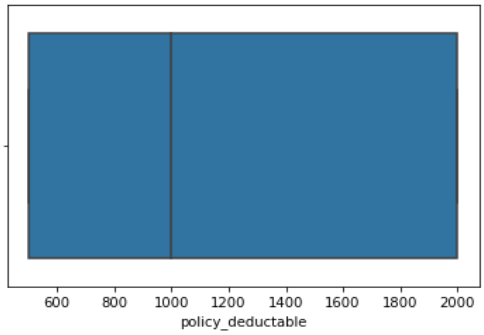
	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occupation
0	328	48	2	1000	1406.91	0	1	4	2
1	228	42	1	2000	1197.22	5000000	1	4	6
2	134	29	2	2000	1413.14	5000000	0	6	11
3	256	41	0	2000	1415.74	6000000	0	6	1
4	228	44	0	1000	1583.91	6000000	1	0	11

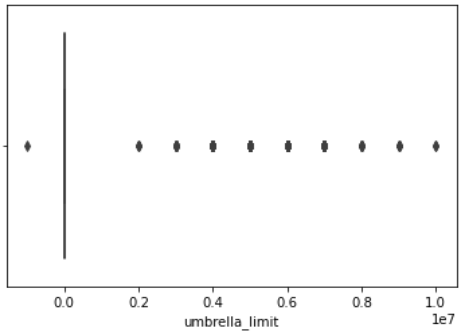
5 rows × 34 columns

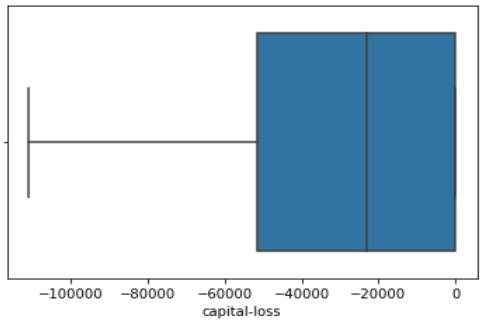
Now we have encoded the dataset using label encoder and the dataset looks like above.

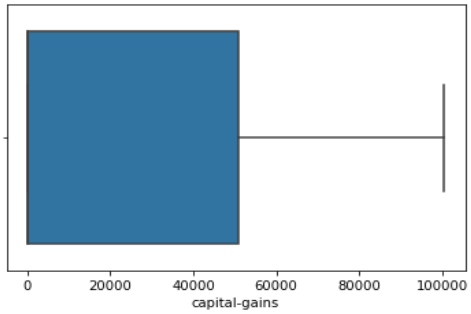
Checking for Outliers in columns with continuous distribution

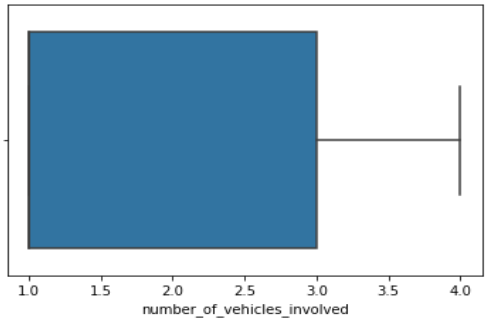


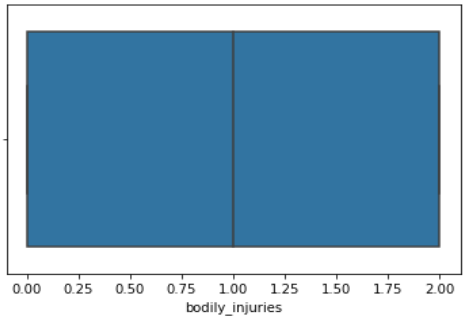

600 800 1000 1200 1400 1600 1800 2000
policy_deductable

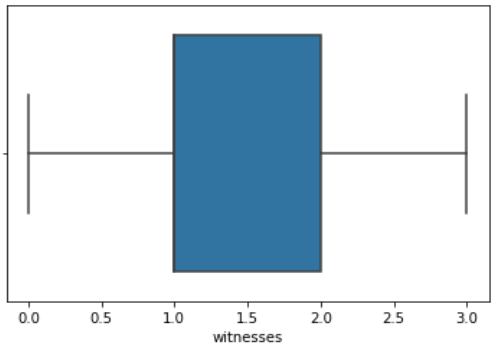

0.0 0.2 0.4 0.6 0.8 1.0
1e7
umbrella_limit

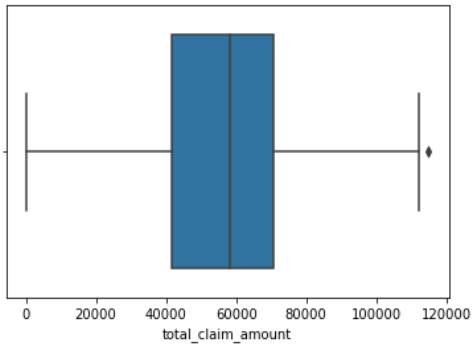

−100000 −80000 −60000 −40000 −20000 0
capital-loss

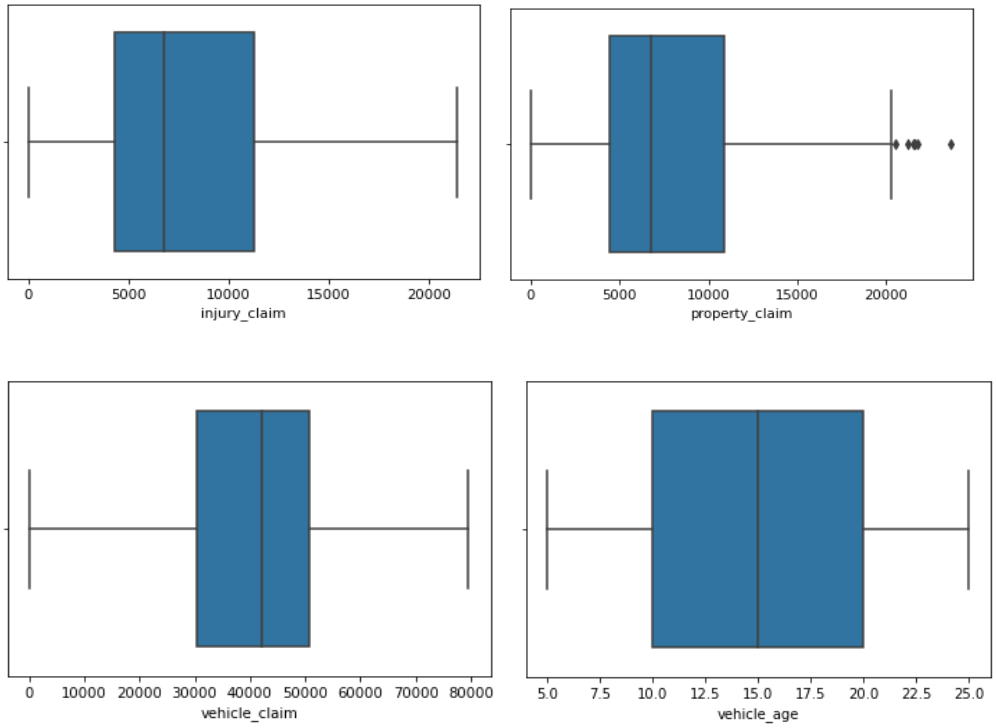

0 20000 40000 60000 80000 100000
capital-gains


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


0.0 0.5 1.0 1.5 2.0 2.5 3.0
witnesses


0 20000 40000 60000 80000 100000 120000
total_claim_amount



Removing the outliers using Zscore method.

```
z=np.abs(zscore(df))
print(np.where(z>3))
```

```
(array([ 31,  48,  88, 115, 119, 229, 248, 262, 314, 430, 458, 500, 503,
       657, 700, 763, 807, 875, 922, 975], dtype=int64), array([ 5,  5,  5,  5,  5,  4,  4,  5,  5,  5,  5, 26,  5,  5,  5,  4, 12,
        5,  5,  5], dtype=int64))
```

```
z=np.abs(zscore(df))
threshold=3
new_df=df[(z<3).all(axis=1)]
print(df.shape)
print(new_df.shape)
```

```
(1000, 34)
(980, 34)
```

```
#checking for how much data we lost
loss_percent=(1000-980)/1000*100
print(loss_percent)
```

```
2.0
```

After removing the outliers, we are checking the data loss percentage by comparing the rows in our original data set and the new data set and 2% data loss is in the acceptable range.

Checking for Skewness in Data

```
#Checking skewness of all columns
new_df.skew()
```

```
months_as_customer            0.362608
age                           0.475385
policy_state                 -0.038157
policy_deductable             0.476090
policy_annual_premium         0.035964
umbrella_limit                1.801424
insured_sex                   0.139324
insured_education_level       0.006286
insured_occupation           -0.055360
insured_hobbies              -0.061488
insured_relationship          0.078339
capital-gains                 0.466619
capital-loss                 -0.376884
incident_type                 0.090563
collision_type               -0.032778
incident_severity             0.277726
authorities_contacted        -0.114044
incident_state               -0.149255
incident_city                 0.043882
number_of_vehicles_involved   0.509725
property_damage               0.853093
bodily_injuries               0.003757
witnesses                     0.026211
police_report_available       0.796221
total_claim_amount           -0.593593
injury_claim                  0.271759
property_claim                0.361356
vehicle_claim                -0.620936
auto_make                    -0.028739
auto_model                   -0.073462
fraud_reported                1.188267
csl_per_person                0.098248
csl_per_accident             -0.094370
vehicle_age                   0.054522
dtype: float64
```

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

```
array([      0,  5000000,  6000000,  4000000,  3000000,  8000000,
        7000000,  9000000, 10000000, -1000000,  2000000], dtype=int64)
```

```
#Looks like umbrella_limit have few values and feels like catagorical, So we'll not change it's skewness.
int_label.remove('umbrella_limit')
```

Reducing skewness using PowerTransformer

As we can see that skewness is present in the dataset, we will try to reduce it using PowerTransformer

```
PT=PowerTransformer()
for i in int_label:
    if abs(new_df.loc[:,i].skew())>0.55:
        new_df.loc[:,i]=PT.fit_transform(new_df.loc[:,i].values.reshape(-1,1))
```

```
new_df.skew()
```

```
months_as_customer           0.362608
age                          0.475385
policy_state                -0.038157
policy_deductable            0.476090
policy_annual_premium        0.035964
umbrella_limit               1.801424
insured_sex                  0.139324
insured_education_level      0.006286
insured_occupation          -0.055360
insured_hobbies             -0.061488
insured_relationship         0.078339
capital-gains                0.466619
capital-loss                -0.376884
incident_type                0.090563
collision_type              -0.032778
incident_severity            0.277726
authorities_contacted       -0.114044
incident_state              -0.149255
incident_city                0.043882
number_of_vehicles_involved  0.509725
property_damage              0.853093
bodily_injuries              0.003757
witnesses                    0.026211
police_report_available      0.796221
total_claim_amount          -0.508540
injury_claim                 0.271759
property_claim               0.361356
vehicle_claim               -0.521805
auto_make                   -0.028739
auto_model                  -0.073462
fraud_reported               1.188267
csl_per_person               0.098248
csl_per_accident            -0.094370
vehicle_age                  0.054522
dtype: float64
```

Skewness has been greatly reduced.

Next step is to select the best features which would build the most accurate Machine Learning Models to predict the target variable.

Separating Input and Output Variables

```
x = new_df.drop("fraud_reported", axis=1)
y = new_df["fraud_reported"]
```

Feature Scaling

Scaling the values in the feature columns using MinMaxScaler.

```
from sklearn.preprocessing import MinMaxScaler
scale = MinMaxScaler() #Initializting MinMaxScaler
new = scale.fit(x)  #fitting our data into MinMaxScaller
scale_x = new.transform(x) #Transforming the data
#Setting up the coulumns after Scaling
scaled_x = pd.DataFrame(scale_x, index=x.index, columns=x.columns)
x=scaled_x
x.head() #Priting top 5 rows of our data
```

	months_as_customer	age	policy_state	policy_deductable	policy_annual_premium	umbrella_limit	insured_sex	insured_education_level	insured_occup
0	0.684760	0.644444	1.0	0.333333	0.606891	0.125	1.0	0.666667	0.1
1	0.475992	0.511111	0.5	1.000000	0.460404	0.750	1.0	0.666667	0.4
2	0.279749	0.222222	1.0	1.000000	0.611243	0.750	0.0	1.000000	0.8
3	0.534447	0.488889	0.0	1.000000	0.613059	0.875	0.0	1.000000	0.0
4	0.475992	0.555556	0.0	0.333333	0.730541	0.875	1.0	0.000000	0.8

5 rows × 33 columns

Building Machine Learning Models:

We will use a machine-learning algorithm to learn from the training set and use the model to predict the testing set and compare it with the predicted data with the target testing set to know how close the values. If the error between the predicted and target testing data is less that means the accuracy of the model is high and we can use this model to predict the result of similar datasets.

In this, we have used 7 Machine learning Algorithms

- LogisticRegression
- KneighborsClassifier
- AdaBoostClassifier
- SupportVectorClassifier (SVC)
- DecisionTreeClassifier
- RandomForestClassifier
- MLPClassifier

We can train and predict the data using the above 7 ML algorithms and save the model which has the highest frequency.

Finding the Best Random State

Let's find the best random state in which we can build the model.

(Random state ensures that the splits that you generate are reproducible. Scikit-learn use random permutations to generate the splits. The random state that you provide is used as a seed to the random number generator. This ensures that the random numbers are generated in the same order.)

```
maxAccu=0
maxRS=0
for i in range(1,200):
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=i)
    LR = LogisticRegression()
    LR.fit(x_train,y_train)
    predrf = LR.predict(x_test)
    acc = accuracy_score(y_test, predrf)
    if acc>maxAccu:
        maxAccu=acc
        maxRS=i
print("Best accuracy is",maxAccu," on Random_state ",maxRS)
```

```
Best accuracy is 0.8231292517006803  on Random_state  63
```

Creating Train-Test split based on random state obtained above:

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=.30,random_state=maxRS)
```

Training the Models Finding Best Algorithm

We will run various algorithms and check which has the highest Accuracy score, Cross Validation Score and (Accuracy Score - Cross Validation Score) for comparison between all the algorithms.

```
model=[LogisticRegression(),KNeighborsClassifier(),AdaBoostClassifier(),SVC(),DecisionTreeClassifier(),RandomForestClassifier(),MLPClassifier()]
for m in model:
    m.fit(x_train,y_train)
    pred=m.predict(x_test)
    print("*******************************************************************************")
    print("||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||")
    print('accuracy score of ->', m)
    print(accuracy_score(y_test,pred))
    print(confusion_matrix(y_test,pred))
    print(classification_report(y_test,pred))
    score=cross_val_score(m,x,y,cv=5)
    print(score)
    print(score.mean())
    print("Difference between Accuracy score and cross validation score is - ",accuracy_score(y_test,pred)-score.mean())
    print("||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||||")
    print("*******************************************************************************")
```

```
**********************************************************************************
accuracy score of -> LogisticRegression()
0.8231292517006803
[[218  16]
 [ 36  24]]
              precision    recall  f1-score   support

           0       0.86      0.93      0.89       234
           1       0.60      0.40      0.48        60

    accuracy                           0.82       294
   macro avg       0.73      0.67      0.69       294
weighted avg       0.81      0.82      0.81       294

[0.78571429 0.75510204 0.75       0.79081633 0.79081633]
0.7744897959183674
Difference between Accuracy score and cross validation score is - 0.04863945578231288
**********************************************************************************
accuracy score of -> KNeighborsClassifier()
0.7517006802721088
[[215  19]
 [ 54   6]]
              precision    recall  f1-score   support
```

```
              precision    recall  f1-score   support

           0       0.80      0.92      0.85       234
           1       0.24      0.10      0.14        60

    accuracy                           0.75       294
   macro avg       0.52      0.51      0.50       294
weighted avg       0.69      0.75      0.71       294

[0.72959184 0.70918367 0.69897959 0.75510204 0.71428571]
0.7214285714285713
Difference between Accuracy score and cross validation score is - 0.030272108843537482
**********************************************************************
accuracy score of -> AdaBoostClassifier()
0.826530612244898
[[209  25]
 [ 26  34]]
              precision    recall  f1-score   support

           0       0.89      0.89      0.89       234
           1       0.58      0.57      0.57        60

    accuracy                           0.83       294
   macro avg       0.73      0.73      0.73       294
weighted avg       0.83      0.83      0.83       294

[0.81122449 0.7755102  0.73979592 0.81632653 0.83673469]
0.7959183673469388
Difference between Accuracy score and cross validation score is - 0.030612244897959218
**********************************************************************
accuracy score of -> SVC()
0.7925170068027211
[[231   3]
 [ 58   2]]
              precision    recall  f1-score   support

           0       0.80      0.99      0.88       234
           1       0.40      0.03      0.06        60

    accuracy                           0.79       294
   macro avg       0.60      0.51      0.47       294
weighted avg       0.72      0.79      0.72       294

[0.75510204 0.75       0.75       0.76020408 0.76020408]
```

0.7551020408163265
Difference between Accuracy score and cross validation score is - 0.0374149659863946
****
accuracy score of -> DecisionTreeClassifier()
0.7891156462585034
[[197 37]
[25 35]]

	precision	recall	f1-score	support
0	0.89	0.84	0.86	234
1	0.49	0.58	0.53	60
accuracy			0.79	294
macro avg	0.69	0.71	0.70	294
weighted avg	0.81	0.79	0.80	294

[0.7755102 0.79591837 0.76530612 0.75 0.77040816]
0.7714285714285714
Difference between Accuracy score and cross validation score is - 0.017687074829932037
****
accuracy score of -> RandomForestClassifier()
0.7925170068027211
[[219 15]
[46 14]]

	precision	recall	f1-score	support
0	0.83	0.94	0.88	234
1	0.48	0.23	0.31	60
accuracy			0.79	294
macro avg	0.65	0.58	0.60	294
weighted avg	0.76	0.79	0.76	294

[0.7755102 0.79081633 0.7244898 0.76020408 0.78571429]
0.7673469387755102
Difference between Accuracy score and cross validation score is - 0.02517006802721089
****
accuracy score of -> MLPClassifier()
0.7993197278911565
[[210 24]
[35 25]]

	precision	recall	f1-score	support

```
              0       0.86      0.90      0.88       234
              1       0.51      0.42      0.46        60

       accuracy                           0.80       294
      macro avg       0.68      0.66      0.67       294
   weighted avg       0.79      0.80      0.79       294

[0.80612245 0.75       0.73979592 0.79591837 0.80102041]
0.7785714285714287
Difference between Accuracy score and cross validation score is - 0.02074829931972777
**********************************************************************
```

Analyzing Model Accuracies

Models	Accuracy Score	F1 Score	Difference between Accuracy score and Cross validation score
LogisticRegression	0.82	0.89	0.048
KNeighborsClassifier	0.75	0.85	0.030
AdaBoostClassifier	0.82	0.89	0.030
SVC	0.79	0.88	0.037
DecisionTreeClassifier	0.78	0.86	0.017
RandomForestClassifier	0.79	0.88	0.025
MLPClassifier	0.79	0.88	0.020

According to Cross val score and accuracy we can see that the DecisionTreeClassifier has the least difference between Accuracy and Cross val score, therefore we select DecisionTreeClassifier model.

Hyper Parameter Tuning

GridSearchCV was used for Hyper Parameter Tuning of the DecisionTreeClassifier model. Based on the input parameter values and after fitting the train datasets, the DecisionTreeClassifier Model was further tuned based on the parameter values yielded from GridsearchCV. Getting the list of the best parameters from Grid Search CV.

```
parameters={'n_estimators': range(0,20),
            'learning_rate': [0.1,0.01,0.001,0.0001,1],
            'algorithm':['SAMME', 'SAMME.R'],
            'random_state':range(0,20)}

clf=GridSearchCV(DecisionTreeClassifier(), parameters,cv=5)
clf.fit(x_train,y_train) #fitting train and test data
clf.best_params_  #Best parameters
```

```
{'algorithm': 'SAMME',
 'learning_rate': 0.1,
 'n_estimators': 1,
 'random_state': 0}
```

```
clf_pred=clf.best_estimator_.predict(x_test) #prediting result based on test based
```

```
accuracy_score(y_test,clf_pred) #finding accuracy score of the data
```

```
0.8435374149659864
```

```
print(accuracy_score(y_test,clf_pred))
print(confusion_matrix(y_test,clf_pred))
print(classification_report(y_test,clf_pred))
```

```
0.8435374149659864
[[204  30]
 [ 16  44]]
              precision    recall  f1-score   support

           0       0.93      0.87      0.90       234
           1       0.59      0.73      0.66        60

    accuracy                           0.84       294
   macro avg       0.76      0.80      0.78       294
weighted avg       0.86      0.84      0.85       294
```

The Final Accuracy Score of the final Model displayed accuracy of 84.35%
We successfully performed the Hyper Parameter Tuning on the Final Model.

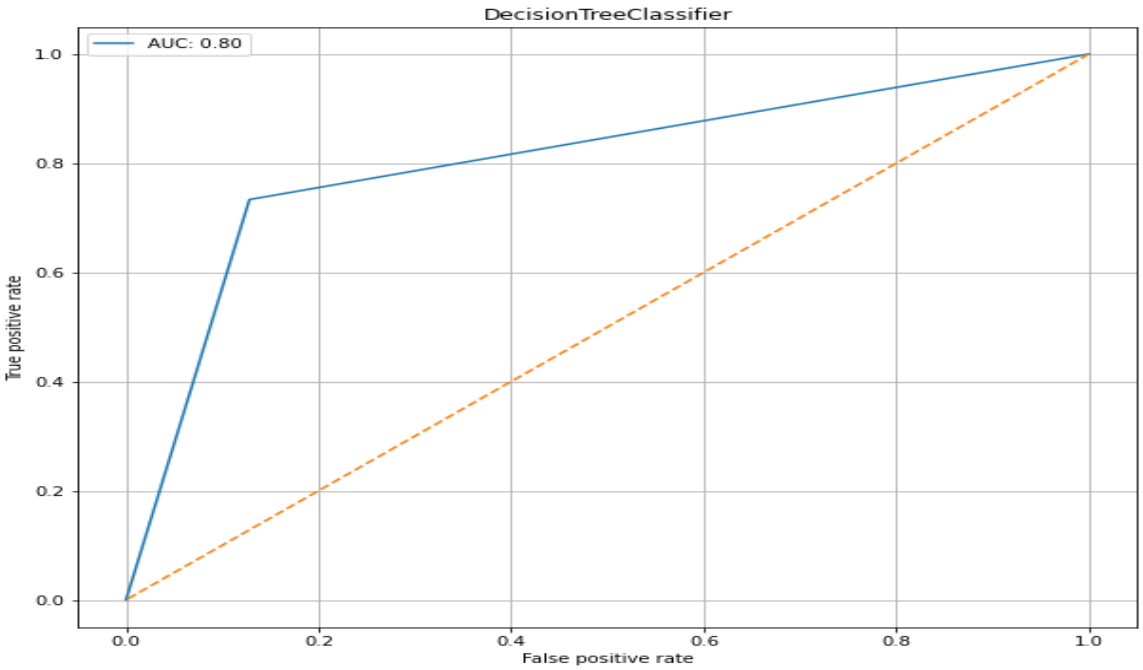
AUC ROC curve

AUC: Area Under the curve; ROC: Receiver Operator Characteristic

The greater the ROC score the better is the model. If ROC=1, then it perfectly fits.

If the maximum of the area falls under True positive then the model is doing good.

```
from sklearn.metrics import roc_curve,auc
fpr,tpr,thresholds=roc_curve(y_test,clf_pred) # calculating fpr, tpr
rf_auc = auc(fpr, tpr) #Model Accuracy
plt.figure(figsize=(10,9)) #plotting the figure, size of 10*9
plt.plot(fpr, tpr, label = 'AUC: %0.2f' % rf_auc)
plt.plot([1,0],[1,0], linestyle = '--')
plt.legend(loc=0) #adding accuracy score at bottom right
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('DecisionTreeClassifier')
plt.grid() #adding the grid
```



We can see that the ROC Score is 0.80 and the area under the curve falls under True Positive Rate, Therefore, we can conclude that the model is performing well and we need to save the model in joblib file for future use

Saving the model in joblib Format

```
import joblib
joblib.dump(clf.best_estimator_,"PJ10_Insurance.obj")
SVR_from_joblib=joblib.load("PJ10_Insurance.obj")
Predicted = SVR_from_joblib.predict(x_test)
Predicted
```

```
array([0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1,
       0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 0])
```

Prediction Conclusion

We will predict the "fraud_reported" target column using the final model sending the “x_test” set for predicting the “y_test” and then compare the original “y_test” and the predicted “y_test” in a dataframe.

```
pd.set_option("display.max_rows", None, "display.max_columns", None)
pd.DataFrame([SVR_from_joblib.predict(x_test)[:],y_test[:]],index=["Predicted","Original"])
```

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38
Predicted	0	0	0	0	0	1	0	0	0	1	0	0	1	0	1	1	0	1	1	0	0	0	0	1	0	0	1	1	0	1	0	0	0	1	1	1	1	0	1
Original	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	1	1	1	1	0	1

Concluding Remarks

In the beginning of the blog, we have discussed about the lifecycle of a Machine Learning Model, you can see how we have touched based on each point and finally reached up to the model building and made the model ready for deployment.

Let’s take a quick recap on all the steps that we went through starting from understanding the Problem Definition then going through the Data Analysis and EDA processes. We went through the necessary Pre-processing Data steps before the final Building Machine Learning Models step came into picture.

The Insurance industry area needs a good vision on data, and in every model building problem Data Analysis and Feature Engineering is the most crucial part.

You can see how we have handled numerical and categorical data and also how we build different machine learning models on the same dataset. And using hyper parameter tuning we can improve our model accuracy. Hence, we ended up with a prediction accuracy of 95% after all the above steps are completed.

Using this machine Learning Model, we can predict whether the insurance claim is fraudulent or not and we could reject those application which will be considered as fraud claims. Saving the Insurance industry from possible fraudsters.