Insurance Claims- Fraud Detection

What is Insurance fraud?

Insurance fraud means when a claimant claims fake loss covered under an insurance policy to get some benefit or advantage. The claimer creates an imaginary situation and shows it is a real incident and he claims losses.

Problem Statement

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

In this project, you are provided 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.
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.

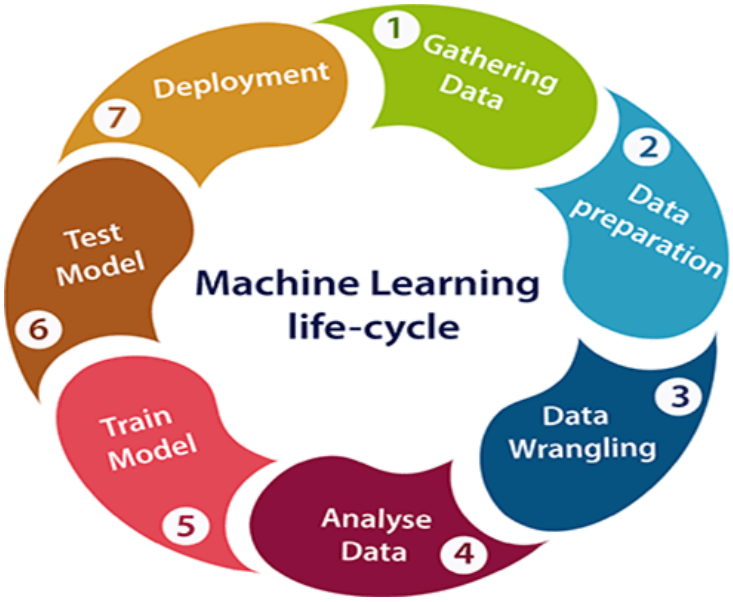
Libraries Used

- Numpy
- Pandas
- Seaborn
- Matplotlib
- Python
- Scikit Learn

Machine learning life cycle is a process of data analysis. This process starts from data collection to data cleaning and processing, and finally to building a model that can be used for prediction.

The process of machine learning consists of four steps: Data Collection, Data Cleaning and Processing, Model Building, and Application.

Data Collection involves collecting the raw data from the sources like web pages or images. Data Cleaning and Processing includes removing the noise from the raw data by using pre-processing techniques like feature extraction or dimensionality reduction. Model Building is about developing a predictive model that can be used for prediction in the future. Application is about applying the model to make predictions on new data examples.



Exploratory data analysis

What is exploratory data analysis (EDA) ?

When we are trying to make any machine learning model, EDA is a very important step that allows us to look at data from different perspectives and try to get familiar with the dataset. If we have a good understanding of the data then we can choose the best suitable machine-learning model.

Let's move to our EDA process

Importing Necessary libraries which will be required to perform EDA

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

- pandas and NumPy are must libraries when we talk about ML models
- Seaborn and matplotlib allow us to visualize data in a very simple way.

Importing Dataset

```
df=pd.read_csv(r'E:\Data Science Certificates\Evaluation project\Insurance Claims- Fraud Detection\Automobile_insurance_fraud.csv')
df
```

Checking the shape of a dataset as we can see we have 1000 rows and 40 columns.

```
df.shape
```

```
(1000, 40)
```

Using head method we can observe more information about the data. below are some points I can observe. Head method always shows is first five columns.

- Numerical and object values are present in Dataset.
- Which column contains what kind of information?
- Is there any columns that I can transform and make more feasible for ML model?

```
df.head()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	...	police_report_avai
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	...	

5 rows × 40 columns

While I was going through column information I noticed that we have a “policy_number” Column present in the dataset which has every value unique. Hence I decided to drop this column because as every data has uniqueness hence ML model will not be able to extract meaningful full information from this column.

Dropping column

```
# I am droping policy_number from data sent as it is always unique for all policies

df.drop("policy_number",axis=1,inplace=True)
```

Using below sample technique I can pick any random rows by giving a count of rows and can review random lines instead on top or bottom lines

```
df.sample(3)
```

	months_as_customer	age	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	...	police_report_avail
691	199	38	26-09-1991	IL	250/500	1000	1281.25	0	467780	FEMALE	...	
851	282	43	19-07-2006	OH	250/500	500	1452.27	0	611996	MALE	...	
756	92	32	29-01-1998	IL	500/1000	500	1592.41	0	474324	MALE	...	

3 rows × 39 columns

Below are the names of all columns which are present in Dataset.

```
#Let us check the name of our columns
df.columns
```

```
Index(['months_as_customer', 'age', 'policy_bind_date', 'policy_state',
       'policy_csl', 'policy_deductable', 'policy_annual_premium',
       'umbrella_limit', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       'insured_relationship', 'capital-gains', 'capital-loss',
       'incident_date', 'incident_type', 'collision_type', 'incident_severity',
       'authorities_contacted', 'incident_state', 'incident_city',
       'incident_location', 'incident_hour_of_the_day',
       'number_of_vehicles_involved', 'property_damage', 'bodily_injuries',
       'witnesses', 'police_report_available', 'total_claim_amount',
       'injury_claim', 'property_claim', 'vehicle_claim', 'auto_make',
       'auto_model', 'auto_year', 'fraud_reported', '_c39'],
      dtype='object')
```

Checking for duplicated values in the dataset so that we can remove them from the data as we should not keep any duplicated rows in the dataset while training the ML model. As we can see below there are no duplicate values present in the dataset.

```
df.duplicated().sum()
```

```
0
```

Checking null values because if we have any null value present in Dataset then we should impute those values by using various imputing methods.

I only found one column where null values are present in Dataset as per below.

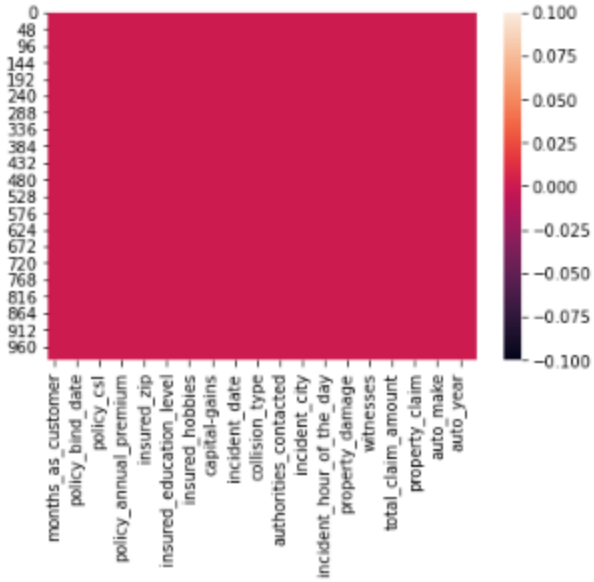
```
auto_year                 0
fraud_reported            0
_c39                   1000
```

We have observed previously that we have a total of 1000 rows and in “_c39” there are 1000 missing values which means we don't have single values present in this column hence we will drop this column for better results.

```
# Droping _c39

df.drop("_c39",axis=1,inplace=True)
```

```
# Visaulize Null values on heatmap
sns.heatmap(df.isna())
```



We have checked null values by using heatmap after removing columns “_c39” and as we can see there are no null values present in the dataset.

Checking unique values

Unique value shows us the uniqueness of data present in each column and looking at this we get some understanding of how much different values are present in the dataset.

```
df.nunique()
```

We found that “incident_location” column has 100 unique value hence we will drop this column again as we did previously for “_c39”

```
# Dropping "incident_location"

df.drop("incident_location",axis=1,inplace=True)
```

Value Counts

We are checking the below count of each value in each column which we give us an idea about which values are highly occurring in data and which are lowest one

```
#Now Let us find the values of each Features in our data
for i in df.columns:
        print(df[i].value_counts())
        print('*'*100)
```

Observation

- collisison_type has "?" present
- Property_damage has "?" present
- police_report_available "?: present

All above "?" we can represent with some meaning full name like "not_sure" etc.

- Finally "fraud_reported" count of yes and no is imbalanced which we need to balance before train model

Date formats

- "policy_bind_date"
- "Incident_date"

As we decided above we have converted values below.

```
# First we will treat "?" by filling some word there just to have meaning

df['collision_type']=df['collision_type'].replace("?","not_sure")
```

```
df['property_damage']=df['property_damage'].replace("?","not_sure")
df['police_report_available']=df['police_report_available'].replace("?","not_sure")
```

```
col=['collision_type','property_damage','police_report_available']

for i in col:
        print(df[i].value_counts())
        print('*'*100)
```

```
Rear Collision     292
Side Collision     276
Front Collision    254
not_sure           178
Name: collision_type, dtype: int64
****************************************************************************************************
not_sure    360
NO          338
YES         302
Name: property_damage, dtype: int64
****************************************************************************************************
NO          343
not_sure    343
YES         314
Name: police_report_available, dtype: int64
****************************************************************************************************
```

Date Format Handling

We should convert the Date format using the datetime module so that our machine learning model understands data. I am separating the date into year, date, and day.

```
# Datetime for policy_bind_date
df['policy_bind_date']=pd.to_datetime(df['policy_bind_date'])
df['Month'] = pd.DatetimeIndex(df['policy_bind_date']).month
df['Year'] = pd.DatetimeIndex(df['policy_bind_date']).year
df['Day'] = df['policy_bind_date'].dt.day
```

Dropping the date column as we already stored values in new separate columns.

```
# we will drop original "policy_bind_date" as we already extracted dates
df.drop(['policy_bind_date'],axis=1,inplace=True)
df
```

Same step I am following for “Incidant_date”

```
df['incident_date']=pd.to_datetime(df['incident_date'])
df['incident_Month'] = pd.DatetimeIndex(df['incident_date']).month
df['incident_ear'] = pd.DatetimeIndex(df['incident_date']).year
df['incident_Day'] = df['incident_date'].dt.day
```

```
df.drop('incident_ear',axis=1,inplace=True)
```

Using the below code I am separating object and numeric datatypes in one variable so that I can use these variables for better visualization if required.

```
# Getting object data
object_datatype = []
for x in df.dtypes.index:
    if df.dtypes[x] == 'O':
        object_datatype.append(x)
print(f"Object Data Type Columns are:\n ",object_datatype)

# getting integers data
number_datatype = []
for x in df.dtypes.index:
    if df.dtypes[x] == 'float64' or df.dtypes[x] == 'int64':
        number_datatype.append(x)
print(f"\nNumber Data Type Columns are:\n ",number_datatype)
```

```
Object Data Type Columns are:
  ['policy_state', 'policy_csl', '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_availabl
e', 'auto_make', 'auto_model', 'fraud_reported']

Number Data Type Columns are:
  ['months_as_customer', 'age', 'policy_deductable', 'policy_annual_premium', 'umbrella_limit', 'insured_zip', 'capital-gains', 'capital-loss', 'incid
ent_hour_of_the_day', 'number_of_vehicles_involved', 'bodily_injuries', 'witnesses', 'total_claim_amount', 'injury_claim', 'property_claim', 'vehicle_
claim', 'auto_year', 'Month', 'Year', 'Day', 'incident_Month', 'incident_Day']
```

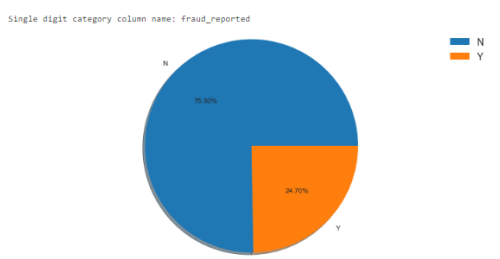
I have just separated columns on basis od Datatypes just because if required in visualization purpose

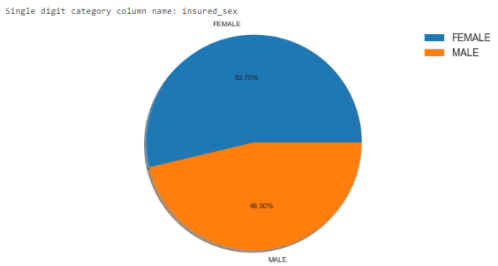
Data visualization

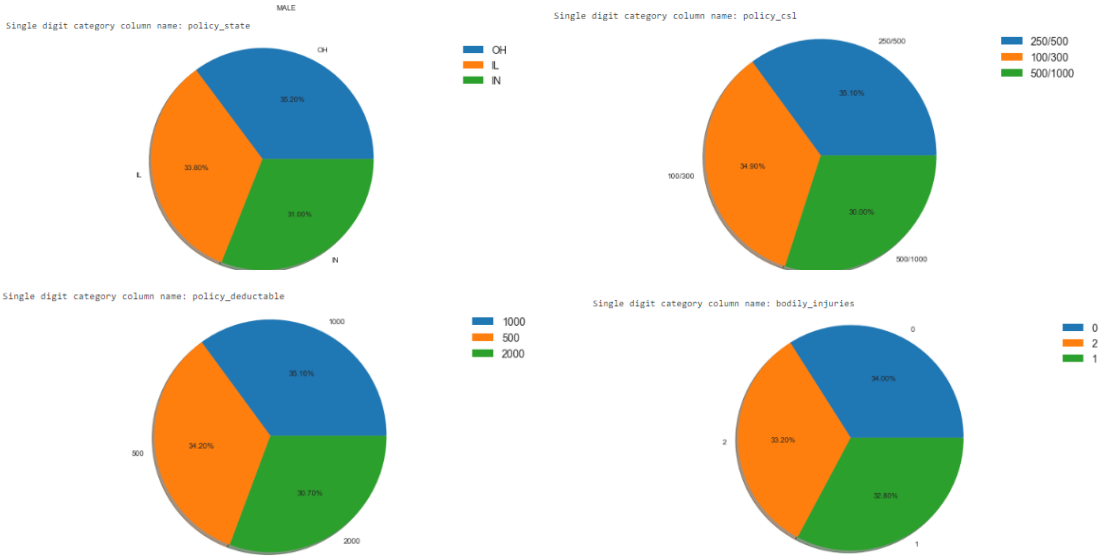
```
def pie_chart(x):
    plt.style.use('seaborn-white')
    plt.figure(figsize=(10,5))
    plt.pie(x.value_counts(), labels=x.value_counts().index, shadow=True, autopct='%1.2f%%')
    plt.legend(prop={'size':14})
    plt.axis('equal')
    plt.tight_layout()
    return plt.show()

col1 = ['fraud_reported', 'insured_sex', 'policy_state', 'policy_csl', 'policy_deductable', 'bodily_injuries',
        'police_report_available', 'property_damage', 'incident_type', 'collision_type', 'incident_severity',
        'number_of_vehicles_involved', 'witnesses', 'authorities_contacted', 'insured_relationship',
        'insured_education_level', 'incident_state', 'incident_city']

for i in df[col1]:
    print(f"Single digit category column name:", i)
    pie_chart(df[i])
```





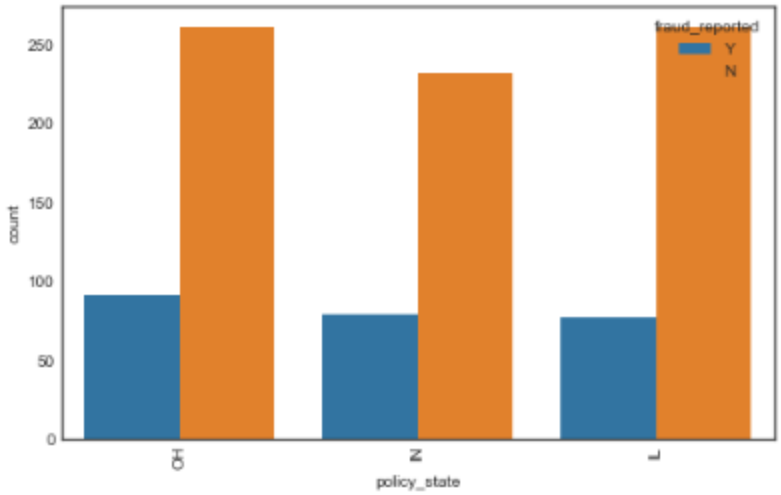


I have got a pie chart as per above for 40 variables that we have and below are some observations based on those charts.

Observation

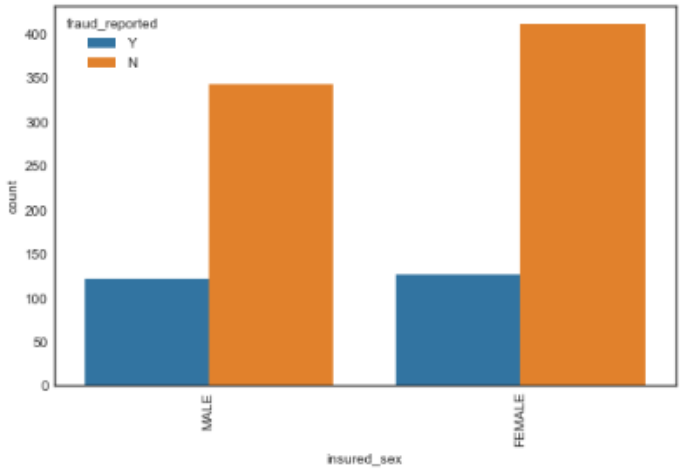
- we can see approx 24% of the data has fraud activity which we need to balance
- Female has more contribution to a policy as we can see female percentages are up to 53%
- Maximum policy are from OH state
- policy deductible is high in 1000 range
- 65 % people are saying that they are bodily_injured
- Many of them approx 34% claims don't have police report available
- we have 30% property damage and 33% no damage
- In incident type we can see Parked car vehicle incident are very less, they are only 8 % however multivehicle collision is high in the range up to 41%
- Rear collision is high in percentage as we can see it is up to 29%
- 58% data says that there were single vehicle involved
- Witness are equally distributed
- We can see that most of the incident or highest incident rate is in NY state followed by SC
- Springfield city has the highest rate of the incident as we can see up to 15 % incident from this city

```
plt.figure(figsize=(8,5))
sns.countplot(df['policy_state'],hue=df['fraud_reported'])
plt.xticks(rotation=90)
plt.show()
```



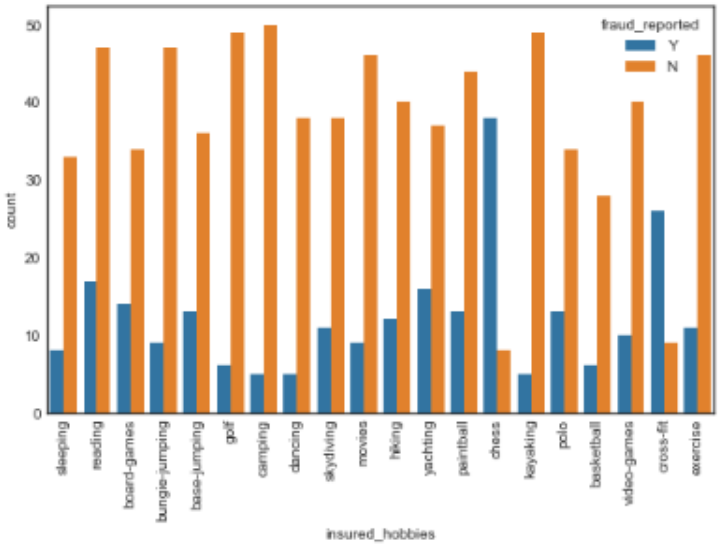
We can see fraud cases are high in OH state and for other two state they are almost same

[28]:
```
plt.figure(figsize=(8,5))
sns.countplot(df['insured_sex'],hue=df['fraud_reported'])
plt.xticks(rotation=90)
plt.show()
```



Female have done more fraudelent activity as we can see in above plot

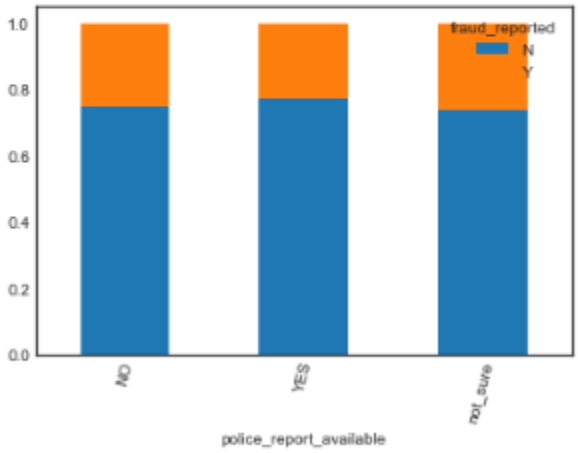
[29]:
```
plt.figure(figsize=(8,5))
sns.countplot(df['insured_hobbies'],hue=df['fraud_reported'])
plt.xticks(rotation=90)
plt.show()
```



We can see fraud in high in Chess catagory

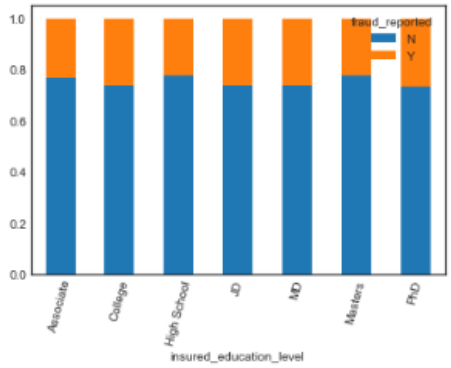
Bivariate Analysis

]:
```
ba = pd.crosstab(df.police_report_available, df.fraud_reported,normalize='index')
ba.plot.bar(stacked=True)
plt.xticks(rotation=75)
plt.show()
```



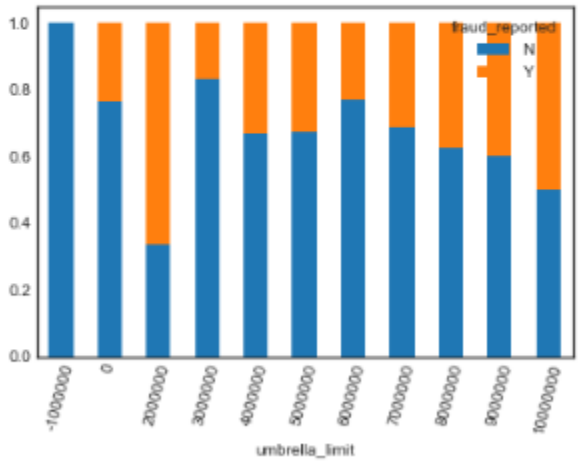
Those who dont have polivy report they might be potentila frauds as we can see count is hight in above graph

```
ba = pd.crosstab(df.insured_education_level, df.fraud_reported,normalize='index')
ba.plot.bar(stacked=True)
plt.xticks(rotation=75)
plt.show()
```



College students and Phd educated are highest contribution in Fraud

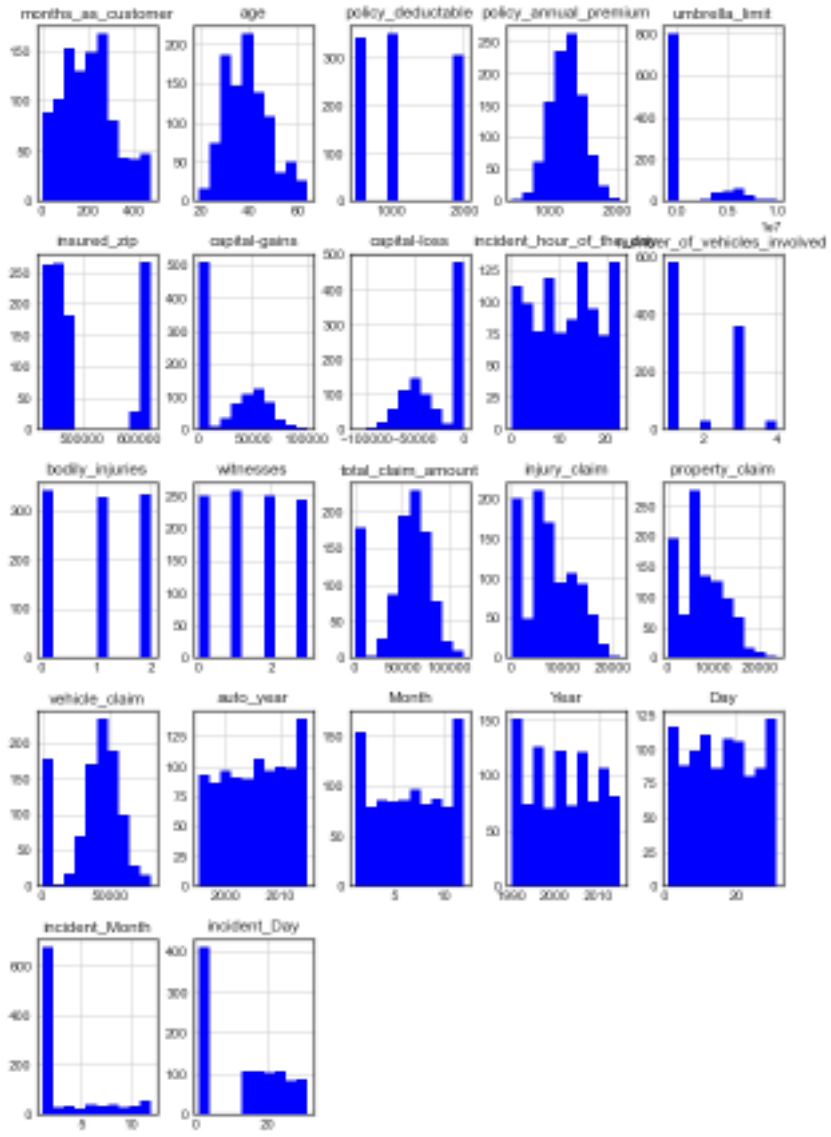
```
ba = pd.crosstab(df.umbrella_limit , df.fraud_reported,normalize='index')
ba.plot.bar(stacked=True)
plt.xticks(rotation=75)
plt.show()
```



Those who are having high umbrella_limit they are high in fraud count

We can see the below hist plot, which shows us a distribution of every column.

```
df.hist(figsize=(10,15),color = 'blue');
```



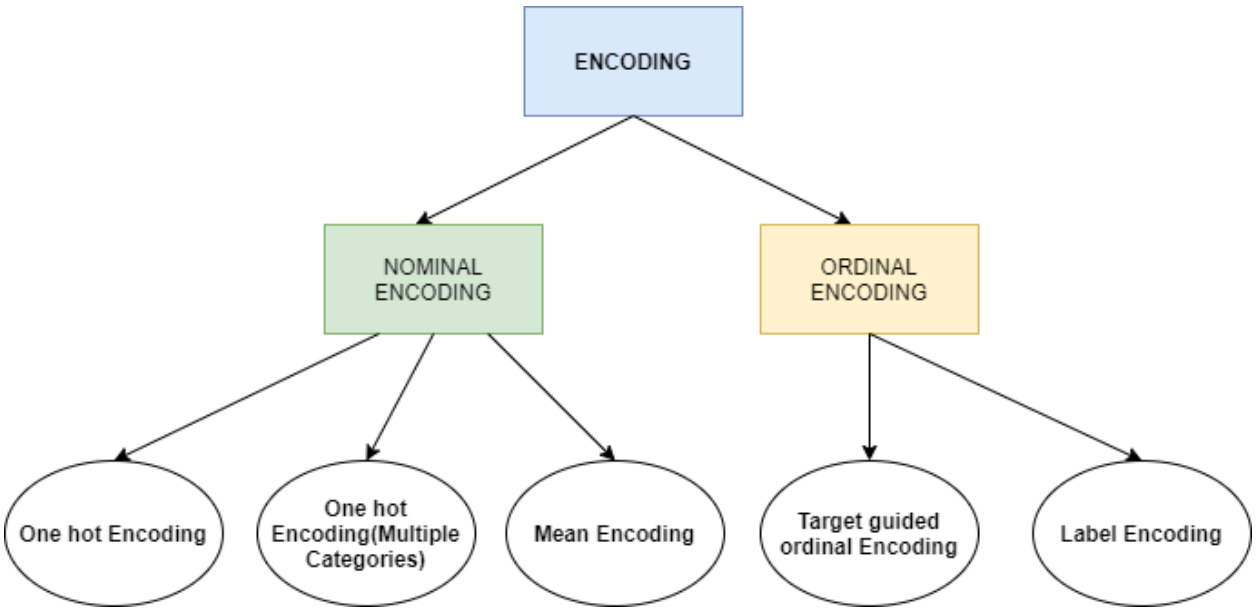
Encoding

Encoding is the process of transforming a string of symbols into a sequence of numbers. It is essential to machine learning because it allows the computer to store data and retrieve it later.

Machine learning models are trained on encoded data, so they can only work with encoded data. The input and output are both in terms of encoded strings, which means that they are not human-readable.

Encoding helps in storing and retrieving information when the size of the dataset is too large to be stored or processed in its raw form.

Below are some examples of Encoding



I am using Labelencoder to transform Objest Datatypes.

```
objects = []
for i in df.dtypes.index:
    if df.dtypes[i] == 'O':
        objects.append(i)
```

```
# Handling Objest using LabelEncoder
from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()

for i in objects:
    df[i]=le.fit_transform(df[i].astype(str))
```

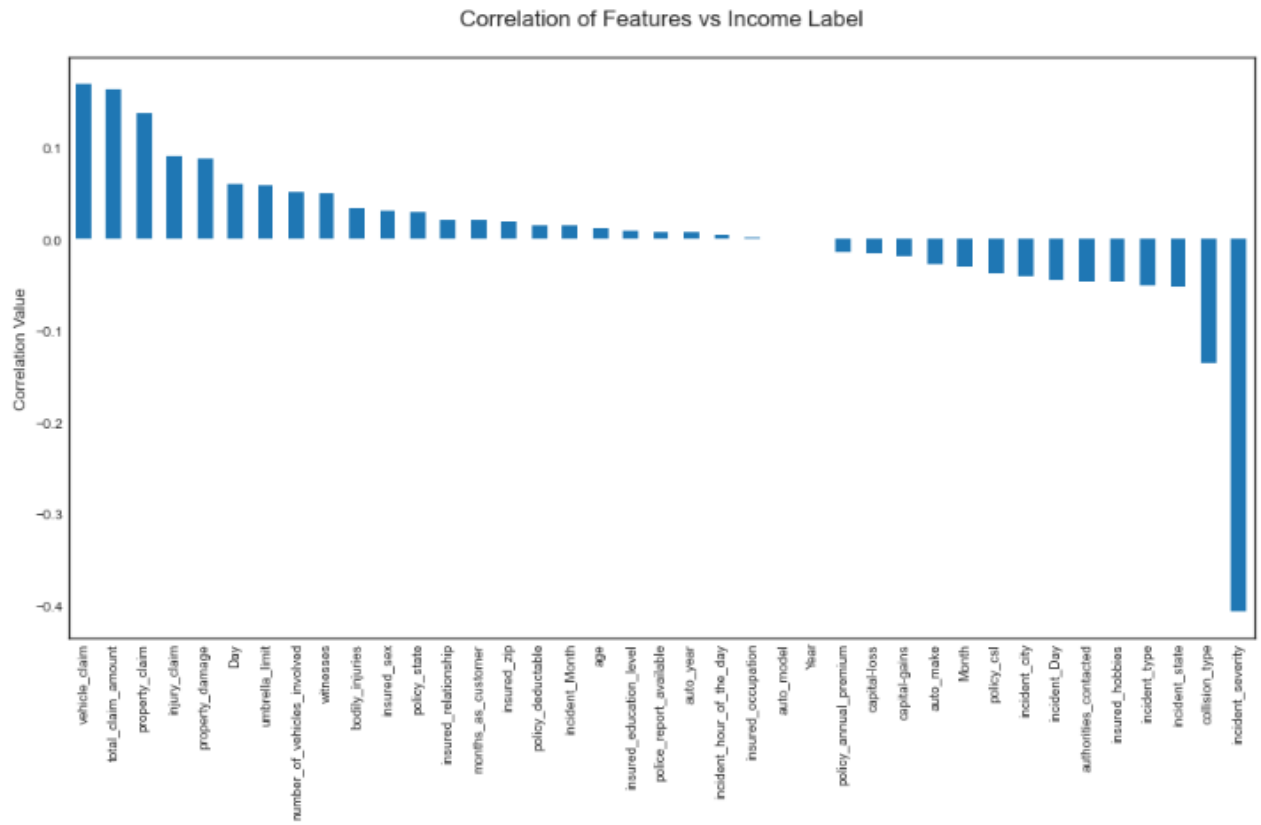
```
df.dtypes
```

```
months_as_customer                    int64
age                                   int64
policy_state                          int32
policy_csl                            int32
policy_deductable                     int64
policy_annual_premium               float64
umbrella_limit                        int64
insured_zip                           int64
insured_sex                           int32
insured_education_level               int32
insured_occupation                    int32
insured_hobbies                       int32
insured_relationship                  int32
capital-gains                         int64
capital-loss                          int64
incident_date                datetime64[ns]
incident_type                         int32
collision_type                        int32
incident_severity                     int32
authorities_contacted                 int32
incident_state                        int32
incident_city                         int32
incident_hour_of_the_day              int64
number_of_vehicles_involved           int64
property_damage                       int32
bodily_injuries                       int64
witnesses                             int64
police_report_available               int32
total_claim_amount                    int64
injury_claim                          int64
property_claim                        int64
vehicle_claim                         int64
auto_make                             int32
auto_model                            int32
auto_year                             int64
fraud_reported                        int32
Month                                 int64
```

We can see that all columns are transformed to Int32 or int64 format which is accepted by ML modules.

What is meant by Correlation ?

Correlation is a statistical measure of the strength of a linear relationship between two variables. The Pearson product-moment correlation coefficient, also known as the Pearson correlation coefficient, is a widely used measure of the strength of a linear association between two variables. It ranges from −1 to +1, where 1 indicates total positive linear association, 0 indicates no linear association, and −1 indicates total negative linear association.



- We can see that half of the columns are positively correlated and more than 40% are negatively correlated we need both to train the model

I can see that there are some columns with almost zero correlation which I can drop from the data set. They are as below

- Year
- auto_model
- Insured_occupation

Dropping column which doesn't have a correlation with the target column.

```
# Dropping no correalted columns

df.drop('Year',axis=1,inplace=True)
df.drop('auto_model',axis=1,inplace=True)
df.drop('insured_occupation',axis=1,inplace=True)
```

Preprocessing pipeline

What are outliers ?

Outliers are individuals who do not fit into the norm. They are different from others and often have a significant impact on the business world.

Outliers can be found in all areas of life, but they are more common in the business world than anywhere else. We should remove outliers because they represent measurement errors, data entry or processing errors, or poor sampling.

Outliers checking and handling

```
plt.figure(figsize=(25,35))

graph=1

for i in df:
    if graph<=38:
        plt.subplot(10,4,graph)
        sns.boxplot(data=df[i])
        plt.xlabel(i)
    graph+=1

plt.show
```

Outliers detected in the below columns

- age
- total claim amount
- incident_Month
- Property claim

Dropping data with 2 % acceptable data loss.

```
# Lets Treat outliers

from scipy.stats import zscore
zscore=zscore(df)
zabs=np.abs(zscore)

z=df[(zabs<3).all(axis=1)]
```

```
z.shape
```

```
(980, 37)
```

```
df.shape
```

```
(1000, 37)
```

```
loss=(1000-980)/1000*100
loss
```

```
2.0
```

2% data loss is acceptable to remove outliers

```
dropindex=df.index.difference(z.index)
dropindex
```

```
Int64Index([ 31,  48,  88, 115, 119, 229, 248, 262, 314, 430, 458, 500, 503,
            657, 700, 763, 807, 875, 922, 975],
           dtype='int64')
```

```
df.drop(dropindex,inplace=True)
```

Skewness of data

What is skewness ?

Skewness is a measure of the asymmetry of a distribution. It defines how easy it is to predict the location of an observation in relation to its mean or average.

Skewness can be negative or positive, but it depends on what type of distribution you are looking at. Positive skewness means that there are more observations on one side than on the other side, and negative skewness means that there are more observations on both sides than in the middle.

In many regression algorithms (e.g. linear regression) normality of residuals is assumed. That is, the error between the predictions and

actual responses is normally (Gaussian) distributed. Having skewed data may manifest itself in the learning function and make the model produce a skewed distribution of residuals.

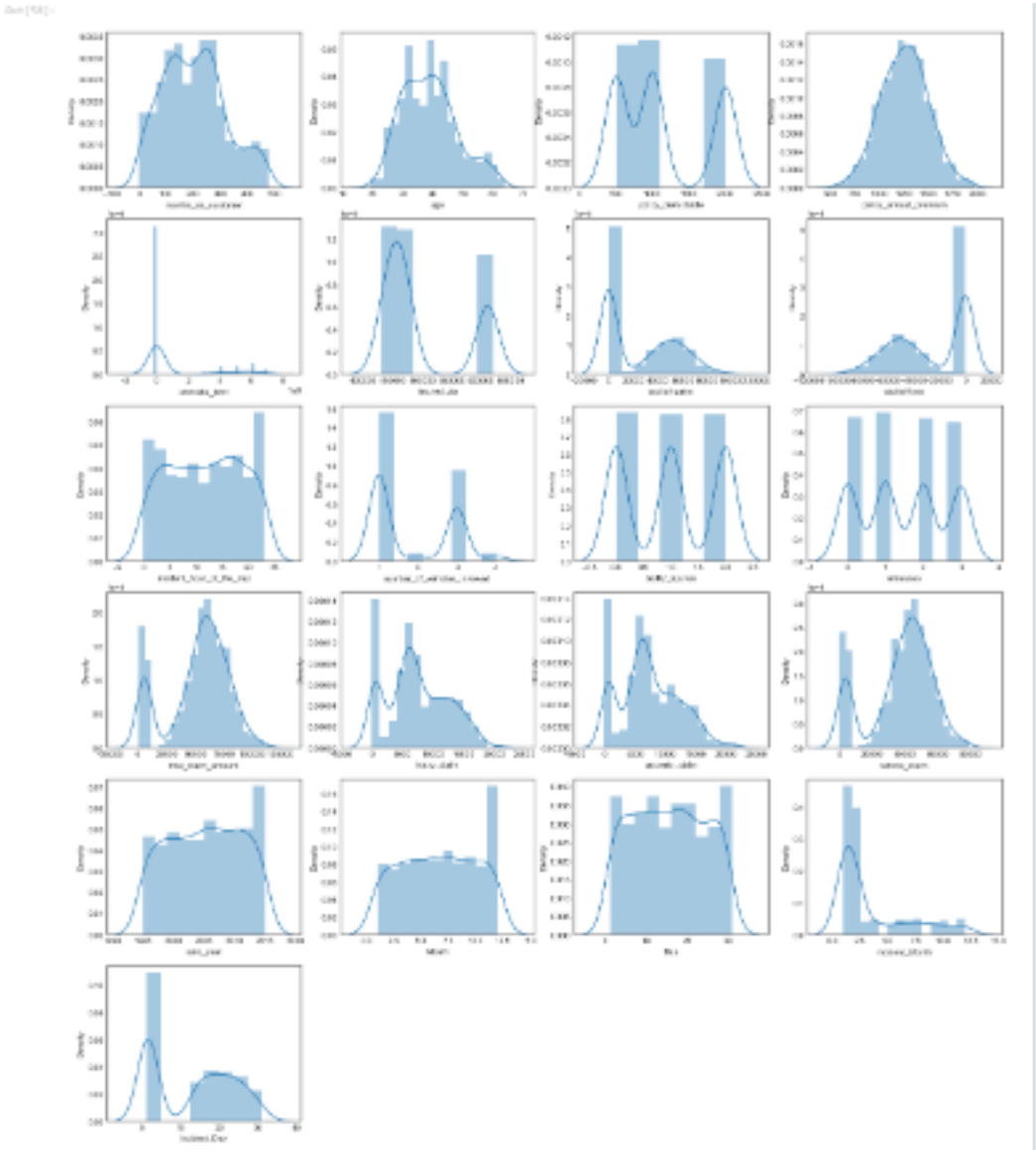
Checking skewness using distplot

```
plt.figure(figsize=(20,25),facecolor="white")

plot=1

for i in number_datatype:
    if plot<=22:
        plt.subplot(6,4,plot)
        sns.distplot(df[i])
        plt.xlabel(i)
    plot+=1

plt.show
```



Below are skewed columns as I can observe

- months_as_customer
- age
- policy_deductable
- umbrella_limit
- insured_zip
- capital-gains
- capital-loss
- number_of_vehicles_involved
- total_claim_amount
- vehicle_claim
- incident_Month

Removing Skewness using yeo-johnson method

In [59]:
```
skewed_features = ['months_as_customer','age','policy_deductable','umbrella_limit','insured_zip','capital-gains','capital-loss','number_of_vehicles_in

from sklearn.preprocessing import PowerTransformer
scale = PowerTransformer(method='yeo-johnson')
```

In [60]:
```
df[skewed_features] = scale.fit_transform(df[skewed_features].values)
df[skewed_features].head()
```

Out[60]:

	months_as_customer	age	policy_deductable	umbrella_limit	insured_zip	capital-gains	capital-loss	number_of_vehicles_involved	total_claim_amount	vehicle_claim	incident_Mo
0	1.044541	1.002330	0.063154	-0.337086	-5.551115e-16	1.028989	1.048516	-0.839387	0.720163	0.758130	-1.100
1	0.299045	0.427171	1.271808	1.482455	-4.996004e-16	-0.986627	1.048516	-0.839387	-1.778146	-1.787485	-1.100
2	-0.515819	-1.136013	1.271808	1.482455	-1.720846e-15	0.955106	1.048516	1.194880	-0.715924	-0.820246	-0.021
3	0.517526	0.324015	1.271808	1.788485	2.164935e-15	1.013773	-1.006291	-0.839387	0.395030	0.681832	1.557
4	0.299045	0.626869	0.063154	1.788485	2.164935e-15	1.066686	-0.941672	-0.839387	-1.730904	-1.740862	-0.021

We have removed outliers and skewness from data and now we are ready to define our x and y where x means independent variable and y is dependent variable for which we are training machine learning model to predict values.

Before processing, I would like to mention here this problem is a classification problem as we have to predict if an insurance claim is a fraud or not. So we only have two categories in the target variable on are yes and no which we have already encoded to o and 1.

Defining x and y variables using the below code.

```
x = df.drop('fraud_reported', axis=1)
y = df['fraud_reported']
x.shape
```

```
(980, 36)
```

```
In [63]: y.shape

Out[63]: (980,)
```

We have see in visualization part about class imbalanec which we need to treat before processing

```
In [64]: y.value_counts()

Out[64]: 0    740
         1    240
         Name: fraud_reported, dtype: int64
```

As we can see count of y above we can understand this is the case of imbalance data.

What is meant by an imbalanced dataset?

The imbalanced dataset problem is a common problem in machine learning. It occurs when there are significantly more data points of one type than another.

An example of an imbalanced dataset would be a set of data that contains 99% cats and 1% dogs.

Handling Imbalance data using SMOT oversampling technique

What is SMOT ?

SMOT oversampling is a technique used in machine learning to improve the performance of a classifier. It is an example of "smart" sampling techniques that are designed to reduce the error rate and improve performance.

This technique works by over-sampling training data from the minority class and under-sampling data from the majority class. The goal is to increase the number of examples from the minority class so that it becomes more likely that there will be enough examples for each possible outcome in testing, which will lead to a more accurate classification.

SMOT for oversampling

In [65]:
```
# adding samples to make all the categorical label values same
from imblearn.over_sampling import SMOTE # importing SMOT
oversample = SMOTE()
x, y = oversample.fit_resample(x, y)
```

In [66]:
```
y.value_counts()
```

Out[66]:
```
1    740
0    740
Name: fraud_reported, dtype: int64
```

Our classes are baalancesd now that means we have added dummy information.

We can see above over classes are balanced now and we can move ahed.

Feature Scaling.

What is feature scaling?

Feature scaling is a technique used to rescale features in machine learning. It helps to avoid the problem of overfitting, which is when a model performs well on training data but poorly on new data.

There are two types of feature scaling: linear and non-linear. Linear feature scaling can be done by using the mean, median or mode for each feature. Non-linear feature scaling can be done by using the square root or logarithm of each feature.

Here in this problem, I will use StanderedScaler to scale our data

```
from sklearn.preprocessing import StandardScaler # Using standard scaller

scaler = StandardScaler()
x = pd.DataFrame(scaler.fit_transform(x), columns=x.columns)
x.head()
```

	months_as_customer	age	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip	insured_sex	insured_education_level	...	total_c
0	1.086265	1.051720	1.353245	0.234426	0.040820	0.659089	-0.403986	-0.422330	1.274407	0.642404	...	
1	0.299301	0.447775	0.091240	0.234426	1.311387	-0.264759	1.608405	-0.381038	1.274407	0.642404	...	
2	-0.560890	-1.193644	1.353245	-1.079781	1.311387	0.686537	1.608405	-1.289465	-0.784679	1.732097	...	
3	0.529934	0.339455	-1.170766	0.234426	1.311387	0.697992	1.946872	1.600985	-0.784679	1.732097	...	
4	0.299301	0.657467	-1.170766	1.548633	0.040820	1.438912	1.946872	1.600985	1.274407	-1.536983	...	

5 rows × 36 columns

We did above feature Scalling to avoid distance bias between values and calculation of them

We are ready to train our different models now

Importing all models and trying to find the best random state.

Finding best random stage

In [80]:

```
import sklearn
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import GradientBoostingClassifier, BaggingClassifier

#metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import r2_score,mean_squared_error
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
```

```
maxAcc = 0
maxRS = 0

for i in range(1,1000):
    x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=.20, random_state=i)
    modRF = RandomForestRegressor()
    modRF.fit(x_train,y_train)
    pred = modRF.predict(x_test)
    acc = r2_score(y_test,pred)
    if acc>maxAcc:
        maxAcc = acc
        maxRs=i
print(f"Best Accuracy is: {maxAcc} on Random State: {maxRs}")
```

```
Best Accuracy is: 0.7452394000365832 on Random State: 44
```

As we can see above we got our best random state. The purpose of getting the best random state is to get the highest accuracy using the best random state.

Below are the different models we have tried using the 44 best random state.

```
LOGR = LogisticRegression(solver = 'liblinear')
RFC = RandomForestClassifier()
DTC = DecisionTreeClassifier()
SV = SVC()
KNN = KNeighborsClassifier()
EXT=ExtraTreesClassifier()
Gbc=GradientBoostingClassifier()
BG=BaggingClassifier()

model=[LOGR,RFC,DTC,SV,KNN,EXT,Gbc,BG]


for i in model:
    i.fit(x_train,y_train)
    predi=i.predict(x_test)
    print('performance matrix of',i,'is:')
    print("Confusion matrix :",confusion_matrix(y_test,predi))
    print("Classification report :", classification_report(y_test,predi,))
    print("Accuracy score :", accuracy_score(y_test,predi))
    print("**********************************************************")
```

```
performance matrix of LogisticRegression(solver='liblinear') is:
Confusion matrix : [[117  25]
 [ 26 128]]
Classification report :               precision    recall  f1-score   support

           0       0.82      0.82      0.82       142
           1       0.84      0.83      0.83       154

    accuracy                           0.83       296
   macro avg       0.83      0.83      0.83       296
weighted avg       0.83      0.83      0.83       296

Accuracy score : 0.8277027027027027
**********************************************************************
performance matrix of RandomForestClassifier() is:
Confusion matrix : [[126  16]
 [ 19 135]]
Classification report :               precision    recall  f1-score   support

           0       0.87      0.89      0.88       142
           1       0.89      0.88      0.89       154

    accuracy                           0.88       296
   macro avg       0.88      0.88      0.88       296
weighted avg       0.88      0.88      0.88       296

Accuracy score : 0.8817567567567568
```

```
performance matrix of DecisionTreeClassifier() is:
Confusion matrix : [[113  29]
 [ 15 139]]
Classification report :              precision    recall  f1-score   support

           0       0.88      0.80      0.84       142
           1       0.83      0.90      0.86       154

    accuracy                           0.85       296
   macro avg       0.86      0.85      0.85       296
weighted avg       0.85      0.85      0.85       296

Accuracy score : 0.8513513513513513
****************************************************************
performance matrix of SVC() is:
Confusion matrix : [[129  13]
 [ 27 127]]
Classification report :              precision    recall  f1-score   support

           0       0.83      0.91      0.87       142
           1       0.91      0.82      0.86       154

    accuracy                           0.86       296
   macro avg       0.87      0.87      0.86       296
weighted avg       0.87      0.86      0.86       296

Accuracy score : 0.8648648648648649
```

```
performance matrix of KNeighborsClassifier() is:
Confusion matrix : [[ 45  97]
 [  4 150]]
Classification report :               precision    recall  f1-score   support

           0       0.92      0.32      0.47       142
           1       0.61      0.97      0.75       154

    accuracy                           0.66       296
   macro avg       0.76      0.65      0.61       296
weighted avg       0.76      0.66      0.62       296

Accuracy score : 0.6587837837837838
**********************************************************************
performance matrix of ExtraTreesClassifier() is:
Confusion matrix : [[127  15]
 [ 17 137]]
Classification report :               precision    recall  f1-score   support

           0       0.88      0.89      0.89       142
           1       0.90      0.89      0.90       154

    accuracy                           0.89       296
   macro avg       0.89      0.89      0.89       296
weighted avg       0.89      0.89      0.89       296

Accuracy score : 0.8918918918918919
```

```
performance matrix of GradientBoostingClassifier() is:
Confusion matrix : [[126  16]
 [ 10 144]]
Classification report :               precision    recall  f1-score   support

           0       0.93      0.89      0.91       142
           1       0.90      0.94      0.92       154

    accuracy                           0.91       296
   macro avg       0.91      0.91      0.91       296
weighted avg       0.91      0.91      0.91       296

Accuracy score : 0.9121621621621622
************************************************************************
performance matrix of BaggingClassifier() is:
Confusion matrix : [[129  13]
 [ 15 139]]
Classification report :               precision    recall  f1-score   support

           0       0.90      0.91      0.90       142
           1       0.91      0.90      0.91       154

    accuracy                           0.91       296
   macro avg       0.91      0.91      0.91       296
weighted avg       0.91      0.91      0.91       296

Accuracy score : 0.9054054054054054
************************************************************************
```

Above are all the matrices of all classification models we can see the Accuracy scores of each model and the precision score.

Cross-validation

What is cross-validation?

Cross-validation is a technique to evaluate the accuracy of a model. This technique splits the input data into different subsets, trains models with each one, and evaluates them on complementary subsets. Use cross-validation to identify overfitting.

Below is the cross-validation Matrix

```
from sklearn.model_selection import cross_val_score
j=[LOGR,RFC,DTC,SV,KNN,EXT,Gbc,BG]
for n in j:
  print('Cross_Validation_score of',n,'is')
  score=cross_val_score(n,x,y,cv=5)
  print(score)
  print(score.mean())
  print(score.std())
  print('\n')
```

```
Cross_Validation_score of LogisticRegression(solver='liblinear') is
[0.68581081 0.75675676 0.83445946 0.80743243 0.85810811]
0.7885135135135135
0.06143077369016596


Cross_Validation_score of RandomForestClassifier() is
[0.73986486 0.81756757 0.91891892 0.91891892 0.94256757]
0.8675675675675676
0.07708035138580356


Cross_Validation_score of DecisionTreeClassifier() is
[0.71621622 0.80067568 0.85810811 0.875      0.84459459]
0.818918918918919
0.056957495718128644



Cross_Validation_score of SVC() is
[0.69256757 0.76013514 0.89864865 0.9222973  0.9222973 ]
0.8391891891891892
0.09497029638277892


Cross_Validation_score of KNeighborsClassifier() is
[0.65540541 0.65202703 0.69594595 0.69932432 0.69256757]
0.6790540540540541
0.020825722983003322


Cross_Validation_score of ExtraTreesClassifier() is
[0.73310811 0.80743243 0.93581081 0.94256757 0.93581081]
0.870945945945946
0.08553103999160357


Cross_Validation_score of GradientBoostingClassifier() is
[0.72635135 0.83108108 0.90540541 0.92567568 0.9222973 ]
0.8621621621621621
0.07606679814085134


Cross_Validation_score of BaggingClassifier() is
[0.78716216 0.84797297 0.89189189 0.90540541 0.90540541]
0.8675675675675676
0.0453961541411914
```

Now it is time to select our best model so that we can hypertunne parameters of the best model if required. As per theory best model is that model where we have less difference between the accuracy score and cross-validation score.

Let's check and find which model has very less difference between the accuracy score and cross-validation score.

In [88]:
```
models=pd.DataFrame({})
models["Models"]=Models#
models["Accuracy"]=Accuracy
models["Cross_va"]=Cross_va
models["Difference"]=score_diff
models
```

Out[88]:

	Models	Accuracy	Cross_va	Differance
0	LOGR	82	78	4
1	RFC	88	86	2
2	DTC	85	81	4
3	SV	86	83	3
4	KNN	65	67	-2
5	EXT	89	87	2
6	Gbc	91	86	5
7	BG	90	86	4

Looking at the above table we can see that Extratreesclassifier is working well as it has a high accuracy score with a minimum difference between the accuracy score and cross-validation score

Hyperparameter tuning for the best model which is Extratreesclassifier

The parameters of a machine learning model are the hyperparameters. They are the knobs that we can turn to adjust the behavior of a model. Hyperparameters are not just tunable, but they also have an impact on the predictive performance of models.

Hyperparameter tuning is a process in which we search for the best set of hyperparameters to use on a given machine-learning task. It is often done by using grid search or random search methods.

Tunning Hyperparameters using GridsearchCV

```
from sklearn.model_selection import GridSearchCV

EXT.get_params()
```

```
{'bootstrap': False,
 'ccp_alpha': 0.0,
 'class_weight': None,
 'criterion': 'gini',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': None,
 'max_samples': None,
 'min_impurity_decrease': 0.0,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'min_weight_fraction_leaf': 0.0,
 'n_estimators': 100,
 'n_jobs': None,
 'oob_score': False,
 'random_state': None,
 'verbose': 0,
 'warm_start': False}
```

```
parameters= {'n_estimators':[10,50,100],
             'criterion':['gini', 'entropy'],
             'max_depth':[1, 2,34,None],
            'min_samples_split':[2, 8, 34],
            'min_samples_leaf':[1, 2,34],
            'max_features': ['auto', 'sqrt', 'log2',],
            'max_leaf_nodes':[2, 5, 8,55]}
```

```
GVC=GridSearchCV(ExtraTreesClassifier(),parameters,cv=3)
```

```
GVC.fit(x_train,y_train)
```

```
GridSearchCV(cv=3, estimator=ExtraTreesClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 2, 34, None],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'max_leaf_nodes': [2, 5, 8, 55],
                         'min_samples_leaf': [1, 2, 34],
                         'min_samples_split': [2, 8, 34],
                         'n_estimators': [10, 50, 100]})
```

```
GVC.best_params_
```

```
{'criterion': 'entropy',
 'max_depth': None,
 'max_features': 'sqrt',
 'max_leaf_nodes': 55,
 'min_samples_leaf': 1,
 'min_samples_split': 2,
 'n_estimators': 100}
```

```
Final_Model = ExtraTreesClassifier(criterion='entropy',
 max_depth=None,
 max_features='sqrt',
 max_leaf_nodes=300,
 min_samples_leaf=1,
 min_samples_split=2,
 n_estimators=100)
```

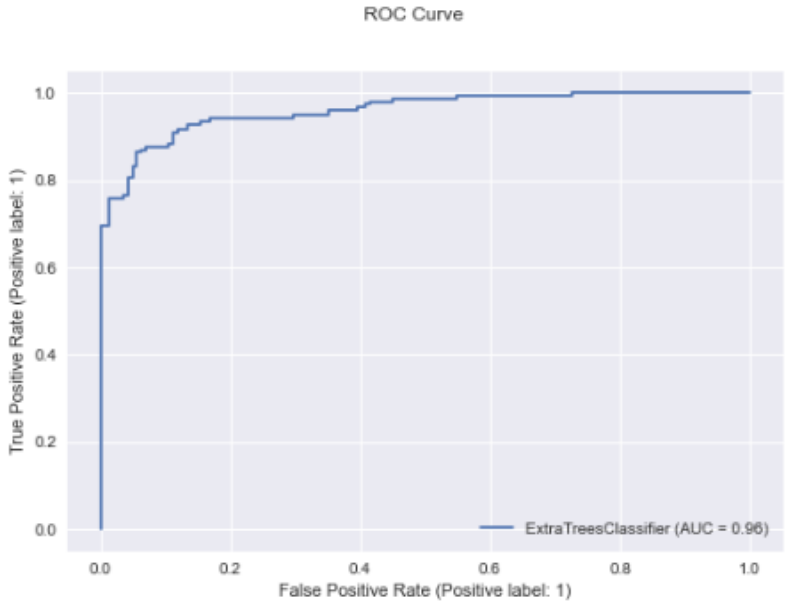
I have found our best parameters and we tried to predict using the best parameters. I got 89 % accuracy after this

```
Classifier = Final_Model.fit(x_train, y_train)
fmod_pred = Final_Model.predict(x_test)
fmod_acc = (accuracy_score(y_test, fmod_pred))*100
print("Accuracy score for the Best Model is:", fmod_acc)
```

```
Accuracy score for the Best Model is: 89.1891891891892
```

11…

```
from sklearn import metrics
plt.style.use('seaborn')
disp = metrics.plot_roc_curve(Final_Model, x_test, y_test)
disp.figure_.suptitle("ROC Curve")
plt.show()
```



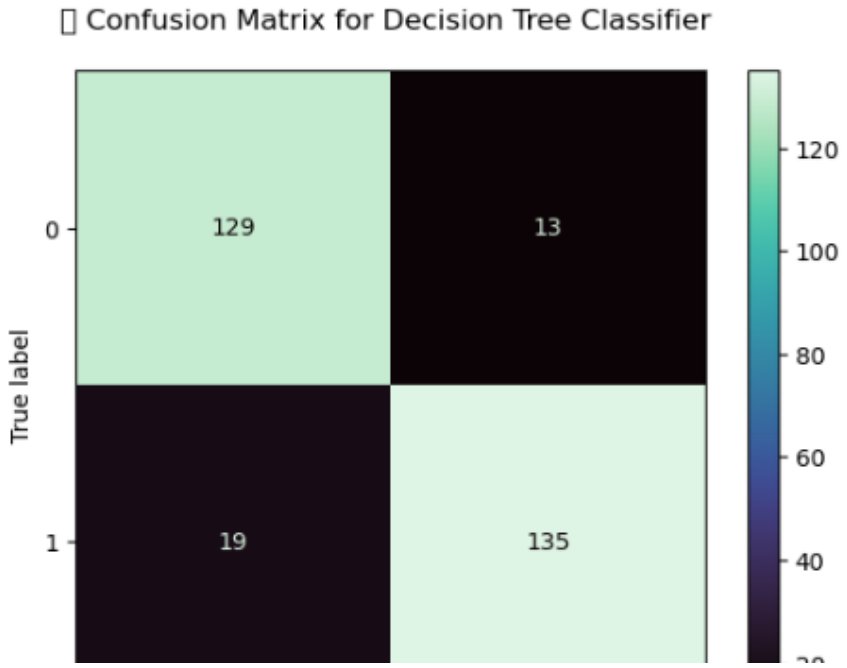
As we can see ROC curve is giving us 96% of accuracy which is good

What is a confusion matrix?

A confusing matrix is a matrix that is used in machine learning to create a confusion matrix. The confusion matrix is a table that compares the predictions of the model with the actual outcomes. It gives information about how accurate the model is.

Confusion Matrix

```
n [112…
plt.style.use('default')
class_names = df.columns
metrics.plot_confusion_matrix(Classifier, x_test, y_test, cmap='mako')
plt.title('\t Confusion Matrix for Decision Tree Classifier \n')
plt.show()
```



Observation

- 129 True positive
- 13 False Positive
- 19 False negative

- 135 True Negative

Saving the final model and using the same model to predict Fraud status.

Saving Final model

```
# Saving the model using .pkl
import joblib
joblib.dump(Final_Model ,"Insur_fraud")
```

```
['Insur_fraud']
```

Lets use model to predict again

```
# Loading the saved model
model=joblib.load("Insur_fraud")

#Prediction
prediction = model.predict(x_test)
prediction
```

```
array([1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1,
       0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0,
       1, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0,
       1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0,
       0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
       0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0,
```

```
In [116…
a=np.array(y_test)
result=pd.DataFrame({"Original":a,"Predicted":prediction})
result
```

Out[116…

	Original	Predicted
0	1	1
1	0	1
2	0	1
3	0	0
4	0	0
...	...	...
291	1	1
292	0	1
293	1	0
294	0	1
295	1	1

296 rows × 2 columns

Above we can see the comparison between the actual value and predicted values by over-model. We can see that our 89 % prediction is correct.

Conclusion

- So at the start we tried to understand the data first with some simple codes and we collected basic knowledge about the dataset
- Then we handled null values and also removed unnecessary columns from the data set
- Handled date formats present in the dataset
- We did deep visualization to understand data and insights.
- We tried to most correlated and less correlated values
- We handles our outlier and skewness of data which were very important steps in building this model
- As our data was imbalanced we used the SMOT technique to oversample data and make balance in data
- We scaled our data using standard scaler

- Finally we trained model and selected best performing model. We also tuned hyperparameters to increase accuracy
- Finally we saved our best model using joblib

If you enjoyed this post, I'd be very grateful if you'd help it spread by emailing it to a friend, or sharing it on Twitter or Facebook.

Welcome any comments or suggestions

Thank you!