**Insurance Claims- Fraud Detection**

**Problem**

**Business case:**  
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 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.

The quality of the wine is depended on the following 40 features.

FEATURES:

* months\_as\_customer
* age
* policy\_number
* policy\_bind\_date
* policy\_state
* policy\_csl
* policy\_deductable
* policy\_annual\_premium
* umbrella\_limit
* insured\_zip
* insured\_sex
* insured\_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

**Data Analysis and EDA**

There are 39 features that determine the insurance claims is farud or not fraud our dataset. Feature variables contains the categorical and continuous values The target variables have two classification Yes or No, So we will be using classification ML algorithms to make the prediction. We need to import necessary python libraries like NumPy, Pandas, Seaborn, Matplotlib etc to perform data analysis. First, we have to read the data using Pandas.

#importing the libraries

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

import sklearn

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, mean\_absolute\_error

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import accuracy\_score

from sklearn.metrics import confusion\_matrix,classification\_report

from scipy.stats import zscore

import warnings

warnings.filterwarnings('ignore')

#loading the data set

df= pd.read\_csv('Automobile\_insurance\_fraud.csv')

df.head()

Table

Description automatically generated

There are 1000 records in the dataset. 40 attributes in the dataset, \_C39 variables contains the all null values so we removing from the

Variables contains the Integer, float and object data type

#Heat map to check the null values

plt.figure(figsize=[8,8])

sns.heatmap(df.isnull())

plt.title('Null values')

plt.show()

A picture containing graphical user interface

Description automatically generated

Dataset contains the no null values

**Cleaning and Pre-processing data:**

“Police\_report\_available” variables contains the symbol need to Remove

**#Replacing ? to none**

df['collision\_type'] = df['collision\_type'].str.replace('?','None')

df['police\_report\_available'] = df['police\_report\_available'].str.replace('?','None')

df['property\_damage'] = df['property\_damage'].str.replace('?','None')

**#sperating day and month, year**

df['Date']=df['incident\_date'].str.split('-').str[0]

df['Month']=df['incident\_date'].str.split('-').str[1]

df['Year']=df['incident\_date'].str.split('-').str[2]

**#converting object to intger type**

df['Date'] = df['Date'].astype(int)

df['Month'] = df['Month'].astype(int)

df['Year'] = df['Year'].astype(int)

#Dropping the column \_c39 which contains the all null values

df.drop(['\_c39','incident\_date'], axis =1, inplace=True)

**Multivariant analysis:**

print(df['witnesses'].value\_counts())

sns.countplot(x='witnesses', hue='fraud\_reported', data=df)

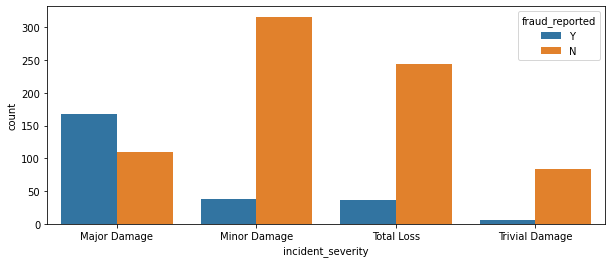
plt.show()



fig=plt.figure(figsize=(10,4))

sns.countplot(x='incident\_severity', hue='fraud\_reported', data=df)

plt.show()

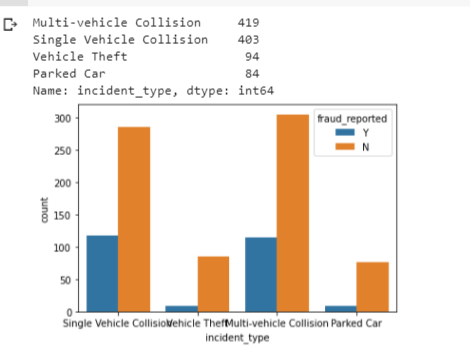


We have observed highest number of insurance fraud happened with major Damage

print(df['incident\_type'].value\_counts())

sns.countplot(x='incident\_type', hue='fraud\_reported', data=df)

plt.show()



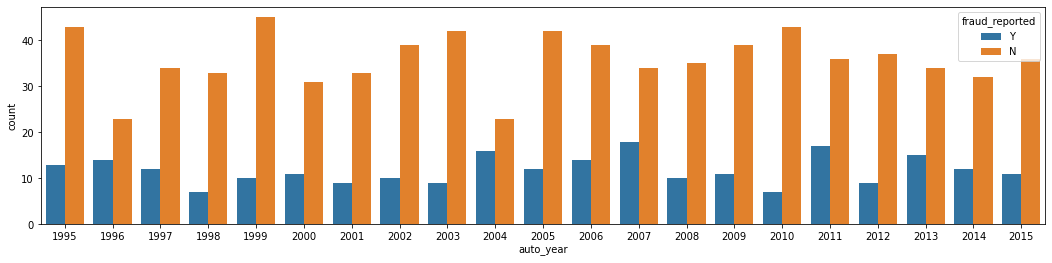
We came to know that highest number incident happened with multi vehicles and fraud reported almost same in both

fig=plt.figure(figsize=(18,4))

print(df['auto\_year'].value\_counts())

sns.countplot(x='auto\_year', hue='fraud\_reported', data=df)

plt.show()



Year 2007 highest number of insurance fraud reported in decade

**Encoding the Categorical variables :**

*We are using the label encoding categorical variable to transfer the numerical*

*#label encoding*

**from** **sklearn.preprocessing** **import** LabelEncoder

le = LabelEncoder()

df['fraud\_reported'] = le.fit\_transform(df['fraud\_reported'])

df['auto\_model'] = le.fit\_transform(df['auto\_model'])

df['auto\_make'] = le.fit\_transform(df['auto\_make'])

df['police\_report\_available'] = le.fit\_transform(df['incident\_city'])

df['property\_damage'] = le.fit\_transform(df['property\_damage'])

df['incident\_location'] = le.fit\_transform(df['incident\_location'])

df['incident\_city'] = le.fit\_transform(df['incident\_city'])

df['incident\_state'] = le.fit\_transform(df['incident\_state'])

df['authorities\_contacted'] = le.fit\_transform(df['authorities\_contacted'])

df['incident\_severity'] = le.fit\_transform(df['incident\_severity'])

df['collision\_type'] = le.fit\_transform(df['collision\_type'])

df['incident\_type'] = le.fit\_transform(df['incident\_type'])

df['insured\_relationship'] = le.fit\_transform(df['insured\_relationship'])

df['insured\_hobbies'] = le.fit\_transform(df['insured\_hobbies'])

df['insured\_occupation'] = le.fit\_transform(df['insured\_occupation'])

df['insured\_education\_level'] = le.fit\_transform(df['insured\_education\_level'])

df['insured\_sex'] = le.fit\_transform(df['insured\_sex'])

df['policy\_csl'] = le.fit\_transform(df['policy\_csl'])

df['policy\_state'] = le.fit\_transform(df['policy\_state'])

**Checking the co relation of the variable using Heat map** I

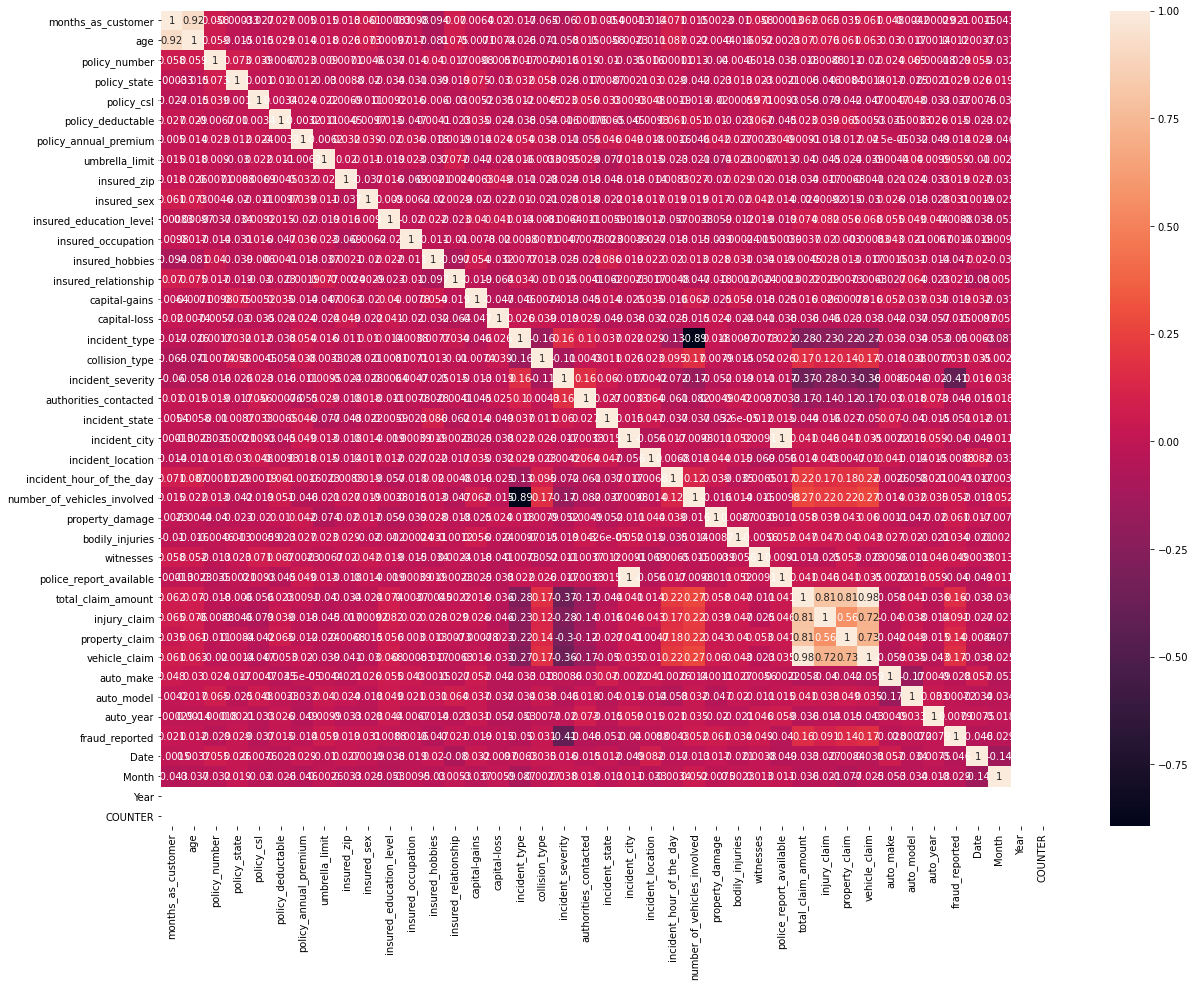
*#To check the corr\_mat Heatmap*

corr\_hmap=df.corr()

plt.figure(figsize=(20,15))

sns.heatmap(corr\_hmap, annot=**True**)

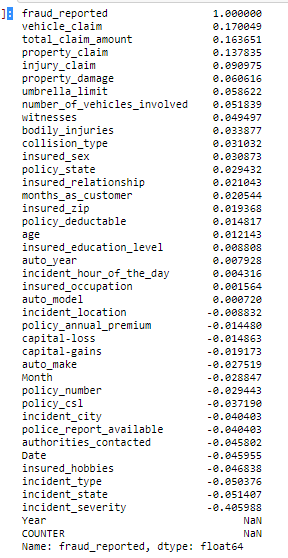
plt.show()



*#to display the co relation with target variable*

corr\_matrix=df.corr()

corr\_matrix['fraud\_reported'].sort\_values(ascending = **False**)



We observed co-relation with dependent variable have highest positive relation with **"vehicle claim total claim”\_** “**amount property claim** "

Also we observed 5 independent variables have almost zero co-relation with the target variables and 16 Variables have negative relation with target variables , Feature variables Year not have any relation with target variable

**Splitting the dataset into X and the Y dataset**

x = df.drop(['fraud\_reported'],axis=1)

y = df['fraud\_reported']

**Splitting the data into training and test set**

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x,y,test\_size=0.25,random\_state=40)

Dependent variable classification data is imbalanced so we are using **Oversampling method** to treat the imbalanced data

#import library

from collections import Counter

#import imblearn

from imblearn import under\_sampling, over\_sampling

from imblearn.over\_sampling import SMOTE

counter1 =Counter(y\_train)

print('Before', counter1)

smote = SMOTE()

x\_smote, y\_smote = smote.fit\_resample(x, y)

counter1 =Counter(y\_smote)

print('after', counter1)



**Model Building**

This problem is an classification so we use multiple classification algorithm which we get highest performance we save the model

#importing the libraries

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import confusion\_matrix,classification\_report,accuracy\_score,f1\_score,precision\_score,roc\_curve,auc

from sklearn.model\_selection import cross\_val\_score

#ML Algo

LR=LogisticRegression()

KNN=KNeighborsClassifier()

SVC=SVC()

DT=DecisionTreeClassifier(random\_state=8)

GNB=GaussianNB()

#Empty List

All\_models= []

Model = []

F1\_score=[]

Accuracy\_score=[]

Precision\_score=[]

CVS = []

rocscore = []

All\_models.append(('LogisticRegression', LR))

All\_models.append(('KNeighborsClassifier', KNN))

All\_models.append(('SVC', SVC))

All\_models.append(('DecisionTreeClassifier', DT))

All\_models.append(('GaussianNB', GNB))

We are passing the all models in for loop and pending the accuracy and ROC\_AUC score, precision score to List

for name, model in All\_models:

    Model.append(name)

    ml=model

    ml.fit(x\_train,y\_train) #unbalanced data

    pred=ml.predict(x\_test)

    F1\_score.append(f1\_score(y\_test,pred))

    Accuracy\_score.append(accuracy\_score(y\_test,pred)\*100)

    Precision\_score.append(precision\_score(y\_test,pred)\*100)

    sc = cross\_val\_score(model, x, y, cv=10, scoring='accuracy').mean()

    CVS.append(sc\*100)

    false\_positive\_rate, true\_positive\_rate, thresholds = roc\_curve(y\_test,pred)

    roc\_auc = auc(false\_positive\_rate, true\_positive\_rate)

    rocscore.append(roc\_auc\*100)

**Creating the data frame**

AccuracyTable=pd.DataFrame({'Model':Model,

                        'F1\_score':F1\_score,

                        'Accuracy\_score': Accuracy\_score,

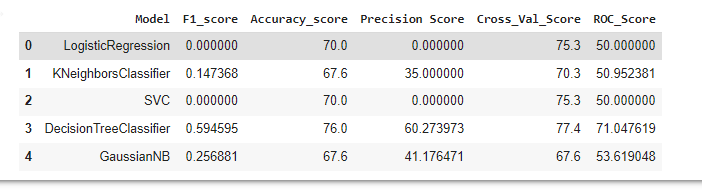
                        'Precision Score': Precision\_score,

                        'Cross\_Val\_Score': CVS,

                        'ROC\_Score':rocscore

                       })

AccuracyTable



**Saving the model:**

We Getting the highest accuracy score in decision tree algorithm so we are saving the model

We are using the pickle method store the model

#importing the library

from sklearn.externals import joblib

# Save the model as a pickle in a file

joblib.dump(DT, 'DT.pkl')

# Load the model from the file

DT\_joblib = joblib.load('DT.pkl')

# Use the loaded model to make predictions

#DT\_joblib.predict(x\_test)