**Insurance Claim Fraud Detection**

**1. Introduction:**

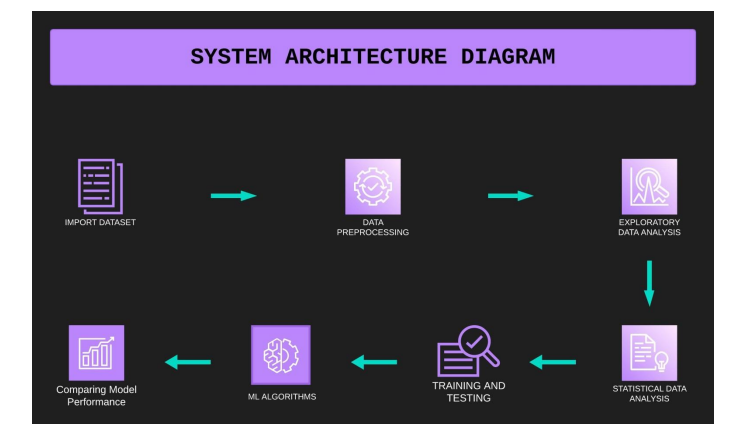
Insurance claim is a huge problem in industry. Especially in Auto Insurance Industry it is very difficult to make out and take decisions about insurance claiming. There is much higher possibility of fraud in claiming value on which the actual incidences happen. To overcome this issue Machine learning is in a unique position to help the Auto Industry.

**1.1 Objective/Problem Defination:**

Creating a predictive model that predicts if an insurance claim is Fraudulent or Not.

**2. System Architecture:**

**2.1 Proposed Block Diagram/Architecture:**



**2.2 The methodology for creating the predictive model of insurance fraud Prediction is as follows:**

* Load the Dataset: Insurance Claim Fraud Detection dataset is loaded using the given link in pd.read\_csv() function. The head() and info() methods are used to display the first few rows and get information about the dataset, respectively.
* Knowing the Dataset: Basic Information about the dataset is generated; numerical and categorical attributes are enlisted.
* The dataset contains the details of the insurance policy along with the customer details (1000Rows and 40 Columns). It also has the details of the accident on the basis of which the claims have been made.
* The dataset contains both categorical and numerical columns. Here "fraud\_reported"is our target column, since it has two categories so it termed to be "Classification Problem “where we need predict if an insurance claim is fraudulent or not.
* Data Cleaning: Any missing values in the dataset are dropped using the dropna() method.
* Data Visualization: Matplotlib and Seaborn libraries are used to visualize the data.
* Splitting the Dataset: The dataset is split into training and testing sets using the train\_test\_split() method from scikit-learn.
* Implementing Machine Learning Algorithms: Logistic Regression, XGBoost, CatBoost, AdaBoost, LightGBM, Decision Tree, and Random Forest classifiers are initialized and trained using the training data.
* Model Evaluation: The accuracy score and confusion matrix are computed to evaluate the performance of each algorithm on the testing data.
* Results: The results, including the accuracy and confusion matrix, are printed for each algorithm.
* Model Performance Comparison: The hvPlot library is used to visualize the ROC curve diagram comparing the performance of all models used.

**2.3 TECHNOLOGY USED**

Technology that would be used in this project is: Python Programming, Machine Learning, Data Analytics, and Statistical Analytics.

**2.4 Importing useful Libraries:**

In order to implement the model, we have used different python libraries that are listed here.

**import** pandas **as** pd

**import** numpy **as** np

**import** seaborn **as** sns

**import** matplotlib.pyplot **as** plt

**import** warnings

warnings**.**filterwarnings('ignore')

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

**from** sklearn.preprocessing **import** StandardScaler

**from** sklearn.decomposition **import** PCA

*# Import Libraries to Build the Model*

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

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classification\_report

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.svm **import** SVC

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.ensemble **import** AdaBoostClassifier

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

**from** sklearn.metrics **import** roc\_curve,roc\_auc\_score

**3. Implementation**

**3.1 Data Exploration and Processing: Compute Size:** In first step, we try to understand the dataset's size and structure at a glance by computing its size.

*df.shape*

*Out[4]:*

*(1000, 40)*

Data Information:*#View summary of dataset*

df**.**info()

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1000 entries, 0 to 999

Data columns (total 40 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 months\_as\_customer 1000 non-null int64

1 age 1000 non-null int64

2 policy\_number 1000 non-null int64

3 policy\_bind\_date 1000 non-null object

4 policy\_state 1000 non-null object

5 policy\_csl 1000 non-null object

6 policy\_deductable 1000 non-null int64

7 policy\_annual\_premium 1000 non-null float64

8 umbrella\_limit 1000 non-null int64

9 insured\_zip 1000 non-null int64

10 insured\_sex 1000 non-null object

11 insured\_education\_level 1000 non-null object

12 insured\_occupation 1000 non-null object

13 insured\_hobbies 1000 non-null object

14 insured\_relationship 1000 non-null object

15 capital-gains 1000 non-null int64

16 capital-loss 1000 non-null int64

17 incident\_date 1000 non-null object

18 incident\_type 1000 non-null object

19 collision\_type 1000 non-null object

20 incident\_severity 1000 non-null object

21 authorities\_contacted 1000 non-null object

22 incident\_state 1000 non-null object

23 incident\_city 1000 non-null object

24 incident\_location 1000 non-null object

25 incident\_hour\_of\_the\_day 1000 non-null int64

26 number\_of\_vehicles\_involved 1000 non-null int64

27 property\_damage 1000 non-null object

28 bodily\_injuries 1000 non-null int64

29 witnesses 1000 non-null int64

30 police\_report\_available 1000 non-null object

31 total\_claim\_amount 1000 non-null int64

32 injury\_claim 1000 non-null int64

33 property\_claim 1000 non-null int64

34 vehicle\_claim 1000 non-null int64

35 auto\_make 1000 non-null object

36 auto\_model 1000 non-null object

37 auto\_year 1000 non-null int64

38 fraud\_reported 1000 non-null object

39 \_c39 0 non-null float64

dtypes: float64(2), int64(17), object(21)

memory usage: 312.6+ KB

We can see that the dataset contains mixture of categorical and numerical variables. Categorical variables have data type object. Numerical variables have data type float64.21 features are categorical and 19 are numerical. Our target variable is categorical.

**3.2 Data Cleaning:**

It is important that the data set is free of defects that could prevent testing or, more seriously, lead to insufficient analysis. These deficiencies or problems caused by redundant records, missing values, or loss of dimension must be effectively resolved. So, in this step bad data will be removed, and missing data will be added. The information we currently have is a comprehensive general information from which we need to remove unnecessary information and perhaps add the missing information.

*#Data Imputation*

*# No. of Missing Values*

df**.**isnull()**.**sum()

Out[13]:

months\_as\_customer 0

age 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\_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

\_c39 1000

dtype: int64

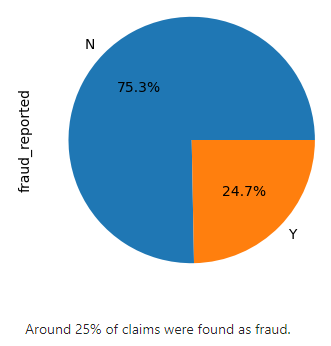
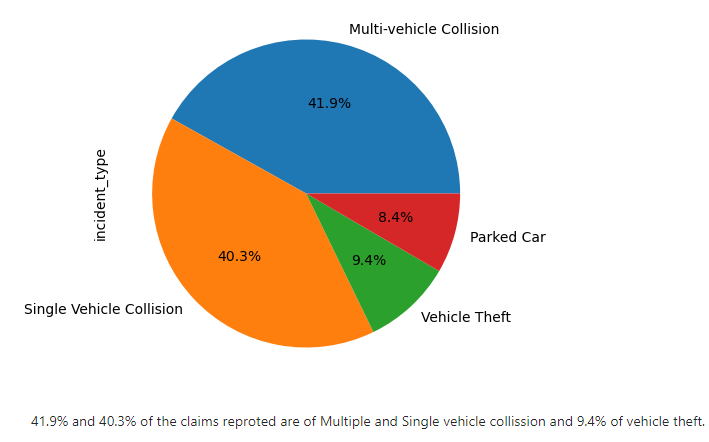
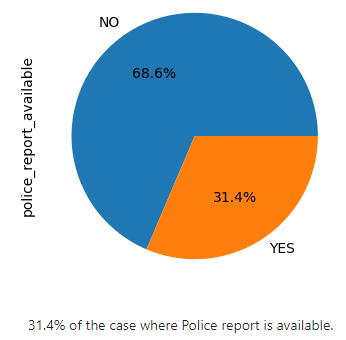
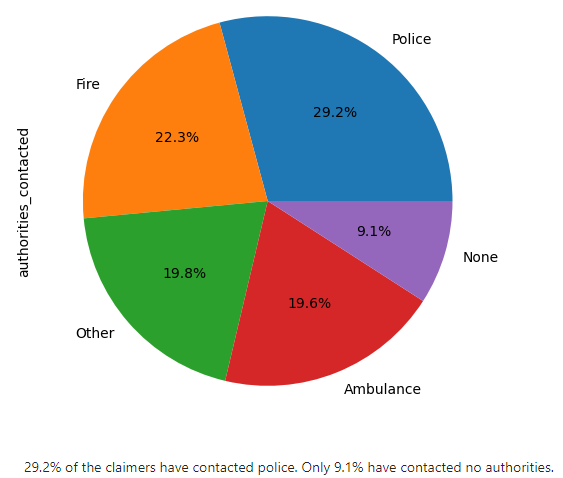
As we can see, there are few missing values in the data set, so handling null values is a very important step in the data cleaning process.

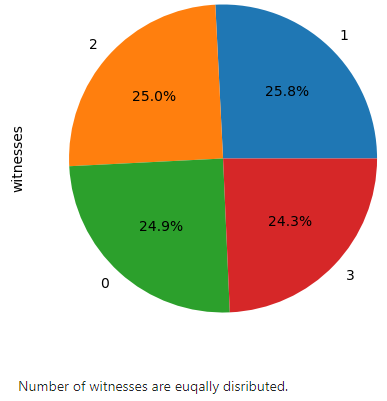
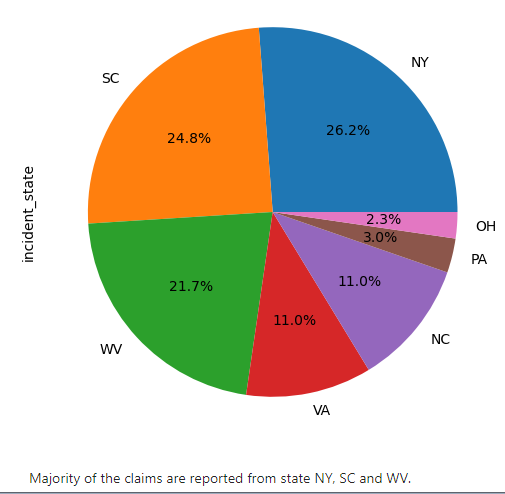
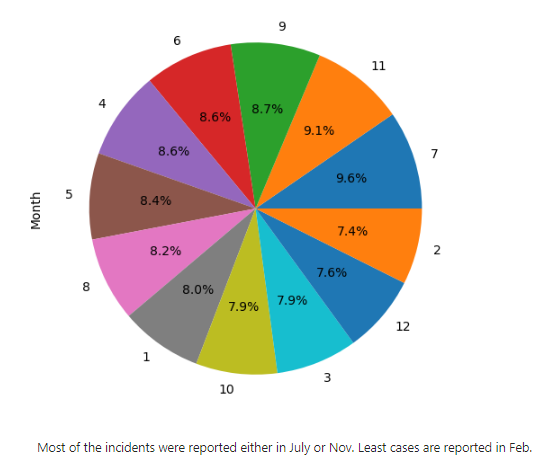
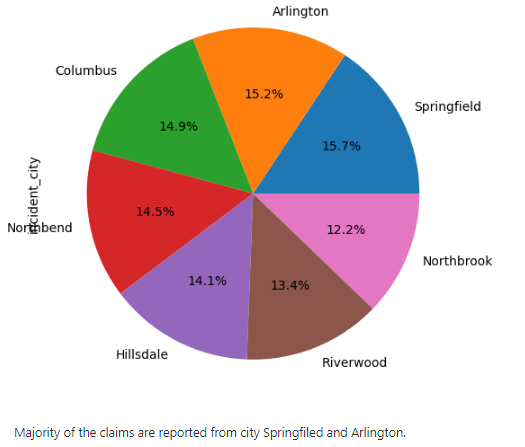
Because it helps to deal with the problems that appear during the subsequent procedures. There are several ways to handle null values and missing values. We can remove all records from the data set or impute missing values using mean, median, mode or regression methods.

We have below Observation after Data Cleaning:

* Data is from year 1990 to 2015
* Minimum age of the customer is 19 and max is 64
* Customer who has claimed insurance is maximum 479 months
* Minimum claim amount given is 100 and maximum is 114920
* We have minimum of 1 witness and max 3.
* Vehicle claim is minimum 70 and maximum is 79560

**4. Data Visualization:**

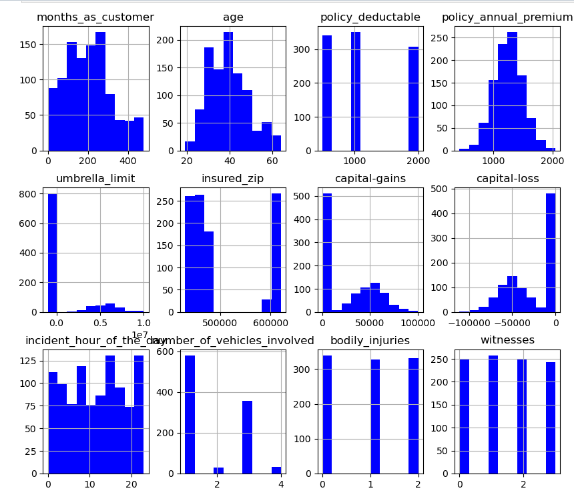
  

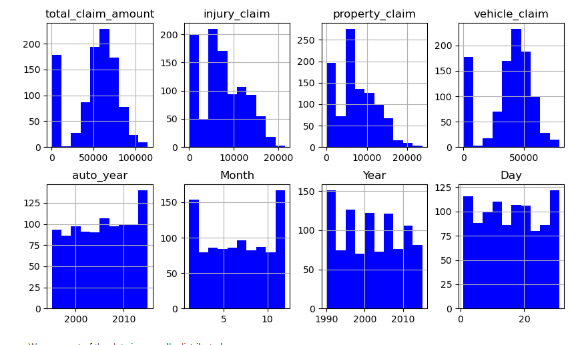
  

Above are few graphical data visualization figures. By doing by Bivariate Analysis and Heat map, overall following conclusions can be made:

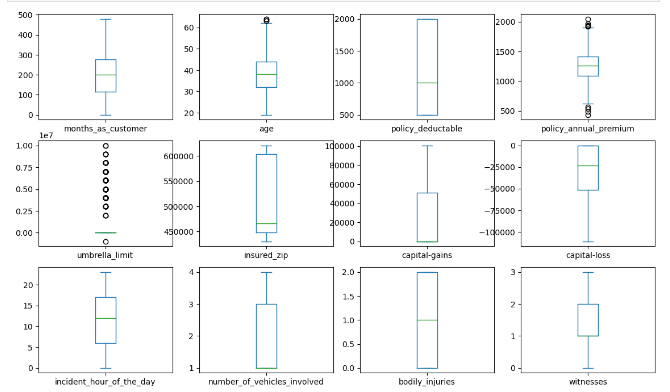
* There were 247 frauds and 753 non-frauds. 24.7% of the data were frauds while 75.3% were non-fraudulent claims.
* Total\_claim\_amount, injury\_claim, property\_claim and Vehicle\_claim are highly correlated with each other
* Around 24% of claims were found as fraud and we only 31.4% of the case where Police report is available.
* Having umbrella unit of 2000000 increases the chance of Insurance fraud.
* Most of the claimers has the reading as hobby while few has basketball and people with Chess and cross fit as hobbies are most likely that their claim is fraud.
* 41.9% and 40.3% of the claims reported are of Multiple and Single vehicle collision and among w29.2% of the claimers have contacted police. Only 9.1% have contacted no authorities. Also Chances of Fraud claim is less if None of the authorities are contacted. It increases if 'Other' is the category of authorities.
* Majority of the claims are reported from state NY, SC and from city Springfield and Arlington.
* Most of the incidents were reported either in July or Nov. Least cases are reported in Feb. However Dec months is more likely to have less fraud claims.

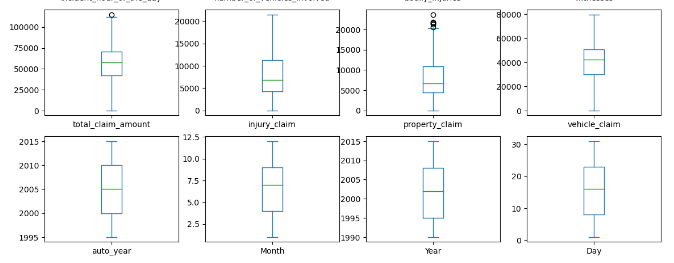
**5. Statistical Analysis’s:**





From above graphs we find most of the data is normally distributed with few outliers.





Umbrella limit and property claim has outliers present.

**When we try to remove these outliner, we find Umbrella limit cannot be removed as we loss most of the data in this process. This cannot be affordable to us.**

# 5.1 Skewness:

df**.**skew()

Out[55]:

months\_as\_customer 0.362177

age 0.478988

policy\_deductable 0.477887

policy\_annual\_premium 0.004402

umbrella\_limit 1.806712

insured\_zip 0.816554

capital-gains 0.478850

capital-loss -0.391472

incident\_hour\_of\_the\_day -0.035584

number\_of\_vehicles\_involved 0.502664

bodily\_injuries 0.014777

witnesses 0.019636

total\_claim\_amount -0.594582

injury\_claim 0.264811

property\_claim 0.368941

vehicle\_claim -0.621098

auto\_year -0.048289

Month -0.029321

Year 0.052511

Day 0.024372

dtype: float64

# We can see that columns umbrella\_limit and insured\_zip has skewness

A variety of methods were used to reduce the skewness, but it did not work. Some were increasing the skew in total claim amount and vehicle claim.

# 6.1 Data Modeling

# Data modeling plays a significant role in Insurance claim prediction model, when integrating machine learning techniques. Machine learning algorithms leverage data models to make predictions, classifications, and recommendations based on patterns and relationships features and labels.

# We use Label Encoder technique and split the features and Labels. With this we observe data is imbalanced.

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

y**=**df['fraud\_reported']

y**=**pd**.**DataFrame(data**=**y)

x**.**shape,y**.**shape

Out[59]:

((1000, 38), (1000, 1))

We will balance the Data using SMOTE(Synthetic Minority Over-sampling Technique)

**from** imblearn.over\_sampling **import** SMOTE

SM**=**SMOTE()

X,Y**=**SM**.**fit\_resample(x,y)

y\_new**=**pd**.**DataFrame(data**=**Y)

x\_new**=**pd**.**DataFrame(data**=**X)

x\_new**.**shape,y\_new**.**shape

Out[60]:

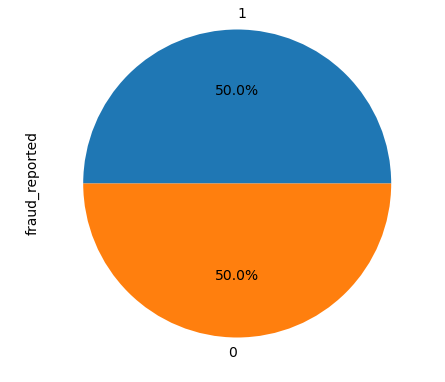
((1506, 38), (1506, 1))

In [61]:

y\_new['fraud\_reported']**.**value\_counts()**.**plot**.**pie(figsize **=** (5,5), autopct **=** '%.1f%%')

Out[61]:

<Axes: ylabel='fraud\_reported'>

****

# 6.2 Data Standardization:

With technique of StandardScaler data standardization done. From co-relation heatmap we have understood that most of the features are co related to each other hence we will use PCA to avoid multi colinearity.

*#PCA*

**from** sklearn.decomposition **import** PCA

testPCA**=**PCA()

Y**=**testPCA**.**fit(sc\_x)

*#Checking the cumulative sum of the expalined variance ratio.*

var\_cumu**=**np**.**cumsum(Y**.**explained\_variance\_ratio\_)**\***100

var\_cumu

Out[63]:

array([ 10.25692742, 15.55381757, 20.05192209, 24.36523611,

27.90211901, 31.23527667, 34.4809619 , 37.69883771,

40.86800758, 43.92571101, 46.94884168, 49.8525298 ,

52.6500886 , 55.38641501, 58.09971239, 60.77365733,

63.4230153 , 65.97865692, 68.44471368, 70.85289575,

73.22903073, 75.5658239 , 77.86847188, 80.09224231,

82.24518142, 84.34191238, 86.38943899, 88.40475553,

90.35896664, 92.29035786, 94.17834173, 96.00994414,

97.67454634, 98.84269024, 99.44607622, 99.81517368,

99.99999399, 100. ])

The variance above shows the number of components in relation to the data we will get. We take 32 components with a variation rate of 95.96%

**After PCA we get new data as**

y**=**y\_new

x**.**shape,y**.**shape

Out[66]:

((1506, 32), (1506, 1))

# 7. 1 Model Building:

*# Import Libraries*

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

**from** sklearn.metrics **import** accuracy\_score, confusion\_matrix, classification\_report

**from** sklearn.linear\_model **import** LogisticRegression

**from** sklearn.svm **import** SVC

**from** sklearn.neighbors **import** KNeighborsClassifier

**from** sklearn.ensemble **import** RandomForestClassifier

**from** sklearn.tree **import** DecisionTreeClassifier

**from** sklearn.naive\_bayes **import** GaussianNB

**from** sklearn.ensemble **import** AdaBoostClassifier

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

**from** sklearn.metrics **import** roc\_curve,roc\_auc\_score

In [68]:

*#check the best possible random state to train our model*

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**=**.20,random\_state**=**i)

mod**=**LogisticRegression()

mod**.**fit(x\_train,y\_train)

pred**=**mod**.**predict(x\_test)

acc**=**accuracy\_score(y\_test,pred)

**if** acc**>**maxAccu:

maxAccu**=**acc

maxRS**=**i

print('Best Accuracy is',maxAccu,' on Random state',maxRS)

Best Accuracy is 0.8211920529801324 on Random state 130

**7.2 Train Test Split:**

It is a function in sklearn model selection for splitting data arrays into two subsets for training data and testing data.

In [69]:

x\_train,x\_test,y\_train,y\_test**=**train\_test\_split(x,y,test\_size**=**.25,random\_state**=**130)

In [70]:

lg**=**LogisticRegression()

sv**=**SVC(probability**=True**)

knn**=**KNeighborsClassifier(n\_neighbors**=**5)

rfc**=**RandomForestClassifier(n\_estimators**=**100)

dtc**=**DecisionTreeClassifier()

gnb**=**GaussianNB()

adc**=**AdaBoostClassifier(n\_estimators**=**100)

model**=**[lg,sv,knn,rfc,dtc,gnb,adc]

Test**=**[]

**for** m **in** model:

m**.**fit(x\_train,y\_train)

pred**=**m**.**predict(x\_test)

acc**=**accuracy\_score(y\_test,pred)

print('Accuracy score of',m)

print(accuracy\_score(y\_test,pred))

Test**.**append(acc)

print(confusion\_matrix(y\_test,pred))

print(classification\_report(y\_test,pred))

print('-------------------------------------------------------------------------------------------')

Accuracy score of LogisticRegression()

0.7824933687002652

[[158 49]

[ 33 137]]

precision recall f1-score support

0 0.83 0.76 0.79 207

1 0.74 0.81 0.77 170

accuracy 0.78 377

macro avg 0.78 0.78 0.78 377

weighted avg 0.79 0.78 0.78 377

-------------------------------------------------------------------------------------------

Accuracy score of SVC(probability=True)

0.8381962864721485

[[176 31]

[ 30 140]]

precision recall f1-score support

0 0.85 0.85 0.85 207

1 0.82 0.82 0.82 170

accuracy 0.84 377

macro avg 0.84 0.84 0.84 377

weighted avg 0.84 0.84 0.84 377

-------------------------------------------------------------------------------------------

Accuracy score of KNeighborsClassifier()

0.6206896551724138

[[ 70 137]

[ 6 164]]

precision recall f1-score support

0 0.92 0.34 0.49 207

1 0.54 0.96 0.70 170

accuracy 0.62 377

macro avg 0.73 0.65 0.60 377

weighted avg 0.75 0.62 0.59 377

-------------------------------------------------------------------------------------------

Accuracy score of RandomForestClassifier()

0.843501326259947

[[173 34]

[ 25 145]]

precision recall f1-score support

0 0.87 0.84 0.85 207

1 0.81 0.85 0.83 170

accuracy 0.84 377

macro avg 0.84 0.84 0.84 377

weighted avg 0.85 0.84 0.84 377

-------------------------------------------------------------------------------------------

Accuracy score of DecisionTreeClassifier()

0.7055702917771883

[[137 70]

[ 41 129]]

precision recall f1-score support

0 0.77 0.66 0.71 207

1 0.65 0.76 0.70 170

accuracy 0.71 377

macro avg 0.71 0.71 0.71 377

weighted avg 0.71 0.71 0.71 377

-------------------------------------------------------------------------------------------

Accuracy score of GaussianNB()

0.8116710875331565

[[167 40]

[ 31 139]]

precision recall f1-score support

0 0.84 0.81 0.82 207

1 0.78 0.82 0.80 170

accuracy 0.81 377

macro avg 0.81 0.81 0.81 377

weighted avg 0.81 0.81 0.81 377

-------------------------------------------------------------------------------------------

Accuracy score of AdaBoostClassifier(n\_estimators=100)

0.7877984084880637

[[157 50]

[ 30 140]]

precision recall f1-score support

0 0.84 0.76 0.80 207

1 0.74 0.82 0.78 170

accuracy 0.79 377

macro avg 0.79 0.79 0.79 377

weighted avg 0.79 0.79 0.79 377

-------------------------------------------------------------------------------------------

**From above we can observe that RandonForest has performed well with 84% accuracy.**

**7.3 Cross Validation:**

In [71]:

*#We will check the Underfitting or Overfitting of the model using Cross Validation:*

cv**=**[]

**for** m **in** model:

score**=**cross\_val\_score(m,x,y,cv**=**5)

cv**.**append(score**.**mean())

print('Mean Accuracy of', m)

print(score**.**mean())

print('--------------------------------------------------------------------------')

Mean Accuracy of LogisticRegression()

0.7378253503773295

--------------------------------------------------------------------------

Mean Accuracy of SVC(probability=True)

0.8102329981738576

--------------------------------------------------------------------------

Mean Accuracy of KNeighborsClassifier()

0.6766627796968163

--------------------------------------------------------------------------

Mean Accuracy of RandomForestClassifier()

0.8155354117621174

--------------------------------------------------------------------------

Mean Accuracy of DecisionTreeClassifier()

0.6959824866339573

--------------------------------------------------------------------------

Mean Accuracy of GaussianNB()

0.7849838287386415

--------------------------------------------------------------------------

Mean Accuracy of AdaBoostClassifier(n\_estimators=100)

0.744472068821368

--------------------------------------------------------------------------

**RandomForestClassifier is the only model with above 80% score on Cross validation and We will check the ROC\_AUC score to pick the final model.**

Performance**=**{'Model':['LogisticRegression','SVC','KNeighborsClassifier','RandomForestClassifier','DecisionTreeClassifier','GaussianNB','AdaBoostClassifier'],

'Test Score':Test,'Cross Validation Score':cv,'ROC AUC Score':auc}

Performance**=**pd**.**DataFrame(data**=**Performance)

Performance

Out[74]:

|  | **Model** | **Test Score** | **Cross Validation Score** | **ROC AUC Score** |
| --- | --- | --- | --- | --- |
| **0** | LogisticRegression | 0.782493 | 0.737825 | 0.784584 |
| **1** | SVC | 0.838196 | 0.810233 | 0.836885 |
| **2** | KNeighborsClassifier | 0.620690 | 0.676663 | 0.651435 |
| **3** | RandomForestClassifier | 0.843501 | 0.815535 | 0.873544 |
| **4** | DecisionTreeClassifier | 0.705570 | 0.695982 | 0.718627 |
| **5** | GaussianNB | 0.811671 | 0.784984 | 0.812205 |
| **6** | AdaBoostClassifier | 0.787798 | 0.744472 | 0.790992 |

**We can conclude that Random Forest Classifier and GaussianNB has performed well with less over fitting and under fitting.**

**7.4 Hyper Parameter Tuning:**

In [75]:

**from** sklearn.model\_selection **import** GridSearchCV

rfc**.**get\_params()

Out[75]:

{'bootstrap': True,

'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}

In [76]:

parameters **=** {'max\_depth': [10, 20, 30, 40, **None**],

'max\_features': ['auto', 'sqrt'],

'min\_samples\_leaf': [1, 2, 4],

'min\_samples\_split': [2, 5, 10],

'n\_estimators': [5, 10, 20, 30, 50,100],

'criterion':['gini', 'entropy']}

In [77]:

GCV**=**GridSearchCV(RandomForestClassifier(),parameters,cv**=**3)

GCV**.**fit(x\_train,y\_train)

GCV**.**best\_params\_

Out[77]:

{'criterion': 'gini',

'max\_depth': None,

'max\_features': 'sqrt',

'min\_samples\_leaf': 1,

'min\_samples\_split': 5,

'n\_estimators': 100}

In [78]:

Finalmod**=**RandomForestClassifier(max\_features**=** 'auto', min\_samples\_leaf**=** 1, min\_samples\_split**=**5,n\_estimators**=**100,max\_depth**=**40,criterion**=**'entropy')

Finalmod**.**fit(x\_train,y\_train)

pred1**=**Finalmod**.**predict(x\_test)

acc1**=**accuracy\_score(y\_test,pred1)

cvs1**=**cross\_val\_score(Finalmod,x,y,cv**=**5)

y\_pred\_prob**=**Finalmod**.**predict\_proba(x\_test)[:,1]

fpr,tpr,thresholds**=**roc\_curve(y\_test,y\_pred\_prob)

auc\_score1**=**roc\_auc\_score(y\_test,Finalmod**.**predict(x\_test))

print('Random Forest classifier Performance after HyperTuning')

print('-----------------------------------------------')

print('Accuracy Score',acc1**\***100)

print('Cross Validation Score',cvs1**.**mean()**\***100)

print('AUC ROC Score',auc\_score1**\***100)

print('\n')

print('AUC ROC Curve with Final Mod')

plt**.**plot([0,1],[0,1],'k--')

plt**.**plot(fpr,tpr,label**=**RandomForestClassifier)

plt**.**xlabel('False Positive rate')

plt**.**ylabel('True Positive rate')

plt**.**title(Finalmod)

plt**.**show()

Random Forest classifier Performance after HyperTuning

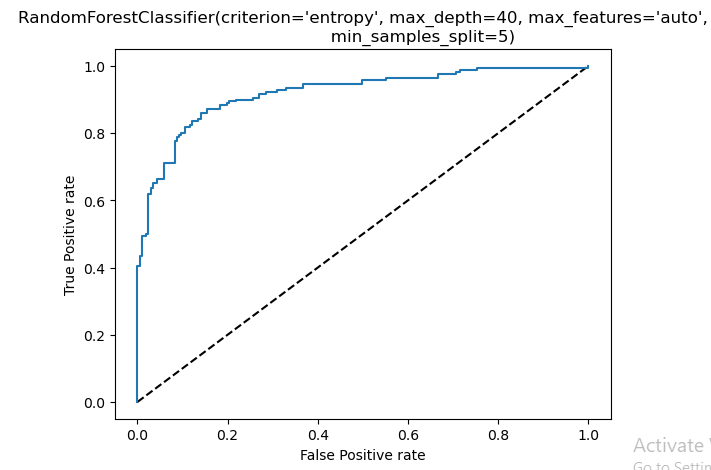
-----------------------------------------------

Accuracy Score 85.41114058355438

Cross Validation Score 80.6910739037645

AUC ROC Score 85.50582551861325

**AUC ROC Curve with Final Mod**



**If we observe the above metrics, We got good values with hyper parameter tuning model compare to model without hyper parameter tuning with Random Forest Classifier, Increased accuracy from 83.1 to 85.41% with the CV score of 80.69% and AUC ROS score of 85.50%**

**8. Conclusion:**

This project has built a model that can detect auto insurance fraud. In doing so, the model can reduce losses for insurance companies. The challenge behind fraud detection in machine learning is that frauds are far less common as compared to legit insurance claims.

Six different classifiers were used in this project: logistic regression, K-nearest neighbors, Random forest, Decision tree, GaussianNB, AdaBoostClassifier,and SVC. Handling imbalance classes were tested out with these model with class weighting, oversampling with SMOTE, hyper parameter tuning, and plotting roc curve of the models.

The best and final fitted model was Random Forest  Classifier that yelled a F1 score of 85.41% and a ROC AUC of 85.50%. The model performed excellent.