### Automobile Insurance Claims- Fraud Detection



Insurance fraud is a critical problem in industry since the beginning of insurance as a commercial enterprise. Every year automobile fraud claims cost large sum to insurance industries. About 90 percent of automobile insurance frauds are resulting from padding (which means to add damages, injuries and fictitious passengers to insurance claims).

It difficult to recognize fraud claims. But **Machine Learning** is in a unique position to solve this issue and help out auto insurance industries from this problem.

**Problem Statement:**

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.

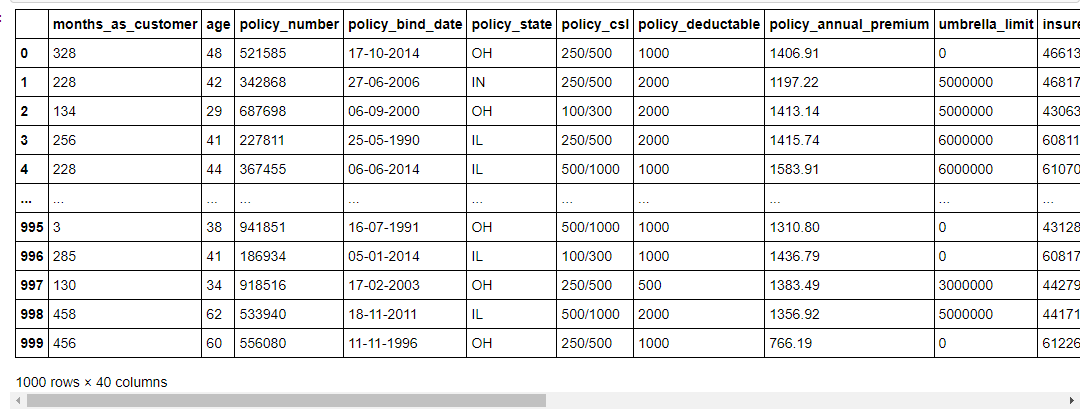
In this example, you will be working with some auto insurance data to demonstrate how you can create a predictive model that predicts if an insurance claim is fraudulent or not.

**Importing Libraries**

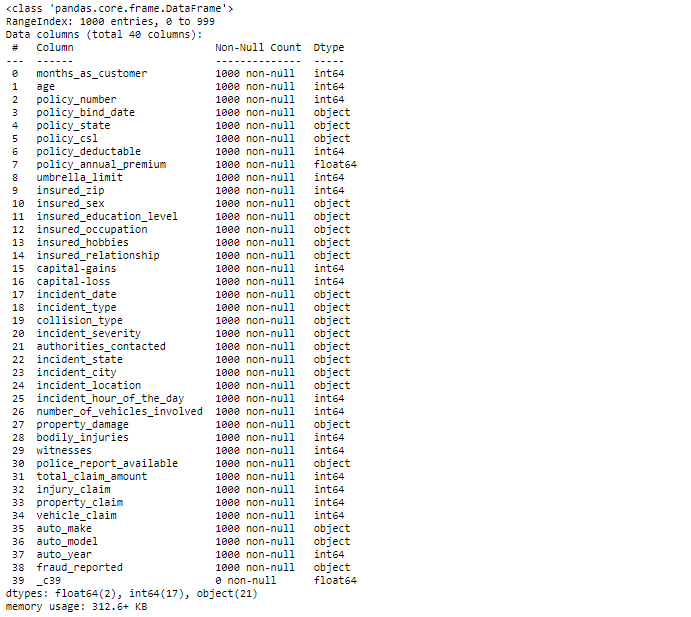
#import Neccessory libraries  
**import** **pandas** **as** **pd**  
**import** **numpy** **as** **np**#data visualization **import** **matplotlib.pyplot** **as** **plt**  
**import** **seaborn** **as** **sns**  
  
**from** **sklearn.preprocessing** **import** StandardScaler  
  
**from** **sklearn.model\_selection** **import** train\_test\_split, GridSearchCV  
  
#import required evaluation metrics  
**from** **sklearn.metrics** **import** accuracy\_score, confusion\_matrix, classification\_report  
**from** **sklearn.metrics** **import** roc\_curve, roc\_auc\_score  
**from** **sklearn.model\_selection** **import** KFold, cross\_val\_score  
  
**import** **warnings**  
warnings.filterwarnings('ignore')  
%matplotlib inline

#### ****Getting Data****

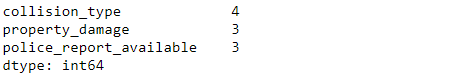
df = pd.read\_csv(r"Automobile\_insurance\_fraud.csv")  
df



**The dataset is having 1000 samples and 38 features + one target variable (fraud\_reported),** we can see dataset is having features with different data types.

As per observations from data info, we can conclude that the data set is not having any null values.

By doing some observation I came to know that some of the columns having entries marked as **“?”.** I will check for columns with this ? mark by running below code.

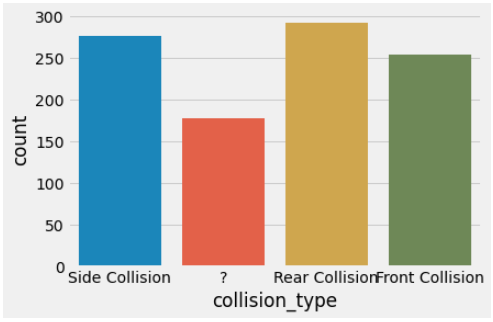
df[df.columns[(df == '?').any()]].nunique()

The above three columns which are ‘collision\_type’, ‘property\_damage’ and ‘police\_report\_available’ are having some entries with **“?”** mark

#### ****Data Analyzing****

1. **collision\_type**

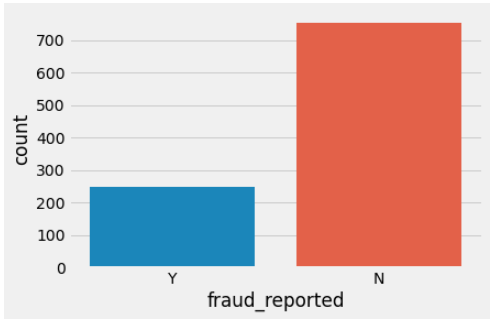
plt.style.use('fivethirtyeight')  
sns.countplot(x = 'collision\_type', data = df)  
plt.show()



We can say that most of the cases are of rear collision. we will treat ‘?’ as a other type of collision.

**2. fraud\_reported (target variable)**

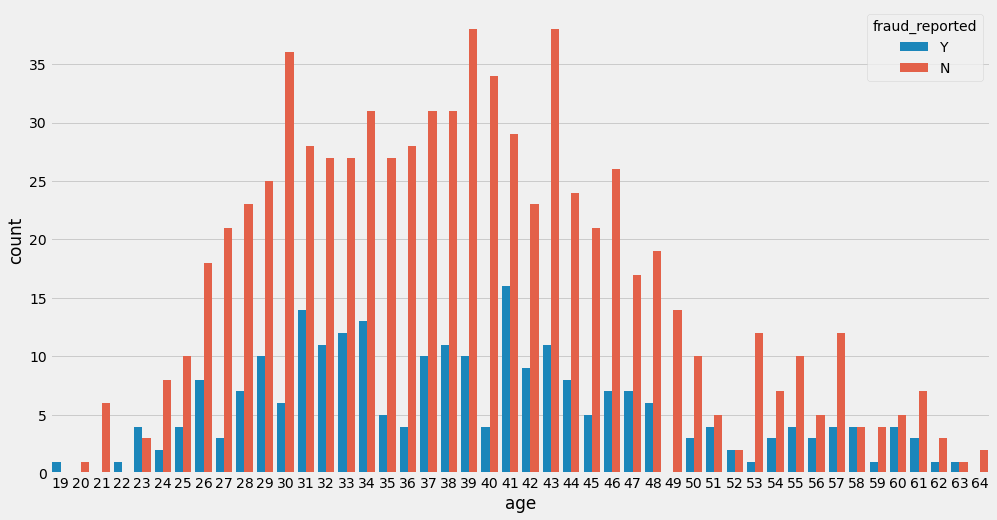
sns.countplot(df['fraud\_reported'])  
print(df['fraud\_reported'].value\_counts())  
plt.show()



Here we see that our target variable is showing the problem of class imbalance, almost 25% customers reported as fraud.

**3. age VS fraud\_reported**

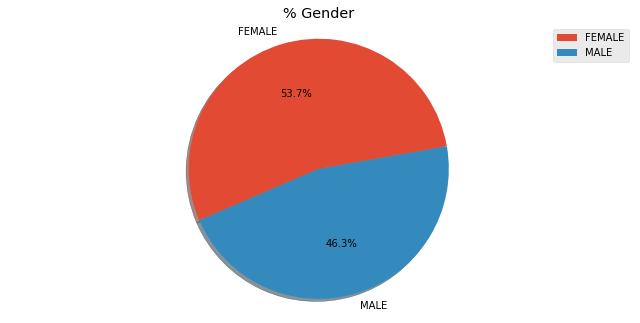
plt.figure(figsize = (15,8))  
sns.countplot(x = 'age', hue = 'fraud\_reported', data = df)  
plt.show()



Looking at the above plot we can see that most number of customers are in the age between 25 to 50 years. And the rate of being reported as a fraud is high for age 19, 22 and 23.

**4. insured\_sex**

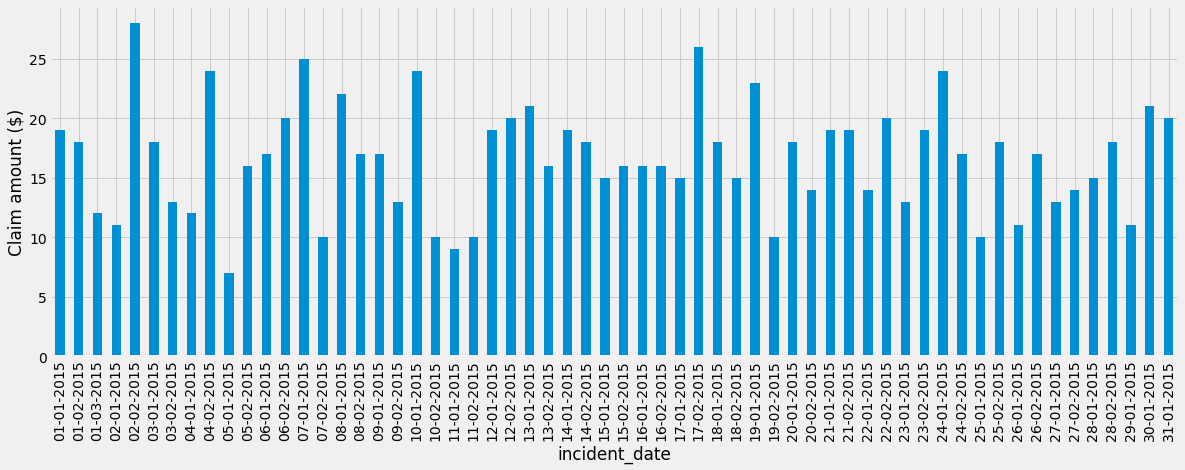
plt.style.use('ggplot')  
gender = df['insured\_sex'].value\_counts()  
plt.figure(figsize=(10, 5))  
plt.pie(gender.values, labels = gender.index, startangle=10, shadow = **True**, autopct='**%1.1f%%**')  
plt.title('**% G**ender ')  
plt.legend()  
plt.legend(prop={'size': 10})  
plt.axis('equal')  
plt.show()



We can say that the number of females is more than that of the males.

**5. Bar plot for incident\_date vs Claim amount($)**

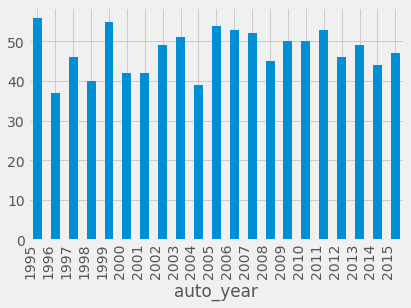
fig = plt.figure(figsize=(18,6))  
ax = df.groupby('incident\_date').total\_claim\_amount.count().plot.bar(ylim=0)  
ax.set\_ylabel('Claim amount ($)')  
plt.show()



Looking at above bar plot we can see that, all the cases are from the months of January and February 2015 and only one case is from march 2015.

**6. auto\_year**

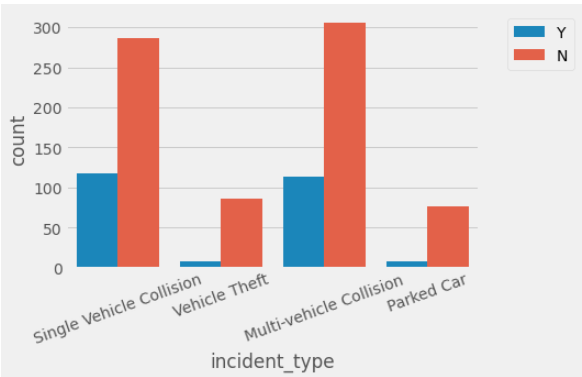
ax = df.groupby('auto\_year').vehicle\_claim.count().plot.bar()  
ax.set\_xticklabels(ax.get\_xticklabels(), rotation=90, ha="right")  
plt.show()



Looking at this plot we can say we are having the vehicles which are made since 1995 to 2015, less number of vehicles are there which are made in the year of 1996 and 2004.

**7. incident\_type**

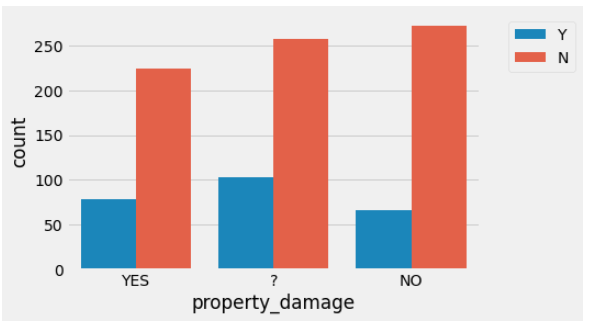
sns.countplot( x = 'incident\_type', hue = 'fraud\_reported', data = df)  
plt.legend( bbox\_to\_anchor = (1.05, 1), loc = 'upper left')  
plt.xticks(rotation = 20)  
plt.show()



By observing this plot we can say that more cases for insurance claims are from single Vehicle collision and Multi-vehicle collision, and rate of getting fraud cases are also more in these categories.

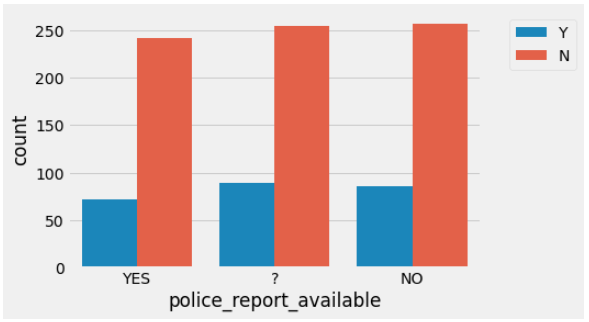
**8. property\_damage**

sns.countplot(x = 'property\_damage',hue = 'fraud\_reported', data = df)  
plt.legend( bbox\_to\_anchor = (1.05, 1), loc = 'upper left')  
plt.show()



**police\_report\_available**

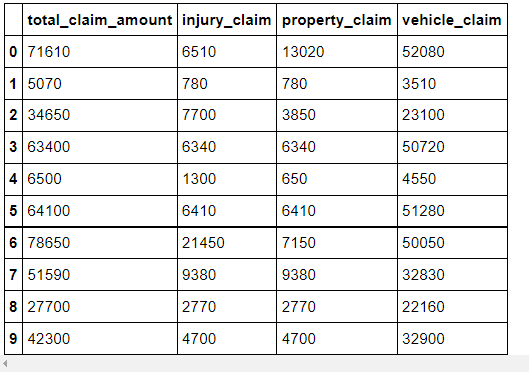
sns.countplot(x = 'police\_report\_available',hue = 'fraud\_reported', data = df)  
plt.legend( bbox\_to\_anchor = (1.05, 1), loc = 'upper left')  
plt.show()



Looking at both the plots for **property\_damage** and **police\_report\_available** we can observe thatthese columns are having entries in the form of **YES** and **NO** and some of entries are **‘?’** mark. I will replace this ? mark with most frequent element of that particular column.

**We will check columns which are for claims**

claim = pd.DataFrame(df, columns = ['total\_claim\_amount','injury\_claim','property\_claim','vehicle\_claim'])  
claim.head(10)



Looking at these all columns I observe that total claim amount is the sum of other claims like injury claims, property claim, vehicle claims.

#### ****Data Processing****

First I will replace ‘?’ from **property\_damage** & **police\_report\_available** with most frequent element of that column

df['property\_damage'].replace('YES', 1, inplace=**True**)  
df['property\_damage'].replace('NO', 0, inplace=**True**)  
df['property\_damage'].replace('?', 0, inplace=**True**)  
df['police\_report\_available'].replace('YES', 1, inplace=**True**)  
df['police\_report\_available'].replace('NO',0, inplace=**True**)  
df['police\_report\_available'].replace('?', 0, inplace=**True**)

Because **policy\_number** is just a identity number for particular policy and it is not contributing to target variable. And also **policy\_bind\_date** is not showing any relation with target variable, I decided to drop these columns.

df.drop(columns = ['policy\_number','policy\_bind\_date'], inplace = **True**)

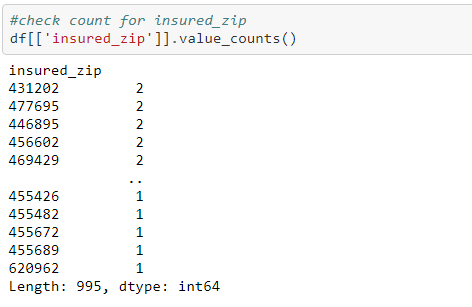
Column **“\_c39”** is having all null values hence I will drop this column as well

df.drop(columns = '\_c39', inplace = **True**)

**insured\_zip**

As insured\_zip represents zip code for particular insurance, I will change its data type to object type.

df[['insured\_zip']] = df[['insured\_zip']].astype(object)

Check for value\_counts

As we can see the value counts for **insured\_zip** column is having 995 unique entries out of 1000 entries. So I decided to drop this column.

df.drop(columns = "insured\_zip", inplace = **True**)

**policy\_csl** column is having data in the form of ratio of two numbers. I will make two different columns from this column as **csl\_per\_person** & **csl\_per\_accident.**

df['csl\_per\_person'] = df.policy\_csl.str.split('/', expand=**True**)[0]  
df['csl\_per\_accident'] = df.policy\_csl.str.split('/', expand=**True**)[1]

**auto\_year**

I will derive new column using **auto\_year** as **vehicle\_age** by assuming the data is collected from the year 2018for better understanding.

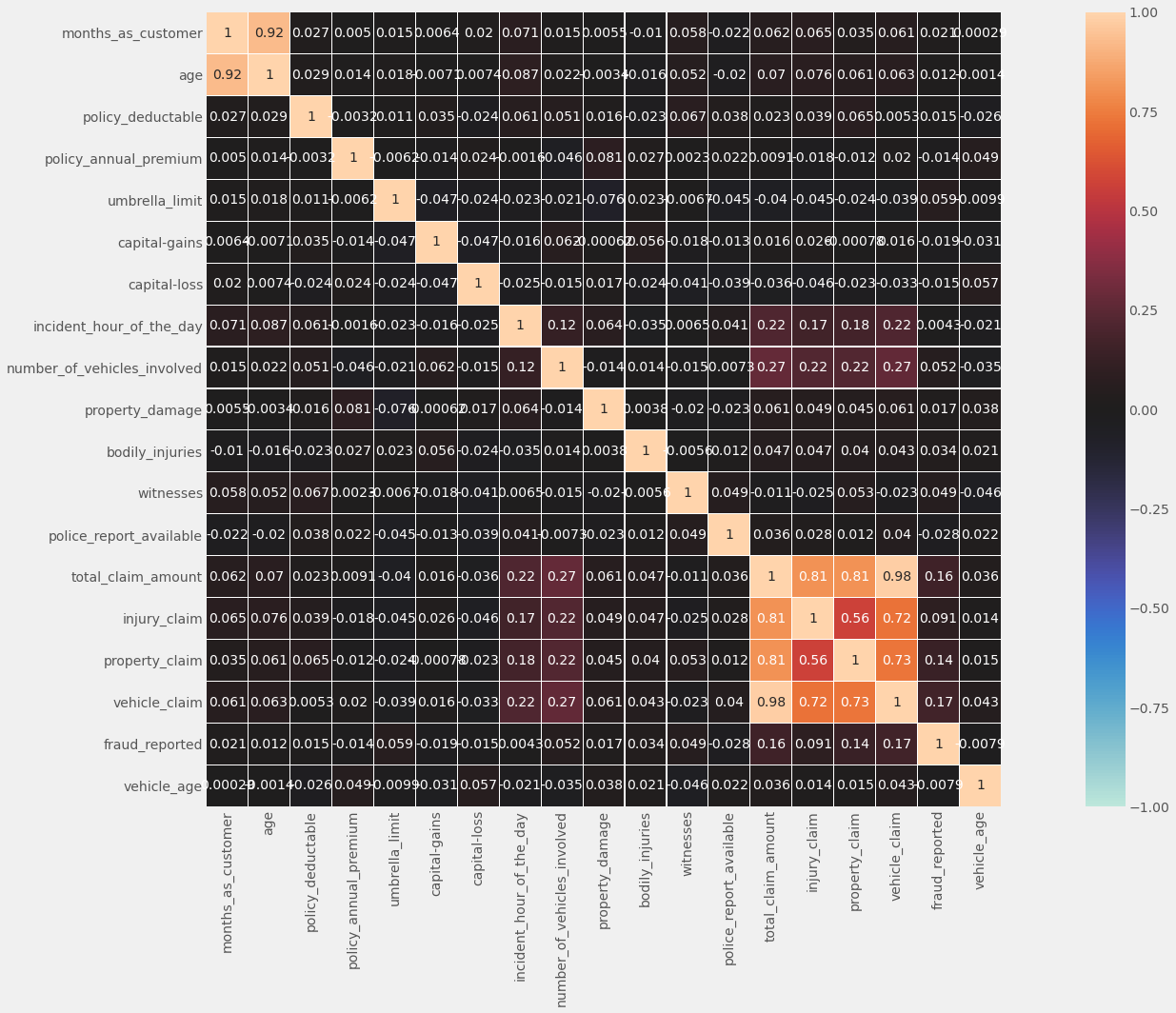
df['vehicle\_age'] = 2018 - df['auto\_year']

The column **incident\_date** contains data only from two months of the year 2015 and for column **policy\_csl** and **auto\_year** I have derived separate columns so I am dropping these columns

df.drop(columns = ["policy\_csl","auto\_year","incident\_date"], inplace = **True**)

Now we have done much of the required data processing successfully.

#### Heat Map for checking correlation

df\_corr = df.corr()  
plt.figure(figsize = (25,15))  
sns.heatmap(df\_corr,vmin=-1,vmax=1,annot=**True**,square=**True**,center=0,fmt='.2g',linewidths=0.1)  
plt.tight\_layout()

* We can see all features are having very less coefficient of correlation with the target variable.
* Total amount claim and vehicle claim having maximum correlation with target variable and that is of 0.16 and 0.17 respectively.
* **age** and **months\_as\_customer** columns are strongly related to each other.
* All four columns representing claim amounts are in good relation with each other.

#### ****Skewness****

df.skew()

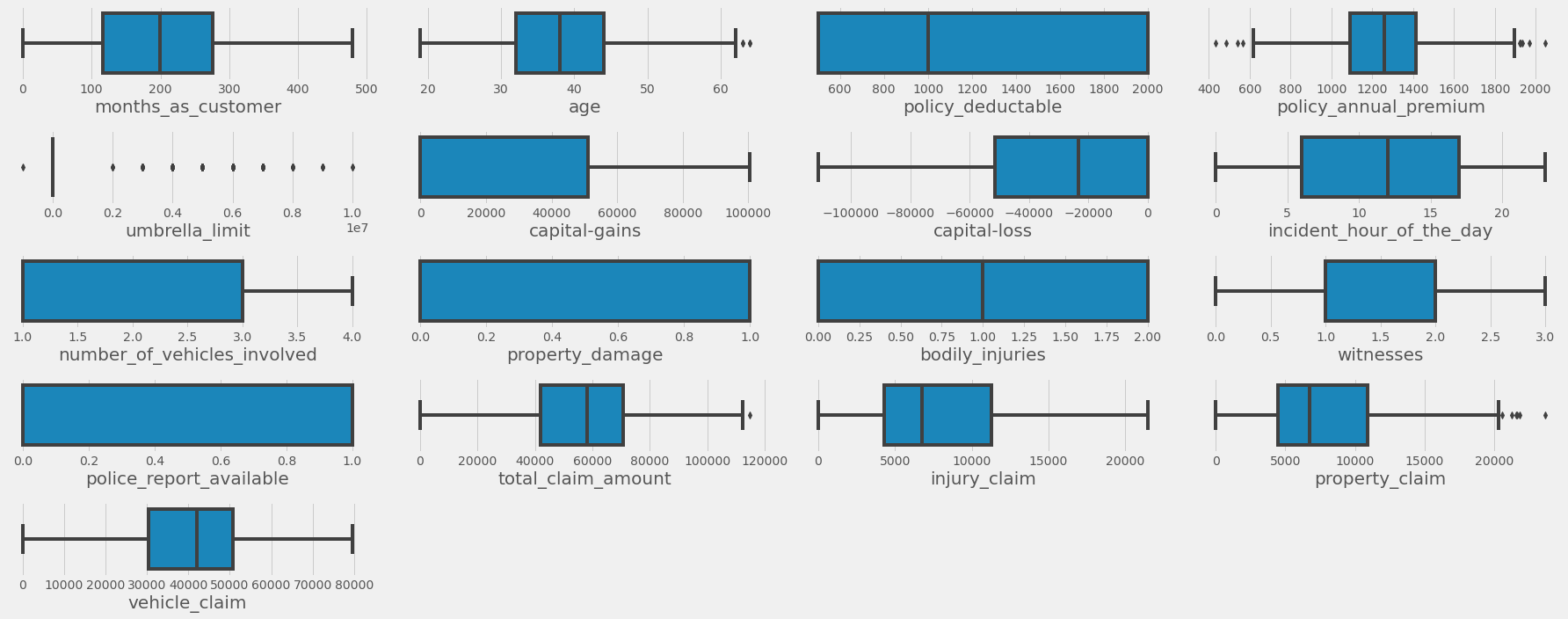


We can see many columns are with skewed data, I will treat this skewness after removing outliers.

#### ****Checking for Outliers****

Below is the code for box plots for numerical columns.

num\_data = df.\_get\_numeric\_data()  
plt.figure(figsize = (25,10))  
plotnumber = 1  
**for** column **in** num\_data:  
 **if** plotnumber <=17:  
 ax = plt.subplot(5,4,plotnumber)  
 sns.boxplot(num\_data[column])  
 plt.xlabel(column,fontsize = 20)  
 plotnumber+=1  
plt.tight\_layout()



Looking at above bar plots I can say that there are some outliers present in age, policy\_annual\_premium, umbrella\_limit, total\_claim\_amount and property\_claim columns. We will remove outliers using z-score method.

**Outliers removing**

**from** **scipy** **import** stats  
**from** **scipy.stats** **import** zscore  
z\_score = zscore(df[["age","policy\_annual\_premium","umbrella\_limit","total\_claim\_amount","property\_claim"]])  
abs\_z\_score = np.abs(z\_score)  
filtering\_entry = (abs\_z\_score < 3).all(axis = 1)  
df = df[filtering\_entry]  
df.reset\_index(inplace = **True**)

df.shape



We can see that after removing outliers from the data set we are left with 981 observations out of 1000. Which means we are loosing about 2% of data which is affordable for better results.

Dropping **index** column

df.drop(columns = 'index', inplace = **True**)

#### Separate features and label as x and y respectively

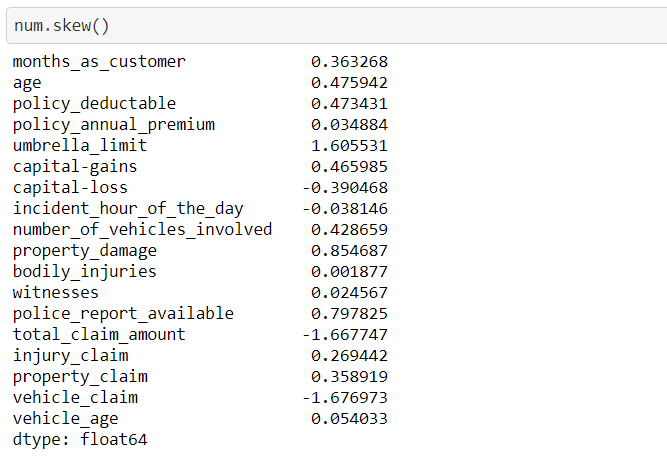
x = df.drop(columns = 'fraud\_reported')  
y = df['fraud\_reported']

Now I have separated our features and label and we know that many columns are with skewed data. I will separate features into numerical and categorical data. I will treat skewness from numerical data and apply Standard Scaler for this data to bring all columns into common scale. and then will combine again both these datasets.

num = x.\_get\_numeric\_data()  
cat = x.select\_dtypes(include=['object'])

I will apply cube root for positively skewed data and use log transformation for negatively skewed data for treating the skewness from numerical features.

**for** index **in** num.skew().index:  
 **if** num.skew().loc[index]>0.5:  
 num[index]=np.cbrt(num[index])  
 **if** num.skew().loc[index]<-0.5:  
 num[index]=np.log1p(num[index])



#### ****Feature Scaling****

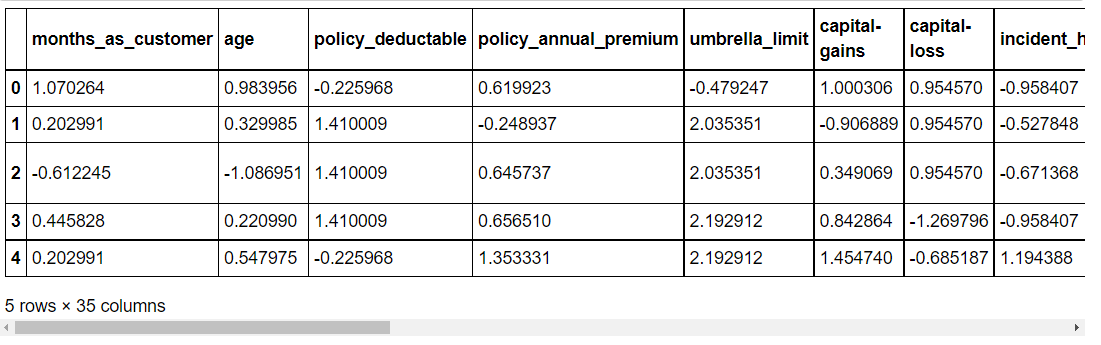
Now we will bring all the numerical features to common scale by applying StandardScaler. StandardScaler will remove the mean and scales each feature to unit variance.

scaler = StandardScaler()  
x\_num = scaler.fit\_transform(num)  
x\_num = pd.DataFrame(x\_num,columns=num.columns)

After scaling the numerical features I will join both numerical and categorical datasets. and we will also see how the data looks after feature scaling.

X = pd.concat([x\_num,cat], axis = 1)

X.head()



#### ****Encoding****

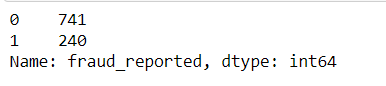
After scaling the numerical data now we need to do encoding for our categorical features. Here I am using Ordinal encoder for this purpose.

**from** **sklearn.preprocessing** **import** OrdinalEncoder  
enc = OrdinalEncoder()  
**for** i **in** X.columns:  
 **if** X[i].dtypes == "object" :  
 X[i] = enc.fit\_transform(X[i].values.reshape(-1,1))

#### Class Imbalance

As our target variable is having binary data so this is a classification problem and we may face class imbalance issue. I will check the count for target variable.

y.value\_counts()

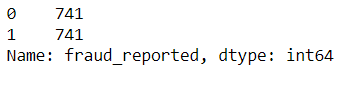


By seeing the count from our target variable we can say there exist the problem of imbalance. To overcome this issue I will go for oversampling the data using SMOTE. It will select random examples from minor class with replacement and will add them to training dataset.

**Over sampling**

**import** **imblearn**  
**from** **imblearn.over\_sampling** **import** SMOTE  
SM = SMOTE()  
x\_over,y\_over = SM.fit\_resample(X,y)

#lets check the target variable now  
y\_over.value\_counts()



Great we have resolved the problem of imbalance.

#### Finding best random state

Random state will **ensure that the splits that we generate will be reproducible.** The random state that we provide is used as a seed to the random number generator. This will ensure that the random numbers are generated in the same order.

I will select best random state that will give us maximum accuracy with LogisticRegression ML model. And with this random state we will split our data for every model.

**from** **sklearn.linear\_model** **import** LogisticRegression  
max\_accu = 0  
max\_rs = 0  
**for** i **in** range(1,1000):  
 x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_over,y\_over,test\_size = 0.25, random\_state = i)  
 LR = LogisticRegression()  
 LR.fit(x\_train,y\_train)  
 pred = LR.predict(x\_test)  
 acc = accuracy\_score(y\_test,pred)  
 **if** acc > max\_accu:  
 max\_accu = acc  
 max\_rs = i  
print("Best accuracy is",max\_accu,"on Random State",max\_rs)



Great we got maximum accuracy of 0.78 with LogisticRegression model for random state 682, now I will use this random state to split our data in to train and test parts.

x\_train,x\_test,y\_train,y\_test = train\_test\_split(x\_over, y\_over, test\_size = 0.25, random\_state = 682 )

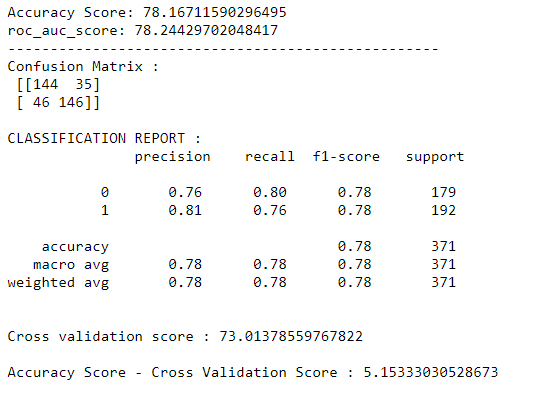
### Model Building with Evaluation Metrics

While building the machine learning model I am using different algorithms to train and test our data after that we will select a best suitable algorithm for final model.

**Evaluation**: I am using many evaluation metrics like cross-validation, confusion matrix ,classification report and AUC and ROC curve for selecting best suitable algorithm.

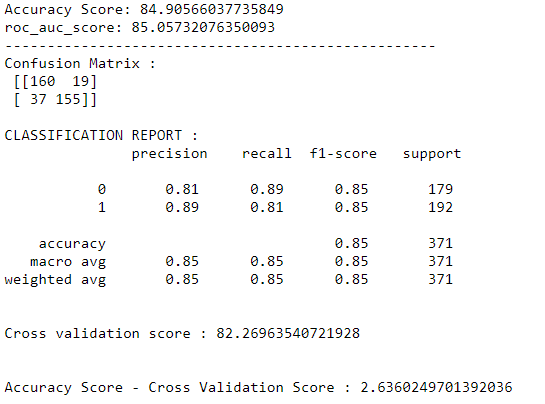
**LogisticRegression**

LR.fit(x\_train,y\_train)  
predlr = LR.predict(x\_test)  
accuracy = accuracy\_score(y\_test,predlr)\*100  
  
print(f"Accuracy Score:", accuracy)  
print(f"roc\_auc\_score: **{**roc\_auc\_score(y\_test,predlr)\*100**}**")  
print("---------------------------------------------------")  
  
#confusion matrix & classification report  
print(f"Confusion Matrix : **\n** **{**confusion\_matrix(y\_test,predlr)**}\n**")  
print(f"CLASSIFICATION REPORT : **\n** **{**classification\_report(y\_test,predlr)**}**")  
  
#cross validation score  
scores = cross\_val\_score(LR, x\_over, y\_over, cv = 10,scoring = "accuracy" ).mean()\*100  
print("**\n**Cross validation score :", scores)  
  
#result of accuracy minus cv score  
result = accuracy - scores  
print("**\n**Accuracy Score - Cross Validation Score :", result)



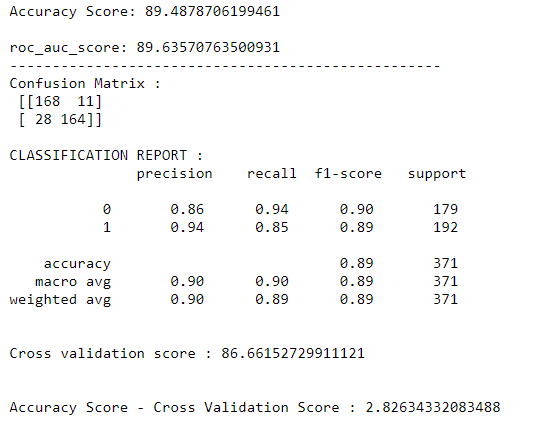
**DecesionTreeClassifier**

**from** **sklearn.tree** **import** DecisionTreeClassifier  
dt = DecisionTreeClassifier()  
dt.fit(x\_train,y\_train)  
pred\_dt = dt.predict(x\_test)  
accuracy = accuracy\_score(y\_test,pred\_dt)\*100  
  
print(f"Accuracy Score:", accuracy)  
print(f"roc\_auc\_score: **{**roc\_auc\_score(y\_test,pred\_dt)\*100**}**")  
print("---------------------------------------------------")  
  
#confusion matrix & classification report  
print(f"Confusion Matrix : **\n** **{**confusion\_matrix(y\_test,pred\_dt)**}\n**")  
print(f"CLASSIFICATION REPORT : **\n** **{**classification\_report(y\_test,pred\_dt)**}**")  
  
#cross validation score  
scores = cross\_val\_score(dt, x\_over, y\_over, cv = 10,scoring = "accuracy" ).mean()\*100  
print("**\n**Cross validation score :", scores)  
  
#result of accuracy minus cv score  
result = accuracy - scores  
print("**\n\n**Accuracy Score - Cross Validation Score :", result)



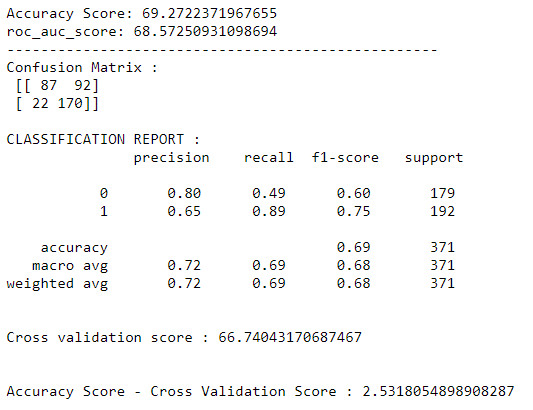
**RandomForestClassifier**

**from** **sklearn.ensemble** **import** RandomForestClassifier  
rf = RandomForestClassifier()  
rf.fit(x\_train,y\_train)  
pred\_rf = rf.predict(x\_test)  
accuracy = accuracy\_score(y\_test,pred\_rf)\*100  
  
print(f"Accuracy Score:", accuracy)  
print(f"**\n**roc\_auc\_score: **{**roc\_auc\_score(y\_test,pred\_rf)\*100**}**")  
print("---------------------------------------------------")  
  
#confusion matrix & classification report  
print(f"Confusion Matrix : **\n** **{**confusion\_matrix(y\_test,pred\_rf)**}\n**")  
print(f"CLASSIFICATION REPORT : **\n** **{**classification\_report(y\_test,pred\_rf)**}**")  
  
#cross validation score  
scores = cross\_val\_score(rf, x\_over, y\_over, cv = 10,scoring = "accuracy" ).mean()\*100  
print("**\n**Cross validation score :", scores)  
  
#result of accuracy minus cv score  
result = accuracy - scores  
print("**\n\n**Accuracy Score - Cross Validation Score :", result)



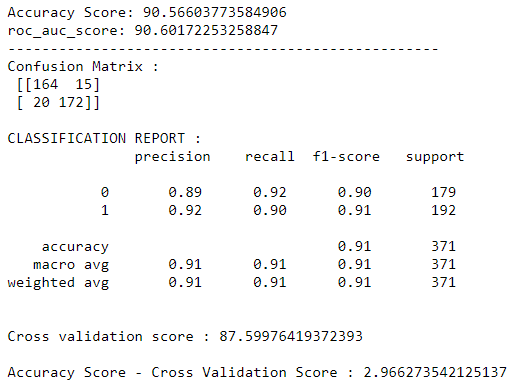
**KNeighborsClassifier**

**from** **sklearn.neighbors** **import** KNeighborsClassifier  
kn = KNeighborsClassifier()  
kn.fit(x\_train,y\_train)  
pred\_kn = kn.predict(x\_test)  
accuracy = accuracy\_score(y\_test,pred\_kn)\*100  
  
print(f"Accuracy Score:", accuracy)  
print(f"roc\_auc\_score: **{**roc\_auc\_score(y\_test,pred\_kn)\*100**}**")  
print("---------------------------------------------------")  
  
#confusion matrix & classification report  
print(f"Confusion Matrix : **\n** **{**confusion\_matrix(y\_test,pred\_kn)**}\n**")  
print(f"CLASSIFICATION REPORT : **\n** **{**classification\_report(y\_test,pred\_kn)**}**")  
  
#cross validation score  
scores = cross\_val\_score(kn, x\_over, y\_over, cv = 10,scoring = "accuracy" ).mean()\*100  
print("**\n**Cross validation score :", scores)  
  
#result of accuracy minus cv score  
result = accuracy - scores  
print("**\n\n**Accuracy Score - Cross Validation Score :", result)



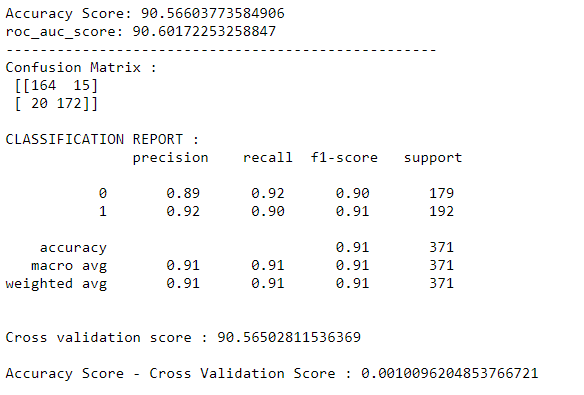
**XGBClassifier**

**from** **xgboost** **import** XGBClassifier  
xgb = XGBClassifier(verbosity = 0)  
xgb.fit(x\_train,y\_train)  
pred\_xgb = xgb.predict(x\_test)  
accuracy = accuracy\_score(y\_test,pred\_xgb)\*100  
  
print(f"Accuracy Score:", accuracy)  
print(f"roc\_auc\_score: **{**roc\_auc\_score(y\_test,pred\_xgb)\*100**}**")  
print("---------------------------------------------------")  
  
#confusion matrix & classification report  
print(f"Confusion Matrix : **\n** **{**confusion\_matrix(y\_test,pred\_xgb)**}\n**")  
print(f"CLASSIFICATION REPORT : **\n** **{**classification\_report(y\_test,pred\_xgb)**}**")  
  
#cross validation score  
scores = cross\_val\_score(xgb, x\_over, y\_over, cv = 10,scoring = "accuracy" ).mean()\*100  
print("**\n**Cross validation score :", scores)  
  
#result of accuracy minus cv score  
result = accuracy - scores  
print("**\n**Accuracy Score - Cross Validation Score :", result)



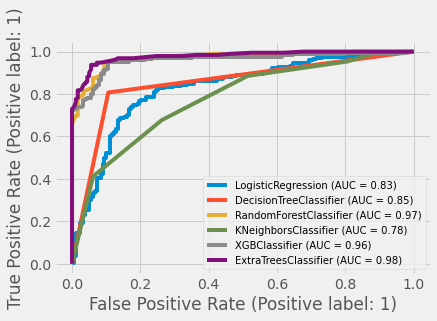
**ExtraTreesClassifier**

**from** **sklearn.ensemble** **import** ExtraTreesClassifier  
ext = ExtraTreesClassifier()  
ext.fit(x\_train,y\_train)  
pred\_ext = xgb.predict(x\_test)  
accuracy = accuracy\_score(y\_test,pred\_ext)\*100  
  
print(f"Accuracy Score:", accuracy)  
print(f"roc\_auc\_score: **{**roc\_auc\_score(y\_test,pred\_ext)\*100**}**")  
print("---------------------------------------------------")  
  
#confusion matrix & classification report  
print(f"Confusion Matrix : **\n** **{**confusion\_matrix(y\_test,pred\_ext)**}\n**")  
print(f"CLASSIFICATION REPORT : **\n** **{**classification\_report(y\_test,pred\_ext)**}**")  
  
#cross validation score  
scores = cross\_val\_score(ext, x\_over, y\_over, cv = 10,scoring = "accuracy" ).mean()\*100  
print("**\n**Cross validation score :", scores)  
  
#result of accuracy minus cv score  
result = accuracy - scores  
print("**\n**Accuracy Score - Cross Validation Score :", result)



#### ****AUC & ROC Curve****

**from** **sklearn.metrics** **import** plot\_roc\_curve  
disp = plot\_roc\_curve(LR, x\_test, y\_test)  
plot\_roc\_curve(dt, x\_test, y\_test, ax = disp.ax\_)  
plot\_roc\_curve(rf, x\_test, y\_test, ax = disp.ax\_)  
plot\_roc\_curve(kn, x\_test, y\_test, ax = disp.ax\_)  
plot\_roc\_curve(xgb, x\_test, y\_test, ax = disp.ax\_)  
plot\_roc\_curve(ext, x\_test, y\_test, ax = disp.ax\_)  
  
plt.legend(prop={"size" :10} ,loc = 'lower right')  
plt.show()



We can see ExtraTreesClassifier is giving least difference in accuracy and cv score which is of 0.001. And looking at AUC and roc curve RandomForestClassifier, XGBClassifier, and ExtraTreesClassifier are giving nearly same performances, among which ExtraTreesClassifier is giving heighst AUC.

After these observations and evaluations I can conclude that the best suitable model is ExtraTreesClassifier.

**ExtraTreesClassifier:** ExtraTreesClassifier will implements a meta estimator which fits a number of randomized decision trees (extra-trees) on various sub-samples of the dataset and use averaging to improve the accuracy in prediction and minimize the over-fitting problem.

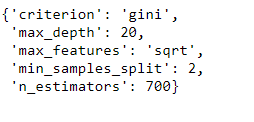
#### Hyperparameter Tuning

Below you can see the code of the hyperparameter tuning for the parameters like criterion, max\_depth, n\_estimators, max\_features and min\_samples\_split with ExtraTreeClassifier.

grid\_params = {  
 'criterion':['gini','entropy'],  
 'max\_depth': [10,12,15,20,22],  
 'n\_estimators':[500,700,1000,1200],  
 'max\_features':['aoto','sqrt','log2'],  
 'min\_samples\_split': [2]  
 }

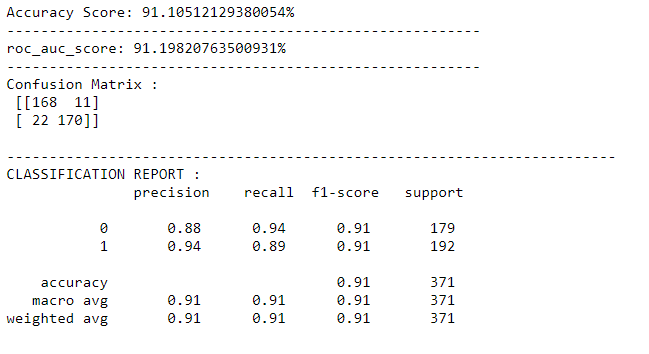
#train the model with given parameters using GridSearchCVGCV = GridSearchCV(ExtraTreesClassifier(), grid\_params, cv = 5)  
GCV.fit(x\_train,y\_train)

GCV.best\_params\_ #printing the best parameters



Great! After hyperparameter tuning we get the above best parameters for ExtraTreeClassifier which best fits our data.

#### Final Model

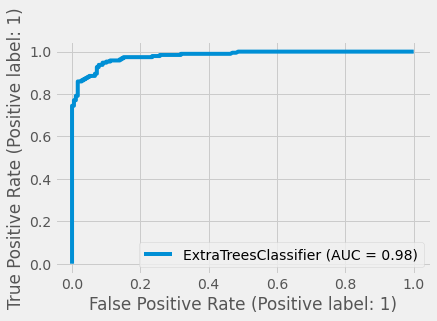
model = ExtraTreesClassifier(criterion = 'gini', max\_depth = 20, max\_features = 'sqrt', min\_samples\_split = 2, n\_estimators = 700)  
model.fit(x\_train,y\_train)  
pred = model.predict(x\_test)  
  
print(f"Accuracy Score: **{**accuracy\_score(y\_test,pred)\*100**}**%")  
print("--------------------------------------------------------")  
  
print(f"roc\_auc\_score: **{**roc\_auc\_score(y\_test,pred)\*100**}**%")  
print("--------------------------------------------------------")  
  
print(f"Confusion Matrix : **\n** **{**confusion\_matrix(y\_test,pred)**}\n**")  
print("------------------------------------------------------------------------")  
print(f"CLASSIFICATION REPORT : **\n** **{**classification\_report(y\_test,pred)**}**")

Great we have got improved accuracy after hyperparameter tuning for final model. we have improved our accuracy from **90.56%** to **91.10%**.

And also we got average **precision** of **91%** which means our final model predicts 91% of the time, a fraud report correctly. And also **recall** will tell us that our model will predict fraud report **91%** of the time as actual results.

### AUC ROC Curve for final model

plot\_roc\_curve(model, x\_test, y\_test)  
plt.show()



We can see the above improved **AUC ROC curve** showing better performance for final model.

### Summary

We started with data loading and data exploration. while exploring the dataset I checked for null values and found that there are no any null values but after doing some observations I came to know that instead of null values some of the columns filled with **‘?’** mark. I replaced it by suitable method. For visualization process I used matplotlib and seaborn.

During data preprocessing we dropped unwanted columns, created new features using other features. Then removed outliers from dataset, treated skewed data and converted categorical data into numerical data by doing encoding.

After that we have selected best random state for our model, and started training and testing for different models and using some evaluation methods selected best suitable model as our final model (ExtraTreeClassifier). By doing hyperparameter tuning we improved our model accuracy for the final model and discussed about model precision and recall.

There is still scope for improvement, like doing a more extensive feature engineering, by comparing and plotting the features against each other. Also we can do more extensive hyperparameter tuning for different machine learning models to get improved overall accuracy for our model so that we can predict automobile fraud report much correctly.