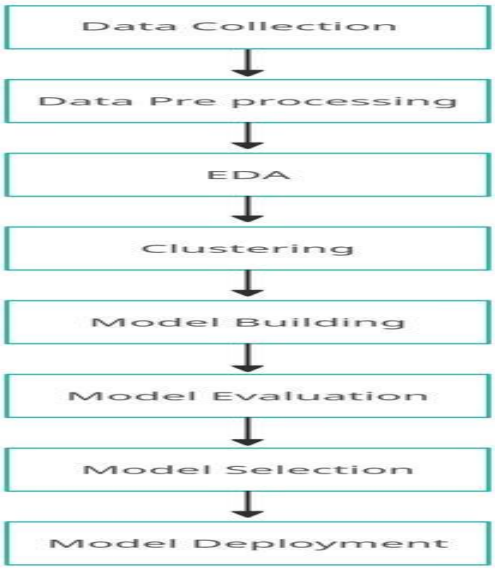
Introduction

Insurance fraud has existed since the beginning of insurance as a commercial enterprise. Fraudulent claims account for a significant portion of all claims received by insurers, and cost billions of dollars annually. Types of insurance fraud are diverse and occur in all areas of insurance. Insurance crimes also range in severity, from slight claims to purposefully causing accidents or damage. Fraudulent activities affect the lives of innocent people, both directly through accident or damage, and indirectly by the crimes leading to higher insurance premiums. Insurance fraud poses a significant problem, and governments and other organizations try to stop such activity.

People who commit insurance fraud include:
• organized criminals who steal large sums through fraudulent business activities,
• professionals and technicians who inflate service costs or charge for services not rendered, and
• ordinary people who want to cover their deductible or view filing a claim as an opportunity to make a little money.

Auto fraud is one of the largest and most well-known problems that insurers face. Fraudulent claims can be highly expensive for each insurer. Therefore, it is important to know which claims are correct and which are not. Ideally, an insurance agent would have the capacity to investigate each case and conclude whether it is genuine or not. However, this process is not only time consuming, but costly. Sourcing and funding the skilled labor required to review each of the thousands of claims that are filed a day is simply unfeasible. This is where machine learning comes in to save the day. Once the proper data is fed to the system it'll be very easy to find out if the claim is genuine or not.



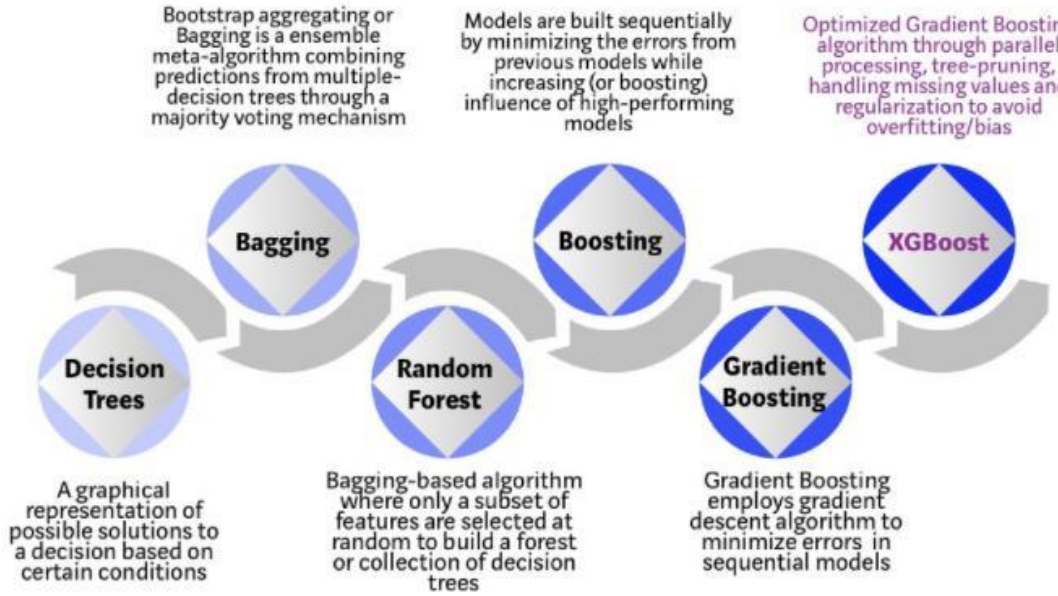
Simple steps required for implementation of machine learning.

Machine Learning algorithm required for the project
1.SVM
2.XGBoost

XGBoost

• XGBoost is a decision-tree-based ensemble Machine Learning algorithm that uses a gradient boosting framework.
• XGBoost approaches the process of sequential tree building using parallelized implementation.
• XGBoost naturally admits sparse features for inputs by automatically 'learning' best missing value depending on training loss and handles different types of patterns in the data more efficiently.
• XGBoost is a popular and efficient open-source implementation of the gradient boosted trees algorithm.
• When using gradient boosting for regression, the weak learners are regression trees, and each regression tree maps an input data point to one of its leafs that contains a continuous score
• The algorithm comes with built-in cross-validation method at each iteration, taking away the need to explicitly program.

XGBoost Steps

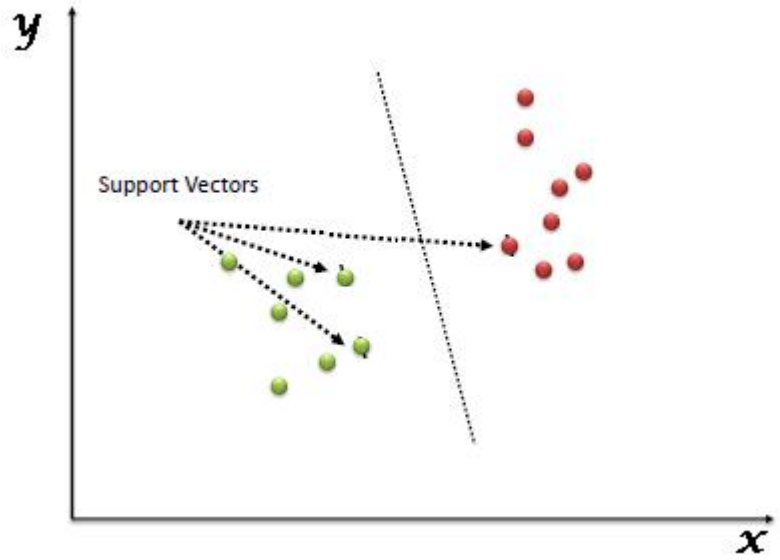


Support Vector Machine (SVM)

What is Support Vector Machine?

• "Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.
• However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.

• Then, we perform classification by finding the hyper-plane that differentiates the two classes very well.



Support Vector Machine

• Support Vectors are simply the co-ordinates of individual observation.
• The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).
• In the SVM classifier, it is easy to have a linear hyper-plane between these two classes. But another burning question which arises is, should we need to add this feature manually to have a hyper plane. No, the SVM algorithm has a technique called the kernel trick.
• The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e., it converts not separable problem to separable problem. It is mostly useful in non-linear separation problem. Simply put, it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.
• When we look at the hyper-plane in original input space it looks like a circle:

DATA PREPROCESSING IN MACHINE LEARNING

• Data preprocessing is a process of preparing the raw data and making it suitable for a machine learning model.
• It is the first and crucial step while creating a machine learning model.
• When creating a machine learning project, it is not always a case that we come across the clean and formatted data.
• Data imputations. Most ML frameworks include methods and APIs for balancing or filling in missing data. Techniques generally include imputing missing values with standard deviation, mean, median and k-nearest neighbors (k-NN) of the data in the given field.
• Oversampling. Bias or imbalance in the dataset can be corrected by generating more observations/samples with methods like repetition, bootstrapping or Synthetic Minority Over Sampling Technique (SMOTE), and then adding them to the under-represented classes.

• Data integration. Combining multiple datasets to get a large corpus can overcome incompleteness in a single dataset.
• Data normalization. The size of a dataset affects the memory and processing required for iterations during training. Normalization reduces the size by reducing the order and magnitude of data.
• And while doing any operation with data, it is mandatory to clean it and put in a formatted way. So, for this, we use data preprocessing task.

EDA (Exploratory Data Analysis)
• Exploratory data analysis (EDA) is used to analyze and investigate data sets and summarize their main characteristics, often employing data visualization methods.
• It helps determine how best to manipulate data sources to get the answers you need, making it easier for data scientists to discover patterns, spot anomalies, test a hypothesis, or check assumptions.
• EDA is primarily used to see what data can reveal beyond the formal modeling or hypothesis testing task and provides a provides a better understanding of data set variables and the relationships between them.
• It can also help determine if the statistical techniques you are considering for data analysis are appropriate. Originally developed by American mathematician John Tukey in the 1970s, EDA techniques continue to be a widely used method in the data discovery process today.

There are four primary types of EDA:
• **Univariate non-graphical.** This is simplest form of data analysis, where the data being analyzed consists of just one variable. Since it's a single variable, it doesn't deal with causes or relationships. The main purpose of univariate analysis is to describe the data and find patterns that exist within it.
• **Univariate graphical.** Non-graphical methods don't provide a full picture of the data. Graphical methods are therefore required. Common types of univariate graphics include:
• Stem-and-leaf plots, which show all data values and the shape of the distribution.
• Histograms, a bar plot in which each bar represents the frequency (count) or proportion (count/total count) of cases for a range of values.
• Box plots, which graphically depict the five-number summary of minimum, first quartile, median, third quartile, and maximum.
• **Multivariate nongraphical**: Multivariate data arises from more than one variable. Multivariate non-graphical EDA techniques generally show the relationship between two or more variables of the data through cross-tabulation or statistics.
• **Multivariate graphical**: Multivariate data uses graphics to display relationships between two or more sets of data. The most used graphic is a grouped bar plot or bar chart with each group representing one level of one of the variables and each bar within a group representing the levels of the other variable.
• Other common types of multivariate graphics include:
• Scatter plot, which is used to plot data points on a horizontal and a vertical axis to show how much one variable is affected by another.
• Multivariate chart, which is a graphical representation of the relationships between factors and a response.
• Run chart, which is a line graph of data plotted over time.

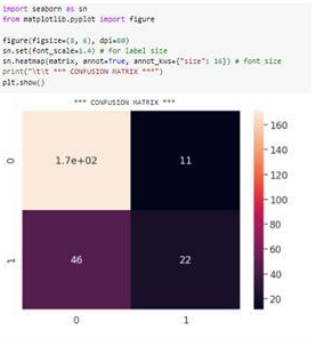
• Bubble chart, which is a data visualization that displays multiple circles (bubbles) in a two dimensional plot.
• Heat map, which is a graphical representation of data where values are depicted by color.
• Building machine learning models that have the ability to generalize well on future data requires thoughtful consideration of the data at hand and of assumptions about various available training algorithms.
• Ultimate evaluation of a machine learning model's quality requires an appropriate selection and interpretation of assessment criteria.
• Machine learning consists of algorithms that can automate analytical model building.
• Using algorithms that iteratively learn from data, machine learning models facilitate computers to find hidden insights from Big Data without being explicitly programmed where to look.
• This has given rise to a of many applications based on Machine learning

Model evaluation

SVM


```
[466] from sklearn.svm import SVC
      sv_classifier=SVC()

[467] y_pred = sv_classifier.fit(train_x, train_y).predict(test_x)

[468] from sklearn.metrics import accuracy_score

[469] sc=accuracy_score(test_y,y_pred)
      print('SVM ACCURACY:',sc)

      SVM ACCURACY: 0.728
```

XGB




```
[396] from xgboost import XGBClassifier

[397] xgb=XGBClassifier()

[398] y_pred = xgb.fit(train_x, train_y).predict(test_x)

[409] ac2=accuracy_score(test_y,y_pred)
      print('XGB ACCURACY:',ac2)

      XGB ACCURACY: 0.772
```

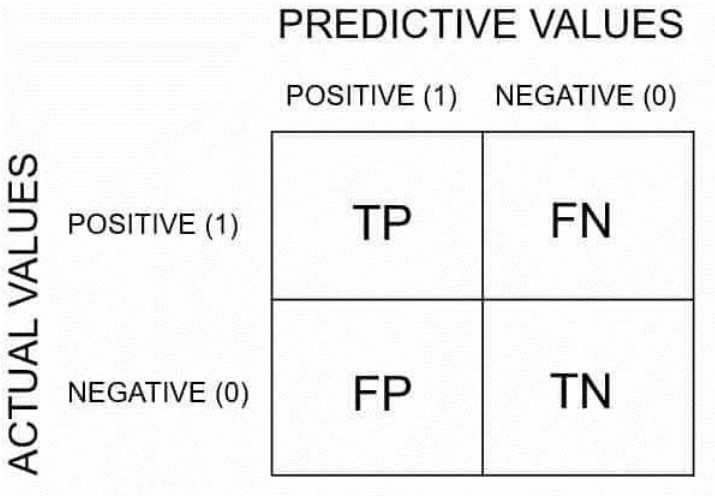
Model evaluation aims to estimate the generalization accuracy of a model on future (unseen/out-of sample) data.

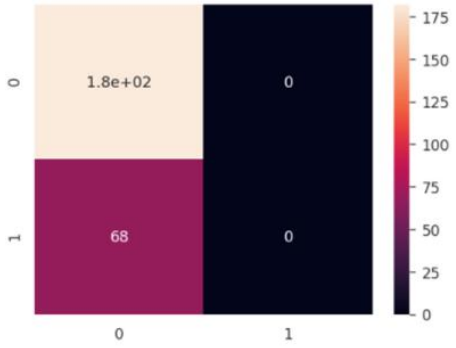
Model Evaluation and Selection

Model Evaluation is a process of evaluating the models based on various performance measuring parameters. Evaluation of the model provides a clear picture of the model's efficiency and helps to select the best model for performing prediction.

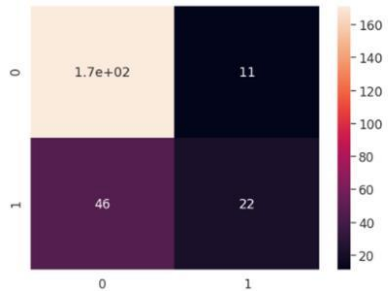
In this study, as we are using two algorithms, "SVM" and "XGBoost", so after building the models, these models are undergone a model evaluation phase. Scikit-learn python libraries are used extensively in this study for practical illustration.

We use various performance measuring parameters like the Confusion matrix which further leads for calculations of Accuracy, Precision, Recall, and F1 Score. Also, we are using Area under the curve (AUC) .





confusion matrix of SVM



Confusion matrix of xgb

Measure	Formula	SVM	XGBoost
Accuracy	$\frac{TP + TN}{TP + TN + FN + FP}$	0.728	0.772
Precision	$\frac{TP}{TP + FP}$	1.000	0.939
Recall	$\frac{TP}{TP + FN}$	0.000	0.323
F1-score	$\frac{2 * Prec * Rec}{(Prec + Rec)}$	0.842	0.857
Error rate	$\frac{FP + FN}{TP + TN + FN + FP}$	0.272	0.228

Based on the confusion matrices, we calculated the other parameters like accuracy, precision, recall, f1 score, and error rate.
The accuracy of XG Boost(77.2%) is higher compared to SVM's(72.8%), though the Precision of SVM is 100% its Recall rate is 0.00% which shows an inconsistency, unlike XG Boost. F1 score is one of the great performance measurements in this XG Boost is shown more rate(85.7%) than SVM's(84.2%). The error rate of the SVM shows up to 27.2% and XG Boost is 22.8% which is less than SVM. In the above measurement, XG Boost seems to be more efficient than SVM.
Although evaluation is not yet finalized based on only these parameters because as we can see in the confusion matrix, data is not distributed uniformly which shows that our dataset is imbalanced therefore these above evaluation parameters are not enough to judge the model. Therefore we use parameters such as True Positive Rate(TPR), True Negative Rate(TNR), False Positive Rate(FPR), False Negative Rate(FNR).

Measure	Formula	SVM	XGBoost
True Positive Rate	TP/P	1.000	0.939
True Negative Rate	TN/N	0.000	0.323
False Positive Rate	FP/P	0.370	0.252
False Negative Rate	FN/N	0.000	0.161

In Ideal scenarios, the True Positive Rate(TPR) and True Negative Rate(TNR) should be high, and False Positive Rate(FPR) and False Negative Rate(FNR) should be low.
Here we can see that the TPR of SVM is high(100%) than XG Boost(93.9%) but the TNR of SVM is 0.00% which is less than the XG boost who is bearing 32.3%.
FPR of XG Boost(23.2%) is less(as recommended) than SVM(37.0%). But FNR of SVM(0.00%) is less than XG Boost(16.1%) in a small margin. In the above illustration, XGBoost seems to be efficient and consistent in all cases compared to SVM. We also using Area under Curve(AUC), AUC represents the degree of separability. That means it tells that how much the model is capable of distinguishing the 33 classes in our case it's fraud and Not fraud. Higher the AUC the better the model is at predicting fraud as fraud and Not fraud as Not fraud. An exceptional model has AUC near to 100%, and an ideal model has above 50%. But when the model is less than 50% or 50%, that means the model is incapable of distinguishing the classes.

Measure	Full form	SVM	XGBoost
AUC	Area under curve	0.500	0.631

In above table, we observe that the AUC rate of XG Boost is 63.1% which is ideal as it is greater than 50%, but whereas SVM is exactly 50%, which shows that in this case, SVM is incapable of distinguishing the classes so SVM is not an ideal model to select. By the process of evaluating, we yield a significant outcome on selecting the best model among SVM and XG Boost, considering the evaluation results we are selecting XG Boost which is shown more efficient than SVM.

Experimental comparison

Here, we present the experimental comparison between the performance score of two algorithms that are used to build ML models to perform prediction on our dataset. Using evaluation technique, we found that the accuracy of XG Boost (77.2%) is higher compared to SVM's (72.8%) though we can't judge a model just by accuracy as it showed an imbalanced distribution of data in the confusion matrix, we performed various model evaluating parameters which briefly explained in Section 2.6. In addition to it, we also used Area under Curve (AUC), AUC represents the degree of separability between classes. In which, the AUC rate of XG Boost is 63.1% which is ideal as it is greater than 50%, but whereas SVM is exactly 50%.

- With the evaluation process result, we select the best model which addresses our problem statement and efficient enough to perform the prediction.
- In the model selection phase of this study, we have chosen XG Boost over SVM based on the performance in the evaluation phase.

Conclusion

• In this study, we imported the relevant dataset which correlated to the problem statement then we used Machine learning techniques like Support Vector Machine (SVM) and XG Boost to build the predictive models.

• These models were undergone a model evaluation process to estimate accuracy, precision, recall, f1 score, error rate, and also AUC rate. By comparing the models on basis of the result of model evaluation, XG Boost outperformed SVM.

• Now this proposed system is ready to predict whether the claimed insurance is "Fraud "or "Not Fraud" .

Some snippets of the code

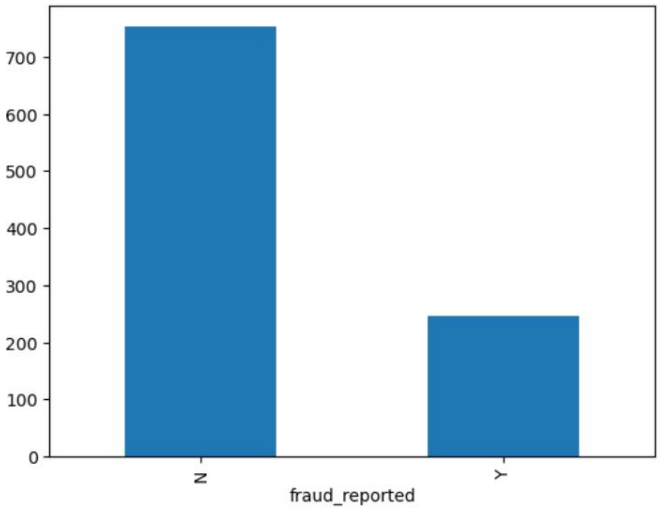
]:
```
import pandas as pd

#Data Loading

data = pd.read_csv(r'C:\Users\Zenbook\Downloads\insurance_claims.csv')

original_data = data.copy()

data.head()
```

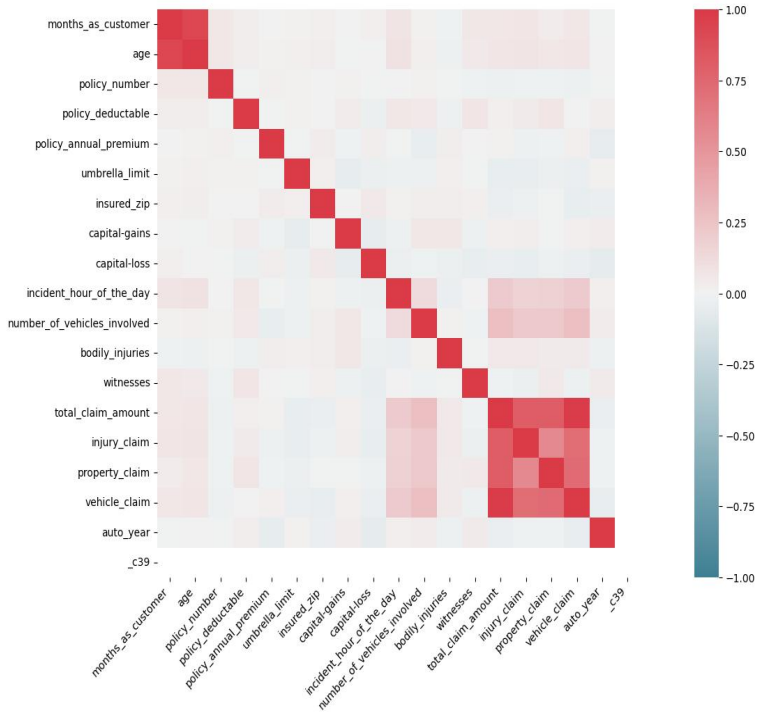
]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	uml
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	

[6]:
```
#Fraud Reported Stats
df_count_fraud = data.groupby(['fraud_reported']).count()
df_fraud = df_count_fraud['policy_number']
df_fraud.plot.bar(x='Fraud Reported', y='Count')
```

:[6]: <Axes: xlabel='fraud_reported'>





3]:
```
#Model Training
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.model_selection import train_test_split

y = data['fraud_reported']
X = data.drop(['fraud_reported'], axis=1)

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
```

4]:
```
from sklearn import neighbors

#K-Nearest Neighbors
print("KNN Score :")
KNNClassifier = neighbors.KNeighborsClassifier(n_neighbors=12, weights='distance')
KNNClassifier.fit(X=X_train,y=y_train)
KNNClassifier.score(X_test,y_test)
```

```
KNN Score :
```

4]:
```
0.7333333333333333
```

5]:
```
KNN_y_predicted = KNNClassifier.predict(X_test)
class_names = np.unique(np.array(y_test))
confusion_matrix(y_test, KNN_y_predicted)
```

5]:
```
array([[108,   5],
       [ 35,   2]], dtype=int64)
```

:
```
from sklearn.metrics import classification_report
from sklearn.model_selection import cross_val_score

print(classification_report(y_test, KNN_y_predicted))

scores = cross_val_score(KNNClassifier, X, y, cv=10, scoring='accuracy')
knn_accuracy = scores.mean()
print('Cross-Validated Accuracy: %0.2f' % knn_accuracy)
```

```
              precision    recall  f1-score   support

           0       0.76      0.96      0.84       113
           1       0.29      0.05      0.09        37

    accuracy                           0.73       150
   macro avg       0.52      0.50      0.47       150
weighted avg       0.64      0.73      0.66       150

Cross-Validated Accuracy: 0.73
```

```
from sklearn.svm import SVC

#Support Vector Machine
SVMClassifier = SVC(kernel='rbf',probability=True,random_state=42, gamma='auto')
SVMClassifier.fit(X_train, y_train)
print("SVM Score :")
SVMClassifier.score(X_test,y_test)
```

```
SVM Score :
0.7533333333333333
```

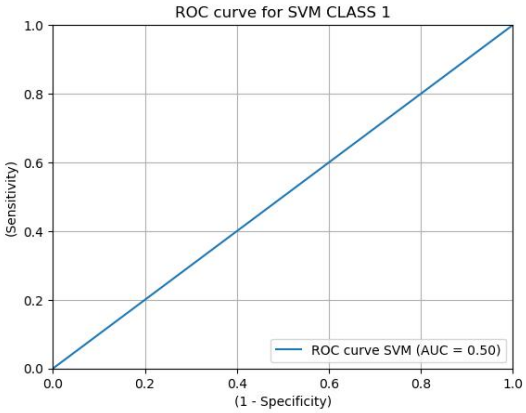
```
SVM_y_predicted = SVMClassifier.predict(X_test)
class_names = np.unique(np.array(y_test))
confusion_matrix(y_test, SVM_y_predicted)
```

```
array([[113,   0],
       [ 37,   0]], dtype=int64)
```

```
from sklearn.metrics import classification_report
report = classification_report(y_test, SVM_y_predicted)
print(report)

scores = cross_val_score(SVMClassifier, X, y, cv=10, scoring='accuracy')
svm_accuracy = scores.mean()
print('Cross-Validated Accuracy: %0.2f' % svm_accuracy)
```

```
              precision    recall  f1-score   support

           0       0.75      1.00      0.86       113
           1       0.00      0.00      0.00        37

    accuracy                           0.75       150
   macro avg       0.38      0.50      0.43       150
weighted avg       0.57      0.75      0.65       150

Cross-Validated Accuracy: 0.75
```

```
svm_pred_prob = SVMClassifier.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, svm_pred_prob)
roc_auc = auc(fpr, tpr)
lw = 2
plt.plot(fpr, tpr,label='ROC curve SVM (AUC = %0.2f)' % roc_auc)
plt.xlim([0.0, 1])
plt.ylim([0.0, 1])
plt.title('ROC curve for SVM CLASS 1')
plt.xlabel('(1 - Specificity)')
plt.ylabel('(Sensitivity)')
plt.grid(True)
plt.legend(loc="lower right")
plt.show()
```



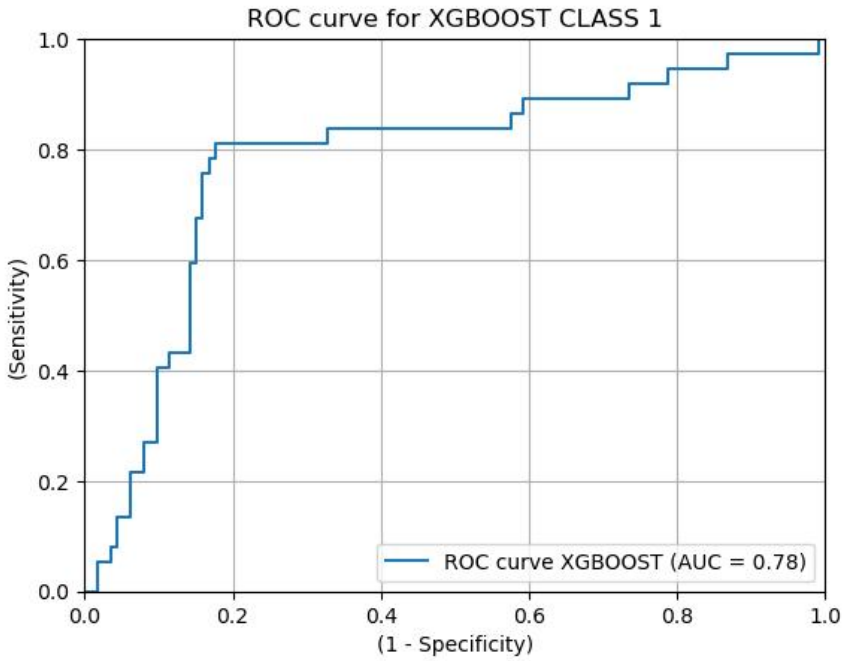
```
from xgboost import XGBClassifier
#XGBOOST Classifier
model_xgb = XGBClassifier()
model_xgb.fit(X_train, y_train, verbose=False)
print("XGBClassifier Score :")
model_xgb.score(X_test,y_test)
```

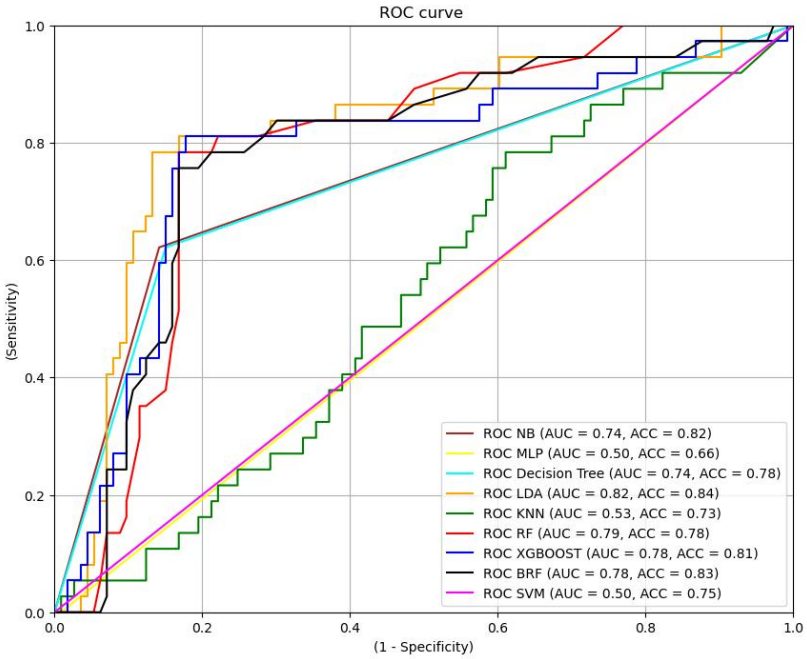
```
XGBClassifier Score :
 0.76
```

```
xgboost_y_predicted = model_xgb.predict(X_test)
report = classification_report(y_test, xgboost_y_predicted)

print(report)

scores = cross_val_score(model_xgb, X, y, cv=10, scoring='accuracy')
xgb_accuracy = scores.mean()
print('Cross-Validated Accuracy: %0.2f' % xgb_accuracy)
```

```
              precision    recall  f1-score   support

           0       0.83      0.86      0.84       113
           1       0.52      0.46      0.49        37

    accuracy                           0.76       150
   macro avg       0.67      0.66      0.66       150
weighted avg       0.75      0.76      0.76       150

Cross-Validated Accuracy: 0.81
```

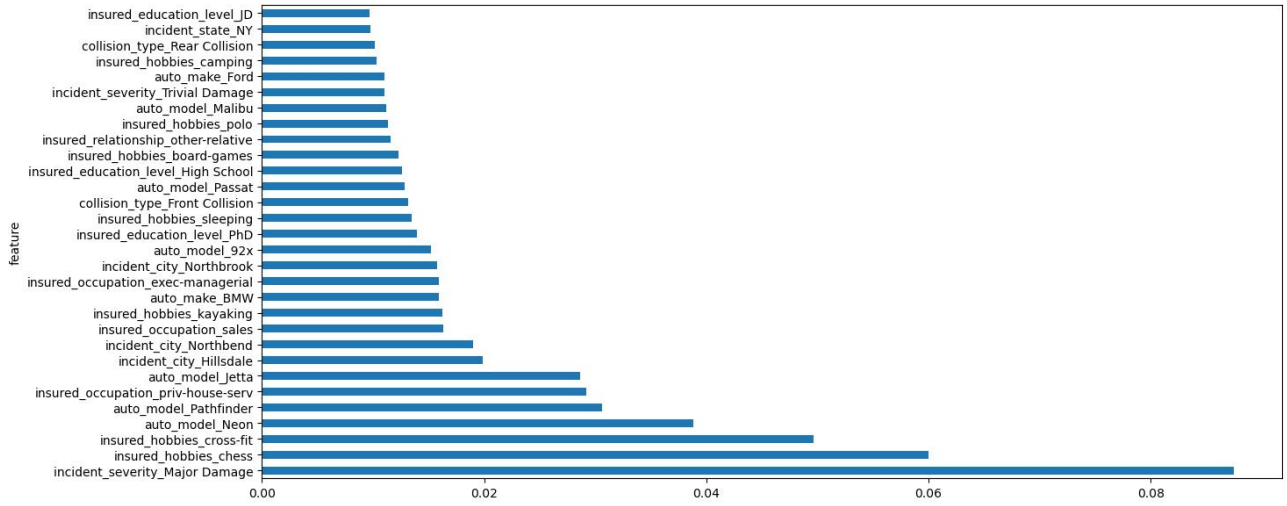
```
xgb_pred_prob = model_xgb.predict_proba(X_test)[:, 1]
fpr, tpr, thresholds = roc_curve(y_test, xgb_pred_prob)
roc_auc = auc(fpr, tpr)
lw = 2
plt.plot(fpr, tpr,label='ROC curve XGBOOST (AUC = %0.2f)' % roc_auc)
plt.xlim([0.0, 1])
plt.ylim([0.0, 1])
plt.title('ROC curve for XGBOOST CLASS 1')
plt.xlabel('(1 - Specificity)')
plt.ylabel('(Sensitivity)')
plt.grid(True)
plt.legend(loc="lower right")
plt.show()
```






```
#XGBOOST model has better ROC Curve and Cross-validated accuracy, plot feature importance
def rf_feat_importance(m, df):
    return pd.DataFrame({'feature':df.columns, 'imp':m.feature_importances_}
                        ).sort_values('imp', ascending=False)
def plot_fi(fi):
    return fi.plot('feature', 'imp', 'barh', figsize=(15,7), legend=False)

fi = rf_feat_importance(model_xgb, X_train); fi[:15]
```

	feature	imp
79	incident_severity_Major Damage	0.087458
50	insured_hobbies_chess	0.060024
51	insured_hobbies_cross-fit	0.049644
149	auto_model_Neon	0.038847
151	auto_model_Pathfinder	0.030619
39	insured_occupation_priv-house-serv	0.029176
142	auto_model_Jetta	0.028615
97	incident_city_Hillsdale	0.019856
98	incident_city_Northbend	0.018978
42	insured_occupation_sales	0.016345
56	insured_hobbies_kayaking	0.016218
110	auto_make_BMW	0.015962
34	insured_occupation_exec-managerial	0.015948




```
#As XGBOOST has the best results with an AUC of 0.82, using this model for scoring
test_target = y_test.copy()
test_target.reset_index(drop=True, inplace=True)
test_target = test_target.replace({1:'Y', 0:'N'})

predicted_target = model_xgb.predict(X_test)
predicted_target = pd.Series(predicted_target).replace({1:'Y', 0:'N'})
```

```
ranks=pd.DataFrame(data=
                   {
                       'RealClass':test_target,
                       'PredictedClass':predicted_target,
                       'rank':xgb_pred_prob
                   })
ranks.sort_values(by=['rank'],ascending=False,inplace=True)
ranks.head()
```

	RealClass	PredictedClass	rank
34	N	Y	0.996424
149	N	Y	0.978848
7	Y	Y	0.976585
127	Y	Y	0.975345
82	N	Y	0.963762

```
top = ranks.where(ranks['rank']>0.5,).dropna()
top.head()
```

	RealClass	PredictedClass	rank
34	N	Y	0.996424
149	N	Y	0.978848
7	Y	Y	0.976585
127	Y	Y	0.975345
82	N	Y	0.963762