**Insurance Claims- Fraud Detection:**

We all are aware that fraud is one of the largest and most well-known problems that insurers face. This project focuses on claim data of a car insurance company. Fraudulent claims can be highly expensive for each insurer. Therefore, it is important to know which claims are correct and which are not. It is not possible for insurance companies to check all claims personally since this will cost simply too much time and money. In this project, we will take advantage of the largest asset which insurers have in the fight against fraud: Data. We employ various attributes about the claims, insured people and other circumstances which are included in the data by the insurer. Separating different groups of claims and the corresponding rates of fraud within those groups provide new insights.

Furthermore, we use machine learning to predict which claims are likely to be fraudulent. This information can narrow down the list of claims that need a further check. It enables an insurer to detect more fraudulent claims.

**Problem Definition:**

The goal of our project is to build a model that can detect auto insurance fraud. The challenge behind fraud detection in machine learning is that frauds are far less common when compared to legit insurance claims.

Insurance fraud detection is a challenging problem, given the variety of fraud patterns and relatively small ratio of known frauds in typical samples. While building detection models, we need to balance the savings from loss prevention with the cost of false alerts. Machine learning techniques allow for improving predictive accuracy, enabling loss control units to achieve higher coverage with low false positive rates.

Insurance frauds cover the range of improper activities which an individual may commit in order to achieve a favorable outcome from the insurance company. This could range from staging the incident, misrepresenting the situation including the relevant actors and the cause of the incident and finally the extent of damage caused.

# **Data Analysis:**

In this project, we have a dataset which has the details of the insurance policy along with the customer details. We also have the details of the accident on the basis of which the claims have been made.

The given dataset contains 1000 rows and 40 columns. The column names are like policy number, policy bind date, policy annual premium, incident severity, incident location, auto model, etc.

The obvious con of this data set is the small sample size. However, there are still many companies who do not have big data sets. The ability to work with what is available is crucial for any company looking to transition into leveraging data science.

By checking the null values in our data, we see that there are some variables which contain the null values character ‘?’.

# **Exploratory data analysis:**

**-Dependent variable:** Exploratory data analysis was conducted starting with the dependent variable, Fraud\_reported. We see that there were 247 frauds and 753 non-frauds. 24.7% of the data were frauds while 75.3% were non-fraudulent claims.

-**Correlations among variables:** Heatmap was plotted for variables with at least 0.3 Pearson’s correlation coefficient, including the DV. Month as a customer and age had a correlation of 0.92. Probably because drivers buy auto insurance when they own a car and this time measure only increases with age. Apart from that, there don’t seem to be many correlations in the data. There don’t seem to be multicollinearity problems except maybe that all the claims are all correlated, and somehow total claims have accounted for them. However, the other claims provide some granularity that will not otherwise be captured by total claims. Thus, these variables were kept.

-**Visualizing variables:** The value of fraud reported differs across hobbies of the customer. It seems like chess players and crossfitters have higher tendencies to fraud.

We see that major incidents' severity have the highest fraud cases that exceed non-fraud cases.

The total\_claim\_amount is high in Saab and Subaru auto\_make.

The Injury\_claim is found highest in the Nissan.

We have also checked the correlation between dependent and independent variables.

# **Pre-processing Pipeline:**

Data preprocessing is a predominant step in machine learning to yield highly accurate and insightful results. Greater the quality of data, the greater the reliability of the produced results. **Incomplete, noisy, and inconsistent data** are the inherent nature of real-world datasets. Data preprocessing helps in increasing the quality of data by filling in missing incomplete data, smoothing noise, and resolving inconsistencies.

**-Incomplete data** can occur due to many reasons. Appropriate data may not be persisted due to a misunderstanding, or because of instrument defects and malfunctions.

-**Noisy data** can occur for a number of reasons like having incorrect feature values. The instruments used for the data collection might be faulty. Data entry may contain human or instrument errors. Data transmission errors might occur as well.

There are many stages involved in data preprocessing which are as below:

-**Data cleaning** attempts to impute missing values, removing outliers from the dataset.

-**Data integration** integrates data from a multitude of sources into a single data warehouse.

-**Data transformation** such as normalization, may be applied. For example, normalization may improve the accuracy and efficiency of mining algorithms involving distance measurement.

-**Data reduction** can reduce the data size by dropping out redundant features. Feature selection and feature extraction techniques can be used.

**Treating null values:**

Sometimes there are certain columns which contain the null value used to indicate missing or unknown values or maybe the value doesn’t exist. In our dataset the null values are present in columns collision\_type, property\_damage, police\_report\_available, and \_c39 with 178, 360, 343 and 1000 number of null values.

There are different ways of replacing null values from the dataset, but we have used fillna to replace the null values in our data.

**Converting labels into numeric:**

In machine learning, we usually deal with datasets which contain multiple labels in one or more than one column. These labels can be in the form of words or numbers. To make the data understandable or in human readable form, the training data is often labeled in words.

In our data there are columns with categorical values. The columns like incident\_severity, incident\_state, incident\_type, insured\_hobbies, authorities\_contacted, incident\_city, police\_report\_available, auto\_make, collision\_type, auto\_model, insured\_occupation, insured\_education\_level, property\_damage, insured\_relationship, policy\_state, insured\_sex, fraud\_reported. These columns have to be treated with one hot encoding or the label encoder. The target variable fraud\_reported has to be converted by using label encoder only.

**Label Encoder** is used to convert the labels into numeric form so as to convert it into the machine readable form. Machine learning algorithms can then decide in a better way on how those labels must be operated. It is an important preprocessing step for the structured dataset in supervised learning.

Label encoding in python can be imported from the Sklearn library. Sklearn provides a very efficient tool for encoding. Label encoders encode labels with a value between 0 and n\_classes-1.

**Outliers** are data points that are distant from other similar points. They may be due to variability in the measurement or may indicate experimental errors. If possible, outliers should be excluded from the data set. However, detecting that anomalous instance might be very difficult, and is not always possible.

**Methods to remove outliers:**

-**Z-score —** Call scipy.stats.zscore() with the given data-frame as its argument to get a numpy array containing the z-score of each value in a dataframe. Call numpy.abs() with the previous result to convert each element in the dataframe to its absolute value. Use the syntax (array < 3).all(axis=1) with array as the previous result to create a boolean array.

-**Interquartile range** — The interquartile range can be used to detect the outliers present in the dataframe.

-Calculate the interquartile range for the data by using scipy.stats.iqr module.

-Multiply the interquartile range by 1.5.

-Add 1.5 x interquartile range to the third quartile. Any number greater than this is a suspected outlier.

-Subtract 1.5 x interquartile range from the first quartile. Any number lesser than this is a suspected outlier.

**Balancing our imbalanced data:**

There are different algorithms present to balance the target variable. We used the SMOTE() algorithm to balance our data.

SMOTE(Synthetic minority oversampling technique) works by randomly picking a point from the minority class and computing the k-nearest neighbors of this point. The synthetic points are added between the chosen point and its neighbors.

SMOTE algorithm works in 4 simple steps:

1. Choose a minority class as input vector.
2. Find its k-nearest neighbors.
3. Choose one of these neighbors and place a synthetic point anywhere on the line joining the point under consideration and its chosen neighbors.
4. Repeat the step until the data is balanced.

The original shape of our data was 753 for fraud\_reported with NO value and 247 for YES. The SMOTE algorithm balances our data with the highest number of values present in it.

# **Building machine learning models:**

For building machine learning models there are several models present inside the Sklearn module.

Sklearn provides two types of models i.e. regression and classification. Our dataset’s target variable is to predict whether fraud is reported or not. So for this kind of problem we use classification models.

But before fitting our dataset to its model first we have to separate the predictor variable and the target variable, then we pass this variable to the train\_test\_split method to create a random test and train subset.

Train\_test\_split is a function in sklearn model selection for splitting data arrays into two subsets for training data and testing data. With this function, we don’t need to divide the dataset manually. By default, sklearn train\_test\_split will make random partitions for the two subsets. However, you can also specify a random state for the operation. It gives four outputs x\_train, x\_test, y\_train and y\_test. The x\_train and x\_test contains the training and testing predictor variables while y\_train and y\_test contains the training and testing target variable.

After performing train\_test\_split we have to choose the models to pass the training variable.

We can build as many models as we want to compare the accuracy given by these models and to select the best model among them.

We have selected 5 models:

**-Logistic Regression from sklearn.linear\_model:**

Logistic regression is a supervised learning classification algorithm used to predict the probability of a target variable. The nature of a target or dependent variable is binary, which means there would be only two possible classes 1 (stands for success/yes) or 0 (stands for failure/no). Mathematically, a logistic regression model predicts P(Y=1) as a function of X. It is one of the simplest ML algorithms that can be used for various classification problems such as spam detection, Diabetes prediction, cancer detection etc.

-**DecisionTreeClassifier from sklearn.tree:**

Decision trees can be constructed by an algorithmic approach that can split the dataset in different ways based on different conditions. The two main entities of a tree are decision nodes, where the data is split and leaves, where we get the outcome.

**-RandomForestClassifier from sklearn.ensemble:**

As we know that a forest is made up of trees and more trees means more robust forest. Similarly, a random forest algorithm creates decision trees on data samples and then gets the prediction from each of them and finally selects the best solution by means of voting. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result.

**-XGBClassifier from XGBoost:**

XGBoost is short for “eXtreme Gradient Boosting.” The “eXtreme” refers to speed enhancements such as parallel computing and cache awareness that makes XGBoost approximately 10 times faster than traditional Gradient Boosting. In addition, XGBoost includes a unique split-finding algorithm to optimise trees, along with built-in regularisation that reduces over-fitting. Generally speaking, XGBoost is a faster, more accurate version of Gradient Boosting.

-**GaussianNB from sklearn.naive\_bayes:**

Naive Bayes algorithms are a classification technique based on applying Bayes’ theorem with a strong assumption that all the predictors are independent to each other. In simple words, the assumption is that the presence of a feature in a class is independent of the presence of any other feature in the same class. It is the simplest Naïve Bayes classifier having the assumption that the data from each label is drawn from a simple Gaussian distribution.

# **Conclusion from models:**

We got our best model i.e. RandomForestClassifier with the accuracy score of 85.39%. Here our model predicts 196 true positive cases out of 218 positive cases and 190 true negative cases out of 234 cases. It predicts 22 false positive cases out of 218 positive cases and 44 false negative cases out of 234 cases. It gives the f1 score of 85.20%.

Understanding what precision recalls and f1 score and accuracy do:

**-F1 score**: this is the harmonic mean of precision and recall and gives a better measure of the incorrectly classified cases than the accuracy matrix.

-**Precision:** It is implied as the measure of the correctly identified positive cases from all the predicted positive cases. Thus, it is useful when the costs of False Positives are high.

-**Recall:** It is the measure of the correctly identified positive cases from all the actual positive cases. It is important when the cost of False Negatives is high.

-**Accuracy:** One of the more obvious metrics, it is the measure of all the correctly identified cases. It is most used when all the classes are equally important.

-**Confusion matrix**:A table that is often used to describe the performance of a classification model (or ‘classifier’) on a set of test data for which the true values are known.

NOTE:

**-TN/True Negative:** the cases were negative and predicted negative.

-**TP/True Positive:** the cases were positive and predicted positive.

-**FN/False Negative:** the cases were positive but predicted negative.

-**TN/True Negative:** the cases were negative but predicted positive.

**Hyper parameter tuning:**

Hyper parameter optimisation in machine learning intends to find the hyper parameters of a given machine learning algorithm that deliver the best performance as measured on a validation set. Hyper parameters, in contrast to model parameters, are set by the machine learning engineer before training. The number of trees in a random forest is a hyper parameter while the weights in a neural network are model parameters learned during training. I like to think of hyper parameters as the model settings to be tuned so that the model can optimally solve the machine learning problem.

We will use GridSearchCV for the hyper parameter tuning.

**GridSearchCV:**

In the GridSearchCV approach, the machine learning model is evaluated for a range of hyper parameter values. This approach is called GridSearchCV, because it searches for the best set of hyper parameters from a grid of hyper parameters values.

**ROC curve:** Itis a performance measurement for the classification problems at various threshold settings. ROC is a probability curve and AUC represents the degree or measure of separability. It tells how much the model is capable of distinguishing between classes. Higher the AUC, the better the model is at predicting 0s as 0s and 1s as 1s. By analogy, the Higher the AUC, the better the model is at distinguishing between patients with the disease and no disease.

The ROC curve is plotted with TPR against the FPR where TPR is on the y-axis and FPR is on the x-axis.

# **Final Remarks:**

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.

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

The best and final fitted model was a weighted **Random Forest** that yelled an F1 score of *0.85* and a ROC AUC of *0.95*. The model performed excellently. The model’s F1 score and ROC AUC scores were the highest amongst the other models. In conclusion, the model was able to correctly distinguish between fraud claims and legit claims with high accuracy.

The study is not without limitations. Firstly, this study is restricted by its small sample size. Statistical models are more stable when data sets are larger. It also generalizes better as it takes a bigger proportion of the actual population. Furthermore, the data only captures incident claims of 3 states.