



INSURANCE FRAUD DETECTION USING MACHINE LEARNING

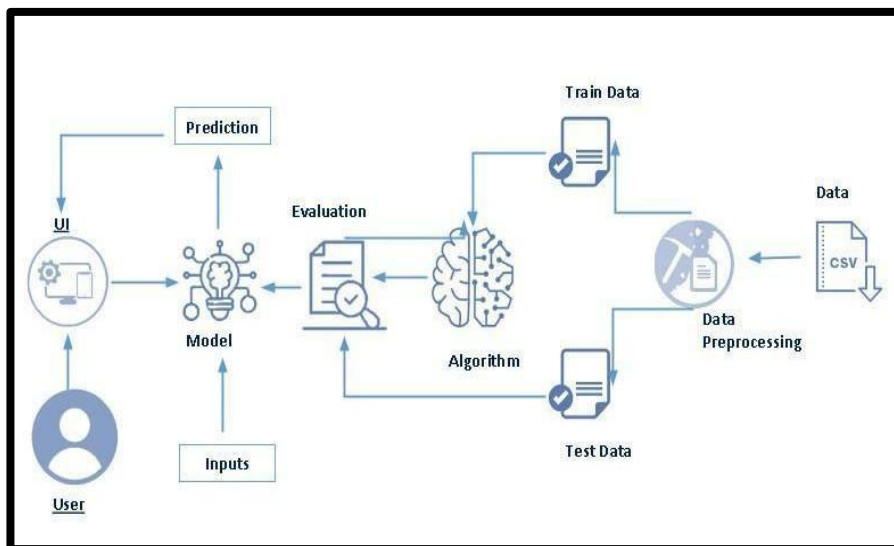
Project Hand-out, Faculty Development Program – NaanMudhalvan

Insurance Fraud Detection Using Machine Learning

Insurance are claimed in order to get a relief amount for any damage cause. Insurance is a means of protection from financial loss , but now-a-days Many people are claiming the Insurance by fraud claims. It can be called as a scam. It is called as fraud claim when a claimant attempts to obtain some benefit or advantage they are not entitled to, or when an insurer knowingly denies some benefit that is due.

These type of Insurance claims cause loss to the company. So, It is necessary to detect the claims which are fraud. The number of cases of insurance fraud that are detected is much lower than the number of acts that are actually committed. So the main purpose of the Insurance Fraud Detection system is to predict the Insurance claim is a Fraud or Legal Claim based on the different appeals and parameters.

Technical Architecture:



Project Flow:

- User interacts with the UI to enter the input.
- Entered input is analysed by the model which is integrated.
- Once model analyses the input the prediction is showcased on the UI

To accomplish this, we have to complete all the activities listed below,

- Define Problem / Problem Understanding
 - Specify the business problem
 - Business requirements
 - Literature Survey
 - Social or Business Impact.
- Data Collection & Preparation
 - Collect the dataset
 - Data Preparation
- Exploratory Data Analysis
 - Descriptive statistical
 - Visual Analysis
- Model Building
 - Training the model in multiple algorithms
 - Testing the model
- Performance Testing & Hyperparameter Tuning
 - Testing model with multiple evaluation metrics
 - Comparing model accuracy before & after applying hyperparameter tuning
- Model Deployment
 - Save the best model
 - Integrate with Web Framework
- Project Demonstration & Documentation
 - Record explanation Video for project end to end solution
 - Project Documentation-Step by step project development procedure

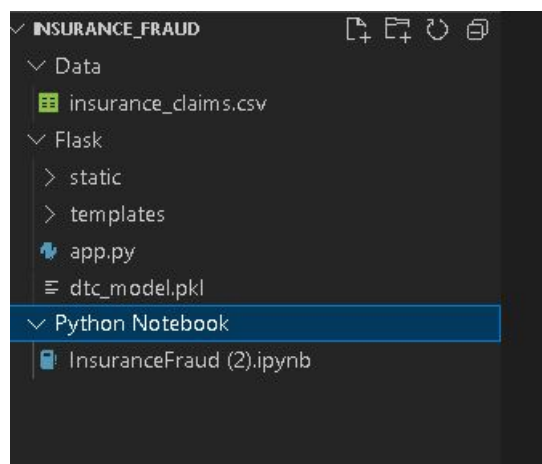
Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

- ML Concepts
 - Supervised learning: <https://www.javatpoint.com/supervised-machine-learning>
 - Unsupervised learning: <https://www.javatpoint.com/unsupervised-machine-learning>
 - Decision tree: <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
 - Random forest: <https://www.javatpoint.com/machine-learning-random-forest-algorithm>
 - KNN: <https://www.javatpoint.com/k-nearest-neighbor-algorithm-for-machine-learning>
 - Xgboost: <https://www.analyticsvidhya.com/blog/2018/09/an-end-to-end-guide-to-understand-the-math-behind-xgboost/>
 - Evaluation metrics: <https://www.analyticsvidhya.com/blog/2019/08/11-important-model-evaluation-error-metrics/>
- Flask Basics : https://www.youtube.com/watch?v=Ij4I_CvBnt0

Project Structure:

Create the Project folder which contains files as shown below



- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- Dtc_model.pkl is our saved model. Further we will use this model for flask integration.
- Data Folder contains the Dataset used
- The Notebook file contains procedure for building th model.

Milestone 1: Define Problem / Problem Understanding

Activity 1: Specify the business problem

Refer Project Description

Activity 2: Business requirements

A drug classification project can have a variety of business requirements, depending on the specific goals and objectives of the project. Some potential requirements may include:

- **Accurate and up-to-date information:** The project should use the most recent and reliable data to classify drugs, in order to ensure that the information is accurate and relevant to current medical practices.
- **Flexibility:** The classification system should be flexible and able to adapt to new drugs and changing information as it becomes available.
- **Compliance:** The project should comply with all relevant laws and regulations, such as FDA guidelines for classifying drugs.
- **User-friendly interface:** The classification system should be easy to use and understand for both medical professionals and patients.

Activity 3: Literature Survey (Student Will Write)

A literature survey for a drug classification project would involve researching and reviewing existing studies, articles, and other publications on the topic of drug classification. The survey would aim to gather information on current classification systems, their strengths and weaknesses, and any gaps in knowledge that the project could address. The literature survey would also look at the methods and techniques used in previous drug classification projects, and any relevant data or findings that could inform the design and implementation of the current project.

Activity 4: Social or Business Impact.

Social Impact :- Improved patient care: By providing accurate and up-to-date information on drugs, a drug classification project can help healthcare professionals make more informed decisions about treatment options, leading to improved patient care.

Business Model/Impact :- New drug development: By providing information on the properties and interactions of different drugs, a drug classification project can assist in the development of new treatments and therapies.

Milestone 2: Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Activity 1: Collect the dataset

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link: <https://www.kaggle.com/datasets/bunttyshah/auto-insurance-claims-data>

As the dataset is downloaded. Let us read and understand the data properly with the help of some visualisation techniques and some analysing techniques.

Note: There are a number of techniques for understanding the data. But here we have used some of it. In an additional way, you can use multiple techniques.

Activity 1.1: Importing the libraries

Import the necessary libraries as shown in the image. (optional) Here we have used visualisation style as fivethirtyeight.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import warnings
import pickle
from scipy import stats
warnings.filterwarnings('ignore')
plt.style.use('fivethirtyeight')
```

Activity 1.2: Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset with the help of pandas.

In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of the csv file.

- For checking the null values, `df.isna().any()` function is used. To sum those null values we use `.sum()` function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
# Reading the csv file
df=pd.read_csv("insurance_claims.csv")
df.head()
```

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit	insured_zip
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0	466132
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000	468176
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	5000000	430632
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	6000000	608117
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	6000000	610706

5 rows x 10 columns

Activity 2: Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

- Handling missing values
- Handling Outliers

Note: These are the general steps of pre-processing the data before using it for machine learning. Depending on the condition of your dataset, you may or may not have to go through all these steps.

Activity 2.1: Handling missing values

- For checking the null values, `df.isna().any()` function is used. To sum those null values we use `.sum()` function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

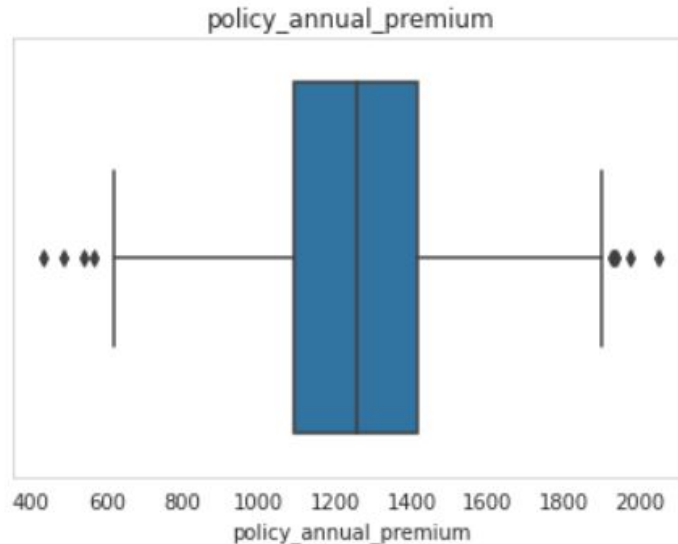
```
# Finding Null values
df.isna().any()
```

months_as_customer	False
age	False
policy_number	False
policy_bind_date	False
policy_state	False
policy_csl	False
policy_deductable	False
policy_annual_premium	False
umbrella_limit	False
insured_zip	False
insured_sex	False
insured_education_level	False
insured_occupation	False

Activity 2.2: Handling Outliers

With the help of boxplot, outliers are visualized. And here we are going to find upper bound and lower bound of policy_annual_premium feature with some mathematical formula.

- From the below diagram, we could visualize that policy_annual_premium feature has outliers. Boxplot from seaborn library is used here.



- To find upper bound we have to multiply IQR (Interquartile range) with 1.5 and add it with 3rd quantile. To find lower bound instead of adding, subtract it with 1st quantile. Take image attached below as your reference.

```
IQR=[]
IQR.append(df['age'].quantile(0.75)-df['age'].quantile(0.25))
IQR.append(df['policy_annual_premium'].quantile(0.75)-df['policy_annual_premium'].quantile(0.25))
IQR.append(df['umbrella_limit'].quantile(0.75)-df['umbrella_limit'].quantile(0.25))
IQR.append(df['total_claim_amount'].quantile(0.75)-df['total_claim_amount'].quantile(0.25))
IQR.append(df['property_claim'].quantile(0.75)-df['property_claim'].quantile(0.25))
IQR
```

```
[12.0, 326.08750000000001, 0.0, 28780.0, 6440.0]
```

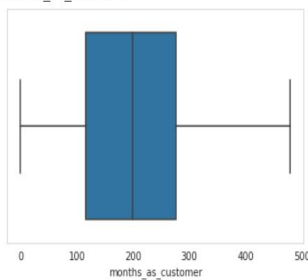
```
[ ] upper=[]
upper.append(df['age'].quantile(0.75)+1.5*(IQR[0]))
upper.append(df['policy_annual_premium'].quantile(0.75)+1.5*(IQR[1]))
upper.append(df['umbrella_limit'].quantile(0.75)+1.5*(IQR[2]))
upper.append(df['total_claim_amount'].quantile(0.75)+1.5*(IQR[3]))
upper.append(df['property_claim'].quantile(0.75)+1.5*(IQR[4]))
upper
```


- To handle the outliers transformation technique is used. Here log transformation is used. We have created a function to visualize the distribution and probability plot of policy_annual_premium feature.

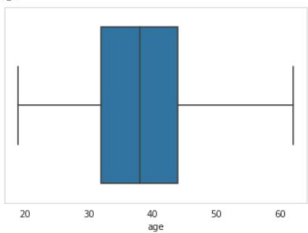
```
[ ] df['age']=np.where(df['age']>62,62,np.where(df['age']<14,14,df['age']))
df['policy_annual_premium']=np.where(df['policy_annual_premium']>upper[1],upper[1],np.where(df['policy_annual_premium']<lower[1],lower[1],df['policy_annual_premium']))
df['umbrella_limit']=np.where(df['umbrella_limit']>upper[2],upper[2],np.where(df['umbrella_limit']<lower[2],lower[2],df['umbrella_limit']))
df['total_claim_amount']=np.where(df['total_claim_amount']>upper[3],upper[3],np.where(df['total_claim_amount']<lower[3],lower[3],df['total_claim_amount']))
df['property_claim']=np.where(df['property_claim']>upper[4],upper[4],np.where(df['property_claim']<lower[4],lower[4],df['property_claim']))
```

```
[ ] for k in df_num_features.columns:
    print(k)
    sns.boxplot(data = df, x = k)
    plt.show()
```

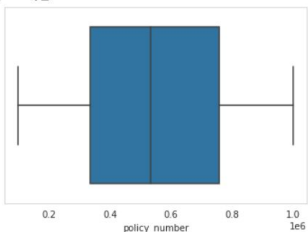
months_as_customer



age



policy_number



Milestone 3: Exploratory Data Analysis

Activity 1: Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

	months_as_customer	age	policy_number	policy_deductable	policy_annual_premium	umbrella_limit
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1.000000e+03
mean	203.954000	38.948000	546238.648000	1136.000000	1256.406150	1.101000e+06
std	115.113174	9.140287	257063.005276	611.864673	244.167395	2.297407e+06
min	0.000000	19.000000	100804.000000	500.000000	433.330000	-1.000000e+06
25%	115.750000	32.000000	335980.250000	500.000000	1089.607500	0.000000e+00
50%	199.500000	38.000000	533135.000000	1000.000000	1257.200000	0.000000e+00
75%	276.250000	44.000000	759099.750000	2000.000000	1415.695000	0.000000e+00
max	479.000000	64.000000	999435.000000	2000.000000	2047.590000	1.000000e+07

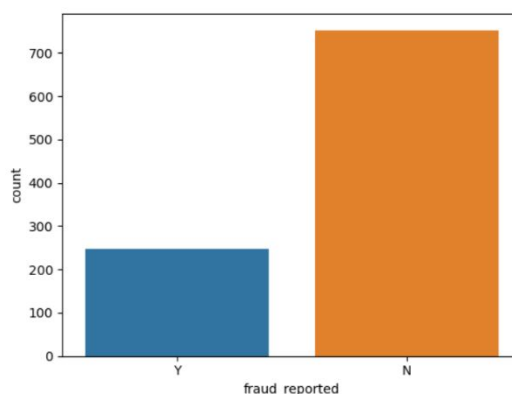
Activity 2: Visual analysis

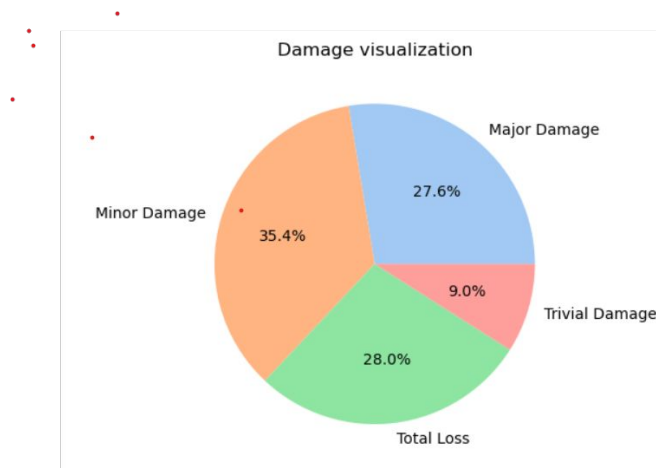
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Activity 2.1: Univariate analysis

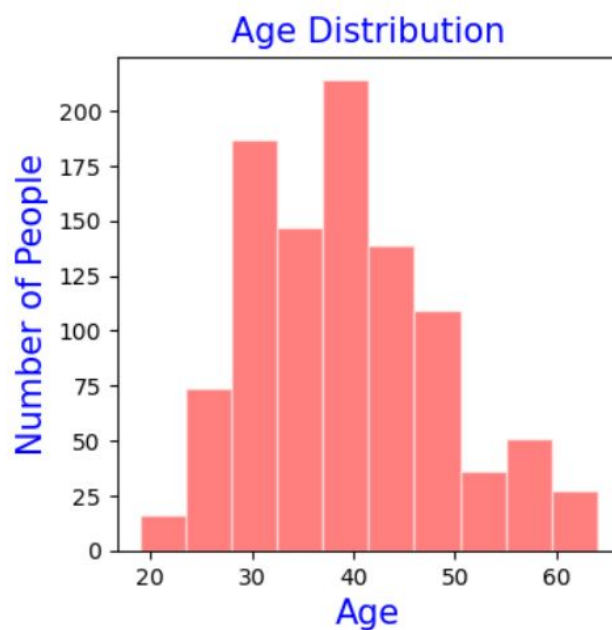
In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as Piechart and countplot.

Seaborn package provides a wonderful function countplot. It is more useful for categorical features. With the help of countplot, we can Number of unique values in the feature. From the countplot we can say that there are only 247 fraud cases reported in 1000 insurance claims.





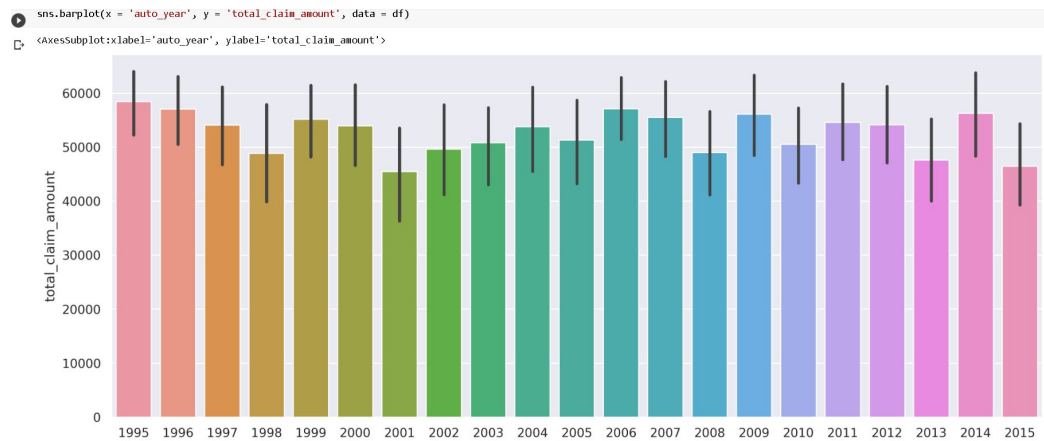
- Pie chart describes the composition of Incident Severity Feature. It describes the 35.4% of Minor damage cases and almost equal composition of Major Damage and Total Loss and only 9% of Trivial Damage.
- From the below histogram we can say that 'age' Feature is almost Normally distributed. Majority of Insurance claims are of age 30 to 50.



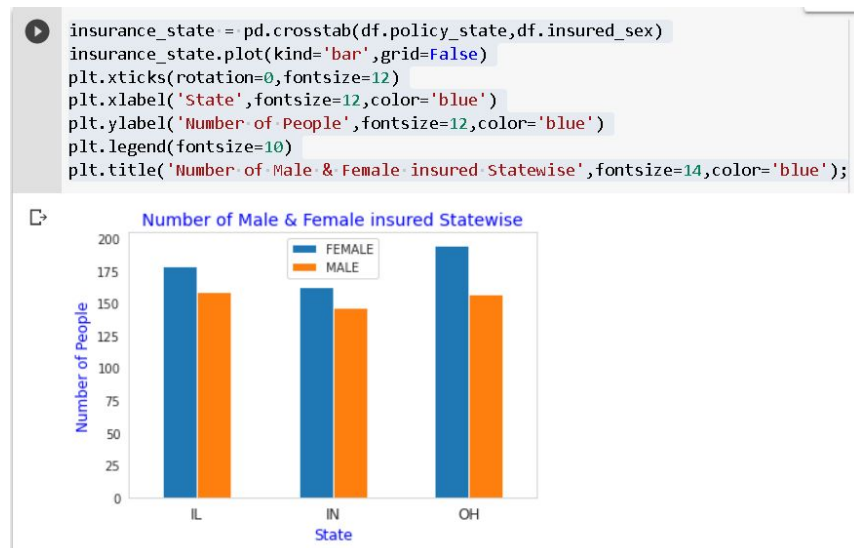
Activity 2.2: Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we can use barplot.

- Barplot is used here. As a 1st parameter we are passing auto_year and as a 2nd parameter we are passing total_claim_amount.
- From the below plot you can understand that distribution of total_claim_amount on auto_year.
- The Most of the claim amounts are of 50,000 to 60,000 from the years 1995 to 2015.



- From the below barplot we can able to figure that Females are more Insured than Males across the different states.



Activity 2.3: Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used heatmap from seaborn package.

- From the below image, we came to a conclusion that there are some features which are highly correlated.
- The correlation between the months_as_customer and age is highly correlated with 0.92 value .
- The features like total_claim_amount, injury_claim, property_claim, vehicle_claim are also highly correlated.
- These Highly correlated features should be dropped.



Encoding the Categorical Features:

- The categorical Features are can't be passed directly to the Machine Learning Model. So we convert them into Numerical data based on their order. This Technique is called Encoding.
- Here we are importing Label Encoder from the Sklearn Library.
- Here we are applying fit_transform to transform the categorical features to numerical features.

```
[36] from sklearn.preprocessing import LabelEncoder
le=LabelEncoder()
for i in daata1.columns:
    if daata1[i].dtype=='O':
        daata1[i] = le.fit_transform(daata1[i])
```

Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the target variable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state.

```
x=daata1.iloc[:,0:30]  
y=daata1.iloc[:,30:]
```

```
[ ] from sklearn.model_selection import train_test_split  
    x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

Handling Imbalanced dataset

- Imbalanced data is a common problem in machine learning and data analysis, where the number of observations in one class is significantly higher or lower than the other class. Handling imbalanced data is important to ensure that the model is not biased towards the majority class and can accurately predict the minority class.
- Here we are using SMOTE Technique.

```
[ ] from imblearn.over_sampling import SMOTE  
    smt=SMOTE()  
    x_train, y_train = smt.fit_resample(x_train, y_train)
```

Scaling

- Scaling is a technique used to transform the values of a dataset to a similar scale to improve the performance of machine learning algorithms. Scaling is important because many machine learning algorithms are sensitive to the scale of the input features.
- Here we are using Standard Scaler.
- This scales the data to have a mean of 0 and a standard deviation of 1. The formula is given by:
$$X_{\text{scaled}} = (X - X_{\text{mean}}) / X_{\text{std}}$$

```
from sklearn.preprocessing import StandardScaler  
std_scaler=StandardScaler()
```

```
x_train = std_scaler.fit_transform(x_train)  
x_train = pd.DataFrame(x_train, columns=x.columns)
```

```
x_test = std_scaler.transform(x_test)  
x_test = pd.DataFrame(x_test, columns=x.columns)
```

Milestone 4: Model Building

Activity 1: Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For this project we are applying three classification algorithms. The best model is saved based on its performance.

Activity 1.1: Decision tree model

First Decision Tree is imported from sklearn Library then DecisionTreeClassifier algorithm is initialised and training data is passed to the model with the .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.

```
from sklearn.tree import DecisionTreeClassifier
dtc=DecisionTreeClassifier()
dtc.fit(X_train,y_train)
y_pred=dtc.predict(X_test)
dtc_train_acc=accuracy_score(y_train,dtc.predict(X_train))
dtc_test_acc=accuracy_score(y_test,y_pred)|
```

Activity 1.2: Random forest model

First Random Forest Model is imported from sklearn Library then RandomForestClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in a new variable. We can find the Train and Test accuracy by X_train and X_test.

```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(criterion='entropy',max_depth=10,max_features='sqrt',min_samples_leaf=1, min_samples_split=3, n_estimators= 140)
rfc.fit(X_train,y_train)
y_pred=rfc.predict(X_test)
rfc_train_acc=100*accuracy_score(y_train,rfc.predict(X_train))
rfc_test_acc=100*accuracy_score(y_test,y_pred)|
```

Activity 1.3: KNN model

KNN Model is imported from sklearn Library then KNeighborsClassifier algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=30)
knn.fit(X_train,y_train)
y_pred=knn.predict(X_test)
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

Activity 1.4: Logistic Regression model

Logistic Regression Model is imported from sklearn Library then Logistic Regression algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix is done.

```
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, classification_report
lg = LogisticRegressionCV(solver='lbfgs', max_iter=5000, cv=10)
lg.fit(X_train, y_train)
print(confusion_matrix(y_test,lrg_pred))
```


Activity 1.5: Naïve Bayes model

Naïve Bayes Model is imported from sklearn Library then Naïve Bayes algorithm is initialised and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. We can find the Train and Test accuracy by X_train and X_test.

```
from sklearn.naive_bayes import CategoricalNB, GaussianNB
gnb=GaussianNB()
model_2=gnb.fit(X_train,y_train)
predict_log=model_2.predict(X_test)
print("Training Accuracy",100*accuracy_score(model_2.predict(X_train),y_train))
print("Testing Accuracy",100*accuracy_score(y_test,predict_log))
```

Activity 1.6: SVM model

SVM Model is imported from sklearn Library then SVM algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, confusion matrix and classification report is done.

```
: from sklearn.svm import SVC

svc = SVC()
svc.fit(X_train, y_train)
y_pred = svc.predict(X_test)

: svc_train_acc = accuracy_score(y_train, svc.predict(X_train))
  svc_test_acc = accuracy_score(y_test, y_pred)
  print(f"Training accuracy of SVC : {svc_train_acc}")
  print(f"Test accuracy of SVC : {svc_test_acc}")
  print(confusion_matrix(y_test,y_pred))
  print(classification_report(y_test,y_pred))
```

Activity 2: Testing the model

Here we have tested with Decision Tree algorithm. You can test with all algorithm. With the help of predict() function.

```
b_dtc.predict([[328,521585,2012,12,250,1000,1406.91,5600,1,100,25,25,50000,0,120,23,56,52,1,123,2,3,1,0,2,1,150000,2,25,2002]])
[: array([0])
```

Milestone 5: Performance Testing & Hyperparameter Tuning

Activity 1: Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for classification tasks including accuracy, precision, recall, support and F1-score.

Activity 1.1: Compare the model

For comparing the above four models, the compareModel function is defined.

```
def comparison(X_test,y_test):  
    print("Logistic Regression: ",100*accuracy_score(y_test, lrg_pred))  
    print("-"*100)  
    print("KNN",100*accuracy_score(y_test,predict_log))  
    print("-"*100)  
    print("SVM",100*svc_train_acc)  
    print("-"*100)  
    print("Naive-Bayes",100*accuracy_score(model_2.predict(X_test),y_test))  
    print("-"*100)  
    print("Decision Tree",100*b_dtc_test_acc)  
    print("-"*100)  
    print("Random Forest",100*b_rfc_test_acc)  
    print("-"*100)
```

```
comparison(X_test,y_test)
```

```
logistic Regression: 66.5
```

```
-----  
KNN 66.5
```

```
-----  
SVM 95.57377049180327
```

```
-----  
Naive-Bayes 66.5
```

```
-----  
Decision Tree 85.0
```

```
-----  
Random Forest 80.0
```

```
# Logistic Regression  
print(confusion_matrix(y_test,lrg_pred))  
print(classification_report(y_test,lrg_pred))
```

```
[[ 91  52]  
 [ 15  42]]
```

	precision	recall	f1-score	support
0	0.86	0.64	0.73	143
1	0.45	0.74	0.56	57
accuracy			0.67	200
macro avg	0.65	0.69	0.64	200
weighted avg	0.74	0.67	0.68	200

```
# SVM
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))
```

```
[[101  42]
 [ 18  39]]
```

	precision	recall	f1-score	support
0	0.85	0.71	0.77	143
1	0.48	0.68	0.57	57
accuracy			0.70	200
macro avg	0.67	0.70	0.67	200
weighted avg	0.74	0.70	0.71	200

```
# Decision Tree
print("Confusion Matrix \n",confusion_matrix(y_test,y_pred),"\n")
print("classification_report \n",classification_report(y_test,y_pred))
```

```
Confusion Matrix
[[101  42]
 [ 18  39]]
```

	precision	recall	f1-score	support
0	0.85	0.71	0.77	143
1	0.48	0.68	0.57	57
accuracy			0.70	200
macro avg	0.67	0.70	0.67	200
weighted avg	0.74	0.70	0.71	200

After calling the function, the results of models are displayed as output. From the above models Decision Tree is performing well.

Activity 2: Comparing model accuracy before & after applying hyperparameter tuning (Hyperparameter tuning is optional. For this project it is not required.)

Evaluating performance of the model From sklearn, cross_val_score is used to evaluate the score of the model. On the parameters, we have given rf (model name), x, y, cv (as 5 folds). Our model is performing well.

Note: To understand cross validation, refer to this [link](#)

```
from sklearn.model_selection import cross_val_score
cv=cross_val_score(b_dtc,X_train,y_train,cv=11)
print(cv)
```

```
[0.8018018  0.82882883 0.86486486 0.85585586 0.90990991 0.89189189
 0.81981982 0.83783784 0.87387387 0.88288288 0.91818182]
```

Milestone 6: Model Deployment

Activity 1: Save the best model

Saving the best model after comparing its performance using different evaluation metrics means selecting the model with the highest performance. This can be useful in avoiding the need to retrain the model every time it is needed and also to be able to use it in the future.

```
import pickle
filename='dtc_model.pkl'
pickle.dump(b_dtc, open(filename, 'wb'))
```

Activity 2: Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the user where he has to enter the values for predictions. The entered values are given to the saved model and prediction is showcased on the UI.

This section has the following tasks

- Building HTML Pages
- Building server-side script
- Run the web application

Activity 2.1: Building Html Page:

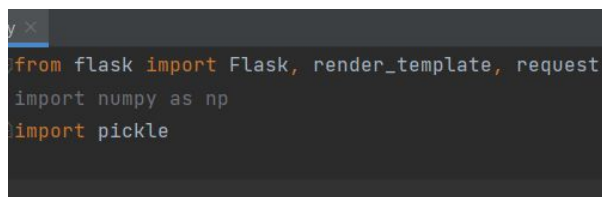
For this project create HTML file namely

- index.html

and save them in the templates folder. Refer this [link](#) for templates.

Activity 2.2: Build Python code:

Import the libraries



```
from flask import Flask, render_template, request
import numpy as np
import pickle
```

Load the saved model. Importing the flask module in the project is mandatory. An object of Flask class is our WSGI application. Flask constructor takes the name of the current module (`_name_`) as argument.

```
model = pickle.load(open("F:\Smart_Internz\Insurance_fraud\Flask\dtc_model.pkl", 'rb'))
scaler=pickle.load(open("F:\Smart_Internz\Insurance_fraud\Flask\Std_Scaler.pkl", 'rb'))
app = Flask(__name__)
```

Render HTML page:

```
@app.route('/')
def welcome():
    return render_template('index.html')
```

Here we will be using a declared constructor to route to the HTML page which we have created earlier.

In the above example, '/' URL is bound with the index.html function. Hence, when the home page of the web server is opened in the browser, the html page will be rendered. Whenever you enter the values from the html page the values can be retrieved using POST Method.

Retrieves the value from UI:

```
@app.route('/predict',methods =['GET','POST'])
def predict():
    months_as_customer = float(request.form["months_as_customer"])
    policy_number =float(request.form['policy_number'])
    policy_bind_date = float(request.form['policy_bind_date'])
    policy_state=float(request.form['policy_state'])
    policy_csl = float(request.form['policy_csl'])
    policy_deductable = float(request.form['policy_deductable'])
    policy_annual_premium= float(request.form['policy_annual_premium'])
    insured_zip= float(request.form['insured_zip'])
    insured_sex = float(request.form['insured_sex'])
    insured_occupation=float(request.form["insured_occupation"])
    insured_hobbies=float(request.form["insured_hobbies"])
    insured_relationship=float(request.form['insured_relationship'])
    capital_gains=float(request.form['capital_gains'])
    capital_loss=float(request.form['capital_loss'])
    incident_date=float(request.form['incident_date'])
    incident_type=float(request.form['incident_type'])
    collision_type=float(request.form['collision_type'])
    incident_severity=float(request.form['incident_severity'])
    authorities_contacted=float(request.form['authorities_contacted'])
    incident_location=float(request.form['incident_location'])
    incident_hour_of_the_day=float(request.form['incident_hour_of_the_day'])
    number_of_vehicles_involved=float(request.form['number_of_vehicles_involved'])
    property_damage=float(request.form['property_damage'])
    bodily_injuries=float(request.form['bodily_injuries'])
    witnesses=float(request.form['witnesses'])
    police_report_available=float(request.form['police_report_available'])
    total_claim_amount=float(request.form['total_claim_amount'])
    auto_make=float(request.form['auto_make'])
    auto_model=float(request.form['auto_model'])
    auto_year=float(request.form['auto_year'])
    total = [[months_as_customer,policy_number, policy_bind_date,policy_state,policy_csl,policy_deductable,policy_annual_premium,insured_zip,insured_
    prediction = model.predict(scaler.transform(total))
    prediction = int(prediction[0])
    if prediction==0:
        return render_template('index.html',predict="Legal Insurance Claim")
    else:
        return render_template('index.html',predict="Fraud Insurance Claim")
```

Here we are routing our app to predict() function. This function retrieves all the values from the HTML page using Post request. That is stored in an array. This array is passed to the model.predict() function. This function returns the prediction. And this prediction value will be rendered to the text that we have mentioned in the submit.html page earlier.

Main Function:

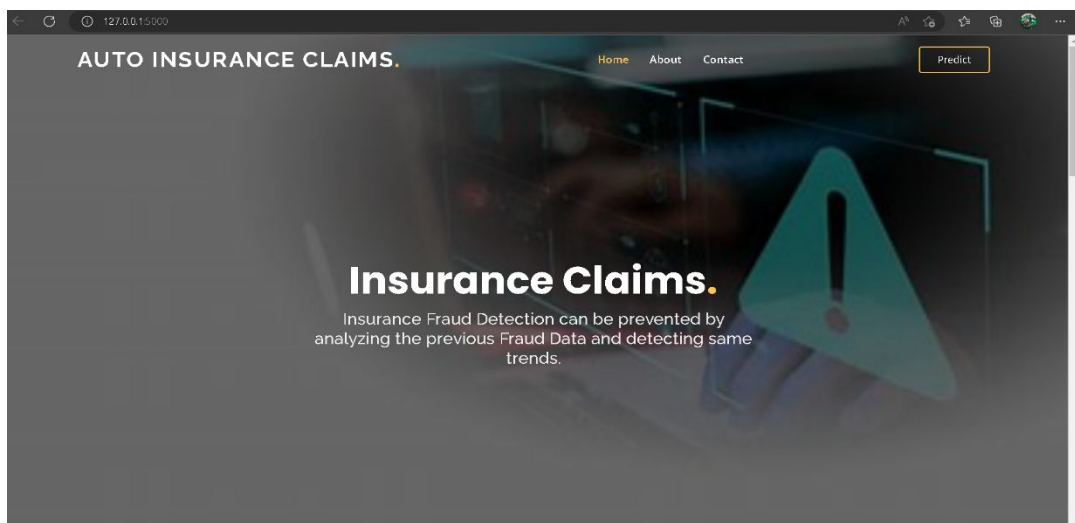
```
if __name__ == '__main__':  
    app.run(debug=True)
```

Activity 2.3: Run the web application

- Open anaconda prompt from the start menu
- Navigate to the folder where your python script is.
- Now type “python app.py” command
- Navigate to the localhost where you can view your web page.
- Click on the predict button from the top left corner, enter the inputs, click on the submit button, and see the result/prediction on the web.

```
base) D:\TheSmartBridge\Projects\2. DrugClassification\Drug c  
* Serving Flask app "app" (lazy loading)  
* Environment: production  
  WARNING: This is a development server. Do not use it in a p  
  Use a production WSGI server instead.  
* Debug mode: off  
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now, Go the web browser and write the localhost url (http://127.0.0.1:5000) to get the below result





- Data Collection and Preprocessing**
The first step involves collecting Insurance data and preprocessing it to handle missing values, Handling Categorical data and outliers, and inconsistencies.
- Feature Engineering and Model Selection**
The second step involves selecting relevant features and transforming them into a format suitable for building a machine learning model, as well as selecting an appropriate algorithm such as KNN, Naive Bayes, Decision Trees, Random Forest, or SVM.
- Model Training and Evaluation**
The third step involves training the selected model using the preprocessed data and evaluating its performance using metrics such as accuracy, precision, recall, and F1-score.
- Model Deployment**
The final step involves deploying the model in a real-world scenario to classify Fraud claims in real-time, so that No Frauds can be happened in Insurance Claims.



Insurance Fraud Detection

months_as_customer: 328	policy_number 521585	policy_bind_date 2012	policy_state 12
policy_csl 250	policy_deductable 1000	policy_annual_premium 1406.91	insured_zip 5600
Insured_sex 1	Insured_occupation 100	Insured_hobbies 25	Insured_relationship 25
capital-gains 50000	capital-loss 0	incident_date 120	incident_type 23
collision_type 56	incident_severity 52	authorities_contacted 01	incident_location -0.830899
incident_hour_of_the_day 2	number_of_vehicles_involved 3	property_damage 1	bodily_injuries 0
witnesses 2	police_report_available 1	total_claim_amount 150000	auto_make 2
auto_model 25	auto_year 2002		

submit

The Insurance Fraud Detection Status:



Insurance Fraud Detection

months_as_customer:	policy_number	policy_bind_date	policy_state
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
policy_csl	policy_deductable	policy_annual_premium	insured_zip
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
insured_sex	insured_occupation	insured_hobbies	insured_relationship
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
capital-gains	capital-loss	incident_date	incident_type
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
collision_type	incident_severity	authorities_contacted	incident_location
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
incident_hour_of_the_day	number_of_vehicles_involved	property_damage	bodily_injuries
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
witnesses	police_report_available	total_claim_amount	auto_make
<input type="text"/>	<input type="text"/>	<input type="text"/>	<input type="text"/>
auto_model	auto_year		
<input type="text"/>	<input type="text"/>		

submit

The Insurance Fraud Detection Status: **Legal Insurance Claim**



Milestone 7: Project Demonstration & Documentation

Below mentioned deliverables to be submitted along with other deliverables

Activity 1:- Record explanation Video for project end to end solution

Activity 2:- Project Documentation-Step by step project development procedure

Create document as per the template provided