Fundamentals of Data Mining

Final Report

Group Members:

Student Register Number	Name
IT20205256	Kishan R.
IT20202668	Heisapirashoban N.
IT20205188	Senarathne R.S.A.M.N. T
IT20201678	Sandarangi R.M. N
IT20202736	Withanage D.U.I.W

Video Reference Link: https://mysliit-my.sharepoint.com/:v:/g/personal/it20202668_my_sliit_lk/Efg5cyo11GtMluLqf56zq6ABNVyLF-KqQydGPHnrQvcGvA?e=EIzdFP

Contents

1. Background

The purpose of doing this project is to find a real-world problem and to find a technical solution for that by creating a model with the help of Data Science and Machine-Learning concepts.

The dataset we have chosen is related to vehicle insurance claim fraud detection and the dataset contains 15420 data points with 14497 non-fraudulent transactions and 923 fraudulent transactions happened in 1994 to 1996. Our goal is to build a model that can detect vehicle insurance fraud with the help of this dataset.

Identifying the Problem:

Conspiring to create fraudulent or inflated claims about property damage or personal injuries because of an accident is known as vehicle insurance fraud. The use of phantom passengers, where individuals who were not even present at the accident scene claim to have suffered severe injuries, staged accidents, where fraudsters purposefully "arrange" for accidents to occur, and false personal injury claims, where personal injuries are grossly exaggerated are a few frequent examples. As frauds are unethical and are losses to the companies, we need a way to cut losses for the insurance company as less losses equates to more earnings.

Solution:

We have planned to build a classification model to solve this problem. The goal of classification is to utilize input training data for the purpose of predicting the likelihood or probability that the data that follows will fall into one of the predetermined categories.

Claim fraud varies on Base policy, address change, age of policy holder, driver rating, vehicle price, etc. Through the model which we are designing it can simply enter the details and predict that it is a fraud or not. So, when claim ensuring person enter the details he could come to a decision about fraudulent status.

First, we will be applying necessary pre-processing steps to dataset as mentioned in coming parts. After that model validation will be done. Next, we have planned to build the front-end part of this application where users can apply data to the model.

Dataset Link: https://www.kaggle.com/datasets/shivamb/vehicle-claim-fraud-detection

Git Hub Link: https://github.com/IT20205256/FDM_Group03.git

2. Identify the problem with business goals

The fraudulent claims are interconnected with false details which cannot be found by the claim provider easily. These fraudsters arrange these accidents to occur, and they use phantom passengers who were not even at the scene.

So, to predict these fraudsters we have planned to evolve machine learning which will be an easy and pretty accuracy process.

3. Description of the Dataset

Column Name	Description	Data Type
MaritalStatus	Single = 0 Married = 1 Divorced = 2 Widow = 3	Int
Fault	Policy Holder = 0 Third Party = 1	Boolean
PolicyType	Sedan - All Perils = 0 Sedan - Collision = 1 Sedan - Liability = 2 Sport - All Perils = 3 Sport - Collision = 4 Sport - Liability = 5 Utility - All Perils = 6 Utility - Collision = 7 Utility - Liability = 8	Int
VehicleCategory	Sedan = 1 Sport = 2 Utility = 3	Int
VehiclePrice	20000 to 29000 = 0 30000 to 39000 = 1 40000 to 59000 = 2 60000 to 69000 = 3 more than 69000 = 4 less than 20000 = 5	Int
DriverRating	Driver rating	Int

Days_Policy_Accident	1 to 7 = 0 8 to 15 = 1 15 to 30 = 2 more than 30 = 3 none =4	Int
Days_Policy_Claim	8 to 15 = 0 15 to 30 = 1 more than 30 = 2 none = 3	Int
PastNumberOfClaims	1 = 1 2 to 4 = 2 more than 4 = 3 none = 4	Int
AgeOfPolicyHolder	16 to 17 = 1 18 to 20 = 2 21 to 25 = 3 26 to 30 = 4 31 to 35 = 5 36 to 40 = 6 41 to 50 = 7 51 to 65 = 8 over 65 = 9	Int
PoliceReportFiled	No = 0 Yes = 1	Boolean
WitnessPresent	No = 0 Yes = 1	Boolean
AgentType	External = 0 Internal = 1	Boolean
AddressChange_Claim	no change = 0 1 year = 1 2 to 3 years = 2 4 to 8 years = 3 under 6 months = 4	Int
BasePolicy	All Perils = 0 Collision = 1 Liability = 2	String

4. Data Identification

- ✓ First, we analyzed the data using below method for analyze first five rows and last five rows in the data set.

```
In [16]: data
```

Out[16]:

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	...	AgeOfVeh
0	Dec	5	Wednesday	Honda	Urban	Tuesday	Jan	1	Female	Single	...	3 ye
1	Jan	3	Wednesday	Honda	Urban	Monday	Jan	4	Male	Single	...	6 ye
2	Oct	5	Friday	Honda	Urban	Thursday	Nov	2	Male	Married	...	7 ye
3	Jun	2	Saturday	Toyota	Rural	Friday	Jul	1	Male	Married	...	more tha
4	Jan	5	Monday	Honda	Urban	Tuesday	Feb	2	Female	Single	...	5 ye
...	...	...	...	...	...	...	...	...	...	...	...	
15415	Nov	4	Friday	Toyota	Urban	Tuesday	Nov	5	Male	Married	...	6 ye
15416	Nov	5	Thursday	Pontiac	Urban	Friday	Dec	1	Male	Married	...	6 ye
15417	Nov	5	Thursday	Toyota	Rural	Friday	Dec	1	Male	Single	...	5 ye
15418	Dec	1	Monday	Toyota	Urban	Thursday	Dec	2	Female	Married	...	2 ye
15419	Dec	2	Wednesday	Toyota	Urban	Thursday	Dec	3	Male	Single	...	5 ye

15420 rows × 33 columns

- ✓ By using describe() function we were able to get a statistical analysis of the data set.

```
In [17]: data.describe()
```

Out[17]:

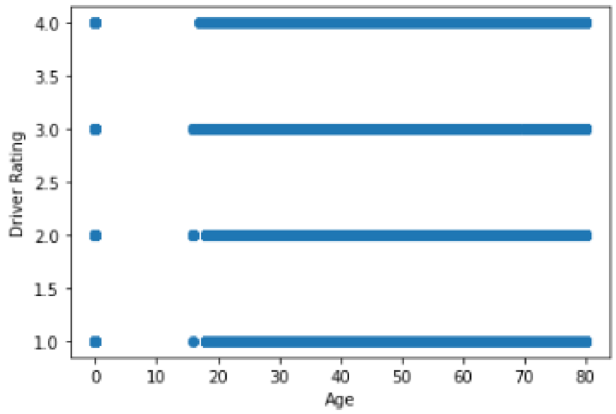
	WeekOfMonth	WeekOfMonthClaimed	Age	FraudFound_P	PolicyNumber	RepNumber	Deductible	DriverRating	Year
count	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000
mean	2.788586	2.693969	39.855707	0.059857	7710.500000	8.483268	407.704280	2.487808	1994.866472
std	1.287585	1.259115	13.492377	0.237230	4451.514911	4.599948	43.950998	1.119453	0.803313
min	1.000000	1.000000	0.000000	0.000000	1.000000	1.000000	300.000000	1.000000	1994.000000
25%	2.000000	2.000000	31.000000	0.000000	3855.750000	5.000000	400.000000	1.000000	1994.000000
50%	3.000000	3.000000	38.000000	0.000000	7710.500000	8.000000	400.000000	2.000000	1995.000000
75%	4.000000	4.000000	48.000000	0.000000	11565.250000	12.000000	400.000000	3.000000	1996.000000
max	5.000000	5.000000	80.000000	1.000000	15420.000000	16.000000	700.000000	4.000000	1996.000000

✓ By using dtypes() method we got a clear idea about data types of each feature in the data set.

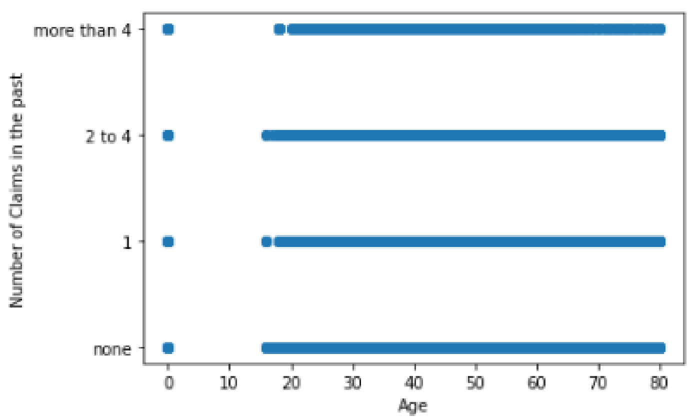
```
In [23]: data.dtypes

Out[23]: Month                   object
         WeekOfMonth              int64
         DayOfWeek               object
         Make                    object
         AccidentArea            object
         DayOfWeekClaimed        object
         MonthClaimed            object
         WeekOfMonthClaimed       int64
         Sex                     object
         MaritalStatus           object
         Age                     object
         Fault                   object
         PolicyType              object
         VehicleCategory         object
         VehiclePrice            object
         FraudFound_P             int64
         PolicyNumber             int64
         RepNumber                int64
         Deductible               int64
         DriverRating             int64
         Days_Policy_Accident    object
         Days_Policy_Claim       object
         PastNumberOfClaims      object
         AgeOfVehicle            object
         AgeOfPolicyHolder       object
         PoliceReportFiled       object
         WitnessPresent          object
         AgentType               object
         NumberOfSuppliments     object
         AddressChange_Claim     object
         NumberOfCars            object
         Year                     int64
         BasePolicy              object
         duplicated                bool
         dtype: object
```

Scatter Plot

In [7]:
```
X_var = data["Age"]
Y_var = data["DriverRating"]
plt.scatter(X_var, Y_var)
plt.ylabel("Driver Rating")
plt.xlabel("Age")
plt.show()
```

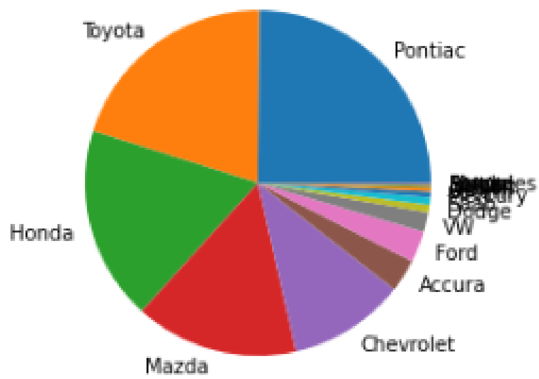


In [8]:
```
X_var1 = data["Age"]
Y_var1 = data["PastNumberOfClaims"]
plt.scatter(X_var1, Y_var1)
plt.ylabel("Number of Claims in the past")
plt.xlabel("Age")
plt.show()
```

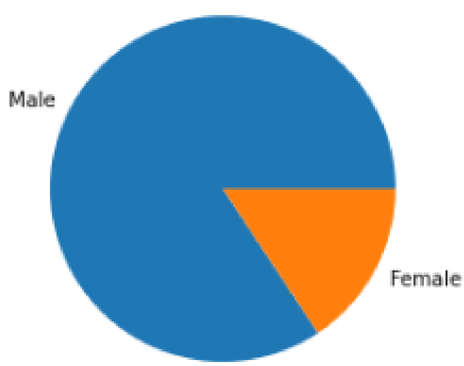


Pie Chart

```
In [10]: plt.pie(Car, labels= Names)
         plt.show()
```

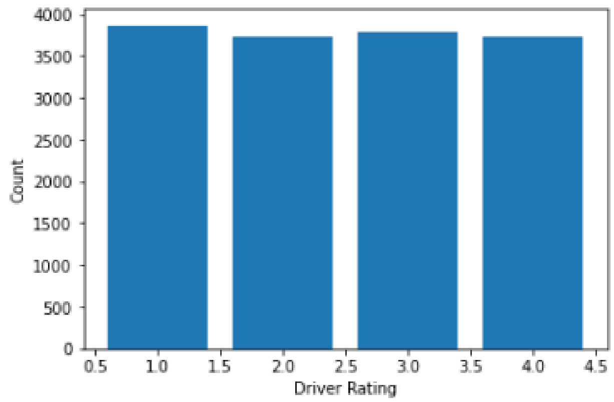



```
In [11]: Sex_count = np.array(data.Sex.value_counts())
         Sex = ["Male", "Female"]
         plt.pie(Sex_count, labels= Sex)
         plt.show()
```

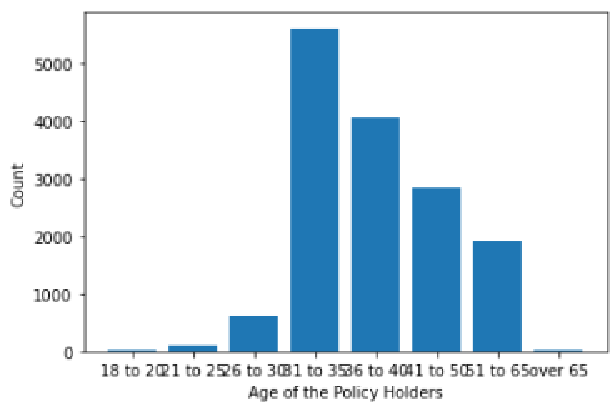


Bar Chart

In [12]:
```
xAxis = np.array(data["DriverRating"].unique())
yAxis = [3866, 3725, 3788, 3721]
plt.bar(xAxis,yAxis)
plt.xlabel('Driver Rating')
plt.ylabel('Count')
plt.show()
```

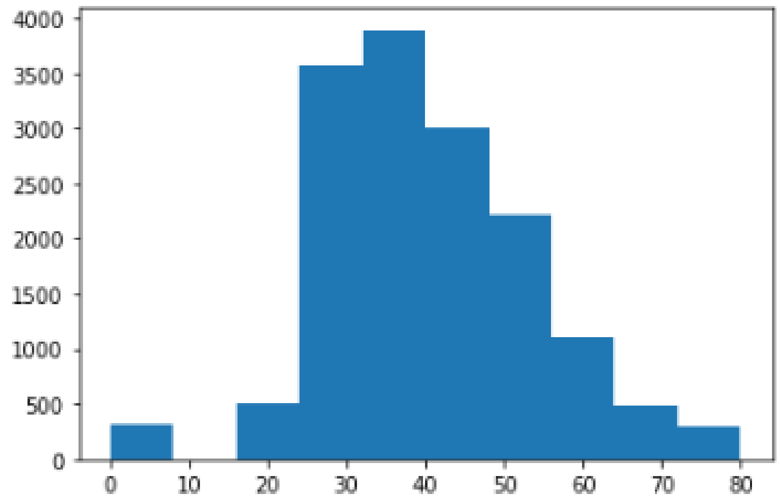


In [13]:
```
xAxis1 = ["18 to 20", "21 to 25", "26 to 30", "31 to 35", "36 to 40", "41 to 50", "51 to 65", "over 65"]
yAxis1 = [15, 108, 613, 5593, 4043, 2828, 1900, 32]
plt.bar(xAxis1,yAxis1)
plt.xlabel('Age of the Policy Holders')
plt.ylabel('Count')
plt.show()
```



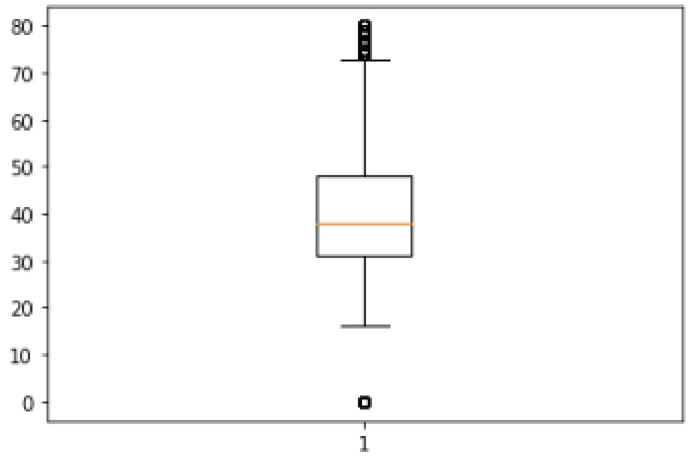
Histogram

```
In [14]: plt.hist(data["Age"], bins=10)
         plt.show()
```

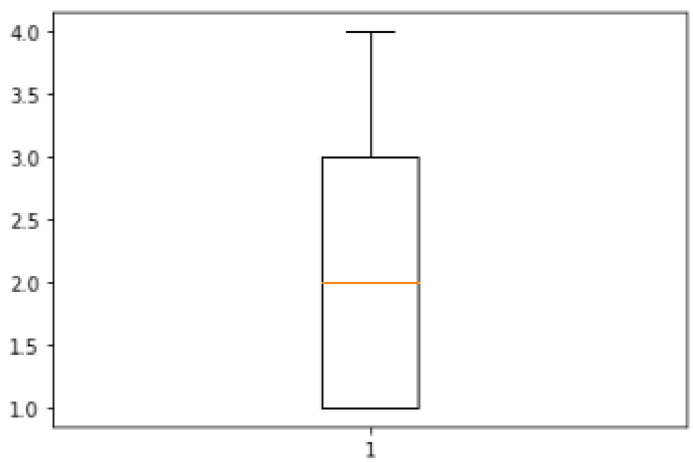


Box Plot

```
In [15]: plt.boxplot(data["Age"])
         plt.show()
```




```
In [16]: plt.boxplot(data["DriverRating"])
         plt.show()
```



5. Data Preprocessing

In this step, basically data cleansing and data pre-processing is being done. Data cleansing is done by handing missing values and eliminating outliers. In this process outliers will be converted in to null values and all null values / missing values will be removed using **.drop()** command. In data pre-processing, check for unique values and if there are unique categorical values, those unique categorical values will be converted in to numeric values using **.loc()** command.

<u>Before pre-processing</u>

```
In [16]: data
```

Out[16]:

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	...	AgeOfVeh
0	Dec	5	Wednesday	Honda	Urban	Tuesday	Jan	1	Female	Single	...	3 ye
1	Jan	3	Wednesday	Honda	Urban	Monday	Jan	4	Male	Single	...	6 ye
2	Oct	5	Friday	Honda	Urban	Thursday	Nov	2	Male	Married	...	7 ye
3	Jun	2	Saturday	Toyota	Rural	Friday	Jul	1	Male	Married	...	more tha
4	Jan	5	Monday	Honda	Urban	Tuesday	Feb	2	Female	Single	...	5 ye
...	...	...	...	...	...	...	...	...	...	...	...	
15415	Nov	4	Friday	Toyota	Urban	Tuesday	Nov	5	Male	Married	...	6 ye
15416	Nov	5	Thursday	Pontiac	Urban	Friday	Dec	1	Male	Married	...	6 ye
15417	Nov	5	Thursday	Toyota	Rural	Friday	Dec	1	Male	Single	...	5 ye
15418	Dec	1	Monday	Toyota	Urban	Thursday	Dec	2	Female	Married	...	2 ye
15419	Dec	2	Wednesday	Toyota	Urban	Thursday	Dec	3	Male	Single	...	5 ye

15420 rows × 33 columns

Check for missing values

In this dataset null values were shown by “0”. Therefore, convert the “0” values as “NaN”.

```
In [10]: data.loc[data["DayOfWeekClaimed"] == "0", "DayOfWeekClaimed"] = "NaN"
         data.loc[data["MonthClaimed"] == "0", "MonthClaimed"] = "NaN"
         data.loc[data["Age"] == "0", "Age"] = "NaN"
```

Before converting “0” values as “NaN”

```
In [9]: display(data.iloc[7])

Month                             Nov
WeekOfMonth                         1
DayOfWeek                      Friday
Make                            Honda
AccidentArea                    Urban
DayOfWeekClaimed              Tuesday
MonthClaimed                      Mar
WeekOfMonthClaimed                  4
Sex                              Male
MaritalStatus                  Single
Age                                 0
Fault                   Policy Holder
PolicyType          Sport - Collision
VehicleCategory                 Sport
VehiclePrice          more than 69000
FraudFound_P                        0
PolicyNumber                        8
RepNumber                           1
Deductible                        400
DriverRating                        4
Days_Policy_Accident     more than 30
Days_Policy_Claim        more than 30
PastNumberOfClaims                  1
AgeOfVehicle                      new
AgeOfPolicyHolder            16 to 17
PoliceReportFiled                  No
WitnessPresent                     No
AgentType                    External
NumberOfSuppliments              none
AddressChange_Claim         no change
NumberOfCars                1 vehicle
Year                             1994
BasePolicy                  Collision
Name: 7, dtype: object
```

After converting "0" values as "NaN"

```
In [11]: display(data.iloc[7])

Month                                    Nov
WeekOfMonth                                1
DayOfWeek                             Friday
Make                                   Honda
AccidentArea                           Urban
DayOfWeekClaimed                     Tuesday
MonthClaimed                             Mar
WeekOfMonthClaimed                         4
Sex                                     Male
MaritalStatus                         Single
Age                                      NaN
Fault                          Policy Holder
PolicyType                  Sport - Collision
VehicleCategory                        Sport
VehiclePrice                 more than 69000
FraudFound_P                               0
PolicyNumber                               8
RepNumber                                  1
Deductible                               400
DriverRating                               4
Days_Policy_Accident            more than 30
Days_Policy_Claim               more than 30
PastNumberOfClaims                         1
AgeOfVehicle                             new
AgeOfPolicyHolder                   16 to 17
PoliceReportFiled                         No
WitnessPresent                            No
AgentType                           External
NumberOfSuppliments                     none
AddressChange_Claim                no change
NumberOfCars                       1 vehicle
Year                                    1994
BasePolicy                         Collision
Name: 7, dtype: object
```

Drop the missing values

```
In [15]: data.drop(data.index[data['DayOfWeekClaimed'] == "NaN"], inplace=True)
         data.drop(data.index[data['MonthClaimed'] == "NaN"], inplace=True)
         data.drop(data.index[data['Age'] == "NaN"], inplace=True)
```

Check for unique Values

In [26]:
```
data.Month.value_counts()
```

Out[26]:
```
Jan    1411
May    1367
Mar    1360
Jun    1321
Oct    1305
Dec    1285
Apr    1280
Feb    1266
Jul    1257
Sep    1240
Nov    1201
Aug    1127
Name: Month, dtype: int64
```

In [26]:
```
data.Make.value_counts()
```

Out[26]:
```
Pontiac      3837
Toyota       3121
Honda        2482
Mazda        2354
Chevrolet    1681
Accura        472
Ford          450
VW            283
Dodge         108
Saab          108
Mercury        83
Saturn         58
Nisson         30
BMW            15
Jaguar          6
Porche          5
Mecedes         4
Ferrari         2
Lexus           1
Name: Make, dtype: int64
```

In [27]:
```
data.Sex.value_counts()
```

Out[27]:
```
Male      13000
Female     2420
Name: Sex, dtype: int64
```

Check for duplicate Values

We have checked the rows whether they have duplicated values with each other using the duplicated() command. Then we created a new column in the data frame to check the Boolean results. Finally, we couldn't retrieve all of the records at once, so we have used the unique() key value to identify the unique values in that columns but there weren't any.

In [22]:
```
data["duplicated"]= data.duplicated()
```

In [23]:
```
data
```

Out[23]:

...	AgeOfPolicyHolder	PoliceReportFiled	WitnessPresent	AgentType	NumberOfSuppliments	AddressChange_Claim	NumberOfCars	Year	BasePolicy	duplicated
...	26 to 30	No	No	External	none	1 year	3 to 4	1994	Liability	False
...	31 to 35	Yes	No	External	none	no change	1 vehicle	1994	Collision	False
...	41 to 50	No	No	External	none	no change	1 vehicle	1994	Collision	False
...	51 to 65	Yes	No	External	more than 5	no change	1 vehicle	1994	Liability	False
...	31 to 35	No	No	External	none	no change	1 vehicle	1994	Collision	False
...	...	...	...	...	...	...	...	...	...	...
...	31 to 35	No	No	External	none	no change	1 vehicle	1996	Collision	False
...	31 to 35	No	No	External	more than 5	no change	3 to 4	1996	Liability	False
...	26 to 30	No	No	External	1 to 2	no change	1 vehicle	1996	Collision	False
...	31 to 35	No	No	External	more than 5	no change	1 vehicle	1996	All Perils	False
...	26 to 30	No	No	External	1 to 2	no change	1 vehicle	1996	Collision	False

In [24]:
```
data["duplicated"].unique()
```

Out[24]:
```
array([False])
```

Categorical Values convert into Numerical Values

```
In [25]: #Function to convert Marital status into Numerical
         def convertMarital(str):
          if str=="Single":
             return 0
          elif str=="Married":
             return 1
          elif str=="Divorced":
             return 2
          else:
                return 3
         data["MaritalStatus"]= data["MaritalStatus"].apply(convertMarital)
```

```
In [26]: #Function to convert Policy Type into Numerical
         def convertPolicyType(str):
          if str=="Sedan - All Perils":
             return 0
          elif str=="Sedan - Collision":
             return 1
          elif str=="Sedan - Liability":
             return 2
          elif str=="Sport - All Perils":
             return 3
          elif str=="Sport - Collision":
             return 4
          elif str=="Sport - Liability":
             return 5
          elif str=="Utility - All Perils":
             return 6
          elif str=="Utility - Collision":
             return 7
          else:
                return 8
         data["PolicyType"]= data["PolicyType"].apply(convertPolicyType)
```

```
In [27]: #Function to convert Vehicle Category into Numerical
         def convertVehicleCat(str):
          if str=="Sedan":
             return 1
          elif str=="Sport":
             return 2
          else:
                return 3
         data["VehicleCategory"]= data["VehicleCategory"].apply(convertVehicleCat)
```

```
In [28]: #Function to convert Vehicle Price into Numerical
         def convertVehiclePrice(str):
          if str=="20000 to 29000":
             return 0
          elif str=="30000 to 39000":
             return 1
          elif str=="40000 to 59000":
             return 2
          elif str=="60000 to 69000":
             return 3
          elif str=="more than 69000":
             return 4
          else:
                return 5
         data["VehiclePrice"]= data["VehiclePrice"].apply(convertVehiclePrice)
```

In [29]:
```
#Function to convert Past Claims into Numerical
def convertPastClaims(str):
 if str=="1":
    return 1
 elif str=="2 to 4":
    return 2
 elif str=="more than 4":
    return 3
 else:
       return 4
data["PastNumberOfClaims"]= data["PastNumberOfClaims"].apply(convertPastClaims)
```

In [30]:
```
#Function to convert Age of Policy Holder into Numerical
def convertPolicyHolderAge(str):
 if str=="18 to 20":
    return 1
 elif str=="21 to 25":
    return 2
 elif str=="26 to 30":
    return 3
 elif str=="31 to 35":
    return 4
 elif str=="36 to 40":
    return 5
 elif str=="41 to 50":
    return 6
 elif str=="51 to 65":
    return 7
 else:
       return 8
data["AgeOfPolicyHolder"]= data["AgeOfPolicyHolder"].apply(convertPolicyHolderAge)
```

In [31]:
```
#Function to convert Police Report Filed into Numerical
def convertReport(str):
 if str=="No":
    return 0
 else:
       return 1
data["PoliceReportFiled"]= data["PoliceReportFiled"].apply(convertReport)
```

In [32]:
```
#Function to convert Witness Present into Numerical
def convertWitness(str):
 if str=="No":
    return 0
 else:
       return 1
data["WitnessPresent"]= data["WitnessPresent"].apply(convertWitness)
```

In [33]:
```
#Function to convert Agent Type into Numerical
def convertAgentType(str):
 if str=="External":
    return 0
 else:
       return 1
data["AgentType"]= data["AgentType"].apply(convertAgentType)
```

In [34]:
```
#Function to convert Address Change into Numerical
def convertAddressChange(str):
 if str=="no change":
    return 0
 elif str=="1 year":
    return 1
 elif str=="2 to 3 years":
    return 2
 elif str=="4 to 8 years":
    return 3
 else:
       return 4
data["AddressChange_Claim"]= data["AddressChange_Claim"].apply(convertAddressChange)
```

In [35]:
```
#Function to convert Base Policy into Numerical
def convertBasePolicy(str):
 if str=="All Perils":
    return 0
 elif str=="Collision":
    return 1
 else:
    return 2
data["BasePolicy"]= data["BasePolicy"].apply(convertBasePolicy)
```

In [36]:
```
#Function to convert Fault into Numerical
def convertFault(str):
 if str=="Policy Holder":
    return 0
 else:
       return 1
data["Fault"]= data["Fault"].apply(convertFault)
```

In [37]:
```
#Function to convert Days Policy Acc into Numerical
def convertPolicyAcc(str):
 if str=="1 to 7":
    return 0
 elif str=="8 to 15":
    return 1
 elif str=="15 to 30":
    return 2
 elif str=="more than 30":
    return 3
 else:
       return 4
data["Days_Policy_Accident"]= data["Days_Policy_Accident"].apply(convertPolicyAcc)
```

In [38]:
```
#Function to convert Days Policy Claim into Numerical
def convertPolicyClaim(str):
 if str=="8 to 15":
    return 0
 elif str=="15 to 30":
    return 1
 elif str=="more than 30":
    return 2
 else:
       return 3
data["Days_Policy_Claim"]= data["Days_Policy_Claim"].apply(convertPolicyClaim)
```

In [39]:
```
data
```

Out[39]:

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	...	AgeOfPoli
0	Dec	5	Wednesday	Honda	Urban	Tuesday	Jan	1	Female	0	...	
1	Jan	3	Wednesday	Honda	Urban	Monday	Jan	4	Male	0	...	
2	Oct	5	Friday	Honda	Urban	Thursday	Nov	2	Male	1	...	
3	Jun	2	Saturday	Toyota	Rural	Friday	Jul	1	Male	1	...	
4	Jan	5	Monday	Honda	Urban	Tuesday	Feb	2	Female	0	...	
...	...	...	...	...	...	...	...	...	...	...	...	
15415	Nov	4	Friday	Toyota	Urban	Tuesday	Nov	5	Male	1	...	
15416	Nov	5	Thursday	Pontiac	Urban	Friday	Dec	1	Male	1	...	
15417	Nov	5	Thursday	Toyota	Rural	Friday	Dec	1	Male	0	...	
15418	Dec	1	Monday	Toyota	Urban	Thursday	Dec	2	Female	1	...	
15419	Dec	2	Wednesday	Toyota	Urban	Thursday	Dec	3	Male	0	...	

15100 rows × 34 columns

Out[39]:

...	AgeOfPolicyHolder	PoliceReportFiled	WitnessPresent	AgentType	NumberOfSuppliments	AddressChange_Claim	NumberOfCars	Year	BasePolicy	duplicated
...	3	0	0	0	none	1	3 to 4	1994	2	False
...	4	1	0	0	none	0	1 vehicle	1994	1	False
...	6	0	0	0	none	0	1 vehicle	1994	1	False
...	7	1	0	0	more than 5	0	1 vehicle	1994	2	False
...	4	0	0	0	none	0	1 vehicle	1994	1	False
...	...	...	...	...	...	...	...	...	...	...
...	4	0	0	0	none	0	1 vehicle	1996	1	False
...	4	0	0	0	more than 5	0	3 to 4	1996	2	False
...	3	0	0	0	1 to 2	0	1 vehicle	1996	1	False
...	4	0	0	0	more than 5	0	1 vehicle	1996	0	False
...	3	0	0	0	1 to 2	0	1 vehicle	1996	1	False

6. Partitioning

- ✓ Before feeding the data into the model, the data has to split into training and testing sets.
- ✓ So, we have used the correlation method to identify the relationship between variables.
- ✓ Then, we have selected the feature and class variables to divide them into train and test.

In [40]:
```
# Splitting the data
```

In [41]:
```
data.corr()
```

Out[41]:

	WeekOfMonth	WeekOfMonthClaimed	MaritalStatus	Fault	PolicyType	VehicleCategory	VehiclePrice	FraudFound_P	PolicyNumber
WeekOfMonth	1.000000	0.277037	-0.020268	0.023900	-0.012488	-0.008754	-0.006957	-0.011276	-0.009808
WeekOfMonthClaimed	0.277037	1.000000	0.003971	-0.006271	0.001143	0.009104	-0.002249	-0.005881	0.011473
MaritalStatus	-0.020268	0.003971	1.000000	-0.004785	-0.016395	-0.014205	0.037796	0.006610	0.010011
Fault	0.023900	-0.006271	-0.004785	1.000000	-0.152964	-0.185389	-0.005088	-0.130917	0.010360
PolicyType	-0.012488	0.001143	-0.016395	-0.152964	1.000000	0.861760	0.156963	-0.053511	-0.010415
VehicleCategory	-0.008754	0.009104	-0.014205	-0.185389	0.861760	1.000000	0.100529	-0.096962	-0.004423
VehiclePrice	-0.006957	-0.002249	0.037796	-0.005088	0.156963	0.100529	1.000000	0.059835	-0.016828
FraudFound_P	-0.011276	-0.005881	0.006610	-0.130917	-0.053511	-0.096962	0.059835	1.000000	-0.012256
PolicyNumber	-0.009808	0.011473	0.010011	0.010360	-0.010415	-0.004423	-0.016828	-0.012256	1.000000
RepNumber	0.005757	0.008018	0.007014	-0.004924	0.002843	0.005896	0.003276	-0.006831	0.010359
Deductible	-0.004131	0.005257	0.031474	-0.003550	0.011249	0.020167	0.002415	0.018201	0.001292
DriverRating	-0.016168	0.002022	0.010869	-0.010607	0.002411	0.005315	0.007254	0.006397	-0.011957
Days_Policy_Accident	-0.006713	0.000725	-0.002253	-0.013681	-0.007192	0.001244	0.002726	0.002648	-0.000379
Days_Policy_Claim	-0.018990	0.003856	-0.007807	-0.021743	0.003669	0.015060	-0.004490	-0.018160	-0.002632
PastNumberOfClaims	-0.007468	-0.002737	-0.002526	0.021107	-0.020742	-0.036014	0.055917	0.019060	0.002696
AgeOfPolicyHolder	-0.011362	0.000326	0.414294	-0.015971	-0.005723	0.030478	0.073759	-0.024090	0.018193
PoliceReportFiled	0.014254	0.024021	-0.011857	0.027911	-0.022477	-0.038740	0.001524	-0.016677	0.023830
WitnessPresent	0.012206	0.009821	-0.007938	0.060782	-0.004444	-0.020751	0.005308	-0.007589	-0.016881
AgentType	0.006634	-0.011200	-0.004417	0.004830	0.039611	0.035498	-0.010503	-0.022859	0.018835
AddressChange_Claim	-0.003655	0.010620	-0.006856	-0.008335	0.004291	0.006376	-0.004596	0.026303	-0.000320
Year	-0.005271	0.012217	0.012262	0.012055	-0.008580	-0.004487	-0.011464	-0.017972	0.936505
BasePolicy	-0.007482	0.015244	-0.046837	-0.201564	0.510926	0.631123	-0.203997	-0.152528	0.016945
duplicated	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

23 rows × 23 columns

Out[41]:

s_Policy_Claim	PastNumberOfClaims	AgeOfPolicyHolder	PoliceReportFiled	WitnessPresent	AgentType	AddressChange_Claim	Year	BasePolicy	duplicated
-0.018990	-0.007468	-0.011362	0.014254	0.012206	0.006634	-0.003655	-0.005271	-0.007482	NaN
0.003856	-0.002737	0.000326	0.024021	0.009821	-0.011200	0.010620	0.012217	0.015244	NaN
-0.007807	-0.002526	0.414294	-0.011857	-0.007938	-0.004417	-0.006856	0.012262	-0.046837	NaN
-0.021743	0.021107	-0.015971	0.027911	0.060782	0.004830	-0.008335	0.012055	-0.201564	NaN
0.003669	-0.020742	-0.005723	-0.022477	-0.004444	0.039611	0.004291	-0.008580	0.510926	NaN
0.015060	-0.036014	0.030478	-0.038740	-0.020751	0.035498	0.006376	-0.004487	0.631123	NaN
-0.004490	0.055917	0.073759	0.001524	0.005308	-0.010503	-0.004596	-0.011464	-0.203997	NaN
-0.018160	0.019060	-0.024090	-0.016677	-0.007589	-0.022859	0.026303	-0.017972	-0.152528	NaN
-0.002632	0.002696	0.018193	0.023830	-0.016881	0.018835	-0.000320	0.936505	0.016945	NaN
0.010941	-0.002515	-0.004911	0.004121	0.006779	0.005742	-0.002906	0.010107	-0.004709	NaN
0.005772	0.011095	0.071358	0.009090	0.000612	-0.004650	0.099173	-0.001962	0.012232	NaN
0.002366	0.013996	-0.000258	0.016352	0.009877	-0.000553	0.007466	-0.013266	-0.005585	NaN
0.424152	-0.020017	0.007396	-0.011576	-0.021457	0.004057	0.001854	-0.003868	0.006977	NaN
1.000000	-0.029249	0.006571	-0.009572	-0.004131	0.008585	-0.003278	-0.005846	0.012784	NaN
-0.029249	1.000000	0.022905	-0.009558	-0.013409	-0.010980	0.011998	0.001692	-0.087192	NaN
0.006571	0.022905	1.000000	-0.008023	-0.002643	-0.007392	-0.004374	0.019317	-0.062889	NaN
-0.009572	-0.009558	-0.008023	1.000000	0.202855	0.023664	-0.009980	0.020970	-0.027819	NaN
-0.004131	-0.013409	-0.002643	0.202855	1.000000	0.011671	-0.007637	-0.017509	-0.032917	NaN
0.008585	-0.010980	-0.007392	0.023664	0.011671	1.000000	-0.026175	0.018047	0.080454	NaN
-0.003278	0.011998	-0.004374	-0.009980	-0.007637	-0.026175	1.000000	-0.000610	0.008688	NaN
-0.005846	0.001692	0.019317	0.020970	-0.017509	0.018047	-0.000610	1.000000	0.012981	NaN
0.012784	-0.087192	-0.062889	-0.027819	-0.032917	0.080454	0.008688	0.012981	1.000000	NaN
NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [42]:
```
features = ["MaritalStatus","Fault", "PolicyType", "VehicleCategory", "VehiclePrice", "DriverRating", "Days_Policy_Accident",
            "Days_Policy_Claim", "PastNumberOfClaims", "AgeOfPolicyHolder", "PoliceReportFiled",
            "WitnessPresent", "AgentType",  "AddressChange_Claim", "BasePolicy"]
x = data.loc[:,features]
y = data.loc[:,"FraudFound_P"]
```

In [43]:
```
x
```

Out[43]:

	MaritalStatus	Fault	PolicyType	VehicleCategory	VehiclePrice	DriverRating	Days_Policy_Accident	Days_Policy_Claim	PastNumberOfClaims	AgeOfPolic
0	0	0	5	2	4	1	3	2	4	
1	0	0	4	2	4	4	3	2	4	
2	1	0	4	2	4	3	3	2	1	
3	1	1	2	2	0	2	3	2	1	
4	0	1	4	2	4	1	3	2	4	
...	...	...	...	...	...	...	...	...	...	
15415	1	0	1	1	0	4	3	2	2	
15416	1	0	2	2	1	3	3	2	3	
15417	0	0	1	1	0	4	3	2	3	
15418	1	1	0	1	0	4	3	2	4	
15419	0	0	1	1	0	4	3	2	4	

15100 rows × 15 columns

Out[43]:

_Policy_Accident	Days_Policy_Claim	PastNumberOfClaims	AgeOfPolicyHolder	PoliceReportFiled	WitnessPresent	AgentType	AddressChange_Claim	BasePolicy
3	2	4	3	0	0	0	1	2
3	2	4	4	1	0	0	0	1
3	2	1	6	0	0	0	0	1
3	2	1	7	1	0	0	0	2
3	2	4	4	0	0	0	0	1
...	...	...	...	...	...	...	...	...
3	2	2	4	0	0	0	0	1
3	2	3	4	0	0	0	0	2
3	2	3	3	0	0	0	0	1
3	2	4	4	0	0	0	0	0
3	2	4	3	0	0	0	0	1

In [44]:

```
y
```

Out[44]:

```
0        0
1        0
2        0
3        0
4        0
        ..
15415    1
15416    0
15417    1
15418    0
15419    1
Name: FraudFound_P, Length: 15100, dtype: int64
```

7. Proposed Data Mining Solutions

We used Decision Tree Classification, Random Forest Classification, Naïve Bayes prediction technique for "**Vehicle Insurance Claim Fraud Detection**" dataset. Classification prediction modeling assigns a class label to the input values. It will predict the class categories/label for the new data. In addition, the goal behind selecting the classification prediction is that it accurately predicts the target class for each case in the data.

Classification

In general, a classification algorithm is a function that weights the input features such that one class is divided into positive values and the other into negative values by the output. And also, they are required labeled data. Binary classification is used to identify the vehicle claim is a fraud or not.

Classification Model

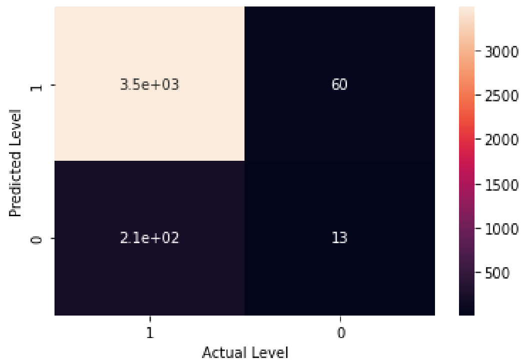
1. <u>Decision Tree</u>

```
In [45]: from sklearn.model_selection import train_test_split

In [46]: X_train, X_test, Y_train, Y_test = train_test_split(x, y, random_state=42, train_size = .75)

In [47]: from sklearn.tree import DecisionTreeClassifier

In [48]: clf = DecisionTreeClassifier(random_state=42)
         clf.fit(X_train,Y_train)

Out[48]: DecisionTreeClassifier(random_state=42)

In [49]: y_pred = clf.predict(X_test)

In [50]: from sklearn import metrics

In [51]: print("Accuracy:",metrics.accuracy_score(Y_test, y_pred))

         Accuracy: 0.9274172185430464

In [52]: from sklearn import tree
         from sklearn.tree import export_graphviz
         import graphviz
         dot_data = tree.export_graphviz(clf, out_file="dec_tree.dot", feature_names= x.columns[0:15], class_names= ["No", "Yes"])
         with open("dec_tree.dot") as f:
             dot_graph = f.read()
         graphviz.Source(dot_graph)

Out[52]: <graphviz.sources.Source at 0x20f8c1e2520>

In [53]: from sklearn.metrics import classification_report, confusion_matrix
```

```
In [75]: print(classification_report(Y_test, y_pred, target_names=["0", "1"]))
         cf_matrix= confusion_matrix(Y_test, y_pred)
         print(cf_matrix)
```

```
              precision    recall  f1-score   support

           0       0.94      0.98      0.96      3548
           1       0.18      0.06      0.09       227

    accuracy                           0.93      3775
   macro avg       0.56      0.52      0.52      3775
weighted avg       0.90      0.93      0.91      3775

[[3488   60]
 [ 214   13]]
```

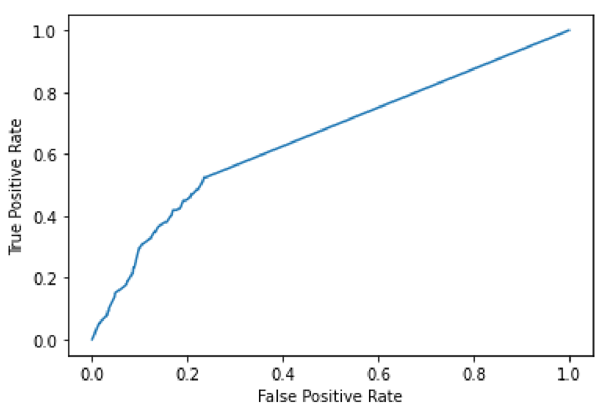
```
In [78]: import seaborn as sns
         x_axis=[1,0]
         y_axis=[1,0]
         ax=sns.heatmap(cf_matrix, annot=True, xticklabels=x_axis, yticklabels=y_axis)
         ax.set_xlabel('Actual Level')
         ax.set_ylabel('Predicted Level');
```




```
In [90]: from sklearn.metrics import roc_curve, roc_auc_score
         y_score = clf.predict_proba(X_test)[:,1]
         false_positive_rate, true_positive_rate, threshold = roc_curve(Y_test, y_score)
         print('ROC_AUC_score for DecisionTree: ', roc_auc_score(Y_test, y_score))
```

```
ROC_AUC_score for DecisionTree:  0.6508258049456416
```

```
In [93]: plt.plot(false_positive_rate,true_positive_rate)
         plt.ylabel('True Positive Rate')
         plt.xlabel('False Positive Rate')
         plt.show()
```



2. Random Forest

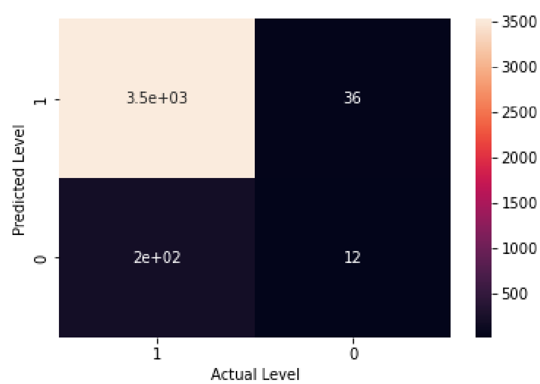
```
In [79]: # Using Random Forest to train the model
         X_train1, X_test1, y_train1, y_test1 = train_test_split(x, y, train_size=0.75)
```

```
In [80]: from sklearn.ensemble import RandomForestClassifier

         #Create a Gaussian Classifier
         ranclf=RandomForestClassifier(n_estimators=100)

         #Train the model using the training sets y_pred=clf.predict(X_test)
         ranclf.fit(X_train1,y_train1)

         y_pred1=ranclf.predict(X_test1)
```

```
In [81]: # Model Accuracy, how often is the classifier correct?
         print("Accuracy:",metrics.accuracy_score(y_test1, y_pred1))

         Accuracy: 0.9366887417218543
```

```
In [85]: from sklearn.metrics import classification_report, confusion_matrix
         print(classification_report(y_test1, y_pred1, target_names=["0", "1"]))
         cf_matrix1= confusion_matrix(y_test1, y_pred1)
         print(cf_matrix1)

                       precision    recall  f1-score   support

                    0       0.95      0.99      0.97      3560
                    1       0.25      0.06      0.09       215

             accuracy                           0.94      3775
            macro avg       0.60      0.52      0.53      3775
         weighted avg       0.91      0.94      0.92      3775

         [[3524   36]
          [ 203   12]]
```

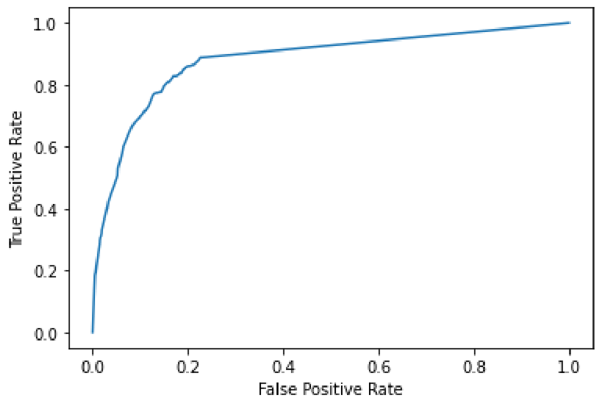
```
In [86]: import seaborn as sns
         x_axis=[1,0]
         y_axis=[1,0]
         ax=sns.heatmap(cf_matrix1, annot=True, xticklabels=x_axis, yticklabels=y_axis)
         ax.set_xlabel('Actual Level')
         ax.set_ylabel('Predicted Level');
```



In [89]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_score1 = clf.predict_proba(X_test1)[:,1]
false_positive_rate1, true_positive_rate1, threshold = roc_curve(y_test1, y_score1)
print('ROC_AUC_score for Random Forest: ', roc_auc_score(y_test1, y_score1))
```

```
ROC_AUC_score for Random Forest:  0.8807068199634178
```

In [92]:
```
plt.plot(false_positive_rate1,true_positive_rate1)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



3. Naive Bayes

In [94]:
```
# Using Naive Bayes Classifier to train the model
X_train2, X_test2, y_train2, y_test2 = train_test_split(x, y, train_size=0.75)
```

In [95]:
```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

#Train the model using the training sets
gnb.fit(X_train2, y_train2)

#Predict the response for test dataset
y_pred2 = gnb.predict(X_test2)
```

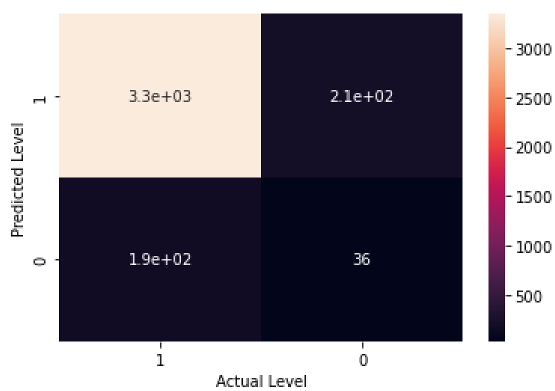
In [96]:
```
# Model Accuracy, how often is the classifier correct?
print("Accuracy:",metrics.accuracy_score(y_test2, y_pred2))
```

```
Accuracy: 0.8945695364238411
```

In [97]:
```
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test2, y_pred2, target_names=["0", "1"]))
cf_matrix2= confusion_matrix(y_test2, y_pred2)
print(cf_matrix2)
```

```
              precision    recall  f1-score   support

           0       0.95      0.94      0.94      3553
           1       0.15      0.16      0.15       222

    accuracy                           0.89      3775
   macro avg       0.55      0.55      0.55      3775
weighted avg       0.90      0.89      0.90      3775

[[3341  212]
 [ 186   36]]
```

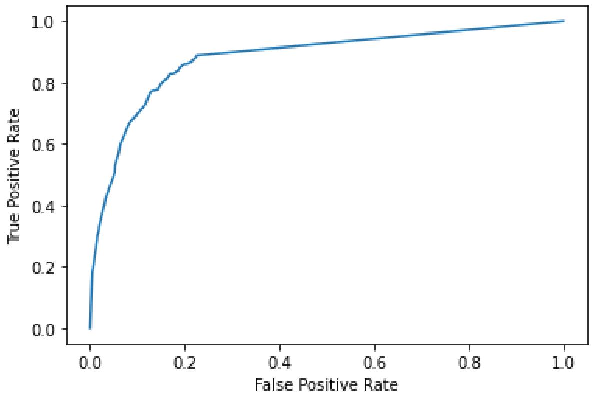
In [98]:
```
import seaborn as sns
x_axis=[1,0]
y_axis=[1,0]
ax=sns.heatmap(cf_matrix2, annot=True, xticklabels=x_axis, yticklabels=y_axis)
ax.set_xlabel('Actual Level')
ax.set_ylabel('Predicted Level');
```



In [99]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_score2 = clf.predict_proba(X_test2)[:,1]
false_positive_rate2, true_positive_rate2, threshold = roc_curve(y_test2, y_score2)
print('ROC_AUC_score for Random Forest: ', roc_auc_score(y_test2, y_score2))
```

```
ROC_AUC_score for Random Forest:  0.872213685681178
```

In [100]:
```
plt.plot(false_positive_rate1,true_positive_rate1)
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



In [58]:
```
import pickle
#Saving model to disk
pickle.dump(clf, open('model.pkl', 'wb'))
```

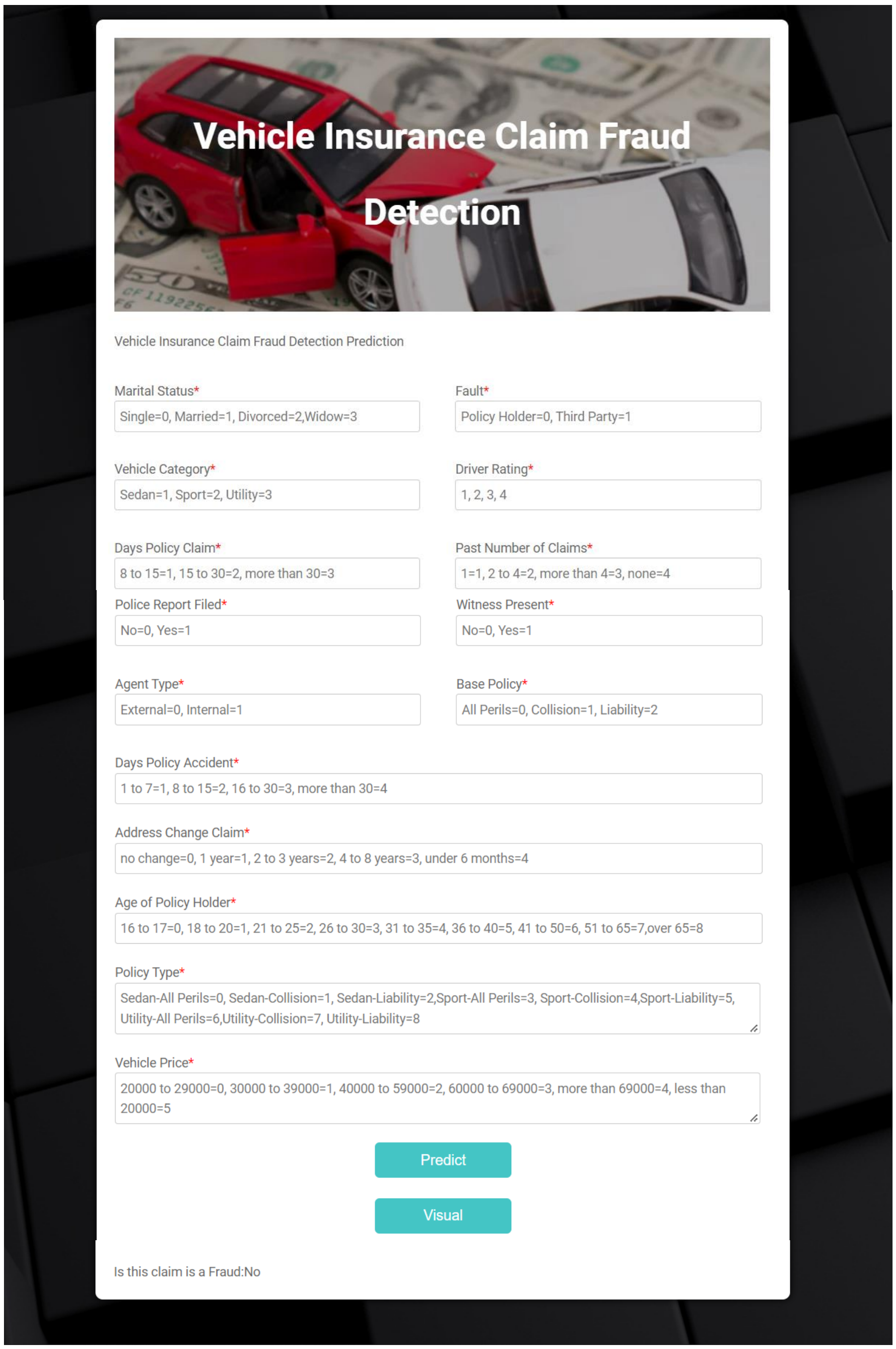
In [59]:
```
model = pickle.load(open('model.pkl','rb'))
```

- Accuracy in Decision Tree Classification Model: 0.9274172
- Accuracy in Random Forest Classification Model: 0.936688
- Accuracy in Naïve Bayes Classification Model: 0.89456953

✓ So, we have selected Random Forest Classifier as our classification model to do the prediction because of its high accuracy than other models.

8. Data Visualization

A simple and easy to use UI has been created so that, provided the feature set, the organization can decide whether the proposed vehicle insurance claim is fair or fraud. The UI was designed in such a way that it was possible to intuitively understand functions. An image of the UI is given below.



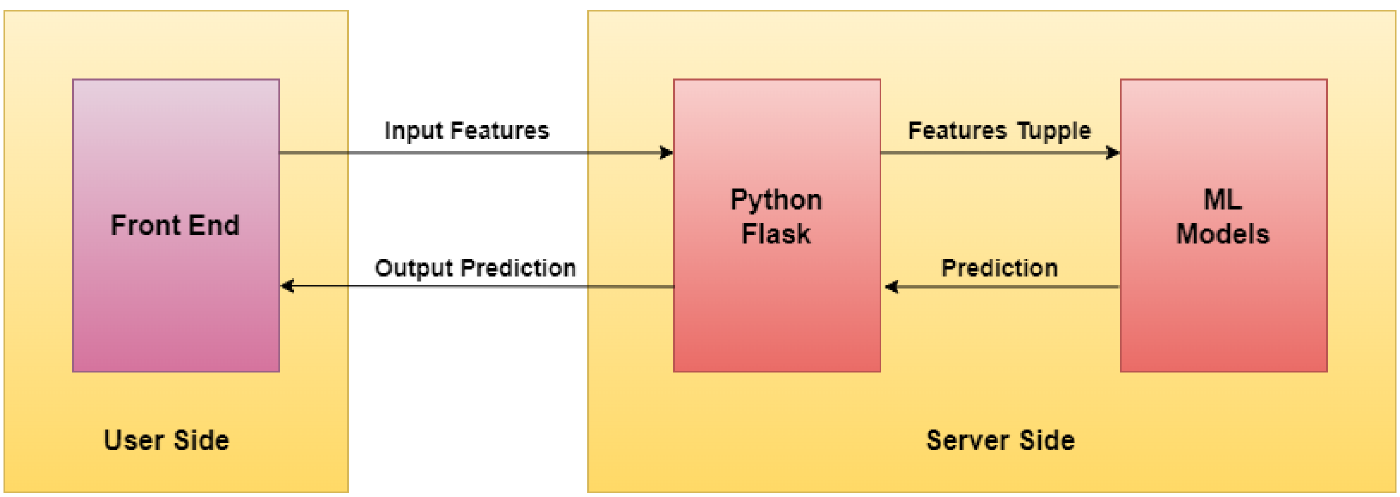
9. Deploying Implementation

Using python, the models are built and trained. As a pickle file(serialization),we have saved the trained model objects. We have built a flask environment with an API endpoint that would encapsulate our trained models and allow them to receive inputs (features) via HTTP / HTPPS POST requests and then return the measured output after the previous serialized models have been serialized.

Via a REST API using Flask, the model is made available to the user. Flask is a micro framework focused on pythons used for designing small-scale websites.

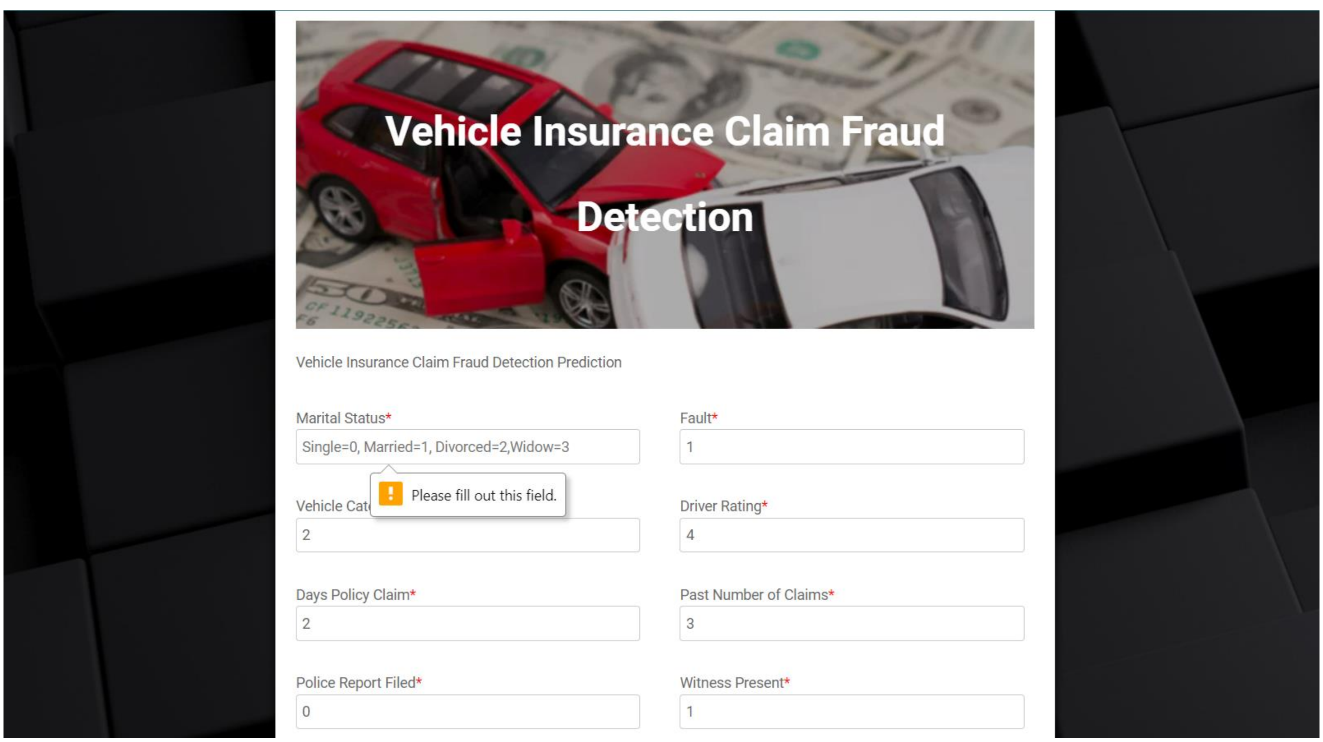
GitHub : https://github.com/IT20205256/FDM_Group03.git

Hosted URL: https://frauddetection-api-1001.herokuapp.com/

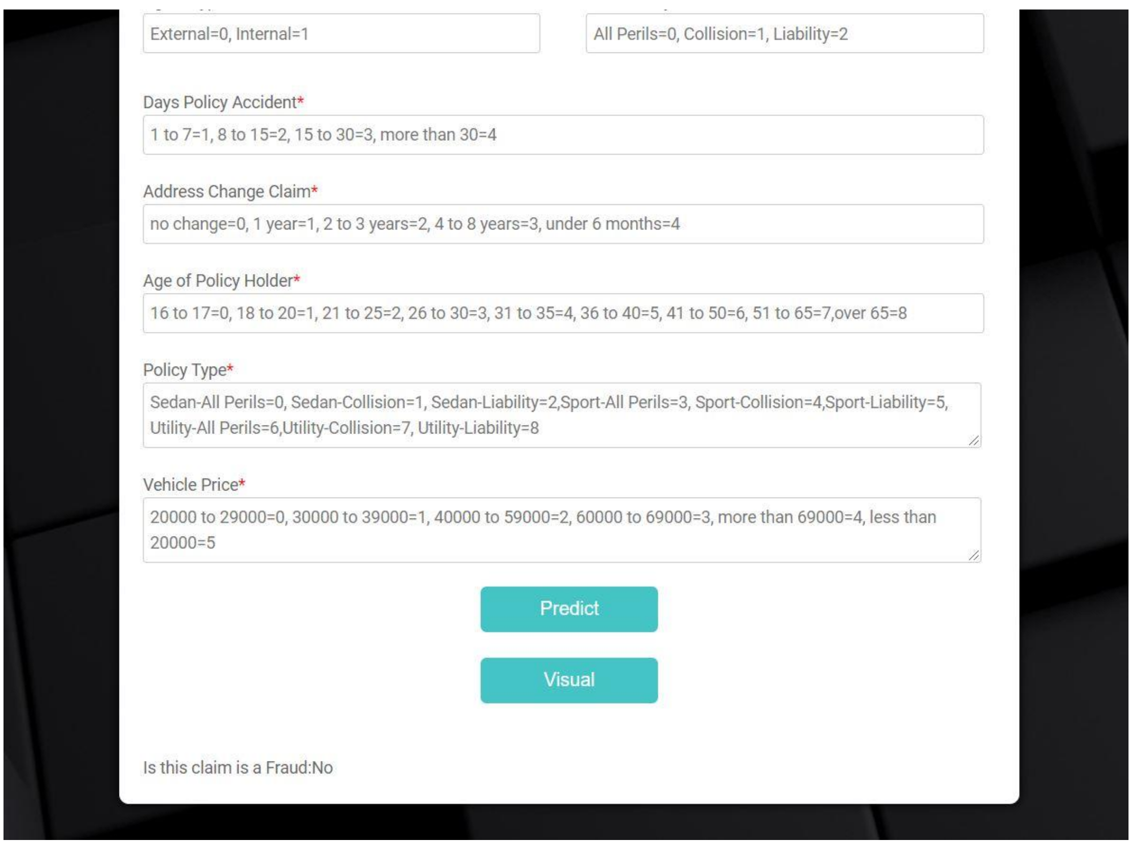


10. Test Cases

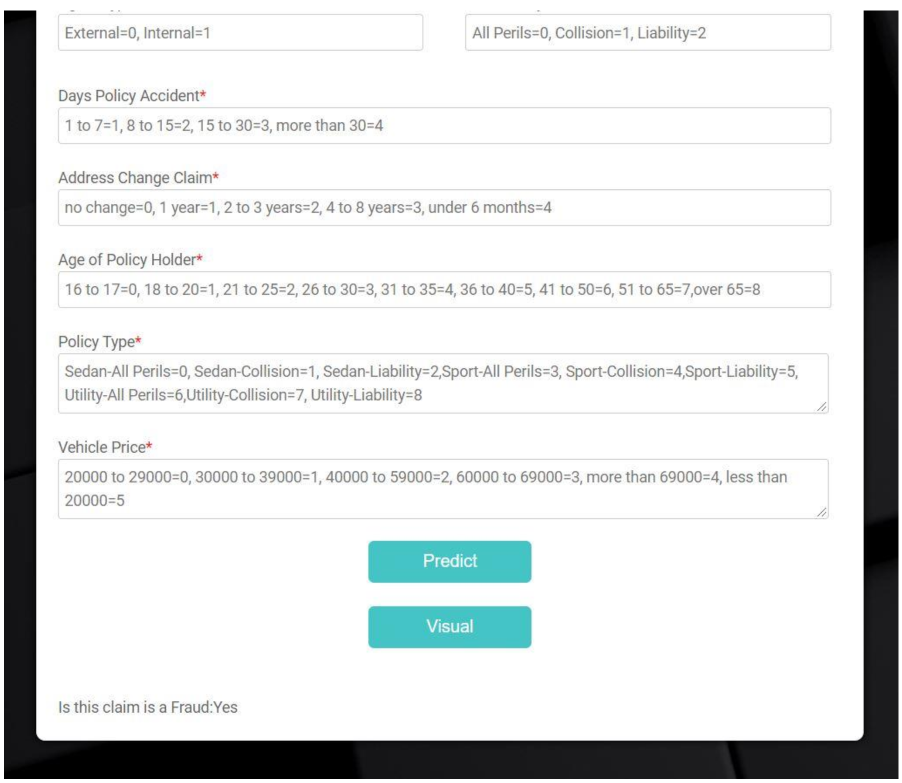
Test Scenario ID	01		
Test Case Description	Test the selected models properly working or not		
Pre-Requisite	Dataset was preprocessed		
Action	Leave some fields and set the inputs and submit		
Inputs	**Expected Output**	**Actual Output**	**Result**
MaritalStatus=? Fault=1 PolicyType=6 VehicleCategory=2 VehiclePrice=1 DriverRating=4 Days_Policy_Accident=1 Days_Policy_Claim=2 PastNumberOfClaims=3 AgeOfPolicyHolder=1 PoliceReportFiled=0 WitnessPresent=1 AgentType=1 AddressChange_Claim=1 BasePolicy=1	A message displays error message as "Please fill out of this field."	A message displays error message as "Please fill out of this field."	Pass



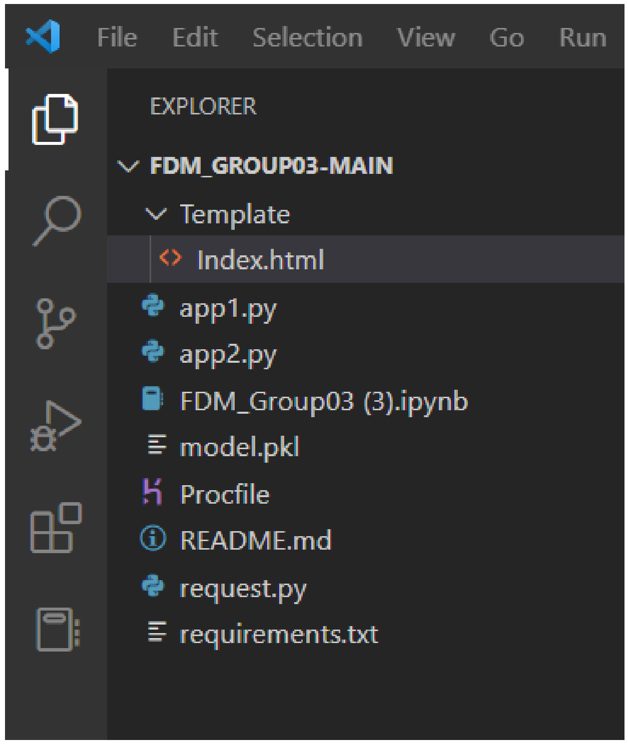
Test Scenario ID	02		
Test Case Description	Test the selected models properly working or not		
Pre-Requisite	Dataset was preprocessed		
Action	Enter all the inputs and click predict		
Inputs	**Expected Output**	**Actual Output**	**Result**
MaritalStatus=1 Fault=1 PolicyType=8 VehicleCategory=2 VehiclePrice=5 DriverRating=4 Days_Policy_Accident=1 Days_Policy_Claim=2 PastNumberOfClaims=3 AgeOfPolicyHolder=0 PoliceReportFiled=0 WitnessPresent=1 AgentType=1 AddressChange_Claim=2 BasePolicy=1	A message displays as "Is this claim is a Fraud:No"	A message displays as "Is this claim is a Fraud:No"	Pass



Test Scenario ID	03		
Test Case Description	Test the selected models properly working or not		
Pre-Requisite	Dataset was preprocessed		
Action	Enter all the inputs and click predict		
Inputs	**Expected Output**	**Actual Output**	**Result**
MaritalStatus=0 Fault=1 PolicyType=0 VehicleCategory=1 VehiclePrice=0 DriverRating=1 Days_Policy_Accident=1 Days_Policy_Claim=1 PastNumberOfClaims=1 AgeOfPolicyHolder=1 PoliceReportFiled=0 WitnessPresent=1 AgentType=1 AddressChange_Claim=0 BasePolicy=2	A message displays as "Is this claim is a Fraud:Yes"	A message displays as "Is this claim is a Fraud:Yes"	Pass



11. The Project Structure



Structure of our project app is arranged below in the following format.

- ✓ **Templates Folder:** This is the default location that template HTML files must be in for flask to render them properly. Any page that interacts with models will be in here.

- ✓ **Index.html:** This is the front-facing HTML file that users can interact with and that your model will output its results to.

- ✓ **app.py:** This file acts as the link between the API file calling the model and the HTML file to display the results and take in user inputs.

12. Benefits of Proposed Solution

Automating the vehicle insurance fraud detection process can help insurers in many ways. It helps reduce the time taken to process a claim, improve the accuracy and quality of claims, and reduce costs associated with underwriting. AI-powered automation allows insurers to manage their customer service operations at scale by reducing human intervention. It is helpful for insurers facing an increasing number of fraudulent claims.

- Reduces the Time Required to Process a Claim
- Allows for Better Record-Keeping
- Automates Internal Processes

13. Conclusion

We use Random Forest Algorithm to build the Classification model to predict and analyze where a claim has a risk of fraudulent or not. Finally, the system was ready with a successfully running model which generates 93% of accurate.

This model provides a computerized system for the insurance department and the insurance claim accepting officer with high risk of fraudulent can reject the claims and be aware of it in the future. This system is user-friendly, easy to use, more efficient and reliable.

Without this system, it is a bit hard job for officers to identify the claims with fraud activities and staged accidents early. This will be a revolution to stop the fraud activities and to force them not to do the same in coming days.

However, finally we finished implementing, testing and integrating successfully.

14. Project Team and Workload

Name	Registration Number	Responsibility
Kishan R.	IT20205256	Scope Planning Implement the Classification Model Select the best model Data Preprocessing Evaluate Model Documentation UI Design and Integrate
Heisapirashoban N.	IT20202668	Scope Planning Implement the Classification Model Select the best model Data Preprocessing Evaluate Model Documentation UI Design and Integrate
Senarathne R.S.A.M.N.T.	IT20205188	Scope Planning Implement the Classification Model Select the best model Data Preprocessing Evaluate Model Documentation UI Design and Integrate
Sandarangi R.M.N.	IT20201678	Scope Planning Implement the Classification Model Select the best model Data Preprocessing Evaluate Model Documentation UI Design and Integrate
Withanage D.U.I.W.	IT20202736	Scope Planning Implement the Classification Model Select the best model Data Preprocessing Evaluate Model Documentation UI Design and Integrate

15. References

- https://www.kaggle.com/datasets/shivamb/vehicle-claim-fraud-detection
- https://medium.com/analytics-vidhya/deploying-a-static-web-application-to-heroku-3f21e07e58a
- https://towardsdatascience.com/create-and-deploy-a-simple-web-application-with-flask-and-heroku-103d867298eb
- https://towardsdatascience.com/deploying-machine-learning-models-with-heroku-4dec1df87f71