

Capstone Project on
Data Exploration and Modelling

Submitted by Gourav Bajaj

Student No. 301148324

To Professor David Parent

Table of Contents

1.0	Dataset Summary	5
2.0	Abstract	7
3.0	Introduction.....	7
4.0	Problem Statement	8
5.0	Objectives	9
6.0	Project Overview	9
7.0	Assumptions.....	10
8.0	Workspace.....	10
9.0	Necessary Libraries for Data Exploration.....	10
10.0	Files Importation.....	11
11.0	Exploratory Data Analysis (EDA).....	11
11.1	Missing values	12
11.2	Data Comprehension	13
11.3	Correlation.	16
11.4	Data distributions and feature engineering	17
11.5	Skewness	33
11.6	Measures of Central tendency and dispersion	33
11.7	Changing datatypes of some columns to Categories	36

12.0	Summary of the EDA Performed.....	36
	Modelling.....	38
1.0	Modelling.....	39
1.1	Libraries for Modelling	39
1.2	Creating Dummies.....	39
1.3	Up-sampling/Balancing the data	40
1.4	Train test split.....	41
2.0	Decision Tree	41
2.1	Full Decision Tree	43
2.2	Decision Tree with max depth 5	45
2.3	Classification Decision Trees Comparison:	48
3.0	Regression Models.....	49
3.1	Linear Regression Model	49
3.2	Forward Regression Model	50
3.3	Backward Regression Model	51
3.4	Regression Model Comparison.	52
4.0	Logistic Regression Model	52
5.0	Decision Tree-GridSearchCV	57
6.0	Random Forest.....	61
7.0	Neural Networks	65

7.1	Neural Network with 40 hidden layers and max_iterations = 1000.....	65
7.2	Neural Network with 52 hidden layers and max_iterations = 1000.....	67
7.3	Neural Network with 45 hidden layers and max_iterations = 1000.....	69
7.4	Neural Network Model Comparison.....	70
8.0	Model Comparison and Selection.....	71
8.1	Classification Models Table.....	71
8.2	Regression Models Table	72
8.3	Model Selection.....	72
9.0	Model Validation Plan	73
10.0	Logistic Regression Outcomes	73
11.0	Recommendation	76
12.0	Future Work	77
	References.....	78

1.0 Dataset Summary

Vehicle insurance claim fraud detection dataset contains vehicle dataset that is attributes, model, accident details. It also contains attributes of the policy holder and policy type. The target is to detect that the application received for claim is a fraud or not – FraudFound_P.

Under this analysis, a predictive analysis has been conducted using python to predict that the application received for claim is a fraud or not. Conspiring to create fraudulent or inflated claims about property damage or personal injuries as a result of an accident is known as vehicle insurance fraud. The use of phantom passengers - people who were not even present at the accident scene claim to have sustained severe injuries, staged accidents, where fraudsters purposefully "arrange" for accidents to occur, and false personal injury claims or where personal injuries are grossly exaggerated are some of the related examples. The dataset taken from Kaggle website (Kaggle, 2021). We will be predicting the fraudulent claims and building the model using the variables tabulated below.

Variables	Description
Month	The 3 letter abbreviations for the months of the year in which accident occurred.
WeekOfMonth	The week in the month the accident occurred.
DayOfWeek	The day of week in which accident occurred.
Make	The list of 19 car manufacturer.
AccidentArea	The area in which accident occurred (Urban/Rural)
DayOfWeekClaimed	The day of week in which the application for claim made.
MonthClaimed	The month of year in which the application for claim made.
WeekOfMonthClaimed	The week of month in which the application for claim made.
Sex	The gender of the policy holder.

MaritalStatus	The Marital status of the policy holder.
Age	The age of the policy holder.
Fault	The person who was at fault (Policy holder/ Third party).
PolicyType	The type of insurance policy and the category of vehicle insured.
VehicleCategory	The category of vehicle insured.
VehiclePrice	The ranges for price of vehicles.
FraudFound_P	The claim application is fraud or not (Binary).
PolicyNumber	The policy number
RepNumber	This column contains integers from 1 to 16 [Assumed as employee ID)
Deductible	The amount to be deducted for the claim.
DriverRating	The driver ratings on the basis of driving skills.
Days_Policy_Accident	The number of days between when the policy was purchased, and the accident occurred.
Days_Policy_Claim	The number of days between when the policy was purchased, and the claim filed.
PastNumberOfClaims	The past number of claims filed by policy holder.
AgeOfVehicle	It states how old is the Vehicle at the time of accident.
AgeOfPolicyHolder	The age of policy holder bucketed into the ranges.
PoliceReportFiled	Indicates whether a police report was filed (Binary).
WitnessPresent	Indicated whether a witness was present at the time of accident (Binary).
AgentType	The agent handling the claim is external or internal.
NumberOfSuppliments	The number of supplements to the claims.
AddressChange_Claim	The difference in years from the address change application to the claim filled
NumberOfCars	The number of cars covered under the insurance policy.
Year	Year in which the accident occurred.
BasePolicy	Type of insurance coverage.

2.0 Abstract

The purpose of our analysis was to determine key factors and build a model that are most closely indicative of a claim application being fraud. Using 33 variables provided by the dataset. Various decision trees, regression models, random forest, logistic regression and neural networks were run for this analysis and a model comparison was conducted to determine which model was most predictive of the target. Using Accuracy scores, F1 Score and RMSE, we determined that the best model was the Logistic Regression Model. The output of this model revealed that Days_Policy_Claim, AgeOfPolicyHolder_21 to 25, NumberOfCars, Sex_Male_odds, Grouped_Make_Non-Luxury are the factors that are most indicative of predicting a fraudulent claim application. From this analysis, we recommend that after being predicted as fraudulent claim, further investigation should be conducted, and strict actions should be taken if found guilty. Moreover, fraud control strategies should be formulated and implemented to prevent fraud in future.

3.0 Introduction

The insurance sector has established itself as a fundamental pillar of our contemporary society. Even though the nature of the insurance business and its value proposition are likely to undergo significant changes due to the pressures of technological development, trends toward globalization, and the deregulation of the real estate and financial markets, it will undoubtedly continue to hold that status in the future. In order to effectively manage risk and complexity for people, social groupings, and enterprises, insurance has become a crucial component. It has given us the ability to handle situations that are getting more complicated and unclear.

However, insurance industries are facing a decline in earnings, reserve deficiencies,

rising loss costs and other insurance expenses, as well as pricing difficulties because of fraudulent claims. Insurance fraud has most certainly been around from the very beginning. Nevertheless, the amounts involved in fraud have certainly increased as insurance made its transition into modern consumer society. They are believed to materially escalate the cost of certain types of insurance. Eventually, they form a threat to the very principle of solidarity that keeps the insurance concept alive. Any stage of the deal may be affected by internal or external fraud. Excessive insurance, fraud, corruption, willful property damage, or forgeries. The fraudulent behavior causes insurance firms to focus on two issues: how fraud is assessed and how it affects the cost of premiums.

Auto/vehicle insurance is the insurance for cars, trucks, motorcycles, and other road vehicles. Its principal role is to provide financial protection from risks like property damage and bodily injury resulting from liability arising from incidents in a vehicle. Property and Casualty insurance covers 51.1% share of total annual premiums received, and auto insurance is one type of property and casualty insurance product. In addition, the Auto Insurance industry as of 2022, is worth about \$316 billion. Vehicle insurance fraud involves conspiring to make false or exaggerated claims involving property damage or personal injuries following an accident. The employment of phantom passengers, in which persons who weren't even there when the accident occurred claim to have suffered severe injuries, staged accidents, fake personal injury claims, and dramatically exaggerated injuries are a few instances of popular fraud techniques.

4.0 Problem Statement

Insurance fraud has accompanied insurance since its inception, but the manner in which these practices and their methods of operation have evolved over time, and the volume and frequency of insurance fraud incidents have recently increased. These fraudulent activities have resulted in

high costs for insurance companies. In order to prevent fraud, there is a need to detect fraudulent claims, fraud investigation, and implement fraud control strategies.

5.0 Objectives

The purpose of this analysis is to detect the fraudulent claims from the dataset that will help in classifying the policyholders to whom the compensation should be made. Moreover, it is important to perform this analysis to increase the company's earnings and reserving potential. Likewise, there is no need of wasting resources on assigning a specialist to analyze every single claim. However, if this analysis is not performed, then, the company have to face pricing difficulties and have to charge more even to those consumers who are not even submitting intimation for the claim. The dataset contains information about the insurance policy as well as about the consumer. Also, the data set provides information about the person who was at fault. This dataset contains vehicle details (attributes, model, etc.) along with policy details (type, tenure, etc.). The target is to detect if a claim application is fraudulent or not.

6.0 Project Overview

The purpose of this project is to develop and predict the best model and key identifiers that can detect vehicle insurance fraudulent claims by analyzing the vehicle insurance claim fraud dataset. The difficulty with machine learning fraud detection is that fraud is significantly less common than legitimate insurance claims, So, data balancing technique will be deployed. Moreover, Exploratory Data Analysis (EDA) techniques will be executed to take insights from data. Moving further, Various models will be developed in order to find the best model for the detection of fraud.

7.0 Assumptions

1. The personal information about the policyholder provided in the dataset is legitimate.
2. The testimony provided by the witness of the accident in the dataset is assumed to be true.
3. The data is stationary through the observation window mentioned (1994-1996).
4. The data is the correct representation of the time frame.
5. “Year” column is assumed to be the year in which the accident occurred.
6. “Days_Policy_Accident” column is assumed to be the number of days between when the policy was purchased, and the accident occurred.
7. “Days_Policy_Claim” column is assumed to be the number of days that pass between the policy was purchased and the claim was filed.
8. “RepNumber” column is assumed to be an employee id.

8.0 Workspace

Python: - Python is a high-level, all-purpose programming language. It is a programming language that is open-source and cost-free to use. Python places a strong emphasis on code readability in its design philosophy. In a nutshell, "The Zen of Python" is a document that captures the fundamental philosophy of the language. It provides vast libraries support, improved productivity, and versatile nature (Wikipedia, n.d.)

9.0 Necessary Libraries for Data Exploration

Importing necessary libraries for data exploration and cleaning.

```

import pandas as pd
import numpy as np
import seaborn as sns
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import matplotlib.pyplot as plt
%matplotlib inline

```

10.0 Files Importation

Uploading vehicle insurance claim fraud dataset which is in CSV format and setting the display columns to maximum in order to view all the columns (variables). The following is the code for the same.

```

[3] pd.set_option('display.max_columns', None)

```

```

[4] fraud = pd.read_csv('fraud_oracle.csv')
     print(fraud.shape)
     fraud.head()

```

(15420, 33)

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	Age	Fault
0	Dec	5	Wednesday	Honda	Urban	Tuesday	Jan	1	Female	Single	21	Policy Holder
1	Jan	3	Wednesday	Honda	Urban	Monday	Jan	4	Male	Single	34	Policy Holder
2	Oct	5	Friday	Honda	Urban	Thursday	Nov	2	Male	Married	47	Policy Holder
3	Jun	2	Saturday	Toyota	Rural	Friday	Jul	1	Male	Married	65	Third Party
4	Jan	5	Monday	Honda	Urban	Tuesday	Feb	2	Female	Single	27	Third Party



11.0 Exploratory Data Analysis (EDA)

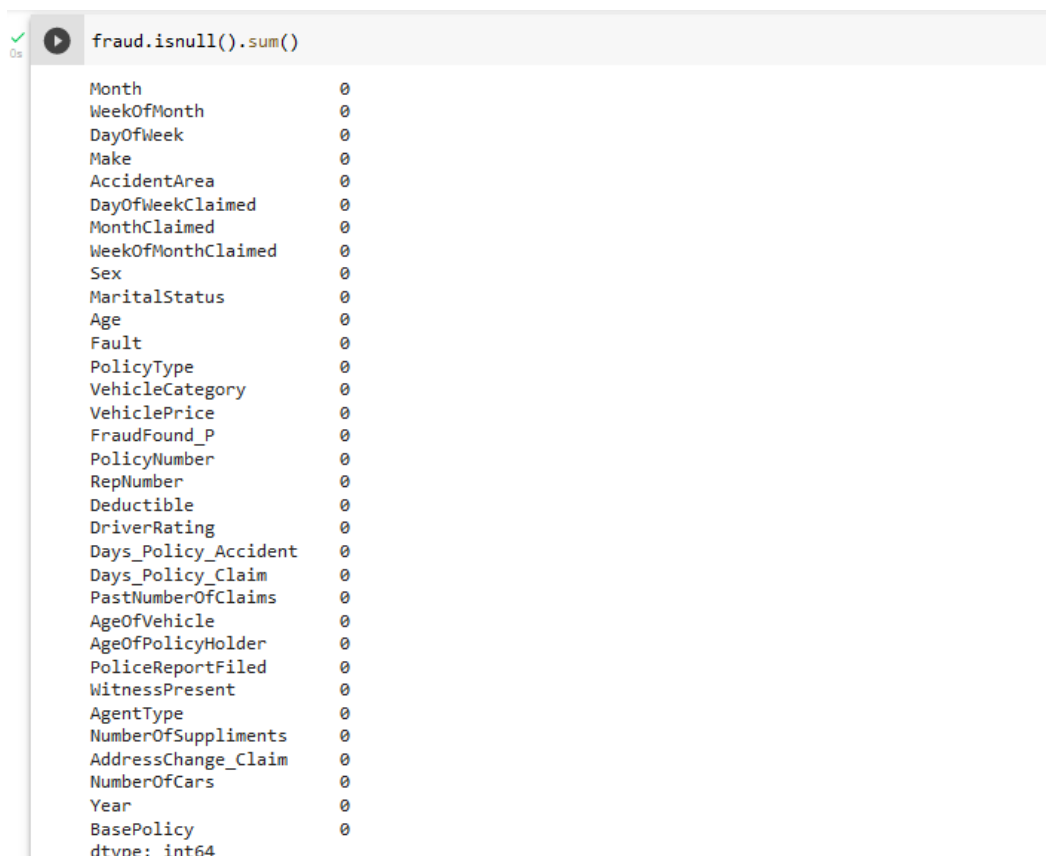
Exploratory data analysis is a technique for analyzing data to discover insights or important features. EDA can be graphical analysis or non-graphical analysis. Any data science or machine learning approach must include EDA. To create a trustworthy and useful output based

on the data, you must examine the data, comprehend the relationships between variables, and comprehend the underlying structure of the data. (analyticsvidhya, 2022)

The following steps will be taken to investigate the data and feature extraction.

11.1 Missing values

Checking if the data has missing values. If missing values are found these can be imputed using the SimpleImputer feature from the Scikit-learn library. Simple methods for imputing missing values are provided by the SimpleImputer class. The statistics (mean, median, or most frequent) of each column in which the missing values are present can be used to impute missing values or they can be replaced with a constant value. Additionally, this class supports several missing value encodings. The vehicle insurance claim fraud dataset doesn't have any missing value.



```
fraud.isnull().sum()
```

Month	0
WeekOfMonth	0
DayOfWeek	0
Make	0
AccidentArea	0
DayOfWeekClaimed	0
MonthClaimed	0
WeekOfMonthClaimed	0
Sex	0
MaritalStatus	0
Age	0
Fault	0
PolicyType	0
VehicleCategory	0
VehiclePrice	0
FraudFound_P	0
PolicyNumber	0
RepNumber	0
Deductible	0
DriverRating	0
Days_Policy_Accident	0
Days_Policy_Claim	0
PastNumberOfClaims	0
AgeOfVehicle	0
AgeOfPolicyHolder	0
PoliceReportFiled	0
WitnessPresent	0
AgentType	0
NumberOfSupplements	0
AddressChange_Claim	0
NumberOfCars	0
Year	0
BasePolicy	0
dtype: int64	

11.2 Data Comprehension

Unique values of each column have been observed. The following are some of the discrepancies found in some variables: -

1. DayOfWeekClaimed: - The DayOfWeekClaimed column contains a unique value as zero ('0').
2. Monthclaimed: - The Monthclaimed column contains a unique value as zero ('0').
3. Age: - The Age column contains a unique value as zero ('0').

✓
0s

```
[7] print('DayOfWeekClaimed',fraud['DayOfWeekClaimed'].unique())
```

```
DayOfWeekClaimed ['Tuesday' 'Monday' 'Thursday' 'Friday' 'Wednesday' 'Saturday' 'Sunday' '0']
```

✓
0s

```
[8] print('MonthClaimed',fraud['MonthClaimed'].unique())
```

```
MonthClaimed ['Jan' 'Nov' 'Jul' 'Feb' 'Mar' 'Dec' 'Apr' 'Aug' 'May' 'Jun' 'Sep' 'Oct' '0']
```

✓
0s

```
print('Age',fraud['Age'].unique())
```

```
Age [21 34 47 65 27 20 36  0 30 42 71 52 28 61 38 41 32 40 63 31 45 60 39 55  
35 44 72 29 37 59 49 50 26 48 64 33 74 23 25 56 16 68 18 51 22 53 46 43  
57 54 69 67 19 78 77 75 80 58 73 24 76 62 79 70 17 66]
```

Checking the rows in which discrepancies are found and replacing the values. The following are the rows in which discrepancies are found:

```
week_0 = fraud["DayOfWeekClaimed"] == "0"
day_of_week_0 = fraud[week_0]
day_of_week_0
```

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	Age	Fault	PolicyType	VehicleCategory
1516	Jul	2	Monday	Honda	Rural	0	0	1	Male	Single	0	Policy Holder	Sedan - All Perils	Sedan

```
month_0 = fraud["MonthClaimed"] == "0"
monthclaimed_0 = fraud[month_0]
monthclaimed_0
```

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	Age	Fault	PolicyType	VehicleCategory
1516	Jul	2	Monday	Honda	Rural	0	0	1	Male	Single	0	Policy Holder	Sedan - All Perils	Sedan

```
age_0 = fraud["Age"] == 0
agefound_0 = fraud[age_0]
print(len(agefound_0))
agefound_0
```

320

	Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	Age	Fault
7	Nov	1	Friday	Honda	Urban	Tuesday	Mar	4	Male	Single	0	Policy Holder
13	Jan	5	Friday	Honda	Rural	Wednesday	Feb	1	Male	Single	0	Third Party
28	Jul	1	Saturday	Honda	Urban	Tuesday	Sep	4	Male	Single	0	Policy Holder
31	Mar	1	Sunday	Honda	Urban	Tuesday	Mar	2	Male	Single	0	Policy Holder
58	May	1	Monday	Honda	Rural	Wednesday	May	4	Male	Single	0	Policy Holder

Row number 1516 is the only row in which zeros are found for both DayOfWeekClaimed and MonthClaimed column. In Age column there are a total of 320 rows in which age is zero.

Replacing the zeros with legitimate values: -

For row number 1516 and column DayOfWeekClaimed, replacing zero with the same categorical value that is in the DayOfWeek which is 'Monday'. For row number 1516 and

column MonthClaimed, replacing zero with month 'Aug' after examining the values for columns WeekOfMonthClaimed and WeekOfMonth.

```
0s [13] fraud["DayOfWeekClaimed"] = fraud["DayOfWeekClaimed"].replace(["0"],["Monday"])

0s [14] print('DayOfWeekClaimed',fraud['DayOfWeekClaimed'].unique())

DayOfWeekClaimed ['Tuesday' 'Monday' 'Thursday' 'Friday' 'Wednesday' 'Saturday' 'Sunday']

0s [15] fraud["MonthClaimed"] = fraud["MonthClaimed"].replace(["0"],["Aug"])

0s [16] print('MonthClaimed',fraud['MonthClaimed'].unique())

MonthClaimed ['Jan' 'Nov' 'Jul' 'Feb' 'Mar' 'Dec' 'Apr' 'Aug' 'May' 'Jun' 'Sep' 'Oct']
```

For Age column, it was observed that all the zeros belong to one class interval from AgeOfPolicyHolder column. So, decided to replace the zero in the column with the mean of the class interval from the column AgeOfPolicyHolder.

```
0s [17] agefound_0['AgeOfPolicyHolder'].unique()

array(['16 to 17'], dtype=object)

0s [18] fraud["Age"] = fraud["Age"].replace([0],[16.5])

0s [19] print('Age',fraud['Age'].unique())

Age [21. 34. 47. 65. 27. 20. 36. 16.5 30. 42. 71. 52. 28. 61.
 38. 41. 32. 40. 63. 31. 45. 60. 39. 55. 35. 44. 72. 29.
 37. 59. 49. 50. 26. 48. 64. 33. 74. 23. 25. 56. 16. 68.
 18. 51. 22. 53. 46. 43. 57. 54. 69. 67. 19. 78. 77. 75.
 80. 58. 73. 24. 76. 62. 79. 70. 17. 66.]
```

Checking, if AgeOfPolicyHolder is formed by bucketing the Age column

```

✓ [20] groupings = []
0s   for holder in fraud['AgeOfPolicyHolder']:
      if 'to' in holder :
          temp = holder.split()
          nr=[int(temp[0]),int(temp[2])]
          groupings.append(nr)
      else :
          temp = holder.split()
          nr = [int(temp[1]),129]
          groupings.append(nr)

```

```

✓ [21] age_idx = []
0s   rw_idx = []
      for r in range(len(fraud['Age'])):
          if (fraud.loc[r,'Age']>= groupings[r][0]) & (fraud.loc[r,'Age']<= groupings[r][1]):
              age_idx.append(0)
          else:
              age_idx.append(1)
              rw_idx.append(r)

```

```

✓ [22] print(len(fraud.loc[list(rw_idx),('Age','AgeOfPolicyHolder')]))
0s   fraud.loc[list(rw_idx),('Age','AgeOfPolicyHolder')].head()

```

6922

	Age	AgeOfPolicyHolder
0	21.0	26 to 30
4	27.0	31 to 35
5	20.0	21 to 25
8	30.0	31 to 35
9	42.0	36 to 40

So, in total there are 6922 rows out of 15420 rows that doesn't belong to the bucket. So, it can be assumed that the person who met with the accident is different from the policyholder.

11.3 Correlation.

There were two columns PolicyNumber and Year are highly correlated with other, with a correlation of 0.94. Also, PolicyNumber column consists of serial number from 1 to 15420.

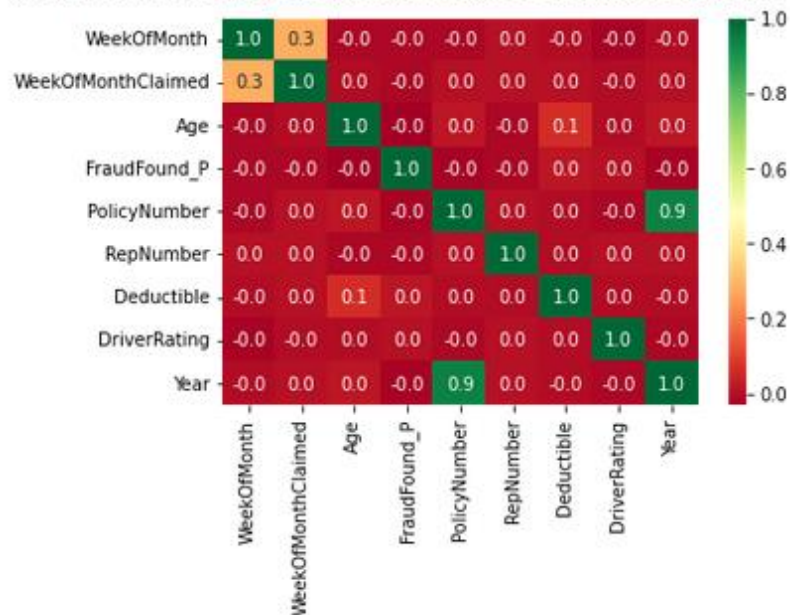
Thus, decided to drop PolicyNumber from final dataset to be used for modelling.


```
[24] fraud_corr = fraud_corr().round(2)
      fraud_corr
```

	WeekOfMonth	WeekOfMonthClaimed	Age	FraudFound_P	PolicyNumber	RepNumber	Deductible	DriverRating	Year
WeekOfMonth	1.00	0.28	-0.01	-0.01	-0.01	0.01	-0.00	-0.02	-0.00
WeekOfMonthClaimed	0.28	1.00	0.00	-0.01	0.01	0.01	0.01	-0.00	0.01
Age	-0.01	0.00	1.00	-0.03	0.02	-0.01	0.07	0.00	0.02
FraudFound_P	-0.01	-0.01	-0.03	1.00	-0.02	-0.01	0.02	0.01	-0.02
PolicyNumber	-0.01	0.01	0.02	-0.02	1.00	0.01	0.00	-0.01	0.94
RepNumber	0.01	0.01	-0.01	-0.01	0.01	1.00	0.00	0.01	0.01
Deductible	-0.00	0.01	0.07	0.02	0.00	0.00	1.00	0.00	-0.00
DriverRating	-0.02	-0.00	0.00	0.01	-0.01	0.01	0.00	1.00	-0.01
Year	-0.00	0.01	0.02	-0.02	0.94	0.01	-0.00	-0.01	1.00

```
sns.heatmap(fraud_corr, annot=True, fmt='.1f', cmap='RdYlGn')
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f3b02126090>



The above is the code to plot heatmap for correlation table (fraud_corr)

11.4 Data distributions and feature engineering.

Data visualization is the process of displaying data using popular images like infographics, charts, and even animations. These informational visual representations convey intricate data

linkages and data-driven insights in a way that is simple to comprehend (IBM, 2021). Data distribution is the graphical representation of the data which helps in investigating the data and its variables. The act of turning raw data into features that may be used to build a predictive model utilizing machine learning or statistical modelling, such as deep learning, is known as feature engineering. The goal of feature engineering is to optimize the performance of machine learning models by creating an input data set that best matches the algorithm (TechTarget, n.d.)

The following are the distributions of the variables of the dataset: -





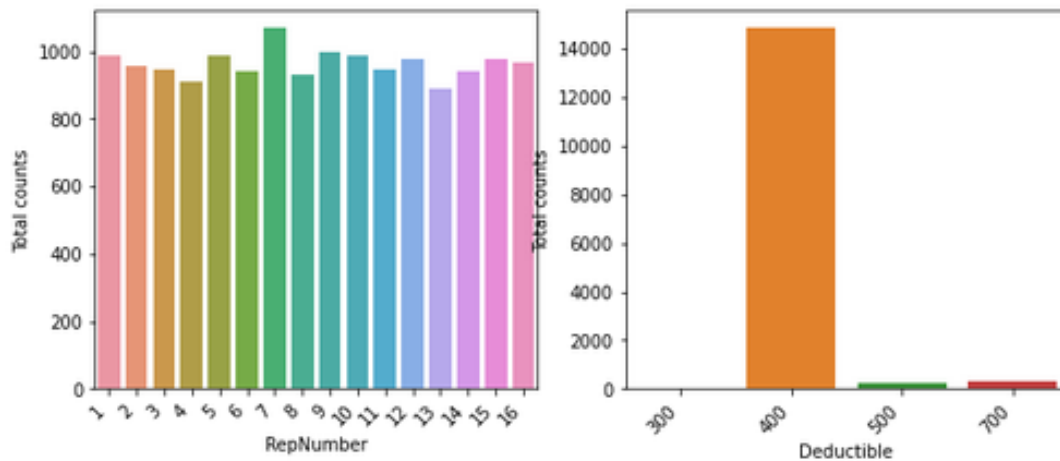
```
RepNumber = fraud.groupby('RepNumber').agg("count").reset_index()
Deductible= fraud.groupby('Deductible').agg("count").reset_index()

fig, (ax1,ax2) = plt.subplots(1,2,figsize=(10,4))
grph1=sns.barplot(x='RepNumber', y="FraudFound_P", data = RepNumber , ax=ax1)
grph2= sns.barplot(x='Deductible', y ="FraudFound_P", data = Deductible, ax=ax2)

ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')

grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right'
                      )
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right'
                      )
```

```
[Text(0, 0, '300'), Text(0, 0, '400'), Text(0, 0, '500'), Text(0, 0, '700')]
```



RepNumber seems equally distributed, however, Deductible variable seems to be highly skewed as most of the payments deducted were 400.



For column Days_Policy_Accident, the bucketed range will be converted to integer and mode value of the bucket range will be fed as value for the respective range.

```

[42] dayspolicy_accident = {"1 to 7" : 7 , "15 to 30" : 30 , "8 to 15" : 15 , "more than 30" : 45 , "none" : 0 }
fraud["Days_Policy_Accident"] = fraud["Days_Policy_Accident"].apply(lambda x: dayspolicy_accident[x])

print('Days_Policy_Accident', fraud['Days_Policy_Accident'].unique())

Days_Policy_Accident [45 30 0 7 15]

```

✓ 1s

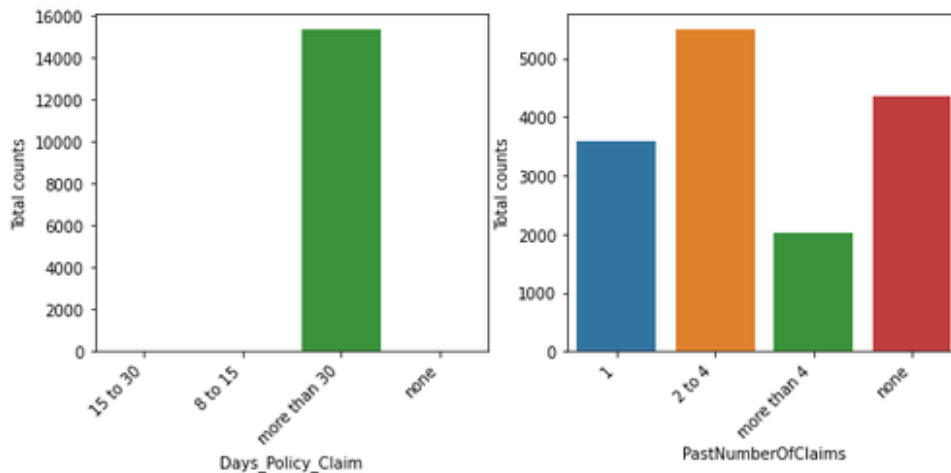
```
Days_Policy_Claim = fraud.groupby('Days_Policy_Claim').agg("count").reset_index()
PastNumberOfClaims = fraud.groupby('PastNumberOfClaims').agg("count").reset_index()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
grph1 = sns.barplot(x='Days_Policy_Claim', y="FraudFound_P", data = Days_Policy_Claim, ax=ax1)
grph2 = sns.barplot(x='PastNumberOfClaims', y="FraudFound_P", data = PastNumberOfClaims, ax=ax2)

ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')

grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right'
                      )
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right'
                      )
```

```
[Text(0, 0, '1'),
Text(0, 0, '2 to 4'),
Text(0, 0, 'more than 4'),
Text(0, 0, 'none')]
```



For columns Days_Policy_Claim and PastNumberOfClaims, the bucketed range will be converted to integer and mode value of the bucket range will be fed as value for the respective range.

✓

0s

```
[44] dayspolicy_claim = {"15 to 30" : 30, "8 to 15" : 15, "more than 30" : 45, "none" : 0 }
      fraud["Days_Policy_Claim"] = fraud['Days_Policy_Claim'].apply(lambda x: dayspolicy_claim[x])
```

✓

0s

```
[45] print("Days_Policy_Claim", fraud["Days_Policy_Claim"].unique())
```

```
Days_Policy_Claim [45 30 15 0]
```

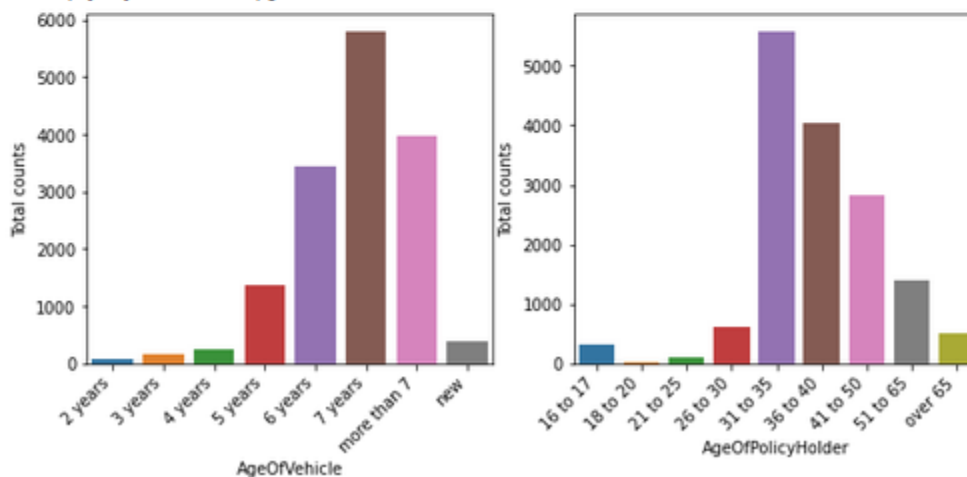
```
[47] pastnumber_claim = {"1" : 1, "2 to 4" : 4, "more than 4" : 5, "none" : 0 }
      fraud["PastNumberOfClaims"] = fraud["PastNumberOfClaims"].apply(lambda x: pastnumber_claim[x])
```

```
print("PastNumberOfClaims", fraud["PastNumberOfClaims"].unique())
```

```
PastNumberOfClaims [0 1 4 5]
```

```
AgeOfVehicle = fraud.groupby('AgeOfVehicle').agg("count").reset_index()
AgeOfPolicyHolder = fraud.groupby('AgeOfPolicyHolder').agg("count").reset_index()
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
grph1 = sns.barplot(x='AgeOfVehicle', y="FraudFound_P", data = AgeOfVehicle, ax=ax1)
grph2 = sns.barplot(x='AgeOfPolicyHolder', y="FraudFound_P", data = AgeOfPolicyHolder, ax=ax2)
ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')
grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
```

```
[Text(0, 0, '16 to 17'),
Text(0, 0, '18 to 20'),
Text(0, 0, '21 to 25'),
Text(0, 0, '26 to 30'),
Text(0, 0, '31 to 35'),
Text(0, 0, '36 to 40'),
Text(0, 0, '41 to 50'),
Text(0, 0, '51 to 65'),
Text(0, 0, 'over 65')]
```



For column AgeOfVehicle, the bucketed range will be converted to integer and mode value of the bucket range will be fed as value for the respective range.

```

[49] agevehicle = {"2 years" : 2, "3 years" : 3, "4 years" : 4, "5 years" : 5, "6 years" : 6, "7 years" : 7, "more than 7" : 8, "new" : 1 }
      fraud["AgeOfVehicle"] = fraud["AgeOfVehicle"].apply(lambda x: agevehicle[x])

```

```

print("AgeOfVehicle", fraud["AgeOfVehicle"].unique())

```

```

AgeOfVehicle [3 6 7 8 5 1 4 2]

```

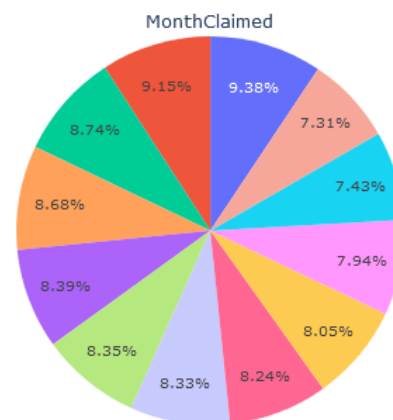
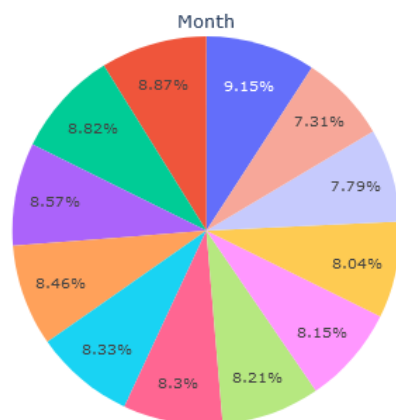
```

[54] fig = make_subplots(rows=1, cols=2, specs=[[{'type':'domain'}], {'type':'domain'}], subplot_titles=['Month', 'MonthClaimed'])

fig.add_trace(go.Pie(labels= fraud["Month"].value_counts().index, values=[x for x in fraud["Month"].value_counts()], name="Month"),
              1, 1)
fig.add_trace(go.Pie(labels= fraud["MonthClaimed"].value_counts().index, values=[x for x in fraud["MonthClaimed"].value_counts()], name="Month"),
              1, 2)

fig.show()

```



Jan
 May
 Mar
 Jun
 Oct
 Dec
 Apr
 Feb
 Jul
 Sep
 Nov
 Aug



Bucketing Saturday and Sunday with Monday to keep the distribution consistent so that the values are logical and follow numerical coherence.

```
[56] group__weekclaimed = {"Tuesday" : "Tuesday", "Monday" : "Monday", "Thursday" : "Thursday", "Friday" : "Friday", "Wednesday" : "Wednesday", "Saturday" : "Monday", "Sunday" : "Monday"}

[58] fraud['DayOfWeekClaimed'] = fraud['DayOfWeekClaimed'].apply(lambda x: group__weekclaimed[x])

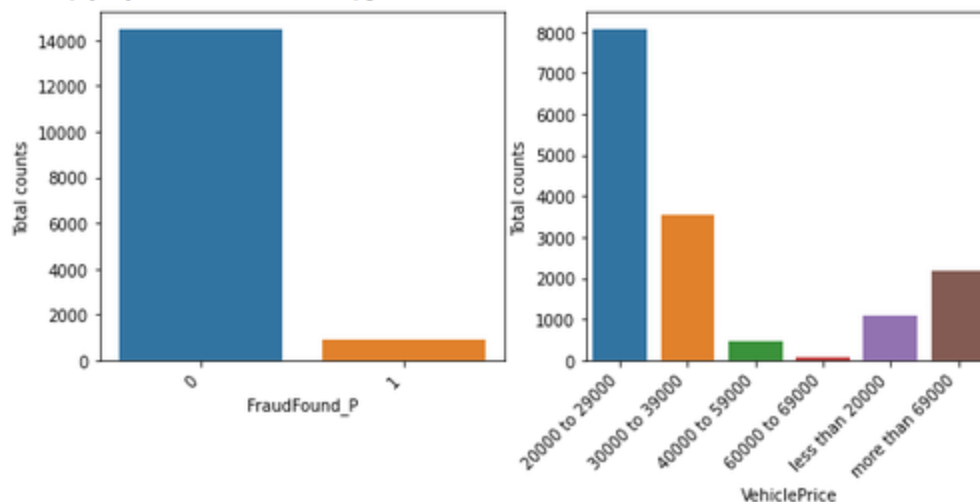
print('DayOfWeekClaimed', fraud['DayOfWeekClaimed'].unique())

DayOfWeekClaimed [ 'Tuesday' 'Monday' 'Thursday' 'Friday' 'Wednesday' ]
```




```
FraudFound_P = fraud.groupby('FraudFound_P').agg("count").reset_index()
VehiclePrice= fraud.groupby('VehiclePrice').agg("count").reset_index()
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(10,4))
grph1=sns.barplot(x='FraudFound_P', y="PolicyNumber", data = FraudFound_P , ax=ax1)
grph2= sns.barplot(x='VehiclePrice', y="PolicyNumber", data = VehiclePrice, ax=ax2)
ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')
grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
```

```
[Text(0, 0, '20000 to 29000'),
Text(0, 0, '30000 to 39000'),
Text(0, 0, '40000 to 59000'),
Text(0, 0, '60000 to 69000'),
Text(0, 0, 'less than 20000'),
Text(0, 0, 'more than 69000')]
```



For column FraudFound_P that is the target variable, the distribution is inconsistent so, the data will be balanced before proceeding with the modeling.

For column VehiclePrice, the bucketed range will be converted to integer and mean value of the bucket range will be fed as value for the respective range.



```
[128] pricevehicle = {"more than 69000" : 79000, "20000 to 29000" : 24500, "30000 to 39000" : 34500,
                    "less than 20000" : 14500, "40000 to 59000" : 49500, "60000 to 69000" : 64500}
fraud["VehiclePrice"] = fraud['VehiclePrice'].apply(lambda x: pricevehicle[x])
```



```
print("VehiclePrice", fraud["VehiclePrice"].unique())
```

```
VehiclePrice [79000 24500 34500 14500 49500 64500]
```



Bucketing Make column into two categories that is Luxury and Non-Luxury into a new column named as Grouped_Make.

```

[139] fraud["Grouped_Make"] = fraud["Make"]

[140] grp_make = {'Honda' : 'Non-Luxury', 'Toyota' : 'Non-Luxury', 'Ford' : 'Non-Luxury', 'Mazda' : 'Non-Luxury',
              'Chevrolet' : 'Non-Luxury', 'Pontiac' : 'Non-Luxury', 'Accura' : 'Luxury', 'Dodge' : 'Non-Luxury',
              'Mercury' : 'Luxury', 'Jaguar' : 'Luxury', 'Nissan' : 'Non-Luxury', 'VW' : 'Non-Luxury', 'Saab' : 'Luxury',
              'Saturn' : 'Non-Luxury', 'Porche' : 'Luxury', 'BMW' : 'Luxury', 'Mecedes' : 'Luxury',
              'Ferrari' : 'Luxury', 'Lexus' : 'Luxury'}
fraud["Grouped_Make"] = fraud['Grouped_Make'].apply(lambda x: grp_make[x])

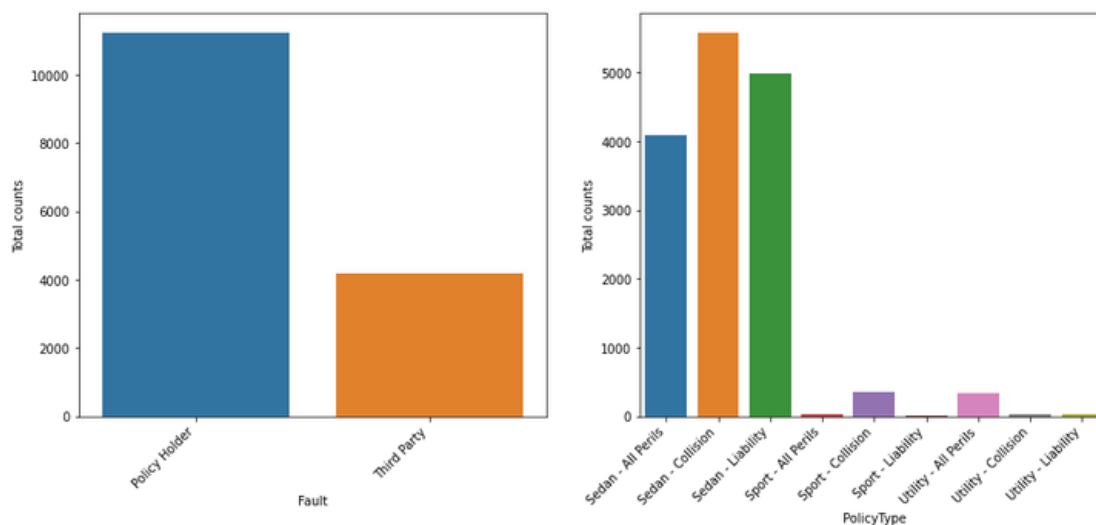
fraud["Grouped_Make"].unique()

array(['Non-Luxury', 'Luxury'], dtype=object)

```

```
[85] Fault = fraud.groupby('Fault').agg("count").reset_index()
PolicyType= fraud.groupby('PolicyType').agg("count").reset_index()
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(15,6))
grph1=sns.barplot(x='Fault', y="PolicyNumber", data = Fault , ax=ax1)
grph2= sns.barplot(x='PolicyType', y="PolicyNumber", data = PolicyType, ax=ax2)
ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')
grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
```

```
[Text(0, 0, 'Sedan - All Perils'),
Text(0, 0, 'Sedan - Collision'),
Text(0, 0, 'Sedan - Liability'),
Text(0, 0, 'Sport - All Perils'),
Text(0, 0, 'Sport - Collision'),
Text(0, 0, 'Sport - Liability'),
Text(0, 0, 'Utility - All Perils'),
Text(0, 0, 'Utility - Collision'),
Text(0, 0, 'Utility - Liability')]
```



It is observed that PolicyType column is formed from the concatenation of the VehicleCategory and BasePolicy columns. The distribution of these columns is presented further in the document.

```

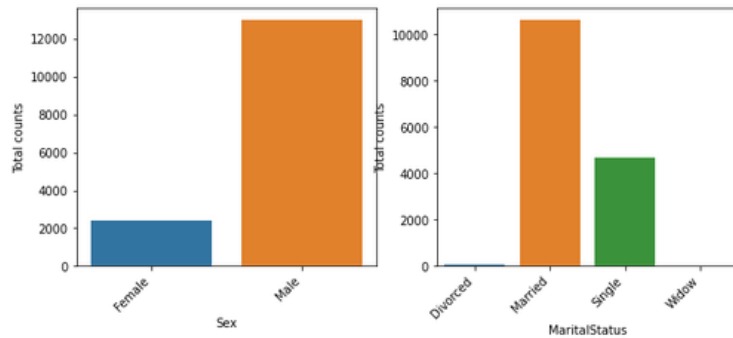
Sex = fraud.groupby('Sex').agg("count").reset_index()
MaritalStatus = fraud.groupby('MaritalStatus').agg("count").reset_index()
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
grph1 = sns.barplot(x='Sex', y="PolicyNumber", data = Sex, ax=ax1)
grph2 = sns.barplot(x='MaritalStatus', y="PolicyNumber", data = MaritalStatus, ax=ax2)
ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')
grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')

```

```

[Text(0, 0, 'Divorced'),
 Text(0, 0, 'Married'),
 Text(0, 0, 'Single'),
 Text(0, 0, 'Widow')]

```



```

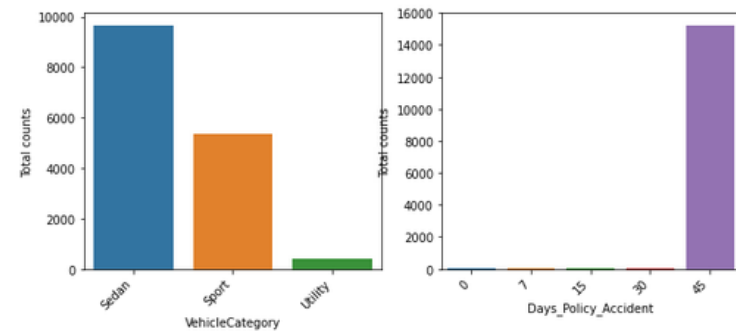
[88] VehicleCategory = fraud.groupby('VehicleCategory').agg("count").reset_index()
Days_Policy_Accident = fraud.groupby('Days_Policy_Accident').agg("count").reset_index()
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
grph1 = sns.barplot(x='VehicleCategory', y="PolicyNumber", data = VehicleCategory, ax=ax1)
grph2 = sns.barplot(x='Days_Policy_Accident', y="PolicyNumber", data = Days_Policy_Accident, ax=ax2)
ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')
grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')

```

```

[Text(0, 0, '0'),
 Text(0, 0, '7'),
 Text(0, 0, '15'),
 Text(0, 0, '30'),
 Text(0, 0, '45')]

```



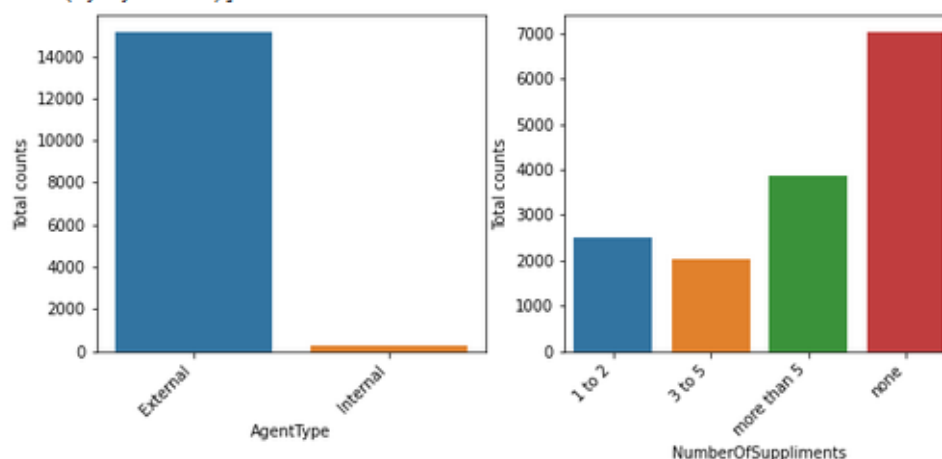


It Can be seen that there are many variables where the distribution of unique values are not balanced that may affect our analysis in the future. So, it is decided to use the SMOTE function in order to balance the dataset and make the data a good fit for the analysis.

SMOTE: SMOTE stands for Synthetic Minority Oversampling Technique used to increase the number of cases in the dataset in a balanced way. It works by generating new instances from existing ones that we supply as input (MLM, 2020)

```
[92] AgentType = fraud.groupby('AgentType').agg("count").reset_index()
      NumberOfSuppliments= fraud.groupby('NumberOfSuppliments').agg("count").reset_index()
      fig, (ax1,ax2) = plt.subplots(1,2,figsize=(10,4))
      grph1=sns.barplot(x='AgentType', y="PolicyNumber", data = AgentType , ax=ax1)
      grph2= sns.barplot(x='NumberOfSuppliments', y="PolicyNumber", data = NumberOfSuppliments, ax=ax2)
      ax2.set(ylabel='Total counts')
      ax1.set(ylabel='Total counts')
      grph1.set_xticklabels(grph1.get_xticklabels(),
                           rotation=45,
                           horizontalalignment='right')
      grph2.set_xticklabels(grph2.get_xticklabels(),
                           rotation=45,
                           horizontalalignment='right')
```

```
[Text(0, 0, '1 to 2'),
 Text(0, 0, '3 to 5'),
 Text(0, 0, 'more than 5'),
 Text(0, 0, 'none')]
```



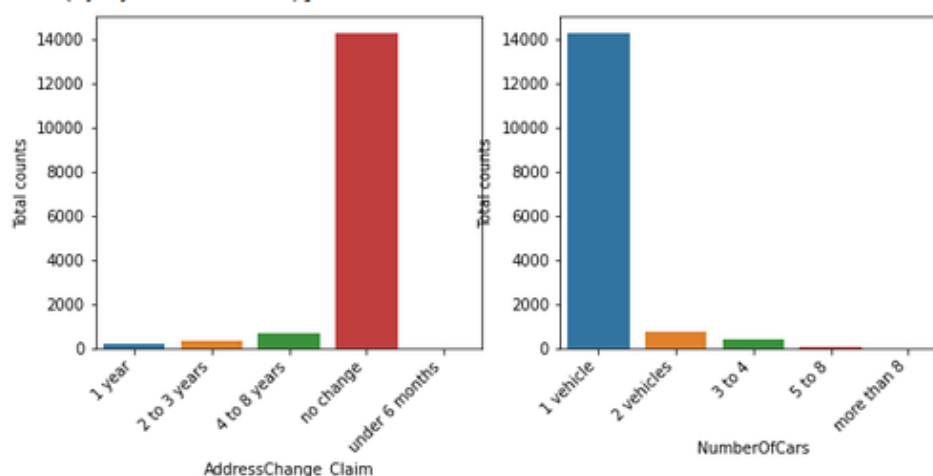
For column NumberOfSuppliments, the bucketed range will be converted to integer and minimum value of the bucket range will be fed as value for the respective range.

```
[94] Nsupplmnt = {"none" : 0, "more than 5" : 6, "3 to 5" : 3, "1 to 2" : 1}
      fraud["NumberOfSuppliments"] = fraud['NumberOfSuppliments'].apply(lambda x: Nsupplmnt[x])
```

```
fraud["NumberOfSuppliments"].unique()
array([0, 6, 3, 1])
```

```
[96] AddressChange_Claim = fraud.groupby('AddressChange_Claim').agg("count").reset_index()
NumberOfCars= fraud.groupby('NumberOfCars').agg("count").reset_index()
fig, (ax1,ax2) = plt.subplots(1,2,figsize=(10,4))
grph1=sns.barplot(x='AddressChange_Claim', y="PolicyNumber", data = AddressChange_Claim , ax=ax1)
grph2= sns.barplot(x='NumberOfCars', y ="PolicyNumber", data = NumberOfCars, ax=ax2)
ax2.set(ylabel='Total counts')
ax1.set(ylabel='Total counts')
grph1.set_xticklabels(grph1.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
grph2.set_xticklabels(grph2.get_xticklabels(),
                      rotation=45,
                      horizontalalignment='right')
```

```
[Text(0, 0, '1 vehicle'),
Text(0, 0, '2 vehicles'),
Text(0, 0, '3 to 4'),
Text(0, 0, '5 to 8'),
Text(0, 0, 'more than 8')]
```



For columns AddressChange_Claim and NumberOfCars, the bucketed range will be converted to integer and mean value of the bucket range will be fed as value for AddressChange_claim column and minimum value will be fed as value for NumberOfCars column, respectively.

```
[98] adresschn = {"1 year" : 1, "no change" : 0, "4 to 8 years" : 6, "2 to 3 years" : 2.5, "under 6 months" : 0.5}
fraud["AddressChange_Claim"] = fraud['AddressChange_Claim'].apply(lambda x: adresschn[x])

[99] fraud["AddressChange_Claim"].unique()

array([1. , 0. , 6. , 2.5, 0.5])
```

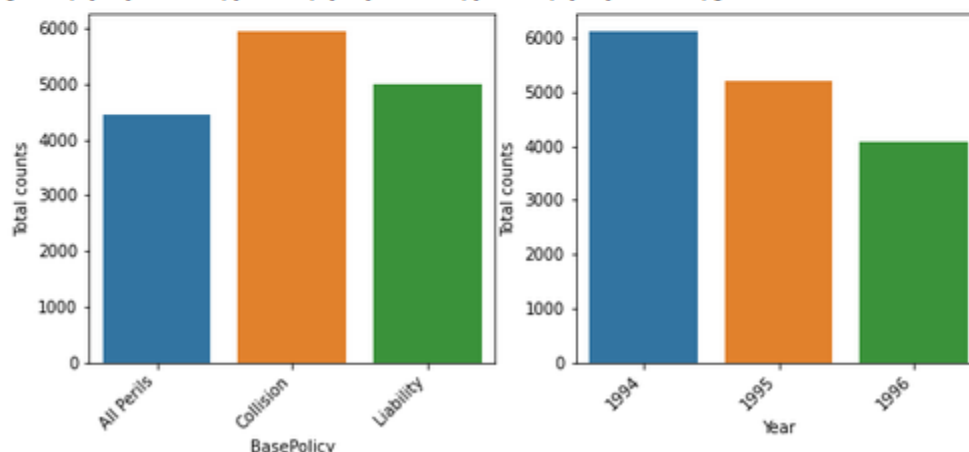
```
✓ [101] NCars = {"3 to 4" : 3.5, "1 vehicle" : 1, "2 vehicles" : 2, "5 to 8" : 6.5, "more than 8" : 9}
      fraud["NumberOfCars"] = fraud["NumberOfCars"].apply(lambda x: NCars[x])
```

```
✓ fraud["NumberOfCars"].unique()

array([3.5, 1. , 2. , 6.5, 9. ])
```

```
✓ [102] BasePolicy = fraud.groupby('BasePolicy').agg("count").reset_index()
      Year = fraud.groupby('Year').agg("count").reset_index()
      fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(10, 4))
      grph1 = sns.barplot(x='BasePolicy', y='PolicyNumber', data=BasePolicy, ax=ax1)
      grph2 = sns.barplot(x='Year', y='PolicyNumber', data=Year, ax=ax2)
      ax2.set(ylabel='Total counts')
      ax1.set(ylabel='Total counts')
      grph1.set_xticklabels(grph1.get_xticklabels(),
                           rotation=45,
                           horizontalalignment='right')
      grph2.set_xticklabels(grph2.get_xticklabels(),
                           rotation=45,
                           horizontalalignment='right')
```

```
[Text(0, 0, '1994'), Text(0, 0, '1995'), Text(0, 0, '1996')]
```



Dropping the following columns in order to select the best data that will be fed to the models.

```
✓ [143] fraud.drop(columns = ["PolicyNumber"], inplace = True)
```

```
✓ fraud.drop(columns = ["PolicyType"], inplace = True)
```


11.5 Skewness

```

fraud.skew(axis=None, skipna=None, level=None, numeric_only=True)

WeekOfMonth          0.115426
WeekOfMonthClaimed    0.158233
Age                  0.620465
VehiclePrice         1.585149
FraudFound_P         3.711164
PolicyNumber         0.000000
RepNumber            0.006628
Deductible           6.078803
DriverRating          0.009283
Days_Policy_Accident -10.822233
Days_Policy_Claim    -17.052852
PastNumberOfClaims    0.039177
AgeOfVehicle         -1.874884
NumberOfSupplements   0.743913
AddressChange_Claim   4.172563
NumberOfCars          5.636080
Year                  0.245689
dtype: float64

```

11.6 Measures of Central tendency and dispersion

↑ ↓ ↺ ⚙ 🗑 ⋮

```
fraud.describe()
```

	WeekOfMonth	WeekOfMonthClaimed	Age	VehiclePrice	FraudFound_P	PolicyNumber	RepNumber	Deductible	DriverRating	Days_Policy_Accident	Days_Policy_Claim
count	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000
mean	2.788586	2.693969	40.198119	34701.880674	0.059857	7710.500000	8.483268	407.704280	2.487808	44.650324	44.901751
std	1.287585	1.259115	12.660221	19248.743890	0.237230	4451.514911	4.599948	43.950998	1.119453	3.512465	1.471262
min	1.000000	1.000000	16.000000	14500.000000	0.000000	1.000000	1.000000	300.000000	1.000000	0.000000	0.000000
25%	2.000000	2.000000	31.000000	24500.000000	0.000000	3855.750000	5.000000	400.000000	1.000000	45.000000	45.000000
50%	3.000000	3.000000	38.000000	24500.000000	0.000000	7710.500000	8.000000	400.000000	2.000000	45.000000	45.000000
75%	4.000000	4.000000	48.000000	34500.000000	0.000000	11565.250000	12.000000	400.000000	3.000000	45.000000	45.000000
max	5.000000	5.000000	80.000000	79000.000000	1.000000	15420.000000	16.000000	700.000000	4.000000	45.000000	45.000000

🔗

PastNumberOfClaims	AgeOfVehicle	NumberOfSupplements	AddressChange_Claim	NumberOfCars	Year
15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000
2.306291	6.605772	2.058495	0.303859	1.114818	1994.866472
1.965465	1.399658	2.475815	1.228803	0.482791	0.803313
0.000000	1.000000	0.000000	0.000000	1.000000	1994.000000
0.000000	6.000000	0.000000	0.000000	1.000000	1994.000000
1.000000	7.000000	1.000000	0.000000	1.000000	1995.000000
4.000000	8.000000	6.000000	0.000000	1.000000	1996.000000
5.000000	8.000000	6.000000	6.000000	9.000000	1996.000000

Mode

fraud.mode()

Month	WeekOfMonth	DayOfWeek	Make	AccidentArea	DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	Age	Fault	PolicyType	VehicleCategory	
0	Jan	3.0	Monday	Pontiac	Urban	Monday	Jan	2.0	Male	Married	30.0	Policy Holder	Sedan - Collision	Sedan

fraud.mode()

FraudFound_P	PolicyNumber	RepNumber	Deductible	DriverRating	Days_Policy_Accident	Days_Policy_Claim	PastNumberOfClaims	AgeOfVehicle	AgeOfPolicyHolder	PoliceReportFiled
0.0	1	7.0	400.0	1.0	45.0	45.0	4.0	7.0	31 to 35	No

WitnessPresent	AgentType	NumberOfSuppliments	AddressChange_Claim	NumberOfCars	Year	BasePolicy	Grouped_Make
No	External	0.0	0.0	1.0	1994.0	Collision	Non-Luxury

Three Standard deviation above mean

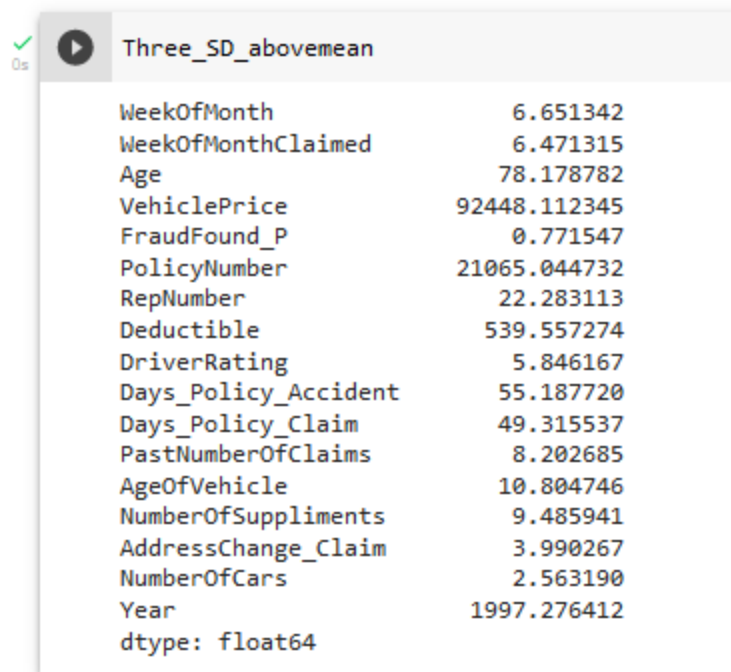
```
[56] fraud.std()

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None')

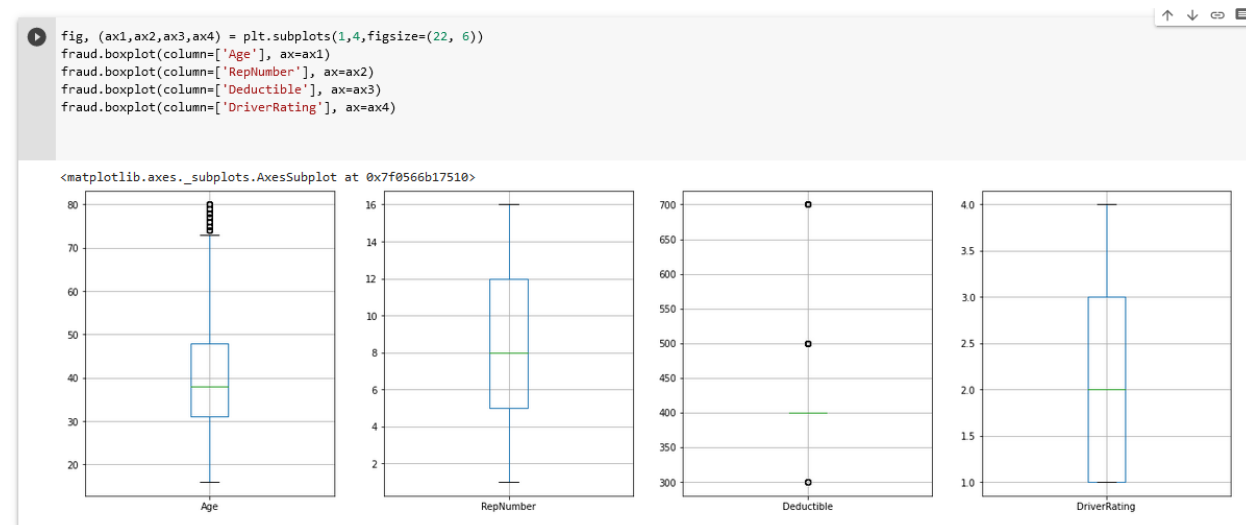
WeekOfMonth          1.287585
WeekOfMonthClaimed    1.259115
Age                  12.660221
VehiclePrice         19248.743890
FraudFound_P          0.237230
PolicyNumber         4451.514911
RepNumber             4.599948
Deductible           43.950998
DriverRating          1.119453
Days_Policy_Accident   3.512465
Days_Policy_Claim      1.471262
PastNumberOfClaims     1.965465
AgeOfVehicle          1.399658
NumberOfSuppliments    2.475815
AddressChange_Claim    1.228803
NumberOfCars           0.482791
Year                  0.803313
dtype: float64
```

```
[57] Three_SD_abovemean = 3*fraud.std() + fraud.mean()

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:1: FutureWarning:
Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None')
```



Box Plot of some variables to check for outliers



11.7 Changing datatypes of some columns to Categories

```

✓ [6] fraud.RepNumber = fraud.RepNumber.astype("category")
Os
✓ [7] fraud.Year = fraud.Year.astype("category")
Os
✓ [8] fraud.WeekOfMonth = fraud.WeekOfMonth.astype("category")
    fraud.WeekOfMonthClaimed = fraud.WeekOfMonthClaimed.astype("category")
Os

```

12.0 Summary of the EDA Performed

1. Replacing zero in columns DayOfWeekClaimed and MonthClaimed with Monday and Aug, respectively.
2. Replacing zeros in Age column with the mean value of its bucket in AgeOfPolicyHolder, which comes out to be 16.5.
3. Some of the values in Age column doesn't belong to its respective buckets in AgeOfPolicyHolder column. So, it is assumed that the person who met with the accident while driving the car is not the policyholder.
4. Converted the buckets in object columns Days_Policy_Accident, Days_Policy_Claim, PastNumberOfClaims, AgeOfVehicle into integer by taking the mode value of each bucketed range for its respective value.
5. Converted the buckets in object columns VehiclePrice, AddressChange_Claim into integer by taking the mean value of each bucketed range for its respective value.
6. Converted the buckets in object columns NumberOfSupplements, NumberOfCars into integer by taking the minimum value of each bucketed range for its respective value.
7. Observed through data visualization that DayOfWeekClaimed column have less than 1% of the proportion for days Saturday and Sunday. Also, it is believed that weekends are

normally holidays for the companies, so, decided to combine Saturdays and Sundays with Monday to get a better distribution for the column.

8. Added a column named Grouped_Make to bucket the Make column into two categories Luxury and Non-Luxury.
9. While looking for skewness of the variables it was observed that some of the variables are highly skewed because the data is imbalanced. Therefore, the data needed be consistent and balanced. Thus, smote function will be used to balance the data.
10. After checking for correlation, decided to drop columns PolicyNumber as it has very high correlation with Year column. Also, PolicyNumber column consists of serial number from 1 to 15420. PolicyType column was also dropped as the column is merely a concatenation of the columns VehicleCategory and BasePolicy.
11. Converted RepNumber, Year, WeekOfMonth, WeekOfMonthClaimed to categories as after converting the columns to categories the accuracy of the model increases and found to be the best.

Modelling

1.0 Modelling

1.1 Libraries for Modelling

Importing necessary libraries, rest some other libraries will be imported during modelling as per the need of the models.

```
%matplotlib inline

from pathlib import Path

import pandas as pd
import numpy as np
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
import matplotlib.pyplot as plt
!pip install dmbs
from dmbs import plotDecisionTree, classificationSummary, regressionSummary

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>
Collecting dmbs
 Downloading dmbs-0.1.0-py3-none-any.whl (11.8 MB)
 |████████████████████| 11.8 MB 4.1 MB/s
Installing collected packages: dmbs
Successfully installed dmbs-0.1.0
no display found. Using non-interactive Agg backend

1.2 Creating Dummies

```
frauddf = pd.get_dummies(fraud, drop_first=True)
print(frauddf.shape)
frauddf.head()
```

(15420, 109)

	Age	VehiclePrice	FraudFound_P	Deductible	DriverRating	Days_Policy_Accident	Days_Policy_Claim	PastNumberOfClaims
0	21.0	79000	0	300	1	45	45	0
1	34.0	79000	0	400	4	45	45	0
2	47.0	79000	0	400	3	45	45	1
3	65.0	24500	0	400	2	45	45	1
4	27.0	79000	0	400	1	45	45	0

Dummy variable is the one that takes only the value 0 or 1 to indicate the absence or presence of some categorical effect that may be expected to shift the outcome. They can be thought of as numeric stand-ins for categorical facts. (Wikipedia, n.d.)

1.3 Up-sampling/Balancing the data

The distribution target variable FraudFound_P is imbalanced. It constitutes of 923 transactions of 1 (fraud found) and 14497 transactions of 0 (fraud not found).

```
✓ [78] fraud.FraudFound_P.value_counts()
0s
      0      14497
      1         923
      Name: FraudFound_P, dtype: int64
```

SMOTE function will be used to balance the data and balance the FraudFound_P variable.

The following codes will import the SMOTE function, select the X_up and y_up variables, will run the smote function to upsample the data. Also, in the end it will print value counts of target variable which is FraudFound_P.

```
✓ [12] from imblearn.over_sampling import SMOTE
0s
```

```
✓ [13] X_up = frauddf.drop(columns = ['FraudFound_P'])
0s      y_up = frauddf['FraudFound_P']
```

```
✓ [14] sm = SMOTE(random_state=42)
0s      X_res, y_res = sm.fit_resample(X_up, y_up)

      df_smote_over = pd.concat([pd.DataFrame(X_res), pd.DataFrame(y_res)], axis = 1)
```

```
✓ [15] y_res.value_counts()
0s
      0      14497
      1      14497
      Name: FraudFound_P, dtype: int64
```


df_smote_over

	Age	VehiclePrice	Deductible	DriverRating	Days_Policy_Accident	Days_Policy_Claim	PastNumberOfClaims
0	21.000000	79000	300	1	45	45	0
1	34.000000	79000	400	4	45	45	0
2	47.000000	79000	400	3	45	45	1
3	65.000000	24500	400	2	45	45	1
4	27.000000	79000	400	1	45	45	0
...
28989	30.390941	24500	400	4	45	45	4
28990	56.607082	24500	400	1	45	45	0
28991	65.780550	24500	400	3	45	45	0
28992	46.545640	79000	400	1	45	45	0
28993	34.547280	14500	400	3	45	45	0

28994 rows × 109 columns

1.4 Train test split

Train test split is performed to prevent the model from overfitting and to accurately evaluate the model.

```

X = df_smote_over.drop(columns = ['FraudFound_P'])
y = df_smote_over['FraudFound_P']
X_train, X_valid, y_train, y_valid = train_test_split(X, y, test_size = 0.4, random_state = 1)

```

2.0 Decision Tree

One of the effective data analytics methods that can accommodate complicated datasets is the decision tree. It is a flexible non-parametric supervised machine learning technique that, unlike other supervised models, can handle multinomial classification output analysis as well as classification and regression problems (KDnuggets, 2022). As this is a classification problem, we are going to use classification decision tree.

Classification Tree

A classification tree is a type of tree model where the target variable might have a defined range of values. Each node or step in this tree has a condition applied to it in order to completely segregate all of the labels or classes found in the dataset.

Working of classification decision tree

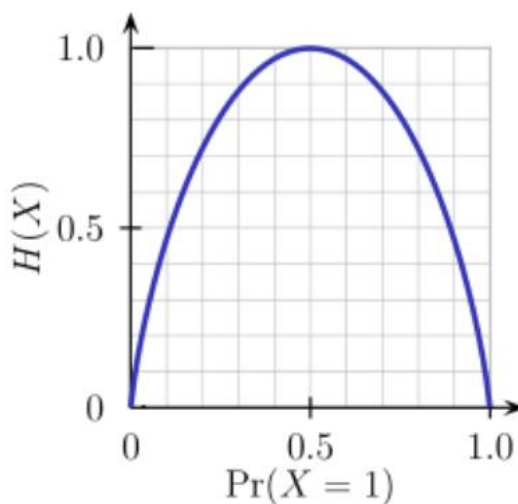
We must comprehend a few basic factors that are crucial to understanding how a decision tree functions.

1.0 Entropy: It measures the impurity and randomness in the data (tds, 2017)

Formula for Entropy = $-P(\text{class1}) * \log(P(\text{class1})) - P(\text{class2}) * \log(P(\text{class2}))$, where P is probability.

According to the formula, the likelihood of correctly predicting either class 1 or class 2 is only 50% given that the dataset contains equal numbers of each class. Entropy will be high as a result. Entropy will also be 0 if the dataset solely contains instances of class 2. It follows that a dataset that is impure or mixed has a high entropy value.

It is clear from the graph that when the probability is either 0 or 1, the entropy $H(X)$ is zero. When the probability is 0.5, which portrays complete randomness in the data and eliminates the possibility of properly predicting the outcome, the entropy is at its highest level. (KDnuggets, 2022)



1.1 Information Gain: Information gain is the decrease in impurity or entropy after the dataset splits based on an attribute. Finding qualities that provide significant information benefits is

essential for decision tree development. This helps in deciding which feature may be used to build an internal node at a specific position (tds, 2017).

Formula for Information Gain = Entropy (s) – [(Weighted average) * (Entropy of each feature)]

1.2 Gini Index: The dataset's impurity is measured by Gini Index. Gini Index is used in constructing a decision tree with the help of CART mechanism. (KDnuggets, 2022)

The formula for Gini Impurity = $1 - (\text{Probability of class 1})^2 - (\text{Probability of class 2})^2$

The procedure for choosing the right value to use when creating a condition. First, the dataset is sorted in ascending order according to the variables' numerical values. The average of the subsequent pairs of numerical values will be discovered. The average value will then be fitted into the condition to see which variable, given the approach we used to construct the decision tree, will offer the lowest Gini Impurity index or the most information gain. Similar to discrete value-based or class-based features, the condition is established when all values fit inside the set. The set or circumstance that results in the lowest Gini Impurity index will be chosen.

Running a full decision tree with all the nodes. (KDnuggets, 2022).

Two decision trees will be built in this project. First, the decision tree with maximum depth and decision tree with maximum depth as 5. Moreover, we are only using classification decision tree as this is a classification problem.

2.1 Full Decision Tree

The following are the codes to build a full decision tree.

✓ [21] `from sklearn import metrics`

✓ [22] `classtreefull = DecisionTreeClassifier(random_state=1)`
`fulltree = classtreefull.fit(X_train,y_train)`

✓ `y_pred_gini = fulltree.predict(X_valid)`

✓ `print('Gini stats')`
`print("Accuracy:",metrics.accuracy_score(y_valid, y_pred_gini))`
`print("balanced_accuracy:",metrics.balanced_accuracy_score(y_valid, y_pred_gini))`
`print("brier_score_loss:",metrics.brier_score_loss(y_valid, y_pred_gini))`
`print("f1_score:",metrics.f1_score(y_valid,y_pred_gini))`
`print("recall_score:",metrics.recall_score(y_valid, y_pred_gini))`
`print("precision_score:",metrics.precision_score(y_valid, y_pred_gini))`
`print("roc_auc_score:",metrics.roc_auc_score(y_valid, y_pred_gini))`

Gini stats
 Accuracy: 0.9282634937058113
 balanced_accuracy: 0.9286144874798528
 brier_score_loss: 0.07173650629418865
 f1_score: 0.9283746556473829
 recall_score: 0.9474609031804604
 precision_score: 0.910042194092827
 roc_auc_score: 0.9286144874798526

✓ [28] `classificationSummary(y_train,classtreefull.predict(X_train))`

Confusion Matrix (Accuracy 1.0000)

	Prediction	
Actual	0	1
0	8590	0
1	0	8806

✓ `classificationSummary(y_valid,classtreefull.predict(X_valid))`

Confusion Matrix (Accuracy 0.9283)

	Prediction	
Actual	0	1
0	5374	533
1	299	5392

It can be observed from the classification summary of both train and valid dataset the model is overfit as validation accuracy is less than training accuracy.

```

y_pred_proba = fulltree.predict_proba(X_valid)[::,1]

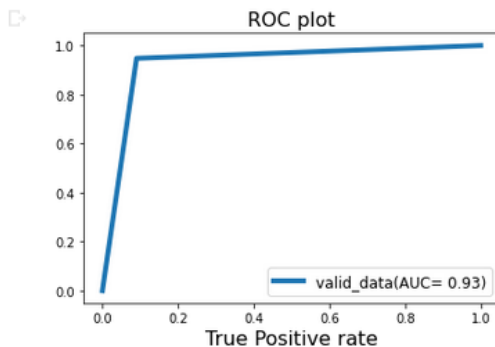
auc = metrics.roc_auc_score(y_valid, y_pred_gini)

fpr1, tpr1, _ = metrics.roc_curve(y_valid, y_pred_proba)

plt.plot(fpr1, tpr1, label= "valid_data(AUC= %0.2f)" % auc, linewidth = 4)

plt.legend(prop={'size' : 12}, loc = "lower right")
plt.title('ROC plot', fontsize = 16)
plt.xlabel('False Positive rate', fontsize = 16)
plt.ylabel('True Positive rate', fontsize = 16)
plt.show()

```



2.2 Decision Tree with max depth 5

The following are the codes to build decision tree with max depth 5.

```

[30] classtree = DecisionTreeClassifier(random_state=1, max_depth=5)
tree = classtree.fit(X_train,y_train)

```

```

[31] y_pred_gini_tree = tree.predict(X_valid)

```

```

[32] print('Gini stats')
print("Accuracy:",metrics.accuracy_score(y_valid, y_pred_gini_tree))
print("balanced_accuracy:",metrics.balanced_accuracy_score(y_valid, y_pred_gini_tree))
print("brier_score_loss:",metrics.brier_score_loss(y_valid, y_pred_gini_tree))
print("f1_score:",metrics.f1_score(y_valid,y_pred_gini_tree))
print("recall_score:",metrics.recall_score(y_valid, y_pred_gini_tree))
print("precision_score:",metrics.precision_score(y_valid, y_pred_gini_tree))
print("roc_auc_score:",metrics.roc_auc_score(y_valid, y_pred_gini_tree))

```

```

Gini stats
Accuracy: 0.8321262286601138
balanced_accuracy: 0.83414498260197
brier_score_loss: 0.16787377133988618
f1_score: 0.8463905325443787
recall_score: 0.9425408539799683
precision_score: 0.7680412371134021
roc_auc_score: 0.8341449826019699

```

```
✓ [34] classificationSummary(y_train,classtree.predict(X_train))
```

Confusion Matrix (Accuracy 0.8407)

	Prediction	
Actual	0	1
0	6276	2314
1	458	8348

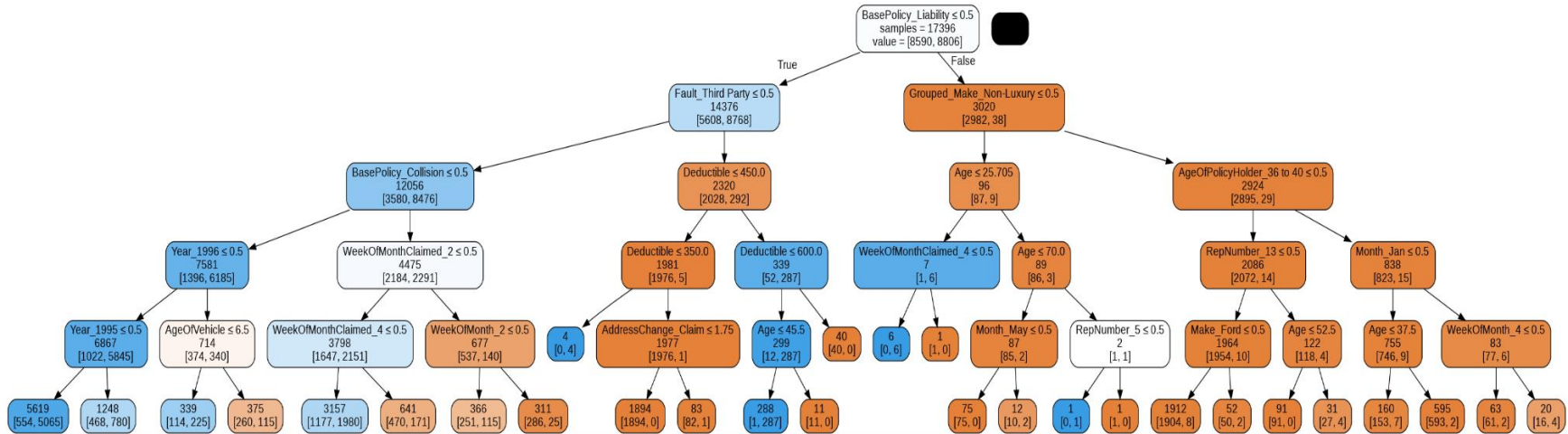
```
✓ [35] classificationSummary(y_valid,classtree.predict(X_valid))
```

Confusion Matrix (Accuracy 0.8321)

	Prediction	
Actual	0	1
0	4287	1620
1	327	5364

From the above accuracy scores, it can be concluded that this model is slightly overfit as the validation dataset accuracy is less than the train dataset.

```
plotDecisionTree(classtree, feature_names = X_train.columns)
```



The following are the codes for plotting ROC curve

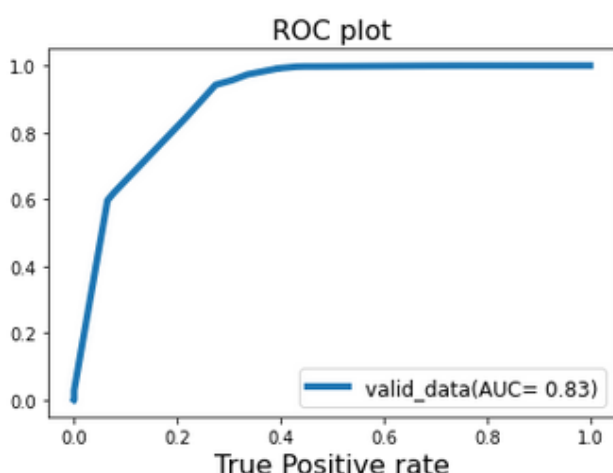
```
[66] y_pred_proba_tree= classtree.predict_proba(X_valid)[::,1]

auc_tree = metrics.roc_auc_score(y_valid, y_pred_gini_tree)

fpr2, tpr2, _ = metrics.roc_curve(y_valid, y_pred_proba_tree)

plt.plot(fpr2, tpr2, label= "valid_data(AUC= %0.2f)" % auc_tree, linewidth = 4)

plt.legend(prop={'size' : 12}, loc = "lower right")
plt.title('ROC plot', fontsize = 16)
plt.xlabel('False Positive rate', fontsize = 16)
plt.xlabel('True Positive rate', fontsize = 16)
plt.show()
```



2.3 Classification Decision Trees Comparison:

Both the trees seem to be overfit. However, the full decision tree with maximum depth seems to be a better tree than the tree with max depth 5. As the valid data of full decision tree is predicting only 299 false negatives whereas, the tree with max depth 5 is predicting 327 false negatives.

This analysis is concerned with predicting frauds. So, the importance will be given to model with lower number of false negatives. Thus, the full decision tree will be a better decision tree between the two.

3.0 Regression Models

A regression tree is a type of decision tree that allows the target variable to also have continuous values. When dealing with a nonlinear complex dataset, a regression tree is preferable over polynomial regression since there are occasions when the distribution is too complicated to examine and choose between the polynomials. A regression tree's leaves reflect continuous numeric values, unlike a classification tree. (tds, 2017)

3.1 Linear Regression Model

The relationship between the scalar answer and one or more dependent variables is modelled using a linear method. This approach for supervised machine learning enables modelling of the connection using linear predictor functions. The linear relationship between one or more variables may be modelled using linear regression. (Machine Learning Mastery, 2020)

The following are the codes for Linear regression of training dataset.

```
✓ [36] from sklearn.linear_model import LinearRegression
```

```
✓ [37] model = LinearRegression()
linearreg = model.fit(X_train,y_train)
```

```
✓ [38] train_pred = model.predict(X_train)
train_pred

array([0.18388755, 0.05033977, 0.16646946, ..., 0.14659587, 0.26012361,
       0.13540785])
```

```
✓ [41] regressionSummary(train_pred, y_train)
```

Regression statistics

```

                Mean Error (ME) : 0.0000
    Root Mean Squared Error (RMSE) : 0.2116
            Mean Absolute Error (MAE) : 0.1399
        Mean Percentage Error (MPE) : 31.1880
    Mean Absolute Percentage Error (MAPE) : 92.4730
```

The following are the codes for Linear regression of validation dataset.

```
[42] train_valid = model.predict(X_valid)
     train_valid
```

```
array([0.58644336, 0.83218376, 0.03684229, ..., 0.67869112, 0.16459815,
       0.29035727])
```

```
[43] regressionSummary(train_valid, y_valid)
```

Regression statistics

```

                Mean Error (ME) : 0.0060
Root Mean Squared Error (RMSE) : 0.2114
                Mean Absolute Error (MAE) : 0.1417
                Mean Percentage Error (MPE) : 42.2429
Mean Absolute Percentage Error (MAPE) : 87.3163
```

3.2 Forward Regression Model

Forward selection is a kind of stepwise regression that starts with a null model and gradually adds variables. You include the one variable that improves your model the most significantly in each step forward (Statistics How To, n.d.). The following are the codes for the same.

```
from dmbs import regressionSummary, classificationSummary
from dmbs import liftChart, gainsChart, adjusted_r2_score
from dmbs import exhaustive_search, backward_elimination, forward_selection, AIC_score, BIC_score
```

```
[46] def train_model(variables):
      if len(variables) == 0:
          return None
      model = LinearRegression()
      model.fit(X_train[variables], y_train)
      return model

      def score_model(model, variables):
          if len(variables) == 0:
              return AIC_score(y_train, [y_train.mean()] * len(y_train), model, df=1)
          return AIC_score(y_train, model.predict(X_train[variables]), model)

      Forward_model, Forward_variables = forward_selection(X_train.columns, train_model, score_model, verbose=True)
      print(Forward_variables)
```

```

Variables: Age, VehiclePrice, Deductible, DriverRating, Days_Policy_Accident, Days_Policy_Claim, PastNumberOfClaims,
Start: score=25253.05, constant
Step: score=21268.96, add BasePolicy_Liability
Step: score=18264.19, add Fault_Third Party
Step: score=16673.21, add BasePolicy_Collision
Step: score=16003.58, add WeekOfMonthsClaimed_2
Step: score=15219.18, add WeekOfMonthsClaimed_3
Step: score=13962.66, add WeekOfMonthsClaimed_4
Step: score=12858.07, add WeekOfMonthsClaimed_5
Step: score=12155.80, add WeekOfMonths_5
Step: score=11632.66, add WeekOfMonths_4
Step: score=11159.74, add WeekOfMonths_3
Step: score=10564.51, add WeekOfMonths_2
..
.....
```

```
[148] regressionSummary(y_train, Forward_model.predict(X_train[Forward_variables]))
```

Regression statistics

```

Mean Error (ME) : 0.0000
Root Mean Squared Error (RMSE) : 0.2116
Mean Absolute Error (MAE) : 0.1400

```

```
[149] regressionSummary(y_valid, Forward_model.predict(X_valid[Forward_variables]))
```

Regression statistics

```

Mean Error (ME) : -0.0060
Root Mean Squared Error (RMSE) : 0.2114
Mean Absolute Error (MAE) : 0.1417

```

3.3 Backward Regression Model

A backward regression model is a stepwise regression model that starts with a complete (saturated) model and gradually reduces variables at each step to obtain a condensed model that best describes the data. Likewise referred to as Backward Elimination regression.

The following are the codes for backward regression model.

```
def train_model(variables1):  
    model1 = LinearRegression()  
    model1.fit(X_train[variables1], y_train)  
    return model1  
  
def score_model(model1, variables1):  
    return AIC_score(y_train, model1.predict(X_train[variables1]), model1)  
  
backward_model, backward_variables = backward_elimination(X_train.columns, train_model1, score_model1, verbose=True)  
  
print(backward_variables)
```

Variables: Age, VehiclePrice, Deductible, DriverRating, Days_Policy_Accident, Days_Policy_Claim, PastNumberOfClaims, AgeOfVehicle

Start: score=-4446.50
Step: score=-4448.50, remove Sex_Male
Step: score=-4450.25, remove WitnessPresent_Yes
Step: score=-4451.99, remove VehiclePrice
Step: score=-4453.67, remove Days_Policy_Claim
Step: score=-4455.48, remove Days_Policy_Accident
Step: score=-4457.10, remove NumberOfCars
Step: score=-4458.73, remove Age
Step: score=-4460.23, remove Make_Lexus
Step: score=-4461.66, remove AgeOfPolicyHolder_21 to 25
Step: score=-4462.99, remove AgentType_Internal
Step: score=-4464.02, remove Make_BMW
Step: score=-4465.01, remove Make_Mercedes
Step: score=-4465.88, remove Make_Porsche
Step: score=-4465.88, remove None
['Deductible', 'DriverRating', 'PastNumberOfClaims', 'AgeOfVehicle', 'NumberOfSupplements', 'AddressChange_Claim', 'Month_Aug',]

```
✓ [151] regressionSummary(y_train, backward_model.predict(X_train[backward_variables])) |
```

```
Regression statistics
```

```
      Mean Error (ME) : -0.0000
Root Mean Squared Error (RMSE) : 0.2116
      Mean Absolute Error (MAE) : 0.1400
```

```
✓ [152] regressionSummary(y_valid, backward_model.predict(X_valid[backward_variables])) |
```

```
Regression statistics
```

```
      Mean Error (ME) : -0.0060
Root Mean Squared Error (RMSE) : 0.2114
      Mean Absolute Error (MAE) : 0.1417
```

3.4 Regression Model Comparison.

The Linear, forward, and backward regression models were performed. The best model among the three to be selected on the basis of Root Mean Squared Error (RMSE). After comparing the RMSE of three models it can be seen that all the three models have the same RMSE for valid and train dataset and all the three models are also good fit for modelling. Thus, all the three models predict with the same level of accuracy.

4.0 Logistic Regression Model

logistic regression is a statistical analysis technique which uses previous data set observations to predict a binary result, such as yes or no. By examining the relationship between one or more already present independent variables, a logistic regression model forecasts a dependent data variable.

The following are the codes to build a logistic regression model and checking the coefficients of each variable.

```
✓ [153] from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
```

```
✓ [154] logit_reg = LogisticRegression(solver='liblinear', C=1e42, random_state=1)
logit_reg.fit(X_train, y_train)
```

```
LogisticRegression(C=1e+42, random_state=1, solver='liblinear')
```

```
✓ [155] print(pd.DataFrame({'coef': logit_reg.coef_[0]}, index=X.columns))
```

```

coef
Age                -0.008672
VehiclePrice        0.000004
Deductible          0.004094
DriverRating        -0.050866
Days_Policy_Accident -0.026444
...                ...
Year_1995            -0.698958
Year_1996            -0.968373
BasePolicy_Collision -0.560331
BasePolicy_Liability -1.948701
Grouped_Make_Non-Luxury 0.007126
```

```
[108 rows x 1 columns]
```

```
✓ [132] Coefficient = pd.DataFrame({'coef': logit_reg.coef_[0]}, index=X.columns)
```

```
✓ [134] Best_coef = Coefficient.sort_values(by = "coef", ascending = False)
```

```
✓ [136] Best_coef.head(10)
```

	coef
Days_Policy_Claim	0.160240
AgeOfPolicyHolder_21 to 25	0.111443
NumberOfCars	0.058998
Sex_Male	0.009229
Grouped_Make_Non-Luxury	0.007126
Deductible	0.004094
AddressChange_Claim	0.001833
AgeOfPolicyHolder_18 to 20	0.001234
Make_Mercedes	0.000683
VehiclePrice	0.000004

Calculating the odds ratio

```
import math
print("Days_Policy_Claim_odds:", math.exp(0.160240))
print("AgeOfPolicyHolder_21to25_odds:", math.exp(0.111443))
print("NumberOfCars_odds:", math.exp(0.058998))
print("Sex_Male_odds:", math.exp(0.009229))
print("Grouped_Make_NonLuxury_odds:", math.exp(0.007126))
print("Deductible_odds:", math.exp(0.004094))
print("AddressChange_Claim_odds:", math.exp(0.001833))
print("AgeOfPolicyHolder_18to20_odds:", math.exp(0.001234))
print("Make_Mercedes_odds:", math.exp(0.000683))
print("VehiclePrice_odds:", math.exp(0.000004))
```

```
Days_Policy_Claim_odds: 1.1737925474006652
AgeOfPolicyHolder_21to25_odds: 1.1178900224582038
NumberOfCars_odds: 1.060773119191799
Sex_Male_odds: 1.0092717185358233
Grouped_Make_NonLuxury_odds: 1.0071514503551608
Deductible_odds: 1.004102391866192
AddressChange_Claim_odds: 1.0018346809714167
AgeOfPolicyHolder_18to20_odds: 1.001234761691277
Make_Mercedes_odds: 1.000683233297611
VehiclePrice_odds: 1.000004000008
```

```
[156] logit_reg_prob = logit_reg.predict_proba(X_valid)
```

```
[157] logit_reg_pred = logit_reg.predict(X_valid)
```

```
[98] print('Gini stats')
print("Accuracy:", metrics.accuracy_score(y_valid, logit_reg_pred))
print("balanced_accuracy:", metrics.balanced_accuracy_score(y_valid, logit_reg_pred))
print("brier_score_loss:", metrics.brier_score_loss(y_valid, logit_reg_pred))
print("f1_score:", metrics.f1_score(y_valid, logit_reg_pred))
print("recall_score:", metrics.recall_score(y_valid, logit_reg_pred))
print("precision_score:", metrics.precision_score(y_valid, logit_reg_pred))
print("roc_auc_score:", metrics.roc_auc_score(y_valid, logit_reg_pred))
```

```
Gini stats
Accuracy: 0.9558544576651147
balanced_accuracy: 0.9556046144514263
brier_score_loss: 0.044145542334885324
f1_score: 0.954432182271271
recall_score: 0.942189421894219
precision_score: 0.9669972948602344
roc_auc_score: 0.9556046144514264
```

```
✓ 0s ▶ classificationSummary(y_train, logit_reg.predict(X_train))
```

Confusion Matrix (Accuracy 0.9553)

	Prediction	
Actual	0	1
0	8338	252
1	526	8280

```
✓ 0s [85] classificationSummary(y_valid, logit_reg.predict(X_valid))
```

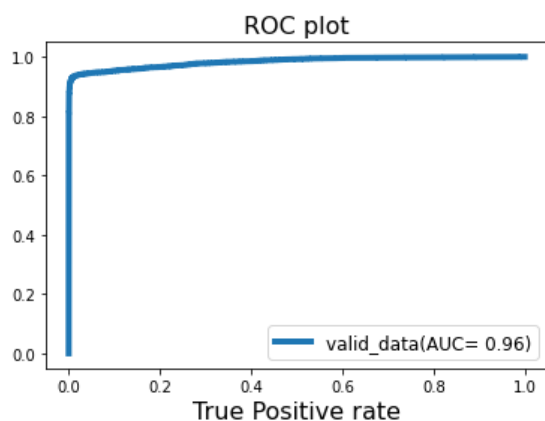
Confusion Matrix (Accuracy 0.9559)

	Prediction	
Actual	0	1
0	5724	183
1	329	5362

The accuracy of the model looks very good and confusion matrix seems better at forecasting future events. Therefore, will provide more value to the business.

The following are the codes for plotting ROC curve and Gains chart.

```
✓ 3s ▶ logit_reg_prob_roc = logit_reg.predict_proba(X_valid)[::,1]
auc_logit = metrics.roc_auc_score(y_valid, logit_reg_pred)
fpr3, tpr3, _ = metrics.roc_curve(y_valid, logit_reg_prob_roc)
plt.plot(fpr3, tpr3, label= "valid_data(AUC= %0.2f)" % auc_logit, linewidth = 4)
plt.legend(prop={'size' : 12}, loc = "lower right")
plt.title('ROC plot', fontsize = 16)
plt.xlabel('False Positive rate', fontsize = 16)
plt.ylabel('True Positive rate', fontsize = 16)
plt.show()
```



```
[83] logit_result = pd.DataFrame({'actual' : y_valid,
                                'p_0' : [p[0] for p in logit_reg_prob],
                                'p_1' : [p[1] for p in logit_reg_prob],
                                'predicted': logit_reg_pred})

logit_result
```

	actual	p_0	p_1	predicted
14810	0	0.388145	0.611855	1
26866	1	0.056372	0.943628	1
3713	0	0.966477	0.033523	0
24587	1	0.013667	0.986333	1
12719	0	0.968427	0.031573	0
...
19197	1	0.004583	0.995417	1
12024	1	0.884033	0.115967	0
21830	1	0.131061	0.868939	1
3597	0	0.910532	0.089468	0
14706	0	0.578878	0.421122	0

11598 rows × 4 columns

```
[85] !pip install dmbs
from dmbs import classificationSummary, gainsChart, liftChart
from dmbs.metric import AIC_score
```

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>
Requirement already satisfied: dmbs in /usr/local/lib/python3.7/dist-packages (0.1.0)

```
Gains = logit_result.sort_values(by=['p_1'], ascending=False)
Gains
```

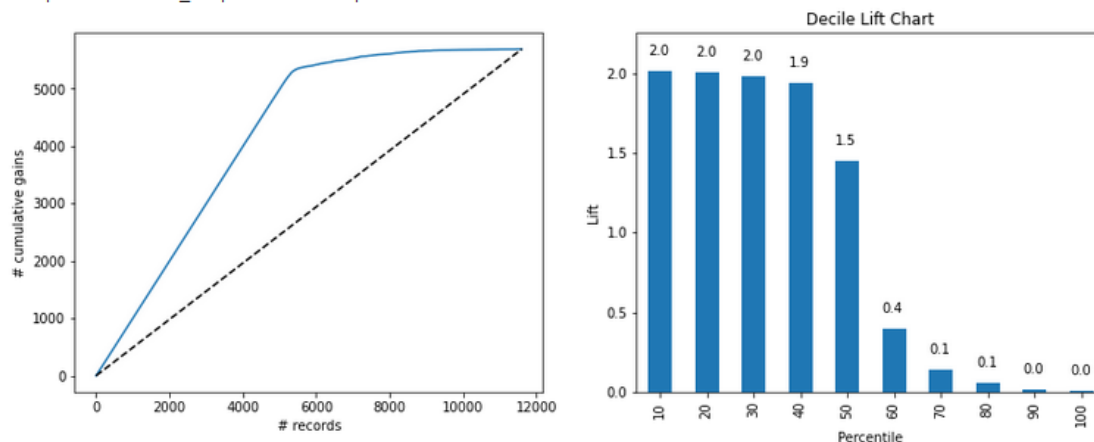
	actual	p_0	p_1	predicted
22406	1	0.000318	0.999682	1
16003	1	0.000390	0.999610	1
27773	1	0.000397	0.999603	1
28798	1	0.000450	0.999550	1
24501	1	0.000467	0.999533	1
...
11441	0	0.999879	0.000121	0
15112	0	0.999899	0.000101	0
13518	0	0.999900	0.000100	0
8243	0	0.999910	0.000090	0
13676	0	0.999914	0.000086	0

11598 rows × 4 columns

The following are the codes for plotting Gains chart and Decile Lift Chart.

```
[88] fig, axes = plt.subplots(1,2, figsize=(14,5))
      gainsChart(Gains.actual,ax=axes[0])
      liftChart(Gains['p_1'],ax=axes[1])

<matplotlib.axes._subplots.AxesSubplot at 0x7f236158f2d0>
```



5.0 Decision Tree-GridSearchCV

GridSearchCV is a method for tuning hyperparameters to find the best values for a particular model. As was already noted, a model's performance is strongly influenced by the value of its hyperparameters. Noting that there is no way to determine the optimum values for hyperparameters in advance, it is desirable to explore every potential value before deciding what the best ones are. We utilize GridSearchCV to automate the tweaking of hyperparameters because doing it manually may take a lot of time and resources. (Great Learning, 2020)

The following are the codes for Performing GridSearchCV

```
[89] param_grid = {
    'max_depth' : [2,3,5,10],
    'min_samples_split' : [0.07,0.05,0.01,0.005],
    'min_impurity_decrease' : [0.05, 0.02, 0.01, 0.001]
}

[90] gridsearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid, cv=5, n_jobs=-1)
gridsearch.fit(X_train,y_train)

GridSearchCV(cv=5, estimator=DecisionTreeClassifier(random_state=1), n_jobs=-1,
             param_grid={'max_depth': [2, 3, 5, 10],
                         'min_impurity_decrease': [0.05, 0.02, 0.01, 0.001],
                         'min_samples_split': [0.07, 0.05, 0.01, 0.005]})

gridsearch.best_score_

0.8872150307427488
```

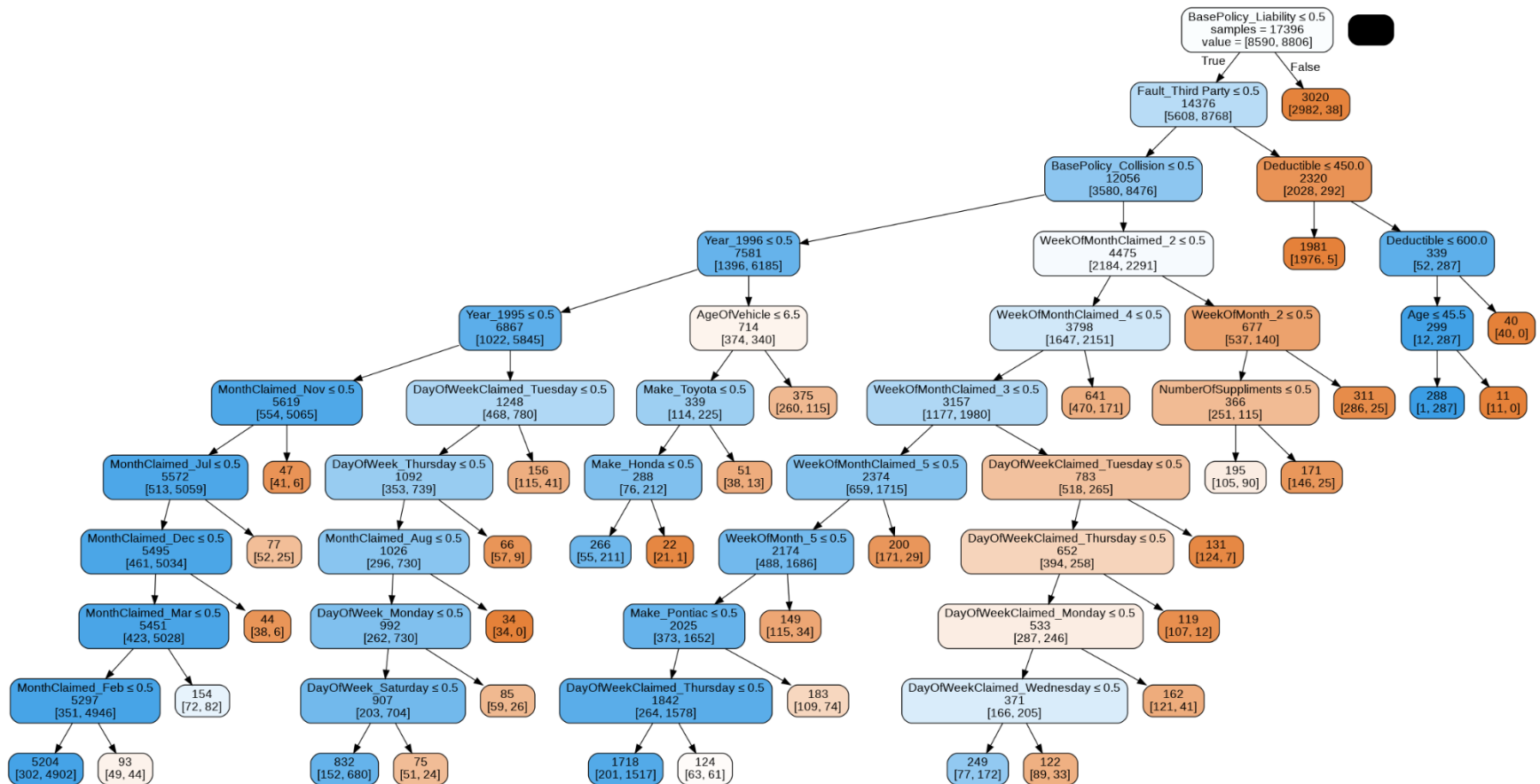
The following are the codes for GridSearch best estimator tree.

```
[93] gridsearch.best_estimator_

DecisionTreeClassifier(max_depth=10, min_impurity_decrease=0.001,
                      min_samples_split=0.005, random_state=1)

[94] gridsearchtree = gridsearch.best_estimator_
plotDecisionTree(gridsearchtree, feature_names = X_train.columns)
```

GridSearch Best Estimator Tree



```
[95] regressionSummary(y_train, gridsearchtree.predict(X_train))
```

Regression statistics

```

Mean Error (ME) : 0.0055
Root Mean Squared Error (RMSE) : 0.3230
Mean Absolute Error (MAE) : 0.1043

```

```
[96] regressionSummary(y_valid, gridsearchtree.predict(X_valid))
```

Regression statistics

```

                Mean Error (ME) : 0.0008
Root Mean Squared Error (RMSE) : 0.3342
                Mean Absolute Error (MAE) : 0.1117

```

```
[97] classificationSummary(y_train, gridsearchtree.predict(X_train))
```

Confusion Matrix (Accuracy 0.8957)

```

      Prediction
Actual    0    1
0  7730  860
1   955 7851

```

```
[100] classificationSummary(y_valid, gridsearchtree.predict(X_valid))
```

Confusion Matrix (Accuracy 0.8883)

```

      Prediction
Actual    0    1
0  5264  643
1   652 5039

```

From the above accuracy scores of gridsearchtree, it can be concluded that this model is slightly overfit as the validation dataset accuracy is less than the train dataset. Also, this model is predicting 652 false negatives which is quite a high number as compared to other models above.

The following are the codes for plotting ROC curve.

```

[110] y_pred_proba_grid= gridsearchtree.predict_proba(X_valid)[::,1]

      y_pred_gini_grid = gridsearchtree.predict(X_valid)

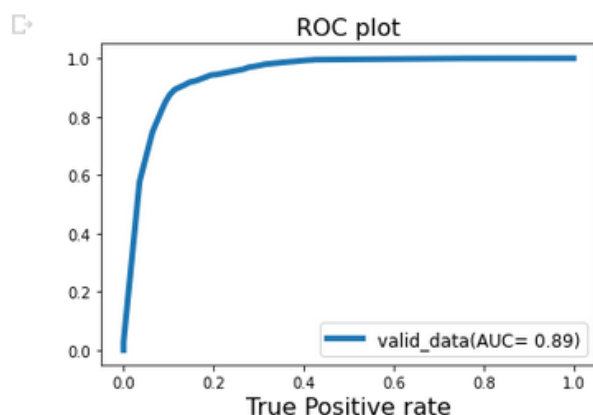
      auc_gridtree = metrics.roc_auc_score(y_valid, y_pred_gini_grid)

      fpr4, tpr4, _ = metrics.roc_curve(y_valid, y_pred_proba_grid)

      plt.plot(fpr4, tpr4, label= "valid_data(AUC= %.2f)" % auc_gridtree, linewidth = 4)

      plt.legend(prop={'size' : 12}, loc = "lower right")
      plt.title('ROC plot', fontsize = 16)
      plt.xlabel('False Positive rate', fontsize = 16)
      plt.xlabel('True Positive rate', fontsize = 16)
      plt.show()

```



```
print('Gini stats')
print("Accuracy:",metrics.accuracy_score(y_valid, y_pred_gini_grid))
print("balanced_accuracy:",metrics.balanced_accuracy_score(y_valid, y_pred_gini_grid))
print("brier_score_loss:",metrics.brier_score_loss(y_valid, y_pred_gini_grid))
print("f1_score:",metrics.f1_score(y_valid,y_pred_gini_grid))
print("recall_score:",metrics.recall_score(y_valid, y_pred_gini_grid))
print("precision_score:",metrics.precision_score(y_valid, y_pred_gini_grid))
print("roc_auc_score:",metrics.roc_auc_score(y_valid, y_pred_gini_grid))
```

Gini stats
Accuracy: 0.8883428177271944
balanced_accuracy: 0.8882896189478473
brier_score_loss: 0.11165718227280566
f1_score: 0.8861338257276005
recall_score: 0.8854331400456862
precision_score: 0.8868356212601197
roc_auc_score: 0.8882896189478472

6.0 Random Forest

Random forest is a Supervised Machine Learning Algorithm that is used widely in Classification and Regression problems. It creates decision trees from several samples, relying on their majority for classification and average for regression. One of the key characteristics of the Random Forest Algorithm is its ability to handle data sets with both continuous variables, as in regression, and categorical variables, as in classifications. For classification issues, it produces superior results. (Analytics Vidhya, 2022)

In this modelling we are going to use random forest classifier as this is a classification problem and random forest classifier is best suited for classification problems.

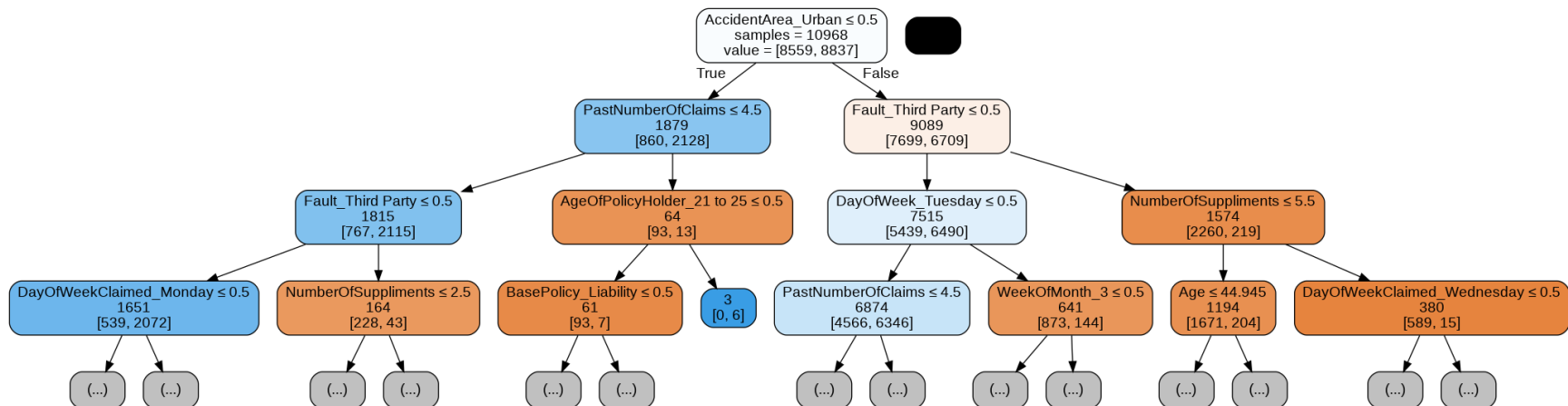
The following are the codes for RandomForestClassifier and plotting Random Forest decision tree with max depth 4

```
✓ [104] from sklearn import tree
1s      from sklearn.ensemble import RandomForestClassifier
```

```
✓ [105] random_forest = RandomForestClassifier(random_state=1, n_estimators=500)
3s      random_forest.fit(X_train, y_train)
```

```
RandomForestClassifier(n_estimators=500, random_state=1)
```

```
✓ [169] randomforest = random_forest.estimators_[3]
1s      plotDecisionTree(randomforest, feature_names = X_train.columns, max_depth=4)
```



The following are the codes for drafting table of variable importance.

```
[106] importance = random_forest.feature_importances_
      std = np.std([tree.feature_importances_ for tree in random_forest.estimators_], axis= 0)

[107] random_forest_df = pd.DataFrame({"feature" : X_train.columns,
                                     "importance" : importance,
                                     "Std" : std})
      print(random_forest_df.sort_values("importance", ascending = False))
```

	feature	importance	Std
106	BasePolicy_Liability	0.092351	0.074845
74	Fault_Third Party	0.075527	0.033614
75	VehicleCategory_Sport	0.064569	0.065793
104	Year_1996	0.029257	0.022171
103	Year_1995	0.028551	0.022459
..
41	Make_Mercedes	0.000026	0.000093
45	Make_Porsche	0.000015	0.000077
38	Make_Jaguar	0.000011	0.000056
35	Make_Ferrari	0.000007	0.000046
39	Make_Lexus	0.000004	0.000033

[108 rows x 3 columns]

It can be observed from the Random Forest tree and feature importance table that the first split of tree is not the most important feature. The reason behind this is the tree select split with the lowest variance but the most important variable in the feature importance has a standard deviation of 0.0748.

The following are the codes for classification summary for random forest predictions.

```
✓ [108] classificationSummary(y_train, random_forest.predict(X_train))
```

Confusion Matrix (Accuracy 1.0000)

	Prediction	
Actual	0	1
0	8590	0
1	0	8806

```
✓ [109] classificationSummary(y_valid, random_forest.predict(X_valid))
```

Confusion Matrix (Accuracy 0.9677)

	Prediction	
Actual	0	1
0	5850	57
1	318	5373

After examining the confusion matrix above, it can be concluded that the model is overfit as the accuracy of valid data is low as compared to its train data.

The following are the codes for plotting ROC curve.

```

y_pred_proba_rf= random_forest.predict_proba(X_valid)[::,1]

y_pred_gini_rf = random_forest.predict(X_valid)

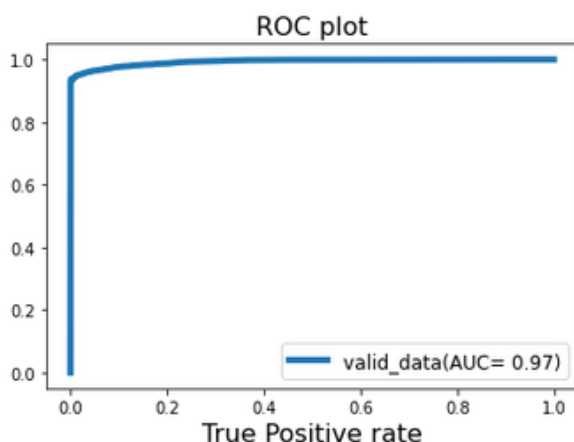
auc_rf = metrics.roc_auc_score(y_valid, y_pred_gini_rf)

fpr5, tpr5, _ = metrics.roc_curve(y_valid, y_pred_proba_rf)

plt.plot(fpr5, tpr5, label= "valid_data(AUC= %0.2f)" % auc_rf, linewidth = 4)

plt.legend(prop={'size' : 12}, loc = "lower right")
plt.title('ROC plot', fontsize = 16)
plt.xlabel('False Positive rate', fontsize = 16)
plt.ylabel('True Positive rate', fontsize = 16)
plt.show()

```



```

print('Gini stats')
print("Accuracy:",metrics.accuracy_score(y_valid, y_pred_gini_rf))
print("balanced_accuracy:",metrics.balanced_accuracy_score(y_valid, y_pred_gini_rf))
print("brier_score_loss:",metrics.brier_score_loss(y_valid, y_pred_gini_rf))
print("f1_score:",metrics.f1_score(y_valid,y_pred_gini_rf))
print("recall_score:",metrics.recall_score(y_valid, y_pred_gini_rf))
print("precision_score:",metrics.precision_score(y_valid, y_pred_gini_rf))
print("roc_auc_score:",metrics.roc_auc_score(y_valid, y_pred_gini_rf))

```

```

Gini stats
Accuracy: 0.9676668391101914
balanced_accuracy: 0.9672363650285274
brier_score_loss: 0.03233316088980859
f1_score: 0.9662800107903965
recall_score: 0.9441222983658408
precision_score: 0.9895027624309393
roc_auc_score: 0.9672363650285273

```


7.0 Neural Networks

Artificial neural networks are forecasting techniques that are based on basic mathematical models of the brain. They enable intricate nonlinear interactions between the response variable and its predictors. (Forecasting, n.d.). A neural network is a potent tool for predictive analysis currently available. The technique is a development of logistic regression. Another way to describe it is as logistic regression on a set of hidden units.

In this Analysis, we are going to use the following activation function and solver:

Activation Function: - ReLU is the activation function to be used as it sets all negative values in the matrix x to zero while maintaining the constant values for all other values. (StackExchange, n.d.)

Solver: - Adam is the solver to be used as it is an adaptive learning rate optimization algorithm that's been designed specifically for training deep neural networks. (tds, n.d.)

7.1 Neural Network with 40 hidden layers and max_iterations = 1000

The following are the codes for constructing neural network with 40 hidden layers.

```
[112] from sklearn.model_selection import train_test_split
      from sklearn.preprocessing import MinMaxScaler
      from sklearn.neural_network import MLPClassifier, MLPRegressor

      clf=MLPClassifier(hidden_layer_sizes=40, activation='relu', solver='adam', random_state=1, max_iter=1000)
      clf.fit(X_train, y_train)

      MLPClassifier(hidden_layer_sizes=40, max_iter=1000, random_state=1)
```

```
[128] classificationSummary(y_train, clf.predict(X_train))

Confusion Matrix (Accuracy 0.9413)

      Prediction
Actual 0 1
0 8028 562
1 460 8346
```

```
[129] classificationSummary(y_valid, clf.predict(X_valid))

Confusion Matrix (Accuracy 0.9402)

      Prediction
Actual 0 1
0 5509 398
1 295 5396
```

After examining the confusion matrix above, it can be concluded that the model is slightly overfit as the accuracy of valid data is low as compared to its train data. But this can be ignored as the model is predicting less false negatives than the other models.

The following are the codes for plotting ROC curve.

```
[139] y_pred_proba_clf= clf.predict_proba(X_valid)[::,1]

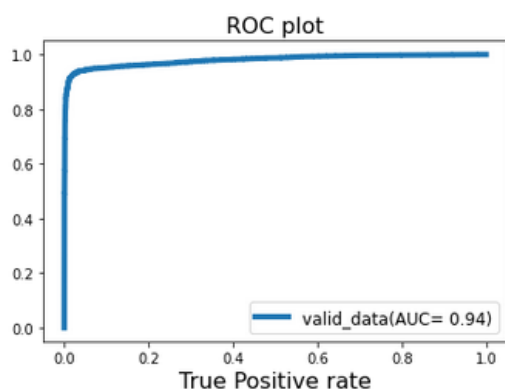
y_pred_gini_clf = clf.predict(X_valid)

auc_clf = metrics.roc_auc_score(y_valid, y_pred_gini_clf)

fpr5, tpr5, _ = metrics.roc_curve(y_valid, y_pred_proba_clf)

plt.plot(fpr5, tpr5, label= "valid_data(AUC= %0.2f)" % auc_clf, linewidth = 4)

plt.legend(prop={'size' : 12}, loc = "lower right")
plt.title('ROC plot', fontsize = 16)
plt.xlabel('False Positive rate', fontsize = 16)
plt.ylabel('True Positive rate', fontsize = 16)
plt.show()
```



```
print('Gini stats')
print("Accuracy:",metrics.accuracy_score(y_valid, y_pred_gini_clf))
print("balanced_accuracy:",metrics.balanced_accuracy_score(y_valid, y_pred_gini_clf))
print("brier_score_loss:",metrics.brier_score_loss(y_valid, y_pred_gini_clf))
print("f1_score:",metrics.f1_score(y_valid,y_pred_gini_clf))
print("recall_score:",metrics.recall_score(y_valid, y_pred_gini_clf))
print("precision_score:",metrics.precision_score(y_valid, y_pred_gini_clf))
print("roc_auc_score:",metrics.roc_auc_score(y_valid, y_pred_gini_clf))
```

```
Gini stats
Accuracy: 0.9402483186756337
balanced_accuracy: 0.94039303993127
brier_score_loss: 0.05975168132436627
f1_score: 0.9396604266434481
recall_score: 0.9481637673519593
precision_score: 0.9313082499137039
roc_auc_score: 0.9403930399312701
```

7.2 Neural Network with 52 hidden layers and max_iterations = 1000

The following are the codes for constructing neural network with 52 hidden layers.

```
✓ [136] clf1 = MLPClassifier(hidden_layer_sizes=52,activation='relu',solver='adam',random_state=1,
6s      max_iter=1000)
      clf1.fit(X_train, y_train)

MLPClassifier(hidden_layer_sizes=52, max_iter=1000, random_state=1)
```

```
✓ [137] classificationSummary(y_train,(clf1.predict(X_train)))
js

Confusion Matrix (Accuracy 0.8009)

      Prediction
Actual   0     1
0  8568    22
1  3442  5364
```

```
✓ [138] classificationSummary(y_valid, clf1.predict(X_valid))
js

Confusion Matrix (Accuracy 0.8064)

      Prediction
Actual   0     1
0  5882    25
1  2220  3471
```

After examining the confusion matrix above, it is observed that the model is predicting very high number of false negatives that is 2220. Selecting this model will give a huge loss to the company.

The following are the codes for plotting ROC curve.

```
[140] y_pred_proba_clf1= clf1.predict_proba(X_valid)[:,:1]

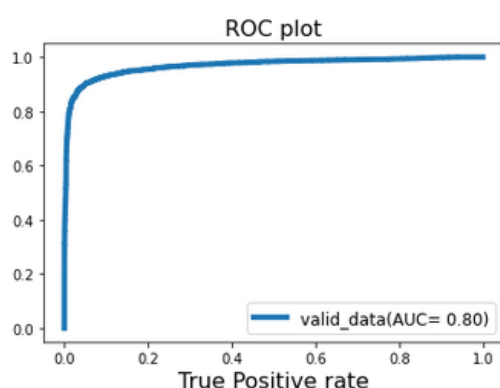
y_pred_gini_clf1 = clf1.predict(X_valid)

auc_clf1 = metrics.roc_auc_score(y_valid, y_pred_gini_clf1)

fpr6, tpr6, _ = metrics.roc_curve(y_valid, y_pred_proba_clf1)

plt.plot(fpr6, tpr6, label= "valid_data(AUC= %0.2f)" % auc_clf1, linewidth = 4)

plt.legend(prop={'size' : 12}, loc = "lower right")
plt.title('ROC plot', fontsize = 16)
plt.xlabel('False Positive rate', fontsize = 16)
plt.ylabel('True Positive rate', fontsize = 16)
plt.show()
```



```
print('Gini stats')
print("Accuracy:",metrics.accuracy_score(y_valid, y_pred_gini_clf1))
print("balanced_accuracy:",metrics.balanced_accuracy_score(y_valid, y_pred_gini_clf1))
print("brier_score_loss:",metrics.brier_score_loss(y_valid, y_pred_gini_clf1))
print("f1_score:",metrics.f1_score(y_valid,y_pred_gini_clf1))
print("recall_score:",metrics.recall_score(y_valid, y_pred_gini_clf1))
print("precision_score:",metrics.precision_score(y_valid, y_pred_gini_clf1))
print("roc_auc_score:",metrics.roc_auc_score(y_valid, y_pred_gini_clf1))
```

```
Gini stats
Accuracy: 0.8064321434730126
balanced_accuracy: 0.8028390590080173
brier_score_loss: 0.1935678565269874
f1_score: 0.7556329596168497
recall_score: 0.6099103848181339
precision_score: 0.9928489702517163
roc_auc_score: 0.8028390590080173
```

7.3 Neural Network with 45 hidden layers and max_iterations = 1000

The following are the codes for constructing neural network with 45 hidden layers.

```
✓ [176] clf2 = MLPClassifier(hidden_layer_sizes=45,activation='relu',solver='adam',random_state=1,
3s                               max_iter=1000)
      clf2.fit(X_train, y_train)

      MLPClassifier(hidden_layer_sizes=45, max_iter=1000, random_state=1)
```

```
✓ [177] classificationSummary(y_train,(clf2.predict(X_train)))
3s

      Confusion Matrix (Accuracy 0.9453)

                Prediction
      Actual    0      1
      0  8107  483
      1   469  8337
```

```
✓ [178] classificationSummary(y_valid,(clf2.predict(X_valid)))
3s

      Confusion Matrix (Accuracy 0.9430)

                Prediction
      Actual    0      1
      0  5544  363
      1   298  5393
```

After examining the confusion matrix above, it can be concluded that the model is slightly overfit as the accuracy of valid data is low as compared to its train data. But this can be ignored as the model is predicting less false negatives than the other models.

The following are the codes for plotting ROC curve.

```

y_pred_proba_clf2= clf2.predict_proba(X_valid)[::,1]

y_pred_gini_clf2 = clf2.predict(X_valid)

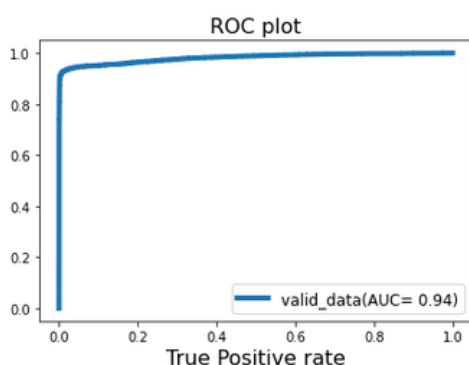
auc_clf2 = metrics.roc_auc_score(y_valid, y_pred_gini_clf2)

fpr7, tpr7, _ = metrics.roc_curve(y_valid, y_pred_proba_clf2)

plt.plot(fpr7, tpr7, label= "valid_data(AUC= %.2f)" % auc_clf2, linewidth = 4)

plt.legend(prop={'size' : 12}, loc = "lower right")
plt.title('ROC plot', fontsize = 16)
plt.xlabel('False Positive rate', fontsize = 16)
plt.ylabel('True Positive rate', fontsize = 16)
plt.show()

```



```

print('Gini stats')
print("Accuracy:",metrics.accuracy_score(y_valid, y_pred_gini_clf2))
print("balanced_accuracy:",metrics.balanced_accuracy_score(y_valid, y_pred_gini_clf2))
print("brier_score_loss:",metrics.brier_score_loss(y_valid, y_pred_gini_clf2))
print("f1_score:",metrics.f1_score(y_valid,y_pred_gini_clf2))
print("recall_score:",metrics.recall_score(y_valid, y_pred_gini_clf2))
print("precision_score:",metrics.precision_score(y_valid, y_pred_gini_clf2))
print("roc_auc_score:",metrics.roc_auc_score(y_valid, y_pred_gini_clf2))

```

```

Gini stats
Accuracy: 0.943007415071564
balanced_accuracy: 0.9430920526284273
brier_score_loss: 0.05699258492843594
f1_score: 0.9422556128243207
recall_score: 0.9476366192233351
precision_score: 0.9369353717859624
roc_auc_score: 0.9430920526284272

```

7.4 Neural Network Model Comparison.

After carefully examining the confusion matrix of all three Neural Network models, it can be concluded that the Neural Network with 40 hidden layers and maximum iterations 1000 will be

the best among the three as it is predicting very few false negatives that is 295 and the model will be best suited for predicting fraud, among the three neural networks.

8.0 Model Comparison and Selection






To determine which of our models has the best performance, we are going to compare the gini stats, accuracies, and root mean squared error (RMSE) of the models. We build Decision Trees, Regression Models, Logistic Regression, GridSearchCV Regression Tree, Random Forest, Neural Networks. The following table will help in better comparison of the models.

8.1 Classification Models Table

Model Name	Training Accuracy	Validation Accuracy	F1 Score
Full Decision Tree	1.0	0.9283	0.9283
Decision Tree (Max depth = 5)	0.8407	0.8321	0.8463
Logistic Regression	0.9553	0.9559	0.9544
GridSearchCV Decision Tree	0.8957	0.8883	0.8861
Random Forest	1.0	0.9677	0.9662
Neural Network (40 Hidden layers)	0.9413	0.9402	0.9396
Neural Network (52 Hidden layers)	0.8009	0.8064	0.7556
Neural Network (45 Hidden layers)	0.9453	0.9430	0.9422

8.2 Regression Models Table

Model Name	Training RMSE	Validation RMSE
Linear Regression	0.2116	0.2114
Forward Regression	0.2116	0.2114
Backward Regression	0.2116	0.2114

	Overfitted models		Overfitted models predicting less false negatives
	The best model		Worst model
	Models that are good but not best fit		

8.3 Model Selection.

Among all the models, Full Decision tree, GridSearchCV, Decision Tree (max depth 5), Random Forest, Neural Network (40 Hidden layers), Neural Network (45 Hidden layers) are overfitted models as training accuracy is more than the validation accuracy. Overfitting is a situation when a model performs well on training data but poorly on validation data or unseen data. This indicates that the model is remembering the training data instead of learning the relationship between features and labels. However, the Random Forest, Neural Networks with 40 and 45 hidden layers are predicting a smaller number of false negatives (Not predicting frauds) that is 318, 295, and 295, respectively. These models may be a good fit for analysis but not a best fit. Moving further, Neural Network (52 Hidden layers) is the worst model as it has the highest number of false negatives that is 2220, which will escalate the cost caused by frauds to the company. Furthermore, linear regression, forward regression, and backward regression model provides the same results, but these are not considered to be the best model because these are used to handle regression problems and provides continuous output whereas classification problems mandate discrete values. In the classification model table, we can see that Random

Forest model has the highest F1 score but to eliminate the risk and costs associated with overfitting, Logistic Regression model will be considered as the best model for the analysis.

9.0 Model Validation Plan

For model validation, every month, ROC or AUC, False Negatives and True Positive value analysis will be performed on the classification model (See section 4.0 for codes). If we observe a 3-4% drift from the baseline value (See accuracy rate in section 4.0) in the model performance, immediate action will be taken to retain the model score. The following approaches can be followed:

1. The model will be re-tuned using the same variables. We can use the Principal Component Analysis (PCA) method to find the components that can best describe the data.
2. To regain the standards, the model might be updated with new pertinent dimensions. Additional relevant dimensions can be some other features/traits introduced with the time that can play an important role in decision making. A consistent methodology should be adopted.
3. Regularization techniques should be performed in order to prevent overfitting of the model in the future.

10.0 Logistic Regression Outcomes

Coefficients: This is the result of the logistic regression equation, which uses the independent variable to forecast the dependent variable. The units for these values are log-odds. In essence, these estimates show how the dependent variable, which is on the logit scale, is related to the independent variable. The following are the top 10 coefficients in descending order, see section 4.0 for codes (UCLA, n.d.).

```
[136] Best_coef.head(10)
```

	coef
Days_Policy_Claim	0.160240
AgeOfPolicyHolder_21 to 25	0.111443
NumberOfCars	0.058998
Sex_Male	0.009229
Grouped_Make_Non-Luxury	0.007126
Deductible	0.004094
AddressChange_Claim	0.001833
AgeOfPolicyHolder_18 to 20	0.001234
Make_Mercedes	0.000683
VehiclePrice	0.000004

Odds Ratio: A measure of the relationship between an exposure and a result is an odds ratio. In comparison to the chances of the event occurring without that exposure, the odds ratio shows the likelihood that an outcome will occur given a certain exposure. Case-control studies employ odds ratios the most frequently, although cross-sectional and cohort study designs can also make use of them (NIH, 2015).

Odds Ratio and Logistic Regression: When a logistic regression is calculated, the regression coefficient (b_1) is the estimated increase in the log odds of the outcome per unit increase in the value of the exposure. In other words, the exponential function of the regression coefficient (e^{b_1}) is the odds ratio associated with a one-unit increase in the exposure. (NIH, 2015)

The following are the odds ratio estimates of those variables that will increase the chance of predicting fraud with an increase in single unit of each variable in descending order (See section 4.0 for codes).

Calculating the odds ratio

```

import math
print("Days_Policy_Claim_odds:", math.exp(0.160240))
print("AgeOfPolicyHolder_21to25_odds:", math.exp(0.111443))
print("NumberOfCars_odds:", math.exp(0.058998))
print("Sex_Male_odds:", math.exp(0.009229))
print("Grouped_Make_NonLuxury_odds:", math.exp(0.007126))
print("Deductible_odds:", math.exp(0.004094))
print("AddressChange_Claim_odds:", math.exp(0.001833))
print("AgeOfPolicyHolder_18to20_odds:", math.exp(0.001234))
print("Make_Mercedes_odds:", math.exp(0.000683))
print("VehiclePrice_odds:", math.exp(0.000004))

Days_Policy_Claim_odds: 1.1737925474006652
AgeOfPolicyHolder_21to25_odds: 1.1178900224582038
NumberOfCars_odds: 1.060773119191799
Sex_Male_odds: 1.0092717185358233
Grouped_Make_NonLuxury_odds: 1.0071514503551608
Deductible_odds: 1.004102391866192
AddressChange_Claim_odds: 1.0018346809714167
AgeOfPolicyHolder_18to20_odds: 1.001234761691277
Make_Mercedes_odds: 1.000683233297611
VehiclePrice_odds: 1.000004000008

```

According to the above odds ratio: -

Days_Policy_Claim: If the “Days_Policy_Claim” variable is increased by a single unit, the chance of a claim being a fraud 1.1737925474006652 times higher than usual.

AgeOfPolicyHolder_21 to 25: If the “AgeOfPolicyHolder_21 to 25” variable is increased by a single unit, the chance of a claim being a fraud 1.1178900224582038 times higher than usual.

NumberOfCars: If the “NumberOfCars” variable is increased by a single unit, the chance of a claim being a fraud 1.060773119191799 times higher than usual.

Sex_Male: If the “Sex_Male” variable is increased by a single unit, the chance of a claim being a fraud 1.0092717185358233 times higher than usual.

Grouped_Make_Non-Luxury: If the “Grouped_Make_Non-Luxury” variable is increased by a single unit, the chance of a claim being a fraud 1.0071514503551608 times higher than usual.

Deductible: If the “Deductible” variable is increased by a single unit, the chance of a claim being a fraud 1.004102391866192 times higher than usual.

AddressChange_Claim: If the “AddressChange_Claim” variable is increased by a single unit, the chance of a claim being a fraud 1.0018346809714167 times higher than usual.

AgeOfPolicyHolder_18 to 20: If the “AgeOfPolicyHolder_18 to 20” variable is increased by a single unit, the chance of a claim being a fraud 1.001234761691277 times higher than usual.

Make_Mercedes: If the “Make_Mercedes” variable is increased by a single unit, the chance of a claim being a fraud 1.000683233297611 times higher than usual.

VehiclePrice: If the “VehiclePrice” variable is increased by a single unit, the chance of a claim being a fraud 1.000004000008 times higher than usual.

11.0 Recommendation

From the odds ratio outcomes of the logistic regression, we can predict that the chances of claim being fraud is highly dependent on the following variables: -

- Days_Policy_Claim
- AgeOfPolicyHolder_21 to 25
- NumberOfCars
- Sex_Male_odds
- Grouped_Make_Non-Luxury

Hence, if the person is found being fraud on the basis of these characteristics. further investigation is recommended, and strict actions should be taken, if found guilty. Moreover, fraud control strategies should be formulated and implemented in order to prevent future fraudulent vehicle insurance claims.

12.0 Future Work

Regarding the future work, regularization should be used on quarterly basis in order to prevent overfitting or underfitting of the model in the future. The term "regularization" describes methods for calibrating machine learning models to reduce the adjusted loss function and avoid overfitting or underfitting. Using regularization, we can fit our model accurately with minimal errors.

The Dataset contains a variable “SEX” which classifies the data on the basis of gender, this variable will need some adjustments in the future. As the data is of the period when there were fewer female drivers and to fit this data accurately in the given future the data distribution of the “SEX” variable must be adjusted.

References

Analytics Vidhya. (2022). Retrieved from Understanding Random Forest:

<https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/>

analyticsvidhya. (2022). Retrieved from analyticsvidhya:

<https://www.analyticsvidhya.com/blog/2022/04/exploratory-data-analysis-eda-in-python/>

Forecasting. (n.d.). Retrieved from Neural Networks: <https://otexts.com/fpp2/nnetar.html>

Great Learning. (2020). Retrieved from Great Learning:

<https://www.mygreatlearning.com/blog/gridsearchcv/>

IBM. (2021). Retrieved from IBM: <https://www.ibm.com/cloud/learn/data-visualization>

Kaggle. (2021). Retrieved from Kaggle: <https://www.kaggle.com/datasets/shivamb/vehicle-claim-fraud-detection>

KDnuggets. (2022). Retrieved from KDnuggets: <https://www.kdnuggets.com/2020/01/decision-tree-algorithm-explained.html>

Machine Learning Mastery. (2020). Retrieved from Machine Learning Mastery:

<https://machinelearningmastery.com/linear-regression-for-machine-learning/>

MLM. (2020). Retrieved from MLM: <https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/>

NIH. (2015). Retrieved from NIH: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC2938757/>

StackExchange. (n.d.). Retrieved from StackExchange:

<https://stats.stackexchange.com/questions/226923/why-do-we-use-relu-in-neural-networks-and-how-do-we-use-it>

Statistics How To. (n.d.). Retrieved from Statistics How To:

<https://www.statisticshowto.com/forward-selection/>

tds. (n.d.). Retrieved from tds: <https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c>

tds. (2017). Retrieved from tds: <https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052>

TechTarget. (n.d.). Retrieved from TechTarget:

<https://www.techtarget.com/searchdatamanagement/definition/feature-engineering>

UCLA. (n.d.). Retrieved from UCLA: <https://stats.oarc.ucla.edu/stata/output/logistic-regression-analysis/>

Wikipedia. (n.d.). Retrieved from Wikipedia:

[https://en.wikipedia.org/wiki/Dummy_variable_\(statistics\)](https://en.wikipedia.org/wiki/Dummy_variable_(statistics))

Wikipedia. (n.d.). Retrieved from Wikipedia: https://en.wikipedia.org/wiki/History_of_Python