In [54]:

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

# Scaling
from sklearn.preprocessing import MinMaxScaler, StandardScaler

from sklearn.model_selection import train_test_split,GridSearchCV, RandomizedSearchC
from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score,cor
from sklearn.metrics import classification_report,plot_confusion_matrix,roc_curve,au
from statsmodels.stats.outliers_influence import variance_inflation_factor

import pickle
import json
import warnings
warnings.filterwarnings("ignore")
```

Problem Statement :

In [ ]:

```

```

Data Gathering

In [55]:

```
df = pd.read_csv("fraud_claims_data.csv")
df
```

Out[55]:

DayOfWeekClaimed	MonthClaimed	WeekOfMonthClaimed	Sex	MaritalStatus	...	AgeOfVehi
Tuesday	Jan	1	Female	Single	...	3 ye
Monday	Jan	4	Male	Single	...	6 ye
Thursday	Nov	2	Male	Married	...	7 ye
Friday	Jul	1	Male	Married	...	more tha
Tuesday	Feb	2	Female	Single	...	5 ye
...	...	...	...	...	...	
Tuesday	Nov	5	Male	Married	...	6 ye
Friday	Dec	1	Male	Married	...	6 ye
Friday	Dec	1	Male	Single	...	5 ye
Thursday	Dec	2	Female	Married	...	2 ye
Thursday	Dec	3	Male	Single	...	5 ye

Exploratory Data Analysis

In [56]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15420 entries, 0 to 15419
Data columns (total 33 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Month                 15420 non-null  object
 1   WeekOfMonth           15420 non-null  int64 
 2   DayOfWeek             15420 non-null  object
 3   Make                  15420 non-null  object
 4   AccidentArea          15420 non-null  object
 5   DayOfWeekClaimed      15420 non-null  object
 6   MonthClaimed          15420 non-null  object
 7   WeekOfMonthClaimed    15420 non-null  int64 
 8   Sex                   15420 non-null  object
 9   MaritalStatus         15420 non-null  object
 10  Age                   15420 non-null  int64 
 11  Fault                 15420 non-null  object
 12  PolicyType            15420 non-null  object
 13  VehicleCategory       15420 non-null  object
 14  VehiclePrice          15420 non-null  object
 15  FraudFound_P          15420 non-null  int64 
 16  PolicyNumber          15420 non-null  int64 
 17  RepNumber             15420 non-null  int64 
 18  Deductible            15420 non-null  int64 
 19  DriverRating          15420 non-null  int64 
 20  Days_Policy_Accident  15420 non-null  object
 21  Days_Policy_Claim     15420 non-null  object
 22  PastNumberOfClaims    15420 non-null  object
 23  AgeOfVehicle          15420 non-null  object
 24  AgeOfPolicyHolder     15420 non-null  object
 25  PoliceReportFiled     15420 non-null  object
 26  WitnessPresent        15420 non-null  object
 27  AgentType             15420 non-null  object
 28  NumberOfSuppliments   15420 non-null  object
 29  AddressChange_Claim   15420 non-null  object
 30  NumberOfCars          15420 non-null  object
 31  Year                  15420 non-null  int64 
 32  BasePolicy            15420 non-null  object
dtypes: int64(9), object(24)
memory usage: 3.9+ MB
```

In [57]:

```python
df.isna().sum()           # Checking for missing value
```

Out[57]:

```
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
NumberOfSuppliments     0
AddressChange_Claim     0
NumberOfCars            0
Year                    0
BasePolicy              0
dtype: int64
```

In [58]:

```
df.describe()
```

Out[58]:

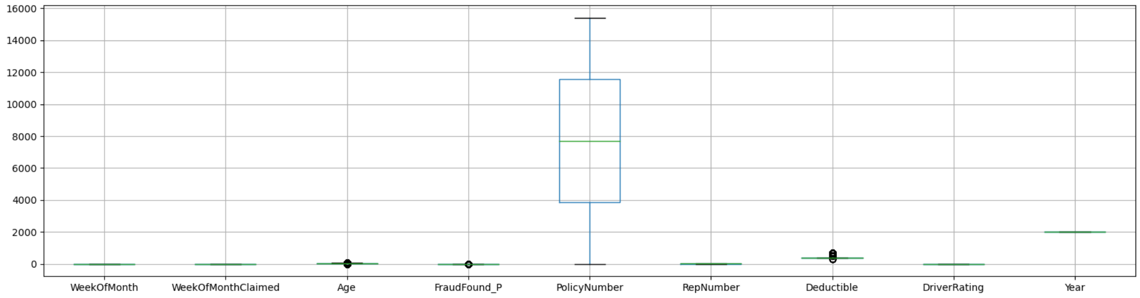
	WeekOfMonth	WeekOfMonthClaimed	Age	FraudFound_P	PolicyNumber	R
count	15420.000000	15420.000000	15420.000000	15420.000000	15420.000000	154
mean	2.788586	2.693969	39.855707	0.059857	7710.500000	
std	1.287585	1.259115	13.492377	0.237230	4451.514911	
min	1.000000	1.000000	0.000000	0.000000	1.000000	
25%	2.000000	2.000000	31.000000	0.000000	3855.750000	
50%	3.000000	3.000000	38.000000	0.000000	7710.500000	
75%	4.000000	4.000000	48.000000	0.000000	11565.250000	
max	5.000000	5.000000	80.000000	1.000000	15420.000000	

In [59]:

```
# Checking for outliers
plt.figure(figsize = (20,5))
df.boxplot()
```

Out[59]:

```
<AxesSubplot:>
```



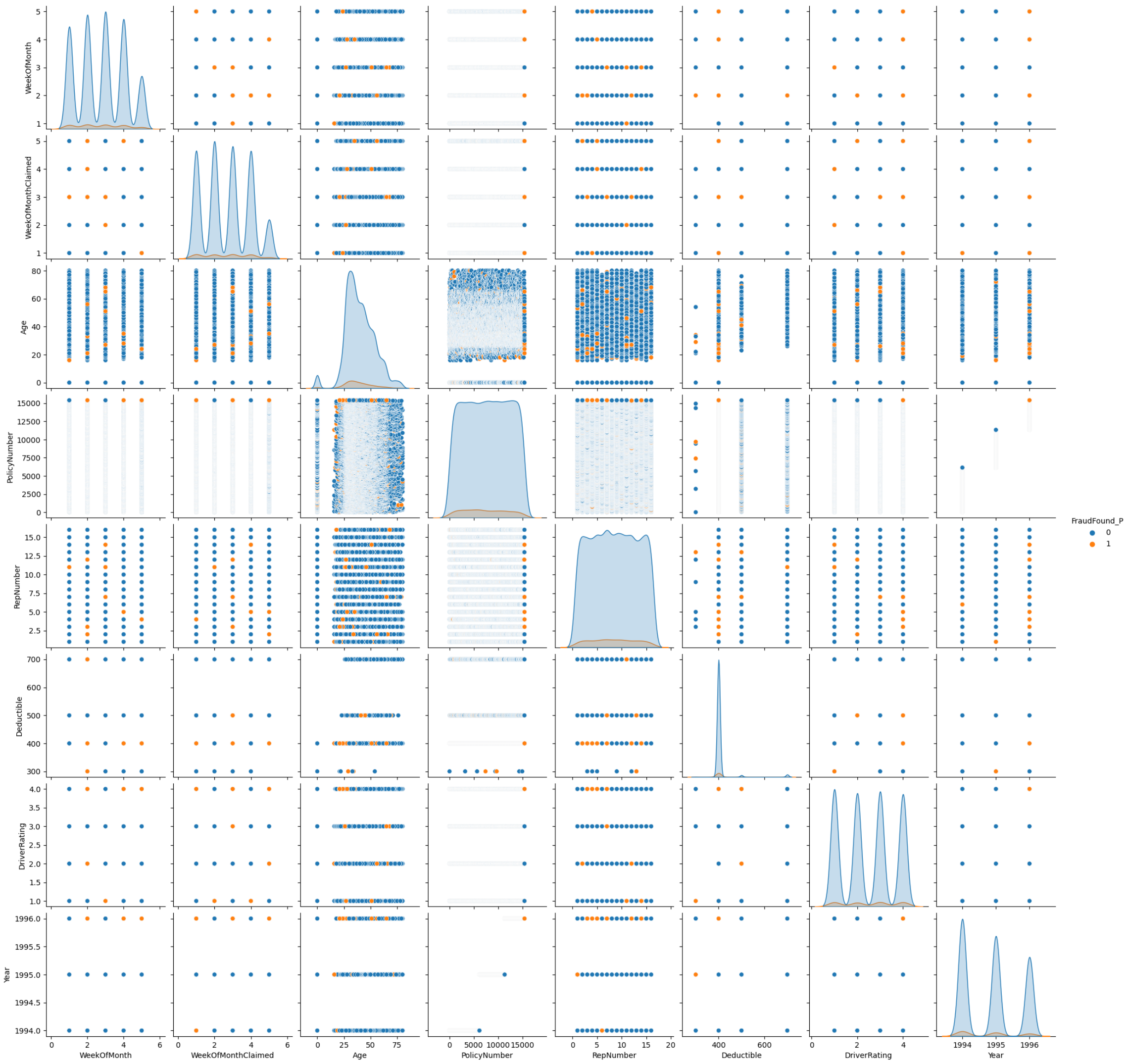
Feature Engineering

In [60]:

```
sns.pairplot(df, hue = 'FraudFound_P')
```

Out[60]:

```
<seaborn.axisgrid.PairGrid at 0x16e5f7469d0>
```



In [61]:

```
### Replacing object data type into Numeric
```

In [62]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15420 entries, 0 to 15419
Data columns (total 33 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   Month                 15420 non-null  object
 1   WeekOfMonth           15420 non-null  int64 
 2   DayOfWeek             15420 non-null  object
 3   Make                  15420 non-null  object
 4   AccidentArea          15420 non-null  object
 5   DayOfWeekClaimed      15420 non-null  object
 6   MonthClaimed          15420 non-null  object
 7   WeekOfMonthClaimed    15420 non-null  int64 
 8   Sex                   15420 non-null  object
 9   MaritalStatus         15420 non-null  object
 10  Age                   15420 non-null  int64 
 11  Fault                 15420 non-null  object
 12  PolicyType            15420 non-null  object
 13  VehicleCategory       15420 non-null  object
 14  VehiclePrice          15420 non-null  object
 15  FraudFound_P          15420 non-null  int64 
 16  PolicyNumber          15420 non-null  int64 
 17  RepNumber             15420 non-null  int64 
 18  Deductible            15420 non-null  int64 
 19  DriverRating          15420 non-null  int64 
 20  Days_Policy_Accident  15420 non-null  object
 21  Days_Policy_Claim     15420 non-null  object
 22  PastNumberOfClaims    15420 non-null  object
 23  AgeOfVehicle          15420 non-null  object
 24  AgeOfPolicyHolder     15420 non-null  object
 25  PoliceReportFiled     15420 non-null  object
 26  WitnessPresent        15420 non-null  object
 27  AgentType             15420 non-null  object
 28  NumberOfSuppliments   15420 non-null  object
 29  AddressChange_Claim   15420 non-null  object
 30  NumberOfCars          15420 non-null  object
 31  Year                  15420 non-null  int64 
 32  BasePolicy            15420 non-null  object
dtypes: int64(9), object(24)
memory usage: 3.9+ MB
```

Month

In [63]:

```
df['Month'].value_counts()
```

Out[63]:

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

In [64]:

```
df = pd.get_dummies(df,columns=['Month'])
```

DayOfWeek

In [65]:

```
df['DayOfWeek'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 15420 entries, 0 to 15419
Series name: DayOfWeek
Non-Null Count  Dtype 
--------------  ----- 
15420 non-null  object
dtypes: object(1)
memory usage: 120.6+ KB
```

In [66]:

```
df['DayOfWeek'].value_counts()
```

Out[66]:

```
Monday       2616
Friday       2445
Tuesday      2300
Thursday     2173
Wednesday    2159
Saturday     1982
Sunday       1745
Name: DayOfWeek, dtype: int64
```

In [67]:

```
df = pd.get_dummies(df,columns=['DayOfWeek'])
```

Make

In [68]:

```
df['Make'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 15420 entries, 0 to 15419
Series name: Make
Non-Null Count  Dtype 
--------------  ----- 
15420 non-null  object
dtypes: object(1)
memory usage: 120.6+ KB
```

In [69]:

```
df['Make'].value_counts()
```

Out[69]:

```
Pontiac      3837
Toyota       3121
Honda        2801
Mazda        2354
Chevrolet    1681
Accura        472
Ford          450
VW            283
Dodge         109
Saab          108
Mercury        83
Saturn         58
Nisson         30
BMW            15
Jaguar          6
Porche          5
Mecedes         4
Ferrari         2
Lexus           1
Name: Make, dtype: int64
```

In [70]:

```
df = pd.get_dummies(df,columns=['Make'])
```

In [71]:

```
# df.info()
```

AccidentArea

In [72]:

```
df['AccidentArea'].value_counts()
```

Out[72]:

```
Urban    13822
Rural     1598
Name: AccidentArea, dtype: int64
```

In [73]:

```
df['AccidentArea'].value_counts().to_dict()
```

Out[73]:

```
{'Urban': 13822, 'Rural': 1598}
```

In [74]:

```
df['AccidentArea'].replace({'Urban': 1, 'Rural': 0}, inplace= True)
```

In [75]:

```
df['AccidentArea'].value_counts()
```

Out[75]:

```
1    13822
0     1598
Name: AccidentArea, dtype: int64
```

In [76]:

```
AccidentArea_value = {'Urban': 1, 'Rural': 0}
```

DayOfWeekClaimed

In [77]:

```
df['DayOfWeekClaimed'].value_counts()
```

Out[77]:

```
Monday       3757
Tuesday      3375
Wednesday    2951
Thursday     2660
Friday       2497
Saturday      127
Sunday         52
0               1
Name: DayOfWeekClaimed, dtype: int64
```

In [78]:

```
df['DayOfWeekClaimed'].replace({'0':'Monday'}, inplace = True)
```

In [79]:

```
df['DayOfWeekClaimed'].value_counts()
```

Out[79]:

```
Monday       3758
Tuesday      3375
Wednesday    2951
Thursday     2660
Friday       2497
Saturday      127
Sunday         52
Name: DayOfWeekClaimed, dtype: int64
```

In [80]:

```
df = pd.get_dummies(df,columns=['DayOfWeekClaimed'])
```

MonthClaimed

In [81]:

```
df['MonthClaimed'].value_counts()
```

Out[81]:

```
Jan    1446
May    1411
Mar    1348
Oct    1339
Jun    1293
Feb    1287
Nov    1285
Apr    1271
Sep    1242
Jul    1225
Dec    1146
Aug    1126
0         1
Name: MonthClaimed, dtype: int64
```

In [82]:

```
df = pd.get_dummies(df,columns=['MonthClaimed'])
```

Sex

In [83]:

```
df['Sex'].value_counts()
```

Out[83]:

```
Male      13000
Female     2420
Name: Sex, dtype: int64
```

In [84]:

```
df['Sex'].value_counts().to_dict()
```

Out[84]:

```
{'Male': 13000, 'Female': 2420}
```

In [85]:

```
df['Sex'].replace({'Male': 1, 'Female': 0}, inplace= True)
```

In [86]:

```
df['Sex'].value_counts()
```

Out[86]:

```
1    13000
0     2420
Name: Sex, dtype: int64
```

In [87]:

```
Sex_value = {'Male': 1, 'Female': 0}
```

MaritalStatus

In [88]:

```
df['MaritalStatus'].value_counts()
```

Out[88]:

```
Married     10625
Single       4684
Divorced       76
Widow          35
Name: MaritalStatus, dtype: int64
```

In [89]:

```
df['MaritalStatus'].value_counts().to_dict()
```

Out[89]:

```
{'Married': 10625, 'Single': 4684, 'Divorced': 76, 'Widow': 35}
```

In [90]:

```
df['MaritalStatus'].replace({'Married': 0, 'Single': 1, 'Divorced': 3, 'Widow': 4},
```

In [91]:

```
df['MaritalStatus'].value_counts()
```

Out[91]:

```
0    10625
1     4684
3       76
4       35
Name: MaritalStatus, dtype: int64
```

In [92]:

```
MaritalStatus_value = {'Married': 0, 'Single': 1, 'Divorced': 3, 'Widow': 4}
```

Fault

In [93]:

```
df['Fault'].value_counts()
```

Out[93]:

```
Policy Holder    11230
Third Party       4190
Name: Fault, dtype: int64
```

In [94]:

```
df['Fault'].value_counts().to_dict()
```

Out[94]:

```
{'Policy Holder': 11230, 'Third Party': 4190}
```

In [95]:

```
df['Fault'].replace({'Policy Holder': 1, 'Third Party': 0}, inplace= True)
```

In [96]:

```
df['Fault'].value_counts()
```

Out[96]:

```
1    11230
0     4190
Name: Fault, dtype: int64
```

In [97]:

```
Fault_value = {'Policy Holder': 1, 'Third Party': 0}
```

PolicyType

In [98]:

```
df['PolicyType'].value_counts()
```

Out[98]:

```
Sedan - Collision       5584
Sedan - Liability       4987
Sedan - All Perils      4087
Sport - Collision        348
Utility - All Perils     340
Utility - Collision       30
Sport - All Perils        22
Utility - Liability       21
Sport - Liability          1
Name: PolicyType, dtype: int64
```

In [99]:

```
df['PolicyType'].value_counts().to_dict()
```

Out[99]:

```
{'Sedan - Collision': 5584,
 'Sedan - Liability': 4987,
 'Sedan - All Perils': 4087,
 'Sport - Collision': 348,
 'Utility - All Perils': 340,
 'Utility - Collision': 30,
 'Sport - All Perils': 22,
 'Utility - Liability': 21,
 'Sport - Liability': 1}
```

In [100]:

```
df['PolicyType'].replace({'Sedan - Collision': 'Sedan_Collision',
 'Sedan - Liability': 'Sedan_Liability',
 'Sedan - All Perils': 'Sedan_All_Perils',
 'Sport - Collision': 'Sport_Collision',
 'Utility - All Perils': 'Utility_All_Perils',
 'Utility - Collision': 'Utility_Collision',
 'Sport - All Perils': 'Sport_All_Perils',
 'Utility - Liability': 'Utility_Liability',
 'Sport - Liability': 'Sport_Liability'}, inplace= True)
```

In [101]:

```
df['PolicyType'].value_counts()
```

Out[101]:

```
Sedan_Collision       5584
Sedan_Liability       4987
Sedan_All_Perils      4087
Sport_Collision        348
Utility_All_Perils     340
Utility_Collision       30
Sport_All_Perils        22
Utility_Liability       21
Sport_Liability          1
Name: PolicyType, dtype: int64
```

In [102]:

```
df = pd.get_dummies(df, columns=['PolicyType'],prefix='PolicyType',)
```

VehicleCategory

In [103]:

```
df['VehicleCategory'].value_counts()
```

Out[103]:

```
Sedan      9671
Sport      5358
Utility     391
Name: VehicleCategory, dtype: int64
```

In [104]:

```
df['VehicleCategory'].value_counts().to_dict()
```

Out[104]:

```
{'Sedan': 9671, 'Sport': 5358, 'Utility': 391}
```

In [105]:

```
df['VehicleCategory'].replace({'Sedan': 0, 'Sport': 1, 'Utility': 2}, inplace= True)
```

In [106]:

```
df['VehicleCategory'].value_counts()
```

Out[106]:

```
0    9671
1    5358
2     391
Name: VehicleCategory, dtype: int64
```

In [107]:

```
VehicleCategory_value = {'Sedan': 0, 'Sport': 1, 'Utility': 2}
```

Days_Policy_Accident

In [108]:

```
df['Days_Policy_Accident'].value_counts()
```

Out[108]:

```
more than 30    15247
none               55
8 to 15            55
15 to 30           49
1 to 7             14
Name: Days_Policy_Accident, dtype: int64
```

Days_Policy_Claim

In [109]:

```
df['Days_Policy_Claim'].value_counts()
```

Out[109]:

```
more than 30    15342
15 to 30           56
8 to 15            21
none                1
Name: Days_Policy_Claim, dtype: int64
```

In [110]:

```
df['Days_Policy_Claim'].value_counts().to_dict()
```

Out[110]:

```
{'more than 30': 15342, '15 to 30': 56, '8 to 15': 21, 'none': 1}
```

In [111]:

```
df['Days_Policy_Claim'].replace({'more than 30': 30, '15 to 30': 15, '8 to 15': 8, '
```

In [112]:

```
df['Days_Policy_Claim'].value_counts()
```

Out[112]:

```
30    15342
15       56
8        21
1         1
Name: Days_Policy_Claim, dtype: int64
```

In [113]:

```
Days_Policy_Claim_value = {'more than 30': 30, '15 to 30': 15, '8 to 15': 8, 'none':
```

PoliceReportFiled

In [114]:

```
df['PoliceReportFiled'].value_counts()
```

Out[114]:

```
No     14992
Yes      428
Name: PoliceReportFiled, dtype: int64
```

In [115]:

```
df['PoliceReportFiled'].value_counts().to_dict()
```

Out[115]:

```
{'No': 14992, 'Yes': 428}
```

In [116]:

```
df['PoliceReportFiled'].replace({'No': 0, 'Yes': 1}, inplace= True)
```

In [117]:

```
df['PoliceReportFiled'].value_counts()
```

Out[117]:

```
0    14992
1      428
Name: PoliceReportFiled, dtype: int64
```

In [118]:

```
PoliceReportFiled_value = {'No': 0, 'Yes': 1}
```

WitnessPresent

In [119]:

```
df['WitnessPresent'].value_counts()
```

Out[119]:

```
No     15333
Yes       87
Name: WitnessPresent, dtype: int64
```

In [120]:

```
df['WitnessPresent'].value_counts().to_dict()
```

Out[120]:

```
{'No': 15333, 'Yes': 87}
```

In [121]:

```
df['WitnessPresent'].replace({'No': 0, 'Yes': 1}, inplace= True)
```

In [122]:

```
df['WitnessPresent'].value_counts()
```

Out[122]:

```
0    15333
1       87
Name: WitnessPresent, dtype: int64
```

In [123]:

```
WitnessPresent_value = {'No': 0, 'Yes': 1}
```

AgentType

In [124]:

```
df['AgentType'].value_counts()
```

Out[124]:

```
External    15179
Internal      241
Name: AgentType, dtype: int64
```

In [125]:

```
df['AgentType'].value_counts().to_dict()
```

Out[125]:

```
{'External': 15179, 'Internal': 241}
```

In [126]:

```
df['AgentType'].replace({'External': 0, 'Internal': 1}, inplace= True)
```

In [127]:

```
df['AgentType'].value_counts()
```

Out[127]:

```
0    15179
1      241
Name: AgentType, dtype: int64
```

In [128]:

```
AgentType_value = {'External': 0, 'Internal': 1}
```

NumberOfSuppliments

In [129]:

```
df['NumberOfSuppliments'].value_counts()
```

Out[129]:

```
none           7047
more than 5    3867
1 to 2         2489
3 to 5         2017
Name: NumberOfSuppliments, dtype: int64
```

In [130]:

```
df['NumberOfSuppliments'].value_counts().to_dict()
```

Out[130]:

```
{'none': 7047, 'more than 5': 3867, '1 to 2': 2489, '3 to 5': 2017}
```

In [131]:

```
df['NumberOfSuppliments'].replace({'none': 0, 'more than 5': 5, '1 to 2': 1, '3 to 5
```

In [132]:

```
df['NumberOfSuppliments'].value_counts()
```

Out[132]:

```
0    7047
5    3867
1    2489
3    2017
Name: NumberOfSuppliments, dtype: int64
```

In [133]:

```
NumberOfSuppliments_value = {'none': 0, 'more than 5': 5, '1 to 2': 1, '3 to 5': 3}
```

AddressChange_Claim

In [134]:

```
df['AddressChange_Claim'].value_counts()
```

Out[134]:

```
no change         14324
4 to 8 years        631
2 to 3 years        291
1 year              170
under 6 months        4
Name: AddressChange_Claim, dtype: int64
```

In [135]:

```
df['AddressChange_Claim'].value_counts().to_dict()
```

Out[135]:

```
{'no change': 14324,
 '4 to 8 years': 631,
 '2 to 3 years': 291,
 '1 year': 170,
 'under 6 months': 4}
```

In [136]:

```
df['AddressChange_Claim'].replace({'no change': 0,'4 to 8 years': 4,'2 to 3 years':
```

In [137]:

```
df['AddressChange_Claim'].value_counts()
```

Out[137]:

```
0    14324
4      631
2      291
1      170
6        4
Name: AddressChange_Claim, dtype: int64
```

In [138]:

```
AddressChange_Claim_value = {'no change': 0,'4 to 8 years': 4,'2 to 3 years': 2,'1 y
```

NumberOfCars

In [139]:

```
df['NumberOfCars'].value_counts()
```

Out[139]:

```
1 vehicle      14316
2 vehicles       709
3 to 4           372
5 to 8            21
more than 8        2
Name: NumberOfCars, dtype: int64
```

In [140]:

```
df['NumberOfCars'].value_counts().to_dict()
```

Out[140]:

```
{'1 vehicle': 14316,
 '2 vehicles': 709,
 '3 to 4': 372,
 '5 to 8': 21,
 'more than 8': 2}
```

In [141]:

```
df['NumberOfCars'].replace({'1 vehicle': 1,'2 vehicles': 2,'3 to 4': 3,'5 to 8': 5,'
```

In [142]:

```
df['NumberOfCars'].value_counts()
```

Out[142]:

```
1    14316
2      709
3      372
5       21
8        2
Name: NumberOfCars, dtype: int64
```

In [143]:

```
NumberOfCars_value = {'1 vehicle': 1,'2 vehicles': 2,'3 to 4': 3,'5 to 8': 5,'more t
```

BasePolicy

In [144]:

```
df['BasePolicy'].value_counts()
```

Out[144]:

```
Collision     5962
Liability     5009
All Perils    4449
Name: BasePolicy, dtype: int64
```

In [145]:

```
df = pd.get_dummies(df,columns=['BasePolicy'])
```

VehiclePrice

In [146]:

```
df['VehiclePrice'].value_counts()
```

Out[146]:

```
20000 to 29000     8079
30000 to 39000     3533
more than 69000    2164
less than 20000    1096
40000 to 59000      461
60000 to 69000       87
Name: VehiclePrice, dtype: int64
```

In [147]:

```
df['VehiclePrice'].value_counts().to_dict()
```

Out[147]:

```
{'20000 to 29000': 8079,
 '30000 to 39000': 3533,
 'more than 69000': 2164,
 'less than 20000': 1096,
 '40000 to 59000': 461,
 '60000 to 69000': 87}
```

In [148]:

```
df['VehiclePrice'].replace({'20000 to 29000': 2,'30000 to 39000': 3,'more than 69000
                            '40000 to 59000': 4,'60000 to 69000': 6}, inplace= True)
```

In [149]:

```
df['VehiclePrice'].value_counts()
```

Out[149]:

```
2    8079
3    3533
7    2164
1    1096
4     461
6      87
Name: VehiclePrice, dtype: int64
```

In [150]:

```
VehiclePrice_value = {'20000 to 29000': 2,'30000 to 39000': 3,'more than 69000': 7,'
                      '40000 to 59000': 4,'60000 to 69000': 6}
```

Days_Policy_Accident

In [151]:

```
df['Days_Policy_Accident'].value_counts()
```

Out[151]:

```
more than 30    15247
none               55
8 to 15            55
15 to 30           49
1 to 7             14
Name: Days_Policy_Accident, dtype: int64
```

In [152]:

```
df['Days_Policy_Accident'].value_counts().to_dict()
```

Out[152]:

```
{'more than 30': 15247,
 'none': 55,
 '8 to 15': 55,
 '15 to 30': 49,
 '1 to 7': 14}
```

In [153]:

```
df['Days_Policy_Accident'].replace({'more than 30': 30,'none': 0,'8 to 15': 8,'15 tc
```

In [154]:

```
df['Days_Policy_Accident'].value_counts()
```

Out[154]:

```
30    15247
0        55
8        55
15       49
1        14
Name: Days_Policy_Accident, dtype: int64
```

In [155]:

```
Days_Policy_Accident_value = {'more than 30': 30,'none': 0,'8 to 15': 8,'15 to 30':
```

PastNumberOfClaims

In [156]:

```
df['PastNumberOfClaims'].value_counts()
```

Out[156]:

```
2 to 4         5485
none           4352
1              3573
more than 4    2010
Name: PastNumberOfClaims, dtype: int64
```

In [157]:

```
df['PastNumberOfClaims'].value_counts().to_dict()
```

Out[157]:

```
{'2 to 4': 5485, 'none': 4352, '1': 3573, 'more than 4': 2010}
```

In [158]:

```
df['PastNumberOfClaims'].replace({'2 to 4': 2, 'none': 0, '1': 1, 'more than 4': 4},
```

In [159]:

```
df['PastNumberOfClaims'].value_counts()
```

Out[159]:

```
2    5485
0    4352
1    3573
4    2010
Name: PastNumberOfClaims, dtype: int64
```

In [160]:

```
PastNumberOfClaims_value = {'2 to 4': 2, 'none': 0, '1': 1, 'more than 4': 4}
```

AgeOfVehicle

In [161]:

```
df['AgeOfVehicle'].value_counts()
```

Out[161]:

```
7 years        5807
more than 7    3981
6 years        3448
5 years        1357
new             373
4 years         229
3 years         152
2 years          73
Name: AgeOfVehicle, dtype: int64
```

In [162]:

```
df['AgeOfVehicle'].value_counts().to_dict()
```

Out[162]:

```
{'7 years': 5807,
 'more than 7': 3981,
 '6 years': 3448,
 '5 years': 1357,
 'new': 373,
 '4 years': 229,
 '3 years': 152,
 '2 years': 73}
```

In [163]:

```
df['AgeOfVehicle'].replace({'7 years': 7,'more than 7': 8,'6 years': 6,'5 years': 5,
                            '4 years': 4,'3 years': 3,'2 years': 2}, inplace= True)
```

In [164]:

```
df['AgeOfVehicle'].value_counts()
```

Out[164]:

```
7    5807
8    3981
6    3448
5    1357
0     373
4     229
3     152
2      73
Name: AgeOfVehicle, dtype: int64
```

In [165]:

```
AgeOfVehicle_value = {'7 years': 7,'more than 7': 8,'6 years': 6,'5 years': 5,'new':
                            '4 years': 4,'3 years': 3,'2 years': 2}
```

AgeOfPolicyHolder

In [166]:

```
df['AgeOfPolicyHolder'].value_counts()
```

Out[166]:

```
31 to 35    5593
36 to 40    4043
41 to 50    2828
51 to 65    1392
26 to 30     613
over 65      508
16 to 17     320
21 to 25     108
18 to 20      15
Name: AgeOfPolicyHolder, dtype: int64
```

In [167]:

```
df['AgeOfPolicyHolder'].value_counts().to_dict()
```

Out[167]:

```
{'31 to 35': 5593,
 '36 to 40': 4043,
 '41 to 50': 2828,
 '51 to 65': 1392,
 '26 to 30': 613,
 'over 65': 508,
 '16 to 17': 320,
 '21 to 25': 108,
 '18 to 20': 15}
```

In [168]:

```
df['AgeOfPolicyHolder'].replace({'31 to 35': 31,'36 to 40': 36,'41 to 50': 41,'51 to
                                 'over 65': 65,'16 to 17': 16,'21 to 25': 21,'18 to
```

In [169]:

```
df['AgeOfPolicyHolder'].value_counts()
```

Out[169]:

```
31    5593
36    4043
41    2828
51    1392
16     933
65     508
21     108
18      15
Name: AgeOfPolicyHolder, dtype: int64
```

In [170]:

```
AgeOfPolicyHolder_value = {'31 to 35': 31,'36 to 40': 36,'41 to 50': 41,'51 to 65':
                                 'over 65': 65,'16 to 17': 16,'21 to 25': 21,'18 to
```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [ ]:

```

```

In [171]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15420 entries, 0 to 15419
Data columns (total 96 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   WeekOfMonth                    15420 non-null  int64
 1   AccidentArea                   15420 non-null  int64
 2   WeekOfMonthClaimed             15420 non-null  int64
 3   Sex                            15420 non-null  int64
 4   MaritalStatus                  15420 non-null  int64
 5   Age                            15420 non-null  int64
 6   Fault                          15420 non-null  int64
 7   VehicleCategory                15420 non-null  int64
 8   VehiclePrice                   15420 non-null  int64
 9   FraudFound_P                   15420 non-null  int64
 10  PolicyNumber                   15420 non-null  int64
 11  RepNumber                      15420 non-null  int64
 12  Deductible                     15420 non-null  int64
 13  DriverRating                   15420 non-null  int64
 14  Days_Policy_Accident           15420 non-null  int64
 15  Days_Policy_Claim              15420 non-null  int64
 16  PastNumberOfClaims             15420 non-null  int64
 17  AgeOfVehicle                   15420 non-null  int64
 18  AgeOfPolicyHolder              15420 non-null  int64
 19  PoliceReportFiled              15420 non-null  int64
 20  WitnessPresent                 15420 non-null  int64
 21  AgentType                      15420 non-null  int64
 22  NumberOfSuppliments            15420 non-null  int64
 23  AddressChange_Claim            15420 non-null  int64
 24  NumberOfCars                   15420 non-null  int64
 25  Year                           15420 non-null  int64
 26  Month_Apr                      15420 non-null  uint8
 27  Month_Aug                      15420 non-null  uint8
 28  Month_Dec                      15420 non-null  uint8
 29  Month_Feb                      15420 non-null  uint8
 30  Month_Jan                      15420 non-null  uint8
 31  Month_Jul                      15420 non-null  uint8
 32  Month_Jun                      15420 non-null  uint8
 33  Month_Mar                      15420 non-null  uint8
 34  Month_May                      15420 non-null  uint8
 35  Month_Nov                      15420 non-null  uint8
 36  Month_Oct                      15420 non-null  uint8
 37  Month_Sep                      15420 non-null  uint8
 38  DayOfWeek_Friday               15420 non-null  uint8
 39  DayOfWeek_Monday               15420 non-null  uint8
 40  DayOfWeek_Saturday             15420 non-null  uint8
 41  DayOfWeek_Sunday               15420 non-null  uint8
 42  DayOfWeek_Thursday             15420 non-null  uint8
 43  DayOfWeek_Tuesday              15420 non-null  uint8
 44  DayOfWeek_Wednesday            15420 non-null  uint8
 45  Make_Accura                    15420 non-null  uint8
 46  Make_BMW                       15420 non-null  uint8
 47  Make_Chevrolet                 15420 non-null  uint8
 48  Make_Dodge                     15420 non-null  uint8
 49  Make_Ferrari                   15420 non-null  uint8
 50  Make_Ford                      15420 non-null  uint8
 51  Make_Honda                     15420 non-null  uint8
 52  Make_Jaguar                    15420 non-null  uint8
 53  Make_Lexus                     15420 non-null  uint8
 54  Make_Mazda                     15420 non-null  uint8
 55  Make_Mecedes                   15420 non-null  uint8
 56  Make_Mercury                   15420 non-null  uint8
 57  Make_Nisson                    15420 non-null  uint8
 58  Make_Pontiac                   15420 non-null  uint8
 59  Make_Porche                    15420 non-null  uint8
 60  Make_Saab                      15420 non-null  uint8
 61  Make_Saturn                    15420 non-null  uint8
 62  Make_Toyota                    15420 non-null  uint8
 63  Make_VW                        15420 non-null  uint8
 64  DayOfWeekClaimed_Friday        15420 non-null  uint8
 65  DayOfWeekClaimed_Monday        15420 non-null  uint8
 66  DayOfWeekClaimed_Saturday      15420 non-null  uint8
 67  DayOfWeekClaimed_Sunday        15420 non-null  uint8
 68  DayOfWeekClaimed_Thursday      15420 non-null  uint8
 69  DayOfWeekClaimed_Tuesday       15420 non-null  uint8
 70  DayOfWeekClaimed_Wednesday     15420 non-null  uint8
 71  MonthClaimed_0                 15420 non-null  uint8
 72  MonthClaimed_Apr               15420 non-null  uint8
 73  MonthClaimed_Aug               15420 non-null  uint8
 74  MonthClaimed_Dec               15420 non-null  uint8
 75  MonthClaimed_Feb               15420 non-null  uint8
 76  MonthClaimed_Jan               15420 non-null  uint8
 77  MonthClaimed_Jul               15420 non-null  uint8
 78  MonthClaimed_Jun               15420 non-null  uint8
 79  MonthClaimed_Mar               15420 non-null  uint8
 80  MonthClaimed_May               15420 non-null  uint8
 81  MonthClaimed_Nov               15420 non-null  uint8
 82  MonthClaimed_Oct               15420 non-null  uint8
 83  MonthClaimed_Sep               15420 non-null  uint8
 84  PolicyType_Sedan_All_Perils    15420 non-null  uint8
 85  PolicyType_Sedan_Collision     15420 non-null  uint8
 86  PolicyType_Sedan_Liability     15420 non-null  uint8
 87  PolicyType_Sport_All_Perils    15420 non-null  uint8
 88  PolicyType_Sport_Collision     15420 non-null  uint8
 89  PolicyType_Sport_Liability     15420 non-null  uint8
 90  PolicyType_Utility_All_Perils  15420 non-null  uint8
 91  PolicyType_Utility_Collision   15420 non-null  uint8
 92  PolicyType_Utility_Liability   15420 non-null  uint8
 93  BasePolicy_All Perils          15420 non-null  uint8
 94  BasePolicy_Collision           15420 non-null  uint8
 95  BasePolicy_Liability           15420 non-null  uint8
dtypes: int64(26), uint8(70)
memory usage: 4.1 MB
```

Feature Selection

In [ ]:

```

```

Model Building

In [172]:

```
# Train-Test Split

x = df.drop('FraudFound_P', axis = 1)
y = df['FraudFound_P']

x_train, x_test, y_train, y_test = train_test_split(x,y, test_size=0.30, stratify= y

print("Training data size : ", x_train.shape)
print("Testing data size : ", x_test.shape)
```

```
Training data size :  (10794, 95)
Testing data size :  (4626, 95)
```

In [173]:

```
model_details = []
Testing_accuracy = []
Training_accuracy = []
best_params_list = []
```

In [174]:

```
logistic_model = LogisticRegression()
logistic_model.fit(x_train, y_train)
model_details.append("Logistic Regression")
```

In [175]:

```
def model_evalution_testing(logistic_model, x_test, y_test):

    # Testing Data Evaluation
    y_pred = logistic_model.predict(x_test)

    cnf_matrix = confusion_matrix(y_pred, y_test)
    print("Confusion Matrix:\n",cnf_matrix)

    print("*"*84)

    accuracy = accuracy_score(y_pred, y_test)
    print("Accuracy Score", accuracy)

   # We are appending testing accuracy in list
    Testing_accuracy.append(accuracy)
    print("*"*84)

    clf_report = classification_report(y_pred, y_test)
    print("Classification report:\n",clf_report)
```

In [176]:

```
def model_evalution_training(logistic_model, x_train, y_train):
    # Training Data Evaluation
    y_pred_train = logistic_model.predict(x_train)

    cnf_matrix = confusion_matrix(y_pred_train, y_train)
    print("Confusion Matrix:\n",cnf_matrix)

    print("*"*84)

    accuracy = accuracy_score(y_pred_train, y_train)
    print("Accuracy Score", accuracy)
# We are appending training accuracy in list
    Training_accuracy.append(accuracy)

    print("*"*84)

    clf_report = classification_report(y_pred_train, y_train)
    print("Classification report:\n",clf_report)
```

In [177]:

```
model_evalution_training(logistic_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10148   646]
 [    0     0]]
**************************************************************************
**********
Accuracy Score 0.9401519362608857
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97     10794
           1       0.00      0.00      0.00         0

    accuracy                           0.94     10794
   macro avg       0.50      0.47      0.48     10794
weighted avg       1.00      0.94      0.97     10794
```

In [178]:

```
model_evalution_testing(logistic_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4349  277]
 [   0    0]]
**************************************************************************
**********
Accuracy Score 0.9401210549070471
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97      4626
           1       0.00      0.00      0.00         0

    accuracy                           0.94      4626
   macro avg       0.50      0.47      0.48      4626
weighted avg       1.00      0.94      0.97      4626
```

In [179]:

```
## Checking Accuracy using Decision Tree Algorithm
```

In [180]:

```
dt_model = DecisionTreeClassifier(random_state= 5)
dt_model.fit(x_train, y_train)
model_details.append("Decision Tree Classifier")
```

In [181]:

```
model_evalution_training(dt_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10148     0]
 [    0   646]]
**************************************************************************
**********
Accuracy Score 1.0
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     10148
           1       1.00      1.00      1.00       646

    accuracy                           1.00     10794
   macro avg       1.00      1.00      1.00     10794
weighted avg       1.00      1.00      1.00     10794
```

In [182]:

```
model_evalution_testing(dt_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4195  166]
 [ 154  111]]
**************************************************************************
**********
Accuracy Score 0.9308257674016429
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       0.96      0.96      0.96      4361
           1       0.40      0.42      0.41       265

    accuracy                           0.93      4626
   macro avg       0.68      0.69      0.69      4626
weighted avg       0.93      0.93      0.93      4626
```

```
## Hyper Parameter Tunning for Decision Tree Model
dt_model = DecisionTreeClassifier()
param_grid = {'criterion':['gini', 'entropy'],
                'max_depth' :np.arange(3,8),
                'min_samples_split' : np.arange(2,20),
               'min_samples_leaf' :np.arange(2,15)}

gscv_dt_model =  GridSearchCV(dt_model, param_grid, cv=5)
gscv_dt_model.fit(x_train, y_train)
gscv_dt_model.best_estimator_
```

In [183]:

```
## Hyper Parameter Tunning for Decision Tree Model
dt_model = DecisionTreeClassifier()
param_grid = {'criterion':['gini', 'entropy'],
                'max_depth' :np.arange(3,8),
                'min_samples_split' : np.arange(2,20),
               'min_samples_leaf' :np.arange(2,15)}

gscv_dt_model =  RandomizedSearchCV(dt_model, param_grid, cv=5)
gscv_dt_model.fit(x_train, y_train)
gscv_dt_model.best_estimator_
```

Out[183]:

```
DecisionTreeClassifier(max_depth=6, min_samples_leaf=2, min_samples_split=
4)
```

In [184]:

```
gscv_dt_model = gscv_dt_model.best_estimator_
gscv_dt_model.fit(x_train, y_train)
model_details.append("Decision Tree Classifier with Hyper parameter tunning")
```

In [185]:

```
model_evalution_training(gscv_dt_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10131   581]
 [   17    65]]
**************************************************************************
**********
Accuracy Score 0.9445988512136372
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.95      0.97     10712
           1       0.10      0.79      0.18        82

    accuracy                           0.94     10794
   macro avg       0.55      0.87      0.57     10794
weighted avg       0.99      0.94      0.97     10794
```

In [186]:

```
model_evalution_testing(gscv_dt_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4332  264]
 [  17   13]]
**************************************************************************
**********
Accuracy Score 0.9392563769995677
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97      4596
           1       0.05      0.43      0.08        30

    accuracy                           0.94      4626
   macro avg       0.52      0.69      0.53      4626
weighted avg       0.99      0.94      0.96      4626
```

In [187]:

```
# Prunning
```

In [188]:

```
result = dt_model.cost_complexity_pruning_path(x_train, y_train)
ccp_alpha_list = result['ccp_alphas']
ccp_alpha_list
```

Out[188]:

```
array([0.00000000e+00, 5.88216264e-05, 6.01790485e-05, 6.05973736e-05,
       6.10764554e-05, 6.11292440e-05, 6.17360168e-05, 6.22722328e-05,
       7.21648479e-05, 7.33272933e-05, 7.86316348e-05, 7.93085927e-05,
       8.10635538e-05, 8.42218741e-05, 8.42218741e-05, 8.49237231e-05,
       8.60266286e-05, 8.64677907e-05, 8.66670253e-05, 8.66670253e-05,
       8.70976078e-05, 8.71944108e-05, 8.74971692e-05, 8.76543127e-05,
       8.77680583e-05, 8.77680583e-05, 8.80118584e-05, 8.80118584e-05,
       8.87838923e-05, 8.87838923e-05, 8.89382991e-05, 8.93742711e-05,
       8.99192362e-05, 8.99192362e-05, 8.99456908e-05, 9.00706154e-05,
       9.03962110e-05, 9.06300602e-05, 9.06444541e-05, 9.10896310e-05,
       9.13013940e-05, 9.15278680e-05, 9.17357864e-05, 9.17862461e-05,
       9.18375733e-05, 9.23563470e-05, 9.25909094e-05, 9.26440615e-05,
       9.26440615e-05, 9.26440615e-05, 9.26440615e-05, 9.26440615e-05,
       9.33241151e-05, 1.01048102e-04, 1.01704566e-04, 1.04003857e-04,
       1.05878927e-04, 1.07547671e-04, 1.09800369e-04, 1.11172874e-04,
       1.14375385e-04, 1.14375385e-04, 1.15452141e-04, 1.17499785e-04,
       1.20334091e-04, 1.23525415e-04, 1.23525415e-04, 1.23525415e-04,
       1.23525415e-04, 1.23525415e-04, 1.23525415e-04, 1.23525415e-04,
       1.23525415e-04, 1.23525415e-04, 1.23525415e-04, 1.23525415e-04,
       1.23525415e-04, 1.23525415e-04, 1.23525415e-04, 1.23525415e-04,
       1.23525415e-04, 1.23525415e-04, 1.23525415e-04, 1.31760443e-04,
       1.33983901e-04, 1.38966092e-04, 1.38966092e-04, 1.38966092e-04,
       1.38966092e-04, 1.38966092e-04, 1.38966092e-04, 1.39951212e-04,
       1.40436633e-04, 1.43063810e-04, 1.43717070e-04, 1.44781412e-04,
       1.47314636e-04, 1.48230498e-04, 1.48230498e-04, 1.48230498e-04,
       1.48230498e-04, 1.48230498e-04, 1.48230498e-04, 1.49401302e-04,
       1.49465753e-04, 1.51318634e-04, 1.51599373e-04, 1.54406769e-04,
       1.54406769e-04, 1.54406769e-04, 1.54406769e-04, 1.54406769e-04,
       1.54890625e-04, 1.55012286e-04, 1.55356965e-04, 1.56217856e-04,
       1.57588801e-04, 1.57982505e-04, 1.58441800e-04, 1.59987359e-04,
       1.60908107e-04, 1.63489520e-04, 1.63952606e-04, 1.64057192e-04,
       1.64700554e-04, 1.64700554e-04, 1.64700554e-04, 1.64700554e-04,
       1.66759311e-04, 1.66759311e-04, 1.66759311e-04, 1.66759311e-04,
       1.67489648e-04, 1.69458178e-04, 1.69610787e-04, 1.70465073e-04,
       1.71126858e-04, 1.72886221e-04, 1.72935581e-04, 1.74388822e-04,
       1.74994338e-04, 1.75536117e-04, 1.75680591e-04, 1.77232118e-04,
       1.77567785e-04, 1.77935420e-04, 1.78380026e-04, 1.78898877e-04,
       1.79673331e-04, 1.80204190e-04, 1.83108984e-04, 1.83295778e-04,
       1.86149928e-04, 1.90039101e-04, 1.91184685e-04, 1.91547742e-04,
       1.94111367e-04, 1.94453445e-04, 2.05655707e-04, 2.16996250e-04,
       2.18881178e-04, 2.29294052e-04, 2.29370407e-04, 2.40032341e-04,
       2.47050831e-04, 2.47050831e-04, 2.53367471e-04, 2.53589159e-04,
       2.64697319e-04, 2.65498376e-04, 2.67283371e-04, 2.69035746e-04,
       2.72878872e-04, 2.77932185e-04, 2.92081459e-04, 2.97253673e-04,
       3.07990700e-04, 3.09309971e-04, 3.12673708e-04, 3.13734194e-04,
       3.13960431e-04, 3.17896290e-04, 3.20277071e-04, 3.23411997e-04,
       3.30613337e-04, 3.37319591e-04, 3.47415231e-04, 3.48944353e-04,
       3.55870839e-04, 3.61479177e-04, 3.69936287e-04, 3.71017408e-04,
       3.75208449e-04, 3.80078201e-04, 3.85431826e-04, 3.95281329e-04,
       4.04222534e-04, 4.32473477e-04, 4.46240943e-04, 4.53055017e-04,
       5.00598777e-04, 5.24217484e-04, 5.26950520e-04, 6.48963824e-04,
       6.73046587e-04, 6.78067099e-04, 7.02896163e-04, 1.15007254e-03,
       1.63990638e-03, 1.65588339e-03, 3.05204910e-03])
```

In [189]:

```
train_accuracy_list = []
test_accuracy_list = []

for i in ccp_alpha_list:
    decision_tree_model = DecisionTreeClassifier(ccp_alpha=i, random_state=11)
    decision_tree_model.fit(x_train, y_train)

    training_accuracy = decision_tree_model.score(x_train, y_train)
    train_accuracy_list.append(training_accuracy)

    testing_accuracy = decision_tree_model.score(x_test, y_test)
    test_accuracy_list.append(testing_accuracy)
```

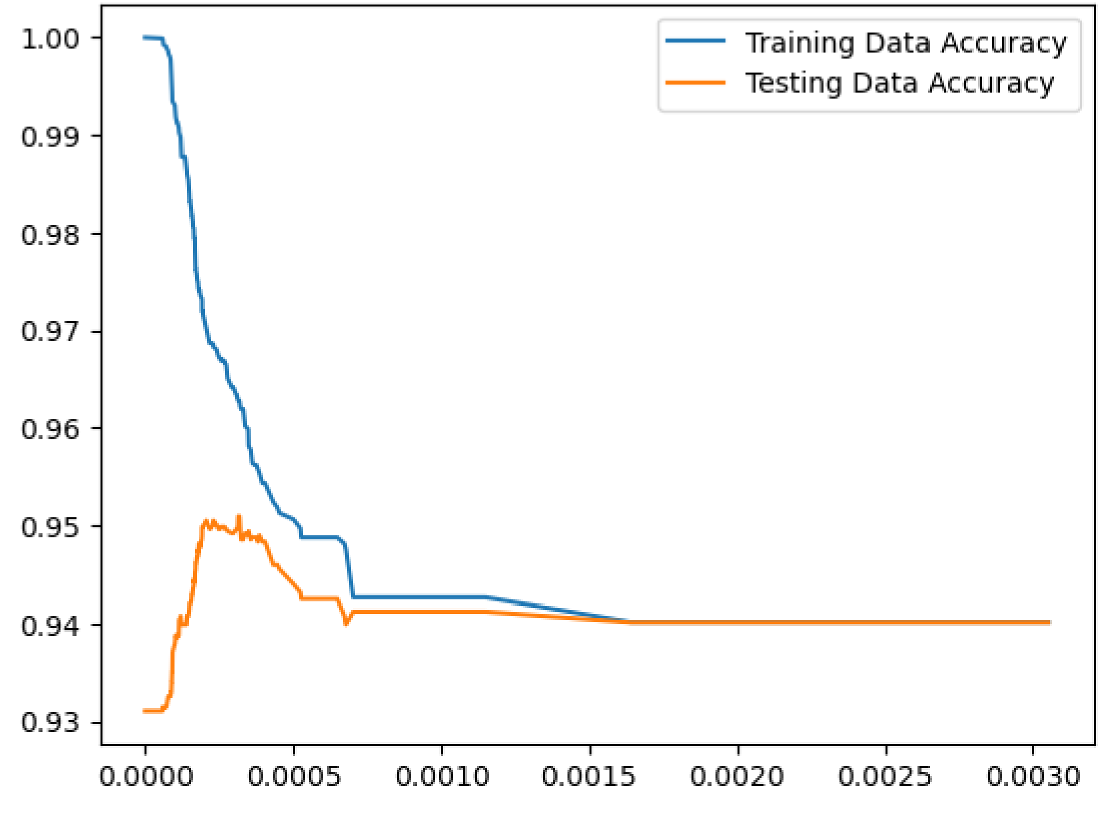
In [190]:

```
fig, ax = plt.subplots()
ax.plot(ccp_alpha_list, train_accuracy_list, label = "Training Data Accuracy")
ax.plot(ccp_alpha_list, test_accuracy_list, label = "Testing Data Accuracy")
ax.legend()

```

Out[190]:

```
<matplotlib.legend.Legend at 0x16e6b5d66d0>
```



In [191]:

```
max_test = test_accuracy_list.index(max(test_accuracy_list))
max_test
```

Out[191]:

```
179
```

In [192]:

```
best_ccp = ccp_alpha_list[max_test]
best_ccp
```

Out[192]:

```
0.00031373419366748957
```

In [193]:

```
dt_prunning_model = DecisionTreeClassifier(ccp_alpha= best_ccp, random_state=11)
dt_prunning_model.fit(x_train, y_train)
model_details.append("Decision Tree with Prunnung")
```

In [194]:

```
model_evalution_training(dt_prunning_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10079   333]
 [   69   313]]
**************************************************************************
**********
Accuracy Score 0.962757087270706
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       0.99      0.97      0.98     10412
           1       0.48      0.82      0.61       382

    accuracy                           0.96     10794
   macro avg       0.74      0.89      0.79     10794
weighted avg       0.98      0.96      0.97     10794
```

In [195]:

```
model_evalution_testing(dt_prunning_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4302  180]
 [  47   97]]
**************************************************************************
**********
Accuracy Score 0.9509295287505404
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       0.99      0.96      0.97      4482
           1       0.35      0.67      0.46       144

    accuracy                           0.95      4626
   macro avg       0.67      0.82      0.72      4626
weighted avg       0.97      0.95      0.96      4626
```

In [196]:

```
rf_model = RandomForestClassifier(random_state=12)
rf_model.fit(x_train, y_train)
model_details.append("Random Forest Classifier")
```

In [197]:

```
model_evalution_training(rf_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10148     0]
 [    0   646]]
**************************************************************************
**********
Accuracy Score 1.0
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     10148
           1       1.00      1.00      1.00       646

    accuracy                           1.00     10794
   macro avg       1.00      1.00      1.00     10794
weighted avg       1.00      1.00      1.00     10794
```

In [198]:

```
model_evalution_testing(rf_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4348  275]
 [   1    2]]
**************************************************************************
**********
Accuracy Score 0.940337224383917
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97      4623
           1       0.01      0.67      0.01         3

    accuracy                           0.94      4626
   macro avg       0.50      0.80      0.49      4626
weighted avg       1.00      0.94      0.97      4626
```

In [199]:

```
# Hyper-parameter tunning for random forest

rf_model = RandomForestClassifier(random_state=12)

param_grid = {'criterion':['gini', 'entropy'],
              'n_estimators': np.arange(50,200),
                'max_depth' :np.arange(3,8),
                'min_samples_split' : np.arange(2,20),
               'min_samples_leaf' :np.arange(2,15),
              'oob_score': [False, True]}

rscv_rf_model =  RandomizedSearchCV(rf_model, param_grid, cv=5)
rscv_rf_model.fit(x_train, y_train)
rscv_rf_model.best_estimator_
```

Out[199]:

```
RandomForestClassifier(criterion='entropy', max_depth=7, min_samples_leaf=
9,
                       min_samples_split=3, n_estimators=141, random_state
=12)
```

In [200]:

```
rscv_rf_model = rscv_rf_model.best_estimator_
rscv_rf_model.fit(x_train, y_train)
model_details.append("DT with RandmizedSearchCV")
```

In [201]:

```
model_evalution_training(rscv_rf_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10148   646]
 [    0     0]]
***************************************************************************
**********
Accuracy Score 0.9401519362608857
***************************************************************************
**********
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97     10794
           1       0.00      0.00      0.00         0

    accuracy                           0.94     10794
   macro avg       0.50      0.47      0.48     10794
weighted avg       1.00      0.94      0.97     10794
```

In [202]:

```
model_evalution_testing(rscv_rf_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4349  277]
 [   0    0]]
***************************************************************************
**********
Accuracy Score 0.9401210549070471
***************************************************************************
**********
Classification report:
              precision    recall  f1-score   support

           0       1.00      0.94      0.97      4626
           1       0.00      0.00      0.00         0

    accuracy                           0.94      4626
   macro avg       0.50      0.47      0.48      4626
weighted avg       1.00      0.94      0.97      4626
```

```
# Hyper-parameter tunning for random forest by using GridSearchCV

rf_model = RandomForestClassifier(random_state=12)

param_grid = {'criterion':['gini', 'entropy'],
              'n_estimators': np.arange(50,200),
                'max_depth' :np.arange(3,8),
                'min_samples_split' : np.arange(2,20),
               'min_samples_leaf' :np.arange(2,15),
              'oob_score': [False, True]}

gscv_rf_model =  GridSearchCV(rf_model, param_grid, cv=5)
gscv_rf_model.fit(x_train, y_train)
gscv_rf_model.best_estimator_
```

In [203]:

```
adb_model = AdaBoostClassifier(random_state= 14)
adb_model.fit(x_train, y_train)
model_details.append("Adaboost Model")
```

In [204]:

```
model_evalution_training(adb_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10092   636]
 [   56    10]]
**************************************************************************
**********
Accuracy Score 0.9358903094311655
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       0.99      0.94      0.97     10728
           1       0.02      0.15      0.03        66

    accuracy                           0.94     10794
   macro avg       0.50      0.55      0.50     10794
weighted avg       0.99      0.94      0.96     10794
```

In [205]:

```
model_evalution_testing(adb_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4324  272]
 [  25    5]]
**************************************************************************
**********
Accuracy Score 0.9357976653696498
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       0.99      0.94      0.97      4596
           1       0.02      0.17      0.03        30

    accuracy                           0.94      4626
   macro avg       0.51      0.55      0.50      4626
weighted avg       0.99      0.94      0.96      4626
```

In [206]:

```
# Hyper-parameter tunning for Adaboost by using RandomizedSearchCv

adb_model = AdaBoostClassifier(random_state=12)

param_grid = {"n_estimators":np.arange(30, 100),
             "learning_rate": np.arange(0,2,0.001)}

rscv_adb_model =  RandomizedSearchCV(adb_model, param_grid, cv=5, n_jobs= -1)
rscv_adb_model.fit(x_train, y_train)
rscv_adb_model.best_estimator_
```

Out[206]:

```
AdaBoostClassifier(learning_rate=0.11800000000000001, n_estimators=64,
                   random_state=12)
```

In [207]:

```
rscv_adb_model = rscv_adb_model.best_estimator_
rscv_adb_model.fit(x_train, y_train)
model_details.append("Adaboost with RandomizedSearchCV")
```

In [208]:

```
model_evalution_training(rscv_adb_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10148   646]
 [    0     0]]
**************************************************************************
**********
Accuracy Score 0.9401519362608857
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97     10794
           1       0.00      0.00      0.00         0

    accuracy                           0.94     10794
   macro avg       0.50      0.47      0.48     10794
weighted avg       1.00      0.94      0.97     10794
```

In [209]:

```
model_evalution_testing(rscv_adb_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4349  277]
 [   0    0]]
**************************************************************************
**********
Accuracy Score 0.9401210549070471
**************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97      4626
           1       0.00      0.00      0.00         0

    accuracy                           0.94      4626
   macro avg       0.50      0.47      0.48      4626
weighted avg       1.00      0.94      0.97      4626
```

In [210]:

```
# Hyper-parameter tunning for Adaboost by using RandomizedSearchCv

adb_model = AdaBoostClassifier(random_state=12)

param_grid = {"n_estimators":np.arange(30, 100),
            "learning_rate": np.arange(0,2,0.001)}

rscv_adb_model =  RandomizedSearchCV(adb_model, param_grid, cv=5, n_jobs= -1)
rscv_adb_model.fit(x_train, y_train)
rscv_adb_model.best_estimator_
```

Out[210]:

```
AdaBoostClassifier(learning_rate=0.365, n_estimators=77, random_state=12)
```

```
# Hyper-parameter tunning for Adaboost by using RandomizedSearchCv

adb_model = AdaBoostClassifier(random_state=12)

param_grid = {"n_estimators":np.arange(30, 100),
            "learning_rate": np.arange(0,2,0.001)}

gscv_adb_model =  GridSearchCV(adb_model, param_grid, cv=5, n_jobs= -1)
gscv_adb_model.fit(x_train, y_train)
gscv_adb_model.best_estimator_
```

In [211]:

```
rscv_adb_model = rscv_adb_model.best_estimator_
rscv_adb_model.fit(x_train, y_train)
model_details.append("Adaboost with GridSearchCV")
```

In [212]:

```
model_evalution_training(rscv_adb_model, x_train, y_train)
```

```
Confusion Matrix:
 [[10098   637]
 [   50     9]]
***************************************************************************
**********
Accuracy Score 0.9363535297387438
***************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97     10735
           1       0.01      0.15      0.03        59

    accuracy                           0.94     10794
   macro avg       0.50      0.55      0.50     10794
weighted avg       0.99      0.94      0.96     10794
```

In [213]:

```
model_evalution_testing(rscv_adb_model, x_test, y_test)
```

```
Confusion Matrix:
 [[4330  274]
 [  19    3]]
***************************************************************************
**********
Accuracy Score 0.9366623432771293
***************************************************************************
**********
Classification report:
               precision    recall  f1-score   support

           0       1.00      0.94      0.97      4604
           1       0.01      0.14      0.02        22

    accuracy                           0.94      4626
   macro avg       0.50      0.54      0.49      4626
weighted avg       0.99      0.94      0.96      4626
```

In [214]:

```
comparison_df = pd.DataFrame({"Model_details":model_details, "Training_accuracy":Tra
                              "Testing_accuracy":Testing_accuracy})
comparison_df
```

Out[214]:

```
                                 Model_details  Training_accuracy  Testing_accuracy
0                          Logistic Regression           0.940152          0.940121
1                     Decision Tree Classifier           1.000000          0.930826
2  Decision Tree Classifier with Hyper parameter ...     0.944599          0.939256
3                     Decision Tree with Prunnung        0.962757          0.950930
4                     Random Forest Classifier           1.000000          0.940337
5                  DT with RandmizedSearchCV             0.940152          0.940121
6                              Adaboost Model            0.935890          0.935798
7            Adaboost with RandomizedSearchCV            0.940152          0.940121
8                 Adaboost with GridSearchCV             0.936354          0.936662
```

Create JSON File

In [215]:

```
json_data = {'AccidentArea': AccidentArea_value,
             'Sex':Sex_value,
             'MaritalStatus':MaritalStatus_value,
             'Fault':Fault_value,
             'VehicleCategory':VehicleCategory_value,
             'Days_Policy_Claim':Days_Policy_Claim_value,
             'PoliceReportFiled':PoliceReportFiled_value,
             'WitnessPresent':WitnessPresent_value,
             'AgentType':AgentType_value,
             'NumberOfSuppliments':NumberOfSuppliments_value,
             'AddressChange_Claim':AddressChange_Claim_value,
             'NumberOfCars':NumberOfCars_value,
             'VehiclePrice':VehiclePrice_value,
             'Days_Policy_Accident':Days_Policy_Accident_value,
             'PastNumberOfClaims':PastNumberOfClaims_value,
             'AgeOfVehicle':AgeOfVehicle_value,
             'AgeOfPolicyHolder':AgeOfPolicyHolder_value,
             'columns':list(x.columns)}

json_data
```

```
  'RepNumber',
  'Deductible',
  'DriverRating',
  'Days_Policy_Accident',
  'Days_Policy_Claim',
  'PastNumberOfClaims',
  'AgeOfVehicle',
  'AgeOfPolicyHolder',
  'PoliceReportFiled',
  'WitnessPresent',
  'AgentType',
  'NumberOfSuppliments',
  'AddressChange_Claim',
  'NumberOfCars',
  'Year',
  'Month_Apr',
  'Month_Aug',
  'Month_Dec',
  'Month_Feb',
  'Month_Jan',
```

In [216]:

```
with open ("project_data.json", 'w') as f:
    json.dump(json_data,f)
```

Create Pickle File

In [217]:

```
with open(" Decision Tree_Prunnung.pkl", "wb") as f:
    pickle.dump(dt_prunning_model, f)
```

Testing for User Input Values

In [231]:

```
column_names = x.columns
column_names
```

Out[231]:

```
Index(['WeekOfMonth', 'AccidentArea', 'WeekOfMonthClaimed', 'Sex',
       'MaritalStatus', 'Age', 'Fault', 'VehicleCategory', 'VehiclePrice',
       'PolicyNumber', 'RepNumber', 'Deductible', 'DriverRating',
       'Days_Policy_Accident', 'Days_Policy_Claim', 'PastNumberOfClaims',
       'AgeOfVehicle', 'AgeOfPolicyHolder', 'PoliceReportFiled',
       'WitnessPresent', 'AgentType', 'NumberOfSuppliments',
       'AddressChange_Claim', 'NumberOfCars', 'Year', 'Month_Apr', 'Month_
Aug',
       'Month_Dec', 'Month_Feb', 'Month_Jan', 'Month_Jul', 'Month_Jun',
       'Month_Mar', 'Month_May', 'Month_Nov', 'Month_Oct', 'Month_Sep',
       'DayOfWeek_Friday', 'DayOfWeek_Monday', 'DayOfWeek_Saturday',
       'DayOfWeek_Sunday', 'DayOfWeek_Thursday', 'DayOfWeek_Tuesday',
       'DayOfWeek_Wednesday', 'Make_Accura', 'Make_BMW', 'Make_Chevrolet',
       'Make_Dodge', 'Make_Ferrari', 'Make_Ford', 'Make_Honda', 'Make_Jagu
ar',
       'Make_Lexus', 'Make_Mazda', 'Make_Mecedes', 'Make_Mercury',
       'Make_Nisson', 'Make_Pontiac', 'Make_Porche', 'Make_Saab',
       'Make_Saturn', 'Make_Toyota', 'Make_VW', 'DayOfWeekClaimed_Friday',
       'DayOfWeekClaimed_Monday', 'DayOfWeekClaimed_Saturday',
       'DayOfWeekClaimed_Sunday', 'DayOfWeekClaimed_Thursday',
       'DayOfWeekClaimed_Tuesday', 'DayOfWeekClaimed_Wednesday',
       'MonthClaimed_0', 'MonthClaimed_Apr', 'MonthClaimed_Aug',
       'MonthClaimed_Dec', 'MonthClaimed_Feb', 'MonthClaimed_Jan',
       'MonthClaimed_Jul', 'MonthClaimed_Jun', 'MonthClaimed_Mar',
       'MonthClaimed_May', 'MonthClaimed_Nov', 'MonthClaimed_Oct',
       'MonthClaimed_Sep', 'PolicyType_Sedan_All_Perils',
       'PolicyType_Sedan_Collision', 'PolicyType_Sedan_Liability',
       'PolicyType_Sport_All_Perils', 'PolicyType_Sport_Collision',
       'PolicyType_Sport_Liability', 'PolicyType_Utility_All_Perils',
       'PolicyType_Utility_Collision', 'PolicyType_Utility_Liability',
       'BasePolicy_All Perils', 'BasePolicy_Collision',
       'BasePolicy_Liability'],
      dtype='object')
```

In [220]:

```
WeekOfMonth = 5
AccidentArea = 'Urban'
WeekOfMonthClaimed = 1
Sex = 'Female'
MaritalStatus = 'Single'
Age = 21
Fault = 'Policy Holder'
VehicleCategory = 'Sport'
VehiclePrice = 'more than 69000'
PolicyNumber = 1
RepNumber = 12
Deductible = 300
DriverRating = 1
Days_Policy_Accident = 'more than 30'
Days_Policy_Claim = 'more than 30'
PastNumberOfClaims =  'none'
AgeOfVehicle = '3 years'
AgeOfPolicyHolder = '26 to 30'
PoliceReportFiled = 'No'
WitnessPresent = 'No'
AgentType = 'External'
NumberOfSuppliments = 'none'
AddressChange_Claim = '1 year'
NumberOfCars = '3 to 4'
Year = 1994

# onehot encoded columns

Month_inp = 'Dec'
DayOfWeek_inp = 'Wednesday'
Make_inp = 'Honda'
DayOfWeekClaimed_inp = 'Tuesday'
MonthClaimed_inp = 'Jan'
PolicyType_inp = 'Sport - Liability'
BasePolicy_inp =  'Liability'
```

In [221]:

```
len(x.columns)
```

Out[221]:

```
95
```

In [222]:

```
test_array = np.zeros(len(x.columns), dtype = int)
(test_array)
```

Out[222]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0])
```

In [229]:

```
PolicyType_inp = PolicyType_inp.replace(" - ",'_')

Month_col = 'Month_'+ Month_inp
DayOfWeek_col = 'DayOfWeek_'+ DayOfWeek_inp
Make_col = 'Make_' + Make_inp
DayOfWeekClaimed_col = 'DayOfWeekClaimed_'+ DayOfWeekClaimed_inp
MonthClaimed_col = 'MonthClaimed_'+ MonthClaimed_inp
PolicyType_col = 'PolicyType_' + PolicyType_inp
BasePolicy_col =  'BasePolicy_' + BasePolicy_inp

print(PolicyType_col)
print(Make_col)
print(Month_col)
print(DayOfWeek_col)
print(DayOfWeekClaimed_col)
print(MonthClaimed_col)
print(BasePolicy_col)
```

```
PolicyType_Sport_Liability
Make_Honda
Month_Dec
DayOfWeek_Wednesday
DayOfWeekClaimed_Tuesday
MonthClaimed_Jan
BasePolicy_Liability
```

In [232]:

```
# Find the index of this column -->

PolicyType_index = np.where(column_names == PolicyType_col)[0][0]
Month_index = np.where(column_names == Month_col)[0][0]
DayOfWeek_index = np.where(column_names == DayOfWeek_col)[0][0]
Make_index = np.where(column_names == Make_col)[0][0]
DayOfWeekClaimed_index = np.where(column_names == DayOfWeekClaimed_col)[0][0]
MonthClaimed_index = np.where(column_names == MonthClaimed_col)[0][0]
BasePolicy_index = np.where(column_names == BasePolicy_col)[0][0]
```

In [248]:

```
# df.info()
```

In [242]:

```
test_array[0] = WeekOfMonth
test_array[1] = json_data['AccidentArea'][AccidentArea]
test_array[2] = WeekOfMonthClaimed
test_array[3] = json_data['Sex'][Sex]
test_array[4] = json_data['MaritalStatus'][MaritalStatus]
test_array[5] = Age
test_array[6] = json_data['Fault'][Fault]
test_array[7] = json_data['VehicleCategory'][VehicleCategory]
test_array[8] = json_data['VehiclePrice'][VehiclePrice]
test_array[9] = PolicyNumber
test_array[10] = RepNumber
test_array[11] = Deductible
test_array[12] = DriverRating
test_array[13] = json_data['Days_Policy_Accident'][Days_Policy_Accident]
test_array[14] = json_data['Days_Policy_Claim'][Days_Policy_Claim]
test_array[15] = json_data['PastNumberOfClaims'][PastNumberOfClaims]
test_array[16] = json_data['AgeOfVehicle'][AgeOfVehicle]
test_array[17] = json_data['AgeOfPolicyHolder'][AgeOfPolicyHolder]
test_array[18] = json_data['PoliceReportFiled'][PoliceReportFiled]
test_array[19] = json_data['WitnessPresent'][WitnessPresent]
test_array[20] = json_data['AgentType'][AgentType]
test_array[22] = json_data['NumberOfSuppliments'][NumberOfSuppliments]
test_array[23] = json_data['AddressChange_Claim'][AddressChange_Claim]
test_array[24] = json_data['NumberOfCars'][NumberOfCars]
test_array[25] = Year


test_array[PolicyType_index] = 1
test_array[Month_index] = 1
test_array[DayOfWeek_index] = 1
test_array[Make_index] = 1
test_array[DayOfWeekClaimed_index] = 1
test_array[MonthClaimed_index] = 1
test_array[BasePolicy_index] = 1
test_array
```

Out[242]:

```
array([   5,    1,    1,    0,    1,   21,    1,    1,    7,    1,   12,
        300,    1,   30,   30,    0,    3,   16,    0,    0,    0,    0,
          0,    1,    3, 1994,    0,    1,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    1,
          0,    0,    0,    0,    0,    0,    1,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    1,    0,    0,    0,    0,    0,    0,    1,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          1,    0,    0,    0,    0,    0,    1])
```

In [243]:

```
x.head(1)
```

Out[243]:

	WeekOfMonth	AccidentArea	WeekOfMonthClaimed	Sex	MaritalStatus	Age	Fault	Vehicl
0	5	1	1	0	1	21	1	

1 rows × 95 columns

In [250]:

```
charges = dt_prunning_model.predict([test_array])[0]
charges
```

Out[250]:

```
0
```

In [252]:

```
if charges == 1:
    print("Fraud Insurance Claim Detected")
else:
    print('No Fraud found')
```

```
No Fraud found
```

In [ ]:

```

```