CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in this Minor thesis titled **"Vehicle Insurance Data Analysis"** in fulfilment of the requirement for the degree of Bachelor's in Computer Applications (specialization in Data Science) and submitted to "**SATYUG DARSHAN INSTITUTE OF ENGINEERING AND TECHNOLOGY**", is an authentic record of my own work carried out under the supervision of **Mr. Ankit Mishra.**

The work contained in this thesis has not been submitted to any other University or Institute for the award of any other degree or diploma by me.

Gaurav Moudgil
BCA(DS-23/012)

Sanjay
BCA(DS-23/045)

ACKNOWLEDGEMENT

It is with deep sense of gratitude and reverence that I express my sincere thanks to my supervisor **Mr. Ankit Mishra** for her guidance, encouragement, help and useful suggestions throughout. Her untiring and painstaking efforts, methodical approach and individual help made it possible for me to complete this work in time. I consider myself very fortunate for having been associated with the scholar like her. Her affection, guidance and scientific approach served a veritable incentive for completion of this work.

I shall ever remain indebted to the faculty members of **Satyug Darshan Institute of Engineering and Technology, Faridabad,** team and all my classmates for their cooperation, kindness and general help extended to me during the completion of this work.

Although it is not possible to name individually, I cannot forget my well-wishers at Satyug Darshan Institute of Engineering and Technology, Faridabad and outsiders for their persistent support and cooperation.

This acknowledgement will remain incomplete if I fail to express my deep sense of obligation to my parents and God for their consistent blessings and encouragement

(Gaurav Moudgil)

(Sanjay)

CERTIFICATE

ABOUT TRAINING

As each and every sector of the market is growing, data is building up day by day, we need to keep the record of the data which can be helpful for the analytics and evaluation. Now we don't have data in gigabyte or terabyte but in zetta byte and petabyte and this data cannot be handled with the day By day software such as Excel or MATLAB. Therefore, in this report we will be dealing with large data sets with the high-level programming language 'Python'.

The main goal of this training is to aggregate and analyse the data collected from the different data sources available on the internet like Kaggle etc., This project mainly focuses on the usage of the python programming language and Data Analysis. This language has not only it's application in the field of just analysing the data and represent it graphically.

LIST OF ABBRIVATIONS

- pd for pandas
- np for NumPy
- plt for pyplot
- sns for seaborn
- df for data frame
- int for integer
- len for length
- str for string
- bool for Boolean
- arr for array
- loc for location
- info for information
- col for columns

- hist for histogram
- sqrt for square root

LIST OF FIGURES

CHAPTER 1: INTRODUCTION TO DATA SCIENCE

1.1 Introduction to topic

Vehicle insurance is a crucial component of modern transportation systems, providing financial protection against physical damage or bodily injury resulting from traffic collisions, as well as against liability that could arise from incidents in a vehicle. As the industry grows, so does the complexity of evaluating risk and determining appropriate insurance premiums. This is where data science and machine learning come into play.

1.2 Key Components of the Model

1. **Data Collection and Preparation**: Gathering data from various sources such as past insurance claims, driver demographics, vehicle information, and more. Cleaning and preprocessing the data to make it suitable for analysis.

2. **Feature Engineering**: Identifying and creating relevant features that can help in predicting risk and determining premiums. This may include factors like driver age, driving history, vehicle type, and geographic location.
3. **Model Selection and Training**: Choosing appropriate machine learning algorithms and training them on the prepared data. Commonly used algorithms include logistic regression, decision trees, random forests, and gradient boosting machines.
4. **Model Evaluation**: Assessing the performance of the trained model using metrics such as accuracy, precision, recall, and AUC-ROC. Fine-tuning the model to improve its performance.
5. **Deployment and Integration**: Implementing the model into a real-world system where it can be used to make predictions on new data. This includes creating APIs or integrating with existing insurance platforms.

1.3 Objective of training

The primary objectives of a Python model for vehicle insurance include:

1. **Accurate Risk Prediction**: Using historical data to predict the probability of future claims.
2. **Fair Premium Pricing**: Ensuring premiums are fair and reflect the actual risk associated with the insured driver or vehicle.
3. **Enhanced Customer Experience**: Streamlining the process of obtaining insurance and making claims more efficient and user-friendly.
4. **Operational Efficiency**: Reducing the time and resources required to evaluate and process insurance applications and claims.

CHAPTER 2: PYTHON FOR DATA SCIENCE

1.1. Introduction to Python

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics". This language consist of mainly data structures which make it

very easy for the data scientists to analyse the data very effectively. It does not only help in forecasting and analysis it also helps in connecting the two different languages. Two best features of this programming language is that it does not have any compilation step as compared to the other programming language in which compilation is done before the program is being executed and other one is the reuse of the code, it consist of modules and packages due to which we can use the previously written code anywhere in between the program whenever is required. There are multiple languages for example R, Java, SQL, MATLAB available in market which can be used to analyse and evaluate the data, but due to some outstanding features python is the most famous language used in the field of data science.

Python is mostly used and easy among all other programming languages.

1.2 Operators, Conditional Statements …..

OPERATORS - Operators are the symbols in python that are used to perform Arithmetic or logical operations. Following are the different types of operators in python.

Arithmetic operators - Arithmetic operators carry out mathematical operations and they are mostly used with the numeric values.

Arithmetic operators		
Operator	Name	Example
+	Addition	A+B
-	Subtraction	A-B
*	Multiplication	A*B
/	Division	A/B
%	Modulus	A%B
**	Exponentiation	A**B
//	Quotient	A//B

Fig. 1.2.1: Arithmetic

operators A and B are the numeric value

Assignment operators - As the name decides this operators are used for assigning the values to the variables.

ASSIGNMENT OPERATORS		
Operator	Example	may also be written
=	a = 6	a = 6
+=	a += 3	a = a + 3
-=	a -= 4	a = a - 4
*=	a *= 5	a = a * 5
/=	a /= 6	a = a / 6
%=	a %= 7	a = a % 7
//=	a //= 8	a = a // 8
**=	a **= 9	a = a ** 9
&=	a &= 1	a = a & 1

Fig. 1.2.2: Assignment Operators

Here a is any value and number of operations are performed on this value.

Logical operators - These operators are used to join conditional statements

Logical Operators		
Operator	Description	Example
and	if both statements are true it returns true	x <5 **and** x <10 x
or	if any of the two statement is true it returns true	<4 **or** x <8
not	if the result is true it reverses the result and gives false	**not** (x <4 **and** x <8)

Fig. 1.2.3: Logical Operators

Here a is any value provided by us and on which multiple operations can be performed.

Comparison operators - These operators are used to compare two different values.

Comparison operators		
Operator	Name	Example
==	Equal	a == b
!=	Not equal	a!=b
>	Greater than	a >b
<	less than	a <b
>=	Greater than equal to	a>= b
<=	less than equal to	a <=b

Fig. 1.2.4: Comparison operators

Here a and b are two different values and these values are compared.

Membership operators - These operators are used to check membership of a particular value. It is used to check whether a specific value is present in the object or not.

Membership operators		
Operator	Description	Example
in	it returns a True if the value is present inside the object	a **in** b
not in	it returns a True if the value is not present inside the object	a **not in** b

Fig. 1.2.5: Membership operators

Condition statements

If else statements

"The most common type of statement is the if statement. if statement consist of a block which is called as clause", it is the block after if statement, it executed the statement if the condition is true. The statement is omitted if the condition is False. then the statement in the else part is printed

If statement consist of following -

- **If keyword itself** • **Condition which may be True or False** • **Colon**
- **If clause or a block of code** Below is the figure shows how If and else statements are used with description inside it.

```
x = 32


if x<40:
    print('x is less than 40')

else:
    print('x is greater than 40')

#In the above program we have if
#first line we have assigned valu
```

Figure 1.2.6 : if else statement

elif statements

In this statement only one statement is executed, There are many cases in which there is only one possibility to execute. "The elif statement is an else if statement that always follows an if or another elif statement"[8]. The elif statement provides another condition that is checked only if any of the previous conditions were False. In code, an elif statement always consists of the following:. The only difference between if else and elif statement is that in elif statement we have the condition where as in else statement we do not have any condition.

elIf statement consist of following -

- **elIf keyword itself**
- **Condition which may be True or False**
- **Colon**
- **elIf clause or a block of code**

Below is the figure shows how elIf statement is used with description inside it.

```
var = 't'

if var == 'a':
    print('this is the vowel a')
elif var == 'e':
    print('this is the vowel e')
elif var == 'i':
    print('this is the vowel i')
elif var == 'o':
    print('this is the vowel o')
elif var == 'u':
    print('this is the vowel u')
else:
    print('The value in variable var is constant')
```

Figure 1.2.7: elif example

1.3 Understanding Standard Libraries Pandas, Numpy…..

Libraries in Python

Python library is vast. There are built in functions in the library which are written in C lan- guage. This library provide access to system functionality such as file input output and that is not accessible to Python programmers. This modules and library provide solution to the many problems in programming.

Following are some Python libraries.

Matplotlib

Pandas

Numpy

Matplotlib

”Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy”[11]. MATLAB provides an application that is used in graphical user interface tool kits. Another such library is pylab which is almost same as MATLAB.

It is a library for 2D graphics, it finds its application in web application servers, graphical user interface toolkit and shell. Below is the example of a basic plot in python.

```
: import matplotlib.pyplot as plt
#we have imported matplotlib library first
#we have imported the pyplot module from matplotlib library
#we have give name 'plt' instead of using whole function name
plt.plot([1,2,3],[1,3,4])
#we used plot function to plot a graph
#we have take simple list in plot function
plt.xlabel('x label') #This function is used to name x axis
plt.ylabel('y label') #This function is used to name y axis
plt.title('basic plot') #This function used for title of grapho
plt.show() #This function show the graph
```

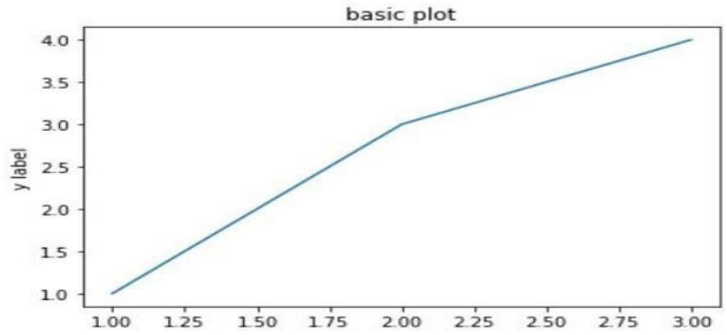


Figure 1.3.1: Matplotlib basic example

Pandas

Pandas is also a library or a data analysis tool in python which is written in python program- ming language. It is mostly used for data analysis and data manipulation. It is also used for data structures and time series.

We can see the application of python in many fields such as - Economics, Recommendation Sys- tems - Spotify, Netflix and Amazon, Stock Prediction, Neuro science, Statistics, Advertising, Analytics, Natural Language Processing. Data can be analyzed in pandas in two ways -

Data frames - In this data is two dimensional and consist of multiple series. Data is always represented in rectangular table.

Series - In this data is one dimensional and consist of single list with index.

```
#SERIES
import pandas as pd
#We are first importing the librabry
#and keeping its name as 'pd' for our convenience
odd_numbers = pd.Series([3,9,13,15])
#we have imported series array from pandas
odd_numbers
```

```
0     3
1     9
2    13
3    15
dtype: int64
```

```
#DATAFRAME - IT HAS TWO OR MORE ARRAYS IN IT
info_stu = {'students':['john','mike','harry','robert'],
     'age':[28,26,29,25],
     'country':['spain','rome','holand','russia']}
#here we have defind our 3 differnet arrays

#then we have imported 'DataFrame' from pandas

info =pd.DataFrame(info_stu)

info

#0,1,2,3 are the index number of different rows
```

	students	age	country
0	john	28	spain

Figure 1.3.2: series and data frame in pandas

NumPy

”NumPy is a library for the Python programming language, adding support for large, multi- dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays”. The previous similar programming of NumPy is Numeric, and this language was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. [12] It is an opensource library and free of cost.

```
import numpy as np  #we have imported numpy library

data = np.arange(60)
#arange is the range of the array function
#we called arange function from numpy libraby
data.shape = (10,6)
#there will be 10 rows and 6 columns in array
#now we have given the number of rows and columns
print(data) #print the data
print(len(data)) #gives length of array
print(data.ndim) #gives dimension of array
```

```
[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]]
```

Figure 1.3.3: NumPy basic example

CHAPTER 4: APPROACH USED (REQUIRED TOOLS)

- **Decision Tree:** A decision tree is a type of supervised machine learning used to categorize or make predictions based on how a previous set of questions were answered.
- **KNN Algorithm:** K-NN algorithm stores all the available data and classifies a new data point based on the similarity.
- **Logistic Regression:** Logistic regression is an example of supervised learning. It is used to calculate or predict the probability of a binary (yes/no) event occurring.

- **REQUIRED TOOLS:**

For application development, the following Software Requirements are:
Operating System: Windows 11

Language: python

Tools: JUPYTER notebook or COLAB, Microsoft Excel (Optional). Technologies used: python.

CHAPTER 5: RESULTS

Column Name	Column Description
id	Unique ID for the customer
Gender	Gender of the customer
Age	Age of the customer
Driving_License	0 - Customer does not have DL, 1 - Customer already has DL
Region_Code	Unique code for the region of the customer
Previously_Insured	1 - Customer already has Vehicle Insurance, 0 - Customer doesn't have Vehicle Insurance
Vehicle_Age	Age of the Vehicle
Vehicle_Damage	1 - Customer got his/her vehicle damaged in the past. 0 - Customer didn't get his/her vehicle damaged in the past.
Annual_Premium	The amount customer needs to pay as premium in the year
Policy_Sales_Channel	Anonymized Code for the channel of outreaching to the customer ie. Different Agents, Over Mail, Over Phone, In Person, etc.
Vintage	Number of Days, Customer has been associated with the company
Response	1 - Customer is interested, 0 - Customer is not interested

This Notebook will cover -

1. Exploratory Data Analysis
2. Data Modelling and Evaluation

Import Libraries

[24]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from imblearn.over_sampling import RandomOverSampler
sns.set(style='whitegrid')
```

Import Dataset

[31]:
```
train=pd.read_csv('../input/health-insurance-cross-sell-prediction/train.csv')
test=pd.read_csv('../input/health-insurance-cross-sell-prediction/test.csv')
```

[4]:
```
test.head()
```

[4]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	381110	Male	25	1	11.0	1	< 1 Year	No	35786.0	152.0	53
1	381111	Male	40	1	28.0	0	1-2 Year	Yes	33762.0	7.0	111
2	381112	Male	47	1	28.0	0	1-2 Year	Yes	40050.0	124.0	199
3	381113	Male	24	1	27.0	1	< 1 Year	Yes	37356.0	152.0	187
4	381114	Male	27	1	28.0	1	< 1 Year	No	59097.0	152.0	297

[3]:
```
train.head()
```

[3]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0

[4]:
```
train.shape
```

[4]:
```
(381109, 12)
```

Check for missing values

[5]:
```
train.isnull().sum()
```

[5]:
```
id                    0
Gender                0
Age                   0
Driving_License       0
Region_Code           0
Previously_Insured    0
Vehicle_Age           0
Vehicle_Damage        0
Annual_Premium        0
Policy_Sales_Channel  0
Vintage               0
Response              0
dtype: int64
```

- No missing data

Exploratory Data Analysis

[6]:
```
numerical_columns=['Age', 'Region_Code','Annual_Premium','Vintage']
categorical_columns=['Gender','Driving_License','Previously_Insured','Vehicle_Age','Vehicle_Damage','Response']
```

[7]:
```
train[numerical_columns].describe()
```

[7]:

	Age	Region_Code	Annual_Premium	Vintage
count	381109.000000	381109.000000	381109.000000	381109.000000
mean	38.822584	26.388807	30564.389581	154.347397
std	15.511611	13.229888	17213.155057	83.671304
min	20.000000	0.000000	2630.000000	10.000000
25%	25.000000	15.000000	24405.000000	82.000000
50%	36.000000	28.000000	31669.000000	154.000000
75%	49.000000	35.000000	39400.000000	227.000000
max	85.000000	52.000000	540165.000000	299.000000

[8]:
```
train
```

[8]:

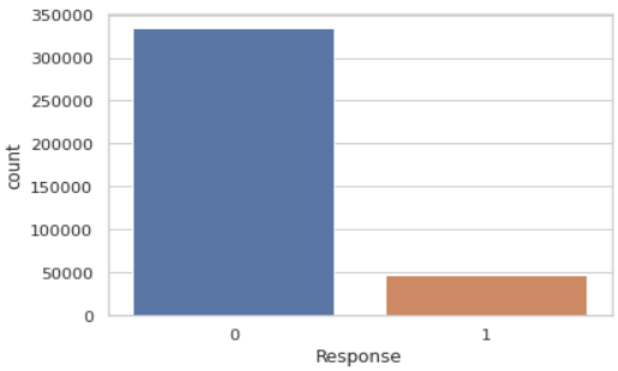
	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0
...	...	...	...	...	...	...	...	...	...	...	...	...
81104	381105	Male	74	1	26.0	1	1-2 Year	No	30170.0	26.0	88	0
81105	381106	Male	30	1	37.0	1	< 1 Year	No	40016.0	152.0	131	0
81106	381107	Male	21	1	30.0	1	< 1 Year	No	35118.0	160.0	161	0
81107	381108	Female	68	1	14.0	0	> 2 Years	Yes	44617.0	124.0	74	0
81108	381109	Male	46	1	29.0	0	1-2 Year	No	41777.0	26.0	237	0

1109 rows × 12 columns

Target Variable (Response)

[9]:
```
sns.countplot(train.Response)
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5f0b9d490>
```



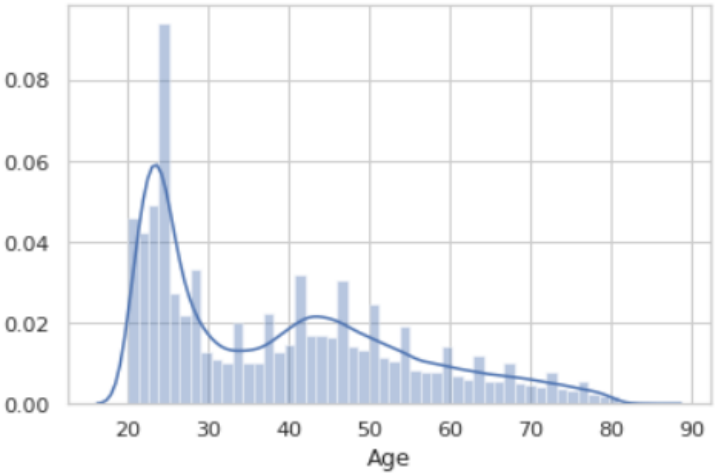
[10]:
```
train.Response.value_counts()
```

[10]:
```
0    334399
1     46710
Name: Response, dtype: int64
```

Age Distribution of Customers

[11]:
```
sns.distplot(train.Age)
```

[11]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5f09ebd50>
```



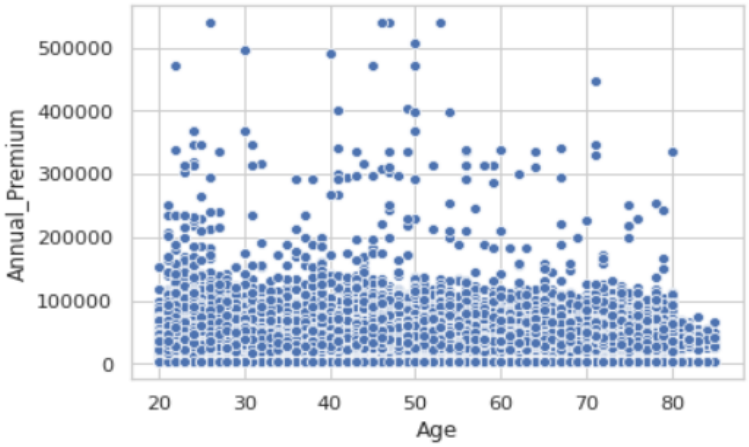
[12]:
```
sns.boxplot(y = 'Age', data = train,palette='Accent')
```

[12]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d9047690>
```

Age Vs Annual premium

[13]:
```
sns.scatterplot(x=train['Age'],y=train['Annual_Premium'])
```

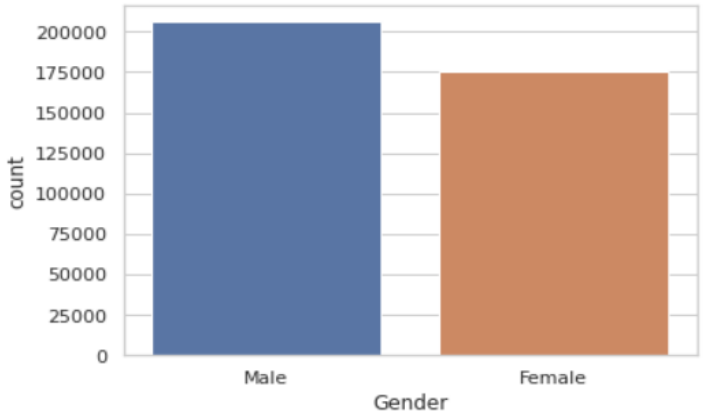
[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d9028590>
```



Gender and Response

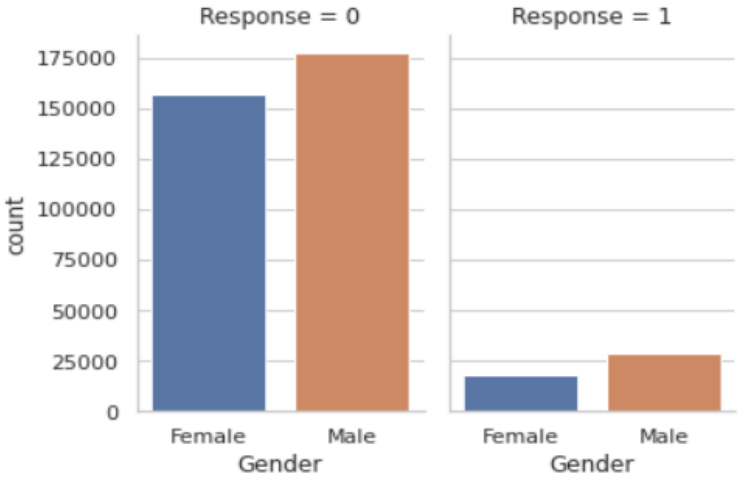
[14]:
```
sns.countplot(train.Gender)
```

[14]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d8f810d0>
```



[15]:
```
df=train.groupby(['Gender','Response'])['id'].count().to_frame().rename(columns={'id':'count'}).reset_index()
```

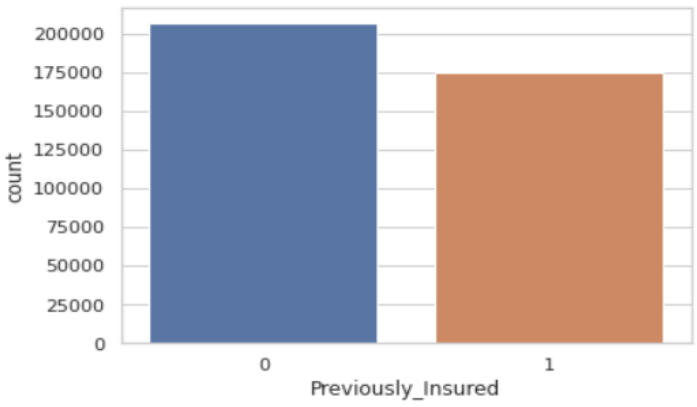
[16]:
```
g = sns.catplot(x="Gender", y="count",col="Response",
                data=df, kind="bar",
                height=4, aspect=.7);
```



Customers having Vehicle insurance already

20]:
```
sns.countplot(train.Previously_Insured)
```

20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d8e6cb10>
```



Driving license by Gender

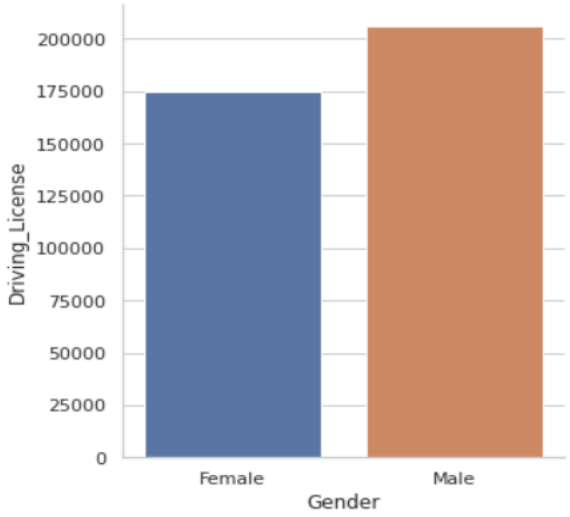
[17]:
```
df=train.groupby(['Gender'])['Driving_License'].count().to_frame().reset_index()
```

[18]:
```
df
```

[18]:

	Gender	Driving_License
0	Female	175020
1	Male	206089

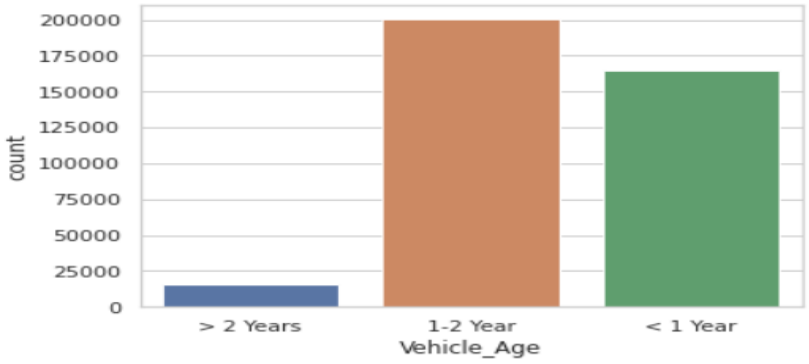
[19]:
```
sns.catplot(x="Gender", y="Driving_License",
                data=df, kind="bar");
```



Vehicle Age ¶

:
```
sns.countplot(train.Vehicle_Age)
```

:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d8defcd0>
```



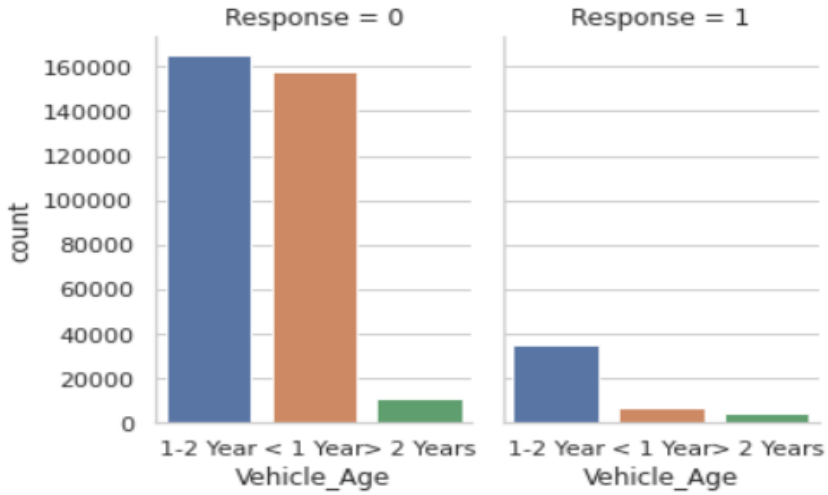
Response and Vehicle age

[22]:
```
df=train.groupby(['Vehicle_Age','Response'])['id'].count().to_frame().rename(columns={'id':'count'}).reset_index()
df
```

[22]:

	Vehicle_Age	Response	count
0	1-2 Year	0	165510
1	1-2 Year	1	34806
2	< 1 Year	0	157584
3	< 1 Year	1	7202
4	> 2 Years	0	11305
5	> 2 Years	1	4702

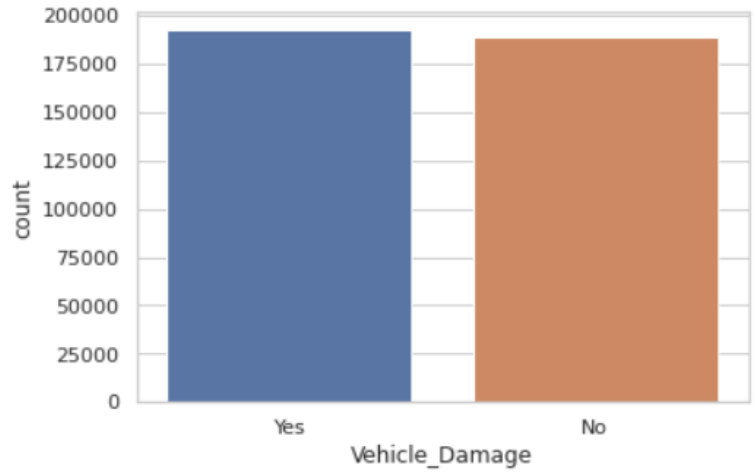
[23]:
```
g = sns.catplot(x="Vehicle_Age", y="count",col="Response",
                data=df, kind="bar",
                height=4, aspect=.7);
```



Customers having damaged vehicle

24]:
```
sns.countplot(train.Vehicle_Damage)
```

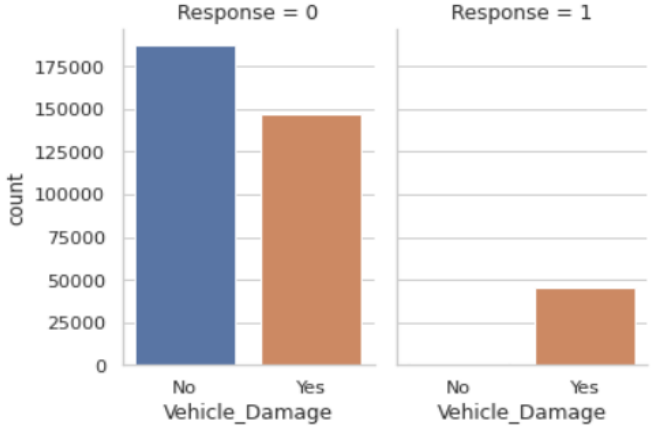
24]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d8c5b650>
```



Damage Vehicle and Response

:
```
df=train.groupby(['Vehicle_Damage','Response'])['id'].count().to_frame().rename(columns={'id':'count'}).reset_index()
```

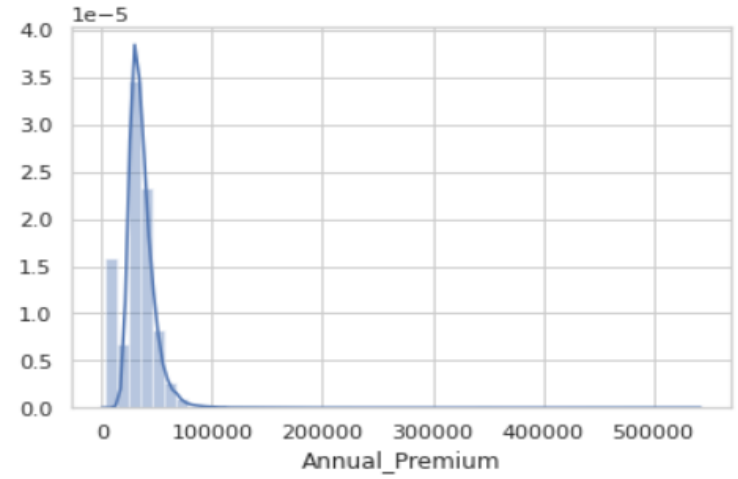
:
```
g = sns.catplot(x="Vehicle_Damage", y="count",col="Response",
                data=df, kind="bar",
                height=4, aspect=.7);
```



Annual Premium Distribution

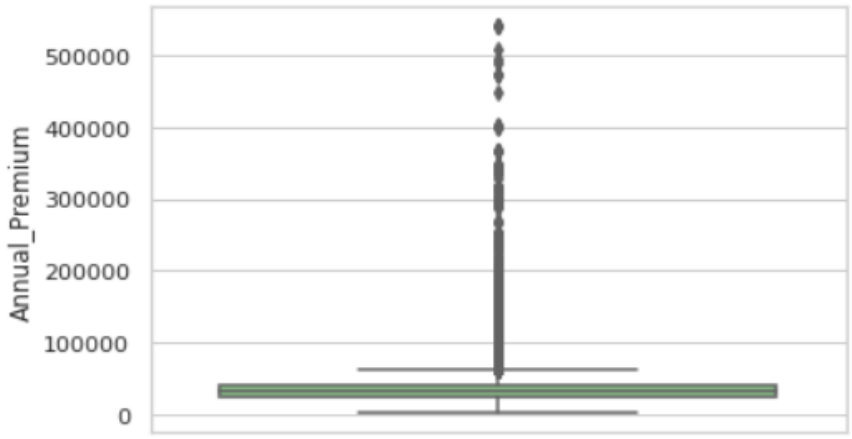
[27]:
```
sns.distplot(train.Annual_Premium)
```

[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d8bbac10>
```



[28]:
```
sns.boxplot(y = 'Annual_Premium', data = train,palette='Accent')
```

[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d8a0ca10>
```

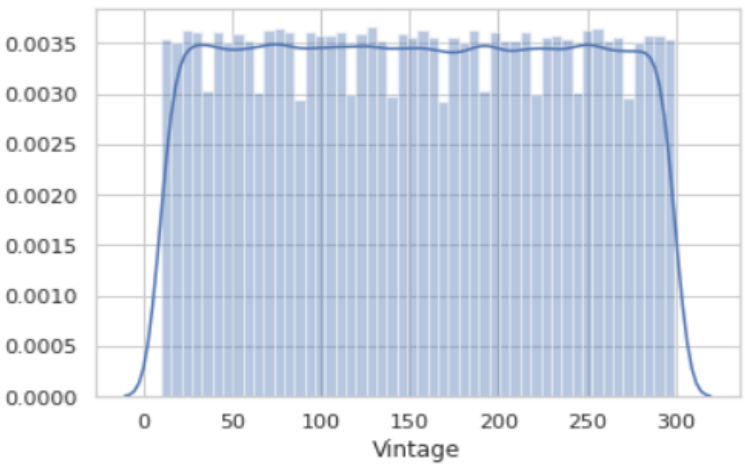


Vintage

Number of Days, Customer has been associated with the company

[9]:
```
sns.distplot(train.Vintage)
```

[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb5d89747d0>
```



Data Preprocessing

42]:
```
num_feat = ['Age','Vintage']
cat_feat = ['Gender', 'Driving_License', 'Previously_Insured', 'Vehicle_Age_lt_1_Year','Vehicle_Age_gt_2_Years','Vehicle_Damage_Yes','Region_Code','Polic
```

32]:
```
train['Gender'] = train['Gender'].map( {'Female': 0, 'Male': 1} ).astype(int)
```

33]:
```
train=pd.get_dummies(train,drop_first=True)
```

34]:
```
train=train.rename(columns={"Vehicle_Age_< 1 Year": "Vehicle_Age_lt_1_Year", "Vehicle_Age_> 2 Years": "Vehicle_Age_gt_2_Years"})
train['Vehicle_Age_lt_1_Year']=train['Vehicle_Age_lt_1_Year'].astype('int')
train['Vehicle_Age_gt_2_Years']=train['Vehicle_Age_gt_2_Years'].astype('int')
train['Vehicle_Damage_Yes']=train['Vehicle_Damage_Yes'].astype('int')
```

36]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, RobustScaler
ss = StandardScaler()
train[num_feat] = ss.fit_transform(train[num_feat])


mm = MinMaxScaler()
train[['Annual_Premium']] = mm.fit_transform(train[['Annual_Premium']])
```

38]:
```
train=train.drop('id',axis=1)
```

43]:
```
for column in cat_feat:
    train[column] = train[column].astype('str')
```

45]:
```
train
```

45]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage	Response	Vehicle_Age_lt_1_Year
0	1	0.333777	1	28.0	0	0.070366	26.0	0.748795	1	0
1	1	2.396751	1	3.0	0	0.057496	26.0	0.342443	0	0
2	1	0.527181	1	28.0	0	0.066347	26.0	-1.521998	1	0
3	1	-1.148985	1	11.0	1	0.048348	152.0	0.581474	0	1
4	0	-0.633242	1	41.0	1	0.046259	152.0	-1.378580	0	1
...	...	...	...	...	...	...	...	...	...	...
381104	1	2.267815	1	26.0	1	0.051234	26.0	-0.792954	0	0
381105	1	-0.568774	1	37.0	1	0.069551	152.0	-0.279037	0	1
381106	1	-1.148985	1	30.0	1	0.060439	160.0	0.079509	0	1
381107	0	1.881007	1	14.0	0	0.078110	124.0	-0.960275	0	0
381108	1	0.462713	1	29.0	0	0.072827	26.0	0.987826	0	0

[46]:
```
test['Gender'] = test['Gender'].map( {'Female': 0, 'Male': 1} ).astype(int)
test=pd.get_dummies(test,drop_first=True)
test=test.rename(columns={"Vehicle_Age_< 1 Year": "Vehicle_Age_lt_1_Year", "Vehicle_Age_> 2 Years": "Vehicle_Age_gt_2_Years"})
test['Vehicle_Age_lt_1_Year']=test['Vehicle_Age_lt_1_Year'].astype('int')
test['Vehicle_Age_gt_2_Years']=test['Vehicle_Age_gt_2_Years'].astype('int')
test['Vehicle_Damage_Yes']=test['Vehicle_Damage_Yes'].astype('int')
```

[47]:
```
from sklearn.preprocessing import LabelEncoder, StandardScaler, MinMaxScaler, RobustScaler
ss = StandardScaler()
test[num_feat] = ss.fit_transform(test[num_feat])



mm = MinMaxScaler()
test[['Annual_Premium']] = mm.fit_transform(test[['Annual_Premium']])
```

[74]:
```
for column in cat_feat:
    test[column] = test[column].astype('str')
```

[65]:
```
from sklearn.model_selection import train_test_split

train_target=train['Response']
train=train.drop(['Response'], axis = 1)
x_train,x_test,y_train,y_test = train_test_split(train,train_target, random_state = 0)
```

[51]:
```
id=test.id
```

[52]:
```
test=test.drop('id',axis=1)
```

:
```
x_train.columns
```

:
```
Index(['Gender', 'Age', 'Driving_License', 'Region_Code', 'Previously_Insured',
       'Annual_Premium', 'Policy_Sales_Channel', 'Vintage',
       'Vehicle_Age_lt_1_Year', 'Vehicle_Age_gt_2_Years',
       'Vehicle_Damage_Yes'],
      dtype='object')
```

Data Modelling and Evaluation

:
```
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from catboost import CatBoostClassifier
from scipy.stats import randint
import pickle
import xgboost as xgb
import lightgbm as lgb
from sklearn.metrics import accuracy_score
# import packages for hyperparameters tuning
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
from sklearn.model_selection import train_test_split, RandomizedSearchCV, StratifiedKFold, KFold, GridSearchCV
from sklearn.metrics import f1_score, roc_auc_score,accuracy_score,confusion_matrix, precision_recall_curve, auc, roc_curve, recall_score, classification
```

Random Forest Classifier

```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
x_train.dtypes
```

```
Gender                    int64
Age                       int64
Driving_License           int64
Region_Code             float64
Previously_Insured        int64
Annual_Premium          float64
Policy_Sales_Channel    float64
Vintage                   int64
Vehicle_Age_lt_1_Year     uint8
Vehicle_Age_gt_2_Years    uint8
Vehicle_Damage_Yes        uint8
dtype: object
```

```
random_search = {'criterion': ['entropy', 'gini'],
                 'max_depth': [2,3,4,5,6,7,10],
                 'min_samples_leaf': [4, 6, 8],
                 'min_samples_split': [5, 7,10],
                 'n_estimators': [300]}

clf = RandomForestClassifier()
model = RandomizedSearchCV(estimator = clf, param_distributions = random_search, n_iter = 10,
                               cv = 4, verbose= 1, random_state= 101, n_jobs = -1)
model.fit(x_train,y_train)
```

```
Fitting 4 folds for each of 10 candidates, totalling 40 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=-1)]: Done  40 out of  40 | elapsed: 20.7min finished
```

```
RandomizedSearchCV(cv=4, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'criterion': ['entropy', 'gini'],
                                        'max_depth': [2, 3, 4, 5, 6, 7, 10],
                                        'min_samples_leaf': [4, 6, 8],
                                        'min_samples_split': [5, 7, 10],
                                        'n_estimators': [300]},
                   random_state=101, verbose=1)
```

Save model

```
filename = 'rf_model.sav'
pickle.dump(model, open(filename, 'wb'))
```

```
filename = 'rf_model.sav'
```

```
rf_load = pickle.load(open(filename, 'rb'))
```

Evaluate Model

```
y_pred=model.predict(x_test)
```

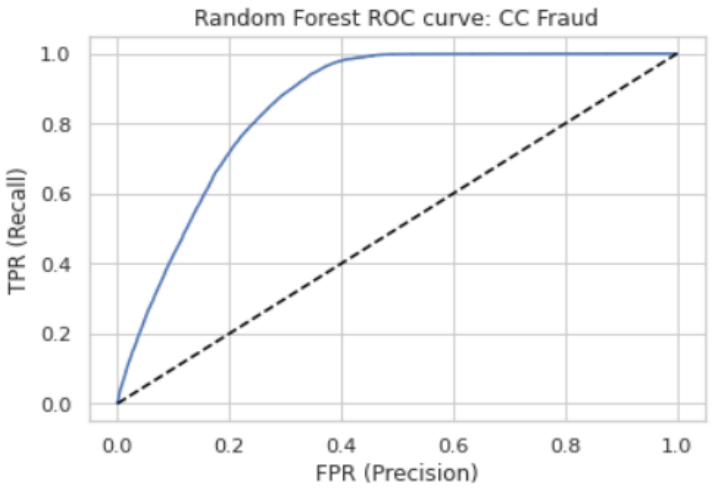
Classification Report ¶

```
print (classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.88      1.00      0.93     83603
           1       0.00      0.00      0.00     11675

    accuracy                           0.88     95278
   macro avg       0.44      0.50      0.47     95278
weighted avg       0.77      0.88      0.82     95278
```

ROC Curve & AUC of Random forest classifier

```
y_score = model.predict_proba(x_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)

title('Random Forest ROC curve: CC Fraud')
xlabel('FPR (Precision)')
ylabel('TPR (Recall)')

plot(fpr,tpr)
plot((0,1), ls='dashed',color='black')
plt.show()
print ('Area under curve (AUC): ', auc(fpr,tpr))
```




```
Area under curve (AUC):  0.8551439982187663
```

```
roc_auc_score(y_test, y_score)
```

```
0.8551439982187663
```

XGBoost Classifier

```
for column in cat_feat:
    x_train[column] = x_train[column].astype('int')
    x_test[column] = x_test[column].astype('int')
```

```
space={ 'max_depth': hp.quniform("max_depth", 3,18,1),
        'gamma': hp.uniform ('gamma', 1,9),
        'reg_alpha' : hp.quniform('reg_alpha', 40,180,1),
        'reg_lambda' : hp.uniform('reg_lambda', 0,1),
        'colsample_bytree' : hp.uniform('colsample_bytree', 0.5,1),
        'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1),
        'n_estimators': 300,
        'seed': 0
      }
```

```
def objective(space):
    clf=xgb.XGBClassifier(
                    n_estimators =space['n_estimators'], max_depth = int(space['max_depth']), gamma = space['gamma'],
                    reg_alpha = int(space['reg_alpha']),min_child_weight=int(space['min_child_weight']),
                    colsample_bytree=int(space['colsample_bytree']))

    evaluation = [( x_train, y_train), ( x_test, y_test)]

    clf.fit(x_train, y_train,
            eval_set=evaluation, eval_metric="auc",
            early_stopping_rounds=10,verbose=False)


    pred = clf.predict(x_test)
    y_score = model.predict_proba(x_test)[:,1]
    accuracy = accuracy_score(y_test, pred>0.5)
    Roc_Auc_Score = roc_auc_score(y_test, y_score)
    print ("ROC-AUC Score:",Roc_Auc_Score)
    print ("SCORE:", accuracy)
    return {'loss': -Roc_Auc_Score, 'status': STATUS_OK }
```

```
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
```

```
trials = Trials()
```

```
best_hyperparams = fmin(fn = objective,
                        space = space,
                        algo = tpe.suggest,
                        max_evals = 100,
                        trials = trials)
```

```
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774428514452445
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774638426499297
ROC-AUC Score:
0.8551439982187663
SCORE:
0.877453347047587
ROC-AUC Score:
0.8551439982187663
SCORE:
0.877453347047587
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774638426499297
ROC-AUC Score:
0.8551439982187663
SCORE:
0.877453347047587
ROC-AUC Score:
0.8551439982187663
```

```
0.8551439982187663
SCORE:
0.877453347047587
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774638426499297
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774638426499297
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774638426499297
ROC-AUC Score:
0.8551439982187663
SCORE:
0.877453347047587
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774638426499297
ROC-AUC Score:
0.8551439982187663
SCORE:
0.877453347047587
ROC-AUC Score:
0.8551439982187663
SCORE:
0.877453347047587
ROC-AUC Score:
0.8551439982187663
SCORE:
0.8774638426499297
ROC-AUC Score:
0.8551439982187663
```

```
print("The best hyperparameters are : ","\n")
print(best_hyperparams)
```

```
The best hyperparameters are :

{'colsample_bytree': 0.9315329293311101, 'gamma': 1.8461594224258304, 'max_depth': 5.0, 'min_child_weight': 4.0, 'reg_alpha': 126.0, 'reg_lambda': 0.4915
0384106319067}
```

```
xgb_model=xgb.XGBClassifier(n_estimators = space['n_estimators'], max_depth = best, gamma = 4.0388607178326605, reg_lambda = 0.26955899476862166,
                            reg_alpha = 66.0, min_child_weight=4.0,colsample_bytree = 0.8844758548525424 )
```

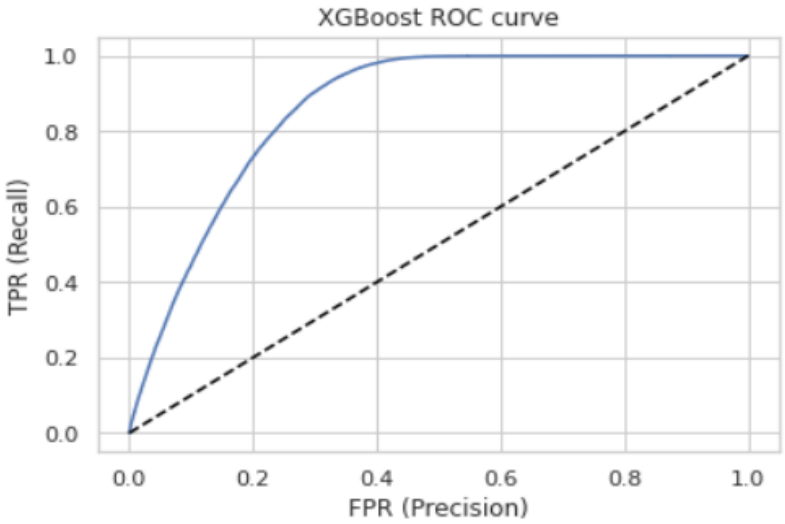
```
xgb_model.fit(x_train,y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=0.8844758548525424,
              gamma=4.0388607178326605, gpu_id=-1, importance_type='gain',
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=7, min_child_weight=4.0, missing=nan,
              monotone_constraints='()', n_estimators=300, n_jobs=0,
              num_parallel_tree=1, random_state=0, reg_alpha=66.0,
              reg_lambda=0.26955899476862166, scale_pos_weight=1, subsample=1,
              tree_method='exact', validate_parameters=1, verbosity=None)
```

```
filename = 'xgboost_model.sav'
pickle.dump(xgb_model, open(filename, 'wb'))
```

ROC Curve & AUC of XG boost classifier

```
y_score = xgb_model.predict_proba(x_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)

title('XGBoost ROC curve')
xlabel('FPR (Precision)')
ylabel('TPR (Recall)')

plot(fpr,tpr)
plot((0,1), ls='dashed',color='black')
plt.show()
print ('Area under curve (AUC): ', auc(fpr,tpr))
```




```
Area under curve (AUC):  0.8610552123033075
```

```
random_state=42
n_iter=50
num_folds=2
kf = KFold(n_splits=num_folds, random_state=random_state,shuffle=True)
```

```
def gb_mse_cv(params, random_state=random_state, cv=kf, X=x_train, y=y_train):
    # the function gets a set of variable parameters in "param"
    params = {'n_estimators': int(params['n_estimators']),
              'max_depth': int(params['max_depth']),
              'learning_rate': params['learning_rate'],
              'gamma': params['gamma'],
              'reg_alpha' : params['reg_alpha'],
              'reg_lambda' : params['reg_lambda'],
              'colsample_bytree' : params['colsample_bytree'],
              'min_child_weight' : params['min_child_weight']
             }



    # we use this params to create a new LGBM Regressor
    model = lgb.LGBMClassifier(random_state=42, **params)

    # and then conduct the cross validation with the same folds as before
    score = -cross_val_score(model, X, y, cv=cv, scoring="roc_auc", n_jobs=-1).mean()

    return score
```

```
%%time

# possible values of parameters
space={'n_estimators': hp.quniform('n_estimators', 100, 200, 1),
       'max_depth' : hp.quniform('max_depth', 2, 8, 1),
       'learning_rate': hp.loguniform("learning_rate",-4,-1),
        'gamma': hp.quniform('gamma',0.1,0.5,0.1),
        'reg_alpha' : hp.quniform('reg_alpha',1.1,1.5,0.1),
        'reg_lambda' : hp.uniform('reg_lambda',1.1,1.5),
        'colsample_bytree' : hp.uniform('colsample_bytree', 0.1,0.5),
        'min_child_weight' : hp.quniform('min_child_weight', 0, 10, 1),
      }

# trials will contain logging information
trials = Trials()

best=fmin(fn=gb_mse_cv, # function to optimize
          space=space,
          algo=tpe.suggest, # optimization algorithm, hyperotp will select its parameters automatically
          max_evals=n_iter, # maximum number of iterations
          trials=trials, # logging
          rstate=np.random.RandomState(random_state) # fixing random state for the reproducibility
         )

# computing the score on the test set
model = lgb.LGBMClassifier(random_state=random_state, n_estimators=int(best['n_estimators']),
                           max_depth=int(best['max_depth']),learning_rate=best['learning_rate'],gamma=best['gamma'],
                           reg_alpha=best['reg_alpha'],reg_lambda=best['reg_lambda'],colsample_bytree=best['colsample_bytree'],
                           min_child_weight=best['min_child_weight'])
model.fit(x_train,y_train)

preds = [pred[1] for pred in model.predict_proba(x_test)]
score = roc_auc_score(y_test, preds, average = 'weighted')
```

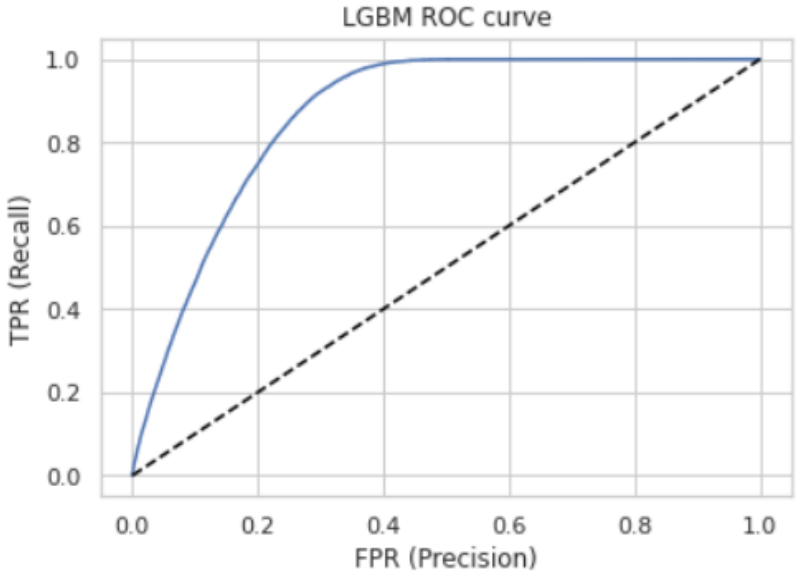
```
best
```

```
{'colsample_bytree': 0.48944638171094995,
 'gamma': 0.4,
 'learning_rate': 0.3597338813116028,
 'max_depth': 7.0,
 'min_child_weight': 2.0,
 'n_estimators': 192.0,
 'reg_alpha': 1.2000000000000002,
 'reg_lambda': 1.1374081667346767}
```

```
print("auc-roc score on Test data",score)
```

```
auc-roc score on Test data 0.8680941572226264
```

```
y_score = model.predict_proba(x_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)

title('LGBM ROC curve')
xlabel('FPR (Precision)')
ylabel('TPR (Recall)')

plot(fpr,tpr)
plot((0,1), ls='dashed',color='black')
plt.show()
print ('Area under curve (AUC): ', auc(fpr,tpr))
```




```
Area under curve (AUC):  0.8680941572226264
```

CatBoost

```
]: #X_cat_train, X_cat_test, y_cat_train, y_cat_test = train_test_split(X_cat, Y_cat, test_size = 0.22, random_state = 22, stratify = Y_cat, shuffle = True)

cat_model = CatBoostClassifier()
cat_model = cat_model.fit(x_train, y_train, cat_features = cat_feat, eval_set = (x_test, y_test), early_stopping_rounds = 10, verbose = 100)

predictions = [pred[1] for pred in cat_model.predict_proba(x_test)]
print('Validation ROC AUC Score:', roc_auc_score(y_test, predictions, average = 'weighted'))
```

```
Learning rate set to 0.128106
0:      learn: 0.5320867        test: 0.5323419 best: 0.5323419 (0)     total: 412ms    remaining: 6m 52s
100:    learn: 0.2630644        test: 0.2639867 best: 0.2639855 (98)    total: 27.8s    remaining: 4m 7s
Stopped by overfitting detector  (10 iterations wait)

bestTest = 0.2639346435
bestIteration = 114

Shrink model to first 115 iterations.
Validation ROC AUC Score: 0.8590208208720521
```

Evaluating on Test data

Catboost

```
: test
```

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage	Vehicle_Age_lt_1_Year
0	1	-0.890089	1	11.0	1	0.070633	152.0	-1.211054	1
1	1	0.079795	1	28.0	0	0.066321	7.0	-0.517782	0
2	1	0.532408	1	28.0	0	0.079717	124.0	0.534079	0
3	1	-0.954748	1	27.0	1	0.073978	152.0	0.390643	1
4	1	-0.760771	1	28.0	1	0.120293	152.0	1.705469	1
...	...	...	...	...	...	...	...	...	...
127032	0	-0.825430	1	37.0	1	0.060154	152.0	-1.175195	1
127033	0	-0.049523	1	28.0	0	0.055538	122.0	0.127678	0
127034	1	-1.148725	1	46.0	1	0.057885	152.0	-0.960042	1
127035	1	2.084224	1	28.0	1	0.128341	26.0	1.322974	0
127036	1	0.144454	1	29.0	1	0.053891	124.0	0.916574	0

```
Preds = [pred[1] for pred in cat_model.predict_proba(test)]
```

```
submission = pd.DataFrame(data = {'id': id, 'Response': Preds})
submission.to_csv('vehicle_insurance_catboost.csv', index = False)
submission.head()
```

	id	Response
0	381110	0.000269
1	381111	0.303041
2	381112	0.292329
3	381113	0.007213
4	381114	0.000188

LGBM

```
id=test.id
#test.drop(['id'],axis=1,inplace=True)
```

```
test['Gender'] = test['Gender'].map( {'Female': 0, 'Male': 1} ).astype(int)
test=pd.get_dummies(test,drop_first=True)
test=test.rename(columns={"Vehicle_Age_< 1 Year": "Vehicle_Age_lt_1_Year", "Vehicle_Age_> 2 Years": "Vehicle_Age_gt_2_Years"})
test=test.rename(columns={"Vehicle_Age_< 1 Year": "Vehicle_Age_lt_1_Year", "Vehicle_Age_> 2 Years": "Vehicle_Age_gt_2_Years"})
```

```
Preds = [pred[1] for pred in model.predict_proba(test)]
```

```
submission = pd.DataFrame(data = {'id': id, 'Response': Preds})
submission.to_csv('vehicle_insurance.csv', index = False)
submission.head()
```

	id	Response
0	381110	0.001377
1	381111	0.799187
2	381112	0.768010
3	381113	0.063009
4	381114	0.004429

CHAPTER 6: SUMMARY & CONCLUSIONS

1. The highest amount of insurance base on categories is major accidents followed by minor accidents .

2. Building a model to predict whether a customer would be interested in Vehicle Insurance is extremely helpful for the company because it can then accordingly plan its communication strategy to reach out to those customers and optimise its business model and revenue.

3. Now, in order to predict, whether the customer would be interested in Vehicle insurance, you have information about demographics (gender, age, region code type), Vehicles (Vehicle Age, Damage), Policy (Premium, sourcing channel) etc.

REFERENCES & BIBLIOGRAPHY

www.kaggle.com
www.wikipedia.com
www.geeksforgeeks.com
www.javatpoint.com

GitHub Link for the Project File –

https://github.com/Gaurav-Moudgil/Project-File