Business Problem:

A health insurance company wants to improve its cross-sales strategy by identifying customers who are likely to purchase an additional vehicle insurance policy. The company has historical data on customer demographics, vehicle details, policy information, and past purchase behavior. However, not all customers respond positively to cross-selling efforts.

Project Objective:

The objective of this project is to develop a machine learning model that can accurately predict whether a vehicle insurance customer is likely to purchase a health insurance policy. By leveraging historical customer data, the model will help the insurance company identify high-potential customers, improve cross-selling efficiency, and optimize marketing strategies.Independant Features: id, Gender, Age, Driving_License, Region_Code, Previously_Insured, Vehicle_Age, Vehicle_Damage, Annual_Premium, Policy_Sales_Channel, Vintage Dependant Feature / Target Columns - categorical (Classification) Response

Tools and Technologies Used:

for this project, we utilized the following tools: * python : Main programming language * pandas : Data manipulation * Numpy : Numerical operations * Matplotlib and seaborn : For data visualization

In [1]:
```
pip install xgboost
```

```
Requirement already satisfied: xgboost in e:\software\jupyter notebook\lib\site-packages (2.1.4)
Requirement already satisfied: numpy in e:\software\jupyter notebook\lib\site-packages (from xgboost) (1.26.4)
Requirement already satisfied: scipy in e:\software\jupyter notebook\lib\site-packages (from xgboost) (1.13.1)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:
```
import numpy as np   # Used for numerical computations
import pandas as pd  # Used for data manipulation
import matplotlib.pyplot as plt   # A plotting library for visualizing data
import seaborn as sns      # providing high-level statistical graphics and visualization

from scipy.stats import zscore,shapiro,normaltest   # Used for statistical tests and probability distributions

from sklearn.model_selection import train_test_split,RandomizedSearchCV  # ML library that provides tools for model training, evaluation.

from statsmodels.stats.outliers_influence import variance_inflation_factor # Detects multicollinearity in datasets.

from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier,plot_tree
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC

from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

import pickle

import warnings as w
w.filterwarnings('ignore')
```

In [3]:
```
df=pd.read_csv(r"C:\Users\Akash\Desktop\P315_AkashNikam\ML\train.csv")
df.head()
```

Out[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	> 2 Years	Yes	40454	26	217	1
1	2	Male	76	1	3	0	1-2 Year	No	33536	26	183	0
2	3	Male	47	1	28	0	> 2 Years	Yes	38294	26	27	1
3	4	Male	21	1	11	1	< 1 Year	No	28619	152	203	0
4	5	Female	29	1	41	1	< 1 Year	No	27496	152	39	0

In [4]:
```
df.size    # returns the total number of elements in the DataFrame (rows * columns)
```

Out[4]:
```
119988
```

In [5]:
```
rows, columns = df.shape    # returns the dimensions of a DataFrame
print(" Total Rows:",rows)
print(" Total Columns:",columns)
```

```
Total Rows: 9999
Total Columns: 12
```

In [6]:
```
df.info()   # provides a summary of the DataFrame
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9999 entries, 0 to 9998
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   id                    9999 non-null   int64
 1   Gender                9999 non-null   object
 2   Age                   9999 non-null   int64
 3   Driving_License       9999 non-null   int64
 4   Region_Code           9999 non-null   int64
 5   Previously_Insured    9999 non-null   int64
 6   Vehicle_Age           9999 non-null   object
 7   Vehicle_Damage        9999 non-null   object
 8   Annual_Premium        9999 non-null   int64
 9   Policy_Sales_Channel  9999 non-null   int64
 10  Vintage               9999 non-null   int64
 11  Response              9999 non-null   int64
dtypes: int64(9), object(3)
memory usage: 937.5+ KB
```

In [7]:
```
df.describe()    # generates summary statistics for numerical columns
```

Out[7]:

	id	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage	Response
count	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000	9999.000000
mean	5000.000000	38.814281	0.997800	26.423542	0.443744	30610.826483	112.410441	155.562756	0.124712
std	2886.607005	15.514946	0.046857	13.152898	0.496850	16562.803144	53.976290	83.927687	0.330409
min	1.000000	20.000000	0.000000	0.000000	0.000000	2630.000000	1.000000	10.000000	0.000000
25%	2500.500000	25.000000	1.000000	15.000000	0.000000	24463.000000	30.000000	83.000000	0.000000
50%	5000.000000	36.000000	1.000000	28.000000	0.000000	31747.000000	136.000000	156.000000	0.000000
75%	7499.500000	49.000000	1.000000	35.000000	1.000000	39572.500000	152.000000	228.000000	0.000000
max	9999.000000	84.000000	1.000000	52.000000	1.000000	267698.000000	163.000000	299.000000	1.000000

In [8]:
```
df.isna()    # method checks for missing (NaN) values
```

Out[8]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Respons
0	False	False	False	False	False	False	False	False	False	False	False	Fal
1	False	False	False	False	False	False	False	False	False	False	False	Fal
2	False	False	False	False	False	False	False	False	False	False	False	Fal
3	False	False	False	False	False	False	False	False	False	False	False	Fal
4	False	False	False	False	False	False	False	False	False	False	False	Fal
...	...	...	...	...	...	...	...	...	...	...	...	
9994	False	False	False	False	False	False	False	False	False	False	False	Fal
9995	False	False	False	False	False	False	False	False	False	False	False	Fal
9996	False	False	False	False	False	False	False	False	False	False	False	Fal
9997	False	False	False	False	False	False	False	False	False	False	False	Fal
9998	False	False	False	False	False	False	False	False	False	False	False	Fal

9999 rows × 12 columns

In [9]:
```
df.isna().sum()    # returns the count of missing (NaN) values
```

Out[9]:
```
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
```

In [10]:
```
df.drop(["id"],axis=1,inplace=True)
df
```

Out[10]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	Male	44	1	28	0	> 2 Years	Yes	40454	26	217	1
1	Male	76	1	3	0	1-2 Year	No	33536	26	183	0
2	Male	47	1	28	0	> 2 Years	Yes	38294	26	27	1
3	Male	21	1	11	1	< 1 Year	No	28619	152	203	0
4	Female	29	1	41	1	< 1 Year	No	27496	152	39	0
...	...	...	...	...	...	...	...	...	...	...	...
9994	Female	35	1	28	0	1-2 Year	Yes	34853	154	21	0
9995	Female	24	1	37	1	< 1 Year	No	21243	152	21	0
9996	Male	28	1	35	1	< 1 Year	No	23589	160	173	0
9997	Female	27	1	15	0	< 1 Year	Yes	33340	152	207	0
9998	Female	28	1	14	1	< 1 Year	No	25426	152	247	0

9999 rows × 11 columns

In [11]:
```
df.columns
```

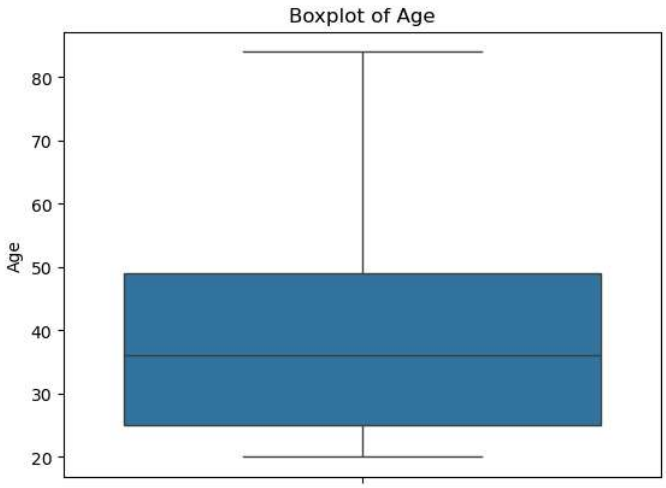
Out[11]:
```
Index(['Gender', 'Age', 'Driving_License', 'Region_Code', 'Previously_Insured',
       'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premium',
       'Policy_Sales_Channel', 'Vintage', 'Response'],
      dtype='object')
```

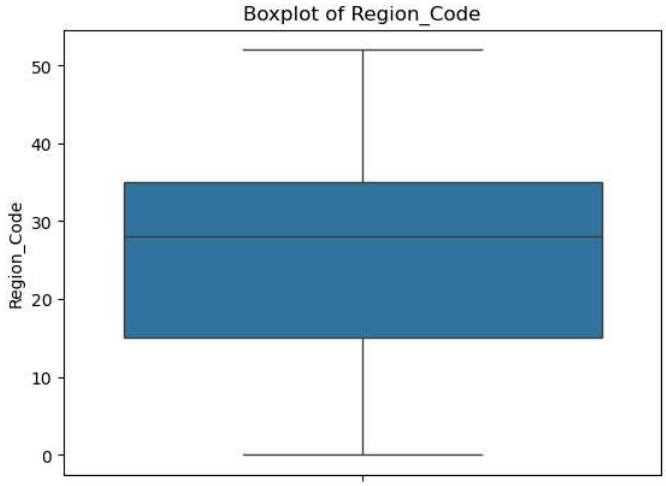
In [12]:
```
num_col=['Age','Region_Code','Annual_Premium','Policy_Sales_Channel', 'Vintage']
stat={}
for col in num_col:
    stat[col]={
        "Mean":df[col].mean(),          # Average value of the column, helps understand the central tendency.
        "Median":df[col].median(),      # Middle value when sorted,  useful when data has outliers.
        "Mode":df[col].mode()[0],       # Most frequent value
        "Skewness":df[col].skew(),      # Measure of asymmetry in data distribution, skewness > 0:Right-skewed.skewness < 0:Left-skewed.
        "Variance":df[col].var(),       # Measure of how far data points are spread from the mean
        "Standard Deviation":df[col].std(),    # Square root of variance (spread of data)
        "IQR":df[col].quantile(0.75)-df[col].quantile(0.25),   # Measures the range of the middle 50% of the data.
        "Range":df[col].max()-df[col].min()    # Difference between max and min values
}
stats=pd.DataFrame(stat)
stats
```

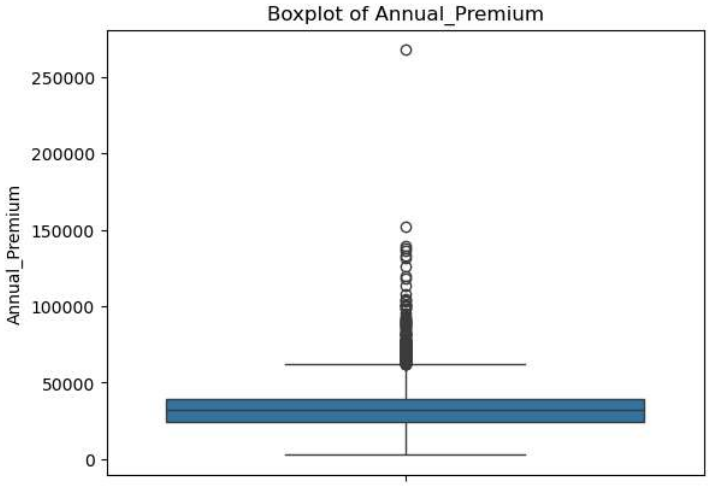
Out[12]:

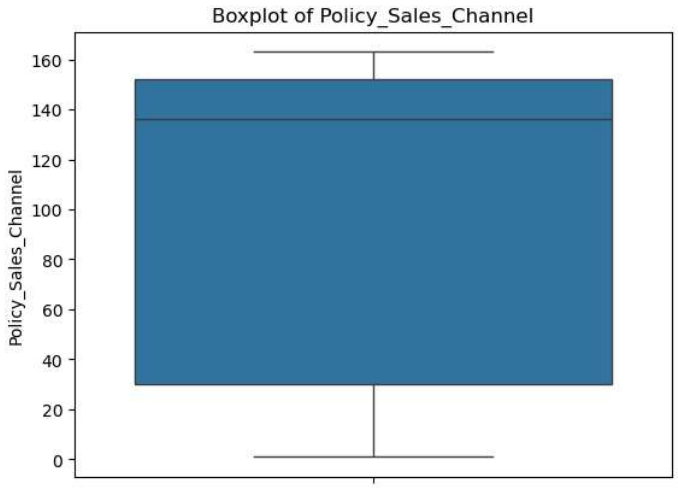
	Age	Region_Code	Annual_Premium	Policy_Sales_Channel	Vintage
Mean	38.814281	26.423542	3.061083e+04	112.410441	155.562756
Median	36.000000	28.000000	3.174700e+04	136.000000	156.000000
Mode	24.000000	28.000000	2.630000e+03	152.000000	258.000000
Skewness	0.669561	-0.124908	4.640154e-01	-0.915043	-0.019968
Variance	240.713555	172.998730	2.743264e+08	2913.439843	7043.856608
Standard Deviation	15.514946	13.152898	1.656280e+04	53.976290	83.927687
IQR	24.000000	20.000000	1.510950e+04	122.000000	145.000000
Range	64.000000	52.000000	2.650680e+05	162.000000	289.000000

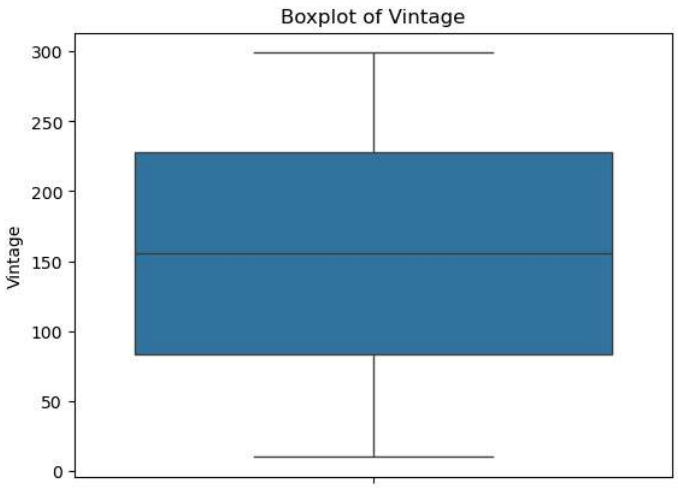
In [13]:
```
for col in num_col:
    sns.boxplot(df[col])    # Identifies outliers,data point that is significantly different from the rest of the dataset
    plt.title(f"Boxplot of {col}")
    plt.show()
```


Boxplot of Age
Age
80
70
60
50
40
30
20


Boxplot of Region_Code
Region_Code
50
40
30
20
10
0


Boxplot of Annual_Premium
Annual_Premium
250000
200000
150000
100000
50000
0

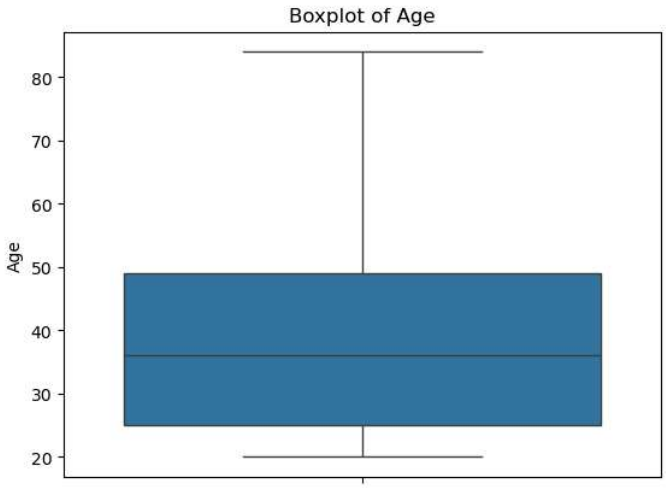




In [14]:
```
#### Checking and Handling outliers
```

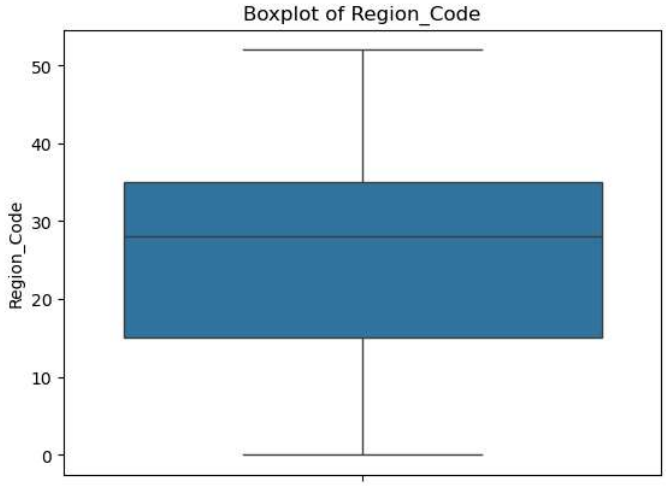
In [15]:
```
for col in num_col:
    q1=df[col].quantile(0.25)
    q3=df[col].quantile(0.75)
    iqr=q3-q1     # Measures the range of the middle 50% of the data.
    lowertail=q1-1.5*iqr   # Any value below this is an outlier.
    uppertail=q3+1.5*iqr   # Any value above this is an outlier.
    print(f"IQR : {iqr}")
    print(f"LowerTail : {lowertail}")
    print(f"UpperTail : {uppertail}")

    # check outlier
    outlier=df[(df[col]<lowertail) | (df[col]>uppertail)]    # This filters rows where values fall outside the lower or upper tail.
    outlier

    # handling outlier
    df.loc[(df[col]<lowertail),col]=lowertail
    df.loc[(df[col]>uppertail),col]=uppertail
    print("\nBoxplot after handling outliers :")
    sns.boxplot(df[col])     # helps visualize data before and after outlier handling.
    plt.title(f"Boxplot of {col}")
    plt.show()
    print("*"*100)
```

```
IQR : 24.0
LowerTail : -11.0
UpperTail : 85.0

Boxplot after handling outliers :
```



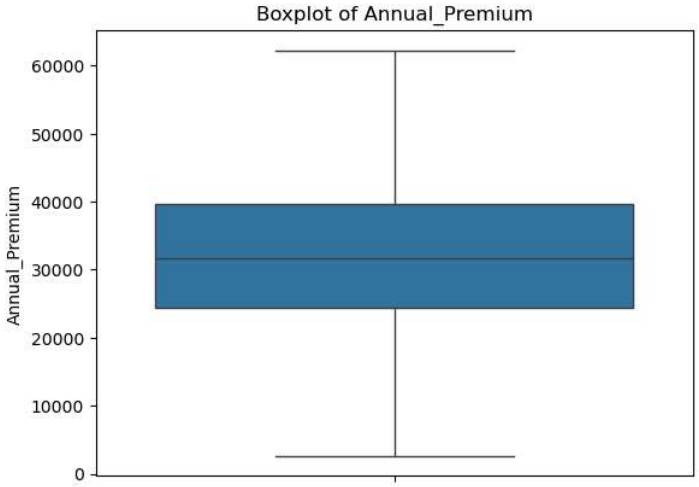
****
IQR : 20.0
LowerTail : -15.0
UpperTail : 65.0

Boxplot after handling outliers :



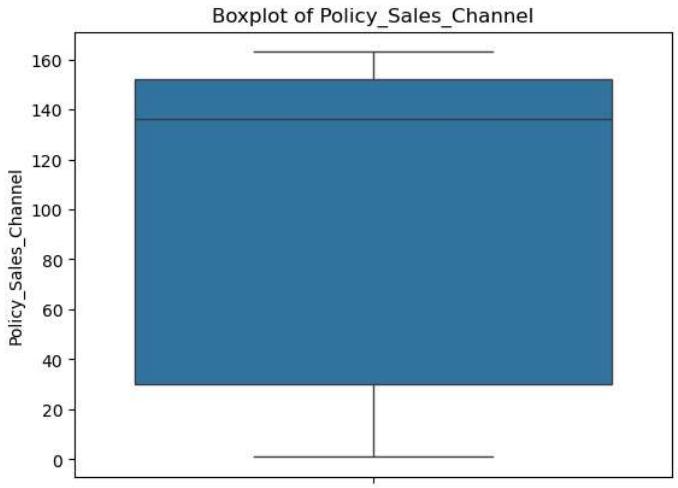
****
IQR : 15109.5
LowerTail : 1798.75
UpperTail : 62236.75

Boxplot after handling outliers :



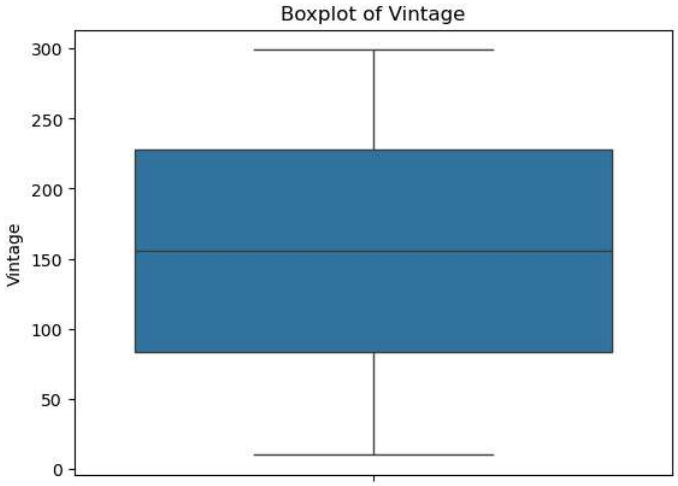
****
IQR : 122.0
LowerTail : -153.0
UpperTail : 335.0

Boxplot after handling outliers :



****
IQR : 145.0
LowerTail : -134.5
UpperTail : 445.5

Boxplot after handling outliers :



****

Zscore of numerical columns

In [16]:
```
for col in num_col:
    print(f"Zscore of {col} : ")
    Zscore=zscore(df[col])  # Measures how many standard deviations a data point is from the mean.Values greater than +3 or less than -3 are outliers
    print(Zscore)
    print("*"*100)
```

```
Zscore of Age :
0       0.334257
1       2.396887
2       0.527629
3      -1.148259
4      -0.632601
          ...
9994   -0.245858
9995   -0.954887
9996   -0.697058
9997   -0.761516
9998   -0.697058
Name: Age, Length: 9999, dtype: float64
**************************************************************************************************
Zscore of Region_Code :
0       0.119862
1      -1.780955
2       0.119862
3      -1.172693
4       1.108287
          ...
9994    0.119862
9995    0.804156
9996    0.652091
9997   -0.868562
9998   -0.944595
Name: Region_Code, Length: 9999, dtype: float64
**************************************************************************************************
Zscore of Annual_Premium :
0       0.657347
1       0.210395
2       0.517795
3      -0.107278
4      -0.179832
          ...
9994    0.295482
9995   -0.583820
9996   -0.432252
9997    0.197732
9998   -0.313568
Name: Annual_Premium, Length: 9999, dtype: float64
**************************************************************************************************
Zscore of Policy_Sales_Channel :
0      -1.600976
1      -1.600976
2      -1.600976
3       0.733499
4       0.733499
          ...
9994    0.770554
9995    0.733499
9996    0.881719
9997    0.733499
9998    0.733499
Name: Policy_Sales_Channel, Length: 9999, dtype: float64
**************************************************************************************************
Zscore of Vintage :
0       0.732063
1       0.326932
2      -1.531904
3       0.565244
4      -1.388917
          ...
9994   -1.603398
9995   -1.603398
9996    0.207775
9997    0.612906
9998    1.089531
Name: Vintage, Length: 9999, dtype: float64
**************************************************************************************************
```

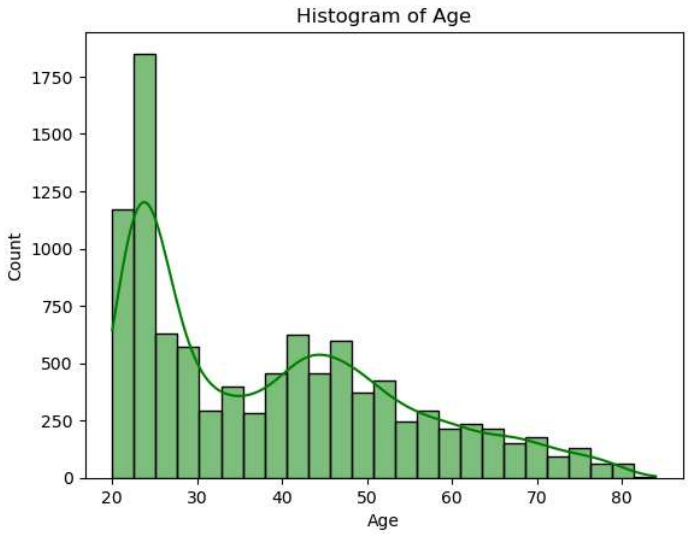
In [17]:
```
### Hypotesis Testing
```

In [18]:
```
for col in num_col:
    stat,p_val=shapiro(df[col])  # check if each numerical column follows a normal distribution.
    print(f"P_Value : {p_val}")
    if p_val>=0.05:
        print(f"Data normally distributed in column {col}")
    else:
        print(f"Data not nomally distributed in column {col}")
    print("-"*100)
```

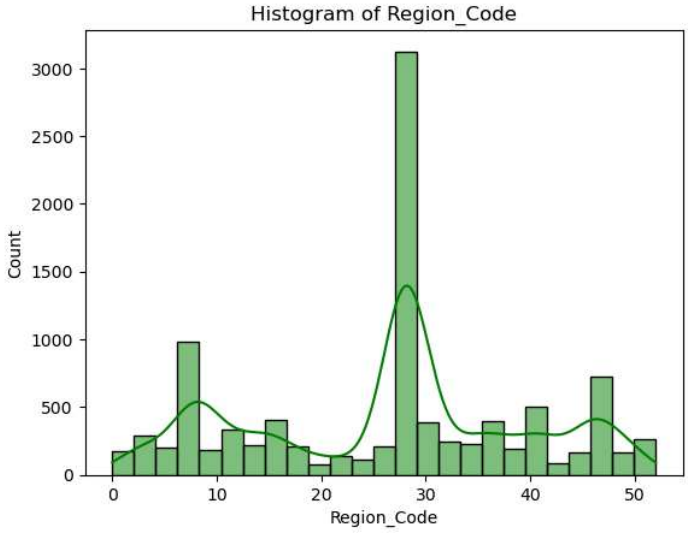
```
P_Value : 3.3365196077198094e-60
Data not nomally distributed in column Age
----------------------------------------------------------------------------------------------------
P_Value : 1.1362214959019626e-52
Data not nomally distributed in column Region_Code
----------------------------------------------------------------------------------------------------
P_Value : 6.000912329705889e-56
Data not nomally distributed in column Annual_Premium
----------------------------------------------------------------------------------------------------
P_Value : 6.060357791392154e-84
Data not nomally distributed in column Policy_Sales_Channel
----------------------------------------------------------------------------------------------------
P_Value : 5.651535550921503e-48
Data not nomally distributed in column Vintage
----------------------------------------------------------------------------------------------------
```

In [19]:
```
### Visualization For Numerical Columns
```

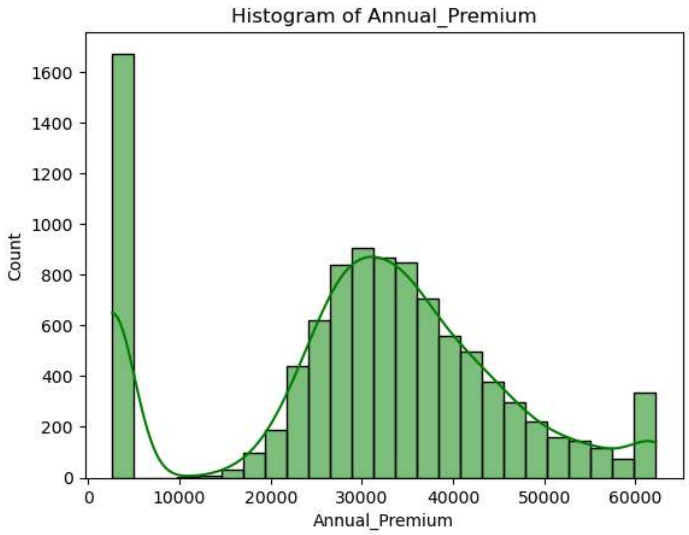
In [20]:
```
for col in num_col:
    sns.histplot(df[col],kde=True, bins=25, color="green")
    plt.title(f"Histogram of {col}")
    plt.show()
    print("*"*100)
```



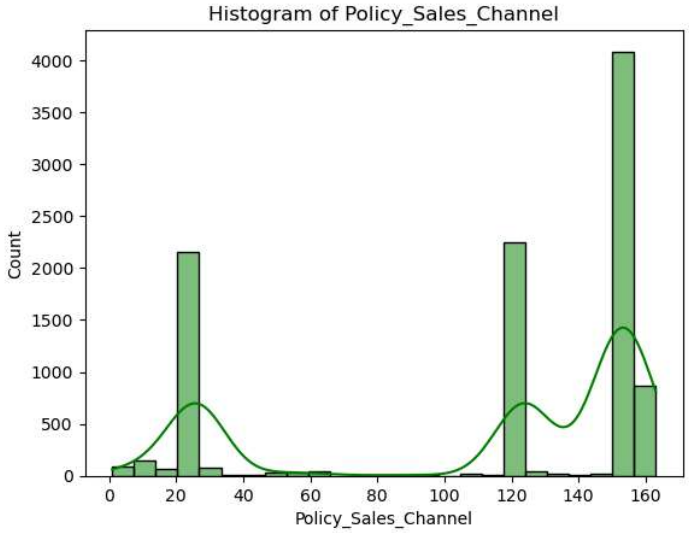

```
****************************************************************************************************
```



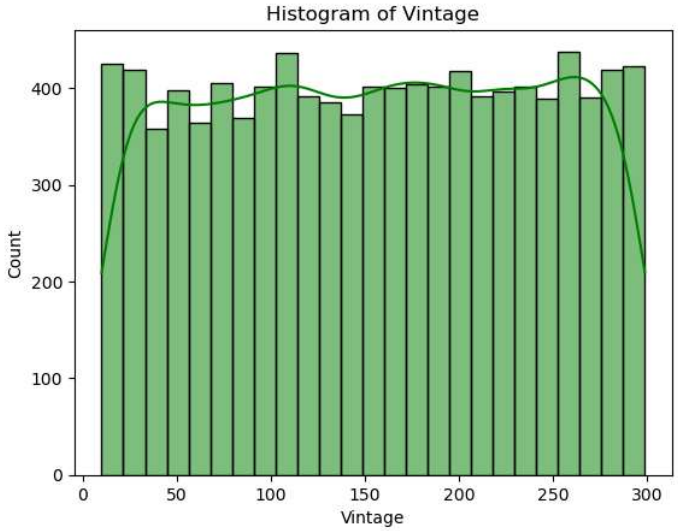

```
****************************************************************************************************
```



****

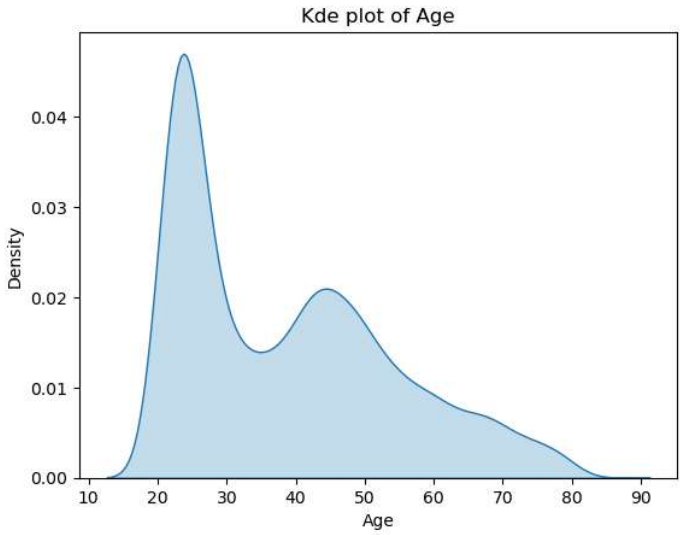


****

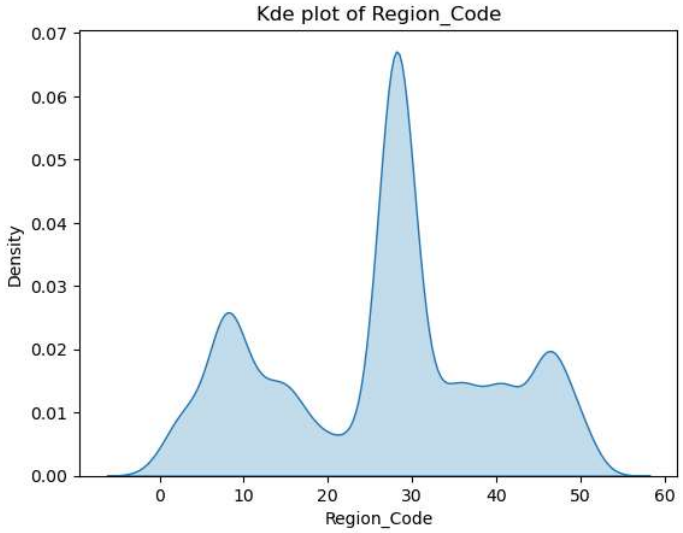


****

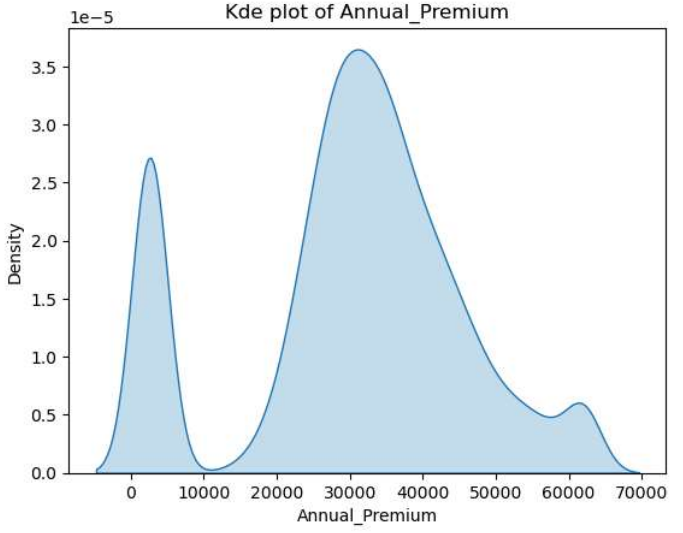
In [21]:
```
for col in num_col:
    sns.kdeplot(df[col],fill=True)
    plt.title(f"Kde plot of {col}")
    plt.show()
    print("*"*100)
```


Kde plot of Age
Density
Age
0.00
0.01
0.02
0.03
0.04
10
20
30
40
50
60
70
80
90

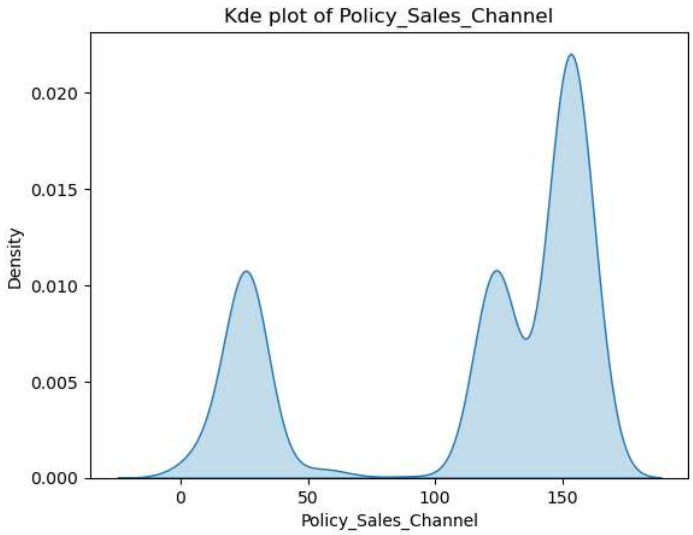
****


Kde plot of Region_Code
Density
Region_Code
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0
10
20
30
40
50
60

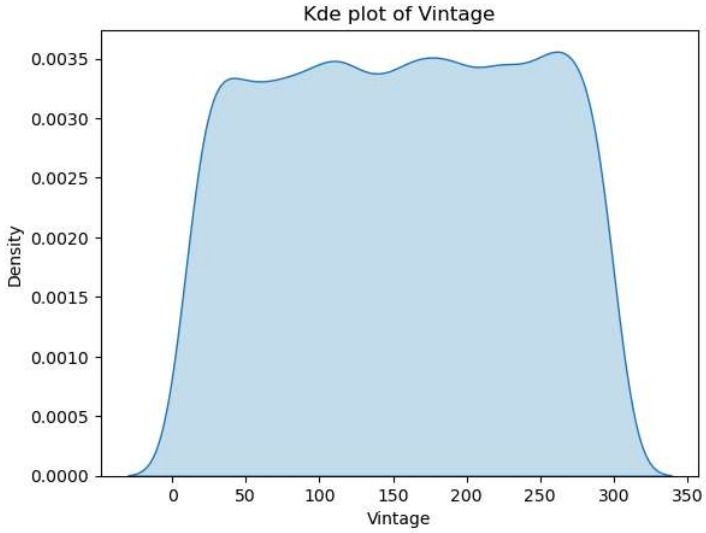
****


Kde plot of Annual_Premium
1e−5
Density
Annual_Premium
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
0
10000
20000
30000
40000
50000
60000
70000

****

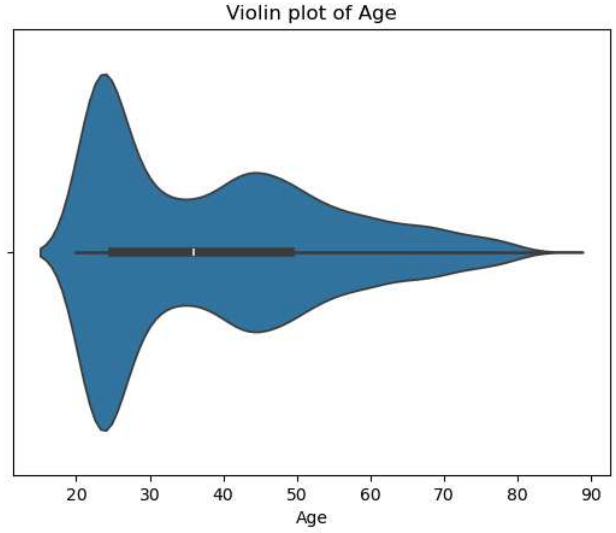


****

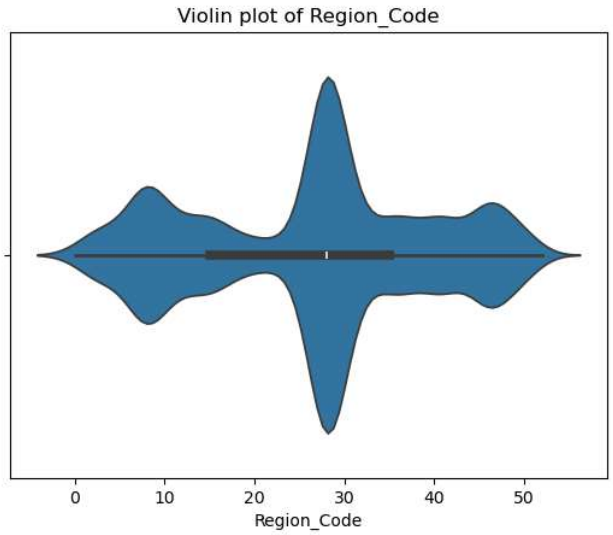


****

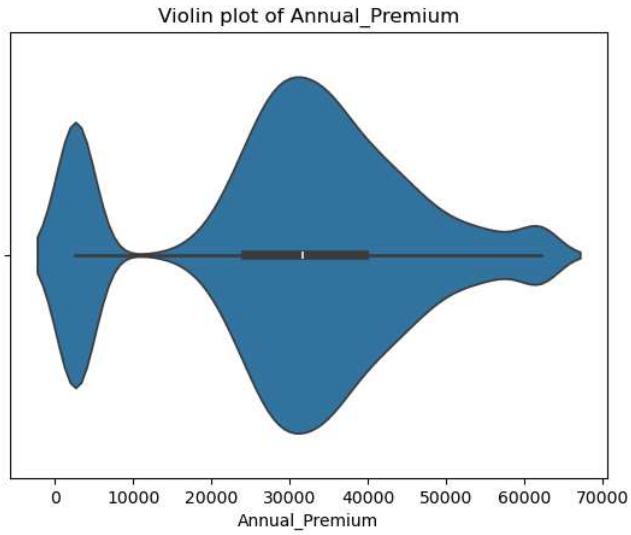
In [22]:
```
for col in num_col:
    sns.violinplot(x=df[col],fill=True)
    plt.title(f"Violin plot of {col}")
    plt.show()
    print("*"*100)
```



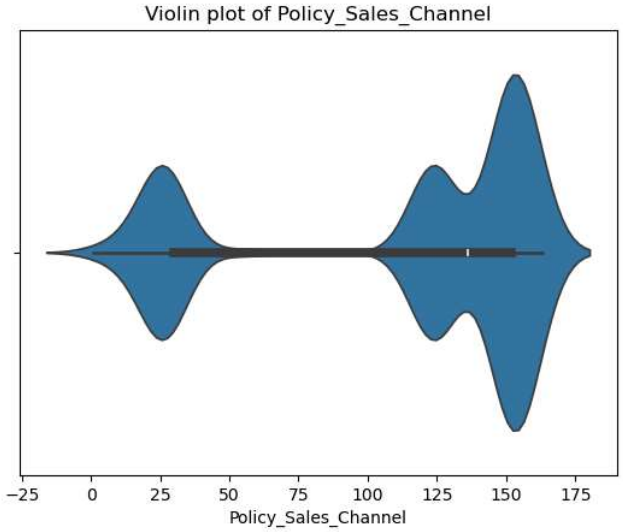
****


Violin plot of Region_Code
0
10
20
30
40
50
Region_Code

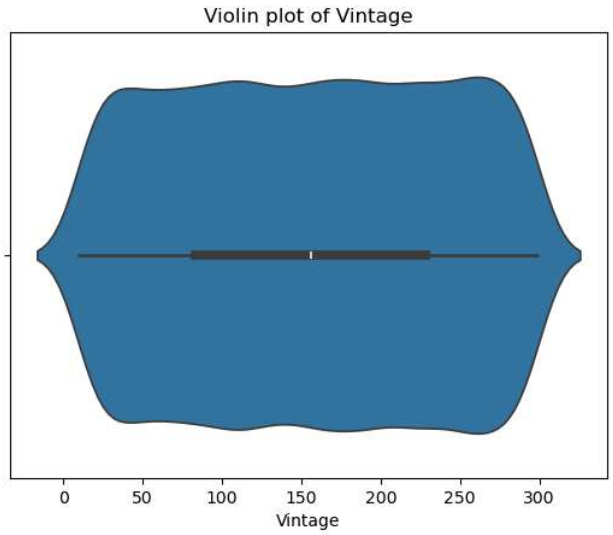
****


Violin plot of Annual_Premium
0
10000
20000
30000
40000
50000
60000
70000
Annual_Premium

****

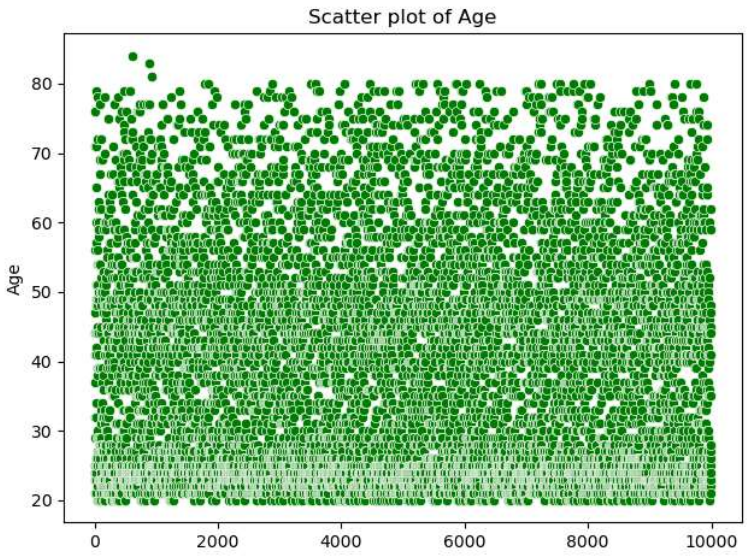

Violin plot of Policy_Sales_Channel
−25
0
25
50
75
100
125
150
175
Policy_Sales_Channel

****



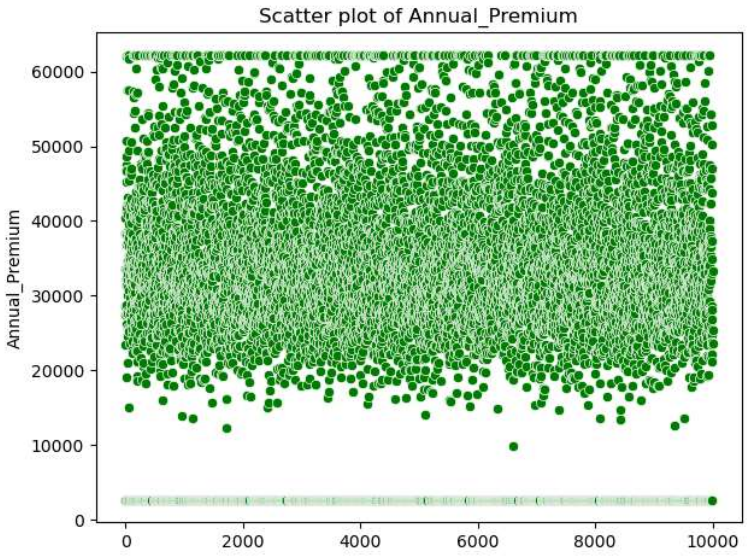
****

In [23]:
```
for col in num_col:
    sns.scatterplot(df[col],color="Green")  # visualize the relationship between two numerical variables.
    plt.title(f"Scatter plot of {col}")
    plt.tight_layout()
    plt.show()
    print("*"*100)
```

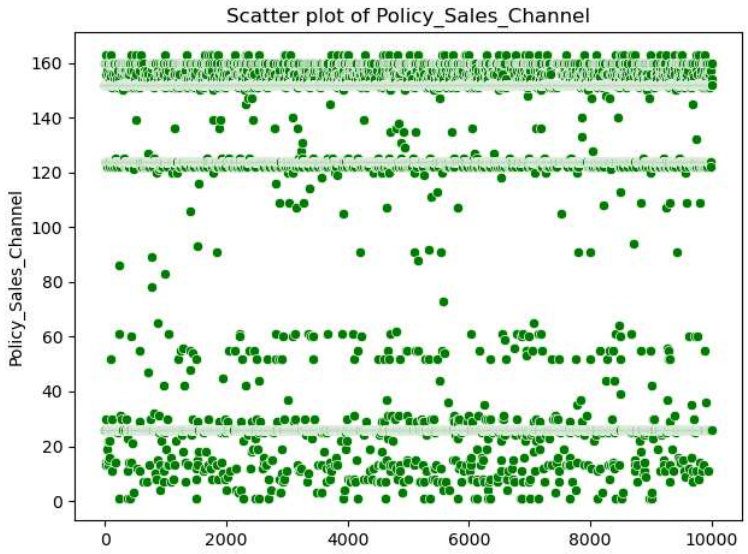


****

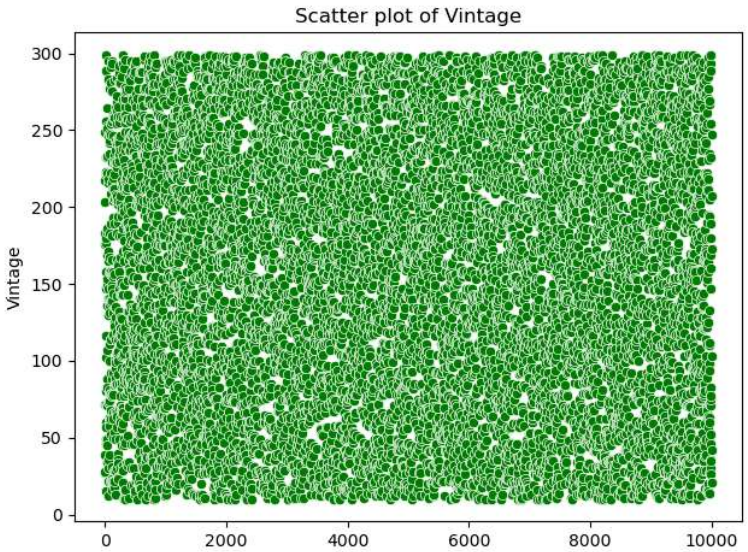
****



****

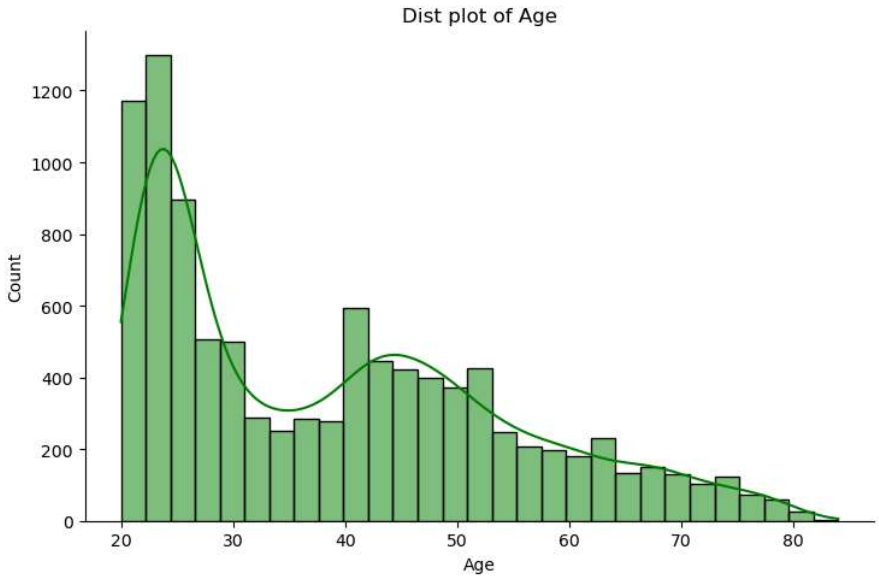


****

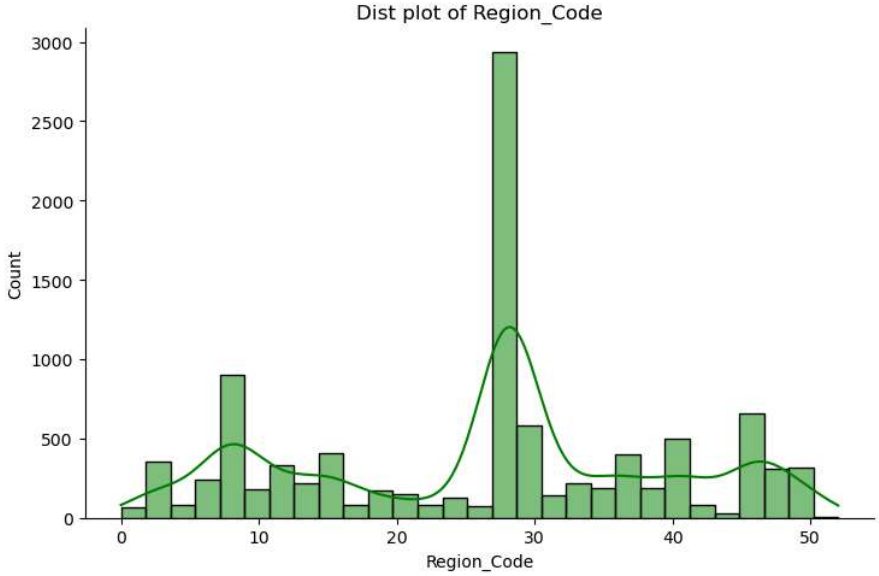


****

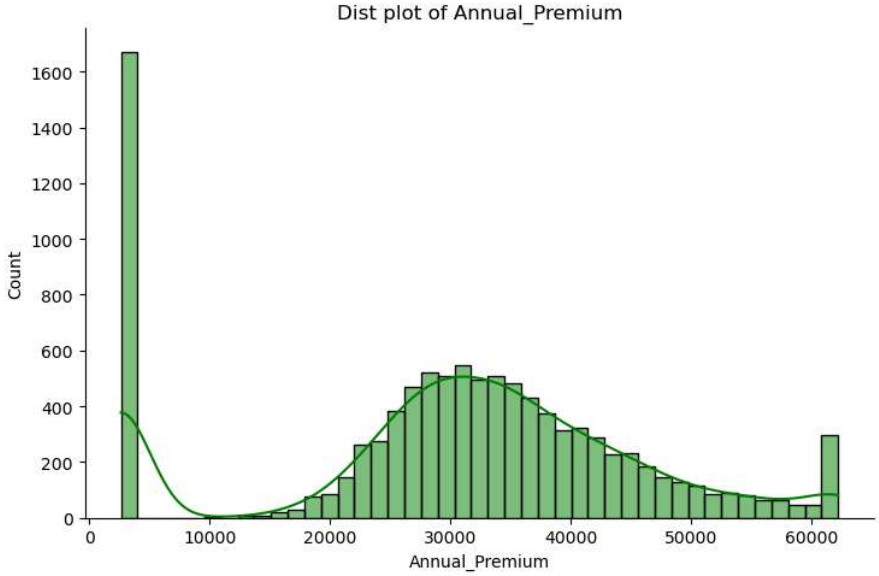
In [24]:
```
for col in num_col:
    sns.displot(df[col], color="Green", kde=True, height=5, aspect=1.5)  # Combines multiple types of distribution plots
    plt.title(f"Dist plot of {col}")
    plt.tight_layout()
    plt.show()
    print("*"*100)
```


Dist plot of Age
Count
Age
0
200
400
600
800
1000
1200
20
30
40
50
60
70
80

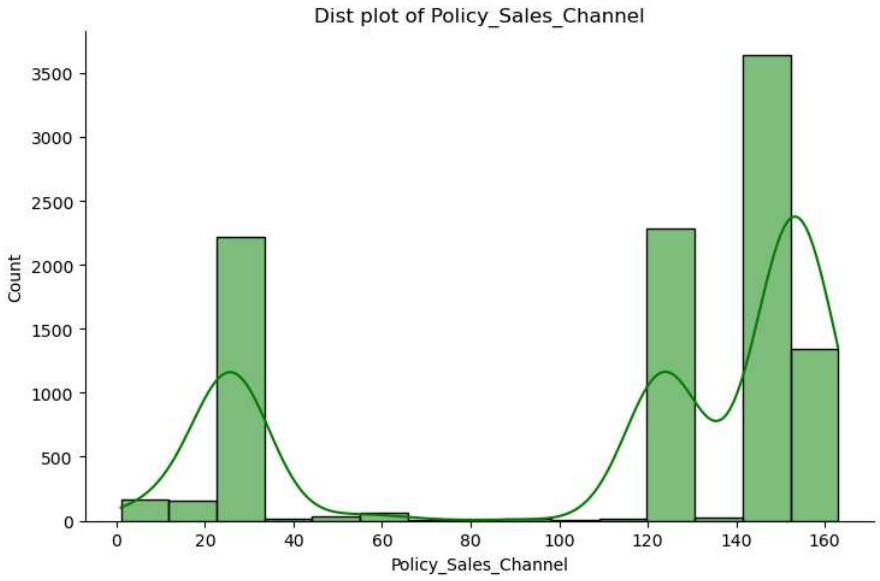
****


Dist plot of Region_Code
Count
Region_Code
0
500
1000
1500
2000
2500
3000
0
10
20
30
40
50

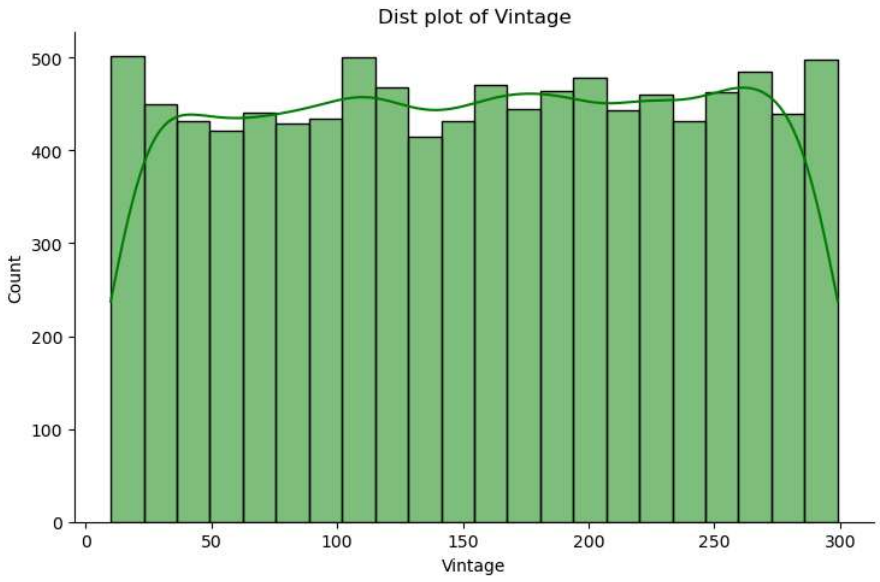
****


Dist plot of Annual_Premium
Count
Annual_Premium
0
200
400
600
800
1000
1200
1400
1600
0
10000
20000
30000
40000
50000
60000

****



****



****

In [25]:
```
### Bivariate Analysis
```

In [26]:
```
df[num_col].cov()
```

Out[26]:

	Age	Region_Code	Annual_Premium	Policy_Sales_Channel	Vintage
Age	240.713555	6.278708	1.963047e+04	-482.397861	0.005205
Region_Code	6.278708	172.998730	-2.160480e+03	-28.285579	-9.368501
Annual_Premium	19630.468610	-2160.480219	2.395981e+08	-106894.838495	-9762.201390
Policy_Sales_Channel	-482.397861	-28.285579	-1.068948e+05	2913.439843	-27.676090
Vintage	0.005205	-9.368501	-9.762201e+03	-27.676090	7043.856608

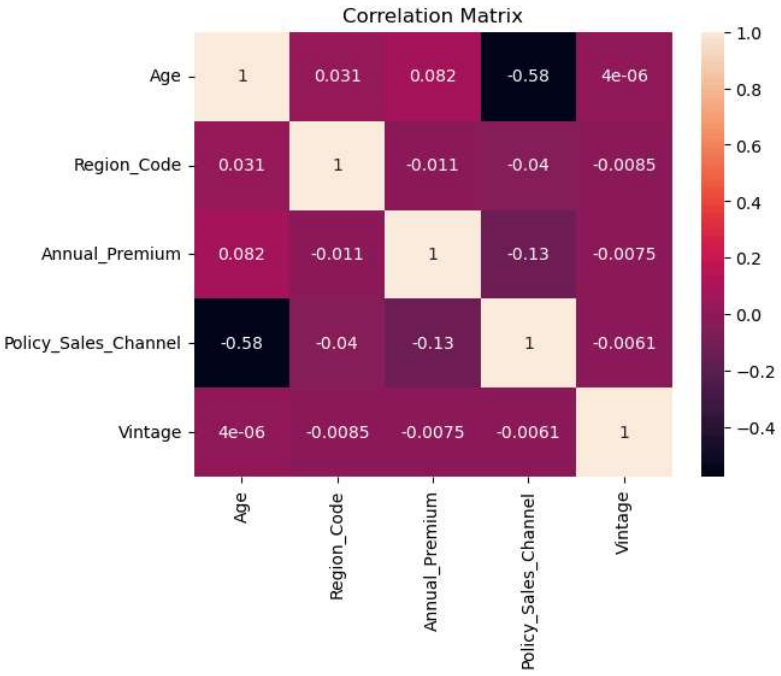
In [27]:
```
df[num_col].corr()
```

Out[27]:

	Age	Region_Code	Annual_Premium	Policy_Sales_Channel	Vintage
Age	1.000000	0.030768	0.081741	-0.576039	0.000004
Region_Code	0.030768	1.000000	-0.010612	-0.039842	-0.008487
Annual_Premium	0.081741	-0.010612	1.000000	-0.127942	-0.007515
Policy_Sales_Channel	-0.576039	-0.039842	-0.127942	1.000000	-0.006109
Vintage	0.000004	-0.008487	-0.007515	-0.006109	1.000000

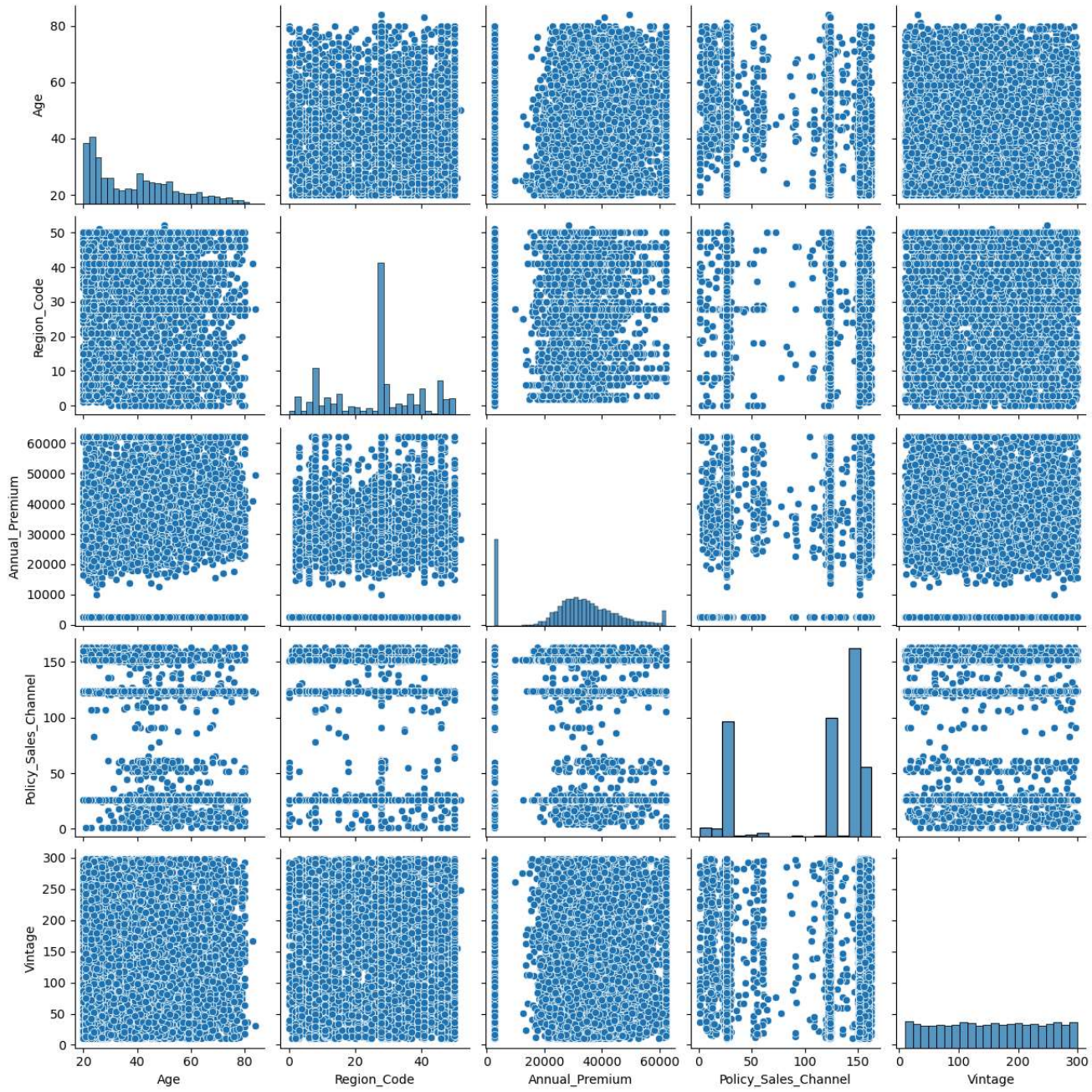
In [28]:
```
sns.heatmap(df[num_col].corr(),annot=True)   # used to analyze correlations between numerical variables
plt.title('Correlation Matrix')
```

```
plt.show()
```



Multivariate Analysis

In [29]:
```
sns.pairplot(df[num_col])
plt.show()
```



Visualization for categorical columns

In [30]:
```
df.head(1)
```

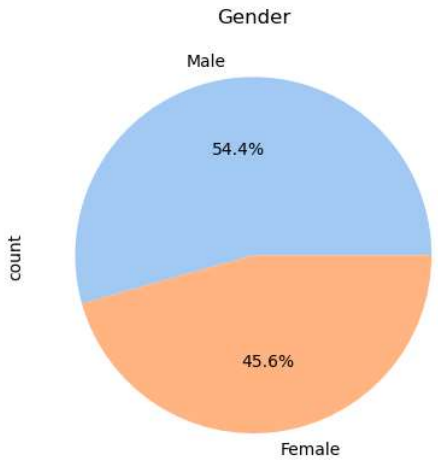
Out[30]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	Male	44	1	28	0	> 2 Years	Yes	40454.0	26	217.0	1

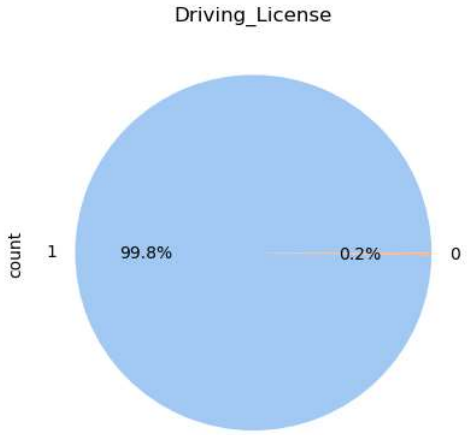
In [31]:
```
cat_col=["Gender","Driving_License","Previously_Insured","Vehicle_Age","Vehicle_Damage","Response"]
for col in cat_col:
    print(f"Statistics for {col} :")
    print(f"Unique Values :",df[col].unique())
    print(f"No. of unique values :",df[col].nunique())
    print(f"Value count :",df[col].value_counts())
    print(f"Mode :",df[col].mode()[0])
    print("*"*100)
```

```
Statistics for Gender :
Unique Values : ['Male' 'Female']
No. of unique values : 2
Value count : Gender
Male      5437
Female    4562
Name: count, dtype: int64
Mode : Male
**********************************************************************************************
Statistics for Driving_License :
Unique Values : [1 0]
No. of unique values : 2
Value count : Driving_License
1    9977
0      22
Name: count, dtype: int64
Mode : 1
**********************************************************************************************
Statistics for Previously_Insured :
Unique Values : [0 1]
No. of unique values : 2
Value count : Previously_Insured
0    5562
1    4437
Name: count, dtype: int64
Mode : 0
**********************************************************************************************
Statistics for Vehicle_Age :
Unique Values : ['> 2 Years' '1-2 Year' '< 1 Year']
No. of unique values : 3
Value count : Vehicle_Age
1-2 Year     5263
< 1 Year     4329
> 2 Years     407
Name: count, dtype: int64
Mode : 1-2 Year
**********************************************************************************************
Statistics for Vehicle_Damage :
Unique Values : ['Yes' 'No']
No. of unique values : 2
Value count : Vehicle_Damage
Yes    5143
No     4856
Name: count, dtype: int64
Mode : Yes
**********************************************************************************************
Statistics for Response :
Unique Values : [1 0]
No. of unique values : 2
Value count : Response
0    8752
1    1247
Name: count, dtype: int64
Mode : 0
**********************************************************************************************
```

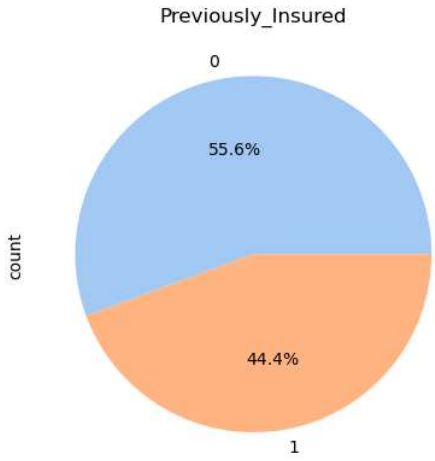
In [32]:
```
for col in cat_col:
    colors = sns.color_palette('pastel')
    df[col].value_counts().plot.pie(autopct='%1.1f%%', colors=colors)
    plt.title(col)
    plt.show()
    print("*"*100)
```



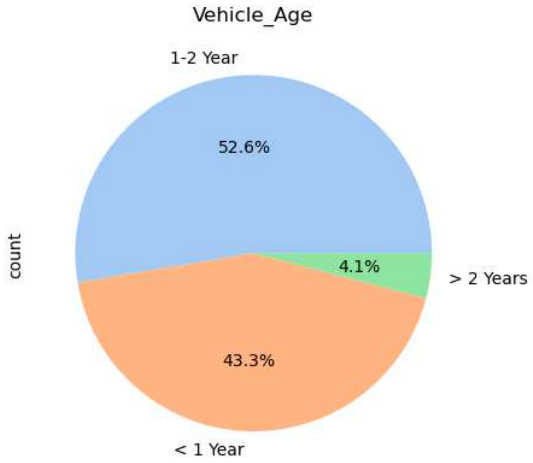

```
**********************************************************************************************
```


Driving_License
count
1
99.8%
0.2%
0

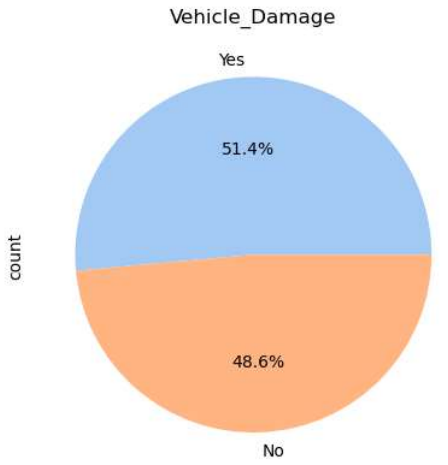
****


Previously_Insured
0
55.6%
count
44.4%
1

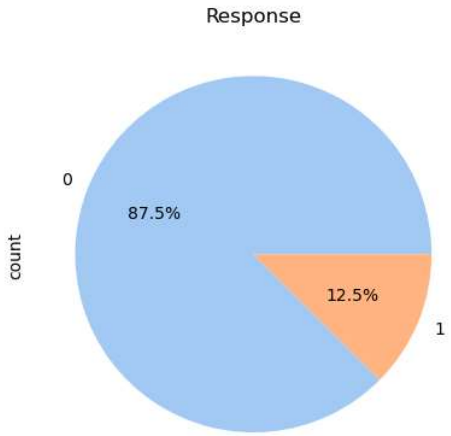
****


Vehicle_Age
1-2 Year
52.6%
count
4.1%
> 2 Years
43.3%
< 1 Year

****



****



****

Converting categorical data into numerical using LabelEncoder or replace

In [33]:
```
df.columns
```

Out[33]:
```
Index(['Gender', 'Age', 'Driving_License', 'Region_Code', 'Previously_Insured',
       'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premium',
       'Policy_Sales_Channel', 'Vintage', 'Response'],
      dtype='object')
```

In [34]:
```
df["Gender"].unique()
```

Out[34]:
```
array(['Male', 'Female'], dtype=object)
```

In [35]:
```
df["Gender"].replace({'Male':1,'Female':0},inplace=True)
```

In [36]:
```
df["Vehicle_Age"].unique()
```

Out[36]:
```
array(['> 2 Years', '1-2 Year', '< 1 Year'], dtype=object)
```

In [37]:
```
df["Vehicle_Age"].replace({'< 1 Year':0,'1-2 Year':1,'> 2 Years':2},inplace=True)
```

In [38]:
```
df['Vehicle_Damage'].unique()
```

Out[38]:
```
array(['Yes', 'No'], dtype=object)
```

In [39]:
```
df['Vehicle_Damage'].replace({'Yes':1,'No':0},inplace=True)
```

In [40]:
```
df.head()
```

Out[40]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	44	1	28	0	2	1	40454.0	26	217.0	1
1	1	76	1	3	0	1	0	33536.0	26	183.0	0
2	1	47	1	28	0	2	1	38294.0	26	27.0	1
3	1	21	1	11	1	0	0	28619.0	152	203.0	0
4	0	29	1	41	1	0	0	27496.0	152	39.0	0

splitting of Data into independant(x) and dependant(y) features for column

In [41]:
```
x=df.drop("Response",axis=1)
x.head()
```

Out[41]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	1	44	1	28	0	2	1	40454.0	26	217.0
1	1	76	1	3	0	1	0	33536.0	26	183.0
2	1	47	1	28	0	2	1	38294.0	26	27.0
3	1	21	1	11	1	0	0	28619.0	152	203.0
4	0	29	1	41	1	0	0	27496.0	152	39.0

Non- multicolinearity

In [42]:
```
vif_df=pd.DataFrame()
vif_df["Independant Featutes"]=x.columns
vif_df
```

Out[42]:

	Independant Featutes
0	Gender
1	Age
2	Driving_License
3	Region_Code
4	Previously_Insured
5	Vehicle_Age
6	Vehicle_Damage
7	Annual_Premium
8	Policy_Sales_Channel
9	Vintage

In [43]:
```
vif_list=[]
for i in range(x.shape[1]):
    vif=variance_inflation_factor(df.to_numpy(),i)
    vif_list.append(vif)
vif_df["VIF"]=vif_list
vif_df
```

Out[43]:

	Independant Featutes	VIF
0	Gender	2.251628
1	Age	18.458872
2	Driving_License	43.221868
3	Region_Code	5.013626
4	Previously_Insured	5.364768
5	Vehicle_Age	6.046053
6	Vehicle_Damage	6.262568
7	Annual_Premium	4.882126
8	Policy_Sales_Channel	8.170869
9	Vintage	4.412026

In [44]:
```
y=df["Response"]
y.head()
```

Out[44]:
```
0    1
1    0
2    1
3    0
4    0
Name: Response, dtype: int64
```

train test split

In [45]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=10)
print("Xtrain :",xtrain.shape)
print("Xtest :",xtest.shape)
print("Ytrain :",ytrain.shape)
print("Ytest :",ytest.shape)
```

```
Xtrain : (7999, 10)
Xtest : (2000, 10)
Ytrain : (7999,)
Ytest : (2000,)
```

Model Training and Model Evaluation

Algorithm 1 : Logistic Regression

In [136…
```
log_reg= LogisticRegression()
log_reg
```

Out[136…
```
LogisticRegression()
```

In [137…
```
log_reg_model = log_reg.fit(xtrain,ytrain)
log_reg_model
```

Out[137…
```
LogisticRegression()
```

In [139…
```
ytrain_pred=log_reg_model.predict(xtrain)
ytest_pred=log_reg_model.predict(xtest)
```

Model Evaluation

In [142…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
--------------------------------------------------------------------------------------------------------
Accuracy Score : 0.8702337792224027
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[6926   89]
 [ 949   35]]
--------------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.88      0.99      0.93      7015
           1       0.28      0.04      0.06       984

    accuracy                           0.87      7999
   macro avg       0.58      0.51      0.50      7999
weighted avg       0.81      0.87      0.82      7999

********************************************************************************************************


Model evaluation for Testing data
********************************************************************************************************
Accuracy Score :  0.8645
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1720   17]
 [ 254    9]]
--------------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.87      0.99      0.93      1737
           1       0.35      0.03      0.06       263

    accuracy                           0.86      2000
   macro avg       0.61      0.51      0.49      2000
weighted avg       0.80      0.86      0.81      2000

--------------------------------------------------------------------------------------------------------
```

Algorithm 2 : Decision Tree

In [143…
```
dt_clf=DecisionTreeClassifier()
dt_clf
```

Out[143…
```
DecisionTreeClassifier()
```

In [144…
```
dt_clf_model=dt_clf.fit(xtrain,ytrain)
dt_clf_model
```

Out[144…
```
DecisionTreeClassifier()
```

In [145…
```
ytrain_pred=dt_clf_model.predict(xtrain)
ytest_pred=dt_clf_model.predict(xtest)
```

Model Evaluation

In [146…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
```

```
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 1.0
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7015    0]
 [   0  984]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7015
           1       1.00      1.00      1.00       984

    accuracy                           1.00      7999
   macro avg       1.00      1.00      1.00      7999
weighted avg       1.00      1.00      1.00      7999

****************************************************************************************************


Model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.814
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1547  190]
 [ 182   81]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.89      0.89      0.89      1737
           1       0.30      0.31      0.30       263

    accuracy                           0.81      2000
   macro avg       0.60      0.60      0.60      2000
weighted avg       0.82      0.81      0.82      2000

----------------------------------------------------------------------------------------------------
```

Decision Tree with Hyperparameter Tunning

In [147…
```
hyperparameters = {"criterion" : ["gini", "entropy", "log_loss"],
                   "max_depth" : np.arange(5,15),
                   "min_samples_split" : np.arange(2,10),
                   "min_samples_leaf" : np.arange(2,10)}
hyperparameters
```

Out[147…
```
{'criterion': ['gini', 'entropy', 'log_loss'],
 'max_depth': array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14]),
 'min_samples_split': array([2, 3, 4, 5, 6, 7, 8, 9]),
 'min_samples_leaf': array([2, 3, 4, 5, 6, 7, 8, 9])}
```

In [148…
```
rscv = RandomizedSearchCV(dt_clf_model, hyperparameters, cv = 5)
rscv = rscv.fit(xtrain, ytrain)
rscv.best_estimator_
```

Out[148…
```
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=3,
                       min_samples_split=3)
```

In [149…
```
dt_clf=DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=3,min_samples_split=3)
dt_clf_rscv_model=dt_clf.fit(xtrain,ytrain)
dt_clf_rscv_model
```

Out[149…
```
DecisionTreeClassifier(criterion='entropy', max_depth=5, min_samples_leaf=3,
                       min_samples_split=3)
```

In [150…
```
ytrain_pred=dt_clf_rscv_model.predict(xtrain)
ytest_pred=dt_clf_rscv_model.predict(xtest)
```

Model Evaluation

In [151…
```
print("Model Evaluation for Training Data")
print("-"*100)
```

```
# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.8769846230778847
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7015    0]
 [ 984    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.88      1.00      0.93      7015
           1       0.00      0.00      0.00       984

    accuracy                           0.88      7999
   macro avg       0.44      0.50      0.47      7999
weighted avg       0.77      0.88      0.82      7999

****************************************************************************************************


Model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.8685
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1737    0]
 [ 263    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.00      0.00      0.00       263

    accuracy                           0.87      2000
   macro avg       0.43      0.50      0.46      2000
weighted avg       0.75      0.87      0.81      2000

----------------------------------------------------------------------------------------------------
```

Algorithm4-Random Forest

In [152…
```
rf=RandomForestClassifier()
rf
```

Out[152…
```
RandomForestClassifier()
```

In [153…
```
rf_model=rf.fit(xtrain,ytrain)
rf_model
```

Out[153…
```
RandomForestClassifier()
```

In [154…
```
ytrain_pred=rf_model.predict(xtrain)
ytest_pred=rf_model.predict(xtest)
```

In [155…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.9998749843730467
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7015    0]
 [   1  983]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      7015
           1       1.00      1.00      1.00       984

    accuracy                           1.00      7999
   macro avg       1.00      1.00      1.00      7999
weighted avg       1.00      1.00      1.00      7999

****************************************************************************************************


Model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.858
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1690   47]
 [ 237   26]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.88      0.97      0.92      1737
           1       0.36      0.10      0.15       263

    accuracy                           0.86      2000
   macro avg       0.62      0.54      0.54      2000
weighted avg       0.81      0.86      0.82      2000

----------------------------------------------------------------------------------------------------
```

Algorithm5-Random Tree Classifier Hyper parameter turnning

In [156…
```
hyperparameters={"criterion":["gini","entropy","log_loss"],
                 "max_depth":np.arange(5,15),
                 "min_samples_split":np.arange(2,10),
                 "min_samples_leaf":np.arange(2,10),
                 "n_estimators":np.arange(1,20)}
hyperparameters
```

Out[156…
```
{'criterion': ['gini', 'entropy', 'log_loss'],
 'max_depth': array([ 5,  6,  7,  8,  9, 10, 11, 12, 13, 14]),
 'min_samples_split': array([2, 3, 4, 5, 6, 7, 8, 9]),
 'min_samples_leaf': array([2, 3, 4, 5, 6, 7, 8, 9]),
 'n_estimators': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19])}
```

In [157…
```
rscv=RandomizedSearchCV(rf_model,hyperparameters,cv=5)
```

In [158…
```
rscv_model=rscv.fit(xtrain,ytrain)
rscv_model.best_estimator_
```

Out[158…
```
RandomForestClassifier(criterion='entropy', max_depth=9, min_samples_leaf=5,
                       min_samples_split=7, n_estimators=7)
```

In [159…
```
rfc=RandomForestClassifier(criterion='entropy', max_depth=9, min_samples_leaf=5,
                       min_samples_split=7, n_estimators=7)
```

In [160…
```
rfc_model=rfc.fit(xtrain,ytrain)
rfc_model
```

Out[160…
```
RandomForestClassifier(criterion='entropy', max_depth=9, min_samples_leaf=5,
                       min_samples_split=7, n_estimators=7)
```

In [161…
```
ytrain_pred=rfc_model.predict(xtrain)
ytest_pred=rfc_model.predict(xtest)
```

Model Evaluation with Random Forest Hyperparameter Tunning

In [162…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.8786098262282785
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7013    2]
 [ 969   15]]
----------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.88      1.00      0.94      7015
           1       0.88      0.02      0.03       984

    accuracy                           0.88      7999
   macro avg       0.88      0.51      0.48      7999
weighted avg       0.88      0.88      0.82      7999

****************************************************************************************************


Model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.868
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1736    1]
 [ 263    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.00      0.00      0.00       263

    accuracy                           0.87      2000
   macro avg       0.43      0.50      0.46      2000
weighted avg       0.75      0.87      0.81      2000

----------------------------------------------------------------------------------------------------
```

6. Adaboost Algorithm

In [163…
```
adb=AdaBoostClassifier()
adb_model=adb.fit(xtrain,ytrain)
adb_model
```

Out[163…
```
AdaBoostClassifier()
```

In [164…
```
ytrain_pred=adb_model.predict(xtrain)
ytest_pred=adb_model.predict(xtest)
```

In [166…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nmodel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
--------------------------------------------------------------------------------------------------------
Accuracy Score : 0.8754844355544443
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[6978   37]
 [ 959   25]]
--------------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.88      0.99      0.93      7015
           1       0.40      0.03      0.05       984

    accuracy                           0.88      7999
   macro avg       0.64      0.51      0.49      7999
weighted avg       0.82      0.88      0.82      7999

********************************************************************************************************


model evaluation for Testing data
********************************************************************************************************
Accuracy Score :  0.8675
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1727   10]
 [ 255    8]]
--------------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.87      0.99      0.93      1737
           1       0.44      0.03      0.06       263

    accuracy                           0.87      2000
   macro avg       0.66      0.51      0.49      2000
weighted avg       0.82      0.87      0.81      2000

--------------------------------------------------------------------------------------------------------
```

7. Adaboost with hyperparameter tunning

In [176…
```
hyperparameters={"n_estimators":np.arange(2,41),
                 "learning_rate":[0,0.1,0.001,0.1,1]
                }
hyperparameters
```

Out[176…
```
{'n_estimators': array([ 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,
        36, 37, 38, 39, 40]),
 'learning_rate': [0, 0.1, 0.001, 0.1, 1]}
```

In [177…
```
rscv_adb=RandomizedSearchCV(adb_model,hyperparameters,cv=5)
rscv_adb_model=rscv_adb.fit(xtrain,ytrain)
rscv_adb_model.best_estimator_
```

Out[177…
```
AdaBoostClassifier
AdaBoostClassifier(learning_rate=0.1, n_estimators=29)
```

In [178…
```
adb=AdaBoostClassifier(learning_rate=0.1, n_estimators=29)
```

In [179…
```
rscv_adb_model=adb.fit(xtrain,ytrain)
rscv_adb_model
```

Out[179…
```
AdaBoostClassifier
AdaBoostClassifier(learning_rate=0.1, n_estimators=29)
```

In [180…
```
ytrain_pred=rscv_adb_model.predict(xtrain)
ytest_pred=rscv_adb_model.predict(xtest)
```

In [181…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)
```

```
print("\n\nmodel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.8769846230778847
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7015    0]
 [ 984    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.88      1.00      0.93      7015
           1       0.00      0.00      0.00       984

    accuracy                           0.88      7999
   macro avg       0.44      0.50      0.47      7999
weighted avg       0.77      0.88      0.82      7999

****************************************************************************************************


model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.8685
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1737    0]
 [ 263    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.00      0.00      0.00       263

    accuracy                           0.87      2000
   macro avg       0.43      0.50      0.46      2000
weighted avg       0.75      0.87      0.81      2000

----------------------------------------------------------------------------------------------------
```

8. GradientBoost Algo

In [182…
```
gb=GradientBoostingClassifier()
gb_model=gb.fit(xtrain,ytrain)
gb_model
```

Out[182…
```
GradientBoostingClassifier()
```

In [183…
```
ytrain_pred=gb_model.predict(xtrain)
ytest_pred=gb_model.predict(xtest)
```

In [184…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nmodel evaluation for Testing data")
```

```
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.8808601075134391
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7013    2]
 [ 951   33]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.88      1.00      0.94      7015
           1       0.94      0.03      0.06       984

    accuracy                           0.88      7999
   macro avg       0.91      0.52      0.50      7999
weighted avg       0.89      0.88      0.83      7999

****************************************************************************************************


model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.867
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1734    3]
 [ 263    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.00      0.00      0.00       263

    accuracy                           0.87      2000
   macro avg       0.43      0.50      0.46      2000
weighted avg       0.75      0.87      0.81      2000

----------------------------------------------------------------------------------------------------
```

9. Gradientboost with hyperparameter tunning

In [185…
```
hyperparameters={"n_estimators":np.arange(2,21),
                 "learning_rate":[0,0.2,0.5,0.009,0.06]
                 }
hyperparameters
```

Out[185…
```
{'n_estimators': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
        19, 20]),
 'learning_rate': [0, 0.2, 0.5, 0.009, 0.06]}
```

In [186…
```
rscv_gb=RandomizedSearchCV(gb_model,hyperparameters,cv=5)
rscv_gb_model=rscv_gb.fit(xtrain,ytrain)
rscv_gb_model.best_estimator_
```

Out[186…
```
GradientBoostingClassifier(learning_rate=0.06, n_estimators=10)
```

In [187…
```
gb=GradientBoostingClassifier(learning_rate=0.06, n_estimators=10)
```

In [188…
```
rscv_gb_model=gb.fit(xtrain,ytrain)
rscv_gb_model
```

Out[188…
```
GradientBoostingClassifier(learning_rate=0.06, n_estimators=10)
```

In [189…
```
ytrain_pred=rscv_gb_model.predict(xtrain)
ytest_pred=rscv_gb_model.predict(xtest)
```

In [190…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nmodel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.8769846230778847
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7015    0]
 [ 984    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.88      1.00      0.93      7015
           1       0.00      0.00      0.00       984

    accuracy                           0.88      7999
   macro avg       0.44      0.50      0.47      7999
weighted avg       0.77      0.88      0.82      7999

****************************************************************************************************


model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.8685
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1737    0]
 [ 263    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.00      0.00      0.00       263

    accuracy                           0.87      2000
   macro avg       0.43      0.50      0.46      2000
weighted avg       0.75      0.87      0.81      2000

----------------------------------------------------------------------------------------------------
```

10. XGBoost Algo

In [191…
```
xgb=XGBClassifier()
xgb_model=xgb.fit(xtrain,ytrain)
xgb_model
```

Out[191…

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [192…

```
ytrain_pred=xgb_model.predict(xtrain)
ytest_pred=xgb_model.predict(xtest)
```

In [193…

```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nmodel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
--------------------------------------------------------------------------------------------------------
Accuracy Score : 0.9662457807225904
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[6998   17]
 [ 253  731]]
--------------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.97      1.00      0.98      7015
           1       0.98      0.74      0.84       984

    accuracy                           0.97      7999
   macro avg       0.97      0.87      0.91      7999
weighted avg       0.97      0.97      0.96      7999

********************************************************************************************************


model evaluation for Testing data
********************************************************************************************************
Accuracy Score :  0.853
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1664   73]
 [ 221   42]]
--------------------------------------------------------------------------------------------------------
Classification Report :
               precision    recall  f1-score   support

           0       0.88      0.96      0.92      1737
           1       0.37      0.16      0.22       263

    accuracy                           0.85      2000
   macro avg       0.62      0.56      0.57      2000
weighted avg       0.81      0.85      0.83      2000

--------------------------------------------------------------------------------------------------------
```

11. XGBoost Algo with hyperparameter tunning

In [194…]
```
hyperparameters={"n_estimators":np.arange(2,21),
                 "learning_rate":[0.5,0.33,0.22,1,0.011]
                }
hyperparameters
```

Out[194…]
```
{'n_estimators': array([ 2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17, 18,
        19, 20]),
 'learning_rate': [0.5, 0.33, 0.22, 1, 0.011]}
```

In [195…]
```
rscv_xgb=RandomizedSearchCV(xgb_model,hyperparameters,cv=5)
rscv_xgb_model=rscv_xgb.fit(xtrain,ytrain)
rscv_xgb_model.best_estimator_
```

Out[195…]

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.011, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [196…]
```
xgb=XGBClassifier(n_estimators=20,learning_rate=0.011)
```

In [197…]
```
rscv_xgb_model=xgb.fit(xtrain,ytrain)
rscv_xgb_model
```

Out[197…]

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=0.011, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
```

In [198…
```
ytrain_pred=rscv_xgb_model.predict(xtrain)
ytest_pred=rscv_xgb_model.predict(xtest)
```

In [199…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.8769846230778847
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7015    0]
 [ 984    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.88      1.00      0.93      7015
           1       0.00      0.00      0.00       984

    accuracy                           0.88      7999
   macro avg       0.44      0.50      0.47      7999
weighted avg       0.77      0.88      0.82      7999

****************************************************************************************************


Model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.8685
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1737    0]
 [ 263    0]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.00      0.00      0.00       263

    accuracy                           0.87      2000
   macro avg       0.43      0.50      0.46      2000
weighted avg       0.75      0.87      0.81      2000

----------------------------------------------------------------------------------------------------
```

12. K-Nearest Neighbors Algorithm

In [200…
```
knn=KNeighborsClassifier(n_neighbors=4)
knn
```

Out[200…
```
KNeighborsClassifier(n_neighbors=4)
```

In [201…
```
knn_model=knn.fit(xtrain,ytrain)
knn_model
```

Out[201…

```
KNeighborsClassifier(n_neighbors=4)
```

In [202…

```
ytrain_pred=knn_model.predict(xtrain)
ytest_pred=knn_model.predict(xtest)
```

In [203…

```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
----------------------------------------------------------------------------------------------------
Accuracy Score : 0.8839854981872735
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[6991   24]
 [ 904   80]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.89      1.00      0.94      7015
           1       0.77      0.08      0.15       984

    accuracy                           0.88      7999
   macro avg       0.83      0.54      0.54      7999
weighted avg       0.87      0.88      0.84      7999

****************************************************************************************************


Model evaluation for Testing data
****************************************************************************************************
Accuracy Score :  0.866
----------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1728    9]
 [ 259    4]]
----------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.87      0.99      0.93      1737
           1       0.31      0.02      0.03       263

    accuracy                           0.87      2000
   macro avg       0.59      0.51      0.48      2000
weighted avg       0.80      0.87      0.81      2000

----------------------------------------------------------------------------------------------------
```

13. K-Nearest Neighbors Algorithm with hyperparameter tunning

In [204…

```
hyperparameters={"p":[1,2],
                 "n_neighbors":np.arange(2,10)
                }
hyperparameters
```

Out[204…
```
{'p': [1, 2], 'n_neighbors': array([2, 3, 4, 5, 6, 7, 8, 9])}
```

In [205…
```
rscv_knn=RandomizedSearchCV(knn_model,hyperparameters,cv=5)
rscv_knn_model=rscv_knn.fit(xtrain,ytrain)
rscv_knn_model.best_estimator_
```

Out[205…
```
KNeighborsClassifier(n_neighbors=8, p=1)
```

In [206…
```
knn=KNeighborsClassifier(n_neighbors=8, p=1)
```

In [207…
```
rscv_knn_model=knn.fit(xtrain,ytrain)
rscv_knn_model
```

Out[207…
```
KNeighborsClassifier(n_neighbors=8, p=1)
```

In [208…
```
ytrain_pred=rscv_knn_model.predict(xtrain)
ytest_pred=rscv_knn_model.predict(xtest)
```

In [209…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
--------------------------------------------------------------------------------------------------------
Accuracy Score : 0.8787348418552319
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7008    7]
 [ 963   21]]
--------------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.88      1.00      0.94      7015
           1       0.75      0.02      0.04       984

    accuracy                           0.88      7999
   macro avg       0.81      0.51      0.49      7999
weighted avg       0.86      0.88      0.83      7999

********************************************************************************************************


Model evaluation for Testing data
********************************************************************************************************
Accuracy Score :  0.8695
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1736    1]
 [ 260    3]]
--------------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.75      0.01      0.02       263

    accuracy                           0.87      2000
   macro avg       0.81      0.51      0.48      2000
weighted avg       0.85      0.87      0.81      2000

--------------------------------------------------------------------------------------------------------
```

In [210…

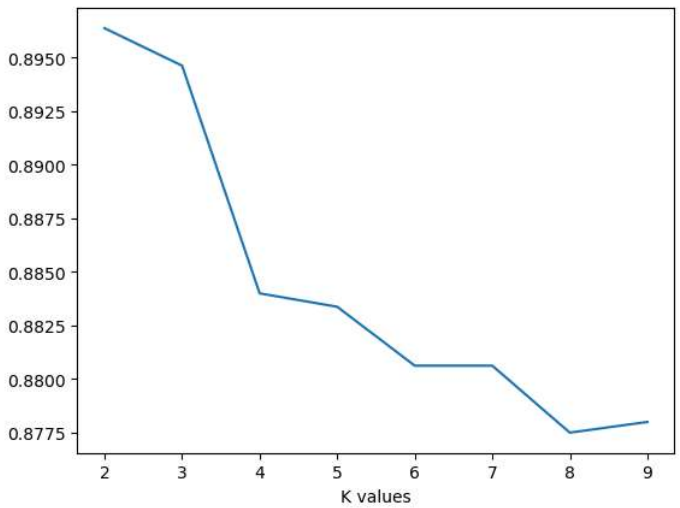
```python
training_acc=[]
testing_acc=[]

for k in range(2,10):
    knn=KNeighborsClassifier(k)
    knn_model=knn.fit(xtrain,ytrain)

    #Predict Training Accuracy
    ytrain_pred=knn_model.predict(xtrain)

    acc=accuracy_score(ytrain,ytrain_pred)
    training_acc.append(acc)

    #Predict Testing Accuracy
    ytest_pred=knn_model.predict(xtest)

    accuracy=accuracy_score(ytest,ytest_pred)
    testing_acc.append(accuracy)
```

Training Accuracy Against K values

In [211…

```python
k = np.arange(2,10)
sns.lineplot(x=k,y=training_acc)
plt.xlabel("K values")
```

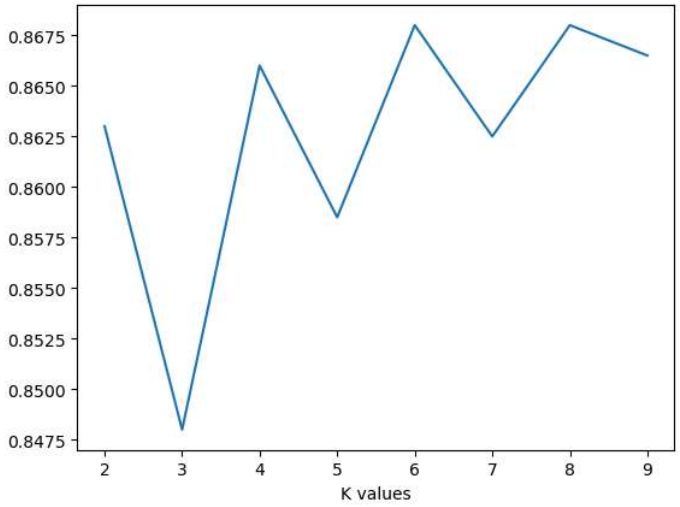
Out[211… Text(0.5, 0, 'K values')



Testing Accuracy Against K values

In [212…
```
k = np.arange(2,10)
sns.lineplot(x=k,y=testing_acc)
plt.xlabel("K values")
```

Out[212…
```
Text(0.5, 0, 'K values')
```



Algorithm 14-SVM(Support vector methods)

In [213…
```
svm=SVC()
svm
```

Out[213…
```
SVC()
```

In [214…
```
svm_model=svm.fit(xtrain,ytrain)
svm_model
```

Out[214…
```
SVC()
```

In [215…
```
ytrain_pred=svm_model.predict(xtrain)
ytest_pred=svm_model.predict(xtest)
```

In [216…
```
print("Model Evaluation for Training Data")
print("-"*100)

# accuracy score
acc= accuracy_score(ytrain,ytrain_pred)
print("Accuracy Score :", acc)
print("-"*100)

# confusion matrix
conf_mat= confusion_matrix(ytrain,ytrain_pred)
print("Confusion Matrix :\n", conf_mat)
print("-"*100)

# classification report
clf_rep= classification_report(ytrain,ytrain_pred)
print("Classification Report :\n", clf_rep)
print("*"*100)

print("\n\nModel evaluation for Testing data")
print("*"*100)

# Accuracy Score
acc = accuracy_score(ytest, ytest_pred)
print("Accuracy Score : ", acc)
print("-"*100)

# Confusion Matrix
conf_mat = confusion_matrix(ytest, ytest_pred)
print("Confusion Matrix : \n",conf_mat)
print("-"*100)

# Classification Report
clf_report = classification_report(ytest, ytest_pred)
print("Classification Report : \n", clf_report)
print("-"*100)
```

```
Model Evaluation for Training Data
--------------------------------------------------------------------------------------------------------
Accuracy Score : 0.8769846230778847
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[7015    0]
 [ 984    0]]
--------------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.88      1.00      0.93      7015
           1       0.00      0.00      0.00       984

    accuracy                           0.88      7999
   macro avg       0.44      0.50      0.47      7999
weighted avg       0.77      0.88      0.82      7999

********************************************************************************************************


Model evaluation for Testing data
********************************************************************************************************
Accuracy Score :  0.8685
--------------------------------------------------------------------------------------------------------
Confusion Matrix :
 [[1737    0]
 [ 263    0]]
--------------------------------------------------------------------------------------------------------
Classification Report :
              precision    recall  f1-score   support

           0       0.87      1.00      0.93      1737
           1       0.00      0.00      0.00       263

    accuracy                           0.87      2000
   macro avg       0.43      0.50      0.46      2000
weighted avg       0.75      0.87      0.81      2000

--------------------------------------------------------------------------------------------------------
```

In [ ]:

Save to pickel

In [120…
```
with open("response_predict.pkl","wb")as f:
    pickle.dump(rf_model,f)
```

In [121…
```
test_data=xtrain.head(2)
test_data
```

Out[121…

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
9295	1	27	1	28	0	0	1	47122.0	152	193.0
6643	1	26	1	36	0	0	1	39164.0	152	37.0

In [122…
```
ytrain.head(2)
```

Out[122…
```
9295    0
6643    0
Name: Response, dtype: int64
```

In [123…
```
with open("response_predict.pkl","rb")as f:
    final_model=pickle.load(f)
```

In [124…
```
final_model.predict(test_data)[0]
```

Out[124…
```
0
```

PREDICT DATA

In [125…
```
with open("response_predict.pkl","rb")as f:
    final_model=pickle.load(f)
def data():
    print("Enter data to predict response if customers will buy health insurance or not")
    Gender=int(input("Enter Gender ['Male':1, 'Female':0] :"))
    Age=int(input("Enter Age :"))
    Driving_License=int(input("Enter Driving_License ['Yes':1, 'No':0] :"))
    Region_Code=float(input("Enter Region :"))
    Previously_Insured=int(input("Enter Previous Insured ['Yes':1, 'No':0] :"))
    Vehicle_Age=int(input("Enter Vehicle Age '< 1 Year':0,'1-2 Year':1,'> 2 Years':2 :"))
    Vehicle_Damage=int(input("Enter Is Vehicle Damage ['Yes':1, 'No':0] :"))
    Annual_Premium=float(input("Enter Anually Premiun in rupees :"))
    Policy_Sales_Channel=float(input("Enter Policy Sales Channel :"))
    Vintage=float(input("Enter Vintage :"))

    testdata={"Gender":[Gender],
    "Age":[Age],
    "Driving_License":[Driving_License],
```

```
    "Region_Code":[Region_Code],
    "Previously_Insured":[Previously_Insured],
    "Vehicle_Age":[Vehicle_Age],
    "Vehicle_Damage":[Vehicle_Damage],
    "Annual_Premium":[Annual_Premium],
    "Policy_Sales_Channel":[Policy_Sales_Channel],
    "Vintage":[Vintage]
            }
    td=pd.DataFrame(testdata)
    display(td)
    return td

input_data=data()

predict=final_model.predict(input_data)

print("\n")
if predict[0]==0:
    print("Response : N0, The customer will not buy health insurance")
else:
    print("Response : Yes, The customer will interested to buy health insurance")
```

```
Enter data to predict response if customers will buy health insurance or not
```

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	2	3	1	22.0	1	2	1	12222222.0	2221.0	22.0

```
Response : N0, The customer will not buy health insurance
```

In [126…

```
%%writefile Health_Cross_Sales_Prediction_app.py
import streamlit as st
import pickle
import pandas as pd

# Load the trained model
@st.cache_resource
def load_model():
    with open("response_predict.pkl", "rb") as f:
        model = pickle.load(f)
    return model

model = load_model()

# Streamlit UI
st.title("🚗 Health Insurance Purchase Prediction")

st.write("Enter customer details to predict whether they will buy health insurance.")

# User Inputs
Gender = st.selectbox("Select Gender", ["Male", "Female"])
Age = st.number_input("Enter Age", min_value=18, max_value=100, step=1)
Driving_License = st.selectbox("Do you have a Driving License?", ["No (0)", "Yes (1)"])
Region_Code = st.number_input("Enter Region Code", min_value=0.0, step=0.1)
Previously_Insured = st.selectbox("Previously Insured?", ["No (0)", "Yes (1)"])
Vehicle_Age = st.selectbox("Vehicle Age", ["< 1 Year", "1-2 Year", "> 2 Years"])
Vehicle_Damage = st.selectbox("Was the Vehicle Damaged?", ["No (0)", "Yes (1)"])
Annual_Premium = st.number_input("Enter Annual Premium (₹)", min_value=1000.0, step=100.0)
Policy_Sales_Channel = st.number_input("Enter Policy Sales Channel", min_value=0.0, step=1.0)
Vintage = st.number_input("Enter Vintage", min_value=0, step=1)

# Convert categorical inputs
Gender = 1 if Gender == "Male" else 0
Driving_License = int(Driving_License.split()[1][1])  # Extracts numeric value
Previously_Insured = int(Previously_Insured.split()[1][1])
Vehicle_Age = {"< 1 Year": 0, "1-2 Year": 1, "> 2 Years": 2}[Vehicle_Age]
Vehicle_Damage = int(Vehicle_Damage.split()[1][1])

# Create DataFrame for Prediction
input_data = pd.DataFrame({
    "Gender": [Gender],
    "Age": [Age],
    "Driving_License": [Driving_License],
    "Region_Code": [Region_Code],
    "Previously_Insured": [Previously_Insured],
    "Vehicle_Age": [Vehicle_Age],
    "Vehicle_Damage": [Vehicle_Damage],
    "Annual_Premium": [Annual_Premium],
    "Policy_Sales_Channel": [Policy_Sales_Channel],
    "Vintage": [Vintage]
})

# Prediction Button
if st.button("Predict 🚀"):
    prediction = model.predict(input_data)
    result = "✅ Will Purchase Insurance" if prediction[0] == 1 else "❌ Will Not Purchase Insurance"
    st.success(f"**Prediction Result:** {result}")
```

```
Overwriting Health_Cross_Sales_Prediction_app.py
```

In [ ]: