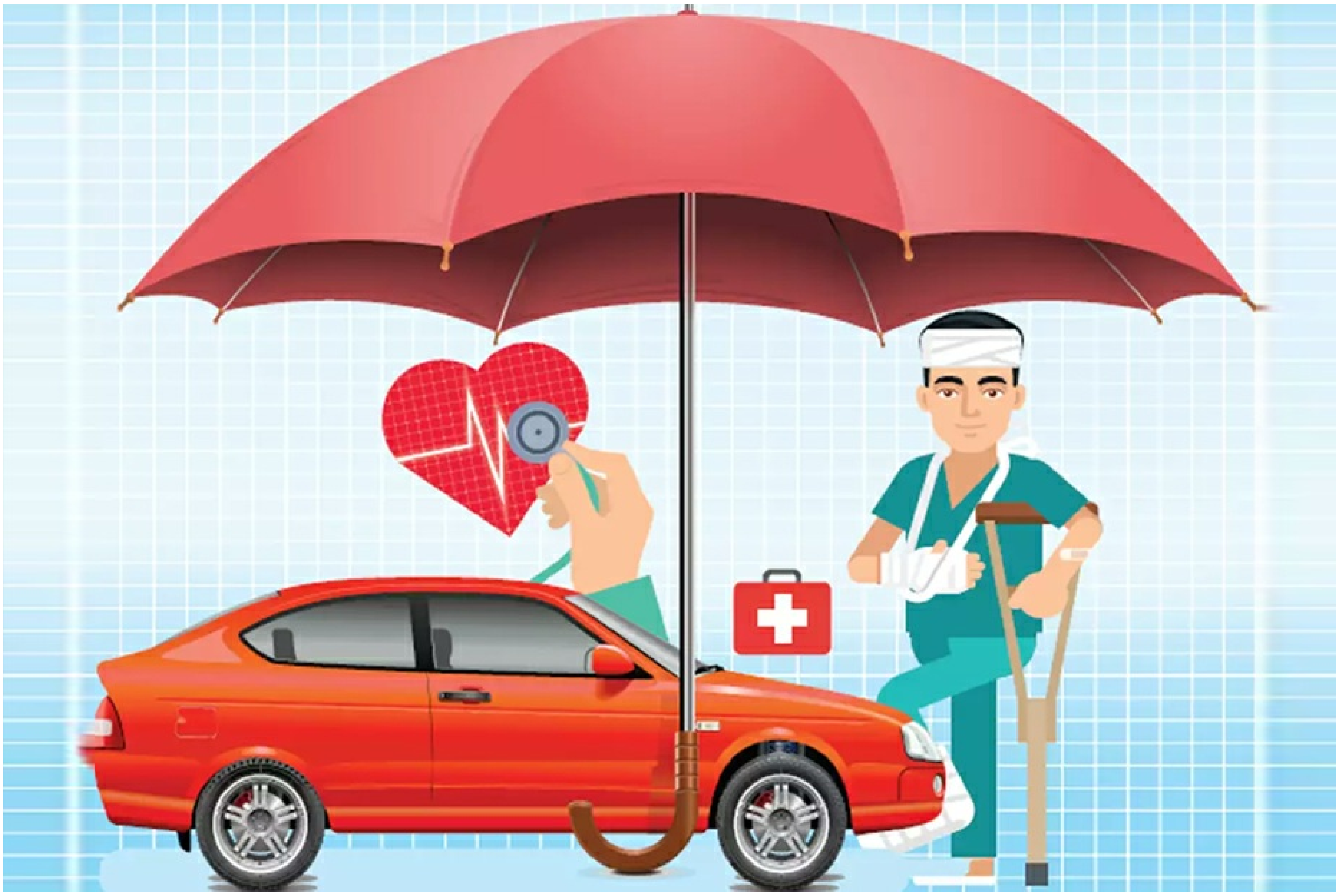
HEALTH INSURANCE CROSS SELL PREDICTION



Our client is an Insurance company that has provided Health Insurance to its customers now they need your help in building a model to predict whether the policyholders (customers) from the past year will also be interested in Vehicle Insurance provided by the company.

An insurance policy is an arrangement by which a company undertakes to provide a guarantee of compensation for specified loss, damage, illness, or death in return for the payment of a specified premium. A premium is a sum of money that the customer needs to pay regularly to an insurance company for this guarantee.

For example, you may pay a premium of Rs. 5000 each year for a health insurance cover of Rs. 200,000/- so that if God forbid, you fall ill and need to be hospitalized in that year, the insurance provider company will bear the cost of hospitalization, etc. for up to Rs. 200,000. Now if you are wondering how can the company bear such high hospitalization costs when it charges a premium of only Rs. 5000/-, that is where the concept of probabilities comes into the picture. For example, like you, there may be 100 customers who would be paying a premium of Rs. 5000 every year, but only a few of them (say 2-3) would get hospitalized that year, and not everyone. This way everyone shares the risk of everyone else.

Just like medical insurance, there is vehicle insurance where every year customer needs to pay a premium of a certain amount to insurance provider company so that in case of unfortunate accident by the vehicle, the insurance provider company will provide compensation (called 'sum assured') to the customer.

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 optimize its business model and revenue.

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.

Data 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

PolicySalesChannel: 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

Import Libraries

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-pytho
n
# For example, here's several helpful packages to load
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
import missingno as msno
from datetime import date
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler, LabelEncoder, StandardScaler, RobustScal
er
from xgboost import XGBClassifier, plot_importance
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files
under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserve
d as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of
the current session
```

```
/kaggle/input/health-insurance-cross-sell-prediction/sample_submission.csv
/kaggle/input/health-insurance-cross-sell-prediction/train.csv
/kaggle/input/health-insurance-cross-sell-prediction/test.csv
```

Import Dataset

In [2]:

```
train=pd.read_csv('/kaggle/input/health-insurance-cross-sell-prediction/train.csv')
test=pd.read_csv('/kaggle/input/health-insurance-cross-sell-prediction/test.csv')
sub = pd.read_csv('/kaggle/input/health-insurance-cross-sell-prediction/sample_submission
.csv')
```

In [3]:

```
train.head()
```

Out[3]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	

In [4]:

```
train.shape
```

Out[4]:

```
(381109, 12)
```

In [5]:

```
train.isnull().sum()
```

Out[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
```

There is no missing data.

In [6]:

```
train.drop("id",axis =1).quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T
```

Out[6]:

	0.00	0.05	0.50	0.95	0.99	1.00
Age	20.0	21.0	36.0	69.0	77.0	85.0
Driving_License	0.0	1.0	1.0	1.0	1.0	1.0
Region_Code	0.0	5.0	28.0	47.0	50.0	52.0
Previously_Insured	0.0	0.0	0.0	1.0	1.0	1.0

Annual_Premium	268000	268006	3160950	5510696	7296399	54016500
Policy_Sales_Channel	1.0	26.0	133.0	160.0	160.0	163.0
Vintage	10.0	24.0	154.0	285.0	297.0	299.0
Response	0.0	0.0	0.0	1.0	1.0	1.0

Check Columns (Categorical or Numeric)

In [7]:

```
def grab_col_names(dataframe, cat_th=10, car_th=20):
    """

    It gives the names of categorical, numerical, and categorical but cardinal variables
in the data set.
    Note: Categorical variables with numerical appearance are also included in categorica
l variables.

    Parameters
    ------
        dataframe: dataframe
        cat_th: int, optional
                the class threshold for numeric but categorical variables
        car_th: int, optinal
                the class threshold for categorical but cardinal variables

    Returns
    ------
        cat_cols: list
                Categorical Variables List
        num_cols: list
                Numeric Variables List
        cat_but_car: list
                Categorical but cardinal variables list

    Examples
    ------
        import seaborn as sns
        df = sns.load_dataset("iris")
        print(grab_col_names(df))


    Notes
    ------
        cat_cols + num_cols + cat_but_car = total number of variables
        num_but_cat is in cat_cols

    """

    # cat_cols, cat_but_car
    cat_cols = [col for col in dataframe.columns if dataframe[col].dtypes == "O"]
    num_but_cat = [col for col in dataframe.columns if dataframe[col].nunique() < cat_th
and
                   dataframe[col].dtypes != "O"]
    cat_but_car = [col for col in dataframe.columns if dataframe[col].nunique() > car_th
and
                   dataframe[col].dtypes == "O"]
    cat_cols = cat_cols + num_but_cat
    cat_cols = [col for col in cat_cols if col not in cat_but_car]

    # num_cols
    num_cols = [col for col in dataframe.columns if dataframe[col].dtypes != "O"]
    num_cols = [col for col in num_cols if col not in num_but_cat]

    print(f"Observations: {dataframe.shape[0]}")
    print(f"Variables: {dataframe.shape[1]}")
    print(f'cat_cols: {len(cat_cols)}')
    print(f'num_cols: {len(num_cols)}')
    print(f'cat_but_car: {len(cat_but_car)}')
```

```
    print(f'num_but_cat: {len(num_but_cat)}')
    return cat_cols, num_cols, cat_but_car

cat_cols, num_cols, cat_but_car = grab_col_names(train)
```

```
Observations: 381109
Variables: 12
cat_cols: 6
num_cols: 6
cat_but_car: 0
num_but_cat: 3
```

In [8]:

```
cat_cols
```

Out[8]:

```
['Gender',
 'Vehicle_Age',
 'Vehicle_Damage',
 'Driving_License',
 'Previously_Insured',
 'Response']
```

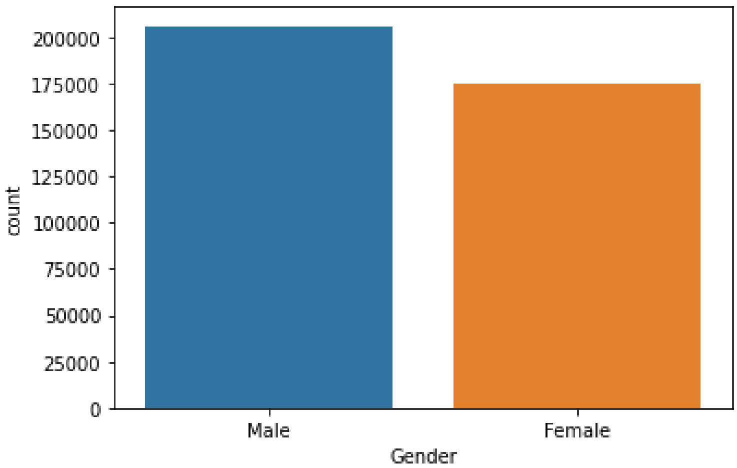
In [9]:

```
num_cols = [col for col in num_cols if "id" not in col]
num_cols = [col for col in num_cols if "Policy_Sales_Channel" not in col]
```

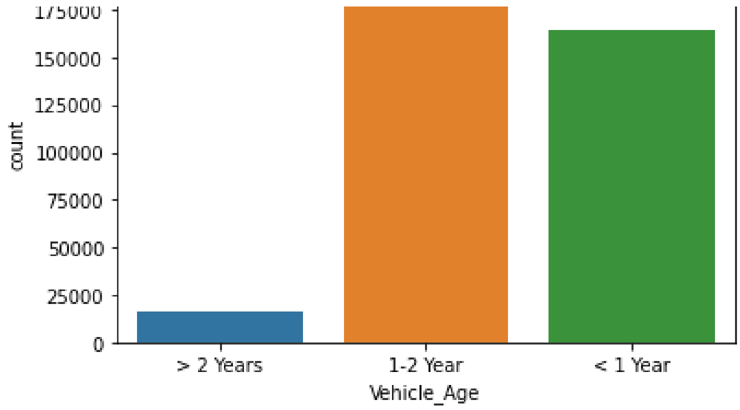
In [10]:

```
# SUMMARY CATEGORICAL COLUMNS
def cat_summary(dataframe, col_name, plot=False):
    print(pd.DataFrame({col_name: dataframe[col_name].value_counts(),
                        "Ratio": 100 * dataframe[col_name].value_counts() / len(dataframe)}))
    if plot:
        sns.countplot(x=dataframe[col_name], data=dataframe)
        plt.show()


for i in cat_cols:
    cat_summary(train, i, plot=True)
```

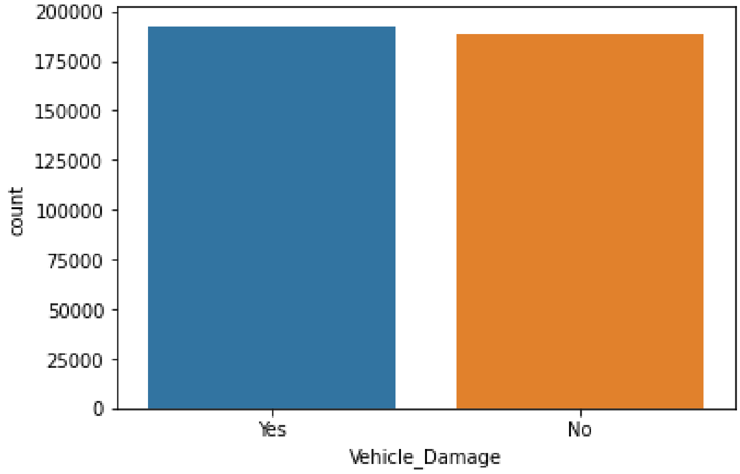
```
        Gender     Ratio
Male    206089  54.07613
Female  175020  45.92387
```



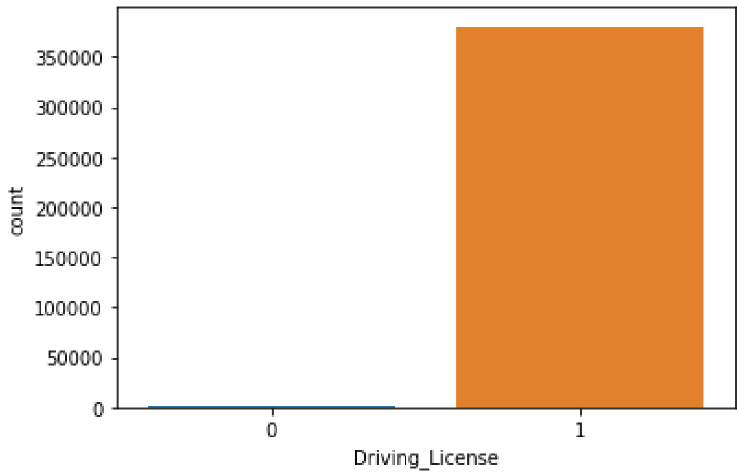

```
           Vehicle_Age      Ratio
1-2 Year        200316  52.561341
< 1 Year        164786  43.238549
> 2 Years        16007   4.200111
```



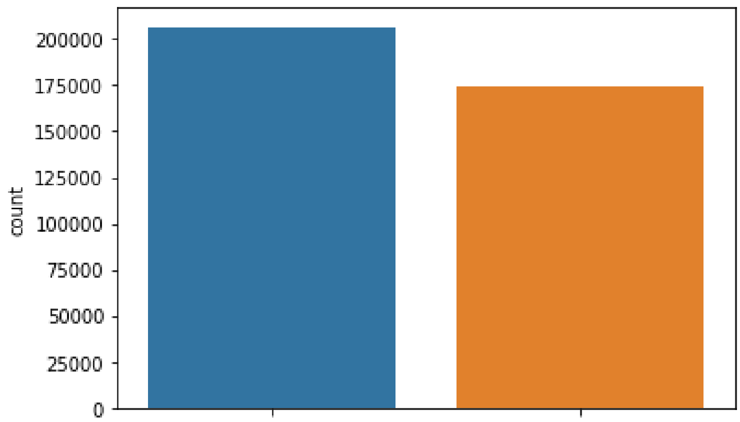
	Vehicle_Damage	Ratio
Yes	192413	50.487656
No	188696	49.512344



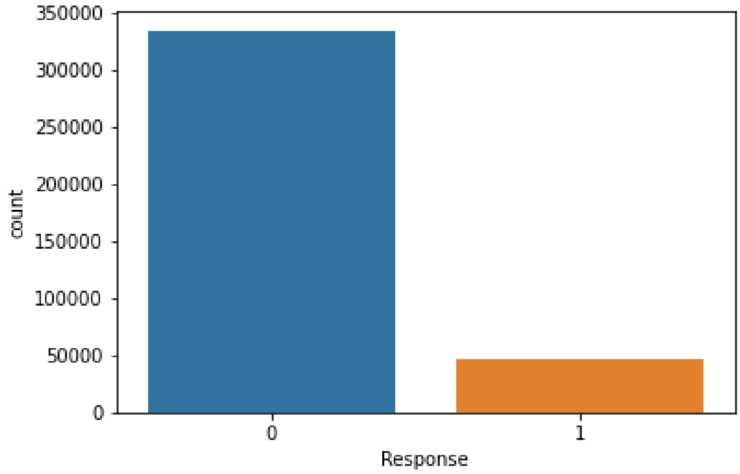
	Driving_License	Ratio
1	380297	99.786938
0	812	0.213062



	Previously_Insured	Ratio
0	206481	54.178988
1	174628	45.821012



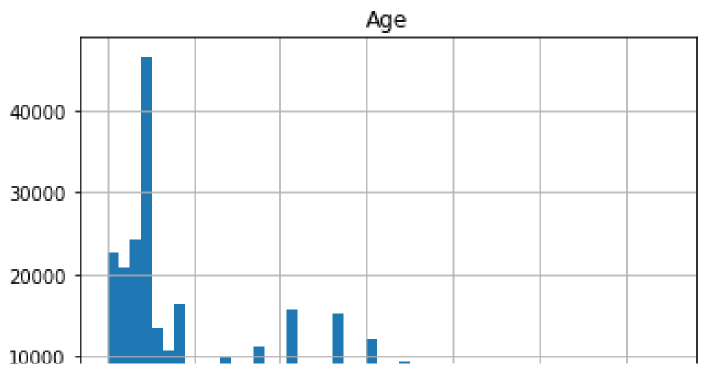

```
   Response      Ratio
0    334399  87.743664
1     46710  12.256336
```

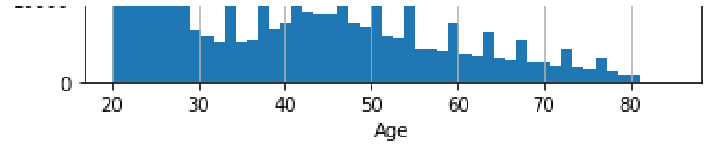


In [11]:

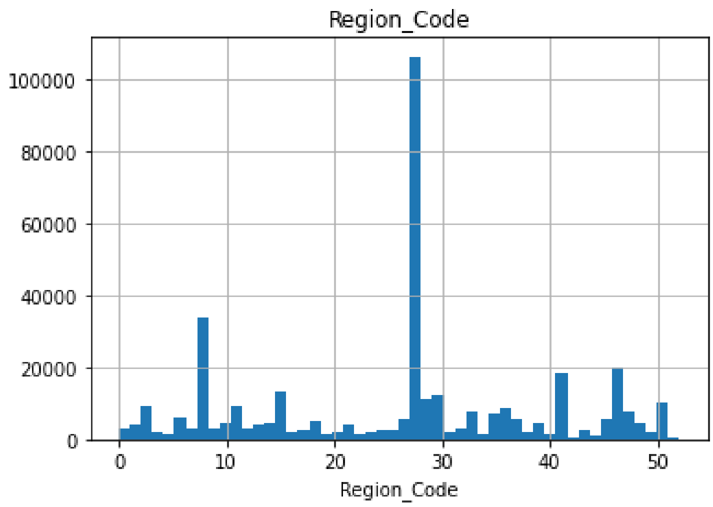
```python
# SUMMARY NUMERIC COLUMNS
def num_summary(dataframe, numerical_col, plot=False):
    quantiles = [0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.95, 0.99]
    print(dataframe[numerical_col].describe(quantiles).T)

    if plot:
        dataframe[numerical_col].hist(bins=50)
        plt.xlabel(numerical_col)
        plt.title(numerical_col)
        plt.show()

    print("#####################################")

for col in num_cols:
    num_summary(train, col, plot=True)
```

```
count    381109.000000
mean         38.822584
std          15.511611
min          20.000000
5%           21.000000
10%          22.000000
20%          24.000000
30%          25.000000
40%          29.000000
50%          36.000000
60%          42.000000
70%          47.000000
80%          53.000000
90%          62.000000
95%          69.000000
99%          77.000000
max          85.000000
Name: Age, dtype: float64
```

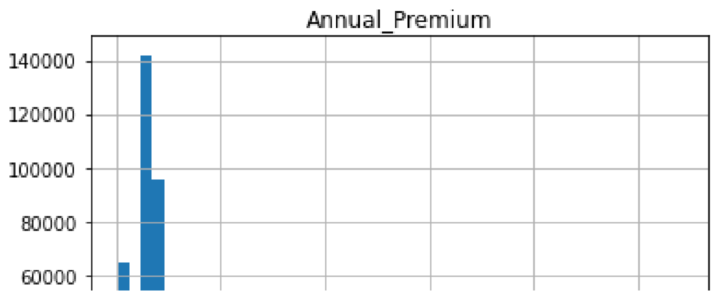


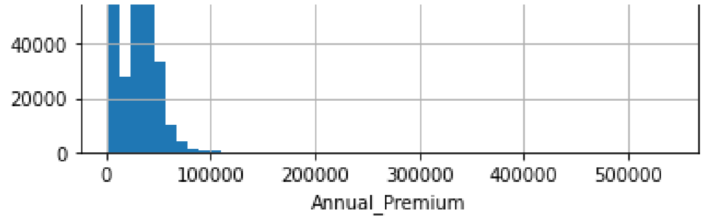



```
####################################
count    381109.000000
mean         26.388807
std          13.229888
min           0.000000
5%            5.000000
10%           8.000000
20%          11.000000
30%          18.000000
40%          28.000000
50%          28.000000
60%          28.000000
70%          31.000000
80%          39.000000
90%          46.000000
95%          47.000000
99%          50.000000
max          52.000000
Name: Region_Code, dtype: float64
```

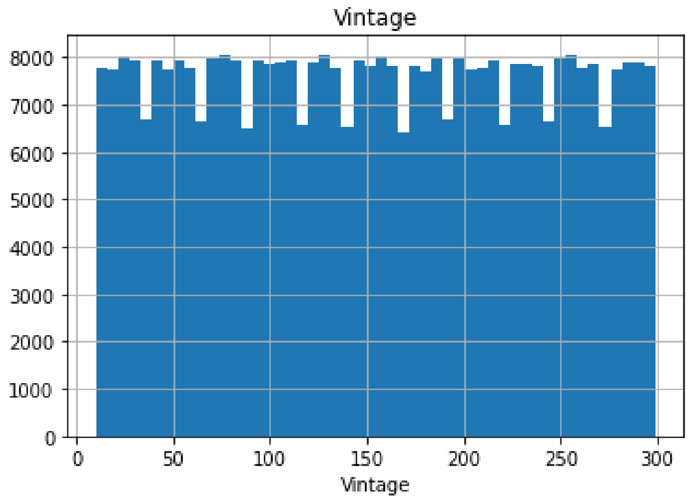



```
####################################
count    381109.000000
mean      30564.389581
std       17213.155057
min        2630.000000
5%         2630.000000
10%        2630.000000
20%       21583.600000
30%       26238.000000
40%       29082.000000
50%       31669.000000
60%       34406.000000
70%       37548.000000
80%       41711.000000
90%       48431.000000
95%       55176.000000
99%       72963.000000
max      540165.000000
Name: Annual_Premium, dtype: float64
```






```
#####################################
count    381109.000000
mean        154.347397
std          83.671304
min          10.000000
5%           24.000000
10%          38.000000
20%          68.000000
30%          96.000000
40%         125.000000
50%         154.000000
60%         183.000000
70%         212.000000
80%         241.000000
90%         270.000000
95%         285.000000
99%         297.000000
max         299.000000
Name: Vintage, dtype: float64
```




```
#####################################
```

In [12]:

```
def target_summary_with_num(dataframe, target, numerical_col):
    print(dataframe.groupby(target).agg({numerical_col: "mean"}), end="\n\n\n")
```

In [13]:

```
for col in num_cols:
    target_summary_with_num(train, "Response", col)
```

```
                Age
Response
0         38.178227
1         43.435560


          Region_Code
Response
0           26.336544
1           26.762963


          Annual_Premium
Response
0           30419.160276
```

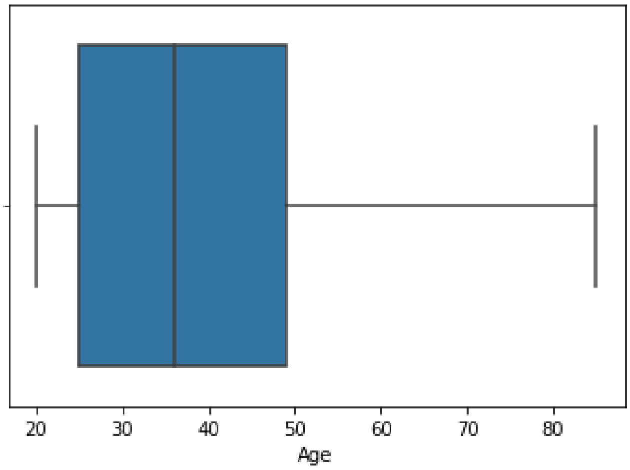
```
1          31604.092742


              Vintage
Response
0          154.380243
1          154.112246
```

In [14]:

```
sns.boxplot(x=train["Age"])
```

Out[14]:

<AxesSubplot:xlabel='Age'>

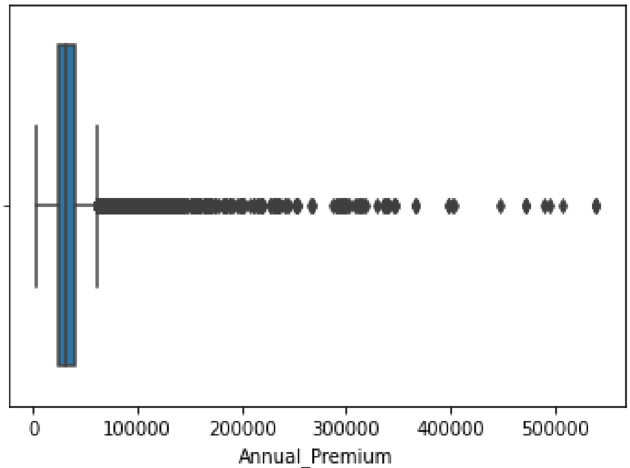


In [15]:

```
sns.boxplot(x=train["Annual_Premium"])
```

Out[15]:

<AxesSubplot:xlabel='Annual_Premium'>

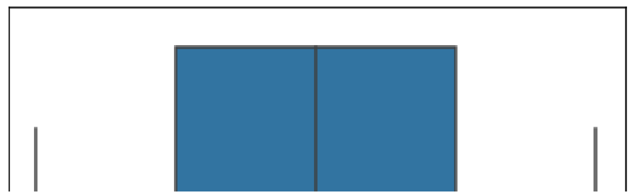


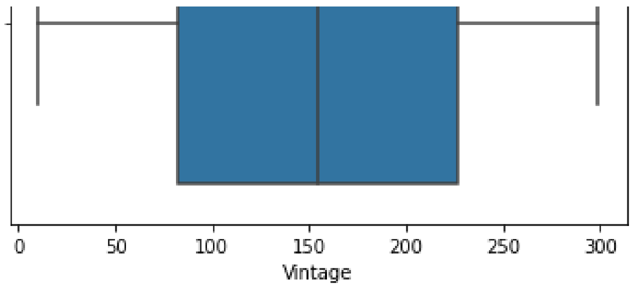
In [16]:

```
sns.boxplot(x=train["Vintage"])
```

Out[16]:

<AxesSubplot:xlabel='Vintage'>





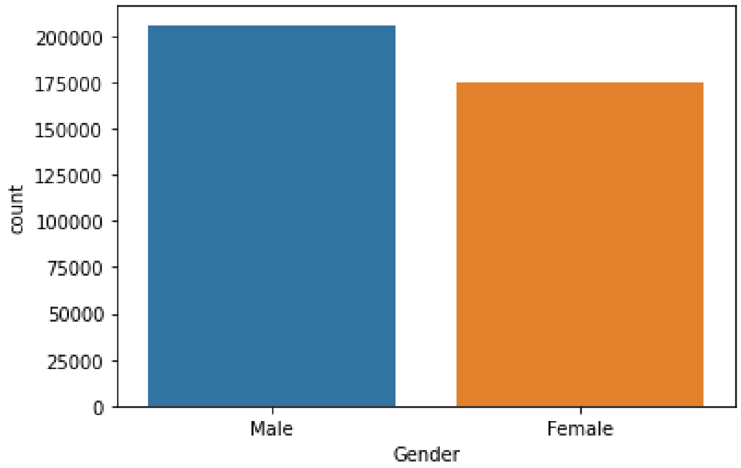
Gender-Response Visualization

In [17]:

```
sns.countplot(train.Gender)
```

Out[17]:

<AxesSubplot:xlabel='Gender', ylabel='count'>

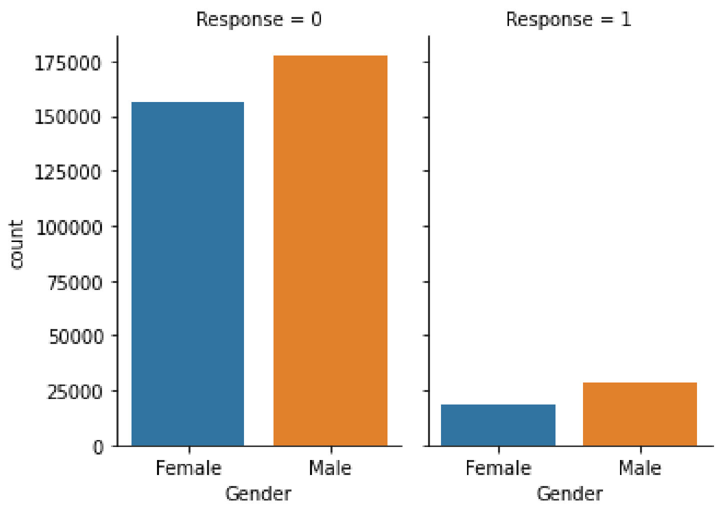


In [18]:

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

In [19]:

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



Driving license-Gender Visualization

In [20]:

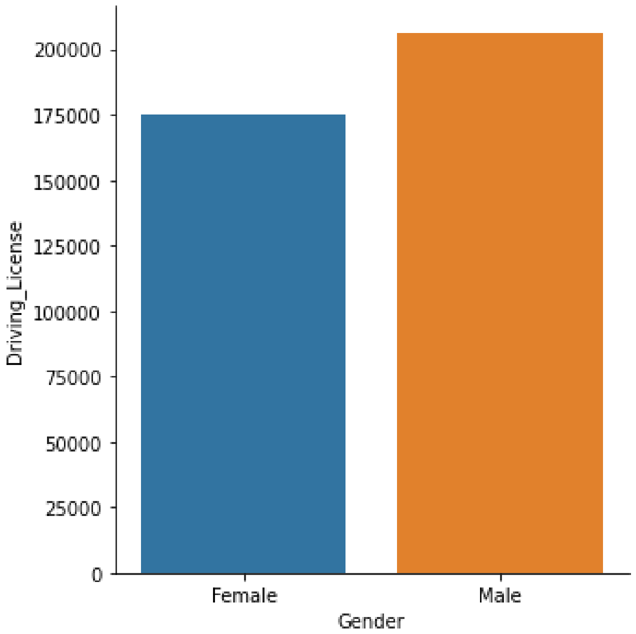
```
df=train.groupby(['Gender'])['Driving_License'].count().to_frame().reset_index()
df
```

Out[20]:

	Gender	Driving_License
0	Female	175020
1	Male	206089

In [21]:

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



Response-Vehicle age Visualization

In [22]:

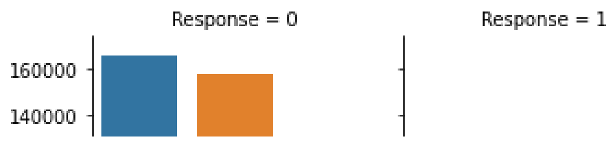
```
df=train.groupby(['Vehicle_Age','Response'])['id'].count().to_frame().rename(columns={'i
d':'count'}).reset_index()
df
```

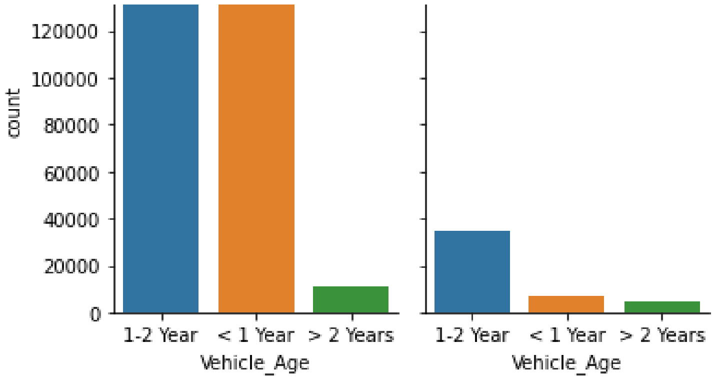
Out[22]:

	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

In [23]:

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





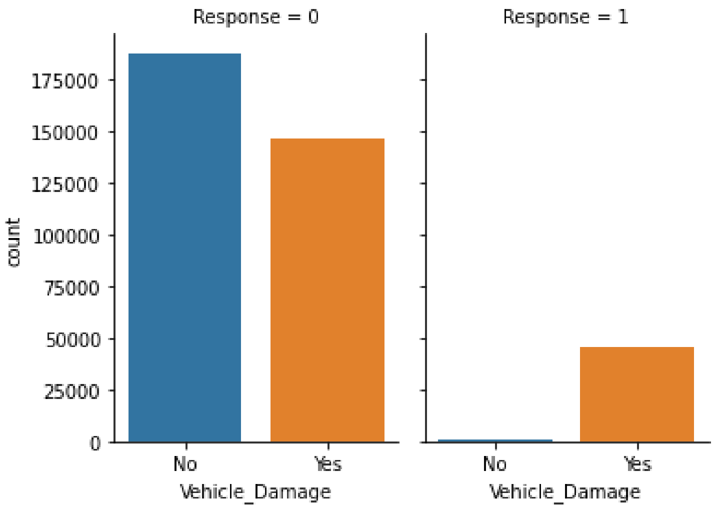
Damage Vehicle-Response Visualization

In [24]:

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

In [25]:

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



DATA PREPROCESSING

OUTLIERS

In [26]:

```
def outlier_thresholds(dataframe, col_name, q1=0.05, q3=0.95):
    quartile1 = dataframe[col_name].quantile(q1)
    quartile3 = dataframe[col_name].quantile(q3)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 3 * interquantile_range
    low_limit = quartile1 - 3 * interquantile_range
    return low_limit, up_limit


def check_outlier(dataframe, col_name):
    low_limit, up_limit = outlier_thresholds(dataframe, col_name)
    if dataframe[(dataframe[col_name] > up_limit) | (dataframe[col_name] < low_limit)].a
ny(axis=None):
        return True
    else:
        return False
```

In [27]:

```
for col in num_cols:
    print(f"{col} : {check_outlier(train,col)}")
```

```
Age : False
Region_Code : False
Annual_Premium : True
Vintage : False
```

In [28]:

```
def grab_outliers(dataframe, col_name, index=False):
    low, up = outlier_thresholds(dataframe, col_name)

    if dataframe[((dataframe[col_name] < low) | (dataframe[col_name] > up))].shape[0] >
10:
        print(dataframe[((dataframe[col_name] < low) | (dataframe[col_name] > up))].head
())
    else:
        print(dataframe[((dataframe[col_name] < low) | (dataframe[col_name] > up))])

    if index:
        outlier_index = dataframe[((dataframe[col_name] < low) | (dataframe[col_name] >
up))].index
        return outlier_index
```

In [29]:

```
for col in num_cols:
    col, grab_outliers(train, col)
```

```
Empty DataFrame
Columns: [id, Gender, Age, Driving_License, Region_Code, Previously_Insured, Vehicle_Age,
Vehicle_Damage, Annual_Premium, Policy_Sales_Channel, Vintage, Response]
Index: []
Empty DataFrame
Columns: [id, Gender, Age, Driving_License, Region_Code, Previously_Insured, Vehicle_Age,
Vehicle_Damage, Annual_Premium, Policy_Sales_Channel, Vintage, Response]
Index: []
          id  Gender  Age  Driving_License  Region_Code  Previously_Insured  \
1412    1413  Female   41                1         28.0                   0
11319  11320  Female   50                1         46.0                   1
13426  13427  Female   47                1         28.0                   0
15024  15025  Female   32                1         28.0                   0
25532  25533    Male   50                1         28.0                   0

      Vehicle_Age Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  \
1412     1-2 Year            Yes        267698.0                 124.0
11319    1-2 Year             No        508073.0                  26.0
13426    1-2 Year            Yes        301762.0                 124.0
15024    1-2 Year            Yes        315565.0                 155.0
25532    1-2 Year            Yes        229935.0                 122.0

       Vintage  Response
1412        63         1
11319      192         0
13426       22         0
15024      150         0
25532       64         1
Empty DataFrame
Columns: [id, Gender, Age, Driving_License, Region_Code, Previously_Insured, Vehicle_Age,
Vehicle_Damage, Annual_Premium, Policy_Sales_Channel, Vintage, Response]
Index: []
```

In [30]:

```
def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit
```

In [31]:

```
for col in num_cols:
    replace_with_thresholds(train, col)
```

In [32]:

```
# CHECK OUTLIERS AGAIN
for col in num_cols:
    print(f"{col} : {check_outlier(train,col)}")
```

```
Age : False
Region_Code : False
Annual_Premium : False
Vintage : False
```

In [33]:

```
train.drop("id",axis =1).quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T
```

Out[33]:

	0.00	0.05	0.50	0.95	0.99	1.00
Age	20.0	21.0	36.0	69.0	77.0	85.0
Driving_License	0.0	1.0	1.0	1.0	1.0	1.0
Region_Code	0.0	5.0	28.0	47.0	50.0	52.0
Previously_Insured	0.0	0.0	0.0	1.0	1.0	1.0
Annual_Premium	2630.0	2630.0	31669.0	55176.0	72963.0	212814.0
Policy_Sales_Channel	1.0	26.0	133.0	160.0	160.0	163.0
Vintage	10.0	24.0	154.0	285.0	297.0	299.0
Response	0.0	0.0	0.0	1.0	1.0	1.0

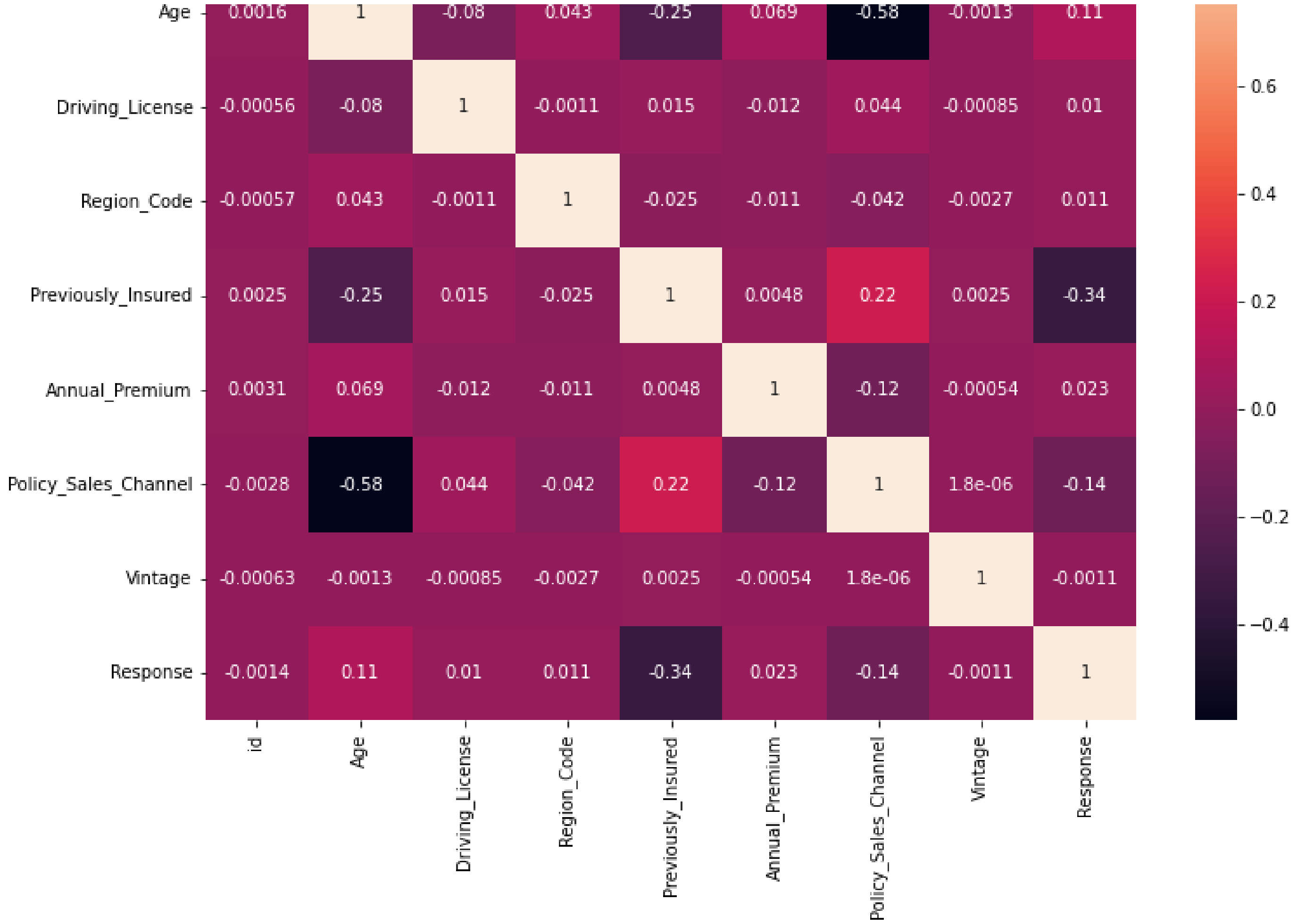
CORRELATION ANALYSIS

In [34]:

```
df.corrwith(train["Response"]).sort_values(ascending=False)
corr_df = train.corr()
plt.figure(figsize=(12, 9))
sns.heatmap(corr_df, annot=True, xticklabels=corr_df.columns, yticklabels=corr_df.column
s)

corr_df = corr_df.corr().unstack().sort_values().drop_duplicates()
corr_df = pd.DataFrame(corr_df, columns=["corr"])
corr_df.index.names = ['1', '2']
corr_df = corr_df.reset_index()
corr_df.sort_values(by="corr", ascending=True).head(30)

high_corr = corr_df[(corr_df["corr"] >= 0.70) | (corr_df["corr"] <= -0.70)]
high_corr
```

Out[34]:

	1	2	corr
0	Policy_Sales_Channel	Age	-0.890519
36	id	id	1.000000



FEATURE ENGINEERING

In [35]:

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

In [36]:

```
train=pd.get_dummies(train,drop_first=True)
```

In [37]:

```
train
```

Out[37]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vint
0	1	1	44	1	28.0	0	40454.0	26.0	
1	2	1	76	1	3.0	0	33536.0	26.0	
2	3	1	47	1	28.0	0	38294.0	26.0	
3	4	1	21	1	11.0	1	28619.0	152.0	
4	5	0	29	1	41.0	1	27496.0	152.0	
...	...	...	...	...	...	...	...	...	
381104	381105	1	74	1	26.0	1	30170.0	26.0	
381105	381106	1	30	1	37.0	1	40016.0	152.0	
381106	381107	1	21	1	30.0	1	35118.0	160.0	
381107	381108	0	68	1	14.0	0	44617.0	124.0	
381108	381109	1	46	1	29.0	0	41777.0	26.0	

381109 rows × 13 columns

In [38]:

```
train=train.rename(columns={"Vehicle_Age_< 1 Year": "Vehicle_Age_lt_1_Year", "Vehicle_Ag
e_> 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')
```

In [39]:

```
train["premium_age_ratio"] = train["Annual_Premium"]/train["Age"]
```

In [40]:

```
train["premium_vintage_ratio"] = train["Annual_Premium"]/train["Vintage"]
```

In [41]:

```
train
```

Out[41]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vint
0	1	1	44	1	28.0	0	40454.0	26.0	
1	2	1	76	1	3.0	0	33536.0	26.0	
2	3	1	47	1	28.0	0	38294.0	26.0	
3	4	1	21	1	11.0	1	28619.0	152.0	
4	5	0	29	1	41.0	1	27496.0	152.0	
...	...	...	...	...	...	...	...	...	
381104	381105	1	74	1	26.0	1	30170.0	26.0	
381105	381106	1	30	1	37.0	1	40016.0	152.0	
381106	381107	1	21	1	30.0	1	35118.0	160.0	
381107	381108	0	68	1	14.0	0	44617.0	124.0	
381108	381109	1	46	1	29.0	0	41777.0	26.0	

381109 rows × 15 columns

In [42]:

```
num_feat = ['Age','Vintage','premium_age_ratio','premium_vintage_ratio']
ss = StandardScaler()
train[num_feat] = ss.fit_transform(train[num_feat])
```

In [43]:

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

In [44]:

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

In [45]:

```
cat_feat = ['Gender', 'Driving_License', 'Previously_Insured', 'Vehicle_Age_lt_1_Year','V
ehicle_Age_gt_2_Years','Vehicle_Damage_Yes']
for column in cat_feat:
    train[column] = train[column].astype('str')
```

Applying the same processes to the Test Data

In [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')
test["premium_age_ratio"] = test["Annual_Premium"]/test["Age"]
test["premium_vintage_ratio"] = test["Annual_Premium"]/test["Vintage"]
test=test.drop('id',axis=1)
```

In [47]:

```
ss = StandardScaler()
test[num_feat] = ss.fit_transform(test[num_feat])


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

In [48]:

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

Train-Test Split

In [49]:

```
training_data, testing_data = train.drop('Response', axis=1),train['Response']
```

In [50]:

```
x_train, x_test, y_train, y_test = train_test_split(training_data,testing_data, test_siz
e=0.2, random_state=42)
```

In [51]:

```
x_train
```

Out[51]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage
332803	0	0.011438	1	15.0	0	0.239200	55.0	0.86831
116248	1	-0.053030	1	11.0	0	0.097096	26.0	1.49809
255005	1	-1.084517	1	30.0	1	0.203098	152.0	0.13926
317474	0	-1.020049	1	41.0	1	0.126090	151.0	1.46588
344212	1	1.107392	1	48.0	0	0.000000	154.0	0.00780
...	...	...	...	...	...	...	...	..
259178	0	-0.955581	1	36.0	1	0.094893	152.0	1.58540
365838	1	1.107392	1	35.0	0	0.183920	124.0	1.71687
131932	0	-1.084517	1	2.0	0	0.077204	152.0	0.93637

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage
146867	1	0.333777	1	32.0	1	0.000000	156.0	1.23516
121958	0	-0.762177	1	37.0	0	0.105198	152.0	0.32684

304887 rows × 13 columns

In [52]:

```
x_test
```

Out[52]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage
200222	0	-1.148985	1	3.0	1	0.084583	160.0	0.98417
49766	1	1.042924	1	15.0	0	0.165893	26.0	0.62563
172201	0	0.140374	1	3.0	0	0.000000	26.0	0.76905
160713	0	-0.826645	1	11.0	0	0.000000	151.0	1.35832
53272	1	0.785053	1	40.0	0	0.149279	124.0	1.32247
...	...	...	...	...	...	...	...	..
258403	1	-1.020049	1	15.0	0	0.098894	152.0	0.21097
234155	0	-1.213453	1	15.0	1	0.140781	160.0	0.94002
24476	0	-0.697709	1	8.0	1	0.223328	152.0	0.60537
60423	0	-1.148985	1	3.0	1	0.142599	160.0	1.19930
185839	1	-1.084517	1	39.0	1	0.167292	152.0	0.74514

76222 rows × 13 columns

In [53]:

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

In [54]:

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

XGBOOST

In [55]:

```
model_xgb = XGBClassifier()
model_xgb.fit(x_train, y_train,eval_metric='mlogloss')

pred_xgb = model_xgb.predict(x_test)
predictions_xgb = [round(value) for value in pred_xgb]
accuracy_xgb = accuracy_score(y_test, predictions_xgb)
print("Accuracy: %.2f%%" % (accuracy_xgb * 100.0))
```

```
Accuracy: 87.49%
```

In [56]:

```
#params = {
    #"max_depth" : range(2,10,1),
    #"n_estimators" : range(60,220,40),
    #"learning_rate" : [0.1,0.01,0.05]}
```

In [57]:

```
#grid_search_xgb = GridSearchCV(model_xgb,
                                #param_grid = params,
                                #scoring = 'roc_auc',
                                #n_jobs = 10,
                                #cv = 10,
                                #verbose = True)
```

In [58]:

```
#grid_search_xgb.fit(x_train,y_train)
```

In [59]:

```
#grid_search_xgb.best_estimator_
```

In [60]:

```
model_xgb_tuned = XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
                                colsample_bynode=1, colsample_bytree=1, enable_categoric
al=False,
                                gamma=0, gpu_id=-1, importance_type=None,
                                interaction_constraints='', learning_rate=0.1, max_delta
_step=0,
                                max_depth=5, min_child_weight=1,eval_metric='mlogloss',
                                monotone_constraints='()', n_estimators=180, n_jobs=12,
                                num_parallel_tree=1, predictor='auto', random_state=0,
                                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample
=1,
                                tree_method='exact', validate_parameters=1, verbosity=No
ne)


model_xgb_tuned.fit(x_train, y_train)
pred_xgb_tuned = model_xgb_tuned.predict(x_test)
predictions_xgb_tuned = [round(value) for value in pred_xgb_tuned]
accuracy_xgb_tuned = accuracy_score(y_test, predictions_xgb_tuned)
print("Accuracy: %.2f%%" % (accuracy_xgb_tuned * 100.0))
```

```
Accuracy: 87.50%
```

Let use all values in the train set to estimate Response values in test data. Then we'll compare prediction Responses according to the Submission data. Because the sample_submission.csv has actual Responses values. We can see how our model is good.

In [61]:

```
X_train, Y_train = train.drop("Response",axis = 1), train["Response"]
```

In [62]:

```
X_train
```

Out[62]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage
0	1	0.333777	1	28.0	0	0.179957	26.0	0.74879
1	1	2.396751	1	3.0	0	0.147043	26.0	0.34244

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintag
2	1	0.527181	1	28.0	0	0.169680	26.0	1.52199
3	1	-1.148985	1	11.0	1	0.123649	152.0	0.58147
4	0	-0.633242	1	41.0	1	0.118306	152.0	1.37858
...	...	...	...	...	...	...	...	..
381104	1	2.267815	1	26.0	1	0.131028	26.0	0.79295
381105	1	-0.568774	1	37.0	1	0.177873	152.0	0.27903
381106	1	-1.148985	1	30.0	1	0.154569	160.0	0.07950
381107	0	1.881007	1	14.0	0	0.199763	124.0	0.96027
381108	1	0.462713	1	29.0	0	0.186251	26.0	0.98782

381109 rows × 13 columns

In [63]:

```
Y_train
```

Out[63]:

```
0         1
1         0
2         1
3         0
4         0
         ..
381104    0
381105    0
381106    0
381107    0
381108    0
Name: Response, Length: 381109, dtype: int64
```

In [64]:

```
test
```

Out[64]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintag
0	1	-0.890089	1	11.0	1	0.070633	152.0	1.21105
1	1	0.079795	1	28.0	0	0.066321	7.0	0.51778
2	1	0.532408	1	28.0	0	0.079717	124.0	0.53407
3	1	-0.954748	1	27.0	1	0.073978	152.0	0.39064
4	1	-0.760771	1	28.0	1	0.120293	152.0	1.70546
...	...	...	...	...	...	...	...	..
127032	0	-0.825430	1	37.0	1	0.060154	152.0	1.17519
127033	0	-0.049523	1	28.0	0	0.055538	122.0	0.12767
127034	1	-1.148725	1	46.0	1	0.057885	152.0	0.96004

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage
127035	1	2.084224	1	28.0	1	0.128341	26.0	1.32297
127036	1	0.144454	1	29.0	1	0.053891	124.0	0.91657

127037 rows × 13 columns

In [65]:

```
for column in cat_feat:
    X_train[column] = X_train[column].astype('int')
    test[column] = test[column].astype('int')
```

In [66]:

```
for column in num_feat:
    X_train[column] = X_train[column].astype('int')
    test[column] = test[column].astype('int')
```

In [67]:

```
from xgboost import XGBClassifier, plot_importance

# fit model no training data
model2 = XGBClassifier(eval_metric='mlogloss')
model2.fit(X_train, Y_train)

y_pred1 = model2.predict(test)
predictions2 = [round(value) for value in y_pred1]
```

In [68]:

```
test_2 = pd.DataFrame()
```

In [69]:

```
test_2['Response'] = sub['Response']
test_2["Pred_Response"] = predictions2
```

In [70]:

```
test_2
```

Out[70]:

	Response	Pred_Response
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
127032	0	0
127033	0	0
127034	0	0
127035	0	0
127036	0	0

127037 rows × 2 columns

In [71]:

```
accuracy_2 = accuracy_score(test_2['Response'].values, test_2['Pred_Response'].values)
```

```
accuracy_2 = accuracy_score(test_2['Response'].values, test_2['Pred_Response'].values)
print("Accuracy: %.2f%%" % (accuracy_2 * 100.0))
```

```
Accuracy: 99.68%
```