Peer-graded Assignment: Course Project - Peer Review

by - Aarohan Verma

Dataset Context

Insurance companies that sell life, health, and property and casualty insurance are using machine learning (ML) to drive improvements in customer service, fraud detection, and operational efficiency. The data provided by an Insurance company which is not excluded from other companies to getting advantage of ML. This company provides Health Insurance to its customers. We can build a model to predict whether the policyholders (customers) from 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 company bear such high hospitalization cost when it charges a premium of only Rs. 5000/-, that is where the concept of probabilities comes in 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 certain amount to insurance provider company so that in case of unfortunate accident by the vehicle, the insurance provider company will provide a compensation (called 'sum assured') to the customer.

Content

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.

We have information about:

Demographics (gender, age, region code type), Vehicles (Vehicle Age, Damage), Policy (Premium, sourcing channel) etc.

In [1]:
```
#importing necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import ExtraTreesClassifier
from imblearn.over_sampling import SMOTE
```

In [2]:
```
#importing the csv file as a pandas dataframe
train=pd.read_csv('train.csv')
test=pd.read_csv('test.csv') #should be treated as future data
```

Note:EDA and Feature Engineering should be performed on training dataset only.

In [4]:
```
train.head()
```

Out[4]:

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

In [5]:
```
#Number of rows
print(train.shape[0])
```

```python
#Number of columns
print(train.shape[1])

#Column Names
print(train.columns.tolist())

#DataTypes
print(train.dtypes)
```

```
381109
12
['id', 'Gender', 'Age', 'Driving_License', 'Region_Code', 'Previously_Insured', 'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premium', 'Policy_Sales_Channel', 'Vintage', 'Response']
id                        int64
Gender                   object
Age                       int64
Driving_License           int64
Region_Code             float64
Previously_Insured       object
Vehicle_Age              object
Vehicle_Damage           object
Annual_Premium          float64
Policy_Sales_Channel    float64
Vintage                   int64
Response                  int64
dtype: object
```

In [6]:

```python
#Statistical Figures about the Dataset
train.describe()
```

Out[6]:

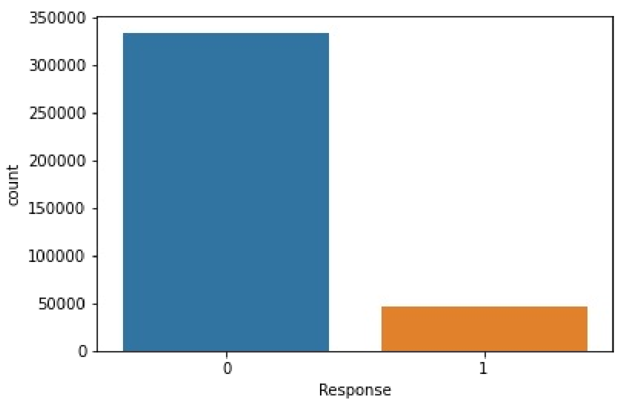
	id	Age	Driving_License	Region_Code	Annual_Premium	Policy_Sales_Channel	Vintage	Response
count	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000
mean	190555.000000	38.822584	0.997869	26.388807	30564.389581	112.034295	154.347397	0.122563
std	110016.836208	15.511611	0.046110	13.229888	17213.155057	54.203995	83.671304	0.327936
min	1.000000	20.000000	0.000000	0.000000	2630.000000	1.000000	10.000000	0.000000
25%	95278.000000	25.000000	1.000000	15.000000	24405.000000	29.000000	82.000000	0.000000
50%	190555.000000	36.000000	1.000000	28.000000	31669.000000	133.000000	154.000000	0.000000
75%	285832.000000	49.000000	1.000000	35.000000	39400.000000	152.000000	227.000000	0.000000
max	381109.000000	85.000000	1.000000	52.000000	540165.000000	163.000000	299.000000	1.000000

In [7]:

```python
#Target Class Ratio
print(train['Response'].value_counts())
print((train['Response'].value_counts()[0]/train['Response'].value_counts().sum())*100) #~88% observations have
print((train['Response'].value_counts()[1]/train['Response'].value_counts().sum())*100) #~12% observations have
sns.countplot(x='Response',data=train)
```

```
0    334399
1     46710
Name: Response, dtype: int64
87.74366388618479
12.256336113815209
```

Out[7]: <AxesSubplot:xlabel='Response', ylabel='count'>



Clearly the dataset is highly imbalanced, however it advisable to perform Data Balancing after the

Clearly the dataset is highly imbalanced, however it advisable to perform Data Balancing after the Feature Engineering techniques

In [8]:

```
cat=train.select_dtypes('object').columns.tolist() #Categorical Features
num=train.select_dtypes('number').columns.tolist() #Numerical Features
print(cat)
print(num)
```

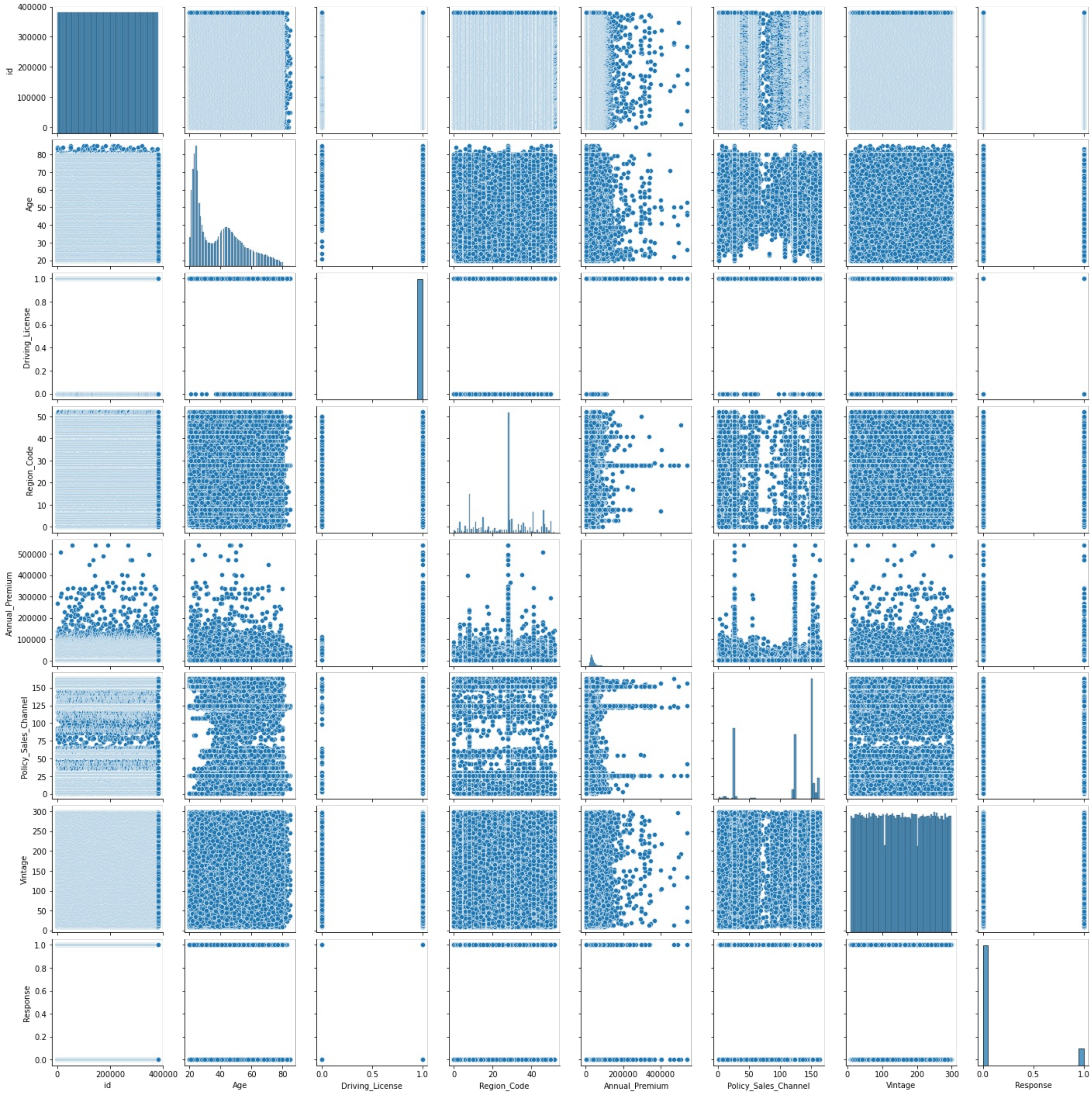
```
['Gender', 'Previously_Insured', 'Vehicle_Age', 'Vehicle_Damage']
['id', 'Age', 'Driving_License', 'Region_Code', 'Annual_Premium', 'Policy_Sales_Channel', 'Vintage', 'Response']
```

In [9]:

```
sns.pairplot(train)
```

Out[9]:

```
<seaborn.axisgrid.PairGrid at 0x254a3bc95b0>
```



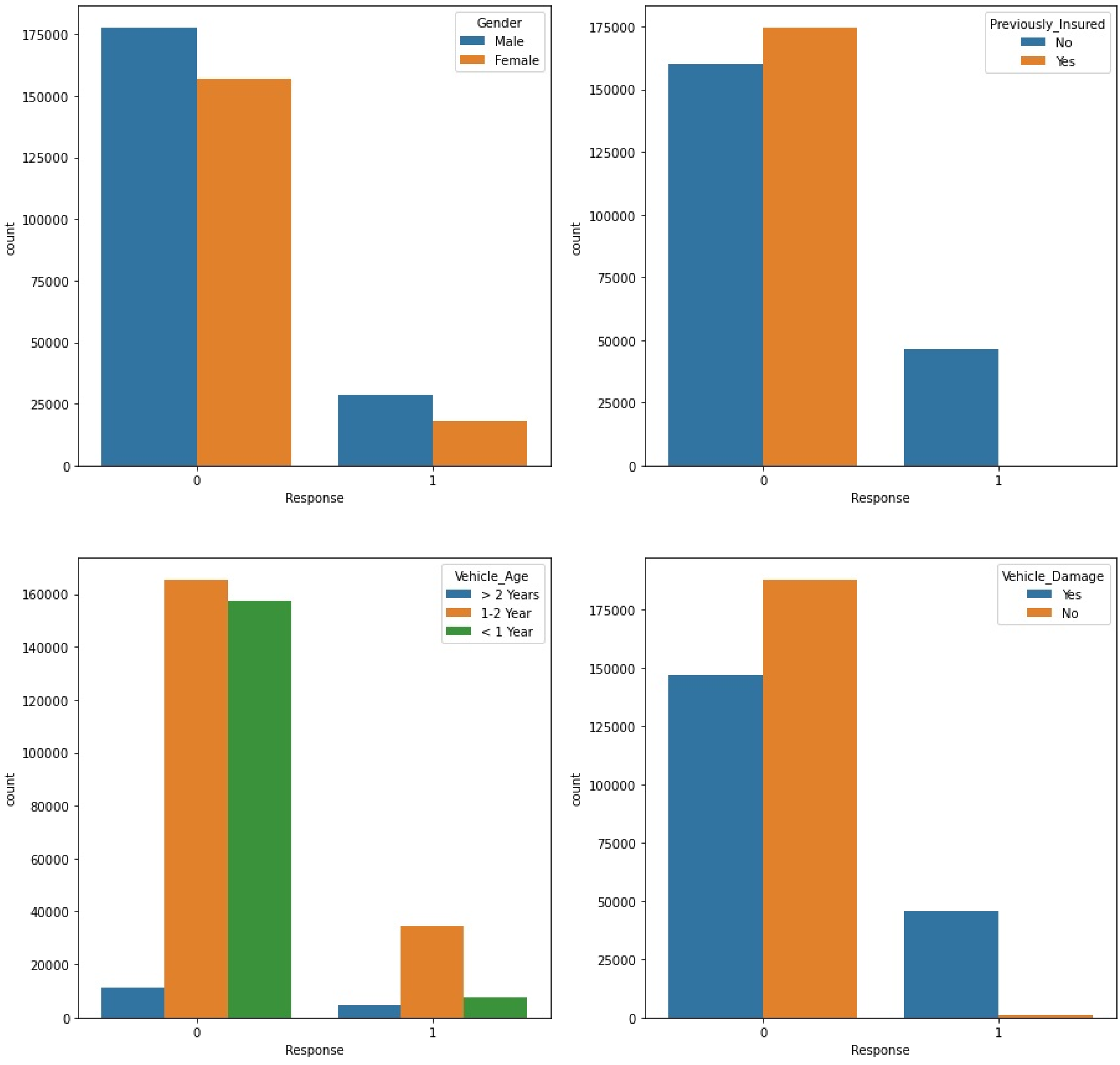
In [10]:

```
fig, axs = plt.subplots(ncols=2,nrows=2,figsize=(15,15))
print(train.groupby('Response')['Gender'].value_counts())
print(train.groupby('Response')['Previously_Insured'].value_counts())
print(train.groupby('Response')['Vehicle_Age'].value_counts())
print(train.groupby('Response')['Vehicle_Damage'].value_counts())
sns.countplot(x='Response',data=train,hue='Gender',ax=axs[0][0])
sns.countplot(x='Response',data=train,hue='Previously_Insured',ax=axs[0][1])
sns.countplot(x='Response',data=train,hue='Vehicle_Age',ax=axs[1][0])
sns.countplot(x='Response',data=train,hue='Vehicle_Damage',ax=axs[1][1])
```

```
Response  Gender
0         Male      177564
          Female    156835
1         Male       28525
          Female     18185
Name: Gender, dtype: int64
Response  Previously_Insured
0         Yes                   174470
          No                    159929
1         No                     46552
          Yes                      158
Name: Previously_Insured, dtype: int64
Response  Vehicle_Age
0         1-2 Year       165510
          < 1 Year       157584
          > 2 Years       11305
1         1-2 Year        34806
          < 1 Year         7202
          > 2 Years        4702
Name: Vehicle_Age, dtype: int64
Response  Vehicle_Damage
0         No                187714
          Yes               146685
1         Yes                45728
          No                   982
Name: Vehicle_Damage, dtype: int64
```

Out[10]:

```
<AxesSubplot:xlabel='Response', ylabel='count'>
```



In [44]:

```
train.groupby('Response')['Age'].describe()
```

Out[44]:

Response	count	mean	std	min	25%	50%	75%	max
0	334399.0	38.178227	15.816052	20.0	24.0	34.0	49.0	85.0
1	46710.0	43.435560	12.168924	20.0	35.0	43.0	51.0	83.0

Inference:An older person would be more likely interested in getting his vehicle insured than a younger person.

In [11]:

```
g=train.groupby('Response')['Gender'].value_counts()
total_males=train[train['Gender']=='Male'].shape[0]
total_females=train[train['Gender']=='Female'].shape[0]
print('Overall:')
print('Response:0')
print('Male:{}%'.format((g[0]['Male']/total_males)*100))
print('Female:{}%'.format((g[0]['Female']/total_females)*100))
print('Response:1')
print('Male:{}%'.format((g[1]['Male']/total_males)*100))
print('Female:{}%'.format((g[1]['Female']/total_females)*100))
print('Within Groups:')
print('Response:0')
print('Male:{}%'.format((g[0]['Male']/g[0].sum())*100))
print('Female:{}%'.format((g[0]['Female']/g[0].sum())*100))
print('Response:1')
print('Male:{}%'.format((g[1]['Male']/g[1].sum())*100))
print('Female:{}%'.format((g[1]['Female']/g[1].sum())*100))
```

```
Overall:
Response:0
Male:86.15889251731048%
Female:89.60975888469889%
Response:1
Male:13.84110748268952%
Female:10.390241115301109%
Within Groups:
Response:0
Male:53.09944108684535%
Female:46.90055891315465%
Response:1
Male:61.06829372725326%
Female:38.93170627274674%
```

About 86% males gave response 0 and ~13% males gave response 1.

About 89% females gave response 0 and 10% females gave response 1.

Within Response 0, there were about 53% males and ~47% females.

Within Response 1, there were about 61% males and ~39% females.

Inference:Males are slightly more interested in getting their vehicle insured than females

In [12]:

```
print('Previously Insured:')
pi=train.groupby('Response')['Previously_Insured'].value_counts()
total_ins_yes=train[train['Previously_Insured']=='Yes'].shape[0]
total_ins_no=train[train['Previously_Insured']=='No'].shape[0]
print('Overall:')
print('Response:0')
print('Yes:{}%'.format((pi[0]['Yes']/total_ins_yes)*100))
print('No:{}%'.format((pi[0]['No']/total_ins_no)*100))
print('Response:1')
print('Yes:{}%'.format((pi[1]['Yes']/total_ins_yes)*100))
print('No:{}%'.format((pi[1]['No']/total_ins_no)*100))
print('Within Groups:')
print('Response:0')
print('Yes:{}%'.format((pi[0]['Yes']/pi[0].sum())*100))
print('No:{}%'.format((pi[0]['No']/pi[0].sum())*100))
print('Response:1')
print('Yes:{}%'.format((pi[1]['Yes']/pi[1].sum())*100))
print('No:{}%'.format((pi[1]['No']/pi[1].sum())*100))
```

```
Previously Insured:
Overall:
Response:0
Yes:99.909521955242%
```

```
No:77.45458419903042%
Response:1
Yes:0.09047804475799986%
No:22.545415800969582%
Within Groups:
Response:0
Yes:52.174199085523576%
No:47.82580091447642%
Response:1
Yes:0.33825733247698564%
No:99.66174266752301%
```

About 99% previously insured gave response 0 and ~0.09% previously insured gave response 1.

About 77% not previously insured gave response 0 and ~23% not previously gave response 1.

Inference:A person who already has a vehicle insurance would very less likely be interested in insuring their vehicle again.

Within Response 0, there were about 52% previously insured and ~48% not previously insured.

Within Response 1, there were about ~0.4% previously insured and ~99.6% not previously insured.

Inference:A person who is interested in insurance would probably not be having any prior insurance for the vehicle.

In [13]:

```
print('Vehicle Age')
va=train.groupby('Response')['Vehicle_Age'].value_counts()
total_va_1=train[train['Vehicle_Age']=='< 1 Year'].shape[0]
total_va_12=train[train['Vehicle_Age']=='1-2 Year'].shape[0]
total_va_2=train[train['Vehicle_Age']=='> 2 Years'].shape[0]
print('Overall:')
print('Response:0')
print('< 1 Year:{}%'.format((va[0]['< 1 Year']/total_va_1)*100))
print('1-2 Year:{}%'.format((va[0]['1-2 Year']/total_va_12)*100))
print('> 2 Years:{}%'.format((va[0]['> 2 Years']/total_va_2)*100))
print('Response:1')
print('< 1 Year:{}%'.format((va[1]['< 1 Year']/total_va_1)*100))
print('1-2 Year:{}%'.format((va[1]['1-2 Year']/total_va_12)*100))
print('> 2 Years:{}%'.format((va[1]['> 2 Years']/total_va_2)*100))
print('Within Groups:')
print('Response:0')
print('< 1 Year:{}%'.format((va[0]['< 1 Year']/va[0].sum())*100))
print('1-2 Year:{}%'.format((va[0]['1-2 Year']/va[0].sum())*100))
print('> 2 Years:{}%'.format((va[0]['> 2 Years']/va[0].sum())*100))
print('Response:1')
print('< 1 Year:{}%'.format((va[1]['< 1 Year']/va[0].sum())*100))
print('1-2 Year:{}%'.format((va[1]['1-2 Year']/va[0].sum())*100))
print('> 2 Years:{}%'.format((va[1]['> 2 Years']/va[0].sum())*100))
```

```
Vehicle Age
Overall:
Response:0
< 1 Year:95.62948308715546%
1-2 Year:82.62445336368538%
> 2 Years:70.62535140875866%
Response:1
< 1 Year:4.370516912844537%
1-2 Year:17.375546636314624%
> 2 Years:29.374648591241332%
Within Groups:
Response:0
< 1 Year:47.12454283655155%
1-2 Year:49.49476523554197%
> 2 Years:3.3806919279064833%
Response:1
< 1 Year:2.1537145745053063%
1-2 Year:10.408523948935255%
> 2 Years:1.4061046833274022%
```

Inference:Older is the age of the vehicle, more likely would the person be interested in vehicle insurance.

In [14]:

```
print('Vehicle Damage')
vd=train.groupby('Response')['Vehicle_Damage'].value_counts()
total_vd_yes=train[train['Vehicle_Damage']=='Yes'].shape[0]
```

```
total_vd_no=train[train['Vehicle_Damage']=='No'].shape[0]
print('Overall:')
print('Response:0')
print('Yes:{}%'.format((vd[0]['Yes']/total_vd_yes)*100))
print('No:{}%'.format((vd[0]['No']/total_vd_no)*100))
print('Response:1')
print('Yes:{}%'.format((vd[1]['Yes']/total_vd_yes)*100))
print('No:{}%'.format((vd[1]['No']/total_vd_no)*100))
print('Within Groups:')
print('Response:0')
print('Yes:{}%'.format((vd[0]['Yes']/vd[0].sum())*100))
print('No:{}%'.format((vd[0]['No']/vd[0].sum())*100))
print('Response:1')
print('Yes:{}%'.format((vd[1]['Yes']/vd[1].sum())*100))
print('No:{}%'.format((vd[1]['No']/vd[1].sum())*100))
```

```
Vehicle Damage
Overall:
Response:0
Yes:76.23445401298248%
No:99.47958621274431%
Response:1
Yes:23.765545987017507%
No:0.5204137872556917%
Within Groups:
Response:0
Yes:43.86526275497235%
No:56.134737245027644%
Response:1
Yes:97.89766645257974%
No:2.1023335474202525%
```

About 76% with damaged vehicle gave response 0 and ~24% with damaged vehicle insured gave response 1.

About ~99.5% with non damaged vehicle gave response 0 and ~0.5% with non damaged vehicle gave response 1.

Inference: A person with non damaged vehicle is very less likely to be interested in vehicle insurance.

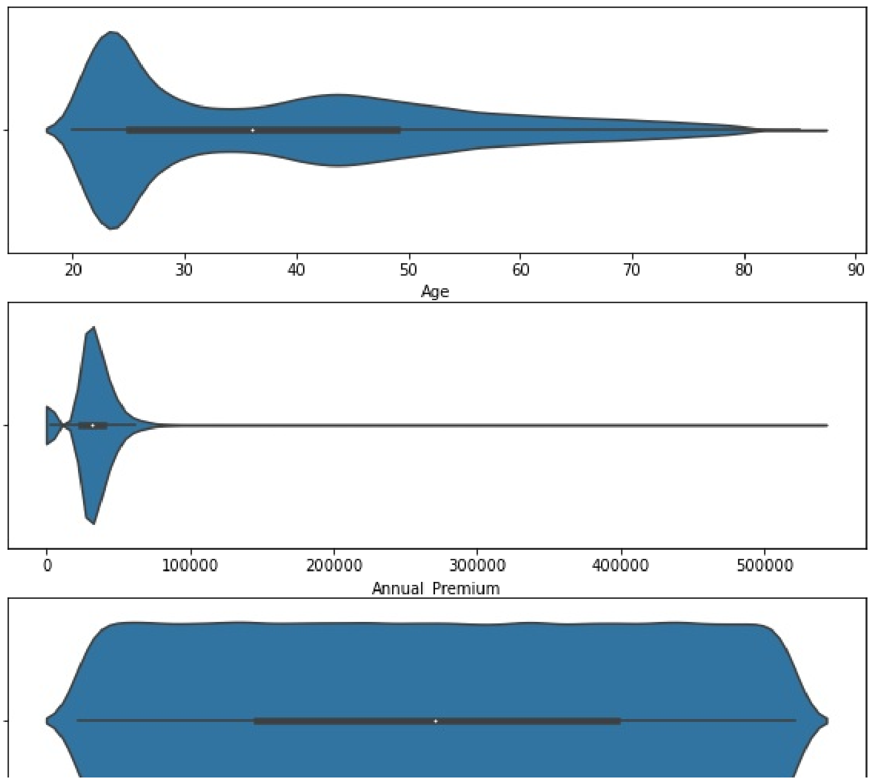
Within Response 0, there were about 44% with damaged vehicle and ~56% with non damaged vehicle.

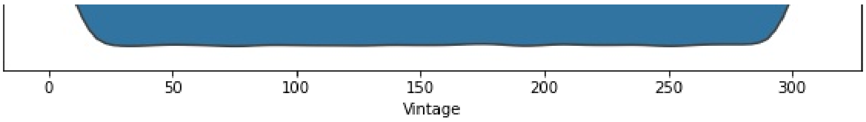
Within Response 1, there were about ~98% with damaged vehicle and ~2% with non damaged vehicle.

Inference: Majority of people interested in vehicle insurance have a damaged vehicle.

In [15]:

```
fig, axs = plt.subplots(nrows=3,figsize=(10,10))
f=['Age','Annual_Premium', 'Vintage']
for i in range(0,3):
    sns.violinplot(x=f[i],data=train,hue='Response',ax=axs[i])
```





In [16]:

```python
x=train.iloc[:,:-1]
y=train.iloc[:,-1]
```

In [17]:

```python
x.drop(['id'],axis=1,inplace=True)
```

Handling Missing Values:

In [18]:

```python
#Handling Missing Values
print(x.isnull().sum())
#for i in cat:
#    x[i].fillna(x[i].mode()[0],inplace=True)
#for i in num:
#    x[i].fillna(x[i].median(),inplace=True)
```

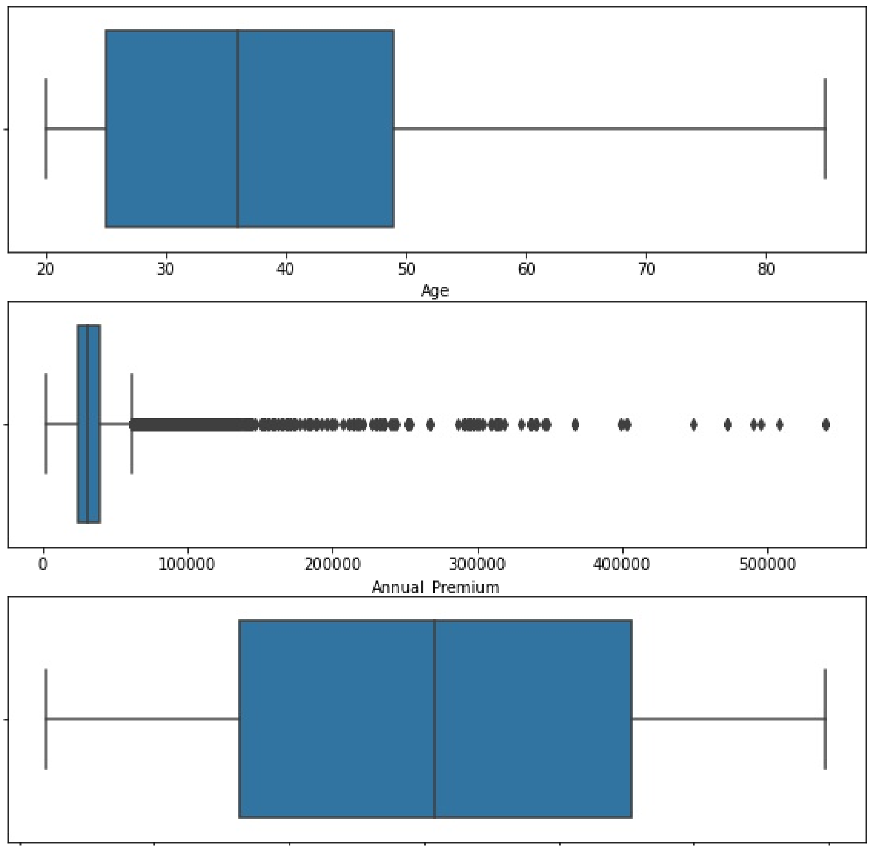
```
Gender                  0
Age                     0
Driving_License         0
Region_Code             0
Previously_Insured      0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Policy_Sales_Channel    0
Vintage                 0
dtype: int64
```

Clearly there are no missing values to handle

Handling Outliers:

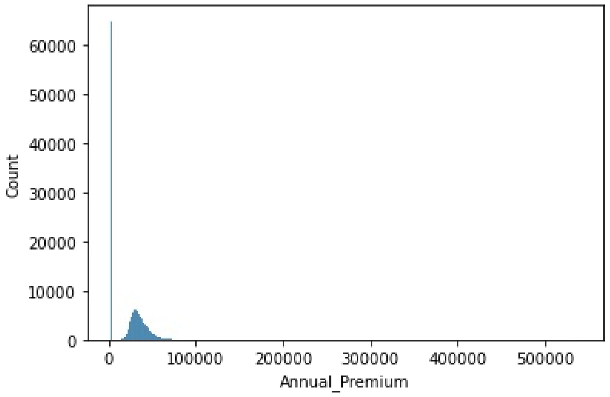
In [19]:

```python
fig, axs = plt.subplots(nrows=3,figsize=(10,10))
f=['Age','Annual_Premium', 'Vintage']
pos=0
for i in range(0,3):
    sns.boxplot(x=f[i],data=train,hue='Response',ax=axs[i])
```



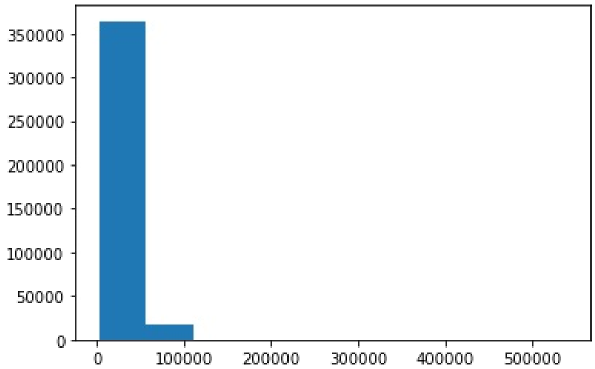
In [20]:
```python
sns.histplot(train['Annual_Premium'])
```

Out[20]:
```
<AxesSubplot:xlabel='Annual_Premium', ylabel='Count'>
```



In [21]:
```python
plt.hist(train['Annual_Premium'])
```

Out[21]:
```
(array([3.64067e+05, 1.65600e+04, 3.20000e+02, 6.10000e+01, 3.40000e+01,
        3.10000e+01, 2.10000e+01, 4.00000e+00, 4.00000e+00, 7.00000e+00]),
 array([  2630. ,  56383.5, 110137. , 163890.5, 217644. , 271397.5,
        325151. , 378904.5, 432658. , 486411.5, 540165. ]),
 <BarContainer object of 10 artists>)
```



In [22]:
```python
train['Annual_Premium'].describe()
```

Out[22]:
```
count    381109.000000
mean      30564.389581
std       17213.155057
min        2630.000000
25%       24405.000000
50%       31669.000000
75%       39400.000000
max      540165.000000
Name: Annual_Premium, dtype: float64
```

In [23]:
```python
q25,q50,q75=np.percentile(sorted(train['Annual_Premium']),[25,50,75])
iqr=q75-q25
min_val=q25-1.5*iqr
max_val=q75+1.5*iqr
print(q25)
print(q50)
print(q75)
print(iqr)
print(min_val)
print(max_val)
cnt_out=len([x for x in train['Annual_Premium'] if x>max_val or x<min_val])
```

```
24405.0
31669.0
39400.0
```

```
14995.0
1912.5
61892.5
```

In [24]:

```
print(cnt_out)
print((cnt_out/train.shape[0])*100)
```

```
10320
2.70788672007221
```

About ~3% of Annual_Premium values have been reported as Outliers.

The outliers must be handled carefully under the guidance of a domain expert as they may represent important characteristics about the data at times.

In [25]:

```
annual_prem_median=train['Annual_Premium'].median()
out1=x[x['Annual_Premium']>max_val].values
out2=x[x['Annual_Premium']<min_val].values
x['Annual_Premium'].replace(out1,annual_prem_median,inplace=True)
x['Annual_Premium'].replace(out2,annual_prem_median,inplace=True)
```

Log Transformation:

In [26]:

```
# Create a list of float colums to check for skewing
mask = x.dtypes == np.float
float_cols = x.columns[mask]

skew_limit = 0.75 # define a limit above which we will log transform
skew_vals = x[float_cols].skew()
```

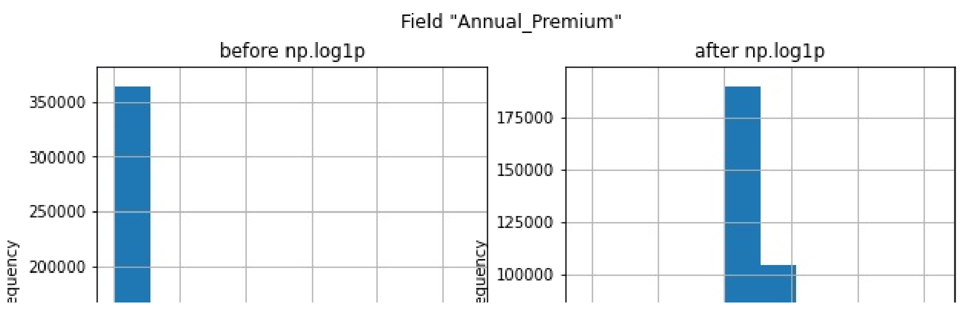
In [27]:

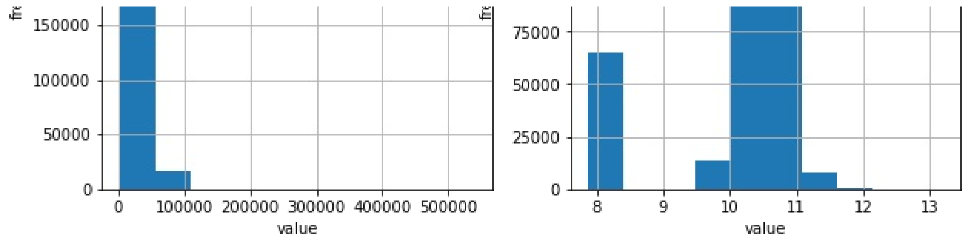
```
# Showing the skewed columns
skew_cols = (skew_vals
             .sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))

print(skew_cols)
```

```
                         Skew
Annual_Premium       1.766087
Policy_Sales_Channel -0.900008
```

In [28]:

```
# Let's look at what happens to one of these features, when we apply np.log1p visually.

# Choose a field
field = "Annual_Premium"

# Create two "subplots" and a "figure" using matplotlib
fig, (ax_before, ax_after) = plt.subplots(1, 2, figsize=(10, 5))

# Create a histogram on the "ax_before" subplot
x[field].hist(ax=ax_before)

# Apply a log transformation (numpy syntax) to this column
x[field].apply(np.log1p).hist(ax=ax_after)

# Formatting of titles etc. for each subplot
ax_before.set(title='before np.log1p', ylabel='frequency', xlabel='value')
ax_after.set(title='after np.log1p', ylabel='frequency', xlabel='value')
fig.suptitle('Field "{}"'.format(field));
```





In [29]:
```
x['Annual_Premium'] = x['Annual_Premium'].apply(np.log1p)
```

In [45]:
```
train.groupby('Response')['Annual_Premium'].describe()
```

Out[45]:

Response	count	mean	std	min	25%	50%	75%	max
0	334399.0	30419.160276	16998.293197	2630.0	24351.0	31504.0	39120.0	540165.0
1	46710.0	31604.092742	18646.508040	2630.0	24868.0	33002.0	41297.0	540165.0

Encoding:

In [30]:
```
#Converting Categorical Data to Numerical Data
from sklearn.preprocessing import OneHotEncoder
#Nominal Features: 'Gender', 'Previously_Insured', 'Vehicle_Damage'
x=pd.get_dummies(data=x,columns=['Gender', 'Previously_Insured', 'Vehicle_Damage'],drop_first=True)
#Ordinal Features: 'Vehicle_Age'
x['Vehicle_Age']=x['Vehicle_Age'].replace({'< 1 Year':1,'1-2 Year':2,'> 2 Years':3})
```

In [31]:
```
x.head()
```

Out[31]:

	Age	Driving_License	Region_Code	Vehicle_Age	Annual_Premium	Policy_Sales_Channel	Vintage	Gender_Male	Previously_Insured_Yes	Ve
0	44	1	28.0	3	10.607946	26.0	217	1	0	
1	76	1	3.0	2	10.420405	26.0	183	1	0	
2	47	1	28.0	3	10.553075	26.0	27	1	0	
3	21	1	11.0	1	10.261861	152.0	203	1	1	
4	29	1	41.0	1	10.221832	152.0	39	0	1	

All the categorical columns have now been encoded as numerical figures

Normalization:

In [32]:
```
scaler1=MinMaxScaler()
scaler1.fit(x)
x_scaled1=scaler1.transform(x)
```

In [33]:
```
pd.DataFrame(x_scaled1,columns=x.columns.tolist())
```

Out[33]:

	Age	Driving_License	Region_Code	Vehicle_Age	Annual_Premium	Policy_Sales_Channel	Vintage	Gender_Male	Previously_Insu
0	0.369231	1.0	0.538462	1.0	0.513254	0.154321	0.716263	1.0	
1	0.861538	1.0	0.057692	0.5	0.478032	0.154321	0.598616	1.0	
2	0.415385	1.0	0.538462	1.0	0.502948	0.154321	0.058824	1.0	
3	0.015385	1.0	0.211538	0.0	0.448255	0.932099	0.667820	1.0	
4	0.138462	1.0	0.788462	0.0	0.440738	0.932099	0.100346	0.0	
...	...	...	...	...	...	...	...	...	
381104	0.830769	1.0	0.500000	0.5	0.458167	0.154321	0.269896	1.0	
381105	0.153846	1.0	0.711538	0.0	0.511209	0.932099	0.418685	1.0	
381106	0.015385	1.0	0.576923	0.0	0.486688	0.981481	0.522491	1.0	
381107	0.738462	1.0	0.269231	1.0	0.531649	0.759259	0.221453	0.0	

381108	0.400000	1.0	0.557692	0.5	0.519297	0.154321	0.785467	1.0

381109 rows × 10 columns

Standardisation:

In [34]:
```
scaler2=StandardScaler()
scaler2.fit(x)
x_scaled2=scaler2.transform(x)
```

In [35]:
```
pd.DataFrame(x_scaled2,columns=x.columns.tolist())
```

Out[35]:

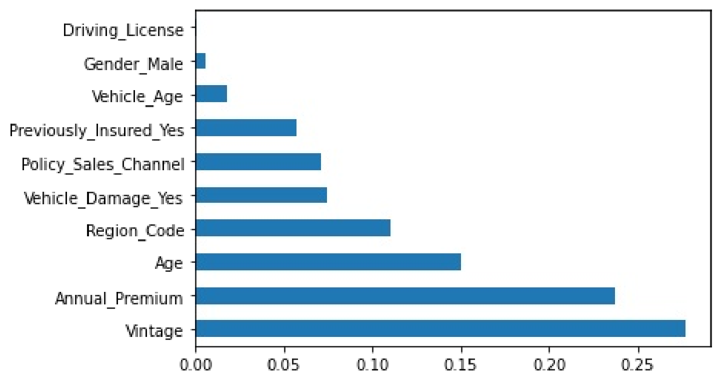
	Age	Driving_License	Region_Code	Vehicle_Age	Annual_Premium	Policy_Sales_Channel	Vintage	Gender_Male	Previously_Insu
0	0.333777	0.046208	0.121784	2.450281	0.590239	-1.587234	0.748795	0.921545	-
1	2.396751	0.046208	-1.767879	0.687976	0.403622	-1.587234	0.342443	0.921545	-
2	0.527181	0.046208	0.121784	2.450281	0.535638	-1.587234	-1.521998	0.921545	-
3	-1.148985	0.046208	-1.163187	-1.074329	0.245859	0.737321	0.581474	0.921545	
4	-0.633242	0.046208	1.104409	-1.074329	0.206027	0.737321	-1.378580	-1.085134	
...	...	...	...	...	...	...	...	...	
381104	2.267815	0.046208	-0.029389	0.687976	0.298374	-1.587234	-0.792954	0.921545	
381105	-0.568774	0.046208	0.802063	-1.074329	0.579406	0.737321	-0.279037	0.921545	
381106	-1.148985	0.046208	0.272958	-1.074329	0.449488	0.884912	0.079509	0.921545	
381107	1.881007	0.046208	-0.936427	2.450281	0.687703	0.220753	-0.960275	-1.085134	-
381108	0.462713	0.046208	0.197371	0.687976	0.622260	-1.587234	0.987826	0.921545	-

381109 rows × 10 columns

Feature Selection:

The ExtraTreesClassifier method will help to give the importance of each independent feature with a dependent feature. Feature importance will give you a score for each feature of your data, the higher the score more important or relevant to the feature towards your output variable.

In [36]:
```
model=ExtraTreesClassifier()
model.fit(x,y)
feat_importances=pd.Series(model.feature_importances_,index=x.columns)
feat_importances.nlargest(10).plot(kind='barh')
plt.show()
```



Data Balancing:

In [37]:
```
smot=SMOTE(sampling_strategy='minority')
x_smot,y_smot=smot.fit_resample(x,y)
```

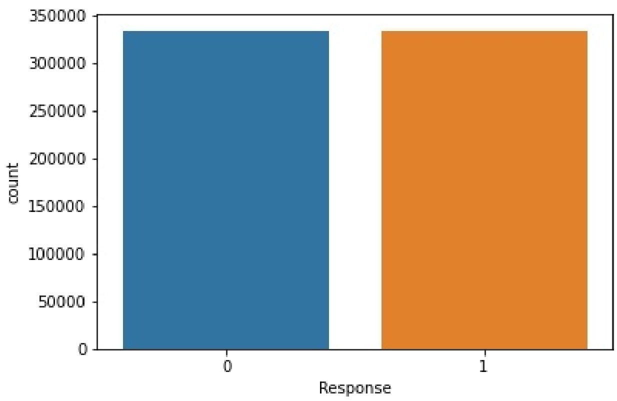
In [41]:
```
#Target Class Ratio
print(y_smot.value_counts())
print((y_smot.value_counts()[0]/y_smot.value_counts().sum())*100) #~88% observations have response 0
```

```
print((y_smot.value_counts()[1]/y_smot.value_counts().sum())*100) #~12% observations have response 0
sns.countplot(x=y_smot)
```

```
1    334399
0    334399
Name: Response, dtype: int64
50.0
50.0
```

Out[41]:

```
<AxesSubplot:xlabel='Response', ylabel='count'>
```



The training dataset has been perfectly balanced using Synthetic Minority Oversampling Technique(SMOTE) which generated synthetic samples for the minority class by imitating the characteristics of similar data points.

Key Points:

Men are more likely to get their vehicle insured than females.

Older is the more greater is the probability that they would be interested vehicle insurance.

A person with damaged car is more likely to be interested in insurance.

Next Steps: Selecting Appropriate Features and involving polynomial/interaction terms, cross validation over different classification models, training appropriate model and testing using test data

Quality of Dataset: The quality of dataset was quite good. There were sufficient number of observations for effecient data balancing. There were no missing values and hardly any problematic outliers. Additional information is needed through domain experts regarding whether to handle 'Annual_Premium' outliers.