In [1]:
```
# importing libraries

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings('ignore')
```

In [2]:
```
# reading training data
train = pd.read_csv('train.csv')
```

In [3]:
```
train.shape
```

Out[3]: (381109, 12)

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

Out[4]:

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

In [5]:
```
# reading test data
test = pd.read_csv('test.csv')
```

In [6]:
```
test.shape
```

Out[6]: (127037, 11)

In [7]:
```
test.head()
```

Out[7]:

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

Combine test and train data for better EDA and preprocessing

In [8]:
```
df = train.append(test)
```

In [9]:
```
df.shape
```

Out[9]: (508146, 12)

In [10]:
```
# first 5 records
df.head()
```

Out[10]:

	id	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	1	Male	44	1	28.0	0	> 2 Years	Yes	40454.0	26.0	217	1.0
1	2	Male	76	1	3.0	0	1-2 Year	No	33536.0	26.0	183	0.0
2	3	Male	47	1	28.0	0	> 2 Years	Yes	38294.0	26.0	27	1.0
3	4	Male	21	1	11.0	1	< 1 Year	No	28619.0	152.0	203	0.0
4	5	Female	29	1	41.0	1	< 1 Year	No	27496.0	152.0	39	0.0

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 508146 entries, 0 to 127036
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   id                    508146 non-null  int64  
 1   Gender                508146 non-null  object 
 2   Age                   508146 non-null  int64  
 3   Driving_License       508146 non-null  int64  
 4   Region_Code           508146 non-null  float64
 5   Previously_Insured    508146 non-null  int64  
 6   Vehicle_Age           508146 non-null  object 
 7   Vehicle_Damage        508146 non-null  object 
 8   Annual_Premium        508146 non-null  float64
 9   Policy_Sales_Channel  508146 non-null  float64
 10  Vintage               508146 non-null  int64  
 11  Response              381109 non-null  float64
dtypes: float64(4), int64(5), object(3)
memory usage: 50.4+ MB
```

Here we have gender, vehicle_age and vehicle damage in object datatype
so we need to apply appropiate encoding technique on those features
Target variable : Response

In [12]:
```
# statistical inforrmation about numerical columns
df.describe()
```

Out[12]:

	id	Age	Driving_License	Region_Code	Previously_Insured	Annual_Premium	Policy_Sales_Channel	Vintage	Response
count	508146.000000	508146.000000	508146.000000	508146.000000	508146.000000	508146.000000	508146.000000	508146.000000	381109.000000
mean	254073.500000	38.808413	0.997936	26.406572	0.458667	30554.453041	111.975838	154.340123	0.122563
std	146689.259281	15.500179	0.045388	13.224921	0.498289	17146.574625	54.246027	83.668793	0.327936
min	1.000000	20.000000	0.000000	0.000000	0.000000	2630.000000	1.000000	10.000000	0.000000
25%	127037.250000	25.000000	1.000000	15.000000	0.000000	24381.000000	26.000000	82.000000	0.000000
50%	254073.500000	36.000000	1.000000	28.000000	0.000000	31661.000000	133.000000	154.000000	0.000000
75%	381109.750000	49.000000	1.000000	35.000000	1.000000	39403.750000	152.000000	227.000000	0.000000
max	508146.000000	85.000000	1.000000	52.000000	1.000000	540165.000000	163.000000	299.000000	1.000000

In [13]:
```
# statistical information about categorical columns

df.describe(include='O')
```

Out[13]:

	Gender	Vehicle_Age	Vehicle_Damage
count	508146	508146	508146
unique	2	3	2
top	Male	1-2 Year	Yes
freq	274325	267015	256248

In [14]:
```
# analysing null values

df.isnull().sum()
```

Out[14]:
```
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               127037
dtype: int64
```

No null values in training data

and we dont need to handle null values as they are of test.csv

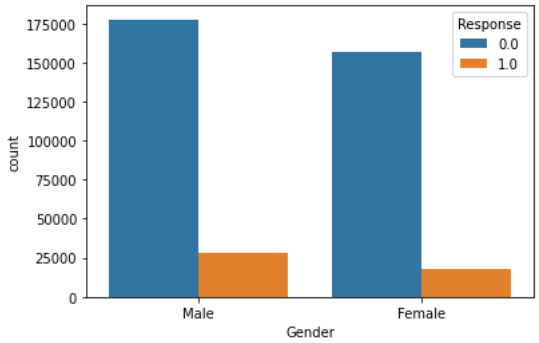
ID

In [15]:
```
#Drop id column
df.drop('id',inplace=True,axis=1)
```

gender

In [16]:
```
sns.countplot(df['Gender'],hue=df['Response'])
```

Out[16]:
```
<AxesSubplot:xlabel='Gender', ylabel='count'>
```



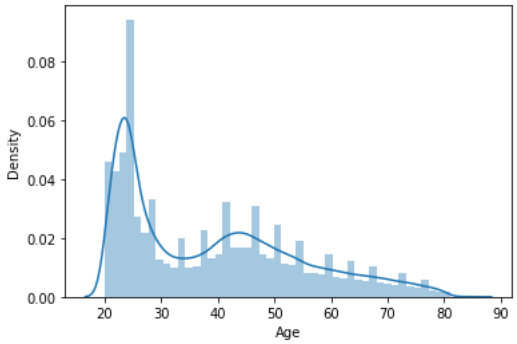
Observation:

Number of Males are more than females
Both male and female choose not to respond

Age

In [17]:
```
sns.distplot(df['Age'])
```

Out[17]:
```
<AxesSubplot:xlabel='Age', ylabel='Density'>
```



Observations:

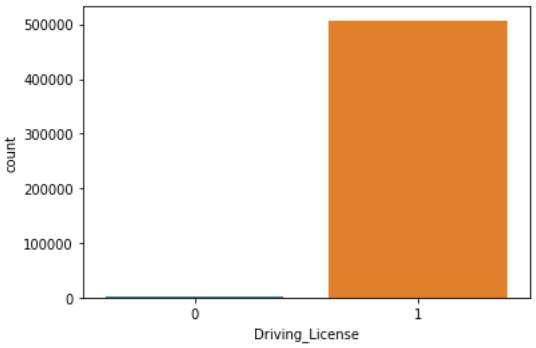
Highly right skewed
The age range is 20 to 80
whereas highest age range is 20 to 30

Driving License

In [18]:
```
print(df['Driving_License'].value_counts())
sns.countplot(df['Driving_License'])
```

```
1    507097
0      1049
Name: Driving_License, dtype: int64
```

Out[18]: <AxesSubplot:xlabel='Driving_License', ylabel='count'>



Observations: Almost everyone 99% people have driving license

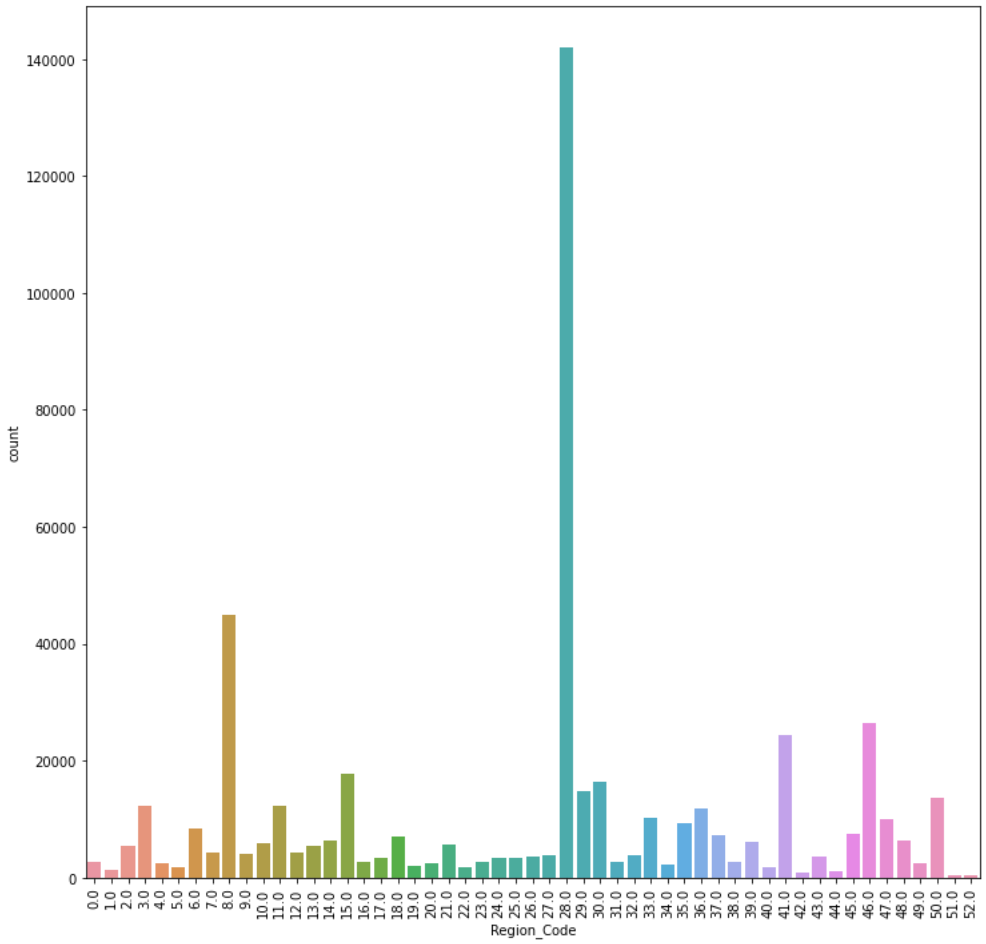
Region_Code

In [19]:
```
df['Region_Code'].value_counts()
```

Out[19]:
```
28.0    141937
8.0      44900
46.0     26357
41.0     24400
15.0     17750
30.0     16276
29.0     14843
50.0     13657
3.0      12349
11.0     12328
36.0     11696
33.0     10307
47.0      9942
35.0      9309
6.0       8351
45.0      7543
37.0      7343
18.0      6903
48.0      6274
14.0      6249
39.0      6138
10.0      5847
21.0      5671
2.0       5398
13.0      5396
7.0       4313
12.0      4235
9.0       4084
32.0      3745
27.0      3711
43.0      3508
26.0      3461
17.0      3455
25.0      3352
24.0      3263
16.0      2727
0.0       2699
38.0      2677
31.0      2635
23.0      2596
20.0      2528
49.0      2388
4.0       2379
34.0      2190
19.0      2059
22.0      1729
40.0      1708
5.0       1698
1.0       1363
44.0      1064
42.0       787
52.0       357
51.0       271
Name: Region_Code, dtype: int64
```

In [20]:
```
    plt.figure(figsize=(12,12))
    plt.xticks(rotation=90)
    sns.countplot(df['Region_Code'])
```

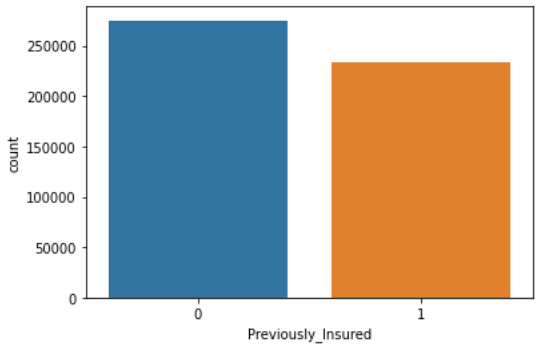
Out[20]: <AxesSubplot:xlabel='Region_Code', ylabel='count'>



Previously_Insured

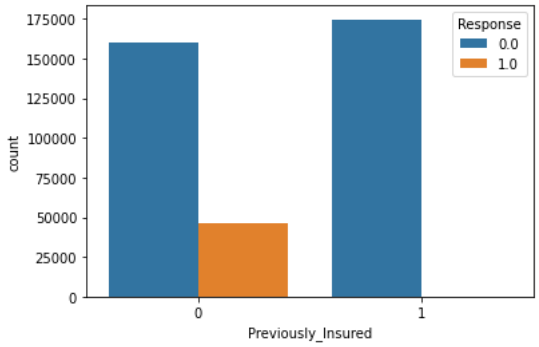
In [21]:
```
    sns.countplot(df['Previously_Insured'])
```

Out[21]: <AxesSubplot:xlabel='Previously_Insured', ylabel='count'>



In [22]:
```
    sns.countplot(df['Previously_Insured'], hue=df['Response'])
```

Out[22]: <AxesSubplot:xlabel='Previously_Insured', ylabel='count'>



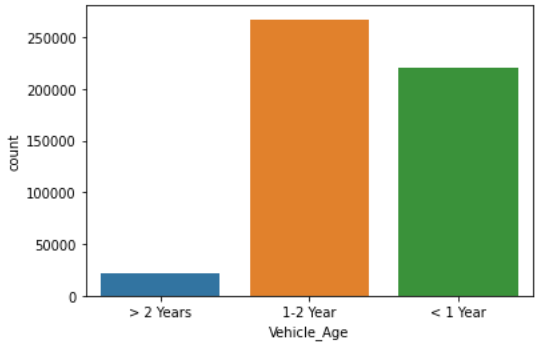
Observations:

No one responed who had previously insured.
30% people respond who had not previously insured

Vehicle_Age

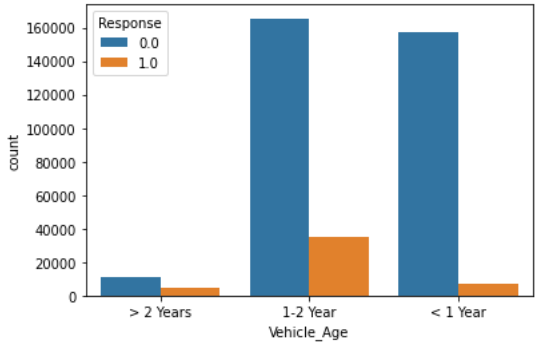
In [23]:
```
sns.countplot(df['Vehicle_Age'])
```

Out[23]: <AxesSubplot:xlabel='Vehicle_Age', ylabel='count'>



In [24]:
```
sns.countplot(df['Vehicle_Age'], hue = df['Response'])
```

Out[24]: <AxesSubplot:xlabel='Vehicle_Age', ylabel='count'>



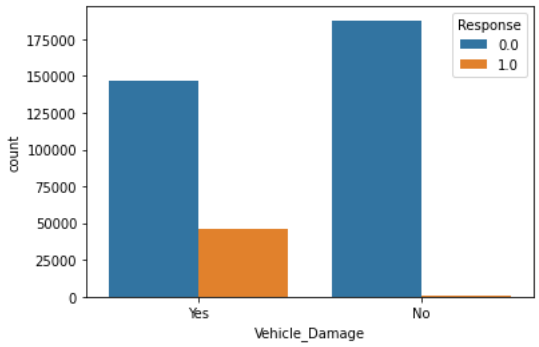
Observation:

Vehicles with 0-2 years have highest number and vehicles more than 2 years are very less.

Vehicle_Damage

In [25]:
```
sns.countplot(df['Vehicle_Damage'], hue=df['Response'])
```

Out[25]: <AxesSubplot:xlabel='Vehicle_Damage', ylabel='count'>



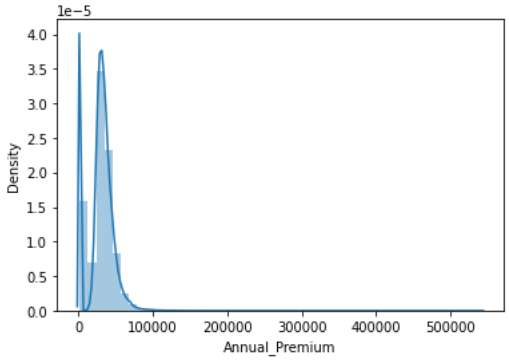
Observation:

naturally Vehicle with damage replied more than vehicle withour damage.

Annual_Premium

In [26]:
```
sns.distplot(df['Annual_Premium'])
```

Out[26]:
```
<AxesSubplot:xlabel='Annual_Premium', ylabel='Density'>
```

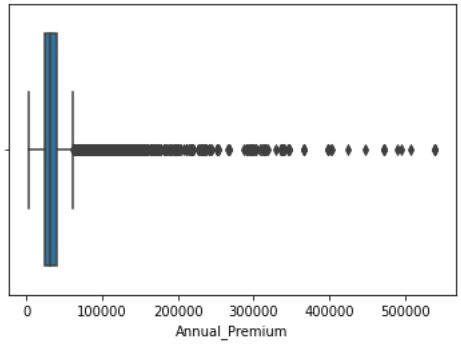


Observations:

```
Highly Right skewed and bimodal i.e. graph has two modes
Most of the price of annual premium is less than 100000
```

In [27]:
```
sns.boxplot(df['Annual_Premium'])
```

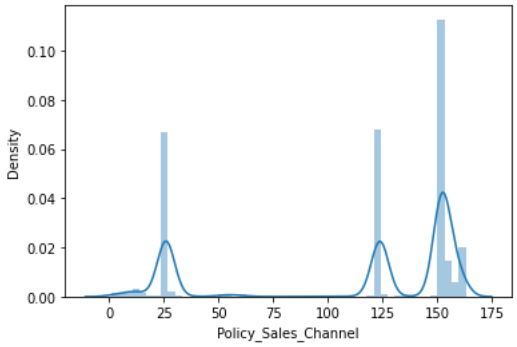
Out[27]:
```
<AxesSubplot:xlabel='Annual_Premium'>
```



Policy_Sales_Channel

In [28]:
```
sns.distplot(df['Policy_Sales_Channel'])
```

Out[28]:
```
<AxesSubplot:xlabel='Policy_Sales_Channel', ylabel='Density'>
```

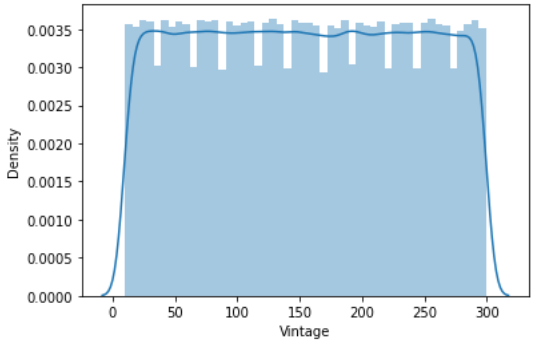


uneven distribution of policy sales channnel with 3 peaks at 25, 125 and 150 the peak at 150 is highest

Vintage

In [29]:
```
sns.distplot(df['Vintage'])
```

Out[29]:
```
<AxesSubplot:xlabel='Vintage', ylabel='Density'>
```

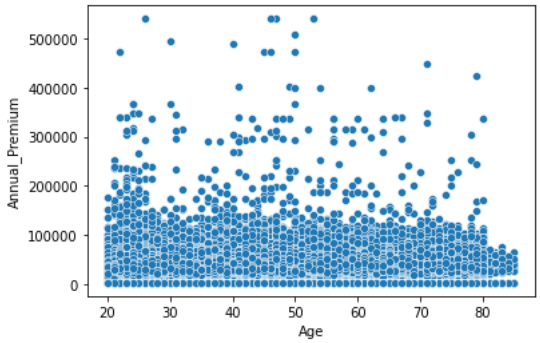


Observation : uniform distribution

Age Vs Annual premium

In [30]:
```
sns.scatterplot(x=df['Age'],y=df['Annual_Premium'])
```

Out[30]:
```
<AxesSubplot:xlabel='Age', ylabel='Annual_Premium'>
```

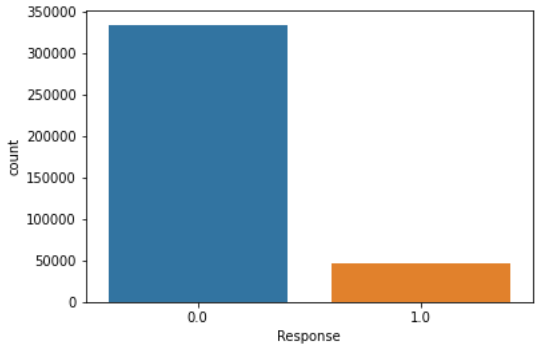


As the age increases annual premium is slightly decreasing

Target variable: Response

In [31]:
```
sns.countplot(df['Response'])
```

Out[31]:
```
<AxesSubplot:xlabel='Response', ylabel='count'>
```



Highly imbalance data People who responded are very less are compared to people who respond

In [ ]:
```
```

Encoding

In [32]:
```
cat_col = df.select_dtypes(include='O').columns
cat_col
```

Out[32]:
```
Index(['Gender', 'Vehicle_Age', 'Vehicle_Damage'], dtype='object')
```

In [33]:
```
df['Gender'] = df['Gender'].map({'Male':0, 'Female':1})
```

In [34]:
```
df['Vehicle_Damage'] = df['Vehicle_Damage'].map({'No':0, 'Yes':1})
```

In [35]:
```
df['Vehicle_Age'] = df['Vehicle_Age'].map({'> 2 Years':2, '1-2 Year':1, '< 1 Year':0 })
```

In [36]:
```
df.head()
```

Out[36]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	0	44	1	28.0	0	2	1	40454.0	26.0	217	1.0
1	0	76	1	3.0	0	1	0	33536.0	26.0	183	0.0
2	0	47	1	28.0	0	2	1	38294.0	26.0	27	1.0
3	0	21	1	11.0	1	0	0	28619.0	152.0	203	0.0
4	1	29	1	41.0	1	0	0	27496.0	152.0	39	0.0

Seperating training and testing data

In [37]:
```
train = df.iloc[:381109]
```

In [38]:
```
train
```

Out[38]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	0	44	1	28.0	0	2	1	40454.0	26.0	217	1.0
1	0	76	1	3.0	0	1	0	33536.0	26.0	183	0.0
2	0	47	1	28.0	0	2	1	38294.0	26.0	27	1.0
3	0	21	1	11.0	1	0	0	28619.0	152.0	203	0.0
4	1	29	1	41.0	1	0	0	27496.0	152.0	39	0.0
...	...	...	...	...	...	...	...	...	...	...	...
381104	0	74	1	26.0	1	1	0	30170.0	26.0	88	0.0
381105	0	30	1	37.0	1	0	0	40016.0	152.0	131	0.0
381106	0	21	1	30.0	1	0	0	35118.0	160.0	161	0.0
381107	1	68	1	14.0	0	2	1	44617.0	124.0	74	0.0
381108	0	46	1	29.0	0	1	0	41777.0	26.0	237	0.0

381109 rows × 11 columns

In [39]:
```
test = df.iloc[381109:,:-1]
```

In [40]:
```
test
```

Out[40]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage
0	0	25	1	11.0	1	0	0	35786.0	152.0	53
1	0	40	1	28.0	0	1	1	33762.0	7.0	111
2	0	47	1	28.0	0	1	1	40050.0	124.0	199
3	0	24	1	27.0	1	0	1	37356.0	152.0	187
4	0	27	1	28.0	1	0	0	59097.0	152.0	297
...	...	...	...	...	...	...	...	...	...	...
127032	1	26	1	37.0	1	0	0	30867.0	152.0	56
127033	1	38	1	28.0	0	1	1	28700.0	122.0	165
127034	0	21	1	46.0	1	0	0	29802.0	152.0	74
127035	0	71	1	28.0	1	1	0	62875.0	26.0	265
127036	0	41	1	29.0	1	1	0	27927.0	124.0	231

127037 rows × 10 columns

In [41]:
```
train.to_csv('cleaned_data_insurance.csv', index=False)
```

In [42]:
```
df = pd.read_csv('cleaned_data_insurance.csv')
```

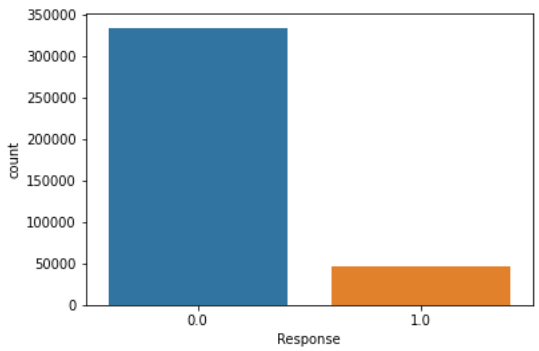
In [43]:
```
df.head()
```

Out[43]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	0	44	1	28.0	0	2	1	40454.0	26.0	217	1.0
1	0	76	1	3.0	0	1	0	33536.0	26.0	183	0.0
2	0	47	1	28.0	0	2	1	38294.0	26.0	27	1.0
3	0	21	1	11.0	1	0	0	28619.0	152.0	203	0.0
4	1	29	1	41.0	1	0	0	27496.0	152.0	39	0.0

In [44]:
```
sns.countplot(df['Response'])
```

Out[44]:
```
<AxesSubplot:xlabel='Response', ylabel='count'>
```



In [45]:
```
from sklearn.model_selection import train_test_split
```

In [46]:
```
X = df.drop('Response', axis=1)
y = df['Response']
```

In [47]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=123, test_size=0.2)
```

In [48]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(304887, 10)
(76222, 10)
(304887,)
(76222,)
```

In [49]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.metrics import classification_report, confusion_matrix, roc_curve, roc_auc_score, auc
```

In [50]:
```
def model(model, X_train, y_train, X_test):

    # initializing model
    cls = model()

    # fitting model
    cls.fit(X_train, y_train)

    # making prediction
    y_train_pred = cls.predict(X_train)
    y_test_pred  = cls.predict(X_test)

    # classification report and confusion matrix
    print('Training data')
    print(classification_report(y_train, y_train_pred))
    print(confusion_matrix(y_train, y_train_pred))

    print('Testing data')
    print(classification_report(y_test, y_test_pred))
    print(confusion_matrix(y_test, y_test_pred))

    return model
```

Logistic Regression

In [51]:
```
model(LogisticRegression, X_train, y_train, X_test)
```

```
Training data
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93    267502
         1.0       0.00      0.00      0.00     37385

    accuracy                           0.88    304887
   macro avg       0.44      0.50      0.47    304887
weighted avg       0.77      0.88      0.82    304887

[[267502      0]
 [ 37385      0]]
Testing data
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93     66897
         1.0       0.00      0.00      0.00      9325

    accuracy                           0.88     76222
   macro avg       0.44      0.50      0.47     76222
weighted avg       0.77      0.88      0.82     76222

[[66897     0]
 [ 9325     0]]
```

Out[51]:
```
sklearn.linear_model._logistic.LogisticRegression
```

RandomForestClassifier

In [52]:
```
model(RandomForestClassifier, X_train, y_train, X_test)
```

```
Training data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    267502
         1.0       1.00      1.00      1.00     37385

    accuracy                           1.00    304887
   macro avg       1.00      1.00      1.00    304887
weighted avg       1.00      1.00      1.00    304887

[[267496      6]
 [    32  37353]]
Testing data
              precision    recall  f1-score   support

         0.0       0.89      0.97      0.93     66897
         1.0       0.36      0.12      0.18      9325

    accuracy                           0.87     76222
   macro avg       0.62      0.55      0.55     76222
weighted avg       0.82      0.87      0.84     76222

[[64856  2041]
 [ 8183  1142]]
```

Out[52]:
```
sklearn.ensemble._forest.RandomForestClassifier
```

XGBoostClassifier

In [53]:
```
model(xgb.XGBClassifier, X_train, y_train, X_test )
```

```
Training data
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.94    267502
         1.0       0.73      0.04      0.08     37385

    accuracy                           0.88    304887
   macro avg       0.81      0.52      0.51    304887
weighted avg       0.86      0.88      0.83    304887

[[266900    602]
 [ 35717   1668]]
Testing data
              precision    recall  f1-score   support

         0.0       0.88      1.00      0.93     66897
         1.0       0.47      0.03      0.05      9325

    accuracy                           0.88     76222
   macro avg       0.67      0.51      0.49     76222
weighted avg       0.83      0.88      0.83     76222

[[66619   278]
 [ 9082   243]]
```

Out[53]:
```
xgboost.sklearn.XGBClassifier
```

Since data is highly imbalance we are getting good accuracy score but poor f-1 score

Perform Oversampling as data is imbalance

Perform Oversampling no minority data

In [54]:
```
from sklearn.utils import resample
```

In [55]:
```
# seperating majority and minority classes

df_majority = df[df['Response'] == 0]
df_minority = df[df['Response'] == 1]

print(df_majority.shape)
print(df_minority.shape)
```

```
(334399, 11)
(46710, 11)
```

In [56]:
```
# upsample minority class

df_minority_oversampled = resample(df_minority,
                                   replace=True,         # sample with replacement
                                   n_samples = 334399,    # to match majority class
                                   random_state = 123)   # reproducible result
```

In [57]:
```
# combine majority_class with upsampled minority class
df_oversampled = pd.concat([df_majority, df_minority_oversampled])
```

In [58]:
```
# display new class count
df_oversampled.Response.value_counts()
```

Out[58]:
```
0.0    334399
1.0    334399
Name: Response, dtype: int64
```

Building models on oversampled data

In [59]:
```
X = df_oversampled.drop('Response', axis=1)
y = df_oversampled['Response']
```

In [60]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=123, test_size=0.2)
```

Logistic Regression on oversampled data

In [61]:
```
    model(LogisticRegression, X_train, y_train, X_test)
```

```
Training data
              precision    recall  f1-score   support

         0.0       0.58      0.63      0.60    267660
         1.0       0.59      0.54      0.57    267378

    accuracy                           0.59    535038
   macro avg       0.59      0.59      0.58    535038
weighted avg       0.59      0.59      0.58    535038

[[168164  99496]
 [122380 144998]]
Testing data
              precision    recall  f1-score   support

         0.0       0.58      0.63      0.60     66739
         1.0       0.59      0.54      0.57     67021

    accuracy                           0.59    133760
   macro avg       0.59      0.59      0.58    133760
weighted avg       0.59      0.59      0.58    133760

[[41840 24899]
 [30537 36484]]
```

Out[61]:
```
sklearn.linear_model._logistic.LogisticRegression
```

Random Forest CLassifier on oversampled data

In [62]:
```
    model(RandomForestClassifier, X_train, y_train, X_test )
```

```
Training data
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00    267660
         1.0       1.00      1.00      1.00    267378

    accuracy                           1.00    535038
   macro avg       1.00      1.00      1.00    535038
weighted avg       1.00      1.00      1.00    535038

[[267613     47]
 [     0 267378]]
Testing data
              precision    recall  f1-score   support

         0.0       1.00      0.90      0.95     66739
         1.0       0.91      1.00      0.95     67021

    accuracy                           0.95    133760
   macro avg       0.95      0.95      0.95    133760
weighted avg       0.95      0.95      0.95    133760

[[60193  6546]
 [  138 66883]]
```

Out[62]:
```
sklearn.ensemble._forest.RandomForestClassifier
```

XGBClassifier on oversampled data

In [63]:
```
model(xgb.XGBClassifier, X_train, y_train, X_test )
```

```
Training data
              precision    recall  f1-score   support

         0.0       0.92      0.69      0.79    267660
         1.0       0.75      0.94      0.84    267378

    accuracy                           0.82    535038
   macro avg       0.84      0.82      0.81    535038
weighted avg       0.84      0.82      0.81    535038

[[185589  82071]
 [ 15472 251906]]
Testing data
              precision    recall  f1-score   support

         0.0       0.92      0.69      0.79     66739
         1.0       0.75      0.94      0.83     67021

    accuracy                           0.81    133760
   macro avg       0.83      0.81      0.81    133760
weighted avg       0.83      0.81      0.81    133760

[[45888 20851]
 [ 4125 62896]]
```

Out[63]:
```
xgboost.sklearn.XGBClassifier
```

For all the 3 models we have better performance (better f1 score) than the previous 3 models

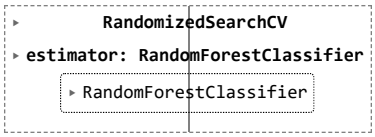
Tuning Random Forest Classifier

In [64]:
```
from sklearn.model_selection import RandomizedSearchCV
```

In [65]:
```
random_search = {'criterion': ['entropy', 'gini'],
                 'max_depth': [2,3,4,5,6,7,10],
                 'min_samples_leaf': [4, 6, 8],
                 'min_samples_split': [5, 7,10],
                 'n_estimators': [300]}

clf = RandomForestClassifier()
model = RandomizedSearchCV(estimator = clf, param_distributions = random_search, n_iter = 10,
                           cv = 5, verbose= 1, random_state= 101, n_jobs = -1)
model.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[65]:



In [66]:
```
model.best_params_
```

Out[66]:
```
{'n_estimators': 300,
 'min_samples_split': 7,
 'min_samples_leaf': 6,
 'max_depth': 10,
 'criterion': 'entropy'}
```

In [67]:
```
model.best_score_
```

Out[67]:
```
0.8003263322577039
```

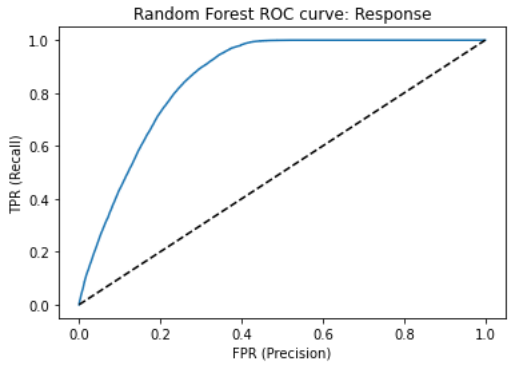
In [68]:
```
y_test_pred = model.predict(X_test)
```

In [69]:
```
print (classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.65      0.77     66739
         1.0       0.73      0.94      0.83     67021

    accuracy                           0.80    133760
   macro avg       0.83      0.80      0.80    133760
weighted avg       0.83      0.80      0.80    133760
```

ROC Curve & AUC of Random forest classifier

In [70]:
```
y_score = model.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)

plt.title('Random Forest ROC curve: Response')
plt.xlabel('FPR (Precision)')
plt.ylabel('TPR (Recall)')

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




```
Area under curve (AUC):  0.8591028566893121
```

In [71]:
```
roc_auc_score(y_test, y_score)
```

Out[71]:
```
0.8591028566893121
```

Hyperparameter tuning for xgboostclassifier

In [72]:
```
## Hyper Parameter Optimization

params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

In [73]:
```
random_search=RandomizedSearchCV(xgb.XGBClassifier(),param_distributions=params,n_iter=5,n_jobs=-1,cv=5,verbose=3)
```

In [74]:
```
random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[74]:
```
RandomizedSearchCV
estimator: XGBClassifier
XGBClassifier
```

In [75]:
```
random_search.best_params_
```

Out[75]:
```
{'min_child_weight': 7,
 'max_depth': 6,
 'learning_rate': 0.25,
 'gamma': 0.3,
 'colsample_bytree': 0.5}
```

In [76]:
```
random_search.best_score_
```

Out[76]:
```
0.8078510302958091
```

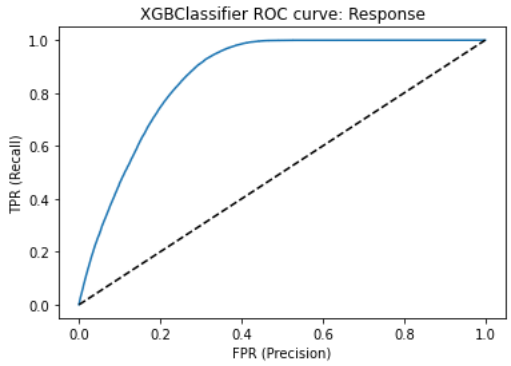
In [77]:
```
y_test_pred = random_search.predict(X_test)
```

In [78]:
```
print (classification_report(y_test, y_test_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.91      0.68      0.78     66739
         1.0       0.75      0.93      0.83     67021

    accuracy                           0.81    133760
   macro avg       0.83      0.81      0.80    133760
weighted avg       0.83      0.81      0.80    133760
```

ROC Curve & AUC of XGBClassifier

In [79]:
```
y_score = random_search.predict_proba(X_test)[:,1]
fpr, tpr, _ = roc_curve(y_test, y_score)

plt.title('XGBClassifier ROC curve: Response')
plt.xlabel('FPR (Precision)')
plt.ylabel('TPR (Recall)')

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




```
Area under curve (AUC):  0.8656989672062186
```

In [80]:
```
import pickle
filename = 'xgb_final.pkl'
pickle.dump(random_search, open(filename, 'wb'))
```

Submission File

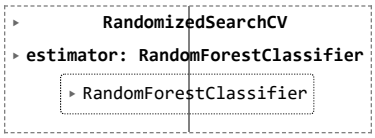
Random Forest Classifier

In [81]:
```
X_train = train.iloc[:,:-1]
X_test = test
y_train = train['Response']
```

In [82]:
```
## Hyper Parameter Optimization
# random forest
random_search = {'criterion': ['entropy', 'gini'],
                 'max_depth': [2,3,4,5,6,7,10],
                 'min_samples_leaf': [4, 6, 8],
                 'min_samples_split': [5, 7,10],
                 'n_estimators': [300]}

clf = RandomForestClassifier()
model = RandomizedSearchCV(estimator = clf, param_distributions = random_search, n_iter = 10,
                                cv = 5, verbose= 1, random_state= 101, n_jobs = -1)
model.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

Out[82]:



In [83]:
```
model.best_params_
```

Out[83]:
```
{'n_estimators': 300,
 'min_samples_split': 7,
 'min_samples_leaf': 8,
 'max_depth': 3,
 'criterion': 'gini'}
```

In [84]:
```
model.best_score_
```

Out[84]:
```
0.8774366388584726
```

In [85]:
```
y_test_pred = model.predict_proba(X_test)
```

In [86]:
```
y_test_pred
```

Out[86]:
```
array([[0.99512992, 0.00487008],
       [0.73840866, 0.26159134],
       [0.7313478 , 0.2686522 ],
       ...,
       [0.99481952, 0.00518048],
       [0.98322238, 0.01677762],
       [0.97449478, 0.02550522]])
```

In [87]:
```
y_test_pred[:,0]
```

Out[87]:
```
array([0.99512992, 0.73840866, 0.7313478 , ..., 0.99481952, 0.98322238,
       0.97449478])
```

In [88]:
```
# create sample submission file and submit
pred_rf = pd.DataFrame(y_test_pred[:,0])
sub_df_rf = pd.read_csv('sample_submission_iA3afxn.csv')
datasets_rf = pd.concat([sub_df_rf['id'],pred_rf],axis=1)
datasets_rf.columns = ['id','Response']
```

In [89]:
```
datasets_rf.to_csv('sample_submission.csv',index=False)
```

XGBoost Classifier

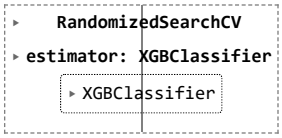
In [90]:
```
## Hyper Parameter Optimization

params={
 "learning_rate"    : [0.05, 0.10, 0.15, 0.20, 0.25, 0.30 ] ,
 "max_depth"        : [ 3, 4, 5, 6, 8, 10, 12, 15],
 "min_child_weight" : [ 1, 3, 5, 7 ],
 "gamma"            : [ 0.0, 0.1, 0.2 , 0.3, 0.4 ],
 "colsample_bytree" : [ 0.3, 0.4, 0.5 , 0.7 ]

}
```

In [91]:
```
random_search=RandomizedSearchCV(xgb.XGBClassifier(),param_distributions=params,n_iter=5,n_jobs=-1,cv=5,verbose=3)
```

In [92]:
```
random_search.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
```

Out[92]:



In [93]:
```
random_search.best_params_
```

Out[93]:
```
{'min_child_weight': 3,
 'max_depth': 6,
 'learning_rate': 0.2,
 'gamma': 0.3,
 'colsample_bytree': 0.5}
```

In [94]:
```
random_search.best_score_
```

Out[94]:
```
0.8775547161060698
```

In [95]:
```
y_test_pred = random_search.predict_proba(X_test)
```

In [96]:
```
y_test_pred
```

Out[96]:
```
array([[9.9910760e-01, 8.9241110e-04],
       [6.6675973e-01, 3.3324024e-01],
       [6.8987769e-01, 3.1012231e-01],
       ...,
       [9.9985951e-01, 1.4051521e-04],
       [9.9984020e-01, 1.5978115e-04],
       [9.9895310e-01, 1.0468853e-03]], dtype=float32)
```

In [97]:
```
# create sample submission file and submit
pred_xgb = pd.DataFrame(y_test_pred[:,0])
sub_df_xgb = pd.read_csv('sample_submission_iA3afxn.csv')
datasets_xgb = pd.concat([sub_df_xgb['id'],pred_xgb],axis=1)
datasets_xgb.columns = ['id','Response']
```

In [98]:
```
    datasets_rf.to_csv('sample_submission_XGB.csv',index=False)
```

In [ ]:
```

```