In [1]:

```
!pip install optuna
```

```
Collecting optuna
  Downloading optuna-2.10.0-py3-none-any.whl (308 kB)
     |████████████████████████████████| 308 kB 7.4 MB/s
Collecting colorlog
  Downloading colorlog-6.6.0-py2.py3-none-any.whl (11 kB)
Collecting alembic
  Downloading alembic-1.7.6-py3-none-any.whl (210 kB)
     |████████████████████████████████| 210 kB 55.5 MB/s
Collecting cliff
  Downloading cliff-3.10.1-py3-none-any.whl (81 kB)
     |████████████████████████████████| 81 kB 9.5 MB/s
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from optun
a) (4.62.3)
Requirement already satisfied: scipy!=1.4.0 in /usr/local/lib/python3.7/dist-packages (fro
m optuna) (1.4.1)
Requirement already satisfied: sqlalchemy>=1.1.0 in /usr/local/lib/python3.7/dist-packages
(from optuna) (1.4.31)
Requirement already satisfied: PyYAML in /usr/local/lib/python3.7/dist-packages (from optu
na) (3.13)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages
(from optuna) (21.3)
Collecting cmaes>=0.8.2
  Downloading cmaes-0.8.2-py3-none-any.whl (15 kB)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (from optun
a) (1.21.5)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-p
ackages (from packaging>=20.0->optuna) (3.0.7)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-package
s (from sqlalchemy>=1.1.0->optuna) (4.11.1)
Requirement already satisfied: greenlet!=0.4.17 in /usr/local/lib/python3.7/dist-packages
(from sqlalchemy>=1.1.0->optuna) (1.1.2)
Requirement already satisfied: importlib-resources in /usr/local/lib/python3.7/dist-packag
es (from alembic->optuna) (5.4.0)
Collecting Mako
  Downloading Mako-1.1.6-py2.py3-none-any.whl (75 kB)
     |████████████████████████████████| 75 kB 4.6 MB/s
Collecting cmd2>=1.0.0
  Downloading cmd2-2.4.0-py3-none-any.whl (150 kB)
     |████████████████████████████████| 150 kB 54.5 MB/s
Collecting pbr!=2.1.0,>=2.0.0
  Downloading pbr-5.8.1-py2.py3-none-any.whl (113 kB)
     |████████████████████████████████| 113 kB 52.9 MB/s
Requirement already satisfied: PrettyTable>=0.7.2 in /usr/local/lib/python3.7/dist-package
s (from cliff->optuna) (3.1.1)
Collecting autopage>=0.4.0
  Downloading autopage-0.5.0-py3-none-any.whl (29 kB)
Collecting stevedore>=2.0.1
  Downloading stevedore-3.5.0-py3-none-any.whl (49 kB)
     |████████████████████████████████| 49 kB 6.4 MB/s
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages
(from cmd2>=1.0.0->cliff->optuna) (3.10.0.2)
Collecting pyperclip>=1.6
  Downloading pyperclip-1.8.2.tar.gz (20 kB)
Requirement already satisfied: wcwidth>=0.1.7 in /usr/local/lib/python3.7/dist-packages (f
rom cmd2>=1.0.0->cliff->optuna) (0.2.5)
Requirement already satisfied: attrs>=16.3.0 in /usr/local/lib/python3.7/dist-packages (fr
om cmd2>=1.0.0->cliff->optuna) (21.4.0)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from i
mportlib-metadata->sqlalchemy>=1.1.0->optuna) (3.7.0)
Requirement already satisfied: MarkupSafe>=0.9.2 in /usr/local/lib/python3.7/dist-packages
(from Mako->alembic->optuna) (2.0.1)
Building wheels for collected packages: pyperclip
  Building wheel for pyperclip (setup.py) ... done
  Created wheel for pyperclip: filename=pyperclip-1.8.2-py3-none-any.whl size=11137 sha256
=2facf09f7975f736bd30c28fdfdfa5de08e6178b10b7218b6ff00cf6b7b758c2
  Stored in directory: /root/.cache/pip/wheels/9f/18/84/8f69f8b08169c7bae2dde6bd7daf0c19fc
```

```
a8c8e500ee620a28
Successfully built pyperclip
Installing collected packages: pyperclip, pbr, stevedore, Mako, cmd2, autopage, colorlog,
cmaes, cliff, alembic, optuna
Successfully installed Mako-1.1.6 alembic-1.7.6 autopage-0.5.0 cliff-3.10.1 cmaes-0.8.2 cm
d2-2.4.0 colorlog-6.6.0 optuna-2.10.0 pbr-5.8.1 pyperclip-1.8.2 stevedore-3.5.0
```

In [2]:

```python
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import os
import optuna
# display all columns of the dataframe
pd.options.display.max_columns = None

# display all rows of the dataframe
pd.options.display.max_rows = None
plt.rcParams['figure.figsize'] = [10,5]
```

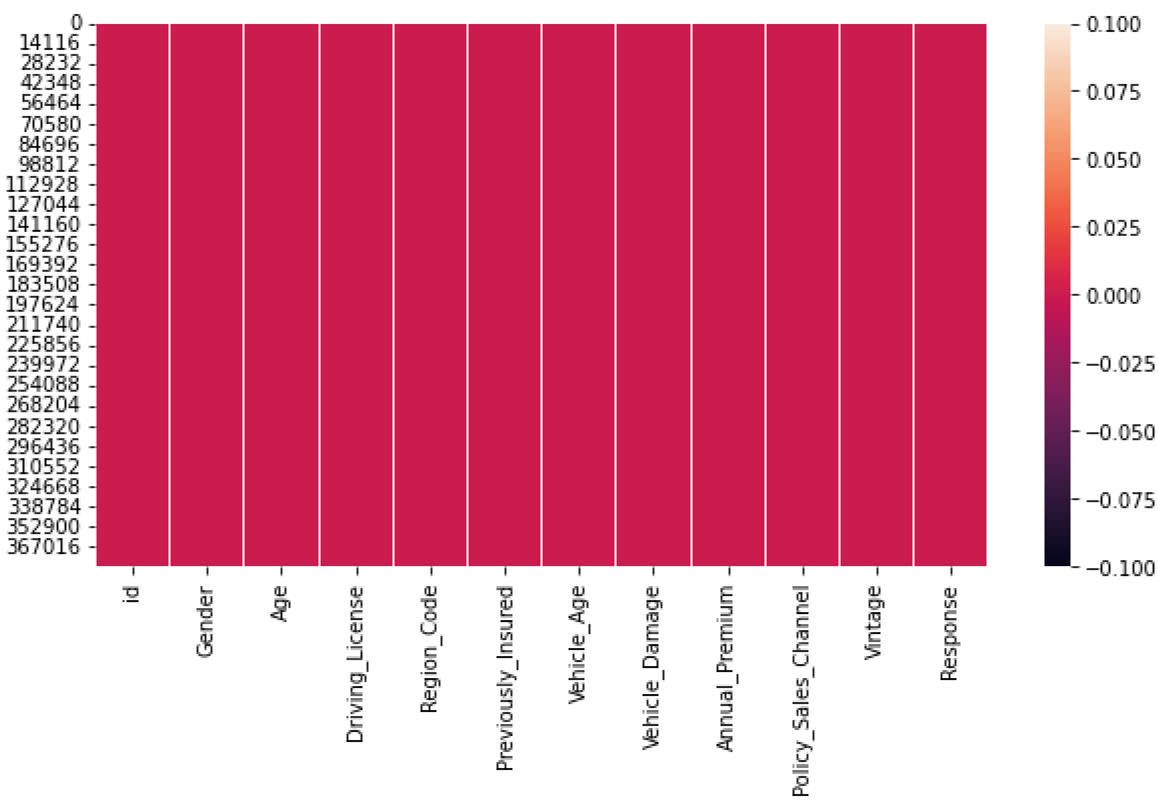
In [3]:

```python
from google.colab import drive
drive.mount('/content/gdrive')
os.chdir("/content/gdrive/My Drive/Capstone")
df=pd.read_csv("DataSet.csv")
```

```
Mounted at /content/gdrive
```

In [4]:

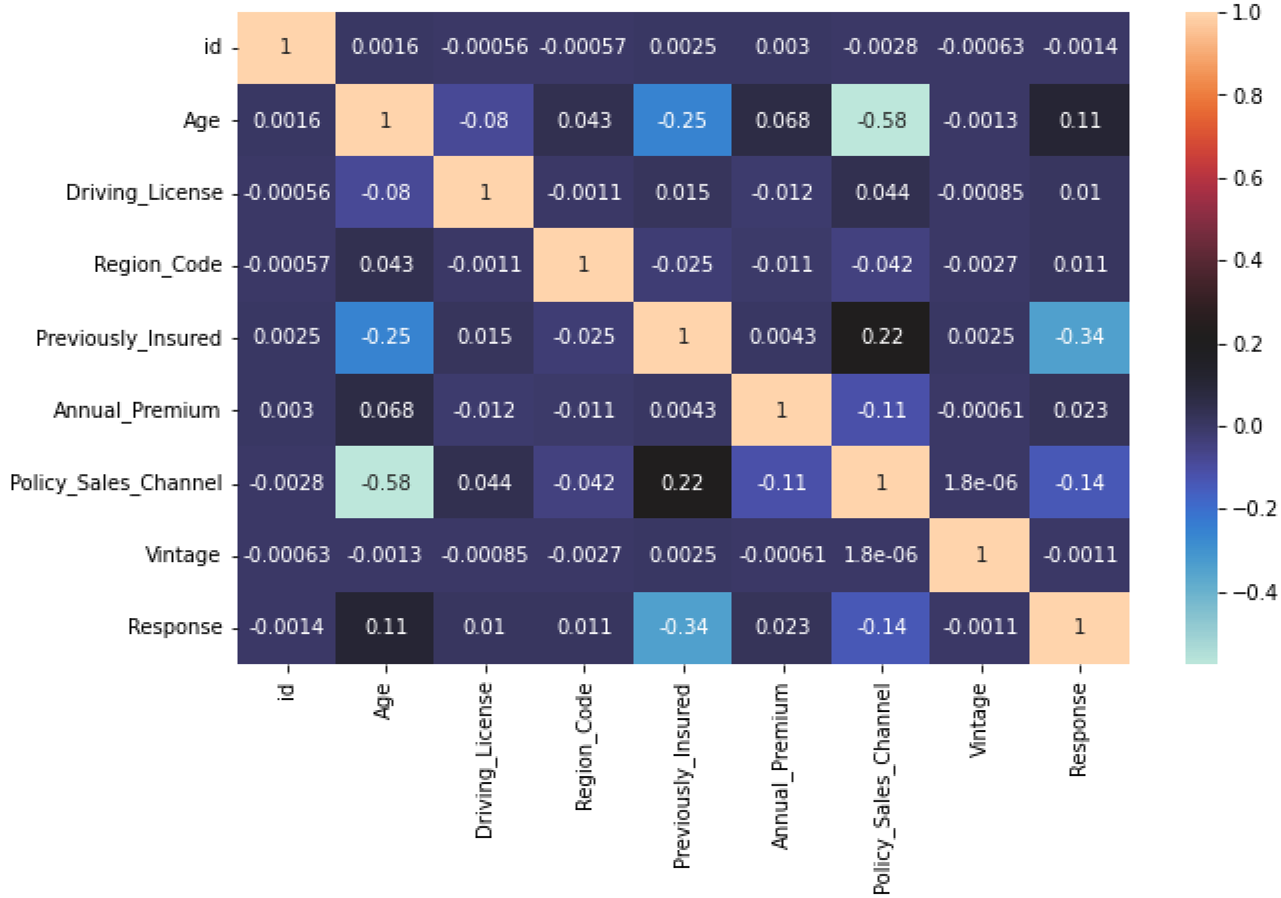
```python
df.shape
#Inference
# It has 381k+ rows and 12 columns.
sns.heatmap(df.isnull())
plt.show()
```



In [5]:

```python
#df.info()
```

```python
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(),annot = True,cmap = "icefire")
plt.show()
# Inference
# From below we can see there are 2 categorical columns and 10 numerical
```



In [6]:

```python
df.describe()
# Inference:
# Since we dont see any nan in below output ,there are no missing values.
# Age varies between 20 and 85 years with median of 36 years old.
# Annual_Premium has high variation as it mean is 30564 , median is 31669 and max value i
# Policy_Sales_Channel varies from  1 to 163,but according to the problem description thi
# Vinatage measures the no of days a customer has been associated with the company.
# Response is the target variable which indicates 1 : Customer is interested, 0 : Custome
```

Out[6]:

	id	Age	Driving_License	Region_Code	Previously_Insured	Annual_Pre
count	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000	381109.(
mean	190555.000000	38.822584	0.997869	26.388807	0.458210	30564.3
std	110016.836208	15.511611	0.046110	13.229888	0.498251	17213.1
min	1.000000	20.000000	0.000000	0.000000	0.000000	2630.(
25%	95278.000000	25.000000	1.000000	15.000000	0.000000	24405.(
50%	190555.000000	36.000000	1.000000	28.000000	0.000000	31669.(
75%	285832.000000	49.000000	1.000000	35.000000	1.000000	39400.(
max	381109.000000	85.000000	1.000000	52.000000	1.000000	540165.(

In [7]:

```python
df.describe(include=object)
# Gender has 2 unique values with more male customers.
```

```
# Vechile_age has 3 unique values with majority of the vehicle age of 1-2 Year.
# Vehicle_Damage has 2 unique values comprising more of damaged vehicles
```

Out[7]:

	Gender	Vehicle_Age	Vehicle_Damage
count	381109	381109	381109
unique	2	3	2
top	Male	1-2 Year	Yes
freq	206089	200316	192413

In [8]:

```
df['id'].nunique()
# All records in this column are unique ,hence we can set index as this column
```

Out[8]:

```
381109
```

In [9]:

```
df['Gender'].value_counts(1)*100
# Here gender is not imbalanced i.e Male percentage is 54.07.
```

Out[9]:

```
Male      54.07613
Female    45.92387
Name: Gender, dtype: float64
```

In [10]:

```
df['Age'].nunique()
# We have 66 unique values for age which can also be categorized further for model optimi
```

Out[10]:

```
66
```

In [11]:

```
df['Driving_License'].value_counts(1)*100
# 0 refers to Customer does not have DL
# 1 : Customer already has DL
# here 99.78% of customers have Driving License
```

Out[11]:

```
1    99.786938
0     0.213062
Name: Driving_License, dtype: float64
```

In [12]:

```
df['Region_Code'].nunique()
# Unique code for the region of the customer.Region_code is incorrectly mapped as float b
# Customers are diversed among 53 regions.
```

Out[12]:

```
53
```

In [13]:

```
df['Region_Code'].value_counts(1)*100
# Customers from 28 region code are densely populated.
```

Out[13]:

```
28.0    27.922458
8.0      8.889058
46.0     5.181982
41.0     4.792067
15.0     3.491914
30.0     3.198822
29.0     2.903369
50.0     2.687683
3.0      2.427390
11.0     2.422404
```

```
36.0     2.308264
33.0     2.008349
47.0     1.951148
35.0     1.821526
6.0      1.647823
45.0     1.470708
37.0     1.443419
18.0     1.352107
48.0     1.228258
14.0     1.227470
39.0     1.218549
10.0     1.147703
21.0     1.119365
2.0      1.059539
13.0     1.059015
7.0      0.860384
12.0     0.839130
9.0      0.813678
27.0     0.740733
32.0     0.731287
43.0     0.692453
17.0     0.686680
26.0     0.678808
25.0     0.656767
24.0     0.633677
38.0     0.531606
0.0      0.530294
16.0     0.526621
31.0     0.514289
23.0     0.514289
20.0     0.507729
49.0     0.480702
4.0      0.472568
34.0     0.436620
19.0     0.402772
22.0     0.343471
40.0     0.339798
5.0      0.335600
1.0      0.264491
44.0     0.212013
42.0     0.155074
52.0     0.070059
51.0     0.048018
Name: Region_Code, dtype: float64
```

In [14]:

```
df['Previously_Insured'].nunique()
# 1 : Customer already has Vehicle Insurance, 0 : Customer doesn't have Vehicle Insurance
# This must be categorical column but it is a int data type which needs to be changed
```

Out[14]:

```
2
```

In [15]:

```
df['Previously_Insured'].value_counts(1)
# Here close to average number of customers do not have vehicle insurance
```

Out[15]:

```
0    0.54179
1    0.45821
Name: Previously_Insured, dtype: float64
```

In [16]:

```
df['Vehicle_Age'].nunique()
#       Vehicle_Age describes the Age of the Vehicle which has  3 categories
```

Out[16]:

```
3
```

In [17]:

```
df['Vehicle_Age'].value_counts(1)*100
```

```
# Here 95% of the customers have vehicle age < 2 years
```

Out[17]:

```
1-2 Year     52.561341
< 1 Year     43.238549
> 2 Years     4.200111
Name: Vehicle_Age, dtype: float64
```

In [18]:

```
df['Vehicle_Damage'].nunique()
# Yes : Customer got his/her vehicle damaged in the past. No : Customer didn't get his/he
```

Out[18]:

```
2
```

In [19]:

```
df['Vehicle_Damage'].value_counts(1)*100
```

Out[19]:

```
Yes    50.487656
No     49.512344
Name: Vehicle_Damage, dtype: float64
```

In [20]:

```
# 50% of the customer vehicles are damaged which can be an important factor for analysis.
```

In [21]:

```
df['Policy_Sales_Channel'].nunique()
# Anonymized Code for the channel of outreaching to the customer ie. Different Agents, Ov
# Dtype for this column mentioned is Float but this is a categorical column describing th
```

Out[21]:

```
155
```

In [22]:

```
df['Policy_Sales_Channel'].value_counts(1)*100
# Here Need to further analyze the data to categorize it.
```

Out[22]:

```
152.0    35.366260
26.0     20.912652
124.0    19.415705
160.0     5.714638
156.0     2.797362
122.0     2.605554
157.0     1.753829
154.0     1.572516
151.0     1.019393
163.0     0.759100
13.0      0.489361
25.0      0.484901
7.0       0.419303
8.0       0.397524
30.0      0.369973
55.0      0.331664
155.0     0.323792
11.0      0.315658
1.0       0.281809
52.0      0.276824
125.0     0.269214
15.0      0.233004
29.0      0.221197
12.0      0.205453
120.0     0.201780
24.0      0.196794
31.0      0.165569
14.0      0.163208
153.0     0.159272
61.0      0.151925
3.0       0.137231
16.0      0.137231
```

60.0	0.135657
4.0	0.133558
158.0	0.129097
23.0	0.110729
22.0	0.087114
150.0	0.081866
10.0	0.069272
19.0	0.058251
136.0	0.048543
147.0	0.048280
109.0	0.045919
145.0	0.045656
9.0	0.044344
18.0	0.043819
91.0	0.041458
116.0	0.040408
37.0	0.039884
21.0	0.038834
139.0	0.037522
128.0	0.035948
42.0	0.034636
59.0	0.033324
138.0	0.032537
131.0	0.031749
127.0	0.028863
140.0	0.028076
113.0	0.027289
119.0	0.027026
44.0	0.026502
135.0	0.026502
54.0	0.026239
64.0	0.023353
133.0	0.022303
148.0	0.020204
35.0	0.019679
103.0	0.018892
111.0	0.017843
56.0	0.017055
121.0	0.016793
47.0	0.016531
132.0	0.016268
65.0	0.015481
107.0	0.014169
106.0	0.013644
36.0	0.013644
159.0	0.013382
86.0	0.012595
45.0	0.012332
94.0	0.012070
129.0	0.011545
108.0	0.009971
88.0	0.008921
53.0	0.008397
93.0	0.007347
20.0	0.007085
90.0	0.006822
92.0	0.006297
114.0	0.006035
78.0	0.006035
130.0	0.005773
98.0	0.005510
32.0	0.005510
48.0	0.005248
63.0	0.004985
66.0	0.004723
118.0	0.004723
46.0	0.004198
146.0	0.004198
96.0	0.004198
17.0	0.004198

```
40.0     0.003936
81.0     0.003673
80.0     0.003673
49.0     0.003673
89.0     0.003673
73.0     0.003411
97.0     0.003411
51.0     0.003149
110.0    0.002886
134.0    0.002624
38.0     0.002624
39.0     0.002624
58.0     0.002362
95.0     0.002362
137.0    0.002099
100.0    0.002099
117.0    0.001837
87.0     0.001837
101.0    0.001837
99.0     0.001837
62.0     0.001574
79.0     0.001574
69.0     0.001574
71.0     0.001312
57.0     0.001312
104.0    0.001312
126.0    0.001312
70.0     0.001050
67.0     0.001050
2.0      0.001050
115.0    0.001050
82.0     0.001050
83.0     0.001050
68.0     0.001050
76.0     0.001050
27.0     0.000787
6.0      0.000787
102.0    0.000787
34.0     0.000787
28.0     0.000787
33.0     0.000787
105.0    0.000787
112.0    0.000525
74.0     0.000525
75.0     0.000525
50.0     0.000525
84.0     0.000262
123.0    0.000262
149.0    0.000262
43.0     0.000262
144.0    0.000262
143.0    0.000262
41.0     0.000262
Name: Policy_Sales_Channel, dtype: float64
```

In [23]:

```
df['Vintage'].nunique()
#Number of Days, Customer has been associated with the company
```

Out[23]:

```
290
```

In [24]:

```
# It has 290 unique values which can also be categorized.
```

In [25]:

```
df['Vintage'].value_counts(1)*100
```

Out[25]:

```
256    0.372072
```

73	0.369973
282	0.366562
158	0.365775
187	0.365250
31	0.364200
226	0.364200
160	0.364200
245	0.363938
131	0.363938
126	0.363675
232	0.363675
298	0.363151
103	0.362888
191	0.362888
215	0.362626
27	0.362626
24	0.361839
65	0.361839
54	0.361576
130	0.361052
197	0.360789
63	0.360789
37	0.360264
42	0.360002
249	0.359477
74	0.358952
284	0.358690
34	0.358428
117	0.358428
76	0.358165
80	0.357903
228	0.357903
263	0.357903
292	0.357640
110	0.357378
92	0.357378
165	0.357116
248	0.357116
195	0.356853
83	0.356591
77	0.356329
241	0.356329
113	0.356066
250	0.356066
56	0.356066
144	0.355804
219	0.355804
200	0.355541
84	0.355279
102	0.354754
90	0.354754
94	0.354492
222	0.354492
147	0.354229
193	0.353967
257	0.353705
173	0.353705
270	0.353705
194	0.353442
151	0.353180
189	0.353180
288	0.352917
11	0.352655
40	0.352393
254	0.352393
251	0.352393
115	0.352393
30	0.352393
227	0.352130
71	0.352130

95	0.352130
230	0.352130
105	0.351868
186	0.351605
49	0.351605
106	0.351605
33	0.351343
216	0.351343
135	0.351343
242	0.351343
268	0.351081
272	0.351081
280	0.350818
69	0.350818
253	0.350818
124	0.350556
142	0.350293
185	0.350293
64	0.350031
123	0.350031
128	0.350031
21	0.348982
13	0.348719
44	0.348719
28	0.348457
100	0.348457
79	0.348457
23	0.348194
273	0.348194
218	0.348194
39	0.348194
22	0.347932
122	0.347932
70	0.347932
293	0.347670
211	0.347670
114	0.347670
91	0.347670
238	0.347407
267	0.347407
136	0.347407
233	0.347145
278	0.347145
177	0.347145
150	0.347145
29	0.346882
172	0.346882
157	0.346882
243	0.346620
35	0.346358
36	0.346358
155	0.346358
107	0.346095
141	0.346095
140	0.346095
98	0.346095
78	0.345570
116	0.345570
109	0.345570
182	0.345308
146	0.345308
213	0.345308
16	0.345046
204	0.344783
20	0.344783
240	0.344783
145	0.344521
119	0.344521
81	0.344521
99	0.344258

179	0.344258
53	0.344258
152	0.343996
10	0.343996
47	0.343996
259	0.343996
217	0.343734
57	0.343734
111	0.343471
46	0.343471
68	0.343209
223	0.343209
120	0.343209
125	0.343209
281	0.342947
190	0.342947
202	0.342947
255	0.342947
88	0.342947
129	0.342684
52	0.342684
234	0.342684
283	0.342684
208	0.342422
75	0.342422
291	0.342422
25	0.342422
59	0.342159
161	0.342159
174	0.342159
121	0.341897
96	0.341897
38	0.341635
296	0.341635
201	0.341635
221	0.341635
192	0.341372
206	0.341372
246	0.341110
207	0.341110
163	0.341110
58	0.341110
43	0.340847
169	0.340847
258	0.340847
148	0.340585
138	0.340585
62	0.340585
198	0.340585
162	0.340585
275	0.340585
209	0.340323
87	0.340323
167	0.340323
86	0.340060
15	0.339798
261	0.339798
271	0.339535
166	0.339535
199	0.339535
51	0.339535
156	0.339535
236	0.339273
26	0.339273
133	0.339011
159	0.339011
149	0.338748
244	0.338486
101	0.338486
143	0.338486

45	0.338223
203	0.337961
66	0.337699
132	0.337174
184	0.337174
180	0.337174
188	0.337174
297	0.336911
112	0.336911
85	0.336649
299	0.336649
41	0.336387
55	0.336387
269	0.336387
175	0.336387
134	0.336124
154	0.336124
153	0.336124
294	0.336124
170	0.335862
276	0.335862
210	0.335600
229	0.335600
289	0.335600
262	0.335600
266	0.335600
60	0.335600
274	0.335337
139	0.335337
196	0.335075
183	0.335075
286	0.335075
61	0.335075
127	0.334812
252	0.334812
260	0.334812
178	0.334812
295	0.334550
214	0.334550
17	0.334288
231	0.334288
287	0.333763
171	0.333763
168	0.333763
247	0.333500
67	0.333500
265	0.333500
82	0.333238
290	0.332976
220	0.332976
212	0.332713
239	0.332451
164	0.332188
72	0.331401
285	0.331401
108	0.331139
97	0.331139
137	0.330614
14	0.330614
12	0.329827
93	0.329827
176	0.329565
279	0.329565
225	0.329565
237	0.329302
118	0.327990
48	0.327203
104	0.327203
264	0.327203
235	0.327203

```
18     0.326941
19     0.326941
50     0.326678
181    0.326153
205    0.324054
89     0.323792
32     0.322742
224    0.321955
277    0.321693
Name: Vintage, dtype: float64
```

In [26]:

```
df['Response'].nunique()
# 1 : Customer is interested, 0 : Customer is not interested
```

Out[26]:

```
2
```

In [27]:

```
df['Response'].value_counts(1)*100
# Target variable is imbalanced with 87 % of customers not interested in vehicle insuranc
```

Out[27]:

```
0    87.743664
1    12.256336
Name: Response, dtype: float64
```

In [28]:

```
# Univariate Analysis
```

In [29]:

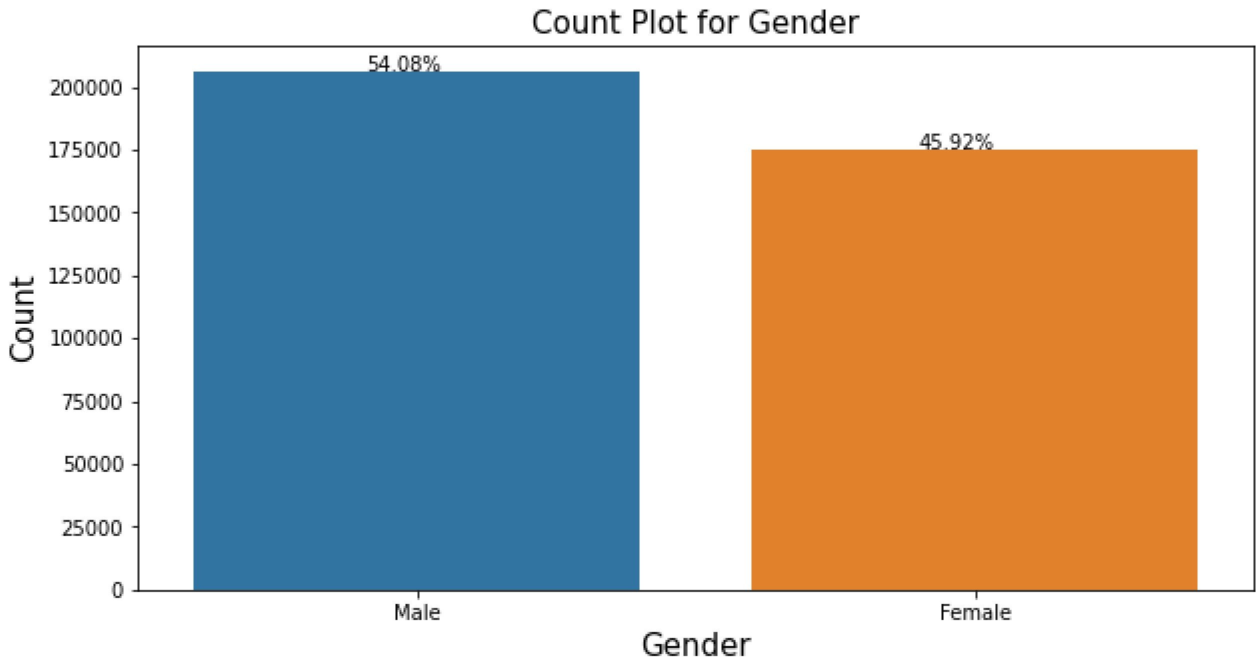
```
df.columns
```

Out[29]:

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

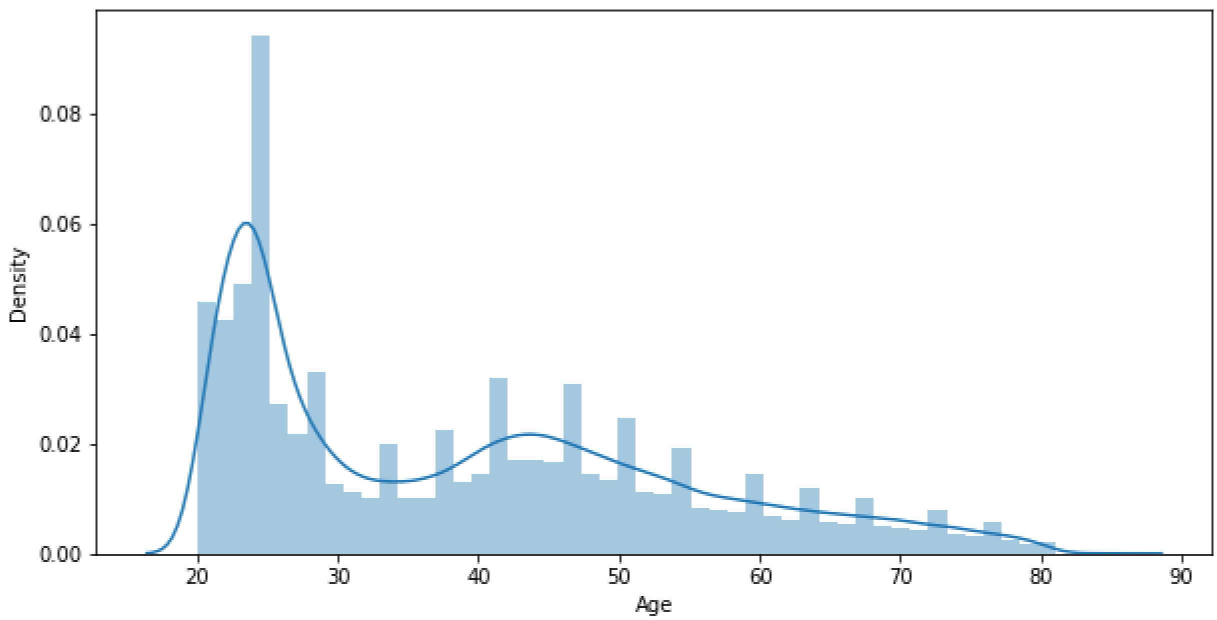
In [30]:

```
sns.countplot(df['Gender'])
plt.text(x = -0.09, y = df['Gender'].value_counts()[0] , s = str(round((df['Gender'].valu
plt.text(x = 0.90, y = df['Gender'].value_counts()[1] , s = str(round((df['Gender'].value_
plt.title('Count Plot for Gender', fontsize = 15)
plt.xlabel('Gender', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.show()
# Insurance company has more male customers compared to female
```



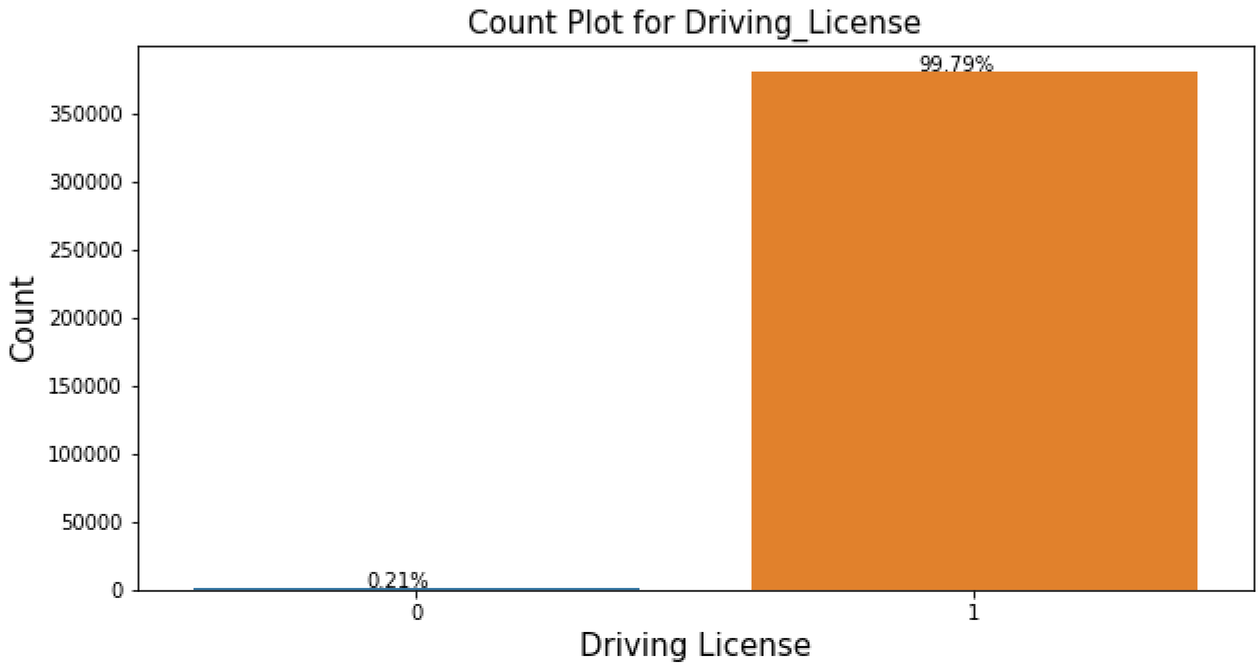
In [31]:

```
sns.distplot(df['Age'])
plt.show()
 # Customer age varying 20 years to 88 years (Right Skewed).
 # Majority of customers from 20-30 age group
```



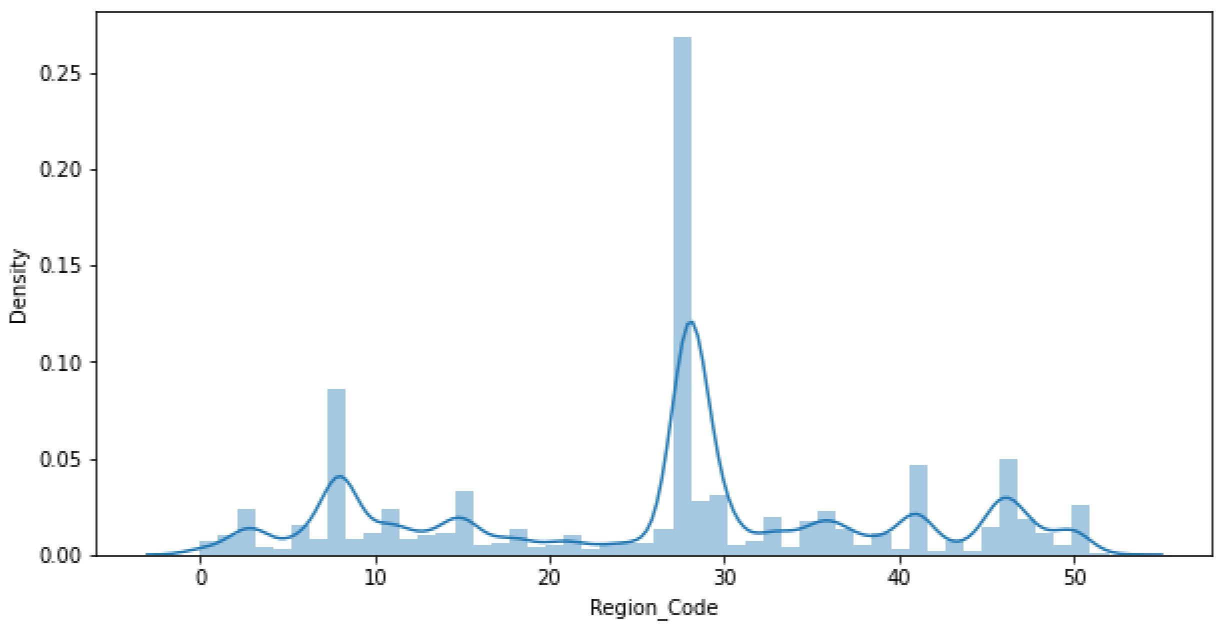
In [32]:

```
sns.countplot(df['Driving_License'])
plt.text(x = -0.09, y = df['Driving_License'].value_counts()[0] , s = str(round((df['Driv
plt.text(x = 0.90, y = df['Driving_License'].value_counts()[1] , s = str(round((df['Drivi
plt.title('Count Plot for Driving_License', fontsize = 15)
plt.xlabel('Driving License', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.show()
# Majority of Customers have Driving License.
```



In [33]:

```
sns.distplot(df['Region_Code'])
plt.show()
# Data spread is almost equal among all,except region code 28.
```

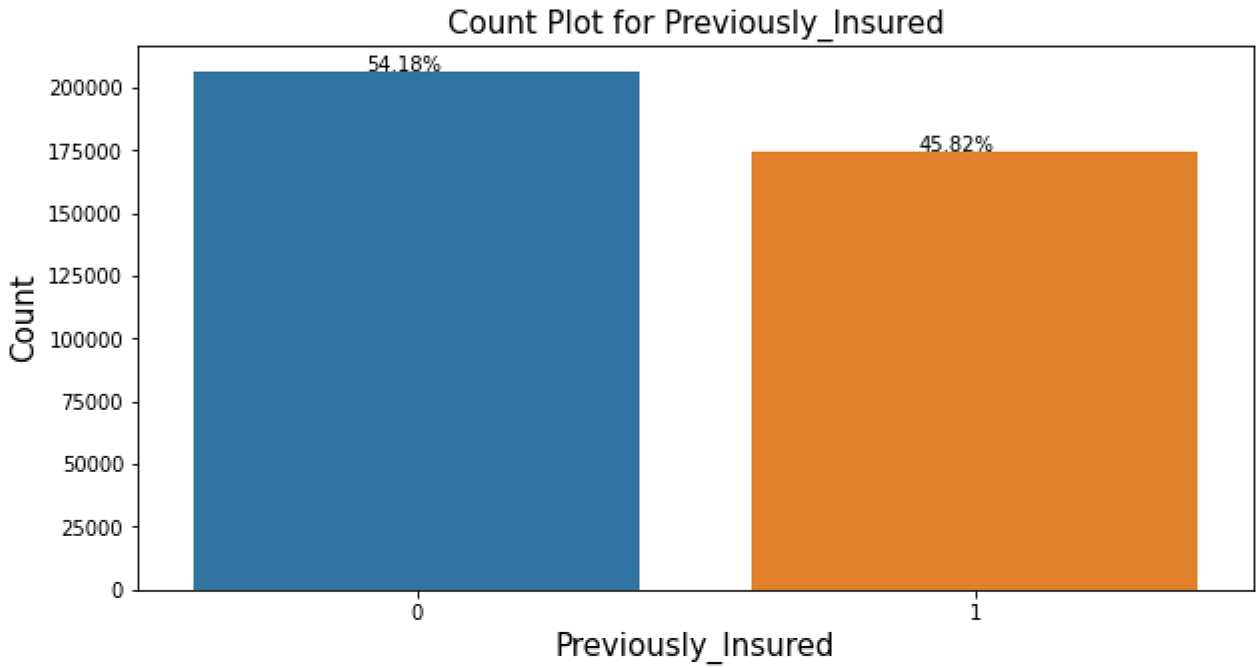


In [34]:

```
    # 'Previously_Insured', 'Vehicle_Age', 'Vehicle_Damage', 'Annual_Premium',
      #  'Policy_Sales_Channel', 'Vintage', 'Response'],
```

In [35]:

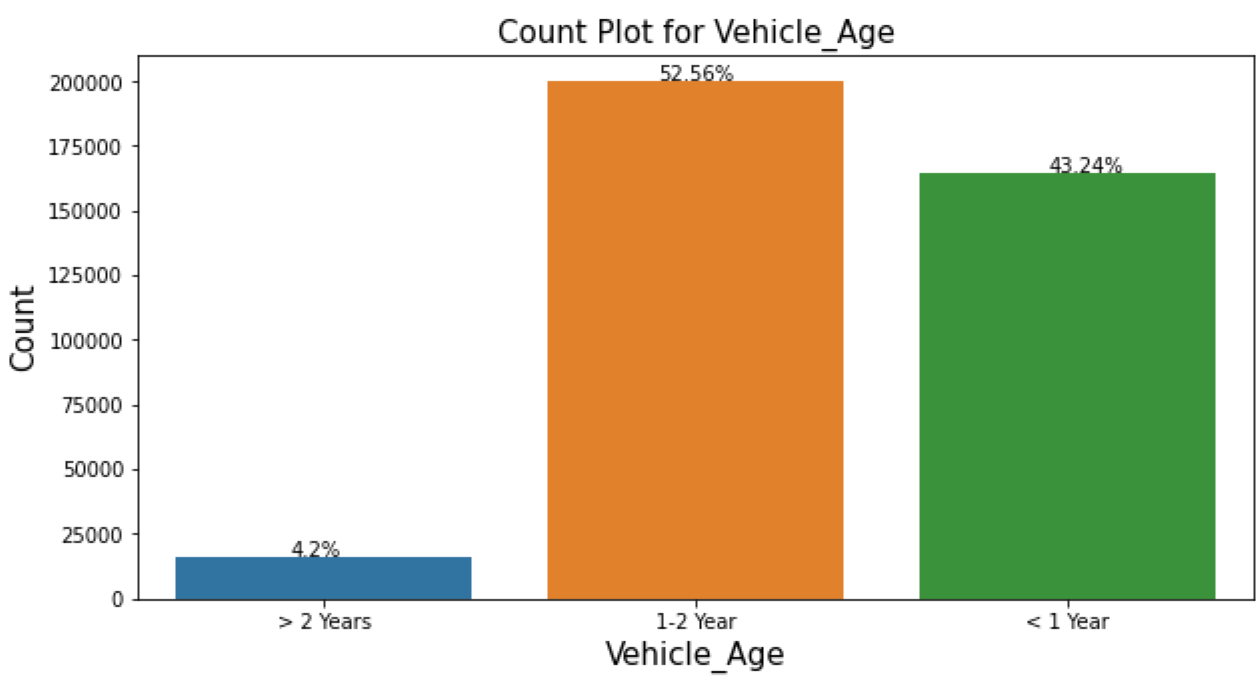
```
sns.countplot(df['Previously_Insured'])
plt.text(x = -0.09, y = df['Previously_Insured'].value_counts()[0] , s = str(round((df['P
plt.text(x = 0.90, y = df['Previously_Insured'].value_counts()[1] , s = str(round((df['Pr
plt.title('Count Plot for Previously_Insured', fontsize = 15)
plt.xlabel('Previously_Insured', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.show()
# More than 50% of the customers have not insured previously
```



In [36]:
```
df["Vehicle_Age"].value_counts(1)
```

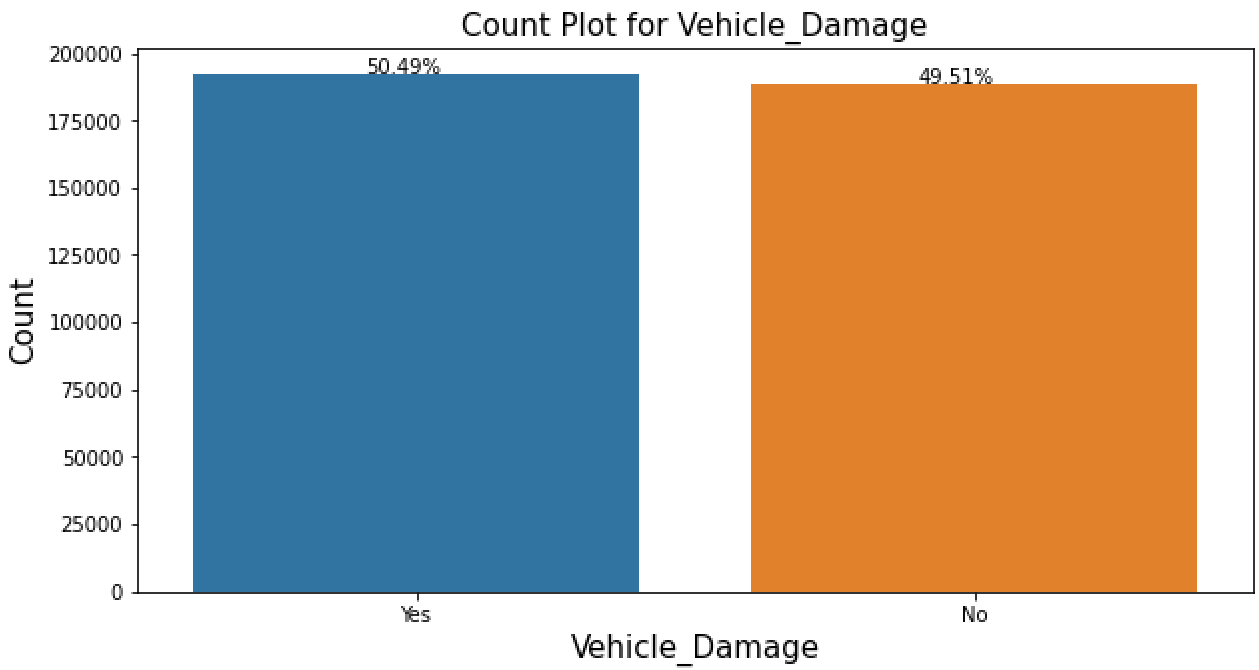
Out[36]:
```
1-2 Year     0.525613
< 1 Year     0.432385
> 2 Years    0.042001
Name: Vehicle_Age, dtype: float64
```

In [37]:
```
sns.countplot(df['Vehicle_Age'])
plt.text(x = -0.09, y = df['Vehicle_Age'].value_counts()[2] , s = str(round((df['Vehicle_A
plt.text(x = 0.90, y = df['Vehicle_Age'].value_counts()[0] , s = str(round((df['Vehicle_Ag
plt.text(x = 1.95, y = df['Vehicle_Age'].value_counts()[1] , s = str(round((df['Vehicle_Ag
plt.title('Count Plot for Vehicle_Age', fontsize = 15)
plt.xlabel('Vehicle_Age', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.show()
#52.56% of the customers has a vehicle age of 1-2 year
```



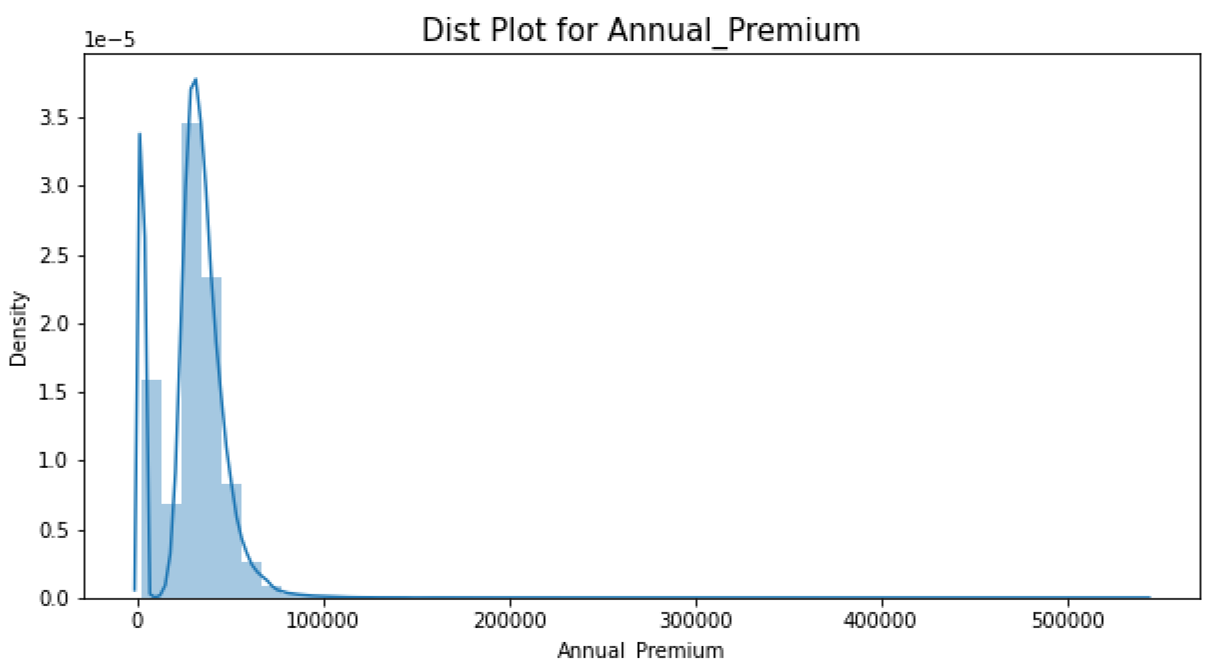
In [38]:
```
sns.countplot(df['Vehicle_Damage'])
```

```
plt.text(x = -0.09, y = df['Vehicle_Damage'].value_counts()[0] , s = str(round((df['Vehic
plt.text(x = 0.9, y = df['Vehicle_Damage'].value_counts()[1] , s = str(round((df['Vehicle_
plt.title('Count Plot for Vehicle_Damage', fontsize = 15)
plt.xlabel('Vehicle_Damage', fontsize = 15)
plt.ylabel('Count', fontsize = 15)
plt.show()
# 50% of the customers vehicle are damaged
```



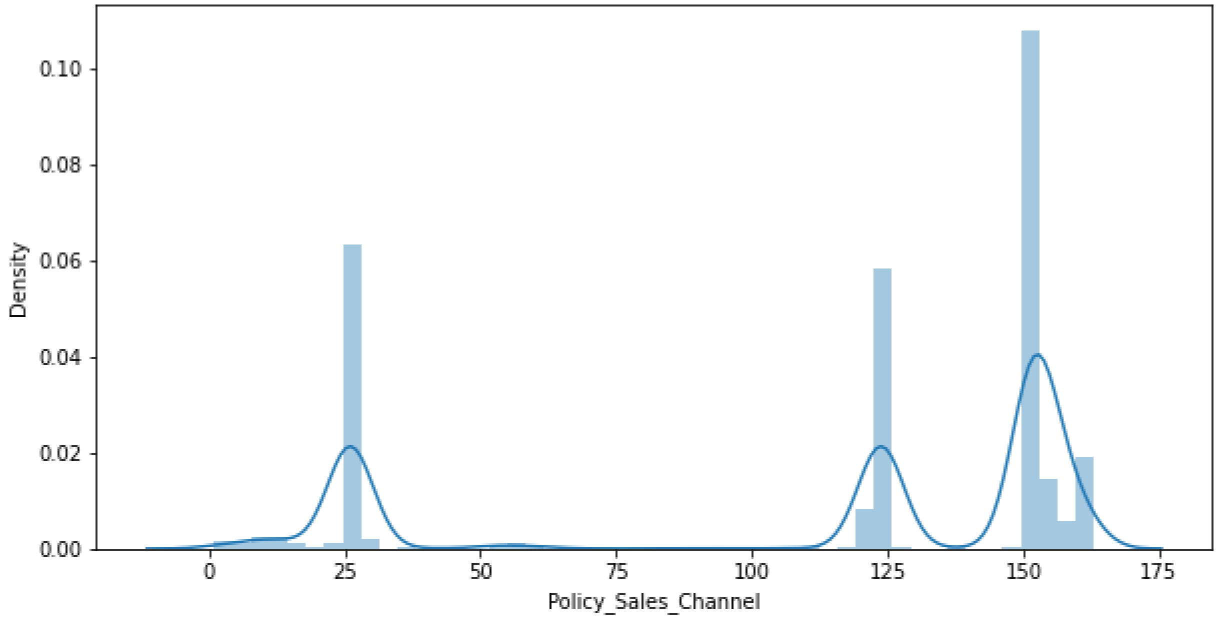
In [39]:

```
sns.distplot(df['Annual_Premium'])
plt.title('Dist Plot for Annual_Premium', fontsize = 15)
plt.show()
# Data is Right skewed.
# Majority of customers have annual premium < 100k
```

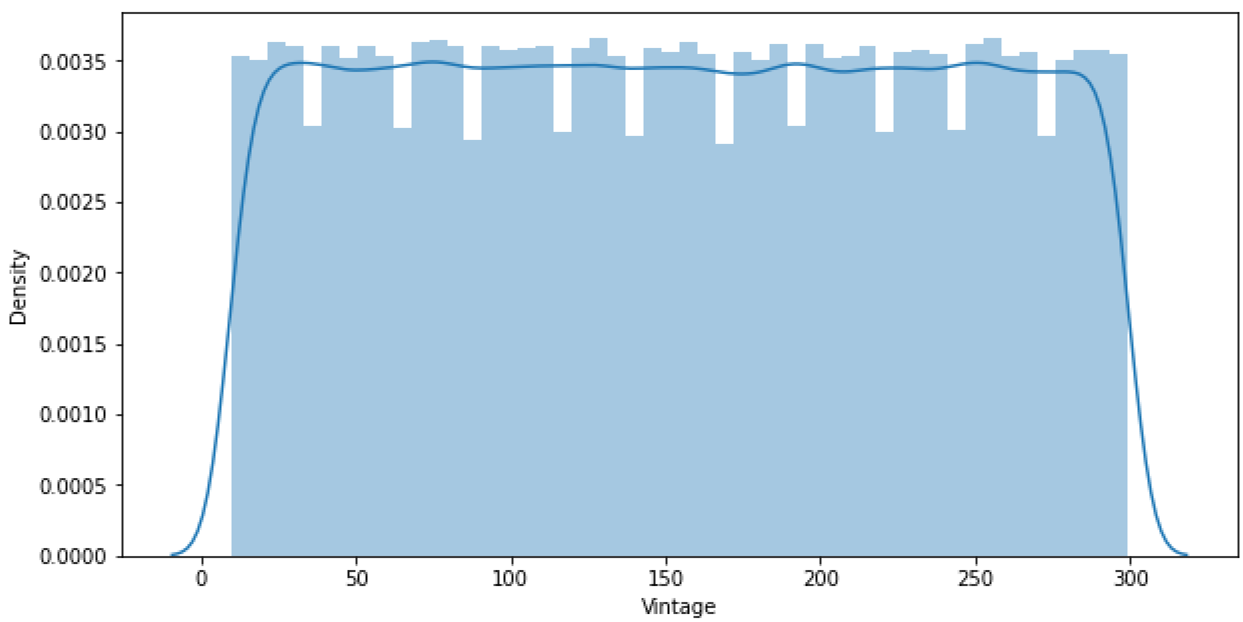


In [40]:

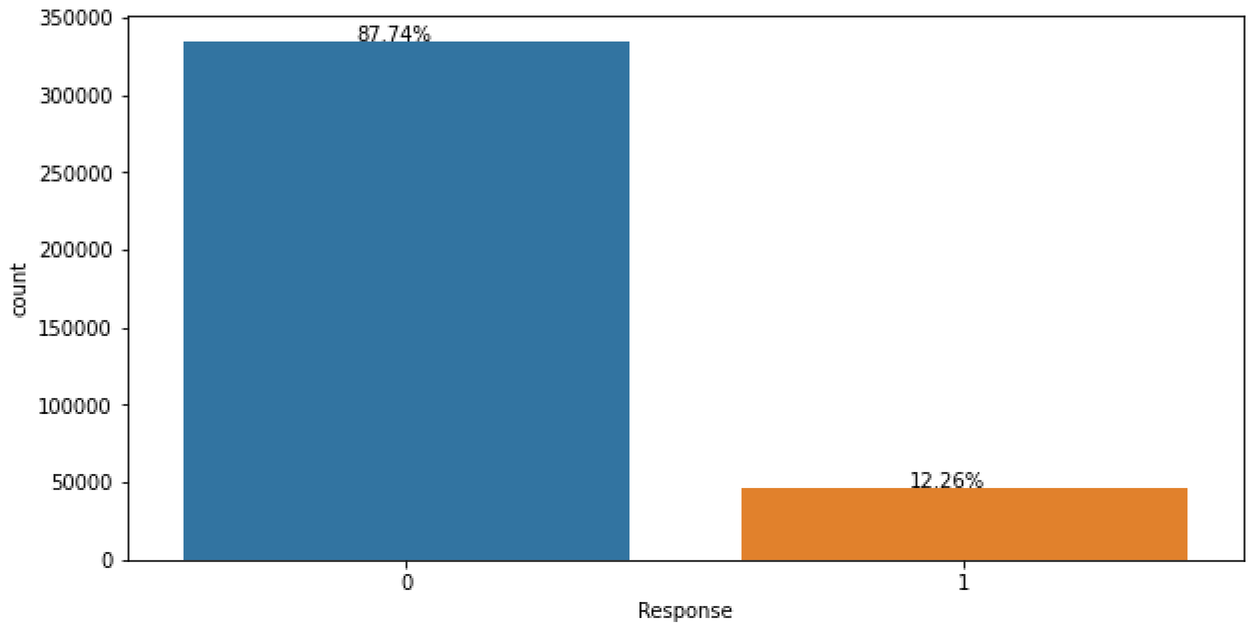
```
sns.distplot(df['Policy_Sales_Channel'])
plt.show()
# From below we can say that 25,125 and ~150 are major contributing policy sales channel
```



In [41]:
```
sns.distplot(df['Vintage'])
plt.show()
# Distribution is uniform
```



In [42]:
```
sns.countplot(df['Response'])
plt.text(x = -0.09, y = df['Response'].value_counts()[0] , s = str(round((df['Response'].v
plt.text(x = 0.9, y = df['Response'].value_counts()[1] , s = str(round((df['Response'].va
plt.show()
# Target variable is imbalanced
```



In [43]:
```
df['Driving_License']= df['Driving_License'].astype(object)
df['Previously_Insured']= df['Previously_Insured'].astype(object)
df['Region_Code']= df['Region_Code'].astype(object)
df['Policy_Sales_Channel']= df['Policy_Sales_Channel'].astype(object)
```

In [44]:
```
(df['Policy_Sales_Channel'].value_counts(1)*100).head()
```

Out[44]:
```
152.0    35.366260
26.0     20.912652
124.0    19.415705
160.0     5.714638
156.0     2.797362
Name: Policy_Sales_Channel, dtype: float64
```

In [45]:
```
df['Policy_Sales_Channel']= df['Policy_Sales_Channel'].replace({152:0,26:1,124:2})
```

In [46]:
```
idx=df[df['Policy_Sales_Channel'] > 2].index
df.loc[idx,'Policy_Sales_Channel']= 3
```

In [47]:
```
(df['Policy_Sales_Channel'].value_counts(1)*100).head()
```

Out[47]:
```
0.0    35.366260
3.0    24.022524
1.0    21.194461
2.0    19.416755
Name: Policy_Sales_Channel, dtype: float64
```

In [48]:
```
df.set_index(keys='id',inplace=True)
```

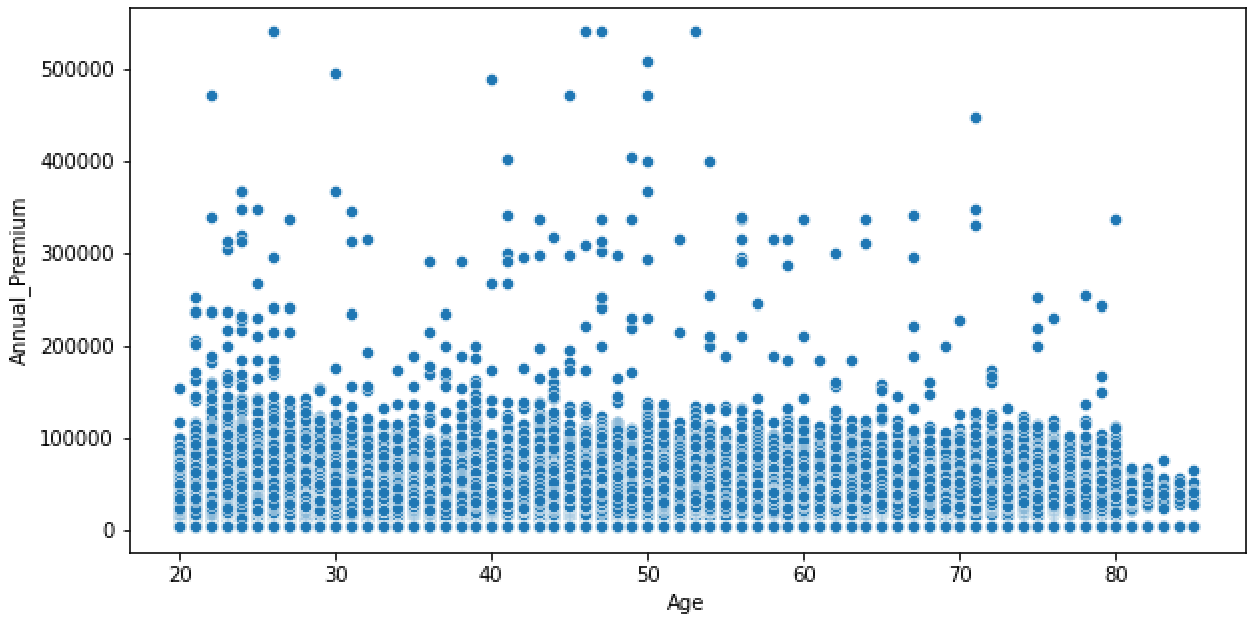
In [49]:
```
# Bi variate Analysis
# Numerical vs Numerical
```

In [50]:
```
df_num = df.select_dtypes(include=np.number)
df_cat = df.select_dtypes(exclude=np.number)
```

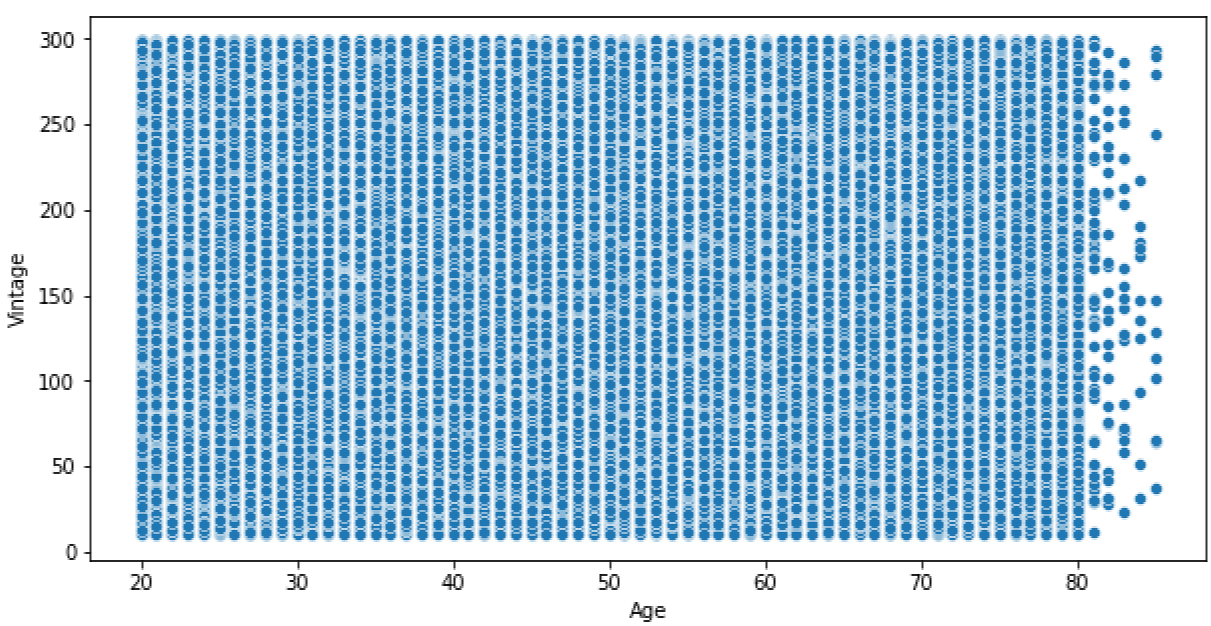
In [51]:
```
df_num.columns
```

Out[51]:
```
Index(['Age', 'Annual_Premium', 'Policy_Sales_Channel', 'Vintage', 'Response'], dtype='obj
ect')
```

In [52]:
```
sns.scatterplot(df_num['Age'],df_num['Annual_Premium'])
plt.show()
# there is no relation between age and Annual_Premium
```

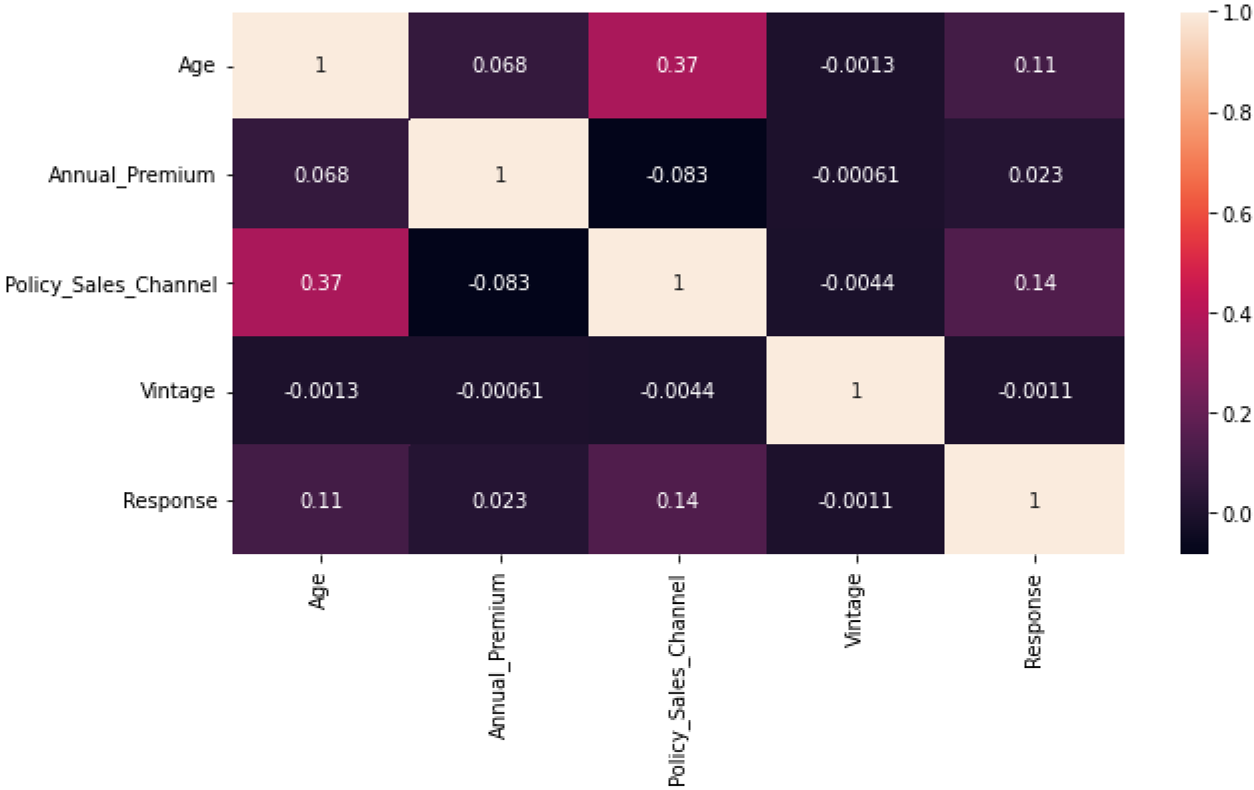


In [53]:
```
sns.scatterplot(df_num['Age'],df_num['Vintage'])
plt.show()
#there is no relation between age and Vintage
```



In [54]:
```
# There is no relationship between age and Vintage
```

In [55]:
```
sns.heatmap(df_num.corr(),annot=True)
plt.show()
# We dont have any strong correlation between the independent and dependent features
```

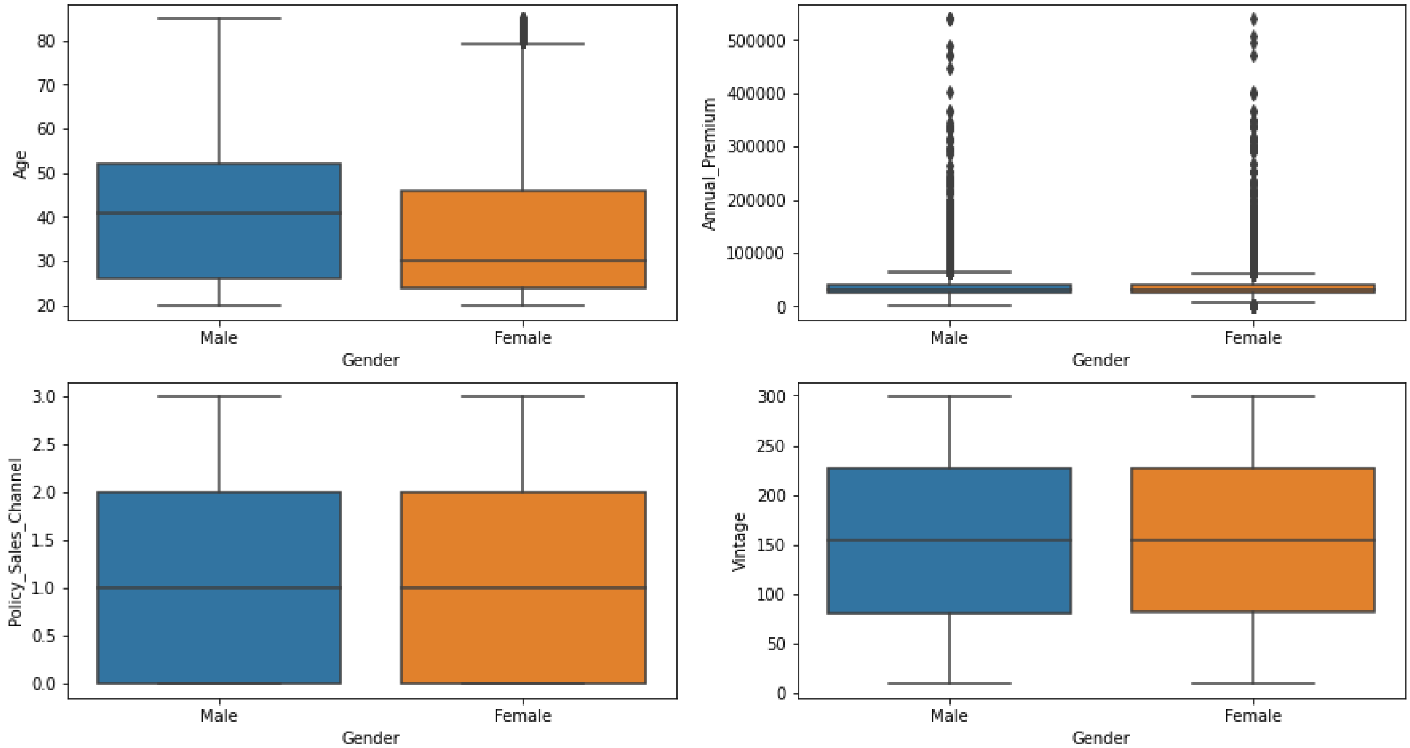


Bivariate Analysis (Numerical vs Categorical)

In [56]:
```
df_num.columns,df_cat.columns
```

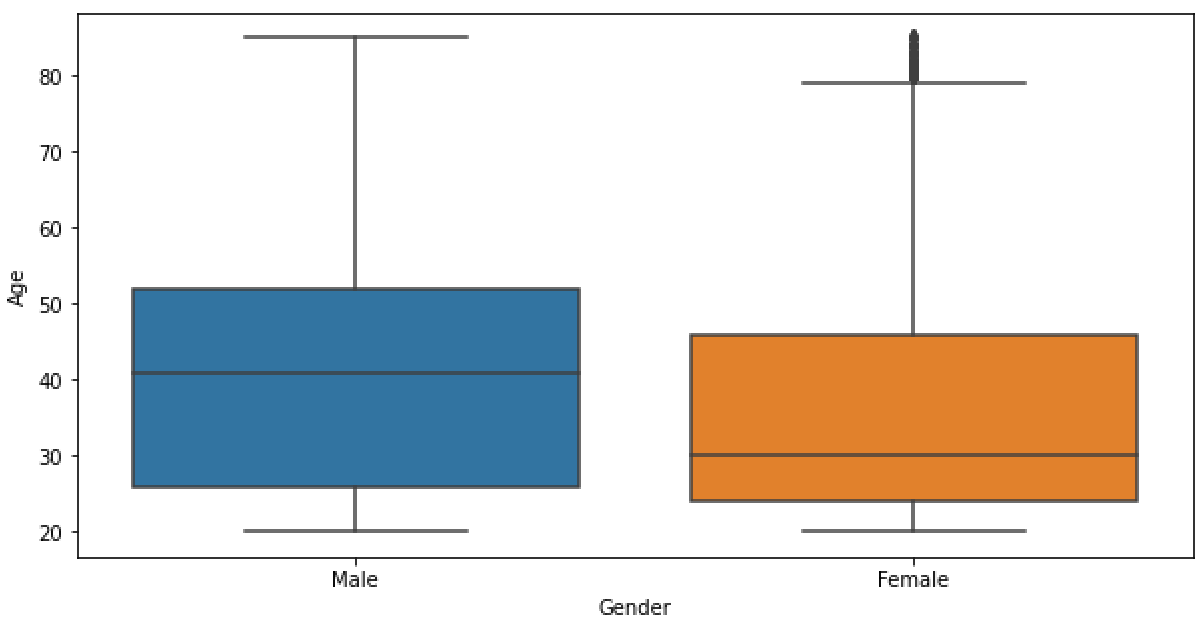
Out[56]:
```
(Index(['Age', 'Annual_Premium', 'Policy_Sales_Channel', 'Vintage', 'Response'], dtype='ob
ject'),
 Index(['Gender', 'Driving_License', 'Region_Code', 'Previously_Insured',
        'Vehicle_Age', 'Vehicle_Damage'],
       dtype='object'))
```

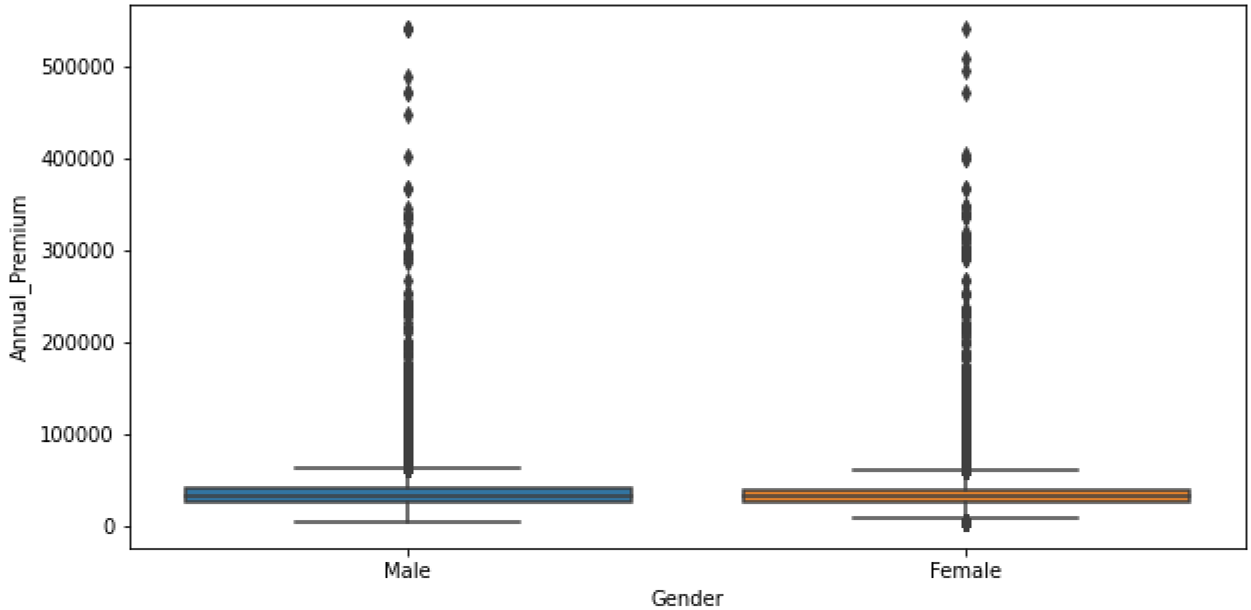
In [57]:
```
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize=(15, 8))
for variable, subplot in zip(df_num.columns, ax.flatten()):
  if variable == "Response":
    continue
  else:
    sns.boxplot(x = df["Gender"],y = df_num[variable], ax = subplot)
plt.show()
```

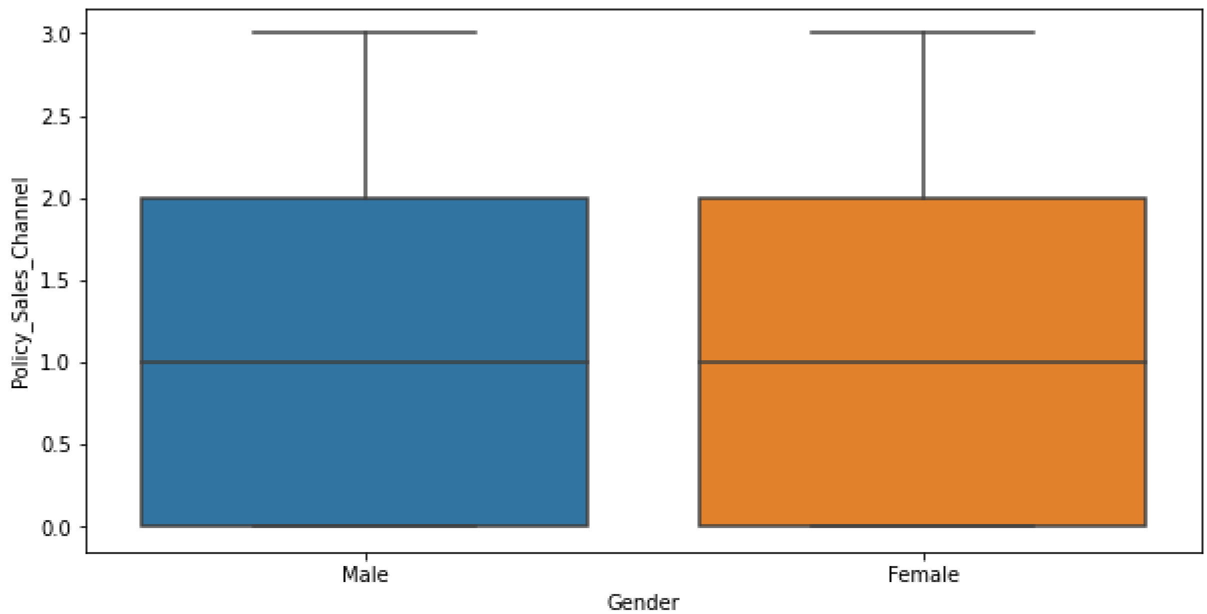


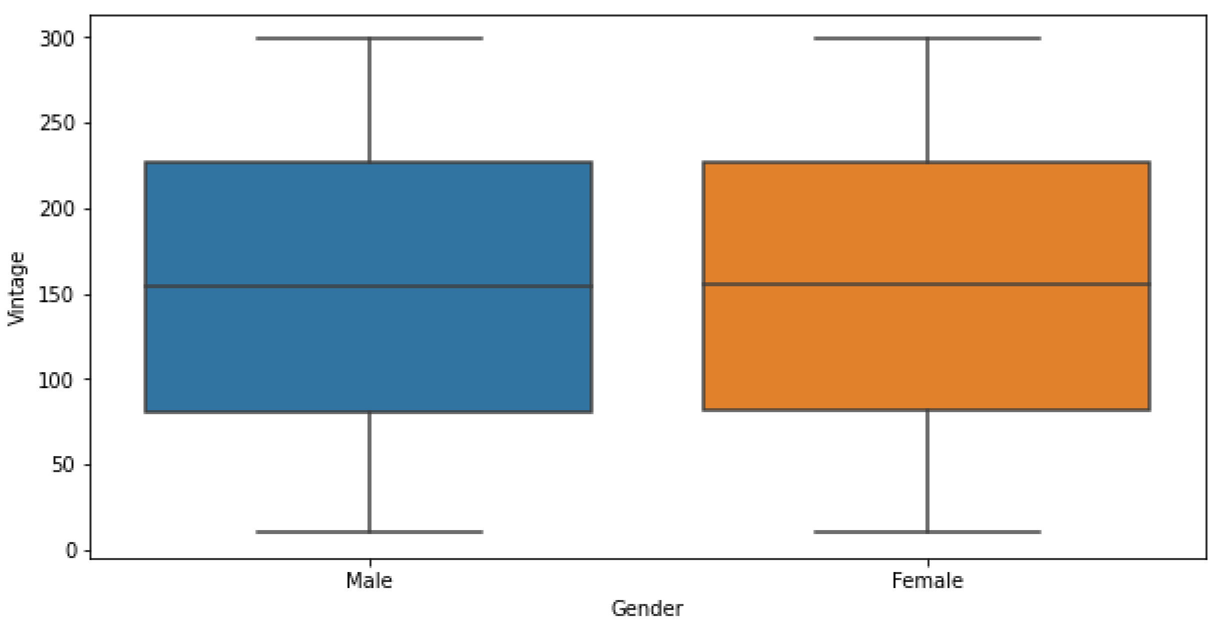
In [58]:

```
for i in df_num.columns:
  if i =='Response':
    continue
  else:
    sns.boxplot(x= df['Gender'],y=df[i])
    plt.show()
```









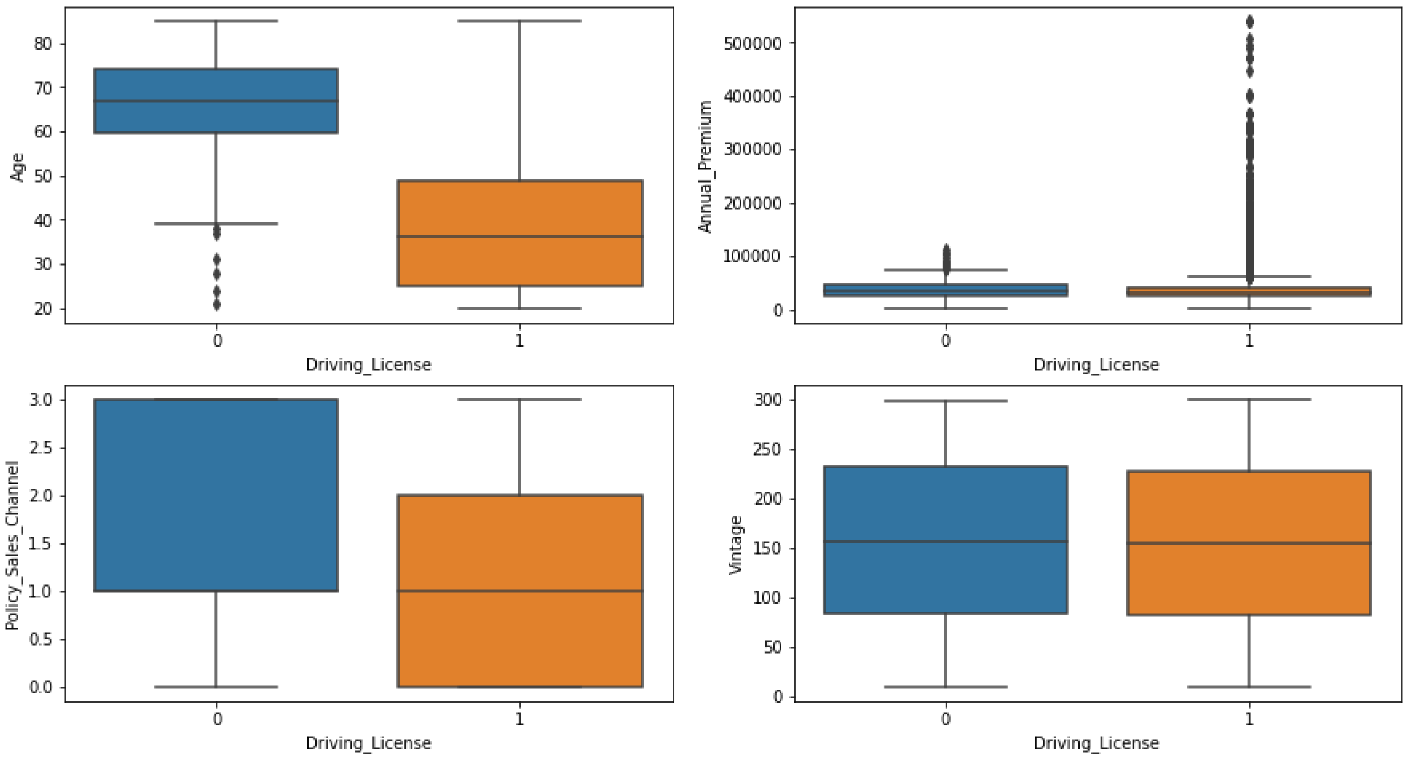
Inference:(Gender vs Numerical)

1.Average Age for male is more as compared to Female.

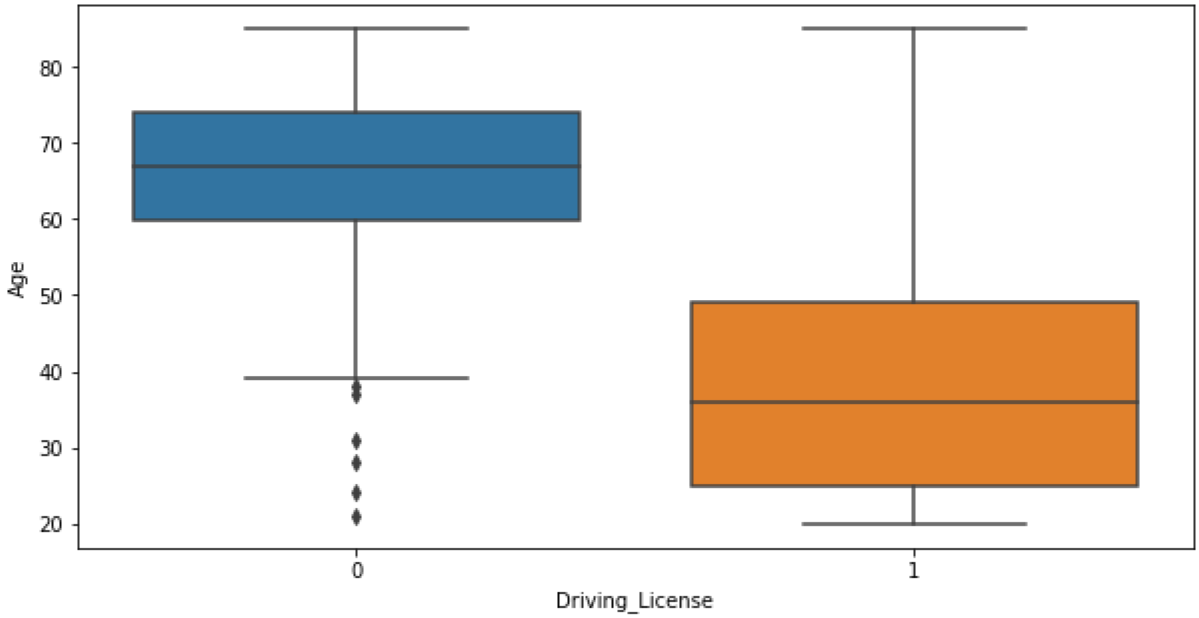
2.Average Annual Premium for male is same as compared to Female.

3.Average number of days with in the company is same for Male and Female.

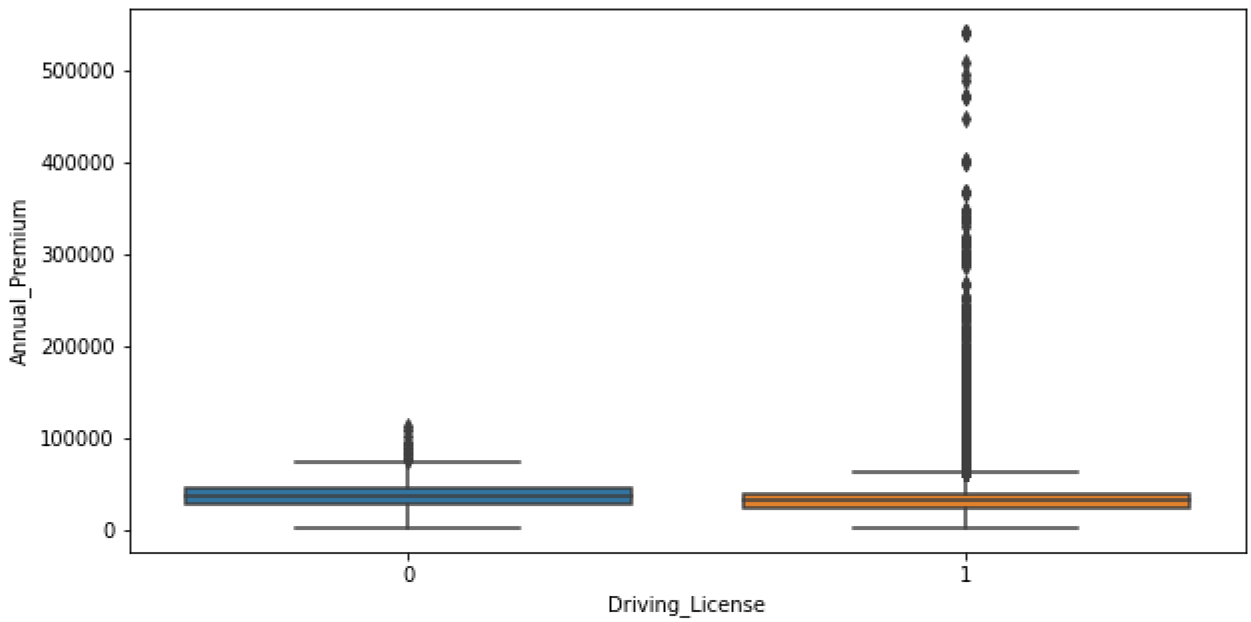
In [59]:
```
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize=(15, 8))
for variable, subplot in zip(df_num.columns, ax.flatten()):
  if variable == "Response":
    continue
  else:
    sns.boxplot(x = df["Driving_License"],y = df_num[variable], ax = subplot)
plt.show()
```



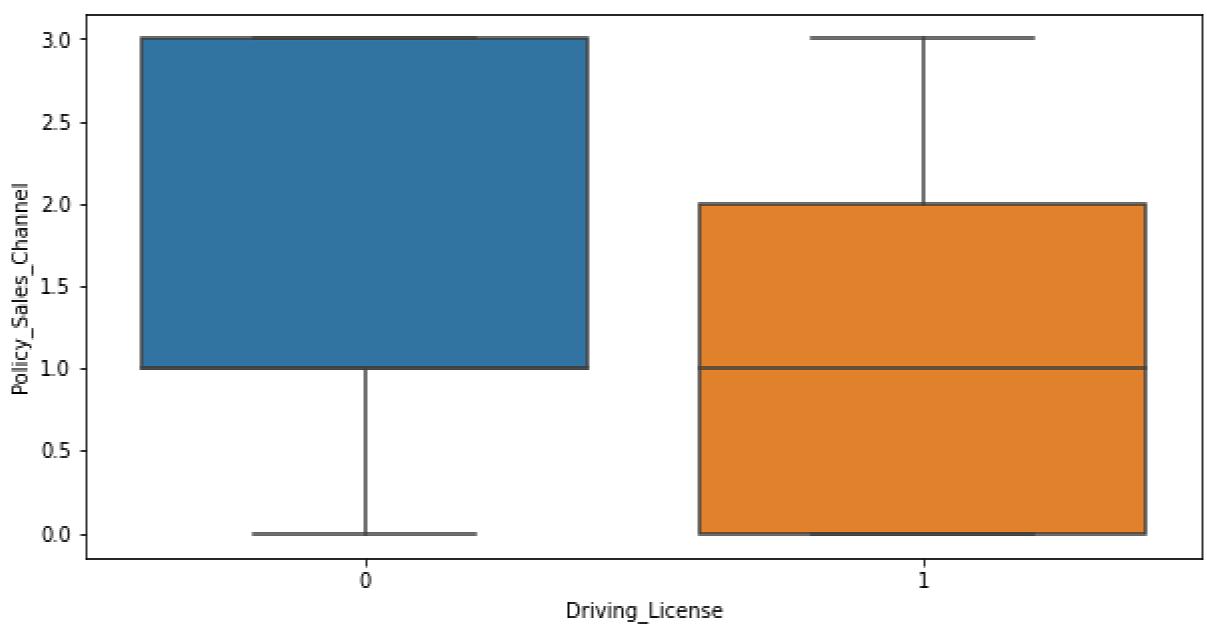
In [60]:
```
for i in df_num.columns:
  if i =='Response':
    continue
  else:
    sns.boxplot(x= df['Driving_License'],y=df[i])
    plt.show()
```



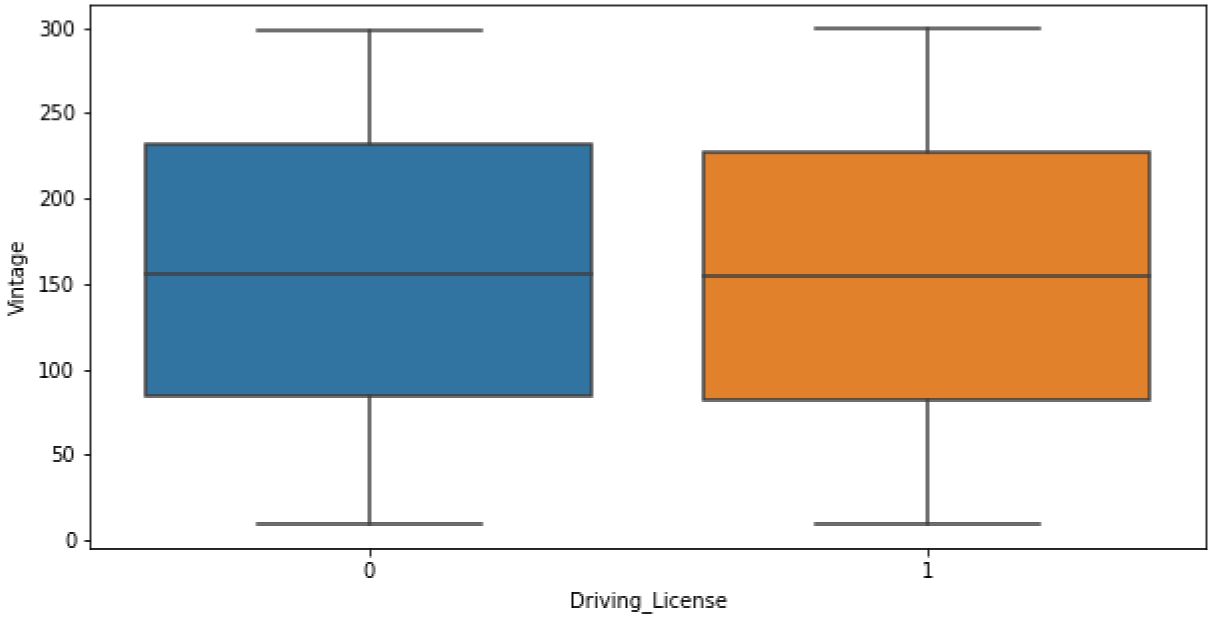
80
70
60
50
40
30
20
Age
0
1
Driving_License



500000
400000
300000
200000
100000
0
Annual_Premium
0
1
Driving_License



3.0
2.5
2.0
1.5
1.0
0.5
0.0
Policy_Sales_Channel
0
1
Driving_License

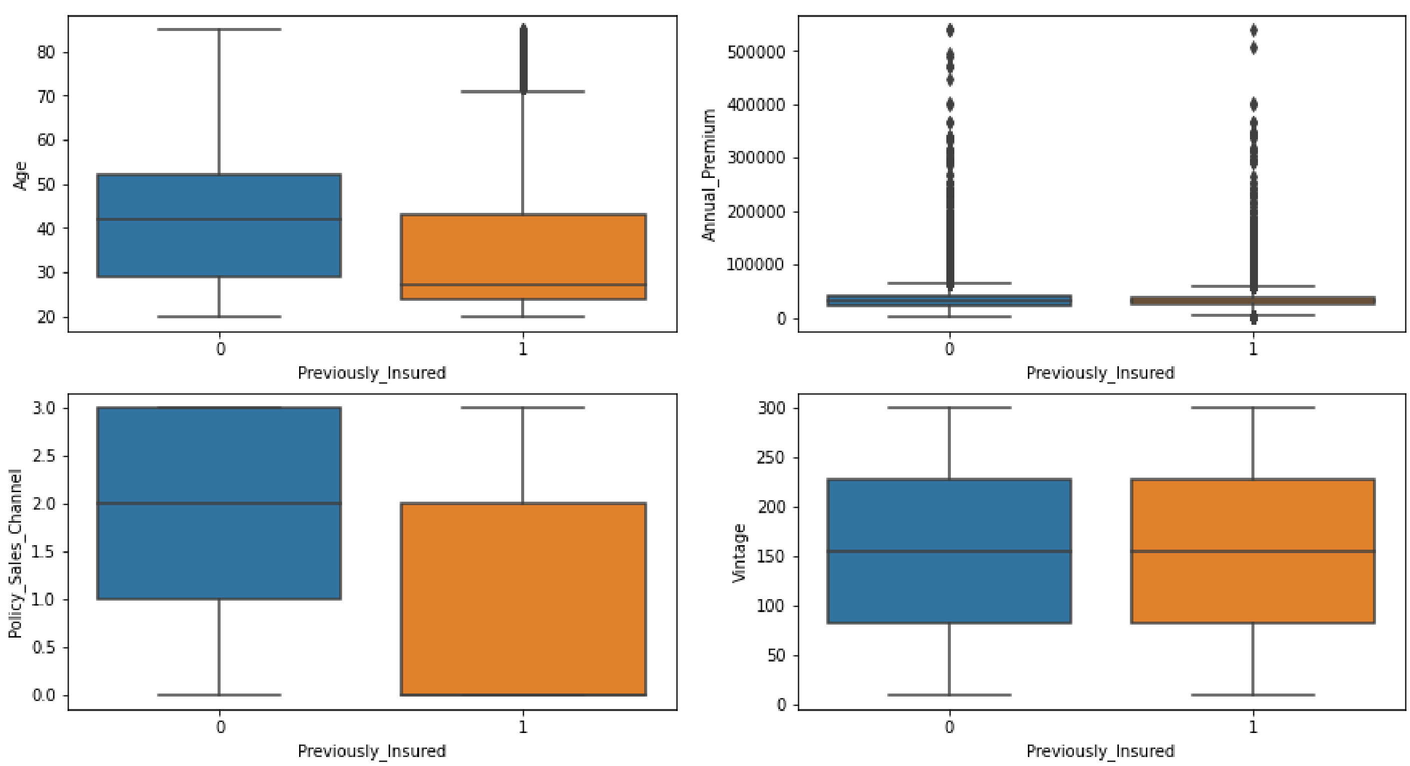


Inference (Driving License vs Numerical)

1. Customers of Age group between 25 to 50 have Driving License.
2. Customers having driving license have high annaul premium.
3. Average number of days associated with the company is same for customers having driving license and not having driving license.

In [61]:

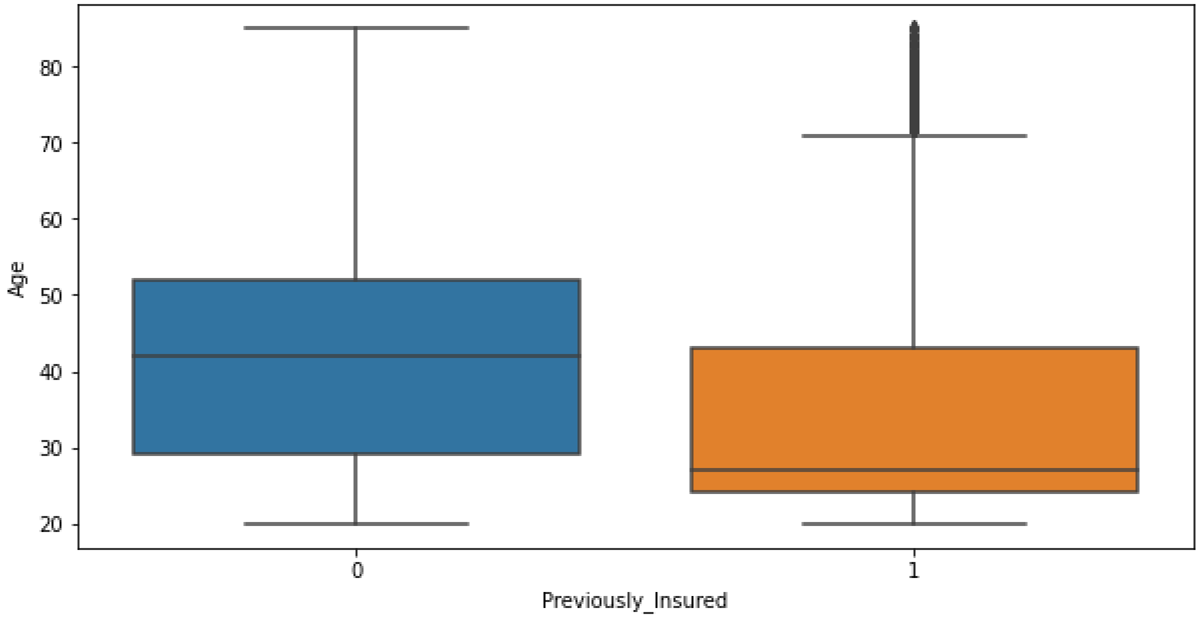
```
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize=(15, 8))
for variable, subplot in zip(df_num.columns, ax.flatten()):
  if variable == "Response":
    continue
  else:
    sns.boxplot(x = df["Previously_Insured"],y = df_num[variable], ax = subplot)
plt.show()
```

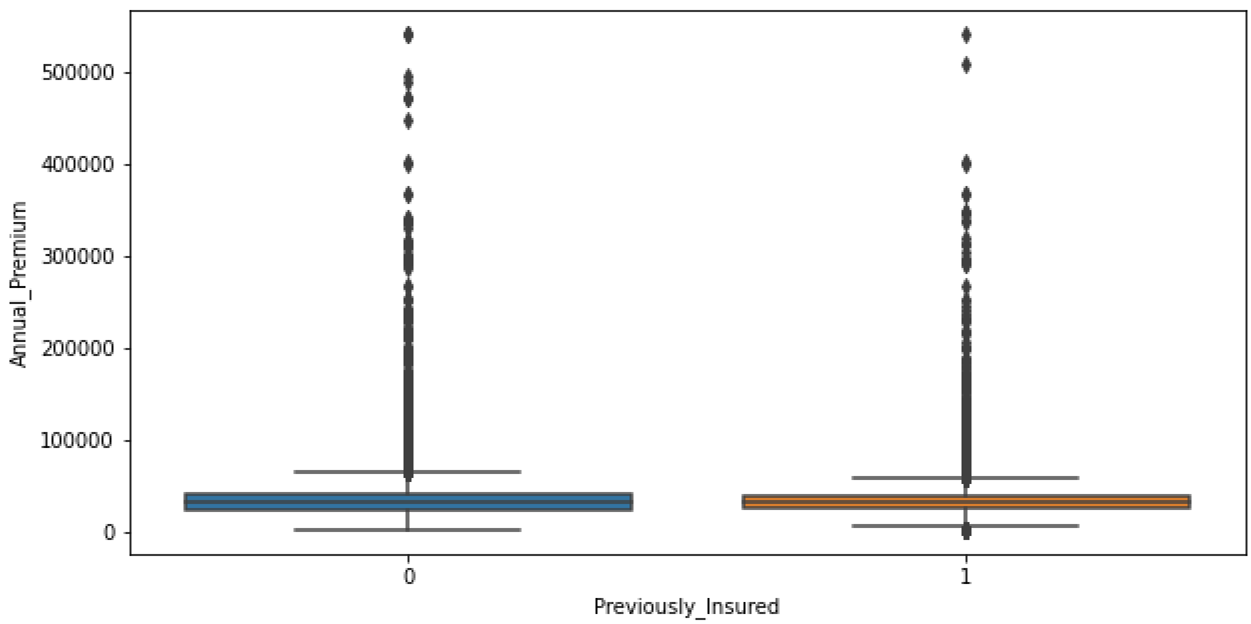


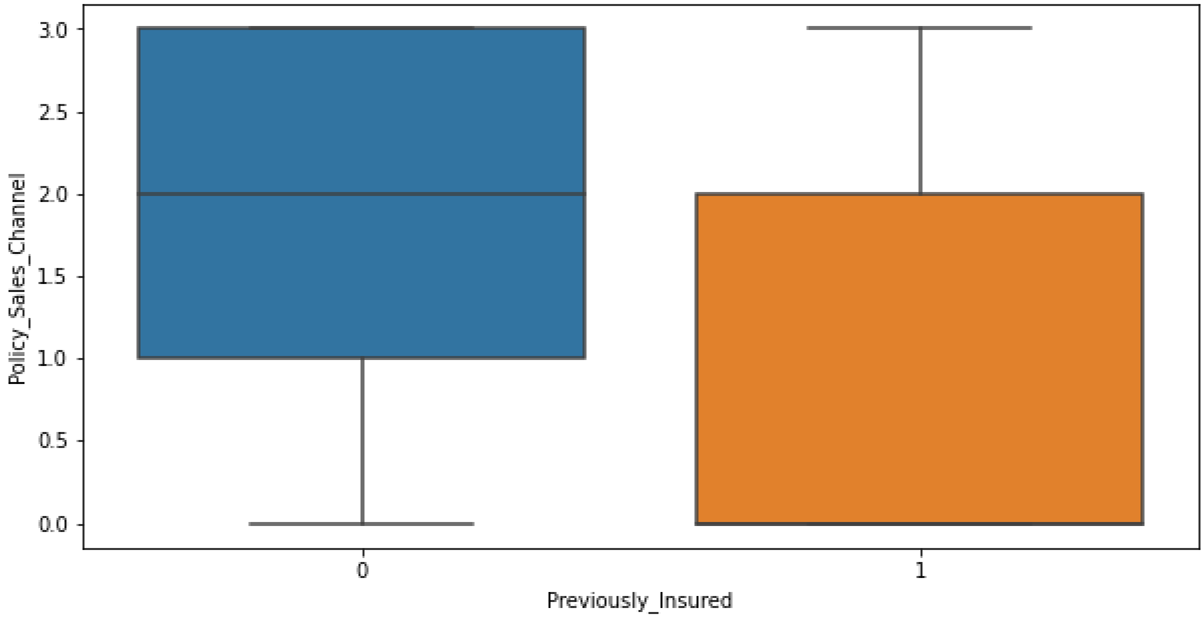
In [62]:

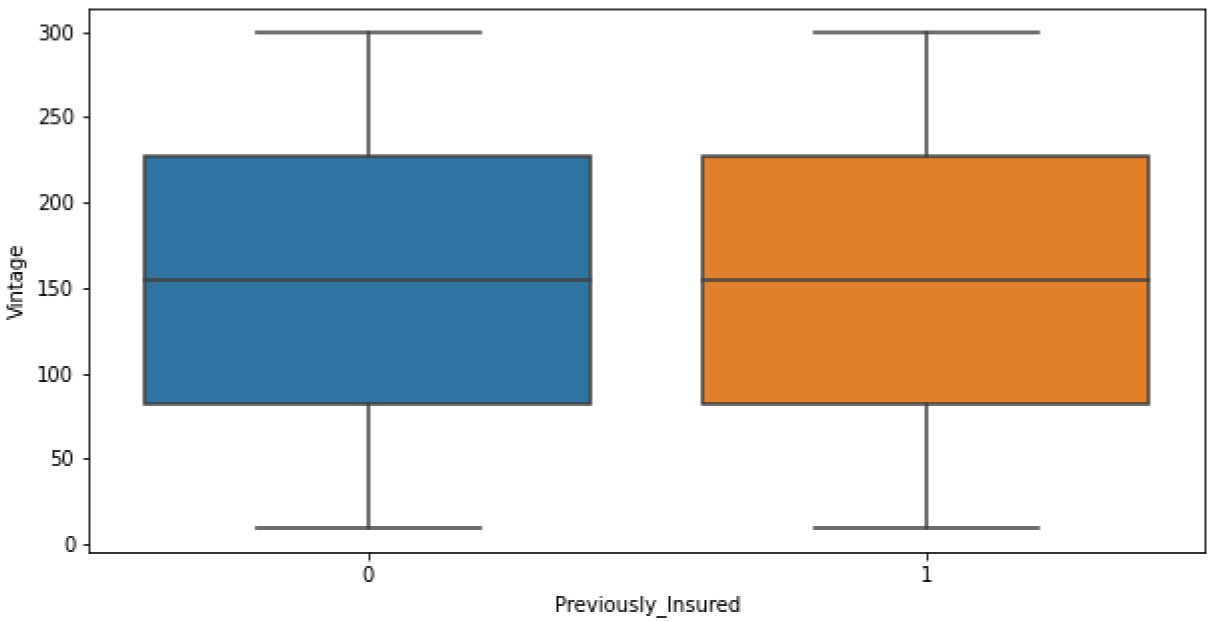
```
for i in df_num.columns:
```

```
    if i =='Response':
      continue
    else:
      sns.boxplot(x= df['Previously_Insured'],y=df[i])
      plt.show()
```









Inference (Previously Insured Vs Numerical)

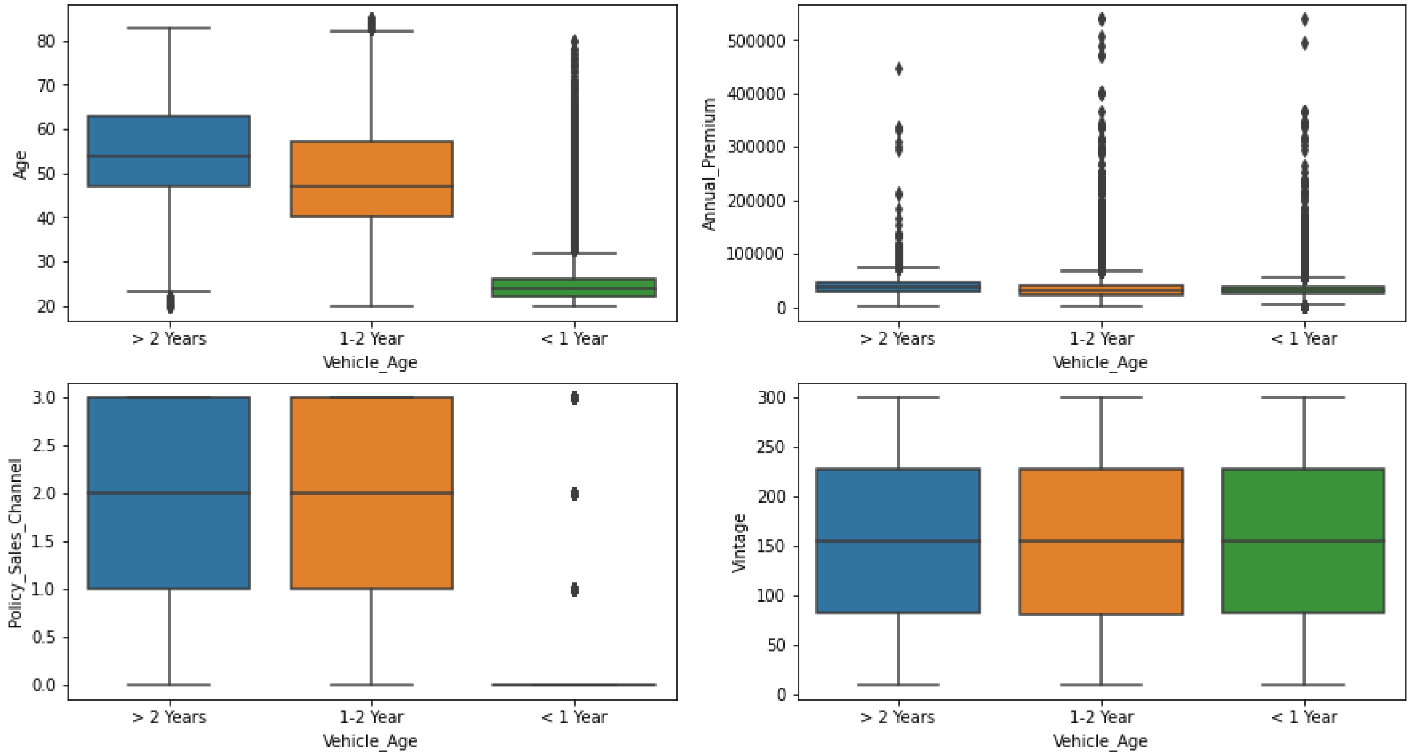
1.Customers of age group~(40-50) have not insured previously.

2.Annual Premium is extremly high for not previously insured and previously insured customer.

3.Average vintage is same for the bot the customers.

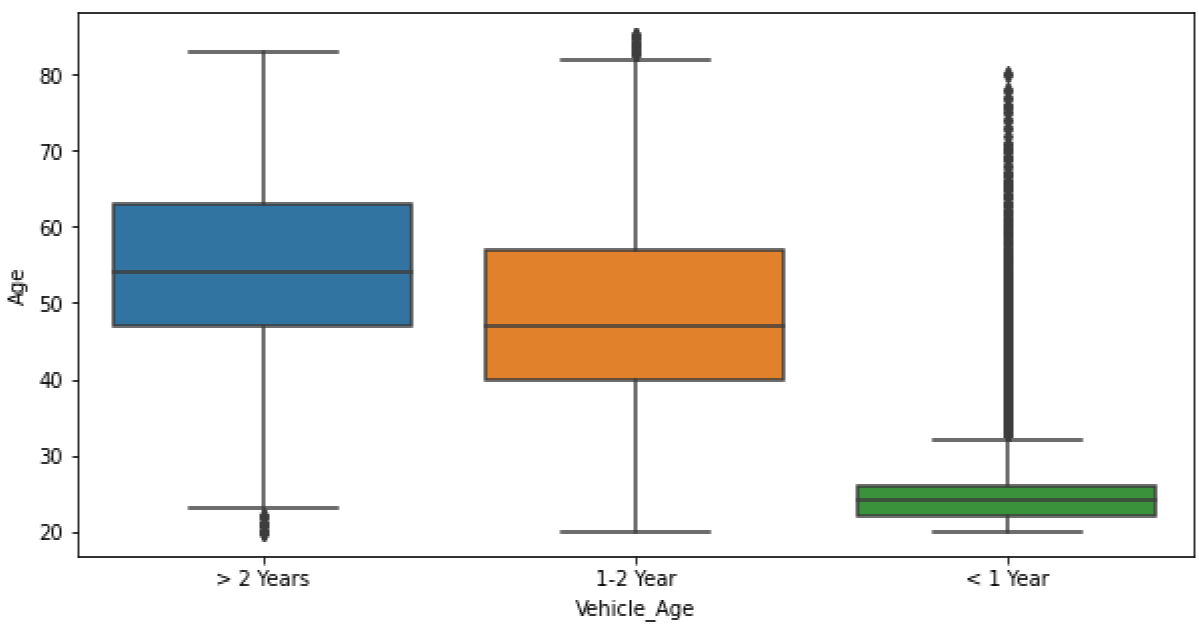
In [63]:

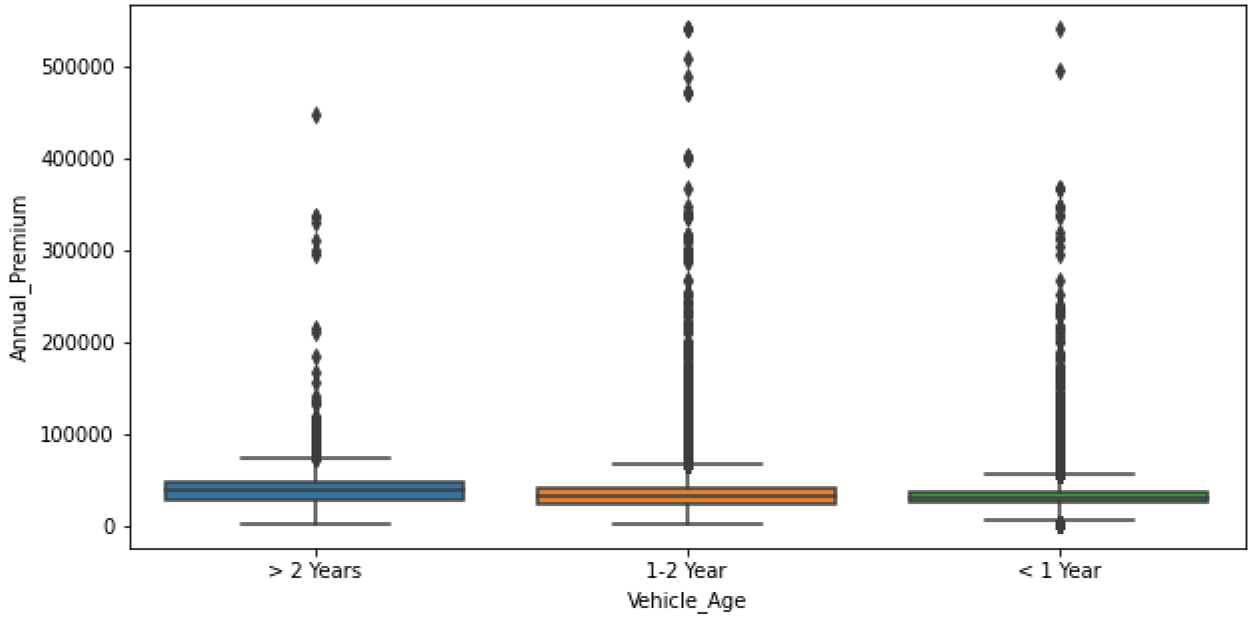
```
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize=(15, 8))
for variable, subplot in zip(df_num.columns, ax.flatten()):
  if variable == "Response":
    continue
  else:
    sns.boxplot(x = df["Vehicle_Age"],y = df_num[variable], ax = subplot)
plt.show()
```

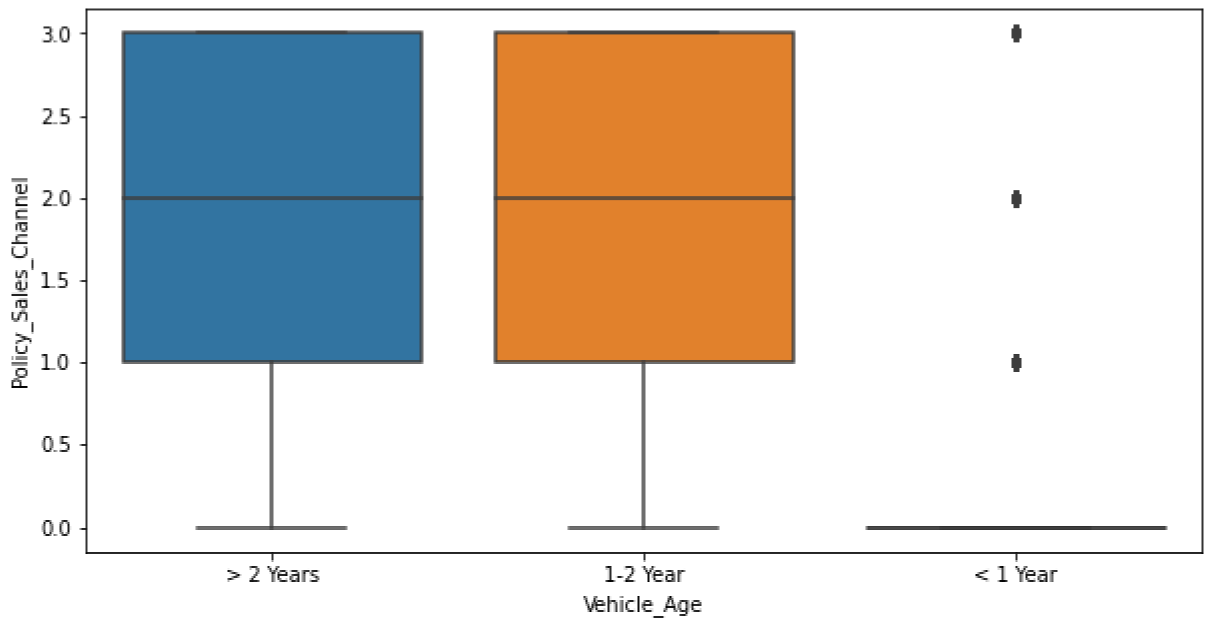


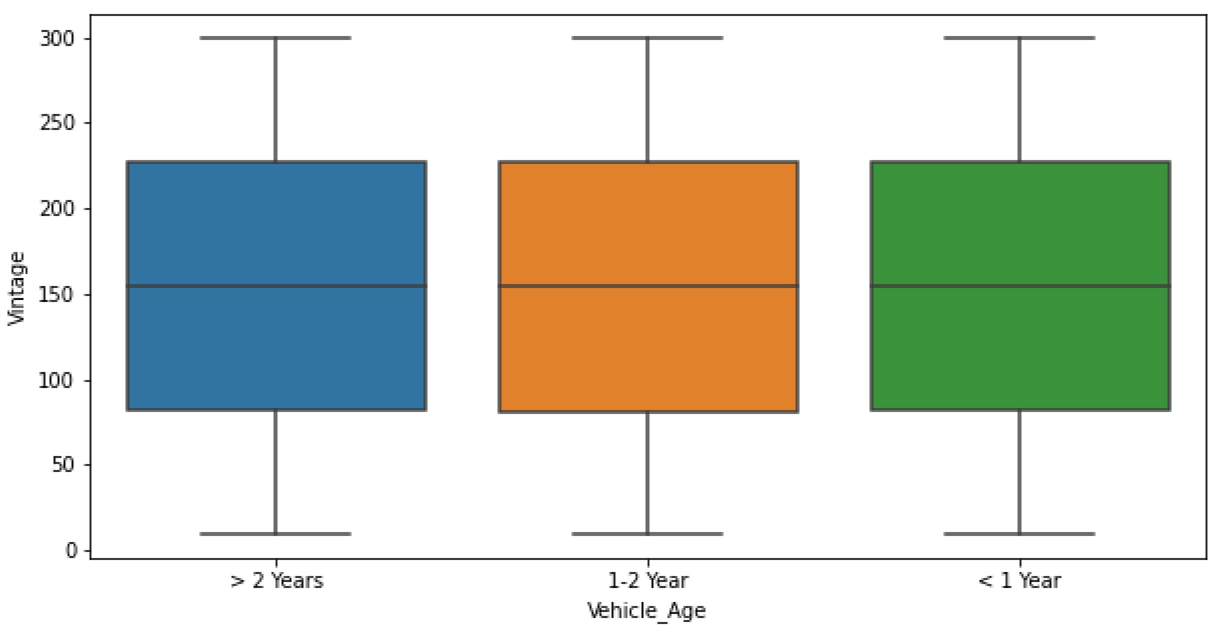
In [64]:

```
for i in df_num.columns:
  if i =='Response':
    continue
  else:
    sns.boxplot(x= df['Vehicle_Age'],y=df[i])
    plt.show()
```









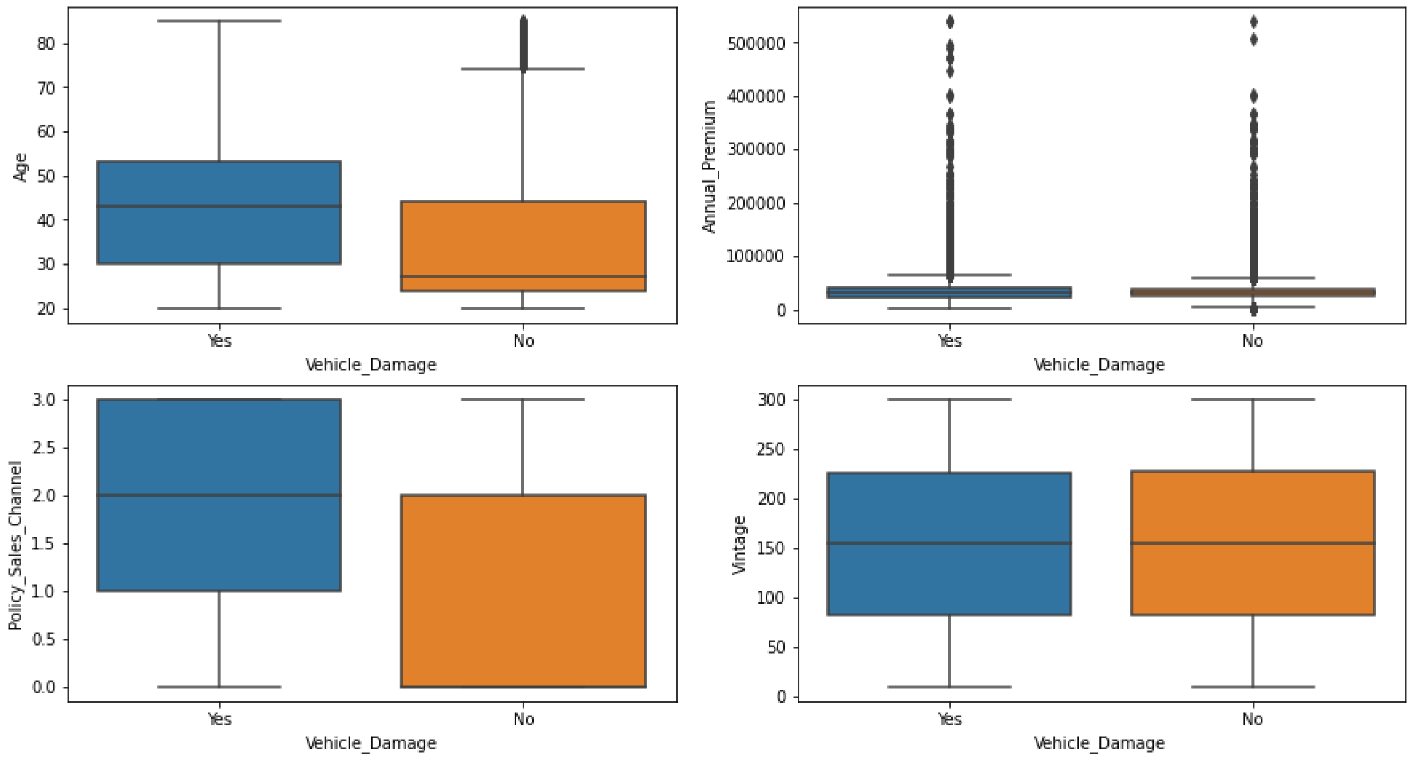
Inference (Vehicle Age Vs Numerical)

1.customer vehicle with less than 1 year have least age.

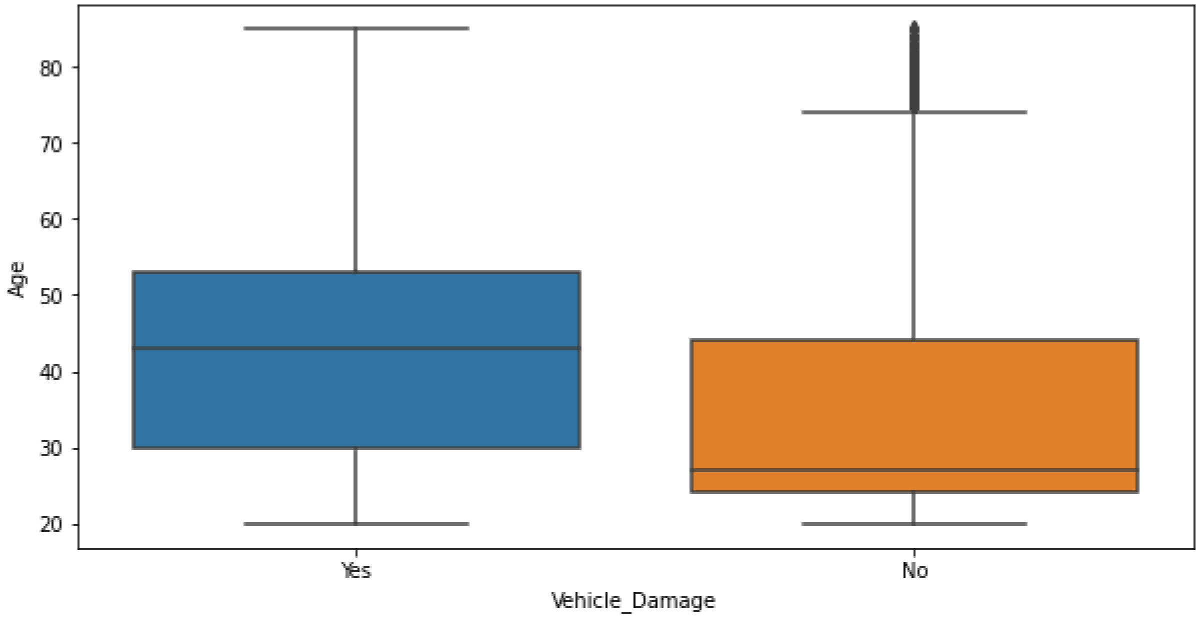
2.Average Annaul premium is same for all categories in vehicle age.

3.Average Vintage is same for all the categories in vehicle age.

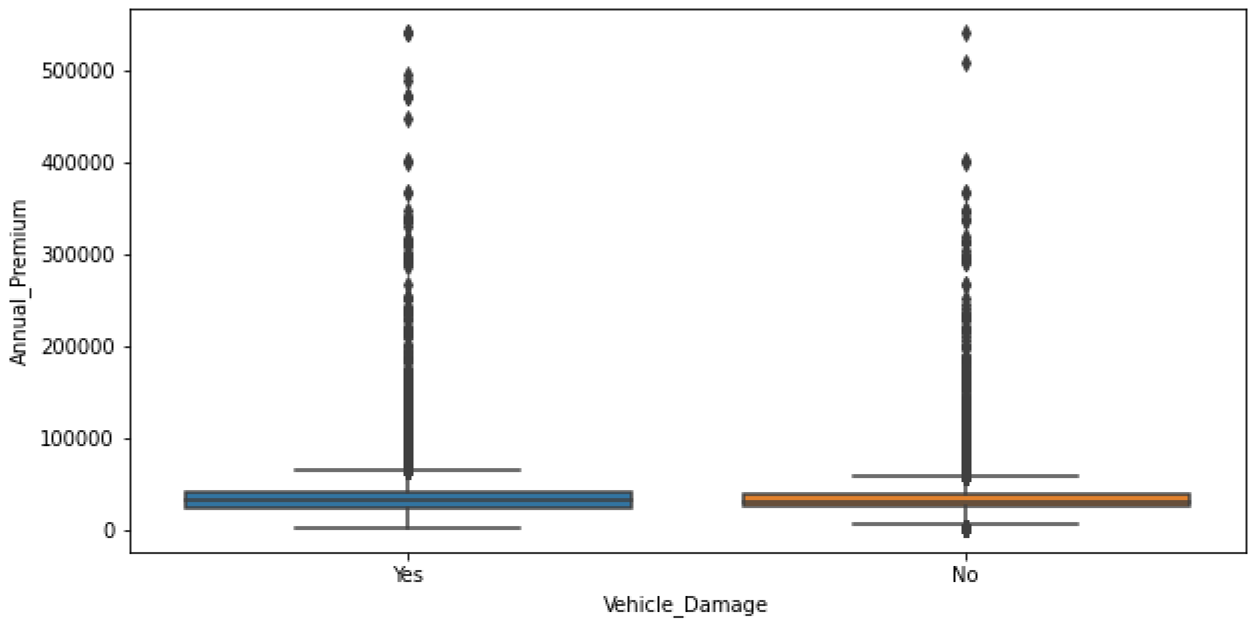
In [65]:
```
fig, ax = plt.subplots(nrows = 2, ncols = 2, figsize=(15, 8))
for variable, subplot in zip(df_num.columns, ax.flatten()):
  if variable == "Response":
    continue
  else:
    sns.boxplot(x = df["Vehicle_Damage"],y = df_num[variable], ax = subplot)
plt.show()
```



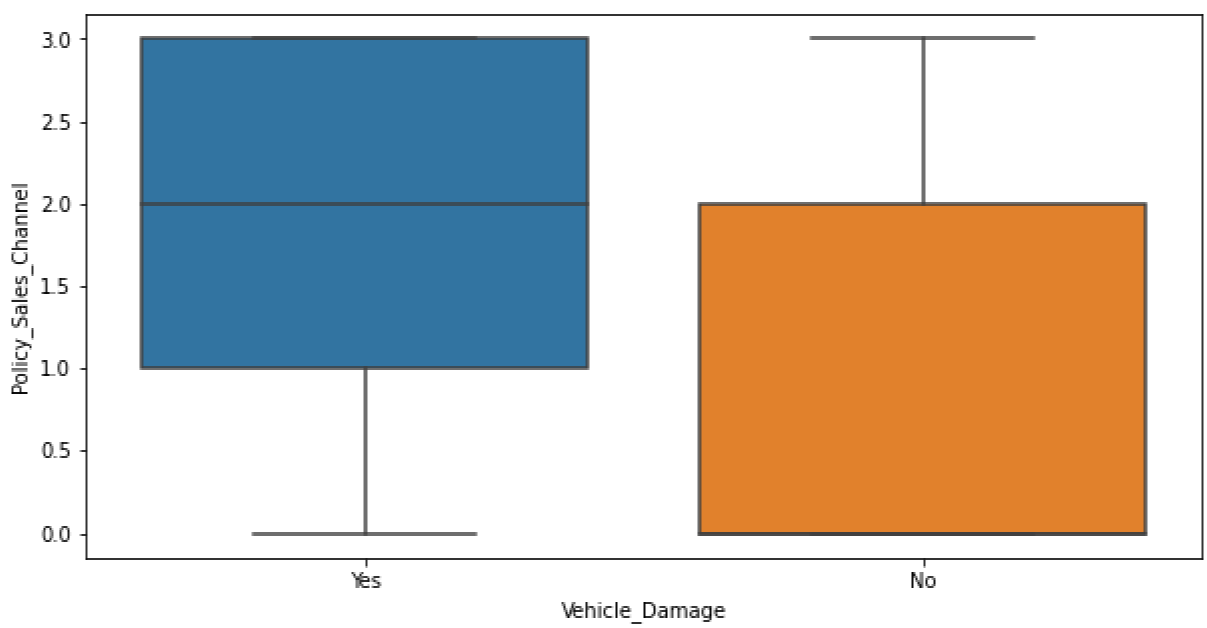
In [66]:
```
for i in df_num.columns:
  if i =='Response':
    continue
  else:
    sns.boxplot(x= df['Vehicle_Damage'],y=df[i])
    plt.show()
```



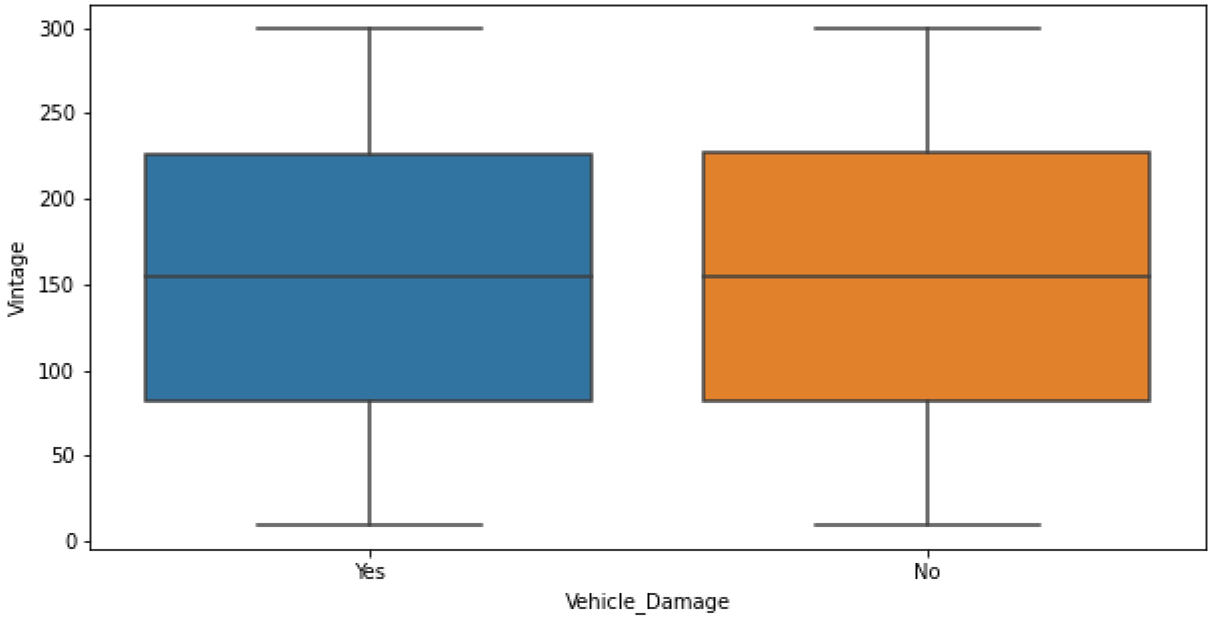
80
70
60
50
40
30
20
Age
Yes
No
Vehicle_Damage



500000
400000
300000
200000
100000
0
Annual_Premium
Yes
No
Vehicle_Damage



3.0
2.5
2.0
1.5
1.0
0.5
0.0
Policy_Sales_Channel
Yes
No
Vehicle_Damage



Inference (Vehical Damage Vs Numerical)

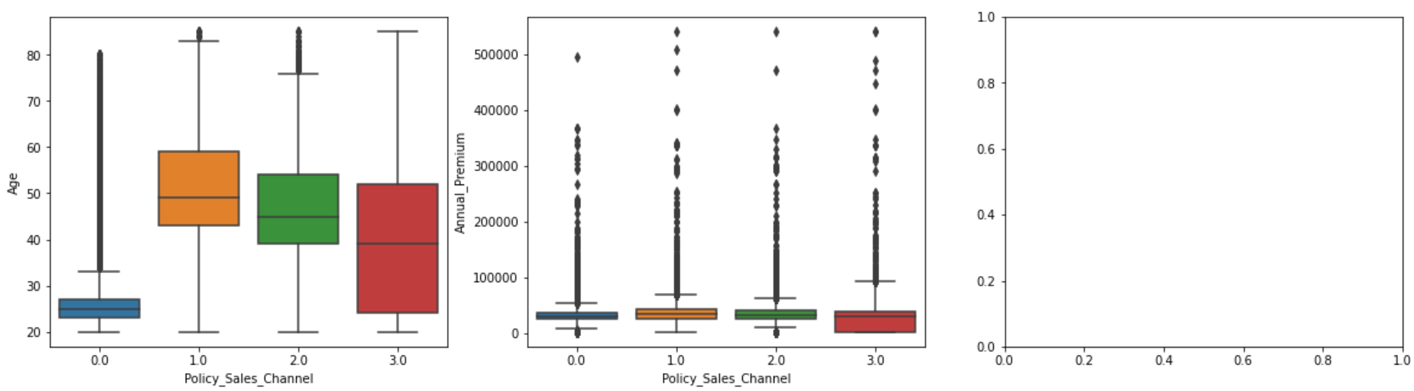
1.Customers of age group 40-50 have damaged their vehicles previously.

2.Average annual premium is same for yes and No category in vehicle damage.

3.Average vinatge is same for both.

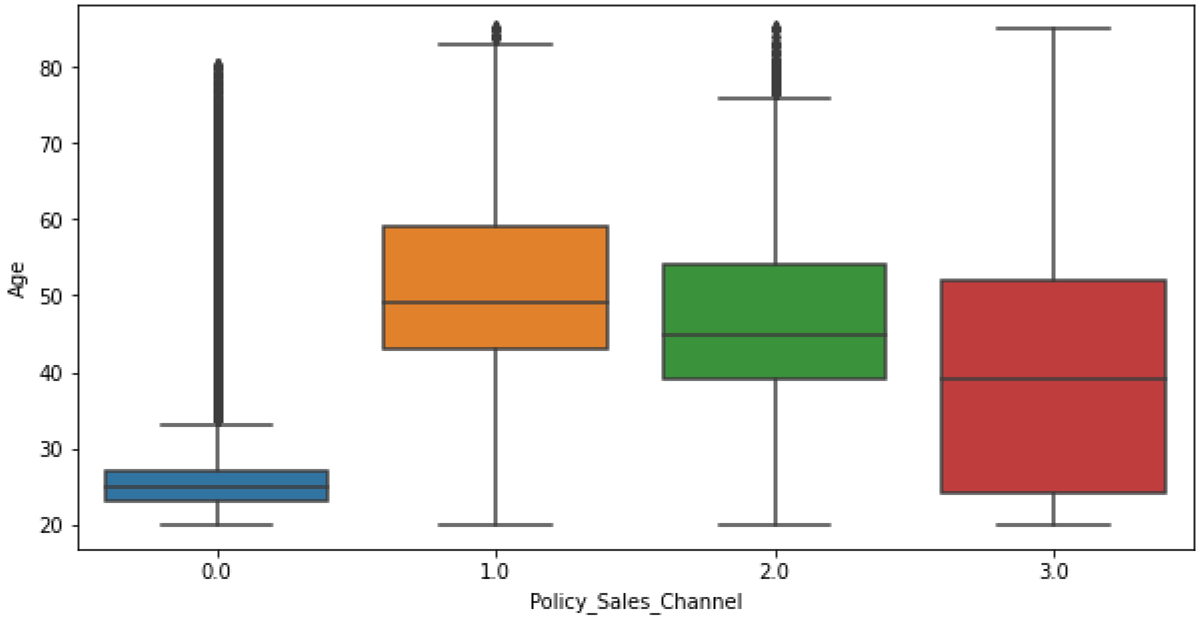
In [67]:

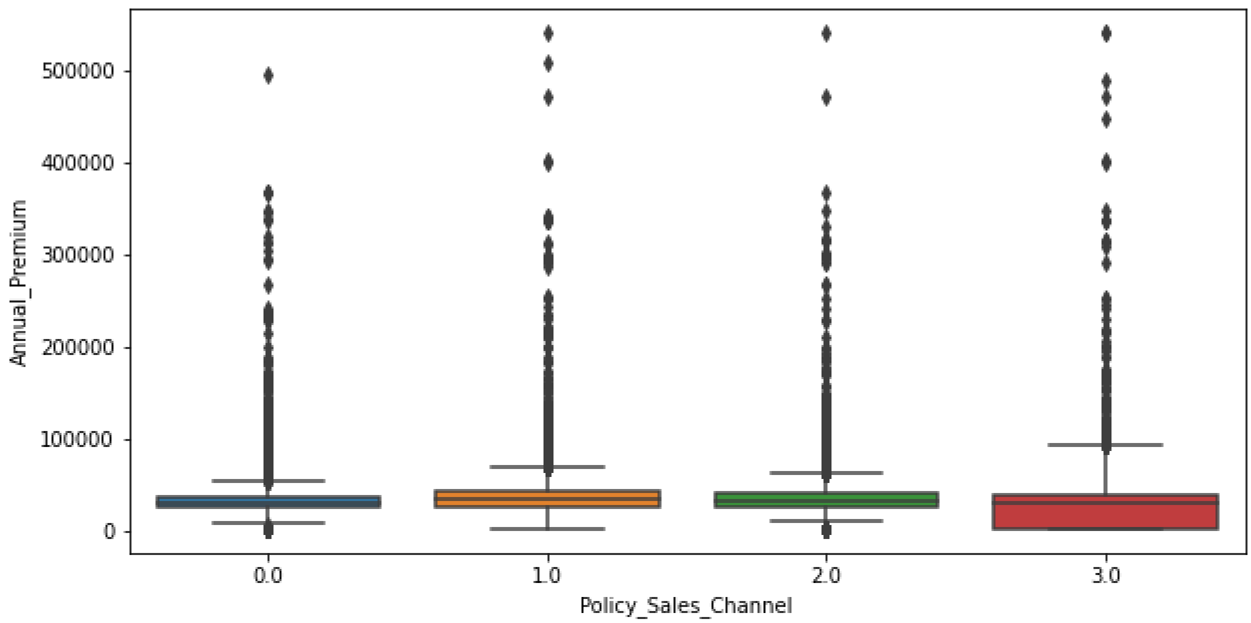
```
fig, ax = plt.subplots(nrows = 1, ncols = 3, figsize=(20, 5))
for variable, subplot in zip(df_num.columns, ax.flatten()):
  if variable == "Response" or variable == "Policy_Sales_Channel":
    continue
  else:
    sns.boxplot(x = df["Policy_Sales_Channel"],y = df_num[variable], ax = subplot)
plt.show()

#Better to use the normal one for this inference alone
```

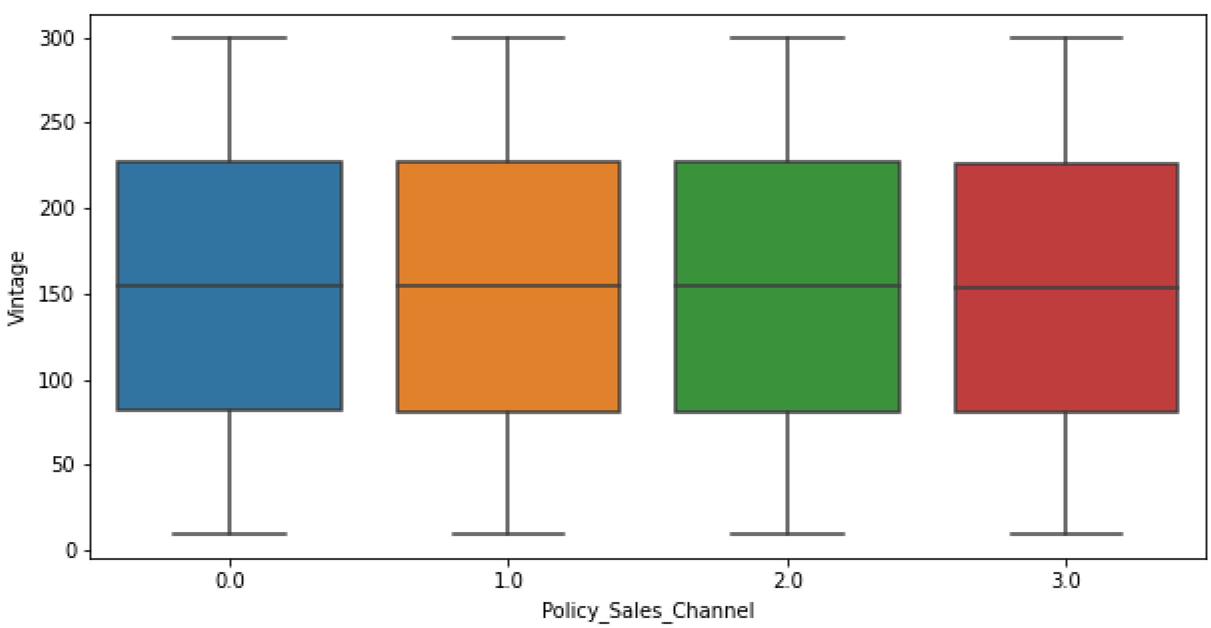


In [68]:

```
for i in df_num.columns:
  if i =='Response' or i == "Policy_Sales_Channel":
    continue
  else:
    sns.boxplot(x= df['Policy_Sales_Channel'],y=df[i])
    plt.show()
```







categorical vs categorical

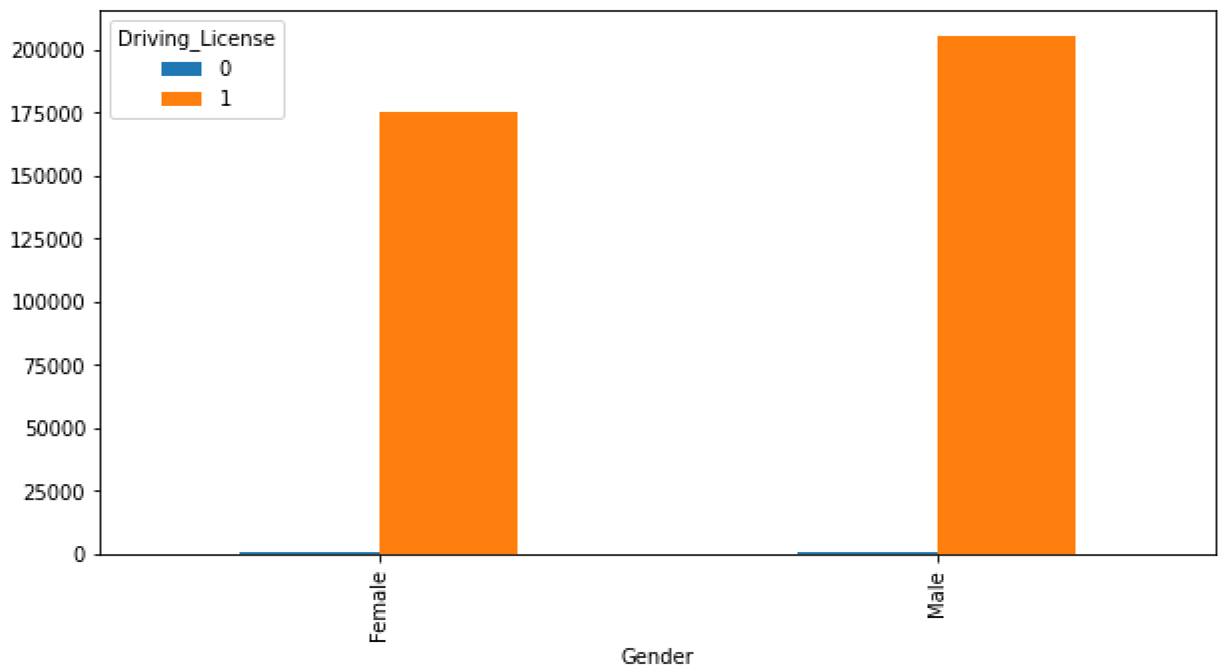
In [69]:

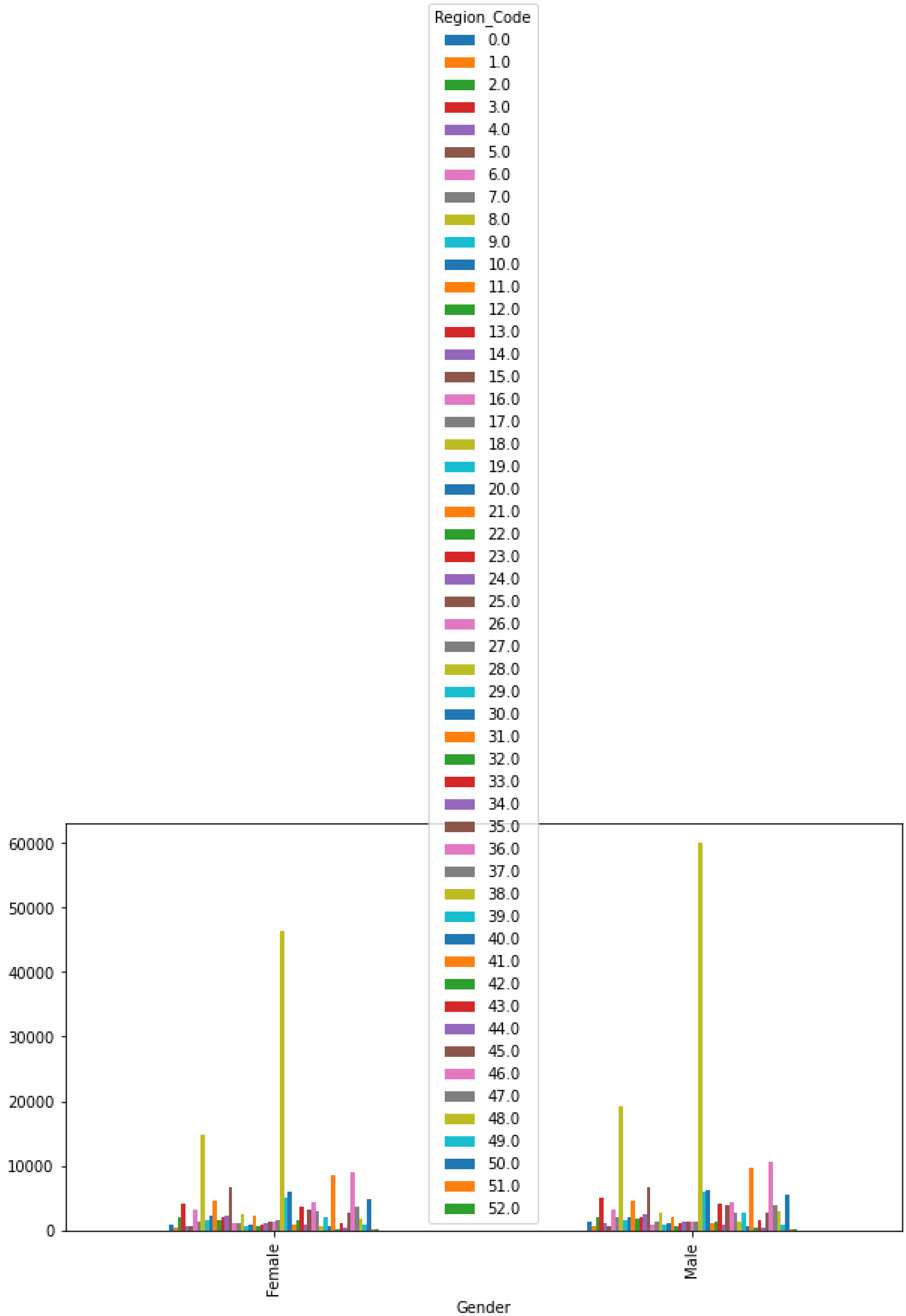
```
df_cat.columns
```

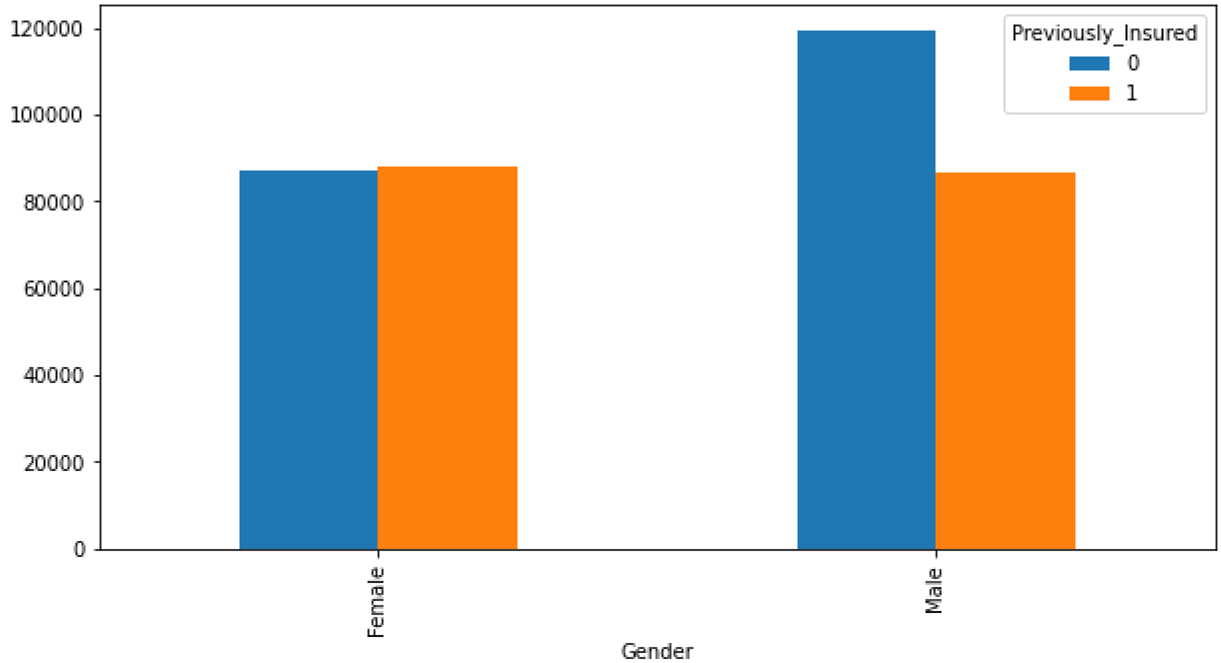
Out[69]:
```
Index(['Gender', 'Driving_License', 'Region_Code', 'Previously_Insured',
       'Vehicle_Age', 'Vehicle_Damage'],
      dtype='object')
```

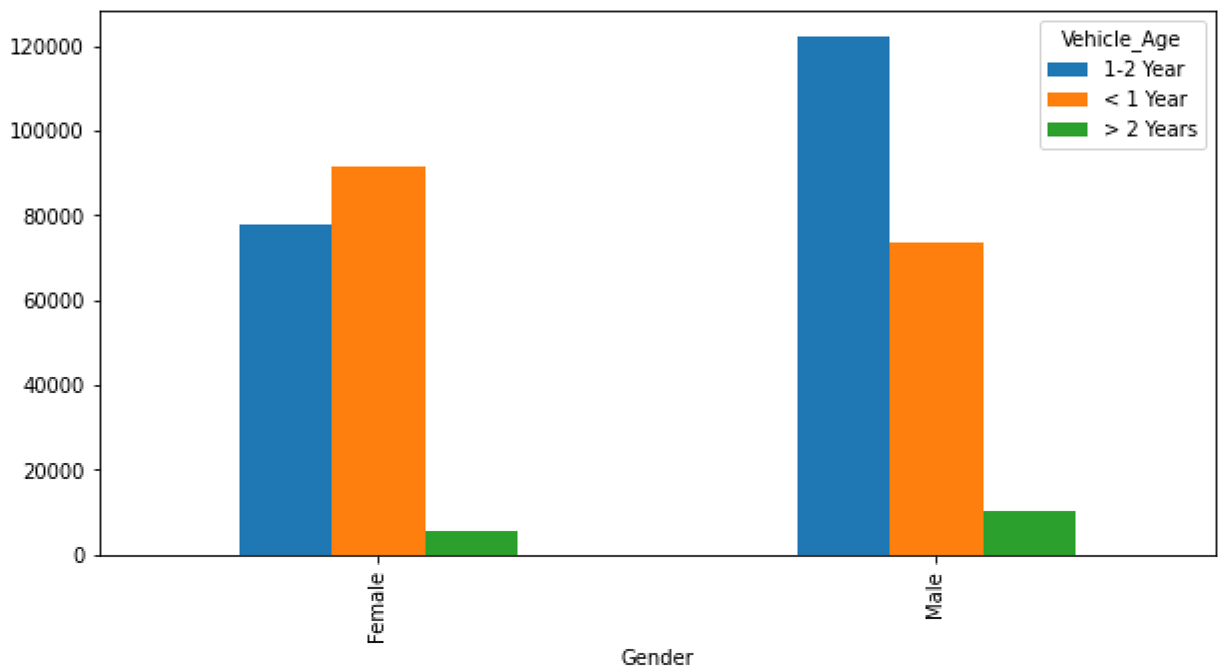
In [69]:

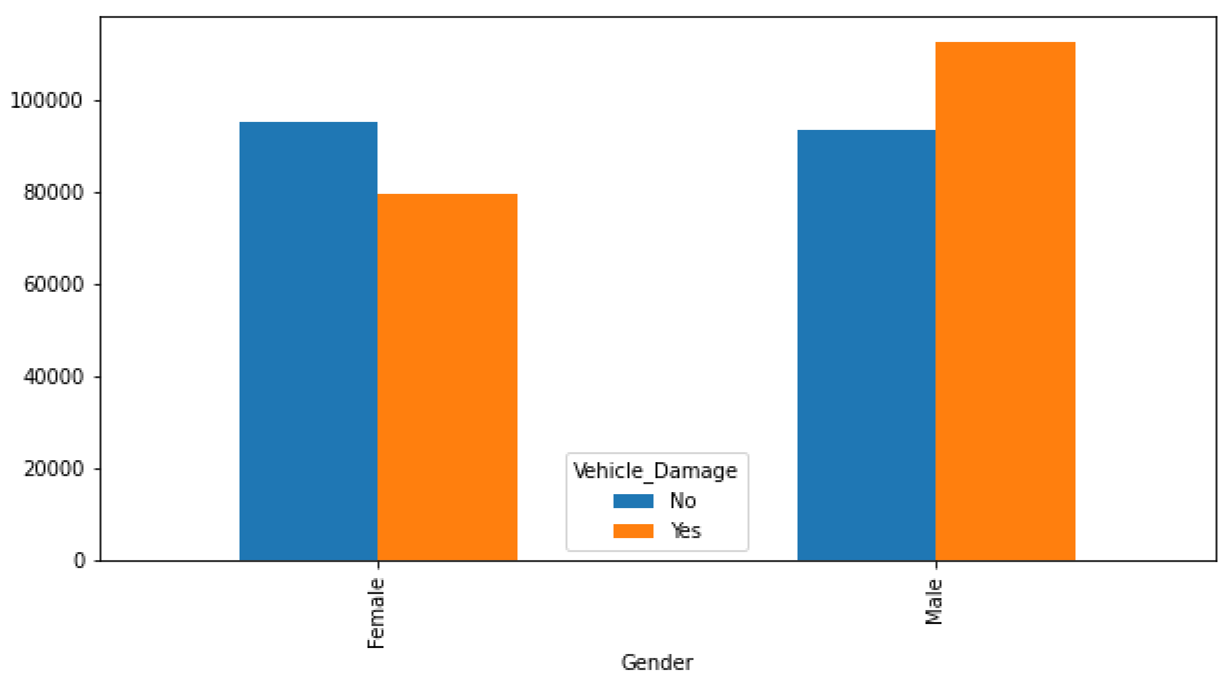
In [70]:
```
for i in df_cat.columns:
  if i == 'Gender':
    continue
  else:
    pd.crosstab(df['Gender'],df[i]).plot(kind='bar')
    plt.show()
# Inference (Gender vs categorical)
#Male customers count is more comapared to female.
#Region code 38 has more customers compared to others
# Most of the male customers have not previously insured compared to female customers
# There are more number of female customers whose vehicle age is less than 1 and there ar
# whose vehicle age is between 1-2 years
# Vehicle damage is more for male customers compared to female customers
# Majority of the customers has been reached through agents irrespective of the gender(i.
```




Region_Code
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
31.0
32.0
33.0
34.0
35.0
36.0
37.0
38.0
39.0
40.0
41.0
42.0
43.0
44.0
45.0
46.0
47.0
48.0
49.0
50.0
51.0
52.0
60000
50000
40000
30000
20000
10000
0
Female
Male
Gender


Previously_Insured
0
1
0
20000
40000
60000
80000
100000
120000
Female
Male
Gender


Vehicle_Age
1-2 Year
< 1 Year
> 2 Years
0
20000
40000
60000
80000
100000
120000
Female
Male
Gender


Vehicle_Damage
No
Yes
0
20000
40000
60000
80000
100000
Female
Male
Gender

Inference (Gender vs categorical)

1.Male customers count is more comapared to female.

2.Region code 38 has more customers compared to others

3.Most of the male customers have not previously insured compared to female customers

4.There are more number of female customers whose vehicle age is less than 1 and there are more number of male customers whose vehicle age is between 1-2 years

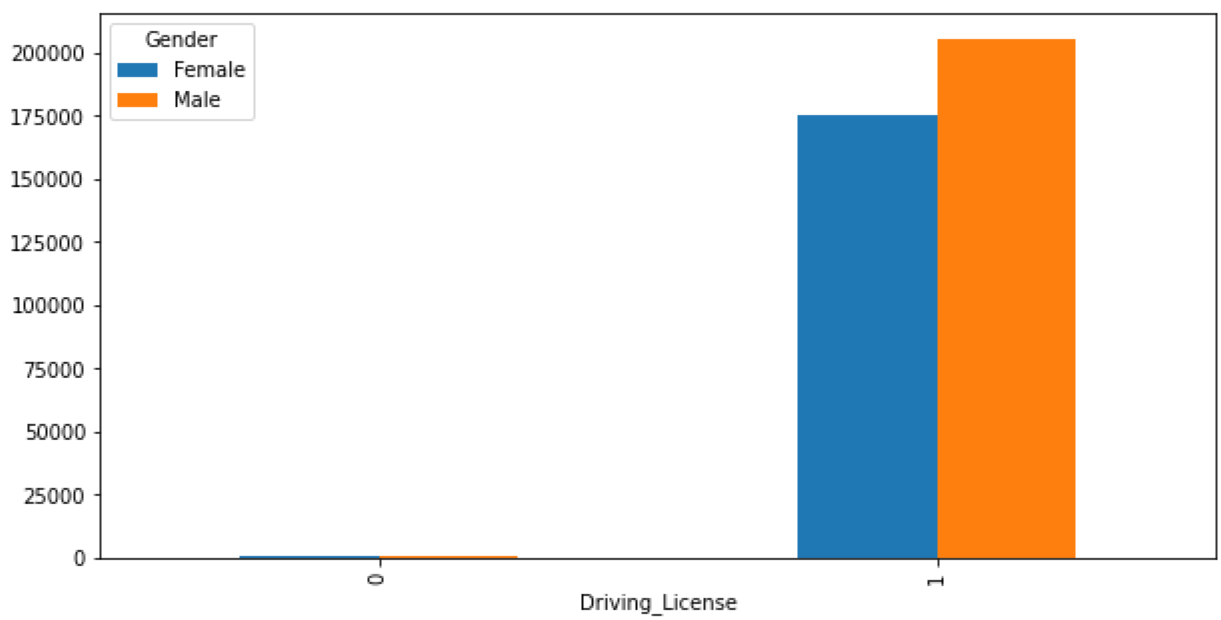
5.Vehicle damage is more for male customers compared to female customers

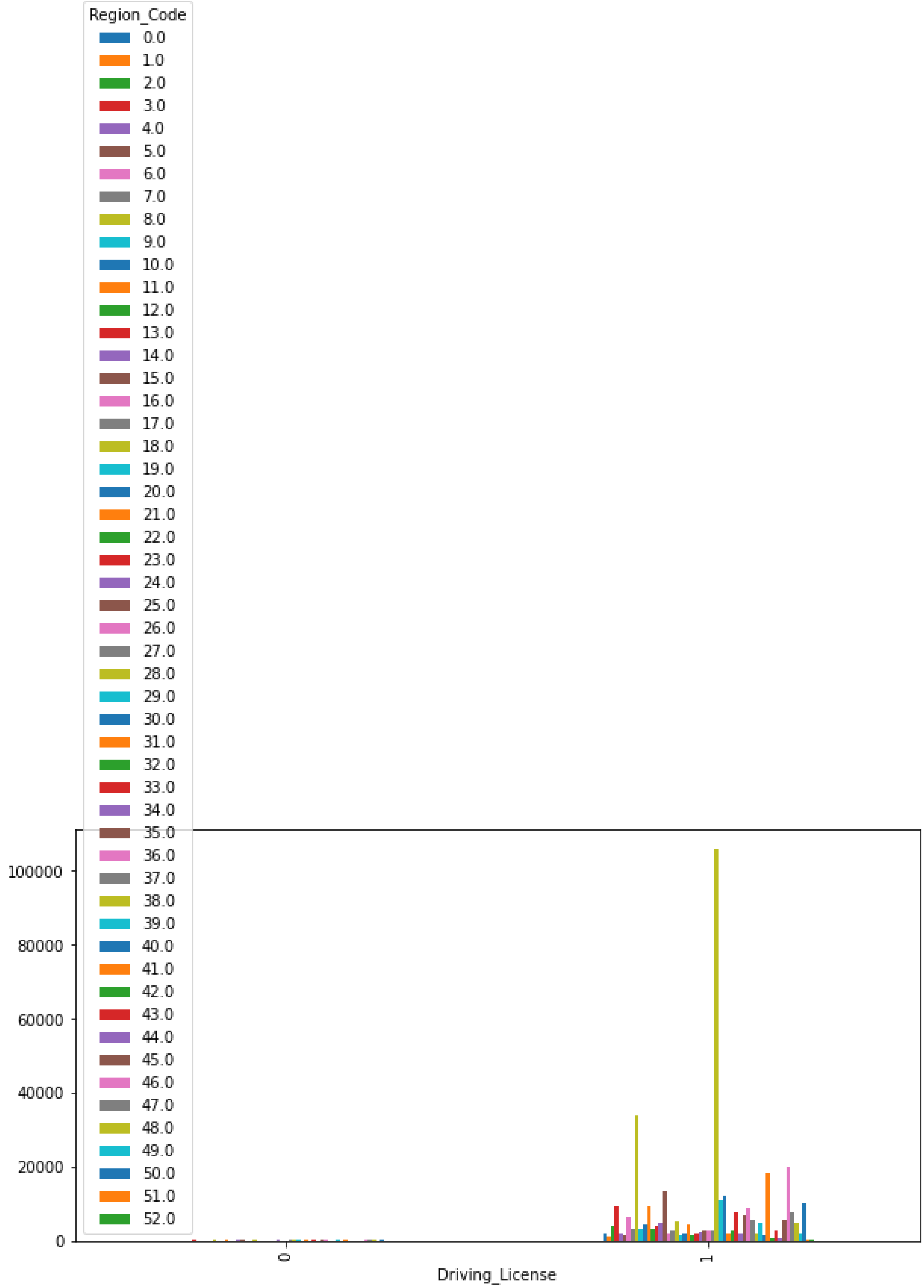
6.Majority of the customers has been reached through agents irrespective of the gender

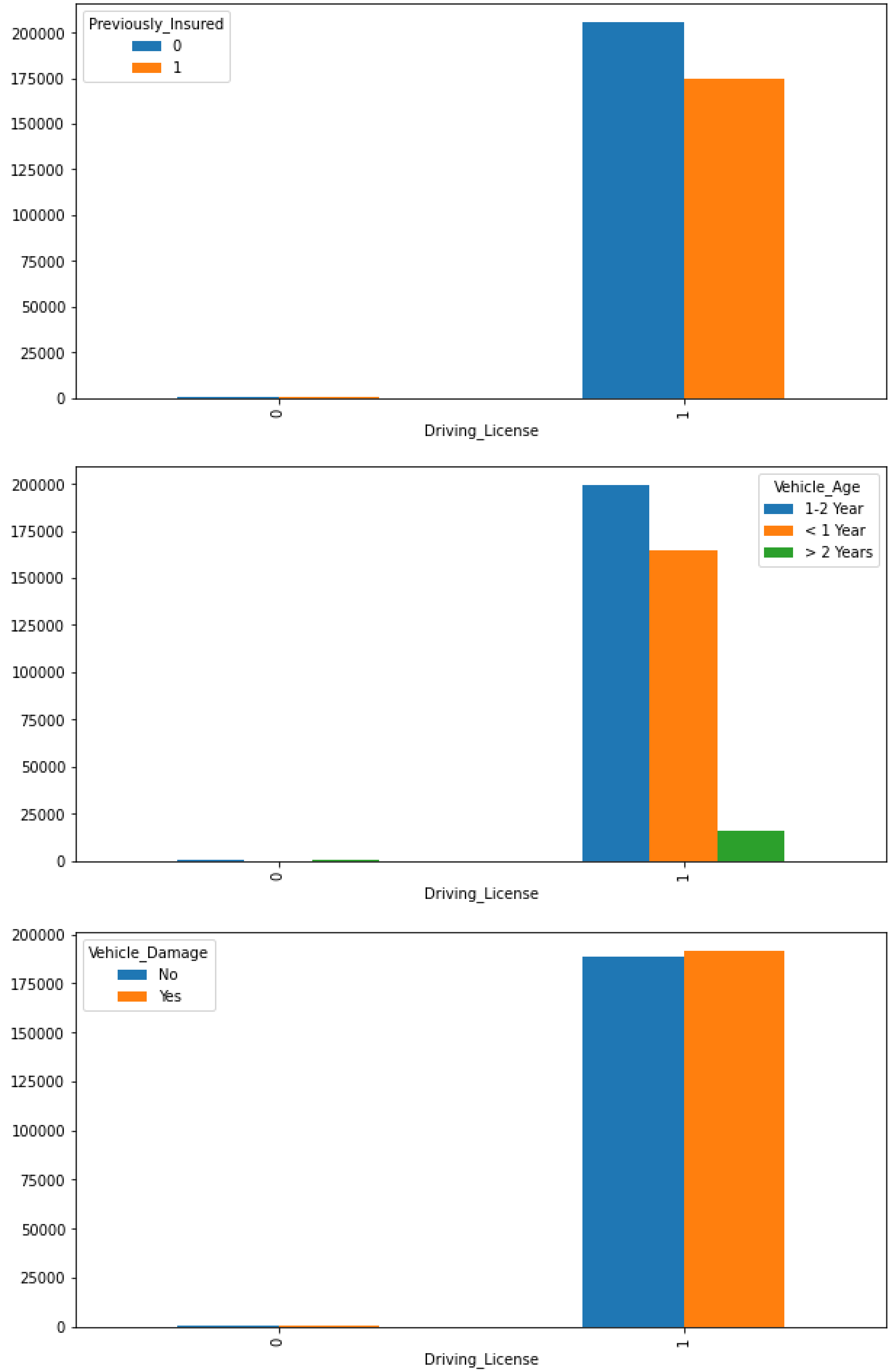
---

In [71]:

```
for i in df_cat.columns:
  if i == 'Driving_License':
    continue
  else:
    pd.crosstab(df['Driving_License'],df[i]).plot(kind='bar')
    plt.show()
```




Region_Code
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
31.0
32.0
33.0
34.0
35.0
36.0
37.0
38.0
39.0
40.0
41.0
42.0
43.0
44.0
45.0
46.0
47.0
48.0
49.0
50.0
51.0
52.0
100000
80000
60000
40000
20000
0
0
1
Driving_License



Inferences (Driving License Vs Categorical)

1.More number of Male Customers has driving license compared to female customers

2.More number of Customers from region code 38 has driving license compared to customers from other regions.

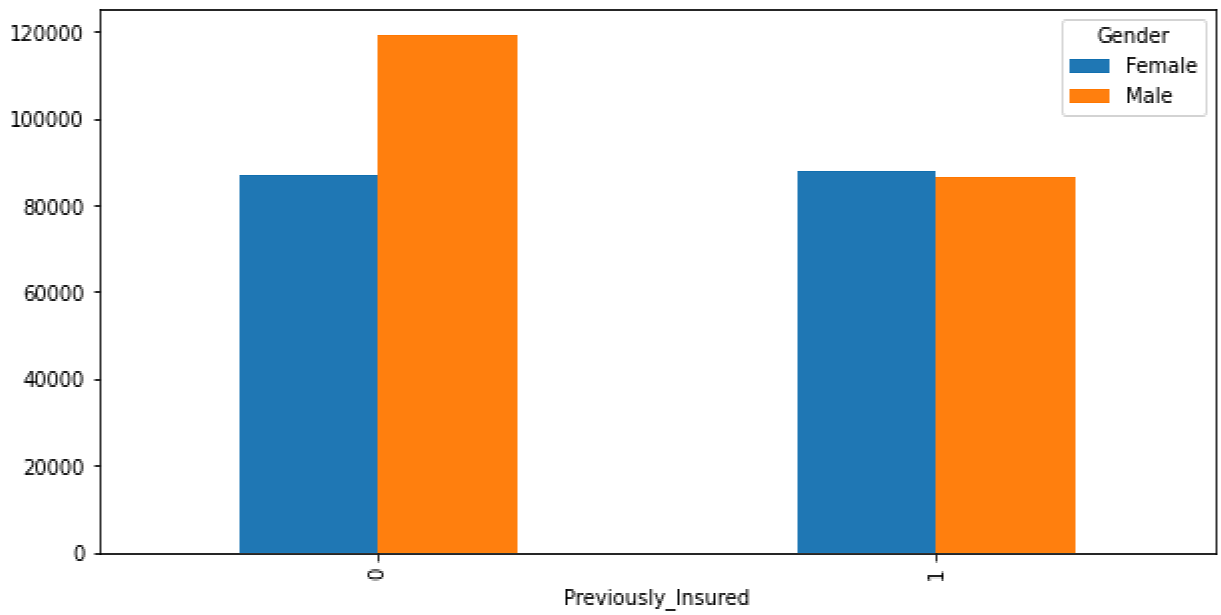
3.Most of the customers having driving license are not previously insured.

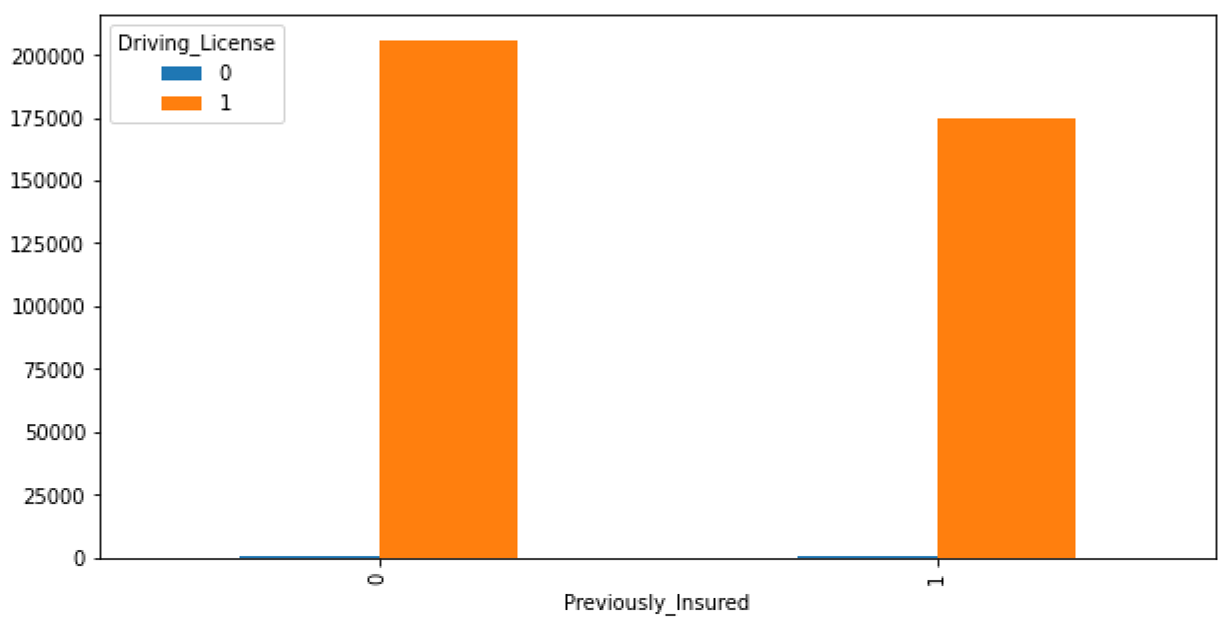
4.Most of the customers having driving license has a vehicle age between 1-2.

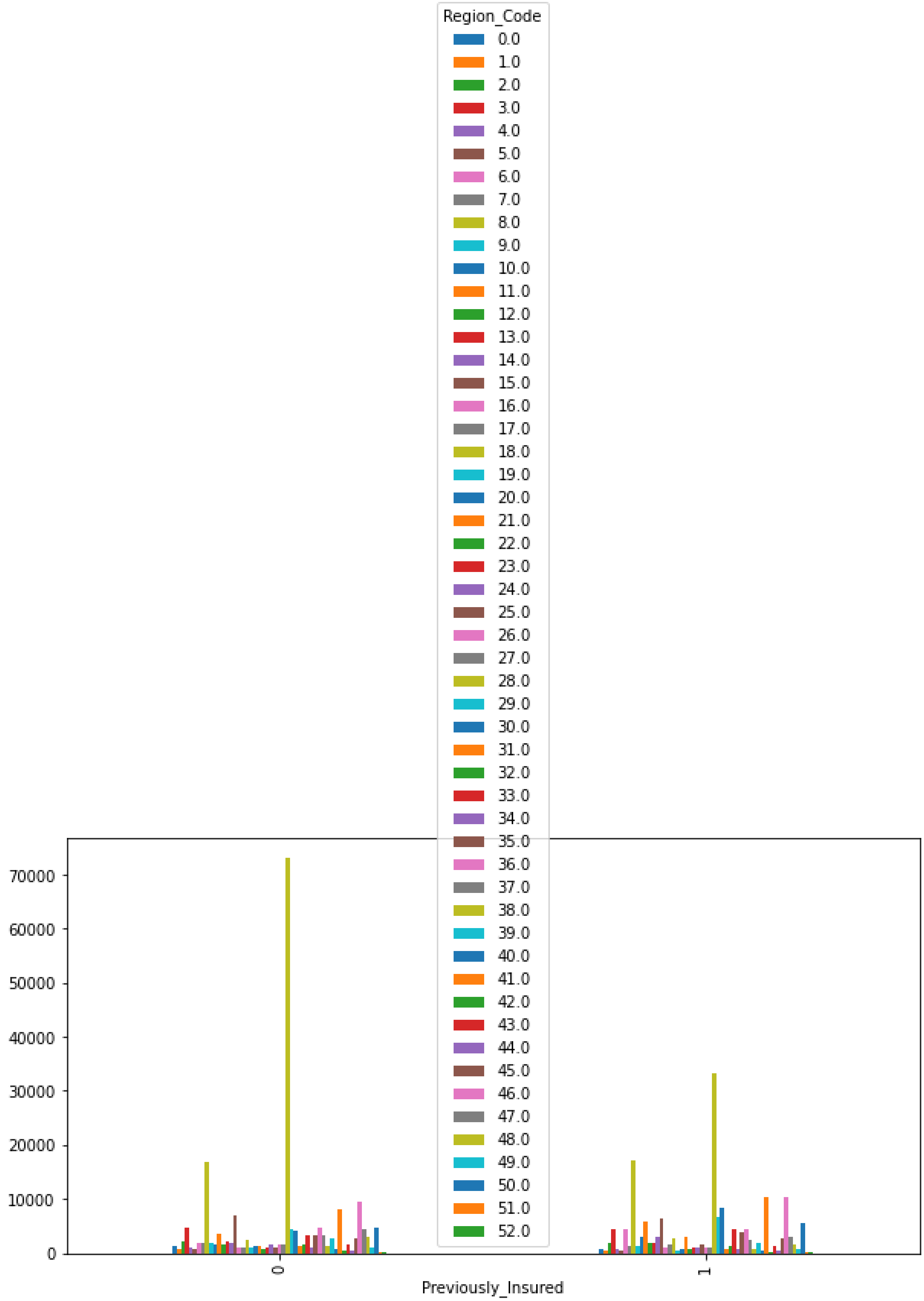
5.The Vehice damage ratio is almost similar for customers having driving license

In [72]:

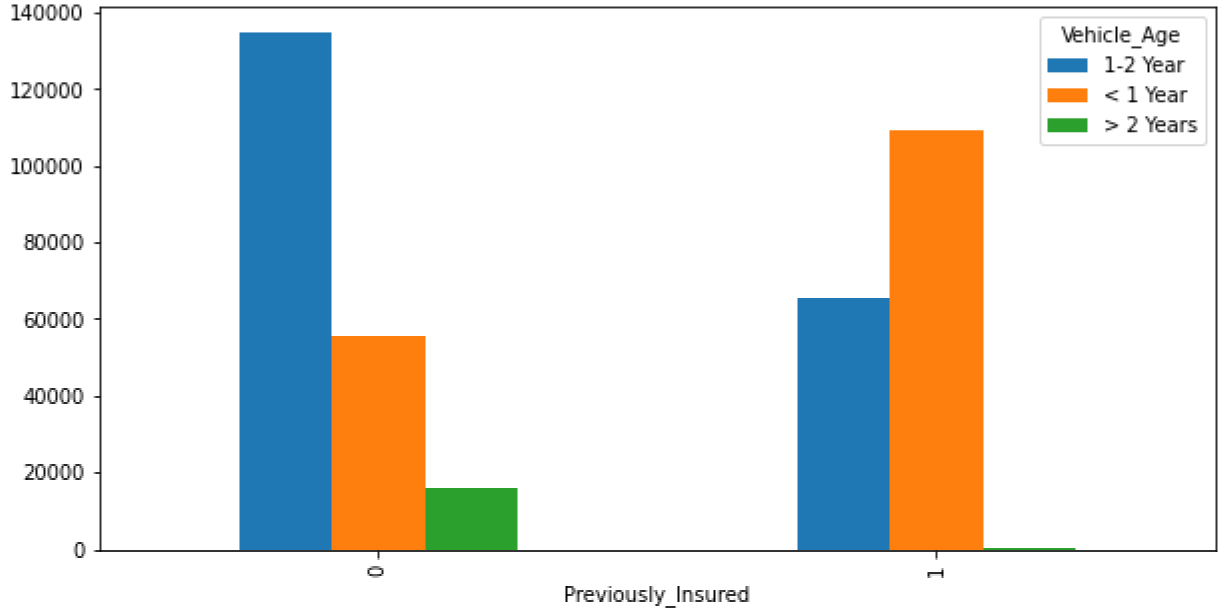
```
for i in df_cat.columns:
  if i == 'Previously_Insured':
    continue
  else:
    pd.crosstab(df['Previously_Insured'],df[i]).plot(kind='bar')
    plt.show()
```

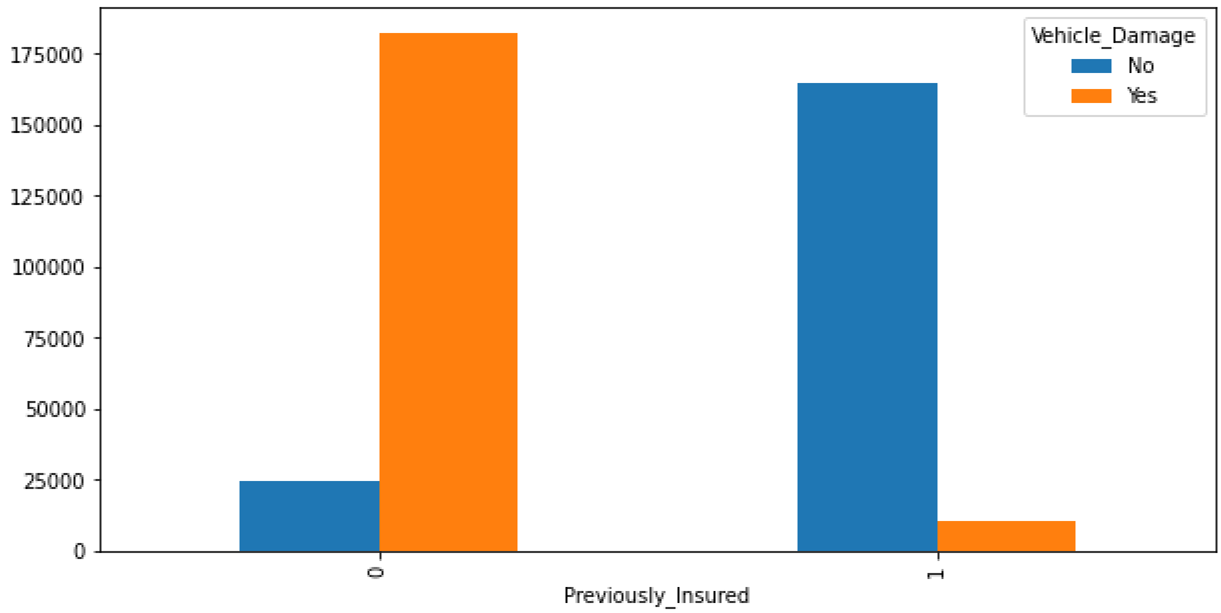






Region_Code
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
31.0
32.0
33.0
34.0
35.0
36.0
37.0
38.0
39.0
40.0
41.0
42.0
43.0
44.0
45.0
46.0
47.0
48.0
49.0
50.0
51.0
52.0
70000
60000
50000
40000
30000
20000
10000
0
0
1
Previously_Insured





Inferences (Previously Insured Vs Categorical)

1.Most of the male customers are not previously insured than female customers

2.Most of the customers have driving liscence and they are not previously insured

3.Most of the customers who are not previously insured are from region 38

4.Most of the customers whose vehicle age is between 1-2 are not previously insured

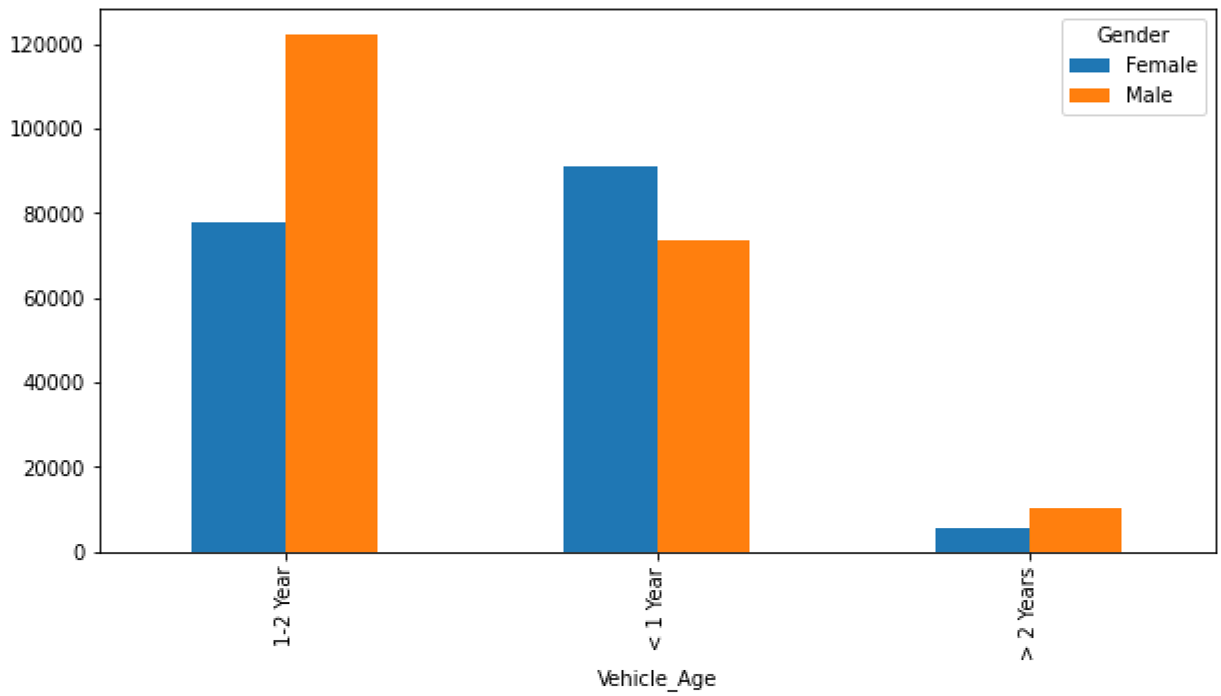
5.Most of the customers who are not previously insured have more vehicle damage and most of the customers who have already insured has no vehicle damage.

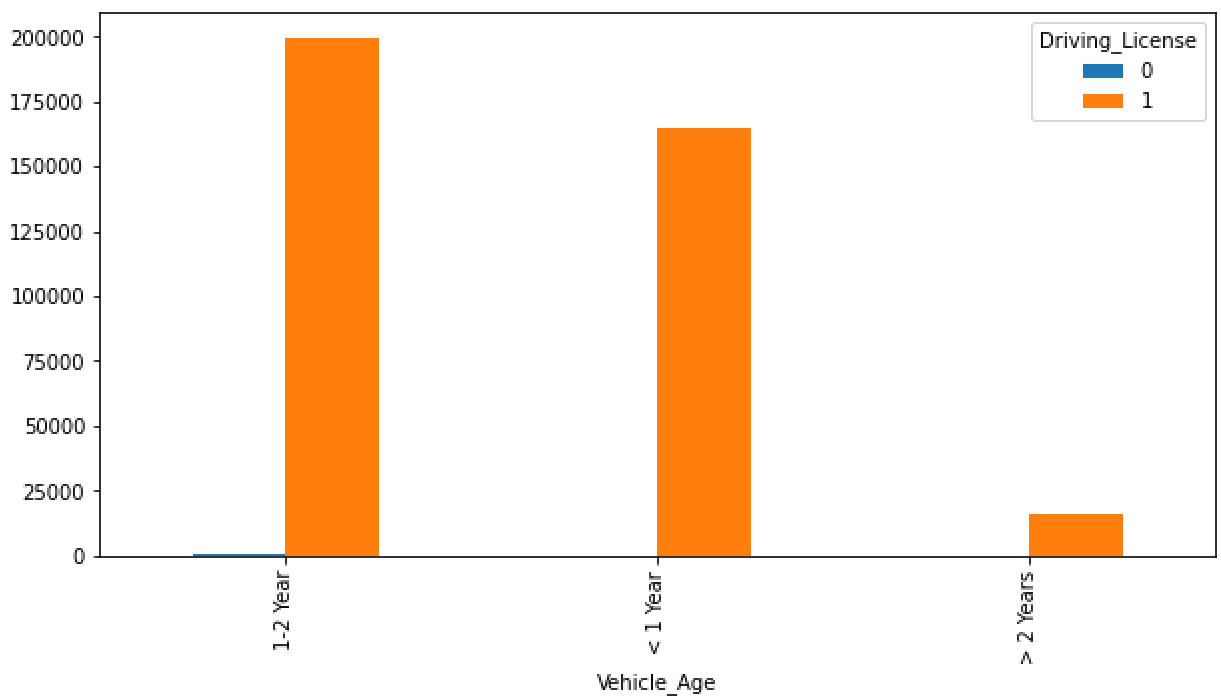
In [73]:

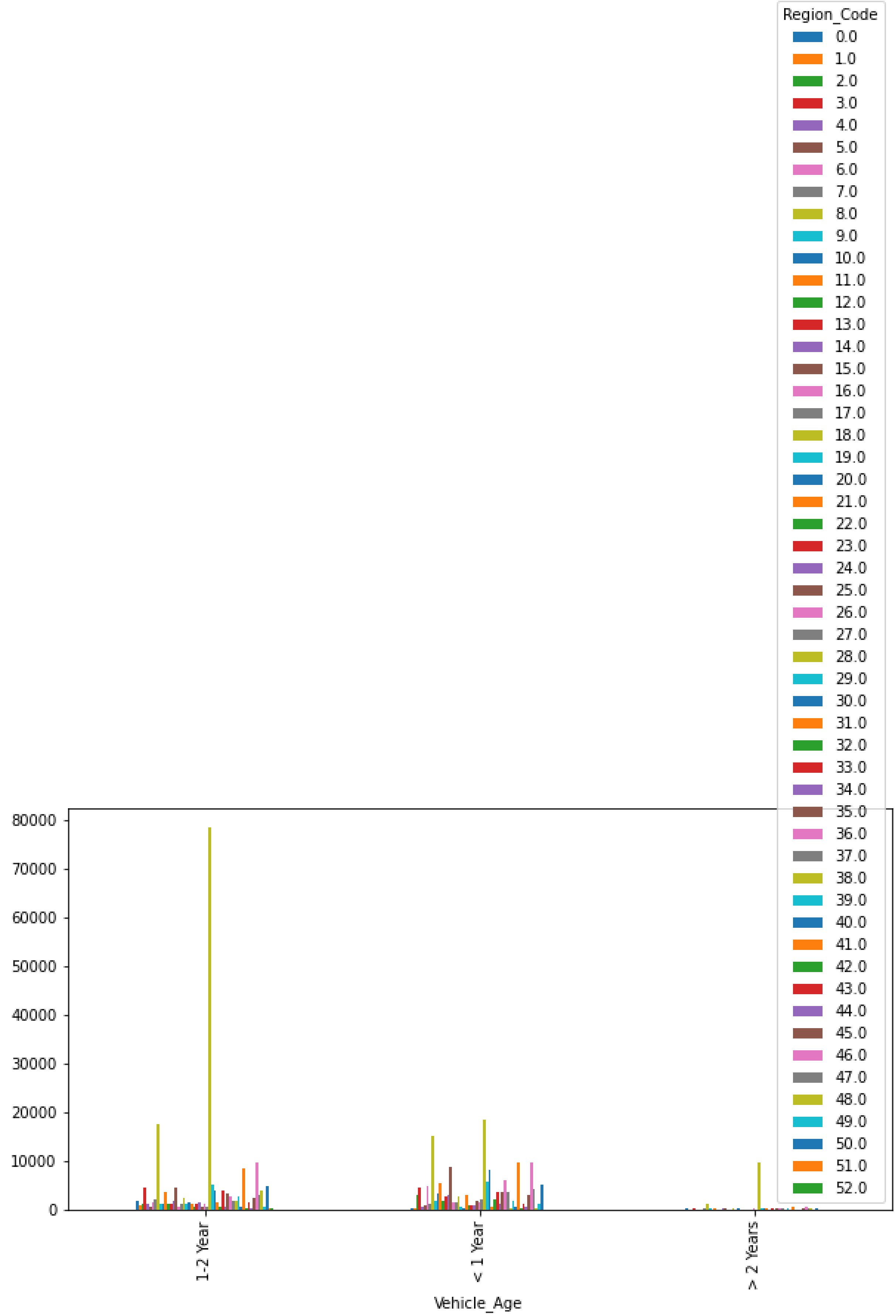
```
# cross1 = pd.crosstab(df["Vehicle_Age"],df["Gender"])
# cross2 = pd.crosstab(df["Vehicle_Age"],df["Driving_License"])
# cross3 = pd.crosstab(df["Vehicle_Age"],df["Previously_Insured"])
# cross4 = pd.crosstab(df["Vehicle_Age"],df["Vehicle_Damage"])

# fig, [[ax1, ax2],[ax3,ax4]] = plt.subplots(2,2, figsize = (20,6))
# ax1 = cross1.plot(kind='bar', stacked=True, rot=0)
```

```
# ax2 = cross2.plot(kind='bar', stacked=True, rot=0)
# ax3 = cross3.plot(kind = "bar",stacked = True,rot = 0)
# ax4 = cross4.plot(kind = "bar",stacked = True,rot = 0)
# plt.show()
```

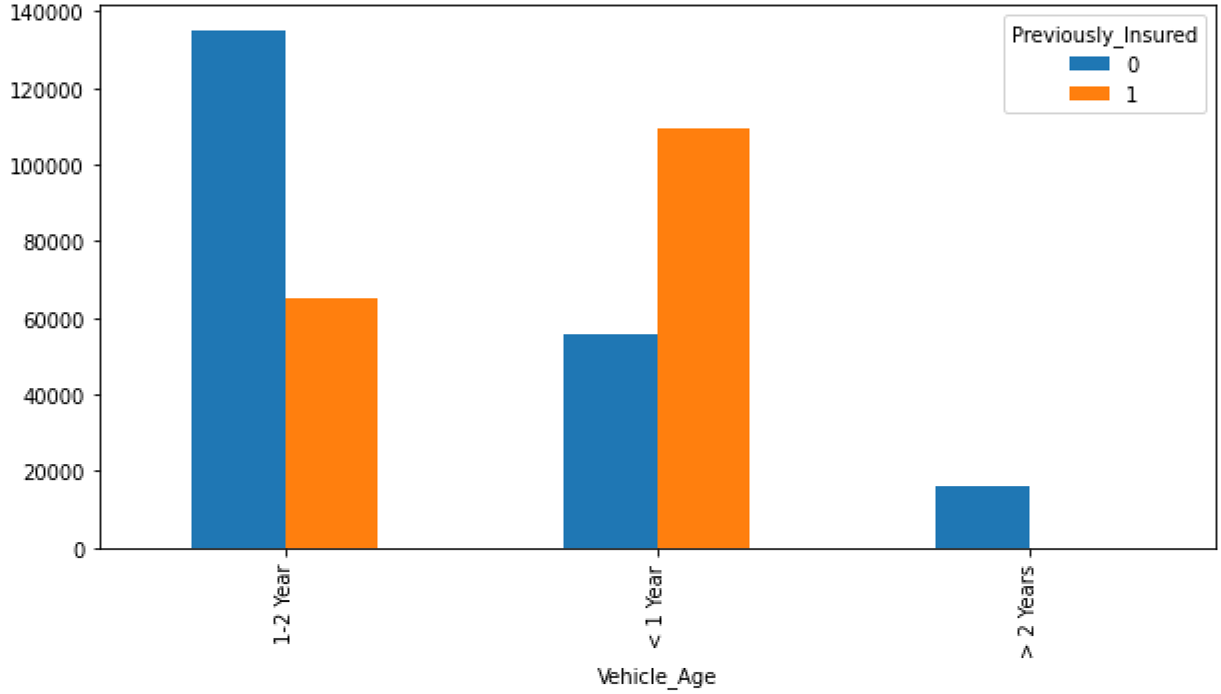
In [74]:
```
for i in df_cat.columns:
  if i == 'Vehicle_Age':
    continue
  else:
    pd.crosstab(df['Vehicle_Age'],df[i]).plot(kind='bar')
    plt.show()
```

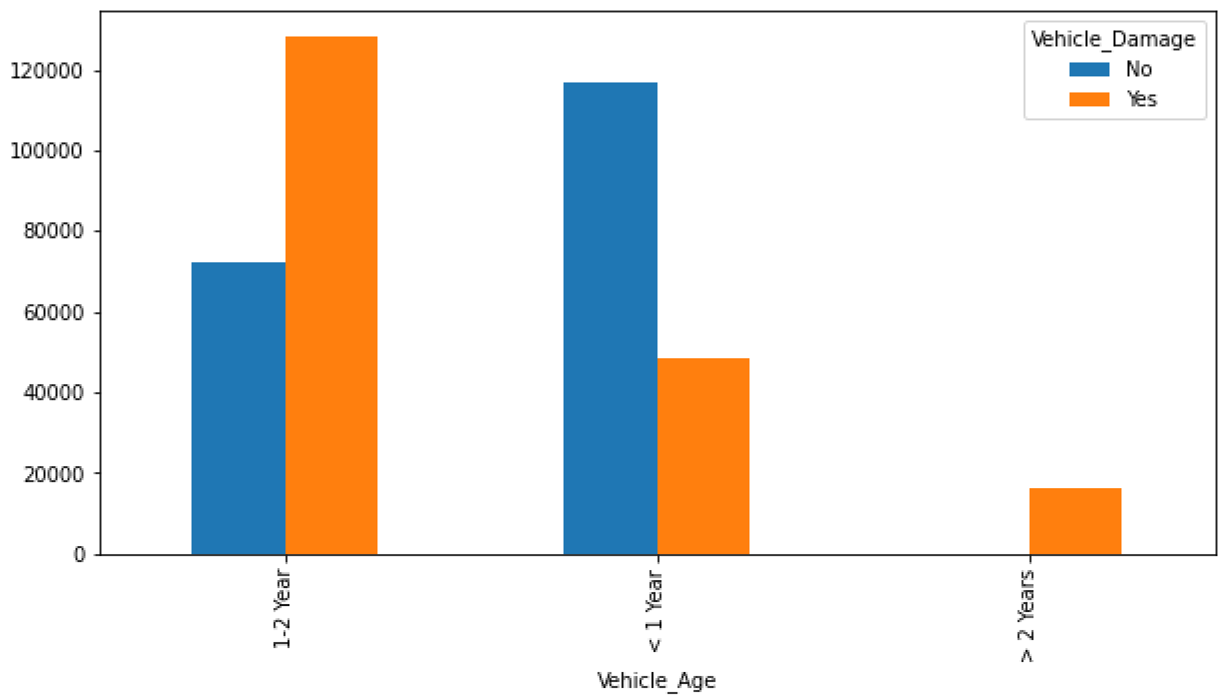






Region_Code
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
31.0
32.0
33.0
34.0
35.0
36.0
37.0
38.0
39.0
40.0
41.0
42.0
43.0
44.0
45.0
46.0
47.0
48.0
49.0
50.0
51.0
52.0
80000
70000
60000
50000
40000
30000
20000
10000
0
1-2 Year
< 1 Year
> 2 Years
Vehicle_Age





Inferences (Vehicle Age vs categorical)

1.Most of the female customers vehicle age is between 1-2 years

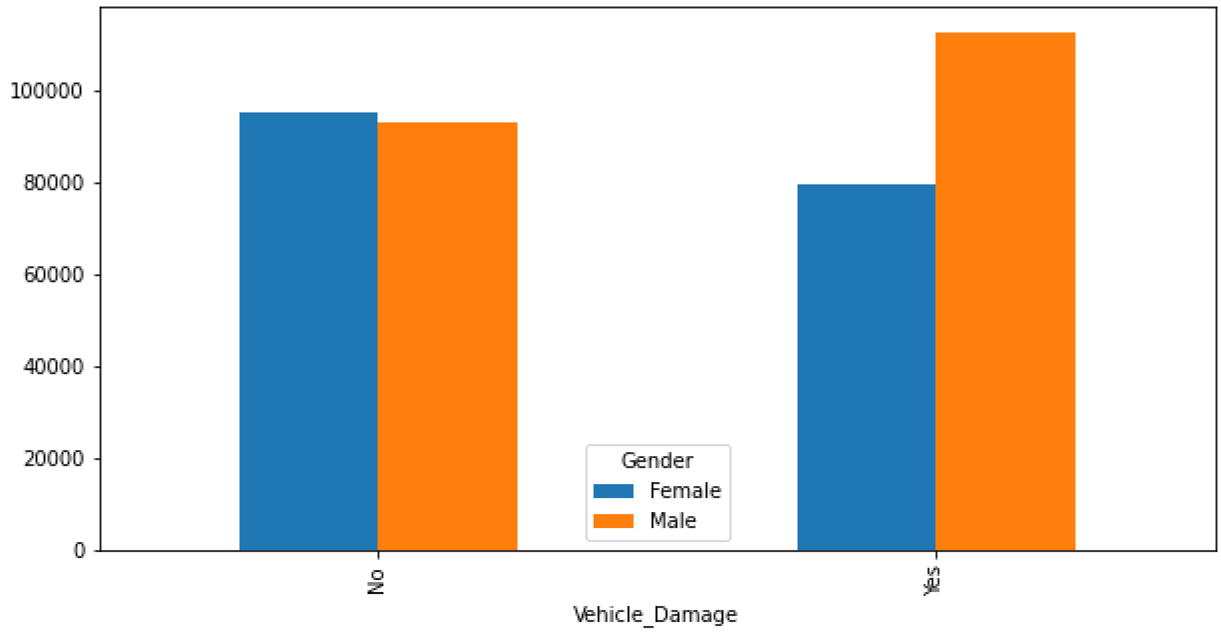
2.In region code 38 most of the customers vehicle age is 1-2 years

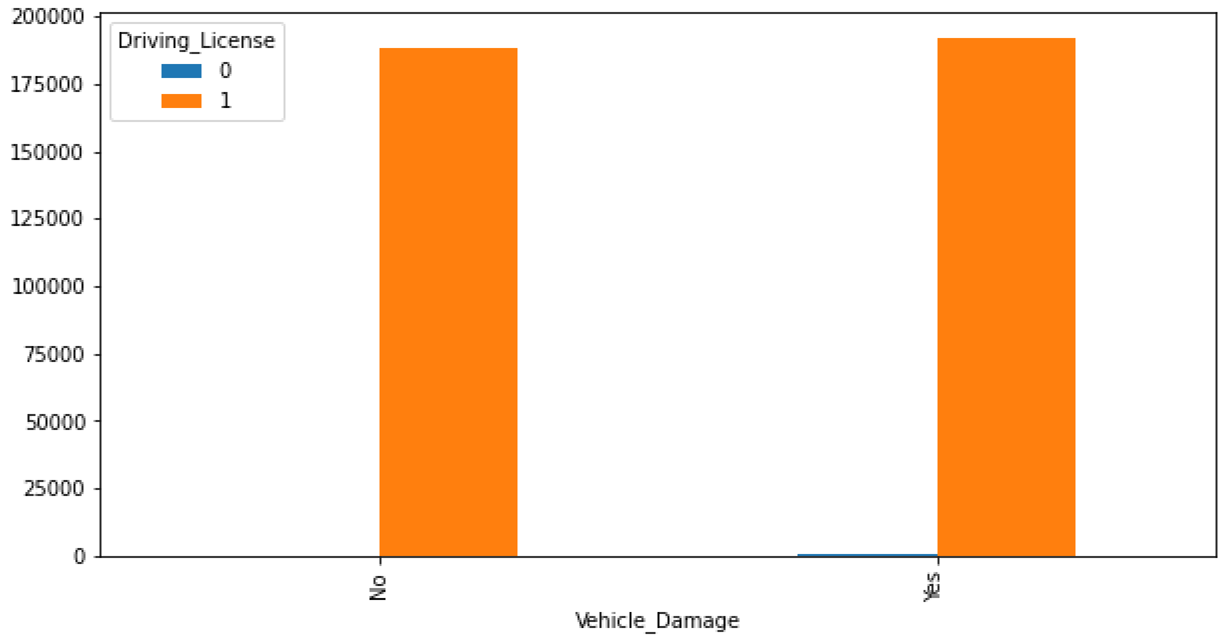
3.Most of the customers whose vehicle age is between 1-2 years are not previously insured.

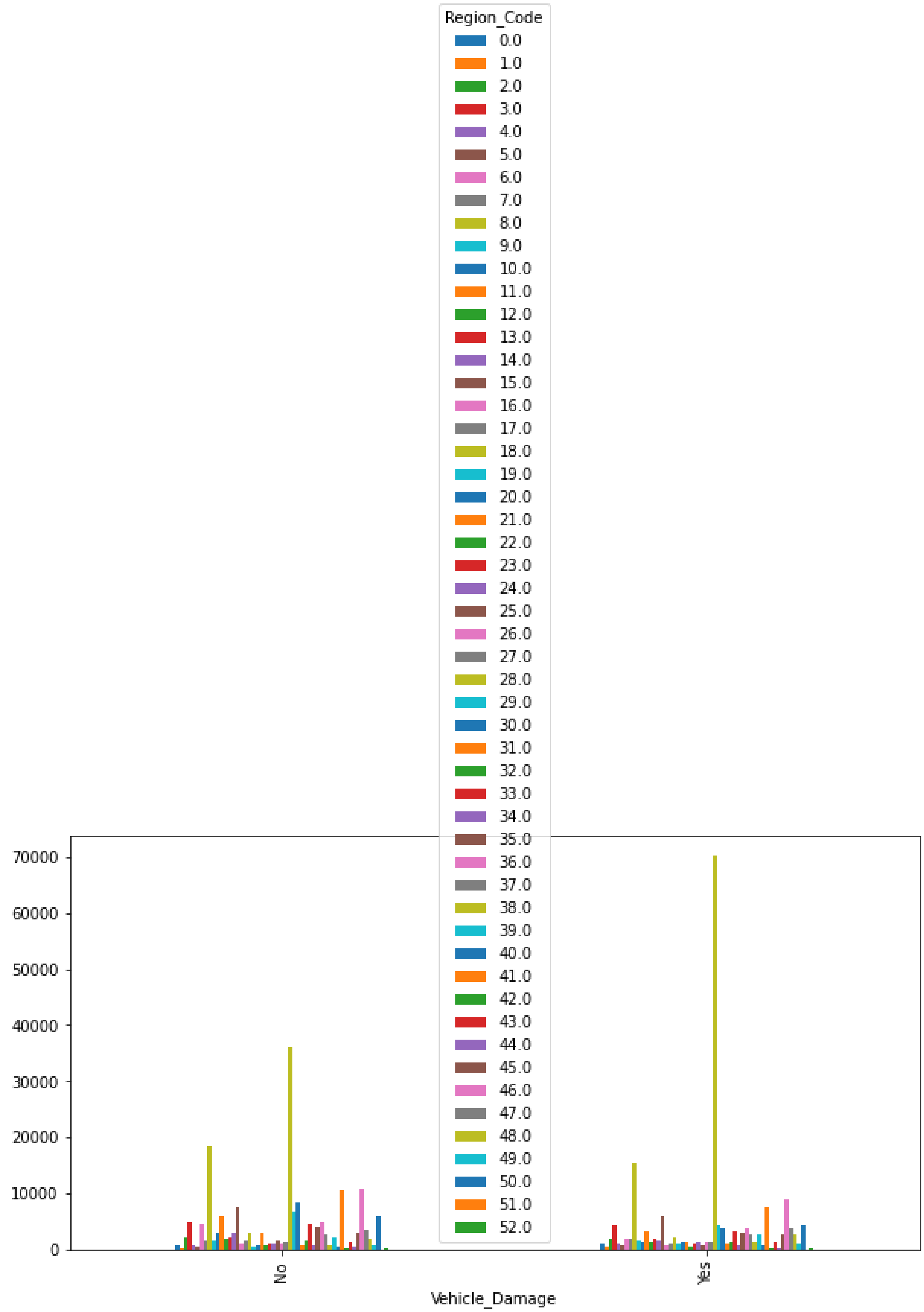
4.Most of the customers whose vehicle age is less than 1 has no vehicle damage

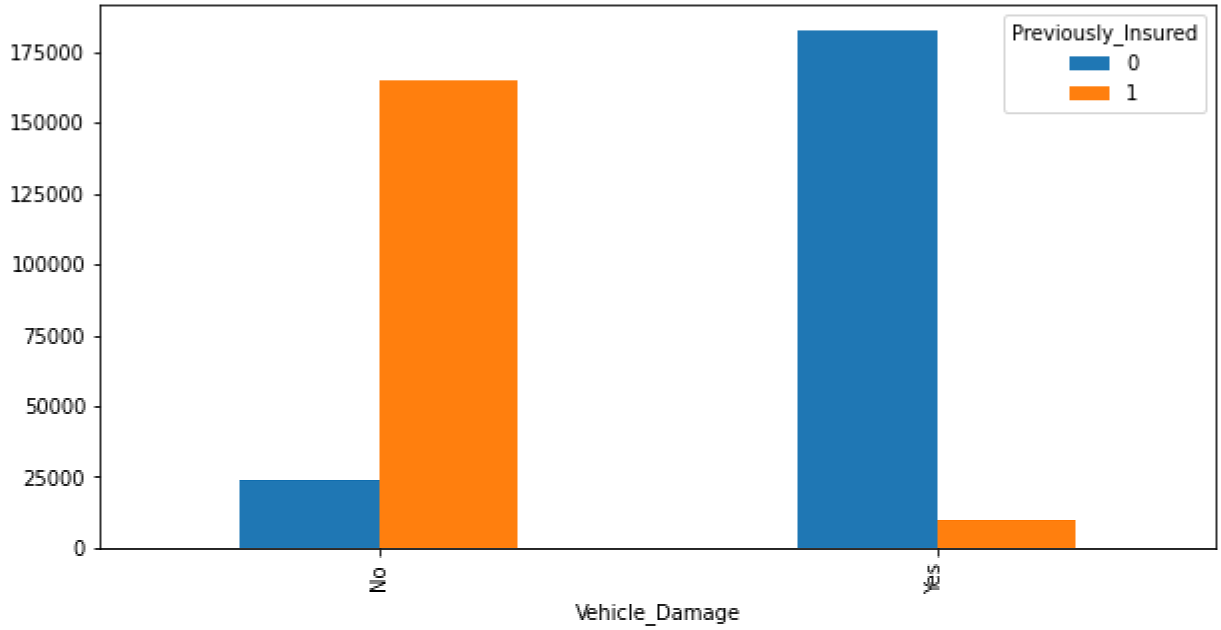
In [75]:

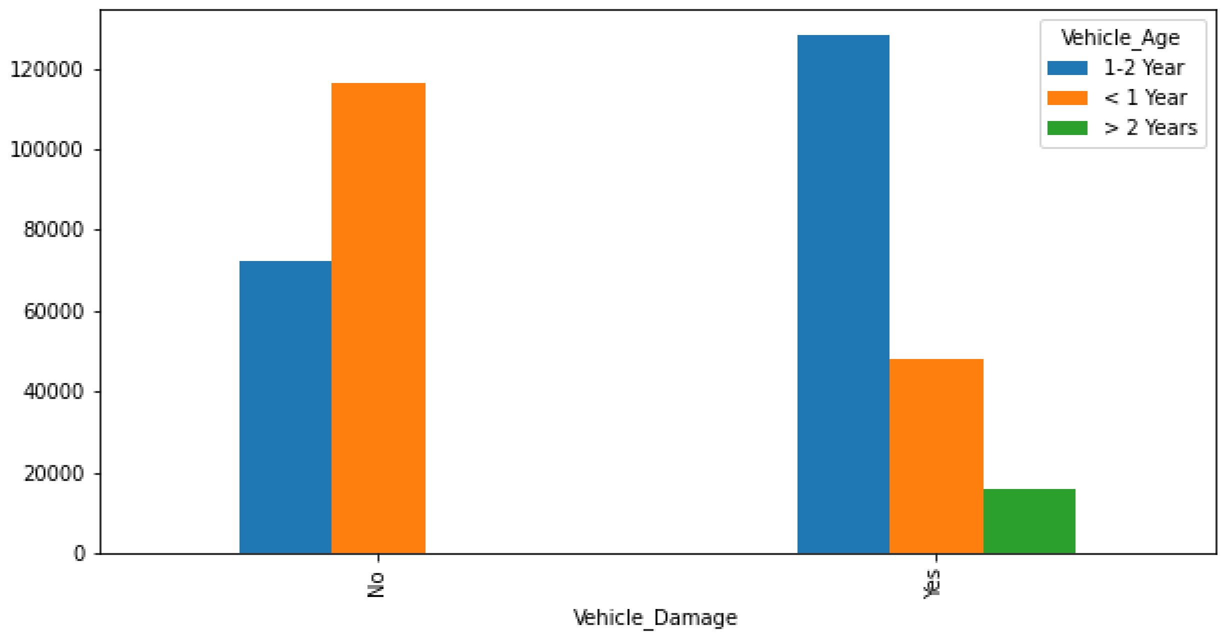
```
for i in df_cat.columns:
  if i == 'Vehicle_Damage':
    continue
  else:
    pd.crosstab(df['Vehicle_Damage'],df[i]).plot(kind='bar')
    plt.show()
```


100000
80000
60000
40000
20000
0
Gender
Female
Male
No
Yes
Vehicle_Damage


200000
175000
150000
125000
100000
75000
50000
25000
0
Driving_License
0
1
No
Yes
Vehicle_Damage


Region_Code
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0
10.0
11.0
12.0
13.0
14.0
15.0
16.0
17.0
18.0
19.0
20.0
21.0
22.0
23.0
24.0
25.0
26.0
27.0
28.0
29.0
30.0
31.0
32.0
33.0
34.0
35.0
36.0
37.0
38.0
39.0
40.0
41.0
42.0
43.0
44.0
45.0
46.0
47.0
48.0
49.0
50.0
51.0
52.0
70000
60000
50000
40000
30000
20000
10000
0
No
Yes
Vehicle_Damage





Inferences (Vehicle Damage vs Categorical)

1.There are more number of mae customers who has a vehicle damage compared to female customers.

2.More number of customers from region 38 has vehicle damage

3.There are more number of customers with vehicle damage who have not previously insured.

4.More number of customers with vehicle age between 1 to 2 has vehicle damage

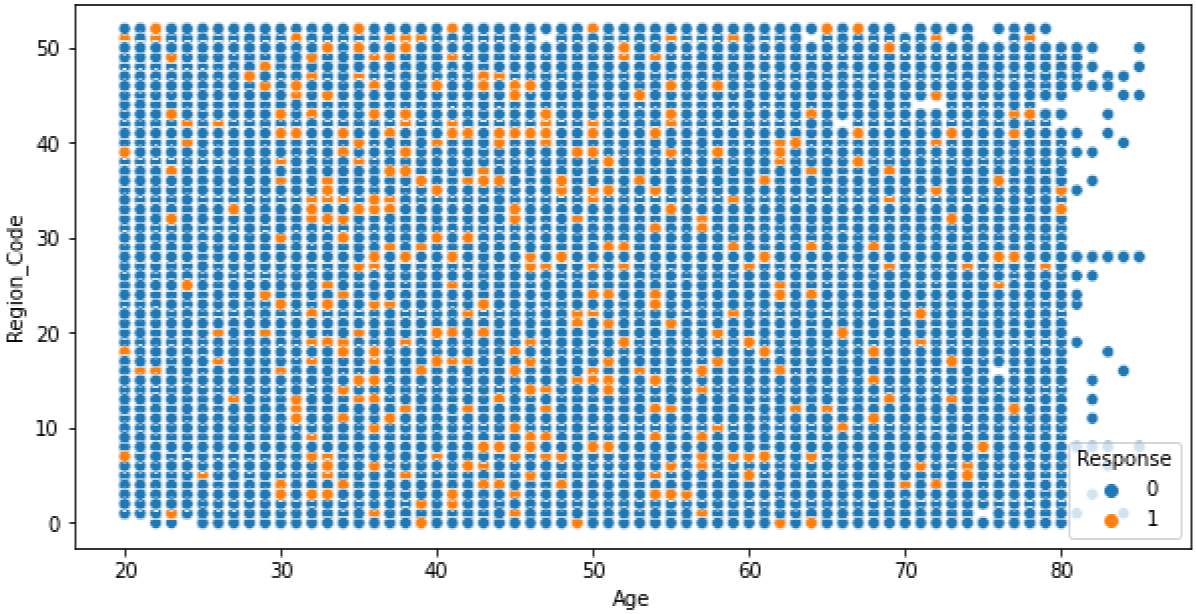
In [76]:

```
# Multivariate
df_num.columns,df_cat.columns
```

Out[76]:

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

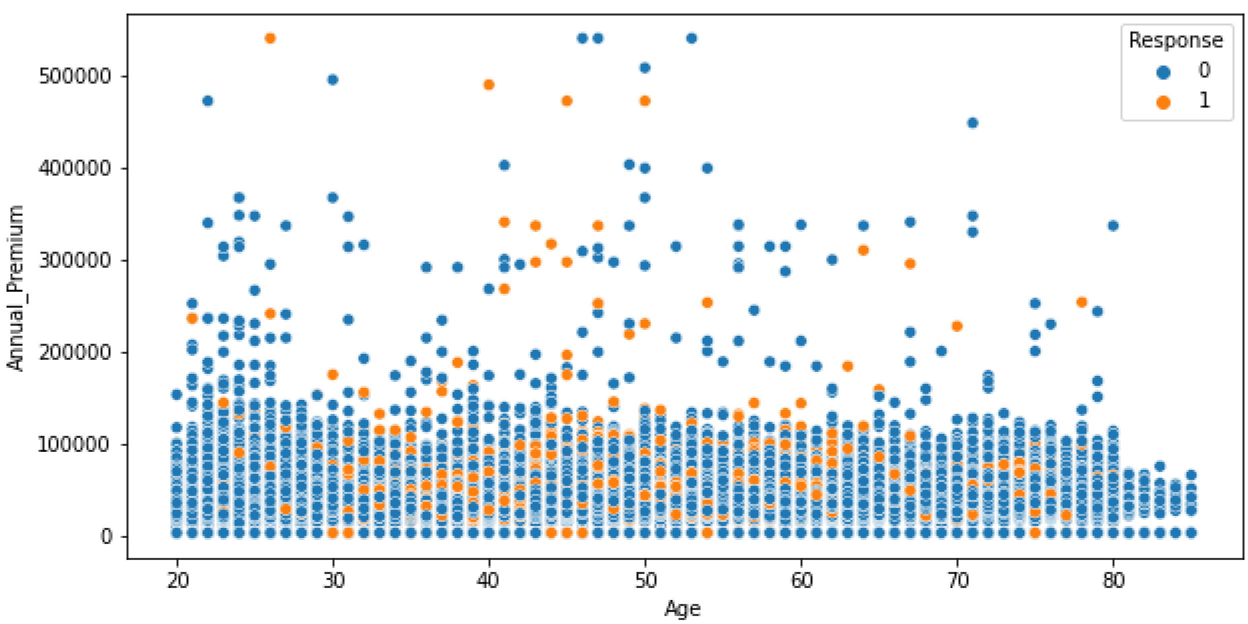
In [77]:
```
sns.scatterplot(df['Age'],df['Region_Code'],hue=df['Response'])
plt.show()
```



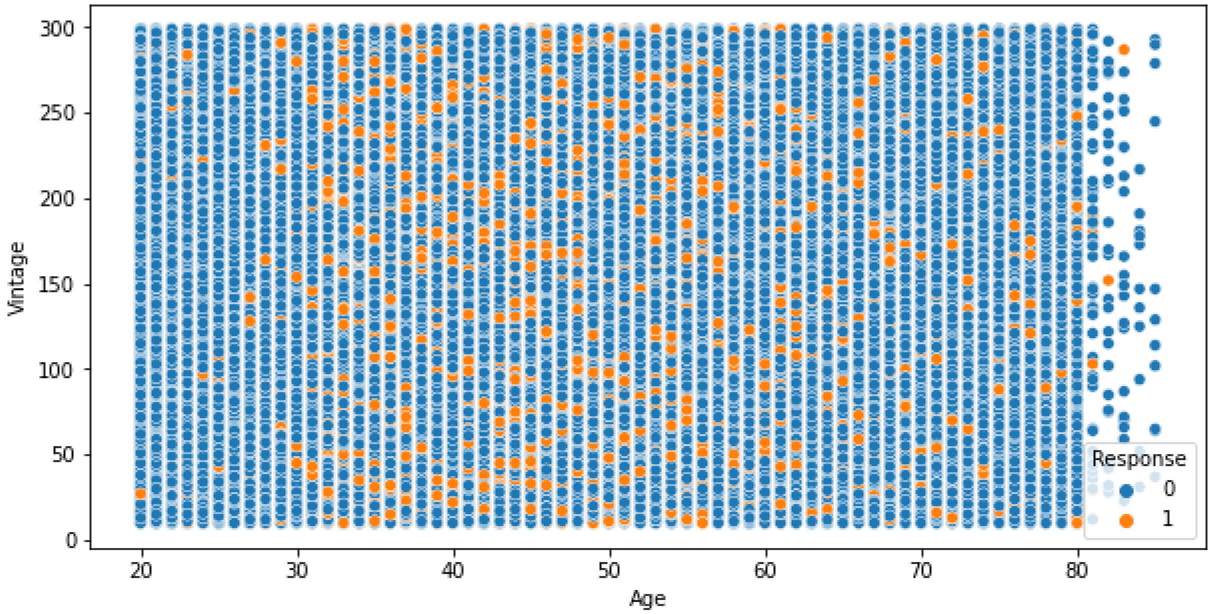
Inferences

Since we have a massively imbalanced dataset we are not able to linearly seperate our data points based on region code and age.

In [78]:
```
sns.scatterplot(df['Age'],df['Annual_Premium'],hue=df['Response'])
plt.show()
```



In [79]:
```
sns.scatterplot(df['Age'],df['Vintage'],hue=df['Response'])
plt.show()
```



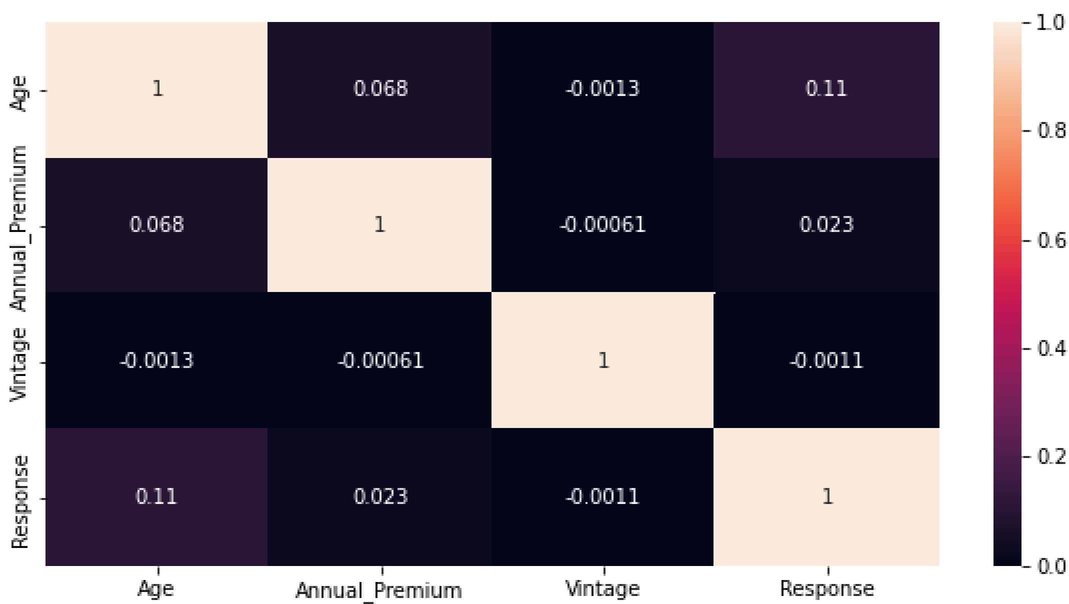
Inference

Since the target variable is massively imbalanced we are not able to find any kind of relationship between (Age,Vintage) and response.

In [80]:
```
df["Policy_Sales_Channel"] = df["Policy_Sales_Channel"].astype(object)
df["Policy_Sales_Channel"].dtype
```

Out[80]: dtype('O')

In [81]:
```
sns.heatmap(df.corr(),annot = True)
plt.show()
```



Inferences

From the above heatmap we can see that all the independent variables has very low correlation with the target variable.

Statistical Testing

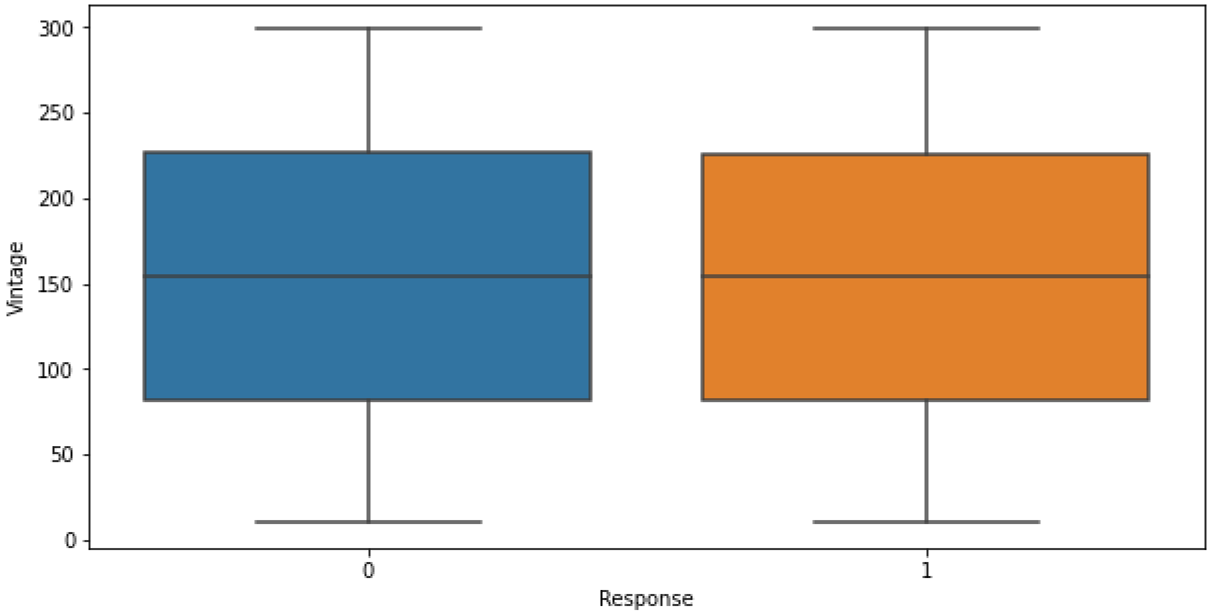
In [82]:
```
from scipy import stats
```

In [83]:
```
p_val = []
sig = []
for i in df.columns:
    if i in df_num:
        stat, p = stats.ttest_ind(df[df['Response'] == 0][i], df[df['Response'] == 1][i])
    else:
        ct = pd.crosstab(df[i], df['Response'])
        stat, p, dof, exp = stats.chi2_contingency(ct)
    p_val.append(p)
    if p < 0.05:
        sig.append('Significant')
    else:
        sig.append("Insignificant")
stats_df = pd.DataFrame({"columns" : df.columns, "p_value" : p_val, "significance" : sig}
stats_df
```

Out[83]:

	columns	p_value	significance
0	Gender	7.665801e-230	Significant
1	Age	0.000000e+00	Significant
2	Driving_License	5.111754e-10	Significant
3	Region_Code	0.000000e+00	Significant
4	Previously_Insured	0.000000e+00	Significant
5	Vehicle_Age	0.000000e+00	Significant
6	Vehicle_Damage	0.000000e+00	Significant
7	Annual_Premium	3.722315e-44	Significant
8	Policy_Sales_Channel	0.000000e+00	Significant
9	Vintage	5.167037e-01	Insignificant
10	Response	0.000000e+00	Significant

In [84]:
```
sns.boxplot(y = df["Vintage"],x= df["Response"])
plt.show()
```



Inference

As we can see that vintage is not a good predictor because the spread of the data for both the labels 0 and 1 are exactly the same.

Data Preprocessing

1.Checking null values

2.Treating Outliers

3.Encoding Categorical features

4.Scaling the data

5.Checking for Multicollinearity

In [85]:
```
df.isnull().sum()
```

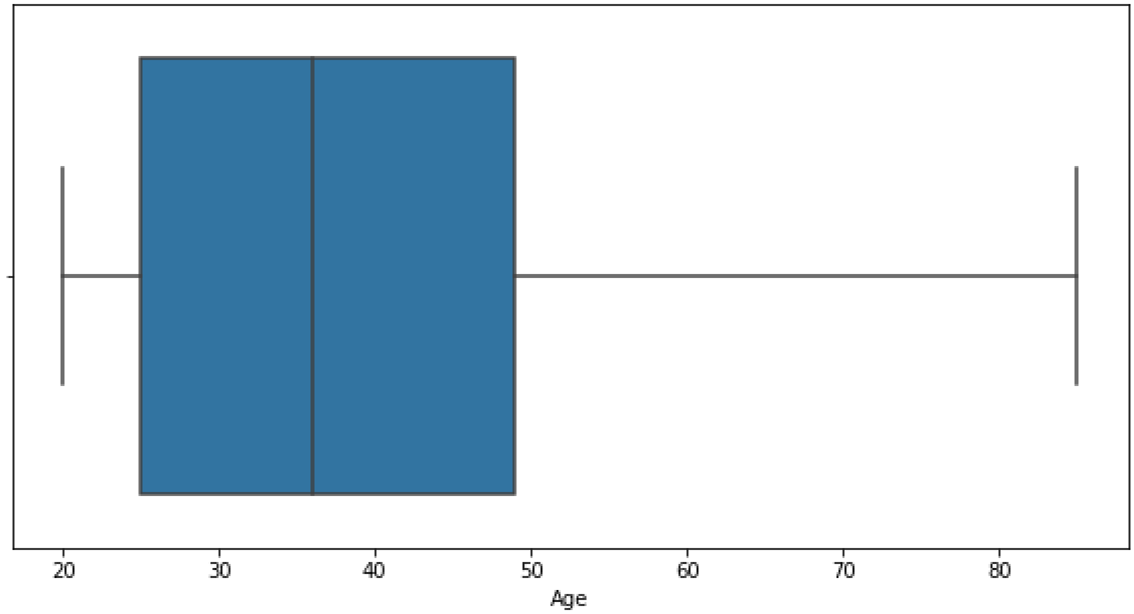
Out[85]:
```
Gender                  0
Age                     0
Driving_License         0
Region_Code             0
Previously_Insured      0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Policy_Sales_Channel    0
Vintage                 0
Response                0
dtype: int64
```

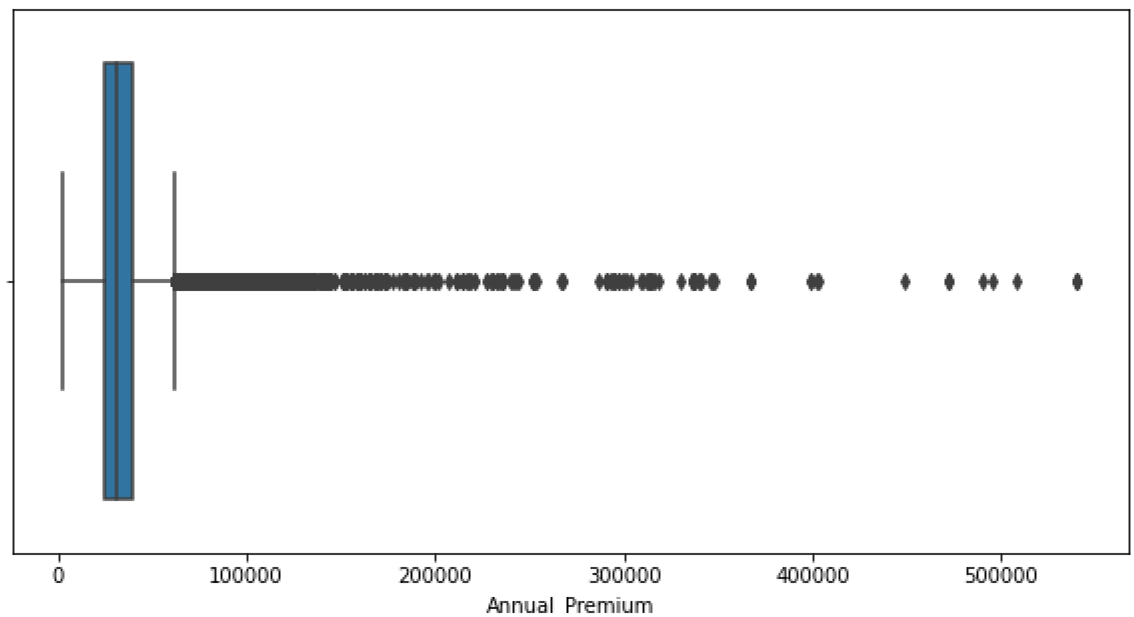
Inference

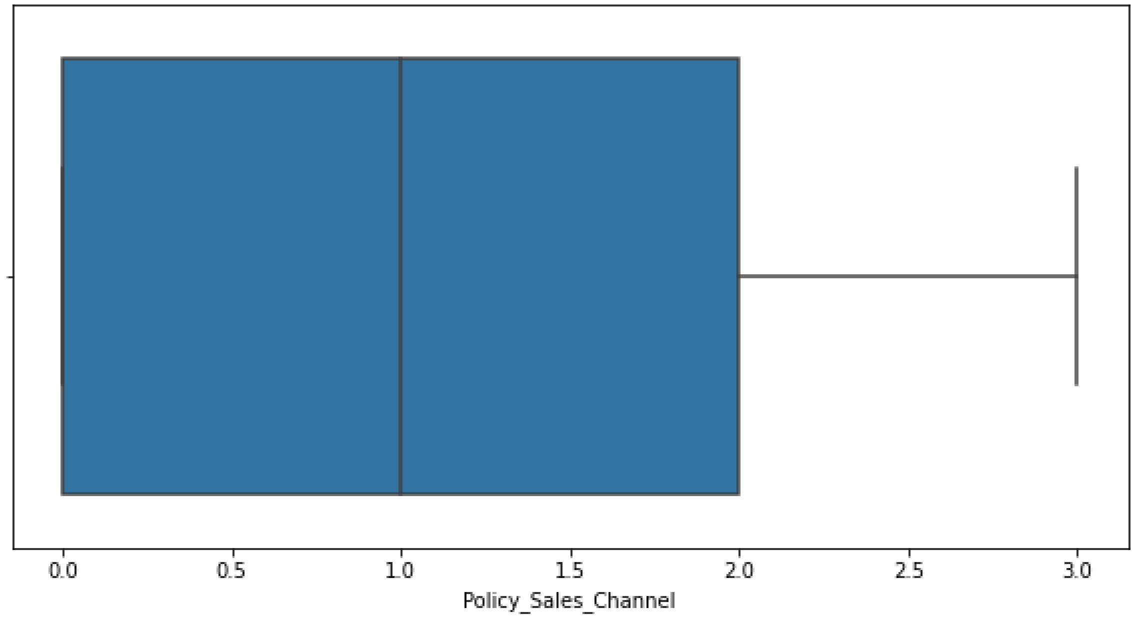
There are no null values in our data

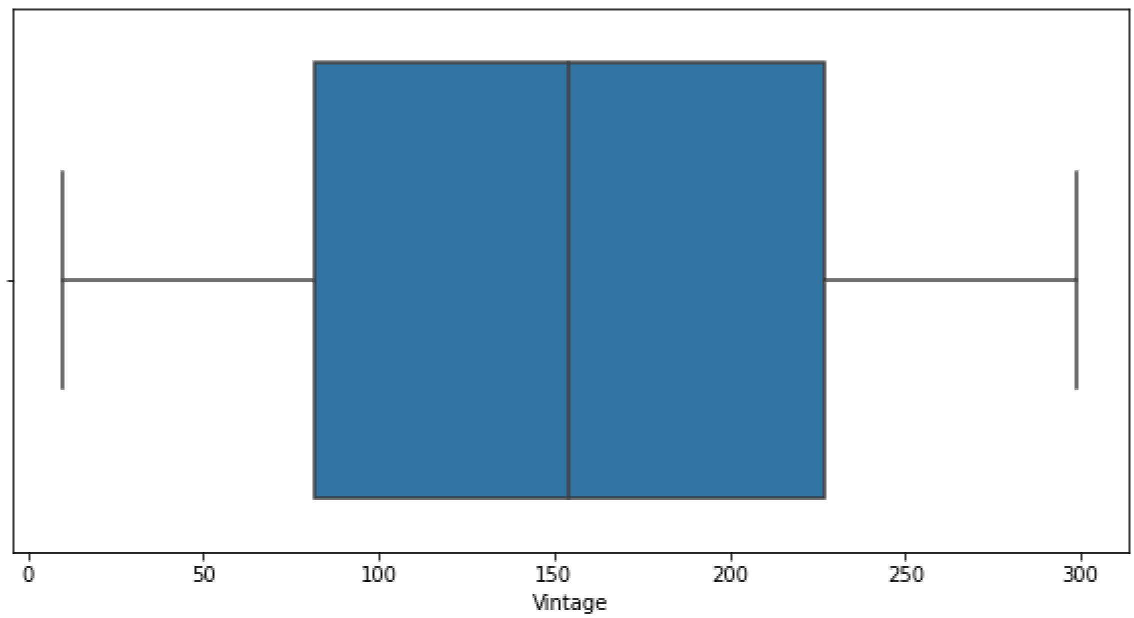
In [86]:
```
for i in df_num:
```

```
  if i == "Response":
    continue
  else:
    sns.boxplot(df_num[i])
    plt.show()
```









From the above boxplots we can see that the column Annual Premium has outliers

In [86]:

```

```

In [87]:

```
df_num.describe()
```

Out[87]:

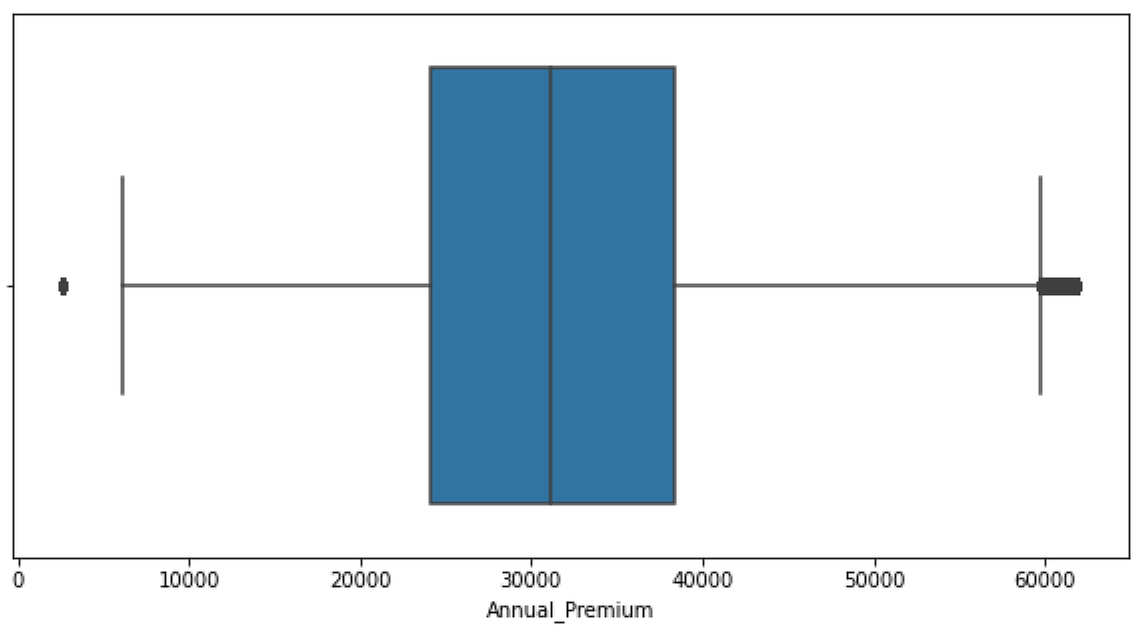
	Age	Annual_Premium	Policy_Sales_Channel	Vintage	Response
count	381109.000000	381109.000000	381109.000000	381109.000000	381109.000000
mean	38.822584	30564.389581	1.320955	154.347397	0.122563
std	15.511611	17213.155057	1.185632	83.671304	0.327936
min	20.000000	2630.000000	0.000000	10.000000	0.000000
25%	25.000000	24405.000000	0.000000	82.000000	0.000000
50%	36.000000	31669.000000	1.000000	154.000000	0.000000
75%	49.000000	39400.000000	2.000000	227.000000	0.000000
max	85.000000	540165.000000	3.000000	299.000000	1.000000

In [88]:
```
len(df_num[df_num["Annual_Premium"] == df_num["Annual_Premium"].max()])
```
Out[88]: 4

In [89]:
```
df_employee = pd.read_csv("DataSet.csv")
```
In [90]:
```
Q1 = df_employee.quantile(0.25)

#calculate the third quartile
Q3 = df_employee.quantile(0.75)

# The Interquartile Range (IQR) is defined as the difference between the third and first
# calculate IQR
IQR = Q3 - Q1

df_employee = df_employee[~((df_employee < (Q1 - 1.5 * IQR)) | (df_employee > (Q3 + 1.5 *
df_employee.shape
```
Out[90]: (324911, 12)

In [91]:
```
df.shape
```
Out[91]: (381109, 11)

In [92]:
```
sns.boxplot(df_employee["Annual_Premium"])
```
Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6aaae807d0>

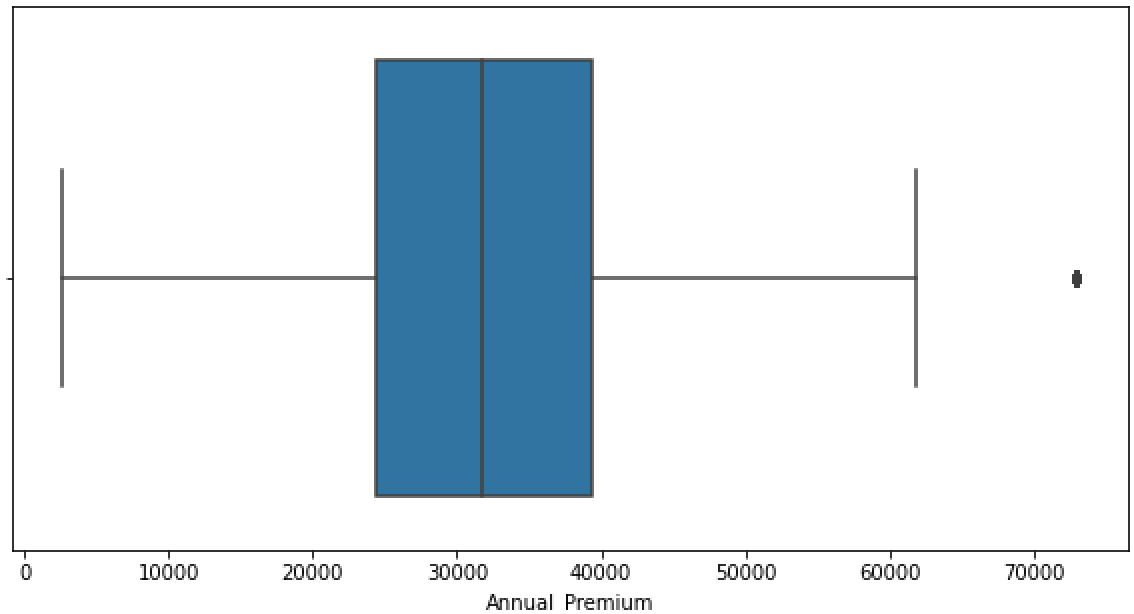


In [93]:
```
s = ((len(df) - len(df_employee))/len(df))*100
```
Inferences

**If we remove the outliers using IQR method we are loosing around 15% of the data points and still there are some outliers left even after removing i.e(out of 381k points we will be loosing 56k points) which leads to loss of information.

**Hence we are proceeding the with capping all the outlier points to the 99th percentile value to prevent the loss of information.

In [94]:
```
def cap(s):
    q1= df_num[s].quantile(0.25)
    q3= df_num[s].quantile(0.75)
    iqr= q3 -q1
    ub= q3 + 1.5 * iqr
    lb= q1 - 1.5 * iqr
    uc = df_num[s].quantile(0.99)
    lc = df_num[s].quantile(0.01)
    ind1=df_num[df_num[s] > ub].index
    ind2=df_num[df_num[s] < lb].index
    df_num.loc[ind1,s]=uc
    df_num.loc[ind2,s]=lc

cap("Annual_Premium")
sns.boxplot(df_num["Annual_Premium"])
plt.show()
```



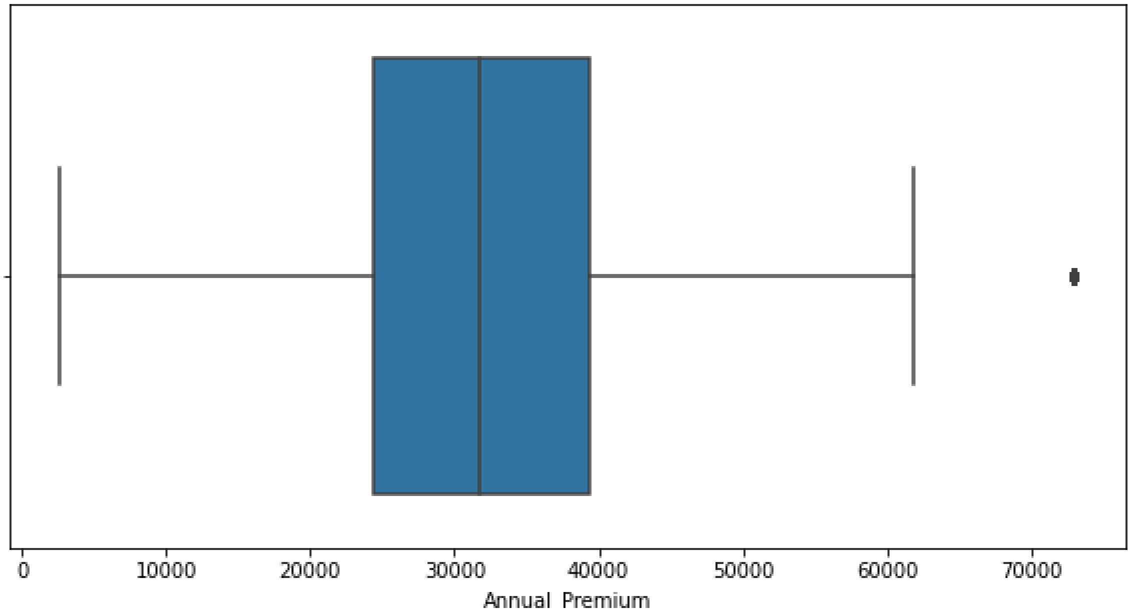
In [95]:
```
len(df_num[df_num["Annual_Premium"] == df_num["Annual_Premium"].max()])
```

Out[95]: 10320

In [96]:
```
q1= df["Annual_Premium"].quantile(0.25)
q3= df["Annual_Premium"].quantile(0.75)
iqr= q3 -q1
ub= q3 + 1.5 * iqr
lb= q1 - 1.5 * iqr
uc = df["Annual_Premium"].quantile(0.99)
lc = df["Annual_Premium"].quantile(0.01)
ind1=df[df["Annual_Premium"] > ub].index
ind2=df[df["Annual_Premium"] < lb].index
df.loc[ind1,"Annual_Premium"]=uc
df.loc[ind2,"Annual_Premium"]=lc
```

In [97]:
```
sns.boxplot(df["Annual_Premium"])
```

Out[97]: <matplotlib.axes._subplots.AxesSubplot at 0x7f6ab0627510>



In [98]:
```
len(df[df["Annual_Premium"] == df["Annual_Premium"].max()])
```

Out[98]: 10320

In [99]:
```
idx = df[df["Annual_Premium"] == df["Annual_Premium"].max()].index
```

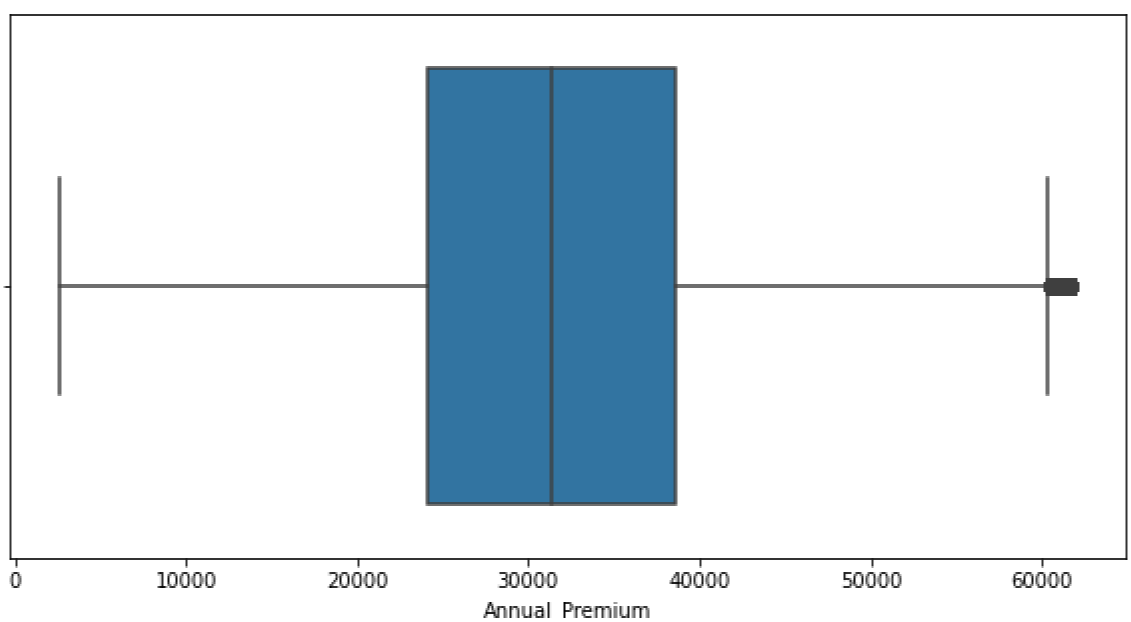
In [100…
```
df.drop(index = idx,inplace = True)
```

In [101…
```
df.shape
```

Out[101… (370789, 11)

In [102…
```
sns.boxplot(df["Annual_Premium"])
```

Out[102… <matplotlib.axes._subplots.AxesSubplot at 0x7f6ab0297d10>



In [103…
```
len(df[df["Annual_Premium"] > 60000])
```

Out[103…
```
1839
```

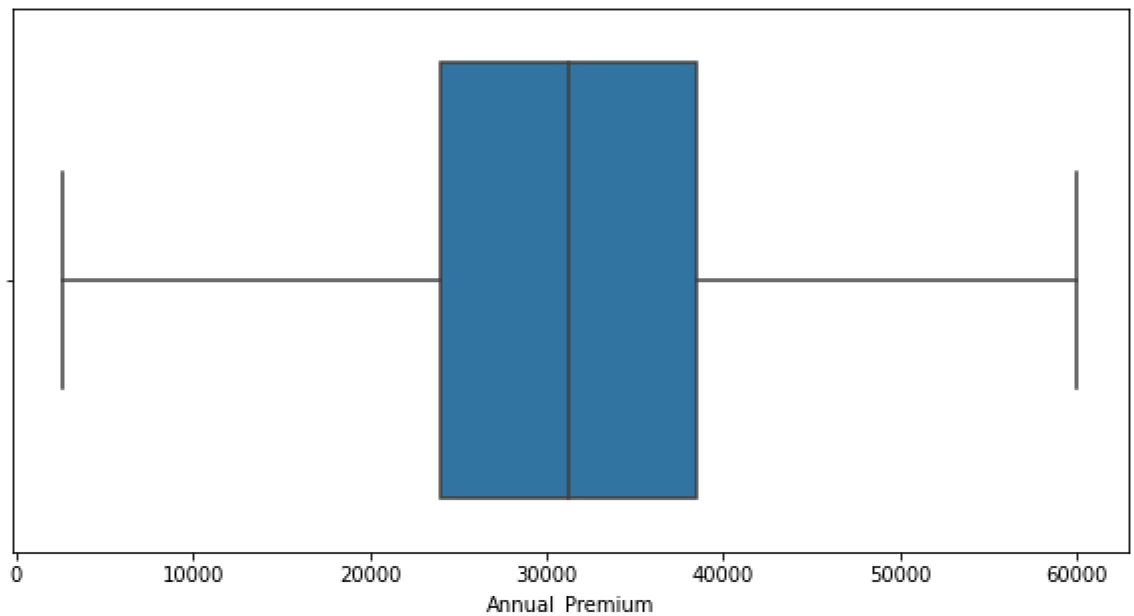
In [104…
```
ind = df[df["Annual_Premium"] > 60000].index
```

In [105…
```
df.drop(index = ind,inplace = True)
df.shape
```

Out[105…
```
(368950, 11)
```

In [106…
```
sns.boxplot(df["Annual_Premium"])
```

Out[106…
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ab1231810>
```



Inference

After outlier Treatment 13k rows has been removed approximately.

In [107…
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 368950 entries, 1 to 381109
Data columns (total 11 columns):
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   Gender                368950 non-null  object 
 1   Age                   368950 non-null  int64  
 2   Driving_License       368950 non-null  object 
 3   Region_Code           368950 non-null  object 
 4   Previously_Insured    368950 non-null  object 
 5   Vehicle_Age           368950 non-null  object 
 6   Vehicle_Damage        368950 non-null  object 
 7   Annual_Premium        368950 non-null  float64
 8   Policy_Sales_Channel  368950 non-null  object 
 9   Vintage               368950 non-null  int64  
 10  Response              368950 non-null  int64  
dtypes: float64(1), int64(3), object(7)
memory usage: 33.8+ MB
```

In [108…

```
df["Region_Code"] = df["Region_Code"].astype(int)
df["Response"] = df["Response"].astype(object)
```

In [109…

```
num_df = df.select_dtypes(include = np.number)
cat_df = df.select_dtypes(exclude = np.number)
print(num_df.shape)
print(cat_df.shape)
```

```
(368950, 4)
(368950, 7)
```

In [110…

```
cat_df.columns
```

Out[110…

```
Index(['Gender', 'Driving_License', 'Previously_Insured', 'Vehicle_Age',
       'Vehicle_Damage', 'Policy_Sales_Channel', 'Response'],
      dtype='object')
```

In [111…

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
df_num_scaled = sc.fit_transform(num_df)

df_scaled = pd.DataFrame(df_num_scaled,columns = num_df.columns)
df_scaled.describe()
```

Out[111…

	Age	Region_Code	Annual_Premium	Vintage
count	3.689500e+05	3.689500e+05	3.689500e+05	3.689500e+05
mean	1.803165e-16	2.710970e-16	-1.733848e-15	-1.317523e-16
std	1.000001e+00	1.000001e+00	1.000001e+00	1.000001e+00
min	-1.208560e+00	-1.984458e+00	-1.812375e+00	-1.725077e+00
25%	-8.843795e-01	-8.587822e-01	-3.461502e-01	-8.645083e-01
50%	-1.711823e-01	1.168032e-01	1.470408e-01	-3.939237e-03
75%	6.716872e-01	7.171634e-01	6.405055e-01	8.685821e-01
max	3.005787e+00	1.917884e+00	2.114671e+00	1.729151e+00

In [112…

```
df_scaled.shape
df_scaled.drop(columns = "Region_Code",inplace = True,axis = 1)
```

In [113…

```
cat_df1 = pd.get_dummies(cat_df,drop_first = True)
cat_df1 = pd.DataFrame(cat_df1,columns = cat_df1.columns)

cat_df1.shape,num_df.shape
```

Out[113…

```
((368950, 10), (368950, 4))
```

In [114…

```
cat_df2 = pd.concat([cat_df1,num_df["Region_Code"]],axis = 1)
cat_df2.shape
```

Out[114…

```
(368950, 11)
```

In [115…

```
df_scaled.shape,cat_df2.shape

df_scaled.reset_index(drop = True,inplace = True)
cat_df2.reset_index(drop = True,inplace = True)
```

In [116…

```
final_df = pd.concat([df_scaled,cat_df2],axis = 1)
final_df.shape
```

Out[116… (368950, 14)

In [117…

```
final_df.head()
#final_df.to_csv("preprocessed_dataset.csv")
final_df["Response_1"].value_counts()
final_df.to_csv("preprocessed_dataset1.csv")
final_df.head()
```

Out[117…

	Age	Annual_Premium	Vintage	Gender_Male	Driving_License_1	Previously_Insured_1	Veh
0	0.347507	0.776723	0.749059	1	1	0	
1	2.422262	0.303178	0.342679	1	1	0	
2	0.542015	0.628869	-1.521887	1	1	0	
3	-1.143724	-0.033397	0.581726	1	1	1	
4	-0.625035	-0.110267	-1.378459	0	1	1	

In [118…

```
x = final_df.drop(columns='Response_1')
from statsmodels.stats.outliers_influence import variance_inflation_factor as vif
vf=[ vif(x.values,i) for i in range(x.shape[1]) ]
pd.DataFrame(vf,index=x.columns,columns=['vif'])
```

Out[118…

	vif
Age	2.766426
Annual_Premium	1.029337
Vintage	1.000039
Gender_Male	2.246470
Driving_License_1	26.487611
Previously_Insured_1	5.738634
Vehicle_Age_< 1 Year	7.000352
Vehicle_Age_> 2 Years	1.126348
Vehicle_Damage_Yes	6.337217
Policy_Sales_Channel_1.0	3.255732
Policy_Sales_Channel_2.0	3.159075
Policy_Sales_Channel_3.0	2.697849
Region_Code	4.918244

Inference

We can see that there is no multicollinearity in the data

In [119…

```
final_df.rename(columns={'Vehicle_Age_< 1 Year':'Vehicle_Age_1_Year','Vehicle_Age_> 2 Yea
```

In [120…

```
final_df["Response_1"] = final_df["Response_1"].astype("int")
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score,confusion_matrix,plot_confusion_matrix,f1_scor
lr = LogisticRegression()

x = final_df.drop(columns='Response_1')
y = final_df["Response_1"]
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2)
lr.fit(x_train,y_train)
y_pred = lr.predict(x_test)
#plot_confusion_matrix(lr,x_train,y_train)
#f1_score(y_test,y_pred)
target_names = ["class_0","class_1"]
print(classification_report(y_test,y_pred,target_names = target_names))
```

```
              precision    recall  f1-score   support

     class_0       0.88      1.00      0.93     64782
     class_1       0.17      0.00      0.00      9008

    accuracy                           0.88     73790
   macro avg       0.52      0.50      0.47     73790
weighted avg       0.79      0.88      0.82     73790
```

In [121…

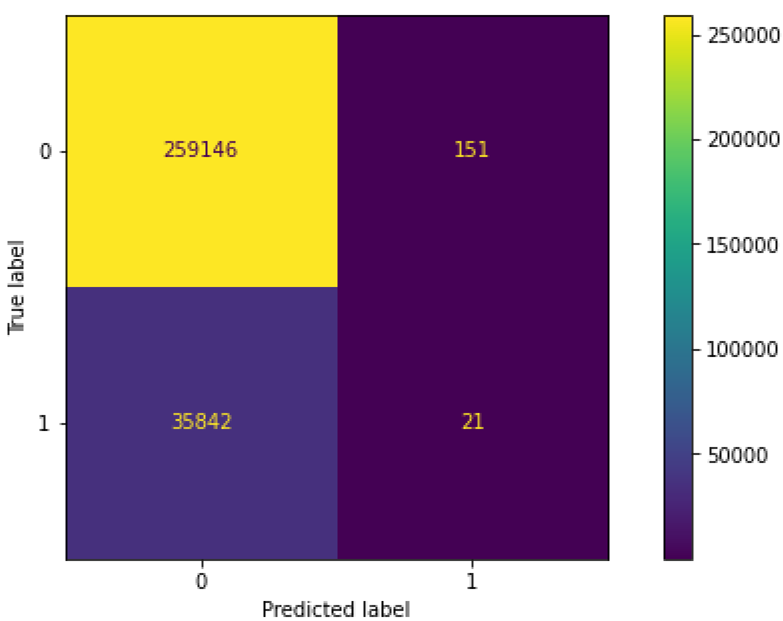
```
y_train.value_counts()
```

Out[121…

```
0    259297
1     35863
Name: Response_1, dtype: int64
```

In [122…

```
plot_confusion_matrix(lr,x_train,y_train)
plt.show()
```



In [123…

```
from sklearn.tree import DecisionTreeClassifier
```

```
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred_train = dt.predict(x_train)
y_pred_test = dt.predict(x_test)
print("classification report train:")
print(classification_report(y_train,y_pred_train))
```

```
classification report train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    259297
           1       1.00      1.00      1.00     35863

    accuracy                           1.00    295160
   macro avg       1.00      1.00      1.00    295160
weighted avg       1.00      1.00      1.00    295160
```

In [124…

```
dt.fit(x_train,y_train)
y_pred_train = dt.predict(x_train)
y_pred_test = dt.predict(x_test)
print("classification report train:")
print(classification_report(y_test,y_pred_test))
```

```
classification report train:
              precision    recall  f1-score   support

           0       0.90      0.89      0.90     64782
           1       0.29      0.30      0.29      9008

    accuracy                           0.82     73790
   macro avg       0.59      0.60      0.60     73790
weighted avg       0.83      0.82      0.82     73790
```

In [125…

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred_train = rf.predict(x_train)
print("classification report train:")
print(classification_report(y_train,y_pred_train))
```

```
classification report train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    259297
           1       1.00      1.00      1.00     35863

    accuracy                           1.00    295160
   macro avg       1.00      1.00      1.00    295160
weighted avg       1.00      1.00      1.00    295160
```

In [126…

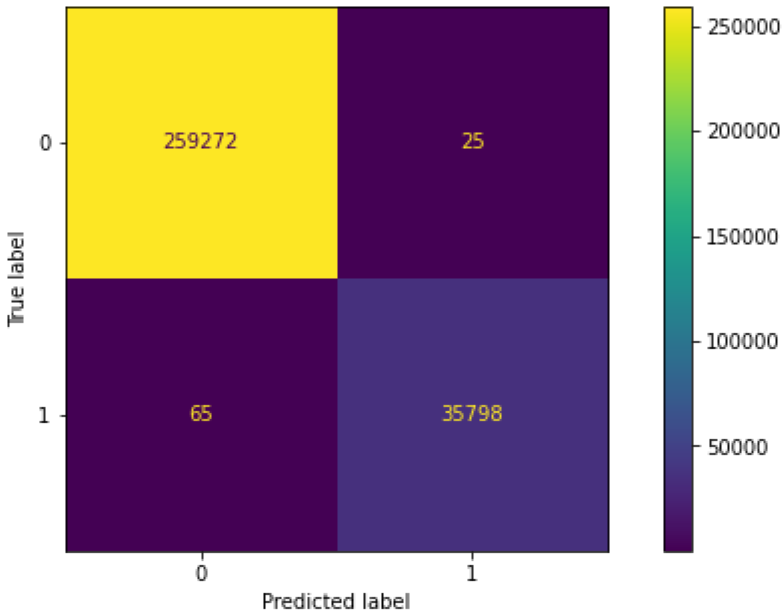
```
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred_test = rf.predict(x_test)
print("classification report train:")
print(classification_report(y_test,y_pred_test))
```

```
classification report train:
              precision    recall  f1-score   support

           0       0.89      0.97      0.93     64782
           1       0.36      0.13      0.19      9008
```

```
    accuracy                           0.87     73790
   macro avg       0.62      0.55      0.56     73790
weighted avg       0.82      0.87      0.84     73790
```

In [127…

```
plot_confusion_matrix(rf,x_train,y_train)
plt.show()
```



Above is the Classification report before balancing the dataset for each model

The model we have built is an underfit model we have to treat the imbalance in the target variable before looking at the results.

In [128…

```
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=42)
x_resampled, y_resampled = sm.fit_resample(x, y)
```

In [129…

```
x_train,x_test,y_train,y_test = train_test_split(x_resampled,y_resampled,test_size =  0.3
lr.fit(x_train,y_train)
y_pred_train_lr = lr.predict(x_train)
print("Classification report for training:")
print(classification_report(y_train,y_pred_train_lr))
```

```
Classification report for training:
              precision    recall  f1-score   support

           0       0.95      0.60      0.74    226839
           1       0.71      0.97      0.82    226871

    accuracy                           0.78    453710
   macro avg       0.83      0.78      0.78    453710
weighted avg       0.83      0.78      0.78    453710
```

In [130…

```
lr.fit(x_train,y_train)
y_pred_test_lr = lr.predict(x_test)
target_names = ["class_0","class_2"]
print("classification report for testing:")
print(classification_report(y_test,y_pred_test_lr))
```

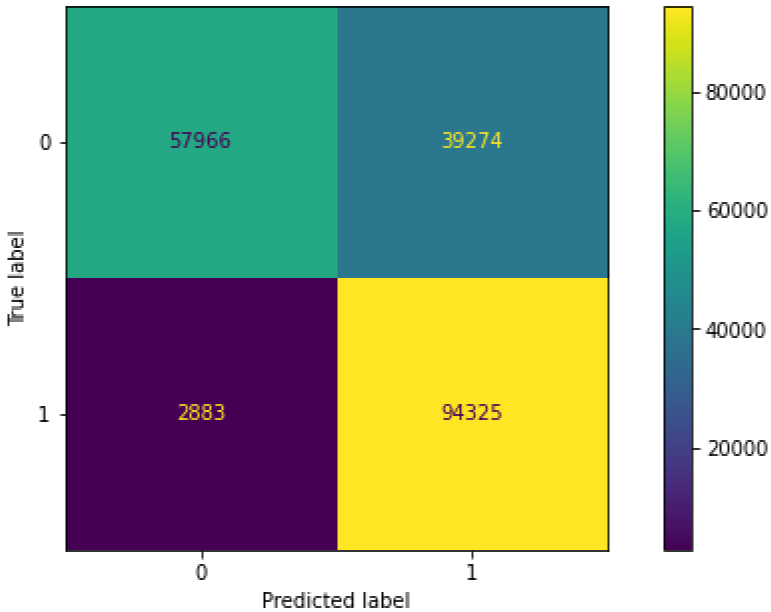
```
classification report for testing:
              precision    recall  f1-score   support
```

```
              0       0.95      0.60      0.73     97240
              1       0.71      0.97      0.82     97208

       accuracy                           0.78    194448
      macro avg       0.83      0.78      0.78    194448
   weighted avg       0.83      0.78      0.78    194448
```

Classification report after balancing the dataset

In [131…

```
plot_confusion_matrix(lr,x_test,y_test)
plt.show()
```



The above built model is a baseline model without any hyperparameter tuned

In [132…

```
y_resampled.value_counts()
```

Out[132…

```
1    324079
0    324079
Name: Response_1, dtype: int64
```

In [133…

```
#x_res1 = x_resampled.iloc[0:100000,:]
#y_res1 = y_resampled[0:100000]
# from sklearn.model_selection import GridSearchCV
# param_grid = {
#     'penalty' : ['l1', 'l2'],
#     'C' : [100,10,1.0,0.1,0.01]}
# gs = GridSearchCV(estimator = lr,param_grid = param_grid,cv = 10)
# gs.fit(x_train,y_train)
# gs.best_params_
```

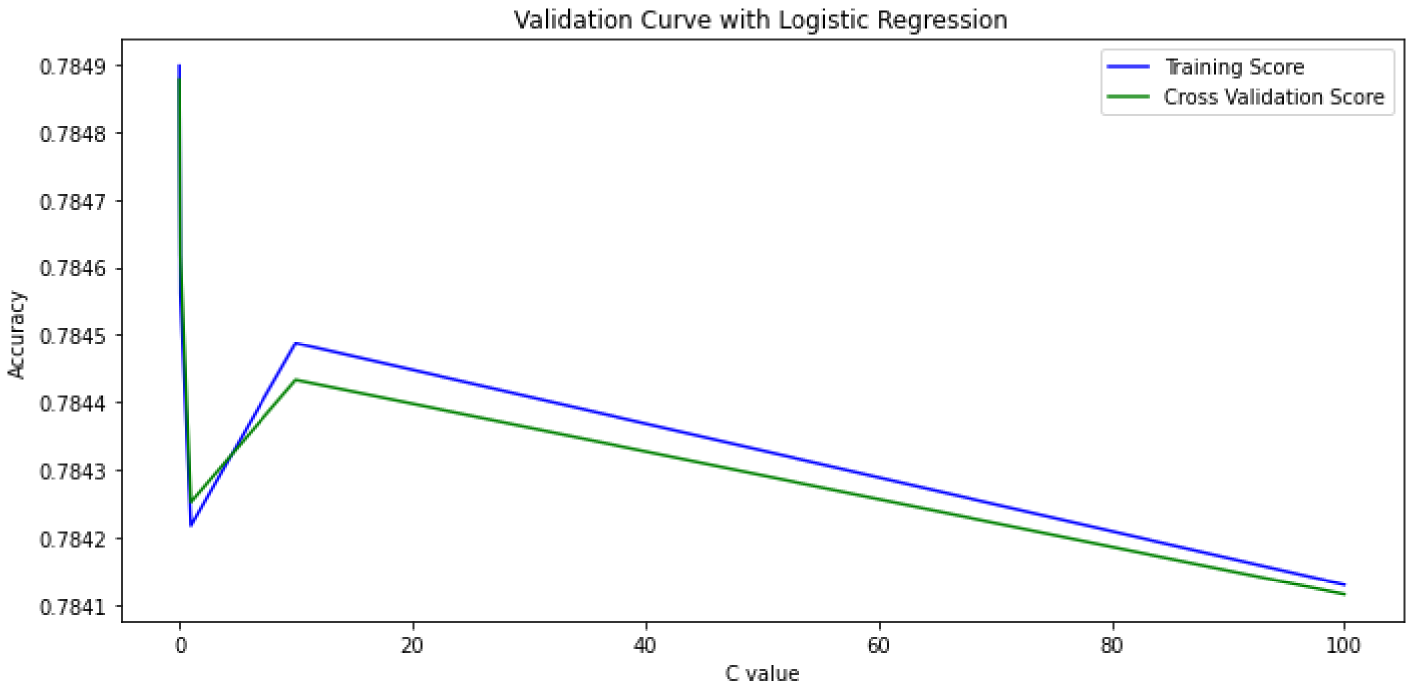
In [133…

```

```

In [134…

```
#x_res1 = x_resampled.iloc[0:100000,:]
#y_res1 = y_resampled[0:100000]
# from sklearn.model_selection import GridSearchCV
# param_grid = {
#     'penalty' : ['l1', 'l2'],
#     'C' : [100,10,1.0,0.1,0.01]}
```

```
# gs = GridSearchCV(estimator = lr,param_grid = param_grid,cv = 10)
# gs.fit(x_resampled,y_resampled)
# gs.best_params_

#Import Required libraries
from sklearn.model_selection import validation_curve

# Setting the range for the parameter (from 1 to 10)
parameter_range = [100,10,1.0,0.1,0.01]

# Calculate accuracy on training and test set using the
# gamma parameter with 5-fold cross validation
train_score, test_score = validation_curve(LogisticRegression(), x_train, y_train,
                                                                         param_name = "C",
                                                                         param_range = par
                                                                                 cv = 10,

# Calculating mean and standard deviation of training score
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

# Calculating mean and standard deviation of testing score
mean_test_score = np.mean(test_score, axis = 1)
std_test_score = np.std(test_score, axis = 1)

# Plot mean accuracy scores for training and testing scores
plt.plot(parameter_range, mean_train_score,
        label = "Training Score", color = 'b')
plt.plot(parameter_range, mean_test_score,
label = "Cross Validation Score", color = 'g')

# Creating the plot
plt.title("Validation Curve with Logistic Regression")
plt.xlabel("C value")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.legend(loc = 'best')
plt.show()
```



Logistic regression after hyperparameter tuning

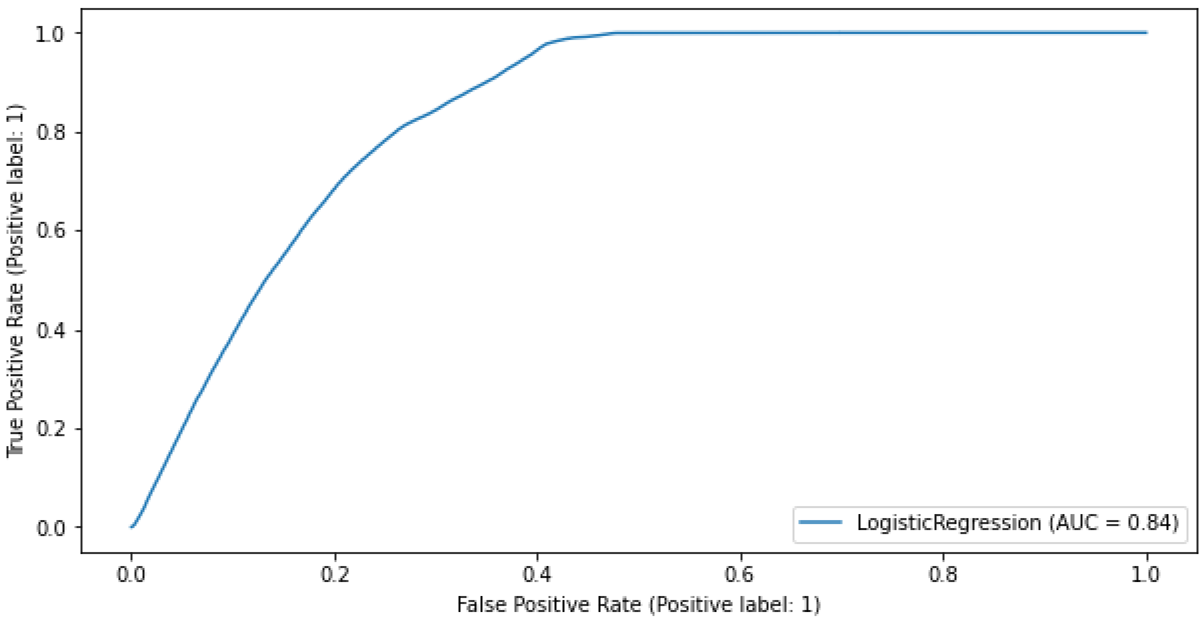
In [135…

```
lr = LogisticRegression(penalty = 'l1',C = 0.01,solver = 'liblinear')
lr.fit(x_train,y_train)
y_pred = lr.predict(x_test)
print(classification_report(y_test,y_pred))
print("Accuracy:",(accuracy_score(y_test,y_pred))*100)
```

```
              precision    recall  f1-score   support

           0       0.96      0.59      0.73     97240
           1       0.71      0.97      0.82     97208

    accuracy                           0.78    194448
   macro avg       0.83      0.78      0.78    194448
weighted avg       0.83      0.78      0.78    194448

Accuracy: 78.36388134616968
```

In [136…

```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(lr,x_test,y_test)
plt.show()
```



Decision Tree Classifier

In [137…

```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
y_pred_train = dt.predict(x_train)
y_pred_test = dt.predict(x_test)
print("classification report train:")
print(classification_report(y_train,y_pred_train))
```

```
classification report train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226839
           1       1.00      1.00      1.00    226871

    accuracy                           1.00    453710
   macro avg       1.00      1.00      1.00    453710
weighted avg       1.00      1.00      1.00    453710
```

In [138…

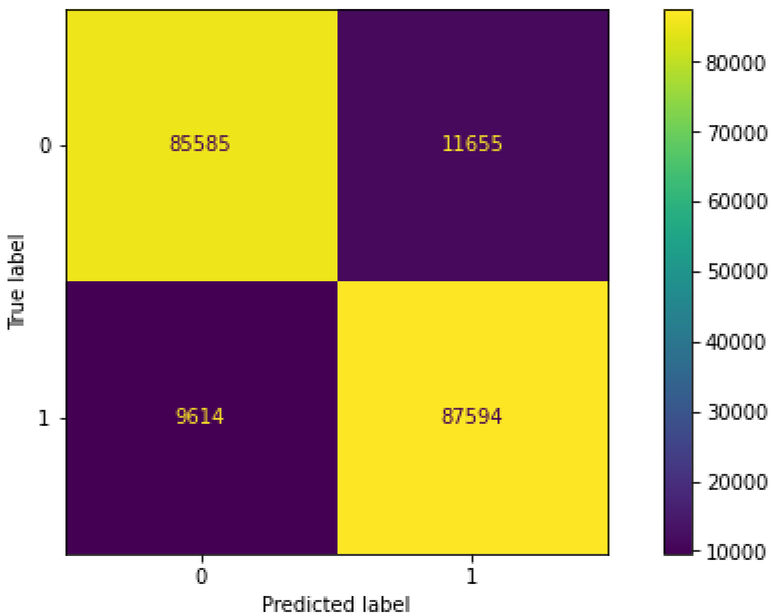
```
print("classification report test:")
```

```
print(classification_report(y_test,y_pred_test))
```

```
classification report test:
              precision    recall  f1-score   support

           0       0.90      0.88      0.89     97240
           1       0.88      0.90      0.89     97208

    accuracy                           0.89    194448
   macro avg       0.89      0.89      0.89    194448
weighted avg       0.89      0.89      0.89    194448
```

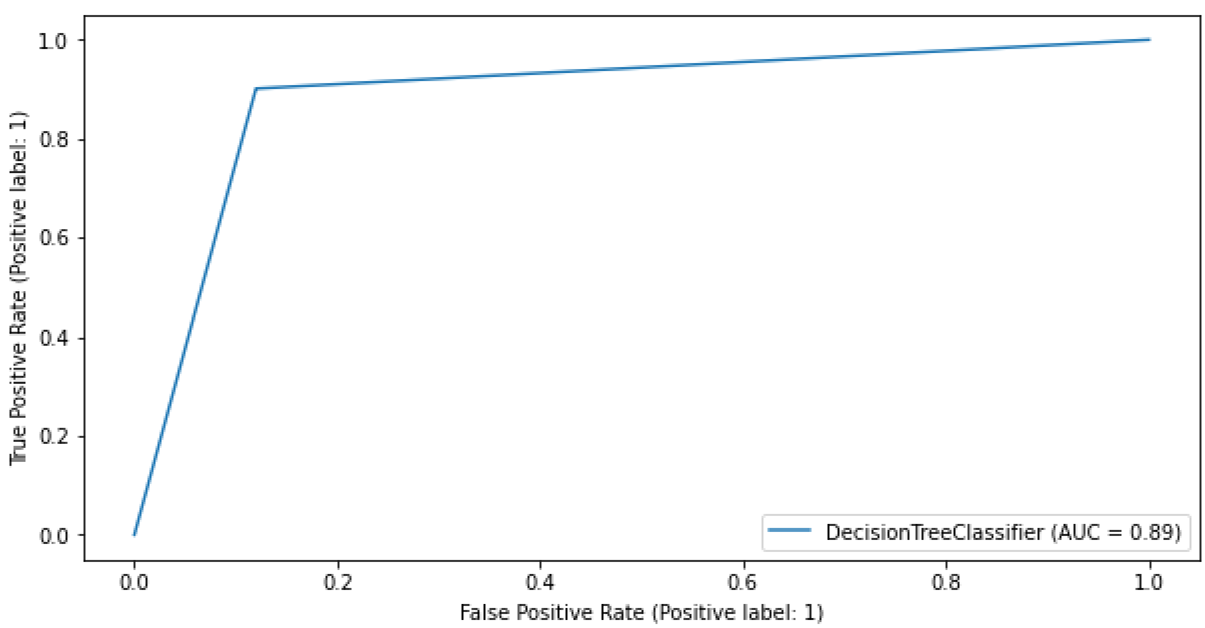
In [139…

```
plot_confusion_matrix(dt,x_test,y_test)
plt.show()
```



In [140…

```
plot_roc_curve(dt,x_test,y_test)
plt.show()
```



Random Forest Classifier

In [141…

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf = RandomForestClassifier()
rf.fit(x_train,y_train)
y_pred_train = rf.predict(x_train)
print("classification report train:")
print(classification_report(y_train,y_pred_train))
```

```
classification report train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226839
           1       1.00      1.00      1.00    226871

    accuracy                           1.00    453710
   macro avg       1.00      1.00      1.00    453710
weighted avg       1.00      1.00      1.00    453710
```

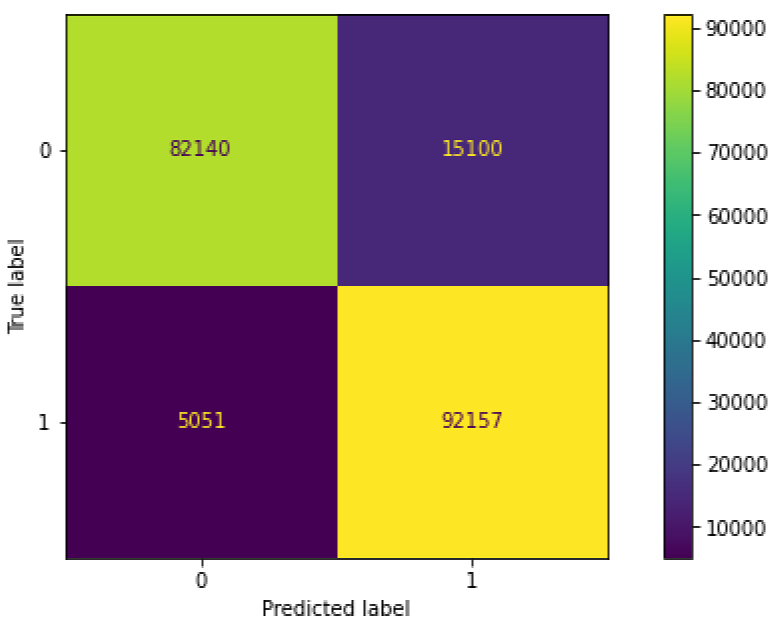
In [142…

```
y_pred_test = rf.predict(x_test)
print("classification report test:")
print(classification_report(y_test,y_pred_test))
```

```
classification report test:
              precision    recall  f1-score   support

           0       0.94      0.84      0.89     97240
           1       0.86      0.95      0.90     97208

    accuracy                           0.90    194448
   macro avg       0.90      0.90      0.90    194448
weighted avg       0.90      0.90      0.90    194448
```

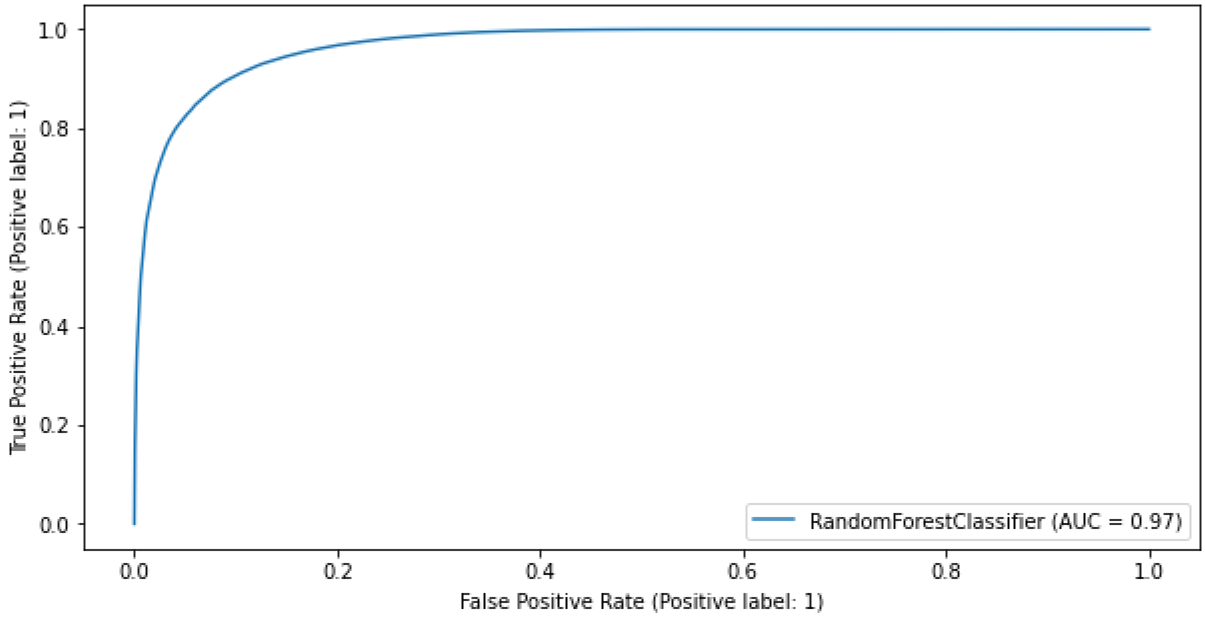
In [143…

```
plot_confusion_matrix(rf,x_test,y_test)
plt.show()
```



In [144…

```
from sklearn.metrics import plot_roc_curve
plot_roc_curve(rf,x_test,y_test)
plt.show()
```



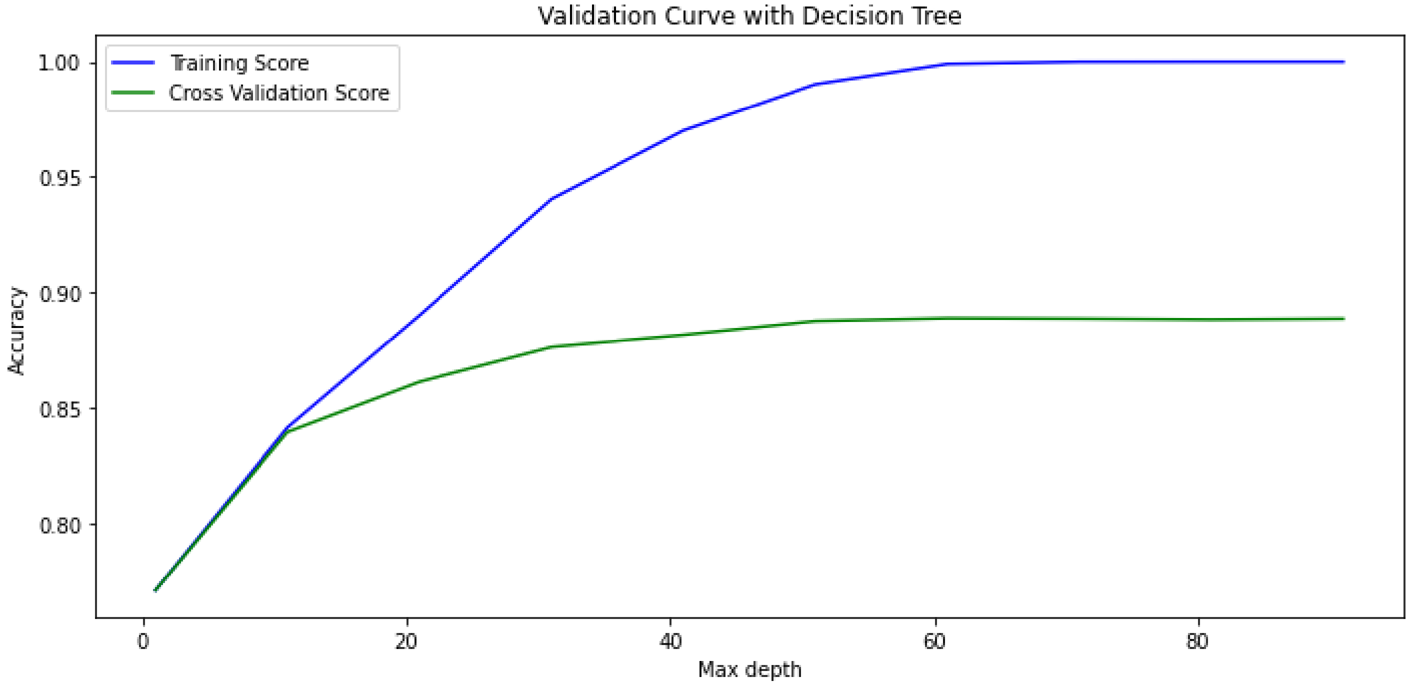
Hyperparameter tuning decision tree

In [145…

```
# from sklearn.model_selection import GridSearchCV
# param_grid = {"criterion":["gini","entropy"],
#                "max_depth": np.arange(1,10,1)}
# gs = GridSearchCV(estimator = dt,param_grid = param_grid,cv = 10)
# gs.fit(x_train,y_train)
# gs.best_params_
```

In [146…

```
parameter_range = np.arange(1,100,10)
train_score, test_score = validation_curve(DecisionTreeClassifier(), x_train, y_train,
                                                                        param_name = "max_
                                                                        param_range = par
                                                                                cv = 10,

# Calculating mean and standard deviation of training score
mean_train_score = np.mean(train_score, axis = 1)
std_train_score = np.std(train_score, axis = 1)

# Calculating mean and standard deviation of testing score
mean_test_score = np.mean(test_score, axis = 1)
std_test_score = np.std(test_score, axis = 1)

# Plot mean accuracy scores for training and testing scores
plt.plot(parameter_range, mean_train_score,
        label = "Training Score", color = 'b')
plt.plot(parameter_range, mean_test_score,
label = "Cross Validation Score", color = 'g')

# Creating the plot
plt.title("Validation Curve with Decision Tree")
plt.xlabel("Max depth")
plt.ylabel("Accuracy")
plt.tight_layout()
plt.legend(loc = 'best')
plt.show()
```



Decision Tree model performance after tuning

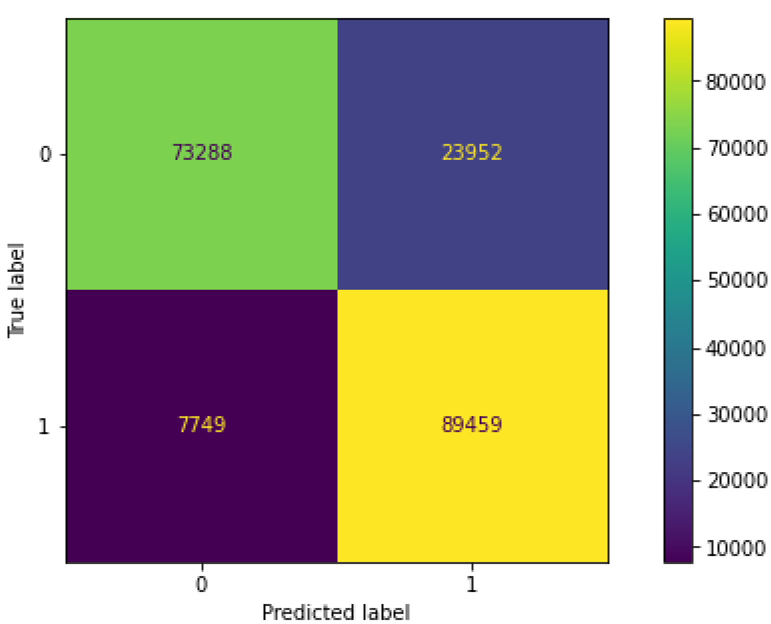
In [147…

```
dt = DecisionTreeClassifier(criterion = "gini",max_depth = 10 )
dt.fit(x_train,y_train)
y_pred_test = dt.predict(x_test)
print(classification_report(y_test,y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.90      0.75      0.82     97240
           1       0.79      0.92      0.85     97208

    accuracy                           0.84    194448
   macro avg       0.85      0.84      0.84    194448
weighted avg       0.85      0.84      0.84    194448
```

In [148…

```
plot_confusion_matrix(dt,x_test,y_test)
plt.show()
```



HyperParameter tuning Random Forest Classifier

In [149…

```
# from sklearn.ensemble import RandomForestClassifier
# from sklearn.model_selection import GridSearchCV
# from sklearn.metrics import classification_report,accuracy_score,plot_confusion_matrix
# x = final_df.drop(columns='Response_1')
# y = final_df["Response_1"]
# n = [50,100,150,200]
# x_train,x_rem,y_train,y_rem = train_test_split(x_resampled,y_resampled,test_size = 0.3,
# x_val,x_test,y_val,y_test = train_test_split(x_rem,y_rem,test_size = 0.5)
# for i in n:
#   rf = RandomForestClassifier(n_estimators = i)
#   rf.fit(x_train,y_train)
#   y_pred = rf.predict(x_val)
#   print("classification report when n_estimators = ",i)
#   print(classification_report(y_val,y_pred))
```

After experimenting with various levels of n_estimators we found out that 100 is the optimal one

Random forest classifier after hyperparameter tuning

In [150…

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators = 100)
rf.fit(x_train,y_train)
y_pred_train = rf.predict(x_train)
print("classification report train:")
print(classification_report(y_train,y_pred_train))
```

```
classification report train:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    226839
           1       1.00      1.00      1.00    226871

    accuracy                           1.00    453710
   macro avg       1.00      1.00      1.00    453710
weighted avg       1.00      1.00      1.00    453710
```

In [151…

```
rf = RandomForestClassifier(n_estimators = 100)
rf.fit(x_train,y_train)
y_pred_test = rf.predict(x_test)
print("classification report train:")
print(classification_report(y_test,y_pred_test))
```

```
classification report train:
              precision    recall  f1-score   support

           0       0.94      0.84      0.89     97240
           1       0.86      0.95      0.90     97208

    accuracy                           0.90    194448
   macro avg       0.90      0.90      0.90    194448
weighted avg       0.90      0.90      0.90    194448
```

Gradient Boosting

In [152…

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier(n_estimators = 100)
gb.fit(x_train,y_train)
y_pred_train = gb.predict(x_train)
print("classification report train:")
print(classification_report(y_train,y_pred_train))
```

```
classification report train:
              precision    recall  f1-score   support

           0       0.94      0.69      0.80    226839
           1       0.76      0.95      0.84    226871

    accuracy                           0.82    453710
   macro avg       0.85      0.82      0.82    453710
weighted avg       0.85      0.82      0.82    453710
```

In [153…

```
gb = GradientBoostingClassifier(n_estimators = 100)
gb.fit(x_train,y_train)
y_pred_test = gb.predict(x_test)
print("classification report train:")
print(classification_report(y_test,y_pred_test))
```

```
classification report train:
              precision    recall  f1-score   support

           0       0.94      0.69      0.79     97240
           1       0.75      0.95      0.84     97208

    accuracy                           0.82    194448
   macro avg       0.85      0.82      0.82    194448
weighted avg       0.85      0.82      0.82    194448
```

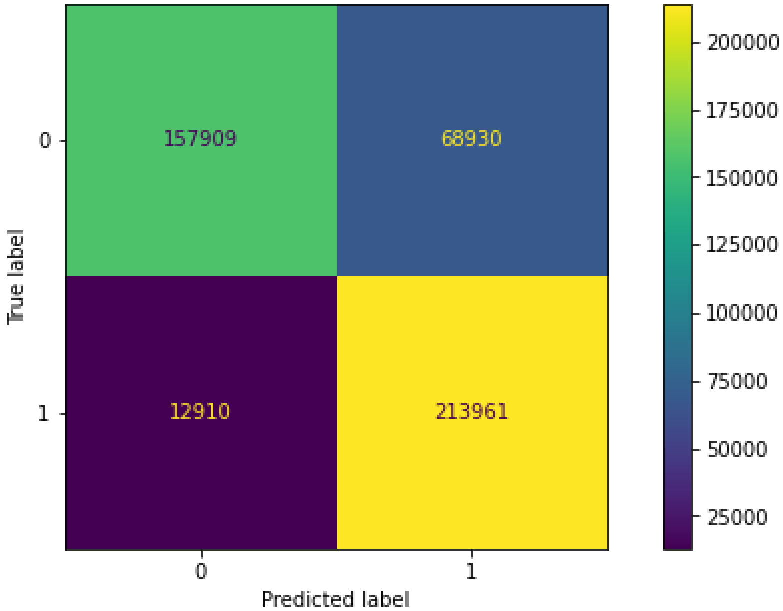
Out of all the models experimented random forest classifier gives the best performance

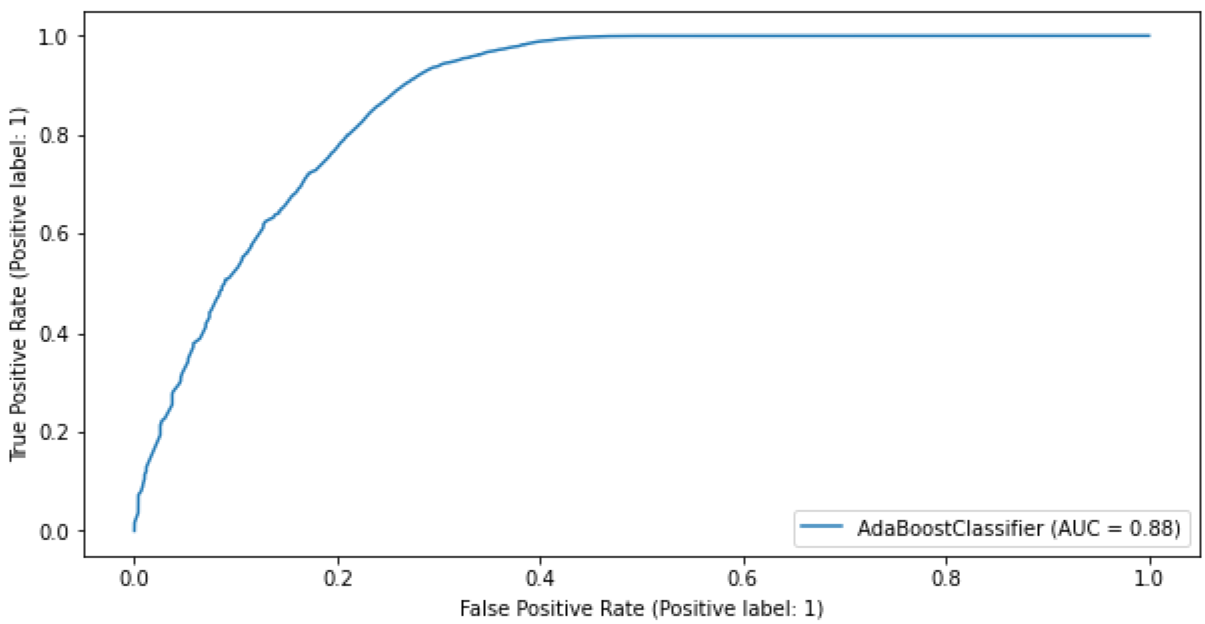
Ada Boost and XGBClassifier Base Model

In [154…

```
from sklearn.ensemble import AdaBoostClassifier
ada_model= AdaBoostClassifier(n_estimators=100,random_state = 40)
ada_model.fit(x_train,y_train)
y_pred_test = ada_model.predict(x_test)
print(classification_report(y_test, y_pred_test))
plot_confusion_matrix(ada_model,x_train,y_train)
plot_roc_curve(ada_model,x_train,y_train)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.92      0.69      0.79     97240
           1       0.76      0.94      0.84     97208

    accuracy                           0.82    194448
   macro avg       0.84      0.82      0.82    194448
weighted avg       0.84      0.82      0.82    194448
```

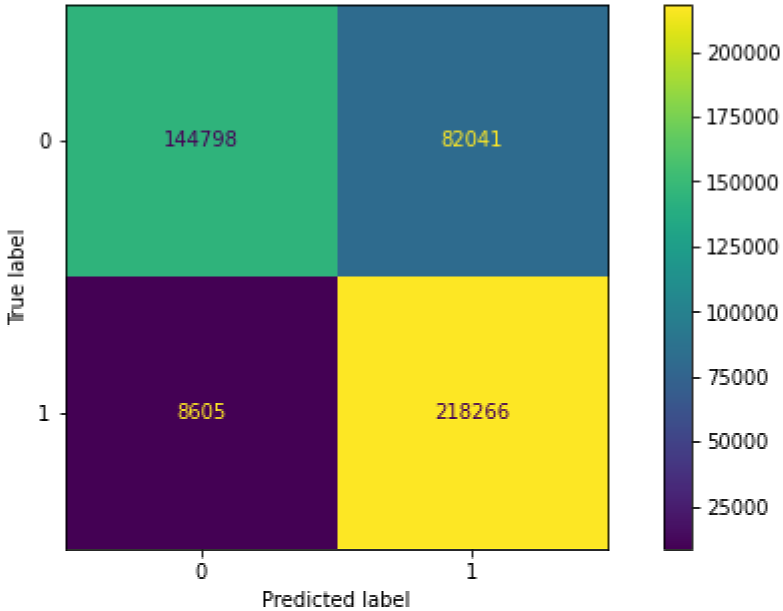


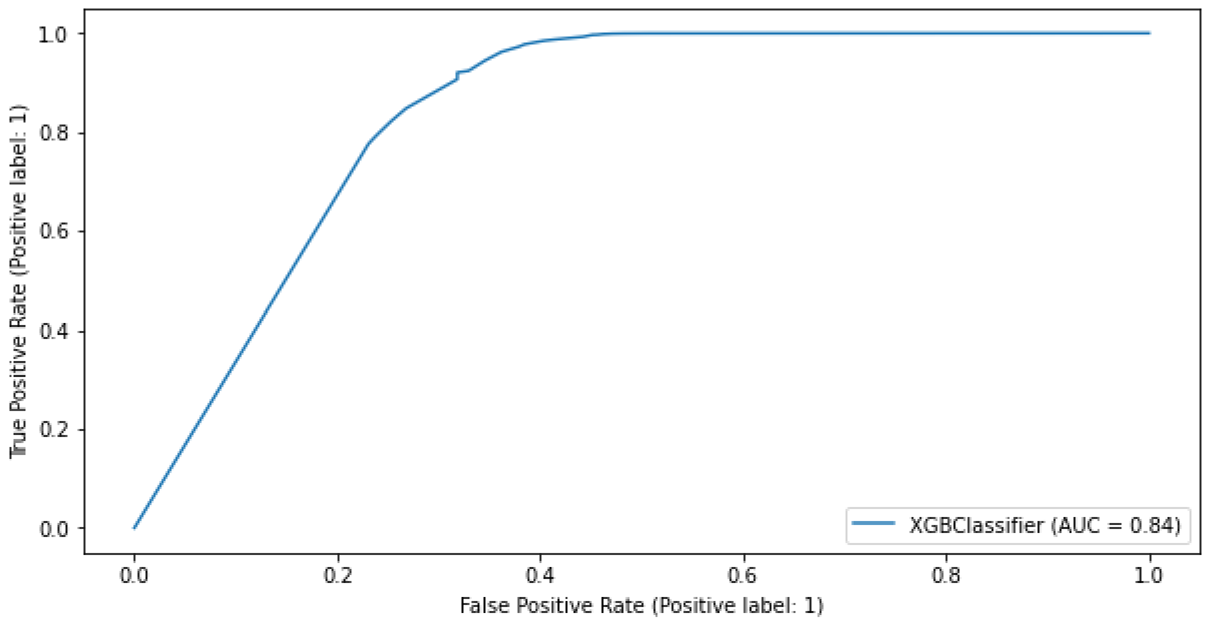


In [155…

```
from xgboost import XGBClassifier
xgb_model= XGBClassifier(learning_rate = 0.01, gamma = 2)
xgb_model.fit(x_train,y_train)
y_pred_test = xgb_model.predict(x_test)
print(classification_report(y_test, y_pred_test))
plot_confusion_matrix(xgb_model,x_train,y_train)
plot_roc_curve(xgb_model,x_train,y_train)
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.94      0.64      0.76     97240
           1       0.72      0.96      0.83     97208

    accuracy                           0.80    194448
   macro avg       0.83      0.80      0.79    194448
weighted avg       0.83      0.80      0.79    194448
```





Hyper Parameter Tuning for Ada Boosting and XGBClassifier

In [156…

```
# tuned_paramaters = [{'n_estimators': [50, 100, 150, 200],
#                       'learning_rate': [0.1, 0.01, 0.001, 0.15, 0.015]}]
```

In [157…

```
# ada_grid = GridSearchCV(estimator = ada_model,
#                         param_grid = tuned_paramaters,
#                         cv = 5,
#                         n_jobs=-1)
# ada_grid.fit(x_train, y_train)
# # get the best parameters
# print('Best parameters for AdaBoost Classifier: ', ada_grid.best_params_, '\n')
```

In [158…

```
# tuned_paramaters = [{'n_estimators': [50, 100, 150, 200],
#                       'learning_rate': [0.1, 0.01, 0.001, 0.15, 0.015],
#                       }]
```

In [159…

```
# gb_grid = GridSearchCV(estimator = xgb_model,
#                         param_grid = tuned_paramaters,
#                         cv = 5,
```

```
#                            n_jobs=-1)

# # fit the model on X_train and y_train using fit()
# gb_grid.fit(x_train, y_train)

# # get the best parameters
# print('Best parameters for Gradient Boositng Classifier: ', gb_grid.best_params_, '\n')
```

In [160…

```
# Optuna and Early Stop
from sklearn.model_selection import cross_val_score
def objective(trial):
    criterion = trial.suggest_categorical("criterion", ["gini", "entropy"])
    max_depth = trial.suggest_int("max_depth", 2, 32, log=True)
    n_estimators = trial.suggest_int("n_estimators", 100,500)

    rf = RandomForestClassifier(criterion =criterion,
            max_depth=max_depth,
            n_estimators=n_estimators
        )

    score = cross_val_score(rf, x, y, n_jobs=-1, cv=5)
    accuracy = score.mean()
    return accuracy


study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=15)
```

```
[I 2022-02-23 06:08:15,368] A new study created in memory with name: no-name-641db996-db54
-4b84-abc8-8900cdad69db
[I 2022-02-23 06:10:22,137] Trial 0 finished with value: 0.8783818945656593 and parameter
s: {'criterion': 'gini', 'max_depth': 3, 'n_estimators': 277}. Best is trial 0 with value:
0.8783818945656593.
[I 2022-02-23 06:14:02,486] Trial 1 finished with value: 0.8783791841712969 and parameter
s: {'criterion': 'gini', 'max_depth': 9, 'n_estimators': 213}. Best is trial 0 with value:
0.8783818945656593.
[I 2022-02-23 06:23:08,645] Trial 2 finished with value: 0.8784198400867327 and parameter
s: {'criterion': 'gini', 'max_depth': 11, 'n_estimators': 457}. Best is trial 2 with valu
e: 0.8784198400867327.
[I 2022-02-23 06:24:28,544] Trial 3 finished with value: 0.8783818945656593 and parameter
s: {'criterion': 'gini', 'max_depth': 3, 'n_estimators': 162}. Best is trial 2 with value:
0.8784198400867327.
[I 2022-02-23 06:26:04,189] Trial 4 finished with value: 0.8783818945656593 and parameter
s: {'criterion': 'gini', 'max_depth': 2, 'n_estimators': 259}. Best is trial 2 with value:
0.8784198400867327.
[I 2022-02-23 06:28:17,280] Trial 5 finished with value: 0.8784496544247187 and parameter
s: {'criterion': 'gini', 'max_depth': 12, 'n_estimators': 103}. Best is trial 5 with valu
e: 0.8784496544247187.
[I 2022-02-23 06:32:59,735] Trial 6 finished with value: 0.8783818945656593 and parameter
s: {'criterion': 'entropy', 'max_depth': 4, 'n_estimators': 453}. Best is trial 5 with val
ue: 0.8784496544247187.
[I 2022-02-23 06:36:56,569] Trial 7 finished with value: 0.876538826399241 and parameters:
{'criterion': 'entropy', 'max_depth': 23, 'n_estimators': 118}. Best is trial 5 with valu
e: 0.8784496544247187.
[I 2022-02-23 06:42:31,713] Trial 8 finished with value: 0.8783818945656593 and parameter
s: {'criterion': 'entropy', 'max_depth': 8, 'n_estimators': 311}. Best is trial 5 with val
ue: 0.8784496544247187.
[I 2022-02-23 06:47:14,826] Trial 9 finished with value: 0.8784144192980078 and parameter
s: {'criterion': 'gini', 'max_depth': 15, 'n_estimators': 190}. Best is trial 5 with valu
e: 0.8784496544247187.
[I 2022-02-23 06:58:27,783] Trial 10 finished with value: 0.8708768125762297 and parameter
s: {'criterion': 'gini', 'max_depth': 29, 'n_estimators': 356}. Best is trial 5 with valu
e: 0.8784496544247187.
[I 2022-02-23 07:09:59,090] Trial 11 finished with value: 0.8784279712698198 and parameter
s: {'criterion': 'gini', 'max_depth': 13, 'n_estimators': 472}. Best is trial 5 with valu
e: 0.8784496544247187.
```

```
[I 2022-02-23 07:20:15,378] Trial 12 finished with value: 0.8782978723404256 and parameters: {'criterion': 'gini', 'max_depth': 16, 'n_estimators': 387}. Best is trial 5 with value: 0.8784496544247187.
[I 2022-02-23 07:26:13,696] Trial 13 finished with value: 0.8783818945656593 and parameters: {'criterion': 'gini', 'max_depth': 5, 'n_estimators': 496}. Best is trial 5 with value: 0.8784496544247187.
[I 2022-02-23 07:28:33,141] Trial 14 finished with value: 0.8784415232416316 and parameters: {'criterion': 'gini', 'max_depth': 14, 'n_estimators': 100}. Best is trial 5 with value: 0.8784496544247187.
```

In [161…

```python
trial = study.best_trial
print('Accuracy: {}'.format(trial.value))
print("Best hyperparameters: {}".format(trial.params))
```

```
Accuracy: 0.8784496544247187
Best hyperparameters: {'criterion': 'gini', 'max_depth': 12, 'n_estimators': 103}
```

In [162…

```python
print("Best params: ", study.best_params)
print("Best value: ", study.best_value)
print("Best Trial: ", study.best_trial)
print("Trials: ", study.trials)
```

```
Best params:  {'criterion': 'gini', 'max_depth': 12, 'n_estimators': 103}
Best value:  0.8784496544247187
Best Trial:  FrozenTrial(number=5, values=[0.8784496544247187], datetime_start=datetime.datetime(2022, 2, 23, 6, 26, 4, 192159), datetime_complete=datetime.datetime(2022, 2, 23, 6, 28, 17, 279898), params={'criterion': 'gini', 'max_depth': 12, 'n_estimators': 103}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=5, state=TrialState.COMPLETE, value=None)
Trials:  [FrozenTrial(number=0, values=[0.8783818945656593], datetime_start=datetime.datetime(2022, 2, 23, 6, 8, 15, 372584), datetime_complete=datetime.datetime(2022, 2, 23, 6, 10, 22, 137169), params={'criterion': 'gini', 'max_depth': 3, 'n_estimators': 277}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=0, state=TrialState.COMPLETE, value=None), FrozenTrial(number=1, values=[0.8783791841712969], datetime_start=datetime.datetime(2022, 2, 23, 6, 10, 22, 140279), datetime_complete=datetime.datetime(2022, 2, 23, 6, 14, 2, 486007), params={'criterion': 'gini', 'max_depth': 9, 'n_estimators': 213}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=1, state=TrialState.COMPLETE, value=None), FrozenTrial(number=2, values=[0.8784198400867327], datetime_start=datetime.datetime(2022, 2, 23, 6, 14, 2, 489050), datetime_complete=datetime.datetime(2022, 2, 23, 6, 23, 8, 645245), params={'criterion': 'gini', 'max_depth': 11, 'n_estimators': 457}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=2, state=TrialState.COMPLETE, value=None), FrozenTrial(number=3, values=[0.8783818945656593], datetime_start=datetime.datetime(2022, 2, 23, 6, 23, 8, 648008), datetime_complete=datetime.datetime(2022, 2, 23, 6, 24, 28, 543709), params={'criterion': 'gini', 'max_depth': 3, 'n_estimators': 162}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=3, state=TrialState.COMPLETE, value=None), FrozenTrial(number=4, values=[0.8783818945656593], datetime_start=datetime.datetime(2022, 2, 23, 6, 24, 28, 545946), datetime_complete=datetime.datetime(2022, 2, 23, 6, 26, 4, 188626), params={'criterion': 'gini', 'max_depth': 2, 'n_estimators': 259}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=4, state=TrialState.COMPLETE, value=None), FrozenTrial(number=5, values=[0.8784496544247187], datetime_start=datetime.datetime(2022, 2, 23, 6, 26, 4, 192159), datetime_complete=datetime.datetime(2022, 2, 23, 6, 28, 17, 279898), params={'criterion': 'gini', 'max_depth': 12, 'n_estimators': 103}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution
```

```
(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, tria
l_id=5, state=TrialState.COMPLETE, value=None), FrozenTrial(number=6, values=[0.8783818945
656593], datetime_start=datetime.datetime(2022, 2, 23, 6, 28, 17, 283632), datetime_comple
te=datetime.datetime(2022, 2, 23, 6, 32, 59, 735181), params={'criterion': 'entropy', 'max
_depth': 4, 'n_estimators': 453}, distributions={'criterion': CategoricalDistribution(choi
ces=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1),
'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_
attrs={}, intermediate_values={}, trial_id=6, state=TrialState.COMPLETE, value=None), Froz
enTrial(number=7, values=[0.876538826399241], datetime_start=datetime.datetime(2022, 2, 2
3, 6, 32, 59, 739426), datetime_complete=datetime.datetime(2022, 2, 23, 6, 36, 56, 56844
6), params={'criterion': 'entropy', 'max_depth': 23, 'n_estimators': 118}, distributions=
{'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUni
formDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500,
low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=7, sta
te=TrialState.COMPLETE, value=None), FrozenTrial(number=8, values=[0.8783818945656593], da
tetime_start=datetime.datetime(2022, 2, 23, 6, 36, 56, 578240), datetime_complete=datetim
e.datetime(2022, 2, 23, 6, 42, 31, 713536), params={'criterion': 'entropy', 'max_depth':
8, 'n_estimators': 311}, distributions={'criterion': CategoricalDistribution(choices=('gin
i', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estima
tors': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={},
intermediate_values={}, trial_id=8, state=TrialState.COMPLETE, value=None), FrozenTrial(nu
mber=9, values=[0.8784144192980078], datetime_start=datetime.datetime(2022, 2, 23, 6, 42,
31, 718988), datetime_complete=datetime.datetime(2022, 2, 23, 6, 47, 14, 825559), params=
{'criterion': 'gini', 'max_depth': 15, 'n_estimators': 190}, distributions={'criterion': C
ategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistributio
n(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=
1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=9, state=TrialState.
COMPLETE, value=None), FrozenTrial(number=10, values=[0.8708768125762297], datetime_start=
datetime.datetime(2022, 2, 23, 6, 47, 14, 832578), datetime_complete=datetime.datetime(202
2, 2, 23, 6, 58, 27, 782328), params={'criterion': 'gini', 'max_depth': 29, 'n_estimator
s': 356}, distributions={'criterion': CategoricalDistribution(choices=('gini', 'entrop
y')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntU
niformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermedia
te_values={}, trial_id=10, state=TrialState.COMPLETE, value=None), FrozenTrial(number=11,
values=[0.8784279712698198], datetime_start=datetime.datetime(2022, 2, 23, 6, 58, 27, 7937
34), datetime_complete=datetime.datetime(2022, 2, 23, 7, 9, 59, 89941), params={'criterio
n': 'gini', 'max_depth': 13, 'n_estimators': 472}, distributions={'criterion': Categorical
Distribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32,
low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_a
ttrs={}, system_attrs={}, intermediate_values={}, trial_id=11, state=TrialState.COMPLETE,
value=None), FrozenTrial(number=12, values=[0.8782978723404256], datetime_start=datetime.d
atetime(2022, 2, 23, 7, 9, 59, 94002), datetime_complete=datetime.datetime(2022, 2, 23, 7,
20, 15, 378558), params={'criterion': 'gini', 'max_depth': 16, 'n_estimators': 387}, distr
ibutions={'criterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth':
IntLogUniformDistribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution
(high=500, low=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, tria
l_id=12, state=TrialState.COMPLETE, value=None), FrozenTrial(number=13, values=[0.87838189
45656593], datetime_start=datetime.datetime(2022, 2, 23, 7, 20, 15, 381027), datetime_comp
lete=datetime.datetime(2022, 2, 23, 7, 26, 13, 695923), params={'criterion': 'gini', 'max_
depth': 5, 'n_estimators': 496}, distributions={'criterion': CategoricalDistribution(choic
es=('gini', 'entropy')), 'max_depth': IntLogUniformDistribution(high=32, low=2, step=1),
'n_estimators': IntUniformDistribution(high=500, low=100, step=1)}, user_attrs={}, system_
attrs={}, intermediate_values={}, trial_id=13, state=TrialState.COMPLETE, value=None), Fro
zenTrial(number=14, values=[0.8784415232416316], datetime_start=datetime.datetime(2022, 2,
23, 7, 26, 13, 700920), datetime_complete=datetime.datetime(2022, 2, 23, 7, 28, 33, 14070
2), params={'criterion': 'gini', 'max_depth': 14, 'n_estimators': 100}, distributions={'cr
iterion': CategoricalDistribution(choices=('gini', 'entropy')), 'max_depth': IntLogUniform
Distribution(high=32, low=2, step=1), 'n_estimators': IntUniformDistribution(high=500, low
=100, step=1)}, user_attrs={}, system_attrs={}, intermediate_values={}, trial_id=14, state
=TrialState.COMPLETE, value=None)]
```

PLOTTING THE STUDY CREATED BY OPTUNA

In [163…

```
optuna.visualization.plot_optimization_history(study)
```

In [164…

```
optuna.visualization.plot_slice(study)
```

In [171…

```
rf = RandomForestClassifier(criterion='entropy', max_depth= 15,n_estimators= 394)
rf.fit(x_train,y_train)
y_pred_train = rf.predict(x_train)
print("classification report train:")
print(classification_report(y_train,y_pred_train))
```

```
classification report train:
              precision    recall  f1-score   support

           0       0.96      0.70      0.81    226839
           1       0.76      0.97      0.86    226871

    accuracy                           0.84    453710
   macro avg       0.86      0.84      0.83    453710
weighted avg       0.86      0.84      0.83    453710
```

In [ ]:

```
y_pred_test = rf.predict(x_test)
print("classification report train:")
print(classification_report(y_test,y_pred_test))
```

In [ ]:

```
plot_confusion_matrix(rf,x_test,y_test)
plt.show()
```

In [ ]:

```
# from sklearn.model_selection import cross_val_score
# def objective(trial):
#     criterion = trial.suggest_categorical("criterion", ["gini", "entropy"])
#     max_depth = trial.suggest_int("max_depth", 2, 32, log=True)
#     #n_estimators = trial.suggest_int("n_estimators", 100,500)

#     rf = DecisionTreeClassifier(criterion =criterion,
#             max_depth=max_depth)

#     score = cross_val_score(rf, x, y, n_jobs=-1, cv=5)
#     accuracy = score.mean()
#     return accuracy


# study = optuna.create_study(direction="maximize")
# study.optimize(objective, n_trials=15)
```

In [ ]:

```
# dt = DecisionTreeClassifier(criterion = "gini",max_depth = 10 )
# dt.fit(x_train,y_train)
# y_pred_test = dt.predict(x_test)
# print(classification_report(y_test,y_pred_test))
```

In [172…

```
# plot_confusion_matrix(dt,x_test,y_test)
# plt.show()
```

In [176…

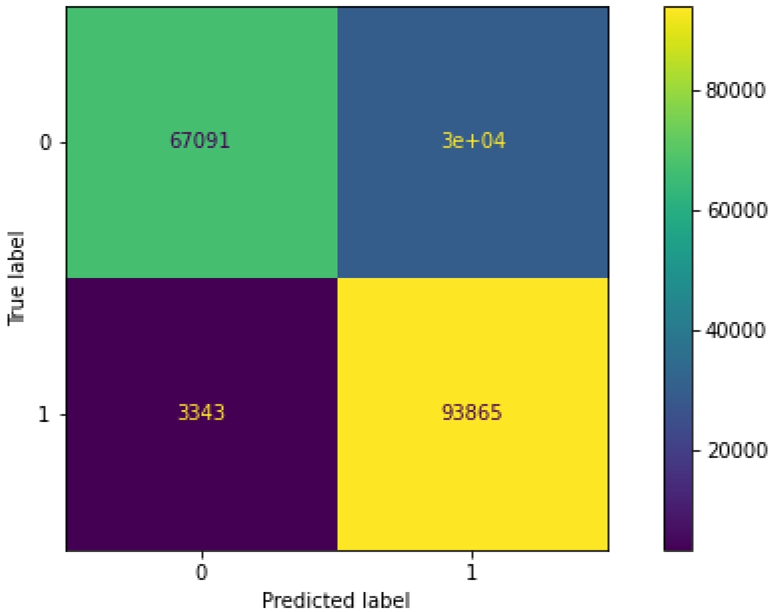
```
y_pred_test = rf.predict(x_test)
print("classification report train:")
print(classification_report(y_test,y_pred_test))
```

```
classification report train:
              precision    recall  f1-score   support

           0       0.95      0.69      0.80     97240
```

```
           1       0.76      0.97      0.85     97208

    accuracy                           0.83    194448
   macro avg       0.85      0.83      0.82    194448
weighted avg       0.85      0.83      0.82    194448
```

In [177…

```
plot_confusion_matrix(rf,x_test,y_test)
plt.show()
```



Feature Importance

In [175…

```
(pd.Series(rf.feature_importances_, index=x_train.columns)
   .nlargest(8)
   .plot(kind='barh'))
```

Out[175…

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6aafdd1250>
```