Health Insurance Cross Sell (HICS)

Roshan Jha

Abstract

An Insurance company that provide Health Insurance to its customers, usually they offer other insurance product to the customers through different kind of marketing channel. In this case we will build a model to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the company. I will try to build a model to predict whether a customer would be interested in Vehicle Insurance for that, I will follow the steps of model building in data science like, data cleaning, EDA, feature selection and the final model building.

Motivation

Our client is an Insurance company that has provided Health Insurance to its customers now they need your help in building a model to predict whether the policyholders (customers) from past year will also be interested in Vehicle Insurance provided by the company. Along with that I will also try to find out answers to what's the major factor that make a health insurance customer not interested with vehicle insurance and much more.

Dataset

Health Insurance Cross Sell (HICS)

With approx. 381K records in a csv file, Health Insurance Cross Sell dataset is around 22 MB in size and have 25 columns like, id, gender, age, region code, vehicle damage, etc.

https://www.kaggle.com/anmolkumar/health-insurance-cross-sell-prediction

License: GPL 2

Data Preparation and Cleaning

1. Recategorizing
2. Binning

Research Question(s)

1. Will a Health Insurance customer be open to buying Vehicle Insurance as well?
2. What's the major factor that make a health insurance customer not interested with vehicle insurance?
3. How does age of a vehicle determining the response of vehicle insurance advertisement?
4. Which customer Generation that’s most likely to be interested in Vehicle Insurance?

Methods

1. Explore distributions of numerical and categorical features and their relationships with the target feature, Response (whether the customer responded to an offer about buying vehicle insurance)
2. Preprocess data in order to model it (look at missing values/outliers/skewed distributions/standardization)
3. Test out different models by tuning their hyperparameters and comparing their performance
4. Explore which features were the most impactful
5. Explore potential interactions between features

Methods (1)

1. Data Cleaning
    a. Recategorize Data
    b. Binning

2. Exploratory Data Analysis to Answer business Question

3. Feature Engineering & Selection For Machine Learning Process
    a. Encoding all the categorical features
    b. Checking correlation between dependent and independent variable
    c. Feature Selection

Methods (2)

4. Model Building :

    a. Splitting data into Training and Testing
    b. Applying the **SMOTE** (Synthetic Minority Oversampling Technique) to the minority target since the data is imbalance
    c. Creating base model of classification algorithm (Logistic Regression, KNN Classifier, Decision Tree Classifier, Random Forest Classifier)
    d. Check The Evaluation matrix for all the base model
    e. HyperParameter tuning
    f. Checking Evaluation Matrix for tuned Model
    g. Choose which model has the best recall score for this case

Findings(1)

From this dataset of health insurance customers almost **95%** of customers have a vehicle age that's less than 2 years. from our analysis, customers who has more than 2 years of vehicle age are more interested with vehicle insurance advertisement, while customers who has less than one year of vehicle age, only **4%** of them are actually interested with vehicle insurance.

We found out that customer who already have vehicle insurance are almost have no interest in another vehicle insurance. Our analysis shows that **99.9% of customers that have a vehicle insurance is not interested in another vehicle insurance**, while customer who doesn't have a vehicle insurance **22.5 %** of them are interested with vehicle insurance

Findings(2)

We also found out that a newer vehicle are more likely to have a vehicle insurance, with vehicle that's less than one year **66% of those are insured** , vehicle that's older than one year but less than 2 years are **33% insured**, while less than **one percent** of vehicle that's older than 2 years are insured. This should explain why customer who owns a newer vehicle are less likely to be interested with insurance promotion, because they probably already have one.

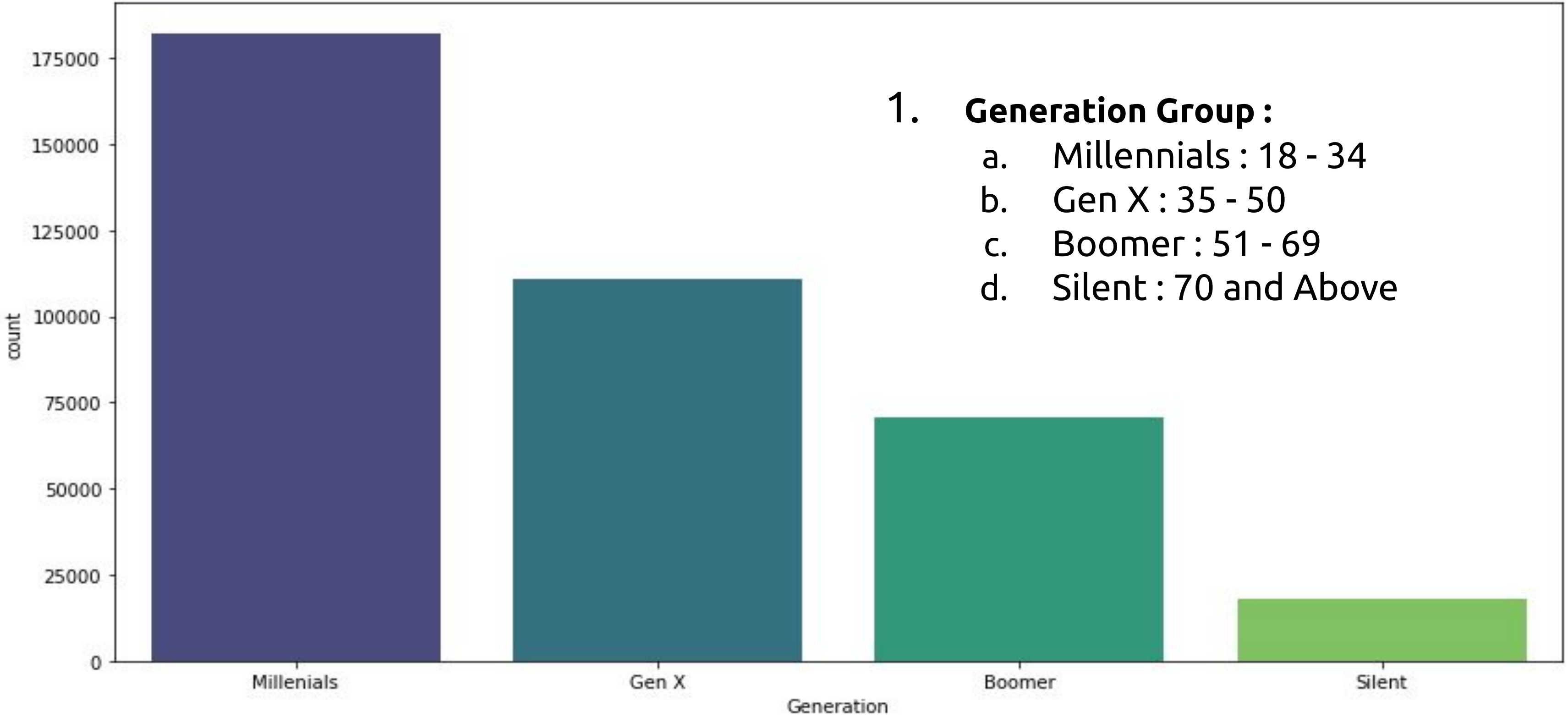
Customers who never had vehicle damaged only 0.5 % of those customers are interested with vehicle insurance, **87%** of customers who never had any vehicle damaged already have a vehicle insurance.

Findings(3)

1. **Which Customer Generation are less likely to be interested with vehicle insurance** the answer is **Millennials** (people in age group of 18 - 34) only **6% of millennials are actually interested with vehicle insurance**, and why is so?
   a. almost **63% of millennials already have vehicle insurance**, from our analysis before owning vehicle insurance is a major factor why someone is not interested with another vehicle insurance
   b. **90% of millennials have a vehicle that's less than one year of age**, and from our analysis before that vehicle that's less than one year are **66% already insured**

This concluded that millenials are more likely to have a vehicle insurance before our vehicle insurance team approached, and that's a major factor why millenials are least likely to be interested with our vehicle insurance, because they already have one.

Findings(3)



1. **Generation Group :**
   a. Millennials : 18 - 34
   b. Gen X : 35 - 50
   c. Boomer : 51 - 69
   d. Silent : 70 and Above

Findings(4)

So who's actually interested with our vehicle insurance ?

From the responses there are **12 % of our health insurance customers are interested with the vehicle insurance product** but who are those people?

1) **Customers who have had a vehicle damaged in the past** from our analysis we found out that **97%** customers who actually interested with vehicle insurance have had their vehicle damaged in the past

    a) **95 % of customers who have had vehicle damage in the past still doesn't have a vehicle insurance**

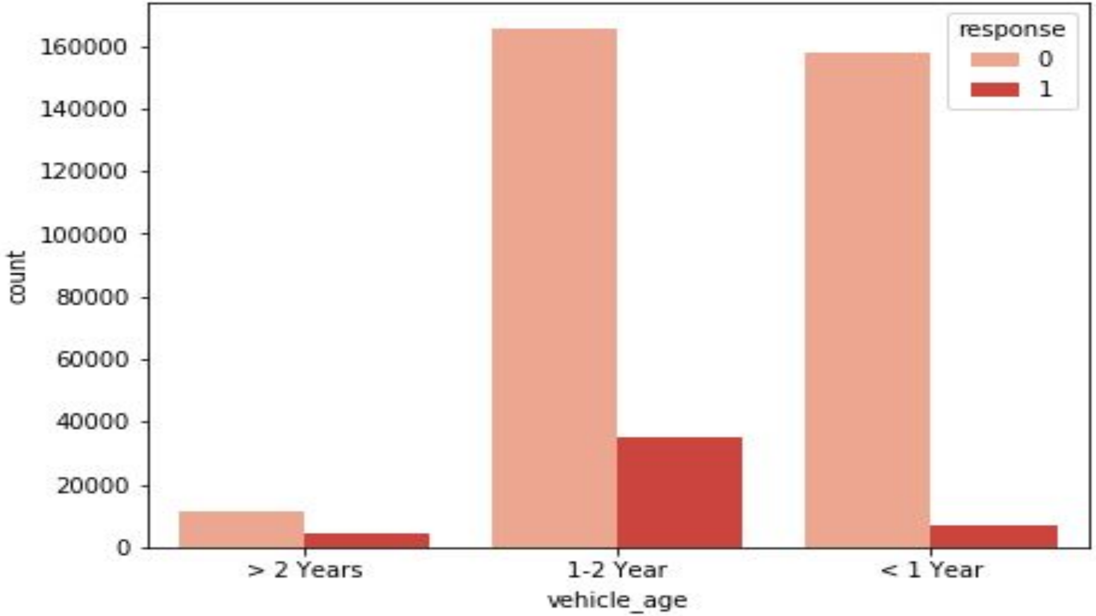
Findings(5)

2. **First, Customer who does not have a a vehicle insurance**, out of all customers who does not have a vehicle insurance **22.5 %** of them says that they're interested with vehicle insurance product.

3. **Customers who has vehicle that's older than 2 years** our analysis before mentioned that only less than **one percent** of car that's older than 2 years are previously insured, by not having a vehicle insurance they're more likely to be interested with our vehicle insurance, our data show that customer who has car that's more than 2 years are **7 times** more likely to be interested with vehicle insurance compared to customer who own a vehicle less than one year.

Findings(5)

From the vehicle age group customer who has a newer vehicle are less likely to be interested with vehicle insurance customer who has a vehicle that's older than 2 years are more likely to be interested to vehicle insurance.



Findings(6)

Which Customer Generation that's most likely to be interested in Vehicle insurance ?

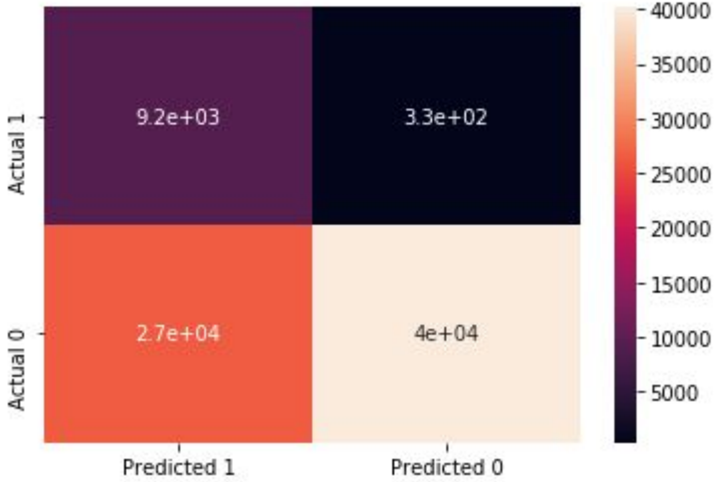
1. **GEN X** (Age Gen X : 35 - 50)
    a. Our analysis shows that GEN X has the highest percentage to be interested with vehicle insurance, to be precise, **21 %** of GEN X are interested with vehicle insurance.
    b. This might be because **72% GEN X** does not have a vehicle insurance, and GEN X has the highest percentage of vehicle damager the past **(67%)** among other generation

Conclusions

Machine learning could predict whether a customer would be interested or not towards vehicle insurance product with recall 0.965 out of 1

Using logistic regression that has been tuned, we focus more on recall instead of accuracy here because of we want to reduce the false negative.

	LogisticReg	KNN	DecisionTree	RandomForest
Accuracy	0.647713	0.735116	0.581551	0.707171
Precision	0.257412	0.229278	0.229454	0.287135
Recall	0.965452	0.474325	0.996220	0.906332
F1-Score	0.406454	0.309129	0.372997	0.436107



Acknowledgements

The data has been taken from Kaggle, which I already mentioned before. I am thankful to the website.

And yes , I used some other informal analysis to inform my work. Taking about feedback , unfortunately I don’t have no one to get feedback from now.

References

Python Libraries:

1. matplotlib
2. pandas
3. numpy
4. seaborn
5. sklearn

My Work can be found here:
https://github.com/roshan89/UCSanDiegoX-DSE200x-Final_Project

HICS-EDA

April 3, 2021

Python Libraries Import

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import sidetable as stb
```

Getting The Basic Understanding of the Data

[2]:
```python
df = pd.read_csv('train.csv')
df.head()
```

[2]:
```
   id  Gender  Age  Driving_License  Region_Code  Previously_Insured  \
0   1    Male   44                1         28.0                   0
1   2    Male   76                1          3.0                   0
2   3    Male   47                1         28.0                   0
3   4    Male   21                1         11.0                   1
4   5  Female   29                1         41.0                   1

  Vehicle_Age Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  \
0   > 2 Years            Yes         40454.0                  26.0      217
1    1-2 Year             No         33536.0                  26.0      183
2   > 2 Years            Yes         38294.0                  26.0       27
3    < 1 Year             No         28619.0                 152.0      203
4    < 1 Year             No         27496.0                 152.0       39

   Response
0         1
1         0
2         1
3         0
4         0
```

[3]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381109 entries, 0 to 381108
Data columns (total 12 columns):
```

```
 #   Column                Non-Null Count   Dtype  
---  ------                --------------   -----  
 0   id                    381109 non-null  int64  
 1   Gender                381109 non-null  object 
 2   Age                   381109 non-null  int64  
 3   Driving_License       381109 non-null  int64  
 4   Region_Code           381109 non-null  float64
 5   Previously_Insured    381109 non-null  int64  
 6   Vehicle_Age           381109 non-null  object 
 7   Vehicle_Damage        381109 non-null  object 
 8   Annual_Premium        381109 non-null  float64
 9   Policy_Sales_Channel  381109 non-null  float64
 10  Vintage               381109 non-null  int64  
 11  Response              381109 non-null  int64  
dtypes: float64(3), int64(6), object(3)
memory usage: 34.9+ MB
```

We can see from this info that there is no null value in every columns so we don't have to worry handling any null value

Getting all the unique value of the columns

[4]:
```python
for column in df.columns:
    print(f"{column} :")
    print(df[column].unique())
    print("")
```

```
id :
[     1      2      3 … 381107 381108 381109]

Gender :
['Male' 'Female']

Age :
[44 76 47 21 29 24 23 56 32 41 71 37 25 42 60 65 49 34 51 26 57 79 48 45
 72 30 54 27 38 22 78 20 39 62 58 59 63 50 67 77 28 69 52 31 33 43 36 53
 70 46 55 40 61 75 64 35 66 68 74 73 84 83 81 80 82 85]

Driving_License :
[1 0]

Region_Code :
[28.  3. 11. 41. 33.  6. 35. 50. 15. 45.  8. 36. 30. 26. 16. 47. 48. 19.
 39. 23. 37.  5. 17.  2.  7. 29. 46. 27. 25. 13. 18. 20. 49. 22. 44.  0.
  9. 31. 12. 34. 21. 10. 14. 38. 24. 40. 43. 32.  4. 51. 42.  1. 52.]

Previously_Insured :
[0 1]
```

```
Vehicle_Age :
['> 2 Years' '1-2 Year' '< 1 Year']

Vehicle_Damage :
['Yes' 'No']

Annual_Premium :
[ 40454.  33536.  38294. …  20706. 101664.  69845.]

Policy_Sales_Channel :
[ 26. 152. 160. 124.  14.  13.  30. 156. 163. 157. 122.  19.  22.  15.
 154.  16.  52. 155.  11. 151. 125.  25.  61.   1.  86.  31. 150.  23.
  60.  21. 121.   3. 139.  12.  29.  55.   7.  47. 127. 153.  78. 158.
  89.  32.   8.  10. 120.  65.   4.  42.  83. 136.  24.  18.  56.  48.
 106.  54.  93. 116.  91.  45.   9. 145. 147.  44. 109.  37. 140. 107.
 128. 131. 114. 118. 159. 119. 105. 135.  62. 138. 129.  88.  92. 111.
 113.  73.  36.  28.  35.  59.  53. 148. 133. 108.  64.  39.  94. 132.
  46.  81. 103.  90.  51.  27. 146.  63.  96.  40.  66. 100.  95. 123.
  98.  75.  69. 130. 134.  49.  97.  38.  17. 110.  80.  71. 117.  58.
  20.  76. 104.  87.  84. 137. 126.  68.  67. 101. 115.  57.  82.  79.
 112.  99.  70.   2.  34.  33.  74. 102. 149.  43.   6.  50. 144. 143.
  41.]

Vintage :
[217 183  27 203  39 176 249  72  28  80  46 289 221  15  58 147 256 299
 158 102 116 177 232  60 180  49  57 223 136 222 149 169  88 253 107 264
 233  45 184 251 153 186  71  34  83  12 246 141 216 130 282  73 171 283
 295 165  30 218  22  36  79  81 100  63 242 277  61 111 167  74 235 131
 243 248 114 281  62 189 139 138 209 254 291  68  92  52  78 156 247 275
  77 181 229 166  16  23  31 293 219  50 155  66 260  19 258 117 193 204
 212 144 234 206 228 125  29  18  84 230  54 123 101  86  13 237  85  98
  67 128  95  89  99 208 134 135 268 284 119 226 105 142 207 272 263  64
  40 245 163  24 265 202 259  91 106 190 162  33 194 287 292  69 239 132
 255 152 121 150 143 198 103 127 285 214 151 199  56  59 215 104 238 120
  21  32 270 211 200 197  11 213  93 113 178  10 290  94 231 296  47 122
 271 278 276  96 240 172 257 224 173 220 185  90  51 205  70 160 137 168
  87 118 288 126 241  82 227 115 164 236 286 244 108 274 201  97  25 174
 182 154  48  20  53  17 261  41 266  35 140 269 146 145  65 298 133 195
  55 188  75  38  43 110  37 129 170 109 267 279 112 280  76 191  26 161
 179 175 252  42 124 187 148 294  44 157 192 262 159 210 250  14 273 297
 225 196]

Response :
[1 0]
```

Storing all the information in a single table just to keep it neat

[5]:
```python
desc       = ["Unique ID for the customer.",
              "Gender of the customer.",
              "Age of the customer.",
              "0: Doesn't have DL, 1: have DL.",
              "Unique code for the region of the customer.",
              "1 : Customer already has Vehicle insurance, 0 : Customer doesn't␣
↪have Vehicle insurance.",
              "Age of the Vehicle.",
              "1 : Customer got his/her vehicle damaged in the past. 0 :␣
↪Customer didn't get his/her vehicle damaged in the past.",
              "The amount customer needs to pay as premium in the year.",
              "Anonymized Code for the channel of outreaching to the customer␣
↪ie. Different Agents, Over Mail, Over Phone, In Person, etc.",
              "Number of Days, Customer has been associated with the company.",
              "1 : Customer is interested, 0 : Customer is not interested."]

df_desc = []
j = 0
for column in df.columns:
  df_desc.append(
      [
       column,
       df[column].dtypes,
       df[column].isnull().sum(),
       round(df[column].isnull().sum()/len(df)*100, 2),
       df[column].nunique(),
       df[column].unique(),
       desc[j]
       ]
       )
  j += 1

column_desc = pd.DataFrame(df_desc, columns = ['Column', 'Dtype', 'Null', 'Null␣
↪(%)', 'nUnique', 'Unique','Description'])
column_desc
```

[5]:
```
                 Column    Dtype  Null  Null (%)  nUnique  \
0                    id    int64     0       0.0   381109
1                Gender   object     0       0.0        2
2                   Age    int64     0       0.0       66
3       Driving_License    int64     0       0.0        2
4           Region_Code  float64     0       0.0       53
5    Previously_Insured    int64     0       0.0        2
6           Vehicle_Age   object     0       0.0        3
7        Vehicle_Damage   object     0       0.0        2
8        Annual_Premium  float64     0       0.0    48838
```

```
9   Policy_Sales_Channel  float64     0       0.0       155
10               Vintage    int64     0       0.0       290
11              Response    int64     0       0.0         2

                                             Unique  \
0   [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14…
1                                    [Male, Female]
2   [44, 76, 47, 21, 29, 24, 23, 56, 32, 41, 71, 3…
3                                            [1, 0]
4   [28.0, 3.0, 11.0, 41.0, 33.0, 6.0, 35.0, 50.0,…
5                                            [0, 1]
6                  [> 2 Years, 1-2 Year, < 1 Year]
7                                         [Yes, No]
8   [40454.0, 33536.0, 38294.0, 28619.0, 27496.0, …
9   [26.0, 152.0, 160.0, 124.0, 14.0, 13.0, 30.0, …
10  [217, 183, 27, 203, 39, 176, 249, 72, 28, 80, …
11                                           [1, 0]

                                        Description
0                       Unique ID for the customer.
1                           Gender of the customer.
2                              Age of the customer.
3                   0: Doesn't have DL, 1: have DL.
4           Unique code for the region of the customer.
5   1 : Customer already has Vehicle insurance, 0 …
6                               Age of the Vehicle.
7   1 : Customer got his/her vehicle damaged in th…
8   The amount customer needs to pay as premium in…
9   Anonymized Code for the channel of outreaching…
10  Number of Days, Customer has been associated w…
11  1 : Customer is interested, 0 : Customer is no…
```

[6]:
```python
df.describe()
```

[6]:
```
                  id            Age  Driving_License    Region_Code  \
count  381109.000000  381109.000000    381109.000000  381109.000000
mean   190555.000000      38.822584         0.997869      26.388807
std    110016.836208      15.511611         0.046110      13.229888
min         1.000000      20.000000         0.000000       0.000000
25%     95278.000000      25.000000         1.000000      15.000000
50%    190555.000000      36.000000         1.000000      28.000000
75%    285832.000000      49.000000         1.000000      35.000000
max    381109.000000      85.000000         1.000000      52.000000

       Previously_Insured  Annual_Premium  Policy_Sales_Channel  \
count       381109.000000   381109.000000         381109.000000
mean             0.458210    30564.389581            112.034295
```

```
std              0.498251     17213.155057                 54.203995
min              0.000000      2630.000000                  1.000000
25%              0.000000     24405.000000                 29.000000
50%              0.000000     31669.000000                133.000000
75%              1.000000     39400.000000                152.000000
max              1.000000    540165.000000                163.000000

             Vintage       Response
count  381109.000000  381109.000000
mean      154.347397       0.122563
std        83.671304       0.327936
min        10.000000       0.000000
25%        82.000000       0.000000
50%       154.000000       0.000000
75%       227.000000       0.000000
max       299.000000       1.000000
```

1. The average customers vintage (numbers of day been insured in the compant is 154 days)

2. No customers in this data set have been with the insurance company for 1 full year

3. The oldest customers in this dataset is 85 while the median is 36

4. The most expensive annual premium is almost 17 times more expensive compared to the median annual premium

[7]:
```python
df.describe(include = 'O')
```

[7]:
```
        Gender Vehicle_Age Vehicle_Damage
count   381109      381109         381109
unique       2           3              2
top       Male    1-2 Year            Yes
freq    206089      200316         192413
```

1. There are more male than female in this dataset

2. Majority of the customer has a vehichle that's more than one year and less than two years

3. Majority of the customer in this dataset have had their vehicle damaged before

Cleaning the column name just because of my preference working with snake casing :P

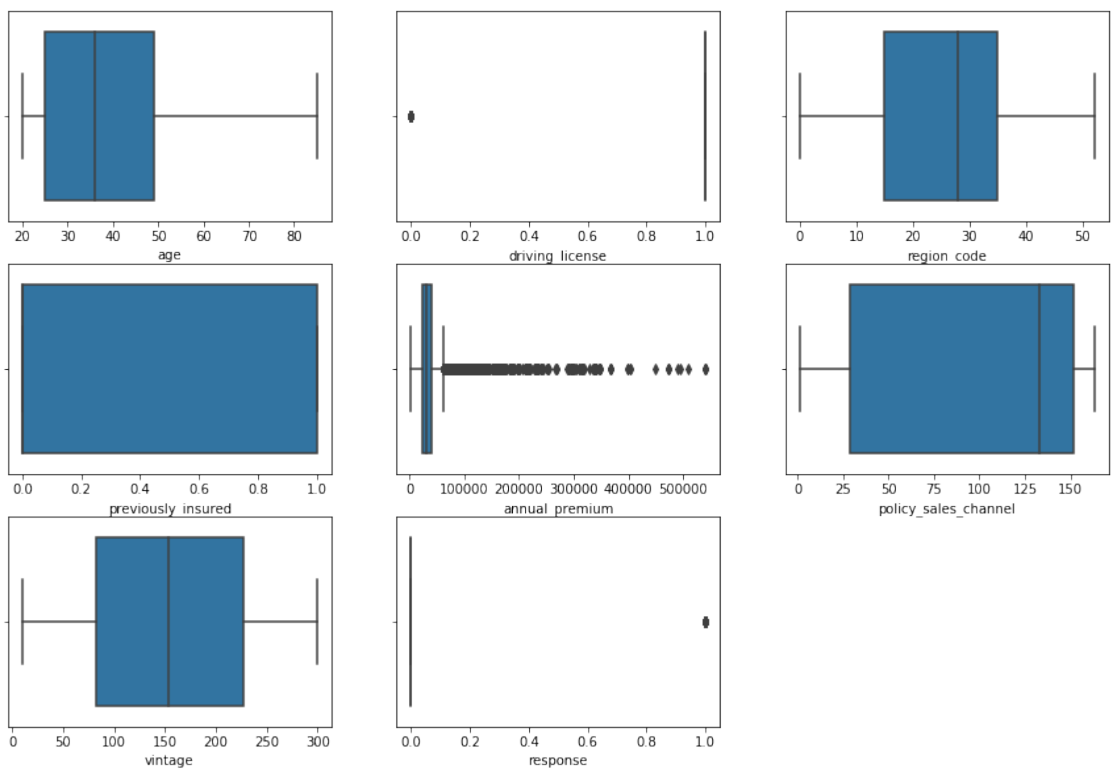
[8]:
```python
df.columns = df.columns.str.lower()
```

Checking and handling missing values and outliers

Since there is no missing values in the dataset we will skip those part and will go straight to checking and handling outliers

Checking Outliers with boxplot

[9]:
```
df_describe = df.drop(columns = ['id', 'vehicle_damage',␣
↪'gender','vehicle_age'])


plt.figure(figsize = (15, 10))
x = 1

for column in df_describe.describe():
    plt.subplot(3,3,x)
    sns.boxplot(df[column])
    x += 1
```



We can see that there's many outliers in this dataset in the annual premium columns

We will be handling it by making a bin for the annual premium

We will not removing the outliers of annual premium since it might hold valueable information related to response

Driving license because it's binary categorical there wouldn't be any outliers

Binning the annual premium into groups

[10]:
```
bin_premium_group = [2600, 25000, 50000, 100000, 200000, df['annual_premium'].
 ↪max()]
label_bin = ['Bronze', 'Silver', 'Gold', 'Platinum', 'Diamond']

df['premium_group'] = pd.cut(df['annual_premium'], bins = bin_premium_group,␣
 ↪labels = label_bin)
df.head()
```

[10]:
```
   id  gender  age  driving_license  region_code  previously_insured  \
0   1    Male   44                1         28.0                   0
1   2    Male   76                1          3.0                   0
2   3    Male   47                1         28.0                   0
3   4    Male   21                1         11.0                   1
4   5  Female   29                1         41.0                   1

  vehicle_age vehicle_damage  annual_premium  policy_sales_channel  vintage  \
0   > 2 Years            Yes         40454.0                  26.0      217
1    1-2 Year             No         33536.0                  26.0      183
2   > 2 Years            Yes         38294.0                  26.0       27
3    < 1 Year             No         28619.0                 152.0      203
4    < 1 Year             No         27496.0                 152.0       39

   response premium_group
0         1        Silver
1         0        Silver
2         1        Silver
3         0        Silver
4         0        Silver
```

Binning age into Age Generation Generation age based on (https://www.weforum.org/agenda/2015/09/how-different-age-groups-identify-with-their-generational-labels/)

[11]:
```
bin_age_generation = [18, 34, 50, 69, df['age'].max()]
label_age_generation = ['Millenials', 'Gen X', 'Boomer', 'Silent']

df['Generation'] = pd.cut(df['age'], bins = bin_age_generation, labels =␣
 ↪label_age_generation)
df.head()
```

[11]:
```
   id  gender  age  driving_license  region_code  previously_insured  \
0   1    Male   44                1         28.0                   0
1   2    Male   76                1          3.0                   0
```

```
2   3    Male   47                 1        28.0                    0
3   4    Male   21                 1        11.0                    1
4   5  Female   29                 1        41.0                    1

  vehicle_age vehicle_damage  annual_premium  policy_sales_channel  vintage  \
0   > 2 Years            Yes         40454.0                  26.0      217
1    1-2 Year             No         33536.0                  26.0      183
2   > 2 Years            Yes         38294.0                  26.0       27
3    < 1 Year             No         28619.0                 152.0      203
4    < 1 Year             No         27496.0                 152.0       39

   response premium_group  Generation
0         1        Silver       Gen X
1         0        Silver      Silent
2         1        Silver       Gen X
3         0        Silver  Millenials
4         0        Silver  Millenials
```

0.1 Exploratory Data Analysis

0.1.1 Univariate

[12]:
```python
df.stb.freq(['gender'], cum_cols = False)
```

[12]:

	gender	count	percent
0	Male	206089	54.07613
1	Female	175020	45.92387

Gender - 54 % of the customer of the health insurance is male

[13]:
```python
plt.figure(figsize = (13, 5))
sns.countplot(df['gender'], palette = 'RdBu')
plt.show()
```

[14]:
```python
df['age'].mean()
```

[14]: 38.822583565331705

The average age for health insurance customers is around 38 - 39

[15]:
```python
df.stb.freq(['driving_license'], cum_cols=False)
```

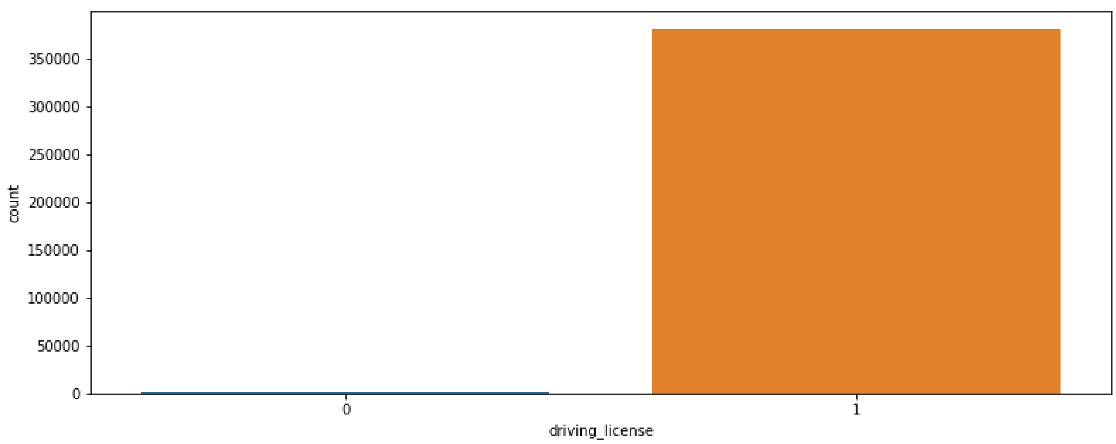
[15]:

	driving_license	count	percent
0	1	380297	99.786938
1	0	812	0.213062

Almost everyone in this dataset has a driving license

[16]:
```python
plt.figure(figsize = (13, 5))
sns.countplot(df['driving_license'])
```

[16]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede37f7a10>



[17]:
```python
df.stb.freq(['region_code'], cum_cols=False).head()
```

[17]:

	region_code	count	percent
0	28.0	106415	27.922458
1	8.0	33877	8.889058
2	46.0	19749	5.181982
3	41.0	18263	4.792067
4	15.0	13308	3.491914

Region code 28 has the highest number of health insurance customers

While region 52 has the lowest number of health insurance customers

[18]:
```
df.stb.freq(['previously_insured'], cum_cols=False).head()
```

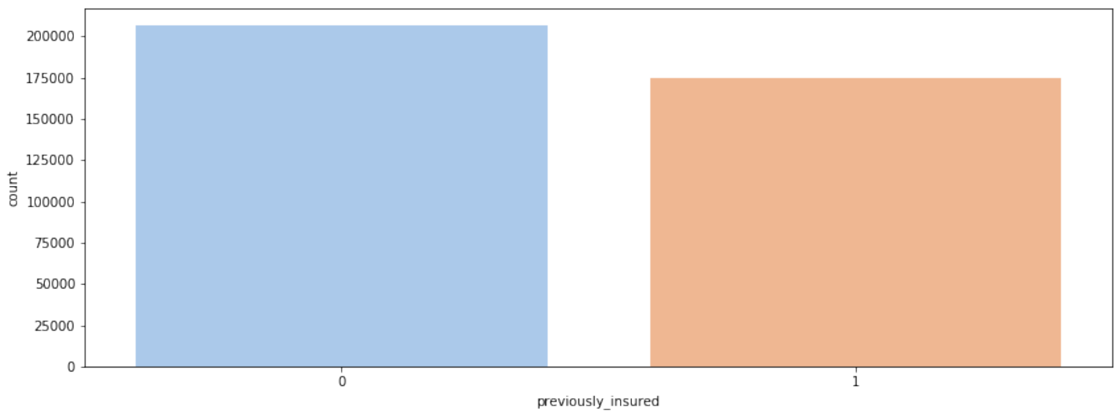
[18]:

	previously_insured	count	percent
0	0	206481	54.178988
1	1	174628	45.821012

More than half of the customers does not have a vehicle insurance

[19]:
```
plt.figure(figsize = (14, 5))
sns.countplot(df['previously_insured'], palette = 'pastel')
```

[19]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede374ecd0>



[20]:
```
df.stb.freq(['vehicle_age'], cum_cols=False)
```

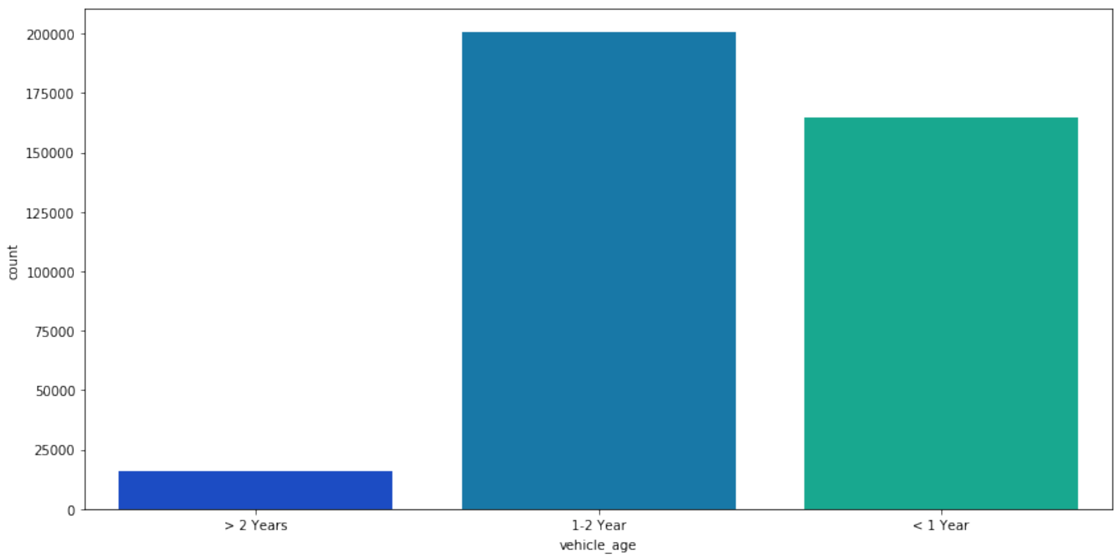
[20]:

	vehicle_age	count	percent
0	1-2 Year	200316	52.561341
1	< 1 Year	164786	43.238549
2	> 2 Years	16007	4.200111

95 % of health insurance customers have vehichle that's less than 2 years of age

[21]:
```
plt.figure(figsize = (14 ,7))
sns.countplot(df['vehicle_age'], palette = 'winter')
```

[21]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede375cf10>



[22]:
```
df.stb.freq(['vehicle_damage'], cum_cols=False)
```

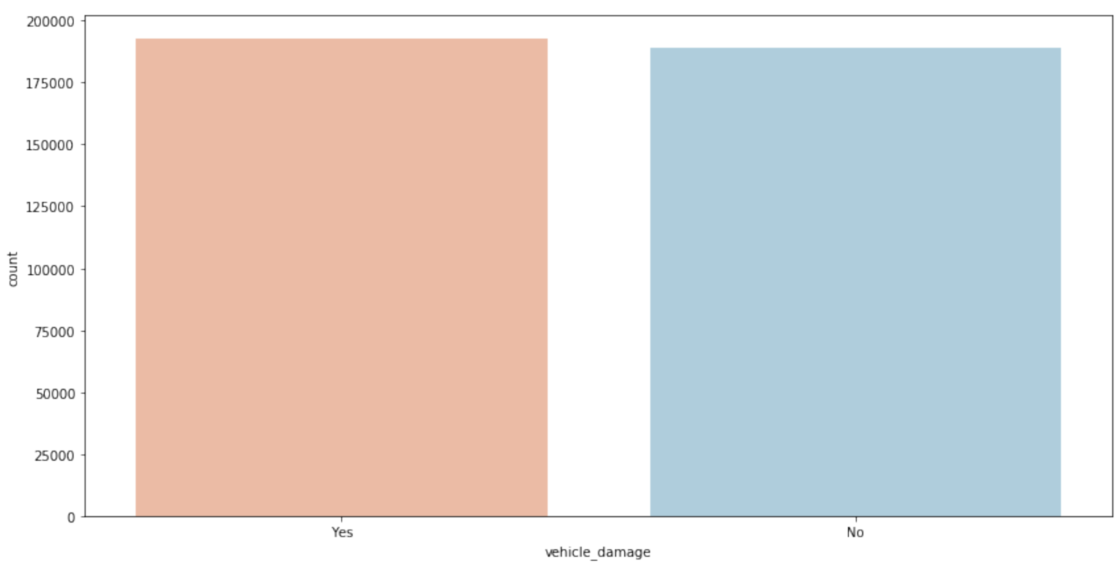
[22]:

	vehicle_damage	count	percent
0	Yes	192413	50.487656
1	No	188696	49.512344

Half of the health insurance customer have had their vehicle damaged and half have not

[23]:
```
plt.figure(figsize = (14 ,7))
sns.countplot(df['vehicle_damage'], palette = 'RdBu')
```

[23]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede4146e10>



[24]:
```
df.stb.freq(['policy_sales_channel'], cum_cols=False).head()
```

[24]:

	policy_sales_channel	count	percent
0	152.0	134784	35.366260
1	26.0	79700	20.912652
2	124.0	73995	19.415705
3	160.0	21779	5.714638
4	156.0	10661	2.797362

Sales channel 152 have the most success selling health insurance product

[25]:
```
df.stb.freq(['response'], cum_cols=False).head()
```

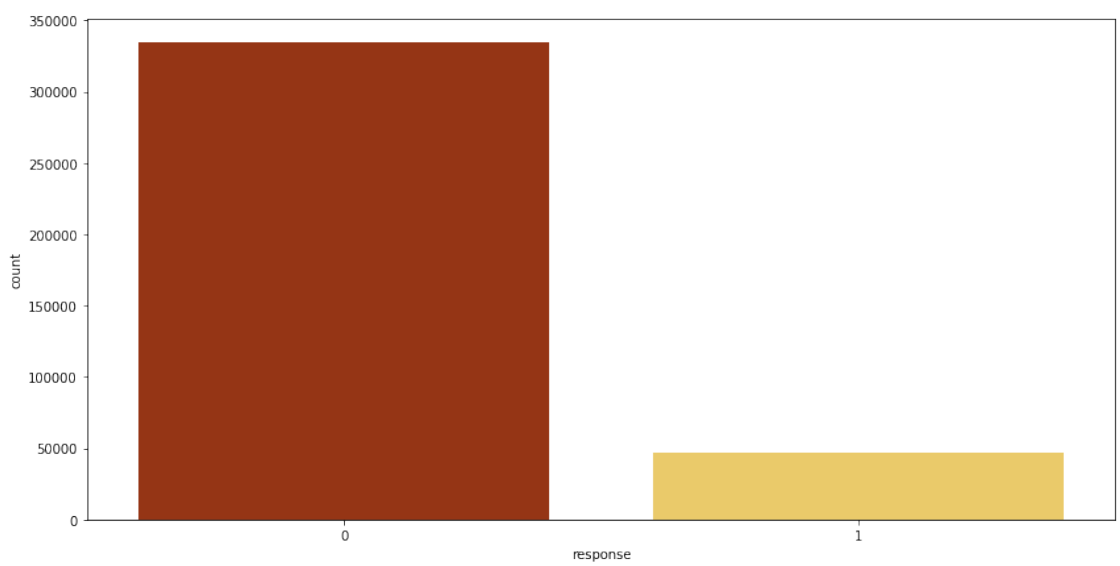
[25]:

	response	count	percent
0	0	334399	87.743664
1	1	46710	12.256336

Only 12 percent that's interested in buying vehicle insurance

[26]:
```
plt.figure(figsize = (14 ,7))
sns.countplot(df['response'], palette = 'afmhot')
```

[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede4060bd0>



[27]:
```
df.stb.freq(['premium_group'], cum_cols=False).head()
```

[27]:

	premium_group	count	percent
0	Silver	247942	65.058028
1	Bronze	100963	26.491896
2	Gold	31426	8.245935
3	Platinum	666	0.174753
4	Diamond	112	0.029388

Silver premium seems to be the most popular among health insurance customers

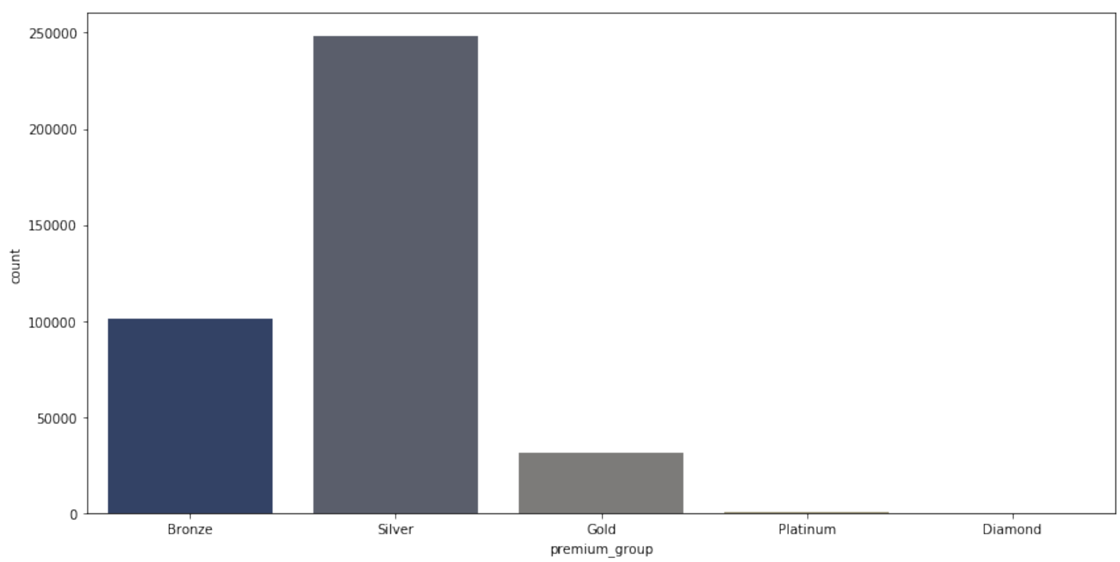
Premium Group

Bronze : 2600 - 25000
Silver = 25001 - 50000
Gold = 50001 - 100000
Platinum = 100001 - 200000
Diamond = 200001 >

There's only a few customers that has platinum and diamond premium

[28]:
```
plt.figure(figsize = (14 ,7))
sns.countplot(df['premium_group'], palette = 'cividis')
```

[28]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede4c3ef10>



[29]:
```
df.stb.freq(['Generation'], cum_cols=False).head()
```

[29]:

	Generation	count	percent
0	Millenials	181876	47.722830
1	Gen X	110689	29.043922

```
2       Boomer   70794  18.575788
3       Silent   17750   4.657460
```

The older generation are less likely to have a health insurance

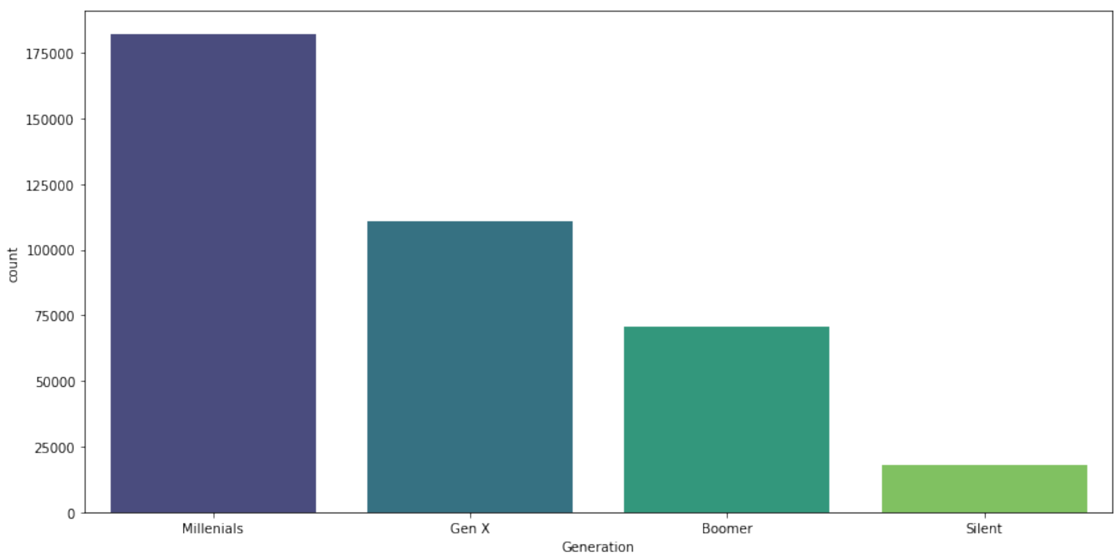
Millenial generation are the highest customer of health insurance

Generation Group :

Millenials : 18 - 34
Gen X : 35 - 50
Boomer : 51 - 69
Silent : 70 and Above

[30]:
```
plt.figure(figsize = (14 ,7))
sns.countplot(df['Generation'], palette = 'viridis')
```

[30]: `<matplotlib.axes._subplots.AxesSubplot at 0x7fede4146490>`



[31]:
```
df.head()
```

[31]:
```
   id  gender  age  driving_license  region_code  previously_insured  \
0   1    Male   44                1         28.0                   0
1   2    Male   76                1          3.0                   0
2   3    Male   47                1         28.0                   0
3   4    Male   21                1         11.0                   1
4   5  Female   29                1         41.0                   1

  vehicle_age vehicle_damage  annual_premium  policy_sales_channel  vintage  \
```

```
0   > 2 Years            Yes         40454.0                    26.0      217
1    1-2 Year             No         33536.0                    26.0      183
2   > 2 Years            Yes         38294.0                    26.0       27
3    < 1 Year             No         28619.0                   152.0      203
4    < 1 Year             No         27496.0                   152.0       39

   response premium_group  Generation
0         1        Silver       Gen X
1         0        Silver      Silent
2         1        Silver       Gen X
3         0        Silver  Millenials
4         0        Silver  Millenials
```

0.1.2 Multivariate

Since our target column is response first we are going to crosstab the response column with all the feature before we dig deeper to other analysis

[32]:
```python
pd.crosstab(index = df['response'], columns = df['gender'], normalize = 'index')
```

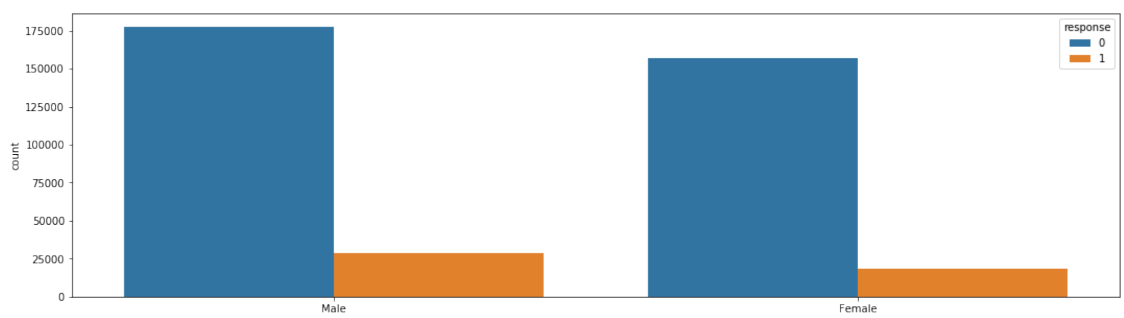
[32]:
```
gender      Female      Male
response
0         0.469006  0.530994
1         0.389317  0.610683
```

Gender and Response

Male are more likely to be interested to vehicle insurance compared to women

61% of interested response are from male respondents

[33]:
```python
plt.figure(figsize = (18, 5))
sns.countplot(df['gender'], hue = df['response'])
plt.show()
```



[34]:
```
pd.crosstab(index = df['response'], columns = 'Average Age', values =␣
↪df['age'], aggfunc='mean')
```

[34]:
```
col_0      Average Age
response
0            38.178227
1            43.435560
```

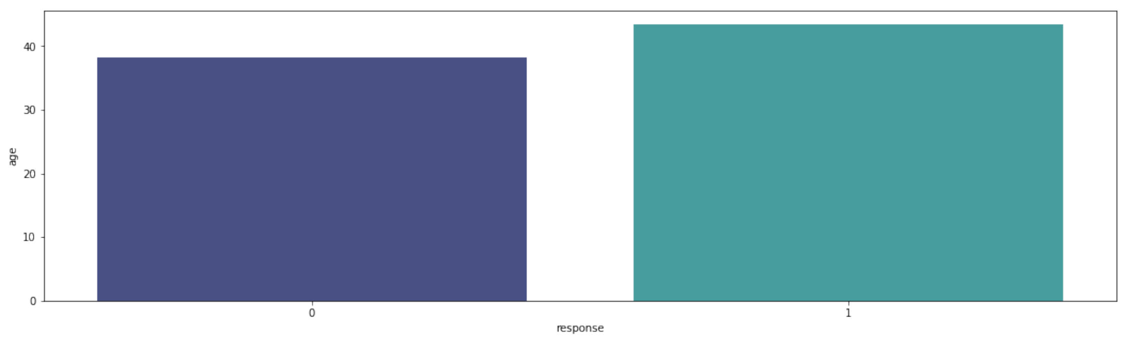
Average age and Response

The average age of customer who is interested with vehicle insurance is 43 years old

While for customers who's not interested are 38 yo

This might show that younger customers are not interested with vehicle insurance

[35]:
```
plt.figure(figsize = (18, 5))
sns.barplot(x = df['response'], y = df['age'], palette = 'mako', ci = False)
plt.show()
```



[36]:
```
### average age and Previously Insured

pd.crosstab(index = df['previously_insured'], columns = 'Average Age', values =␣
↪df['age'], aggfunc='mean')

# The Average age of customer that has a vehicle insurance is 34.5
# and the average age of customer that does not have a vehicle insurance is 42.4

# this tells that young customer will probably have a vehicle insurance␣
↪compared to the older customers
```

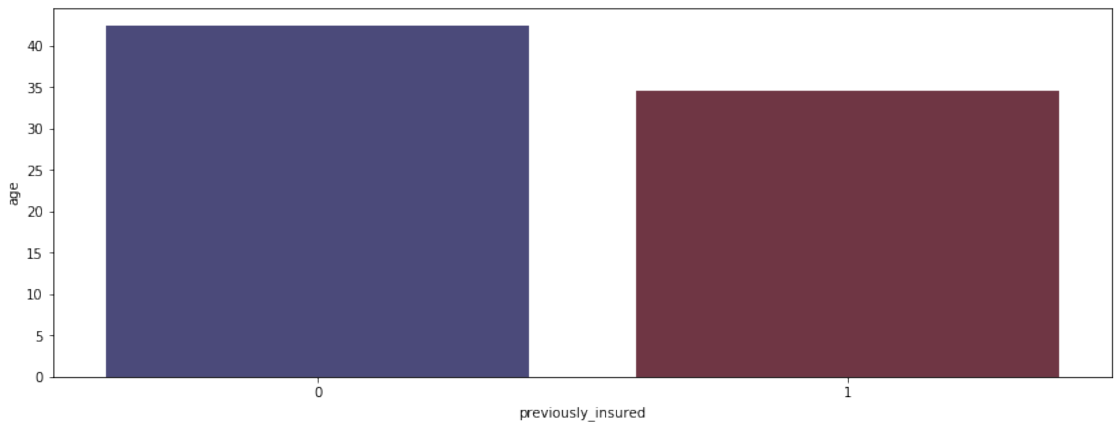
[36]:
```
col_0                 Average Age
previously_insured
0                        42.45564
1                        34.52684
```

Average age and Previously Insured

The Average age of customer that has a vehicle insurance is 34.5 and the average age of customer that does not have a vehicle insurance is 42.4

This tells that young customer will probably have a vehicle insurance compared to the older customers

[37]:
```
plt.figure(figsize = (14, 5))
sns.barplot(x = df['previously_insured'], y = df['age'], palette = 'icefire',␣
↪ci = False)
plt.show()
```



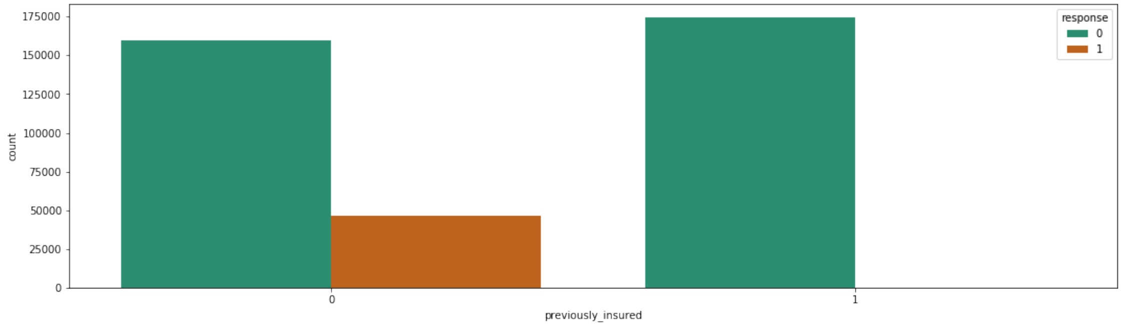
[38]:
```
pd.crosstab(index = df['response'], columns = df['previously_insured'],␣
↪normalize = 'columns')
```

[38]:

previously_insured	0	1
response		
0	0.774546	0.999095
1	0.225454	0.000905

Almost every customer who already have a vehicle insurance is not interested with another vehicle insurance out of all customer who does not have a vehicle insurance almost a quarter of them are intersted with vehicle insurance

[39]:
```
plt.figure(figsize = (18, 5))
sns.countplot(df['previously_insured'], hue = df['response'], palette = 'Dark2')
plt.show()
```

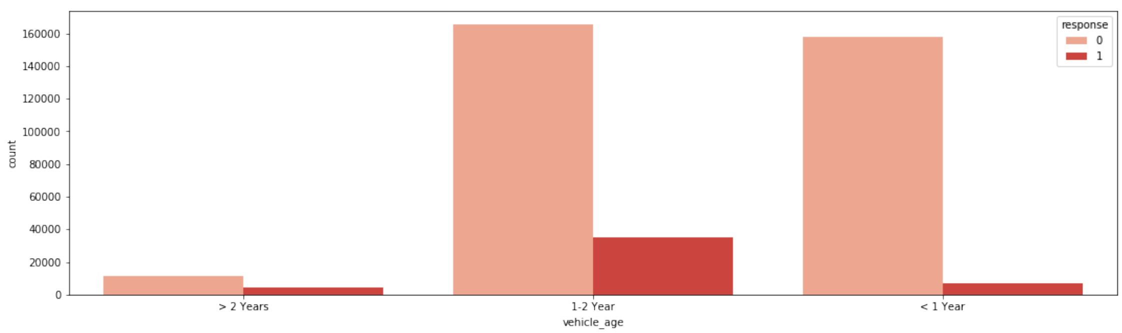


[40]:
```
pd.crosstab(index = df['response'], columns = df['vehicle_age'], normalize =␣
↪'columns')
```

[40]:
```
vehicle_age  1-2 Year  < 1 Year  > 2 Years
response
0            0.826245  0.956295   0.706254
1            0.173755  0.043705   0.293746
```

From the vehicle age group customer who has a newer vehicle are less likely to be interested with vehicle insurance customer who has a vehicle that's older than 2 years are more likely to be interested to vehicle insurance

[41]:
```
plt.figure(figsize = (18, 5))
sns.countplot(df['vehicle_age'], hue = df['response'], palette = 'Reds')
plt.show()
```

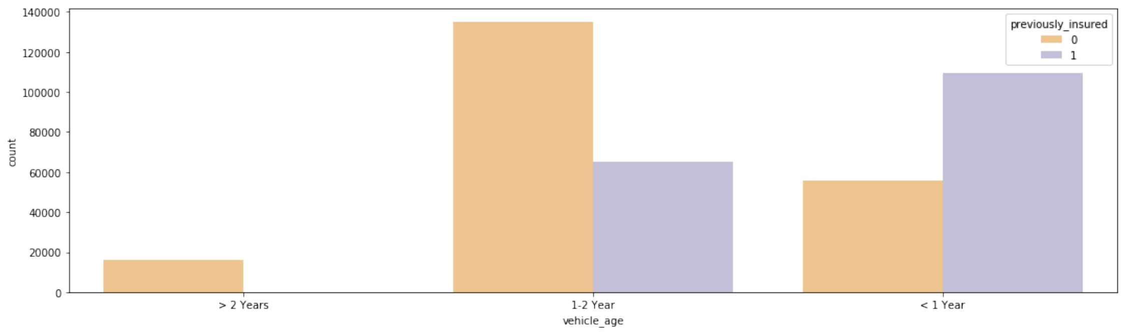


[42]:
```
pd.crosstab(index = df['previously_insured'], columns = df['vehicle_age'],␣
↪normalize = 'columns')
```

[42]:
```
vehicle_age         1-2 Year  < 1 Year  > 2 Years
previously_insured
0                    0.67389  0.336976   0.997126
1                    0.32611  0.663024   0.002874
```

The newer the vehicle the more likely it's insured, this columns below showed that 66.3% of car aged 1 or below are insured

[43]:
```
plt.figure(figsize = (18, 5))
sns.countplot(df['vehicle_age'], hue = df['previously_insured'], palette =␣
↪'PuOr')
plt.show()
```



[44]:
```
pd.crosstab(index = df['response'], columns = [df['vehicle_age'],␣
↪df['previously_insured']], normalize = 'columns')
```

[44]:

vehicle_age	1-2 Year		< 1 Year		> 2 Years	
previously_insured	0	1	0	1	0	1
response						
0	0.742864	0.998546	0.871419	0.999433	0.70547	0.978261
1	0.257136	0.001454	0.128581	0.000567	0.29453	0.021739

Customer who has a newer car are more likely to have their vehicle insured this could be an insight that insurance company needs to work with a dealership to have a bundling product of vehicle & insurance

[45]:
```
pd.crosstab(index = df['response'], columns = df['vehicle_damage'], normalize =␣
↪'index')
```

[45]:

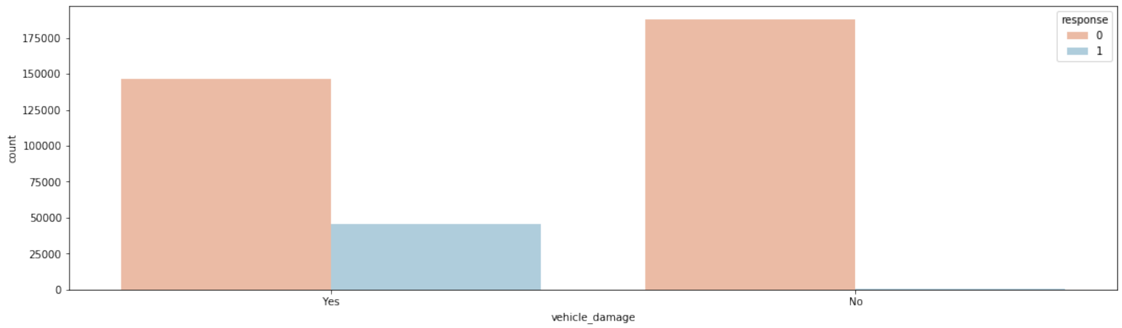
vehicle_damage	No	Yes
response		
0	0.561347	0.438653
1	0.021023	0.978977

Customer who're intersted with vehicle insurance 98% have had a vehicle damage in the past

[46]:
```
pd.crosstab(index = df['response'], columns = df['vehicle_damage'], normalize =␣
↪'columns')
```

[46]:
```
vehicle_damage        No       Yes
response
0               0.994796  0.762345
1               0.005204  0.237655
```

[47]:
```
plt.figure(figsize = (18, 5))
sns.countplot(df['vehicle_damage'], hue = df['response'], palette = 'RdBu')
plt.show()
```



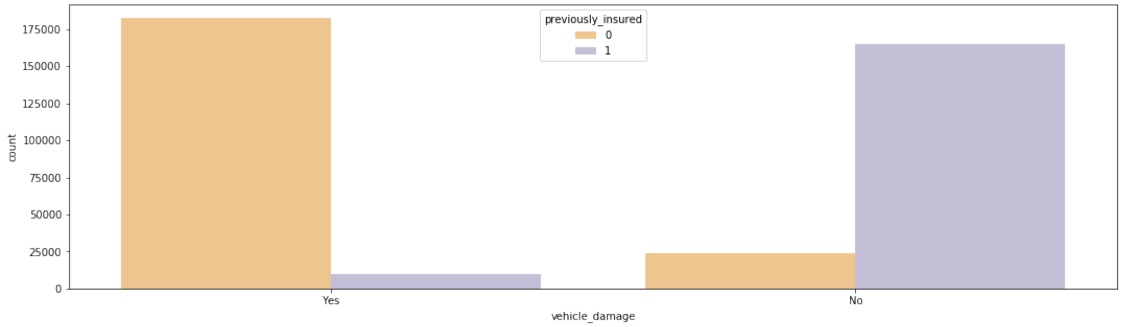
[48]:
```
pd.crosstab(index = df['previously_insured'], columns = df['vehicle_damage'],␣
↪normalize = 'columns')
```

[48]:
```
vehicle_damage            No       Yes
previously_insured
0                   0.127136  0.948434
1                   0.872864  0.051566
```

Almost 95 % customer who have had their vehicle previously damaged doesn't have a vehicle insurance, while 87 % of customer who had never have any vehicle damaged have a vehicle insurance.

People who have vehicle insurance are more likely to be careful to their vehicle.

[49]:
```
plt.figure(figsize = (18, 5))
sns.countplot(df['vehicle_damage'], hue = df['previously_insured'], palette =␣
↪'PuOr')
plt.show()
```



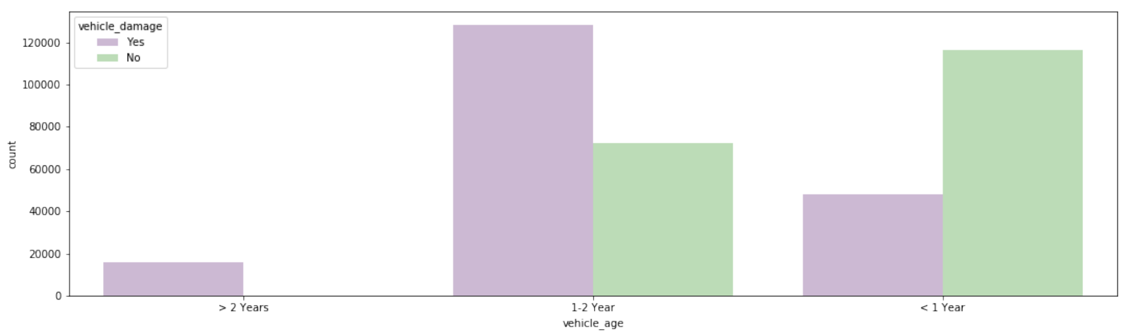
[50]:
```
pd.crosstab(index = df['vehicle_age'], columns = df['vehicle_damage'],␣
↪normalize = 'index')
```

[50]:
```
vehicle_damage        No       Yes
vehicle_age
1-2 Year        0.359886  0.640114
< 1 Year        0.707524  0.292476
> 2 Years       0.000937  0.999063
```

Cars that's more than 2 years of age are the most likely to have had a vehicle damage

The younger the vehicle the less likely that the vehicle has a vehicle damage

[51]:
```
plt.figure(figsize = (18, 5))
sns.countplot(df['vehicle_age'], hue = df['vehicle_damage'], palette = 'PRGn')
plt.show()
```



[52]:
```
pd.crosstab(index = df['response'], columns = [df['vehicle_damage'],␣
↪df['previously_insured']], normalize = 'index')
```

[52]:
```
vehicle_damage            No                  Yes
previously_insured         0         1          0         1
response
```

0	0.069019	0.492328	0.409239	0.029414
1	0.019482	0.001541	0.977136	0.001841

Customer who never had any vehicle damage and has a vehicle insurance are the most likely not interested in another vehicle insurance

From all the customer who is interested 97% of them does not have vehicle insurance and had a vehicle damage in the past

Targeting customer who does not have a vehicle insurance and have had a vehicle damage in the past

[53]:
```
pd.crosstab(index = df['response'], columns = 'Median Premium', values =␣
↪df['annual_premium'], aggfunc='median')
```

[53]:

col_0	Median Premium
response	
0	31504.0
1	33002.0

The median of customer premium doesn't really differentitate of the responses

[54]:
```
pd.crosstab(index = df['response'], columns = 'Average Vintage', values =␣
↪df['vintage'], aggfunc='median')

# customer loyalty doesn't have any effect on the response towards vehicle␣
↪insurance
```

[54]:

col_0	Average Vintage
response	
0	154
1	154

[55]:
```
pd.crosstab(index = df['response'], columns = df['premium_group'], normalize =␣
↪'columns')
```

[55]:

premium_group	Bronze	Silver	Gold	Platinum	Diamond
response					
0	0.882918	0.879044	0.848024	0.848348	0.803571
1	0.117082	0.120956	0.151976	0.151652	0.196429

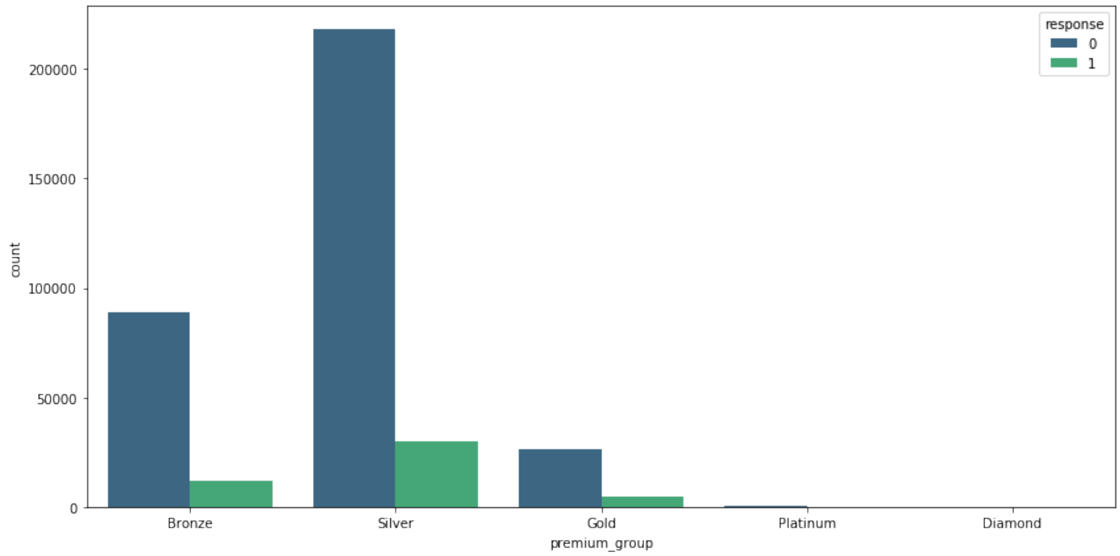
The more expensive the premium group the more likely the customer are interested with the vehicle insurance

Assumption:

```
1. Customer who has more expensive premium group are more likely have a higher income
2. The higher the income the more likely they have money to spend
```

```
[56]: plt.figure(figsize = (14, 7))
      sns.countplot(df['premium_group'], hue = df['response'], palette = 'viridis')
```

```
[56]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede351a0d0>
```



Eventhough platinum and diamond has a higher percentage of intersted responds, however there're only few numbers of customers are in those premium_group

```
[57]: pd.crosstab(index = df['response'], columns = [df['premium_group'],␣
       ↪df['previously_insured']], normalize = 'columns')
```

```
[57]: premium_group          Bronze              Silver              Gold  \
      previously_insured          0         1         0         1         0
      response
      0                    0.791091  0.998879  0.771191  0.999148  0.749921
      1                    0.208909  0.001121  0.228809  0.000852  0.250079

      premium_group                   Platinum        Diamond
      previously_insured          1         0    1         0    1
      response
      0                    0.999353  0.720994  1.0  0.706667  1.0
      1                    0.000647  0.279006  0.0  0.293333  0.0
```

For customer who does not have vehicle insurance before : The more expensive the premium group the more likely the customer are interested with the vehicle insurance

While for customer who's previously have vehicle insurance : The more expensive the group the less likely they will sign

[58]:
```
pd.crosstab(index = df['response'], columns = [df['premium_group'],
 ↪df['vehicle_damage']], normalize = 'columns')
```

[58]:
```
premium_group     Bronze              Silver              Gold              \
vehicle_damage        No       Yes        No       Yes        No       Yes
response
0               0.991645  0.779109  0.995756  0.758805  0.997425  0.739765
1               0.008355  0.220891  0.004244  0.241195  0.002575  0.260235

premium_group  Platinum           Diamond
vehicle_damage       No       Yes      No       Yes
response
0               0.99373  0.714697     1.0  0.681159
1               0.00627  0.285303     0.0  0.318841
```

[59]:
```
pd.crosstab(index = df['response'], columns = df['Generation'], normalize =
 ↪'columns')
```

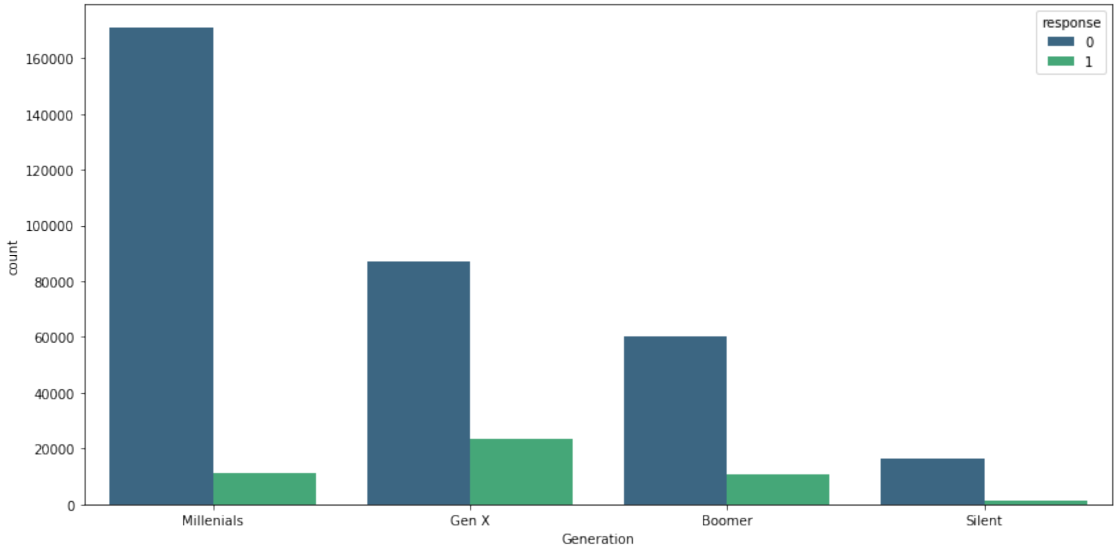
[59]:
```
Generation  Millenials    Gen X   Boomer    Silent
response
0             0.938788  0.78614  0.85075  0.924563
1             0.061212  0.21386  0.14925  0.075437
```

Generation and Response

Millenials shown to be the generation that's less likely to be intersted in vehicle insurance WHY? Gen X and Boomer are 2 generation that's most likely to be interested with vehicle insurance

[60]:
```
plt.figure(figsize = (14, 7))
sns.countplot(df['Generation'], hue = df['response'], palette = 'viridis')
```

[60]: <matplotlib.axes._subplots.AxesSubplot at 0x7fede362bcd0>



[61]:
```
pd.crosstab(index = df['Generation'], columns = [df['vehicle_damage'],␣
↪df['response']], normalize = 'columns')
```

[61]:

vehicle_damage	No		Yes	
response	0	1	0	1
Generation				
Millenials	0.645780	0.503055	0.337601	0.232658
Gen X	0.187306	0.373727	0.353526	0.509644
Boomer	0.128424	0.112016	0.246249	0.228656
Silent	0.038489	0.011202	0.062624	0.029041

Out of all customers that have had vehicle damage in the past Gen X are more likely to response interested to vehicle insurance

[62]:
```
pd.crosstab(index = df['Generation'], columns = df['vehicle_damage'], normalize␣
↪= 'index')
```

[62]:

vehicle_damage	No	Yes
Generation		
Millenials	0.669225	0.330775
Gen X	0.320962	0.679038
Boomer	0.342077	0.657923
Silent	0.407662	0.592338

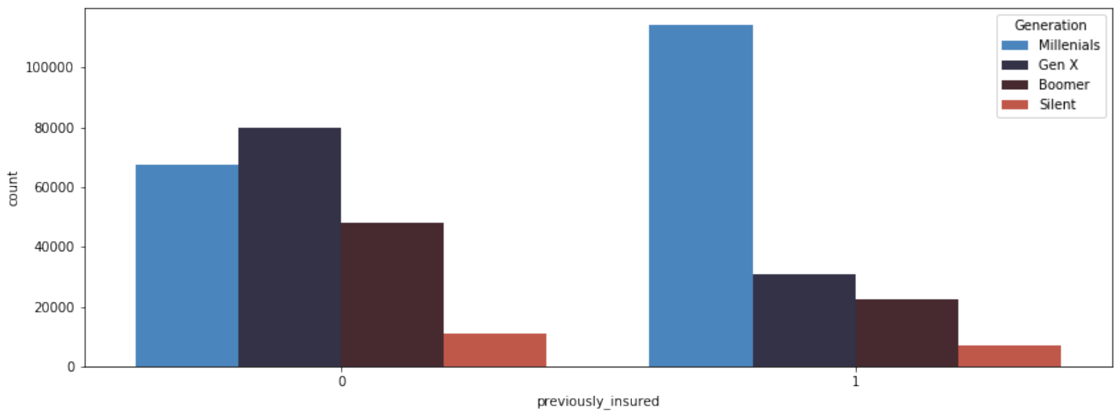
Gen X in generation with the highest vehicle damage percentage

[63]:
```
pd.crosstab(index = df['Generation'], columns = df['previously_insured'],␣
↪normalize = 'index')
```

[63]:
```
previously_insured         0         1
Generation
Millenials          0.372012  0.627988
Gen X               0.720279  0.279721
Boomer              0.681428  0.318572
Silent              0.611437  0.388563
```

This shows that maybe why millenial is not interested in vehicle insurance is because 62.7 % of millenials already have vehicle insurance this shows that Gen X generation the less likely they have a vehicle insurance before

[64]:
```
plt.figure(figsize = (14, 5))
sns.countplot(x = df['previously_insured'], hue = df['Generation'], palette =␣
 ↪'icefire')
plt.show()
```



[65]:
```
pd.crosstab(index = df['response'], columns = [df['Generation'],␣
 ↪df['previously_insured']], normalize = 'index')
```

[65]:
```
Generation          Millenials                Gen X                Boomer  \
previously_insured           0         1          0         1          0
response
0                     0.169271  0.341326   0.167826  0.092393   0.112704
1                     0.236694  0.001648   0.505374  0.001413   0.225926

Generation                          Silent
previously_insured         1         0         1
response
0                   0.067405  0.028457  0.020619
1                   0.000278  0.028623  0.000043
```

This table below shows that millenial who has a vehicle insurance are most likely not to be interested in vehicle insurance

[66]:
```
pd.crosstab(index = df['Generation'], columns = df['vehicle_age'], normalize =␣
↪'index')
```

[66]:

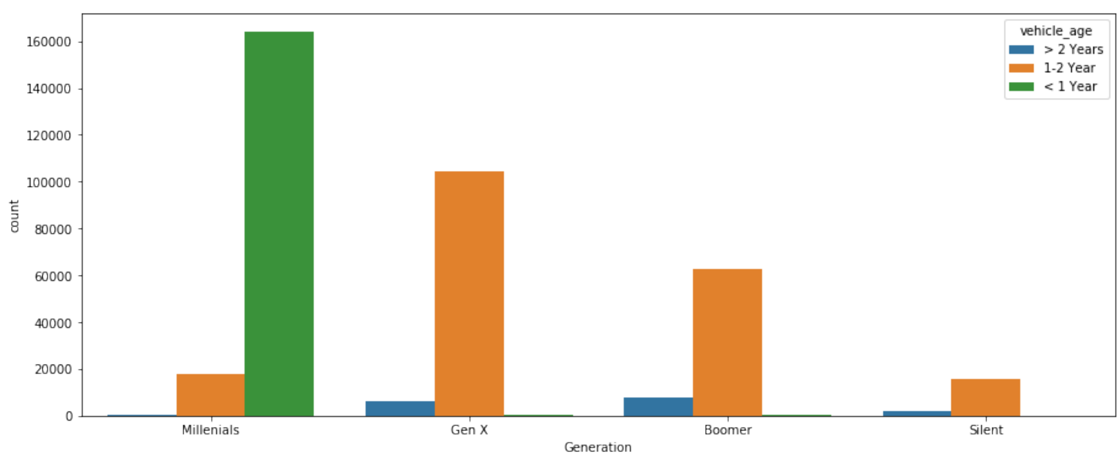
vehicle_age	1-2 Year	< 1 Year	> 2 Years
Generation			
Millenials	0.097539	0.901339	0.001122
Gen X	0.940816	0.005095	0.054088
Boomer	0.886332	0.003757	0.109910
Silent	0.884000	0.001352	0.114648

Majority of millenials 90.1% have a vehicle age below one year, and from our analysis before majority of vehicle that's less then 1 year of age is already insured

There's less than one percent of millenials who has cars over 2 years

This open up on how to target millenial customers by working with a dealership that sells new car, and bundling it with an insurance product to get the millenial generation market and since almost 94% millenials says they're not interested with vehicle insurance product this kind of partnership with dealer will catch the market of millenial that we're missing

[67]:
```
plt.figure(figsize = (15, 6))
sns.countplot(df['Generation'], hue = df['vehicle_age'])
plt.show()
```



[68]:
```
pd.crosstab(index = df['policy_sales_channel'], columns = df['response'],␣
↪normalize = 'columns').sort_values(1, ascending = False).head()
```

[68]:

response	0	1
policy_sales_channel		
26.0	0.190817	0.340206
124.0	0.179423	0.299636
152.0	0.391526	0.082595
156.0	0.025012	0.049176
157.0	0.014623	0.038407

Sales Channel

The policy sales channel no 26 and policy sales channel number 124 are the 2 highest percentage of interested response

There's no further explanataion on what are this number :(

[69]:
```
pd.crosstab(index = df['policy_sales_channel'], columns = df['response'],␣
↪normalize = 'columns').sort_values(0, ascending = False).head()
```

[69]:

response	0	1
policy_sales_channel		
152.0	0.391526	0.082595
26.0	0.190817	0.340206
124.0	0.179423	0.299636
160.0	0.063708	0.010169
156.0	0.025012	0.049176

Most ineffective sales channel of all sales channel, policy sales channel number 152 seems to be the least effective to offer health insurance customers a vehicle insurance

[70]:
```
top_5_region =  pd.crosstab(index = df['region_code'], columns =␣
↪df['response']).sort_values(1, ascending = False).head()
top_5_region
```

[70]:

response	0	1
region_code		
28.0	86498	19917
8.0	30620	3257
41.0	16039	2224
46.0	17717	2032
29.0	9700	1365

Region

Region 28 has the highest number of customers of all region that's maybe why it has the the highest number of interested response

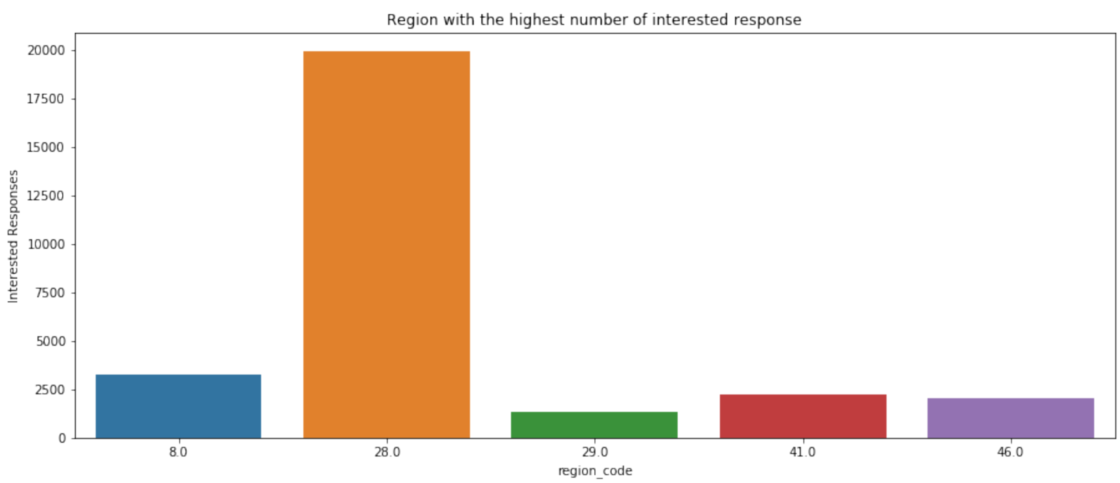
The region 28 has the highest percentage of customers who is interested with vehicle insurance product unfortunately there's no explanation on each number of region of which is where

[71]:
```
pd.crosstab(index = df['region_code'], columns = df['response'], normalize =␣
↪'index').sort_values(1, ascending = False).head()
```

[71]:
```
response            0         1
region_code
38.0         0.807996  0.192004
28.0         0.812837  0.187163
19.0         0.837134  0.162866
4.0          0.841755  0.158245
23.0         0.846939  0.153061
```

If we compared region to region interested rate, region 38 has the highest percentage of interested while region 28 is on the 2nd place

[72]:
```
plt.figure(figsize = (15, 6))

sns.barplot(x = top_5_region.index, y = top_5_region[1])
plt.ylabel('Interested Responses')
plt.title('Region with the highest number of interested response')
plt.show()
```



HICS-ML

April 3, 2021

Python Libraries Import

[1]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,␣
↪GridSearchCV,RandomizedSearchCV
from sklearn.metrics import␣
↪recall_score,precision_score,f1_score,accuracy_score, confusion_matrix,␣
↪classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

Getting The Basic Understanding of the Data

[2]:
```python
df = pd.read_csv('train.csv')
df.head()
```

[2]:
```
   id  Gender  Age  Driving_License  Region_Code  Previously_Insured  \
0   1    Male   44                1         28.0                   0
1   2    Male   76                1          3.0                   0
2   3    Male   47                1         28.0                   0
3   4    Male   21                1         11.0                   1
4   5  Female   29                1         41.0                   1

  Vehicle_Age Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  \
0   > 2 Years            Yes         40454.0                  26.0      217
1    1-2 Year             No         33536.0                  26.0      183
2   > 2 Years            Yes         38294.0                  26.0       27
3    < 1 Year             No         28619.0                 152.0      203
4    < 1 Year             No         27496.0                 152.0       39

   Response
0         1
1         0
2         1
```

```
3          0
4          0
```

Some of the object will need an encoding before processing to the machine learning modeling

'id' column will be dropped because it will not affect anything in our analysis and machine learning process

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381109 entries, 0 to 381108
Data columns (total 12 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   id                    381109 non-null  int64
 1   Gender                381109 non-null  object
 2   Age                   381109 non-null  int64
 3   Driving_License       381109 non-null  int64
 4   Region_Code           381109 non-null  float64
 5   Previously_Insured    381109 non-null  int64
 6   Vehicle_Age           381109 non-null  object
 7   Vehicle_Damage        381109 non-null  object
 8   Annual_Premium        381109 non-null  float64
 9   Policy_Sales_Channel  381109 non-null  float64
 10  Vintage               381109 non-null  int64
 11  Response              381109 non-null  int64
dtypes: float64(3), int64(6), object(3)
memory usage: 34.9+ MB
```

[4]:
```
df.describe()
```

[4]:

	id	Age	Driving_License	Region_Code	\
count	381109.000000	381109.000000	381109.000000	381109.000000	
mean	190555.000000	38.822584	0.997869	26.388807	
std	110016.836208	15.511611	0.046110	13.229888	
min	1.000000	20.000000	0.000000	0.000000	
25%	95278.000000	25.000000	1.000000	15.000000	
50%	190555.000000	36.000000	1.000000	28.000000	
75%	285832.000000	49.000000	1.000000	35.000000	
max	381109.000000	85.000000	1.000000	52.000000	

	Previously_Insured	Annual_Premium	Policy_Sales_Channel	\
count	381109.000000	381109.000000	381109.000000	
mean	0.458210	30564.389581	112.034295	
std	0.498251	17213.155057	54.203995	

```
min          0.000000     2630.000000             1.000000
25%          0.000000    24405.000000            29.000000
50%          0.000000    31669.000000           133.000000
75%          1.000000    39400.000000           152.000000
max          1.000000   540165.000000           163.000000

             Vintage       Response
count  381109.000000  381109.000000
mean      154.347397       0.122563
std        83.671304       0.327936
min        10.000000       0.000000
25%        82.000000       0.000000
50%       154.000000       0.000000
75%       227.000000       0.000000
max       299.000000       1.000000
```

The average customers vintage (numbers of day been insured in the compant is 154 days)

No customers in this data set have been with the insurance company for 1 full year

The oldest customers in this dataset is 85 while the median is 36

The most expensive annual premium is almost 17 times more expensive compared to the median annual premium

This data definitely need a scalling to get a better result in the machine learning process

[5]:
```python
df.describe(include = 'O')
```

[5]:
```
        Gender Vehicle_Age Vehicle_Damage
count   381109      381109         381109
unique       2           3              2
top       Male    1-2 Year            Yes
freq    206089      200316         192413
```

Majority of the health insurance owner is male

Knowing all the unique value in the columns

[6]:
```python
df_unique = df.drop(columns = 'id')

for column in df_unique.columns:
    print(f"{column}: ")
```

```python
    print("")
    print(df_unique[column].unique())
    print("")
```

```
Gender:

['Male' 'Female']

Age:

[44 76 47 21 29 24 23 56 32 41 71 37 25 42 60 65 49 34 51 26 57 79 48 45
 72 30 54 27 38 22 78 20 39 62 58 59 63 50 67 77 28 69 52 31 33 43 36 53
 70 46 55 40 61 75 64 35 66 68 74 73 84 83 81 80 82 85]

Driving_License:

[1 0]

Region_Code:

[28.  3. 11. 41. 33.  6. 35. 50. 15. 45.  8. 36. 30. 26. 16. 47. 48. 19.
 39. 23. 37.  5. 17.  2.  7. 29. 46. 27. 25. 13. 18. 20. 49. 22. 44.  0.
  9. 31. 12. 34. 21. 10. 14. 38. 24. 40. 43. 32.  4. 51. 42.  1. 52.]

Previously_Insured:

[0 1]

Vehicle_Age:

['> 2 Years' '1-2 Year' '< 1 Year']

Vehicle_Damage:

['Yes' 'No']

Annual_Premium:

[ 40454.  33536.  38294. …  20706. 101664.  69845.]

Policy_Sales_Channel:

[ 26. 152. 160. 124.  14.  13.  30. 156. 163. 157. 122.  19.  22.  15.
 154.  16.  52. 155.  11. 151. 125.  25.  61.   1.  86.  31. 150.  23.
  60.  21. 121.   3. 139.  12.  29.  55.   7.  47. 127. 153.  78. 158.
  89.  32.   8.  10. 120.  65.   4.  42.  83. 136.  24.  18.  56.  48.
 106.  54.  93. 116.  91.  45.   9. 145. 147.  44. 109.  37. 140. 107.
```

 128. 131. 114. 118. 159. 119. 105. 135. 62. 138. 129. 88. 92. 111.
 113. 73. 36. 28. 35. 59. 53. 148. 133. 108. 64. 39. 94. 132.
  46. 81. 103. 90. 51. 27. 146. 63. 96. 40. 66. 100. 95. 123.
  98. 75. 69. 130. 134. 49. 97. 38. 17. 110. 80. 71. 117. 58.
  20. 76. 104. 87. 84. 137. 126. 68. 67. 101. 115. 57. 82. 79.
 112. 99. 70.   2. 34. 33. 74. 102. 149. 43.   6. 50. 144. 143.
  41.]

```
Vintage:

[217 183  27 203  39 176 249  72  28  80  46 289 221  15  58 147 256 299
 158 102 116 177 232  60 180  49  57 223 136 222 149 169  88 253 107 264
 233  45 184 251 153 186  71  34  83  12 246 141 216 130 282  73 171 283
 295 165  30 218  22  36  79  81 100  63 242 277  61 111 167  74 235 131
 243 248 114 281  62 189 139 138 209 254 291  68  92  52  78 156 247 275
  77 181 229 166  16  23  31 293 219  50 155  66 260  19 258 117 193 204
 212 144 234 206 228 125  29  18  84 230  54 123 101  86  13 237  85  98
  67 128  95  89  99 208 134 135 268 284 119 226 105 142 207 272 263  64
  40 245 163  24 265 202 259  91 106 190 162  33 194 287 292  69 239 132
 255 152 121 150 143 198 103 127 285 214 151 199  56  59 215 104 238 120
  21  32 270 211 200 197  11 213  93 113 178  10 290  94 231 296  47 122
 271 278 276  96 240 172 257 224 173 220 185  90  51 205  70 160 137 168
  87 118 288 126 241  82 227 115 164 236 286 244 108 274 201  97  25 174
 182 154  48  20  53  17 261  41 266  35 140 269 146 145  65 298 133 195
  55 188  75  38  43 110  37 129 170 109 267 279 112 280  76 191  26 161
 179 175 252  42 124 187 148 294  44 157 192 262 159 210 250  14 273 297
 225 196]

Response:

[1 0]
```

0.0.1 Handling null values, Handling outliers and Encoding process

[7]:
```python
df.isna().sum()
```

[7]:
```
id                      0
Gender                  0
Age                     0
Driving_License         0
Region_Code             0
Previously_Insured      0
Vehicle_Age             0
Vehicle_Damage          0
Annual_Premium          0
Policy_Sales_Channel    0
```

```
Vintage                 0
Response                0
dtype: int64
```

Apparently there is no null value in all the rows and columns so we dont need to do anything about it for now

[8]:
```
df.drop(columns = 'id', inplace = True)
df.head()
```

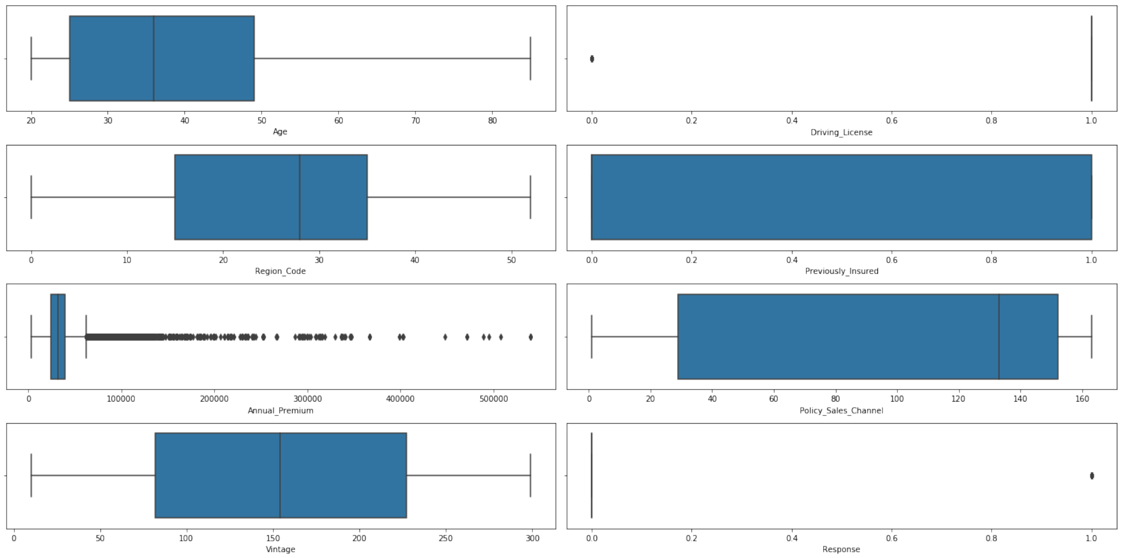
[8]:

	Gender	Age	Driving_License	Region_Code	Previously_Insured	Vehicle_Age
0	Male	44	1	28.0	0	> 2 Years
1	Male	76	1	3.0	0	1-2 Year
2	Male	47	1	28.0	0	> 2 Years
3	Male	21	1	11.0	1	< 1 Year
4	Female	29	1	41.0	1	< 1 Year

	Vehicle_Damage	Annual_Premium	Policy_Sales_Channel	Vintage	Response
0	Yes	40454.0	26.0	217	1
1	No	33536.0	26.0	183	0
2	Yes	38294.0	26.0	27	1
3	No	28619.0	152.0	203	0
4	No	27496.0	152.0	39	0

‘id’ wont be needed in the analysis and it wont be needed for the machine learning process so it’s kind of redundant to keep it

[9]:
```
plt.figure(figsize = (20, 10))
x = 1

for column in df.describe().columns:
    plt.subplot(4,2, x)
    sns.boxplot(df[column])
    x+=1

plt.tight_layout()
plt.show()
```



Looking at the box plot to check all the outliers, we see that there's a lot of outliers in the annual premium this will need to be scaled with robust scaler to better the evaluation matrix / binning

0.0.2 Encoding Object columns

Encoding Gender

[10]:
```
df['Gender'] = df['Gender'].map({'Female':1, 'Male':0})
df.head()
```

[10]:
```
   Gender  Age  Driving_License  Region_Code  Previously_Insured Vehicle_Age  \
0       0   44                1         28.0                   0   > 2 Years
1       0   76                1          3.0                   0    1-2 Year
2       0   47                1         28.0                   0   > 2 Years
3       0   21                1         11.0                   1    < 1 Year
4       1   29                1         41.0                   1    < 1 Year

  Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  Response
0            Yes         40454.0                  26.0      217         1
1             No         33536.0                  26.0      183         0
2            Yes         38294.0                  26.0       27         1
3             No         28619.0                 152.0      203         0
4             No         27496.0                 152.0       39         0
```

Encoding Vehicle_Damage

[11]:
```
df['Vehicle_Damage'] = df['Vehicle_Damage'].map({'Yes':1, 'No':0})
df.head()
```

[11]:
```
   Gender  Age  Driving_License  Region_Code  Previously_Insured Vehicle_Age  \
0       0   44                1         28.0                   0   > 2 Years
1       0   76                1          3.0                   0    1-2 Year
2       0   47                1         28.0                   0   > 2 Years
3       0   21                1         11.0                   1    < 1 Year
4       1   29                1         41.0                   1    < 1 Year

   Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  Response
0               1         40454.0                  26.0      217         1
1               0         33536.0                  26.0      183         0
2               1         38294.0                  26.0       27         1
3               0         28619.0                 152.0      203         0
4               0         27496.0                 152.0       39         0
```

Encoding Vehicle_Age

[12]:
```python
df['Vehicle_Age'] = df['Vehicle_Age'].map({'1-2 Year':1, '< 1 Year':0, '> 2␣
↪Years': 2})
df.head()
```

[12]:
```
   Gender  Age  Driving_License  Region_Code  Previously_Insured  Vehicle_Age  \
0       0   44                1         28.0                   0            2
1       0   76                1          3.0                   0            1
2       0   47                1         28.0                   0            2
3       0   21                1         11.0                   1            0
4       1   29                1         41.0                   1            0

   Vehicle_Damage  Annual_Premium  Policy_Sales_Channel  Vintage  Response
0               1         40454.0                  26.0      217         1
1               0         33536.0                  26.0      183         0
2               1         38294.0                  26.0       27         1
3               0         28619.0                 152.0      203         0
4               0         27496.0                 152.0       39         0
```

[13]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 381109 entries, 0 to 381108
Data columns (total 11 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Gender                381109 non-null  int64
 1   Age                   381109 non-null  int64
 2   Driving_License       381109 non-null  int64
 3   Region_Code           381109 non-null  float64
 4   Previously_Insured    381109 non-null  int64
 5   Vehicle_Age           381109 non-null  int64
 6   Vehicle_Damage        381109 non-null  int64
```

```
 7   Annual_Premium        381109 non-null  float64
 8   Policy_Sales_Channel  381109 non-null  float64
 9   Vintage               381109 non-null  int64
 10  Response              381109 non-null  int64
dtypes: float64(3), int64(8)
memory usage: 32.0 MB
```

Now all column are in int or float value and ready to be machine learning processed

0.1 Mini EDA

Hypothesis Null

- Gender is Corelated With Response
- Age is Correlated with Response
- Driving License is correlated with respponse
- Previosly_Insured correlated with response
- Vehicle_Age is correlated with response
- Vehicle Damage is Correlated with response
- Anuual_Premium
- Vintage is Correlated with response

[14]:
```
df['Gender'].value_counts()

# there are more male in this dataset compared to female
```

[14]:
```
0    206089
1    175020
Name: Gender, dtype: int64
```

[15]:
```
pd.crosstab(index = df['Gender'], columns = df['Response'], normalize = 'index')

# Dataset contain more male  than female so from perspective according to␣
↪gender are equal
```

[15]:
```
Response         0         1
Gender
0         0.861589  0.138411
1         0.896098  0.103902
```

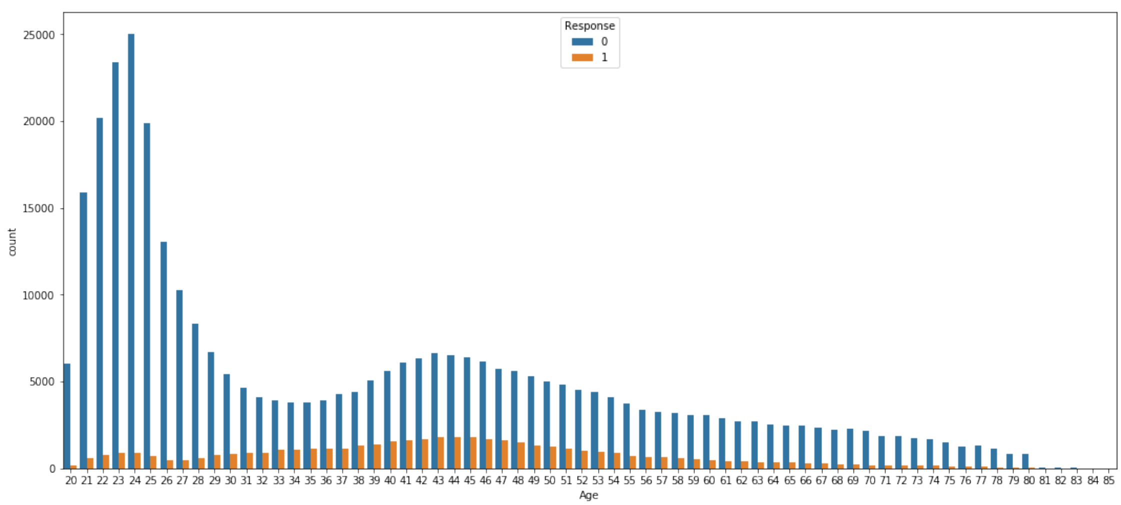
[16]:
```
pd.crosstab(index = df['Age'], columns = df['Response'], normalize = 'columns').
↪sort_values(1, ascending = False)
```

[16]:
```
Response         0         1
Age
44        0.019575  0.038771
43        0.019833  0.038643
45        0.019163  0.038000
```

```
46        0.018457  0.036545
42        0.019007  0.035346
..             …         …
81        0.000156  0.000086
82        0.000084  0.000021
83        0.000063  0.000021
84        0.000033  0.000000
85        0.000033  0.000000

[66 rows x 2 columns]
```

[17]:
```python
plt.figure(figsize = (18, 8))
sns.countplot(df['Age'], hue = df['Response'])

## people ages between from 38 to 50 are more likely to respond
# while people ages between 20 to 30 are less likely to respond
```

[17]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff6c9078110>
```



[18]:
```python
df['Driving_License'].value_counts(normalize= True)
# the number of people who doesn't have a driving license is very small in this␣
↪dataset
```

[18]:
```
1    0.997869
0    0.002131
Name: Driving_License, dtype: float64
```

[19]:
```python
pd.crosstab(index = df['Driving_License'], columns = df['Response'], normalize␣
↪= 'index')
```

```
# Since More most of the people has a driving license, majoriity of the␣
↪response that say yes are from people who has driving license
```

[19]:
```
Response                0         1
Driving_License
0                0.949507  0.050493
1                0.877283  0.122717
```

[20]:
```
df['Previously_Insured'].value_counts(normalize = True)
```

[20]:
```
0    0.54179
1    0.45821
Name: Previously_Insured, dtype: float64
```

[21]:
```
pd.crosstab(index = df['Previously_Insured'], columns = df['Response'],␣
↪normalize = 'index')

# people who previously Insured are less likely to response compared to people␣
↪who was not previously insured
```

[21]:
```
Response                   0         1
Previously_Insured
0                   0.774546  0.225454
1                   0.999095  0.000905
```

[22]:
```
pd.crosstab(index = df['Vehicle_Age'], columns = df['Response'], normalize =␣
↪'columns')

# 0  = Vehicle age < 1 year
# 1 = Vehicle age 1 - 2 year
# 2 = Vehicle Age > 2 years

# people that has vehicle for more than 2 years are more likely to response
# people whos has newer vehicle are less likely to response
```

[22]:
```
Response            0         1
Vehicle_Age
0            0.471245  0.154185
1            0.494948  0.745151
2            0.033807  0.100664
```

[23]:
```
pd.crosstab(index = df['Vehicle_Damage'], columns = df['Response'], normalize =␣
↪'index')

# Peeople who has a vehicle damage are more likely to response since they know␣
↪the concequences
# People who don't have a vehicle Damage Are less likely to response
```

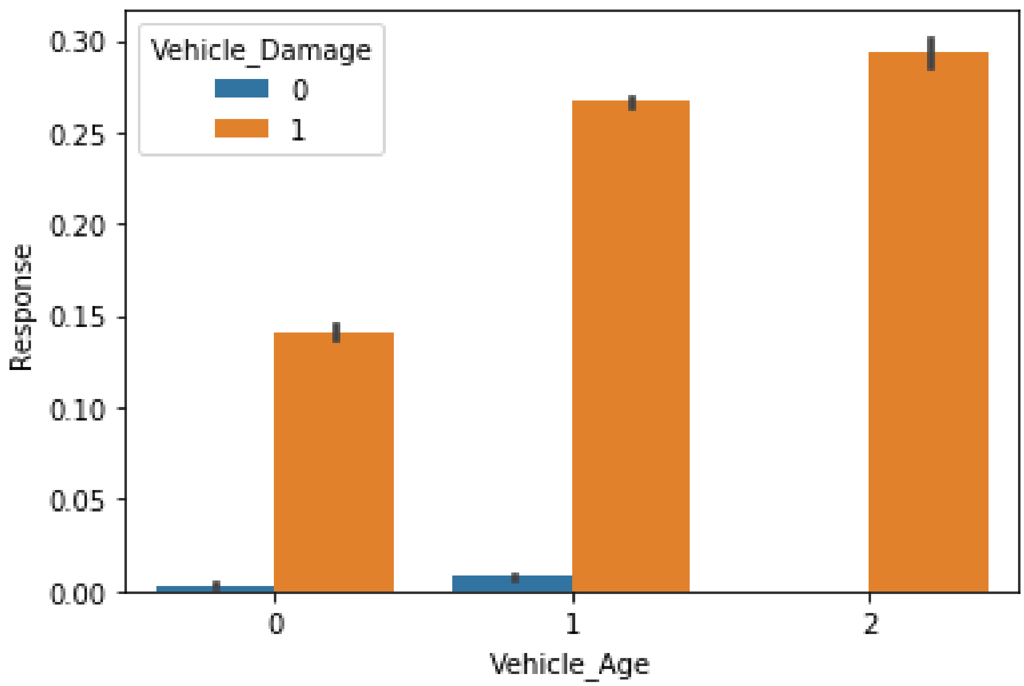
[23]:
```
Response                 0         1
Vehicle_Damage
0                 0.994796  0.005204
1                 0.762345  0.237655
```

[24]:
```
pd.crosstab(index = df['Response'], columns = 'Test', values =␣
↪df['Annual_Premium'], aggfunc = 'median')
```

[24]:
```
col_0         Test
Response
0          31504.0
1          33002.0
```

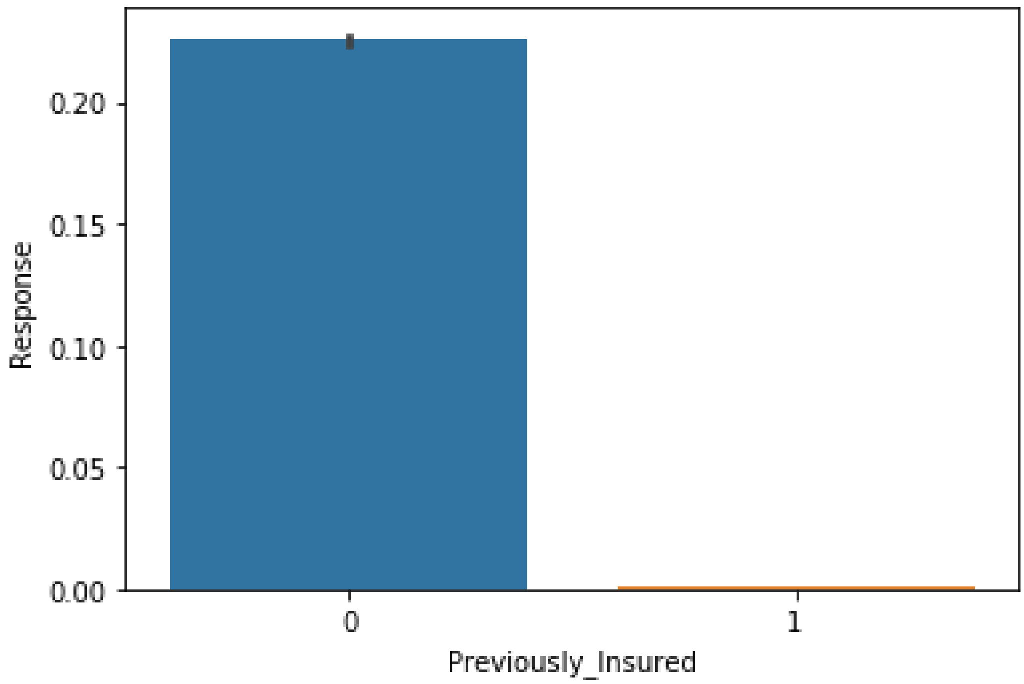
[25]:
```
sns.barplot(x = df['Vehicle_Age'], y= df['Response'], hue =␣
↪df['Vehicle_Damage'])

# Customer who has more than 2 years car age has a more likely to have vehicle␣
↪damage and more likely to response to vehicle insurance
# Customer who has a vehicle damage are more like to response to Insurance as␣
↪well
```

[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6c91e60d0>



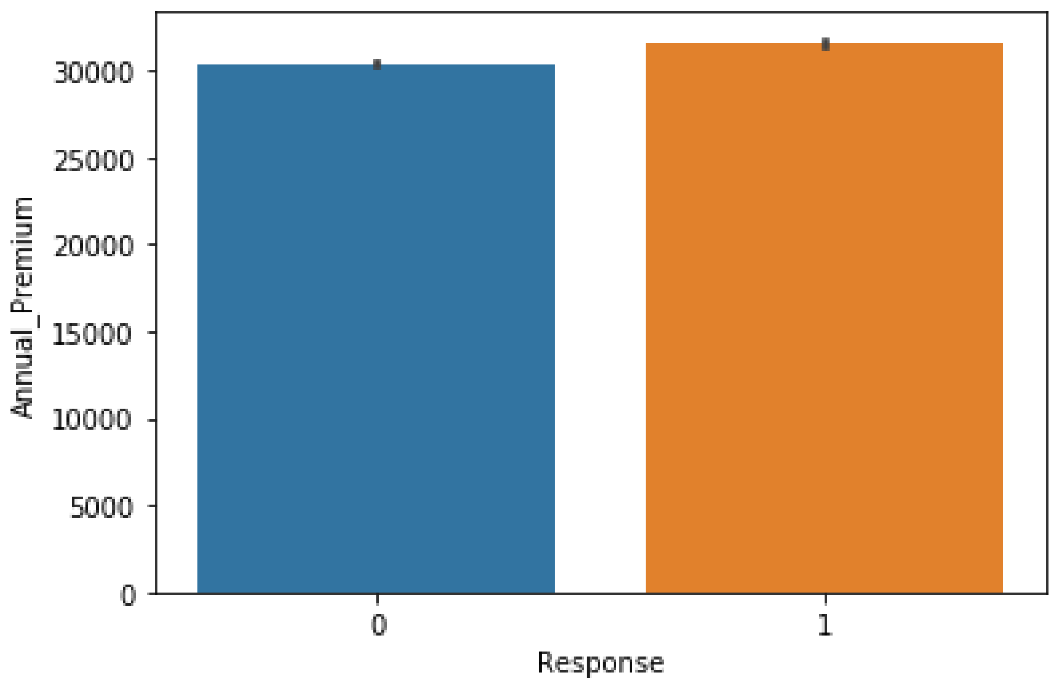
[26]:
```
sns.barplot(x = df['Previously_Insured'], y = df['Response'])

# Customer who was not previously insured are more likely to respond  to the␣
↪vehicle insurance compared to the customer who was previously insured
```

[26]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6c9255b50>



[27]:
```
sns.barplot(x = 'Response', y ='Annual_Premium', data = df)

# People who response have slightly higher annual premium
```

[27]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6c8694410>



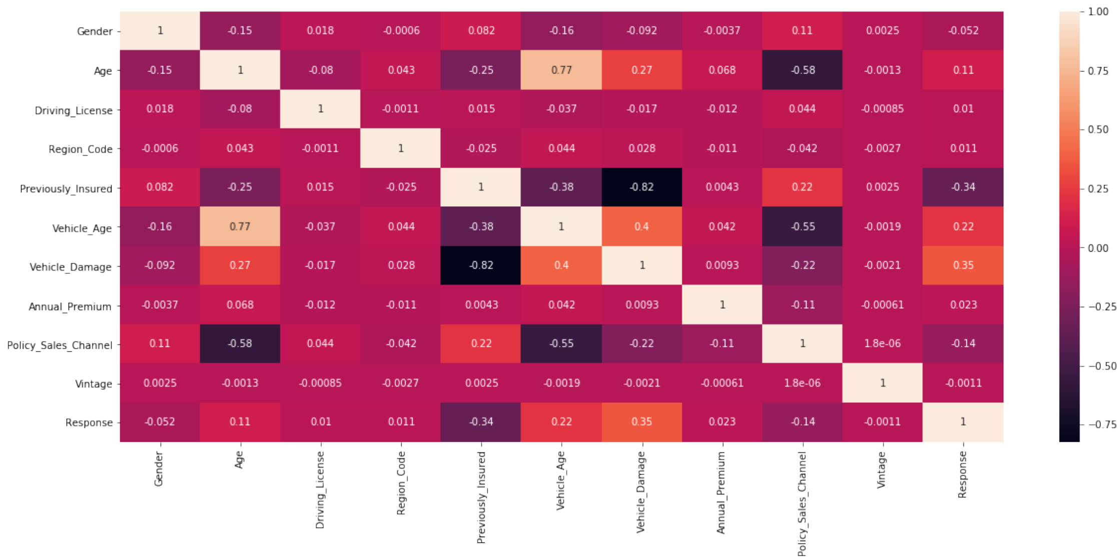
[28]:
```
df['Response'].value_counts(normalize = True)

# This Data is imbalance oversampling is neededm or Smote is required
```

[28]:
```
0    0.877437
1    0.122563
Name: Response, dtype: float64
```

Correlation

[29]:
```
plt.figure(figsize = (20, 8))
sns.heatmap(df.corr(), annot = True)
```

[29]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ff6c8732210>
```



[30]:
```python
correlation = df.corr()
correlation['Response'].sort_values(ascending = False)[1:]

# Sorting Column Correlation
# Policy Sales Channel
```

[30]:
```
Vehicle_Damage          0.354400
Vehicle_Age             0.221874
Age                     0.111147
Annual_Premium          0.022575
Region_Code             0.010570
Driving_License         0.010155
Vintage                -0.001050
Gender                 -0.052440
Policy_Sales_Channel   -0.139042
Previously_Insured     -0.341170
Name: Response, dtype: float64
```

Feature Engineering and Feature Selection

[31]:
```python
X = df.drop(columns = [ 'Driving_License','Response', 'Region_Code',␣
↪'Policy_Sales_Channel', 'Gender', 'Vintage'])
y = df['Response']
```

Model Building

[32]:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42,␣
↪test_size = 0.2)
```

0.1.1 Smote Process Since the data Imbalance

[33]:
```
import imblearn
from imblearn.over_sampling import SMOTE
```

[34]:
```
sm = SMOTE(random_state = 42)
```

[35]:
```
X_train.head()
```

[35]:
```
        Age  Previously_Insured  Vehicle_Age  Vehicle_Damage  Annual_Premium
332803   39                   0            1               1         52906.0
116248   38                   0            1               1         23038.0
255005   22                   1            0               0         45318.0
317474   23                   1            0               0         29132.0
344212   56                   0            2               1          2630.0
```

[36]:
```
X_train_sm, y_train_sm = sm.fit_resample(X_train, y_train)
```

[37]:
```
df['Response'].value_counts()
```

[37]:
```
0    334399
1     46710
Name: Response, dtype: int64
```

[38]:
```
df_smote = pd.concat([X_train_sm, y_train_sm], axis = 1)
```

[39]:
```
df_smote['Response'].value_counts()

# Now the model is balanced we can proceed with model building
```

[39]:
```
1    267700
0    267700
Name: Response, dtype: int64
```

Logistic Regression Model Building

[40]:
```
modelSMOTE = LogisticRegression()
```

[41]:
```
modelSMOTE.fit(X_train_sm, y_train_sm)
```

[41]:
```
LogisticRegression()
```

[42]:
```
y_pred_SMOTE_logreg = modelSMOTE.predict(X_test)
```

[43]:
```
acc_logreg = accuracy_score(y_test, y_pred_SMOTE_logreg)
recall_logreg = recall_score(y_test, y_pred_SMOTE_logreg)
prec_logreg = precision_score(y_test, y_pred_SMOTE_logreg)
f1_logreg = f1_score(y_test, y_pred_SMOTE_logreg)
```

```
print(classification_report(y_test, y_pred_SMOTE_logreg))
```

```
              precision    recall  f1-score   support

           0       0.99      0.60      0.75     66699
           1       0.26      0.97      0.41      9523

    accuracy                           0.65     76222
   macro avg       0.62      0.78      0.58     76222
weighted avg       0.90      0.65      0.71     76222
```

[44]:
```
cm_smote_log_reg = confusion_matrix(y_test, y_pred_SMOTE_logreg, labels = [1,0])
```

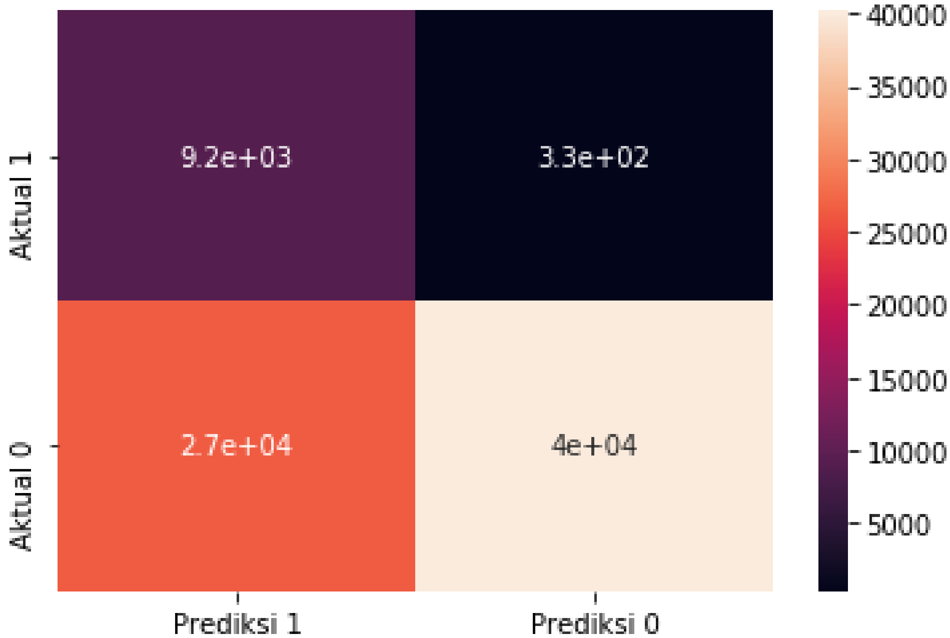
[45]:
```
df_smote_logreg = pd.DataFrame(data = cm_smote_log_reg , index = ["Actual␣
↪1","Actual 0"], columns = ["Predicted 1", "Predicted 0"])
df_smote_logreg
```

[45]:

	Prediksi 1	Prediksi 0
Aktual 1	9194	329
Aktual 0	26521	40178

[46]:
```
sns.heatmap(df_smote_logreg, annot = True)

## Logistic Regression base model has  False Negative amount of 329

# Error Type Interpretation on This Dataset:
  # False Negative -- Actually Interested in Vehicle Insurance, However the␣
↪model predicted that they're not interested
  # False Positive -- Actually not Interested in Vehicle Insurance, However the␣
↪model predicted that they're interested
```

[46]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6c6d0c750>



KNN Classifier

[47]:
```
modelSMOTE_KNN = KNeighborsClassifier()
```

[48]:
```
modelSMOTE_KNN.fit(X_train_sm, y_train_sm)
```

[48]:
```
KNeighborsClassifier()
```

[49]:
```
y_pred_SMOTE_KNN = modelSMOTE_KNN.predict(X_test)
```

[50]:
```
acc_KNN = accuracy_score(y_test, y_pred_SMOTE_KNN)
recall_KNN = recall_score(y_test, y_pred_SMOTE_KNN)
prec_KNN = precision_score(y_test, y_pred_SMOTE_KNN)
f1_KNN = f1_score(y_test, y_pred_SMOTE_KNN)
print(classification_report(y_test, y_pred_SMOTE_KNN))
```

```
              precision    recall  f1-score   support

           0       0.91      0.74      0.82     66699
           1       0.20      0.46      0.28      9523

    accuracy                           0.71     76222
   macro avg       0.55      0.60      0.55     76222
weighted avg       0.82      0.71      0.75     76222
```

[51]:
```
cm_smote_KNN = confusion_matrix(y_test, y_pred_SMOTE_KNN, labels = [1,0])
```

[52]:
```
df_smote_KNN = pd.DataFrame(data = cm_smote_KNN , index = ["Actual 1","Actual␣
↪0"], columns = ["Predicted 1", "Predicted 0"])
df_smote_KNN

## KNN base model has  False Negative amount of 5174

# KNN model has more False Negative means this model doesn't perform well in␣
↪this task
```

[52]:

	Prediksi 1	Prediksi 0
Aktual 1	4349	5174
Aktual 0	17150	49549

[53]:
```
sns.heatmap(df_smote_KNN, annot = True)
```

[53]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6c6c6f650>

Decision Tree

[54]:
```
modelSMOTEDT = DecisionTreeClassifier()
```

[55]:
```
modelSMOTEDT.fit(X_train_sm, y_train_sm)
```

[55]: DecisionTreeClassifier()

[56]:

```
y_pred_SMOTE_DT = modelSMOTEDT.predict(X_test)
```

[57]:

```
acc_DT = accuracy_score(y_test,y_pred_SMOTE_DT)
prec_DT = precision_score(y_test, y_pred_SMOTE_DT)
rec_DT = recall_score(y_test, y_pred_SMOTE_DT)
f1_DT = f1_score(y_test, y_pred_SMOTE_DT)

print(classification_report(y_test, y_pred_SMOTE_DT))
```

```
              precision    recall  f1-score   support

           0       0.93      0.78      0.85     66699
           1       0.28      0.58      0.37      9523

    accuracy                           0.76     76222
   macro avg       0.60      0.68      0.61     76222
weighted avg       0.85      0.76      0.79     76222
```

[58]:

```
cm_DT = confusion_matrix(y_test, y_pred_SMOTE_DT, labels = [1,0])
df_DT = pd.DataFrame(data = cm_DT , index = ["Actual 1","Actual 0"], columns =␣
 ↪["Predicted 1", "Predicted 0"])
df_DT
```

[58]:

	Prediksi 1	Prediksi 0
Aktual 1	5521	4002
Aktual 0	14530	52169

[59]:

```
sns.heatmap(df_DT, annot = True)

## Decision Tree Classifier base model has  False Negative amount of 4005
```

[59]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6c43b5590>

Random Forest Classifier

[60]:
```
modelSMOTERF = RandomForestClassifier()
```

[64]:
```
modelSMOTERF.fit(X_train_sm, y_train_sm)
```

[64]:
```
RandomForestClassifier()
```

[65]:
```
y_pred_SMOTE_RF = modelSMOTERF.predict(X_test)
```

[66]:
```
acc_RF = accuracy_score(y_test,y_pred_SMOTE_RF)
prec_RF = precision_score(y_test, y_pred_SMOTE_RF)
rec_RF = recall_score(y_test, y_pred_SMOTE_RF)
f1_RF = f1_score(y_test, y_pred_SMOTE_RF)

print(classification_report(y_test, y_pred_SMOTE_RF))
```

```
              precision    recall  f1-score   support

           0       0.94      0.76      0.84     66699
           1       0.28      0.63      0.38      9523

    accuracy                           0.75     76222
   macro avg       0.61      0.70      0.61     76222
weighted avg       0.85      0.75      0.78     76222
```

[67]:
```
cm_RF = confusion_matrix(y_test, y_pred_SMOTE_RF, labels = [1,0])
df_RF = pd.DataFrame(data = cm_RF , index = ["Actual 1","Actual 0"], columns =␣
↪["Predicted 1", "Predicted 0"])
df_RF
```

[67]:

	Prediksi 1	Prediksi 0
Aktual 1	6032	3491
Aktual 0	15834	50865

[68]:
```
sns.heatmap(df_RF, annot = True)

## Random Forest Classifier base model has  False Negative amount of 3480
```

[68]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6c004ac90>

[69]:
```
eva_met = {
    "LogisticReg": [acc_logreg,prec_logreg,recall_logreg,f1_logreg],
    "KNN": [acc_KNN, prec_KNN, recall_KNN, f1_KNN],
    "DecisionTree": [acc_DT, prec_DT, rec_DT, f1_DT],
    "RandomForest" : [acc_RF, prec_RF, rec_RF, f1_RF]
    }

eva = pd.DataFrame(data = eva_met, index = ['Accuracy', 'Precision', 'Recall',␣
↪'F1-Score'])
eva
```

```
# From this Evaluation Matrix it shows that logistic Regression has the best␣
↪Recall Score for the base model
```

[69]:
```
           LogisticReg       KNN  DecisionTree  RandomForest
Accuracy      0.647739  0.707119      0.756868      0.746464
Precision     0.257427  0.202288      0.275348      0.275862
Recall        0.965452  0.456684      0.579754      0.633414
F1-Score      0.406472  0.280382      0.373368      0.384338
```

0.1.2 HyperParameter Tuning

Logistic Regression

[70]:
```
logreg_tuning = LogisticRegression()
param_logreg = {'C': [1, 0.5, 0.1, 5, 9],
               'penalty':['l2','l1', 'elasticnet']}
```

[71]:
```
model_logreg_tuned = GridSearchCV(estimator = logreg_tuning, param_grid =␣
↪param_logreg, cv = 3, n_jobs = -1 , verbose = 1, scoring = 'recall')
```

[72]:
```
model_logreg_tuned.fit(X_train_sm, y_train_sm)
```

```
Fitting 3 folds for each of 15 candidates, totalling 45 fits

/home/roshan/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:921: UserWarning: One or more of the
test scores are non-finite: [0.87395932        nan        nan 0.87396679
nan        nan
 0.87397426        nan        nan 0.87395558        nan        nan
 0.87395558        nan        nan]
  category=UserWarning
```

[72]:
```
GridSearchCV(cv=3, estimator=LogisticRegression(), n_jobs=-1,
             param_grid={'C': [1, 0.5, 0.1, 5, 9],
                         'penalty': ['l2', 'l1', 'elasticnet']},
             scoring='recall', verbose=1)
```

[73]:
```
logreg_tuned = model_logreg_tuned.best_estimator_
```

[74]:
```
y_tuned_logreg = logreg_tuned.predict(X_test)
```

[75]:
```
cm_logreg_tuned = confusion_matrix(y_test, y_tuned_logreg, labels = [1,0])
cm_logreg_tuned
```

[75]:
```
array([[ 9194,   329],
       [26523, 40176]])
```

[76]:
```python
acc_logreg_tuned = accuracy_score(y_test, y_tuned_logreg)
prec_logreg_tuned = precision_score(y_test, y_tuned_logreg)
rec_logreg_tuned = recall_score(y_test, y_tuned_logreg)
f1_logreg_tuned = f1_score(y_test,y_tuned_logreg)

print(classification_report(y_test, y_tuned_logreg))
```

```
              precision    recall  f1-score   support

           0       0.99      0.60      0.75     66699
           1       0.26      0.97      0.41      9523

    accuracy                           0.65     76222
   macro avg       0.62      0.78      0.58     76222
weighted avg       0.90      0.65      0.71     76222
```

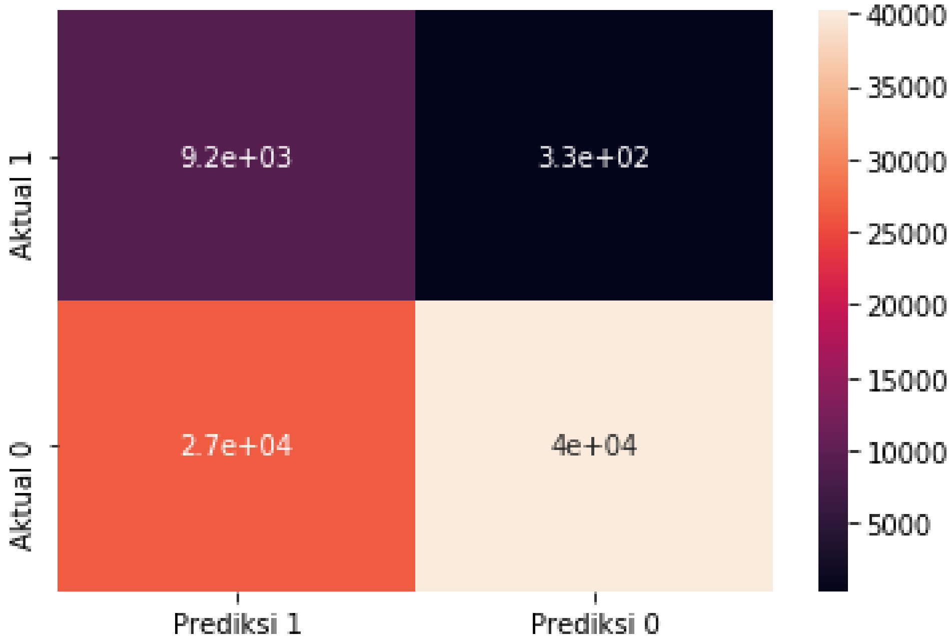
[77]:
```python
df_logreg_tuned = pd.DataFrame(data = cm_logreg_tuned , index = ["Actual␣
↪1","Actual 0"], columns = ["Predicted 1", "Predicted 0"])
df_logreg_tuned
```

[77]:

	Prediksi 1	Prediksi 0
Aktual 1	9194	329
Aktual 0	26523	40176

[78]:
```python
sns.heatmap(df_logreg_tuned, annot = True)

# Logreg Recall score doesn't change after hyper parameter tuning
# Logistic regression base model and tuned model has the same recall score
```

[78]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6ba71bc90>



KNN Tuning

[79]:
```
KNN_tuning = KNeighborsClassifier()
param_KNN = {'n_neighbors':[5,7,9],
              'weights':['uniform','distance'],
              'p':[2, 1]}
```

[80]:
```
model_KNN_tuned = GridSearchCV(estimator = KNN_tuning, param_grid = param_KNN,␣
↪cv = 3, n_jobs = -1 , verbose = 1, scoring = 'recall')
```

[84]:
```
model_KNN_tuned.fit(X_train_sm, y_train_sm)
```

```
Fitting 3 folds for each of 12 candidates, totalling 36 fits
```

[84]:
```
GridSearchCV(cv=3, estimator=KNeighborsClassifier(), n_jobs=-1,
             param_grid={'n_neighbors': [5, 7, 9], 'p': [2, 1],
                         'weights': ['uniform', 'distance']},
             scoring='recall', verbose=1)
```

[85]:
```
KNN_tuned = model_KNN_tuned.best_estimator_
```

[86]:
```
y_tuned_KNN = KNN_tuned.predict(X_test)
```

[87]:
```
cm_KNN_tuned = confusion_matrix(y_test, y_tuned_KNN, labels = [1,0])
cm_KNN_tuned
```

[87]:
```
array([[ 4517,  5006],
       [15184, 51515]])
```

[88]:
```
acc_KNN_tuned = accuracy_score(y_test, y_tuned_KNN)
prec_KNN_tuned = precision_score(y_test, y_tuned_KNN)
rec_KNN_tuned = recall_score(y_test, y_tuned_KNN)
f1_KNN_tuned = f1_score(y_test,y_tuned_KNN)

print(classification_report(y_test, y_tuned_KNN))
```

```
              precision    recall  f1-score   support

           0       0.91      0.77      0.84     66699
           1       0.23      0.47      0.31      9523

    accuracy                           0.74     76222
   macro avg       0.57      0.62      0.57     76222
weighted avg       0.83      0.74      0.77     76222
```

[112]:
```
df_KNN_tuned = pd.DataFrame(data = cm_KNN_tuned , index = ["Actual 1","Actual␣
↪0"], columns = ["Predicted 1", "Predicted 0"])
df_KNN_tuned

# False Negative Goes down after hyperparameter Tuning for KNN
# Recall Score goes up by 0.01 Percent
```

[112]:

	Predicted 1	Predicted 0
Actual 1	4517	5006
Actual 0	15184	51515

[113]:
```
sns.heatmap(df_KNN_tuned, annot = True)
```

[113]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6ba384710>

Decision Tree Classifier Tuning

[91]:
```
DT_tuning = DecisionTreeClassifier()
param_DT =  {
    "max_depth": [None, 4,9,15,20,50],
    "min_samples_leaf": [ 1,4,0.1,2,10],
    "max_features" : [None, 0.2, 0.8, 2.0],
    "min_samples_split": [2,9,15,25]}
```

[92]:
```
model_DT_tuned = GridSearchCV(estimator = DT_tuning, param_grid = param_DT, cv␣
↪= 3, n_jobs = -1 , verbose = 1, scoring = 'recall')
```

[93]:
```
model_DT_tuned.fit(X_train_sm,y_train_sm)
```

```
Fitting 3 folds for each of 480 candidates, totalling 1440 fits
```

```
/home/roshan/anaconda3/lib/python3.7/site-
packages/sklearn/model_selection/_search.py:921: UserWarning: One or more of the
test scores are non-finite: [0.8611917  0.84347411 0.84817341 0.85748607
0.82142331 0.82559589
 0.83852082 0.85000755 0.91487488 0.91487488 0.91487488 0.91487488
 0.80618611 0.83248046 0.84421002 0.85577147 0.84721339 0.84725821
 0.84716483 0.85245433 0.85383645 0.8328017  0.83800157 0.85059777
 0.79498326 0.80192756 0.81223024 0.84416891 0.9630219  0.87818818
 0.9855435  0.96092253 0.78047821 0.81603295 0.82973111 0.84751596
 0.85013456 0.83976104 0.83565195 0.85124403 0.85513641 0.83632805
 0.84215921 0.85194629 0.80824437 0.81167731 0.82714239 0.84465081
```

```
0.94423588 0.89778479 0.91487488 0.91487488 0.78602547 0.81991789
0.83537176 0.84942855 0.84618612 0.84790445 0.84878605 0.8544603
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan 0.90188271 0.90188271 0.90188271 0.90188271
0.90188271 0.90188271 0.90188271 0.90188271 0.91487488 0.91487488
0.91487488 0.91487488 0.90188271 0.90188271 0.90188271 0.90188271
0.90188271 0.90188271 0.90188271 0.90188271 0.9262346  0.88165125
0.89022395 0.8983642  0.85236453 0.8891522  0.90698927 0.9065894
0.88387369 0.92750814 0.9317928  0.9154876  0.89372814 0.93252918
0.90853935 0.95104233 0.88710855 0.9248226  0.88782972 0.87066891
0.90188271 0.89404937 0.89991408 0.90188271 0.90188271 0.89404937
0.90188271 0.90188271 0.89778479 0.93023168 0.94267476 0.91487488
0.91058652 0.90188271 0.88234233 0.88234233 0.90188271 0.90093016
0.89991408 0.90188271        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan 0.9127158  0.91272327
0.91271206 0.91270833 0.9126635  0.91268965 0.91270086 0.91268965
0.91487488 0.91487488 0.91487488 0.91487488 0.91268218 0.91269712
0.91269339 0.91268591 0.9127681  0.9127681  0.9127681  0.91276436
0.90477031 0.9130894  0.9063467  0.9126672  0.90462467 0.90271955
0.91116564 0.89852449 0.90181607 0.8577026  0.92754176 0.96234204
0.91130745 0.88766541 0.89408295 0.90963773 0.89219282 0.88291739
0.9123461  0.88866279 0.90020181 0.91158023 0.90985441 0.91552492
0.91418387 0.91221898 0.90982448 0.90873747 0.89998892 0.91470271
0.89778479 0.94267476 0.90227872 0.9129997  0.90119539 0.90908859
0.91058283 0.91460224 0.91122158 0.91035122        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
0.92263364 0.92218538 0.92177074 0.92143081 0.92169603 0.92162132
0.92163626 0.92116185 0.91487488 0.91487488 0.91487488 0.91487488
0.92166988 0.92189401 0.92156902 0.92129633 0.92031015 0.92032509
0.92032509 0.92030268 0.9176579  0.91004864 0.91165866 0.91128885
0.91298105 0.90960039 0.90696692 0.90574909 0.89037769 0.97236474
0.95025394 0.9171465  0.91490107 0.91517375 0.91611141 0.90784471
0.91277184 0.91158767 0.91500936 0.90989178 0.91748606 0.91892427
0.91676886 0.92018312 0.91369824 0.91657835 0.92069864 0.91353763
0.9024318  0.89778479 0.93523333 0.91487488 0.9182257  0.91312671
0.91859179 0.91205835 0.91464708 0.91805388 0.91789699 0.91517381
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan        nan        nan        nan        nan
       nan        nan 0.9249646  0.92281668 0.92150178 0.92025785
0.91936506 0.91957051 0.9194323  0.91847227 0.91487488 0.91487488
0.91487488 0.91487488 0.92044089 0.92132994 0.92050439 0.91959293
```

```
 0.91533443 0.91534564 0.9153307  0.91551374 0.91428099 0.9108742
 0.91181553 0.91171842 0.90428474 0.90277184 0.90303333 0.90325
 0.96264088 0.99945088 0.99945088 0.99974972 0.90749354 0.90558467
 0.90542408 0.90835647 0.88716114 0.8938253  0.89948084 0.89620859
 0.92199858 0.91819208 0.91507665 0.91394106 0.91299972 0.91419882
 0.91445657 0.91063886 0.94423588 0.94229    0.91487488 0.92740404
 0.91226006 0.91567809 0.91410917 0.91175205 0.90596946 0.90798665
 0.91044088 0.90909236        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan 0.87599933 0.86019432
 0.8622937  0.86828175 0.83657086 0.84002622 0.85038857 0.85973112
 0.91487488 0.91487488 0.91487488 0.91487488 0.82812112 0.84877483
 0.85727315 0.86606658 0.85370199 0.85369078 0.8536609  0.85857685
 0.85768405 0.83644758 0.84264109 0.85227501 0.80459104 0.80429591
 0.82267472 0.83899521 0.8730776  0.88719865 0.88462465 0.84979476
 0.7885208  0.82191639 0.83342931 0.84498326 0.84007854 0.82850218
 0.84115059 0.8527793  0.86316779 0.84287643 0.8478783  0.85665305
 0.81272701 0.81863287 0.831696   0.84656712 0.94229    0.94423588
 0.94229    0.91487488 0.79740016 0.82673523 0.83974233 0.85375055
 0.84893172 0.84839382 0.84853577 0.85500943        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan
        nan        nan        nan        nan        nan        nan]
  category=UserWarning
```

[93]:
```
GridSearchCV(cv=3, estimator=DecisionTreeClassifier(), n_jobs=-1,
             param_grid={'max_depth': [None, 4, 9, 15, 20, 50],
                         'max_features': [None, 0.2, 0.8, 2.0],
                         'min_samples_leaf': [1, 4, 0.1, 2, 10],
                         'min_samples_split': [2, 9, 15, 25]},
             scoring='recall', verbose=1)
```

[94]:
```
DT_tuned = model_DT_tuned.best_estimator_
```

[95]:
```
y_tuned_DT = DT_tuned.predict(X_test)
```

[96]:
```
cm_DT_tuned = confusion_matrix(y_test, y_tuned_DT, labels = [1,0])
cm_DT_tuned
```

[96]:
```
array([[ 9487,    36],
       [31859, 34840]])
```

[97]:
```
acc_DT_tuned = accuracy_score(y_test, y_tuned_DT)
prec_DT_tuned = precision_score(y_test, y_tuned_DT)
rec_DT_tuned = recall_score(y_test, y_tuned_DT)
f1_DT_tuned = f1_score(y_test,y_tuned_DT)
```

```
print(classification_report(y_test, y_tuned_DT))
```

```
              precision    recall  f1-score   support

           0       1.00      0.52      0.69     66699
           1       0.23      1.00      0.37      9523

    accuracy                           0.58     76222
   macro avg       0.61      0.76      0.53     76222
weighted avg       0.90      0.58      0.65     76222
```

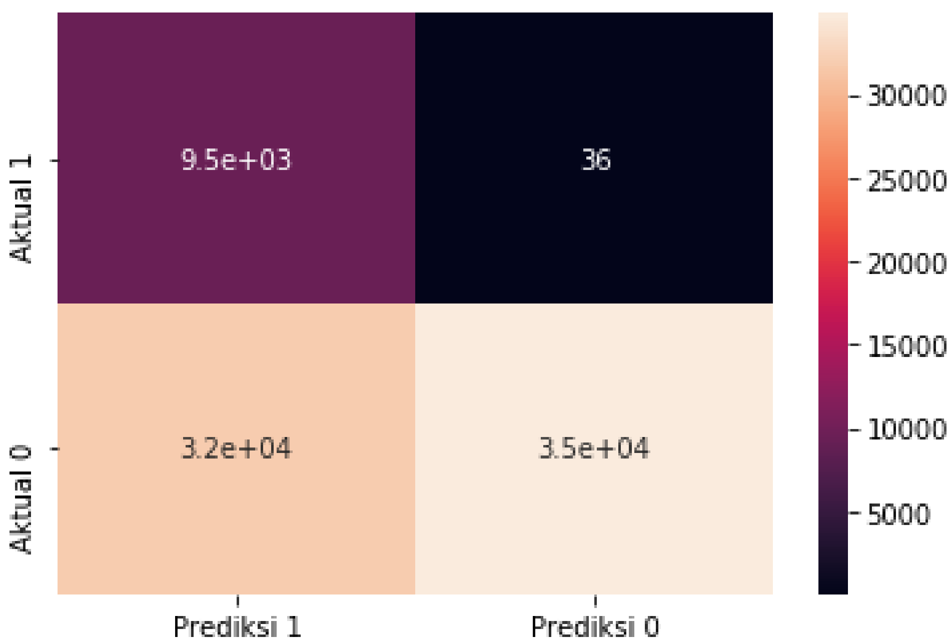
[98]:
```
df_DT_tuned = pd.DataFrame(data = cm_DT_tuned , index = ["Actual 1","Actual␣
↪0"], columns = ["Predicted 1", "Predicted 0"])
df_DT_tuned
```

[98]:

	Prediksi 1	Prediksi 0
Aktual 1	9487	36
Aktual 0	31859	34840

[99]:
```
sns.heatmap(df_DT_tuned, annot = True)

# Recall 1 goes up high with this model however False Positve goes up as well
# Recall Score Goes up by 0.3 after Hyper Param Tuning
```

[99]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6ba55bf10>



[100]:
```
RF_tuning = RandomForestClassifier()
param_DT =  {
    "n_estimators":[100,500,1000],
    "max_depth": [None, 4,6,8],
    "min_samples_leaf": [1,0.06,3,5],
    "min_samples_split" : [2,9,15,25],
    "max_features" : ['auto','sqrt', 'log2'],
    "criterion": ['gini', 'entropy']}
```

[101]:
```
model_RF_tuned = RandomizedSearchCV(estimator=RF_tuning,␣
↪param_distributions=param_DT,scoring = 'recall', verbose = 1, n_jobs = -1,cv␣
↪= 3)
```

[102]:
```
model_RF_tuned.fit(X_train_sm,y_train_sm)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

[102]:
```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_jobs=-1,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [None, 4, 6, 8],
                                        'max_features': ['auto', 'sqrt',
                                                         'log2'],
                                        'min_samples_leaf': [1, 0.06, 3, 5],
                                        'min_samples_split': [2, 9, 15, 25],
                                        'n_estimators': [100, 500, 1000]},
                   scoring='recall', verbose=1)
```

[103]:
```
RF_tuned = model_RF_tuned.best_estimator_

model_RF_tuned.best_estimator_
```

[103]:
```
RandomForestClassifier(max_depth=8, max_features='log2', min_samples_leaf=0.06,
                       min_samples_split=15)
```

[104]:
```
y_tuned_RF = RF_tuned.predict(X_test)
```

[105]:
```
cm_RF_tuned = confusion_matrix(y_test, y_tuned_RF, labels = [1,0])
cm_RF_tuned
```

[105]:
```
array([[ 8631,   892],
       [21428, 45271]])
```

[106]:
```
acc_RF_tuned = accuracy_score(y_test, y_tuned_RF)
prec_RF_tuned = precision_score(y_test, y_tuned_RF)
rec_RF_tuned = recall_score(y_test, y_tuned_RF)
f1_RF_tuned = f1_score(y_test,y_tuned_RF)
```

```
print(classification_report(y_test, y_tuned_RF))
```

```
              precision    recall  f1-score   support

           0       0.98      0.68      0.80     66699
           1       0.29      0.91      0.44      9523

    accuracy                           0.71     76222
   macro avg       0.63      0.79      0.62     76222
weighted avg       0.89      0.71      0.76     76222
```

[107]:
```
df_RF_tuned = pd.DataFrame(data = cm_RF_tuned , index = ["Actual 1","Actual␣
↪0"], columns = ["Predicted 1", "Predicted 0"])
df_RF_tuned
```

[107]:

	Prediksi 1	Prediksi 0
Aktual 1	8631	892
Aktual 0	21428	45271

[108]:
```
sns.heatmap(df_RF_tuned, annot = True)

### Random Forest Classifier Recall Score goes up after hyper parameter tuning
# Recall score goes up by 27 %
```

[108]: <matplotlib.axes._subplots.AxesSubplot at 0x7ff6ba592410>

[109]:
```python
eva_mat_tuned = {
    "LogisticReg": [acc_logreg_tuned, prec_logreg_tuned, rec_logreg_tuned,␣
 ↪f1_logreg_tuned],
    "KNN": [acc_KNN_tuned, prec_KNN_tuned, rec_KNN_tuned, f1_KNN_tuned],
    "DecisionTree": [acc_DT_tuned, prec_DT_tuned, rec_DT_tuned, f1_DT_tuned],
    "RandomForest" : [acc_RF_tuned, prec_RF_tuned, rec_RF_tuned, f1_RF_tuned]
    }

eva_tuned = pd.DataFrame(data = eva_mat_tuned, index = ['Accuracy',␣
 ↪'Precision', 'Recall', 'F1-Score'])
eva_tuned
```

[109]:

	LogisticReg	KNN	DecisionTree	RandomForest
Accuracy	0.647713	0.735116	0.581551	0.707171
Precision	0.257412	0.229278	0.229454	0.287135
Recall	0.965452	0.474325	0.996220	0.906332
F1-Score	0.406454	0.309129	0.372997	0.436107