Table of Content

Step 1: Importing the Relevant Libraries

In [1]:

```
import numpy as np     #Linear Algebra oparations
import pandas as pd    #for manipulation data
import seaborn as sns  #for visualization
import matplotlib.pyplot as plt  #for visualization

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import metrics


import warnings
warnings.filterwarnings('always')
warnings.filterwarnings('ignore')
```

Step 2: Data Inspection

In [2]:

```
train = pd.read_csv("F:\\JOB-A-THON\\train_Df64byy.csv")
test = pd.read_csv("F:\\JOB-A-THON\\test_YCcRUnU.csv")
```

In [3]:

```
train.shape,test.shape
```

Out[3]:

```
((50882, 14), (21805, 13))
```

We have 50882 rows and 14 columns in Train set whereas Test set has 21805 rows and 13 columns.

In [4]:

```
train.head()  #to check top5 rows
```

Out[4]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Low
0	1	C3	3213	Rented	Individual	36	
1	2	C5	1117	Owned	Joint	75	
2	3	C5	3732	Owned	Individual	32	
3	4	C24	4378	Owned	Joint	52	
4	5	C8	2190	Rented	Individual	44	

In [5]:

```
test.head()        #to check top5 rows
```

Out[5]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	I
0	50883	C1	156	Owned	Individual	30	
1	50884	C4	7	Owned	Joint	69	
2	50885	C1	564	Rented	Individual	28	
3	50886	C3	1177	Rented	Individual	23	
4	50887	C1	951	Owned	Individual	75	

In [6]:

```
train.info()  #used to print a concise summary of a DataFrame.including the index dtype
and column dtypes, non-null values and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50882 entries, 0 to 50881
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       50882 non-null  int64  
 1   City_Code                50882 non-null  object 
 2   Region_Code              50882 non-null  int64  
 3   Accomodation_Type        50882 non-null  object 
 4   Reco_Insurance_Type      50882 non-null  object 
 5   Upper_Age                50882 non-null  int64  
 6   Lower_Age                50882 non-null  int64  
 7   Is_Spouse                50882 non-null  object 
 8   Health Indicator         39191 non-null  object 
 9   Holding_Policy_Duration  30631 non-null  object 
 10  Holding_Policy_Type      30631 non-null  float64
 11  Reco_Policy_Cat          50882 non-null  int64  
 12  Reco_Policy_Premium      50882 non-null  float64
 13  Response                 50882 non-null  int64  
dtypes: float64(2), int64(6), object(6)
memory usage: 5.4+ MB
```

In [7]:

```
test.info()  #used to print a concise summary of a DataFrame.including the index dtype
 and column dtypes, non-null values and memory usage.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21805 entries, 0 to 21804
Data columns (total 13 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   ID                       21805 non-null  int64  
 1   City_Code                21805 non-null  object 
 2   Region_Code              21805 non-null  int64  
 3   Accomodation_Type        21805 non-null  object 
 4   Reco_Insurance_Type      21805 non-null  object 
 5   Upper_Age                21805 non-null  int64  
 6   Lower_Age                21805 non-null  int64  
 7   Is_Spouse                21805 non-null  object 
 8   Health Indicator         16778 non-null  object 
 9   Holding_Policy_Duration  13202 non-null  object 
 10  Holding_Policy_Type      13202 non-null  float64
 11  Reco_Policy_Cat          21805 non-null  int64  
 12  Reco_Policy_Premium      21805 non-null  float64
dtypes: float64(2), int64(5), object(6)
memory usage: 2.2+ MB
```

In [8]:

```
train.describe()      #Statistical summary of data frame.
```

Out[8]:

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_F
count	50882.000000	50882.000000	50882.000000	50882.000000	30631.000000	508
mean	25441.500000	1732.788707	44.856275	42.738866	2.439228	
std	14688.512535	1424.081652	17.310271	17.319375	1.025923	
min	1.000000	1.000000	18.000000	16.000000	1.000000	
25%	12721.250000	523.000000	28.000000	27.000000	1.000000	
50%	25441.500000	1391.000000	44.000000	40.000000	3.000000	
75%	38161.750000	2667.000000	59.000000	57.000000	3.000000	
max	50882.000000	6194.000000	75.000000	75.000000	4.000000	

In [9]:

```
test.describe()       #Statistical summary of data frame.
```

Out[9]:

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_F
count	21805.000000	21805.000000	21805.000000	21805.000000	13202.000000	218
mean	61785.000000	1748.737491	44.877734	42.748085	2.440085	
std	6294.705646	1438.358949	17.254898	17.269112	1.037627	
min	50883.000000	1.000000	18.000000	16.000000	1.000000	
25%	56334.000000	535.000000	28.000000	27.000000	1.000000	
50%	61785.000000	1392.000000	44.000000	41.000000	3.000000	
75%	67236.000000	2712.000000	59.000000	57.000000	3.000000	
max	72687.000000	6185.000000	75.000000	75.000000	4.000000	

In [10]:

```
#ratio of null values
train.isnull().sum()/train.shape[0] *100
```

Out[10]:

```
ID                          0.000000
City_Code                   0.000000
Region_Code                 0.000000
Accomodation_Type           0.000000
Reco_Insurance_Type         0.000000
Upper_Age                   0.000000
Lower_Age                   0.000000
Is_Spouse                   0.000000
Health Indicator           22.976691
Holding_Policy_Duration    39.799929
Holding_Policy_Type        39.799929
Reco_Policy_Cat             0.000000
Reco_Policy_Premium         0.000000
Response                    0.000000
dtype: float64
```

We have 22%,39%,39% of missing values in Health Indicator,Holding_Policy_Duration and Holding_Policy_Type.

In [11]:

```
#ratio of null values
test.isnull().sum()/test.shape[0] *100
```

Out[11]:

```
ID                          0.000000
City_Code                   0.000000
Region_Code                 0.000000
Accomodation_Type           0.000000
Reco_Insurance_Type         0.000000
Upper_Age                   0.000000
Lower_Age                   0.000000
Is_Spouse                   0.000000
Health Indicator           23.054345
Holding_Policy_Duration    39.454254
Holding_Policy_Type        39.454254
Reco_Policy_Cat             0.000000
Reco_Policy_Premium         0.000000
dtype: float64
```

We have 23%,39%,39% of missing values in Health Indicator,Holding_Policy_Duration and Holding_Policy_Type.

In [12]:

```
#categorical features
categorical = train.select_dtypes(include =[np.object])
print("Categorical Features in Train Set:",categorical.shape[1])

#numerical features
numerical= train.select_dtypes(include =[np.float64,np.int64])
print("Numerical Features in Train Set:",numerical.shape[1])
```

```
Categorical Features in Train Set: 6
Numerical Features in Train Set: 8
```

In [13]:

```
#categorical features
categorical = test.select_dtypes(include =[np.object])
print("Categorical Features in test Set:",categorical.shape[1])

#numerical features
numerical= test.select_dtypes(include =[np.float64,np.int64])
print("Numerical Features in test Set:",numerical.shape[1])
```

```
Categorical Features in test Set: 6
Numerical Features in test Set: 7
```

Step 3: Data Cleaning

Why missing values treatment is required? Missing data in the training data set can reduce the power / fit of a model or can lead to a biased model because we have not analysed the behavior and relationship with other variables correctly. It can lead to wrong prediction.

In [14]:

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

Out[14]:

```
ID                             0
City_Code                      0
Region_Code                    0
Accomodation_Type              0
Reco_Insurance_Type            0
Upper_Age                      0
Lower_Age                      0
Is_Spouse                      0
Health Indicator           11691
Holding_Policy_Duration    20251
Holding_Policy_Type        20251
Reco_Policy_Cat                0
Reco_Policy_Premium            0
Response                       0
dtype: int64
```

In [15]:

```
test.isnull().sum()
```

Out[15]:

```
ID                          0
City_Code                   0
Region_Code                 0
Accomodation_Type           0
Reco_Insurance_Type         0
Upper_Age                   0
Lower_Age                   0
Is_Spouse                   0
Health Indicator         5027
Holding_Policy_Duration  8603
Holding_Policy_Type      8603
Reco_Policy_Cat             0
Reco_Policy_Premium         0
dtype: int64
```

Health Indicator,Holding_Policy_Duration,Holding_Policy_Type have some missing values in the data

Health Indicator

In [16]:

```
train['Health Indicator'].isnull().sum(),test['Health Indicator'].isnull().sum()
```

Out[16]:

```
(11691, 5027)
```

In [17]:

```
print(train['Health Indicator'].value_counts())
print('*****************************************')
print(test['Health Indicator'].value_counts())
```

```
X1    13010
X2    10332
X3     6762
X4     5743
X5     1727
X6     1280
X7      196
X8       78
X9       63
Name: Health Indicator, dtype: int64
*****************************************
X1    5614
X2    4516
X3    2846
X4    2442
X5     681
X6     514
X7      96
X8      41
X9      28
Name: Health Indicator, dtype: int64
```

Since the Health Indicator is a categorical column, we can impute the missing values by "Mode"(Most Repeated Value) from the column

In [18]:

```
#Imputing with Mode
train['Health Indicator']= train['Health Indicator'].fillna(train['Health Indicator'].m
ode()[0])
test['Health Indicator']= test['Health Indicator'].fillna(test['Health Indicator'].mode
()[0])
```

In [19]:

```
train['Health Indicator'].isnull().sum(),test['Health Indicator'].isnull().sum()
```

Out[19]:

```
(0, 0)
```

Holding_Policy_Duration

Since the Holding_Policy_Duration is a categorical column, we can impute the missing values by "Mode"(Most Repeated Value) from the column

In [20]:

```
#Imputing with Mode
train['Holding_Policy_Duration']= train['Holding_Policy_Duration'].fillna(train['Holdin
g_Policy_Duration'].mode()[0])
test['Holding_Policy_Duration']= test['Holding_Policy_Duration'].fillna(test['Holding_P
olicy_Duration'].mode()[0])
```

In [21]:

```
train['Holding_Policy_Duration'].isnull().sum(),test['Holding_Policy_Duration'].isnull
().sum()
```

Out[21]:

```
(0, 0)
```

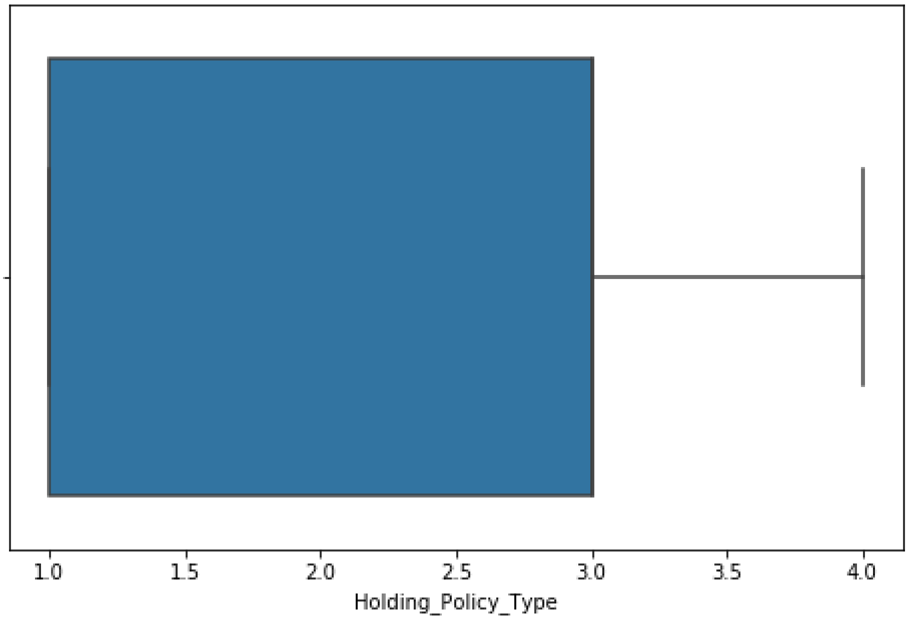
Holding_Policy_Type

In [22]:

```
plt.figure(figsize=(8,5))
sns.boxplot('Holding_Policy_Type',data=train)
```

Out[22]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f65a4108>
```



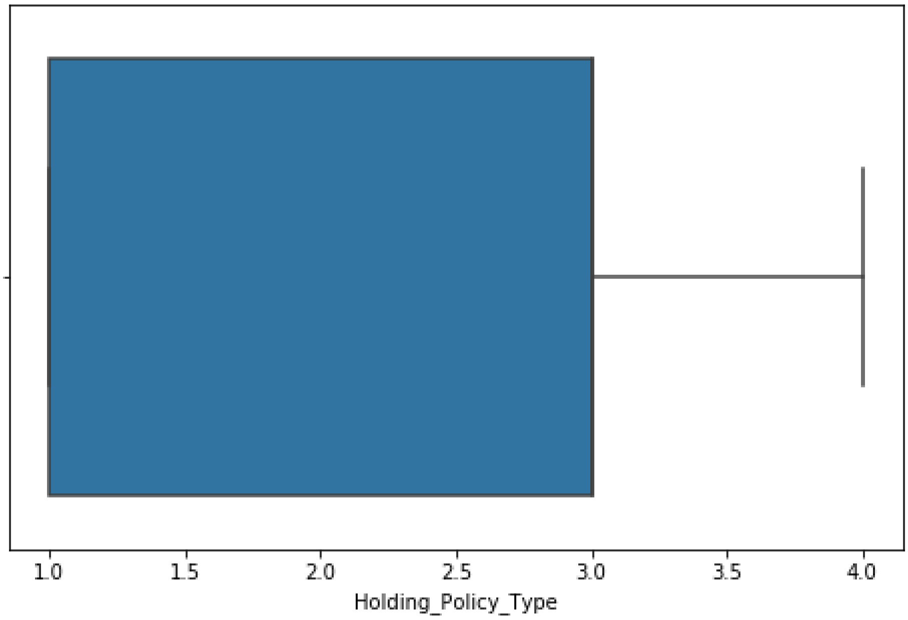
The Box Plots above clearly show no "Outliers" and hence we can impute the missing values with "Mean"

In [23]:

```
plt.figure(figsize=(8,5))
sns.boxplot('Holding_Policy_Type',data=test)
```

Out[23]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f6875e88>
```



The Box Plots above clearly show no "Outliers" and hence we can impute the missing values with "Mean"

In [24]:

```
# Imputing with Mean
train['Holding_Policy_Type']= train['Holding_Policy_Type'].fillna(train['Holding_Policy_Type'].mean())
test['Holding_Policy_Type']= test['Holding_Policy_Type'].fillna(test['Holding_Policy_Type'].mean())
```

In [25]:

```
train['Holding_Policy_Type'].isnull().sum(),test['Holding_Policy_Type'].isnull().sum()
```

Out[25]:

```
(0, 0)
```

We have succesfully imputed the missing values from the column Holding_Policy_Type

Step 4: Exploratory Data Analysis

In [26]:

```
train.columns
```

Out[26]:

```
Index(['ID', 'City_Code', 'Region_Code', 'Accomodation_Type',
       'Reco_Insurance_Type', 'Upper_Age', 'Lower_Age', 'Is_Spouse',
       'Health Indicator', 'Holding_Policy_Duration', 'Holding_Policy_Typ
e',
       'Reco_Policy_Cat', 'Reco_Policy_Premium', 'Response'],
      dtype='object')
```

In [27]:

```
to_drop=['Lower_Age']
train=train.drop(to_drop,axis=1)
train.head()
```

Out[27]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Is_S
0	1	C3	3213	Rented	Individual	36	
1	2	C5	1117	Owned	Joint	75	
2	3	C5	3732	Owned	Individual	32	
3	4	C24	4378	Owned	Joint	52	
4	5	C8	2190	Rented	Individual	44	

In [28]:

```
to_drop=['Lower_Age']
test=test.drop(to_drop,axis=1)
test.head()
```

Out[28]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	I
0	50883	C1	156	Owned	Individual	30	
1	50884	C4	7	Owned	Joint	69	
2	50885	C1	564	Rented	Individual	28	
3	50886	C3	1177	Rented	Individual	23	
4	50887	C1	951	Owned	Individual	75	

In [29]:

```
train.rename(columns = {'Upper_Age':'Age'}, inplace = True)
```

In [30]:

```
test.rename(columns = {'Upper_Age':'Age'}, inplace = True)
```

In [31]:

```
train['Reco_Insurance_Type'].value_counts()
```

Out[31]:

```
Individual    40536
Joint         10346
Name: Reco_Insurance_Type, dtype: int64
```

In [32]:

```
train['Accomodation_Type'].value_counts()
```

Out[32]:

```
Owned     27951
Rented    22931
Name: Accomodation_Type, dtype: int64
```

In [33]:

```
train['Is_Spouse'].value_counts()
```

Out[33]:

```
No     42460
Yes     8422
Name: Is_Spouse, dtype: int64
```

In [34]:

```
train['Health Indicator'].value_counts()
```

Out[34]:

```
X1    24701
X2    10332
X3     6762
X4     5743
X5     1727
X6     1280
X7      196
X8       78
X9       63
Name: Health Indicator, dtype: int64
```

In [35]:

```
train['Holding_Policy_Duration'].value_counts()
```

Out[35]:

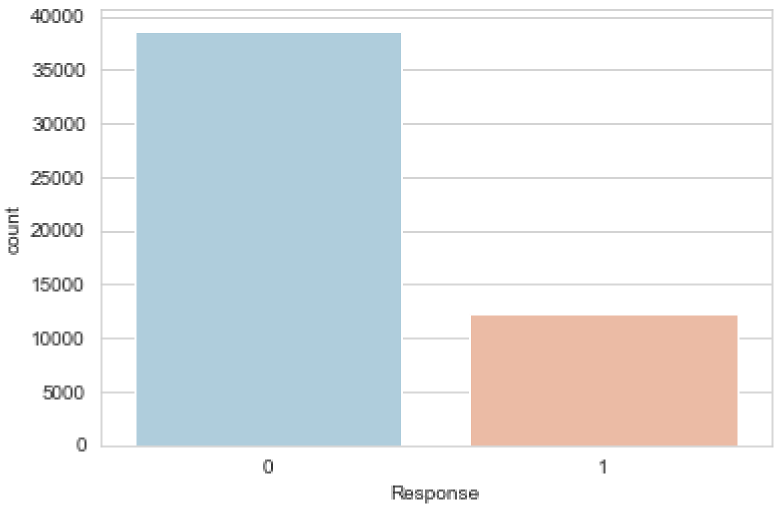
```
1.0     24750
14+      4335
2.0      4260
3.0      3586
4.0      2771
5.0      2362
6.0      1894
7.0      1645
8.0      1316
9.0      1114
10.0      813
11.0      546
12.0      513
13.0      511
14.0      466
Name: Holding_Policy_Duration, dtype: int64
```

In [36]:

```
sns.set_style('whitegrid')
sns.countplot(x='Response',data=train,palette='RdBu_r')
```

Out[36]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f718b948>
```



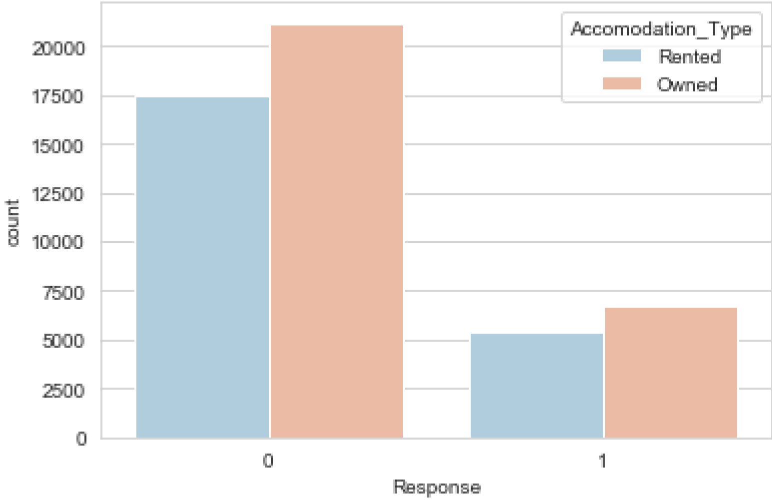
0= Customer did not show interest in the recommended policy, 1=Customer showed interest in the recommended policy, **maximum customer did not show any interest.**

In [37]:

```
sns.set_style('whitegrid')
sns.countplot(x='Response',hue='Accomodation_Type',data=train,palette='RdBu_r')
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f730d408>
```



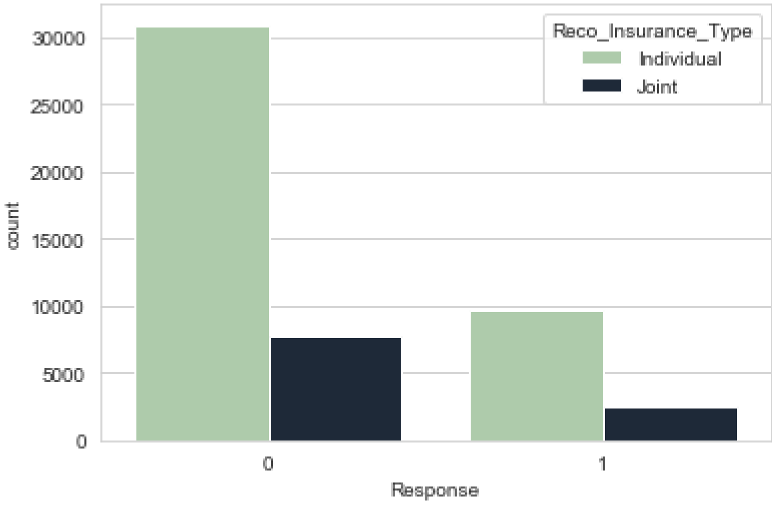
most customers who are not interested in policy most of their accomodation type is owned and little bit are rented and other hand who are interested to take policy little bit of them rented and little bit of them owned

In [38]:

```
sns.set_style('whitegrid')
sns.countplot(x='Response',hue='Reco_Insurance_Type',data=train,palette="ch:r=-.5,l=.7
5")
```

Out[38]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f7a41b08>
```



most customers who are not interested in policy most of their Reco_Insurance_Type is individual and little bit are joint and other hand who are interested to take policy most of them individual and little bit of them joint

In [39]:

```
train.Age.unique()
```

Out[39]:

```
array([36, 75, 32, 52, 44, 28, 59, 21, 66, 20, 27, 34, 43, 55, 23, 18, 22,
       25, 24, 40, 26, 56, 35, 63, 49, 64, 67, 42, 71, 57, 73, 31, 19, 48,
       65, 54, 33, 30, 69, 68, 37, 29, 62, 58, 38, 39, 60, 41, 45, 51, 46,
       70, 61, 74, 53, 72, 50, 47], dtype=int64)
```

In [40]:

```
def age_buckets(x):
    if x < 30:
        return 'Adult'
    elif x < 50:
        return 'Senior'
    elif x < 80:
        return 'Senior-citizen'
    else: return 'other'
```

In [41]:

```
train['agerange'] = train.Age.apply(age_buckets)
to_drop=['Age']
train=train.drop(to_drop,axis=1)
train.head()
```

Out[41]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Is_Spouse	He Indic
0	1	C3	3213	Rented	Individual	No	
1	2	C5	1117	Owned	Joint	No	
2	3	C5	3732	Owned	Individual	No	
3	4	C24	4378	Owned	Joint	No	
4	5	C8	2190	Rented	Individual	No	

In [42]:

```
test['agerange'] = test.Age.apply(age_buckets)
to_drop=['Age']
test=test.drop(to_drop,axis=1)
test.head()
```

Out[42]:

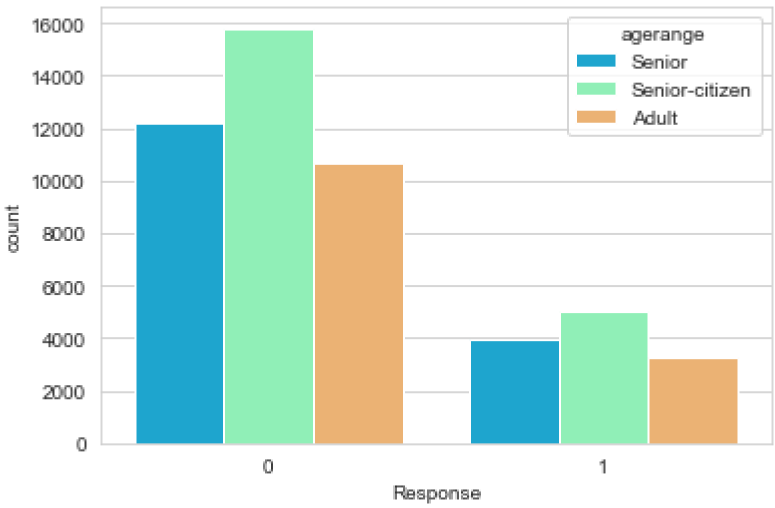
	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Is_Spouse	I
0	50883	C1	156	Owned	Individual	No	
1	50884	C4	7	Owned	Joint	Yes	
2	50885	C1	564	Rented	Individual	No	
3	50886	C3	1177	Rented	Individual	No	
4	50887	C1	951	Owned	Individual	No	

In [43]:

```
sns.set_style('whitegrid')
sns.countplot(x='Response',hue='agerange',data=train,palette="rainbow")
```

Out[43]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f7a9db08>
```



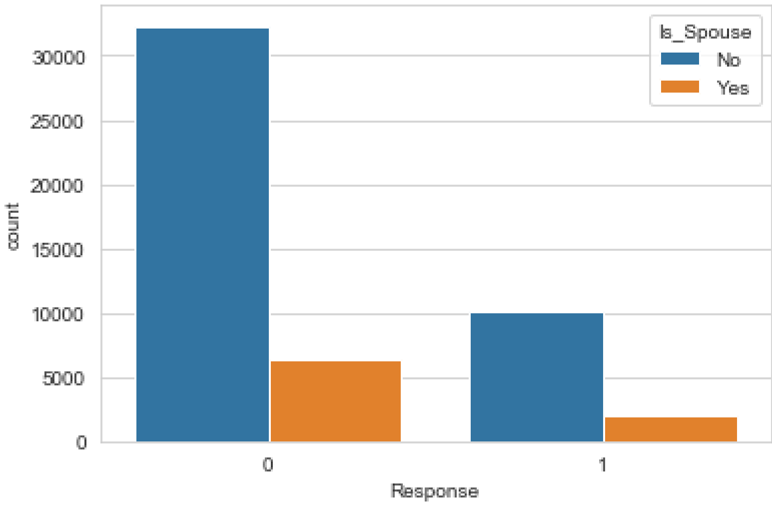
most customers who are not interested in policy most of their agerange in senior-citizen and little bit are senior and adult and other hand who are interested to take policy agerange is little bit same, not very high

In [44]:

```
sns.countplot(x="Response", hue="Is_Spouse", data=train)
```

Out[44]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e1f8dafc88>
```



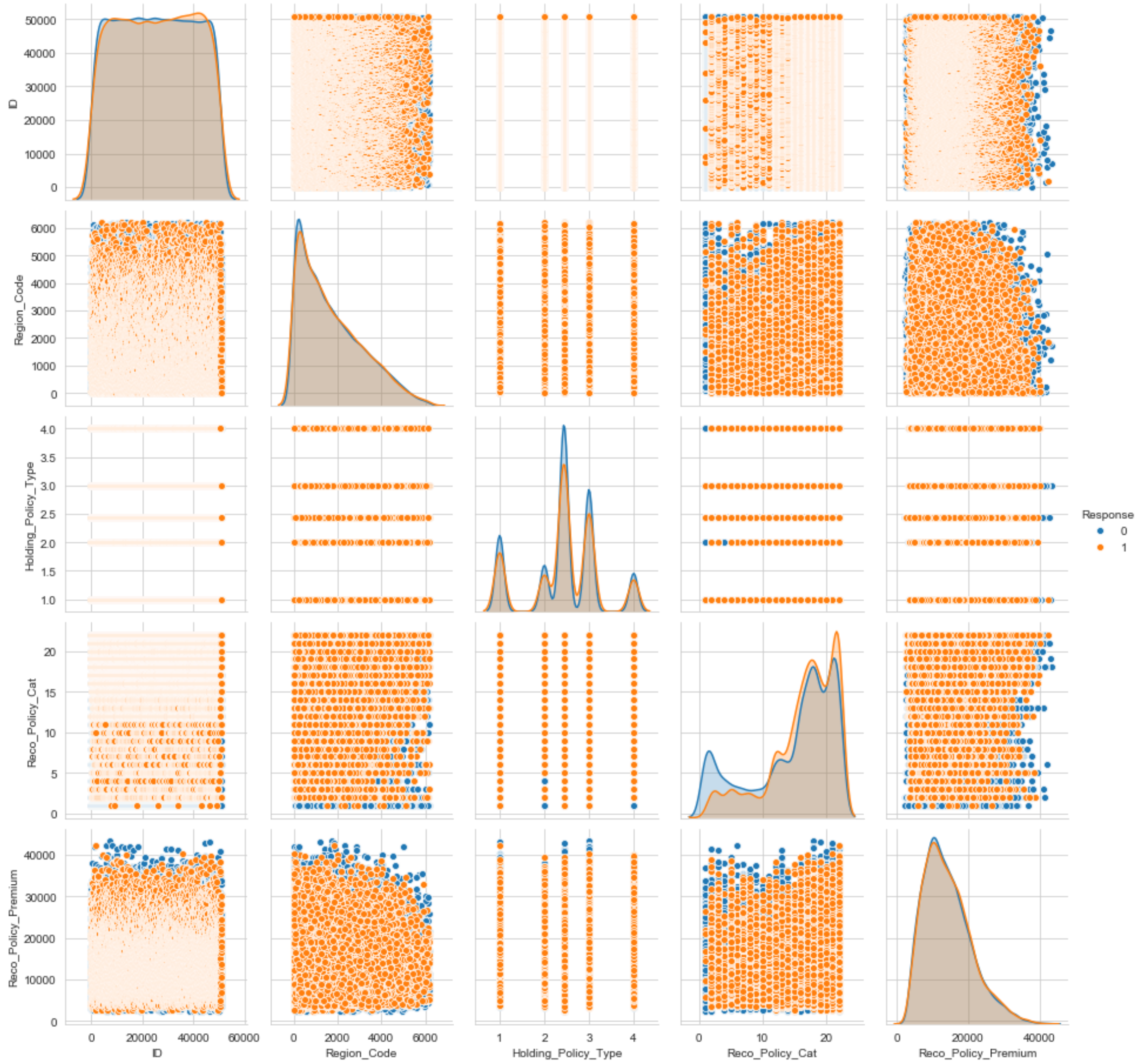
most customers who are not interested in policy they are unmarried and other hand who are interested to take policy they are unmmaried

In [45]:

```
sns.pairplot(train,hue='Response')
```

Out[45]:

```
<seaborn.axisgrid.PairGrid at 0x1e1f8e3a308>
```


ID
Region_Code
Holding_Policy_Type
Reco_Policy_Cat
Reco_Policy_Premium
Response
0
1

In [46]:

```
train.corr()
```

Out[46]:

	ID	Region_Code	Holding_Policy_Type	Reco_Policy_Cat	Reco_
ID	1.000000	-0.000465	0.005153	-0.002235	
Region_Code	-0.000465	1.000000	0.009052	-0.065120	
Holding_Policy_Type	0.005153	0.009052	1.000000	0.061613	
Reco_Policy_Cat	-0.002235	-0.065120	0.061613	1.000000	
Reco_Policy_Premium	-0.002350	-0.010797	0.091535	0.060989	
Response	0.005159	0.001121	0.007221	0.114321	

Step 5: Building Model

In [47]:

```
train.head()
```

Out[47]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Is_Spouse	He Indic
0	1	C3	3213	Rented	Individual	No	
1	2	C5	1117	Owned	Joint	No	
2	3	C5	3732	Owned	Individual	No	
3	4	C24	4378	Owned	Joint	No	
4	5	C8	2190	Rented	Individual	No	

In [48]:

```
# Labelencoding
le = LabelEncoder()
var_mod = train.select_dtypes(include='object').columns
for i in var_mod:
    train[i] = le.fit_transform(train[i])

for i in var_mod:
    test[i] = le.fit_transform(test[i])
```

Encoding the required columns from training and test dataset

In [49]:

```
train.columns
```

Out[49]:

```
Index(['ID', 'City_Code', 'Region_Code', 'Accomodation_Type',
       'Reco_Insurance_Type', 'Is_Spouse', 'Health Indicator',
       'Holding_Policy_Duration', 'Holding_Policy_Type', 'Reco_Policy_Ca
t',
       'Reco_Policy_Premium', 'Response', 'agerange'],
      dtype='object')
```

In [50]:

```
train.head()
```

Out[50]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Is_Spouse	He Indic
0	1	22	3213	1	0	0	
1	2	31	1117	0	1	0	
2	3	31	3732	0	0	0	
3	4	16	4378	0	1	0	
4	5	34	2190	1	0	0	

In [51]:

```
# Seperate Features and Target
X= train.drop(columns = ['Response'], axis=1)
y= train['Response']
```

In [52]:

```
# 20% data as validation set
X_train,X_valid,y_train,y_valid = train_test_split(X,y,test_size=0.2,random_state=22)
```

In [53]:

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_valid = sc.transform(X_valid)
```

In [54]:

```
from sklearn.linear_model import LogisticRegression
```

In [55]:

```
logmodel = LogisticRegression()
logmodel.fit(X_train,y_train)
```

Out[55]:

```
LogisticRegression()
```

In [56]:

```
predictions = logmodel.predict(X_valid)
```

In [57]:

```
predictions
```

Out[57]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [58]:

```
from sklearn.metrics import classification_report,confusion_matrix
```

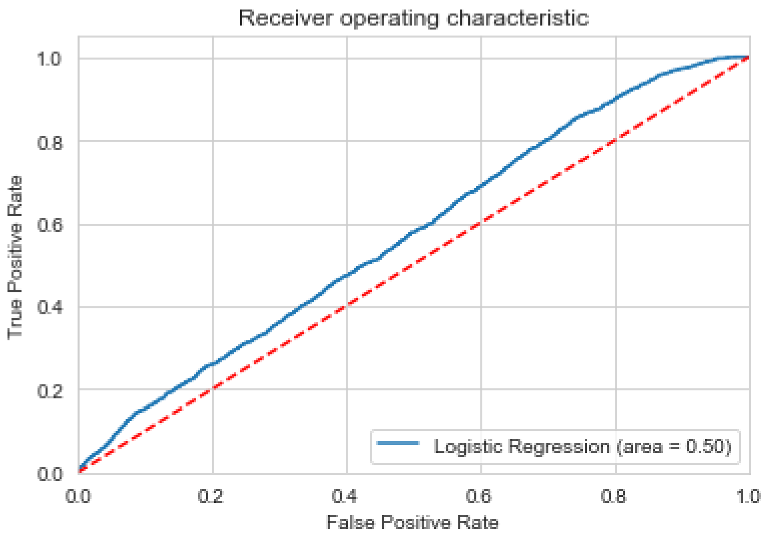
In [59]:

```
print(confusion_matrix(y_valid,logmodel.predict(X_valid)))
```

```
[[7737    0]
 [2440    0]]
```

In [60]:

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_valid, predictions)
fpr, tpr, thresholds = roc_curve(y_valid, logmodel.predict_proba(X_valid)[:,1])
plt.figure()
plt.plot(fpr, tpr, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
#plt.savefig('Log_ROC')
plt.show()
```



In [61]:

```
roc_auc_score(y_valid, logmodel.predict(X_valid))
```

Out[61]:

```
0.5
```

In [62]:

```
submission = pd.read_csv("F:\JOB-A-THON\sample_submission_QrCyCoT.csv")
logmodel = LogisticRegression()
logmodel.fit(X, y)
final_predictions = logmodel.predict(test)
submission['Response'] = final_predictions
submission.to_csv('my_submission.csv', index=False)
```

In [63]:

```
df = pd.read_csv('my_submission.csv')
print(df)
```

```
          ID  Response
0      50883         0
1      50884         0
2      50885         0
3      50886         0
4      50887         0
...      ...       ...
21800  72683         0
21801  72684         0
21802  72685         0
21803  72686         0
21804  72687         0

[21805 rows x 2 columns]
```

In [ ]: