Building the best classifier based on below problem statement:

Problem: To predict whether a person will be interested in the company proposed Health plan/policy given the information about:

- **Demographics (city, age, region etc.)**
- **Information regarding holding policies of the customer**
- **Recommended Policy Information**

This Solution comprises of the following sub divisions:

- **EDA (Exploratory Data Analysis)**
- **Feature Engineering**
- **Feature Selection**
- **Scaling Dataset**
- **Performing all these steps above for Test Data**
- **Multiple Model Training**
- **Predictions from Multiple Models**
- **Finding best solution based on data and model estimation**

1. Exploratory Data Analysis:

Import required libraries:

In [1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:
```
pd.set_option('display.max_columns',None)
```

Load data for visualization:

In [3]:
```
train_df = pd.read_csv('train_data.csv')
train_df.head()
```

Out[3]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_S
0	1	C3	3213	Rented	Individual	36	36	
1	2	C5	1117	Owned	Joint	75	22	
2	3	C5	3732	Owned	Individual	32	32	
3	4	C24	4378	Owned	Joint	52	48	

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Sp
4	5	C8	2190	Rented	Individual	44	44	

In [4]:
```
train_df.columns
```

Out[4]:
```
Index(['ID', 'City_Code', 'Region_Code', 'Accomodation_Type',
       'Reco_Insurance_Type', 'Upper_Age', 'Lower_Age', 'Is_Spouse',
       'Health Indicator', 'Holding_Policy_Duration', 'Holding_Policy_Type',
       'Reco_Policy_Cat', 'Reco_Policy_Premium', 'Response'],
      dtype='object')
```

In [5]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50882 entries, 0 to 50881
Data columns (total 14 columns):
ID                          50882 non-null int64
City_Code                   50882 non-null object
Region_Code                 50882 non-null int64
Accomodation_Type           50882 non-null object
Reco_Insurance_Type         50882 non-null object
Upper_Age                   50882 non-null int64
Lower_Age                   50882 non-null int64
Is_Spouse                   50882 non-null object
Health Indicator            39191 non-null object
Holding_Policy_Duration     30631 non-null object
Holding_Policy_Type         30631 non-null float64
Reco_Policy_Cat             50882 non-null int64
Reco_Policy_Premium         50882 non-null float64
Response                    50882 non-null int64
dtypes: float64(2), int64(6), object(6)
memory usage: 5.4+ MB
```

From the above information, we could see that there are 3 different categories of data present:

- **int64 : Integer values for the 6 columns ID, Region_code, Upper_age, Lower_age, Reco_Policy_cat, Response. ID filed would be removed from further calculations as it does not contribute to the predictions**
- **float64 : Floating values for the 2 columns Holding_Policy_Type, Reco_Policy_Premium**
- **object : Categorical values for 6 columns City_Code, Accomodation_Type, Reco_Insurance_Type, Is_Spouse, Health Indicator, Holding_Policy_Duration.**

In [6]:
```
train_df.describe()
```

Out[6]:

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat
count	50882.000000	50882.000000	50882.000000	50882.000000	30631.000000	50882.000000
mean	25441.500000	1732.788707	44.856275	42.738866	2.439228	15.115188
std	14688.512535	1424.081652	17.310271	17.319375	1.025923	6.340663
min	1.000000	1.000000	18.000000	16.000000	1.000000	1.000000
25%	12721.250000	523.000000	28.000000	27.000000	1.000000	12.000000

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat
50%	25441.500000	1391.000000	44.000000	40.000000	3.000000	17.000000
75%	38161.750000	2667.000000	59.000000	57.000000	3.000000	20.000000
max	50882.000000	6194.000000	75.000000	75.000000	4.000000	22.000000

Categorical Variables:

In [7]:
```
cat_vars = [var for var in train_df.columns if train_df[var].dtypes=='O']
for var in cat_vars:
    print(var)
```

```
City_Code
Accomodation_Type
Reco_Insurance_Type
Is_Spouse
Health Indicator
Holding_Policy_Duration
```

Missing Values:

In [8]:
```
mis_vars = [var for var in train_df.columns if train_df[var].isnull().sum()>0]
train_df[mis_vars].isnull().sum()
```

Out[8]:
```
Health Indicator           11691
Holding_Policy_Duration    20251
Holding_Policy_Type        20251
dtype: int64
```

2. Feature Engineering:

Implementing Mean & Mode Imputation:

In [9]:
```
train_df['Health Indicator'].mode()[0]
```

Out[9]:
```
'X1'
```

In [10]:
```
train_df['Health Indicator'] = train_df['Health Indicator'].fillna(train_df['Health Ind
train_df['Health Indicator'].isnull().sum()
```

Out[10]:
```
0
```

In [11]:
```
train_df.isnull().sum()
```

Out[11]:
```
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           0
Holding_Policy_Duration    20251
Holding_Policy_Type        20251
```

```
Reco_Policy_Cat               0
Reco_Policy_Premium           0
Response                      0
dtype: int64
```

In [12]:
```
train_df['Holding_Policy_Duration'].head()
```

Out[12]:
```
0    14+
1    NaN
2    1.0
3    14+
4    3.0
Name: Holding_Policy_Duration, dtype: object
```

In [13]:
```
train_df['Holding_Policy_Duration'].mode()[0]
```

Out[13]:
```
'1.0'
```

In [14]:
```
train_df['Holding_Policy_Duration'] = train_df['Holding_Policy_Duration'].fillna(train_
train_df['Holding_Policy_Duration'].isnull().sum()
```

Out[14]:
```
0
```

In [15]:
```
np.around(train_df['Holding_Policy_Type'].mean())
```

Out[15]:
```
2.0
```

In [16]:
```
train_df['Holding_Policy_Type'] = train_df['Holding_Policy_Type'].fillna(np.around(trai
train_df['Holding_Policy_Type'].isnull().sum()
```

Out[16]:
```
0
```

In [17]:
```
# Converting Holding_Policy_Duration to float data
train_df.Holding_Policy_Duration = train_df.Holding_Policy_Duration.str.replace('[+]',
train_df['Holding_Policy_Duration'].head()
```

Out[17]:
```
0     14
1    1.0
2    1.0
3     14
4    3.0
Name: Holding_Policy_Duration, dtype: object
```

In [18]:
```
train_df['Holding_Policy_Duration'] = train_df['Holding_Policy_Duration'].astype(float)
```

In [19]:
```
[var for var in train_df.columns if train_df[var].isnull().sum()>0]
```

Out[19]:
```
[]
```

Numerical Variables:

In [20]:
```
num_vars = [var for var in train_df.columns if train_df[var].dtypes!='O']

for var in num_vars:
    print(var)
```

```
ID
Region_Code
```

```
Upper_Age
Lower_Age
Holding_Policy_Duration
Holding_Policy_Type
Reco_Policy_Cat
Reco_Policy_Premium
Response
```

In [21]:
```
train_df[num_vars].head()
```

Out[21]:

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Duration	Holding_Policy_Type	Reco_Policy_
0	1	3213	36	36	14.0	3.0	
1	2	1117	75	22	1.0	2.0	
2	3	3732	32	32	1.0	1.0	
3	4	4378	52	48	14.0	3.0	
4	5	2190	44	44	3.0	1.0	

In [22]:
```
final_cat_vars = [var for var in train_df.columns if train_df[var].dtypes=='O']
for var in final_cat_vars:
    print(train_df[var].value_counts())
```

```
C1     8973
C2     7747
C3     4889
C4     3671
C9     2185
C6     1950
C7     1910
C8     1806
C10    1611
C5     1364
C15    1186
C17    1159
C11    1147
C16    1135
C13    1005
C20     926
C19     899
C12     868
C18     797
C14     746
C21     679
C23     587
C24     553
C22     516
C26     499
C29     387
C25     366
C27     295
C33     286
C28     285
C32     160
C34     130
C30      58
C35      56
C36      36
C31      15
Name: City_Code, dtype: int64
```

```
Owned     27951
Rented    22931
Name: Accomodation_Type, dtype: int64
Individual    40536
Joint         10346
Name: Reco_Insurance_Type, dtype: int64
No     42460
Yes     8422
Name: Is_Spouse, dtype: int64
X1    24701
X2    10332
X3     6762
X4     5743
X5     1727
X6     1280
X7      196
X8       78
X9       63
Name: Health Indicator, dtype: int64
```

Label Encoding categorical variables:

In [23]:
```
from sklearn.preprocessing import LabelEncoder
```

In [24]:
```
final_cat_vars
```

Out[24]:
```
['City_Code',
 'Accomodation_Type',
 'Reco_Insurance_Type',
 'Is_Spouse',
 'Health Indicator']
```

In [25]:
```
city_code_le = LabelEncoder()
city_code_labels = city_code_le.fit_transform(train_df['City_Code'])
city_code_mappings = {index: label for index, label in
                      enumerate(city_code_le.classes_)}
print(city_code_mappings)

Accomodation_Type_le = LabelEncoder()
Accomodation_Type_labels = Accomodation_Type_le.fit_transform(train_df['Accomodation_Ty
Accomodation_Type_mappings = {index: label for index, label in
                      enumerate(Accomodation_Type_le.classes_)}
print(Accomodation_Type_mappings)

Reco_Insurance_Type_le = LabelEncoder()
Reco_Insurance_Type_labels = Reco_Insurance_Type_le.fit_transform(train_df['Reco_Insura
Reco_Insurance_Type_mappings = {index: label for index, label in
                      enumerate(Reco_Insurance_Type_le.classes_)}
print(Reco_Insurance_Type_mappings)

Is_Spouse_le = LabelEncoder()
Is_Spouse_labels = Is_Spouse_le.fit_transform(train_df['Is_Spouse'])
Is_Spouse_mappings = {index: label for index, label in
                      enumerate(Is_Spouse_le.classes_)}
print(Is_Spouse_mappings)

Health_Indicator_le = LabelEncoder()
Health_Indicator_labels = Health_Indicator_le.fit_transform(train_df['Health Indicator'
Health_Indicator_mappings = {index: label for index, label in
                      enumerate(Health_Indicator_le.classes_)}
print(Health_Indicator_mappings)
```

```
{0: 'C1', 1: 'C10', 2: 'C11', 3: 'C12', 4: 'C13', 5: 'C14', 6: 'C15', 7: 'C16', 8: 'C1
7', 9: 'C18', 10: 'C19', 11: 'C2', 12: 'C20', 13: 'C21', 14: 'C22', 15: 'C23', 16: 'C2
4', 17: 'C25', 18: 'C26', 19: 'C27', 20: 'C28', 21: 'C29', 22: 'C3', 23: 'C30', 24: 'C3
1', 25: 'C32', 26: 'C33', 27: 'C34', 28: 'C35', 29: 'C36', 30: 'C4', 31: 'C5', 32: 'C6',
33: 'C7', 34: 'C8', 35: 'C9'}
{0: 'Owned', 1: 'Rented'}
{0: 'Individual', 1: 'Joint'}
{0: 'No', 1: 'Yes'}
{0: 'X1', 1: 'X2', 2: 'X3', 3: 'X4', 4: 'X5', 5: 'X6', 6: 'X7', 7: 'X8', 8: 'X9'}
```

In [26]:
```
train_df['City_Code_Labels'] = city_code_labels
train_df['Accomodation_Type_Labels'] = Accomodation_Type_labels
train_df['Reco_Insurance_Type_Labels'] = Reco_Insurance_Type_labels
train_df['Is_Spouse_Labels'] = Is_Spouse_labels
train_df['Health_Indicator_Labels'] = Health_Indicator_labels
train_df.head()
```

Out[26]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_S
0	1	C3	3213	Rented	Individual	36	36	
1	2	C5	1117	Owned	Joint	75	22	
2	3	C5	3732	Owned	Individual	32	32	
3	4	C24	4378	Owned	Joint	52	48	
4	5	C8	2190	Rented	Individual	44	44	

In [27]:
```
train_df.columns
```

Out[27]:
```
Index(['ID', 'City_Code', 'Region_Code', 'Accomodation_Type',
       'Reco_Insurance_Type', 'Upper_Age', 'Lower_Age', 'Is_Spouse',
       'Health Indicator', 'Holding_Policy_Duration', 'Holding_Policy_Type',
       'Reco_Policy_Cat', 'Reco_Policy_Premium', 'Response',
       'City_Code_Labels', 'Accomodation_Type_Labels',
       'Reco_Insurance_Type_Labels', 'Is_Spouse_Labels',
       'Health_Indicator_Labels'],
      dtype='object')
```

3. Feature Selection:

In [28]:
```
train_df_1 = train_df.copy()
train_df_2 = train_df.copy()
```

Dropping categorical variables:

In [29]:
```
train_df_1.drop(columns=['ID','City_Code','Accomodation_Type','Reco_Insurance_Type','Is
train_df_1.head()
```

Out[29]:

	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Duration	Holding_Policy_Type	Reco_Policy_Cat
0	3213	36	36	14.0	3.0	22
1	1117	75	22	1.0	2.0	22
2	3732	32	32	1.0	1.0	19

	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Duration	Holding_Policy_Type	Reco_Policy_Cat
3	4378	52	48	14.0	3.0	19
4	2190	44	44	3.0	1.0	16

In [30]:
```
train_df_2.drop(columns=['ID','City_Code','Accomodation_Type','Reco_Insurance_Type','Is
train_df_2.head()
```

Out[30]:

	Upper_Age	Lower_Age	Holding_Policy_Duration	Holding_Policy_Type	Reco_Policy_Premium	Respons
0	36	36	14.0	3.0	11628.0	
1	75	22	1.0	2.0	30510.0	
2	32	32	1.0	1.0	7450.0	
3	52	48	14.0	3.0	17780.0	
4	44	44	3.0	1.0	10404.0	

4. Scaling Dataset:

In [31]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [32]:
```
scaler = MinMaxScaler()
scaler2 = MinMaxScaler()
```

In [33]:
```
X_train_1 = train_df_1.drop(['Response'],axis=1).values
y_train_1 = train_df_1['Response'].values
X_train_1.shape, y_train_1.shape
```

Out[33]: ((50882, 12), (50882,))

In [34]:
```
X_train_2 = train_df_2.drop(['Response'],axis=1).values
y_train_2 = train_df_2['Response'].values
X_train_2.shape, y_train_2.shape
```

Out[34]: ((50882, 9), (50882,))

In [35]:
```
X_train_1 = scaler.fit_transform(X_train_1)
X_train_2 = scaler2.fit_transform(X_train_2)
```

5. Repeating all operations for Test Data:

In [36]:
```
# Loading dataset
test_df = pd.read_csv('test_data.csv')
test_df.head()
```

Out[36]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age
0	50883	C1	156	Owned	Individual	30	30

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age
1	50884	C4	7	Owned	Joint	69	68
2	50885	C1	564	Rented	Individual	28	28
3	50886	C3	1177	Rented	Individual	23	23
4	50887	C1	951	Owned	Individual	75	75

In [37]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21805 entries, 0 to 21804
Data columns (total 13 columns):
ID                          21805 non-null int64
City_Code                   21805 non-null object
Region_Code                 21805 non-null int64
Accomodation_Type           21805 non-null object
Reco_Insurance_Type         21805 non-null object
Upper_Age                   21805 non-null int64
Lower_Age                   21805 non-null int64
Is_Spouse                   21805 non-null object
Health Indicator            16778 non-null object
Holding_Policy_Duration     13202 non-null object
Holding_Policy_Type         13202 non-null float64
Reco_Policy_Cat             21805 non-null int64
Reco_Policy_Premium         21805 non-null float64
dtypes: float64(2), int64(5), object(6)
memory usage: 2.2+ MB
```

In [38]:
```
test_df.columns
```

Out[38]:
```
Index(['ID', 'City_Code', 'Region_Code', 'Accomodation_Type',
       'Reco_Insurance_Type', 'Upper_Age', 'Lower_Age', 'Is_Spouse',
       'Health Indicator', 'Holding_Policy_Duration', 'Holding_Policy_Type',
       'Reco_Policy_Cat', 'Reco_Policy_Premium'],
      dtype='object')
```

In [39]:
```
# Variables with missing data
test_mis_vars = [var for var in test_df.columns if test_df[var].isnull().sum()>0]
test_df[test_mis_vars].isnull().sum()
```

Out[39]:
```
Health Indicator           5027
Holding_Policy_Duration    8603
Holding_Policy_Type        8603
dtype: int64
```

In [40]:
```
# Mode Imputation for missing categorical data:
test_df['Health Indicator'] = test_df['Health Indicator'].fillna(test_df['Health Indica
test_df['Holding_Policy_Duration'] = test_df['Holding_Policy_Duration'].fillna(test_df[
test_df['Holding_Policy_Type'] = test_df['Holding_Policy_Type'].fillna(np.around(test_d
test_df.isnull().sum()
```

Out[40]:
```
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          0
Holding_Policy_Duration   0
Holding_Policy_Type       0
Reco_Policy_Cat           0
Reco_Policy_Premium       0
dtype: int64
```

In [41]:
```
# Converting Holding_Policy_Duration to float
test_df.Holding_Policy_Duration = test_df.Holding_Policy_Duration.str.replace('[+]', ''
test_df['Holding_Policy_Duration'] = test_df['Holding_Policy_Duration'].astype(float)
test_df['Holding_Policy_Duration'].head()
```

Out[41]:
```
0     6.0
1     3.0
2     2.0
3     3.0
4    14.0
Name: Holding_Policy_Duration, dtype: float64
```

In [42]:
```
# Categorical variables in test data:
test_cat_vars = [var for var in test_df.columns if test_df[var].dtypes=='O']
test_cat_vars
```

Out[42]:
```
['City_Code',
 'Accomodation_Type',
 'Reco_Insurance_Type',
 'Is_Spouse',
 'Health Indicator']
```

In [43]:
```
# Label Encoding for categorical variables:
test_City_Code_le = LabelEncoder()
test_City_Code_labels = test_City_Code_le.fit_transform(test_df['City_Code'])
test_City_Code_mappings = {index: label for index, label in
                  enumerate(test_City_Code_le.classes_)}
print(test_City_Code_mappings)

test_Accomodation_Type_le = LabelEncoder()
test_Accomodation_Type_labels = test_Accomodation_Type_le.fit_transform(test_df['Accomo
test_Accomodation_Type_mappings = {index: label for index, label in
                  enumerate(test_Accomodation_Type_le.classes_)}
print(test_Accomodation_Type_mappings)

test_Reco_Insurance_Type_le = LabelEncoder()
test_Reco_Insurance_Type_labels = test_Reco_Insurance_Type_le.fit_transform(test_df['Re
test_Reco_Insurance_Type_mappings = {index: label for index, label in
                  enumerate(test_Reco_Insurance_Type_le.classes_)}
print(test_Reco_Insurance_Type_mappings)

test_Is_Spouse_le = LabelEncoder()
test_Is_Spouse_labels = test_Is_Spouse_le.fit_transform(test_df['Is_Spouse'])
test_Is_Spouse_mappings = {index: label for index, label in
                  enumerate(test_Is_Spouse_le.classes_)}
print(test_Is_Spouse_mappings)

test_Health_Indicator_le = LabelEncoder()
test_Health_Indicator_labels = test_Health_Indicator_le.fit_transform(test_df['Health I
test_Health_Indicator_mappings = {index: label for index, label in
                  enumerate(test_Health_Indicator_le.classes_)}
print(test_Health_Indicator_mappings)
```

```
{0: 'C1', 1: 'C10', 2: 'C11', 3: 'C12', 4: 'C13', 5: 'C14', 6: 'C15', 7: 'C16', 8: 'C1
7', 9: 'C18', 10: 'C19', 11: 'C2', 12: 'C20', 13: 'C21', 14: 'C22', 15: 'C23', 16: 'C2
4', 17: 'C25', 18: 'C26', 19: 'C27', 20: 'C28', 21: 'C29', 22: 'C3', 23: 'C30', 24: 'C3
1', 25: 'C32', 26: 'C33', 27: 'C34', 28: 'C35', 29: 'C36', 30: 'C4', 31: 'C5', 32: 'C6',
33: 'C7', 34: 'C8', 35: 'C9'}
{0: 'Owned', 1: 'Rented'}
{0: 'Individual', 1: 'Joint'}
{0: 'No', 1: 'Yes'}
{0: 'X1', 1: 'X2', 2: 'X3', 3: 'X4', 4: 'X5', 5: 'X6', 6: 'X7', 7: 'X8', 8: 'X9'}
```

In [44]:
```
test_df['City_Code_Labels'] = test_City_Code_labels
test_df['Accomodation_Type_Labels'] = test_Accomodation_Type_labels
test_df['Reco_Insurance_Type_Labels'] = test_Reco_Insurance_Type_labels
test_df['Is_Spouse_Labels'] = test_Is_Spouse_labels
test_df['Health_Indicator_Labels'] = test_Health_Indicator_labels
test_df.head()
```

Out[44]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age
0	50883	C1	156	Owned	Individual	30	30
1	50884	C4	7	Owned	Joint	69	68
2	50885	C1	564	Rented	Individual	28	28
3	50886	C3	1177	Rented	Individual	23	23
4	50887	C1	951	Owned	Individual	75	75

In [45]:
```
test_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21805 entries, 0 to 21804
Data columns (total 18 columns):
ID                              21805 non-null int64
City_Code                       21805 non-null object
Region_Code                     21805 non-null int64
Accomodation_Type               21805 non-null object
Reco_Insurance_Type             21805 non-null object
Upper_Age                       21805 non-null int64
Lower_Age                       21805 non-null int64
Is_Spouse                       21805 non-null object
Health Indicator                21805 non-null object
Holding_Policy_Duration         21805 non-null float64
Holding_Policy_Type             21805 non-null float64
Reco_Policy_Cat                 21805 non-null int64
Reco_Policy_Premium             21805 non-null float64
City_Code_Labels                21805 non-null int32
Accomodation_Type_Labels        21805 non-null int32
Reco_Insurance_Type_Labels      21805 non-null int32
Is_Spouse_Labels                21805 non-null int32
Health_Indicator_Labels         21805 non-null int32
dtypes: float64(3), int32(5), int64(5), object(5)
memory usage: 2.6+ MB
```

In [46]:
```
# Dropping Categorical Variables from test data:
test_df_1 = test_df.copy()
test_df_2 = test_df.copy()
```

In [47]:
```
test_df_1.drop(columns=['ID','City_Code','Accomodation_Type','Reco_Insurance_Type','Is_
test_df_1.head()
```

Out[47]:

	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Duration	Holding_Policy_Type	Reco_Policy_Cat
0	156	30	30	6.0	3.0	5
1	7	69	68	3.0	3.0	18
2	564	28	28	2.0	4.0	17
3	1177	23	23	3.0	3.0	18
4	951	75	75	14.0	2.0	5

In [48]:
```
test_df_2.drop(columns=['ID','City_Code','Accomodation_Type','Reco_Insurance_Type','Is_
test_df_2.head()
```

Out[48]:

	Upper_Age	Lower_Age	Holding_Policy_Duration	Holding_Policy_Type	Reco_Policy_Premium	Accomo
0	30	30	6.0	3.0	11934.0	
1	69	68	3.0	3.0	32204.8	
2	28	28	2.0	4.0	9240.0	
3	23	23	3.0	3.0	9086.0	
4	75	75	14.0	2.0	22534.0	

In [49]:
```
X_test_1 = test_df_1.values
X_test_2 = test_df_2.values
```

In [50]:
```
X_test_1 = scaler.transform(X_test_1)
X_test_2 = scaler2.transform(X_test_2)
```

Checking for Imbalanced Data:

In [51]:
```
train_df['Response'].value_counts()
```

Out[51]:
```
0    38673
1    12209
Name: Response, dtype: int64
```

The data above clearly shows 75% values belong to majority class while 25% belong to minority class.

- **Hence, this is an example of Imbalanced Dataset**
- **We will use techniques like Bagging, Boosting, Over & Under sampling & Hybrid models to tackle the predictions**

We will compare the performance of:

- **just re-sampling**
- **just boosting or bagging**
- **bagging + resampling**

- **boosting + resampling**
- **bagging + boosting + resampling**

6. Multiple Model creation & training:

Ensemble Algorithms: With & without resampling:

In [52]:
```
from sklearn.ensemble import (
    RandomForestClassifier,
    AdaBoostClassifier,
)

from imblearn.ensemble import (
    BalancedRandomForestClassifier,
    RUSBoostClassifier,
)

from sklearn.metrics import roc_auc_score
from collections import Counter
```

In [53]:
```
# function to train ada boost and evaluate performance

def run_adaboost(X_train, X_test, y_train):

    ada = AdaBoostClassifier(n_estimators=200, random_state=100)

    ada.fit(X_train, y_train)

    print('Train set')
    pred = ada.predict_proba(X_train)
    print(
        'AdaBoost roc-auc: {}'.format(roc_auc_score(y_train, pred[:, 1])))
    final_pred = ada.predict(X_test)
    return final_pred
```

In [54]:
```
# train model and store result
ada_preds = run_adaboost(X_train_1, X_test_1, y_train_1)
print()
```

```
Train set
AdaBoost roc-auc: 0.6321337892995574
```

In [55]:
```
ada_preds_data = pd.DataFrame({'ID':test_df.ID, 'Response':ada_preds})
ada_preds_data.head()
```

Out[55]:

	ID	Response
0	50883	0
1	50884	0
2	50885	0
3	50886	0
4	50887	0

In [56]:
```
ada_preds_data.to_csv('Imbalanced-Ada-Boost.csv',index=False)
```

Random Under Sampling:

In [57]:
```
class_0, class_1 = train_df_1['Response'].value_counts()
print(class_0, class_1)
```

```
38673 12209
```

In [58]:
```
# import library
from imblearn.under_sampling import RandomUnderSampler

rus = RandomUnderSampler(random_state=42, replacement=True)# fit predictor and target v
x_rus, y_rus = rus.fit_resample(X_train_1, y_train_1)

print('original dataset shape:', Counter(y_train_1))
print('Resample dataset shape', Counter(y_rus))
```

```
original dataset shape: Counter({0: 38673, 1: 12209})
Resample dataset shape Counter({0: 12209, 1: 12209})
```

In [59]:
```
def run_random_forest(X_train, X_test, y_train, n_est, max_dep):

    rf = RandomForestClassifier(
        n_estimators=n_est, random_state=40, max_depth=max_dep, n_jobs=4)
    rf.fit(X_train, y_train)

    print('Train set')
    pred = rf.predict_proba(X_train)
    print(
        'Random Forests roc-auc: {}'.format(roc_auc_score(y_train, pred[:, 1])))
    final_pred = rf.predict(X_test)
    return final_pred
```

In [60]:
```
# Model training and predictions:
random_forest_model = run_random_forest(x_rus, X_test_1, y_rus, 500, 3)
```

```
Train set
Random Forests roc-auc: 0.6001035484572115
```

Random OverSampling:

In [61]:
```
from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)

# fit predictor and target variable
x_ros, y_ros = ros.fit_resample(X_train_1, y_train_1)

print('Original dataset shape', Counter(y_train_1))
print('Resample dataset shape', Counter(y_ros))
```

```
Original dataset shape Counter({0: 38673, 1: 12209})
Resample dataset shape Counter({0: 38673, 1: 38673})
```

In [62]:
```
random_forest_model_ros = run_random_forest(x_ros, X_test_1, y_ros, 500, 3)
```

```
Train set
Random Forests roc-auc: 0.5994817318009301
```

SMOTE:

In [63]:
```
from imblearn.over_sampling import SMOTE

smote = SMOTE()

# fit predictor and target variable
x_smote, y_smote = smote.fit_resample(X_train_1, y_train_1)

print('Original dataset shape', Counter(y_train_1))
print('Resample dataset shape', Counter(y_smote))
```

```
Original dataset shape Counter({0: 38673, 1: 12209})
Resample dataset shape Counter({0: 38673, 1: 38673})
```

In [68]:
```
random_forest_model_smote = run_random_forest(x_smote, X_test_1, y_smote, 200, 5)
# best model => run_random_forest(x_smote, X_test_1, y_smote, 500, 30) =>But Overfittin
```

```
Train set
Random Forests roc-auc: 0.7511621280224545
```

In [69]:
```
random_forest_model_smote_data = pd.DataFrame({'ID':test_df.ID, 'Response':random_fores
random_forest_model_smote_data.to_csv('Smote-Random-Forest-Preds.csv',index=False)
random_forest_model_smote_data.head()
```

Out[69]:

	ID	Response
0	50883	0
1	50884	0
2	50885	1
3	50886	1
4	50887	0

In [66]:
```
new_ada_preds = run_adaboost(x_smote, X_test_1, y_smote)
```

```
Train set
AdaBoost roc-auc: 0.8247407550921627
```

In [67]:
```
new_ada_preds_data = pd.DataFrame({'ID':test_df.ID, 'Response':new_ada_preds})
new_ada_preds_data.to_csv('New-AdaBoost-Preds.csv',index=False)
new_ada_preds_data.head()
```

Out[67]:

	ID	Response
0	50883	0
1	50884	0
2	50885	0
3	50886	0
4	50887	0

In [ ]:
```
```