Cross Selling Use Case

The Client wants to cross sell health insurance to potential customers who may or may not be having insurance with the company currently. We have been provided with the data containing the Demographic information,Holding policy info and the recommended policy details with Response variable as the target variable.

Here are the different sections or steps followed while building a model-

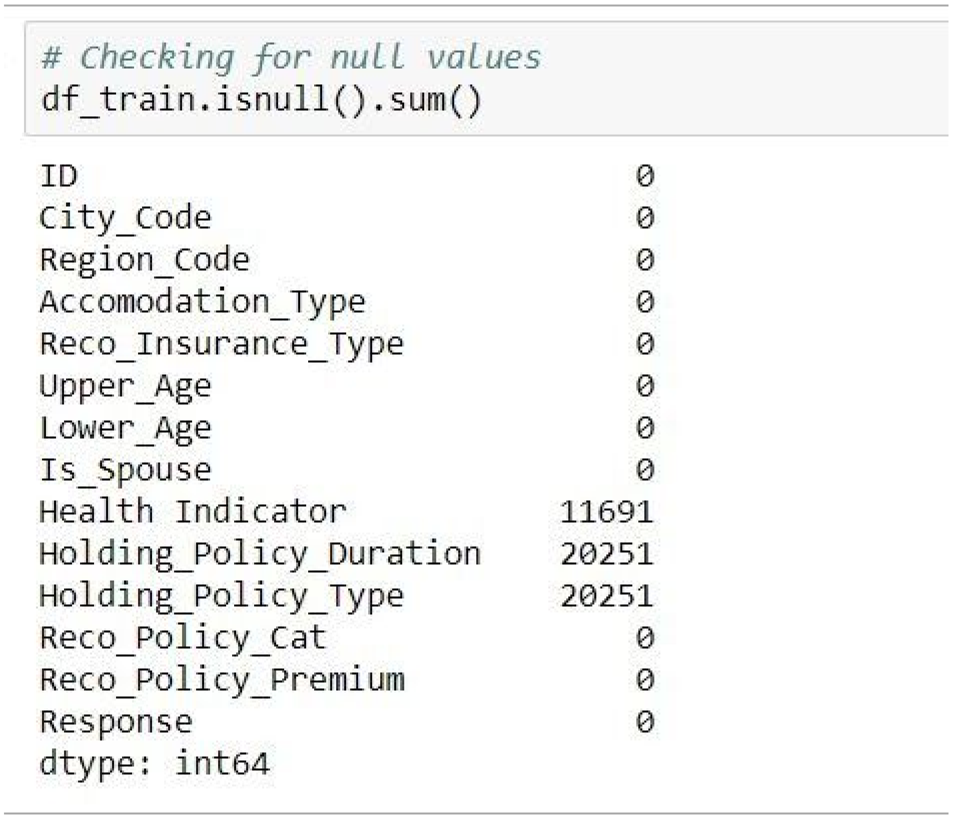
1. Loading the train data and importing relevant libraries.
2. Cleaning the data and removing missing values in the data and Label Encoding the data.
3. Analysis and Feature Engineering
4. Model Building And Submissions

Loading The train data and importing relevant libraries

The train data is loaded and libraries are imported.

Cleaning the Data and Removing Missing Values in the Data-

The data is cleaned by looking for missing values and imputing any missing values in the data.



```
# Checking for null values
df_train.isnull().sum()
```

```
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
```

Here, Features like Health Indicator, Holding_Policy_Duration, Holding_Policy_Type had missing values. So, the missing values were imputed.

Feature Name	Imputed Value
Health Indicator(dtype-object)	mode
Holding_Policy_Duration(dtype-object)	‘0.0’
Holding_Policy_Type(dtype-float)	0.0

The above table shows the imputed value for each column with null values.

For **Holding_Policy_Duration** and **Holding_Policy_Type,** I imputed the value 0.

The **reason for 0 value imputation** is the fact that the person **may or may not have** any insurance currently with the firm. So, Wherever these two columns are NaN, we can assume that they do not hold any policy currently. Hence, the value 0 is imputed.

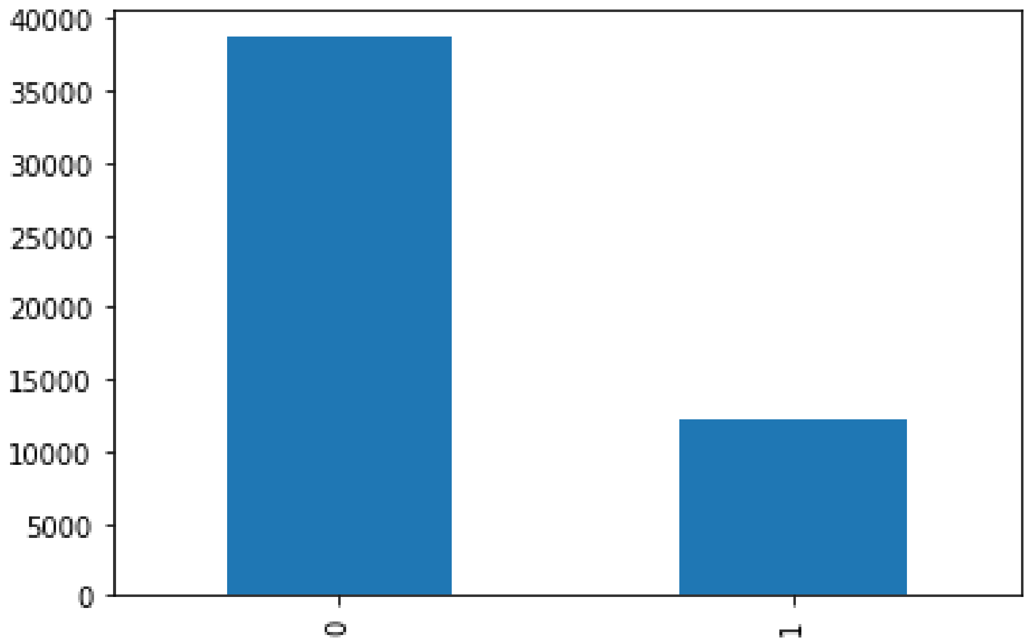
Once the Data Cleaning part is done, we proceed for Label Encoding and we label encode the features which have object data types.

Analysis and Feature Engineering

Once the Label Encoding Part is Done, It is the time for the analysis of the data and also the feature engineering part

Analysis Part-

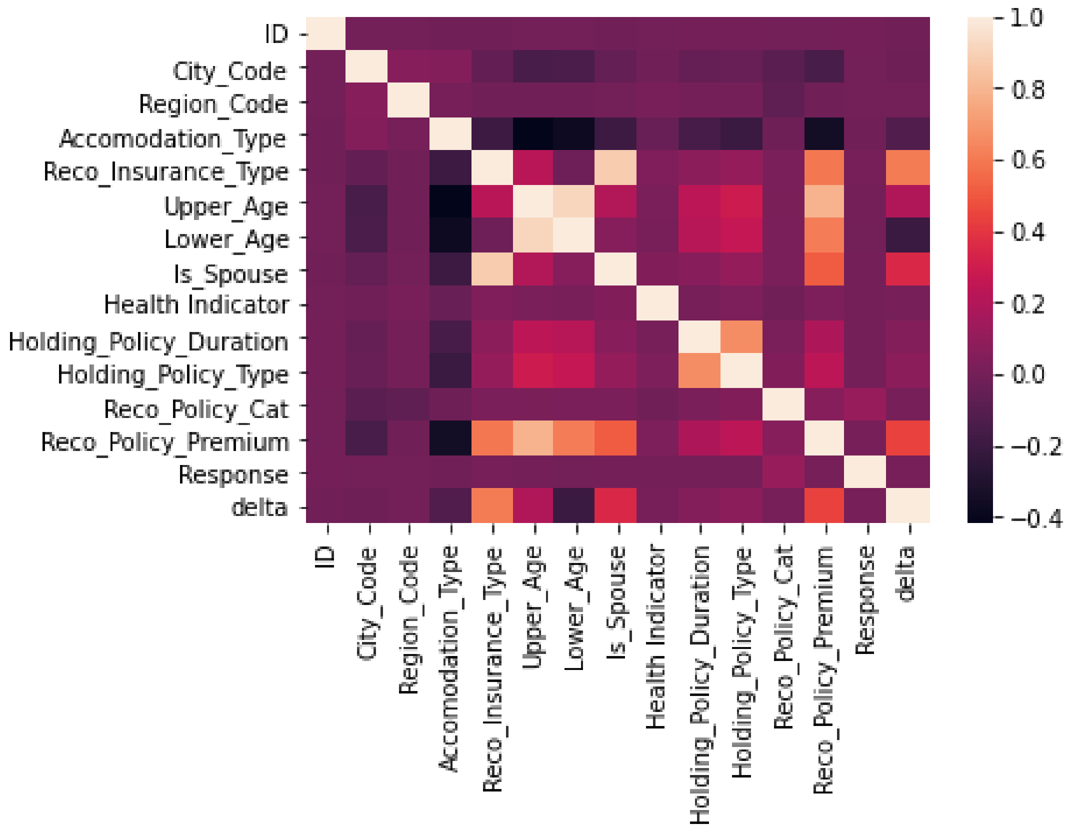
First , Let us look at the number of samples for different classes. It can be seen in the bar plot below



Value Counts for Response Variable shows the response variable is imbalanced

As can be seen, There is an Imbalance in our data with 0 valued samples more than 1 valued samples. So, this imbalance needs to be treated. In our Case, I used Oversampling to treat the imbalance in our dataset.

Then, I also looked at the correlation heatmaps to observe if there are any correlated features. The correlation heatmap can be seen below-



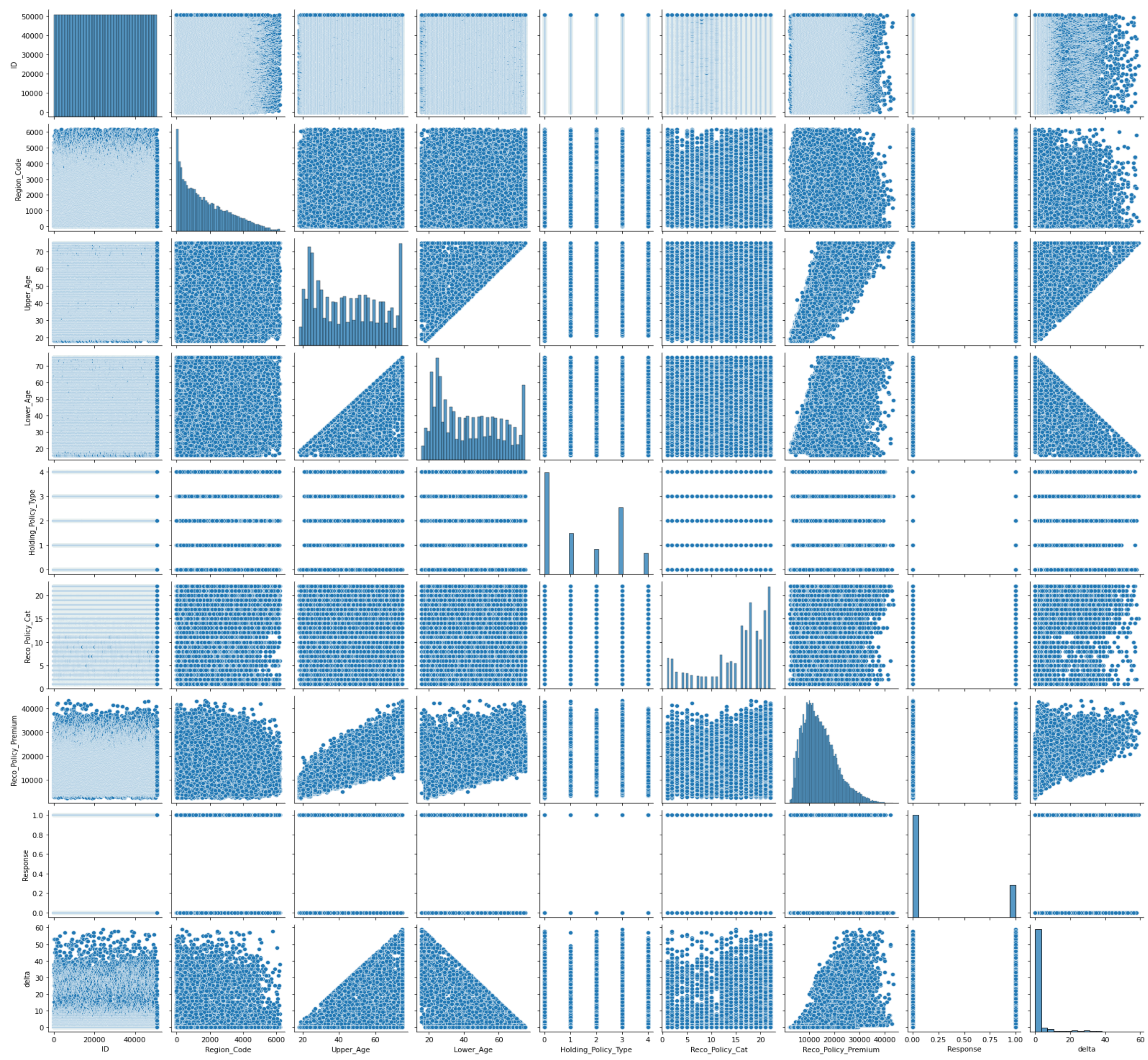
Correlation Heatmap - Delta= (Upper_Age-Lower_Age)

Here are some points that can be inferred from the correlation map-

1. We observe higher correlation between Upper_Age and Lower_Age. So, we used a new feature called - **Delta** = ***(Upper_Age-Lower_Age) .*** This feature was introduced to provide the model with the information that Upper_Age was not able to provide. We observed that the delta was 0 majority of the time.
2. Reco_Insurance Type has positive correlation with the Is_Spouse Feature.

Feature Added-
Delta = (Upper_Age-Lower_Age)

Scaling of features was not required in my case as I was using tree based models like Random Forest and XGBoost.



Pairplot showing the pairwise relationship plot for the whole data

Pair Plot was also done to observe the **pairwise relationships between variables.** It did not provide much significant information. But it provided us with the information about the distribution of different variables. For example, the **Reco_Policy_Premium** had a positively skewed distribution. Also, the upper age and lower age had similar distribution. Delta also has positively skewed distribution with majority of values 0.

Model Building And Submissions-

Now we proceed with the modelling part. So, here, we chose the Tree based Algorithm because-

1. These models are robust to outliers.
2. The models don't require scaling for numerical features.

While training the model, the training data was split into training and validation sets(80% training and 20% validation).
The List of Models along with their logic in brief can be found below-

Submission No.	Score	Description
1	0.62272192076006	Random forest model used with default parameters with **Oversampling** for Tackling Imbalance
2	0.59006998067310	Random forest model used with default parameters with **SMOTE** for Tackling Imbalance
3	0.63072612756656	Random forest model used with default parameters with **Oversampling** for Tackling Imbalance And Using New Feature Delta
4	0.63067472506919	Random forest model used with default parameters with **Oversampling** for Tackling Imbalance And Using New Feature Delta and by making changes into the imputation strategy for Holding_Policy_Duration and Holding_Policy_Type.
5	0.64456827234719	Tuned XGBoost using Random Search and generated the results.(Data used was same as that used in submission 4)
6	0.62255737472754	Random Search on Random Forest.(Data used was same as that used in submission 4)

In all Submissions except submission 2, I used Oversampling as a strategy to tackle imbalance in data.
Also, to tune the models, I found random search was working better as compared to grid search.

Final Model- Submission 5(XGBoost using Random Search). Random Forest also gave comparable results. But, XGBoost was marginally better.

More Implementation details can be found in the Jupyter Notebook.