In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
sns.set_theme(color_codes=True)
```

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

Out[2]:

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator
0	1	C3	3213	Rented	Individual	36	36	No	X1
1	2	C5	1117	Owned	Joint	75	22	No	X2
2	3	C5	3732	Owned	Individual	32	32	No	NaN
3	4	C24	4378	Owned	Joint	52	48	No	X1
4	5	C8	2190	Rented	Individual	44	44	No	X2

Data Preprocessing Part 1

In [3]:
```
# Print the size of the dataset
df.shape
```

Out[3]:
```
(50882, 14)
```

In [4]:
```
#Check the number of unique value from all of the object datatype
df.select_dtypes(include='object').nunique()
```

Out[4]:
```
City_Code                 36
Accomodation_Type          2
Reco_Insurance_Type        2
Is_Spouse                  2
Health Indicator           9
Holding_Policy_Duration   15
dtype: int64
```

In [5]:
```
# Remove unnecesary column
df.drop(columns = ['ID', 'City_Code', 'Region_Code'], inplace=True)
df.head()
```

Out[5]:

	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Holding_Policy_Duration	Holdi
0	Rented	Individual	36	36	No	X1	14+	
1	Owned	Joint	75	22	No	X2	NaN	
2	Owned	Individual	32	32	No	NaN	1.0	
3	Owned	Joint	52	48	No	X1	14+	
4	Rented	Individual	44	44	No	X2	3.0	

In [6]:
```
# Create a dictionary to define the replacement values for Exploratory Data Analysis
replace_dict = {0: 'no', 1: 'yes'}

# Replace the values in the "Response" column using the dictionary
df['Response'] = df['Response'].replace(replace_dict)
```

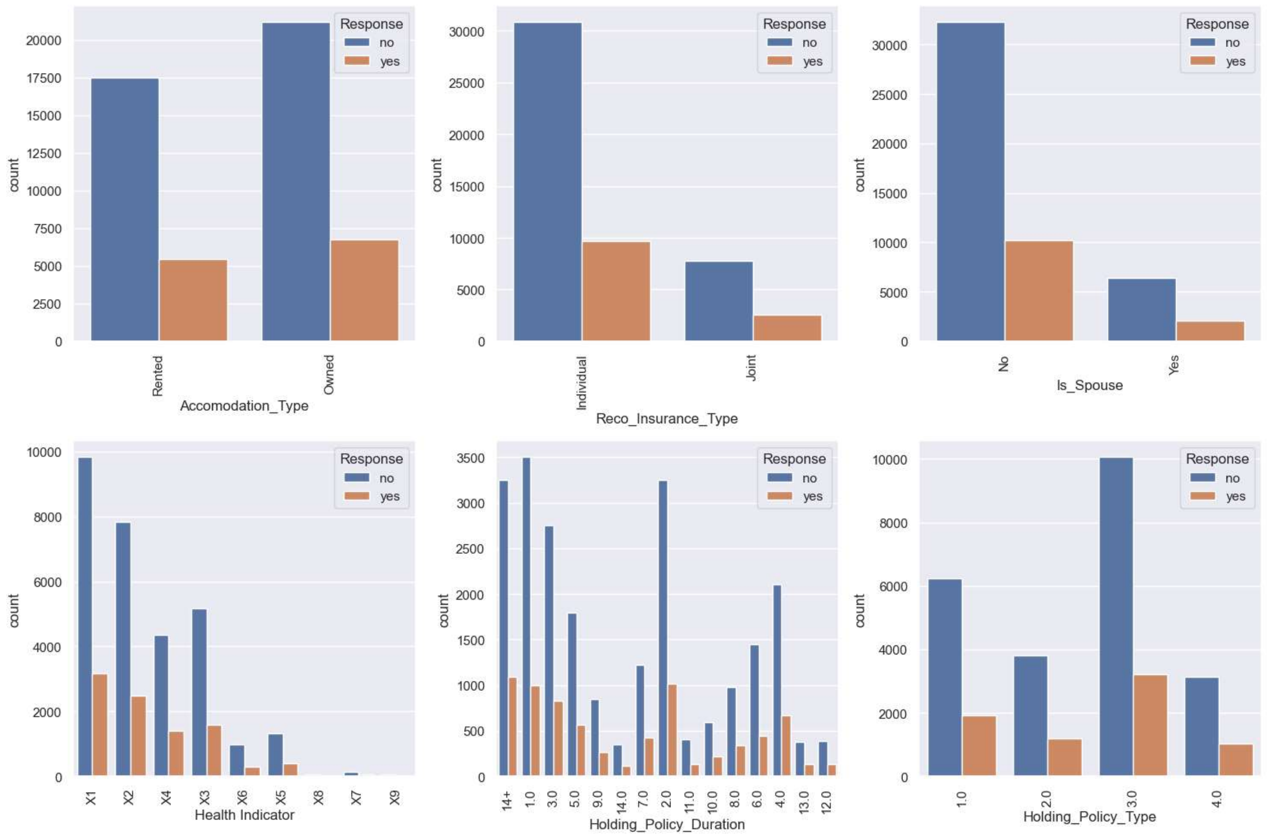
In [7]:
```
df.head()
```

Out[7]:

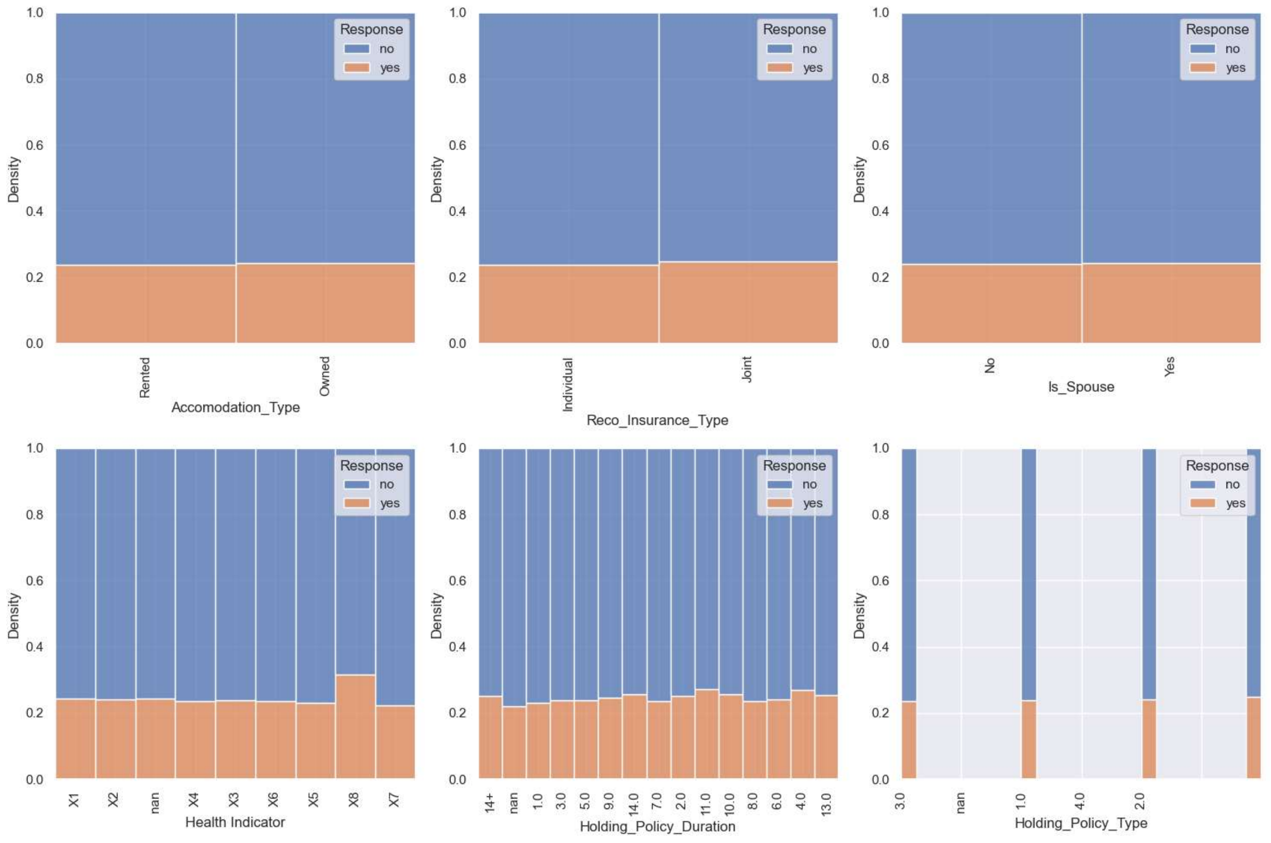
	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Holding_Policy_Duration	Holdi
0	Rented	Individual	36	36	No	X1	14+	
1	Owned	Joint	75	22	No	X2	NaN	
2	Owned	Individual	32	32	No	NaN	1.0	
3	Owned	Joint	52	48	No	X1	14+	
4	Rented	Individual	44	44	No	X2	3.0	

Exploratory Data Analysis

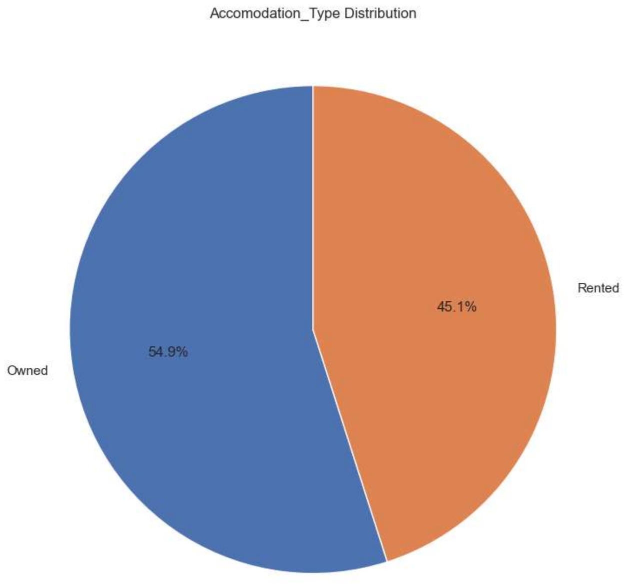
In [8]:
```
# list of categorical variables to plot
cat_vars = ['Accomodation_Type', 'Reco_Insurance_Type', 'Is_Spouse',
            'Health Indicator', 'Holding_Policy_Duration',
            'Holding_Policy_Type']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
axs = axs.flatten()

# create barplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.countplot(x=var, hue='Response', data=df, ax=axs[i])
    axs[i].set_xticklabels(axs[i].get_xticklabels(), rotation=90)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```

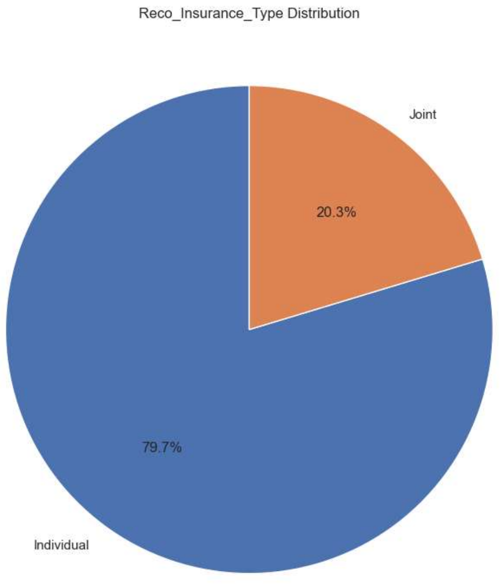


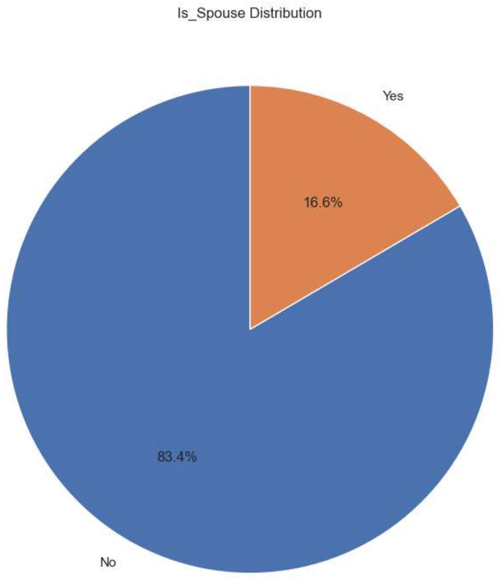
In [9]:
```
import warnings
warnings.filterwarnings("ignore")
# get list of categorical variables
cat_vars = ['Accomodation_Type', 'Reco_Insurance_Type', 'Is_Spouse',
            'Health Indicator', 'Holding_Policy_Duration',
            'Holding_Policy_Type']

# create figure with subplots
fig, axs = plt.subplots(nrows=2, ncols=3, figsize=(15, 10))
axs = axs.flatten()

# create histplot for each categorical variable
for i, var in enumerate(cat_vars):
    sns.histplot(x=var, hue='Response', data=df, ax=axs[i], multiple="fill", kde=False, element="ba
    axs[i].set_xticklabels(df[var].unique(), rotation=90)
    axs[i].set_xlabel(var)

# adjust spacing between subplots
fig.tight_layout()

# show plot
plt.show()
```

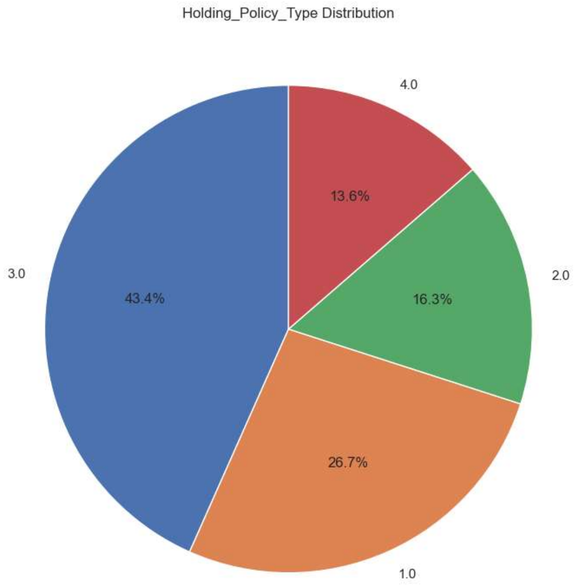


In [10]:
```
cat_vars = ['Accomodation_Type', 'Reco_Insurance_Type', 'Is_Spouse', 'Holding_Policy_Type']

# create a figure and axes
fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(15, 15))

# create a pie chart for each categorical variable
for i, var in enumerate(cat_vars):
    if i < len(axs.flat):
        # count the number of occurrences for each category
        cat_counts = df[var].value_counts()

        # create a pie chart
        axs.flat[i].pie(cat_counts, labels=cat_counts.index, autopct='%1.1f%%', startangle=90)

        # set a title for each subplot
        axs.flat[i].set_title(f'{var} Distribution')

# adjust spacing between subplots
fig.tight_layout()

# show the plot
plt.show()
```

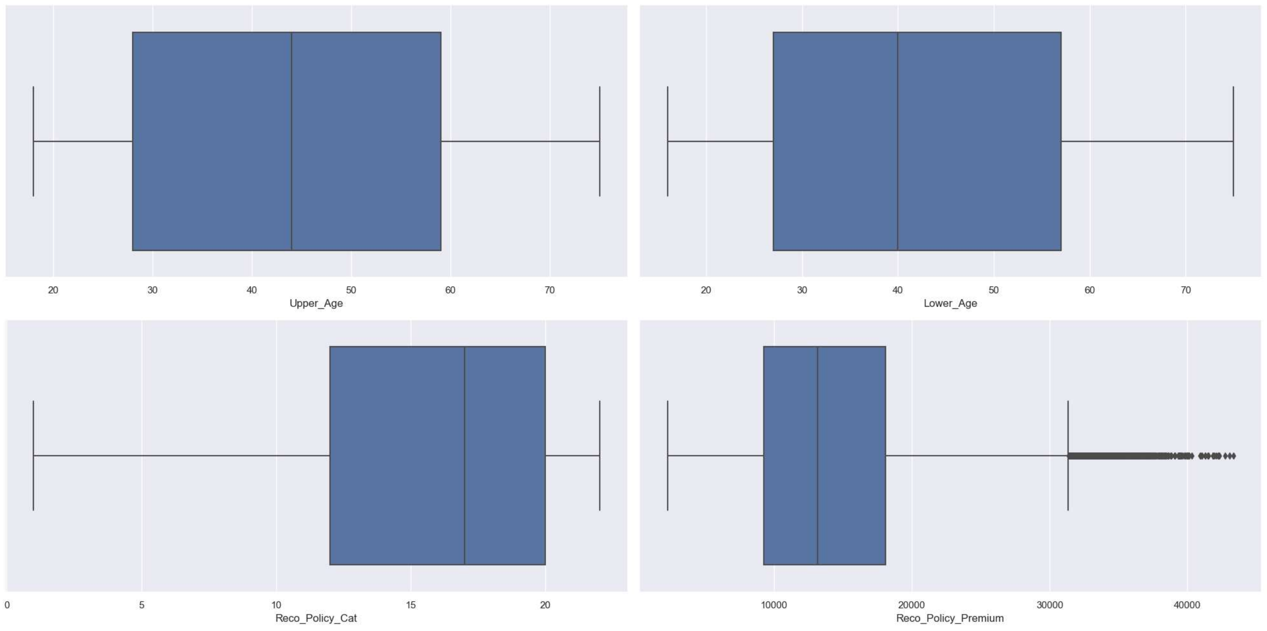

Accomodation_Type Distribution
Owned
54.9%
Rented
45.1%


Reco_Insurance_Type Distribution
Joint
20.3%
Individual
79.7%

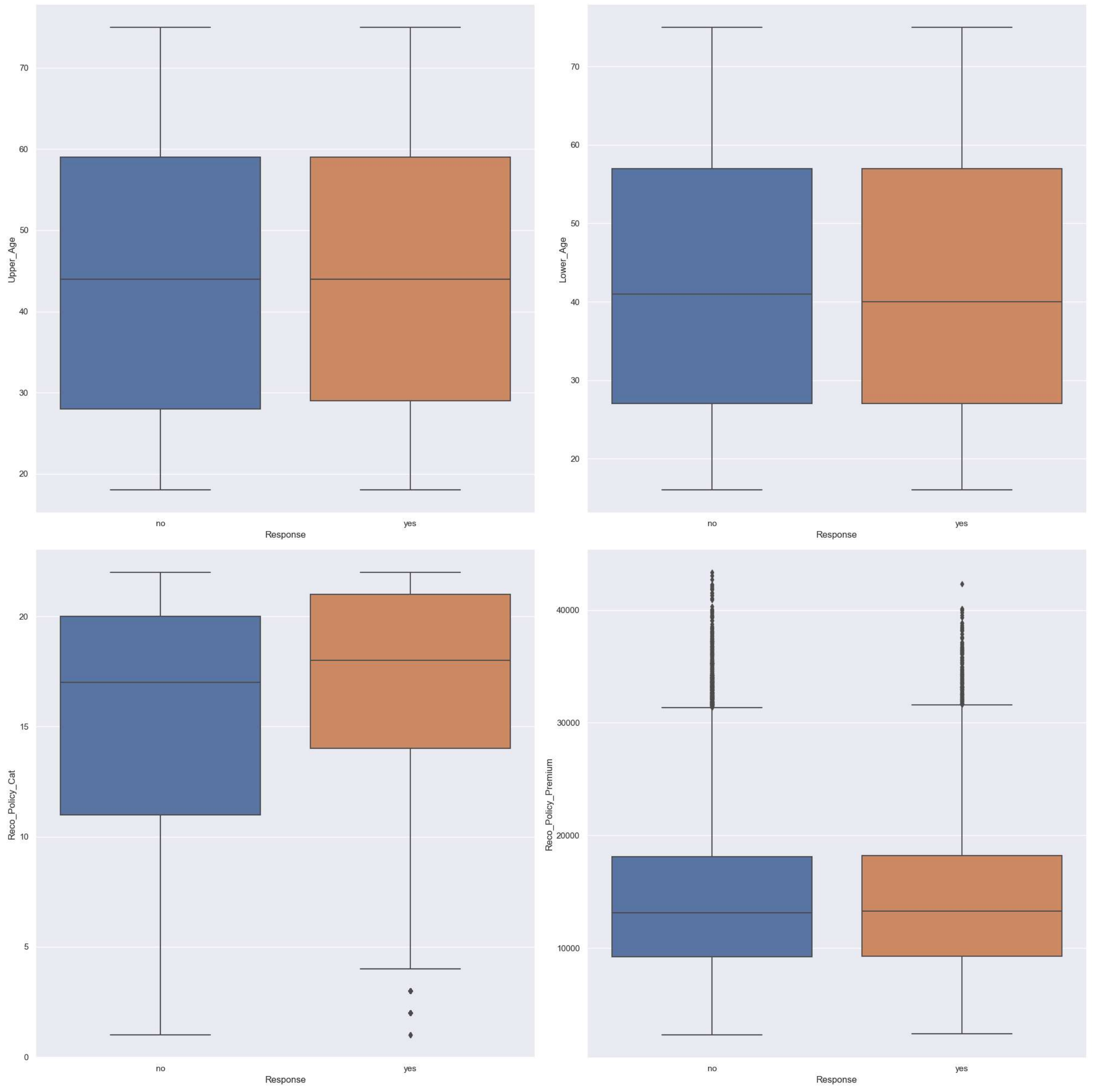

Is_Spouse Distribution
Yes
16.6%
No
83.4%


Holding_Policy_Type Distribution
4.0
13.6%
2.0
16.3%
1.0
26.7%
3.0
43.4%

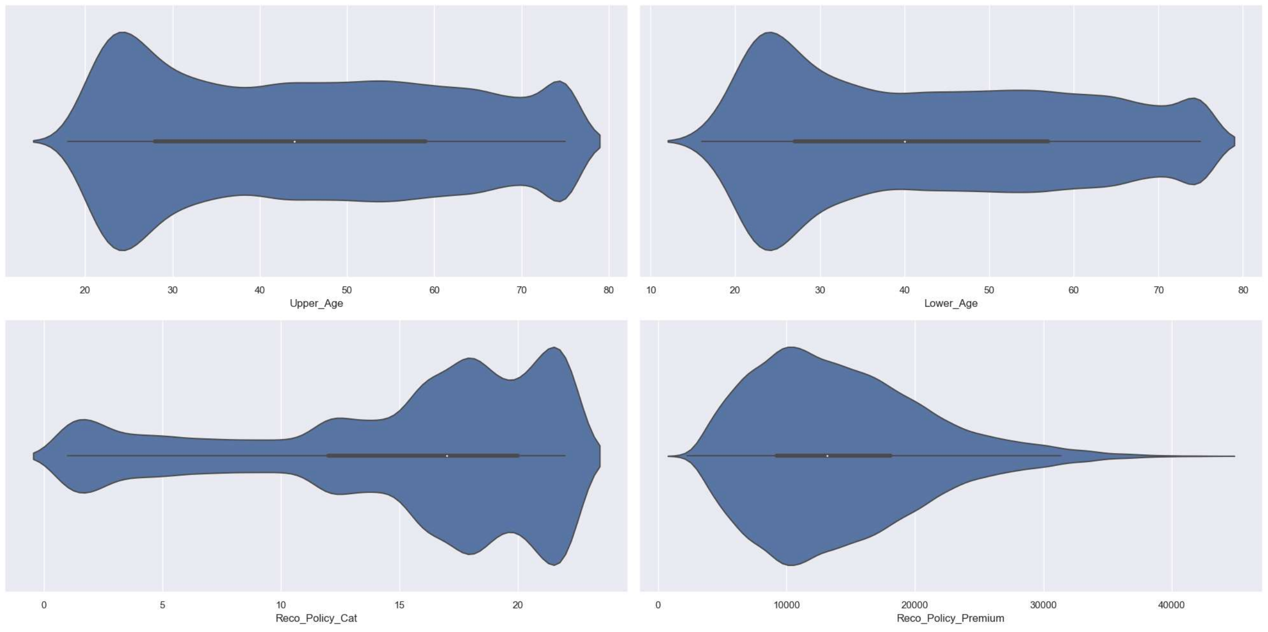
In [11]:
```
num_vars = ['Upper_Age', 'Lower_Age', 'Reco_Policy_Cat', 'Reco_Policy_Premium']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(x=var, data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



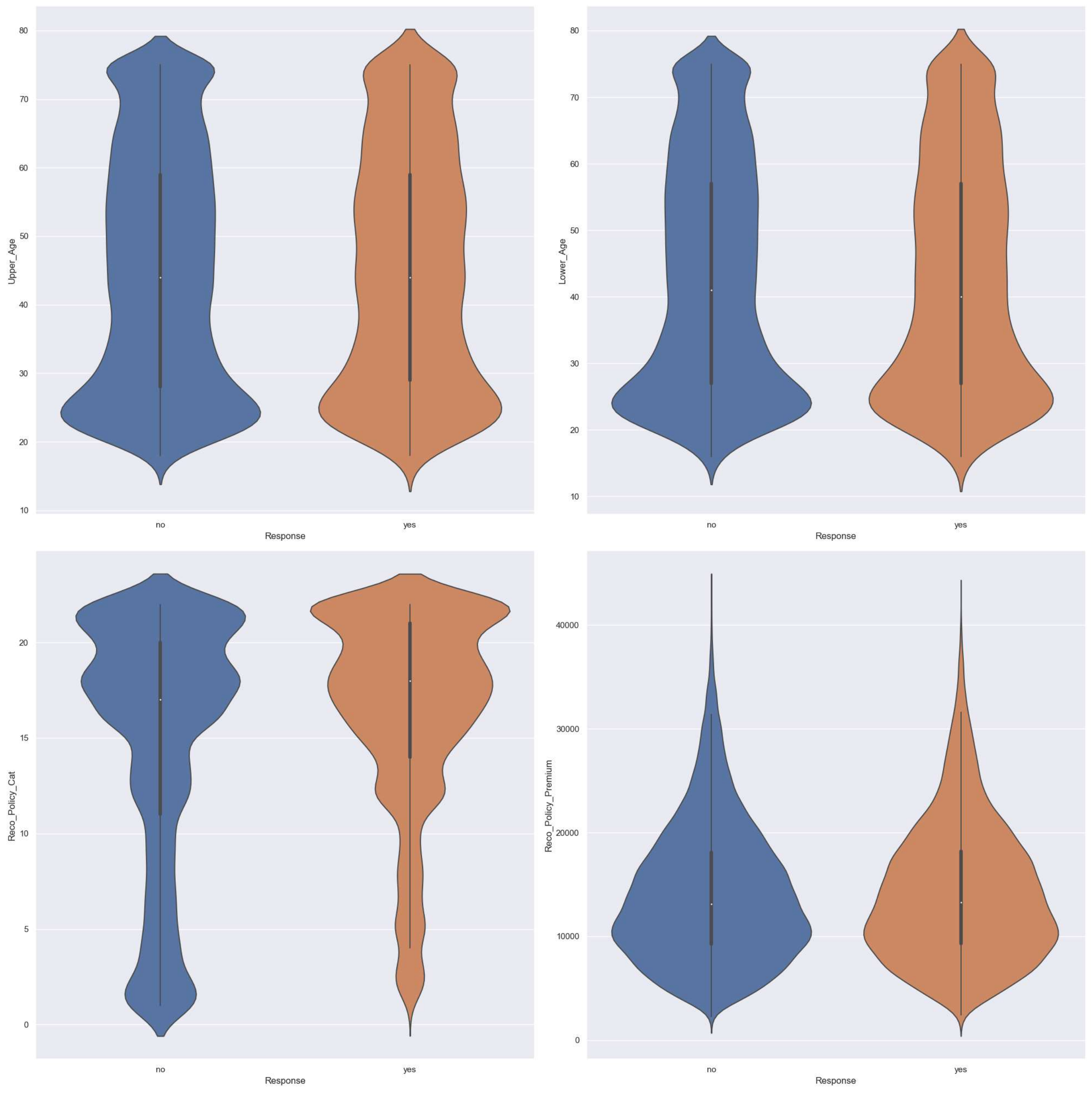
In [12]:
```
num_vars = ['Upper_Age', 'Lower_Age', 'Reco_Policy_Cat', 'Reco_Policy_Premium']

fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 20))
axs = axs.flatten()

for i, var in enumerate(num_vars):
    sns.boxplot(y=var, x='Response', data=df, ax=axs[i])

fig.tight_layout()

plt.show()
```



```
In [13]: num_vars = ['Upper_Age', 'Lower_Age', 'Reco_Policy_Cat', 'Reco_Policy_Premium']

         fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 10))
         axs = axs.flatten()

         for i, var in enumerate(num_vars):
             sns.violinplot(x=var, data=df, ax=axs[i])

         fig.tight_layout()

         plt.show()
```

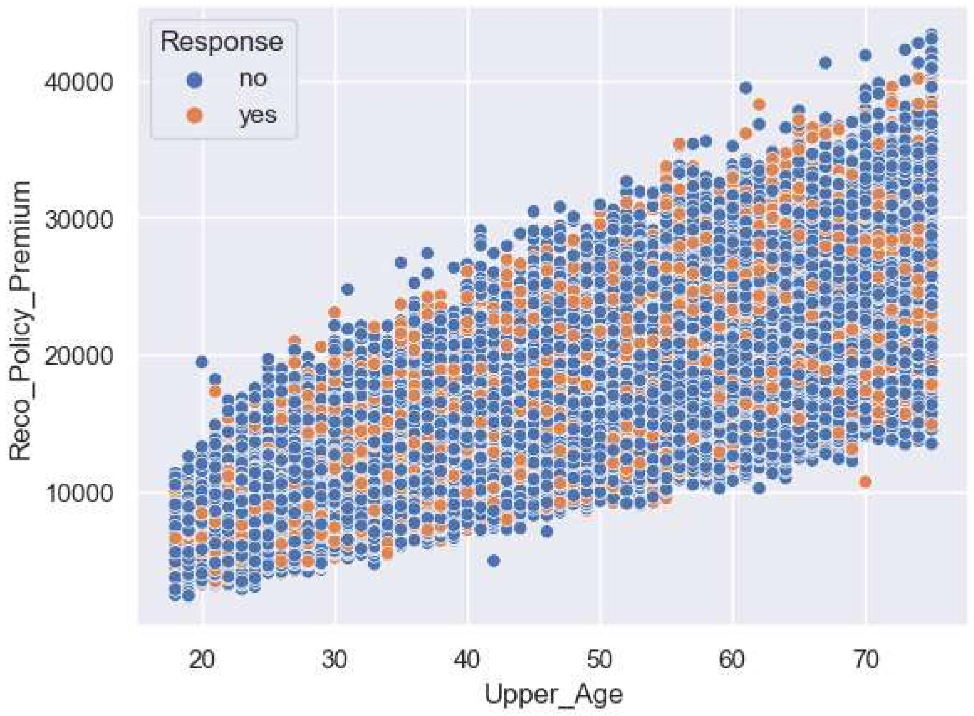


```
In [14]: num_vars = ['Upper_Age', 'Lower_Age', 'Reco_Policy_Cat', 'Reco_Policy_Premium']

         fig, axs = plt.subplots(nrows=2, ncols=2, figsize=(20, 20))
         axs = axs.flatten()

         for i, var in enumerate(num_vars):
             sns.violinplot(y=var, data=df, x='Response', ax=axs[i])

         fig.tight_layout()

         plt.show()
```



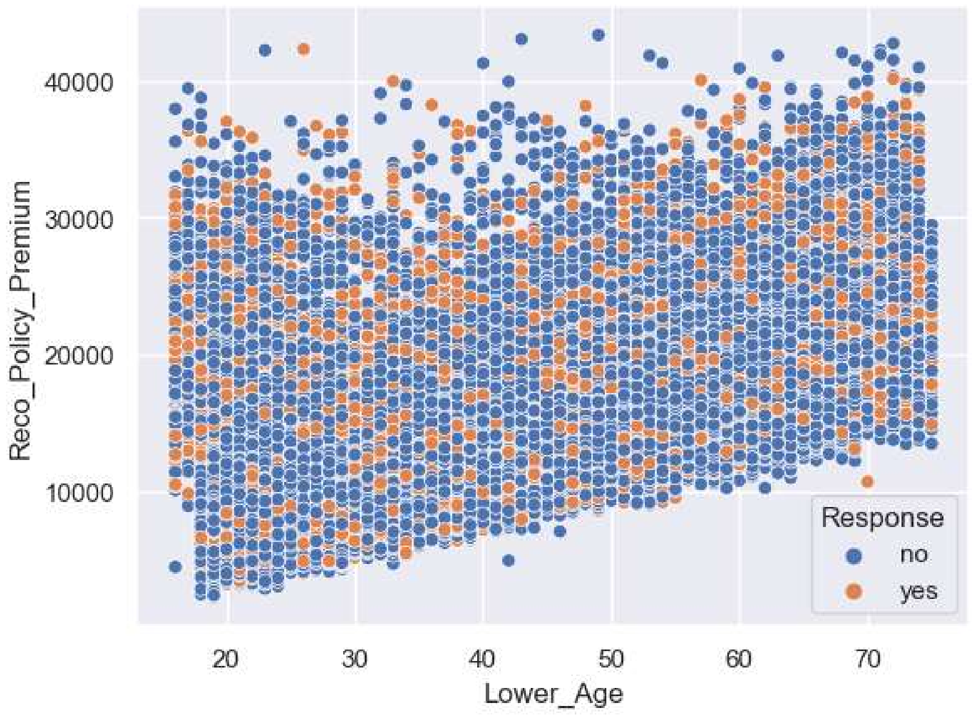
In [15]:
```
sns.scatterplot(x='Upper_Age', y='Reco_Policy_Premium', hue='Response', data=df)
```

Out[15]:
```
<AxesSubplot:xlabel='Upper_Age', ylabel='Reco_Policy_Premium'>
```



In [16]:
```
sns.scatterplot(x='Lower_Age', y='Reco_Policy_Premium', hue='Response', data=df)
```

Out[16]:
```
<AxesSubplot:xlabel='Lower_Age', ylabel='Reco_Policy_Premium'>
```



Data Preprocessing Part 2

In [17]:
```
#Check missing value
check_missing = df.isnull().sum() * 100 / df.shape[0]
check_missing[check_missing > 0].sort_values(ascending=False)
```

Out[17]:
```
Holding_Policy_Duration    39.799929
Holding_Policy_Type        39.799929
Health Indicator           22.976691
dtype: float64
```

In [18]:
```
# Drop null value each row that have null value
df.drop(columns = ['Holding_Policy_Duration', 'Holding_Policy_Type'], inplace=True)
df = df.dropna(subset=['Health Indicator'])
df.head()
```

Out[18]:

	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Reco_Policy_Cat	Reco_Policy_
0	Rented	Individual	36	36	No	X1	22	
1	Owned	Joint	75	22	No	X2	22	
3	Owned	Joint	52	48	No	X1	19	
4	Rented	Individual	44	44	No	X2	16	
5	Rented	Individual	52	52	No	X2	22	

Label Encoding for each Object datatype

In [19]:
```
# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Print the column name and the unique values
    print(f"{col}: {df[col].unique()}")
```

```
Accomodation_Type: ['Rented' 'Owned']
Reco_Insurance_Type: ['Individual' 'Joint']
Is_Spouse: ['No' 'Yes']
Health Indicator: ['X1' 'X2' 'X4' 'X3' 'X6' 'X5' 'X8' 'X7' 'X9']
Response: ['no' 'yes']
```

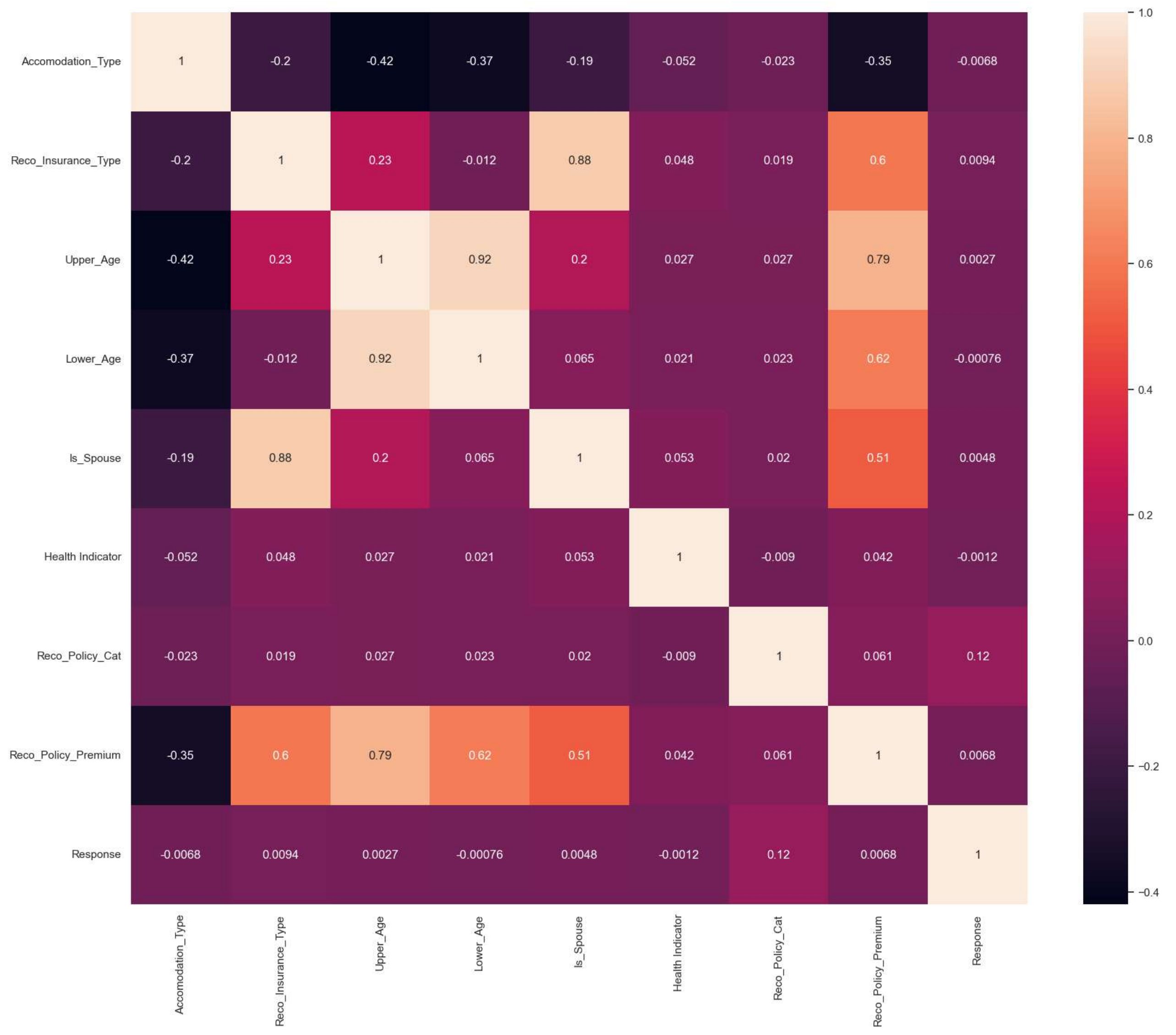
In [20]:
```
from sklearn import preprocessing

# Loop over each column in the DataFrame where dtype is 'object'
for col in df.select_dtypes(include=['object']).columns:

    # Initialize a LabelEncoder object
    label_encoder = preprocessing.LabelEncoder()

    # Fit the encoder to the unique values in the column
    label_encoder.fit(df[col].unique())

    # Transform the column using the encoder
    df[col] = label_encoder.transform(df[col])

    # Print the column name and the unique encoded values
    print(f"{col}: {df[col].unique()}")
```

```
Accomodation_Type: [1 0]
Reco_Insurance_Type: [0 1]
Is_Spouse: [0 1]
Health Indicator: [0 1 3 2 5 4 7 6 8]
Response: [0 1]
```

Correlation Heatmap

In [21]:
```
#Correlation Heatmap (print the correlation score each variables)
plt.figure(figsize=(20, 16))
sns.heatmap(df.corr(), fmt='.2g', annot=True)
```

Out[21]:
```
<AxesSubplot:>
```



	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Reco_Policy_Cat	Reco_Policy_Premium	Response
Accomodation_Type	1	-0.2	-0.42	-0.37	-0.19	-0.052	-0.023	-0.35	-0.0068
Reco_Insurance_Type	-0.2	1	0.23	-0.012	0.88	0.048	0.019	0.6	0.0094
Upper_Age	-0.42	0.23	1	0.92	0.2	0.027	0.027	0.79	0.0027
Lower_Age	-0.37	-0.012	0.92	1	0.065	0.021	0.023	0.62	-0.00076
Is_Spouse	-0.19	0.88	0.2	0.065	1	0.053	0.02	0.51	0.0048
Health Indicator	-0.052	0.048	0.027	0.021	0.053	1	-0.009	0.042	-0.0012
Reco_Policy_Cat	-0.023	0.019	0.027	0.023	0.02	-0.009	1	0.061	0.12
Reco_Policy_Premium	-0.35	0.6	0.79	0.62	0.51	0.042	0.061	1	0.0068
Response	-0.0068	0.0094	0.0027	-0.00076	0.0048	-0.0012	0.12	0.0068	1

Train Test Split

In [22]:
```
from sklearn.model_selection import train_test_split
# Select the features (X) and the target variable (y)
X = df.drop('Response', axis=1)
y = df['Response']

# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Remove the Outlier from train data using Z-Score

In [23]:
```
from scipy import stats

# Define the columns for which you want to remove outliers
selected_columns = ['Reco_Policy_Premium']

# Calculate the Z-scores for the selected columns in the training data
z_scores = np.abs(stats.zscore(X_train[selected_columns]))

# Set a threshold value for outlier detection (e.g., 3)
threshold = 3

# Find the indices of outliers based on the threshold
outlier_indices = np.where(z_scores > threshold)[0]

# Remove the outliers from the training data
X_train = X_train.drop(X_train.index[outlier_indices])
y_train = y_train.drop(y_train.index[outlier_indices])
```

Decision Tree

In [24]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree = DecisionTreeClassifier(class_weight='balanced')
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(dtree, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': 8, 'min_samples_leaf': 1, 'min_samples_split': 3, 'random_state': 0}
```

In [30]:
```
from sklearn.tree import DecisionTreeClassifier
dtree = DecisionTreeClassifier(random_state=0, max_depth=8, min_samples_leaf=1, min_samples_split=3
dtree.fit(X_train, y_train)
```

Out[30]:
```
DecisionTreeClassifier(class_weight='balanced', max_depth=8,
                       min_samples_split=3, random_state=0)
```

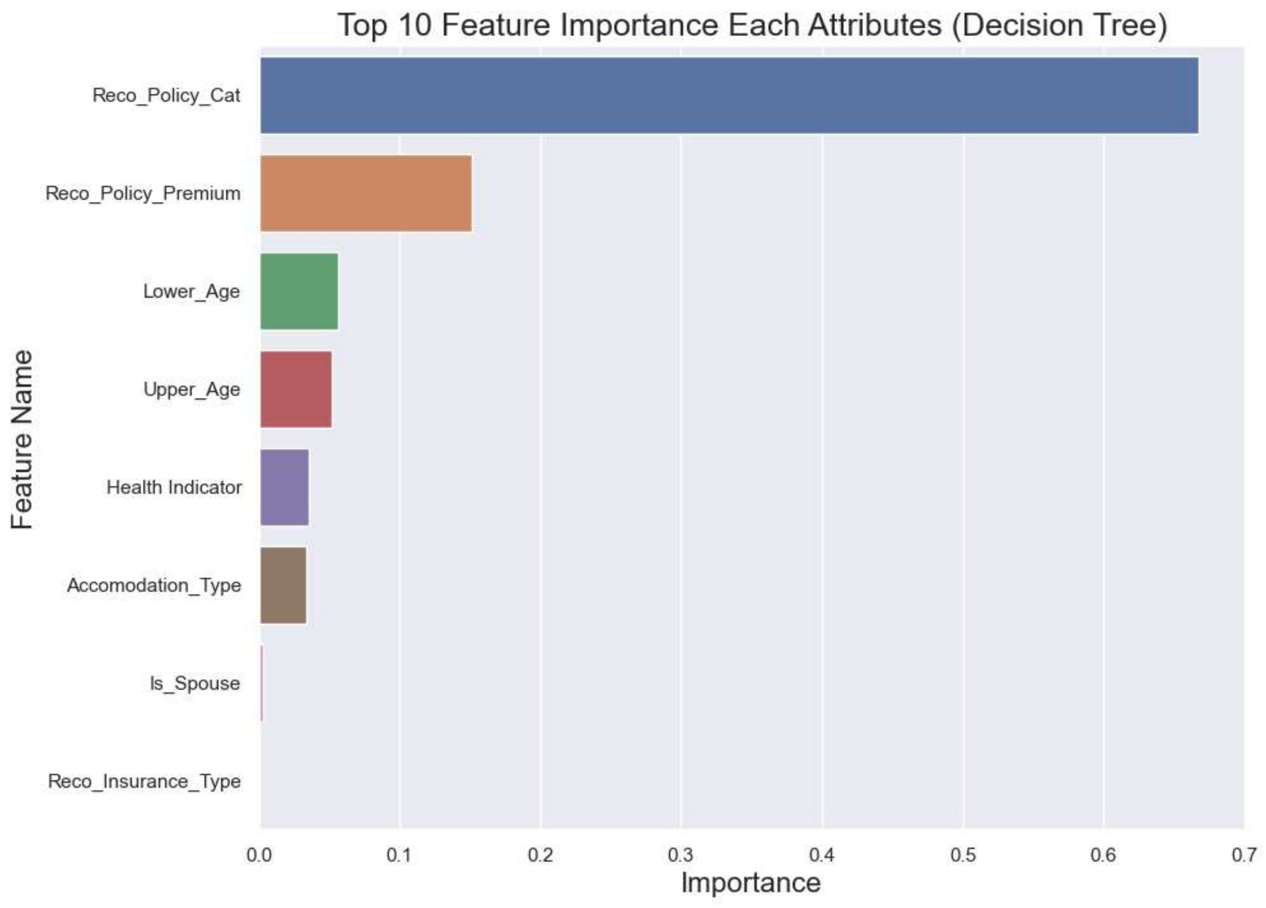
In [31]:
```
from sklearn.metrics import accuracy_score
y_pred = dtree.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 53.34 %
```

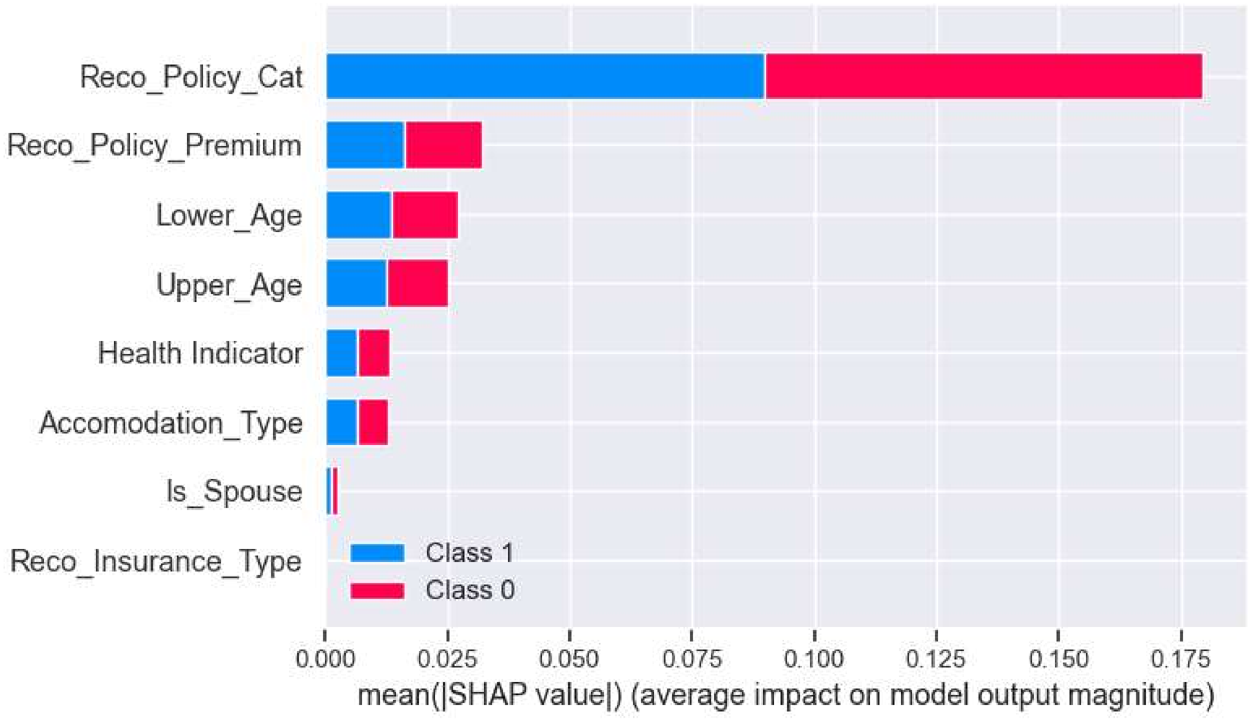
In [32]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score,
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.5333588467916827
Precision Score :  0.5333588467916827
Recall Score :  0.5333588467916827
Jaccard Score :  0.36366008523962773
Log Loss :  16.117525146875323
```

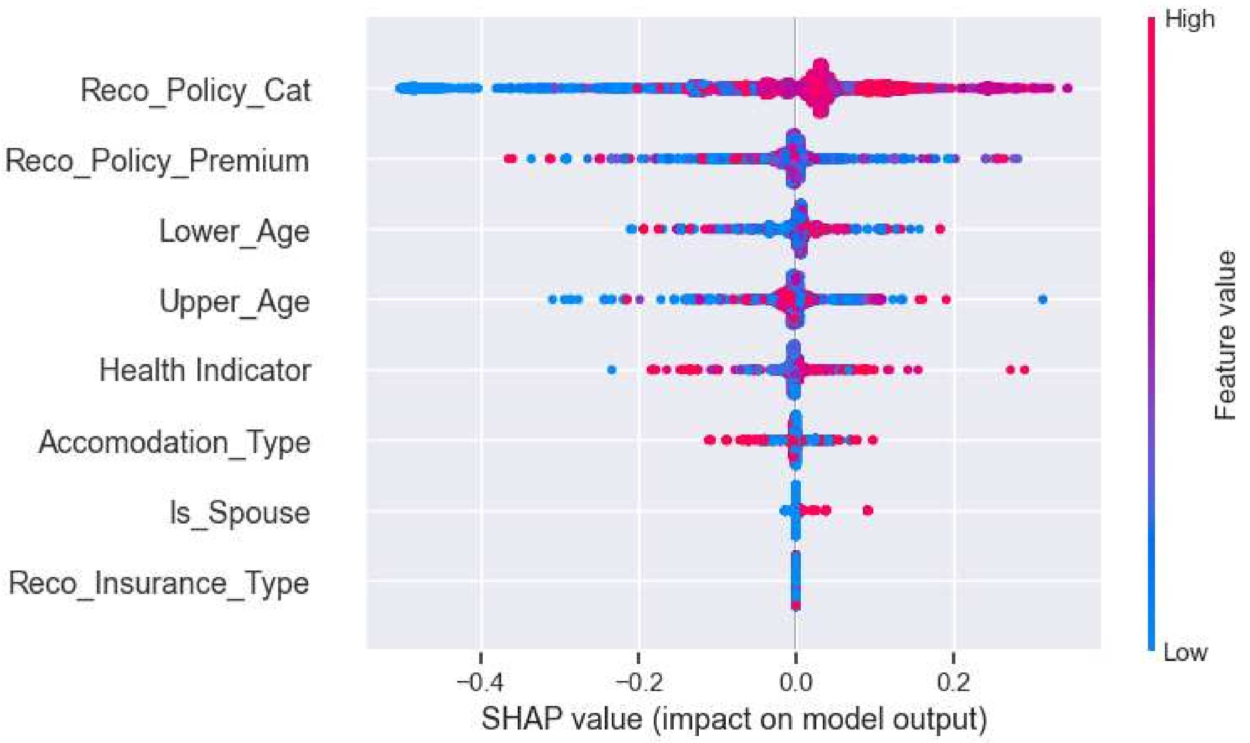
In [33]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": dtree.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Decision Tree)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```



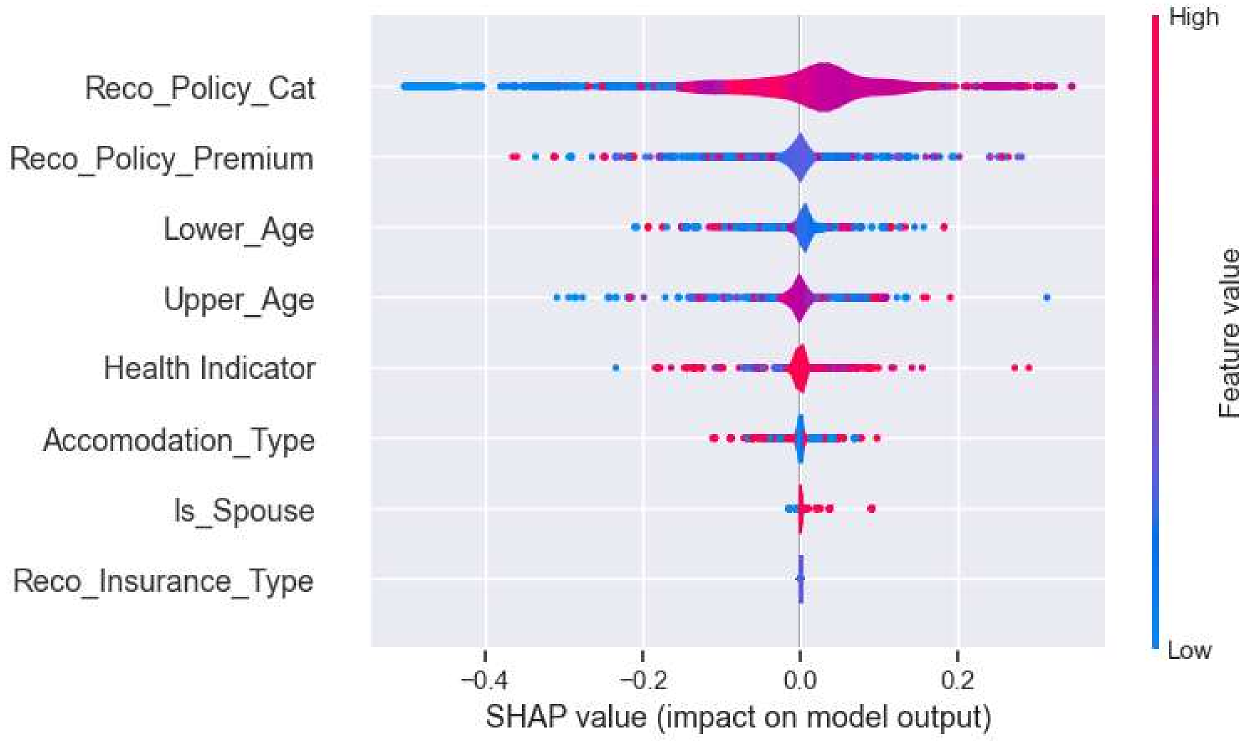
In [34]:
```
import shap
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```



In [35]:
```
# compute SHAP values
explainer = shap.TreeExplainer(dtree)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns)
```

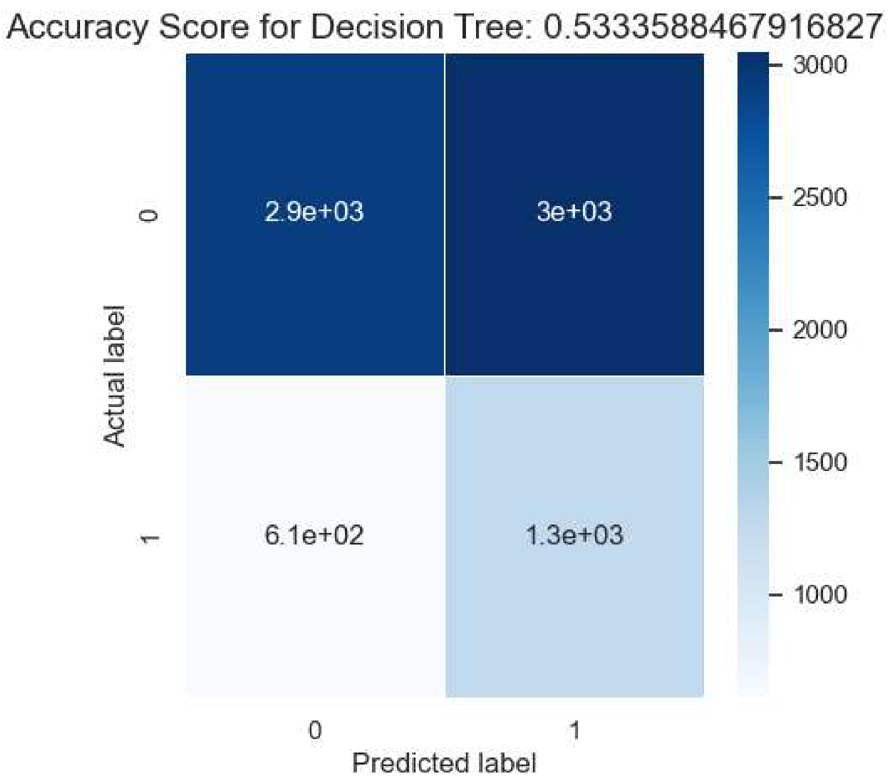


```
In [36]: # compute SHAP values
         explainer = shap.TreeExplainer(dtree)
         shap_values = explainer.shap_values(X_test)
         shap.summary_plot(shap_values[1], X_test.values, feature_names = X_test.columns, plot_type="violin"
```



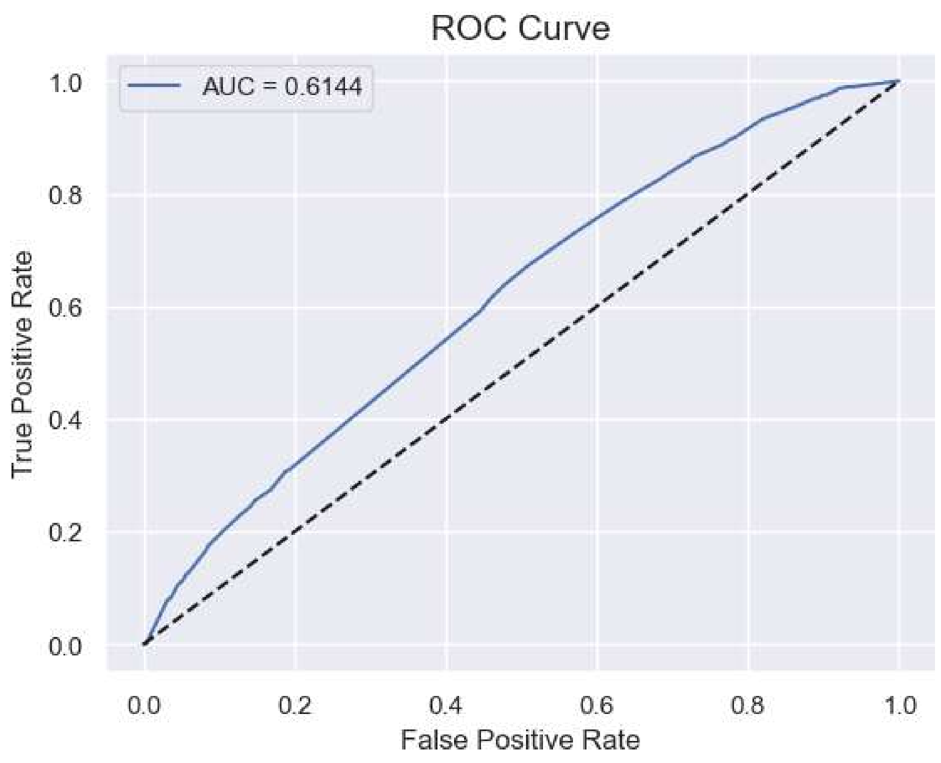
In [37]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(5,5))
sns.heatmap(data=cm,linewidths=.5, annot=True,  cmap = 'Blues')
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
all_sample_title = 'Accuracy Score for Decision Tree: {0}'.format(dtree.score(X_test, y_test))
plt.title(all_sample_title, size = 15)
```

Out[37]:
```
Text(0.5, 1.0, 'Accuracy Score for Decision Tree: 0.5333588467916827')
```



In [38]:
```
from sklearn.metrics import roc_curve, roc_auc_score
y_pred_proba = dtree.predict_proba(X_test)[:][:,1]

df_actual_predicted = pd.concat([pd.DataFrame(np.array(y_test), columns=['y_actual']), pd.DataFrame
df_actual_predicted.index = y_test.index

fpr, tpr, tr = roc_curve(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])
auc = roc_auc_score(df_actual_predicted['y_actual'], df_actual_predicted['y_pred_proba'])

plt.plot(fpr, tpr, label='AUC = %0.4f' %auc)
plt.plot(fpr, fpr, linestyle = '--', color='k')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve', size = 15)
plt.legend()
```

Out[38]:
```
<matplotlib.legend.Legend at 0x251eb9256a0>
```



Random Forest

In [39]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rfc = RandomForestClassifier(class_weight='balanced')
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'random_state': [0, 42]
}

# Perform a grid search with cross-validation to find the best hyperparameters
grid_search = GridSearchCV(rfc, param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Print the best hyperparameters
print(grid_search.best_params_)
```

```
{'max_depth': None, 'max_features': 'log2', 'n_estimators': 100, 'random_state': 42}
```

In [40]:
```
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42, max_features='log2', n_estimators=100, class_weight='
rfc.fit(X_train, y_train)
```

Out[40]:
```
RandomForestClassifier(class_weight='balanced', max_features='log2',
                       random_state=42)
```

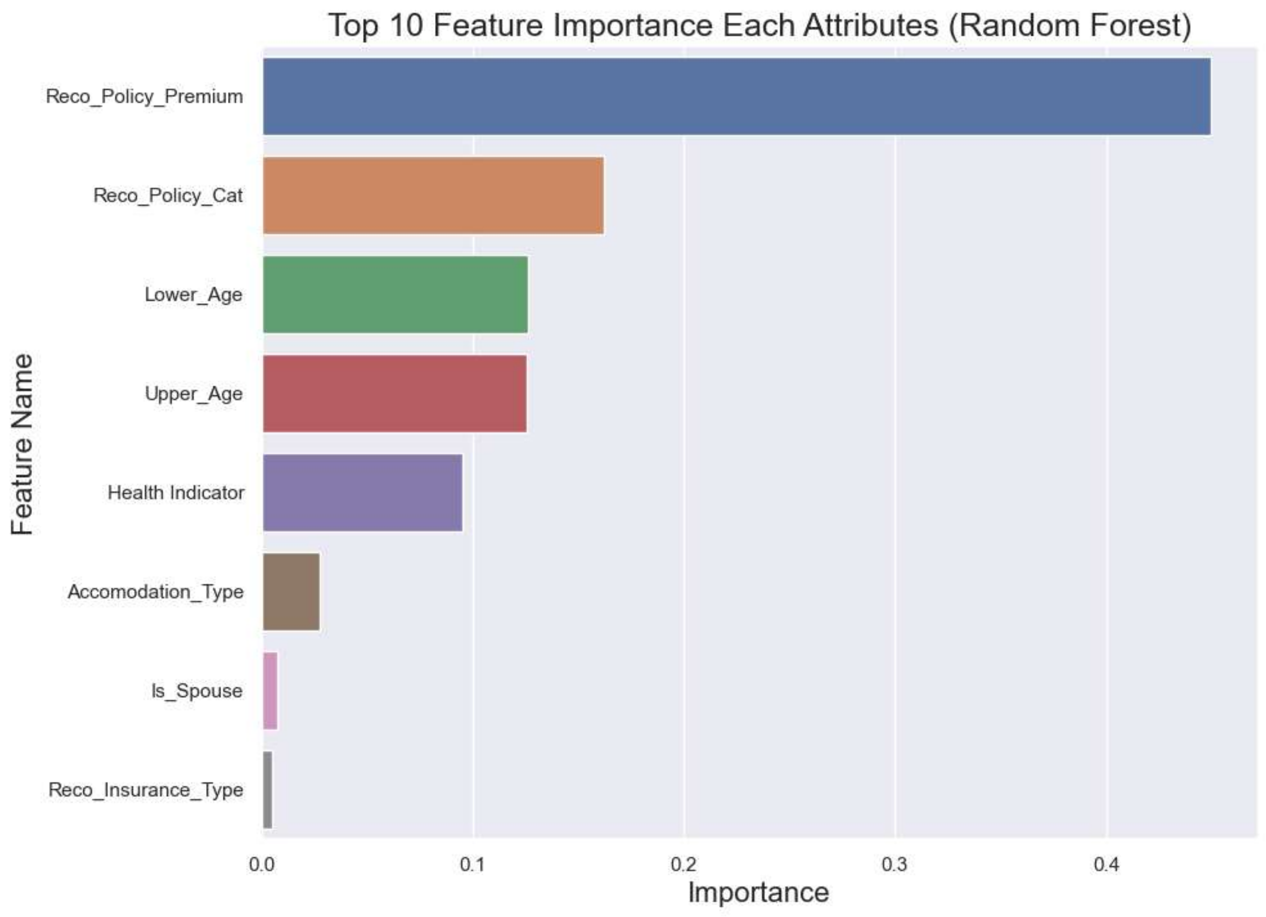
In [41]:
```
y_pred = rfc.predict(X_test)
print("Accuracy Score :", round(accuracy_score(y_test, y_pred)*100 ,2), "%")
```

```
Accuracy Score : 72.38 %
```

In [42]:
```
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_score, jaccard_score,
print('F-1 Score : ',(f1_score(y_test, y_pred, average='micro')))
print('Precision Score : ',(precision_score(y_test, y_pred, average='micro')))
print('Recall Score : ',(recall_score(y_test, y_pred, average='micro')))
print('Jaccard Score : ',(jaccard_score(y_test, y_pred, average='micro')))
print('Log Loss : ',(log_loss(y_test, y_pred)))
```

```
F-1 Score :  0.7238168133690521
Precision Score :  0.7238168133690521
Recall Score :  0.7238168133690521
Jaccard Score :  0.5671731307477009
Log Loss :  9.53908196035918
```

In [43]:
```
imp_df = pd.DataFrame({
    "Feature Name": X_train.columns,
    "Importance": rfc.feature_importances_
})
fi = imp_df.sort_values(by="Importance", ascending=False)

fi2 = fi.head(10)
plt.figure(figsize=(10,8))
sns.barplot(data=fi2, x='Importance', y='Feature Name')
plt.title('Top 10 Feature Importance Each Attributes (Random Forest)', fontsize=18)
plt.xlabel ('Importance', fontsize=16)
plt.ylabel ('Feature Name', fontsize=16)
plt.show()
```


Top 10 Feature Importance Each Attributes (Random Forest)
Reco_Policy_Premium
Reco_Policy_Cat
Lower_Age
Upper_Age
Health Indicator
Accomodation_Type
Is_Spouse
Reco_Insurance_Type
Feature Name
Importance