Health Insurance Lead Prediction Model

Contents

Double-click (or enter) to edit

Import and Define Necessities

Import Libraries and Packages

```
pip install lazypredict
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting lazypredict
  Downloading lazypredict-0.2.12-py2.py3-none-any.whl (12 kB)
Requirement already satisfied: xgboost in /usr/local/lib/python3.8/dist-packages (from lazypredict) (0.90)
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from lazypredict) (1.2.0)
Requirement already satisfied: lightgbm in /usr/local/lib/python3.8/dist-packages (from lazypredict) (2.2.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from lazypredict) (4.64.1)
Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from lazypredict) (7.1.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.8/dist-packages (from lazypredict) (1.3.5)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.8/dist-packages (from lazypredict) (1.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.8/dist-packages (from lightgbm->lazypredict) (1.7.3)
Requirement already satisfied: numpy in /usr/local/lib/python3.8/dist-packages (from lightgbm->lazypredict) (1.21.6)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.8/dist-packages (from pandas->lazypredict) (2022.7)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.8/dist-packages (from pandas->lazypredict) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.8/dist-packages (from scikit-learn->lazypredict) (3.1.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.8/dist-packages (from python-dateutil>=2.7.3->pandas->lazypredict) (1.
Installing collected packages: lazypredict
Successfully installed lazypredict-0.2.12
```

```
pip install tensorflow_addons
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting tensorflow_addons
  Downloading tensorflow_addons-0.19.0-cp38-cp38-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.1/1.1 MB 17.6 MB/s eta 0:00:00
Requirement already satisfied: packaging in /usr/local/lib/python3.8/dist-packages (from tensorflow_addons) (21.3)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.8/dist-packages (from tensorflow_addons) (2.7.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging->tensorflow_addons) (3
Installing collected packages: tensorflow_addons
Successfully installed tensorflow_addons-0.19.0
```

```
# importing the packages necessary for this assignment problem

import re
import time
import numpy as np
import pandas as pd
import math

import lazypredict
from lazypredict.Supervised import LazyClassifier

from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn import metrics
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.feature_extraction.text import CountVectorizer

from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV
from sklearn.feature_selection import SelectKBest, f_classif
```

```
from sklearn.decomposition import IncrementalPCA
from sklearn.metrics import accuracy_score

#for decision tree object
from sklearn.tree import DecisionTreeClassifier
#for checking testing results
from sklearn.metrics import classification_report, confusion_matrix
#for visualizing tree
from sklearn.tree import plot_tree

from sklearn.feature_selection import mutual_info_classif
from scipy.stats import chi2_contingency

from sklearn.preprocessing import LabelEncoder

import statsmodels.api as sm

import matplotlib.pyplot as plt
plt.rc("font", size=14)
import matplotlib.cm as cm
from mpl_toolkits.mplot3d import Axes3D
from matplotlib.ticker import AutoLocator, MaxNLocator, LinearLocator, MultipleLocator, FixedLocator, NullLocator
from sklearn.model_selection import train_test_split
import seaborn as sns
sns.set(style="white")
sns.set(style="whitegrid", color_codes=True)

import tensorflow as tf
import tensorflow_addons as tfa
print("TF version:-", tf.__version__)
import keras as k

import pickle

# and we want to view the charts inline
%matplotlib inline
```

```
TF version:- 2.9.2
```

Importing the Dataset

```
# Mount Google Drive

from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
dir_HILP = "/content/drive/MyDrive/Colab Notebooks/HILP Dataset/"
```

```
df_init = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/HILP Dataset/train.csv")
df_init.head()
```

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

Define Functions

```
# FUNCTION 1: Find and print unique values of all categorical variables in the dataframe

def print_unique_vals(df, col_list):
  i = 0
  lim = len(cat_list)

  for i in range(lim):
    print("\n Unique values for", col_list[i], "are: \n")
    print(df[col_list[i]].unique())
    i = i+1
  return


# FUNCTION 2: Visualize heatmap of correlation matrix for first cut of numerical variables

def print_correlation_matrix(df):
  plt.figure(figsize=[20,10])
  sns.heatmap(df.corr(),cmap='viridis',annot=True)
  return


# FUNCTION 3: Remove Outliers using IQR

def remove_outliers_iqr(df, column_name, threshold = 1.5):
    column = df[column_name]
    quartile_1, quartile_3 = np.percentile(column, [25, 75])
    iqr = quartile_3 - quartile_1
    lower_bound = quartile_1 - (iqr * threshold)
    upper_bound = quartile_3 + (iqr * threshold)
    df_without_outliers = df[(column > lower_bound) & (column < upper_bound)]
    df_without_outliers = df_without_outliers.reset_index()
    return df_without_outliers


# FUNCTION 4: Split Dataset and Run LazyPredict

# Split the dataset for training and testing

def split_eval_print(df):
  X = df.drop(['Response'], axis=1)
  y = df[['Response']]
  X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, random_state =123)

  # Check the scores for the dataset

  clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
  models,predictions = clf.fit(X_train, X_test, y_train, y_test)
  print(models)

  return


# FUNCTION 5: Reset dataframe after usage

def df_reset(df):
  df = df_init.copy()
  return df


# FUNCTION 6: Label encode selected variables

def label_encode(df, cols_to_encode):
  for col in cols_to_encode:
      le = LabelEncoder()
      df[col] = le.fit_transform(df[col])
      return df


# FUNCTION 7: Feature Selection by Filter Method

def feature_select(df,n):
  # Perform feature selection
  X = df.drop(columns=['Response'])
  y = df['Response']
  selector = SelectKBest(f_classif, k=n)
  selector.fit(X, y)

  # get the selected features
```

```
    selected_features = X.columns[selector.get_support()]
    return selected_features
```

Data Visualization and Exploration

Explore the Dataset

```
df_init.head()
```

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

```
# Display info regarding the Dataset
df_init.info()
```

```
<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
```

```
# Describe the dataset
df_init.describe()
```

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium	Response
count	50882.00	50882.00	50882.00	50882.00	30631.00	50882.00	50882.00	50882.00
mean	25441.50	1732.79	44.86	42.74	2.44	15.12	14183.95	0.24
std	14688.51	1424.08	17.31	17.32	1.03	6.34	6590.07	0.43
min	1.00	1.00	18.00	16.00	1.00	1.00	2280.00	0.00
25%	12721.25	523.00	28.00	27.00	1.00	12.00	9248.00	0.00
50%	25441.50	1391.00	44.00	40.00	3.00	17.00	13178.00	0.00
75%	38161.75	2667.00	59.00	57.00	3.00	20.00	18096.00	0.00
max	50882.00	6194.00	75.00	75.00	4.00	22.00	43350.40	1.00

```
# Look at the distribution of target variable
df_init['Response'].value_counts()
```

```
0    38673
1    12209
```

```
Name: Response, dtype: int64
```

View Numerical Data Distribution

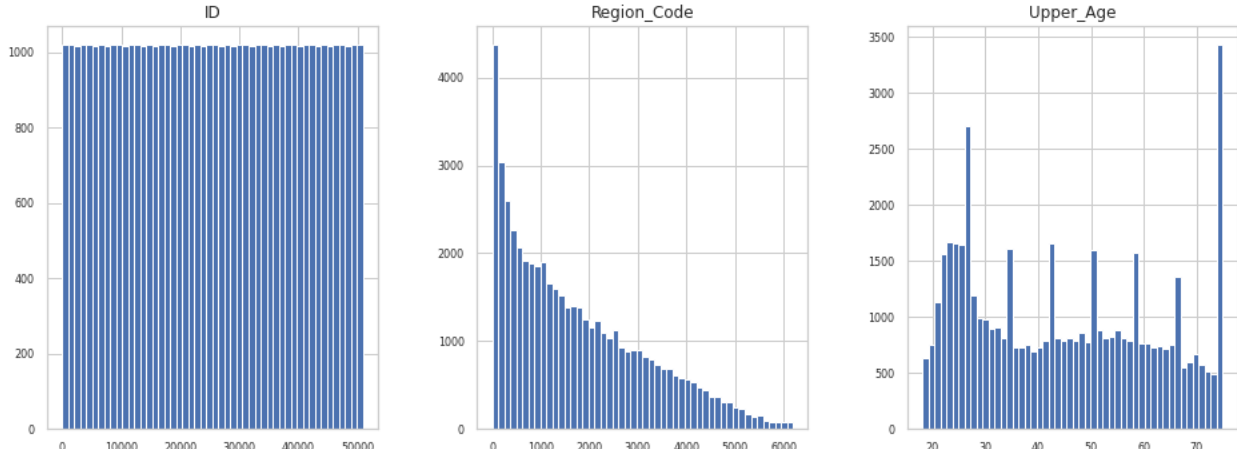
```
# Create a deep copy of current dataset
df_dtypes = df_init.copy(deep=True)

# Collect all Datatypes of the dataset into a list
dtypes_list = list(set(df_dtypes.dtypes.tolist()))
print(dtypes_list)
```

```
[dtype('float64'), dtype('int64'), dtype('O')]
```

```
# Find numeric data
df_numeric = df_dtypes.select_dtypes(include = ['float64', 'int64'], )
df_numeric.head()
```

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium	Response
0	1	3213	36	36	3.00	22	11628.00	0
1	2	1117	75	22	NaN	22	30510.00	0
2	3	3732	32	32	1.00	19	7450.00	1
3	4	4378	52	48	3.00	19	17780.00	0
4	5	2190	44	44	1.00	16	10404.00	0

```
# Plot numerical distribution
df_numeric.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8);
```



Split the Variables into Categorical and Continuous

Lower_Age Holding_Policy_Type Reco_Policy_Cat

```
continuous_vars = df_init.select_dtypes(include=[np.number])
categorical_vars = df_init.select_dtypes(include=[np.object])

continuous_vars = continuous_vars.drop(['Response'], axis=1)

# Display continuous variables
continuous_vars.head()
```

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium
0	1	3213	36	36	3.00	22	11628.00
1	2	1117	75	22	NaN	22	30510.00
2	3	3732	32	32	1.00	19	7450.00
3	4	4378	52	48	3.00	19	17780.00
4	5	2190	44	44	1.00	16	10404.00

Observations:

- *The column 'ID' can be removed as it is a system generated key.*
- *The column 'Region Code' is a categorical variable. It must be treated as such.*
- *The column 'Holding_Policy_Type' is a categorical variable. It must be treated as such.*
- *The column 'Reco_Policy_Cat' is a categorical variable. It must be treated as such.*

```
# Display categorical variables
categorical_vars.tail()
```

	City_Code	Accomodation_Type	Reco_Insurance_Type	Is_Spouse	Health Indicator	Holding_Policy_Duration
50877	C4	Rented	Individual	No	X3	NaN
50878	C5	Rented	Individual	No	X3	7.0
50879	C1	Rented	Individual	No	X2	14+
50880	C1	Owned	Joint	No	X2	2.0
50881	C3	Rented	Individual	No	X3	2.0

```
# Print Unique values of categorical columns
cat_list = list(categorical_vars.columns)
cat_list += ['Region_Code', 'Holding_Policy_Type', 'Reco_Policy_Cat']
print(cat_list)

print_unique_vals(df_init, cat_list)
```

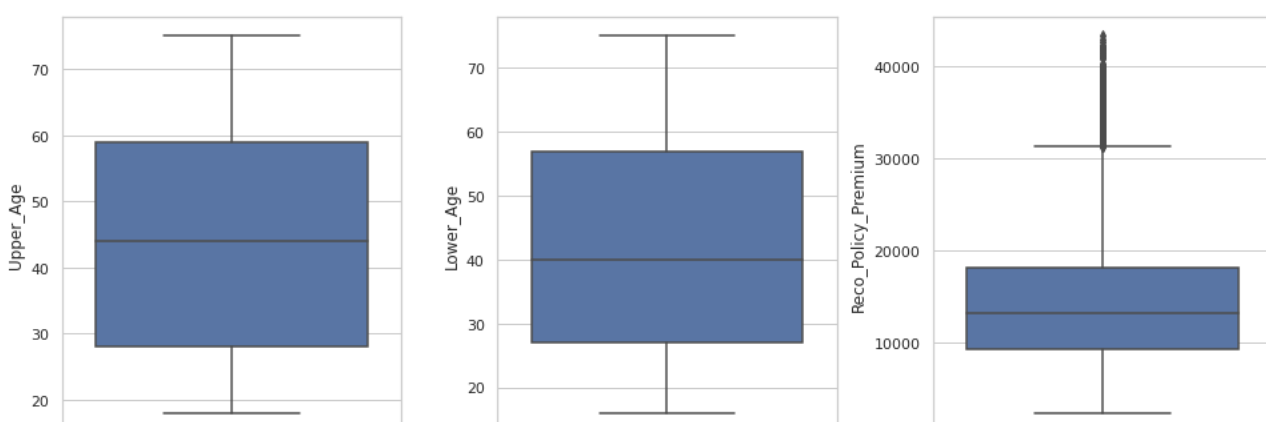
```
['City_Code', 'Accomodation_Type', 'Reco_Insurance_Type', 'Is_Spouse', 'Health Indicator', 'Holding_Policy_Duration', 'Region_Code', 'Hc

 Unique values for City_Code are:
```

```
['C3' 'C5' 'C24' 'C8' 'C9' 'C1' 'C15' 'C28' 'C27' 'C7' 'C20' 'C25' 'C4'
 'C2' 'C34' 'C10' 'C17' 'C18' 'C16' 'C29' 'C33' 'C26' 'C19' 'C6' 'C12'
 'C13' 'C11' 'C14' 'C22' 'C23' 'C21' 'C36' 'C32' 'C30' 'C35' 'C31']

 Unique values for Accomodation_Type are:

['Rented' 'Owned']

 Unique values for Reco_Insurance_Type are:

['Individual' 'Joint']

 Unique values for Is_Spouse are:

['No' 'Yes']

 Unique values for Health Indicator are:

['X1' 'X2' nan 'X4' 'X3' 'X6' 'X5' 'X8' 'X7' 'X9']

 Unique values for Holding_Policy_Duration are:

['14+' nan '1.0' '3.0' '5.0' '9.0' '14.0' '7.0' '2.0' '11.0' '10.0' '8.0'
 '6.0' '4.0' '13.0' '12.0']

 Unique values for Region_Code are:

[3213 1117 3732 ... 5326 6149 5450]

 Unique values for Holding_Policy_Type are:

[ 3. nan  1.  4.  2.]

 Unique values for Reco_Policy_Cat are:

[22 19 16 17  1 18 21 13 20  9  2  4 12  6 14 11  3  8  7 10 15  5]
```

Observation: One-hot encoding can be aptly applied on all columns except Holding_Policy_Duration. For this column, we will apply either binning or label encoding. nan values in Holding_Policy_Duration will be considered as someone who is holding policy for < 1 year duration.

Detect and Analyze Outliers

```
# Create boxplots of truly numerical features to check for outliers

continuous_vars = continuous_vars.drop(['ID', 'Region_Code', 'Holding_Policy_Type', 'Reco_Policy_Cat'], axis=1)

fig_box = plt.figure(figsize=(18,18))
limit = len(continuous_vars.columns)

for i in range(limit):
    fig_box.add_subplot(4, 4, i+1)
    sns.boxplot(y=continuous_vars.iloc[:,i])

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



Observation: Outliers have been detected in Reco_Policy_Premium. Let's check if this has a pattern.

```
# Look at the distribution of target variable
df_init['Response'].value_counts()
```

```
0    38673
1    12209
Name: Response, dtype: int64
```

Observation: Within the full total, 24% of the responders have accepted to opt-in and the rest 76% did not.

```
# Look at the distribution of target variable for the outlier values
df_init.query('Reco_Policy_Premium > 32000')['Response'].value_counts()
```

```
0    527
1    168
Name: Response, dtype: int64
```

Observations:

- *Within the outliers, ~24% of the responders have accepted to opt-in and the rest ~76% did not. This is consistent with the total distribution. It is reasonable to assume that the outliers do not have a high correlation with the target variable.*
- *Thus, we can safely remove them using z-scores.*
- *The rest of the values can be scaled for this column.*

Find Missing Values

```
# Find missing values
missing_values = df_init.isnull().sum()

# Print missing values
print(missing_values)
```

```
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
```

```
# Find missing values
missing_values = df_init.isnull().sum()

# Calculate percentage of missing values
missing_values_percent = (missing_values / len(df_init)) * 100

# Print percentage of missing values
print(missing_values_percent)
```

```
ID                          0.00
City_Code                   0.00
Region_Code                 0.00
Accomodation_Type           0.00
Reco_Insurance_Type         0.00
Upper_Age                   0.00
Lower_Age                   0.00
Is_Spouse                   0.00
Health Indicator           22.98
Holding_Policy_Duration    39.80
Holding_Policy_Type        39.80
Reco_Policy_Cat             0.00
Reco_Policy_Premium         0.00
```

```
Response                  0.00
dtype: float64
```

```
# View distinct values of Health Indicator column
df_init['Health Indicator'].unique()
```

```
array(['X1', 'X2', nan, 'X4', 'X3', 'X6', 'X5', 'X8', 'X7', 'X9'],
      dtype=object)
```

```
# find rows where 'Health Indicator' column is null
null_rows = df_init[df_init['Health Indicator'].isnull()]
print(null_rows)
```

```
          ID City_Code  Region_Code Accomodation_Type Reco_Insurance_Type  \
2          3        C5         3732             Owned          Individual
6          7        C3          679             Owned          Individual
9         10        C1          530             Owned               Joint
12        13        C7         3453             Owned          Individual
19        20       C20          973             Owned          Individual
...      ...       ...          ...               ...                 ...
50859  50860        C1          217             Owned          Individual
50865  50866       C21         4915             Owned          Individual
50869  50870       C11         1770            Rented          Individual
50871  50872       C10          224            Rented          Individual
50876  50877       C26          579             Owned          Individual

       Upper_Age  Lower_Age Is_Spouse Health Indicator  \
2             32         32        No              NaN
6             28         28        No              NaN
9             59         26       Yes              NaN
12            66         66        No              NaN
19            27         27        No              NaN
...          ...        ...       ...              ...
50859         70         70        No              NaN
50865         74         74        No              NaN
50869         45         45        No              NaN
50871         21         21        No              NaN
50876         37         37        No              NaN

      Holding_Policy_Duration  Holding_Policy_Type  Reco_Policy_Cat  \
2                         1.0                 1.00               19
6                         NaN                  NaN               17
9                         7.0                 4.00               18
12                        1.0                 2.00               20
19                        NaN                  NaN                4
...                       ...                  ...              ...
50859                     6.0                 3.00               20
50865                     NaN                  NaN               14
50869                     1.0                 1.00               20
50871                     1.0                 1.00               13
50876                     2.0                 1.00               12

       Reco_Policy_Premium  Response
2                  7450.00         1
6                 10640.00         0
9                 21100.80         1
12                17192.00         1
19                 8050.00         0
...                    ...       ...
50859             19448.00         0
50865             19944.00         0
50869             10944.00         0
50871             11840.00         0
50876             13222.00         0

[11691 rows x 14 columns]
```

Three columns - Health Indicator, Holding_Policy_Duration and Holdin_Policy_Type have missing values. Their % of misses is 22.98%, 39.8% and 39.8% respectively.

All of these are categorical values. It is not known what the absence of these values indicate. Thus, it might be worth checking if the absence of these values themselves is an indicator for target prediction.

Thus, the decision is to perform one-hot encoding on the categorical variables and assume a bucket (non-entity) to indicate the absence.

Correlation Analysis

```
print_correlation_matrix(df_init)
```

Observation: We do not see a good correlation for any of the predictor variables (numerical) with the target variable. Let's do this exercise again after feature encoding of categorical variables.

Data Preparation

Here, we will experiment with methods to handle missing values, outliers and feature selection. The efficacy of the different methods will be tested using lazypredict. This will help us find the optimal method to be applied in each case.

Drop Unwanted Columns

```
# Drop the ID Column as it does not have any investigative value - DIMENSIONALITY REDUCTION
# This is common for all cases below so we will apply this before moving on

df_init = df_init.drop(['ID'], axis=1)
```

Remove Outliers

```
df_without_outliers = remove_outliers_iqr(df_init, "Reco_Policy_Premium")
df_without_outliers.tail()
```

index	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Holding_Policy

```
df_without_outliers.describe()
```

	index	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium	Response
count	50061.00	50061.00	50061.00	50061.00	30052.00	50061.00	50061.00	50061.00
mean	25450.52	1734.88	44.43	42.50	2.43	15.10	13855.81	0.24
std	14690.14	1424.57	17.12	17.23	1.03	6.35	6113.80	0.43
min	0.00	1.00	18.00	16.00	1.00	1.00	2280.00	0.00
25%	12728.00	526.00	28.00	26.00	1.00	12.00	9184.00	0.00
50%	25459.00	1392.00	43.00	40.00	3.00	17.00	13046.00	0.00
75%	38170.00	2670.00	59.00	57.00	3.00	20.00	17820.00	0.00
max	50881.00	6194.00	75.00	75.00	4.00	22.00	31365.00	1.00

Scale the Values

```
# Select columns to scale
columns_to_scale = ['Upper_Age', 'Lower_Age', 'Reco_Policy_Premium']
data_to_scale = df_without_outliers[columns_to_scale]

scaler = MinMaxScaler()
scaled_values = scaler.fit_transform(data_to_scale)

# Update the selected columns with the scaled values
df_without_outliers[columns_to_scale] = scaled_values
df_without_outliers.tail()
```

	index	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Holding_Policy_
50056	50877	C4	845	Rented	Individual	0.07	0.10	No	X3	
50057	50878	C5	4188	Rented	Individual	0.16	0.19	No	X3	
50058	50879	C1	442	Rented	Individual	0.79	0.80	No	X2	
50059	50880	C1	4	Owned	Joint	0.93	0.56	No	X2	
50060	50881	C3	3866	Rented	Individual	0.11	0.14	No	X3	

One-Hot Encoding

```
# Specify the columns to be one-hot encoded
cols_to_onehot_encode = ['City_Code', 'Region_Code', 'Accomodation_Type', 'Reco_Insurance_Type', 'Is_Spouse', 'Health Indicator', 'Holding_Po

# Perform one-hot encoding on the specified columns
df_encoded = pd.get_dummies(df_without_outliers, columns=cols_to_onehot_encode)

print(list(df_encoded.columns))

# Drop redundant columns
df_encoded = df_encoded.drop(['index', 'City_Code_C1', 'Region_Code_3213', 'Accomodation_Type_Rented', 'Reco_Insurance_Type_Individual', 'Is_

# Print the encoded dataframe
df_encoded.tail()
```

```
['index', 'Upper_Age', 'Lower_Age', 'Holding_Policy_Duration', 'Reco_Policy_Premium', 'Response', 'City_Code_C1', 'City_Code_C10', 'City
```

	Upper_Age	Lower_Age	Holding_Policy_Duration	Reco_Policy_Premium	Response	City_Code_C10	City_Code_C11	City_Code_C12	City_C
50056	0.07	0.10	NaN	0.19	0	0	0	0	
50057	0.16	0.19	7.0	0.11	0	0	0	0	
50058	0.79	0.80	14+	0.31	0	0	0	0	
50059	0.93	0.56	2.0	0.89	1	0	0	0	

Feature Encoding

```
# Label Encoding

cols_to_label_encode = ['Holding_Policy_Duration']
df_fully_encoded = label_encode(df_encoded, cols_to_label_encode)

# Print the encoded dataframe
df_fully_encoded.tail()
```

	Upper_Age	Lower_Age	Holding_Policy_Duration	Reco_Policy_Premium	Response	City_Code_C10	City_Code_C11	City_Code_C12	City_C
50056	0.07	0.10	15	0.19	0	0	0	0	
50057	0.16	0.19	12	0.11	0	0	0	0	
50058	0.79	0.80	5	0.31	0	0	0	0	
50059	0.93	0.56	7	0.89	1	0	0	0	
50060	0.11	0.14	7	0.31	0	0	0	0	

5 rows × 5384 columns

Case 1: Model Experimentation with Original Dataset

```
split_eval_print(df_init)
```

```
100%|██████████| 29/29 [03:49<00:00,  7.91s/it]                 Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
DecisionTreeClassifier             0.67               0.55     0.55      0.67
BaggingClassifier                  0.74               0.53     0.53      0.69
ExtraTreeClassifier                0.65               0.52     0.52      0.65
LabelPropagation                   0.65               0.51     0.51      0.65
LabelSpreading                     0.65               0.51     0.51      0.65
SGDClassifier                      0.47               0.51     0.51      0.51
NearestCentroid                    0.48               0.51     0.51      0.52
Perceptron                         0.64               0.51     0.51      0.64
ExtraTreesClassifier               0.75               0.51     0.51      0.67
KNeighborsClassifier               0.71               0.51     0.51      0.67
LGBMClassifier                     0.76               0.51     0.51      0.67
RandomForestClassifier             0.76               0.50     0.50      0.67
QuadraticDiscriminantAnalysis      0.48               0.50     0.50      0.52
PassiveAggressiveClassifier        0.57               0.50     0.50      0.60
GaussianNB                         0.76               0.50     0.50      0.66
XGBClassifier                      0.76               0.50     0.50      0.66
RidgeClassifierCV                  0.76               0.50     0.50      0.66
RidgeClassifier                    0.76               0.50     0.50      0.66
SVC                                0.76               0.50     0.50      0.66
AdaBoostClassifier                 0.76               0.50     0.50      0.66
LogisticRegression                 0.76               0.50     0.50      0.66
LinearDiscriminantAnalysis         0.76               0.50     0.50      0.66
DummyClassifier                    0.76               0.50     0.50      0.66
CalibratedClassifierCV             0.76               0.50     0.50      0.66
BernoulliNB                        0.76               0.50     0.50      0.66
LinearSVC                          0.76               0.50     0.50      0.66

                               Time Taken
Model
DecisionTreeClassifier               0.48
BaggingClassifier                    2.41
ExtraTreeClassifier                  0.15
LabelPropagation                    24.26
LabelSpreading                      37.76
```

```
SGDClassifier                      0.89
NearestCentroid                    0.17
Perceptron                         0.21
ExtraTreesClassifier               4.18
KNeighborsClassifier               8.35
LGBMClassifier                     0.61
RandomForestClassifier             5.75
QuadraticDiscriminantAnalysis      0.23
PassiveAggressiveClassifier        0.16
GaussianNB                         0.13
XGBClassifier                     55.90
RidgeClassifierCV                  0.28
RidgeClassifier                    0.16
SVC                               65.11
AdaBoostClassifier                 1.94
LogisticRegression                 0.54
LinearDiscriminantAnalysis         0.31
DummyClassifier                    0.11
CalibratedClassifierCV            14.42
BernoulliNB                        0.15
LinearSVC                          4.30
```

Feature Selection

```
# Perform feature selection
selected_features = feature_select(df_fully_encoded, 100)
print(selected_features)
```

```
Index(['Region_Code_55', 'Region_Code_100', 'Region_Code_150',
       'Region_Code_253', 'Region_Code_411', 'Region_Code_838',
       'Region_Code_959', 'Region_Code_989', 'Region_Code_1263',
       'Region_Code_1321', 'Region_Code_1426', 'Region_Code_1457',
       'Region_Code_1578', 'Region_Code_1627', 'Region_Code_1666',
       'Region_Code_1744', 'Region_Code_1788', 'Region_Code_1833',
       'Region_Code_1841', 'Region_Code_1890', 'Region_Code_1979',
       'Region_Code_2048', 'Region_Code_2112', 'Region_Code_2181',
       'Region_Code_2230', 'Region_Code_2384', 'Region_Code_2463',
       'Region_Code_2561', 'Region_Code_2661', 'Region_Code_2689',
       'Region_Code_2694', 'Region_Code_2707', 'Region_Code_2737',
       'Region_Code_2748', 'Region_Code_3001', 'Region_Code_3079',
       'Region_Code_3119', 'Region_Code_3277', 'Region_Code_3281',
       'Region_Code_3318', 'Region_Code_3451', 'Region_Code_3456',
       'Region_Code_3458', 'Region_Code_3557', 'Region_Code_3660',
       'Region_Code_3709', 'Region_Code_3726', 'Region_Code_3752',
       'Region_Code_3789', 'Region_Code_3794', 'Region_Code_3863',
       'Region_Code_3890', 'Region_Code_4058', 'Region_Code_4085',
       'Region_Code_4102', 'Region_Code_4157', 'Region_Code_4240',
       'Region_Code_4279', 'Region_Code_4316', 'Region_Code_4364',
       'Region_Code_4371', 'Region_Code_4562', 'Region_Code_4586',
       'Region_Code_4592', 'Region_Code_4666', 'Region_Code_4678',
       'Region_Code_4704', 'Region_Code_4785', 'Region_Code_4797',
       'Region_Code_4812', 'Region_Code_5029', 'Region_Code_5154',
       'Region_Code_5194', 'Region_Code_5208', 'Region_Code_5277',
       'Region_Code_5313', 'Region_Code_5333', 'Region_Code_5349',
       'Region_Code_5352', 'Region_Code_5474', 'Region_Code_5500',
       'Region_Code_5512', 'Region_Code_5514', 'Health Indicator_X7',
       'Reco_Policy_Cat_1', 'Reco_Policy_Cat_2', 'Reco_Policy_Cat_3',
       'Reco_Policy_Cat_4', 'Reco_Policy_Cat_5', 'Reco_Policy_Cat_6',
       'Reco_Policy_Cat_7', 'Reco_Policy_Cat_9', 'Reco_Policy_Cat_10',
       'Reco_Policy_Cat_11', 'Reco_Policy_Cat_12', 'Reco_Policy_Cat_15',
       'Reco_Policy_Cat_17', 'Reco_Policy_Cat_18', 'Reco_Policy_Cat_19',
       'Reco_Policy_Cat_21'],
      dtype='object')
```

```
df_feature_selected = df_fully_encoded[selected_features]
df_feature_selected = df_feature_selected.assign(Response = df_fully_encoded['Response'])
df_feature_selected.tail()
```

	Region_Code_55	Region_Code_100	Region_Code_150	Region_Code_253	Region_Code_411	Region_Code_838	Region_Code_959	Region_Code
50056	0	0	0	0	0	0	0	
50057	0	0	0	0	0	0	0	

Case 2: Model Experimentation After Feature Selection

```
split_eval_print(df_feature_selected)
```

```
100%|██████████| 29/29 [04:33<00:00,  9.42s/it]                    Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
NearestCentroid                   0.67               0.56     0.56      0.67
PassiveAggressiveClassifier       0.74               0.54     0.54      0.69
Perceptron                        0.71               0.53     0.53      0.68
KNeighborsClassifier              0.74               0.52     0.52      0.68
QuadraticDiscriminantAnalysis     0.76               0.51     0.51      0.67
GaussianNB                        0.76               0.51     0.51      0.67
SVC                               0.76               0.51     0.51      0.67
BernoulliNB                       0.76               0.51     0.51      0.67
LinearDiscriminantAnalysis        0.76               0.51     0.51      0.67
LogisticRegression                0.76               0.51     0.51      0.67
RidgeClassifierCV                 0.76               0.51     0.51      0.67
RidgeClassifier                   0.76               0.51     0.51      0.67
LabelPropagation                  0.76               0.51     0.51      0.66
LabelSpreading                    0.76               0.51     0.51      0.66
LinearSVC                         0.76               0.51     0.51      0.66
CalibratedClassifierCV            0.76               0.51     0.51      0.66
DecisionTreeClassifier            0.76               0.51     0.51      0.66
SGDClassifier                     0.76               0.51     0.51      0.66
BaggingClassifier                 0.76               0.51     0.51      0.66
RandomForestClassifier            0.76               0.50     0.50      0.66
ExtraTreesClassifier              0.76               0.50     0.50      0.66
ExtraTreeClassifier               0.76               0.50     0.50      0.66
AdaBoostClassifier                0.76               0.50     0.50      0.66
LGBMClassifier                    0.76               0.50     0.50      0.66
XGBClassifier                     0.76               0.50     0.50      0.66
DummyClassifier                   0.76               0.50     0.50      0.66

                               Time Taken
Model
NearestCentroid                      0.16
PassiveAggressiveClassifier          0.30
Perceptron                           0.22
KNeighborsClassifier                 7.51
QuadraticDiscriminantAnalysis        0.32
GaussianNB                           0.16
SVC                                 84.11
BernoulliNB                          0.18
LinearDiscriminantAnalysis           0.67
LogisticRegression                   0.30
RidgeClassifierCV                    0.41
RidgeClassifier                      0.18
LabelPropagation                    24.26
LabelSpreading                      36.86
LinearSVC                           20.58
CalibratedClassifierCV              75.48
DecisionTreeClassifier               0.37
SGDClassifier                        1.07
BaggingClassifier                    1.69
RandomForestClassifier               3.56
ExtraTreesClassifier                 4.75
ExtraTreeClassifier                  0.17
AdaBoostClassifier                   1.69
LGBMClassifier                       1.07
XGBClassifier                        6.65
DummyClassifier                      0.11
```

Case 3: Dimensionality Reduction

```
# Remove Target Variable
df_pre_ipca = df_fully_encoded.copy()
df_pre_ipca = df_pre_ipca.drop(['Response'], axis=1)

# Using Principal Component Analysis (PCA) for Dimensionality Reduction (n=10)
ipca_10 = IncrementalPCA(n_components=10, batch_size=10)
```

```
df_ipca_10 = ipca_10.fit_transform(df_pre_ipca)
print(ipca_10.components_)
```

```
[[-1.24972159e-02 -1.09632960e-02  9.98892809e-01 ... -2.59805763e-04
   2.17965386e-04 -1.93203522e-04]
 [ 3.09420545e-01  2.35935823e-01  2.19946338e-02 ... -1.30417185e-02
  -9.97813185e-03  2.66179984e-02]
 [-1.46886915e-01 -2.25077892e-01 -8.80305764e-03 ...  5.48383484e-02
   2.63035219e-02 -6.18537371e-02]
 ...
 [ 4.00997504e-01  3.95886098e-01  1.53169565e-02 ...  1.05342429e-01
  -7.60629968e-02 -1.03016342e-01]
 [-3.40202388e-01 -3.43982922e-01  1.15229640e-03 ...  1.45460380e-01
  -1.16825078e-01  3.60930675e-02]
 [-4.03583668e-02 -3.61957607e-02 -1.15119723e-03 ...  3.63565366e-02
  -1.54619139e-02 -4.18331659e-01]]
```

```
# Using Principal Component Analysis (PCA) for Dimensionality Reduction (n=3)
ipca_3 = IncrementalPCA(n_components=3, batch_size=10)
df_ipca_3 = ipca_3.fit_transform(df_pre_ipca)
print(ipca_3.components_)
```

```
[[-1.24960716e-02 -1.09621542e-02  9.98892782e-01 ... -2.60479815e-04
   2.18276863e-04 -1.93498204e-04]
 [ 3.10746739e-01  2.37258970e-01  2.19596227e-02 ... -1.40067698e-02
  -9.92947184e-03  2.54634907e-02]
 [-1.29294721e-01 -1.92352787e-01 -1.83262559e-03 ... -1.87635488e-02
  -4.54259120e-03  2.11312356e-02]]
```

```
# Convert the 10-darray to a DataFrame
df_ipca_10 = pd.DataFrame(df_ipca_10)

# Add response variable
df_ipca_10 = df_ipca_10.assign(Response = df_fully_encoded['Response'])

df_ipca_10.head()
```

	0	1	2	3	4	5	6	7	8	9	Response
0	-5.12	-0.62	-0.25	0.65	0.75	0.22	-0.40	-0.08	-0.27	-0.08	0
1	4.87	0.92	0.28	-0.50	-0.19	0.56	-0.10	0.04	-0.05	-0.04	0
2	-10.12	-0.42	0.23	-0.08	-0.80	-0.08	-0.24	-0.10	0.80	0.18	1
3	-5.14	0.60	-0.04	0.82	0.39	0.21	-0.40	0.02	0.29	-0.09	0
4	-2.12	-0.65	0.38	-0.68	-0.46	0.65	-0.21	0.61	0.06	-0.17	0

```
# Convert the 3-darray to a DataFrame
df_ipca_3 = pd.DataFrame(df_ipca_3)

# Add response variable
df_ipca_3 = df_ipca_3.assign(Response = df_fully_encoded['Response'])

df_ipca_3.head()
```

	0	1	2	Response
0	-5.12	-0.62	0.72	0
1	4.87	0.92	-0.35	0
2	-10.12	-0.42	-0.40	1
3	-5.14	0.60	0.72	0
4	-2.12	-0.65	-0.58	0

Case 3: Model Experimentation After Dimensionality Reduction

```
split_eval_print(df_ipca_10)
```

```
100%|██████████| 29/29 [06:02<00:00, 12.51s/it]            Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
DecisionTreeClassifier            0.64               0.52     0.52      0.64
ExtraTreesClassifier              0.72               0.51     0.51      0.67
LabelPropagation                  0.70               0.51     0.51      0.67
```

```
LabelSpreading                  0.70              0.51     0.51      0.67
RandomForestClassifier          0.74              0.51     0.51      0.67
KNeighborsClassifier            0.71              0.51     0.51      0.67
BaggingClassifier               0.73              0.51     0.51      0.67
ExtraTreeClassifier             0.63              0.51     0.51      0.64
Perceptron                      0.71              0.50     0.50      0.66
LGBMClassifier                  0.76              0.50     0.50      0.66
PassiveAggressiveClassifier     0.59              0.50     0.50      0.61
NearestCentroid                 0.51              0.50     0.50      0.55
QuadraticDiscriminantAnalysis   0.76              0.50     0.50      0.66
RidgeClassifierCV               0.76              0.50     0.50      0.66
SGDClassifier                   0.76              0.50     0.50      0.66
RidgeClassifier                 0.76              0.50     0.50      0.66
SVC                             0.76              0.50     0.50      0.66
XGBClassifier                   0.76              0.50     0.50      0.66
LinearSVC                       0.76              0.50     0.50      0.66
LogisticRegression              0.76              0.50     0.50      0.66
LinearDiscriminantAnalysis      0.76              0.50     0.50      0.66
GaussianNB                      0.76              0.50     0.50      0.66
DummyClassifier                 0.76              0.50     0.50      0.66
CalibratedClassifierCV          0.76              0.50     0.50      0.66
BernoulliNB                     0.76              0.50     0.50      0.66
AdaBoostClassifier              0.76              0.50     0.50      0.66

                               Time Taken
Model
DecisionTreeClassifier               1.06
ExtraTreesClassifier                 4.41
LabelPropagation                    23.56
LabelSpreading                      37.42
RandomForestClassifier              22.18
KNeighborsClassifier                 0.86
BaggingClassifier                    6.81
ExtraTreeClassifier                  0.07
Perceptron                           0.08
LGBMClassifier                       0.54
PassiveAggressiveClassifier          0.07
NearestCentroid                      0.04
QuadraticDiscriminantAnalysis        0.05
RidgeClassifierCV                    0.12
SGDClassifier                        0.18
RidgeClassifier                      0.04
SVC                                248.14
XGBClassifier                        2.61
LinearSVC                            1.98
LogisticRegression                   0.06
LinearDiscriminantAnalysis           0.13
GaussianNB                           0.04
DummyClassifier                      0.03
CalibratedClassifierCV               8.86
BernoulliNB                          0.04
AdaBoostClassifier                   3.25
```

```
split_eval_print(df_ipca_3)
```

```
100%|██████████| 29/29 [01:48<00:00,  3.73s/it]                    Accuracy  Balanced Accuracy  ROC AUC  F1 Score  \
Model
DecisionTreeClassifier          0.64              0.51     0.51      0.64
ExtraTreesClassifier            0.72              0.51     0.51      0.67
RandomForestClassifier          0.73              0.51     0.51      0.67
KNeighborsClassifier            0.71              0.50     0.50      0.66
NearestCentroid                 0.52              0.50     0.50      0.56
BaggingClassifier               0.72              0.50     0.50      0.66
Perceptron                      0.25              0.50     0.50      0.11
ExtraTreeClassifier             0.63              0.50     0.50      0.64
LabelPropagation                0.76              0.50     0.50      0.66
LabelSpreading                  0.76              0.50     0.50      0.66
LogisticRegression              0.76              0.50     0.50      0.66
RidgeClassifierCV               0.76              0.50     0.50      0.66
RidgeClassifier                 0.76              0.50     0.50      0.66
QuadraticDiscriminantAnalysis   0.76              0.50     0.50      0.66
LinearSVC                       0.76              0.50     0.50      0.66
LinearDiscriminantAnalysis      0.76              0.50     0.50      0.66
SVC                             0.76              0.50     0.50      0.66
GaussianNB                      0.76              0.50     0.50      0.66
DummyClassifier                 0.76              0.50     0.50      0.66
CalibratedClassifierCV          0.76              0.50     0.50      0.66
BernoulliNB                     0.76              0.50     0.50      0.66
SGDClassifier                   0.76              0.50     0.50      0.66
XGBClassifier                   0.76              0.50     0.50      0.66
LGBMClassifier                  0.76              0.50     0.50      0.66
AdaBoostClassifier              0.76              0.50     0.50      0.66
PassiveAggressiveClassifier     0.64              0.50     0.50      0.64
```

```
                               Time Taken
Model
DecisionTreeClassifier               0.49
ExtraTreesClassifier                 2.69
RandomForestClassifier              10.87
KNeighborsClassifier                 0.39
NearestCentroid                      0.04
BaggingClassifier                    2.99
Perceptron                           0.07
ExtraTreeClassifier                  0.05
LabelPropagation                    20.30
LabelSpreading                      29.66
LogisticRegression                   0.05
RidgeClassifierCV                    0.08
RidgeClassifier                      0.04
QuadraticDiscriminantAnalysis        0.04
LinearSVC                            0.82
LinearDiscriminantAnalysis           0.08
SVC                                 33.47
GaussianNB                           0.03
DummyClassifier                      0.02
CalibratedClassifierCV               3.02
BernoulliNB                          0.03
SGDClassifier                        0.13
XGBClassifier                        1.15
LGBMClassifier                       0.29
AdaBoostClassifier                   1.31
PassiveAggressiveClassifier          0.06
```

Try Deep Learning

```
THRESHOLD = .999
bestModelPath = './best_model.hdf5'

class myCallback(k.callbacks.Callback):
    def on_epoch_end(self, epoch, logs={}):
        if(logs.get('val_accuracy') > THRESHOLD):
            print("\n\nStopping training as we have reached our goal.")
            self.model.stop_training = True

mycb = myCallback()
checkpoint = k.callbacks.ModelCheckpoint(filepath=bestModelPath, monitor='val_loss', verbose=1, save_best_only=True)

callbacks_list = [mycb,checkpoint]
```

Case 4a: Model Experimentation with Deep Learning with PCA=10

```
X = df_ipca_10.drop(['Response'], axis=1)
y = df_ipca_10[['Response']]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.25, random_state =123)


epochs = 40

model_1 = k.models.Sequential([k.layers.Dense(2048, activation='relu', input_shape=(X_train.shape[1],)),
                               k.layers.Dense(1024, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(512, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(128, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(1, activation='sigmoid'),])
print(model_1.summary())

model_1.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history_1 = model_1.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=epochs, batch_size=2048, callbacks=[callbacks_list])
```

```
Epoch 29/40
19/19 [==============================] - ETA: 0s - loss: 0.5407 - accuracy: 0.7603
Epoch 29: val_loss did not improve from 0.54922
19/19 [==============================] - 22s 1s/step - loss: 0.5407 - accuracy: 0.7603 - val_loss: 0.5534 - val_accuracy: 0.7595
Epoch 30/40
19/19 [==============================] - ETA: 0s - loss: 0.5406 - accuracy: 0.7603
Epoch 30: val_loss did not improve from 0.54922
19/19 [==============================] - 15s 768ms/step - loss: 0.5406 - accuracy: 0.7603 - val_loss: 0.5528 - val_accuracy: 0.7595
Epoch 31/40
19/19 [==============================] - ETA: 0s - loss: 0.5397 - accuracy: 0.7603
Epoch 31: val_loss did not improve from 0.54922
19/19 [==============================] - 15s 804ms/step - loss: 0.5397 - accuracy: 0.7603 - val_loss: 0.5530 - val_accuracy: 0.7595
Epoch 32/40
19/19 [==============================] - ETA: 0s - loss: 0.5393 - accuracy: 0.7603
Epoch 32: val_loss did not improve from 0.54922
19/19 [==============================] - 14s 758ms/step - loss: 0.5393 - accuracy: 0.7603 - val_loss: 0.5505 - val_accuracy: 0.7595
Epoch 33/40
19/19 [==============================] - ETA: 0s - loss: 0.5393 - accuracy: 0.7603
Epoch 33: val_loss did not improve from 0.54922
19/19 [==============================] - 22s 1s/step - loss: 0.5393 - accuracy: 0.7603 - val_loss: 0.5514 - val_accuracy: 0.7595
Epoch 34/40
19/19 [==============================] - ETA: 0s - loss: 0.5393 - accuracy: 0.7603
Epoch 34: val_loss did not improve from 0.54922
19/19 [==============================] - 19s 1s/step - loss: 0.5393 - accuracy: 0.7603 - val_loss: 0.5514 - val_accuracy: 0.7595
Epoch 35/40
19/19 [==============================] - ETA: 0s - loss: 0.5375 - accuracy: 0.7603
Epoch 35: val_loss did not improve from 0.54922
19/19 [==============================] - 19s 969ms/step - loss: 0.5375 - accuracy: 0.7603 - val_loss: 0.5548 - val_accuracy: 0.7595
Epoch 36/40
19/19 [==============================] - ETA: 0s - loss: 0.5372 - accuracy: 0.7603
Epoch 36: val_loss did not improve from 0.54922
19/19 [==============================] - 15s 792ms/step - loss: 0.5372 - accuracy: 0.7603 - val_loss: 0.5528 - val_accuracy: 0.7595
Epoch 37/40
19/19 [==============================] - ETA: 0s - loss: 0.5374 - accuracy: 0.7603
Epoch 37: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 688ms/step - loss: 0.5374 - accuracy: 0.7603 - val_loss: 0.5554 - val_accuracy: 0.7595
Epoch 38/40
19/19 [==============================] - ETA: 0s - loss: 0.5368 - accuracy: 0.7603
Epoch 38: val_loss did not improve from 0.54922
19/19 [==============================] - 20s 1s/step - loss: 0.5368 - accuracy: 0.7603 - val_loss: 0.5543 - val_accuracy: 0.7595
Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.5370 - accuracy: 0.7604
Epoch 39: val_loss did not improve from 0.54922
19/19 [==============================] - 20s 1s/step - loss: 0.5370 - accuracy: 0.7604 - val_loss: 0.5571 - val_accuracy: 0.7595
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.5368 - accuracy: 0.7603
Epoch 40: val_loss did not improve from 0.54922
19/19 [==============================] - 15s 781ms/step - loss: 0.5368 - accuracy: 0.7603 - val_loss: 0.5534 - val_accuracy: 0.7595
```

Case 4b: Model Experimentation with Deep Learning with PCA=3

```
X_pca3 = df_ipca_3.drop(['Response'], axis=1)
y_pca3 = df_ipca_3[['Response']]
X_train_pca3, X_test_pca3, y_train_pca3, y_test_pca3 = train_test_split(X_pca3, y_pca3, test_size=.25, random_state =123)

epochs = 40

model_2 = k.models.Sequential([k.layers.Dense(2048, activation='relu', input_shape=(X_train_pca3.shape[1],)),
                               k.layers.Dense(1024, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(512, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(128, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(1, activation='sigmoid'),])
print(model_2.summary())

model_2.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history_2 = model_2.fit(X_train_pca3, y_train_pca3, validation_data=(X_test_pca3, y_test_pca3), epochs=epochs, batch_size=2048, callbacks=[ca
```

```
Epoch 30/40
19/19 [==============================] - ETA: 0s - loss: 0.5523 - accuracy: 0.7603
Epoch 30: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 671ms/step - loss: 0.5523 - accuracy: 0.7603 - val_loss: 0.5515 - val_accuracy: 0.7595
Epoch 31/40
19/19 [==============================] - ETA: 0s - loss: 0.5527 - accuracy: 0.7603
Epoch 31: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 677ms/step - loss: 0.5527 - accuracy: 0.7603 - val_loss: 0.5538 - val_accuracy: 0.7595
Epoch 32/40
19/19 [==============================] - ETA: 0s - loss: 0.5527 - accuracy: 0.7603
Epoch 32: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 672ms/step - loss: 0.5527 - accuracy: 0.7603 - val_loss: 0.5519 - val_accuracy: 0.7595
Epoch 33/40
19/19 [==============================] - ETA: 0s - loss: 0.5531 - accuracy: 0.7603
Epoch 33: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 671ms/step - loss: 0.5531 - accuracy: 0.7603 - val_loss: 0.5519 - val_accuracy: 0.7595
Epoch 34/40
19/19 [==============================] - ETA: 0s - loss: 0.5516 - accuracy: 0.7603
Epoch 34: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 670ms/step - loss: 0.5516 - accuracy: 0.7603 - val_loss: 0.5523 - val_accuracy: 0.7595
Epoch 35/40
19/19 [==============================] - ETA: 0s - loss: 0.5517 - accuracy: 0.7603
Epoch 35: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 670ms/step - loss: 0.5517 - accuracy: 0.7603 - val_loss: 0.5528 - val_accuracy: 0.7595
Epoch 36/40
19/19 [==============================] - ETA: 0s - loss: 0.5514 - accuracy: 0.7603
Epoch 36: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 678ms/step - loss: 0.5514 - accuracy: 0.7603 - val_loss: 0.5519 - val_accuracy: 0.7595
Epoch 37/40
19/19 [==============================] - ETA: 0s - loss: 0.5515 - accuracy: 0.7603
Epoch 37: val_loss did not improve from 0.54922
19/19 [==============================] - 15s 758ms/step - loss: 0.5515 - accuracy: 0.7603 - val_loss: 0.5541 - val_accuracy: 0.7595
Epoch 38/40
19/19 [==============================] - ETA: 0s - loss: 0.5518 - accuracy: 0.7603
Epoch 38: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 669ms/step - loss: 0.5518 - accuracy: 0.7603 - val_loss: 0.5516 - val_accuracy: 0.7595
Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.5512 - accuracy: 0.7603
Epoch 39: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 674ms/step - loss: 0.5512 - accuracy: 0.7603 - val_loss: 0.5517 - val_accuracy: 0.7595
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.5514 - accuracy: 0.7603
Epoch 40: val_loss did not improve from 0.54922
19/19 [==============================] - 13s 669ms/step - loss: 0.5514 - accuracy: 0.7603 - val_loss: 0.5524 - val_accuracy: 0.7595
```

Case 4c: Model Experimentation with Deep Learning with Selected Features

```
X_feature_selected = df_feature_selected.drop(['Response'], axis=1)
y_feature_selected = df_feature_selected[['Response']]
X_train_feature_selected, X_test_feature_selected, y_train_feature_selected, y_test_feature_selected = train_test_split(X_feature_selected, y

epochs = 40

model_3 = k.models.Sequential([k.layers.Dense(2048, activation='relu', input_shape=(X_feature_selected.shape[1],)),
                               k.layers.Dense(1024, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(512, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(128, activation='relu'), k.layers.Dropout(0.2),
                               k.layers.Dense(1, activation='sigmoid'),])
print(model_3.summary())

model_3.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history_3 = model_3.fit(X_train_feature_selected, y_train_feature_selected, validation_data=(X_test_feature_selected, y_test_feature_selected
```

```
19/19 [==============================] - 14s 716ms/step - loss: 0.5217 - accuracy: 0.7660 - val_loss: 0.5373 - val_accuracy: 0.7613
Epoch 31/40
19/19 [==============================] - ETA: 0s - loss: 0.5224 - accuracy: 0.7663
Epoch 31: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 712ms/step - loss: 0.5224 - accuracy: 0.7663 - val_loss: 0.5369 - val_accuracy: 0.7613
Epoch 32/40
19/19 [==============================] - ETA: 0s - loss: 0.5224 - accuracy: 0.7662
Epoch 32: val_loss did not improve from 0.52764
19/19 [==============================] - 15s 804ms/step - loss: 0.5224 - accuracy: 0.7662 - val_loss: 0.5370 - val_accuracy: 0.7613
Epoch 33/40
19/19 [==============================] - ETA: 0s - loss: 0.5225 - accuracy: 0.7661
Epoch 33: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 710ms/step - loss: 0.5225 - accuracy: 0.7661 - val_loss: 0.5413 - val_accuracy: 0.7613
Epoch 34/40
19/19 [==============================] - ETA: 0s - loss: 0.5225 - accuracy: 0.7659
Epoch 34: val_loss did not improve from 0.52764
19/19 [==============================] - 14s 736ms/step - loss: 0.5225 - accuracy: 0.7659 - val_loss: 0.5369 - val_accuracy: 0.7610
Epoch 35/40
19/19 [==============================] - ETA: 0s - loss: 0.5231 - accuracy: 0.7661
Epoch 35: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 708ms/step - loss: 0.5231 - accuracy: 0.7661 - val_loss: 0.5424 - val_accuracy: 0.7616
Epoch 36/40
19/19 [==============================] - ETA: 0s - loss: 0.5235 - accuracy: 0.7660
Epoch 36: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 708ms/step - loss: 0.5235 - accuracy: 0.7660 - val_loss: 0.5385 - val_accuracy: 0.7614
Epoch 37/40
19/19 [==============================] - ETA: 0s - loss: 0.5223 - accuracy: 0.7660
Epoch 37: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 704ms/step - loss: 0.5223 - accuracy: 0.7660 - val_loss: 0.5384 - val_accuracy: 0.7614
Epoch 38/40
19/19 [==============================] - ETA: 0s - loss: 0.5217 - accuracy: 0.7661
Epoch 38: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 709ms/step - loss: 0.5217 - accuracy: 0.7661 - val_loss: 0.5395 - val_accuracy: 0.7616
Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.5217 - accuracy: 0.7662
Epoch 39: val_loss did not improve from 0.52764
19/19 [==============================] - 14s 720ms/step - loss: 0.5217 - accuracy: 0.7662 - val_loss: 0.5422 - val_accuracy: 0.7613
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.5217 - accuracy: 0.7661
Epoch 40: val_loss did not improve from 0.52764
19/19 [==============================] - 13s 711ms/step - loss: 0.5217 - accuracy: 0.7661 - val_loss: 0.5396 - val_accuracy: 0.7614
```

Case 4d: Model Experimentation with Deep Learning with Encoded Dataset

```
X_orig = df_fully_encoded.drop(['Response'], axis=1)
y_orig = df_fully_encoded[['Response']]
X_train_orig, X_test_orig, y_train_orig, y_test_orig = train_test_split(X_orig, y_orig, test_size=.25, random_state =123)

epochs = 40

model_4 = k.models.Sequential([k.layers.Dense(2048, activation='relu', input_shape=(X_orig.shape[1],)),
                              k.layers.Dense(1024, activation='relu'), k.layers.Dropout(0.2),
                              k.layers.Dense(512, activation='relu'), k.layers.Dropout(0.2),
                              k.layers.Dense(128, activation='relu'), k.layers.Dropout(0.2),
                              k.layers.Dense(1, activation='sigmoid'),])
print(model_4.summary())

model_4.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
history_4 = model_4.fit(X_train_orig, y_train_orig, validation_data=(X_test_orig, y_test_orig), epochs=epochs, batch_size=2048, callbacks=[ca
```

```
19/19 [==============================] - 46s 2s/step - loss: 0.0175 - accuracy: 0.9930 - val_loss: 2.1666 - val_accuracy: 0.6854
Epoch 32/40
19/19 [==============================] - ETA: 0s - loss: 0.0180 - accuracy: 0.9926
Epoch 32: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.0180 - accuracy: 0.9926 - val_loss: 1.9231 - val_accuracy: 0.7185
Epoch 33/40
19/19 [==============================] - ETA: 0s - loss: 0.0151 - accuracy: 0.9942
Epoch 33: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.0151 - accuracy: 0.9942 - val_loss: 1.9135 - val_accuracy: 0.7159
Epoch 34/40
19/19 [==============================] - ETA: 0s - loss: 0.0135 - accuracy: 0.9943
Epoch 34: val_loss did not improve from 0.52764
19/19 [==============================] - 49s 3s/step - loss: 0.0135 - accuracy: 0.9943 - val_loss: 2.1291 - val_accuracy: 0.7192
Epoch 35/40
19/19 [==============================] - ETA: 0s - loss: 0.0124 - accuracy: 0.9949
Epoch 35: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.0124 - accuracy: 0.9949 - val_loss: 2.1197 - val_accuracy: 0.7220
Epoch 36/40
19/19 [==============================] - ETA: 0s - loss: 0.0119 - accuracy: 0.9948
Epoch 36: val_loss did not improve from 0.52764
19/19 [==============================] - 45s 2s/step - loss: 0.0119 - accuracy: 0.9948 - val_loss: 2.1130 - val_accuracy: 0.7284
Epoch 37/40
19/19 [==============================] - ETA: 0s - loss: 0.0142 - accuracy: 0.9944
Epoch 37: val_loss did not improve from 0.52764
19/19 [==============================] - 45s 2s/step - loss: 0.0142 - accuracy: 0.9944 - val_loss: 2.0628 - val_accuracy: 0.7383
Epoch 38/40
19/19 [==============================] - ETA: 0s - loss: 0.0140 - accuracy: 0.9946
Epoch 38: val_loss did not improve from 0.52764
19/19 [==============================] - 48s 3s/step - loss: 0.0140 - accuracy: 0.9946 - val_loss: 2.0085 - val_accuracy: 0.7061
Epoch 39/40
19/19 [==============================] - ETA: 0s - loss: 0.0110 - accuracy: 0.9957
Epoch 39: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.0110 - accuracy: 0.9957 - val_loss: 2.1969 - val_accuracy: 0.7230
Epoch 40/40
19/19 [==============================] - ETA: 0s - loss: 0.0100 - accuracy: 0.9951
Epoch 40: val_loss did not improve from 0.52764
19/19 [==============================] - 46s 2s/step - loss: 0.0100 - accuracy: 0.9951 - val_loss: 2.3880 - val_accuracy: 0.7229
```

```
test_loss, test_acc = model_4.evaluate(X_test_orig, y_test_orig)
```

```
392/392 [==============================] - 12s 30ms/step - loss: 2.3880 - accuracy: 0.7229
```

```
train_loss, train_acc = model_4.evaluate(X_train_orig, y_train_orig)
```

```
1174/1174 [==============================] - 34s 29ms/step - loss: 0.0096 - accuracy: 0.9958
```

Model Creation