Customer Segmentation based in a Life Insurance Industry Using Machine Learning Algorithms

Final Report

P. Vyshnavi

B. Gnan

P. Kranthi Kumar

P. Mohan Sai Krishna Reddy

**Introduction**

Insurance policies give a sense of security to the policy holder in case of uneventful things in life such as death, disability.

At the same time the policy issuer has to make sure they do not end up losing money if there are more claims paid out than the premiums collected for insuring people.

This project will use various data analysis techniques to predict the risk of insuring each customer based on risk factors such as age, weight, height, BMI, sex, medical history, employment history etc.

Predicted response is the classification of applicants into eight classes

|  |  |
| --- | --- |
| **Variable** | **Description** |
| **Id** | A unique identifier associated with an application. |
| **Product\_Info\_1-7** | A set of normalized variables relating to the product applied for |
| **Ins\_Age** | Normalized age of applicant |
| **Ht** | Normalized height of applicant |
| **Wt** | Normalized weight of applicant |
| **BMI** | Normalized BMI of applicant |
| **Employment\_Info\_1-6** | A set of normalized variables relating to the employment history of the applicant. |
| **InsuredInfo\_1-6** | A set of normalized variables providing information about the applicant. |
| **Insurance\_History\_1-9** | A set of normalized variables relating to the insurance history of the applicant. |
| **Family\_Hist\_1-5** | A set of normalized variables relating to the family history of the applicant. |
| **Medical\_History\_1-41** | A set of normalized variables relating to the medical history of the applicant. |
| **Medical\_Keyword\_1-48** | A set of dummy variables relating to the presence of/absence of a medical keyword being associated with the application. |
| **Response** | This is the target variable, an ordinal variable relating to the final decision associated with an application |

**Dataset – Life Insurance data**

Table 1: Dataset Features

The dataset used is Life Insurance application classified into eight different categories.

It is having a total of 59,381 rows with 128 features.

**Exploratory Data Analysis**

As per the given Data set, many features were already label/One hot encoded. Few of the features were already normalized.

So, the EDA of the given dataset is as follows:

1. Dropping of Insignificant features.
2. One hot Encoding of categorical Variables
3. Missing Value Imputation.
4. Visualization, Detection and Treatment of Outliers.
5. Checking for Skewness.
6. Data Balancing.

**Dropping of Insignificant features:**

Insignificant features like Id, Ht, Wt, etc were dropped from the given data set.

**One hot Encoding of categorical Variables:**

Product\_Info\_2 feature in the given dataset is having categorical variables which was One hot Encoded increasing the features to 135.

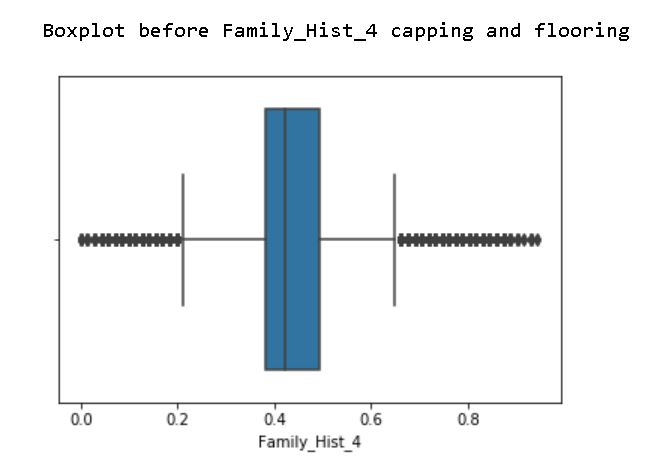
**Missing Value Imputation:**

Features having missing values >50% were dropped from the given dataset.

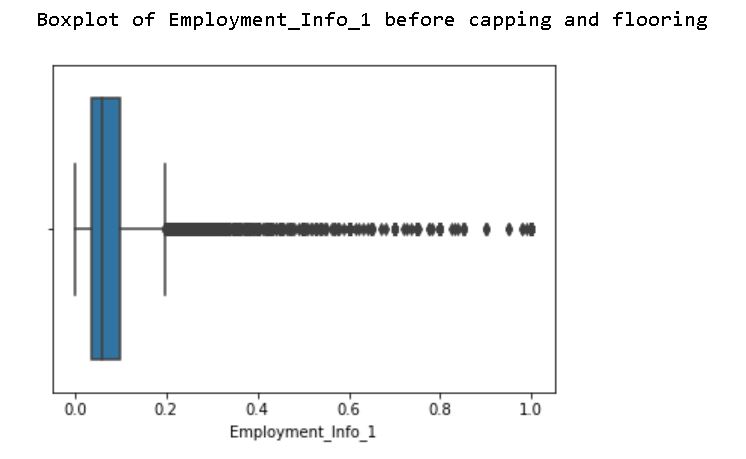
Missing values in the given data set were identified and imputed using mean of the respective features as per the structure of the features.

**Visualization, Detection and Treatment of Outliers:**

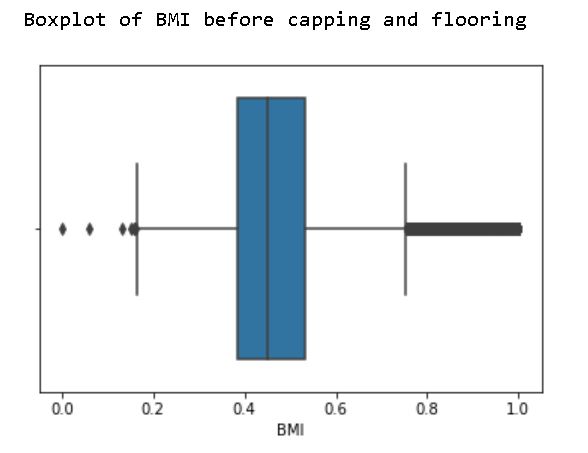
Visualization:



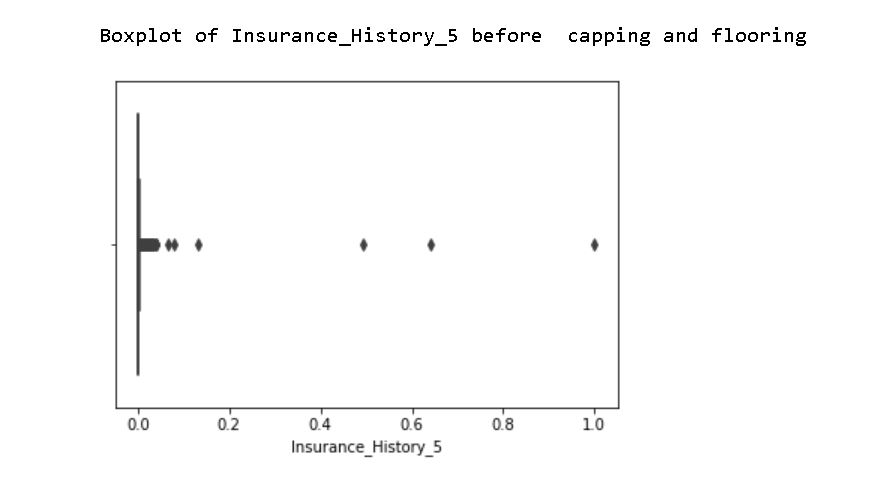
**Fig:1**



**Fig:2**



**Fig:3**



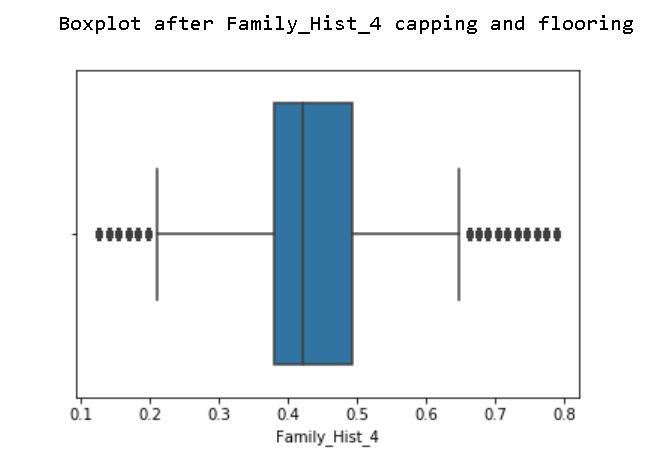
**Fig:4**

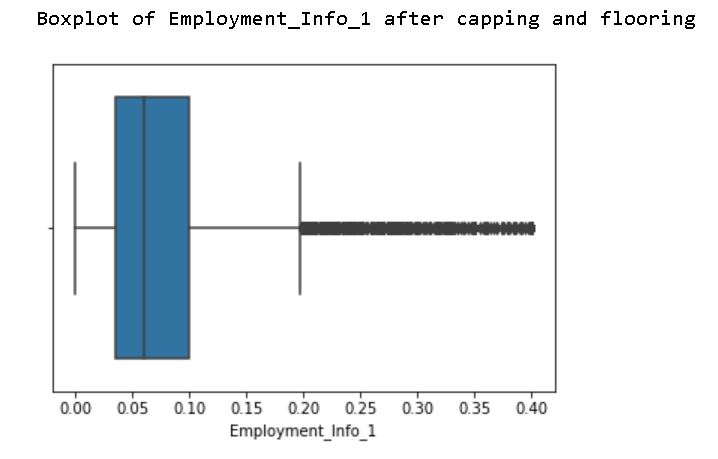
Detection:

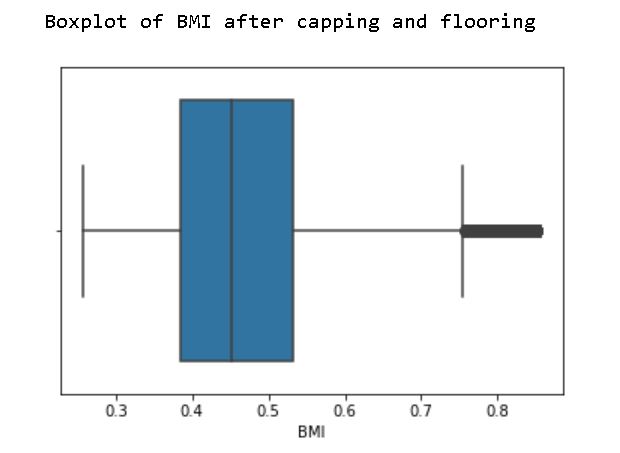
Continuous features in the given dataset were having many outliers in the given dataset, which were visualized.

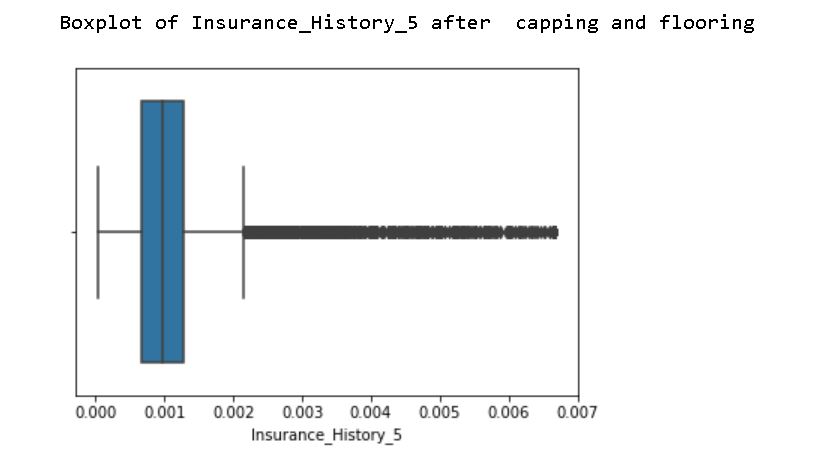
Treatment of Outliers:

Outlier Treatment is done by Capping and Flooring techniques for the respective features having outliers.





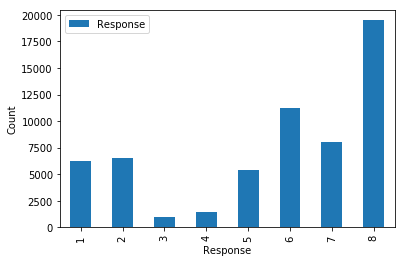




**Checking for Skewness:**

Post Outlier treatment, skewness of the features was checked and found to be normally distributed.

**Data balancing:**



**Fig:5**

The Response of the given dataset is having 8 classes which shows the risk levels of each and every customer. The distribution of various risk levels is found to be imbalanced.

So, Synthetic Minority Oversampling Technique(SMOTE) is used for balancing various risk levels in the response.

**Dimensionality Reduction**

The given data is having high multicollinearity between independent features. So Principle Component Analysis is used to optimize Multicollinearity there by achieving Dimensionality Reduction.

**Model Building**

Preliminary Machine Learning Models like Decision Tree Classifier and Random Forest Classifier were constructed on the Dataset and the following metrics were found.

The least accuracy was found in Gaussian Naive Bayes algorithm and the highest was found in Random Forest Classifier after applying PCA.

**Appendix**

**Raw code:**

import pandas as pd

import numpy as np

import warnings

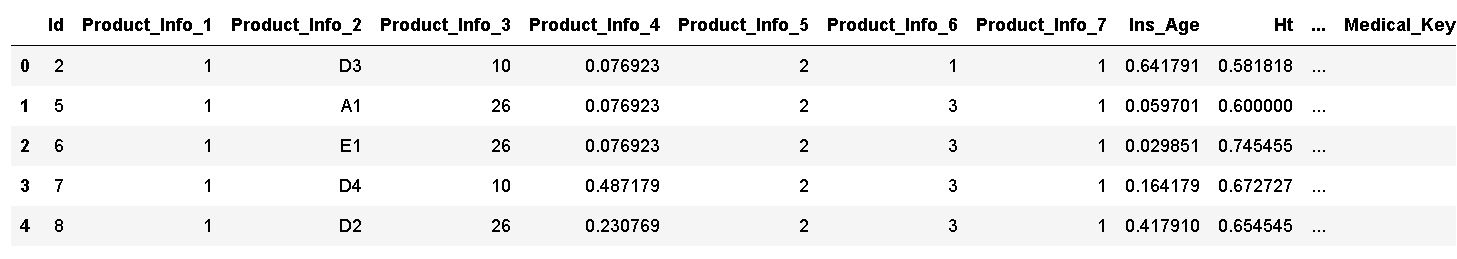
warnings.filterwarnings("ignore")

**Input:**

d=pd.read\_csv('DATA.csv')

d.head()

**output:**

****

**# Finding the percent of null values**

x=list(d.columns)

for y in range (0,128):

a=d[x[y]].isnull().sum()

if a>0:

b=(a/59381)\*100

print(x[y],' = ',b,'%')

y=y+1

**Output:**

Employment\_Info\_1 = 0.031996766642528755 %

Employment\_Info\_4 = 11.41610952998434 %

Employment\_Info\_6 = 18.278573954631952 %

Insurance\_History\_5 = 42.7678887186137 %

Family\_Hist\_2 = 48.25786025833179 %

Family\_Hist\_3 = 57.66322561088564 %

Family\_Hist\_4 = 32.3066300668564 %

Family\_Hist\_5 = 70.41141105740894 %

Medical\_History\_1 = 14.969434667654639 %

Medical\_History\_10 = 99.06198952526903 %

Medical\_History\_15 = 75.1014634310638 %

Medical\_History\_24 = 93.59896263114464 %

Medical\_History\_32 = 98.1357673329853 %

**#Dropping the columns which have more than 50% of null values and are insignificant**

**Input:**

d.drop(['Id','Ht','Wt','Family\_Hist\_2','Family\_Hist\_3','Family\_Hist\_5','Medical\_History\_10','Medical\_History\_15','Medical\_History\_24','Medical\_History\_32'],axis=1,inplace=True)

**# Imputing null values in features with less than 50 % null values**

**Input:**

d['Employment\_Info\_1'].fillna(d['Employment\_Info\_1'].median(),inplace=True)

d['Employment\_Info\_4'].fillna(d['Employment\_Info\_4'].median(),inplace=True)

d['Employment\_Info\_6'].fillna(d['Employment\_Info\_6'].median(),inplace=True)

d['Insurance\_History\_5'].fillna(d['Insurance\_History\_5'].median(),inplace=True)

d['Family\_Hist\_4'].fillna(d['Family\_Hist\_4'].median(),inplace=True)

d['Medical\_History\_1'].fillna(1,inplace=True)

**#converting the correlation matrix into dataframe**

d1=pd.DataFrame(d.corr())

**# Extracting the columns which have correlation more than 0.4**

df = d1.where(np.triu(np.ones(d1.shape)).astype(np.bool))

a=list(df.columns)

b=a

for y in range(0,117):

q=list(df[a[y]])

for i in q:

if i> 0.4 and a[y]!=b[q.index(i)]:

print(a[y],'--',b[q.index(i)],' = ',i)

**Output:**

InsuredInfo\_7 -- InsuredInfo\_2 = 0.5087155135728652

Insurance\_History\_4 -- Insurance\_History\_1 = 0.5672153896999304

Insurance\_History\_7 -- Insurance\_History\_1 = 0.4586056676825122

Insurance\_History\_7 -- Insurance\_History\_4 = 0.9195259814197974

Insurance\_History\_8 -- Insurance\_History\_1 = 0.7877234559145828

Insurance\_History\_9 -- Insurance\_History\_1 = 0.5085844516104937

Insurance\_History\_9 -- Insurance\_History\_4 = 0.9386544745126755

Insurance\_History\_9 -- Insurance\_History\_7 = 0.9625280994167774

Family\_Hist\_4 -- Ins\_Age = 0.8952612373182529

Medical\_History\_36 -- Medical\_History\_25 = 0.9541101722846125

Medical\_Keyword\_1 -- Medical\_History\_16 = 0.46476010319662225

Medical\_Keyword\_16 -- Medical\_History\_18 = 0.4689211178690135

Medical\_Keyword\_22 -- Medical\_History\_21 = 0.536244883658995

Medical\_Keyword\_30 -- Medical\_History\_19 = 0.8409466745887633

Medical\_Keyword\_34 -- Medical\_History\_18 = 0.6053300357526505

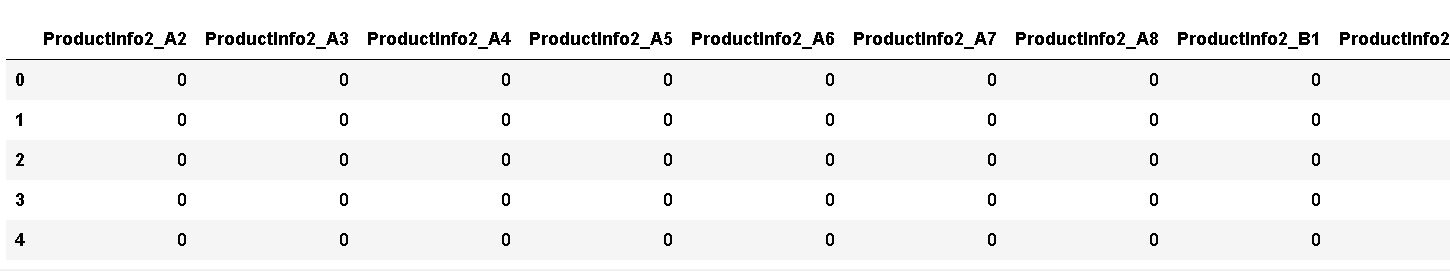
Medical\_Keyword\_41 -- Medical\_History\_30 = 0.4283274496571706

**#performing dummification on feature product\_info\_2**

x=pd.get\_dummies(d['Product\_Info\_2'],prefix='ProductInfo2',drop\_first=True)

x.head()

Output:



**#concatenating the one hot encoded dataframe and original dataset dataframe**

d=pd.concat([d,x],axis=1)

**#dropping the feature which is onehot encoded**

d.drop('Product\_Info\_2',axis=1,inplace=True)

**#Finding the lower and upper boundaries for features which have outliers**

j=[]

l=[]

u=[]

a=list(d.columns)

for y in range(0,134):

q3=d[a[y]].quantile(0.75)

q1=d[a[y]].quantile(0.25)

iqr=q3-q1

ul=q3+1.5\*iqr

ll=q1-1.5\*iqr

if ul!=ll:

for i in d[a[y]]:

if i>ul or i<ll:

j.append(a[y])

l.append(ll)

u.append(ul)

print('Columns with outliers',set(j))

print('lower boundary of columns',set(l))

print('Upper boundary of columns',set(u))

**Output:**

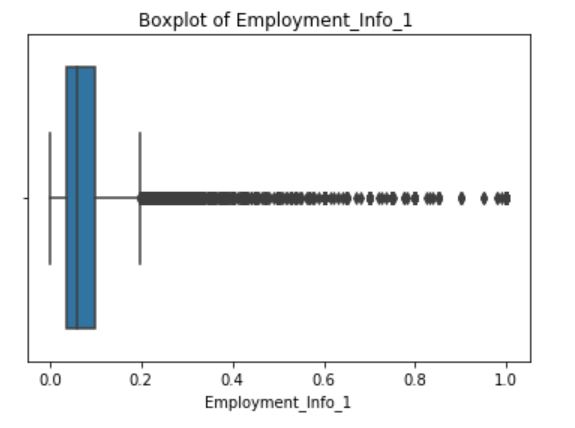
Features with outliers {'Employment\_Info\_1', 'Medical\_History\_1', 'Insurance\_History\_5', 'Family\_Hist\_4', 'BMI'}

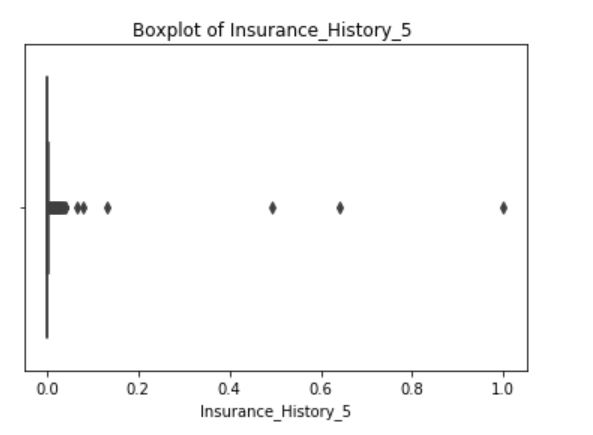
lower boundary of features {0.1645042030000001, -0.0625, 0.21126760599999997, -0.00023333299999999996, -9.5}

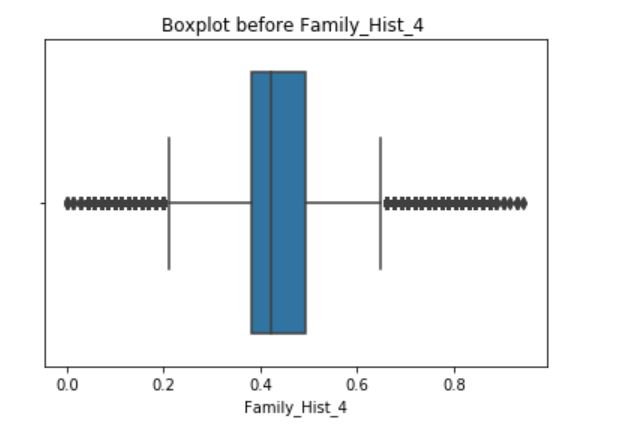
Upper boundary of features {0.7538703469999999, 0.1975, 0.66197183, 0.0021666669999999997, 18.5}

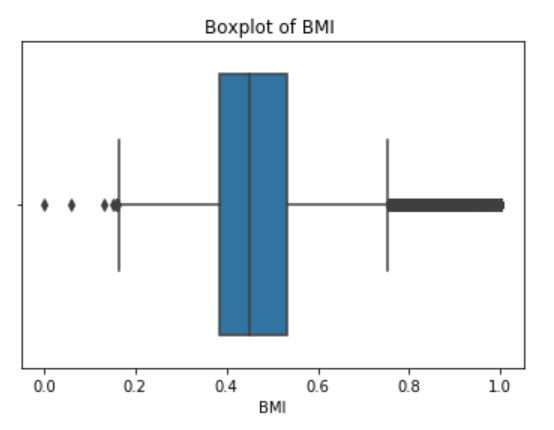
**#Plotting the box plot for features**

'Employment\_Info\_1', 'Medical\_History\_1', 'Insurance\_History\_5', 'Family\_Hist\_4', 'BMI’

****

****

****

****

**#Calculating Skewness of the features:**

**Input:**

print('Skewness of features')

print('Insurance\_History\_5 = ',d['Insurance\_History\_5'].skew())

print('Family\_Hist\_4 = ',d['Family\_Hist\_4'].skew())

print('Employment\_Info\_1 = ',d['Employment\_Info\_1'].skew())

print('BMI = ',d['BMI'].skew())

**Output:**

Skewness of features

Insurance\_History\_5 = 134.89155609994165

Family\_Hist\_4 = 0.4091113228795351

Employment\_Info\_1 = 4.709403418654449

BMI = 0.944260717685289

**#Applying capping and flooring technique to treat outliers:**

**Input:**

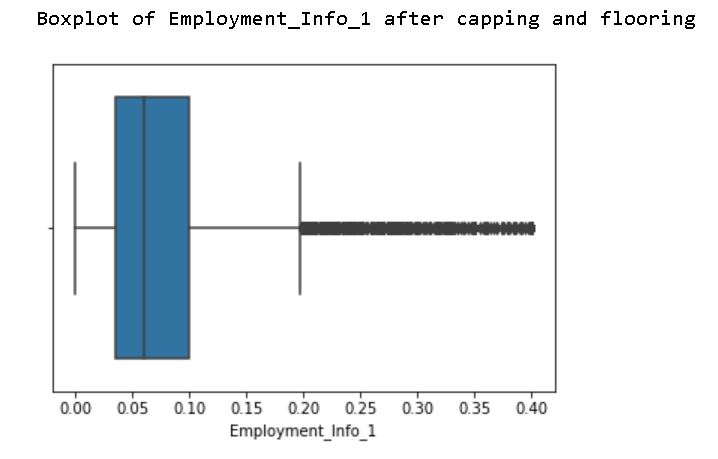
for col in ['Insurance\_History\_5','Family\_Hist\_4','Employment\_Info\_1','BMI']:

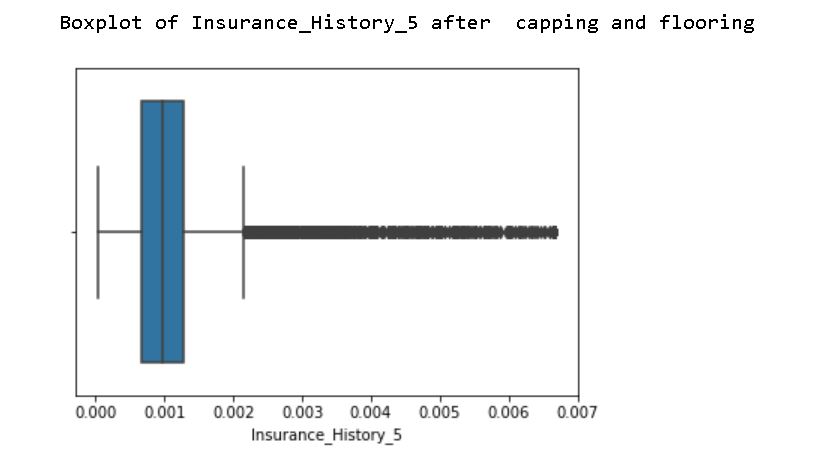
percentiles = d[col].quantile([0.01,0.99]).values

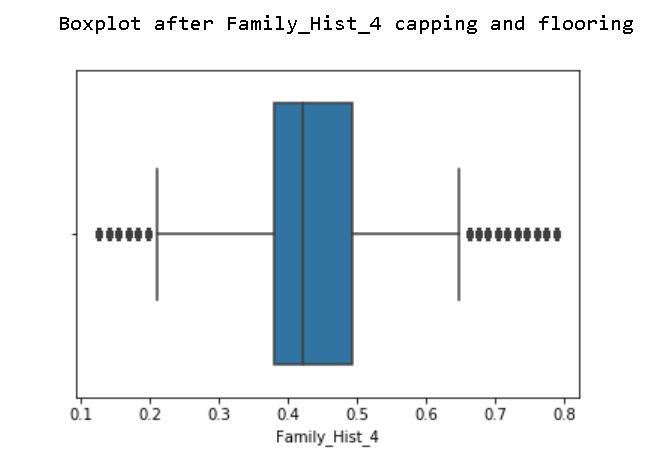
d[col][d[col] <= percentiles[0]] = percentiles[0]

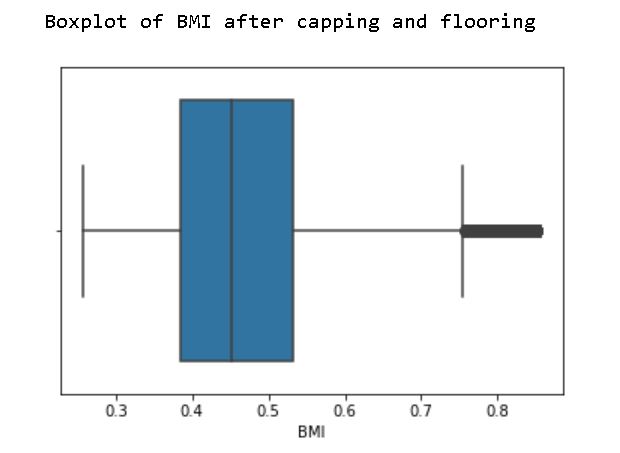
d[col][d[col] >= percentiles[1]] = percentiles[1]

**#Plotting the boxplot for features after capping and flooring technique**

****

****

****

****

**#Calculating skewness for features after capping and flooring**

**Input:**

print('Skewness of features')

print('Insurance\_History\_5 = ',d['Insurance\_History\_5'].skew())

print('Family\_Hist\_4 = ',d['Family\_Hist\_4'].skew())

print('Employment\_Info\_1 = ',d['Employment\_Info\_1'].skew())

print('BMI = ',d['BMI'].skew())

**Output:**

Skewness of features

Insurance\_History\_5 = 2.6319732668048093

Family\_Hist\_4 = 0.4072228128371821

Employment\_Info\_1 = 2.2854012387923675

BMI = 0.8166862152362031

**#Checking the count for each response**

**Input:**

d.groupby(['Response'])['Employment\_Info\_1'].count()

Output:

Response

1 6207

2 6552

3 1013

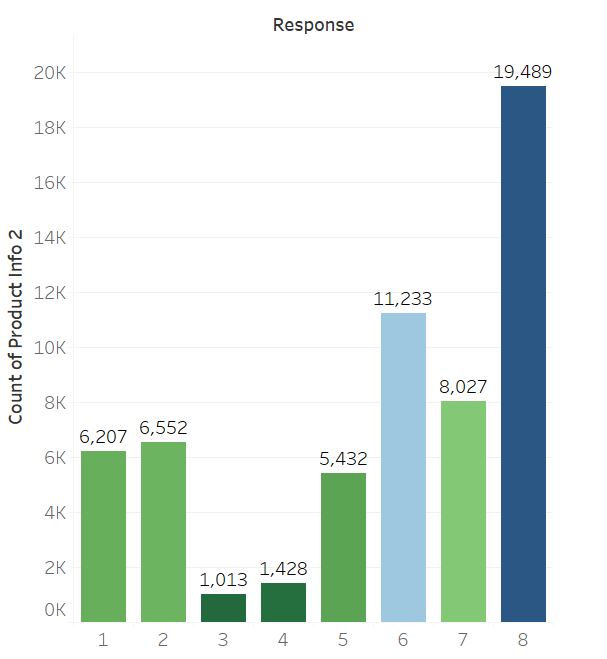
4 1428

5 5432

6 11233

7 8027

8 19489



**As the data is imbalanced SMOTE technique is used to balance the data.**

**Input:**

from imblearn.over\_sampling import SMOTE

smt = SMOTE()

x,y= smt.fit\_sample(x,y)

**#Splitting the data into train and test in the ratio of 70% and 30% respectively**

from sklearn.model\_selection import train\_test\_split

x\_train,x\_test,y\_train,y\_test=train\_test\_split(x,y,test\_size=0.3,random\_state=100)

**#Building base model of Random Forest**

from sklearn.ensemble import RandomForestClassifier

model=RandomForestClassifier(random\_state=100)

model.fit(x\_train,y\_train)

y\_pred=model.predict(x\_test)

from sklearn.metrics import accuracy\_score

print(accuracy\_score(y\_test,y\_pred))

**Output:**

**Accuracy Score: 0.6948304613674263**

**#As we have high multicollinearity we need to eliminate it. So Principal Component Analysis is used.**

from sklearn.decomposition import PCA

pca=PCA()

X\_train\_2 = pca.fit\_transform(x\_train)

X\_test\_2 = pca.transform(x\_test)

**#Building models after applying PCA**

**Random Forest:**

rf=RandomForestClassifier(n\_estimators=20, random\_state=100)

rf.fit(X\_train\_2,y\_train)

y\_pred\_rf = rf.predict(X\_test\_2)

ac\_PCA = accuracy\_score(y\_test, y\_pred\_rf)

print("Accuracy Score:", ac\_PCA)

**Output:**

**Accuracy Score: 64.89**

**#applying GridSearch on base rf model**

rf=RandomForestClassifier(random\_state=100)

param={'n\_estimators':np.arange(1,5),'criterion':['gini','entropy'],'max\_depth':np.arange(1,5)}

GS\_rf=GridSearchCV(rf,param,cv=5,scoring='precision\_weighted')

GS\_rf.fit(X\_train\_2,y\_train)

**Output:**

**Accuracy Score: 74.6**

**Gaussian Naïve Bayes:**

**Input:**

from sklearn.naive\_bayes import GaussianNB

gnb = GaussianNB()

gnb.fit(X\_train\_2, y\_train)

y\_pred\_nb = gnb.predict(X\_test\_2)

ac\_PCA = accuracy\_score(y\_test, y\_pred\_nb)

print("Accuracy Score:", ac\_PCA)

**Output:**

**Accuracy Score: 23.59**

**Applying bagging Naive Bayes:**

nb\_bag=BaggingClassifier(base\_estimator=gnb,random\_state=100)

nb\_bag.fit(X\_train\_2,y\_train)

nb\_bag\_pred=nb\_bag.predict(X\_test\_2)

ac\_nb\_bag=accuracy\_score(nb\_bag\_pred,y\_test)

print('Accuracy Score:', ac\_nb\_bag)

**Output:**

**Accuracy Score: 0.24797964681233164**

**Applying gs on bagging nb model**

kf=KFold(n\_splits=5,shuffle=True,random\_state=100)

param={'n\_estimators':np.arange(1,20)}

GS\_nb\_bag=GridSearchCV(nb\_bag,param,cv=kf,scoring='precision\_weighted')

GS\_nb\_bag.fit(X\_train\_2,y\_train)

**Output:**

**Accuracy Score: 0.24797964681233164**

**Decision Tree Classifier:**

from sklearn.tree import DecisionTreeClassifier

dt=DecisionTreeClassifier(random\_state=100)

dt.fit(X\_train\_2,y\_train)

y\_pred\_dt=dt.predict(X\_test\_2)

ac\_pca=accuracy\_score(y\_pred,y\_pred\_dt)

print('Accuracy Score:', ac\_pca)

**Output:**

Accuracy Score: 0.5049172617266002

**Applying gs on base dt model**

dt=DecisionTreeClassifier()

kf=KFold(n\_splits=5,shuffle=True,random\_state=100)

param={'criterion':['gini','entropy'],'max\_depth':np.arange(10,20,2)}

GS\_dt=GridSearchCV(dt,param,cv=kf,scoring='precision\_weighted', n\_jobs=-1)

GS\_dt.fit(X\_train\_2,y\_train)

**Applying gs on base dt model**

dt\_gs=DecisionTreeClassifier(criterion='entropy',max\_depth=18)

dt\_gs.fit(X\_train\_2,y\_train)

y\_pred\_dt=dt\_gs.predict(X\_test\_2)

ac\_dt=accuracy\_score(y\_pred\_dt,y\_test)

print('Accuracy Score:', ac\_dt)

**KNeighborsClassifier:**

from sklearn.neighbors import KNeighborsClassifier

Knn=KNeighborsClassifier()

knn.fit(X\_train\_2,y\_train)

y\_pred\_knn=knn.predict(X\_test\_2)

ac\_pca=accuracy\_score(y\_pred,y\_pred\_knn)

print('Accuracy Score:', ac\_pca)

**Output:**

**Accuracy Score: 0.5182152477872323**

**Applying gs on base knn:**

from sklearn.model\_selection import GridSearchCV

knn=KNeighborsClassifier()

param={'n\_neighbors':np.arange(1,50),'weights':['uniform','distance']}

GS\_knn=GridSearchCV(knn,param,cv=kf,scoring='recall\_weighted')

GS\_knn.fit(X\_train\_2,y\_train)

**Output:**

**Accuracy Score: 0.5182152477872323**

**Applying Bagged knn:**

knn\_bag=BaggingClassifier(base\_estimator=knn,random\_state=100)

knn\_bag.fit(X\_train\_2,y\_train)

knn\_bag\_pred=knn\_bag.predict(X\_test\_2)

ac\_knn=accuracy\_score(knn\_bag\_pred,y\_test)

print('Accuracy Score:', ac\_knn)

**Output:**

**Accuracy Score: 0.5182152477872323**

Random forest is giving the best accuracy.