```
#Importing all the required libraries

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import seaborn as sns
from random import randrange, uniform
```

```
#Reading the train data

train_dataset = pd.read_csv("/content/train_Df64byy.csv")
train_dataset.head()
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Holc
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	

```
#Describing the train data

train_dataset.describe()
```

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium
count	66720.000000	66720.000000	66720.000000	66720.000000	40123.000000	66720.000000	66720.000000
mean	29600.349460	1733.354332	44.846927	42.734937	2.442664	15.112125	14183.940833
std	15003.069766	1424.021443	17.299650	17.310907	1.025191	6.340442	6585.290312
min	1.000000	1.000000	18.000000	16.000000	1.000000	1.000000	2280.000000
25%	16679.750000	527.000000	28.000000	27.000000	1.000000	12.000000	9252.000000

```
#Returning the data types of train

train_dataset.dtypes
```

```
ID                          int64
City_Code                  object
Region_Code               float64
Accomodation_Type          object
Reco_Insurance_Type        object
Upper_Age                 float64
Lower_Age                 float64
Is_Spouse                  object
Health Indicator           object
Holding_Policy_Duration    object
Holding_Policy_Type       float64
Reco_Policy_Cat           float64
Reco_Policy_Premium       float64
Response                  float64
dtype: object
```

```
#Returning the no of classified and non classified

print("No of observations classified as 1 are : ", len(train_dataset[train_dataset['Response']==1]))

print("No of observations classified as 0 are : ", len(train_dataset[train_dataset['Response']==0]))
```

```
No of observations classified as 1 are :  16080
No of observations classified as 0 are :  50640
```

```
#Checking no of missing values in train data

missing_val = pd.DataFrame(train_dataset.isnull().sum())
missing_val
```

	0
ID	0
City_Code	1
Region_Code	1
Accomodation_Type	1
Reco_Insurance_Type	1
Upper_Age	1
Lower_Age	1
Is_Spouse	1
Health Indicator	11692
Holding_Policy_Duration	20252
Holding_Policy_Type	20252
Reco_Policy_Cat	1
Reco_Policy_Premium	1
Response	1

```
#Returning the value counts for each unique value in Holding_Policy_Duration Column

train_dataset['Holding_Policy_Duration'].value_counts()
```

```
1.0     5889
14+     5647
2.0     5582
3.0     4724
4.0     3612
```

```
5.0     3103
6.0     2495
7.0     2161
8.0     1726
9.0     1451
10.0    1070
11.0     709
13.0     688
12.0     659
14.0     607
Name: Holding_Policy_Duration, dtype: int64
```

```
#Replacing 14+ value in Holding_Policy_Duration with 15

# mapping = {'14+':'15'}
train_dataset['Holding_Policy_Duration'] = train_dataset['Holding_Policy_Duration'].replace("14+", "15")
```

```
#Returning the value counts for each unique value in Health_Indicator Column

train_dataset["Health Indicator"].value_counts()
```

```
X1    17087
X2    13522
X3     8890
X4     7550
X5     2242
X6     1677
X7      258
X8      105
X9       86
Name: Health Indicator, dtype: int64
```

```
#Returning the value counts for each unique value in Holding_Policy_Type Column

train_dataset["Holding_Policy_Type"].value_counts()
```

```
3.0    17432
1.0    10647
2.0     6556
```

```
4.0     5488
Name: Holding_Policy_Type, dtype: int64
```

```
#Checking the percentage of missing values for train data

missing_val = missing_val.reset_index()
missing_val = missing_val.rename(columns = {'index':'Variables', 0: 'Missing_percentage'})
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(train_dataset))*100
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
```

	Variables	Variables	Missing_percentage
0	1	City_Code	0.000003
1	2	Region_Code	0.000003
2	3	Accomodation_Type	0.000003
3	4	Reco_Insurance_Type	0.000003
4	5	Upper_Age	0.000003
5	6	Lower_Age	0.000003
6	7	Is_Spouse	0.000003
7	11	Reco_Policy_Cat	0.000003
8	12	Reco_Policy_Premium	0.000003
9	13	Response	0.000003
10	0	ID	0.000000
11	8	Health Indicator	0.000000
12	9	Holding_Policy_Duration	0.000000
13	10	Holding_Policy_Type	0.000000

```
#Imputing the missing values for required columns
```

```
from sklearn.impute import SimpleImputer

impute_size = SimpleImputer(strategy = "most_frequent")
train_dataset['Health Indicator'] = impute_size.fit_transform(train_dataset[['Health Indicator']])
train_dataset['Holding_Policy_Duration'] = impute_size.fit_transform(train_dataset[['Holding_Policy_Duration']])
train_dataset['Holding_Policy_Type'] = train_dataset['Holding_Policy_Type'].fillna(train_dataset['Holding_Policy_Type']
```

```
#Checking the percentage of missing values for train data after imputing the missing values

missing_val = pd.DataFrame(train_dataset.isnull().sum())
missing_val = missing_val.reset_index()
missing_val = missing_val.rename(columns = {'index':'Variables', 0: 'Missing_percentage'})
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(train_dataset))*100
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
```

	Variables	Missing_percentage

```
#Still we can some missing values percentage, so we have to drop columns which have nan value

train_dataset.dropna(how='any')
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Healt Indicat
0	1	C3	3213.0	Rented	Individual	36.0	36.0	No	›
1	2	C5	1117.0	Owned	Joint	75.0	22.0	No	›
2	3	C5	3732.0	Owned	Individual	32.0	32.0	No	›
3	4	C24	4378.0	Owned	Joint	52.0	48.0	No	›
4	5	C8	2190.0	Rented	Individual	44.0	44.0	No	›
...	...	...	...	...	...	...	...	...	
50877	35040	C1	102.0	Rented	Joint	48.0	47.0	Yes	›
50878	35041	C1	332.0	Owned	Individual	70.0	70.0	No	›
50879	35042	C6	1165.0	Owned	Joint	57.0	56.0	Yes	›
50880	35043	C11	1032.0	Owned	Joint	72.0	69.0	Yes	›
50881	35044	C1	729.0	Rented	Individual	25.0	25.0	No	›

50882 rows × 14 columns

```
#Checking the percentage of missing values for train data after dropping the missing values

missing_val = pd.DataFrame(train_dataset.isnull().sum())
missing_val = missing_val.reset_index()
missing_val = missing_val.rename(columns = {'index':'Variables', 0: 'Missing_percentage'})
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(train_dataset))*100
missing_val1 = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val1
```

	Variables	Missing_percentage
0	ID	0.0
1	City_Code	0.0
2	Region_Code	0.0
3	Accomodation_Type	0.0
4	Reco_Insurance_Type	0.0
5	Upper_Age	0.0
6	Lower_Age	0.0
7	Is_Spouse	0.0
8	Health Indicator	0.0
9	Holding_Policy_Duration	0.0
10	Holding_Policy_Type	0.0
11	Reco_Policy_Cat	0.0
12	Reco_Policy_Premium	0.0
13	Response	0.0

I'm trying to consider only 'Reco_Policy_Premium' column for outliers because it's the only column which deals with money and all other columns belongs to ID, Age Duration Etc

```
#Process to detect outliers with the help of quartile

max_thresold = train_dataset['Reco_Policy_Premium'].quantile(0.95)
max_thresold
```

```
26852.0
```

```
#Checking the data which are outliers with max_thresold
```

```

train_dataset[train_dataset['Reco_Policy_Premium'] > max_thresold]
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Healt Indicato
1	2	C5	1117.0	Owned	Joint	75.0	22.0	No	›
7	8	C1	3175.0	Owned	Joint	75.0	73.0	Yes	›
8	9	C15	3497.0	Owned	Joint	52.0	43.0	No	›
48	49	C2	2858.0	Owned	Joint	57.0	55.0	Yes	›
49	50	C1	85.0	Owned	Joint	73.0	68.0	Yes	›
...	...	...	...	...	...	...	...	...	
50793	34956	C1	3014.0	Owned	Joint	47.0	46.0	Yes	›
50798	34961	C2	484.0	Owned	Joint	69.0	68.0	Yes	›
50812	34975	C1	907.0	Owned	Joint	75.0	72.0	Yes	›
50842	35005	C1	4370.0	Owned	Joint	71.0	68.0	No	›
50852	35015	C1	1383.0	Owned	Joint	59.0	58.0	Yes	›

2544 rows × 14 columns

```
#Process to detect outliers with the help of quartile

min_thresold = train_dataset['Reco_Policy_Premium'].quantile(0.05)
min_thresold
```

```
5224.0
```

```
#Checking the data which are outliers with min_thresold

train_dataset[train_dataset['Reco_Policy_Premium'] < min_thresold]
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Healt Indicato
10	11	C28	600.0	Owned	Individual	21.0	21.0	No	X
15	16	C3	1484.0	Rented	Individual	20.0	20.0	No	X
22	23	C25	787.0	Rented	Individual	18.0	18.0	No	X
27	28	C9	855.0	Rented	Individual	21.0	21.0	No	X
46	47	C3	1475.0	Rented	Individual	21.0	21.0	No	X
...	...	...	...	...	...	...	...	...	
50775	34938	C3	1246.0	Owned	Individual	19.0	19.0	No	X
50814	34977	C6	155.0	Rented	Individual	28.0	28.0	No	X
50822	34985	C1	1984.0	Rented	Individual	19.0	19.0	No	X
50836	34999	C3	1418.0	Rented	Individual	20.0	20.0	No	X
50874	35037	C27	3277.0	Owned	Individual	29.0	29.0	No	X

2537 rows × 14 columns

```
train_dataset = train_dataset[(train_dataset['Reco_Policy_Premium'] < max_thresold) & (train_dataset['Reco_Policy_Premi
```

```
train_dataset
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Healt Indicato
0	1	C3	3213.0	Rented	Individual	36.0	36.0	No	❯
2	3	C5	3732.0	Owned	Individual	32.0	32.0	No	❯
3	4	C24	4378.0	Owned	Joint	52.0	48.0	No	❯
4	5	C8	2190.0	Rented	Individual	44.0	44.0	No	❯
5	6	C9	1785.0	Rented	Individual	52.0	52.0	No	❯
...	...	...	...	...	...	...	...	...	

```
#Returning the no of classified and non classified after imputing missing values and removing outliers from train data

print("No of observations classified as 1 are : ", len(train_dataset[train_dataset['Response'] == 1]))

print("No of observations classified as 0 are : ", len(train_dataset[train_dataset['Response'] == 0]))
```

```
No of observations classified as 1 are :  16080
No of observations classified as 0 are :  50640
```

45770 rows x 14 columns

```
#Describing the test_data
test_dataset =pd.read_csv('/content/test_YCcRUnU.csv')
test_dataset.describe()
```

ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium

```
#Checking the data types for test data
test_dataset = pd.read_csv("/content/test_YCcRUnU.csv")

test_dataset.dtypes
```

```
ID                          int64
City_Code                  object
Region_Code                 int64
Accomodation_Type          object
Reco_Insurance_Type        object
Upper_Age                   int64
Lower_Age                   int64
Is_Spouse                  object
Health Indicator           object
Holding_Policy_Duration    object
Holding_Policy_Type       float64
Reco_Policy_Cat             int64
Reco_Policy_Premium       float64
dtype: object
```

```
#Checking the percentage of missing values for test data

missing_val = pd.DataFrame(test_dataset.isnull().sum())
missing_val = missing_val.reset_index()
missing_val = missing_val.rename(columns = {'index':'Variables', 0: 'Missing_percentage'})
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(test_dataset))*100
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
```

	Variables	Missing_percentage
0	Holding_Policy_Duration	39.454254
1	Holding_Policy_Type	39.454254
2	Health Indicator	23.054345
3	ID	0.000000
4	City_Code	0.000000
5	Region_Code	0.000000
6	Accomodation_Type	0.000000
7	Reco_Insurance_Type	0.000000

```
#Returning the value counts for each unique value in Holding_Policy_Duration Column

test_dataset['Holding_Policy_Duration'].value_counts()
```

```
14+     1892
1.0     1891
2.0     1772
3.0     1606
4.0     1205
5.0      992
6.0      903
7.0      664
8.0      569
9.0      493
10.0     333
11.0     254
13.0     221
14.0     211
12.0     196
Name: Holding_Policy_Duration, dtype: int64
```

```
#Replacing 14+ value with 15 for 'Holding_Policy_Duration' column

test_dataset['Holding_Policy_Duration'] = test_dataset['Holding_Policy_Duration'].replace("14+", "15")
```

```
#Returning the value counts for each unique value in Health Indicator Column

test_dataset['Health Indicator'].value_counts()
```

```
X1    5614
X2    4516
X3    2846
X4    2442
X5     681
X6     514
X7      96
X8      41
X9      28
Name: Health Indicator, dtype: int64
```

```
#Returning the value counts for each unique value in Holding_Policy_Type Column

test_dataset['Holding_Policy_Type'].value_counts()
```

```
3.0    5572
1.0    3574
2.0    2150
4.0    1906
Name: Holding_Policy_Type, dtype: int64
```

```
#Imputing missing values for test data

impute_size = SimpleImputer(strategy = "most_frequent")
test_dataset['Health Indicator'] = impute_size.fit_transform(test_dataset[['Health Indicator']])
test_dataset['Holding_Policy_Duration'] = impute_size.fit_transform(test_dataset[['Holding_Policy_Duration']])
test_dataset['Holding_Policy_Type'] = test_dataset['Holding_Policy_Type'].fillna(test_dataset['Holding_Policy_Type'].me
```

```
#Checking the percentage of missing values for test data after imputing the missing values

missing_val = pd.DataFrame(test_dataset.isnull().sum())
missing_val = missing_val.reset_index()
missing_val = missing_val.rename(columns = {'index':'Variables', 0: 'Missing_percentage'})
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(test_dataset))*100
```

```
missing_val['Missing_percentage'] = (missing_val['Missing_percentage']/len(test_dataset))*100
missing_val = missing_val.sort_values('Missing_percentage', ascending = False).reset_index(drop = True)
missing_val
```

	Variables	Missing_percentage
0	ID	0.0
1	City_Code	0.0
2	Region_Code	0.0
3	Accomodation_Type	0.0
4	Reco_Insurance_Type	0.0
5	Upper_Age	0.0
6	Lower_Age	0.0
7	Is_Spouse	0.0
8	Health Indicator	0.0
9	Holding_Policy_Duration	0.0
10	Holding_Policy_Type	0.0
11	Reco_Policy_Cat	0.0
12	Reco_Policy_Premium	0.0

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

le = LabelEncoder()
objList = train_dataset.select_dtypes(include="object").columns

for feat in objList:
    train_dataset[feat] = le.fit_transform(train_dataset[feat].astype(str))
```

```
train_dataset
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Healt Indicato
0	1	22	3213	1	0	36	36	0	
1	2	31	1117	0	1	75	22	0	
2	3	31	3732	0	0	32	32	0	
3	4	16	4378	0	1	52	48	0	
4	5	34	2190	1	0	44	44	0	
...	...	...	...	...	...	...	...	...	
66715	50878	30	845	1	0	22	22	0	
66716	50879	31	4188	1	0	27	27	0	
66717	50880	0	442	1	0	63	63	0	
66718	50881	0	4	0	1	71	49	0	
66719	50882	22	3866	1	0	24	24	0	

66720 rows × 14 columns

```
from sklearn.preprocessing import LabelEncoder, OneHotEncoder

le = LabelEncoder()
objList = test_dataset.select_dtypes(include="object").columns

for feat in objList:
    test_dataset[feat] = le.fit_transform(test_dataset[feat].astype(str))
```

```
test_dataset.head()
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	H
0	50883	0	156	0	0	30	30	0	0	
1	50884	30	7	0	1	69	68	1	0	
4	50887	0	951	0	0	75	75	0	2	

```python
X = train_dataset.drop("Response", axis=1)   #Feature Matrix
y = train_dataset["Response"]
```

```python
X.head()
```

	ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Holc
0	1	22	3213	1	0	36	36	0	0	
2	3	31	3732	0	0	32	32	0	0	
3	4	16	4378	0	1	52	48	0	0	
4	5	34	2190	1	0	44	44	0	1	
5	6	35	1785	1	0	52	52	0	1	

```python
# separate dataset into train and test
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(
    X,
    y,
    test_size=0.2,
    random_state=39)

X_train.shape, X_val.shape
```

```
((53376, 13), (13344, 13))
```

Handling Imablance

```
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from matplotlib import pyplot

# plot no skill and model roc curves
def plot_roc_curve(test_y, naive_probs, model_probs):
  # plot naive skill roc curve
  fpr, tpr, _ = roc_curve(test_y, naive_probs)
  pyplot.plot(fpr, tpr, linestyle='--', label='No Skill')
  # plot model roc curve
  fpr, tpr, _ = roc_curve(test_y, model_probs)
  pyplot.plot(fpr, tpr, marker='.', label='Logistic')
  # axis labels
  pyplot.xlabel('False Positive Rate')
  pyplot.ylabel('True Positive Rate')
  # show the legend
  pyplot.legend()
  # show the plot
  pyplot.show()
from sklearn.metrics import precision_recall_curve
```

```
# no skill model, stratified random class predictions
model = DummyClassifier(strategy='stratified')
model.fit(X_train,y_train)
yhat = model.predict_proba(X_val)
pos_probs = yhat[:, 1]
# calculate the precision-recall auc
precision, recall, _ = precision_recall_curve(y_val, pos_probs)
auc_score = auc(recall, precision)
print('No Skill PR AUC: %.3f' % auc_score)
```

```
No Skill PR AUC: 0.333
```

```
# skilled model
model = LogisticRegression(solver='lbfgs')
model.fit(X_train, y_train)
yhat = model.predict_proba(X_val)
model_probs = yhat[:, 1]
# calculate roc auc
roc_auc = roc_auc_score(y_val, model_probs)
print('Logistic ROC AUC %.3f' % roc_auc)

```

```
Logistic ROC AUC 0.501
```

```
# example of evaluating a decision tree with random oversampling
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from imblearn.pipeline import Pipeline
from imblearn.over_sampling import RandomOverSampler
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/externals/six.py:31: FutureWarning: The module is deprecated in version
  "(https://pypi.org/project/six/).", FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning: The sklearn.neighbors.base modu
  warnings.warn(message, FutureWarning)
```

```
# define pipeline
steps = [('over', RandomOverSampler()), ('model', DecisionTreeClassifier())]
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X, y, scoring='f1_micro', cv=cv, n_jobs=-1)
score = mean(scores)
```

```
print('F-measure: %.3f' % score)
```

```
F-measure: 0.668
```

```
from numpy import where
from collections import Counter
from imblearn.over_sampling import SMOTE
# summarize class distribution
counter = Counter(y)
print(counter)
# scatter plot of examples by class label
# transform the dataset
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)
```

```
Counter({0: 34779, 1: 10991})
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprec
  warnings.warn(msg, category=FutureWarning)
```

```
# summarize the new class distribution
counter = Counter(y)
print(counter)
```

```
Counter({0: 34779, 1: 34779})
```

```
# define model
model = DecisionTreeClassifier()
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.758
```

```
# define pipeline
steps = [('over', SMOTE()), ('model', DecisionTreeClassifier())]
```

```
pipeline = Pipeline(steps=steps)
# evaluate pipeline
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
scores = cross_val_score(pipeline, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.758
```

```
from imblearn.over_sampling import BorderlineSMOTE
# summarize class distribution
counter = Counter(y)
print(counter)
# transform the dataset
oversample = BorderlineSMOTE()
X, y = oversample.fit_resample(X_train, y_train)
# summarize the new class distribution
counter = Counter(y)
# summarize the new class distribution
counter = Counter(y)
print(counter)
```

```
Counter({0: 9900, 1: 100})
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprec
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprec
  warnings.warn(msg, category=FutureWarning)
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: Function safe_indexing is deprec
  warnings.warn(msg, category=FutureWarning)
Counter({1: 27835, 0: 27835})
```

```
# combined SMOTE and Tomek Links sampling for imbalanced classification
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from imblearn.pipeline import Pipeline
from sklearn.tree import DecisionTreeClassifier
from imblearn.combine import SMOTETomek
```

```
from imblearn.under_sampling import TomekLinks

# define model
model = DecisionTreeClassifier()
# define sampling
resample = SMOTETomek(tomek=TomekLinks(sampling_strategy='majority'))
# define pipeline
pipeline = Pipeline(steps=[('r', resample), ('m', model)])
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(pipeline, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.529
```

```
# fit a logistic regression model on an imbalanced classification dataset
from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.linear_model import LogisticRegression

# define model
weights = {0:0.01, 1:1.0}
model = LogisticRegression(solver='lbfgs', class_weight=weights)
model = LogisticRegression(solver='lbfgs')
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.507
```

```
# calculate heuristic class weighting
```

```
from sklearn.utils.class_weight import compute_class_weight
# calculate class weighting
weighting = compute_class_weight('balanced', [0,1], y_train)
print(weighting)
```

```
[0.65773307 2.08495616]
```

```
# define model
model = LogisticRegression(solver='lbfgs', class_weight='balanced')
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=25, n_repeats=10, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.552
```

```
# define model
model = LogisticRegression(solver='lbfgs')
# define grid
balance = [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}]
param_grid = dict(class_weight=balance)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=cv,
scoring='roc_auc')
# execute the grid search
grid_result = grid.fit(X_train, y_train)
# report the best configuration
print('Best: %f using %s' % (grid_result.best_score_, grid_result.best_params_))
# report all configurations
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
    print('%f (%f) with: %r' % (mean, stdev, param))
```

```
    print('%f (%f) with: %r' % (mean, stdev, param))
```

```
Best: 0.561343 using {'class_weight': {0: 1, 1: 10}}
0.478032 (0.009518) with: {'class_weight': {0: 100, 1: 1}}
0.479372 (0.011027) with: {'class_weight': {0: 10, 1: 1}}
0.506829 (0.009842) with: {'class_weight': {0: 1, 1: 1}}
0.561343 (0.011775) with: {'class_weight': {0: 1, 1: 10}}
0.555669 (0.011346) with: {'class_weight': {0: 1, 1: 100}}
```

```
# fit a decision tree on an imbalanced classification dataset
from numpy import mean
# define model
model = DecisionTreeClassifier()
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.544
```

```
# define model
model = DecisionTreeClassifier(class_weight='balanced')
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
Mean ROC AUC: 0.542
```

```
#define model
model = DecisionTreeClassifier()
# define grid
balance = [{0:100,1:1}, {0:10,1:1}, {0:1,1:1}, {0:1,1:10}, {0:1,1:100}]
param_grid = dict(class_weight=balance)
# define evaluation procedure
```

```
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=cv,
scoring='roc_auc')
# execute the grid search
grid_result = grid.fit(X_train, y_train)
# report the best configuration
print('Best: %f using %s' % (grid_result.best_score_, grid_result.best_params_))
# report all configurations
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print('%f (%f) with: %r' % (mean, stdev, param))
```

```
Best: 0.544592 using {'class_weight': {0: 10, 1: 1}}
0.542249 (0.008030) with: {'class_weight': {0: 100, 1: 1}}
0.544592 (0.011834) with: {'class_weight': {0: 10, 1: 1}}
0.543369 (0.009397) with: {'class_weight': {0: 1, 1: 1}}
0.536144 (0.008191) with: {'class_weight': {0: 1, 1: 10}}
0.536047 (0.006741) with: {'class_weight': {0: 1, 1: 100}}
```

```
# define model
model = SVC(gamma='scale', class_weight='balanced')
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X, y, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
from sklearn.metrics import roc_auc_score
from keras.layers import Dense
from keras.models import Sequential
# define the neural network model
def define_model(n_input):
# define model
  model = Sequential()
```

```
# define first hidden layer and visible layer
  model.add(Dense(10, input_dim=n_input, activation='relu',
  kernel_initializer='he_uniform'))
# define output layer
  model.add(Dense(1, activation='sigmoid'))
# define loss and optimizer
  model.compile(loss='binary_crossentropy', optimizer='sgd')
  return model
```

```
# define the model
n_input = X_train.shape[1]
model = define_model(n_input)
```

```
# fit model
model.fit(X_train,y_train,epochs=100, verbose=1)
# make predictions on the test dataset
yhat = model.predict(y_val)
# evaluate the ROC AUC of the predictions
score = roc_auc_score(testy, yhat)
print('ROC AUC: %.3f' % score)
```

```
model = define_model(n_input)
# fit model
weights = {0:1, 1:100}
history = model.fit(X_train,y_train, class_weight=weights, epochs=100, verbose=1)
# evaluate model
yhat = model.predict(X_val)
score = roc_auc_score(y_val, yhat)
print('ROC AUC: %.3f' % score)
```

```
Epoch 1/100
1145/1145 [==============================] - 2s 2ms/step - loss: 12005825.8617
Epoch 2/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3963
Epoch 3/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3979
Epoch 4/100
```

```
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3947
Epoch 5/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3975
Epoch 6/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3968
Epoch 7/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3958
Epoch 8/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3965
Epoch 9/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3964
Epoch 10/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3955
Epoch 11/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3959
Epoch 12/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3961
Epoch 13/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3973
Epoch 14/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3957
Epoch 15/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3970
Epoch 16/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3957
Epoch 17/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3955
Epoch 18/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3959
Epoch 19/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3939
Epoch 20/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3957
Epoch 21/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3974
Epoch 22/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3978
Epoch 23/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3959
Epoch 24/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3956
Epoch 25/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3973
```

```
Epoch 26/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3957
Epoch 27/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3961
Epoch 28/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3959
Epoch 29/100
1145/1145 [==============================] - 2s 2ms/step - loss: 3.3967
```

```
# fit xgboost on an imbalanced classification dataset
from numpy import mean
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from xgboost import XGBClassifier
```

```
# define model
model = XGBClassifier()
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.5f' % mean(scores))
```

```
Mean ROC AUC: 0.62498
```

```
# define model
model = XGBClassifier(scale_pos_weight=99)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.5f' % mean(scores))
```

```
Mean ROC AUC: 0.62435
```

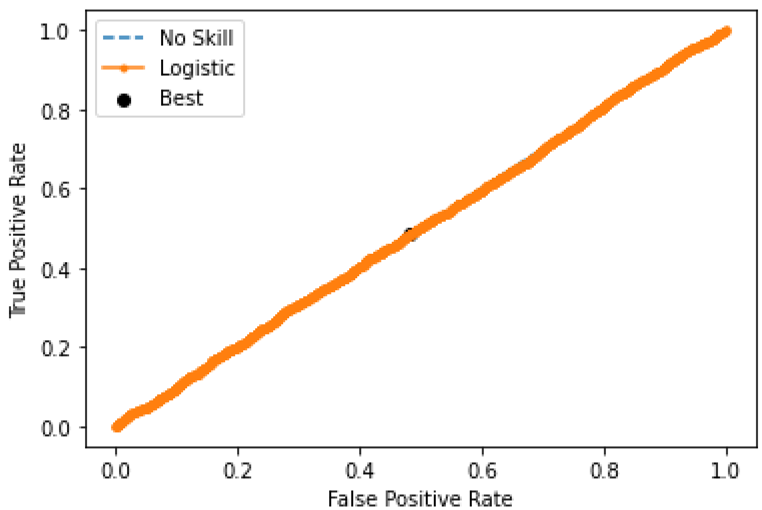
```
# define grid
```

```
# define grid
weights = [1, 10, 25, 50, 75, 99, 100, 1000]
param_grid = dict(scale_pos_weight=weights)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# define grid search
grid = GridSearchCV(estimator=model, param_grid=param_grid, n_jobs=-1, cv=cv,
scoring='roc_auc')
# execute the grid search
grid_result = grid.fit(X_train, y_train)
# report the best configuration
print("Best: %f using %s" % (grid_result.best_score_, grid_result.best_params_))
# report all configurations
means = grid_result.cv_results_['mean_test_score']
stds = grid_result.cv_results_['std_test_score']
params = grid_result.cv_results_['params']
for mean, stdev, param in zip(means, stds, params):
  print("%f (%f) with: %r" % (mean, stdev, param))
```

```
# class imbalnce with Gaussian

from numpy import sqrt
from numpy import argmax
from sklearn.datasets import make_classification
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import roc_curve
from matplotlib import pyplot


# fit a model
model = LogisticRegression(solver='lbfgs')
model.fit(X_train, y_train)
# predict probabilities
yhat = model.predict_proba(X_val)
# keep probabilities for the positive outcome only
yhat = yhat[:, 1]
# calculate roc curves
fpr, tpr, thresholds = roc_curve(y_val, yhat)
```

```
# calculate the g-mean for each threshold
gmeans = sqrt(tpr * (1-fpr))
# locate the index of the largest g-mean
ix = argmax(gmeans)
print('Best Threshold=%f, G-mean=%.3f' % (thresholds[ix], gmeans[ix]))
# plot the roc curve for the model
pyplot.plot([0,1], [0,1], linestyle='--', label='No Skill')
pyplot.plot(fpr, tpr, marker='.', label='Logistic')
pyplot.scatter(fpr[ix], tpr[ix], marker='o', color='black', label='Best')
# axis labels
pyplot.xlabel('False Positive Rate')
pyplot.ylabel('True Positive Rate')
pyplot.legend()
# show the plot
pyplot.show()
```

```
Best Threshold=0.252692, G-mean=0.501
```




```
# Class imbalance using SVM

from numpy import mean
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.calibration import CalibratedClassifierCV
```

```
from sklearn.calibration import CalibratedClassifierCV
from sklearn.svm import SVC
```

```
# define model
model = SVC(gamma='scale')
# wrap the model
calibrated = CalibratedClassifierCV(model, method='isotonic', cv=3)
# define evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate model
scores = cross_val_score(calibrated, X_train, y_train, scoring='roc_auc', cv=cv, n_jobs=-1)
# summarize performance
print('Mean ROC AUC: %.3f' % mean(scores))
```

```
/usr/local/lib/python3.7/dist-packages/joblib/externals/loky/process_executor.py:691: UserWarning: A worker stopped whi
  "timeout or by a memory leak.", UserWarning
```

```
train_dataset
```

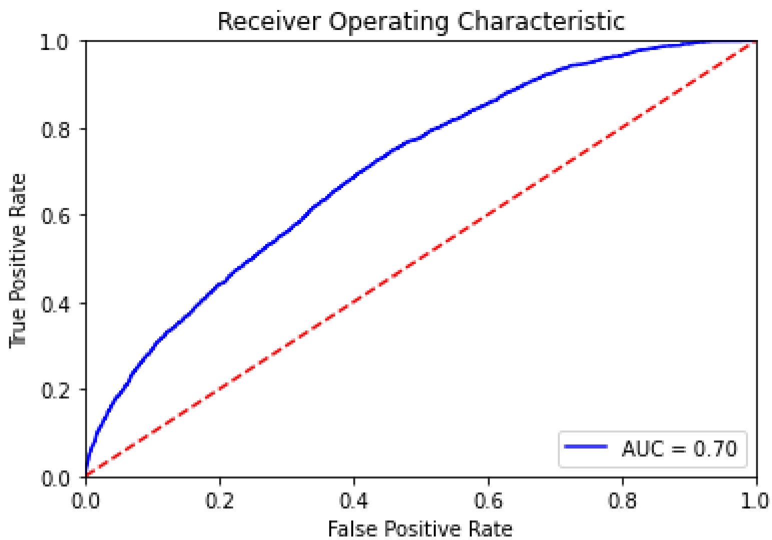
	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Hol
0	22	3213.0	1	0	36.0	36.0	0	0	

```
y = train_dataset.values
#train_dataset.drop(['ID', 'Response'], inplace=True, axis=1)

x = train_dataset.values

```

```
from lightgbm import LGBMClassifier

lgbm = LGBMClassifier(random_state=5)

lgbm.fit(X_train, y_train)

y_pred = lgbm.predict(X_val)
```

50881	0	729.0	1	0	25.0	25.0	0	0

```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = lgbm.predict_proba(X_val)
preds = probs[:,1]
fpr, tpr, threshold = metrics.roc_curve(y_val, preds)
roc_auc = metrics.auc(fpr, tpr)

# method I: plt
import matplotlib.pyplot as plt
plt.title('Receiver Operating Characteristic')
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```




```
import sklearn.metrics as metrics
# calculate the fpr and tpr for all thresholds of the classification
probs = lgbm.predict_proba(test_dataset)
preds = probs[:,1]

```

```
ID = test_dataset['ID'].values

```

```
submission = pd.DataFrame({'ImageId':ID,
                            'Response':preds,
                           }).set_index('ImageId')
```

```
submission.head()
```

	Response
ImageId	
50883	0.221873
50884	0.368947

```
submission.to_csv('mnist_lgbm.csv', columns=['Response'])
```

These all observation.

1. Random Forest class_balanced will never impacted your model accuracy
2. class imbalnce with Gaussian never gives us right accuracy
3. Weights Paremeters doen't work imbalance
4. Probabalistics methods also didn't solved the problem imbalance
5. Poor Pefroramace on STOME
6. Poor Perforomance on SMOTE +SMOTETomek
7. No performance impormace over SMOTE over sampling
8. U never think of Performance if your using Undersampling
7. Lots f Books and Online class room talks about imbalance data. but i may work on small sample.
8. XG Boost works some better
9. LightGBM Worked very well.its master piece for class imbalance.
10.Even trained Model on Deep learning No improvement perforamance with help of

Pycaret having paramete class_imbalance is True, But No effect on IT.even pycaret works well because it handles XGBOOST, LIGHTGBM perfe

AutoViml having class imbalance = SMOTE, I will never effect model ROC AUC score.

These all are my exprience with analystics vidya competition.

```
ned Model on Deep learning No improvement perforamance

Bottom line : Don't follow Bilindly on online platform posts. understand the core concept


Please find the experinments for the same.



## This is formatted as c **bold text**ode
```

https://neptune.ai/blog/lightgbm-parameters-guide

```
1
```