Travel insurance Project

https://drive.google.com/drive/folders/1MNre59Ma59HLxKIhUgSa7adhNDjJ9T1V?usp=sharing (https://drive.google.com/drive/folders/1MNre59Ma59HLxKIhUgSa7adhNDjJ9T1V?usp=sharing)

In this project we create a model that can predict for whether a customer can claim for Travel Insurance or not.

case study :

Insurance companies take risks over customers. Risk management is a very important aspect of the insurance industry. Insurers consider every quantifiable factor to develop profiles of high and low insurance risks. Insurers collect vast amounts of information about policyholders and analyse the data. As a Data scientist in an insurance company, you need to analyse the available data and predict whether to approve the insurance or not.

About Dataset

Feature Description

ID Unique identifier

Agency Agency name

Agency Type Type of travel insurance agency

Distribution Channel Online/Offline distribution channel

Product Name Travel insurance product name

Duration Duration of travel

Destination Destination of travel

Net sales Net sales of travel insurance policies

Commision The commission received by travel insurance agency

Gender Traveller's gender

Age Traveller's Age

In [1]:

```
# Basic Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Cross-Validation
from sklearn.model_selection import train_test_split

# LabelEncoding
from sklearn.preprocessing import LabelEncoder

# Evaluation
from sklearn.metrics import classification_report

# Scaling
from sklearn.preprocessing import MinMaxScaler

# Ridge, Lasso
from sklearn.linear_model import Ridge, Lasso

# Logistic Regression
from sklearn.linear_model import LogisticRegression

# Decision Tree
from sklearn.tree import DecisionTreeClassifier

# GridSearchCV
from sklearn.model_selection import GridSearchCV

# Boosting, RandomForest
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomForestCl
from xgboost import XGBClassifier

# Ensemble
from sklearn.ensemble import VotingClassifier, BaggingClassifier

# Feature Selection
from sklearn.feature_selection import chi2, SelectKBest

# SVM
from sklearn.svm import LinearSVC, SVC

# Skewness
from scipy.stats import skew

# Over and Under Sampling
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler
from collections import Counter

# Pickle
import pickle

# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
# Reading the data and viewing a small part of it to get some understanding of the data.

df = pd.read_csv("E:\MLcsv\data.csv")
print(df.shape)
df.head(8)
```

```
(50553, 12)
```

Out[2]:

	ID	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales
0	3433	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	7	MALAYSIA	0.0
1	4339	EPX	Travel Agency	Online	Cancellation Plan	0	85	SINGAPORE	69.0
2	34590	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	11	MALAYSIA	19.8
3	55816	EPX	Travel Agency	Online	2 way Comprehensive Plan	0	16	INDONESIA	20.0
4	13816	EPX	Travel Agency	Online	Cancellation Plan	0	10	KOREA, REPUBLIC OF	15.0
5	50349	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	64	THAILAND	49.5
6	9921	JZI	Airlines	Online	Value Plan	0	23	JAPAN	-69.0
7	21923	JZI	Airlines	Online	Basic Plan	0	31	HONG KONG	26.0

In [3]:

```
# We will get a list of the number of unique values for each column

df.nunique()
```

Out[3]:

```
ID                      50553
Agency                     16
Agency Type                 2
Distribution Channel        2
Product Name               25
Claim                       2
Duration                  444
Destination               102
Net Sales                1053
Commision (in value)      964
Gender                      2
Age                        88
dtype: int64
```

In [4]:

```
# We will check for null values and the Dtype of each feature.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50553 entries, 0 to 50552
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   ID                    50553 non-null  int64  
 1   Agency                50553 non-null  object 
 2   Agency Type           50553 non-null  object 
 3   Distribution Channel  50553 non-null  object 
 4   Product Name          50553 non-null  object 
 5   Claim                 50553 non-null  int64  
 6   Duration              50553 non-null  int64  
 7   Destination           50553 non-null  object 
 8   Net Sales             50553 non-null  float64
 9   Commision (in value)  50553 non-null  float64
 10  Gender                14600 non-null  object 
 11  Age                   50553 non-null  int64  
dtypes: float64(2), int64(4), object(6)
memory usage: 4.6+ MB
```

In [5]:

```
((df.isnull().sum())*100)/len(df)
```

Out[5]:

```
ID                       0.000000
Agency                   0.000000
Agency Type              0.000000
Distribution Channel     0.000000
Product Name             0.000000
Claim                    0.000000
Duration                 0.000000
Destination              0.000000
Net Sales                0.000000
Commision (in value)     0.000000
Gender                  71.119419
Age                      0.000000
dtype: float64
```

71% of the Gender column have null values.
We will drop the column as there does not seem to be any other feature that could help us with filling in the missing data.

In [6]:

```
df.drop("Gender", axis=1, inplace=True)
```

In [7]:

```
# Having a look at all the unique values of each feature.

for cols in df:
    print("\n{:20} - {}" .format(cols.title(), df[cols].unique()))
```

```
 'VENEZUELA' 'IRAN, ISLAMIC REPUBLIC OF' 'CAYMAN ISLANDS']

Net Sales            - [  0.     69.     19.8   ... 145.     42.4    12.58]

Commision (In Value) - [1.7820e+01 0.0000e+00 1.1880e+01 2.9700e+01 2.4150
e+01 9.1000e+00
 4.5000e+00 2.3760e+01 1.9600e+00 9.5700e+00 6.3000e+00 1.5500e+01
 4.0000e+00 7.3800e+00 7.7000e+00 4.3100e+00 1.4440e+01 3.1000e-01
 4.7520e+01 6.0000e+00 4.1580e+01 1.2950e+01 8.1300e+00 2.7360e+01
 5.4000e+01 3.7400e+00 8.3800e+00 2.5600e+00 9.7500e+00 4.6300e+00
 1.3650e+01 3.1200e+01 1.5000e+01 9.7340e+01 1.2400e+01 6.8080e+01
 2.8130e+01 6.7500e+00 1.4700e+01 3.2180e+01 1.1020e+01 3.5640e+01
 1.0500e+01 4.8800e+00 1.5600e+01 5.6300e+00 1.8000e+01 2.5510e+01
 5.9400e+00 5.5300e+00 5.8450e+01 6.4380e+01 3.8000e-01 1.6250e+01
 6.5340e+01 1.2090e+01 4.0250e+01 1.4000e+01 1.3380e+01 1.6653e+02
 2.7300e+01 4.5500e+01 1.2250e+01 6.3210e+01 5.0000e+00 5.8800e+00
 1.1700e+01 1.8600e+00 5.9400e+01 1.8200e+01 9.2000e+00 1.7750e+01
 1.8600e+01 5.3460e+01 3.7500e+00 1.7710e+01 4.8300e+01 1.7150e+01
 1.0000e+01 3.3800e+00 1.3250e+01 3.0450e+01 8.1000e+00 1.3100e+00
 1.5750e+01 7.7000e-01 8.8800e+00 3.6000e-01 1.1250e+01 1.2000e+01
```

In [8]:

```
# Checking for correlation

df.corr()
```

Out[8]:

	ID	Claim	Duration	Net Sales	Commision (in value)	Age
ID	1.000000	0.040265	0.029771	0.084391	0.114668	0.009026
Claim	0.040265	1.000000	0.076442	0.138323	0.102009	-0.012106
Duration	0.029771	0.076442	1.000000	0.437004	0.349193	0.003212
Net Sales	0.084391	0.138323	0.437004	1.000000	0.657851	0.039119
Commision (in value)	0.114668	0.102009	0.349193	0.657851	1.000000	0.119167
Age	0.009026	-0.012106	0.003212	0.039119	0.119167	1.000000

We will also drop the ID column.
Each value is unique and does not seem to affect the data.

In [9]:

```
df.drop("ID", axis=1, inplace=True)
```

In [10]:

```
# Having a look at how many claims and non-claims are present in the dataset.
print(df["Claim"].value_counts(), "\n")
(df["Claim"].value_counts()*100)/len(df)
```

```
0    49812
1      741
Name: Claim, dtype: int64
```

Out[10]:

```
0    98.534212
1     1.465788
Name: Claim, dtype: float64
```

We can see that there is a huge imbalance between the claims and non-claims.
We will build a baseline model before we perform Over Sampline and Under Sampling.

In [11]:

```
# Finding out how many customers have their age input as over 100yrs old

len(df[df["Age"] > 100])
```

Out[11]:

```
795
```

Values above 100 years would most likely be outliers. Here we take the mean of all customers above the age of 70, and replace the values over 100yrs with the new mean value

In [12]:

```
# Over here, create a variable to calculate the mean of all Senior customers.

mean_senior = df["Age"][df["Age"] > 70].mean()
```

We will now separate the categorical and numerical data.

In [13]:

```
df.nunique()
```

Out[13]:

```
Agency                   16
Agency Type               2
Distribution Channel      2
Product Name             25
Claim                     2
Duration                444
Destination             102
Net Sales              1053
Commision (in value)    964
Age                      88
dtype: int64
```

Apart from the target, "Claim", there are two more features that are bivariate - "Agency Type" and "Distribution Channel".
We could look to perform Hot Encoding on them.

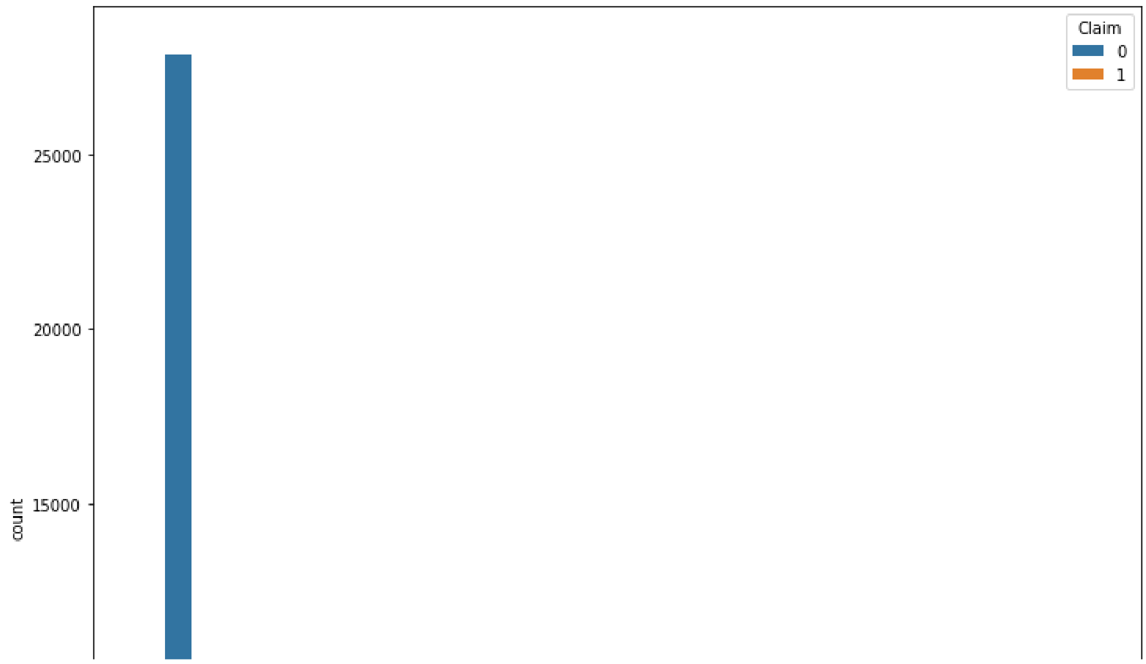
We will separate the Categorical and Numerical features, and explore them further.

In [14]:

```
cat = ["Agency", "Agency Type", "Distribution Channel", "Product Name", "Destination"]
num = ["Duration", "Net Sales", "Commision (in value)", "Age"]
```

In [15]:

```
for cols in cat:
    if (cols == "Product Name") or (cols == "Destination"):
        plt.figure(figsize=(20,30))
        sns.countplot(data=df, hue=df["Claim"], y=cols)
    else:
        plt.figure(figsize=(12,12))
        sns.countplot(data=df, hue=df["Claim"], x=cols)
    plt.xticks(rotation=90)
    plt.show()
```



In [16]:

```
for cols in num:
    plt.figure(figsize=(8,8))
    sns.boxplot(data=df, x="Claim", y=cols)
    plt.show()
```

We would need to manage only some of the outliers, and not all as it could lead to a lot of data loss.

In [17]:

```
df.describe()
```

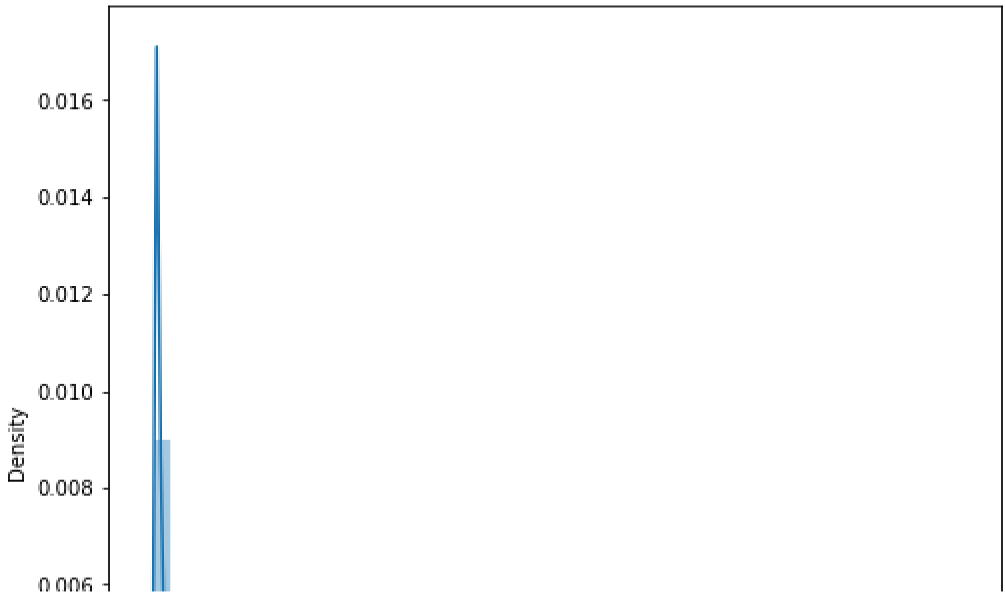
Out[17]:

	Claim	Duration	Net Sales	Commision (in value)	Age
count	50553.000000	50553.000000	50553.000000	50553.00000	50553.000000
mean	0.014658	49.425969	40.800977	9.83809	40.011236
std	0.120180	101.434647	48.899683	19.91004	14.076566
min	0.000000	-2.000000	-389.000000	0.00000	0.000000
25%	0.000000	9.000000	18.000000	0.00000	35.000000
50%	0.000000	22.000000	26.500000	0.00000	36.000000
75%	0.000000	53.000000	48.000000	11.55000	44.000000
max	1.000000	4881.000000	810.000000	283.50000	118.000000

In [18]:

```
for cols in num:
    skew_cols = skew(df[cols])
    print("{:<25} : {}" .format(cols, skew_cols))
    plt.figure(figsize=(8,8))
    sns.distplot(df[cols])
    plt.show()
```

```
Duration                  : 22.872063891229274
```



In [19]:

```
for cols in num:
    print("\n", cols)
    print(df[cols].value_counts().sort_index())
```

```
 Duration
-2         1
-1         2
 0        54
 1       647
 2      1181
        ...
 4815      1
 4829      1
 4844      1
 4857      1
 4881      1
Name: Duration, Length: 444, dtype: int64

 Net Sales
-389.00    1
-291.75    2
-289.00    1
-287.10    1
-259.20    1
          ..
 539.00    1
 599.00    5
 666.00    2
 682.00    1
 810.00    1
Name: Net Sales, Length: 1053, dtype: int64

 Commision (in value)
0.00      28079
0.02         11
0.04          1
0.05          9
0.09         10
          ...
209.95        2
210.21       33
262.60        2
262.76        6
283.50        1
Name: Commision (in value), Length: 964, dtype: int64

 Age
0        2
1        4
2        1
3        4
5        3
      ...
85       9
86       3
87       6
88       4
```

```
118    795
Name: Age, Length: 88, dtype: int64
```

There is some skewness within the data. This will be handled later on.

In [20]:

```
# The entries where the duration is -ve, we will drop those rows.

duration = df[df["Duration"] < 0].index
df.drop(duration, inplace=True)
```

In [21]:

```
df[(df["Net Sales"] < 0) & (df["Claim"] == 0)]
```

Out[21]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales
6	JZI	Airlines	Online	Value Plan	0	23	JAPAN	-69.0
128	EPX	Travel Agency	Online	Cancellation Plan	0	192	CANADA	-80.0
139	EPX	Travel Agency	Online	2 way Comprehensive Plan	0	55	CHINA	-77.0
173	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	198	NETHERLANDS	-9.9
336	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	109	AUSTRALIA	-19.8
...	...	...	...	...	...	...	...	...
50121	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	81	JAPAN	-99.0
50149	ART	Airlines	Online	24 Protect	0	2	MALAYSIA	-1.4
50177	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	75	UNITED STATES	-49.5
50394	JZI	Airlines	Online	Basic Plan	0	15	VIET NAM	-22.0
50399	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	135	AUSTRALIA	-49.5

525 rows × 10 columns

In [22]:

```
df[(df["Net Sales"] < 0) & (df["Claim"] == 1)]
```

Out[22]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Com (in
6597	EPX	Travel Agency	Online	Cancellation Plan	1	28	SPAIN	-10.0	
26666	JZI	Airlines	Online	Basic Plan	1	3	HONG KONG	-22.0	
30272	EPX	Travel Agency	Online	2 way Comprehensive Plan	1	128	KOREA, REPUBLIC OF	-37.0	

LabelEncoding, One Hot Encoding, Frequency Encoding

In [23]:

```
# Label Encoding

for cols in cat:
    le = LabelEncoder()
    df[cols] = le.fit_transform(df[cols])

df.head(8)
```

Out[23]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Ag
0	6	1	1	16	0	7	56	0.0	17.82	3
1	7	1	1	10	0	85	79	69.0	0.00	3
2	6	1	1	16	0	11	56	19.8	11.88	7
3	7	1	1	1	0	16	38	20.0	0.00	3
4	7	1	1	10	0	10	47	15.0	0.00	2
5	6	1	1	16	0	64	88	49.5	29.70	3
6	9	0	1	24	0	23	43	-69.0	24.15	2
7	9	0	1	8	0	31	34	26.0	9.10	6

```
# Frequency Encoding

fe = df.groupby('Destination').size()/len(df)
df.loc[:,'Dest Freq'] = df['Destination'].map(fe)
```

```
df.drop(columns='Destination',axis=1,inplace=True)

fe_1 = df.groupby('Agency').size()/len(df)
df.loc[:,'Agency Freq'] = df['Agency'].map(fe_1)
df.drop(columns='Agency',axis=1,inplace=True)

fe_2 = df.groupby('Product Name').size()/len(df)
df.loc[:,'Product Name Freq'] = df['Product Name'].map(fe_2)
df.drop(columns='Product Name',axis=1,inplace=True)
```

```
# One-Hot Encoding

df = pd.get_dummies(df, columns=["Agency Type", "Distribution Channel"], drop_first=True)
df.head()
```

In [24]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

Train, Test, Split

Function to train, test, and split

In [25]:

```
def tts(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1

    return X_train, X_test, y_train, y_test
```

Fit and Predict

Function to fit and predict the model, and to display the report

In [26]:

```
def model_sel(model, X_train, X_test, y_train, y_test):

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    return classification_report(y_test, y_pred)
```

All Models

Function where all the models will be defined and then passed to 'model_sel' for the model to be created.

In [27]:

```
def models(X_train, y_train, X_test="None", y_test="None", sampled="No"):
    if sampled == "No":
        X_train, X_test, y_train, y_test = tts(X_train, y_train)
    else:
        pass

    lr = LogisticRegression()
    dtc = DecisionTreeClassifier()
    abc = AdaBoostClassifier(n_estimators=100)
    gbc = GradientBoostingClassifier(n_estimators=100)
    xbc = XGBClassifier(n_estimators=200, reg_alpha=1)
    rfc = RandomForestClassifier()
    lsvc = LinearSVC(random_state=1)
    svc = SVC(random_state=1)
    print("{} \n {}\n" .format("LOGISTIC REGRESSION", model_sel(lr, X_train, X_test, y_trai
    print("{} \n {}\n" .format("DECISION TREE", model_sel(dtc, X_train, X_test, y_train, y_
    print("{} \n {}\n" .format("ADABOOST", model_sel(abc, X_train, X_test, y_train, y_test)
    print("{} \n {}\n" .format("GRADIENT BOOST", model_sel(gbc, X_train, X_test, y_train, y
    print("{} \n {}\n" .format("XGBOOST", model_sel(xbc, X_train, X_test, y_train, y_test))
    print("{} \n {}\n" .format("RANDOM FOREST", model_sel(rfc, X_train, X_test, y_train, y_
    print("{} \n {}\n" .format("LINEAR SVM", model_sel(lsvc, X_train, X_test, y_train, y_te
    print("{} \n {}\n" .format("SVM", model_sel(svc, X_train, X_test, y_train, y_test)))

    return lr, abc, gbc, xbc, rfc, lsvc, svc
```

Manual Under Sampling

We will match the number of non-claims to claims. Below are the steps

```
1. Get the count of undersampled and oversampled Claims.<br>
2. Create new variable that will randomly select the same number of oversampled Cl
aims as there is undersampled.<br>
3. Concatenate the two into a numpy array.<br>
4. Create a new DataFrame taking the indexes from the concatenated array.<br>
5. Use this DataFrame to run the models.<br>
```

In [28]:

```
def sampling(df):
    min_claim = len(df[df["Claim"] == 1])
    min_claim_ind = df[df["Claim"] == 1].index

    maj_claim_ind = df[df["Claim"] == 0].index

    random_major = np.random.choice(maj_claim_ind, min_claim, replace=False)

    sample_ind = np.concatenate([min_claim_ind, random_major])

    under_sample = df.loc[sample_ind]

    # print(sns.countplot(data=under_sample, x="Claim"))

    X = under_sample.loc[:, df.columns!="Claim"]
    y = under_sample.loc[:, df.columns=="Claim"]

    lr, abc, gbc, xbc, rfc, lsvc, svc = models(X, y)
    return lr, abc, gbc, xbc, rfc, lsvc, svc, X, y
```

Over Sampling

The number of minority values will be made to equal the number of majority values.

In [29]:

```
def over_sample():
    X = df.drop("Claim", axis=1)
    y = df["Claim"]
    print(Counter(y))
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
    oversample = RandomOverSampler(sampling_strategy='minority')
    X_over, y_over = oversample.fit_resample(X_train, y_train)
    print(Counter(y_over))

    lr, abc, gbc, xbc, rfc, lsvc, svc = models(X_over, y_over, X_test, y_test, "Yes")
    return lr, abc, gbc, xbc, rfc, lsvc, svc, X_over, y_over
```

Under Sampling

The number of majority values will be reduced down to equal the number of minority values.

In [30]:

```
def under_sample():
    X = df.drop("Claim", axis=1)
    y = df["Claim"]
    print(Counter(y))
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1
    undersample = RandomUnderSampler(sampling_strategy='majority')
    X_under, y_under = undersample.fit_resample(X_train, y_train)
    print(Counter(y_under))

    lr, abc, gbc, xbc, rfc, lsvc, svc = models(X_under, y_under, X_test, y_test, "Yes")
    return lr, abc, gbc, xbc, rfc, lsvc, svc, X_under, y_under
```

```
def samp_model_sel(model, X_train, y_train, X_test, y_test):

    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    return classification_report(y_test, y_pred)
```

```
def samp_models(X, y, X_test, y_test):
    lr = LogisticRegression()
    dtc = DecisionTreeClassifier()
    abc = AdaBoostClassifier(n_estimators=100)
    gbc = GradientBoostingClassifier(n_estimators=100)
    xbc = XGBClassifier(n_estimators=200, reg_alpha=1)
    rfc = RandomForestClassifier()
    lsvc = LinearSVC(random_state=1)
    svc = SVC(random_state=1)
    print("{} \n {}\n" .format("LOGISTIC REGRESSION", samp_model_sel(lr, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("DECISION TREE", samp_model_sel(dtc, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("ADABOOST", samp_model_sel(abc, X, y, X_test, y_test)))
    print("{} \n {}\n" .format("GRADIENT BOOST", samp_model_sel(gbc, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("XGBOOST", samp_model_sel(xbc, X, y, X_test, y_test)))
    print("{} \n {}\n" .format("RANDOM FOREST", samp_model_sel(rfc, X, y, X_test,
y_test)))
    print("{} \n {}\n" .format("LINEAR SVM", samp_model_sel(lsvc, X, y, X_test, y_test)))
    print("{} \n {}\n" .format("SVM", samp_model_sel(svc, X, y, X_test, y_test)))

    return lr, abc, gbc, xbc, rfc, lsvc, svc
```

GridSearchCV

By passing the model along with parameters that it can carry, this function will iterate using the model parameters, and deliver the best model.

In [31]:

```
def gridsearch(model, paramater, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1

    gscv = GridSearchCV(estimator=model, param_grid=parameter)
    gscv.fit(X_train, y_train)
    y_pred = gscv.predict(X_test)
    print(classification_report(y_test, y_pred))
    print(gscv.best_estimator_)
    return gscv
```

First Baseline Models

We will build four models - No Sampling, Manual Under Sampling, Over Sampled, Under Sampled.

In [32]:

```
# Without Sampling

lr, abc, gbc, xbc, rfc, lscv, svc = models(X, y)
```

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


DECISION TREE
              precision    recall  f1-score   support

           0       0.99      0.98      0.99     14952
           1       0.05      0.06      0.06       213

    accuracy                           0.97     15165
   macro avg       0.52      0.52      0.52     15165
weighted avg       0.97      0.97      0.97     15165


ADABOOST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


XGBOOST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.98     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.98      0.98     15165
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.16      0.02      0.04       213

    accuracy                           0.98     15165
   macro avg       0.57      0.51      0.52     15165
weighted avg       0.97      0.98      0.98     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


SVM
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165
```

In [33]:

```
# With manual Under Sampling

lr_sample, abc_sample, gbc_sample, xbc_sample, rfc_sample, lsvc_sample, svc_sample, X, y =
```

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.69      0.80      0.74       227
           1       0.75      0.63      0.68       218

    accuracy                           0.71       445
   macro avg       0.72      0.71      0.71       445
weighted avg       0.72      0.71      0.71       445


DECISION TREE
              precision    recall  f1-score   support

           0       0.67      0.66      0.66       227
           1       0.65      0.66      0.65       218

    accuracy                           0.66       445
   macro avg       0.66      0.66      0.66       445
weighted avg       0.66      0.66      0.66       445


ADABOOST
              precision    recall  f1-score   support

           0       0.72      0.78      0.75       227
           1       0.75      0.69      0.72       218

    accuracy                           0.74       445
   macro avg       0.74      0.74      0.74       445
weighted avg       0.74      0.74      0.74       445


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.72      0.75      0.73       227
           1       0.73      0.70      0.71       218

    accuracy                           0.72       445
   macro avg       0.72      0.72      0.72       445
weighted avg       0.72      0.72      0.72       445


XGBOOST
              precision    recall  f1-score   support

           0       0.73      0.74      0.74       227
           1       0.73      0.72      0.73       218

    accuracy                           0.73       445
   macro avg       0.73      0.73      0.73       445
weighted avg       0.73      0.73      0.73       445
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.74      0.76      0.75       227
           1       0.74      0.72      0.73       218

    accuracy                           0.74       445
   macro avg       0.74      0.74      0.74       445
weighted avg       0.74      0.74      0.74       445


LINEAR SVM
              precision    recall  f1-score   support

           0       0.74      0.69      0.71       227
           1       0.70      0.75      0.72       218

    accuracy                           0.72       445
   macro avg       0.72      0.72      0.72       445
weighted avg       0.72      0.72      0.72       445


SVM
              precision    recall  f1-score   support

           0       0.66      0.82      0.73       227
           1       0.75      0.56      0.64       218

    accuracy                           0.69       445
   macro avg       0.70      0.69      0.68       445
weighted avg       0.70      0.69      0.68       445
```

In [34]:

```
# Over Sampled

lr_over, abc, gbc_over, xbc_over, rfc_over, lsvc_over, svc_over, X, y = over_sample()
```

```
Counter({0: 49809, 1: 741})
Counter({0: 49809, 1: 49809})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14952
           1       0.05      0.62      0.10       213

    accuracy                           0.84     15165
   macro avg       0.52      0.73      0.50     15165
weighted avg       0.98      0.84      0.90     15165


DECISION TREE
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14952
           1       0.83      1.00      0.91       213

    accuracy                           1.00     15165
   macro avg       0.92      1.00      0.95     15165
weighted avg       1.00      1.00      1.00     15165


ADABOOST
              precision    recall  f1-score   support

           0       1.00      0.81      0.89     14952
           1       0.05      0.73      0.10       213

    accuracy                           0.81     15165
   macro avg       0.52      0.77      0.49     15165
weighted avg       0.98      0.81      0.88     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       1.00      0.82      0.90     14952
           1       0.06      0.76      0.11       213

    accuracy                           0.82     15165
   macro avg       0.53      0.79      0.51     15165
weighted avg       0.98      0.82      0.89     15165


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     14952
           1       0.25      1.00      0.40       213

    accuracy                           0.96     15165
   macro avg       0.63      0.98      0.69     15165
weighted avg       0.99      0.96      0.97     15165
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14952
           1       0.83      1.00      0.91       213

    accuracy                           1.00     15165
   macro avg       0.92      1.00      0.95     15165
weighted avg       1.00      1.00      1.00     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      0.29      0.45     14952
           1       0.02      0.87      0.03       213

    accuracy                           0.30     15165
   macro avg       0.51      0.58      0.24     15165
weighted avg       0.98      0.30      0.44     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     14952
           1       0.05      0.67      0.09       213

    accuracy                           0.81     15165
   macro avg       0.52      0.74      0.49     15165
weighted avg       0.98      0.81      0.88     15165
```

In [35]:

```
# Under Sampled

lr_under, abc_under, gbc_under, xbc_under, rfc_under, lsvc_under, svc_under, X, y = under_s
```

```
Counter({0: 49809, 1: 741})
Counter({0: 528, 1: 528})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14952
           1       0.05      0.62      0.10       213

    accuracy                           0.84     15165
   macro avg       0.52      0.73      0.50     15165
weighted avg       0.98      0.84      0.90     15165


DECISION TREE
              precision    recall  f1-score   support

           0       0.99      0.68      0.81     14952
           1       0.03      0.66      0.05       213

    accuracy                           0.68     15165
   macro avg       0.51      0.67      0.43     15165
weighted avg       0.98      0.68      0.80     15165


ADABOOST
              precision    recall  f1-score   support

           0       0.99      0.78      0.88     14952
           1       0.04      0.72      0.08       213

    accuracy                           0.78     15165
   macro avg       0.52      0.75      0.48     15165
weighted avg       0.98      0.78      0.86     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.99      0.80      0.89     14952
           1       0.05      0.71      0.09       213

    accuracy                           0.80     15165
   macro avg       0.52      0.76      0.49     15165
weighted avg       0.98      0.80      0.88     15165


XGBOOST
              precision    recall  f1-score   support

           0       0.99      0.75      0.85     14952
           1       0.04      0.71      0.07       213

    accuracy                           0.75     15165
   macro avg       0.52      0.73      0.46     15165
weighted avg       0.98      0.75      0.84     15165
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.99      0.76      0.86     14952
           1       0.04      0.69      0.07       213

    accuracy                           0.76     15165
   macro avg       0.52      0.72      0.47     15165
weighted avg       0.98      0.76      0.85     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       1.00      0.10      0.19     14952
           1       0.02      0.99      0.03       213

    accuracy                           0.12     15165
   macro avg       0.51      0.54      0.11     15165
weighted avg       0.98      0.12      0.19     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     14952
           1       0.04      0.54      0.07       213

    accuracy                           0.81     15165
   macro avg       0.52      0.68      0.48     15165
weighted avg       0.98      0.81      0.88     15165
```

Result

The scores are all zero for the base model without Sampling. For all the sampling models, the scores increased drastically. Over Sampled models produced the best results. Going forward, we will not run the models where no sampling is done.

```
### Outliers

<b>Those over 100yrs will be replaced by the mean of Senior aged customers, and where the Duration is more that 360 will be replaced by 360.
```

In [36]:

```
df["Age"][df["Age"] > 60] = mean_senior
```

In [37]:

```
df["Duration"][df["Duration"] > 360] = 360
```

In [38]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [39]:

```
# lr_out, abc_out, gbc_out, xbc_out, rfc_out, lsvc_out, svc_out = models(X, y)
```

In [40]:

```
# Manual Under Sampling and Outliers

lr_out_sample, abc_out_sample, gbc_out_sample, xbc_out_sample, rfc_out_sample, lsvc_out_sam
```

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.72      0.83      0.77       227
           1       0.79      0.66      0.72       218

    accuracy                           0.75       445
   macro avg       0.75      0.74      0.74       445
weighted avg       0.75      0.75      0.74       445


DECISION TREE
              precision    recall  f1-score   support

           0       0.65      0.63      0.64       227
           1       0.62      0.64      0.63       218

    accuracy                           0.63       445
   macro avg       0.63      0.63      0.63       445
weighted avg       0.63      0.63      0.63       445


ADABOOST
              precision    recall  f1-score   support

           0       0.75      0.78      0.76       227
           1       0.76      0.73      0.75       218

    accuracy                           0.76       445
   macro avg       0.76      0.75      0.75       445
weighted avg       0.76      0.76      0.75       445


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.74      0.78      0.76       227
           1       0.76      0.72      0.74       218

    accuracy                           0.75       445
   macro avg       0.75      0.75      0.75       445
weighted avg       0.75      0.75      0.75       445


XGBOOST
              precision    recall  f1-score   support

           0       0.73      0.74      0.74       227
           1       0.73      0.72      0.72       218

    accuracy                           0.73       445
   macro avg       0.73      0.73      0.73       445
weighted avg       0.73      0.73      0.73       445
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.75      0.78      0.76       227
           1       0.76      0.72      0.74       218

    accuracy                           0.75       445
   macro avg       0.75      0.75      0.75       445
weighted avg       0.75      0.75      0.75       445


LINEAR SVM
              precision    recall  f1-score   support

           0       0.74      0.52      0.61       227
           1       0.62      0.81      0.70       218

    accuracy                           0.66       445
   macro avg       0.68      0.67      0.66       445
weighted avg       0.68      0.66      0.66       445


SVM
              precision    recall  f1-score   support

           0       0.68      0.77      0.72       227
           1       0.72      0.62      0.67       218

    accuracy                           0.70       445
   macro avg       0.70      0.70      0.69       445
weighted avg       0.70      0.70      0.69       445
```

In [41]:

```
# Over Sampling and Outliers

lr_out_over, abc_out_over, gbc_out_over, xbc_out_over, rfc_out_over, lsvc_out_over, svc_out
```

```
Counter({0: 49809, 1: 741})
Counter({0: 49809, 1: 49809})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14952
           1       0.05      0.63      0.10       213

    accuracy                           0.83     15165
   macro avg       0.52      0.74      0.50     15165
weighted avg       0.98      0.83      0.90     15165


DECISION TREE
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     14952
           1       0.53      1.00      0.69       213

    accuracy                           0.99     15165
   macro avg       0.76      0.99      0.84     15165
weighted avg       0.99      0.99      0.99     15165


ADABOOST
              precision    recall  f1-score   support

           0       1.00      0.80      0.89     14952
           1       0.05      0.73      0.09       213

    accuracy                           0.80     15165
   macro avg       0.52      0.77      0.49     15165
weighted avg       0.98      0.80      0.88     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       1.00      0.83      0.90     14952
           1       0.06      0.75      0.11       213

    accuracy                           0.83     15165
   macro avg       0.53      0.79      0.51     15165
weighted avg       0.98      0.83      0.89     15165


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     14952
           1       0.24      1.00      0.39       213

    accuracy                           0.96     15165
   macro avg       0.62      0.98      0.68     15165
weighted avg       0.99      0.96      0.97     15165
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     14952
           1       0.53      1.00      0.69       213

    accuracy                           0.99     15165
   macro avg       0.76      0.99      0.84     15165
weighted avg       0.99      0.99      0.99     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      0.97      0.98     14952
           1       0.08      0.20      0.12       213

    accuracy                           0.96     15165
   macro avg       0.54      0.58      0.55     15165
weighted avg       0.98      0.96      0.97     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     14952
           1       0.05      0.66      0.09       213

    accuracy                           0.81     15165
   macro avg       0.52      0.74      0.49     15165
weighted avg       0.98      0.81      0.88     15165
```

In [42]:

```
# Under Sampling and Outliers

lr_out_under, abc_out_under, gbc_out_under, xbc_out_under, rfc_out_under, lsvc_out_under, s
```

```
Counter({0: 49809, 1: 741})
Counter({0: 528, 1: 528})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.85      0.91     14952
           1       0.05      0.62      0.10       213

    accuracy                           0.84     15165
   macro avg       0.52      0.74      0.51     15165
weighted avg       0.98      0.84      0.90     15165


DECISION TREE
              precision    recall  f1-score   support

           0       0.99      0.66      0.79     14952
           1       0.03      0.66      0.05       213

    accuracy                           0.66     15165
   macro avg       0.51      0.66      0.42     15165
weighted avg       0.98      0.66      0.78     15165


ADABOOST
              precision    recall  f1-score   support

           0       1.00      0.78      0.88     14952
           1       0.05      0.73      0.09       213

    accuracy                           0.78     15165
   macro avg       0.52      0.76      0.48     15165
weighted avg       0.98      0.78      0.87     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.99      0.78      0.88     14952
           1       0.04      0.71      0.08       213

    accuracy                           0.78     15165
   macro avg       0.52      0.75      0.48     15165
weighted avg       0.98      0.78      0.87     15165


XGBOOST
              precision    recall  f1-score   support

           0       0.99      0.73      0.84     14952
           1       0.03      0.67      0.07       213

    accuracy                           0.73     15165
   macro avg       0.51      0.70      0.45     15165
weighted avg       0.98      0.73      0.83     15165
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.99      0.75      0.86     14952
           1       0.04      0.66      0.07       213

    accuracy                           0.75     15165
   macro avg       0.52      0.71      0.46     15165
weighted avg       0.98      0.75      0.85     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       1.00      0.37      0.54     14952
           1       0.02      0.92      0.04       213

    accuracy                           0.38     15165
   macro avg       0.51      0.65      0.29     15165
weighted avg       0.98      0.38      0.53     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.79      0.88     14952
           1       0.04      0.55      0.07       213

    accuracy                           0.79     15165
   macro avg       0.51      0.67      0.48     15165
weighted avg       0.98      0.79      0.87     15165
```

Skewness

In [43]:

```
print("{:<15} : {}" .format("Duration", skew(df["Duration"])))
print("{:<15} : {}" .format("Commision (in value)", skew(df["Commision (in value)"])))
print("{:<15} : {}" .format("Age", skew(df["Age"])))
```

```
Duration        : 3.0381146944318584
Commision (in value) : 4.077929249694879
Age             : 2.5043395070093535
```

In [44]:

```
df["Duration"] = np.sqrt(df["Duration"])
df["Commision (in value)"] = np.sqrt(df["Commision (in value)"])
df["Age"] = np.sqrt(df["Age"])
```

In [45]:

```
print("{:<15} : {}" .format("Duration", skew(df["Duration"])))
print("{:<15} : {}" .format("Commision (in value)", skew(df["Commision (in value)"])))
print("{:<15} : {}" .format("Age", skew(df["Age"])))
```

```
Duration        : 1.6361035948952554
Commision (in value) : 1.3513398200630384
Age             : 1.9417002579049916
```

In [46]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [47]:

```
# lr_skew, abc_skew, gbc_skew, xbc_skew, rfc_skew, lsvc_skew, svc_skew = models(X, y)
```

In [48]:

```
# Manual Under Sampling and Skewing

lr_skew_sample, abc_skew_sample, gbc_skew_sample, xbc_skew_sample, rfc_skew_sample, lsvc_sk
```

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.70      0.82      0.76       227
           1       0.77      0.63      0.69       218

    accuracy                           0.73       445
   macro avg       0.74      0.73      0.72       445
weighted avg       0.74      0.73      0.73       445


DECISION TREE
              precision    recall  f1-score   support

           0       0.68      0.70      0.69       227
           1       0.67      0.65      0.66       218

    accuracy                           0.67       445
   macro avg       0.67      0.67      0.67       445
weighted avg       0.67      0.67      0.67       445


ADABOOST
              precision    recall  f1-score   support

           0       0.71      0.74      0.73       227
           1       0.72      0.69      0.70       218

    accuracy                           0.71       445
   macro avg       0.71      0.71      0.71       445
weighted avg       0.71      0.71      0.71       445


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.70      0.73      0.72       227
           1       0.71      0.67      0.69       218

    accuracy                           0.70       445
   macro avg       0.70      0.70      0.70       445
weighted avg       0.70      0.70      0.70       445


XGBOOST
              precision    recall  f1-score   support

           0       0.70      0.70      0.70       227
           1       0.69      0.68      0.69       218

    accuracy                           0.69       445
   macro avg       0.69      0.69      0.69       445
weighted avg       0.69      0.69      0.69       445
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.71      0.72      0.71       227
           1       0.70      0.70      0.70       218

    accuracy                           0.71       445
   macro avg       0.71      0.71      0.71       445
weighted avg       0.71      0.71      0.71       445


LINEAR SVM
              precision    recall  f1-score   support

           0       0.59      0.89      0.71       227
           1       0.75      0.37      0.49       218

    accuracy                           0.63       445
   macro avg       0.67      0.63      0.60       445
weighted avg       0.67      0.63      0.60       445


SVM
              precision    recall  f1-score   support

           0       0.67      0.76      0.71       227
           1       0.71      0.61      0.66       218

    accuracy                           0.69       445
   macro avg       0.69      0.69      0.69       445
weighted avg       0.69      0.69      0.69       445
```

In [49]:

```
# Over Sampling and Skewing

lr_skew_over, abc_skew_over, gbc_skew_over, xbc_skew_over, rfc_skew_over, lsvc_skew_over, s
```

```
Counter({0: 49809, 1: 741})
Counter({0: 49809, 1: 49809})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.83      0.91     14952
           1       0.05      0.63      0.10       213

    accuracy                           0.83     15165
   macro avg       0.52      0.73      0.50     15165
weighted avg       0.98      0.83      0.90     15165


DECISION TREE
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     14952
           1       0.53      1.00      0.69       213

    accuracy                           0.99     15165
   macro avg       0.76      0.99      0.84     15165
weighted avg       0.99      0.99      0.99     15165


ADABOOST
              precision    recall  f1-score   support

           0       1.00      0.81      0.89     14952
           1       0.05      0.75      0.10       213

    accuracy                           0.81     15165
   macro avg       0.52      0.78      0.49     15165
weighted avg       0.98      0.81      0.88     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       1.00      0.82      0.90     14952
           1       0.06      0.75      0.11       213

    accuracy                           0.82     15165
   macro avg       0.53      0.79      0.50     15165
weighted avg       0.98      0.82      0.89     15165


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     14952
           1       0.24      1.00      0.39       213

    accuracy                           0.96     15165
   macro avg       0.62      0.98      0.69     15165
weighted avg       0.99      0.96      0.97     15165
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     14952
           1       0.53      1.00      0.69       213

    accuracy                           0.99     15165
   macro avg       0.76      0.99      0.84     15165
weighted avg       0.99      0.99      0.99     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       1.00      0.01      0.02     14952
           1       0.01      1.00      0.03       213

    accuracy                           0.03     15165
   macro avg       0.51      0.51      0.03     15165
weighted avg       0.99      0.03      0.02     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.83      0.90     14952
           1       0.05      0.66      0.09       213

    accuracy                           0.82     15165
   macro avg       0.52      0.74      0.50     15165
weighted avg       0.98      0.82      0.89     15165
```

In [50]:

```
# Under Sampling and Skewing

lr_skew_under, abc_skew_under, gbc_skew_under, xbc_skew_under, rfc_skew_under, lsvc_skew_un
```

```
Counter({0: 49809, 1: 741})
Counter({0: 528, 1: 528})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     14952
           1       0.05      0.65      0.09       213

    accuracy                           0.81     15165
   macro avg       0.52      0.73      0.49     15165
weighted avg       0.98      0.81      0.88     15165


DECISION TREE
              precision    recall  f1-score   support

           0       0.99      0.67      0.80     14952
           1       0.03      0.62      0.05       213

    accuracy                           0.67     15165
   macro avg       0.51      0.64      0.43     15165
weighted avg       0.98      0.67      0.79     15165


ADABOOST
              precision    recall  f1-score   support

           0       0.99      0.76      0.86     14952
           1       0.04      0.68      0.07       213

    accuracy                           0.76     15165
   macro avg       0.52      0.72      0.47     15165
weighted avg       0.98      0.76      0.85     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.99      0.76      0.86     14952
           1       0.04      0.69      0.08       213

    accuracy                           0.76     15165
   macro avg       0.52      0.73      0.47     15165
weighted avg       0.98      0.76      0.85     15165


XGBOOST
              precision    recall  f1-score   support

           0       0.99      0.72      0.83     14952
           1       0.03      0.69      0.06       213

    accuracy                           0.72     15165
   macro avg       0.51      0.70      0.45     15165
```

```
weighted avg       0.98      0.72      0.82     15165


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.99      0.75      0.85     14952
           1       0.04      0.69      0.07       213

    accuracy                           0.75     15165
   macro avg       0.52      0.72      0.46     15165
weighted avg       0.98      0.75      0.84     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      0.94      0.97     14952
           1       0.06      0.25      0.10       213

    accuracy                           0.94     15165
   macro avg       0.52      0.60      0.53     15165
weighted avg       0.98      0.94      0.95     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.80      0.89     14952
           1       0.04      0.57      0.07       213

    accuracy                           0.80     15165
   macro avg       0.52      0.68      0.48     15165
weighted avg       0.98      0.80      0.87     15165
```

Performing Chi-squared test

In [51]:

```
len(df.columns)
```

Out[51]:

```
10
```

In [52]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [53]:

```
X_cols = []

for col in X:
    X_cols.append(col)
```

In [54]:

```
def model_chi(model, X):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1

    chi_test = SelectKBest(score_func=chi2, k=6)

    X_train_chi = chi_test.fit_transform(X_train, y_train)
    X_test_chi = chi_test.transform(X_test)

    model.fit(X_train_chi, y_train)
    y_pred = model.predict(X_test_chi)

    print(classification_report(y_test, y_pred))

    num = 0

    for each in chi_test.scores_:
        print("{:2} {:20} - {}" .format(num, X_cols[num], each))
        num += 1
```

In [55]:

```
def model_new(X):
    lr = LogisticRegression()
    dtc = DecisionTreeClassifier(criterion="entropy")
    abc = AdaBoostClassifier(n_estimators=100)
    gbc = GradientBoostingClassifier(n_estimators=100)
    xbc = XGBClassifier(n_estimators=200, reg_alpha=1)
    rfc = RandomForestClassifier()
    print("{} \n {}\n" .format("LOGISTIC REGRESSION", model_chi(lr,X)))
    print("{} \n {}\n" .format("DECISION TREE", model_sel(dtc,X)))
    print("{} \n {}\n" .format("ADABOOST", model_chi(abc,X)))
    print("{} \n {}\n" .format("GRADIENT BOOST", model_chi(gbc,X)))
    print("{} \n {}\n" .format("XGBOOST", model_chi(xbc,X)))
    print("{} \n {}\n" .format("RANDOM FOREST", model_chi(rfc,X)))

    return lr, abc, gbc, xbc, rfc
```

lr_chi, abc_chi, gbc_chi, xbc_chi, rfc_chi = model_new(X)

RandomForest still has the best score.

In [56]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [57]:

```
X_cols = []

for col in X:
    X_cols.append(col)
```

In [58]:

```
# lr_chi, abc_chi, gbc_chi, xbc_chi, rfc_chi = model_new(X)
```

Scaling

In [59]:

```
df_old = df.copy(deep=True)
```

In [60]:

```
mm = MinMaxScaler()

X = df.drop("Claim", axis=1)
cols = X.columns.to_list()
df[cols] = mm.fit_transform(df[cols])
```

In [61]:

```
df.head()
```

Out[61]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commisio (in value
0	0.400000	1.0	1.0	0.666667	0	0.139443	0.554455	0.324437	0.25071
1	0.466667	1.0	1.0	0.416667	0	0.485913	0.782178	0.381985	0.00000
2	0.400000	1.0	1.0	0.666667	0	0.174801	0.554455	0.340951	0.20470
3	0.466667	1.0	1.0	0.041667	0	0.210819	0.376238	0.341118	0.00000
4	0.466667	1.0	1.0	0.416667	0	0.166667	0.465347	0.336947	0.00000

In [62]:

```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [63]:

```
# lr_scale, abc_scale, gbc_scale, xbc_scale, rfc_scale, lsvc_scale, svc_scale = models(X, y
```

In [64]:

```
# Manual Under Sampling and Scalling

lr_scale_sample, abc_scale_sample, gbc_scale_sample, xbc_scale_sample, rfc_scale_sample, ls
```

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.68      0.78      0.73       227
           1       0.73      0.63      0.67       218

    accuracy                           0.70       445
   macro avg       0.71      0.70      0.70       445
weighted avg       0.71      0.70      0.70       445


DECISION TREE
              precision    recall  f1-score   support

           0       0.70      0.64      0.67       227
           1       0.65      0.71      0.68       218

    accuracy                           0.67       445
   macro avg       0.68      0.67      0.67       445
weighted avg       0.68      0.67      0.67       445


ADABOOST
              precision    recall  f1-score   support

           0       0.72      0.74      0.73       227
           1       0.72      0.71      0.71       218

    accuracy                           0.72       445
   macro avg       0.72      0.72      0.72       445
weighted avg       0.72      0.72      0.72       445


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.71      0.76      0.74       227
           1       0.73      0.68      0.70       218

    accuracy                           0.72       445
   macro avg       0.72      0.72      0.72       445
weighted avg       0.72      0.72      0.72       445


XGBOOST
              precision    recall  f1-score   support

           0       0.69      0.69      0.69       227
           1       0.68      0.68      0.68       218

    accuracy                           0.69       445
   macro avg       0.69      0.69      0.69       445
weighted avg       0.69      0.69      0.69       445
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.70      0.72      0.71       227
           1       0.70      0.68      0.69       218

    accuracy                           0.70       445
   macro avg       0.70      0.70      0.70       445
weighted avg       0.70      0.70      0.70       445


LINEAR SVM
              precision    recall  f1-score   support

           0       0.70      0.81      0.75       227
           1       0.76      0.63      0.69       218

    accuracy                           0.72       445
   macro avg       0.73      0.72      0.72       445
weighted avg       0.73      0.72      0.72       445


SVM
              precision    recall  f1-score   support

           0       0.70      0.82      0.76       227
           1       0.78      0.63      0.70       218

    accuracy                           0.73       445
   macro avg       0.74      0.73      0.73       445
weighted avg       0.74      0.73      0.73       445
```

In [65]:

```
# Over Sampling and Scalling

lr_scale_over, abc_scale_over, gbc_scale_over, xbc_scale_over, rfc_scale_over, lsvc_scale_o
```

```
Counter({0: 49809, 1: 741})
Counter({0: 49809, 1: 49809})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.83      0.91     14952
           1       0.05      0.63      0.09       213

    accuracy                           0.83     15165
   macro avg       0.52      0.73      0.50     15165
weighted avg       0.98      0.83      0.89     15165


DECISION TREE
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     14952
           1       0.53      1.00      0.69       213

    accuracy                           0.99     15165
   macro avg       0.76      0.99      0.84     15165
weighted avg       0.99      0.99      0.99     15165


ADABOOST
              precision    recall  f1-score   support

           0       1.00      0.80      0.89     14952
           1       0.05      0.73      0.09       213

    accuracy                           0.80     15165
   macro avg       0.52      0.76      0.49     15165
weighted avg       0.98      0.80      0.87     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       1.00      0.82      0.90     14952
           1       0.06      0.76      0.10       213

    accuracy                           0.82     15165
   macro avg       0.53      0.79      0.50     15165
weighted avg       0.98      0.82      0.89     15165


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.96      0.98     14952
           1       0.24      1.00      0.39       213

    accuracy                           0.96     15165
   macro avg       0.62      0.98      0.68     15165
```

```
weighted avg       0.99      0.96      0.97     15165


RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     14952
           1       0.53      1.00      0.69       213

    accuracy                           0.99     15165
   macro avg       0.76      0.99      0.84     15165
weighted avg       0.99      0.99      0.99     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14952
           1       0.05      0.63      0.10       213

    accuracy                           0.83     15165
   macro avg       0.52      0.74      0.50     15165
weighted avg       0.98      0.83      0.90     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14952
           1       0.06      0.67      0.10       213

    accuracy                           0.84     15165
   macro avg       0.53      0.76      0.51     15165
weighted avg       0.98      0.84      0.90     15165
```

In [66]:

```
# Under Sampling and Scalling

lr_scale_under, abc_scale_under, gbc_scale_under, xbc_scale_under, rfc_scale_under, lsvc_sc
```

```
Counter({0: 49809, 1: 741})
Counter({0: 528, 1: 528})
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      0.82      0.90     14952
           1       0.05      0.63      0.09       213

    accuracy                           0.82     15165
   macro avg       0.52      0.73      0.50     15165
weighted avg       0.98      0.82      0.89     15165


DECISION TREE
              precision    recall  f1-score   support

           0       0.99      0.68      0.80     14952
           1       0.03      0.66      0.05       213

    accuracy                           0.68     15165
   macro avg       0.51      0.67      0.43     15165
weighted avg       0.98      0.68      0.79     15165


ADABOOST
              precision    recall  f1-score   support

           0       0.99      0.77      0.87     14952
           1       0.04      0.70      0.08       213

    accuracy                           0.77     15165
   macro avg       0.52      0.74      0.47     15165
weighted avg       0.98      0.77      0.86     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.99      0.77      0.87     14952
           1       0.04      0.69      0.08       213

    accuracy                           0.77     15165
   macro avg       0.52      0.73      0.47     15165
weighted avg       0.98      0.77      0.86     15165


XGBOOST
              precision    recall  f1-score   support

           0       0.99      0.73      0.85     14952
           1       0.04      0.70      0.07       213

    accuracy                           0.73     15165
   macro avg       0.52      0.72      0.46     15165
```

```
weighted avg       0.98      0.73      0.83     15165


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.99      0.76      0.86     14952
           1       0.04      0.65      0.07       213

    accuracy                           0.76     15165
   macro avg       0.52      0.71      0.47     15165
weighted avg       0.98      0.76      0.85     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14952
           1       0.05      0.64      0.10       213

    accuracy                           0.83     15165
   macro avg       0.52      0.74      0.50     15165
weighted avg       0.98      0.83      0.90     15165


SVM
              precision    recall  f1-score   support

           0       0.99      0.86      0.92     14952
           1       0.06      0.62      0.11       213

    accuracy                           0.86     15165
   macro avg       0.53      0.74      0.52     15165
weighted avg       0.98      0.86      0.91     15165
```

Saving best model in a file through Pickle

In [67]:

```
file = open("TravelInsurance.ser", "wb")
pickle.dump(rfc_under, file)
file.close()
```

```
<hr>
```

Verdict

Version 1) Following are the process involved -

```
a) Read and analyzed dataset.
b) Removed 'Gender' as it had 71% null values.
c) Performed Label Encoding.
d) Created defintions for fitting and predicting models.
e) Skewness, Outliers, Scaling, Chi-Squared Test, Boosting.
```

Result -The scores achieved for each and every model in this version was zero (as you can see below). A different approach was required.

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


DECISION TREE
              precision    recall  f1-score   support

           0       0.99      0.98      0.99     14952
           1       0.05      0.06      0.06       213

    accuracy                           0.97     15165
   macro avg       0.52      0.52      0.52     15165
weighted avg       0.97      0.97      0.97     15165


ADABOOST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165
```

```
XGBOOST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.98     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.98      0.98     15165


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.16      0.02      0.04       213

    accuracy                           0.98     15165
   macro avg       0.57      0.51      0.52     15165
weighted avg       0.97      0.98      0.98     15165


LINEAR SVM
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


SVM
              precision    recall  f1-score   support

           0       0.99      1.00      0.99     14952
           1       0.00      0.00      0.00       213

    accuracy                           0.99     15165
   macro avg       0.49      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165
```

Version 2

From this version onwards, Sampling techniques were added. This helped increase the score value greatly. The definition added was 'sampling(df)'. This technique manually applied undersampling. Some of the best scores achieved are shown below. Also, updates we done to Boosting. Along with other models, they were added to Bagging Classifier with parameters, and then passed to GridSearchCV. It is important to note that Sampling should only be done on the Training data, and not on the entire dataset.

Adaboost Baseline Sampling

```
ADABOOST
              precision    recall  f1-score   support

           0       0.75      0.76      0.76       227
           1       0.75      0.73      0.74       218

    accuracy                           0.75       445
   macro avg       0.75      0.75      0.75       445
weighted avg       0.75      0.75      0.75       445
```

RandomForest Skew Sampling

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.74      0.75      0.75       227
           1       0.74      0.73      0.73       218

    accuracy                           0.74       445
   macro avg       0.74      0.74      0.74       445
weighted avg       0.74      0.74      0.74       445
```

XGBoost and RandomForest Scaling Sampling

```
XGBOOST
              precision    recall  f1-score   support

           0       0.74      0.74      0.74       227
           1       0.73      0.73      0.73       218

    accuracy                           0.73       445
   macro avg       0.73      0.73      0.73       445
weighted avg       0.73      0.73      0.73       445


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       227
           1       0.77      0.72      0.74       218

    accuracy                           0.76       445
   macro avg       0.76      0.76      0.76       445
weighted avg       0.76      0.76      0.76       445
```

Gradient Boosting GridSearch Sampling

```
              precision    recall  f1-score   support

           0       0.75      0.74      0.74       227
           1       0.73      0.75      0.74       218

    accuracy                           0.74       445
   macro avg       0.74      0.74      0.74       445
weighted avg       0.74      0.74      0.74       445

GradientBoostingClassifier(max_depth=6, n_estimators=46)
```

LinearSVC Baseline Sampling

```
LINEAR SVM
              precision    recall  f1-score   support

           0       0.96      0.23      0.37       227
           1       0.55      0.99      0.71       218

    accuracy                           0.60       445
   macro avg       0.76      0.61      0.54       445
weighted avg       0.76      0.60      0.54       445
```

LinearSVC Outliers Sampling

```
LINEAR SVM
              precision    recall  f1-score   support

           0       0.83      0.43      0.56       227
           1       0.60      0.91      0.73       218

    accuracy                           0.66       445
   macro avg       0.72      0.67      0.64       445
weighted avg       0.72      0.66      0.64       445
```

Final Version

Here, we added the function 'under_sample()' and 'over_sample()'. All Boosting, Bagging, and GridSearch code blocks were changed to Raw in this version. Reason being that Over Sampling greatly increased the score values right from the Baseline models (screenshot below) onwards, especially for DecisionTree and RandomForest.

```
DECISION TREE
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14952
           1       0.83      1.00      0.91       213

    accuracy                           1.00     15165
   macro avg       0.92      1.00      0.95     15165
weighted avg       1.00      1.00      1.00     15165

RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     14952
           1       0.83      1.00      0.91       213

    accuracy                           1.00     15165
   macro avg       0.92      1.00      0.95     15165
weighted avg       1.00      1.00      1.00     15165
```

Overall, RandomForest produced the best results. Even after some EDA and Preprocessing, the scores achieved for DecisionTree and RandomForest after each EDA process were almost identical, although there was a bit of variance in scores between the models. For this, we saved the model 'rfc_under' into a serial file through Pickle.