Machine Learning Project

The goal of this project is to create a model that can predict for whether a customer can claim for Travel Insurance or not.

Problem Statement

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.

The Steps performed will be mentioned as we go through the project.

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, RandomFore
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("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	Commisio (in valu
0	3433	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	7	MALAYSIA	0.0	17.8
1	4339	EPX	Travel Agency	Online	Cancellation Plan	0	85	SINGAPORE	69.0	0.0
2	34590	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	11	MALAYSIA	19.8	11.8
3	55816	EPX	Travel Agency	Online	2 way Comprehensive Plan	0	16	INDONESIA	20.0	0.0
4	13816	EPX	Travel Agency	Online	Cancellation Plan	0	10	KOREA, REPUBLIC OF	15.0	0.0
5	50349	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	64	THAILAND	49.5	29.7
6	9921	JZI	Airlines	Online	Value Plan	0	23	JAPAN	-69.0	24.1
7	21923	JZI	Airlines	Online	Basic Plan	0	31	HONG KONG	26.0	9.1

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()))
```

```
Id                   - [ 3433  4339 34590 ... 54146 28667 50880]
```

```
Agency               - ['CWT' 'EPX' 'JZI' 'C2B' 'SSI' 'CSR' 'KML' 'RAB' 'ADM' 'JWT' 'LW
C' 'TST'
 'ART' 'TTW' 'CBH' 'CCR']

Agency Type          - ['Travel Agency' 'Airlines']

Distribution Channel - ['Online' 'Offline']

Product Name         - ['Rental Vehicle Excess Insurance' 'Cancellation Plan'
 '2 way Comprehensive Plan' 'Value Plan' 'Basic Plan' 'Bronze Plan'
 'Ticket Protector' '1 way Comprehensive Plan' 'Comprehensive Plan'
 'Silver Plan' 'Premier Plan' 'Annual Silver Plan' 'Annual Gold Plan'
 'Single Trip Travel Protect Silver' 'Travel Cruise Protect' '24 Protect'
 'Annual Travel Protect Gold' 'Single Trip Travel Protect Platinum'
 'Single Trip Travel Protect Gold' 'Spouse or Parents Comprehensive Plan'
 'Gold Plan' 'Annual Travel Protect Silver'
 'Individual Comprehensive Plan' 'Annual Travel Protect Platinum'
 'Child Comprehensive Plan']

Claim                - [0 1]

Duration             - [   7   85   11   16   10   64   23   31    5  100   71   18   15
20
   27  162  240  140    8   36    3   14   50   41   72   26   19    6
    1   81  133    2   21   12   30   56  132   55   28   61   69    4
   35  386   47   32  102   37   34   40   57   46   58  147   17   70
  375  186   13   76   62  431  138  127   74   52   45  192   22  109
   29  129  219   80  137   24  198  126  264    9   91    0  307  135
   38  166  403   66  365   63  180   87  174   68  125   94   92   43
  115  391   77   84   33   65   79   48   73   42   25  101  104  123
  373  366   95  106   78  151   75   83  112  364   39  208   97  371
   67   59  261  148  367   44   88  167   89  121  182   53  243  110
  369   54  202  122  142  163  124  268  150  170  117   49  372   86
  390  224  381   99   93  141  374  204  155   60  508   90  387  168
  217  252  159  160  116  236  205  247  114  183  370   51  176  190
   96   82  119  228  392  156  143  144  378  145  545  153  257  113
  189  200  253  191  214  107  108  368  131  377  128  275  199  397
  395  310  379  318  380  179  154  118  231  385  105  120  270  399
  149  284  384  300  169  146  197  103  255  218  376  196  238  383
  244  230  165  161  187  394  181  130  389  188  410  426  111  227
  478  195  211  152   98  461  287  164  388  134  427  269  303  249
  309  194  233  215   -1  203  157  136  173  305  317  398  178  193
  223  416  289  158  382  177  325  490  235  277  428  322  226  411
  445  294  206  419  319  222  172  267  175  302  263  297  274  436
  400  420  402  547  185  237  245  448  225  417  435  291  418  279
  306  248  312  246  472  278  216  272  242  221  456  207  184  296
  531  254  241  406  271  260  326  440  220  404  432  330  229  273
  286  450  487  234  299  232  251  171 4609  425  285  210  276  421
  463  256  209  314  281  139  298  327  401  414 4580  393  412  288
  408  396 4685  213  430  311 4736  323  266  351  258  265  304  280
  212  352  250  529  444 4844  292  259  201  283  239  334  321  405
  433  512 4857  413 4815  424  474  494  350  315 4652  488 4829  282
  740  262  293  332  329  316  328  465  342  519  331  290  407  423
  457 4738  409 4881  459  295   -2  415  324  434]

Destination          - ['MALAYSIA' 'SINGAPORE' 'INDONESIA' 'KOREA, REPUBLIC OF' 'THAILAN
D'
 'JAPAN' 'HONG KONG' 'AUSTRALIA' 'UNITED STATES' 'CHINA' 'SPAIN' 'MEXICO'
 'UNITED KINGDOM' 'TAIWAN, PROVINCE OF CHINA' 'CANADA' 'PHILIPPINES'
 'ARGENTINA' 'VIET NAM' 'BRUNEI DARUSSALAM' 'UNITED ARAB EMIRATES'
 'NORWAY' 'BANGLADESH' 'ISRAEL' 'CAMBODIA' 'INDIA' 'SWITZERLAND'
 "LAO PEOPLE'S DEMOCRATIC REPUBLIC" 'NEW ZEALAND' 'MYANMAR' 'TURKEY'
 'FRANCE' 'MALDIVES' 'AUSTRIA' 'PORTUGAL' 'NETHERLANDS' 'ICELAND' 'ITALY'
 'GERMANY' 'RUSSIAN FEDERATION' 'NEPAL' 'SLOVENIA' 'PERU' 'GREECE'
```

```
 'COLOMBIA' 'MAURITIUS' 'IRELAND' 'GEORGIA' 'SRI LANKA' 'SOUTH AFRICA'
 'BRAZIL' 'CROATIA' 'FIJI' 'KAZAKHSTAN' 'DENMARK' 'KENYA' 'SWEDEN' 'CHILE'
 'MOROCCO' 'FINLAND' 'BOLIVIA' 'TANZANIA, UNITED REPUBLIC OF' 'BELGIUM'
 'PAKISTAN' 'SAUDI ARABIA' 'QATAR' 'CZECH REPUBLIC' 'MACAO' 'MALTA'
 'UGANDA' 'POLAND' 'BAHRAIN' 'GUAM' 'UZBEKISTAN' 'EGYPT' 'HUNGARY'
 'LATVIA' 'JORDAN' 'OMAN' 'ETHIOPIA' 'UKRAINE' 'COSTA RICA' 'ROMANIA'
 'MONGOLIA' 'AZERBAIJAN' 'LITHUANIA' 'KUWAIT' 'CYPRUS' 'LEBANON'
 'LUXEMBOURG' 'BELARUS' 'ESTONIA' 'TUNISIA' 'VANUATU' 'TURKMENISTAN'
 'NORTHERN MARIANA ISLANDS' 'KYRGYZSTAN' 'BERMUDA' 'BHUTAN' 'ZAMBIA'
 '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 2.4150e+01 9.1000e+0
0
 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
 3.6100e+01 2.6400e+01 2.0300e+01 6.4800e+01 2.8000e+01 5.8500e+00
 4.3900e+00 1.8850e+01 5.1450e+01 2.2050e+01 1.4790e+01 5.2500e+00
 8.6300e+00 2.2000e+01 1.7390e+01 1.0640e+01 7.1280e+01 5.9000e+01
 8.3600e+00 6.8800e+00 1.2600e+01 1.0250e+01 2.0640e+01 2.1350e+01
 4.2500e+00 6.1300e+00 1.5450e+02 4.9600e+01 3.2800e+00 2.2000e-01
 1.5560e+01 1.2750e+01 1.1750e+01 7.4000e-01 1.0920e+02 2.4800e+01
 7.3500e+00 3.6000e+01 7.6400e+00 2.4000e+01 5.2330e+01 1.6800e+01
 1.2380e+01 1.2500e+01 4.6960e+01 1.0150e+01 1.4937e+02 1.4160e+01
 3.9000e+00 1.8620e+01 1.1550e+01 1.7850e+01 4.3550e+01 5.2000e+00
 2.0850e+01 1.7380e+01 1.3500e+01 3.6100e+00 1.9500e+01 2.0000e+01
 1.4100e+01 3.4250e+01 2.9050e+01 2.1130e+01 2.4100e+00 6.9000e-01
 2.0380e+01 3.4100e+00 8.3160e+01 2.8500e+01 3.3950e+01 1.6750e+01
 1.3160e+01 1.2540e+01 4.1270e+01 8.9100e+01 1.0725e+02 1.6450e+01
 6.6600e+00 1.2130e+01 2.2250e+01 5.1000e-01 5.0600e+00 2.7400e+00
 3.7200e+01 5.7500e+00 1.7500e+01 2.7600e+00 3.8350e+01 4.8420e+01
 6.0100e+00 2.2230e+01 2.8690e+01 8.3250e+01 4.0000e+01 2.9750e+01
 3.8150e+01 3.2680e+01 2.7500e+01 6.5600e+00 1.3068e+02 1.0050e+01
 2.8100e+00 2.0150e+01 2.5000e-01 3.2300e+00 5.6000e-01 8.8100e+00
 2.8600e+01 9.5900e+00 2.0200e+00 9.5040e+01 4.3750e+01 4.4000e+00
 7.7220e+01 1.1540e+01 6.6300e+00 8.7700e+00 2.3600e+01 2.2130e+01
 6.0000e+01 1.1500e+01 4.2180e+01 4.9400e+01 1.6950e+01 1.3200e+01
 3.2100e+00 1.0900e+00 2.3750e+01 1.3630e+01 2.7500e+00 2.9500e+01
 4.1420e+01 7.2940e+01 3.7000e-01 2.0800e+02 1.3450e+02 6.3800e+00
 3.9000e+01 1.4130e+01 1.5400e+01 1.8000e+00 5.7400e+01 1.3662e+02
 2.6550e+01 3.5630e+01 4.2350e+01 1.1600e+00 3.3300e+00 7.1600e+00
 3.3600e+01 2.0960e+01 9.4000e-01 1.6000e-01 8.9000e+00 5.4190e+01
 8.5000e+00 5.9300e+00 7.1300e+00 2.0000e-01 1.8380e+01 5.0490e+01
 1.4300e+00 1.0098e+02 3.1530e+01 2.1850e+01 1.7230e+01 7.1250e+01
 2.8250e+01 7.5250e+01 1.5000e-01 1.6410e+01 7.4700e+00 1.2140e+01
 5.9880e+01 3.2830e+01 2.5680e+01 2.2360e+01 1.1380e+01 2.1021e+02
 1.8400e+00 8.5600e+00 1.6650e+02 2.4600e+00 8.8000e-01 1.1860e+01
 3.0250e+01 1.7250e+01 1.2630e+01 7.2500e+00 4.8000e+00 2.1600e+01
 3.1540e+01 1.5380e+01 3.4600e+00 1.0890e+01 1.4950e+02 2.8800e+00
 8.2200e+00 1.3299e+02 3.4130e+01 1.0130e+01 6.5160e+01 2.0300e+00
 1.4250e+01 3.2500e+01 2.9500e+00 4.0750e+01 7.9600e+00 4.0500e+00
 5.0000e-01 2.0800e+01 1.5100e+00 1.9140e+01 2.0650e+01 1.4975e+02
```

```
1.3310e+01 1.1800e+00 7.9100e+00 2.0060e+01 2.9200e+00 9.0000e-01
1.5880e+01 1.2050e+02 2.6000e+01 5.5860e+01 4.0950e+01 1.0822e+02
1.6090e+01 1.0380e+01 6.2500e+00 2.0000e+00 5.9100e+00 9.9000e-01
3.8020e+01 1.9600e+01 2.6000e-01 2.2100e+01 1.0800e+02 1.7550e+01
1.6350e+01 6.9710e+01 1.7630e+01 1.4880e+01 1.6900e+01 1.9070e+01
2.0000e-02 1.1200e+00 2.6750e+01 4.2800e+00 1.0800e+00 1.4750e+01
2.9000e+00 9.9900e+01 3.0200e+00 2.3450e+01 7.1830e+01 7.0200e+01
7.9500e+00 2.3500e+01 6.0040e+01 3.1500e+00 6.8000e-01 7.5000e+00
4.2000e+00 9.6000e-01 2.2910e+01 4.6250e+01 1.1210e+01 8.1200e+01
4.5000e-01 3.1690e+01 2.9130e+01 1.3000e+00 2.2040e+01 1.3700e+00
1.7540e+01 7.7600e+00 7.0250e+01 1.4850e+02 2.2000e+00 4.7200e+00
7.2800e+01 1.2940e+01 1.0255e+02 9.3600e+01 6.9400e+00 1.0692e+02
2.6276e+02 3.1000e+01 1.9250e+01 5.1750e+01 1.2474e+02 1.2400e+00
3.1380e+01 2.0995e+02 2.3000e-01 6.9600e+00 7.1850e+01 2.1450e+01
1.2450e+01 6.9300e+01 3.1000e+00 2.8850e+01 2.7250e+01 1.7820e+02
5.5500e+00 7.8900e+00 1.6300e+00 2.3730e+01 2.0280e+01 1.3950e+01
2.1500e+00 1.5190e+01 4.4840e+01 3.1880e+01 1.4630e+01 2.0250e+01
2.2750e+01 5.2600e+00 3.4380e+01 8.0500e+01 2.2400e+00 1.1060e+01
1.3490e+01 9.2500e+00 2.1700e+00 2.3630e+01 1.4550e+01 1.7050e+02
9.9700e+00 8.2500e+00 1.1880e+02 1.1286e+02 3.6400e+01 9.7250e+01
8.6000e-01 2.3180e+01 2.4380e+01 1.4460e+01 1.1900e+01 3.8000e+01
8.2100e+00 1.1425e+02 1.8240e+01 1.4380e+01 1.2288e+02 9.0000e+00
1.7130e+01 1.0100e+00 2.5130e+01 4.8590e+01 1.1000e+00 5.1300e+00
5.4000e-01 1.4800e+00 5.9000e-01 1.5440e+01 5.0000e-02 9.6000e+01
6.2650e+01 3.9250e+01 1.6160e+01 1.5930e+01 6.8400e+00 4.9400e+00
2.7000e-01 8.5130e+01 1.2875e+02 5.9150e+01 4.0200e+00 8.7600e+00
2.6400e+00 2.4000e-01 2.3400e+01 2.6500e+01 1.3406e+02 2.2200e+00
1.6600e+00 6.6250e+01 3.8000e+00 1.7500e+00 6.4700e+00 2.8760e+01
2.9600e+00 2.7190e+01 6.3200e+00 9.3000e+00 2.1600e+00 1.5400e+00
2.0880e+01 6.3000e-01 2.3060e+01 8.0500e+00 9.7000e-01 3.6800e+00
1.1800e+02 7.2000e+01 2.0130e+01 5.9800e+00 8.8500e+00 4.9100e+00
3.2000e-01 1.2300e+00 1.5280e+01 1.2500e+00 2.6260e+02 1.8671e+02
2.7300e+00 3.2200e+01 9.2600e+00 4.3250e+01 1.3000e-01 9.5000e-01
2.5620e+01 5.6000e+00 1.1720e+01 2.0600e+00 3.1750e+01 2.1630e+01
6.4050e+01 1.3200e+00 1.2560e+01 2.2960e+01 2.4500e+01 4.1800e+00
3.9330e+01 2.0700e+01 6.2000e+00 1.6000e+00 4.5400e+00 2.0480e+01
4.8000e+01 1.6800e+00 5.0700e+01 4.4690e+01 8.9380e+01 1.2300e+01
7.5000e-01 5.1200e+00 1.9630e+01 7.0000e-01 2.4500e+00 2.6980e+01
6.6000e-01 8.7000e+00 2.2470e+01 9.1900e+00 8.0000e+00 2.3290e+01
9.0000e-02 7.5600e+01 1.6500e+01 7.8280e+01 7.4100e+01 1.1700e+00
3.5200e+00 4.0300e+00 3.9400e+00 1.2775e+02 1.8800e+00 6.9900e+00
2.5840e+01 1.0200e+00 2.5550e+01 1.1130e+01 3.6000e+00 2.4450e+01
1.2900e+00 2.4860e+01 2.5200e+01 2.1500e+01 1.1110e+01 3.8500e+01
1.9350e+01 3.3900e+00 4.9650e+01 5.7130e+01 3.5000e+01 5.2150e+01
1.2200e+00 1.9340e+01 3.0100e+00 2.0750e+01 1.0300e+00 1.0630e+01
3.7700e+00 1.3690e+01 8.2360e+01 4.8630e+01 3.5500e+00 1.6640e+02
9.5500e+01 3.7130e+01 1.2100e+00 5.6250e+01 3.0550e+01 1.8560e+01
4.7880e+01 6.3350e+01 1.9010e+01 2.9000e-01 1.1340e+01 4.3800e+00
4.6800e+01 2.0690e+01 1.8730e+01 6.3750e+01 1.6038e+02 2.6200e+00
1.8800e+01 3.8250e+01 4.6400e+00 3.1400e+00 2.0440e+01 5.7900e+00
4.4200e+00 6.5000e-01 8.0000e+01 2.8950e+01 1.9700e+00 1.9130e+01
1.2070e+01 1.6400e+00 4.1000e-01 1.0600e+00 7.1000e-01 7.8000e+01
1.7300e+00 1.7226e+02 3.6560e+01 1.3300e+00 2.8800e+01 1.7290e+01
1.8130e+01 9.3800e+00 3.4500e+00 8.7500e+00 1.4500e+01 1.6900e+00
8.2600e+01 4.7290e+01 1.6570e+01 3.3130e+01 2.3800e+00 3.9200e+00
1.5960e+01 2.9570e+01 3.0000e+01 3.4000e-01 2.2300e+00 5.5480e+01
3.5590e+01 1.4000e-01 2.5880e+01 1.1231e+02 1.3920e+01 1.3570e+01
1.5900e+01 1.2600e+00 3.0500e+00 8.4900e+00 8.4130e+01 5.4500e+01
4.6400e+01 6.9250e+01 7.6900e+00 1.9900e+00 1.6632e+02 1.0220e+01
1.5300e+00 8.0000e-01 4.7000e-01 2.4750e+01 2.0980e+01 2.4090e+01
3.3200e+00 4.0400e+00 3.2500e+00 3.0710e+01 1.3210e+01 1.4256e+02
7.0200e+00 1.5500e+00 2.5200e+00 8.5200e+01 5.0250e+01 8.9600e+00
1.3880e+01 4.4500e+01 3.3800e+01 3.0400e+00 1.7200e+00 3.4750e+01
7.3450e+01 2.7460e+01 3.6860e+01 5.1980e+01 8.7800e+00 2.3250e+01
4.7400e+00 1.1200e+01 1.2675e+02 7.8000e-01 1.0000e+00 8.1130e+01
```

```
 7.2130e+01 3.6200e+00 5.0130e+01 5.7600e+01 5.7040e+01 6.9160e+01
 4.6000e+00 7.0500e+00 1.8200e+00 7.6700e+00 8.0600e+00 7.3250e+01
 1.0850e+01 2.0816e+02 8.5000e-01 1.1630e+01 4.4700e+00 7.3600e+01
 4.3470e+01 4.1300e+00 3.4500e+01 8.0440e+01 2.9440e+01 2.9380e+01
 4.9730e+01 9.4500e+00 3.0800e+00 5.3800e+00 4.0130e+01 2.5520e+01
 4.1380e+01 2.4300e+01 2.2500e+00 7.4260e+01 1.8900e+01 1.3400e+00
 3.3500e+00 1.8300e+00 1.2700e+00 1.3130e+01 7.9630e+01 2.4850e+01
 5.9500e+00 1.7160e+02 4.2000e-01 6.3380e+01 2.5080e+01 5.6000e+01
 2.8930e+01 4.6500e+00 2.1300e+00 6.0380e+01 1.4700e+00 6.5330e+01
 2.1000e-01 1.3000e+01 2.2400e+01 1.5015e+02 2.4200e+00 6.5000e+01
 2.8700e+00 1.6050e+01 5.0580e+01 3.7300e+00 4.9240e+01 1.1100e+01
 6.4550e+01 2.4440e+01 1.5444e+02 4.1130e+01 1.6100e+00 9.1300e+00
 1.9500e+00 1.0660e+01 4.4000e-01 2.5250e+01 7.2100e+00 5.1130e+01
 1.3750e+01 4.5900e+00 8.6800e+00 1.1580e+01 5.0500e+01 5.0400e+00
 4.3060e+01 1.9950e+01 3.6730e+01 3.7000e+01 1.2000e+00 1.5900e+00
 1.7400e+00 5.2650e+01 1.5700e+00 2.0500e+01 3.2660e+01 1.0320e+01
 6.3900e+00 2.9100e+00 6.1910e+01 1.0690e+01 6.4980e+01 6.5100e+00
 8.1000e-01 1.6610e+01 2.1750e+01 2.5020e+01 1.9000e+01 2.8180e+01
 6.5000e+00 2.1900e+00 9.0090e+01 3.2000e+01 4.9300e+00 9.8000e-01
 1.3020e+01 1.1000e-01 4.9000e-01 5.8000e-01 8.0300e+00 3.6600e+00
 6.5700e+00 1.4200e+00 8.4000e-01 1.0240e+02 5.5000e+00 4.5500e+00
 7.8700e+00 6.7250e+01 5.1200e+01 3.0900e+00 1.3475e+02 4.9900e+00
 5.4600e+01 1.7100e+00 1.6690e+01 4.3400e+01 2.3970e+01 1.6624e+02
 7.7130e+01 1.1083e+02 1.5570e+01 6.6700e+00 6.7750e+01 5.7300e+00
 1.6700e+01 5.8350e+01 7.2250e+01 1.1100e+00 9.3000e-01 3.4630e+01
 2.7000e+01 5.3490e+01 6.7630e+01 2.6590e+01 1.9000e+00 2.8280e+01
 4.5600e+01 1.1780e+01 5.9500e+01 2.3500e+00 1.9990e+01 2.6600e+00
 2.2310e+01 7.9000e-01 1.3510e+02 3.1050e+01 2.8350e+02 2.6630e+01
 8.9900e+00 7.3600e+00 5.7200e+00 8.9960e+01 8.9250e+01 1.2390e+01
 1.1900e+00 6.2000e+01 3.2900e+00 6.2400e+00 1.4530e+01 6.7000e-01
 2.8900e+00 1.5160e+01 1.9400e+00 4.3000e-01 4.1250e+01 1.2960e+01
 2.8000e+00 1.1300e+00 2.8300e+00 1.0113e+02 1.1830e+01 3.7250e+01
 2.5100e+00 3.5340e+01 2.1000e+01 1.8250e+01 5.4900e+01 8.7000e-01
 3.5380e+01 5.1400e+00 1.0330e+01 2.4700e+00 4.2100e+00 1.4180e+01
 1.1261e+02 5.2850e+01 2.6600e+01 2.0030e+01 3.8500e+00 2.4400e+01
 2.5480e+01 1.7000e+00 1.3860e+01 2.8350e+01 1.9050e+01 4.0000e-02
 1.0180e+01 1.1000e+01 1.6800e+02 1.1520e+02 8.3000e-01 2.9400e+01
 1.8655e+02 3.7800e+01 1.8040e+01 6.4200e+00 6.8200e+00 3.0500e+01
 5.6230e+01 2.3600e+00 7.3690e+01 1.0830e+01 1.6270e+01 2.3400e+00
 4.9500e+01 1.3900e+00 2.2730e+01 1.8414e+02 1.2170e+01 2.5500e+01
 3.8700e+00 4.4300e+00 1.0300e+02 3.0750e+01 3.8510e+01 1.3420e+01
 1.1295e+02 6.1000e-01 6.1700e+00 4.4060e+01 2.1770e+01 2.1700e+01
 3.0160e+01 7.8000e+00 1.0700e+00 3.9500e+00 7.5000e+01 2.3810e+01
 4.3000e+00 1.7600e+00 3.2300e+01 2.5000e+00 6.8500e+00 2.4170e+01
 4.5800e+00 2.7750e+01 2.7560e+01 3.5300e+00]

Age                  - [ 31  36  75  32  29  26  60  57  47  50  42  45  48  35  28  23
30  54
  37  38  58  39  22  46  53  33  27  63  69  34  43 118  40  76  56  71
  52  41  61  49  25  55  70  64  51  81   5  78  20  44  85  24  59  74
  67  66  72  21  65  84  68  17  83  73  82  18   0  79  62  19   7  86
  10  80  16  14   8  11   9  77  15  88   1  12  13  87   3   2]
```

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

	ID	Claim	Duration	Net Sales	Commision (in value)	Age
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

The below information from online states that a customer for Travel Insurance is regarded as a Senior citizen from the 71 years and above. And while some companies offer Travel Insurance up to a certain age, others do not have any restriction.

70 years

Ans: The maximum age limit is up to **70 years** for which a majority of insurers offer senior citizen travel insurance plans. Although, there are certain plans that provide offer senior citizen travel insurance for people up to 99 years of age. Mar 18, 2020

www.policybazaar.com › travel-insurance › senior-citizen...

Senior Citizen Travel Insurance | Compare, Buy or Renew ...

Is there an age limit on travel insurance? −

s per the plans offered by all insurance companies, the normal age of coverage is upto 70 years of age. However some of them have Senior Citizen plans without any age capping and other offering coverage between 80 and 99 years. The maximum age limit offered by the various insurance providers is tabled below for your immediate perusal.

Apollo Munich Health Insurance	Upto 80 years
Bajaj Allianz General Insurance	Upto 99 years
Cholamandalam Travel Insurance	Upto 80 years
Future Generali India Insurance Co Ltd	Upto 80 years
Religare Health Insurance Company Limited	No Max Age Restriction
Reliance General Insurance Co Ltd	No Max Age Restriction
Royal Sundaram General Insurance Co Ltd	No Max Age Restriction
Tata AIG General Insurance Co Ltd	No Max Age Restriction

Values above 100 years would most likely be outliers. However, we would need to see how to consider them (or change them) without effecting the data. One possible method is to 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
```

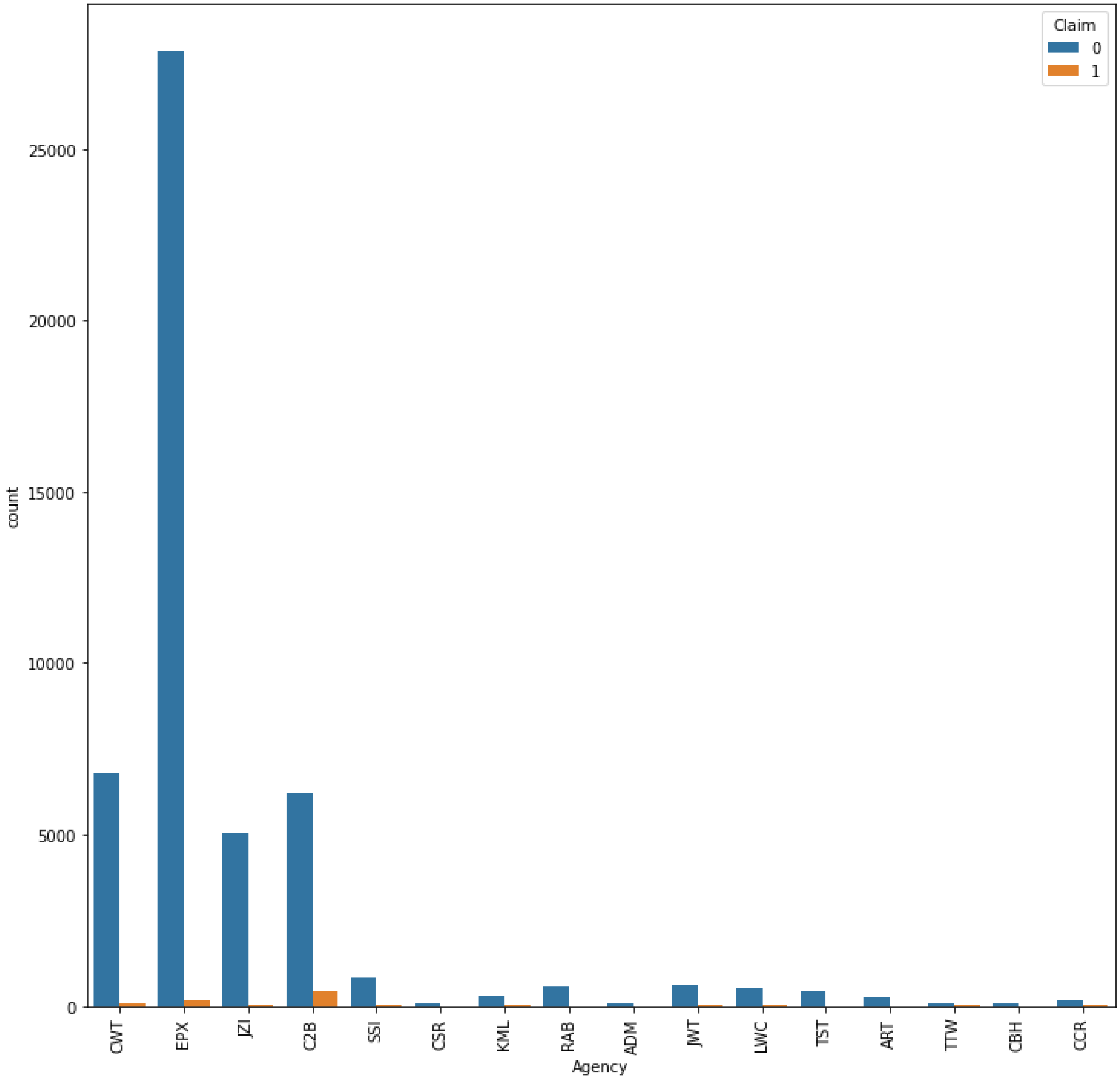
Apart from the target, "Claim", there are two more features that are bivariate - "Agency Type" and "Distribution Channel".
We could look to perform Hot Encoding on them.

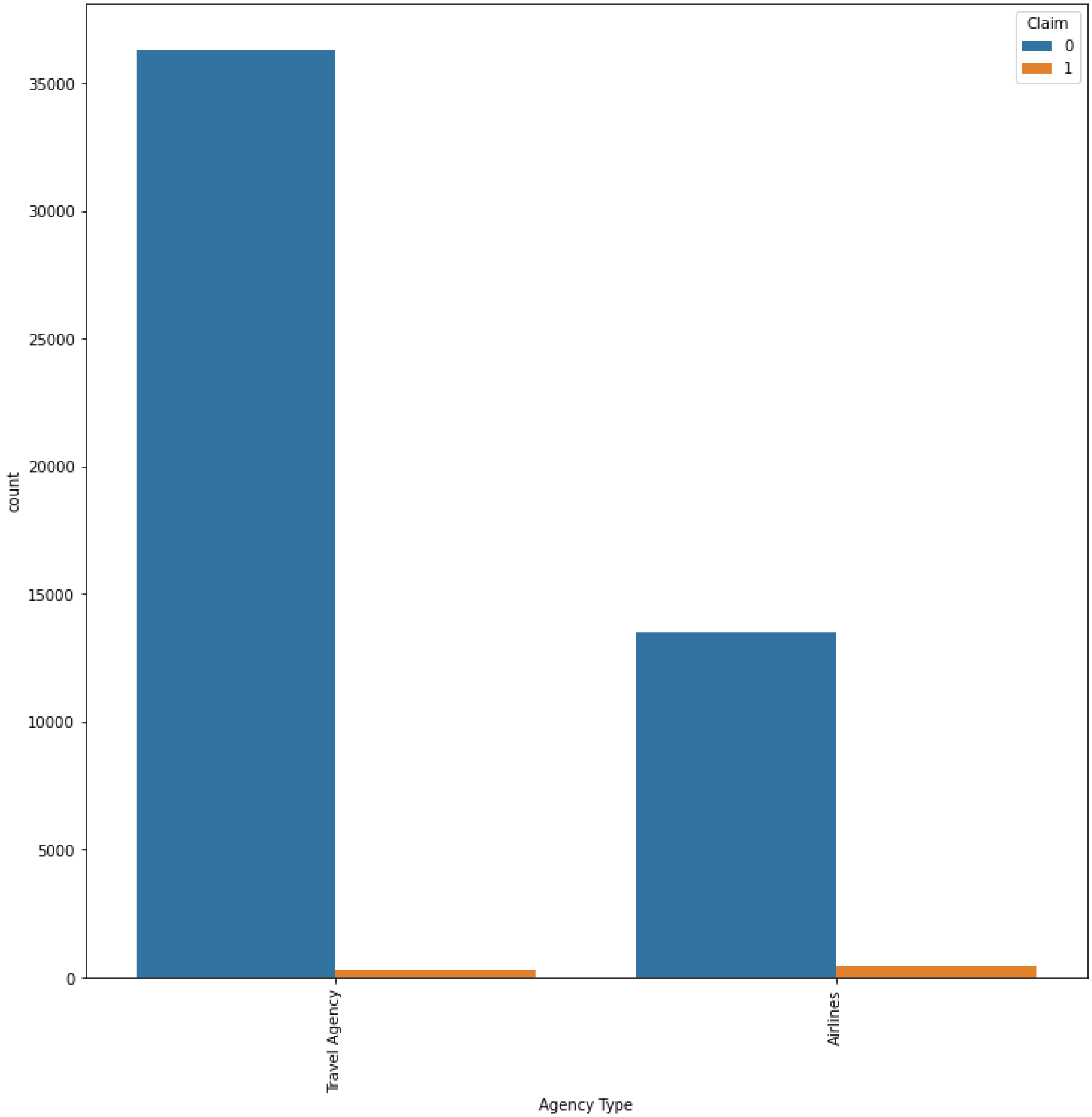
We will separate the Categorical and Numerical features, and explore them further.

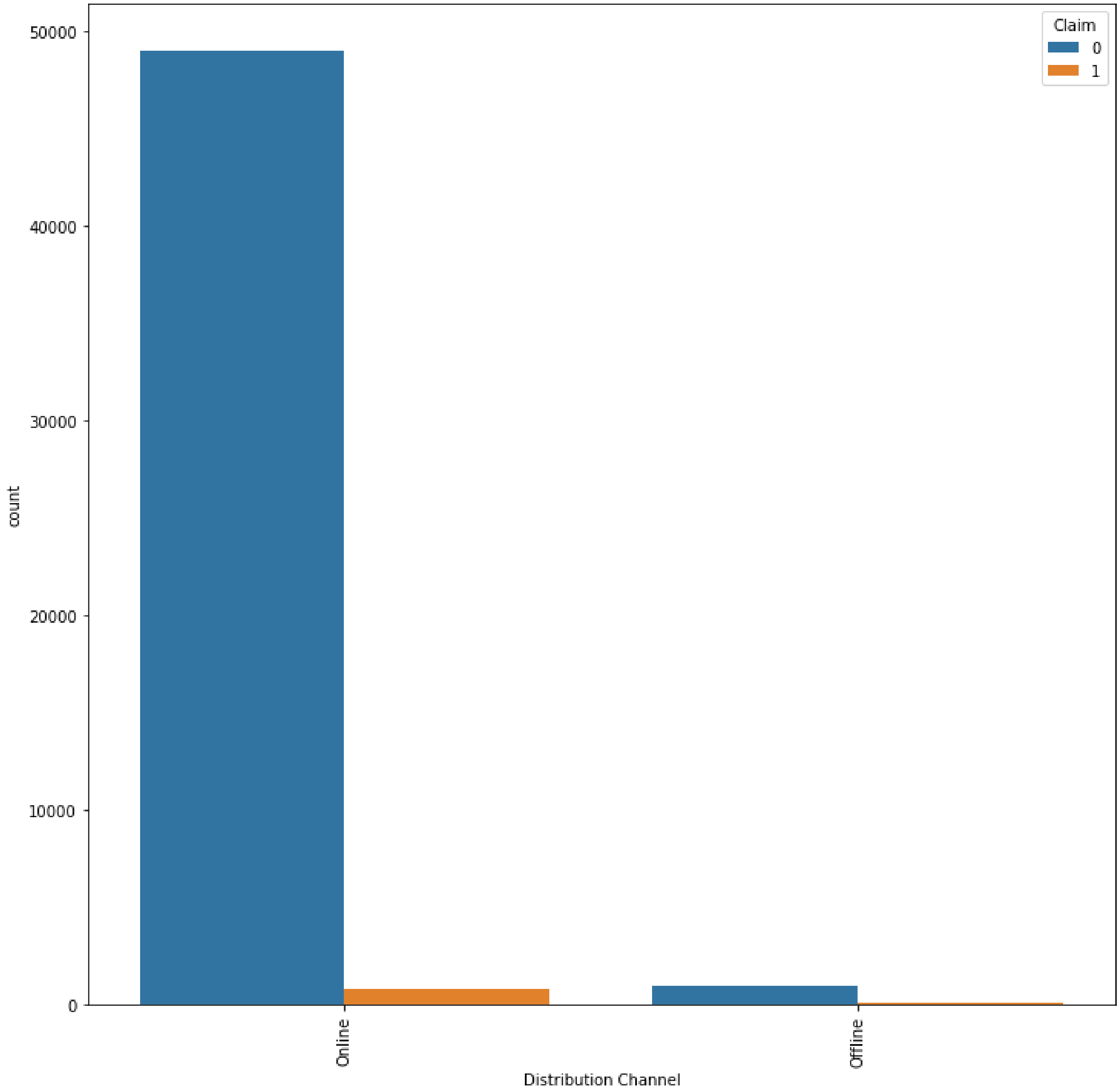
In [14]:
```
cat = ["Agency", "Agency Type", "Distribution Channel", "Product Name", "Destination"]
num = ["Duration", "Net Sales", "Commision (in value)", "Age"]
```

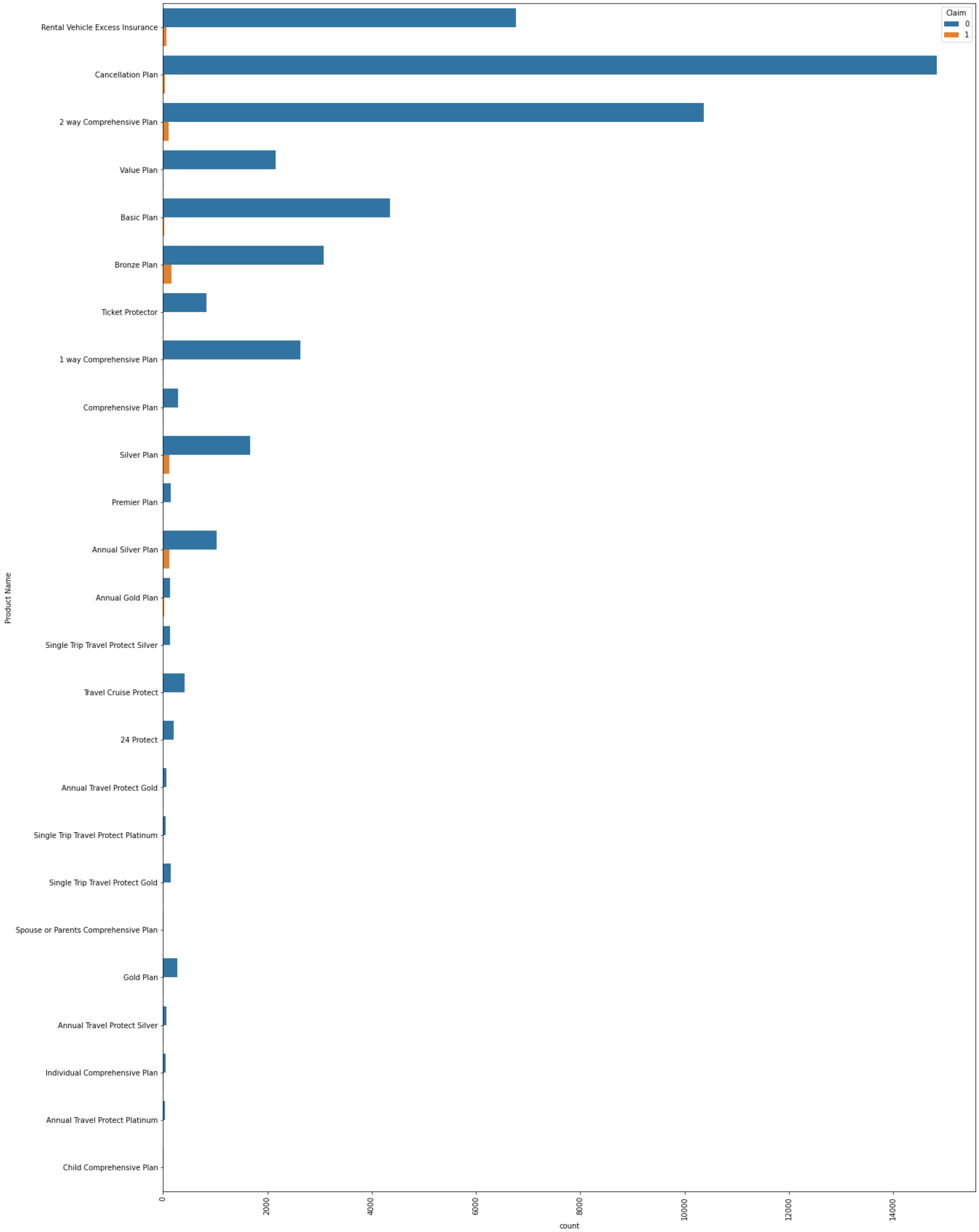
In [15]:
```
for cols in cat:
    if (cols == "Product Name") or (cols == "Destination"):
        plt.figure(figsize=(20,30))
        sns.countplot(data=df, hue=df["Claim"], y=cols)
    else:
```

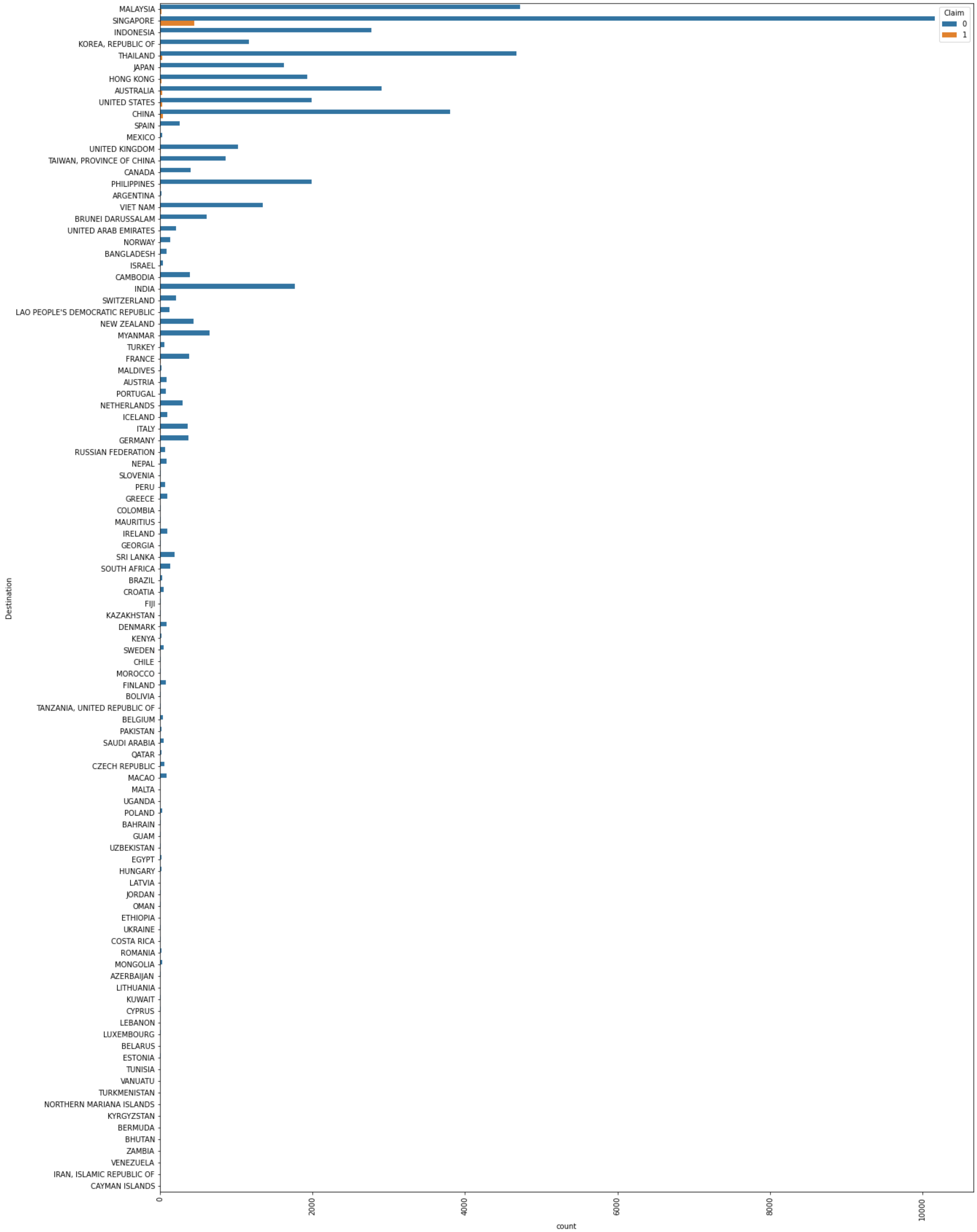
```
        plt.figure(figsize=(12,12))
        sns.countplot(data=df, hue=df["Claim"], x=cols)
    plt.xticks(rotation=90)
    plt.show()
```



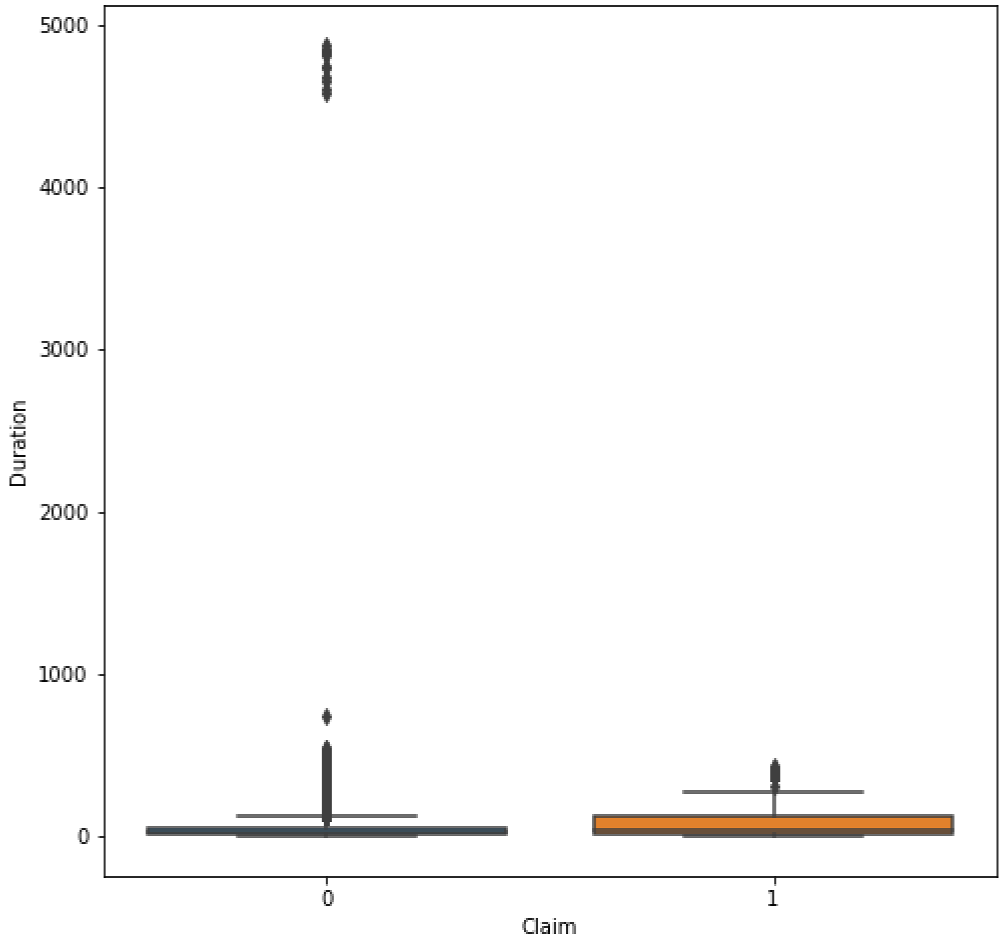

Claim
0
1
35000
30000
25000
20000
15000
10000
5000
0
count
Travel Agency
Airlines
Agency Type

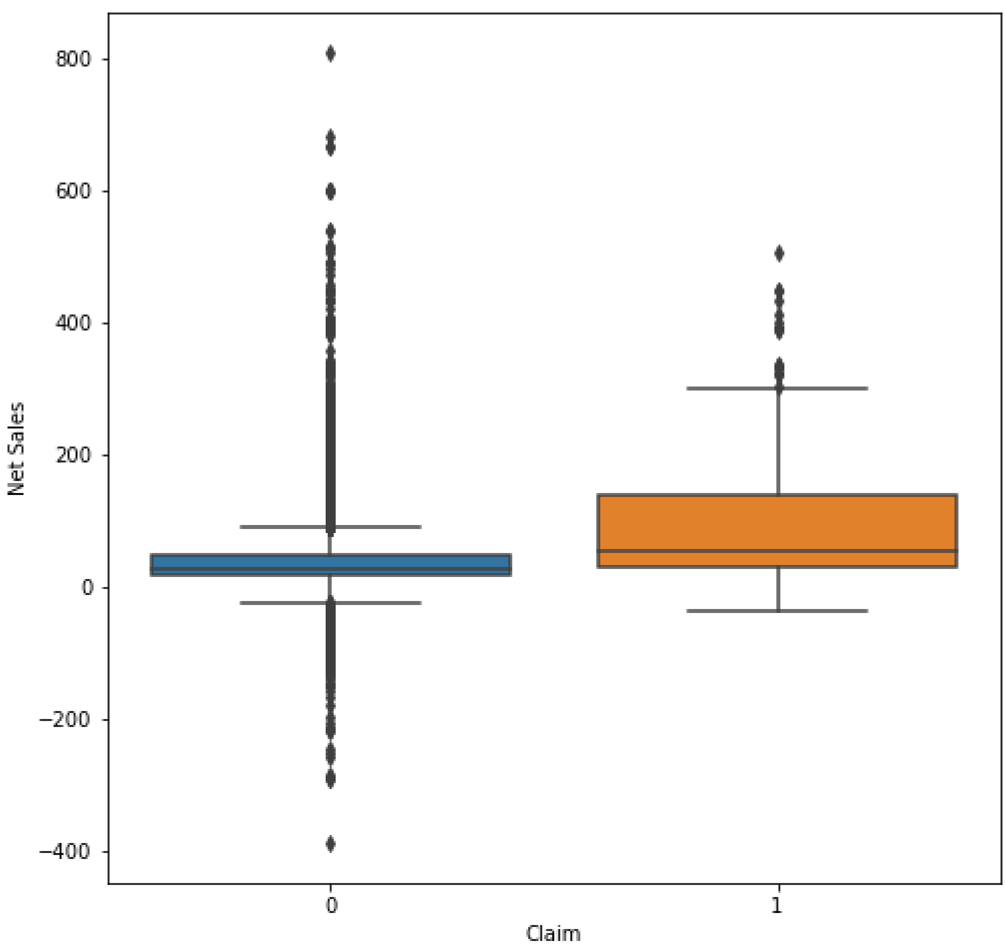

Claim
0
1
50000
40000
30000
20000
10000
0
count
Online
Offline
Distribution Channel

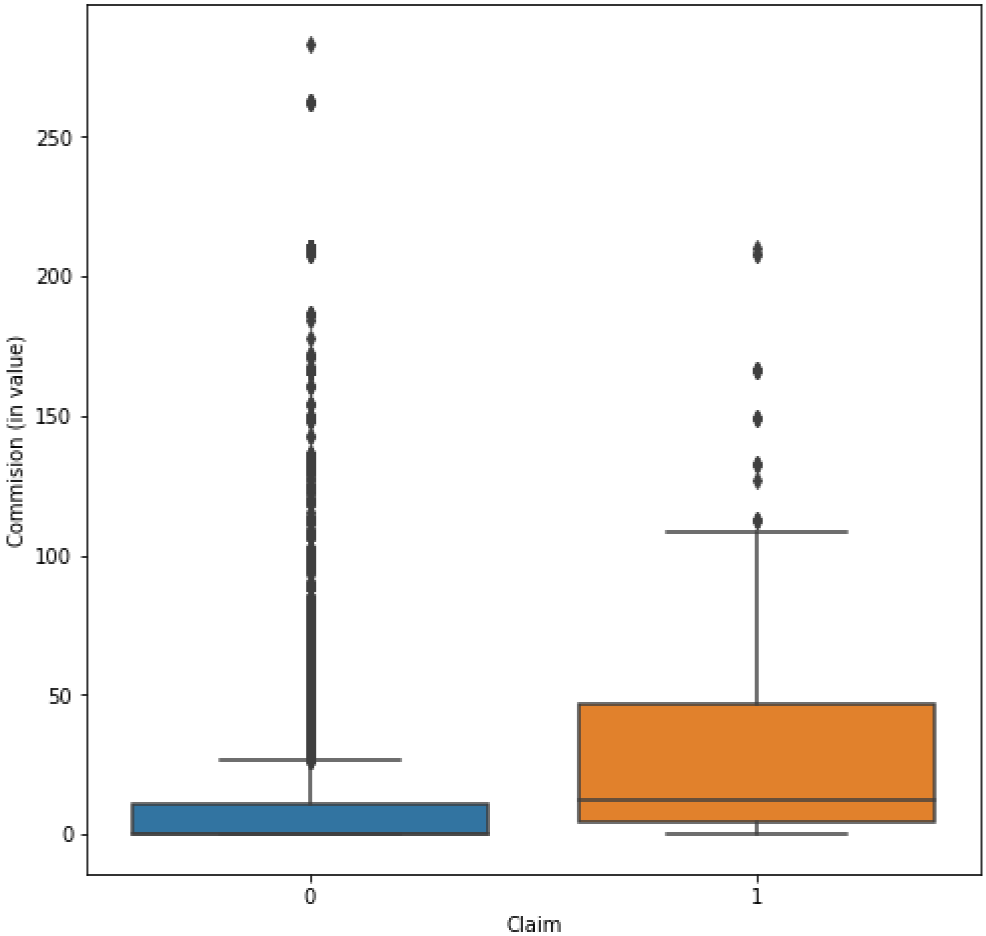

Claim
0
1
Rental Vehicle Excess Insurance
Cancellation Plan
2 way Comprehensive Plan
Value Plan
Basic Plan
Bronze Plan
Ticket Protector
1 way Comprehensive Plan
Comprehensive Plan
Silver Plan
Premier Plan
Annual Silver Plan
Annual Gold Plan
Single Trip Travel Protect Silver
Travel Cruise Protect
24 Protect
Annual Travel Protect Gold
Single Trip Travel Protect Platinum
Single Trip Travel Protect Gold
Spouse or Parents Comprehensive Plan
Gold Plan
Annual Travel Protect Silver
Individual Comprehensive Plan
Annual Travel Protect Platinum
Child Comprehensive Plan
Product Name
0
2000
4000
6000
8000
10000
12000
14000
count

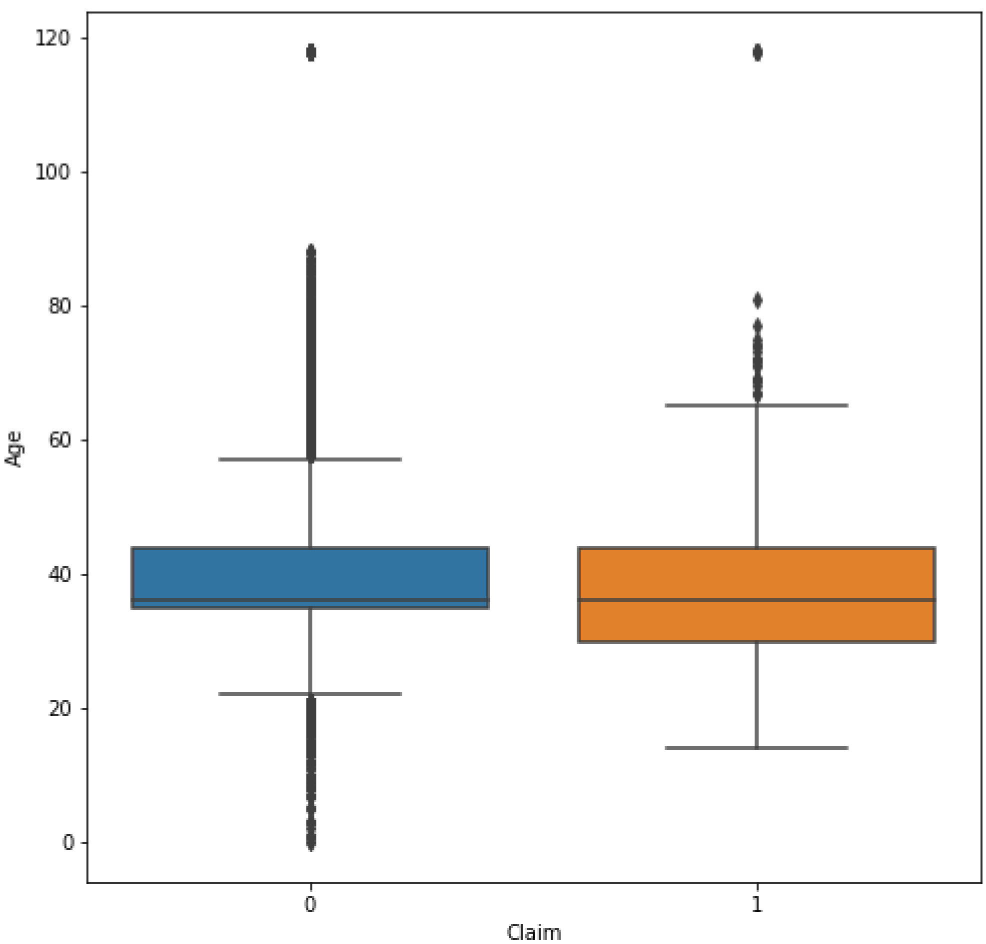


In [16]:
```python
for cols in num:
    plt.figure(figsize=(8,8))
    sns.boxplot(data=df, x="Claim", y=cols)
    plt.show()
```


5000
4000
3000
2000
1000
0
Duration
0
1
Claim


800
600
400
200
0
−200
−400
Net Sales
0
1
Claim


250
200
150
100
50
0
Commision (in value)
0
1
Claim


120
100
80
60
40
20
0
Age
0
1
Claim

We would need to manage only some of the outliers, and not all as it could lead to a lot of data loss. Apart from Age, another would be Duration. From the information below, we could replace all values in duration that are greater than 360, wtih 360.

Policy Duration: Cover trips from as short as 1 day to max of 360 days. Most of the Insurance Companies provides coverage for 180 days which can be extended for a further period of 180 days, provided there is no claim.

http://www.insurancepandit.com/travel/individual_travel_health_insurance.php

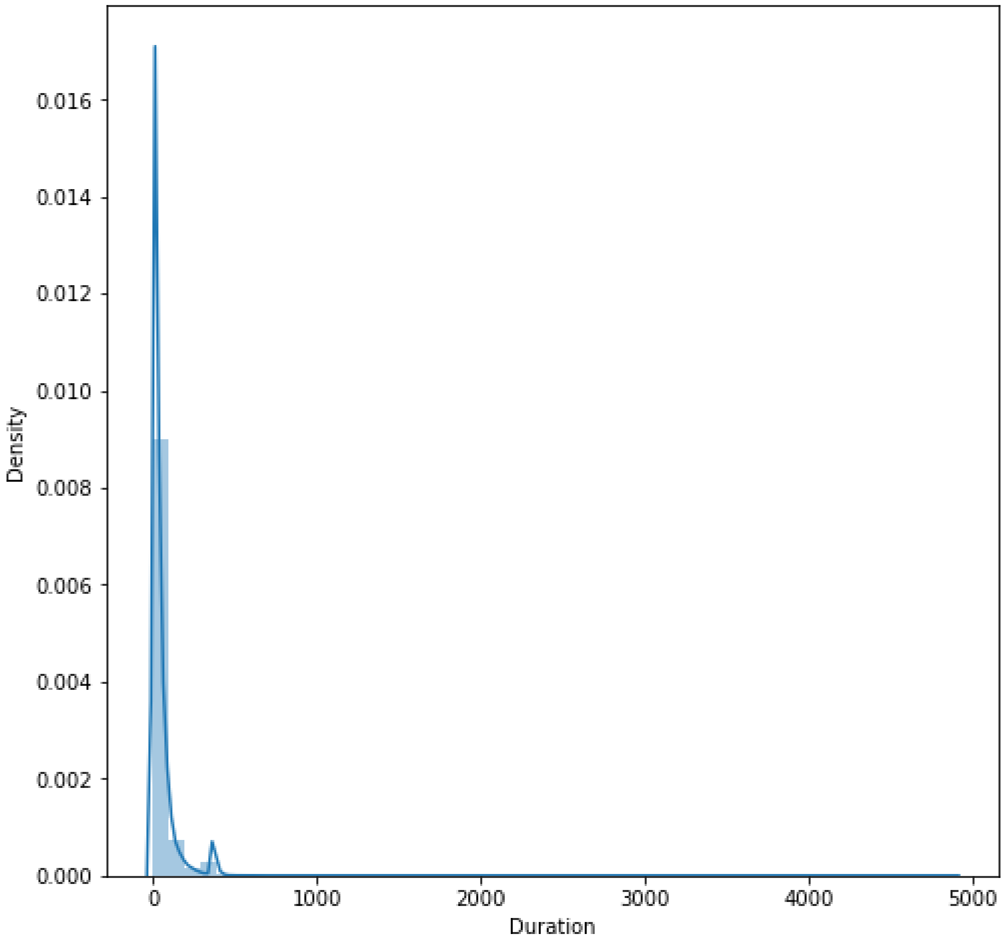
In [17]:
```
df.describe()
```

Out[17]:

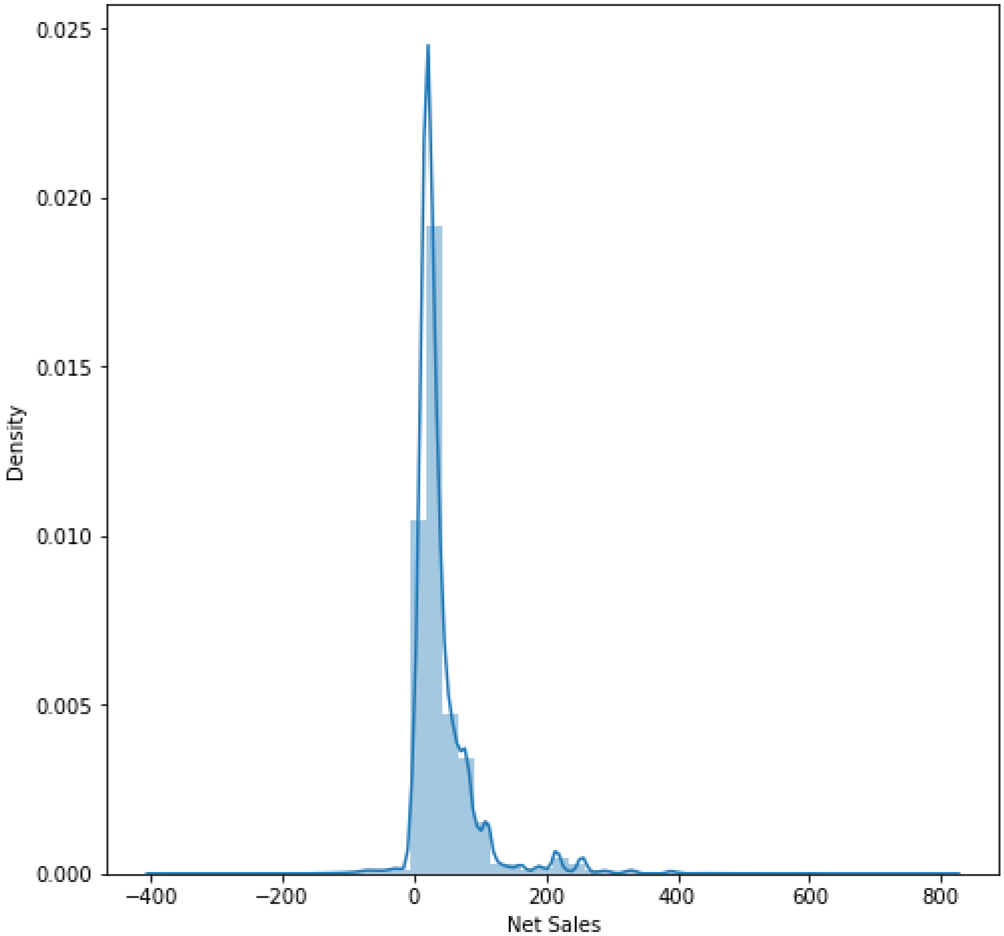
	Claim	Duration	Net Sales	Commision (in value)	Age
count	50553.000000	50553.000000	50553.000000	50553.00000	50553.000000
mean	0.014658	49.425969	40.800977	9.83809	40.011236
std	0.120180	101.434647	48.899683	19.91004	14.076566
min	0.000000	-2.000000	-389.000000	0.00000	0.000000
25%	0.000000	9.000000	18.000000	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()
```

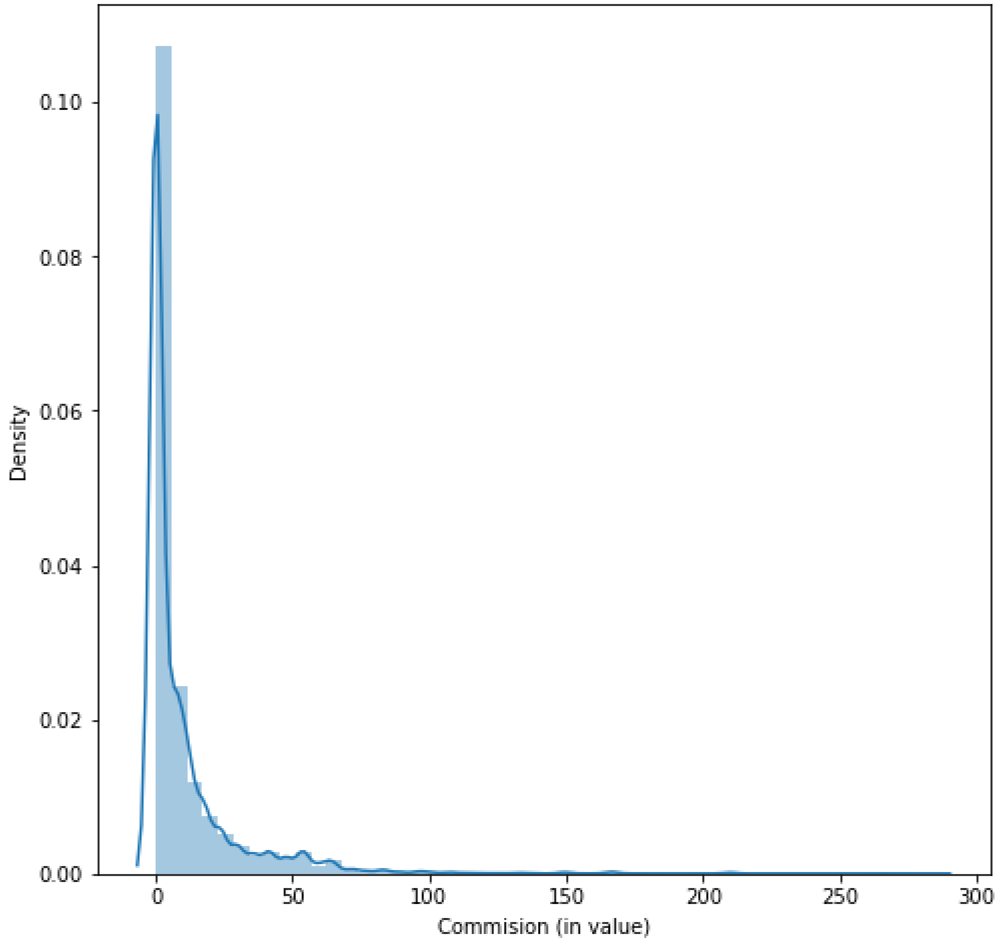
```
Duration                  : 22.872063891229274
```



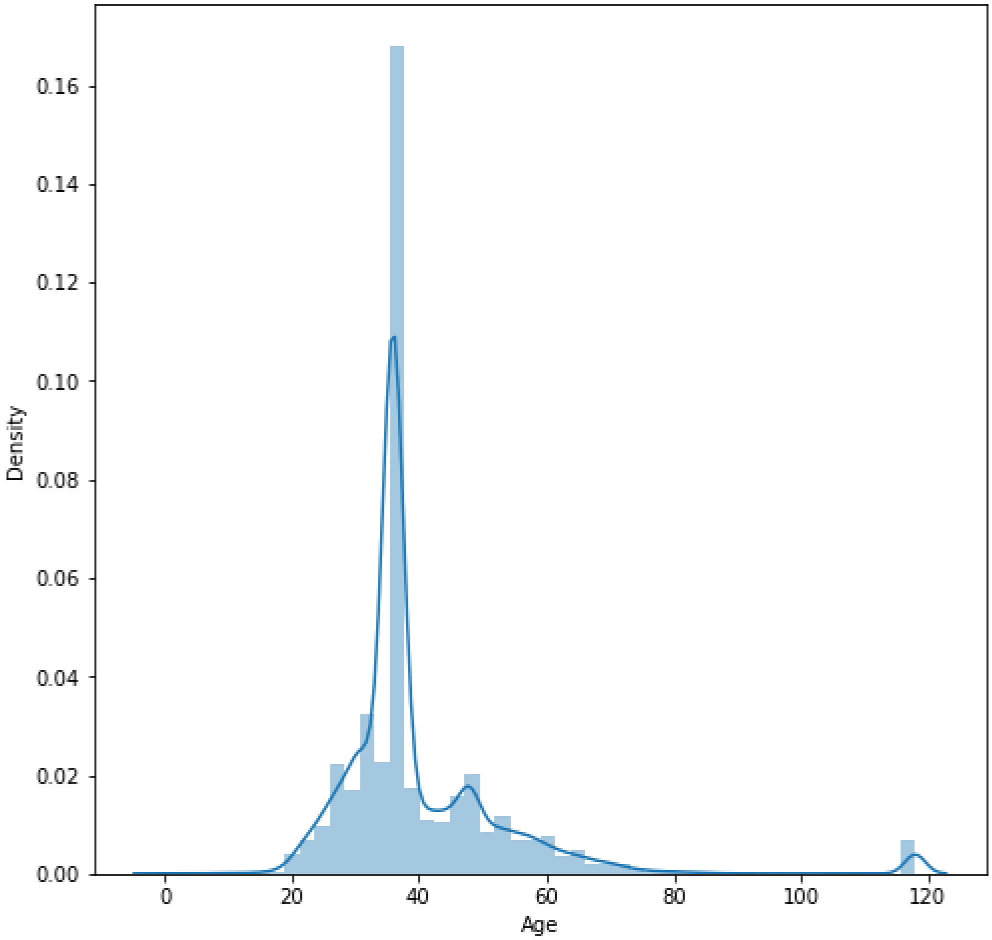

```
Net Sales                   : 3.3281441910342053
```




```
Commision (in value)      : 4.0780684356634636
```



Age : 2.9783898494112435



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

Out[21]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commis (in val
6	JZI	Airlines	Online	Value Plan	0	23	JAPAN	-69.0	24
128	EPX	Travel Agency	Online	Cancellation Plan	0	192	CANADA	-80.0	0
139	EPX	Travel Agency	Online	2 way Comprehensive Plan	0	55	CHINA	-77.0	0
173	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	198	NETHERLANDS	-9.9	5
336	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	109	AUSTRALIA	-19.8	11
...	...	...	...	...	...	...	...	...	
50121	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	81	JAPAN	-99.0	59

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commis (in val
50149	ART	Airlines	Online	24 Protect	0	2	MALAYSIA	-1.4	0
50177	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	75	UNITED STATES	-49.5	29
50394	JZI	Airlines	Online	Basic Plan	0	15	VIET NAM	-22.0	7
50399	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	135	AUSTRALIA	-49.5	29

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	Commision (in value)
6597	EPX	Travel Agency	Online	Cancellation Plan	1	28	SPAIN	-10.0	0.0
26666	JZI	Airlines	Online	Basic Plan	1	3	HONG KONG	-22.0	7.7
30272	EPX	Travel Agency	Online	2 way Comprehensive Plan	1	128	KOREA, REPUBLIC OF	-37.0	0.0

LabelEncoding, One Hot Encoding, Frequency Encoding

Label Encoding -> Each unique categorical value for a feature is replaced with a discrete number.

One Hot Encoding -> A separate column is created for each unique categorical value from a feature.

Frequency Encoding -> Finding out the frequency of each categorical unique value from a feature.

To get a better model, we would be running this file a few times as the model would have either one type of encoding, or a combination of all. Accordingly, the best models will be selected.
Once selected, if a particular encoding type lowered the scores, we will change the block code type to 'Raw'.

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)	Age
0	6	1	1	16	0	7	56	0.0	17.82	31
1	7	1	1	10	0	85	79	69.0	0.00	36
2	6	1	1	16	0	11	56	19.8	11.88	75
3	7	1	1	1	0	16	38	20.0	0.00	32
4	7	1	1	10	0	10	47	15.0	0.00	29
5	6	1	1	16	0	64	88	49.5	29.70	36
6	9	0	1	24	0	23	43	-69.0	24.15	26
7	9	0	1	8	0	31	34	26.0	9.10	60

 # 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"]
```

Fit and Predict

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

In [25]:

```
def model_sel(model, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

    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 [26]:

```
def models(X, y):
    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, y)))
    print("{} \n {}\n" .format("DECISION TREE", model_sel(dtc, X, y)))
    print("{} \n {}\n" .format("ADABOOST", model_sel(abc, X, y)))
    print("{} \n {}\n" .format("GRADIENT BOOST", model_sel(gbc, X, y)))
    print("{} \n {}\n" .format("XGBOOST", model_sel(xbc, X, y)))
    print("{} \n {}\n" .format("RANDOM FOREST", model_sel(rfc, X, y)))
    print("{} \n {}\n" .format("LINEAR SVM", model_sel(lsvc, X, y)))
    print("{} \n {}\n" .format("SVM", model_sel(svc, X, y)))

    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
Claims 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 [27]:

```
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 [28]:

```
def over_sample():
    X = df.drop("Claim", axis=1)
    y = df["Claim"]
    print(Counter(y))
    oversample = RandomOverSampler(sampling_strategy='minority')
```

```
    X_over, y_over = oversample.fit_resample(X, y)
    print(Counter(y_over))

    lr, abc, gbc, xbc, rfc, lsvc, svc = models(X_over, y_over)
    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 [29]:
```
def under_sample():
    X = df.drop("Claim", axis=1)
    y = df["Claim"]
    print(Counter(y))
    undersample = RandomUnderSampler(sampling_strategy='majority')
    X_under, y_under = undersample.fit_resample(X, y)
    print(Counter(y_under))

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

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 [30]:
```
def gridsearch(model, paramater, X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

    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 [31]:
```
# 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      0.98     14952
           1       0.05      0.06      0.05       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       0.11      0.01      0.02       213

    accuracy                           0.98     15165
   macro avg       0.55      0.51      0.51     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 [32]:
```
# With manual Under Sampling

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

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.70      0.87      0.78       227
           1       0.82      0.61      0.70       218

    accuracy                           0.74       445
   macro avg       0.76      0.74      0.74       445
weighted avg       0.76      0.74      0.74       445


DECISION TREE
              precision    recall  f1-score   support

           0       0.68      0.69      0.69       227
           1       0.67      0.66      0.67       218

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


ADABOOST
              precision    recall  f1-score   support

           0       0.74      0.77      0.75       227
           1       0.75      0.71      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


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.75      0.78      0.77       227
           1       0.77      0.73      0.75       218

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

```
XGBOOST
              precision    recall  f1-score   support

           0       0.72      0.74      0.73       227
           1       0.72      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


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.73      0.74      0.74       227
           1       0.73      0.71      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


LINEAR SVM
              precision    recall  f1-score   support

           0       0.75      0.18      0.29       227
           1       0.52      0.94      0.67       218

    accuracy                           0.55       445
   macro avg       0.64      0.56      0.48       445
weighted avg       0.64      0.55      0.48       445


SVM
              precision    recall  f1-score   support

           0       0.69      0.80      0.74       227
           1       0.75      0.62      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
```

In [33]:

```
# 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       0.71      0.84      0.77     15096
           1       0.79      0.65      0.71     14790

    accuracy                           0.74     29886
   macro avg       0.75      0.74      0.74     29886
weighted avg       0.75      0.74      0.74     29886


DECISION TREE
```

```
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15096
           1       0.98      1.00      0.99     14790

    accuracy                           0.99     29886
   macro avg       0.99      0.99      0.99     29886
weighted avg       0.99      0.99      0.99     29886


ADABOOST
              precision    recall  f1-score   support

           0       0.77      0.80      0.78     15096
           1       0.78      0.75      0.77     14790

    accuracy                           0.77     29886
   macro avg       0.78      0.77      0.77     29886
weighted avg       0.78      0.77      0.77     29886


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.78      0.81      0.80     15096
           1       0.80      0.76      0.78     14790

    accuracy                           0.79     29886
   macro avg       0.79      0.79      0.79     29886
weighted avg       0.79      0.79      0.79     29886


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.95      0.97     15096
           1       0.95      1.00      0.97     14790

    accuracy                           0.97     29886
   macro avg       0.97      0.97      0.97     29886
weighted avg       0.97      0.97      0.97     29886


RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     15096
           1       0.99      1.00      0.99     14790

    accuracy                           0.99     29886
   macro avg       0.99      0.99      0.99     29886
weighted avg       0.99      0.99      0.99     29886


LINEAR SVM
              precision    recall  f1-score   support

           0       0.69      0.12      0.21     15096
           1       0.51      0.95      0.67     14790

    accuracy                           0.53     29886
   macro avg       0.60      0.53      0.44     29886
weighted avg       0.60      0.53      0.43     29886
```

```
SVM
              precision    recall  f1-score   support

           0       0.74      0.80      0.77     15096
           1       0.78      0.71      0.74     14790

    accuracy                           0.76     29886
   macro avg       0.76      0.76      0.76     29886
weighted avg       0.76      0.76      0.76     29886
```

In [34]:

```python
# Under Sampled

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

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

           0       0.69      0.79      0.73       218
           1       0.76      0.65      0.70       227

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


DECISION TREE
              precision    recall  f1-score   support

           0       0.63      0.68      0.66       218
           1       0.67      0.62      0.65       227

    accuracy                           0.65       445
   macro avg       0.65      0.65      0.65       445
weighted avg       0.65      0.65      0.65       445


ADABOOST
              precision    recall  f1-score   support

           0       0.70      0.75      0.72       218
           1       0.74      0.69      0.71       227

    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       0.69      0.74      0.71       218
           1       0.73      0.68      0.70       227

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


XGBOOST
              precision    recall  f1-score   support
```

```
           0       0.68      0.72      0.70       218
           1       0.71      0.67      0.69       227

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


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.67      0.75      0.71       218
           1       0.73      0.64      0.68       227

    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       0.50      0.97      0.66       218
           1       0.70      0.06      0.11       227

    accuracy                           0.51       445
   macro avg       0.60      0.52      0.39       445
weighted avg       0.60      0.51      0.38       445


SVM
              precision    recall  f1-score   support

           0       0.61      0.77      0.68       218
           1       0.70      0.53      0.60       227

    accuracy                           0.64       445
   macro avg       0.66      0.65      0.64       445
weighted avg       0.66      0.64      0.64       445
```

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

As mentioned earlier, 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 [35]:
```
df["Age"][df["Age"] > 60] = mean_senior
```

In [36]:
```
df["Duration"][df["Duration"] > 360] = 360
```

In [37]:
```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [38]:
```
# lr_out, abc_out, gbc_out, xbc_out, rfc_out, lsvc_out, svc_out = models(X, y)
```

In [39]:
```
# Manual Under Sampling and Outliers

lr_out_sample, abc_out_sample, gbc_out_sample, xbc_out_sample, rfc_out_sample, lsvc_out
```

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.70      0.85      0.77       227
           1       0.80      0.63      0.70       218

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


DECISION TREE
              precision    recall  f1-score   support

           0       0.69      0.72      0.71       227
           1       0.70      0.66      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


ADABOOST
              precision    recall  f1-score   support

           0       0.76      0.82      0.79       227
           1       0.80      0.73      0.76       218

    accuracy                           0.78       445
   macro avg       0.78      0.77      0.77       445
weighted avg       0.78      0.78      0.77       445


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.74      0.81      0.77       227
           1       0.78      0.70      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


XGBOOST
              precision    recall  f1-score   support

           0       0.75      0.79      0.77       227
           1       0.77      0.72      0.75       218

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

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       0.74      0.82      0.78       227
           1       0.79      0.70      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


LINEAR SVM
              precision    recall  f1-score   support

           0       0.51      0.99      0.68       227
           1       0.67      0.03      0.05       218

    accuracy                           0.52       445
   macro avg       0.59      0.51      0.36       445
weighted avg       0.59      0.52      0.37       445


SVM
              precision    recall  f1-score   support

           0       0.69      0.80      0.74       227
           1       0.75      0.62      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
```

In [40]:

```
# Over Sampling and Outliers

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

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

           0       0.71      0.84      0.77     15096
           1       0.80      0.64      0.71     14790

    accuracy                           0.74     29886
   macro avg       0.75      0.74      0.74     29886
weighted avg       0.75      0.74      0.74     29886


DECISION TREE
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     15096
           1       0.97      1.00      0.98     14790

    accuracy                           0.98     29886
   macro avg       0.98      0.98      0.98     29886
weighted avg       0.98      0.98      0.98     29886


ADABOOST
              precision    recall  f1-score   support
```

```
           0       0.77      0.80      0.78     15096
           1       0.78      0.75      0.77     14790

    accuracy                           0.77     29886
   macro avg       0.78      0.77      0.77     29886
weighted avg       0.78      0.77      0.77     29886


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.78      0.81      0.80     15096
           1       0.80      0.77      0.78     14790

    accuracy                           0.79     29886
   macro avg       0.79      0.79      0.79     29886
weighted avg       0.79      0.79      0.79     29886


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.94      0.97     15096
           1       0.94      1.00      0.97     14790

    accuracy                           0.97     29886
   macro avg       0.97      0.97      0.97     29886
weighted avg       0.97      0.97      0.97     29886


RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15096
           1       0.98      1.00      0.99     14790

    accuracy                           0.99     29886
   macro avg       0.99      0.99      0.99     29886
weighted avg       0.99      0.99      0.99     29886


LINEAR SVM
              precision    recall  f1-score   support

           0       0.67      0.86      0.76     15096
           1       0.80      0.58      0.67     14790

    accuracy                           0.72     29886
   macro avg       0.74      0.72      0.71     29886
weighted avg       0.74      0.72      0.71     29886


SVM
              precision    recall  f1-score   support

           0       0.73      0.80      0.77     15096
           1       0.78      0.70      0.74     14790

    accuracy                           0.75     29886
   macro avg       0.76      0.75      0.75     29886
weighted avg       0.76      0.75      0.75     29886
```

In [41]:

```
# Under Sampling and Outliers
```

```
lr_out_under, abc_out_under, gbc_out_under, xbc_out_under, rfc_out_under, lsvc_out_unde
```

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

           0       0.68      0.81      0.74       218
           1       0.78      0.64      0.70       227

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


DECISION TREE
              precision    recall  f1-score   support

           0       0.65      0.66      0.66       218
           1       0.67      0.66      0.67       227

    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       0.68      0.76      0.72       218
           1       0.74      0.66      0.70       227

    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       0.69      0.78      0.73       218
           1       0.76      0.67      0.71       227

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


XGBOOST
              precision    recall  f1-score   support

           0       0.70      0.72      0.71       218
           1       0.72      0.70      0.71       227

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


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.69      0.74      0.72       218
           1       0.73      0.68      0.70       227
```

```
    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       0.82      0.34      0.48       218
           1       0.59      0.93      0.73       227

    accuracy                           0.64       445
   macro avg       0.71      0.63      0.60       445
weighted avg       0.71      0.64      0.61       445


SVM
              precision    recall  f1-score   support

           0       0.61      0.76      0.68       218
           1       0.70      0.54      0.61       227

    accuracy                           0.64       445
   macro avg       0.65      0.65      0.64       445
weighted avg       0.65      0.64      0.64       445
```

Skewness

In [42]:
```
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 [43]:
```
df["Duration"] = np.sqrt(df["Duration"])
df["Commision (in value)"] = np.sqrt(df["Commision (in value)"])
df["Age"] = np.sqrt(df["Age"])
```

In [44]:
```
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 [45]:
```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [46]:
```
# lr_skew, abc_skew, gbc_skew, xbc_skew, rfc_skew, lsvc_skew, svc_skew = models(X, y)
```

In [47]:
```
# Manual Under Sampling and Skewing

lr_skew_sample, abc_skew_sample, gbc_skew_sample, xbc_skew_sample, rfc_skew_sample, lsv
```

```
LOGISTIC REGRESSION
```

```
              precision    recall  f1-score   support

           0       0.70      0.85      0.77       227
           1       0.80      0.61      0.70       218

    accuracy                           0.74       445
   macro avg       0.75      0.73      0.73       445
weighted avg       0.75      0.74      0.73       445


DECISION TREE
              precision    recall  f1-score   support

           0       0.67      0.63      0.65       227
           1       0.64      0.67      0.66       218

    accuracy                           0.65       445
   macro avg       0.65      0.65      0.65       445
weighted avg       0.65      0.65      0.65       445


ADABOOST
              precision    recall  f1-score   support

           0       0.73      0.76      0.75       227
           1       0.74      0.71      0.72       218

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


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.72      0.80      0.76       227
           1       0.76      0.68      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


XGBOOST
              precision    recall  f1-score   support

           0       0.72      0.73      0.72       227
           1       0.71      0.70      0.71       218

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


RANDOM FOREST
              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
```

```
LINEAR SVM
              precision    recall  f1-score   support

           0       0.66      0.42      0.51       227
           1       0.56      0.78      0.65       218

    accuracy                           0.60       445
   macro avg       0.61      0.60      0.58       445
weighted avg       0.61      0.60      0.58       445


SVM
              precision    recall  f1-score   support

           0       0.68      0.78      0.73       227
           1       0.73      0.62      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
```

In [48]:

```
# Over Sampling and Skewing

lr_skew_over, abc_skew_over, gbc_skew_over, xbc_skew_over, rfc_skew_over, lsvc_skew_ove
```

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

           0       0.72      0.82      0.76     15096
           1       0.78      0.67      0.72     14790

    accuracy                           0.74     29886
   macro avg       0.75      0.74      0.74     29886
weighted avg       0.75      0.74      0.74     29886


DECISION TREE
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     15096
           1       0.97      1.00      0.98     14790

    accuracy                           0.98     29886
   macro avg       0.99      0.98      0.98     29886
weighted avg       0.99      0.98      0.98     29886


ADABOOST
              precision    recall  f1-score   support

           0       0.78      0.79      0.78     15096
           1       0.78      0.77      0.78     14790

    accuracy                           0.78     29886
   macro avg       0.78      0.78      0.78     29886
weighted avg       0.78      0.78      0.78     29886


GRADIENT BOOST
              precision    recall  f1-score   support
```

```
           0       0.78      0.81      0.80     15096
           1       0.80      0.77      0.78     14790

    accuracy                           0.79     29886
   macro avg       0.79      0.79      0.79     29886
weighted avg       0.79      0.79      0.79     29886


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.94      0.97     15096
           1       0.95      1.00      0.97     14790

    accuracy                           0.97     29886
   macro avg       0.97      0.97      0.97     29886
weighted avg       0.97      0.97      0.97     29886


RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15096
           1       0.98      1.00      0.99     14790

    accuracy                           0.99     29886
   macro avg       0.99      0.99      0.99     29886
weighted avg       0.99      0.99      0.99     29886


LINEAR SVM
              precision    recall  f1-score   support

           0       0.75      0.72      0.74     15096
           1       0.73      0.75      0.74     14790

    accuracy                           0.74     29886
   macro avg       0.74      0.74      0.74     29886
weighted avg       0.74      0.74      0.74     29886


SVM
              precision    recall  f1-score   support

           0       0.72      0.82      0.77     15096
           1       0.78      0.67      0.72     14790

    accuracy                           0.75     29886
   macro avg       0.75      0.75      0.75     29886
weighted avg       0.75      0.75      0.75     29886
```

In [49]:

```python
# Under Sampling and Skewing

lr_skew_under, abc_skew_under, gbc_skew_under, xbc_skew_under, rfc_skew_under, lsvc_ske
```

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

           0       0.70      0.83      0.76       218
           1       0.80      0.65      0.72       227
```

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


DECISION TREE
              precision    recall  f1-score   support

           0       0.63      0.72      0.67       218
           1       0.69      0.59      0.64       227

    accuracy                           0.65       445
   macro avg       0.66      0.66      0.65       445
weighted avg       0.66      0.65      0.65       445


ADABOOST
              precision    recall  f1-score   support

           0       0.71      0.79      0.75       218
           1       0.78      0.69      0.73       227

    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      0.80      0.76       218
           1       0.78      0.70      0.74       227

    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       0.71      0.78      0.74       218
           1       0.77      0.69      0.73       227

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


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.69      0.78      0.73       218
           1       0.76      0.67      0.71       227

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


LINEAR SVM
              precision    recall  f1-score   support

           0       0.59      0.85      0.70       218
           1       0.76      0.44      0.56       227
```

```
    accuracy                           0.64       445
   macro avg       0.68      0.65      0.63       445
weighted avg       0.68      0.64      0.63       445


SVM
              precision    recall  f1-score   support

           0       0.64      0.82      0.72       218
           1       0.76      0.56      0.65       227

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

Performing Chi-squared test

In [50]:
```
len(df.columns)
```

Out[50]:
```
10
```

In [51]:
```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [52]:
```
X_cols = []

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

In [53]:
```
def model_chi(model, X):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_sta

    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 [54]:
```
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 [55]:
```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [56]:
```
X_cols = []

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

In [57]:
```
# lr_chi, abc_chi, gbc_chi, xbc_chi, rfc_chi = model_new(X)
```

Scaling

In [58]:
```
df_old = df.copy(deep=True)
```

In [59]:
```
mm = MinMaxScaler()

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

In [60]:
```
df.head()
```

Out[60]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	
0	0.400000	1.0	1.0	0.666667	0	0.139443	0.554455	0.324437	0.250713	0.55
1	0.466667	1.0	1.0	0.416667	0	0.485913	0.782178	0.381985	0.000000	0.59
2	0.400000	1.0	1.0	0.666667	0	0.174801	0.554455	0.340951	0.204707	1.00
3	0.466667	1.0	1.0	0.041667	0	0.210819	0.376238	0.341118	0.000000	0.56
4	0.466667	1.0	1.0	0.416667	0	0.166667	0.465347	0.336947	0.000000	0.53

In [61]:
```
X = df.drop("Claim", axis=1)
y = df["Claim"]
```

In [62]:
```
# lr_scale, abc_scale, gbc_scale, xbc_scale, rfc_scale, lsvc_scale, svc_scale = models(
```

In [63]:
```
# Manual Under Sampling and Scalling
```

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

```
LOGISTIC REGRESSION
              precision    recall  f1-score   support

           0       0.69      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


DECISION TREE
              precision    recall  f1-score   support

           0       0.67      0.64      0.66       227
           1       0.64      0.68      0.66       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       0.73      0.75      0.74       227
           1       0.74      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


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.75      0.76      0.76       227
           1       0.75      0.74      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       0.72      0.70      0.71       227
           1       0.70      0.71      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


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.74      0.73      0.73       227
           1       0.72      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


LINEAR SVM
              precision    recall  f1-score   support

           0       0.70      0.82      0.76       227
           1       0.77      0.64      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


SVM
              precision    recall  f1-score   support

           0       0.70      0.83      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 [64]:

```
# Over Sampling and Scalling

lr_scale_over, abc_scale_over, gbc_scale_over, xbc_scale_over, rfc_scale_over, lsvc_sca
```

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

           0       0.72      0.83      0.77     15096
           1       0.79      0.66      0.72     14790

    accuracy                           0.75     29886
   macro avg       0.75      0.75      0.75     29886
weighted avg       0.75      0.75      0.75     29886


DECISION TREE
              precision    recall  f1-score   support

           0       1.00      0.97      0.98     15096
           1       0.97      1.00      0.98     14790

    accuracy                           0.98     29886
   macro avg       0.99      0.98      0.98     29886
weighted avg       0.99      0.98      0.98     29886


ADABOOST
              precision    recall  f1-score   support

           0       0.77      0.80      0.78     15096
           1       0.79      0.76      0.77     14790

    accuracy                           0.78     29886
   macro avg       0.78      0.78      0.78     29886
weighted avg       0.78      0.78      0.78     29886
```

```
GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.78      0.81      0.80     15096
           1       0.80      0.76      0.78     14790

    accuracy                           0.79     29886
   macro avg       0.79      0.79      0.79     29886
weighted avg       0.79      0.79      0.79     29886


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.94      0.97     15096
           1       0.95      1.00      0.97     14790

    accuracy                           0.97     29886
   macro avg       0.97      0.97      0.97     29886
weighted avg       0.97      0.97      0.97     29886


RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15096
           1       0.98      1.00      0.99     14790

    accuracy                           0.99     29886
   macro avg       0.99      0.99      0.99     29886
weighted avg       0.99      0.99      0.99     29886


LINEAR SVM
              precision    recall  f1-score   support

           0       0.71      0.83      0.77     15096
           1       0.79      0.65      0.72     14790

    accuracy                           0.74     29886
   macro avg       0.75      0.74      0.74     29886
weighted avg       0.75      0.74      0.74     29886


SVM
              precision    recall  f1-score   support

           0       0.73      0.84      0.78     15096
           1       0.81      0.68      0.74     14790

    accuracy                           0.76     29886
   macro avg       0.77      0.76      0.76     29886
weighted avg       0.77      0.76      0.76     29886
```

In [65]:

```python
# Under Sampling and Scalling

lr_scale_under, abc_scale_under, gbc_scale_under, xbc_scale_under, rfc_scale_under, lsv
```

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

```
           0       0.70      0.83      0.76       218
           1       0.81      0.66      0.72       227

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


DECISION TREE
              precision    recall  f1-score   support

           0       0.63      0.72      0.67       218
           1       0.69      0.60      0.64       227

    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       0.71      0.82      0.76       218
           1       0.79      0.67      0.73       227

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


GRADIENT BOOST
              precision    recall  f1-score   support

           0       0.70      0.79      0.74       218
           1       0.77      0.67      0.72       227

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


XGBOOST
              precision    recall  f1-score   support

           0       0.68      0.77      0.72       218
           1       0.75      0.65      0.70       227

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


RANDOM FOREST
              precision    recall  f1-score   support

           0       0.68      0.79      0.73       218
           1       0.76      0.64      0.69       227

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


LINEAR SVM
```

```
              precision    recall  f1-score   support

           0       0.71      0.83      0.77       218
           1       0.81      0.67      0.73       227

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


SVM
              precision    recall  f1-score   support

           0       0.70      0.87      0.78       218
           1       0.84      0.65      0.73       227

    accuracy                           0.76       445
   macro avg       0.77      0.76      0.75       445
weighted avg       0.77      0.76      0.75       445
```

Saving best model in a file through Pickle

In [67]:
```
file = open("TravelInsurance.ser", "wb")
pickle.dump(rfc_under, file)
file.close()
```

Below blocks of code are not in use anymore

dtc = DecisionTreeClassifier() parameter = {"criterion" : ("entropy", "gini"), "max_depth" : ([i for i in range(1,21)]), "min_samples_leaf": ([i for i in range(15,26)])} dtc_gscv = gridsearch(dtc, parameter, X, y)abc = AdaBoostClassifier() parameter = {"learning_rate" : (np.arange(0.1, 1.1, 0.1)), "n_estimators" : ([i for i in range(50,201,50)])} abc_gscv = gridsearch(abc, parameter, X, y)gbc = GradientBoostingClassifier() # parameter = {"learning_rate" : (np.arange(0.1, 1.1, 0.1)), "n_estimators" : ([i for i in range(50,201,50)]), "max_depth" : ([i for i in range(1,21)]), "min_samples_leaf": ([i for i in range(15,26)])} parameter = {"n_estimators" : ([46]), "max_depth" : ([6])} gbc_gscv = gridsearch(gbc, parameter, X, y)xgb = XGBClassifier() # parameter = {"n_estimators" : ([i for i in range(45,50)]), "max_depth" : ([i for i in range(3,8)])} parameter = {"n_estimators" : ([i for i in range(30,41)])} xgb_gscv = gridsearch(xgb, parameter, X, y)rfc = RandomForestClassifier() parameter = {"n_estimators" : ([i for i in range(45,50)]), "max_depth" : ([7])} rfc_gscv = gridsearch(rfc, parameter, X, y)lsvc = LinearSVC(random_state=1) parameter = {} lsvc_gscv = gridsearch(lsvc, parameter, X, y)svc = SVC(random_state=1) # parameter = {"C" : (np.arange(0.1, 1.1, 0.1))} parameter = {} svc_gscv = gridsearch(svc, parameter, X, y)

---

lr = LogisticRegression() abc = AdaBoostClassifier(n_estimators=100) gbc = GradientBoostingClassifier(n_estimators=100) xgb = XGBClassifier(n_estimators=200, reg_alpha=1) rfc = RandomForestClassifier()all_models = [("log_reg" ,lr), ("abc", abc), ("gbc", gbc), ("xgb", xgb), ("rfc", rfc)]X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) vc = VotingClassifier(estimators=all_models) vc.fit(X_train, y_train)y_pred_vc = vc.predict(X_test) print(classification_report(y_test, y_pred_vc))

Conclusion

The overall project went through changes from start till the end.

Version 1.0

Most time was spent on this version. Here is all that was done -

1) Read and analyzed dataset.

2) Removed 'Gender' as it had 71% null values.

3) Performed Label Encoding.

4) Created defintions for fitting and predicting models.

5) 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       0.06      0.07      0.06       213

    accuracy                           0.97     15165
   macro avg       0.52      0.53      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       0.06      0.00      0.01       213

    accuracy                           0.99     15165
   macro avg       0.52      0.50      0.50     15165
weighted avg       0.97      0.99      0.98     15165


RANDOM FOREST
              precision    recall  f1-score   support
```

```
           0       0.99      1.00      0.99     14952
           1       0.09      0.01      0.02       213

    accuracy                           0.98     15165
   macro avg       0.54      0.51      0.51     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 2.0

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.

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, XGBoost, and RandomForest.

```
DECISION TREE
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15096
           1       0.98      1.00      0.99     14790

    accuracy                           0.99     29886
   macro avg       0.99      0.99      0.99     29886
weighted avg       0.99      0.99      0.99     29886


XGBOOST
              precision    recall  f1-score   support

           0       1.00      0.94      0.97     15096
           1       0.95      1.00      0.97     14790

    accuracy                           0.97     29886
   macro avg       0.97      0.97      0.97     29886
weighted avg       0.97      0.97      0.97     29886
```

```
RANDOM FOREST
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     15096
           1       0.99      1.00      0.99     14790

    accuracy                           0.99     29886
   macro avg       0.99      0.99      0.99     29886
weighted avg       0.99      0.99      0.99     29886
```

Overall, RandomForest produced the best results. Even after some EDA and Preprocessing, the score for RandomForest we all the same as the above Baseline RandomForest model. For this, we saved the model 'rfc_under' into a serial file through Pickle.