Import required packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

Read the data

```
df=pd.read_csv("/content/drive/MyDrive/Colab Notebooks/travel insurance claim.csv")
```

Performing EDA

```
df.shape
```

```
(50553, 12)
```

```
df.head()
```

```
      ID Agency    Agency Type  ... Commision (in value) Gender  Age
0   3433    CWT  Travel Agency  ...                17.82    NaN   31
1   4339    EPX  Travel Agency  ...                 0.00    NaN   36
2  34590    CWT  Travel Agency  ...                11.88    NaN   75
3  55816    EPX  Travel Agency  ...                 0.00    NaN   32
4  13816    EPX  Travel Agency  ...                 0.00    NaN   29

[5 rows x 12 columns]
```

```
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
```

Here as we can see that Gender Column Having more then 70% of NaN value So We Drop that Gender Column

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

```
df.describe()
```

```
              Claim      Duration  ...  Commision (in value)
Age
count  50553.000000  50553.000000  ...           50553.00000
50553.000000
mean       0.014658     49.425969  ...               9.83809
40.011236
std        0.120180    101.434647  ...              19.91004
14.076566
min        0.000000     -2.000000  ...               0.00000
0.000000
25%        0.000000      9.000000  ...               0.00000
35.000000
50%        0.000000     22.000000  ...               0.00000
36.000000
75%        0.000000     53.000000  ...              11.55000
44.000000
max        1.000000   4881.000000  ...             283.50000
118.000000

[8 rows x 5 columns]
```

```
df[df["Duration"] <0]
```

```
      Agency Agency Type  ... Commision (in value)  Age
4063     JZI    Airlines  ...                  6.3  118
38935    JZI    Airlines  ...                  6.3  118
48367    JZI    Airlines  ...                  7.7  118

[3 rows x 10 columns]
```

```
df[df["Age"] > 100]
```

```
      Agency    Agency Type  ... Commision (in value)  Age
90       JWT       Airlines  ...                31.20  118
108      JWT       Airlines  ...                12.40  118
140      JWT       Airlines  ...                15.60  118
153      JWT       Airlines  ...                31.20  118
181      JWT       Airlines  ...                12.40  118
...      ...            ...  ...                  ...  ...
50158    JWT       Airlines  ...                24.00  118
50179    JWT       Airlines  ...                12.40  118
```

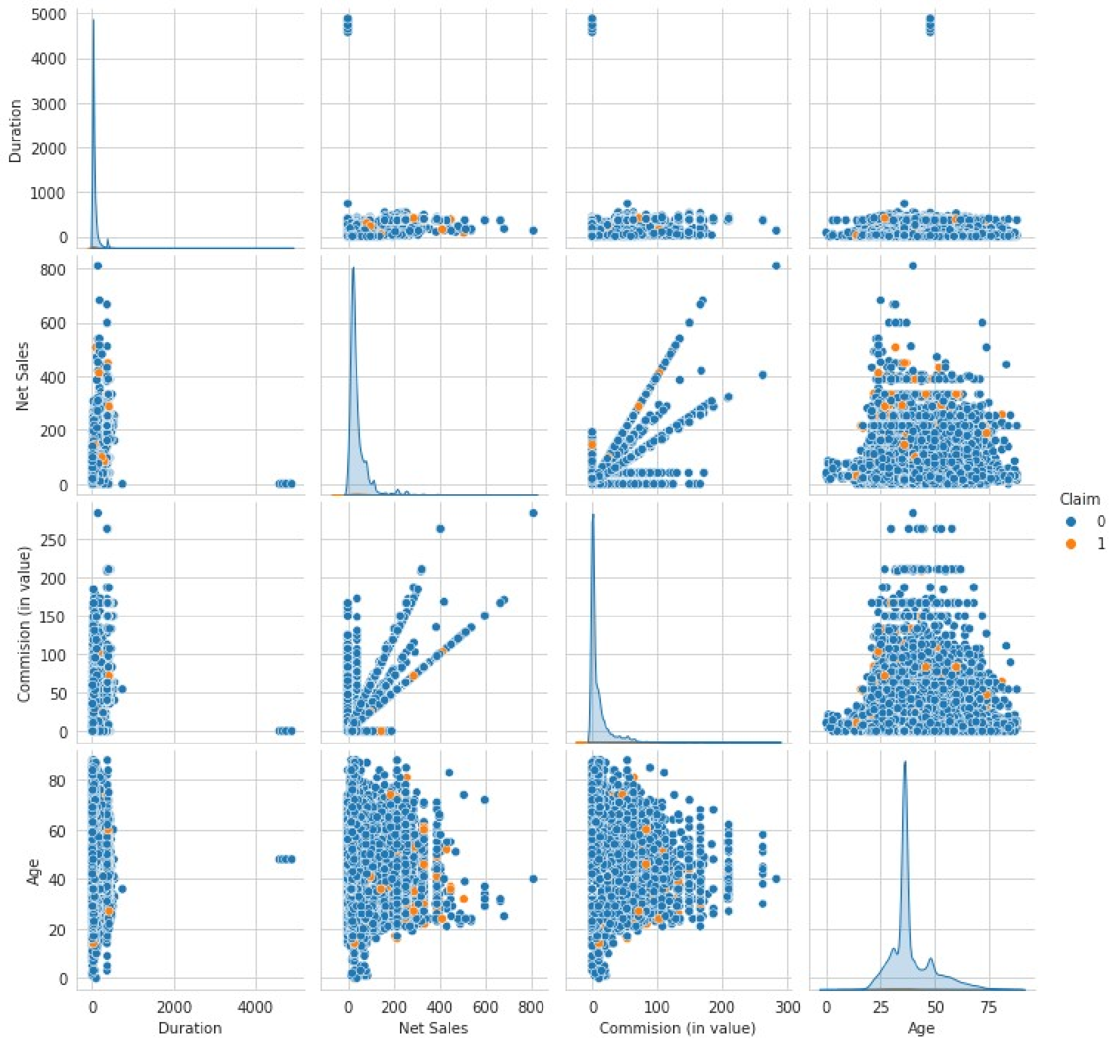
```
50250    JWT       Airlines  ...                 12.40  118
50429    JZI       Airlines  ...                 12.25  118
50478    CCR  Travel Agency  ...                  9.57  118

[795 rows x 10 columns]
```

```
df[df["Net Sales"] <0]
```

```
      Agency    Agency Type  ... Commision (in value)  Age
6        JZI       Airlines  ...                24.15   26
128      EPX  Travel Agency  ...                 0.00   37
139      EPX  Travel Agency  ...                 0.00   46
173      CWT  Travel Agency  ...                 5.94   31
336      CWT  Travel Agency  ...                11.88   31
...      ...            ...  ...                  ...  ...
50121    CWT  Travel Agency  ...                59.40   45
50149    ART       Airlines  ...                 0.49  118
50177    CWT  Travel Agency  ...                29.70   49
50394    JZI       Airlines  ...                 7.70   57
50399    CWT  Travel Agency  ...                29.70   31

[528 rows x 10 columns]
```

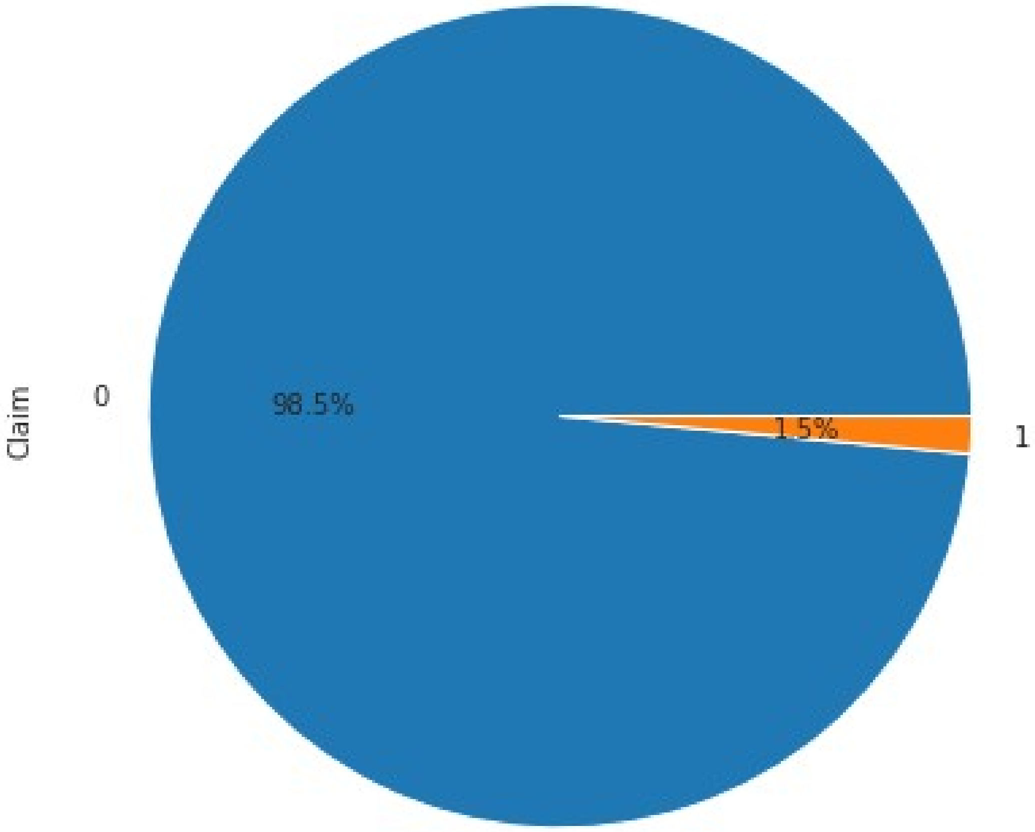
```
df.loc[df['Duration'] < 0, 'Duration'] = 49.425969
df.loc[df['Age'] > 100, 'Age'] = 40.011236
df.loc[df['Net Sales'] < 0, 'Net Sales'] = 40.800977
```

```
sns.set_style("whitegrid");
sns.pairplot(df, hue="Claim");
plt.show()
```



```
print(df["Claim"].value_counts())
print("---------------------------------------------------")
plt.figure(figsize=(7,7))
df["Claim"].value_counts().plot.pie(autopct="%.1f%%")
plt.show()
```

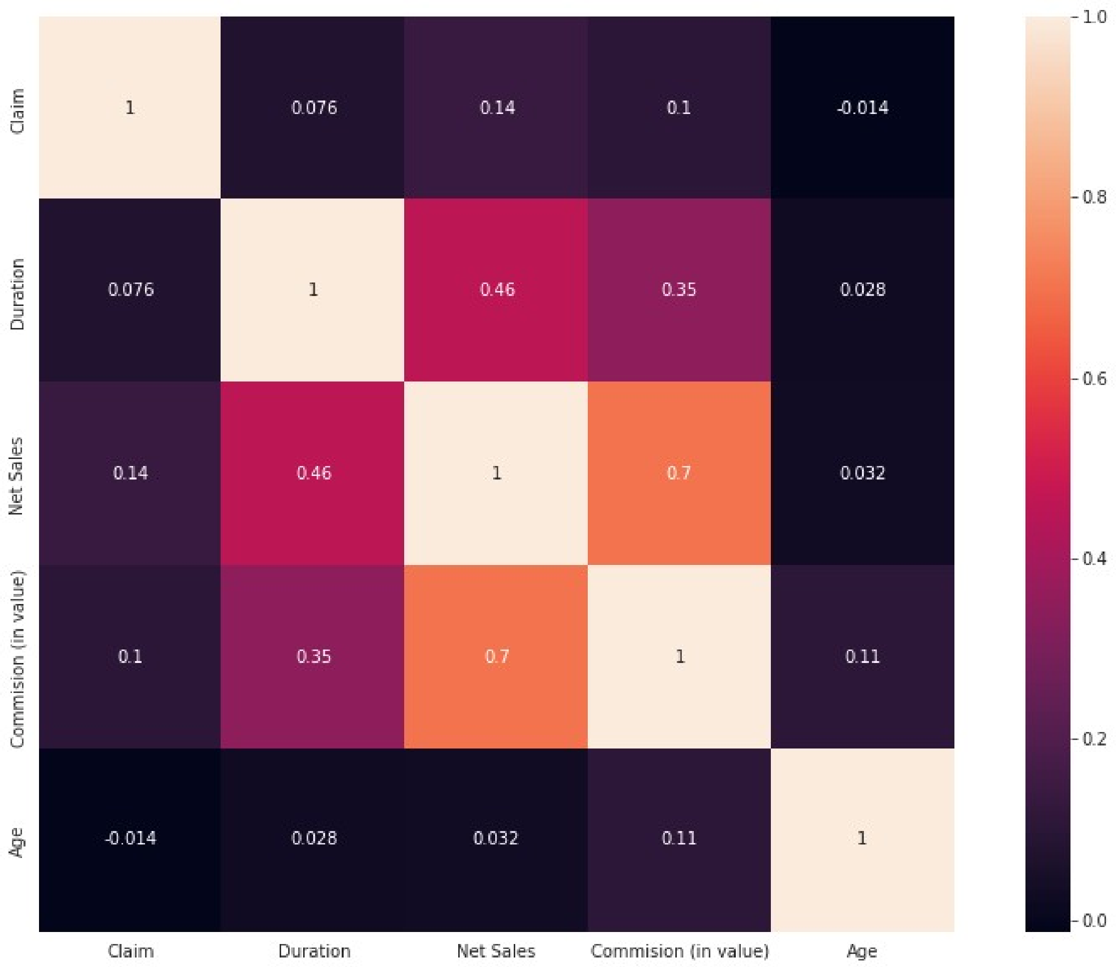
```
0    49812
1      741
Name: Claim, dtype: int64
---------------------------------------------------
```



Looking For Correlations

```
 ax = plt.subplots(figsize=(15, 10))
sns.heatmap(df.corr(), square=True,annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa3f8ccfd0>
```



Commision (in value) column having High Correlation between Feature 'Net Sales' and very Low Correlation with the 'Target' so we are Droping The Commision (in value) Column

```
df=df.drop(['Commision (in value)'],axis=1)
```

```
df.head()
```

```
  Agency    Agency Type  ... Net Sales   Age
0    CWT  Travel Agency  ...       0.0  31.0
1    EPX  Travel Agency  ...      69.0  36.0
2    CWT  Travel Agency  ...      19.8  75.0
3    EPX  Travel Agency  ...      20.0  32.0
4    EPX  Travel Agency  ...      15.0  29.0

[5 rows x 9 columns]
```

Separating categorical and numerical data

```
df_num = df.select_dtypes(["int64","float64"])
df_cat = df.select_dtypes("object")
```

For df_num

```
df_num.head()
```

```
   Claim  Duration  Net Sales   Age
0      0       7.0        0.0  31.0
1      0      85.0       69.0  36.0
2      0      11.0       19.8  75.0
3      0      16.0       20.0  32.0
4      0      10.0       15.0  29.0
```

```
df_cat.head()
```

```
  Agency    Agency Type  ...                     Product Name
Destination
0    CWT  Travel Agency  ...  Rental Vehicle Excess Insurance
MALAYSIA
1    EPX  Travel Agency  ...                Cancellation Plan
SINGAPORE
2    CWT  Travel Agency  ...  Rental Vehicle Excess Insurance
MALAYSIA
3    EPX  Travel Agency  ...         2 way Comprehensive Plan
INDONESIA
4    EPX  Travel Agency  ...                Cancellation Plan  KOREA,
REPUBLIC OF

[5 rows x 5 columns]
```

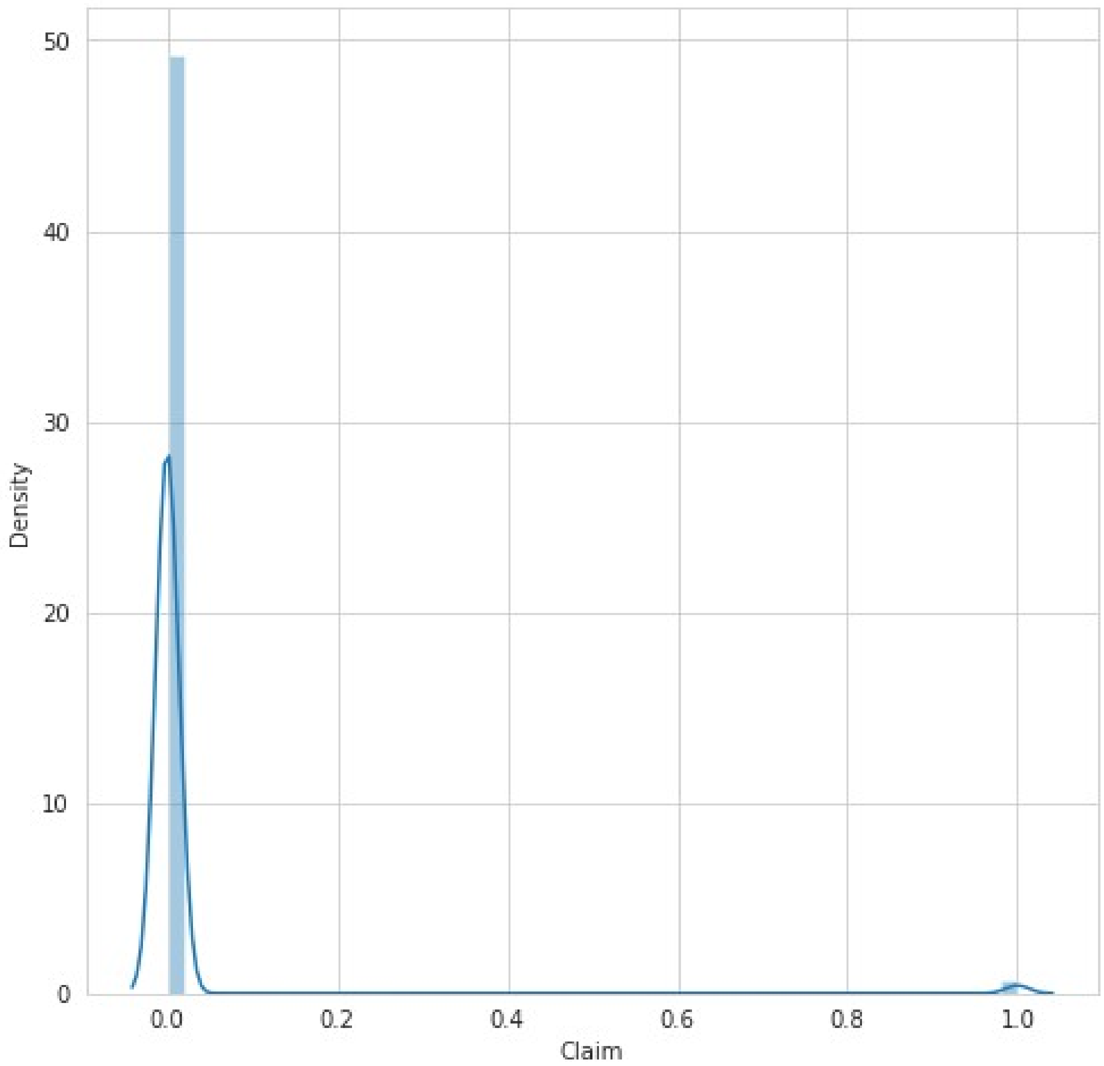
```
df_num.describe()
```

```
              Claim      Duration     Net Sales           Age
count  50553.000000  50553.000000  50553.000000  50553.000000
mean       0.014658     49.428981     41.852794     38.784779
std        0.120180    101.433893     47.536249     10.049564
min        0.000000      0.000000      0.000000      0.000000
25%        0.000000      9.000000     18.500000     35.000000
50%        0.000000     22.000000     27.000000     36.000000
75%        0.000000     53.000000     48.000000     42.000000
max        1.000000   4881.000000    810.000000     88.000000
```

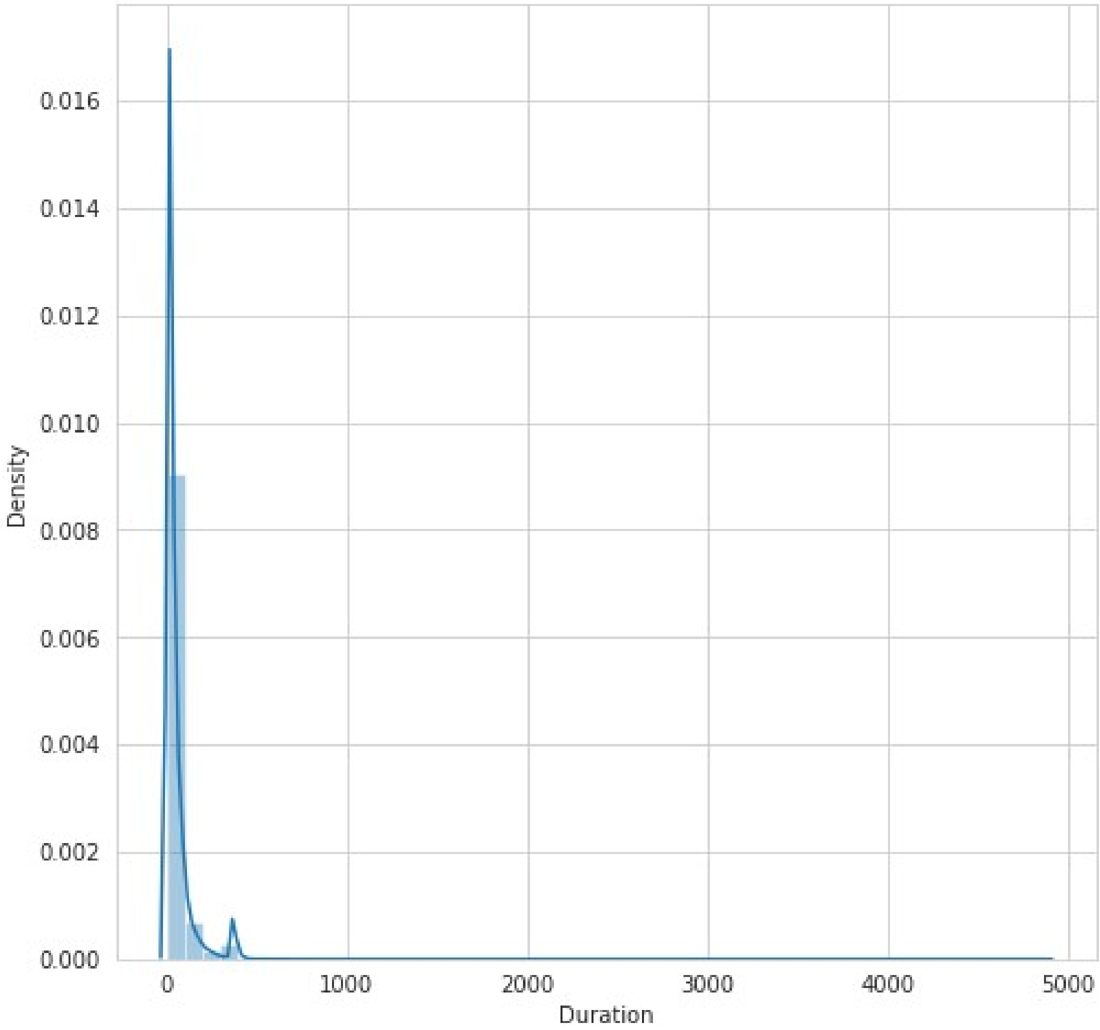
Skewness

```
from scipy.stats import skew

for col in df_num:
  print(col,":-",skew(df_num[col]))
  plt.figure(figsize=(8,8))
  sns.distplot(df_num[col])
  plt.show()
```

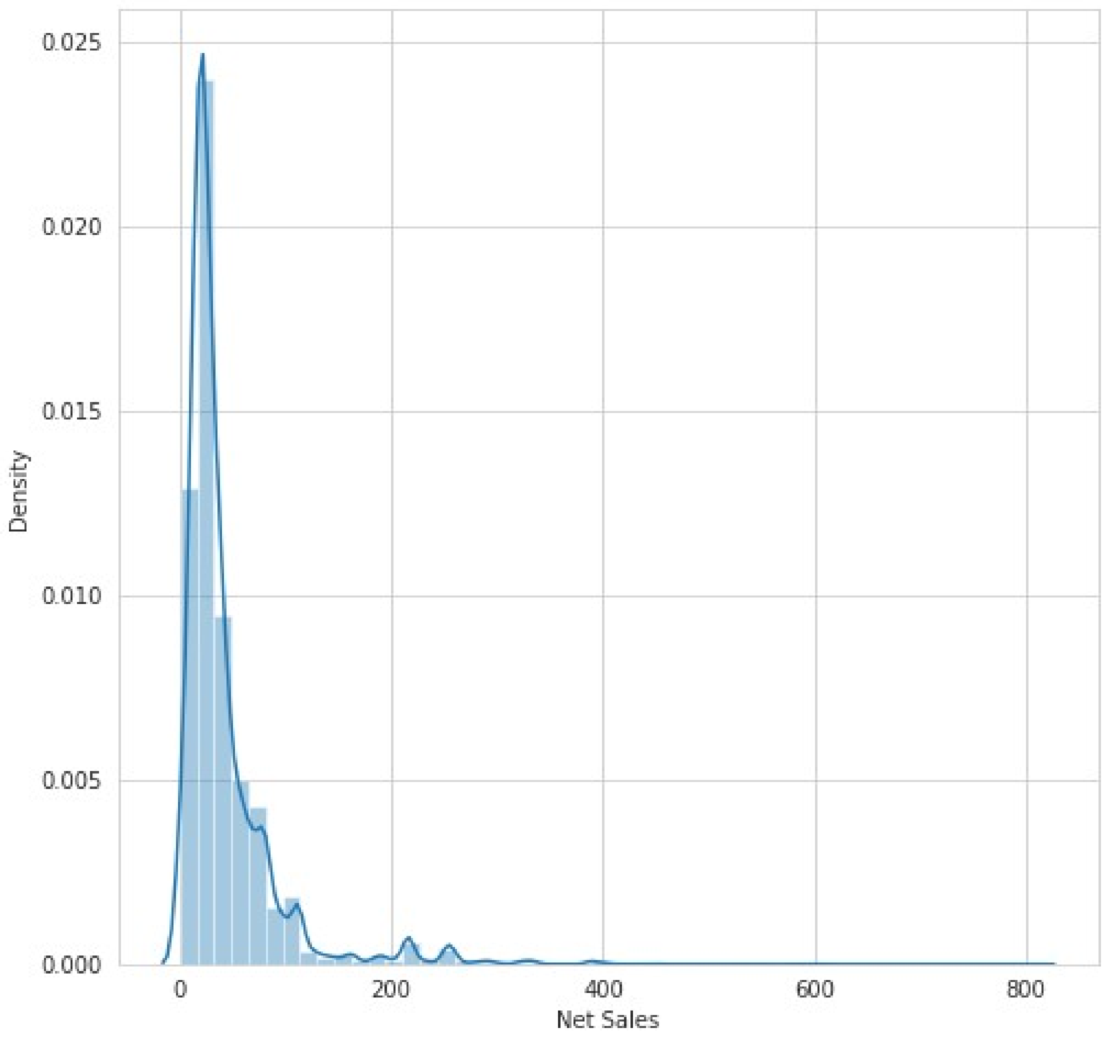
```
Claim :- 8.076976414369875
```



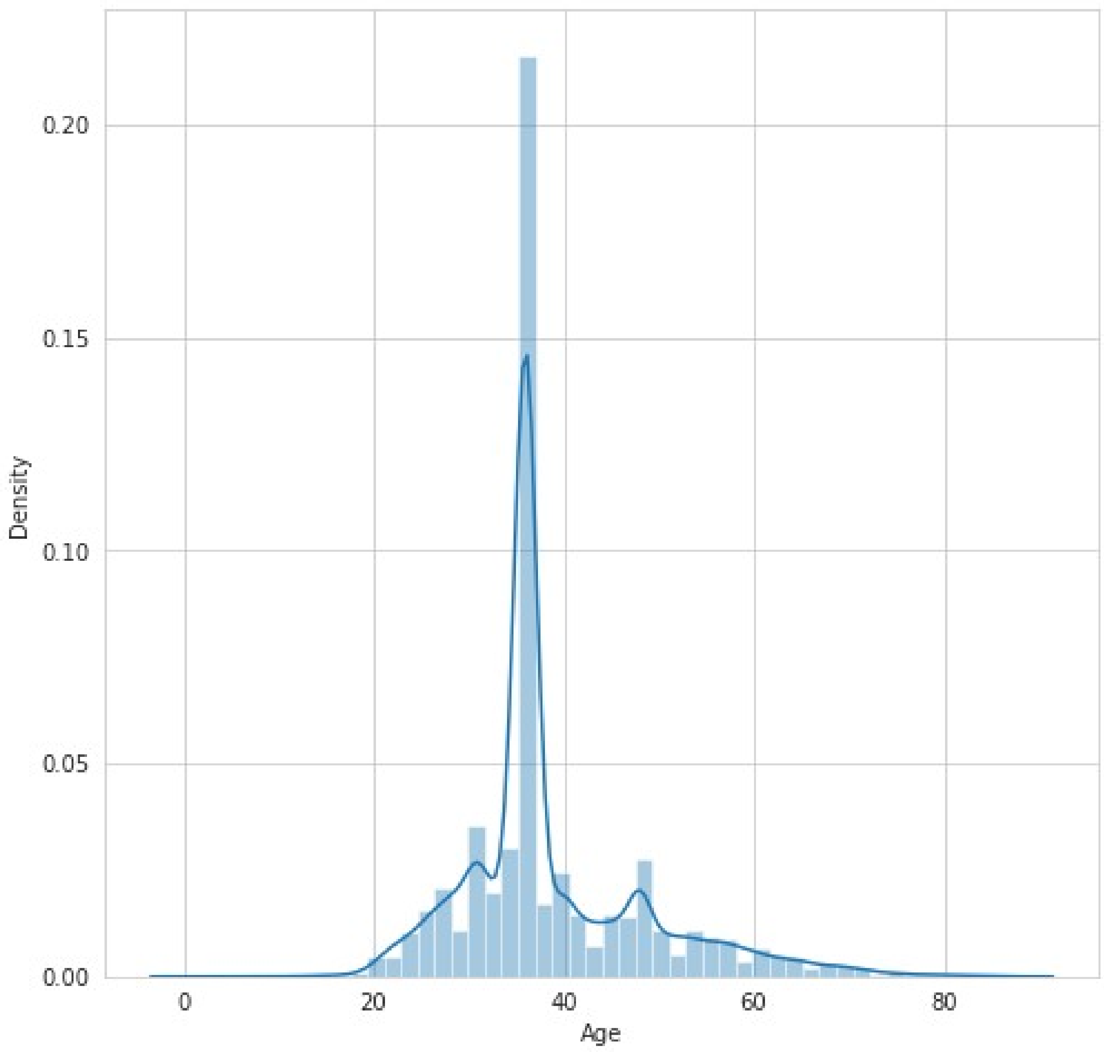
```
Duration :- 22.872492167935484
```



```
Net Sales :- 3.751192202882967
```

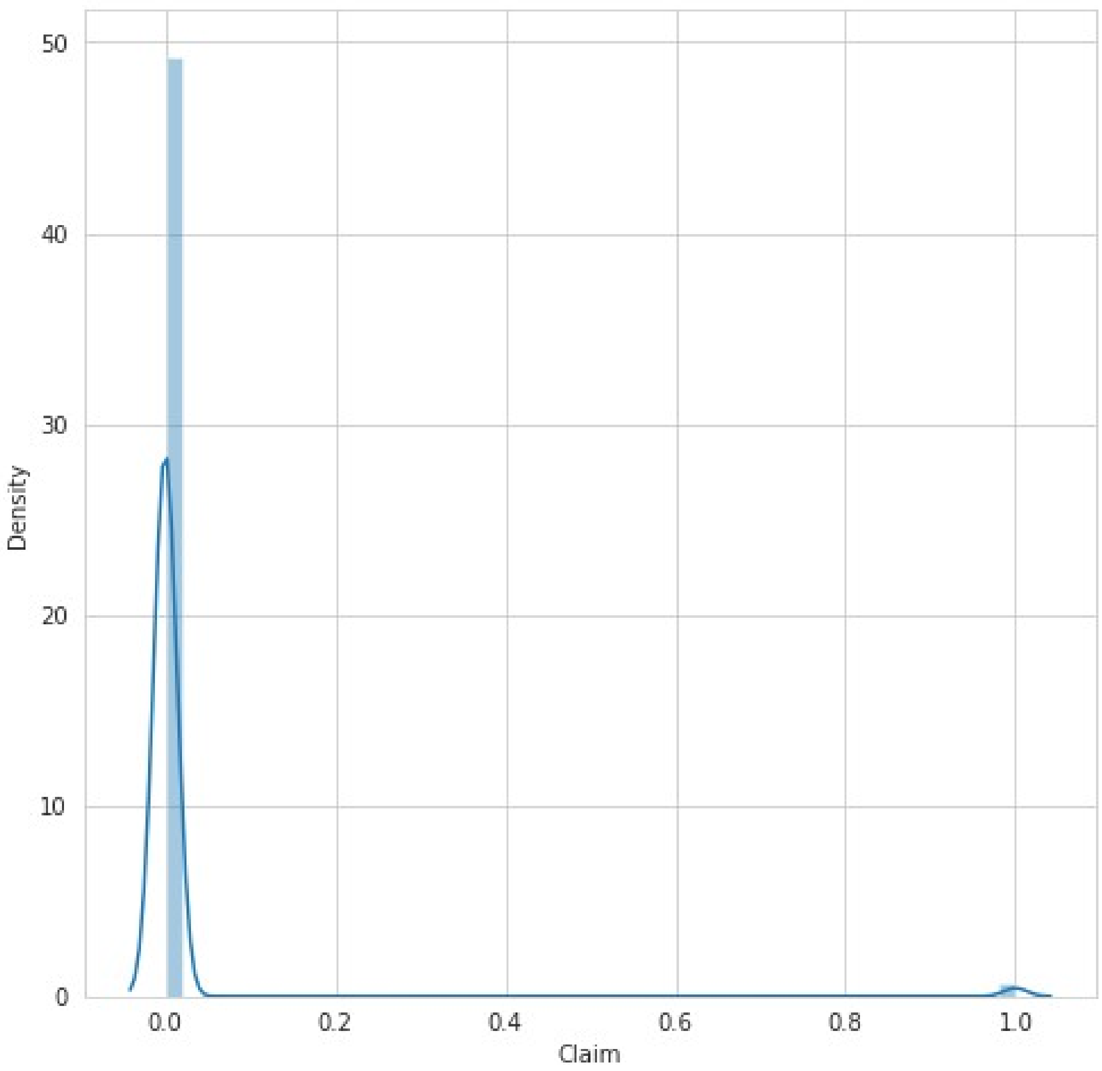


```
Age :- 1.2031538221807883
```

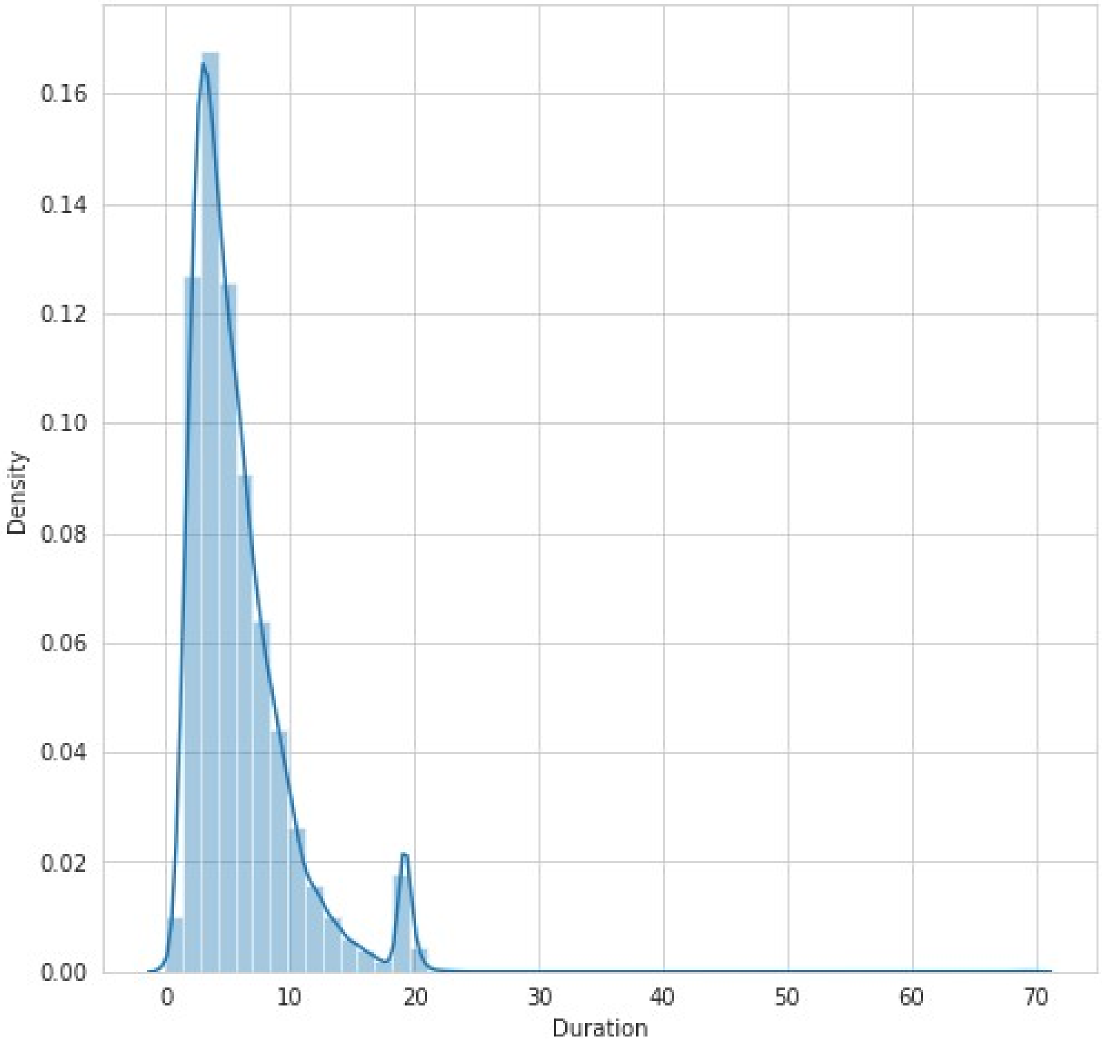


```
#removing skewness
for col in df_num:
  if skew(df_num[col]) >= 0.5 or skew(df_num[col]) <= -0.5:
    df_num[col] = np.sqrt(df_num[col])

for col in df_num:
  print(col,":-",skew(df_num[col]))
  plt.figure(figsize=(8,8))
  sns.distplot(df_num[col])
  plt.show()
```

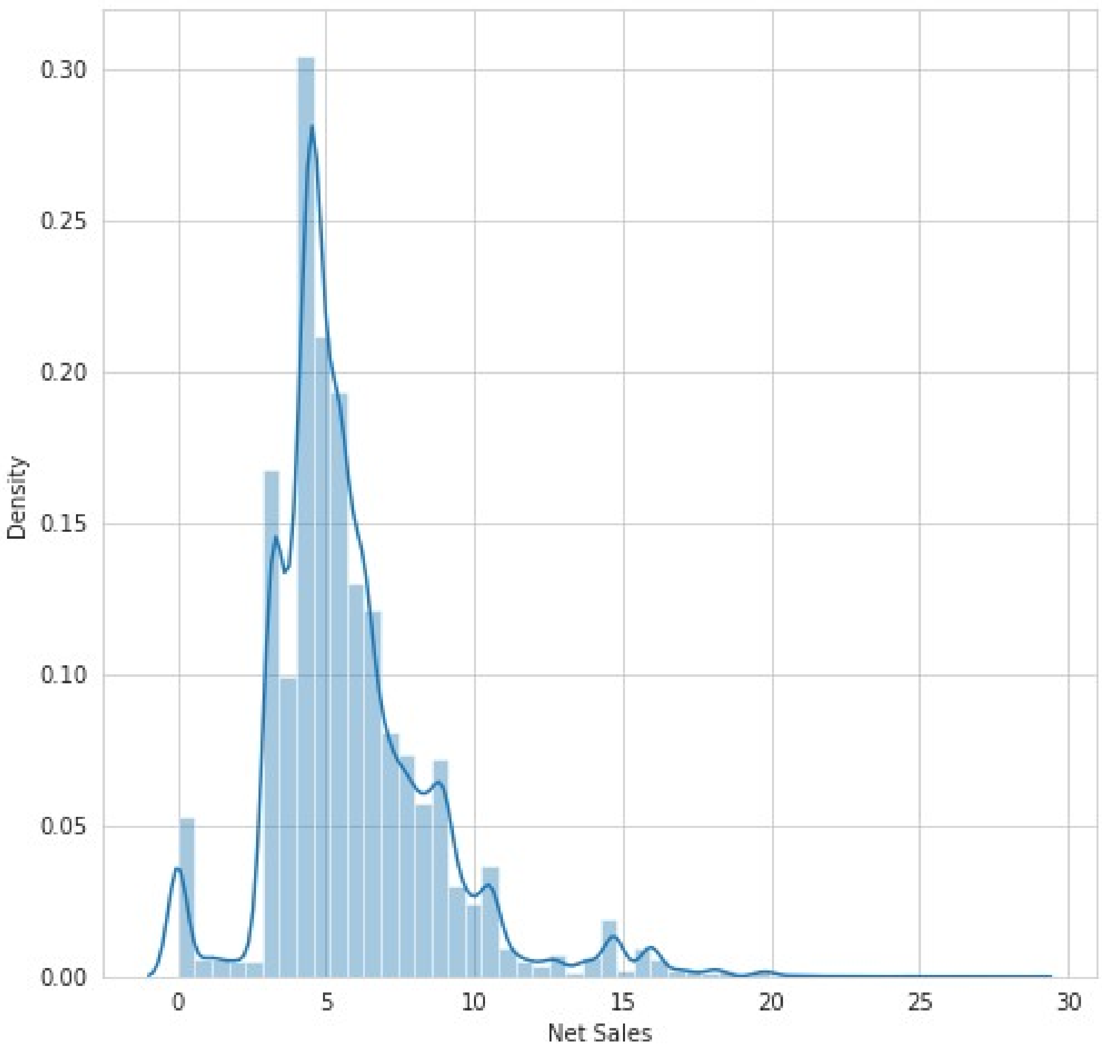
```
Claim :- 8.076976414369875
```



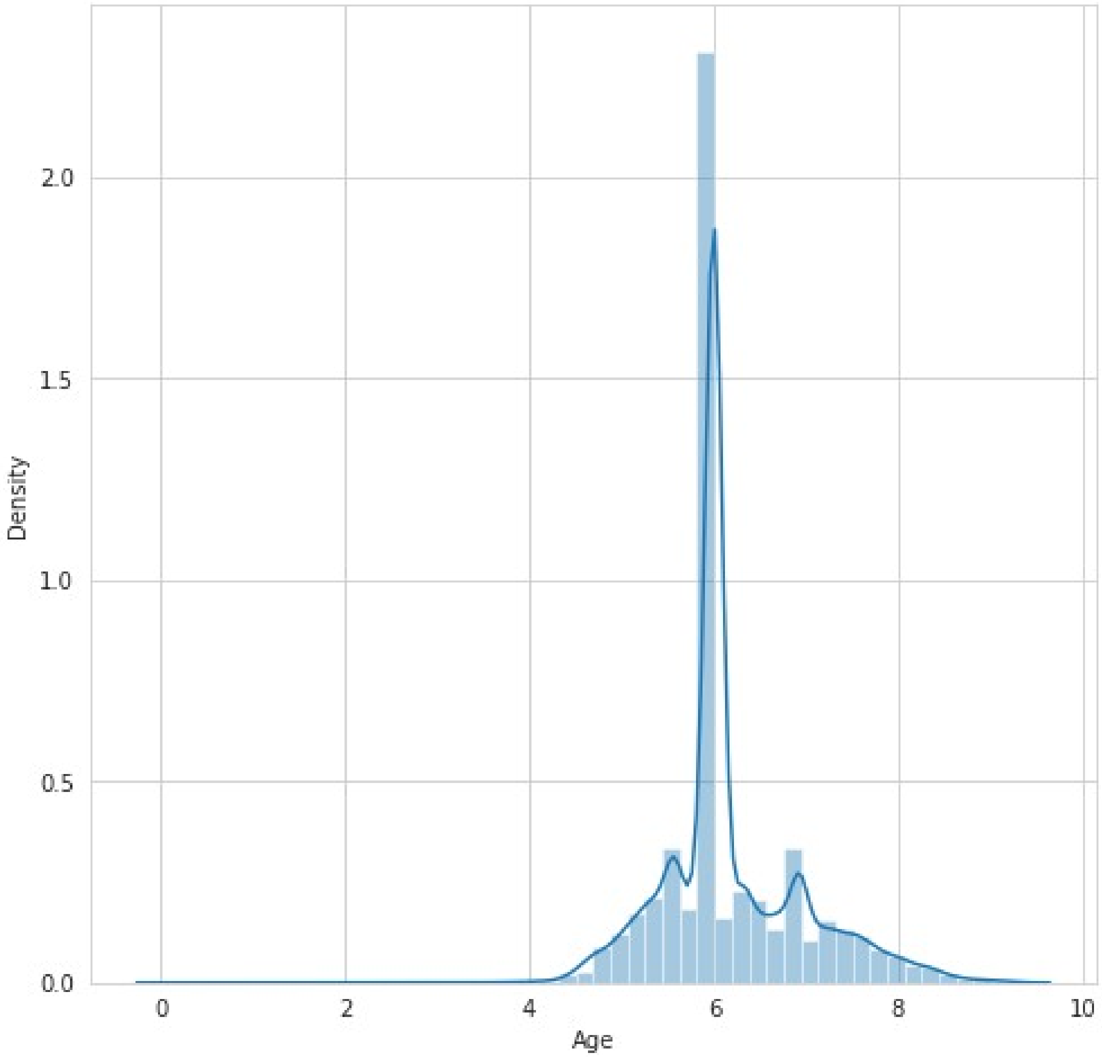
```
Duration :- 2.409425954993082
```



```
Net Sales :- 1.4439989184434878
```

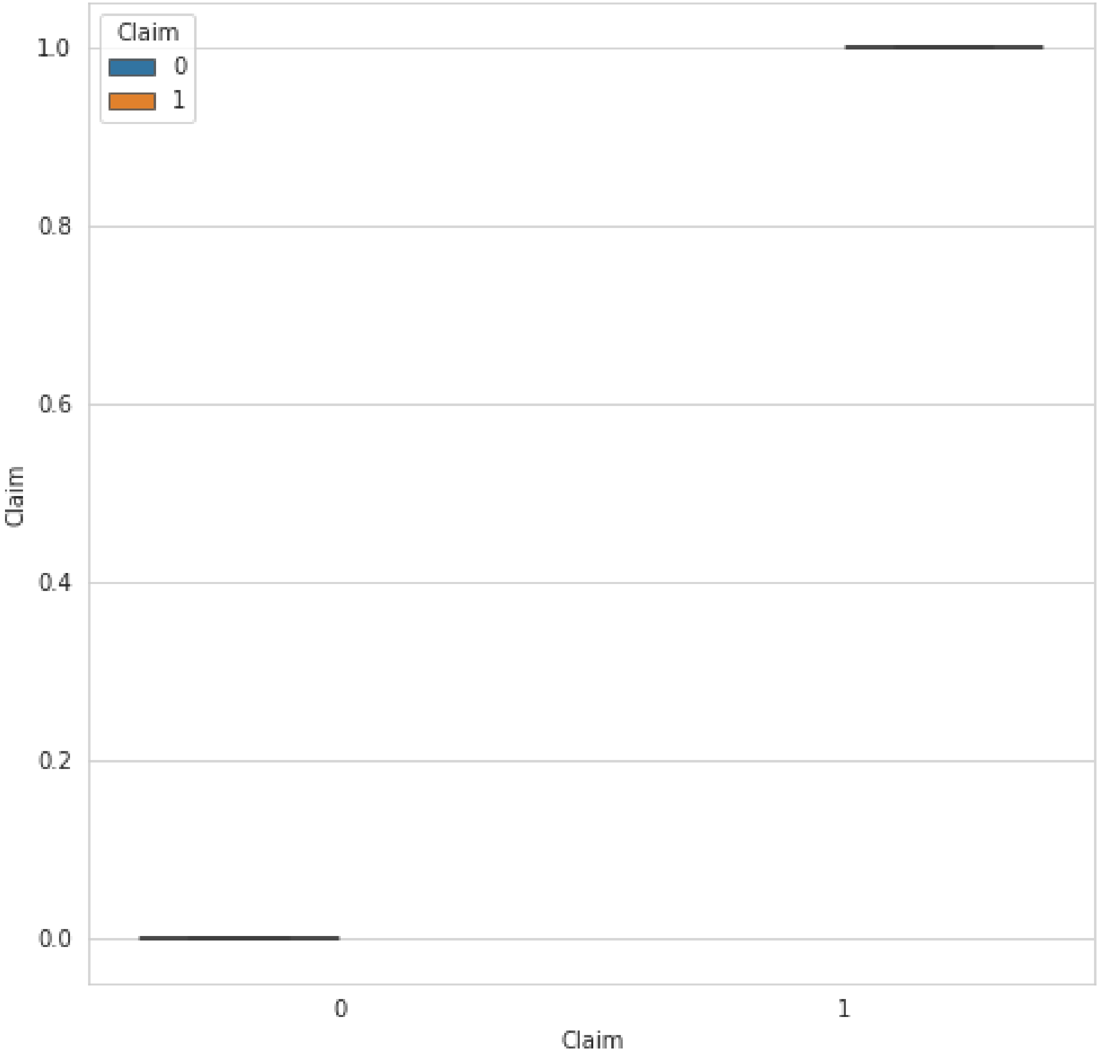


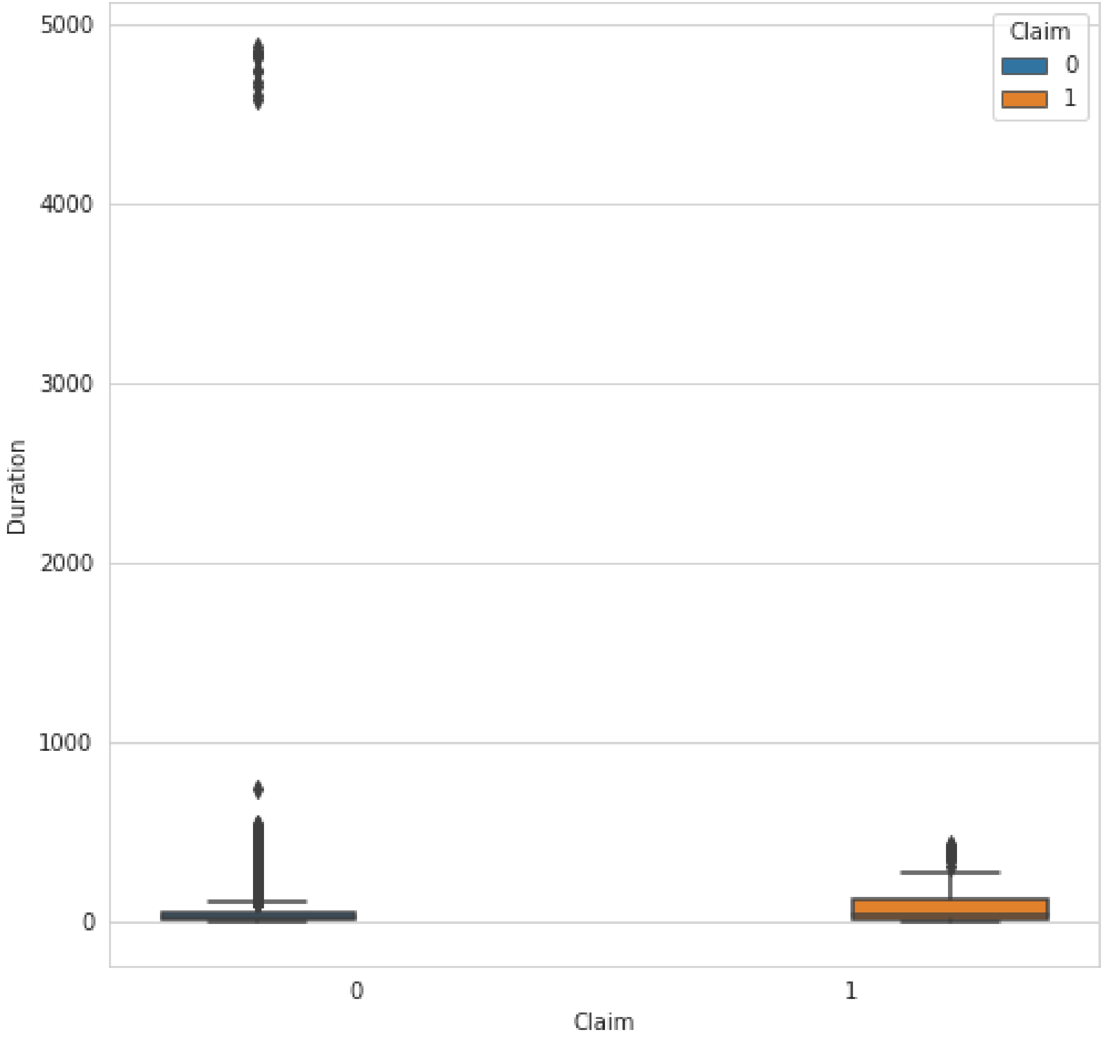
```
Age :- 0.6848711397302739
```

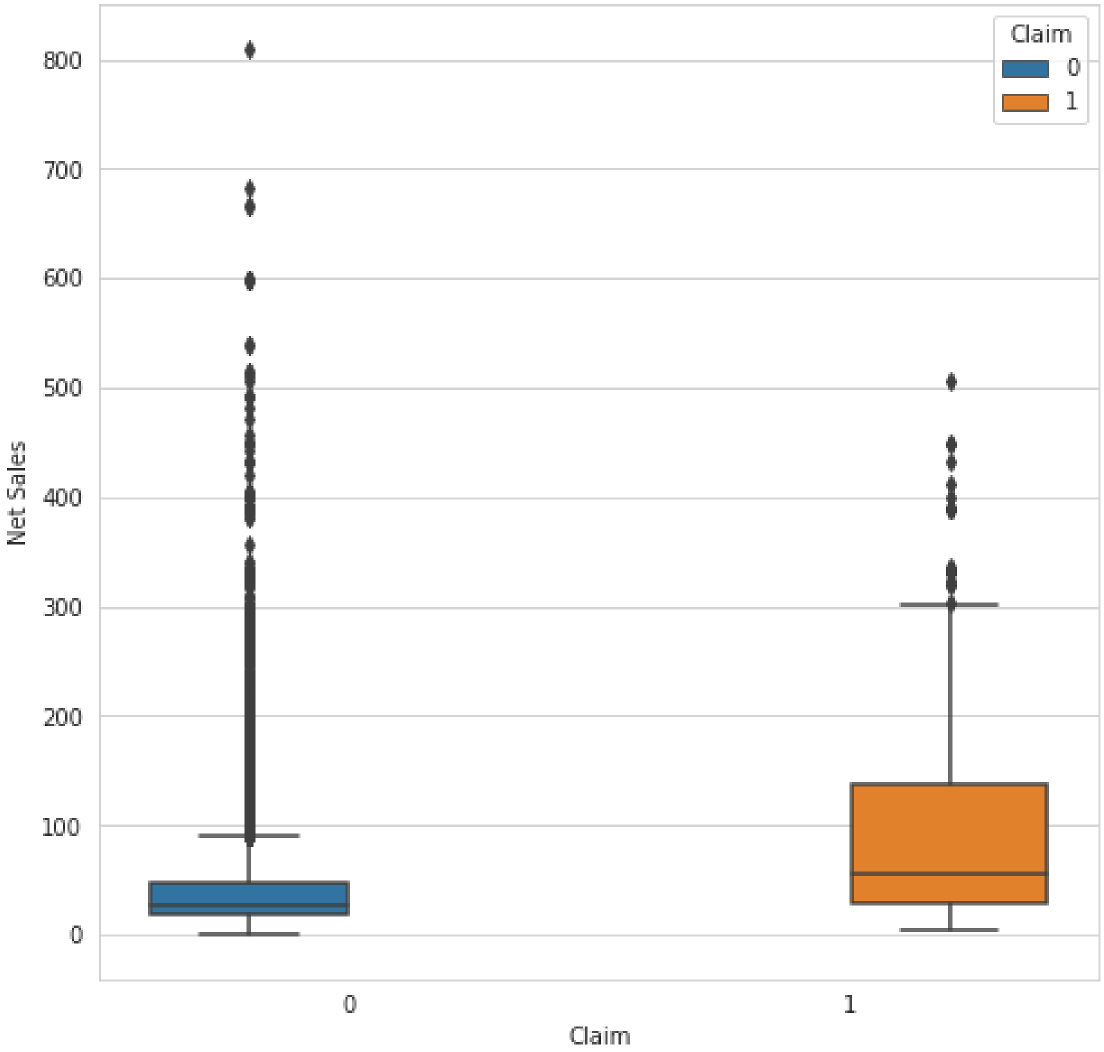


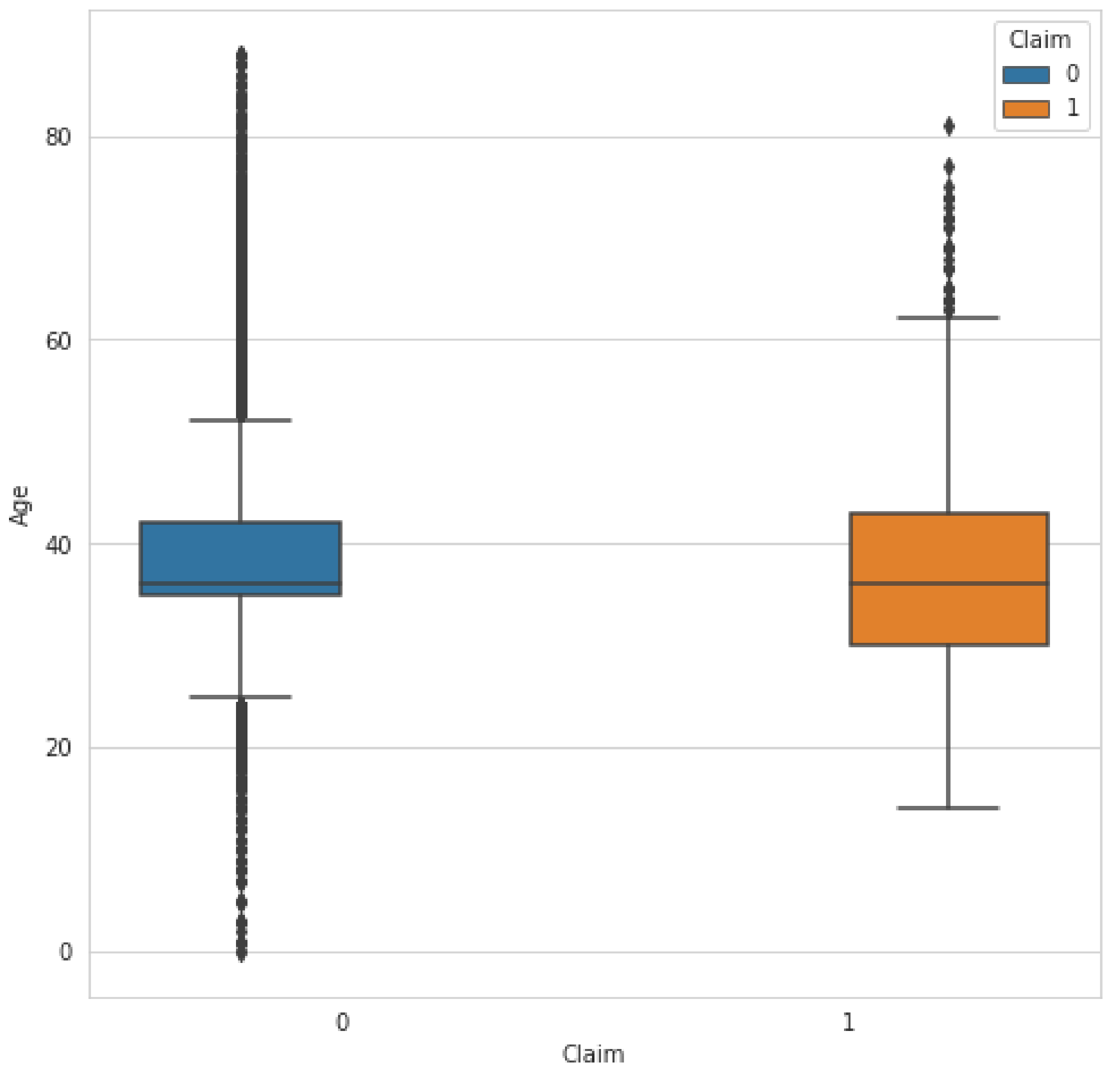
Bivariate Analysis against the traget

```
for col in df_num:
  plt.figure(figsize=(8,8))
  sns.boxplot(data=df,x="Claim",y=col , hue='Claim')
  plt.show()
```


Claim
0
1
1.0
0.8
0.6
0.4
0.2
0.0
Claim
0
1
Claim


Claim
0
1
5000
4000
3000
2000
1000
0
Duration
0
1
Claim


Claim
0
1
800
700
600
500
400
300
200
100
0
Net Sales
0
1
Claim



Scaling

```
from sklearn.preprocessing import MinMaxScaler

for col in df_num:
  ms = MinMaxScaler()
  df_num[col] = ms.fit_transform(df_num[[col]])

df_num.head()
```

```
   Claim  Duration  Net Sales       Age
0    0.0  0.037870   0.000000  0.593526
1    0.0  0.131964   0.291865  0.639602
2    0.0  0.047472   0.156347  0.923186
3    0.0  0.057254   0.157135  0.603023
4    0.0  0.045263   0.136083  0.574060
```

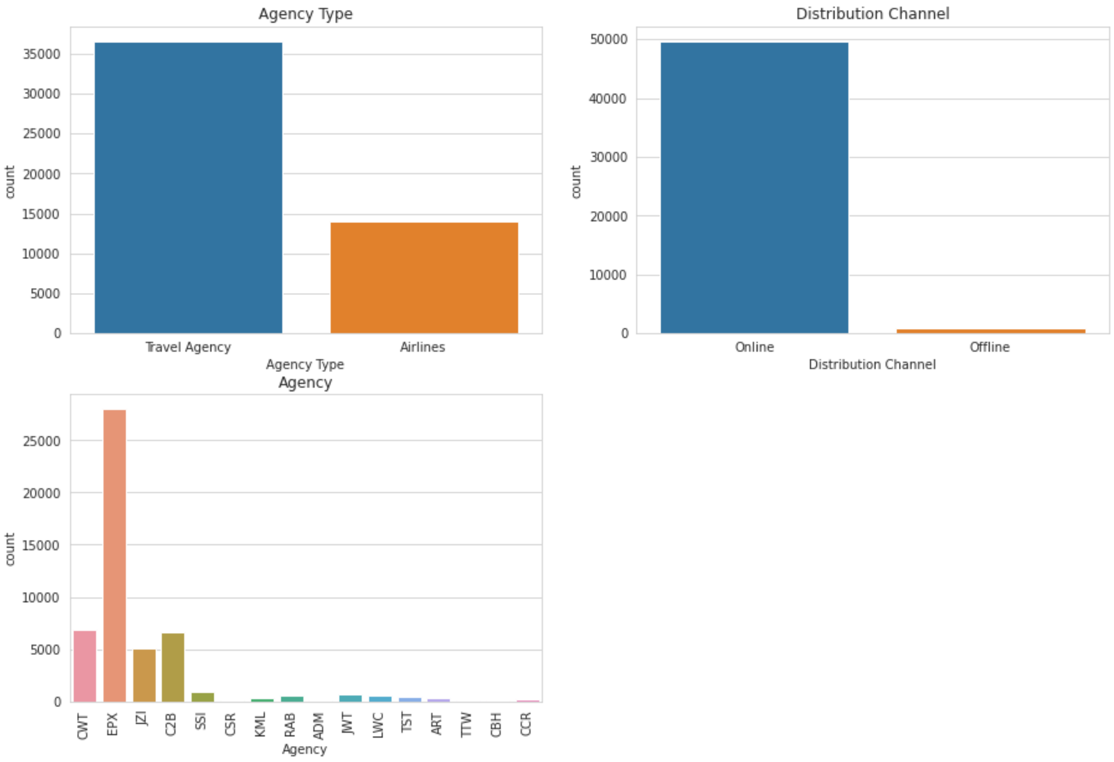
For df_cat

```
df_cat.head()
```

```
  Agency    Agency Type  ...                     Product Name
Destination
0    CWT  Travel Agency  ...  Rental Vehicle Excess Insurance
MALAYSIA
1    EPX  Travel Agency  ...                Cancellation Plan
SINGAPORE
2    CWT  Travel Agency  ...  Rental Vehicle Excess Insurance
MALAYSIA
3    EPX  Travel Agency  ...         2 way Comprehensive Plan
INDONESIA
4    EPX  Travel Agency  ...                Cancellation Plan  KOREA,
REPUBLIC OF

[5 rows x 5 columns]
```

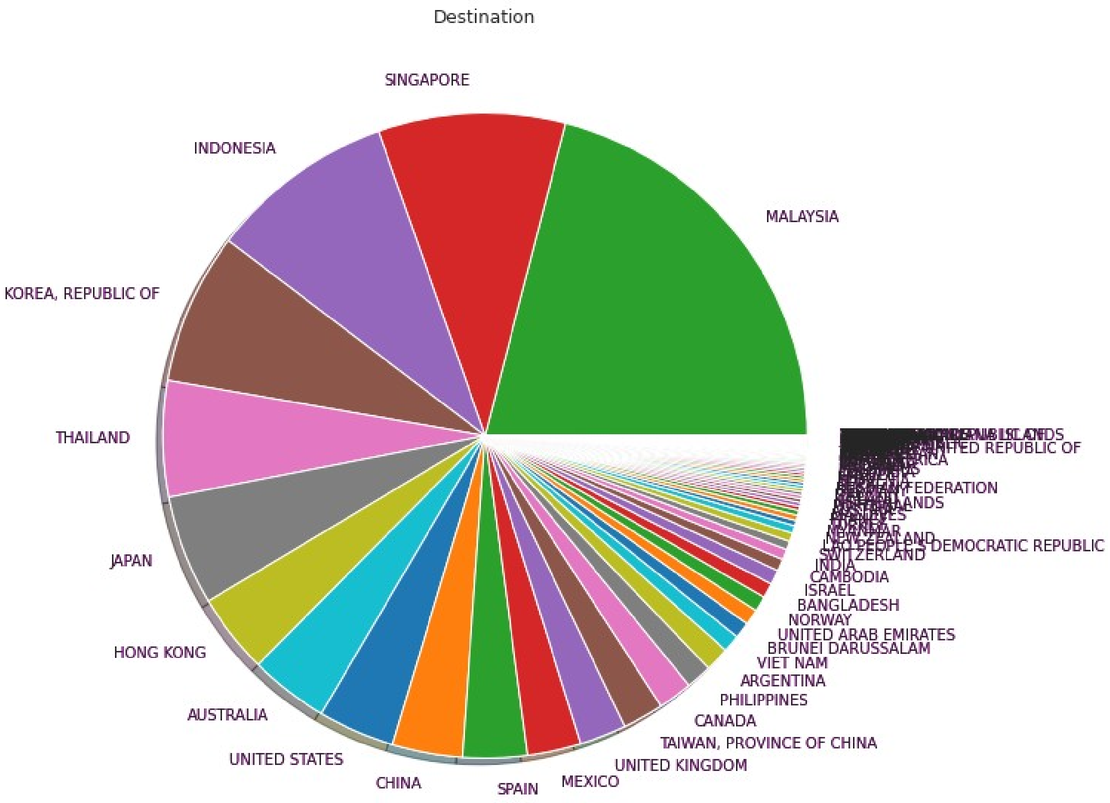
```
plt.figure(figsize=(15,10))
plt.subplot(2,2,1)
sns.countplot(df['Agency Type'])
plt.title('Agency Type')
plt.subplot(2,2,2)
sns.countplot(df['Distribution Channel'])
plt.title('Distribution Channel')

plt.subplot(2,2,3)
sns.countplot(df['Agency'])
plt.xticks(rotation=90)
plt.title('Agency')
```

```
Text(0.5, 1.0, 'Agency')
```



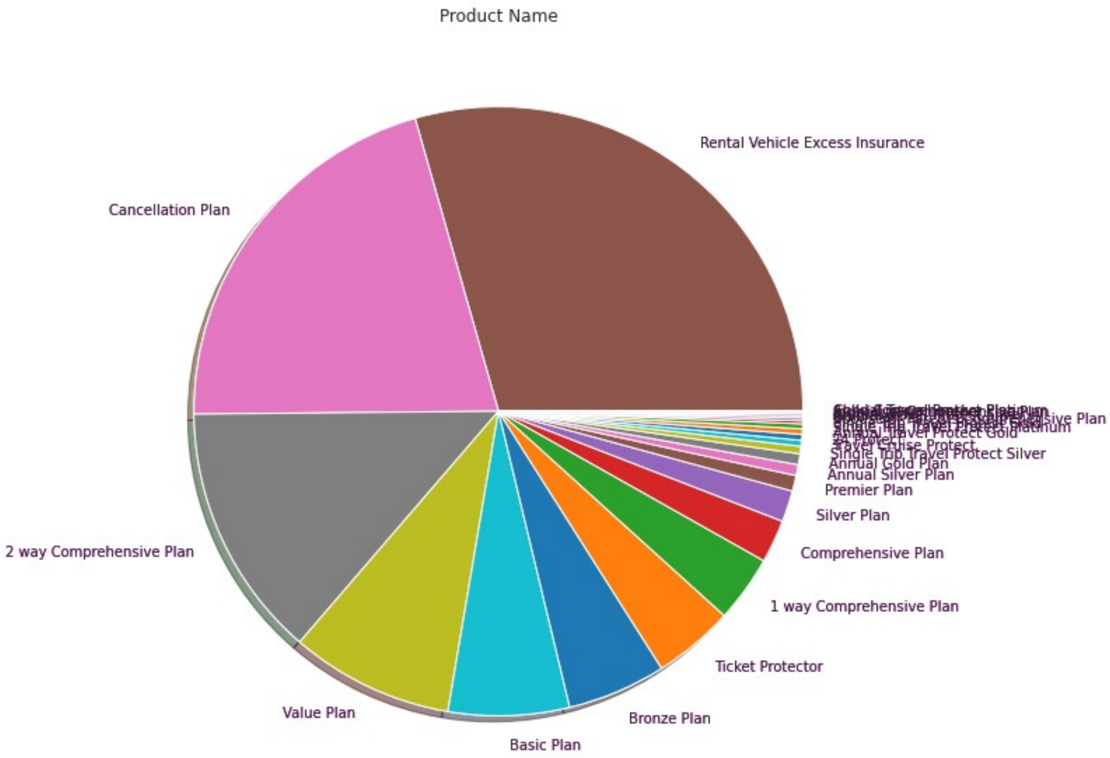
```
plt.figure(figsize=(15,10))

wedges, texts = plt.pie(df['Destination'].value_counts(),
                                  labels = df['Destination'].unique(),

                                  shadow = True,
                                  textprops = dict(color ="magenta"))
plt.pie(df['Destination'].value_counts(),labels=df['Destination'].uniq
ue())
plt.title('Destination')
```

```
Text(0.5, 1.0, 'Destination')
```



```
plt.figure(figsize=(15,10))

wedges, texts = plt.pie(df['Product Name'].value_counts(),
                                  labels = df['Product
Name'].unique(),
                                  shadow = True,
                                  textprops = dict(color ="magenta"))
plt.pie(df['Product Name'].value_counts(),labels=df['Product
Name'].unique())
plt.title('Product Name')
```

```
Text(0.5, 1.0, 'Product Name')
```



Encoding

```
from sklearn.preprocessing import LabelEncoder

for col in df_cat:
  le = LabelEncoder()
  df_cat[col] = le.fit_transform(df_cat[[col]])

df_cat.head()
```

```
   Agency  Agency Type  Distribution Channel  Product Name
Destination
0       6            1                     1            16
56
1       7            1                     1            10
79
2       6            1                     1            16
56
3       7            1                     1             1
38
4       7            1                     1            10
47
```

Concatenate Both df_num and df_cat

```
df_new = pd.concat([df_num,df_cat], axis=1)
```

```
df_new.head()
```

```
   Claim  Duration  Net Sales  ...  Distribution Channel  Product Name  
Destination  
0    0.0  0.037870   0.000000  ...                     1            16  
56  
1    0.0  0.131964   0.291865  ...                     1            10  
79  
2    0.0  0.047472   0.156347  ...                     1            16  
56  
3    0.0  0.057254   0.157135  ...                     1             1  
38  
4    0.0  0.045263   0.136083  ...                     1            10  
47  

[5 rows x 9 columns]
```

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

Train-Test Splitting bold text

```
from sklearn.model_selection import train_test_split
```

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

#**Baseline Model**

LogisticRegression

```
from sklearn.linear_model import LogisticRegression
```

```
model = LogisticRegression()
```

```
model.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
y_pred = model.predict(X_test)
```

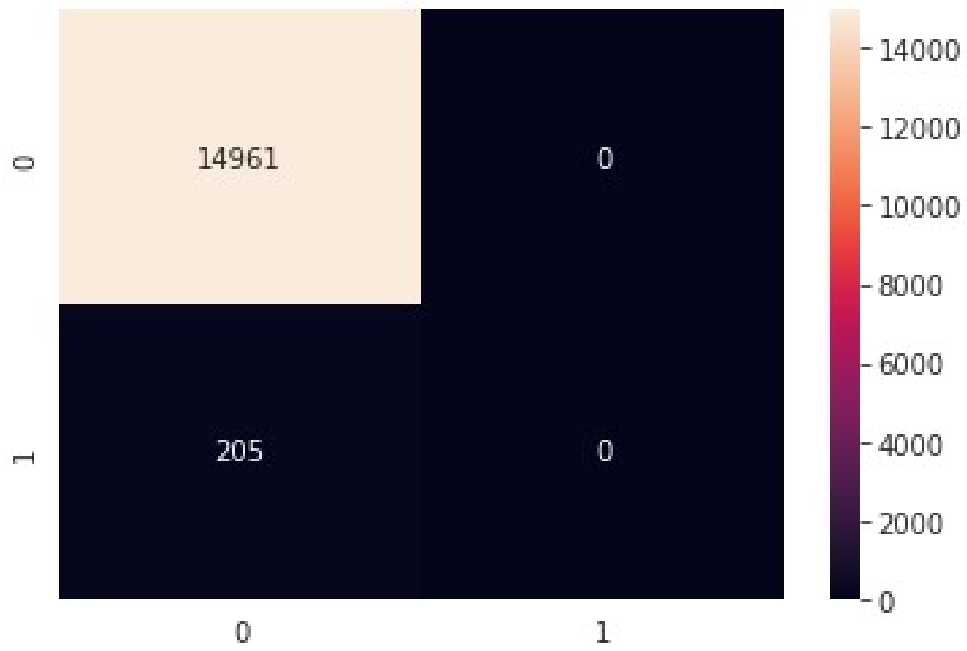
```
from sklearn.metrics import accuracy_score, precision_score, 
recall_score, f1_score, classification_report, confusion_matrix
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.99      1.00      0.99     14961
         1.0       0.00      0.00      0.00       205

    accuracy                           0.99     15166
   macro avg       0.49      0.50      0.50     15166
```

```
weighted avg       0.97      0.99      0.98     15166
```

```
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d')
plt.show()
```



Using RandomOverSampler to Balanced the training data

```
from imblearn.over_sampling import RandomOverSampler
```

```
ros = RandomOverSampler(random_state=1)
```

```
X_sample1, y_sample1 = ros.fit_resample(X_train,y_train)
```

```
pd.Series(y_sample1).value_counts()
```

```
1.0    34851
0.0    34851
Name: Claim, dtype: int64
```

```
lr2 = LogisticRegression()
lr2.fit(X_sample1, y_sample1)
```

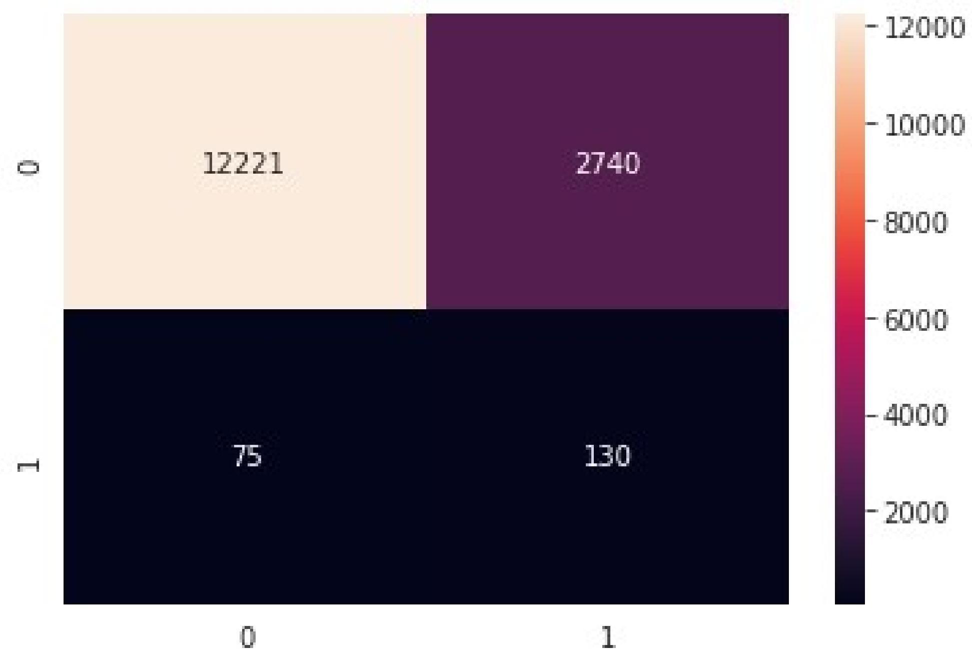
```
LogisticRegression()
```

```
y_pred1= lr2.predict(X_test)
```

```
print(classification_report(y_test,y_pred1))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.82      0.90     14961
         1.0       0.05      0.63      0.08       205

    accuracy                           0.81     15166
   macro avg       0.52      0.73      0.49     15166
weighted avg       0.98      0.81      0.89     15166
```

```
sns.heatmap(confusion_matrix(y_test, y_pred1), annot=True, fmt='d')
plt.show()
```



under sampling

```
from imblearn.under_sampling import RandomUnderSampler
```

```
rus = RandomUnderSampler(random_state=1)
```

```
X_sample2, y_sample2 = rus.fit_resample(X_train,y_train)
```

```
pd.Series(y_sample2).value_counts()
```

```
1.0    536
0.0    536
Name: Claim, dtype: int64
```

```
lr3 = LogisticRegression()
lr3.fit(X_sample2, y_sample2)
```

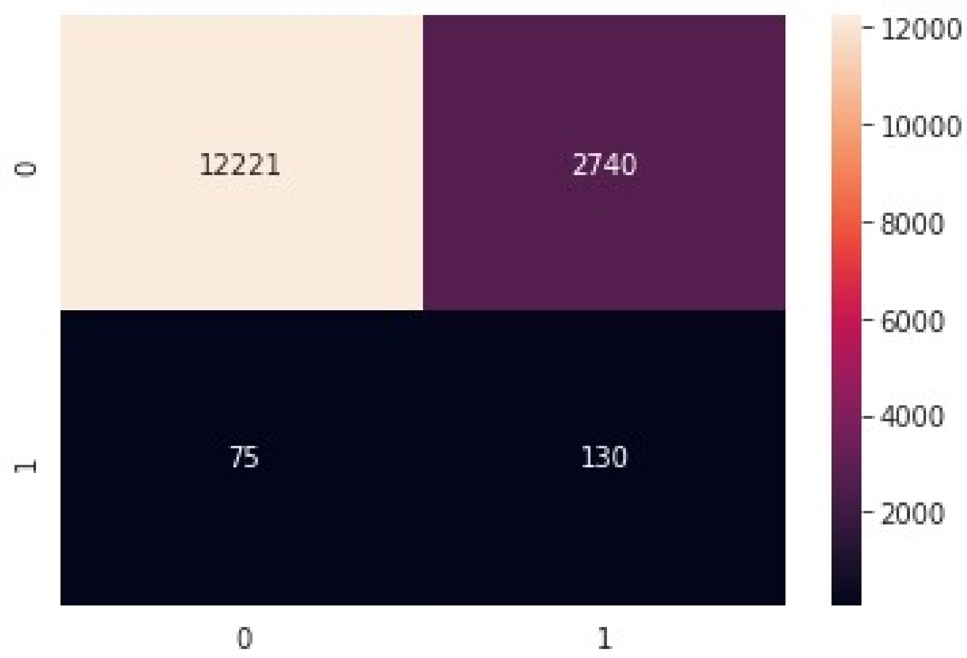
```
LogisticRegression()
```

```
y_pred2 = lr2.predict(X_test)
```

```
print(classification_report(y_test,y_pred2))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.82      0.90     14961
         1.0       0.05      0.63      0.08       205

    accuracy                           0.81     15166
   macro avg       0.52      0.73      0.49     15166
weighted avg       0.98      0.81      0.89     15166
```

```
from sklearn.metrics import confusion_matrix, classification_report
sns.heatmap(confusion_matrix(y_test, y_pred2), annot=True, fmt='d')
plt.show()
```



#**Decision tree** Here we use DecisionTree with RandomOverSampler Technique to Balanced The DataFrame and to get classification report

```
from sklearn.tree import DecisionTreeClassifier

ros = RandomOverSampler(random_state=1)

X_sample3, y_sample3 = ros.fit_resample(X_train,y_train)

pd.Series(y_sample3).value_counts()

1.0    34851
0.0    34851
Name: Claim, dtype: int64

dtc=
DecisionTreeClassifier(max_depth=8 ,min_samples_leaf=50,criterion="ent
ropy")
dtc.fit(X_sample3, y_sample3)
```

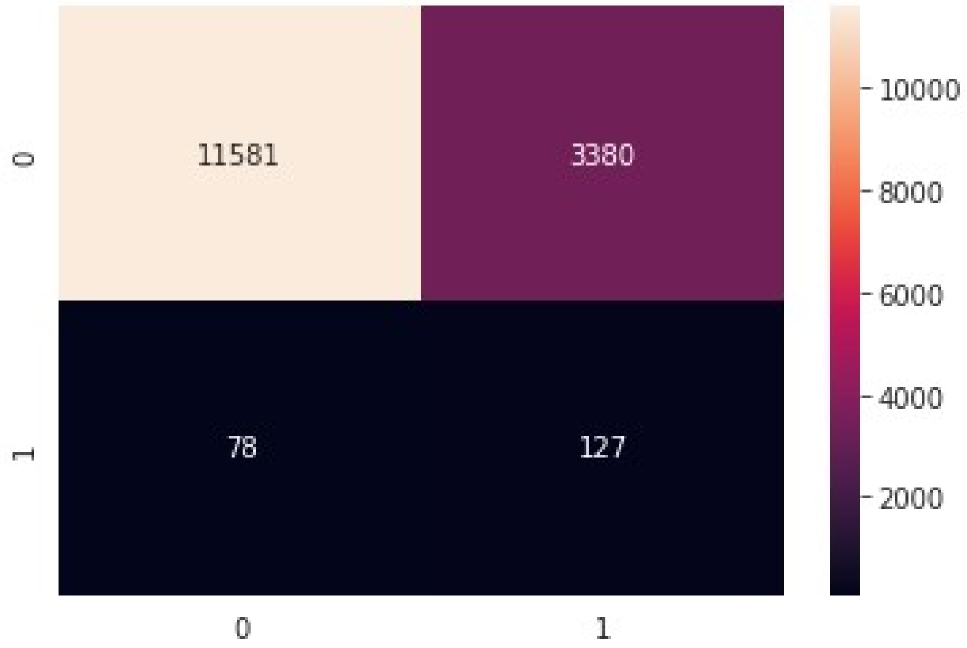
```
DecisionTreeClassifier(criterion='entropy', max_depth=8,
min_samples_leaf=50)
```

```
y_pred3 = dtc.predict(X_test)
```

```
print(classification_report(y_test,y_pred3))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.77      0.87     14961
         1.0       0.04      0.62      0.07       205

    accuracy                           0.77     15166
   macro avg       0.51      0.70      0.47     15166
weighted avg       0.98      0.77      0.86     15166
```

```
sns.heatmap(confusion_matrix(y_test, y_pred3), annot=True, fmt='d')
plt.show()
```



Random forest

```
from sklearn.ensemble import RandomForestClassifier
```

```
ros = RandomOverSampler(random_state=1)
```

```
X_sample4, y_sample4 = ros.fit_resample(X_train,y_train)
```

```
pd.Series(y_sample4).value_counts()
```

```
1.0    34851
0.0    34851
Name: Claim, dtype: int64
```

```
rtc=
RandomForestClassifier(n_estimators=100,random_state=1,max_features=8,
max_depth=5)
rtc.fit(X_sample4, y_sample4)
```

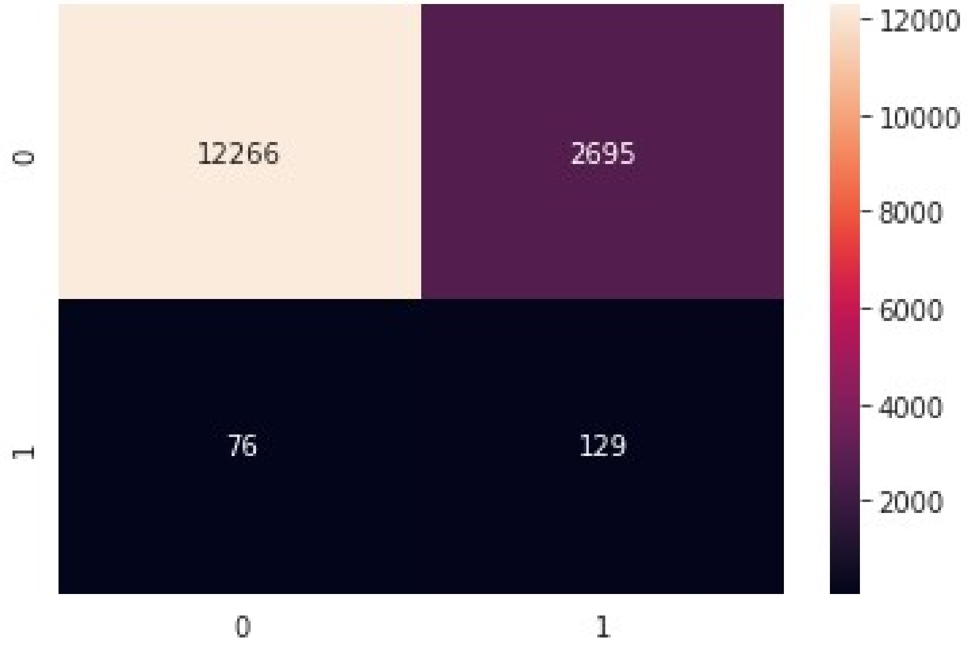
```
RandomForestClassifier(max_depth=5, max_features=8, random_state=1)
```

```
y_pred4 = rtc.predict(X_test)
```

```
print(classification_report(y_test,y_pred4))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.82      0.90     14961
         1.0       0.05      0.63      0.09       205

    accuracy                           0.82     15166
   macro avg       0.52      0.72      0.49     15166
weighted avg       0.98      0.82      0.89     15166
```

```
sns.heatmap(confusion_matrix(y_test, y_pred4), annot=True, fmt='d')
plt.show()
```



#Gradient Boosting Classifier

```
#Gradient Boost
from sklearn.ensemble import GradientBoostingClassifier

ros = RandomOverSampler(random_state=1)

X_sample5, y_sample5 = ros.fit_resample(X_train,y_train)

pd.Series(y_sample5).value_counts()
```

```
1.0    34851
0.0    34851
Name: Claim, dtype: int64
```

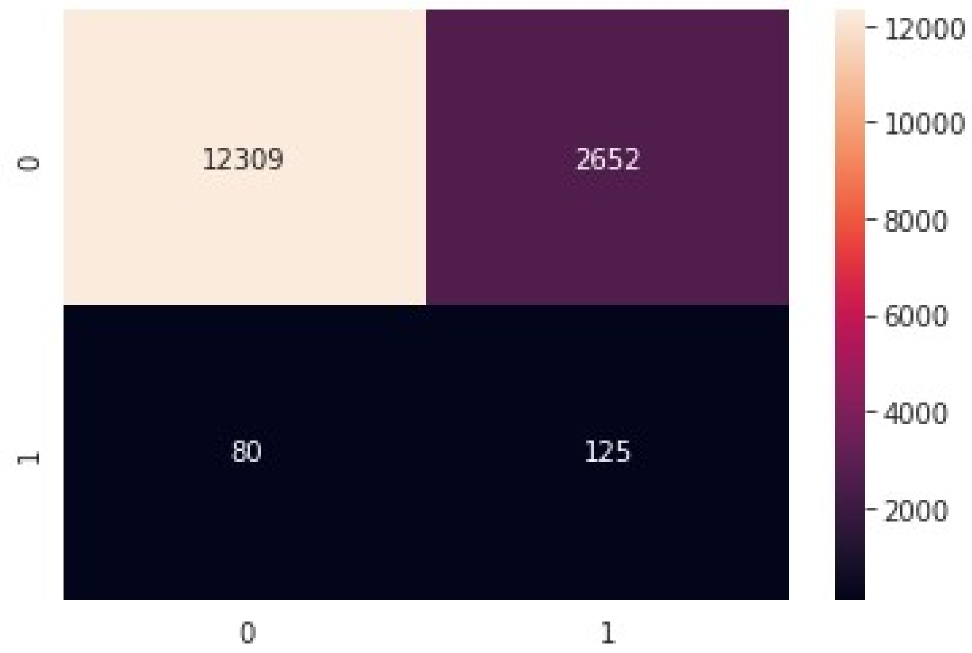
```
gb=GradientBoostingClassifier(n_estimators=100)

gb.fit(X_sample5, y_sample5)
```

```
GradientBoostingClassifier()
```

```
y_pred5 = gb.predict(X_test)

print(classification_report(y_test,y_pred5))
```

```
              precision    recall  f1-score   support

         0.0       0.99      0.82      0.90     14961
         1.0       0.05      0.61      0.08       205

    accuracy                           0.82     15166
   macro avg       0.52      0.72      0.49     15166
weighted avg       0.98      0.82      0.89     15166
```

```
sns.heatmap(confusion_matrix(y_test, y_pred5), annot=True, fmt='d')
plt.show()
```

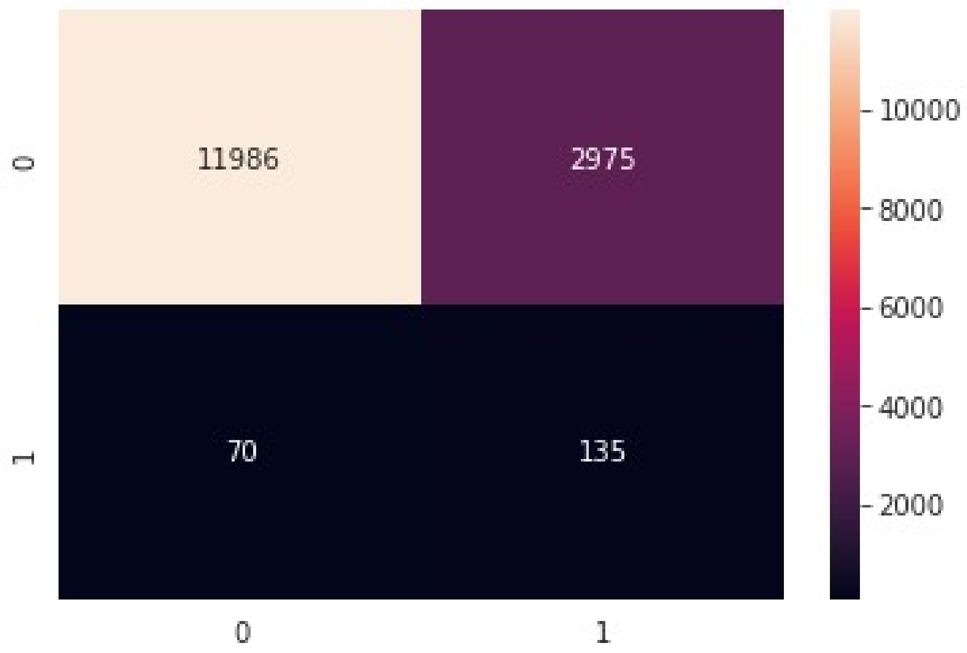


#Adaboost Boosting Classifier

```
from sklearn.ensemble import AdaBoostClassifier

ros = RandomOverSampler(random_state=1)

X_sample6, y_sample6 = ros.fit_resample(X_train,y_train)

pd.Series(y_sample5).value_counts()

1.0    34851
0.0    34851
Name: Claim, dtype: int64

ab=AdaBoostClassifier(n_estimators=100)

ab.fit(X_sample6, y_sample6)

AdaBoostClassifier(n_estimators=100)

y_pred6 = ab.predict(X_test)

print(classification_report(y_test,y_pred6))

              precision    recall  f1-score   support

         0.0       0.99      0.80      0.89     14961
         1.0       0.04      0.66      0.08       205

    accuracy                           0.80     15166
   macro avg       0.52      0.73      0.48     15166
```

```
weighted avg       0.98      0.80      0.88     15166
```

```
sns.heatmap(confusion_matrix(y_test, y_pred6), annot=True, fmt='d')
plt.show()
```


11986
2975
70
135
0
1
0
1
10000
8000
6000
4000
2000