Goal:

The goal of this project is to analyse the available data and predict whether to approved the insurance or not based on some of features.

Problem statement:

For any insurance policy, we would like to know whether some features lead to the policy claimed. Some features like age, gender or commision etc.

In [1]:

```
##Importing Libraries:
```

In [2]:

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

from scipy.stats import skew

from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler


from sklearn.model_selection import train_test_split

from sklearn.model_selection import cross_val_score

from sklearn.metrics import classification_report,confusion_matrix
from sklearn.metrics import roc_auc_score ,roc_curve

from sklearn.feature_selection import chi2
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import SelectKBest

from sklearn.ensemble import VotingClassifier

from sklearn.linear_model import LogisticRegression
from sklearn.tree import  DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.svm import LinearSVC

import warnings
warnings.filterwarnings("ignore")
```

In [3]:

```
##Let’s import the dataset using read_csv method and assign it to the variable ‘df’.
```

In [4]:

```
df = pd.read_csv("data.csv")
```

EDA & Preprocessing

In [5]:

```
##Size of the dataset:
##We can get the size of the dataset using the .shape method
```

In [6]:

```
df.shape
```

Out[6]:

```
(50553, 12)
```

Describe

describe() is used to view some basic statistical details like count, percentiles, mean, std and maximum value of a data frame or a series of numeric values. As it gives the count of each variable, we can identify the missing values using this method.

In [7]:

```
df.describe()
```

Out[7]:

	ID	Claim	Duration	Net Sales	Commision (in value)	Age
count	50553.000000	50553.000000	50553.000000	50553.000000	50553.00000	50553.000000
mean	31679.740134	0.014658	49.425969	40.800977	9.83809	40.011236
std	18288.265350	0.120180	101.434647	48.899683	19.91004	14.076566
min	0.000000	0.000000	-2.000000	-389.000000	0.00000	0.000000
25%	15891.000000	0.000000	9.000000	18.000000	0.00000	35.000000
50%	31657.000000	0.000000	22.000000	26.500000	0.00000	36.000000
75%	47547.000000	0.000000	53.000000	48.000000	11.55000	44.000000
max	63325.000000	1.000000	4881.000000	810.000000	283.50000	118.000000

In [8]:

```
##We can get the null values ,data types and count
```

In [9]:

```
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 [10]:

```
df.head()
```

Out[10]:

	ID	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales
0	3433	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	7	MALAYSIA	0.0
1	4339	EPX	Travel Agency	Online	Cancellation Plan	0	85	SINGAPORE	69.0
2	34590	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	11	MALAYSIA	19.8
3	55816	EPX	Travel Agency	Online	2 way Comprehensive Plan	0	16	INDONESIA	20.0
4	13816	EPX	Travel Agency	Online	Cancellation Plan	0	10	KOREA, REPUBLIC OF	15.0

When we import our dataset from a CSV file, many blank columns are imported as null values into the Data Frame which can later create problems while operating that data frame

In [11]:

```
df.isnull().sum()
```

Out[11]:

```
ID                          0
Agency                      0
Agency Type                 0
Distribution Channel        0
Product Name                0
Claim                       0
Duration                    0
Destination                 0
Net Sales                   0
Commision (in value)        0
Gender                  35953
Age                         0
dtype: int64
```

In above code we have observed that only "Gender" have The Null Values

35953/50552 are null values

here we can remove the gender colomn because of very less data exist

we can also the remove ID column because it's not needed

In [12]:

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

In [13]:

```
df.isnull().sum()
```

Out[13]:

```
Agency                  0
Agency Type             0
Distribution Channel    0
Product Name            0
Claim                   0
Duration                0
Destination             0
Net Sales               0
Commision (in value)    0
Age                     0
dtype: int64
```

In [14]:

```
#separate numerical and categorical data
```

There are Numerical Columns And Categorical Columns ,so we can separate numerical and categorical data

In [15]:

```
df_num = df.select_dtypes(["int64","float"])
```

In [16]:

```
df_num.head()
```

Out[16]:

	Claim	Duration	Net Sales	Commision (in value)	Age
0	0	7	0.0	17.82	31
1	0	85	69.0	0.00	36
2	0	11	19.8	11.88	75
3	0	16	20.0	0.00	32
4	0	10	15.0	0.00	29

In [17]:

```
df_cat = df.select_dtypes(object)
```

In [18]:

```
df_cat.head()
```

Out[18]:

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

Numerical data

In [19]:

```
df_num.describe()
```

Out[19]:

	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

Clean Data

Some of data in some of columns are suspicious. If needed, we will transform the value of the data. Like Duration, negative values are strange.

In [20]:

```
df_num["Duration"].describe()
```

Out[20]:

```
count    50553.000000
mean        49.425969
std        101.434647
min         -2.000000
25%          9.000000
50%         22.000000
75%         53.000000
max       4881.000000
Name: Duration, dtype: float64
```

In [21]:

```
df_num[df_num["Duration"]<0]
```

Out[21]:

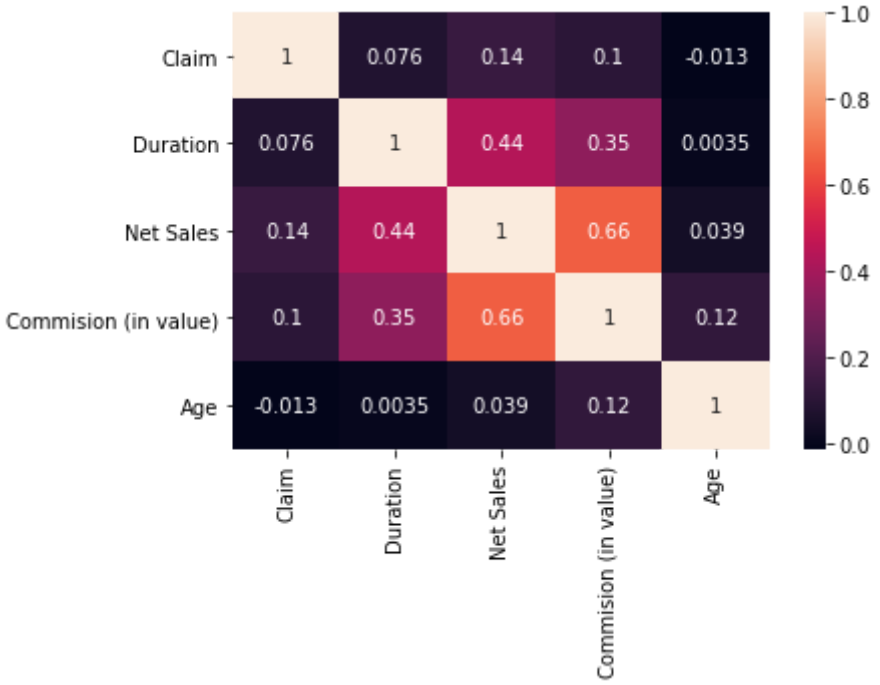
	Claim	Duration	Net Sales	Commision (in value)	Age
4063	0	-1	18.0	6.3	118
38935	0	-1	18.0	6.3	118
48367	0	-2	22.0	7.7	118

In [22]:

```
df_num.loc[df_num["Duration"]<0]=1
```

In [23]:

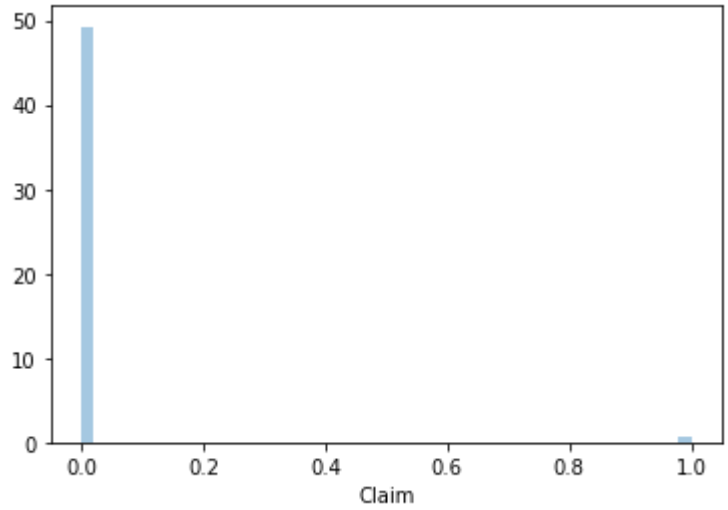
```
plt.figure()
sns.heatmap(df_num.corr(),annot=True)
plt.show()
```



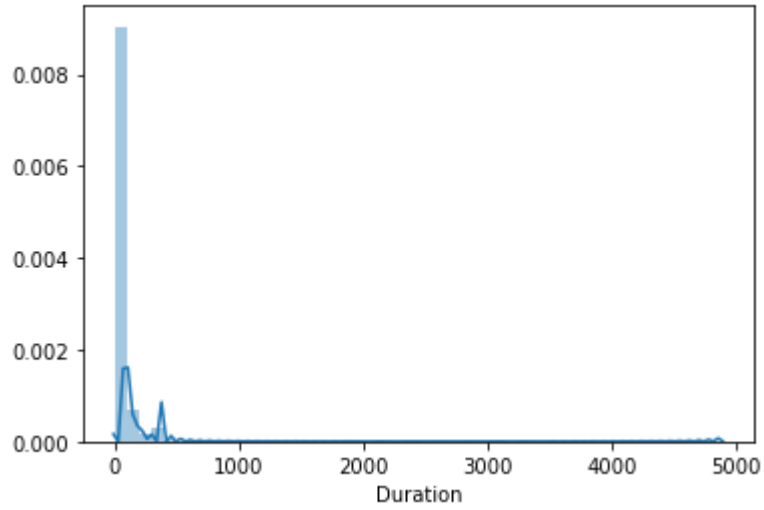
Here we clearly see that Commision and Net Values are corelated

In [24]:

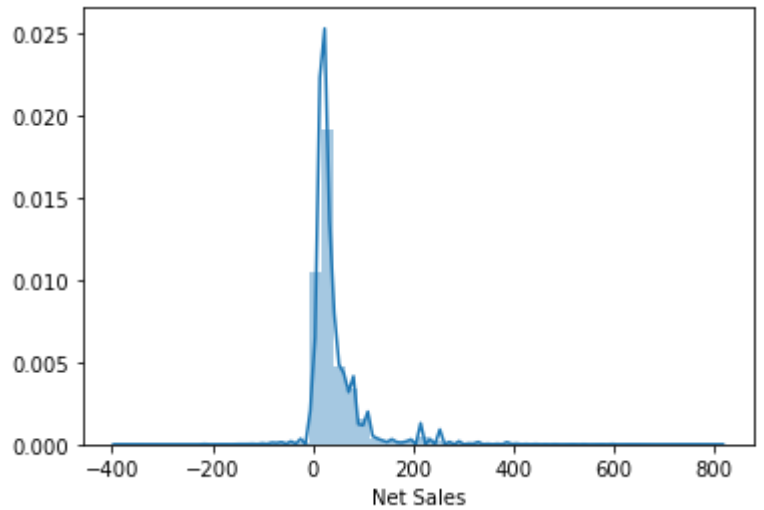
```
for col in df_num:
    plt.figure()
    sns.distplot(df_num[col])
    plt.show()
    print(skew(df_num[col]))
```



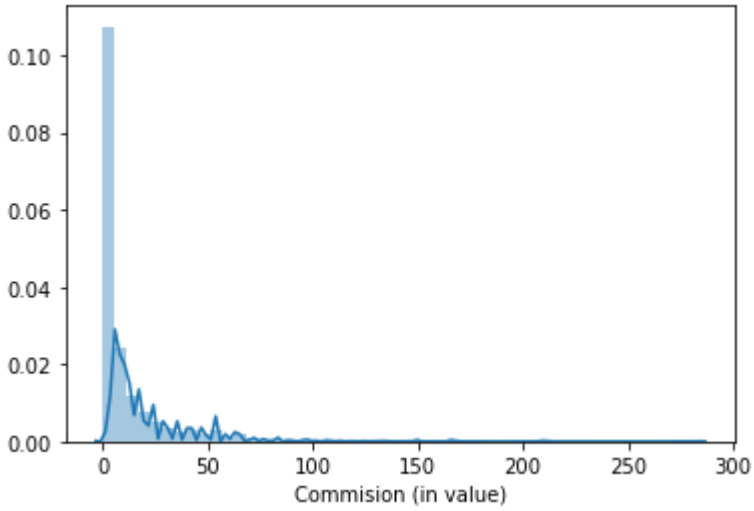

```
8.059932859815197
```



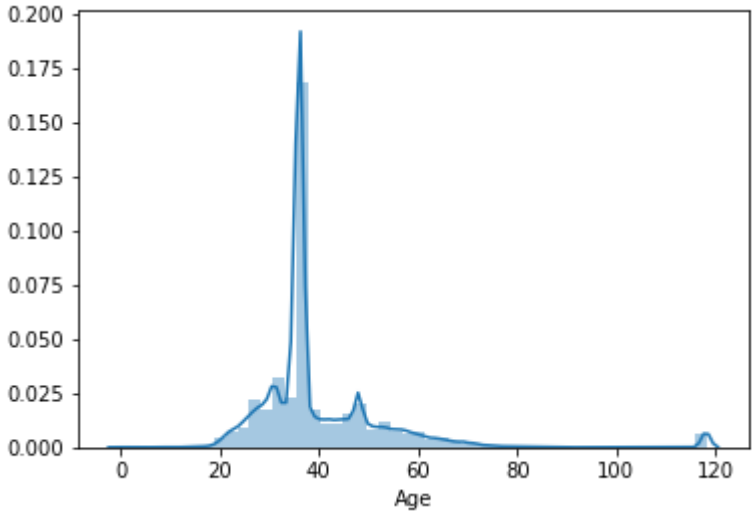

```
22.872106617407425
```




```
3.328045352346187
```




```
4.078052564095901
```




```
2.9746051396188773
```

From the above :

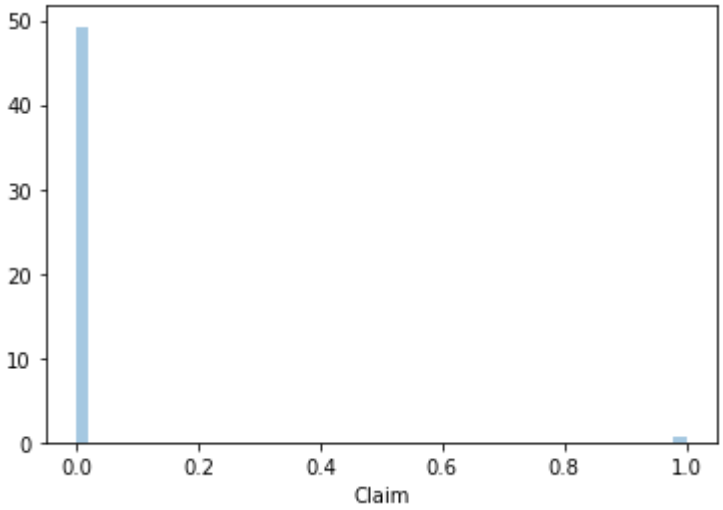
```
1.Commision and Net Values are corelated and distributions look same

2.So we can drop any of the feature
   "Commision" or "Net Sales".so that it shouldd not effect the model.
3.Also there won't be a -ve values in Net Sales Amount.Might be outlier
s,
 these should be removed.
```

In [25]:

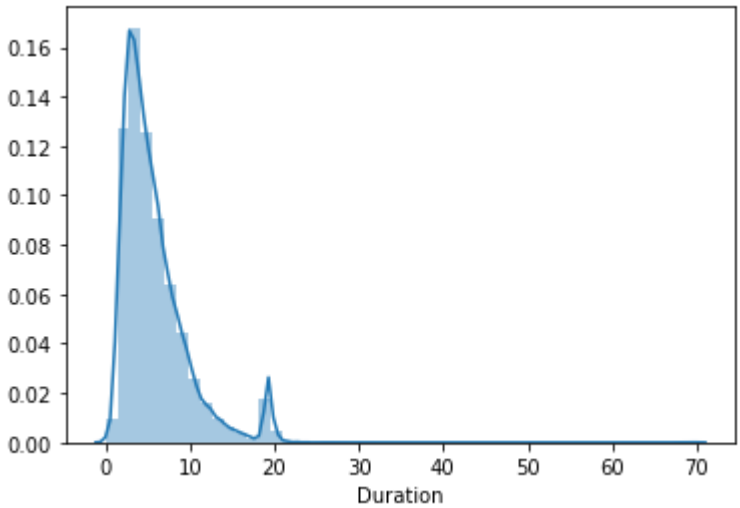
```
for col in df_num:
    if skew(df_num[col]) > 0.5 or skew(df_num[col]) <-0.5:
        df_num[col] = np.sqrt(df_num[col])
```

In [26]:

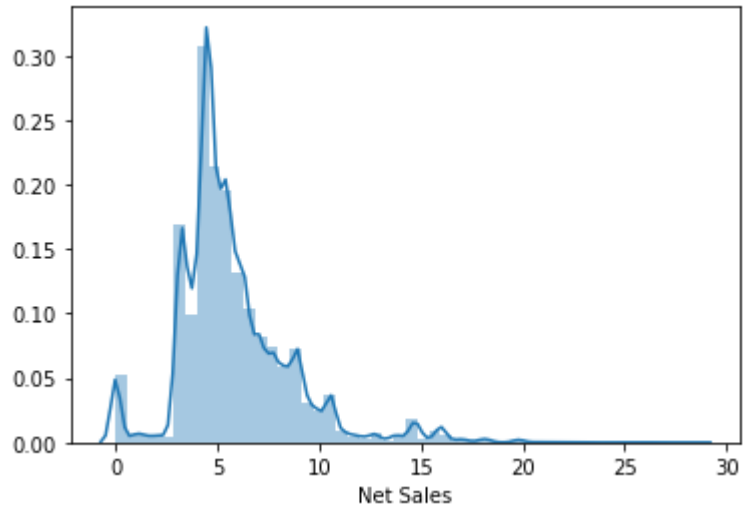
```
for col in df_num:
    plt.figure()
    sns.distplot(df_num[col])
    plt.show()
    print(skew(df_num[col]))
```



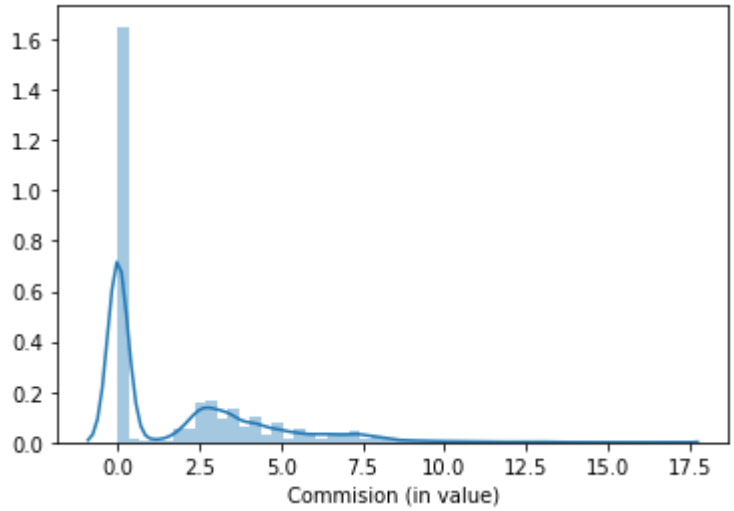
8.059932859815197



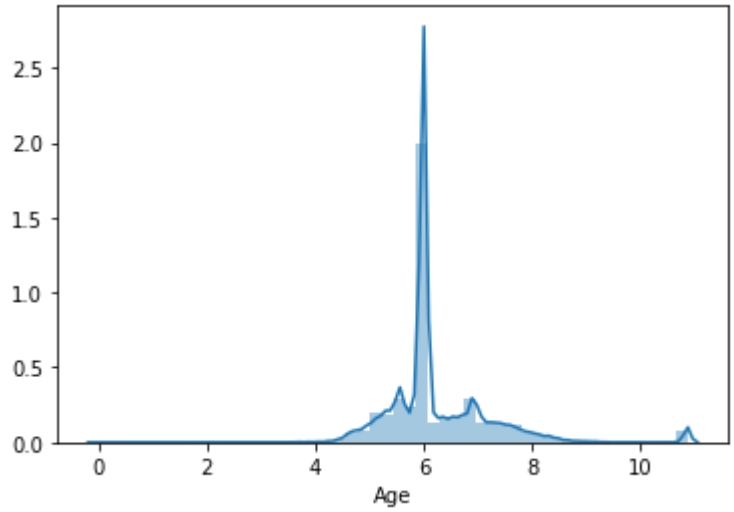
2.4093033561163213



nan



1.3514245184317524



```
1.8845842793921406
```

In [27]:

```
df_num=df_num.drop(["Commision (in value)"],axis=1)
```

In [28]:

```
df_num = df_num[df_num["Net Sales"]>0]
```

In [29]:

```
df_num["Net Sales"].describe()
```

Out[29]:

```
count    48536.000000
mean         5.984404
std          2.708162
min          0.264575
25%          4.449719
50%          5.291503
75%          7.035624
max         28.460499
Name: Net Sales, dtype: float64
```

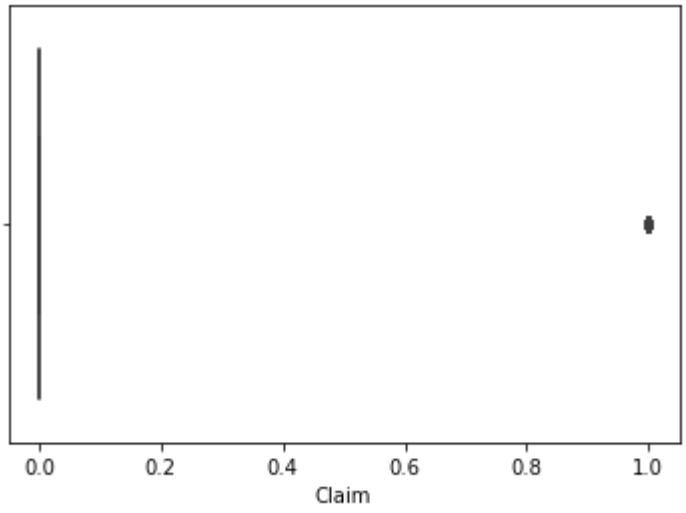
In [30]:

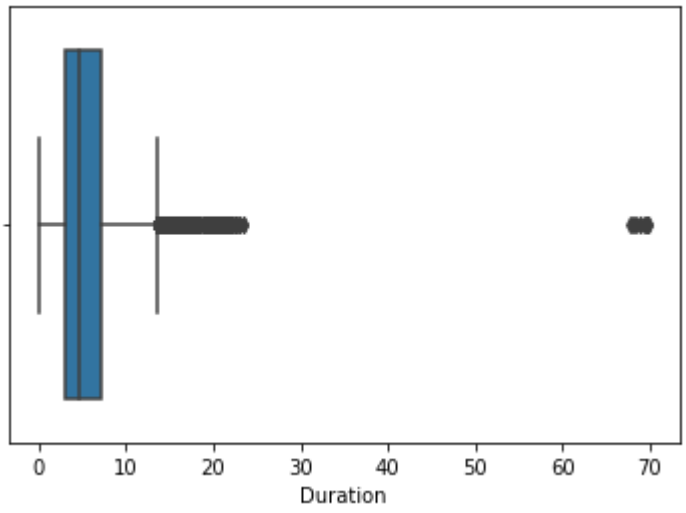
```
df_num.info()
```

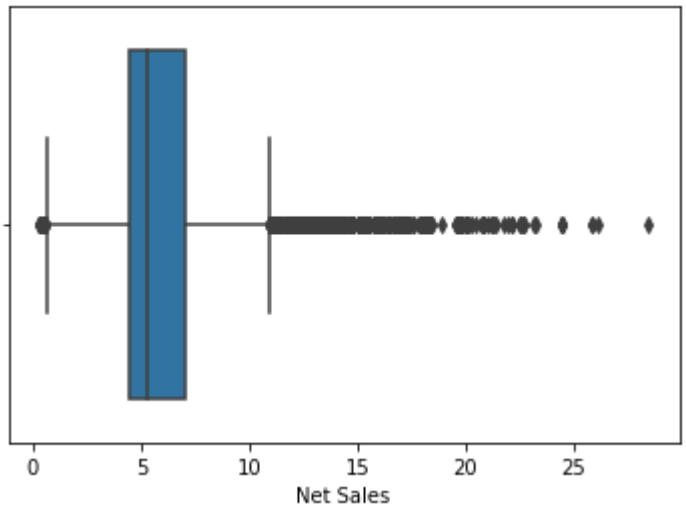
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48536 entries, 1 to 50552
Data columns (total 4 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Claim      48536 non-null  float64
 1   Duration   48536 non-null  float64
 2   Net Sales  48536 non-null  float64
 3   Age        48536 non-null  float64
dtypes: float64(4)
memory usage: 1.9 MB
```

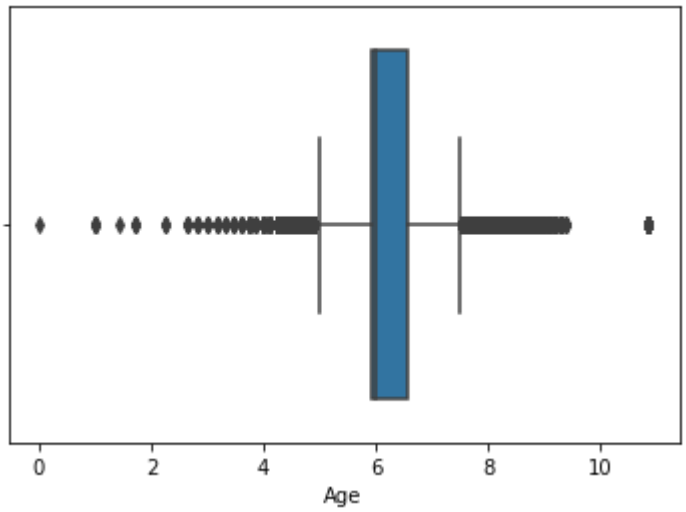
In [31]:

```
for col in df_num:
    plt.figure()
    sns.boxplot(df_num[col])
    plt.show()
```









Categorical Columns

The nunique() function in Pandas returns a series with a number of distinct observations in a column. Similarly, unique() function of pandas returns the list of unique values in the dataset.

In [32]:

```
print(df_cat["Agency Type"].unique())
```

```
['Travel Agency' 'Airlines']
```

In [33]:

```
print(df_cat["Product Name"].unique())
```

```
['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']
```

In [34]:

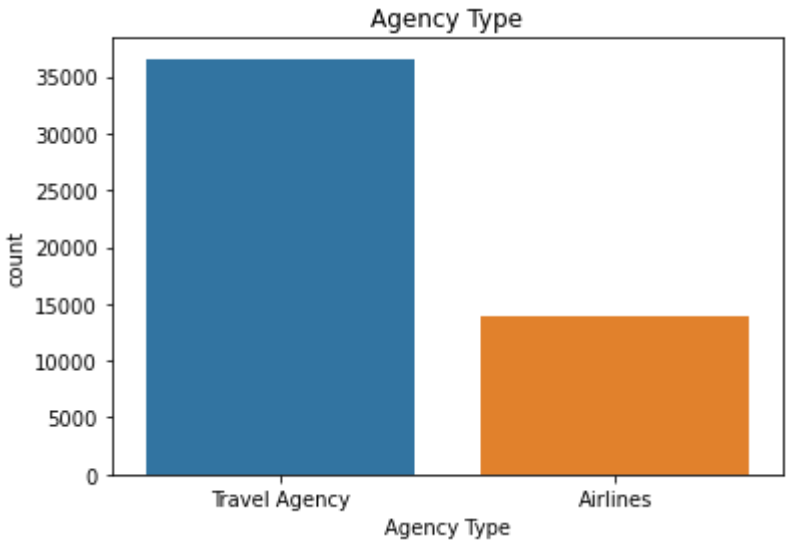
```
print(df_cat["Distribution Channel"].unique())
```

```
['Online' 'Offline']
```

Checking distribution for Agency Type in dataset

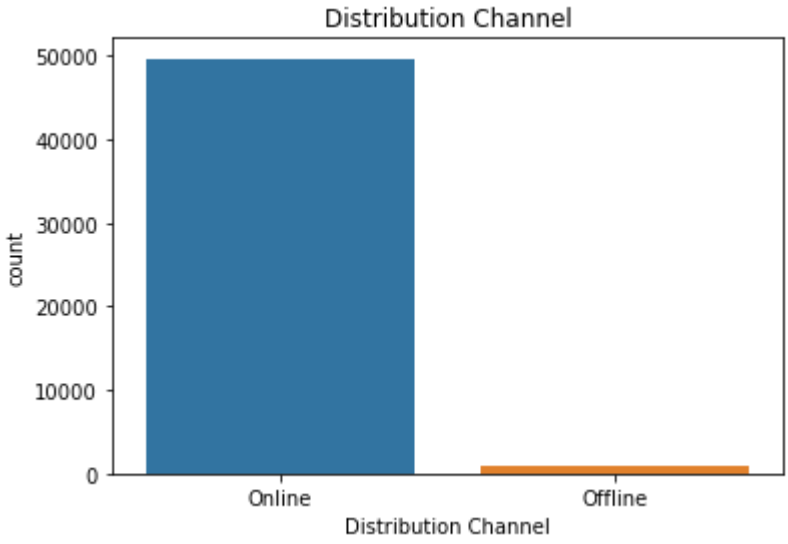
In [35]:

```
plt.figure()
sns.countplot(df_cat["Agency Type"])
plt.title("Agency Type")
plt.show()
```



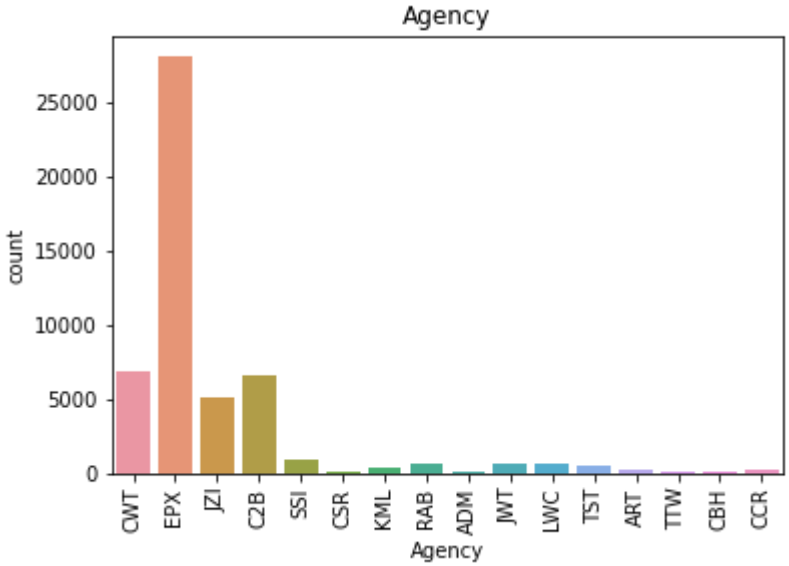
In [36]:

```
plt.figure()
sns.countplot(df_cat["Distribution Channel"])
plt.title("Distribution Channel")
plt.show()
```



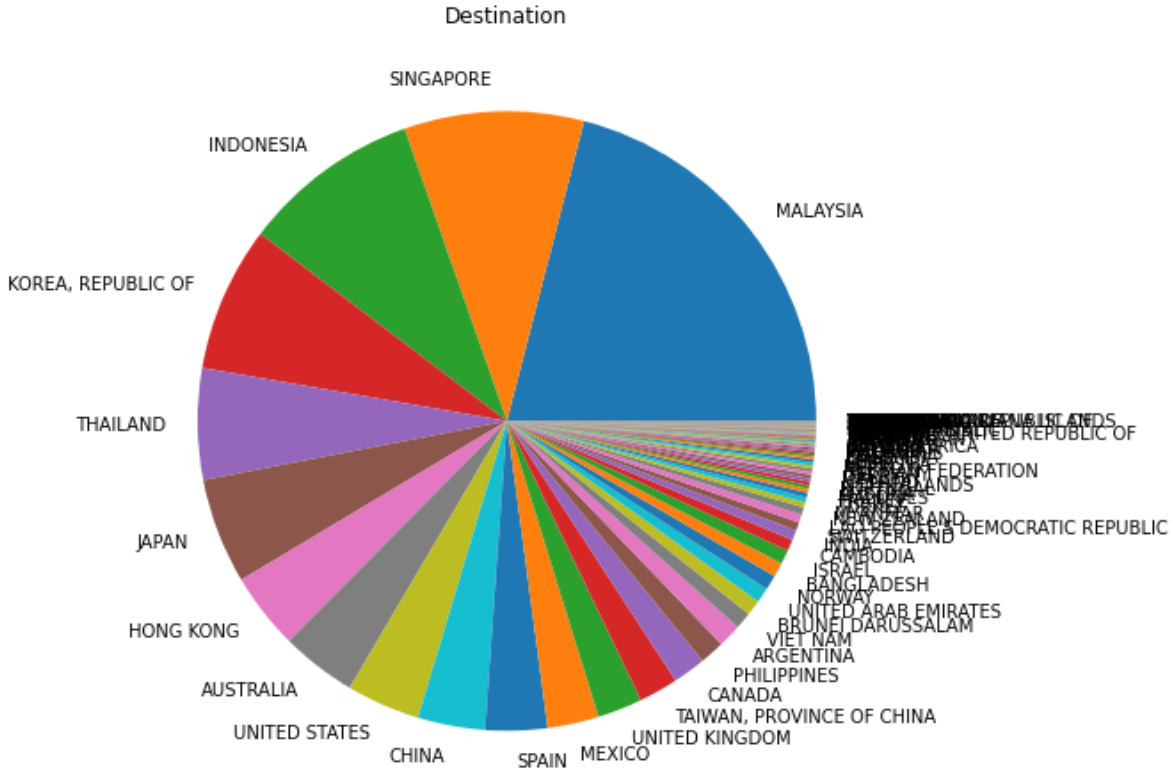
In [37]:

```
plt.figure()
sns.countplot(df_cat["Agency"])
plt.xticks(rotation=90)
plt.title("Agency")
plt.show()
```



In [38]:

```
plt.figure(figsize=(8,8))
plt.pie(df_cat["Destination"].value_counts(),labels=df_cat["Destination"].unique())
plt.title("Destination")
plt.show()
```



Converting Agency,Product Name,Destination ,Agency Type and Distribution channel Feature from categorical to numeric feature using One Hot Encoding

In [39]:

```
print("Claimed")
print(df[df["Claim"] == 1]["Claim"].count())
print("Not Claimed")
print(df[df["Claim"] == 0]["Claim"].count())
```

```
Claimed
741
Not Claimed
49812
```

Here we can see the data set is completely imbalanced. we are trying below methods to get best model,

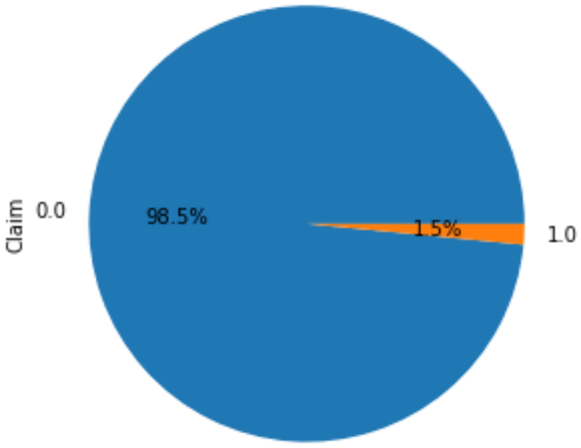
1.changing models like Logistic Regression,SVM,DT,Random Forest and bossting algorithms .

2.oversmpling

3.undersampling

In [40]:

```
plt.figure(figsize=(5,5))
df_num["Claim"].value_counts().plot.pie(autopct="%.1f%%")
plt.show()
```



Here, 1 = Claimed policies

0 = Not Claimed policies

In [41]:

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

In [42]:

```
df_new = df_new.dropna()
```

In [43]:

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

In [44]:

```
y = df_new["Claim"]
```

In [45]:

```
y=y.astype("int64")
```

One Hot Encoding for categorical data

One-hot encoding creates a column for each category in a categorical variable

One-hot encoding in python, panda's get_dummies() function is required.

In [46]:

```
X_dummy = pd.get_dummies(X,columns=["Agency","Product Name","Destination","Distribution Cha
```

In [47]:

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

Filter Method -- Feature Selection

Feature selection keeps the features intact and chooses n best features among them

In [48]:

```
fs = SelectKBest(score_func=chi2,k=5)
X_train_fs=fs.fit_transform(X_train,y_train)
X_test_fs = fs.transform(X_test)
```

StandardScaler

A standard scaler converts a distribution, such that it has 0 mean and 1 standard deviation.

In [49]:

```
ss = StandardScaler()
```

In [50]:

```
X_train_ss = ss.fit_transform(X_train_fs)
```

In [51]:

```
X_test_ss = ss.transform(X_test_fs)
```

undersampling/oversampling

Oversampling and undersampling in data analysis are techniques used to adjust the class distribution of a data set (i.e. the ratio between the different classes/categories represented). These terms are used both in statistical sampling, survey design methodology and in machine learning.

Under Sampling

In [52]:

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

In [53]:

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

In [54]:

```
X_sample1, y_sample1 = rus.fit_resample(X_train_ss,y_train)
```

In [55]:

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

Out[55]:

```
1    524
0    524
Name: Claim, dtype: int64
```

Over sampling

In [56]:

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

In [57]:

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

In [58]:

```
X_sample2 ,y_sample2 = ros.fit_resample(X_train_ss,y_train)
```

In [59]:

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

Out[59]:

```
1    33451
0    33451
Name: Claim, dtype: int64
```

Imbalanced dataset appears here. Oversample method will be performed to deal with imblancing problems.

Modeling Process

Model 1-->LogisticRegression

logistic regression is similar to linear regression – but the latter is not used to predict continuous values (such as age or height). Instead, it's used to predict binary classes – has the client churned or not, has the person survived or not, or is the disease malignant or benign. To simplify, logistic regression is used to predict the Yes/No type of response.

LogisticRegression With Undersampling

In [60]:

```
lr = LogisticRegression()
```

In [61]:

```
score = cross_val_score(lr,X_sample1,y_sample1,cv=5)
```

In [62]:

```
score
```

Out[62]:

```
array([0.74285714, 0.71428571, 0.72857143, 0.76076555, 0.80382775])
```

In [63]:

```
np.mean(score)
```

Out[63]:

```
0.7500615174299385
```

In [64]:

```
lr.fit(X_sample1,y_sample1)
```

Out[64]:

```
LogisticRegression()
```

In [65]:

```
y_pred = lr.predict(X_test_ss)
```

In [66]:

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

```
              precision    recall  f1-score   support

           0       0.99      0.82      0.90     14344
           1       0.05      0.69      0.10       217

    accuracy                           0.82     14561
   macro avg       0.52      0.75      0.50     14561
weighted avg       0.98      0.82      0.89     14561
```

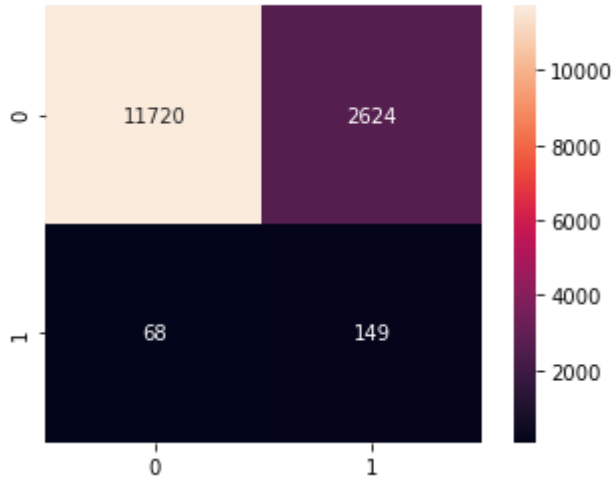
here,We observed that model classification accuracy is 82% .

In [67]:

```
mat = confusion_matrix(y_test,y_pred)
sns.heatmap(mat,square=True,fmt="d",annot=True)
```

Out[67]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21355831a60>
```



roc_auc_curve

The Receiver Operator Characteristic (ROC) curve is an evaluation metric for binary classification problems. It is a probability curve that plots the TPR against FPR at various threshold values and essentially separates the 'signal' from the 'noise'. The Area Under the Curve (AUC) is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

In [68]:

```
y_pred
```

Out[68]:

```
array([0, 0, 1, ..., 1, 0, 1], dtype=int64)
```

In [69]:

```
# we tring to increses a true positive rate (recall) & presion value will reduce
```

In [70]:

```
# to get probabilty value of y_pred
```

In [71]:

```
prob = lr.predict_proba(X_test_ss)[:,-1]
```

In [72]:

```
thresholds = [0.5,0.4,0.3,0.2,0.1]
```

In [73]:

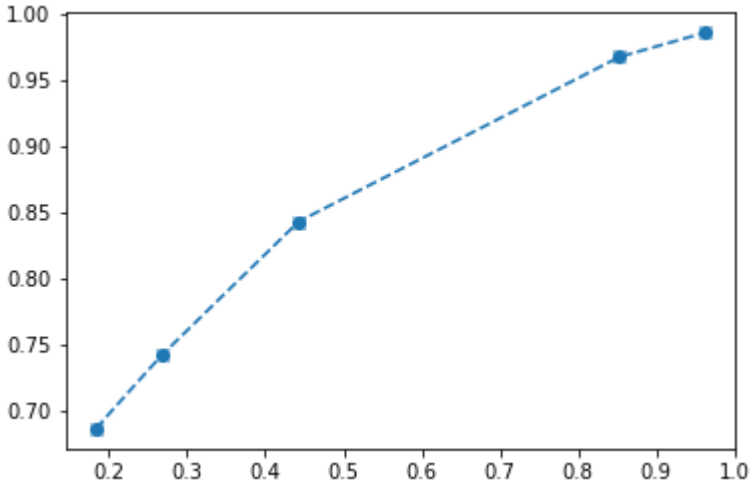
```
tprs = []
fprs = []
```

In [74]:

```
for th in thresholds:
    y_pred = np.where(prob >=th,1,0)
    tn , fp , fn , tp = confusion_matrix(y_test,y_pred).ravel()

    tpr = tp/(tp+fn)
    fpr = fp/(fp+tn)

    tprs.append(tpr)
    fprs.append(fpr)
```

In [75]:

```
plt.figure()
plt.plot(fprs,tprs,"x--")
plt.scatter(fprs,tprs)
plt.show()
```



In [76]:

```
y_pred = np.where(prob>=0.3,1,0)
```

In [77]:

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

```
              precision    recall  f1-score   support

           0       1.00      0.56      0.71     14344
           1       0.03      0.84      0.05       217

    accuracy                           0.56     14561
   macro avg       0.51      0.70      0.38     14561
weighted avg       0.98      0.56      0.70     14561
```

here,We observed that model classification accuracy is 56%.

In [78]:

```
scores = lr.predict_proba(X_test_ss)
```

In [79]:

```
print(scores)
```

```
[[0.81967391 0.18032609]
 [0.80541069 0.19458931]
 [0.16860523 0.83139477]
 ...
 [0.45198266 0.54801734]
 [0.69424709 0.30575291]
 [0.14169705 0.85830295]]
```

In [80]:

```
y_test.value_counts()
```

Out[80]:

```
0    14344
1      217
Name: Claim, dtype: int64
```

In [81]:

```
roc_auc_score(y_test,y_pred)
```

Out[81]:

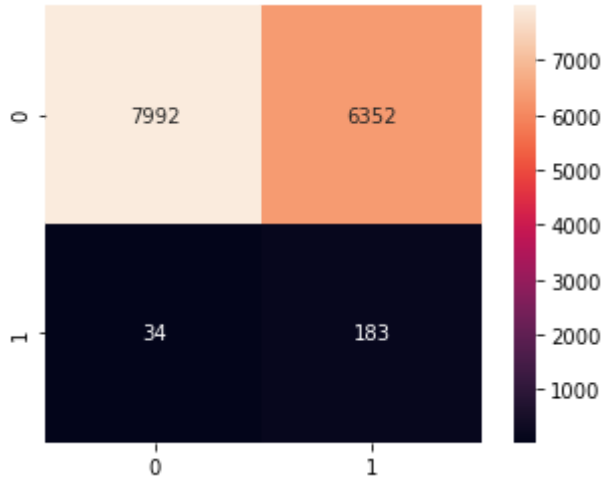
```
0.700242365985489
```

In [82]:

```
mat = confusion_matrix(y_test,y_pred)
sns.heatmap(mat,square=True,fmt="d",annot=True)
```

Out[82]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21356adb310>
```



LogisticRegression with Oversampling

In [83]:

```
lr1 = LogisticRegression()
```

In [84]:

```
lr1.fit(X_sample2,y_sample2)
```

Out[84]:

```
LogisticRegression()
```

In [85]:

```
score =cross_val_score(lr1,X_sample2,y_sample2,cv=5)
```

In [86]:

```
score
```

Out[86]:

```
array([0.74912189, 0.74740303, 0.74484305, 0.75201794, 0.74865471])
```

In [87]:

```
np.mean(score)
```

Out[87]:

```
0.7484081236932226
```

In [88]:

```
y_pred1 = lr1.predict(X_test_ss)
```

In [89]:

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

```
              precision    recall  f1-score   support

           0       0.99      0.83      0.90     14344
           1       0.06      0.67      0.10       217

    accuracy                           0.82     14561
   macro avg       0.52      0.75      0.50     14561
weighted avg       0.98      0.82      0.89     14561
```

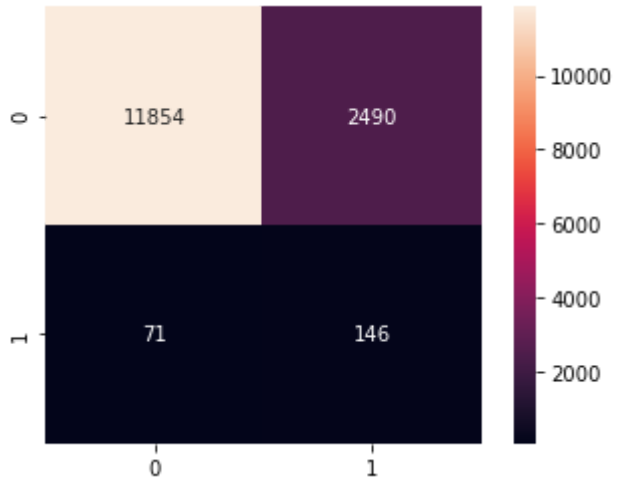
here,We observed that model classification accuracy is 82%

In [90]:

```
mat1 = confusion_matrix(y_test,y_pred1)
sns.heatmap(mat1,square=True,fmt="d",annot=True)
```

Out[90]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2135571c4f0>
```



roc_auc_score

In [91]:

```
scores = lr1.predict_proba(X_test_ss)
```

In [92]:

```
print(scores)
```

```
[[0.81503514 0.18496486]
 [0.79168539 0.20831461]
 [0.17564943 0.82435057]
 ...
 [0.46827817 0.53172183]
 [0.68967358 0.31032642]
 [0.13437405 0.86562595]]
```

In [93]:

```
roc_auc_score(y_test,y_pred1)
```

Out[93]:

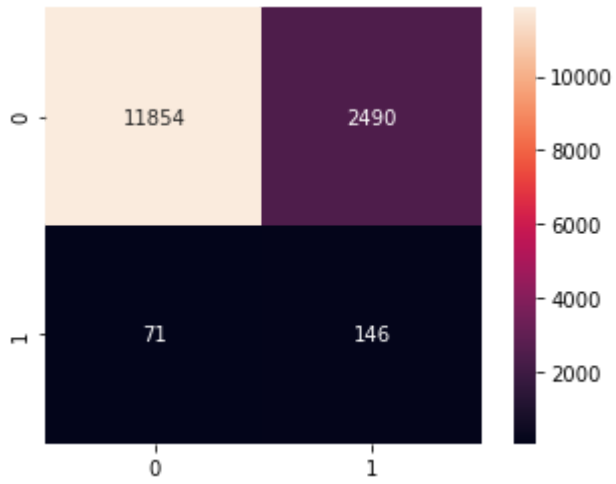
```
0.7496096571150994
```

In [94]:

```
mat1 = confusion_matrix(y_test,y_pred1)
sns.heatmap(mat1,square=True,fmt="d",annot=True)
```

Out[94]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21356ad6e50>
```



Model 2---> DecisionTreeClassifier

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

In [95]:

```
#### DT Classifier with undersampling (criterion="gini")
```

In [96]:

```
dt = DecisionTreeClassifier(max_depth=25,min_samples_leaf=30)
```

In [97]:

```
##  pruning
```

Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning can be used to set the minimum number of samples required at a leaf node or set the maximum depth of the tree are necessary to avoid this problem

In [98]:

```
dt.fit(X_sample1,y_sample1)
```

Out[98]:

```
DecisionTreeClassifier(max_depth=25, min_samples_leaf=30)
```

In [99]:

```
score = cross_val_score(dt,X_sample1,y_sample1,cv=5)
```

In [100]:

```
score
```

Out[100]:

```
array([0.73809524, 0.7       , 0.71428571, 0.74641148, 0.79904306])
```

In [101]:

```
np.mean(score)
```

Out[101]:

```
0.7395670995670995
```

In [102]:

```
y_pred2 = dt.predict(X_test_ss)
```

In [103]:

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

```
              precision    recall  f1-score   support

           0       0.99      0.74      0.85     14344
           1       0.04      0.72      0.08       217

    accuracy                           0.74     14561
   macro avg       0.52      0.73      0.46     14561
weighted avg       0.98      0.74      0.84     14561
```

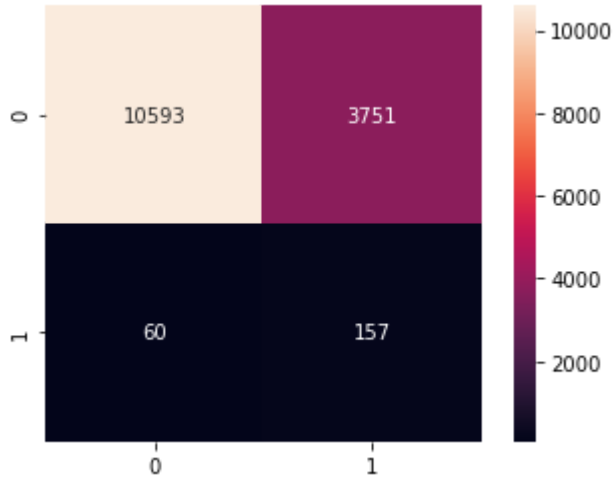
here,We observed that model classification accuracy is 74%

In [104]:

```
mat2 = confusion_matrix(y_test,y_pred2)
sns.heatmap(mat2,square=True,fmt="d",annot=True)
```

Out[104]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x213575e2d90>
```



roc_auc_score

In [105]:

```
scores = dt.predict_proba(X_test_ss)
```

In [106]:

```
print(scores)
```

```
[[0.93939394 0.06060606]
 [0.58064516 0.41935484]
 [0.125      0.875     ]
 ...
 [0.24390244 0.75609756]
 [0.38709677 0.61290323]
 [0.26470588 0.73529412]]
```

In [107]:

```
roc_auc_score(y_test,y_pred2)
```

Out[107]:

```
0.7309996183314014
```

DT with oversampling (criterion="gini")

In [108]:

```
dt1 = DecisionTreeClassifier( max_depth=10, min_samples_leaf=35)
```

In [109]:

```
dt1.fit(X_sample2,y_sample2)
```

Out[109]:

```
DecisionTreeClassifier(max_depth=10, min_samples_leaf=35)
```

In [110]:

```
score = cross_val_score(dt1,X_sample2,y_sample2,cv=5)
```

In [111]:

```
score
```

Out[111]:

```
array([0.79590464, 0.78805769, 0.78594918, 0.7961136 , 0.79185351])
```

In [112]:

```
np.mean(score)
```

Out[112]:

```
0.7915757255256405
```

In [113]:

```
y_pred3 = dt1.predict(X_test_ss)
```

In [114]:

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

```
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14344
           1       0.05      0.60      0.10       217

    accuracy                           0.83     14561
   macro avg       0.52      0.72      0.50     14561
weighted avg       0.98      0.83      0.90     14561
```

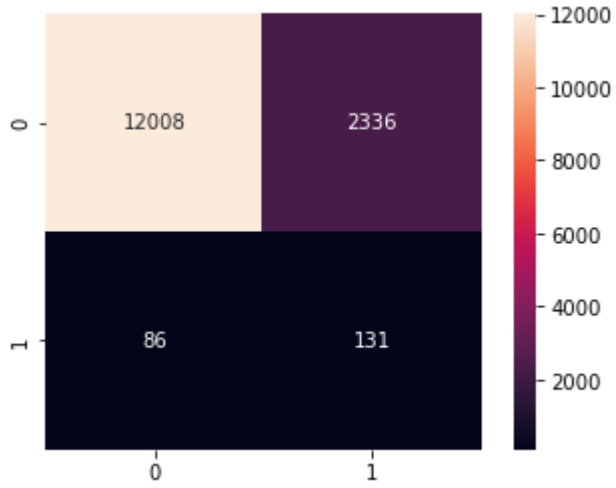
we can see that the model achieved an estimated classification accuracy of about 83%

In [115]:

```
mat3 = confusion_matrix(y_test,y_pred3)
sns.heatmap(mat3,square=True,fmt="d",annot=True)
```

Out[115]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x213580e0790>
```



roc_auc_score

In [116]:

```
scores = dt1.predict_proba(X_test_ss)
```

In [117]:

```
print(scores)
```

```
[[1.         0.        ]
 [1.         0.        ]
 [0.11550985 0.88449015]
 ...
 [0.34298851 0.65701149]
 [0.68635579 0.31364421]
 [0.17163578 0.82836422]]
```

In [118]:

```
roc_auc_score(y_test,y_pred3)
```

Out[118]:

```
0.7204155432930419
```

DecisionTreeClassifier with undersampling (criterion="entropy")

In [119]:

```
dt2 = DecisionTreeClassifier(criterion='entropy', max_depth=25, min_samples_leaf=30)
```

In [120]:

```
dt2.fit(X_sample1,y_sample1)
```

Out[120]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=25, min_samples_leaf=3
0)
```

In [121]:

```
score = cross_val_score(dt2,X_sample1,y_sample1,cv=5)
```

In [122]:

```
score
```

Out[122]:

```
array([0.71428571, 0.7       , 0.71428571, 0.73205742, 0.79904306])
```

In [123]:

```
np.mean(score)
```

Out[123]:

```
0.7319343814080657
```

In [124]:

```
y_pred4 = dt2.predict(X_test_ss)
```

In [125]:

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

```
              precision    recall  f1-score   support

           0       0.99      0.74      0.85     14344
           1       0.04      0.72      0.08       217

    accuracy                           0.74     14561
   macro avg       0.52      0.73      0.46     14561
weighted avg       0.98      0.74      0.84     14561
```

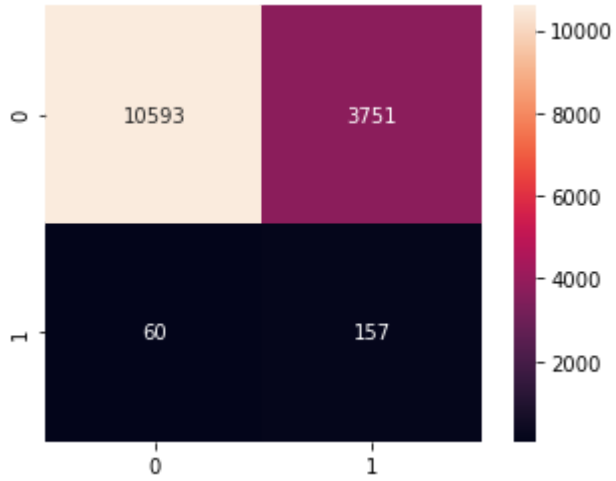
here,We observed that model classification accuracy is 74%.

In [126]:

```
mat4 = confusion_matrix(y_test,y_pred4)
sns.heatmap(mat4,square=True,fmt="d",annot=True)
```

Out[126]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21358174e20>
```



roc_auc_score

In [127]:

```
scores = dt2.predict_proba(X_test_ss)
```

In [128]:

```
print(scores)
```

```
[[0.93939394 0.06060606]
 [0.58064516 0.41935484]
 [0.125      0.875     ]
 ...
 [0.24390244 0.75609756]
 [0.38709677 0.61290323]
 [0.26470588 0.73529412]]
```

In [129]:

```
roc_auc_score(y_test,y_pred4)
```

Out[129]:

```
0.7309996183314014
```

DT with oversampling (criterion="entropy")

In [130]:

```
dt3 = DecisionTreeClassifier(criterion="entropy", max_depth=25, min_samples_leaf=30)
```

In [131]:

```
dt3.fit(X_sample2,y_sample2)
```

Out[131]:

```
DecisionTreeClassifier(criterion='entropy', max_depth=25, min_samples_leaf=3
0)
```

In [132]:

```
score = cross_val_score(dt3,X_sample2,y_sample2,cv=5)
```

In [133]:

```
score
```

Out[133]:

```
array([0.88281892, 0.86196846, 0.88056801, 0.87301943, 0.87047833])
```

In [134]:

```
np.mean(score)
```

Out[134]:

```
0.8737706309807909
```

In [135]:

```
y_pred5 = dt3.predict(X_test_ss)
```

In [136]:

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

```
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     14344
           1       0.04      0.52      0.07       217

    accuracy                           0.81     14561
   macro avg       0.52      0.66      0.48     14561
weighted avg       0.98      0.81      0.88     14561
```

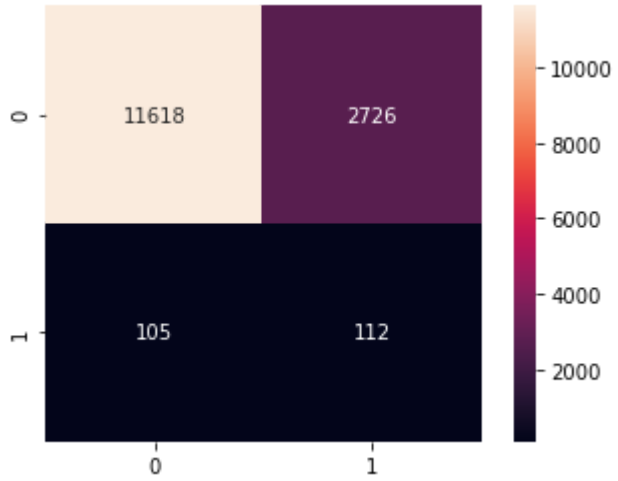
here,We observed that model classification accuracy is 81%.

In [137]:

```
mat5 = confusion_matrix(y_test,y_pred5)
sns.heatmap(mat5,square=True,annot=True,fmt='d')
```

Out[137]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x213558337f0>
```



In [138]:

```
scores = dt3.predict_proba(X_test_ss)
```

In [139]:

```
print(scores)
```

```
[[1.         0.        ]
 [1.         0.        ]
 [0.0890411  0.9109589 ]
 ...
 [0.28089888 0.71910112]
 [1.         0.        ]
 [0.25806452 0.74193548]]
```

In [140]:

```
y_test.value_counts()
```

Out[140]:

```
0    14344
1      217
Name: Claim, dtype: int64
```

In [141]:

```
roc_auc_score(y_test,y_pred5)
```

Out[141]:

```
0.6630422071496681
```

Model 3---> RandomForestClassifier

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

Random Forest using undersampling

In [142]:

```
rfc = RandomForestClassifier(n_estimators=100,random_state=1)
```

In [143]:

```
rfc.fit(X_sample1,y_sample1)
```

Out[143]:

```
RandomForestClassifier(random_state=1)
```

In [144]:

```
score = cross_val_score(rfc,X_sample1,y_sample1,cv=5)
```

In [145]:

```
score
```

Out[145]:

```
array([0.71904762, 0.66190476, 0.66190476, 0.68421053, 0.72727273])
```

In [146]:

```
np.mean(score)
```

Out[146]:

```
0.690868079289132
```

In [147]:

```
y_pred6 = rfc.predict(X_test_ss)
```

In [148]:

```
print(classification_report(y_test,y_pred6))
```

```
              precision    recall  f1-score   support

           0       0.99      0.72      0.83     14344
           1       0.04      0.68      0.07       217

    accuracy                           0.72     14561
   macro avg       0.51      0.70      0.45     14561
weighted avg       0.98      0.72      0.82     14561
```

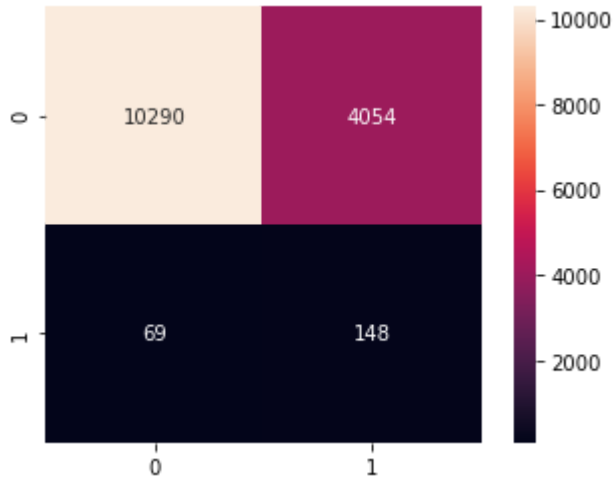
here,We observed that model classification accuracy is 72%.

In [149]:

```
mat6 = confusion_matrix(y_test,y_pred6)
sns.heatmap(mat6,square=True,annot=True,fmt='d')
```

Out[149]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2135688eb50>
```



roc_auc_score

In [150]:

```
scores = rfc.predict_proba(X_test_ss)
```

In [151]:

```
print(scores)
```

```
[[0.99       0.01      ]
 [0.2        0.8       ]
 [0.45453968 0.54546032]
 ...
 [0.43       0.57      ]
 [0.64       0.36      ]
 [0.37       0.63      ]]
```

In [152]:

```
roc_auc_score(y_test,y_pred6)
```

Out[152]:

```
0.6997003837247258
```

Random Forest using oversampling

In [153]:

```
rfc1 = RandomForestClassifier(n_estimators=100,random_state=1)
```

In [154]:

```
rfc1.fit(X_sample2,y_sample2)
```

Out[154]:

```
RandomForestClassifier(random_state=1)
```

In [155]:

```
score = cross_val_score(rfc1,X_sample2,y_sample2,cv=5)
```

In [156]:

```
score
```

Out[156]:

```
array([0.96674389, 0.96906061, 0.96696562, 0.96651719, 0.96599402])
```

In [157]:

```
np.mean(score)
```

Out[157]:

```
0.967056266001511
```

In [158]:

```
y_pred7 = rfc1.predict(X_test_ss)
```

In [159]:

```
print(classification_report(y_test,y_pred7))
```

```
              precision    recall  f1-score   support

           0       0.99      0.94      0.96     14344
           1       0.06      0.24      0.09       217

    accuracy                           0.93     14561
   macro avg       0.52      0.59      0.53     14561
weighted avg       0.97      0.93      0.95     14561
```

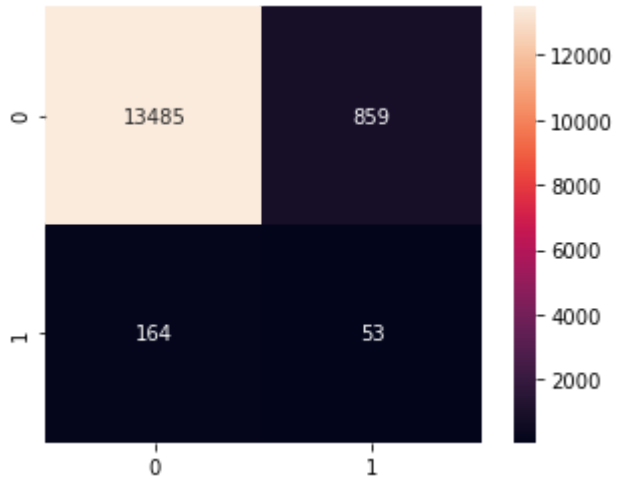
here,We observed that model classification accuracy is 93%.

In [160]:

```
mat7 = confusion_matrix(y_test,y_pred7)
sns.heatmap(mat7,square=True,annot=True,fmt='d')
```

Out[160]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x213556cbc70>
```



roc_auc_score

In [161]:

```
scores = rfc1.predict_proba(X_test_ss)
```

In [162]:

```
print(scores)
```

```
[[1.         0.        ]
 [1.         0.        ]
 [0.08873261 0.91126739]
 ...
 [0.93101158 0.06898842]
 [1.         0.        ]
 [0.99       0.01      ]]
```

In [163]:

```
roc_auc_score(y_test,y_pred7)
```

Out[163]:

```
0.5921769824278235
```

Model 4---> Gradient Boosting

Gradient Boosting is a popular boosting algorithm. In gradient boosting, each predictor corrects its predecessor's error. In contrast to Adaboost, the weights of the training instances are not tweaked, instead, each predictor is trained using the residual errors of predecessor as labels.

Gradient boosting using Undersampling

In [164]:

```
gb = GradientBoostingClassifier()
```

In [165]:

```
gb.fit(X_sample1,y_sample1)
```

Out[165]:

```
GradientBoostingClassifier()
```

In [166]:

```
score = cross_val_score(gb,X_sample1,y_sample1,cv=5)
```

In [167]:

```
score
```

Out[167]:

```
array([0.73333333, 0.69047619, 0.73333333, 0.73205742, 0.77511962])
```

In [168]:

```
np.mean(score)
```

Out[168]:

```
0.732863978127136
```

In [169]:

```
y_pred8 = gb.predict(X_test_ss)
```

In [170]:

```
print(classification_report(y_test,y_pred8))
```

```
              precision    recall  f1-score   support

           0       0.99      0.79      0.88     14344
           1       0.05      0.68      0.09       217

    accuracy                           0.79     14561
   macro avg       0.52      0.73      0.48     14561
weighted avg       0.98      0.79      0.87     14561
```

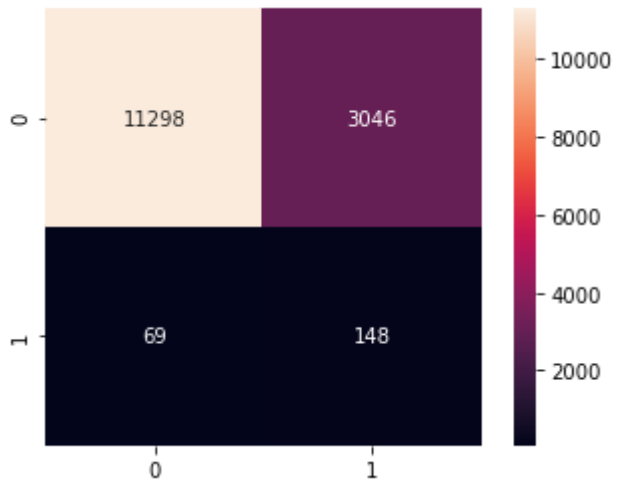
here,We observed that model classification accuracy is 79%.

In [171]:

```
mat8 = confusion_matrix(y_test,y_pred8)
sns.heatmap(mat8,square=True,annot=True,fmt="d")
```

Out[171]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21356955640>
```



roc_auc_score

In [172]:

```
scores = gb.predict_proba(X_test_ss)
```

In [173]:

```
print(scores)
```

```
[[0.90891763 0.09108237]
 [0.68728215 0.31271785]
 [0.20750994 0.79249006]
 ...
 [0.51371832 0.48628168]
 [0.66455812 0.33544188]
 [0.14183155 0.85816845]]
```

In [174]:

```
roc_auc_score(y_test,y_pred8)
```

Out[174]:

```
0.7348370262233315
```

Gradient boosting using oversampling

In [175]:

```
gb1 = GradientBoostingClassifier()
```

In [176]:

```
gb1.fit(X_sample2,y_sample2)
```

Out[176]:

```
GradientBoostingClassifier()
```

In [177]:

```
score = cross_val_score(gb1,X_sample2,y_sample2,cv=5)
```

In [178]:

```
score
```

Out[178]:

```
array([0.77535311, 0.76892609, 0.77518685, 0.7793722 , 0.77892377])
```

In [179]:

```
np.mean(score)
```

Out[179]:

```
0.7755524024035598
```

In [180]:

```
y_pred9 = gb1.predict(X_test_ss)
```

In [181]:

```
print(classification_report(y_test,y_pred9))
```

```
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14344
           1       0.06      0.62      0.10       217

    accuracy                           0.84     14561
   macro avg       0.52      0.73      0.51     14561
weighted avg       0.98      0.84      0.90     14561
```

here,We observed that model classification accuracy is 84%.

In [182]:

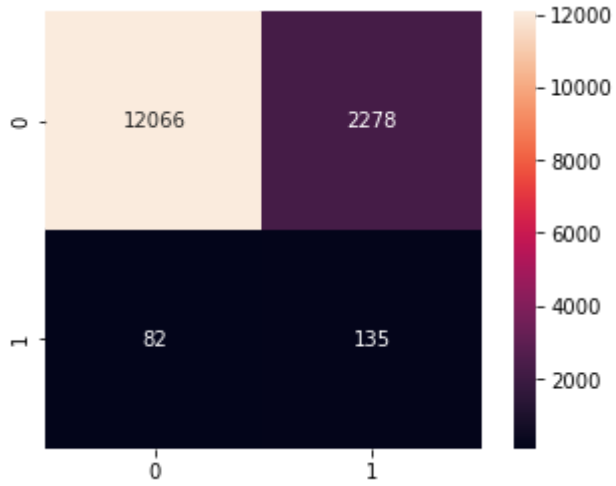
```
mat9 =confusion_matrix(y_test,y_pred9)
```

In [183]:

```
sns.heatmap(mat9,square=True,annot=True,fmt='d')
```

Out[183]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21357198430>
```



roc_auc_score

In [184]:

```
scores = gb1.predict_proba(X_test_ss)
```

In [185]:

```
print(scores)
```

```
[[0.93821629 0.06178371]
 [0.79911765 0.20088235]
 [0.12283619 0.87716381]
 ...
 [0.52734187 0.47265813]
 [0.64333975 0.35666025]
 [0.12498626 0.87501374]]
```

In [186]:

```
roc_auc_score(y_test,y_pred9)
```

Out[186]:

```
0.731653884409673
```

Model 5 ---> Adaboost Classifier

AdaBoost was the first really successful boosting algorithm developed for the purpose of binary classification. AdaBoost is short for Adaptive Boosting and is a very popular boosting technique which combines multiple “weak classifiers” into a single “strong classifier”

Adaboost classifier unsing undersampling

In [187]:

```
aba = AdaBoostClassifier(n_estimators=100)
```

In [188]:

```
aba.fit(X_sample1,y_sample1)
```

Out[188]:

```
AdaBoostClassifier(n_estimators=100)
```

In [189]:

```
score = cross_val_score(aba,X_sample1,y_sample1,cv=5)
```

In [190]:

```
score
```

Out[190]:

```
array([0.73809524, 0.6952381 , 0.72380952, 0.73684211, 0.77511962])
```

In [191]:

```
np.mean(score)
```

Out[191]:

```
0.733820915926179
```

In [192]:

```
y_pred10 = aba.predict(X_test_ss)
```

In [193]:

```
print(classification_report(y_test,y_pred10))
```

```
              precision    recall  f1-score   support

           0       0.99      0.78      0.87     14344
           1       0.04      0.69      0.08       217

    accuracy                           0.78     14561
   macro avg       0.52      0.73      0.48     14561
weighted avg       0.98      0.78      0.86     14561
```

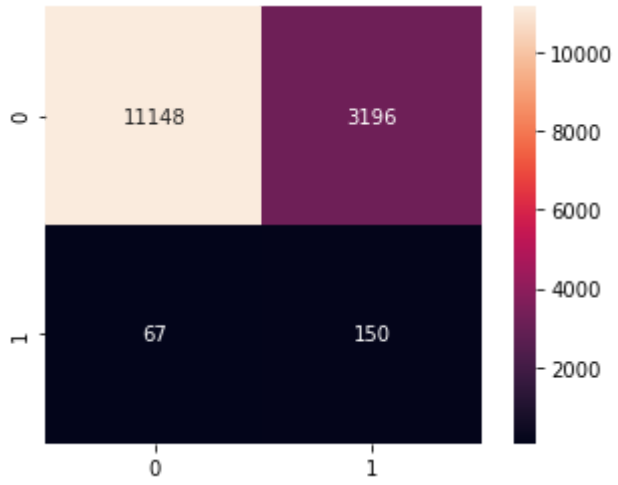
we can see that the model achieved an estimated classification accuracy of about 78%.

In [194]:

```
mat10 = confusion_matrix(y_test,y_pred10)
sns.heatmap(mat10 ,square=True,annot=True,fmt='d')
```

Out[194]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2135721cee0>
```



roc_auc_score

In [195]:

```
scores = aba.predict_proba(X_test_ss)
```

In [196]:

```
print(scores)
```

```
[[0.50623058 0.49376942]
 [0.50432542 0.49567458]
 [0.49715241 0.50284759]
 ...
 [0.49922692 0.50077308]
 [0.50097437 0.49902563]
 [0.496382   0.503618  ]]
```

In [197]:

```
roc_auc_score(y_test,y_pred10)
```

Out[197]:

```
0.7342166541157239
```

Adaboost classifier unsing oversampling

In [198]:

```
aba1 = AdaBoostClassifier(n_estimators=100)
```

In [199]:

```
aba1.fit(X_sample2,y_sample2)
```

Out[199]:

```
AdaBoostClassifier(n_estimators=100)
```

In [200]:

```
score = cross_val_score(aba1,X_sample2,y_sample2,cv=5)
```

In [201]:

```
score
```

Out[201]:

```
array([0.76376952, 0.75883716, 0.75358744, 0.76240658, 0.76180867])
```

In [202]:

```
np.mean(score)
```

Out[202]:

```
0.7600818743395947
```

In [203]:

```
y_pred11 = aba1.predict(X_test_ss)
```

In [204]:

```
print(classification_report(y_test,y_pred11))
```

```
              precision    recall  f1-score   support

           0       0.99      0.80      0.89     14344
           1       0.05      0.66      0.09       217

    accuracy                           0.80     14561
   macro avg       0.52      0.73      0.49     14561
weighted avg       0.98      0.80      0.87     14561
```

here,We observed that model classification accuracy is 80%.

In [205]:

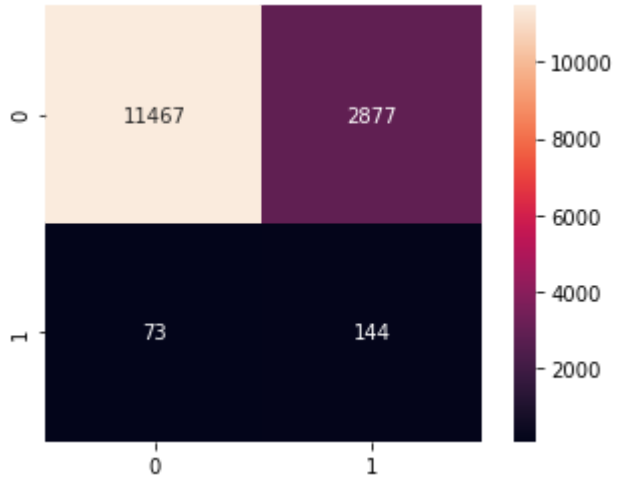
```
mat11 = confusion_matrix(y_test,y_pred11)
```

In [206]:

```
sns.heatmap(mat11,square=True,annot=True,fmt="d")
```

Out[206]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2135887a160>
```



roc_auc_score

In [207]:

```
scores = aba1.predict_proba(X_test_ss)
```

In [208]:

```
print(scores)
```

```
[[0.50705755 0.49294245]
 [0.50512655 0.49487345]
 [0.49504108 0.50495892]
 ...
 [0.49995975 0.50004025]
 [0.50110857 0.49889143]
 [0.49486754 0.50513246]]
```

In [209]:

```
roc_auc_score(y_test,y_pred11)
```

Out[209]:

```
0.7315114012249377
```

Model 6 ---> SVC

An SVM model is basically a representation of different classes in a hyperplane in multidimensional space. The hyperplane will be generated in an iterative manner by SVM so that the error can be minimized. The goal of SVM is to divide the datasets into classes to find a maximum marginal hyperplane (MMH).

LinearSVC usning Undersampling

In [210]:

```
lsvc = LinearSVC(random_state=1)
```

In [211]:

```
lsvc.fit(X_sample1,y_sample1)
```

Out[211]:

```
LinearSVC(random_state=1)
```

In [212]:

```
score = cross_val_score(lsvc,X_sample1,y_sample1,cv=5)
```

In [213]:

```
score
```

Out[213]:

```
array([0.73809524, 0.7       , 0.73333333, 0.76555024, 0.81339713])
```

In [214]:

```
np.mean(score)
```

Out[214]:

```
0.7500751879699248
```

In [215]:

```
y_pred14 = lsvc.predict(X_test_ss)
```

In [216]:

```
print(classification_report(y_test,y_pred14))
```

```
              precision    recall  f1-score   support

           0       0.99      0.83      0.91     14344
           1       0.06      0.66      0.10       217

    accuracy                           0.83     14561
   macro avg       0.52      0.75      0.50     14561
weighted avg       0.98      0.83      0.89     14561
```

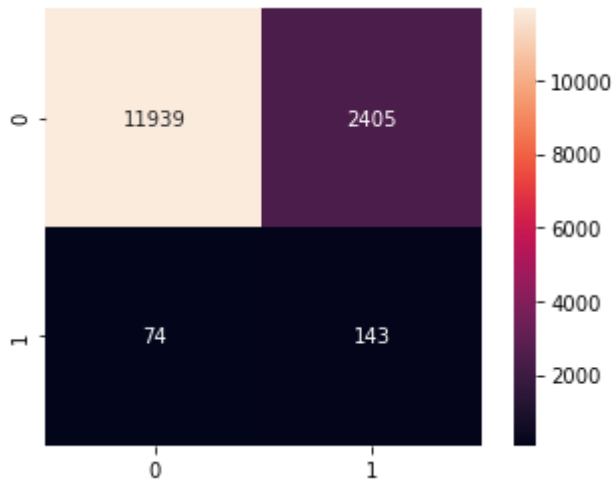
here,We observed that model classification accuracy is 83%.

In [217]:

```
mat14 = confusion_matrix(y_test,y_pred14)
sns.heatmap(mat14 ,square=True,annot=True,fmt='d')
```

Out[217]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2135886d8e0>
```



Soft Margin

In [218]:

```
lsv = LinearSVC(C=0.5,random_state=1)
```

In [219]:

```
# regularization
```

In [220]:

```
lsv.fit(X_sample1,y_sample1)
```

Out[220]:

```
LinearSVC(C=0.5, random_state=1)
```

In [221]:

```
score = cross_val_score(lsv,X_sample1,y_sample1,cv=5)
```

In [222]:

```
score
```

Out[222]:

```
array([0.73809524, 0.7       , 0.73333333, 0.76555024, 0.81339713])
```

In [223]:

```
np.mean(score)
```

Out[223]:

```
0.7500751879699248
```

In [224]:

```
y_pred16 = lsv.predict(X_test_ss)
```

In [225]:

```
print(classification_report(y_test,y_pred16))
```

```
              precision    recall  f1-score   support

           0       0.99      0.83      0.91     14344
           1       0.06      0.66      0.10       217

    accuracy                           0.83     14561
   macro avg       0.52      0.75      0.50     14561
weighted avg       0.98      0.83      0.89     14561
```

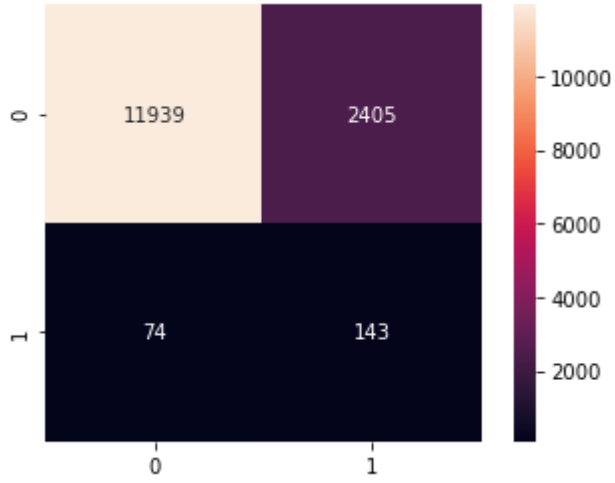
here,We observed that model classification accuracy is 82

In [226]:

```
mat16 = confusion_matrix(y_test,y_pred16)
sns.heatmap(mat16,square=True,annot=True,fmt='d')
```

Out[226]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x213592f0520>
```



LinearSVC using Oversampling

In [227]:

```
lsvc1 = LinearSVC(random_state=1)
```

In [228]:

```
lsvc1.fit(X_sample2,y_sample2)
```

Out[228]:

```
LinearSVC(random_state=1)
```

In [229]:

```
score = cross_val_score(lsvc1,X_sample2,y_sample2,cv=5)
```

In [230]:

```
score
```

Out[230]:

```
array([0.74598311, 0.74501158, 0.7435725 , 0.7509716 , 0.74745889])
```

In [231]:

```
np.mean(score)
```

Out[231]:

```
0.7465995367011364
```

In [232]:

```
y_pred17 = lsvc1.predict(X_test_ss)
```

In [233]:

```
print(classification_report(y_test,y_pred17))
```

```
              precision    recall  f1-score   support

           0       0.99      0.85      0.91     14344
           1       0.06      0.65      0.11       217

    accuracy                           0.84     14561
   macro avg       0.53      0.75      0.51     14561
weighted avg       0.98      0.84      0.90     14561
```

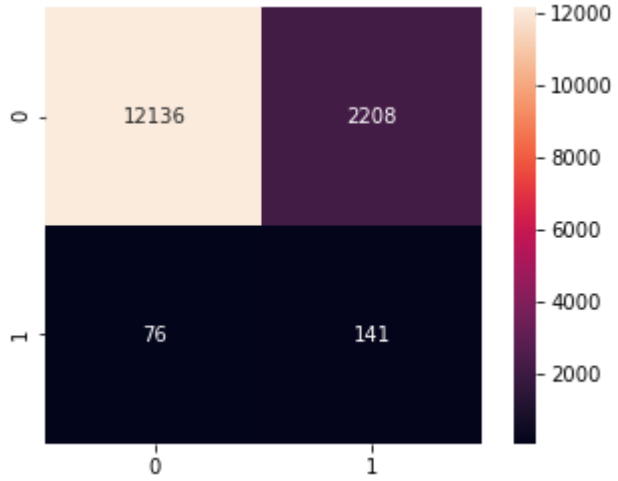
we can see that the model achieved an estimated classification accuracy of about 84%.

In [234]:

```
mat17 = confusion_matrix(y_test,y_pred17)
sns.heatmap(mat17,square=True,annot=True,fmt='d')
```

Out[234]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x21359383910>
```



Soft Margin

In [235]:

```
lsv1 = LinearSVC(C=0.5,random_state=1)
```

In [236]:

```
lsv1.fit(X_sample2,y_sample2)
```

Out[236]:

```
LinearSVC(C=0.5, random_state=1)
```

In [237]:

```
score = cross_val_score(lsv1,X_sample2,y_sample2,cv=5)
```

In [238]:

```
score
```

Out[238]:

```
array([0.74598311, 0.74501158, 0.7435725 , 0.75089686, 0.74753363])
```

In [239]:

```
np.mean(score)
```

Out[239]:

```
0.7465995367011364
```

In [240]:

```
y_pred18 = lsv1.predict(X_test_ss)
```

In [241]:

```
print(classification_report(y_test,y_pred18))
```

```
              precision    recall  f1-score   support

           0       0.99      0.85      0.91     14344
           1       0.06      0.65      0.11       217

    accuracy                           0.84     14561
   macro avg       0.53      0.75      0.51     14561
weighted avg       0.98      0.84      0.90     14561
```

we can see that the model achieved an estimated classification accuracy of about 84%.

In [242]:

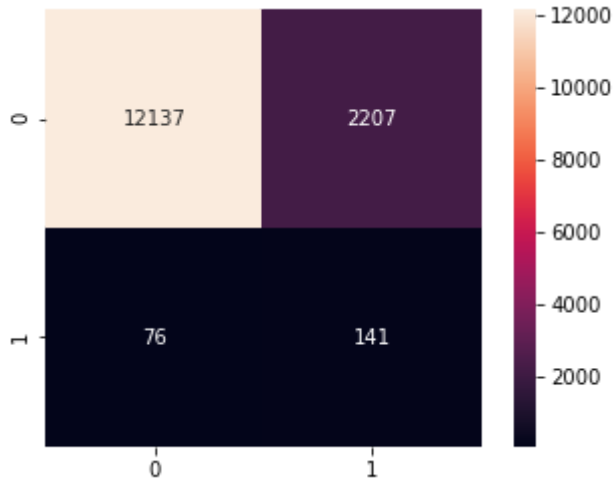
```
mat18 = confusion_matrix(y_test,y_pred18)
```

In [243]:

```
sns.heatmap(mat18,square=True,annot=True,fmt='d')
```

Out[243]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x213593fc370>
```



Ensembling (Undersampling)

nsemble methods combine several trees base algorithms to construct better predictive performance than a single tree base algorithm. The main principle behind the ensemble model is that a group of weak learners come together to form a strong learner, thus increasing the accuracy of the model. When we try to predict the target variable using any machine learning technique, the main causes of difference in actual and predicted values are noise, variance, and bias. Ensemble helps to reduce these factors (except noise, which is irreducible error).

Naive Aggregation

Hard Voting

In [244]:

```
model_list = [("lr",lr),("dt",dt),("dt2",dt2),("rfc",rfc),("gb",gb),("aba",aba),("lsv",lsv)
```

In [245]:

```
vc1 = VotingClassifier(estimators=model_list)
```

In [246]:

```
vc1.fit(X_sample1,y_sample1)
y_pred20 = vc1.predict(X_test_ss)
```

In [247]:

```
print(classification_report(y_test,y_pred20))
```

```
              precision    recall  f1-score   support

           0       0.99      0.79      0.88     14344
           1       0.05      0.70      0.09       217

    accuracy                           0.79     14561
   macro avg       0.52      0.75      0.49     14561
weighted avg       0.98      0.79      0.87     14561
```

Soft Voting

In [248]:

```
vc2 = VotingClassifier(estimators=model_list,voting="soft")
```

In [249]:

```
vc2.fit(X_sample1,y_sample1)
y_pred21 = vc1.predict(X_test_ss)
```

In [250]:

```
print(classification_report(y_test,y_pred21))
```

```
              precision    recall  f1-score   support

           0       0.99      0.79      0.88     14344
           1       0.05      0.70      0.09       217

    accuracy                           0.79     14561
   macro avg       0.52      0.75      0.49     14561
weighted avg       0.98      0.79      0.87     14561
```

In [251]:

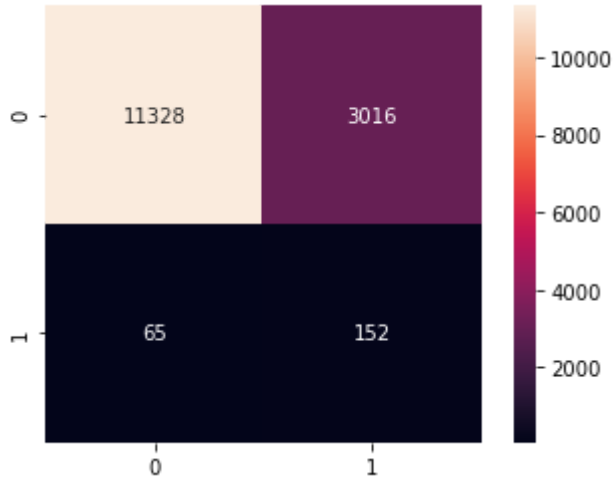
```
mat21 = confusion_matrix(y_test,y_pred21)
```

In [252]:

```
sns.heatmap(mat21,square=True,annot=True,fmt='d')
```

Out[252]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x213594b1610>
```



here,We observed that model classification accuracy is 79%.

Ensembling (oversampling)

Hard voting

In [253]:

```
model_list1 = [("lr1",lr1),("dt1",dt1),("dt3",dt3),("rfc1",rfc1),("gb1",gb1),("aba1",aba1),
```

In [254]:

```
vc3 = VotingClassifier(estimators=model_list1)
```

In [255]:

```
vc3.fit(X_sample2,y_sample2)
y_pred22 = vc1.predict(X_test_ss)
```

In [256]:

```
print(classification_report(y_test,y_pred22))
```

```
              precision    recall  f1-score   support

           0       0.99      0.79      0.88     14344
           1       0.05      0.70      0.09       217

    accuracy                           0.79     14561
   macro avg       0.52      0.75      0.49     14561
weighted avg       0.98      0.79      0.87     14561
```

Soft Voting

In [257]:

```
vc4 = VotingClassifier(estimators=model_list1,voting="soft")
```

In [258]:

```
vc4.fit(X_sample2,y_sample2)
y_pred23 = vc1.predict(X_test_ss)
```

In [259]:

```
print(classification_report(y_test,y_pred23))
```

```
              precision    recall  f1-score   support

           0       0.99      0.79      0.88     14344
           1       0.05      0.70      0.09       217

    accuracy                           0.79     14561
   macro avg       0.52      0.75      0.49     14561
weighted avg       0.98      0.79      0.87     14561
```

In [260]:

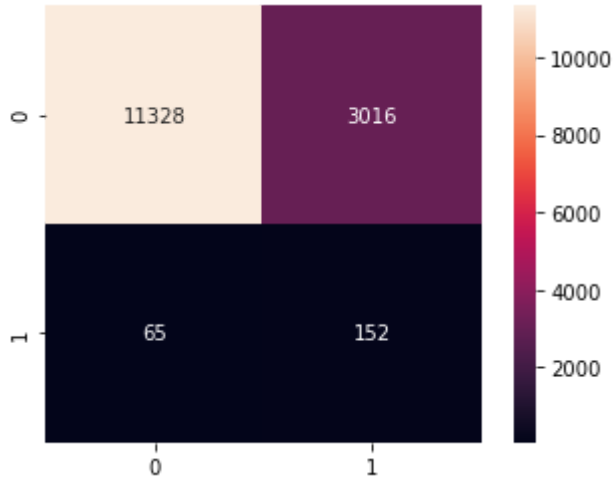
```
mat23 = confusion_matrix(y_test,y_pred23)
```

In [261]:

```
sns.heatmap(mat23,square=True,annot=True,fmt='d')
```

Out[261]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2135957a0d0>
```



here,We observed that model classification accuracy is 79%.

Conclusion

In this project, we have used different ways of addressing the task with unbalanced data.Like Supervised learning LinearSVC, Adaboost, GradientBoosting ,DecisionTreeClassifier,Logistic Regression and Random Forest.As per their's result we conclude that Random forest is the best algorithm model since it's giving high accuracy than other's. The highest accuracy is 93%.

In [ ]: