Insurance Claim Status Prediction

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.

Prepaird By Humera Shaikh

Importing Library

In [1]:
```
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 [2]:
```
df=pd.read_csv("insurane claim.csv")
```

In [3]:
```
df.head()
```

Out[3]:

	ID	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Gender	Age
0	3433	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	7	MALAYSIA	0.0	17.82	NaN	31
1	4339	EPX	Travel Agency	Online	Cancellation Plan	0	85	SINGAPORE	69.0	0.00	NaN	36
2	34590	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	11	MALAYSIA	19.8	11.88	NaN	75
3	55816	EPX	Travel Agency	Online	2 way Comprehensive Plan	0	16	INDONESIA	20.0	0.00	NaN	32
4	13816	EPX	Travel Agency	Online	Cancellation Plan	0	10	KOREA, REPUBLIC OF	15.0	0.00	NaN	29

In [4]:
```
df.shape
```

Out[4]:
```
(50553, 12)
```

EDA

In [5]:
```
# Checking the weightage of null values.
(df['Gender'].isnull().sum()/df.shape[0]*100).round(2)
```

Out[5]:
```
71.12
```

we can see 71% of the data does not have Gender updated. there are no other features that will help us find out whether the individual is male or female. Thus we will drop this feature

We can see from data ID column is not usefull data.hnce we will drop ID column

In [6]:
```
df.drop(["Gender"],axis=1,inplace=True)
```

In [7]:
```
df.drop(["ID"],axis=1,inplace=True)
```

In [8]:
```
df.describe()
```

Out[8]:

	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

Duration is negative so checking destination for rplace negative value

In [9]:
```
df[(df["Duration"]<0) & (df["Destination"])]
```

Out[9]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age
4063	JZI	Airlines	Online	Basic Plan	0	-1	BRUNEI DARUSSALAM	18.0	6.3	118
38935	JZI	Airlines	Online	Basic Plan	0	-1	INDONESIA	18.0	6.3	118
48367	JZI	Airlines	Online	Basic Plan	0	-2	BANGLADESH	22.0	7.7	118

In [10]:
```
#BANGLADESH,INDONESIA,BRUNEI DARUSSALAM has negaitive values
#replacing this negative value with mean value of its own
```

In [11]:
```
ban=df[(df["Duration"]) & (df["Destination"]=="BANGLADESH")]
ban["Duration"].mean()
```

Out[11]:
```
15.958333333333334
```

In [12]:
```
indo=df[(df["Duration"]) & (df["Destination"]=="INDONESIA")]
indo["Duration"].mean()
```

Out[12]:
```
31.335203366058906
```

In [13]:
```
BRUNEI=df[(df["Duration"]) & (df["Destination"]=="BRUNEI DARUSSALAM")]
BRUNEI["Duration"].mean()
```

Out[13]:
```
22.37012987012987
```

In [14]:
```
des=df[(df["Duration"]==0) & (df["Destination"])]
des
```

Out[14]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age
181	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
314	RAB	Airlines	Online	Value Plan	0	0	BRUNEI DARUSSALAM	15.00	6.00	53
1864	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
3068	C2B	Airlines	Online	Silver Plan	0	0	SINGAPORE	0.00	5.63	51
4282	LWC	Travel Agency	Online	Single Trip Travel Protect Gold	0	0	MALAYSIA	27.00	17.55	56
7324	RAB	Airlines	Online	Value Plan	0	0	BRUNEI DARUSSALAM	15.00	6.00	24
7482	JZI	Airlines	Online	Basic Plan	0	0	THAILAND	18.00	6.30	118
7885	JZI	Airlines	Online	Basic Plan	0	0	THAILAND	18.00	6.30	118
8205	JWT	Airlines	Online	Value Plan	0	0	INDIA	62.00	24.80	118
8512	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
8897	JWT	Airlines	Online	Value Plan	0	0	INDIA	62.00	24.80	118
9501	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
10465	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
10558	JWT	Airlines	Online	Value Plan	0	0	INDIA	62.00	24.80	118
11515	JZI	Airlines	Online	Basic Plan	0	0	THAILAND	18.00	6.30	118
12579	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	5.80	1.63	48
12924	C2B	Airlines	Online	Bronze Plan	0	0	SINGAPORE	-14.40	3.60	49
16251	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	1.80	0.50	48
17641	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	0.51	0.14	48
17741	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	1.35	0.38	48
18198	JWT	Airlines	Online	Value Plan	0	0	INDIA	62.00	24.80	118
18299	RAB	Airlines	Online	Value Plan	0	0	BRUNEI DARUSSALAM	15.00	6.00	33
19419	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	14.40	4.04	48
20538	JWT	Airlines	Online	Value Plan	0	0	INDIA	62.00	24.80	118
21442	C2B	Airlines	Online	Bronze Plan	0	0	SINGAPORE	27.00	6.75	23
23561	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	1.93	0.54	48
23606	JZI	Airlines	Online	Basic Plan	0	0	MYANMAR	18.00	6.30	24
23766	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	2.44	0.68	48
24505	JZI	Airlines	Online	Basic Plan	0	0	MYANMAR	18.00	6.30	26
25266	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	0	MALAYSIA	9.90	5.94	33
25785	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	4.44	1.25	48
26076	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	0.84	0.24	48
27002	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
30597	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	3.99	1.12	48
31258	EPX	Travel Agency	Offline	1 way Comprehensive Plan	0	0	MYANMAR	20.00	0.00	33
31932	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	2.86	0.80	48
33674	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	4.88	1.37	48
34178	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
34511	JZI	Airlines	Online	Basic Plan	0	0	CHINA	22.00	7.70	34
36144	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
36403	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
39014	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
39027	JZI	Airlines	Online	Basic Plan	0	0	INDONESIA	18.00	6.30	118
39318	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.00	12.40	118
41302	JWT	Airlines	Online	Value Plan	0	0	INDIA	62.00	24.80	118
41861	JWT	Airlines	Online	Value Plan	0	0	INDIA	93.00	37.20	118
43410	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	9.77	2.74	48
43464	RAB	Airlines	Online	Value Plan	0	0	BRUNEI DARUSSALAM	15.00	6.00	24
43485	JZI	Airlines	Online	Basic Plan	0	0	VIET NAM	18.00	6.30	58
43719	SSI	Airlines	Online	Ticket Protector	0	0	SINGAPORE	1.03	0.29	48
45395	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	0	UNITED STATES	9.90	5.94	59
45474	RAB	Airlines	Online	Value Plan	0	0	BRUNEI DARUSSALAM	15.00	6.00	38
47625	LWC	Travel Agency	Online	Single Trip Travel Protect Gold	0	0	INDIA	35.25	22.91	24
50357	C2B	Airlines	Online	Bronze Plan	0	0	SINGAPORE	14.40	3.60	49

In [15]:
```
des["Destination"].mode()
```

Out[15]:
```
0    INDIA
dtype: object
```

In [16]:
```
ind=df[(df["Duration"]) & (df["Destination"]=="INDIA")]
ind["Duration"].mean()
```

Out[16]:
```
30.41387024608501
```

In [17]:
```
#duration has negative value
for i in range(len(df)):
    if df['Duration'][i]<0:
        print(df['Duration'].iloc[i])
```

```
-1
-1
-2
```

In [18]:
```
df[df["Age"] >100]
```

Out[18]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age
90	JWT	Airlines	Online	Value Plan	0	58	INDIA	78.0	31.20	118
108	JWT	Airlines	Online	Value Plan	0	15	INDIA	31.0	12.40	118
140	JWT	Airlines	Online	Value Plan	0	8	INDIA	39.0	15.60	118
153	JWT	Airlines	Online	Value Plan	0	4	INDIA	78.0	31.20	118
181	JWT	Airlines	Online	Value Plan	0	0	INDIA	31.0	12.40	118
...	...	...	...	...	...	...	...	...	...	...
50158	JWT	Airlines	Online	Value Plan	0	41	INDIA	60.0	24.00	118
50179	JWT	Airlines	Online	Value Plan	0	62	INDIA	31.0	12.40	118
50250	JWT	Airlines	Online	Value Plan	0	15	INDIA	31.0	12.40	118
50429	JZI	Airlines	Online	Basic Plan	0	19	SRI LANKA	35.0	12.25	118
50478	CCR	Travel Agency	Offline	Comprehensive Plan	0	6	THAILAND	29.0	9.57	118

795 rows × 10 columns

In [19]:
```
df["Age"].mean()
```

Out[19]:
```
40.011235732795285
```

#we can see from data there is 795 rows so we replace age with mean value 40

In [20]:
```
#we can see from data there is 795 rows so we replace age with mean value 40.011235732795285
df.loc[df['Age'] > 100, 'Age'] = 40
```

In [21]:
```
df[df["Age"] >100]
```

Out[21]:

Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age

In [22]:
```
df[df["Duration"]==-2]
```

Out[22]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age
48367	JZI	Airlines	Online	Basic Plan	0	-2	BANGLADESH	22.0	7.7	40

In [23]:
```
# we replace duration -1 value with 27  INDONESIA,BRUNEI DARUSSALAM travel duration mean time is 27
# we replace duration -2 value with 16  Bangladesh  duration mean time is 16

df.loc[df['Duration']==-1, 'Duration'] =27
df.loc[df['Duration']==-2, 'Duration'] =16
```

In [24]:
```
df[df["Duration"]==-1]
```

Out[24]:

Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age

In [25]:
```
# we replace duration 0 value with 30 because mode of 0 value desitnation is india.and indias meaan is 30
df.loc[df['Duration']==0, 'Duration'] =30
```

In [26]:
```
df[df["Duration"]==0]
```

Out[26]:

Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age

In [27]:
```
df.head()
```

Out[27]:

	Agency	Agency Type	Distribution Channel	Product Name	Claim	Duration	Destination	Net Sales	Commision (in value)	Age
0	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	7	MALAYSIA	0.0	17.82	31
1	EPX	Travel Agency	Online	Cancellation Plan	0	85	SINGAPORE	69.0	0.00	36
2	CWT	Travel Agency	Online	Rental Vehicle Excess Insurance	0	11	MALAYSIA	19.8	11.88	75
3	EPX	Travel Agency	Online	2 way Comprehensive Plan	0	16	INDONESIA	20.0	0.00	32
4	EPX	Travel Agency	Online	Cancellation Plan	0	10	KOREA, REPUBLIC OF	15.0	0.00	29

In [28]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50553 entries, 0 to 50552
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Agency                50553 non-null  object 
 1   Agency Type           50553 non-null  object 
 2   Distribution Channel  50553 non-null  object 
 3   Product Name          50553 non-null  object 
 4   Claim                 50553 non-null  int64  
 5   Duration              50553 non-null  int64  
 6   Destination           50553 non-null  object 
 7   Net Sales             50553 non-null  float64
 8   Commision (in value)  50553 non-null  float64
 9   Age                   50553 non-null  int64  
dtypes: float64(2), int64(3), object(5)
memory usage: 3.9+ MB
```

In [ ]:

In [29]:
```
#changing data of claim int to objetive
df["Claim"]=df["Claim"].astype('int')
```

In [30]:
```
#net sale wont be negative so removing those record
df=df[df['Net Sales'] > 0]
df['Net Sales'].describe().T
```

Out[30]:
```
count    48536.000000
mean        43.148207
std         47.939269
min          0.070000
25%         19.800000
50%         28.000000
75%         49.500000
max        810.000000
Name: Net Sales, dtype: float64
```

In [31]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48536 entries, 1 to 50552
Data columns (total 10 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   Agency                48536 non-null  object 
 1   Agency Type           48536 non-null  object 
 2   Distribution Channel  48536 non-null  object 
 3   Product Name          48536 non-null  object 
 4   Claim                 48536 non-null  int32  
 5   Duration              48536 non-null  int64  
 6   Destination           48536 non-null  object 
 7   Net Sales             48536 non-null  float64
 8   Commision (in value)  48536 non-null  float64
 9   Age                   48536 non-null  int64  
dtypes: float64(2), int32(1), int64(2), object(5)
memory usage: 3.9+ MB
```

Data visualization

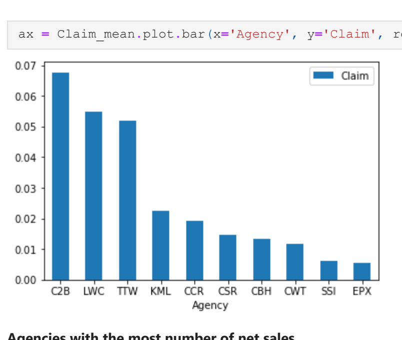
Agencies with maximum claims

In [32]:
```
#ploting diffrant diffrant graphs for understnding
# Finding the Agencies with the Maximum Claims
Claim_mean = (df.groupby('Agency', as_index=False)['Claim'].mean()).nlargest(10,'Claim')
Claim_mean
```

Out[32]:

	Agency	Claim
2	C2B	0.067726
11	LWC	0.055028
15	TTW	0.051948
10	KML	0.022364
4	CCR	0.019355
5	CSR	0.014706
3	CBH	0.013333
6	CWT	0.011810
13	SSI	0.006010
7	EPX	0.005468

In [33]:
```
ax = Claim_mean.plot.bar(x='Agency', y='Claim', rot=0)
```



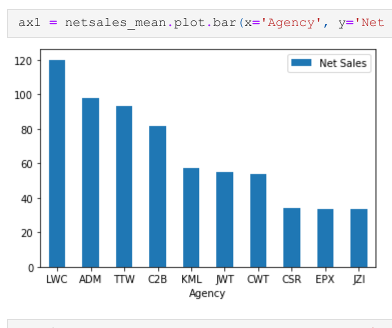
Agencies with the most number of net sales

In [34]:
```
netsales_mean = (df.groupby('Agency', as_index=False)['Net Sales'].mean()).nlargest(10,'Net Sales')
netsales_mean
```

Out[34]:

	Agency	Net Sales
11	LWC	120.021442
0	ADM	97.697674
15	TTW	93.194805
2	C2B	81.687274
10	KML	57.329073
8	JWT	55.108514
6	CWT	53.973741
5	CSR	33.808824
7	EPX	33.473937
9	JZI	33.370798

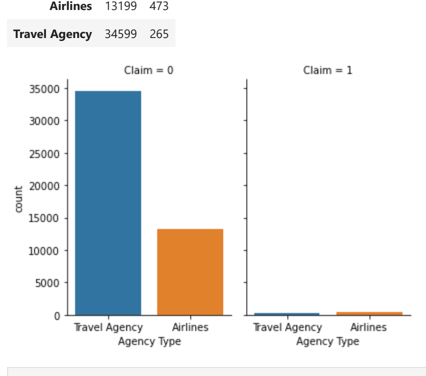
In [35]:
```
ax1 = netsales_mean.plot.bar(x='Agency', y='Net Sales', rot=0)
```



In [36]:
```
plot2 = sns.catplot(x="Agency Type", col="Claim",data=df, kind="count",height=4, aspect=.7)
df.pivot_table(index=['Agency Type'],columns=['Claim'], values=['Agency'], aggfunc='count')
```

Out[36]:

	Agency	
Claim	0	1
Agency Type		
Airlines	13199	473
Travel Agency	34599	265



In [37]:
```
df.pivot_table(index = ['Product Name'], values = ['Net Sales'], aggfunc = 'mean')
```

Out[37]:

	Net Sales
Product Name	
1 way Comprehensive Plan	28.476154
2 way Comprehensive Plan	43.524684
24 Protect	24.324706
Annual Gold Plan	362.430000
Annual Silver Plan	227.285451
Annual Travel Protect Gold	249.142593
Annual Travel Protect Platinum	335.104444
Annual Travel Protect Silver	211.761765
Basic Plan	27.075577
Bronze Plan	33.803918
Cancellation Plan	27.256394
Child Comprehensive Plan	74.000000
Comprehensive Plan	30.021127
Gold Plan	86.322500
Individual Comprehensive Plan	97.000000
Premier Plan	79.138889
Rental Vehicle Excess Insurance	53.973741
Silver Plan	51.496579
Single Trip Travel Protect Gold	42.277333
Single Trip Travel Protect Platinum	47.684545
Single Trip Travel Protect Silver	36.148438
Spouse or Parents Comprehensive Plan	86.000000
Ticket Protector	6.488774
Travel Cruise Protect	30.000000
Value Plan	50.639810

We can see from above Desity,Commision,Net sale,duration,clasim positivly skewd

In [38]:
```
#seprating numeric and catagorical data
df_num=df.select_dtypes(["int","float","int64","float64"])
df_cat=df.select_dtypes("object")
```

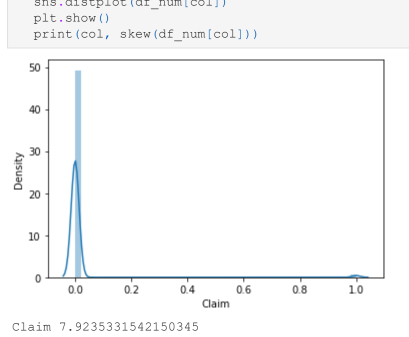
In [39]:
```
df_num.head()
```

Out[39]:

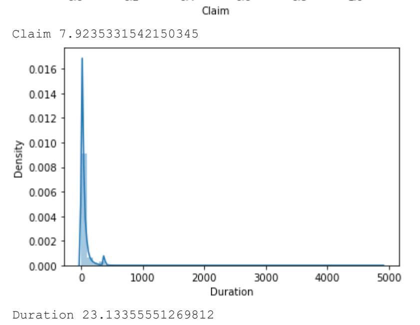
	Claim	Duration	Net Sales	Commision (in value)	Age
1	0	85	69.0	0.00	36
2	0	11	19.8	11.88	75
3	0	16	20.0	0.00	32
4	0	10	15.0	0.00	29
5	0	64	49.5	29.70	36

Skewness

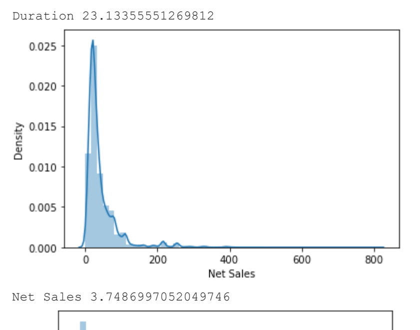
In [40]:
```
from scipy.stats import skew

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



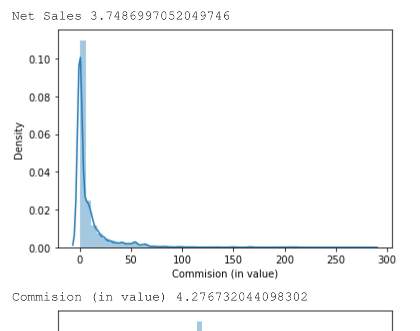

```
Claim 7.9235331542150345
```



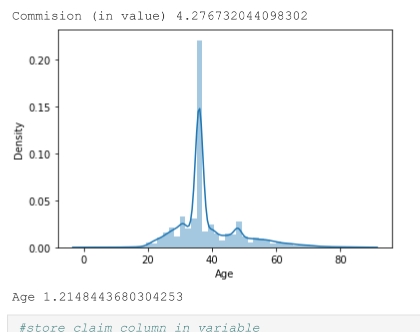

```
Duration 23.13355551269812
```




```
Net Sales 3.7486997052049746
```




```
Commision (in value) 4.276732044098302
```




```
Age 1.2148443680304253
```

In [41]:
```
#store claim column in variable
df_numnew=df_num["Claim"]
df_numwitoutclaim=df_num[["Duration","Net Sales","Commision (in value)","Age"]]
```

In [42]:
```
df_numwitoutclaim
```

Out[42]:

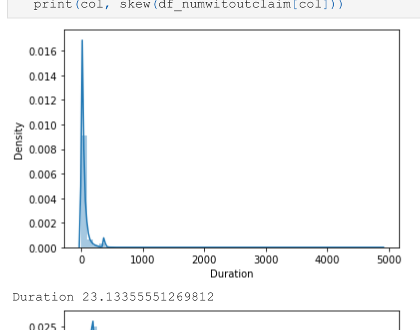
	Duration	Net Sales	Commision (in value)	Age
1	85	69.0	0.00	36
2	11	19.8	11.88	75
3	16	20.0	0.00	32
4	10	15.0	0.00	29
5	64	49.5	29.70	36
...	...	...	...	...
50548	11	99.0	59.40	50
50549	48	20.0	0.00	36
50550	24	28.0	10.64	48
50551	25	15.0	0.00	36
50552	9	20.0	0.00	59

48536 rows × 4 columns

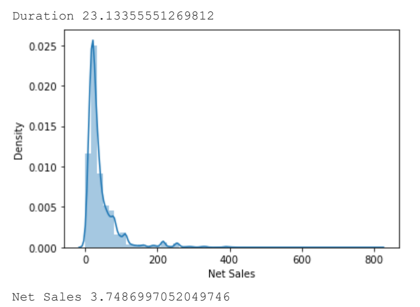
In [43]:
```
df_numnew
```

Out[43]:
```
1        0
2        0
3        0
4        0
5        0
        ..
50548    0
50549    0
50550    0
50551    0
50552    0
Name: Claim, Length: 48536, dtype: int32
```

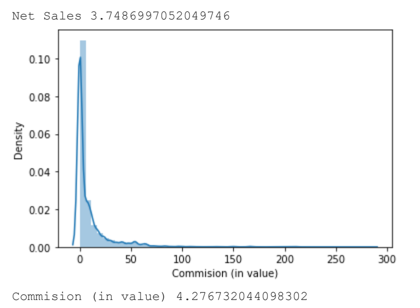
In [44]:
```
#lets see skewness
for col in df_numwitoutclaim:
  plt.figure()
  sns.distplot(df_numwitoutclaim[col])
  plt.show()
  print(col, skew(df_numwitoutclaim[col]))
```



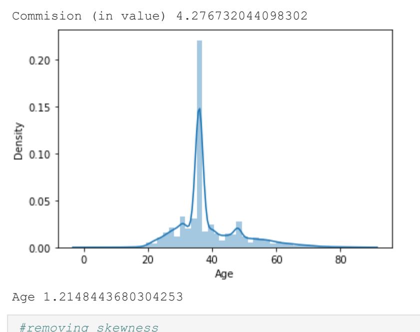

```
Duration 23.13355551269812
```




```
Net Sales 3.7486997052049746
```




```
Commision (in value) 4.276732044098302
```




```
Age 1.2148443680304253
```

In [45]:
```
#removing skewness
for col in df_numwitoutclaim:
  if skew(df_numwitoutclaim[col]) >= 0.5 or skew(df_numwitoutclaim[col]) <= -0.5:
    df_numwitoutclaim[col] = np.sqrt(df_numwitoutclaim[col])
```

In [46]:
```
for col in df_numwitoutclaim:
  print(col, skew(df_numwitoutclaim[col]))
```

```
Duration 2.4681989896626795
Net Sales 1.8479734387767213
Commision (in value) 1.4209919803987445
Age 0.6971452234312263
```

Combine Data

In [47]:
```
df_num=pd.concat([df_numwitoutclaim,df_numnew],axis=1)
```

In [48]:
```
df_num
```

Out[48]:

	Duration	Net Sales	Commision (in value)	Age	Claim
1	9.219544	8.306624	0.000000	6.000000	0
2	3.316625	4.449719	3.446738	8.660254	0
3	4.000000	4.472136	0.000000	5.656854	0
4	3.162278	3.872983	0.000000	5.385165	0
5	8.000000	7.035624	5.449771	6.000000	0
...	...	...	...	...	...
50548	3.316625	9.949874	7.707140	7.071068	0
50549	6.928203	4.472136	0.000000	6.000000	0
50550	4.898979	5.291503	3.261901	6.928203	0
50551	5.000000	3.872983	0.000000	6.000000	0
50552	3.000000	4.472136	0.000000	7.681146	0

48536 rows × 5 columns

Encoding of data

In [49]:
```
from sklearn.preprocessing import LabelEncoder
for col in df_cat:
    le = LabelEncoder()
    df_cat[col] = le.fit_transform(df_cat[col])
```

In [50]:
```
df_cat
```

Out[50]:

	Agency	Agency Type	Distribution Channel	Product Name	Destination
1	7	1	1	10	79
2	6	1	1	16	56
3	7	1	1	1	38
4	7	1	1	10	47
5	6	1	1	16	88
...	...	...	...	...	...
50548	6	1	1	16	65
50549	7	1	1	0	38
50550	10	1	1	24	38
50551	7	1	1	10	7
50552	7	1	1	0	79

48536 rows × 5 columns

In [51]:
```
df_new=pd.concat([df_num,df_cat],axis=1)
```

In [52]:
```
df_new
```

Out[52]:

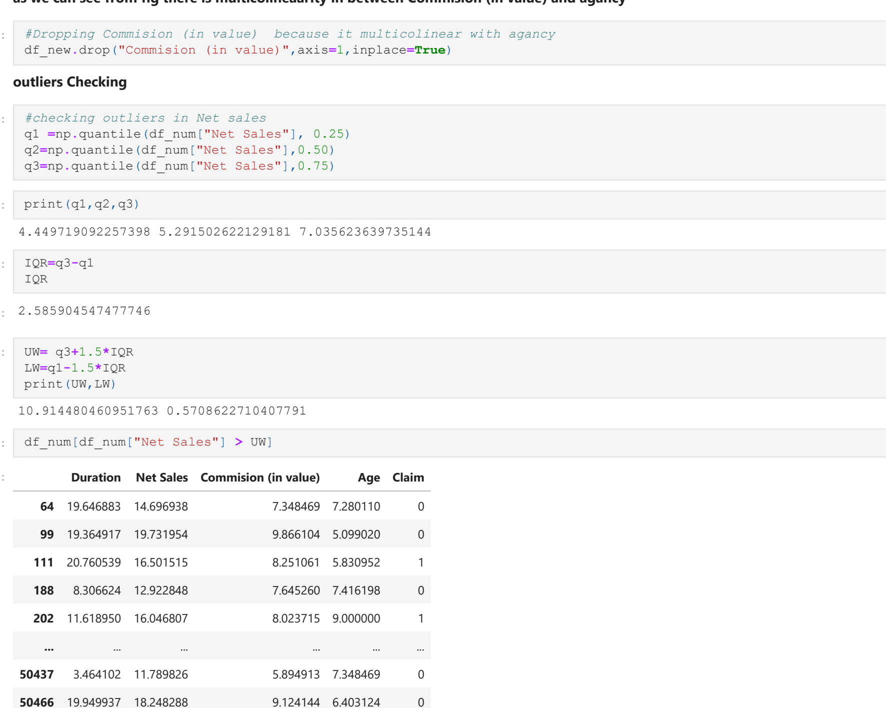
	Duration	Net Sales	Commision (in value)	Age	Claim	Agency	Agency Type	Distribution Channel	Product Name	Destination
1	9.219544	8.306624	0.000000	6.000000	0	7	1	1	10	79
2	3.316625	4.449719	3.446738	8.660254	0	6	1	1	16	56
3	4.000000	4.472136	0.000000	5.656854	0	7	1	1	1	38
4	3.162278	3.872983	0.000000	5.385165	0	7	1	1	10	47
5	8.000000	7.035624	5.449771	6.000000	0	6	1	1	16	88
...	...	...	...	...	...	...	...	...	...	...
50548	3.316625	9.949874	7.707140	7.071068	0	6	1	1	16	65
50549	6.928203	4.472136	0.000000	6.000000	0	7	1	1	0	38
50550	4.898979	5.291503	3.261901	6.928203	0	10	1	1	24	38
50551	5.000000	3.872983	0.000000	6.000000	0	7	1	1	10	7
50552	3.000000	4.472136	0.000000	7.681146	0	7	1	1	0	79

48536 rows × 10 columns

Ploting Data for multicolinearity

In [53]:
```
#ploting heatmap
plt.show()
plt.figure(figsize=(18,12))
sns.heatmap(df_new.corr(),annot=True)
```

Out[53]: <AxesSubplot:>



as we can see from fig there is multicolineaarity in between Commision (in value) and agancy

In [54]:
```
#Dropping Commision (in value)  because it multicolinear with agancy
df_new.drop("Commision (in value)",axis=1,inplace=True)
```

outliers Checking

In [55]:
```
#checking outliers in Net sales
q1 =np.quantile(df_num["Net Sales"], 0.25)
q2=np.quantile(df_num["Net Sales"],0.50)
q3=np.quantile(df_num["Net Sales"],0.75)
```

In [56]:
```
print(q1,q2,q3)
```

```
4.449719092257398 5.291502622129181 7.035623639735144
```

In [57]:
```
IQR=q3-q1
IQR
```

Out[57]: 2.585904547477746

In [58]:
```
UW= q3+1.5*IQR
LW=q1-1.5*IQR
print(UW,LW)
```

```
10.914480460951763 0.5708622710407791
```

In [59]:
```
df_num[df_num["Net Sales"] > UW]
```

Out[59]:

	Duration	Net Sales	Commision (in value)	Age	Claim
64	19.646883	14.696938	7.348469	7.280110	0
99	19.364917	19.731954	9.866104	5.099020	0
111	20.760539	16.501515	8.251061	5.830952	1
188	8.306624	12.922848	7.645260	7.416198	0
202	11.618950	16.046807	8.023715	9.000000	1
...	...	...	...	...	...
50437	3.464102	11.789826	5.894913	7.348469	0
50466	19.949937	18.248288	9.124144	6.403124	0
50493	19.157244	12.688578	6.344289	7.071068	0
50513	19.104973	15.901258	7.950472	5.567764	0
50536	19.416488	15.901258	7.950472	7.874008	0

2194 rows × 5 columns

In [60]:
```
#Duration
q1 =np.quantile(df_new["Duration"], 0.25)
q2=np.quantile(df_new["Duration"],0.50)
q3=np.quantile(df_new["Duration"],0.75)
print(q1,q2,q3)
```

```
3.0 4.69041575982343 7.211102550927978
```

In [61]:
```
IQR=q3-q1
IQR
```

Out[61]: 4.211102550927978

In [62]:
```
UW= q3+1.5*IQR
LW=q1-1.5*IQR
print(UW,LW)
```

```
13.527756377319946 -3.316653826391967
```

In [63]:
```
df_new[df_new["Duration"] > UW]
```

Out[63]:

	Duration	Net Sales	Age	Claim	Agency	Agency Type	Distribution Channel	Product Name	Destination
18	15.491933	5.385165	5.099020	0	5	1	0	12	88
64	19.646883	14.696938	7.280110	0	2	0	1	4	79
99	19.364917	19.731954	5.099020	0	2	0	1	3	79
100	13.638182	6.557439	6.000000	0	7	1	1	1	17
111	20.760539	16.501515	5.830952	1	2	0	1	4	79
...	...	...	...	...	...	...	...	...	...
50493	19.157244	12.688578	7.071068	0	2	0	1	4	79
50513	19.104973	15.901258	5.567764	0	2	0	1	4	79
50525	16.309506	6.292853	6.403124	0	6	1	1	16	36
50536	19.416488	15.901258	7.874008	0	2	0	1	4	79
50546	13.784049	3.162278	6.000000	0	7	1	1	10	88

2502 rows × 9 columns

In [64]:
```
df_new[df_new["Duration"] < UW]
```

Out[64]:

	Duration	Net Sales	Age	Claim	Agency	Agency Type	Distribution Channel	Product Name	Destination
1	9.219544	8.306624	6.000000	0	7	1	1	10	79
2	3.316625	4.449719	8.660254	0	6	1	1	16	56
3	4.000000	4.472136	5.656854	0	7	1	1	1	38
4	3.162278	3.872983	5.385165	0	7	1	1	10	47
5	8.000000	7.035624	6.000000	0	6	1	1	16	88
...	...	...	...	...	...	...	...	...	...
50548	3.316625	9.949874	7.071068	0	6	1	1	16	65
50549	6.928203	4.472136	6.000000	0	7	1	1	0	38
50550	4.898979	5.291503	6.928203	0	10	1	1	24	38
50551	5.000000	3.872983	6.000000	0	7	1	1	10	7
50552	3.000000	4.472136	7.681146	0	7	1	1	0	79

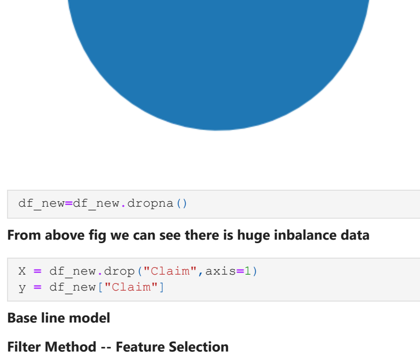
46034 rows × 9 columns

Lets Check the ration of Claim and Non Claim data

In [65]:
```
LABELS = ["Non_claim","Clamin"]
```

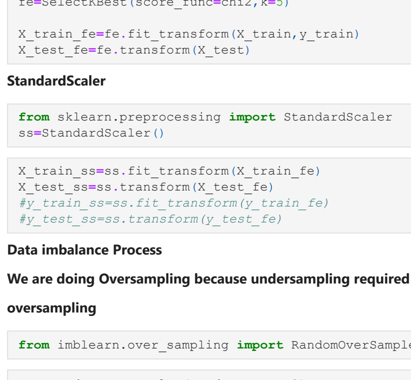
In [66]:
```
count_classes = pd.value_counts(df_new['Claim'], sort = True)

count_classes.plot(kind = 'bar', rot=0)

plt.title("Insurance Claim")

plt.xticks(range(2), LABELS)

plt.xlabel("Claim")

plt.ylabel("Frequency")
```

Out[66]: Text(0, 0.5, 'Frequency')



In [67]:
```
print(df_new["Claim"].value_counts())
print("--------------------------------------------------")
plt.figure(figsize=(7,7))
df["Claim"].value_counts().plot.pie(autopct="%.1f%%")
plt.show()
```

```
0    47798
1      738
Name: Claim, dtype: int64
--------------------------------------------------
```



In [68]:
```
df_new=df_new.dropna()
```

From above fig we can see there is huge inbalance data

In [69]:
```
X = df_new.drop("Claim",axis=1)
y = df_new["Claim"]
```

Base line model

Filter Method -- Feature Selection

In [70]:
```
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)
```

In [71]:
```
from sklearn.feature_selection import chi2
from sklearn.feature_selection import f_classif
from sklearn.feature_selection import SelectKBest

fe=SelectKBest(score_func=chi2,k=5)

X_train_fe=fe.fit_transform(X_train,y_train)
X_test_fe=fe.transform(X_test)
```

StandardScaler

In [72]:
```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
```

In [73]:
```
X_train_ss=ss.fit_transform(X_train_fe)
X_test_ss=ss.transform(X_test_fe)
#y_train_ss=ss.fit_transform(y_train_fe)
#y_test_ss=ss.transform(y_test_fe)
```

Data imbalance Process

We are doing Oversampling because undersampling required huge amount of data and chances of lossing data is high

oversampling

In [74]:
```
from imblearn.over_sampling import RandomOverSampler
```

In [75]:
```
ros=RandomOverSampler(random_state=1)
X_sample2 ,y_sample2=ros.fit_resample(X_train_ss,y_train)


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

Out[75]:
```
1    33452
0    33452
Name: Claim, dtype: int64
```

Modeling Process Model-1 LogisticRegression

In [76]:
```
from sklearn.linear_model import LogisticRegression
```

In [77]:
```
lr1=LogisticRegression()
```

In [78]:
```
lr1.fit(X_sample2,y_sample2)
```

Out[78]: LogisticRegression()

In [79]:
```
score=cross_val_score(lr1,X_sample2,y_sample2,cv=8)
```

In [80]:
```
score
```

Out[80]:
```
array([0.74458926, 0.74291522, 0.74530671, 0.74183905, 0.74231735,
       0.74291522, 0.74614373, 0.74040416])
```

In [81]:
```
y_pred1=lr1.predict(X_test_ss)
```

In [82]:
```
df1=pd.DataFrame(X_test_ss)
df1['Prediction']=y_pred1
df1
```

Out[82]:

	0	1	2	3	4	Prediction
0	-1.018648	-1.040520	0.136643	0.623883	-0.072304	0
1	0.019381	-1.040520	0.136643	0.623883	-0.697499	0
2	3.380968	2.481709	-1.957570	-1.602865	0.726556	1
3	0.368898	1.258712	0.136643	0.623883	-1.878422	0
4	0.403685	0.391601	-0.282199	0.623883	-1.982621	0
...	...	...	...	...	...	...
14556	-1.018648	-1.040520	0.136643	0.623883	0.726556	0
14557	-0.544192	-0.085753	0.974328	-1.602865	-0.072304	0
14558	0.735889	1.560864	0.136643	0.623883	1.317017	1
14559	0.164281	-0.053811	0.136643	0.623883	-0.384901	0
14560	-0.070199	1.066149	-1.957570	-1.602865	0.726556	1

14561 rows × 6 columns

In [83]:
```
df1['Prediction'].value_counts()
```

Out[83]:
```
0    11718
1     2843
Name: Prediction, dtype: int64
```

In [85]:
```
print(classification_report(y_test,y_pred1))

#pd.DataFrame(classification_report(y_test, y_pred1, output_dict=True))
```

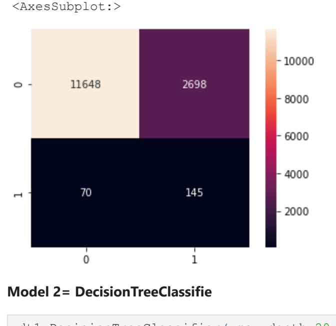
```
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     14346
           1       0.05      0.67      0.09       215

    accuracy                           0.81     14561
   macro avg       0.52      0.74      0.49     14561
weighted avg       0.98      0.81      0.88     14561
```

In [86]:
```
#Logestic regression Testing Training Accuracy
print("Train Accuracy:",lr1.score(X_sample2, y_sample2))
print("Test Accuracy:",lr1.score(X_test_ss, y_test))
```

```
Train Accuracy: 0.7436775080712663
Test Accuracy: 0.8099031659913467
```

In [87]:
```
mat1=confusion_matrix(y_test,y_pred1)
sns.heatmap(mat1,square=True,fmt="d",annot=True)
```

Out[87]: <AxesSubplot:>



we can see our base model logistic regression give us acuracy of 81%

roc_auc_score

In [88]:
```
scores=lr1.predict_proba(X_test_ss)
```

In [89]:
```
print(scores)
```

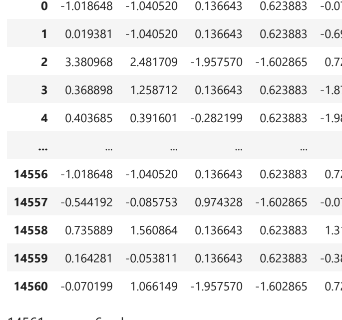
```
[[0.79548903 0.20451097]
 [0.79810906 0.20189094]
 [0.09097915 0.90902085]
 ...
 [0.45407006 0.54592994]
 [0.69544252 0.30455748]
 [0.19179342 0.80820658]]
```

In [90]:
```
roc_auc_score(y_test,y_pred1)
```

Out[90]: 0.743176122345099

In [91]:
```
mat1=confusion_matrix(y_test,y_pred1)
sns.heatmap(mat1,square=True,fmt="d",annot=True)
```

Out[91]: <AxesSubplot:>



Model 2= DecisionTreeClassifie

In [92]:
```
dt1=DecisionTreeClassifier( max_depth=20, min_samples_leaf=35)
```

In [93]:
```
dt1.fit(X_sample2,y_sample2)
```

Out[93]: DecisionTreeClassifier(max_depth=20, min_samples_leaf=35)

In [94]:
```
score=cross_val_score(dt1,X_sample2,y_sample2,cv=5)
```

In [95]:
```
score
```

Out[95]: array([0.89836335, 0.88954488, 0.89955908, 0.89701816, 0.8887145])

In [96]:
```
y_pred3=dt1.predict(X_test_ss)
```

In [97]:
```
df2=pd.DataFrame(X_test_ss)
df2['Prediction']=y_pred3
df2
```

Out[97]:

	0	1	2	3	4	Prediction
0	-1.018648	-1.040520	0.136643	0.623883	-0.072304	0
1	0.019381	-1.040520	0.136643	0.623883	-0.697499	0
2	3.380968	2.481709	-1.957570	-1.602865	0.726556	1
3	0.368898	1.258712	0.136643	0.623883	-1.878422	0
4	0.403685	0.391601	-0.282199	0.623883	-1.982621	0
...	...	...	...	...	...	...
14556	-1.018648	-1.040520	0.136643	0.623883	0.726556	0
14557	-0.544192	-0.085753	0.974328	-1.602865	-0.072304	1
14558	0.735889	1.560864	0.136643	0.623883	1.317017	0
14559	0.164281	-0.053811	0.136643	0.623883	-0.384901	0
14560	-0.070199	1.066149	-1.957570	-1.602865	0.726556	1

14561 rows × 6 columns

In [98]:
```
df2['Prediction'].value_counts()
#count of prediction 0 and 1
```

Out[98]:
```
0    11942
1     2619
Name: Prediction, dtype: int64
```

In [99]:
```
print(classification_report(y_test,y_pred3))
#pd.DataFrame(classification_report(y_test, y_pred3, output_dict=True))
```

```
              precision    recall  f1-score   support

           0       0.99      0.83      0.90     14346
           1       0.05      0.55      0.08       215

    accuracy                           0.82     14561
   macro avg       0.52      0.69      0.49     14561
weighted avg       0.98      0.82      0.89     14561
```

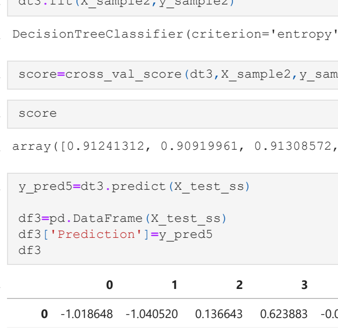
In [100…
```
print("Train Accuracy:",dt1.score(X_sample2, y_sample2))
print("Test Accuracy:",dt1.score(X_test_ss, y_test))
```

```
Train Accuracy: 0.9125313882578022
Test Accuracy: 0.8215781883112424
```

From above fig we can see our model accuracy with gimi index is 82%

In [101…
```
mat3=confusion_matrix(y_test,y_pred3)
sns.heatmap(mat3,square=True,fmt="d",annot=True)
```

Out[101… <AxesSubplot:>



roc_auc_score

In [102…
```
scores=dt1.predict_proba(X_test_ss)
```

In [103…
```
print(scores)
```

```
[[1.         0.        ]
 [1.         0.        ]
 [0.10886076 0.89113924]
 ...
 [1.         0.        ]
 [1.         0.        ]
 [0.27586207 0.72413793]]
```

In [104…
```
roc_auc_score(y_test,y_pred3)
```

Out[104… 0.6872514500436067

Decesion tree with (criterion="entropy")

In [105…
```
dt3=DecisionTreeClassifier(criterion="entropy", max_depth=25, min_samples_leaf=30)
```

In [106…
```
dt3.fit(X_sample2,y_sample2)
```

Out[106… DecisionTreeClassifier(criterion='entropy', max_depth=25, min_samples_leaf=30)

In [107…
```
score=cross_val_score(dt3,X_sample2,y_sample2,cv=5)
```

In [108…
```
score
```

Out[108… array([0.91241312, 0.90919961, 0.91308572, 0.9062103 , 0.90814649])

In [109…
```
y_pred5=dt3.predict(X_test_ss)

df3=pd.DataFrame(X_test_ss)
df3['Prediction']=y_pred5
df3
```

Out[109…

	0	1	2	3	4	Prediction
0	-1.018648	-1.040520	0.136643	0.623883	-0.072304	0
1	0.019381	-1.040520	0.136643	0.623883	-0.697499	0
2	3.380968	2.481709	-1.957570	-1.602865	0.726556	1
3	0.368898	1.258712	0.136643	0.623883	-1.878422	0
4	0.403685	0.391601	-0.282199	0.623883	-1.982621	1
...	...	...	...	...	...	...
14556	-1.018648	-1.040520	0.136643	0.623883	0.726556	0
14557	-0.544192	-0.085753	0.974328	-1.602865	-0.072304	1
14558	0.735889	1.560864	0.136643	0.623883	1.317017	0
14559	0.164281	-0.053811	0.136643	0.623883	-0.384901	0
14560	-0.070199	1.066149	-1.957570	-1.602865	0.726556	1

14561 rows × 6 columns

In [110…
```
df3['Prediction'].value_counts()
# Prediction results for 1 and 0
```

Out[110…
```
0    12177
1     2384
Name: Prediction, dtype: int64
```

In [111…
```
print(classification_report(y_test,y_pred5))
#pd.DataFrame(classification_report(y_test, y_pred5, output_dict=True))
```

```
              precision    recall  f1-score   support

           0       0.99      0.84      0.91     14346
           1       0.05      0.51      0.08       215

    accuracy                           0.84     14561
   macro avg       0.52      0.68      0.50     14561
weighted avg       0.98      0.84      0.90     14561
```

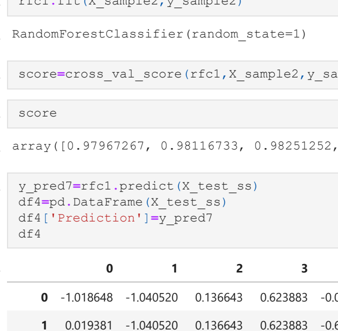
In [112…
```
print("Train Accuracy:",dt3.score(X_sample2, y_sample2))
print("Test Accuracy:",dt3.score(X_test_ss, y_test))
```

```
Train Accuracy: 0.9258639244290326
Test Accuracy: 0.8366183641233432
```

From above fig we can see our model accuracy with entropy is 83%

In [113…
```
mat5=confusion_matrix(y_test,y_pred5)
sns.heatmap(mat5,square=True,annot=True,fmt='d')
```

Out[113… <AxesSubplot:>



In [114…
```
scores=dt3.predict_proba(X_test_ss)
print(scores)
```

```
[[1.         0.        ]
 [1.         0.        ]
 [0.12418301 0.87581699]
 ...
 [1.         0.        ]
 [1.         0.        ]
 [0.27586207 0.72413793]]
```

In [115…
```
y_test.value_counts()
```

Out[115…
```
0    14346
1      215
Name: Claim, dtype: int64
```

In [116…
```
roc_auc_score(y_test,y_pred5)
```

Out[116… 0.6765584118739848

Model 3== RandomForestClassifier

In [117…
```
rfc1=RandomForestClassifier(n_estimators=100,random_state=1)
```

In [118…
```
rfc1.fit(X_sample2,y_sample2)
```

Out[118… RandomForestClassifier(random_state=1)

In [119…
```
score=cross_val_score(rfc1,X_sample2,y_sample2,cv=5)
```

In [120…
```
score
```

Out[120… array([0.97967267, 0.98116733, 0.98251252, 0.98027053, 0.97944694])

In [121…
```
y_pred7=rfc1.predict(X_test_ss)
df4=pd.DataFrame(X_test_ss)
df4['Prediction']=y_pred7
df4
```

Out[121…

	0	1	2	3	4	Prediction
0	-1.018648	-1.040520	0.136643	0.623883	-0.072304	0
1	0.019381	-1.040520	0.136643	0.623883	-0.697499	0
2	3.380968	2.481709	-1.957570	-1.602865	0.726556	1
3	0.368898	1.258712	0.136643	0.623883	-1.878422	0
4	0.403685	0.391601	-0.282199	0.623883	-1.982621	0
...	...	...	...	...	...	...
14556	-1.018648	-1.040520	0.136643	0.623883	0.726556	0
14557	-0.544192	-0.085753	0.974328	-1.602865	-0.072304	0
14558	0.735889	1.560864	0.136643	0.623883	1.317017	0
14559	0.164281	-0.053811	0.136643	0.623883	-0.384901	0
14560	-0.070199	1.066149	-1.957570	-1.602865	0.726556	0

14561 rows × 6 columns

In [122…
```
df4['Prediction'].value_counts()
#prediction of random forest
```

Out[122…
```
0    13972
1      589
Name: Prediction, dtype: int64
```

In [123…
```
print(classification_report(y_test,y_pred7))
```

```
              precision    recall  f1-score   support

           0       0.99      0.96      0.98     14346
           1       0.08      0.23      0.12       215

    accuracy                           0.95     14561
   macro avg       0.54      0.60      0.55     14561
weighted avg       0.97      0.95      0.96     14561
```

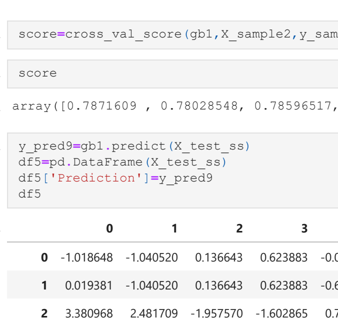
In [124…
```
print("Train Accuracy:",rfc1.score(X_sample2, y_sample2))
print("Test Accuracy:",rfc1.score(X_test_ss, y_test))
```

```
Train Accuracy: 0.9840966160468732
Test Accuracy: 0.9516516722752558
```

From above fig we can see our Random model accuracy is 95%

In [125…
```
mat7=confusion_matrix(y_test,y_pred7)
sns.heatmap(mat7,square=True,annot=True,fmt='d')
```

Out[125… <AxesSubplot:>



In [126…
```
scores=rfc1.predict_proba(X_test_ss)
```

In [127…
```
print(scores)
```

```
[[1.         0.        ]
 [1.         0.        ]
 [0.09003336 0.90996664]
 ...
 [0.71394866 0.28605134]
 [1.         0.        ]
 [0.99       0.01      ]]
```

In [128…
```
roc_auc_score(y_test,y_pred7)
```

Out[128… 0.5974933455237502

Model 4 Gradient Boosting

In [129…
```
from sklearn.ensemble import GradientBoostingClassifier
gb1=GradientBoostingClassifier()
```

In [130…
```
gb1.fit(X_sample2,y_sample2)
```

Out[130… GradientBoostingClassifier()

In [131…
```
score=cross_val_score(gb1,X_sample2,y_sample2,cv=5)
```

In [132…
```
score
```

Out[132… array([0.7871609 , 0.78028548, 0.78596517, 0.78902922, 0.79005979])

In [133…
```
y_pred9=gb1.predict(X_test_ss)
df5=pd.DataFrame(X_test_ss)
df5['Prediction']=y_pred9
df5
```

Out[133…

	0	1	2	3	4	Prediction
0	-1.018648	-1.040520	0.136643	0.623883	-0.072304	0
1	0.019381	-1.040520	0.136643	0.623883	-0.697499	0
2	3.380968	2.481709	-1.957570	-1.602865	0.726556	1
3	0.368898	1.258712	0.136643	0.623883	-1.878422	0
4	0.403685	0.391601	-0.282199	0.623883	-1.982621	0
...	...	...	...	...	...	...
14556	-1.018648	-1.040520	0.136643	0.623883	0.726556	0
14557	-0.544192	-0.085753	0.974328	-1.602865	-0.072304	0
14558	0.735889	1.560864	0.136643	0.623883	1.317017	1
14559	0.164281	-0.053811	0.136643	0.623883	-0.384901	0
14560	-0.070199	1.066149	-1.957570	-1.602865	0.726556	1

14561 rows × 6 columns

In [134…
```
df5['Prediction'].value_counts()
```

Out[134…
```
0    11854
1     2707
Name: Prediction, dtype: int64
```

In [135…
```
print(classification_report(y_test,y_pred9))
```

```
              precision    recall  f1-score   support

           0       0.99      0.82      0.90     14346
           1       0.05      0.67      0.10       215

    accuracy                           0.82     14561
   macro avg       0.52      0.74      0.50     14561
weighted avg       0.98      0.82      0.89     14561
```

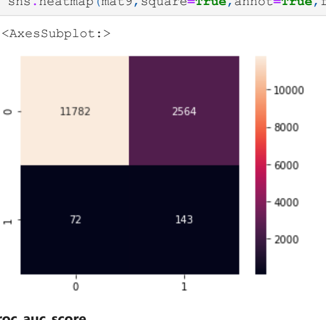
In [136…
```
print("Train Accuracy:",gb1.score(X_sample2, y_sample2))
print("Test Accuracy:",gb1.score(X_test_ss, y_test))
```

```
Train Accuracy: 0.789967714934832
Test Accuracy: 0.8189684774397363
```

From above fig we cn see our model accuracy with gimi index is 82%

```
In [137…
mat9=confusion_matrix(y_test,y_pred9)
sns.heatmap(mat9,square=True,annot=True,fmt='d')
```

```
Out[137… <AxesSubplot:>
```



roc_auc_score

```
In [138…
scores=gb1.predict_proba(X_test_ss)
```

```
In [139…
print(scores)
```

```
[[0.91943705 0.08056295]
 [0.83979365 0.16020635]
 [0.12315918 0.87684082]
 ...
 [0.41687725 0.58312275]
 [0.64941052 0.35058948]
 [0.13712463 0.86287537]]
```

```
In [140…
roc_auc_score(y_test,y_pred9)
```

```
Out[140… 0.7431952509248183
```

Model 5= Adaboost Classifier

```
In [141…
from sklearn.ensemble import AdaBoostClassifier
aba1=AdaBoostClassifier(n_estimators=100)
```

```
In [142…
aba1.fit(X_sample2,y_sample2)
```

```
Out[142… AdaBoostClassifier(n_estimators=100)
```

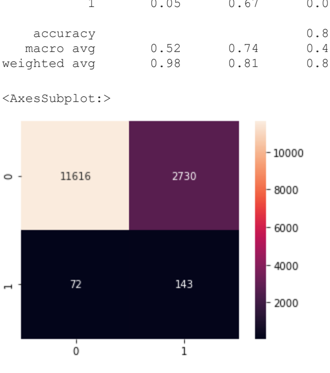
```
In [143…
score=cross_val_score(aba1,X_sample2,y_sample2,cv=5)
score
```

```
Out[143… array([0.76915029, 0.76608624, 0.76339586, 0.76787983, 0.76674141])
```

```
In [144…
y_pred11=aba1.predict(X_test_ss)
print(classification_report(y_test,y_pred11))
mat11=confusion_matrix(y_test,y_pred11)
sns.heatmap(mat11,square=True,annot=True,fmt="d")
```

```
              precision    recall  f1-score   support

           0       0.99      0.81      0.89     14346
           1       0.05      0.67      0.09       215

    accuracy                           0.81     14561
   macro avg       0.52      0.74      0.49     14561
weighted avg       0.98      0.81      0.88     14561
```

```
Out[144… <AxesSubplot:>
```



From above fig we cn see our model accuracy with gimi index is 82%

roc_auc_score

```
In [145…
scores=aba1.predict_proba(X_test_ss)
scores=aba1.predict_proba(X_test_ss)
```

```
In [146…
print(scores)
```

```
[[0.50735607 0.49264393]
 [0.50528433 0.49471567]
 [0.49455375 0.50544625]
 ...
 [0.50002849 0.49997151]
 [0.50190542 0.49809458]
 [0.49492806 0.50507194]]
```

```
In [147…
roc_auc_score(y_test,y_pred11)
```

```
Out[147… 0.7374096660928093
```

Model 6 =SVC

```
In [148…
lsvc1=LinearSVC()
```

```
In [149…
lsvc1.fit(X_sample2,y_sample2)
```

```
Out[149… LinearSVC()
```

```
In [150…
score=cross_val_score(lsvc1,X_sample2,y_sample2,cv=5)
```

```
In [151…
score
```

```
Out[151… array([0.74164861, 0.74022868, 0.74022868, 0.74336746, 0.74050822])
```

```
In [152…
y_pred17=lsvc1.predict(X_test_ss)
```

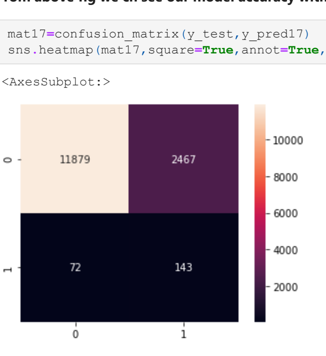
```
In [153…
print(classification_report(y_test,y_pred17))
```

```
              precision    recall  f1-score   support

           0       0.99      0.83      0.90     14346
           1       0.05      0.67      0.10       215

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

From above fig we cn see our model accuracy with gimi index is 83%

```
In [154…
mat17=confusion_matrix(y_test,y_pred17)
sns.heatmap(mat17,square=True,annot=True,fmt='d')
```

```
Out[154… <AxesSubplot:>
```



Soft Margin

```
In [155…
lsv1=LinearSVC(C=0.5,random_state=1)
```

```
In [156…
lsv1.fit(X_sample2,y_sample2)
```

```
Out[156… LinearSVC(C=0.5, random_state=1)
```

```
In [157…
score=cross_val_score(lsv1,X_sample2,y_sample2,cv=5)
```

```
In [158…
score
```

```
Out[158… array([0.74164861, 0.74022868, 0.74022868, 0.74336746, 0.74050822])
```

```
In [159…
y_pred18=lsv1.predict(X_test_ss)
```

```
In [160…
print(classification_report(y_test,y_pred18))
```

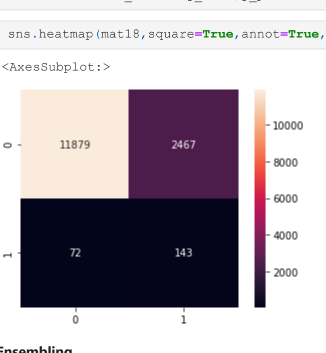
```
              precision    recall  f1-score   support

           0       0.99      0.83      0.90     14346
           1       0.05      0.67      0.10       215

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

From above fig we cn see our model accuracy with gimi index is 82%

```
In [161…
mat18=confusion_matrix(y_test,y_pred18)
```

```
In [162…
sns.heatmap(mat18,square=True,annot=True,fmt='d')
```

```
Out[162… <AxesSubplot:>
```



Ensembling

Hard voting

```
In [163…
from sklearn.ensemble import VotingClassifier
```

```
In [164…
model_list1 = [("lr1",lr1),("dt1",dt1),("dt3",dt3),("rfc1",rfc1),("gb1",gb1),("aba1",aba1),("lsvc1",lsvc1)]
```

```
In [165…
vc1=VotingClassifier(estimators=model_list1)
vc1.fit(X_sample2,y_sample2)
```

```
Out[165… VotingClassifier(estimators=[('lr1', LogisticRegression()),
                              ('dt1',
                               DecisionTreeClassifier(max_depth=20,
                                                      min_samples_leaf=35)),
                              ('dt3',
                               DecisionTreeClassifier(criterion='entropy',
                                                      max_depth=25,
                                                      min_samples_leaf=30)),
                              ('rfc1', RandomForestClassifier(random_state=1)),
                              ('gb1', GradientBoostingClassifier()),
                              ('aba1', AdaBoostClassifier(n_estimators=100)),
                              ('lsvc1', LinearSVC())])
```

```
In [166…
vc3=VotingClassifier(estimators=model_list1)
```

```
In [167…
vc3.fit(X_sample2,y_sample2)
y_pred22=vc1.predict(X_test_ss)
print(classification_report(y_test,y_pred22))
```

```
              precision    recall  f1-score   support

           0       0.99      0.85      0.92     14346
           1       0.06      0.62      0.11       215

    accuracy                           0.85     14561
   macro avg       0.53      0.74      0.51     14561
weighted avg       0.98      0.85      0.91     14561
```

From above fig we cn see our model accuracy with gimi index is 85%

soft voting

```
In [168…
vc4=VotingClassifier(estimators=model_list1,voting="soft")
```

```
In [169…
vc4.fit(X_sample2,y_sample2)
y_pred23=vc1.predict(X_test_ss)
```

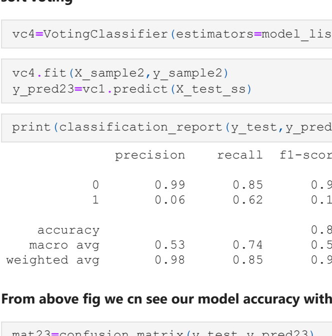
```
In [170…
print(classification_report(y_test,y_pred23))
```

```
              precision    recall  f1-score   support

           0       0.99      0.85      0.92     14346
           1       0.06      0.62      0.11       215

    accuracy                           0.85     14561
   macro avg       0.53      0.74      0.51     14561
weighted avg       0.98      0.85      0.91     14561
```

From above fig we cn see our model accuracy with gimi index is 85%

```
In [171…
mat23=confusion_matrix(y_test,y_pred23)
sns.heatmap(mat23,square=True,annot=True,fmt='d')
```

```
Out[171… <AxesSubplot:>
```



ConclusionIn In this project, I have used different ways of addressing the task with unbalanced data.Like Supervisedlearning LinearSVC, Adaboost, GradientBoosting ,DecisionTreeClassifier,Logistic Regression and RandomForest.As per their's result I conclude that Random forest is the best algorithm model since it's giving highaccuracy than other's. The highest accuracy is 95%

```
In [ ]:
```