In [5]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

New Section

In [6]:
```python
df=pd.read_csv("CarClaim.csv")
df.head()
```

Out[6]:

	claim_number	age_of_driver	gender	marital_status	safty_rating	annual_income	high_educatio
0	1	46	M	1.0	85	38301	
1	3	21	F	0.0	75	30445	
2	4	49	F	0.0	87	38923	
3	5	58	F	1.0	58	40605	
4	6	38	M	1.0	95	36380	

5 rows × 25 columns

In [6]:

Data Preprocessing Part 1

In [7]:
```python
#Remove identifier column
df=df.drop(["claim_number","zip_code"],axis=1)
df.head()
```

Out[7]:

	age_of_driver	gender	marital_status	safty_rating	annual_income	high_education_ind	address_
0	46	M	1.0	85	38301	1	
1	21	F	0.0	75	30445	0	
2	49	F	0.0	87	38923	0	
3	58	F	1.0	58	40605	1	
4	38	M	1.0	95	36380	1	

5 rows × 23 columns

In [8]:
```python
#Check the number of unique value
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 17998 entries, 0 to 17997
Data columns (total 23 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   age_of_driver            17998 non-null  int64  
 1   gender                   17998 non-null  object 
 2   marital_status           17993 non-null  float64
 3   safty_rating             17998 non-null  int64  
 4   annual_income            17998 non-null  int64  
 5   high_education_ind       17998 non-null  int64  
 6   address_change_ind       17998 non-null  int64  
 7   living_status            17998 non-null  object 
 8   claim_date               17998 non-null  object 
 9   claim_day_of_week        17998 non-null  object 
 10  accident_site            17998 non-null  object 
 11  past_num_of_claims       17998 non-null  int64  
 12  witness_present_ind      17866 non-null  float64
 13  liab_prct                17998 non-null  int64  
 14  channel                  17998 non-null  object 
 15  policy_report_filed_ind  17998 non-null  int64  
 16  claim_est_payout         17981 non-null  float64
 17  age_of_vehicle           17990 non-null  float64
 18  vehicle_category         17998 non-null  object 
 19  vehicle_price            17998 non-null  float64
 20  vehicle_color            17998 non-null  object 
 21  vehicle_weight           17998 non-null  float64
 22  fraud                    17998 non-null  int64  
dtypes: float64(6), int64(9), object(8)
memory usage: 3.2+ MB
```

In [9]:
```python
df.select_dtypes("object").nunique()
```

Out[9]:
```
gender                 2
living_status          2
claim_date           731
claim_day_of_week      7
accident_site          3
channel                3
vehicle_category       3
vehicle_color          7
dtype: int64
```

In [10]:
```python
#extract year on clain-date
df["claim_date"]=df["claim_date"].str[6:].astype(int)
df.head()
```

Out[10]:

	age_of_driver	gender	marital_status	safty_rating	annual_income	high_education_ind	address_
0	46	M	1.0	85	38301	1	
1	21	F	0.0	75	30445	0	
2	49	F	0.0	87	38923	0	
3	58	F	1.0	58	40605	1	
4	38	M	1.0	95	36380	1	

5 rows × 23 columns

In [11]:
```python
#Again checking the numbr of unique value from all the object datatype
df.select_dtypes(include="object").nunique()
```

Out[11]:
```
gender               2
living_status        2
claim_day_of_week    7
accident_site        3
channel              3
vehicle_category     3
vehicle_color        7
dtype: int64
```

In [12]:
```python
#Replace 1 with "yes" and 0 with "no" in the categorical column
df["fraud"]=df["fraud"].replace({0:"no",1:"yes"})
df["high_education_ind"]=df["high_education_ind"].replace({0:"no",1:"yes"})
df["marital_status"]=df["marital_status"].replace({0:"no",1:"yes"})
df["address_change_ind"]=df["address_change_ind"].replace({0:"no",1:"yes"})
df["policy_report_filed_ind"]=df["policy_report_filed_ind"].replace({0:"no",1:"yes'
df["witness_present_ind"]=df["witness_present_ind"].replace({0:"no",1:"yes"})
```

In [13]:
```python
df.head()
```

Out[13]:

	age_of_driver	gender	marital_status	safty_rating	annual_income	high_education_ind	address_
0	46	M	yes	85	38301	yes	
1	21	F	no	75	30445	no	
2	49	F	no	87	38923	no	
3	58	F	yes	58	40605	yes	
4	38	M	yes	95	36380	yes	

5 rows × 23 columns

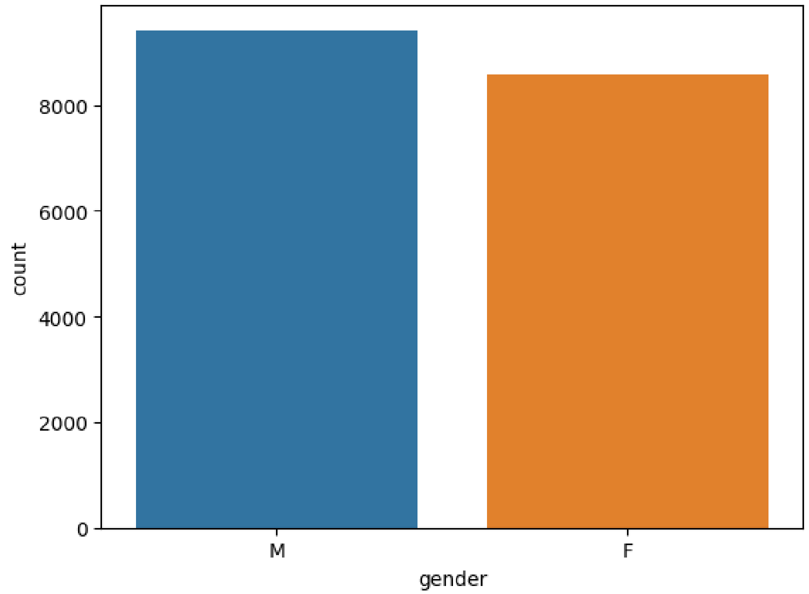
In [13]:

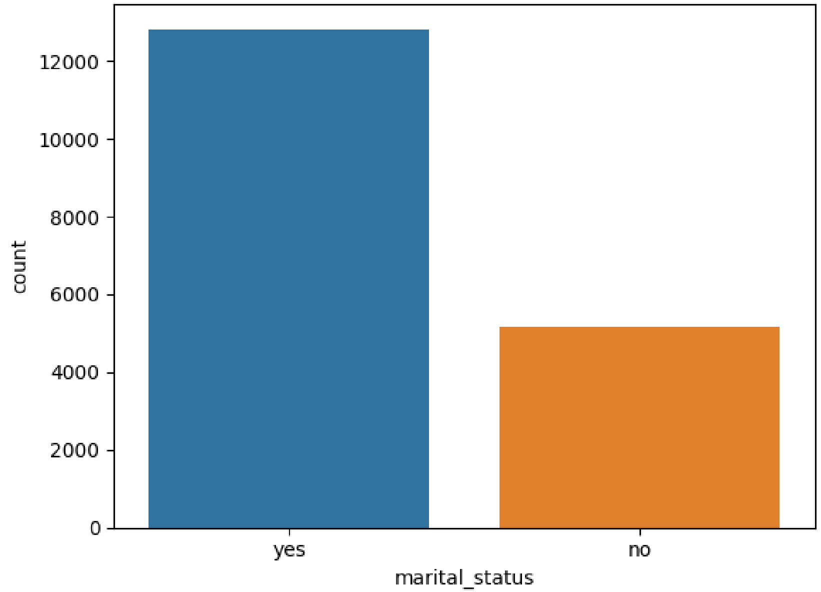
Exploratory Data Analysis

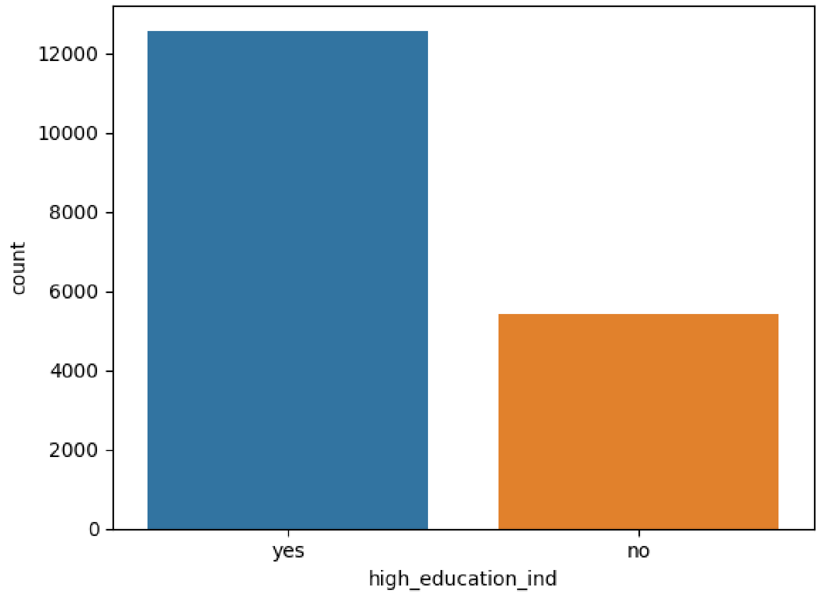
In [14]:
```python
#Get the names of all the columns with data type "object"
categorical=df.select_dtypes(include="object").columns.tolist()
categorical
```

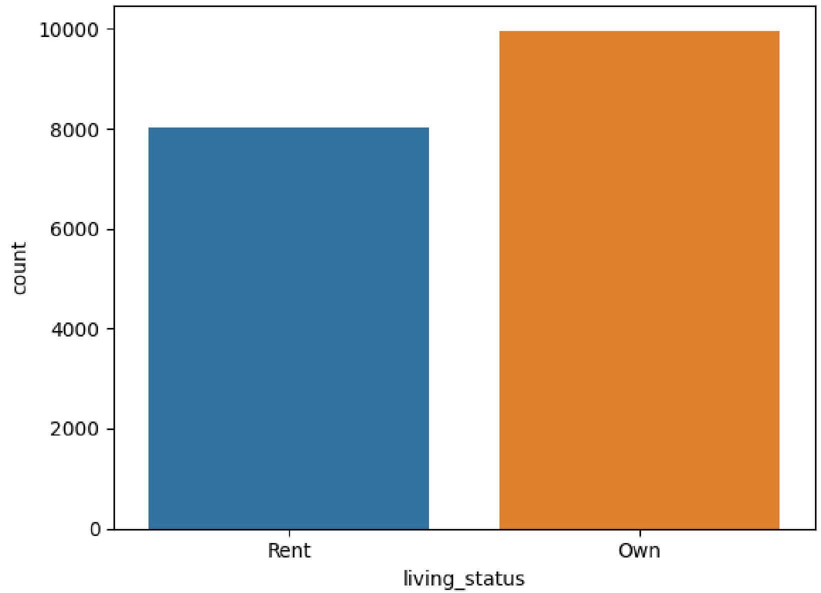
Out[14]:
```
['gender',
 'marital_status',
 'high_education_ind',
 'address_change_ind',
 'living_status',
 'claim_day_of_week',
 'accident_site',
 'witness_present_ind',
 'channel',
 'policy_report_filed_ind',
 'vehicle_category',
 'vehicle_color',
 'fraud']
```

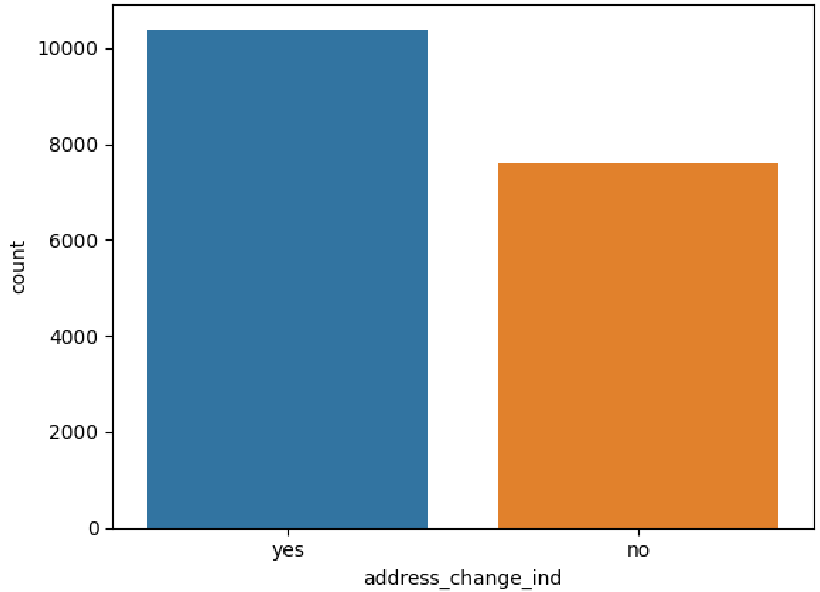
In [15]:
```python
#Create a countplot for each categorical variable using seaborn
for i in categorical:
  sns.countplot(x=i,data=df)

  plt.show()
```

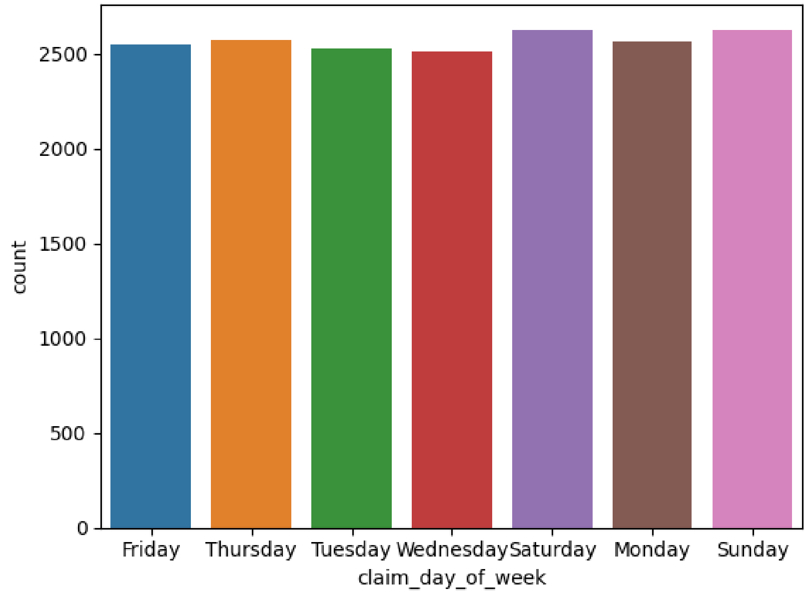


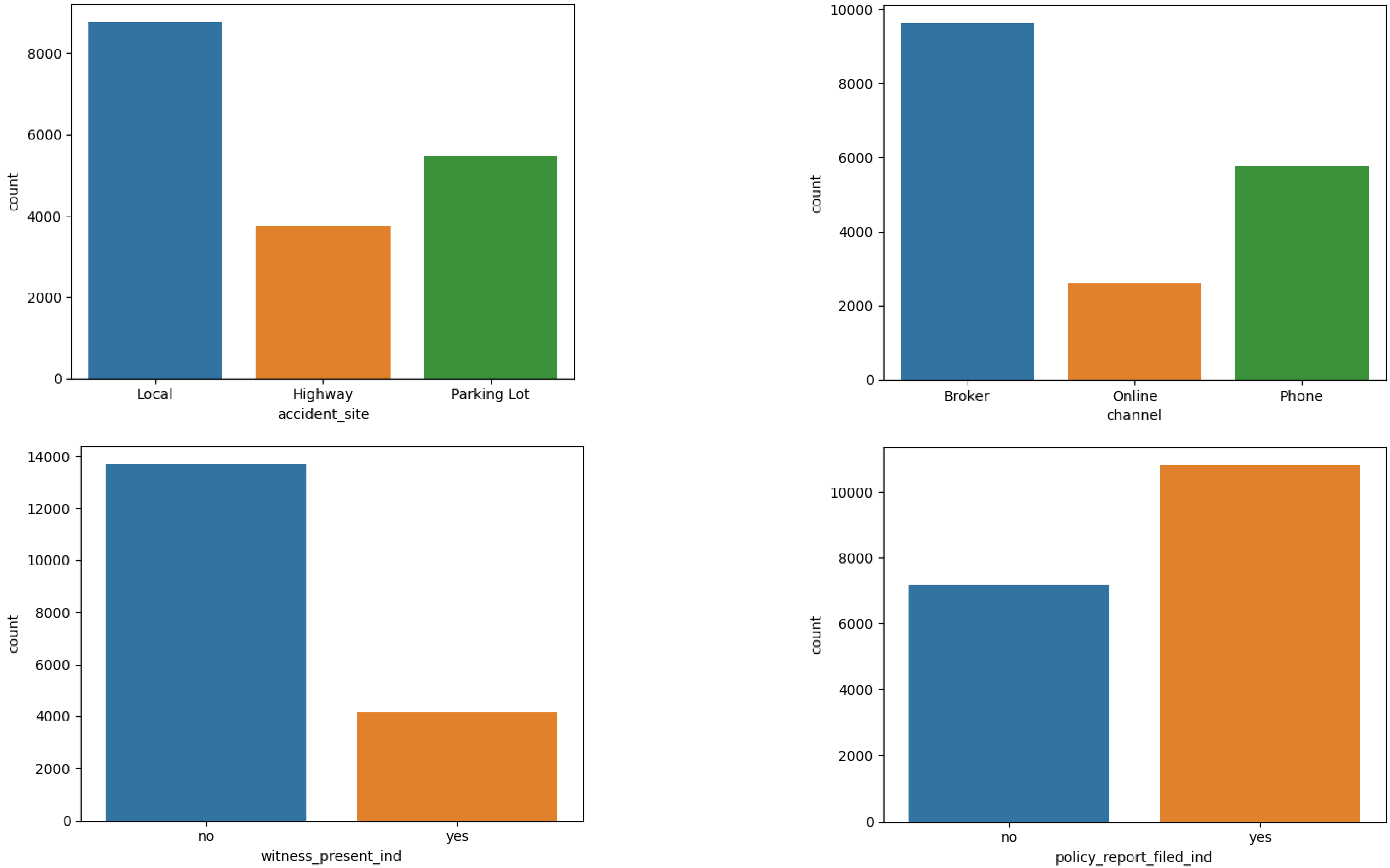


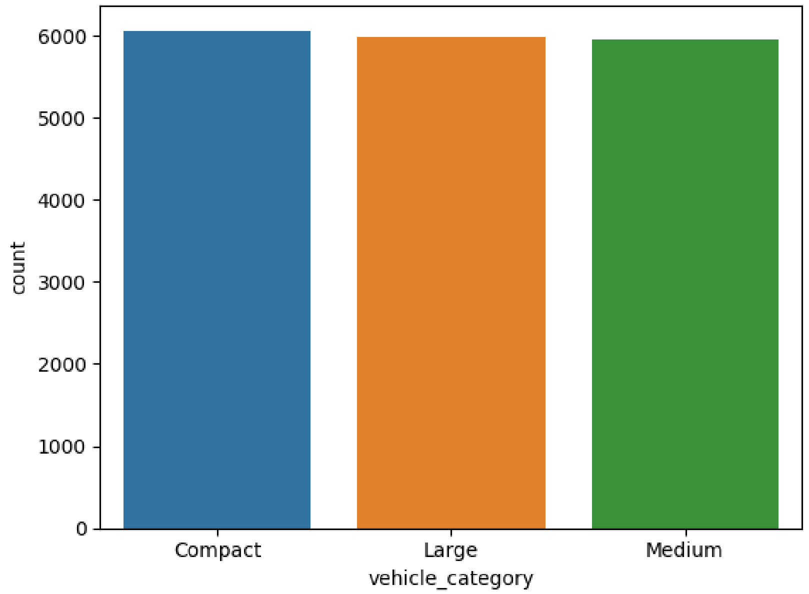

count
0
2000
4000
6000
8000
10000
12000
yes
no
high_education_ind

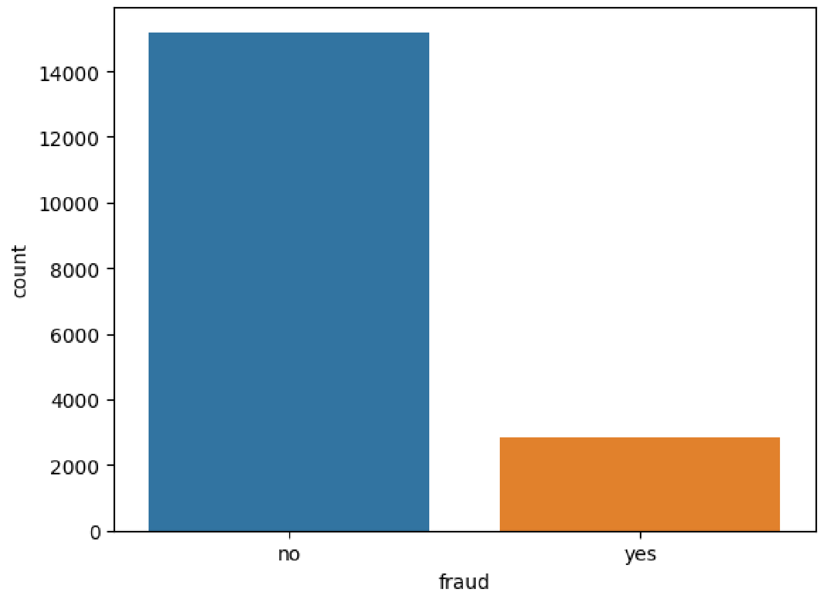

count
0
2000
4000
6000
8000
10000
Rent
Own
living_status

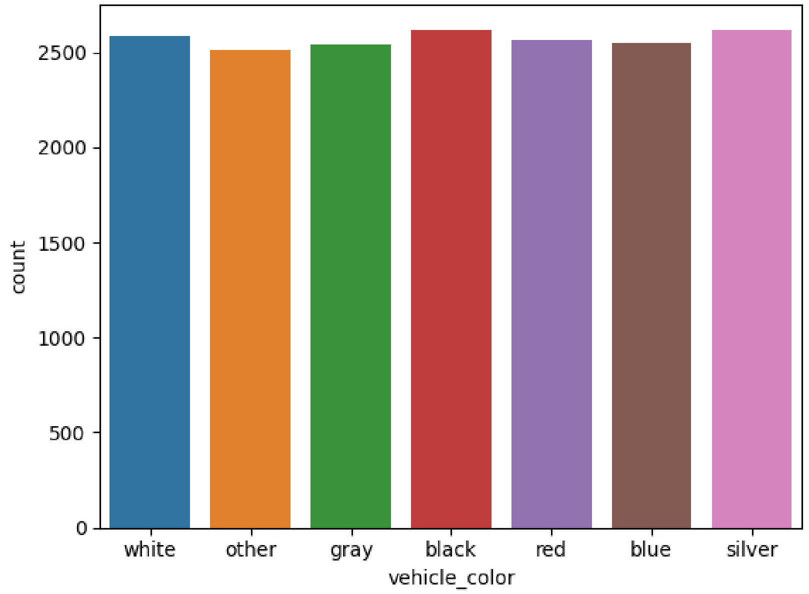

count
0
2000
4000
6000
8000
10000
yes
no
address_change_ind


count
0
500
1000
1500
2000
2500
Friday
Thursday
Tuesday
Wednesday
Saturday
Monday
Sunday
claim_day_of_week


count
accident_site
Local
Highway
Parking Lot
channel
Broker
Online
Phone
witness_present_ind
no
yes
policy_report_filed_ind



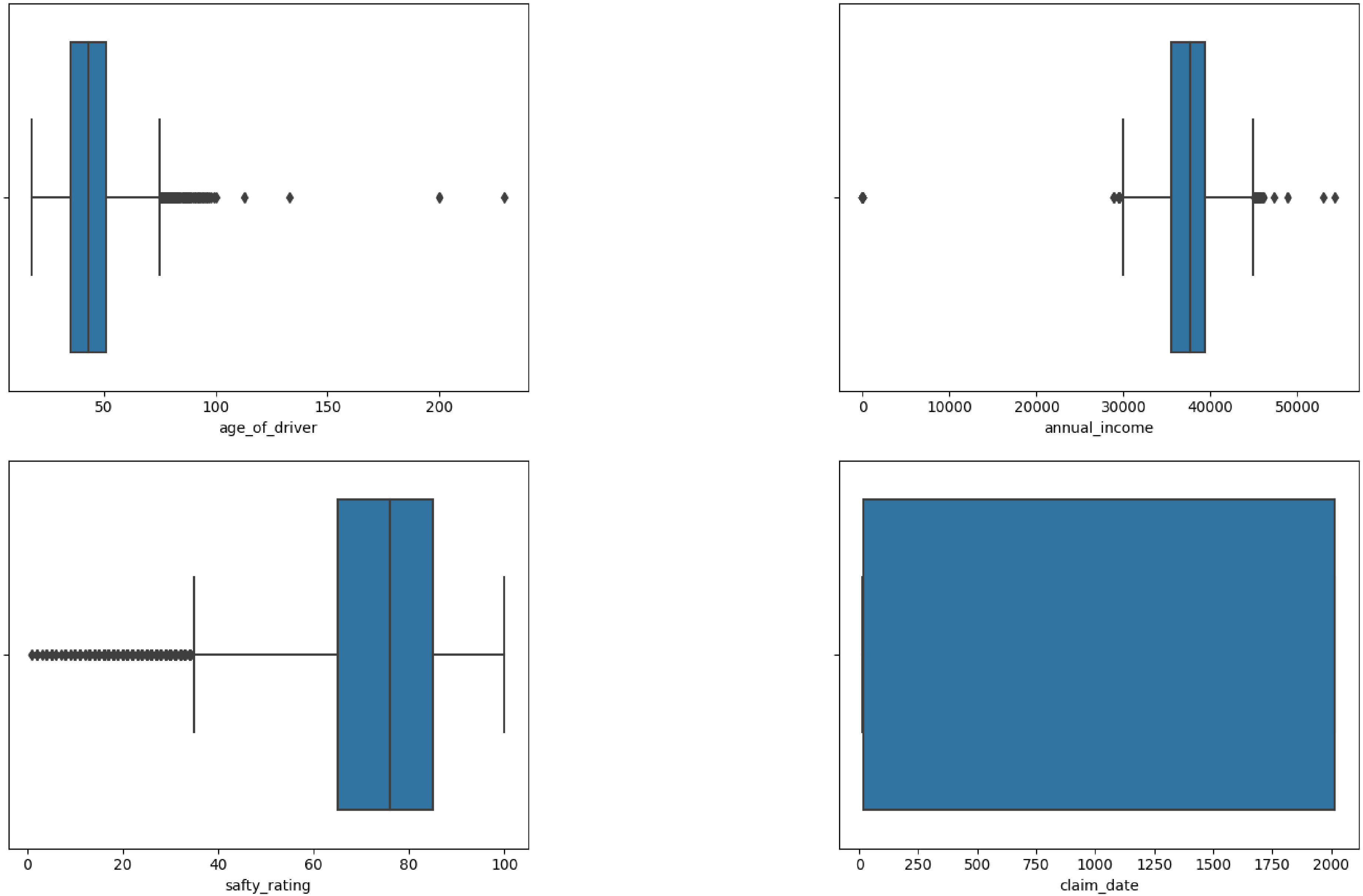


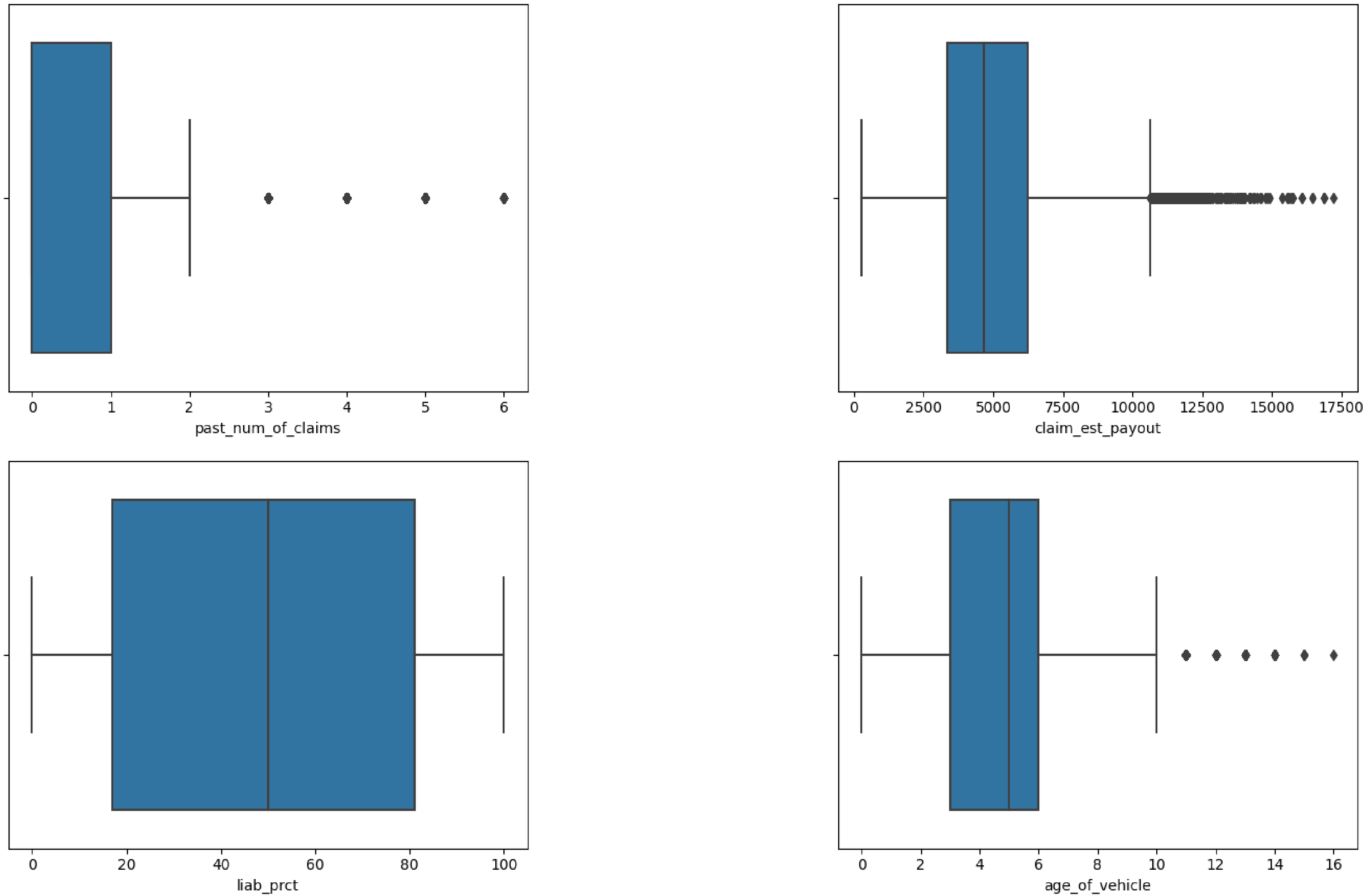


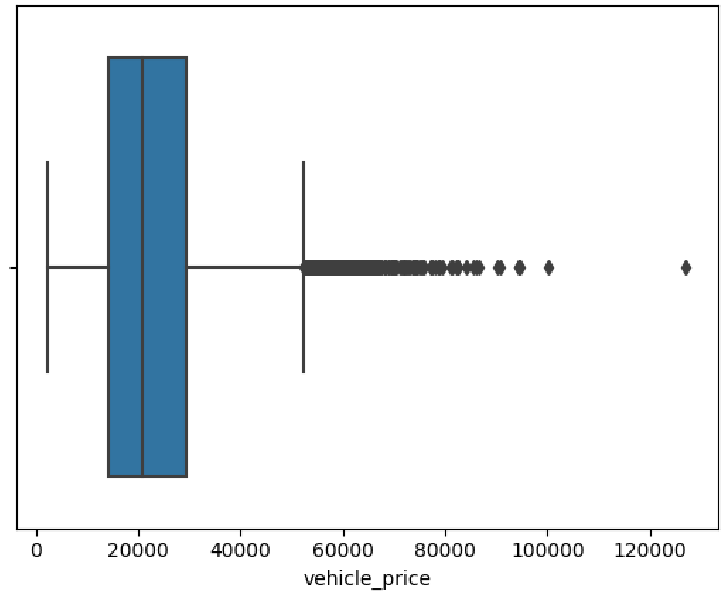
In [16]:
```
#Get the names of all the columns with data type "int"
numerical=df.select_dtypes(["int","float"]).columns.tolist()
numerical
```

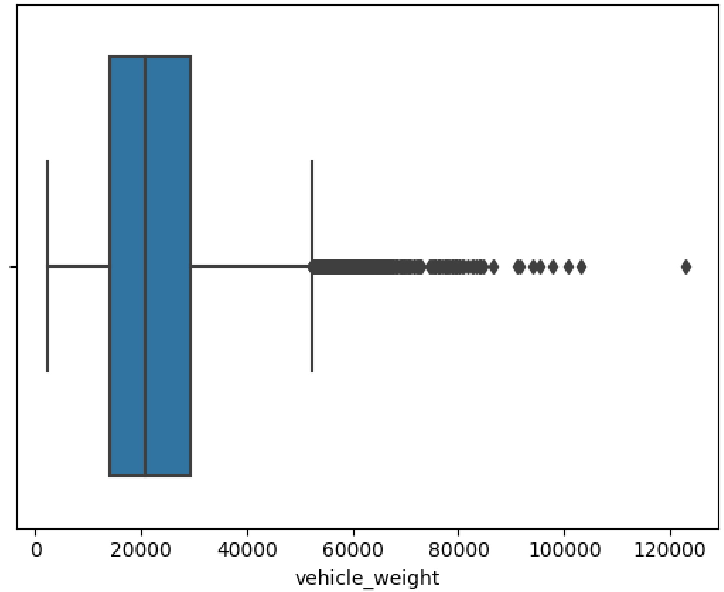
Out[16]:
```
['age_of_driver',
 'safty_rating',
 'annual_income',
 'claim_date',
 'past_num_of_claims',
 'liab_prct',
 'claim_est_payout',
 'age_of_vehicle',
 'vehicle_price',
 'vehicle_weight']
```

In [17]:
```
#Creating a box plot
for i in numerical:
  sns.boxplot(x=i,data=df)
  plt.show()
```

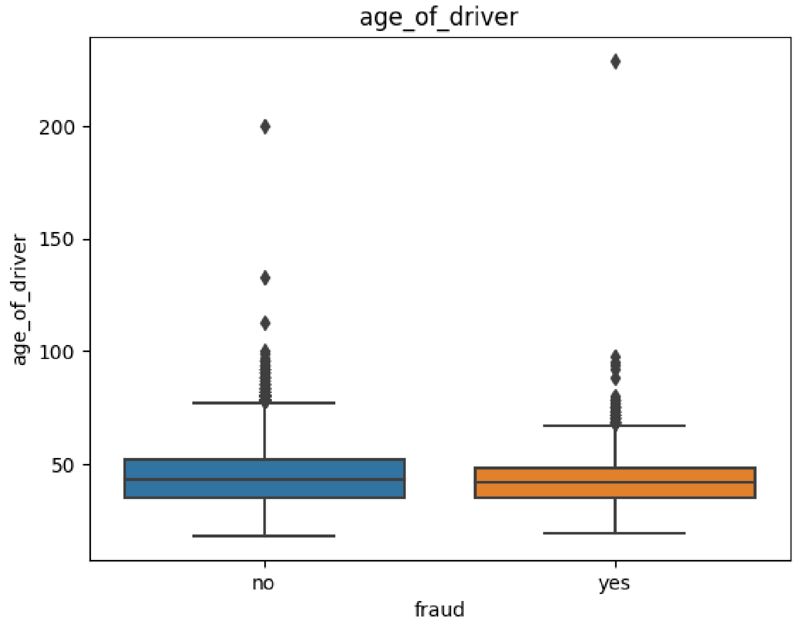

50 100 150 200
age_of_driver
0 10000 20000 30000 40000 50000
annual_income
0 20 40 60 80 100
safty_rating
0 250 500 750 1000 1250 1500 1750 2000
claim_date

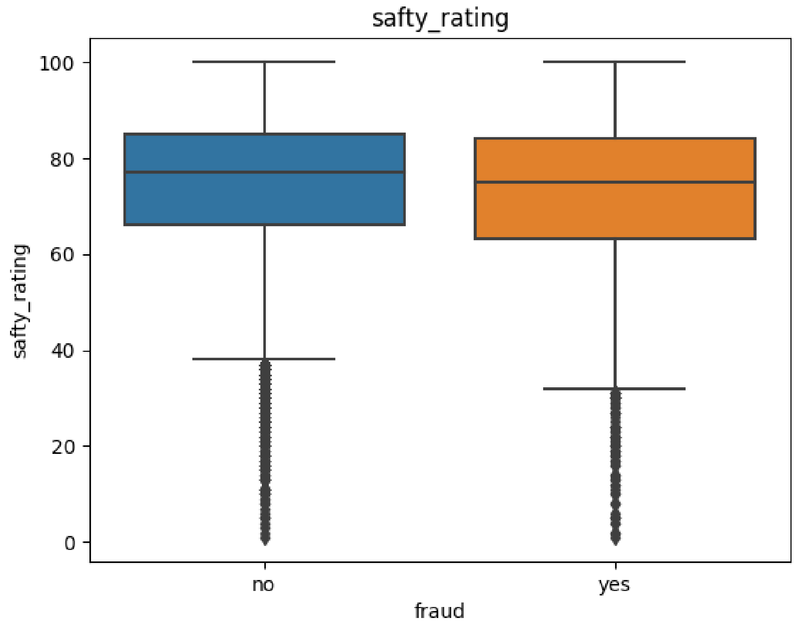

0 1 2 3 4 5 6
past_num_of_claims
0 2500 5000 7500 10000 12500 15000 17500
claim_est_payout
0 20 40 60 80 100
liab_prct
0 2 4 6 8 10 12 14 16
age_of_vehicle

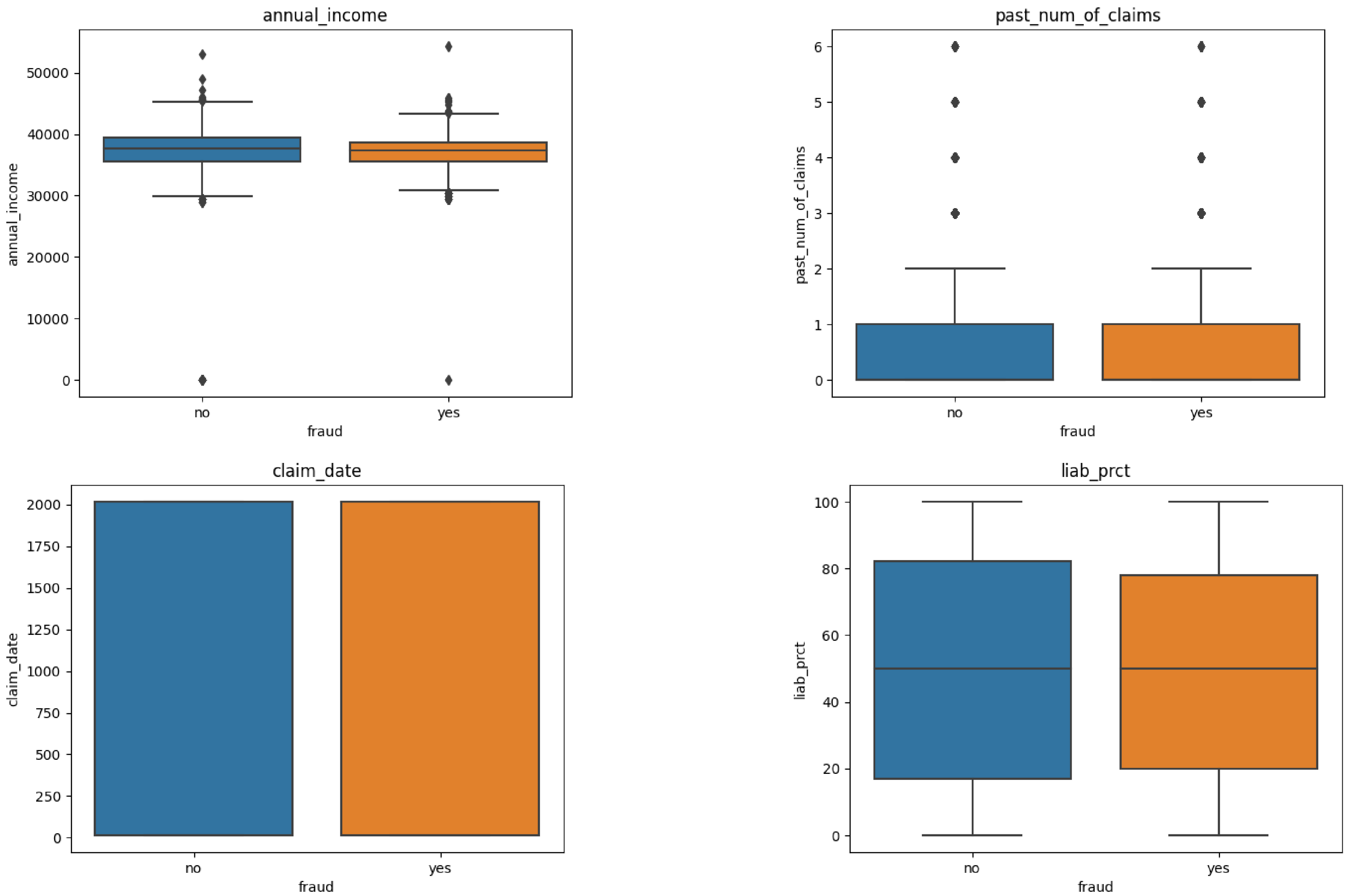




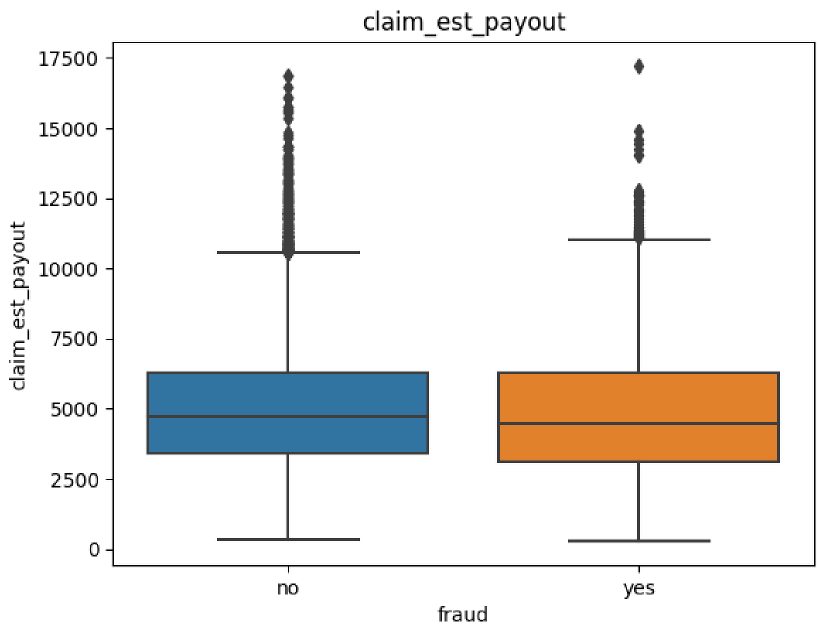
In [18]:
```
for i in numerical:
  sns.boxplot(y=i,x="fraud",data=df)
  plt.title(i)
  plt.show()
```

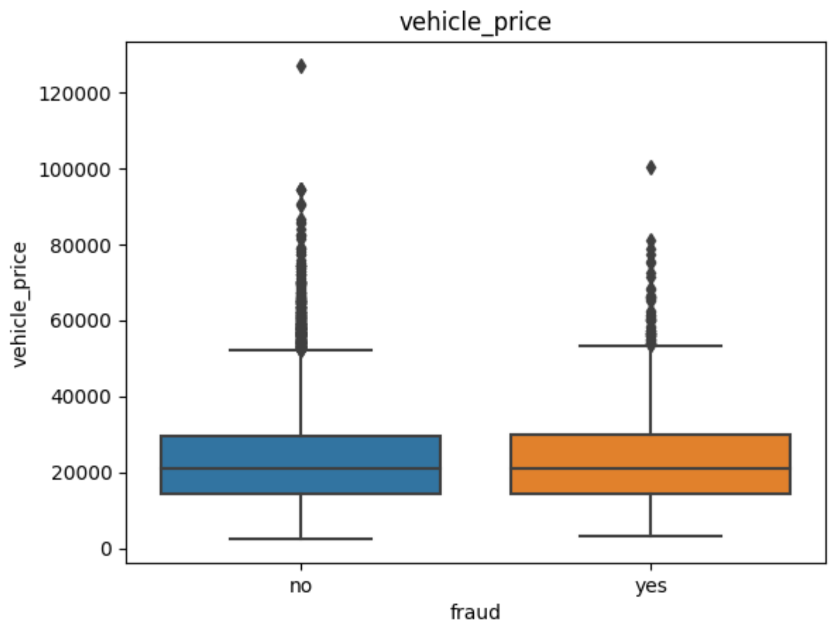


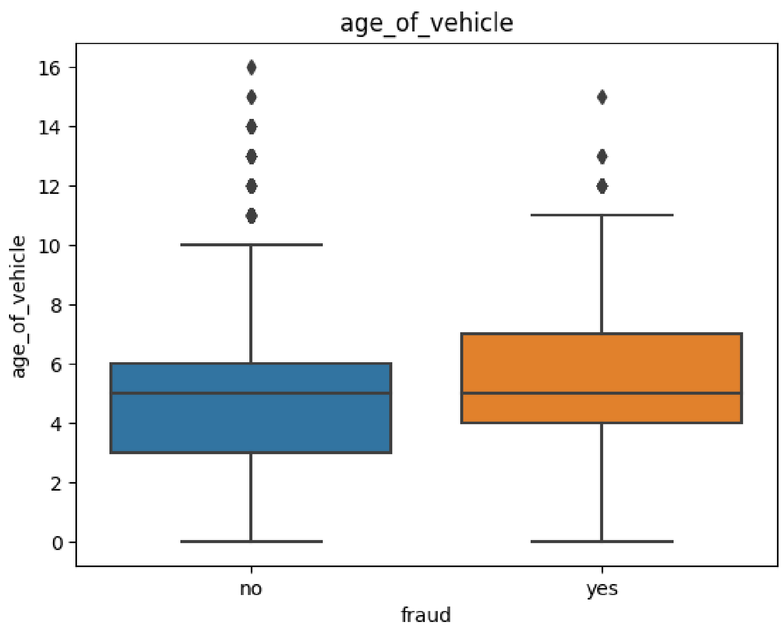


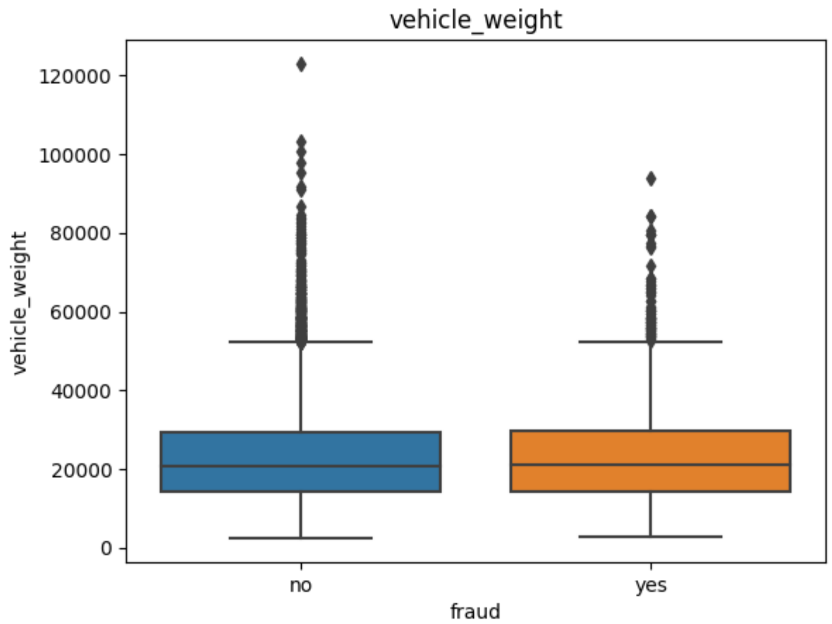


annual_income
50000
40000
30000
20000
10000
0
annual_income
no
yes
fraud
past_num_of_claims
6
5
4
3
2
1
0
past_num_of_claims
no
yes
fraud
claim_date
2000
1750
1500
1250
1000
750
500
250
0
claim_date
no
yes
fraud
liab_prct
100
80
60
40
20
0
liab_prct
no
yes
fraud

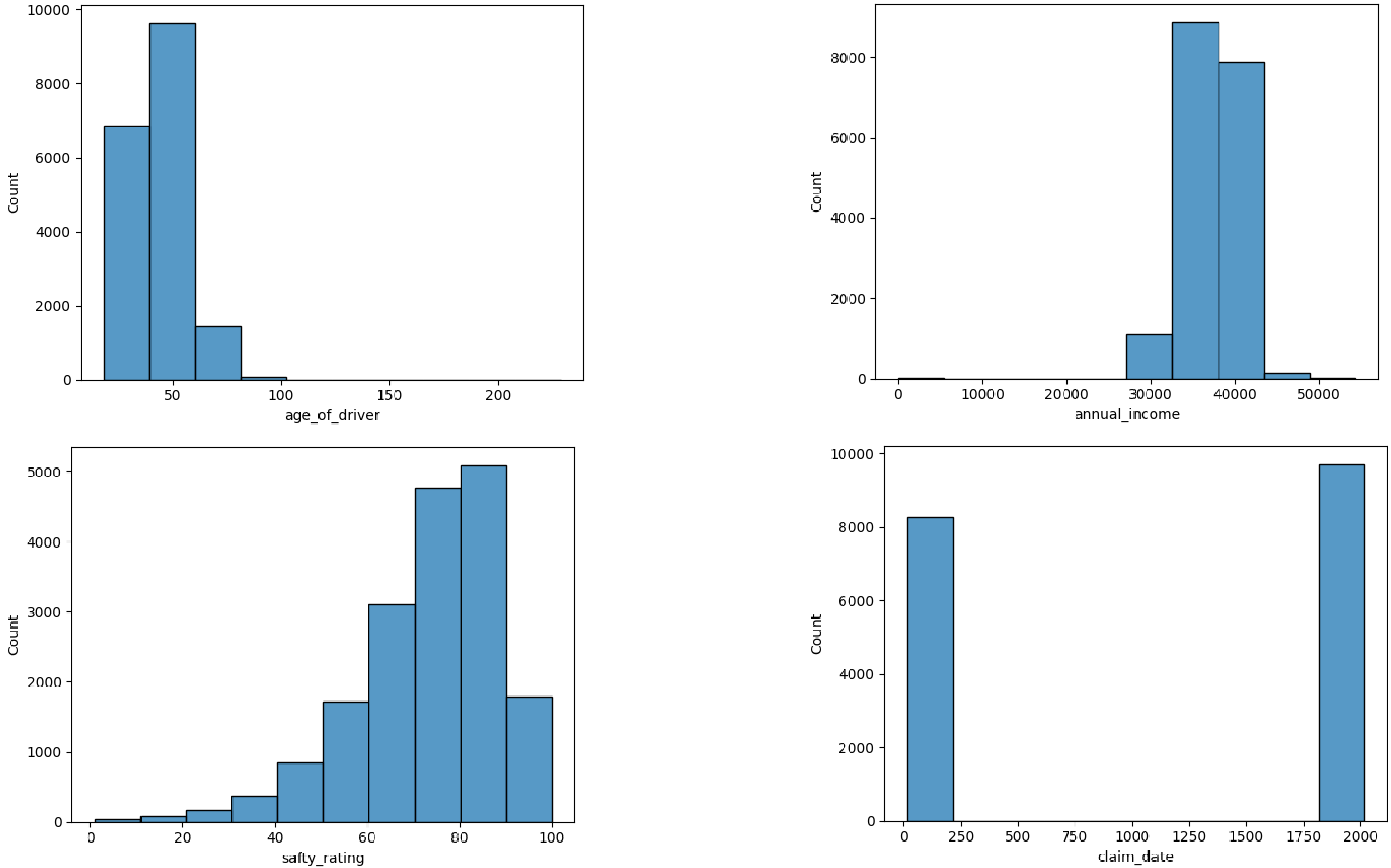


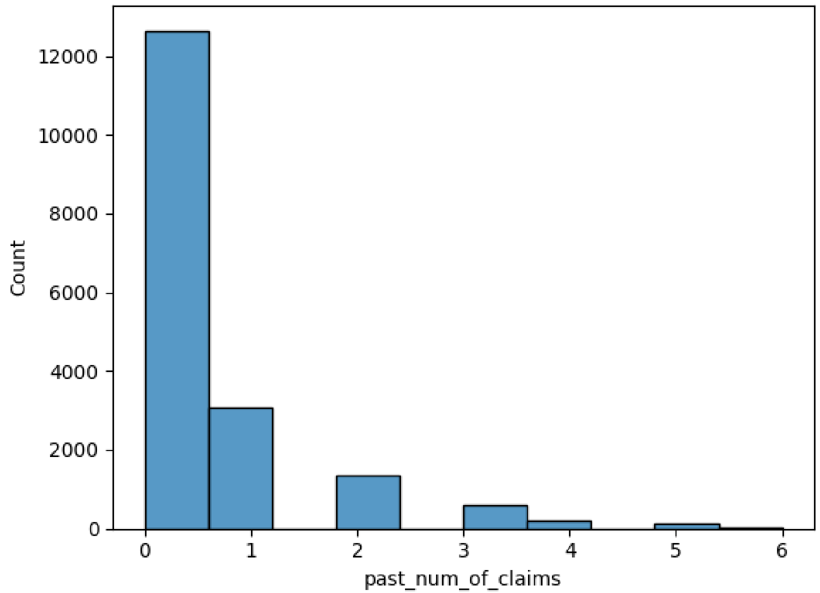




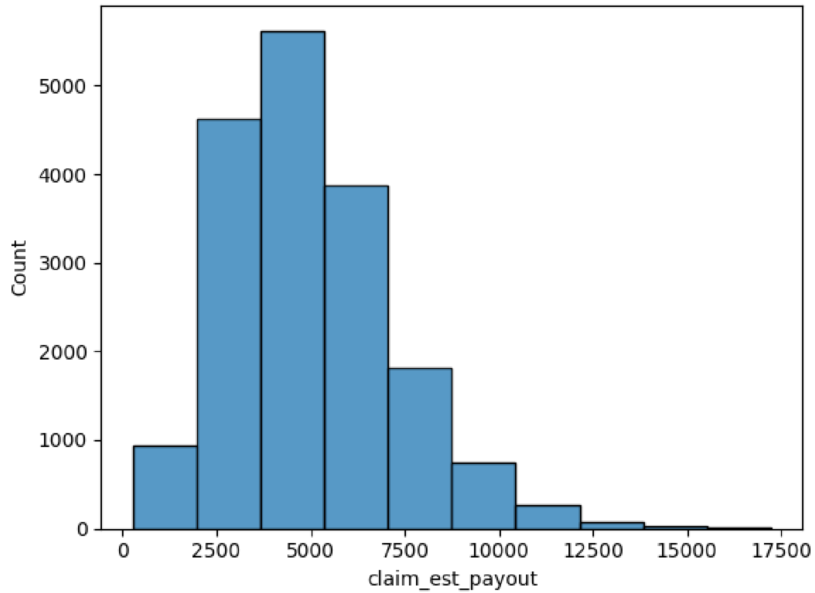


In [19]:
```
#Create a histogram for each integer variable
for i in numerical:
  sns.histplot(df[i],bins=10)
  plt.show()
```

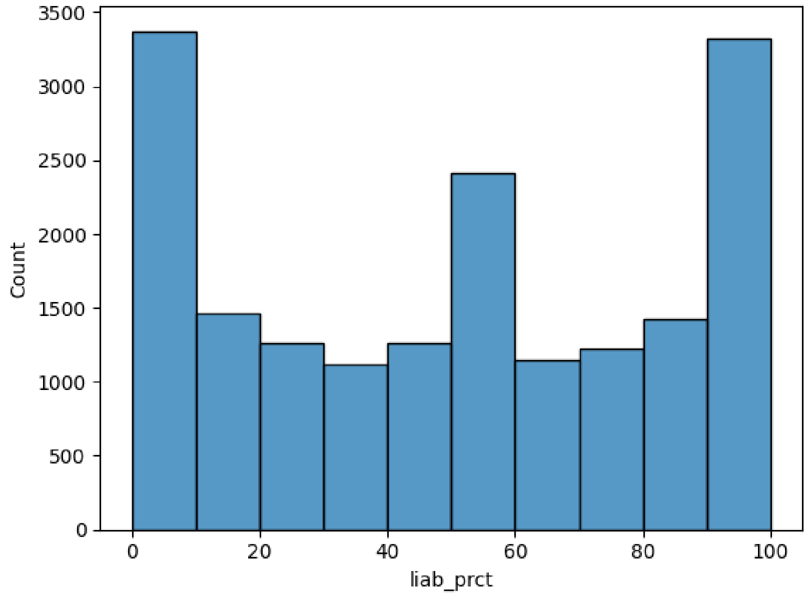

Count
age_of_driver
Count
annual_income
Count
safty_rating
Count
claim_date



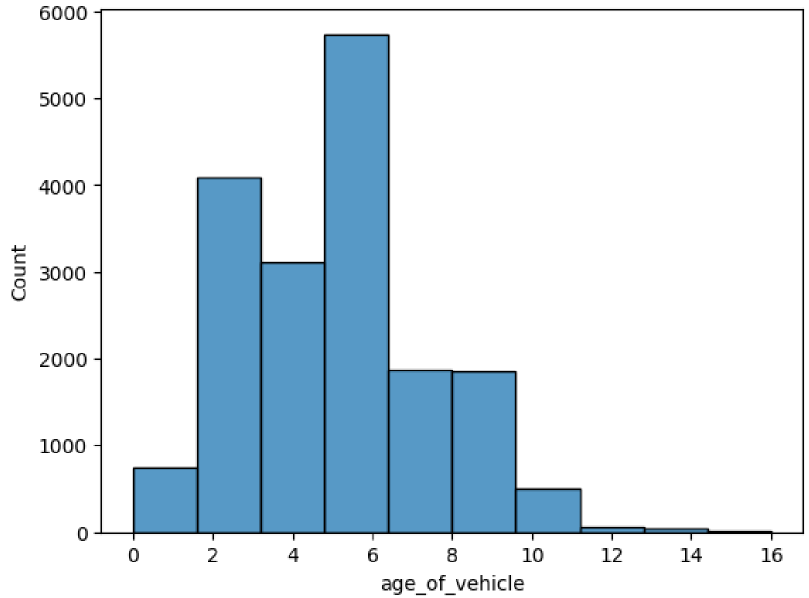
Count
past_num_of_claims
0
2000
4000
6000
8000
10000
12000
0
1
2
3
4
5
6



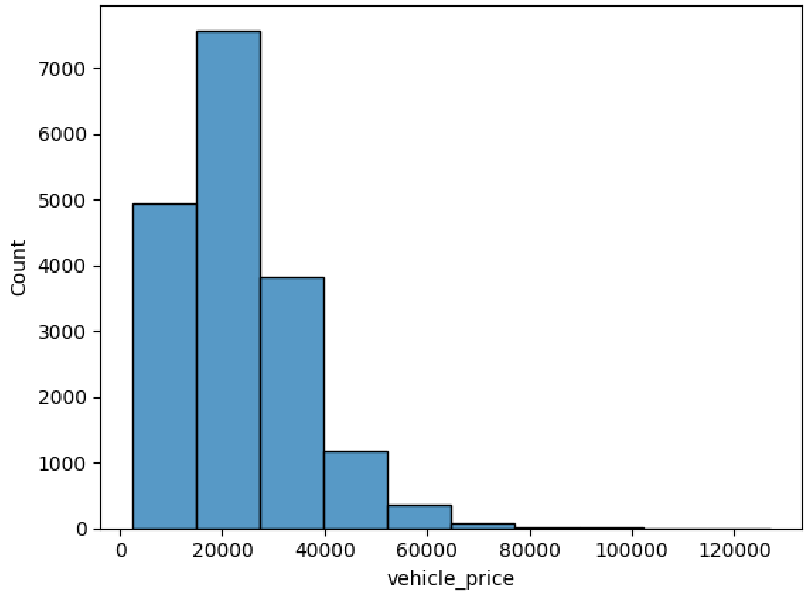
Count
claim_est_payout
0
1000
2000
3000
4000
5000
0
2500
5000
7500
10000
12500
15000
17500

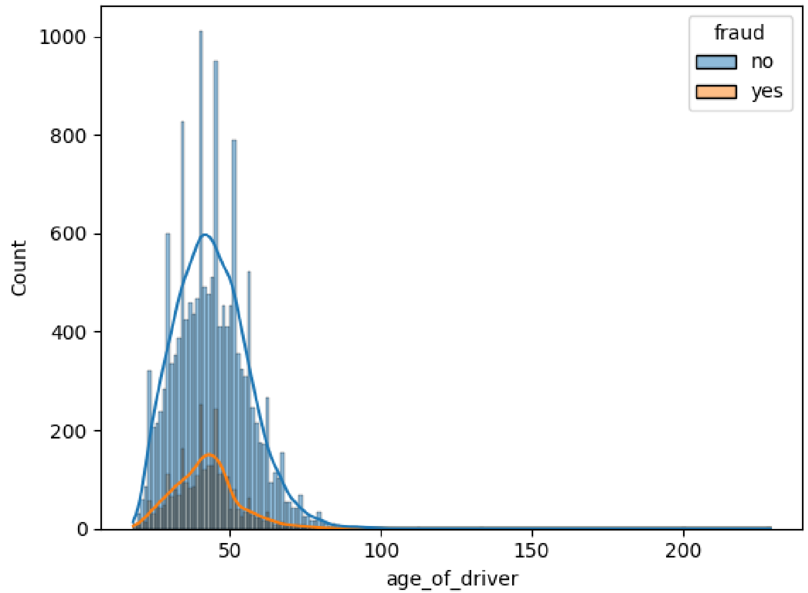


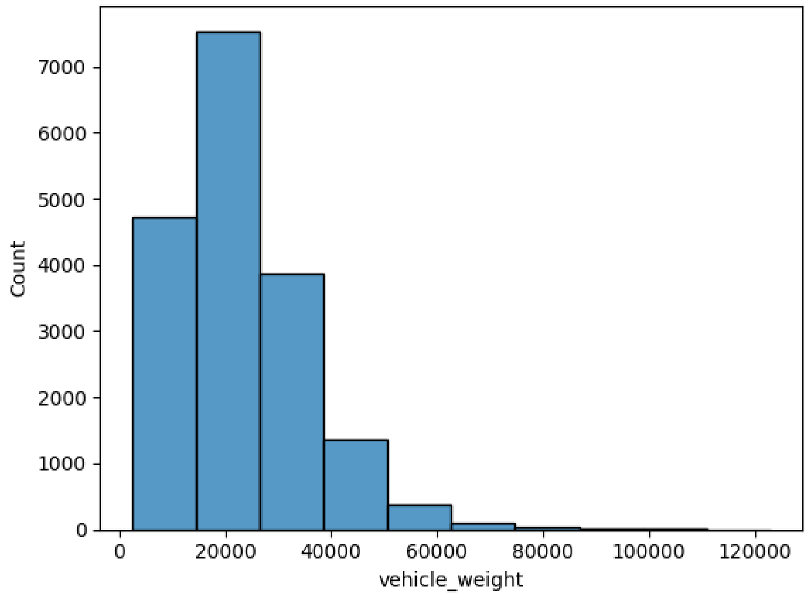
Count
liab_prct
0
500
1000
1500
2000
2500
3000
3500
0
20
40
60
80
100

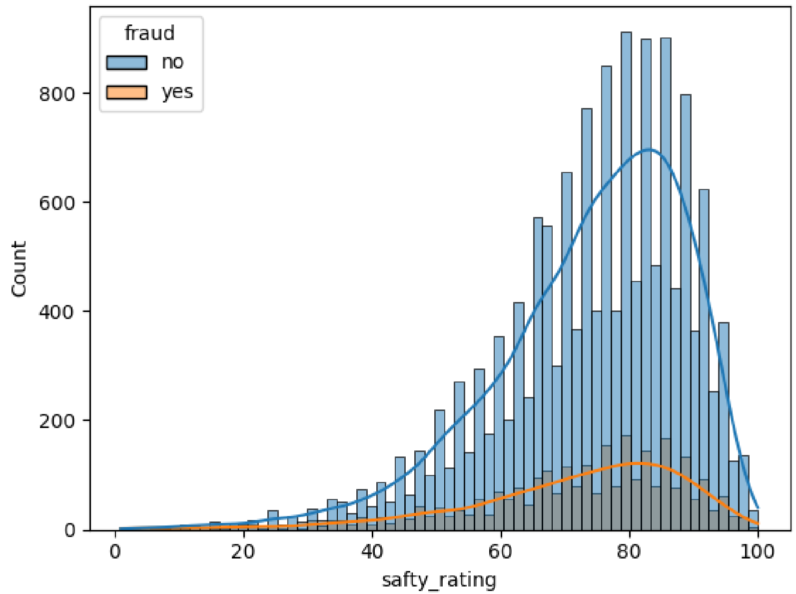


Count
age_of_vehicle
0
1000
2000
3000
4000
5000
6000
0
2
4
6
8
10
12
14
16



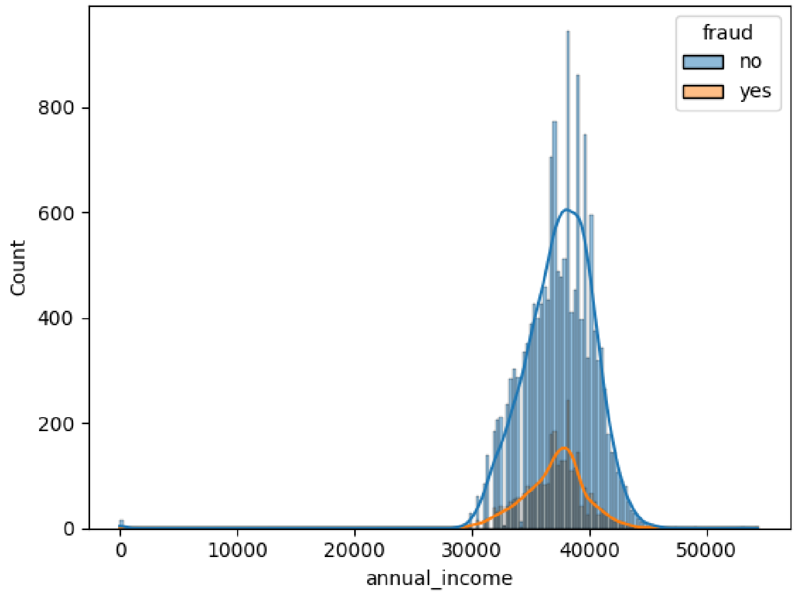


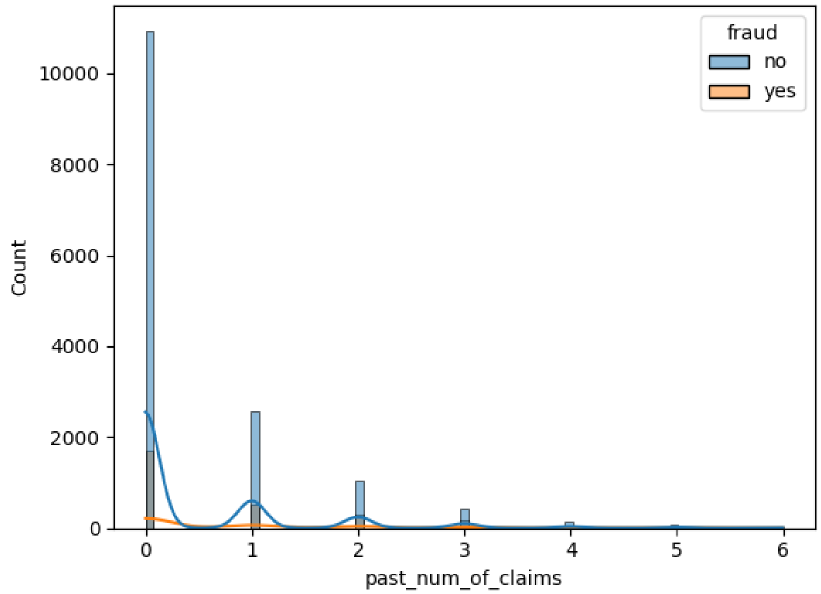


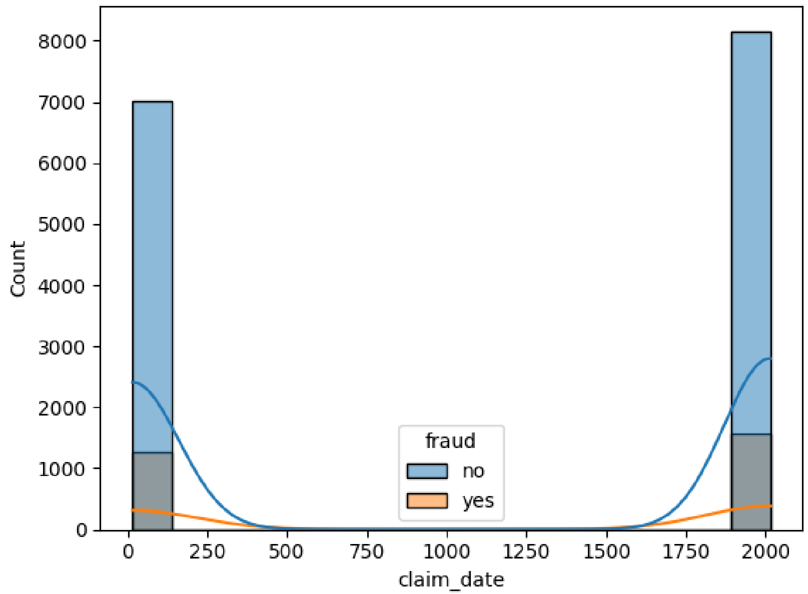


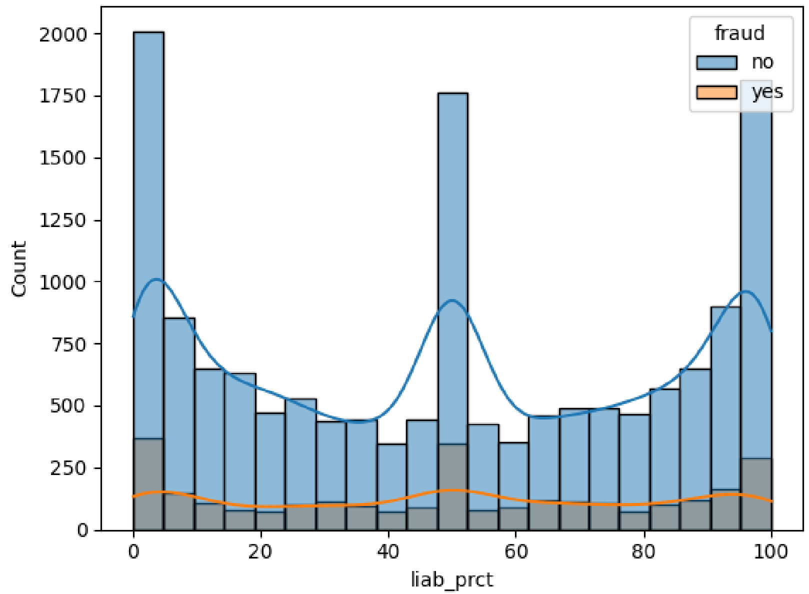
In [20]:

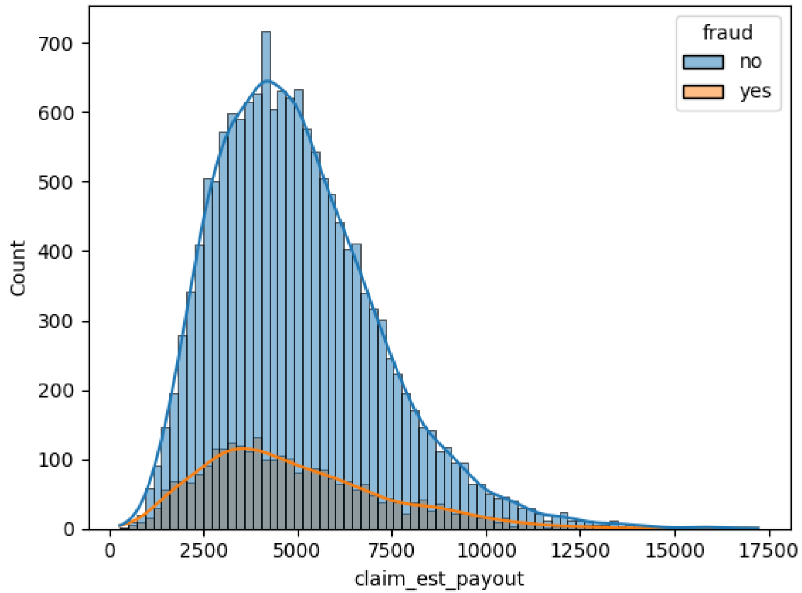
```
#Create a histogram for each integer variable with hue ="Attrition"
for i in numerical:
  sns.histplot(x=i,data=df,hue="fraud",kde=True)
  plt.show()
```

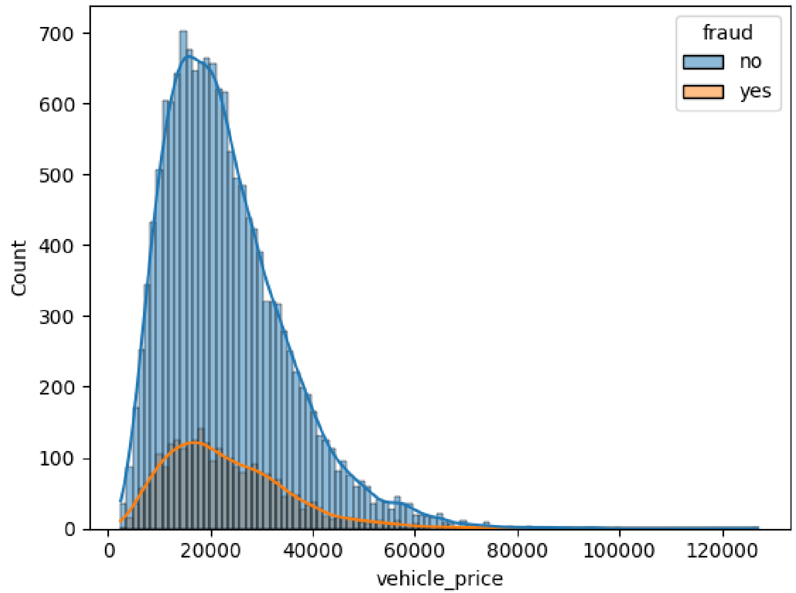

fraud
no
yes
Count
annual_income

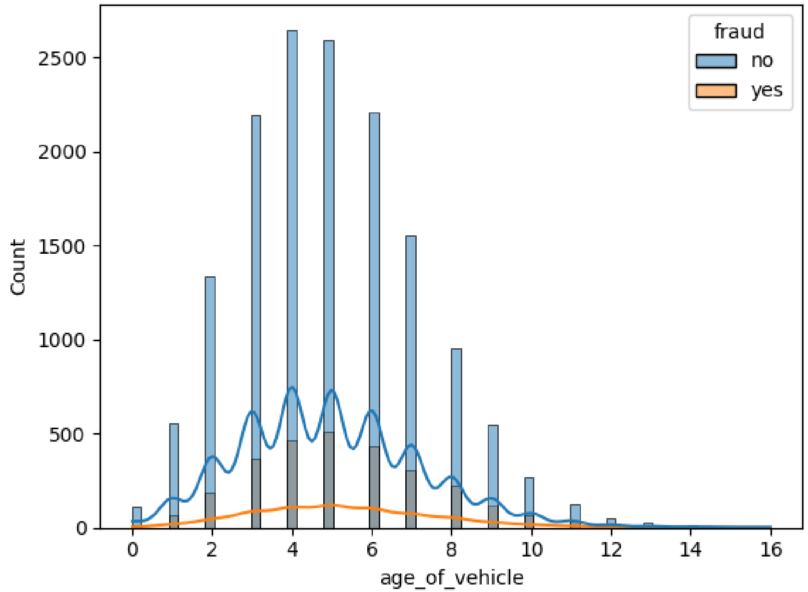

fraud
no
yes
Count
past_num_of_claims

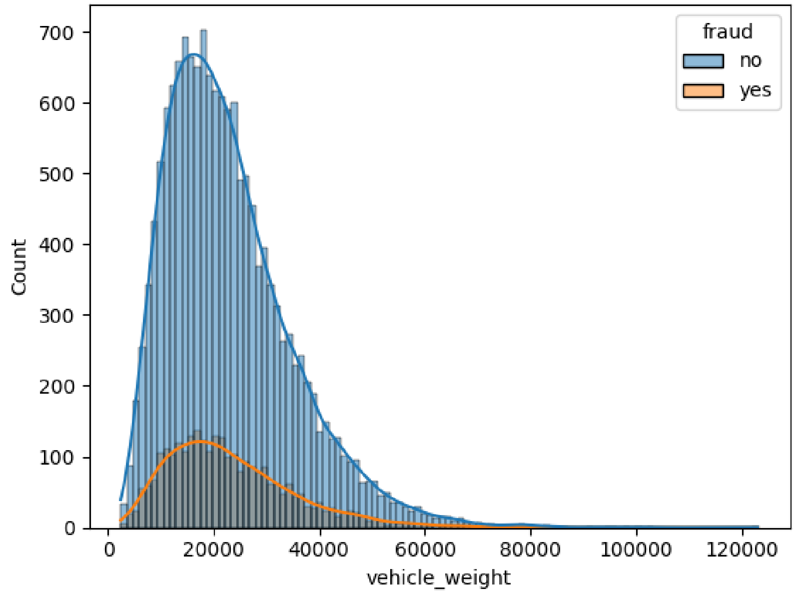

fraud
no
yes
Count
claim_date


fraud
no
yes
Count
liab_prct





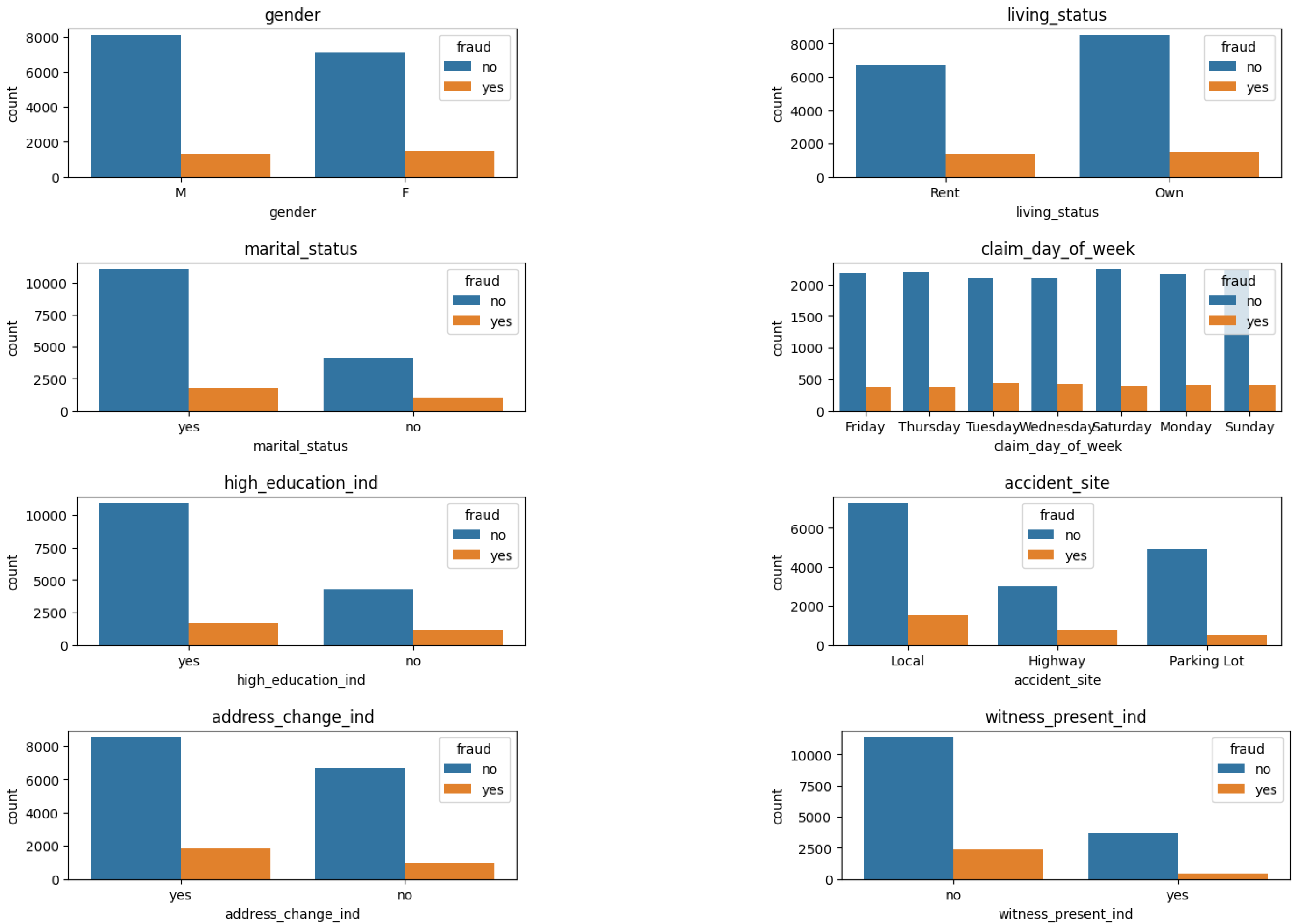


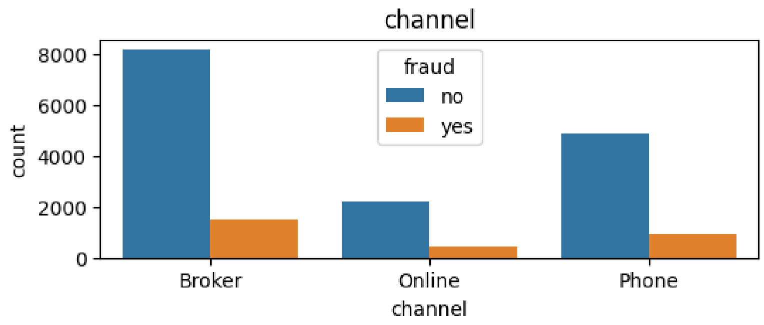


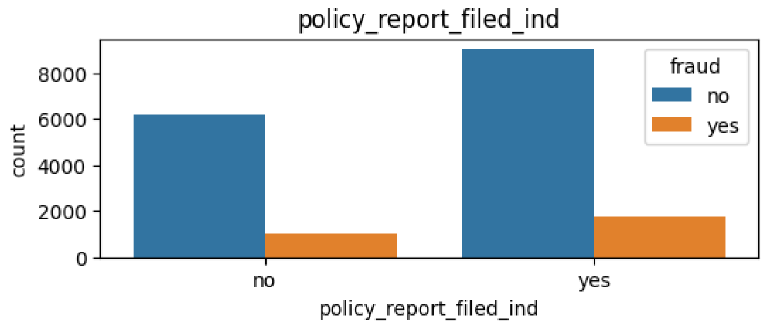
In [20]:

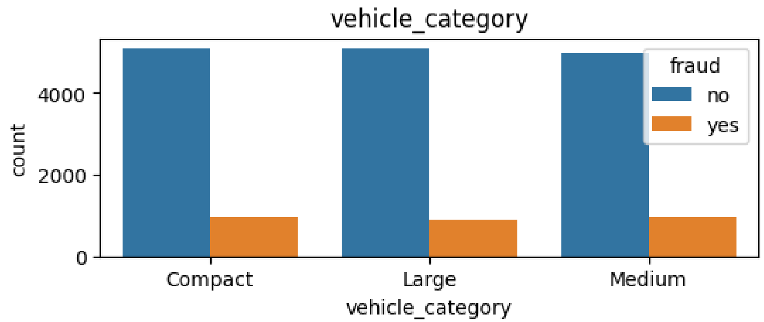
In [21]:

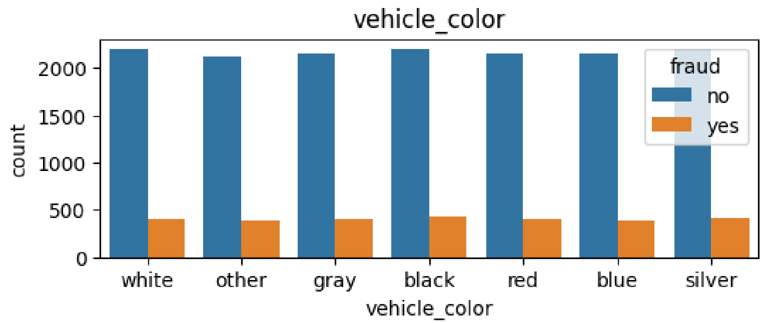
```
#Creating a countplot for each categorical variable
for i in categorical:
  plt.figure(figsize=(6,2))
  sns.countplot(x=i,data=df,hue="fraud")
  plt.title(i)
  plt.show()
```


gender
fraud
no
yes
count
8000
6000
4000
2000
0
M
F
gender
living_status
fraud
no
yes
count
8000
6000
4000
2000
0
Rent
Own
living_status
marital_status
fraud
no
yes
count
10000
7500
5000
2500
0
yes
no
marital_status
claim_day_of_week
fraud
no
yes
count
2000
1500
1000
500
0
Friday
Thursday
Tuesday
Wednesday
Saturday
Monday
Sunday
claim_day_of_week
high_education_ind
fraud
no
yes
count
10000
7500
5000
2500
0
yes
no
high_education_ind
accident_site
fraud
no
yes
count
6000
4000
2000
0
Local
Highway
Parking Lot
accident_site
address_change_ind
fraud
no
yes
count
8000
6000
4000
2000
0
yes
no
address_change_ind
witness_present_ind
fraud
no
yes
count
10000
7500
5000
2500
0
no
yes
witness_present_ind



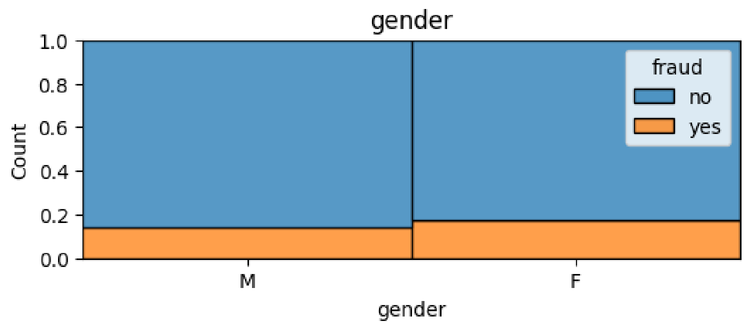


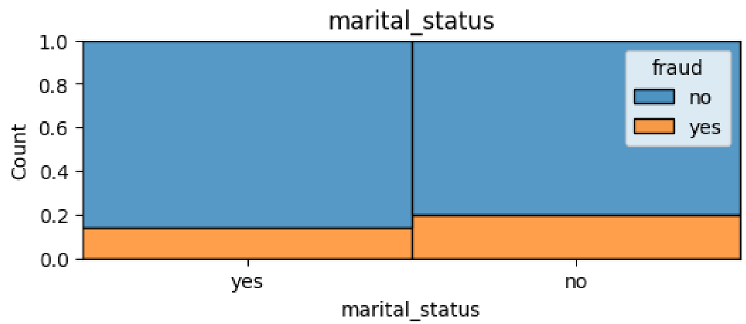


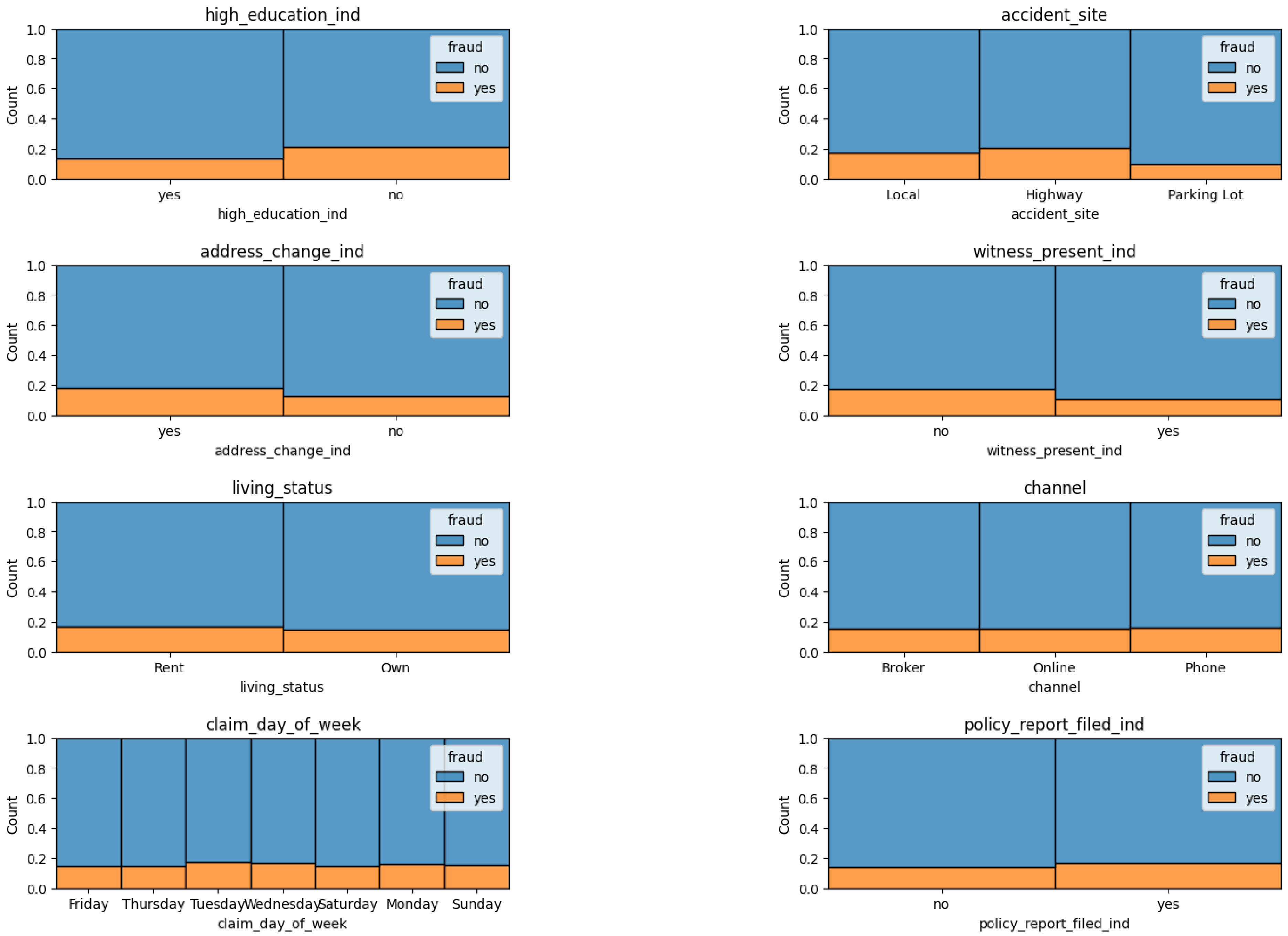


In [22]:

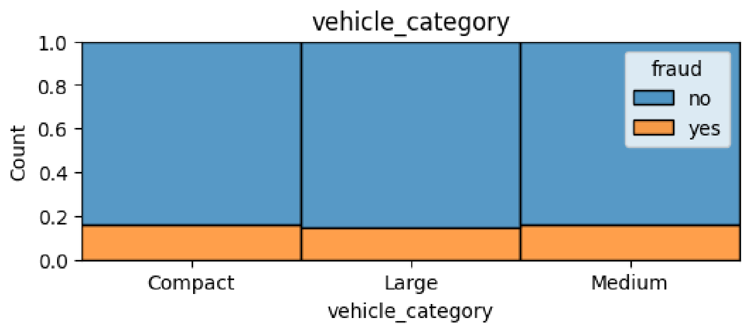
```
#Creating a Density Plot for categorical values
for i in categorical:
  plt.figure(figsize=(6,2))
  sns.histplot(x=i,data=df,hue="fraud",kde=False,multiple="fill")
  plt.title(i)
  plt.show()
```

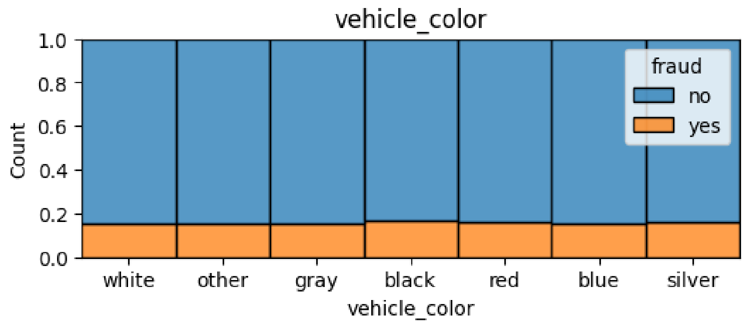






high_education_ind
Count
1.0
0.8
0.6
0.4
0.2
0.0
fraud
no
yes
yes
no
high_education_ind
accident_site
Local
Highway
Parking Lot
accident_site
address_change_ind
yes
no
address_change_ind
witness_present_ind
no
yes
witness_present_ind
living_status
Rent
Own
living_status
channel
Broker
Online
Phone
channel
claim_day_of_week
Friday
Thursday
Tuesday
Wednesday
Saturday
Monday
Sunday
claim_day_of_week
policy_report_filed_ind
no
yes
policy_report_filed_ind





In [22]:

Data PreProcessing Part 2

In [23]:
```
#Check the amount of missing values
check_missing=df.isnull().sum()*100/df.shape[0]
check_missing[check_missing>0].sort_values(ascending=False)
```

Out[23]:
```
witness_present_ind     0.733415
claim_est_payout        0.094455
age_of_vehicle          0.044449
marital_status          0.027781
dtype: float64
```

In [24]:
```
df.shape[0]
```

Out[24]:
```
17998
```

In [25]:
```
#Drop all the null value because the amount of null value is very small
df.dropna(inplace=True)
df.shape
```

Out[25]:
```
(17836, 23)
```

In [25]:

Label Encoding for object data types

In [26]:
```
for i in categorical:
  print(i,df[i].unique())#Print the column name and the unique values.....
```

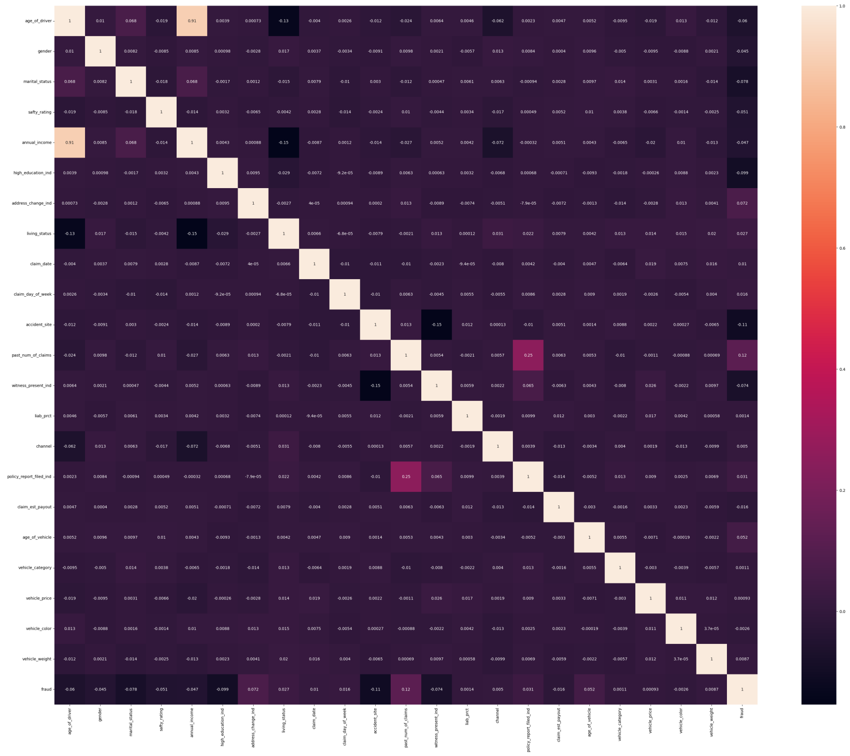
```
gender ['M' 'F']
marital_status ['yes' 'no']
high_education_ind ['yes' 'no']
address_change_ind ['yes' 'no']
living_status ['Rent' 'Own']
claim_day_of_week ['Friday' 'Thursday' 'Tuesday' 'Wednesday' 'Saturday' 'Monday'
 'Sunday']
accident_site ['Local' 'Highway' 'Parking Lot']
witness_present_ind ['no' 'yes']
channel ['Broker' 'Online' 'Phone']
policy_report_filed_ind ['no' 'yes']
vehicle_category ['Compact' 'Large' 'Medium']
vehicle_color ['white' 'other' 'gray' 'black' 'red' 'blue' 'silver']
fraud ['no' 'yes']
```

In [27]:
```
from sklearn import preprocessing

#Loop over each column in the DataFrame where dtpe="object"
for col in categorical:
#Initialize a labelEncoder object
     label_encoder=preprocessing.LabelEncoder()

#Fit the encoder to the unique values
     label_encoder.fit(df[col].unique())

#Transform the column using the encoder
     df[col]=label_encoder.transform(df[col])

     #Print the encoded values
     print(col,df[col].unique())
```

```
gender [1 0]
marital_status [1 0]
high_education_ind [1 0]
address_change_ind [1 0]
living_status [1 0]
claim_day_of_week [0 4 5 6 2 1 3]
accident_site [1 0 2]
witness_present_ind [0 1]
channel [0 1 2]
policy_report_filed_ind [0 1]
vehicle_category [0 1 2]
vehicle_color [6 3 2 0 4 1 5]
fraud [0 1]
```

In [28]:
```python
#Correlation Heatmap
plt.figure(figsize=(40,32))
sns.heatmap(df.corr(),annot=True)
```

Out[28]: <Axes: >



In [28]:

Train Test Split

In [29]:
```python
x=df.drop("fraud",axis=1)
x
```

Out[29]:

	age_of_driver	gender	marital_status	safty_rating	annual_income	high_education_ind
0	46	1	1	85	38301	1
1	21	0	0	75	30445	0
2	49	0	0	87	38923	0
3	58	0	1	58	40605	1
4	38	1	1	95	36380	1
...	...	...	...	...	...	...
17993	69	1	1	93	42338	1
17994	35	0	0	22	35579	1
17995	27	0	1	81	32953	0
17996	52	0	1	86	39519	1
17997	61	0	0	60	41126	1

17836 rows × 22 columns

In [30]:
```python
y=df["fraud"]
y  #target column
```

Out[30]:
```
0        0
1        0
2        1
3        1
4        0
        ..
17993    0
17994    1
17995    0
17996    0
17997    0
Name: fraud, Length: 17836, dtype: int64
```

In [31]:
```python
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [31]:

Remove Outliers from Train Data using z-score

In [32]:
```python
from scipy import stats
#Define the column for which you want to remove outliers
selected_col=["age_of_driver","safty_rating","annual_income","claim_est_payout","ag
#Calculate the z-score for the selectedcolumns in training data
z_score=np.abs(stats.zscore(x_train[selected_col]))
z_score
```

Out[32]:

	age_of_driver	safty_rating	annual_income	claim_est_payout	age_of_vehicle	vehicle_price
13040	1.656155	1.081805	1.909362	0.784467	1.780562	0.302276
7575	0.358461	0.682352	0.448576	2.847324	1.331006	1.657016
15125	0.946057	1.074387	0.918531	0.826649	0.886496	1.667041
8942	0.610288	0.420995	0.660770	0.608390	0.886496	0.008561
288	1.152501	0.943708	1.140499	0.920544	0.447033	0.181042
...	...	...	...	...	...	...
9301	0.313078	0.493753	0.164882	1.055723	0.886496	0.327084
13235	0.480962	0.747691	0.341711	0.343466	0.447033	1.033901
9923	0.732789	1.074387	0.619195	0.541101	0.002523	0.770138
10887	0.229135	1.139726	0.081229	0.468125	1.780562	0.923592
2759	0.061251	1.335743	0.078256	0.041269	1.780562	2.028992

14268 rows × 7 columns

In [33]:

```
#Set a threshold value for outlier detection
threshold=3
#Find the indices of the outliers based on threshold
outlier_indices=np.where(z_score>threshold)[0]
outlier_indices
```

Out[33]:

```
array([  29,   32,   45,   45,   67,   67,   79,   85,   92,
         93,  100,  148,  173,  188,  229,  229,  242,  263,
        271,  277,  278,  304,  342,  366,  368,  379,  405,
        423,  424,  431,  439,  472,  499,  509,  515,  536,
        541,  542,  544,  580,  587,  590,  592,  611,  617,
        642,  659,  666,  672,  706,  706,  736,  749,  761,
        767,  773,  774,  796,  814,  821,  832,  832,  836,
        845,  847,  854,  855,  855,  885,  888,  904,  936,
        941,  951,  954,  968,  977,  995, 1014, 1038, 1044,
       1058, 1076, 1084, 1096, 1097, 1117, 1138, 1158, 1170,
       1187, 1221, 1223, 1274, 1290, 1300, 1318, 1374, 1389,
       1391, 1403, 1411, 1431, 1445, 1470, 1484, 1490, 1496,
       1496, 1520, 1524, 1551, 1557, 1564, 1625, 1632, 1652,
       1660, 1684, 1689, 1712, 1720, 1733, 1737, 1812, 1838,
       1869, 1916, 1919, 1956, 1958, 1977, 2034, 2048, 2065,
       2075, 2079, 2087, 2092, 2106, 2107, 2141, 2141, 2152,
       2168, 2232, 2261, 2309, 2347, 2358, 2379, 2401, 2412,
       2423, 2427, 2440, 2448, 2449, 2454, 2471, 2474, 2493,
       2496, 2525, 2575, 2578, 2585, 2596, 2612, 2617, 2628,
       2631, 2641, 2656, 2687, 2692, 2694, 2704, 2708, 2715,
       2718, 2720, 2723, 2751, 2755, 2758, 2791, 2811, 2821,
       2827, 2828, 2850, 2862, 2900, 2911, 2968, 2990, 3002,
       3028, 3032, 3045, 3056, 3056, 3085, 3121, 3147, 3162,
       3185, 3198, 3219, 3220, 3236, 3240, 3250, 3285, 3295,
       3298, 3323, 3323, 3335, 3341, 3348, 3389, 3432, 3461,
       3467, 3502, 3508, 3540, 3558, 3571, 3578, 3598, 3615,
       3621, 3653, 3662, 3680, 3684, 3723, 3726, 3778, 3789,
       3801, 3810, 3831, 3846, 3858, 3867, 3875, 3892, 3894,
       3926, 3930, 3942, 3979, 3982, 3995, 4022, 4041, 4069,
       4077, 4080, 4084, 4151, 4159, 4209, 4219, 4220, 4233,
       4246, 4247, 4257, 4257, 4265, 4288, 4343, 4353, 4365,
       4388, 4390, 4404, 4405, 4417, 4430, 4441, 4443, 4445,
       4451, 4512, 4534, 4548, 4550, 4573, 4606, 4616, 4623,
       4632, 4634, 4638, 4670, 4672, 4682, 4692, 4707, 4713,
       4728, 4729, 4762, 4784, 4792, 4801, 4802, 4811, 4815,
       4853, 4854, 4924, 4941, 4942, 4971, 5000, 5006, 5026,
       5031, 5031, 5037, 5082, 5109, 5114, 5125, 5175, 5195,
       5196, 5229, 5231, 5261, 5308, 5324, 5349, 5363, 5382,
       5406, 5409, 5412, 5433, 5439, 5441, 5446, 5449, 5451,
       5501, 5501, 5504, 5517, 5523, 5548, 5599, 5601, 5604,
       5625, 5627, 5655, 5656, 5667, 5671, 5683, 5714, 5741,
       5768, 5791, 5806, 5807, 5826, 5855, 5866, 5881, 5904,
       5961, 5971, 5980, 6028, 6031, 6052, 6057, 6077, 6102,
       6105, 6105, 6121, 6124, 6128, 6145, 6161, 6182, 6194,
       6202, 6207, 6210, 6217, 6233, 6236, 6267, 6283, 6286,
       6289, 6293, 6324, 6375, 6393, 6407, 6439, 6473, 6483,
       6533, 6536, 6536, 6545, 6560, 6586, 6611, 6690, 6700,
       6707, 6712, 6720, 6743, 6746, 6760, 6795, 6827, 6832,
       6847, 6867, 6871, 6872, 6957, 6984, 6996, 7012, 7028,
       7033, 7033, 7073, 7084, 7090, 7092, 7094, 7111, 7124,
       7150, 7155, 7170, 7173, 7187, 7193, 7231, 7249, 7272,
       7284, 7348, 7370, 7392, 7431, 7433, 7445, 7453, 7454,
       7468, 7474, 7493, 7508, 7512, 7574, 7590, 7611, 7618,
       7655, 7663, 7671, 7679, 7706, 7721, 7744, 7745, 7767,
       7770, 7782, 7857, 7860, 7866, 7892, 7899, 7925, 7939,
       7952, 7957, 8011, 8059, 8060, 8071, 8083, 8115, 8232,
       8257, 8289, 8291, 8308, 8315, 8330, 8365, 8391, 8470,
       8510, 8521, 8569, 8613, 8627, 8628, 8640, 8667, 8735,
       8807, 8807, 8824, 8831, 8834, 8836, 8856, 8864, 8928,
       8939, 8953, 9008, 9017, 9023, 9034, 9069, 9070, 9072,
       9104, 9104, 9119, 9120, 9140, 9175, 9205, 9211, 9223,
       9283, 9284, 9329, 9339, 9362, 9368, 9369, 9375, 9398,
       9423, 9425, 9440, 9448, 9469, 9474, 9518, 9546, 9558,
       9585, 9585, 9633, 9663, 9670, 9675, 9681, 9708, 9736,
```

```
        9754,  9756,  9782,  9812,  9823,  9837,  9882,  9901,  9912,
        9921,  9928,  9941,  9957,  9966,  9967, 10004, 10041, 10042,
       10048, 10071, 10078, 10098, 10168, 10196, 10208, 10218, 10305,
       10306, 10325, 10346, 10418, 10439, 10469, 10482, 10534, 10547,
       10557, 10563, 10580, 10586, 10615, 10617, 10619, 10641, 10646,
       10673, 10685, 10707, 10712, 10731, 10775, 10778, 10780, 10784,
       10788, 10795, 10798, 10806, 10833, 10875, 10876, 10943, 10961,
       10980, 11000, 11001, 11017, 11022, 11036, 11043, 11047, 11053,
       11069, 11083, 11111, 11213, 11244, 11256, 11259, 11272, 11308,
       11338, 11404, 11406, 11409, 11411, 11421, 11440, 11466, 11471,
       11511, 11537, 11570, 11588, 11589, 11631, 11641, 11655, 11683,
       11707, 11714, 11720, 11770, 11775, 11776, 11857, 11893, 11895,
       11902, 11930, 11958, 11961, 11963, 11968, 11982, 12001, 12016,
       12026, 12039, 12043, 12046, 12072, 12074, 12084, 12106, 12106,
       12112, 12113, 12131, 12182, 12191, 12196, 12200, 12205, 12213,
       12214, 12217, 12226, 12239, 12284, 12307, 12310, 12330, 12330,
       12347, 12348, 12413, 12416, 12439, 12469, 12503, 12503, 12508,
       12534, 12538, 12552, 12562, 12596, 12602, 12603, 12630, 12656,
       12658, 12658, 12659, 12693, 12694, 12716, 12724, 12745, 12783,
       12806, 12833, 12847, 12863, 12866, 12971, 12975, 12976, 12991,
       13003, 13057, 13111, 13177, 13189, 13197, 13201, 13236, 13256,
       13286, 13289, 13293, 13310, 13315, 13377, 13382, 13402, 13406,
       13452, 13491, 13500, 13563, 13571, 13578, 13582, 13596, 13621,
       13676, 13705, 13735, 13745, 13805, 13825, 13850, 13858, 13864,
       13878, 13887, 14029, 14041, 14056, 14165, 14183, 14191, 14192,
       14239, 14261])
```

In [34]:
```
#Remove that outliers from the training data
x_train=x_train.drop(x_train.index[outlier_indices])
y_train=y_train.drop(y_train.index[outlier_indices])
```

Decision Tree Classifier

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
dtree=DecisionTreeClassifier(class_weight="balanced")
param_grid = {
    'max_depth': [3, 4, 5, 6, 7, 8],
    'min_samples_split': [2, 3, 4],
    'min_samples_leaf': [1, 2, 3, 4],
    'random_state': [0, 42]
}
#Perform a grid search with cross validation to find the best HP
gscv=GridSearchCV(dtree,param_grid,cv=5,verbose=3)
gscv.fit(x_train,y_train)
```

In [36]:
```
#Print the best hyper parameters
print(gscv.best_params_)
```

```
{'max_depth': 8, 'min_samples_leaf': 1, 'min_samples_split': 2, 'random_state': 0}
```

In [37]:
```
dtree=DecisionTreeClassifier(max_depth=8,min_samples_leaf=1,min_samples_split=2,ra
dtree.fit(x_train,y_train)
```

Out[37]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier(max_depth=8, random_state=0)
```

In [38]:
```
from sklearn.metrics import accuracy_score
y_pred=dtree.predict(x_test)
accu_score=round(accuracy_score(y_test,y_pred),2)
print("AccuracyScore is",accu_score*100,"%")
```

```
AccuracyScore is 83.0 %
```

In [39]:
```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,lo
fscore=f1_score(y_test,y_pred,average="micro")
pscore=precision_score(y_test,y_pred,average="micro")
rscore=recall_score(y_test,y_pred,average="micro")
jscore=jaccard_score(y_test,y_pred,average="micro")
Lloss=log_loss(y_test,y_pred)
print("F1 score ",fscore)
print("Precision score ",pscore)
print("Recall score ",rscore)
print("Jaccard score ",jscore)
print("Log Loss",Lloss)
```

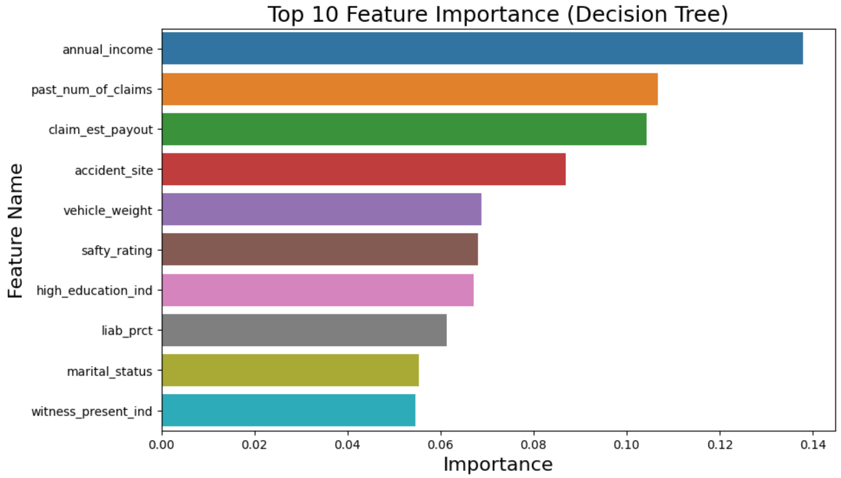
```
F1 score  0.83071748878892377
Precision score  0.83071748878892377
Recall score  0.83071748878892377
Jaccard score  0.710450623202301
Log Loss 6.101560158920057
```

In [40]:
```
imp_df=pd.DataFrame({
    "Feature Name":x_train.columns,
    "Importance":dtree.feature_importances_
})

fi=imp_df.sort_values(by="Importance",ascending=False)
fi2=fi.head(10)
fi2
```

Out[40]:

	Feature Name	Importance
4	annual_income	0.137908
11	past_num_of_claims	0.106637
16	claim_est_payout	0.104215
10	accident_site	0.086982
21	vehicle_weight	0.068807
3	safty_rating	0.068106
5	high_education_ind	0.067187
13	liab_prct	0.061350
2	marital_status	0.055311
12	witness_present_ind	0.054588

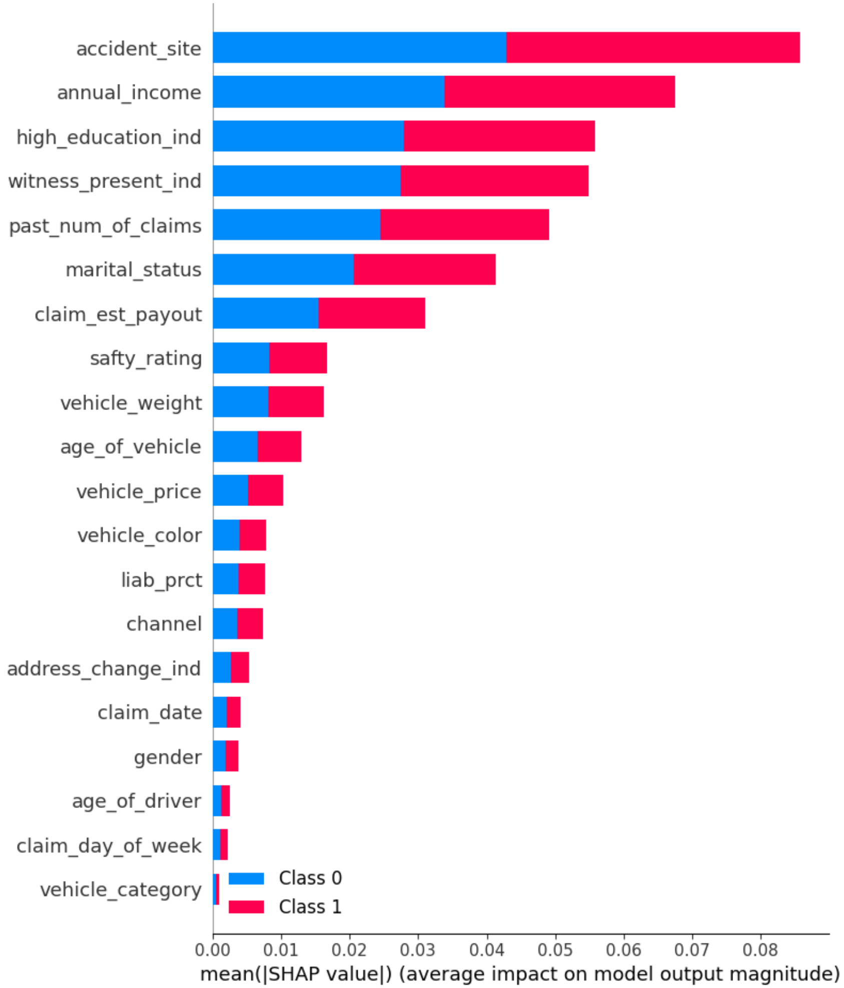
In [41]:
```
plt.figure(figsize=(10,6))
sns.barplot(data=fi2,x="Importance",y="Feature Name")
plt.title("Top 10 Feature Importance (Decision Tree)",fontsize=18)
plt.xlabel("Importance",fontsize=16)
plt.ylabel("Feature Name",fontsize=16)
plt.show()
```



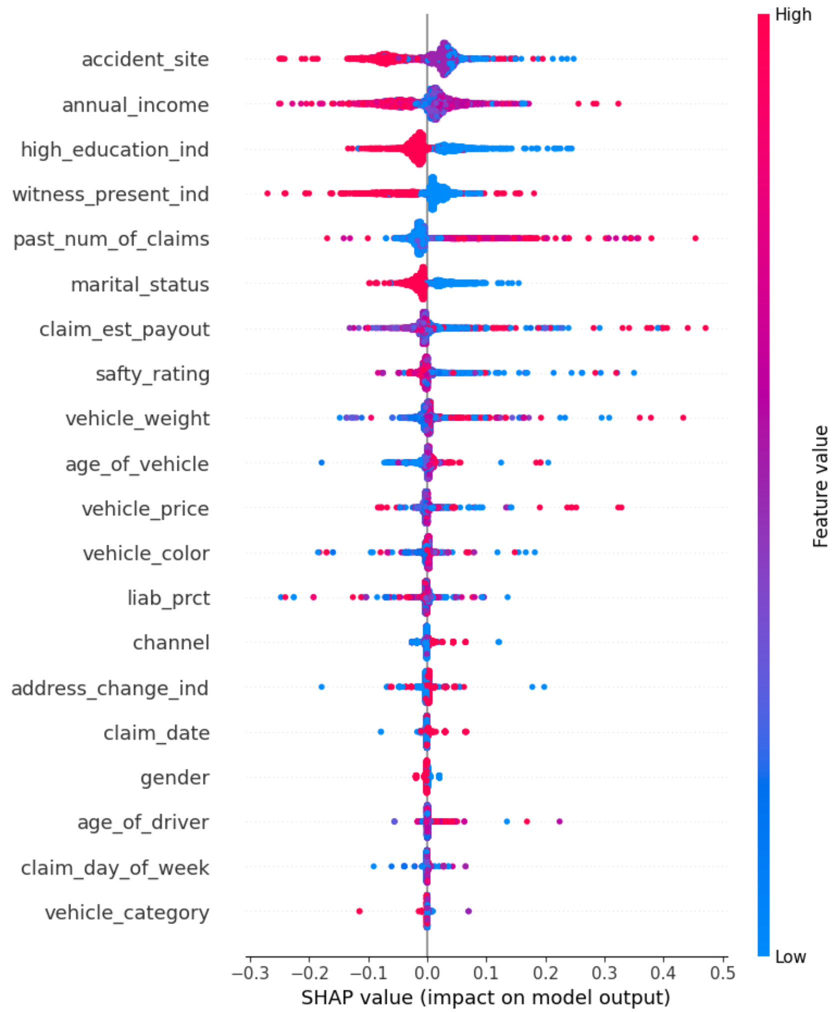
In [42]:
```
!pip install shap
```

```
Collecting shap
  Downloading shap-0.43.0-cp310-cp310-manylinux_2_12_x86_64.manylinux2010_x86_64.m
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (532 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 532.9/532.9 kB 7.7 MB/s eta 0:00:00
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (f
rom shap) (1.23.5)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (f
rom shap) (1.11.3)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-pack
ages (from shap) (1.2.2)
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages
(from shap) (1.5.3)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.10/dist-pack
ages (from shap) (4.66.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.10/dist-pa
ckages (from shap) (23.2)
Collecting slicer==0.0.7 (from shap)
  Downloading slicer-0.0.7-py3-none-any.whl (14 kB)
Requirement already satisfied: numba in /usr/local/lib/python3.10/dist-packages (f
rom shap) (0.56.4)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.10/dist-packa
ges (from shap) (2.2.1)
Requirement already satisfied: llvmlite<0.40,>=0.39.0dev0 in /usr/local/lib/python
3.10/dist-packages (from numba->shap) (0.39.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packag
es (from numba->shap) (67.7.2)
Requirement already satisfied: python-dateutil>=2.8.1 in /usr/local/lib/python3.1
0/dist-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-pack
ages (from pandas->shap) (2023.3.post1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-pac
kages (from scikit-learn->shap) (1.3.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/d
ist-packages (from scikit-learn->shap) (3.2.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages
(from python-dateutil>=2.8.1->pandas->shap) (1.16.0)
Installing collected packages: slicer, shap
Successfully installed shap-0.43.0 slicer-0.0.7
```

In [43]:
```
import shap
explainer=shap.TreeExplainer(dtree)
shap_values=explainer.shap_values(x_test)

shap.summary_plot(shap_values,x_test)
```



In [44]:
```
# compute SHAP values

explainer=shap.TreeExplainer(dtree)
shap_values=explainer.shap_values(x_test)

shap.summary_plot(shap_values[1],x_test.values,feature_names=x_test.columns)
```



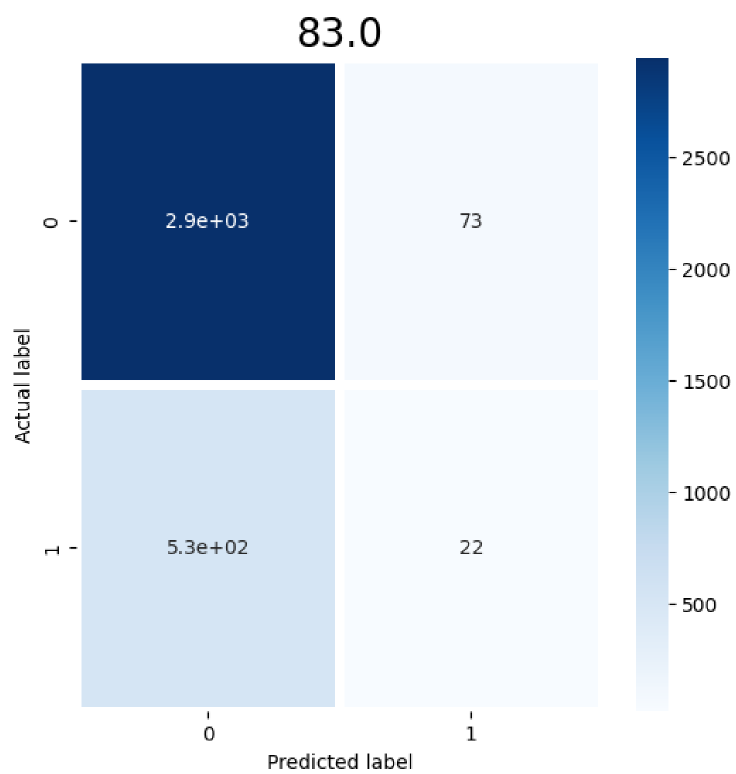
In [45]:
```
shap_values[1]
```

Out[45]:
```
array([[ 4.41342291e-04,  1.24651612e-03, -1.16415354e-02, ...,
        -2.28092024e-03,  1.29648309e-03,  6.55574598e-04],
       [ 3.52561710e-04,  2.71089462e-03, -1.75680169e-02, ...,
         1.63710853e-03,  1.46067949e-06,  1.62835912e-03],
       [-4.55643156e-04,  1.42176066e-03,  5.18903137e-02, ...,
         1.75432185e-03,  2.49871735e-03,  3.18569336e-03],
       ...,
       [-5.45483557e-05, -2.88769132e-03, -7.53621657e-03, ...,
         1.71328229e-02,  8.56224620e-03,  1.98232446e-02],
       [ 2.79479436e-04,  9.49054766e-04, -4.37247910e-02, ...,
         1.55745671e-03,  1.97148227e-04,  3.81103579e-03],
       [ 1.89367439e-04,  4.40777108e-04, -1.65145424e-02, ...,
        -2.96926009e-04, -1.82232540e-04,  7.74490886e-04]])
```

In [46]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred)
```

```
plt.figure(figsize=(6,6))
sns.heatmap(data=cm,linewidths=5,annot=True,cmap="Blues")
plt.ylabel("Actual label")
plt.xlabel("Predicted label")
plt.title(accu_score*100,size=20)
```

Out[46]:
```
Text(0.5, 1.0, '83.0')
```



In [46]:

Random Forest Classifier

In [47]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
rf=RandomForestClassifier(class_weight="balanced")

param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 5, 10],
    'max_features': ['sqrt', 'log2', None],
    'random_state': [0, 42]
}

#Perform a grid search with cross-validation to find the best hyper-param
gscv=GridSearchCV(rf,param_grid,cv=5,verbose=3)
gscv.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV 1/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=0;,
score=0.844 total time=   2.2s
[CV 2/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=0;,
score=0.844 total time=   2.4s
[CV 3/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=0;,
score=0.844 total time=   4.1s
[CV 4/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=0;,
score=0.844 total time=   5.5s
[CV 5/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=0;,
score=0.844 total time=   3.8s
[CV 1/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=4
2;, score=0.844 total time=   4.2s
[CV 2/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=4
2;, score=0.844 total time=   5.7s
[CV 3/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=4
2;, score=0.844 total time=   4.4s
[CV 4/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=4
2;, score=0.844 total time=   4.5s
[CV 5/5] END max_depth=None, max_features=sqrt, n_estimators=100, random_state=4
2;, score=0.844 total time=   5.0s
[CV 1/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=0;,
score=0.844 total time=   4.3s
[CV 2/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=0;,
score=0.844 total time=   4.6s
[CV 3/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=0;,
score=0.844 total time=   4.6s
[CV 4/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=0;,
score=0.844 total time=   4.2s
[CV 5/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=0;,
score=0.844 total time=   4.9s
[CV 1/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=4
2;, score=0.844 total time=   4.4s
[CV 2/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=4
2;, score=0.844 total time=   4.2s
[CV 3/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=4
2;, score=0.844 total time=   5.0s
[CV 4/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=4
2;, score=0.844 total time=   4.2s
[CV 5/5] END max_depth=None, max_features=sqrt, n_estimators=200, random_state=4
2;, score=0.845 total time=   4.2s
[CV 1/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=0;,
score=0.844 total time=   2.4s
[CV 2/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=0;,
score=0.844 total time=   2.7s
[CV 3/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=0;,
score=0.844 total time=   2.1s
[CV 4/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=0;,
score=0.844 total time=   2.1s
[CV 5/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=0;,
score=0.844 total time=   2.1s
[CV 1/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=4
2;, score=0.844 total time=   2.1s
[CV 2/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=4
2;, score=0.844 total time=   2.7s
[CV 3/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=4
2;, score=0.844 total time=   2.4s
[CV 4/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=4
2;, score=0.844 total time=   2.1s
[CV 5/5] END max_depth=None, max_features=log2, n_estimators=100, random_state=4
2;, score=0.844 total time=   2.1s
[CV 1/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=0;,
score=0.844 total time=   4.2s
[CV 2/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=0;,
score=0.844 total time=   5.1s
[CV 3/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=0;,
score=0.844 total time=   4.2s
[CV 4/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=0;,
score=0.844 total time=   4.3s
[CV 5/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=0;,
score=0.844 total time=   5.7s
[CV 1/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=4
2;, score=0.844 total time=   4.2s
[CV 2/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=4
2;, score=0.844 total time=   6.9s
[CV 3/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=4
2;, score=0.844 total time=   4.2s
[CV 4/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=4
2;, score=0.844 total time=   4.2s
[CV 5/5] END max_depth=None, max_features=log2, n_estimators=200, random_state=4
2;, score=0.845 total time=   5.5s
[CV 1/5] END max_depth=None, max_features=None, n_estimators=100, random_state=0;,
score=0.843 total time=   9.2s
[CV 2/5] END max_depth=None, max_features=None, n_estimators=100, random_state=0;,
score=0.841 total time=  17.4s
[CV 3/5] END max_depth=None, max_features=None, n_estimators=100, random_state=0;,
score=0.840 total time=  14.6s
[CV 4/5] END max_depth=None, max_features=None, n_estimators=100, random_state=0;,
score=0.841 total time=   9.2s
[CV 5/5] END max_depth=None, max_features=None, n_estimators=100, random_state=0;,
score=0.843 total time=   8.3s
[CV 1/5] END max_depth=None, max_features=None, n_estimators=100, random_state=4
2;, score=0.844 total time=   9.3s
[CV 2/5] END max_depth=None, max_features=None, n_estimators=100, random_state=4
2;, score=0.842 total time=   9.2s
[CV 3/5] END max_depth=None, max_features=None, n_estimators=100, random_state=4
2;, score=0.841 total time=   8.3s
[CV 4/5] END max_depth=None, max_features=None, n_estimators=100, random_state=4
2;, score=0.842 total time=   9.1s
[CV 5/5] END max_depth=None, max_features=None, n_estimators=100, random_state=4
2;, score=0.842 total time=   9.2s
[CV 1/5] END max_depth=None, max_features=None, n_estimators=200, random_state=0;,
score=0.843 total time=  17.5s
[CV 2/5] END max_depth=None, max_features=None, n_estimators=200, random_state=0;,
score=0.840 total time=  18.8s
[CV 3/5] END max_depth=None, max_features=None, n_estimators=200, random_state=0;,
score=0.841 total time=  18.6s
[CV 4/5] END max_depth=None, max_features=None, n_estimators=200, random_state=0;,
score=0.842 total time=  17.6s
[CV 5/5] END max_depth=None, max_features=None, n_estimators=200, random_state=0;,
score=0.841 total time=  18.1s
[CV 1/5] END max_depth=None, max_features=None, n_estimators=200, random_state=4
2;, score=0.844 total time=  17.6s
[CV 2/5] END max_depth=None, max_features=None, n_estimators=200, random_state=4
2;, score=0.842 total time=  18.2s
[CV 3/5] END max_depth=None, max_features=None, n_estimators=200, random_state=4
2;, score=0.842 total time=  17.4s
[CV 4/5] END max_depth=None, max_features=None, n_estimators=200, random_state=4
2;, score=0.844 total time=  17.2s
[CV 5/5] END max_depth=None, max_features=None, n_estimators=200, random_state=4
2;, score=0.842 total time=  18.2s
[CV 1/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=0;, sc
ore=0.637 total time=   1.0s
[CV 2/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=0;, sc
ore=0.627 total time=   1.0s
[CV 3/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=0;, sc
ore=0.646 total time=   1.0s
[CV 4/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=0;, sc
```

```
ore=0.631 total time=   0.9s
[CV 5/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=0;, sc
ore=0.649 total time=   1.0s
[CV 1/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=42;, s
core=0.638 total time=   1.0s
[CV 2/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=42;, s
core=0.623 total time=   1.0s
[CV 3/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=42;, s
core=0.654 total time=   1.0s
[CV 4/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=42;, s
core=0.621 total time=   1.0s
[CV 5/5] END max_depth=5, max_features=sqrt, n_estimators=100, random_state=42;, s
core=0.653 total time=   1.4s
[CV 1/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=0;, sc
ore=0.635 total time=   2.3s
[CV 2/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=0;, sc
ore=0.627 total time=   1.9s
[CV 3/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=0;, sc
ore=0.647 total time=   2.0s
[CV 4/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=0;, sc
ore=0.629 total time=   1.9s
[CV 5/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=0;, sc
ore=0.656 total time=   1.9s
[CV 1/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=42;, s
core=0.637 total time=   2.2s
[CV 2/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=42;, s
core=0.633 total time=   2.5s
[CV 3/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=42;, s
core=0.652 total time=   2.0s
[CV 4/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=42;, s
core=0.628 total time=   1.9s
[CV 5/5] END max_depth=5, max_features=sqrt, n_estimators=200, random_state=42;, s
core=0.652 total time=   1.9s
[CV 1/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=0;, sc
ore=0.637 total time=   1.0s
[CV 2/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=0;, sc
ore=0.627 total time=   1.0s
[CV 3/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=0;, sc
ore=0.646 total time=   1.0s
[CV 4/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=0;, sc
ore=0.631 total time=   1.1s
[CV 5/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=0;, sc
ore=0.649 total time=   1.4s
[CV 1/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=42;, s
core=0.638 total time=   1.3s
[CV 2/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=42;, s
core=0.623 total time=   1.0s
[CV 3/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=42;, s
core=0.654 total time=   1.0s
[CV 4/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=42;, s
core=0.621 total time=   1.0s
[CV 5/5] END max_depth=5, max_features=log2, n_estimators=100, random_state=42;, s
core=0.653 total time=   0.9s
[CV 1/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=0;, sc
ore=0.635 total time=   1.9s
[CV 2/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=0;, sc
ore=0.627 total time=   1.9s
[CV 3/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=0;, sc
ore=0.647 total time=   1.9s
[CV 4/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=0;, sc
ore=0.629 total time=   2.5s
[CV 5/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=0;, sc
ore=0.656 total time=   2.1s
[CV 1/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=42;, s
core=0.637 total time=   1.8s
[CV 2/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=42;, s
core=0.633 total time=   1.8s
[CV 3/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=42;, s
core=0.652 total time=   1.9s
[CV 4/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=42;, s
core=0.628 total time=   1.9s
[CV 5/5] END max_depth=5, max_features=log2, n_estimators=200, random_state=42;, s
core=0.652 total time=   2.3s
[CV 1/5] END max_depth=5, max_features=None, n_estimators=100, random_state=0;, sc
ore=0.628 total time=   4.0s
[CV 2/5] END max_depth=5, max_features=None, n_estimators=100, random_state=0;, sc
ore=0.599 total time=   3.5s
[CV 3/5] END max_depth=5, max_features=None, n_estimators=100, random_state=0;, sc
ore=0.616 total time=   3.5s
[CV 4/5] END max_depth=5, max_features=None, n_estimators=100, random_state=0;, sc
ore=0.620 total time=   4.4s
[CV 5/5] END max_depth=5, max_features=None, n_estimators=100, random_state=0;, sc
ore=0.621 total time=   3.5s
[CV 1/5] END max_depth=5, max_features=None, n_estimators=100, random_state=42;, s
core=0.631 total time=   3.5s
[CV 2/5] END max_depth=5, max_features=None, n_estimators=100, random_state=42;, s
core=0.600 total time=   4.1s
[CV 3/5] END max_depth=5, max_features=None, n_estimators=100, random_state=42;, s
core=0.612 total time=   3.8s
[CV 4/5] END max_depth=5, max_features=None, n_estimators=100, random_state=42;, s
core=0.614 total time=   3.5s
[CV 5/5] END max_depth=5, max_features=None, n_estimators=100, random_state=42;, s
core=0.614 total time=   3.5s
[CV 1/5] END max_depth=5, max_features=None, n_estimators=200, random_state=0;, sc
ore=0.633 total time=   7.9s
[CV 2/5] END max_depth=5, max_features=None, n_estimators=200, random_state=0;, sc
ore=0.600 total time=   7.9s
[CV 3/5] END max_depth=5, max_features=None, n_estimators=200, random_state=0;, sc
ore=0.622 total time=   7.1s
[CV 4/5] END max_depth=5, max_features=None, n_estimators=200, random_state=0;, sc
ore=0.620 total time=   7.9s
[CV 5/5] END max_depth=5, max_features=None, n_estimators=200, random_state=0;, sc
ore=0.618 total time=   7.0s
[CV 1/5] END max_depth=5, max_features=None, n_estimators=200, random_state=42;, s
core=0.634 total time=   7.8s
[CV 2/5] END max_depth=5, max_features=None, n_estimators=200, random_state=42;, s
core=0.598 total time=   7.4s
[CV 3/5] END max_depth=5, max_features=None, n_estimators=200, random_state=42;, s
core=0.616 total time=   7.5s
[CV 4/5] END max_depth=5, max_features=None, n_estimators=200, random_state=42;, s
core=0.613 total time=   7.9s
[CV 5/5] END max_depth=5, max_features=None, n_estimators=200, random_state=42;, s
core=0.618 total time=   7.0s
[CV 1/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=0;, s
core=0.761 total time=   1.6s
[CV 2/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=0;, s
core=0.766 total time=   2.1s
[CV 3/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=0;, s
core=0.789 total time=   1.9s
[CV 4/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=0;, s
core=0.769 total time=   1.6s
[CV 5/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=0;, s
core=0.766 total time=   1.6s
[CV 1/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=42;,
score=0.768 total time=   1.6s
[CV 2/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=42;,
score=0.777 total time=   1.6s
[CV 3/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=42;,
```

```
score=0.785 total time=   1.6s
[CV 4/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=42;,
score=0.759 total time=   1.7s
[CV 5/5] END max_depth=10, max_features=sqrt, n_estimators=100, random_state=42;,
score=0.766 total time=   2.3s
[CV 1/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=0;, s
core=0.768 total time=   3.3s
[CV 2/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=0;, s
core=0.765 total time=   3.1s
[CV 3/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=0;, s
core=0.795 total time=   3.1s
[CV 4/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=0;, s
core=0.767 total time=   4.0s
[CV 5/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=0;, s
core=0.770 total time=   3.1s
[CV 1/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=42;,
score=0.766 total time=   3.1s
[CV 2/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=42;,
score=0.773 total time=   3.1s
[CV 3/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=42;,
score=0.791 total time=   4.0s
[CV 4/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=42;,
score=0.767 total time=   3.1s
[CV 5/5] END max_depth=10, max_features=sqrt, n_estimators=200, random_state=42;,
score=0.765 total time=   3.1s
[CV 1/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=0;, s
core=0.761 total time=   1.6s
[CV 2/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=0;, s
core=0.766 total time=   1.6s
[CV 3/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=0;, s
core=0.789 total time=   2.3s
[CV 4/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=0;, s
core=0.769 total time=   1.7s
[CV 5/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=0;, s
core=0.766 total time=   1.6s
[CV 1/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=42;,
score=0.768 total time=   1.6s
[CV 2/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=42;,
score=0.777 total time=   1.6s
[CV 3/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=42;,
score=0.785 total time=   1.6s
[CV 4/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=42;,
score=0.759 total time=   1.6s
[CV 5/5] END max_depth=10, max_features=log2, n_estimators=100, random_state=42;,
score=0.766 total time=   1.9s
[CV 1/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=0;, s
core=0.768 total time=   3.7s
[CV 2/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=0;, s
core=0.765 total time=   3.1s
[CV 3/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=0;, s
core=0.795 total time=   3.1s
[CV 4/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=0;, s
core=0.767 total time=   3.7s
[CV 5/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=0;, s
core=0.770 total time=   3.4s
[CV 1/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=42;,
score=0.766 total time=   3.1s
[CV 2/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=42;,
score=0.773 total time=   3.1s
[CV 3/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=42;,
score=0.791 total time=   4.0s
[CV 4/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=42;,
score=0.767 total time=   3.2s
[CV 5/5] END max_depth=10, max_features=log2, n_estimators=200, random_state=42;,
score=0.765 total time=   3.1s
[CV 1/5] END max_depth=10, max_features=None, n_estimators=100, random_state=0;, s
core=0.746 total time=   7.2s
[CV 2/5] END max_depth=10, max_features=None, n_estimators=100, random_state=0;, s
core=0.744 total time=   6.3s
[CV 3/5] END max_depth=10, max_features=None, n_estimators=100, random_state=0;, s
core=0.769 total time=   7.3s
[CV 4/5] END max_depth=10, max_features=None, n_estimators=100, random_state=0;, s
core=0.742 total time=   6.3s
[CV 5/5] END max_depth=10, max_features=None, n_estimators=100, random_state=0;, s
core=0.755 total time=   7.2s
[CV 1/5] END max_depth=10, max_features=None, n_estimators=100, random_state=42;,
score=0.743 total time=   6.4s
[CV 2/5] END max_depth=10, max_features=None, n_estimators=100, random_state=42;,
score=0.755 total time=   7.1s
[CV 3/5] END max_depth=10, max_features=None, n_estimators=100, random_state=42;,
score=0.774 total time=   6.3s
[CV 4/5] END max_depth=10, max_features=None, n_estimators=100, random_state=42;,
score=0.746 total time=   7.2s
[CV 5/5] END max_depth=10, max_features=None, n_estimators=100, random_state=42;,
score=0.750 total time=   6.3s
[CV 1/5] END max_depth=10, max_features=None, n_estimators=200, random_state=0;, s
core=0.747 total time=  13.5s
[CV 2/5] END max_depth=10, max_features=None, n_estimators=200, random_state=0;, s
core=0.746 total time=  13.6s
[CV 3/5] END max_depth=10, max_features=None, n_estimators=200, random_state=0;, s
core=0.774 total time=  13.8s
[CV 4/5] END max_depth=10, max_features=None, n_estimators=200, random_state=0;, s
core=0.744 total time=  13.7s
[CV 5/5] END max_depth=10, max_features=None, n_estimators=200, random_state=0;, s
core=0.755 total time=  13.7s
[CV 1/5] END max_depth=10, max_features=None, n_estimators=200, random_state=42;,
score=0.741 total time=  13.5s
[CV 2/5] END max_depth=10, max_features=None, n_estimators=200, random_state=42;,
score=0.757 total time=  13.4s
[CV 3/5] END max_depth=10, max_features=None, n_estimators=200, random_state=42;,
score=0.773 total time=  13.5s
[CV 4/5] END max_depth=10, max_features=None, n_estimators=200, random_state=42;,
score=0.755 total time=  13.3s
[CV 5/5] END max_depth=10, max_features=None, n_estimators=200, random_state=42;,
score=0.755 total time=  13.4s
```

Out[47]:

```
▸ GridSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [48]:

```
print(gscv.best_params_)
```

```
{'max_depth': None, 'max_features': 'sqrt', 'n_estimators': 200, 'random_state': 4
2}
```

In [49]:

```
from sklearn.utils import class_weight
rf=RandomForestClassifier(max_features="sqrt",max_depth=None,n_estimators=200,rand
rf.fit(x_train,y_train)
```

Out[49]:

```
▾ RandomForestClassifier
RandomForestClassifier(class_weight='balanced', n_estimators=200,
                       random_state=42)
```

In [60]:
```
y_pred2=rf.predict(x_test)
accu_score=round(accuracy_score(y_test,y_pred2),2)
print("Accuracy Score",round(accuracy_score(y_test,y_pred2)*100,2),"%")
```

```
Accuracy Score 84.56 %
```

In [61]:
```
from sklearn.metrics import accuracy_score,f1_score,precision_score,recall_score,lo
fscore=f1_score(y_test,y_pred2,average="micro")
pscore=precision_score(y_test,y_pred2,average="micro")
rscore=recall_score(y_test,y_pred2,average="micro")
jscore=jaccard_score(y_test,y_pred2,average="micro")
Lloss=log_loss(y_test,y_pred2)
print("F1 score ",fscore)
print("Precision score ",pscore)
print("Recall score ",rscore)
print("Jaccard score ",jscore)
print("Log Loss",Lloss)
```

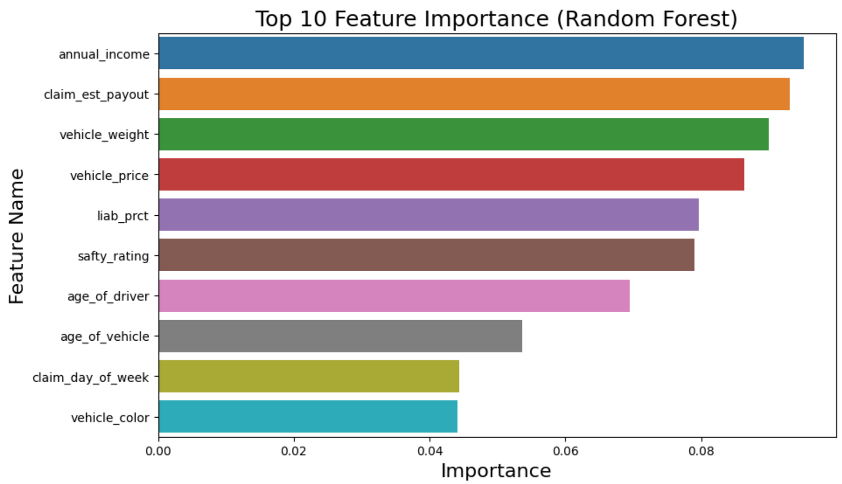
```
F1 score  0.8455717488789237
Precision score  0.8455717488789237
Recall score  0.8455717488789237
Jaccard score  0.7324593347899976
Log Loss 5.566158356895614
```

In [62]:
```
imp_df=pd.DataFrame({
    "Feature Name":x_train.columns,
    "Importance":rf.feature_importances_
})

fi=imp_df.sort_values(by="Importance",ascending=False)
fi2=fi.head(10)
fi2
```

Out[62]:

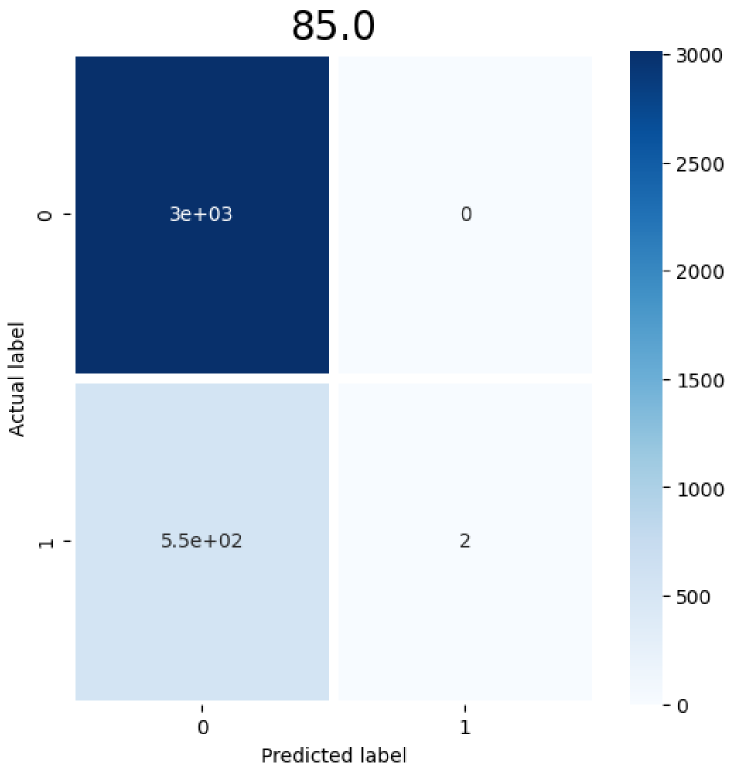
	Feature Name	Importance
4	annual_income	0.095062
16	claim_est_payout	0.093071
21	vehicle_weight	0.089984
19	vehicle_price	0.086360
13	liab_prct	0.079648
3	safty_rating	0.079032
0	age_of_driver	0.069482
17	age_of_vehicle	0.053655
9	claim_day_of_week	0.044307
20	vehicle_color	0.044168

In [63]:
```
plt.figure(figsize=(10,6))
sns.barplot(data=fi2,x="Importance",y="Feature Name")
plt.title("Top 10 Feature Importance (Random Forest)",fontsize=18)
plt.xlabel("Importance",fontsize=16)
plt.ylabel("Feature Name",fontsize=16)
plt.show()
```



In [64]:
```
from sklearn.metrics import confusion_matrix
cm=confusion_matrix(y_test,y_pred2)
plt.figure(figsize=(6,6))
sns.heatmap(data=cm,linewidths=5,annot=True,cmap="Blues")
plt.ylabel("Actual label")
plt.xlabel("Predicted label")
plt.title(accu_score*100,size=20)
```

Out[64]:
```
Text(0.5, 1.0, '85.0')
```



In [64]:
```
#Completed.........................
```