Insurance Claims Fraud Detection

Problem Statement

- Business case:

Insurance fraud is a huge problem in the industry. It's difficult to identify fraud claims. Machine Learning is in a unique position to help the Auto Insurance industry with this problem.

In this project, you are provided a dataset which has the details of the insurance policy along with the customer details. It also has the details of the accident on the basis of which the claims have been made.

In this example, you will be working with some auto insurance data to demonstrate how you can create a predictive model that predicts if an insurance claim is fraudulent or not.

In [ ]:

Import the Libraries

In [1]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Import the Dataset

In [2]:
```
data = pd.read_csv("Insurance_Fraud_Claim.csv")
data.head()
```

Out[2]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deducta
0	328	48	521585	17-10-2014	OH	250/500	1
1	228	42	342868	27-06-2006	IN	250/500	2
2	134	29	687698	06-09-2000	OH	100/300	2
3	256	41	227811	25-05-1990	IL	250/500	2
4	228	44	367455	06-06-2014	IL	500/1000	1

5 rows × 40 columns

In [3]:
```
#checking the rows and columns
data.shape
```

Out[3]:
```
(1000, 40)
```

In [4]:
```
#show the all columns
data.columns
```

Out[4]:
```
Index(['months_as_customer', 'age', 'policy_number', 'policy_bind_date',
       'policy_state', 'policy_csl', 'policy_deductable',
       'policy_annual_premium', 'umbrella_limit', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       'insured_relationship', 'capital-gains', 'capital-loss',
       'incident_date', 'incident_type', 'collision_type', 'incident_severity',
       'authorities_contacted', 'incident_state', 'incident_city',
       'incident_location', 'incident_hour_of_the_day',
       'number_of_vehicles_involved', 'property_damage', 'bodily_injuries',
       'witnesses', 'police_report_available', 'total_claim_amount',
       'injury_claim', 'property_claim', 'vehicle_claim', 'auto_make',
       'auto_model', 'auto_year', 'fraud_reported', '_c39'],
      dtype='object')
```

In [5]:
```
#checking the information of the datasets
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 40 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   months_as_customer           1000 non-null   int64  
 1   age                          1000 non-null   int64  
 2   policy_number                1000 non-null   int64  
 3   policy_bind_date             1000 non-null   object 
 4   policy_state                 1000 non-null   object 
 5   policy_csl                   1000 non-null   object 
 6   policy_deductable            1000 non-null   int64  
 7   policy_annual_premium        1000 non-null   float64
 8   umbrella_limit               1000 non-null   int64  
 9   insured_zip                  1000 non-null   int64  
 10  insured_sex                  1000 non-null   object 
 11  insured_education_level      1000 non-null   object 
 12  insured_occupation           1000 non-null   object 
 13  insured_hobbies              1000 non-null   object 
 14  insured_relationship         1000 non-null   object 
 15  capital-gains                1000 non-null   int64  
 16  capital-loss                 1000 non-null   int64  
 17  incident_date                1000 non-null   object 
 18  incident_type                1000 non-null   object 
 19  collision_type               1000 non-null   object 
 20  incident_severity            1000 non-null   object 
 21  authorities_contacted        1000 non-null   object 
 22  incident_state               1000 non-null   object 
 23  incident_city                1000 non-null   object 
 24  incident_location            1000 non-null   object 
 25  incident_hour_of_the_day     1000 non-null   int64  
 26  number_of_vehicles_involved  1000 non-null   int64  
 27  property_damage              1000 non-null   object 
 28  bodily_injuries              1000 non-null   int64  
 29  witnesses                    1000 non-null   int64  
 30  police_report_available      1000 non-null   object 
 31  total_claim_amount           1000 non-null   int64  
 32  injury_claim                 1000 non-null   int64  
 33  property_claim               1000 non-null   int64  
 34  vehicle_claim                1000 non-null   int64  
 35  auto_make                    1000 non-null   object 
 36  auto_model                   1000 non-null   object 
 37  auto_year                    1000 non-null   int64  
 38  fraud_reported               1000 non-null   object 
 39  _c39                         0 non-null      float64
dtypes: float64(2), int64(17), object(21)
memory usage: 312.6+ KB
```

In [6]:
```
#cheking the missing values
data.isnull().sum()
```

Out[6]:
```
months_as_customer              0
age                             0
policy_number                   0
policy_bind_date                0
policy_state                    0
policy_csl                      0
policy_deductable               0
policy_annual_premium           0
umbrella_limit                  0
insured_zip                     0
insured_sex                     0
insured_education_level         0
insured_occupation              0
insured_hobbies                 0
insured_relationship            0
capital-gains                   0
capital-loss                    0
incident_date                   0
incident_type                   0
collision_type                  0
incident_severity               0
authorities_contacted           0
incident_state                  0
incident_city                   0
incident_location               0
incident_hour_of_the_day        0
number_of_vehicles_involved     0
property_damage                 0
bodily_injuries                 0
witnesses                       0
police_report_available         0
total_claim_amount              0
injury_claim                    0
property_claim                  0
vehicle_claim                   0
auto_make                       0
auto_model                      0
auto_year                       0
fraud_reported                  0
_c39                         1000
dtype: int64
```

In [7]:
```
data.describe()
```

Out[7]:

	months_as_customer	age	policy_number	policy_deductable	policy_annual_premium
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000
mean	203.954000	38.948000	546238.648000	1136.000000	1256.406150
std	115.113174	9.140287	257063.005276	611.864673	244.167395
min	0.000000	19.000000	100804.000000	500.000000	433.330000
25%	115.750000	32.000000	335980.250000	500.000000	1089.607500
50%	199.500000	38.000000	533135.000000	1000.000000	1257.200000
75%	276.250000	44.000000	759099.750000	2000.000000	1415.695000
max	479.000000	64.000000	999435.000000	2000.000000	2047.590000

In [8]:
```
# Dropping columns
data.drop('_c39',axis=1,inplace=True)
```

In [9]:
```
#lets do Lable enconding coding to make more features

le = LabelEncoder()
le_count = 0

# Iterate through the columns
for col in data:
    if data[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(data[col].unique())) <= 2:
            # Train on the training data
            le.fit(data[col])
            # Transform both training and testing data
            data[col] = le.transform(data[col])
            # Keep track of how many columns were label encoded
            le_count += 1

print('%d columns were label encoded.' % le_count)
```

```
2 columns were label encoded.
```

In [10]:
```
data.head()
```

Out[10]:

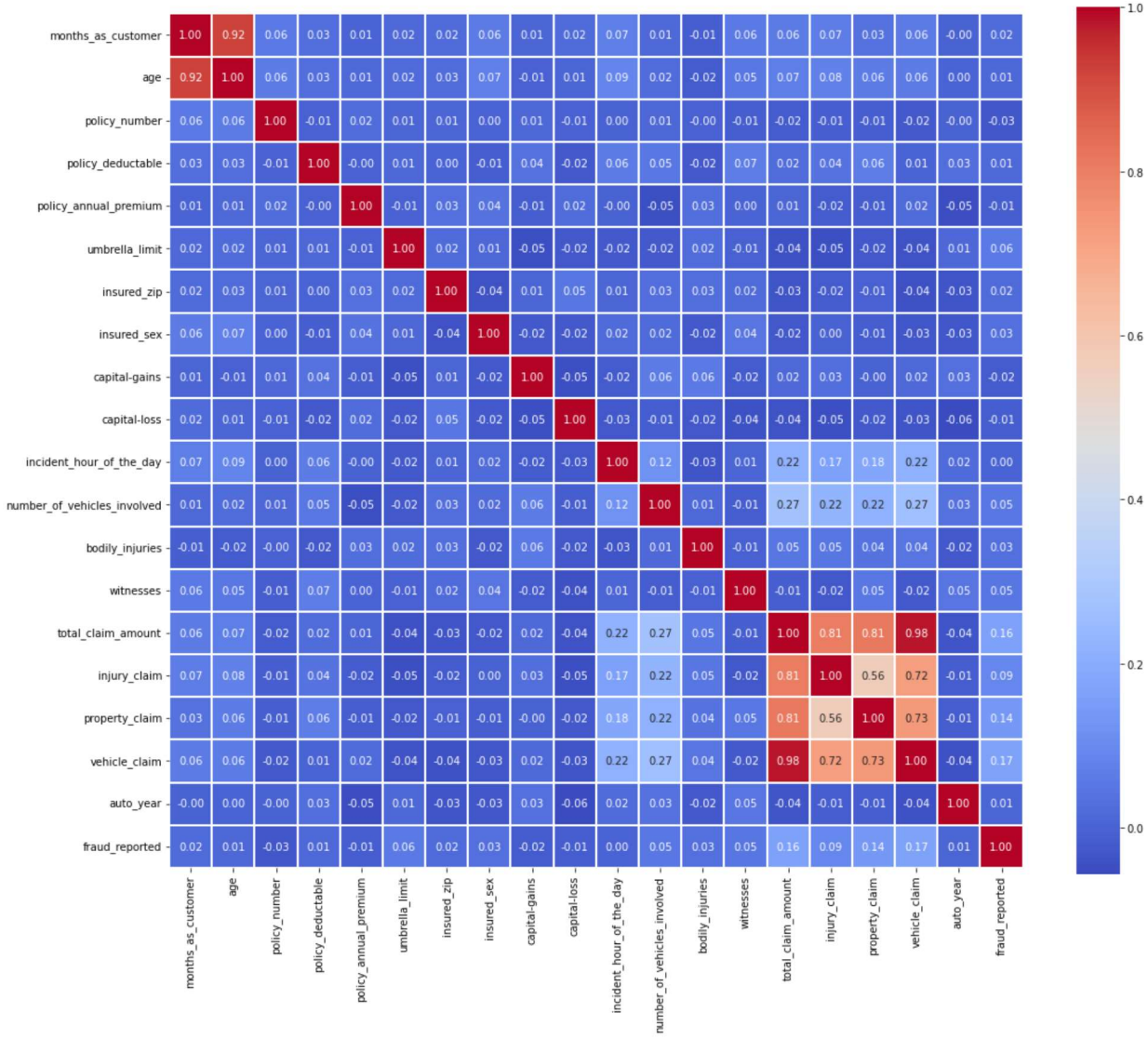
	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deducta
0	328	48	521585	17-10-2014	OH	250/500	1
1	228	42	342868	27-06-2006	IN	250/500	2
2	134	29	687698	06-09-2000	OH	100/300	2
3	256	41	227811	25-05-1990	IL	250/500	2
4	228	44	367455	06-06-2014	IL	500/1000	1

5 rows × 39 columns

Correlation Matrix

In [11]:
```
plt.figure(figsize=(18,15))
sns.heatmap(data.corr(),cbar=True,square=True,fmt='.2f',linewidths=0.1,annot=True,annc
```

Out[11]:
```
<AxesSubplot:>
```



In [ ]:

In [12]:
```
#Count the values
colum_name =[]
unique_value=[]
# Iterate through the columns
for col in data:
    if data[col].dtype == 'object':
        # If 2 or fewer unique categories
        colum_name.append(str(col))
        unique_value.append(data[col].nunique())
table= pd.DataFrame()
table['Col_name'] = colum_name
table['Value']= unique_value

table=table.sort_values('Value',ascending=False)
table
```

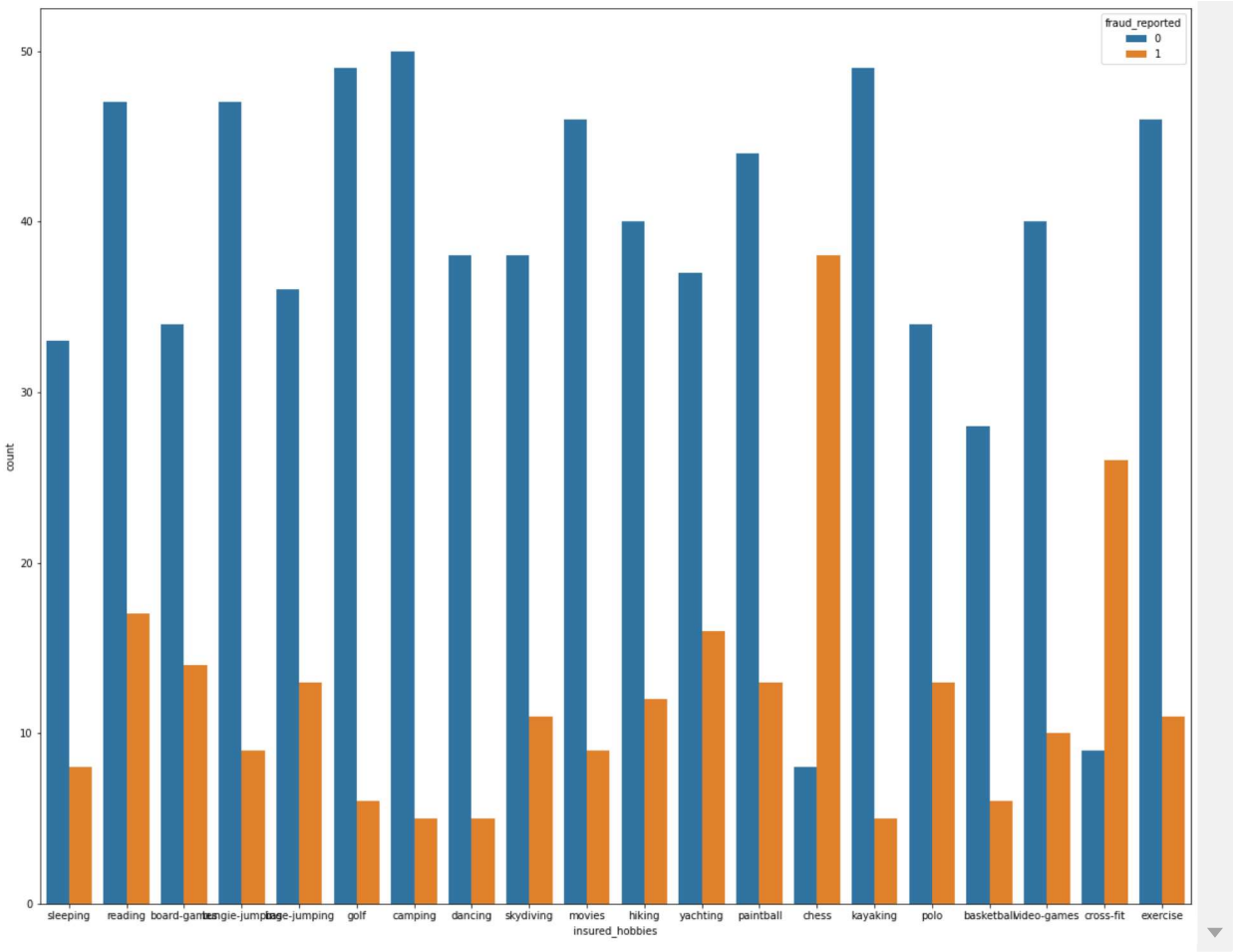
Out[12]:

	Col_name	Value
14	incident_location	1000
0	policy_bind_date	951
7	incident_date	60
18	auto_model	39
5	insured_hobbies	20
4	insured_occupation	14
17	auto_make	14
3	insured_education_level	7
12	incident_state	7
13	incident_city	7
6	insured_relationship	6
11	authorities_contacted	5
8	incident_type	4
10	incident_severity	4
9	collision_type	4
1	policy_state	3
2	policy_csl	3
15	property_damage	3
16	police_report_available	3

In [13]:
```
# droping columns based on above result
data.drop(['incident_location','policy_bind_date','incident_date','auto_model','insure
```

In [14]:
```
f, ax = plt.subplots(figsize=(20, 16))
sns.countplot(x='insured_hobbies',hue='fraud_reported',data=data)
```

Out[14]:
```
<AxesSubplot:xlabel='insured_hobbies', ylabel='count'>
```

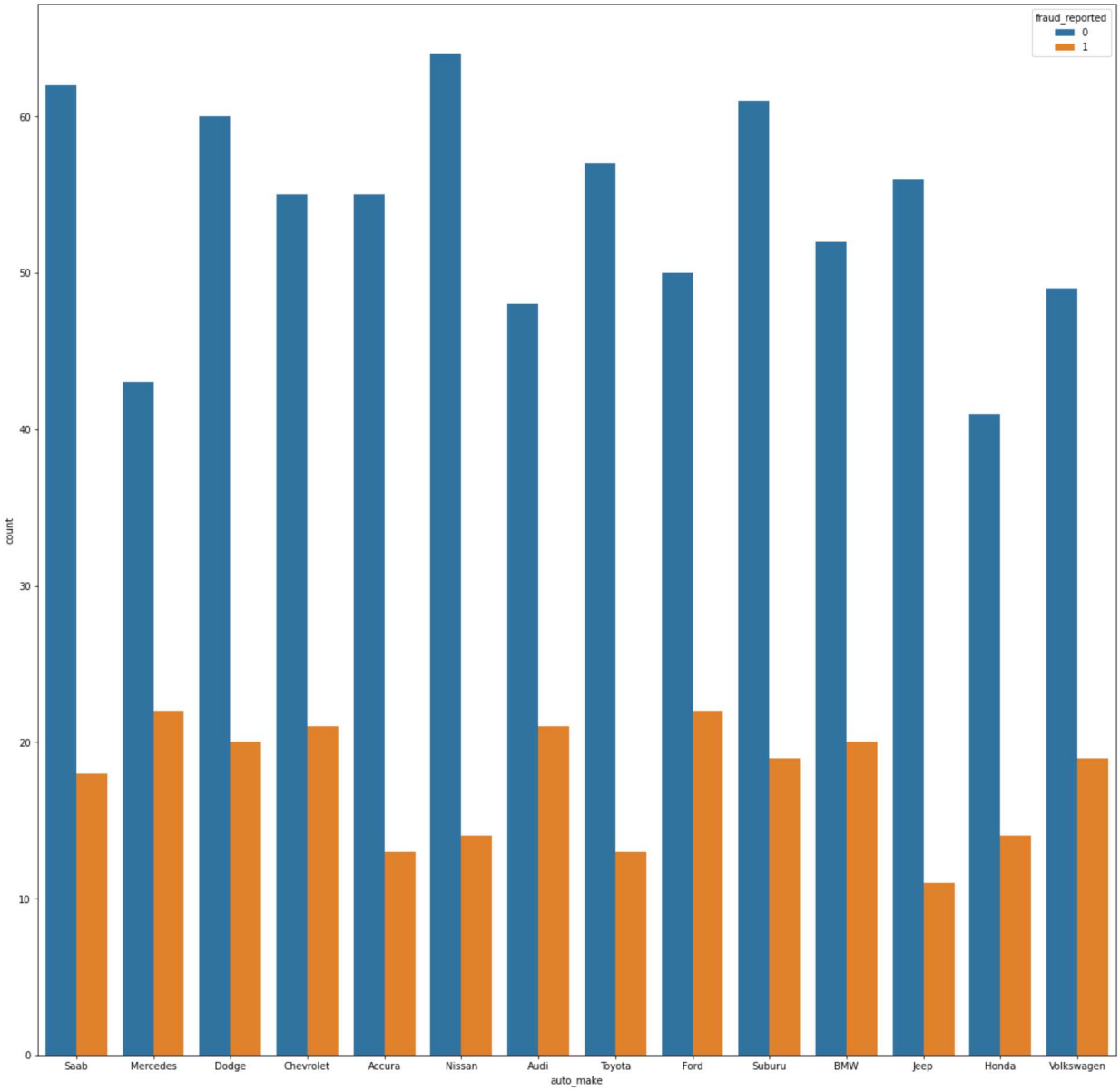


Create additional 'other' column if Insured hobbies are not chess and cross fit.

In [15]:
```
data['insured_hobbies']=data['insured_hobbies'].apply(lambda x :'Other' if x!='chess'
```

In [16]:
```
f, ax = plt.subplots(figsize=(20, 20))
sns.countplot(x='auto_make',hue='fraud_reported',data=data)
```

Out[16]:
```
<AxesSubplot:xlabel='auto_make', ylabel='count'>
```



In [17]:
```
data['insured_hobbies'].unique()
```

Out[17]:
```
array(['Other', 'chess', 'cross-fit'], dtype=object)
```

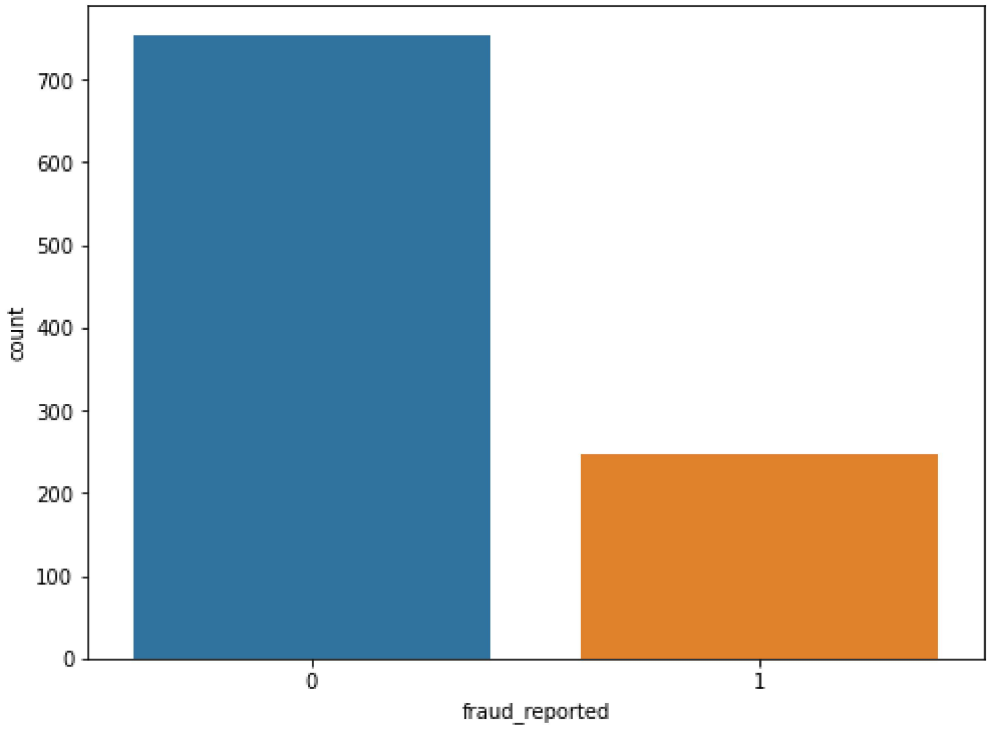
In [18]:
```
data = pd.get_dummies(data)
print('Training Features shape: ', data.shape)
```

```
Training Features shape:  (1000, 92)
```

Lets check if Data is balanced data or not

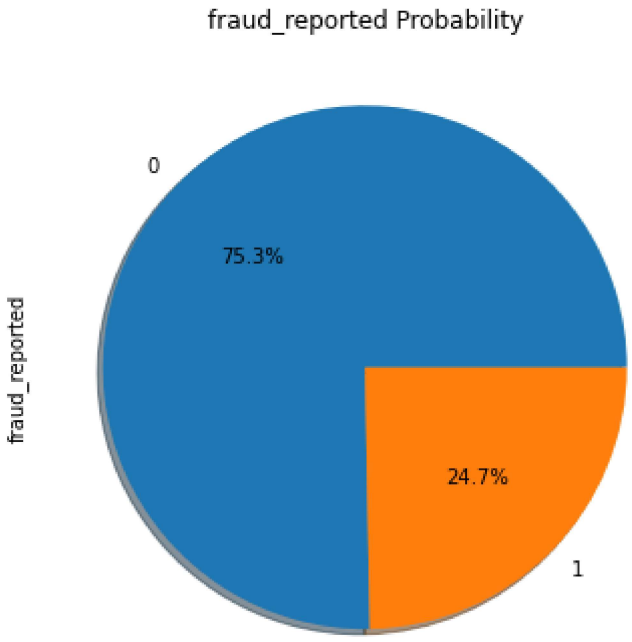
In [19]:
```
f, ax = plt.subplots(figsize=(8, 6))
sns.countplot(x='fraud_reported',data=data)
```

Out[19]:
```
<AxesSubplot:xlabel='fraud_reported', ylabel='count'>
```



In [20]:
```
ax=plt.subplots(figsize=(6,6))
ax=data['fraud_reported'].value_counts().plot.pie(explode=[0,0],autopct='%1.1f%%',shad
ax.set_title('fraud_reported Probability')
```

Out[20]:
```
Text(0.5, 1.0, 'fraud_reported Probability')
```



Splitting features and Target

In [21]:
```
y=data['fraud_reported']
x= data.drop('fraud_reported',axis=1)
```

Splitting the data into Training data & Testing Data

In [22]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,random_state=0

print('shape of x_train =', x_train.shape)
print('shape of y_train =', y_train.shape)
print('shape of x_test =', x_test.shape)
print('shape of y_test =', y_test.shape)
```

```
shape of x_train = (700, 91)
shape of y_train = (700,)
shape of x_test = (300, 91)
shape of y_test = (300,)
```

Model Training

In [23]:
```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

Logistic Regression

In [24]:
```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(x_train,y_train)
predictions = lr.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[221   2]
 [ 76   1]]
              precision    recall  f1-score   support

           0       0.74      0.99      0.85       223
           1       0.33      0.01      0.03        77

    accuracy                           0.74       300
   macro avg       0.54      0.50      0.44       300
weighted avg       0.64      0.74      0.64       300

0.74
```

Decision Tree Classifier

In [25]:
```
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train,y_train)
predictions = dt.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[201  22]
 [ 38  39]]
              precision    recall  f1-score   support

           0       0.84      0.90      0.87       223
           1       0.64      0.51      0.57        77

    accuracy                           0.80       300
   macro avg       0.74      0.70      0.72       300
weighted avg       0.79      0.80      0.79       300

0.8
```

Random Forest Classifier

In [33]:
```
from sklearn.ensemble import RandomForestClassifier
rfc=RandomForestClassifier(n_estimators=100,criterion='gini')
rfc.fit(x_train,y_train)
predictions = rfc.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[209  14]
 [ 44  33]]
              precision    recall  f1-score   support

           0       0.83      0.94      0.88       223
           1       0.70      0.43      0.53        77

    accuracy                           0.81       300
   macro avg       0.76      0.68      0.71       300
weighted avg       0.79      0.81      0.79       300

0.8066666666666666
```

Gussian Naive Bayes

In [34]:
```
from sklearn.naive_bayes import GaussianNB
gnb=GaussianNB()
gnb.fit(x_train,y_train)
predictions = gnb.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[184  39]
 [ 54  23]]
              precision    recall  f1-score   support

           0       0.77      0.83      0.80       223
           1       0.37      0.30      0.33        77

    accuracy                           0.69       300
   macro avg       0.57      0.56      0.56       300
weighted avg       0.67      0.69      0.68       300

0.69
```

K Nearest Neighbors Classifier

In [35]:
```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=5,metric='minkowski',p=2)
knn.fit(x_train,y_train)
predictions = knn.predict(x_test)
print(confusion_matrix(y_test, predictions))
print(classification_report(y_test, predictions))
print(accuracy_score(y_test, predictions))
```

```
[[201  22]
 [ 67  10]]
              precision    recall  f1-score   support

           0       0.75      0.90      0.82       223
           1       0.31      0.13      0.18        77

    accuracy                           0.70       300
   macro avg       0.53      0.52      0.50       300
weighted avg       0.64      0.70      0.66       300

0.7033333333333334
```

Conclusion:

In [36]:
```
import numpy as np
a=np.array(y_test)
predicted=np.array(rfc.predict(x_test))
data.com=pd.DataFrame({"original":a,"predicted":predicted},index=range(len(a)))
data.com
```

Out[36]:

	original	predicted
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
295	0	0
296	0	1
297	0	1
298	0	0
299	0	0

300 rows × 2 columns

In [ ]: