Insurance fraud detection is a challenging problem, given the variety of fraud patterns and relatively small ratio of known frauds in typical samples. This model helps to detect fraud in insurance claims which ultimately helps insurance companies to save their losses.

Insurance frauds cover the range of improper activities which an individual may commit in order to achieve a favorable outcome from the insurance company. This could range from staging the incident, misrepresenting the situation including the relevant actors and the cause of incident and finally the extent of damage caused.

Potential situations could include:

- Covering-up for a situation that wasn't covered under insurance (e.g. drunk driving, performing risky acts, illegal activities etc.)
- Misrepresenting the context of the incident: This could include transferring the blame to incidents where the insured party is to blame, failure to take agreed upon safety measures
- Infiating the impact of the incident: Increasing the estimate of loss incurred either through addition of unrelated losses (faking losses) or attributing increased cost to the losses
- The insurance industry has grappled with the challenge of insurance claim fraud from the very start. On one hand, there is the challenge of impact to customer satisfaction through delayed payouts or prolonged investigation during a period of stress. Additionally, there are costs of investigation and pressure from insurance industry regulators. On the other hand, improper payouts cause a hit to profitability and encourage similar delinquent behavior from other policy holders.

Analysis Content

Import Packages

In [2]:
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import seaborn as sns
import matplotlib.pyplot as plt
```

Data Read

In [3]:
```
df=pd.read_csv("file:///C:/Users/angsh/OneDrive/Desktop/PRAXIS/Own%20Projects/ML/insurance_claims.csv")
```

In [4]:
```
df.head(2)
```

Out[4]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000

2 rows × 40 columns

In [5]:
```
df.tail(2)
```

Out[5]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_lin
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.92	50000
999	456	60	556080	1996-11-11	OH	250/500	1000	766.19	

2 rows × 40 columns

EDA

In [6]:
```
df.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 [7]:
```
df.shape
```

Out[7]:
```
(1000, 40)
```

Check for Missing Values

In [8]:
```
df.isnull().sum()
```

Out[8]:

```
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 [9]:
```
df.duplicated().sum()
```

Out[9]:
```
0
```

In [10]:
```
df.drop('_c39',axis=1,inplace=True)
```

In [11]:
```
# Deleteing the c_39 column as all the values in the feature set is NAN.
df.head(2)
```

Out[11]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	0
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	5000000

2 rows × 39 columns

In [12]:
```
df.describe().transpose()
```

Out[12]:

	count	mean	std	min	25%	50%	75%	max
months_as_customer	1000.0	2.039540e+02	1.151132e+02	0.00	115.7500	199.5	276.250	479.00
age	1000.0	3.894800e+01	9.140287e+00	19.00	32.0000	38.0	44.000	64.00
policy_number	1000.0	5.462386e+05	2.570630e+05	100804.00	335980.2500	533135.0	759099.750	999435.00
policy_deductable	1000.0	1.136000e+03	6.118647e+02	500.00	500.0000	1000.0	2000.000	2000.00
policy_annual_premium	1000.0	1.256406e+03	2.441674e+02	433.33	1089.6075	1257.2	1415.695	2047.59
umbrella_limit	1000.0	1.101000e+06	2.297407e+06	-1000000.00	0.0000	0.0	0.000	10000000.00
insured_zip	1000.0	5.012145e+05	7.170161e+04	430104.00	448404.5000	466445.5	603251.000	620962.00
capital-gains	1000.0	2.512610e+04	2.787219e+04	0.00	0.0000	0.0	51025.000	100500.00
capital-loss	1000.0	-2.679370e+04	2.810410e+04	-111100.00	-51500.0000	-23250.0	0.000	0.00
incident_hour_of_the_day	1000.0	1.164400e+01	6.951373e+00	0.00	6.0000	12.0	17.000	23.00
number_of_vehicles_involved	1000.0	1.839000e+00	1.018880e+00	1.00	1.0000	1.0	3.000	4.00
bodily_injuries	1000.0	9.920000e-01	8.201272e-01	0.00	0.0000	1.0	2.000	2.00
witnesses	1000.0	1.487000e+00	1.111335e+00	0.00	1.0000	1.0	2.000	3.00
total_claim_amount	1000.0	5.276194e+04	2.640153e+04	100.00	41812.5000	58055.0	70592.500	114920.00
injury_claim	1000.0	7.433420e+03	4.880952e+03	0.00	4295.0000	6775.0	11305.000	21450.00
property_claim	1000.0	7.399570e+03	4.824726e+03	0.00	4445.0000	6750.0	10885.000	23670.00
vehicle_claim	1000.0	3.792895e+04	1.888625e+04	70.00	30292.5000	42100.0	50822.500	79560.00
auto_year	1000.0	2.005103e+03	6.015861e+00	1995.00	2000.0000	2005.0	2010.000	2015.00

In [13]:
```
df.columns
```

Out[13]:
```
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'],
      dtype='object')
```

In [14]:
```
df['fraud_reported'].value_counts()
```

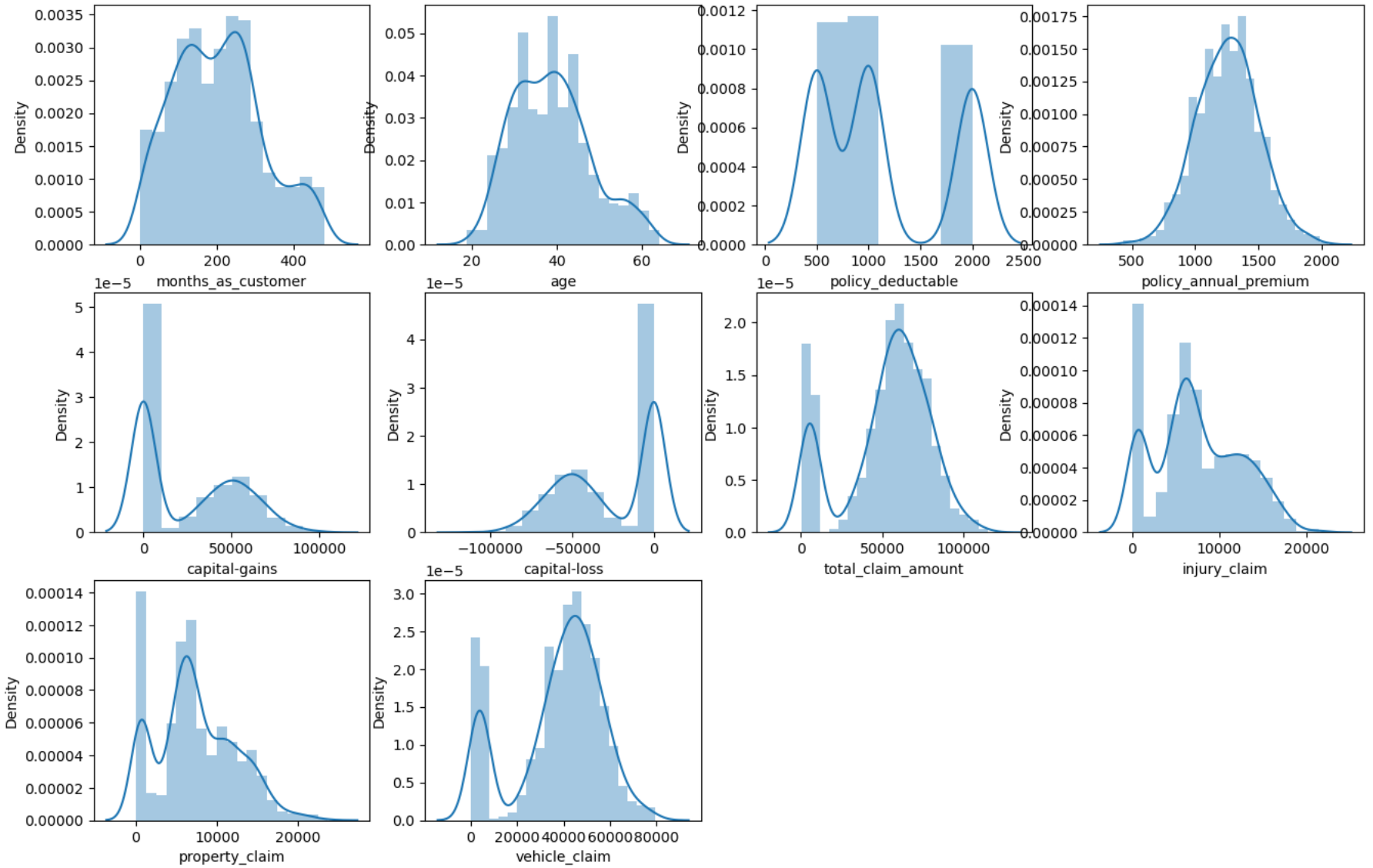
Out[14]:
```
N    753
Y    247
Name: fraud_reported, dtype: int64
```

In [15]:
```
fraud=df[df['fraud_reported']=='Y']
```

In [16]:
```
col=['months_as_customer','age','policy_deductable','policy_annual_premium','capital-gains','capital-loss','total_cla


plt.figure(figsize=(16,14))
k=1
for i in col :
    plt.subplot(4,4,k)
    sns.distplot(df[i])
    k=k+1
plt.show()
```

```
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



All the above data are almost normally distributed

In [17]:
```
# An umbrella policy is a policy that provides excess limits and gives additional excess coverage over the normal lim
df['umbrella_limit'].value_counts()
```

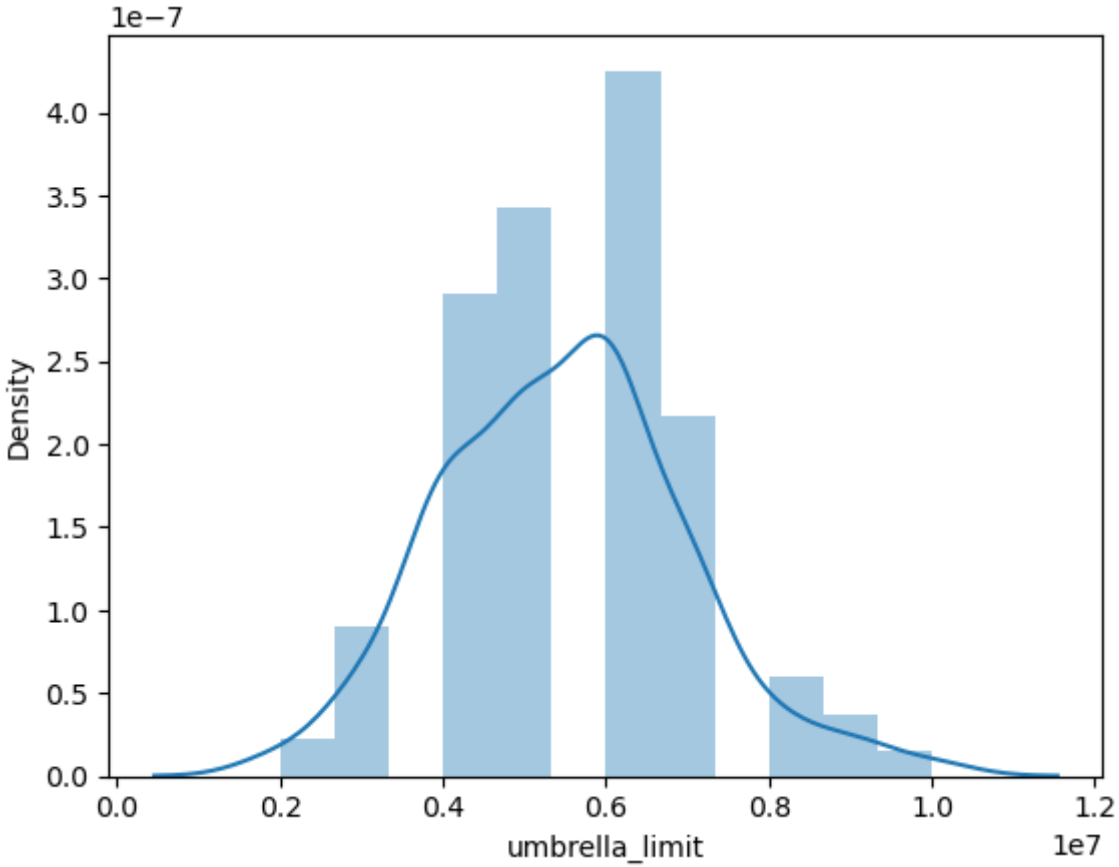
Out[17]:
```
 0           798
 6000000      57
 5000000      46
 4000000      39
 7000000      29
 3000000      12
 8000000       8
 9000000       5
 2000000       3
 10000000      2
-1000000       1
Name: umbrella_limit, dtype: int64
```

In [18]:
```
a=df[df['umbrella_limit'] >0]
```

In [19]:
```
#To know the distribution of the umbrella limit
sns.distplot(a['umbrella_limit'])
```

```
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level fun
ction with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

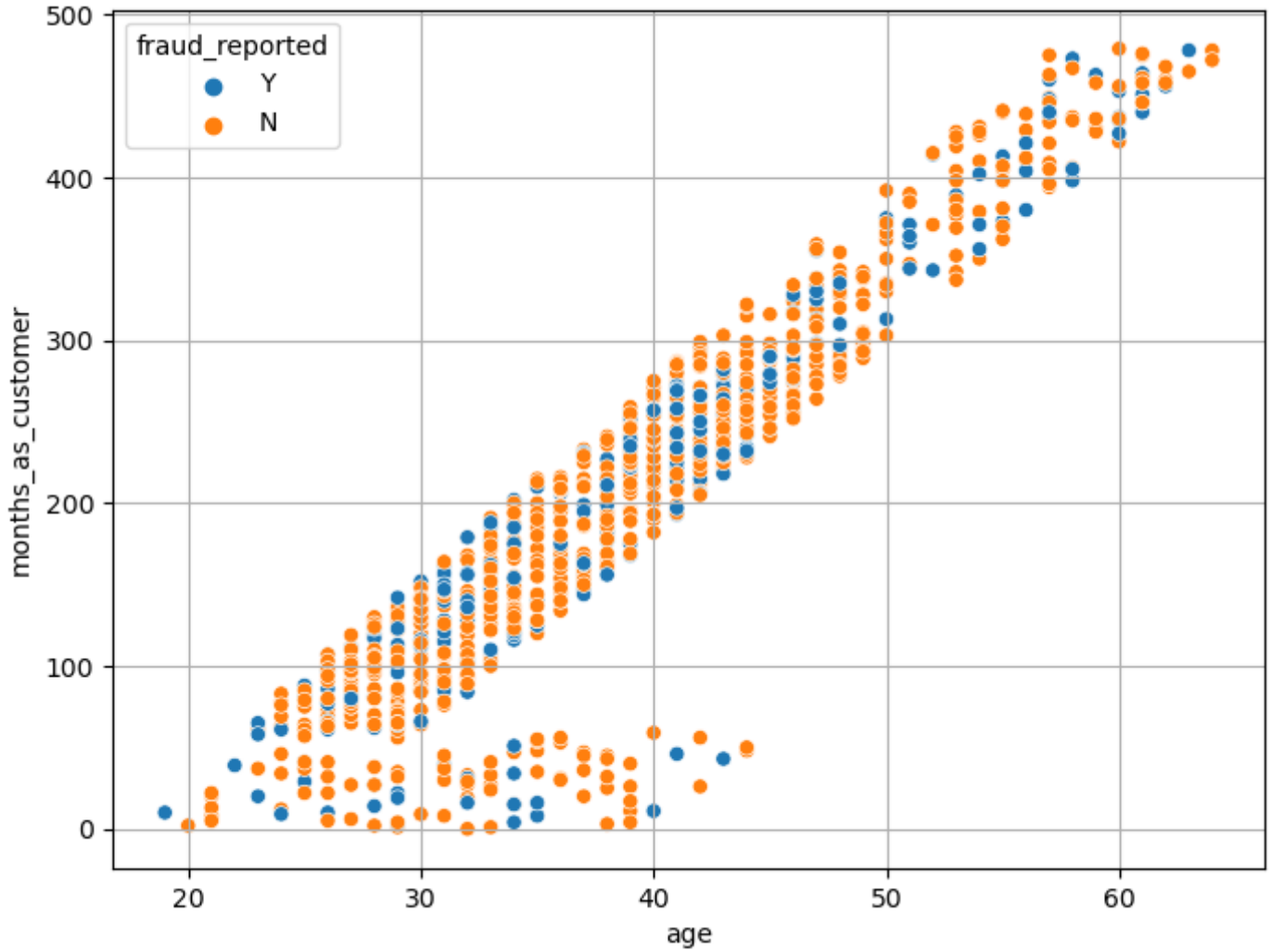
Out[19]: <AxesSubplot:xlabel='umbrella_limit', ylabel='Density'>



In [20]:
```
df.columns
```

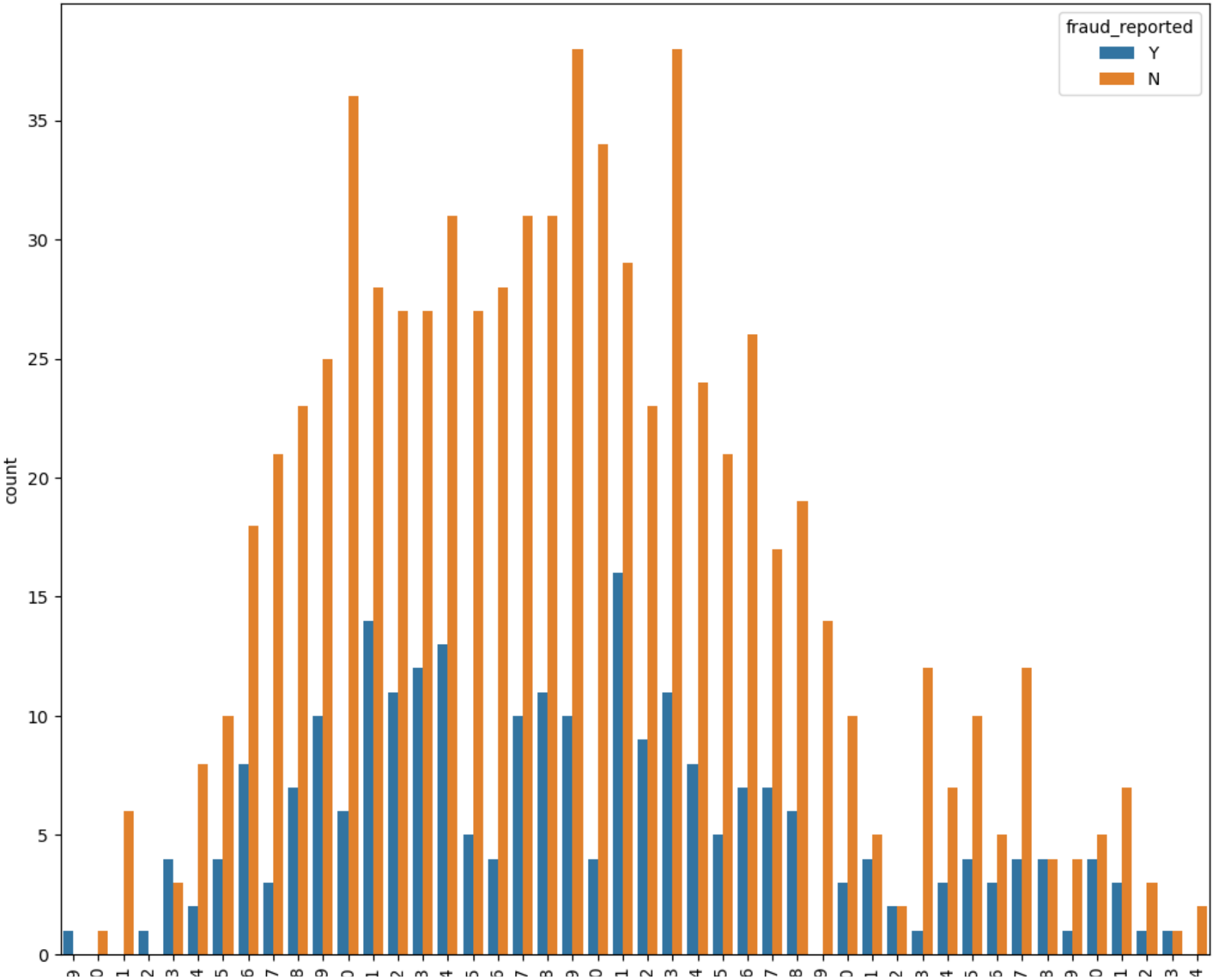
Out[20]:
```
Index(['months_as_customer', 'age', 'policy_number', 'policy_bind_date',
       'policy_state', 'policy_csl', 'policy_deductable',
       'policy_annual_premium', 'umbrella_limit', 'insured_zip', 'insured_sex',
       'insured_education_level', 'insured_occupation', 'insured_hobbies',
       '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'],
      dtype='object')
```

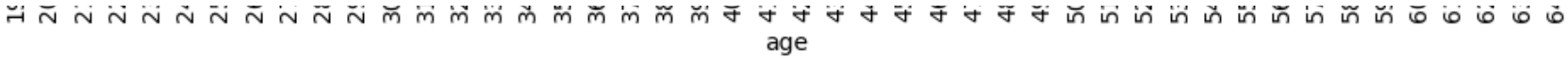
In [21]:
```
plt.figure(figsize=(8,6))
sns.scatterplot(x='age', y='months_as_customer',hue='fraud_reported',data=df)
plt.grid(True)
plt.show()
```



We do not have any pattern which justifies if customers with more years with the company are claiming fraud insurance.

In [22]:
```
plt.figure(figsize=(12,10))
sns.countplot(x='age',hue='fraud_reported',data=df)
plt.xticks(rotation=90)
plt.show()
```


fraud_reported
Y
N
count
0
5
10
15
20
25
30
35


age

In [23]:
```
# Taking a as a new dataframe where fraude reported is yes in the dataset.
a=df[df['fraud_reported'] == 'Y']

a[['policy_number','insured_occupation','insured_education_level','total_claim_amount']].sort_values('total_claim_amo
```

Out[23]:

	policy_number	insured_occupation	insured_education_level	total_claim_amount
149	217938	craft-repair	JD	112320
163	346940	prof-specialty	Masters	107900
479	753844	sales	MD	104610
145	515050	exec-managerial	Associate	99320
247	187775	other-service	JD	98670
91	127754	tech-support	Associate	98340
974	291006	transport-moving	JD	98280
23	115399	priv-house-serv	MD	98160
41	616337	transport-moving	Associate	97080
796	728025	machine-op-inspct	Masters	92730
926	752504	transport-moving	Masters	91520
848	953334	craft-repair	MD	90860
66	356590	tech-support	High School	89700
185	442795	tech-support	JD	88660
517	243226	armed-forces	High School	87960
628	730819	protective-serv	JD	87890
727	691115	farming-fishing	JD	86130
722	334749	handlers-cleaners	Associate	85900
829	951863	protective-serv	Masters	84920
593	209177	craft-repair	JD	84590

We have top 20 fraud claims with policy number and their occupation listed Policy number 217938 has claimed highest amount of 112320$ and is working as craft-repair

In [24]:
```
a['insured_education_level'].value_counts()
```

Out[24]:
```
JD             42
MD             38
High School    36
Associate      34
PhD            33
College        32
Masters        32
Name: insured_education_level, dtype: int64
```

In [25]:
```
#People who have educxation level as JD has claimed more fraud transactions
```

In [26]:
```
a['insured_occupation'].value_counts()
```

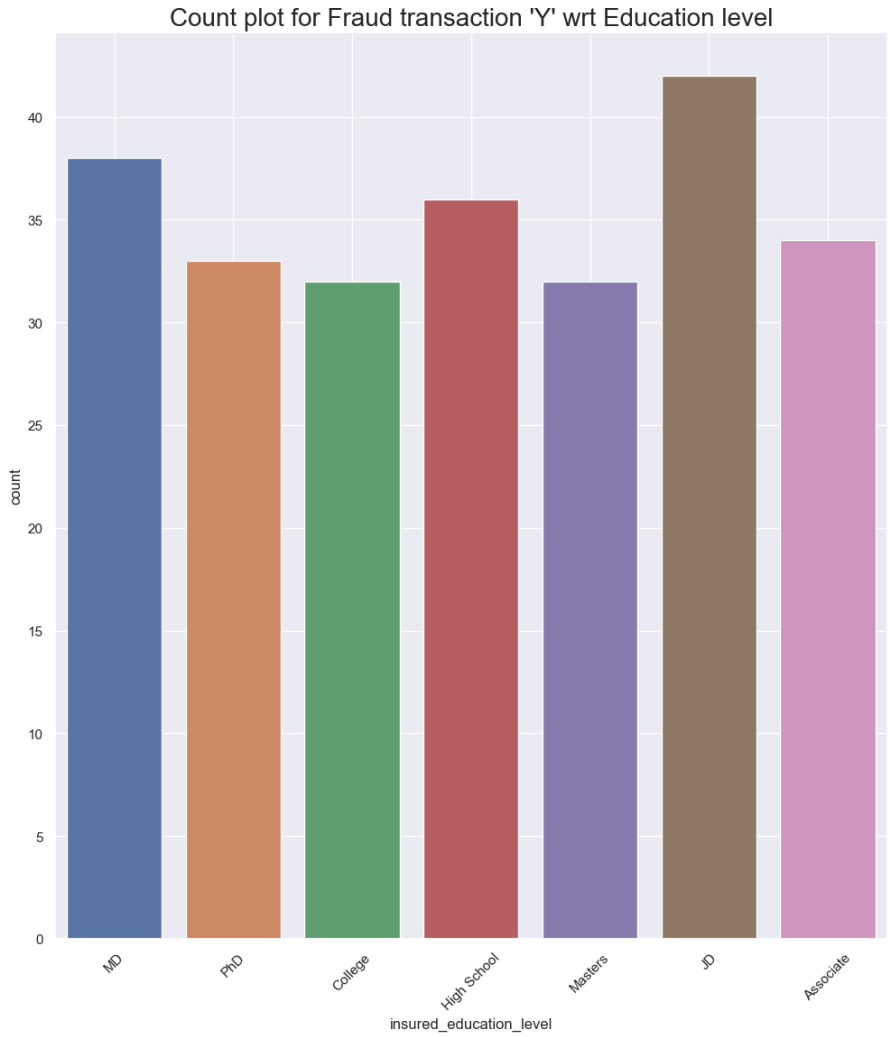
Out[26]:
```
exec-managerial      28
craft-repair         22
machine-op-inspct    22
tech-support         22
transport-moving     21
sales                21
prof-specialty       18
armed-forces         17
farming-fishing      16
protective-serv      14
priv-house-serv      12
other-service        12
handlers-cleaners    11
adm-clerical         11
Name: insured_occupation, dtype: int64
```

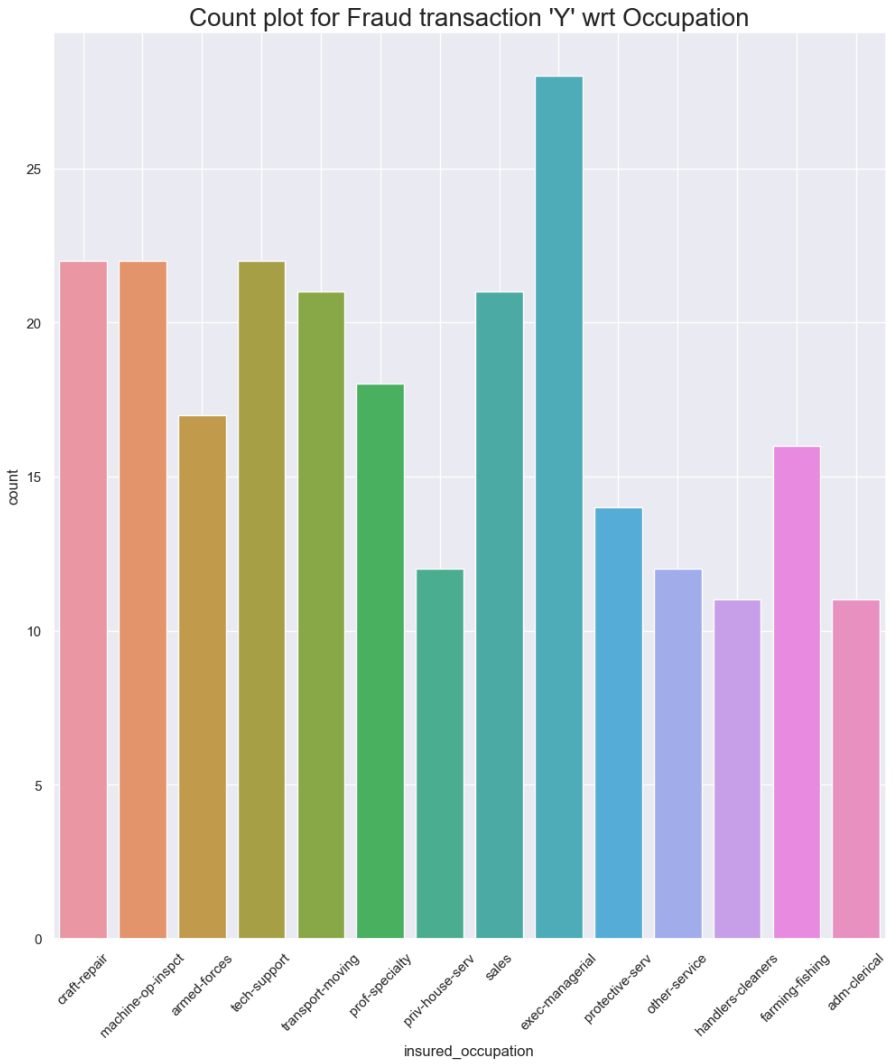
People who are working as exec-manager has claimed more fraud transactions

In [27]:
```
sns.set(style="darkgrid")

plt.figure(figsize=(20,12))
plt.subplot(1,2,1)
plt.title("Count plot for Fraud transaction 'Y' wrt Education level",fontsize=20)
sns.countplot('insured_education_level',data=a)
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.subplot(1,2,2)
plt.title("Count plot for Fraud transaction 'Y' wrt Occupation",fontsize=20)
sns.countplot('insured_occupation',data=a)
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```





People with occupation as Exec Manager seems to be doing more fraud transactions and people with JD level of education are also involved in more fraud transactions. Comparitively people with less education are claiming more fraud claims.

In [28]:
```
#Looking at below maximum clam amount as per insured occupation and education level.
a_claims=pd.pivot_table(a,values='total_claim_amount',index=['insured_occupation','insured_education_level']).sort_va

cm = sns.light_palette("blue", as_cmap=True)
a_claims.style.background_gradient(cmap=cm)
```

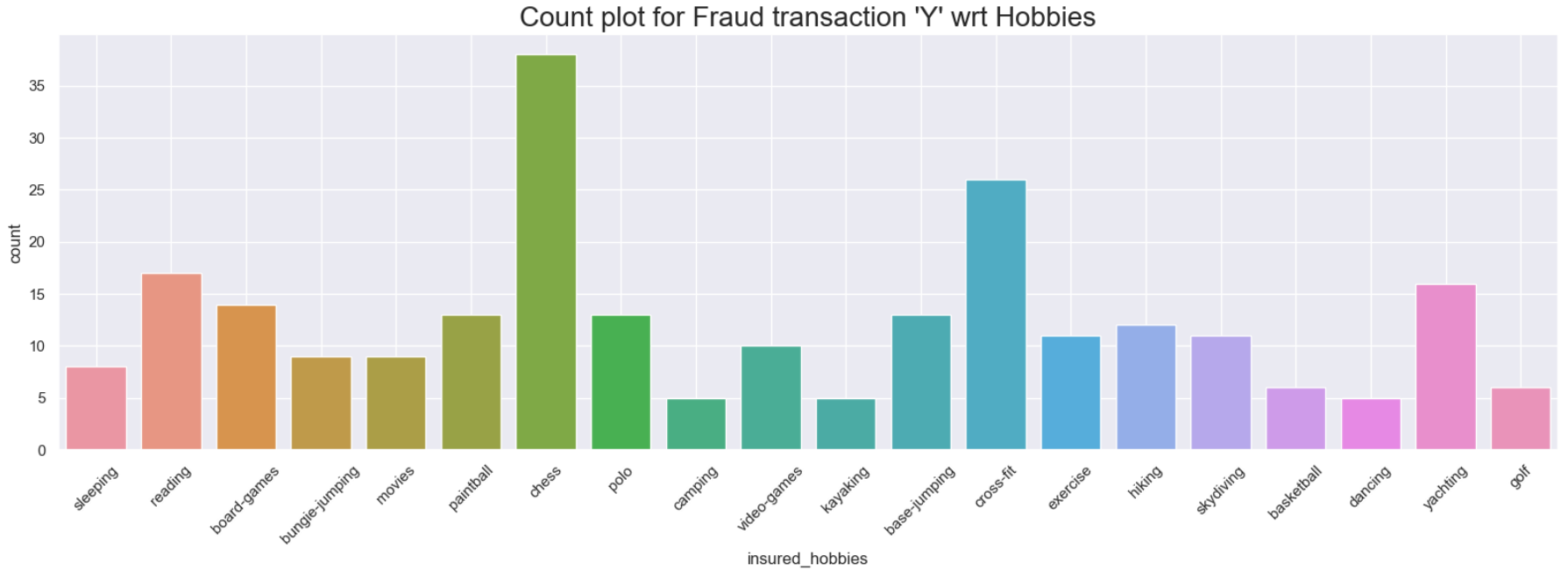
Out[28]:

		total_claim_amount
insured_occupation	**insured_education_level**	
protective-serv	**JD**	87890.000000
handlers-cleaners	**Associate**	85900.000000
priv-house-serv	**MD**	81353.333333
other-service	**JD**	81135.000000
tech-support	**Associate**	80113.333333
transport-moving	**JD**	79225.000000
	Associate	76873.333333
	High School	76010.000000
protective-serv	**MD**	75290.000000
	College	75185.000000

People from occupation sector Protective-services and education level of JD has highest fraud claimed amount of 87,890$

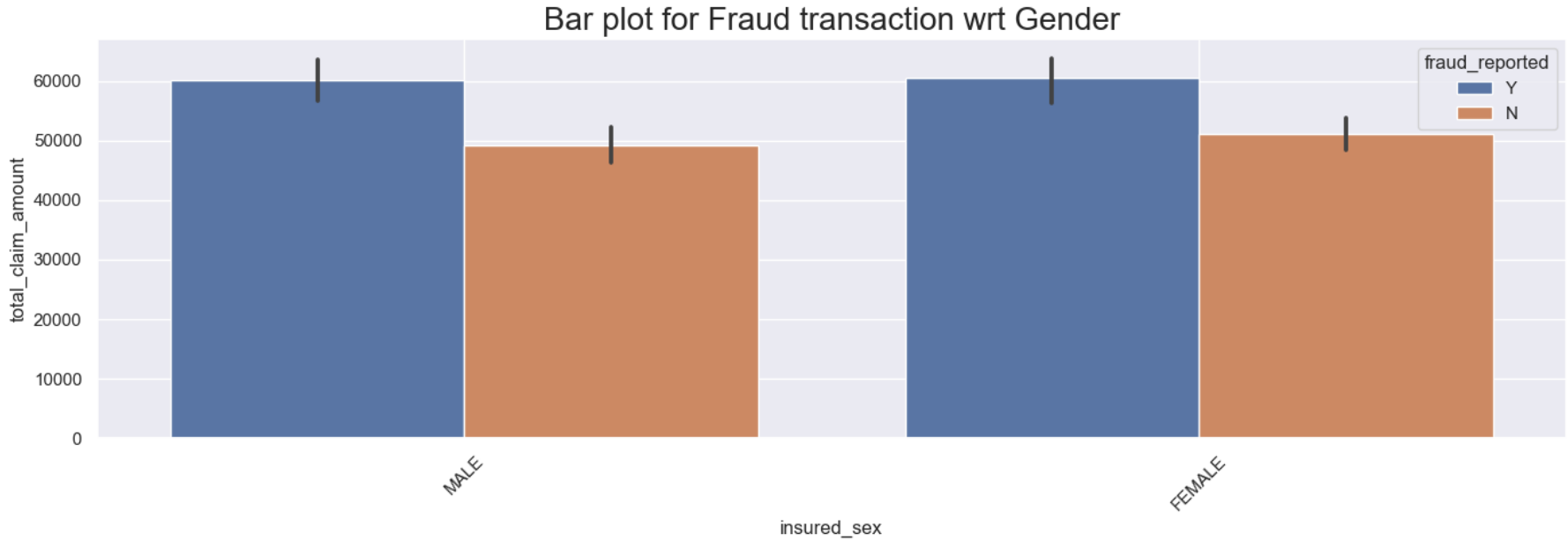
In [29]:
```
plt.figure(figsize=(16,6))
plt.title("Count plot for Fraud transaction 'Y' wrt Hobbies",fontsize=20)
sns.countplot('insured_hobbies',data=a)
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



People who play more chess have claimed more fraud followed by crossfit

In [30]:
```
plt.figure(figsize=(14,5))
plt.title("Bar plot for Fraud transaction wrt Gender",fontsize=20)
sns.barplot(x='insured_sex',y='total_claim_amount',hue='fraud_reported',data=df)
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.show()
```



Both Male and female have caimed same amount which are fraud

In [31]:
```
a['insured_relationship'].value_counts()
```

Out[31]:
```
other-relative    52
not-in-family     45
wife              42
own-child         39
husband           35
unmarried         34
Name: insured_relationship, dtype: int64
```

In [32]:
```
# calculating profit based on capital gains and capital loss for the insurance company
profit=df['capital-gains']-df['capital-loss']
df1=df
df1['profit']=profit
```

In [33]:
```
df1.columns
```

Out[33]:
```
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', 'profit'],
      dtype='object')
```

In [34]:
```
df[['policy_number','profit']].sort_values('profit',ascending=False)[0:20]
```

Out[34]:

	policy_number	profit
807	250833	192000
533	840806	164100
59	485372	153300
679	774303	151100
353	958785	150600
523	190588	149400
613	831053	148000
846	545506	142500
507	925128	142300
598	507545	141600
916	727443	141200
22	285496	140900
305	771236	139500
517	243226	139000
974	291006	138000
83	960680	137900
431	497347	136500
426	131478	136300
970	844129	136200
790	607259	136000

The policy number 250833 gives us the highest profit or low claim.

In [35]:
```
df[['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']]
```

Out[35]:

	incident_date	incident_type	collision_type	incident_severity	authorities_contacted	incident_state	incident_city	incident_location	incident_h
0	2015-01-25	Single Vehicle Collision	Side Collision	Major Damage	Police	SC	Columbus	9935 4th Drive	
1	2015-01-21	Vehicle Theft	?	Minor Damage	Police	VA	Riverwood	6608 MLK Hwy	
2	2015-02-22	Multi-vehicle Collision	Rear Collision	Minor Damage	Police	NY	Columbus	7121 Francis Lane	
3	2015-01-10	Single Vehicle Collision	Front Collision	Major Damage	Police	OH	Arlington	6956 Maple Drive	
4	2015-02-17	Vehicle Theft	?	Minor Damage	None	NY	Arlington	3041 3rd Ave	
...	...	...	...	...	...	...	...	...	
995	2015-02-22	Single Vehicle Collision	Front Collision	Minor Damage	Fire	NC	Northbrook	6045 Andromedia St	
996	2015-01-24	Single Vehicle Collision	Rear Collision	Major Damage	Fire	SC	Northbend	3092 Texas Drive	
997	2015-01-23	Multi-vehicle Collision	Side Collision	Minor Damage	Police	NC	Arlington	7629 5th St	
998	2015-02-26	Single Vehicle Collision	Rear Collision	Major Damage	Other	NY	Arlington	6128 Elm Lane	
999	2015-02-26	Parked Car	?	Minor Damage	Police	WV	Columbus	1416 Cherokee Ridge	

1000 rows × 10 columns

In [36]:
```
pd.pivot_table(a,values=['number_of_vehicles_involved','total_claim_amount','vehicle_claim','incident_hour_of_the_day
```

Out[36]:

		incident_hour_of_the_day	number_of_vehicles_involved	total_claim_amount	vehicle_claim
incident_type	**collision_type**				
Single Vehicle Collision	**Front Collision**	11.595238	1.000000	66596.190476	49030.000000
	Side Collision	11.787879	1.000000	68009.696970	48481.212121
	Rear Collision	11.071429	1.000000	65782.857143	47616.666667
Multi-vehicle Collision	**Side Collision**	11.432432	2.972973	62281.621622	45308.648649
	Front Collision	11.892857	2.964286	60970.000000	43860.000000
	Rear Collision	14.142857	3.081633	61152.448980	43474.693878
Parked Car	**?**	7.000000	1.000000	5093.750000	3711.250000
Vehicle Theft	**?**	5.375000	1.000000	5197.500000	3665.000000

For auto claims, single vehicle side collision have claimed highest

In [37]:
```
df['number_of_vehicles_involved'].nunique()
```

Out[37]:
```
4
```

In [38]:
```
df['number_of_vehicles_involved'].value_counts()
```

Out[38]:
```
1    581
3    358
4     31
2     30
Name: number_of_vehicles_involved, dtype: int64
```

In [39]:
```
df['incident_type'].value_counts()
```

Out[39]:
```
Multi-vehicle Collision     419
Single Vehicle Collision    403
Vehicle Theft                94
Parked Car                   84
Name: incident_type, dtype: int64
```

In [40]:
```
df['collision_type'].value_counts()
```

Out[40]:
```
Rear Collision     292
Side Collision     276
Front Collision    254
?                  178
Name: collision_type, dtype: int64
```

In [41]:
```
# For collision type we have few ? values, which are nan values and has to be replaced/removed
#Let us see for what kind of incident we have for value ?
coll=a[['incident_type','collision_type']]
coll
```

Out[41]:

	incident_type	collision_type
0	Single Vehicle Collision	Side Collision
1	Vehicle Theft	?
3	Single Vehicle Collision	Front Collision
5	Multi-vehicle Collision	Rear Collision
14	Single Vehicle Collision	Rear Collision
...	...	...
974	Multi-vehicle Collision	Side Collision
977	Multi-vehicle Collision	Side Collision
982	Multi-vehicle Collision	Front Collision
986	Single Vehicle Collision	Rear Collision
987	Single Vehicle Collision	Side Collision

247 rows × 2 columns

In [42]:
```
res=coll.loc[coll['collision_type']=='?']
res
```

Out[42]:

	incident_type	collision_type
1	Vehicle Theft	?
27	Vehicle Theft	?
196	Vehicle Theft	?
281	Vehicle Theft	?
364	Parked Car	?
365	Parked Car	?
373	Parked Car	?
437	Parked Car	?
474	Parked Car	?
478	Vehicle Theft	?
538	Parked Car	?
552	Vehicle Theft	?
597	Parked Car	?
635	Parked Car	?
837	Vehicle Theft	?
964	Vehicle Theft	?

In [43]:
```
res['incident_type'].value_counts()
```

Out[43]:
```
Vehicle Theft    8
Parked Car       8
Name: incident_type, dtype: int64
```

8 Cars which are parked and 8 cars which are theft have claimed fraud.

In [44]:
```
coll_df=df[['incident_type','collision_type']]
```

In [45]:
```
res_df=coll_df.loc[coll_df['collision_type']=='?']
```

In [46]:
```
res_df['incident_type'].value_counts()
```

Out[46]:
```
Vehicle Theft    94
Parked Car       84
Name: incident_type, dtype: int64
```

Cars which are theft and Parked are marked as ?, we can replace them wiht either NA or No collisison

In [47]:
```
# Replace it with NA as some other values could bring the biasness.
df['collision_type']=df['collision_type'].replace("?","Not Applicable")
```

In [48]:
```
df['collision_type'].value_counts()
```

Out[48]:
```
Rear Collision     292
Side Collision     276
Front Collision    254
Not Applicable     178
Name: collision_type, dtype: int64
```

In [49]:
```
# Now, we are trying to see in which city the incidents are more
a['incident_city'].value_counts()
```

Out[49]:
```
Arlington      44
Columbus       39
Springfield    38
Hillsdale      35
Northbend      34
Riverwood      30
Northbrook     27
Name: incident_city, dtype: int64
```

People from Arlington have more auto related incidents which are claimed to be fraud

In [50]:
```
pd.pivot_table(a,values=['total_claim_amount','vehicle_claim'],index=['incident_state','incident_city']).sort_values(
```

Out[50]:

		total_claim_amount	vehicle_claim
incident_state	**incident_city**		
SC	**Riverwood**	78980.000000	56553.333333
OH	**Columbus**	78100.000000	54670.000000
NC	**Northbrook**	76653.333333	55346.666667
WV	**Northbrook**	75205.000000	53997.500000
SC	**Springfield**	73116.666667	50267.500000
NY	**Northbrook**	69730.000000	50758.571429
NC	**Springfield**	69270.000000	51032.000000
NY	**Springfield**	67157.000000	47976.000000
OH	**Northbrook**	66550.000000	46585.000000
PA	**Hillsdale**	66480.000000	49860.000000
WV	**Arlington**	66077.142857	45911.428571
	Hillsdale	65931.666667	46198.333333
	Northbend	65786.000000	49410.000000
NY	**Columbus**	65067.777778	47463.333333
SC	**Northbend**	64135.555556	46540.000000
NY	**Hillsdale**	64012.857143	47301.428571
	Northbend	63797.500000	45491.250000
SC	**Arlington**	63037.500000	44675.000000
VA	**Springfield**	61665.000000	47475.000000
	Columbus	61635.000000	45662.500000

Riverwood city from SC state have claimed maximum amount of fraud for auto insurance

In [51]:
```
df.columns
```

Out[51]:
```
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', 'profit'],
      dtype='object')
```

In [52]:
```
# Trying to see the the vehicle claim data.
a.loc[(a['property_claim'] == 0.0 )&(a['vehicle_claim'] != 0.0 )]
```

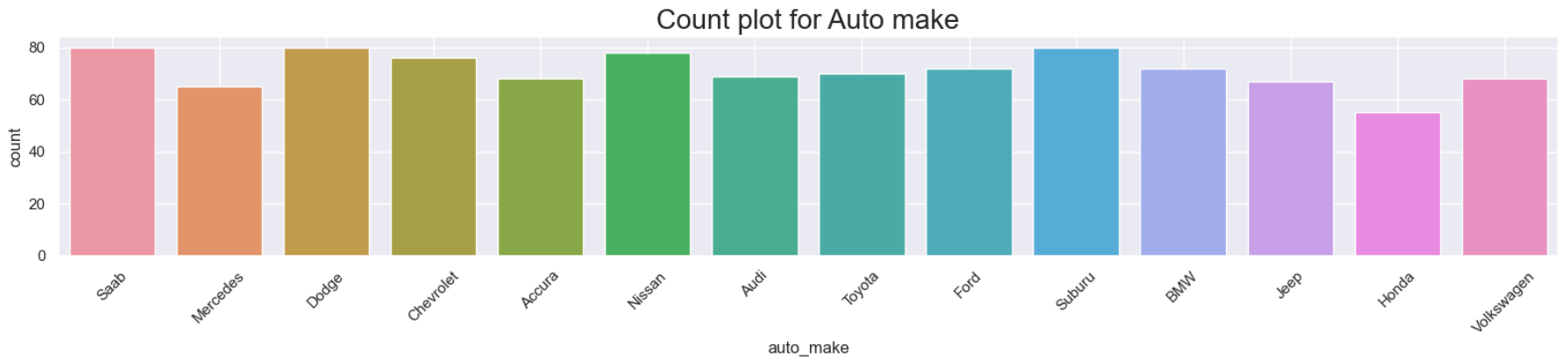
Out[52]:

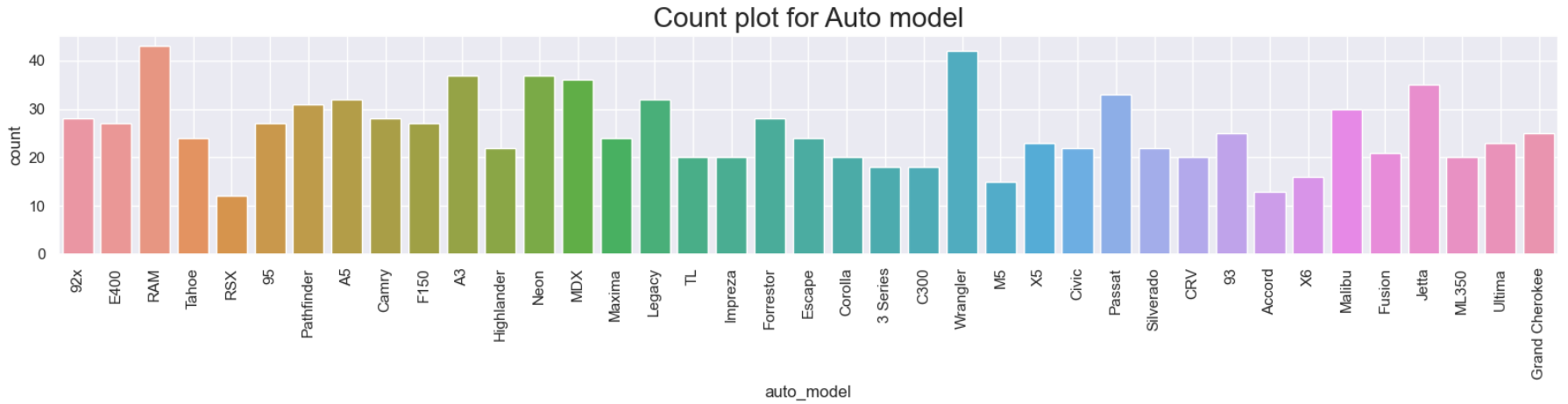
	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_lin
60	154	34	598554	1990-02-14	IN	100/300	500	795.23	
155	375	50	120485	2007-02-18	OH	100/300	1000	1275.39	
705	274	45	589094	2003-05-27	IN	250/500	1000	1353.53	
803	123	29	379268	2012-08-05	IN	250/500	500	1209.63	
843	297	48	264221	2014-07-28	IL	500/1000	1000	1243.68	
938	147	31	746630	1997-02-10	IN	250/500	500	1054.92	60000

6 rows × 39 columns

In [53]:
```
plt.figure(figsize=(16,8))
plt.subplot(2,1,1)
plt.title("Count plot for Auto make",fontsize=20)
sns.countplot('auto_make',data=df)
plt.xticks(rotation=45)
plt.grid(True)
plt.tight_layout()
plt.subplot(2,1,2)
plt.title("Count plot for Auto model",fontsize=20)
sns.countplot('auto_model',data=df)
plt.xticks(rotation=90)
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```


Count plot for Auto make
count
0
20
40
60
80
Saab
Mercedes
Dodge
Chevrolet
Accura
Nissan
Audi
Toyota
Ford
Suburu
BMW
Jeep
Honda
Volkswagen
auto_make

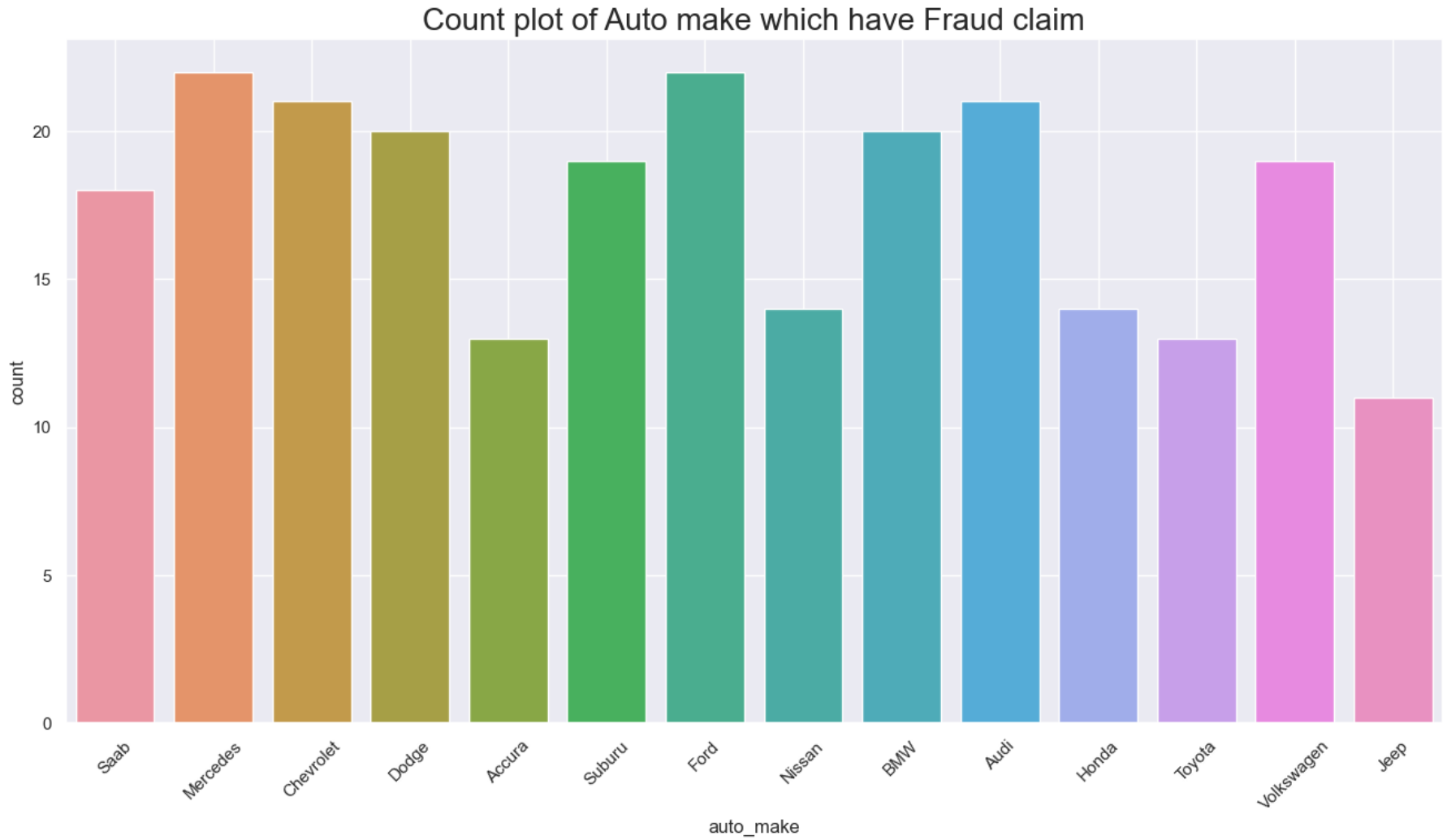

Count plot for Auto model
count
0
10
20
30
40
92x
E400
RAM
Tahoe
RSX
95
Pathfinder
A5
Camry
F150
A3
Highlander
Neon
MDX
Maxima
Legacy
TL
Impreza
Forrestor
Escape
Corolla
3 Series
C300
Wrangler
M5
X5
Civic
Passat
Silverado
CRV
93
Accord
X6
Malibu
Fusion
Jetta
ML350
Ultima
Grand Cherokee
auto_model

In [54]:
```
a['auto_make'].value_counts()
```

Out[54]:
```
Mercedes      22
Ford          22
Chevrolet     21
Audi          21
Dodge         20
BMW           20
Suburu        19
Volkswagen    19
Saab          18
Nissan        14
Honda         14
Accura        13
Toyota        13
Jeep          11
Name: auto_make, dtype: int64
```

In [55]:
```
plt.figure(figsize=(16,8))
plt.title("Count plot of Auto make which have Fraud claim",fontsize=20)
sns.countplot('auto_make',data=a)
plt.xticks(rotation=45)
plt.grid(True)
plt.show()
```

```
C:\Users\angsh\anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as
a keyword arg: x. From version 0.12, the only valid positional argument will be `data`, and passing other arguments
without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```



Auto make of 'Ford' and 'Mercedes' are having highest Fraud claim, 'Chevorlet' and 'Audi' also seems to be claimed equally having Fraud claims

In [56]:
```
# Vehicle claim as per auto make and model and policy number wise. As a is the dataset here which have false reported
pd.pivot_table(a,values=['vehicle_claim'],index=['auto_model','auto_make','policy_number']).sort_values('vehicle_clai
```

Out[56]:

			vehicle_claim
auto_model	**auto_make**	**policy_number**	
Impreza	**Suburu**	**217938**	77760
TL	**Accura**	**515050**	76400
Tahoe	**Chevrolet**	**291006**	75600
Neon	**Dodge**	**346940**	75530
RAM	**Dodge**	**115399**	73620
Accord	**Honda**	**127754**	71520
Highlander	**Toyota**	**209177**	69210
Tahoe	**Chevrolet**	**187775**	68310
Escape	**Ford**	**626208**	67590
E400	**Mercedes**	**728025**	67440

Policy 217938 who have Impreza-Suburu has claimed amount of 77,760$ and is identified as Fraud claim

Data Preprocessing

In [57]:
```
df.columns
```

Out[57]:
```
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', 'profit'],
      dtype='object')
```

In [58]:
```
a['bodily_injuries'].value_counts()
```

Out[58]:
```
2    90
0    80
1    77
Name: bodily_injuries, dtype: int64
```

In [59]:
```
a['police_report_available'].value_counts()
```

Out[59]:
```
?      89
NO     86
YES    72
Name: police_report_available, dtype: int64
```

Even for the policies which have police report have done fraud claims

In [60]:
```
# Here, we again found ? values in police report, which need to be replace.
df['police_report_available']=df['police_report_available'].replace("?","Unknown")
```

In [61]:
```
df.columns
```

Out[61]:
```
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', 'profit'],
      dtype='object')
```

In [62]:
```
df.umbrella_limit.unique()
```

Out[62]:
```
array([       0,  5000000,  6000000,  4000000,  3000000,  8000000,
        7000000,  9000000, 10000000, -1000000,  2000000], dtype=int64)
```

In [63]:
```
# Umbrella limit can't be in negetive. So, we need to replace it with 0.
df['umbrella_limit']=df['umbrella_limit'].replace(-1000000,0)
```

In [64]:
```
df2=df
df2
```

Out[64]:

	months_as_customer	age	policy_number	policy_bind_date	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_lin
0	328	48	521585	2014-10-17	OH	250/500	1000	1406.91	
1	228	42	342868	2006-06-27	IN	250/500	2000	1197.22	50000
2	134	29	687698	2000-09-06	OH	100/300	2000	1413.14	50000
3	256	41	227811	1990-05-25	IL	250/500	2000	1415.74	60000
4	228	44	367455	2014-06-06	IL	500/1000	1000	1583.91	60000
...	...	...	...	...	...	...	...	...	
995	3	38	941851	1991-07-16	OH	500/1000	1000	1310.80	
996	285	41	186934	2014-01-05	IL	100/300	1000	1436.79	
997	130	34	918516	2003-02-17	OH	250/500	500	1383.49	30000
998	458	62	533940	2011-11-18	IL	500/1000	2000	1356.92	50000
999	456	60	556080	1996-11-11	OH	250/500	1000	766.19	

1000 rows × 40 columns

In [65]:
```
# Dropping columns as they they are not that important to find out fraud detect pattern.
df2=df2.drop(['policy_number','policy_bind_date','insured_zip','incident_date','authorities_contacted','profit','auto
```

In [66]:
```
df2.columns
```

Out[66]:
```
Index(['months_as_customer', 'age', 'policy_state', 'policy_csl',
       'policy_deductable', 'policy_annual_premium', 'umbrella_limit',
       'insured_sex', 'insured_education_level', 'insured_occupation',
       'insured_hobbies', 'insured_relationship', 'capital-gains',
       'capital-loss', 'incident_type', 'collision_type', 'incident_severity',
       '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_year', 'fraud_reported'],
      dtype='object')
```

In [67]:
```
df2=pd.get_dummies(df2,columns=['policy_state','policy_csl','insured_sex','insured_education_level','insured_occupati
                                'insured_hobbies','insured_relationship','incident_type','collision_type','incident_s
                                'incident_city','incident_location','property_damage','police_report_available'],drop
```

First dummy variable for each category is dropped to avoid multicollinearity in regression models.

In [68]:
```
df2.shape
```

Out[68]:
```
(1000, 1089)
```

In [69]:
```
df2.columns
```

Out[69]:
```
Index(['months_as_customer', 'age', 'policy_deductable',
       'policy_annual_premium', 'umbrella_limit', 'capital-gains',
       'capital-loss', 'incident_hour_of_the_day',
       'number_of_vehicles_involved', 'bodily_injuries',
       ...
       'incident_location_9918 Andromedia Drive',
       'incident_location_9929 Rock Drive', 'incident_location_9935 4th Drive',
       'incident_location_9942 Tree Ave', 'incident_location_9980 Lincoln Ave',
       'incident_location_9988 Rock Ridge', 'property_damage_NO',
       'property_damage_YES', 'police_report_available_Unknown',
       'police_report_available_YES'],
      dtype='object', length=1089)
```

In [70]:
```
df2['fraud_reported'].value_counts()
```

Out[70]:
```
N    753
Y    247
Name: fraud_reported, dtype: int64
```

From here we can understand it is an imbalance dataset as the target features have imbalance data.

In [71]:
```
df.isnull().sum()
```

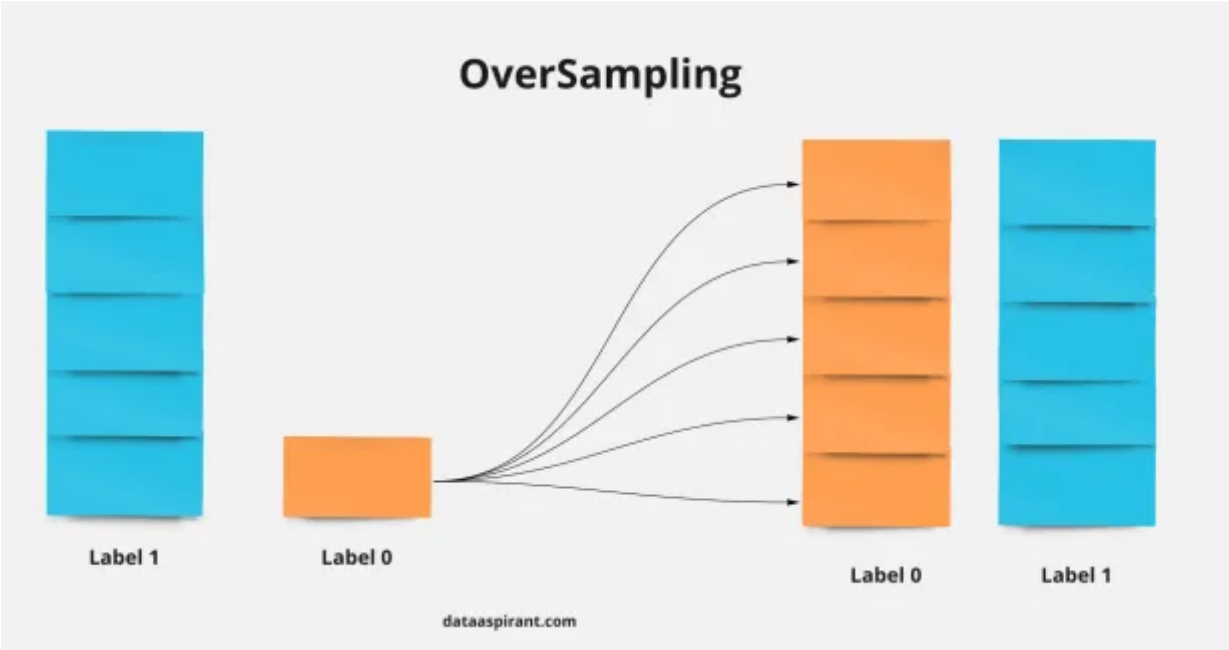
Out[71]:

```
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
profit                         0
dtype: int64
```

In [72]:
```
x=df2.drop(['fraud_reported'],axis=1)
```

In [73]:
```
y=df2['fraud_reported']
```

Up Sampling/Over sampling of Data



SMOTE Oversampling

In [74]:
```
!pip install imbalanced-learn
```

```
Requirement already satisfied: imbalanced-learn in c:\users\angsh\anaconda3\lib\site-packages (0.10.1)
Requirement already satisfied: joblib>=1.1.1 in c:\users\angsh\anaconda3\lib\site-packages (from imbalanced-learn)
(1.2.0)
Requirement already satisfied: scipy>=1.3.2 in c:\users\angsh\anaconda3\lib\site-packages (from imbalanced-learn)
(1.9.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\angsh\anaconda3\lib\site-packages (from imbalanced-l
earn) (2.2.0)
Requirement already satisfied: scikit-learn>=1.0.2 in c:\users\angsh\anaconda3\lib\site-packages (from imbalanced-le
arn) (1.0.2)
Requirement already satisfied: numpy>=1.17.3 in c:\users\angsh\anaconda3\lib\site-packages (from imbalanced-learn)
(1.21.5)
```

In [75]:
```
from imblearn.over_sampling import SMOTE

x_upsample, y_upsample  = SMOTE().fit_resample(x, y)

print(x_upsample.shape)
print(y_upsample.shape)
```

```
(1506, 1088)
(1506,)
```

In [76]:
```
y_upsample.value_counts()
```

Out[76]:
```
Y    753
N    753
Name: fraud_reported, dtype: int64
```

By sampling technique now we made the target variable as balance dataset.

Machine Learning Modelling

In [77]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_upsample, y_upsample, test_size=0.2, random_state=42)
```

In [78]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
#from sklearn.preprocessing import StandardScaler
#sc=StandardScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train),columns=list(X_train.columns))
X_test = pd.DataFrame(scaler.transform(X_test),columns=list(X_test.columns))
```

In [79]:
```
from sklearn.linear_model import LogisticRegression

# Create an instance of the LogisticRegression model
logreg = LogisticRegression()

# Fit the model on the training data
logreg.fit(X_train, y_train)

train_accuracy = logreg.score(X_train, y_train)
print("Training Accuracy:", train_accuracy)
```

```
Training Accuracy: 0.9526578073089701
```

In [80]:
```
from sklearn.metrics import classification_report

# Predict the labels for the training data
y_pred_train = logreg.predict(X_train)

# Generate the classification report
classification_rep_train = classification_report(y_train, y_pred_train)

# Print the classification report
print("Classification Report (Training Data):")
print(classification_rep_train)
```

```
Classification Report (Training Data):
              precision    recall  f1-score   support

           N       0.95      0.96      0.95       611
           Y       0.96      0.95      0.95       593

    accuracy                           0.95      1204
   macro avg       0.95      0.95      0.95      1204
weighted avg       0.95      0.95      0.95      1204
```

In [81]:
```
# Predict the target variable for the testing data
y_pred = logreg.predict(X_test)

# Evaluate the model's performance
test_accuracy = logreg.score(X_test, y_test)
print("Testing Accuracy:", test_accuracy)
```

```
Testing Accuracy: 0.8543046357615894
```

In [82]:
```
clf=classification_report(y_pred,y_test)
print("Classification Report (Testing Data):")
print(clf)
```

```
Classification Report (Testing Data):
              precision    recall  f1-score   support

           N       0.88      0.82      0.85       152
           Y       0.83      0.89      0.86       150

    accuracy                           0.85       302
   macro avg       0.86      0.85      0.85       302
weighted avg       0.86      0.85      0.85       302
```

In [83]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x_upsample, y_upsample, test_size=0.2)
```

In [84]:
```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train = pd.DataFrame(scaler.fit_transform(X_train),columns=list(X_train.columns))
X_test = pd.DataFrame(scaler.transform(X_test),columns=list(X_test.columns))
```

In [85]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV

# Create a Random Forest classifier
rf_classifier = RandomForestClassifier()

# Define the hyperparameters and their possible values
param_grid = {
    'n_estimators': [50, 100, 200],  # Number of trees in the forest
    'max_depth': [3, 3, 5],  # Maximum depth of each tree
    'min_samples_split': [2, 5, 10],  # Minimum number of samples required to split an internal node
    'min_samples_leaf': [1, 2, 4],  # Minimum number of samples required to be at a leaf node
    'max_features': ['auto', 'sqrt']  # Number of features to consider when looking for the best split
}

# Perform grid search with cross-validation
grid_search = GridSearchCV(estimator=rf_classifier, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Get the best hyperparameters and the corresponding model
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Calculate the training score of the best model
training_score = best_model.score(X_train, y_train)

# Print the best hyperparameters and the training score
print("Best Hyperparameters:", best_params)
print("Training Score:", training_score)
```

```
Best Hyperparameters: {'max_depth': 5, 'max_features': 'auto', 'min_samples_leaf': 4, 'min_samples_split': 2, 'n_est
imators': 100}
Training Score: 0.8945182724252492
```

In [86]:
```
from sklearn.metrics import classification_report

# Predict the labels for the training data
y_pred_train = grid_search.predict(X_train)

# Generate the classification report
classification_rep = classification_report(y_train, y_pred_train)

# Print the classification report
print("Classification Report(Training Data):")
print(classification_rep)
```

```
Classification Report(Training Data):
              precision    recall  f1-score   support

           N       0.91      0.88      0.89       594
           Y       0.88      0.91      0.90       610

    accuracy                           0.89      1204
   macro avg       0.90      0.89      0.89      1204
weighted avg       0.89      0.89      0.89      1204
```

In [87]:
```
# Calculate the test score
test_score = grid_search.score(X_test, y_test)

# Print the test score
print("Test Score:", test_score)
```

```
Test Score: 0.8112582781456954
```

In [88]:
```
from sklearn.metrics import classification_report

# Predict the labels for the test data
y_pred_test = grid_search.predict(X_test)

# Generate the classification report
classification_rep_test = classification_report(y_test, y_pred_test)

# Print the classification report
print("Classification Report (Test Data):")
print(classification_rep_test)
```

```
Classification Report (Test Data):
              precision    recall  f1-score   support

           N       0.83      0.81      0.82       159
           Y       0.79      0.82      0.80       143

    accuracy                           0.81       302
   macro avg       0.81      0.81      0.81       302
weighted avg       0.81      0.81      0.81       302
```

Conclusion

So, By comparing those above two models, it is found that Random forest is performing better than the Logistic Regression model.

After doing the hyperparameter tuning with n_estimators, max_depth, min_samples_split, min_samples_leaf and max_features and With Grid search technique to search for the best combination of hyperparameters for a machine learning model, The training score is nearly at 89 percent and Test score is at 81 percent.

And the recall is .90 for yes class in training and .82 for yes class in test data which signifies a good model to predict the insurance fraud based on various factors.

But from above both the models we will select Random Forest model as it predicts slightly better than Logistic regression and it decrease the model overfitting by increasing the generalization power of the model as Random forest is a bagging algo which is build

In [ ]: