PROJECT : Insurance Claims- Fraud Detection

In [36]:

```
#Importing the libreris like pandas, numpy for selecing the data and convering the data to
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [37]:

```
#Creating the variable df and loading the dataset to the variable df
df=pd.read_csv('data')
df
```

Out[37]:

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

1000 rows × 40 columns

In [4]:

```
# let's get the information about the dataset

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
         elationship              1000 non-null   object
```

Loading [MathJax]/extensions/Safe.js

```
 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 [38]:

```
# let's check whether the data has any null values or not.

# but there is '?' in the datset which we have to replace by NaN Values
df = df.replace('?',np.NaN)

df.isnull().any()
```

Out[38]:

```
months_as_customer             False
age                            False
policy_number                  False
policy_bind_date               False
policy_state                   False
policy_csl                     False
policy_deductable              False
policy_annual_premium          False
umbrella_limit                 False
insured_zip                    False
insured_sex                    False
insured_education_level        False
insured_occupation             False
insured_hobbies                False
insured_relationship           False
capital-gains                  False
capital-loss                   False
incident_date                  False
incident_type                  False
collision_type                  True
incident_severity              False
authorities_contacted          False
incident_state                 False
incident_city                  False
incident_location              False
incident_hour_of_the_day       False
number_of_vehicles_involved    False
property_damage                 True
                          s    False
```

```
witnesses                     False
police_report_available        True
total_claim_amount            False
injury_claim                  False
property_claim                False
vehicle_claim                 False
auto_make                     False
auto_model                    False
auto_year                     False
fraud_reported                False
_c39                           True
dtype: bool
```

In [39]:

```python
# filling the null values

# we will replace the '?' by the most common collision type as we are unaware of the type.
df['collision_type'].fillna(df['collision_type'].mode()[0], inplace = True)

# It may be the case that there are no responses for property damage then we might take it
df['property_damage'].fillna('NO', inplace = True)

# again, if there are no responses fpr police report available then we might take it as No
df['police_report_available'].fillna('NO', inplace = True)
```

In [40]:

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

Out[40]:

```
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
                               0
```

```
auto_year                    0
fraud_reported               0
_c39                      1000
dtype: int64
```

In [41]:

```
#droping the _c39 column, _c39 column doesnot have any impact on target column
df=df.drop(columns='_c39')
df
```

Out[41]:

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

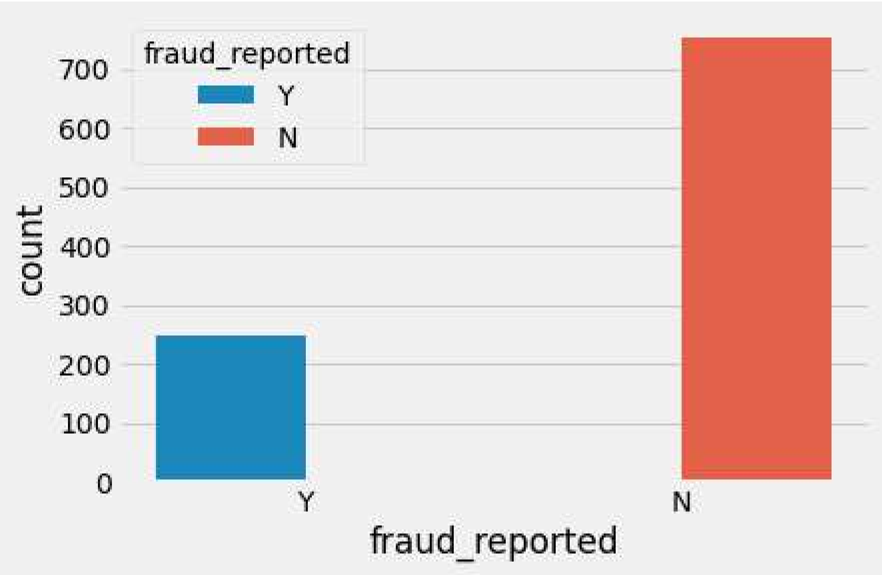
1000 rows × 39 columns

In [9]:

```
import matplotlib.pyplot as plt
import seaborn as sns
plt.style.use('fivethirtyeight')
ax = sns.countplot(x='fraud_reported', data=df, hue='fraud_reported')
df['fraud_reported'].value_counts()
```

Out[9]:

```
N    753
Y    247
Name: fraud_reported, dtype: int64
```



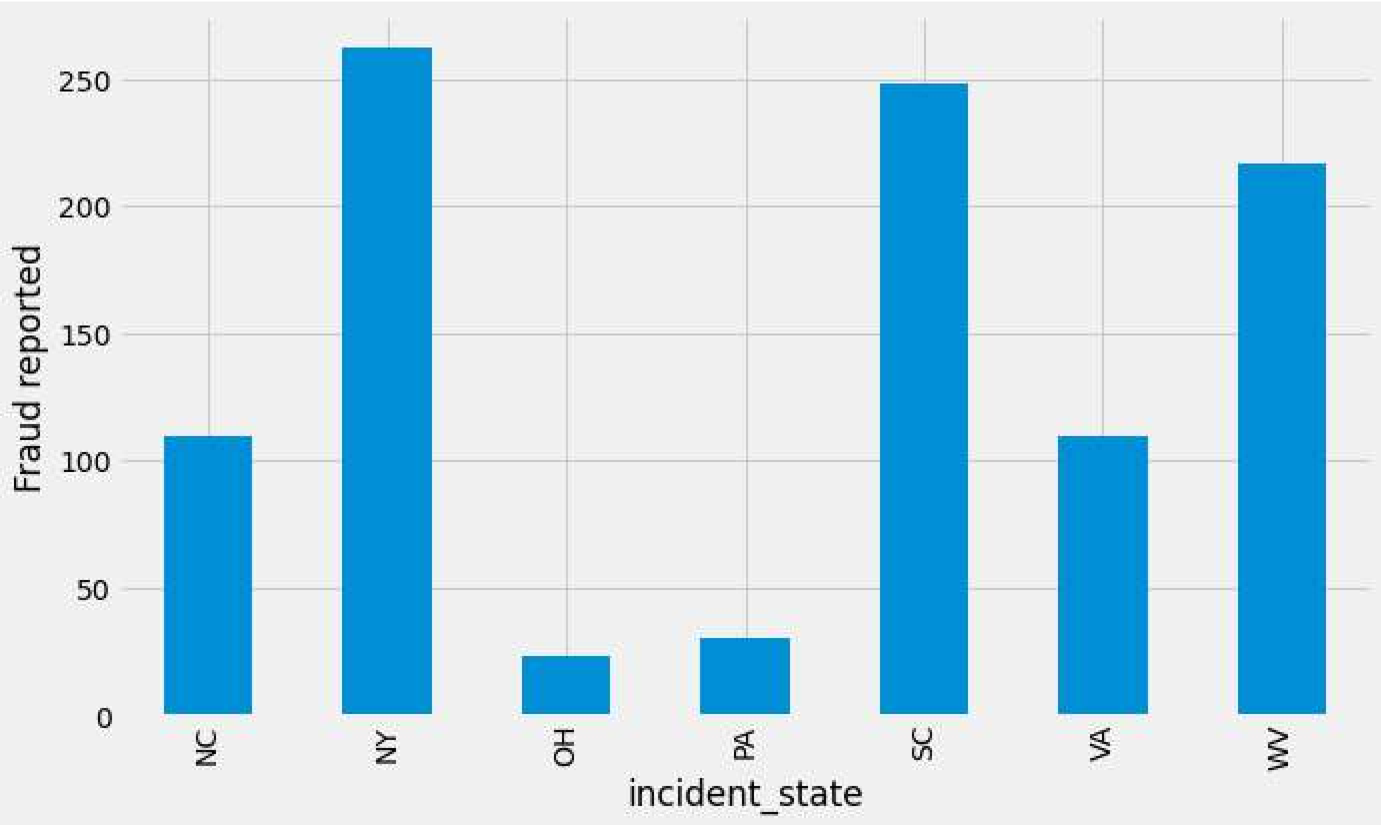
In [10]:

```
print(df['incident_state'].value_counts())
plt.style.use('fivethirtyeight')
...ure(figsize=(10,6))
```

Loading [MathJax]/extensions/Safe.js

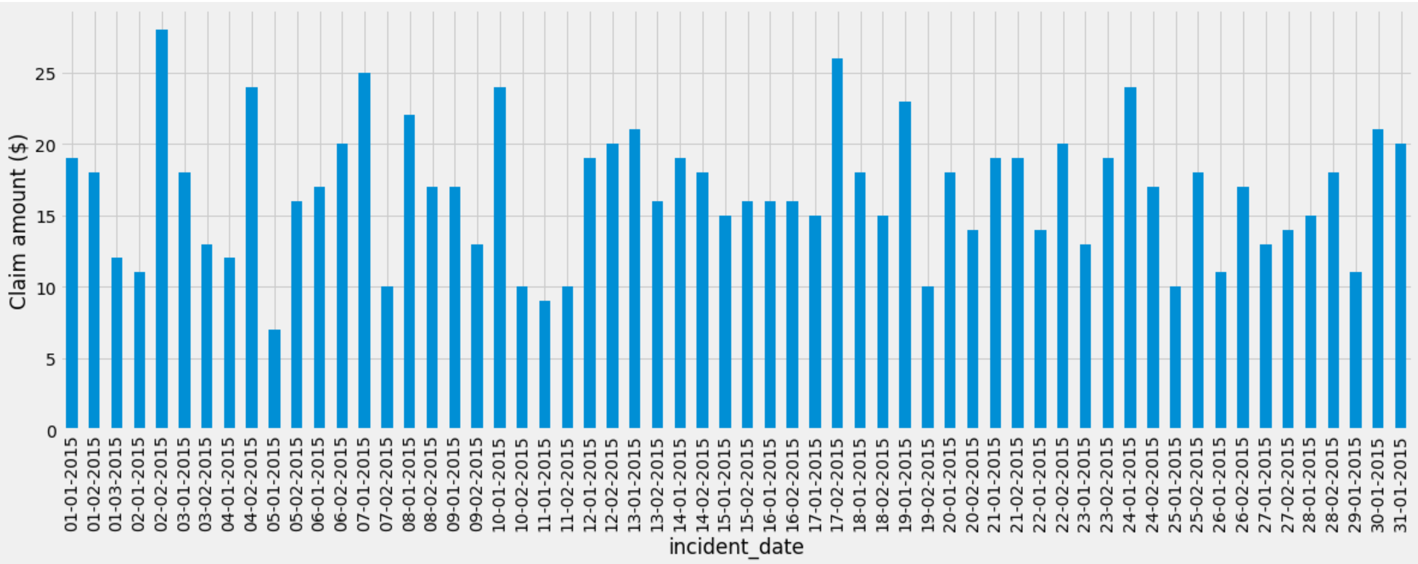
```
ax = df.groupby('incident_state').fraud_reported.count().plot.bar(ylim=0)
ax.set_ylabel('Fraud reported')
plt.show()
```

```
NY    262
SC    248
WV    217
VA    110
NC    110
PA     30
OH     23
Name: incident_state, dtype: int64
```



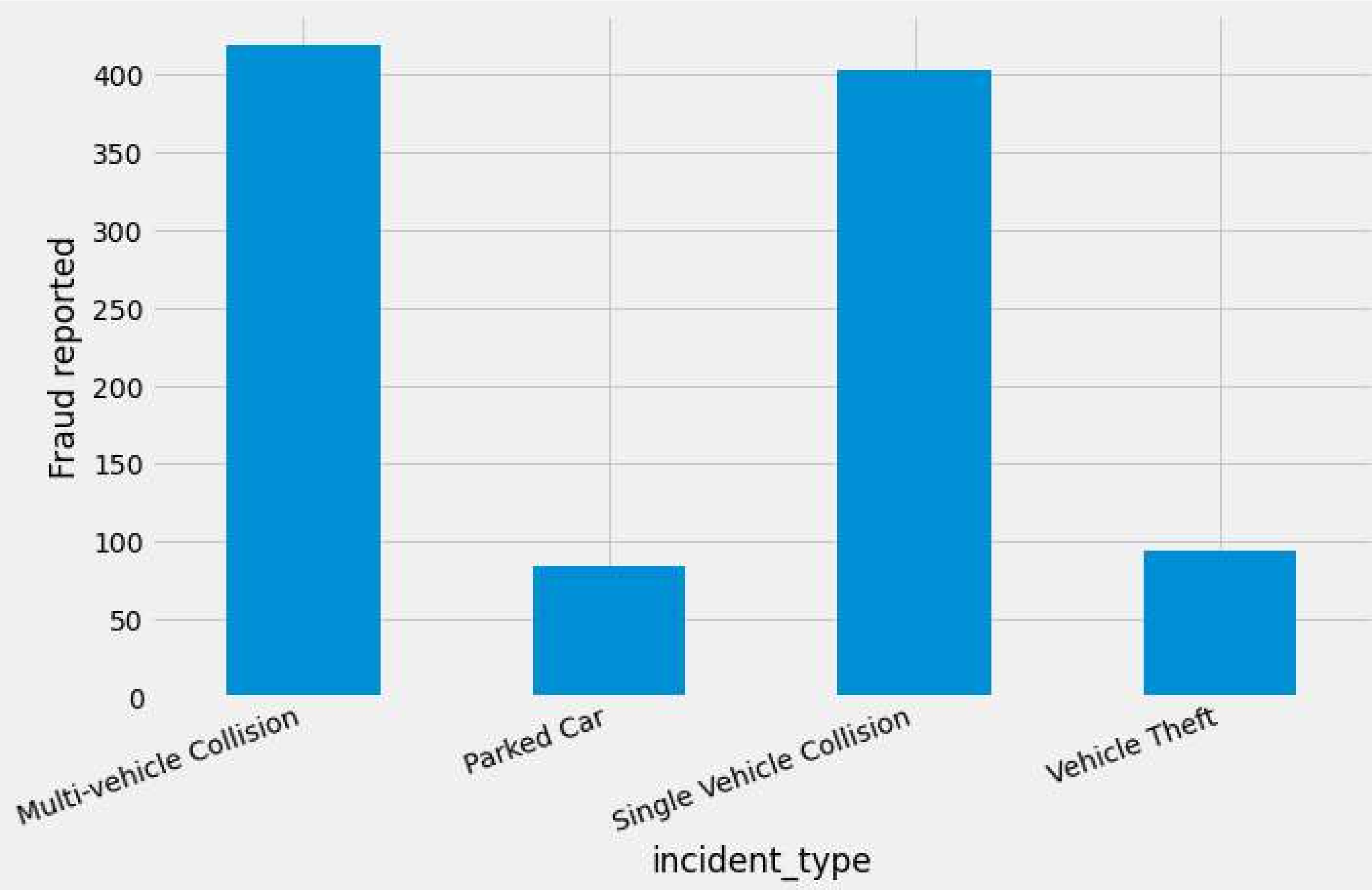
In [11]:

```
plt.style.use('fivethirtyeight')
fig = plt.figure(figsize=(18,6))
ax = df.groupby('incident_date').total_claim_amount.count().plot.bar(ylim=0)
ax.set_ylabel('Claim amount ($)')
plt.show()
```



In [12]:

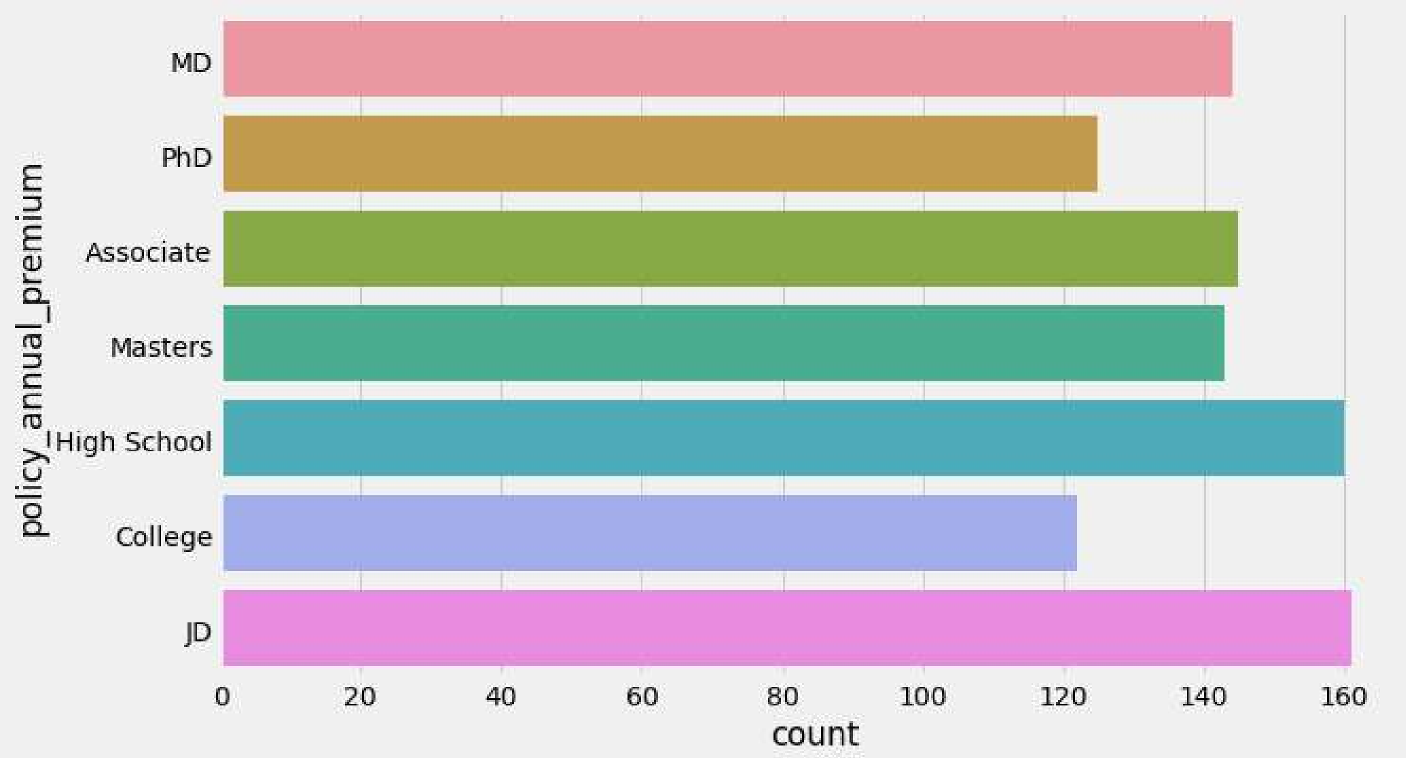
```
plt.style.use('fivethirtyeight')
fig = plt.figure(figsize=(10,6))
ax = df.groupby('incident_type').fraud_reported.count().plot.bar(ylim=0)
ax.set_xticklabels(ax.get_xticklabels(), rotation=20, ha="right")
ax.set_ylabel('Fraud reported')
plt.show()
```



In [13]:

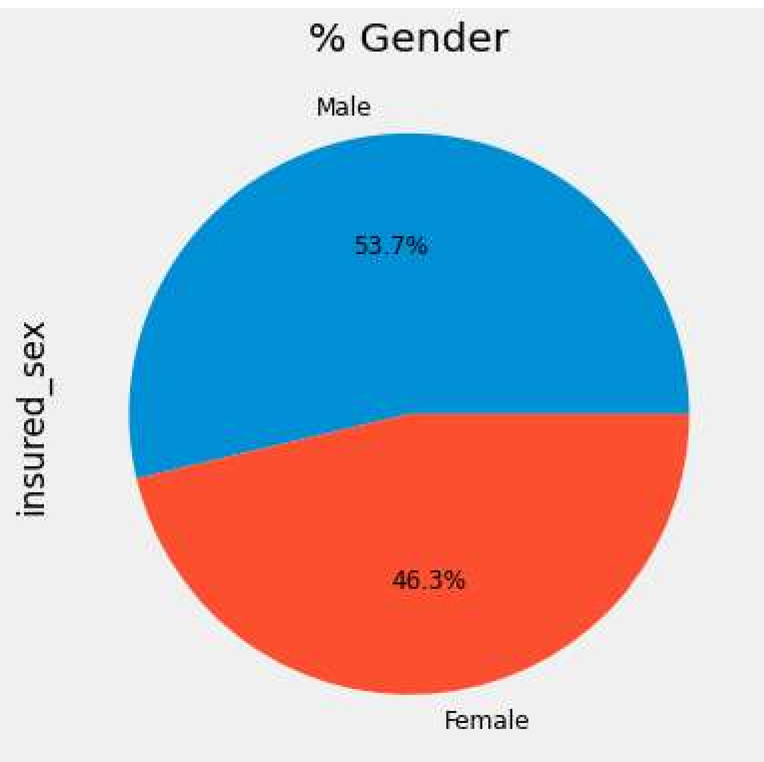
```
fig = plt.figure(figsize=(10,6))
ax = sns.countplot(y = 'insured_education_level', data=df)
ax.set_ylabel('policy_annual_premium')
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [14]:

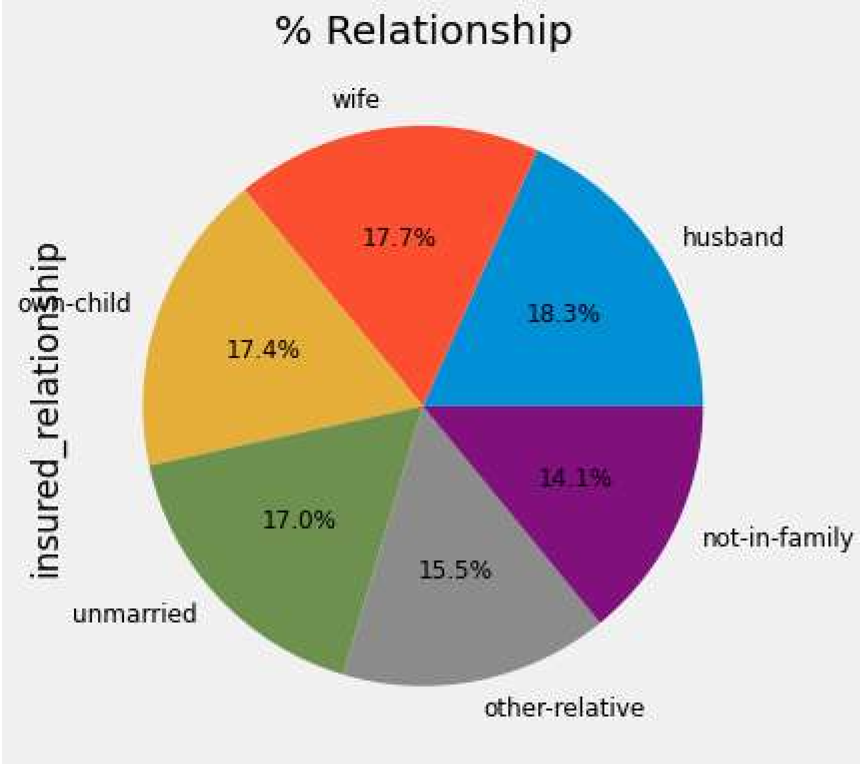
```
fig = plt.figure(figsize=(10,6))
ax = (df['insured_sex'].value_counts()*100.0 /len(df))\
.plot.pie(autopct='%.1f%%', labels = ['Male', 'Female'], fontsize=12)
ax.set_title('% Gender')
plt.show()
```



In [15]:

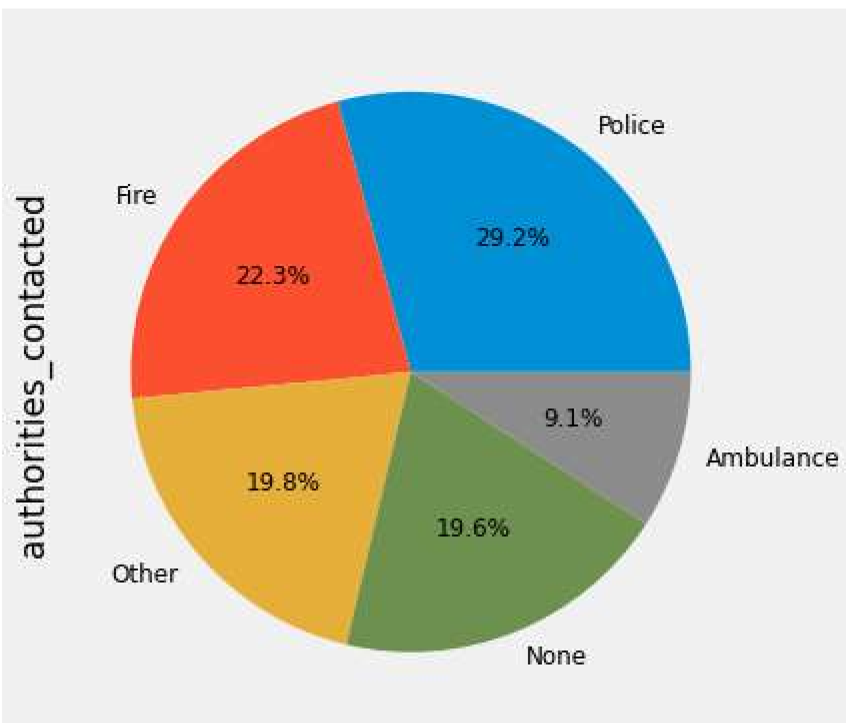
```
fig = plt.figure(figsize=(10,6))
ax = (df['insured_relationship'].value_counts()*100.0 /len(df))\
.plot.pie(autopct='%.1f%%', labels = ['husband', 'wife', 'own-child', 'unmarried', 'other-
         fontsize=12)
ax.set_title('% Relationship')
plt.show()
```

Loading [MathJax]/extensions/Safe.js



In [16]:

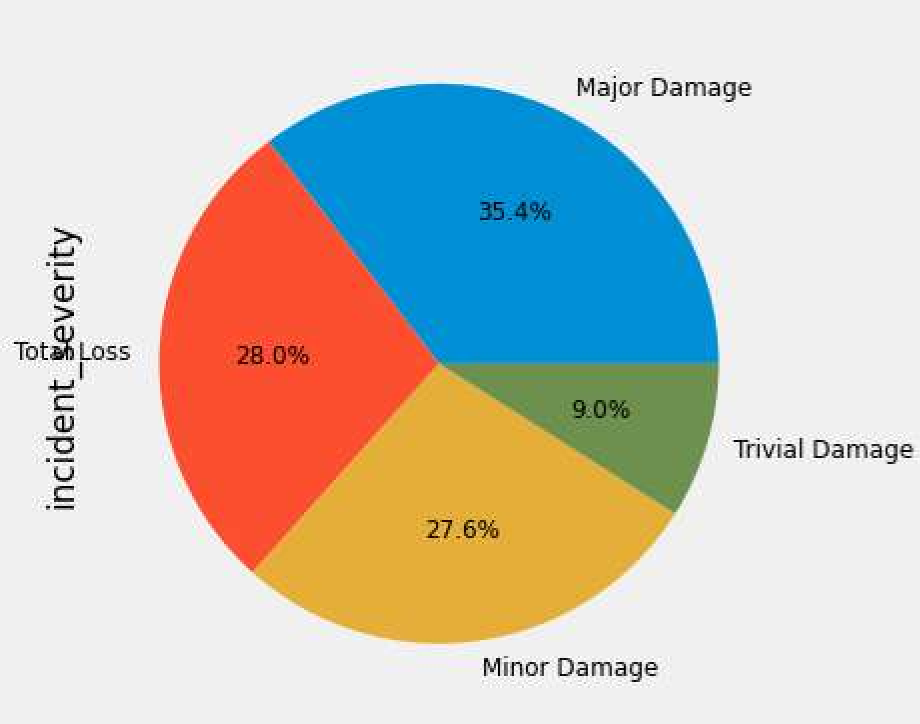
```
fig = plt.figure(figsize=(10,6))
ax = (df['authorities_contacted'].value_counts()*100.0 /len(df))\
.plot.pie(autopct='%.1f%%', labels = ['Police', 'Fire', 'Other', 'None', 'Ambulance'],
         fontsize=12)
```



In [17]:

```
fig = plt.figure(figsize=(10,6))
ax = (df['incident_severity'].value_counts()*100.0 /len(df))\
.plot.pie(autopct='%.1f%%', labels = ['Major Damage', 'Total Loss', 'Minor Damage', 'Trivi
         fontsize=12)
```

Loading [MathJax]/extensions/Safe.js



In [19]:

```
# let's extrat days, month and year from policy bind date

df['policy_bind_date'] = pd.to_datetime(df['policy_bind_date'], errors = 'coerce')
df
```

Out[19]:

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

1000 rows × 39 columns

In [42]:

```
# let's encode the fraud report to numerical values

df['fraud_reported'] = df['fraud_reported'].replace(('Y','N'),(0,1))
```

In [25]:

```
df.columns
```

Out[25]:

```
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',
       '     d_education_level', 'insured_occupation', 'insured_hobbies',
```

Loading [MathJax]/extensions/Safe.js

```
       '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 [43]:

```
# let's check the correlation of authorities_contacted with the target
# changing all the categorical column into numerical columns

df[['auto_model','fraud_reported']].groupby(['auto_model'],
                as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[43]:

	auto_model	fraud_reported
0	3 Series	0.944444
31	RSX	0.916667
25	Malibu	0.900000
36	Wrangler	0.880952
29	Pathfinder	0.870968
35	Ultima	0.869565
9	Camry	0.857143
11	Corolla	0.850000
8	CRV	0.850000
21	Legacy	0.843750
27	Neon	0.837838
3	95	0.814815
33	TL	0.800000
2	93	0.800000
23	MDX	0.777778
6	Accord	0.769231
17	Grand Cherokee	0.760000
13	Escape	0.750000
12	E400	0.740741
4	A3	0.729730
18	Highlander	0.727273
28	Passat	0.727273
1	92x	0.714286
20	Jetta	0.714286
16	Fusion	0.714286
15	Forrestor	0.714286
26	Maxima	0.708333
19	Impreza	0.700000
[illegible]	[illegible]	0.695652

Loading [MathJax]/extensions/Safe.js

	auto_model	fraud_reported
30	RAM	0.674419
22	M5	0.666667
5	A5	0.656250
10	Civic	0.636364
14	F150	0.629630
34	Tahoe	0.625000
7	C300	0.611111
24	ML350	0.600000
32	Silverado	0.590909
38	X6	0.562500

In [113…

```
# let's perform target encoding for auto make

df['auto_model'] = df['auto_model'].replace(('3 Series','RSX','Malibu','Wrangler','Pathfir
                'Corolla','CRV','Legacy','Neon','95','TL','93','MDX','Accord','Grand Cherc
            'A3','Highlander','Passat','92x','Jetta','Fusion','Forrestor','Maxima','Imprez
                'Civic','F150','Tahoe','C300','ML350','Silverado','X6'),
                (0.95,0.91, 0.90,0.88,0.87,0.86,0.855,0.85,0.85,0.84,0.83,0.81,0.80,0.80,0
                 0.73,0.72,0.72,0.71,0.71,0.71,0.71,0.70,0.70,0.69,0.67,0.66,0.65,0.64,0.6
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, ke
y, method, tolerance)
   3360             try:
-> 3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:

C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.In
dexEngine.get_loc()

C:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.In
dexEngine.get_loc()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_it
em()

pandas\_libs\hashtable_class_helper.pxi in pandas._libs.hashtable.PyObjectHashTable.get_it
em()

KeyError: 'auto_model'

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_8104/2100930274.py in <module>
      1 # let's perform target encoding for auto make
      2
----> 3 df['auto_model'] = df['auto_model'].replace(('3 Series','RSX','Malibu','Wrangle
r','Pathfinder','Ultima','Camry',
      4                 'Corolla','CRV','Legacy','Neon','95','TL','93','MDX','Accord','Gra
nd Cherokee','Escape','E400',
      5             'A3','Highlander','Passat','92x','Jetta','Fusion','Forrestor','Maxima'
,'Impreza','X5','RAM','M5','A5',
```

Loading [MathJax]/extensions/Safe.js

```
C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\frame.py in __getitem__(self, key)
   3456             if self.columns.nlevels > 1:
   3457                 return self._getitem_multilevel(key)
-> 3458             indexer = self.columns.get_loc(key)
   3459             if is_integer(indexer):
   3460                 indexer = [indexer]

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, ke
y, method, tolerance)
   3361                 return self._engine.get_loc(casted_key)
   3362             except KeyError as err:
-> 3363                 raise KeyError(key) from err
   3364
   3365         if is_scalar(key) and isna(key) and not self.hasnans:

KeyError: 'auto_model'
```

In [51]:

```
#let's check the correlation auto make with the target

df[['auto_make','fraud_reported']].groupby(['auto_make'],
                as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[51]:

	auto_make	fraud_reported
12	0.84	0.835821
11	0.82	0.820513
10	0.81	0.814286
9	0.80	0.808824
8	0.77	0.775000
7	0.76	0.762500
6	0.75	0.750000
5	0.74	0.745455
4	0.73	0.723684
3	0.72	0.722222
2	0.71	0.720588
1	0.69	0.695035
0	0.66	0.661538

In [52]:

```
# let's perform target encoding for auto make

df['auto_make'] = df['auto_make'].replace(('Jeep','Nissan','Toyota','Accura','Saab','Subu
                                 'Dodge','Honda','Chevrolet','BMW','Volkswagen','Audi','Fo
                                              (0.84,0.82,0.81,0.80,0.77,0.76,0.75,0.74,0.7
```

In [54]:

```
# changing all the categorical column into numerical columns

df[['police_report_available','fraud_reported']].groupby(['police_report_available'],
                as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[54]:

	police_report_available	fraud_reported
1	YES	0.770701
	NO	0.744898

Loading [MathJax]/extensions/Safe.js

In [56]:

```
df['police_report_available'] = df['police_report_available'].replace(('NO','YES'),(0.77,0
```

In [57]:

```
df[['property_damage','fraud_reported']].groupby(['property_damage'],
                        as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[57]:

	property_damage	fraud_reported
0	NO	0.757880
1	YES	0.741722

In [58]:

```
df['property_damage'] = df['property_damage'].replace(('NO','YES'),(0.76,0.74))
```

In [60]:

```
df[['incident_city','fraud_reported']].groupby(['incident_city'],
                        as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[60]:

	incident_city	fraud_reported
4	Northbrook	0.778689
5	Riverwood	0.776119
3	Northbend	0.765517
6	Springfield	0.757962
2	Hillsdale	0.751773
1	Columbus	0.738255
0	Arlington	0.710526

In [63]:

```
df['incident_city'] = df['incident_city'].replace(('Northbrook','Riverwood','Northbend','S
                                            'Hillsdale','Columbus','Arlington'),(0.78,0.77,0.76,0.
```

In [64]:

```
df[['incident_state','fraud_reported']].groupby(['incident_state'],
                        as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[64]:

	incident_state	fraud_reported
6	WV	0.820276
1	NY	0.778626
5	VA	0.772727
3	PA	0.733333
4	SC	0.705645
0	NC	0.690909
2	OH	0.565217

In [65]:

```
df['incident_state'] = df['incident_state'].replace(('WV','NY','VA','PA','SC','NC','OH'),
                                                        (0.82,0.77,0.76,0.73,0.70,0.69,0.5
```

Loading [MathJax]/extensions/Safe.js

```
df[['authorities_contacted','fraud_reported']].groupby(['authorities_contacted'],
                 as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[66]:

	authorities_contacted	fraud_reported
2	None	0.934066
4	Police	0.791096
1	Fire	0.730942
0	Ambulance	0.709184
3	Other	0.681818

In [69]:

```
df['authorities_contacted'] = df['authorities_contacted'].replace(('None','Police','Fire',
                                                                  (0.94,0.79,0.73,0.70
```

In [70]:

```
df[['incident_severity','fraud_reported']].groupby(['incident_severity'],
                 as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[70]:

	incident_severity	fraud_reported
3	Trivial Damage	0.933333
1	Minor Damage	0.892655
2	Total Loss	0.871429
0	Major Damage	0.394928

In [71]:

```
df['incident_severity'] = df['incident_severity'].replace(('Trivial Damage','Minor Damage'
                                                         'Major Damage'),(0.94,0.89,0
```

In [72]:

```
df[['collision_type','fraud_reported']].groupby(['collision_type'],
                 as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[72]:

	collision_type	fraud_reported
1	Rear Collision	0.772340
2	Side Collision	0.746377
0	Front Collision	0.724409

In [73]:

```
df['collision_type'] = df['collision_type'].replace(('Rear Collision', 'Side Collision', '
                                                   (0.78,0.74,0.72))
```

In [74]:

```
df[['incident_type','fraud_reported']].groupby(['incident_type'],
                 as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[74]:

	incident_type	fraud_reported
3	Vehicle Theft	0.914894
1	Parked Car	0.904762
0	Multi-vehicle Collision	0.727924
2	Single Vehicle Collision	0.709677

In [75]:

```
df['incident_type'] = df['incident_type'].replace(('Vehicle Theft','Parked Car','Multi-veh
                                'Single Vehicle Collision'),(0.91, 0.90, 0.72,0.70))
```

In [76]:

```
df['incident_date'] = pd.to_datetime(df['incident_date'], errors = 'coerce')

# extracting days and month from date
df['incident_month'] = df['incident_date'].dt.month
df['incident_day'] = df['incident_date'].dt.day
```

In [77]:

```
df[['insured_relationship','fraud_reported']].groupby(['insured_relationship'],
                as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[77]:

	insured_relationship	fraud_reported
0	husband	0.794118
3	own-child	0.786885
4	unmarried	0.758865
1	not-in-family	0.741379
5	wife	0.729032
2	other-relative	0.706215

In [78]:

```
df['insured_relationship'] = df['insured_relationship'].replace(('husband','own-child','un
                                'not-in-family','wife','other-relative'),(0.79,0.7
```

In [79]:

```
df[['insured_hobbies','fraud_reported']].groupby(['insured_hobbies'],
                as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[79]:

	insured_hobbies	fraud_reported
4	camping	0.909091
11	kayaking	0.907407
9	golf	0.890909
7	dancing	0.883721
3	bungie-jumping	0.839286
12	movies	0.836364
1	basketball	0.823529
8	exercise	0.807018
17	sleeping	0.804878
18	video-games	0.800000
16	skydiving	0.775510
13	paintball	0.771930
10	hiking	0.769231
0	base-jumping	0.734694
15	reading	0.734375
[illegible]	olo	0.723404

	insured_hobbies	fraud_reported
2	board-games	0.708333
19	yachting	0.698113
6	cross-fit	0.257143
5	chess	0.173913

In [80]:

```
df['insured_hobbies'] = df['insured_hobbies'].replace(('camping', 'kayaking', 'golf','danc
        'bungie-jumping','movies', 'basketball','exercise','sleeping','video-games','skydi
            'hiking','base-jumping','reading','polo','board-games','yachting', 'cross-fit'
                0.89, 0.88,0.84,0.83,0.82,0.81,0.805,0.80,0.78,0.77,0.76,0.73,0.73,0.72,0.
```

In [81]:

```
df[['insured_occupation','fraud_reported']].groupby(['insured_occupation'],
                as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[81]:

	insured_occupation	fraud_reported
7	other-service	0.830986
8	priv-house-serv	0.830986
0	adm-clerical	0.830769
5	handlers-cleaners	0.796296
9	prof-specialty	0.788235
10	protective-serv	0.777778
6	machine-op-inspct	0.763441
1	armed-forces	0.753623
11	sales	0.723684
12	tech-support	0.717949
13	transport-moving	0.708333
2	craft-repair	0.702703
4	farming-fishing	0.698113
3	exec-managerial	0.631579

In [82]:

```
df['insured_occupation'] = df['insured_occupation'].replace(('other-service','priv-house-s
                        'adm-clerical','handlers-cleaners','prof-specialty','protective-se
                'machine-op-inspct','armed-forces','sales','tech-support','transport-movir
                    'farming-fishing','exec-managerial'),(0.84, 0.84,0.83, 0.79,0.78,0.77,
                                                        0.705,0.70,0.69,0.63))
```

In [83]:

```
df[['insured_education_level','fraud_reported']].groupby(['insured_education_level'],
                as_index = False).mean().sort_values(by = 'fraud_reported', ascending = Fa
```

Out[83]:

	insured_education_level	fraud_reported
5	Masters	0.776224
2	High School	0.775000
0	Associate	0.765517
3	JD	0.739130

	insured_education_level	fraud_reported
1	College	0.737705
4	MD	0.736111
6	PhD	0.736000

In [84]:

```
df['insured_education_level'] = df['insured_education_level'].replace(('Masters', 'High Sc
                                        'JD','College', 'MD','PhD'),(0.78,0.77,0.76,0.74,(
```

In [85]:

```
df[['insured_sex','fraud_reported']].groupby(['insured_sex'], as_index = False).mean().sor
    by = 'fraud_reported', ascending = False)
```

Out[85]:

	insured_sex	fraud_reported
0	FEMALE	0.765363
1	MALE	0.738661

In [86]:

```
df['insured_sex'] = df['insured_sex'].replace(('FEMALE','MALE'),(0.76,0.73))
```

In [87]:

```
df[['policy_csl','fraud_reported']].groupby(['policy_csl'], as_index = False).mean().sort_
    by = 'fraud_reported', ascending = False)
```

Out[87]:

	policy_csl	fraud_reported
2	500/1000	0.783333
0	100/300	0.742120
1	250/500	0.737892

In [88]:

```
df['policy_csl'] = df['policy_csl'].replace(('500/1000','100/300','250/500'),(0.78,0.74,0.
```

In [89]:

```
df[['policy_state','fraud_reported']].groupby(['policy_state'], as_index = False).mean().s
    by = 'fraud_reported', ascending = False)
```

Out[89]:

	policy_state	fraud_reported
0	IL	0.772189
1	IN	0.745161
2	OH	0.741477

In [90]:

```
df['policy_state'] = df['policy_state'].replace(('IL','IN','OH'),(0.77,0.745,0.74))
```

In [91]:

```
# let's delete unnecassary columns

df = df.drop(['policy_number','policy_bind_date', 'incident_date','incident_location','aut

# let's check the columns after deleting the columns
df.columns
```

Out[91]:
```
Index(['months_as_customer', 'age', '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_type', 'collision_type',
       'incident_severity', 'authorities_contacted', 'incident_state',
       'incident_city', '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_year', 'fraud_reported', 'incident_month', 'incident_day'],
      dtype='object')
```

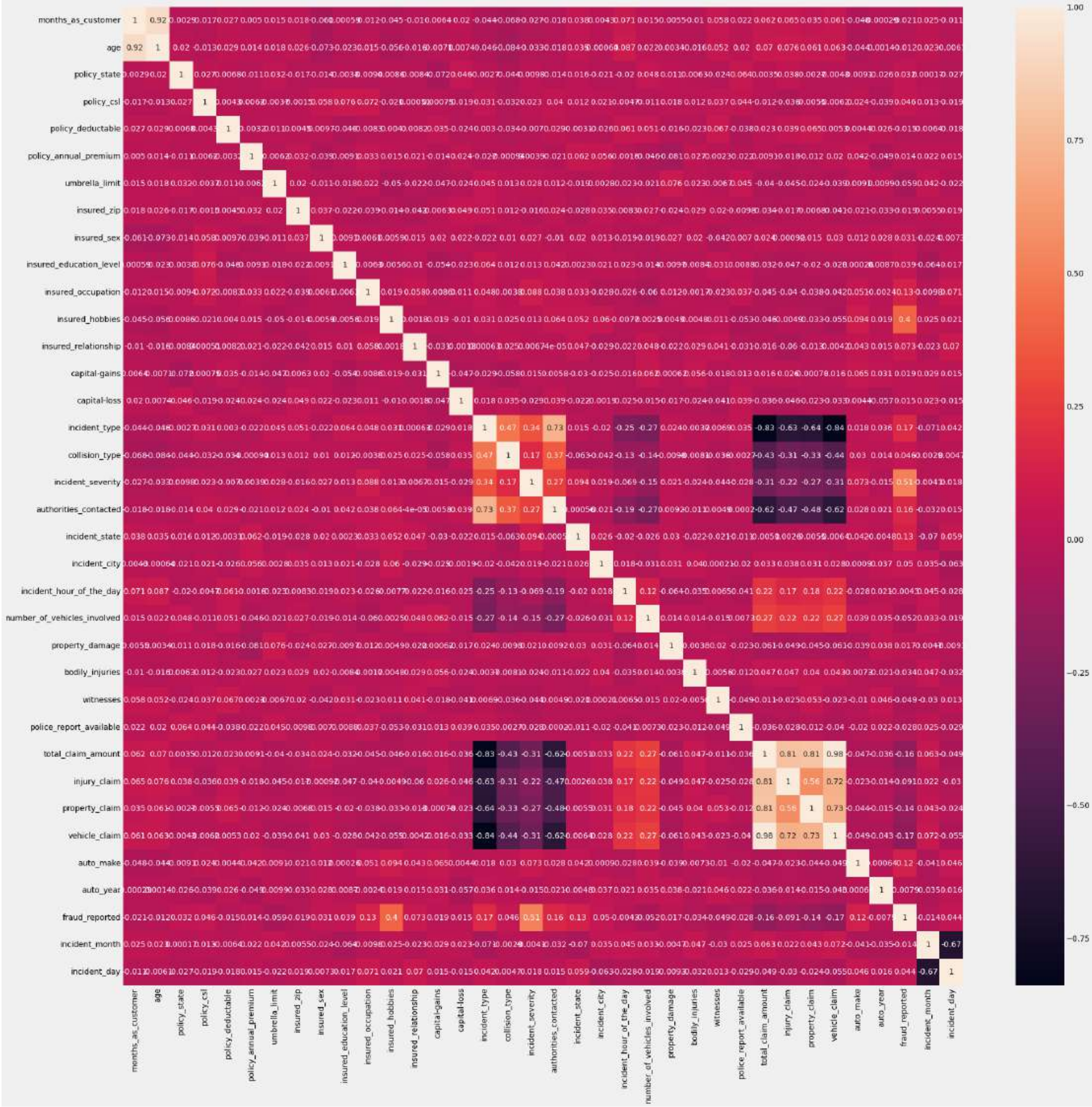
In [127…
```
# now all the categorical columns are converted into numerical columns
# All the unnecassary columns also droped from df dataset
# now lets see the df dataset
df
```

Out[127…

	months_as_customer	age	policy_state	policy_csl	policy_deductable	policy_annual_premium	umbrella_limit
0	328	48	0.740	0.73	1000	1406.91	0
1	228	42	0.745	0.73	2000	1197.22	5000000
2	134	29	0.740	0.74	2000	1413.14	5000000
3	256	41	0.770	0.73	2000	1415.74	6000000
4	228	44	0.770	0.78	1000	1583.91	6000000
...	...	...	...	...	...	...	...
995	3	38	0.740	0.78	1000	1310.80	0
996	285	41	0.770	0.74	1000	1436.79	0
997	130	34	0.740	0.73	500	1383.49	3000000
998	458	62	0.770	0.78	2000	1356.92	5000000
999	456	60	0.740	0.73	1000	766.19	0

1000 rows × 36 columns

In [129…
```
# checking the co-relation values with the help of heat map
import matplotlib.pyplot as plt
corrmat = df.corr()
plt.figure(figsize=(30,30))
#plot heat map
g=sns.heatmap(corrmat,annot=True)
```



In [92]:

```
# let's split the data into dependent and independent sets

x = df.drop(['fraud_reported'], axis = 1)
y = df['fraud_reported']
# checking the x, y shape
print("Shape of x :", x.shape)
print("Shape of y :", y.shape)
```

```
Shape of x : (1000, 35)
Shape of y : (1000,)
```

In [93]:

```
#removing the skewness by yeo johnson method
from sklearn.preprocessing import power_transform
x=power_transform(x,method='yeo-johnson')
x
```

Out[93]:

```
array([[ 1.05127872,  1.00873272,  0.        , ..., -0.19745541,
        ...45698,  1.04604418],
```

Loading [MathJax]/extensions/Safe.js

```
       [ 0.30453584,  0.43143333,  0.        , ...,  0.30262249,
        -1.10045698,  0.82475977],
       [-0.51122603, -1.13951302,  0.        , ...,  0.30262249,
        -0.02541209,  0.88243021],
       ...,
       [-0.54970642, -0.47044379,  0.        , ..., -1.49627768,
        -1.10045698,  0.93846168],
       [ 1.9126537 ,  2.13055434,  0.        , ..., -1.17624558,
        -0.02541209,  1.09778562],
       [ 1.90009963,  1.98572613,  0.        , ...,  0.30262249,
        -0.02541209,  1.09778562]])
```

In [94]:
```
pd.DataFrame(x).skew()
```

Out[94]:
```
0    -0.135661
1    -0.001945
2     0.000000
3     0.000000
4     0.023988
5     0.004758
6    -7.865930
7     0.000000
8    -0.148630
9    -0.002118
10   -0.020558
11   -0.294249
12   -0.020572
13    0.038722
14    0.090488
15    0.941516
16   -0.072601
17   -0.828981
18    0.231201
19   -0.038724
20   -0.049267
21   -0.256957
22    0.363693
23   -0.863806
24   -0.128799
25   -0.153648
26   -0.802728
27   -0.510354
28   -0.415781
29   -0.358814
30   -0.522718
31    0.009484
32   -0.012491
33    0.306468
34   -0.221642
dtype: float64
```

In [95]:
```
from sklearn.preprocessing import MinMaxScaler
mms=MinMaxScaler()
x1=mms.fit_transform(x)
```

In [96]:
```
# importing all the algorithems for checking the accuracy_score  and model perforfance
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split,cross_val_score,GridSearchCV
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report,f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
f    sklearn.svm import SVC
```

```python
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
```

In [98]:

```python
model=[RandomForestClassifier(),DecisionTreeClassifier(),SVC(),KNeighborsClassifier(),Gaus
max_r2_score=0
for i_state in range(0,10):
    x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=i_state,test_size=0.2
    for a in model:
        a.fit(x_train,y_train)
        pred=a.predict(x_test)
        score=accuracy_score(y_test,pred)
        print('score for random_state',i_state,'is',score)
        if score>max_r2_score:
            max_r2_score=score
            Final_state=i_state
            Final_model= a
print('accuracy_score ',max_r2_score,'for random state ',Final_state, 'and model is ',Fina
```

```
score for random_state 0 is 0.785
score for random_state 0 is 0.83
score for random_state 0 is 0.81
score for random_state 0 is 0.7
score for random_state 0 is 0.795
score for random_state 0 is 0.84
score for random_state 1 is 0.845
score for random_state 1 is 0.775
score for random_state 1 is 0.82
score for random_state 1 is 0.75
score for random_state 1 is 0.84
score for random_state 1 is 0.84
score for random_state 2 is 0.83
score for random_state 2 is 0.855
score for random_state 2 is 0.855
score for random_state 2 is 0.76
score for random_state 2 is 0.865
score for random_state 2 is 0.87
score for random_state 3 is 0.845
score for random_state 3 is 0.745
score for random_state 3 is 0.825
score for random_state 3 is 0.75
score for random_state 3 is 0.83
score for random_state 3 is 0.865
score for random_state 4 is 0.805
score for random_state 4 is 0.775
score for random_state 4 is 0.805
score for random_state 4 is 0.685
score for random_state 4 is 0.84
score for random_state 4 is 0.83
score for random_state 5 is 0.835
score for random_state 5 is 0.78
score for random_state 5 is 0.81
score for random_state 5 is 0.755
score for random_state 5 is 0.805
score for random_state 5 is 0.815
score for random_state 6 is 0.85
score for random_state 6 is 0.79
score for random_state 6 is 0.845
score for random_state 6 is 0.795
score for random_state 6 is 0.795
score for random_state 6 is 0.85
score for random_state 7 is 0.83
score for random_state 7 is 0.76
om_state 7 is 0.79
```

```
score for random_state 7 is 0.72
score for random_state 7 is 0.77
score for random_state 7 is 0.815
score for random_state 8 is 0.83
score for random_state 8 is 0.73
score for random_state 8 is 0.835
score for random_state 8 is 0.785
score for random_state 8 is 0.815
score for random_state 8 is 0.855
score for random_state 9 is 0.84
score for random_state 9 is 0.75
score for random_state 9 is 0.81
score for random_state 9 is 0.74
score for random_state 9 is 0.77
score for random_state 9 is 0.835
accuracy_score  0.87 for random state  2 and model is  LogisticRegression()
```

In [99]:

```
# we are training the model with LogisticRegression for randomstate 2 and checking the acc
lr=LogisticRegression()
x_train,x_test,y_train,y_test=train_test_split(x1,y,random_state=2,test_size=0.2)
lr.fit(x_train,y_train)
lr.score(x_train,y_train)
pred_y=lr.predict(x_test)
lrs=accuracy_score(y_test,pred_y)
print('accuracy_score =',lrs*100)
print(classification_report(y_test,pred_y))
print(confusion_matrix(y_test,pred_y))
print('F1_score = ',f1_score(y_test,pred_y)*100)
from sklearn.model_selection import cross_val_score
cv_score=cross_val_score(lr,x1,y,cv=5)
cv_mean=cv_score.mean()
print("cross_val_score=",cv_mean*100)
```

```
accuracy_score = 87.0
              precision    recall  f1-score   support

           0       0.78      0.65      0.71        49
           1       0.89      0.94      0.92       151

    accuracy                           0.87       200
   macro avg       0.84      0.80      0.81       200
weighted avg       0.87      0.87      0.87       200

[[ 32  17]
 [  9 142]]
F1_score =  91.61290322580645
cross_val_score= 83.9
```

In [101…

```
# for selecting the best parameters
grid={"C":np.logspace(-4,4,20), "penalty":["l1","l2"]}
lr=LogisticRegression()
clf=GridSearchCV(lr,grid)
clf.fit(x_train,y_train)
print(clf.best_params_)
```

```
{'C': 11.288378916846883, 'penalty': 'l2'}
```

In [109…

```
# After selecting the best parameter we need to implement them on the algorithms for check
lr=LogisticRegression(C=11.28837,penalty="l2")
lr.fit(x_train,y_train)
lr.score(x_train,y_train)
pred_lr=lr.predict(x_test)
```

Loading [MathJax]/extensions/Safe.js

```
lras=accuracy_score(y_test,pred_lr)
print('accuracy_score =',lras*100)
```

accuracy_score = 87.5

There is no much difference in the accuracy score value after perforfing the hyperparameter tuning by gridsearchCV

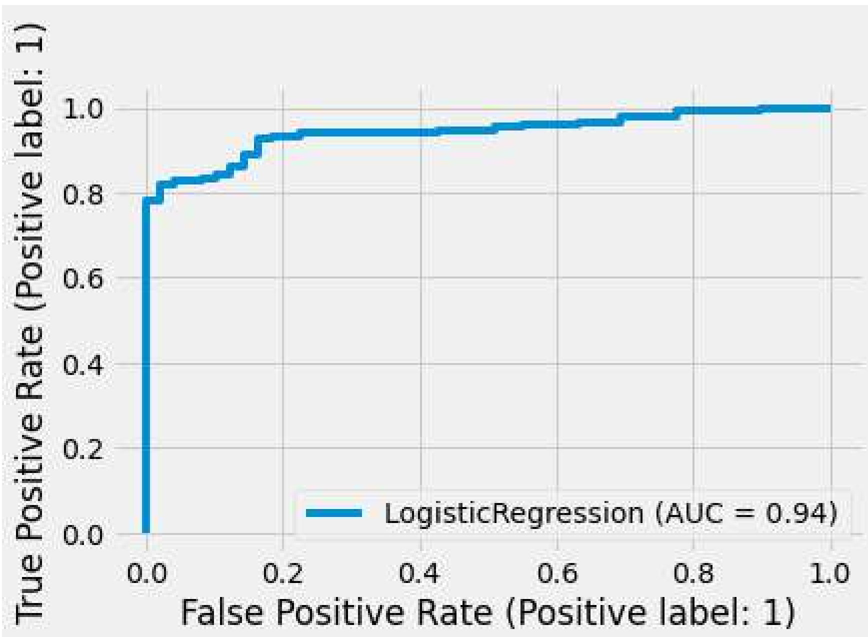
Accuracy_score is 87.5

In [126…

```
from sklearn import metrics
metrics.plot_roc_curve(lr, x_test, y_test)
metrics.roc_auc_score(y_test, pred_lr, average=None)
```

Out[126…

0.8069333693742398



By the above graph we can say that area under the curve is 94% which is very good value

CONCLUSION

1) area under the curve is 94percent which is very good value

2) The LogisticRegression giving the best accuracy value

3) accuracy_score, F1_score, Classification_report, Confussion_matrix and AUC value is shown in the above table

4) LogisticRegression is giving the best accuracy score so we need to save the LogisticRegression predicted values by using pickel