1: Importing Libraries

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import (confusion_matrix ,ConfusionMatrixDisplay ,accur
```

2: Loading Dataset

In [2]:
```
# Loading dataset,
# Data quick Check.
df = pd.read_csv('Car Insurance prediction Domain-BFSI (1).csv')
```

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

Out[3]:

	policy_id	policy_tenure	age_of_car	age_of_policyholder	area_cluster	popu
0	ID00001	0.515874	0.05	0.644231	C1	
1	ID00002	0.672619	0.02	0.375000	C2	
2	ID00003	0.841110	0.02	0.384615	C3	
3	ID00004	0.900277	0.11	0.432692	C4	
4	ID00005	0.596403	0.11	0.634615	C5	

5 rows × 44 columns

In [4]:
```
#Checking shape of the dataset
df.shape
```

Out[4]:
```
(58592, 44)
```

In [5]:
```
# Checking the size of dataset
df.size
```

Out[5]: 2578048

In [6]:
```
# Checking column names of dataset
df.columns
```

Out[6]:
```
Index(['policy_id', 'policy_tenure', 'age_of_car', 'age_of_policyholder',
       'area_cluster', 'population_density', 'make', 'segment', 'model',
       'fuel_type', 'max_torque', 'max_power', 'engine_type', 'airbags',
       'is_esc', 'is_adjustable_steering', 'is_tpms', 'is_parking_sensors',
       'is_parking_camera', 'rear_brakes_type', 'displacement', 'cylinder',
       'transmission_type', 'gear_box', 'steering_type', 'turning_radius',
       'length', 'width', 'height', 'gross_weight', 'is_front_fog_lights',
       'is_rear_window_wiper', 'is_rear_window_washer',
       'is_rear_window_defogger', 'is_brake_assist', 'is_power_door_locks',
       'is_central_locking', 'is_power_steering',
       'is_driver_seat_height_adjustable', 'is_day_night_rear_view_mirror',
       'is_ecw', 'is_speed_alert', 'ncap_rating', 'is_claim'],
      dtype='object')
```

In [7]:
```
# Checking the Data-Types of the columns
df.dtypes
```

Out[7]:
```
policy_id                            object
policy_tenure                       float64
age_of_car                          float64
age_of_policyholder                 float64
area_cluster                         object
population_density                    int64
make                                  int64
segment                              object
model                                object
fuel_type                            object
max_torque                           object
max_power                            object
engine_type                          object
airbags                               int64
is_esc                               object
is_adjustable_steering               object
is_tpms                              object
is_parking_sensors                   object
is_parking_camera                    object
rear_brakes_type                     object
displacement                          int64
cylinder                              int64
transmission_type                    object
gear_box                              int64
steering_type                        object
turning_radius                      float64
length                                int64
width                                 int64
height                                int64
gross_weight                          int64
is_front_fog_lights                  object
is_rear_window_wiper                 object
is_rear_window_washer                object
is_rear_window_defogger              object
is_brake_assist                      object
is_power_door_locks                  object
is_central_locking                   object
is_power_steering                    object
is_driver_seat_height_adjustable     object
is_day_night_rear_view_mirror        object
is_ecw                               object
is_speed_alert                       object
ncap_rating                           int64
is_claim                              int64
dtype: object
```

In [8]:
```
#Checking the information of the dataset
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 58592 entries, 0 to 58591
Data columns (total 44 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   policy_id                         58592 non-null  object 
 1   policy_tenure                     58592 non-null  float64
 2   age_of_car                        58592 non-null  float64
 3   age_of_policyholder               58592 non-null  float64
 4   area_cluster                      58592 non-null  object 
 5   population_density                58592 non-null  int64  
 6   make                              58592 non-null  int64  
 7   segment                           58592 non-null  object 
 8   model                             58592 non-null  object 
 9   fuel_type                         58592 non-null  object 
 10  max_torque                        58592 non-null  object 
 11  max_power                         58592 non-null  object 
 12  engine_type                       58592 non-null  object 
 13  airbags                           58592 non-null  int64  
 14  is_esc                            58592 non-null  object 
 15  is_adjustable_steering            58592 non-null  object 
 16  is_tpms                           58592 non-null  object 
 17  is_parking_sensors                58592 non-null  object 
 18  is_parking_camera                 58592 non-null  object 
 19  rear_brakes_type                  58592 non-null  object 
 20  displacement                      58592 non-null  int64  
 21  cylinder                          58592 non-null  int64  
 22  transmission_type                 58592 non-null  object 
 23  gear_box                          58592 non-null  int64  
 24  steering_type                     58592 non-null  object 
 25  turning_radius                    58592 non-null  float64
 26  length                            58592 non-null  int64  
 27  width                             58592 non-null  int64  
 28  height                            58592 non-null  int64  
 29  gross_weight                      58592 non-null  int64  
 30  is_front_fog_lights               58592 non-null  object 
 31  is_rear_window_wiper              58592 non-null  object 
 32  is_rear_window_washer             58592 non-null  object 
 33  is_rear_window_defogger           58592 non-null  object 
 34  is_brake_assist                   58592 non-null  object 
 35  is_power_door_locks               58592 non-null  object 
 36  is_central_locking                58592 non-null  object 
 37  is_power_steering                 58592 non-null  object 
 38  is_driver_seat_height_adjustable  58592 non-null  object 
 39  is_day_night_rear_view_mirror     58592 non-null  object 
 40  is_ecw                            58592 non-null  object 
 41  is_speed_alert                    58592 non-null  object 
 42  ncap_rating                       58592 non-null  int64  
 43  is_claim                          58592 non-null  int64  
dtypes: float64(4), int64(12), object(28)
memory usage: 19.7+ MB
```

In [9]:
```
# is_claim Is the Target column
# Checking the value counts for number of unique values
df['is_claim'].value_counts()
```

Out[9]:
```
is_claim
0    54844
1     3748
Name: count, dtype: int64
```

In [10]:
```
# Checking the percentage of number of unique values are present
df['is_claim'].value_counts(normalize=True)*100
```

Out[10]:
```
is_claim
0    93.603222
1     6.396778
Name: proportion, dtype: float64
```

Note 1: Target column is imballance

3 Checking missing & duplicate data

In [11]:
```
df.isnull().sum() # Zero Null values
```

```
Out[11]: policy_id                            0
         policy_tenure                        0
         age_of_car                           0
         age_of_policyholder                  0
         area_cluster                         0
         population_density                   0
         make                                 0
         segment                              0
         model                                0
         fuel_type                            0
         max_torque                           0
         max_power                            0
         engine_type                          0
         airbags                              0
         is_esc                               0
         is_adjustable_steering               0
         is_tpms                              0
         is_parking_sensors                   0
         is_parking_camera                    0
         rear_brakes_type                     0
         displacement                         0
         cylinder                             0
         transmission_type                    0
         gear_box                             0
         steering_type                        0
         turning_radius                       0
         length                               0
         width                                0
         height                               0
         gross_weight                         0
         is_front_fog_lights                  0
         is_rear_window_wiper                 0
         is_rear_window_washer                0
         is_rear_window_defogger              0
         is_brake_assist                      0
         is_power_door_locks                  0
         is_central_locking                   0
         is_power_steering                    0
         is_driver_seat_height_adjustable     0
         is_day_night_rear_view_mirror        0
         is_ecw                               0
         is_speed_alert                       0
         ncap_rating                          0
         is_claim                             0
         dtype: int64
```

```
In [12]: #Checking duplicated values
         df.duplicated().sum() #-- Zero duplicated values
```

```
Out[12]: 0
```

Note 2: zero missing values

Note 3: zero duplicate values

4 Sorting Categorical and Numerical Columns in list

In [13]:
```
categorical=[]
numerical=[]
```

seperating categorical columns

In [14]:
```
for i in df.select_dtypes(include='object').columns:
    print(i,":",df[i].nunique())
    categorical.append(i)
print("\n\n")
print(categorical)
```

```
policy_id : 58592
area_cluster : 22
segment : 6
model : 11
fuel_type : 3
max_torque : 9
max_power : 9
engine_type : 11
is_esc : 2
is_adjustable_steering : 2
is_tpms : 2
is_parking_sensors : 2
is_parking_camera : 2
rear_brakes_type : 2
transmission_type : 2
steering_type : 3
is_front_fog_lights : 2
is_rear_window_wiper : 2
is_rear_window_washer : 2
is_rear_window_defogger : 2
is_brake_assist : 2
is_power_door_locks : 2
is_central_locking : 2
is_power_steering : 2
is_driver_seat_height_adjustable : 2
is_day_night_rear_view_mirror : 2
is_ecw : 2
is_speed_alert : 2



['policy_id', 'area_cluster', 'segment', 'model', 'fuel_type', 'max_torque',
'max_power', 'engine_type', 'is_esc', 'is_adjustable_steering', 'is_tpms',
'is_parking_sensors', 'is_parking_camera', 'rear_brakes_type', 'transmission
_type', 'steering_type', 'is_front_fog_lights', 'is_rear_window_wiper', 'is_
rear_window_washer', 'is_rear_window_defogger', 'is_brake_assist', 'is_power
_door_locks', 'is_central_locking', 'is_power_steering', 'is_driver_seat_hei
ght_adjustable', 'is_day_night_rear_view_mirror', 'is_ecw', 'is_speed_aler
t']
```

columns which are numerical in nature but they are categorical (as its unique values are very less)

In [15]:
```
# Aproximatly 60K rows and unique values which are less than 25 we are consi

for i in df.select_dtypes(include=['int64','float64']).columns:
    print(i,":",df[i].nunique())
    if df[i].nunique()>25:
        numerical.append(i)
    else:
        categorical.append(i)
print("\n\n")
print(numerical)
```

```
policy_tenure : 58592
age_of_car : 49
age_of_policyholder : 75
population_density : 22
make : 5
airbags : 3
displacement : 9
cylinder : 2
gear_box : 2
turning_radius : 9
length : 9
width : 10
height : 11
gross_weight : 10
ncap_rating : 5
is_claim : 2



['policy_tenure', 'age_of_car', 'age_of_policyholder']
```

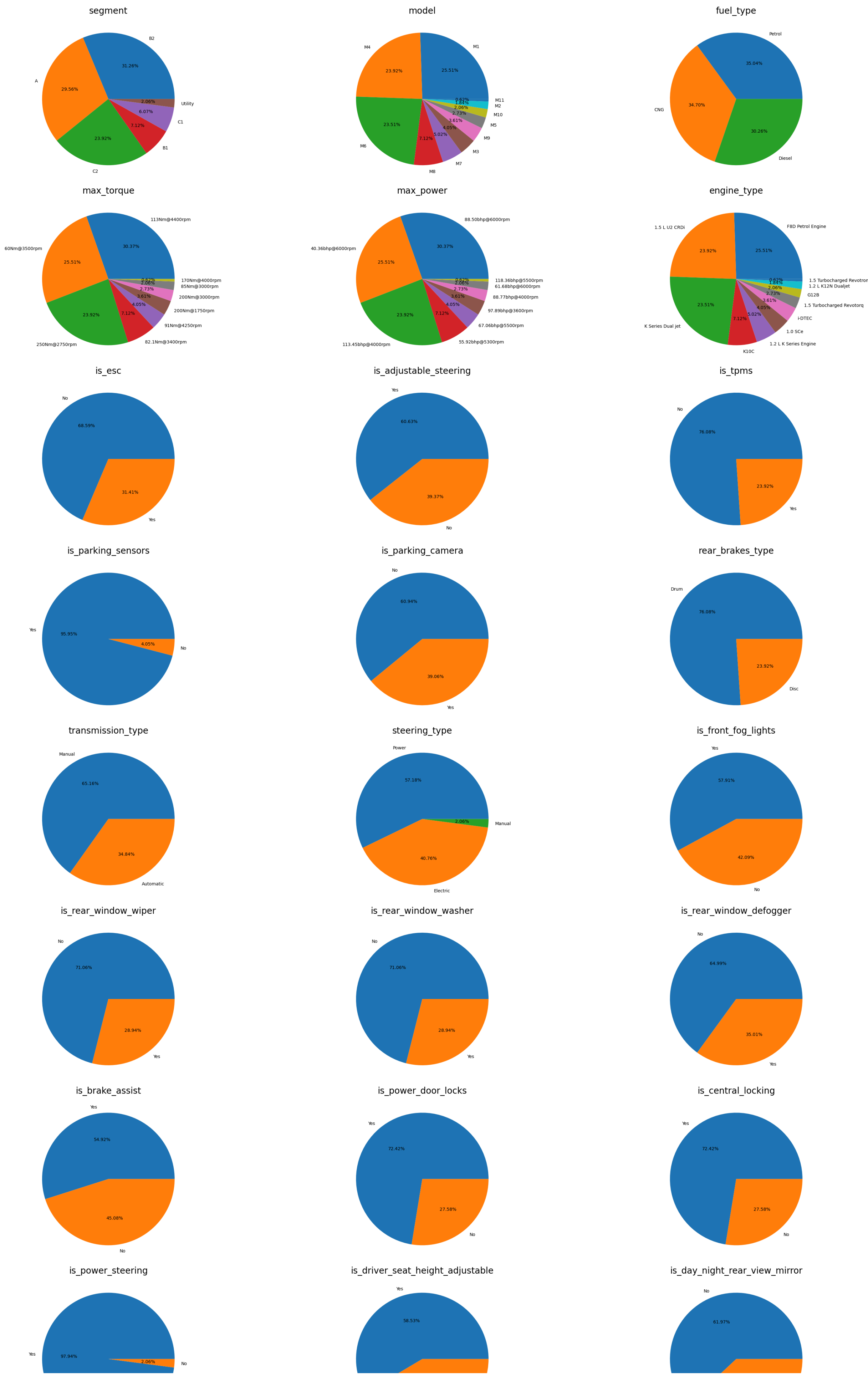
5 Exploratory Data Analysis(EDA)

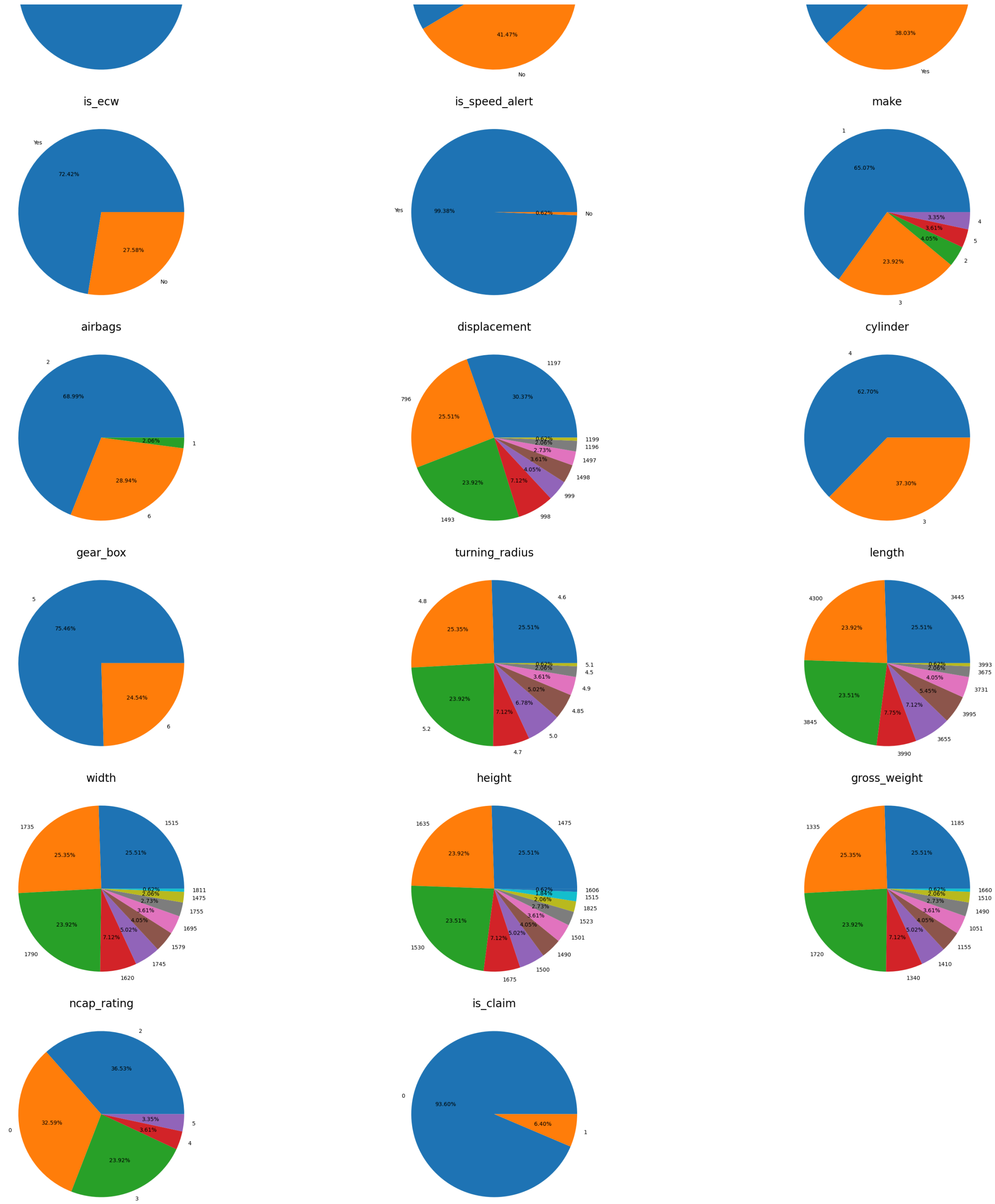
A] Categorical EDA

Pie Chart

In [16]:
```
# # Only 2 features are above 11 for categorical..
# # only 2 features are == 11 rest are below 11 for cat...
# # Only 1 feature is having 22 for that bar chart preffer..
fig=plt.figure(figsize=(30,80))
count=1
for i in categorical:
     if df[i].nunique() < 15:
         keys=df[i].value_counts().keys()
         values=df[i].value_counts().values
         plt.subplot(14,3,count)
         plt.pie(x=values, labels=keys, autopct='%0.2f%%')
```

```
        plt.title(i, fontsize=20)
        fig.tight_layout()
        count+=1
plt.show()
```

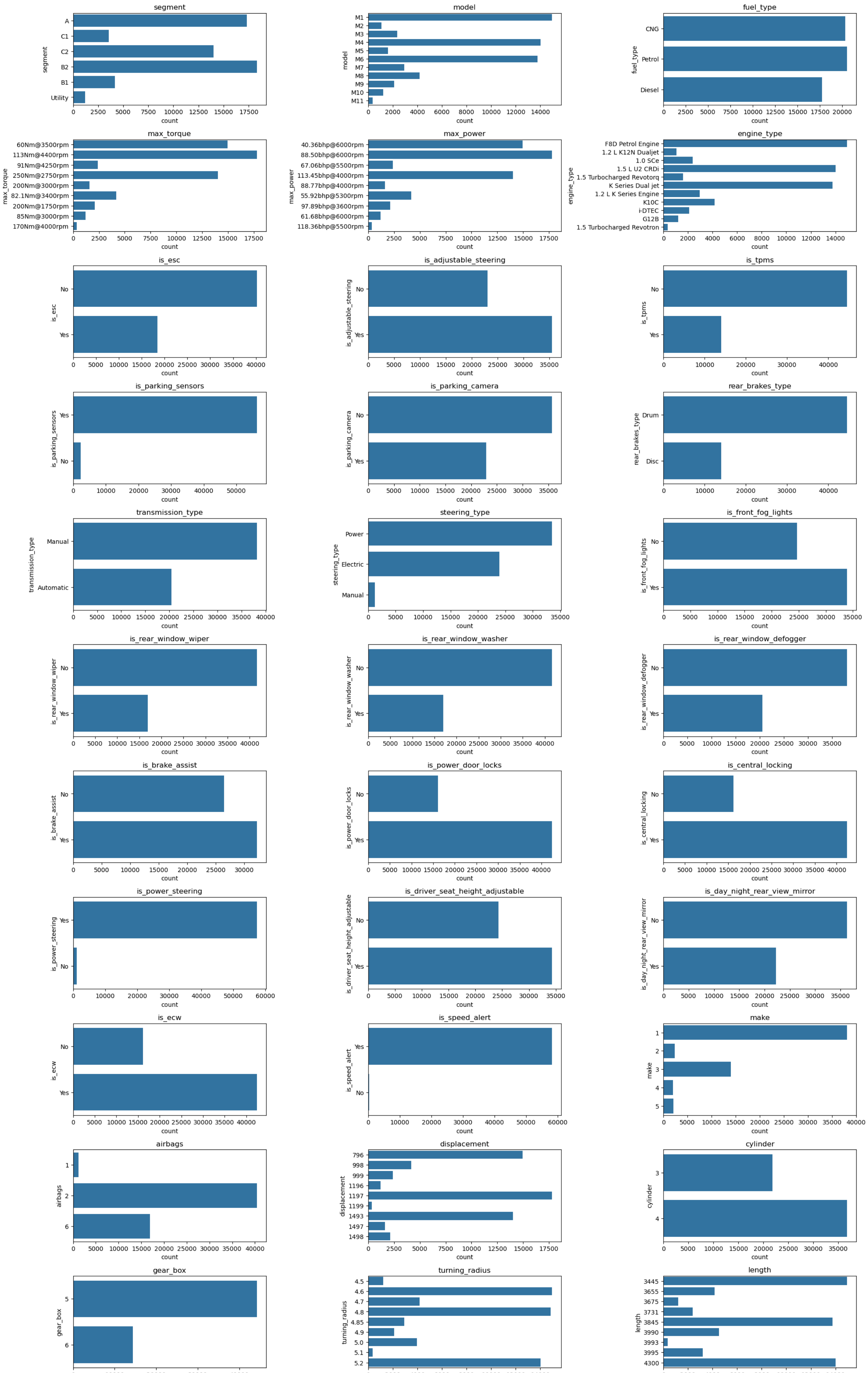

segment
B2
31.26%
A
29.56%
2.06%
Utility
6.07%
C1
7.12%
B1
23.92%
C2
model
M4
M1
23.92%
25.51%
M11
M2
M10
M5
M9
M3
M7
M8
M6
23.51%
7.12%
5.02%
4.05%
3.61%
2.73%
2.06%
1.84%
0.62%
fuel_type
Petrol
35.04%
CNG
34.70%
30.26%
Diesel
max_torque
113Nm@4400rpm
30.37%
60Nm@3500rpm
25.51%
170Nm@4000rpm
85Nm@3000rpm
200Nm@3000rpm
200Nm@1750rpm
91Nm@4250rpm
82.1Nm@3400rpm
250Nm@2750rpm
23.92%
7.12%
4.05%
3.61%
2.73%
max_power
88.50bhp@6000rpm
30.37%
40.36bhp@6000rpm
25.51%
118.36bhp@5500rpm
61.68bhp@6000rpm
88.77bhp@4000rpm
97.89bhp@3600rpm
67.06bhp@5500rpm
55.92bhp@5300rpm
113.45bhp@4000rpm
23.92%
7.12%
4.05%
3.61%
2.73%
engine_type
1.5 L U2 CRDi
F8D Petrol Engine
23.92%
25.51%
1.5 Turbocharged Revotron
1.2 L K12N Dualjet
G12B
1.5 Turbocharged Revotorq
i-DTEC
1.0 SCe
1.2 L K Series Engine
K10C
K Series Dual jet
23.51%
7.12%
5.02%
4.05%
3.61%
2.73%
2.06%
0.62%
is_esc
No
68.59%
31.41%
Yes
is_adjustable_steering
Yes
60.63%
39.37%
No
is_tpms
No
76.08%
23.92%
Yes
is_parking_sensors
Yes
95.95%
4.05%
No
is_parking_camera
No
60.94%
39.06%
Yes
rear_brakes_type
Drum
76.08%
23.92%
Disc
transmission_type
Manual
65.16%
34.84%
Automatic
steering_type
Power
57.18%
2.06%
Manual
40.76%
Electric
is_front_fog_lights
Yes
57.91%
42.09%
No
is_rear_window_wiper
No
71.06%
28.94%
Yes
is_rear_window_washer
No
71.06%
28.94%
Yes
is_rear_window_defogger
No
64.99%
35.01%
Yes
is_brake_assist
Yes
54.92%
45.08%
No
is_power_door_locks
Yes
72.42%
27.58%
No
is_central_locking
Yes
72.42%
27.58%
No
is_power_steering
Yes
97.94%
2.06%
No
is_driver_seat_height_adjustable
Yes
58.53%
is_day_night_rear_view_mirror
No
61.97%



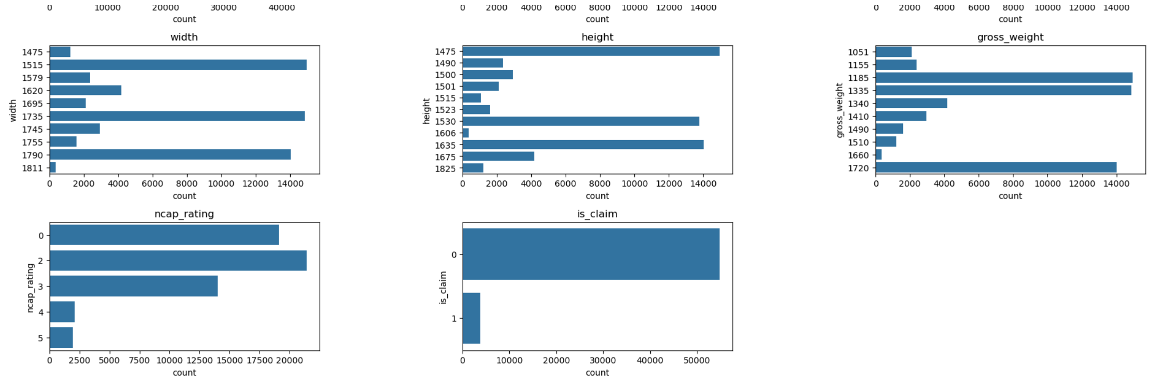
Countplot

In [17]:
```python
# Example countplot for a single categorical variable
count=1
fig=plt.figure(figsize=(20,40))
for i in categorical:
     if df[i].nunique() <= 11:
        plt.subplot(14,3,count)
        sns.countplot(y=i, data=df)
        plt.title(i)
        plt.xlabel('count')
        plt.ylabel(i)
        fig.tight_layout() #
```

```
        count+=1
plt.show()
```



segment
A
C1
C2
B2
B1
Utility
count
model
M1
M2
M3
M4
M5
M6
M7
M8
M9
M10
M11
count
fuel_type
CNG
Petrol
Diesel
count
max_torque
60Nm@3500rpm
113Nm@4400rpm
91Nm@4250rpm
250Nm@2750rpm
200Nm@3000rpm
82.1Nm@3400rpm
200Nm@1750rpm
85Nm@3000rpm
170Nm@4000rpm
count
max_power
40.36bhp@6000rpm
88.50bhp@6000rpm
67.06bhp@5500rpm
113.45bhp@4000rpm
88.77bhp@4000rpm
55.92bhp@5300rpm
97.89bhp@3600rpm
61.68bhp@6000rpm
118.36bhp@5500rpm
count
engine_type
F8D Petrol Engine
1.2 L K12N Dualjet
1.0 SCe
1.5 L U2 CRDi
1.5 Turbocharged Revotorq
K Series Dual jet
1.2 L K Series Engine
K10C
i-DTEC
G12B
1.5 Turbocharged Revotron
count
is_esc
No
Yes
count
is_adjustable_steering
No
Yes
count
is_tpms
No
Yes
count
is_parking_sensors
Yes
No
count
is_parking_camera
No
Yes
count
rear_brakes_type
Drum
Disc
count
transmission_type
Manual
Automatic
count
steering_type
Power
Electric
Manual
count
is_front_fog_lights
No
Yes
count
is_rear_window_wiper
No
Yes
count
is_rear_window_washer
No
Yes
count
is_rear_window_defogger
No
Yes
count
is_brake_assist
No
Yes
count
is_power_door_locks
No
Yes
count
is_central_locking
No
Yes
count
is_power_steering
Yes
No
count
is_driver_seat_height_adjustable
No
Yes
count
is_day_night_rear_view_mirror
No
Yes
count
is_ecw
No
Yes
count
is_speed_alert
Yes
No
count
make
1
2
3
4
5
count
airbags
1
2
6
count
displacement
796
998
999
1196
1197
1199
1493
1497
1498
count
cylinder
3
4
count
gear_box
5
6
turning_radius
4.5
4.6
4.7
4.8
4.85
4.9
5.0
5.1
5.2
length
3445
3655
3675
3731
3845
3990
3993
3995
4300



In [18]:

```
# Counting no.of unique values in every catogarical columns
df[categorical].nunique()
```

Out[18]:
```
policy_id                           58592
area_cluster                           22
segment                                 6
model                                  11
fuel_type                               3
max_torque                              9
max_power                               9
engine_type                            11
is_esc                                  2
is_adjustable_steering                  2
is_tpms                                 2
is_parking_sensors                      2
is_parking_camera                       2
rear_brakes_type                        2
transmission_type                       2
steering_type                           3
is_front_fog_lights                     2
is_rear_window_wiper                    2
is_rear_window_washer                   2
is_rear_window_defogger                 2
is_brake_assist                         2
is_power_door_locks                     2
is_central_locking                      2
is_power_steering                       2
is_driver_seat_height_adjustable        2
is_day_night_rear_view_mirror           2
is_ecw                                  2
is_speed_alert                          2
population_density                     22
make                                    5
airbags                                 3
displacement                            9
cylinder                                2
gear_box                                2
turning_radius                          9
length                                  9
width                                  10
height                                 11
gross_weight                           10
ncap_rating                             5
is_claim                                2
dtype: int64
```

In [19]:
```python
# checking unique values of categorical columns which are less than 25
for i in categorical:
    if df[i].nunique()<25:
        print(i,df[i].unique(),end="\n\n")
```

```
area_cluster ['C1' 'C2' 'C3' 'C4' 'C5' 'C6' 'C7' 'C8' 'C9' 'C10' 'C11' 'C12'
'C13'
 'C14' 'C15' 'C16' 'C17' 'C18' 'C19' 'C20' 'C21' 'C22']

segment ['A' 'C1' 'C2' 'B2' 'B1' 'Utility']

model ['M1' 'M2' 'M3' 'M4' 'M5' 'M6' 'M7' 'M8' 'M9' 'M10' 'M11']

fuel_type ['CNG' 'Petrol' 'Diesel']

max_torque ['60Nm@3500rpm' '113Nm@4400rpm' '91Nm@4250rpm' '250Nm@2750rpm'
 '200Nm@3000rpm' '82.1Nm@3400rpm' '200Nm@1750rpm' '85Nm@3000rpm'
 '170Nm@4000rpm']

max_power ['40.36bhp@6000rpm' '88.50bhp@6000rpm' '67.06bhp@5500rpm'
 '113.45bhp@4000rpm' '88.77bhp@4000rpm' '55.92bhp@5300rpm'
 '97.89bhp@3600rpm' '61.68bhp@6000rpm' '118.36bhp@5500rpm']

engine_type ['F8D Petrol Engine' '1.2 L K12N Dualjet' '1.0 SCe' '1.5 L U2 CR
Di'
 '1.5 Turbocharged Revotorq' 'K Series Dual jet' '1.2 L K Series Engine'
 'K10C' 'i-DTEC' 'G12B' '1.5 Turbocharged Revotron']

is_esc ['No' 'Yes']

is_adjustable_steering ['No' 'Yes']

is_tpms ['No' 'Yes']

is_parking_sensors ['Yes' 'No']

is_parking_camera ['No' 'Yes']

rear_brakes_type ['Drum' 'Disc']

transmission_type ['Manual' 'Automatic']

steering_type ['Power' 'Electric' 'Manual']

is_front_fog_lights ['No' 'Yes']

is_rear_window_wiper ['No' 'Yes']

is_rear_window_washer ['No' 'Yes']

is_rear_window_defogger ['No' 'Yes']

is_brake_assist ['No' 'Yes']

is_power_door_locks ['No' 'Yes']

is_central_locking ['No' 'Yes']

is_power_steering ['Yes' 'No']

is_driver_seat_height_adjustable ['No' 'Yes']
```

```
is_day_night_rear_view_mirror ['No' 'Yes']

is_ecw ['No' 'Yes']

is_speed_alert ['Yes' 'No']

population_density [ 4990 27003  4076 21622 34738 13051  6112  8794 17804 73
430  6108 34791
  5410  7788   290 16206 65567 35036 27742 20905  3264 16733]

make [1 2 3 4 5]

airbags [2 6 1]

displacement [ 796 1197  999 1493 1497  998 1498 1196 1199]

cylinder [3 4]

gear_box [5 6]

turning_radius [4.6  4.8  5.   5.2  4.85 4.7  4.9  4.5  5.1 ]

length [3445 3995 3731 4300 3990 3845 3655 3675 3993]

width [1515 1735 1579 1790 1755 1745 1620 1695 1475 1811]

height [1475 1515 1490 1635 1523 1530 1500 1675 1501 1825 1606]

gross_weight [1185 1335 1155 1720 1490 1410 1340 1051 1510 1660]

ncap_rating [0 2 3 5 4]

is_claim [0 1]
```

In [20]:

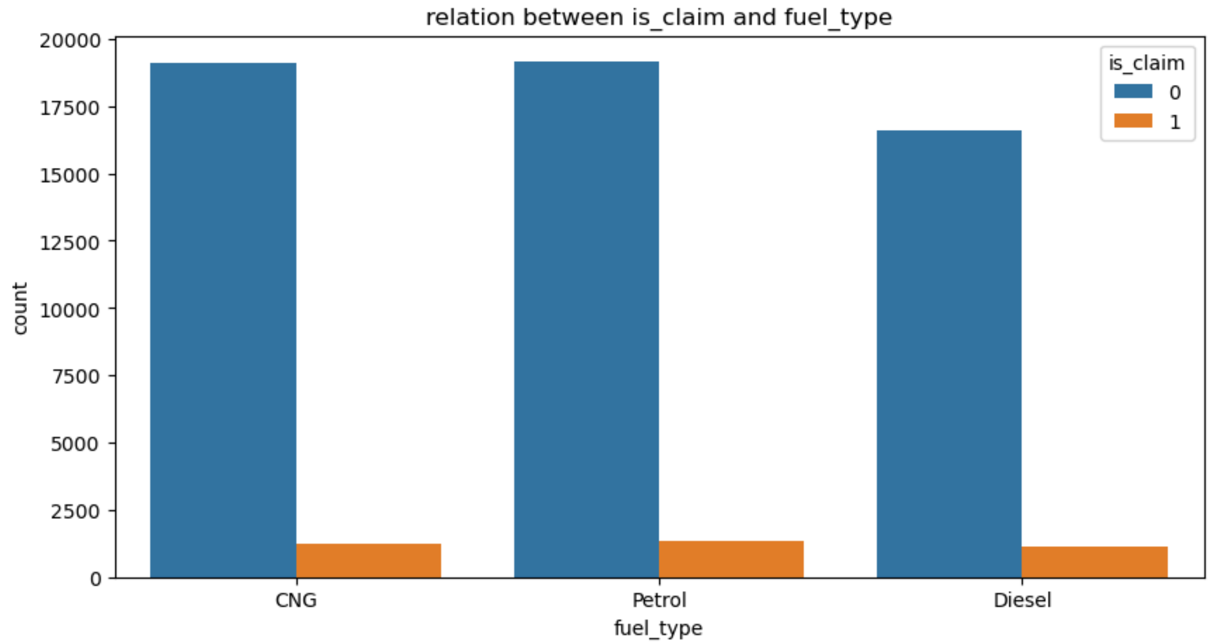
```
# fig=plt.figure(figsize=(10,10))
# count=1
# for i in categorical:
#      if df[i].nunique() <= 11:
#          keys=df[i].value_counts().keys()
#          values=df[i].value_counts().values
#          plt.figure(figsize = (10,10))
#          plt.bar(keys, values)
#          plt.title(i)
#          plt.xlabel('count')
#          plt.ylabel(i)
#          plt.bar(keys, values, color=['Red','blue','green'])
# plt.show()
```

Count plot with hue of target column(target_col= 'is_claim')

In [21]:

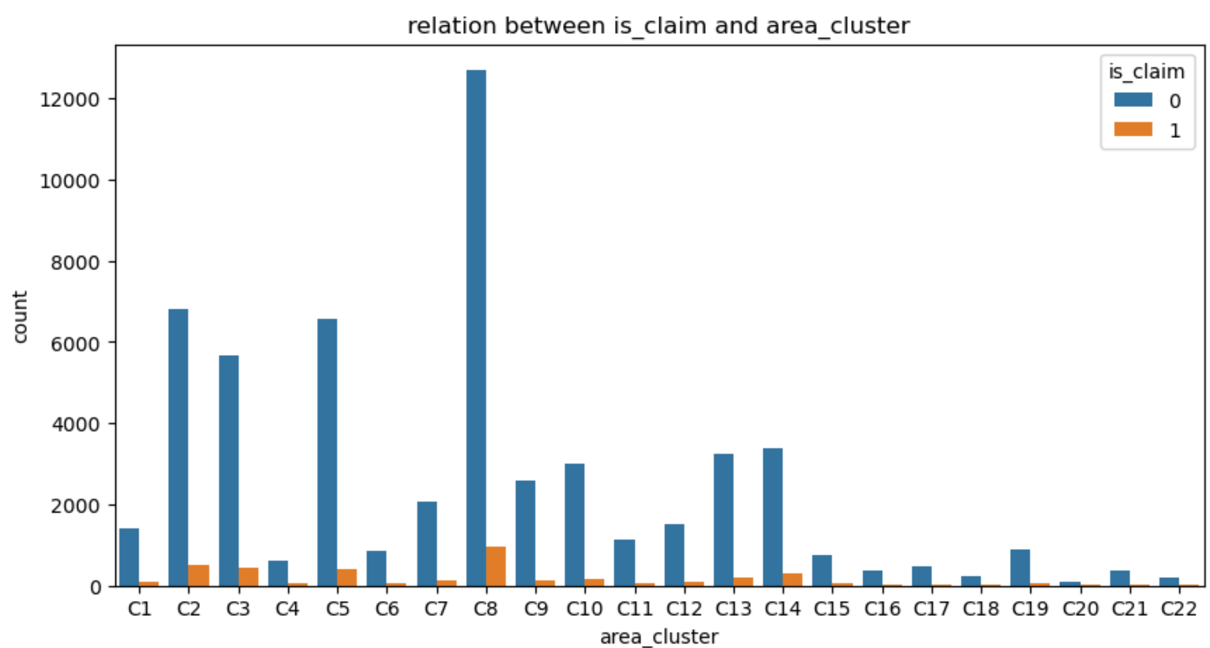
```
# Count plot with hue of target column
plt.figure(figsize=(10,5))
sns.countplot(data = df ,x= "fuel_type" , hue ="is_claim")
```

```
plt.title("relation between is_claim and fuel_type")
plt.show()
```



In [22]:

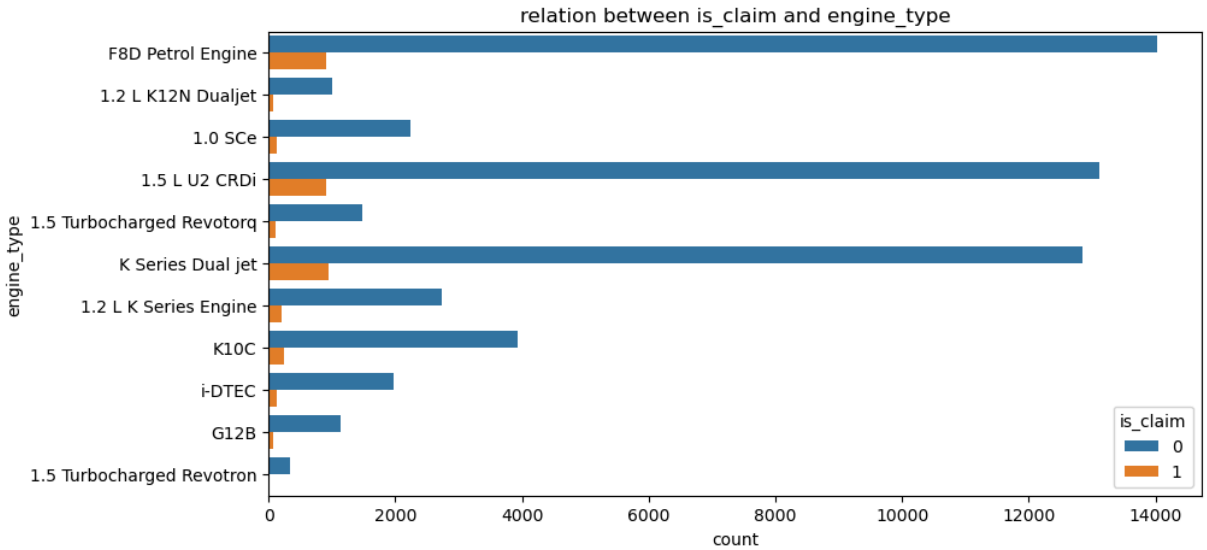
```
# Count plot with hue of target column
plt.figure(figsize=(10,5))
sns.countplot(data = df ,x= "area_cluster" , hue ="is_claim")
plt.title("relation between is_claim and area_cluster")
plt.show()
```



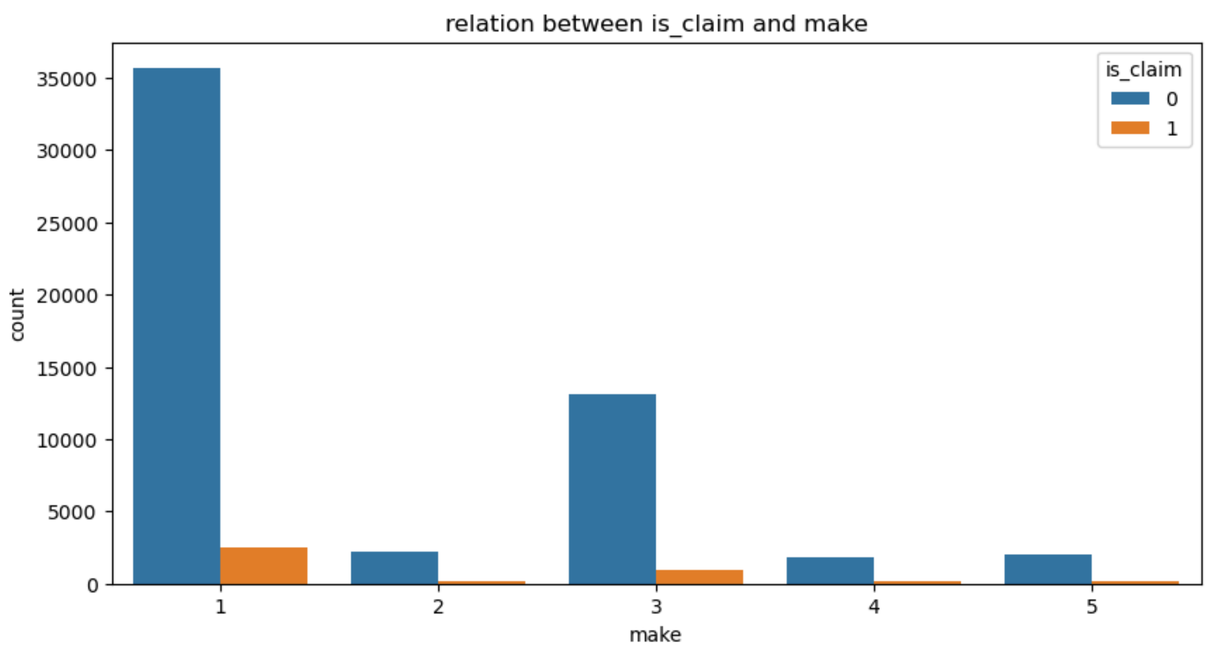
In [23]:

```
# Count plot with hue of target column
fig=plt.figure(figsize=(40,40))
plt.figure(figsize=(10,5))
sns.countplot(data = df, y= "engine_type", hue ="is_claim")
plt.title("relation between is_claim and engine_type")
plt.show()
```

```
<Figure size 4000x4000 with 0 Axes>
```



In [24]:
```
plt.figure(figsize=(10,5))
sns.countplot(data = df ,x= "make" , hue ="is_claim")
plt.title("relation between is_claim and make")
plt.show()
```



In [25]:
```
# pip install seaborn --upgrade ----> upgraded seaborn because 'hue' was not
```

Note7: (no. of rows in df = no. of unique values in policy_id. Need to check wheather to keep or drop (policy_id))

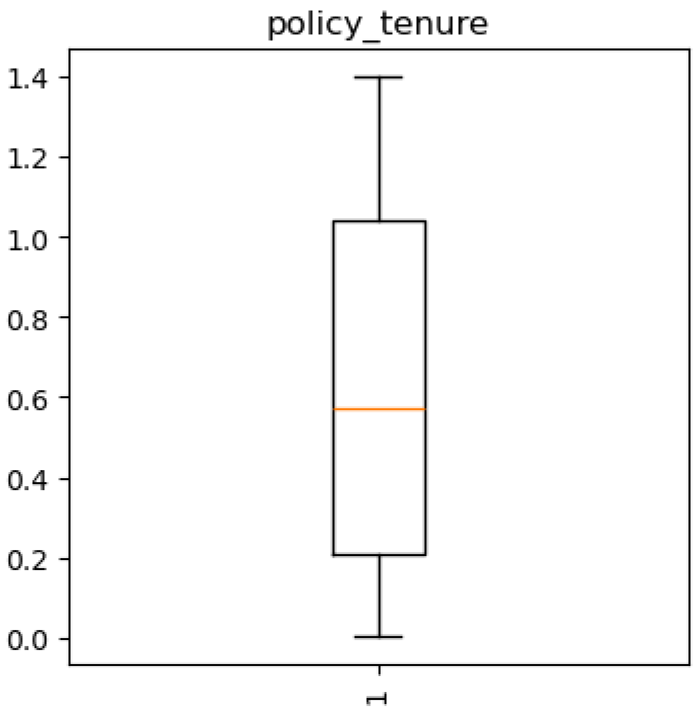
Numerical EDA

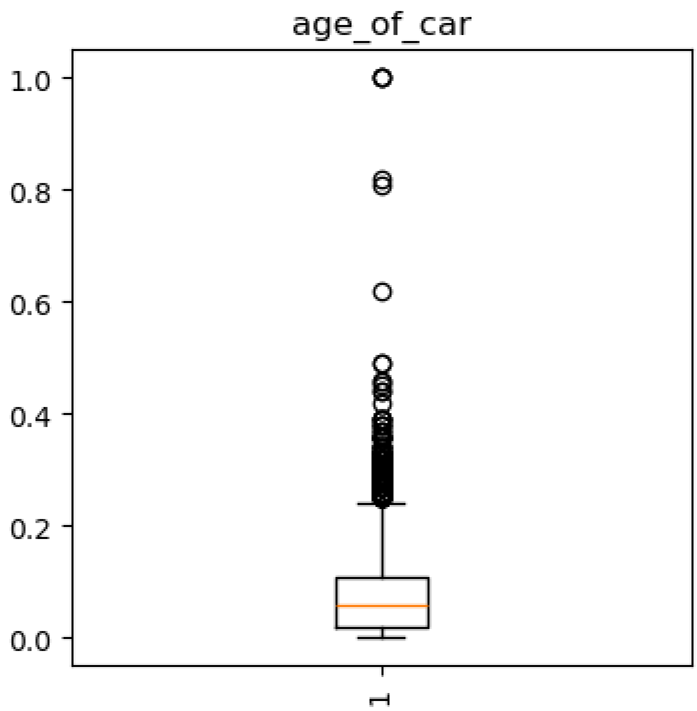
Distplot

In [26]:
```
# # Distplot is no longer supported in updated seaborn
# for i in numerical:
#  sns.distplot(df[i], bins=10)
#  plt.show()
```

Boxplot

In [27]:
```
# whole numerical col. visulizing in boxplot
for i in numerical:
   # if df[i].nunique()<1000:

       plt.figure(figsize=(4,4))
       plt.title(i)
       plt.boxplot(x=i, data=df)
       plt.xticks(rotation=90)
plt.show()
```





NOTE 8: age_of_car & age_of_policyholder has many outliers
population_density has less outliers

Five points summary of numerical columns

In [28]:
```
df[numerical].describe()
```

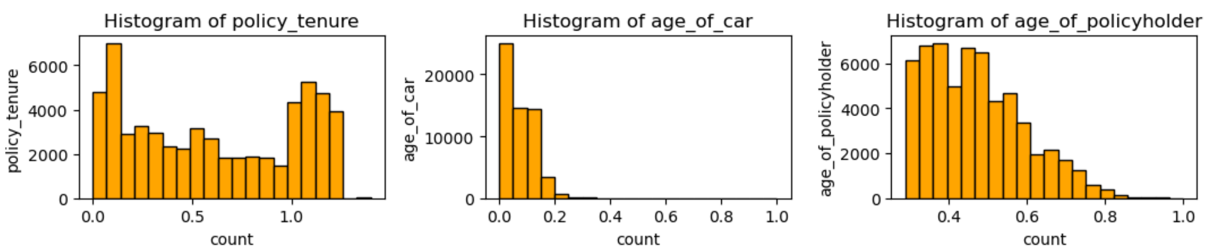
Out[28]:

	policy_tenure	age_of_car	age_of_policyholder
count	58592.000000	58592.000000	58592.000000
mean	0.611246	0.069424	0.469420
std	0.414156	0.056721	0.122886
min	0.002735	0.000000	0.288462
25%	0.210250	0.020000	0.365385
50%	0.573792	0.060000	0.451923
75%	1.039104	0.110000	0.548077
max	1.396641	1.000000	1.000000

Histogram

In [29]:

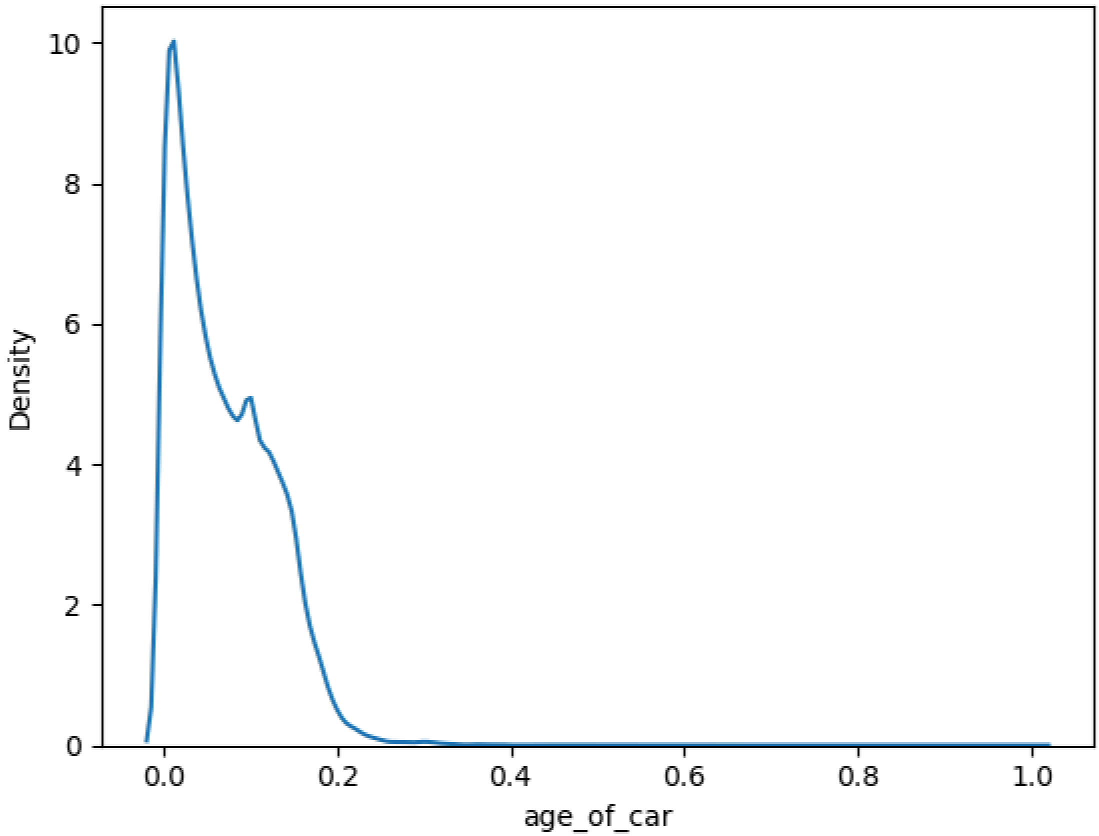
```
plt.figure(figsize=(18, 10))
for idx, i in enumerate(numerical):
    plt.subplot(5, 5, idx + 1)  # Adjust grid size (5x5) based on number of
    plt.hist(df[i], bins=20, color='orange', edgecolor='black')
    plt.title(f'Histogram of {i}')
    plt.xlabel('count')
    plt.ylabel(i)
plt.tight_layout()
plt.show()
```

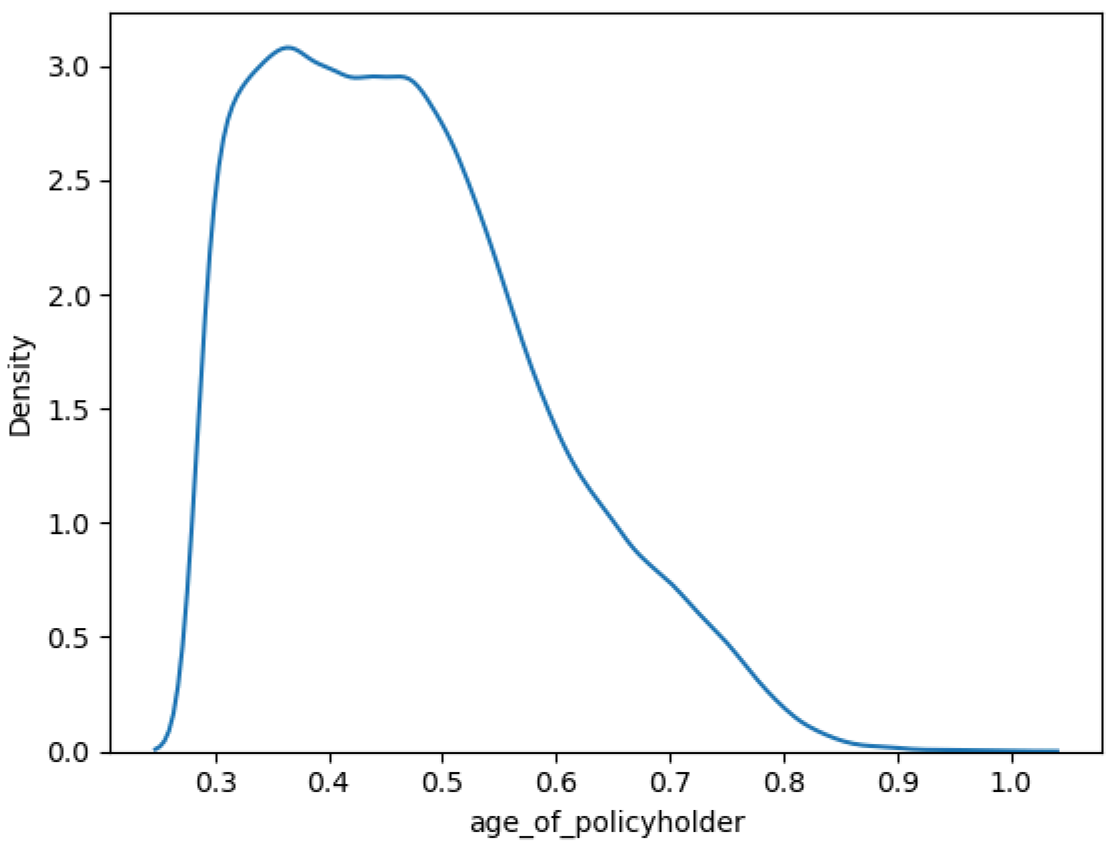


KDEplot

In [30]:

```
for i in numerical:
    if df[i].nunique() < 1000:
        sns.kdeplot(data=df[i])
        plt.show()
```





Correlation matrix of numerical column

In [31]:
```
df[numerical].corr()
```

Out[31]:

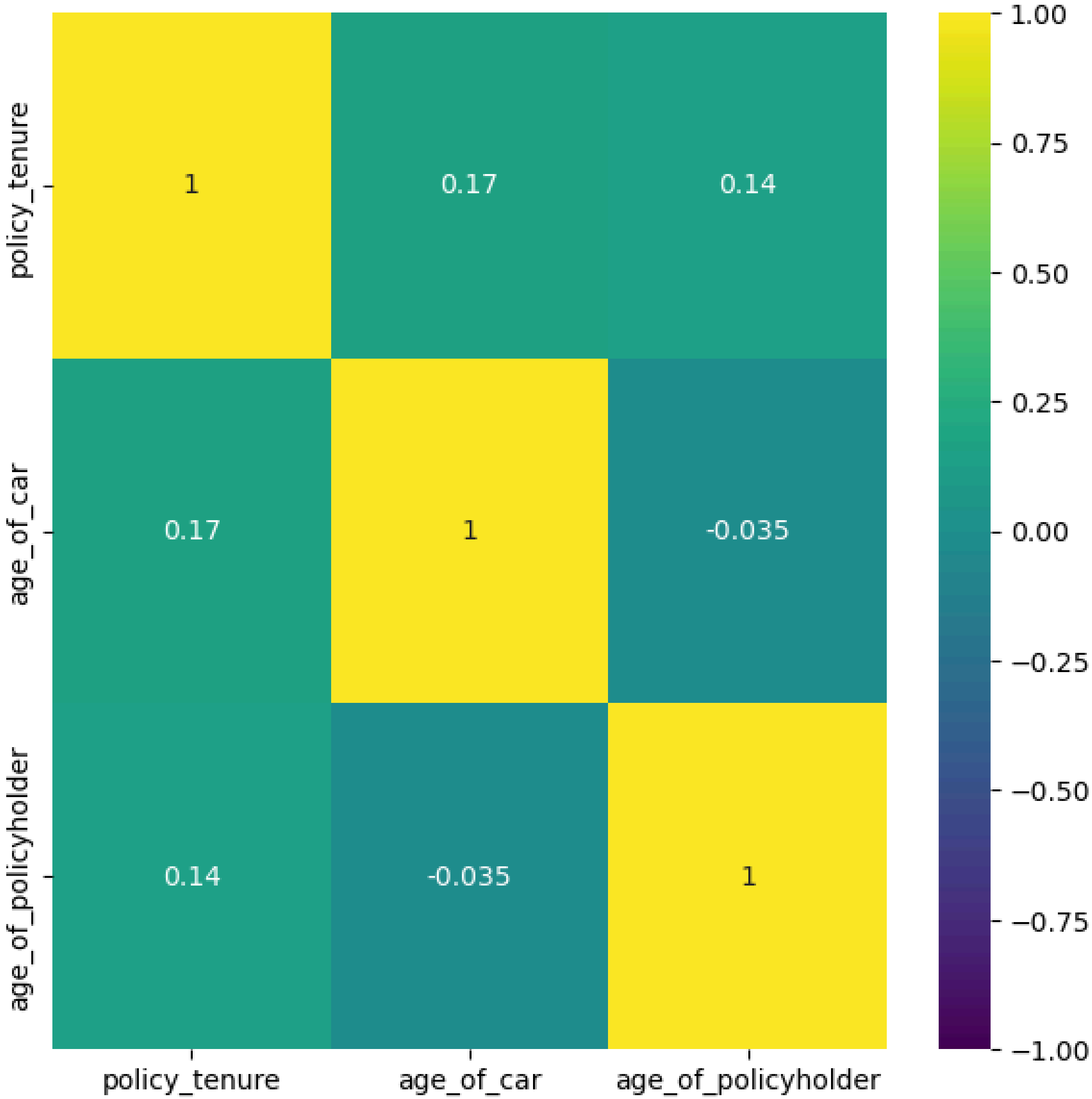
	policy_tenure	age_of_car	age_of_policyholder
policy_tenure	1.000000	0.166312	0.143676
age_of_car	0.166312	1.000000	-0.035427
age_of_policyholder	0.143676	-0.035427	1.000000

multicolinearity is very less

Heatmap of numerical columns

In [32]:

```
plt.figure(figsize=(7,7))
sns.heatmap(data=df[numerical].corr(),cmap='viridis',vmin=-1,vmax=1,annot=Tr
plt.show()
```



6 Outlier Analysis

In [33]:
```
numerical
```

Out[33]:
```
['policy_tenure', 'age_of_car', 'age_of_policyholder']
```

In [34]:
```
#### IQR Method To Handle Outlier
```

In [35]:
```
# Creating Upper limit & lower limit

for i in ['age_of_car','age_of_policyholder']:
    q1 = np.percentile(df[i], 25)
    q3 = np.percentile(df[i], 75)
    IQR=q3 - q1
    ul=q3+1.5*IQR
    ll=q1-1.5*IQR
    print(f"{i}:\nq1={q1},q3={q3},IQR={IQR}")
    print("upper limit",ul,"lower limit",ll)
    print("")
    df[i]=np.where(df[i]>ul,ul,df[i])
    df[i]=np.where(df[i]<ll,ll,df[i])
```

```
age_of_car:
q1=0.02,q3=0.11,IQR=0.09
upper limit 0.245 lower limit -0.115

age_of_policyholder:
q1=0.365384615384615,q3=0.548076923076923,IQR=0.18269230769230804
upper limit 0.822115384615385 lower limit 0.09134615384615291
```

In [36]:
```
# after handeling oulier there is no impact on five point summary
df[numerical].describe()
```

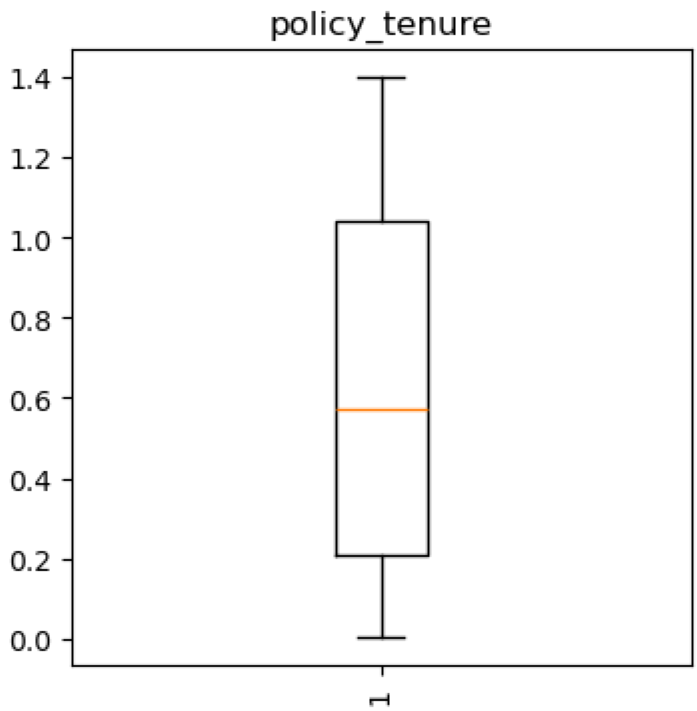
Out[36]:

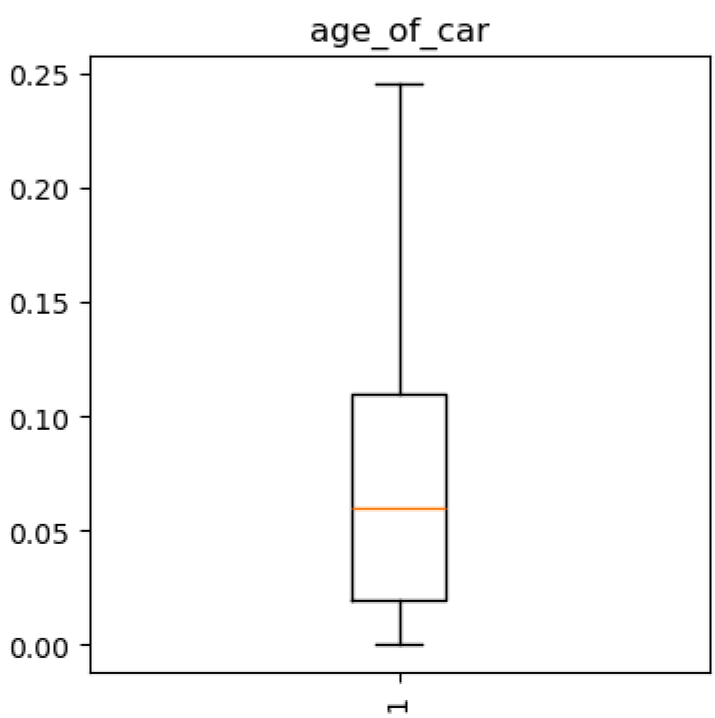
	policy_tenure	age_of_car	age_of_policyholder
count	58592.000000	58592.000000	58592.000000
mean	0.611246	0.069129	0.469279
std	0.414156	0.055227	0.122437
min	0.002735	0.000000	0.288462
25%	0.210250	0.020000	0.365385
50%	0.573792	0.060000	0.451923
75%	1.039104	0.110000	0.548077
max	1.396641	0.245000	0.822115

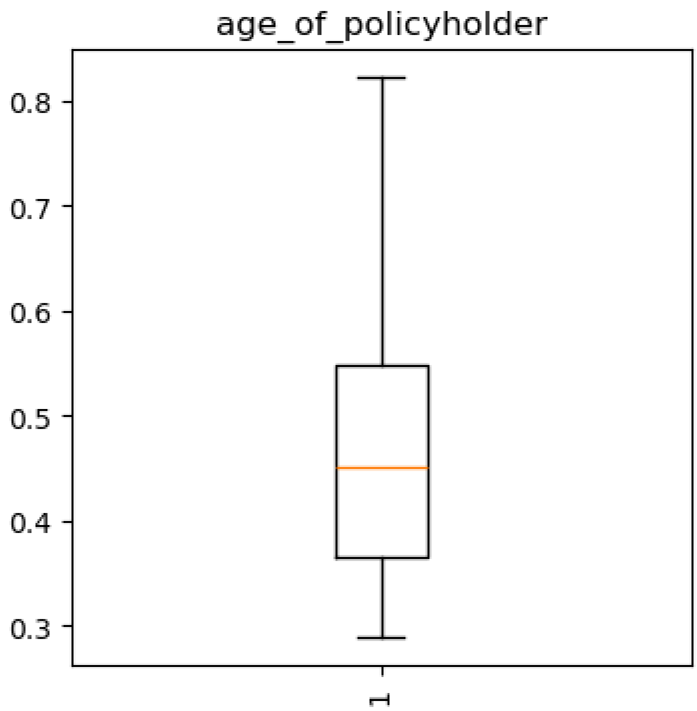
Boxplot After Outlier Handling

In [37]:
```
# whole numerical col. visulizing in boxplot
for i in numerical:
   # if df[i].nunique()<1000:
```

```
        plt.figure(figsize=(4,4))
        plt.title(i)
        plt.boxplot(x=i, data=df)
        plt.xticks(rotation=90)
plt.show()
```

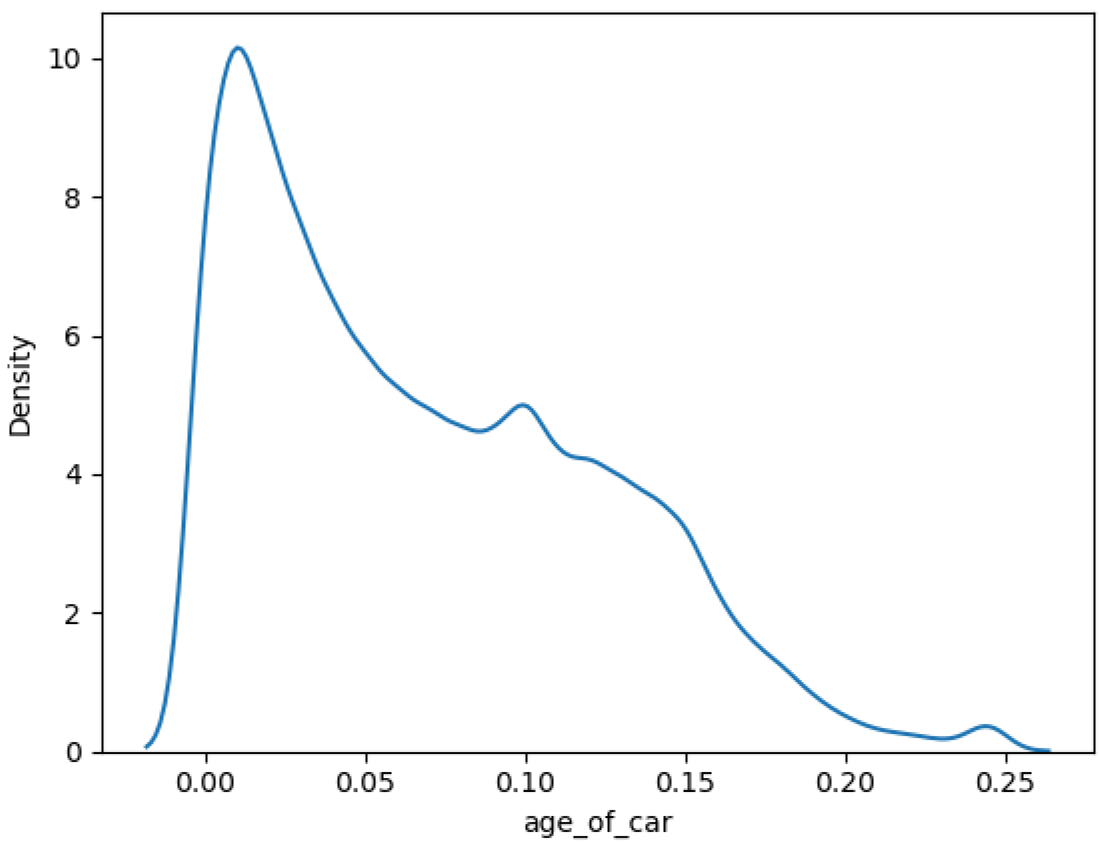


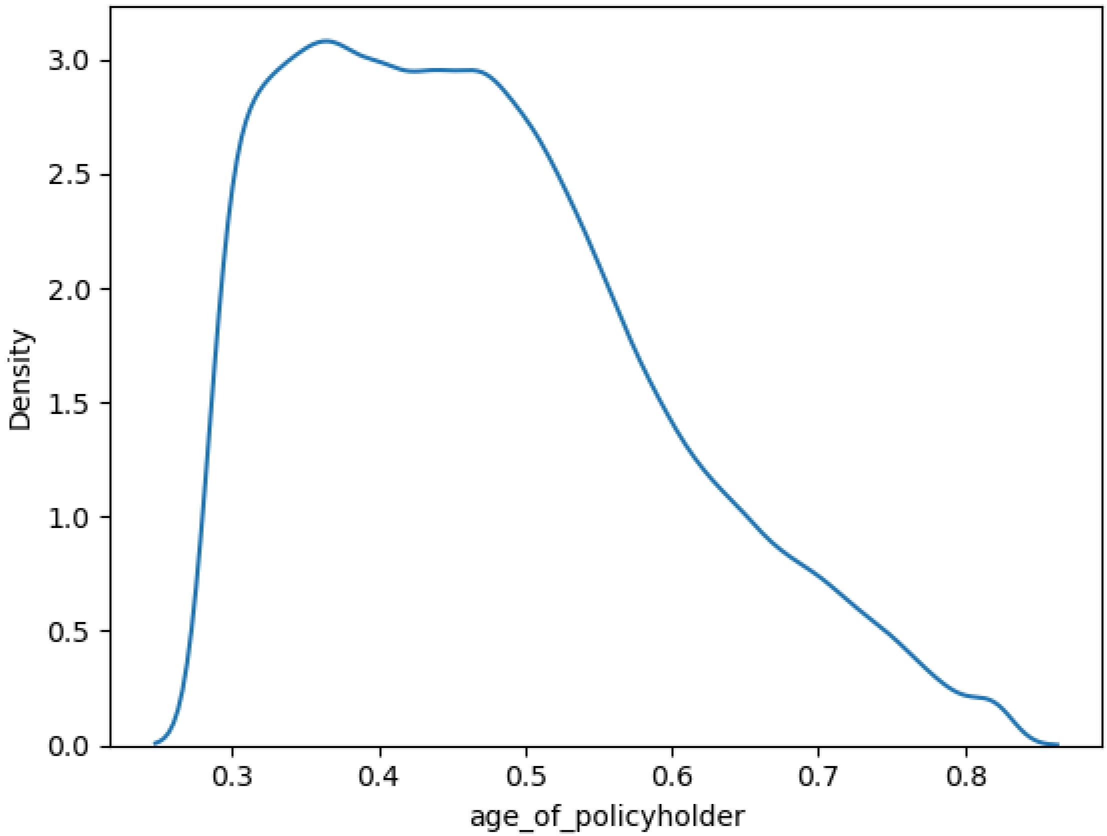




KDE Plot After Outlier Handling

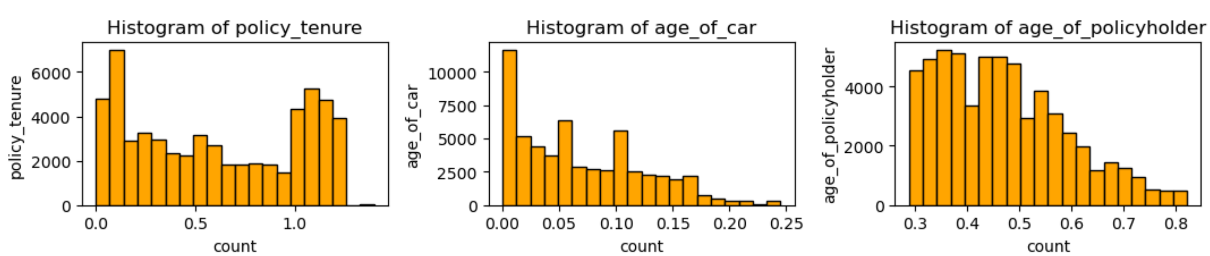
In [38]:
```
for i in numerical:
    if df[i].nunique() < 1000:
        sns.kdeplot(data=df[i])
        plt.show()
```





Histogram After Outlier Handling

In [39]:
```
plt.figure(figsize=(18, 10))
for idx, i in enumerate(numerical):
    plt.subplot(5, 5, idx + 1)  # Adjust grid size (5x5) based on number of
    plt.hist(df[i], bins=20, color='orange', edgecolor='black')
    plt.title(f'Histogram of {i}')
    plt.xlabel('count')
    plt.ylabel(i)
plt.tight_layout()
plt.show()
```



In [40]:
```
df.head()
```

Out[40]:

	policy_id	policy_tenure	age_of_car	age_of_policyholder	area_cluster	popu
0	ID00001	0.515874	0.05	0.644231	C1	
1	ID00002	0.672619	0.02	0.375000	C2	
2	ID00003	0.841110	0.02	0.384615	C3	
3	ID00004	0.900277	0.11	0.432692	C4	
4	ID00005	0.596403	0.11	0.634615	C5	

5 rows × 44 columns

In [41]:
```
df.tail()
```

Out[41]:

	policy_id	policy_tenure	age_of_car	age_of_policyholder	area_cluster
58587	ID58588	0.355089	0.13	0.644231	C8
58588	ID58589	1.199642	0.02	0.519231	C14
58589	ID58590	1.162273	0.05	0.451923	C5
58590	ID58591	1.236307	0.14	0.557692	C8
58591	ID58592	0.124429	0.02	0.442308	C8

5 rows × 44 columns

7 X&Y Train Test Split

In [42]:
```
x=df.drop(columns=['is_claim'],axis=1)
y=df['is_claim']
categorical.remove('is_claim') # as we are splitting it in x & y
```

In [43]:
```
df[numerical].corr()
```

Out[43]:

	policy_tenure	age_of_car	age_of_policyholder
policy_tenure	1.000000	0.168040	0.143546
age_of_car	0.168040	1.000000	-0.038852
age_of_policyholder	0.143546	-0.038852	1.000000

In [44]:
```
df.shape
```

Out[44]:
```
(58592, 44)
```

In [45]:
```
df['policy_id'].shape
```

Out[45]:
```
(58592,)
```

In [46]:
```
# In policy_id col. all values are unique. Hence we can drop the col.coz pol
df['policy_id'].nunique()
```

Out[46]: 58592

In [47]:
```
if df['policy_id'].shape[0]==df['policy_id'].nunique():
    x.drop(columns=['policy_id'],axis=1,inplace=True)
    print("'policy_id' dropped from x as all are unique values")
    categorical.remove('policy_id')
else:
    print("'policy_id' NOT dropped from x")
```

```
'policy_id' dropped from x as all are unique values
```

Label Encoding

In [48]:
```
from sklearn.preprocessing import LabelEncoder
```

In [49]:
```
le=LabelEncoder()
```

In [50]:
```
for i in categorical:
    x[i]=le.fit_transform(x[i])
```

In [51]:
```
x.head()
```

Out[51]:

	policy_tenure	age_of_car	age_of_policyholder	area_cluster	population_der
0	0.515874	0.05	0.644231	0	
1	0.672619	0.02	0.375000	11	
2	0.841110	0.02	0.384615	15	
3	0.900277	0.11	0.432692	16	
4	0.596403	0.11	0.634615	17	

5 rows × 42 columns

In [52]:
```
# checkin label encoding on categorical col.
for i in categorical:
    print(x[i].unique())
    print("")
```

```
[ 0 11 15 16 17 18 19 20 21  1  2  3  4  5  6  7  8  9 10 12 13 14]
[0 3 4 2 1 5]
[ 0  3  4  5  6  7  8  9 10  1  2]
[0 2 1]
[5 0 8 4 3 6 2 7 1]
[2 6 5 0 7 3 8 4 1]
[ 6  2  0  3  4  8  1  9 10  7  5]
[0 1]
[0 1]
[0 1]
[1 0]
[0 1]
[1 0]
[1 0]
[2 0 1]
[0 1]
[0 1]
[0 1]
[0 1]
[0 1]
[0 1]
[0 1]
[1 0]
[0 1]
[0 1]
[0 1]
[1 0]
[ 3 15  2 14 17  9  6  8 12 21  5 18  4  7  0 10 20 19 16 13  1 11]
```

```
[0 1 2 3 4]

[1 2 0]

[0 4 2 6 7 1 8 3 5]

[0 1]

[0 1]

[1 3 6 8 4 2 5 0 7]

[0 7 3 8 5 4 1 2 6]

[1 5 2 8 7 6 3 4 0 9]

[ 0  4  1  8  5  6  2  9  3 10  7]

[2 3 1 9 6 5 4 0 7 8]

[0 1 2 4 3]
```

In [53]:
```
y.unique()
```

Out[53]:
```
array([0, 1], dtype=int64)
```

In [54]:
```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
for i in numerical:
    x[i]=sc.fit_transform(x[[i]])
```

In [55]:
```
x.head()
```

Out[55]:

	policy_tenure	age_of_car	age_of_policyholder	area_cluster	population_de
0	-0.230283	-0.346366	1.428923	0	
1	0.148188	-0.889583	-0.770024	11	
2	0.555022	-0.889583	-0.691490	15	
3	0.697883	0.740067	-0.298821	16	
4	-0.035840	0.740067	1.350389	17	

5 rows × 42 columns

8 Model Building

1 LogisticRegression

logistic regression before balancing y

In [56]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42,test_size
```

In [57]:
```
for i in [x_train,x_test,y_train,y_test]:
    print(i.shape)
```

```
(46873, 42)
(11719, 42)
(46873,)
(11719,)
```

In [58]:
```
from sklearn.linear_model import LogisticRegression
lr=LogisticRegression()
```

In [59]:
```
from sklearn.metrics import accuracy_score,precision_score,recall_score
```

In [60]:
```
lr.fit(x_train,y_train)
y_train_predict_lr=lr.predict(x_train)
y_test_predict_lr=lr.predict(x_test)
```

```
C:\Users\Welcome\anaconda3\Lib\site-packages\sklearn\linear_model\_logistic.
py:458: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion
  n_iter_i = _check_optimize_result(
```

In [61]:
```
print("train")
print(f"accuracy_score{accuracy_score(y_train,y_train_predict_lr)},precision
```

```
train
accuracy_score0.9360399377040087,precision_score0.0,recall_score0.0
```

```
C:\Users\Welcome\anaconda3\Lib\site-packages\sklearn\metrics\_classificatio
n.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 due to no predicted samples. Use `zero_division` parameter to control th
is behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [62]:
```
print("test")
print(f"accuracy_score{accuracy_score(y_test,y_test_predict_lr)},precision_s
```

```
test
accuracy_score0.9360013653042069,precision_score0.0,recall_score0.0
```

```
C:\Users\Welcome\anaconda3\Lib\site-packages\sklearn\metrics\_classificatio
n.py:1344: UndefinedMetricWarning: Precision is ill-defined and being set to
0.0 due to no predicted samples. Use `zero_division` parameter to control th
is behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Imbalance Treatment Using Oversampling

In [63]:
```
from imblearn.over_sampling import RandomOverSampler
oversample = RandomOverSampler(sampling_strategy='minority')
x, y = oversample.fit_resample(x, y)
```

In [64]:
```
x.shape
```

Out[64]:
```
(109688, 42)
```

In [65]:
```
y.shape
```

Out[65]:
```
(109688,)
```

In [66]:
```
y.value_counts()
```

Out[66]:
```
is_claim
0    54844
1    54844
Name: count, dtype: int64
```

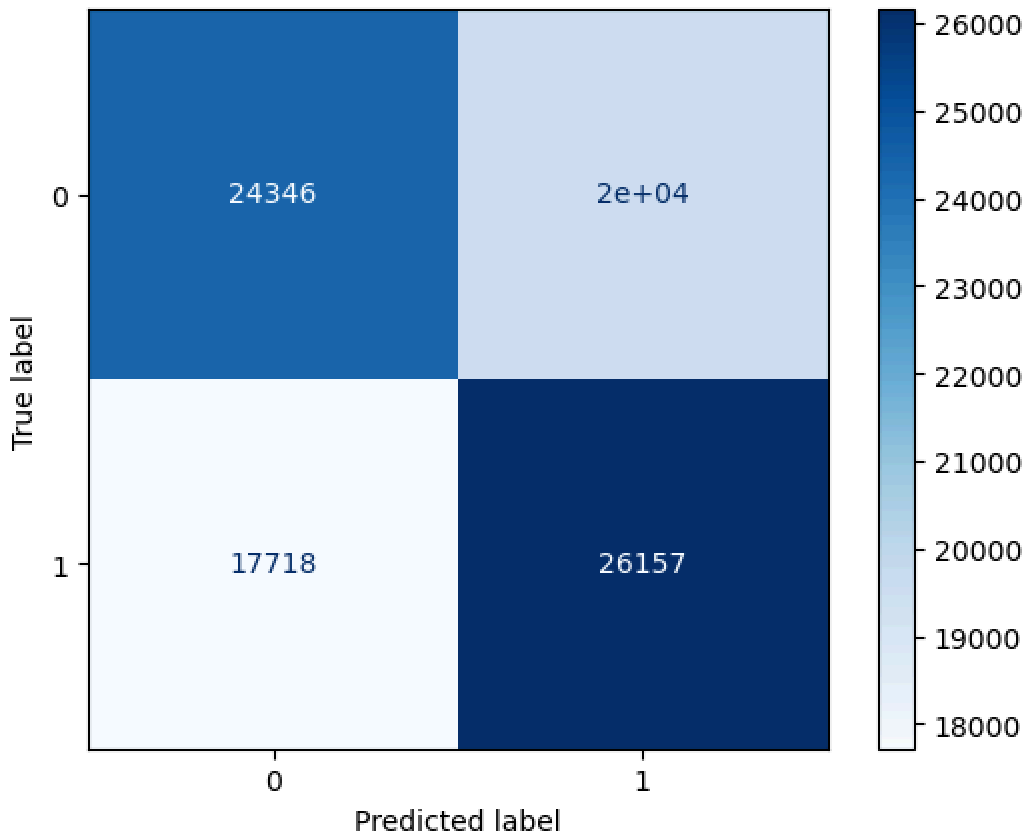
In [67]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42,test_size
```

logistic regression after Imbalance Treatment Using Oversampling

In [68]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=42,test_size
for i in [x_train,x_test,y_train,y_test]:
    print(i.shape)
lr.fit(x_train,y_train)
y_train_predict_lr=lr.predict(x_train)
y_test_predict_lr=lr.predict(x_test)
print("train")
print(f"accuracy_score={accuracy_score(y_train,y_train_predict_lr)},precisio
print("test")
print(f"accuracy_score={accuracy_score(y_test,y_test_predict_lr)},precision_
```

```
(87750, 42)
(21938, 42)
(87750,)
(21938,)
train
accuracy_score=0.5755327635327635,precision_score=0.5725386332793416,recall_
score=0.5961709401709402
test
accuracy_score=0.5809554198194913,precision_score=0.577963125548727,recall_s
core=0.6001458656212963
```

In [69]:
```
cm = confusion_matrix(y_train,y_train_predict_lr )
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.show()
```

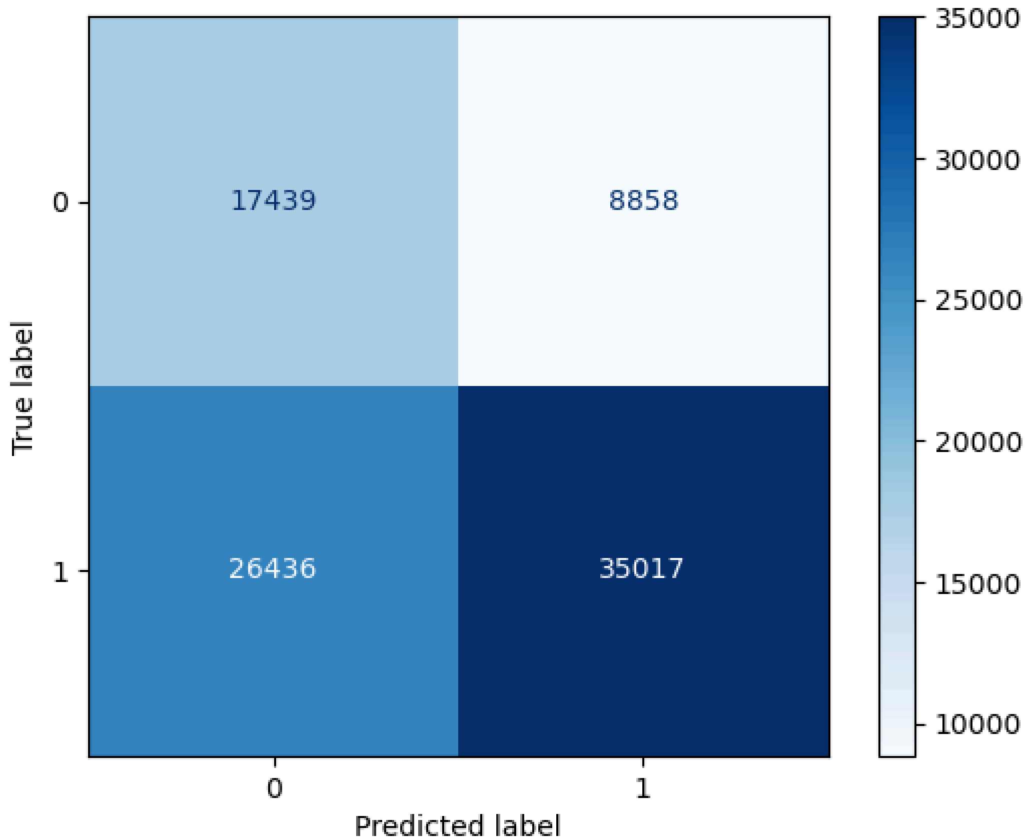


2 Decion Tree Model

In [70]:

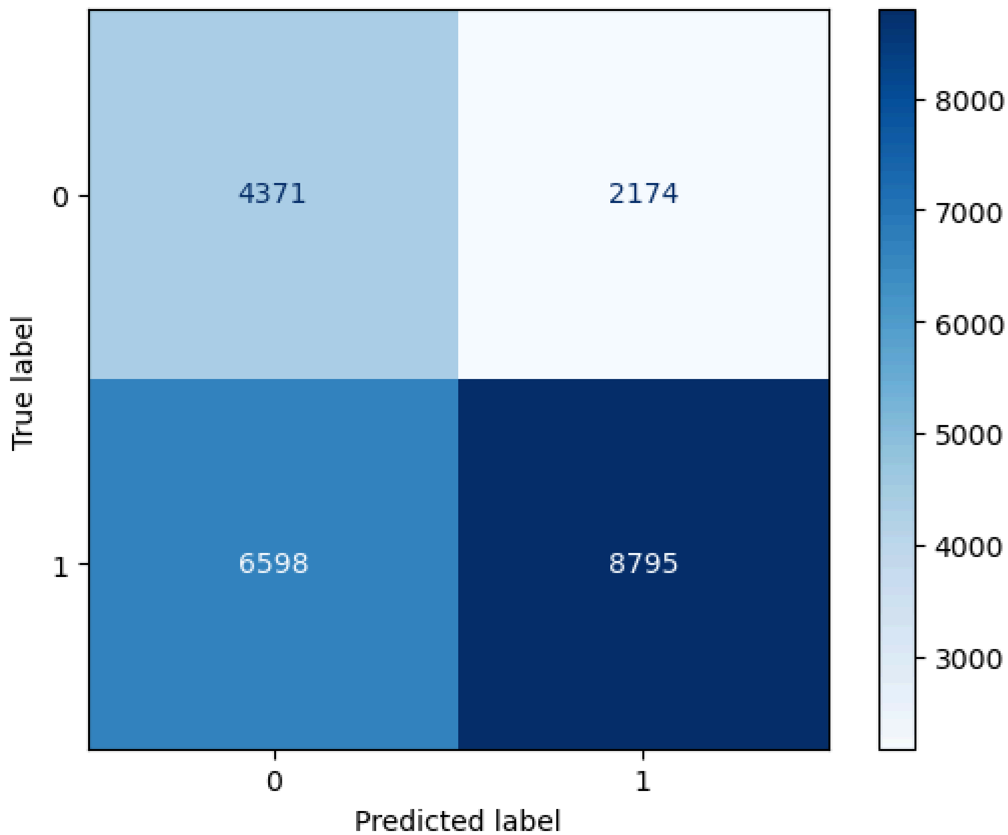
```
#Decision Tree
DTC =DecisionTreeClassifier(max_depth=2,criterion='entropy')
DTC.fit(x_train,y_train)

y_pred_train = DTC.predict(x_train)
cm = confusion_matrix(y_pred_train,y_train )

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

print("recall acc for train : " , recall_score(y_pred_train,y_train))
print("precision for train : " ,precision_score(y_pred_train,y_train))
print("f1_score for train : " ,f1_score(y_pred_train,y_train))
print("accuracy : " ,accuracy_score(y_pred_train,y_train))
disp.plot(cmap='Blues')

plt.show()
```

```
recall acc for train :  0.5698175841700162
precision for train :  0.7981082621082621
f1_score for train :  0.6649134133373843
accuracy :  0.5977891737891738
```



In [71]:
```
y_pred_test = DTC.predict(x_test)
cm = confusion_matrix(y_pred_test,y_test )

disp = ConfusionMatrixDisplay(confusion_matrix=cm)

print("recall acc for test : " , recall_score(y_pred_test,y_test))
print("precision for test : " ,  precision_score(y_pred_test,y_test))
print("f1_score for test : " ,   f1_score(y_pred_test,y_test))
print("accuracy : " ,accuracy_score(y_pred_test,y_test))

disp.plot(cmap='Blues')

plt.show()
```

```
recall acc for test :  0.5713636068342753
precision for test :  0.8018050870635427
f1_score for test :  0.6672483119641909
accuracy :  0.6001458656212963
```



3.1 Random Forest Model

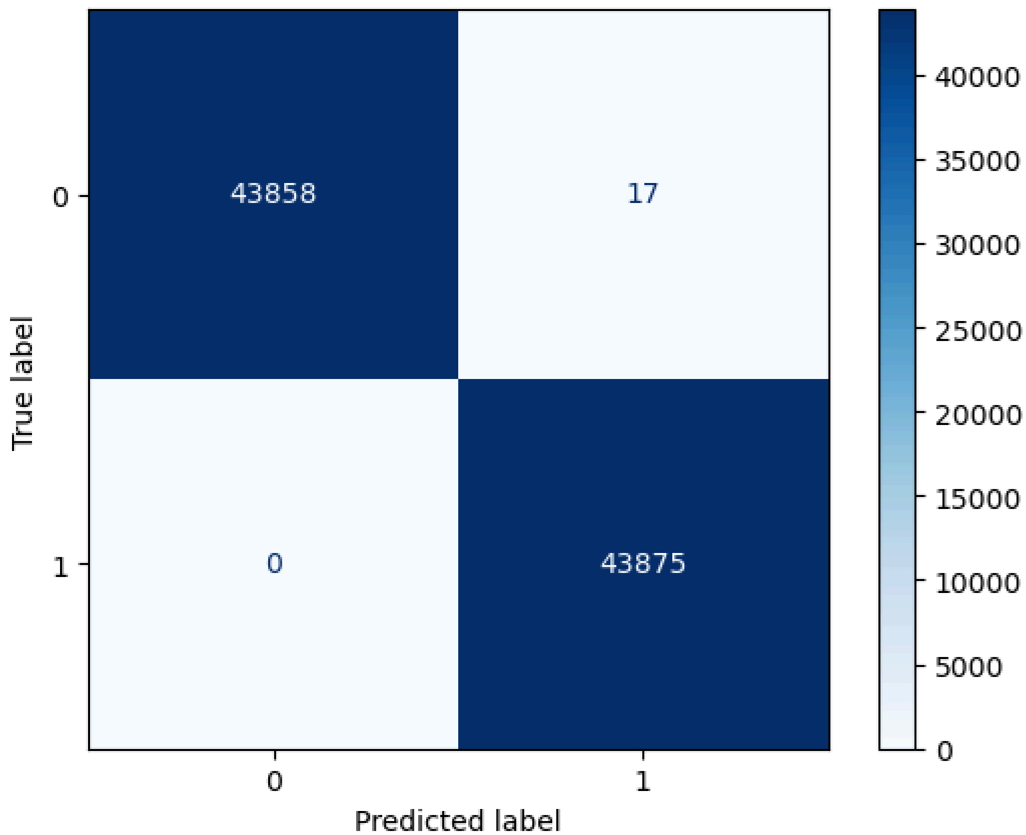
In [72]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [73]:
```
for i in range(1,25,5):
    rfc=RandomForestClassifier(n_estimators=i,criterion = 'entropy',max_feat
    rfc.fit(x_train,y_train)
    y_train_predict=rfc.predict(x_train)
    y_test_predict=rfc.predict(x_test)
    print("n_estimator =",i)
    print("train")
    print(f"accuracy_score={accuracy_score(y_train,y_train_predict)},precisi
    print("test")
    print(f"accuracy_score={accuracy_score(y_test,y_test_predict)},precision
    print("")
```

```
n_estimator = 1
train
accuracy_score=0.9787578347578347,precision_score=0.9596490076369286,recall_score=0.9995441595441595
test
accuracy_score=0.942109581547999,precision_score=0.896816954422715,recall_score=0.9991795058802079

n_estimator = 6
train
accuracy_score=0.9991225071225072,precision_score=0.9982480888241718,recall_score=1.0
test
accuracy_score=0.9832254535509162,precision_score=0.9675399135573785,recall_score=1.0

n_estimator = 11
train
accuracy_score=0.9992820512820513,precision_score=0.9985661614092585,recall_score=1.0
test
accuracy_score=0.9776643267389917,precision_score=0.9572388515577276,recall_score=1.0

n_estimator = 16
train
accuracy_score=0.9997720797720798,precision_score=0.9995443672400045,recall_score=1.0
test
accuracy_score=0.9829063725043304,precision_score=0.9669428772919605,recall_score=1.0

n_estimator = 21
train
accuracy_score=0.9998062678062678,precision_score=0.9996126856830402,recall_score=1.0
test
accuracy_score=0.9819491293645729,precision_score=0.965156181258249,recall_score=1.0
```

In [74]:

```
cm = confusion_matrix(y_train,y_train_predict )
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.show()
```



3.2 Finding best parameter for Random Forest using GridSearchCV

In [75]:
```
from sklearn.model_selection import GridSearchCV
param_dict={
    'n_estimators':[5,10,15,20,25],
    'max_depth':[3,5,7,9],
    'bootstrap':[True,False],
    'criterion':['gini','entropy'],
    'max_features':['auto', 'log2','sqrt']
}
gscv=GridSearchCV(RandomForestClassifier(),param_grid=param_dict,n_jobs=2,cv
gscv.fit(x_train,y_train)
gscv.best_params_
```

Out[75]:
```
{'bootstrap': False,
 'criterion': 'gini',
 'max_depth': 9,
 'max_features': 'log2',
 'n_estimators': 25}
```

In [76]:
```
param_dict={
    'n_estimators':[30,35,40,45,50],
    'max_depth':[9,11,13,15],
    'bootstrap':[True,False],
    'criterion':['gini','entropy'],
    'max_features':['auto', 'log2','sqrt']
}
```

```
gscv=GridSearchCV(RandomForestClassifier(),param_grid=param_dict,n_jobs=2,cv
gscv.fit(x_train,y_train)
gscv.best_params_
```

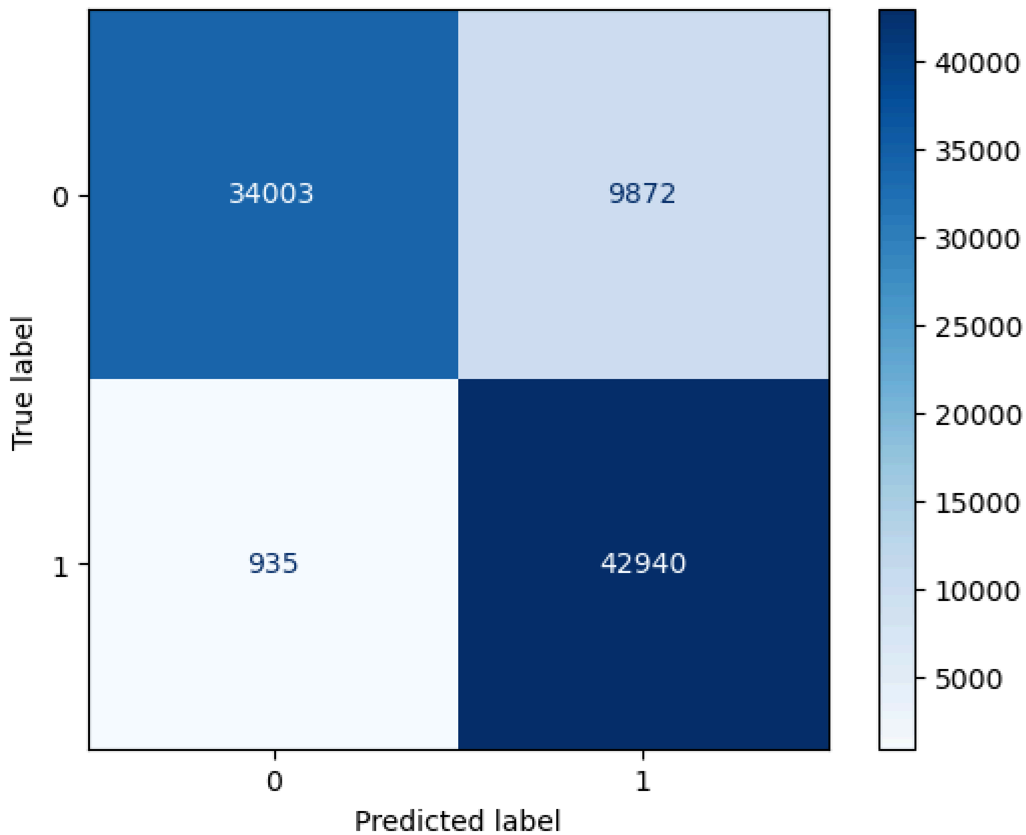
Out[76]:
```
{'bootstrap': False,
 'criterion': 'gini',
 'max_depth': 15,
 'max_features': 'sqrt',
 'n_estimators': 45}
```

3.3: Using best parameters on Random Forest

In [77]:
```
rfc=RandomForestClassifier(bootstrap=False,criterion = 'gini',max_depth=15,m
rfc.fit(x_train,y_train)
y_train_predict=rfc.predict(x_train)
y_test_predict=rfc.predict(x_test)
print("train")
print(f"accuracy_score={accuracy_score(y_train,y_train_predict)},precision_s
print("test")
print(f"accuracy_score={accuracy_score(y_test,y_test_predict)},precision_sco
print("")
```

```
train
accuracy_score=0.8768433048433049,precision_score=0.8130727864879194,recall_
score=0.9786894586894587
test
accuracy_score=0.85823684930258,precision_score=0.7899785993653605,recall_sc
ore=0.9759321724860972
```

In [78]:
```
cm = confusion_matrix(y_train,y_train_predict )
disp = ConfusionMatrixDisplay(confusion_matrix=cm)
disp.plot(cmap='Blues')
plt.show()
```



In [82]:
```
def calculate_acc(xtrain ,x_test ,y_train ,y_test):
    models =[LogisticRegression(),DecisionTreeClassifier(max_depth=2,criter
    data_frame = pd.DataFrame()
    acc =[]
    recall =[]
    precision =[]
    f1=[]
    for mod in models :
        model_ = mod
        model_.fit(x_train ,y_train)

        y_pred_test =model_.predict(x_test)
        acc.append(np.round(accuracy_score(y_pred_test,y_test),2))
        recall.append(np.round(recall_score(y_pred_test,y_test),2))
        precision.append(precision_score(y_pred_test,y_test))
        f1.append(f1_score(y_pred_test,y_test).round(2))


    tabel =pd.DataFrame(index=["LogisticRegression","DecisionTreeClassifier
                            columns=["acc" ,"recall","precision","F1"] )
    tabel["acc"]     = acc
    tabel["recall"] =recall
    tabel["precision"] = precision
    tabel["F1"] =f1
    return tabel
print("Accuracy Measurement")
calculate_acc(x_train, x_test, y_train, y_test)
```

```
Accuracy Measurement
```

Out[82]:

	acc	recall	precision	F1
LogisticRegression	0.58	0.58	0.600146	0.59
DecisionTreeClassifier	0.60	0.57	0.801805	0.67
RandomForestClassifier	0.86	0.79	0.975932	0.87

In [87]:
```
tabel=calculate_acc(x_train, x_test, y_train, y_test)
```

In [88]:
```
tabel['acc']
```

Out[88]:
```
LogisticRegression        0.58
DecisionTreeClassifier    0.60
RandomForestClassifier    0.86
Name: acc, dtype: float64
```

In [89]:
```
tabel['recall']
```

Out[89]:
```
LogisticRegression        0.58
DecisionTreeClassifier    0.57
RandomForestClassifier    0.79
Name: recall, dtype: float64
```

In [90]:
```
tabel['precision']
```

Out[90]:
```
LogisticRegression        0.600146
DecisionTreeClassifier    0.801805
RandomForestClassifier    0.975932
Name: precision, dtype: float64
```

In [92]:
```
tabel['F1']
```

Out[92]:
```
LogisticRegression        0.59
DecisionTreeClassifier    0.67
RandomForestClassifier    0.87
Name: F1, dtype: float64
```

In [95]:
```
tabel.index
```

Out[95]:
```
Index(['LogisticRegression', 'DecisionTreeClassifier',
       'RandomForestClassifier'],
      dtype='object')
```

In [101…
```
plt.bar(tabel.index, tabel['acc'], color='orange')
plt.title('Accuracy of Models')
plt.xlabel(tabel.index)
plt.ylabel('Accuracy')
plt.figure()
plt.show()
```


```
&lt;Figure size 640x480 with 0 Axes&gt;
```

END

In [ ]:

This notebook was converted with convert.ploomber.io