2023AIML544_E2EProj

July 14, 2024

FEATURE ENGINEERING End-to End PROJECT (30M)

AIML Certification Programme

0.1 Student Name and ID:

Mention your name and ID if done individually If done as a group,clearly mention the contribution from each group member qualitatively and as a precentage.

1. Sylaja M - 2023AIML544

0.2 Business Understanding (1M)

Students are expected to identify a regression problem of your choice. You have to detail the Business Understanding part of your problem under this heading which basically addresses the following questions.

1. What is the business problem that you are trying to solve? **Answer : To analyse the car and its features and predict the car resale price in India.**
2. What data do you need to answer the above problem? What are the different sources of data? **Answer : We need the car model name, car mileage, age of the car, engine capacity, car transmission type, car resale price etc for analysis.We need the list of cars which has been re-selled in a year. We can get this information from online.**

0.3 Data Requirements and Data Collection (3+1M)

In the initial data collection stage, data scientists identify and gather the available data resources. These can be in the form of structured, unstructured, and even semi-structured data relevant to the problem domain.

Identify the required data that fulfills the data requirements stage of the data science methodology Mention the source of the data.(Give the link if you have sourced it from any public data set) Briefly explain the data set identified .

0.3.1 Data link : https://www.kaggle.com/datasets/rahulmenon1758/car-resale-prices

This dataset contains Car resale prices all over the cities from India, updated as of August 2023. This dataset contains information in raw/unclean format.

[1162]:
```python
import pandas as pd
```

Import the above data and read it into a data frame

[1163]:
```python
df = pd.read_csv("car_resale_prices.csv")
```

Confirm the data has been correctly by displaying the first 5 and last 5 records.

[1164]:
```python
df.head()
```

[1164]:
```
   Unnamed: 0                    full_name resale_price registered_year  \
0           0  2017 Maruti Baleno 1.2 Alpha    5.45 Lakh            2017
1           1            2018 Tata Hexa XTA      10 Lakh            2018
2           2   2015 Maruti Swift Dzire VXI    4.50 Lakh            2015
3           3   2015 Maruti Swift Dzire VXI    4.50 Lakh            2015
4           4    2009 Hyundai i10 Magna 1.1    1.60 Lakh            2009

  engine_capacity             insurance transmission_type  kms_driven  \
0         1197 cc  Third Party insurance            Manual  40,000 Kms
1         2179 cc  Third Party insurance         Automatic  70,000 Kms
2         1197 cc  Third Party insurance            Manual  70,000 Kms
3         1197 cc  Third Party insurance            Manual  70,000 Kms
4         1086 cc  Third Party insurance            Manual  80,000 Kms

     owner_type fuel_type  max_power  seats     mileage  body_type  city
0   First Owner    Petrol    83.1bhp  5.000   21.4 kmpl  Hatchback  Agra
1   First Owner    Diesel  153.86bhp  7.000   17.6 kmpl        MUV  Agra
2  Second Owner    Petrol   83.14bhp  5.000  20.85 kmpl      Sedan  Agra
3  Second Owner    Petrol   83.14bhp  5.000  20.85 kmpl      Sedan  Agra
4   First Owner    Petrol   68.05bhp  5.000  19.81 kmpl  Hatchback  Agra
```

[1165]:
```python
df.tail()
```

[1165]:
```
       Unnamed: 0                              full_name  resale_price  \
17441       17441            2013 Honda Amaze VX i-Vtech     3.25 Lakh
17442       17442           2016 Toyota Camry 2.5 Hybrid    20.75 Lakh
17443       17443        2016 Toyota Corolla Altis GL MT     8.35 Lakh
17444       17444  2019 Hyundai Creta 1.6 CRDi AT SX Plus   13.95 Lakh
17445       17445            2017 Maruti Swift Dzire VDi     6.50 Lakh

      registered_year engine_capacity     insurance transmission_type  \
17441        Jul 2013         1198 cc  Comprehensive            Manual
17442        Jun 2016         2494 cc  Comprehensive         Automatic
17443        Jun 2016         1798 cc  Comprehensive            Manual
17444        Jun 2019         1582 cc  Comprehensive         Automatic
17445        Jun 2017         1248 cc  Comprehensive            Manual

       kms_driven    owner_type fuel_type  max_power  seats     mileage  \
```

```
17441  89,000 Kms  Second Owner    Petrol     86.7bhp  5.000      18 kmpl
17442  68,000 Kms   First Owner    Petrol    157.7bhp  5.000  19.16 kmpl
17443  81,000 Kms   First Owner    Petrol   138.03bhp  5.000  14.28 kmpl
17444  20,000 Kms   First Owner    Diesel    126.2bhp  5.000  17.01 kmpl
17445  32,000 Kms   First Owner    Diesel     73.9bhp  5.000   19.3 kmpl

      body_type   city
17441     Sedan  Delhi
17442     Sedan  Delhi
17443     Sedan  Delhi
17444       SUV  Delhi
17445     Sedan  Delhi
```

Get the dimensions of the dataframe.

[1166]:
```
df.shape
```

[1166]:
```
(17446, 15)
```

The dataset has 17446 rows and 15 columns

Display the description and statistical summary of the data.

0.3.2 Dataset description

Dataset Name: Car resale prices

Source : https://www.kaggle.com/datasets/rahulmenon1758/car-resale-prices

Description : This dataset contains Car resale prices all over the cities from India in 2023.

Variables: 1. **Unnamed:0** : Index column this will be dropped 2. **full_name** : Name of the car along with model 3. **resale_price** : Resale price of the car 4. **registered_year** : Year the car was registered 5. **engine_capacity** : Engine Displacement of car (cc) 6. **insurance** : Type of insurance made available for the car (if any) 7. **transmission_type** : Transmission type of the car 8. **kms_driven** : Total kilometers the car was driven for 9. **owner_type** : Number of owners who previously owned the car 10. **fuel_type** : Type of fuel the car uses 11. **max_power** : Maximum power of the car (bhp) 12. **seats** : Number of seats the car has 13. **mileage** : Mileage of the car 14. **body_type** : Body configuration of the car 15. **city** : City in India the car is sold in

Data Format : CSV file with 17446 records and 15 columns.

[1167]:
```
df.describe()
```

[1167]:
```
       Unnamed: 0     seats
count   17446.000 17436.000
mean     8722.500     5.205
std      5036.371     0.669
min         0.000     2.000
25%      4361.250     5.000
```

```
50%        8722.500      5.000
75%       13083.750      5.000
max       17445.000     14.000
```

The statistical summary is given only for numerical attributes. Many of the attributes are considered as objects. So lets look at this statistical summary once we convert the columns to appropriate datatypes

Display the columns and their respective data types.

[1168]:
```
df.dtypes
```

[1168]:
```
Unnamed: 0              int64
full_name              object
resale_price           object
registered_year        object
engine_capacity        object
insurance              object
transmission_type      object
kms_driven             object
owner_type             object
fuel_type              object
max_power              object
seats                 float64
mileage                object
body_type              object
city                   object
dtype: object
```

Convert the columns to appropriate data types

Lets take attributes one by one and we will convert it into appropriate datatypes

The first attribute is Unnamed:0 is nothing but the serial number/index.This attribute is not needed so lets just remove it.

[1169]:
```
df.drop(columns=['Unnamed: 0'], inplace=True)
```

The second attribute is full_name. This column has the full car name so this can be as object type itself.

The third attribute is resale_price. This column has resale price in lakhs. We can convert this into a numeric attribute.

[1170]:
```
# Function to convert currency strings to numerical values
def currency_to_numeric(currency_string):
    # Remove ' ' and ',' and split by space to separate amount and unit
    parts = currency_string.replace(' ', '').replace(',', '').split()
    amount = float(parts[0])  # Convert amount to float
    if len(parts) == 1:
        return amount
```

```python
    unit = parts[1].lower()   # Get the unit (Lakh, Crore, Thousand)

    # Convert based on unit
    if unit == 'lakh':
        return amount * 100000
    elif unit == 'crore':
        return amount * 10000000
    elif unit == 'thousand':
        return amount * 1000
    else:
        return None  # Handle unrecognized units or errors
```

[1171]:
```python
df['resale_price'] = df['resale_price'].apply(currency_to_numeric)
```

The forth attribute is registered_year. This column has the car registeration year. We need only the year so we will convert this also to a date numerical attribute.

[1172]:
```python
# Convert 'Date' column to datetime format
df['registered_year'] = pd.to_datetime(df['registered_year'],format='mixed')

# Extract the year from the datetime format
df['registered_year'] = df['registered_year'].dt.year.astype('Int64')
```

The fifth attribute is engine_capacity. This column has the car engine capacity in cc. We can convert this also to a numerical attribute by removing the 'cc'

[1173]:
```python
df['engine_capacity'] = df['engine_capacity'].str.replace(' cc', '').
 ↪astype('Int64')
```

The sixth attribute is insurance. This column has the car insurance type so this can be in object type itself

The seventh attribute is transmission_type. This column has the car transmission type such as automatic or manual so this can be an object type itself

The eigth attribute is kms_driven. This column has the total number of kilometer the car has been drove till now. We can convert this also to a numerical attribute by removing the units

[1174]:
```python
df['kms_driven'] = df['kms_driven'].str.replace(' Kms', '').str.replace(',',␣
 ↪'').astype('Int64')
```

The ninth attribute is owner_type. This column has the car owner type such as first owner, second owner etc so this can be an object type itself

The tenth attribute is fuel_type. This column has the car fuel type such as petrol, diesel etc so this can be an object type itself

The eleventh attribute is max_power. This column has the maximum power of the car. This has to be converted to a numerical data type

[1175]:
```
# Function to convert car power to brake horse power(bhp) values
def power_to_bhp(car_power):
    if isinstance(car_power, str) :
        if "bhp".lower() in car_power.lower():
            power_in_bhp = car_power.lower().split('bhp')[0].strip()
            if "kw".lower() in power_in_bhp.lower() :
                return float(power_in_bhp.lower().split('kw')[0].strip()) * 1.
↪341
            else :
                return float(power_in_bhp)
        elif "hp".lower() in car_power.lower():
            return float(car_power.lower().split('hp')[0].strip())
        elif "ps".lower() in car_power.lower():
            return float(car_power.lower().split('ps')[0].strip()) * 0.9863
        elif "kw".lower() in car_power.lower():
            return float(car_power.lower().split('kw')[0].strip()) * 1.341
        else :
            return car_power
    else :
        return car_power
```

[1176]:
```
df['max_power'] = df['max_power'].apply(power_to_bhp)
```

The max_power has been converted to bhp but there are still some data inconsistency issue which we can handle in the later section

The twelfth attribute is seats. This column has the number of seats in the car so this can be converted to an integer

[1177]:
```
df['seats'] = df['seats'].round().astype('Int64')
```

The thirteenth attribute is mileage. This column has the mileage that the car provides. We can convert this also to a numerical attribute

[1178]:
```
# Function to convert car mileage to kmpl values
def mileage_to_kmpl(mileage):
    if isinstance(mileage, str) :
        if "kmpl".lower() in mileage.lower():
            return float(mileage.lower().split('kmpl')[0].strip())
        elif "km/kg".lower() in mileage.lower():
            return float(mileage.lower().split('km/kg')[0].strip()) * 1.5
        else :
            return float(mileage)
    else :
        return float(mileage)
```

[1179]:
```
df['mileage'] = df['mileage'].apply(mileage_to_kmpl)
```

The forteenth attribute is body_type. This column gives info about the body type of

the car so this can be in object type itself

The fifteenth attribute is city. This column gives info about the city in which the car is sold in so it can be in object type itself

[1180]:
```
df.dtypes
```

[1180]:
```
full_name             object
resale_price         float64
registered_year        Int64
engine_capacity        Int64
insurance             object
transmission_type     object
kms_driven             Int64
owner_type            object
fuel_type             object
max_power             object
seats                  Int64
mileage              float64
body_type             object
city                  object
dtype: object
```

0.3.3 Note : max_power will also be converted into float once the data inconsistencies is handled in the further sections

[1181]:
```
df.describe()
```

[1181]:
```
       resale_price  registered_year  engine_capacity  kms_driven     seats  \
count     17446.000        17377.000        17432.000   17443.000 17436.000
mean     882326.099         2016.416         1423.135   58628.224     5.205
std     1093610.835            3.661          474.684   64264.640     0.669
min       28000.000         2002.000            0.000     286.000     2.000
25%      379000.000         2014.000         1197.000   31922.000     5.000
50%      585000.000         2017.000         1248.000   54817.000     5.000
75%      913000.000         2019.000         1498.000   79913.000     5.000
max    22500000.000         2023.000         5998.000 6275000.000    14.000

        mileage
count 16938.000
mean     19.525
std       4.921
min       6.700
25%      17.000
50%      18.900
75%      21.630
max     140.000
```

Now we can see the statistical summary for all the numerical attributes after conversion

Write your observations from the above. When a dataset is generated by collecting data from multiple sources, all values in a column will not be in the same format. It is always recommended to bring all the values in the same format so that it will be easy for analysis. That is what we have done in the above sections.

0.3.4 Check for Data Quality Issues (1.5M)

- duplicate data
- missing data
- data inconsistencies

[1182]:
```python
duplicate_rows = df[df.duplicated()]
print(duplicate_rows)
```

```
                                     full_name  resale_price  \
3                    2015 Maruti Swift Dzire VXI    450000.000
25                 2011 Mitsubishi Outlander 2.4    397000.000
34                   2013 Mahindra XUV500 W6 2WD    350000.000
36          2013 Tata New Safari DICOR 2.2 EX 4x2    296000.000
47                2011 Tata Manza Aqua Quadrajet    150000.000
…                                            …           …
16779  2010 Tata Manza Aura Plus Quadrajet BS IV    245000.000
17117                  1998 Maruti Omni 5 Str STD     58000.000
17237                 2022 MG Astor Sharp CVT BSVI   1575000.000
17255                  2013 Hyundai Elantra SX AT    575000.000
17259     2018 Skoda Octavia 1.4 TSI MT Ambition   1350000.000

       registered_year  engine_capacity              insurance  \
3                 2015             1197  Third Party insurance
25                2011             2360  Third Party insurance
34                2013             2179  Third Party insurance
36                2013             2179  Third Party insurance
47                2011             1248  Third Party insurance
…                  …               …                    …
16779             2010             1248  Third Party insurance
17117             <NA>              796  Third Party insurance
17237             2022             1498          Comprehensive
17255             2013             1797  Third Party insurance
17259             2018             1395          Comprehensive

      transmission_type  kms_driven    owner_type fuel_type max_power  seats  \
3                Manual       70000  Second Owner    Petrol    83.140      5
25            Automatic       80000  Second Owner    Petrol   167.671      5
34               Manual      100000   Third Owner    Diesel   140.000      7
36               Manual       90000  Second Owner    Diesel   138.100      7
```

```
47                Manual      150000   First Owner    Diesel    88.767      5
…                   …          …          …          …         …       …
16779             Manual       80000  Second Owner    Diesel    88.767      5
17117             Manual       80000   Fifth Owner    Petrol    35.000      5
17237          Automatic       12000   First Owner    Petrol   108.490      5
17255          Automatic       76000   First Owner    Petrol   147.500      5
17259             Manual       65000   First Owner    Petrol   148.000      5

       mileage body_type       city
3       20.850     Sedan       Agra
25      11.300       SUV       Agra
34      15.100       SUV       Agra
36      11.570       SUV       Agra
47      18.800     Sedan       Agra
…         …        …        …
16779   19.000     Sedan  Hyderabad
17117   14.000  Minivans  Bangalore
17237   14.820       SUV      Delhi
17255   14.500     Sedan      Delhi
17259   16.700     Sedan      Delhi

[212 rows x 14 columns]
```

212 rows are duplicated

[1183]:
```
df.isnull().sum()
```

[1183]:
```
full_name              0
resale_price           0
registered_year       69
engine_capacity       14
insurance              7
transmission_type      0
kms_driven             3
owner_type            45
fuel_type              0
max_power            102
seats                 10
mileage              508
body_type              0
city                   0
dtype: int64
```

- **registered_year has 69 missing values**
- **engine_capacity has 14 missing values**
- **insurance has 7 missing values**
- **kms_driven has 3 missing values**
- **owner_type has 45 missing values**
- **max_power has 102 missing values**

- **seats has 10 missing values**
- **mileage has 508 missing values**

Data inconsistencies max_power should contain only numeric values. We can check this using a regex match.

[1184]:
```
import re

# Define regex pattern for float-like values
pattern = r'^$|[-+]?[0-9]*\.?[0-9]+$'

# Function to filter non-float values
def filter_float_values(value):
    if pd.isna(value) or re.match(pattern, str(value)):
        return False
    else:
        return True

# Apply filter and create new DataFrame with only float values
filtered_df = df[df['max_power'].apply(filter_float_values)]
print("Inconsistent Powers:")
print(filtered_df['max_power'])
```

```
Inconsistent Powers:
1020                90(66)
2279                90(66)
3018                90(66)
3059          132/4000-6000
3127                90(66)
3504                90(66)
6124                90(66)
7510                90(66)
7864      165 [224] at 3800
8350      165 [224] at 3800
8386          66(90) / 4000
8820          66(90) / 4000
8943      165 [224] at 3800
9058          110(150)/5700
11832     165 [224] at 3800
12779               90(66)
14302               90(66)
14585     165 [224] at 3800
14759               90(66)
16097     165 [224] at 3800
16822     165 [224] at 3800
Name: max_power, dtype: object
```

The max_power has some inconsistent values as shown above. Here we have the power in various formats. We need to correct this

0.3.5 Identify outliers

Lets define a function to identify outliers in numerical attributes

[1185]:
```
import numpy as np
def identifyOutliers(df, featureVariable) :
    print("Old Shape: ", df.shape)
    # IQR
    # Calculate the upper and lower limits
    Q1 = df[featureVariable].quantile(0.25)
    Q3 = df[featureVariable].quantile(0.75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5*IQR
    upper = Q3 + 1.5*IQR

    upper_array = np.array(df[featureVariable] >= upper)
    lower_array = np.array(df[featureVariable] <= lower)

    print("Upper limit: ", upper)
    print("No. of datapoints greater than upper limit: ", upper_array.sum())

    print("Lower limit: ", lower)
    print("No. of datapoints less than lower limit: ", lower_array.sum())
```

[1186]:
```
identifyOutliers(df, "registered_year")
```

```
Old Shape:  (17446, 14)
Upper limit:  2026.5
No. of datapoints greater than upper limit:  <NA>
Lower limit:  2006.5
No. of datapoints less than lower limit:  <NA>
```

There are no outliers in registered_year column

[1187]:
```
identifyOutliers(df, "engine_capacity")
```

```
Old Shape:  (17446, 14)
Upper limit:  1949.5
No. of datapoints greater than upper limit:  <NA>
Lower limit:  745.5
No. of datapoints less than lower limit:  <NA>
```

There are no outliers in engine_capacity column

[1188]:
```
identifyOutliers(df, "kms_driven")
```

```
Old Shape:  (17446, 14)
Upper limit:  151899.5
No. of datapoints greater than upper limit:  <NA>
Lower limit:  -40064.5
No. of datapoints less than lower limit:  <NA>
```

There are no outliers in kms_driven column

[1189]:
```
identifyOutliers(df, "seats")
```

```
Old Shape:  (17446, 14)
Upper limit:  5.0
No. of datapoints greater than upper limit:  <NA>
Lower limit:  5.0
No. of datapoints less than lower limit:  <NA>
```

There are no outliers in seats column

[1190]:
```
identifyOutliers(df, "mileage")
```

```
Old Shape:  (17446, 14)
Upper limit:  28.574999999999996
No. of datapoints greater than upper limit:  300
Lower limit:  10.055000000000001
No. of datapoints less than lower limit:  66
```

There are outliers in mileage column. This has to be removed in the next steps.

Now lets calculate the outliers for categorical attributes. Categories that occur very infrequently compared to others in the dataset may be considered outliers.

[1191]:
```
value_counts = df['full_name'].value_counts()
print(value_counts)
```

```
full_name
2016 Hyundai Grand i10 Sportz      51
2017 Maruti Baleno 1.2 Delta       41
2015 Maruti Swift VXI              38
2016 Maruti Baleno 1.2 Delta       35
2015 Hyundai Grand i10 Sportz      35
                                   ..
2021 Tata Nexon XMA AMT S BSVI      1
2015 Honda City i DTEC S            1
2008 Honda CR-V RVi MT              1
2013 BMW 3 Series 320d Prestige     1
2017 Maruti Swift Dzire VDi         1
Name: count, Length: 6923, dtype: int64
```

For model names we cannot consider the rare frequency categories as outlier because here it means in last year only one car of this model has been resaled. It is not a outliers

[1192]:
```
value_counts = df['insurance'].value_counts()
print(value_counts)
```

```
insurance
Third Party insurance    7559
Comprehensive            6414
Third Party              1973
```

```
Zero Dep                    834
Not Available               651
1                             5
2                             3
Name: count, dtype: int64
```

In india there are only 5 car insurance types. 1. Third party insurance 2. OD insurance 3. Personal accident cover 4. Zero depreciation 5. Comprehensive

Here we have something as 1,2 which doesnot make sense also that category is very rare. So this is an outlier and this can be removed.

[1193]:
```python
value_counts = df['transmission_type'].value_counts()
print(value_counts)
```

```
transmission_type
Manual       12541
Automatic     4905
Name: count, dtype: int64
```

No rare categories thus no outliers in transmission_type

[1194]:
```python
value_counts = df['owner_type'].value_counts()
print(value_counts)
```

```
owner_type
First Owner     12293
Second Owner     4150
Third Owner       780
Fourth Owner      127
Fifth Owner        51
Name: count, dtype: int64
```

No rare categories thus no outliers in owner_type

[1195]:
```python
value_counts = df['fuel_type'].value_counts()
print(value_counts)
```

```
fuel_type
Petrol      11336
Diesel       5516
CNG           504
Electric       61
LPG            29
Name: count, dtype: int64
```

No rare categories thus no outliers in fuel_type

[1196]:
```python
value_counts = df['body_type'].value_counts()
print(value_counts)
```

```
body_type
Hatchback        7343
Sedan            4781
SUV              4406
MUV               759
Minivans           65
Maruti             19
Pickup             13
Coupe              10
Cars                8
Tata                7
Mercedes-Benz       6
Mahindra            4
Chevrolet           3
Jaguar              3
Wagon               3
BMW                 2
Toyota              2
Datsun              2
Honda               2
Convertibles        2
Audi                1
Porsche             1
Volvo               1
Hyundai             1
Skoda               1
Isuzu               1
Name: count, dtype: int64
```

Below is the list of car body types in india Types of Cars in India 1. Hatchback 2. Sedan/Saloon/Notchback 3. Compact Sedan 4. Coupe 5. Micro Car 6. CUV/Crossover 7. Crossover Hatchback 8. MPV/Minivan 9. SUV (Sports Utility Vehicle) 10. Crossover SUV 11. Coupe SUV 12. Compact SUV 13. 4-Door Coupe 14. Station Wagon 15. Convertible/Spyder/Cabriolet 16. Pick-Up Truck 17. MUV (Multi utility vehicle)

So the list of body types apart from this are very rare and it can be removed

[1197]:
```
value_counts = df['city'].value_counts()
print(value_counts)
```

```
city
Delhi         3036
Bangalore     2334
Mumbai        2109
Hyderabad     1584
Pune          1394
Chennai       1344
Ahmedabad     1330
Kolkata       1181
```

```
Gurgaon        1043
Jaipur          897
Lucknow         551
Chandigarh      437
Agra            206
Name: count, dtype: int64
```

No rare categories thus no outliers in city

0.3.6 Handling the data quality issues(1.5M)

Apply techniques * to remove duplicate data * to impute or remove missing data * to remove data inconsistencies Give detailed explanation for each column how you handle the data quality issues.

First lets remove the duplicate rows

[1198]:
```
df.shape
```

[1198]:
```
(17446, 14)
```

[1199]:
```
df.drop_duplicates(inplace=True)
```

[1200]:
```
df.shape
```

[1200]:
```
(17234, 14)
```

212 rows were duplicated and that has been removed

Now lets resolve the data inconsistency issue in max_power column. Take a value from the inconsistent data 132/4000-6000 this means the car power is 132 bhp at 4000 to 6000 rpm. Here we need only bhp. So we can take only the first number from the inconsistent data which gives us the bhp value.

[1201]:
```
def remove_power_inconsistency(car_power):
    if isinstance(car_power, str) :
        car_power = car_power.split(' ')[0].strip()
        car_power = car_power.split('/')[0].strip()
        car_power = car_power.split('(')[0].strip()
        return car_power
    return car_power
```

[1202]:
```
df["max_power"] = df["max_power"].apply(remove_power_inconsistency)
```

[1203]:
```
# Apply filter and create new DataFrame with only float values
filtered_df = df[df['max_power'].apply(filter_float_values)]
print("Inconsistent Powers:")
print(filtered_df['max_power'])
```

```
Inconsistent Powers:
Series([], Name: max_power, dtype: object)
```

Now there are no inconsistent power values. This column can be converted to a float column.

[1204]:
```
df['max_power'] = df['max_power'].astype('float')
```

[1205]:
```
df.dtypes
```

[1205]:
```
full_name             object
resale_price         float64
registered_year        Int64
engine_capacity        Int64
insurance             object
transmission_type     object
kms_driven             Int64
owner_type            object
fuel_type             object
max_power            float64
seats                  Int64
mileage              float64
body_type             object
city                  object
dtype: object
```

0.3.7 Now lets impute missing values

[1206]:
```
df.isnull().sum()
```

[1206]:
```
full_name              0
resale_price           0
registered_year       68
engine_capacity       13
insurance              7
transmission_type      0
kms_driven             3
owner_type            45
fuel_type              0
max_power            100
seats                 10
mileage              501
body_type              0
city                   0
dtype: int64
```

0.3.8 1. Registered_year

It has 68 missing values. Lets use the median of the registered_year to impute the missing value.Just going for the middle year in the range of years we have in the dataset.

[1207]:
```python
median_year = df['registered_year'].median()
print(median_year)

# Replace missing values with median
df.fillna({'registered_year':median_year}, inplace=True)
```

```
2017.0
```

0.3.9 2. Engine_capacity

It has 13 missing values.The engine capacity of the cars is a numerical attribute so we can use mean to impute the missing values

[1208]:
```python
mean_engine = df['engine_capacity'].mean().round()
print(mean_engine)

# Replace missing values with mean
df.fillna({'engine_capacity':mean_engine}, inplace=True)
```

```
1423.0
```

0.3.10 3. Insurance

It has 7 missing values.Lets see the value counts for insurance since it is a categorical attribute

[1209]:
```python
value_counts = df['insurance'].value_counts()
print(value_counts)
```

```
insurance
Third Party insurance    7365
Comprehensive            6397
Third Party              1973
Zero Dep                  834
Not Available             650
1                           5
2                           3
Name: count, dtype: int64
```

Since we already have a category called 'Not Available' lets use the same to impute the missing values

[1210]:
```python
df.fillna({'insurance':'Not Available'}, inplace=True)
```

0.3.11 4. kms_driven

It has 3 missing values.The kms driven of the cars is a numerical attribute so we can use mean to impute the missing values

[1211]:
```python
mean_kms = df['kms_driven'].mean().round()
print(mean_kms)

# Replace missing values with mean
df.fillna({'kms_driven':mean_engine}, inplace=True)
```

```
58606.0
```

0.3.12 5. Owner_type

It has 45 missing values.Lets see the value counts since this is a categorical attribute

[1212]:
```python
value_counts = df['owner_type'].value_counts()
print(value_counts)
```

```
owner_type
First Owner     12169
Second Owner     4086
Third Owner       761
Fourth Owner      124
Fifth Owner        49
Name: count, dtype: int64
```

Since this a categorical attribute we can use mode imputation

[1213]:
```python
mode_owner_type = df['owner_type'].mode().iloc[0]
print(mode_owner_type)

# Replace missing values with mode
df.fillna({'owner_type':mode_owner_type}, inplace=True)
```

```
First Owner
```

0.3.13 6. max_power

It has 100 missing values. Since this is an numerical attribute and the maximum power of a car which is ready for resale will be more or less the same so we can go for mean imputation

[1214]:
```python
mean_power = df['max_power'].mean()
print(mean_power)

# Replace missing values with mean
df.fillna({'max_power':mean_power}, inplace=True)
```

```
103.869768463873
```

0.3.14 7. seats

It has 10 missing values. Most of the cars in India has the same seats which is 5 only few cars like SUV has more number of seats. So we can go for mode imputation here

[1215]:
```python
mode_seats = df['seats'].mode().iloc[0]
print(mode_seats)

# Replace missing values with mode
df.fillna({'seats':mode_seats}, inplace=True)
```

```
5
```

0.3.15 8. mileage

It has 501 missing values. Since this is an numerical attribute and the mileage of a car which is ready for resale will be more or less the same so we can go for mean imputation

[1216]:
```python
mean_mileage = df['mileage'].mean()
print(mean_mileage)

# Replace missing values with mean
df.fillna({'mileage':mean_mileage}, inplace=True)
```

```
19.521191059582858
```

[1217]:
```python
df.isnull().sum()
```

[1217]:
```
full_name           0
resale_price        0
registered_year     0
engine_capacity     0
insurance           0
transmission_type   0
kms_driven          0
owner_type          0
fuel_type           0
max_power           0
seats               0
mileage             0
body_type           0
city                0
dtype: int64
```

All missing values have been imputed

0.3.16 Now lets remove the outliers from numerical attributes.

[1218]:
```python
df.shape
```

[1218]:
```
(17234, 14)
```

Now since max_power has also been converted to numerical attribute we can see if there is any outliers in max_power too.

[1219]:
```python
identifyOutliers(df, "max_power")
```

```
Old Shape:  (17234, 14)
Upper limit:  177.52499999999998
No. of datapoints greater than upper limit:  1213
Lower limit:  19.725000000000023
No. of datapoints less than lower limit:  0
```

max_power has 1213 outliers

Now lets remove outliers from max_power and mileage

[1220]:
```python
def remove_outliers_iqr(df, columns):
    # Copy the original dataframe
    df_cleaned = df.copy()

    for col in columns:
        # Calculate Q1 (25th percentile) and Q3 (75th percentile)
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)

        # Calculate IQR
        IQR = Q3 - Q1

        # Calculate bounds
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR

        # Filter values between lower and upper bounds
        df_cleaned = df_cleaned[(df_cleaned[col] >= lower_bound) &
 ↪(df_cleaned[col] <= upper_bound)]

    return df_cleaned

df = remove_outliers_iqr(df, ['mileage','max_power'])
```

[1221]:
```python
df.shape
```

[1221]: (15618, 14)

0.3.17 As explained above the categorical attributes insurance and body_type has some outliers

0.3.18 1. Insurance

[1222]:
```python
value_counts = df['insurance'].value_counts()
print(value_counts)
```

```
insurance
Third Party insurance    6761
```

```
Comprehensive           5561
Third Party             1892
Zero Dep                 770
Not Available            626
1                          5
2                          3
Name: count, dtype: int64
```

As mentioned earlier 1 and 2 are very rare and it is not a type of insuracen in india so this can be removed

[1223]:
```
df['insurance'] = df['insurance'].astype(str)

# Remove rows where column 'insurance' equals "1"
df = df[df['insurance'] != "1"]
```

[1224]:
```
# Remove rows where column 'insurance' equals "2"
df = df[df['insurance'] != "2"]
```

[1225]:
```
df.shape
```

[1225]:
```
(15610, 14)
```

0.3.19 2. Body_type

[1226]:
```
value_counts = df['body_type'].value_counts()
print(value_counts)
```

```
body_type
Hatchback        6992
Sedan            4037
SUV              3780
MUV               700
Minivans           42
Maruti             15
Pickup             13
Tata                7
Mahindra            4
Chevrolet           3
Cars                2
Honda               2
Datsun              2
Mercedes-Benz       2
Toyota              2
Volvo               1
Audi                1
Hyundai             1
Wagon               1
Skoda               1
```

```
Isuzu                1
Convertibles         1
Name: count, dtype: int64
```

Lets remove the rare categories which is not a body type

Maruti is a rare one and it is not a valid body type so lets remove it

[1227]:
```python
df['body_type'] = df['body_type'].astype(str)

# Remove rows where column 'body_type' equals "Maruti"
df = df[df['body_type'] != "Maruti"]
```

Tata is a rare one and it is not a valid body type so lets remove it

[1228]:
```python
# Remove rows where column 'body_type' equals "Tata"
df = df[df['body_type'] != "Tata"]
```

Mahindra is a rare one and it is not a valid body type so lets remove it

[1229]:
```python
# Remove rows where column 'body_type' equals "Mahindra"
df = df[df['body_type'] != "Mahindra"]
```

Chevrolet is a rare one and it is not a valid body type so lets remove it

[1230]:
```python
# Remove rows where column 'body_type' equals "Chevrolet"
df = df[df['body_type'] != "Chevrolet"]
```

Cars is a rare one and it is not a valid body type so lets remove it

[1231]:
```python
# Remove rows where column 'body_type' equals "Cars"
df = df[df['body_type'] != "Cars"]
```

Honda is a rare one and it is not a valid body type so lets remove it

[1232]:
```python
# Remove rows where column 'body_type' equals "Honda"
df = df[df['body_type'] != "Honda"]
```

Datsun is a rare one and it is not a valid body type so lets remove it

[1233]:
```python
# Remove rows where column 'body_type' equals "Datsun"
df = df[df['body_type'] != "Datsun"]
```

Mercedes-Benz is a rare one and it is not a valid body type so lets remove it

[1234]:
```python
# Remove rows where column 'body_type' equals "Mercedes-Benz"
df = df[df['body_type'] != "Mercedes-Benz"]
```

Toyota is a rare one and it is not a valid body type so lets remove it

[1235]:
```python
# Remove rows where column 'body_type' equals "Toyota"
df = df[df['body_type'] != "Toyota"]
```

Volvo is a rare one and it is not a valid body type so lets remove it

[1236]:
```
# Remove rows where column 'body_type' equals "Volvo"
df = df[df['body_type'] != "Volvo"]
```

Audi is a rare one and it is not a valid body type so lets remove it

[1237]:
```
# Remove rows where column 'body_type' equals "Audi"
df = df[df['body_type'] != "Audi"]
```

Hyundai is a rare one and it is not a valid body type so lets remove it

[1238]:
```
# Remove rows where column 'body_type' equals "Hyundai"
df = df[df['body_type'] != "Hyundai"]
```

Wagon is a rare one and it is not a valid body type so lets remove it

[1239]:
```
# Remove rows where column 'body_type' equals "Wagon"
df = df[df['body_type'] != "Wagon"]
```

Skoda is a rare one and it is not a valid body type so lets remove it

[1240]:
```
# Remove rows where column 'body_type' equals "Skoda"
df = df[df['body_type'] != "Skoda"]
```

Izusu is a rare one and it is not a valid body type so lets remove it

[1241]:
```
# Remove rows where column 'body_type' equals "Isuzu"
df = df[df['body_type'] != "Isuzu"]
```

Convertibles is also rare but it is a valid body type no lets keep it

[1242]:
```
value_counts = df['body_type'].value_counts()
print(value_counts)
```

```
body_type
Hatchback       6992
Sedan           4037
SUV             3780
MUV              700
Minivans          42
Pickup            13
Convertibles       1
Name: count, dtype: int64
```

[1243]:
```
df.shape
```

[1243]:
```
(15565, 14)
```

0.3.20 Standardise the data (1M)

Standardization is the process of transforming data into a common format which you to make the meaningful comparison.

All columns have already been transformed to a common format in the above steps. Insurance column has some more scope to convert into a common format

[1244]:
```
value_counts = df['insurance'].value_counts()
print(value_counts)
```

```
insurance
Third Party insurance    6730
Comprehensive            5549
Third Party              1891
Zero Dep                  770
Not Available             625
Name: count, dtype: int64
```

Here third party insurance and third party means the same. So we can replace thid party with third party insurance.

[1245]:
```
df['insurance'] = df['insurance'].replace('Third Party', 'Third Party␣
↪insurance')
```

[1246]:
```
value_counts = df['insurance'].value_counts()
print(value_counts)
```

```
insurance
Third Party insurance    8621
Comprehensive            5549
Zero Dep                  770
Not Available             625
Name: count, dtype: int64
```

Now all columns have been standardised

0.3.21 Normalise the data wherever necessary(1M)

We can normalize all numerical attributes except registered_year, seats and mileage because they do not exhibit extreme variability or wide scales that would necessitate normalization to bring values into a standardized range.

[1247]:
```
from sklearn.preprocessing import MinMaxScaler

# Calculate min and max values of resale_price so that it can be used later to␣
↪find the actual price
resale_price_min = np.min(df["resale_price"])
resale_price_max = np.max(df["resale_price"])

# Example dataframe 'df' with selected columns to normalize
```

```
columns_to_normalize = ['resale_price', 'engine_capacity', 'kms_driven',␣
 ↪'max_power']

# Initialize the MinMaxScaler
scaler = MinMaxScaler()

# Fit and transform the selected columns
df[columns_to_normalize] = scaler.fit_transform(df[columns_to_normalize])

# View the normalized dataframe
print(df.head())
```

```
                    full_name  resale_price  registered_year  \
0  2017 Maruti Baleno 1.2 Alpha         0.072             2017
1            2018 Tata Hexa XTA         0.136             2018
2   2015 Maruti Swift Dzire VXI         0.059             2015
4    2009 Hyundai i10 Magna 1.1         0.018             2009
5   2015 Hyundai i20 Active 1.2         0.062             2015

   engine_capacity             insurance transmission_type  kms_driven  \
0            0.399  Third Party insurance            Manual       0.012
1            0.727  Third Party insurance         Automatic       0.021
2            0.399  Third Party insurance            Manual       0.021
4            0.362  Third Party insurance            Manual       0.024
5            0.399  Third Party insurance            Manual       0.021

     owner_type fuel_type  max_power  seats  mileage  body_type  city
0   First Owner    Petrol      0.348      5   21.400  Hatchback  Agra
1   First Owner    Diesel      0.837      7   17.600        MUV  Agra
2  Second Owner    Petrol      0.348      5   20.850      Sedan  Agra
4   First Owner    Petrol      0.244      5   19.810  Hatchback  Agra
5   First Owner    Petrol      0.339      5   17.190  Hatchback  Agra
```

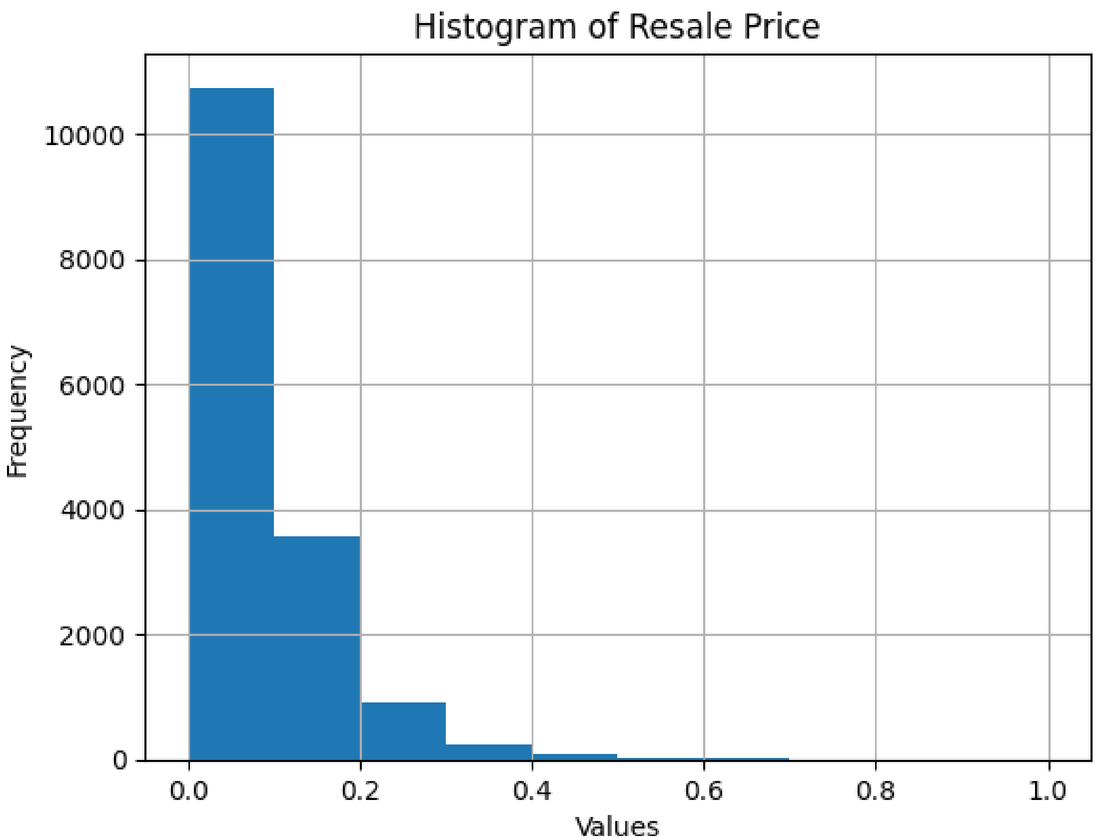
0.3.22 Perform Binning (1M)

Binning is a process of transforming continuous numerical variables into discrete categorical 'bins', for grouped analysis.

We can do binning for a numerical attributes which is uniformly distributed. Lets take the numerical attributes one by one and we will plot an histogram to see the distribtuion

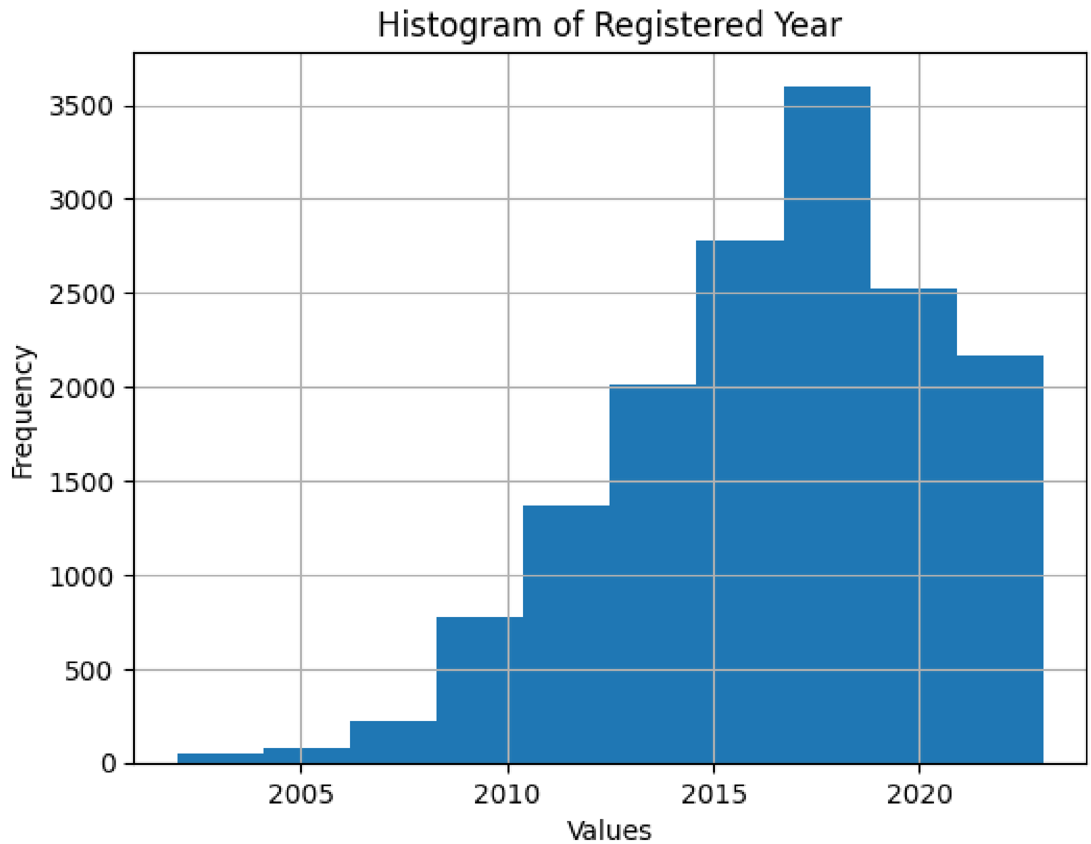
[1248]:
```
import matplotlib.pyplot as plt

# Plot a histogram of a single column in the DataFrame
df.hist(column='resale_price')

# Set the title and axis labels
```

```
plt.title('Histogram of Resale Price')
plt.xlabel('Values')
plt.ylabel('Frequency')

# Display the histogram
plt.show()
```



Resale price distribution is skewed so we can discretize this column in the further sections

[1249]:
```
# Plot a histogram of a single column in the DataFrame
df.hist(column='registered_year')

# Set the title and axis labels
plt.title('Histogram of Registered Year')
plt.xlabel('Values')
plt.ylabel('Frequency')

# Display the histogram
plt.show()
```



Registered year distribution is uniformly distributed to some extent so we can do binning

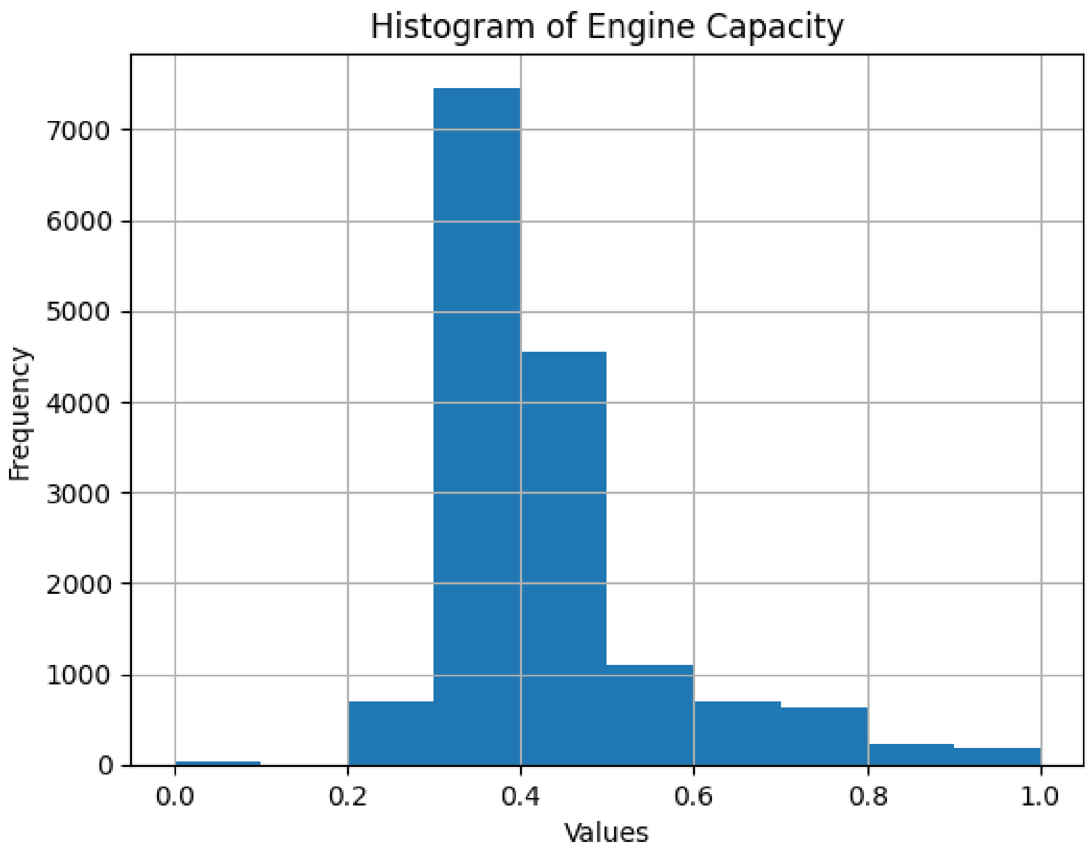
[1250]:
```
# Binning for registered_year
df['registered_year_bin'] = pd.cut(df['registered_year'], bins=3,␣
↪labels=['Vintage', 'Old', 'New'])
```

[1251]:
```
value_counts = df['registered_year_bin'].value_counts()
print(value_counts)
```

```
registered_year_bin
New        8292
Old        6646
Vintage     627
Name: count, dtype: int64
```

[1252]:
```
df.hist(column='engine_capacity')

# Set the title and axis labels
plt.title('Histogram of Engine Capacity')
plt.xlabel('Values')
```

```
plt.ylabel('Frequency')

# Display the histogram
plt.show()
```



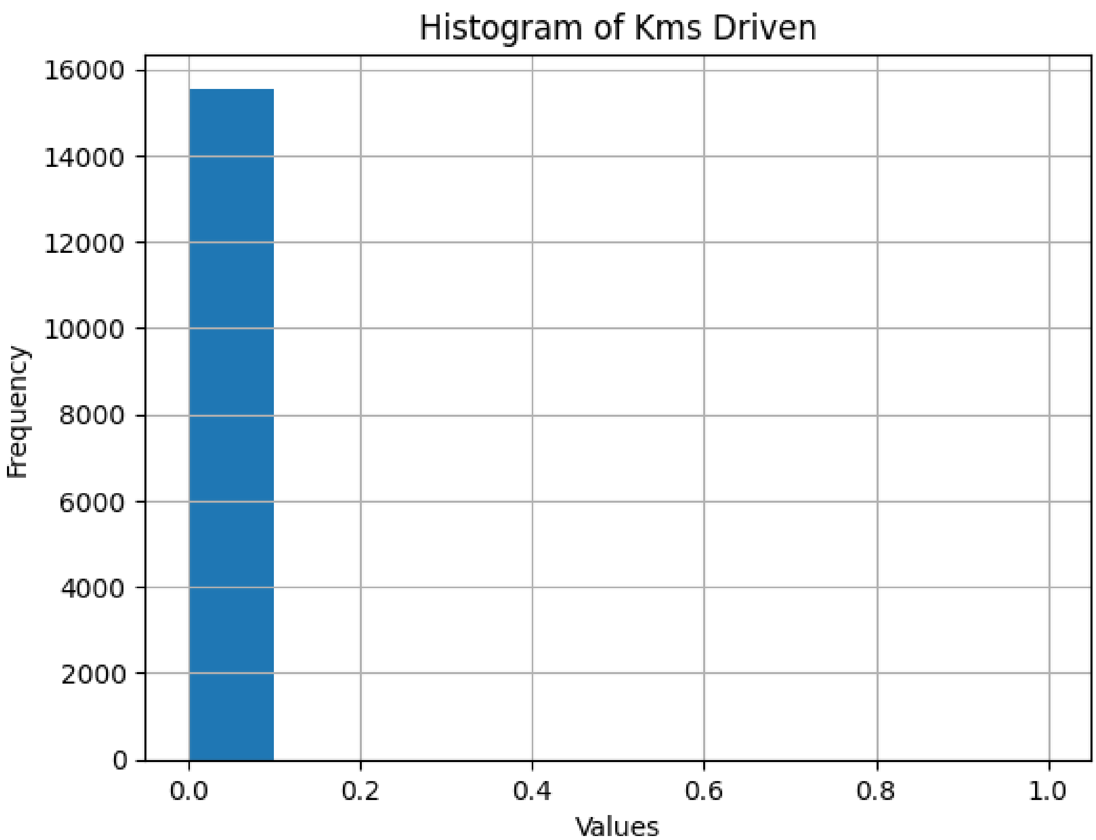
Engine capacity distribution is uniformly distributed to some extent so we can do binning

[1253]:
```
# Binning for engine_capacity
df['engine_capacity_bin'] = pd.cut(df['engine_capacity'], bins=3,␣
 ↪labels=['Low', 'Medium', 'High'])
```

[1254]:
```
value_counts = df['engine_capacity_bin'].value_counts()
print(value_counts)
```

```
engine_capacity_bin
Medium    11907
Low        2619
High       1039
Name: count, dtype: int64
```

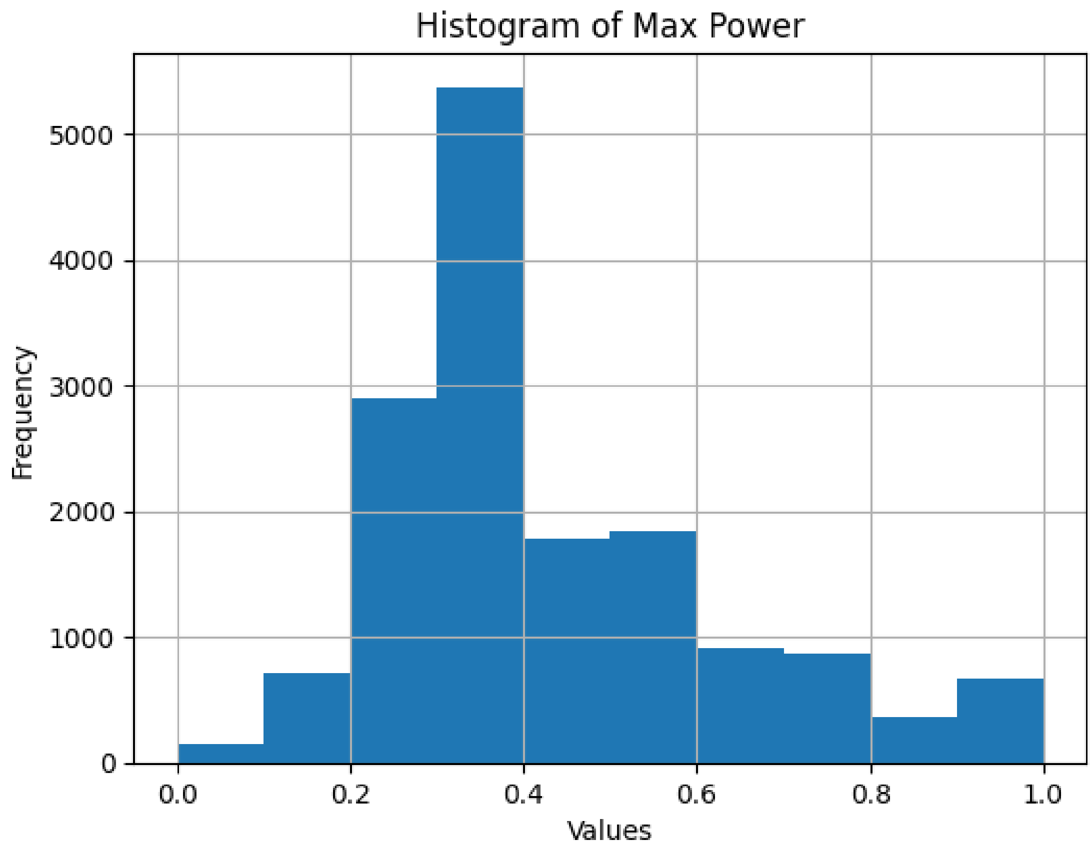
[1255]:
```
df.hist(column='kms_driven')

# Set the title and axis labels
plt.title('Histogram of Kms Driven')
plt.xlabel('Values')
plt.ylabel('Frequency')

# Display the histogram
plt.show()
```



Kms driven distribution is skewed so we can discretize this column in the further sections

[1256]:
```
df.hist(column='max_power')

# Set the title and axis labels
plt.title('Histogram of Max Power')
plt.xlabel('Values')
plt.ylabel('Frequency')
```

```
# Display the histogram
plt.show()
```



Max power distribution is uniformly distributed to some extent so we can do binning

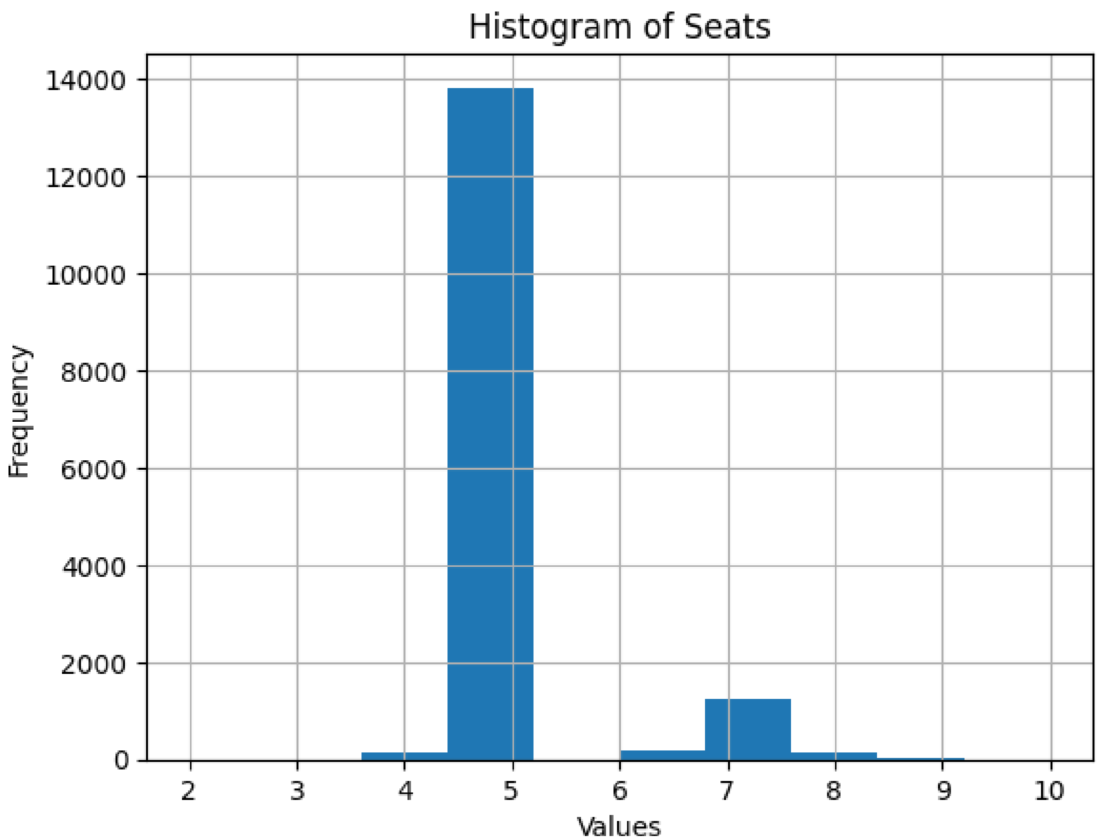
[1257]:
```
# Binning for max_power
df['max_power_bin'] = pd.cut(df['max_power'], bins=3, labels=['Low', 'Medium',␣
↪'High'])
```

[1258]:
```
value_counts = df['max_power_bin'].value_counts()
print(value_counts)
```

```
max_power_bin
Medium    9360
Low       4236
High      1969
Name: count, dtype: int64
```

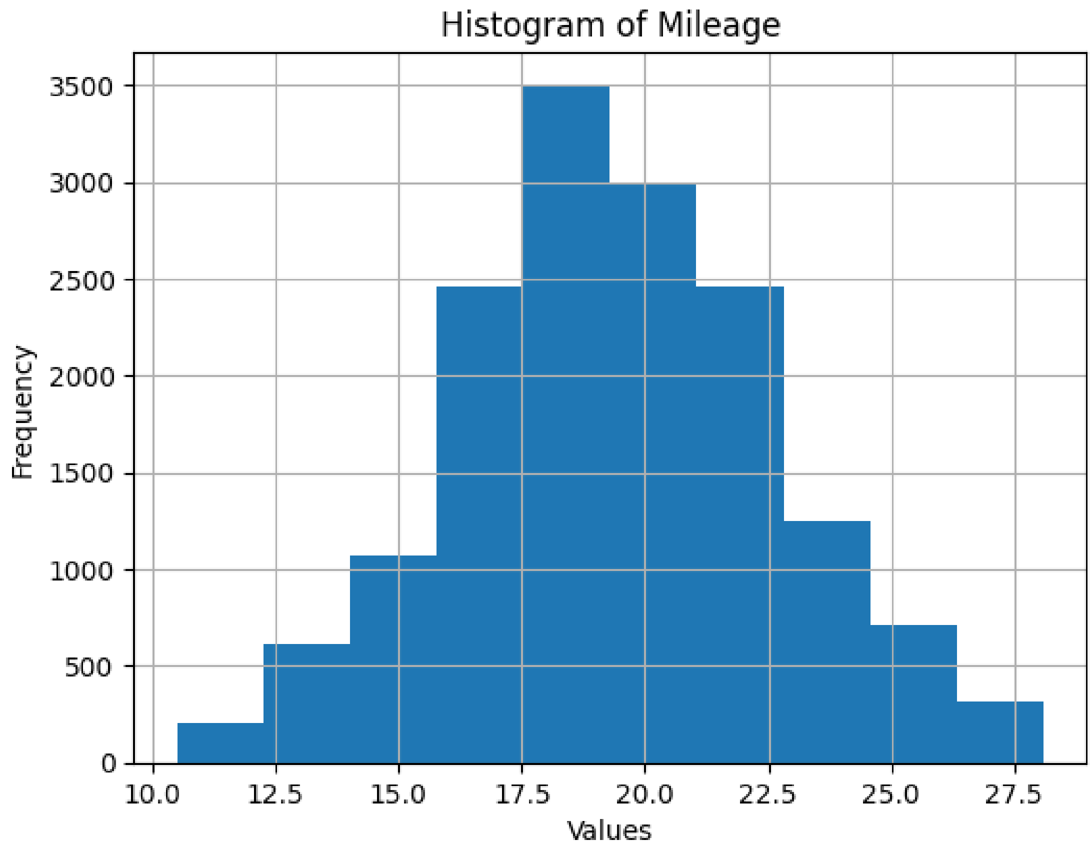
[1259]:
```
df.hist(column='seats')

# Set the title and axis labels
```

```
plt.title('Histogram of Seats')
plt.xlabel('Values')
plt.ylabel('Frequency')

# Display the histogram
plt.show()
```



Seats distribution is skewed so we can discretize this column in the further sections

[1260]:
```
df.hist(column='mileage')

# Set the title and axis labels
plt.title('Histogram of Mileage')
plt.xlabel('Values')
plt.ylabel('Frequency')

# Display the histogram
plt.show()
```



Mileage distribution is uniformly distributed to some extent so we can do binning

[1261]:
```
# Binning for mileage
df['mileage_bin'] = pd.cut(df['mileage'], bins=3, labels=['Low', 'Medium',␣
↪'High'])
```

[1263]:
```
value_counts = df['mileage_bin'].value_counts()
print(value_counts)
```

```
mileage_bin
Medium    10034
High       3055
Low        2476
Name: count, dtype: int64
```

0.3.23 Perform encoding (1M)

Lets take categorical attribute one by one and do encoding

Since full_name column has many unique categories we can do binary encoding

[1264]:
```python
from category_encoders import BinaryEncoder
```

[1265]:
```python
# Binary encoding
encoder = BinaryEncoder(cols=['full_name'])
df=encoder.fit_transform(df)
```

[1266]:
```python
df.columns
```

[1266]:
```
Index(['full_name_0', 'full_name_1', 'full_name_2', 'full_name_3',
       'full_name_4', 'full_name_5', 'full_name_6', 'full_name_7',
       'full_name_8', 'full_name_9', 'full_name_10', 'full_name_11',
       'full_name_12', 'resale_price', 'registered_year', 'engine_capacity',
       'insurance', 'transmission_type', 'kms_driven', 'owner_type',
       'fuel_type', 'max_power', 'seats', 'mileage', 'body_type', 'city',
       'registered_year_bin', 'engine_capacity_bin', 'max_power_bin',
       'mileage_bin'],
      dtype='object')
```

Insurance, transmission_type, fuel_type, body_type, city attributes have only few categories and it has no specific order so we can do one hot encoding

[1267]:
```python
from category_encoders import OneHotEncoder

# One-hot encoding
encoder =␣
↪OneHotEncoder(cols=['insurance','transmission_type','fuel_type','body_type','city'])
df = encoder.fit_transform(df)
```

Owner_type is categorical ordinal attribute we can use ordinal encoding

[1268]:
```python
from category_encoders import OrdinalEncoder

# Ordinal encoding
encoder = OrdinalEncoder(cols=['owner_type'], mapping=[{'col': 'owner_type',␣
↪'mapping': {'First Owner': 0, 'Second Owner': 1, 'Third Owner': 2, 'Fourth␣
↪Owner': 3, 'Fifth Owner': 5}}])
df = encoder.fit_transform(df)
```

[1269]:
```python
df.columns
```

[1269]:
```
Index(['full_name_0', 'full_name_1', 'full_name_2', 'full_name_3',
       'full_name_4', 'full_name_5', 'full_name_6', 'full_name_7',
       'full_name_8', 'full_name_9', 'full_name_10', 'full_name_11',
       'full_name_12', 'resale_price', 'registered_year', 'engine_capacity',
       'insurance_1', 'insurance_2', 'insurance_3', 'insurance_4',
       'transmission_type_1', 'transmission_type_2', 'kms_driven',
       'owner_type', 'fuel_type_1', 'fuel_type_2', 'fuel_type_3',
       'fuel_type_4', 'fuel_type_5', 'max_power', 'seats', 'mileage',
```

```
       'body_type_1', 'body_type_2', 'body_type_3', 'body_type_4',
       'body_type_5', 'body_type_6', 'body_type_7', 'city_1', 'city_2',
       'city_3', 'city_4', 'city_5', 'city_6', 'city_7', 'city_8', 'city_9',
       'city_10', 'city_11', 'city_12', 'city_13', 'registered_year_bin',
       'engine_capacity_bin', 'max_power_bin', 'mileage_bin'],
      dtype='object')
```

0.3.24 Perform Data Discretization(2M)

As mentioned above kms_driven and seats are in skewly distributed we will use entropy based discretization to discretize these columns. This will give more meaningful splits

[1270]:
```python
from sklearn.tree import DecisionTreeRegressor
from sklearn.preprocessing import KBinsDiscretizer

# Step 1: Initialize DecisionTreeRegressor
dt_regressor_kms_driven = DecisionTreeRegressor(max_depth=2)
dt_regressor_seats = DecisionTreeRegressor(max_depth=2)

# Step 2: Fit DecisionTreeRegressor on kms_driven and seats with resale_price␣
 ↪as target
X = df[['kms_driven', 'seats']]
y = df['resale_price']

dt_regressor_kms_driven.fit(X[['kms_driven']], y)
dt_regressor_seats.fit(X[['seats']], y)

# Step 3: Extract split points from decision trees
split_points_kms_driven = dt_regressor_kms_driven.tree_.
 ↪threshold[dt_regressor_kms_driven.tree_.threshold != -2]
split_points_seats = dt_regressor_seats.tree_.threshold[dt_regressor_seats.
 ↪tree_.threshold != -2]

print(f"Split points for kms_driven: {split_points_kms_driven}")
print(f"Split points for seats: {split_points_seats}")

# Step 4: Discretize the continuous variables (kms_driven and seats) using␣
 ↪KBinsDiscretizer
discretizer_kms_driven = KBinsDiscretizer(n_bins=len(split_points_kms_driven),␣
 ↪encode='ordinal', strategy='quantile')
df['kms_driven_discretized'] = discretizer_kms_driven.
 ↪fit_transform(df[['kms_driven']])

discretizer_seats = KBinsDiscretizer(n_bins=len(split_points_seats),␣
 ↪encode='ordinal', strategy='quantile')
df['seats_discretized'] = discretizer_seats.fit_transform(df[['seats']])
```

```
Split points for kms_driven: [0.01  0.004 0.018]
Split points for seats: [5.5 4.5 7.5]
```

```
/opt/miniconda3/envs/wilpenv/lib/python3.12/site-
packages/sklearn/preprocessing/_discretization.py:322: UserWarning: Bins whose
width are too small (i.e., <= 1e-8) in feature 0 are removed. Consider
decreasing the number of bins.
  warnings.warn(
```

resale_price is also skewed so we will do quantile based discretization

[1271]:
```python
# Step 1: Initialize KBinsDiscretizer
discretizer = KBinsDiscretizer(n_bins=3, encode='ordinal', strategy='quantile')

# Step 2: Fit and transform the data
df['resale_price_discretized'] = discretizer.fit_transform(df[['resale_price']])
```

[1272]:
```python
df.columns
```

[1272]:
```
Index(['full_name_0', 'full_name_1', 'full_name_2', 'full_name_3',
       'full_name_4', 'full_name_5', 'full_name_6', 'full_name_7',
       'full_name_8', 'full_name_9', 'full_name_10', 'full_name_11',
       'full_name_12', 'resale_price', 'registered_year', 'engine_capacity',
       'insurance_1', 'insurance_2', 'insurance_3', 'insurance_4',
       'transmission_type_1', 'transmission_type_2', 'kms_driven',
       'owner_type', 'fuel_type_1', 'fuel_type_2', 'fuel_type_3',
       'fuel_type_4', 'fuel_type_5', 'max_power', 'seats', 'mileage',
       'body_type_1', 'body_type_2', 'body_type_3', 'body_type_4',
       'body_type_5', 'body_type_6', 'body_type_7', 'city_1', 'city_2',
       'city_3', 'city_4', 'city_5', 'city_6', 'city_7', 'city_8', 'city_9',
       'city_10', 'city_11', 'city_12', 'city_13', 'registered_year_bin',
       'engine_capacity_bin', 'max_power_bin', 'mileage_bin',
       'kms_driven_discretized', 'seats_discretized',
       'resale_price_discretized'],
      dtype='object')
```

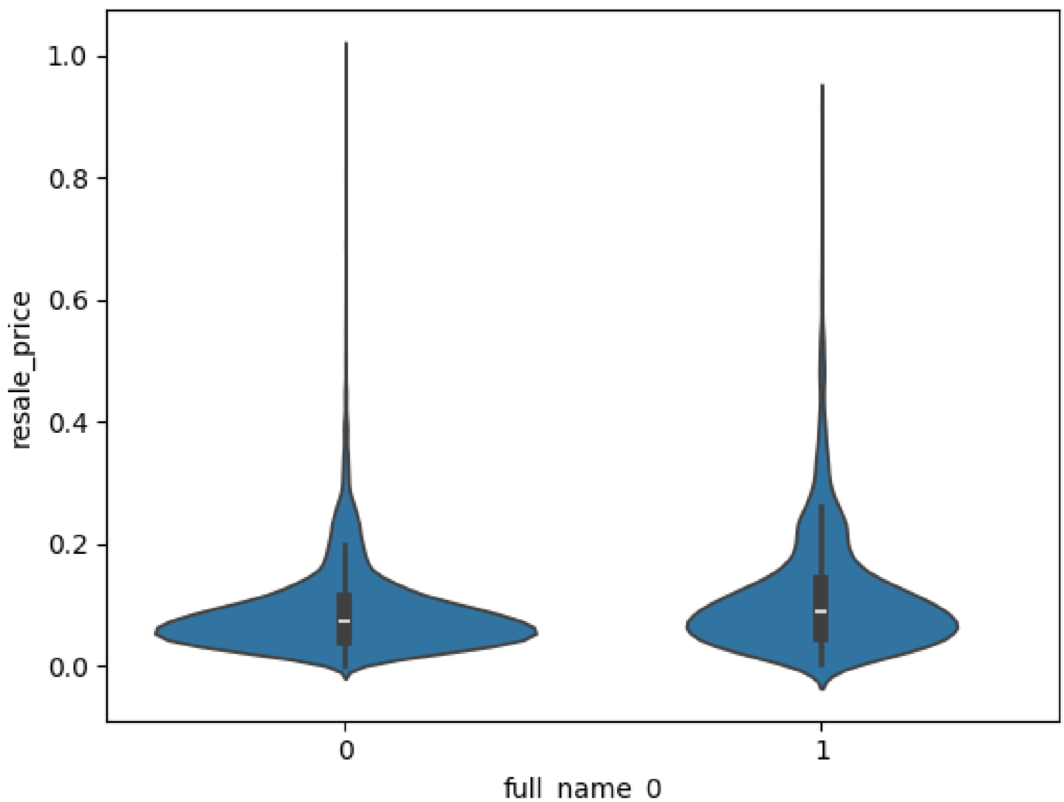
0.3.25 EDA using Visuals(3M)

Use any 3 or more visualisation methods (Boxplot,Scatterplot,histogram,....etc) to perform Exploratory data analysis and briefly give interpretations from each visual.

We will use violin plot for all categorical attributes because violin plot combines aspects of a box plot and a density plot to show the distribution of the target variable across different categories of features.

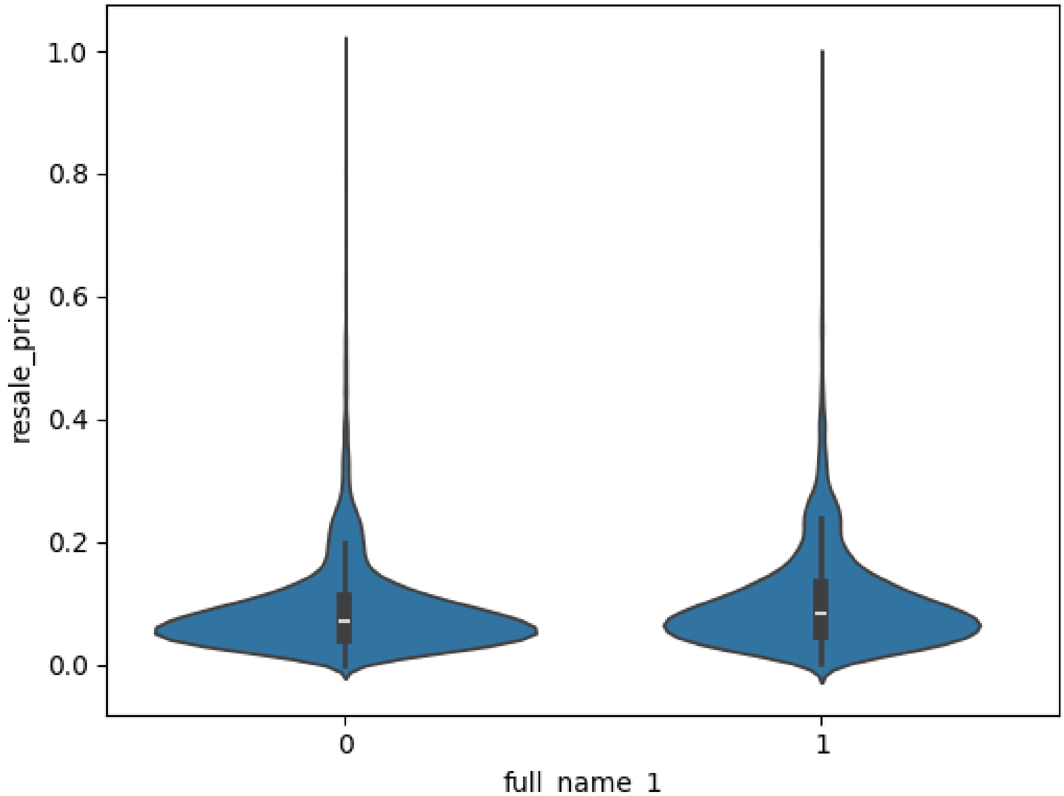
[1273]:
```python
import seaborn as sns

sns.violinplot(x="full_name_0", y="resale_price", data=df)
```

[1273]:
```
<Axes: xlabel='full_name_0', ylabel='resale_price'>
```



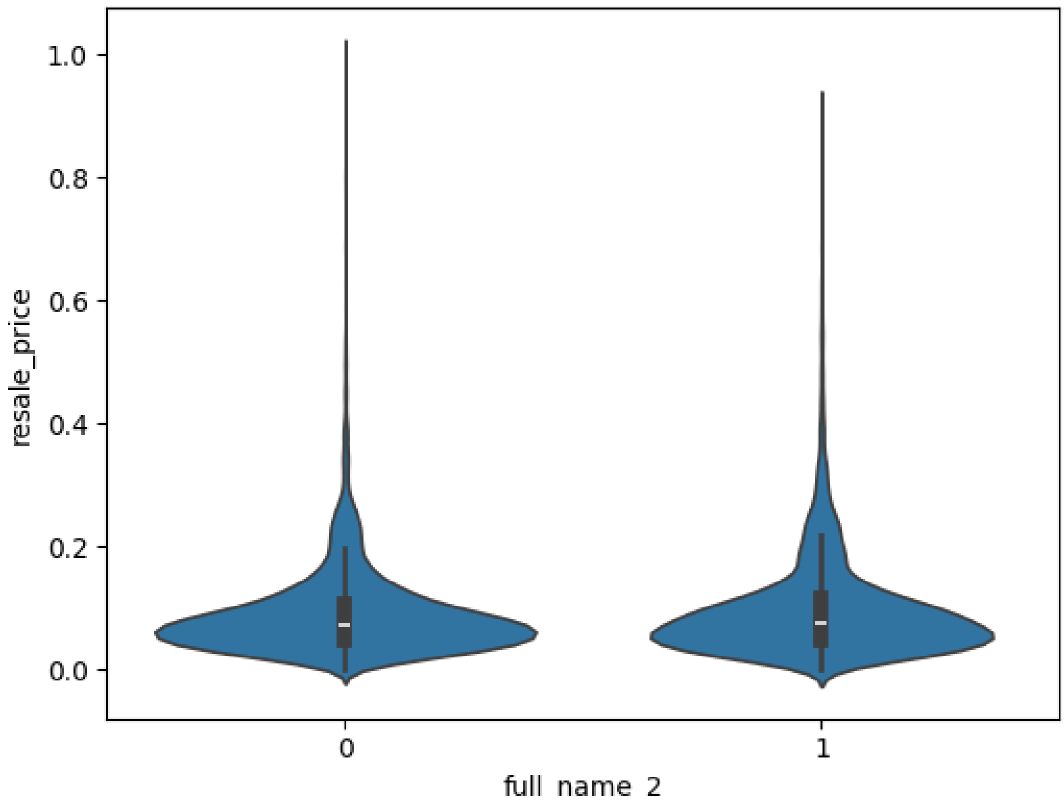

```
[1274]: sns.violinplot(x="full_name_1", y="resale_price", data=df)
```

```
[1274]: <Axes: xlabel='full_name_1', ylabel='resale_price'>
```



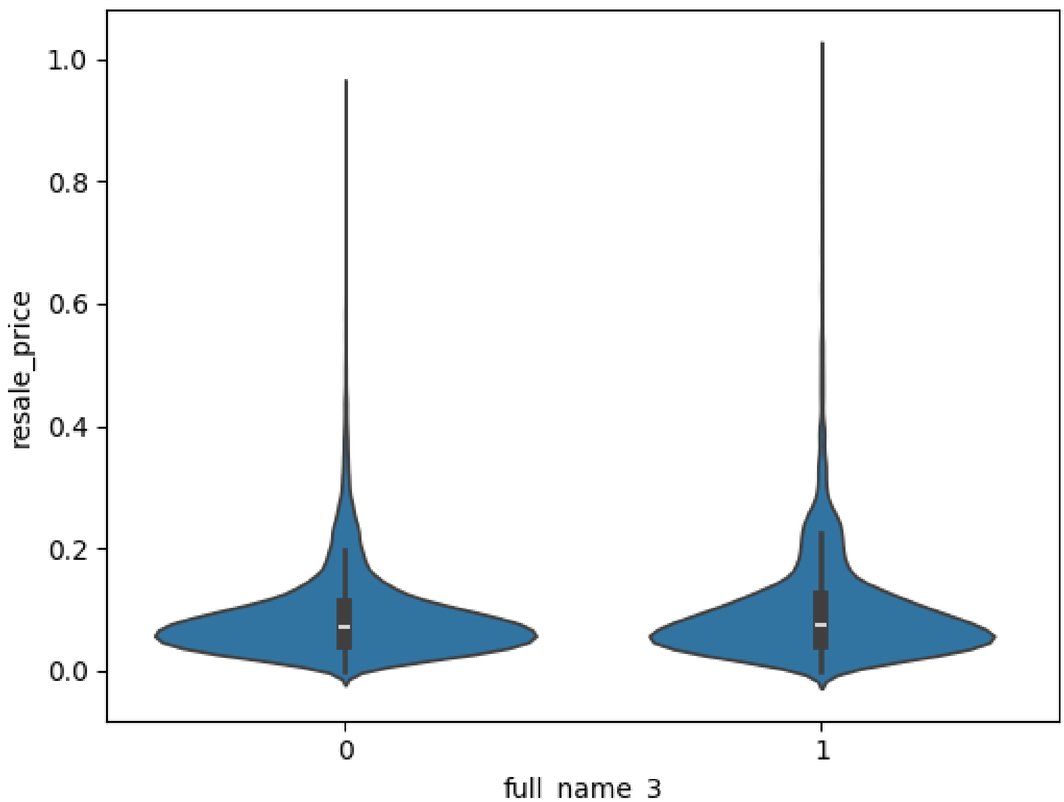
[1275]:
```
sns.violinplot(x="full_name_2", y="resale_price", data=df)
```

[1275]: <Axes: xlabel='full_name_2', ylabel='resale_price'>



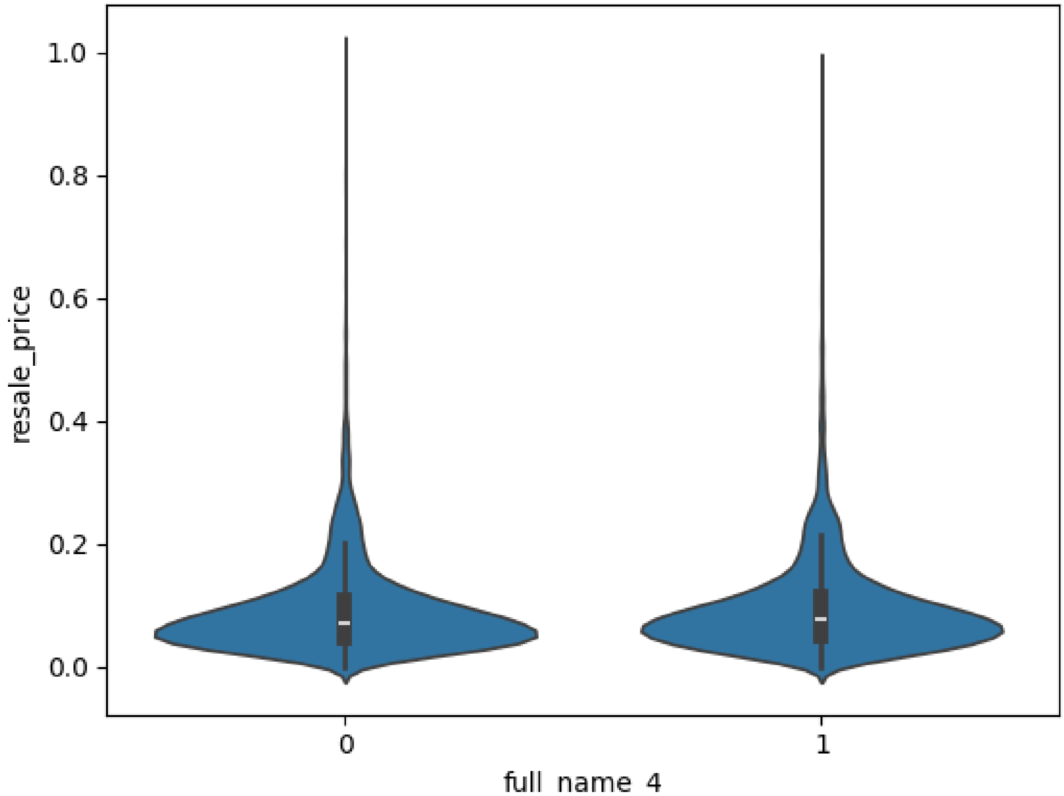
[1276]:
```
sns.violinplot(x="full_name_3", y="resale_price", data=df)
```

[1276]: <Axes: xlabel='full_name_3', ylabel='resale_price'>



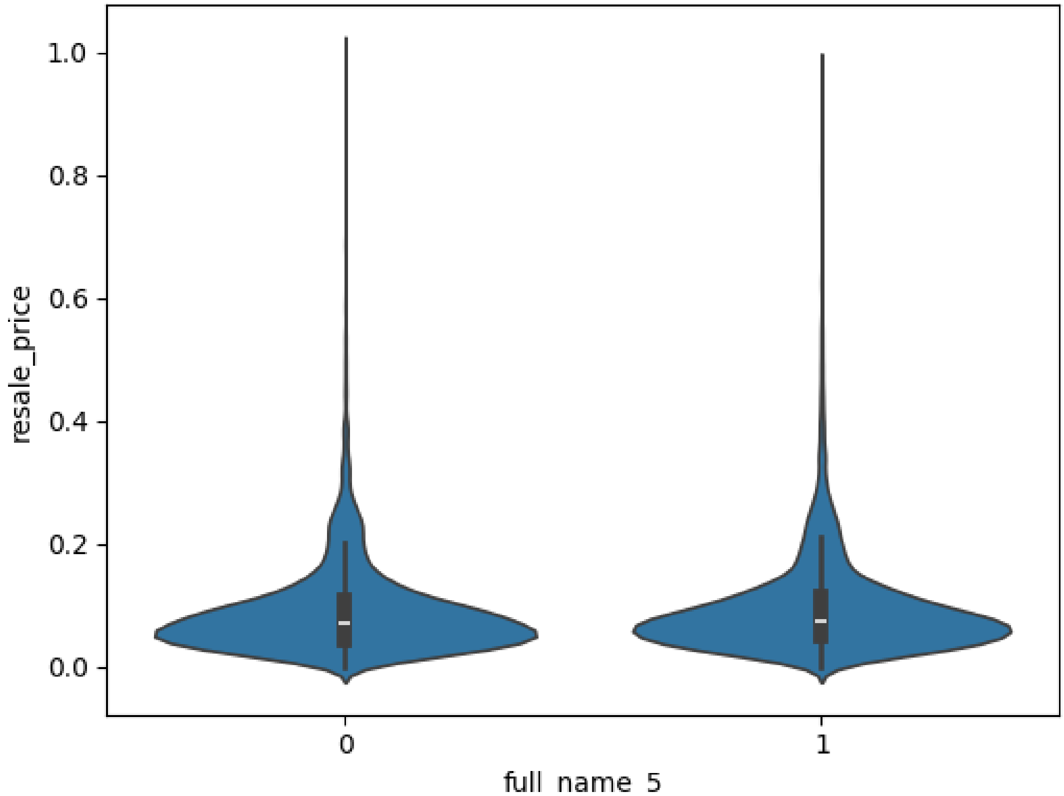
[1277]:
```
sns.violinplot(x="full_name_4", y="resale_price", data=df)
```

[1277]: <Axes: xlabel='full_name_4', ylabel='resale_price'>



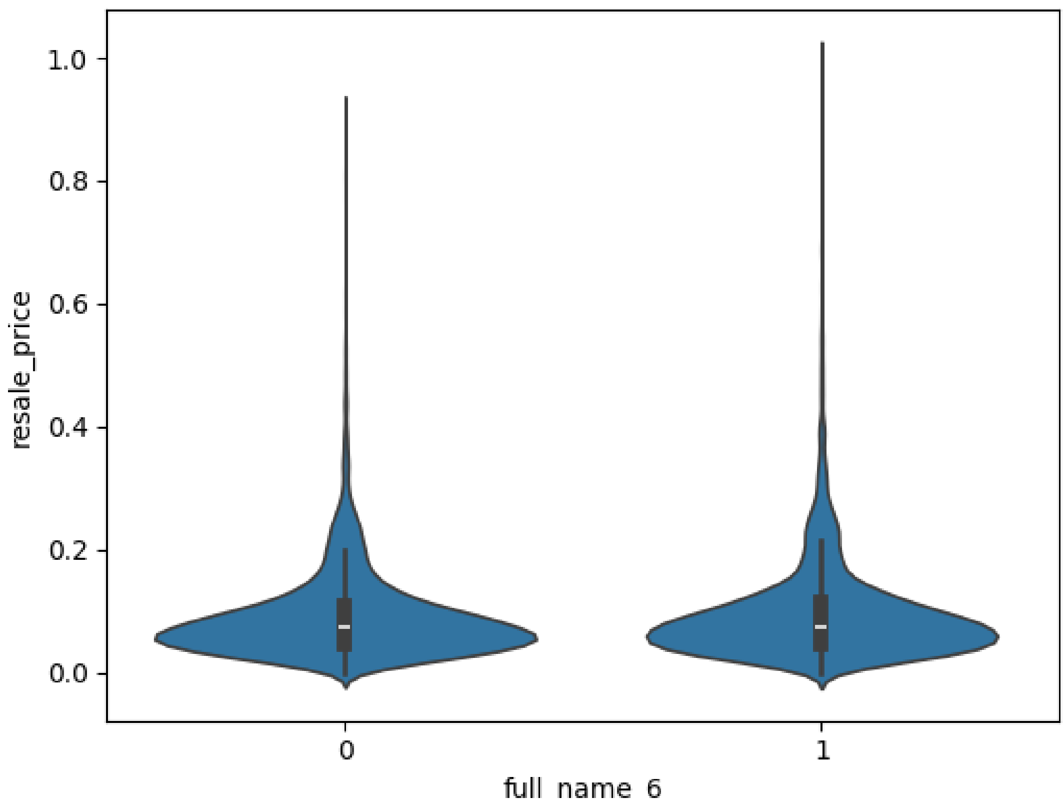
[1278]:
```
sns.violinplot(x="full_name_5", y="resale_price", data=df)
```

[1278]: <Axes: xlabel='full_name_5', ylabel='resale_price'>



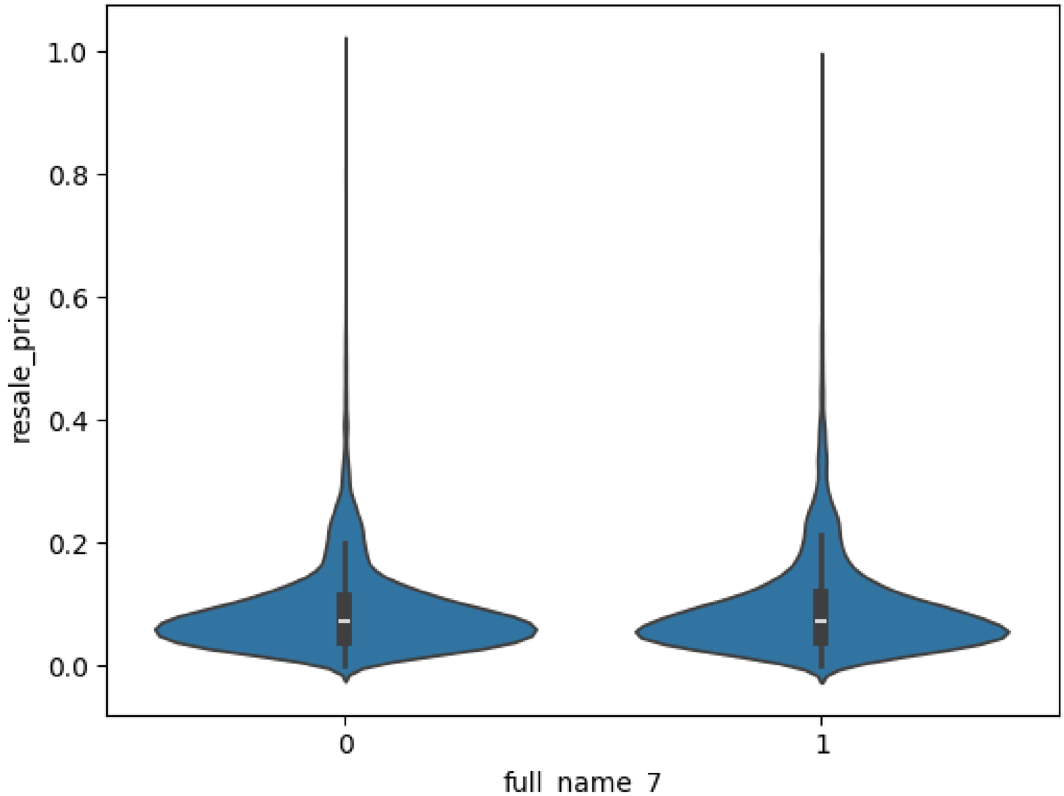
[1279]:
```
sns.violinplot(x="full_name_6", y="resale_price", data=df)
```

[1279]: <Axes: xlabel='full_name_6', ylabel='resale_price'>



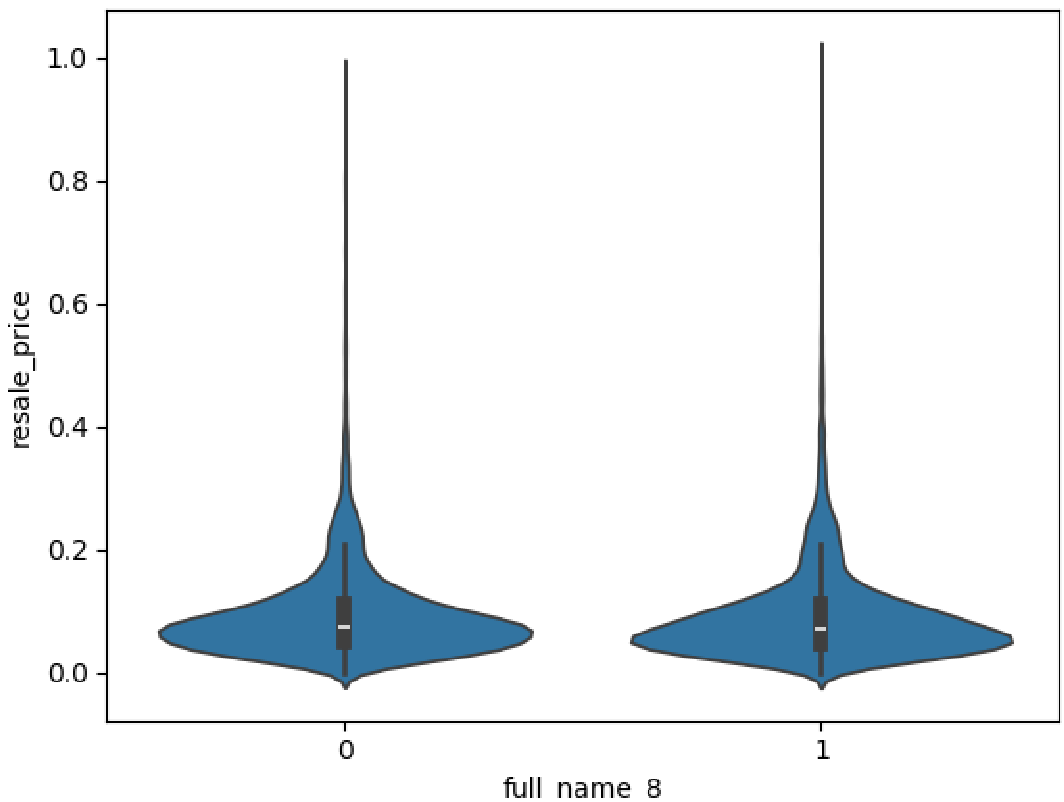
[1280]:
```
sns.violinplot(x="full_name_7", y="resale_price", data=df)
```

[1280]: <Axes: xlabel='full_name_7', ylabel='resale_price'>



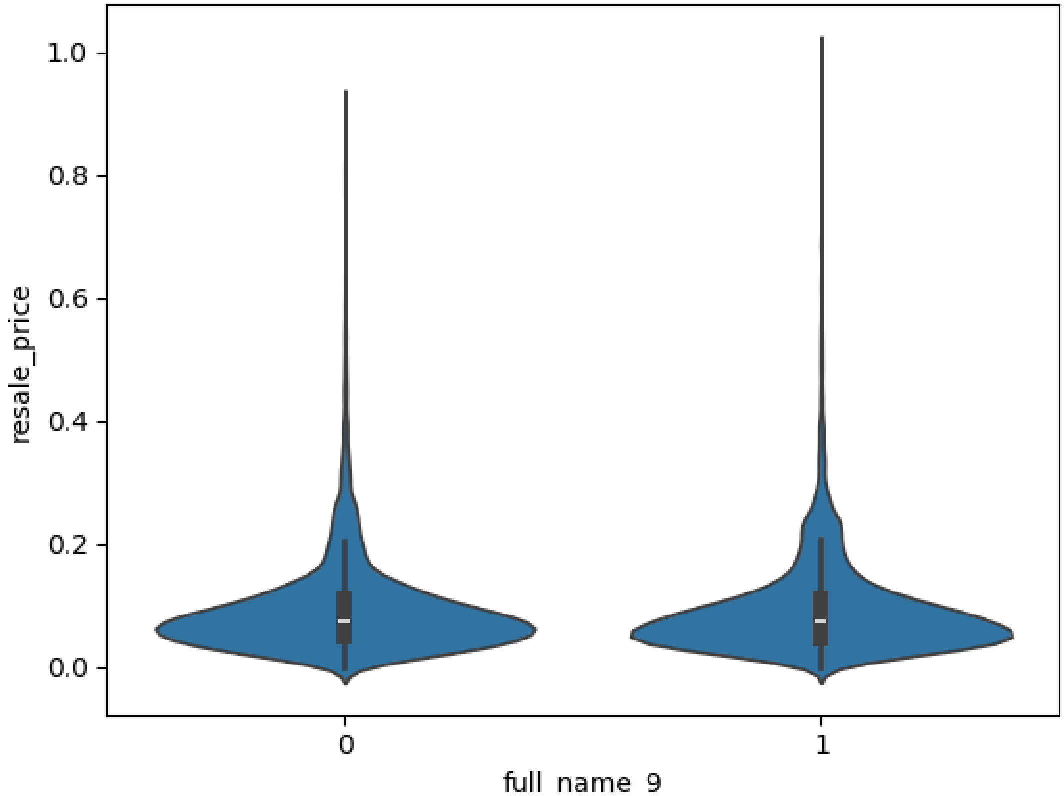
[1281]:
```
sns.violinplot(x="full_name_8", y="resale_price", data=df)
```

[1281]: <Axes: xlabel='full_name_8', ylabel='resale_price'>



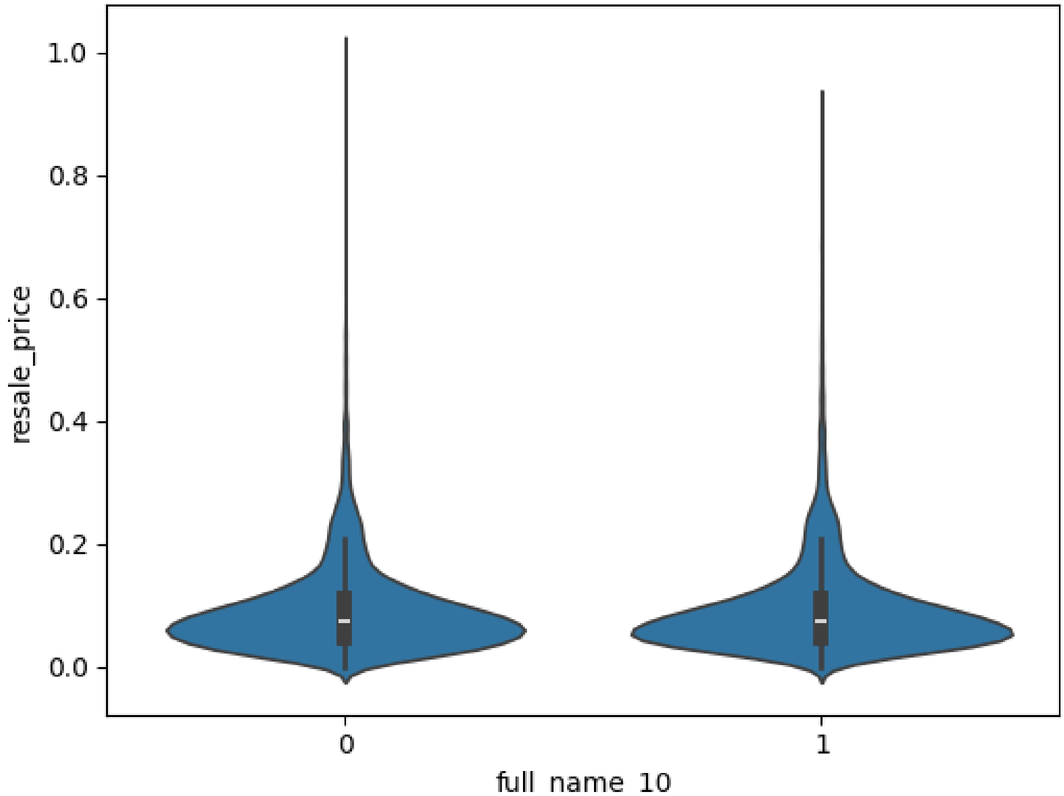

```
[1282]: sns.violinplot(x="full_name_9", y="resale_price", data=df)
```

```
[1282]: <Axes: xlabel='full_name_9', ylabel='resale_price'>
```



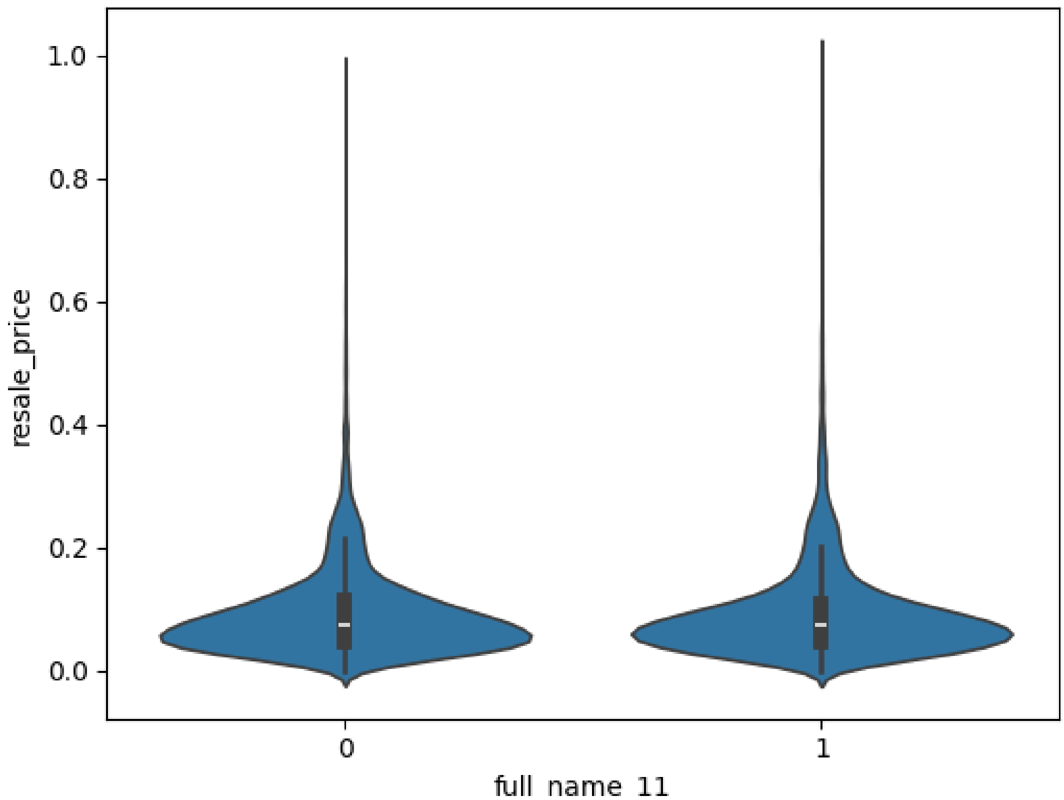
[1283]:
```
sns.violinplot(x="full_name_10", y="resale_price", data=df)
```

[1283]: <Axes: xlabel='full_name_10', ylabel='resale_price'>



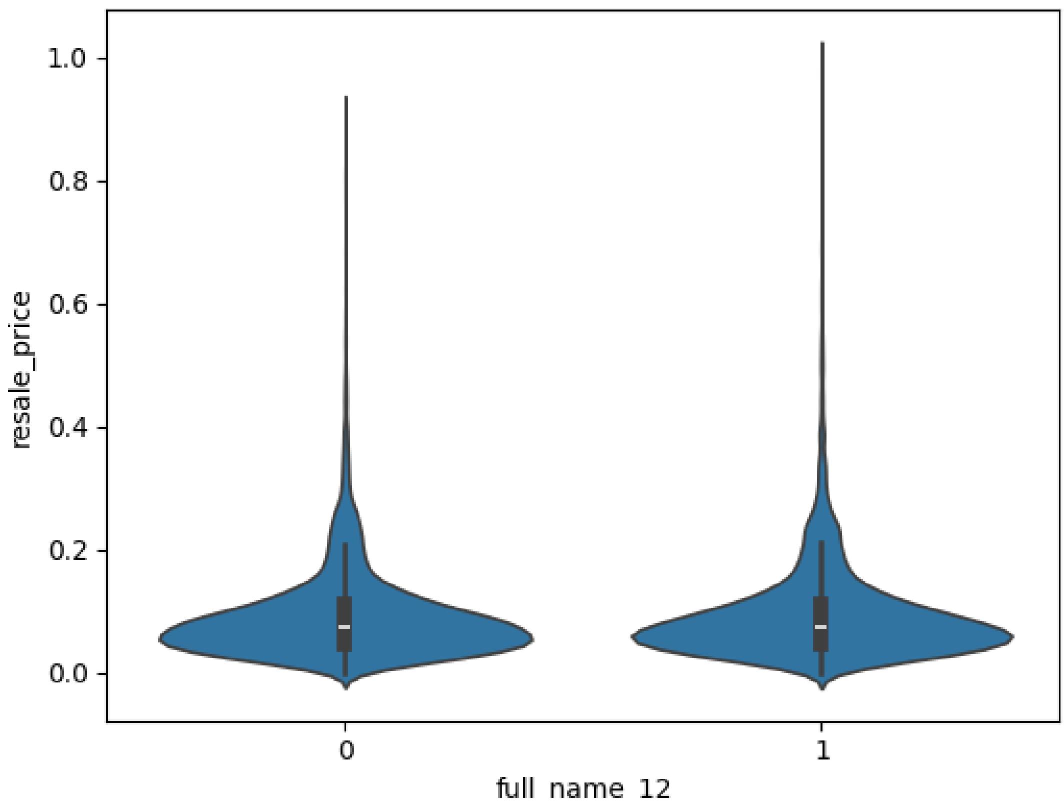
[1284]:
```
sns.violinplot(x="full_name_11", y="resale_price", data=df)
```

[1284]: `<Axes: xlabel='full_name_11', ylabel='resale_price'>`



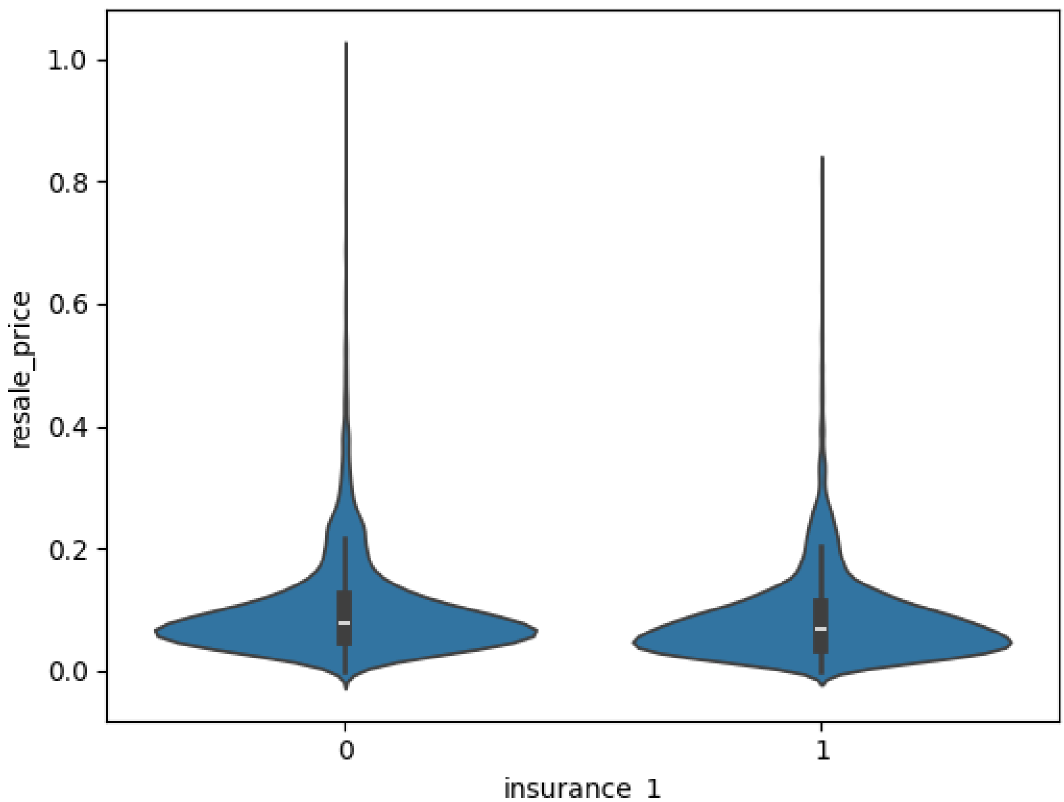
[1285]:
```
sns.violinplot(x="full_name_12", y="resale_price", data=df)
```

[1285]: <Axes: xlabel='full_name_12', ylabel='resale_price'>



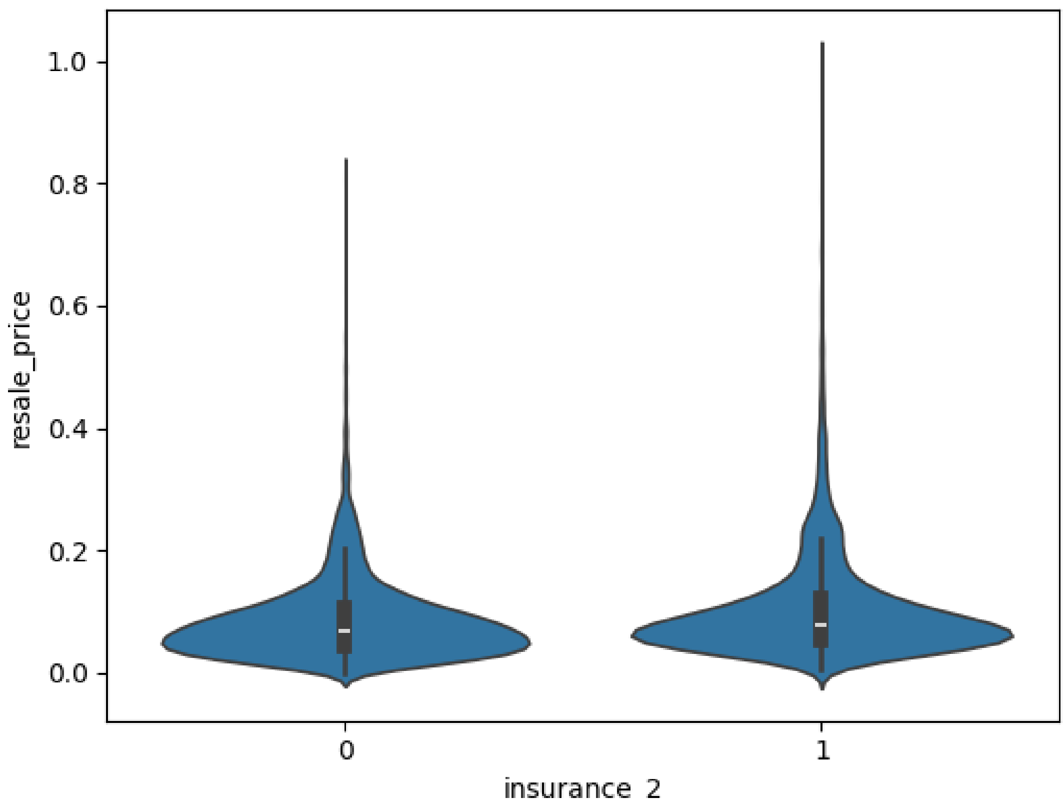
[1286]:
```
sns.violinplot(x="insurance_1", y="resale_price", data=df)
```

[1286]: <Axes: xlabel='insurance_1', ylabel='resale_price'>



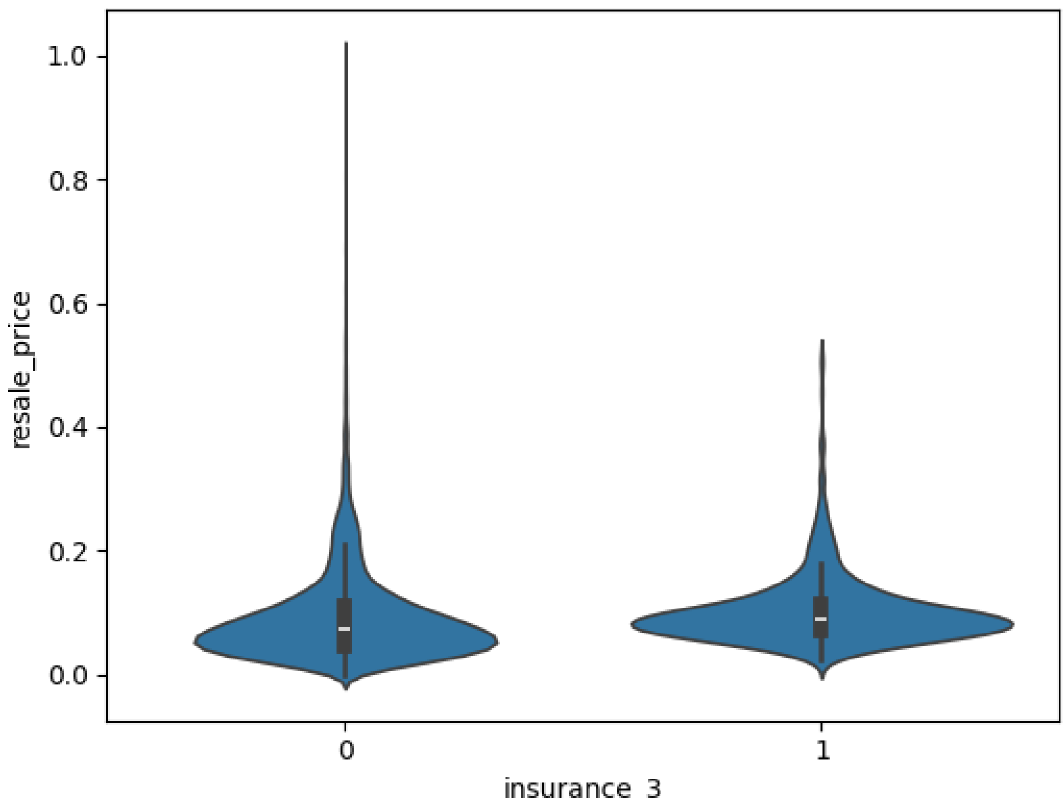
[1287]:
```
sns.violinplot(x="insurance_2", y="resale_price", data=df)
```

[1287]: <Axes: xlabel='insurance_2', ylabel='resale_price'>



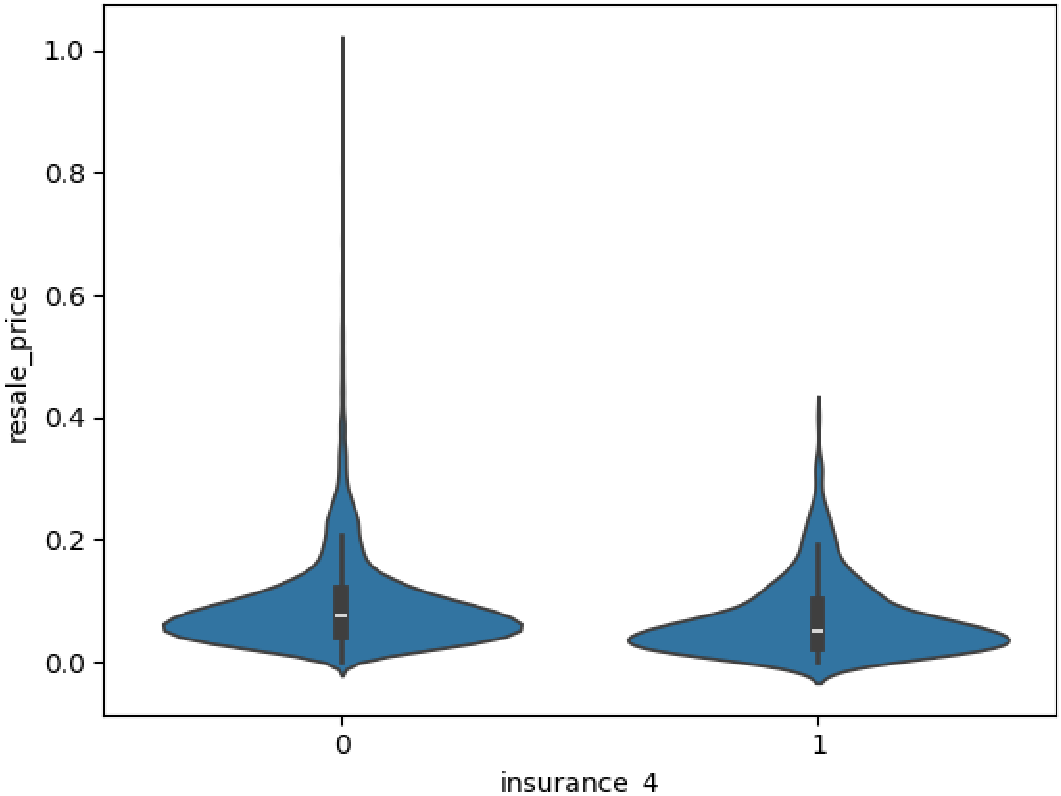
[1288]:
```
sns.violinplot(x="insurance_3", y="resale_price", data=df)
```

[1288]: <Axes: xlabel='insurance_3', ylabel='resale_price'>



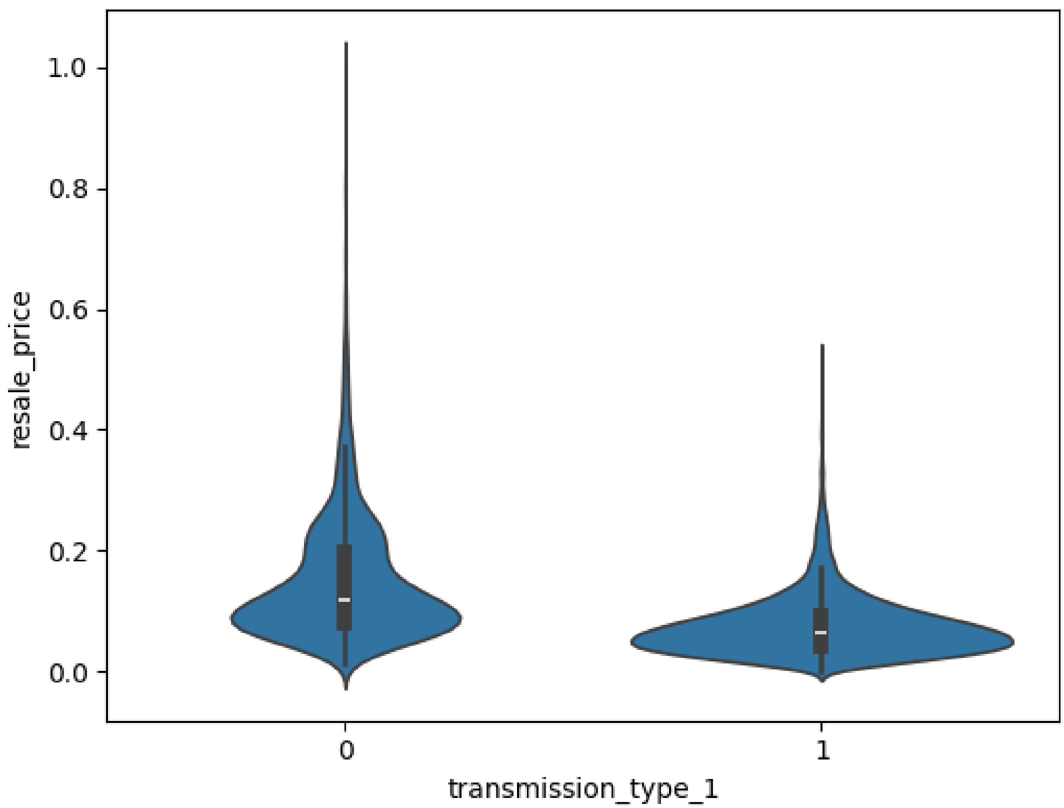
[1289]:
```
sns.violinplot(x="insurance_4", y="resale_price", data=df)
```

[1289]: <Axes: xlabel='insurance_4', ylabel='resale_price'>



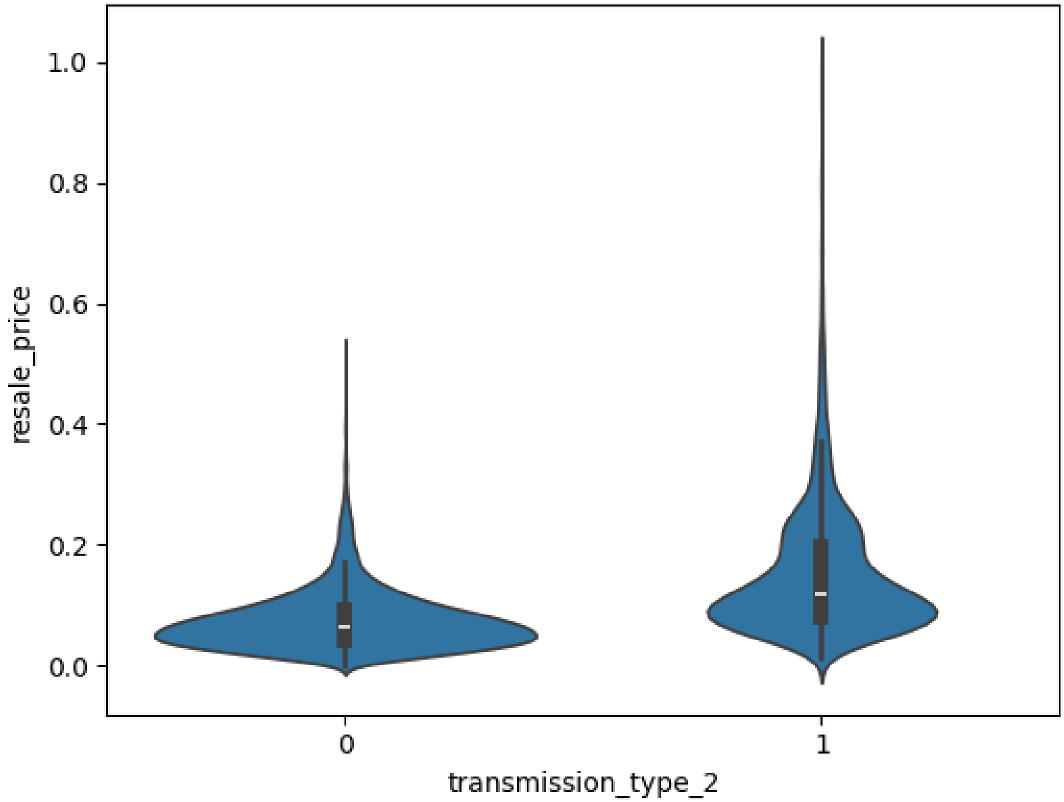
[1290]:
```
sns.violinplot(x="transmission_type_1", y="resale_price", data=df)
```

[1290]: <Axes: xlabel='transmission_type_1', ylabel='resale_price'>



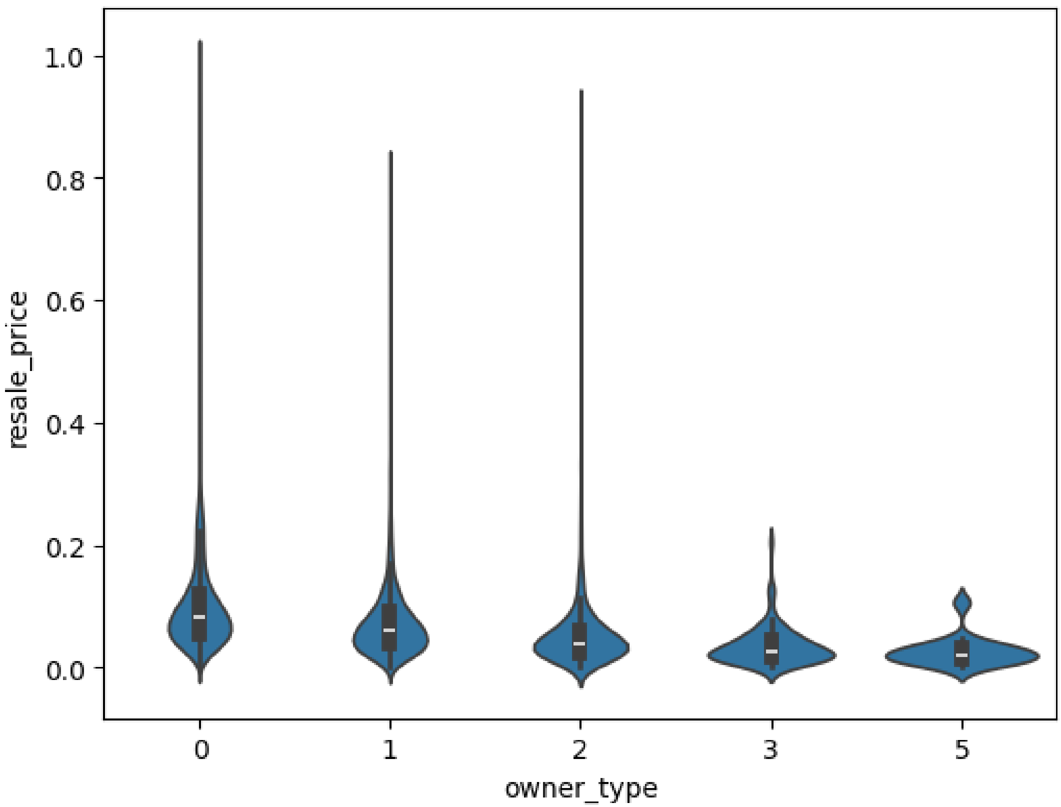

```
[1291]: sns.violinplot(x="transmission_type_2", y="resale_price", data=df)
```

```
[1291]: <Axes: xlabel='transmission_type_2', ylabel='resale_price'>
```



[1292]:
```
sns.violinplot(x="owner_type", y="resale_price", data=df)
```

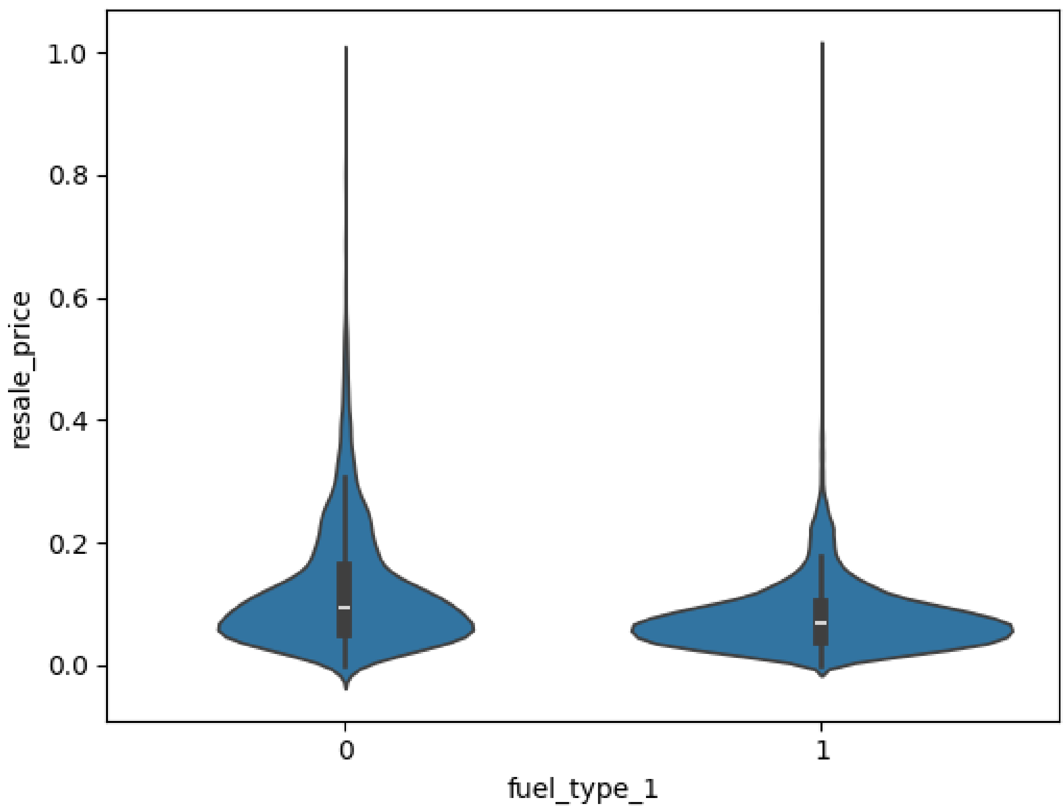
[1292]: <Axes: xlabel='owner_type', ylabel='resale_price'>



From the above graph we could see that the resale price of a car decreases when the number of owners increases

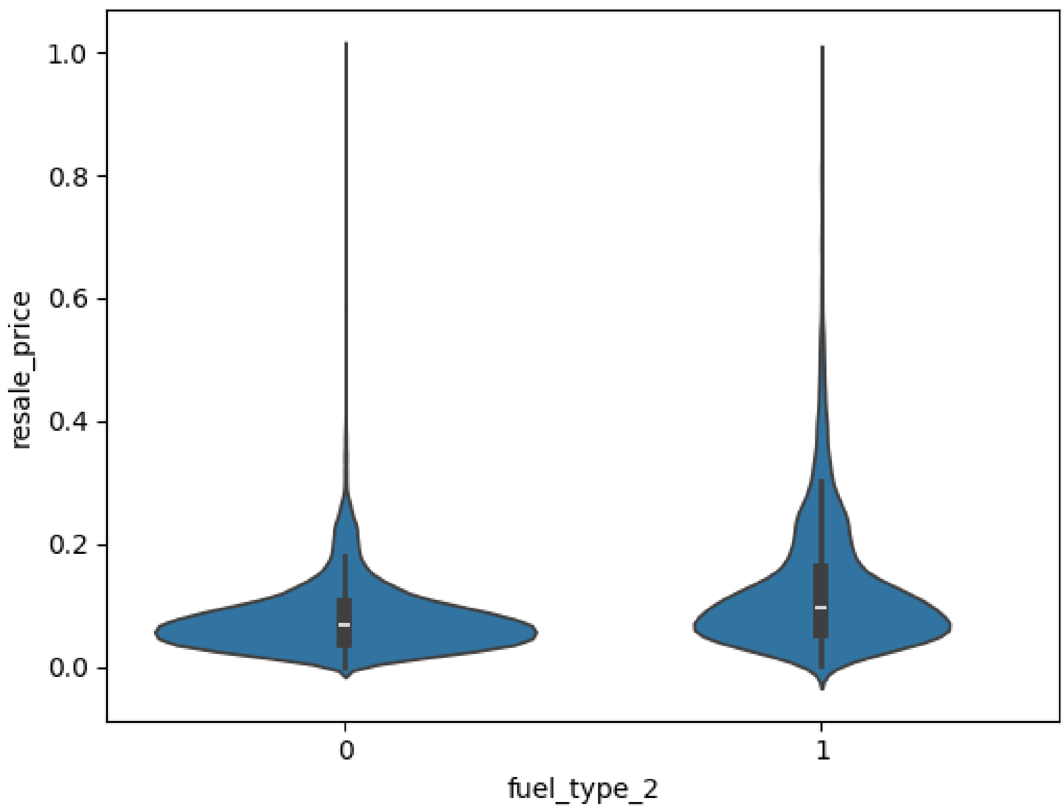
[1293]:
```
sns.violinplot(x="fuel_type_1", y="resale_price", data=df)
```

[1293]: <Axes: xlabel='fuel_type_1', ylabel='resale_price'>



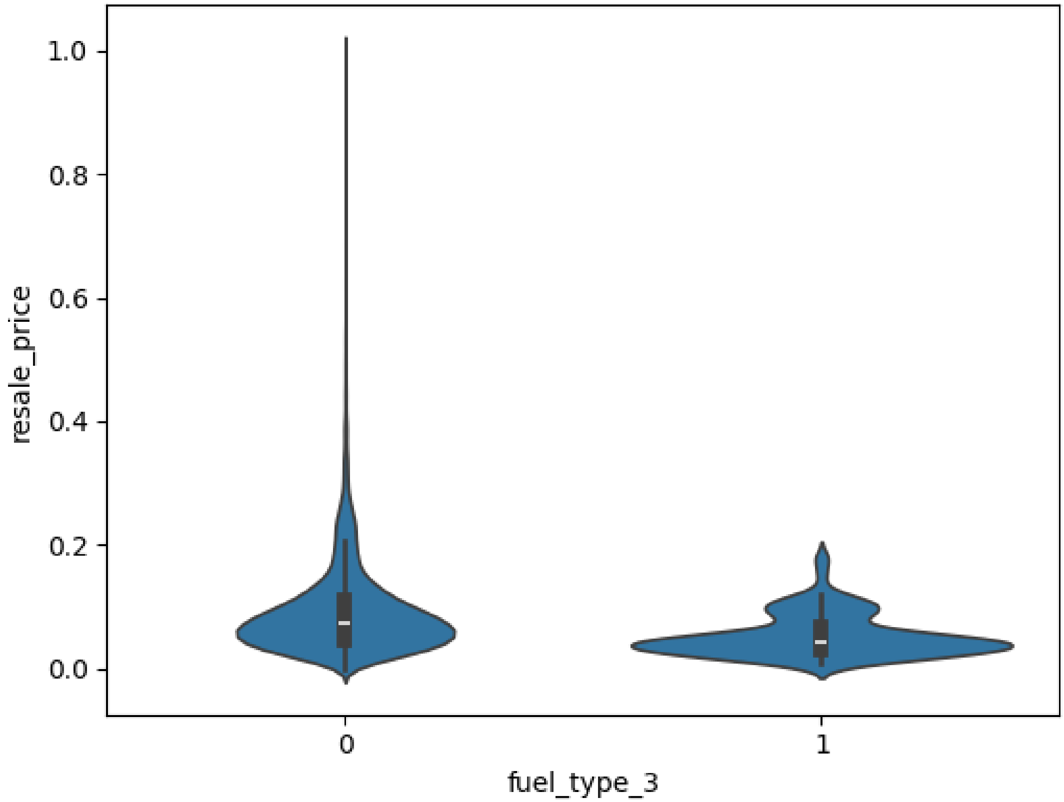
[1294]:
```
sns.violinplot(x="fuel_type_2", y="resale_price", data=df)
```

[1294]: <Axes: xlabel='fuel_type_2', ylabel='resale_price'>



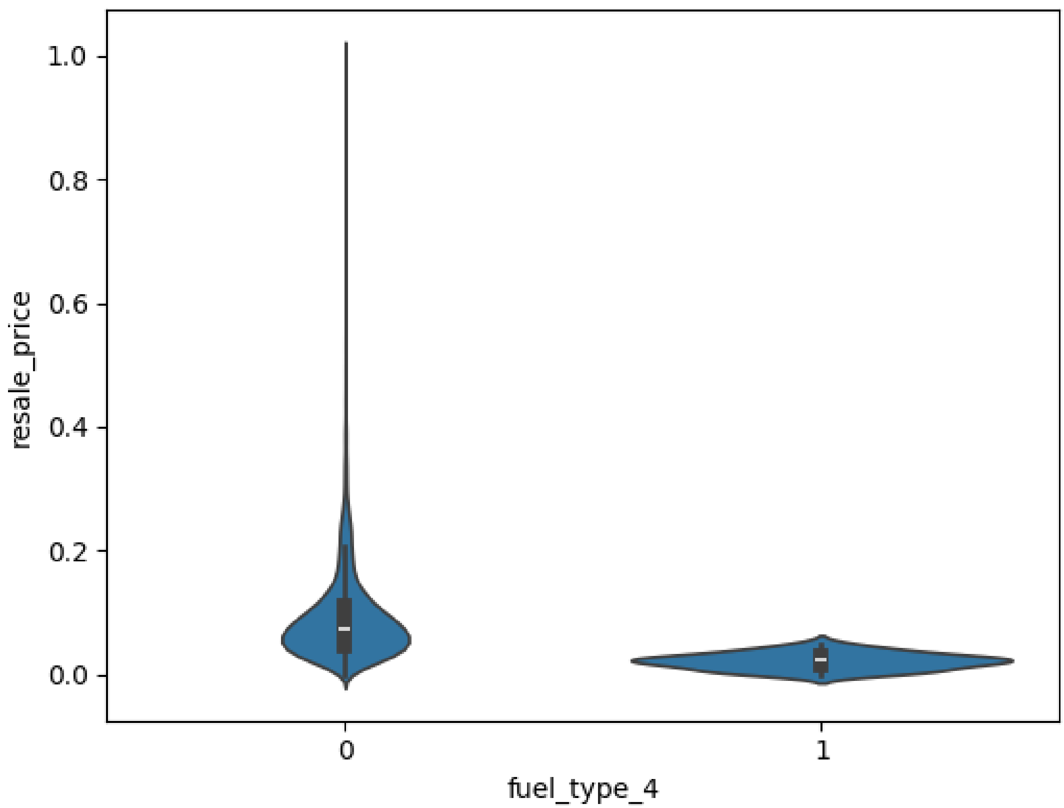
[1295]:
```
sns.violinplot(x="fuel_type_3", y="resale_price", data=df)
```

[1295]: <Axes: xlabel='fuel_type_3', ylabel='resale_price'>



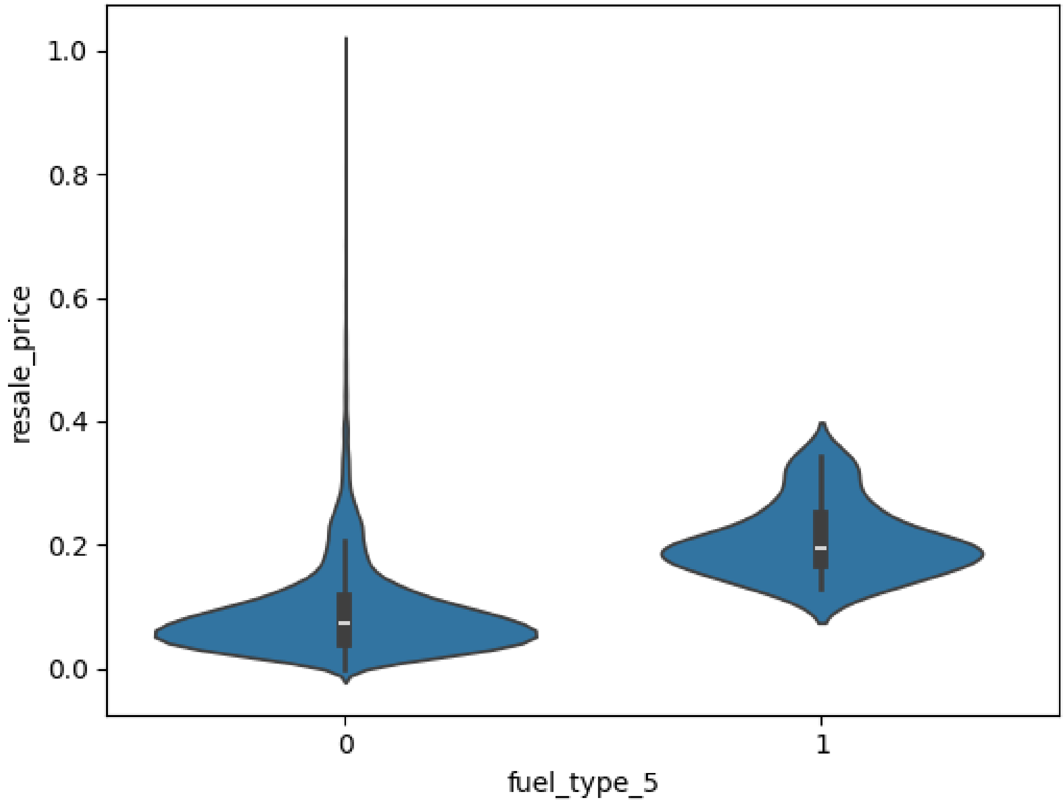
[1296]:
```
sns.violinplot(x="fuel_type_4", y="resale_price", data=df)
```

[1296]: <Axes: xlabel='fuel_type_4', ylabel='resale_price'>



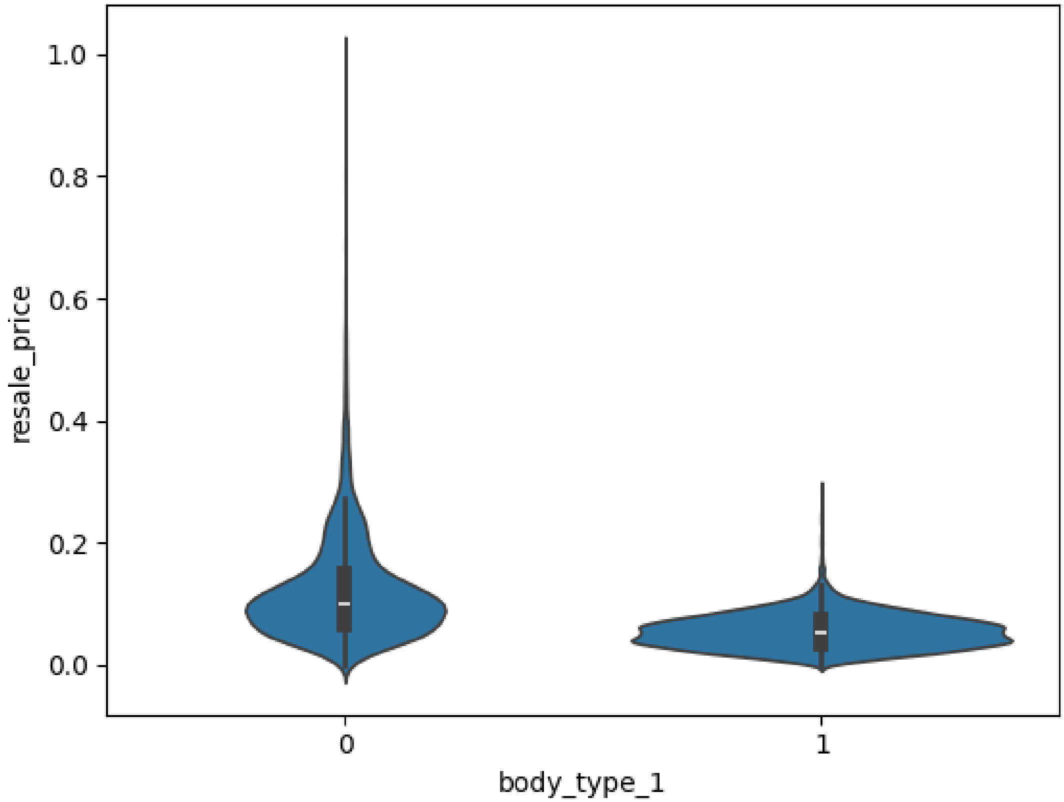
[1297]:
```
sns.violinplot(x="fuel_type_5", y="resale_price", data=df)
```

[1297]: <Axes: xlabel='fuel_type_5', ylabel='resale_price'>



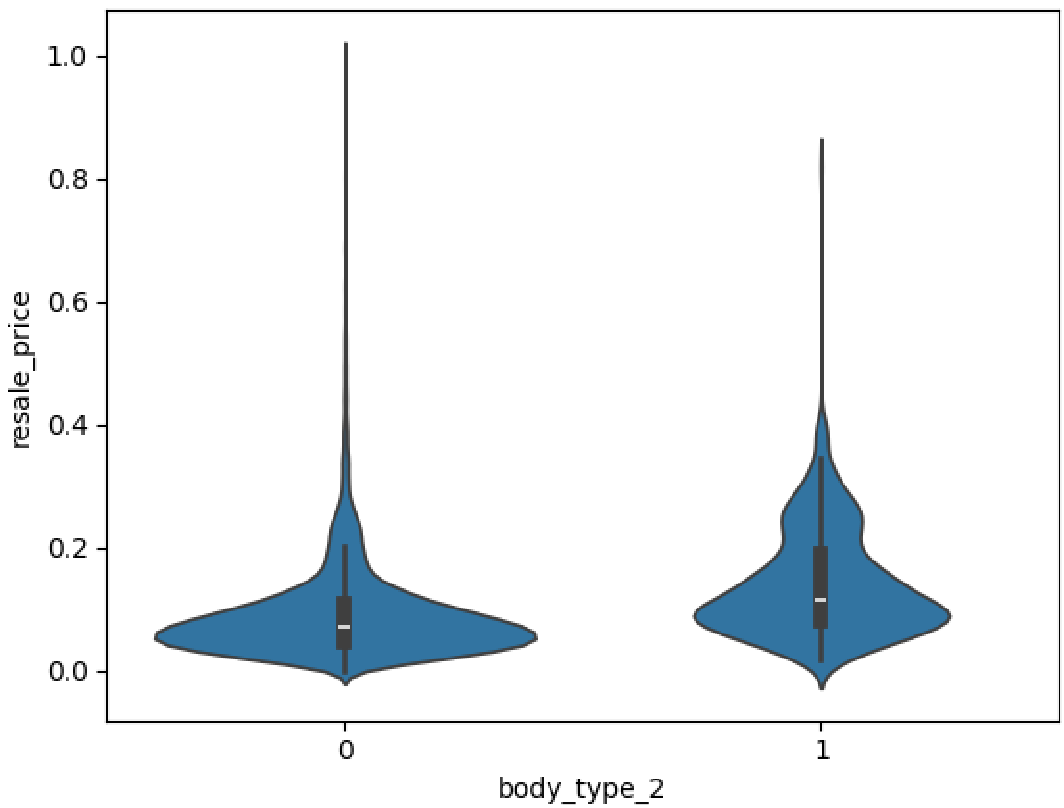
[1298]:
```
sns.violinplot(x="body_type_1", y="resale_price", data=df)
```

[1298]: <Axes: xlabel='body_type_1', ylabel='resale_price'>



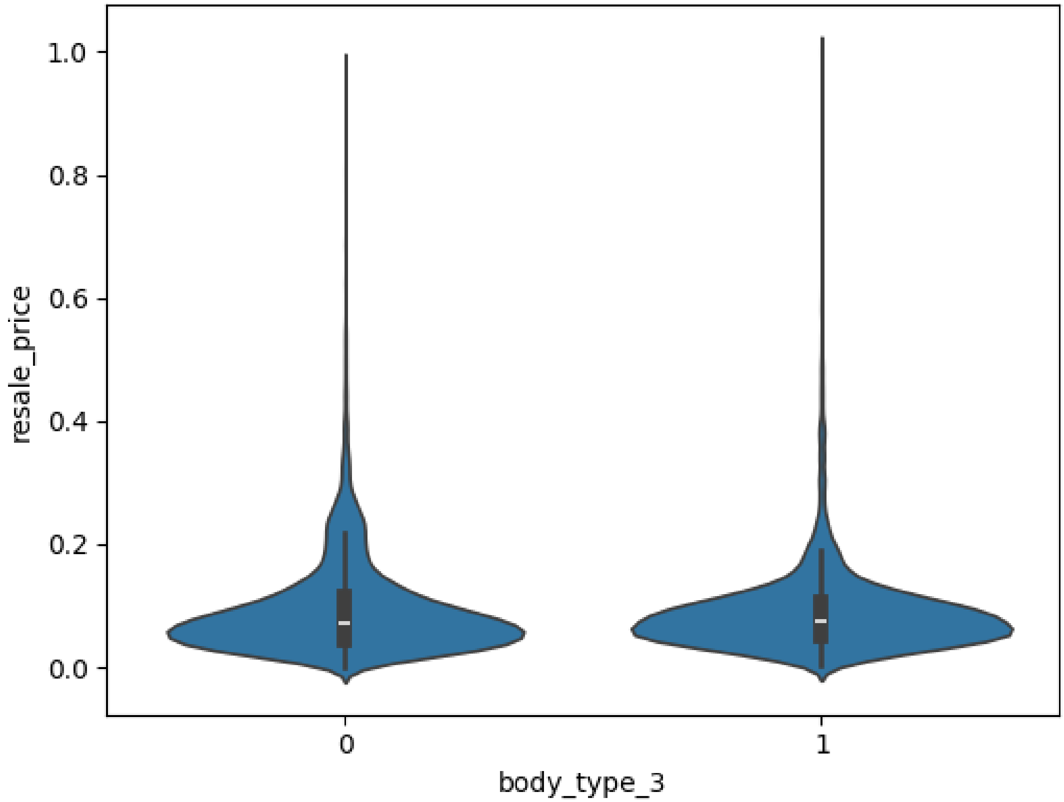

```
[1299]: sns.violinplot(x="body_type_2", y="resale_price", data=df)
```

```
[1299]: <Axes: xlabel='body_type_2', ylabel='resale_price'>
```



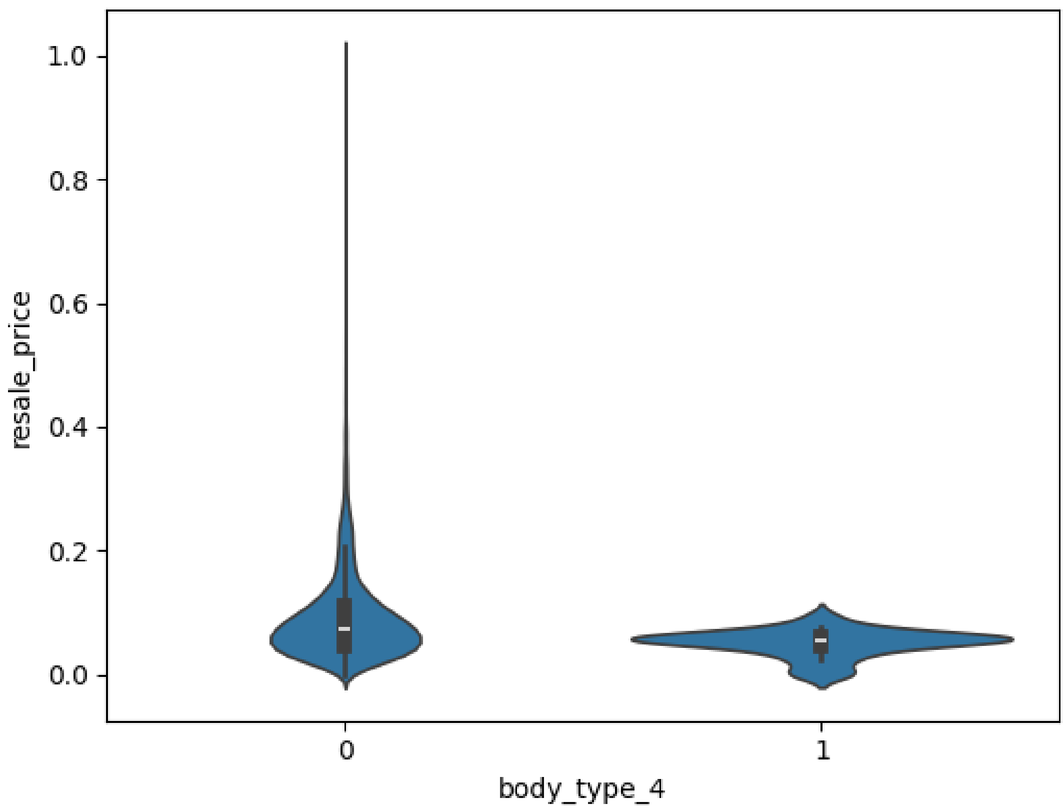

```
[1300]: sns.violinplot(x="body_type_3", y="resale_price", data=df)
```

```
[1300]: <Axes: xlabel='body_type_3', ylabel='resale_price'>
```



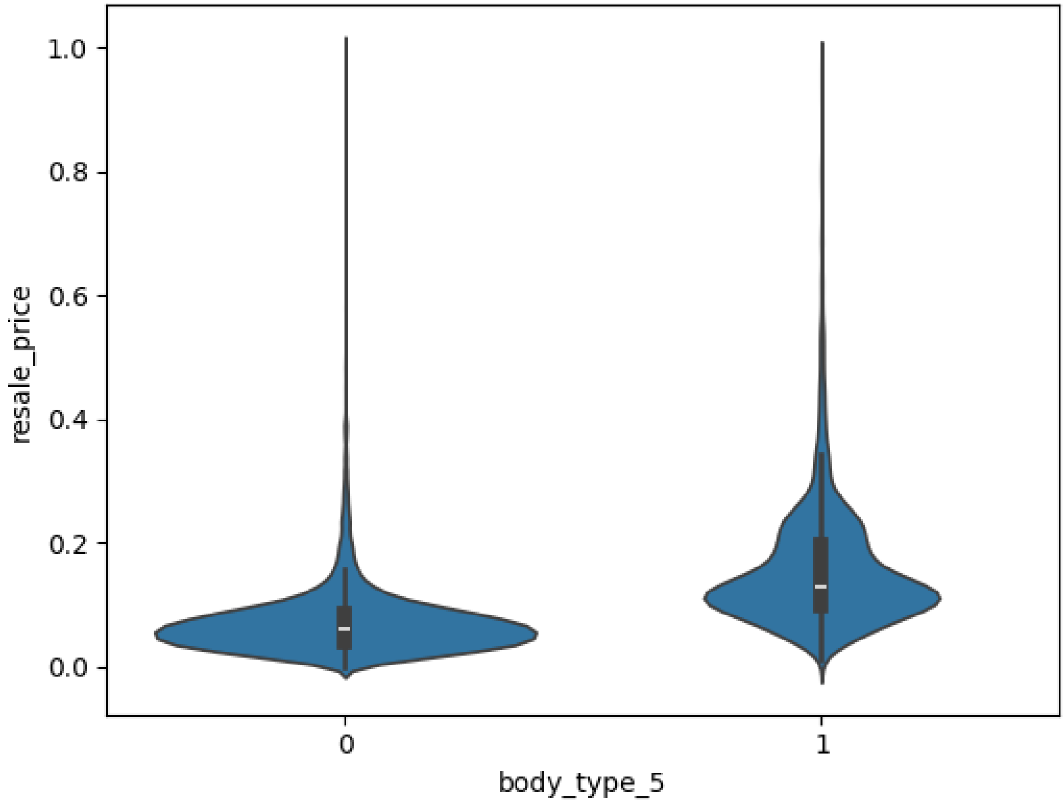

```
[1301]: sns.violinplot(x="body_type_4", y="resale_price", data=df)
```

```
[1301]: <Axes: xlabel='body_type_4', ylabel='resale_price'>
```



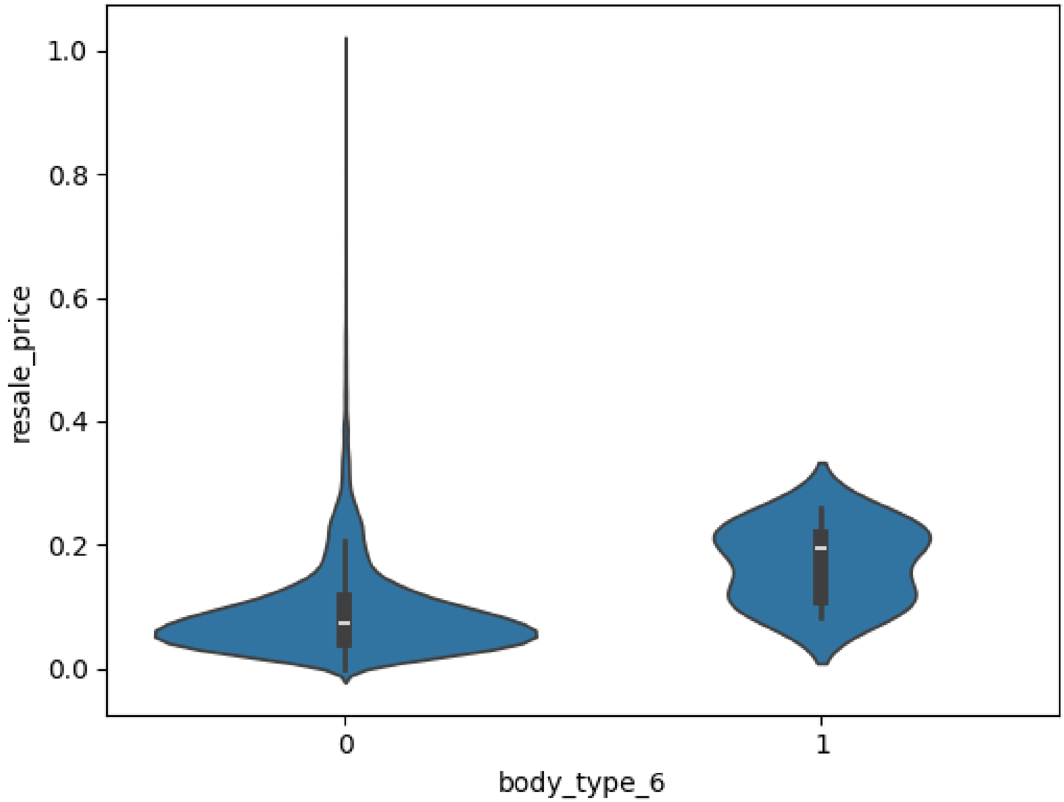

```
[1302]: sns.violinplot(x="body_type_5", y="resale_price", data=df)
```

```
[1302]: <Axes: xlabel='body_type_5', ylabel='resale_price'>
```



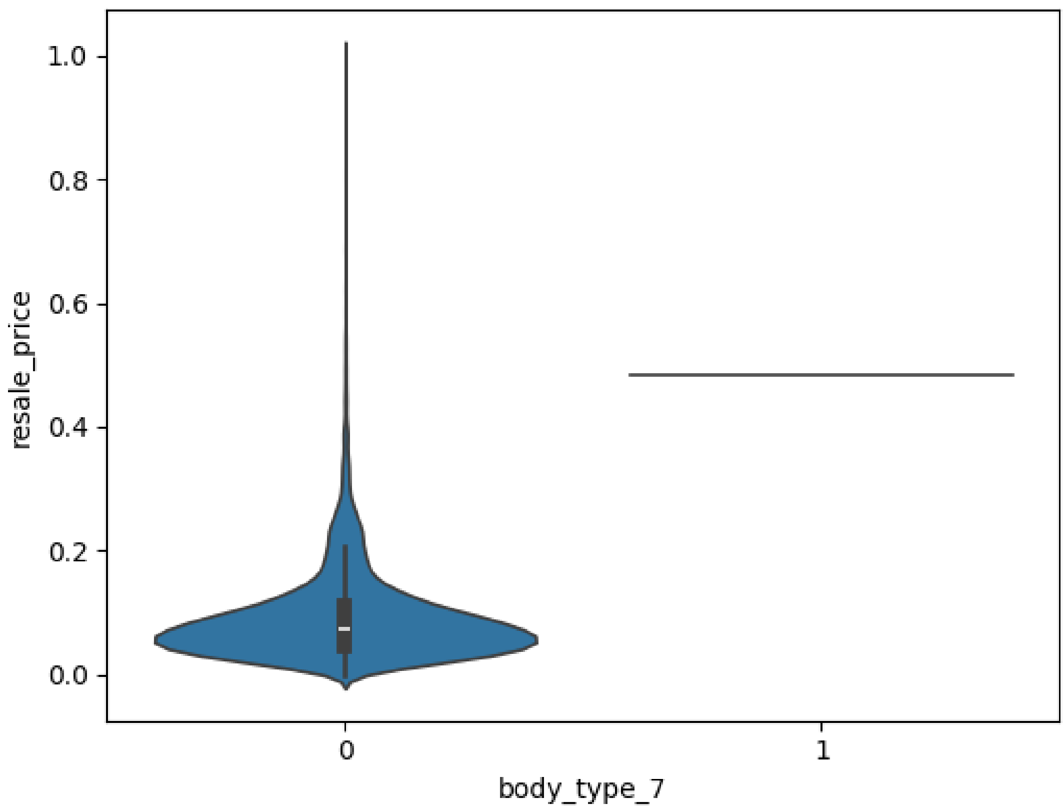
[1303]:
```
sns.violinplot(x="body_type_6", y="resale_price", data=df)
```

[1303]: <Axes: xlabel='body_type_6', ylabel='resale_price'>



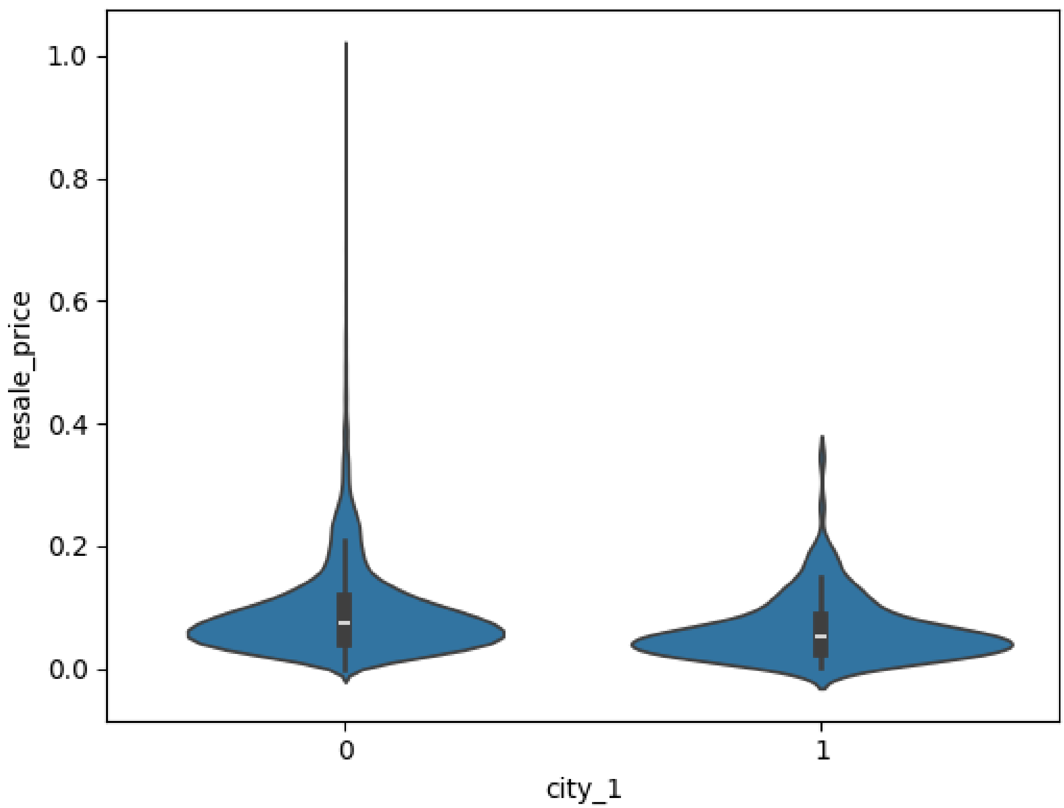

```
[1304]: sns.violinplot(x="body_type_7", y="resale_price", data=df)
```

```
[1304]: <Axes: xlabel='body_type_7', ylabel='resale_price'>
```



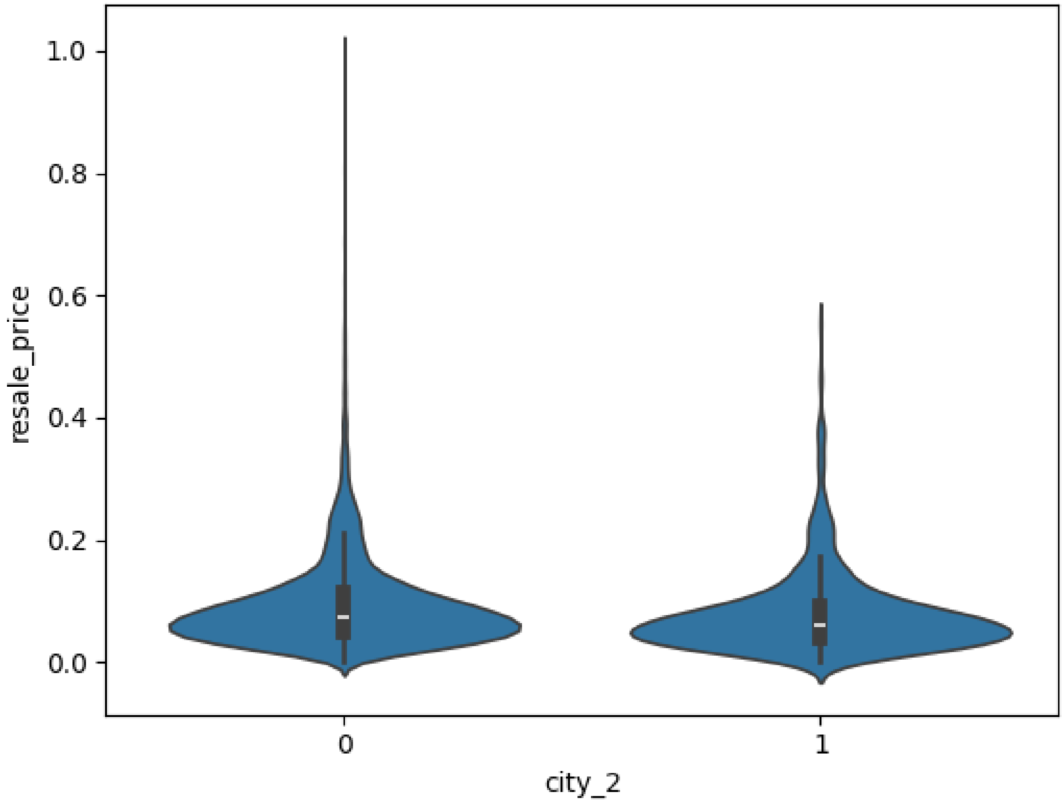

```
[1305]: sns.violinplot(x="city_1", y="resale_price", data=df)
```

```
[1305]: <Axes: xlabel='city_1', ylabel='resale_price'>
```



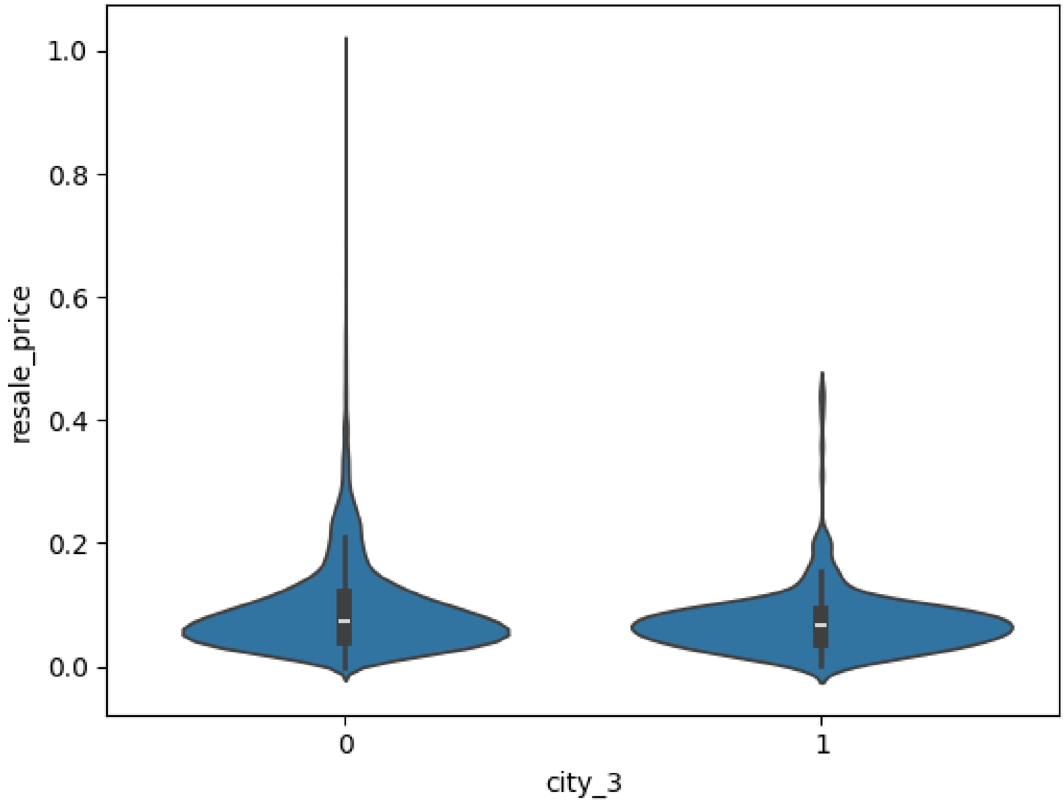

```
[1306]: sns.violinplot(x="city_2", y="resale_price", data=df)
```

```
[1306]: <Axes: xlabel='city_2', ylabel='resale_price'>
```



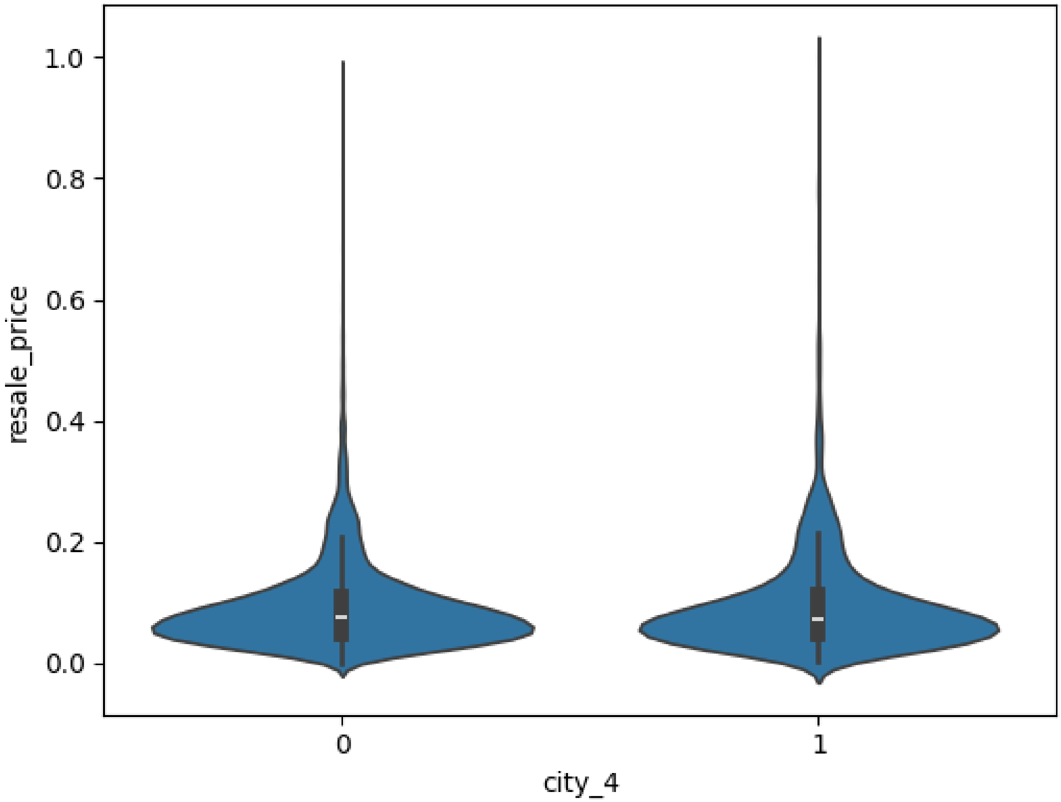
[1307]:
```
sns.violinplot(x="city_3", y="resale_price", data=df)
```

[1307]: <Axes: xlabel='city_3', ylabel='resale_price'>



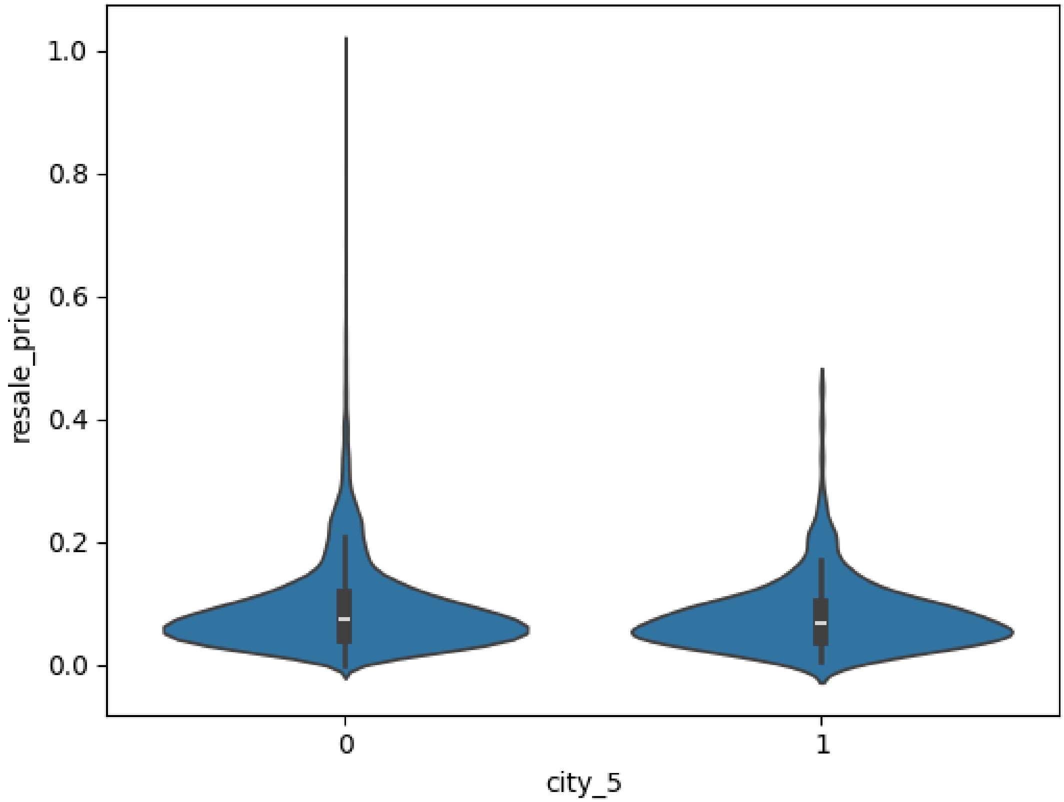
[1308]:
```
sns.violinplot(x="city_4", y="resale_price", data=df)
```

[1308]:
```
<Axes: xlabel='city_4', ylabel='resale_price'>
```



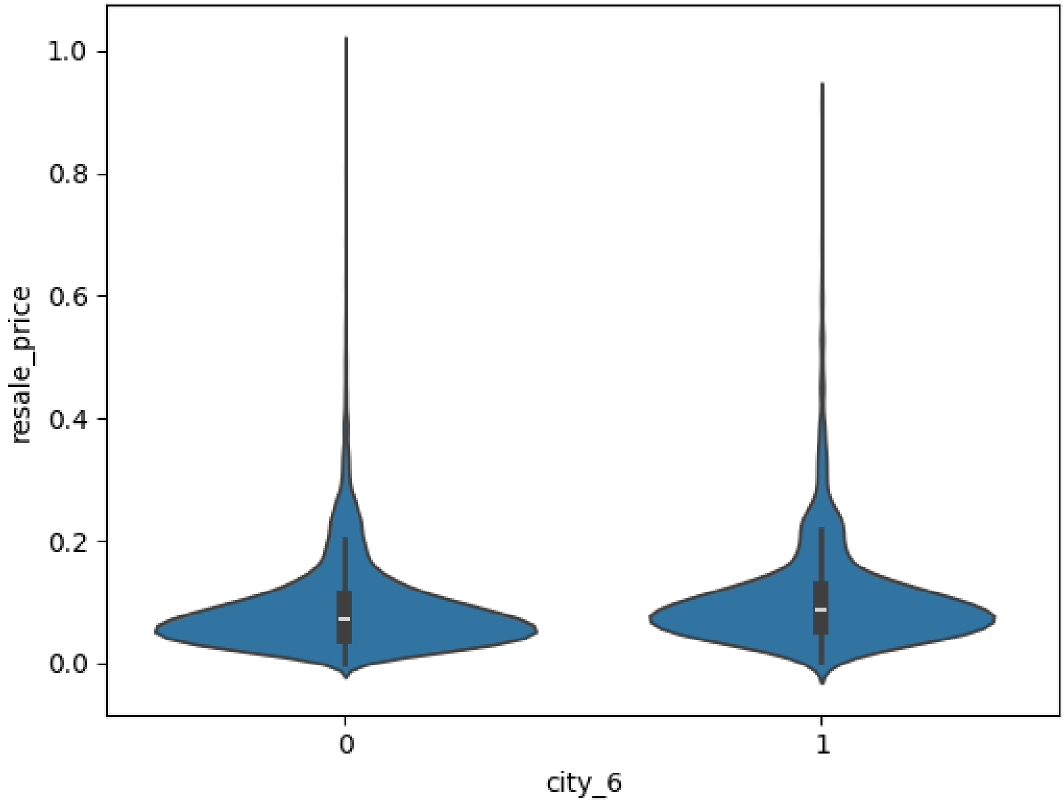
[1309]:
```
sns.violinplot(x="city_5", y="resale_price", data=df)
```

[1309]: <Axes: xlabel='city_5', ylabel='resale_price'>



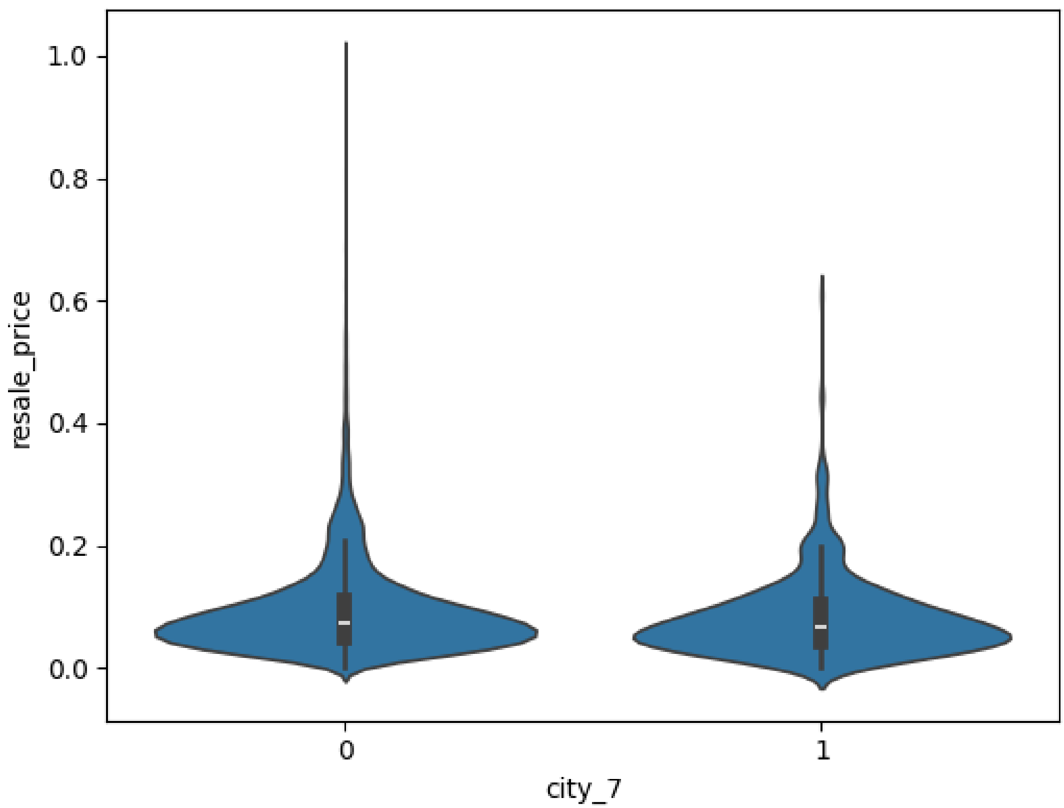
[1310]:
```
sns.violinplot(x="city_6", y="resale_price", data=df)
```

[1310]: <Axes: xlabel='city_6', ylabel='resale_price'>



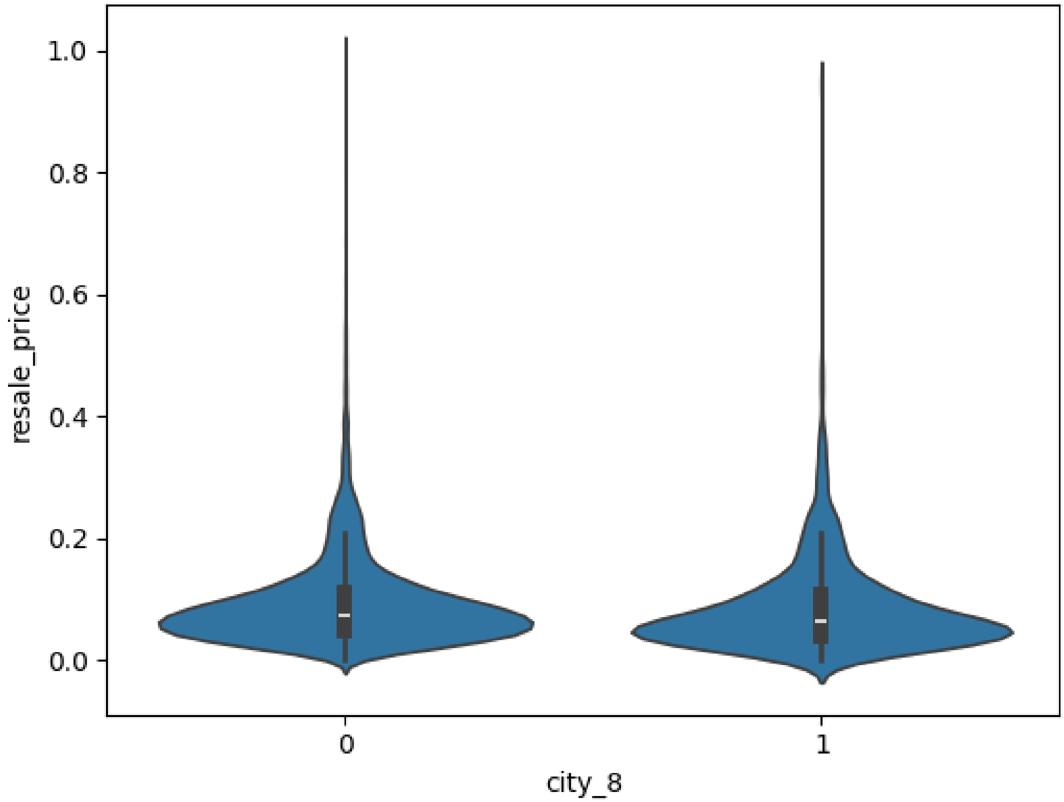
[1311]:
```
sns.violinplot(x="city_7", y="resale_price", data=df)
```

[1311]: <Axes: xlabel='city_7', ylabel='resale_price'>



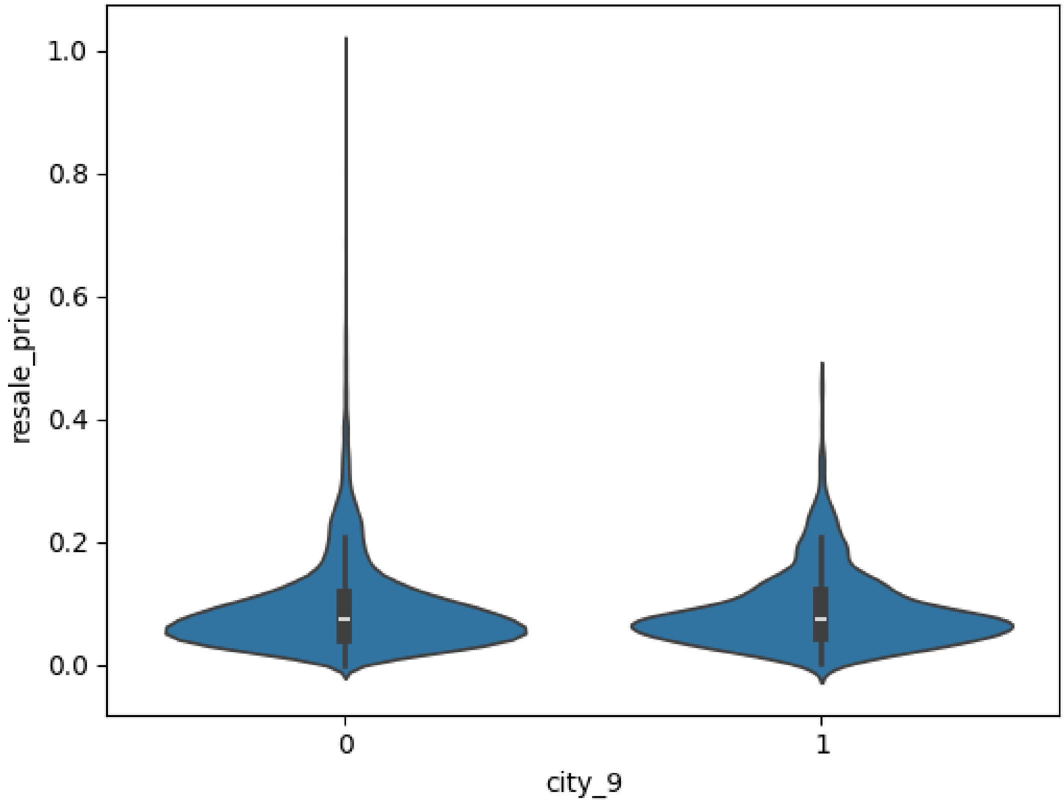
[1312]:
```
sns.violinplot(x="city_8", y="resale_price", data=df)
```

[1312]:
```
<Axes: xlabel='city_8', ylabel='resale_price'>
```



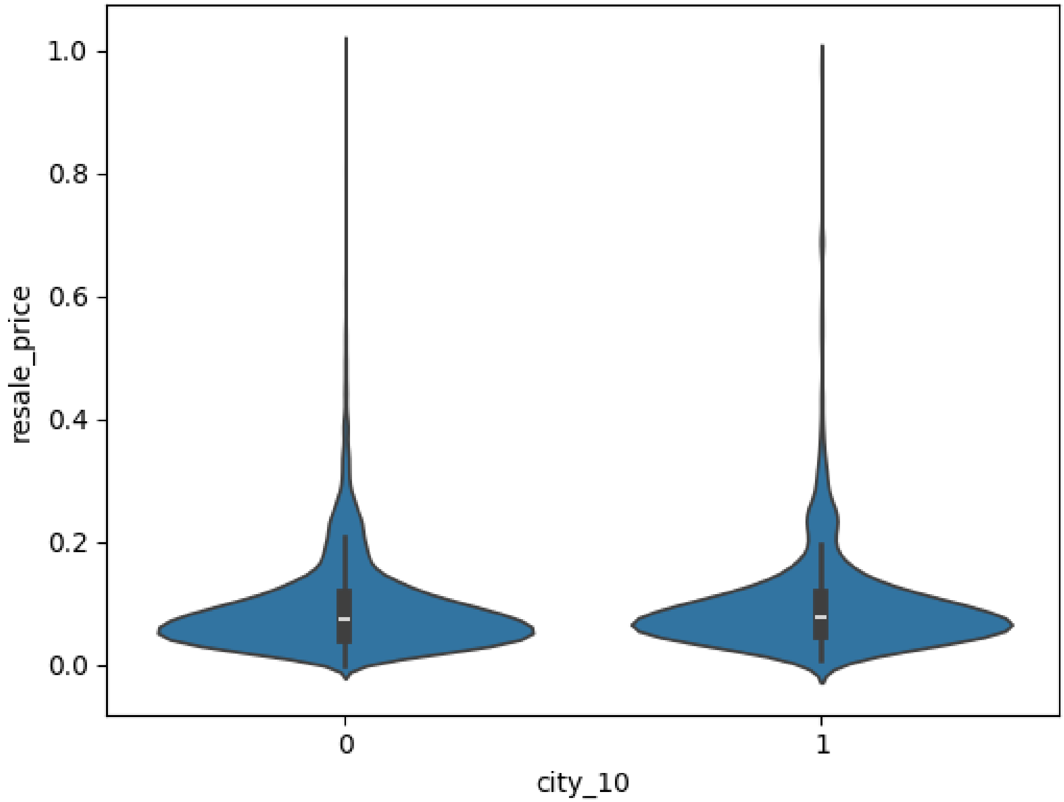
[1313]:
```
sns.violinplot(x="city_9", y="resale_price", data=df)
```

[1313]: <Axes: xlabel='city_9', ylabel='resale_price'>



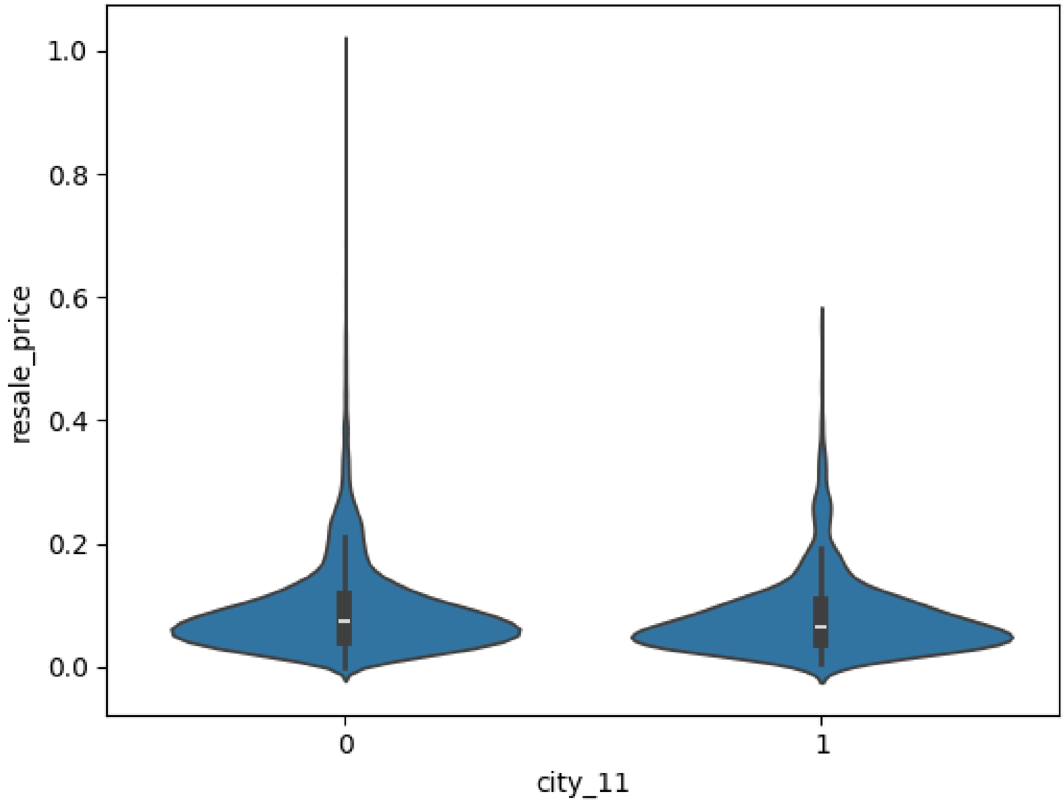
[1314]:
```
sns.violinplot(x="city_10", y="resale_price", data=df)
```

[1314]: <Axes: xlabel='city_10', ylabel='resale_price'>



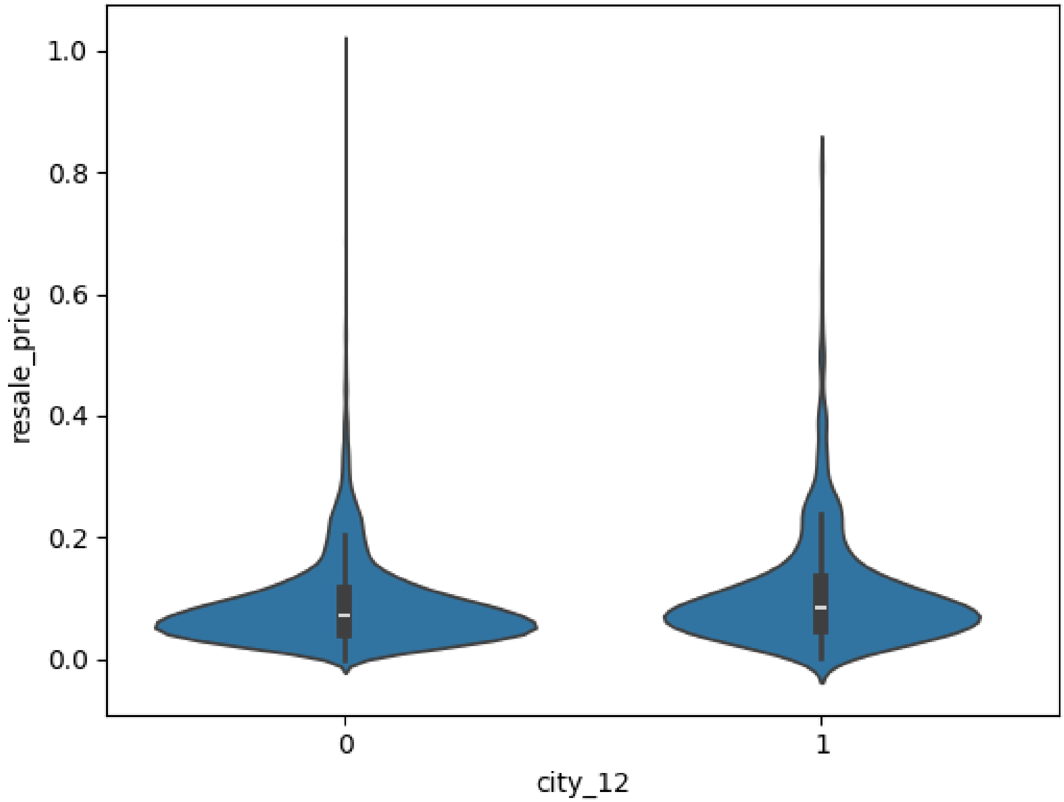
[1315]:
```
sns.violinplot(x="city_11", y="resale_price", data=df)
```

[1315]: <Axes: xlabel='city_11', ylabel='resale_price'>



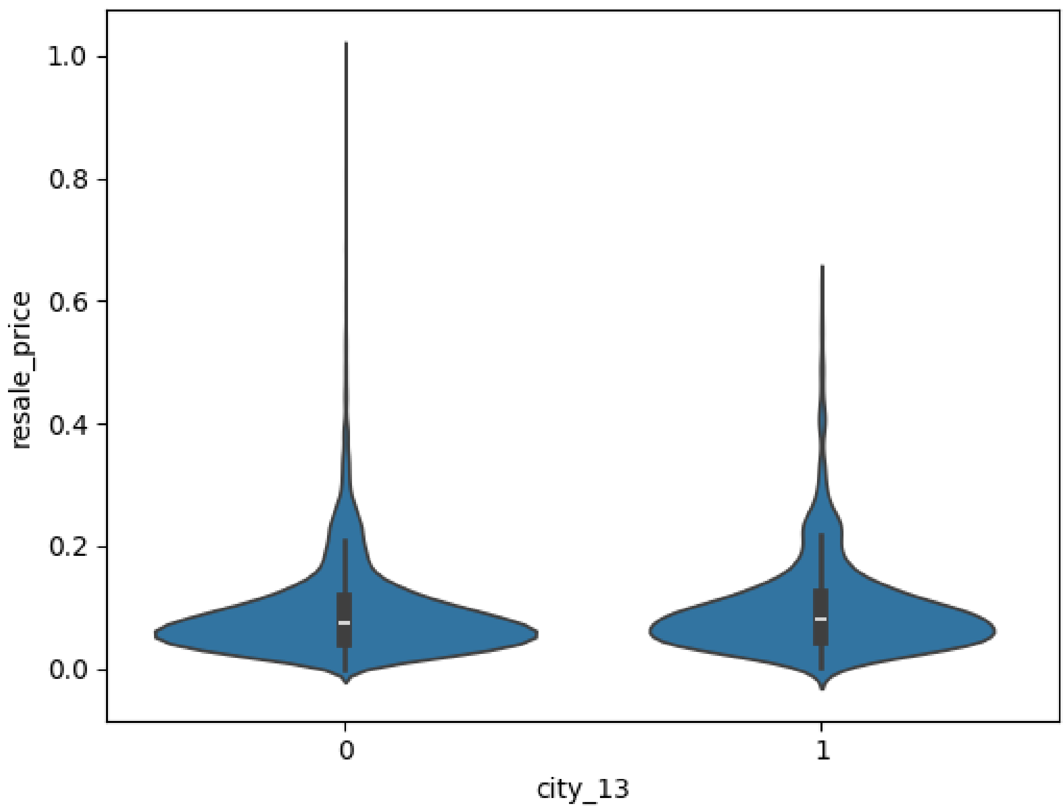
[1316]:
```
sns.violinplot(x="city_12", y="resale_price", data=df)
```

[1316]: <Axes: xlabel='city_12', ylabel='resale_price'>



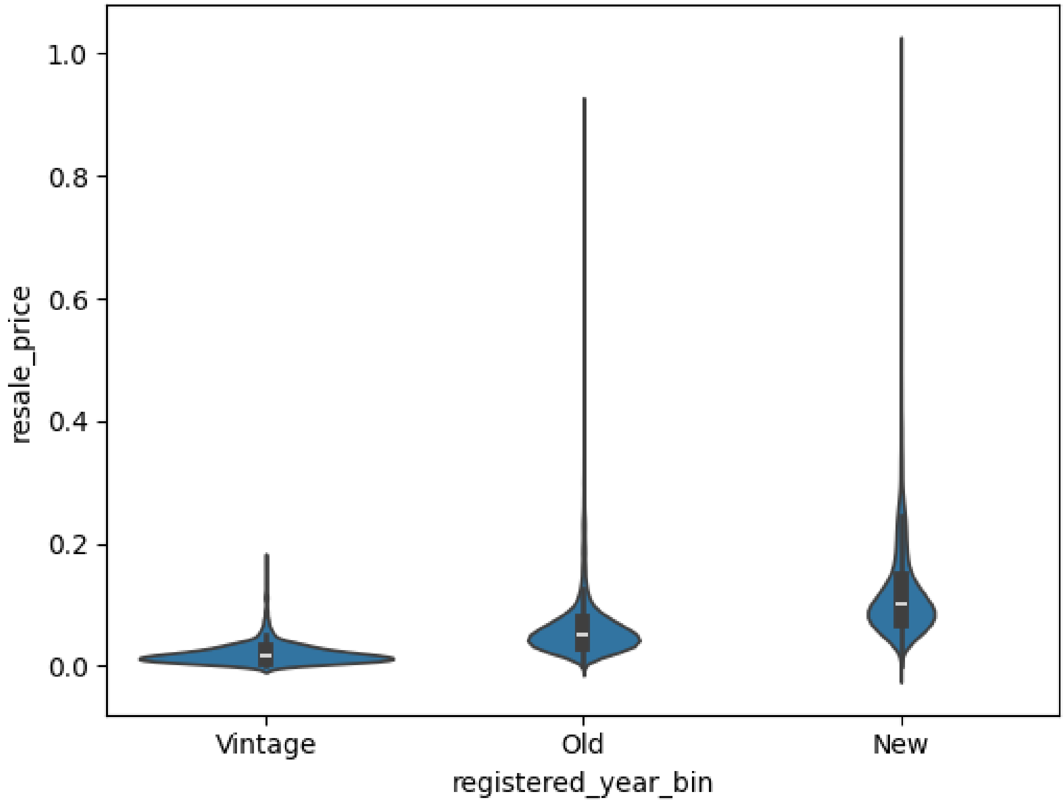

```
[1317]: sns.violinplot(x="city_13", y="resale_price", data=df)
```

```
[1317]: <Axes: xlabel='city_13', ylabel='resale_price'>
```



[1318]:
```
sns.violinplot(x="registered_year_bin", y="resale_price", data=df)
```

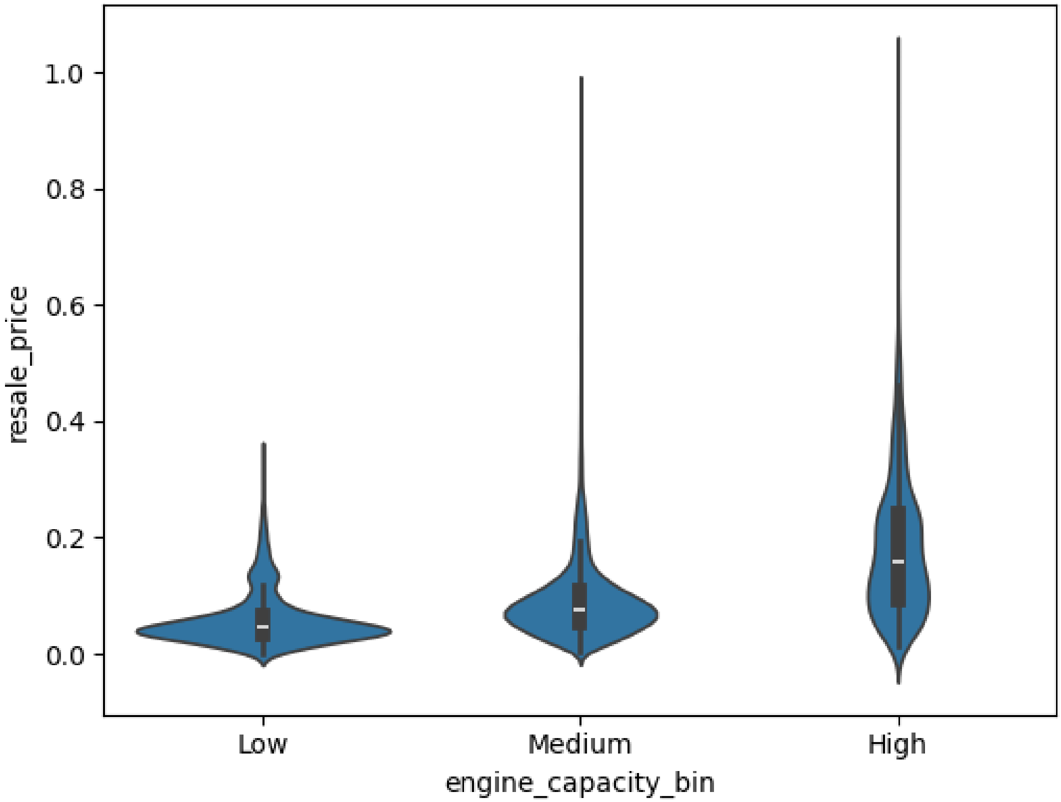
[1318]: <Axes: xlabel='registered_year_bin', ylabel='resale_price'>



From the above graph we could see that the resale price of a car increases when the car is new. The number of resale cars in the dataset is vintage

[1319]:
```
sns.violinplot(x="engine_capacity_bin", y="resale_price", data=df)
```

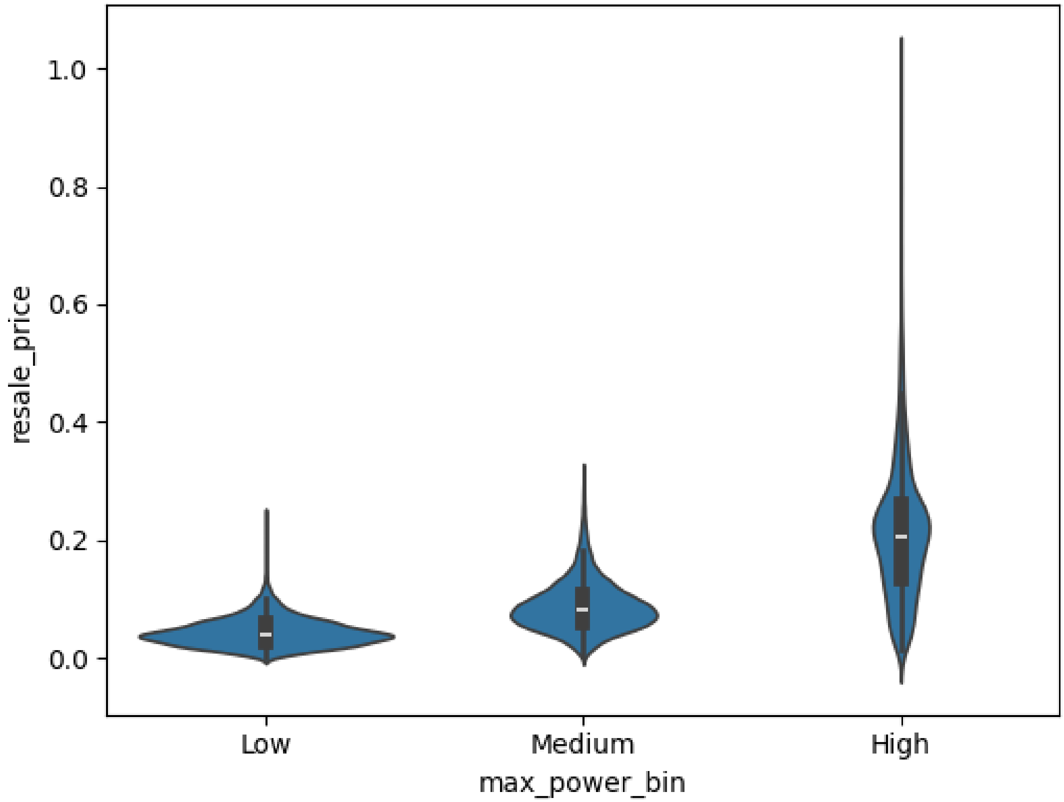
[1319]: <Axes: xlabel='engine_capacity_bin', ylabel='resale_price'>



From the above graph we could see that the resale price of a car increases when the engine capacity is high. The number of resale cars in the dataset has low engine capacity

[1320]:
```
sns.violinplot(x="max_power_bin", y="resale_price", data=df)
```

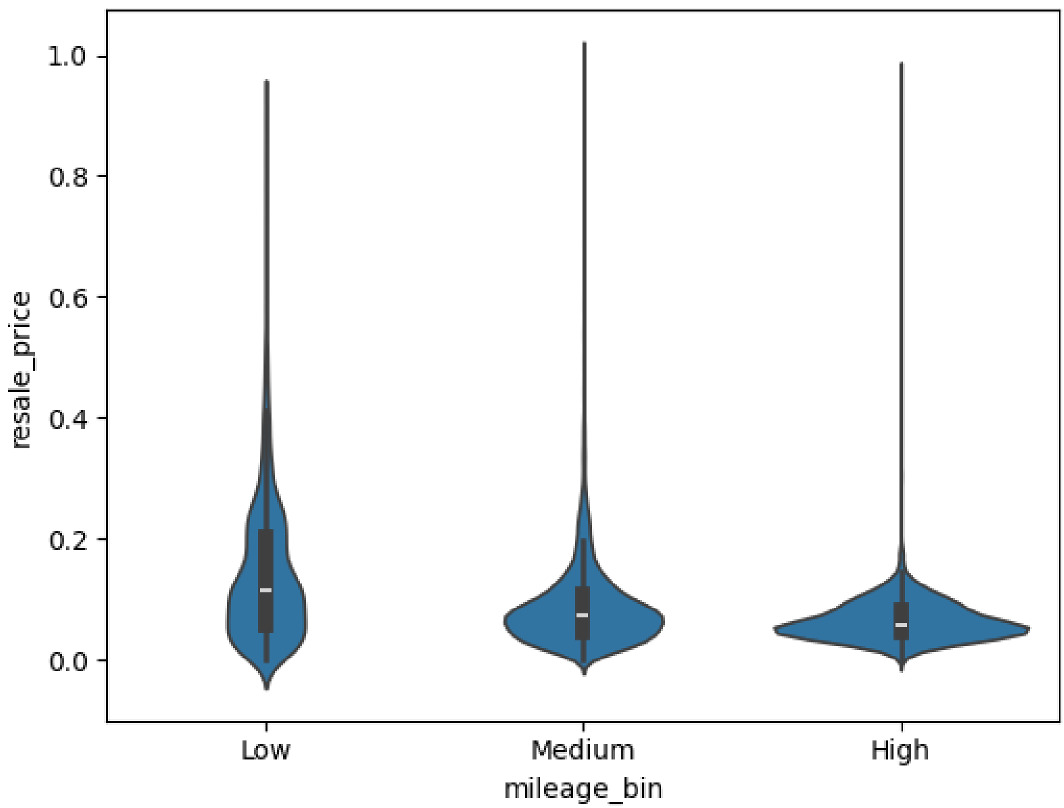
[1320]: <Axes: xlabel='max_power_bin', ylabel='resale_price'>



From the above graph we could see that the resale price of a car increases when the max power is high. The number of resale cars in the dataset has low max power

[1321]:
```
sns.violinplot(x="mileage_bin", y="resale_price", data=df)
```

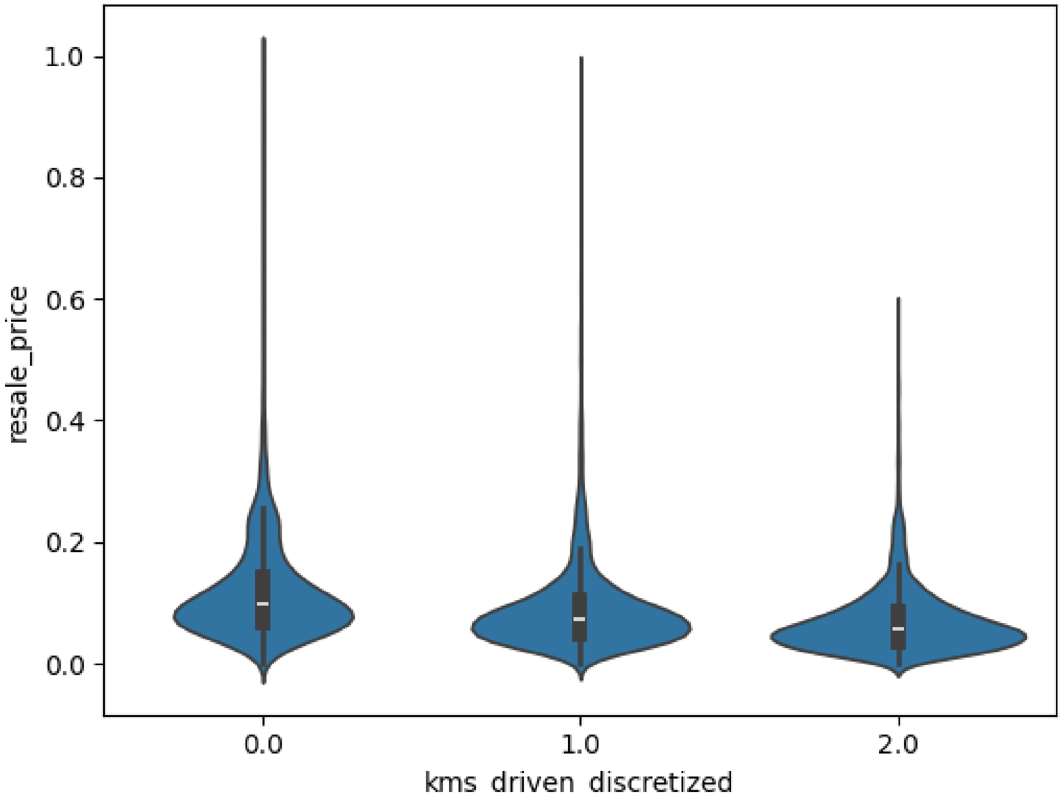
[1321]: `<Axes: xlabel='mileage_bin', ylabel='resale_price'>`



From the above graph we could see that the resale price of a car decreases when the mileage is high. The number of resale cars in the dataset has low to medium mileage

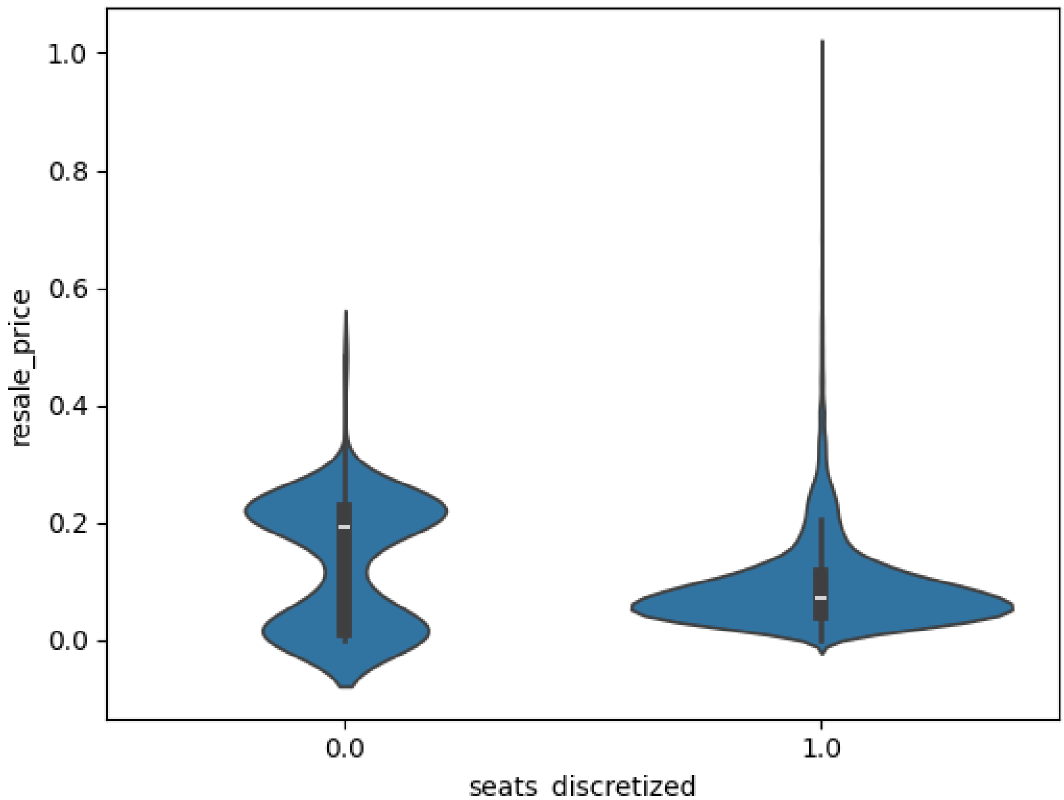
[1322]:
```
sns.violinplot(x="kms_driven_discretized", y="resale_price", data=df)
```

[1322]: <Axes: xlabel='kms_driven_discretized', ylabel='resale_price'>



[1323]:
```
sns.violinplot(x="seats_discretized", y="resale_price", data=df)
```

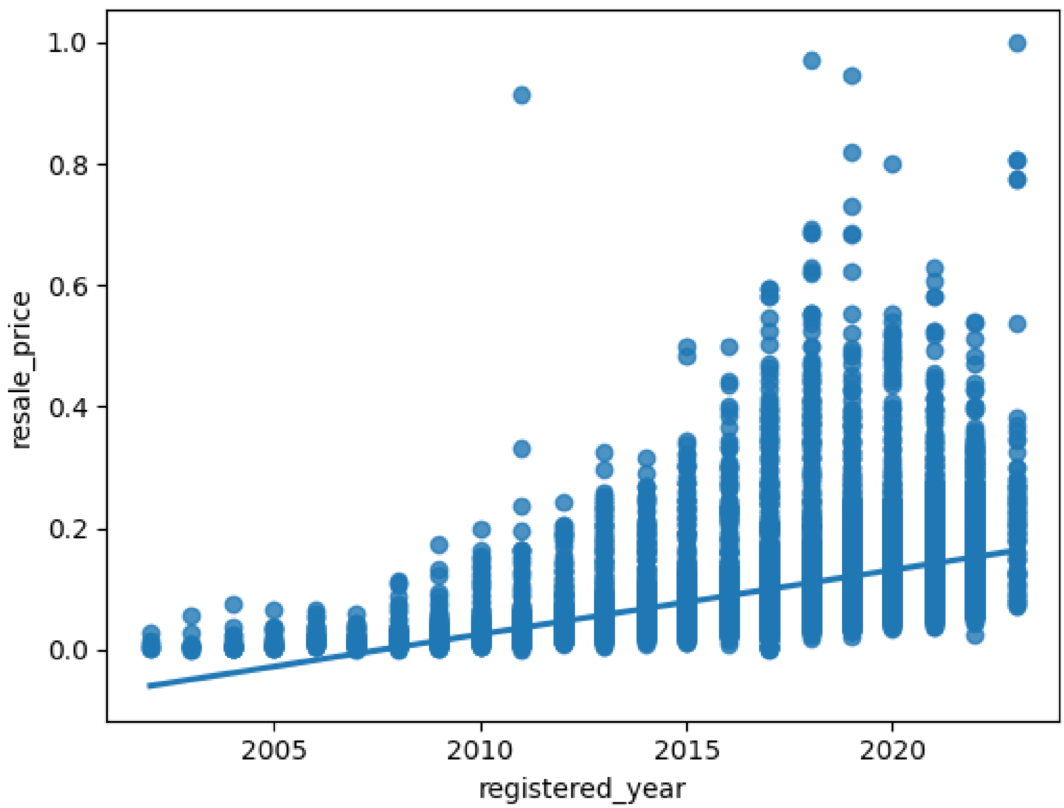
[1323]: <Axes: xlabel='seats_discretized', ylabel='resale_price'>



Lets use reg plot to see the relationship between the numerical attributes and resale_price

[1324]:
```
sns.regplot(x="registered_year", y="resale_price", data=df)
```

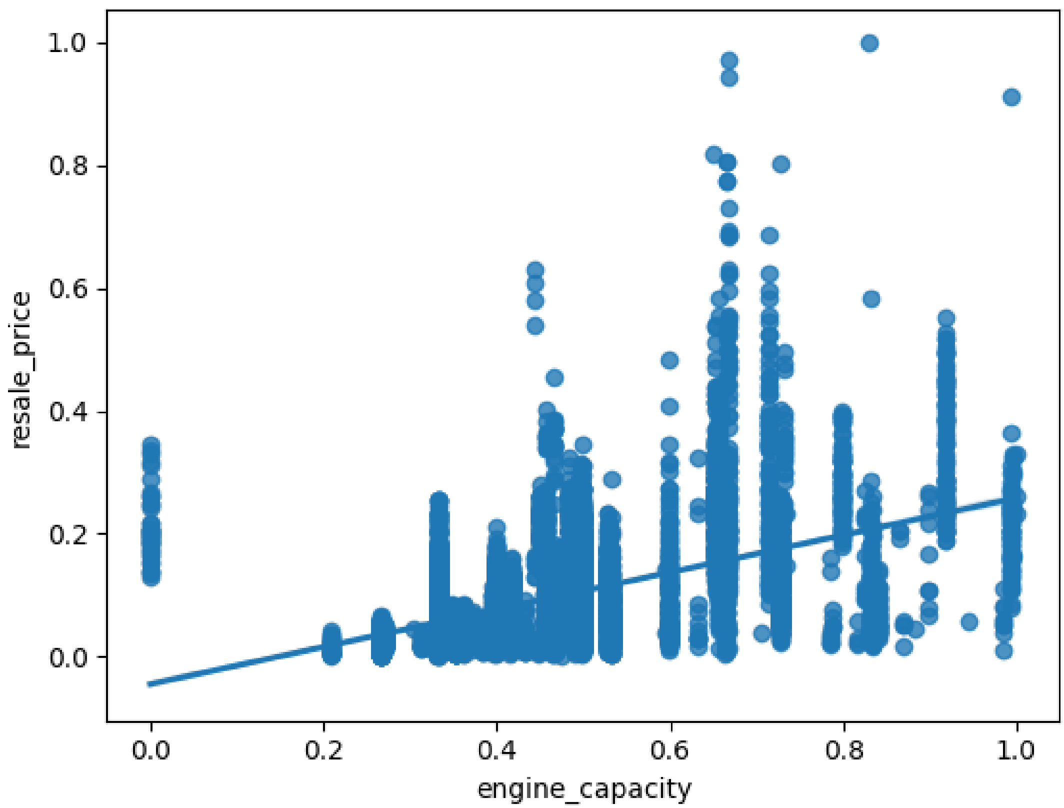
[1324]: <Axes: xlabel='registered_year', ylabel='resale_price'>



From the above we could infer that the recently registered car has high resale price.

[1325]:
```
sns.regplot(x="engine_capacity", y="resale_price", data=df)
```

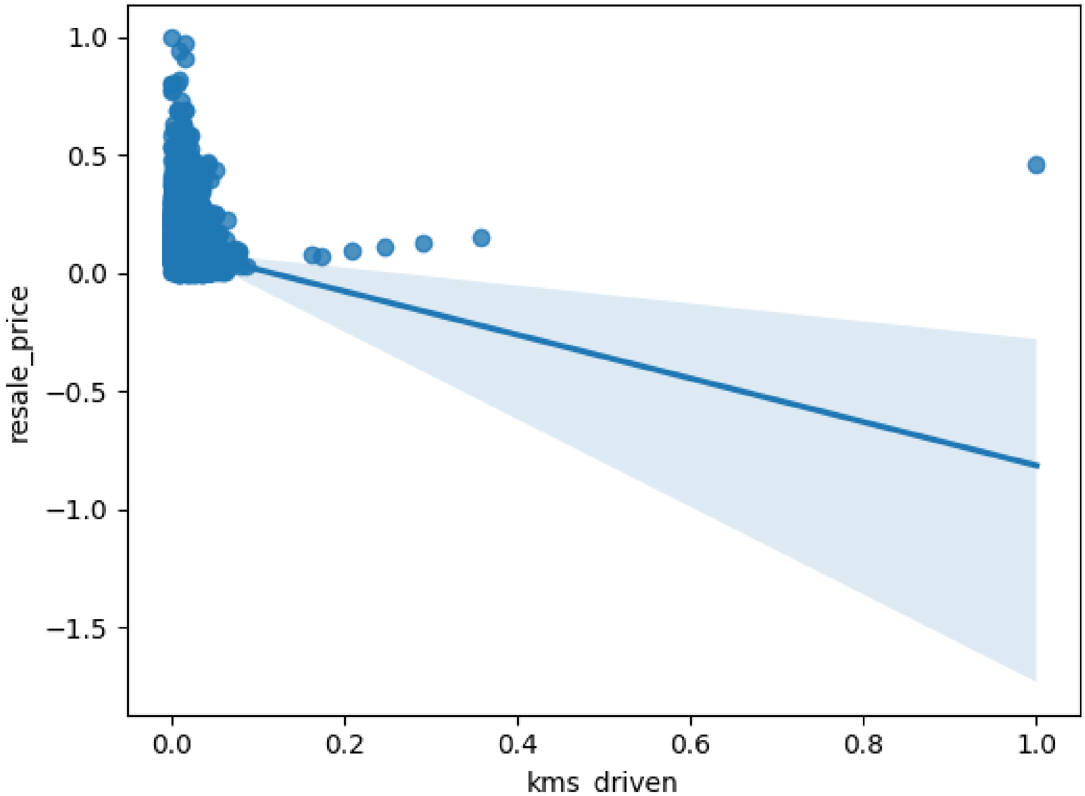
[1325]: <Axes: xlabel='engine_capacity', ylabel='resale_price'>



From the above we could see that the resale price increases when the engine capacity more.

[1326]:
```
sns.regplot(x="kms_driven", y="resale_price", data=df)
```

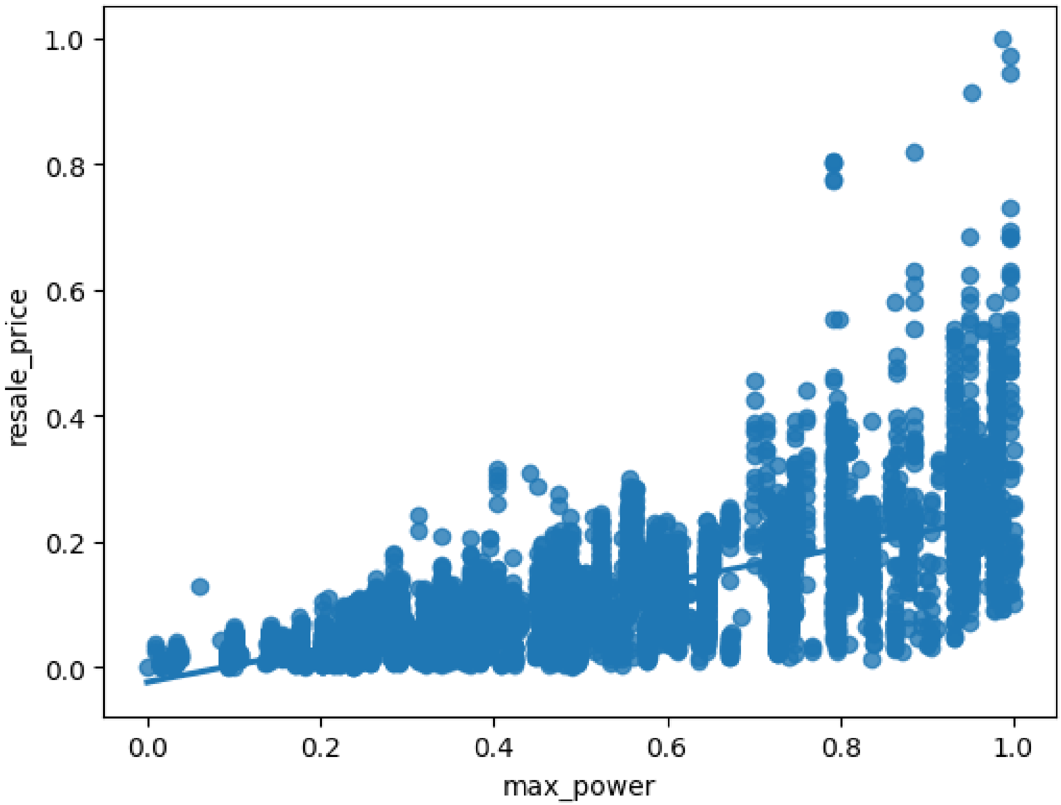
[1326]: <Axes: xlabel='kms_driven', ylabel='resale_price'>



From the above we could see that the resale price decreases when the kms driven is more.

[1327]:
```
sns.regplot(x="max_power", y="resale_price", data=df)
```

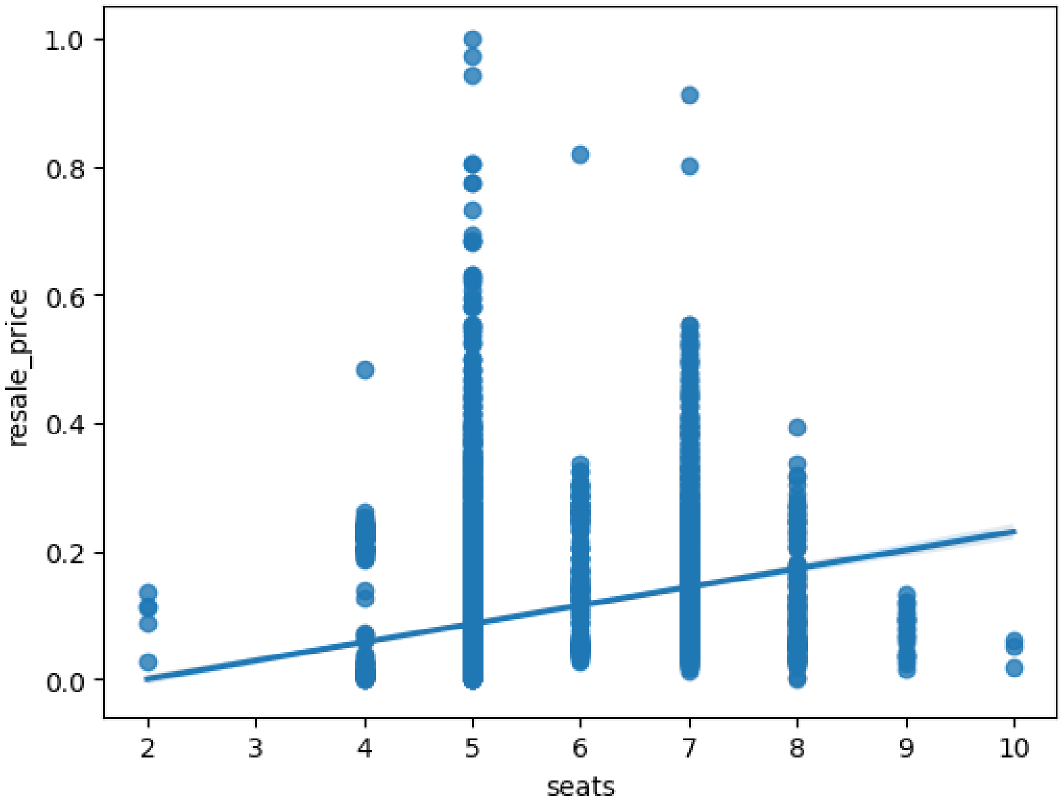
[1327]: <Axes: xlabel='max_power', ylabel='resale_price'>



From the above we could see that the resale price increases when the max power increases.

[1328]:
```
sns.regplot(x="seats", y="resale_price", data=df)
```

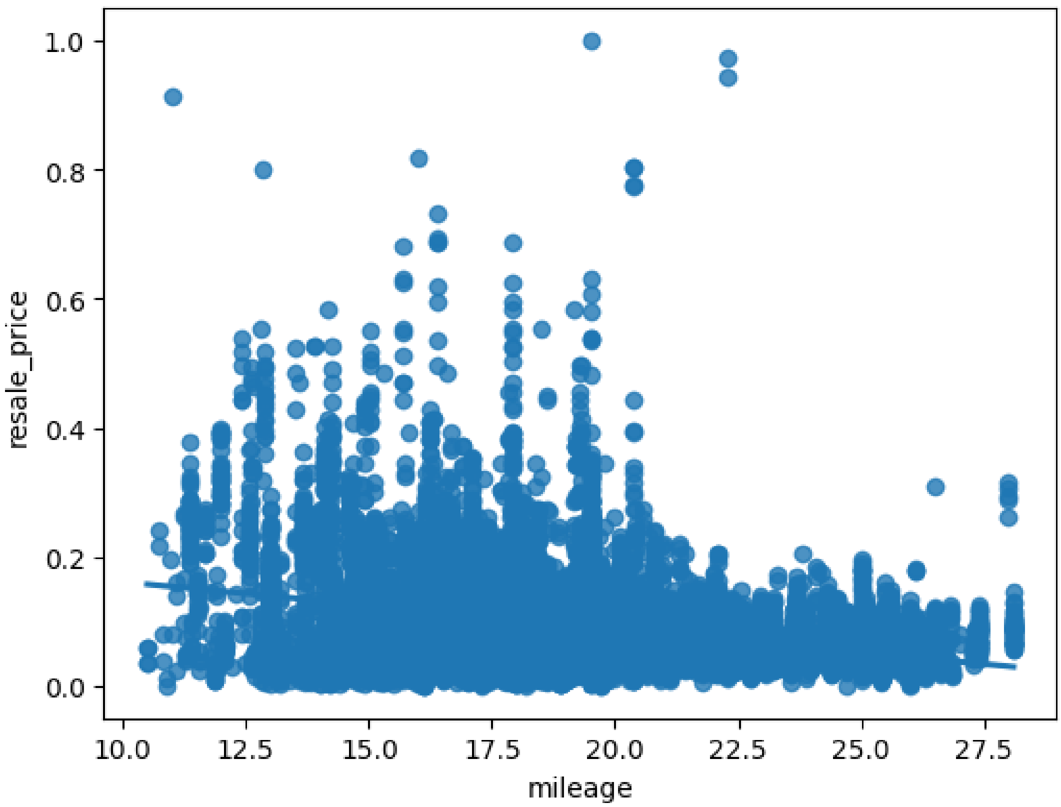
[1328]:
```
<Axes: xlabel='seats', ylabel='resale_price'>
```



From the above we could see that the resale price increases when the number of seats increases and number of resale cars with 5 seats is more

[1329]:
```
sns.regplot(x="mileage", y="resale_price", data=df)
```

[1329]: <Axes: xlabel='mileage', ylabel='resale_price'>

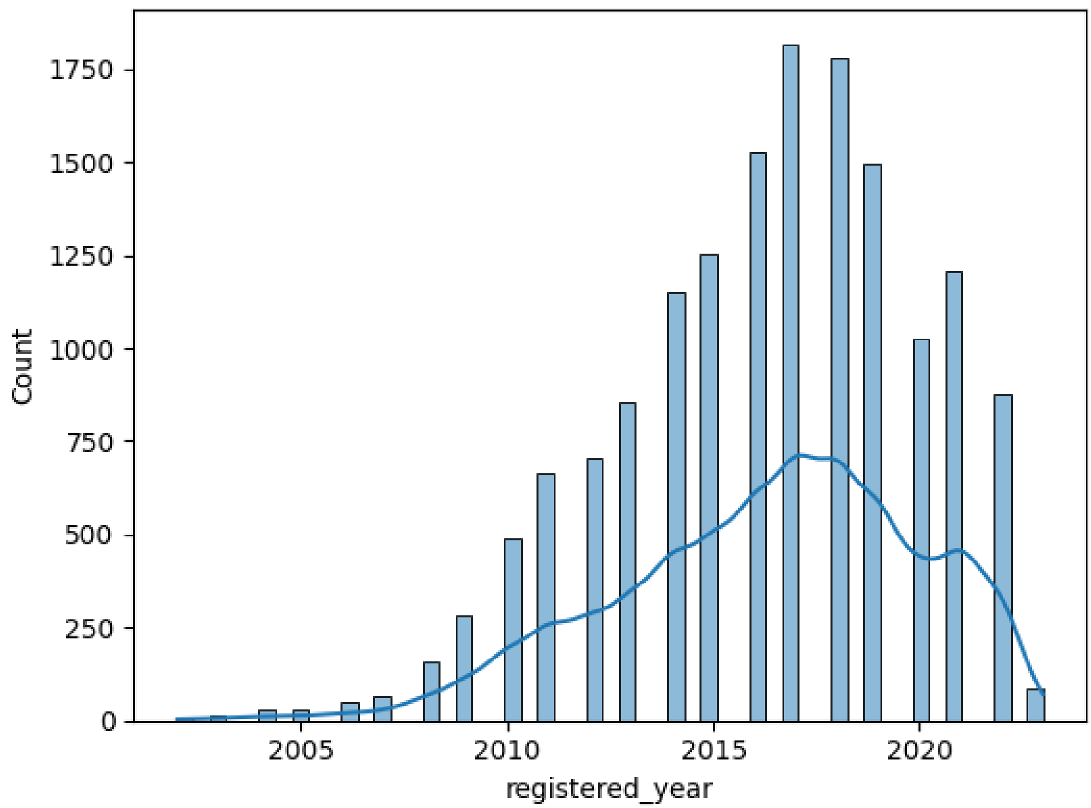


From the above we could see that the resale price decreases when the mileage increases and the number of resale cars within 12 to 20 mileage is high

Lets analyse the numerical attributes in depth

[1330]:
```
sns.histplot(df['registered_year'], kde=True)
```

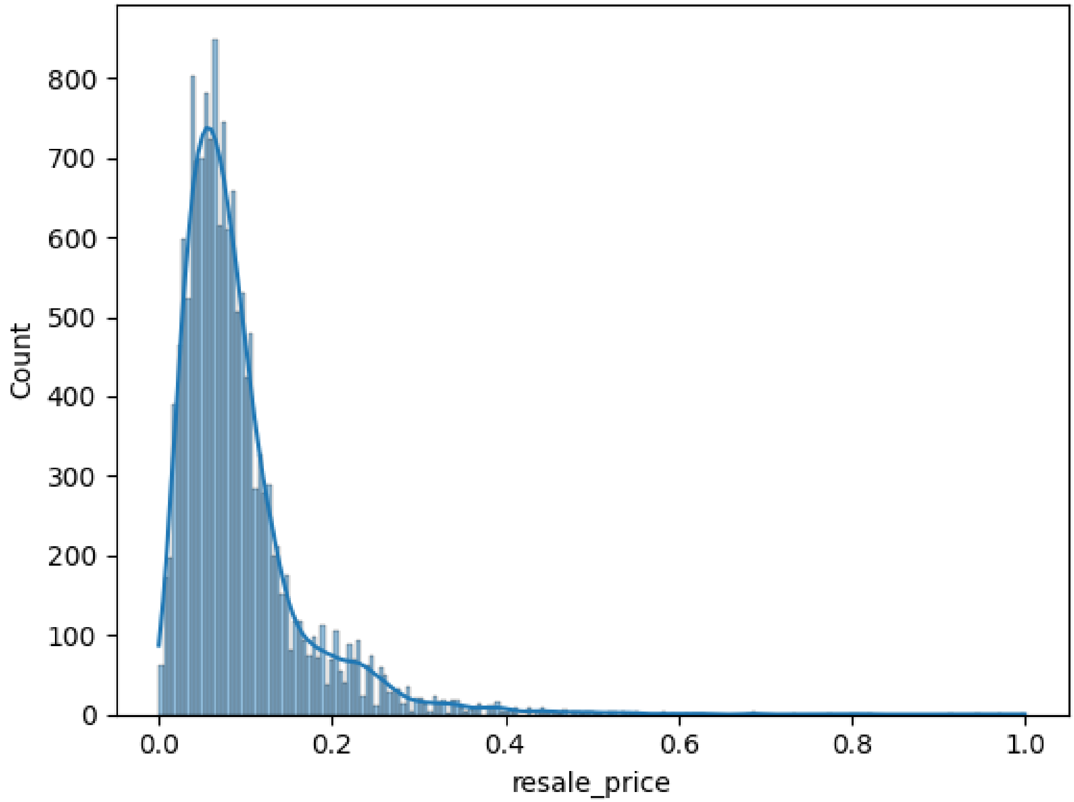
[1330]: <Axes: xlabel='registered_year', ylabel='Count'>



From above plot we can infer that high number of resale cars was registered between 2015 to 2020

[1331]:
```
sns.histplot(df['resale_price'], kde=True)
```

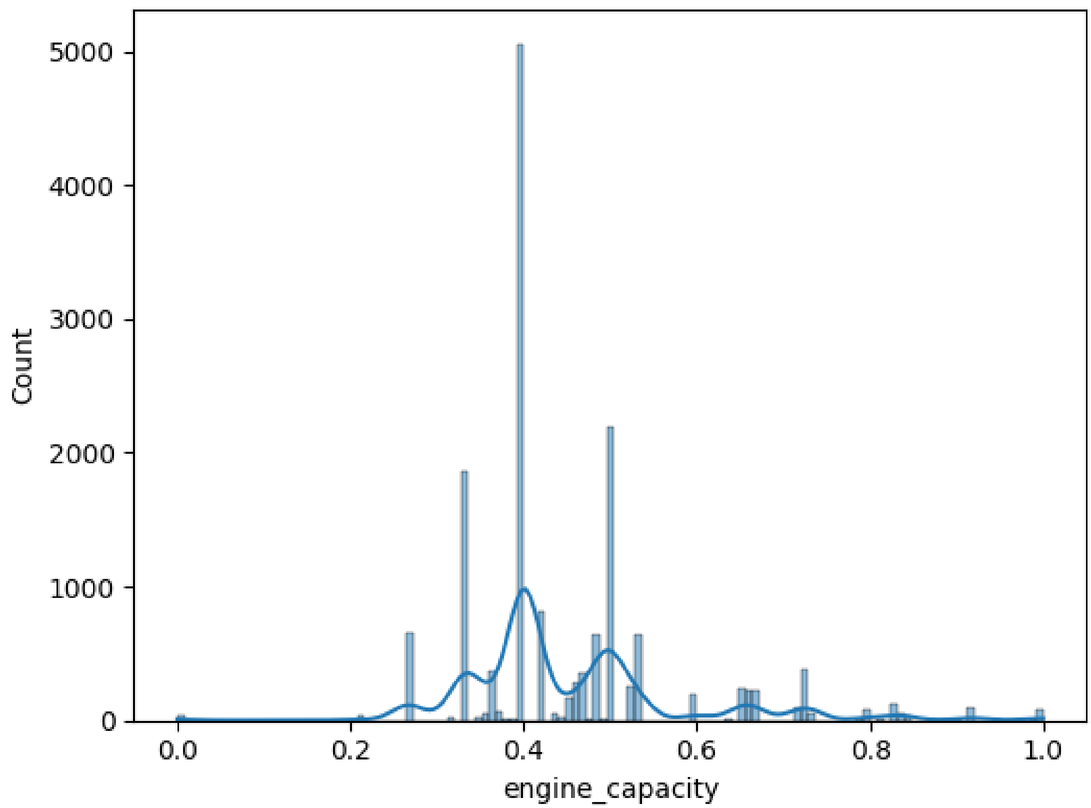
[1331]: <Axes: xlabel='resale_price', ylabel='Count'>



The resale price of all cars in the dataset falls in the range of 0.0 to 0.4 normalized values

[1332]:
```
sns.histplot(df['engine_capacity'], kde=True)
```

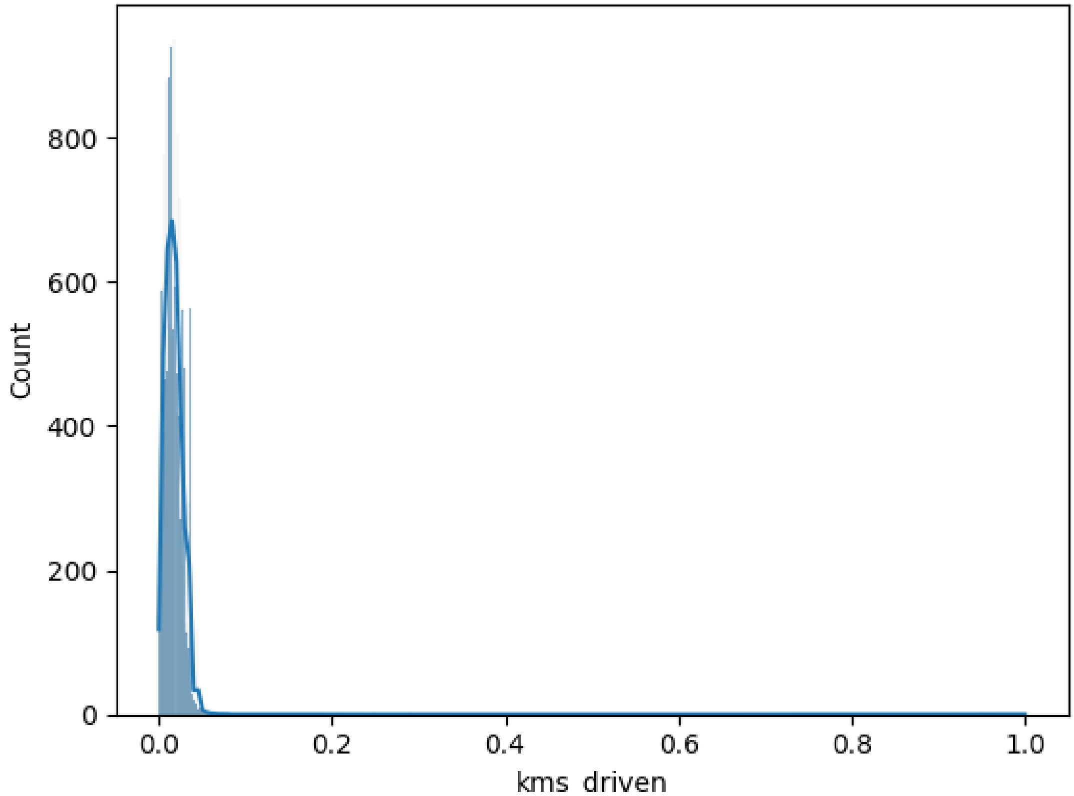
[1332]: <Axes: xlabel='engine_capacity', ylabel='Count'>



From above plot we can infer that high number of resale cars has engine capacity between 0.3 to 0.6 normalized values

[1333]:
```
sns.histplot(df['kms_driven'], kde=True)
```

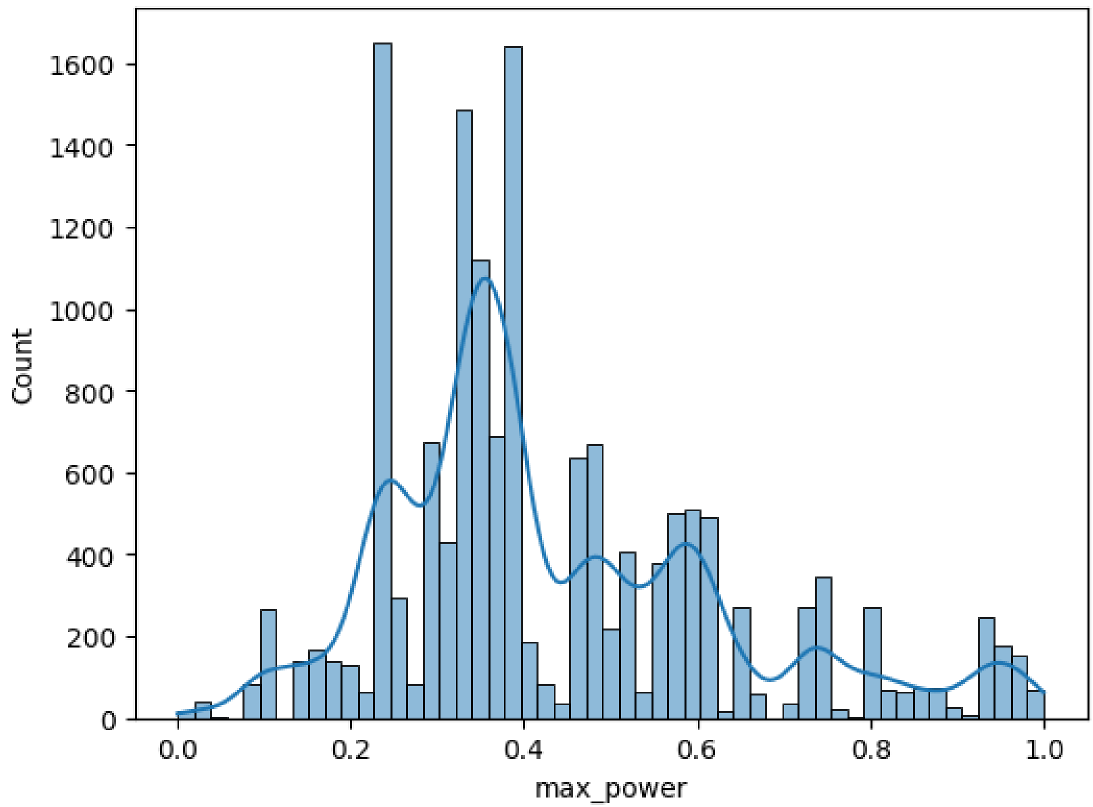
[1333]: <Axes: xlabel='kms_driven', ylabel='Count'>



From above plot we can infer that high number of resale cars has kms driven between 0.0 to 0.1 normalized values

[1334]:
```
sns.histplot(df['max_power'], kde=True)
```

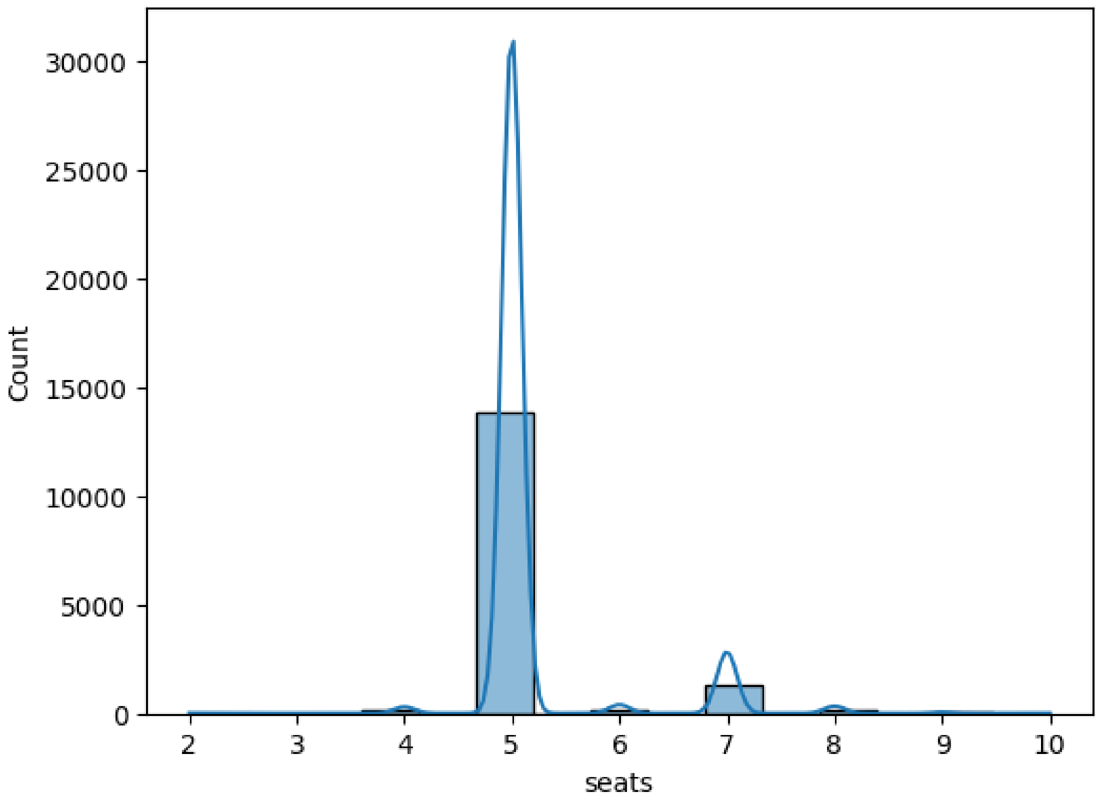
[1334]: <Axes: xlabel='max_power', ylabel='Count'>



From above plot we can infer that high number of resale cars has max power between 0.2 to 0.6 normalized values

[1335]:
```
sns.histplot(df['seats'], kde=True)
```

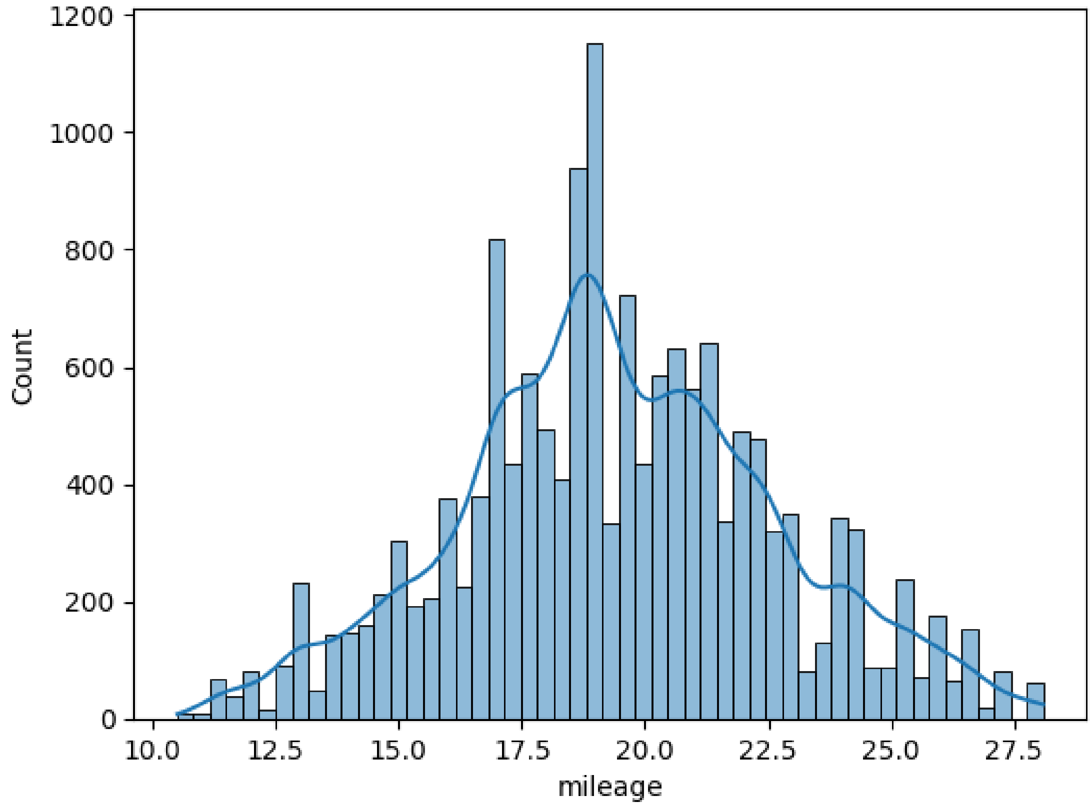
[1335]: <Axes: xlabel='seats', ylabel='Count'>



From above plot we can infer that high number of resale cars has 5 seats

[1336]:
```
sns.histplot(df['mileage'], kde=True)
```

[1336]: <Axes: xlabel='mileage', ylabel='Count'>



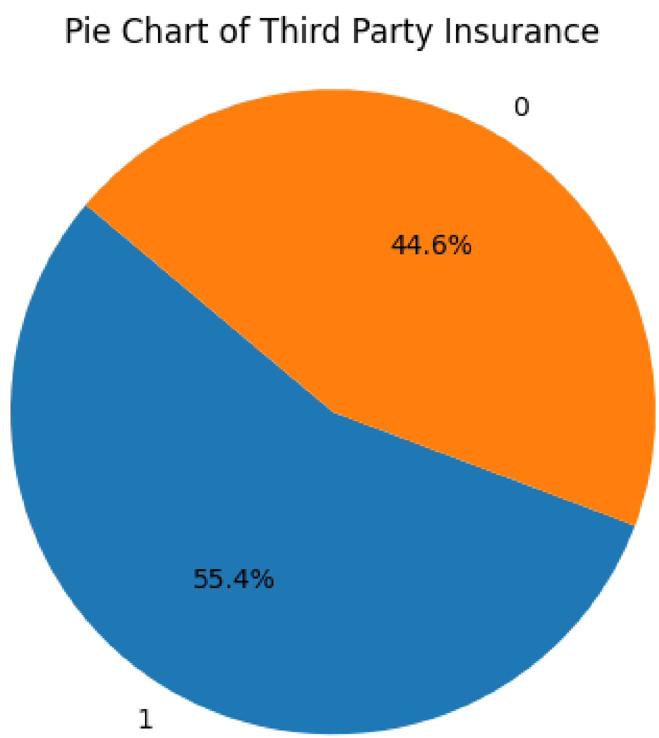
From above plot we can infer that high number of resale cars has mileage between 25 to 22

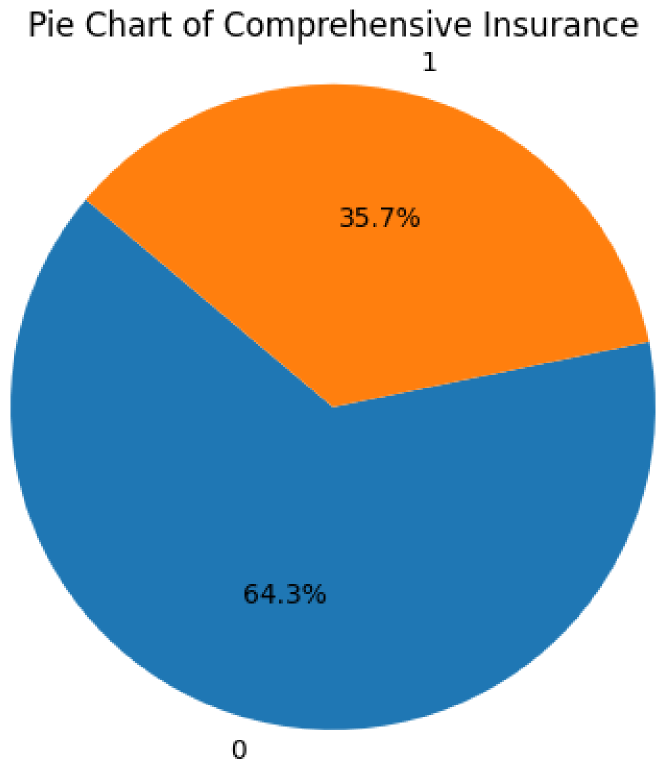
Lets analyse the insurance categorical encoded features

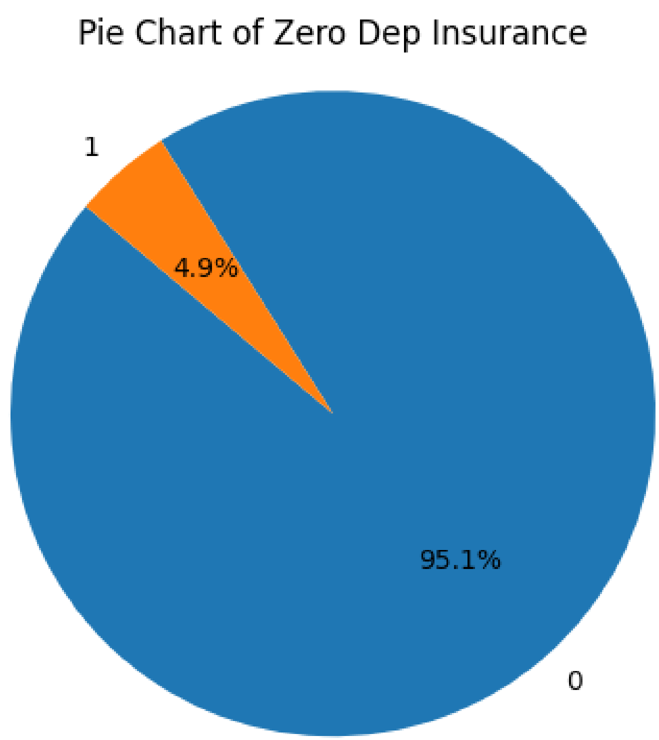
Different types of insurance in the dataset are 1. Third Party insurance - this is encoded in insurance_1 attribute 2. Comprehensive - this is encoded in insurance_2 attribute
3. Zero Dep - this is encoded in insurance_3 attribute
4. Not Available - this is encoded in insurance_4 attribute

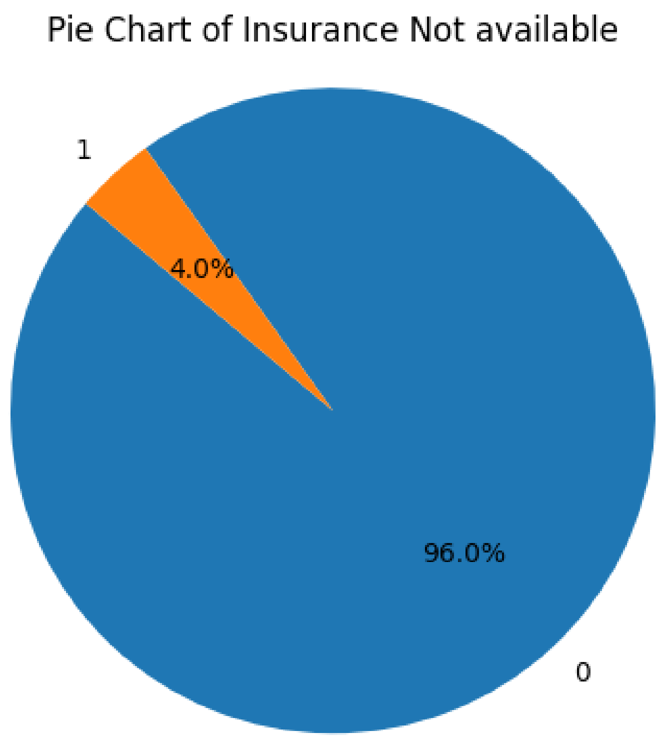
[1337]:
```
category_counts = df['insurance_1'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
 ↪startangle=140)
plt.title('Pie Chart of Third Party Insurance')
plt.axis('equal')
plt.show()

category_counts = df['insurance_2'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
 ↪startangle=140)
plt.title('Pie Chart of Comprehensive Insurance')
plt.axis('equal')
```

```
plt.show()

category_counts = df['insurance_3'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
 ↪startangle=140)
plt.title('Pie Chart of Zero Dep Insurance')
plt.axis('equal')
plt.show()

category_counts = df['insurance_4'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
 ↪startangle=140)
plt.title('Pie Chart of Insurance Not available')
plt.axis('equal')
plt.show()
```




Pie Chart of Comprehensive Insurance
1
35.7%
64.3%
0


Pie Chart of Zero Dep Insurance
1
4.9%
95.1%
0

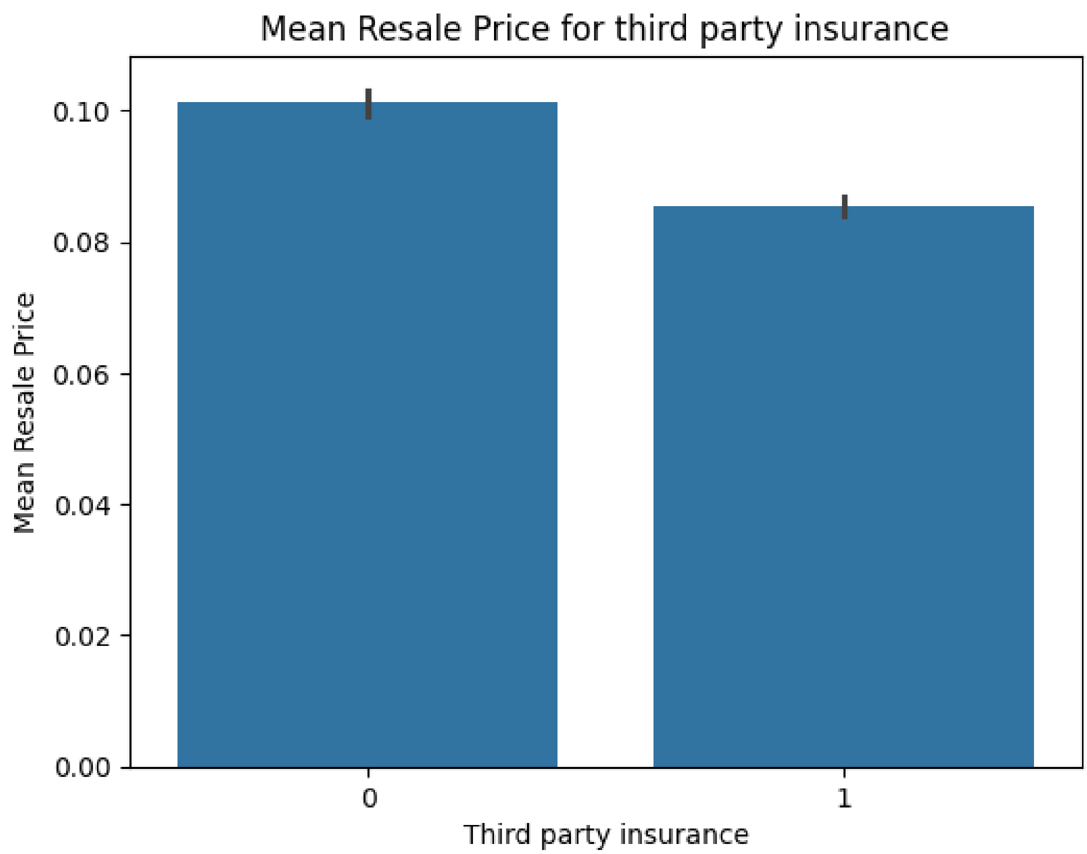


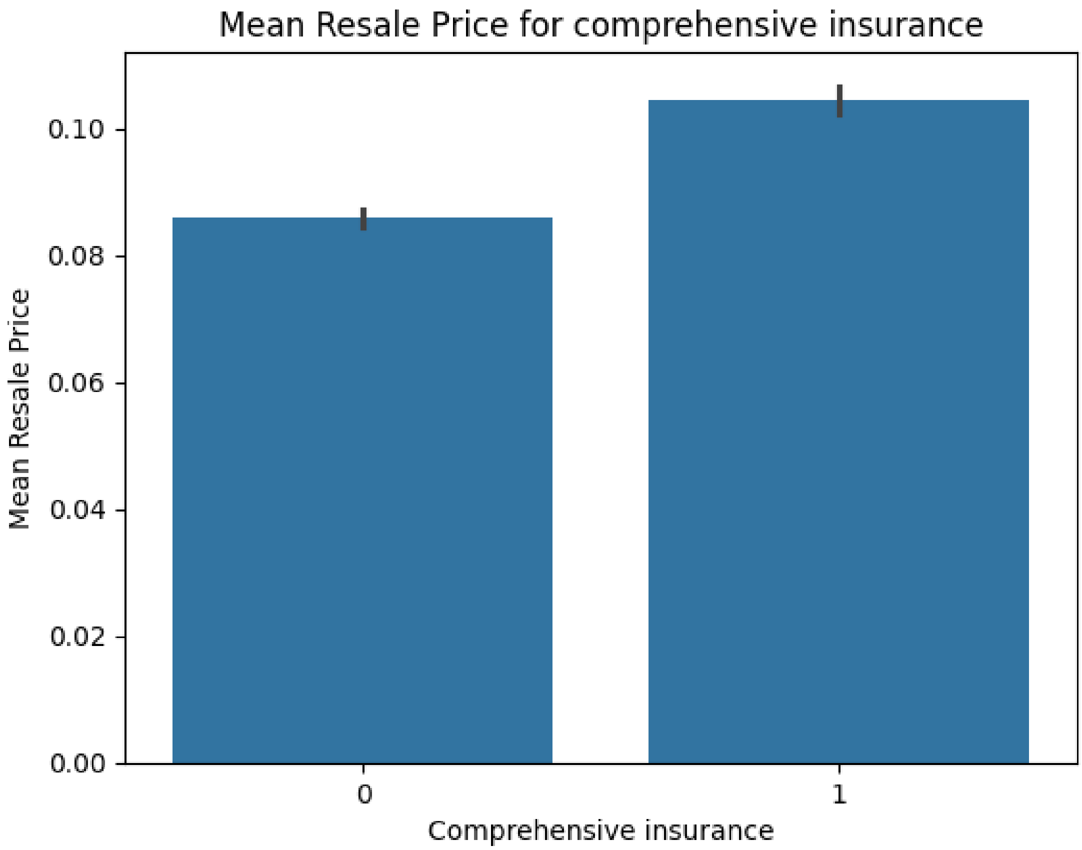
From the above we can infer that the no of resale cars with third party insurance is high, next highest is Comprehensive insurance, next highest is Zero dep insurance and then comes the cars with no insurance

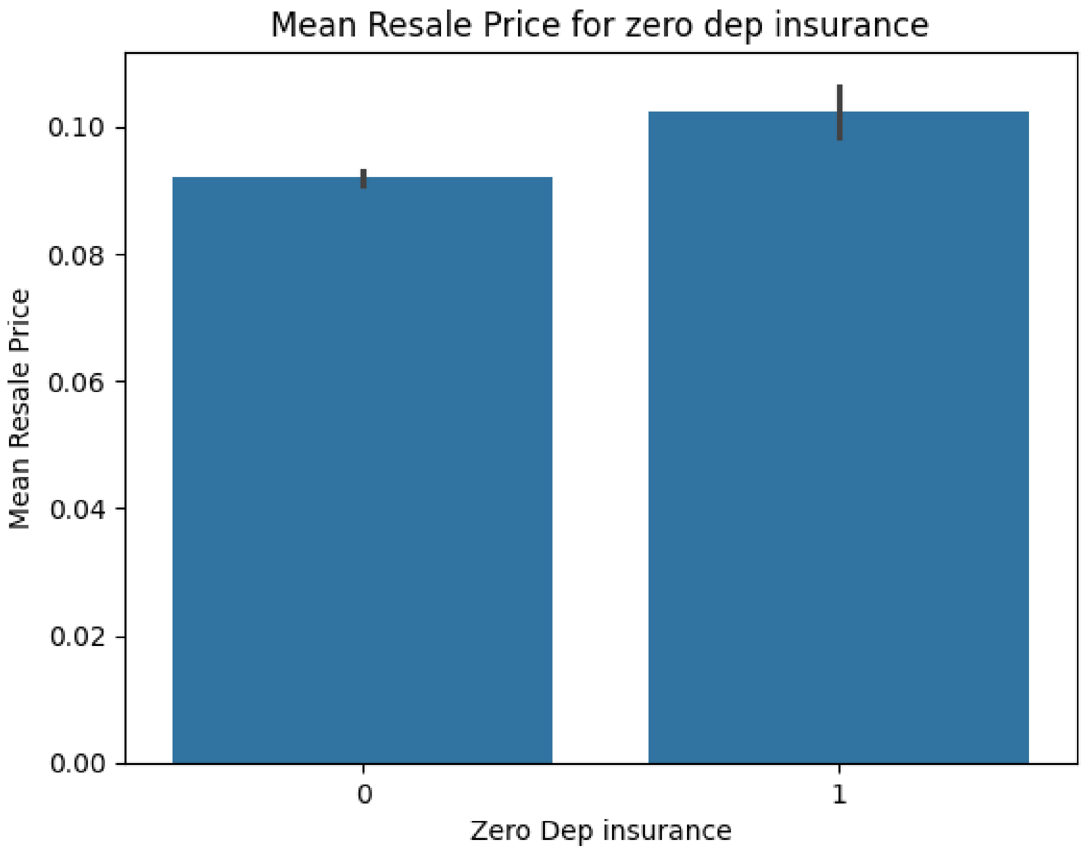
Now lets use barplot to idenify which insurance type car costs more by taking an average of the categories

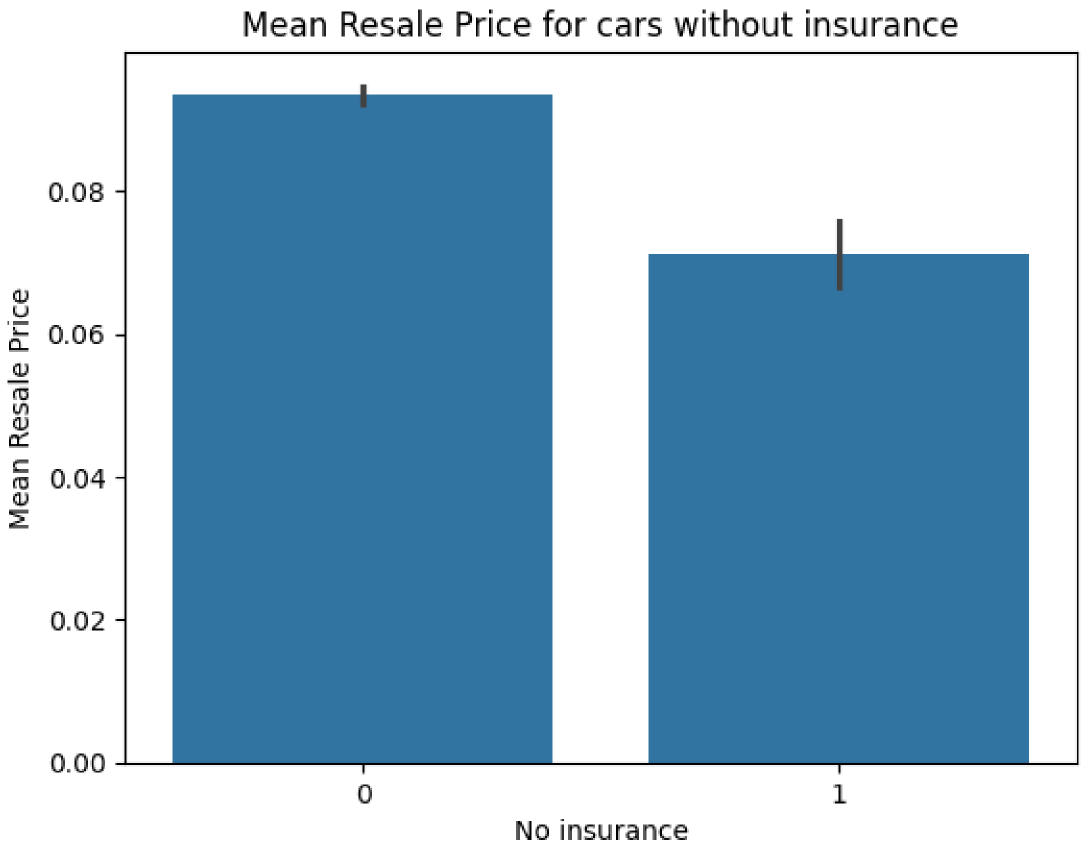
[1338]:
```
sns.barplot(x='insurance_1', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for third party insurance')
plt.xlabel('Third party insurance')
plt.ylabel('Mean Resale Price')
plt.show()

sns.barplot(x='insurance_2', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for comprehensive insurance')
plt.xlabel('Comprehensive insurance')
plt.ylabel('Mean Resale Price')
plt.show()
```

```
sns.barplot(x='insurance_3', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for zero dep insurance')
plt.xlabel('Zero Dep insurance')
plt.ylabel('Mean Resale Price')
plt.show()

sns.barplot(x='insurance_4', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for cars without insurance')
plt.xlabel('No insurance')
plt.ylabel('Mean Resale Price')
plt.show()
```




Mean Resale Price for comprehensive insurance
Mean Resale Price
0.00
0.02
0.04
0.06
0.08
0.10
0
1
Comprehensive insurance


Mean Resale Price for zero dep insurance
Mean Resale Price
0.00
0.02
0.04
0.06
0.08
0.10
0
1
Zero Dep insurance



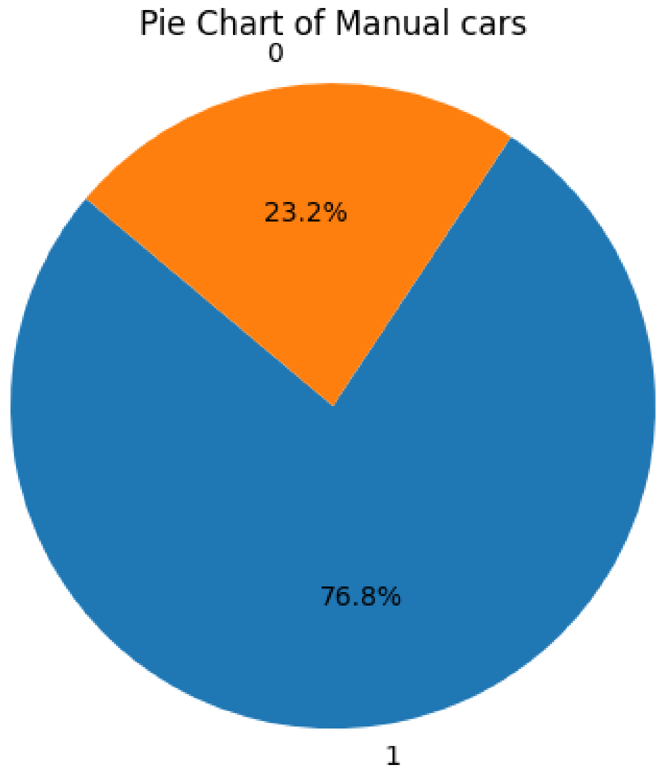
Number of resale cars with comprehensive and zero dep insurance has higher resale price, next highest is the ones with third party insurance and lastly is the ones with no insurance

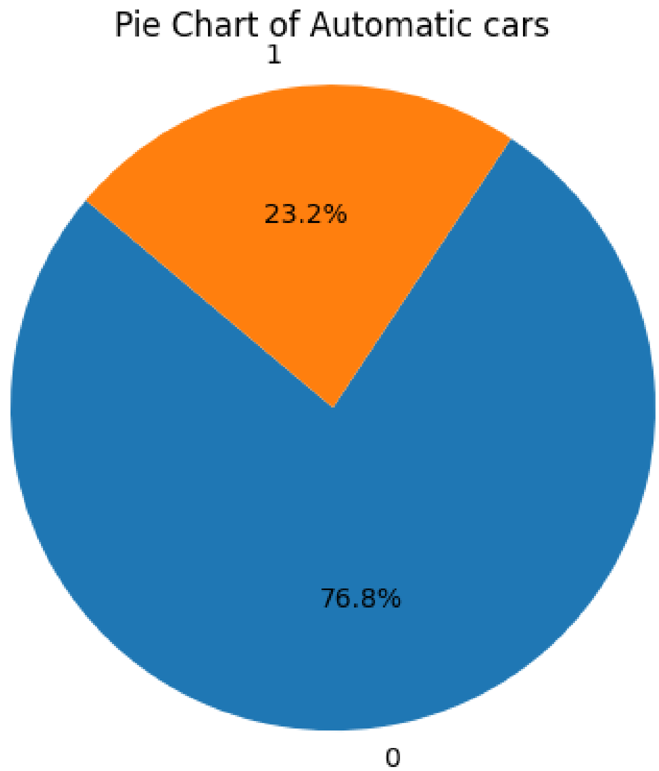
Now lets do the same for transmission type

Different types of transmission type in the dataset are 1. Manual - this is encoded in transmission_type_1 attribute 2. Automatic - this is encoded in transmission_type_2 attribute

[1339]:
```
category_counts = df['transmission_type_1'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
 ↪startangle=140)
plt.title('Pie Chart of Manual cars')
plt.axis('equal')
plt.show()

category_counts = df['transmission_type_2'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
 ↪startangle=140)
plt.title('Pie Chart of Automatic cars')
plt.axis('equal')
```

```
plt.show()
```

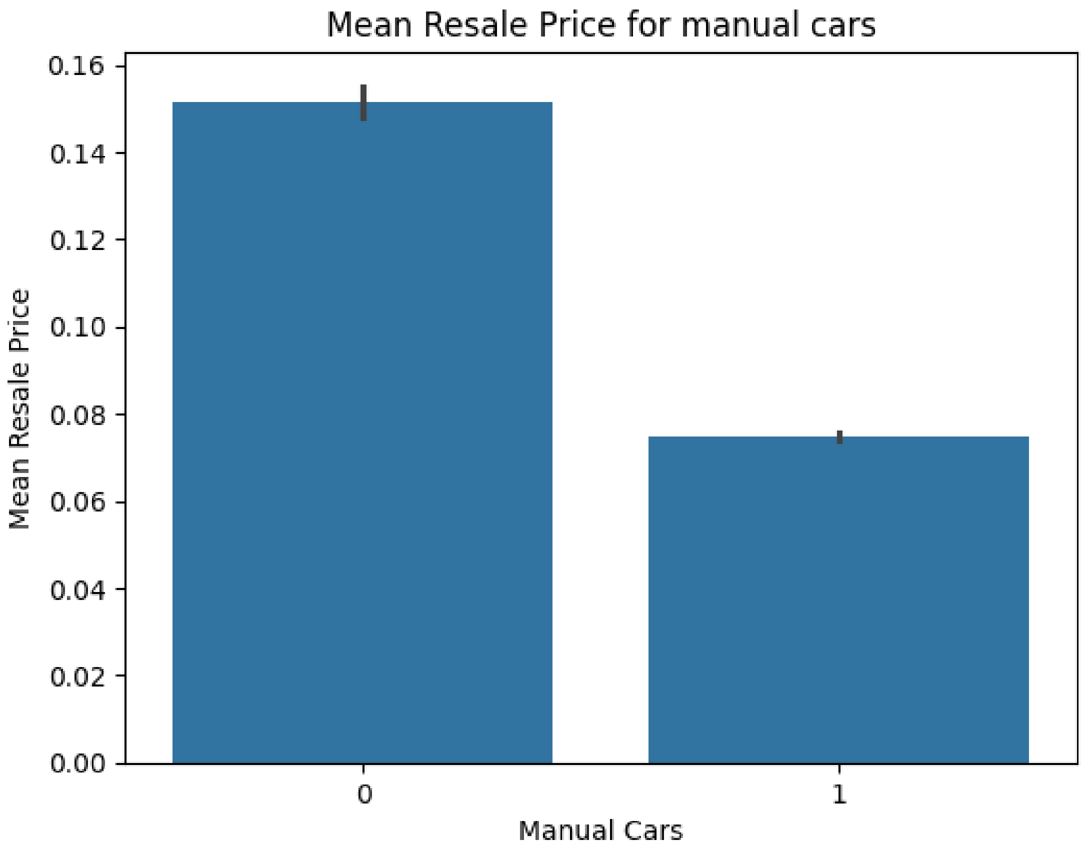


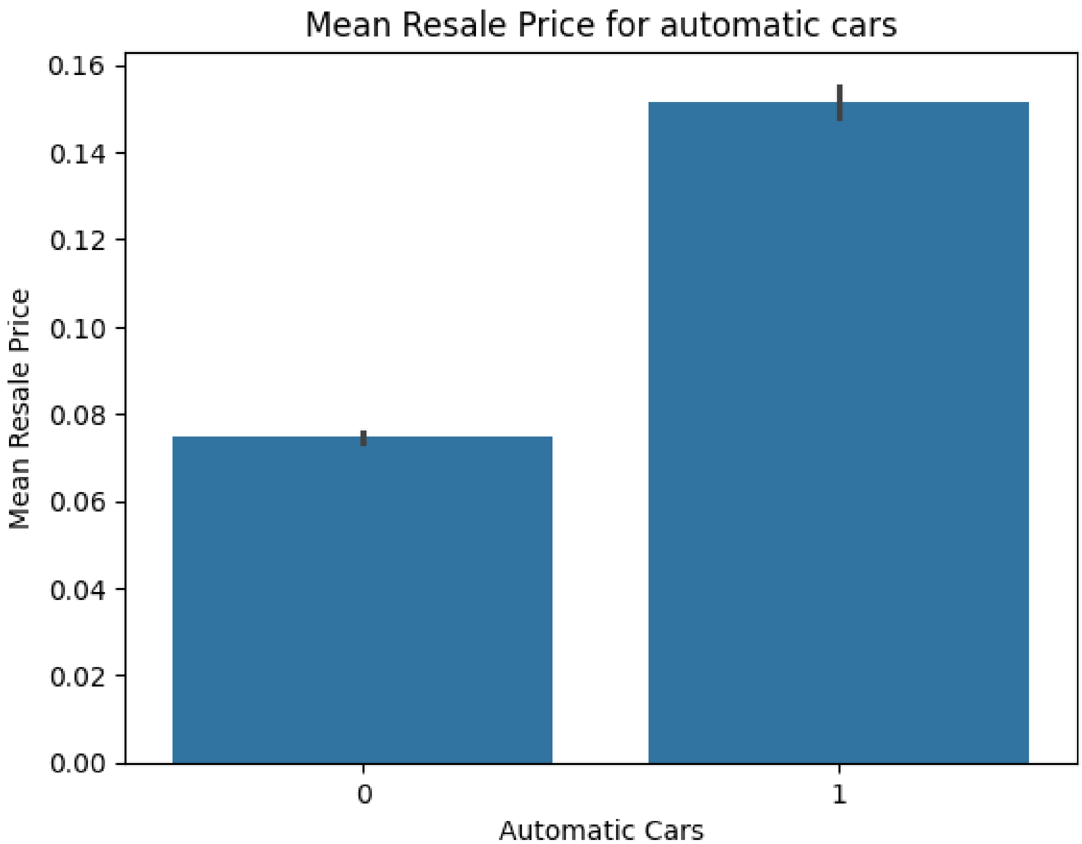


From the above charts we can infer that the number of manual cars is more than number of automatic cars

Now lets use barplot to idenify which transmission type car costs more by taking an average of the categories

[1340]:
```
sns.barplot(x='transmission_type_1', y='resale_price', data=df, estimator=np.
  ↪mean)  # Use np.mean for continuous target
plt.title('Mean Resale Price for manual cars')
plt.xlabel('Manual Cars')
plt.ylabel('Mean Resale Price')
plt.show()

sns.barplot(x='transmission_type_2', y='resale_price', data=df, estimator=np.
  ↪mean)  # Use np.mean for continuous target
plt.title('Mean Resale Price for automatic cars')
plt.xlabel('Automatic Cars')
plt.ylabel('Mean Resale Price')
plt.show()
```


Mean Resale Price for manual cars
Mean Resale Price
0.16
0.14
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1
Manual Cars



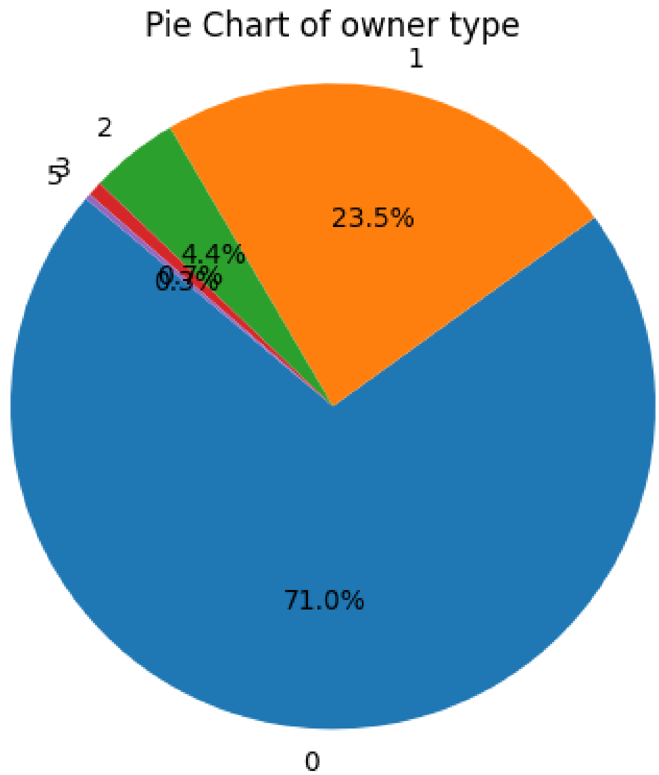
Automatic cars has high resale value when compared to manual cars

Now lets do the same for owner type

Different types of owner type in the dataset are

1. First owner - this is encoded as 0
2. Second owner - this is encoded as 1
3. Third owner - this is encoded as 2
4. Fourth owner - this is encoded as 3
5. Fifth owner - this is encoded as 4

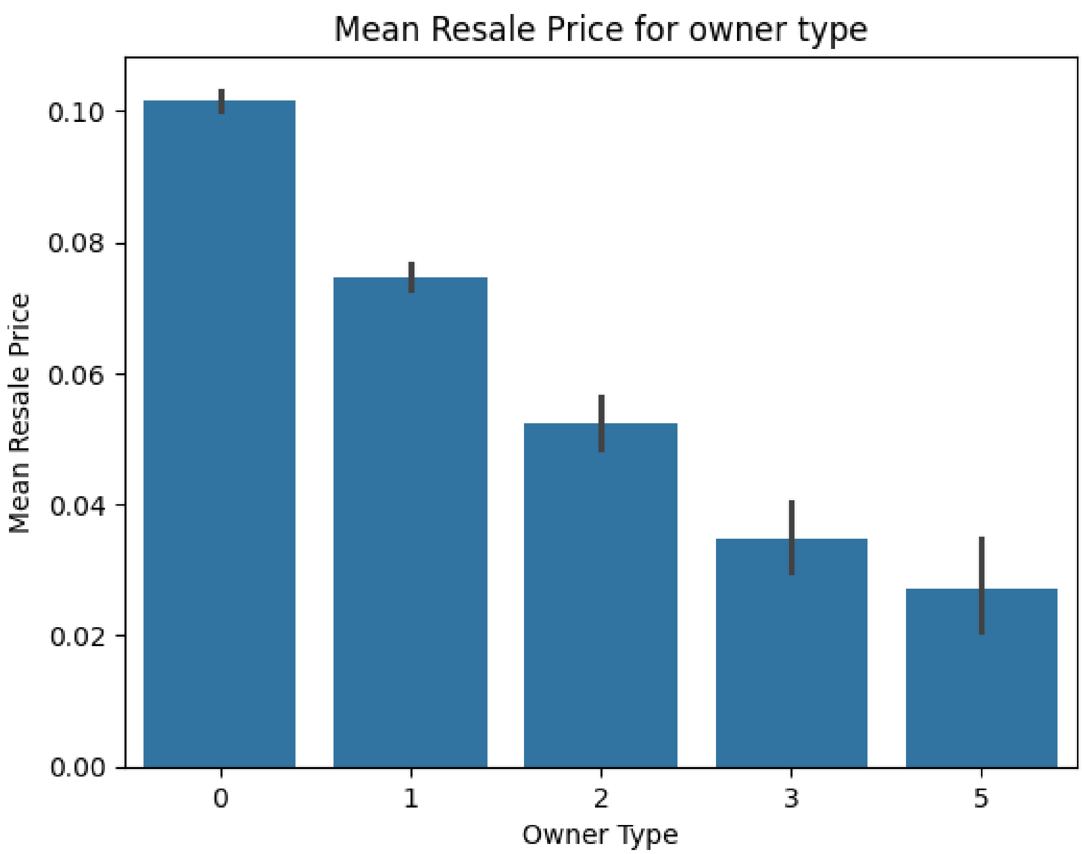
[1341]:
```
category_counts = df['owner_type'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
↪startangle=140)
plt.title('Pie Chart of owner type')
plt.axis('equal')
plt.show()
```



The number of first owner cars is highest in the dataset, next highest is the second owner cars then comes third owner cars then comes Fourth owner cars and then comes the Fifth owner cars

Now lets use barplot to idenify which owner type car costs more by taking an average of the categories

[1342]:
```
sns.barplot(x='owner_type', y='resale_price', data=df, estimator=np.mean)  #␣
↪Use np.mean for continuous target
plt.title('Mean Resale Price for owner type')
plt.xlabel('Owner Type')
plt.ylabel('Mean Resale Price')
plt.show()
```



First owner cars has high resale price whereas the 5 th owner cars has low resale price

Now lets do the same for fuel type

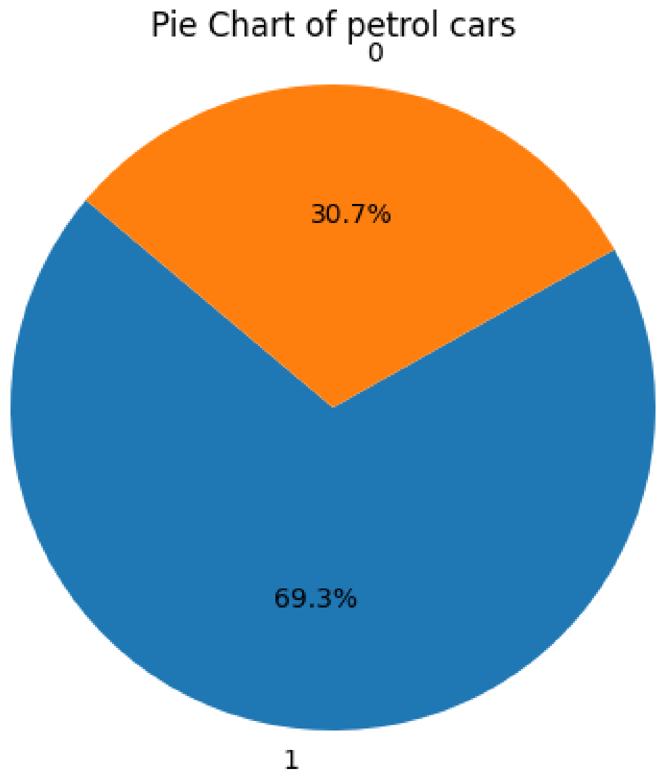
Different types of fuel type in the dataset are

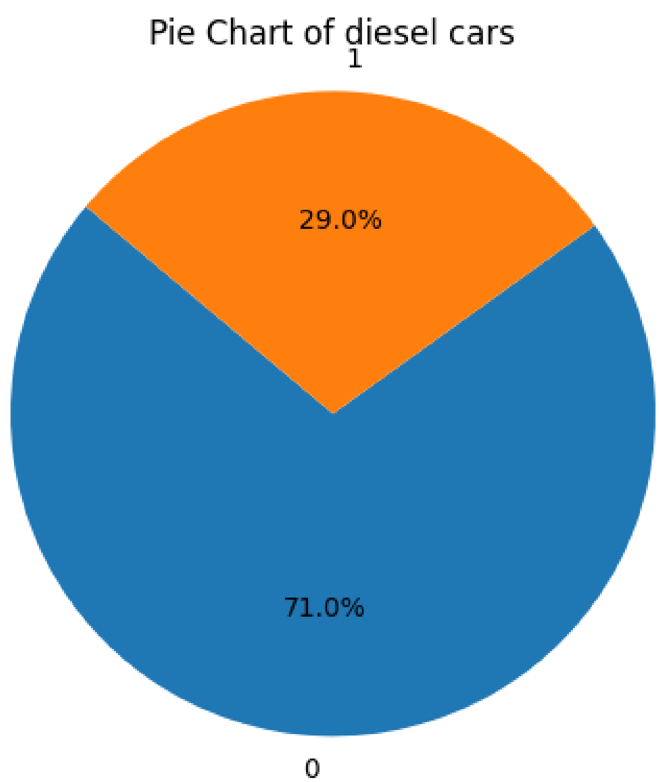
1. Petrol - this is encoded in fuel_type_1 attribute

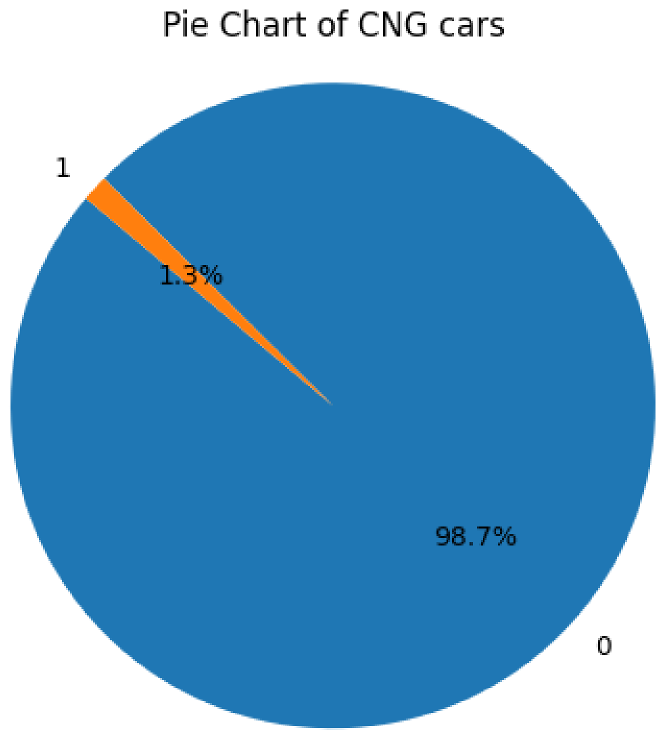
2. Diesel - this is encoded in fuel_type_2 attribute
3. CNG - this is encoded in fuel_type_3 attribute
4. Electric - this is encoded in fuel_type_4 attribute
5. LPG - this is encoded in fuel_type_5 attribute

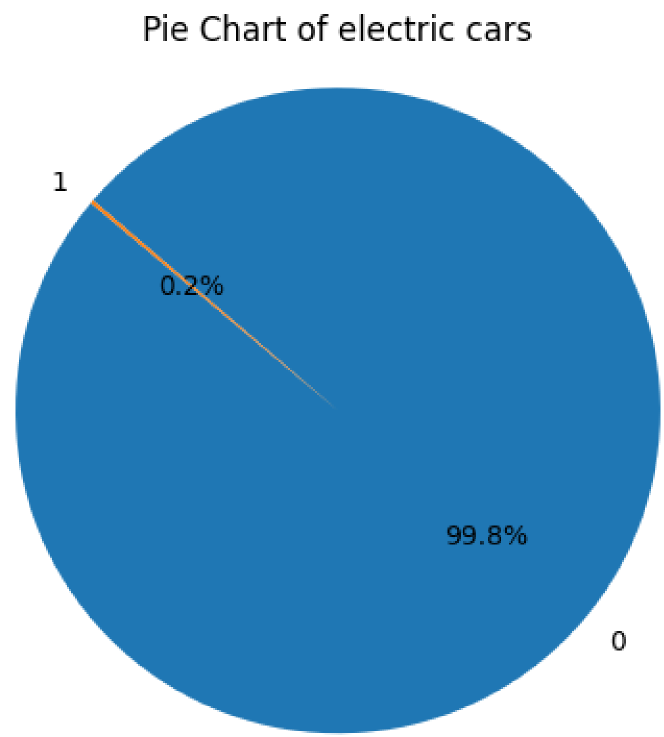
[1343]:
```
category_counts = df['fuel_type_1'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
↪startangle=140)
plt.title('Pie Chart of petrol cars')
plt.axis('equal')
plt.show()

category_counts = df['fuel_type_2'].value_counts()
```

```
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
↪startangle=140)
plt.title('Pie Chart of diesel cars')
plt.axis('equal')
plt.show()

category_counts = df['fuel_type_3'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
↪startangle=140)
plt.title('Pie Chart of CNG cars')
plt.axis('equal')
plt.show()

category_counts = df['fuel_type_4'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
↪startangle=140)
plt.title('Pie Chart of electric cars')
plt.axis('equal')
plt.show()

category_counts = df['fuel_type_5'].value_counts()
plt.pie(category_counts, labels=category_counts.index, autopct='%1.1f%%',␣
↪startangle=140)
plt.title('Pie Chart of LPG cars')
plt.axis('equal')
plt.show()
```

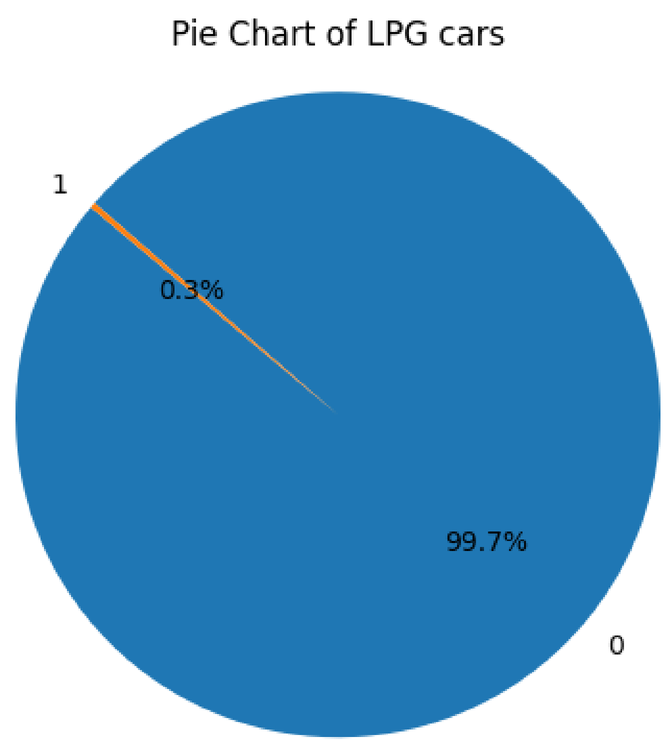

Pie Chart of petrol cars
0
30.7%
69.3%
1


Pie Chart of diesel cars
1
29.0%
71.0%
0


Pie Chart of CNG cars
1
1.3%
98.7%
0



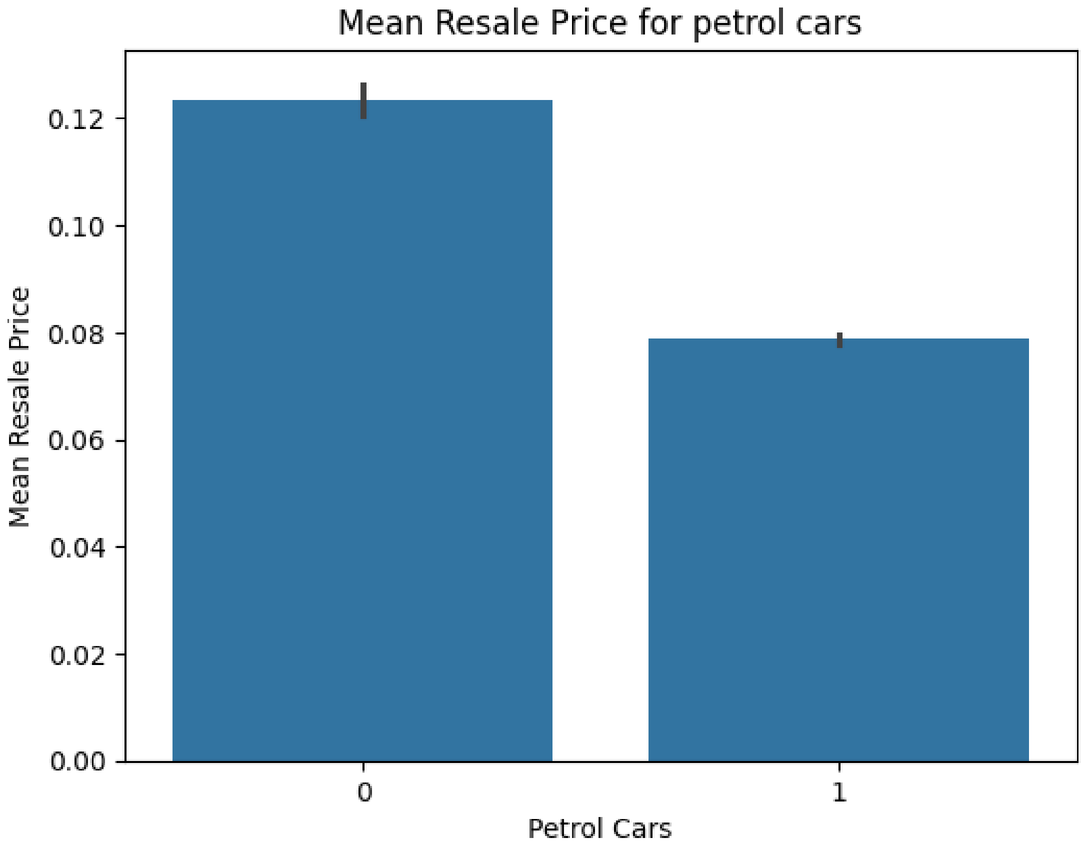
Pie Chart of electric cars
1
0.2%
99.8%
0

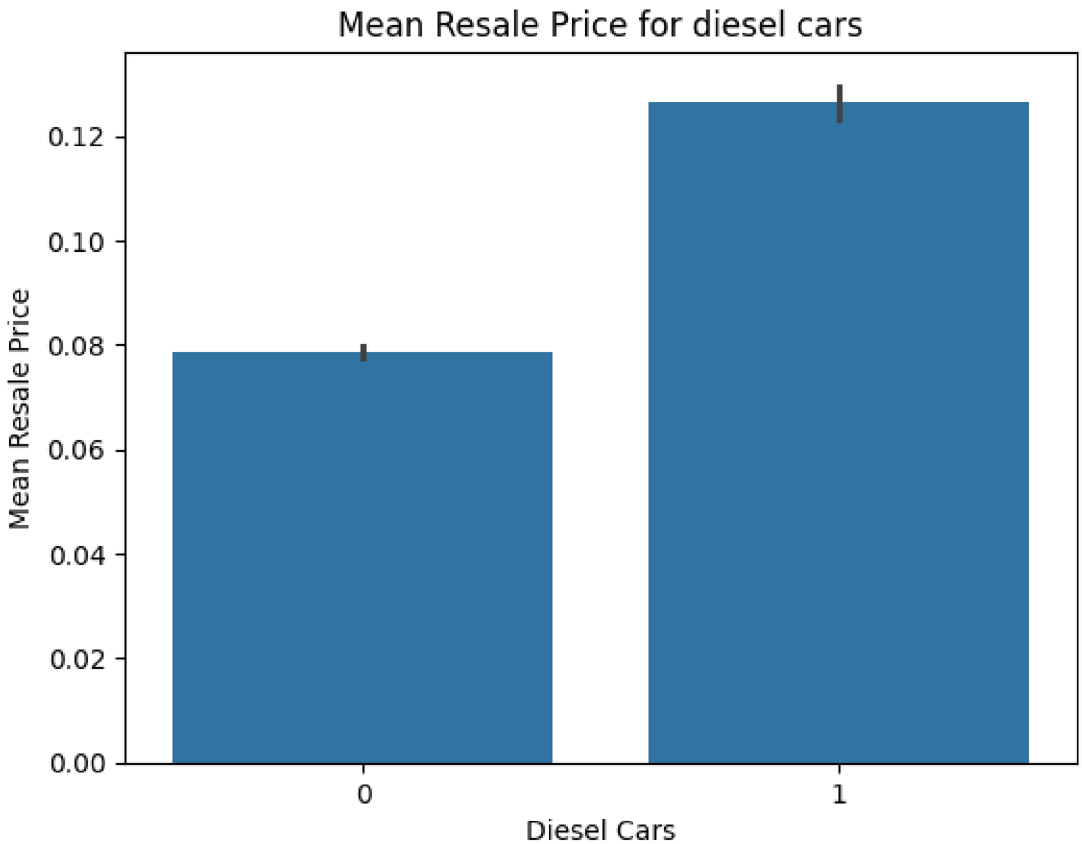


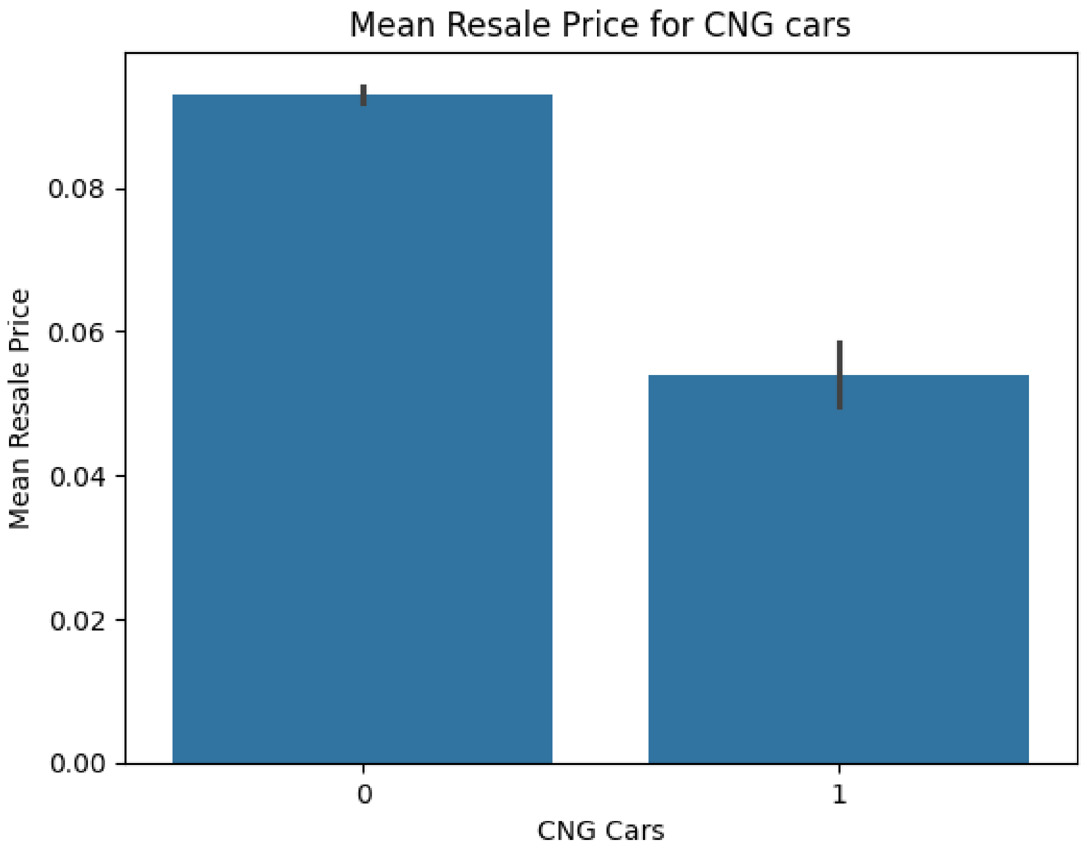
Pie Chart of LPG cars
1
0.3%
99.7%
0

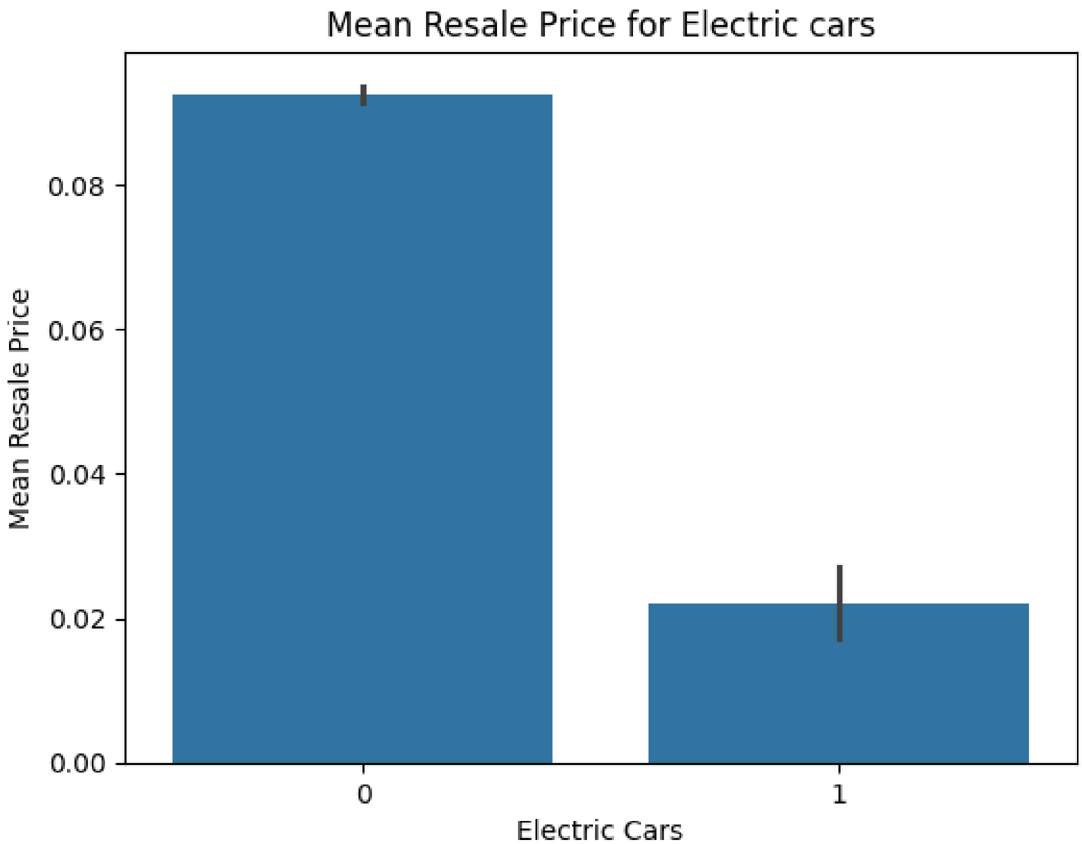
From the above charts we can infer that the number of petrol cars is the highest, then comes the diesel cars, then comes CNG cars, then comes electric cars and then comes LPG cars

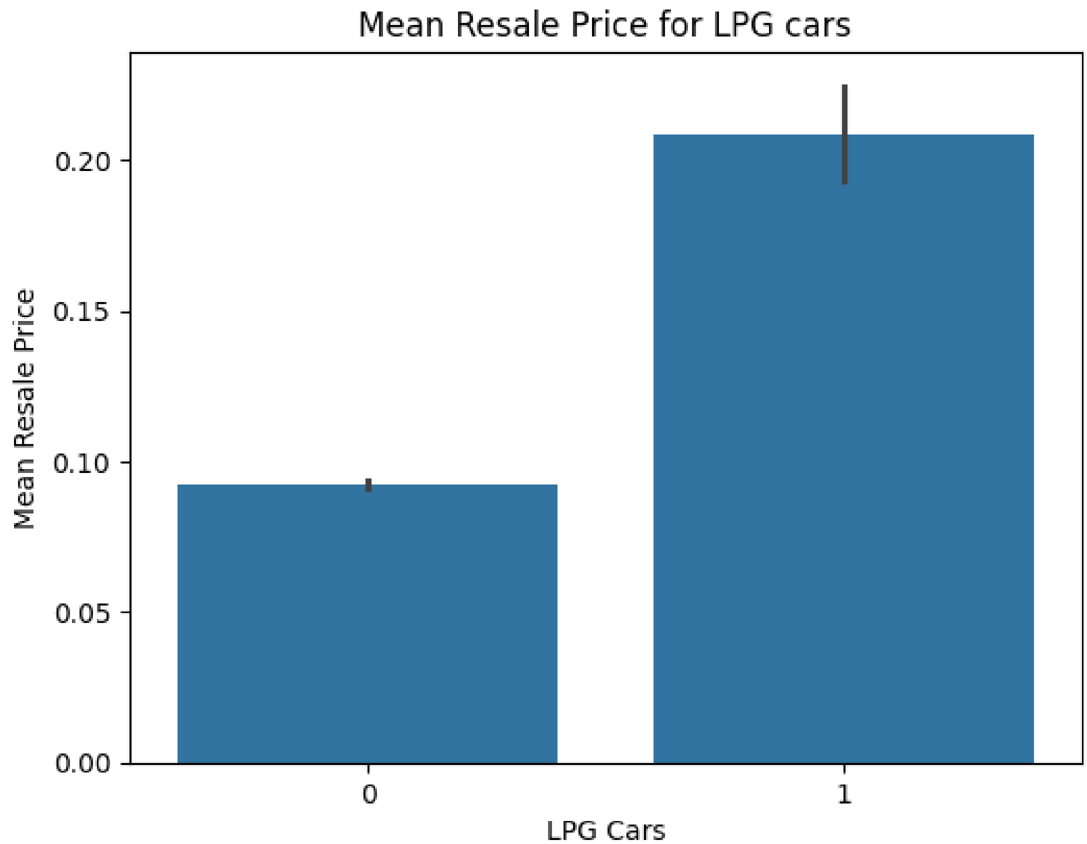
[1344]:
```
sns.barplot(x='fuel_type_1', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for petrol cars')
plt.xlabel('Petrol Cars')
plt.ylabel('Mean Resale Price')
plt.show()

sns.barplot(x='fuel_type_2', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for diesel cars')
plt.xlabel('Diesel Cars')
plt.ylabel('Mean Resale Price')
plt.show()

sns.barplot(x='fuel_type_3', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for CNG cars')
plt.xlabel('CNG Cars')
plt.ylabel('Mean Resale Price')
plt.show()

sns.barplot(x='fuel_type_4', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for Electric cars')
plt.xlabel('Electric Cars')
plt.ylabel('Mean Resale Price')
plt.show()

sns.barplot(x='fuel_type_5', y='resale_price', data=df, estimator=np.mean)  #␣
 ↪Use np.mean for continuous target
plt.title('Mean Resale Price for LPG cars')
plt.xlabel('LPG Cars')
plt.ylabel('Mean Resale Price')
plt.show()
```


Mean Resale Price for petrol cars
Mean Resale Price
0.12
0.10
0.08
0.06
0.04
0.02
0.00
0
1
Petrol Cars


Mean Resale Price for diesel cars
Mean Resale Price
0.00
0.02
0.04
0.06
0.08
0.10
0.12
0
1
Diesel Cars


Mean Resale Price for CNG cars
Mean Resale Price
0.00
0.02
0.04
0.06
0.08
0
1
CNG Cars


Mean Resale Price for Electric cars
0.08
0.06
0.04
0.02
0.00
Mean Resale Price
0
1
Electric Cars



The resale price of LPG cars is high, then comes the diesel, petrol, CNG and electric.

0.3.26 Feature Selection(2M)

Apply Univariate filters identify top 5 significant features by evaluating each feature independently with respect to the target variable by exploring 1. Mutual Information (Information Gain) 2. Gini index 3. Gain Ratio 4. Chi-Squared test 5. Fisher Score (From the above 5 you are required to use any two)

Lets apply Chi-Squared test on all categorical attributes and resale_price_discretized

[1345]:
```
df.columns
```

[1345]:
```
Index(['full_name_0', 'full_name_1', 'full_name_2', 'full_name_3',
       'full_name_4', 'full_name_5', 'full_name_6', 'full_name_7',
       'full_name_8', 'full_name_9', 'full_name_10', 'full_name_11',
       'full_name_12', 'resale_price', 'registered_year', 'engine_capacity',
       'insurance_1', 'insurance_2', 'insurance_3', 'insurance_4',
       'transmission_type_1', 'transmission_type_2', 'kms_driven',
       'owner_type', 'fuel_type_1', 'fuel_type_2', 'fuel_type_3',
       'fuel_type_4', 'fuel_type_5', 'max_power', 'seats', 'mileage',
```

```
       'body_type_1', 'body_type_2', 'body_type_3', 'body_type_4',
       'body_type_5', 'body_type_6', 'body_type_7', 'city_1', 'city_2',
       'city_3', 'city_4', 'city_5', 'city_6', 'city_7', 'city_8', 'city_9',
       'city_10', 'city_11', 'city_12', 'city_13', 'registered_year_bin',
       'engine_capacity_bin', 'max_power_bin', 'mileage_bin',
       'kms_driven_discretized', 'seats_discretized',
       'resale_price_discretized'],
      dtype='object')
```

[1346]:
```python
from sklearn.feature_selection import chi2
categorical_cols = ['full_name_0', 'full_name_1', 'full_name_2', 'full_name_3',
       'full_name_4', 'full_name_5', 'full_name_6', 'full_name_7',
       'full_name_8', 'full_name_9', 'full_name_10', 'full_name_11',
       'full_name_12', 'insurance_1', 'insurance_2', 'insurance_3',␣
 ↪'insurance_4',
       'transmission_type_1', 'transmission_type_2',
       'owner_type', 'fuel_type_1', 'fuel_type_2', 'fuel_type_3',
       'fuel_type_4', 'fuel_type_5',
       'body_type_1', 'body_type_2', 'body_type_3', 'body_type_4',
       'body_type_5', 'body_type_6', 'body_type_7', 'city_1', 'city_2',
       'city_3', 'city_4', 'city_5', 'city_6', 'city_7', 'city_8', 'city_9',
       'city_10', 'city_11', 'city_12', 'city_13',
       'kms_driven_discretized', 'seats_discretized']

# Calculate chi-square scores and p-values
chi2_scores, p_values = chi2(df[categorical_cols],␣
 ↪df['resale_price_discretized'])

# Set NumPy print options to suppress scientific notation
pd.set_option('display.float_format', lambda x: '%.3f' % x)

# Display chi-square scores and p-values for each feature
chi2_results = pd.DataFrame({
    'Feature': categorical_cols,
    'Chi2 Score': chi2_scores,
    'P-value': p_values
})

print("Chi-square Test Results:")
print(chi2_results)
```

```
Chi-square Test Results:
                   Feature  Chi2 Score  P-value
0              full_name_0     143.895    0.000
1              full_name_1      95.845    0.000
2              full_name_2      13.640    0.001
3              full_name_3      22.057    0.000
4              full_name_4      13.575    0.001
```

5	full_name_5	11.616	0.003
6	full_name_6	5.525	0.063
7	full_name_7	3.567	0.168
8	full_name_8	13.711	0.001
9	full_name_9	5.240	0.073
10	full_name_10	0.974	0.615
11	full_name_11	2.830	0.243
12	full_name_12	1.476	0.478
13	insurance_1	105.746	0.000
14	insurance_2	128.528	0.000
15	insurance_3	190.092	0.000
16	insurance_4	112.204	0.000
17	transmission_type_1	512.000	0.000
18	transmission_type_2	1694.945	0.000
19	owner_type	1347.179	0.000
20	fuel_type_1	225.083	0.000
21	fuel_type_2	576.973	0.000
22	fuel_type_3	119.386	0.000
23	fuel_type_4	55.033	0.000
24	fuel_type_5	87.856	0.000
25	body_type_1	2205.963	0.000
26	body_type_2	286.260	0.000
27	body_type_3	24.643	0.000
28	body_type_4	21.471	0.000
29	body_type_5	3245.392	0.000
30	body_type_6	15.805	0.000
31	body_type_7	1.997	0.368
32	city_1	46.183	0.000
33	city_2	65.837	0.000
34	city_3	47.284	0.000
35	city_4	1.224	0.542
36	city_5	11.841	0.003
37	city_6	112.040	0.000
38	city_7	19.158	0.000
39	city_8	50.638	0.000
40	city_9	4.403	0.111
41	city_10	21.089	0.000
42	city_11	25.403	0.000
43	city_12	56.721	0.000
44	city_13	11.680	0.003
45	kms_driven_discretized	958.827	0.000
46	seats_discretized	0.513	0.774

Lets apply Mutual information on all categorical attributes and resale_price_discretized

[1347]:
```
from sklearn.feature_selection import mutual_info_classif
```

```python
mi_scores = mutual_info_classif(df[categorical_cols],
    df['resale_price_discretized'])

# Display mutual information or information gain scores for each feature
mi_scores_df = pd.DataFrame(data=mi_scores, index=categorical_cols,
    columns=['Mutual Information Score'])
print("Mutual Information Scores:")
print(mi_scores_df)
```

```
Mutual Information Scores:
                        Mutual Information Score
full_name_0                                0.006
full_name_1                                0.007
full_name_2                                0.003
full_name_3                                0.007
full_name_4                                0.011
full_name_5                                0.000
full_name_6                                0.002
full_name_7                                0.008
full_name_8                                0.000
full_name_9                                0.007
full_name_10                               0.005
full_name_11                               0.000
full_name_12                               0.003
insurance_1                                0.011
insurance_2                                0.000
insurance_3                                0.004
insurance_4                                0.002
transmission_type_1                        0.077
transmission_type_2                        0.070
owner_type                                 0.036
fuel_type_1                                0.026
fuel_type_2                                0.022
fuel_type_3                                0.001
fuel_type_4                                0.007
fuel_type_5                                0.002
body_type_1                                0.146
body_type_2                                0.011
body_type_3                                0.000
body_type_4                                0.000
body_type_5                                0.137
body_type_6                                0.000
body_type_7                                0.000
city_1                                     0.000
city_2                                     0.004
city_3                                     0.000
city_4                                     0.000
```

```
city_5                                     0.004
city_6                                     0.005
city_7                                     0.001
city_8                                     0.000
city_9                                     0.000
city_10                                    0.003
city_11                                    0.002
city_12                                    0.000
city_13                                    0.000
kms_driven_discretized                     0.050
seats_discretized                          0.002
```

0.3.27 Report observations (2M)

Write your observations from the results of each of the above method(1M). Clearly justify your choice of the method.(1M)

In Chi suqared test if the Chi square value is high and P value is low then it means that those features can be used to predict the resale price to a great extent. With this we can see that body_type, owner_type, transmission_type, kms_driven_discretized, fuel_type and insurance are the top features which can be used to predict the resale_price correctly.

The Chi-squared test is used for categorical attributes because it is a statistical method used to determine if there is a significant association between categorical variables.

In mutual information gain if the score is high then it means that those features can be used to predict the resale price to a great extent. With this we can see that body_type, transmission_type, kms_driven_discretized owner_type and fuel_type are the top features which can be used to predict the resale_price correctly.

The mutual information gain is used for categorical attributes because when dealing with datasets containing many features (high-dimensional data), MI gain can help identify the most informative features. Unlike correlation coefficients, MI gain can capture non-linear relationships between variables.

0.3.28 Correlation Analysis (3 M)

Perform correlation analysis(1M) and plot the visuals(1M).Briefly explain each process,why is it used and interpret the result(1M).

[1348]:
```
df.columns
```

[1348]:
```
Index(['full_name_0', 'full_name_1', 'full_name_2', 'full_name_3',
       'full_name_4', 'full_name_5', 'full_name_6', 'full_name_7',
       'full_name_8', 'full_name_9', 'full_name_10', 'full_name_11',
       'full_name_12', 'resale_price', 'registered_year', 'engine_capacity',
       'insurance_1', 'insurance_2', 'insurance_3', 'insurance_4',
       'transmission_type_1', 'transmission_type_2', 'kms_driven',
       'owner_type', 'fuel_type_1', 'fuel_type_2', 'fuel_type_3',
```

```
       'fuel_type_4', 'fuel_type_5', 'max_power', 'seats', 'mileage',
       'body_type_1', 'body_type_2', 'body_type_3', 'body_type_4',
       'body_type_5', 'body_type_6', 'body_type_7', 'city_1', 'city_2',
       'city_3', 'city_4', 'city_5', 'city_6', 'city_7', 'city_8', 'city_9',
       'city_10', 'city_11', 'city_12', 'city_13', 'registered_year_bin',
       'engine_capacity_bin', 'max_power_bin', 'mileage_bin',
       'kms_driven_discretized', 'seats_discretized',
       'resale_price_discretized'],
      dtype='object')
```

For the numerical attributes we can use pearson correlation coefficient to identify the correlation with target variable resale_price

Pearson correlation coefficient is used because it is a statistical measure that quantifies the linear relationship between two continuous variables.

[1349]:
```
from scipy.stats import pearsonr
corr_coef_age, p_value = pearsonr(df['registered_year'], df['resale_price'])
print('{:f}'.format(corr_coef_age))
```

```
0.514241
```

registered_year has a good postive correlation with the resale_price

[1350]:
```
from scipy.stats import pearsonr
corr_coef_age, p_value = pearsonr(df['engine_capacity'], df['resale_price'])
print('{:f}'.format(corr_coef_age))
```

```
0.516259
```

engine_capacity has a good postive correlation with the resale_price

[1351]:
```
from scipy.stats import pearsonr
corr_coef_age, p_value = pearsonr(df['kms_driven'], df['resale_price'])
print('{:f}'.format(corr_coef_age))
```

```
-0.164516
```

kms_driven has a negative correlation with the resale_price

[1352]:
```
from scipy.stats import pearsonr
corr_coef_age, p_value = pearsonr(df['max_power'], df['resale_price'])
print('{:f}'.format(corr_coef_age))
```

```
0.707157
```

max_power has a strong postive correlation with the resale_price

[1353]:
```
from scipy.stats import pearsonr
corr_coef_age, p_value = pearsonr(df['seats'], df['resale_price'])
print('{:f}'.format(corr_coef_age))
```

```
0.247481
```

seats has a postive correlation with the resale_price

[1354]:
```
from scipy.stats import pearsonr
corr_coef_age, p_value = pearsonr(df['mileage'], df['resale_price'])
print('{:f}'.format(corr_coef_age))
```

```
-0.314678
```

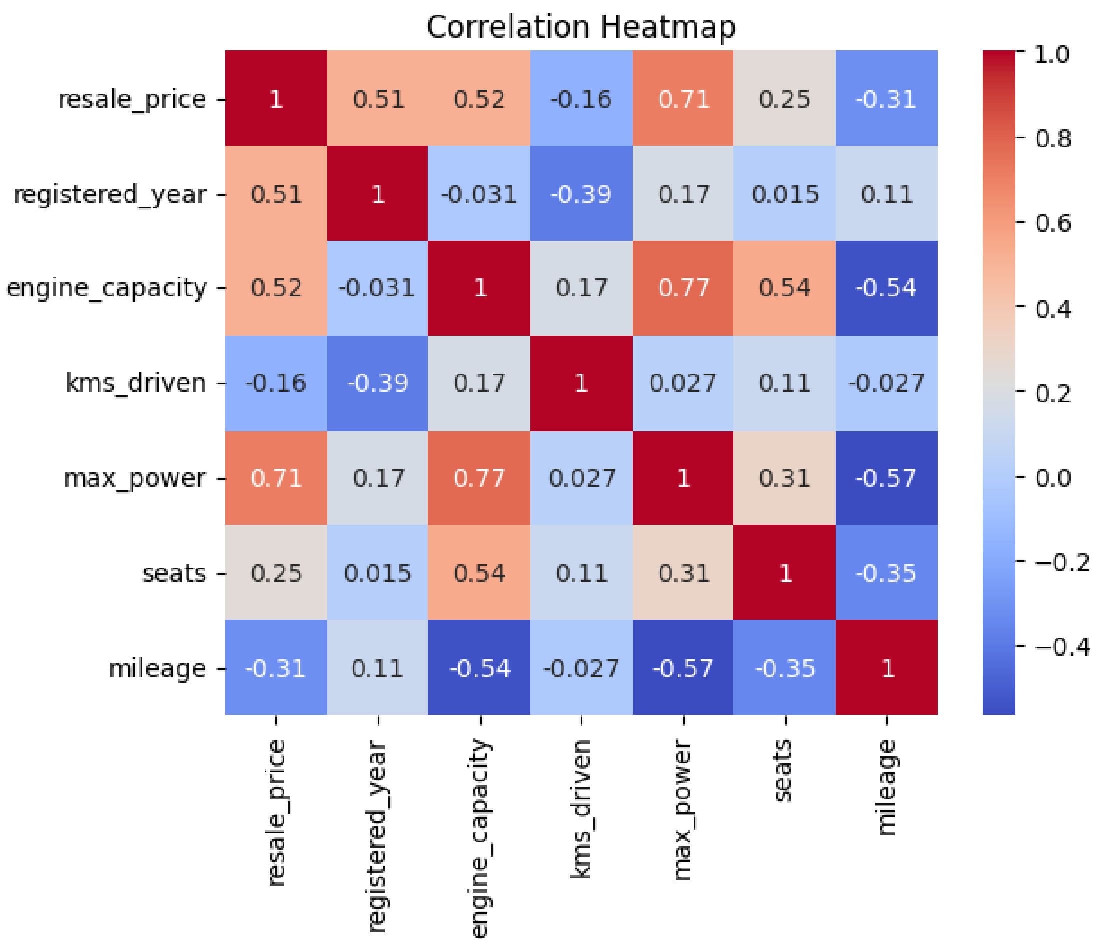
mileage has a good negative correlation with the resale_price

The correlation between the numerical attributes can be clearly visualized using a heatmap

[1355]:
```
df[['resale_price', 'registered_year',␣
↪'engine_capacity','kms_driven','max_power', 'seats', 'mileage']].corr()
```

[1355]:
```
                   resale_price  registered_year  engine_capacity  kms_driven  \
resale_price              1.000            0.514            0.516      -0.165
registered_year           0.514            1.000           -0.031      -0.390
engine_capacity           0.516           -0.031            1.000       0.174
kms_driven               -0.165           -0.390            0.174       1.000
max_power                 0.707            0.171            0.774       0.027
seats                     0.247            0.015            0.536       0.109
mileage                  -0.315            0.111           -0.544      -0.027

                   max_power  seats  mileage
resale_price           0.707  0.247   -0.315
registered_year        0.171  0.015    0.111
engine_capacity        0.774  0.536   -0.544
kms_driven             0.027  0.109   -0.027
max_power              1.000  0.307   -0.565
seats                  0.307  1.000   -0.347
mileage               -0.565 -0.347    1.000
```

[1356]:
```
sns.heatmap(df[['resale_price', 'registered_year',␣
↪'engine_capacity','kms_driven','max_power', 'seats', 'mileage']].corr(),␣
↪annot=True, cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```



From the above we can infer that registered_year, engine_capacity, max_power has a good correlation with the target variable. Enginer_capacity has a good correlation with max_power, seats and mileage. Highly correlated predictors can lead to collinearity issues and this can greatly affect the model performance. So it is better to consider only the ones which has high correlation value with the target variable. So we will consider only registered_year, engine_capacity, max_power for model prediction

0.3.29 Model Building and Prediction (4M)

Fit a linear regression model using the most important features identified(1M).Plot the visuals(1M).Briefly explain the regression model,equation (1M) and perform one prediction using the same(1M).

From the above feature selection and correlation analysis we have identified the list of important features. Lets use them to fit a linear regression model

[1357]:
```
# Selecting specific columns
X = df.loc[:, ['registered_year', 'engine_capacity', 'max_power',␣
↪'transmission_type_1','transmission_type_2','owner_type',
```

```
                'fuel_type_1', 'fuel_type_2', 'fuel_type_3',
        'fuel_type_4', 'fuel_type_5', 'body_type_1', 'body_type_2',␣
↪'body_type_3', 'body_type_4',
        'body_type_5', 'body_type_6', 'body_type_7','kms_driven_discretized']]
```

[1358]:
```
# Selecting the target variable
Y = df.loc[:,['resale_price']]
```

[1359]:
```
#Spliting the dataset into training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3,␣
↪random_state=0)
```

[1360]:
```
#Multi linear regression model from sklearn (least square errors)
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train, Y_train)
```

[1360]:
```
LinearRegression()
```

[1361]:
```
coefficients = model.coef_[0]
print('Coefficients = ')
# Printing coefficients with 3 decimal places using format method
for coeff in coefficients:
    print("{:.3f}".format(coeff))

intercept = model.intercept_
print('Intercept = ')
# Printing intercept with 3 decimal places using format method
for inter in intercept:
    print("{:.3f}".format(inter))
```

```
Coefficients =
0.006
0.019
0.203
-20615213.894
-20615213.869
-0.004
-3431306178.217
-3431306178.199
-3431306178.197
-3431306178.188
-3431306178.189
-21351042355.923
```

```
-21351042355.911
-21351042355.939
-21351042355.927
-21351042355.920
-21351042355.911
-21351042355.668
-0.009
Intercept =
24802963735.468
```

[1362]:
```
equation_parts = []
for i, col in enumerate(X.columns):
    equation_parts.append(f"{coefficients[i]:.3f} * {col}")
equation = " + ".join(equation_parts) + f" + {intercept[0]:.3f}"
print(f"Regression Equation: \nresale_price = {equation}")
```

```
Regression Equation:
resale_price = 0.006 * registered_year + 0.019 * engine_capacity + 0.203 *
max_power + -20615213.894 * transmission_type_1 + -20615213.869 *
transmission_type_2 + -0.004 * owner_type + -3431306178.217 * fuel_type_1 +
-3431306178.199 * fuel_type_2 + -3431306178.197 * fuel_type_3 + -3431306178.188
* fuel_type_4 + -3431306178.189 * fuel_type_5 + -21351042355.923 * body_type_1 +
-21351042355.911 * body_type_2 + -21351042355.939 * body_type_3 +
-21351042355.927 * body_type_4 + -21351042355.920 * body_type_5 +
-21351042355.911 * body_type_6 + -21351042355.668 * body_type_7 + -0.009 *
kms_driven_discretized + 24802963735.468
```

This model uses the above regression equation to predict the resale price

[1363]:
```
#Predicting the test data
Y_predicted = model.predict(X_test)
```

[1364]:
```
#printing the first actual and predicted values for comparison
print("Actual price : " + Y_test.values[0][0].astype(str))
print("Predicted price : " + Y_predicted[0][0].astype(str))
```

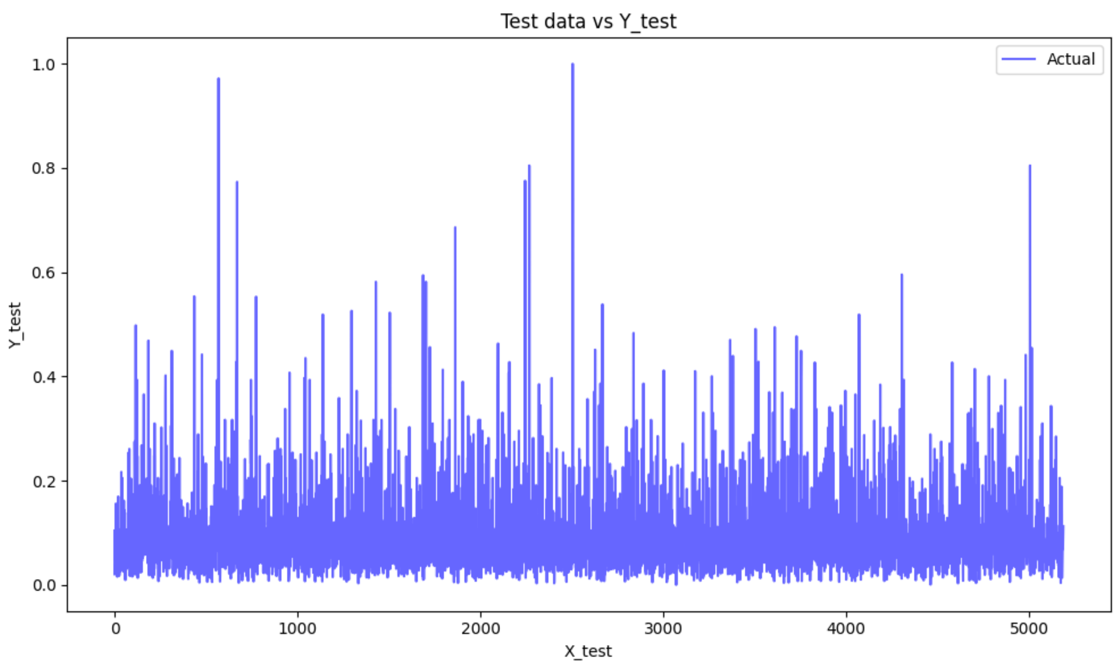
```
Actual price : 0.020496374790853318
Predicted price : 0.0179443359375
```

The above value is the normalized value. Lets do reverse normalize to find the actual price.

[1365]:
```
# Reverse normalization formula
actual_price = Y_test.values[0][0] * (resale_price_max - resale_price_min) +
 ↪resale_price_min
predicted_price = Y_predicted[0][0] * (resale_price_max - resale_price_min) +
 ↪resale_price_min
print("Actual price : Rs." + actual_price.astype(str))
print("Predicted price : Rs." + predicted_price.astype(str))
```

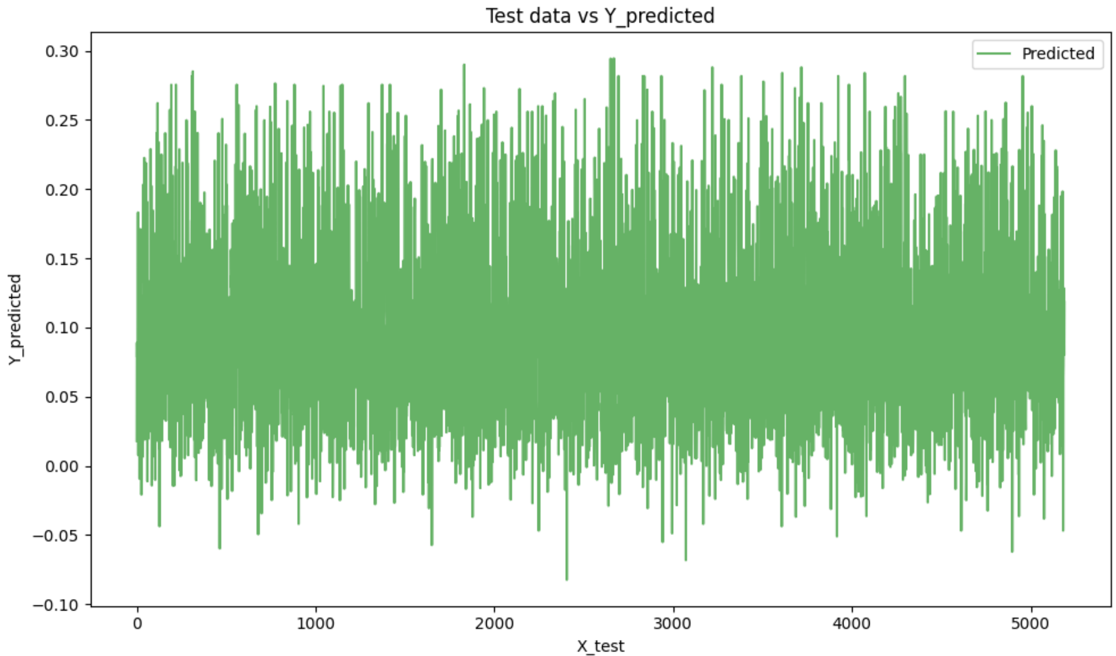
```
Actual price : Rs.175000.0
Predicted price : Rs.156696.77734375
```

[1366]:
```python
# Plotting test data vs Y_test
plt.figure(figsize=(10, 6))
plt.plot(range(len(X_test)), Y_test, color='blue', label='Actual', alpha=0.6)
plt.title('Test data vs Y_test')
plt.xlabel('X_test')
plt.ylabel('Y_test')
plt.legend()
plt.tight_layout()
plt.show()
```



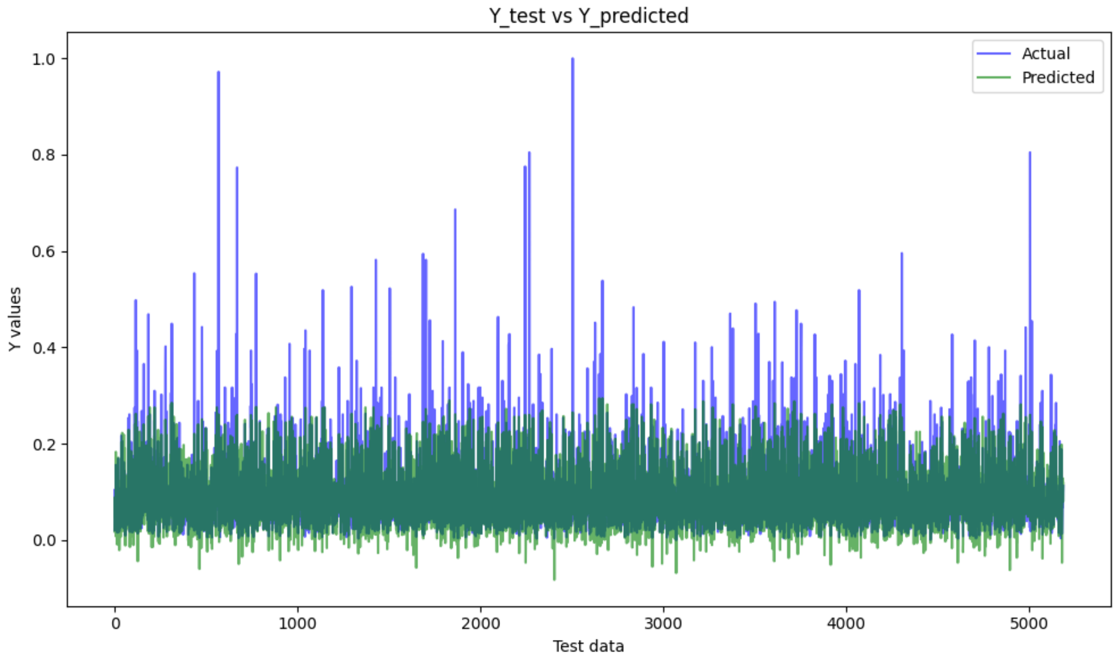
Above graph shoes the actual resale_price values for the test data

[1367]:
```python
# Plotting test data vs Y_predicted
plt.figure(figsize=(10, 6))
plt.plot(range(len(X_test)), Y_predicted, color='green', label='Predicted',␣
↪alpha=0.6)
plt.title('Test data vs Y_predicted')
plt.xlabel('X_test')
plt.ylabel('Y_predicted')
plt.legend()
plt.tight_layout()
plt.show()
```



Above graph shoes the predicted resale_price values for the test data

[1368]:
```
# Plotting Y_test vs Y_predicted
plt.figure(figsize=(10, 6))
plt.plot(range(len(X_test)), Y_test, color='blue', label='Actual', alpha=0.6)
plt.plot(range(len(X_test)), Y_predicted, color='green', label='Predicted',␣
↪alpha=0.6)
plt.title('Y_test vs Y_predicted')
plt.xlabel('Test data')
plt.ylabel('Y values')
plt.legend()
plt.tight_layout()
plt.show()
```



The above plot has both actual resale_price and predicted resale_price for the test data. Here we can see that the predicted price and actual price overlaps except in very rare scenarios.This means that our model's performance is good

[1369]:
```
from sklearn.metrics import mean_squared_error, r2_score
import math
def calculateModelMetrics(Y_actual,Y_predicted) :
    mse = mean_squared_error(Y_actual, Y_predicted)
    rmse = math.sqrt(mse)
    print("Mean squared error = ", mse)
    print("Root Mean squared error = ", rmse)
    print('Variance score = ', r2_score(Y_actual, Y_predicted))
```

[1370]:
```
calculateModelMetrics(Y_test, Y_predicted)
```

```
Mean squared error =  0.0018154901412748504
Root Mean squared error =  0.04260856887147056
Variance score =  0.692636963349542
```

0.3.30 Observations and Conclusions(1M)

Using the given data set we have followed the feature engineering principles to build a linear regression model with very less root mean squared error and high variance score to predict the resale price of a car based on its features.

0.3.31 Solution (1M)

What is the solution that is proposed to solve the business problem discussed in the beginning. Also share your learnings while working through solving the problem in terms of challenges, observations, decisions made etc.

Using the above model we can predict the resale price of a car based on its engine capacity, maximum power, fuel_type, transmission_type, registered_year etc. If we know the features of the car then it is easy to predict the resale price of a car. This can be used in multiple sites like olx, cars24, carwale etc to quote a price of an used car based on its features.

While working in this project I understood that a ML engineer should spent most of his time in feature engineering. Because it is important that the dataset should be cleaned and prepared properly to improve the model performance. We need to do analysis with the given dataset to understand the data better so that we can build a model which does the work with minimal errors.