Sheetal Waghmare

ID NO: 23692928

Starting with importing all the Necessary Libraries.

```
%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.stats import chi2_contingency
import seaborn as sns
sns.set(
    { "figure.figsize": (6, 4) },
    style='ticks',
    color_codes=True,
    font_scale=0.8
)
%config InlineBackend.figure_format = 'retina'
import warnings
warnings.filterwarnings('ignore')
```

```
cars=pd.read_csv('adverts.csv')
```

We have loaded our dataset in a variable named cars.

```
sample_cars=cars.sample(50000, ignore_index=True)
```

For reducing computational time and cognitive load we have taken a random sample of 50,000 rows from our dataset and stored in a variable named sample_cars.

1. Data Understanding and Exploration

### 1.1 Meaning and Type of Features; Analysis of Distributions

```
cars.columns
```

```
Index(['public_reference', 'mileage', 'reg_code', 'standard_colour',
       'standard_make', 'standard_model', 'vehicle_condition',
       'year_of_registration', 'price', 'body_type', 'crossover_car_and_van',
       'fuel_type'],
      dtype='object')
```

**Here we are looking at all the columns that are present in our dataset.**

```
cars['mileage'].describe()
```

| | mileage |
|---|---|
| **count** | 401878.000000 |
| **mean** | 37743.595656 |
| **std** | 34831.724018 |
| **min** | 0.000000 |
| **25%** | 10481.000000 |
| **50%** | 28629.500000 |
| **75%** | 56875.750000 |
| **max** | 999999.000000 |

***The mileage column indicates the total number of miles the car has been driven. Since higher mileage typically leads to greater depreciation in a car's value, it is a crucial feature in our dataset. The .describe() method provides the distribution statistics for the***

```
cars['price'].describe().round()
```

```
count      402005.0
mean        17342.0
std         46437.0
min           120.0
25%          7495.0
50%         12600.0
75%         20000.0
max       9999999.0
Name: price, dtype: float64
```

***The price is our target variable, and all other features assist in predicting the vehicle's price. The .describe() method provides the distribution statistics for the vehicle prices in our dataset, and .round(2) rounds the prices to two decimal places.***

```
cars['standard_make'].describe()
```

| | standard_make |
|---|---|
| **count** | 402005 |
| **unique** | 110 |
| **top** | BMW |
| **freq** | 37376 |

**dtype:** object

**The standard make feature gives us the information about the manufacturer of the car. We can see here that we have 110 different manufacturers, and the most common manufacturer is BMW, which has appeared more than 37,000 times in our dataset. The price of the car depends on manufacturer because some manufacturers only make high end luxury or very powerful sports cars which in general costs more than an average car.**

## 1.2 Analysis of Distributions (3-4)

```
cars['year_of_registration'].unique()
```

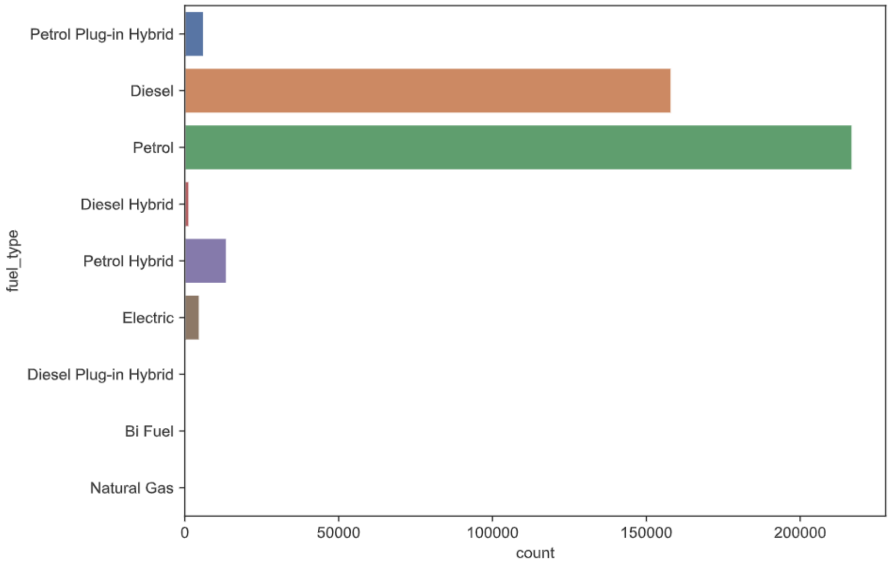
```
array([  nan, 2011., 2017., 2016., 2015., 2013., 2008., 2019., 2010.,
       2012., 2018., 2009., 1984., 2014., 2003., 2006., 2020., 2005.,
       2000., 2002., 2007., 2004., 1991., 2001., 1986., 1998., 1990.,
       1993., 1987., 1994., 1999., 1970., 1988., 1995., 1997., 1969.,
       1992., 1989., 1996., 1976., 1983., 1980., 1973., 1962., 1967.,
       1972., 1982., 1968., 1979., 1964., 1933., 1981., 1985., 1978.,
       1971., 1974., 1966., 1977., 1961., 1965., 1007., 1957., 1515.,
       1963., 1063., 1954., 1975., 1955., 1009., 1016., 1960., 1956.,
       1959., 1909., 1934., 1958., 1010., 1950., 1008., 1018.,  999.,
       1017., 1952., 1006., 1015.])
```

```
cars['year_of_registration'].describe()
```

```
count    368694.000000
mean       2015.006206
std           7.962667
min         999.000000
25%        2013.000000
50%        2016.000000
75%        2018.000000
max        2020.000000
Name: year_of_registration, dtype: float64
```

The year_of_registration column gives us the year the car was registered in. .describe() method helps us analyze the distribution of the feature. We can see that the majority of the cars are registered around the year 2016. The newest one is for the year 2020 and the oldest we have in our dataset is from the 999, which is most likely a wrong entry because cars were not made until $19^{th}$ century.

```
sns.countplot(y='fuel_type',data=cars)
plt.show()
```

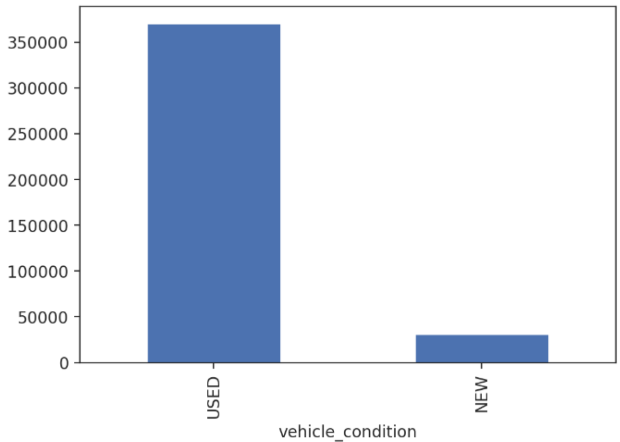



```
cars['fuel_type'].value_counts()
```

```
Petrol                   216929
Diesel                   158120
Petrol Hybrid             13602
Petrol Plug-in Hybrid      6160
Electric                   4783
Diesel Hybrid              1403
Bi Fuel                     221
Diesel Plug-in Hybrid       185
Natural Gas                   1
Name: fuel_type, dtype: int64
```

Here, we are using a countplot to visualize the fuel_type feature. This feature indicates the type of fuel the cars in our dataset use. The graph shows that the majority of cars run on Petrol or Diesel. To get the exact numbers, we use the .value_counts() method, which provides the precise count of vehicles for each fuel type. Over 210,000 cars, accounting for more than 50% of our dataset, run on petrol, while over 150,000 cars use diesel. Together, nearly 90% of the cars in our dataset are either petrol or diesel-powered, with only a small number using other fuels and just one car running on natural gas.

```
cars['vehicle_condition'].value_counts().plot.bar();
```




```
cars['vehicle_condition'].value_counts()
```

```
USED    370756
NEW      31249
Name: vehicle_condition, dtype: int64
```

We are examining the vehicle_condition feature here. The .unique() method provides all the unique values in this column. We can see that this feature is an object datatype containing only two values: "USED" and "NEW." Typically, new cars are priced higher than used ones.With the help of bar plot we can see that majority of the cars in our dataset are used, which is more than 90% and les than 10% are new cars.

**2 Data Pre-Processing**

**2.1. Data Cleaning**

```
cars.isnull().sum()
```

```
public_reference             0
mileage                    127
reg_code                 31857
standard_colour           5378
standard_make                0
standard_model               0
vehicle_condition            0
year_of_registration     33311
price                        0
body_type                  837
crossover_car_and_van        0
fuel_type                  601
Mileage_Range            16334
dtype: int64
```

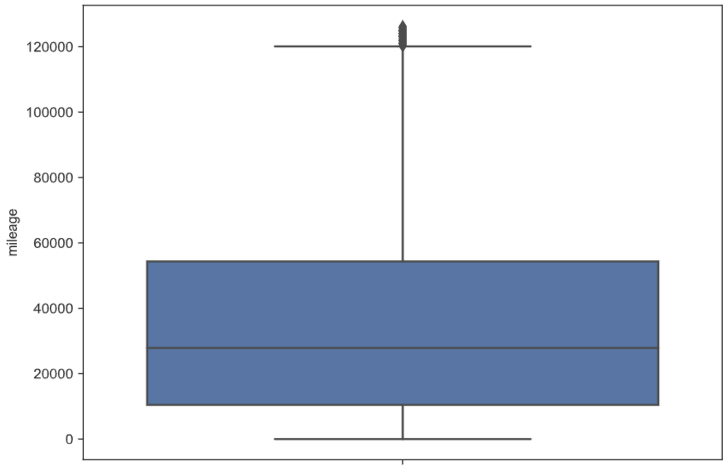
The isnull() function identifies whether a column contains any null values, and .sum() totals the number of null values within the column. From this, we can observe that the columns mileage, reg_code, standard_colour, year_of_registration, body_type, and fuel_type all contain null values.

```
def iqr(df,col):
    q1=df[col].quantile(0.25)
    q3=df[col].quantile(0.75)
    IQR=q3-q1
    outliers = (df[col]<(q1-1.5*IQR)) | (df[col]>(q3+1.5*IQR))
    df.drop(df[outliers].index, inplace=True)
```

```
iqr(cars,'mileage')
```

***For mileage feature, To deal with the outliers we are using IQR, we have created a function which uses IQR to drop outliers from our column.***

```
sns.boxplot(y='mileage', data=sam)
plt.show()
```




***The spread of mileage feature after dropping the outliers, majority of values are in the range of 10,000 to 55,000, with few going above 100,000.***

```python
def get_outliers_dataframe(data_frame, column_name):
    Q1 = data_frame[column_name].quantile(0.25)
    Q3 = data_frame[column_name].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    outliers = (data_frame[column_name] < lower_bound) | (data_frame[column_name] > upper_bound)
    outliers_df = data_frame[outliers]
    return outliers_df
outliers_dataframe = get_outliers_dataframe(cars, 'year_of_registration')
outliers_dataframe['year_of_registration'].describe()
```

```
count    9843.000000
mean     1999.415727
std        40.802102
min       999.000000
25%      2001.000000
50%      2003.000000
75%      2005.000000
max      2005.000000
Name: year_of_registration, dtype: float64
```

For the `year_of_registration` feature, we're using the IQR method again, but instead of directly removing outliers, we've stored them in a separate dataframe. By using `.describe()`, we can examine the outliers in the `year_of_registration` column, where the IQR indicates that any registration year prior to 2005 is considered an outlier.

```python
cars = cars[(cars['year_of_registration'] >= 1980) | (np.isnan(cars['year_of_registration']))]
```

***We are dropping all the cars which are registered before 1980 because they are too old and there is less chance someone will buy or sell car that old.***

```python
cars.query('price<9999999').sort_values(by='price')
```

| | public_reference | mileage | reg_code | standard_colour | standard_make | standard_model | vehicle_condition | year_of_registration | price | body_type |
|---|---|---|---|---|---|---|---|---|---|---|
| **332532** | 202010315653263 | 78000.0 | W | Blue | Citroen | Saxo | USED | 2000.0 | 120 | Hatchback |
| **300445** | 202011015671489 | 89000.0 | W | Green | Vauxhall | Corsa | USED | 2000.0 | 122 | Hatchback |
| **109133** | 202010295564975 | 117500.0 | 52 | Green | Citroen | C3 | USED | 2002.0 | 180 | Hatchback |
| **235899** | 202010165069699 | 115000.0 | 53 | Black | Vauxhall | Agila | USED | 2003.0 | 200 | Hatchback |
| **91878** | 202009083473638 | 100000.0 | 56 | Silver | Renault | Clio | USED | 2006.0 | 200 | Hatchback |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **94033** | 202007020778467 | 1900.0 | 18 | White | Pagani | Huayra | USED | NaN | 2400000 | Convertible |
| **223835** | 202007081011555 | 1000.0 | 67 | Blue | Bugatti | Chiron | USED | 2018.0 | 2500000 | Coupe |
| **64910** | 202006039766650 | 189.0 | NaN | Black | McLaren | P1 | USED | NaN | 2695000 | Coupe |
| **51741** | 202002257718775 | 4400.0 | 14 | Black | Bugatti | Veyron | USED | 2014.0 | 2850000 | Coupe |
| **198060** | 202008252907180 | 300.0 | 17 | NaN | Ferrari | LaFerrari | USED | 2017.0 | 3799995 | NaN |

393137 rows × 13 columns

For the `price` feature, we noticed that a few cars were listed with a price of 9,999,999, which are likely incorrect entries. Using `.query()`, we filtered the data to include only prices below 9,999,999. We found that while some cars are priced over 3 million, they belong to high-end luxury brands like Bugatti and Ferrari, which typically have such high prices. Therefore, we decided to retain these entries and remove only the cars with the price of 9,999,999.

**2.2. Feature Engineering**

```python
#Group by 'standard_make' and calculating the mean price
brand_means = cars.groupby('standard_make')['price'].mean().sort_values()

#Create a mapping from 'standard_make' to label based on mean values
brand_mapping = {standard_make: label for label, standard_make in enumerate(brand_means.index)}

#Map the 'standard_make' column to labels
cars['standard_make'] = cars['standard_make'].map(brand_mapping)
```

```
cars['year_of_registration']=cars['year_of_registration'].fillna(-1)
cars['year_of_registration']=cars['year_of_registration'].astype('int64')
```

```
reg={'A':1983,'B':1984,'C':1985,'CA':1985,'D':1986,'E':1987,'F':1988,'G':1989,'H':1990,'J':1991,
    'L':1993,'M':1994,'N':1995,'P':1996,'R':1997,'S':1998,'T':1999,'V':2000,'Y':2001}
cars.loc[cars['year_of_registration'] == -1, 'year_of_registration'] = cars['reg_code'].map(reg)
```

***Using information from the Wikipedia page on UK vehicle registration plates, we identified the corresponding values for the letters in the `reg_code` column and added this data to our dataset. For any NaN values, we replaced them with -1 to simplify identification and handling, and we also converted the `year_of_registration` datatype to an integer.***

```
cars['year_of_registration'] = cars['year_of_registration'].mask((cars['vehicle_condition']=='NEW'), datetime.now().year)
```

***Wherever the condition of the vehicle was NEW we have put the current_year in the year_of_registration column.***

```
cars['body_type'] = cars['body_type'].replace({'Pickup':'Van','Combi Van':'Van','Minibus': 'Van', 'Camper': 'Van',
'Panel Van': 'Van','Window Van':'Van','Chassis Cab':'Van','Car Derived Van': 'Van'})

print(cars['body_type'].value_counts())
```

```
Hatchback      165755
SUV            114136
Saloon          34933
Estate          23404
Coupe           22674
Convertible     15599
MPV             15473
Van              1031
Limousine         132
Name: body_type, dtype: int64
```

For the `body_type` feature, we've merged the categories (pickup, combi van, minibus, camper, panel van, window van, chassis cab, car derived van) into a single category called "Van," since they are all variations of vans. This consolidation helps reduce dimensionality in the dataset.

```
body = cars.groupby('body_type')['price'].mean().sort_values()

body_mapping = {body_type: label for label, body_type in enumerate(body.index)}

cars['body_type'] = cars['body_type'].map(body_mapping)
```

```
print(cars['body_type'].value_counts())
```

```
0    165755
5    114136
3     34933
2     23404
7     22674
6     15599
1     15473
4      1031
8       132
Name: body_type, dtype: int64
```

```
print(body_mapping)
```

```
{'Hatchback': 0, 'MPV': 1, 'Estate': 2, 'Saloon': 3, 'Van': 4, 'SUV': 5, 'Convertible': 6, 'Coupe': 7, 'Limousine': 8}
```

We then applied the same Label Encoding method used for the `standard_make` feature to transform the `body_type` feature. Hatchback was encoded as 0 since it has the lowest average price, while limousine received the highest label due to its highest average price.

**2.3. Sub setting**

```python
from sklearn.feature_selection import SelectKBest, f_regression

X = cars.drop(columns=['price','standard_model'])
y = cars['price']

# Selecting top 5 features
selector = SelectKBest(f_regression, k=5)
selector.fit(X, y)

# Get names of top 5 features
top_features = X.columns[selector.get_support()]
top_features
```

```
Index(['mileage', 'standard_make', 'vehicle_condition', 'year_of_registration',
       'body_type'],
      dtype='object')
```

The top five features most strongly correlated with the price variable are `mileage`, `standard_make`, `vehicle_condition`, `year_of_registration`, and `body_type`. This correlation makes sense; for instance, lower mileage typically means a higher resale value. The `standard_make` affects car value since luxury brands retain high prices even when used or older, due to their premium materials and engineering. `Vehicle_condition` is straightforward—new vehicles are priced higher than used ones. The `year_of_registration` also impacts price, as newer registrations often indicate more recent models with lower mileage. Lastly, `body_type` matters, as different types of cars, like coupes, generally cost more than hatchbacks.

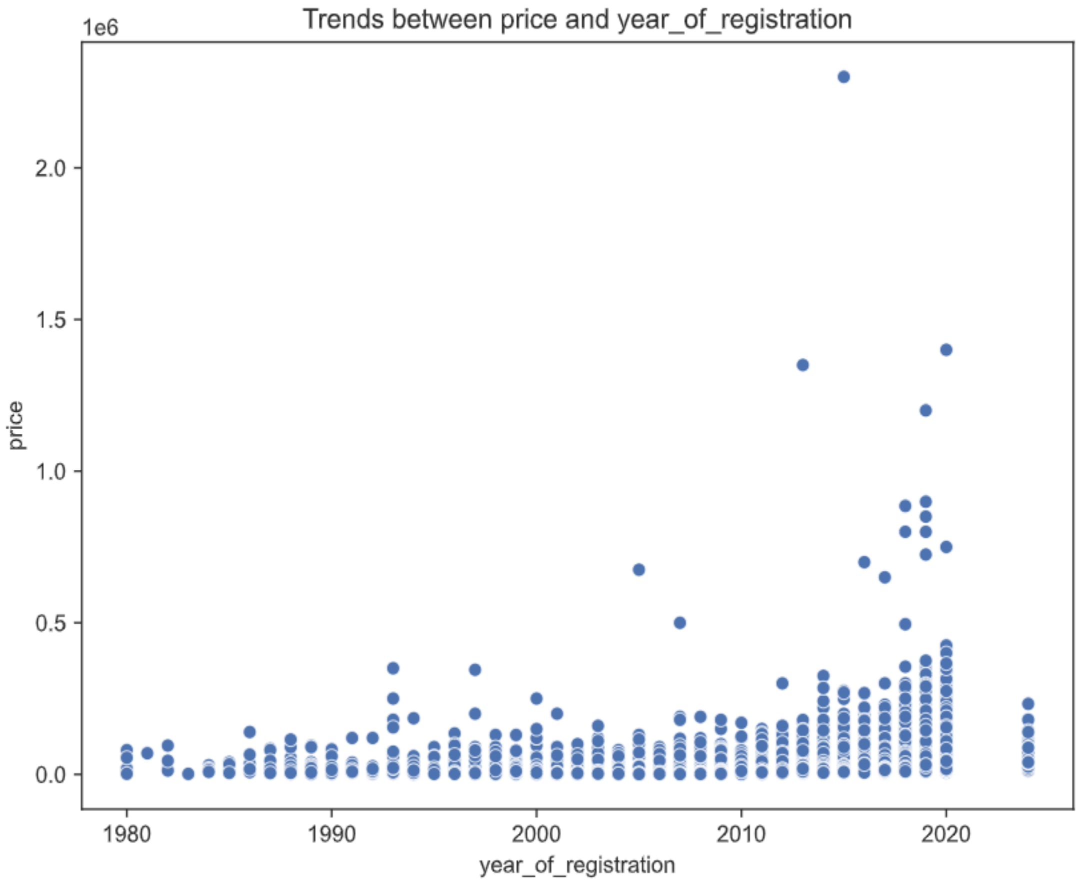
```python
cars.drop(columns=['reg_code','standard_model','public_reference'])
```

**We are using .drop for dropping columns[reg_code, standard_model, public_reference] which are less correlated with our Target variable(Price)**

**3 Analysis of Associations and Group Differences**

**3.1. Quantitative-Quantitative**

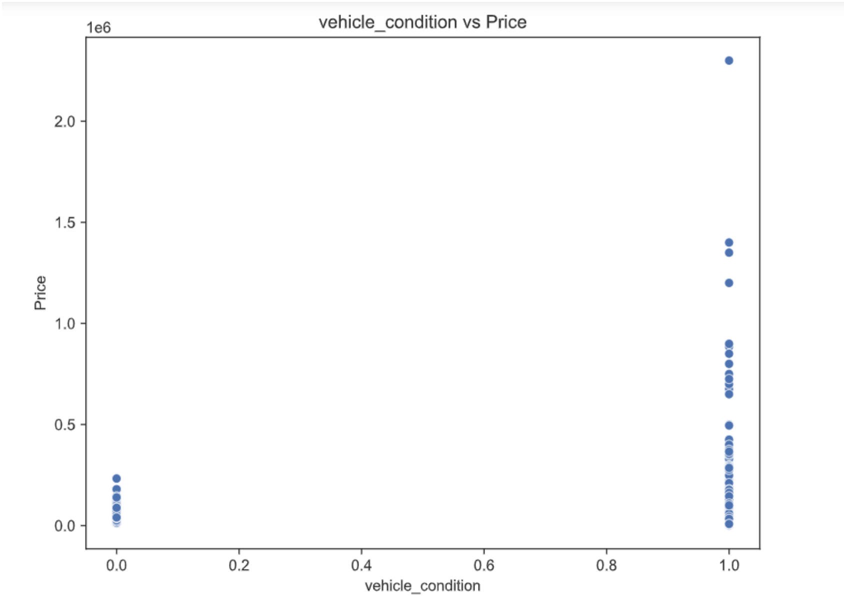
```python
sns.scatterplot(x='year_of_registration', y='price', data=cars_sample)
plt.xlabel('year_of_registration')
plt.ylabel('price')
plt.title('Trends between price and year_of_registration')
plt.show()
```




We are using a scatterplot to explore the relationship between the `year_of_registration` and `price` features. The plot shows that older cars tend to have lower values compared to newer ones. There is a noticeable upward trend, with prices increasing as the year of registration becomes more recent.

**3.2 Categorical – Quantitative**

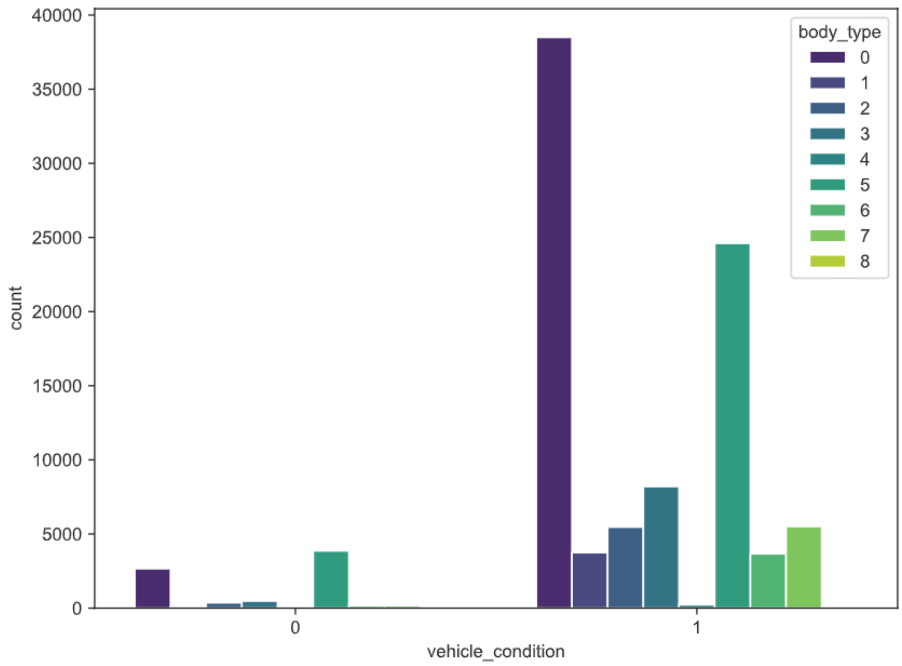
```
sns.scatterplot(x='vehicle_condition', y='price', data=cars_sample)
plt.title("vehicle_condition vs Price")
plt.xlabel("vehicle_condition")
plt.ylabel("Price")
plt.show()
```




Some used vehicles have very high prices due to luxury brands like Ferrari, which broadens the price range for used cars. In contrast, new cars have a much narrower price range.

### 3.3 Categorical-Categorical

```
sns.countplot(x='vehicle_condition', hue='body_type', data=cars_sample, palette='viridis', saturation=0.8)
plt.show()
```




```
print(body_mapping)
```

```
{'Hatchback': 0, 'MPV': 1, 'Estate': 2, 'Saloon': 3, 'Van': 4, 'SUV': 5, 'Convertible': 6, 'Coupe': 7, 'Limousine': 8}
```

Here we can see that we have huge number of used hatchback stating it's the most popular body_type in our dataframe, we also have good amount of convertibles making it the second most popular while very less number of used SUV in the market.
In the New category we again have two of the most popular body types as Convertible and hatchback.