Università degli Studi di Bari Aldo Moro

FACOLTÀ DI INFORMATICA
Corso di Laurea Triennale in Informatica

Caso di Studio del corso in Ingegneria della Conoscenza

SISTEMA INTELLIGENTE PER LA PREDIZIONE DEL PREZZO DI AUTO BASATO SU ALGORITMI DI APPRENDIMENTO SUPERVISIONATO
-
GESTIONE DEI VEICOLI ATTRAVERSO UNA KNOWLEDGE BASE

GitHub Repository: https://github.com/UniBaBoyz/Corona

Autori:

Tanzi Giuseppe
Matricola 697177

Papeo Alessandro
Matricola 703108

Stelluti Michele
Matricola 705052

Susso Vincenzo
Matricola 697538

Anno Accademico 2020–2021

1 Introduzione

Una società automobilistica giapponese aspira ad entrare nel mercato statunitense stabilendo lì la propria fabbrica di produzione e producendo auto a livello locale per dare concorrenza alle controparti statunitensi ed europee.

Una società di consulenza automobilistica è stata incaricata per comprendere i fattori da cui dipende il prezzo delle auto. Nello specifico, si vogliono comprendere i fattori che influenzano il prezzo delle auto nel mercato americano, poiché questi possono essere molto diversi dal mercato giapponese. L'azienda, in particolare, vuole conoscere:

```
- Quali sono le variabili significative nella previsione del prezzo di un'auto
- In che misura queste variabili influenzano il prezzo di un'auto
```

Sulla base di varie indagini di mercato, la società di consulenza ha raccolto un ampio set di dati di diversi tipi di auto in tutto il mercato americano.

2 Obiettivo aziendale

È necessario modellare il prezzo delle auto con le variabili indipendenti disponibili. Esse saranno utilizzate dal management per capire quanto variano esattamente i prezzi con le variabili indipendenti. Di conseguenza si potrà modellare il design delle auto, la strategia aziendale, ecc… per soddisfare determinati livelli di prezzo. Inoltre, il modello di predizione sarà un buon modo per comprendere le dinamiche di prezzo di un nuovo mercato.

[1]:
```
#Importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

3 Step 1: Leggere e comprendere il dataset

Iniziamo con i seguenti passi:

1. Importare il dataset usando la libreria di pandas
2. Comprendere la struttura del dataset

[2]:
```
cars = pd.read_csv('data/CarPrice.csv')
cars.head()
```

[2]:

	car_ID	symboling	CarName	fueltype	aspiration	doornumber	\
0	1	3	alfa-romero giulia	gas	std	two	
1	2	3	alfa-romero stelvio	gas	std	two	
2	3	1	alfa-romero Quadrifoglio	gas	std	two	
3	4	2	audi 100 ls	gas	std	four	
4	5	2	audi 100ls	gas	std	four	

	carbody	drivewheel	enginelocation	wheelbase	...	enginesize	\

```
0  convertible        rwd          front       88.6  ...         130
1  convertible        rwd          front       88.6  ...         130
2    hatchback        rwd          front       94.5  ...         152
3        sedan        fwd          front       99.8  ...         109
4        sedan        4wd          front       99.4  ...         136

   fuelsystem  boreratio  stroke compressionratio horsepower  peakrpm citympg  \
0        mpfi       3.47    2.68              9.0        111     5000       21
1        mpfi       3.47    2.68              9.0        111     5000       21
2        mpfi       2.68    3.47              9.0        154     5000       19
3        mpfi       3.19    3.40             10.0        102     5500       24
4        mpfi       3.19    3.40              8.0        115     5500       18

   highwaympg    price
0          27  13495.0
1          27  16500.0
2          26  16500.0
3          30  13950.0
4          22  17450.0

[5 rows x 26 columns]
```

[3]:
```
cars.shape
```

[3]:
```
(205, 26)
```

[4]:
```
cars.describe()
```

[4]:
```
           car_ID   symboling   wheelbase   carlength    carwidth   carheight  \
count  205.000000  205.000000  205.000000  205.000000  205.000000  205.000000
mean   103.000000    0.834146   98.756585  174.049268   65.907805   53.724878
std     59.322565    1.245307    6.021776   12.337289    2.145204    2.443522
min      1.000000   -2.000000   86.600000  141.100000   60.300000   47.800000
25%     52.000000    0.000000   94.500000  166.300000   64.100000   52.000000
50%    103.000000    1.000000   97.000000  173.200000   65.500000   54.100000
75%    154.000000    2.000000  102.400000  183.100000   66.900000   55.500000
max    205.000000    3.000000  120.900000  208.100000   72.300000   59.800000

         curbweight  enginesize   boreratio      stroke  compressionratio  \
count   205.000000  205.000000  205.000000  205.000000        205.000000
mean   2555.565854  126.907317    3.329756    3.255415         10.142537
std     520.680204   41.642693    0.270844    0.313597          3.972040
min    1488.000000   61.000000    2.540000    2.070000          7.000000
25%    2145.000000   97.000000    3.150000    3.110000          8.600000
50%    2414.000000  120.000000    3.310000    3.290000          9.000000
75%    2935.000000  141.000000    3.580000    3.410000          9.400000
max    4066.000000  326.000000    3.940000    4.170000         23.000000
```

```
       horsepower      peakrpm     citympg  highwaympg         price
count  205.000000   205.000000  205.000000  205.000000    205.000000
mean   104.117073  5125.121951   25.219512   30.751220  13276.710571
std     39.544167   476.985643    6.542142    6.886443   7988.852332
min     48.000000  4150.000000   13.000000   16.000000   5118.000000
25%     70.000000  4800.000000   19.000000   25.000000   7788.000000
50%     95.000000  5200.000000   24.000000   30.000000  10295.000000
75%    116.000000  5500.000000   30.000000   34.000000  16503.000000
max    288.000000  6600.000000   49.000000   54.000000  45400.000000
```

[5]:
```
cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   car_ID            205 non-null    int64
 1   symboling         205 non-null    int64
 2   CarName           205 non-null    object
 3   fueltype          205 non-null    object
 4   aspiration        205 non-null    object
 5   doornumber        205 non-null    object
 6   carbody           205 non-null    object
 7   drivewheel        205 non-null    object
 8   enginelocation    205 non-null    object
 9   wheelbase         205 non-null    float64
 10  carlength         205 non-null    float64
 11  carwidth          205 non-null    float64
 12  carheight         205 non-null    float64
 13  curbweight        205 non-null    int64
 14  enginetype        205 non-null    object
 15  cylindernumber    205 non-null    object
 16  enginesize        205 non-null    int64
 17  fuelsystem        205 non-null    object
 18  boreratio         205 non-null    float64
 19  stroke            205 non-null    float64
 20  compressionratio  205 non-null    float64
 21  horsepower        205 non-null    int64
 22  peakrpm           205 non-null    int64
 23  citympg           205 non-null    int64
 24  highwaympg        205 non-null    int64
 25  price             205 non-null    float64
dtypes: float64(8), int64(8), object(10)
memory usage: 41.8+ KB
```

4 Step 2 : Pulizia e preparazione del dataset

- Separiamo il nome della macchina dal nome dell'azienda ed eliminiamo la feature del nome della macchina dal dataset, poichè inutile.

[6]:
```
#Splitting company name from CarName column
CompanyName = cars['CarName'].apply(lambda x : x.split(' ')[0])
cars.insert(3,"CompanyName",CompanyName)
cars.drop(['CarName'],axis=1,inplace=True)
cars.head()
```

[6]:
```
   car_ID  symboling  CompanyName fueltype aspiration doornumber      carbody  \
0       1          3  alfa-romero      gas        std        two  convertible
1       2          3  alfa-romero      gas        std        two  convertible
2       3          1  alfa-romero      gas        std        two    hatchback
3       4          2         audi      gas        std       four        sedan
4       5          2         audi      gas        std       four        sedan

  drivewheel enginelocation  wheelbase  ...  enginesize  fuelsystem  \
0        rwd          front       88.6  ...         130        mpfi
1        rwd          front       88.6  ...         130        mpfi
2        rwd          front       94.5  ...         152        mpfi
3        fwd          front       99.8  ...         109        mpfi
4        4wd          front       99.4  ...         136        mpfi

   boreratio  stroke compressionratio horsepower  peakrpm citympg  highwaympg  \
0       3.47    2.68              9.0        111     5000      21          27
1       3.47    2.68              9.0        111     5000      21          27
2       2.68    3.47              9.0        154     5000      19          26
3       3.19    3.40             10.0        102     5500      24          30
4       3.19    3.40              8.0        115     5500      18          22

     price
0  13495.0
1  16500.0
2  16500.0
3  13950.0
4  17450.0

[5 rows x 26 columns]
```

- Vediamo i nomi di tutte le aziende presenti nel dataset.

[7]:
```
cars.CompanyName.unique()
```

[7]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'maxda', 'mazda', 'buick', 'mercury',
       'mitsubishi', 'Nissan', 'nissan', 'peugeot', 'plymouth', 'porsche',
```

```
       'porcshce', 'renault', 'saab', 'subaru', 'toyota', 'toyouta',
       'vokswagen', 'volkswagen', 'vw', 'volvo'], dtype=object)
```

4.1 Rinominazione dei nomi delle aziende

- Sembrano essersi alcuni errori di spelling nella colonna CompanyName.
  - maxda = mazda
  - Nissan = nissan
  - porsche = porcshce
  - toyota = toyouta
  - vokswagen = volkswagen = vw

[8]:
```
cars.CompanyName = cars.CompanyName.str.lower()

def replace_name(a,b):
    cars.CompanyName.replace(a,b,inplace=True)

replace_name('maxda','mazda')
replace_name('porcshce','porsche')
replace_name('toyouta','toyota')
replace_name('vokswagen','volkswagen')
replace_name('vw','volkswagen')

cars.CompanyName.unique()
```

[8]:
```
array(['alfa-romero', 'audi', 'bmw', 'chevrolet', 'dodge', 'honda',
       'isuzu', 'jaguar', 'mazda', 'buick', 'mercury', 'mitsubishi',
       'nissan', 'peugeot', 'plymouth', 'porsche', 'renault', 'saab',
       'subaru', 'toyota', 'volkswagen', 'volvo'], dtype=object)
```

[9]:
```
#Checking for duplicates
cars.loc[cars.duplicated()]
```

[9]:
```
Empty DataFrame
Columns: [car_ID, symboling, CompanyName, fueltype, aspiration, doornumber,
carbody, drivewheel, enginelocation, wheelbase, carlength, carwidth, carheight,
curbweight, enginetype, cylindernumber, enginesize, fuelsystem, boreratio,
stroke, compressionratio, horsepower, peakrpm, citympg, highwaympg, price]
Index: []

[0 rows x 26 columns]
```

- Vediamo tutte le feature del dataset

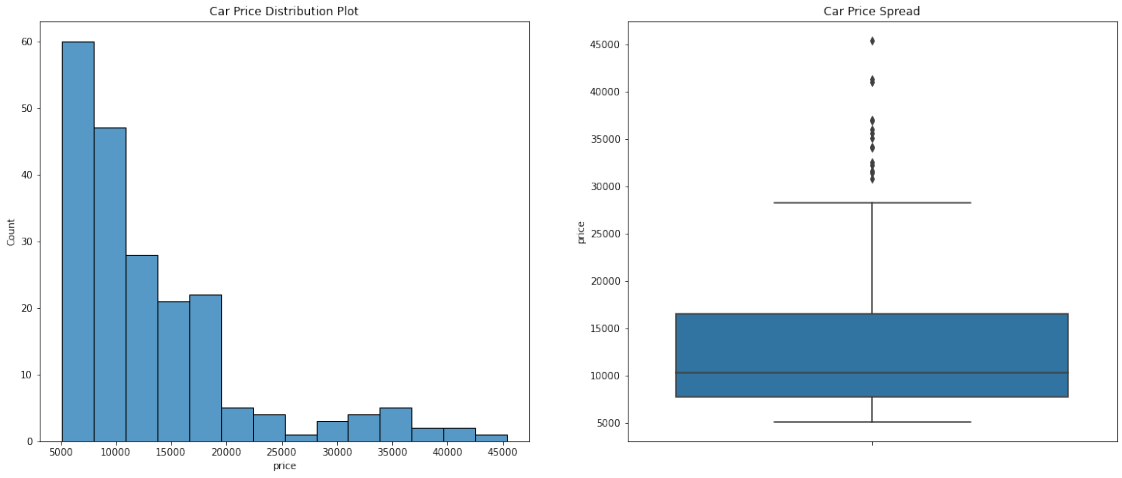
[10]:
```
cars.columns
```

[10]:
```
Index(['car_ID', 'symboling', 'CompanyName', 'fueltype', 'aspiration',
       'doornumber', 'carbody', 'drivewheel', 'enginelocation', 'wheelbase',
```

```
       'carlength', 'carwidth', 'carheight', 'curbweight', 'enginetype',
       'cylindernumber', 'enginesize', 'fuelsystem', 'boreratio', 'stroke',
       'compressionratio', 'horsepower', 'peakrpm', 'citympg', 'highwaympg',
       'price'],
      dtype='object')
```

5 Step 3: Visualizzazione delle feature

[11]:
```
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
plt.title('Car Price Distribution Plot')
sns.histplot(cars.price)

plt.subplot(1,2,2)
plt.title('Car Price Spread')
sns.boxplot(y=cars.price)

plt.show()
```



[12]:
```
print(cars.price.describe(percentiles = [0.25,0.50,0.75,0.85,0.90,1]))
```

```
count      205.000000
mean     13276.710571
std       7988.852332
min       5118.000000
25%       7788.000000
50%      10295.000000
75%      16503.000000
85%      18500.000000
```

```
90%       22563.000000
100%      45400.000000
max       45400.000000
Name: price, dtype: float64
```

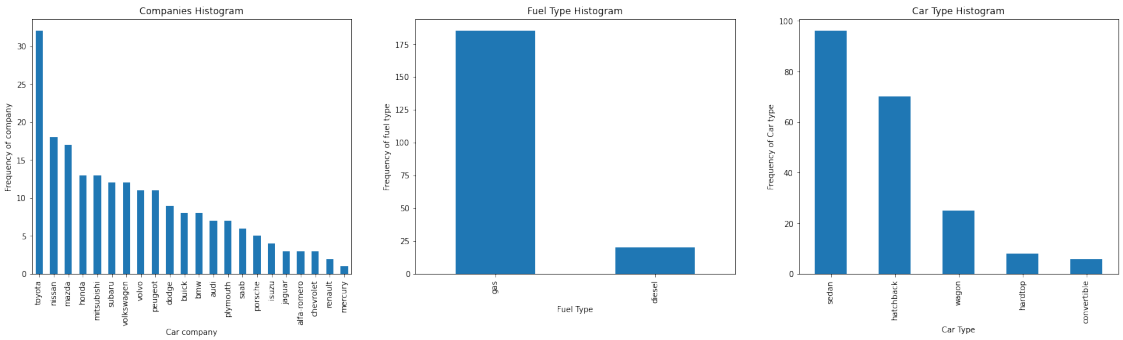
5.0.1 Inferenza :

1. Il grafico mostra che la maggior parte dei prezzi nel set di dati è bassa (inferiore a 15.000).
2. Esiste una differenza significativa tra la `media` e la `mediana` della distribuzione dei prezzi.
3. I dati sono molto distanti dalla `media`, il che indica un'elevata variazione dei prezzi delle auto (l'85% dei prezzi è inferiore a 18.500, mentre il restante 15% è compreso tra 18.500 e 45.400).

5.1 Step 3.1 : Visualizzazione delle feature categoriche

```
- CompanyName
- Symboling
- fueltype
- enginetype
- carbody
- doornumber
- enginelocation
- fuelsystem
- cylindernumber
- aspiration
- drivewheel
```

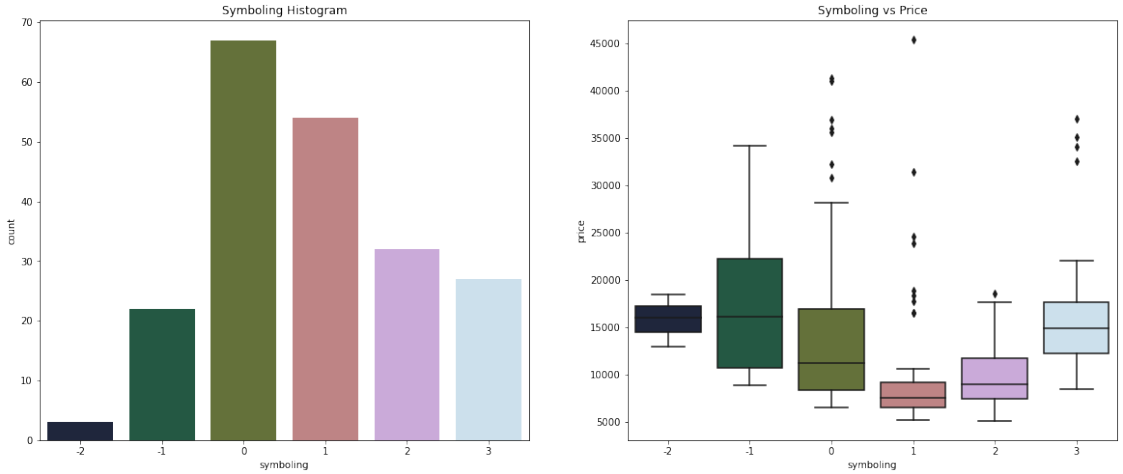
[13]:
```
plt.figure(figsize=(25, 6))

plt.subplot(1,3,1)
plt1 = cars.CompanyName.value_counts().plot(kind ='bar')
plt.title('Companies Histogram')
plt1.set(xlabel = 'Car company', ylabel='Frequency of company')

plt.subplot(1,3,2)
plt1 = cars.fueltype.value_counts().plot(kind = 'bar')
plt.title('Fuel Type Histogram')
plt1.set(xlabel = 'Fuel Type', ylabel='Frequency of fuel type')

plt.subplot(1,3,3)
plt1 = cars.carbody.value_counts().plot(kind = 'bar')
plt.title('Car Type Histogram')
plt1.set(xlabel = 'Car Type', ylabel='Frequency of Car type')

plt.show()
```



5.1.1 Inferenza :

1. `Toyota` sembra essere l'azienda favorita.
2. Il numero delle macchine alimentate a `gas` sono di numero maggiore rispetto a quelle alimentate a `diesel`.
3. `sedan` è il tipo di macchina preferito.

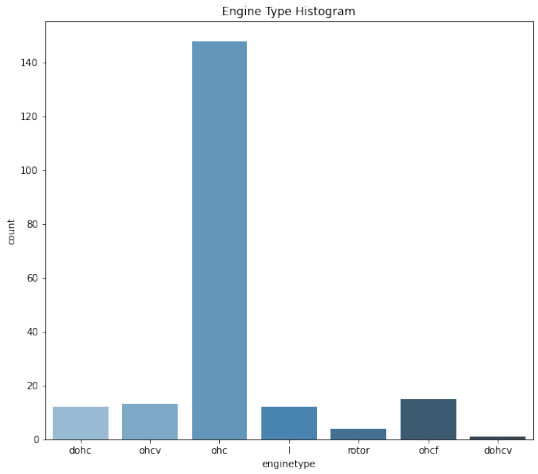
[14]:
```
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
plt.title('Symboling Histogram')
sns.countplot(x=cars.symboling, palette=("cubehelix"))

plt.subplot(1,2,2)
plt.title('Symboling vs Price')
sns.boxplot(x=cars.symboling, y=cars.price, palette=("cubehelix"))

plt.show()
```

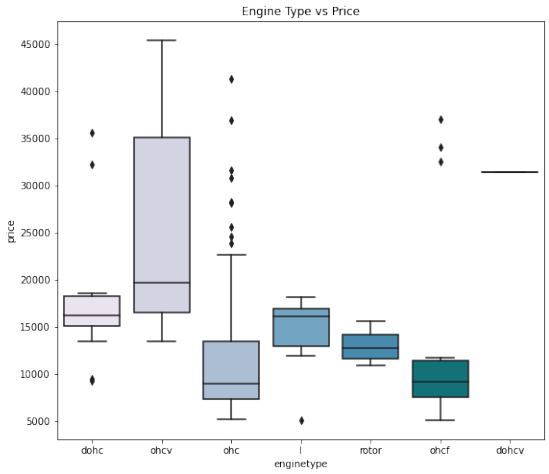


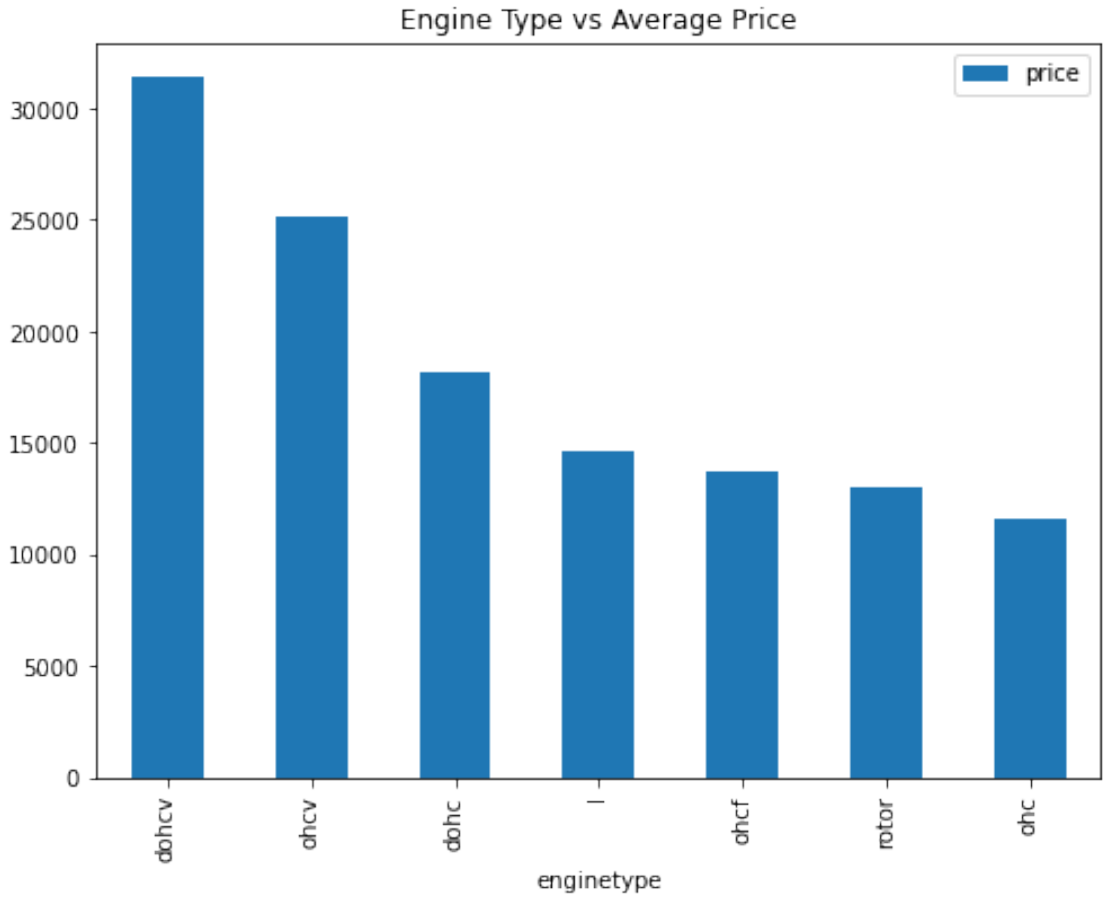
5.1.2 Inferenza :

1. Sembra che i simboli con i valori `0` e `1` abbiano un numero elevato di righe (ovvero sono i più venduti).
2. Le auto con il simbolo `-1` sembrano avere un prezzo elevato, ma sembra che il simbolo con il valore `3` abbia una fascia di prezzo simile al valore `-2`.
3. C'è un calo di prezzo al simbolo `1`.

[15]:
```
plt.figure(figsize=(20,8))

plt.subplot(1,2,1)
plt.title('Engine Type Histogram')
sns.countplot(x=cars.enginetype, palette=("Blues_d"))

plt.subplot(1,2,2)
plt.title('Engine Type vs Price')
sns.boxplot(x=cars.enginetype, y=cars.price, palette=("PuBuGn"))

plt.show()

df = pd.DataFrame(cars.groupby(['enginetype'])['price'].mean().
 ↪sort_values(ascending = False))
df.plot.bar(figsize=(8,6))
plt.title('Engine Type vs Average Price')
plt.show()
```







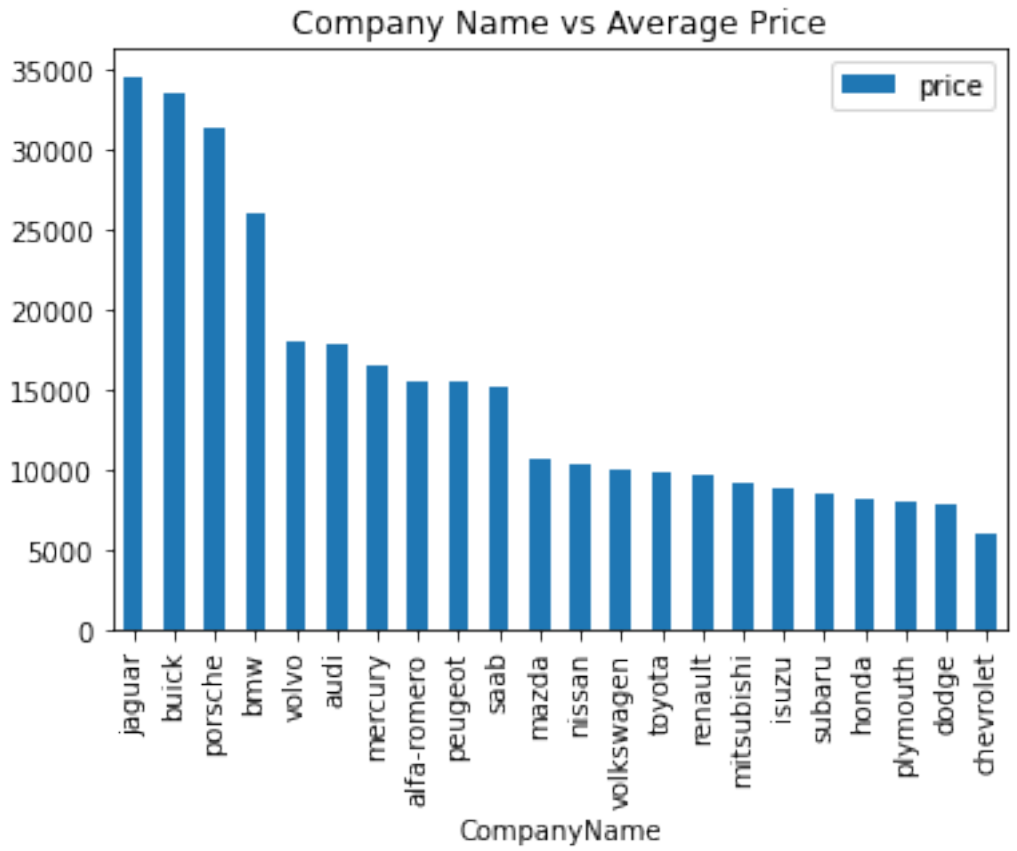
5.1.3 Inferenza :

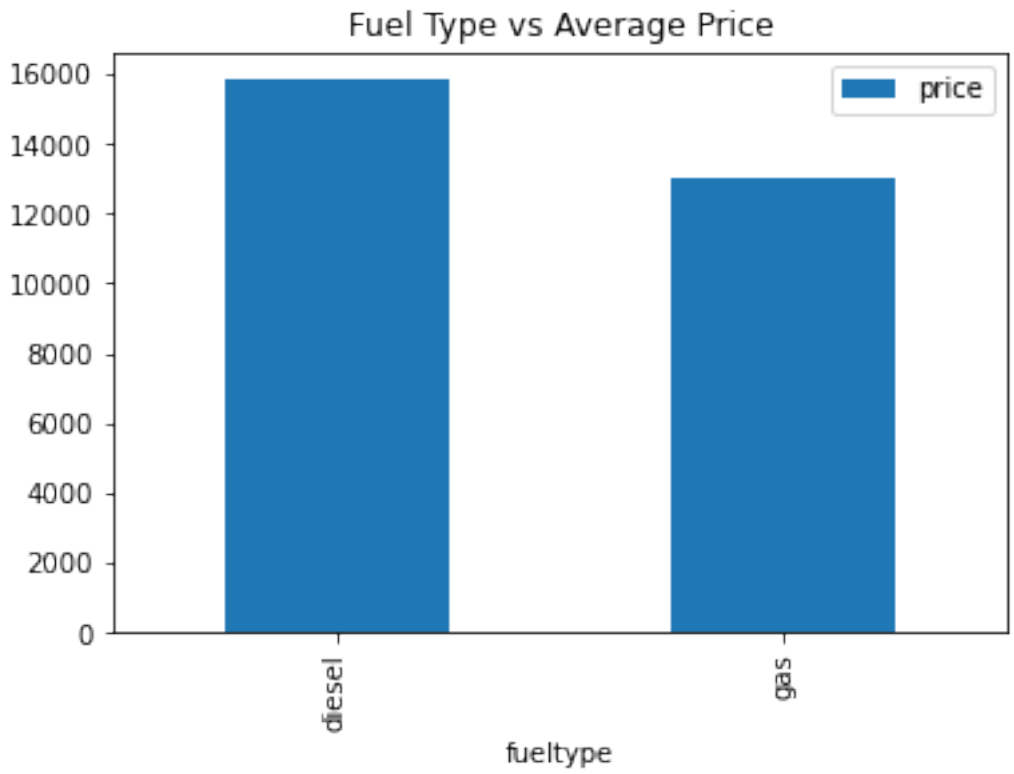
1. `ohc` sembra essere il tipo di motore preferito.
2. `ohcv` ha la fascia di prezzo più elevata (mentre `dohcv` ha solo una riga); `ohc` e `ohcf` hanno la fascia di prezzo più bassa.

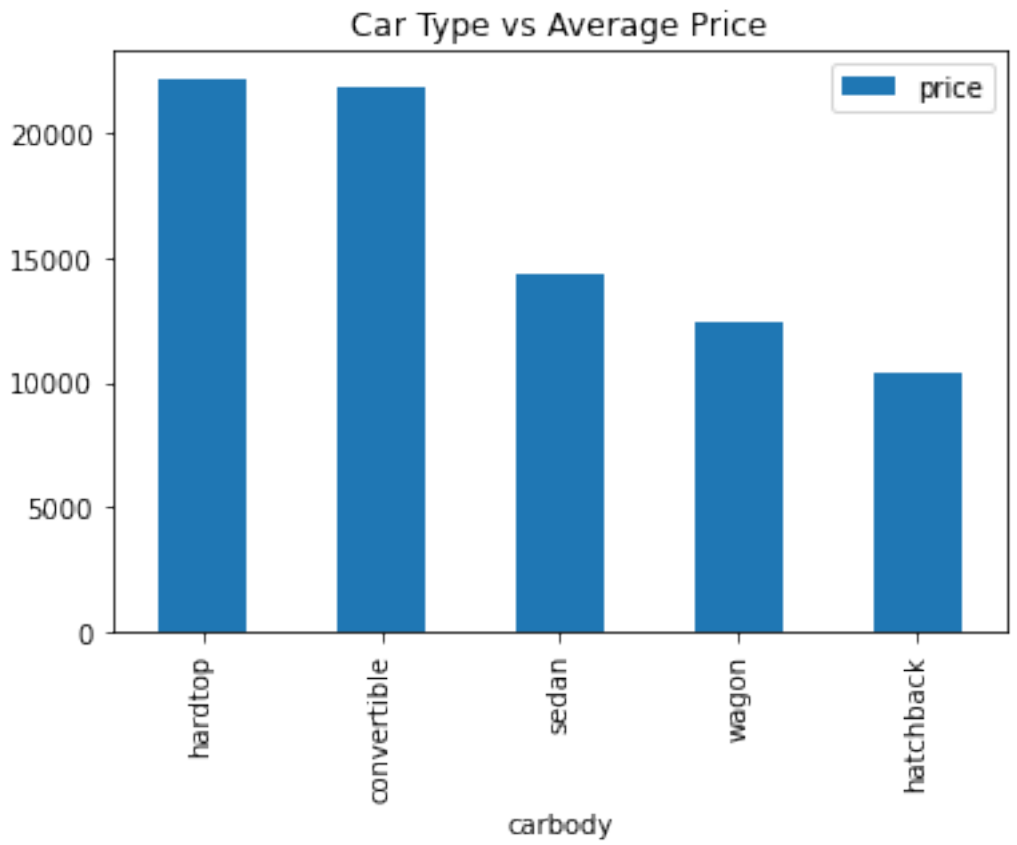
[16]:
```
plt.figure(figsize=(25, 6))

df = pd.DataFrame(cars.groupby(['CompanyName'])['price'].mean().
 ↪sort_values(ascending = False))
df.plot.bar()
plt.title('Company Name vs Average Price')
plt.show()

df = pd.DataFrame(cars.groupby(['fueltype'])['price'].mean().
 ↪sort_values(ascending = False))
df.plot.bar()
plt.title('Fuel Type vs Average Price')
```

```
plt.show()

df = pd.DataFrame(cars.groupby(['carbody'])['price'].mean().
 ↪sort_values(ascending = False))
df.plot.bar()
plt.title('Car Type vs Average Price')
plt.show()
```

```
<Figure size 1800x432 with 0 Axes>
```




Fuel Type vs Average Price
price
16000
14000
12000
10000
8000
6000
4000
2000
0
diesel
gas
fueltype

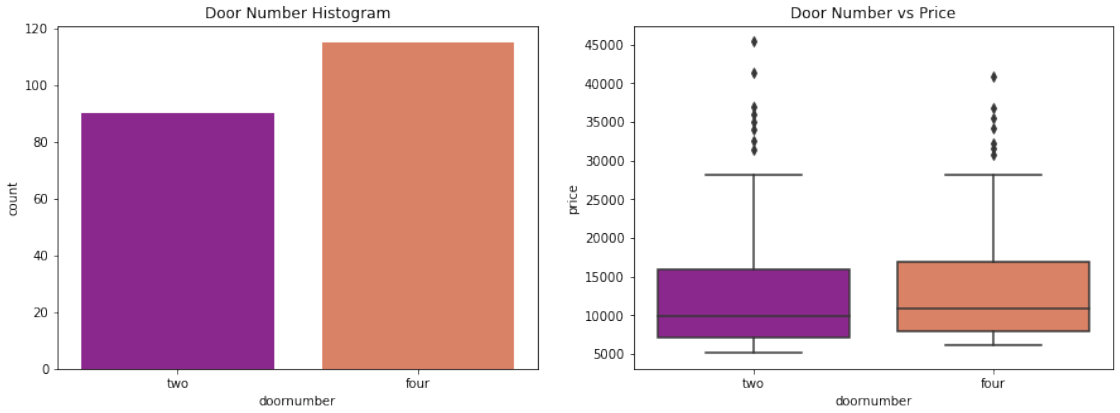


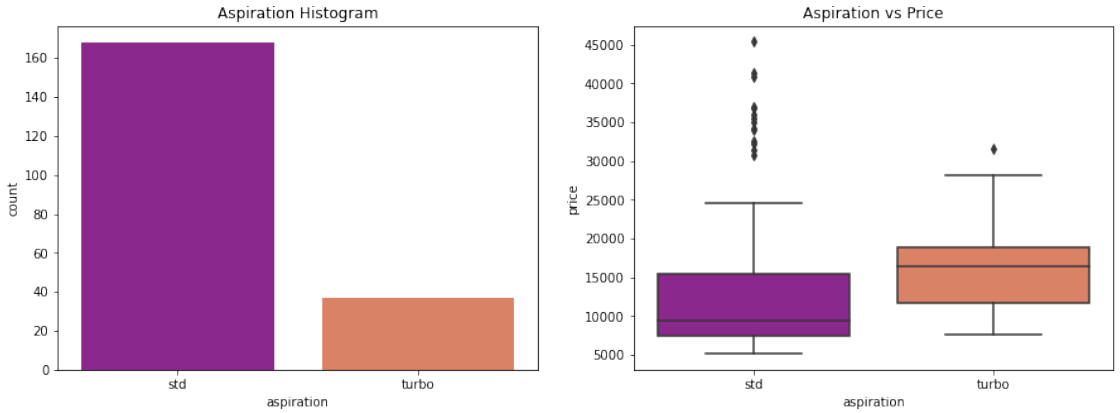
5.1.4 Inferenza :

1. `Jaguar` e `Buick` sembrano avere la media di prezzo più alta.
2. `diesel` ha la media prezzo più alta rispetto al `gas`.
3. `hardtop` e `convertible` hanno la media di prezzo più alta.

[17]:
```
plt.figure(figsize=(15,5))

plt.subplot(1,2,1)
plt.title('Door Number Histogram')
sns.countplot(x=cars.doornumber, palette=("plasma"))

plt.subplot(1,2,2)
plt.title('Door Number vs Price')
sns.boxplot(x=cars.doornumber, y=cars.price, palette=("plasma"))

plt.show()

plt.figure(figsize=(15,5))
```

```
plt.subplot(1,2,1)
plt.title('Aspiration Histogram')
sns.countplot(x=cars.aspiration, palette=("plasma"))

plt.subplot(1,2,2)
plt.title('Aspiration vs Price')
sns.boxplot(x=cars.aspiration, y=cars.price, palette=("plasma"))

plt.show()
```

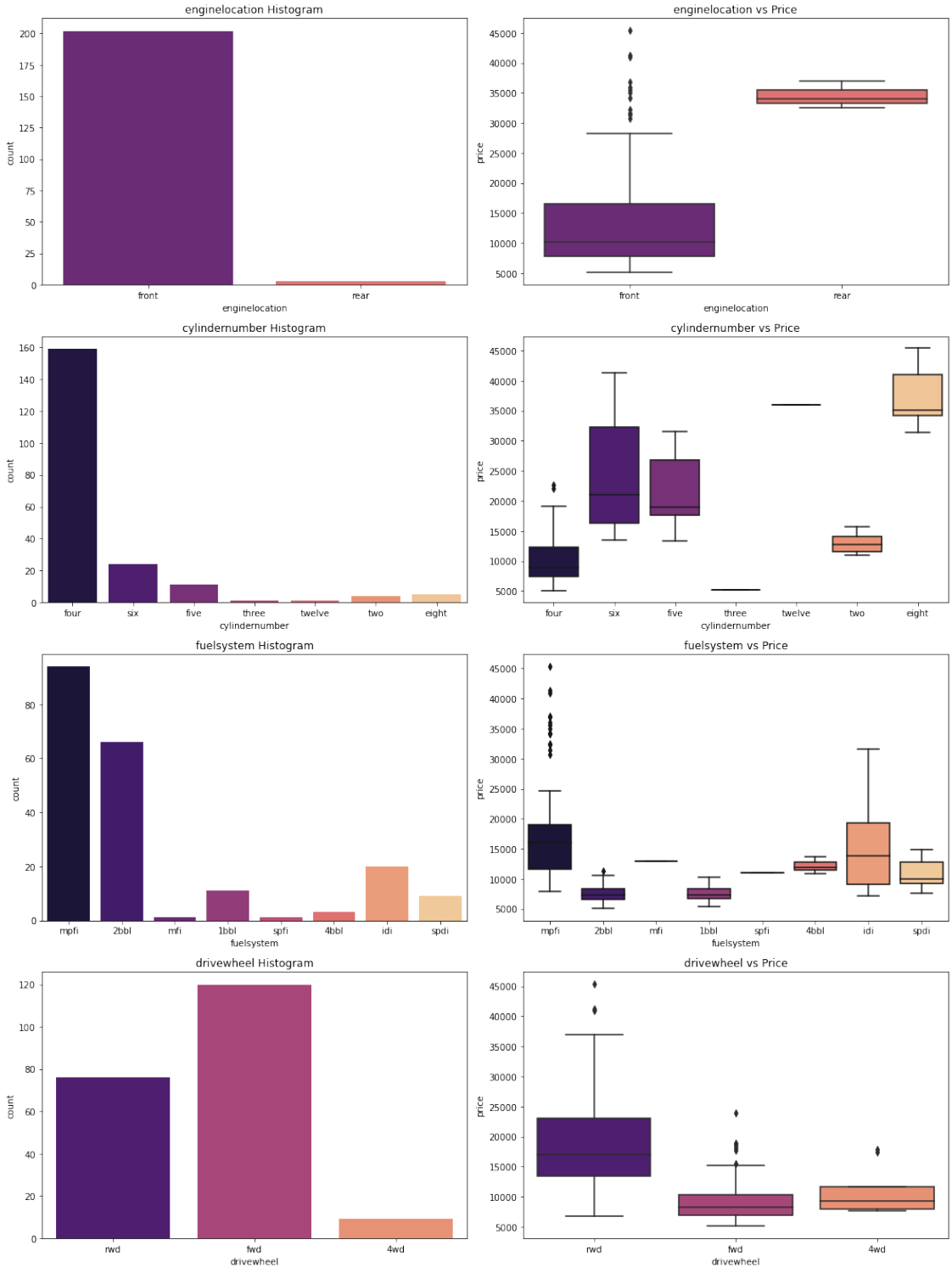




5.1.5 Inferenza :

1. `doornumber` non influenza molto il prezzo: non c'è una significativa differenza di prezzo tra le due categorie.
2. Sembra che l'aspirazione con `turbo` abbia la fascia di prezzo più elevata rispetto all'aspirazione `std` (sebbene sembra che l'aspirazione standard sia la più venduta).

[18]:
```
def plot_count(x,fig):
    plt.subplot(4,2,fig)
    plt.title(x+' Histogram')
    sns.countplot(x=cars[x],palette=("magma"))
    plt.subplot(4,2,(fig+1))
    plt.title(x+' vs Price')
    sns.boxplot(x=cars[x], y=cars.price, palette=("magma"))

plt.figure(figsize=(15,20))

plot_count('enginelocation', 1)
plot_count('cylindernumber', 3)
plot_count('fuelsystem', 5)
plot_count('drivewheel', 7)

plt.tight_layout()
```



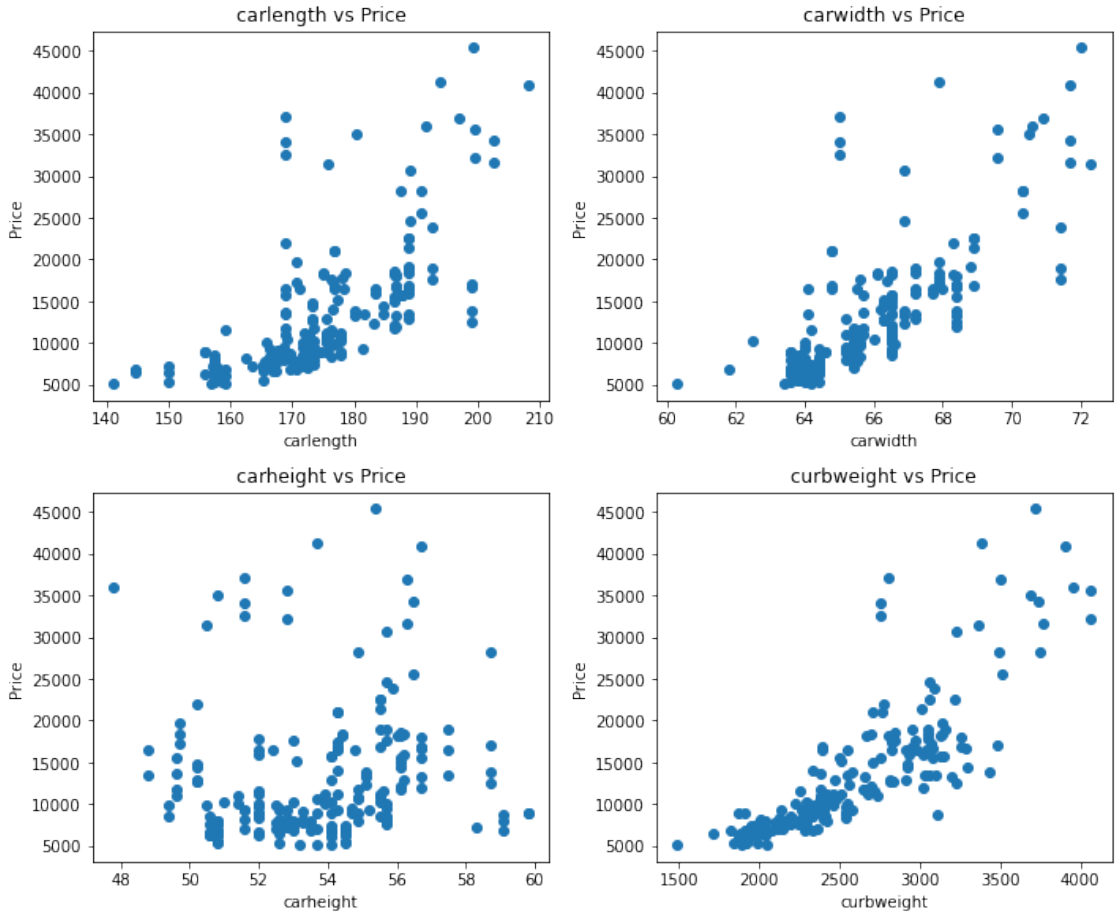
5.1.6 Inferenza :

1. Ci sono troppi pochi dati di `enginelocation` per poter ottenere un'inferenza.

2. I più comuni numeri di cilindri sono `four`, `six` e `five`. Anche se `eight` ha la fascia di prezzo più elevata.
3. `mpfi` e `2bbl` sono i tipi più comuni del sistema di alimentazione. `mpfi` e `idi` hanno la fascia di prezzo più elevata, ma ci sono troppi pochi dati delle altre categorie per derivare qualsiasi significante inferenza
4. Un'importante differenza si ha nella feature `drivewheel`. Le macchine `rwd` sembrano essere quelle con la fascia di prezzo più alta.

5.2 Step 3.2 : Visualizzazione delle feature numeriche

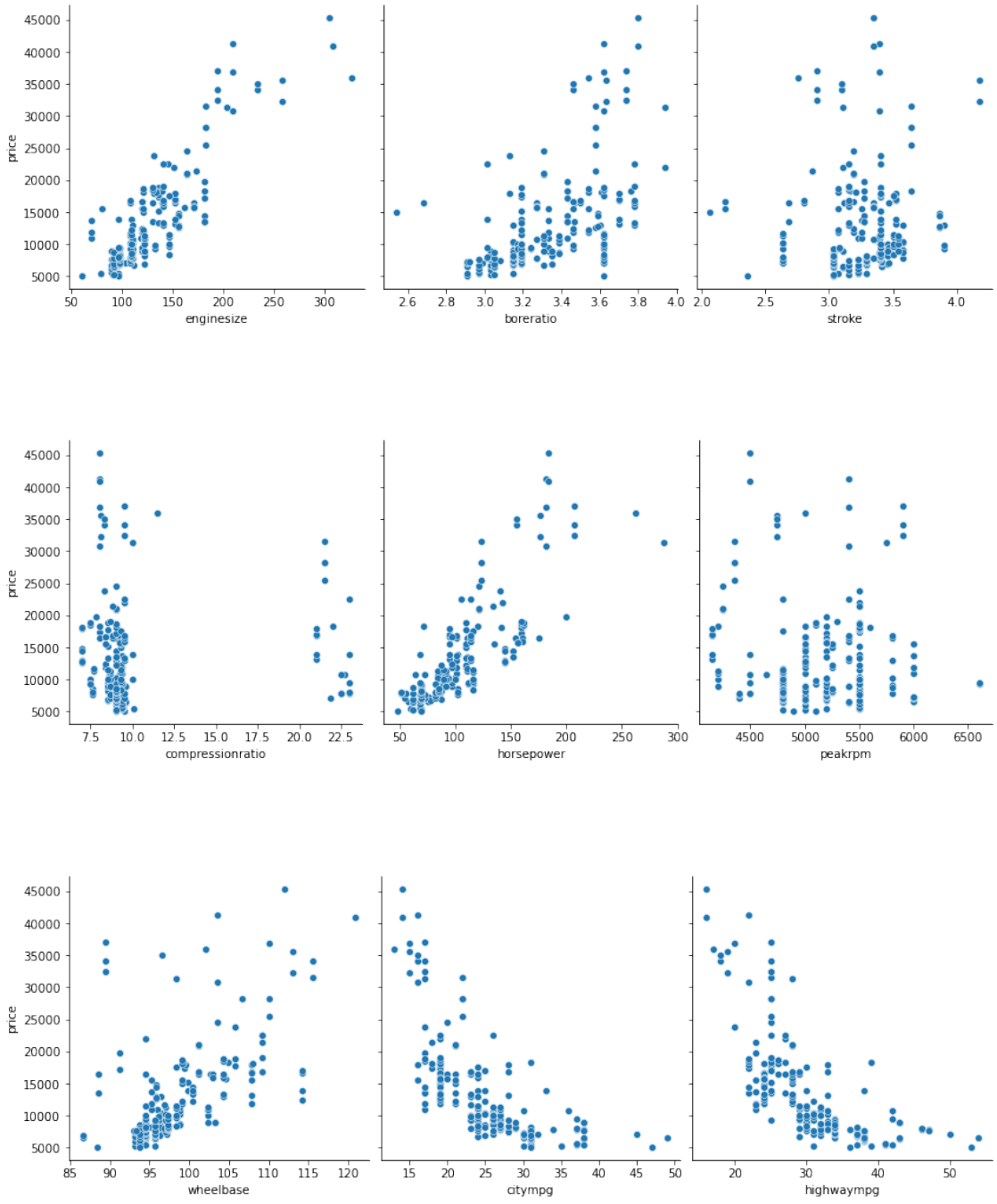
[19]:
```
def scatter(x,fig):
    plt.subplot(5,2,fig)
    plt.scatter(cars[x],cars['price'])
    plt.title(x+' vs Price')
    plt.ylabel('Price')
    plt.xlabel(x)

plt.figure(figsize=(10,20))

scatter('carlength', 1)
scatter('carwidth', 2)
scatter('carheight', 3)
scatter('curbweight', 4)

plt.tight_layout()
```



5.2.1 Inferenza :

1. `carwidth`, `carlength` e `curbweight` sembrano avere una significativa correlazione positiva con `price`.
2. `carheight` non mostra nessuna significativa correlazione con `price`.

[20]:
```
def pp(x,y,z):
    sns.pairplot(cars, x_vars=[x,y,z], y_vars='price',height=4, aspect=1,␣
 ↪kind='scatter')
    plt.show()

pp('enginesize', 'boreratio', 'stroke')
pp('compressionratio', 'horsepower', 'peakrpm')
pp('wheelbase', 'citympg', 'highwaympg')
```



5.2.2 Inferenza :

1. `enginesize`, `boreratio`, `horsepowere` `wheelbase` sembrano avere una significativa correlazione positiva con `price`.
2. `citympg` e `highwaympg` sembrano avere una significativa correlazione negativa con `price`.

6 Step 4 : Derivazione di nuove features

- Si stima che ogni macchina consumi il 55% del proprio carburante in città e la restante parte, cioè il 45%, fuori città.

[21]:
```
#Fuel economy
cars['fueleconomy'] = (0.55 * cars['citympg']) + (0.45 * cars['highwaympg'])
```

[22]:
```
#Binning the Car Companies based on avg prices of each Company.
cars['price'] = cars['price'].astype('int')
temp = cars.copy()
table = temp.groupby(['CompanyName'])['price'].mean()
temp = temp.merge(table.reset_index(), how='left',on='CompanyName')
bins = [0,10000,20000,40000]
cars_bin=['Budget','Medium','Highend']
cars['carsrange'] = pd.cut(temp['price_y'],bins,right=False,labels=cars_bin)
cars
```

[22]:

	car_ID	symboling	CompanyName	fueltype	aspiration	doornumber	\
0	1	3	alfa-romero	gas	std	two	
1	2	3	alfa-romero	gas	std	two	
2	3	1	alfa-romero	gas	std	two	
3	4	2	audi	gas	std	four	
4	5	2	audi	gas	std	four	
..	...	...	...	...	...	...	
200	201	-1	volvo	gas	std	four	
201	202	-1	volvo	gas	turbo	four	
202	203	-1	volvo	gas	std	four	
203	204	-1	volvo	diesel	turbo	four	
204	205	-1	volvo	gas	turbo	four	

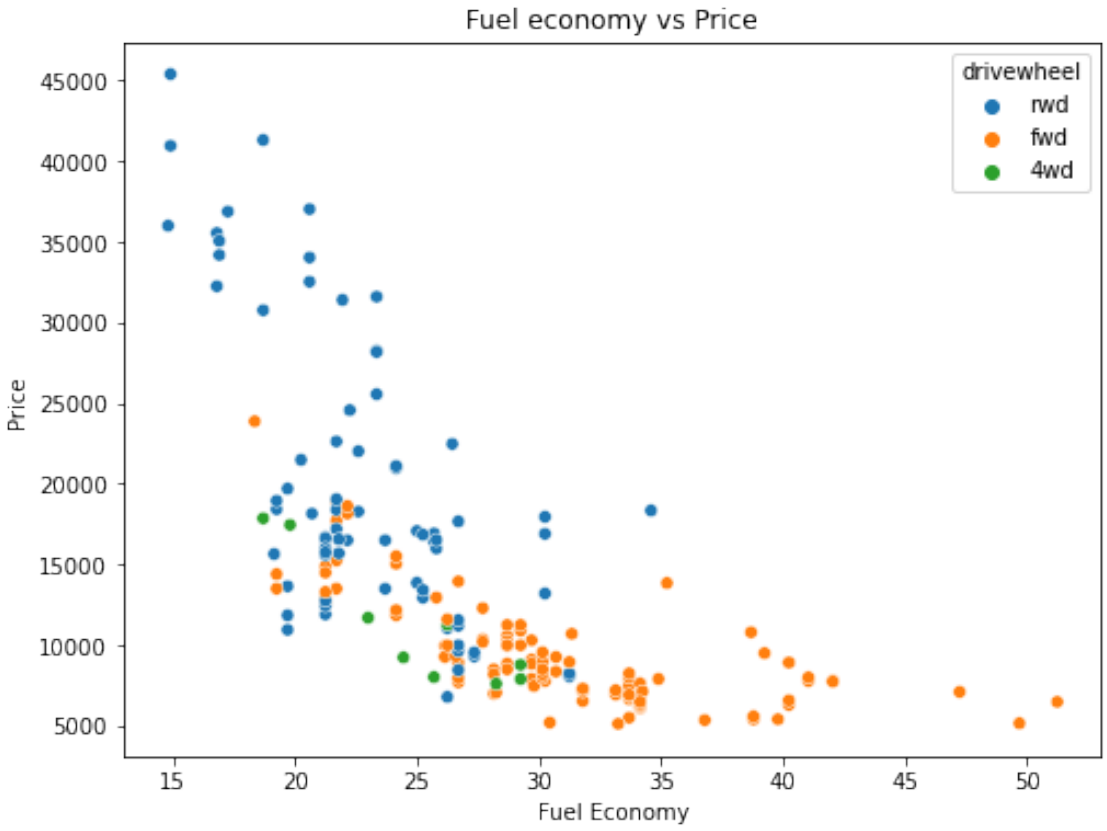
	carbody	drivewheel	enginelocation	wheelbase	...	boreratio	stroke	\
0	convertible	rwd	front	88.6	...	3.47	2.68	
1	convertible	rwd	front	88.6	...	3.47	2.68	
2	hatchback	rwd	front	94.5	...	2.68	3.47	
3	sedan	fwd	front	99.8	...	3.19	3.40	
4	sedan	4wd	front	99.4	...	3.19	3.40	
..	...	...	...	...	...	...	...	
200	sedan	rwd	front	109.1	...	3.78	3.15	
201	sedan	rwd	front	109.1	...	3.78	3.15	
202	sedan	rwd	front	109.1	...	3.58	2.87	
203	sedan	rwd	front	109.1	...	3.01	3.40	
204	sedan	rwd	front	109.1	...	3.78	3.15	

	compressionratio	horsepower	peakrpm	citympg	highwaympg	price	\
0	9.0	111	5000	21	27	13495	
1	9.0	111	5000	21	27	16500	
2	9.0	154	5000	19	26	16500	

```
3                10.0         102    5500       24          30  13950
4                 8.0         115    5500       18          22  17450
..                ...         ...     ...      ...         ...    ...
200               9.5         114    5400       23          28  16845
201               8.7         160    5300       19          25  19045
202               8.8         134    5500       18          23  21485
203              23.0         106    4800       26          27  22470
204               9.5         114    5400       19          25  22625

     fueleconomy  carsrange
0          23.70     Medium
1          23.70     Medium
2          22.15     Medium
3          26.70     Medium
4          19.80     Medium
..           ...        ...
200        25.25     Medium
201        21.70     Medium
202        20.25     Medium
203        26.45     Medium
204        21.70     Medium

[205 rows x 28 columns]
```

7 Step 5 : Analisi

[23]:
```
plt.figure(figsize=(8,6))

plt.title('Fuel economy vs Price')
sns.scatterplot(x=cars['fueleconomy'],y=cars['price'],hue=cars['drivewheel'])
plt.xlabel('Fuel Economy')
plt.ylabel('Price')

plt.show()
plt.tight_layout()
```



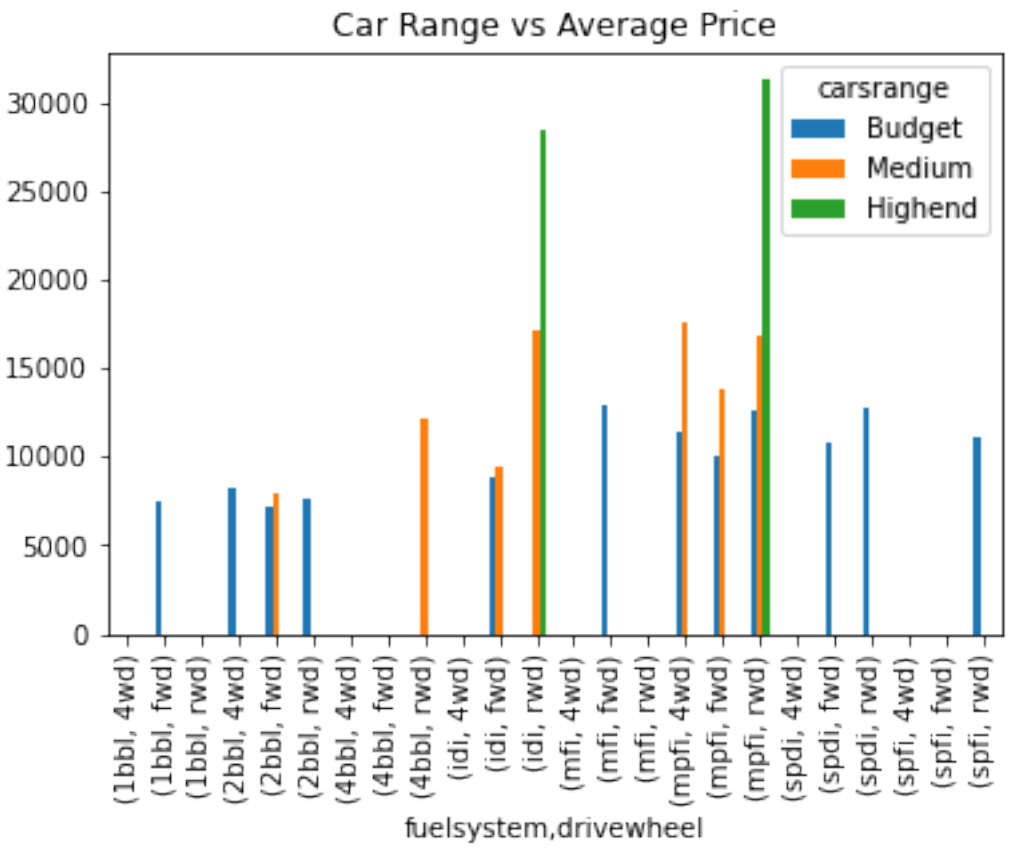

```
<Figure size 432x288 with 0 Axes>
```

7.0.1 Inferenza :

1. `fueleconomy` ha un'importante correlazione negativa con `price`.

[24]:
```
plt.figure(figsize=(25, 6))

df = pd.DataFrame(cars.groupby(['fuelsystem','drivewheel','carsrange'])['price'].
 ↪mean().unstack(fill_value=0))
df.plot.bar()
plt.title('Car Range vs Average Price')
plt.show()
```

```
<Figure size 1800x432 with 0 Axes>
```



7.0.2 Inferenza :

1. Le auto di alta gamma preferiscono la ruota motrice `rwd` con sistema di alimentazione `idi` o `mpfi`.

7.0.3 Lista delle variabili significative dopo l'analisi effettuata:

- Car Range
- Engine Type
- Fuel type
- Car Body
- Aspiration
- Cylinder Number
- Drivewheel
- Curbweight
- Car Length
- Car width
- Engine Size
- Boreratio
- Horse Power

- Wheel base
- Fuel Economy

[25]:
```
cars = cars[['price', 'fueltype', 'aspiration','carbody',␣
 ↪'drivewheel','wheelbase',
                 'curbweight', 'enginetype', 'cylindernumber', 'enginesize',␣
 ↪'boreratio','horsepower',
                  'fueleconomy', 'carlength','carwidth', 'carsrange']]
cars.head()
```

[25]:

	price	fueltype	aspiration	carbody	drivewheel	wheelbase	curbweight	\
0	13495	gas	std	convertible	rwd	88.6	2548	
1	16500	gas	std	convertible	rwd	88.6	2548	
2	16500	gas	std	hatchback	rwd	94.5	2823	
3	13950	gas	std	sedan	fwd	99.8	2337	
4	17450	gas	std	sedan	4wd	99.4	2824	

	enginetype	cylindernumber	enginesize	boreratio	horsepower	fueleconomy	\
0	dohc	four	130	3.47	111	23.70	
1	dohc	four	130	3.47	111	23.70	
2	ohcv	six	152	2.68	154	22.15	
3	ohc	four	109	3.19	102	26.70	
4	ohc	five	136	3.19	115	19.80	

	carlength	carwidth	carsrange
0	168.8	64.1	Medium
1	168.8	64.1	Medium
2	171.2	65.5	Medium
3	176.6	66.2	Medium
4	176.6	66.4	Medium

8 Step 6 : Variabili fittizie

- Si trasformano le variabili categoriche in variabili fittizie

[26]:
```
# Defining the map function
def dummies(x,df):
    temp = pd.get_dummies(df[x], drop_first = True)
    df = pd.concat([df, temp], axis = 1)
    df.drop([x], axis = 1, inplace = True)
    return df
# Applying the function to the cars_lr

cars = dummies('fueltype',cars)
cars = dummies('aspiration',cars)
cars = dummies('carbody',cars)
cars = dummies('drivewheel',cars)
```

```
cars = dummies('enginetype',cars)
cars = dummies('cylindernumber',cars)
cars = dummies('carsrange',cars)
```

[27]:
```
cars.head()
```

[27]:

	price	wheelbase	curbweight	enginesize	boreratio	horsepower	\
0	13495	88.6	2548	130	3.47	111	
1	16500	88.6	2548	130	3.47	111	
2	16500	94.5	2823	152	2.68	154	
3	13950	99.8	2337	109	3.19	102	
4	17450	99.4	2824	136	3.19	115	

	fueleconomy	carlength	carwidth	gas	...	ohcv	rotor	five	four	six	\
0	23.70	168.8	64.1	1	...	0	0	0	1	0	
1	23.70	168.8	64.1	1	...	0	0	0	1	0	
2	22.15	171.2	65.5	1	...	1	0	0	0	1	
3	26.70	176.6	66.2	1	...	0	0	0	1	0	
4	19.80	176.6	66.4	1	...	0	0	1	0	0	

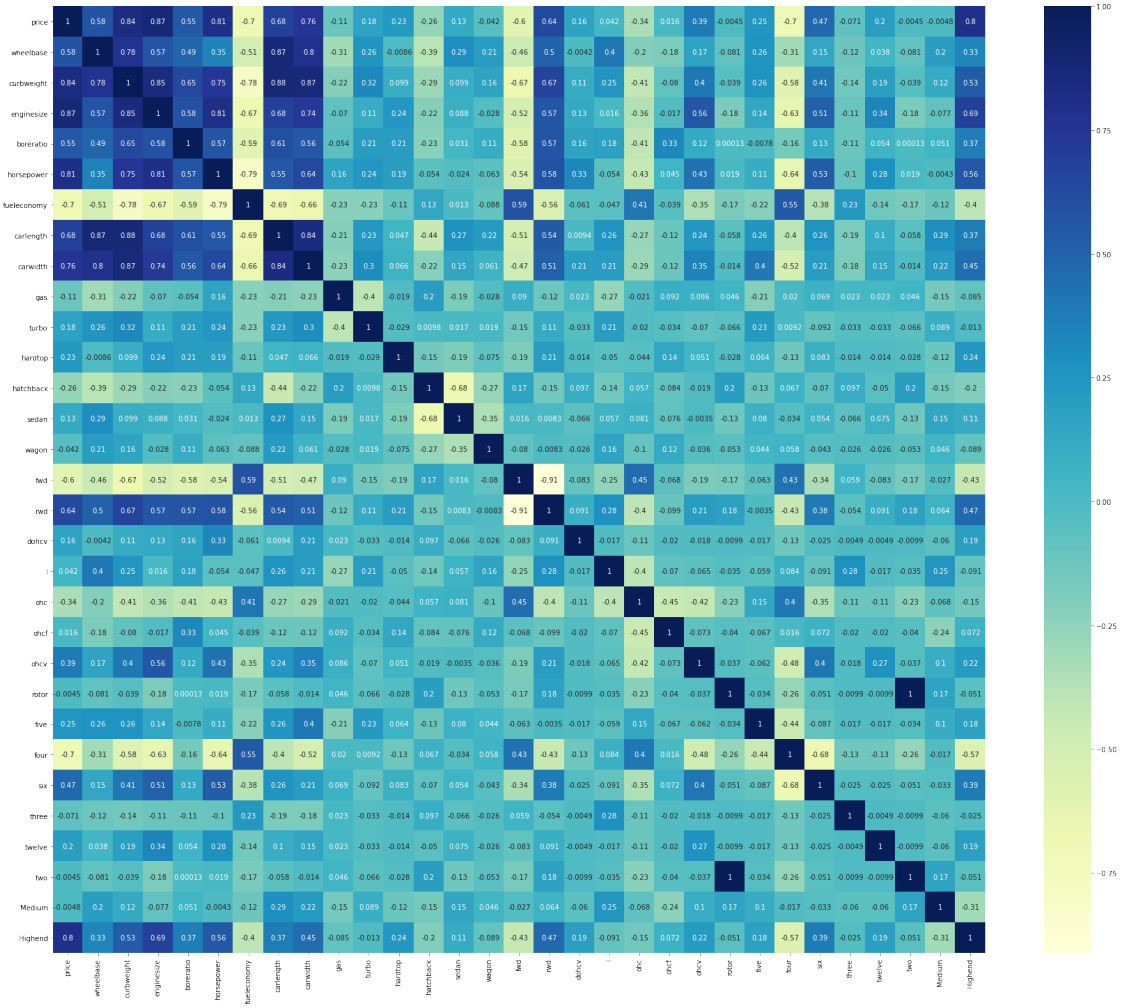
	three	twelve	two	Medium	Highend
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	1	0

[5 rows x 31 columns]

[28]:
```
cars.shape
```

[28]:
```
(205, 31)
```

[29]:
```
#Correlation using heatmap
plt.figure(figsize = (30, 25))
sns.heatmap(cars.corr(), annot = True, cmap="YlGnBu")
plt.show()
```

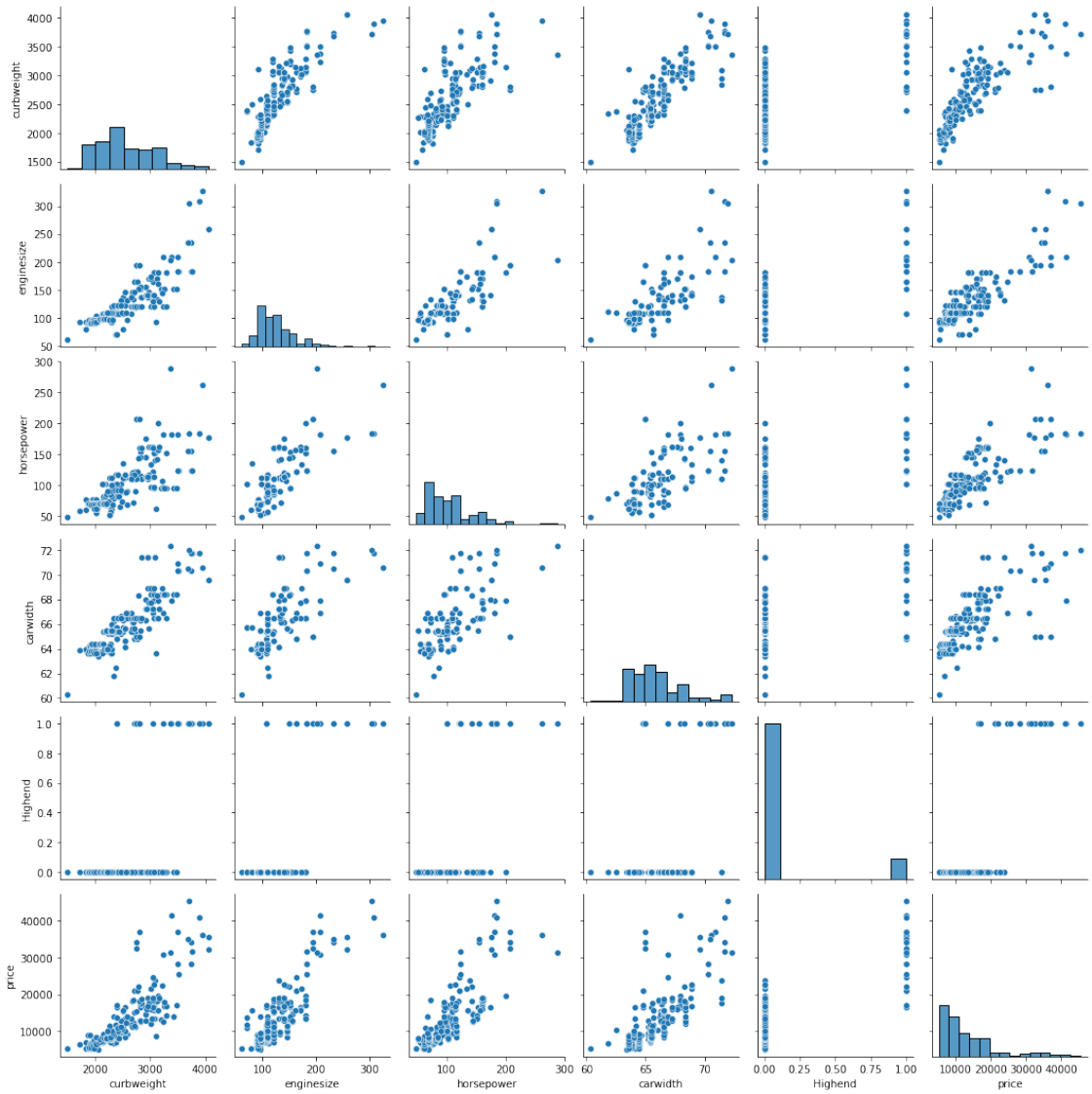


8.0.1 Inferenza

1. Le variabili più correlate al prezzo sono: `curbweight`, `enginesize`, `horsepower`, `carwidth` e `Highend`.

[30]:
```
vars = ["curbweight", "enginesize", "horsepower", "carwidth", "Highend", "price"]
```

[31]:
```
sns.pairplot(cars[vars])
plt.show()
```



9 Step 7 : Train-Test Split e standardizzazione

[32]:
```
from sklearn.model_selection import train_test_split

X = cars[vars].drop("price", axis = 1).values
Y = cars["price"].values

np.random.seed(0)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.3,␣
 ↪random_state = 100)
```

9.1 Step 7.1: standardizzazione

- Si standardizzano le variabili numeriche

[33]:
```
from sklearn.preprocessing import StandardScaler

ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

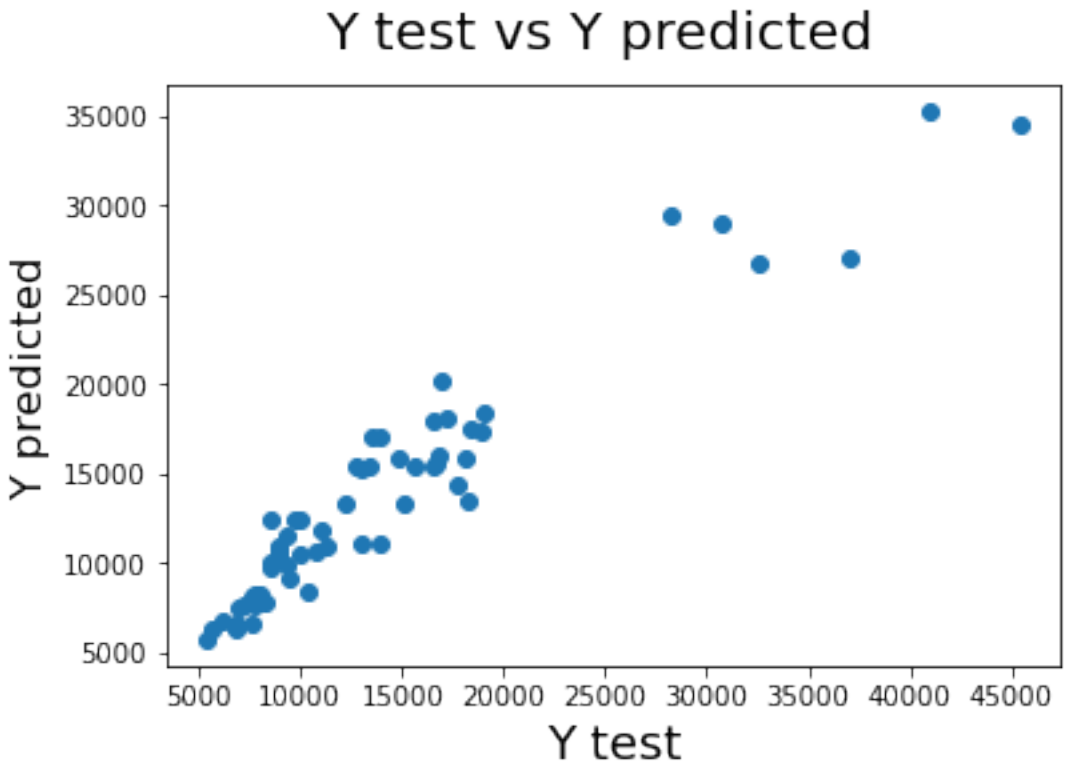
10 Step 8 : Ricerca del modello di predizione più adatto

[34]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score, mean_squared_log_error
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.preprocessing import PolynomialFeatures

#Test the model in input
def try_model(model,parameters, X_train, Y_train, X_test, Y_test):
    mod = GridSearchCV(model, parameters, cv=None)
    mod.fit(X_train, Y_train)
    Y_pred_test = mod.predict(X_test)
    Y_pred_train = mod.predict(X_train)


    print("\nTrain Metrics: ")
    print("Mean squared log error train: ", mean_squared_log_error(Y_train,␣
 ↪Y_pred_train))
    print("R2 score pred train: ", r2_score(Y_train, Y_pred_train))
    print("\nTest Metrics: ")
    print("Mean squared log error test: ", mean_squared_log_error(Y_test,␣
 ↪Y_pred_test))
    print("R2 score pred test: ", r2_score(Y_test, Y_pred_test))

    # EVALUATION OF THE MODEL
    # Plotting y_test and y_pred to understand the spread.
    fig = plt.figure()
    plt.scatter(Y_test, Y_pred_test)
    fig.suptitle('Y test vs Y predicted', fontsize=20)  # Plot heading
    plt.xlabel('Y test', fontsize=18)  # X-label
    plt.ylabel('Y predicted', fontsize=16)
    plt.show()
```

10.1 REGRESSIONE LINEARE

[35]:
```
parameters = {'fit_intercept': [True, False], 'normalize': [True, False],␣
 ↪'copy_X': [True, False]}
try_model(LinearRegression(), parameters, X_train, Y_train, X_test, Y_test)
```

```
Train Metrics:
Mean squared log error train:  0.03526899049237338
R2 score pred train:  0.9102931286100621

Test Metrics:
Mean squared log error test:  0.022349529792874162
R2 score pred test:  0.8926447826156122
```

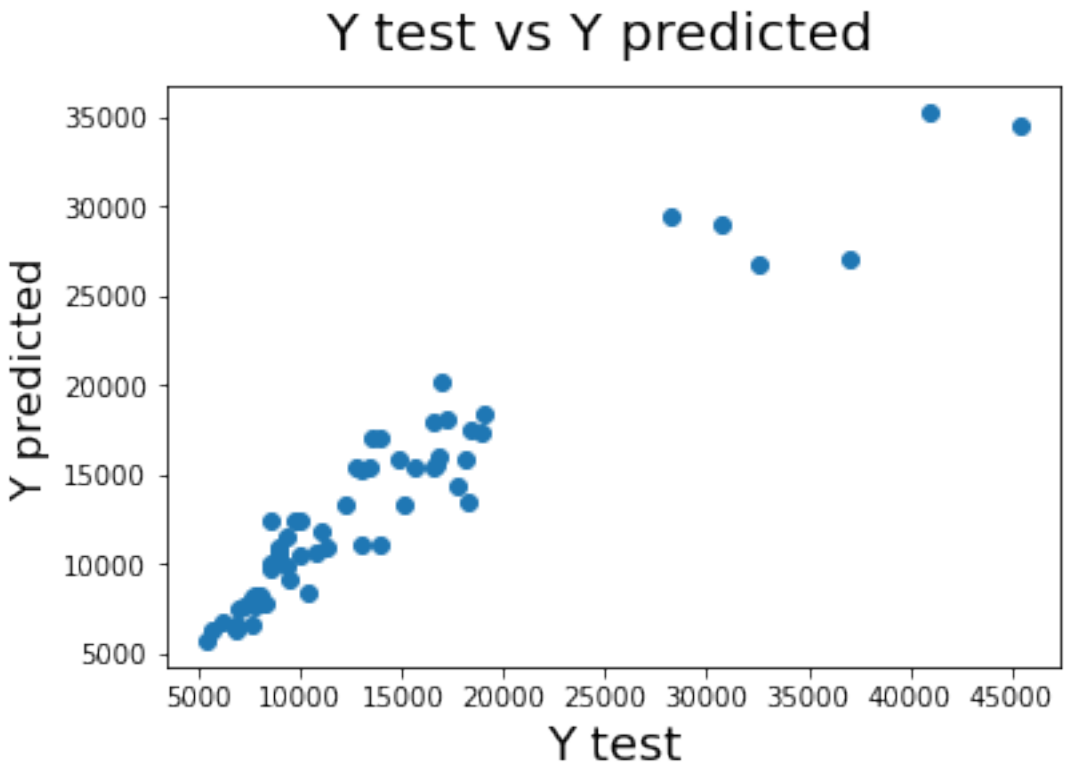


10.2 MODELLO LASSO

[36]:
```
parameters = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1., 10.], 'fit_intercept':␣
 ↪[True, False], 'normalize': [True, False],
              'copy_X': [True, False],
              'precompute': [True, False], 'max_iter': [i for i in range(1000,␣
 ↪10000, 500)],
```

```
              'warm_start': [True, False],
              'positive': [True, False], 'selection': ['cyclic', 'random']}
try_model(Lasso(), parameters, X_train, Y_train, X_test, Y_test)
```

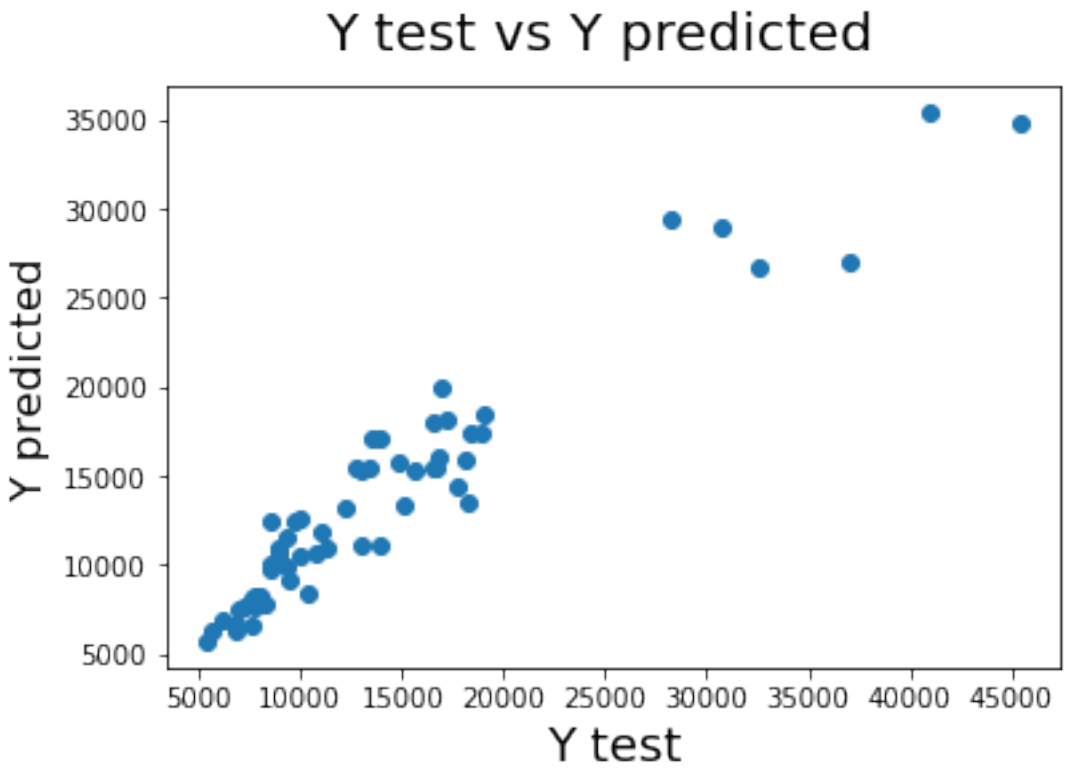
```
Train Metrics:
Mean squared log error train:  0.03504887470678584
R2 score pred train:  0.9102907477342123

Test Metrics:
Mean squared log error test:  0.022395610977003638
R2 score pred test:  0.8922727990566587
```



10.3 MODELLO RIDGE

[37]:
```
parameters = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1., 10.], 'fit_intercept':␣
 ↪[True, False], 'normalize': [True, False],
              'copy_X': [True, False],
              'max_iter': [i for i in range(1000, 10000, 500)],
              'solver': ['auto', 'svd', 'cholesky', 'lsqr', 'sparse_cg', 'sag',␣
 ↪'saga']}
try_model(Ridge(), parameters, X_train, Y_train, X_test, Y_test)
```

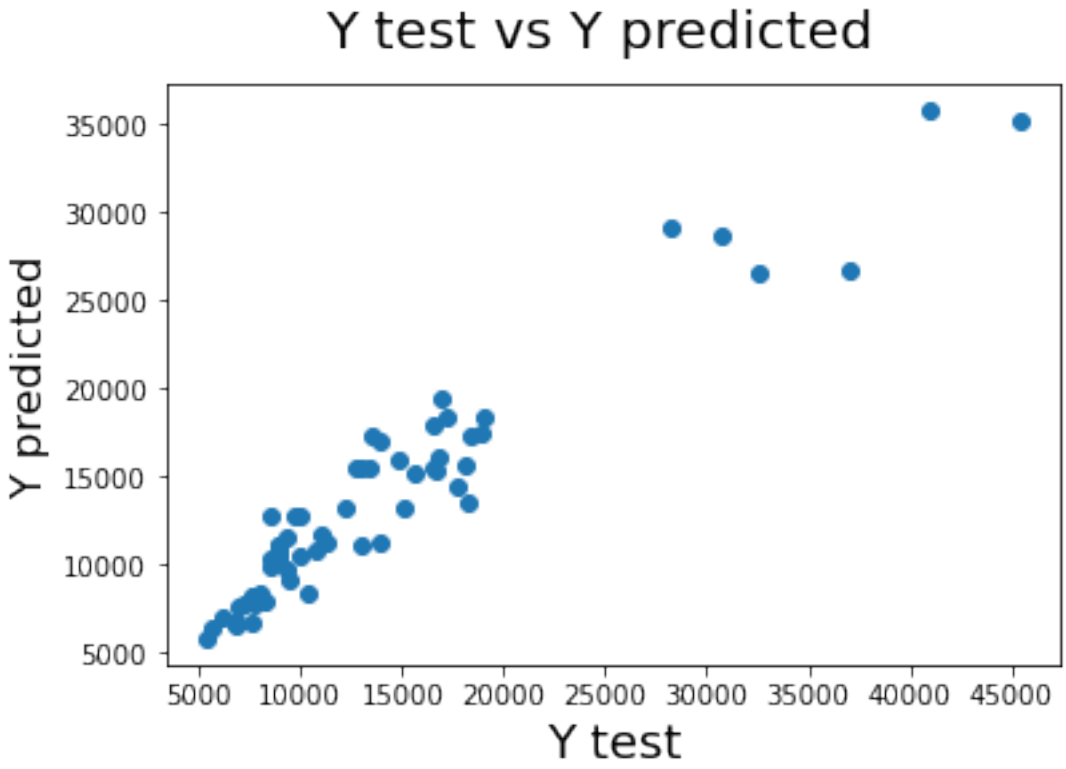
```
Train Metrics:
Mean squared log error train:  0.03549177891552918
R2 score pred train:  0.9102576926309367

Test Metrics:
Mean squared log error test:  0.022591263000712333
R2 score pred test:  0.8930200192604221
```



10.4 MODELLO ELASTICNET

[38]:
```
parameters = {'alpha': [0.0001, 0.001, 0.01, 0.1, 1., 10.], 'l1_ratio': [0.2, 0.
 ↪4, 0.6, 0.8, 1.0],
              'fit_intercept': [True, False], 'normalize': [True, False],
              'copy_X': [True, False],
              'precompute': [True, False],
              'warm_start': [True, False],
              'max_iter': [i for i in range(1000, 10000, 500)],
              'selection': ['cyclic', 'random']}
try_model(ElasticNet(), parameters, X_train, Y_train, X_test, Y_test)
```

```
Train Metrics:
Mean squared log error train:  0.035941623637315456
R2 score pred train:  0.9095775386979942

Test Metrics:
Mean squared log error test:  0.02360185060543485
R2 score pred test:  0.8926655295167302
```

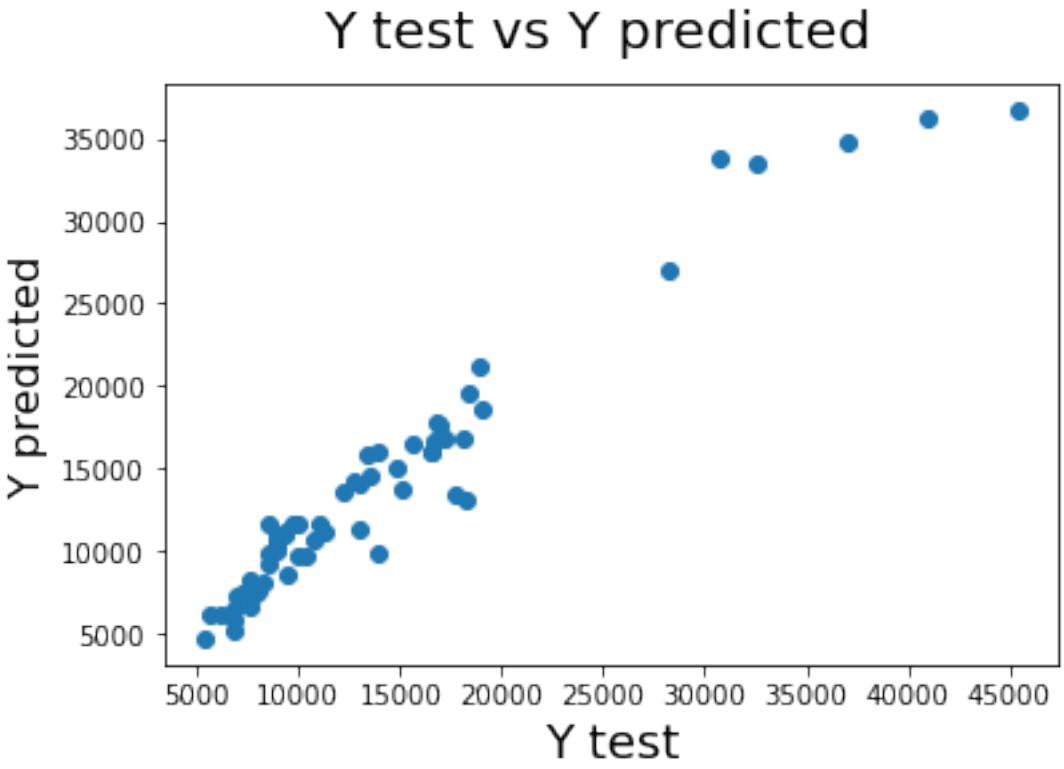


10.5 REGRESSIONE POLINOMIALE DI GRADO 2

[39]:
```
parameters = {'fit_intercept': [True, False], 'normalize': [True, False],␣
 ↪'copy_X': [True, False]}
polyfeats = PolynomialFeatures(degree=2)
X_train_poly = polyfeats.fit_transform(X_train)
X_test_poly = polyfeats.transform(X_test)
try_model(LinearRegression(), parameters, X_train_poly, Y_train, X_test_poly,␣
 ↪Y_test)
```

```
Train Metrics:
Mean squared log error train:  0.02221263994929535
R2 score pred train:  0.9442367326282824
```

```
Test Metrics:
Mean squared log error test:  0.017551984459917416
R2 score pred test:  0.9409112621374752
```



10.5.1 Inferenza:

1. Si evince che il modello più adatto è la regressione polinomiale di grado 2.
2. Non è presente overfitting.
3. R-squared della regressione polinomiale è di 0.9409 e il Mean Squared Log Error è di 0.017.

11 KNOWLEDGE BASE

11.1 Introduzione

1. Si è creato un algoritmo che legge i dati delle auto presenti nel dataset in formato csv.
2. Si sono estratte le features di interesse, le quali sono state inserite all'interno della knowledge base.

- Si è implementata una knowledge base nella quale sono stati inseriti:
  - tutti i nomi delle aziende automobilistiche;
  - i sistemi di alimentazione;
  - il tipo di alimentazione di ogni modello di auto;

- tutti i modelli di ogni azienda automibilistica;
- la tipologia di macchina per ogni modello;
- la fascia di prezzo di ogni azienda automobilistica.

[40]:
```
from pyswip import Prolog

prolog = Prolog()
prolog.consult("data/knowledge_base.pl")
```

11.2 Funzioni di base della Knowledge Base

11.2.1 Funzioni di gestione di clausole e query

- Si sono create funzioni di aggiunta e cancellazione di clausole e di interrogazione della knoweledge base in Prolog.

[41]:
```
def addAssert(prolog, str):
    prolog.assertz(str)


def deleteAssert(prolog, str):
    prolog.retract(str)


def query(prolog, str):
    qr = (str + ".")
    return list(prolog.query(qr))
```

11.2.2 Funzione di ricerca di modelli di auto in base al carburante

[42]:
```
def fuelSearch():
    a = input("Quale tipologia di carburante ti interessa? ")
    a = a.lower()
    print(query(prolog, "fuelCar(X,\"" + a + "\")"))
```

11.2.3 Funzione di ricerca di modelli di auto in base alla società automobilistica

[43]:
```
def modelSearch():
    a = input("Di quale marca vuoi visualizzare i modelli disponibili? ")
    a = a.lower()
    print(query(prolog, "companyModel(\"" + a + "\", Y)"))
```

11.2.4 Funzione di ricerca della società automobilistica di un dato modello di auto

[44]:
```
def companySearch():
    a = input("Di quale modello vuoi sapere la marca? ")
    a = a.lower()
    print(query(prolog, "companyModel(X,\"" + a + "\")"))
```

11.2.5 Funzione di ricerca di società automobilistiche di una data fascia di prezzo

[45]:
```
def budgetSearch():
    a = input("Quale fascia di prezzo di marca ti interessa (bassa/media/alta)?␣
 ↪")
    a = a.lower()
    if a == "bassa":
        a = "budget"
    elif a == "media":
        a = "medium"
    elif a == "alta":
        a = "highend"
    else:
        print("Scelta errata")
    print(query(prolog, "carsrange(X,\"" + a + "\")"))
```

11.2.6 Funzione di riceca del modello in base alla tipologia di auto

[46]:
```
def modelBodySearch():
    a = input("Quale tipologia di auto ti interessa (hatchback, sedan, wagon,␣
 ↪hardtop, convertible)? ")
    a = a.lower()
    print(query(prolog, "modelBody(X,\"" + a + "\")"))
```

11.2.7 Interfaccia per l'interazione con la Knowledge Base

[47]:
```
print("########## BENVENUTO ##########")
answer = input("Area di interesse:\n"
               "1) CARBURANTE \n"
               "2) MARCA->MODELLO \n"
               "3) MODELLO->MARCA \n"
               "4) TIPOLOGIA\n"
               "5) BUDGET\n"
               "X) USCITA\n"
               "Quale area di interesse visualizzare?: ")
while answer[0] != ("x") and answer[0] != ("X"):
    if answer[0] == "1":
        fuelSearch()
    elif answer[0] == "2":
        modelSearch()
    elif answer[0] == "3":
        companySearch()
    elif answer[0] == "4":
        modelBodySearch()
    elif answer[0] == "5":
        budgetSearch()
    else:
```

```
        print("RISPOSTA ERRATA!")
    answer = input("Quale area di interesse visualizzare?: ")
```

```
########## BENVENUTO ##########
Area di interesse:
1) CARBURANTE
2) MARCA->MODELLO
3) MODELLO->MARCA
4) TIPOLOGIA
5) BUDGET
X) USCITA
Quale area di interesse visualizzare?: 5
Quale fascia di prezzo di marca ti interessa (bassa/media/alta)? media
[{'X': b'alfa-romero'}, {'X': b'audi'}, {'X': b'mazda'}, {'X': b'mercury'},
{'X': b'nissan'}, {'X': b'peugeot'}, {'X': b'saab'}, {'X': b'volkswagen'}, {'X':
b'volvo'}]
```

- 'X' è la feature ricercata

12 PROGETTAZIONE E SVILUPPO

12.1 Sviluppi futuri

In futuro, il progetto da noi sviluppato potrà essere utilizzato in larga scala da più aziende automobilistiche per la predizione dei prezzi di vendita in diversi mercati del mondo, ma anche in diversi contesti d'uso: un esempio consiste nel riaddattare il software per la stima di prezzi di auto usate. Per far ciò sarà necessario utilizzare un dataset. Per quanto riguarda la base di conoscenza, ciò che è stato inserito nel nostro progetto, è una piccola demo a dimostrazione delle grandi potenzialità che lo strumento implementato può offrire: ad oggi è utilizzato per visualizzare informazioni sulle auto ma in uno sviluppo futuro potrebbe essere utilizzato per la diagnostica di malfunzionamenti delle auto usate che potrebbe, quindi, influenzare il costo di vendita.