In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression , Ridge , Lasso
from sklearn.svm import SVR

from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
import warnings
warnings.filterwarnings("ignore")
```

In [2]:

```
df = pd.read_csv("cars.csv")
df.head()
```

Out[2]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type
0	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc
1	3	?	alfa-romero	gas	convertible	rwd	front	64.1	48.8	dohc
2	1	?	alfa-romero	gas	hatchback	rwd	front	65.5	52.4	ohcv
3	2	164	audi	gas	sedan	fwd	front	66.2	54.3	ohc
4	2	164	audi	gas	sedan	4wd	front	66.4	54.3	ohc

In [3]:

```
#Checking the number of nan values
```

In [4]:

```
df.isna().sum()
```

Out[4]:

```
symboling            0
normalized-losses    0
make                 0
fuel-type            0
body-style           0
drive-wheels         0
engine-location      0
width                0
height               0
engine-type          0
engine-size          0
horsepower           0
city-mpg             0
highway-mpg          0
price                0
dtype: int64
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   body-style         205 non-null    object 
 5   drive-wheels       205 non-null    object 
 6   engine-location    205 non-null    object 
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   engine-type        205 non-null    object 
 10  engine-size        205 non-null    int64  
 11  horsepower         205 non-null    object 
 12  city-mpg           205 non-null    int64  
 13  highway-mpg        205 non-null    int64  
 14  price              205 non-null    int64  
dtypes: float64(2), int64(5), object(8)
memory usage: 24.1+ KB
```

In [6]:

```
df["normalized-losses"].value_counts()
```

Out[6]:

```
?      41
161    11
91      8
150     7
104     6
128     6
134     6
85      5
94      5
65      5
168     5
102     5
74      5
103     5
95      5
106     4
118     4
148     4
93      4
122     4
115     3
125     3
154     3
101     3
83      3
137     3
192     2
129     2
89      2
153     2
81      2
87      2
197     2
194     2
158     2
108     2
145     2
164     2
113     2
119     2
110     2
188     2
256     1
107     1
98      1
142     1
78      1
77      1
90      1
121     1
231     1
186     1
Name: normalized-losses, dtype: int64
```

In [7]:

```
df["horsepower"].value_counts()
```

Out[7]:

```
68     19
70     11
69     10
116     9
110     8
95      7
88      6
160     6
101     6
114     6
62      6
82      5
76      5
97      5
102     5
84      5
145     5
111     4
86      4
92      4
123     4
182     3
121     3
85      3
207     3
152     3
73      3
90      3
176     2
52      2
100     2
184     2
56      2
?       2
162     2
94      2
161     2
112     2
156     2
155     2
72      1
48      1
200     1
106     1
60      1
78      1
115     1
288     1
120     1
142     1
143     1
140     1
175     1
```

```
134     1
55      1
135     1
64      1
262     1
154     1
58      1
Name: horsepower, dtype: int64
```

Handling the Nan values

In [8]:

```
#Replacing the values with np.nan
df["normalized-losses"].replace("?", np.nan, inplace=True)
df["horsepower"].replace("?", np.nan, inplace=True)

#Changing the datatype
df["normalized-losses"] = df["normalized-losses"].astype("float")
df["horsepower"] = df["horsepower"].astype("float")

#Getting the mean value
nmean = df["normalized-losses"].mean()
hmean = df["horsepower"].mean()

#Filling the missing values with mean values
df["normalized-losses"].fillna(nmean , inplace=True)
df["horsepower"].fillna(hmean , inplace=True)
```

In [9]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          205 non-null    int64  
 1   normalized-losses  205 non-null    float64
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   body-style         205 non-null    object 
 5   drive-wheels       205 non-null    object 
 6   engine-location    205 non-null    object 
 7   width              205 non-null    float64
 8   height             205 non-null    float64
 9   engine-type        205 non-null    object 
 10  engine-size        205 non-null    int64  
 11  horsepower         205 non-null    float64
 12  city-mpg           205 non-null    int64  
 13  highway-mpg        205 non-null    int64  
 14  price              205 non-null    int64  
dtypes: float64(4), int64(5), object(6)
memory usage: 24.1+ KB
```

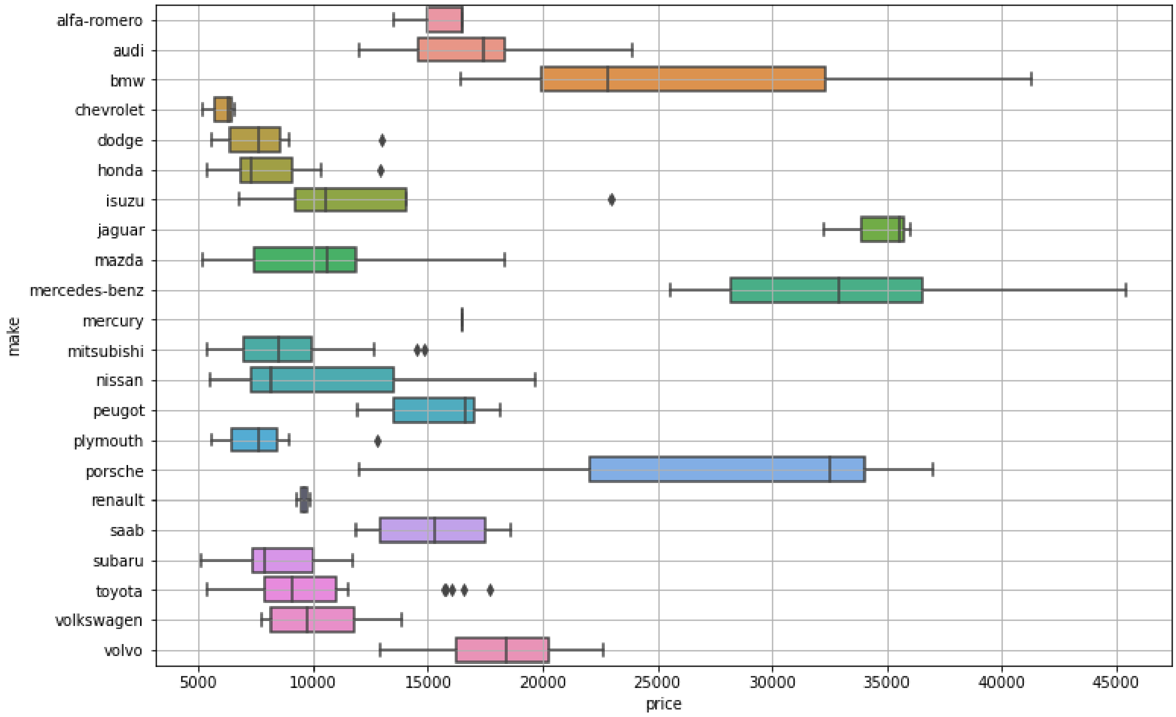
- It is observed that the datatype of columns containing numeric values now have proper datatype.
- Also the operation of handling nan values is performed succesfully.

Outliers

- Extracting the outliers with the help of boxplot.

In [10]:

```
plt.figure(figsize=(12,8))
sns.boxplot(data=df , x="price" , y="make")
plt.grid(True)
plt.show()
```



Droping the rows containing outliers and cleaning the data.

In [11]:

```
df[(df["make"]=="dodge") & (df["price"]>11000)]
```

Out[11]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type
29	3	145.0	dodge	gas	hatchback	fwd	front	66.3	50.2	ohc

In [12]:

```
df.drop(29, inplace=True)
```

In [13]:

```
df[(df["make"]=="honda") & (df["price"]>11000)]
```

Out[13]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engi s
41	0	85.0	honda	gas	sedan	fwd	front	65.2	54.1	ohc	

In [14]:

```
df.drop(41, inplace=True)
```

In [15]:

```
df[(df["make"]=="isuzu") & (df["price"]>15000)]
```

Out[15]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type	engi s
45	0	122.0	isuzu	gas	sedan	fwd	front	63.6	52.0	ohc	

In [16]:

```
df.drop(45, inplace=True)
```

In [17]:

```
df[(df["make"]=="mitsubishi") & (df["price"]>13000)]
```

Out[17]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine typ
83	3	122.0	mitsubishi	gas	hatchback	fwd	front	66.3	50.2	oh
84	3	122.0	mitsubishi	gas	hatchback	fwd	front	66.3	50.2	oh

In [18]:

```
df.drop([83,84], inplace=True)
```

In [19]:

```
df[(df["make"]=="plymouth") & (df["price"]>10000)]
```

Out[19]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-typ
124	3	122.0	plymouth	gas	hatchback	rwd	front	66.3	50.2	oh

In [20]:

```
df.drop(124, inplace=True)
```

In [21]:

```
df[(df["make"]=="toyota") & (df["price"]>12000)]
```

Out[21]:

	symboling	normalized-losses	make	fuel-type	body-style	drive-wheels	engine-location	width	height	engine-type
172	2	134.0	toyota	gas	convertible	rwd	front	65.6	53.0	ohc
178	3	197.0	toyota	gas	hatchback	rwd	front	67.7	52.0	dohc
179	3	197.0	toyota	gas	hatchback	rwd	front	67.7	52.0	dohc
180	-1	90.0	toyota	gas	sedan	rwd	front	66.5	54.1	dohc
181	-1	122.0	toyota	gas	wagon	rwd	front	66.5	54.1	dohc

In [22]:

```
df.drop([172,178,179,180,181], inplace=True)
```

Encoding the cleaned data.

In [23]:

```
df_cat = df.select_dtypes(object)
df_num = df.select_dtypes(["int64", "float64"])
```

In [24]:

```
df_num
```

Out[24]:

	symboling	normalized-losses	width	height	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	122.0	64.1	48.8	130	111.0	21	27	13495
1	3	122.0	64.1	48.8	130	111.0	21	27	16500
2	1	122.0	65.5	52.4	152	154.0	19	26	16500
3	2	164.0	66.2	54.3	109	102.0	24	30	13950
4	2	164.0	66.4	54.3	136	115.0	18	22	17450
...	...	...	...	...	...	...	...	...	...
200	-1	95.0	68.9	55.5	141	114.0	23	28	16845
201	-1	95.0	68.8	55.5	141	160.0	19	25	19045
202	-1	95.0	68.9	55.5	173	134.0	18	23	21485
203	-1	95.0	68.9	55.5	145	106.0	26	27	22470
204	-1	95.0	68.9	55.5	141	114.0	19	25	22625

194 rows × 9 columns

In [25]:

```
df_cat
```

Out[25]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type
0	alfa-romero	gas	convertible	rwd	front	dohc
1	alfa-romero	gas	convertible	rwd	front	dohc
2	alfa-romero	gas	hatchback	rwd	front	ohcv
3	audi	gas	sedan	fwd	front	ohc
4	audi	gas	sedan	4wd	front	ohc
...	...	...	...	...	...	...
200	volvo	gas	sedan	rwd	front	ohc
201	volvo	gas	sedan	rwd	front	ohc
202	volvo	gas	sedan	rwd	front	ohcv
203	volvo	diesel	sedan	rwd	front	ohc
204	volvo	gas	sedan	rwd	front	ohc

194 rows × 6 columns

In [26]:

```
from sklearn.preprocessing import LabelEncoder
```

In [27]:

```
for col in df_cat:
    le = LabelEncoder()
    df_cat[col] = le.fit_transform(df_cat[col])
```

In [28]:

```
df_cat
```

Out[28]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type
0	0	1	0	2	0	0
1	0	1	0	2	0	0
2	0	1	2	2	0	5
3	1	1	3	1	0	3
4	1	1	3	0	0	3
...	...	...	...	...	...	...
200	21	1	3	2	0	3
201	21	1	3	2	0	3
202	21	1	3	2	0	5
203	21	0	3	2	0	3
204	21	1	3	2	0	3

194 rows × 6 columns

- Succesfully converted categorical data into numerical data using LabelEncoder.

Creating new df by combining df_cat & df_num

In [29]:

```
df_num
```

Out[29]:

	symboling	normalized-losses	width	height	engine-size	horsepower	city-mpg	highway-mpg	price
0	3	122.0	64.1	48.8	130	111.0	21	27	13495
1	3	122.0	64.1	48.8	130	111.0	21	27	16500
2	1	122.0	65.5	52.4	152	154.0	19	26	16500
3	2	164.0	66.2	54.3	109	102.0	24	30	13950
4	2	164.0	66.4	54.3	136	115.0	18	22	17450
...	...	...	...	...	...	...	...	...	...
200	-1	95.0	68.9	55.5	141	114.0	23	28	16845
201	-1	95.0	68.8	55.5	141	160.0	19	25	19045
202	-1	95.0	68.9	55.5	173	134.0	18	23	21485
203	-1	95.0	68.9	55.5	145	106.0	26	27	22470
204	-1	95.0	68.9	55.5	141	114.0	19	25	22625

194 rows × 9 columns

In [30]:

```
df_cat
```

Out[30]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type
0	0	1	0	2	0	0
1	0	1	0	2	0	0
2	0	1	2	2	0	5
3	1	1	3	1	0	3
4	1	1	3	0	0	3
...	...	...	...	...	...	...
200	21	1	3	2	0	3
201	21	1	3	2	0	3
202	21	1	3	2	0	5
203	21	0	3	2	0	3
204	21	1	3	2	0	3

194 rows × 6 columns

In [31]:

```
df = pd.concat([df_cat, df_num], axis=1)
```

In [32]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 194 entries, 0 to 204
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   make               194 non-null    int32  
 1   fuel-type          194 non-null    int32  
 2   body-style         194 non-null    int32  
 3   drive-wheels       194 non-null    int32  
 4   engine-location    194 non-null    int32  
 5   engine-type        194 non-null    int32  
 6   symboling          194 non-null    int64  
 7   normalized-losses  194 non-null    float64
 8   width              194 non-null    float64
 9   height             194 non-null    float64
 10  engine-size        194 non-null    int64  
 11  horsepower         194 non-null    float64
 12  city-mpg           194 non-null    int64  
 13  highway-mpg        194 non-null    int64  
 14  price              194 non-null    int64  
dtypes: float64(4), int32(6), int64(5)
memory usage: 19.7 KB
```

EDA & Preprocessing

In [33]:

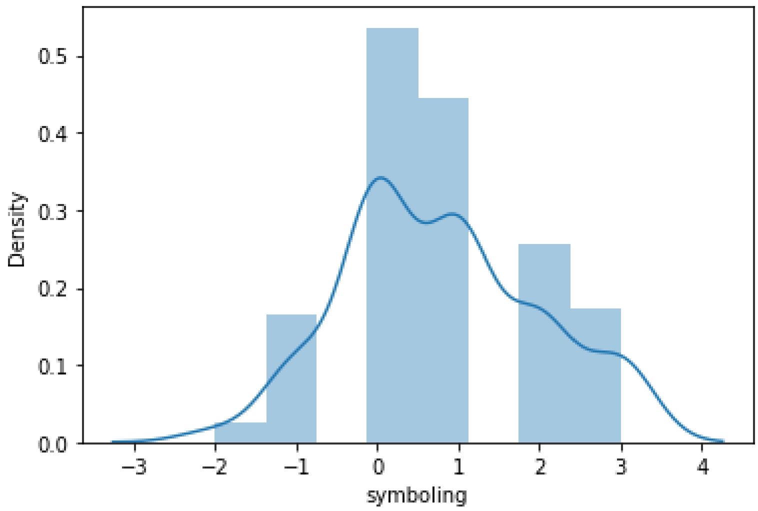
```
from scipy.stats import skew
```

In [34]:

```
for col in df:
    print(col)
    print(skew( df[col] ))

    plt.figure()
    sns.distplot(df[col])
    plt.show()
```

```
0.21386866184357742
```



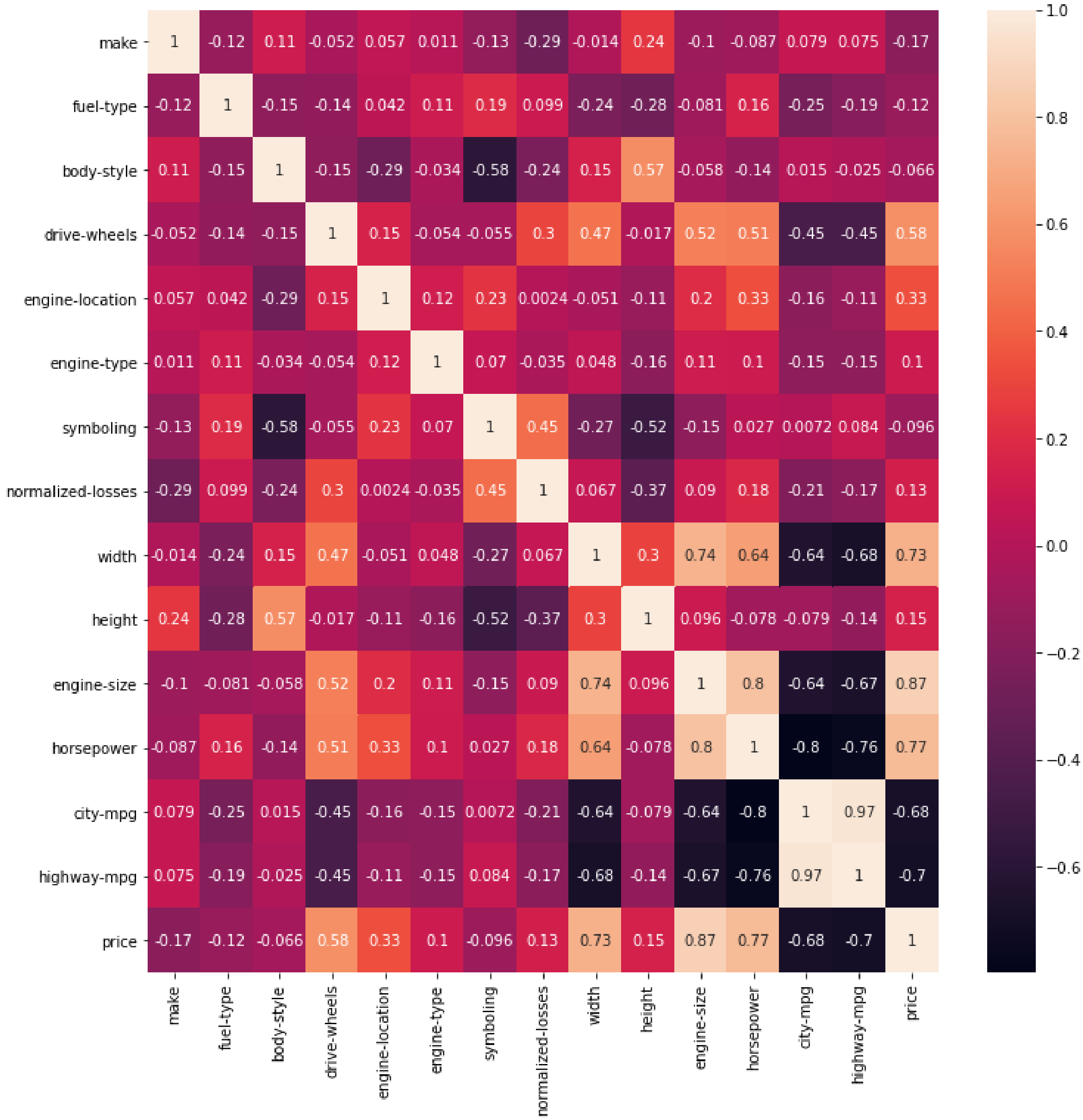

```
normalized-losses
0.848205953606264
```

In [35]:

```
plt.figure(figsize=(12,12))
sns.heatmap(df.corr() , annot=True)
```

Out[35]:

```
<AxesSubplot:>
```



In [36]:

```
df.corr()["price"].sort_values()
```

Out[36]:

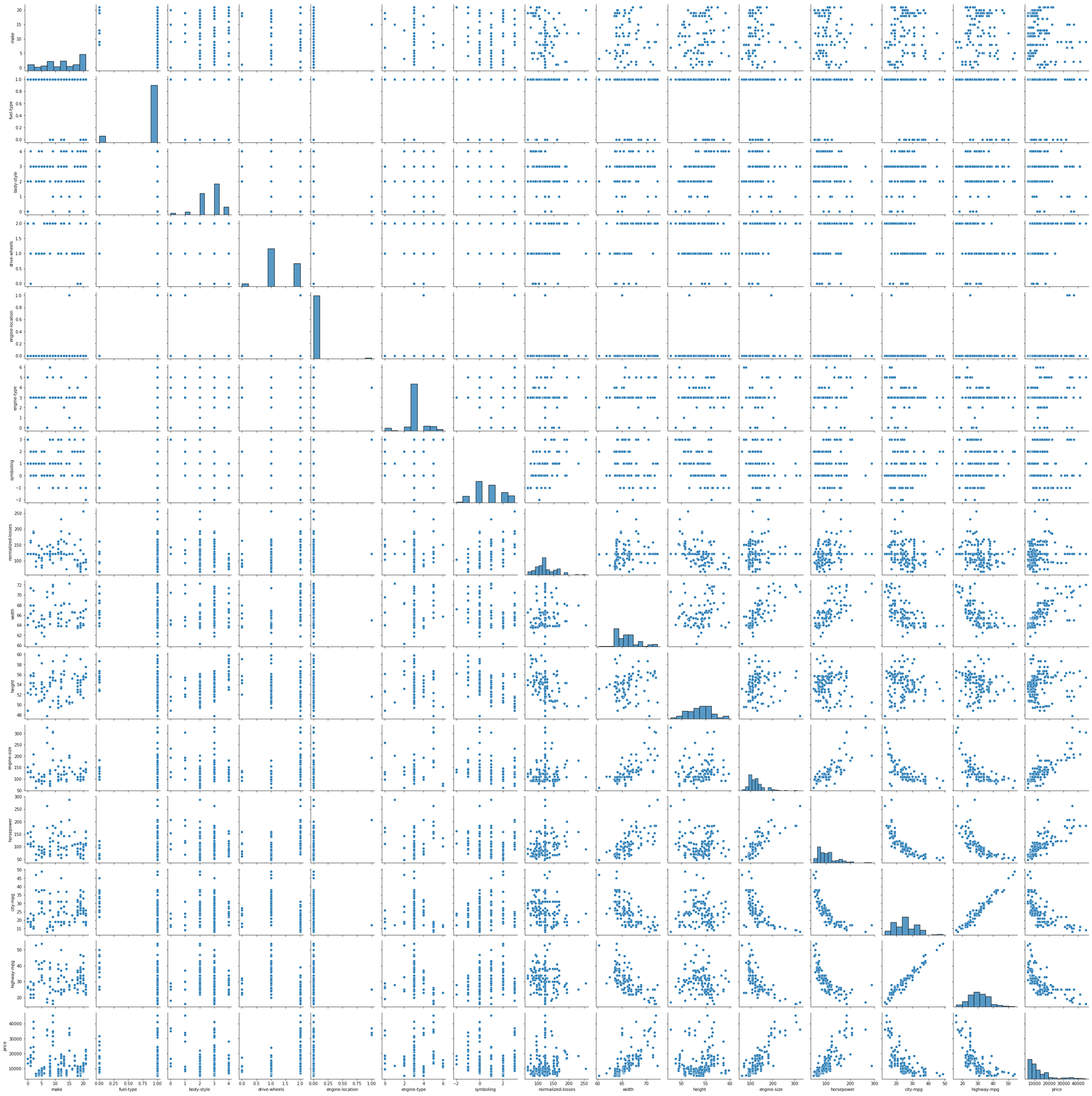
```
highway-mpg         -0.704846
city-mpg            -0.680412
make                -0.173792
fuel-type           -0.115791
symboling           -0.095905
body-style          -0.065831
engine-type          0.102758
normalized-losses    0.129973
height               0.147010
engine-location      0.333620
drive-wheels         0.584485
width                0.730503
horsepower           0.768921
engine-size          0.869638
price                1.000000
Name: price, dtype: float64
```

In [37]:

```
sns.pairplot(df)
```

Out[37]:

<seaborn.axisgrid.PairGrid at 0x22476f2b520>

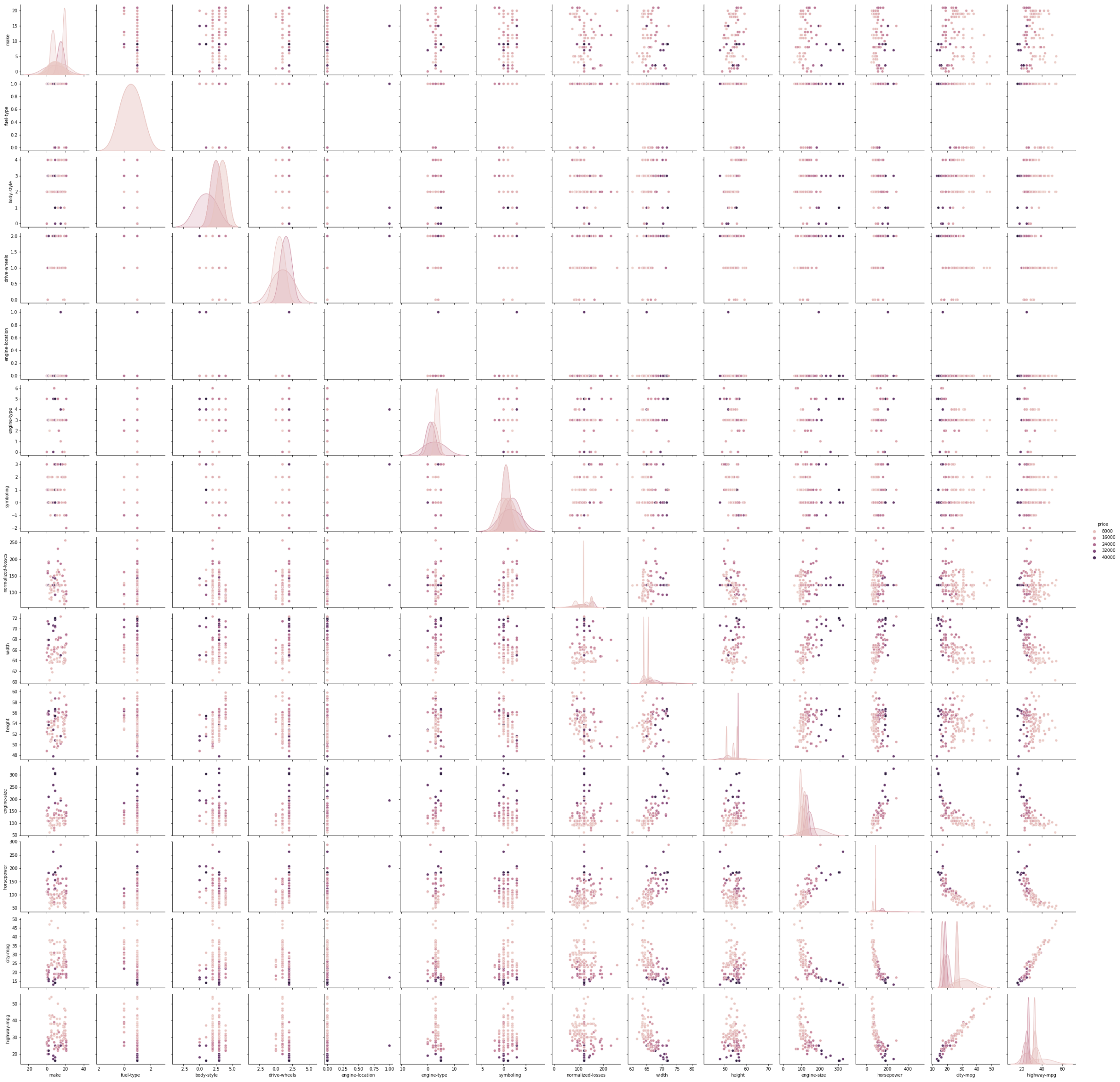


In [38]:

```
sns.pairplot(data=df , hue="price")
```

Out[38]:

```
<seaborn.axisgrid.PairGrid at 0x2247cc2f610>
```



Model Creation

In [39]:

```
df.head()
```

Out[39]:

	make	fuel-type	body-style	drive-wheels	engine-location	engine-type	symboling	normalized-losses	width	height	engine-siz
0	0	1	0	2	0	0	3	122.0	64.1	48.8	13
1	0	1	0	2	0	0	3	122.0	64.1	48.8	13
2	0	1	2	2	0	5	1	122.0	65.5	52.4	15
3	1	1	3	1	0	3	2	164.0	66.2	54.3	10
4	1	1	3	0	0	3	2	164.0	66.4	54.3	13

In [40]:

```
x = df.iloc[:,:-1]
y = df.iloc[:,-1]
```

In [41]:

```
xtrain, xtest, ytrain, ytest = train_test_split(x,y, test_size=0.2, random_state=1)
```

In [42]:

```
def mymodel(model):
    model.fit(xtrain, ytrain)
    ypred = model.predict(xtest)

    ac = r2_score(ytest, ypred)
    mae = mean_absolute_error(ytest, ypred)
    mse = mean_squared_error(ytest, ypred)
    rmse = np.sqrt(mse)

    train = model.score(xtrain,ytrain)
    test = r2_score(ytest,ypred)

    print(f"Training accuracy :- {train}")
    print(f"Testing accuracy  :- {test}")
    print()
    print(f"R2_score :- \n{ac}\n\nMAE :-\n{mae}\n\nMSE :-\n{mse}\n\nRMSE :-\n{rmse}")
```

In [43]:

```
models = []

models.append(("Linreg        :- ", LinearRegression()))
models.append(("KNN           :- ", KNeighborsRegressor()))
models.append(("SVM-l         :- ", SVR(kernel="linear")))
models.append(("SVM-r         :- ", SVR(kernel="rbf")))



for name, model in models:
    print(name)
    mymodel(model)
    print("\n\n\n")
```

```
Linreg        :-
Training accuracy :- 0.8941185334320063
Testing accuracy  :- 0.7355527822266537

R2_score :-
0.7355527822266537

MAE :-
3184.5155994121515

MSE :-
21864056.153787263

RMSE :-
4675.901640730615




KNN           :-
Training accuracy :- 0.9158165653923884
Testing accuracy  :- 0.6671791815444368

R2_score :-
0.6671791815444368

MAE :-
3298.7692307692305

MSE :-
27517071.743589748

RMSE :-
5245.671715194323




SVM-l         :-
Training accuracy :- 0.8339318161071796
Testing accuracy  :- 0.6151716897711921

R2_score :-
0.6151716897711921
```

```
MAE :-
3860.550037158177

MSE :-
31816964.667864848

RMSE :-
5640.652858301497




SVM-r         :-
Training accuracy :- -0.13936583233415778
Testing accuracy  :- -0.1791158963225512

R2_score :-
-0.1791158963225512

MAE :-
5925.926834043423

MSE :-
97487341.27774158

RMSE :-
9873.56780894027
```

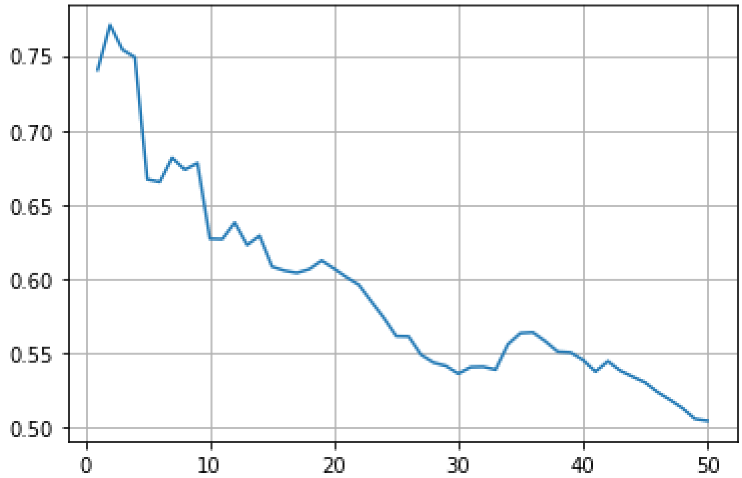
In [44]:

```
accuracy=[]

for i in range(1,51):
    knn = KNeighborsRegressor(n_neighbors=i)
    knn.fit(xtrain , ytrain)
    ypred = knn.predict(xtest)
    accuracy.append(r2_score(ytest,ypred))
```

In [45]:

```
plt.plot(range(1,51),accuracy)
plt.grid(True)
plt.show()
```



In [46]:

```
nl2 = []
for i in range(100):
    l2 = Ridge(alpha=i)
    l2.fit(xtrain,ytrain)
    nl2.append((i,l2.score(xtest,ytest)))

nl2
```

Out[46]:

```
[(0, 0.7355527822266543),
 (1, 0.7016542422732281),
 (2, 0.6847205630664456),
 (3, 0.6751690951781149),
 (4, 0.6689336369274135),
 (5, 0.6644614007235592),
 (6, 0.661043949885788 3),
 (7, 0.6583127977364198),
 (8, 0.6560570137239903),
 (9, 0.6541467733216799),
 (10, 0.6524975027598207),
 (11, 0.6510515309727207),
 (12, 0.6497680257188643),
 (13, 0.6486171525383218),
 (14, 0.6475765216649285),
 (15, 0.6466289392910423),
 (16, 0.6457609352149597),
 (17, 0.6449617700485919),
 (18, 0.6442227483265888),
 (19, 0.6435367323275265),
 (20, 0.6428977909174949),
 (21, 0.6423009412692836),
 (22, 0.6417419557549824),
 (23, 0.6412172154047594),
 (24, 0.6407235971861645),
 (25, 0.6402583862165862),
 (26, 0.6398192066102026),
 (27, 0.639403966428487),
 (28, 0.6390108134298362),
 (29, 0.6386380991777165),
 (30, 0.6382843496833958),
 (31, 0.6379482412052111),
 (32, 0.6376285801525738),
 (33, 0.6373242862842188),
 (34, 0.6370343785705443),
 (35, 0.636757963226026),
 (36, 0.6364942235212865),
 (37, 0.6362424110640186),
 (38, 0.6360018382995689),
 (39, 0.6357718720300747),
 (40, 0.6355519277888122),
 (41, 0.6353414649362952),
 (42, 0.6351399823684665),
 (43, 0.6349470147463903),
 (44, 0.6347621291722089),
 (45, 0.634584922248591 1),
 (46, 0.6344150174690382),
 (47, 0.6342520628947241),
 (48, 0.6340957290803899),
```

```
 (49, 0.633945707217447),
 (50, 0.6338017074671669),
 (51, 0.6336634574607254),
 (52, 0.6335307009461641),
 (53, 0.6334031965650795),
 (54, 0.6332807167441894),
 (55, 0.633163046688879),
 (56, 0.6330499834675272),
 (57, 0.6329413351768225),
 (58, 0.6328369201795169),
 (59, 0.6327365664071011),
 (60, 0.6326401107208057),
 (61, 0.6325473983250953),
 (62, 0.6324582822285081),
 (63, 0.6323726227472874),
 (64, 0.632290287047744),
 (65, 0.6322111487237436),
 (66, 0.6321350874061213),
 (67, 0.6320619884011252),
 (68, 0.6319917423553328),
 (69, 0.6319242449447335),
 (70, 0.6318593965858967),
 (71, 0.6317971021673566),
 (72, 0.6317372707995456),
 (73, 0.6316798155817325),
 (74, 0.6316246533846166),
 (75, 0.631571704647304),
 (76, 0.6315208931875617),
 (77, 0.6314721460242934),
 (78, 0.6314253932113333),
 (79, 0.6313805676816733),
 (80, 0.6313376051013773),
 (81, 0.6312964437324361),
 (82, 0.6312570243039552),
 (83, 0.6312192898910409),
 (84, 0.6311831858008601),
 (85, 0.6311486594653745),
 (86, 0.6311156603402739),
 (87, 0.6310841398096971),
 (88, 0.631054051096347),
 (89, 0.631025349176634),
 (90, 0.6309979907005325),
 (91, 0.6309719339158172),
 (92, 0.630947138596424),
 (93, 0.6309235659746519),
 (94, 0.6309011786769745),
 (95, 0.6308799406632299),
 (96, 0.6308598171689843),
 (97, 0.6308407746508724),
 (98, 0.6308227807347309),
 (99, 0.6308058041663689)]
```

In [47]:

```
nl1 = []
for i in range(100):
    l1 = Lasso(alpha=i)
    l1.fit(xtrain,ytrain)
    nl1.append((i,l1.score(xtest,ytest)))

nl1
```

Out[47]:

```
[(0, 0.7355527822266535),
 (1, 0.7349043968321456),
 (2, 0.734224129247808),
 (3, 0.7335119482950561),
 (4, 0.7327678216112494),
 (5, 0.7319918069060299),
 (6, 0.7311838864554618),
 (7, 0.7303440009125781),
 (8, 0.7294722457664095),
 (9, 0.728568558705089),
 (10, 0.7276329142276274),
 (11, 0.7266654426694357),
 (12, 0.7256659278294073),
 (13, 0.7246344541902952),
 (14, 0.7235711776815625),
 (15, 0.7224758377877629),
 (16, 0.7213487164982898),
 (17, 0.7201895305351079),
 (18, 0.7189984282153095),
 (19, 0.7177754221166048),
 (20, 0.7165203766929641),
 (21, 0.71523345414556),
 (22, 0.7139146862983363),
 (23, 0.7125639314145946),
 (24, 0.711812791847517),
 (25, 0.7097664480053986),
 (26, 0.7083200026249673),
 (27, 0.7068417949659476),
 (28, 0.7053387530630352),
 (29, 0.703825264895338),
 (30, 0.7022833223294923),
 (31, 0.7007156529267059),
 (32, 0.6991170851658983),
 (33, 0.6974860191261013),
 (34, 0.6958241390246818),
 (35, 0.6941295302583039),
 (36, 0.6924042214194559),
 (37, 0.690646107618992),
 (38, 0.6888562462795077),
 (39, 0.6870357825159619),
 (40, 0.6851823850675423),
 (41, 0.683298529611013),
 (42, 0.6813815856828743),
 (43, 0.6794343568301573),
 (44, 0.677453863516487 5),
 (45, 0.6754432355182265),
 (46, 0.6734010488724398),
 (47, 0.6713249820637119),
```

```
 (48, 0.6692193955744843),
 (49, 0.6670797874674177),
 (50, 0.6649084727382653),
 (51, 0.6627054026816726),
 (52, 0.6604705772976398),
 (53, 0.6582040808387541),
 (54, 0.6559060756859811),
 (55, 0.6535910737116393),
 (56, 0.6512443878924672),
 (57, 0.6488660296640226),
 (58, 0.6464561105279234),
 (59, 0.644014572224729),
 (60, 0.6415413778776521),
 (61, 0.6390365387265051),
 (62, 0.6365160593232179),
 (63, 0.6340051835687787),
 (64, 0.6320858674588254),
 (65, 0.6319941472925529),
 (66, 0.6319023132801276),
 (67, 0.6318125461834823),
 (68, 0.6317200169467454),
 (69, 0.6316273948684725),
 (70, 0.6315345696511616),
 (71, 0.6314438329521539),
 (72, 0.6313504617482983),
 (73, 0.6312569097328515),
 (74, 0.6311631558307014),
 (75, 0.6310694268918934),
 (76, 0.6309754456784029),
 (77, 0.6308836678692045),
 (78, 0.630789171403301),
 (79, 0.6306943923956252),
 (80, 0.6305993617142169),
 (81, 0.6305040879516217),
 (82, 0.6304086000844271),
 (83, 0.6303128605110933),
 (84, 0.6302168863052457),
 (85, 0.630120682703242),
 (86, 0.6300242273703448),
 (87, 0.6299275197257057),
 (88, 0.6298305598564977),
 (89, 0.6297333708063313),
 (90, 0.6296359391984844),
 (91, 0.6295382558267923),
 (92, 0.6294427720509725),
 (93, 0.6293567923373555),
 (94, 0.6292773020643387),
 (95, 0.6291975819920547),
 (96, 0.629117638518707),
 (97, 0.6290374703564884),
 (98, 0.6289570770217647),
 (99, 0.6288764657124297)]
```

In [48]:

```
models = []

models.append(("Linreg        :- ", LinearRegression()))
models.append(("KNN           :- ", KNeighborsRegressor(n_neighbors=1)))
models.append(("SVM-l         :- ", SVR(kernel="linear")))
models.append(("SVM-r         :- ", SVR(kernel="rbf")))



for name, model in models:
    print(name)
    mymodel(model)
    print("\n\n\n")
```

```
Linreg        :-
Training accuracy :- 0.8941185334320063
Testing accuracy  :- 0.7355527822266537

R2_score :-
0.7355527822266537

MAE :-
3184.5155994121515

MSE :-
21864056.153787263

RMSE :-
4675.901640730615




KNN           :-
Training accuracy :- 0.9928862905865443
Testing accuracy  :- 0.74027773895488

R2_score :-
0.74027773895488

MAE :-
2592.5384615384614

MSE :-
21473404.589743588

RMSE :-
4633.940503474725




SVM-l         :-
Training accuracy :- 0.8339318161071796
Testing accuracy  :- 0.6151716897711921

R2_score :-
0.6151716897711921
```

```
MAE :-
3860.550037158177

MSE :-
31816964.667864848

RMSE :-
5640.652858301497



SVM-r         :-
Training accuracy :- -0.13936583233415778
Testing accuracy  :- -0.1791158963225512

R2_score :-
-0.1791158963225512

MAE :-
5925.926834043423

MSE :-
97487341.27774158

RMSE :-
9873.56780894027
```

Cross Validation Score

In [49]:

```
print("Name                Accuracy                   STD")
for name, model in models:
    cvs = cross_val_score(model, x, y, cv=5,)
    print(f"{name} {cvs.mean()} {cvs.std()}")
```

```
Name              Accuracy                  STD
Linreg        :-  0.5155626638216251 0.29417502732645967
KNN           :-  0.3998279418728215 0.3576296707751772
SVM-l         :-  0.5323432027166152 0.33526075240067404
SVM-r         :-  -0.1881168806138271 0.13892274676036426
```

In [ ]:

```

```