Used cars EDA and Forecasting

```
#Objective:
Provide the best-performing model to determine the price of the used car.
Providing the most important features which determine the price.
```

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

df_cars=pd.read_csv('cars.csv')
df_cars.head()
```

Out[1]:

	Id	year	brand	full_model_name	model_name	price	distance_travelled(kms)	fuel_type	city	brand_rank	car_age
0	0	2016	Honda	Honda Brio S MT	Brio	425000.0	9680.0	Petrol	Mumbai	7	5.0
1	1	2012	Nissan	Nissan Sunny XV Diesel	Sunny	325000.0	119120.0	Diesel	Mumbai	11	9.0
2	2	2017	Toyota	Toyota Fortuner 2.8 4x2 MT [2016-2020]	Fortuner	2650000.0	64593.0	Diesel	Thane	1	4.0
3	3	2017	Mercedes-Benz	Mercedes-Benz E-Class E 220d Expression [2019-...	E-Class	4195000.0	25000.0	Diesel	Mumbai	2	4.0
4	4	2012	Hyundai	Hyundai Verna Fluidic 1.6 CRDi SX	Verna	475000.0	23800.0	Diesel	Mumbai	14	9.0

In [2]:
```
df_cars.shape
```

Out[2]:
```
(1725, 11)
```

In [3]:
```
df_cars.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1725 entries, 0 to 1724
Data columns (total 11 columns):
 #   Column                   Non-Null Count  Dtype  
---  ------                   --------------  -----  
 0   Id                       1725 non-null   int64  
 1   year                     1725 non-null   int64  
 2   brand                    1725 non-null   object 
 3   full_model_name          1725 non-null   object 
 4   model_name               1725 non-null   object 
 5   price                    1725 non-null   float64
 6   distance_travelled(kms)  1725 non-null   float64
 7   fuel_type                1725 non-null   object 
 8   city                     1725 non-null   object 
 9   brand_rank               1725 non-null   int64  
 10  car_age                  1725 non-null   float64
dtypes: float64(3), int64(3), object(5)
memory usage: 148.4+ KB
```

In [4]:
```
df_cars['year'].unique()
```

Out[4]:
```
array([2016, 2012, 2017, 2019, 2018, 2015, 2010, 2013, 2014, 2020, 2009,
       2006, 2021, 2008, 2011, 2007, 2005, 2004, 1990], dtype=int64)
```

In [5]:
```
df_cars['brand'].unique()
```

Out[5]:
```
array(['Honda', 'Nissan', 'Toyota', 'Mercedes-Benz', 'Hyundai',
       'Maruti Suzuki', 'Renault', 'Volkswagen', 'Skoda', 'BMW', 'Tata',
       'Audi', 'Bentley', 'Ford', 'Mahindra', 'Jaguar', 'Lamborghini',
       'MINI', 'Land Rover', 'Chevrolet', 'Datsun', 'Jeep', 'Porsche',
       'Volvo', 'MG', 'Lexus', 'Mitsubishi', 'Kia', 'Fiat', 'Isuzu',
       'Mahindra-Renault'], dtype=object)
```

In [6]:
```
df_cars['full_model_name'].unique()
```

Out[6]:
```
array(['Honda Brio S MT', 'Nissan Sunny XV Diesel',
       'Toyota Fortuner 2.8 4x2 MT [2016-2020]',
       'Mercedes-Benz E-Class E 220d Expression [2019-2019]',
       'Hyundai Verna Fluidic 1.6 CRDi SX',
       'Hyundai i20 Sportz 1.2 BS-IV', 'Toyota Glanza V',
       'Mercedes-Benz GLE 250 d',
       'Hyundai Grand i10 Sportz (O) AT 1.2 Kappa VTVT [2017-2018]',
       'Maruti Suzuki Swift Dzire ZXI', 'Hyundai Xcent SX 1.2 (O)',
       'Toyota Innova Crysta 2.4 G 7 STR [2016-2017]',
       'Maruti Suzuki Baleno Alpha 1.2',
       'Renault Pulse RxL Petrol [2015-2017]',
       'Maruti Suzuki Baleno Zeta 1.2 AT', 'Hyundai Xcent S AT 1.2 (O)',
       'Toyota Corolla Altis VL AT Petrol', 'Maruti Suzuki Swift ZDi',
       'Volkswagen Polo Highline1.2L (P)', 'Honda WR-V VX MT Petrol',
       'Honda WR-V S MT Petrol', 'Maruti Suzuki Ritz VXI BS-IV',
       'Hyundai Grand i10 Sportz 1.2 Kappa VTVT [2013-2016]',
       'Skoda Rapid Style 1.5 TDI AT', 'Honda City V Diesel',
       'Maruti Suzuki Celerio VXi AMT ABS', 'BMW 3 Series 320d Prestige',
       'Volkswagen Vento TSI', 'Honda Jazz V AT Petrol',
```

In [7]:
```
df_cars['model_name'].unique()
```

Out[7]:
```
array(['Brio', 'Sunny', 'Fortuner', 'E-Class', 'Verna', 'i20', 'Glanza',
       'GLE', 'Grand', 'Swift', 'Xcent', 'Innova', 'Baleno', 'Pulse',
       'Corolla', 'Polo', 'WR-V', 'Ritz', 'Rapid', 'City', 'Celerio', '3',
       'Vento', 'Jazz', 'Nano', 'GLA', 'A-Star', 'Q5', 'X1', 'Z4', 'A3',
       'A4', 'X3', 'Continental', 'Q3', 'Q7', 'A6', '7', 'Endeavour',
       'XUV500', 'F-Pace', 'XE', 'Gallardo', 'Countryman', 'C-Class',
       'Evoque', 'S-Class', 'Lodgy', 'CLA', 'Creta', 'A8', 'B-Class',
       'A-Class', '5', 'Cooper', 'Terrano', 'CR-V', 'Freelander', 'Ciaz',
       'Beat', 'KUV100', 'Duster', 'redi-GO', 'Tiago', 'Altroz', 'TUV300',
       'Vitara', 'Etios', 'Figo', 'Civic', 'Compass', 'Elite', 'Bolero',
       'SX4', 'Cayenne', 'V40', 'Superb', 'Dzire', 'i10', 'Zest', 'BR-V',
       'S-Cross', 'Elantra', 'Discovery', 'Accord', 'Scorpio', 'X5',
       'Estilo', 'XF', 'Santro', '6', 'GO', 'Passat', 'Wagon', 'Jetta',
       'Safari', 'EcoSport', 'Kwid', 'Hector', 'Mustang', 'SLK', 'X7',
       '718', 'Aria', '3.0', 'Land', 'GLC', 'NX', 'Thar', 'ES', 'GLS',
       'Ertiga', 'Ameo', 'Sport', 'Harrier', 'Outlander', 'Seltos',
       'Amaze', 'Octavia', 'M-Class', 'GL', 'XUV300', 'Avventura',
       'Micra', 'Eeco', 'Eon', 'Fluidic', 'Alto', 'Cross', 'Yaris',
       'Triber', 'Nexon', 'Tigor', 'Aspire', 'Venue', 'Freestyle', '4.4',
       'XL6', 'Qualis', 'Enjoy', 'Verito', 'S60', 'XC90', 'G-Class',
       'MU-X', 'Tiguan', 'Sonet', 'XJ', 'S90', 'Tucson', 'Indica',
       'Laura', 'Hexa', 'Captur', 'Yeti', 'Quanto', 'Camry', 'XC60',
       'Fiesta', 'CLS', 'X4', 'Ignis', 'Alturas', 'Abarth', 'Fabia',
       'Pajero', 'Cruze', 'Logan', 'Jeep'], dtype=object)
```

In [8]:
```
df_cars['fuel_type'].unique()
```

Out[8]:
```
array(['Petrol', 'Diesel', 'Petrol + 1', 'CNG + 1', 'Hybrid'],
      dtype=object)
```

In [9]:
```
df_cars['city'].unique()
```

Out[9]:
```
array(['Mumbai', 'Thane', 'Dehradun', 'Navi Mumbai', 'Delhi', 'Noida',
       'Ghaziabad', 'Panchkula', 'Faridabad', 'Agra', 'Lucknow',
       'Bangalore', 'Hyderabad', 'Chennai', 'Pune'], dtype=object)
```

In [10]:
```
df_cars['brand_rank'].unique()
```

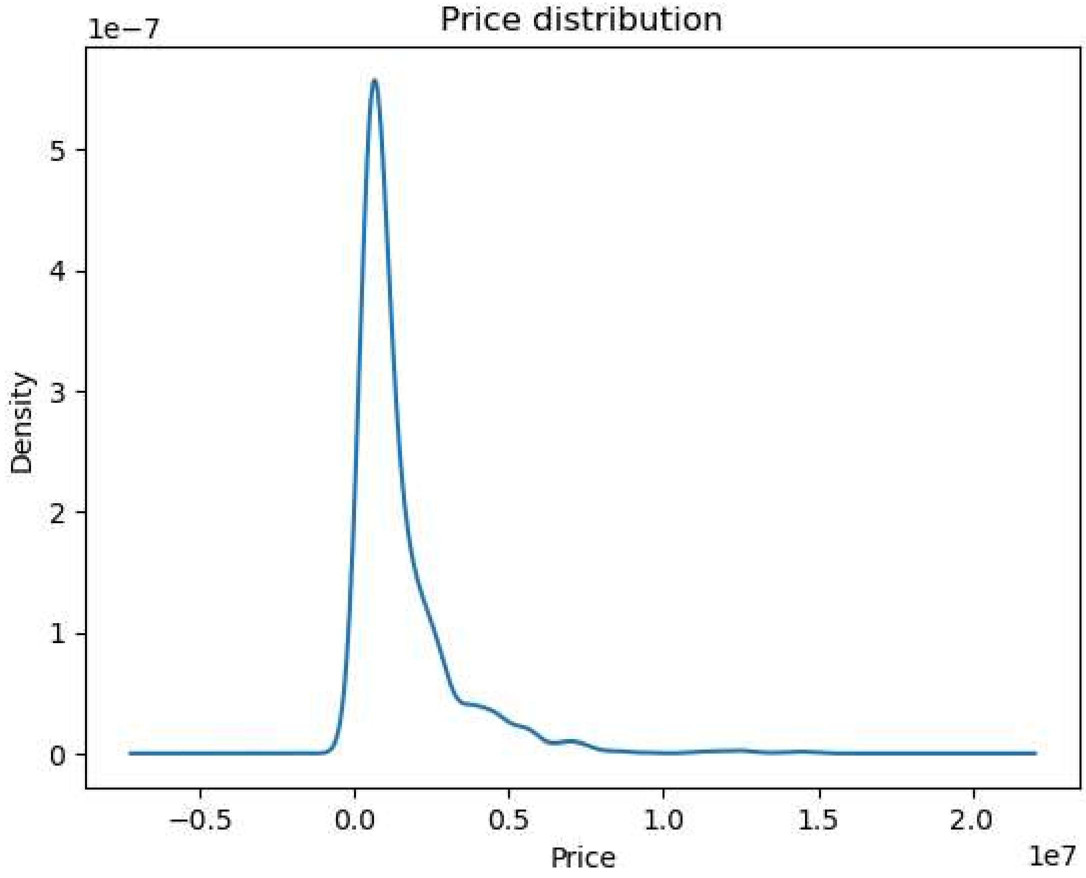
Out[10]:
```
array([ 7, 11,  1,  2, 14, 32, 15,  3, 27,  4, 40, 10, 44,  8, 24, 45, 39,
       18, 12, 50, 19,  5,  9, 81, 16, 46, 20, 43, 37], dtype=int64)
```

In [11]:
```
df_cars.describe()
```

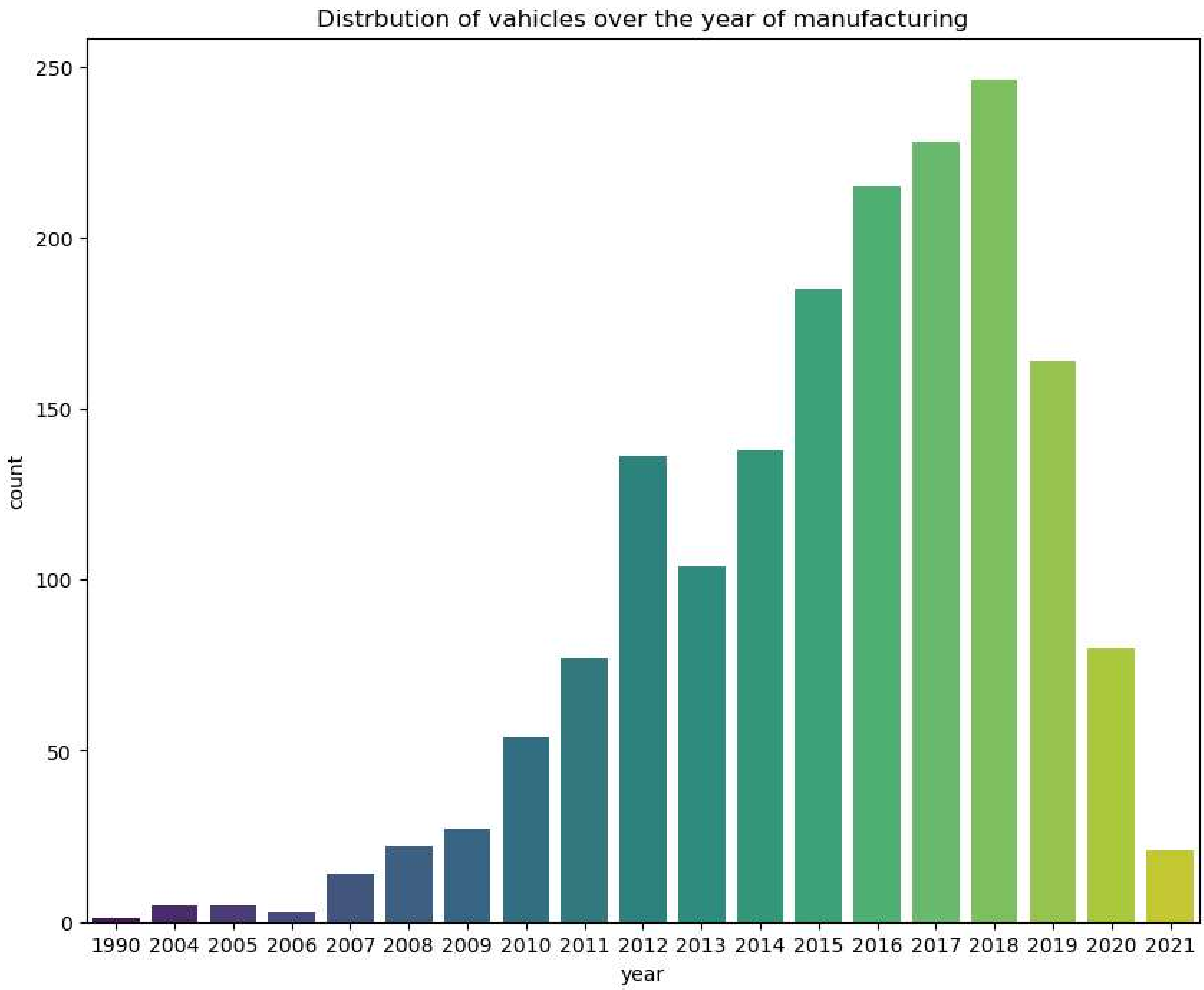
Out[11]:

	Id	year	price	distance_travelled(kms)	brand_rank	car_age
count	1725.000000	1725.000000	1.725000e+03	1725.000000	1725.000000	1725.000000
mean	862.000000	2015.390725	1.494837e+06	53848.256232	15.731014	5.609275
std	498.108924	3.207504	1.671658e+06	44725.541963	12.951122	3.207504
min	0.000000	1990.000000	6.250000e+04	350.000000	1.000000	0.000000
25%	431.000000	2013.000000	5.450000e+05	29000.000000	5.000000	3.000000
50%	862.000000	2016.000000	8.750000e+05	49000.000000	14.000000	5.000000
75%	1293.000000	2018.000000	1.825000e+06	70500.000000	24.000000	8.000000
max	1724.000000	2021.000000	1.470000e+07	790000.000000	81.000000	31.000000

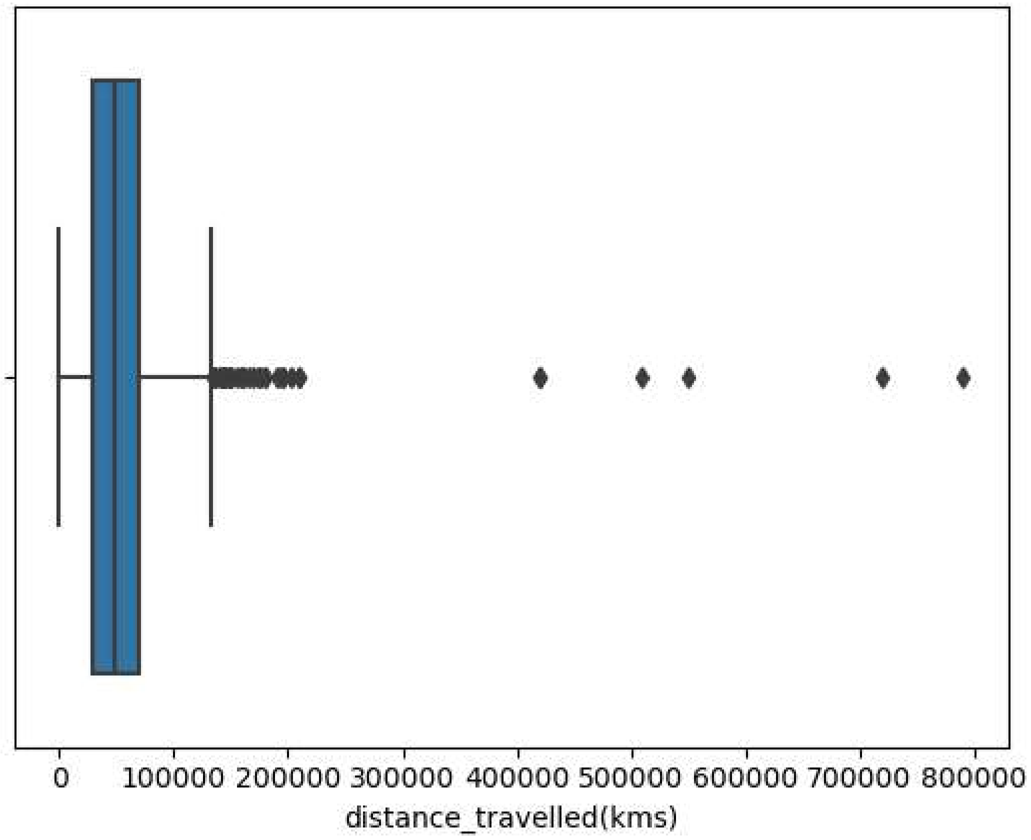
In [12]:
```
#Price distribution (probability density function) for price of listed cars

ax=df_cars['price'].plot(kind='kde',title='Price distribution')
ax.set_xlabel('Price')
plt.show()
```



In [13]:
```
#univariate analysis of year of manufacturing of listed cars
plt.figure(figsize=(10,8))
plt.title('Distribution of vahicles over the year of manufacturing')
sns.countplot(data=df_cars,x='year',palette='viridis')
plt.show()
```



In [14]:
```
#univariate analysis for distance travelled(kms)
sns.boxplot(data=df_cars,x='distance_travelled(kms)')
plt.title('Distance travelled by used cars')
plt.show()
```

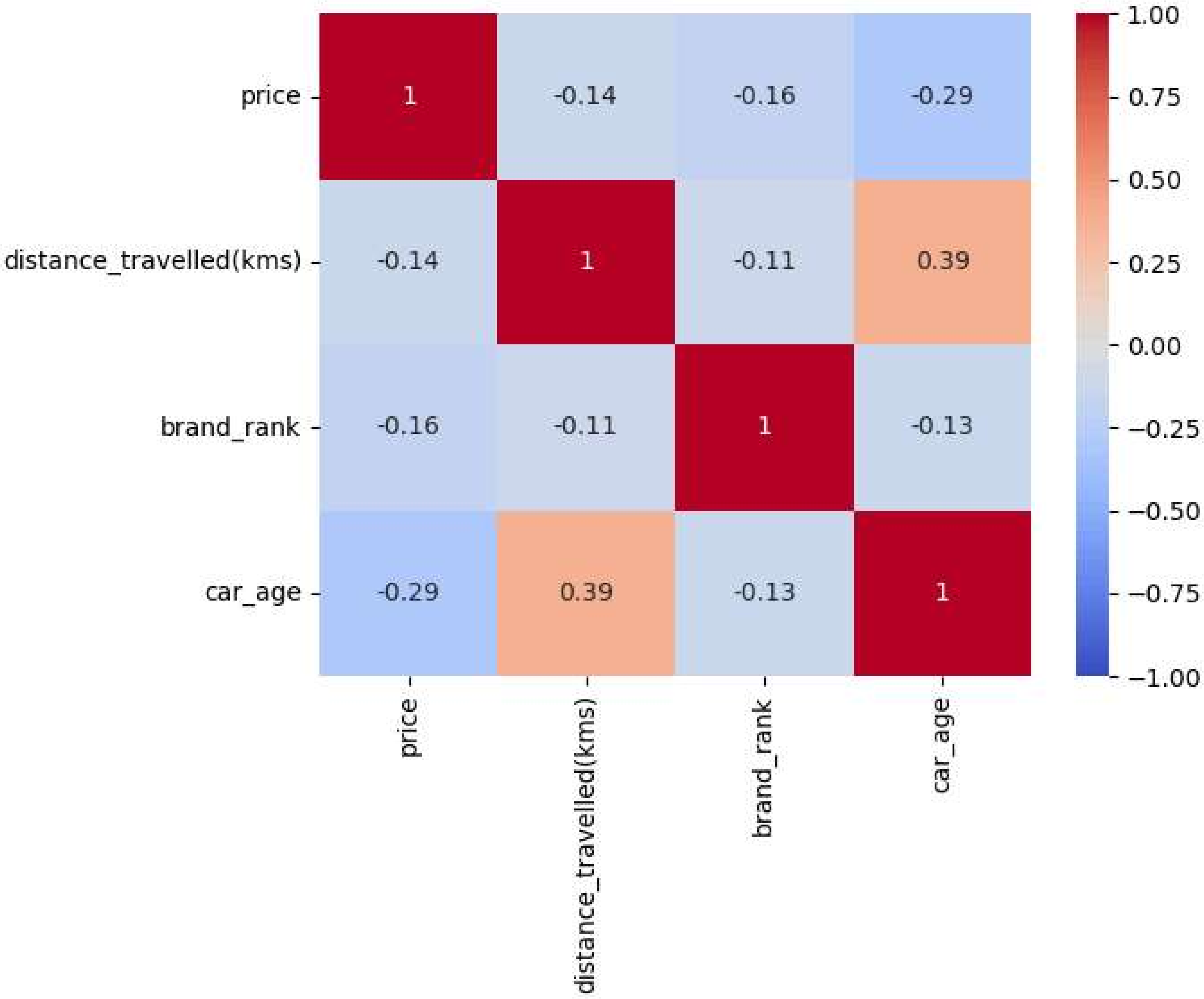


In [15]:
```
#correlation between numerical values
df_cars1=df_cars[['price','distance_travelled(kms)','brand_rank','car_age']]
correlation=df_cars1.corr()
correlation
```

Out[15]:

	price	distance_travelled(kms)	brand_rank	car_age
price	1.000000	-0.137351	-0.164591	-0.288483
distance_travelled(kms)	-0.137351	1.000000	-0.111406	0.386107
brand_rank	-0.164591	-0.111406	1.000000	-0.134275
car_age	-0.288483	0.386107	-0.134275	1.000000

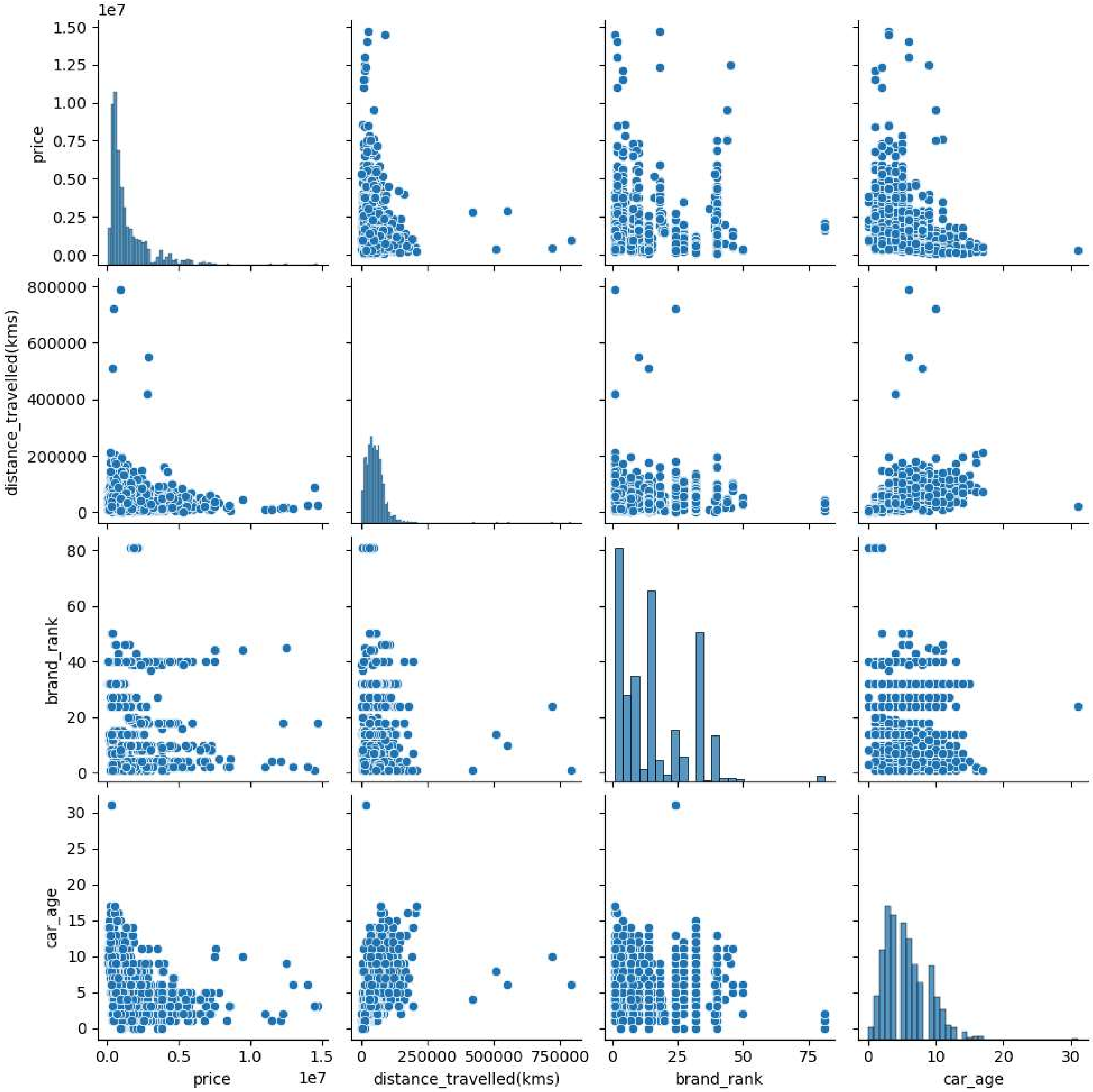
In [16]:
```
#correlation heatmap showing less correlation,negetive correlation across numerical variables
sns.heatmap(correlation,annot=True,vmin=-1,vmax=1,cmap='coolwarm')
plt.show()
```



In [21]:
```
#multi variate analysis over different numerical variables. Pair plot shows correaltion between numerical variables.
sns.pairplot(df_cars1)
plt.show()
```

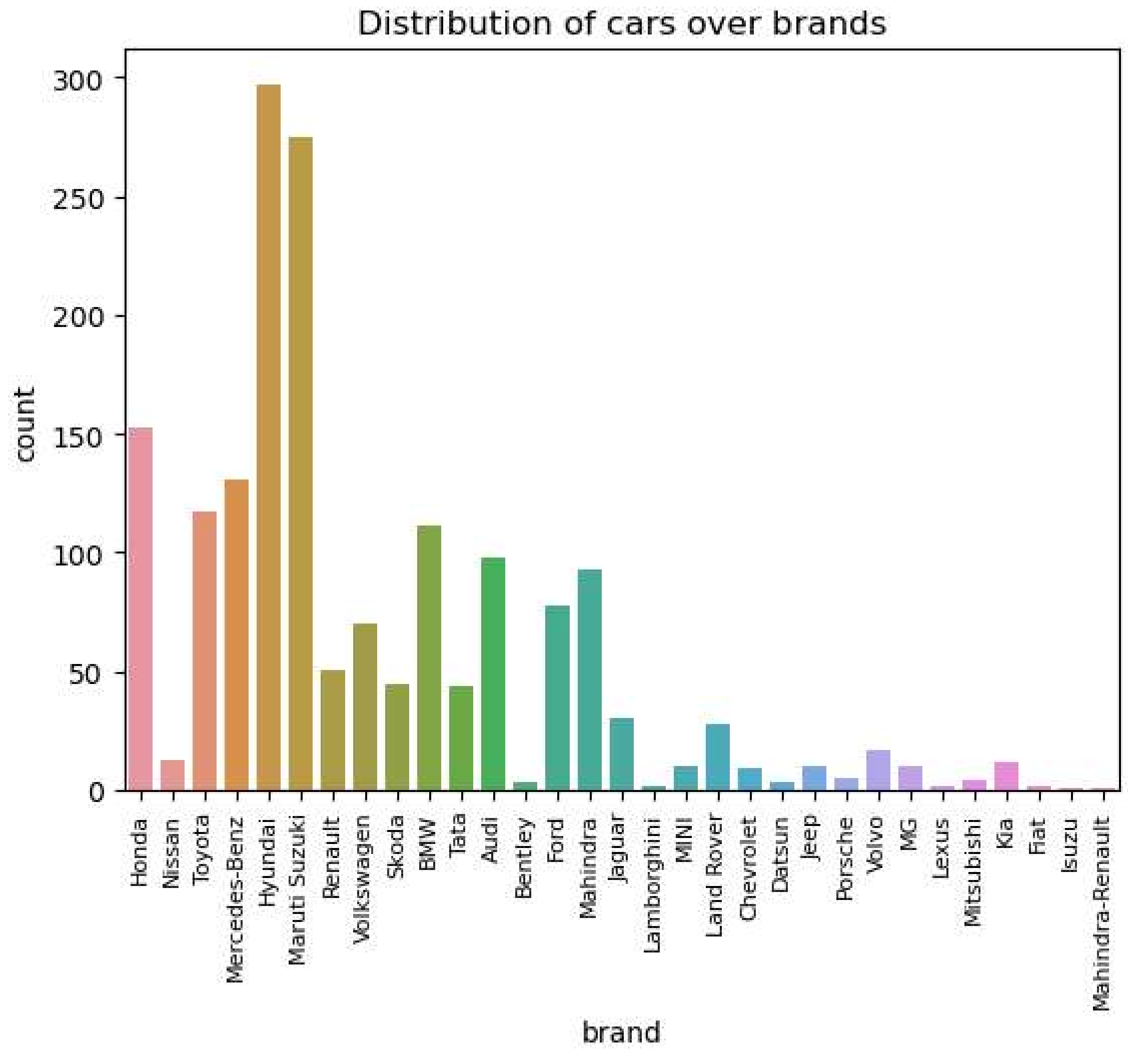
```
C:\Users\radhi\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure layout has changed to t
ight
  self._figure.tight_layout(*args, **kwargs)
```

Out[21]: <seaborn.axisgrid.PairGrid at 0x21042692dd0>



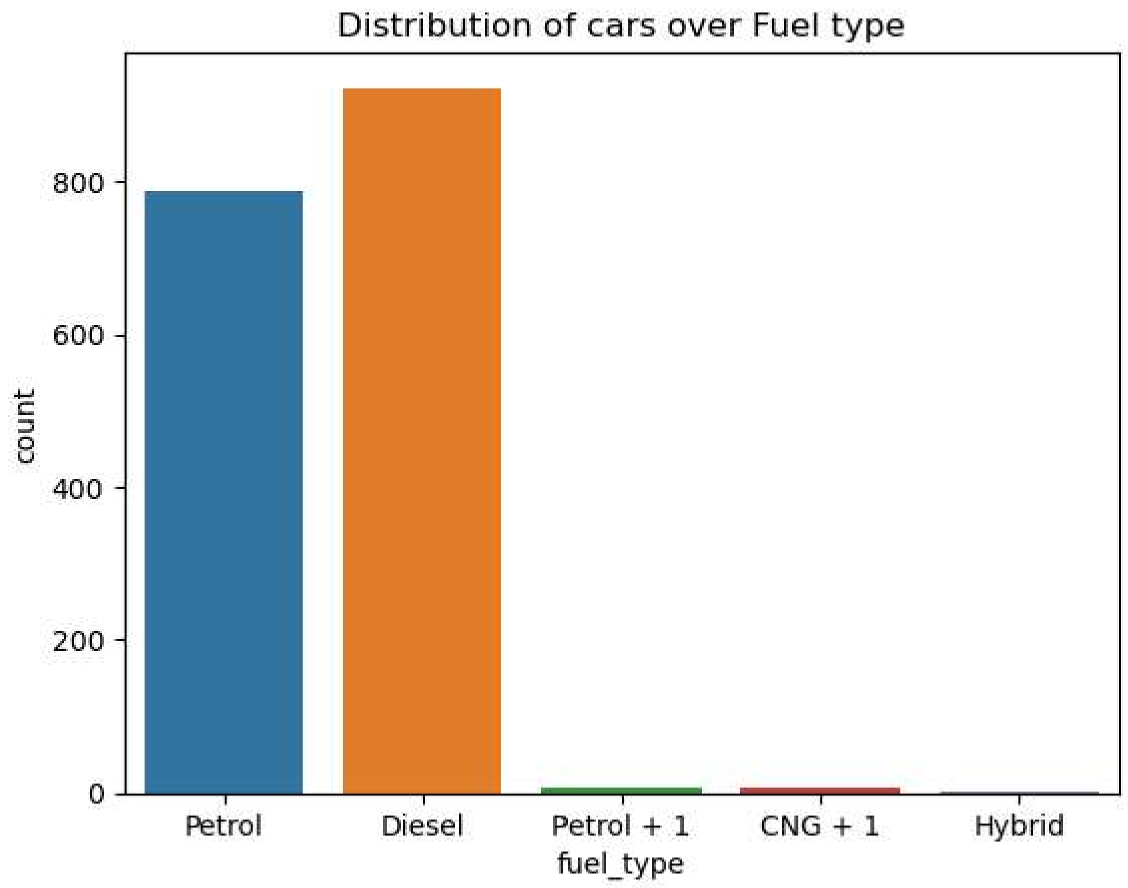
In [17]:

```
#univariate analysis. brand distribution in listed cars.
sns.countplot(data=df_cars,x='brand')
plt.xticks(rotation=90,fontsize=8)
plt.title('Distribution of cars over brands')
plt.show()
```

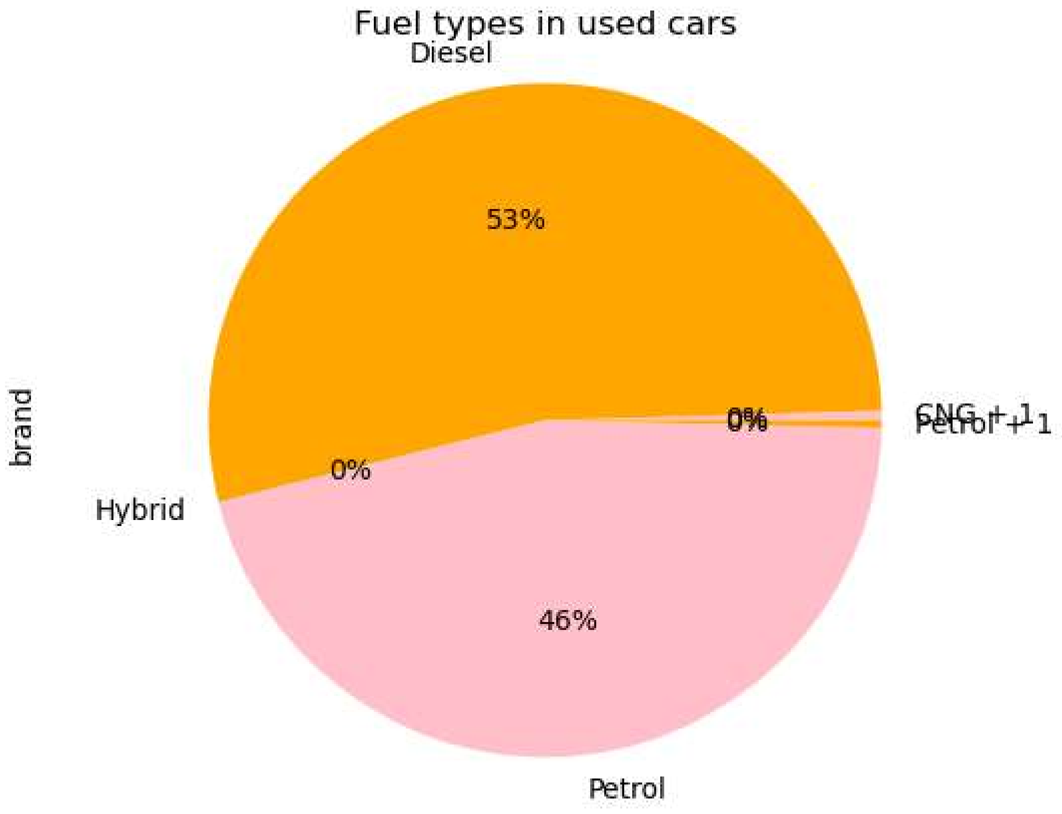


In [20]:

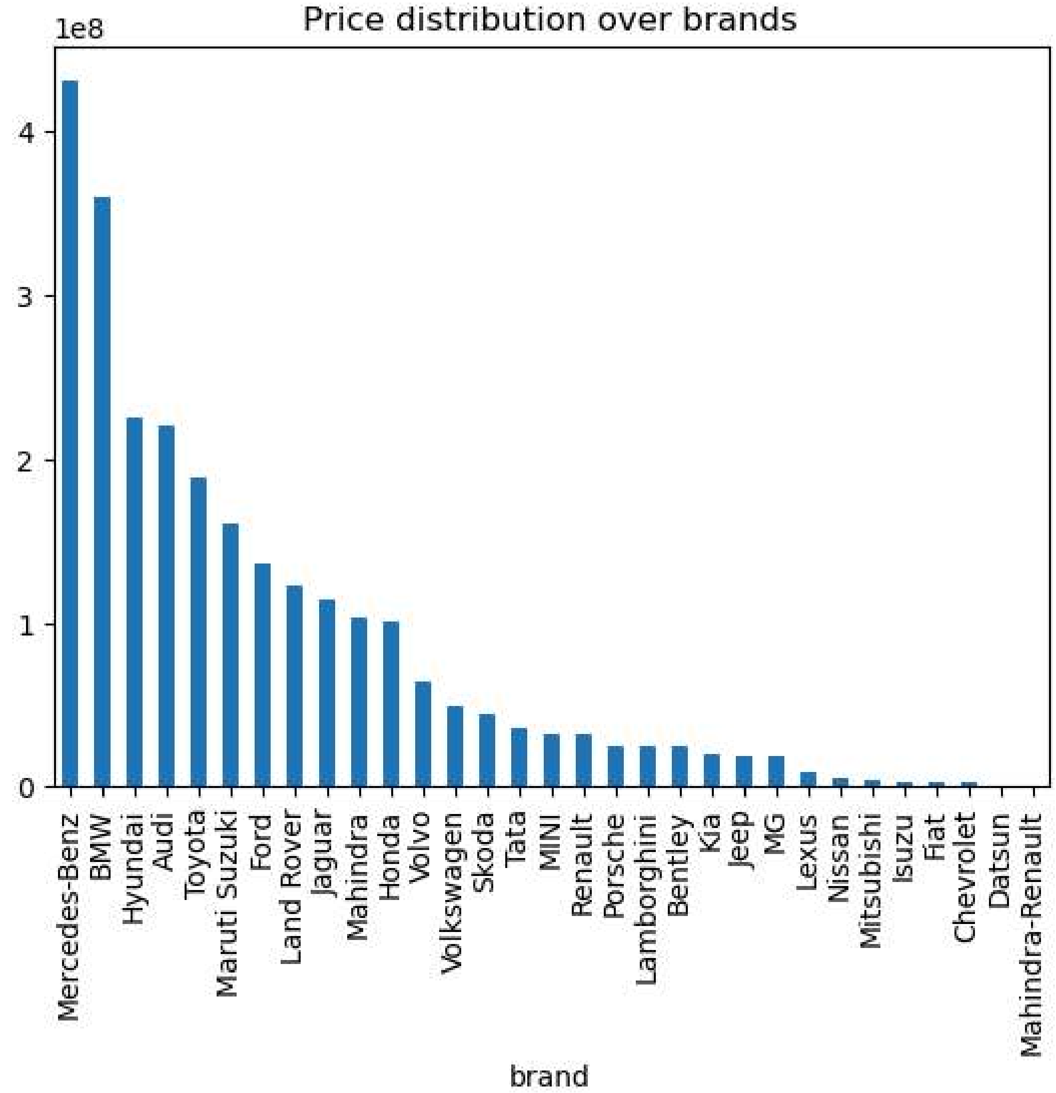
```
#univariate analysis. count of fuel types in listed cars
plt.title('Distribution of cars over Fuel type')
sns.countplot(data=df_cars,x='fuel_type')
plt.show()
```



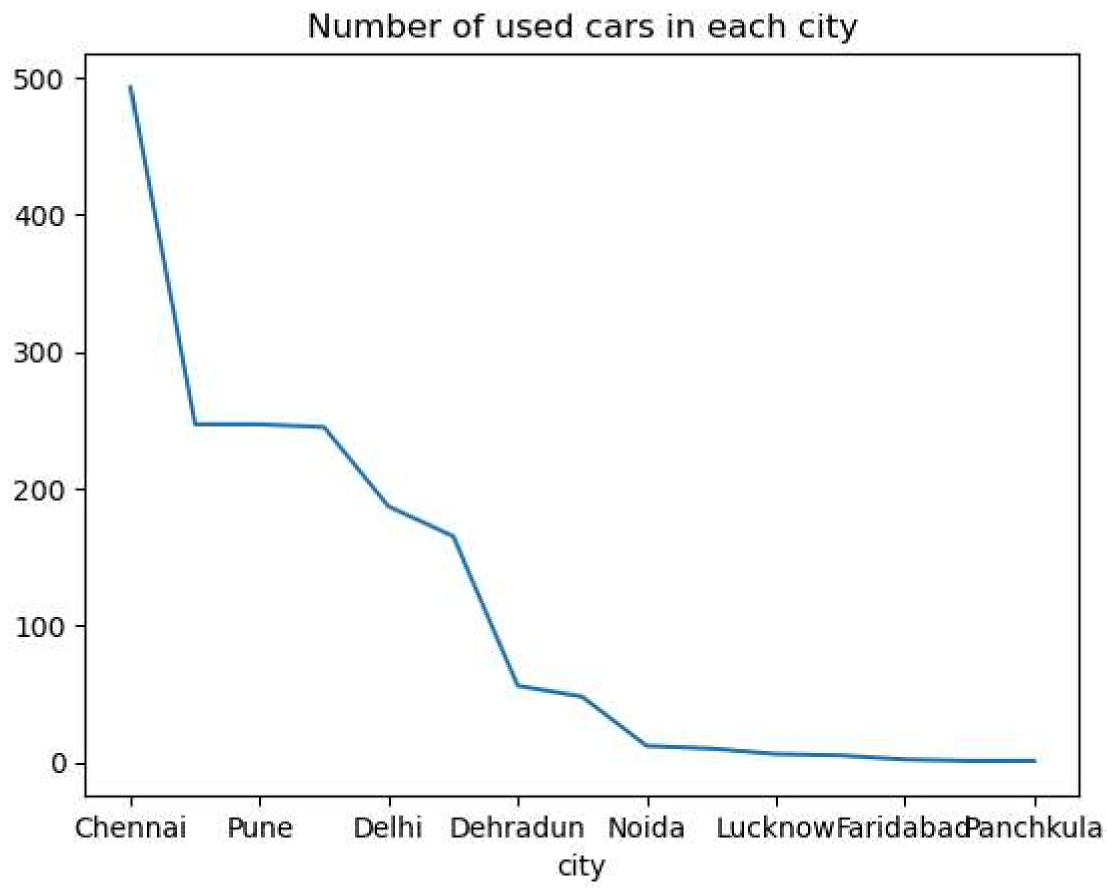
In [38]:
```
colors = ['pink', 'orange', 'silver']
df_cars6=df_cars[['brand','fuel_type']]
df_cars6.groupby('fuel_type')['brand'].count().plot(kind='pie',title='Fuel types in used cars',autopct='%1.0f%%',colo
plt.axis('equal')
plt.show()
```



In [30]:
```
#multivariate analysis. this graph shows price distrbution over different brands
df_cars2=df_cars[['brand','price']]

df_cars3=df_cars.groupby('brand')['price'].sum().sort_values(ascending=False).plot(kind='bar',title='Price distributi

plt.show()
```



In [32]:
```
#multivariate analyisis.this graph analyse the number listed used cars and their registered city
df_cars4=df_cars[['brand','city']]
df_cars5=df_cars4.groupby('city')['brand'].count().sort_values(ascending=False).plot(kind='line',title='Number of use
plt.show()
```



In [51]:
```
from sklearn.model_selection import train_test_split
import lazypredict
from lazypredict.Supervised import LazyClassifier
from sklearn.preprocessing import LabelEncoder

x=df_cars.drop(['Id','year','price'],axis=1)
y=df_cars['price']
```

In [52]:
```
x[['brand','full_model_name','model_name','fuel_type','city']]= x[['brand','full_model_name','model_name','fuel_type

x
```

Out[52]:

	brand	full_model_name	model_name	distance_travelled(kms)	fuel_type	city	brand_rank	car_age
0	7	131	28	9680.00	3	9	7	5.00
1	23	570	129	119120.00	1	9	11	9.00
2	28	676	69	64593.00	1	14	1	4.00
3	21	526	51	25000.00	1	9	2	4.00
4	8	259	144	23800.00	1	9	14	9.00
...	...	...	...	...	...	...	...	...
1720	8	221	59	38000.00	3	13	14	6.00
1721	2	80	41	36000.00	3	13	44	10.00
1722	19	391	96	142522.00	1	13	24	13.00
1723	18	347	89	18581.00	1	13	24	31.00
1724	8	201	45	31028.00	1	13	14	4.00

1725 rows × 8 columns

In [58]:
```python
from lazypredict.Supervised import LazyRegressor
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=8)
reg=LazyRegressor(verbose=0,ignore_warnings=True)

train,test=reg.fit(xtrain,xtest,ytrain,ytest)

train
```

```
100%|██████████| 42/42 [00:03<00:00, 10.76it/s]

[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.000416 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 711
[LightGBM] [Info] Number of data points in the train set: 1380, number of used features: 8
[LightGBM] [Info] Start training from score 1502441.666667
```

Out[58]:

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
XGBRegressor	0.70	0.71	922585.05	0.08
RandomForestRegressor	0.64	0.65	1015955.20	0.89
HistGradientBoostingRegressor	0.64	0.64	1018012.79	0.28
ExtraTreesRegressor	0.62	0.63	1036347.13	0.49
LGBMRegressor	0.62	0.63	1043893.91	0.06
BaggingRegressor	0.61	0.62	1058312.55	0.09
GradientBoostingRegressor	0.61	0.61	1061171.42	0.22
DecisionTreeRegressor	0.54	0.55	1144436.25	0.03
ExtraTreeRegressor	0.50	0.51	1196049.98	0.00
KNeighborsRegressor	0.43	0.44	1277589.21	0.01
PoissonRegressor	0.20	0.22	1507540.06	0.02
SGDRegressor	0.17	0.19	1542295.21	0.02
Ridge	0.16	0.18	1544285.91	0.02
LassoCV	0.16	0.18	1545436.33	0.06
LassoLarsCV	0.16	0.18	1545842.87	0.02
RidgeCV	0.16	0.18	1545938.18	0.01
Lasso	0.16	0.18	1546161.20	0.02
LassoLars	0.16	0.18	1546161.21	0.02
LarsCV	0.16	0.18	1546162.32	0.01
LassoLarsIC	0.16	0.18	1546162.32	0.02
Lars	0.16	0.18	1546162.32	0.02
LinearRegression	0.16	0.18	1546162.32	0.00
TransformedTargetRegressor	0.16	0.18	1546162.32	0.01
OrthogonalMatchingPursuitCV	0.14	0.16	1566194.37	0.02
ElasticNet	0.13	0.15	1578799.14	0.02
TweedieRegressor	0.11	0.13	1597117.60	0.03
GammaRegressor	0.10	0.12	1603446.41	0.19
RANSACRegressor	0.09	0.11	1608900.19	0.06
HuberRegressor	0.08	0.11	1615548.58	0.02
OrthogonalMatchingPursuit	0.06	0.08	1635313.28	0.00
AdaBoostRegressor	0.04	0.06	1658320.82	0.10
ElasticNetCV	-0.02	0.00	1708153.42	0.06
GaussianProcessRegressor	-0.02	-0.00	1708826.31	0.14
BayesianRidge	-0.02	-0.00	1708851.03	0.06
DummyRegressor	-0.02	-0.00	1708851.03	0.00
NuSVR	-0.05	-0.03	1729919.82	0.10
PassiveAggressiveRegressor	-0.08	-0.06	1755313.32	0.06
SVR	-0.15	-0.12	1807236.43	0.13
KernelRidge	-0.64	-0.60	2161729.83	0.05
LinearSVR	-0.77	-0.73	2249269.46	0.02
MLPRegressor	-0.78	-0.73	2249589.78	0.45

In [59]: test

Out[59]:

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken
XGBRegressor	0.70	0.71	922585.05	0.08
RandomForestRegressor	0.64	0.65	1015955.20	0.89
HistGradientBoostingRegressor	0.64	0.64	1018012.79	0.28
ExtraTreesRegressor	0.62	0.63	1036347.13	0.49
LGBMRegressor	0.62	0.63	1043893.91	0.06
BaggingRegressor	0.61	0.62	1058312.55	0.09
GradientBoostingRegressor	0.61	0.61	1061171.42	0.22
DecisionTreeRegressor	0.54	0.55	1144436.25	0.03
ExtraTreeRegressor	0.50	0.51	1196049.98	0.00
KNeighborsRegressor	0.43	0.44	1277589.21	0.01
PoissonRegressor	0.20	0.22	1507540.06	0.02
SGDRegressor	0.17	0.19	1542295.21	0.02
Ridge	0.16	0.18	1544285.91	0.02
LassoCV	0.16	0.18	1545436.33	0.06
LassoLarsCV	0.16	0.18	1545842.87	0.02
RidgeCV	0.16	0.18	1545938.18	0.01
Lasso	0.16	0.18	1546161.20	0.02
LassoLars	0.16	0.18	1546161.21	0.02
LarsCV	0.16	0.18	1546162.32	0.01
LassoLarsIC	0.16	0.18	1546162.32	0.02
Lars	0.16	0.18	1546162.32	0.02
LinearRegression	0.16	0.18	1546162.32	0.00
TransformedTargetRegressor	0.16	0.18	1546162.32	0.01
OrthogonalMatchingPursuitCV	0.14	0.16	1566194.37	0.02
ElasticNet	0.13	0.15	1578799.14	0.02
TweedieRegressor	0.11	0.13	1597117.60	0.03
GammaRegressor	0.10	0.12	1603446.41	0.19
RANSACRegressor	0.09	0.11	1608900.19	0.06
HuberRegressor	0.08	0.11	1615548.58	0.02
OrthogonalMatchingPursuit	0.06	0.08	1635313.28	0.00
AdaBoostRegressor	0.04	0.06	1658320.82	0.10
ElasticNetCV	-0.02	0.00	1708153.42	0.06
GaussianProcessRegressor	-0.02	-0.00	1708826.31	0.14
BayesianRidge	-0.02	-0.00	1708851.03	0.06
DummyRegressor	-0.02	-0.00	1708851.03	0.00
NuSVR	-0.05	-0.03	1729919.82	0.10
PassiveAggressiveRegressor	-0.08	-0.06	1755313.32	0.06
SVR	-0.15	-0.12	1807236.43	0.13
KernelRidge	-0.64	-0.60	2161729.83	0.05
LinearSVR	-0.77	-0.73	2249269.46	0.02
MLPRegressor	-0.78	-0.73	2249589.78	0.45

In [77]:
```python
import xgboost as xg

reg=xg.XGBRegressor(n_estimators=1000,max_depth=6,radom_state=42)

reg.fit(xtrain,ytrain)
```

Out[77]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, device=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=6, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             multi_strategy=None, n_estimators=1000, n_jobs=None,
             num_parallel_tree=None, radom_state=42, ...)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [84]:

```
from sklearn.metrics import mean_squared_error
from math import sqrt
ypred=reg.predict(xtest)

ypred
```

Out[84]:

```
array([ 818354.25, 1526461.9 , 1527604.2 ,  237504.8 ,  405434.2 ,
         625000.9 , 1090000.2 ,  726107.56, 1174999.6 ,  311177.97,
         741461.7 , 1199997.8 , 1169013.4 ,  679997.6 ,  989999.7 ,
        4850000.5 , 1989997.6 ,  836969.7 ,  291074.94, 2453157.2 ,
         654983.1 , 1854407.1 ,  680155.4 , 5998366.5 ,  949999.1 ,
         869481.44,  602838.56,  138141.78, 1025007.44, 1699999.5 ,
         214090.27, 2749993.5 ,  870004.9 , 1999999.1 ,  689993.75,
         908885.3 ,  345999.66, 4200001.  ,  625000.1 ,  973866.6 ,
         748949.25,  201832.55,  308753.78, 1689896.2 , 3724170.5 ,
        1000296.44,  725004.44,  559115.94, 3812604.8 ,  698999.5 ,
        1090006.2 ,  784998.3 ,  506306.8 , 2331038.  , 3486879.2 ,
        3104939.2 , 1115595.4 ,  878885.3 ,  717376.2 , 2688431.2 ,
         111052.99,  484996.2 , 1526461.9 , 1462482.  , 2074997.4 ,
        2578088.8 ,  885834.25, 5414387.5 , 1289999.  ,  525002.94,
         618206.2 ,  600001.56,  996564.8 ,  361907.25, 5255870.5 ,
        6800001.  ,  460002.2 , 1780892.1 ,  695000.5 ,  563641.94,
         239999.48,  924999.5 , 2350749.  , 3115871.5 ,  515220.78,
         958127.1 ,  926890.3 , 2699999.  ,  442487.3 , 2348275.  ,
        1450000.5 , 1200000.  ,  787463.25, 2779150.  ,  170999.31,
         732154.5 , 1400002.1 ,  798460.94,  705959.44, 1099654.2 ,
        1789738.5 ,  500000.06, 2399999.2 , 5200001.  , 1449999.5 ,
         559115.94,  974313.1 ,  925001.56,  233716.45, 2400000.8 ,
         968595.75, 2765505.8 ,  904700.8 ,  958127.1 ,  944389.1 ,
         460000.06, 2091159.9 ,  491719.2 ,  506183.53,  346689.44,
         438379.5 ,  950004.4 ,  451103.22, 1578651.2 , 3200001.2 ,
         835005.8 ,  390003.22, 1290000.6 ,  690265.44,  526944.25,
         638640.4 , 2580930.5 ,  664104.9 ,  870000.75, 2119537.  ,
        1730143.6 , 1115313.1 ,  529487.25, 1650000.4 , 1331739.1 ,
        5850006.5 , 1729094.9 ,  460002.2 ,  836569.56,  990000.4 ,
        5299999.5 , 3135965.5 , 1008628.94,  580534.6 ,  870226.56,
         898267.9 , 3895818.  ,  371158.75, 1741424.4 ,  469999.6 ,
        2839125.2 , 1117522.4 ,  550003.8 ,  564573.5 ,  610002.06,
         711607.5 , 4699999.5 ,  874998.44, 1135232.1 , 1506225.9 ,
        1800000.4 , 1089999.6 , 1739260.1 ,  291544.8 ,  909994.3 ,
        4205905.  ,  663014.5 , 1396220.9 ,  911308.  ,  515681.75,
        1488103.9 ,  245002.78,  309999.94, 3404644.  , 3850000.2 ,
        1825700.5 , 1450000.5 , 5909099.5 ,  613409.94,  478714.6 ,
         439801.94,  749992.25,  563641.94,  809999.7 , 2790000.2 ,
        1271116.2 ,  334999.7 , 1376049.1 , 5470662.  ,  874998.44,
         825006.06,  592354.44,  379999.47, 3125516.2 , 3681168.5 ,
         469459.56, 1086801.9 ,  544999.9 ,  749999.44, 1091550.8 ,
        2300000.5 , 1249991.9 , 2040010.8 ,  570295.06,  953121.44,
        1079410.5 , 5862420.5 , 5014558.5 ,  750000.3 ,  874999.25,
        2900000.  ,  740395.44, 2955654.5 , 1249998.4 ,  630002.2 ,
         417456.7 ,  596075.  , 1389999.8 , 2069412.  ,  464998.38,
         591759.  , 1149995.6 , 1194465.2 , 1608804.2 ,  374999.56,
         799999.4 ,  635346.  , 2861904.8 , 3724170.5 ,   80000.86,
         299121.66,  289897.  , 2099398.2 , 1292019.4 ,  366030.44,
        3699997.2 , 1897268.8 ,  698999.9 ,  698999.9 ,  790003.4 ,
        2319750.8 , 1073051.  ,  833455.25, 1610299.2 ,  941779.3 ,
         579999.75,  631860.44,  535766.1 , 1154337.5 , 1343692.9 ,
        2200000.2 ,  977395.4 ,  214090.27,  689993.06, 7199998.5 ,
         685001.4 ,  634999.4 ,  349999.25, 1205199.8 ,  550000.5 ,
        1202444.8 ,  649995.94,  349350.97,  850501.1 ,  838244.5 ,
        2799997.5 ,  451387.2 , 2400000.5 ,  796532.75,  621295.06,
        1501353.4 ,  331947.75,  303290.7 ,  709815.06, 3104939.2 ,
         475065.53, 2649999.2 ,  697293.9 , 1251593.4 , 4977142.5 ,
        1699998.5 , 1352849.2 ,  722398.4 ,  975275.1 ,  610001.06,
        3443616.  , 1240573.2 ,  675002.3 , 4035058.8 ,  598997.94,
        4965834.  ,  276617.38,  792979.6 , 1292019.4 , 2372835.5 ,
         309999.9 , 1024959.5 ,  294009.47,  337646.97, 1244566.1 ,
         599998.6 ,  145830.34, 1506225.9 ,  444618.28,  135431.58,
        4799995.5 , 1123516.1 , 2373877.5 ,  840020.  ,  575006.4 ,
        1982142.5 ,  745000.3 , 1232700.1 ,  679999.6 ,  825003.75,
        2250000.2 ,  649995.94,  520000.66, 1897268.8 ,  541705.6 ,
        1250000.1 ,  319999.62,  989999.7 ,  412788.06, 2365365.5 ,
         615164.25,  968202.3 ,  239999.7 , 4499998.5 ,  912224.3 ,
        4754160.  ,  789999.6 , 2714745.2 ,  815001.1 , 1126621.6 ,
         953121.44, 1707775.9 , 1850000.6 ,  590000.  ,  598923.  ],
      dtype=float32)
```

In [85]:

```
mse=mean_squared_error(ypred,ytest)
rmse=sqrt(mse)

print(mse)
print(rmse)
```

```
842185244052.5292
917706.5130271928
```

In [79]:
```
feature_imp=reg.feature_importances_
feature_imp
```

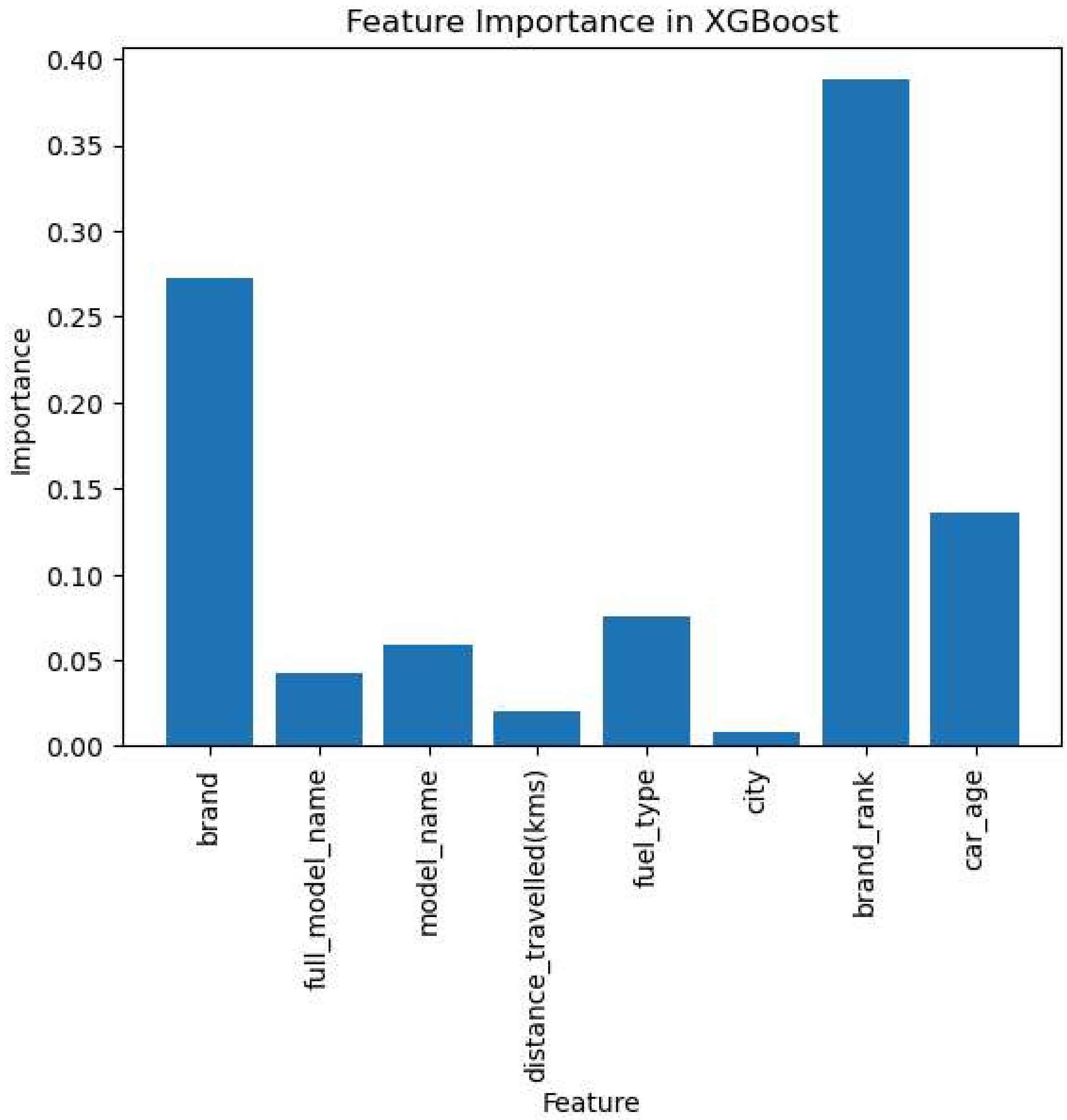
Out[79]:
```
array([0.2719252 , 0.04212234, 0.05898914, 0.02034225, 0.07504267,
       0.007917  , 0.3874883 , 0.13617304], dtype=float32)
```

In [83]:
```
#fetching features that predict price of listed cars
feature_imp_pd=pd.DataFrame({'Feature':xtrain.columns,'Importances':feature_imp})

feature_imp_pd
```

Out[83]:

	Feature	Importances
0	brand	0.27
1	full_model_name	0.04
2	model_name	0.06
3	distance_travelled(kms)	0.02
4	fuel_type	0.08
5	city	0.01
6	brand_rank	0.39
7	car_age	0.14

In [87]:
```
#plotting best features that predict price of listed cars
plt.bar(range(len(feature_imp)), feature_imp)
plt.xticks(range(len(feature_imp)), xtrain.columns, rotation=90)
plt.xlabel("Feature")
plt.ylabel("Importance")
plt.title("Feature Importance in XGBoost")
plt.show()
```



Conclusion: Brand_rank and brand of listed cars plays crucial part in predicting price listed cars.