car price prediction project

In [49]:
```
import pandas as pd  # make data frame ARE IN STRUCTURED TABULAR FORM #EASIER TO ANLAYISE AND DO PROCESSSING ON DATASET #library
import matplotlib.pyplot as plt # used for ploting or draw a line or ploting a graph #data visualisation #library
import seaborn as sns #vislise random distribuition or library use in #library
from sklearn.model_selection import train_test_split #sk learn  IS Library split original data into train data and test data this function is useful sor this #
from sklearn.linear_model import LinearRegression # algorthim or statical method for predict anlaysis relationship between the data-points to draw a straight l
from sklearn.linear_model import Lasso # modifies the loss function by adding the penalty (shrinkage quantity) equivalent to the summation of the absolute valu
from sklearn import metrics #evalute our model by find error score accuracies etc
```

In [16]:
```
#data collectiopn and processing
#loading the data from car file to pandas dataframe
car_dataset=pd.read_csv("car data.csv")#pandas has a  function read_csv loading csv file to a data frame
```

In [13]:
```
car_dataset.head() # head functionprint the first 5 rows of the data #cardataset is the name of data frame
```

Out[13]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

In [17]:
```
# checking the no of row and coloum by function shape
car_dataset.shape
```

Out[17]:
```
(301, 9)
```

In [18]:
```
#getting some information of dataset opr dataframe
car_dataset.info() #object means categories of data petrol diesal or cng in datatype
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Car_Name       301 non-null    object 
 1   Year           301 non-null    int64  
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Driven_kms     301 non-null    int64  
 5   Fuel_Type      301 non-null    object 
 6   Selling_type   301 non-null    object 
 7   Transmission   301 non-null    object 
 8   Owner          301 non-null    int64  
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

In [19]:
```
#checking the no of missing values
car_dataset.isnull().sum()#how many missingh values of each column
```

Out[19]:
```
Car_Name         0
Year             0
Selling_Price    0
Present_Price    0
Driven_kms       0
Fuel_Type        0
Selling_type     0
Transmission     0
Owner            0
dtype: int64
```

In [24]:
```
#checking the distribution of categorical data
print(car_dataset.Fuel_Type.value_counts())#value counts count or no of the for petrol and diesal
print(car_dataset.Selling_type.value_counts())
print(car_dataset.Transmission.value_counts())
```

```
Petrol    239
Diesel     60
CNG         2
Name: Fuel_Type, dtype: int64
Dealer        195
Individual    106
Name: Selling_type, dtype: int64
Manual       261
Automatic     40
Name: Transmission, dtype: int64
```

In [35]:
```
#ml model cannot understand text so that converted into numerical values
#encoding the categorical data fuel type coloumn
car_dataset.replace({"Fuel_Type":{'Petrol':0, 'Diesel':1,'CNG':2}},inplace=True)#for changing the value use inplace parameters
#encoding the categorical data fuel type coloumn
car_dataset.replace({"Selling_type":{'Dealer':0,'Individual':1}},inplace=True)
#encoding the categorical data fuel type coloumn
car_dataset.replace({"Transmission":{'Manual':0,'Automatic':1}},inplace=True)
```

In [36]:
```
car_dataset.head()
```

Out[36]:

	Car_Name	Year	Selling_Price	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	0	0	0	0
1	sx4	2013	4.75	9.54	43000	1	0	0	0
2	ciaz	2017	7.25	9.85	6900	0	0	0	0
3	wagon r	2011	2.85	4.15	5200	0	0	0	0
4	swift	2014	4.60	6.87	42450	1	0	0	0

In [41]:
```
#spliting the data and target data
#selling price target and othr features becomes the data
#remove selling price and store it into y all the remaing features were come in X variable REMOVE CAR NAME BECAUSE IT CANNOT BE USED IN THE PREDICTION
X=car_dataset.drop(['Car_Name','Selling_Price'],axis=1) #droping a coloumn axis 1 dropping a row axis 0
Y=car_dataset['Selling_Price']
```

In [39]:
```
print(X)
```

```
     Year  Present_Price  Driven_kms  Fuel_Type  Selling_type  Transmission  \
0    2014           5.59       27000          0             0             0   
1    2013           9.54       43000          1             0             0   
2    2017           9.85        6900          0             0             0   
3    2011           4.15        5200          0             0             0   
4    2014           6.87       42450          1             0             0   
..    ...            ...         ...        ...           ...           ...   
296  2016          11.60       33988          1             0             0   
297  2015           5.90       60000          0             0             0   
298  2009          11.00       87934          0             0             0   
299  2017          12.50        9000          1             0             0   
300  2016           5.90        5464          0             0             0   

     Owner  
0        0  
1        0  
2        0  
3        0  
4        0  
..     ...  
296      0  
297      0  
298      0  
299      0  
300      0  

[301 rows x 7 columns]
```

In [43]:
```
print(Y) #3.35 means 3 lakh 35000 indian rupees# target variable orprice of the car which we want to analyse
```

```
0       3.35
1       4.75
2       7.25
3       2.85
4       4.60
       ...  
296     9.50
297     4.00
298     3.35
299    11.50
300     5.30
Name: Selling_Price, Length: 301, dtype: float64
```

In [45]:
```
#spliting traing and test data #seperate data into training data and test data #traing data in the variable x train test data in the variable x test # thenpric
X_train,X_test,Y_train,Y_test = train_test_split(X,Y ,test_size= 0.1, random_state=2)#10 %testdata 90% training data# dataset large prediction will be better
```

In [77]:
```
X_train
```

Out[77]:

	Year	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
204	2015	4.430	28282	0	0	0	0
249	2016	7.600	17000	0	0	0	0
277	2015	13.600	21780	0	0	0	0
194	2008	0.787	50000	0	1	0	0
244	2013	9.400	49000	1	0	0	0
...	...	...	...	...	...	...	...
75	2015	6.800	36000	0	0	0	0
22	2011	8.010	50000	0	0	1	0
72	2013	18.610	56001	0	0	0	0
15	2016	10.790	43000	1	0	0	0
168	2013	0.730	12000	0	1	0	0

270 rows × 7 columns

In [78]:
```
Y_train
```

Out[78]:
```
204    2.75
249    5.25
277    9.70
194    0.20
244    5.95
       ... 
75     3.95
22     4.40
72     7.45
15     7.75
168    0.42
Name: Selling_Price, Length: 270, dtype: float64
```

In [79]:
```
X_test
```

Out[79]:

	Year	Present_Price	Driven_kms	Fuel_Type	Selling_type	Transmission	Owner
99	2010	20.450	50024	1	0	0	0
161	2014	0.826	23000	0	1	0	0
89	2014	6.760	40000	0	0	0	0
30	2012	5.980	51439	1	0	0	0
232	2015	14.790	12900	0	0	1	0
290	2014	6.400	19000	0	0	0	0
35	2011	7.740	49998	2	0	0	0
7	2015	8.610	33429	1	0	0	0
183	2013	0.470	21000	0	1	0	0
13	2015	7.710	26000	0	0	0	0
269	2015	10.000	18828	0	0	0	0
65	2014	6.950	45000	1	0	0	0
178	2014	0.520	19000	0	1	1	0
258	2015	13.600	25000	0	0	0	0
227	2011	4.430	57000	0	0	0	0
133	2016	0.950	500	0	1	0	0
130	2017	0.870	11000	0	1	0	0
156	2017	0.520	15000	0	1	0	0
237	2015	13.600	68000	1	0	0	0
262	2015	5.800	40023	0	0	0	0
112	2014	2.400	7000	0	1	0	0
282	2014	14.000	63000	1	0	0	0
164	2016	0.540	14000	0	1	0	0
275	2016	13.600	30753	0	0	1	0
154	2014	0.880	8000	0	1	0	0
29	2015	10.380	45000	1	0	0	0
141	2016	0.800	20000	0	1	0	0
192	2007	0.750	49000	0	1	0	1
216	2016	4.430	12500	0	0	0	0
3	2011	4.150	5200	0	0	0	0
159	2017	0.510	4000	0	1	1	0

In [80]:
```
Y_test
```

Out[80]:
```
99      9.65
161     0.45
89      4.75
30      3.10
232    11.45
290     4.50
35      2.95
7       6.50
183     0.27
13      6.10
269     6.70
65      4.75
178     0.35
258     8.40
227     2.55
133     0.72
130     0.75
156     0.48
237    11.25
262     4.00
112     1.15
282     8.25
164     0.45
275    10.90
154     0.50
29      7.45
141     0.60
192     0.20
216     2.90
3       2.85
159     0.45
Name: Selling_Price, dtype: float64
```

In [50]:
```
#model traing
#linear regreesion model loading to a variable called lin_rig_model
lin_reg_model = LinearRegression()
```

In [ ]:
```
#fit our data to the linear regression model
```

In [51]:
```
#fit fuction is used to train our model#linear regression is just a line equ y=mx+c
lin_reg_model.fit(X_train,Y_train)
```

Out[51]:
```
LinearRegression()
```

In [52]:
```
#evalute our model
#prediction on training data by using a function.predict# predict the price value baesd on other data we are giving
training_data_prediction =lin_reg_model.predict(X_train)
```

In [81]:
```
lin_reg_model.predict(X_train)
```

Out[81]:
```
array([ 3.73021336e+00,  5.60557769e+00,  7.79651028e+00, -1.87998528e+00,
        6.71732485e+00,  8.14298867e+00,  5.83860743e+00,  6.17807702e+00,
        6.16007875e+00,  1.59435570e+01,  2.15178639e+00,  1.11542236e+00,
       -4.03566158e-01,  6.69018458e+00,  6.81730001e+00,  1.02823074e+00,
        6.74434076e+00,  1.43840178e+01,  2.99972885e+00,  7.95100696e+00,
       -8.35827371e-01,  9.01998153e+00,  1.21122729e+00,  2.19835982e+00,
       -8.22859063e-01,  7.52654580e-01,  1.00169815e+01, -4.06906872e-01,
       -2.39970547e+00,  9.99810395e+00,  6.81635445e+00,  1.09134689e+00,
        4.36005070e+00,  9.35912908e-02,  3.49554774e+00,  2.78126418e+00,
        1.50310108e+00,  4.66039889e-01,  1.68540670e+00,  1.18623031e+00,
        4.32915189e+00,  6.09991944e+00,  9.49841057e-01,  7.57239833e-01,
        6.06044794e+00,  3.49340827e+00,  2.40754453e+00,  7.52178300e+00,
        4.63218039e+00,  4.44151068e+00,  4.22818099e+00,  2.71097979e+00,
        4.46834250e+00,  1.75948281e+00,  2.86122534e-01,  3.95048847e+00,
        1.36655084e+00,  4.63150500e+00,  1.44231527e+00,  7.27805289e+00,
        7.26394322e+00,  6.54781623e+00,  2.36094207e+00,  2.19195632e+01,
        2.16577319e+00,  5.65903467e+00,  2.81121561e+00,  7.49954464e+00,
        5.62852385e+00,  9.98192042e+00,  2.06603262e+00,  2.39715724e+00,
        1.25461894e+00,  2.48415503e+00,  9.87386064e+00,  2.00009723e+00,
       -5.96174924e-01,  5.43130373e+00,  1.14777240e+01,  7.40627935e-02,
        1.16911287e+00,  1.59431096e+00,  5.41288483e+00,  1.41364975e+00,
        7.74517296e+00,  1.84074211e+01,  2.57930632e+00,  2.07587412e-01,
        1.89596397e+00,  9.26619285e+00,  3.98838512e+00,  7.06940463e+00,
        1.68505004e+01,  1.02708461e+01,  3.68253268e+00,  1.49972409e+00,
        4.31163494e+00, -4.34395723e-01,  1.71198073e+00,  8.27811181e+00,
        1.14570644e+00,  7.50185460e+00, -2.04820283e+00,  8.29860671e-01,
        7.27052964e+00,  8.01752980e+00,  7.36889110e+00, -3.78843619e+00,
        3.75727695e+00,  8.25767071e+00,  3.51367020e+00,  2.53512361e+00,
        4.16080008e+00, -3.23491702e+00,  5.18526238e+00,  1.86200542e+00,
        3.49406867e+00,  7.72953074e+00, -2.58888366e-02,  1.85445965e+00,
        1.61626911e+00,  1.86613865e+00,  3.44271519e+00, -2.60911111e+00,
        2.84248698e-01,  6.46837999e+00,  6.81925914e+00,  1.27770682e+00,
        1.98097704e+00,  7.77611326e+00,  3.98834359e+00,  7.12451285e-01,
        7.91361269e+00,  6.23437252e+00,  1.29303095e+00, -1.96375497e+00,
        5.28380366e+00,  6.04429610e+00,  5.15977723e+00,  4.22857827e+00,
        9.52747826e-01,  7.35206032e+00,  4.02067824e+00, -1.04667913e+00,
        1.09170167e+01,  4.39638274e+00,  2.86984160e+00,  1.91816504e+00,
        4.96758029e+00,  4.77821450e+00,  4.24986005e+00,  1.71038563e+01,
       -1.06935542e+00,  1.26393734e+00,  5.21116574e+00,  2.12203362e+00,
        5.99026427e+00,  3.41070624e+00,  4.80690714e+00,  3.39434249e+00,
        5.17639363e-01,  4.44072086e+00,  2.00888667e+01,  3.83904260e+00,
        4.52782361e+00,  2.19493306e+00,  2.97801507e+00,  1.63374244e-01,
        7.45102096e+00, -9.61785447e-01,  2.07068986e+01,  1.55772760e-01,
        7.67126450e+00,  1.96921097e+00,  1.58138092e+00,  8.59354216e+00,
        2.03387639e+01,  3.12357420e+00,  4.91724524e+00,  5.12034777e+00,
        4.44797076e+00,  6.32517690e+00,  6.14304094e+00,  1.63199948e+00,
        4.86541580e-01,  7.15884665e+00,  8.19381235e+00,  2.20908559e+00,
       -2.99877309e+00,  4.28784743e+00,  4.26152494e+00,  5.13695581e+00,
       -3.39727342e-01,  6.40082386e+00,  4.40350143e+00,  8.19915104e+00,
        8.08355349e-01,  1.09129446e+01,  5.67311748e+00, -9.81370968e-01,
        6.42130914e+00,  5.08147086e+00,  6.91552670e+00,  4.93779362e+00,
        5.77603798e+00,  2.12511905e+00,  2.52819990e+00,  4.89321786e+00,
        3.15523099e+00, -1.52287748e+00,  4.17908194e+01,  5.81937081e+00,
        5.14264713e+00,  3.92942807e-01,  9.05374467e+00,  6.56521387e+00,
        7.18150970e+00,  1.66208008e+00,  7.11374587e+00,  9.07843813e+00,
        4.46727937e+00,  3.36316142e+00,  6.35570960e+00,  6.55994577e+00,
        1.66006420e+01,  9.38983794e+00,  6.26565026e+00, -1.37604038e-01,
        2.55764075e+00,  2.97586269e+00,  5.85815608e+00,  5.06731925e+00,
        9.32107270e-01,  8.12878274e+00,  5.75160599e+00,  7.71397666e+00,
        2.18350107e+00,  1.67444157e+01,  4.87711576e+00,  1.46386665e+01,
        1.05270698e+01,  2.12230658e+00, -5.01586122e-02,  9.93585366e+00,
       -3.52807959e+00,  3.49708155e+00,  2.73628808e+00,  3.41048924e+00,
        7.39425455e+00,  3.96792187e+00,  1.84074211e+01,  8.15489898e+00,
        2.54420014e+00,  8.77086720e-01,  6.36086778e-01,  5.77720292e-01,
        4.37093114e-01,  4.94953904e+00,  6.47901168e+00, -3.18527138e-01,
        3.10834662e+00,  9.00971216e+00,  8.42896554e+00,  8.34603228e-02,
        5.65618275e+00,  4.71767684e+00,  5.02756294e+00,  8.94568101e+00,
        8.59354216e+00,  3.94955908e-01])
```

In [82]:
```
Y_test
```

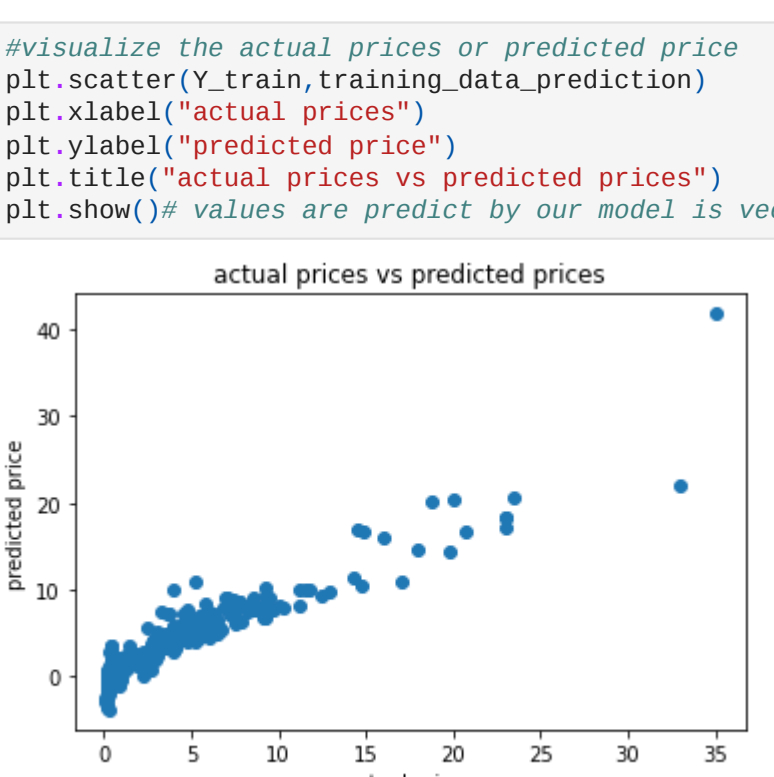
Out[82]:
```
99      9.65
161     0.45
89      4.75
30      3.10
232    11.45
290     4.50
35      2.95
7       6.50
183     0.27
13      6.10
269     6.70
65      4.75
178     0.35
258     8.40
227     2.55
133     0.72
130     0.75
156     0.48
237    11.25
262     4.00
112     1.15
282     8.25
164     0.45
275    10.90
154     0.50
29      7.45
141     0.60
192     0.20
216     2.90
3       2.85
159     0.45
Name: Selling_Price, dtype: float64
```

In [ ]:

In [54]:
```
# compare value predicted our model by original value r squaed error
error_score=metrics.r2_score(Y_train,training_data_prediction)
print('R squared Error:',error_score)
```

```
R squared Error: 0.8796483009370206
```

In [57]:
```
#visualize the actual prices or predicted price
plt.scatter(Y_train,training_data_prediction)
plt.xlabel("actual prices")
plt.ylabel("predicted price")
plt.title("actual prices vs predicted prices")
plt.show()# values are predict by our model is veery close as sold price#estamiationg model performance by plotting the original values in cases of scatter plo
```

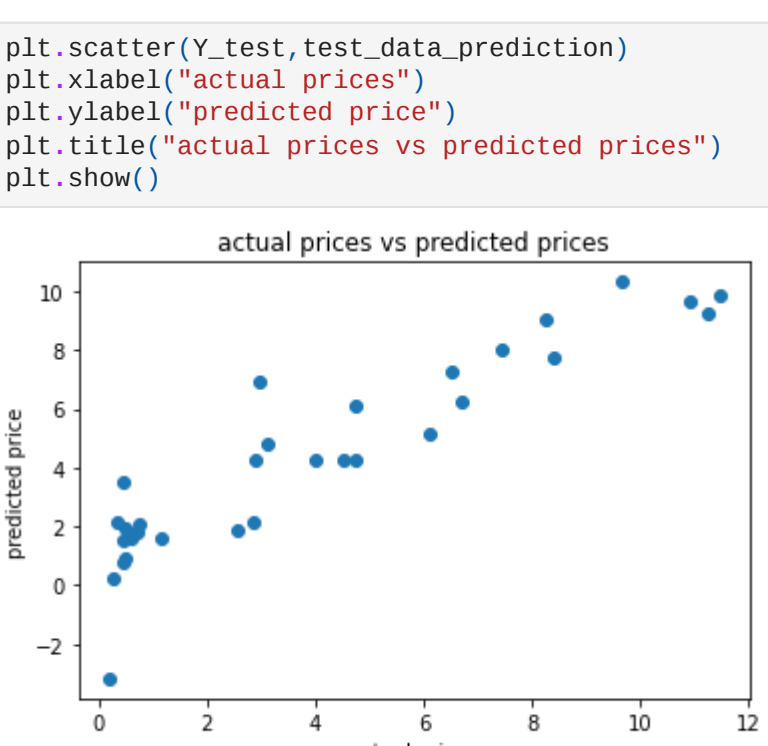


In [60]:
```
#prediction on test data by using a function.predict
test_data_prediction =lin_reg_model.predict(X_test)
```

In [61]:
```
#compare value predicted our model by original value r squaed error
error_score=metrics.r2_score(Y_test,test_data_prediction)
print('R squared Error:',error_score)
```

```
R squared Error: 0.8365861023208354
```

In [62]:
```
plt.scatter(Y_test,test_data_prediction)
plt.xlabel("actual prices")
plt.ylabel("predicted price")
plt.title("actual prices vs predicted prices")
plt.show()
```



In [ ]:
```
#linear regresion perform well in those caese were one value increase other also increase
#when there is multe features and multiple column then we will use lasso and other regression
```

In [64]:
```
#model traing
lasso_reg_model = Lasso()
```

In [65]:
```
#fit our data to the linear regression model
```

In [66]:
```
lasso_reg_model.fit(X_train,Y_train
```

Out[66]:
```
Lasso()
```

In [67]:
```
#evalute our model
#prediction on training data by using a function.predict# predict the price value baesd on other data we are giving
training_data_prediction =lasso_reg_model.predict(X_train)
```

In [68]:
```
# compare value predicted our model by original value r squaed error
error_score=metrics.r2_score(Y_train,training_data_prediction)
print('R squared Error:',error_score)
```

```
R squared Error: 0.8424480718240743
```

In [69]:
```
#visualize the actual prices or predicted price
plt.scatter(Y_train,training_data_prediction)
plt.xlabel("actual prices")
plt.ylabel("predicted price")
plt.title("actual prices vs predicted prices")
plt.show()# values are predict by our model is veery close as sold price#estamiationg model performance by plotting the original values in cases of scatter plo
```

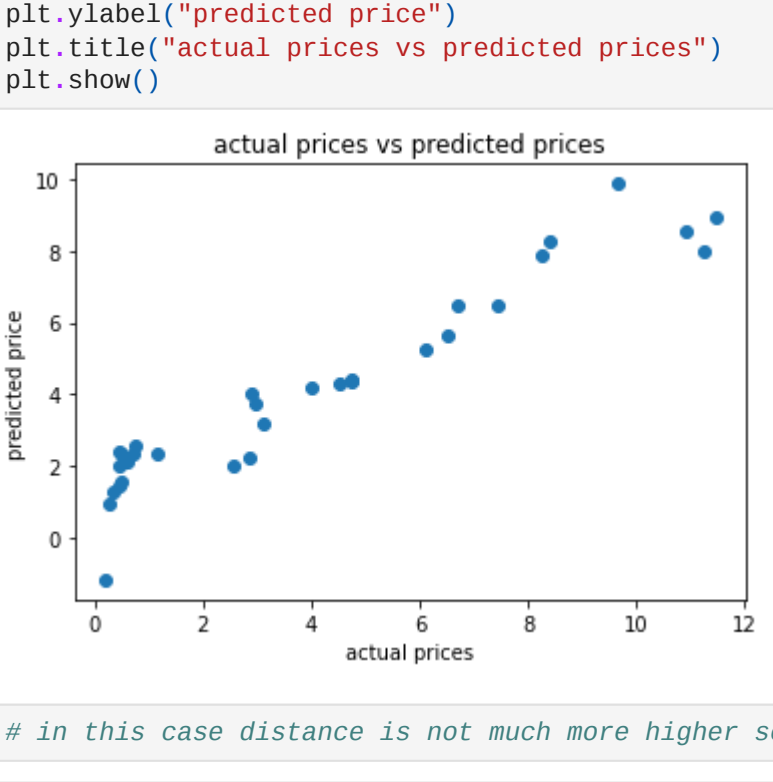
In [ ]:
```
#in lasso regreesion value is much more closer thenlinear regression so lasso is much more good phir for this particular project
```

In [73]:
```
#prediction on test data by using a function.predict
test_data_prediction =lasso_reg_model.predict(X_test)
```

In [74]:
```
#compare value predicted our model by original value r squaed error
error_score=metrics.r2_score(Y_test,test_data_prediction)
print('R squared Error:',error_score)
```

```
R squared Error: 0.8709763132343395
```

In [76]:
```
plt.scatter(Y_test,test_data_prediction)
plt.xlabel("actual prices")
plt.ylabel("predicted price")
plt.title("actual prices vs predicted prices")
plt.show()
```



In [ ]:
```
# in this case distance is not much more higher so lasso in testing is also god fit for this project
```

In [ ]: