yxq76abae

January 31, 2024

Regression is a type of supervised machine learning technique used for predicting a continuous target variable based on one or more input features. The goal of regression is to find a mathematical relationship (model) that maps the input features to the target variable. There are several regression algorithms you can use, depending on the nature of your data and the problem you are trying to solve. Here's an overview of some common regression techniques:

1. **Linear Regression**:
   - Simple linear regression models the relationship between a single input feature and the target variable as a straight line.
   - Multiple linear regression extends this to multiple input features.
   - The model assumes a linear relationship between the features and the target.
2. **Support Vector Regression (SVR)**:

- SVR is a regression technique based on support vector machines.
- It seeks to find a hyperplane that best fits the data, allowing a margin of error.

3. K-Nearest Neighbors (KNN) can be used for regression tasks as well as classification. When applied to regression, it's often referred to as "K-Nearest Neighbors Regression" or "KNN Regression." Instead of predicting a class label, KNN regression predicts a continuous target variabl

4. **Polynomial Regression**:

   - Polynomial regression extends linear regression by fitting a polynomial equation to the data, allowing it to capture more complex relationships.
   - It can model curved relationships, which a simple linear regression cannot.

5. **Decision Tree Regression**:

   - Decision tree regression uses decision trees to partition the data into regions and predict the target variable based on the average of data points in each region.

[2]:
```
#import library
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[3]:
```
# Data To Dataframe
df=pd.read_csv("CarPrice_Assignment.csv")
```

1 Exploratory data analysis Python

[4]:
```
# Shallow copy
df_copy=df.copy
```

[5]:
```
# The shape of a DataFrame.
df.shape
```

[5]:
```
(205, 26)
```

[6]:
```
# column labels of the Dataframe
df.columns
```

[6]:
```
Index(['car_ID', 'symboling', 'CarName', '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')
```

[7]:
```
# Returns a specified number of rows, string from the top.
df.head()
```

[7]:

	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]
```

[8]:
```
# Return the last 5 rows
df.tail()
```

[8]:
```
     car_ID  symboling          CarName fueltype aspiration doornumber  \
200     201         -1  volvo 145e (sw)      gas        std       four
201     202         -1      volvo 144ea      gas      turbo       four
202     203         -1      volvo 244dl      gas        std       four
203     204         -1        volvo 246   diesel      turbo       four
204     205         -1      volvo 264gl      gas      turbo       four

    carbody drivewheel enginelocation  wheelbase  …  enginesize  fuelsystem  \
200   sedan        rwd          front      109.1  …         141        mpfi
201   sedan        rwd          front      109.1  …         141        mpfi
202   sedan        rwd          front      109.1  …         173        mpfi
203   sedan        rwd          front      109.1  …         145         idi
204   sedan        rwd          front      109.1  …         141        mpfi

     boreratio  stroke compressionratio horsepower  peakrpm citympg  \
200       3.78    3.15              9.5        114     5400      23
201       3.78    3.15              8.7        160     5300      19
202       3.58    2.87              8.8        134     5500      18
203       3.01    3.40             23.0        106     4800      26
204       3.78    3.15              9.5        114     5400      19

     highwaympg    price
200          28  16845.0
201          25  19045.0
202          23  21485.0
203          27  22470.0
204          25  22625.0

[5 rows x 26 columns]
```

[9]:
```
# specific rows of a DataFrame ( "integer location" Method)
df.iloc[100:200]
```

[9]:
```
     car_ID  symboling       CarName fueltype aspiration doornumber  \
100     101          0  nissan nv200      gas        std       four
101     102          0   nissan dayz      gas        std       four
102     103          0   nissan fuga      gas        std       four
```

```
103     104          0   nissan otti      gas        std       four
104     105          3  nissan teana      gas        std        two
..      …         …           …      …        …        …
195     196         -1   volvo 144ea      gas        std       four
196     197         -2   volvo 244dl      gas        std       four
197     198         -1     volvo 245      gas        std       four
198     199         -2   volvo 264gl      gas      turbo       four
199     200         -1  volvo diesel      gas      turbo       four

       carbody drivewheel enginelocation  wheelbase  …  enginesize  \
100      sedan        fwd          front       97.2  …         120
101      sedan        fwd          front      100.4  …         181
102      wagon        fwd          front      100.4  …         181
103      sedan        fwd          front      100.4  …         181
104  hatchback        rwd          front       91.3  …         181
..         …        …            …        …  …          …
195      wagon        rwd          front      104.3  …         141
196      sedan        rwd          front      104.3  …         141
197      wagon        rwd          front      104.3  …         141
198      sedan        rwd          front      104.3  …         130
199      wagon        rwd          front      104.3  …         130

     fuelsystem  boreratio  stroke compressionratio horsepower  peakrpm  \
100        2bbl       3.33    3.47              8.5         97     5200
101        mpfi       3.43    3.27              9.0        152     5200
102        mpfi       3.43    3.27              9.0        152     5200
103        mpfi       3.43    3.27              9.0        152     5200
104        mpfi       3.43    3.27              9.0        160     5200
..          …        …     …               …        …      …
195        mpfi       3.78    3.15              9.5        114     5400
196        mpfi       3.78    3.15              9.5        114     5400
197        mpfi       3.78    3.15              9.5        114     5400
198        mpfi       3.62    3.15              7.5        162     5100
199        mpfi       3.62    3.15              7.5        162     5100

    citympg  highwaympg    price
100      27          34   9549.0
101      17          22  13499.0
102      17          22  14399.0
103      19          25  13499.0
104      19          25  17199.0
..      …          …       …
195      23          28  13415.0
196      24          28  15985.0
197      24          28  16515.0
198      17          22  18420.0
199      17          22  18950.0
```

```
[100 rows x 26 columns]
```

[10]:
```
# describe() method gives us summary statistics for numerical columns in DataFrame.
df.describe().T
```

[10]:

	count	mean	std	min	25%	\
car_ID	205.0	103.000000	59.322565	1.00	52.00	
symboling	205.0	0.834146	1.245307	-2.00	0.00	
wheelbase	205.0	98.756585	6.021776	86.60	94.50	
carlength	205.0	174.049268	12.337289	141.10	166.30	
carwidth	205.0	65.907805	2.145204	60.30	64.10	
carheight	205.0	53.724878	2.443522	47.80	52.00	
curbweight	205.0	2555.565854	520.680204	1488.00	2145.00	
enginesize	205.0	126.907317	41.642693	61.00	97.00	
boreratio	205.0	3.329756	0.270844	2.54	3.15	
stroke	205.0	3.255415	0.313597	2.07	3.11	
compressionratio	205.0	10.142537	3.972040	7.00	8.60	
horsepower	205.0	104.117073	39.544167	48.00	70.00	
peakrpm	205.0	5125.121951	476.985643	4150.00	4800.00	
citympg	205.0	25.219512	6.542142	13.00	19.00	
highwaympg	205.0	30.751220	6.886443	16.00	25.00	
price	205.0	13276.710571	7988.852332	5118.00	7788.00	

	50%	75%	max
car_ID	103.00	154.00	205.00
symboling	1.00	2.00	3.00
wheelbase	97.00	102.40	120.90
carlength	173.20	183.10	208.10
carwidth	65.50	66.90	72.30
carheight	54.10	55.50	59.80
curbweight	2414.00	2935.00	4066.00
enginesize	120.00	141.00	326.00
boreratio	3.31	3.58	3.94
stroke	3.29	3.41	4.17
compressionratio	9.00	9.40	23.00
horsepower	95.00	116.00	288.00
peakrpm	5200.00	5500.00	6600.00
citympg	24.00	30.00	49.00
highwaympg	30.00	34.00	54.00
price	10295.00	16503.00	45400.00

[11]:
```
# prints information about the DataFrame. [number of columns, column labels, column data types, memory usage, range index,]
df.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
```

[12]:
```
# Including only string columns in a DataFrame description
df.describe(include=object)
```

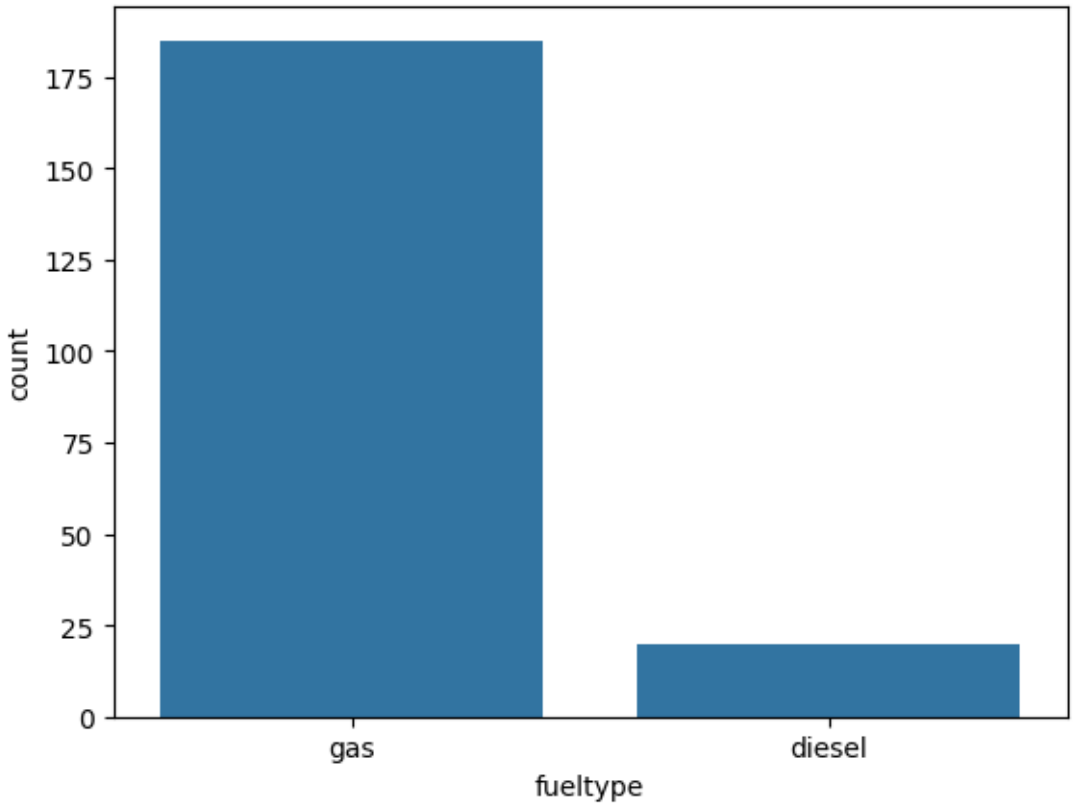
[12]:

	CarName	fueltype	aspiration	doornumber	carbody	drivewheel	enginelocation	enginetype	cylindernumber	fuelsystem
count	205	205	205	205	205	205	205	205	205	205
unique	147	2	2	2	5	3	2	7	7	8
top	toyota corona	gas	std	four	sedan	fwd	front	ohc	four	mpfi
freq	6	185	168	115	96	120				

```
freq              202        148             159          94
```

[13]:
```
sns.countplot(data=df,x='fueltype')
```

[13]:
```
<Axes: xlabel='fueltype', ylabel='count'>
```



#DATA CLEANING

[14]:
```
# To check for duplicate values in a  DataFrame
df.duplicated().sum()
```

[14]:
```
0
```

[15]:
```
# Missing values in the dataset.
df.isnull().sum()
```

[15]:
```
car_ID            0
symboling         0
CarName           0
fueltype          0
aspiration        0
```

```
doornumber          0
carbody             0
drivewheel          0
enginelocation      0
wheelbase           0
carlength           0
carwidth            0
carheight           0
curbweight          0
enginetype          0
cylindernumber      0
enginesize          0
fuelsystem          0
boreratio           0
stroke              0
compressionratio    0
horsepower          0
peakrpm             0
citympg             0
highwaympg          0
price               0
dtype: int64
```

[16]:
```
# index row columns remove
df=df.drop(["car_ID"],axis=1)
```

[17]:
```
df.dtypes
```

[17]:
```
symboling             int64
CarName              object
fueltype             object
aspiration           object
doornumber           object
carbody              object
drivewheel           object
enginelocation       object
wheelbase           float64
carlength           float64
carwidth            float64
carheight           float64
curbweight            int64
enginetype           object
cylindernumber       object
enginesize            int64
fuelsystem           object
boreratio           float64
stroke              float64
```

```
compressionratio    float64
horsepower            int64
peakrpm               int64
citympg               int64
highwaympg            int64
price               float64
dtype: object
```

[18]:
```
df['CarName'].unique()
```

[18]:
```
array(['alfa-romero giulia', 'alfa-romero stelvio',
       'alfa-romero Quadrifoglio', 'audi 100 ls', 'audi 100ls',
       'audi fox', 'audi 5000', 'audi 4000', 'audi 5000s (diesel)',
       'bmw 320i', 'bmw x1', 'bmw x3', 'bmw z4', 'bmw x4', 'bmw x5',
       'chevrolet impala', 'chevrolet monte carlo', 'chevrolet vega 2300',
       'dodge rampage', 'dodge challenger se', 'dodge d200',
       'dodge monaco (sw)', 'dodge colt hardtop', 'dodge colt (sw)',
       'dodge coronet custom', 'dodge dart custom',
       'dodge coronet custom (sw)', 'honda civic', 'honda civic cvcc',
       'honda accord cvcc', 'honda accord lx', 'honda civic 1500 gl',
       'honda accord', 'honda civic 1300', 'honda prelude',
       'honda civic (auto)', 'isuzu MU-X', 'isuzu D-Max ',
       'isuzu D-Max V-Cross', 'jaguar xj', 'jaguar xf', 'jaguar xk',
       'maxda rx3', 'maxda glc deluxe', 'mazda rx2 coupe', 'mazda rx-4',
       'mazda glc deluxe', 'mazda 626', 'mazda glc', 'mazda rx-7 gs',
       'mazda glc 4', 'mazda glc custom l', 'mazda glc custom',
       'buick electra 225 custom', 'buick century luxus (sw)',
       'buick century', 'buick skyhawk', 'buick opel isuzu deluxe',
       'buick skylark', 'buick century special',
       'buick regal sport coupe (turbo)', 'mercury cougar',
       'mitsubishi mirage', 'mitsubishi lancer', 'mitsubishi outlander',
       'mitsubishi g4', 'mitsubishi mirage g4', 'mitsubishi montero',
       'mitsubishi pajero', 'Nissan versa', 'nissan gt-r', 'nissan rogue',
       'nissan latio', 'nissan titan', 'nissan leaf', 'nissan juke',
       'nissan note', 'nissan clipper', 'nissan nv200', 'nissan dayz',
       'nissan fuga', 'nissan otti', 'nissan teana', 'nissan kicks',
       'peugeot 504', 'peugeot 304', 'peugeot 504 (sw)', 'peugeot 604sl',
       'peugeot 505s turbo diesel', 'plymouth fury iii',
       'plymouth cricket', 'plymouth satellite custom (sw)',
       'plymouth fury gran sedan', 'plymouth valiant', 'plymouth duster',
       'porsche macan', 'porcshce panamera', 'porsche cayenne',
       'porsche boxter', 'renault 12tl', 'renault 5 gtl', 'saab 99e',
       'saab 99le', 'saab 99gle', 'subaru', 'subaru dl', 'subaru brz',
       'subaru baja', 'subaru r1', 'subaru r2', 'subaru trezia',
       'subaru tribeca', 'toyota corona mark ii', 'toyota corona',
       'toyota corolla 1200', 'toyota corona hardtop',
       'toyota corolla 1600 (sw)', 'toyota carina', 'toyota mark ii',
```

```
       'toyota corolla', 'toyota corolla liftback',
       'toyota celica gt liftback', 'toyota corolla tercel',
       'toyota corona liftback', 'toyota starlet', 'toyota tercel',
       'toyota cressida', 'toyota celica gt', 'toyouta tercel',
       'vokswagen rabbit', 'volkswagen 1131 deluxe sedan',
       'volkswagen model 111', 'volkswagen type 3', 'volkswagen 411 (sw)',
       'volkswagen super beetle', 'volkswagen dasher', 'vw dasher',
       'vw rabbit', 'volkswagen rabbit', 'volkswagen rabbit custom',
       'volvo 145e (sw)', 'volvo 144ea', 'volvo 244dl', 'volvo 245',
       'volvo 264gl', 'volvo diesel', 'volvo 246'], dtype=object)
```

[19]:
```
df.insert(1, 'Company', df['CarName'].str.split(' ').str[0])
#df.insert(1,'company',df['CarName'].str.split('').str[0])  #error
```

[20]:
```
df.drop(columns=['CarName'],inplace=True)
```

[21]:
```
df
```

[21]:

	symboling	Company	fueltype	aspiration	doornumber	carbody	\
0	3	alfa-romero	gas	std	two	convertible	
1	3	alfa-romero	gas	std	two	convertible	
2	1	alfa-romero	gas	std	two	hatchback	
3	2	audi	gas	std	four	sedan	
4	2	audi	gas	std	four	sedan	
..	…	…	…	…	…	…	
200	-1	volvo	gas	std	four	sedan	
201	-1	volvo	gas	turbo	four	sedan	
202	-1	volvo	gas	std	four	sedan	
203	-1	volvo	diesel	turbo	four	sedan	
204	-1	volvo	gas	turbo	four	sedan	

	drivewheel	enginelocation	wheelbase	carlength	…	enginesize	\
0	rwd	front	88.6	168.8	…	130	
1	rwd	front	88.6	168.8	…	130	
2	rwd	front	94.5	171.2	…	152	
3	fwd	front	99.8	176.6	…	109	
4	4wd	front	99.4	176.6	…	136	
..	…	…	…	…	…	…	
200	rwd	front	109.1	188.8	…	141	
201	rwd	front	109.1	188.8	…	141	
202	rwd	front	109.1	188.8	…	173	
203	rwd	front	109.1	188.8	…	145	
204	rwd	front	109.1	188.8	…	141	

	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	\
0	mpfi	3.47	2.68	9.0	111	5000	
1	mpfi	3.47	2.68	9.0	111	5000	

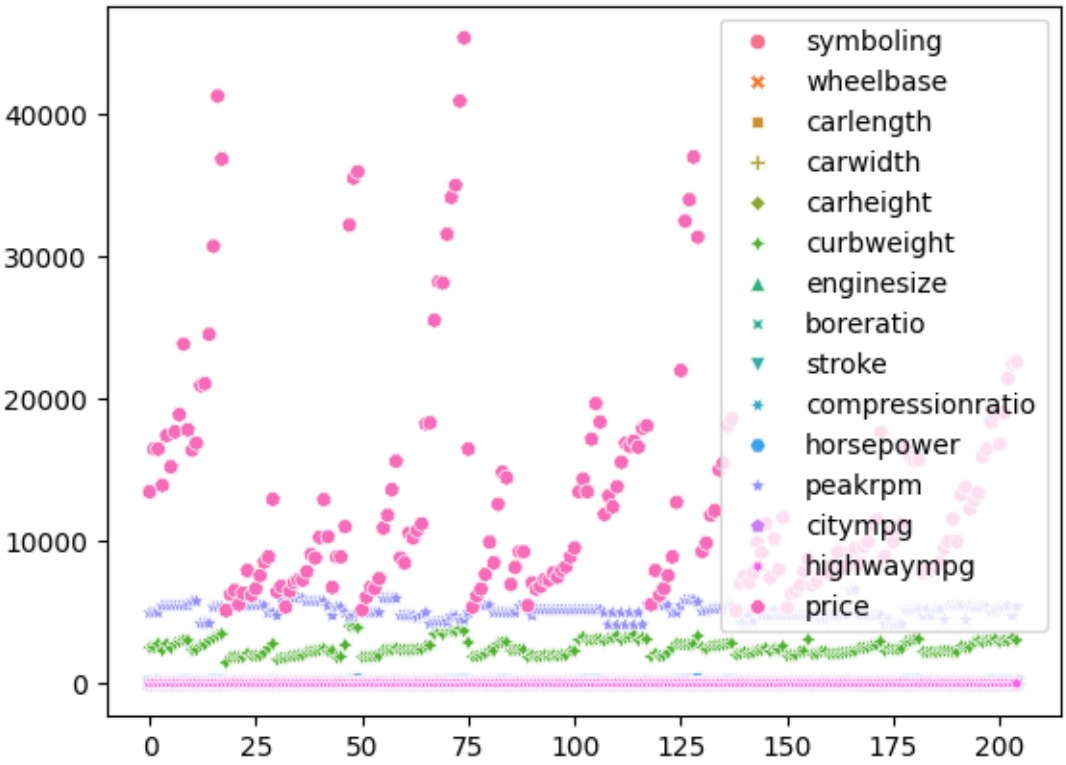
```
2          mpfi       2.68   3.47              9.0         154    5000
3          mpfi       3.19   3.40             10.0         102    5500
4          mpfi       3.19   3.40              8.0         115    5500
..          …          …      …                …           …      …
200        mpfi       3.78   3.15              9.5         114    5400
201        mpfi       3.78   3.15              8.7         160    5300
202        mpfi       3.58   2.87              8.8         134    5500
203         idi       3.01   3.40             23.0         106    4800
204        mpfi       3.78   3.15              9.5         114    5400

     citympg  highwaympg    price
0         21          27  13495.0
1         21          27  16500.0
2         19          26  16500.0
3         24          30  13950.0
4         18          22  17450.0
..       …           …       …
200       23          28  16845.0
201       19          25  19045.0
202       18          23  21485.0
203       26          27  22470.0
204       19          25  22625.0

[205 rows x 25 columns]
```

[22]:
```
df['Company'].unique()
```

[22]:
```
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)
```

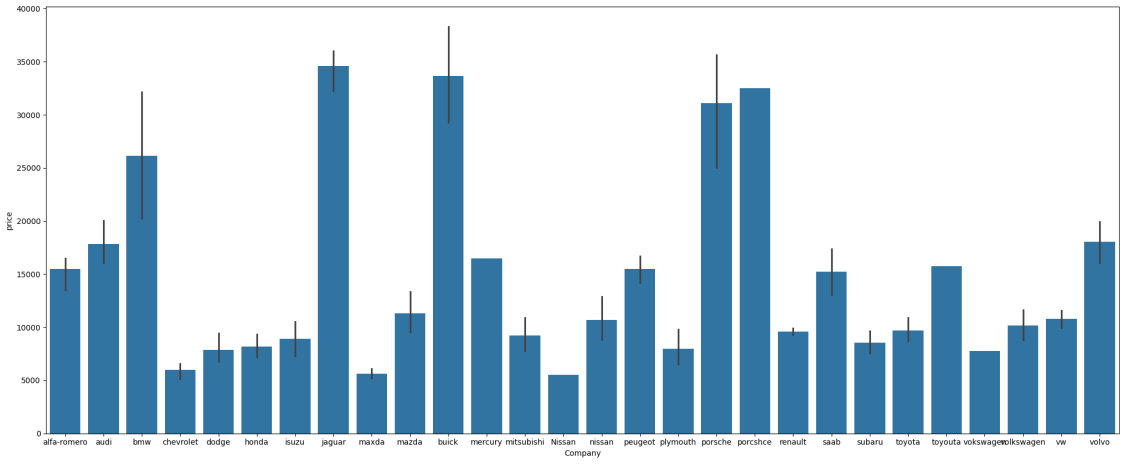
[23]:
```
sns.scatterplot(df)
```

[23]:
```
<Axes: >
```



[24]:
```
plt.subplots(figsize=(25,10))
sns.barplot(data=df,x='Company',y='price')
```

[24]: <Axes: xlabel='Company', ylabel='price'>



[25]:
```
# To select strings you must use the object dtype, but note that this will␣
 ↪return all object dtype columns
df.select_dtypes(include=object).columns.to_list()
#df.select_dtypes(include=object).columns.tolist()
```

[25]:
```
['Company',
 'fueltype',
 'aspiration',
 'doornumber',
 'carbody',
 'drivewheel',
 'enginelocation',
 'enginetype',
 'cylindernumber',
 'fuelsystem']
```

[26]:
```
# unues columns
df.drop(columns=['symboling','wheelbase'],inplace=True)
```

[27]:
```
# Numerical columns list
numerical_columns = df.select_dtypes(include=['int64', 'float64']).columns.
 ↪tolist()
numerical_columns.remove('price')
```

[28]:
```
# Label encoding
```

[29]:
```
from sklearn.preprocessing import LabelEncoder
lb=LabelEncoder()
df['Company']=lb.fit_transform(df['Company'])
df['fueltype']=lb.fit_transform(df['fueltype'])
df['aspiration']=lb.fit_transform(df['aspiration'])
df['doornumber']=lb.fit_transform(df['doornumber'])
df['carbody']=lb.fit_transform(df['carbody'])
df['drivewheel']=lb.fit_transform(df['drivewheel'])
df['enginelocation']=lb.fit_transform(df['enginelocation'])
df['enginetype']=lb.fit_transform(df['enginetype'])
df['cylindernumber']=lb.fit_transform(df['cylindernumber'])
df['fuelsystem']=lb.fit_transform(df['fuelsystem'])
```

[30]:
```
numerical_columns
```

[30]:
```
['carlength',
 'carwidth',
 'carheight',
 'curbweight',
 'enginesize',
 'boreratio',
```

```
 'stroke',
 'compressionratio',
 'horsepower',
 'peakrpm',
 'citympg',
 'highwaympg']
```

[31]:
```
from sklearn.preprocessing import StandardScaler
ssr=StandardScaler()
df[numerical_columns]=ssr.fit_transform(df[numerical_columns])
```

[32]:
```
df
```

[32]:

	Company	fueltype	aspiration	doornumber	carbody	drivewheel	\
0	1	1	0	1	0	2	
1	1	1	0	1	0	2	
2	1	1	0	1	2	2	
3	2	1	0	0	3	1	
4	2	1	0	0	3	0	
..	…	…	…	…	…	…	
200	26	1	0	0	3	2	
201	26	1	1	0	3	2	
202	26	1	0	0	3	2	
203	26	0	1	0	3	2	
204	26	1	1	0	3	2	

	enginelocation	carlength	carwidth	carheight	…	enginesize	\
0	0	-0.426521	-0.844782	-2.020417	…	0.074449	
1	0	-0.426521	-0.844782	-2.020417	…	0.074449	
2	0	-0.231513	-0.190566	-0.543527	…	0.604046	
3	0	0.207256	0.136542	0.235942	…	-0.431076	
4	0	0.207256	0.230001	0.235942	…	0.218885	
..	…	…	…	…	…	…	
200	0	1.198549	1.398245	0.728239	…	0.339248	
201	0	1.198549	1.351515	0.728239	…	0.339248	
202	0	1.198549	1.398245	0.728239	…	1.109571	
203	0	1.198549	1.398245	0.728239	…	0.435538	
204	0	1.198549	1.398245	0.728239	…	0.339248	

	fuelsystem	boreratio	stroke	compressionratio	horsepower	peakrpm	\
0	5	0.519071	-1.839377	-0.288349	0.174483	-0.262960	
1	5	0.519071	-1.839377	-0.288349	0.174483	-0.262960	
2	5	-2.404880	0.685946	-0.288349	1.264536	-0.262960	
3	5	-0.517266	0.462183	-0.035973	-0.053668	0.787855	
4	5	-0.517266	0.462183	-0.540725	0.275883	0.787855	
..	…	…	…	…	…	…	
200	5	1.666445	-0.336970	-0.162161	0.250533	0.577692	

```
201           5   1.666445 -0.336970         -0.364062    1.416637  0.367529
202           5   0.926204 -1.232021         -0.338824    0.757535  0.787855
203           3  -1.183483  0.462183          3.244916    0.047732 -0.683286
204           5   1.666445 -0.336970         -0.162161    0.250533  0.577692

      citympg  highwaympg    price
0   -0.646553   -0.546059  13495.0
1   -0.646553   -0.546059  16500.0
2   -0.953012   -0.691627  16500.0
3   -0.186865   -0.109354  13950.0
4   -1.106241   -1.273900  17450.0
..        …         …       …
200 -0.340094   -0.400490  16845.0
201 -0.953012   -0.837195  19045.0
202 -1.106241   -1.128332  21485.0
203  0.119594   -0.546059  22470.0
204 -0.953012   -0.837195  22625.0

[205 rows x 23 columns]
```

[33]:
```
# x and y split
```

[34]:
```
x=df.iloc[ : , :-1]
y=df.iloc[ : , -1]
```

[35]:
```
# Train and Test split
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.
 ↪3,random_state=11)
```

2 LinearRegression

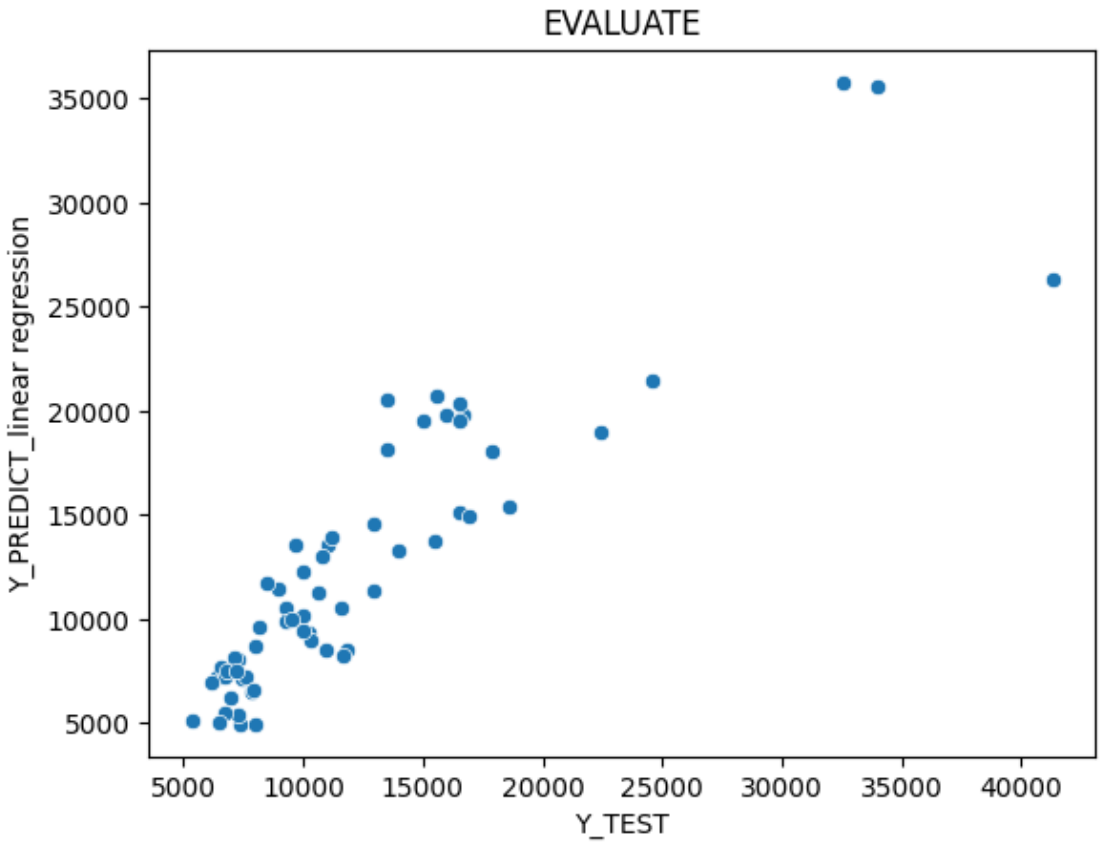
[36]:
```
# LinearRegression
from sklearn.linear_model import LinearRegression

# Create a Linear Regression model
lr=LinearRegression()

# Train the model on the training data
lr.fit(x_train,y_train)

# Make predictions on the test data
y_pred_lr=lr.predict(x_test)

# Evaluate the model's performance
from sklearn.metrics import mean_absolute_error,mean_squared_error
```

```
print('MAE:',mean_absolute_error(y_test,y_pred_lr))
mse_lr=mean_squared_error(y_test,y_pred_lr)
print('MSE:',mse_lr)
```

```
MAE: 2142.9703979715828
MSE: 9351120.379315864
```

[37]:
```
# Create a scatter plot
sns.scatterplot(x=y_test,y=y_pred_lr)

# Add labels and a title
plt.xlabel('Y_TEST')
plt.ylabel('Y_PREDICT_linear regression')
plt.title('EVALUATE')
```

[37]:
```
Text(0.5, 1.0, 'EVALUATE')
```



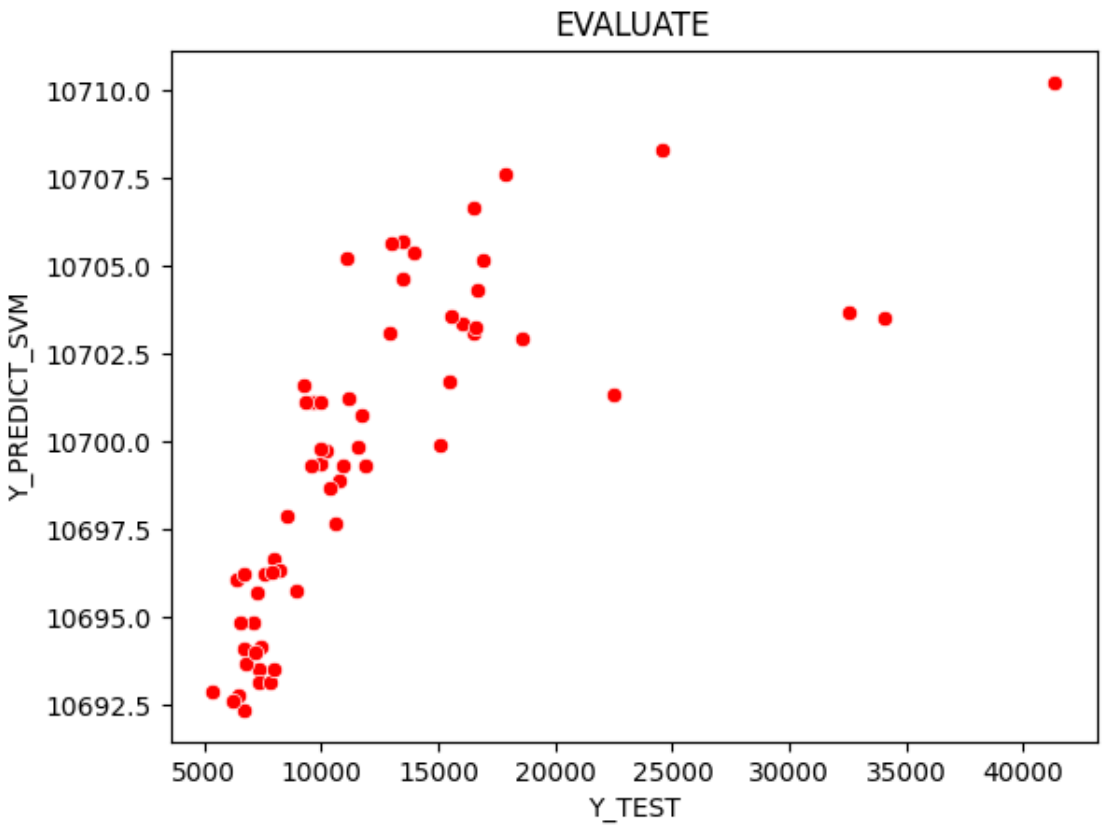
3 Support Vector Machine (SVM)

[38]:
```
#svm
from sklearn.svm import SVR

# Create a svm model
svr=SVR()

# Train the model on the training data
svr.fit(x_train,y_train)

# Make predictions on the test data
y_pred_svm=svr.predict(x_test)

#  Evaluate the model
from sklearn.metrics import mean_absolute_error,mean_squared_error
print('MAE:',mean_absolute_error(y_test,y_pred_svm))
mse_svm=mean_squared_error(y_test,y_pred_svm)
print("mse",mse_svm)
```

```
MAE: 4427.266337086366
mse 48774553.675131686
```

[39]:
```
# Create a scatter plot
sns.scatterplot(x=y_test,y=y_pred_svm,color = "red")

# Add labels and a title
plt.xlabel('Y_TEST')
plt.ylabel('Y_PREDICT_SVM')
plt.title('EVALUATE')
```

[39]:
```
Text(0.5, 1.0, 'EVALUATE')
```



4 KNeighborsRegressor

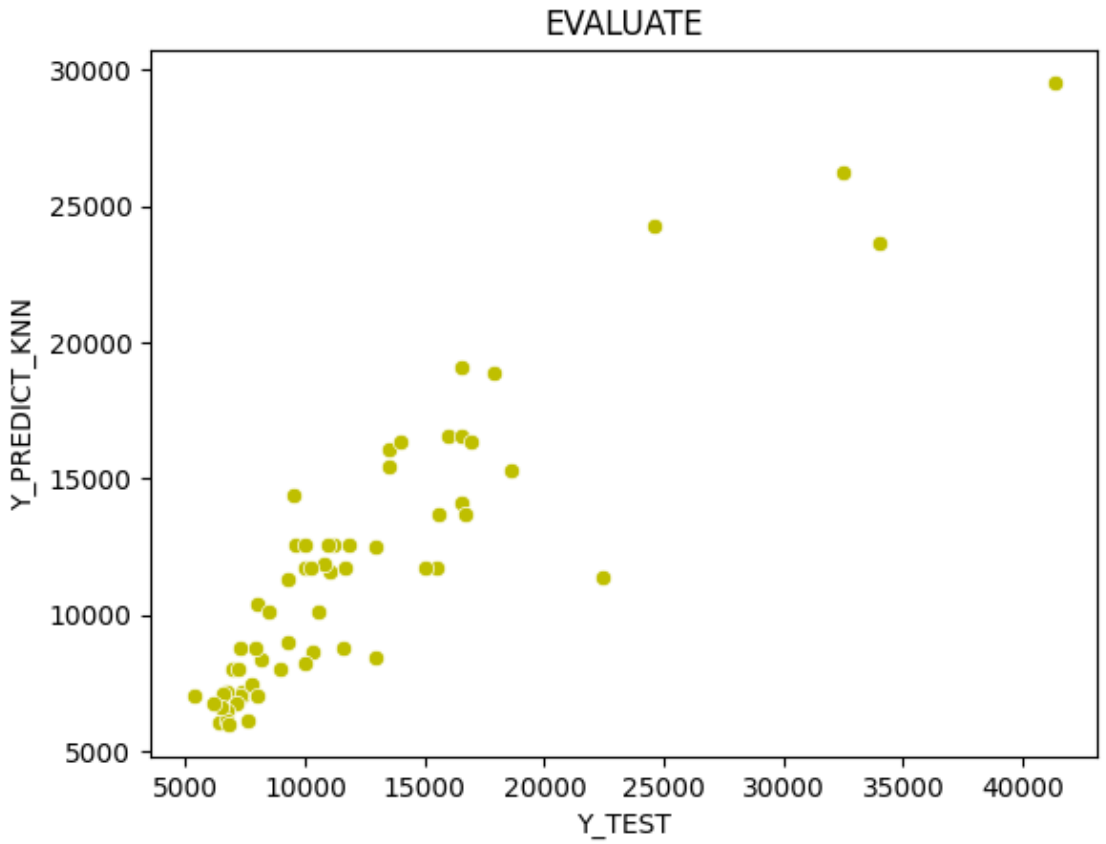
[40]:
```
# KNeighborsRegressor
from sklearn.neighbors import KNeighborsRegressor

## Create KNN model
k=3
knn=KNeighborsRegressor(n_neighbors=k)

# Train the model on the training data
knn.fit(x_train,y_train)

# Make predictions on the test data
y_pred_knn=knn.predict(x_test)

# Evaluate the model's performance
from sklearn.metrics import mean_absolute_error,mean_squared_error
print('MAE:',mean_absolute_error(y_test,y_pred_knn))
mse_knn=mean_squared_error(y_test,y_pred_knn)
```

```
print("MSE",mse_knn)
```

```
MAE: 1955.4408548387094
MSE 9697257.796603944
```

[41]:
```
# Create a scatter plot
sns.scatterplot(x=y_test,y=y_pred_knn,color = "y")

# Add labels and a title
plt.xlabel('Y_TEST')
plt.ylabel('Y_PREDICT_KNN')
plt.title('EVALUATE')
```

[41]:
```
Text(0.5, 1.0, 'EVALUATE')
```



[42]:
```
#Choosing 'k' by elbow method
```

[43]:
```
knn_mse=[]
for i in range(1,60):
  knn_i=KNeighborsRegressor(n_neighbors=i)
```

```
    # train data
    knn_i.fit(x_train,y_train)
    # predict
    y_pred_knn_i=knn.predict(x_test)
    knn_mse.append(mean_squared_error(y_test,y_pred_knn_i))
```

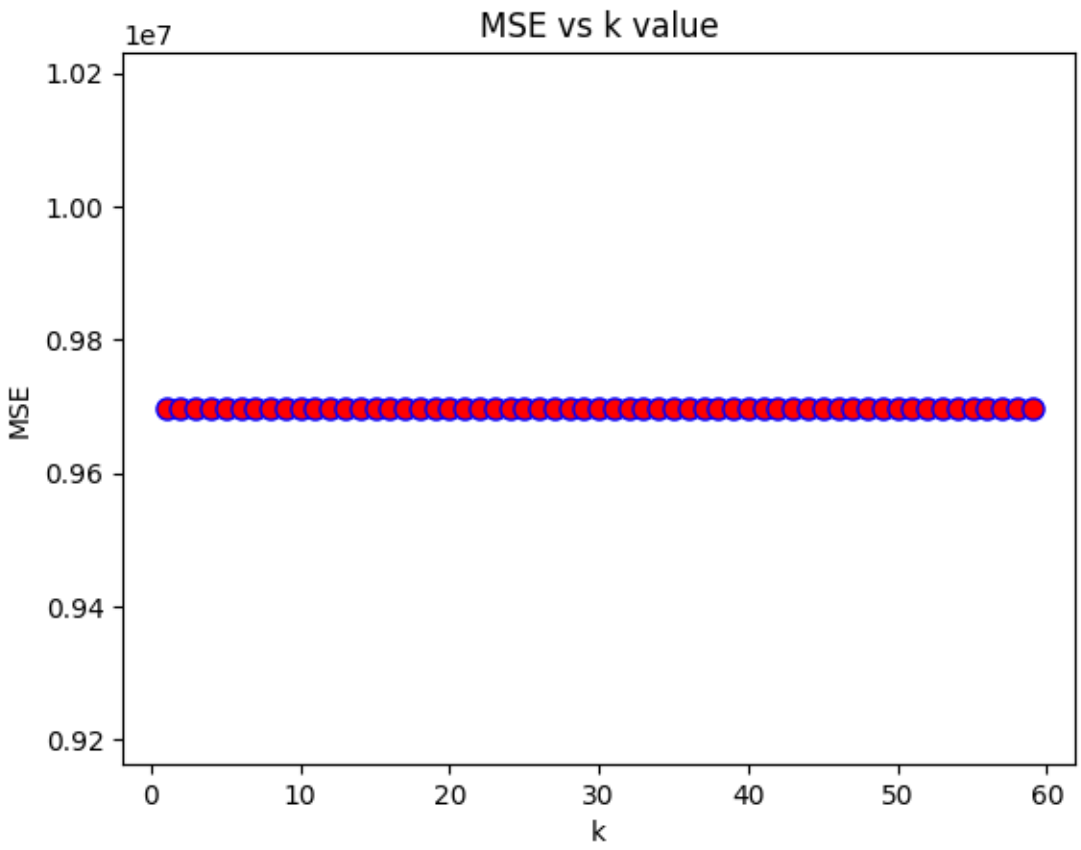
[44]:
```
knn_mse
```

[44]:
```
[9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
```

```
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944,
 9697257.796603944]
```

[45]:
```
# PLOT FOR ERROR VALUE
plt.
 ↪plot(range(1,60),knn_mse,color='blue',linestyle='dashed',marker='o',markerfacecolor='red',␣
 ↪markersize=8)
plt.title('MSE vs k value')
plt.xlabel('k')
plt.ylabel("MSE")
```

[45]:
```
Text(0, 0.5, 'MSE')
```



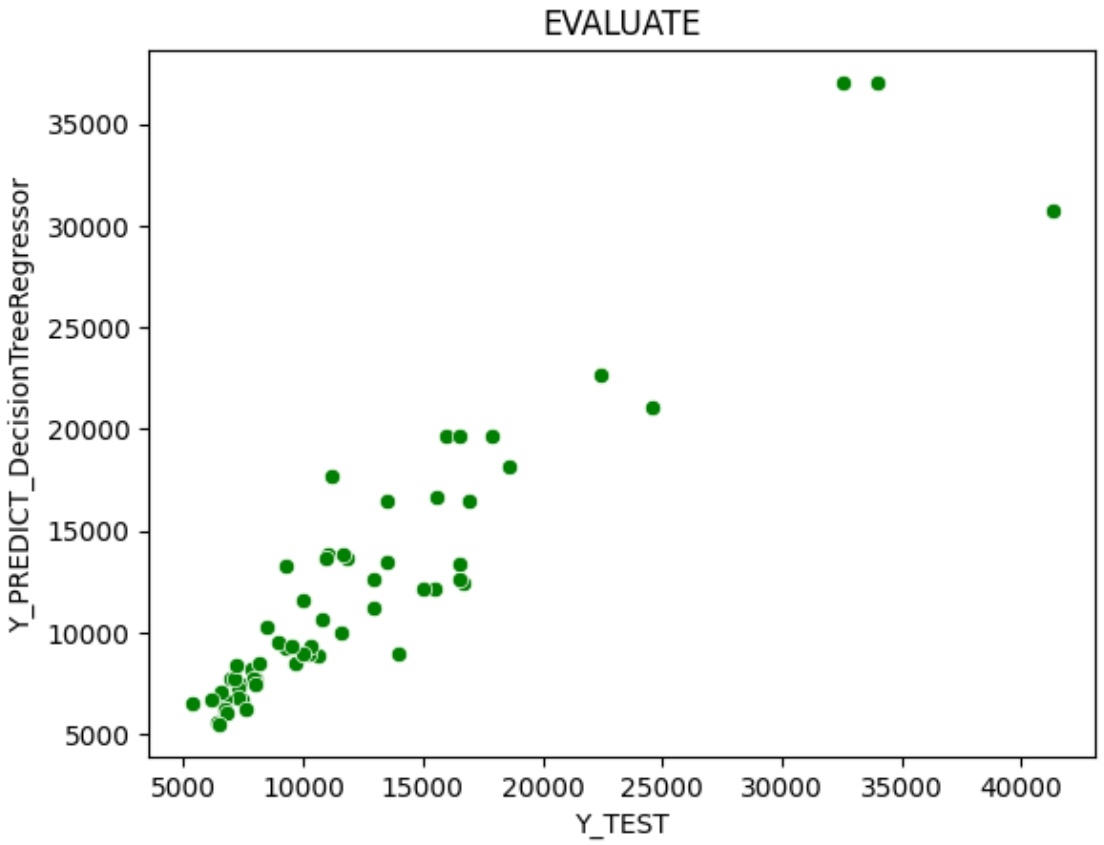
[46]:
```
#DecisionTreeRegressor
```

[47]:
```
# DecisionTreeRegressor

from sklearn.tree import DecisionTreeRegressor

# Create a # Create a Linear Regression model model
dt=DecisionTreeRegressor()

# Train the model on the training data
dt.fit(x_train,y_train)

# Make predictions on the test data
y_pred_dt=dt.predict(x_test)

# Evaluate the model's performance
print('MAE:',mean_absolute_error(y_test,y_pred_dt))
mse_dt=mean_squared_error(y_test,y_pred_dt)
print('MSE:',mse_dt)
```

```
MAE: 1700.9166612903225
MSE: 6275232.620449822
```

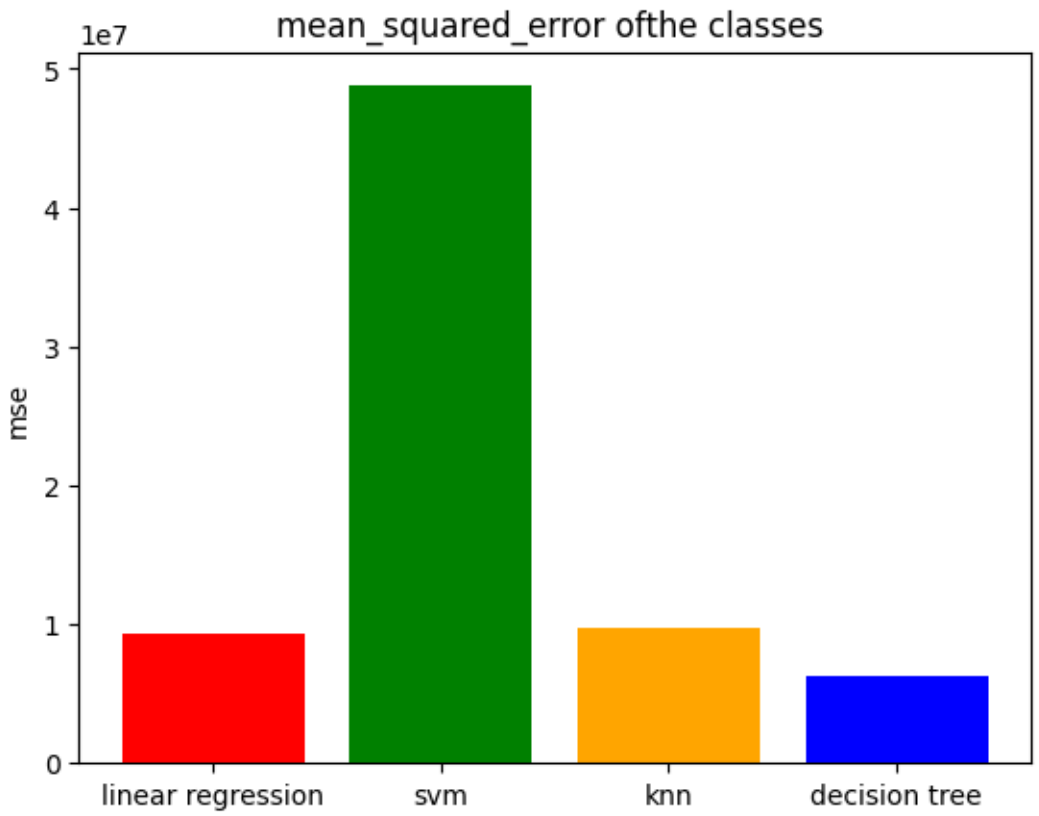
[48]:
```
# Create a scatter plot
sns.scatterplot(x=y_test,y=y_pred_dt,color = "green")
# Add labels and a title
plt.xlabel('Y_TEST')
plt.ylabel('Y_PREDICT_DecisionTreeRegressor')
plt.title('EVALUATE')
```

[48]: Text(0.5, 1.0, 'EVALUATE')



[52]:
```
class_name = ('linear regression','svm','knn','decision tree')
class_score=(mse_lr,mse_svm,mse_knn,mse_dt)
colors=('r','g','orange','b','pink')
plt.bar(class_name,class_score,color=colors)
plt.title ("mean_squared_error ofthe classes ")
plt.ylabel('mse')
```

[52]: Text(0, 0.5, 'mse')



5 r2_score

[53]:
```
from sklearn.metrics import r2_score
```

[54]:
```
r2_lr=r2_score(y_test,y_pred_lr)
print('r2_score_lr',r2_lr)
r2_svm=r2_score(y_test,y_pred_svm)
print('r2_score_svm',r2_svm)
r2_knn=r2_score(y_test,y_pred_knn)
print('r2_score_knn',r2_knn)
r2_dt=r2_score(y_test,y_pred_dt)
print('r2_score_knn',r2_dt)
```

```
r2_score_lr 0.7995224343756604
r2_score_svm -0.045671896902632625
r2_score_knn 0.7921016351586058
r2_score_knn 0.8654660288346897
```

[55]:
```
#r2 in function
```

[56]:
```
def r2score(Y_test,Y_pred):
  print('R2 SCORE',r2_score(Y_test,Y_pred))
```

[57]:
```
#LR
r2score(y_test,y_pred_lr)
#KNN
r2score(y_test,y_pred_knn)
# SVM
r2score(y_test,y_pred_svm)
# DT
r2score(y_test,y_pred_dt)
```

```
R2 SCORE 0.7995224343756604
R2 SCORE 0.7921016351586058
R2 SCORE -0.045671896902632625
R2 SCORE 0.8654660288346897
```

[58]:
```
#data frame
```

[59]:
```
data={"model":['LinearRegression','SVM','KNN','DecisionTreeRegressor'],
      "mse":[mse_lr,mse_svm,mse_knn,mse_dt],
      'r2score':[r2_lr,r2_svm,r2_knn,r2_dt]}
df2=pd.DataFrame(data)
df2.T
```

[59]:

	0	1	2	\
model	LinearRegression	SVM	KNN	
mse	9351120.379316	48774553.675132	9697257.796604	
r2score	0.799522	-0.045672	0.792102	

	3
model	DecisionTreeRegressor
mse	6275232.62045
r2score	0.865466