Project-1 Regression Problem (Classified Ads for Cars - Used cars for sale in Germany and Czech Republic since 2015)

Data description

Link to the dataset : [https://www.kaggle.com/mirosval/personal-cars-classifieds (https://www.kaggle.com/mirosval/personal-cars-classifieds)](https://www.kaggle.com/mirosval/personal-cars-classifieds)

There are roughly 3,5 Million rows and the following columns:

maker - normalized all lowercase

model - normalized all lowercase

mileage - in KM

manufacture_year ¶

engine_displacement - in ccm

engine_power - in kW

body_type - almost never present, but I scraped only personal cars, no motorcycles or utility vehicles

color_slug - also almost never present

stk_year - year of the last emission control

transmission - automatic or manual

door_count - Number of doors for the vehicle

seat_count - Number of seats available in the vehicle

fuel_type - gasoline, diesel, cng, lpg, electric

date_created - when the ad was scraped

datelastseen - when the ad was last seen. Our policy was to remove all ads older than 60 days

price_eur - list price converted to EUR

Problem Statement:

Which factors determine the price of a car?

With what accuracy can the price be predicted?

Can a model trained on all cars be used to accurately predict prices of models with only a few samples?

Importing libraries

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

In [3]:
```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.preprocessing import PolynomialFeatures
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.svm import SVR
from sklearn.svm import LinearSVR
from sklearn.model_selection import GridSearchCV
```

Loading dataset

In [4]:
```
df_car=pd.read_csv("all_anonymized_2015_11_2017_03.csv")
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py:272
8: DtypeWarning: Columns (7,8,10,11,12) have mixed types.Specify dtype option o
n import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Data Exploration

In [5]:
```
df_car.info()   # Giving Dataset size and attibute details along with their data
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3552912 entries, 0 to 3552911
Data columns (total 16 columns):
 #   Column               Dtype  
---  ------               -----  
 0   maker                object 
 1   model                object 
 2   mileage              float64
 3   manufacture_year     float64
 4   engine_displacement  float64
 5   engine_power         float64
 6   body_type            object 
 7   color_slug           object 
 8   stk_year             object 
 9   transmission         object 
 10  door_count           object 
 11  seat_count           object 
 12  fuel_type            object 
 13  date_created         object 
 14  date_last_seen       object 
 15  price_eur            float64
dtypes: float64(5), object(11)
memory usage: 433.7+ MB
```

In [6]:
```
df_car.describe().T    # Getting descriptive statistics excluding NAN values on
```

Out[6]:

	count	mean	std	min	25%	50%	75%
mileage	3190328.0	1.158140e+05	3.422508e+05	0.00	18800.00	86415.00	158025.00
manufacture_year	3182334.0	2.000871e+03	8.172588e+01	0.00	2004.00	2009.00	2013.00
engine_displacement	2809498.0	2.043958e+03	1.973958e+03	0.00	1400.00	1798.00	1997.00
engine_power	2998035.0	9.846796e+01	4.907309e+01	1.00	68.00	86.00	110.00
price_eur	3552912.0	1.625812e+06	2.025622e+09	0.04	1295.34	7364.91	16284.23

In [7]:
```
df_car.head(10)
```

Out[7]:

	maker	model	mileage	manufacture_year	engine_displacement	engine_power	body_type	col
0	ford	galaxy	151000.0	2011.0	2000.0	103.0	NaN	
1	skoda	octavia	143476.0	2012.0	2000.0	81.0	NaN	
2	bmw	NaN	97676.0	2010.0	1995.0	85.0	NaN	
3	skoda	fabia	111970.0	2004.0	1200.0	47.0	NaN	
4	skoda	fabia	128886.0	2004.0	1200.0	47.0	NaN	
5	skoda	fabia	140932.0	2003.0	1200.0	40.0	NaN	
6	skoda	fabia	167220.0	2001.0	1400.0	74.0	NaN	
7	bmw	NaN	148500.0	2009.0	2000.0	130.0	NaN	
8	skoda	octavia	105389.0	2003.0	1900.0	81.0	NaN	
9	NaN	NaN	301381.0	2002.0	1900.0	88.0	NaN	

In [12]:
```
df_car.shape    # Number of rows and columns - DataFrame Size
```

Out[12]:
```
(3552912, 16)
```

In [13]:
```
df_car.isnull().sum()    # Checking Null values
```

Out[13]:
```
maker                   518915
model                  1133361
mileage                 362584
manufacture_year        370578
engine_displacement     743414
engine_power            554877
body_type              1122914
color_slug             3343411
stk_year               1708156
transmission            741630
door_count              614373
seat_count              749489
fuel_type              1847606
date_created                 0
date_last_seen               0
price_eur                    0
dtype: int64
```

In [16]:
```
df_car.isnull().mean()*100     # checking percentage of Nulls
```

Out[16]:
```
maker                  14.605343
model                  31.899495
mileage                10.205263
manufacture_year       10.430261
engine_displacement    20.924076
engine_power           15.617527
body_type              31.605455
color_slug             94.103400
stk_year               48.077633
transmission           20.873863
door_count             17.292097
seat_count             21.095062
fuel_type              52.002583
date_created            0.000000
date_last_seen          0.000000
price_eur               0.000000
dtype: float64
```

In [17]:
```
### Noticing large percentage of missing values for the predictors - 'model','co
### Imputing dataframe by dropping these columns
```

In [18]:
```
df_car = df_car.drop(columns =['model','color_slug', 'stk_year'])   #Dropping co
```

In [19]:
```
df_car['date_created'].head   # checking date format
```

Out[19]:
```
<bound method NDFrame.head of 0          2015-11-14 18:10:06.838319+00
1          2015-11-14 18:10:06.853411+00
2          2015-11-14 18:10:06.861792+00
3          2015-11-14 18:10:06.872313+00
4          2015-11-14 18:10:06.880335+00
                       ...
3552907     2017-03-16 18:57:35.46558+00
3552908    2017-03-16 18:57:37.761349+00
3552909    2017-03-16 18:57:40.435847+00
3552910    2017-03-16 18:57:43.595523+00
3552911    2017-03-16 19:22:23.946774+00
Name: date_created, Length: 3552912, dtype: object>
```

In [20]:
```
df_car['date_created'] = df_car['date_created'].str.slice(0, 10)   # Datepart su
df_car['date_last_seen'] = df_car['date_last_seen'].str.slice(0, 10) # Datepart :
```

In [21]:
```python
df_car['date_created'].head
```

Out[21]:
```
<bound method NDFrame.head of 0          2015-11-14
1          2015-11-14
2          2015-11-14
3          2015-11-14
4          2015-11-14
              ...
3552907    2017-03-16
3552908    2017-03-16
3552909    2017-03-16
3552910    2017-03-16
3552911    2017-03-16
Name: date_created, Length: 3552912, dtype: object>
```

In [22]:
```python
# For Body_type & Fuel_type we are noticing somewhat significant of missing value
# as we have large dataset
```

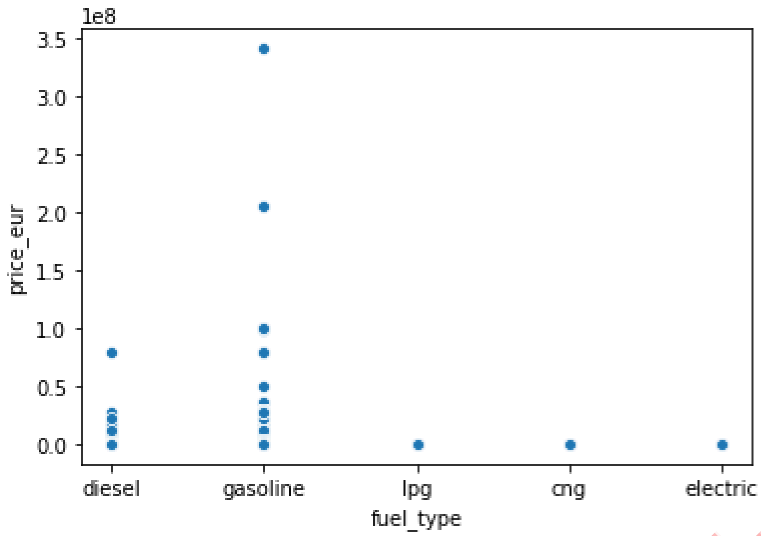
In [23]:
```python
# checking categotical vaues for Body_type and sort by descending
df_car['body_type'].value_counts(sort=True)
```

Out[23]:
```
other           1964289
compact          241948
coupe             70576
stationwagon      69895
van               31300
offroad           22549
sedan             19669
convertible        5332
transporter        4440
Name: body_type, dtype: int64
```

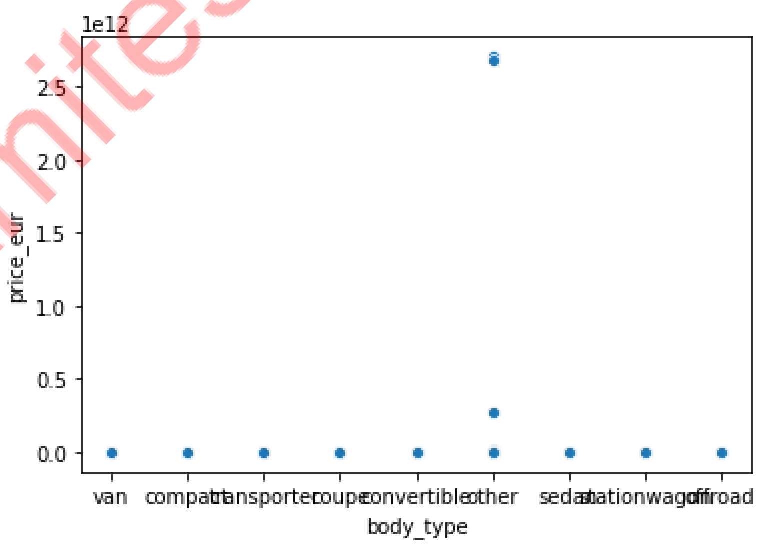
In [16]:
```python
# checking categotical vaues for Body_type and sort by descending
df_car['fuel_type'].value_counts(sort=True)
```

Out[16]:
```
gasoline    902222
diesel      768207
electric     26350
lpg           7403
cng           1124
Name: fuel_type, dtype: int64
```

In [25]:
```
sns.scatterplot(x='fuel_type',y='price_eur',data=df_car)
```
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1402c0bd400>



In [26]:
```
sns.scatterplot(x='body_type',y='price_eur',data=df_car)
```
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x13f902d11d0>



In [17]:
```
df_car.shape   # latest size before filtering rows for NAN values for 'body_type
```

Out[17]: (3552912, 13)

In [27]:
```
#Imputing nan's since the dataset is very large deleting rows where we have nulls
df_car = df_car[df_car['fuel_type'].notna()]
df_car = df_car[df_car['body_type'].notna()]
```

In [28]:
```
df_car.shape   # latest size
```

Out[28]: (582392, 13)

In [29]:
```
df_car['body_type'].value_counts()
```

Out[29]:
```
compact          241948
other            122159
coupe             68738
stationwagon      68092
van               30728
offroad           21835
sedan             19149
convertible        5303
transporter        4440
Name: body_type, dtype: int64
```

In [30]:
```
df_car['seat_count'].value_counts()
```

Out[30]:
```
None      237765
5.0       160410
5          13172
0.0        11322
7.0         9805
4.0         9043
4           6372
3.0         4346
2.0         3841
2           2869
7           2467
3           1729
9.0         1721
6.0         1517
8.0          886
6            462
9            380
1.0          190
8            156
1             11
21.0           9
50.0           5
14.0           5
51.0           4
18.0           4
57             4
52.0           3
15.0           3
16.0           3
17.0           3
81.0           3
29.0           3
53.0           2
512.0          2
58.0           2
32.0           2
33.0           2
36.0           2
57.0           2
44.0           2
13.0           2
49.0           2
55.0           2
17             2
10.0           1
30.0           1
19.0           1
23.0           1
25.0           1
517.0          1
43.0           1
54.0           1
59.0           1
85.0           1
138.0          1
```

```
515.0         1
45.0          1
Name: seat_count, dtype: int64
```

In [31]:
```
df_car['door_count'].value_counts()
```

Out[31]:
```
None    233523
5.0     160334
4.0      17846
5        13231
3.0       9338
0.0       7668
2         7290
4         7053
3         4242
2.0       3839
6.0        229
1.0         89
6           41
7.0         15
55.0         9
1            8
58.0         2
9.0          2
8.0          2
7            1
17.0         1
45.0         1
49.0         1
22.0         1
Name: door_count, dtype: int64
```

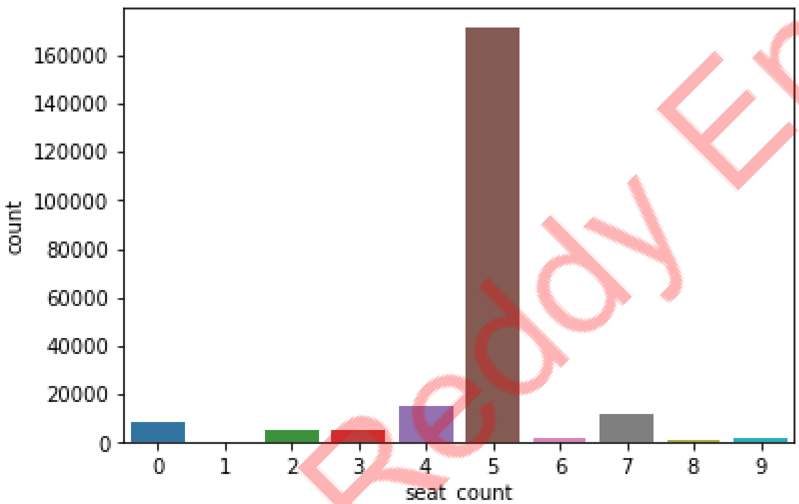
In [32]:
```
#drop all the None values
df_car = df_car[df_car['seat_count'] != 'None']
df_car = df_car[df_car['door_count'] != 'None']

#convert everything to numeric
df_car[['seat_count', 'door_count']] = df_car[['seat_count', 'door_count']].appl

df_car = df_car[df_car['seat_count'] <= 9] #drop values greater than 9
df_car['seat_count'] = df_car['seat_count'].astype(np.int64)
df_car = df_car[df_car['door_count'] <= 7] #drop values greater than 7
df_car['door_count'] = df_car['door_count'].astype(np.int64)
```

In [33]:
```
df_car['seat_count'].value_counts().sort_index()
```

Out[33]:
```
0      8629
1       198
2      5095
3      5251
4     15148
5    170986
6      1893
7     12023
8       991
9      2018
Name: seat_count, dtype: int64
```

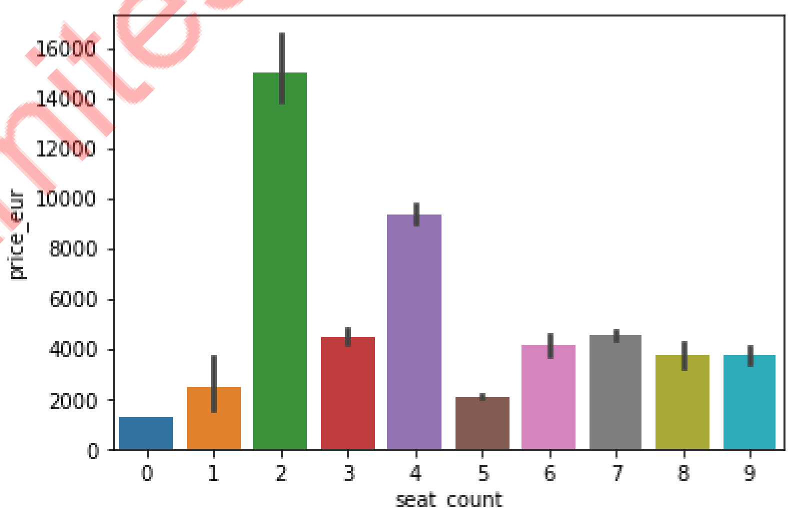
In [34]:
```
sns.countplot(x="seat_count",data=df_car)
```

Out[34]: `<matplotlib.axes._subplots.AxesSubplot at 0x13fe151bbe0>`



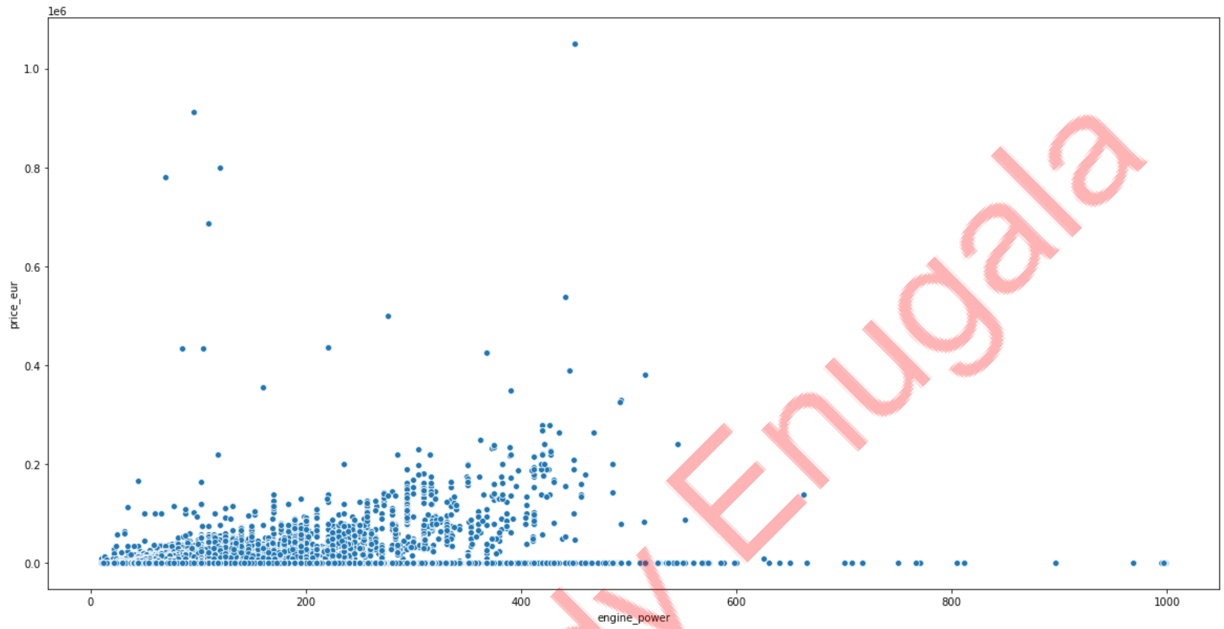
In [35]:
```
sns.barplot(x='seat_count',y='price_eur',data=df_car)  #aggregate the data by me
```

Out[35]: `<matplotlib.axes._subplots.AxesSubplot at 0x13fe14f82e8>`



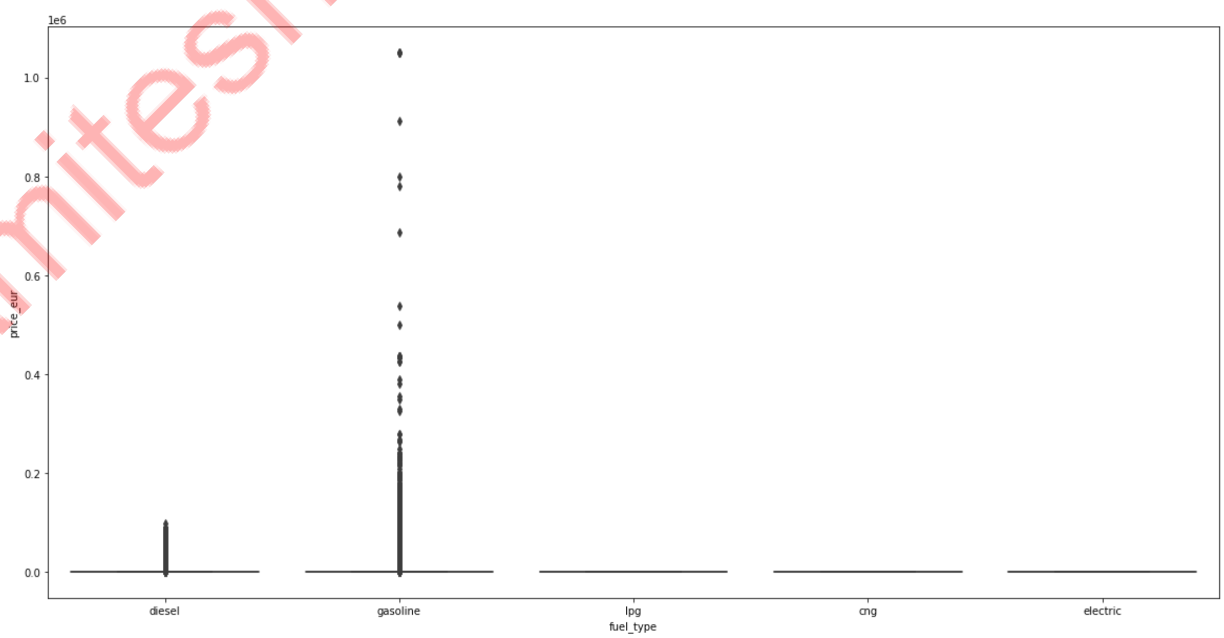
In [36]:
```
#Scatter plot Engine_power verss Price_eur (target)
a4_dims = (20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.scatterplot(x='engine_power',y='price_eur',data=df_car)
```

Out[36]: <matplotlib.axes._subplots.AxesSubplot at 0x13fe16407f0>



In [37]:
```
#box plot fuel_type verss Price_eur (target)
a4_dims = (20,10)
fig, ax = plt.subplots(figsize=a4_dims)
sns.boxplot(x='fuel_type',y='price_eur',data=df_car)
```

Out[37]: <matplotlib.axes._subplots.AxesSubplot at 0x13fe16c8358>

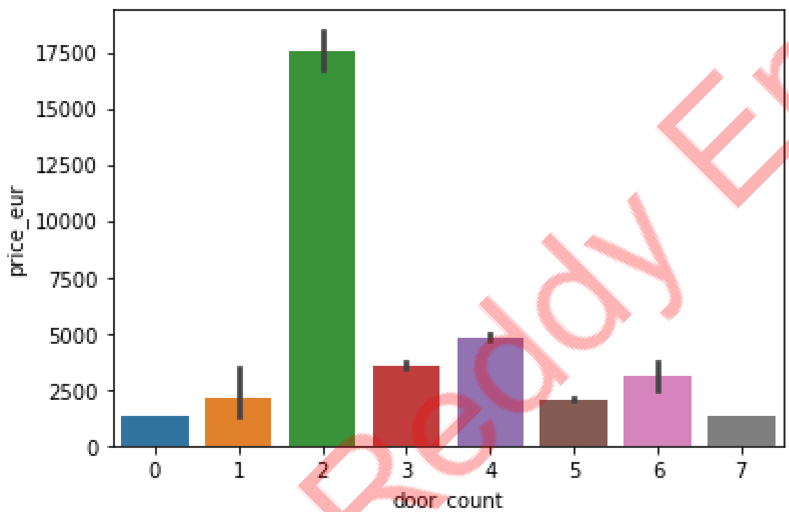


In [38]:
```
df_car['door_count'].value_counts().sort_index()
```

Out[38]:
```
0      7653
1        85
2     10010
3     12587
4     23968
5    167650
6       264
7        15
Name: door_count, dtype: int64
```

In [39]:
```
#door_count vs Price_eur relationship
sns.barplot(x='door_count',y='price_eur',data=df_car)  #aggregate the data by mea
```

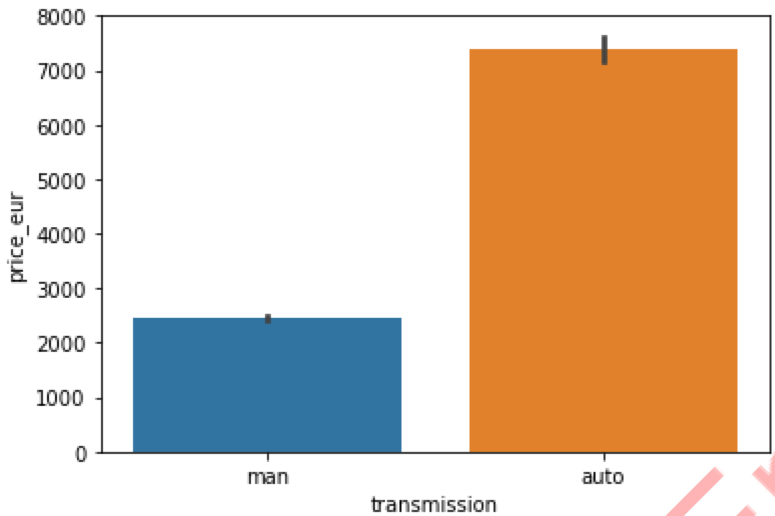
Out[39]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x13fe17849e8>
```



In [40]:
```
#imputing Data
df_car['maker'] = df_car['maker'].fillna(df_car['maker'].mode()[0])
df_car['transmission'] = df_car['transmission'].fillna(df_car['transmission'].mod

df_car['manufacture_year'] = df_car['manufacture_year'].fillna(df_car['manufactur
df_car['manufacture_year'] = df_car['manufacture_year'].astype(np.int64)

df_car = df_car[df_car['mileage'] != 0] # remove cars without any mileage
df_car['mileage'] = df_car['mileage'].fillna(df_car['mileage'].mean())
df_car['mileage (log)'] = np.log(df_car['mileage'])
df_car = df_car.drop(columns=['mileage'])

df_car['engine_displacement'] = df_car['engine_displacement'].fillna(df_car['engi
df_car['engine_power'] = df_car['engine_power'].fillna(df_car['engine_power'].mea
df_car['engine_displacement (log)'] = np.log(df_car['engine_displacement'])
df_car = df_car.drop(columns=['engine_displacement'])
```

In [41]:
```
##transmission vs Price_eur relationship

sns.barplot(x='transmission',y='price_eur',data=df_car)  #aggregate the data by r
```

Out[41]: <matplotlib.axes._subplots.AxesSubplot at 0x13fe1d523c8>



In [42]:
```
df_car[['date_created','date_last_seen']] = df_car[['date_created','date_last_se
df_car['ad_day_count'] = (df_car['date_last_seen'] - df_car['date_created']).dt.
df_car = df_car.drop(columns =['date_created','date_last_seen'])
```

In [43]:
```
import plotly.figure_factory as ff
from tqdm import tqdm
import plotly.offline as py
import plotly.express as px
def pie_plot(variable):
    """
        input: variable ex: "Model"
        output: pie plot & value count
    """
    #get feature
    var = df_car[variable]
    #count number of categorical variable(value/sample)
    varValue = var.value_counts()
    #visualize
    fig = px.pie(values=varValue,names=varValue.index,template="plotly_white",ti
    fig.update_traces(rotation=90, pull=0.05, textinfo="percent+label")
    fig.show()
    print("{}:\n{}".format(variable,varValue))
```

In [44]:
```
df_car.isnull().sum()
```

Out[44]:
```
maker                        0
manufacture_year             0
engine_power                 0
body_type                    0
transmission                 0
door_count                   0
seat_count                   0
fuel_type                    0
price_eur                    0
mileage (log)                0
engine_displacement (log)    0
ad_day_count                 0
dtype: int64
```

In [45]:
```
df_car.head(10)
```

Out[45]:

	maker	manufacture_year	engine_power	body_type	transmission	door_count	seat_coun
507	mercedes-benz	2011	120.0	van	man	5	
577	skoda	2007	96.0	van	man	5	
583	skoda	2005	128.0	van	man	5	
898	ford	2011	103.0	van	man	5	
972	skoda	2016	79.0	van	man	5	
1007	hyundai	2002	90.0	van	man	5	
1038	skoda	2016	55.0	van	man	3	
1079	chrysler	2012	214.0	van	auto	5	
1320	skoda	2004	96.0	van	man	4	
1603	fiat	2006	77.0	van	man	4	

In [46]:
```
#Creating Dummies for categorical 'maker'
df_car = pd.concat([df_car,pd.get_dummies(df_car.maker)],axis="columns")
df_car = df_car.drop(columns=['maker'])    #
df_car = df_car.drop(columns=['manufacture_year'])
```

In [47]:
```
df_car = pd.concat([df_car,pd.get_dummies(df_car.body_type)],axis="columns")
df_car = df_car.drop(columns=['body_type'])
```

In [48]:
```
df_car.head(10)
```

Out[48]:

	engine_power	transmission	door_count	seat_count	fuel_type	price_eur	mileage (log)	engine
507	120.0	man	5	5	diesel	22168.76	12.223878	
577	96.0	man	5	9	diesel	8475.20	12.409013	
583	128.0	man	5	8	diesel	9215.40	12.421184	
898	103.0	man	5	5	diesel	9437.45	11.830040	
972	79.0	man	5	5	diesel	4441.15	12.055250	
1007	90.0	man	5	5	gasoline	1073.28	12.111762	
1038	55.0	man	3	2	diesel	5162.84	11.716307	
1079	214.0	auto	5	7	gasoline	18467.80	11.718312	
1320	96.0	man	4	6	diesel	8845.30	11.957611	
1603	77.0	man	4	5	diesel	3515.91	11.955192	

10 rows × 64 columns

In [49]:
```
df_car['transmission'] = df_car['transmission'].map( {'man': 1, 'auto': 0} ).ast
```

In [50]:
```
df_car = pd.concat([df_car,pd.get_dummies(df_car.door_count,prefix='door_count')
df_car = df_car.drop(columns=['door_count'])
```

In [51]:
```
df_car = pd.concat([df_car,pd.get_dummies(df_car.seat_count,prefix='seat_count')
df_car = df_car.drop(columns=['seat_count'])
```

In [52]:
```
df_car = pd.concat([df_car,pd.get_dummies(df_car.fuel_type)],axis="columns")
df_car = df_car.drop(columns=['fuel_type'])
```

In [53]:
```
df_car.head(10)
```

Out[53]:

	engine_power	transmission	price_eur	mileage (log)	engine_displacement (log)	ad_day_count	alfa rome
507	120.0	1	22168.76	12.223878	7.669962	74	
577	96.0	1	8475.20	12.409013	7.808323	74	
583	128.0	1	9215.40	12.421184	7.808323	74	
898	103.0	1	9437.45	11.830040	7.600902	74	
972	79.0	1	4441.15	12.055250	7.599401	74	
1007	90.0	1	1073.28	12.111762	7.492760	74	
1038	55.0	1	5162.84	11.716307	7.286876	74	
1079	214.0	0	18467.80	11.718312	8.189800	74	
1320	96.0	1	8845.30	11.957611	7.824046	74	
1603	77.0	1	3515.91	11.955192	7.554859	74	

10 rows × 84 columns

In [54]:
```
df_car.shape
```

Out[54]:
```
(215738, 84)
```

In [55]:
```
df_car_subset = df_car.sample(n = 25000, random_state=0)
df_car_subset = df_car_subset.loc[:, (df_car_subset != 0).any(axis=0)]
```

In [59]:
```
#regress sample
X = df_car_subset.loc[:, df_car_subset.columns != 'price_eur']
y = np.log(df_car_subset['price_eur'])
```

Regression Algorithms

In [60]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import r2_score
```

In [61]:
```
X_train_org, X_test_org, y_train, y_test = train_test_split(X,y,random_state=0)

sc = StandardScaler()
X_train = sc.fit_transform(X_train_org)
X_test = sc.transform(X_test_org)
```

Linear Regression

In [62]:
```
from sklearn.linear_model import LinearRegression

lr = LinearRegression()
lr.fit(X_train, y_train)
lr_score = lr.score(X_test, y_test)
print(lr.score(X_train, y_train))
print(lr.score(X_test, y_test))
```

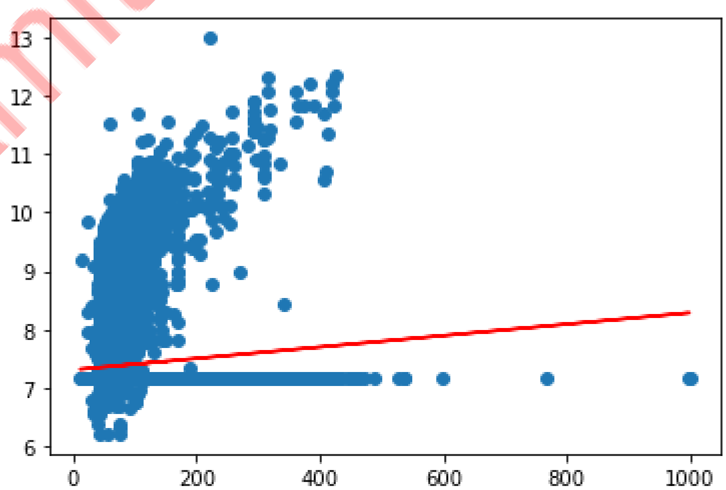
```
0.504814475849747
0.4947212251201301
```

In [63]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(lr , X_train, y_trai
scores = cross_val_score(lr , X_train, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[-1.06816034e+26 -7.29699300e+23  5.03942825e-01 -3.31801229e+21
 -1.82035575e+23  5.19440499e-01]
-1.7955181220289719e+25
```

In [64]:
```
#PLOT

%matplotlib inline
import matplotlib.pyplot as plt

X_train_array = X_train_org.to_numpy()

X_train_rm = X_train_array[:,0].reshape(-1,1)
lr.fit(X_train_rm, y_train)
y_predict = lr.predict(X_train_rm)

plt.plot(X_train_rm, y_predict, c = 'r')
plt.scatter(X_train_rm,y_train)
```

Out[64]: <matplotlib.collections.PathCollection at 0x14018510550>



Lasso Regression

In [65]:
```
grid_parms_lasso = {'alpha': [0.01, 0.1, 1, 10,100]}
```

In [66]:
```
lasso = Lasso()
grid_search_lasso = GridSearchCV(estimator = lasso,param_grid = grid_parms_lasso
grid_search_lasso.fit(X_train, y_train)
print("Best parameters: {}".format(grid_search_lasso.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search_lasso.best_score_
```

```
Best parameters: {'alpha': 0.01}
Best cross-validation score: 0.4899
```

In [67]:
```
lass = Lasso(alpha = 0.01)
lass.fit(X_train, y_train)
print(lass.score(X_train, y_train))
print(lass.score(X_test, y_test))
```

```
0.4968545687064581
0.48782113624813
```

In [68]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(lass , X_train, y_tra
scores = cross_val_score(lass , X_train, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.49257331 0.47311011 0.50059249 0.47296707 0.49475928 0.51481229]
0.49146909285165513
```

In [69]:
```
result_lasso = pd.DataFrame(grid_search_lasso.cv_results_)
result_lasso
```

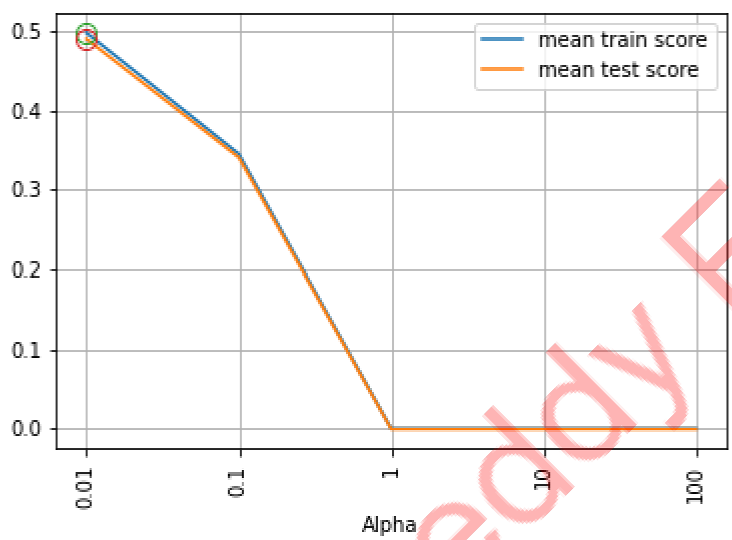
Out[69]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_
0	0.162767	0.010218	0.003790	0.000399	0.01	{'alpha': 0.01}	0.4
1	0.122872	0.013072	0.003391	0.000489	0.1	{'alpha': 0.1}	0.3
2	0.103921	0.005023	0.002793	0.000747	1	{'alpha': 1}	-0.0
3	0.096343	0.008431	0.003191	0.000399	10	{'alpha': 10}	-0.0
4	0.090756	0.011511	0.001796	0.000746	100	{'alpha': 100}	-0.0

5 rows × 21 columns

In [70]:
```
%matplotlib inline

plt.plot(range(result_lasso.shape[0]), result_lasso['mean_train_score'], label =
plt.plot(range(result_lasso.shape[0]), result_lasso['mean_test_score'], label =
plt.xticks(range(result_lasso.shape[0]), result_lasso['param_alpha'], rotation =
plt.plot([grid_search_lasso.best_index_], result_lasso['mean_train_score'][grid_
plt.plot([grid_search_lasso.best_index_], result_lasso['mean_test_score'][grid_s
plt.grid()
plt.legend()
plt.xlabel('Alpha')
```

Out[70]: Text(0.5, 0, 'Alpha')



Ridge Regression

In [71]:
```
grid_parms_ridge = {'alpha': [0.01, 0.1, 1, 10, 100]}
```

In [72]:
```
ridge = Ridge()
grid_search_ridge = GridSearchCV(estimator = ridge,param_grid = grid_parms_ridge
grid_search_ridge.fit(X_train, y_train)
print("Best parameters: {}".format(grid_search_ridge.best_params_))

print("Best cross-validation score: {:.4f}".format(grid_search_ridge.best_score_
```

```
Best parameters: {'alpha': 100}
Best cross-validation score: 0.4942
```

In [73]:
```
ridge = Ridge(alpha = 100)
ridge.fit(X_train, y_train)
print(ridge.score(X_train, y_train))
print(ridge.score(X_test, y_test))
```

```
0.5048065545958941
0.49479695877804675
```

In [74]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(ridge , X_train, y_tı
scores = cross_val_score(ridge , X_train, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.4943882  0.48582766 0.50401297 0.4774348  0.49361643 0.5195419 ]
0.4958036598049622
```

In [75]:
```
result_ridge = pd.DataFrame(grid_search_ridge.cv_results_)
result_ridge
```

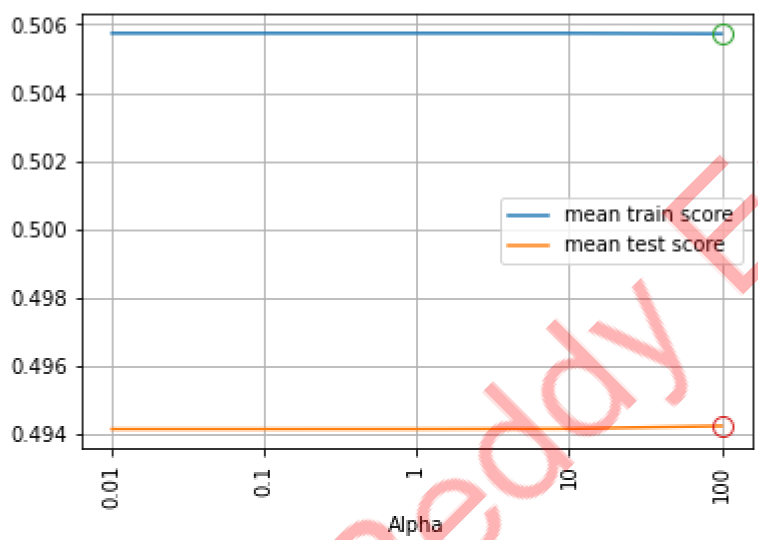
Out[75]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_alpha	params	split0_test_
0	0.202238	0.003277	0.003789	0.000400	0.01	{'alpha': 0.01}	0.4
1	0.143485	0.061234	0.002991	0.000631	0.1	{'alpha': 0.1}	0.4
2	0.062036	0.001163	0.002792	0.000399	1	{'alpha': 1}	0.4
3	0.074007	0.017754	0.002194	0.000746	10	{'alpha': 10}	0.4
4	0.068024	0.009952	0.002394	0.000797	100	{'alpha': 100}	0.4

5 rows × 21 columns

In [76]:
```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(range(result_ridge.shape[0]), result_ridge['mean_train_score'], label =
plt.plot(range(result_ridge.shape[0]), result_ridge['mean_test_score'], label =
plt.xticks(range(result_ridge.shape[0]), result_ridge['param_alpha'], rotation =
plt.plot([grid_search_ridge.best_index_], result_ridge['mean_train_score'][grid_
plt.plot([grid_search_ridge.best_index_], result_ridge['mean_test_score'][grid_s
plt.grid()
plt.legend()
plt.xlabel('Alpha')
```

Out[76]: Text(0.5, 0, 'Alpha')



Polynomial Regression

In [77]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression
from sklearn.pipeline import make_pipeline

def PolynomialRegression(degree=2, **kwargs):
    return make_pipeline(PolynomialFeatures(degree),
                         LinearRegression(**kwargs))
```

In [78]:
```
param_grid_poly = {'polynomialfeatures__degree': np.arange(3)}

grid_poly = GridSearchCV(PolynomialRegression(), param_grid_poly,return_train_sc
```

In [79]:
```
grid_poly.fit(X_train, y_train)
```

Out[79]:
```
GridSearchCV(cv=5, error_score=nan,
             estimator=Pipeline(memory=None,
                                steps=[('polynomialfeatures',
                                        PolynomialFeatures(degree=2,
                                                           include_bias=True,
                                                           interaction_only=Fal
se,
                                                           order='C')),
                                       ('linearregression',
                                        LinearRegression(copy_X=True,
                                                         fit_intercept=True,
                                                         n_jobs=None,
                                                         normalize=False))],
                                verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'polynomialfeatures__degree': array([0, 1, 2])},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [80]:
```
print("Best parameters: {}".format(grid_poly.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_poly.best_score_))
```

```
Best parameters: {'polynomialfeatures__degree': 2}
Best cross-validation score: -2157856213069988992.0000
```

In [91]:
```
pol = PolynomialFeatures(degree = 2)
X_pol = pol.fit_transform(X_train_org)
Xt_pol = pol.fit_transform(X_test_org)
pol_reg = LinearRegression()
pol_reg.fit(X_pol,y_train)
print(pol_reg.score(X_pol, y_train))
print(pol_reg.score(Xt_pol, y_test))
```

```
0.7634865802020515
-963743283.376124
```

In [92]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(pol_reg , X_pol, y_tı
scores = cross_val_score(pol_reg , X_pol, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[-3.97810910e+12 -2.27737285e+13 -6.58845560e+09 -4.06390112e+12
 -5.83225835e+10 -6.44889883e+08]
-5146882444672.452
```

In [93]:
```
result_poly = pd.DataFrame(grid_poly.cv_results_)
result_poly
```

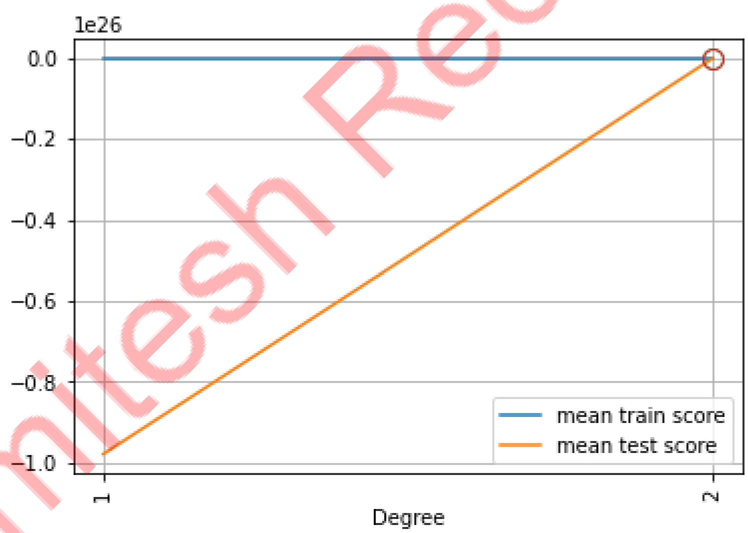
Out[93]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_polynomialfeatures__degree
0	0.060041	0.005725	0.000000	0.000000	0
1	0.186502	0.010888	0.011968	0.001411	1
2	71.289975	0.450231	0.277060	0.036674	2

3 rows × 21 columns

In [94]:
```
plt.plot(range(result_poly.shape[0]), result_poly['mean_train_score'], label = 'm
plt.plot(range(result_poly.shape[0]), result_poly['mean_test_score'], label = 'me
plt.xticks(range(result_poly.shape[0]), result_poly['param_polynomialfeatures__de
plt.plot([grid_poly.best_index_], result_poly['mean_train_score'][grid_poly.best_
plt.plot([grid_poly.best_index_], result_poly['mean_test_score'][grid_poly.best_i
plt.grid()
plt.xlabel('Degree')
plt.legend()
```

Out[94]: <matplotlib.legend.Legend at 0x1401d715dd8>



KNN Regressor

In [95]:
```
grid_parms_knn = {'n_neighbors':[1,5,10,15,20]}
```

In [96]:
```
knn = KNeighborsRegressor()
grid_search_knn = GridSearchCV(knn, grid_parms_knn,cv=6,return_train_score=True,n
grid_search_knn.fit(X_train, y_train)
```

Out[96]:
```
GridSearchCV(cv=6, error_score=nan,
             estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30,
                                           metric='minkowski',
                                           metric_params=None, n_jobs=None,
                                           n_neighbors=5, p=2,
                                           weights='uniform'),
             iid='deprecated', n_jobs=-1,
             param_grid={'n_neighbors': [1, 5, 10, 15, 20]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [97]:
```
print("Best parameters: {}".format(grid_search_knn.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_search_knn.best_score_))
pd.DataFrame(grid_search_knn.cv_results_)
```

```
Best parameters: {'n_neighbors': 5}
Best cross-validation score: 0.6518
```

Out[97]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_n_neighbors	params
0	1.958930	0.174267	8.738965	0.142655	1	{'n_neighbors': 1}
1	1.800351	0.262096	9.395876	0.226932	5	{'n_neighbors': 5}
2	1.421033	0.110027	9.656679	0.207227	10	{'n_neighbors': 10}
3	1.368007	0.068856	9.728154	0.241581	15	{'n_neighbors': 15}
4	1.443639	0.027227	9.368781	0.358239	20	{'n_neighbors': 20}

5 rows × 23 columns

In [98]:
```
knn = KNeighborsRegressor(n_neighbors = 10)
knn.fit(X_train, y_train)
print(knn.score(X_train, y_train))
print(knn.score(X_test, y_test))
```

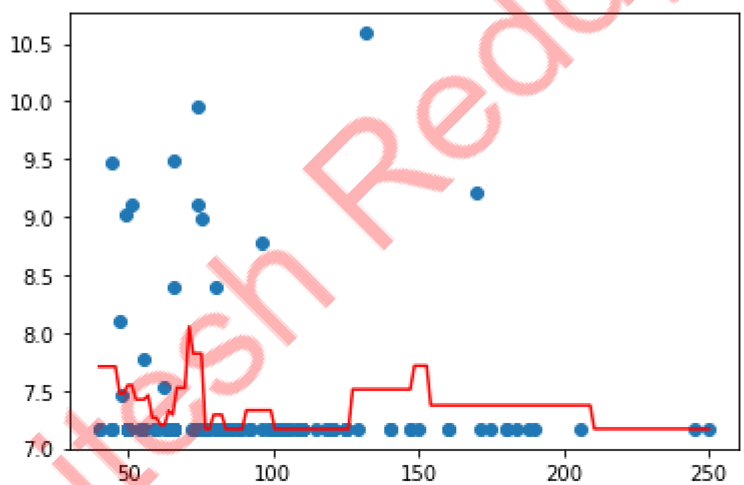
```
0.71304983776784
0.6396830216868226
```

In [99]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(knn , X_train, y_tra
scores = cross_val_score(knn , X_train, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.62608255 0.61033277 0.64452814 0.63660802 0.60934938 0.64486792]
0.6286281287553287
```

In [100]:
```
X_b = X_train_array[:150,0].reshape(-1,1)
y_b = y_train[:150]

knn_reg = KNeighborsRegressor(10)
knn_reg.fit(X_b, y_b)

X_new=np.linspace(X_b.min(), X_b.max(), 150).reshape(150, 1)
y_predict = knn_reg.predict(X_new)

plt.plot(X_new, y_predict, c = 'r')
plt.scatter(X_b, y_b)
```

Out[100]: <matplotlib.collections.PathCollection at 0x1401a7fd710>



Simple SVM

In [101]:
```
grid_parms_svrl = {'C': [0.01, 0.1, 1, 10, 100], 'epsilon' : [0.01, 0.1, 1, 10, 
```

In [102]:
```
linearsvr = LinearSVR()
grid_svrl = GridSearchCV(estimator = linearsvr,param_grid = grid_parms_svrl,retu
```

In [113]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [114]:
```
grid_svrl.fit(X_train,y_train)
```

Out[114]:
```
GridSearchCV(cv=10, error_score=nan,
             estimator=LinearSVR(C=1.0, dual=True, epsilon=0.0,
                                 fit_intercept=True, intercept_scaling=1.0,
                                 loss='epsilon_insensitive', max_iter=1000,
                                 random_state=None, tol=0.0001, verbose=0),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10, 100],
                         'epsilon': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [115]:
```
print("Best parameters: {}".format(grid_svrl.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_svrl.best_score_))
```

```
Best parameters: {'C': 1, 'epsilon': 0.1}
Best cross-validation score: 0.3584
```

In [116]:
```
lsvr = LinearSVR(C = 1, epsilon = 0.1)

lsvr.fit(X_train, y_train)

print(lsvr.score(X_train, y_train))
print(lsvr.score(X_test, y_test))
```

```
0.3985325193389513
0.3832208403212748
```

In [117]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score


kfold = KFold(n_splits=10)
print("Cross-validation scores:\n{}".format(cross_val_score(lsvr , X_train, y_tr
scores = cross_val_score(lsvr, X_train, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.36991843 0.31113101 0.35329707 0.35765491 0.38398139 0.36139695
 0.30006746 0.31737761 0.46723395 0.38949682]
0.3576358618901767
```

In [118]:
```
result_linearsvr = pd.DataFrame(grid_svrl.cv_results_)
result_linearsvr
```

Out[118]:

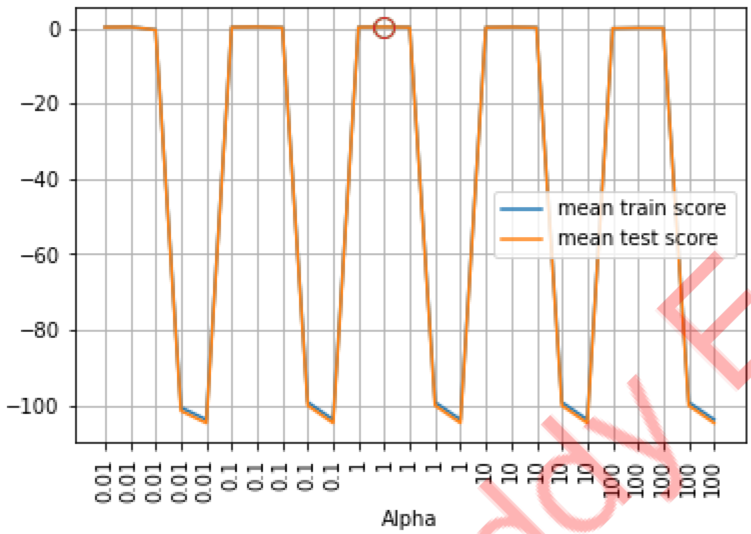
	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_epsilon	param
0	8.126272	0.488445	0.001796	5.980971e-04	0.01	0.01	{'C 0.0 'epsilon 0.0
1	2.407463	0.276146	0.001895	2.993831e-04	0.01	0.1	{'C 0.0 'epsilon 0.
2	1.217943	0.186802	0.001895	2.991522e-04	0.01	1	{'C 0.0 'epsilon
3	0.122970	0.012703	0.001896	2.984932e-04	0.01	10	{'C 0.0 'epsilon 1(
4	0.097039	0.013329	0.002494	2.196499e-03	0.01	100	{'C 0.0 'epsilon 10(
5	12.619156	0.290892	0.001696	4.570598e-04	0.1	0.01	{'C': 0. 'epsilon 0.0
6	10.666877	0.863332	0.001796	3.989699e-04	0.1	0.1	{'C': 0. 'epsilon 0.
7	6.686720	1.842683	0.001896	2.993615e-04	0.1	1	{'C': 0. 'epsilon
8	0.127658	0.009196	0.001996	9.655217e-07	0.1	10	{'C': 0. 'epsilon 1(
9	0.098637	0.013473	0.001896	2.995993e-04	0.1	100	{'C': 0. 'epsilon 10(
10	14.224763	0.312129	0.002095	1.041567e-03	1	0.01	{'C': 'epsilon 0.0
11	13.513665	0.307672	0.001895	2.991685e-04	1	0.1	{'C': 'epsilon 0.
12	12.872578	0.306976	0.001895	2.987558e-04	1	1	{'C': 'epsilon
13	0.141122	0.023022	0.001696	4.570336e-04	1	10	{'C': 'epsilon 1(
14	0.086169	0.002531	0.001796	3.987558e-04	1	100	{'C': 'epsilon 10(

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_epsilon	param
15	14.928182	0.446323	0.001895	2.995505e-04	10	0.01	{'C': 1('epsilon 0.0'
16	15.114086	0.241788	0.001895	2.993271e-04	10	0.1	{'C': 1('epsilon 0.'
17	12.934314	0.252427	0.001995	1.012648e-06	10	1	{'C': 1('epsilon
18	0.133344	0.020763	0.001696	4.569611e-04	10	10	{'C': 1('epsilon 1(
19	0.101228	0.012491	0.001896	2.991685e-04	10	100	{'C': 1('epsilon 10(
20	15.450287	0.351933	0.001696	4.571476e-04	100	0.01	{'C 10('epsilon 0.0'
21	14.950422	0.308257	0.001995	1.029350e-06	100	0.1	{'C 10('epsilon 0.'
22	11.553505	1.648973	0.001696	4.570595e-04	100	1	{'C 10('epsilon
23	0.129553	0.023160	0.001896	5.363931e-04	100	10	{'C 10('epsilon 1(
24	0.092052	0.010278	0.001697	4.564822e-04	100	100	{'C 10('epsilon 10(

25 rows × 32 columns

In [119]:
```
plt.plot(range(result_linearsvr.shape[0]), result_linearsvr['mean_train_score'],
plt.plot(range(result_linearsvr.shape[0]), result_linearsvr['mean_test_score'], 
plt.xticks(range(result_linearsvr.shape[0]), result_linearsvr['param_C'], rotati
plt.plot([grid_svrl.best_index_], result_linearsvr['mean_train_score'][grid_svrl
plt.plot([grid_svrl.best_index_], result_linearsvr['mean_test_score'][grid_svrl.
plt.grid()
plt.legend()
plt.xlabel('Alpha')
```

Out[119]: Text(0.5, 0, 'Alpha')



SVM with kernels

In [120]:
```
#Linear
```

In [121]:
```
from sklearn.svm import SVR, LinearSVR

svm_simple_params = {'C':[0.01,0.1,1,10,100],
                     'epsilon':[0.01,0.1,1,10,100]
                    }

svm_simple_reg = LinearSVR()
svm_simple = GridSearchCV(estimator=svm_simple_reg,param_grid=svm_simple_params,

svm_simple.fit(X, y)
```

Out[121]:
```
GridSearchCV(cv=2, error_score=nan,
             estimator=LinearSVR(C=1.0, dual=True, epsilon=0.0,
                                 fit_intercept=True, intercept_scaling=1.0,
                                 loss='epsilon_insensitive', max_iter=1000,
                                 random_state=None, tol=0.0001, verbose=0),
             iid='deprecated', n_jobs=None,
             param_grid={'C': [0.01, 0.1, 1, 10, 100],
                         'epsilon': [0.01, 0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [122]:
```
#evaluate the model
print(svm_simple.score(X_train, y_train))
print(svm_simple.score(X_test, y_test))
```

```
-36.009788690212034
-34.070436057767836
```

In [123]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score

kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(svm_simple , X_train
scores = cross_val_score(svm_simple , X_train, y_train, cv=kfold)
linear_score = np.mean(scores)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.36661283 0.30175356 0.38774296 0.31191419 0.36775562 0.4341616 ]
0.3610014631781489
```

In [124]:
```
result_svr_linear = pd.DataFrame(svm_simple.cv_results_)
result_svr_linear
```

Out[124]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_epsilon	param
0	1.006806	0.012465	0.007480	4.984140e-04	0.01	0.01	{'C 0.0 'epsilon 0.0
1	0.961928	0.013463	0.007979	1.192093e-07	0.01	0.1	{'C 0.0 'epsilon 0.
2	0.864189	0.001496	0.007480	4.984140e-04	0.01	1	{'C 0.0 'epsilon
3	0.040890	0.011968	0.007979	9.970665e-04	0.01	10	{'C 0.0 'epsilon 1
4	0.024934	0.001995	0.007978	9.973049e-04	0.01	100	{'C 0.0 'epsilon 10
5	0.989353	0.000997	0.007478	4.988909e-04	0.1	0.01	{'C': 0. 'epsilon 0.0
6	1.048197	0.037899	0.006981	8.344650e-07	0.1	0.1	{'C': 0. 'epsilon 0.
7	0.888126	0.000498	0.007480	4.981756e-04	0.1	1	{'C': 0. 'epsilon
8	0.040406	0.014476	0.007979	9.983778e-04	0.1	10	{'C': 0. 'epsilon 1
9	0.024434	0.001496	0.007979	9.976625e-04	0.1	100	{'C': 0. 'epsilon 10
10	1.002319	0.000001	0.007480	4.976988e-04	1	0.01	{'C': 'epsilon 0.0
11	1.031742	0.000498	0.007977	2.384186e-07	1	0.1	{'C': 'epsilon 0.
12	0.934998	0.004489	0.007480	4.986525e-04	1	1	{'C': 'epsilon
13	0.034408	0.010472	0.008476	1.494646e-03	1	10	{'C': 'epsilon 1
14	0.030916	0.000997	0.009475	4.988909e-04	1	100	{'C': 'epsilon 10

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_epsilon	param
15	1.152418	0.093251	0.007482	4.993677e-04	10	0.01	{'C': 1 'epsilon 0.0
16	1.095572	0.033411	0.007977	3.576279e-07	10	0.1	{'C': 1 'epsilon 0.
17	0.918543	0.001994	0.007480	4.986525e-04	10	1	{'C': 1 'epsilon
18	0.042387	0.012467	0.008477	1.496077e-03	10	10	{'C': 1 'epsilon 1
19	0.031415	0.001496	0.008478	1.495719e-03	10	100	{'C': 1 'epsilon 10
20	1.139966	0.020930	0.008462	5.142689e-04	100	0.01	{'C 10 'epsilon 0.0
21	1.041216	0.029919	0.007479	4.979372e-04	100	0.1	{'C 10 'epsilon 0.
22	0.900093	0.016456	0.007479	4.996061e-04	100	1	{'C 10 'epsilon
23	0.034906	0.004986	0.008976	9.979010e-04	100	10	{'C 10 'epsilon 1
24	0.029422	0.001495	0.008976	0.000000e+00	100	100	{'C 10 'epsilon 10

In [ ]:

In [125]:
```
#SVR with Kernel=Poly
```

In [126]:
```
grid_parms_svrp = {'C': [1, 10, 100],'degree':[1,3]}

svr_poly = SVR(kernel='poly')
grid_svr_poly = GridSearchCV(estimator = svr_poly,param_grid = grid_parms_svrp,r

grid_svr_poly.fit(X_train,y_train)
```

Out[126]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='scale', kernel='poly',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [1, 10, 100], 'degree': [1, 3]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [127]:
```
print("Best parameters: {}".format(grid_svr_poly.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_svr_poly.best_score_))
pd.DataFrame(grid_svr_poly.cv_results_)
```

```
Best parameters: {'C': 1, 'degree': 3}
Best cross-validation score: 0.6327
```

Out[127]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_degree	params
0	11.400849	0.594278	2.111356	0.027398	1	1	{'C': 1, 'degree': 1}
1	11.381900	0.708704	1.935824	0.027507	1	3	{'C': 1, 'degree': 3}
2	23.829281	3.180959	2.118004	0.393854	10	1	{'C': 10, 'degree': 1}
3	27.288698	0.904935	2.253308	0.063629	10	3	{'C': 10, 'degree': 3}
4	109.821676	16.080262	1.507969	0.266082	100	1	{'C': 100, 'degree': 1}
5	147.181477	5.941316	1.694127	0.060537	100	3	{'C': 100, 'degree': 3}

In [128]:
```
svr_p = SVR(kernel='poly',C=1,degree = 3)
svr_p.fit(X_train, y_train)
svr_p.score(X_train, y_train)
svr_p.score(X_test, y_test)
```

Out[128]:
```
0.598114070264627
```

In [129]:

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
#scores = cross_val_score(logreg, iris.data, iris.target)
kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(svr_p, X_train, y_tr
scores = cross_val_score(svr_p, X_train, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.6154685  0.6533766  0.67785693 0.64384216 0.6691138  0.63434642]
0.6490007360288077
```

In [130]:

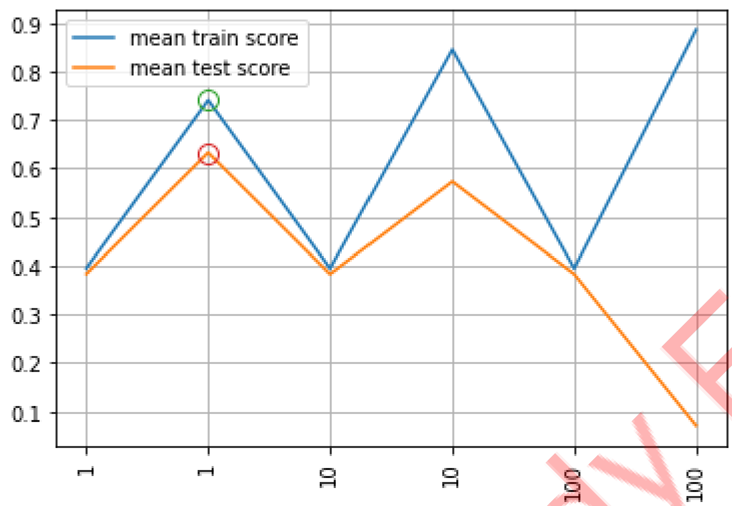
```
result_svr_poly= pd.DataFrame(grid_svr_poly.cv_results_)
result_svr_poly
```

Out[130]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_degree	params
0	11.400849	0.594278	2.111356	0.027398	1	1	{'C': 1, 'degree': 1}
1	11.381900	0.708704	1.935824	0.027507	1	3	{'C': 1, 'degree': 3}
2	23.829281	3.180959	2.118004	0.393854	10	1	{'C': 10, 'degree': 1}
3	27.288698	0.904935	2.253308	0.063629	10	3	{'C': 10, 'degree': 3}
4	109.821676	16.080262	1.507969	0.266082	100	1	{'C': 100, 'degree': 1}
5	147.181477	5.941316	1.694127	0.060537	100	3	{'C': 100, 'degree': 3}

In [131]:
```
plt.plot(range(result_svr_poly.shape[0]), result_svr_poly['mean_train_score'], la
plt.plot(range(result_svr_poly.shape[0]), result_svr_poly['mean_test_score'], lal
plt.xticks(range(result_svr_poly.shape[0]), result_svr_poly['param_C'], rotation
plt.plot([grid_svr_poly.best_index_], result_svr_poly['mean_train_score'][grid_sv
plt.plot([grid_svr_poly.best_index_], result_svr_poly['mean_test_score'][grid_svr
plt.grid()
plt.legend()
```

Out[131]: <matplotlib.legend.Legend at 0x1401dc55e10>



In [ ]:

In [134]:
```
#SVR with Kernel=rbf
```

In [135]:
```
grid_parms_rbf = {'C': [0.1, 1, 10, 100],'gamma':[0.1, 1, 10, 100]}

svr_rbf = SVR(kernel='rbf')
grid_svr_rbf = GridSearchCV(estimator = svr_rbf,param_grid = grid_parms_rbf,retur

grid_svr_rbf.fit(X_train,y_train)
```

Out[135]:
```
GridSearchCV(cv=3, error_score=nan,
             estimator=SVR(C=1.0, cache_size=200, coef0=0.0, degree=3,
                           epsilon=0.1, gamma='scale', kernel='rbf',
                           max_iter=-1, shrinking=True, tol=0.001,
                           verbose=False),
             iid='deprecated', n_jobs=-1,
             param_grid={'C': [0.1, 1, 10, 100], 'gamma': [0.1, 1, 10, 100]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=0)
```

In [136]:
```
print("Best parameters: {}".format(grid_svr_rbf.best_params_))
print("Best cross-validation score: {:.4f}".format(grid_svr_rbf.best_score_))
```

```
Best parameters: {'C': 10, 'gamma': 0.1}
Best cross-validation score: 0.6003
```

In [137]:
```
svr_rbf = SVR(kernel='rbf',C=10,gamma=0.1)
svr_rbf.fit(X_train, y_train)
svr_rbf.score(X_train, y_train)
svr_rbf.score(X_test, y_test)
```

Out[137]:
```
0.62171077470267
```

In [138]:
```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
kfold = KFold(n_splits=6)
print("Cross-validation scores:\n{}".format(cross_val_score(svr_rbf, X_train, y_
scores = cross_val_score(svr_rbf, X_train, y_train, cv=kfold)
print(np.mean(scores))
```

```
Cross-validation scores:
[0.6065262  0.63007108 0.63975189 0.6313607  0.63894711 0.56321822]
0.6183125321906283
```

In [139]:
```
result_rbf = pd.DataFrame(grid_svr_rbf.cv_results_)
result_rbf
```

Out[139]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_C	param_gamma	param
0	13.243920	0.080244	3.926835	0.100205	0.1	0.1	{'C': 0. 'gamma 0.
1	27.210242	0.842308	6.952742	0.182945	0.1	1	{'C': 0. 'gamma
2	99.301474	9.801126	11.956363	0.095391	0.1	10	{'C': 0. 'gamma 1
3	148.238286	14.436809	16.193367	0.387654	0.1	100	{'C': 0. 'gamma 10
4	12.519523	0.097333	3.529562	0.151883	1	0.1	{'C': 'gamma 0.
5	50.292189	6.896945	6.257601	0.231739	1	1	{'C': 'gamma
6	127.087841	10.409320	12.467995	0.166779	1	10	{'C': 'gamma 1
7	190.539178	15.761467	16.254203	0.221916	1	100	{'C': 'gamma 10
8	14.274835	0.361317	3.599702	0.092553	10	0.1	{'C': 1 'gamma 0.
9	76.238808	7.805714	6.896560	0.271700	10	1	{'C': 1 'gamma
10	149.704035	12.165018	12.655493	0.257687	10	10	{'C': 1 'gamma 1
11	189.956401	0.859799	16.188712	0.179071	10	100	{'C': 1 'gamma 10
12	18.629186	0.218625	3.733351	0.062720	100	0.1	{'C': 10 'gamma 0.
13	86.802894	1.736365	6.842703	0.290038	100	1	{'C': 10 'gamma
14	147.293900	23.700062	10.348662	0.158979	100	10	{'C': 10 'gamma 1
15	159.492949	4.739642	10.724656	0.031289	100	100	{'C': 10 'gamma 10

In [140]:
```
plt.plot(range(result_rbf.shape[0]), result_rbf['mean_train_score'], label = 'mea
plt.plot(range(result_rbf.shape[0]), result_rbf['mean_test_score'], label = 'mean
plt.xticks(range(result_svr_poly.shape[0]), result_rbf['param_C'], rotation = 90
plt.plot([grid_svr_rbf.best_index_], result_rbf['mean_train_score'][grid_svr_rbf
plt.plot([grid_svr_rbf.best_index_], result_rbf['mean_test_score'][grid_svr_rbf.
plt.grid()
plt.legend()
```

Out[140]:
```
<matplotlib.legend.Legend at 0x1402cc5a9b0>
```

Regression summary

In [141]:
```
d = {'Model': ['Linear Regression', 'KNN Regression','Ridge Regression','Lasso Re
     'Cross-Validation Score': [lr_score, grid_search_knn.best_score_,grid_searc
```

In [144]:
```
result = pd.DataFrame(data=d)
print(result)
```

```
                     Model  Cross-Validation Score
0        Linear Regression            4.947212e-01
1           KNN Regression            6.517521e-01
2         Ridge Regression            4.942250e-01
3         Lasso Regression            4.898854e-01
4  Polynominal Regression            -2.157856e+19
5               Simple SVR            3.584214e-01
6    SVR with Linear kernel           3.610015e-01
7      SVR with Poly kernel           6.327155e-01
8       SVR with rbf kernel           6.003021e-01
```

In [143]:
```
#sort dataframe
sorted_df = result.sort_values(by='Cross-Validation Score', ascending=False)
print(sorted_df)
```

```
                  Model  Cross-Validation Score
1        KNN Regression            6.517521e-01
7   SVR with Poly kernel            6.327155e-01
8    SVR with rbf kernel            6.003021e-01
0     Linear Regression            4.947212e-01
2      Ridge Regression            4.942250e-01
3      Lasso Regression            4.898854e-01
6  SVR with Linear kernel          3.610015e-01
5            Simple SVR            3.584214e-01
4  Polynominal Regression         -2.157856e+19
```

Conclusion

The KNN regression model works the best for the regressing data. It had a training score of 0.7131, and a test set score of 0.6397. It has the highest cross-validation score.Hence KNN Regression is good model for this regression problem for predicting the price of the car.

In [ ]: