House Price Prediction

TEESSIDE UNIVERSITY

Machine Learning Project

Programmer : Keyhan Azarjoo

DataSet Information:

This dataset has been downloaded from Kaggle database URL :
https://www.kaggle.com/datasets/mokar2001/house-price-tehran-iran

---

Description: this dataset provides data related to house prices in different part of Tehran, Cappital of Iran, the columns are as below:

```
1. Area in square meters

2. Number of bedrooms

3. Has Parking or not

4. Has elevator or not

5. Has warehouse or not

6. The region where the house is placed

7. Price in Toman and USD

8. Price in Dollar (Every USD is equal to 30,000 Tomans)
```

We train data in two different ways

```
1. converting object columns to integer

2. using dummy for converting object columns to boolean
```

for first way we use Training 1 and for second way we use Training 2

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sb
from scipy.stats import pearsonr
from scipy.stats import spearmanr
from scipy.stats import chi2_contingency
from sklearn.model_selection import train_test_split, GridSearchCV, KFold
from sklearn.linear_model import Ridge, Lasso, LinearRegression, ElasticNet
import sklearn.metrics as metrics
import time
```

```
import sklearn.metrics as metrics
from sklearn.model_selection import cross_val_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, GradientBoost
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import cross_val_score
```

data preprocessing and visualization

In [2]:
```
house_price_df = pd.read_csv(r'C:\Users\B1674080\OneDrive - Teesside University\ML\I
```

In [3]:
```
house_price_df.head()
```

Out[3]:

	Area	Room	Parking	Warehouse	Elevator	Address	Price	Price(USD)
0	63	1	True	True	True	Shahran	1850000000	61666.67
1	60	1	True	True	True	Shahran	1850000000	61666.67
2	79	2	True	True	True	Pardis	550000000	18333.33
3	95	2	True	True	True	Shahrake Qods	902500000	30083.33
4	123	2	True	True	True	Shahrake Gharb	7000000000	233333.33

In [4]:
```
house_price_df.describe()
```

Out[4]:

	Room	Price	Price(USD)
count	3479.000000	3.479000e+03	3.479000e+03
mean	2.079908	5.359023e+09	1.786341e+05
std	0.758275	8.099935e+09	2.699978e+05
min	0.000000	3.600000e+06	1.200000e+02
25%	2.000000	1.418250e+09	4.727500e+04
50%	2.000000	2.900000e+09	9.666667e+04
75%	2.000000	6.000000e+09	2.000000e+05
max	5.000000	9.240000e+10	3.080000e+06

In [5]:
```
house_price_df.shape
```

Out[5]: (3479, 8)

as we need one Y Lable and having 1 Price is enough, we delete Toman Price

In [6]:
```
house_price_df.drop('Price', axis = 1, inplace = True)
```

we rename the Price(USD) to Price

In [7]:
```
house_price_df = house_price_df.rename(columns={'Price(USD)' : 'Price'})
```

In [8]:
```
house_price_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3479 entries, 0 to 3478
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype  
---  ------     --------------  -----  
 0   Area       3479 non-null   object 
 1   Room       3479 non-null   int64  
 2   Parking    3479 non-null   bool   
 3   Warehouse  3479 non-null   bool   
 4   Elevator   3479 non-null   bool   
 5   Address    3456 non-null   object 
 6   Price      3479 non-null   float64
dtypes: bool(3), float64(1), int64(1), object(2)
memory usage: 119.0+ KB
```

for finding the null values :

In [9]:
```
house_price_df.isnull().sum()
```

Out[9]:
```
Area          0
Room          0
Parking       0
Warehouse     0
Elevator      0
Address      23
Price         0
dtype: int64
```

In [10]:
```
Address_Nul_Number = house_price_df.Address.isnull().sum()
print('There are {} null value in Address'.format(Address_Nul_Number))
```

```
There are 23 null value in Address
```

as we have 23 null value and only address column has null valiue, we use this code to eleminate those 23 null valiue

In [11]:
```
house_price_df.dropna(axis=0 , how='any',inplace = True)
house_price_df.reset_index(drop=True, inplace=True)
```

here we show some information about prices and rooms as they are the only integer data

In [12]:
```
house_price_df.describe()
```

Out[12]:

	Room	Price
count	3456.000000	3.456000e+03
mean	2.081308	1.793319e+05
std	0.759723	2.707243e+05
min	0.000000	1.200000e+02
25%	2.000000	4.733333e+04
50%	2.000000	9.666667e+04
75%	2.000000	2.000000e+05
max	5.000000	3.080000e+06

based on these information, we have 3 boolean columns and 2 object columns

we have to check are those information clear and chenge their type to int or use dommy for make them undrestandable to models

we need to find any data that is not int or have Error values

then we should change booleans filds to int for

First we should check is there any error in the data or not and as Address and Area are object, we should check them for Errors

In [13]:
```
Numeric_Count = 0
Index = 0
Area_errors=[]
for i in house_price_df.Area :
    if i.isnumeric():
        Numeric_Count += 1
    else:
        Area_errors.append([Index,i])
    Index += 1
```

as you can see errors in Area are as below:

In [14]:
```
Area_errors
```

Out[14]:
```
[[569, '3,310,000,000'],
 [706, '16,160,000,000'],
 [804, '1,000'],
 [1598, '8,400,000,000'],
 [2161, '3,600'],
 [2788, '2,550,000,000']]
```

In [15]:
```
print('Number of Numeric data in this columns : ',Numeric_Count)
```

```
Number of Numeric data in this columns :  3450
```

the only issues with type of data is using ',' , we have to delete all "," and convert them to int

In [16]:
```
for j in Area_errors:
    x = house_price_df.iloc[j[0]].Area
    print('Index Number : ', j[0])
    print('Old String Value     : ','"',x,'"')
    x = x.replace(',', '')
    house_price_df.at[j[0],'Area']=x
    print('New Integer Value    : ', house_price_df.at[j[0],'Area'])
    print('--------------------------')
```

```
Index Number :  569
Old String Value     :  " 3,310,000,000 "
New Integer Value    :  3310000000
--------------------------
Index Number :  706
Old String Value     :  " 16,160,000,000 "
New Integer Value    :  16160000000
--------------------------
Index Number :  804
Old String Value     :  " 1,000 "
New Integer Value    :  1000
--------------------------
```

```
Index Number :  1598
Old String Value     :  " 8,400,000,000 "
New Integer Value    :  8400000000
--------------------------
Index Number :  2161
Old String Value     :  " 3,600 "
New Integer Value    :  3600
--------------------------
Index Number :  2788
Old String Value     :  " 2,550,000,000 "
New Integer Value    :  2550000000
--------------------------
```

then we should delete unreal data we delete those houses which the price for theme are less than 1000 because in the real word in Tehran this prices are not real

In [17]:

```
count = 0
Index = 0
Area_errors=[]
for i in house_price_df.Price:
    if i <= 1000:
        count += 1
        Area_errors.append([Index,i])
    Index +=1
print('Count for less than 1000 : ', count)

count = 0
for i in house_price_df.Price:
    if i >= 1000:
        count += 1
print('Count for more than 1000 : ', count)
house_price_df.reset_index(drop = True, inplace = True)
```

```
Count for less than 1000 :  1
Count for more than 1000 :  3455
```

In [18]:

```
Outliers = []
for i in house_price_df['Price'].index:
    if house_price_df['Price'][i] <= 1000:
        Outliers.append(i)

house_price_df.drop(Outliers, inplace = True)
house_price_df.reset_index(drop = True, inplace = True)
```

in this project we train data in 2 different way:

```
1. by converting data to int

2. by using dummy
```

then we should check some rar data which are real but they are rare, because this data can affect our models while they might be a few.

so we have to check thes information and find that how many outlier we have

In [19]:

```
house_price_df["Area"] = house_price_df["Area"].astype(str).astype(np.int64)
house_price_df["Area"]
```

Out[19]:

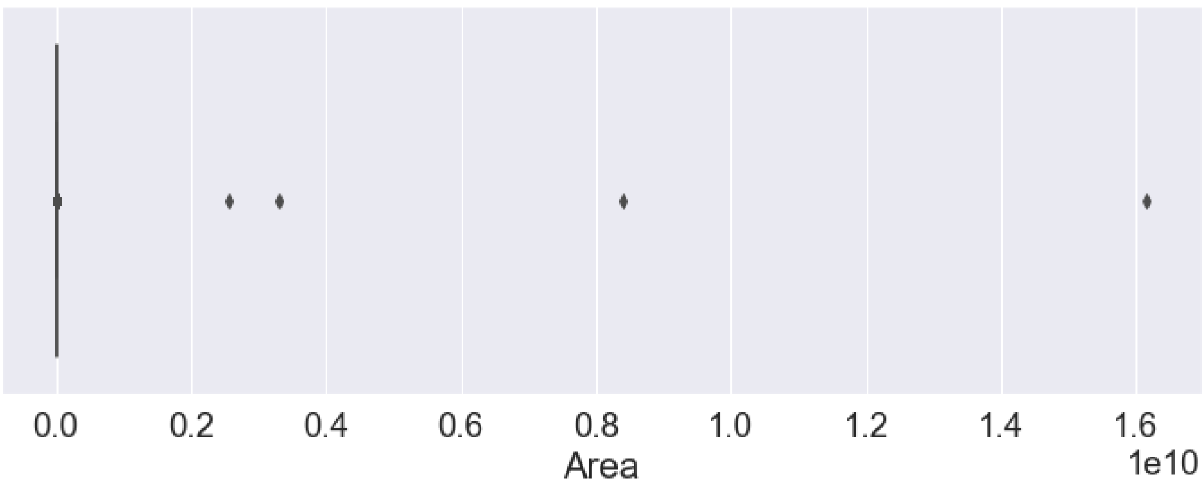
```
0        63
1        60
2        79
```

```
3        95
4       123
       ...
3450     86
3451     83
3452     75
3453    105
3454     82
Name: Area, Length: 3455, dtype: int64
```

In [20]:

```
plt.figure(figsize = (12,4))
sb.set_style("whitegrid")
sb.set(font_scale = 1.7)
sb.boxplot(x = house_price_df.Area)
```

Out[20]: <AxesSubplot:xlabel='Area'>



as you can see there are some outlier which affect our data badly and we have to fin and delete them

we decide to find lower and upper area based on IQR = Q3 - Q1 formulia

In [21]:

```
def lower_upper(x):
    Q1 = np.percentile(x, 25)
    Q3 = np.percentile(x, 75)
    IQR = Q3 - Q1
    lower = Q1 - 1.5 * IQR
    upper = Q3 + 1.5 * IQR
    return lower, upper

lower_area, upper_area = lower_upper(house_price_df['Area'])

print(f"Lower limit for area: {lower_area}")
print(f"Upper limit for area: {upper_area}")
```

```
Lower limit for area: -8.25
Upper limit for area: 197.75
```

as you can see we found a upper and lower limitation. But based on the knowledg related to Area in square meters in Tehran and houses in tehran, number of houses which are bigger than 250 in square meters are rare and they counted as mansion or villa. We can change the cofficient or filter them based on our knowledg .

Furthermore, we dont have any house smaller than 20 in square meters.

as we want to predict the houses in tehran, we eleminate these outliers baced on our knowledg insteed of lower_upper function.

so we should find rows that their Area is less then 20 and more than 250

In [22]:

```
count = 0
for i in house_price_df.Area:
    if i <= 20:
        count += 1
print('Count for less than 20 : ', count)


count = 0
for i in house_price_df.Area:
    if i >= 250:
        count += 1
print('Count for more than 250 : ', count)
house_price_df.reset_index(drop = True, inplace = True)
```

```
Count for less than 20 :  0
Count for more than 250 :  112
```

In [23]:

```
Outliers = []
for i in house_price_df['Area'].index:
    if house_price_df['Area'][i] > 250:
        Outliers.append(i)

house_price_df.drop(Outliers, inplace = True)
house_price_df.reset_index(drop = True, inplace = True)
```

In [24]:

```
count = 0
for j in house_price_df.Area:
    if j > 250:
        count += 1
print('Count for more than 250  : ', count)
```

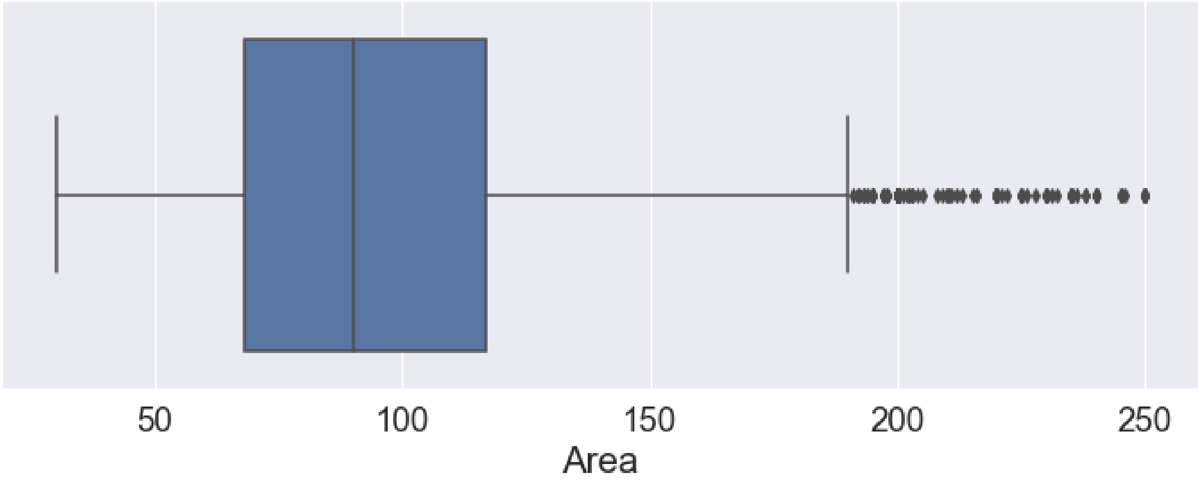
```
Count for more than 250  :  0
```

then we can see that howmuch those out liers, 112 rows, affected our data

In [25]:

```
plt.figure(figsize = (12,4))
sb.set_style("whitegrid")

sb.set(font_scale = 1.7)
sb.boxplot(x = house_price_df.Area)
```

Out[25]: <AxesSubplot:xlabel='Area'>



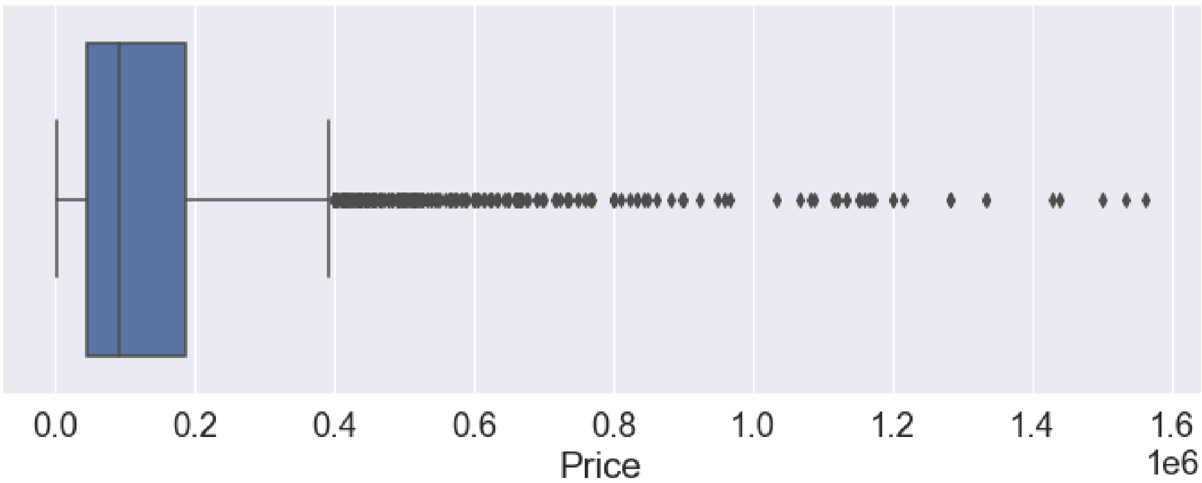
now you can see a better variance in this column.

Then we should find the outliers for Prices as well

Because might there are some rar houses which are too expensive and they might affect our models

In [26]:
```
plt.figure(figsize = (12,4))
sb.set_style("whitegrid")
sb.set(font_scale = 1.7)
sb.boxplot(x = house_price_df.Price)
```

Out[26]: <AxesSubplot:xlabel='Price'>



In [27]:
```
lower_Price, upper_Price = lower_upper(house_price_df['Price'])

print(f"Lower limit for area: {lower_Price}")
print(f"Upper limit for area: {upper_Price}")
```

```
Lower limit for area: -165000.005
Upper limit for area: 397666.67500000005
```

In [28]:
```
count = 0
for i in house_price_df.Price:
    if i <= lower_Price:
        count += 1
print('Count for less than Lower limit : ', count)
```

```
count = 0
for i in house_price_df.Price:
    if i >= upper_Price:
        count += 1
print('Count for more than Upper limit : ', count)
```

```
Count for less than Lower limit :  0
Count for more than Upper limit :  282
```

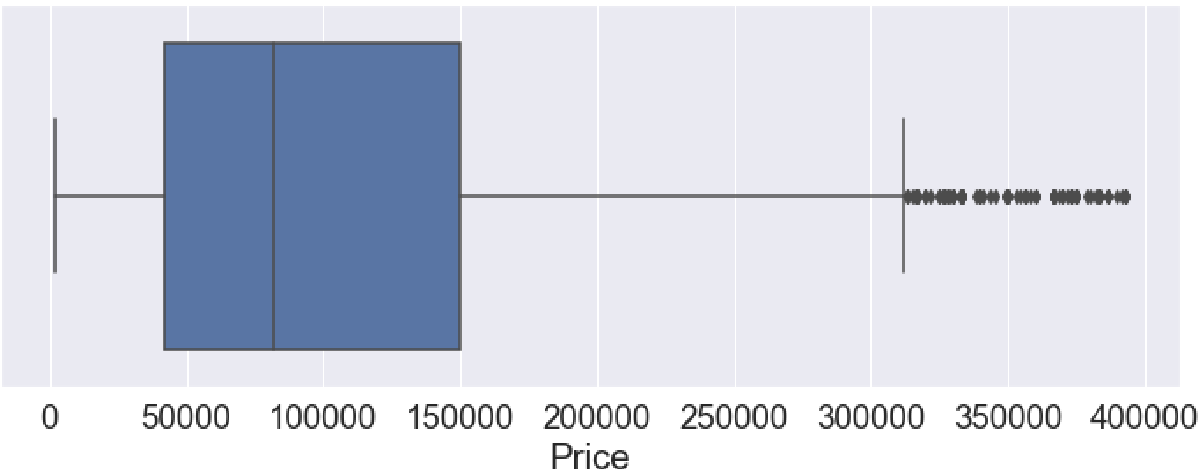
In [29]:

```
Outliers = []
for i in house_price_df['Price'].index:
    if house_price_df['Price'][i] > upper_Price:
        Outliers.append(i)


house_price_df.drop(Outliers, inplace = True)
house_price_df.reset_index(drop = True, inplace = True)
```

In [30]:

```
plt.figure(figsize = (12,4))
sb.set_style("whitegrid")
sb.set(font_scale = 1.7)
sb.boxplot(x = house_price_df.Price)
```

Out[30]: <AxesSubplot:xlabel='Price'>



now you can see a better variance in this column.

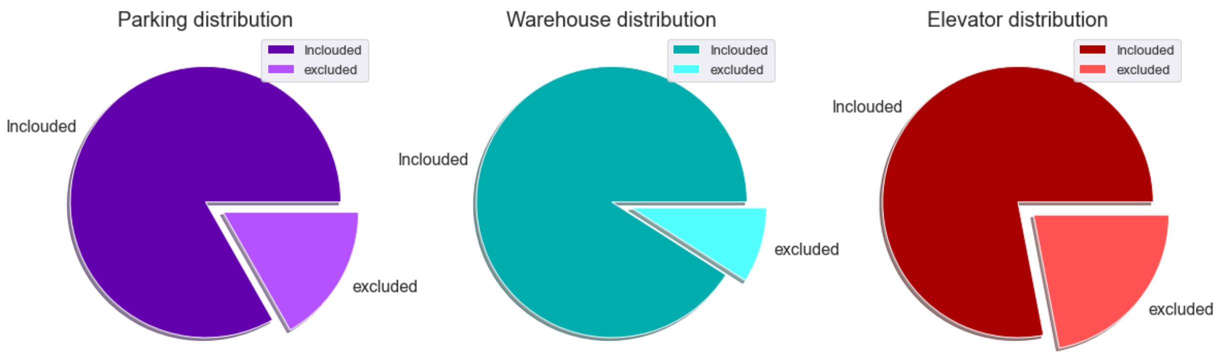
In [ ]:

then we should know how many of the houses have how many Parking, Warehouse, and Elevator.

here we show the variance for these 3 columns

In [31]:

```
fig, ax = plt.subplots(ncols=3, figsize=(20,9))

colors = [['#6200ad', '#b452ff'], ['#00adad', '#52ffff'], ['#a80000', '#ff5252']]
explode = [0, 0.15]
columns = ['Parking', 'Warehouse', 'Elevator']
for i in range(3):
        data = house_price_df[columns[i]].value_counts()
        ax[i].pie(data, labels=['Inclouded','excluded'], textprops={'fontsize': 16},
        ax[i].legend(labels=['Inclouded','excluded'], fontsize=13)
        ax[i].set_title('{} distribution'.format(columns[i]), size = 20)
```



now we check the distribution for Room in Prices and Area

In [32]:

```
sb.lmplot( x="Price", y="Area", data=house_price_df, fit_reg=False, hue='Room', lege
plt.legend(loc='lower right')
plt.show()
```

distribution for parking, warehouse and elevator in comparison with Price and Area:

In [33]:

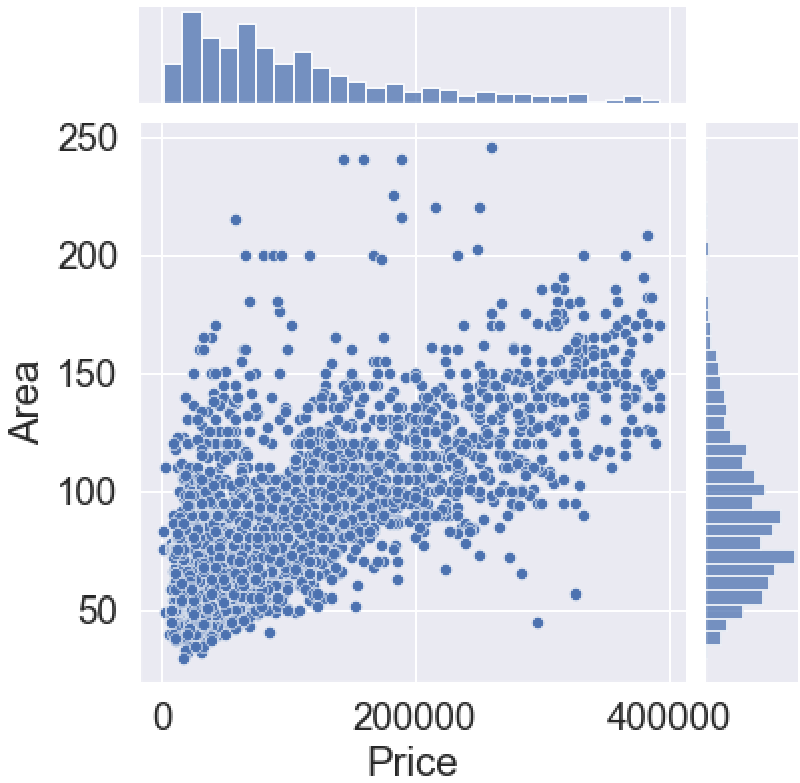
```
fig, ax = plt.subplots(ncols=3, figsize=(30,10))
sb.scatterplot(data=house_price_df,x='Price',y='Area', hue='Parking', ax=ax[0])
```

```
sb.scatterplot(data=house_price_df,x='Price',y='Area', hue='Warehouse',ax=ax[1])
sb.scatterplot(data=house_price_df,x='Price',y='Area', hue='Elevator',ax=ax[2])

ax[0].set_title('Parking distribution', size = 20)
ax[1].set_title('Warehouse distribution', size = 20)
ax[2].set_title('Elevator distribution', size = 20)
```

Out[33]: Text(0.5, 1.0, 'Elevator distribution')

destribution betwen Area and Price :

In [34]:
```
sb.jointplot(x='Price', y='Area', data = house_price_df, kind= 'scatter',  ax=ax[0])
plt.show()
```



Converting Address to Int

for showing information better and preparing data for Training 1

we convert all data to int here:

False values to 0 and True values to 1

In [35]:
```
house_price_int_addrss_df = house_price_df.copy()
```

In [36]:
```
#house_price_int_addrss_df.info()
```

Converting boolean value to int :

In [37]:
```
house_price_int_addrss_df.Parking.value_counts()
```

Out[37]:
```
True     2554
False     514
Name: Parking, dtype: int64
```

In [38]:
```
house_price_int_addrss_df.Parking.replace({'True': 1},inplace = True)
house_price_int_addrss_df.Parking.replace({'False': 0},inplace = True)
house_price_int_addrss_df = house_price_int_addrss_df.astype({'Parking':'int'})
house_price_int_addrss_df.Parking.value_counts()
```

Out[38]:
```
1    2554
0     514
Name: Parking, dtype: int64
```

In [39]:
```
house_price_int_addrss_df.Warehouse.value_counts()
```

Out[39]:
```
True     2791
False     277
Name: Warehouse, dtype: int64
```

In [40]:
```
house_price_int_addrss_df.Warehouse.replace({'True': 1},inplace = True)
house_price_int_addrss_df.Warehouse.replace({'False': 0},inplace = True)
house_price_int_addrss_df = house_price_int_addrss_df.astype({'Warehouse':'int'})
house_price_int_addrss_df.Warehouse.value_counts()
```

Out[40]:
```
1    2791
0     277
Name: Warehouse, dtype: int64
```

In [41]:
```
house_price_int_addrss_df.Elevator.value_counts()
```

Out[41]:
```
True     2394
False     674
Name: Elevator, dtype: int64
```

In [42]:
```
house_price_int_addrss_df.Elevator.replace({'True': 1},inplace = True)
house_price_int_addrss_df.Elevator.replace({'False': 0},inplace = True)
house_price_int_addrss_df = house_price_int_addrss_df.astype({'Elevator':'int'})
house_price_int_addrss_df.Elevator.value_counts()
```

Out[42]:
```
1    2394
0     674
Name: Elevator, dtype: int64
```

now we can convert addresses to int

In [43]:
```
house_price_int_addrss_df.Address.value_counts()
```

Out[43]:
```
Punak                    161
Pardis                   146
```

```
West Ferdows Boulevard    143
Shahran                   130
Parand                    127
                         ...
Pakdasht KhatunAbad         1
Chardivari                  1
Kazemabad                   1
Enghelab                    1
Shahrake Apadana            1
Name: Address, Length: 187, dtype: int64
```

In [44]:

```
Addresses = house_price_int_addrss_df.Address.value_counts()
Addresses
```

Out[44]:

```
Punak                     161
Pardis                    146
West Ferdows Boulevard    143
Shahran                   130
Parand                    127
                         ...
Pakdasht KhatunAbad         1
Chardivari                  1
Kazemabad                   1
Enghelab                    1
Shahrake Apadana            1
Name: Address, Length: 187, dtype: int64
```

In [45]:

```
Addresses_Name = []
Addresses_Code = []
for i in range(Addresses.count()):
    Addresses_Name.append(Addresses.index[i])
    Addresses_Code.append(i+1)
```

In [46]:

```
my_dict = {'Code':Addresses_Code,'Address':Addresses_Name}
my_df = pd.DataFrame(my_dict)
my_df
```

Out[46]:

	Code	Address
0	1	Punak
1	2	Pardis
2	3	West Ferdows Boulevard
3	4	Shahran
4	5	Parand
...	...	...
182	183	Pakdasht KhatunAbad
183	184	Chardivari
184	185	Kazemabad
185	186	Enghelab
186	187	Shahrake Apadana

187 rows × 2 columns

In [47]:

```
for i in my_df.index:
    house_price_int_addrss_df.Address.replace({my_df.loc[i].Address: my_df.loc[i].Co
    #print(my_df.loc[i].Address , '  --  ' ,  my_df.loc[i].Code)
house_price_int_addrss_df
```

Out[47]:

	Area	Room	Parking	Warehouse	Elevator	Address	Price
0	63	1	1	1	1	4	61666.67
1	60	1	1	1	1	4	61666.67
2	79	2	1	1	1	2	18333.33
3	95	2	1	1	1	16	30083.33
4	123	2	1	1	1	24	233333.33
...	...	...	...	...	...	...	...
3063	86	2	1	1	1	8	116666.67
3064	83	2	1	1	1	37	226666.67
3065	75	2	0	0	0	5	12166.67
3066	105	2	1	1	1	81	186666.67
3067	82	2	0	1	1	5	12000.00

3068 rows × 7 columns

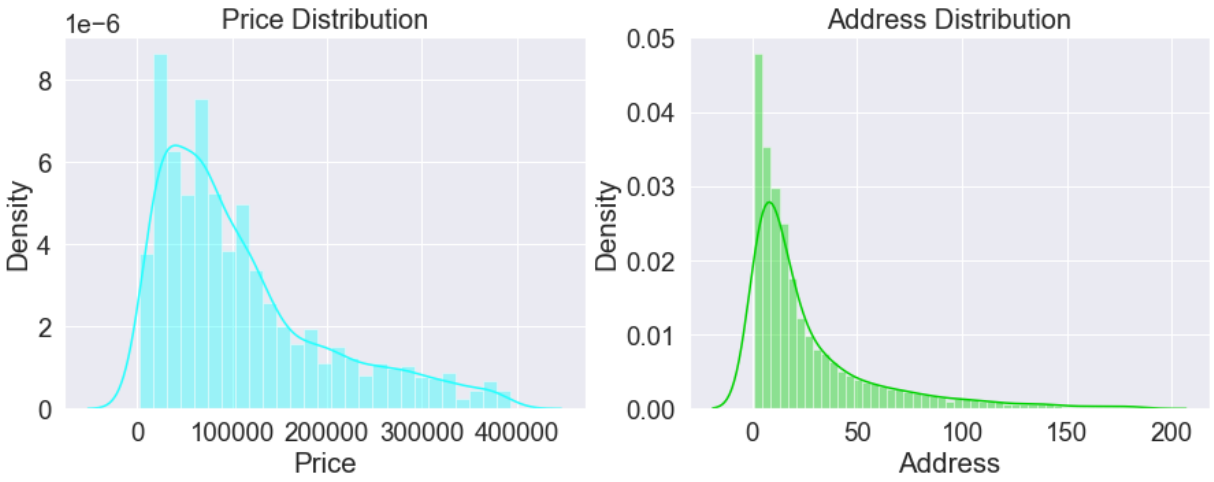
now all information in the data are integer

Density of Price and Address :

In [48]:

```
plt.subplots(ncols=2, figsize=(15,5))

plt.subplot(1,2,1)
sb.distplot(house_price_int_addrss_df.Price, color='#24ffff')
plt.title('Price Distribution', size = 20)


plt.subplot(1,2,2)
sb.distplot(house_price_int_addrss_df.Address, color='#00d100')
plt.title('Address Distribution', size = 20)

plt.show()
```

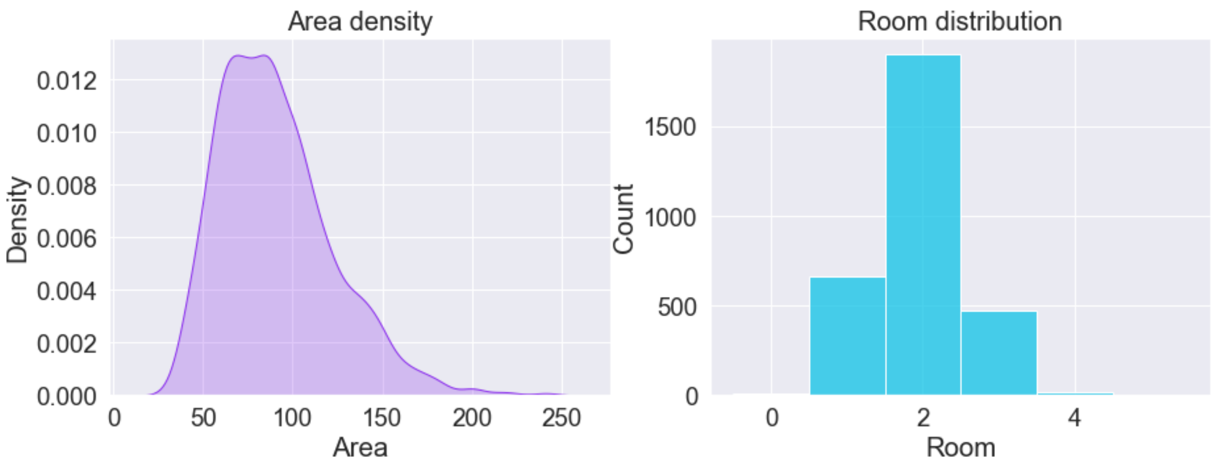
```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\distributions.py:2557: FutureWarn
ing: `distplot` is a deprecated function and will be removed in a future version. Pl
ease adapt your code to use either `displot` (a figure-level function with similar f
lexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



Density of Area and Room :

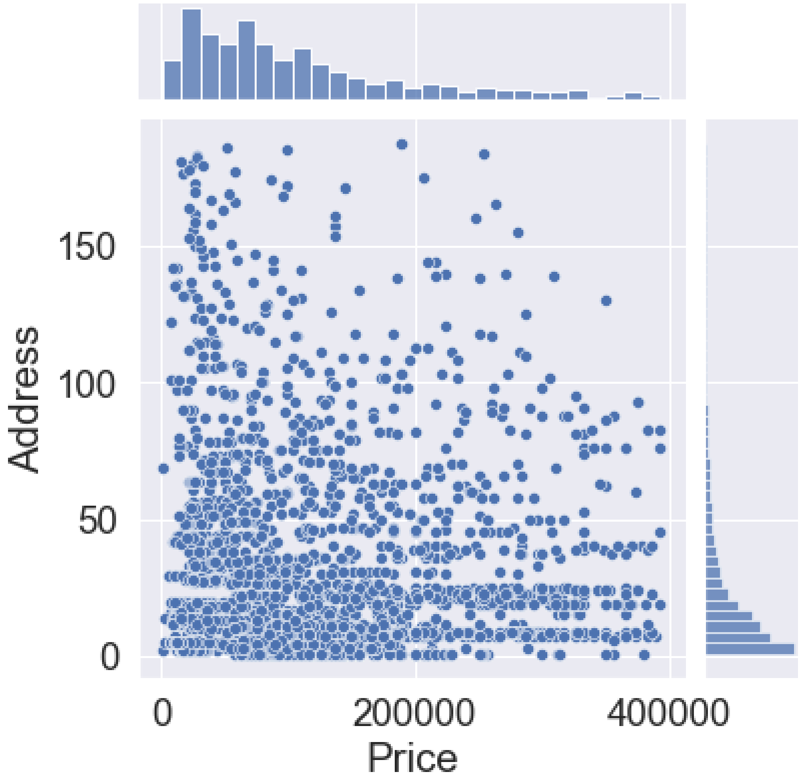
In [49]:
```
fig, ax = plt.subplots(ncols=2, figsize=(15,5))
sb.kdeplot(house_price_int_addrss_df['Area'], shade=True,  color='#892BED', ax=ax[0]
sb.histplot(data=house_price_int_addrss_df, x='Room', color='#0EC3E7', discrete=True

ax[0].set_title('Area density', size = 20)
ax[1].set_title('Room distribution', size = 20)
```

Out[49]:
```
Text(0.5, 1.0, 'Room distribution')
```



destribution betwen address and price

In [50]:
```
sb.jointplot(x='Price', y='Address', data = house_price_int_addrss_df, kind= 'scatte
plt.show()
```



correlation between all columns :

In [51]:
```
#pd.plotting.scatter_matrix(house_price_int_addrss_df, c=house_price_int_addrss_df.P
#plt.show()
```

Statistic Information

Covariance between Price and Address :

In [52]:
```
np.cov(house_price_int_addrss_df.Price, house_price_int_addrss_df.Address)
```

Out[52]:
```
array([[7.94932302e+09, 1.80113321e+05],
       [1.80113321e+05, 1.08769204e+03]])
```

Covariance between Price and Area :

In [53]:
```
np.cov(house_price_int_addrss_df.Price, house_price_int_addrss_df.Area)
```

Out[53]:
```
array([[7.94932302e+09, 1.87588883e+06],
       [1.87588883e+06, 1.01405838e+03]])
```

pearson_coefficent for Area and Price :

In [54]:
```
pearson_coefficent , _ = pearsonr(house_price_int_addrss_df.Area, house_price_int_ad
pearson_coefficent
```

Out[54]:
```
0.6607094909173736
```

as the result showed, there are %66 positive coefficent between Area and Price

In [55]:
```
pearson_coefficent , _ = pearsonr(house_price_int_addrss_df.Address, house_price_int
pearson_coefficent
```

Out[55]: 0.06125305955716133

but there is no coefficent between Address and Price

In [56]:
```
pearson_coefficent , _ = pearsonr(house_price_int_addrss_df.Room, house_price_int_ad
pearson_coefficent
```

Out[56]: 0.5027688313126805

and %50 positive coefficent between Room and Price

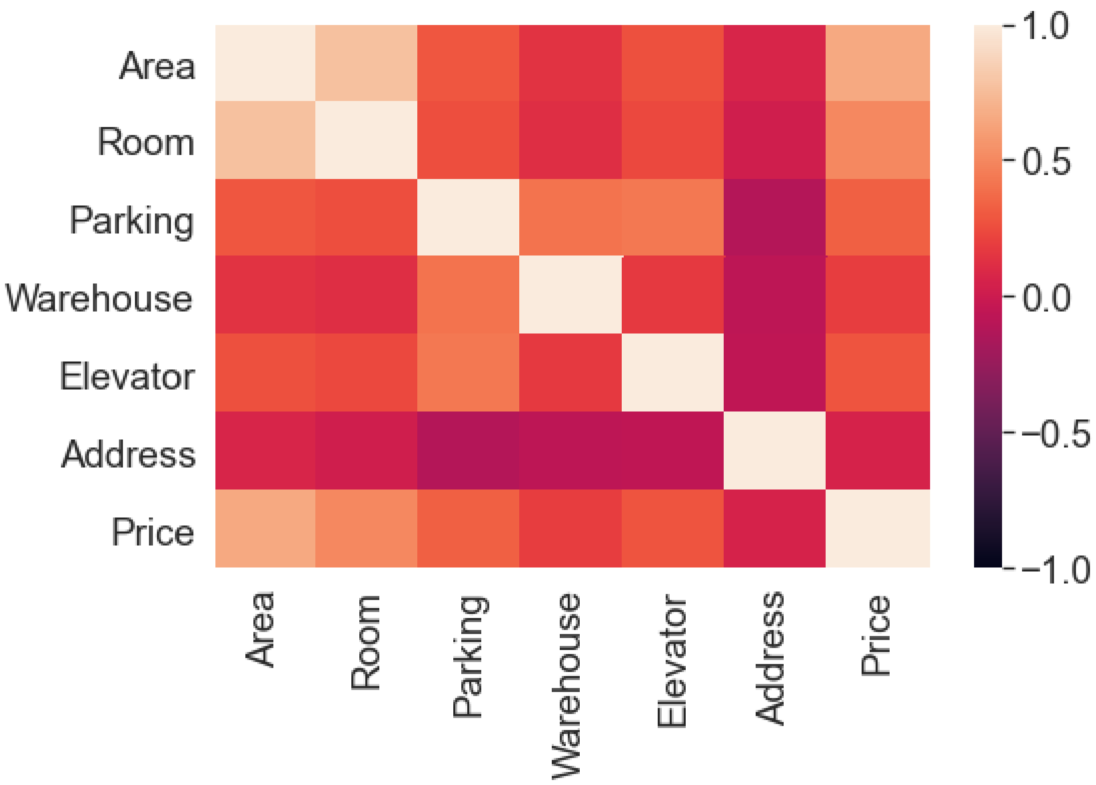
here you can see correlation between all columns

In [57]:
```
corr = house_price_int_addrss_df.corr()
corr
```

Out[57]:

	Area	Room	Parking	Warehouse	Elevator	Address	Price
Area	1.000000	0.776401	0.295796	0.143153	0.267082	0.071179	0.660709
Room	0.776401	1.000000	0.265140	0.117202	0.236295	0.023687	0.502769
Parking	0.295796	0.265140	1.000000	0.412933	0.432311	-0.109452	0.333021
Warehouse	0.143153	0.117202	0.412933	1.000000	0.178952	-0.063438	0.189137
Elevator	0.267082	0.236295	0.432311	0.178952	1.000000	-0.058506	0.288933
Address	0.071179	0.023687	-0.109452	-0.063438	-0.058506	1.000000	0.061253
Price	0.660709	0.502769	0.333021	0.189137	0.288933	0.061253	1.000000

In [58]:
```
plt.figure(figsize = (8,5))
sb.heatmap(corr, xticklabels=corr.columns, yticklabels = corr.columns, vmin = -1, vm
plt.show()
```



there are some relation between room and area, prices and area, prices and room.

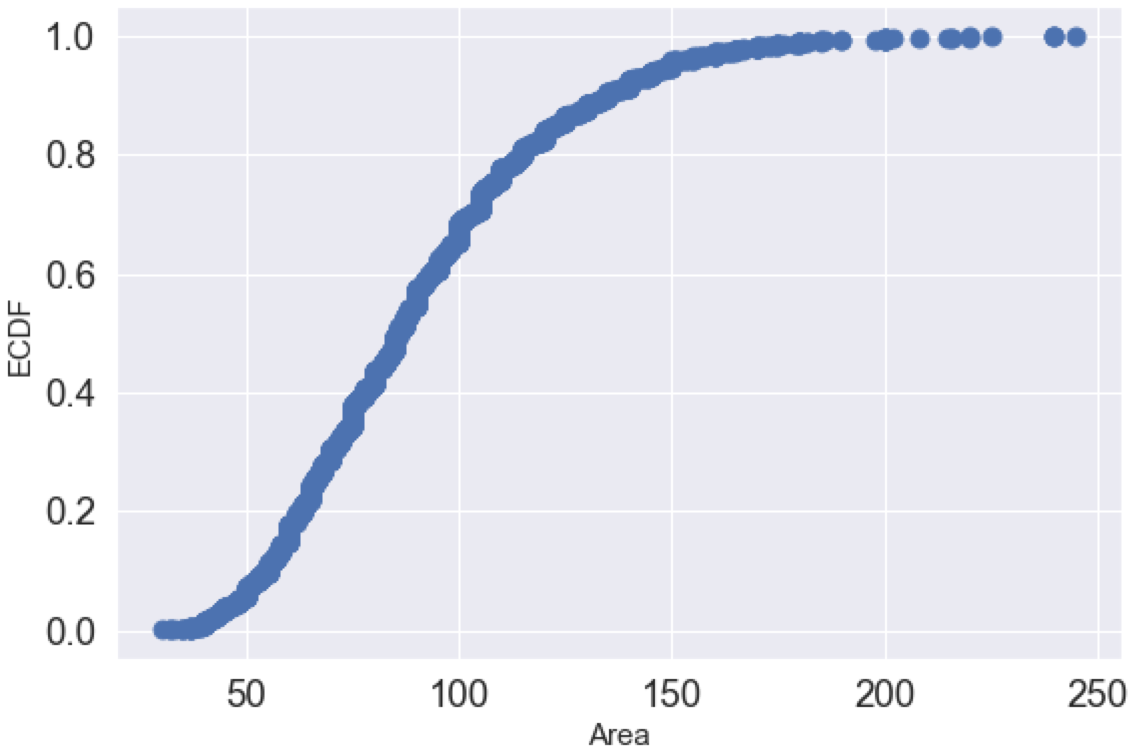
Empirical cumulative distribution function :

In [59]:

```
def ECDF(data):
    n = len(data)
    x =  np.sort(data)
    y = np.arange(1,n+1) / n
    return x,y
```

In [60]:

```
x, y = ECDF(house_price_int_addrss_df.Area)
plt.figure(figsize=(9,6))
plt.scatter(x,y, s=80)
plt.margins(0.05)
plt.xlabel('Area', fontsize = 15)
plt.ylabel('ECDF', fontsize = 15)
plt.show()
```



in this figure you can see ECDF (Empirical cumulative distribution function) for Area

In [61]:

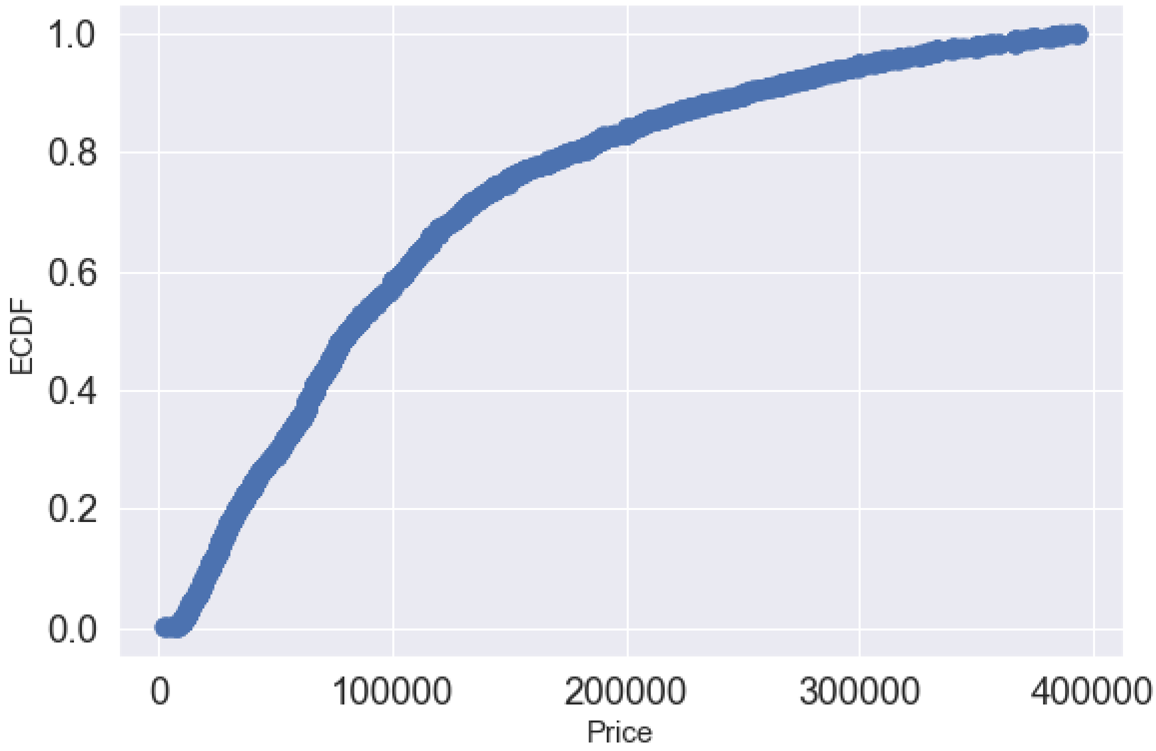
```
print('the Mean is {:.2f} and the Variance is {:.2f} and the Standard Deviation is {
```

```
the Mean is 90.46 and the Variance is 1013.73 and the Standard Deviation is 31.84
```

and this figure is for Price

In [62]:

```
x, y = ECDF(house_price_int_addrss_df.Price)
plt.figure(figsize=(9,6))
plt.scatter(x,y, s=80)
plt.margins(0.05)
plt.xlabel('Price', fontsize = 15)
plt.ylabel('ECDF', fontsize = 15)
plt.show()
```



Correlation between un parametric metods:

Spearman's rank correlation for Room and Area :

In [63]:
```
spearmanr_coefficent , _ = spearmanr(house_price_int_addrss_df.Room, house_price_int
spearmanr_coefficent
```

Out[63]: 0.78923253209026

as you can see there is a posetive correlation between Room and Area

Chi-square for Parking and Room :

In [64]:
```
table = pd.crosstab(house_price_int_addrss_df.Parking, house_price_int_addrss_df.Roo
table
```

Out[64]:

Room	0	1	2	3	4	5
Parking						
0	7	241	241	21	1	3
1	2	424	1657	453	16	2

In [65]:
```
Chi2, p_value, degreeoffreedom, expected = chi2_contingency(table.values)
```

In [66]:
```
print('Chi2             : ', Chi2)
print('p_value          : ', p_value)
print('degreeoffreedom : ', degreeoffreedom)
```

```
Chi2            :  287.24399018818104
p_value         :  5.52691954182817e-60
degreeoffreedom :  5
```

In [67]:
```
expected     # expected table for each situation
```

Out[67]:
```
array([[1.50782269e+00, 1.11411343e+02, 3.17983051e+02, 7.94119948e+01,
        2.84810952e+00, 8.37679270e-01],
       [7.49217731e+00, 5.53588657e+02, 1.58001695e+03, 3.94588005e+02,
        1.41518905e+01, 4.16232073e+00]])
```

degree of freedom is 5 and based on the Freedom Table the 0.05 for DOF 5 is 11.07

H0 has been rejected as 11.07 < 287.24

so we can find that there is no relatin between Parking and Room

Different View:

For having a better view abut data, we grouped data based on their address and put the mean for other columns

In [68]:
```
Address_Group_df = house_price_df.copy()
condition = Address_Group_df['Address'].value_counts() > 5
Address_Group_df = Address_Group_df[Address_Group_df['Address'].apply(lambda l: cond
Address_Group_df = Address_Group_df.groupby('Address').mean().sort_values(by='Area',
Address_Group_df.head()
```

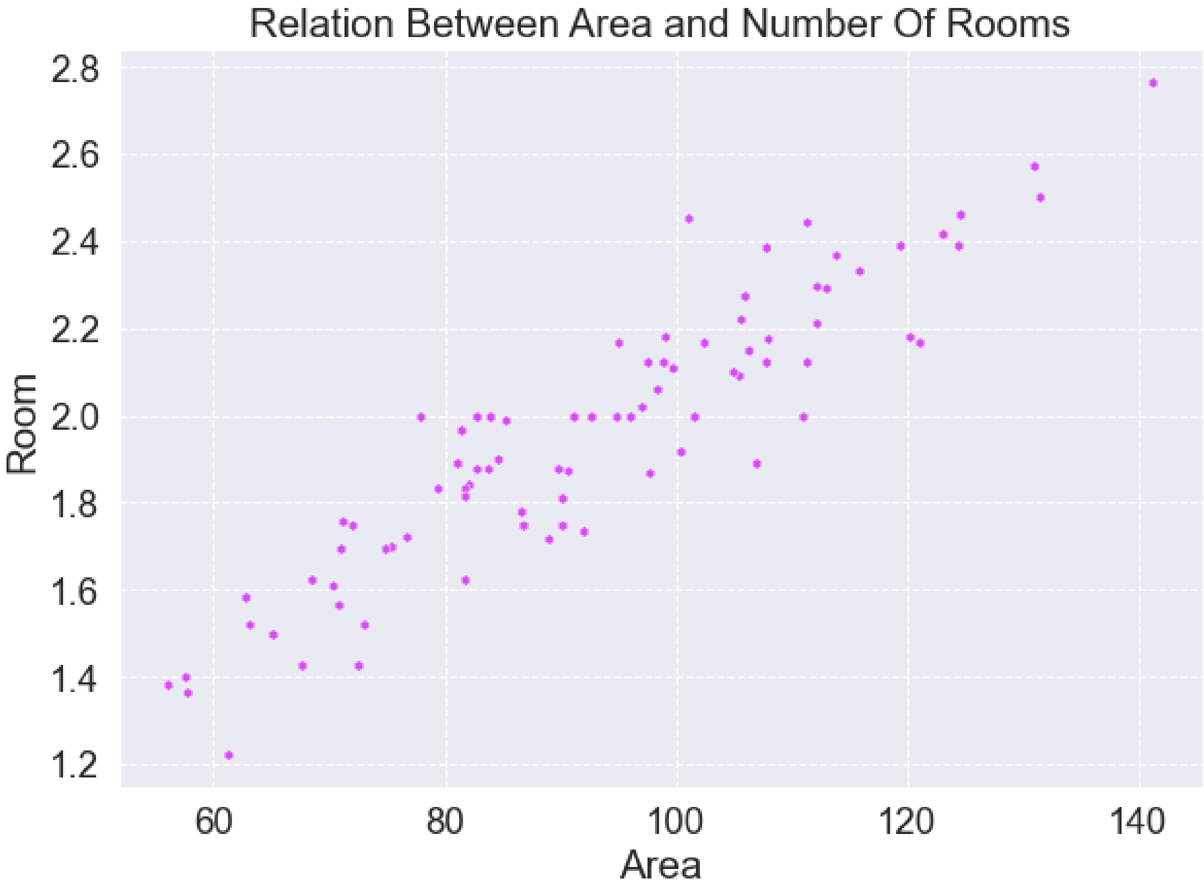
Out[68]:

Address	Area	Room	Parking	Warehouse	Elevator	Price
Marzdaran	141.235294	2.764706	0.941176	1.000000	0.941176	271068.627647
Sadeghieh	131.500000	2.500000	1.000000	1.000000	1.000000	158166.666667
Zaferanieh	131.000000	2.571429	1.000000	1.000000	1.000000	339000.000000
Heravi	124.615385	2.461538	0.974359	0.974359	0.974359	227176.068205
Pasdaran	124.470588	2.392157	1.000000	0.980392	0.960784	264062.745098

we chose 5 as the we have some rare info in address and chose those addressess which have been repeted in the dataset more than 5 time

In [69]:
```
fig, ax = plt.subplots(figsize=(10,7))
plt.grid(b=True, linestyle='dashed')
plt.title('Relation Between Area and Number Of Rooms')
sb.scatterplot(x=Address_Group_df['Area'], y=Address_Group_df['Room'], color='#D74CF
```

Out[69]:
```
<AxesSubplot:title={'center':'Relation Between Area and Number Of Rooms'}, xlabel='A
rea', ylabel='Room'>
```



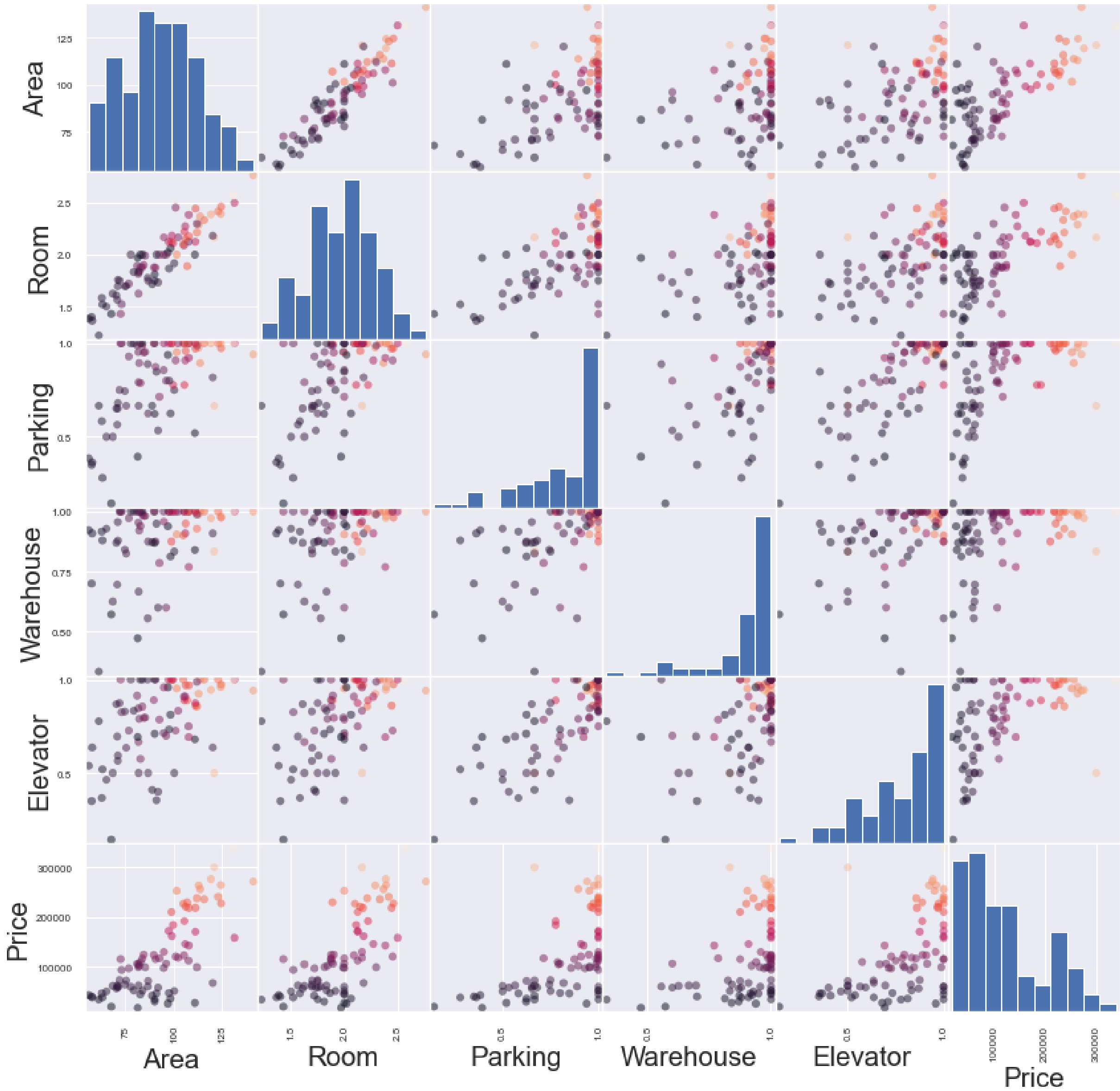
as you can see, there is a lineer regreasion connection between the nuber of rooms and the area

In [70]:
```
fig, ax = plt.subplots(figsize=(8, 7))
plt.grid(b=True, linestyle='dashed')
plt.title('Relation Between Area and Price')
sb.scatterplot(x=Address_Group_df['Area'], y=Address_Group_df['Price'], color='#2929
```

Out[70]:
```
<AxesSubplot:title={'center':'Relation Between Area and Price'}, xlabel='Area', ylab
el='Price'>
```

as you can see there is a linear relation between price and Area as well.

In [71]:
```
pd.plotting.scatter_matrix(Address_Group_df, c=Address_Group_df.Price,figsize=[14,14
plt.show()
```



as you can see there are a huge amount of linear relation between columns.

Now we preaper data and show the relation between the columns and find some statistic information which help us to find models for our data

in the first part, Training 1 we use columns which have been converted to int

Training 1

In [88]:

```
x = house_price_int_addrss_df.iloc[:,0:6]
y = house_price_int_addrss_df.iloc[:,-1]
```

we defined these variable for comparing the results at the end

In [89]:

```
my_columns = ['Model Name','Train R2','Test R2','EVS', 'MAE', 'MSE', 'RMSE', 'MedAE'
T1_Result_df = pd.DataFrame(columns=my_columns)
T1_Result_df
Temporary_T1_Result = []
```

This function has been wroted to help us to find the best parameters for our models

in this function, first we seperate data to test and train then train modle by GridSearchCV for finding the best parameters

and show the result for each models

it show :

```
The best parameters for model

Training Coefficient of determination (R2 score)

Test Coefficient of determination (R2 score)

Explain Variance Score

Mean Absolute Error

MSE (Mean Squared Error)

Square-Root of MSE

Median Absolute Error

Runtime of the program
```

In [90]:

```
def Trainer_by_Parameter(My_Model, My_Parameters, X, Y):

    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, rando
    Start_Time = time.time()

    Model = GridSearchCV(My_Model, param_grid = My_Parameters, refit = True, cv = KF

    Model_Fit = Model.fit(x_train, y_train)
    y_pred = Model_Fit.predict(x_test)

    Train_R2_score = round(Model_Fit.score(x_train, y_train), 4)
    Test_R2_score = round(Model_Fit.score(x_test, y_test), 4)
    EVS = round(metrics.explained_variance_score(y_test, y_pred), 2)
    MAB = round(metrics.mean_absolute_error(y_test, y_pred), 2)
    MSE = metrics.mean_squared_error(y_test, y_pred)
    RMSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
    MedAE = round(metrics.median_absolute_error(y_test, y_pred), 2)

    Model_name = str(My_Model).split('(')[0]

    Finish_Time = time.time()
    RunTime = Finish_Time - Start_Time
    print(f"The best parameters for {Model_name} model is: \n {Model_Fit.best_params
    print("----------------------------------------------------------------")
    print("Training Info : \n")
    print(f"  Coefficient of determination (R2 score)  :  {Train_R2_score:0.2%}.")
    print("\n----------------------------------------------------------------")
    print("Testing Info : \n")
    print(f"  Coefficient of determination (R2 score)  :  {Test_R2_score:0.2%}.")
    print("----------------------------------------------------------------")
    print("  Explain Variance Score                    : ", EVS)
    print("----------------------------------------------------------------")
    print("  Mean Absolute Error                       : ", MAB)
    print("----------------------------------------------------------------")
    print("  MSE (Mean Squared Error)                  : ", MSE)
    print("----------------------------------------------------------------")
```

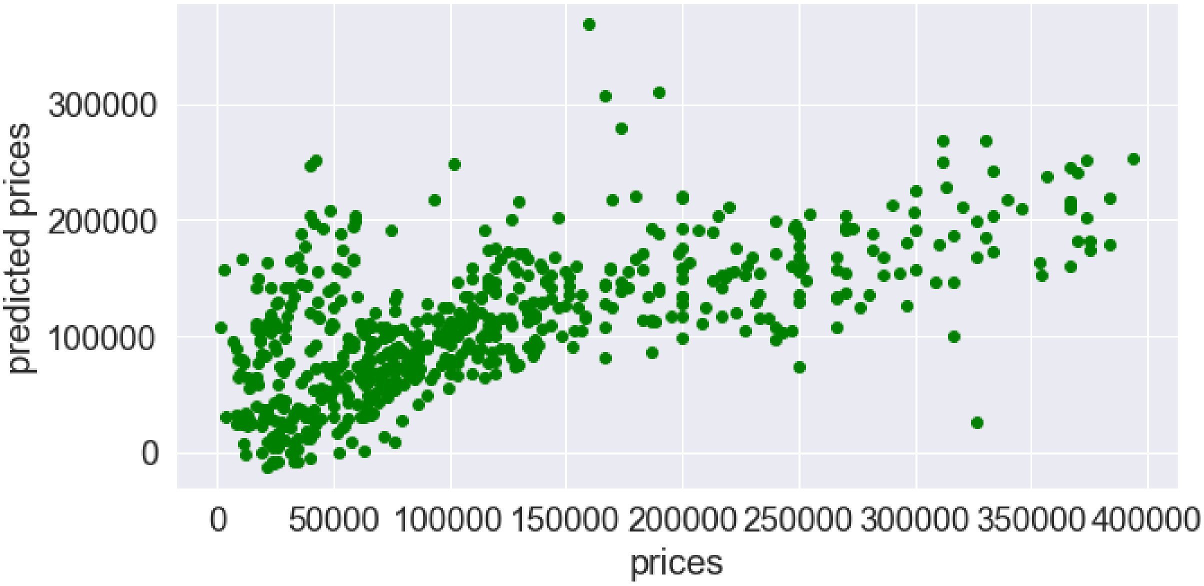
```
    print("  Square-Root of MSE                              : ", RMSE)
    print("------------------------------------------------------------------")
    print("  Median Absolute Error                           : ", MedAE)
    print("------------------------------------------------------------------")
    print(f" Runtime of the program                          :  {RunTime:0.2f}")
    print("------------------------------------------------------------------")


    my_dict = {'Model Name':Model_name,'Train R2': Train_R2_score,'Test R2': Test_R2
    Temporary_T1_Result.append(my_dict)


    plt.figure(figsize=(10,5))
    plt.scatter(y_test, y_pred,color = 'green')
    plt.xlabel('prices', fontsize = 20)
    plt.ylabel('predicted prices', fontsize = 20)
    plt.show()


    return Model_name, Train_R2_score, Test_R2_score, RMSE, MSE, RunTime
```

linear Regression

In [91]:

```
LR = LinearRegression(n_jobs = -1)
Model_name, LR_Train_R2_score, LR_Test_R2_score, LR_Sqr_MSE, LR_MSE, LR_RunTime  = T
```

```
The best parameters for LinearRegression model is:
 {}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  47.90%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  41.08%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.42
----------------------------------------------------------------
  Mean Absolute Error                      :  51159.99
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  5020989262.550838
----------------------------------------------------------------
  Square-Root of MSE                       :  70859
----------------------------------------------------------------
  Median Absolute Error                    :  32557.23
----------------------------------------------------------------
 Runtime of the program                   :  0.03
----------------------------------------------------------------
```



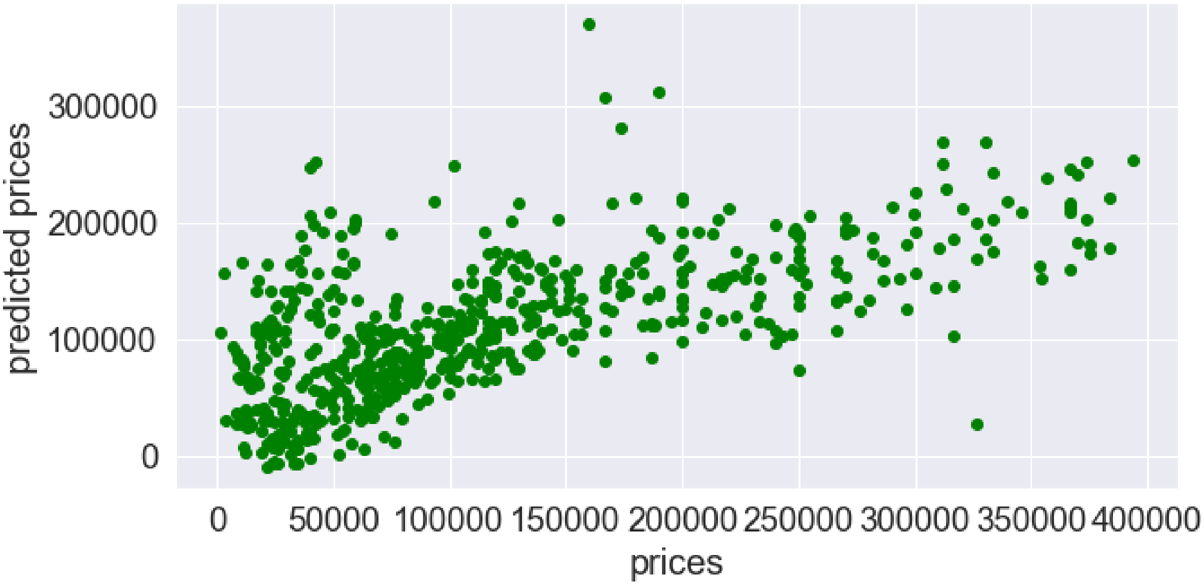
in the linear regression r2 score is %41 and it's not a good model for our data

Elastic Net Regression

In [92]:

```
ENR = ElasticNet(random_state = 1) # Linear regression with combined L1 and L2 prior
Param_ENR = {'alpha': [0.001, 0.01, 0.1, 1, 10],
            'l1_ratio': [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]}
Model_name, ENR_Train_R2_score, ENR_Test_R2_score, ENR_Sqr_MSE, ENR_MSE, ENR_RunTime
```

```
The best parameters for ElasticNet model is:
 {'alpha': 0.1, 'l1_ratio': 0.8}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  47.87%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  41.02%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.42
----------------------------------------------------------------
  Mean Absolute Error                      :  51168.35
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  5026237012.120527
----------------------------------------------------------------
  Square-Root of MSE                       :  70896
----------------------------------------------------------------
  Median Absolute Error                    :  32186.26
----------------------------------------------------------------
 Runtime of the program                   :  0.34
----------------------------------------------------------------
```



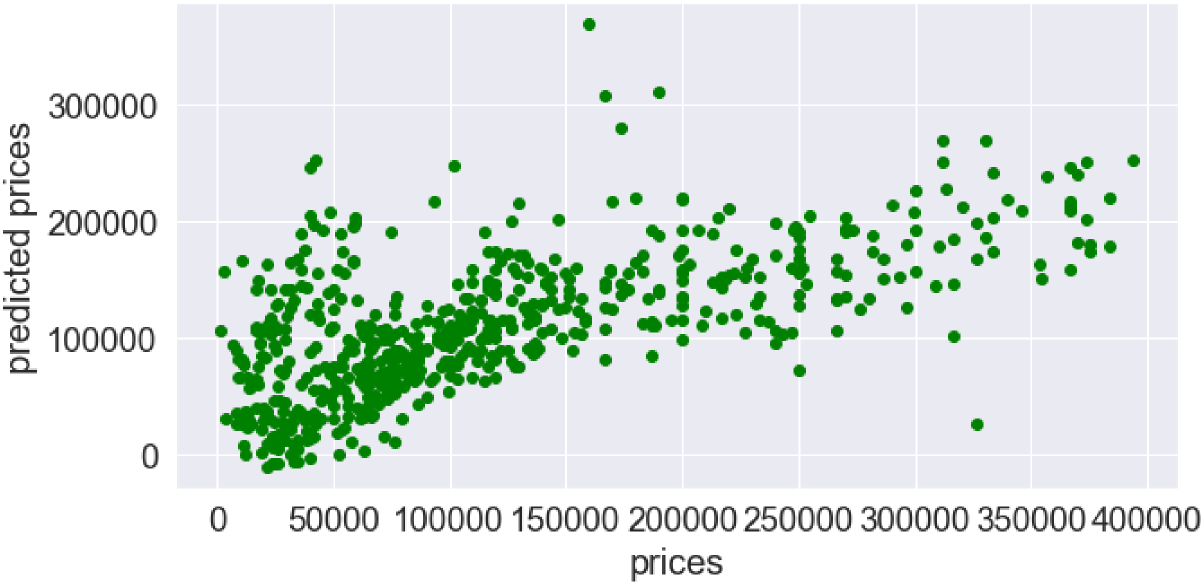
In [ ]:

Ridge Regression

In [93]:

```
RR = Ridge(random_state = 1)
param_ridge = {'alpha': [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1, 10, 20, 30, 40]}
Model_name, RR_Train_R2_score, RR_Test_R2_score, RR_Sqr_MSE, RR_MSE, RR_RunTime  = T
```

```
The best parameters for Ridge model is:
 {'alpha': 30}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  47.89%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  41.04%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.42
----------------------------------------------------------------
  Mean Absolute Error                      :  51164.12
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  5023973372.985397
----------------------------------------------------------------
  Square-Root of MSE                       :  70880
----------------------------------------------------------------
  Median Absolute Error                    :  32258.61
----------------------------------------------------------------
 Runtime of the program                   :  0.12
----------------------------------------------------------------
```



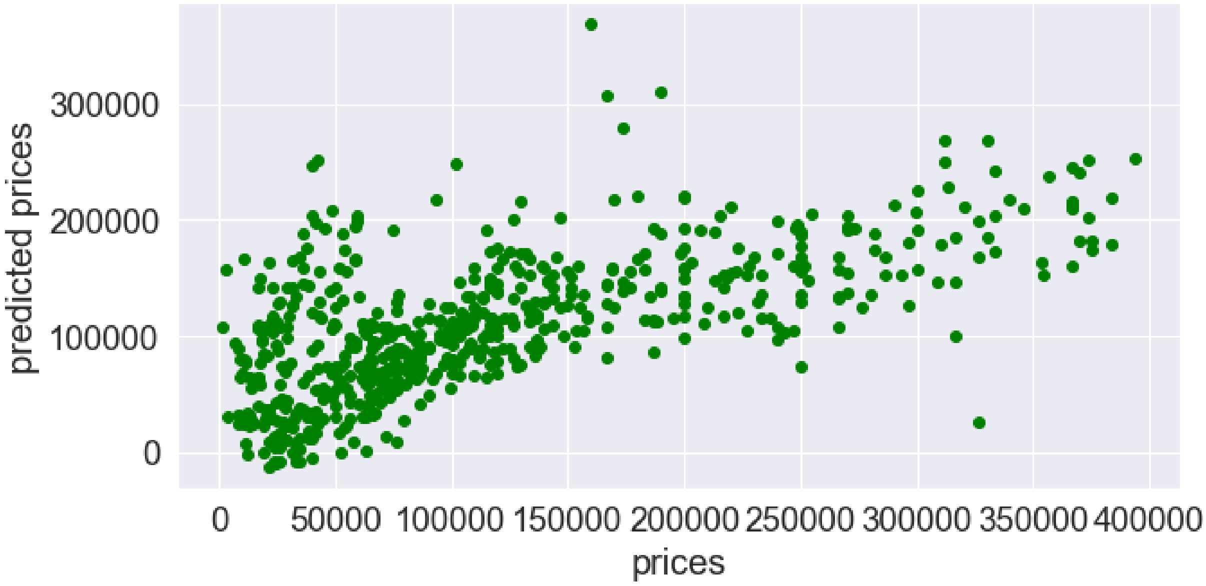
In [ ]:

Lasso Regression

In [94]:

```
LA = Lasso(random_state = 1)
param_LA = {'alpha': [0.001, 0.01, 0.1, 1, 10]}
Model_name, LA_Train_R2_score, LA_Test_R2_score, LA_Sqr_MSE, LA_MSE, LA_RunTime  = T
```

```
The best parameters for Lasso model is:
 {'alpha': 10}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  47.90%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  41.08%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.42
----------------------------------------------------------------
  Mean Absolute Error                      :  51160.52
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  5021189596.529721
----------------------------------------------------------------
  Square-Root of MSE                       :  70860
----------------------------------------------------------------
  Median Absolute Error                    :  32542.4
----------------------------------------------------------------
 Runtime of the program                  :  0.07
----------------------------------------------------------------
```



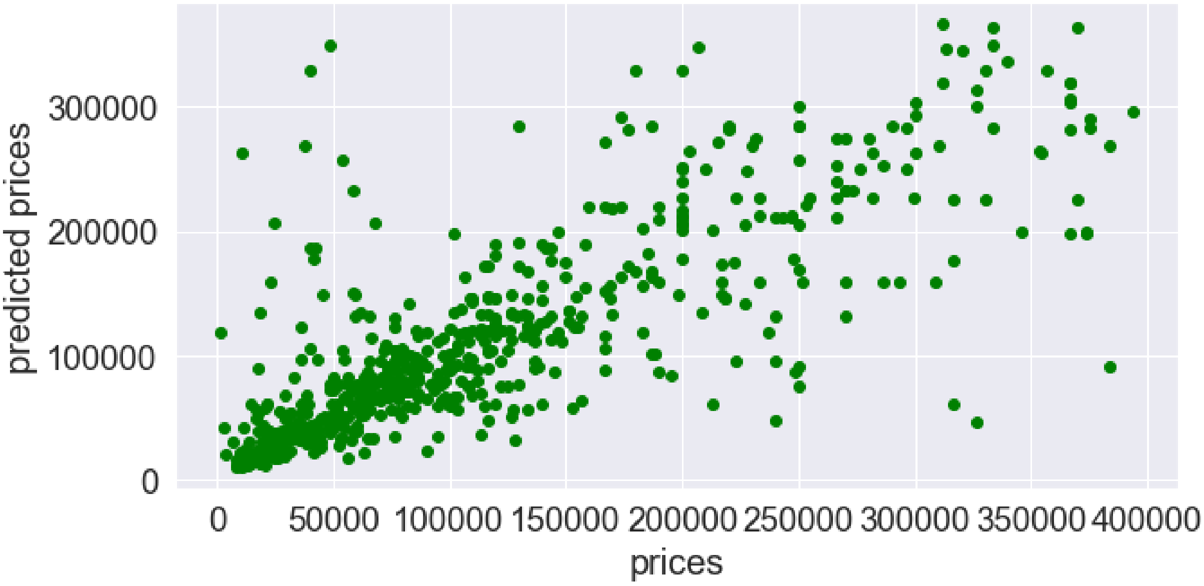
In [ ]:

Decision Tree Regression

In [95]:

```
DTR = DecisionTreeRegressor(random_state = 1)
Param_DTR = {'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
            'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7,8 ,9 , 10]}
Model_name, DTR_Train_R2_score, DTR_Test_R2_score, DTR_Sqr_MSE, DTR_MSE, DTR_RunTime
```

```
The best parameters for DecisionTreeRegressor model is:
 {'min_samples_leaf': 3, 'min_samples_split': 8}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  88.05%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  62.39%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.63
----------------------------------------------------------------
  Mean Absolute Error                      :  33792.29
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  3205245806.6909227
----------------------------------------------------------------
  Square-Root of MSE                       :  56615
----------------------------------------------------------------
  Median Absolute Error                    :  16738.1
----------------------------------------------------------------
 Runtime of the program                  :  1.00
----------------------------------------------------------------
```



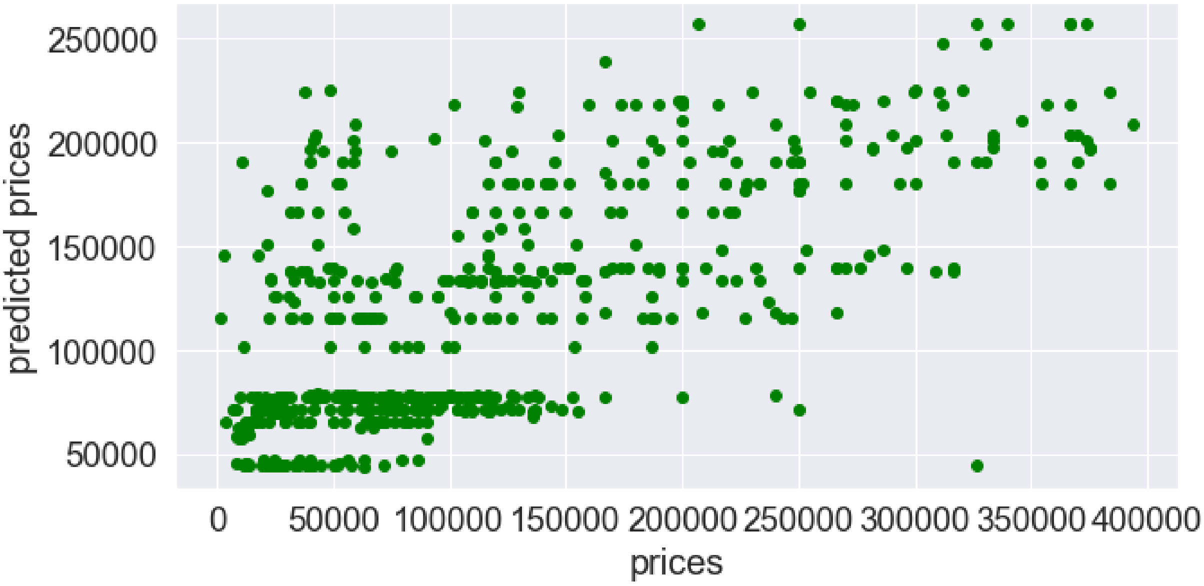
In [ ]:

Ada Boost Regression

In [96]:

```
ABR = AdaBoostRegressor()
Param_ABR = {'n_estimators': [10, 20, 30, 40, 50, 60, 70, 80, 90, 100],
             'learning_rate': [0.01, 0.1, 1, 2, 3, 4, 5, 10]}
Model_name, ABR_Train_R2_score, ABR_Test_R2_score, ABR_Sqr_MSE, ABR_MSE, ABR_RunTime
```

```
The best parameters for AdaBoostRegressor model is:
 {'learning_rate': 1, 'n_estimators': 20}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  55.34%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  46.58%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.47
----------------------------------------------------------------
  Mean Absolute Error                      :  50142.82
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  4552042779.420662
----------------------------------------------------------------
  Square-Root of MSE                       :  67469
----------------------------------------------------------------
  Median Absolute Error                    :  38233.34
----------------------------------------------------------------
 Runtime of the program                  :  10.04
----------------------------------------------------------------
```



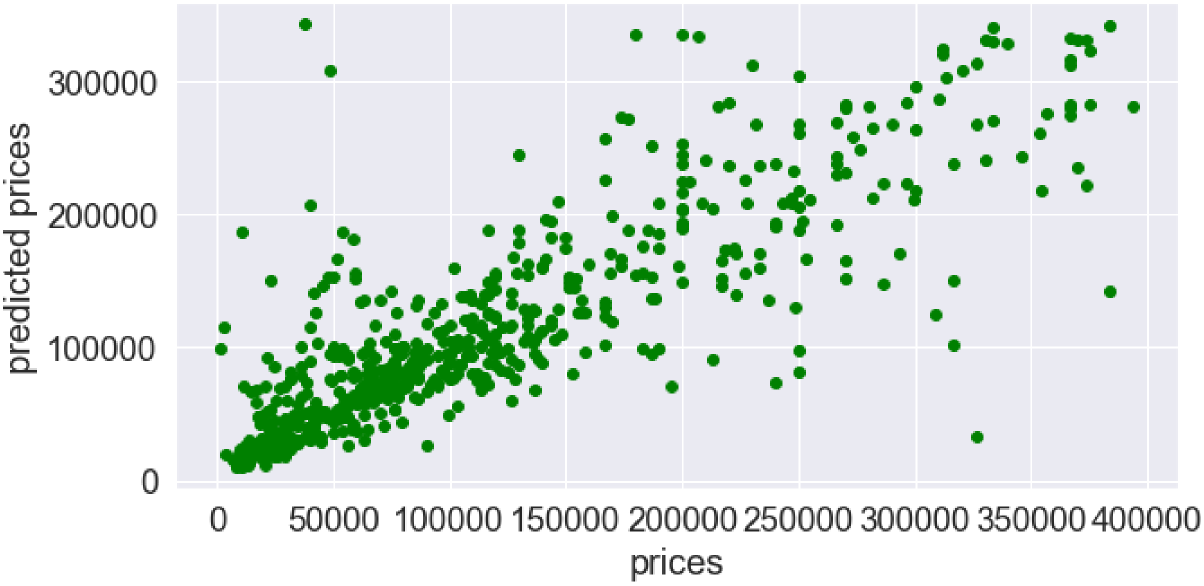
In [ ]:

Random Forest Regression

In [97]:

```
RFR = RandomForestRegressor(random_state = 1, n_jobs = -1)
Param_RFR = {'min_samples_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],
            'min_samples_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
Model_name, RFR_Train_R2_score, RFR_Test_R2_score, RFR_Sqr_MSE, RFR_MSE, RFR_RunTime
```

```
The best parameters for RandomForestRegressor model is:
 {'min_samples_leaf': 1, 'min_samples_split': 5}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  93.34%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  70.49%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.71
----------------------------------------------------------------
  Mean Absolute Error                      :  30828.78
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  2514640119.6843524
----------------------------------------------------------------
  Square-Root of MSE                       :  50146
----------------------------------------------------------------
  Median Absolute Error                    :  16409.97
----------------------------------------------------------------
 Runtime of the program                   :  40.19
----------------------------------------------------------------
```



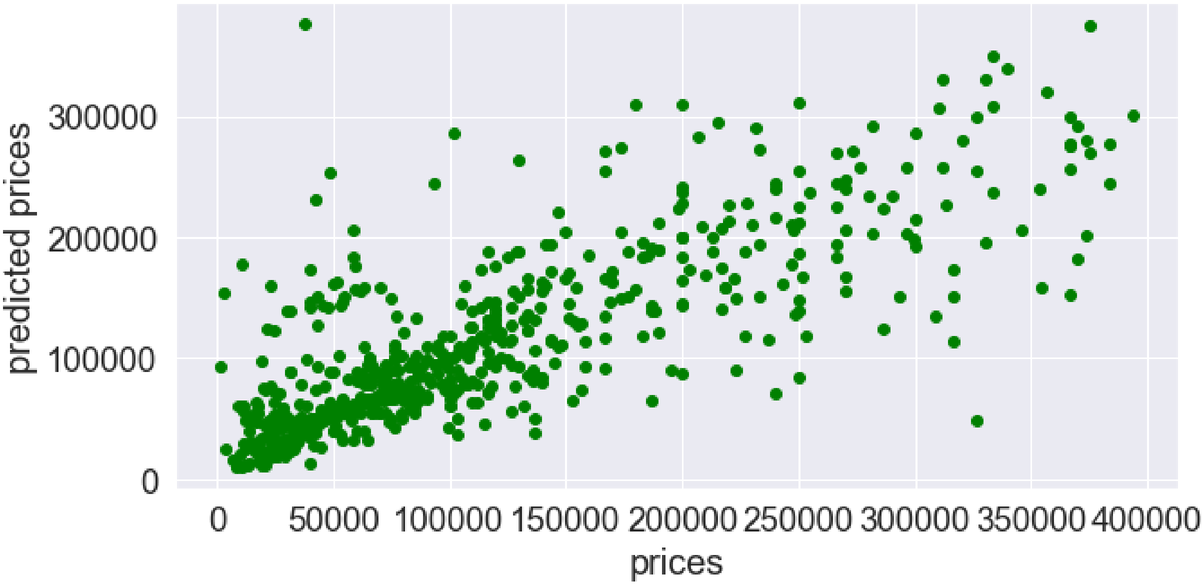
In [ ]:

K-Neighbors Regression

In [98]:

```
KNR = KNeighborsRegressor(n_jobs = -1)
Param_KNR = {'n_neighbors': [5, 10, 15, 20],
            'weights': ['uniform', 'distance']}
Model_name, KNR_Train_R2_score, KNR_Test_R2_score, KNR_Sqr_MSE, KNR_MSE, KNR_RunTime
```

```
The best parameters for KNeighborsRegressor model is:
 {'n_neighbors': 15, 'weights': 'distance'}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  98.75%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  62.51%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.63
----------------------------------------------------------------
  Mean Absolute Error                      :  35839.32
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  3194595242.3499613
----------------------------------------------------------------
  Square-Root of MSE                       :  56521
----------------------------------------------------------------
  Median Absolute Error                    :  20582.69
----------------------------------------------------------------
 Runtime of the program                   :  0.28
----------------------------------------------------------------
```



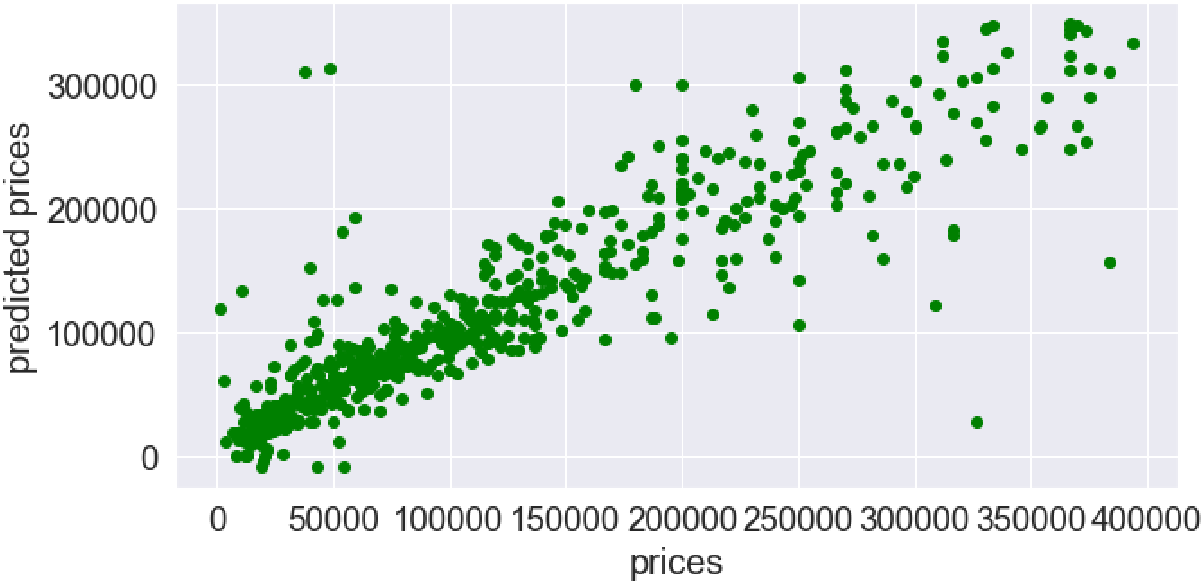
In [ ]:

Gradient Boosting Regression

In [99]:

```
GBR = GradientBoostingRegressor()
Param_GBR = {'learning_rate': [0.01, 0.1,0.2, 0.3, 0.4, 0.5],
            'alpha': [0.01, 0.1,0.2, 0.3, 0.4, 0.5],
            'max_depth': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]}
Model_name, GBR_Train_R2_score, GBR_Test_R2_score, GBR_Sqr_MSE, GBR_MSE, GBR_RunTime
```

```
The best parameters for GradientBoostingRegressor model is:
 {'alpha': 0.4, 'learning_rate': 0.5, 'max_depth': 3}
----------------------------------------------------------------
Training Info :

  Coefficient of determination (R2 score)  :  92.48%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  81.03%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.81
----------------------------------------------------------------
  Mean Absolute Error                      :  23676.82
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  1616751003.5427833
----------------------------------------------------------------
  Square-Root of MSE                       :  40209
----------------------------------------------------------------
  Median Absolute Error                    :  13279.15
----------------------------------------------------------------
 Runtime of the program                   :  90.89
----------------------------------------------------------------
```



after training data, we compare them to gather and the result are as below:

Comparing

In [100…

```
Temporary_T1_Result.sort
T1_Result_df.append(Temporary_T1_Result, ignore_index=True)
```

Out[100…

	Model Name	Train R2	Test R2	EVS	MAE	MSE	RMSE	MedAE
0	LinearRegression	0.4790	0.4108	0.42	51159.99	5.020989e+09	70859	32557.23
1	ElasticNet	0.4787	0.4102	0.42	51168.35	5.026237e+09	70896	32186.26
2	Ridge	0.4789	0.4104	0.42	51164.12	5.023973e+09	70880	32258.61
3	Lasso	0.4790	0.4108	0.42	51160.52	5.021190e+09	70860	32542.40
4	DecisionTreeRegressor	0.8805	0.6239	0.63	33792.29	3.205246e+09	56615	16738.10
5	AdaBoostRegressor	0.5534	0.4658	0.47	50142.82	4.552043e+09	67469	38233.34
6	RandomForestRegressor	0.9334	0.7049	0.71	30828.78	2.514640e+09	50146	16409.97
7	KNeighborsRegressor	0.9875	0.6251	0.63	35839.32	3.194595e+09	56521	20582.69
8	GradientBoostingRegressor	0.9248	0.8103	0.81	23676.82	1.616751e+09	40209	13279.15

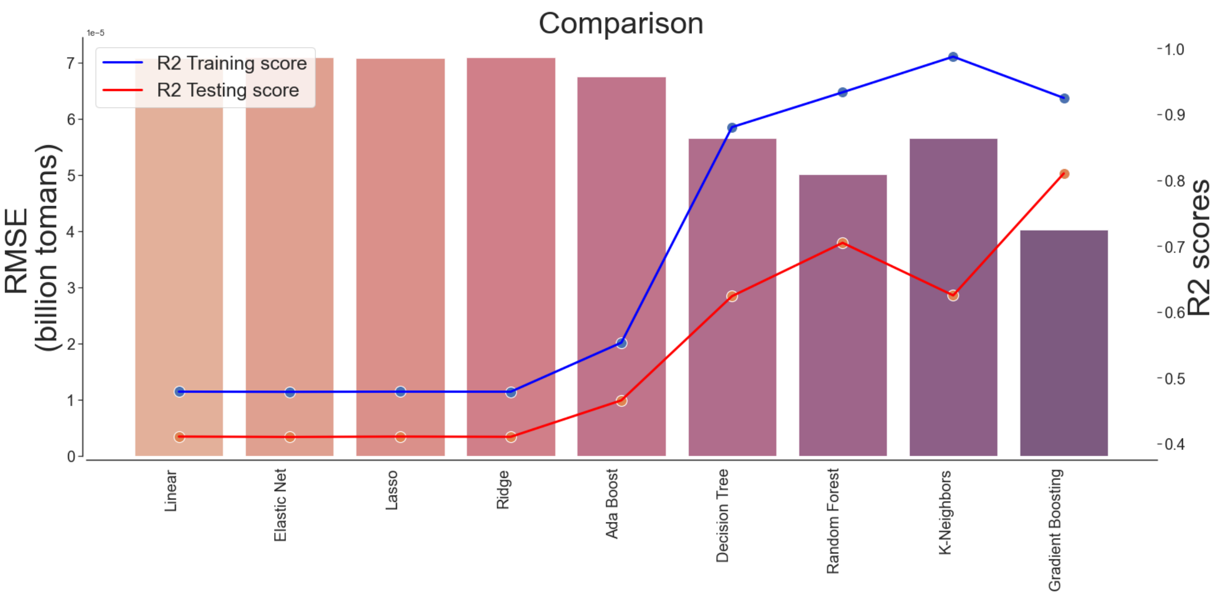
In [101…

```
models_score = pd.DataFrame({'Training r2 score': [LR_Train_R2_score, ENR_Train_R2_s
                             'Testing r2 score': [LR_Test_R2_score, ENR_Test_R2_scor
                             'RMSE': [LR_Sqr_MSE, ENR_Sqr_MSE, LA_Sqr_MSE, RR_Sqr_MS
                             index = ['Linear', 'Elastic Net', 'Lasso', 'Ridge', 'Ad
print(models_score)
```

```
                   Training r2 score  Testing r2 score   RMSE
Linear                        0.4790            0.4108  70859
Elastic Net                   0.4787            0.4102  70896
Lasso                         0.4790            0.4108  70860
Ridge                         0.4789            0.4104  70880
Ada Boost                     0.5534            0.4658  67469
Decision Tree                 0.8805            0.6239  56615
Random Forest                 0.9334            0.7049  50146
K-Neighbors                   0.9875            0.6251  56521
Gradient Boosting             0.9248            0.8103  40209
```

In [149…

```
fig, ax = plt.subplots(figsize=(25,10))
sb.set(style='white')
ax.set_title("Comparison", fontsize = 40)

ax = sb.barplot(x = list(models_score.index), y = models_score['RMSE']/1000000000, a
ax.set_ylabel("RMSE\n(billion tomans)", fontsize = 40)

sec_ax = ax.twinx()
sec_ax = sb.lineplot(x = list(models_score.index), y = models_score['Training r2 sco
sec_ax = sb.scatterplot(x = list(models_score.index), y = models_score['Training r2
sec_ax = sb.lineplot(x = list(models_score.index), y = models_score['Testing r2 scor
sec_ax = sb.scatterplot(x = list(models_score.index), y = models_score['Testing r2 s
sec_ax.set_ylabel("R2 scores", fontsize = 40)
sec_ax.legend(labels = ['R2 Training score', 'R2 Testing score'], fontsize = 25)
sb.despine(offset = 5)
for label in ax.get_xticklabels():
    label.set_rotation(90)
    label.set_size(20)
    label.set_ha('right')
for label in ax.get_yticklabels():
    label.set_size(20)
for label in sec_ax.get_yticklabels():
    label.set_size(20)
plt.show()
```



based on the result the best model for our data is Gradient Boosting

Training 2

now we convert all non integer data to boolean in seperated columns by dummy function and trained data

in the address column, we have 192 different addresses we have to convert these addresses to boolean we use get_dummies function

In [103…

```
Dummies_Address_House_Price_df = house_price_df.copy()
Dummies_Address_House_Price_df
```

Out[103…

	Area	Room	Parking	Warehouse	Elevator	Address	Price
0	63	1	True	True	True	Shahran	61666.67

	Area	Room	Parking	Warehouse	Elevator	Address	Price
1	60	1	True	True	True	Shahran	61666.67
2	79	2	True	True	True	Pardis	18333.33
3	95	2	True	True	True	Shahrake Qods	30083.33
4	123	2	True	True	True	Shahrake Gharb	233333.33
...	...	...	...	...	...	...	...
3063	86	2	True	True	True	Southern Janatabad	116666.67
3064	83	2	True	True	True	Niavaran	226666.67
3065	75	2	False	False	False	Parand	12166.67
3066	105	2	True	True	True	Dorous	186666.67
3067	82	2	False	True	True	Parand	12000.00

3068 rows × 7 columns

In [104…

```
dummy = pd.get_dummies(Dummies_Address_House_Price_df['Address'])
Dummies_Address_House_Price_df = house_price_df.merge(dummy, left_index = True, righ
Dummies_Address_House_Price_df.drop(columns = 'Address', inplace = True)
```

In [105…

```
dummy = pd.get_dummies(Dummies_Address_House_Price_df['Parking']).rename(columns=lam
Dummies_Address_House_Price_df = Dummies_Address_House_Price_df.merge(dummy, left_in
Dummies_Address_House_Price_df.drop(columns = 'Parking', inplace = True)
```

In [106…

```
dummy = pd.get_dummies(Dummies_Address_House_Price_df['Warehouse']).rename(columns=l
Dummies_Address_House_Price_df = Dummies_Address_House_Price_df.merge(dummy, left_in
Dummies_Address_House_Price_df.drop(columns = 'Warehouse', inplace = True)
```

In [107…

```
dummy = pd.get_dummies(Dummies_Address_House_Price_df['Elevator']).rename(columns=la
Dummies_Address_House_Price_df = Dummies_Address_House_Price_df.merge(dummy, left_in
Dummies_Address_House_Price_df.drop(columns = 'Elevator', inplace = True)
```

In [108…

```
Dummies_Address_House_Price_df.head(3)
```

Out[108…

	Area	Room	Price	Abazar	Abbasabad	Abuzar	Afsarieh	Ahang	Air force	Ajudaniye	...	Zafa
0	63	1	61666.67	0	0	0	0	0	0	0	...	
1	60	1	61666.67	0	0	0	0	0	0	0	...	
2	79	2	18333.33	0	0	0	0	0	0	0	...	

3 rows × 196 columns

We seperate X lables and Y lable

In [109…

```
x = Dummies_Address_House_Price_df.drop(columns = 'Price')
y = Dummies_Address_House_Price_df['Price']
```

these variables are for comparing the result at the end

In [110…

```
my_columns = ['Model Name','Train R2','Test R2','EVS', 'MAE', 'MSE', 'RMSE', 'MedAE'
T2_Result_df = pd.DataFrame(columns=my_columns)
T2_Result_df
Temporary_T2_Result = []
```

we wrote a function to finding the score for different modles and show the accuracy in the chart

In [111…

```
def Model_Trainer(My_Model, X, Y):

    x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = 0.2, rando
    Start_Time = time.time()

    My_Model.fit(x_train, y_train)
    y_pred = My_Model.predict(x_test)


    Train_R2_score = round(My_Model.score(x_train, y_train), 4)
    Test_R2_score = round(My_Model.score(x_test, y_test), 4)
    EVS = round(metrics.explained_variance_score(y_test, y_pred), 2)
    MAB = round(metrics.mean_absolute_error(y_test, y_pred), 2)
    MSE = metrics.mean_squared_error(y_test, y_pred)
    RMSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
    MedAE = round(metrics.median_absolute_error(y_test, y_pred), 2)

    Model_name = str(My_Model).split('(')[0]

    Finish_Time = time.time()
    RunTime = Finish_Time - Start_Time
    print("Training Info : \n")
    print(f"  Coefficient of determination (R2 score)  :  {Train_R2_score:0.2%}.")
    print("\n-----------------------------------------------------------------")
    print("Testing Info : \n")
    print(f"  Coefficient of determination (R2 score)  :  {Test_R2_score:0.2%}.")
    print("-----------------------------------------------------------------")
    print("  Explain Variance Score                    : ", EVS)
    print("-----------------------------------------------------------------")
    print("  Mean Absolute Error                       : ", MAB)
    print("-----------------------------------------------------------------")
    print("  MSE (Mean Squared Error)                  : ", MSE)
    print("-----------------------------------------------------------------")
    print("  Square-Root of MSE                        : ", RMSE)
    print("-----------------------------------------------------------------")
    print("  Median Absolute Error                     : ", MedAE)
    print("-----------------------------------------------------------------")
    print(f" Runtime of the program                    :  {RunTime:0.2f}")
    print("-----------------------------------------------------------------")


    my_dict = {'Model Name':Model_name,'Train R2': Train_R2_score,'Test R2': Test_R2
    Temporary_T2_Result.append(my_dict)

    plt.figure(figsize=(10,5))
    plt.scatter(y_test, y_pred,color = 'blue')
    plt.xlabel('prices', fontsize = 20)
    plt.ylabel('predicted prices', fontsize = 20)
    plt.show()

    return Train_R2_score, Test_R2_score, RMSE, MSE, RunTime
```

here we seperate data to test and train

In [112…

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_st
```

in the first part of each modle, we train modle by finding Cross Validation Score for each CV this is to show how model act without any parameters in the best CV

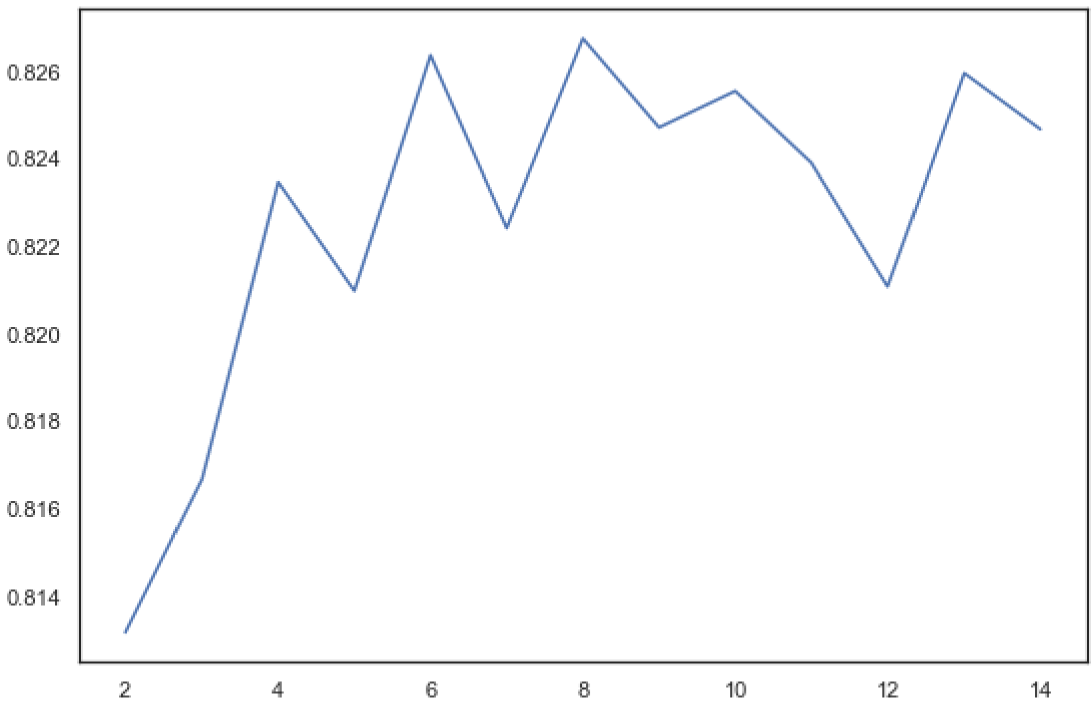
in the second part we train modle with different parameters and we find the best parameter

Ridge Regression

Cross Validation Score :

In [113…

```
RidgeR = Ridge()
_cv = []
_Cv_scores = []
for i in range(2,15):
    cv_scores = np.mean(cross_val_score(RidgeR, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)

plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```




```
Cross Validation score is 0.826760 with CV = 8.0.
```

Finding the best parameters :

In [114…

```
_alpha = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
_Sqr_MSE = []
```

```
for i in _alpha:
    RidgeR = Ridge(alpha = i).fit(x_train,y_train)
    y_pred = RidgeR.predict(x_test)
    Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
    _Sqr_MSE.append(Sqr_MSE)


my_dict = {'alpha':_alpha,'RMSE':_Sqr_MSE}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'],inplace = True)
my_df
```

Out[114…

	alpha	RMSE
6	0.7	31932
7	0.8	31936
5	0.6	31940
8	0.9	31950
4	0.5	31963
9	1.0	31973
3	0.4	32004
2	0.3	32069
1	0.2	32162
0	0.1	32295

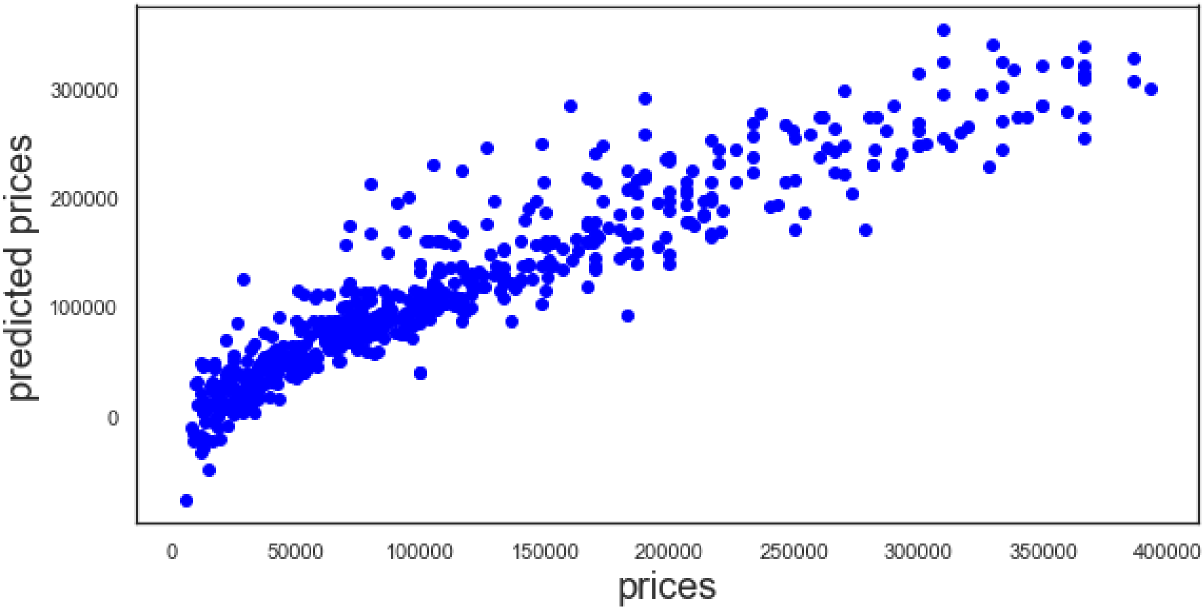
as you can see here, alpha 0.7 has the best score and 31932 MSE

train and show the result with the best parameters :

In [115…

```
RidgeR = Ridge(alpha = 0.7)
T2RR_Train_R2_score, T2RR_Test_R2_score, T2RR_Sqr_MSE, T2RR_MSE, T2RR_RunTime = Mode
```

```
Training Info :

  Coefficient of determination (R2 score)  :  86.10%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  87.09%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.87
----------------------------------------------------------------
  Mean Absolute Error                      :  22139.81
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  1019670487.1306739
----------------------------------------------------------------
  Square-Root of MSE                       :  31932
----------------------------------------------------------------
  Median Absolute Error                    :  14542.95
----------------------------------------------------------------
 Runtime of the program                   :  0.02
----------------------------------------------------------------
```



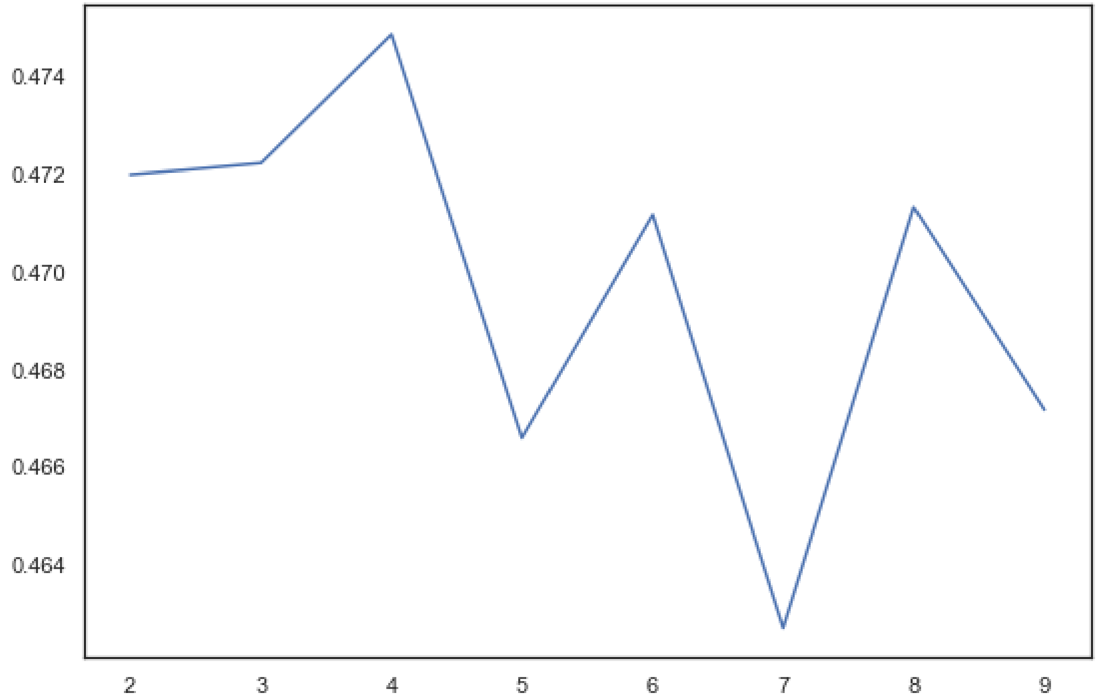
In [ ]:

Elastic Net

Cross Validation Score :

In [116…

```
ENR = ElasticNet()
_cv = []
_Cv_scores = []
for i in range(2,10):
    cv_scores = np.mean(cross_val_score(ENR, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)
plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```




```
Cross Validation score is 0.474876 with CV = 4.0.
```

Finding the best parameters :

In [117…

```
__alpha = [0.001, 0.01, 0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1]
__l1_ratio = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]
# The ElasticNet mixing parameter, with 0 <= l1_ratio <= 1. For l1_ratio = 0 the pen

_alpha = []
_l1_ratio = []
_Sqr_MSE = []


for i in __alpha:
    for j in __l1_ratio:
        ENR = ElasticNet(alpha = i, l1_ratio = j).fit(x_train,y_train)
        y_pred = ENR.predict(x_test)
        Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
        _alpha.append(i)
        _Sqr_MSE.append(Sqr_MSE)
        _l1_ratio.append(j)

my_dict = {'alpha':_alpha,'l1_ratio': _l1_ratio ,'RMSE':_Sqr_MSE}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 947762287932.3348, tolerance: 1952667167.76601
58
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 1330479048946.2502, tolerance: 1952667167.7660
158
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 1482692765084.1606, tolerance: 1952667167.7660
158
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
```

```
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 1520376122265.8013, tolerance: 1952667167.7660
158
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 1502961001513.9255, tolerance: 1952667167.7660
158
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 1456533348813.3867, tolerance: 1952667167.7660
158
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 1389911878986.4255, tolerance: 1952667167.7660
158
  model = cd_fast.enet_coordinate_descent(
```

Out[117…

	alpha	l1_ratio	RMSE
4	0.001	0.7	31932
5	0.001	0.8	31966
3	0.001	0.6	31968
2	0.001	0.5	32047
6	0.001	0.9	32116
...	...	...	...
71	0.900	0.4	62128
63	0.800	0.3	62206
78	1.000	0.4	62350
70	0.900	0.3	62448
77	1.000	0.3	62651

84 rows × 3 columns

az you can see the best parameter for alpha is 0.001 and for l1_ratio is 0.7

train and show the result with the best parameters :

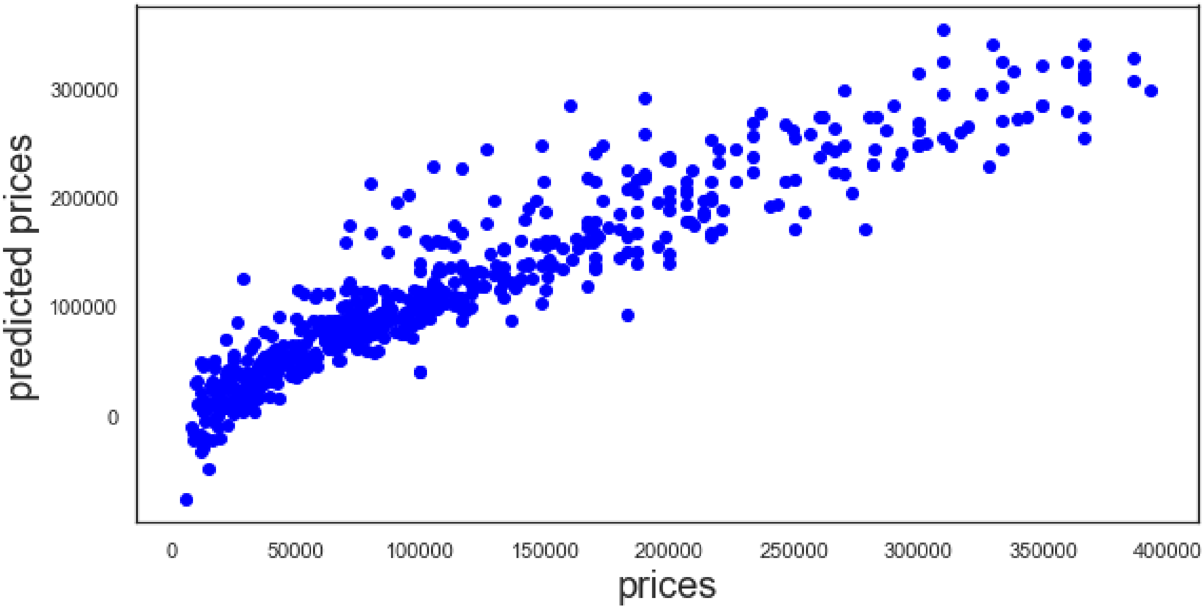
In [118…

```python
ENR = ElasticNet(alpha = 0.001, l1_ratio = 0.7)
T2ENR_Train_R2_score, T2ENR_Test_R2_score, T2ENR_Sqr_MSE, T2ENR_MSE, T2ENR_RunTime =
```

```
Training Info :

  Coefficient of determination (R2 score)  :  86.07%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  87.09%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.87
----------------------------------------------------------------
  Mean Absolute Error                      :  22145.26
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  1019683670.9782329
----------------------------------------------------------------
  Square-Root of MSE                       :  31932
```

```
-----------------------------------------------------------------
  Median Absolute Error                       :  14426.17
-----------------------------------------------------------------
 Runtime of the program                     :  0.36
-----------------------------------------------------------------
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 1502961001513.9255, tolerance: 1952667167.7660
158
  model = cd_fast.enet_coordinate_descent(
```



In [ ]:

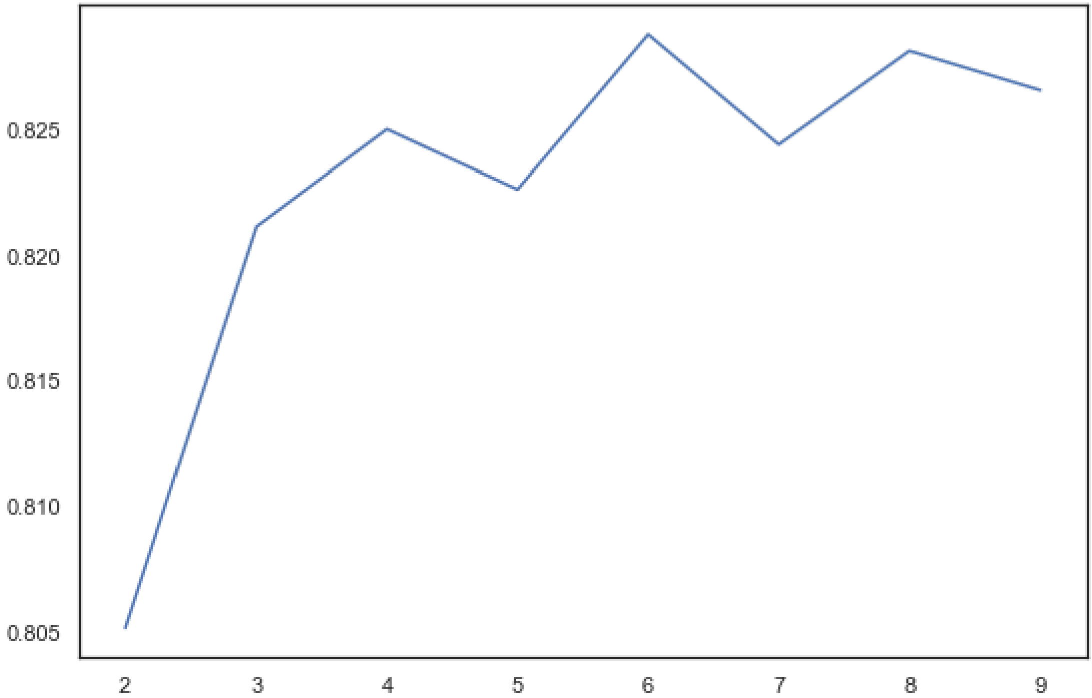
Lasso

Cross Validation Score :

In [119…

```python
Las = Lasso()
_cv = []
_Cv_scores = []
for i in range(2,10):
    cv_scores = np.mean(cross_val_score(Las, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)
plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 5578120521.798828, tolerance: 1779030885.52179
55
  model = cd_fast.enet_coordinate_descent(
```




```
Cross Validation score is 0.828802 with CV = 6.0.
```

Finding the best parameters :

In [120…

```
__alpha = [0.001, 0.01, 0.1 , 0.2 ,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,5,10]
_alpha = []
_Sqr_MSE = []

for i in __alpha:
    Las = Lasso(normalize = True, alpha = i).fit(x_train,y_train)
    y_pred = Las.predict(x_test)
    Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
    _alpha.append(i)
    _Sqr_MSE.append(Sqr_MSE)

my_dict = {'alpha':_alpha,'RMSE':_Sqr_MSE}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'])
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 66224557315.33765, tolerance: 1952667167.76601
58
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 59911319114.77759, tolerance: 1952667167.76601
58
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 4737601409.944336, tolerance: 1952667167.76601
58
  model = cd_fast.enet_coordinate_descent(
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\_coordinate_descent.
py:530: ConvergenceWarning: Objective did not converge. You might want to increase t
he number of iterations. Duality gap: 2666391994.26416, tolerance: 1952667167.766015
8
  model = cd_fast.enet_coordinate_descent(
```

Out[120…

	alpha	RMSE
12	5.000	32345

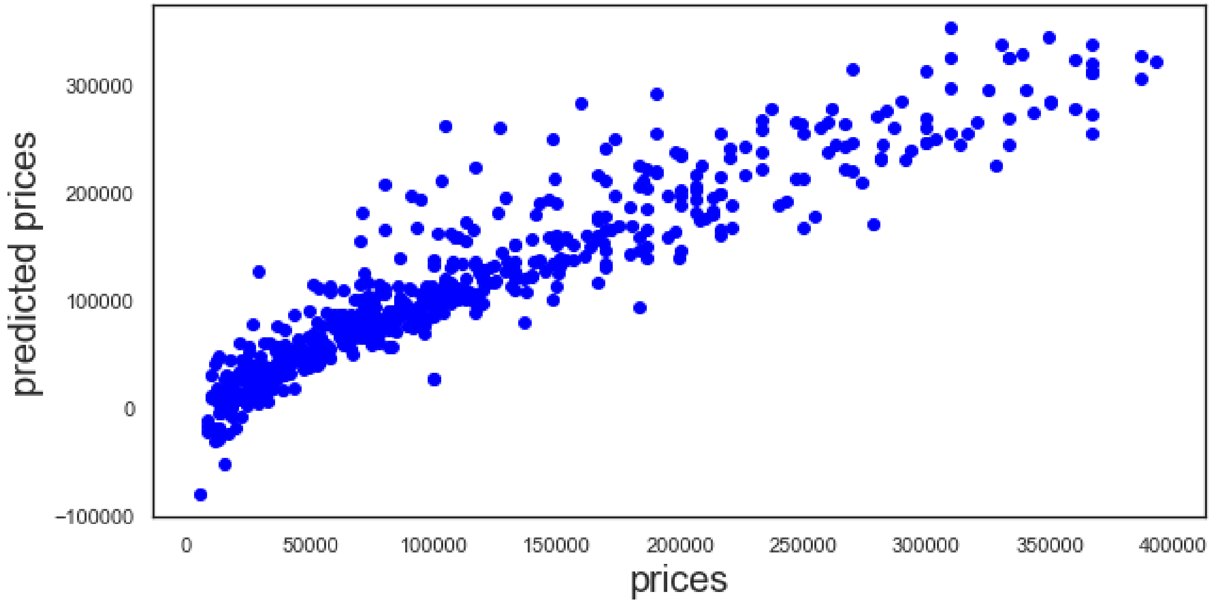
	alpha	RMSE
11	1.000	32406
10	0.900	32410
9	0.800	32414
8	0.700	32419
7	0.600	32424
6	0.500	32428
5	0.400	32433
4	0.300	32438
3	0.200	32446
2	0.100	32457
1	0.010	32479
0	0.001	32484
13	10.000	32497

train and show the result with the best parameters :

In [121…

```
Las = Lasso(normalize = True, alpha = 5)
T2LSR_Train_R2_score, T2LSR_Test_R2_score, T2LSR_Sqr_MSE, T2LSR_MSE, T2LSR_RunTime =
```

```
Training Info :

  Coefficient of determination (R2 score)  :  86.36%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  86.75%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.87
----------------------------------------------------------------
  Mean Absolute Error                      :  22063.4
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  1046191681.527863
----------------------------------------------------------------
  Square-Root of MSE                       :  32345
----------------------------------------------------------------
  Median Absolute Error                    :  13845.09
----------------------------------------------------------------
 Runtime of the program                   :  0.06
----------------------------------------------------------------
```



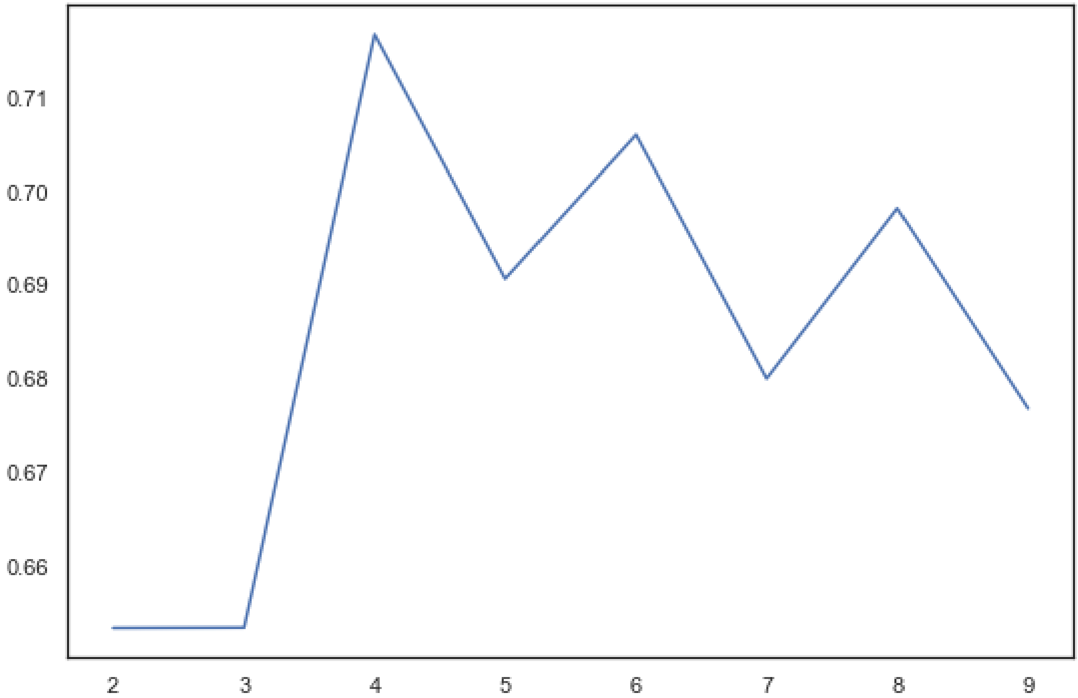
In [ ]:

Decision Tree Regressor

Cross Validation Score :

In [122…

```
DTR = DecisionTreeRegressor()
_cv = []
_Cv_scores = []
for i in range(2,10):
    cv_scores = np.mean(cross_val_score(DTR, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)
plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```




```
Cross Validation score is 0.716711 with CV = 4.0.
```

Finding the best parameters :

In [123…

```
__max_depth = [2,3,5,10,15,20,50,60,70,80,90,100,110,150]

_min_samples_split = []
_max_depth = []
_Sqr_MSE = []


for i in range(2,50):
    for j in __max_depth:
        DTR = DecisionTreeRegressor(random_state = 1,min_samples_split = i, max_dept
        y_pred = DTR.predict(x_test)
        Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
        _min_samples_split.append(i)
        _max_depth.append(j)
        _Sqr_MSE.append(Sqr_MSE)

my_dict = {'min_samples_split':_min_samples_split,'max_depth': _max_depth ,'RMSE':_S
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'])
```

Out[123…

	min_samples_split	max_depth	RMSE
69	6	150	38770
68	6	110	38770
67	6	100	38770
66	6	90	38770
65	6	80	38770
...	...	...	...
322	25	2	63524
112	10	2	63524
630	47	2	63524

	min_samples_split	max_depth	RMSE
420	32	2	63524
0	2	2	63524

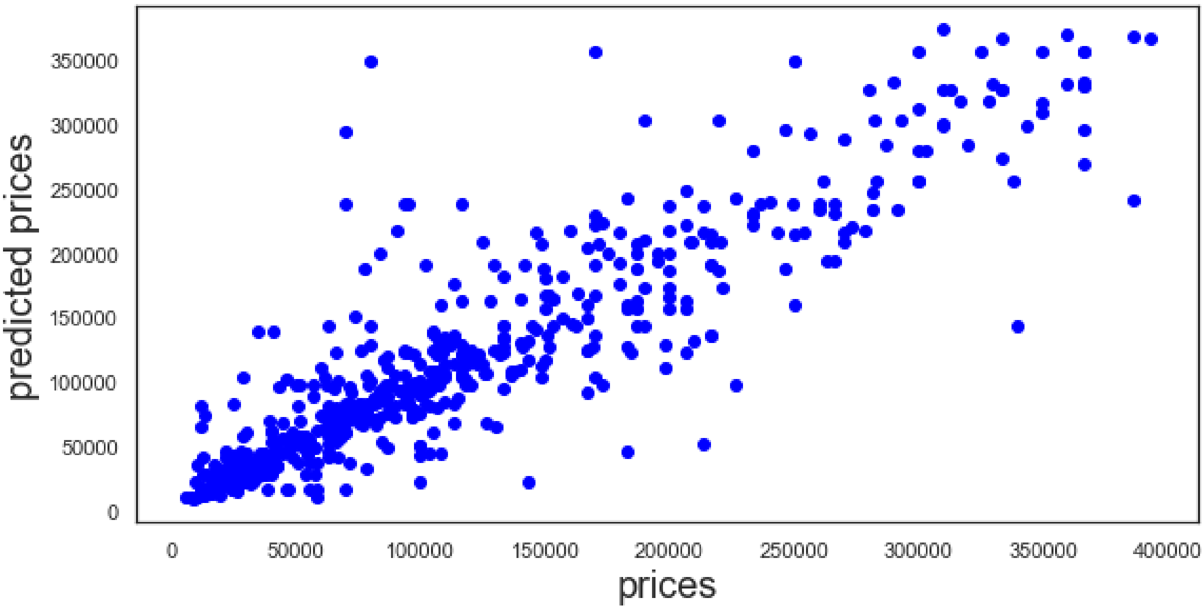
672 rows × 3 columns

train and show the result with the best parameters :

In [124…

```
DTR = DecisionTreeRegressor(random_state = 1,min_samples_split = 6, max_depth = 150)
T2DTR_Train_R2_score, T2DTR_Test_R2_score, T2DTR_Sqr_MSE, T2DTR_MSE, T2DTR_RunTime =
```

```
Training Info :

  Coefficient of determination (R2 score)  :  96.42%.

-----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  80.97%.
-----------------------------------------------------------------
  Explain Variance Score                   :  0.81
-----------------------------------------------------------------
  Mean Absolute Error                      :  23018.9
-----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  1503129619.5360315
-----------------------------------------------------------------
  Square-Root of MSE                       :  38770
-----------------------------------------------------------------
  Median Absolute Error                    :  11111.11
-----------------------------------------------------------------
 Runtime of the program                   :  0.07
-----------------------------------------------------------------
```



In [ ]:

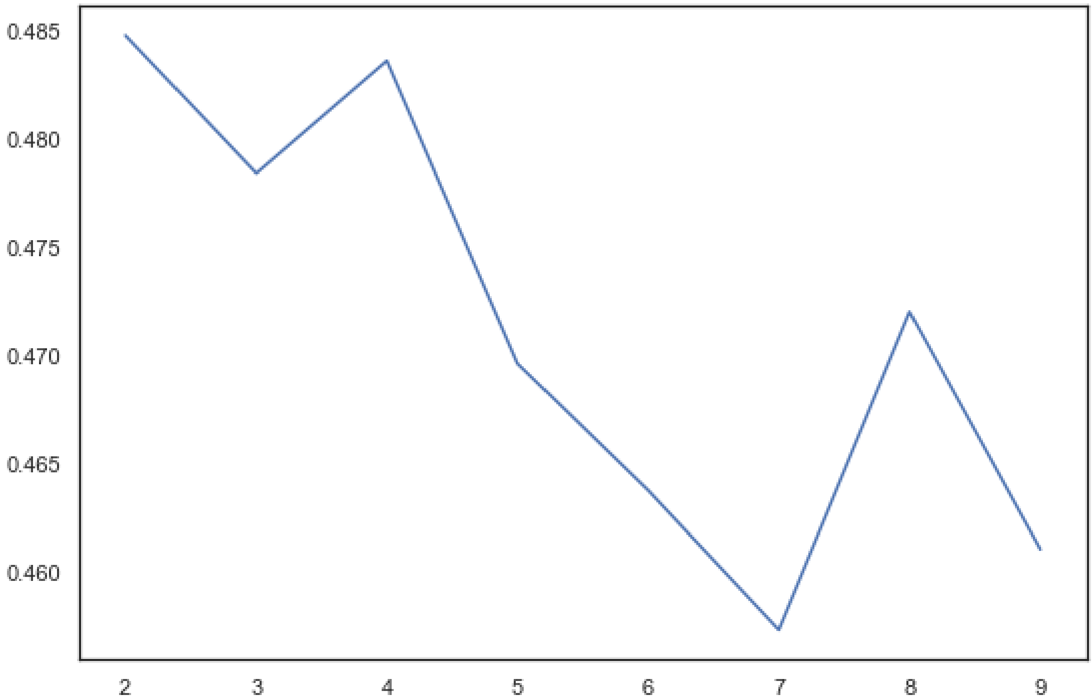
AdaBoostRegressor

Cross Validation Score :

In [125…

```
ABR  = AdaBoostRegressor()
```

```
_cv = []
_Cv_scores = []
for i in range(2,10):
    cv_scores = np.mean(cross_val_score(ABR, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)
plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```




```
Cross Validation score is 0.484798 with CV = 2.0.
```

Finding the best parameters :

In [126…

```
__n_estimators = [5, 10,15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85,
__learning_rate = [0.01, 0.1, 1, 2, 3, 4, 5, 10]

_n_estimators = []
_learning_rate = []
_Sqr_MSE = []

for i in __n_estimators:
    for j in __learning_rate:
        ABR  = AdaBoostRegressor(n_estimators = i, learning_rate = j).fit(x_train,y_
        y_pred = ABR.predict(x_test)
        Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
        _n_estimators.append(i)
        _learning_rate.append(j)
        _Sqr_MSE.append(Sqr_MSE)

my_dict = {'n_estimators':_n_estimators,'learning_rate': _learning_rate ,'RMSE':_Sqr
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'])
```

Out[126…

	n_estimators	learning_rate	RMSE
32	25	0.01	58188
16	15	0.01	58326
56	40	0.01	58407
64	45	0.01	58537
44	30	3.00	58585
...	...	...	...
95	60	10.00	204462
86	55	5.00	206921
23	15	10.00	216949
111	70	10.00	227900
15	10	10.00	289085

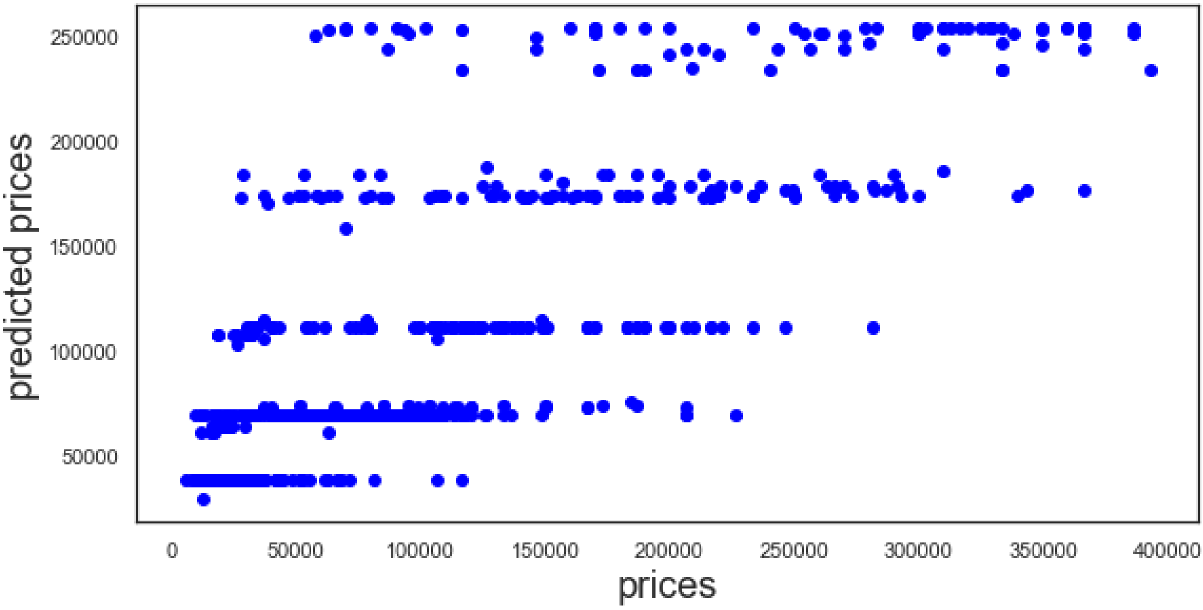
160 rows × 3 columns

train and show the result with the best parameters :

In [127…

```
ABR  = AdaBoostRegressor(n_estimators = 30, learning_rate = 0.01)
T2ABR_Train_R2_score, T2ABR_Test_R2_score, T2ABR_Sqr_MSE, T2ABR_MSE, T2ABR_RunTime =
```

```
Training Info :

  Coefficient of determination (R2 score)  :  54.58%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  56.83%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.57
----------------------------------------------------------------
  Mean Absolute Error                      :  42823.42
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  3410132833.590839
----------------------------------------------------------------
  Square-Root of MSE                       :  58396
----------------------------------------------------------------
  Median Absolute Error                    :  30030.71
----------------------------------------------------------------
 Runtime of the program                   :  0.56
----------------------------------------------------------------
```



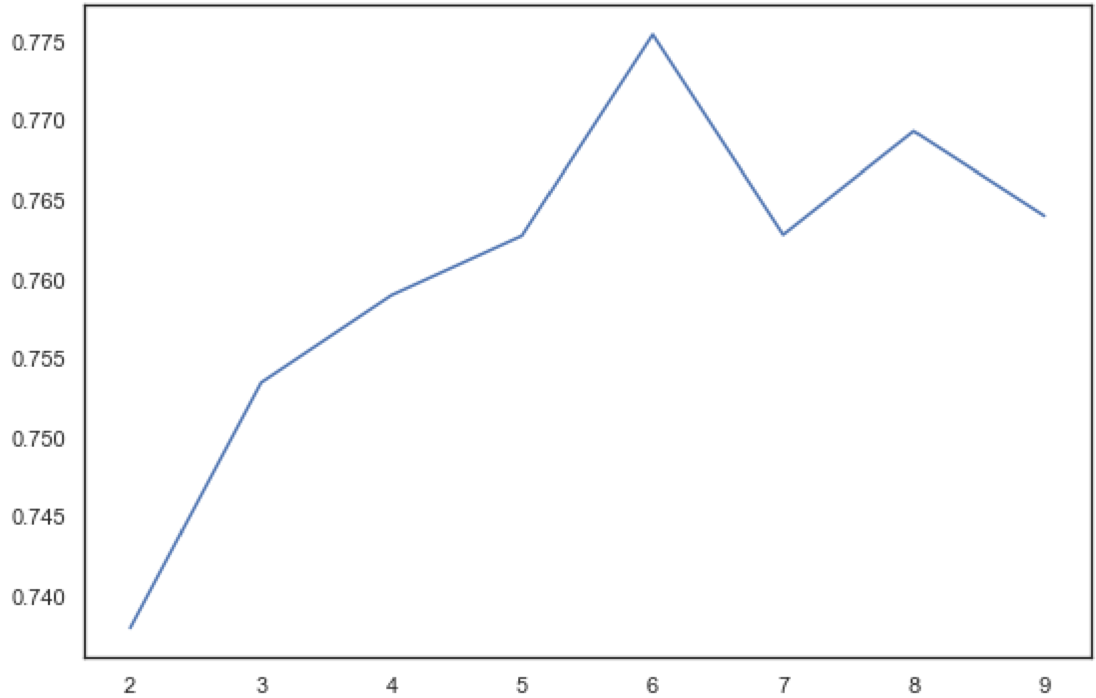
In [ ]:

RandomForestRegressor

Cross Validation Score :

In [128…

```
RFR  = RandomForestRegressor()
_cv = []
_Cv_scores = []
for i in range(2,10):
    cv_scores = np.mean(cross_val_score(RFR, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)
plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```




```
Cross Validation score is 0.775500 with CV = 6.0.
```

Finding the best parameters :

In [129…

```
__n_estimators = [5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100,150,200]
__min_samples_split = [2, 3, 4, 5,6,7,8,9, 10,15,20,25,30,40,50]

_n_estimators = []
_min_samples_split = []
_Sqr_MSE = []

for i in __n_estimators:
    for j in __min_samples_split:
        RFR = RandomForestRegressor(random_state = 1, n_jobs = -1, n_estimators = i
        y_pred = RFR.predict(x_test)
        Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
        _n_estimators.append(i)
        _min_samples_split.append(j)
        _Sqr_MSE.append(Sqr_MSE)

my_dict = {'n_estimators':_n_estimators,'min_samples_split': _min_samples_split ,'RM
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'])
```

Out[129…

	n_estimators	min_samples_split	RMSE
19	10	6	36043
45	30	2	36101
49	30	6	36106
25	10	20	36114
23	10	10	36164
...	...	...	...
0	5	2	37856
5	5	7	37982
3	5	5	38050

	n_estimators	min_samples_split	RMSE
2	5	4	38347
1	5	3	38619

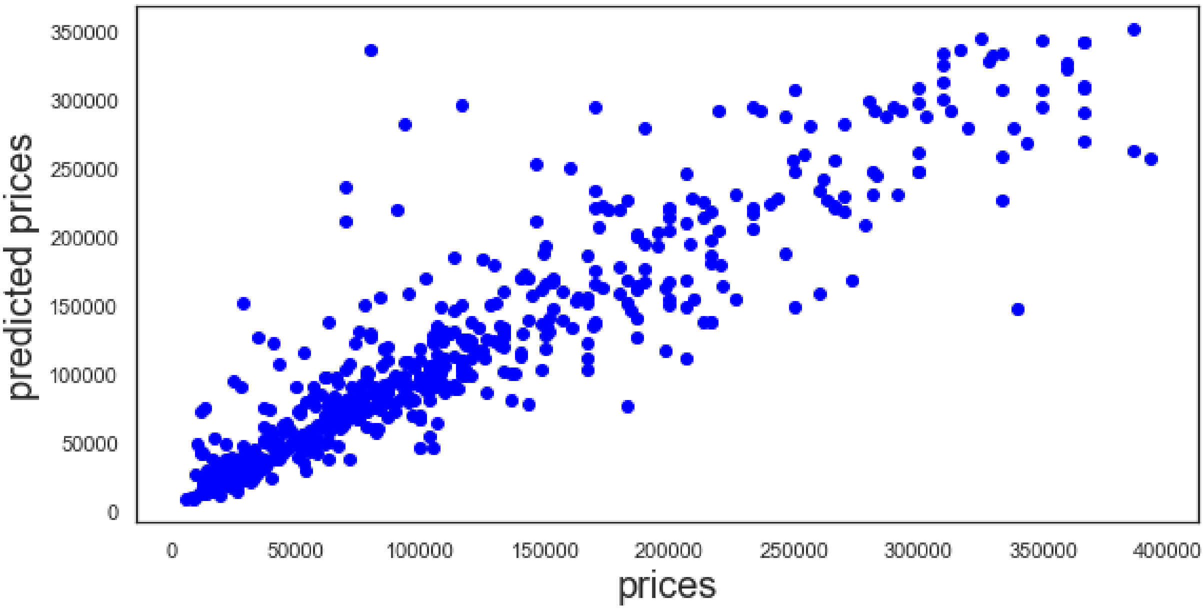
195 rows × 3 columns

train and show the result with the best parameters :

In [130…

```
RFR = RandomForestRegressor(random_state = 1, n_jobs = -1, n_estimators = 10 , min_s
T2RFR_Train_R2_score, T2RFR_Test_R2_score, T2RFR_Sqr_MSE, T2RFR_MSE, T2RFR_RunTime =
```

```
Training Info :

  Coefficient of determination (R2 score)  :  93.32%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  83.55%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.84
----------------------------------------------------------------
  Mean Absolute Error                      :  21705.98
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  1299096029.1120205
----------------------------------------------------------------
  Square-Root of MSE                       :  36043
----------------------------------------------------------------
  Median Absolute Error                    :  11789.58
----------------------------------------------------------------
 Runtime of the program                   :  0.16
----------------------------------------------------------------
```



In [ ]:

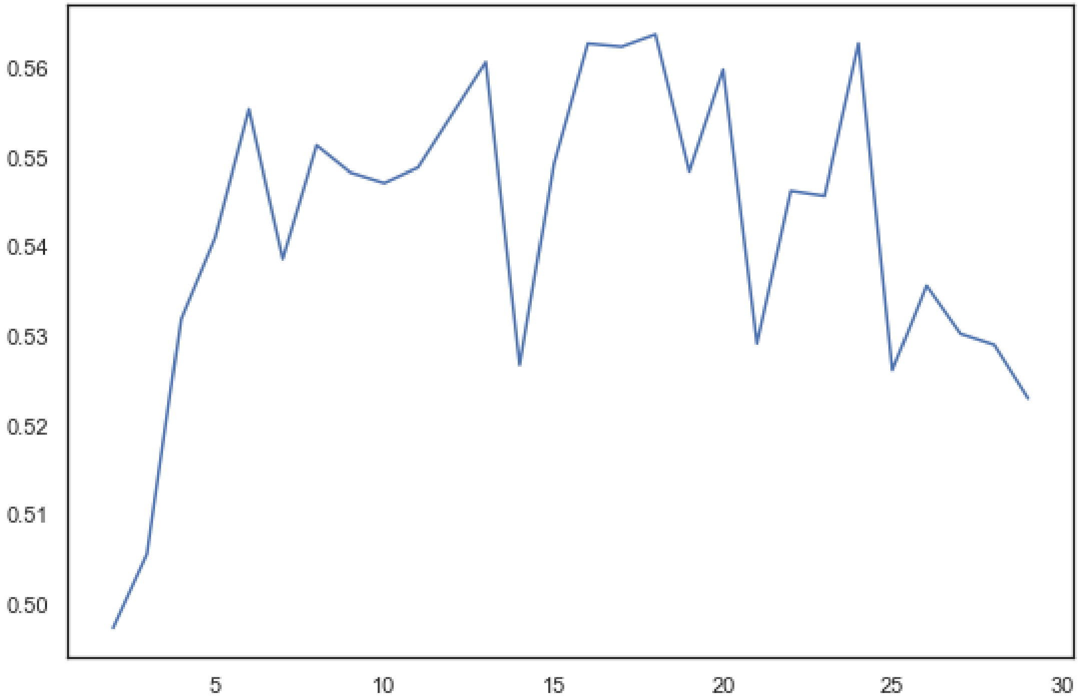
KNeighborsRegressor

Cross Validation Score :

In [131…

```
KNR  = KNeighborsRegressor()
```

```
_cv = []
_Cv_scores = []
for i in range(2,30):
    cv_scores = np.mean(cross_val_score(KNR, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)
plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```




```
Cross Validation score is 0.563755 with CV = 18.0.
```

Finding the best parameters :

In [132…

```
__weights = ['uniform', 'distance']

_n_neighbors = []
_weights = []
_Sqr_MSE = []

for i in range(2,50):
    for j in __weights:
        KNR = KNeighborsRegressor(n_jobs = -1, n_neighbors = i , weights = j).fit(x_
        y_pred = KNR.predict(x_test)
        Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
        _n_neighbors.append(i)
        _weights.append(j)
        _Sqr_MSE.append(Sqr_MSE)

my_dict = {'n_neighbors':_n_neighbors,'weights': _weights ,'RMSE':_Sqr_MSE}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'])
```

Out[132…

	n_neighbors	weights	RMSE

	n_neighbors	weights	RMSE
15	9	distance	50446
17	10	distance	50538
57	30	distance	50617
59	31	distance	50676
51	27	distance	50679
...	...	...	...
88	46	uniform	59665
84	44	uniform	59688
90	47	uniform	59801
92	48	uniform	59888
94	49	uniform	59889

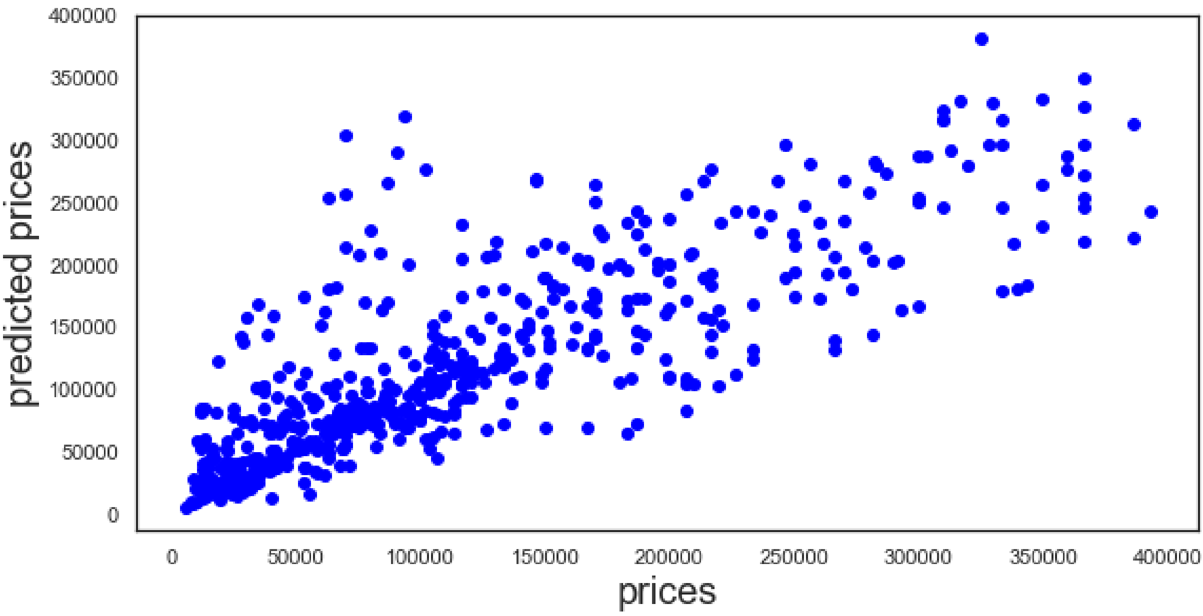
96 rows × 3 columns

train and show the result with the best parameters :

In [133…

```
KNR = KNeighborsRegressor(n_jobs = -1, n_neighbors = 9 , weights = 'distance')
T2KNR_Train_R2_score, T2KNR_Test_R2_score, T2KNR_Sqr_MSE, T2KNR_MSE, T2KNR_RunTime =
```

```
Training Info :

  Coefficient of determination (R2 score)  :  98.28%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  67.78%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.68
----------------------------------------------------------------
  Mean Absolute Error                      :  31696.84
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  2544795994.79584
----------------------------------------------------------------
  Square-Root of MSE                       :  50446
----------------------------------------------------------------
  Median Absolute Error                    :  16666.66
----------------------------------------------------------------
 Runtime of the program                  :  0.29
----------------------------------------------------------------
```



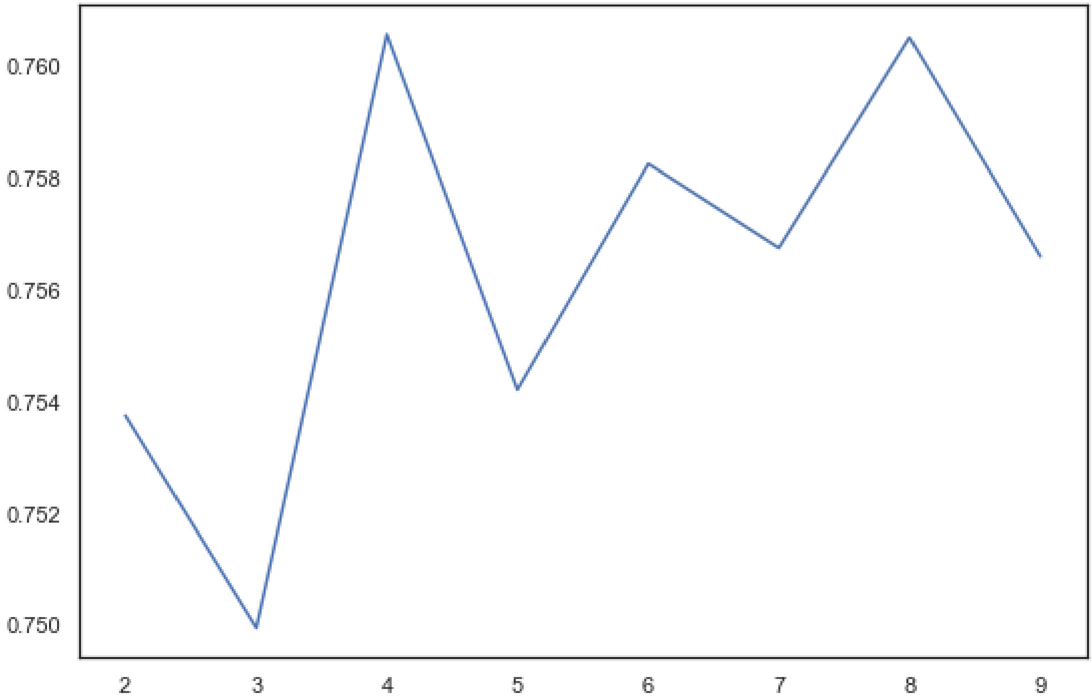
In [ ]:

GradientBoostingRegressor

Cross Validation Score :

In [134…

```
GBR  = GradientBoostingRegressor()
_cv = []
_Cv_scores = []
for i in range(2,10):
    cv_scores = np.mean(cross_val_score(GBR, x, y, cv=i))
    _Cv_scores.append(cv_scores)
    _cv.append(i)

my_dict = {'cv':_cv,'cv_scores': _Cv_scores}
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['cv_scores'],ascending=False, inplace = True)
plt.figure(figsize=(9,6))
plt.plot(_cv,_Cv_scores)
plt.show()

print(f"Cross Validation score is {my_df.iloc[0].cv_scores :0.6f} with CV = {my_df.i
```




```
Cross Validation score is 0.760586 with CV = 4.0.
```

Finding the best parameters :

In [135…

```
__learning_rate = [0.01, 0.1,0.2, 0.3, 0.4, 0.5]
__alpha = [0.01, 0.1,0.2, 0.3, 0.4, 0.5]
__max_depth = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

_learning_rate = []
_alpha = []
_max_depth = []
_Sqr_MSE = []

for i in __learning_rate:
    for j in __alpha:
        for k in __max_depth:
            GBR = GradientBoostingRegressor(learning_rate = i, alpha = j, max_depth
            y_pred = GBR.predict(x_test)
            Sqr_MSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
            _learning_rate.append(i)
            _alpha.append(j)
            _max_depth.append(k)
            _Sqr_MSE.append(Sqr_MSE)

my_dict = {'learning_rate':_learning_rate,'alpha': _alpha ,'max_depth':_max_depth,'R
my_df = pd.DataFrame(my_dict)
my_df.sort_values(by=['RMSE'])
```

Out[135…

	learning_rate	alpha	max_depth	RMSE
254	0.40	0.10	5	29538
178	0.20	0.50	9	29557
168	0.20	0.40	9	29573
194	0.30	0.10	5	29578
248	0.40	0.01	9	29598
...	...	...	...	...

	learning_rate	alpha	max_depth	RMSE
20	0.01	0.20	1	69943
30	0.01	0.30	1	69943
40	0.01	0.40	1	69943
10	0.01	0.10	1	69943
0	0.01	0.01	1	69943

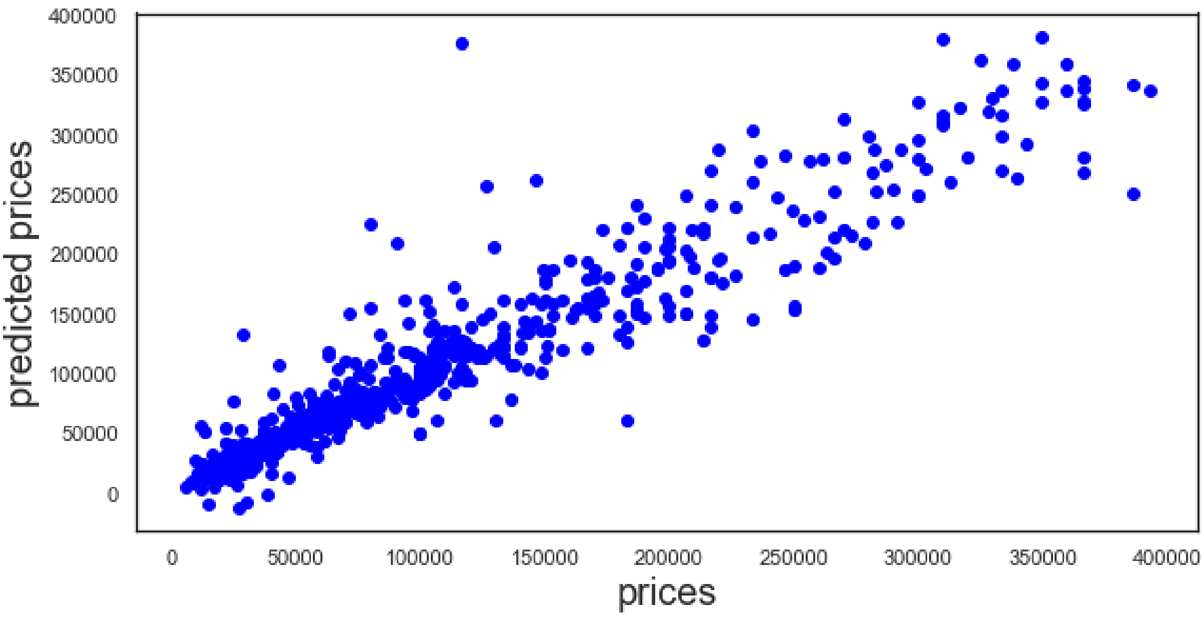
360 rows × 4 columns

train and show the result with the best parameters :

In [136…

```
GBR = GradientBoostingRegressor(learning_rate = 0.40, alpha = 0.1, max_depth = 9)
T2GBR_Train_R2_score, T2GBR_Test_R2_score, T2GBR_Sqr_MSE, T2GBR_MSE, T2GBR_RunTime =
```

```
Training Info :

  Coefficient of determination (R2 score)  :  97.75%.

----------------------------------------------------------------
Testing Info :

  Coefficient of determination (R2 score)  :  88.64%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.89
----------------------------------------------------------------
  Mean Absolute Error                      :  18125.76
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  897134407.1344947
----------------------------------------------------------------
  Square-Root of MSE                       :  29952
----------------------------------------------------------------
  Median Absolute Error                    :  9537.53
----------------------------------------------------------------
 Runtime of the program                  :  2.54
----------------------------------------------------------------
```



now we compare the model results

Comparing

In [137…

```
Temporary_T2_Result.sort
T2_Result_df.append(Temporary_T2_Result, ignore_index=True)
```

Out[137…

	Model Name	Train R2	Test R2	EVS	MAE	MSE	RMSE	MedAE
0	Ridge	0.8610	0.8709	0.87	22139.81	1.019670e+09	31932	14542.95
1	ElasticNet	0.8607	0.8709	0.87	22145.26	1.019684e+09	31932	14426.17
2	Lasso	0.8636	0.8675	0.87	22063.40	1.046192e+09	32345	13845.09
3	DecisionTreeRegressor	0.9642	0.8097	0.81	23018.90	1.503130e+09	38770	11111.11
4	AdaBoostRegressor	0.5458	0.5683	0.57	42823.42	3.410133e+09	58396	30030.71
5	RandomForestRegressor	0.9332	0.8355	0.84	21705.98	1.299096e+09	36043	11789.58
6	KNeighborsRegressor	0.9828	0.6778	0.68	31696.84	2.544796e+09	50446	16666.66
7	GradientBoostingRegressor	0.9775	0.8864	0.89	18125.76	8.971344e+08	29952	9537.53

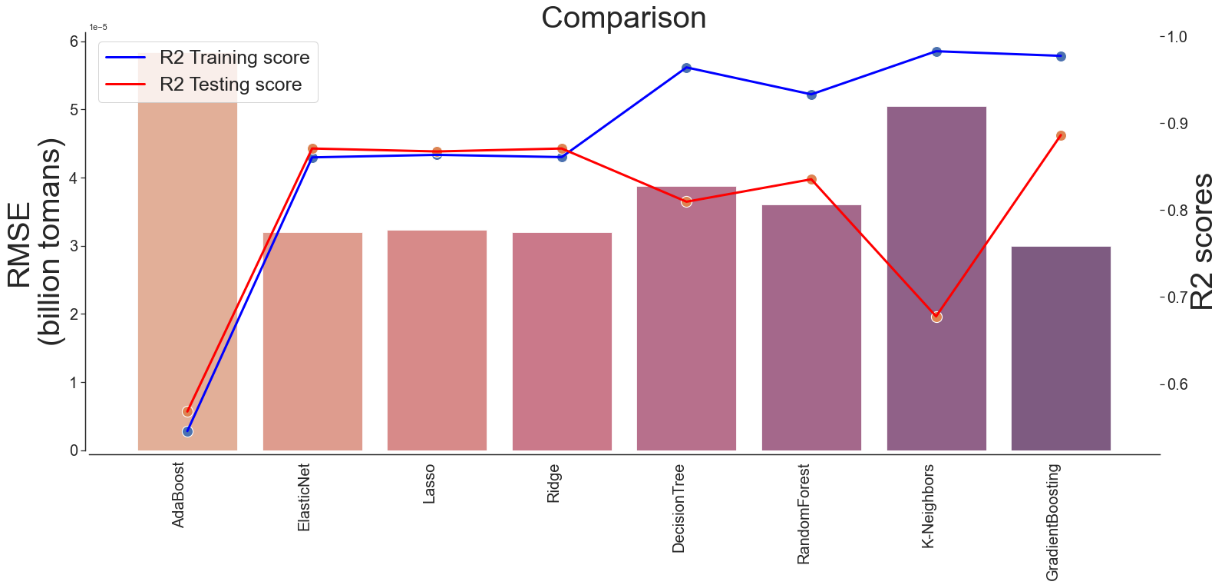
In [150…

```
T2models_score = pd.DataFrame({'Training r2 score': [T2ABR_Train_R2_score,  T2ENR_Tr
                               'Testing r2 score': [T2ABR_Test_R2_score,  T2ENR_Test_R
                               'RMSE': [ T2ABR_Sqr_MSE,  T2ENR_Sqr_MSE, T2LSR_Sqr_MSE,
                               index = [ 'AdaBoost', 'ElasticNet', 'Lasso', 'Ridge','D
print(T2models_score)
```

```
                  Training r2 score  Testing r2 score   RMSE
AdaBoost                     0.5458            0.5683  58396
ElasticNet                   0.8607            0.8709  31932
Lasso                        0.8636            0.8675  32345
Ridge                        0.8610            0.8709  31932
DecisionTree                 0.9642            0.8097  38770
RandomForest                 0.9332            0.8355  36043
K-Neighbors                  0.9828            0.6778  50446
GradientBoosting             0.9775            0.8864  29952
```

In [151…

```
fig, ax = plt.subplots(figsize=(25,10))
sb.set(style='white')
ax.set_title("Comparison", fontsize = 40)

ax = sb.barplot(x = list(T2models_score.index), y = T2models_score['RMSE']/100000000
ax.set_ylabel("RMSE\n(billion tomans)", fontsize = 40)

sec_ax = ax.twinx()
sec_ax = sb.lineplot(x = list(T2models_score.index), y = T2models_score['Training r2
sec_ax = sb.scatterplot(x = list(T2models_score.index), y = T2models_score['Training
sec_ax = sb.lineplot(x = list(T2models_score.index), y = T2models_score['Testing r2
sec_ax = sb.scatterplot(x = list(T2models_score.index), y = T2models_score['Testing
sec_ax.set_ylabel("R2 scores", fontsize = 40)
sec_ax.legend(labels = ['R2 Training score', 'R2 Testing score'], fontsize = 25)
sb.despine(offset = 5)
for label in ax.get_xticklabels():
    label.set_rotation(90)
    label.set_size(20)
    label.set_ha('right')
for label in ax.get_yticklabels():
    label.set_size(20)
for label in sec_ax.get_yticklabels():
    label.set_size(20)
plt.show()
```



as the results show, Gradient Boosting Regressor has the best result

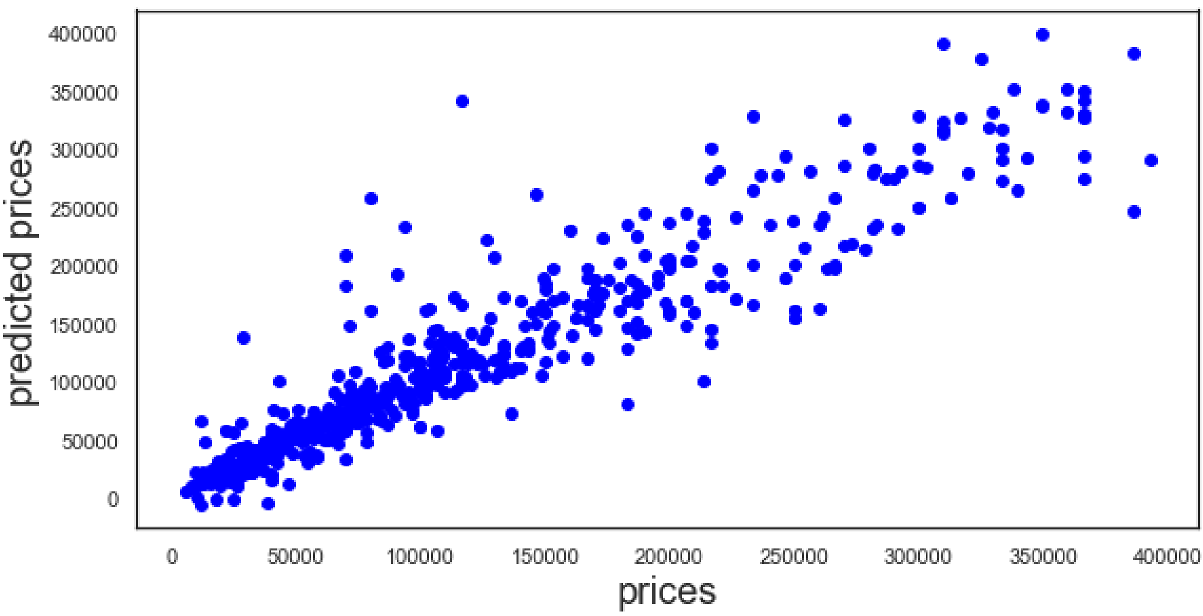
we train our modle with all rows and the results are as below:

In [140…

```
GBR = GradientBoostingRegressor(learning_rate = 0.40, alpha = 0.1, max_depth = 9)
Start_Time = time.time()
GBR.fit(x, y)
Score = round(GBR.score(x, y), 4)
EVS = round(metrics.explained_variance_score(y_test, y_pred), 2)
MAB = round(metrics.mean_absolute_error(y_test, y_pred), 2)
MSE = metrics.mean_squared_error(y_test, y_pred)
RMSE = round(np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
MedAE = round(metrics.median_absolute_error(y_test, y_pred), 2)
Finish_Time = time.time()
RunTime = Finish_Time - Start_Time
print(f"  Coefficient of determination (R2 score)  :  {Score:0.2%}.")
print("----------------------------------------------------------------")
print("  Explain Variance Score                    : ", EVS)
print("----------------------------------------------------------------")
print("  Mean Absolute Error                       : ", MAB)
print("----------------------------------------------------------------")
print("  MSE (Mean Squared Error)                  : ", MSE)
print("----------------------------------------------------------------")
print("  Square-Root of MSE                        : ", RMSE)
print("----------------------------------------------------------------")
print("  Median Absolute Error                     : ", MedAE)
print("----------------------------------------------------------------")
print(f" Runtime of the program                    :  {RunTime:0.2f}")
print("----------------------------------------------------------------")
plt.figure(figsize=(10,5))
plt.scatter(y_test, y_pred,color = 'blue')
plt.xlabel('prices', fontsize = 20)
plt.ylabel('predicted prices', fontsize = 20)
plt.show()
```

```
  Coefficient of determination (R2 score)  :  97.71%.
----------------------------------------------------------------
  Explain Variance Score                   :  0.88
----------------------------------------------------------------
  Mean Absolute Error                      :  18724.6
----------------------------------------------------------------
  MSE (Mean Squared Error)                 :  977726459.7605728
----------------------------------------------------------------
  Square-Root of MSE                       :  31269
----------------------------------------------------------------
```

```
 Median Absolute Error                          :  9791.15
----------------------------------------------------------------
 Runtime of the program                        :  2.28
----------------------------------------------------------------
```



In [141…

```
ddd = Dummies_Address_House_Price_df.copy()
ddd.drop("Price", axis=1, inplace=True)
```

Then a function has been wroten for preparing new data with new format for predicting the price

In [142…

```
def PrePare_Info(_Area,_Room,_Parking,_Warehouse,_Elevator,_Address):

    For_Pred = Dummies_Address_House_Price_df.copy()
    For_Pred.drop("Price", axis=1, inplace=True)
    For_Pred = For_Pred.iloc[0:0]
    For_Pred.reindex()

    Area = _Area
    Room = _Room
    Parking = _Parking
    Warehouse = _Warehouse
    Elevator = _Elevator
    Address = _Address

    my_dict = {'Area':Area,'Room': Room}

    Addresses = house_price_df.Address.value_counts()
    Addresses.sort_index(inplace = True)
    Addresses = Addresses.index
    for ad in Addresses:
        if ad == Address:
            D = {ad:[1]}
            my_dict.update(D)
        else:
            D = {ad:[0]}
            my_dict.update(D)
    my_dict
    if Parking == 1:
        D = {"Parking_True" : [1],'Parking_False' : [0]}
    else:
        D = {'Parking_True' : [0],'Parking_False' : [1]}
    my_dict.update(D)
```

```
    if Warehouse == 1:
        D = {'Warehouse_True' : [1],'Warehouse_False' : [0]}
    else:
        D = {'Warehouse_True' : [0],'Warehouse_False' : [1]}
    my_dict.update(D)
    if Elevator == 1:
        D = {'Elevator_True' : [1],'Elevator_False' : [0]}
    else:
        D = {'Elevator_True' : [0],'Elevator_False' : [1]}
    my_dict.update(D)

    For_Pred = pd.DataFrame(my_dict)

    return For_Pred
```

In [143…

```
For_Predict = PrePare_Info(150,3,1,0,0,'parand')
For_Predict
```

Out[143…

	Area	Room	Abazar	Abbasabad	Abuzar	Afsarieh	Ahang	Air force	Ajudaniye	Alborz Complex	...	Zaf
0	150	3	0	0	0	0	0	0	0	0	...	

1 rows × 195 columns

In [144…

```
y_pred = GBR.predict(For_Predict)
print("Predicted price = ", int(y_pred))
```

```
Predicted price =  213051
```

In [ ]:

FINISHED

In [ ]: