LINEAR REGRESSION MODEL

- Linear regression is a statistical modeling technique used to establish a relationship between a dependent variable and one or more independent variables.
- The goal of linear regression is to find the best-fitting line that minimizes the difference between the observed values and the predicted values.

Types

(1) SIMPLE LINEAR REGRESSION MODEL ~ Linear regression model that involves only one independent variable and one dependent variable.

- Equation ~ Y = MX + C (M = Coefficient / Slope & C = Intercept)

(2) MULTIPLE LINEAR REGRESSION MODEL ~ Linear regression model that involves more then one independent variable and one dependent variable.

- Equation ~ Y = M1X1 + M2X2 + M3X3.....+MnXn (M = Coefficient / Slope & C = Intercept)

Assumptions must be followed while building a perfect LR model

- LINEARITY ~ The relationship between the independent & dependent variables must be linear.
- NORMALITY OF RESIDUAL ~ The errors must be normally distributed.
- HOMOSCEDATICITY ~ The error terms must have constant variance.
- NO MULTI-COLLINEARITY ~ The independent variables should not be correlated with each other.
- NO ENDOGENEITY ~ There should be no relation between the errors & the independent variables.
- NO AUTO-CORRELATION ~ There should be no correlation between the error terms.

In [133]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [134]:
```
import warnings
warnings.filterwarnings('ignore')
```

In [135]:
```
df=pd.read_csv(r'C:\Users\lenovo\Downloads\USA_Housing.csv')
df
```

Out[135]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	4.09	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	5.13	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	3.26	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	4.23	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386
...	...	...	...	...	...	...	...
4995	60567.94414	7.830362	6.137356	3.46	22837.36103	1.060194e+06	USNS Williams\nFPO AP 30153-7653
4996	78491.27543	6.999135	6.576763	4.02	25616.11549	1.482618e+06	PSC 9258, Box 8489\nAPO AA 42991-3352
4997	63390.68689	7.250591	4.805081	2.13	33266.14549	1.030730e+06	4215 Tracy Garden Suite 076\nJoshualand, VA 01...
4998	68001.33124	5.534388	7.130144	5.44	42625.62016	1.198657e+06	USS Wallace\nFPO AE 73316
4999	65510.58180	5.992305	6.792336	4.07	46501.28380	1.298950e+06	37778 George Ridges Apt. 509\nEast Holly, NV 2...

5000 rows × 7 columns

In [136]:
```
df.shape
```

Out[136]:
```
(5000, 7)
```

In [137]:
```
df.columns
```

Out[137]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Avg. Area Number of Bedrooms', 'Area Population', 'Price', 'Address'],
      dtype='object')
```

In [138]:
```
### Dropping this ['Avg. Area Number of Bedrooms'] variable bcoz it's not a significant variable
### We have ['Avg. Area Number of Rooms'], so no need of ['Avg. Area Number of Bedrooms']
```

In [139]:
```
df=df.drop(['Avg. Area Number of Bedrooms'],axis=1)
df
```

Out[139]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386
...	...	...	...	...	...	...
4995	60567.94414	7.830362	6.137356	22837.36103	1.060194e+06	USNS Williams\nFPO AP 30153-7653
4996	78491.27543	6.999135	6.576763	25616.11549	1.482618e+06	PSC 9258, Box 8489\nAPO AA 42991-3352
4997	63390.68689	7.250591	4.805081	33266.14549	1.030730e+06	4215 Tracy Garden Suite 076\nJoshualand, VA 01...
4998	68001.33124	5.534388	7.130144	42625.62016	1.198657e+06	USS Wallace\nFPO AE 73316
4999	65510.58180	5.992305	6.792336	46501.28380	1.298950e+06	37778 George Ridges Apt. 509\nEast Holly, NV 2...

5000 rows × 6 columns

In [140]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Avg. Area Income          4990 non-null   float64
 1   Avg. Area House Age       5000 non-null   float64
 2   Avg. Area Number of Rooms 4995 non-null   float64
 3   Area Population           5000 non-null   float64
 4   Price                     5000 non-null   float64
 5   Address                   5000 non-null   object 
dtypes: float64(5), object(1)
memory usage: 234.5+ KB
```

Data Pre-processing

(1) Null Values checking & treating them

In [141]:
```
df.head()
```

Out[141]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

In [142]:
```
df.isnull().sum()
```

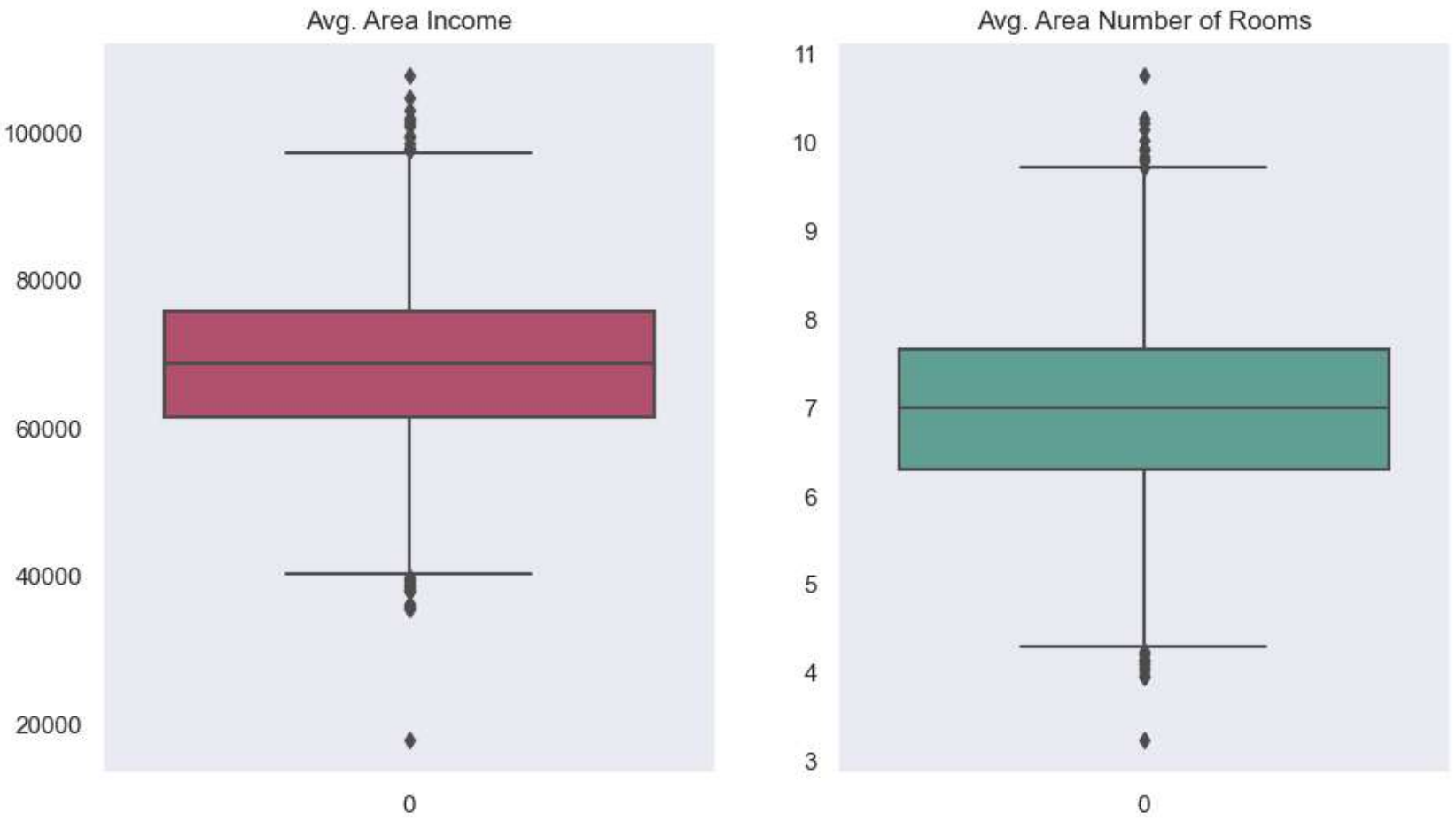
Out[142]:
```
Avg. Area Income            10
Avg. Area House Age          0
Avg. Area Number of Rooms    5
Area Population              0
Price                        0
Address                      0
dtype: int64
```

In [143]:
```
df.isnull().sum()/len(df)*100
```

Out[143]:
```
Avg. Area Income              0.2
Avg. Area House Age           0.0
Avg. Area Number of Rooms     0.1
Area Population               0.0
Price                         0.0
Address                       0.0
dtype: float64
```

In [144]:
```
# Avg. area income,Avg. Area Number of Rooms & Avg. Area Number of Bedrooms ~ NULL TREATMENT
```

In [145]:
```
plt.figure(figsize=(17,6))
plt.subplot(1,3,1)
plt.title('Avg. Area Income')
sns.boxplot(df['Avg. Area Income'],palette='flare')

plt.subplot(1,3,2)
plt.title('Avg. Area Number of Rooms')
sns.boxplot(df['Avg. Area Number of Rooms'],palette='dark:#5A9_r')

plt.show()
```



In [146]:
```
df['Avg. Area Income']=df['Avg. Area Income'].fillna(df['Avg. Area Income'].median())
```

In [147]:
```
df['Avg. Area Number of Rooms']=df['Avg. Area Number of Rooms'].fillna(df['Avg. Area Number of Rooms'].median
```

In [148]:
```
df.isnull().sum()
```

Out[148]:
```
Avg. Area Income              0
Avg. Area House Age           0
Avg. Area Number of Rooms     0
Area Population               0
Price                         0
Address                       0
dtype: int64
```

In [149]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Avg. Area Income             5000 non-null   float64
 1   Avg. Area House Age          5000 non-null   float64
 2   Avg. Area Number of Rooms    5000 non-null   float64
 3   Area Population              5000 non-null   float64
 4   Price                        5000 non-null   float64
 5   Address                      5000 non-null   object 
dtypes: float64(5), object(1)
memory usage: 234.5+ KB
```

(2) Checking the categorical variable whether it requires ENCODING CONCEPT / not

In [150]:
```
df.head()
```

Out[150]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price	Address
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06	188 Johnson Views Suite 079\nLake Kathleen, CA...
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06	9127 Elizabeth Stravenue\nDanieltown, WI 06482...
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06	USS Barnett\nFPO AP 44820
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05	USNS Raymond\nFPO AE 09386

In [151]:
```
df['Address'].unique()
```

Out[151]:
```
array(['208 Michael Ferry Apt. 674\nLaurabury, NE 37010-5101',
       '188 Johnson Views Suite 079\nLake Kathleen, CA 48958',
       '9127 Elizabeth Stravenue\nDanieltown, WI 06482-3489', ...,
       '4215 Tracy Garden Suite 076\nJoshualand, VA 01707-9165',
       'USS Wallace\nFPO AE 73316',
       '37778 George Ridges Apt. 509\nEast Holly, NV 29290-3595'],
      dtype=object)
```

In [152]:
```
# Doing ANOVA Testing to check whether the ADDRESS variable is a significant variable or not.....
```

In [153]:
```
# 1st we have to do Label Encoding on address column before doing ANOVA Testing....
```

In [154]:
```
df['Address']=df['Address'].astype('category')
df['Address']=df['Address'].cat.codes
```

In [155]:
```
import statsmodels.api as sm
from statsmodels.formula.api import ols

model=ols('Price ~ Address',data=df).fit()
anova_result=sm.stats.anova_lm(model,type=2)
print(anova_result)
```

```
                df        sum_sq       mean_sq          F    PR(>F)
Address        1.0  4.729103e+10  4.729103e+10   0.379215  0.538051
Residual  4998.0  6.232883e+14  1.247075e+11        NaN       NaN
```

In [156]:
```
# As here the P value(P>0.05) so we can't consider it as a significant variable....NO need of Encoding Concep
```

In [157]:
```
# We should drop the variable now....
```

In [158]:
```
df=df.drop(['Address'],axis=1)
df
```

Out[158]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05
...	...	...	...	...	...
4995	60567.94414	7.830362	6.137356	22837.36103	1.060194e+06
4996	78491.27543	6.999135	6.576763	25616.11549	1.482618e+06
4997	63390.68689	7.250591	4.805081	33266.14549	1.030730e+06
4998	68001.33124	5.534388	7.130144	42625.62016	1.198657e+06
4999	65510.58180	5.992305	6.792336	46501.28380	1.298950e+06

5000 rows × 5 columns

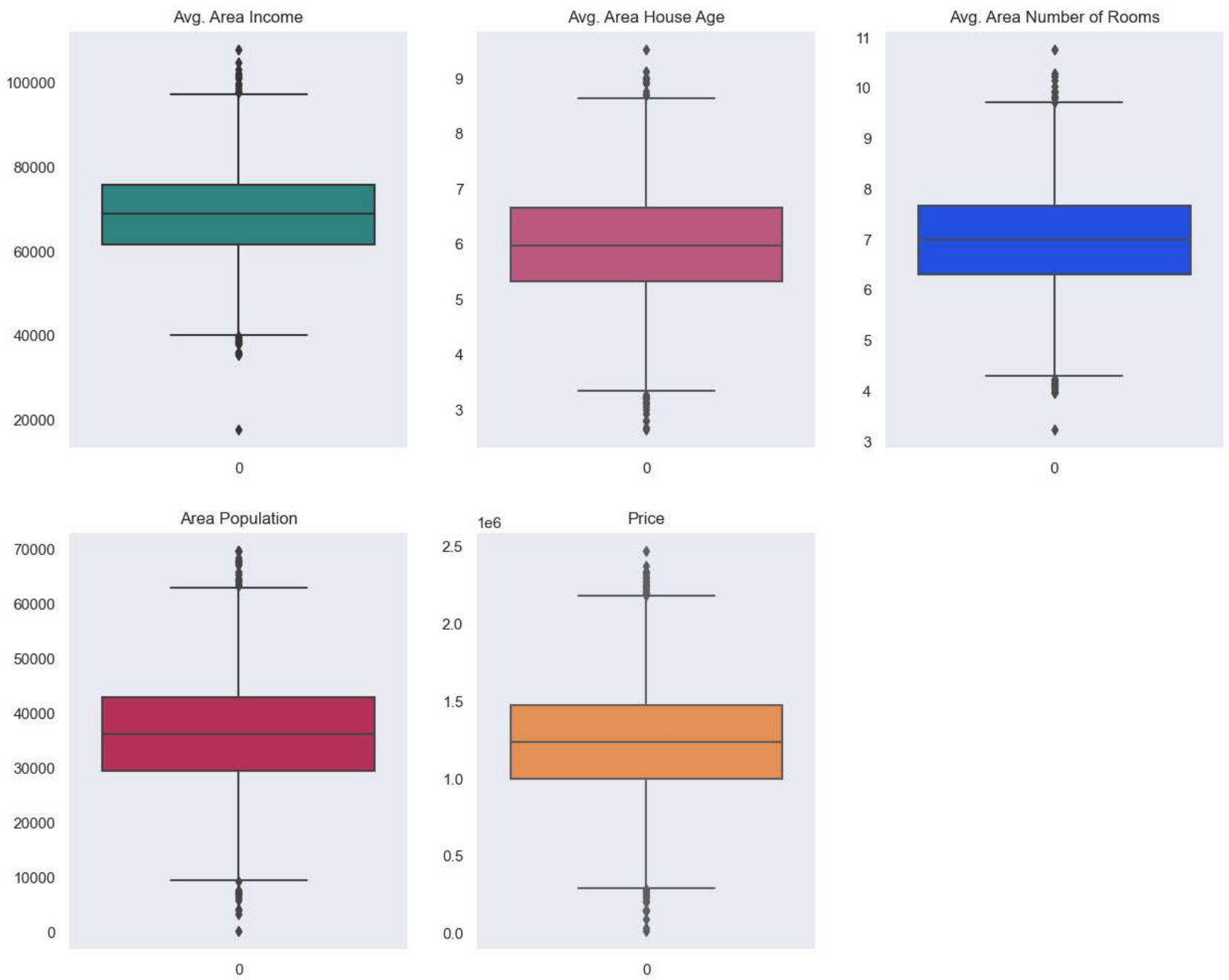
(3) Outliers checking & handling them

In [159]:
```
df.head()
```

Out[159]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05

In [160]:
```
plt.figure(figsize=(15,12))
plt.subplot(2,3,1)
plt.title('Avg. Area Income')
sns.boxplot(df['Avg. Area Income'],palette='viridis')

plt.subplot(2,3,2)
plt.title('Avg. Area House Age')
sns.boxplot(df['Avg. Area House Age'],palette='plasma')

plt.subplot(2,3,3)
plt.title('Avg. Area Number of Rooms')
sns.boxplot(df['Avg. Area Number of Rooms'],palette='bright')

plt.subplot(2,3,4)
plt.title('Area Population')
sns.boxplot(df['Area Population'],palette='rocket')

plt.subplot(2,3,5)
plt.title('Price')
sns.boxplot(df['Price'],palette='YlOrRd')

plt.show()
```



In [161]:
```
def distplots(col):
    sns.distplot(df[col],color='purple')
    sns.set(style='dark')
    plt.show()

for i in list(df.columns)[0:]:
    distplots(i)
```

In [162]:
```
# We can clearly see there are outliers.....need to treat them
```

In [163]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 5 columns):
 #   Column                       Non-Null Count  Dtype  
---  ------                       --------------  -----  
 0   Avg. Area Income             5000 non-null   float64
 1   Avg. Area House Age          5000 non-null   float64
 2   Avg. Area Number of Rooms    5000 non-null   float64
 3   Area Population              5000 non-null   float64
 4   Price                        5000 non-null   float64
dtypes: float64(5)
memory usage: 195.4 KB
```

In [164]:
```
df.describe()
```

Out[164]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68585.145895	5.977222	6.987708	36163.516039	1.232073e+06
std	10640.538021	0.991456	1.005435	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	172.610686	1.593866e+04
25%	61485.150192	5.322283	6.299692	29403.928700	9.975771e+05
50%	68797.671885	5.970429	7.002940	36199.406690	1.232669e+06
75%	75766.519103	6.650808	7.665281	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	69621.713380	2.469066e+06

In [165]:
```
new_df=df.copy()
```

In [166]:
```
new_df.head()
```

Out[166]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05

In [167]:
```
Q1=df['Avg. Area Income'].quantile(0.25)
Q3=df['Avg. Area Income'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [168]:
```
new_df['Avg. Area Income']=np.where(new_df['Avg. Area Income']>Higher_limit,Higher_limit,
                                    np.where(new_df['Avg. Area Income']<Lower_limit,Lower_limit,new_df['Avg. Area
```

In [169]:
```
Q1=df['Avg. Area House Age'].quantile(0.25)
Q3=df['Avg. Area House Age'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [170]:
```
new_df['Avg. Area House Age']=np.where(new_df['Avg. Area House Age']>Higher_limit,Higher_limit,
                                    np.where(new_df['Avg. Area House Age']<Lower_limit,Lower_limit,new_df['Avg. Ar
```

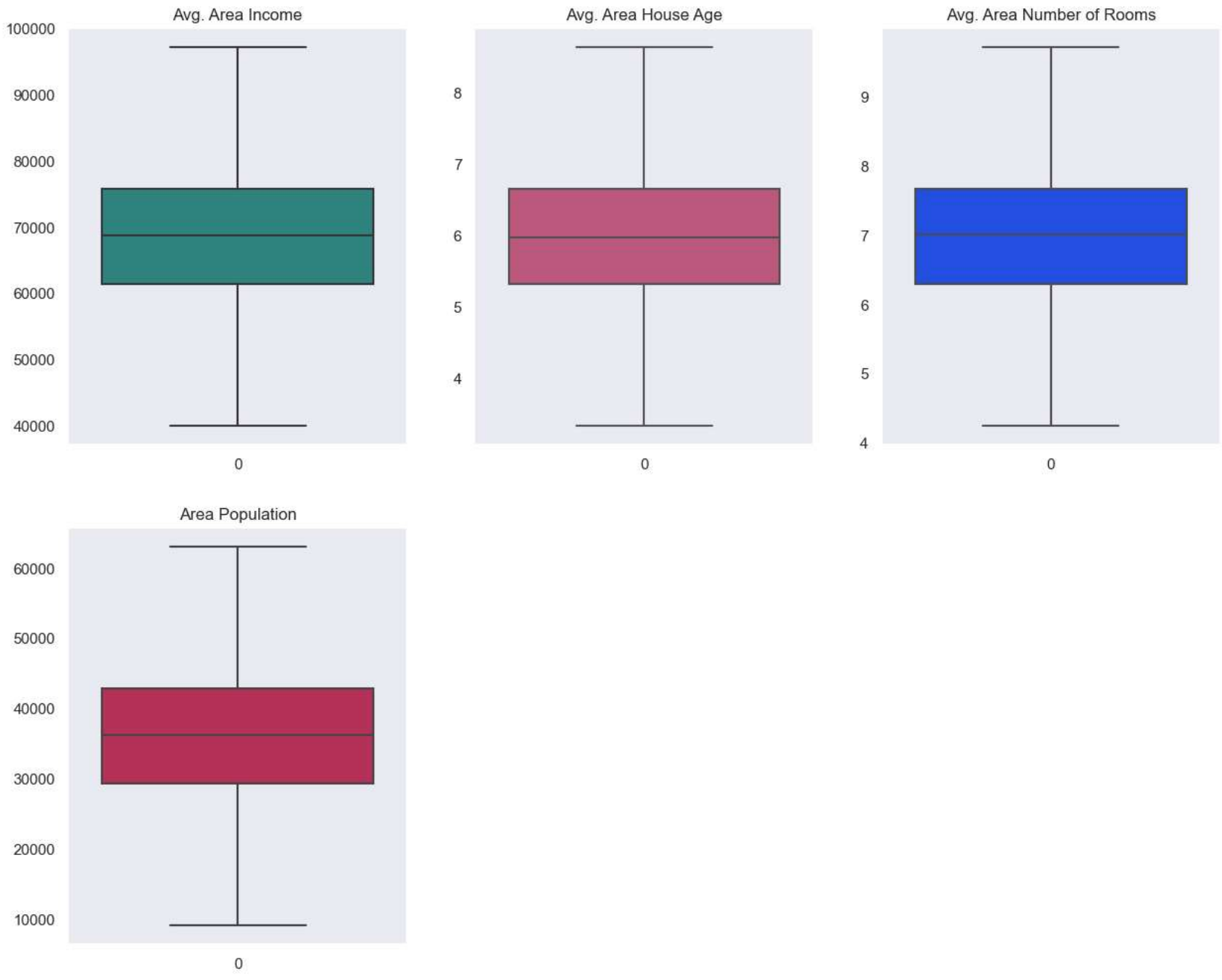
In [171]:
```
Q1=df['Avg. Area Number of Rooms'].quantile(0.25)
Q3=df['Avg. Area Number of Rooms'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [172]:
```
new_df['Avg. Area Number of Rooms']=np.where(new_df['Avg. Area Number of Rooms']>Higher_limit,Higher_limit,
                                    np.where(new_df['Avg. Area Number of Rooms']<Lower_limit,Lower_limit,new_df['A
```

In [173]:
```
Q1=df['Area Population'].quantile(0.25)
Q3=df['Area Population'].quantile(0.75)
IQR=Q3-Q1
Higher_limit=Q3+1.5*IQR
Lower_limit=Q1-1.5*IQR
```

In [174]:
```
new_df['Area Population']=np.where(new_df['Area Population']>Higher_limit,Higher_limit,
                                    np.where(new_df['Area Population']<Lower_limit,Lower_limit,new_df['Area Popula
```

In [175]:
```
plt.figure(figsize=(15,12))
plt.subplot(2,3,1)
plt.title('Avg. Area Income')
sns.boxplot(new_df['Avg. Area Income'],palette='viridis')

plt.subplot(2,3,2)
plt.title('Avg. Area House Age')
sns.boxplot(new_df['Avg. Area House Age'],palette='plasma')

plt.subplot(2,3,3)
plt.title('Avg. Area Number of Rooms')
sns.boxplot(new_df['Avg. Area Number of Rooms'],palette='bright')

plt.subplot(2,3,4)
plt.title('Area Population')
sns.boxplot(new_df['Area Population'],palette='rocket')

plt.show()
```



In [176]:
```
# Can clearly see now all outliers are treated with Capping method (np.where(condition,x,y))........
```

(4) Feature Scaling

Feature scaling is done only on independent variables

In [177]:
```
new_df.head()
```

Out[177]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05

In [178]:
```
# We need the split the data
```

In [179]:
```
X=new_df.drop(['Price'],axis=1)
```

In [180]:
```
X.head()   #all independent variables
```

Out[180]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population
0	79545.45857	5.682861	7.009188	23086.80050
1	79248.64245	6.002900	6.730821	40173.07217
2	61287.06718	5.865890	8.512727	36882.15940
3	63345.24005	7.188236	5.586729	34310.24283
4	59982.19723	5.040555	7.839388	26354.10947

In [181]:
```
Y=new_df.Price.copy()
```

In [182]:
```
pd.DataFrame(Y).head()     #Dependent variable
```

Out[182]:

	Price
0	1.059034e+06
1	1.505891e+06
2	1.058988e+06
3	1.260617e+06
4	6.309435e+05

In [183]:
```
from sklearn.preprocessing import StandardScaler
```

In [184]:
```
Std=StandardScaler()
```

In [185]:
```
fs_x=Std.fit_transform(X)
pd.DataFrame(fs_x)
```

Out[185]:

	0	1	2	3
0	1.036382	-0.298541	0.021620	-1.325622
1	1.008309	0.025747	-0.256381	0.407049
2	-0.690457	-0.113082	1.523179	0.073326
3	-0.495800	1.226822	-1.398967	-0.187484
4	-0.813869	-0.949376	0.850726	-0.994293
...	...	...	...	...
4995	-0.758470	1.877474	-0.849064	-1.350917
4996	0.936679	1.035210	-0.410236	-1.069131
4997	-0.491501	1.290004	-2.179585	-0.293363
4998	-0.055437	-0.448985	0.142416	0.655755
4999	-0.291006	0.015012	-0.194947	1.048775

5000 rows × 4 columns

HEATMAP Visualization to check the Multi-collinearity

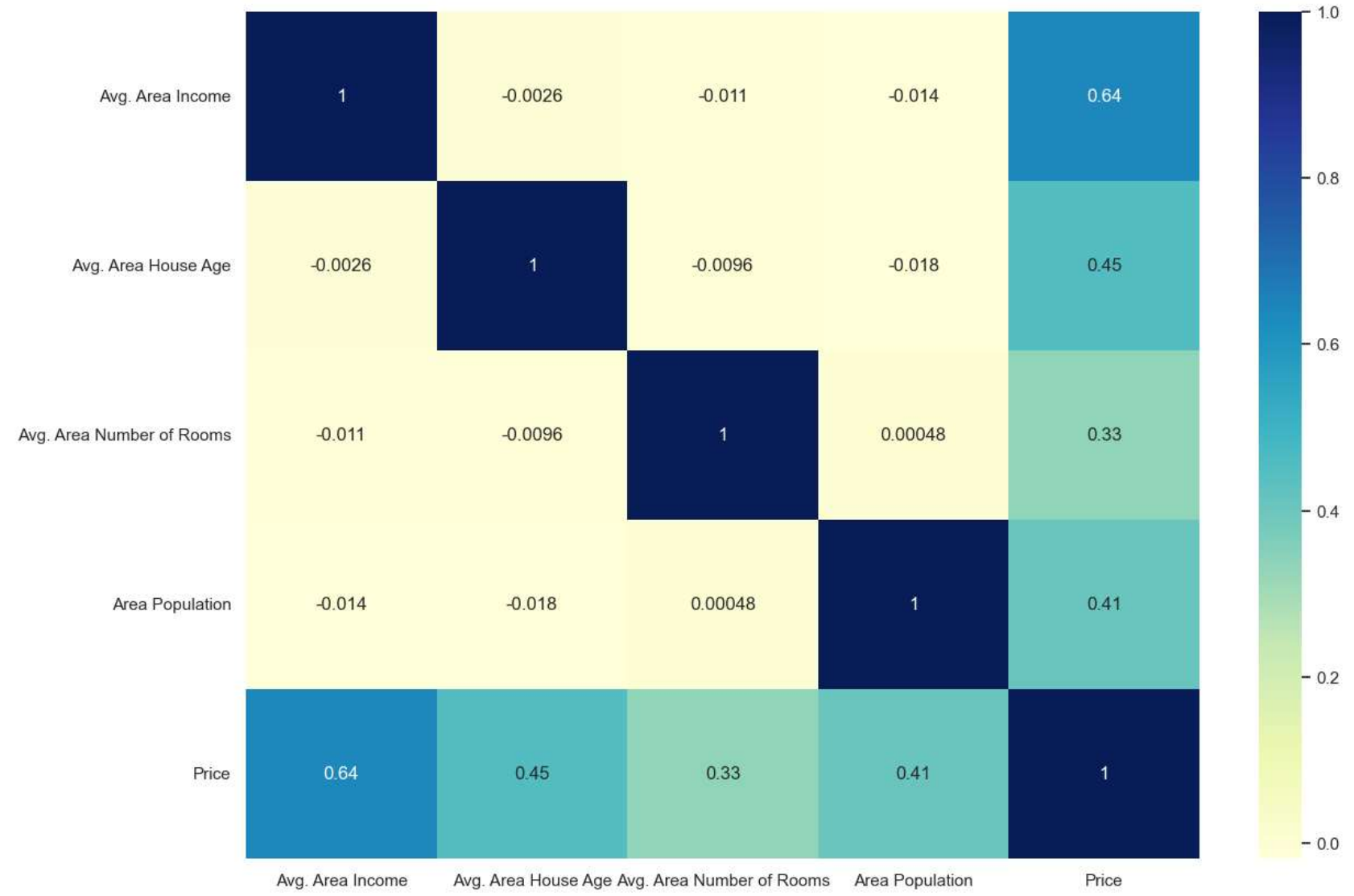
- Multicollinearity occurs when two or more independent variables in a regression model are highly correlated with each other.
- In the HEATMAP, if high correlation values (close to 1 or -1) exists betwn. pairs of independent variables.....Multicollinearity exists.
- If also the value of 2 or more independent variables are same then also they are higly correlated.
- Need to drop one or more correlated variables or can treat them using regularization (Lasso & Ridge) techniques.

In [186]:
```
new_df.head()
```

Out[186]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06
4	59982.19723	5.040555	7.839388	26354.10947	6.309435e+05

In [187]:
```
plt.figure(figsize=(14,10))
corr=new_df.corr()
sns.heatmap(corr,annot=True,cmap='YlGnBu')
plt.show()
```



In [188]:
```
# We can clearly see there is no multi-collinearity betwn. the independent variables....
```

VIF - Variance Inflation Factor

One of the stats method to check multi-collinearity

In [189]:
```
# If any feature VIF value is > 5 then there is a multi-collinearity....
```

In [190]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [191]:
```
ind_variable=fs_x
```

In [192]:
```
ind_variable.shape
```

Out[192]: (5000, 4)

In [193]:
```
VIF=pd.DataFrame()
```

In [194]:
```
VIF['Features']=X.columns
```

In [195]:
```
VIF['Variance Inflation Factor']=[variance_inflation_factor(ind_variable,i) for i in range(ind_variable.shape
```

In [196]:
```
VIF
```

Out[196]:

	Features	Variance Inflation Factor
0	Avg. Area Income	1.000335
1	Avg. Area House Age	1.000432
2	Avg. Area Number of Rooms	1.000214
3	Area Population	1.000537

In [197]:
```
# NO VIF value is > 5......so NO Multi-collinearity
```

TRAIN - TEST SPLIT Method

- The train-test split method is a ML method which is used to split your data into a training set & a testing set
- By this we can train our models on training set, & then test their accuracy on the unseen testing set
- Training set ~ 70-80% of data & Testing set ~ 20-30% of data

In [198]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.25,random_state=101)
```

In [199]:
```
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[199]:
```
((3750, 4), (1250, 4), (3750,), (1250,))
```

Building a Linear Regression Model

Approach - 1 (Linear Regression Method)

In [200]:
```
from sklearn.linear_model import LinearRegression
Model=LinearRegression()
Model.fit(x_train,y_train)
```

Out[200]:
```
LinearRegression()
```

In [201]:
```
y_test_pred_value=Model.predict(x_test)
y_test_pred_value
```

Out[201]:
```
array([1258927.55438276,  818030.31383002, 1745948.45303454, ...,
       1119387.19935068,  717217.37633985, 1516456.6242839 ])
```

In [202]:
```
y_test
```

Out[202]:
```
1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
            ...
1881    1.727211e+06
2800    1.707270e+06
1216    1.167450e+06
1648    7.241217e+05
3063    1.561234e+06
Name: Price, Length: 1250, dtype: float64
```

In [203]:
```
y_train_pred_value=Model.predict(x_train)
y_train_pred_value
```

Out[203]:
```
array([ 976450.46193941, 1006891.19880824, 1309553.93680634, ...,
        976148.79235854,  921184.48675295, 2134163.10702162])
```

In [204]:
```
y_train
```

Out[204]:
```
3430    9.958132e+05
3979    8.735092e+05
2153    1.360502e+06
4764    1.742351e+06
36      1.233220e+06
            ...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 3750, dtype: float64
```

Checking the accuracy of the model

In [205]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_test_pred_value)
```

Out[205]:
```
0.913609424096595
```

In [206]:
```
r2_score(y_train,y_train_pred_value)
```

Out[206]:
```
0.9164810029819419
```

Approach - 2 (OLS Method)

In [207]:
```
from statsmodels.regression.linear_model import OLS
import statsmodels.regression.linear_model as srl
```

In [208]:
```
Reg_model=srl.OLS(endog=y_train,exog=x_train).fit()
```

In [209]:
```
Reg_model.summary()
```

Out[209]:

OLS Regression Results

Dep. Variable:	Price	**R-squared (uncentered):**	0.964
Model:	OLS	**Adj. R-squared (uncentered):**	0.964
Method:	Least Squares	**F-statistic:**	2.513e+04
Date:	Fri, 14 Jul 2023	**Prob (F-statistic):**	0.00
Time:	14:19:13	**Log-Likelihood:**	-51813.
No. Observations:	3750	**AIC:**	1.036e+05
Df Residuals:	3746	**BIC:**	1.037e+05
Df Model:	4		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
Avg. Area Income	10.2138	0.314	32.572	0.000	9.599	10.829
Avg. Area House Age	4.928e+04	3477.982	14.169	0.000	4.25e+04	5.61e+04
Avg. Area Number of Rooms	-8043.4871	3213.779	-2.503	0.012	-1.43e+04	-1742.559
Area Population	8.5551	0.382	22.388	0.000	7.806	9.304

Omnibus:	0.318	**Durbin-Watson:**	1.997
Prob(Omnibus):	0.853	**Jarque-Bera (JB):**	0.368
Skew:	-0.002	**Prob(JB):**	0.832
Kurtosis:	2.952	**Cond. No.**	7.91e+04

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 7.91e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

The Durbin-Watson statistic is a measure used in statistical analysis to detect the presence of autocorrelation (also known as serial correlation) in the residuals of a regression analysis.Autocorrelation refers to the correlation between the error terms (residuals) of a regression model at different points in time.

- d = 2: No autocorrelation. The residuals are independent of each other, indicating that the regression model is well specified.
- d < 2: Positive autocorrelation. The residuals are positively correlated, indicating that there is a pattern of increasing values followed by increasing values or decreasing values followed by decreasing values.
- d > 2: Negative autocorrelation. The residuals are negatively correlated, indicating that there is a pattern of increasing values followed by decreasing values or decreasing values followed by increasing values.
- d near 0 or 4: Strong autocorrelation. The residuals are strongly correlated, indicating a clear pattern in the data.

Approach - 3 (Regularization Method)

Lasso Model

In [210]:
```
from sklearn.linear_model import Lasso
ls_model=Lasso(alpha=0.1)
ls_model.fit(x_train,y_train)
```

Out[210]:
```
Lasso(alpha=0.1)
```

In [211]:
```
ls_model.coef_
```

Out[211]:
```
array([2.17361817e+01, 1.65748364e+05, 1.22571580e+05, 1.52958076e+01])
```

In [212]:
```
y_test_pred_value=ls_model.predict(x_test)
y_test_pred_value
```

Out[212]:
```
array([1258927.49875592,  818030.38000865, 1745948.22950837, ...,
       1119387.12311172,  717217.63798877, 1516456.64076885])
```

In [213]:
```
y_test
```

Out[213]:
```
1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
            ...
1881    1.727211e+06
2800    1.707270e+06
1216    1.167450e+06
1648    7.241217e+05
3063    1.561234e+06
Name: Price, Length: 1250, dtype: float64
```

In [214]:
```
y_train_pred_value=ls_model.predict(x_train)
```

In [215]:
```
y_train_pred_value
```

Out[215]:
```
array([ 976450.4885673 , 1006891.2349294 , 1309554.07043195, ...,
        976149.04643434,  921184.60081324, 2134162.78028237])
```

In [216]:
```
y_train
```

Out[216]:
```
3430    9.958132e+05
3979    8.735092e+05
2153    1.360502e+06
4764    1.742351e+06
36      1.233220e+06
            ...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 3750, dtype: float64
```

In [217]:
```
print('Test Accuracy:',r2_score(y_test,y_test_pred_value))
print()
print('Train Accuracy:',r2_score(y_train,y_train_pred_value))
```

```
Test Accuracy: 0.9136094231024974

Train Accuracy: 0.9164810029817745
```

Ridge Model

In [218]:
```
from sklearn.linear_model import Ridge
rdg_model=Ridge(alpha=0.3)
```

In [219]:
```
rdg_model.fit(x_train,y_train)
```

Out[219]:
```
Ridge(alpha=0.3)
```

In [220]:
```
rdg_model.coef_
```

Out[220]:
```
array([2.17361606e+01, 1.65734410e+05, 1.22561619e+05, 1.52958135e+01])
```

In [221]:
```
y_test_pred_value=rdg_model.predict(x_test)
y_train_pred_value=rdg_model.predict(x_train)
```

In [222]:
```
print('Y_test_pred_value:',y_test_pred_value)
print()
print('Y_train_pred_value:',y_train_pred_value)
```

```
Y_test_pred_value: [1258922.99736832  818040.61547006 1745923.57622127 ... 1119380.18686373
  717248.13617621 1516455.85329504]

Y_train_pred_value: [ 976456.07457575 1006895.74356773 1309573.30278918 ...  976181.81182594
  921195.85427675 2134124.89049101]
```

In [223]:
```
print('Y_test:',y_test)
print()
print('Y_train:',y_train)
```

```
Y_test: 1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
           ...
1881    1.727211e+06
2800    1.707270e+06
1216    1.167450e+06
1648    7.241217e+05
3063    1.561234e+06
Name: Price, Length: 1250, dtype: float64

Y_train: 3430    9.958132e+05
3979    8.735092e+05
2153    1.360502e+06
4764    1.742351e+06
36      1.233220e+06
           ...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 3750, dtype: float64
```

In [224]:
```
print('Test Accuracy:',r2_score(y_test,y_test_pred_value))
print()
print('Train Accuracy:',r2_score(y_train,y_train_pred_value))
```

```
Test Accuracy: 0.9136091673700831

Train Accuracy: 0.9164810006924011
```

Elastic-Net Model

In [225]:
```
from sklearn.linear_model import ElasticNet
elastic_model = ElasticNet(alpha=0.3, l1_ratio=0.1)
elastic_model.fit(x_train, y_train)
```

Out[225]:
```
ElasticNet(alpha=0.3, l1_ratio=0.1)
```

In [226]:
```
y_test_pred_value=elastic_model.predict(x_test)
y_train_pred_value=elastic_model.predict(x_train)
```

In [227]:
```
print('Y_test_pred_value:',y_test_pred_value)
print()
print('Y_train_pred_value:',y_train_pred_value)
```

```
Y_test_pred_value: [1246818.06542494  844978.96583018 1680415.86721494 ... 1100819.62719308
  798142.16132046 1514593.20033514]

Y_train_pred_value: [ 991065.40410824 1018829.60296031 1360280.78414799 ... 1062826.89543301
  951214.99702309 2033609.05501852]
```

In [228]:
```
print('Y_test:',y_test)
print()
print('Y_train:',y_train)
```

```
Y_test: 1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
           ...
1881    1.727211e+06
2800    1.707270e+06
1216    1.167450e+06
1648    7.241217e+05
3063    1.561234e+06
Name: Price, Length: 1250, dtype: float64

Y_train: 3430    9.958132e+05
3979    8.735092e+05
2153    1.360502e+06
4764    1.742351e+06
36      1.233220e+06
           ...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 3750, dtype: float64
```

In [229]:
```
print('Test Accuracy:',r2_score(y_test,y_test_pred_value))
print()
print('Train Accuracy:',r2_score(y_train,y_train_pred_value))
```

```
Test Accuracy: 0.8957612813811744

Train Accuracy: 0.9006377625329316
```

Approach - 4 (Gradient Descent Method)

In [230]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(fs_x,Y,test_size=0.25,random_state=101)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
```

Out[230]:
```
((3750, 4), (1250, 4), (3750,), (1250,))
```

In [231]:
```
from sklearn.linear_model import SGDRegressor
gd_model=SGDRegressor()
```

In [232]:
```
gd_model.fit(x_train,y_train)
```

Out[232]:
```
SGDRegressor
SGDRegressor()
```

In [233]:
```
y_test_pred_value=gd_model.predict(x_test)
y_train_pred_value=gd_model.predict(x_train)
```

In [234]:
```
print('Y_test_pred_value:',y_test_pred_value)
print()
print('Y_train_pred_value:',y_train_pred_value)
```

```
Y_test_pred_value: [1257927.08419367  817175.10055143 1744701.34422096 ... 1117737.53320219
  717877.63142226 1516963.91333358]

Y_train_pred_value: [ 975930.56698922 1006568.19736192 1309770.54123983 ...  975872.81125148
  921498.42407054 2132820.19018438]
```

In [235]:
```
print('Y TEST:',y_test)
print()
print('Y TRAIN:',y_train)
```

```
Y TEST: 1718    1.251689e+06
2511    8.730483e+05
345     1.696978e+06
2521    1.063964e+06
54      9.487883e+05
           ...
1881    1.727211e+06
2800    1.707270e+06
1216    1.167450e+06
1648    7.241217e+05
3063    1.561234e+06
Name: Price, Length: 1250, dtype: float64

Y TRAIN: 3430    9.958132e+05
3979    8.735092e+05
2153    1.360502e+06
4764    1.742351e+06
36      1.233220e+06
           ...
4171    7.296417e+05
599     1.599479e+06
1361    1.102641e+06
1547    8.650995e+05
4959    2.108376e+06
Name: Price, Length: 3750, dtype: float64
```

In [236]:
```
print('TEST ACCURACY:',r2_score(y_test,y_test_pred_value))
print()
print('TRAIN ACCURACY:',r2_score(y_train,y_train_pred_value))
```

```
TEST ACCURACY: 0.9136214476424529

TRAIN ACCURACY: 0.9164717709711558
```

Performance Matrix

In [237]:
```
from sklearn import metrics
```

In [238]:
```
# Mean absolute error
print("MAE :", metrics.mean_absolute_error(y_test, y_test_pred_value))
```

```
MAE : 83051.16989510444
```

In [244]:
```
# Mean absolute percentage error
print("MAPE :", metrics.mean_absolute_percentage_error(y_test, y_test_pred_value)/100)
```

```
MAPE : 0.0008004482387862877
```

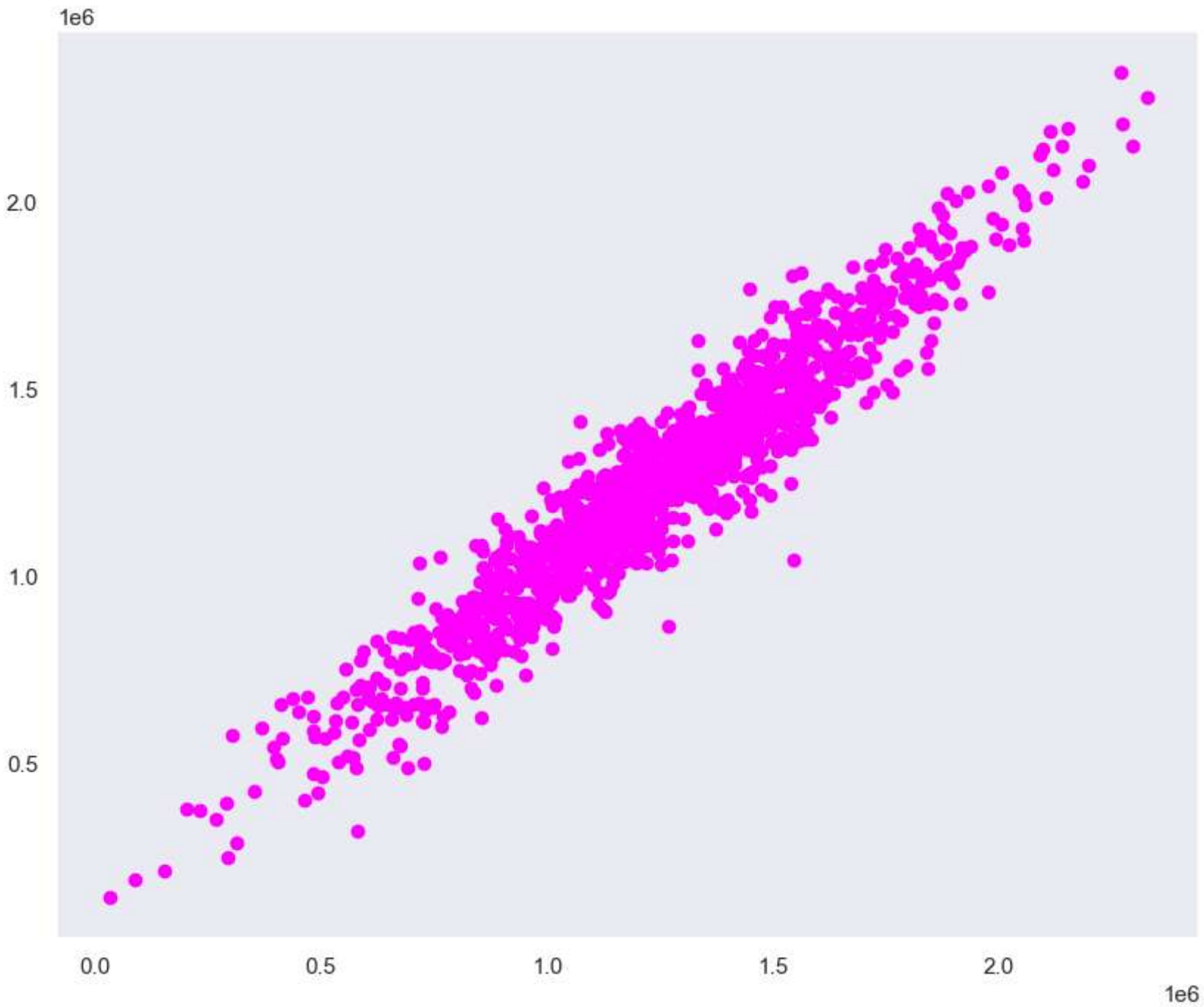
In [245]:
```
# Mean squared error
print("MSE :", metrics.mean_squared_error(y_test, y_test_pred_value))
```

```
MSE : 10794805951.669369
```

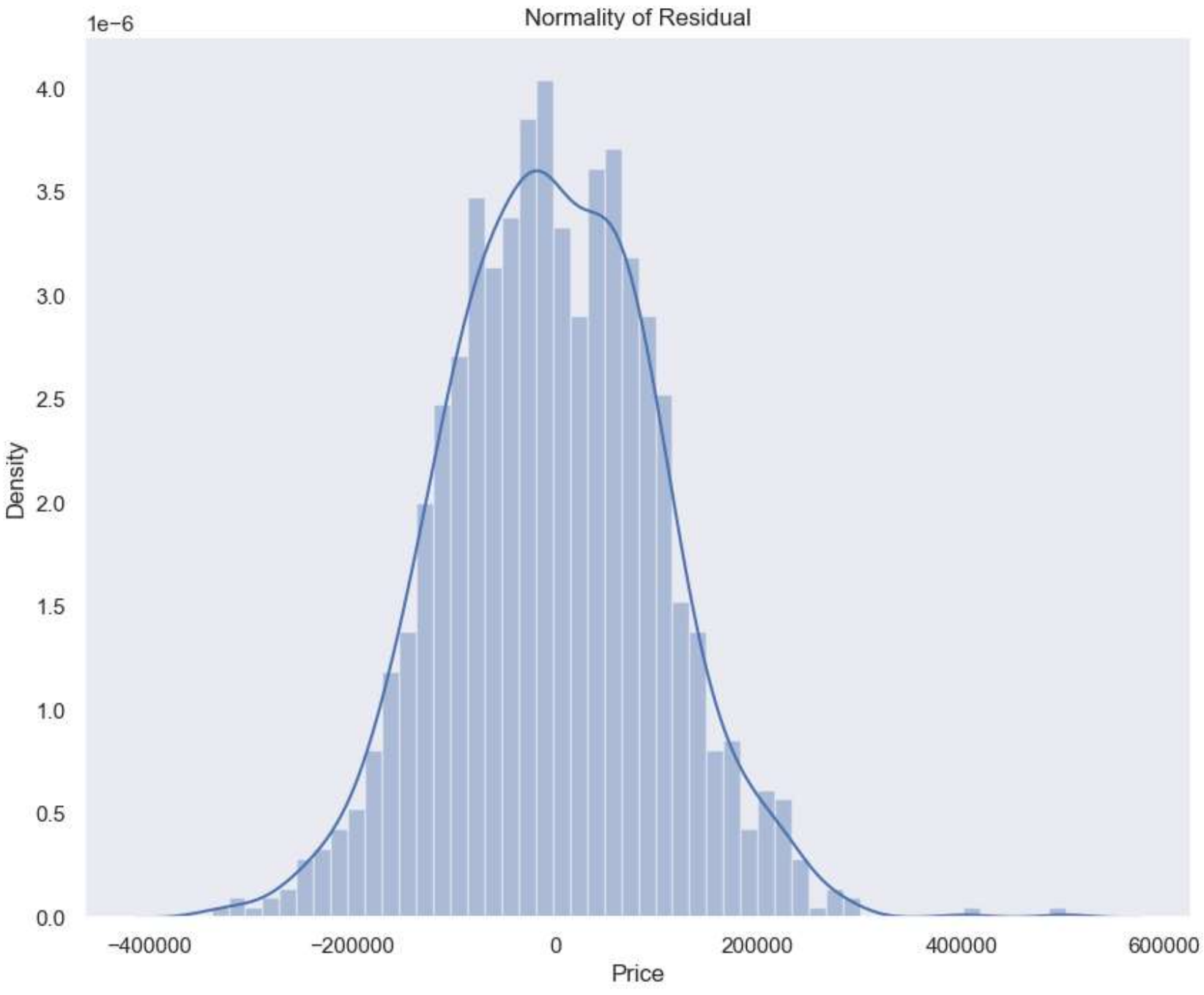
In [247]:
```
# Root mean squared error
print("RMSE :", np.sqrt(metrics.mean_squared_error(y_test, y_test_pred_value)))
```

```
RMSE : 103898.05557212979
```

In [248]:
```
# Checking Linearity
plt.figure(figsize=(10,8))
plt.scatter(y_test,y_test_pred_value,color='magenta')
plt.show()
```



In [249]:
```
# Normality of Residual/Errors
plt.figure(figsize=(10,8))
plt.title('Normality of Residual')
sns.distplot((y_test - y_test_pred_value),bins=50)
plt.show()
```



CONCLUSION

- The model we have built can be used for inference of how the different independent variables influence the outcome.
- We saw above that by using all the methods the train & test accuracy of our model is more then >90%.
- Only in the elastic-net model the train & test accuracy came 87% ~ also a good accuracy.
- Here our model is also satisfying each and every assumptions.
- Now We can say we have built a perfect Linear Regression Model.