House Price Prediction - Multiple Linear Regression Model

Objective of the Project

- The main objective is to build Linear Regression model to predict the house price with respect to the features. For this purpose I have taken USA Housing dataset which contains features like Avg. Area Income, Avg. Area House Age, Avg. Area Number of Rooms, Avg. Area Number of Bedrooms, Area Population and the target variable will be price.

What is Linear Regression?

- Linear regression is a type of supervised machine learning algorithm that computes the linear relationship between a dependent variable and one or more independent features.
- When the number of the independent feature, is 1 then it is known as Univariate Linear regression, and in the case of more than one feature, it is known as multivariate linear regression.
- The goal of the algorithm is to find the best linear equation that can predict the value of the dependent variable based on the independent variables.

Linear Regression formula - y= β0+ β 1x+ ε

- y= Dependent Variable
- x= Independent Variable
- β0= intercept of the line
- β1 = Linear regression coefficient (slope of the line)
- ε = random error

Assumptions for Linear Regression Model for a reliable and accurate model

- Linearity: The output must have a linear association with the input values, and it only suits data that has a linear relationship between the two entities.
- Homoscedasticity: The standard deviation and the variance of the residuals (difference of (y-yˆ)2) must be the same for any value of x. Multiple Linear Regression assumes that the amount of error in the residuals is similar at each point of the linear model. We can check the Homoscedasticity using Scatter plots.
- Non-multicollinearity: The independent variables should not be highly correlated with each other. We can check the data for this using a correlation matrix.
- No Autocorrelation: When data are obtained across time, we assume that successive values of the disturbance component are momentarily independent in the conventional Linear Regression model. When this assumption is not followed, the situation is referred to be autocorrelation.
- Not applicable to Outliers: The value of the dependent variable cannot be estimated for a value of an independent variable which lies outside the range of values in the sample data.
- No Endogeneity - Endogeneity refers to situations in which a predictor (e.g., treatment variable) in a linear regression model is correlated to the error term.

Steps followed

- Import required Libraries
- Load the dataset
- Check basic information of the dataset
- Data Preprocessing
- Split data into dependent and independent variables
- Feature Scaling
- Split data into Train and test
- Find the Correlation
- Check Variance Inflation factor
- Build Linear Regression Model
- Predict the data
- Evaluate the model
- Conclusion

Import the required libraries

In [1]:

```
import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
%matplotlib inline

import warnings
warnings.filterwarnings('ignore')
```

Load the Dataset - USA Housing

In [2]:

```
USAHousing=pd.read_csv(r"C:\Users\manme\Documents\Priya\Stas and ML\Dataset\USA_Housing.
USAHousing.head(2)
```

Out[2]:

	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	1059033.558	208 Michael Ferry Apt. 674\nLaurabury, NE 3701...
1	79248.64245	6.002900	6.730821	3.09	40173.07217	1505890.915	188 Johnson Views Suite 079\nLake Kathleen, CA...

Basic information about the dataset

In [3]:

```
USAHousing.info()  #Address is the object column rest all are numerical column
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 7 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   Avg. Area Number of Bedrooms  4994 non-null   float64
 4   Area Population              5000 non-null   float64
 5   Price                        5000 non-null   float64
 6   Address                      5000 non-null   object 
dtypes: float64(6), object(1)
memory usage: 273.6+ KB
```

In [4]:

```
USAHousing.describe()     #It provides statistical summary
```

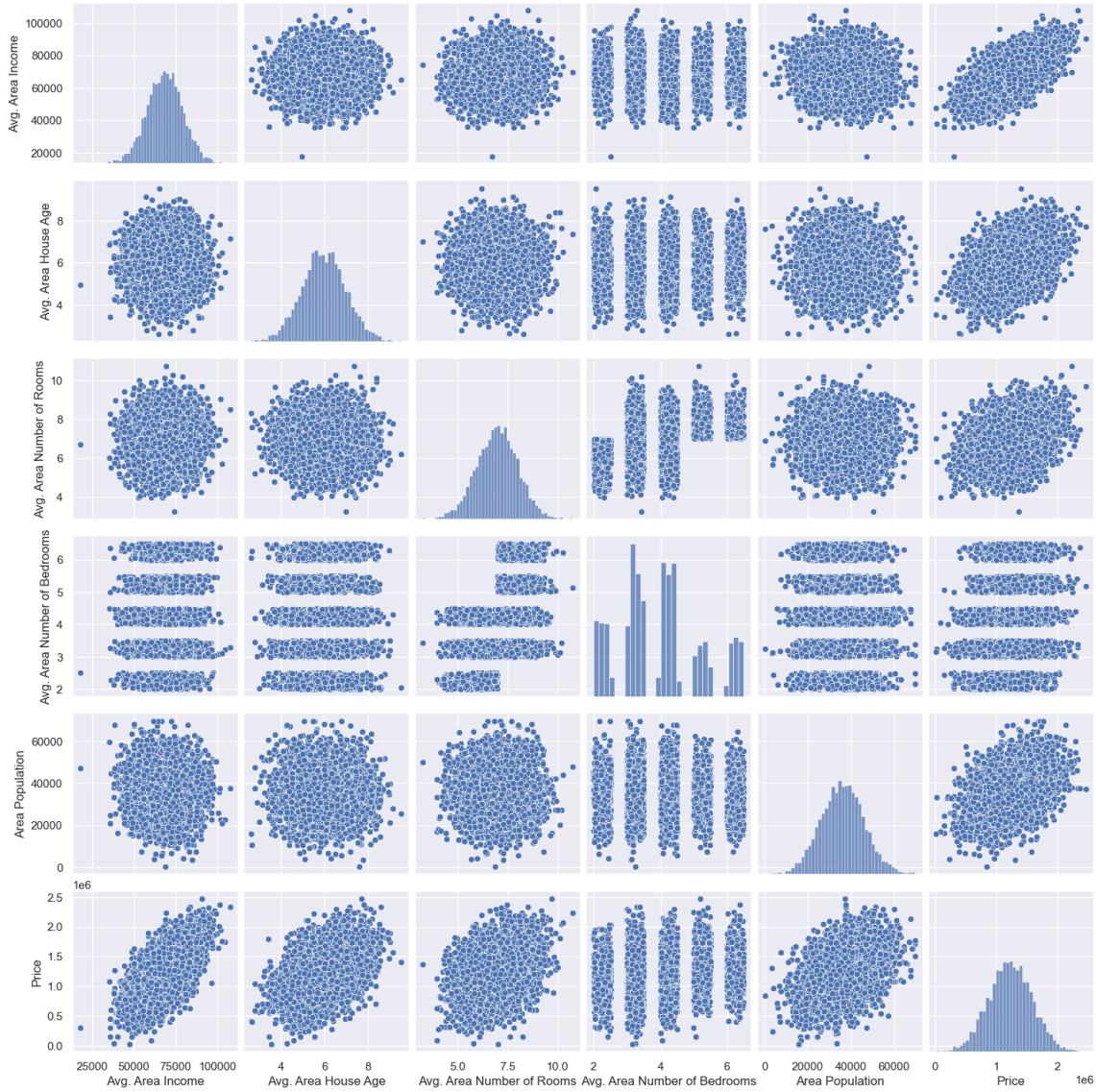
Out[4]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	4990.000000	5000.000000	4995.000000	4994.000000	5000.000000	5.000000e+03
mean	68584.719991	5.977222	6.987693	3.981874	36163.516039	1.232073e+06
std	10651.192423	0.991456	1.005938	1.234497	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61481.465105	5.322283	6.299156	3.140000	29403.928700	9.975771e+05
50%	68797.671885	5.970429	7.002940	4.050000	36199.406690	1.232669e+06
75%	75779.145465	6.650808	7.665622	4.490000	42861.290770	1.471210e+06
max	107701.748400	9.519088	10.759588	6.500000	69621.713380	2.469066e+06

Data Visualization

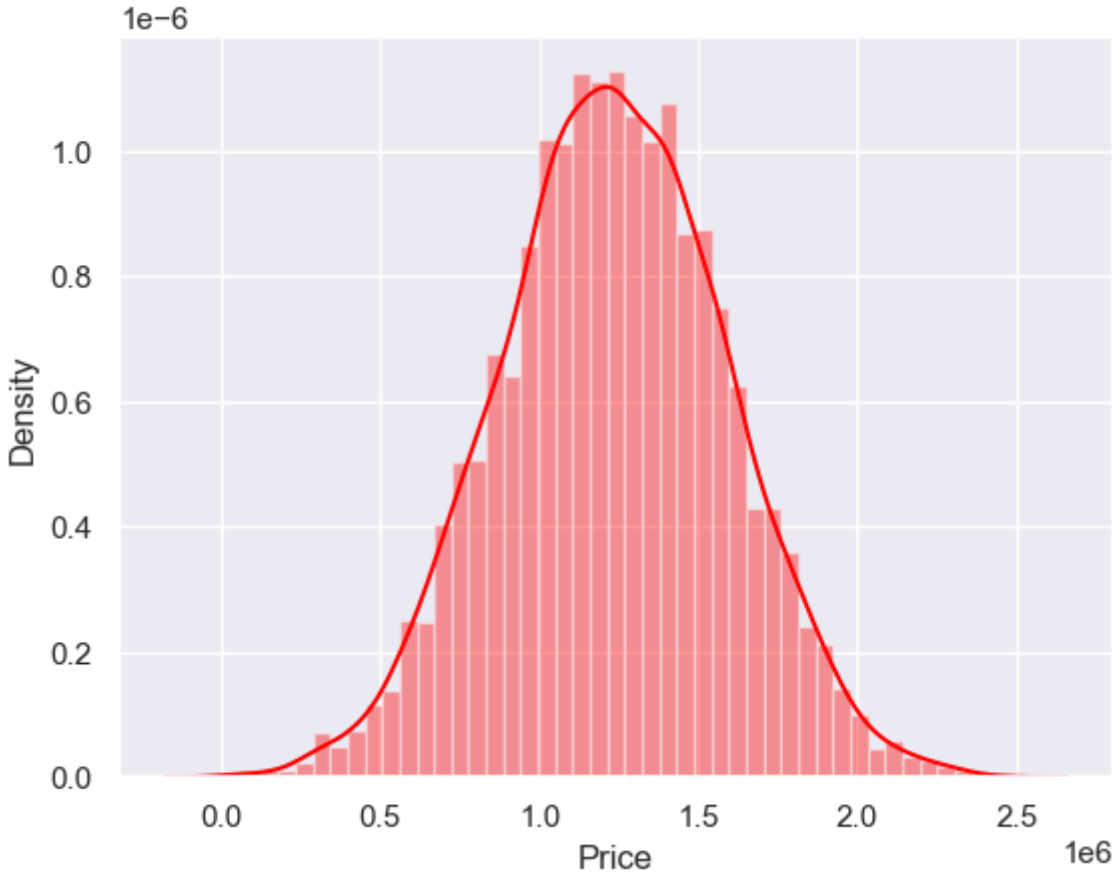
In [5]:

```
sns.pairplot(USAHousing)
plt.show()
```



In [6]:

```
sns.distplot(USAHousing['Price'],color='red')
plt.show()
```



Data Pre Processing

- It is an important step and involves cleaning and transforming raw data to make it suitable for analysis.
- Checking and removing duplicate values.
- Check for missing values and treat them
- Check for outliers and treat them

Checking duplicate values

In [7]:

```
USAHousing.duplicated().sum()
```

Out[7]:

```
0
```

Drop variable

- Avg. Area Number of Bedrooms as Avg. Area Number of Rooms is also given

In [8]:

```
USAHousing=USAHousing.drop(['Avg. Area Number of Bedrooms'],axis=1)
```

Check missing values

In [9]:

```
USAHousing.isnull().sum()
```

Out[9]:

```
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 [10]:

```
USAHousing.isnull().sum()/len(USAHousing)*100
```

Out[10]:

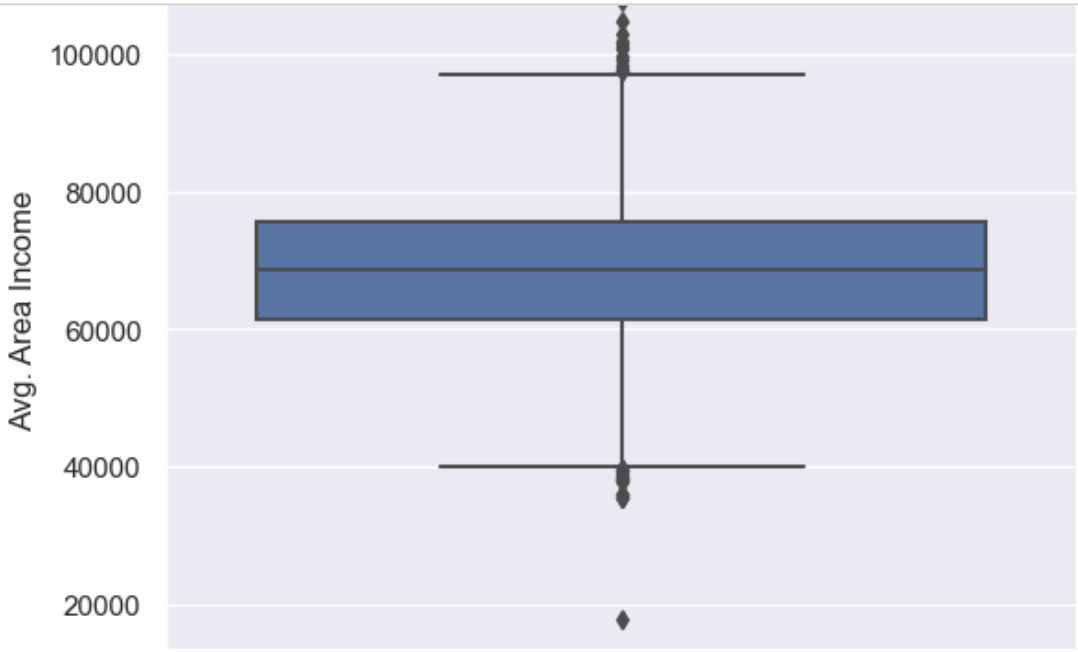
```
Avg. Area Income             0.2
Avg. Area House Age          0.0
Avg. Area Number of Rooms    0.1
Area Population              0.0
Price                        0.0
Address                      0.0
dtype: float64
```

***Note - Imputation will be done as missing values are less than 25%**

***Check for Outliers and then decide whether we have to use mean or median approach**

In [11]:

```
sns.boxplot(y='Avg. Area Income',data=USAHousing)
plt.show()
```

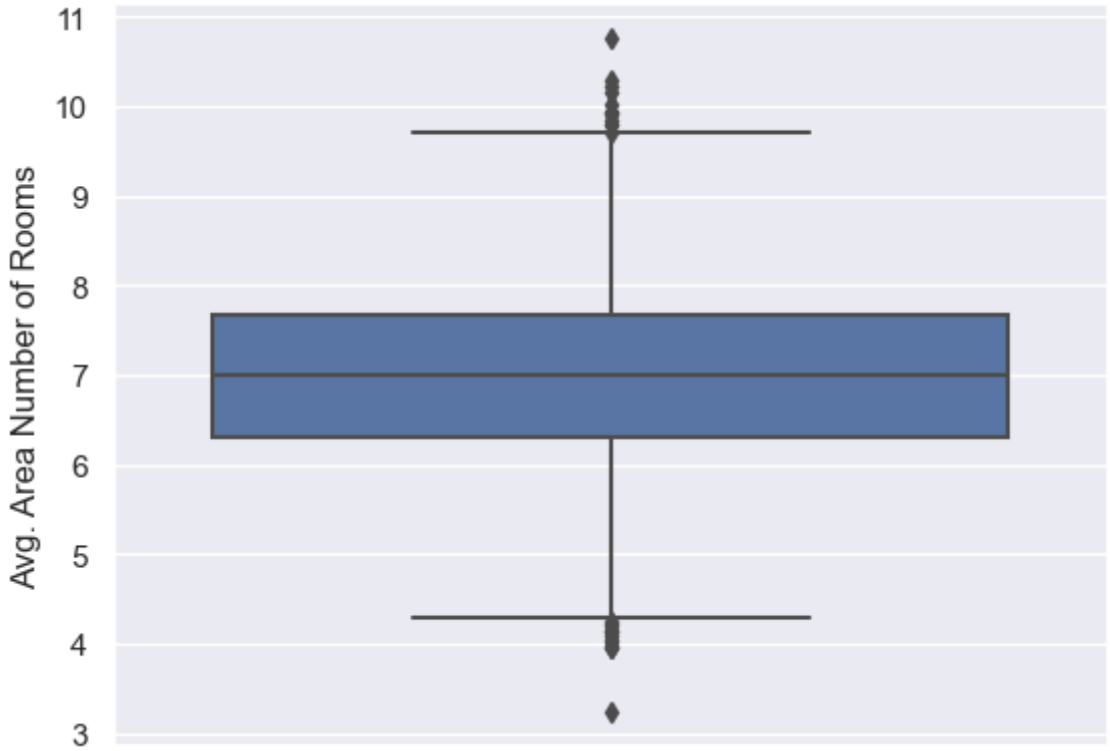


In [12]:

```
USAHousing['Avg. Area Income']=USAHousing['Avg. Area Income'].fillna(USAHousing['Avg. Ar
```

In [13]:

```
sns.boxplot(y='Avg. Area Number of Rooms',data=USAHousing)
plt.show()
```



In [14]:

```
ing['Avg. Area Number of Rooms']=USAHousing['Avg. Area Number of Rooms'].fillna(USAHousir
```

In [15]:

```
USAHousing.isnull().sum()   # All missing values have been treated
```

Out[15]:

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

Label Encoder

- Address variable

In [16]:

```
USAHousing['Address']=USAHousing['Address'].astype('category')
USAHousing['Address']=USAHousing['Address'].cat.codes
```

ANOVA Testing

- ANOVA abbreviates for Analysis of Variance. It is one of the statistical tests that opted to study the statistical differences between both numerical and categorical sets of features of the data. It generally tries to decode the correlation among the various features of the data.

In [17]:

```
import statsmodels.api as sm
from statsmodels.formula.api import ols
model=ols('Price ~ Address',data=USAHousing).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
```

Dropping Address variables (non-significant)

- As p value is greater than 0.05 (ANOVA testing)

In [18]:

```
USAHousing=USAHousing.iloc[:,0:-1]
```

In [19]:

```
USAHousing.head()
```

Out[19]:

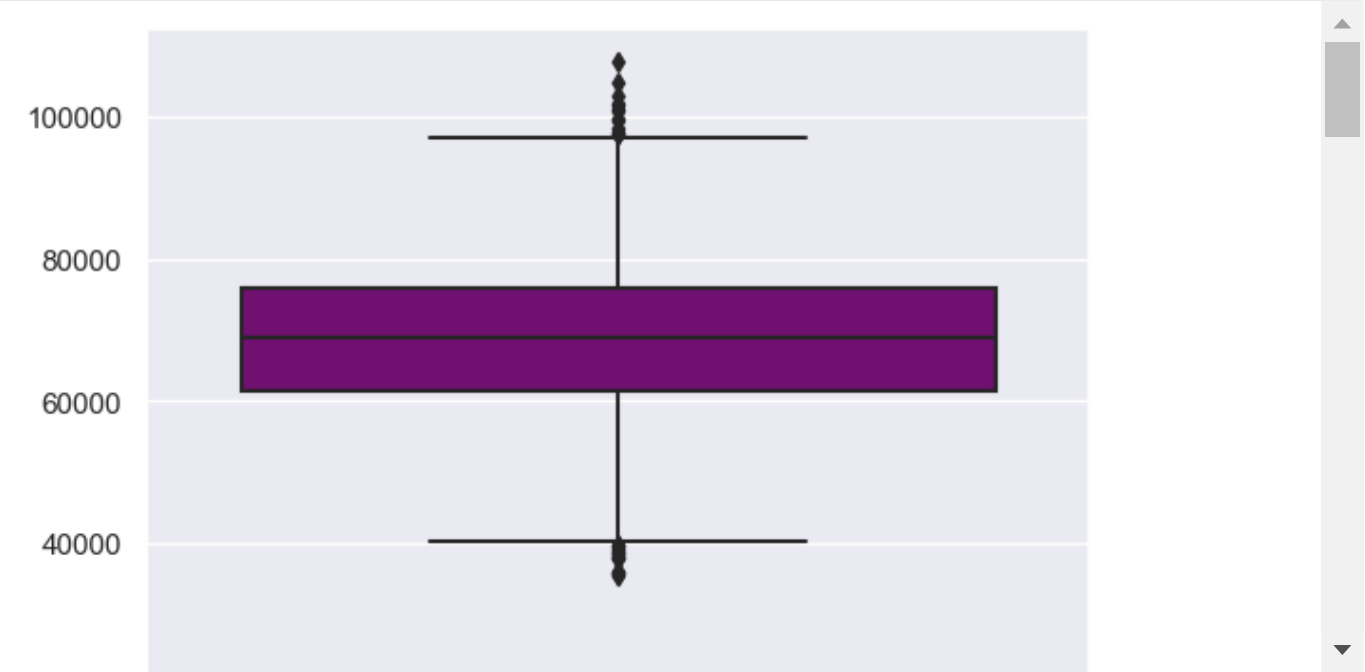
	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
0	79545.45857	5.682861	7.009188	23086.80050	1.059034e+06
1	79248.64245	6.002900	6.730821	40173.07217	1.505891e+06
2	61287.06718	5.865890	8.512727	36882.15940	1.058988e+06
3	63345.24005	7.188236	5.586729	34310.24283	1.260617e+06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

Checking for outliers

- Outliers can be higher or lower than expected, or displaced more to the right or left than expected. Outliers can effect regression lines, making the regression lines less accurate in predicting other data.

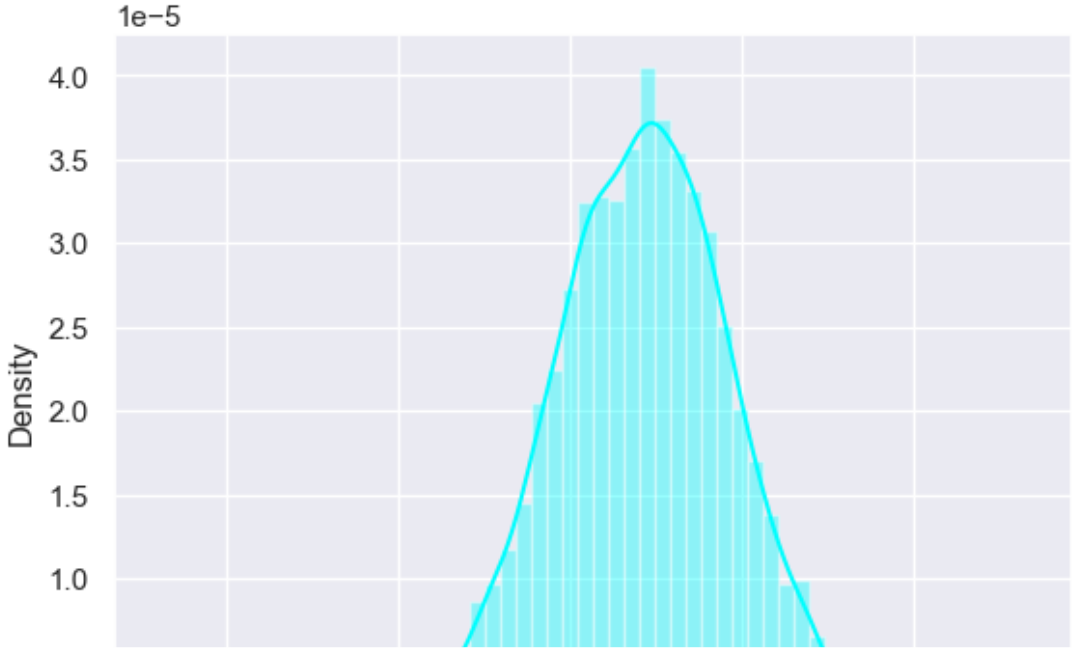
In [20]:

```
def boxplots(col):
    sns.boxplot(USAHousing[col],color='purple')
    plt.show()

for i in list(USAHousing.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```



In [21]:

```
def distplots(col):
    sns.distplot(USAHousing[col],color='aqua')
    plt.show()

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



Handle Outliers

- Capping method

In [22]:

```
USAHousing.columns
```

Out[22]:

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Roo
ms',
       'Area Population', 'Price'],
      dtype='object')
```

In [23]:

```
#Avg. Area Income
income_q1=USAHousing['Avg. Area Income'].quantile(0.25)
income_q3=USAHousing['Avg. Area Income'].quantile(0.75)
income_iqr=income_q3-income_q1
income_upper= income_q3 +1.5*income_iqr
income_lower= income_q1- 1.5 *income_iqr
```

In [24]:

```
USAHousing['Avg. Area Income']=np.where(USAHousing['Avg. Area Income']>income_upper, inc
                                        np.where(USAHousing['Avg. Area Income']<income_lower,
                                                 USAHousing['Avg. Area Income']))
```

In [25]:

```
#Avg. Area House Age
age_q1=USAHousing['Avg. Area House Age'].quantile(0.25)
age_q3=USAHousing['Avg. Area House Age'].quantile(0.75)
age_iqr=age_q3-age_q1
age_upper= age_q3 +1.5*age_iqr
age_lower= age_q1- 1.5 *age_iqr
```

In [26]:

```
USAHousing['Avg. Area House Age']=np.where(USAHousing['Avg. Area House Age']>age_upper,
                                        np.where(USAHousing['Avg. Area House Age']<age_lower,
                                                 USAHousing['Avg. Area House Age']))
```

In [27]:

```
#Avg. Area Number of Rooms
room_q1=USAHousing['Avg. Area Number of Rooms'].quantile(0.25)
room_q3=USAHousing['Avg. Area Number of Rooms'].quantile(0.75)
room_iqr=room_q3-room_q1
room_upper= room_q3 +1.5*room_iqr
room_lower= room_q1- 1.5 *room_iqr
```

In [28]:

```
USAHousing['Avg. Area Number of Rooms']=np.where(USAHousing['Avg. Area Number of Rooms']
                                        np.where(USAHousing['Avg. Area Number of Rooms']<room
                                                 USAHousing['Avg. Area Number of Rooms']))
```

In [29]:

```
#Area Population
pop_q1=USAHousing['Area Population'].quantile(0.25)
pop_q3=USAHousing['Area Population'].quantile(0.75)
pop_iqr=pop_q3-pop_q1
pop_upper= pop_q3 +1.5*pop_iqr
pop_lower= pop_q1- 1.5 *pop_iqr
```

In [30]:

```
USAHousing['Area Population']=np.where(USAHousing['Area Population']>pop_upper, pop_uppe
                                        np.where(USAHousing['Area Population']<pop_lower,pop_
                                                 USAHousing['Area Population']))
```

In [31]:

```
def boxplots(col):
    sns.boxplot(USAHousing[col],color='purple')
    plt.show()

for i in list(USAHousing.select_dtypes(exclude=['object']).columns)[0:]:
    boxplots(i)
```

Split the data into independent variable and dependent variable

In [32]:

```
USAHousing.head(2)
```

Out[32]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Area Population	Price
0	79545.45857	5.682861	7.009188	23086.80050	1059033.558
1	79248.64245	6.002900	6.730821	40173.07217	1505890.915

In [33]:

```
x=USAHousing.iloc[:,0:-1]
y=USAHousing['Price']
```

In [34]:

```
x.head()
```

Out[34]:

	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 [35]:

```
y.head()
```

Out[35]:

```
0    1.059034e+06
1    1.505891e+06
2    1.058988e+06
3    1.260617e+06
4    6.309435e+05
Name: Price, dtype: float64
```

Feature Scaling

- Feature scaling is a method used to normalize the range of independent variables or features of data.
- We can only do with independent variable not with dependent variable.

In [36]:

```
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
sc_x=sc.fit_transform(x)
pd.DataFrame(sc_x)
```

Out[36]:

	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

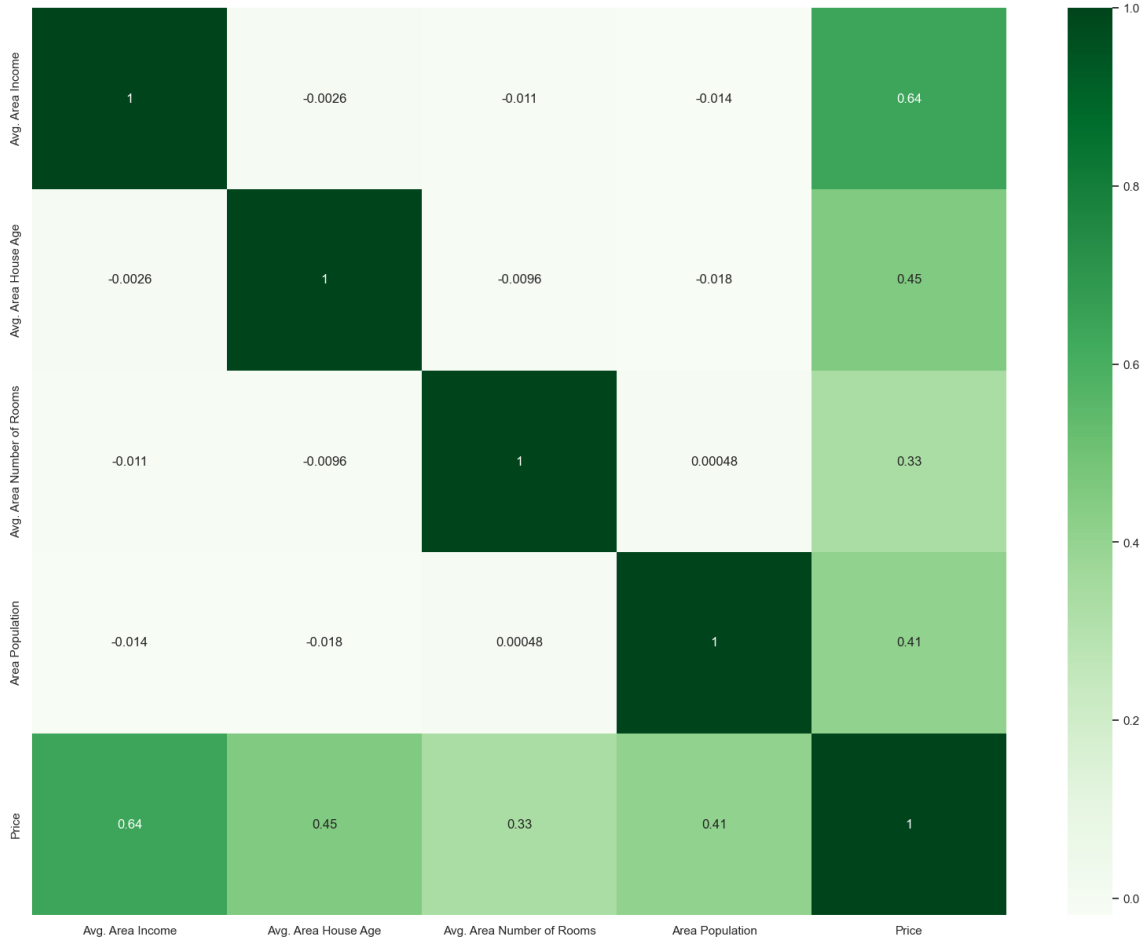
Check Multicollinearity

1. Find the correlation

- Heatmaps are a great way to show multicollinearity as it provides better visualization. The colors get darker as collinearity increases.

In [37]:

```
plt.figure(figsize=(20,15))
corr=USAHousing.corr()
sns.heatmap(corr, annot=True, cmap='Greens')
plt.show()
```



2. Variance Inflation factor (VIF) -

- A variance inflation factor (VIF) provides a measure of multicollinearity among the independent variables in a multiple regression model.
- A large VIF on an independent variable indicates a highly collinear relationship to the other variables that should be considered or adjusted for in the structure of the model and selection of independent variables.
- Usually, VIF equal to 1 = variables are not correlated, VIF between 1 and 5 = variables are moderately correlated ,VIF greater than 5 = variables are highly correlated

In [38]:

```
variable=sc_x
variable.shape
```

Out[38]:

```
(5000, 4)
```

In [39]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
variable=sc_x
vif=pd.DataFrame()
vif['Variance Inflation Factor'] =[variance_inflation_factor(variable,i) for i in range(
vif['Features']=x.columns
```

In [40]:

```
vif
```

Out[40]:

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

***No multi collinearity found as all values are less than 5**

Split the data into training and test

In [41]:

```
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)

print(x_train.shape,x_test.shape,y_train.shape,y_test.shape)
```

```
(3750, 4) (1250, 4) (3750,) (1250,)
```

Building Linear Regression Model

Approach - 1

- Linear Regression Method

In [42]:

```
from sklearn.linear_model import LinearRegression
lm=LinearRegression()
lm.fit(x_train,y_train)
```

Out[42]:

```
LinearRegression
LinearRegression()
```

In [43]:

```
print(lm.intercept_)
print()
print(lm.coef_)
```

```
-2658303.392038106

[2.17361819e+01 1.65748470e+05 1.22571682e+05 1.52958075e+01]
```

In [44]:

```
x.columns
```

Out[44]:

```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Roo
ms',
       'Area Population'],
      dtype='object')
```

```
price=intercept+slope1*Avg. Area Income+slope2*Avg. Area House Age+ slope3*Avg. Area
Number of Rooms + slope4**Area Population
```

In [45]:

```
price=-2658303.392038106+2.17361819e+01*85000+1.65748470e+05*2+1.22571682e+05*3+1.529580
price
```

Out[45]:

```
469724.7404618938
```

Predict house price by using LR model with test dataset

In [46]:

```
y_pred_price =lm.predict(x_test)
y_pred_price_train=lm.predict(x_train)
```

In [47]:

```
y_pred_price
```

Out[47]:

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

In [48]:

```
y_pred_price_train
```

Out[48]:

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

In [49]:

```
y_test
```

Out[49]:

```
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
```

Validate the actual price of test data and predicted price

- By Sq. Loss function

In [50]:

```
from sklearn.metrics import r2_score
r2_score (y_test,y_pred_price)
```

Out[50]:

```
0.913609424096595
```

In [51]:

```
r2_score (y_train,y_pred_price_train)
```

Out[51]:

```
0.9164810029819419
```

Approach No 2

- Ordinary Least Squares (OLS) Method
- The method relies on minimizing the sum of squared residuals between the actual and predicted values. It can be used to find the best-fit line for data by minimizing the sum of squared errors or residuals between the actual and predicted values.

In [52]:

```
from statsmodels.regression.linear_model import OLS
import statsmodels.regression.linear_model as smf
```

In [53]:

```
reg_model=smf.OLS(endog=y_train,exog=x_train).fit()
```

In [54]:

```
reg_model.summary()
```

Out[54]:

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, 21 Jul 2023	**Prob (F-statistic):**	0.00
Time:	16:58:10	**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.

Check for Auto correlation by Durbin Watson

- The Durbin Watson (DW) statistic is a test for autocorrelation in the residuals from a statistical model or regression analysis.
- It will always have a value ranging between 0 and 4.
- A value of 2.0 indicates there is no autocorrelation detected in the sample,values from 0 to less than 2 point to positive autocorrelation and values from 2 to 4 means negative autocorrelation.

Approach - 3

- Regularization in Linear Regression
- It is a technique in machine learning that tries to achieve the generalization of the model. It means that our model works well not only with training or test data, but also with the data it'll receive in the future. It avoids overfitting and reduces the variance of the model.

a. Lasso regression (L1 regularization)

- It is a technique that increases the cost function by a penalty equal to the sum of the absolute values of the non-intercept weights from linear regression.
- In this coefficient/slope will be exactly zero or closer to zero.

In [55]:

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(x_train, y_train)
print("Lasso Model :", (lasso.coef_))
```

```
Lasso Model : [2.17361817e+01 1.65748364e+05 1.22571580e+05 1.52958076e+0
1]
```

In [56]:

```
y_pred_train_lasso = lasso.predict(x_train)
y_pred_test_lasso = lasso.predict(x_test)
```

In [57]:

```
print('Training Accuracy :',r2_score(y_train, y_pred_train_lasso))
print()
print('Test Accuracy :',r2_score(y_test, y_pred_test_lasso))
```

```
Training Accuracy : 0.9164810029817745

Test Accuracy : 0.9136094231024974
```

b. Ridge Regression (L2 regularization)

- It is a regularization technique, which is used to reduce the complexity of the model.
- In this technique, the cost function is altered by adding the penalty term to it.
- In this coefficient/slope will be closer to zero but it will not make zero.

In [58]:

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=0.3)
ridge.fit(x_train, y_train)
print("Ridge Model :", (ridge.coef_))
```

```
Ridge Model : [2.17361606e+01 1.65734410e+05 1.22561619e+05 1.52958135e+0
1]
```

In [59]:

```
y_pred_train_ridge = ridge.predict(x_train)
y_pred_test_ridge = ridge.predict(x_test)
```

In [60]:

```
print('Training Accuracy :',r2_score(y_train, y_pred_train_ridge))
print()
print('Test Accuracy :', r2_score(y_test, y_pred_test_ridge))
```

```
Training Accuracy : 0.916481000692401

Test Accuracy : 0.9136091673700831
```

c. Elastic Net Regression

- It is a combination of Lasso & Ridge Regression

In [61]:

```
from sklearn.linear_model import ElasticNet
elastic = ElasticNet(alpha=0.3, l1_ratio=0.1)
elastic.fit(x_train, y_train)
```

Out[61]:

```
ElasticNet(alpha=0.3, l1_ratio=0.1)
```

In [62]:

```
y_pred_train_elastic = elastic.predict(x_train)
y_pred_test_elastic = elastic.predict(x_test)
```

In [63]:

```
print('Training Accuracy :', r2_score(y_train, y_pred_train_elastic))
print()
print('Test Accuracy :', r2_score(y_test, y_pred_test_elastic))
```

```
Training Accuracy : 0.9006377625329316

Test Accuracy : 0.8957612813811743
```

Approach - 4

- Gradient Descent
- It is an optimization algorithms to train machine learning models by means of minimizing errors between actual and expected results.

In [64]:

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(sc_x, y, test_size=0.25, random_stat

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

```
(3750, 4) (1250, 4) (3750,) (1250,)
```

In [65]:

```
from sklearn.linear_model import SGDRegressor
```

In [66]:

```
gd_model = SGDRegressor()
gd_model.fit(x_train, y_train)
```

Out[66]:

SGDRegressor

SGDRegressor()

In [67]:

```
y_pred_gd_train = gd_model.predict(x_train)
y_pred_gd_test = gd_model.predict(x_test)
```

In [68]:

```
print('GD Training Accuracy :', r2_score(y_train, y_pred_gd_train))
print()
print('GD Test Accuracy :', r2_score(y_test, y_pred_gd_test))
```

```
GD Training Accuracy : 0.916442528777437

GD Test Accuracy : 0.9135374972585752
```

Performance Matrix

- It is a measure of how good a model performs and how well it approximates the relationship.

a. Mean Absolute Error (MAE)

- This is simply the average of the absolute difference between the target value and the value predicted

In [69]:

```
from sklearn import metrics
```

In [70]:

```
print('MAE:', metrics.mean_absolute_error(y_test,y_pred_price))
```

```
MAE: 83069.03172980985
```

b. Mean Absolute Percent Error (MAPE)

In [71]:

```
print('MAPE:', metrics.mean_absolute_error(y_test,y_pred_price)/100)
```

```
MAPE: 830.6903172980985
```

c. Mean Squared Error (MSE)

- It is the average of the squared difference between the predicted and actual value.

In [72]:

```
print('MSE:', metrics.mean_squared_error(y_test,y_pred_price))
```

```
MSE: 10796308545.089209
```

Root Mean Squared Error (RMSE)

In [73]:

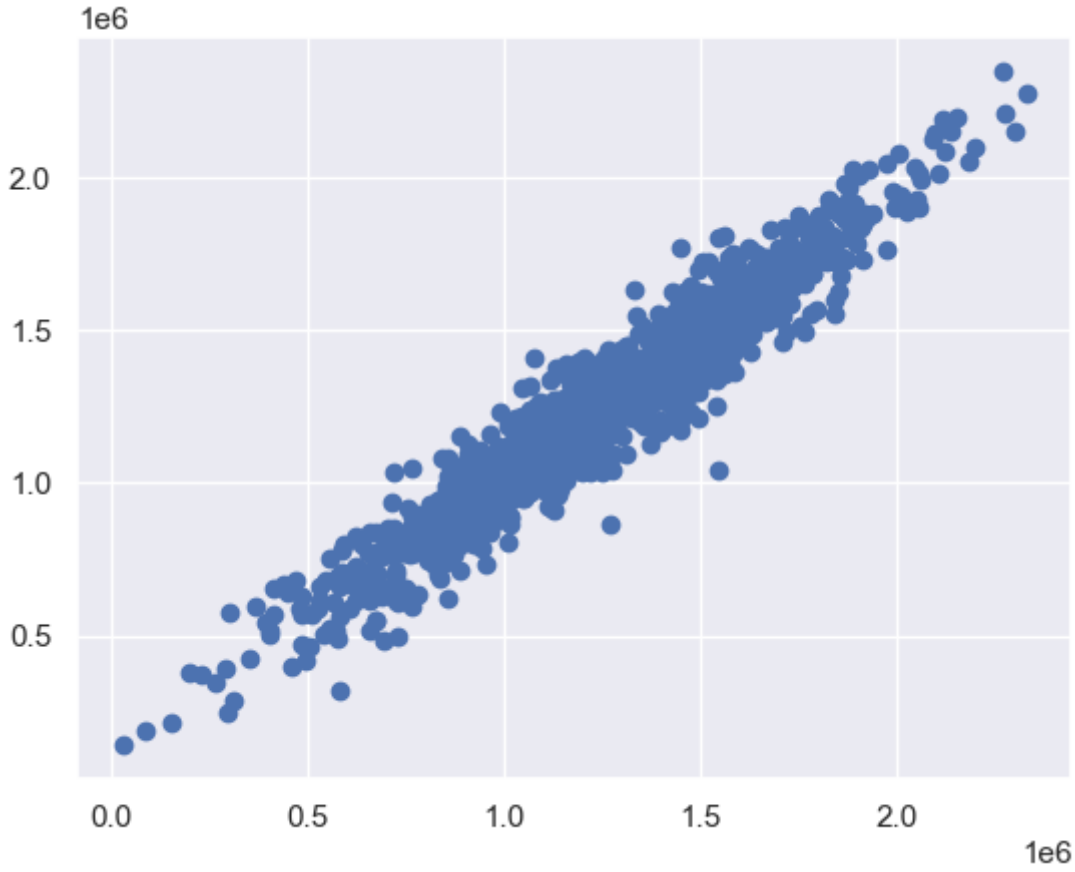
```
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,y_pred_price)))
```

```
RMSE: 103905.2864155102
```

Check Linearity

In [74]:

```
plt.scatter(y_test,y_pred_price)
plt.show()
```

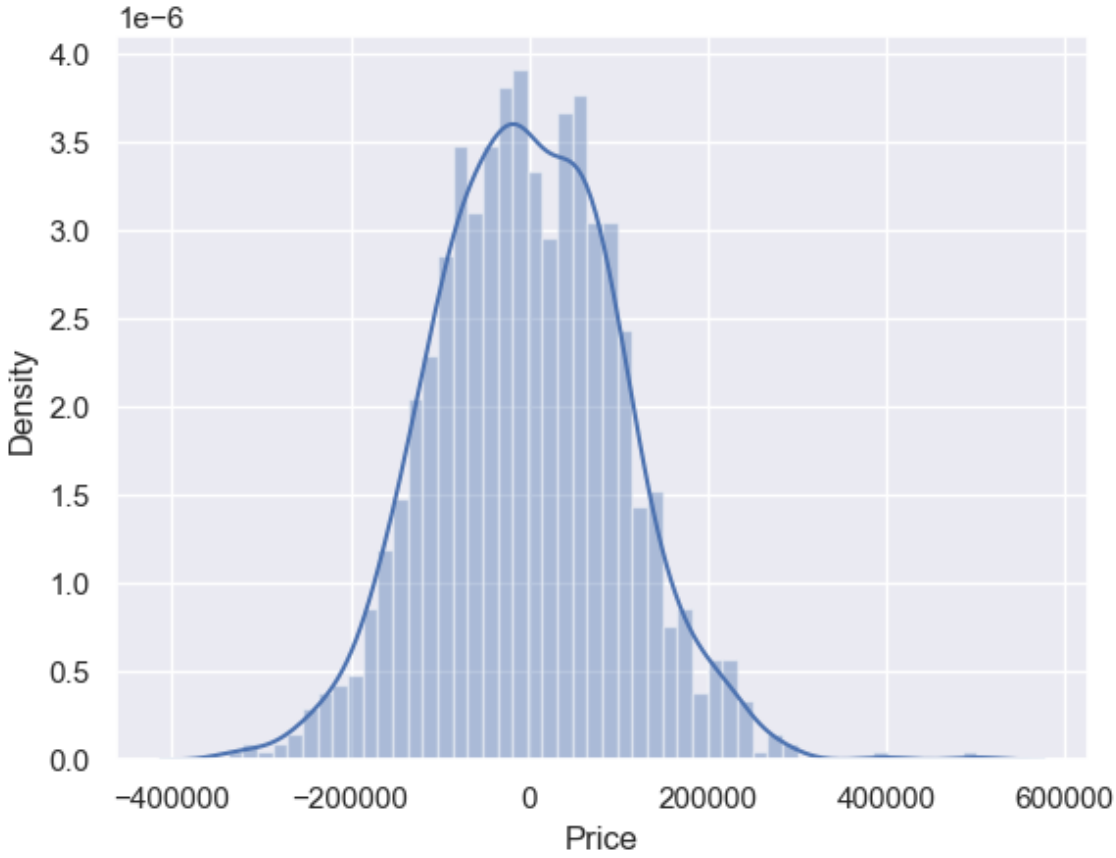


Normality of Residual

- Data is normally distributed or not?

In [75]:

```
sns.distplot((y_test-y_pred_price),bins=50)
plt.show()
```



Conclusion

- Adjested R squared (uncentred): 0.964
- All variables are statistically significant (p<0.05)
- Check Underfitting & Underfitting problem - No bias and variance found
- All the assumptions are satisfied.
- 1. Linearity - satisfied
- 2. Normality of Residual- Satisfied
- 3. Homoscedacidity - Satisfied(No outlier and the residual is normally distributed)
- 4. No autocorrelation - satisfied
- 5. No or little multi collinearity - satisfied
- 6. No Endogenity problem - Satisfied

In [ ]:

In [ ]: