import the library ¶

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')
```

import dataset

In [2]:
```
USAHousing = pd.read_csv('USA_Housing.csv')
USAHousing.head()
```

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	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

To find the information about the dataset

In [3]:
```
USAHousing.info()
```

```
<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()
```

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

Dropping the "Avg. Area Number of Bedrooms" as it is not significant because we already have "Avg. Area Number of Rooms"

In [5]:
```
USAHousing=USAHousing.drop(["Avg. Area Number of Bedrooms"],axis=1)
```

Data Preprocessing

Missing value treatement

In [6]:
```
USAHousing.isnull().sum()
```

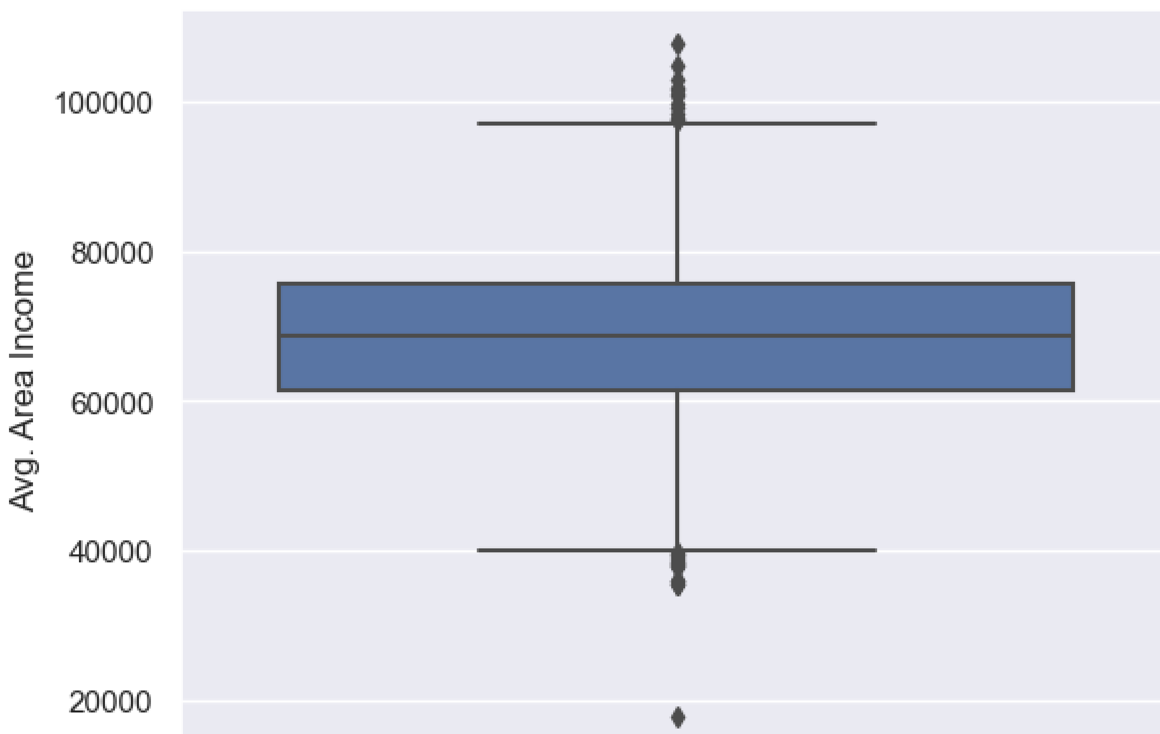
Out[6]:
```
Avg. Area Income             10
Avg. Area House Age           0
Avg. Area Number of Rooms     5
Area Population               0
Price                         0
Address                       0
dtype: int64
```

In [7]:
```
USAHousing.isnull().sum()/len(USAHousing)*100
```

Out[7]:
```
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
```

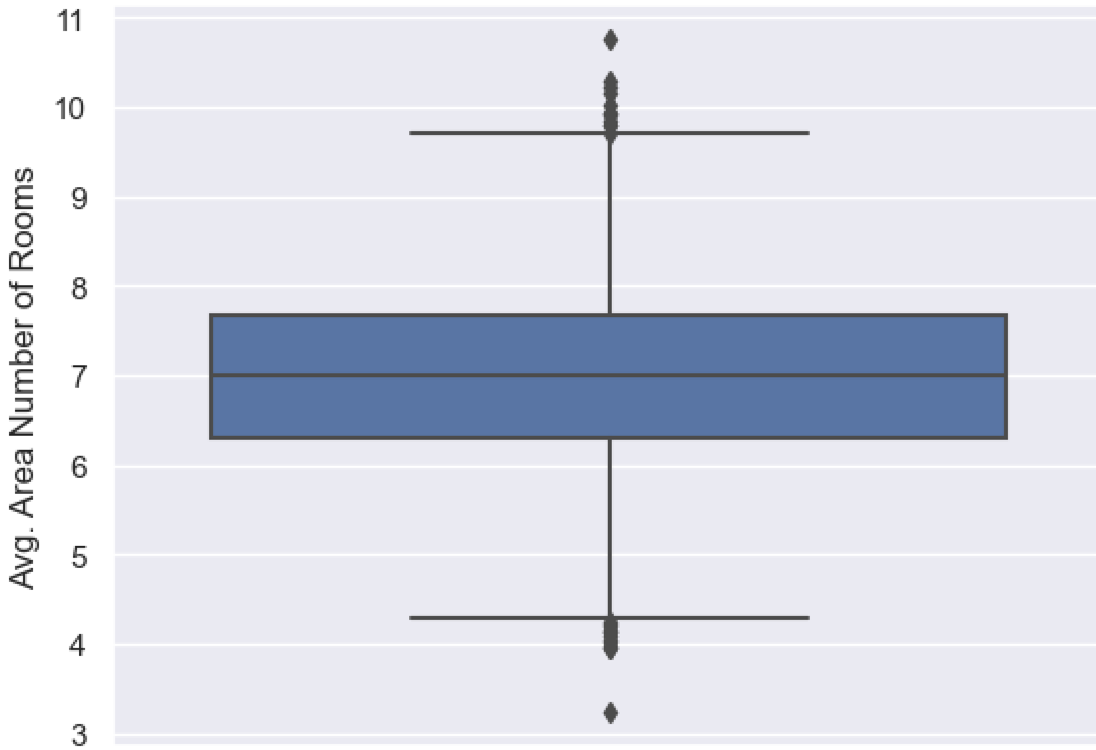
Check outlier and then will decide whether we have to use mean or median approach

In [8]:
```
sns.boxplot(y = 'Avg. Area Income', data=USAHousing)
plt.show()
```



In [9]:
```
USAHousing['Avg. Area Income'] = USAHousing['Avg. Area Income'].fillna(USAHousing
```

In [10]:
```
sns.boxplot(y = 'Avg. Area Number of Rooms', data=USAHousing)
plt.show()
```



In [11]:
```
USAHousing['Avg. Area Number of Rooms'] = USAHousing['Avg. Area Number of Rooms']
```

In [12]:
```
USAHousing.isnull().sum()
```

Out[12]:
```
Avg. Area Income            0
Avg. Area House Age         0
Avg. Area Number of Rooms   0
Area Population             0
Price                       0
Address                     0
dtype: int64
```

Encoding concept

In [13]:
```
USAHousing.head(2)
```

Out[13]:

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

In [14]:
```
USAHousing['Address'] = USAHousing['Address'].astype('category')
USAHousing['Address'] = USAHousing['Address'].cat.codes
```

Before proceding further e need to chek if "Address" column is significant or not through ANOVA Testing

ANOVA Testing - two way or multiple way anova

In [15]:
```
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, typ=2)
print(anova_result)
```

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

As we can conclude P value is greator than 0.5, the column is non-significant we can rop it.

In [ ]:

Address is non-significant variable to predict USA Housing price. hence, we have to drop this variable

In [16]:
```
USAHousing = USAHousing.iloc[:,0:-1]
```

In [17]:
```
USAHousing.head()
```

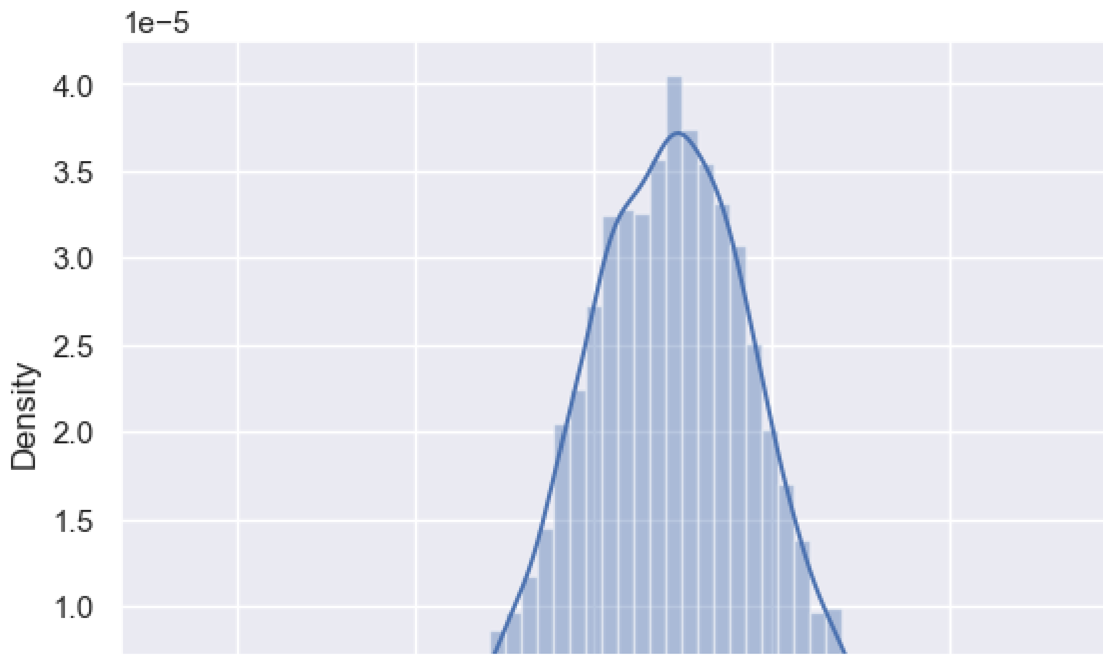
Out[17]:

	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

Handling outlier

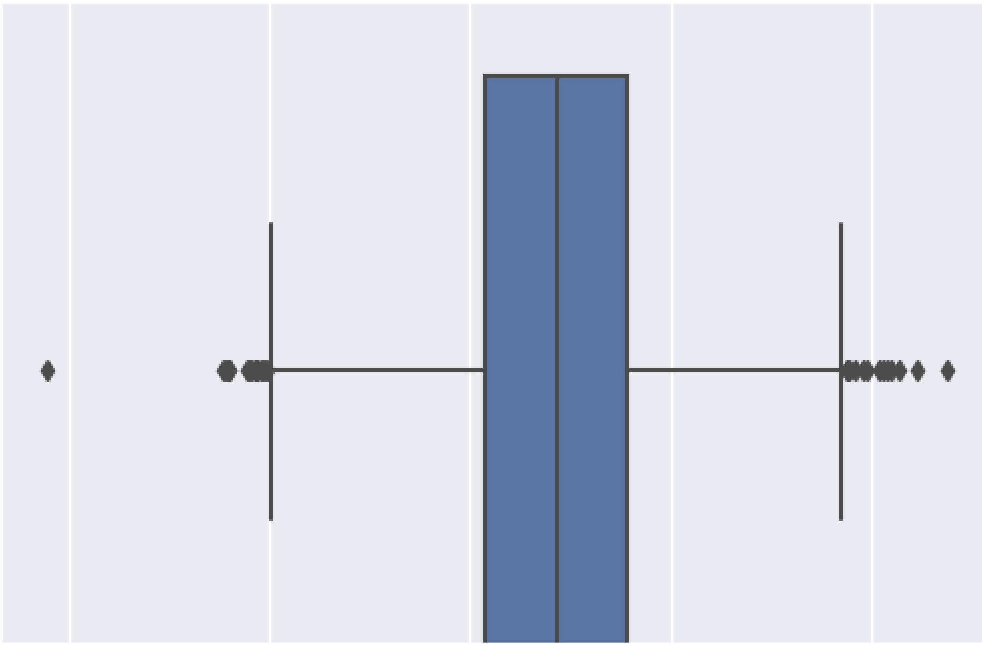
In [18]:
```
def distplots(col):
    sns.distplot(USAHousing[col])
    plt.show()

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



In [19]:
```
def boxplots(col):
    sns.boxplot(USAHousing[col])
    plt.show()

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



In [20]:
```
USAHousing.columns
```

Out[20]:
```
Index(['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
       'Area Population', 'Price'],
      dtype='object')
```

In [21]:
```
Q1 = USAHousing.quantile(0.25)
Q3 = USAHousing.quantile(0.75)
IQR = Q3 - Q1

pos_outlier = Q3 + 1.5 * IQR

neg_outlier = Q1 - 1.5 * IQR
```

In [22]:
```
print(Q1)
print("*************"*5)
print(Q3)
print("*************"*5)
print(IQR)
print("*************"*5)
print(pos_outlier)
print("*************"*5)
print(neg_outlier)
print("*************"*5)
```

```
Avg. Area Income                 61485.150192
Avg. Area House Age                  5.322283
Avg. Area Number of Rooms            6.299692
Area Population                  29403.928700
Price                           997577.135075
Name: 0.25, dtype: float64
*****************************************************************
Avg. Area Income                7.576652e+04
Avg. Area House Age             6.650808e+00
Avg. Area Number of Rooms       7.665281e+00
Area Population                 4.286129e+04
Price                           1.471210e+06
Name: 0.75, dtype: float64
*****************************************************************
Avg. Area Income                 14281.368910
Avg. Area House Age                  1.328525
Avg. Area Number of Rooms            1.365589
Area Population                  13457.362070
Price                           473633.069425
dtype: float64
*****************************************************************
Avg. Area Income                9.718857e+04
Avg. Area House Age             8.643597e+00
Avg. Area Number of Rooms       9.713664e+00
Area Population                 6.304733e+04
Price                           2.181660e+06
dtype: float64
*****************************************************************
Avg. Area Income                 40063.096827
Avg. Area House Age                  3.329495
Avg. Area Number of Rooms            4.251308
Area Population                   9217.885595
Price                           287127.530937
dtype: float64
*****************************************************************
```

In [23]:
```
new_df = USAHousing.copy()
```

In [24]:
```
income_q1 = new_df['Avg. Area Income'].quantile(0.25)
income_q3 = new_df['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 [25]:
```
new_df['Avg. Area Income'] = np.where(new_df['Avg. Area Income'] > income_upper,
                                      np.where(new_df['Avg. Area Income'] < income
                                               new_df['Avg. Area Income']) )
```

In [26]:
```
age_q1 = new_df['Avg. Area House Age'].quantile(0.25)
age_q3 = new_df['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 [27]:
```
new_df['Avg. Area House Age'] = np.where(new_df['Avg. Area House Age'] > age_uppe
                                      np.where(new_df['Avg. Area House Age'] < age
                                               new_df['Avg. Area House Age']) )
```

In [28]:
```
room_q1 = new_df['Avg. Area Number of Rooms'].quantile(0.25)
room_q3 = new_df['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
```

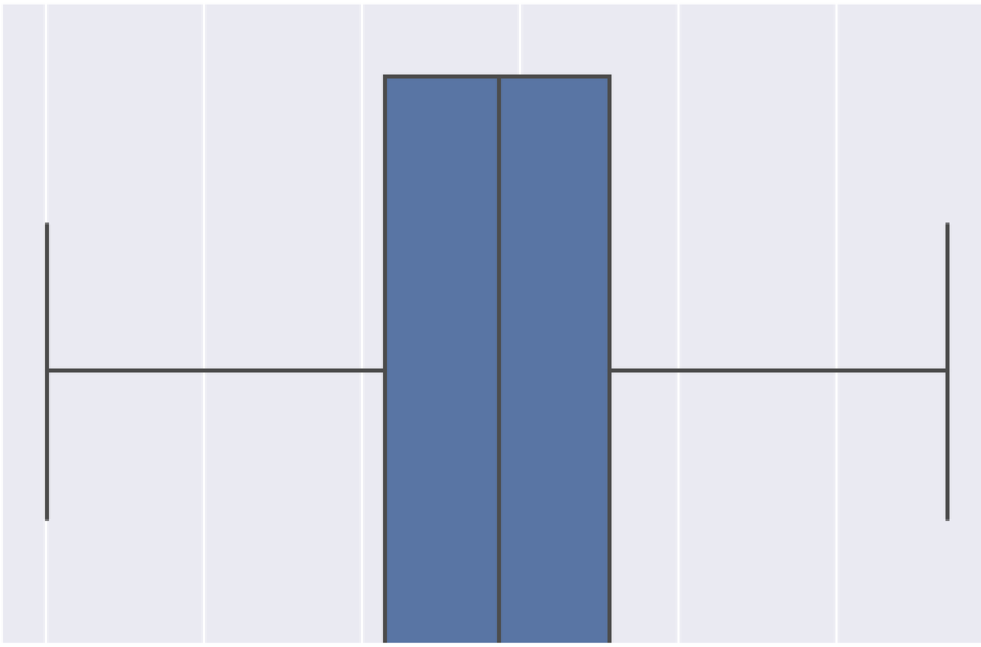
In [29]:
```
new_df['Avg. Area Number of Rooms'] = np.where(new_df['Avg. Area Number of Rooms'
                                      np.where(new_df['Avg. Area Number of Rooms']
                                               new_df['Avg. Area Number of Rooms'])
```

In [30]:
```
pop_q1 = new_df['Area Population'].quantile(0.25)
pop_q3 = new_df['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 [31]:
```
new_df['Area Population'] = np.where(new_df['Area Population'] > pop_upper,pop_up
                                      np.where(new_df['Area Population'] < pop_low
                                               new_df['Area Population']) )
```

In [32]:
```
def boxplots(col):
    sns.boxplot(new_df[col])
    plt.show()

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



In [33]:
```
new_df.head(2)
```

Out[33]:

	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

Spliting the data into independent variable and dependent variable

In [34]:
```
x = new_df.iloc[:,0:-1]
y = new_df['Price']
```

In [35]:
```
x.head()
```

Out[35]:

	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 [36]:
```
y.head()
```

Out[36]:
```
0    1.059034e+06
1    1.505891e+06
2    1.058988e+06
3    1.260617e+06
4    6.309435e+05
Name: Price, dtype: float64
```

Feture Scaling

Only done on Independend variable

In [37]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
sc_x = sc.fit_transform(x)
pd.DataFrame(sc_x)
```

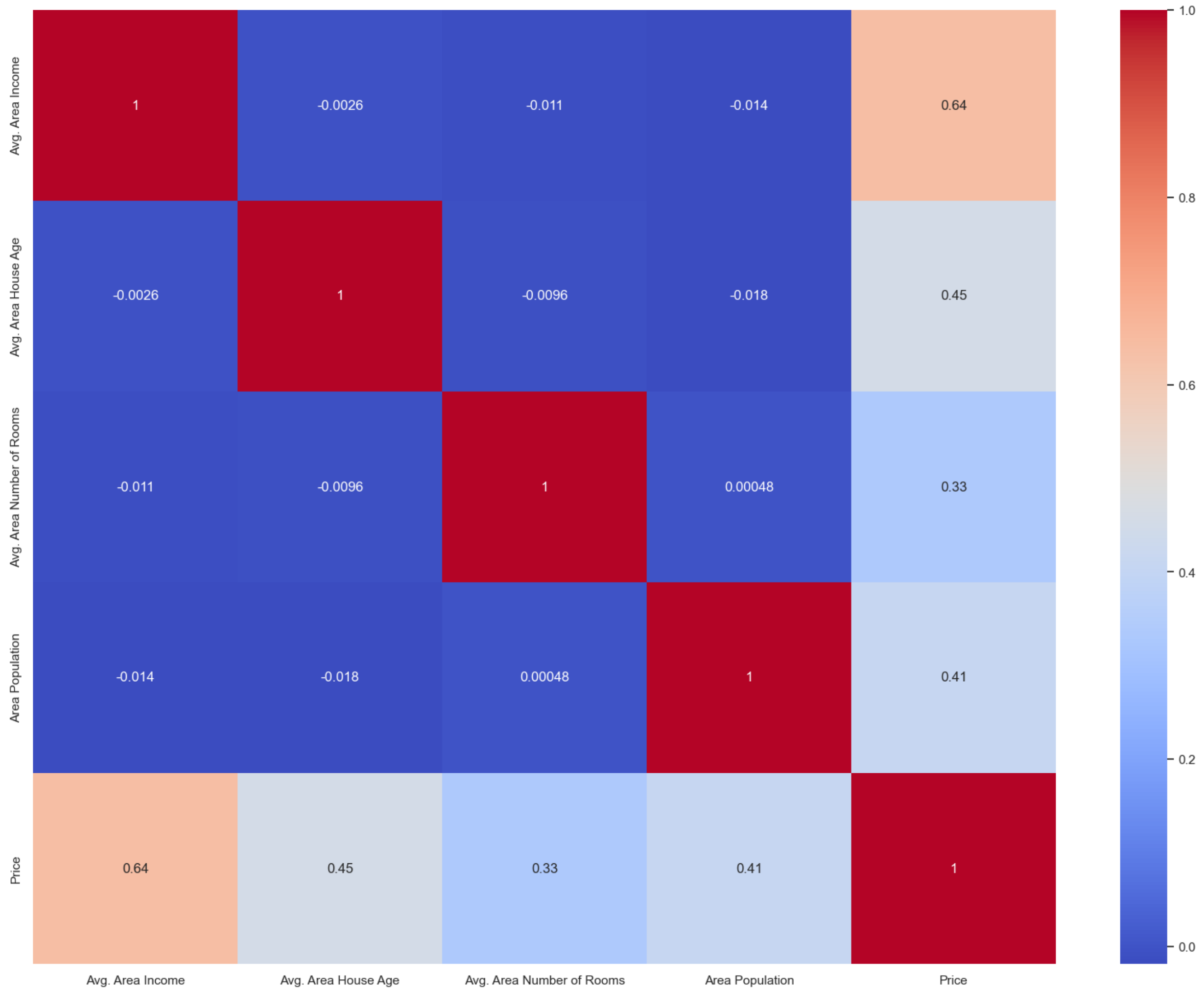
Out[37]:

	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

Finding correlation

In [38]:
```
plt.figure(figsize=(20,15))
corr = new_df.corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.show()
```



VIF - Variance Inflation Factor - to check multicollinearity

In [39]:
```
variable = sc_x
variable.shape
```

Out[39]: (5000, 4)

In [40]:
```
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

vif['Features'] = x.columns
```

A variance inflation factor (VIF) is a measure of the amount of multicollinearity in regression analysis. Multicollinearity exists when there is a correlation between multiple independent variables in a multiple regression model.

In [41]:
```
vif
```

Out[41]:

	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

Split the data into training and test for building the model and for prediction

In [42]:
```
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_

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

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

Building Linear Regression Model

In [43]:
```
from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train, y_train)
```

Out[43]:
```
LinearRegression()
```

In [44]:
```
print(lm.intercept_)
print()
print(lm.coef_)
```

```
-2658303.392038105

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

Predict house price by using linear Regression model with test dataset

In [45]:
```

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

In [46]:
```
y_pred_price
```

Out[46]:
```
array([1258927.55438276,  818030.31383002, 1745948.45303454, ...,
       1119387.19935068,  717217.37633985, 1516456.6242839 ])
```

In [47]:
```
y_test
```

Out[47]:
```
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 the test data and predicted price

In [48]:
```
from sklearn.metrics import r2_score
r2_score(y_test, y_pred_price)
```

Out[48]:
```
0.913609424096595
```

In [49]:
```
r2_score(y_train, y_pred_price_train)
```

Out[49]:
```
0.9164810029819419
```

OLS Method

In [50]:
```
from statsmodels.regression.linear_model import OLS
import statsmodels.regression.linear_model as smf
```

In [51]:
```
reg_model = smf.OLS(endog = y_train, exog=x_train).fit()
```

In [52]:
```
reg_model.summary()
```

Out[52]:

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:	Tue, 18 Jul 2023	**Prob (F-statistic):**	0.00
Time:	17:13:50	**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

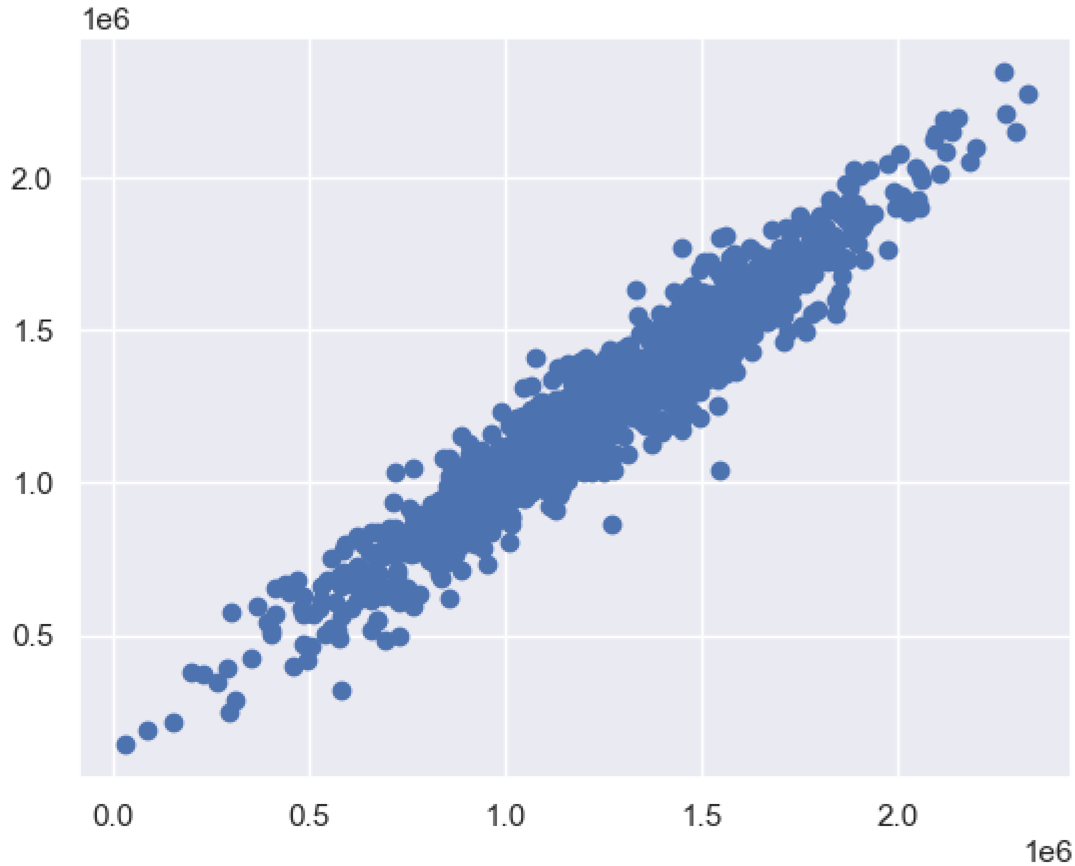
Omnibus:	0.318	**Durbin-Watson:**	1.997
Prob(Omnibus):	0.853	**Jarque-Bera (JB):**	0.368
Skew:	-0.002	**Prob(JB):**	0.832
Kurtosis:	2.952	**Cond. No.**	7.91e+04

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.

Checking linearity

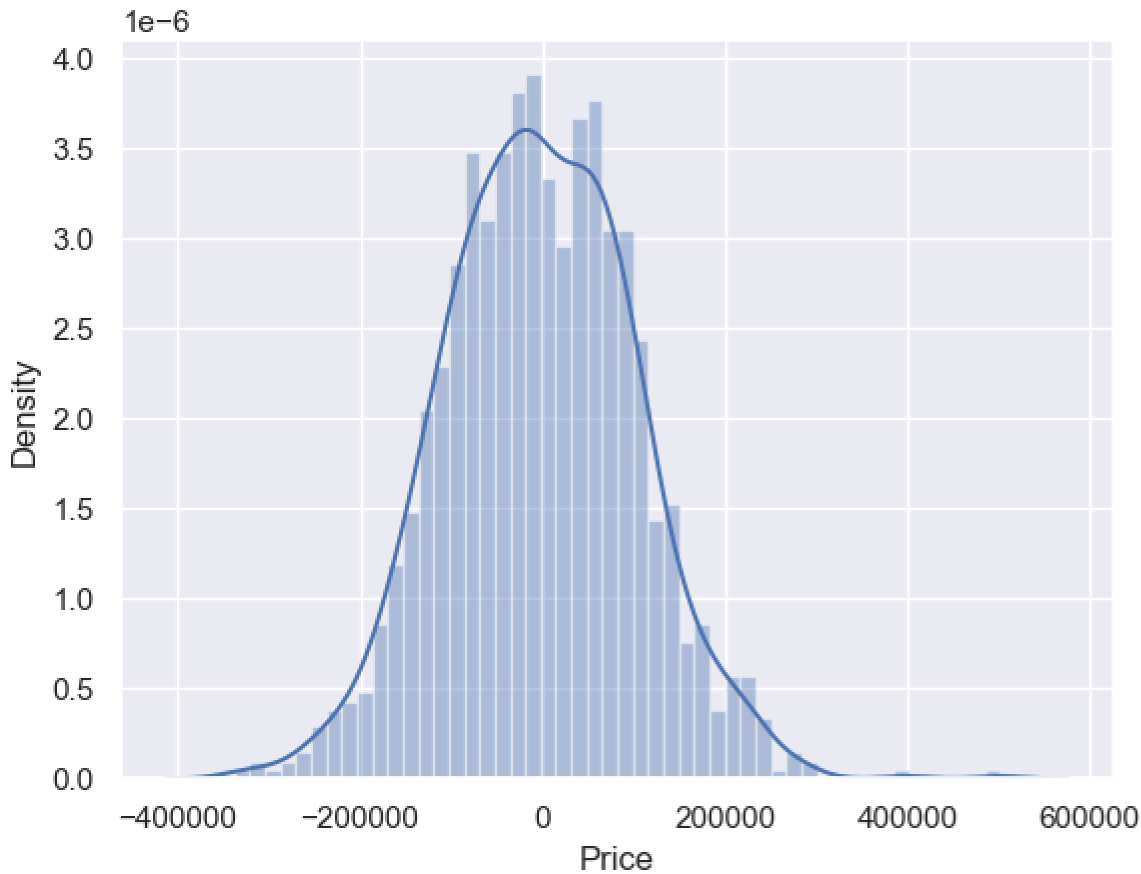
In [53]:
```
plt.scatter(y_test, y_pred_price)
```

Out[53]: <matplotlib.collections.PathCollection at 0x29144fd7700>



Checking Normality of Residual

In [54]:
```
sns.distplot((y_test - y_pred_price), bins=50)
plt.show()
```



Concluding this model

Adj. R-squared (uncentered): 0.964

All variable is statically significant (p <= 0.05)

No bias and variance found

Assumptions

1) Linearity - Satisfied

2) Normality of Residuals- Satisfied

3) Homoscedasticity - Satisfied (there is no outlier and residual is normaly distributed)

In [ ]:

```

```

In [ ]:

```

```

By using sklearn linear model

training accuracy : 91.6%

test accuracy = 91.3%

Lasso regularization

In [56]:

```
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+01]
```

In [57]:

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

In [58]:

```
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
```

Ridge Regression (L2- Regularization)

In [59]:
```
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+01]
```

In [60]:
```
y_pred_train_ridge = ridge.predict(x_train)
y_pred_test_ridge = ridge.predict(x_test)
```

In [61]:
```
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 : 0.9164810006924011

Test Accuracy : 0.9136091673700831
```

In [ ]:

ElasticNet

In [62]:
```
from sklearn.linear_model import ElasticNet
elastic = ElasticNet(alpha=0.3, l1_ratio=0.1)
elastic.fit(x_train, y_train)
```

Out[62]:
```
ElasticNet(alpha=0.3, l1_ratio=0.1)
```

In [63]:
```
y_pred_train_elastic = elastic.predict(x_train)
y_pred_test_elastic = elastic.predict(x_test)
```

In [64]:
```
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 : 0.8957612813811744
```

In [ ]:

Performance matrix

Mean Absolute Error (MAE)

In [65]:
```
from sklearn import metrics
```

In [66]:
```
print("MAE :", metrics.mean_absolute_error(y_test, y_pred_price))
```

```
MAE : 83069.03172980985
```

In [ ]:

Mean Absolute Percent Error (MAPE)

In [67]:
```
print("MAPE :", metrics.mean_absolute_error(y_test, y_pred_price)/100)
```

```
MAPE : 830.6903172980985
```

In [ ]:

Mean Squared Error (MSE)

In [68]:
```
print("MSE :", metrics.mean_squared_error(y_test, y_pred_price))
```

```
MSE : 10796308545.089207
```

In [ ]:

Root Mean Squared Error (RMSE)

In [69]:
```
print("RMSE :", np.sqrt(metrics.mean_squared_error(y_test, y_pred_price)))
```

```
RMSE : 103905.28641551019
```

In [ ]:

Gradient Descent

In [ ]:

In [70]:
```
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, ranc

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

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

In [ ]:

In [71]:
```
from sklearn.linear_model import SGDRegressor
```

In [72]:
```
gd_model = SGDRegressor()
gd_model.fit(x_train, y_train)
```

Out[72]:
```
SGDRegressor()
```

In [73]:
```
y_pred_gd_train = gd_model.predict(x_train)

y_pred_gd_test = gd_model.predict(x_test)
```

In [74]:
```
print("GD Trainging Accuracy :", r2_score(y_train, y_pred_gd_train))

print()

print("GD Test Accuracy :", r2_score(y_test, y_pred_gd_test))
```

```
GD Trainging Accuracy : 0.9164489870225674

GD Test Accuracy : 0.9136865188216712
```

In [ ]: