Importing required Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

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

Loading the csv file

In [2]:
```
df=pd.read_csv("RealEstates.csv")
```

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

Out[3]:

	ids	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	
0	0	79545.458574	missing	?	NaN	23086.800503	$1059033
1	1	79248.642455	6.0028998082752425	6.730821019094919	3.09	40173.072174	Rs20078.54
2	2	61287.067179	5.865889840310001	8.512727430375099	5.13	36882.159400	$1058987
3	3	63345.240046	7.1882360945186425	?	NaN	34310.242831	Rs16808.22
4	4	59982.197226	5.040554523106283	7.839387785120487	4.23	26354.109472	$630943

Checking for the info:

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 9 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   ids                           5000 non-null   int64  
 1   Avg. Area Income              5000 non-null   float64
 2   Avg. Area House Age           5000 non-null   object 
 3   Avg. Area Number of Rooms     5000 non-null   object 
 4   Avg. Area Number of Bedrooms  3333 non-null   float64
 5   Area Population               5000 non-null   float64
 6   Price                         5000 non-null   object 
 7   Address                       5000 non-null   object 
 8   Avg Area Comfort              200 non-null    float64
dtypes: float64(4), int64(1), object(4)
memory usage: 351.7+ KB
```

-From here we can see that, in this dataset total 9 columns and 5000 5ows are present

-In Avg. Area Number of Bedrooms & Avg Area Comfort we can see null values are present.

-For that we need to perform, df.fillna() inorder to remove the null values.

In [5]:
```
df["Avg. Area Number of Bedrooms"].replace("NaN",np.nan, inplace=True)
df["Avg. Area Number of Bedrooms"]=df["Avg. Area Number of Bedrooms"].astype("flo
amean = df["Avg. Area Number of Bedrooms"].mean()
df["Avg. Area Number of Bedrooms"].fillna(amean, inplace = True)
```

-As we know that, if more than 80% of values are null in a particular column we have to drop it by using df.drop()

-We are going to drop Avg. Area Comfort.

In [6]:
```
df.drop('Avg Area Comfort',axis=1,inplace=True)
```

Now agian we are going to check df.head()

In [7]:
```
df.head()
```

Out[7]:

	ids	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	
0	0	79545.458574	missing	?	3.998083	23086.800503	$1059033
1	1	79248.642455	6.0028998082752425	6.730821019094919	3.090000	40173.072174	Rs20078.54
2	2	61287.067179	5.865889840310001	8.512727430375099	5.130000	36882.159400	$1058987
3	3	63345.240046	7.1882360945186425	?	3.998083	34310.242831	Rs16808.22
4	4	59982.197226	5.040554523106283	7.839387785120487	4.230000	26354.109472	$630943

-Here we can see, in Avg. Area House Age and Avg. Area Number of Rooms , there is a missing value, now we have to fill it using df.fillna()

In [8]:
```
df["Avg. Area House Age"].replace("missing",np.nan, inplace=True)
df["Avg. Area House Age"]=df["Avg. Area House Age"].astype("float64")
cmean = df["Avg. Area House Age"].mean()
df["Avg. Area House Age"].fillna(amean, inplace = True)
```

In [9]:
```
df["Avg. Area Number of Rooms"].replace("?",np.nan, inplace=True)
df["Avg. Area Number of Rooms"]=df["Avg. Area Number of Rooms"].astype("float64")
dmean = df["Avg. Area Number of Rooms"].mean()
df["Avg. Area Number of Rooms"].fillna(amean, inplace = True)
```

Now again we will check df.head()

In [10]:
```
df.head()
```

Out[10]:

	ids	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	
0	0	79545.458574	3.998083	3.998083	3.998083	23086.800503	$1059033.5578701235	208 Mi 674\
1	1	79248.642455	6.002900	6.730821	3.090000	40173.072174	Rs20078.545531292668	188
2	2	61287.067179	5.865890	8.512727	5.130000	36882.159400	$1058987.9878760849	Stravenu
3	3	63345.240046	7.188236	3.998083	3.998083	34310.242831	Rs16808.224088392624	USS B
4	4	59982.197226	5.040555	7.839388	4.230000	26354.109472	$630943.4893385402	USNS

In [11]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 8 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   ids                           5000 non-null   int64  
 1   Avg. Area Income              5000 non-null   float64
 2   Avg. Area House Age           5000 non-null   float64
 3   Avg. Area Number of Rooms     5000 non-null   float64
 4   Avg. Area Number of Bedrooms  5000 non-null   float64
 5   Area Population               5000 non-null   float64
 6   Price                         5000 non-null   object 
 7   Address                       5000 non-null   object 
dtypes: float64(5), int64(1), object(2)
memory usage: 312.6+ KB
```

Now we can see, no NULL values are present.

Now going to split state from the given address column:

In [12]:
```
df["Address"][4].split()[-2]
```

Out[12]:
```
'AE'
```

In [13]:
```
def state(x):
    s = x.split()[-2]
    return s
```

In [14]:
```
df["State"] = df["Address"].apply(state)
```

In [15]:
```
df["State"]
```

Out[15]:
```
0       NE
1       CA
2       WI
3       AP
4       AE
        ..
4995    AP
4996    AA
4997    VA
4998    AE
4999    NV
Name: State, Length: 5000, dtype: object
```

In [16]:
```
df.head()
```

Out[16]:

	ids	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	
0	0	79545.458574	3.998083	3.998083	3.998083	23086.800503	$1059033.5578701235	208 Mi 674\
1	1	79248.642455	6.002900	6.730821	3.090000	40173.072174	Rs20078.545531292668	188
2	2	61287.067179	5.865890	8.512727	5.130000	36882.159400	$1058987.9878760849	Stravenu
3	3	63345.240046	7.188236	3.998083	3.998083	34310.242831	Rs16808.224088392624	USS B
4	4	59982.197226	5.040555	7.839388	4.230000	26354.109472	$630943.4893385402	USNS F

Droping the Address and ids columns:

In [17]:
```
df.drop("ids",axis = 1, inplace=True)
```

In [18]:
```
df.drop("Address",axis = 1, inplace=True)
```

In [19]:
```
df.head()
```

Out[19]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price	State
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	$1059033.5578701235	NE
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	Rs20078.545531292668	CA
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	$1058987.9878760849	WI
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	Rs16808.224088392624	AP
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	$630943.4893385402	AE

Now we are going to convert all the price in Rs. to Dollar:

In [20]:
```
df["Price"][0][0]
```

Out[20]:
```
'$'
```

Creating a function in order to change the Rs. into Dollars

In [21]:
```
def convert(x):
    if(x[0]=="R"):
        x=x.split("R")
        x=x[-1]
        x=float(x)*75
        return x
    else:
        x=x.split("$")
        x=x[-1]
        return x
```

In [22]:
```
df['Price']=df['Price'].str.replace('s','')
```

In [23]:
```
df["Price(in dollars)"]=df["Price"].apply(convert)
```

In [24]:
```
df["Price(in dollars)"]
```

Out[24]:
```
0       1059033.5578701235
1           1505890.914847
2       1058987.9878760849
3           1260616.806629
4        630943.4893385402
              ...
4995        1060193.785885
4996     1482617.728622024
4997        1030729.583152
4998    1198656.8724076871
4999        1298950.480267
Name: Price(in dollars), Length: 5000, dtype: object
```

In [25]:
```
df.drop("Price", axis=1,inplace=True)
```

Checking df.head()

In [26]:
```
df.head()
```

Out[26]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1059033.5578701235
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1505890.914847
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1058987.9878760849
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1260616.806629
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	630943.4893385402

Checking the details by using df.describe()

In [27]:
```
df.describe()
```

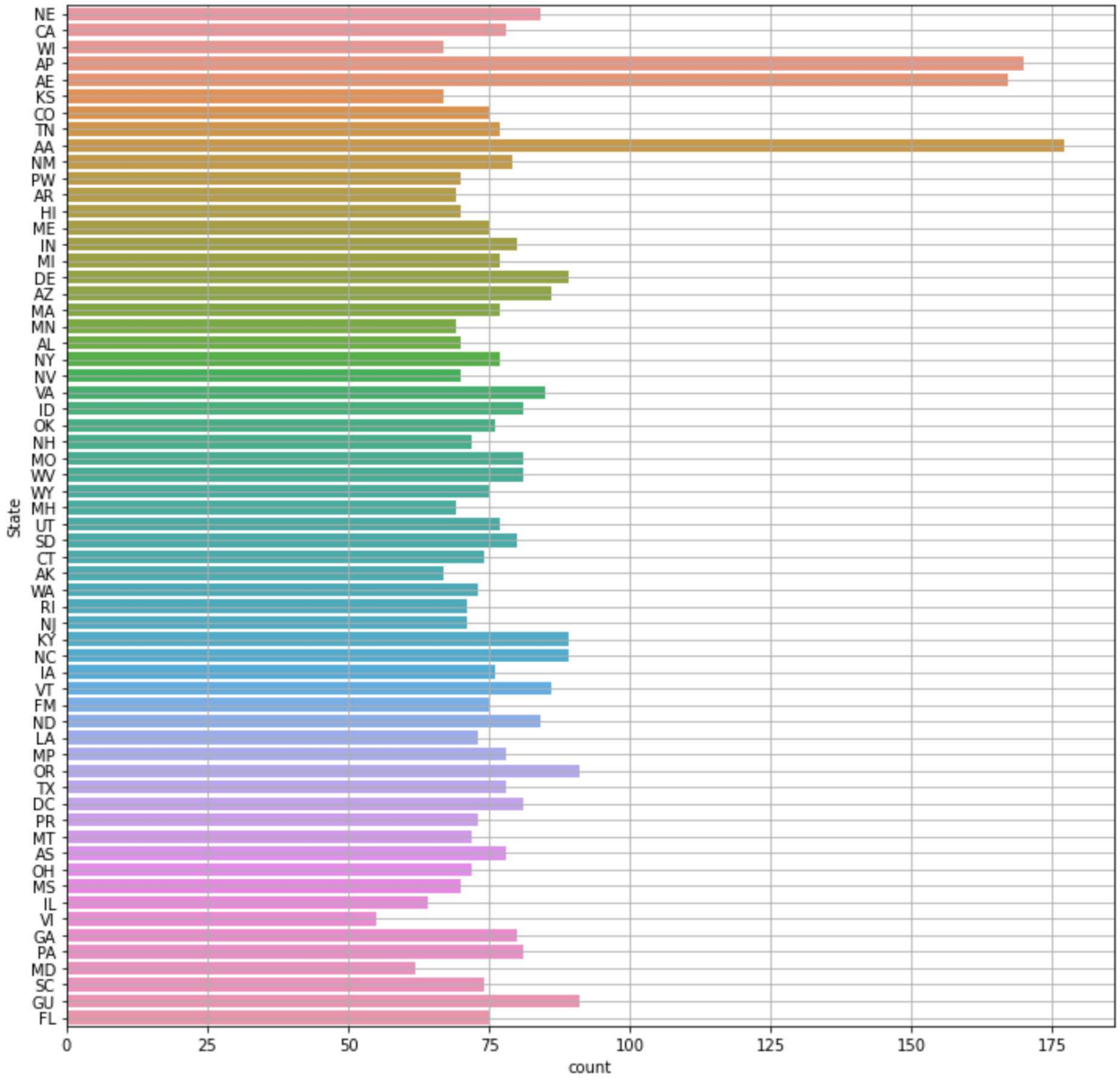
Out[27]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000
mean	68583.108984	5.974841	5.995437	3.998083	36163.516039
std	10657.991214	0.992272	1.638331	1.011621	9925.650114
min	17796.631190	2.644304	3.236194	2.000000	172.610686
25%	61480.562388	5.320337	3.998083	3.360000	29403.928702
50%	68804.286404	5.969828	6.290059	3.998083	36199.406689
75%	75783.338666	6.649509	7.341897	4.280000	42861.290769
max	107701.748378	9.519088	10.219902	6.500000	69621.713378

EDA PREPROCESSING

Now going to check: How many states are present in the dataset by using Countplot().

In [28]:
```
plt.figure(figsize=(12,12))
sns.countplot(data=df, y="State")
plt.grid(True)
plt.show()
```



-From this we can see, The count of AA state is more in dataset and VI state has less count.

In [29]:
```
df["Price(in dollars)"]
```

Out[29]:
```
0       1059033.5578701235
1           1505890.914847
2       1058987.9878760849
3           1260616.806629
4        630943.4893385402
              ...
4995        1060193.785885
4996     1482617.728622024
4997        1030729.583152
4998    1198656.8724076871
4999        1298950.480267
Name: Price(in dollars), Length: 5000, dtype: object
```

Here, the price is in object we have to convert it into Float. for that we have to use astype()

In [30]:
```
df["Price(in dollars)"]=df["Price(in dollars)"].astype("float64")
```

In [31]:
```
df["Price(in dollars)"]
```

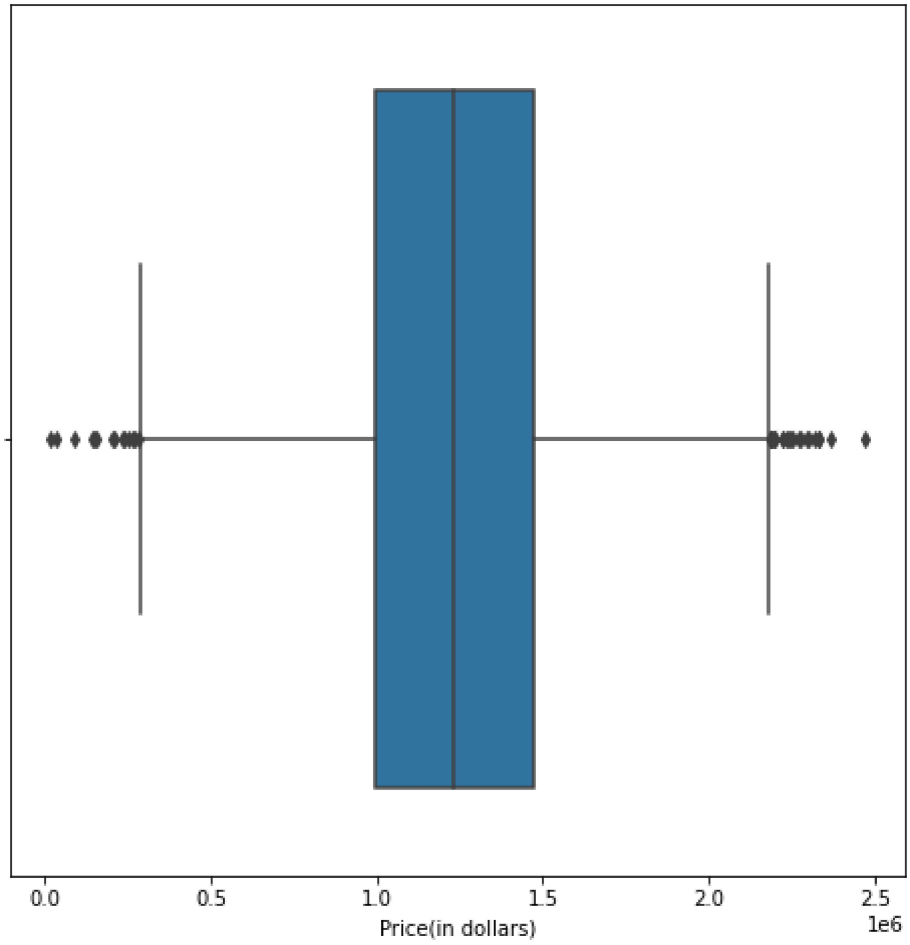
Out[31]:
```
0       1.059034e+06
1       1.505891e+06
2       1.058988e+06
3       1.260617e+06
4       6.309435e+05
              ...
4995    1.060194e+06
4996    1.482618e+06
4997    1.030730e+06
4998    1.198657e+06
4999    1.298950e+06
Name: Price(in dollars), Length: 5000, dtype: float64
```

Now we can see the price got converted into float.

Now we will use boxplot on price to check

whether it have any outliers or not.

In [32]:
```
plt.figure(figsize=(8,8))
sns.boxplot(data=df, x="Price(in dollars)")
plt.show()
```



Here, in boxplot we can see outliers are present, so we need to delete these outliers by using df.drop()

Treating the OUTLIERS

In [33]:
```
df[(df["Price(in dollars)"]<-2)]
```

Out[33]:

Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)

In [34]:
```
df.drop([38,39,90,107,110,4744,4855,4889,4898,4990],inplace=True)
```

In [35]:
```
df[(df["Price(in dollars)"]<-2)]
```

Out[35]:

Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)

In [36]:
```
df.drop([128,137,263,269,437,4637,4781,4715,4729,4733],inplace=True)
```

In [37]:
```
df[(df["Price(in dollars)"]<-2)]
```

Out[37]:

Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)

In [38]:
```
df.drop([638,696,825,856,993,4375,4411,4451,4589,4681],inplace=True)
```

In [39]:
```
df[(df["Price(in dollars)"]<-2)]
```

Out[39]:

Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)

In [40]:
```
df.drop([1064,1074,1081,1088,1107,4182,4217,4280,4310,4357],inplace=True)
```

In [41]:
```
df[(df["Price(in dollars)"]<-2)]
```

Out[41]:

Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)

In [42]:
```
df.drop([1141,1212,1221,1259,1271,3785,3787,3922,4034,4116],inplace=True)
```

In [43]:
```
df[(df["Price(in dollars)"]<-2)]
```

Out[43]:

Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)

In [44]:
```
df.drop([1354,1356,1363,1459,1492,1510,1578,1592,1601,1615,1628,1657,1661,1672,17
```

In [45]:
```
df[(df["Price(in dollars)"]>2)]
```

Out[45]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1.059034e+06
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1.505891e+06
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1.058988e+06
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1.260617e+06
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	6.309435e+05
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	3.998083	3.998083	22837.361035	AP	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.020000	25616.115489	AA	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.130000	33266.145490	VA	1.030730e+06
4998	68001.331235	5.534388	3.998083	3.998083	42625.620156	AE	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.070000	46501.283803	NV	1.298950e+06

4890 rows × 7 columns

In [46]:
```
df.drop([20,94,126,256,334,4487,4597,4682,4959,4962], inplace=True)
```

In [47]:
```
df[(df["Price(in dollars)"]>2)]
```

Out[47]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1.059034e+06
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1.505891e+06
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1.058988e+06
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1.260617e+06
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	6.309435e+05
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	3.998083	3.998083	22837.361035	AP	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.020000	25616.115489	AA	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.130000	33266.145490	VA	1.030730e+06
4998	68001.331235	5.534388	3.998083	3.998083	42625.620156	AE	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.070000	46501.283803	NV	1.298950e+06

4880 rows × 7 columns

In [48]:
```
df.drop([352,355,417,465,498,4372,4400,4439,4448,4450], inplace=True)
```

In [49]:
```
df[(df["Price(in dollars)"]>2)]
```

Out[49]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1.059034e+06
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1.505891e+06
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1.058988e+06
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1.260617e+06
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	6.309435e+05
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	3.998083	3.998083	22837.361035	AP	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.020000	25616.115489	AA	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.130000	33266.145490	VA	1.030730e+06
4998	68001.331235	5.534388	3.998083	3.998083	42625.620156	AE	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.070000	46501.283803	NV	1.298950e+06

4870 rows × 7 columns

In [50]:
```
df.drop([567,581,622,693,700,4030,4123,4129,4184,4197], inplace=True)
```

In [51]:
```
df[(df["Price(in dollars)"]>2)]
```

Out[51]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1.059034e+06
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1.505891e+06
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1.058988e+06
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1.260617e+06
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	6.309435e+05
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	3.998083	3.998083	22837.361035	AP	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.020000	25616.115489	AA	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.130000	33266.145490	VA	1.030730e+06
4998	68001.331235	5.534388	3.998083	3.998083	42625.620156	AE	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.070000	46501.283803	NV	1.298950e+06

4860 rows × 7 columns

In [52]:
```
df.drop([715,770,795,880,901,3798,3861,3905,3947,3967], inplace=True)
```

In [53]:
```
df[(df["Price(in dollars)"]>2)]
```

Out[53]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1.059034e+06
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1.505891e+06
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1.058988e+06
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1.260617e+06
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	6.309435e+05
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	3.998083	3.998083	22837.361035	AP	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.020000	25616.115489	AA	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.130000	33266.145490	VA	1.030730e+06
4998	68001.331235	5.534388	3.998083	3.998083	42625.620156	AE	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.070000	46501.283803	NV	1.298950e+06

4850 rows × 7 columns

In [54]:
```
df.drop([924,962,971,990,1016,3658,3664,3698,3712,3765], inplace=True)
```

In [55]:
```
df[(df["Price(in dollars)"]>2)]
```

Out[55]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1.059034e+06
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1.505891e+06
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1.058988e+06
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1.260617e+06
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	6.309435e+05
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	3.998083	3.998083	22837.361035	AP	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.020000	25616.115489	AA	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.130000	33266.145490	VA	1.030730e+06
4998	68001.331235	5.534388	3.998083	3.998083	42625.620156	AE	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.070000	46501.283803	NV	1.298950e+06

4840 rows × 7 columns

In [56]:
```
df.drop([1096,1128,1195,1208,1236,1248,1281,1321,1322,1326,1436,1462,1485,1561,15
```

In [57]:
```
df[(df["Price(in dollars)"]>2)]
```

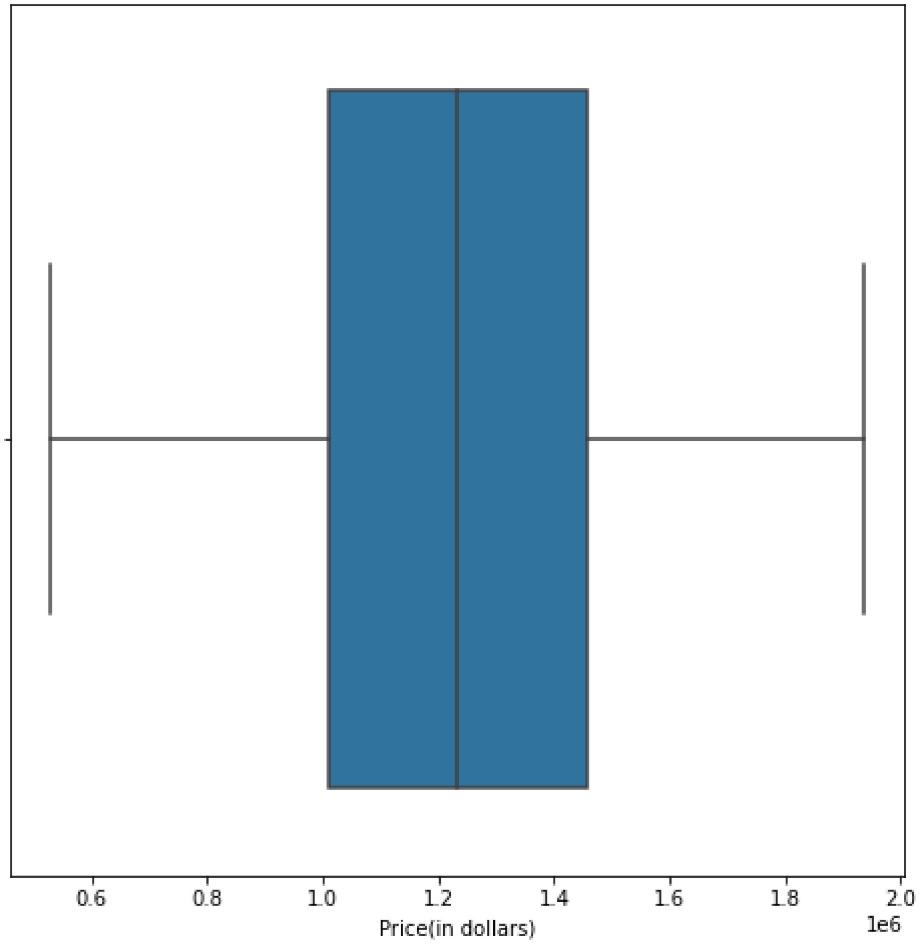
Out[57]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	79545.458574	3.998083	3.998083	3.998083	23086.800503	NE	1.059034e+06
1	79248.642455	6.002900	6.730821	3.090000	40173.072174	CA	1.505891e+06
2	61287.067179	5.865890	8.512727	5.130000	36882.159400	WI	1.058988e+06
3	63345.240046	7.188236	3.998083	3.998083	34310.242831	AP	1.260617e+06
4	59982.197226	5.040555	7.839388	4.230000	26354.109472	AE	6.309435e+05
...	...	...	...	...	...	...	...
4995	60567.944140	7.830362	3.998083	3.998083	22837.361035	AP	1.060194e+06
4996	78491.275435	6.999135	6.576763	4.020000	25616.115489	AA	1.482618e+06
4997	63390.686886	7.250591	4.805081	2.130000	33266.145490	VA	1.030730e+06
4998	68001.331235	5.534388	3.998083	3.998083	42625.620156	AE	1.198657e+06
4999	65510.581804	5.992305	6.792336	4.070000	46501.283803	NV	1.298950e+06

4781 rows × 7 columns

In [58]:
```
df.drop([1516], inplace=True)
```

In [59]:
```
plt.figure(figsize=(8,8))
sns.boxplot(data=df, x="Price(in dollars)")
plt.show()
```



We dropped all the outliers from the price

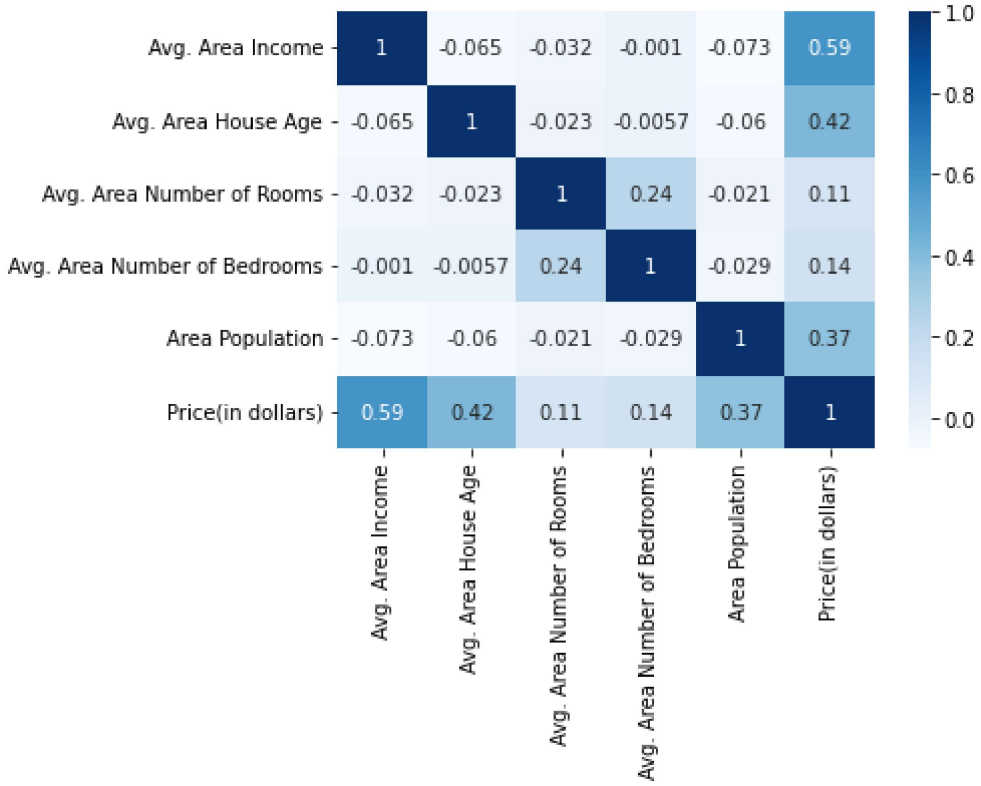
Now let's see the correlation between the datapoints

In [60]:
```
df.corr()["Price(in dollars)"].sort_values()
```

Out[60]:
```
Avg. Area Number of Rooms       0.114002
Avg. Area Number of Bedrooms    0.135489
Area Population                 0.371776
Avg. Area House Age             0.417126
Avg. Area Income                0.590662
Price(in dollars)               1.000000
Name: Price(in dollars), dtype: float64
```

In [61]:
```
sns.heatmap(df.corr(), cmap="Blues", annot=True)
```

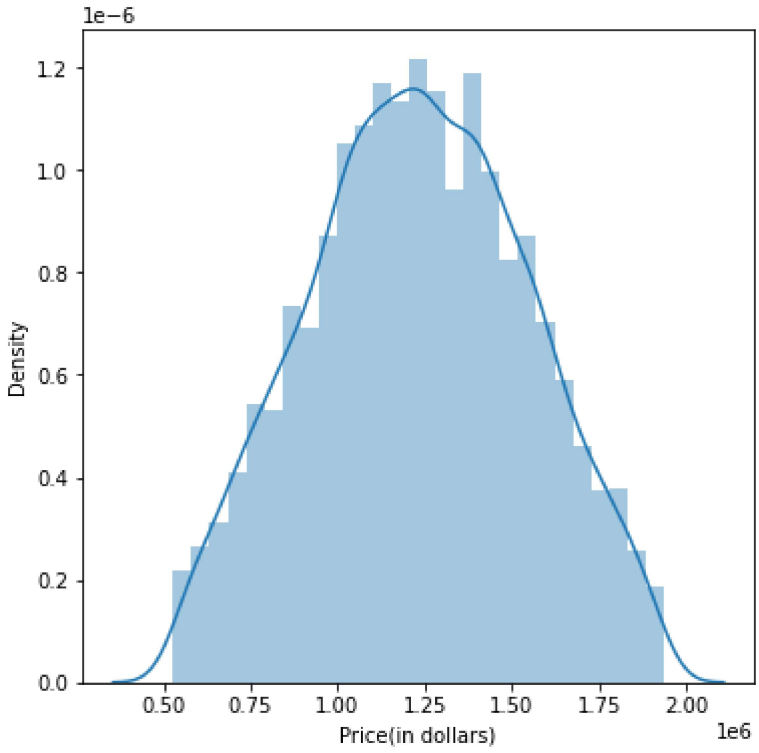
Out[61]: <AxesSubplot:>



In [62]:
```
df.skew()
```

Out[62]:
```
Avg. Area Income               -0.047727
Avg. Area House Age            -0.015939
Avg. Area Number of Rooms      -0.009659
Avg. Area Number of Bedrooms    0.451226
Area Population                 0.057389
Price(in dollars)              -0.005557
dtype: float64
```

In [63]:
```
plt.figure(figsize=(6,6))
sns.distplot(df["Price(in dollars)"])
plt.show()
```



Label Encoding

We can see that the state is in categorical form we have to convert it in numeric values for that we going to use Label Encoding

In [64]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df["State"]= le.fit_transform(df["State"])
```

In [65]:
```
df["State"]
```

Out[65]:
```
0       37
1        8
2       59
3        4
4        1
        ..
4995     4
4996     0
4997    55
4998     1
4999    41
Name: State, Length: 4780, dtype: int32
```

In [66]:
```
from sklearn.preprocessing import StandardScaler
for col in df:
    sc=StandardScaler()
    df[col]= sc.fit_transform(df[[col]])
```

Now we are going to separate, columns in x and y

In x we will have independent variable and in y we will be having dependent variable

In [67]:
```
x = x=df[['Avg. Area Income','Avg. Area House Age','Avg. Area Number of Rooms','A
y = df.iloc[:,-1]
```

In [68]:
```
x
```

Out[68]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State
0	1.083215	-2.023217	-1.220425	-0.006785	-1.340776	0.435222
1	1.053970	0.028262	0.454311	-0.903619	0.409213	-1.124754
2	-0.715812	-0.111937	1.546337	1.111110	0.072155	1.618652
3	-0.513017	1.241186	-1.220425	-0.006785	-0.191263	-1.339923
4	-0.844383	-0.956482	1.133687	0.222259	-1.006136	-1.501300
...	...	...	...	...	...	...
4995	-0.786669	1.898258	-1.220425	-0.006785	-1.366324	-1.339923
4996	0.979345	1.047684	0.359897	0.014861	-1.081722	-1.555092
4997	-0.508539	1.304992	-0.725863	-1.851727	-0.298200	1.403483
4998	-0.054246	-0.451154	-1.220425	-0.006785	0.660405	-1.501300
4999	-0.299663	0.017421	0.492010	0.064241	1.057353	0.650391

4780 rows × 6 columns

In [69]:
```
y
```

Out[69]:
```
0      -0.552753
1       0.876728
2      -0.552898
3       0.092105
4      -1.922198
          ...
4995   -0.549041
4996    0.802278
4997   -0.643296
4998   -0.106103
4999    0.214733
Name: Price(in dollars), Length: 4780, dtype: float64
```

Now we will use train_test_split to train and test the data

In [70]:
```
from sklearn.model_selection import train_test_split, cross_val_score
xtrain,xtest,ytrain,ytest = train_test_split(x,y,test_size=0.30, random_state=1)
```

Now we are going to use Linear Regression

In [71]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(xtrain,ytrain)
ypred= lr.predict(xtest)
```

Importing Evaluation metrics for Regression

In [72]:
```
from sklearn.metrics import mean_absolute_error,mean_squared_error,r2_score
print("Accuracy : ",r2_score(ytest,ypred))
print("MAE : ",mean_absolute_error(ytest,ypred))
print("MSE : ",mean_squared_error(ytest,ypred))
print("RMSE : ",np.sqrt(mean_squared_error(ytest,ypred)))
```

```
Accuracy :  0.7904736851215167
MAE :  0.3613421582899399
MSE :  0.21196227344480914
RMSE :  0.4603936070850779
```

Here we can see, we are getting accuracy : 0.7904736851215167 i.e 79%

Now we are going to perfrom Hyperparameter tuning

In [73]:
```
from sklearn.linear_model import Ridge,Lasso
```

In [74]:
```
for i in range(1,100):
    l2=Ridge(alpha=i)
    l2.fit(xtrain,ytrain)
    print(f"{i}= {l2.score(xtest,ytest)}")
```

```
1= 0.7904715403407295
2= 0.7904692107439805
3= 0.7904666967158726
4= 0.7904639986402097
5= 0.790461116899998
6= 0.790458051877448
7= 0.7904548039539764
8= 0.7904513735102077
9= 0.7904477609259765
10= 0.7904439665803286
11= 0.7904399908515236
12= 0.7904358341170362
13= 0.7904314967535582
14= 0.7904269791370003
15= 0.7904222816424938
16= 0.7904174046443926
17= 0.7904123485162744
18= 0.7904071136309436
19= 0.7904017003604321
20= 0.7903961090760013
21= 0.7903903401481441
22= 0.7903843939465865
23= 0.7903782708402894
24= 0.7903719711974505
25= 0.7903654953855057
26= 0.7903588437711311
27= 0.7903520167202447
28= 0.7903450145980083
29= 0.790337837768829
30= 0.7903304865963611
31= 0.7903229614435073
32= 0.7903152626724215
33= 0.7903073906445095
34= 0.7902993457204311
35= 0.790291128260102
36= 0.7902827386226952
37= 0.7902741771666426
38= 0.7902654442496373
39= 0.7902565402286347
40= 0.7902474654598541
41= 0.7902382202987813
42= 0.7902288051001692
43= 0.7902192202180398
44= 0.7902094660056865
45= 0.7901995428156748
46= 0.7901894509998446
47= 0.7901791909093115
48= 0.790168762894469
49= 0.7901581673049893
50= 0.7901474044898259
51= 0.7901364747972144
52= 0.7901253785746746
```

```
53= 0.790114116169012
54= 0.7901026879263194
55= 0.790091094191979
56= 0.790079335310663
57= 0.790067411626336
58= 0.7900553234822566
59= 0.7900430712209788
60= 0.7900306551843534
61= 0.7900180757135302
62= 0.7900053331489587
63= 0.7899924278303908
64= 0.7899793600968813
65= 0.7899661302867903
66= 0.7899527387377844
67= 0.7899391857868382
68= 0.7899254717702361
69= 0.7899115970235738
70= 0.7898975618817599
71= 0.7898833666790172
72= 0.7898690117488845
73= 0.7898544974242182
74= 0.7898398240371938
75= 0.789824991919307
76= 0.7898100014013761
77= 0.7897948528135428
78= 0.7897795464852739
79= 0.7897640827453632
80= 0.7897484619219327
81= 0.7897326843424338
82= 0.7897167503336495
83= 0.7897006602216956
84= 0.7896844143320219
85= 0.7896680129894144
86= 0.789651456517996
87= 0.7896347452412287
88= 0.7896178794819145
89= 0.7896008595621976
90= 0.7895836858035649
91= 0.7895663585268484
92= 0.7895488780522262
93= 0.7895312446992242
94= 0.7895134587867174
95= 0.7894955206329313
96= 0.7894774305554435
97= 0.7894591888711853
98= 0.789440795896443
99= 0.789422251946859
```

In [75]:
```
l2=Ridge(alpha=1)
l2.fit(xtrain,ytrain)
l2.score(xtest,ytest)
```

Out[75]: 0.7904715403407295

In [76]:
```
l1=Ridge(alpha=1)
l1.fit(xtrain,ytrain)
l1.score(xtest,ytest)
```

Out[76]:
```
0.7904715403407295
```

In [77]:
```
df.head()
```

Out[77]:

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	State	Price(in dollars)
0	1.083215	-2.023217	-1.220425	-0.006785	-1.340776	0.435222	-0.552753
1	1.053970	0.028262	0.454311	-0.903619	0.409213	-1.124754	0.876728
2	-0.715812	-0.111937	1.546337	1.111110	0.072155	1.618652	-0.552898
3	-0.513017	1.241186	-1.220425	-0.006785	-0.191263	-1.339923	0.092105
4	-0.844383	-0.956482	1.133687	0.222259	-1.006136	-1.501300	-1.922198

Now creating a function for various model:

In [78]:
```
def mymodel(model):
    model.fit(xtrain,ytrain)
    ypred=model.predict(xtest)
    print("Accuracy : ",r2_score(ytest,ypred))
    print("mae: ", mean_absolute_error(ytest,ypred))
    print("mse: ", mean_squared_error(ytest,ypred))
    print("rmae: ", np.sqrt(mean_absolute_error(ytest,ypred)))
    return model
```

Imporing SVM

In [79]:
```
from sklearn.svm import SVR
svm=SVR()
svm.fit(xtrain,ytrain)
ypred=svm.predict(xtest)
```

In [80]:
```
mymodel(svm)
```

```
Accuracy :  0.8295754338218292
mae:  0.3246562551094187
mse:  0.1724059267635219
rmae:  0.5697861485763046
```

Out[80]:
```
SVR()
```

In [81]:
```
svm=SVR(kernel="linear")
mymodel(svm)
```

```
Accuracy :  0.7908277659860975
mae:  0.3610387836399438
mse:  0.21160407602658335
rmae:  0.6008650294699666
```

Out[81]:
```
SVR(kernel='linear')
```

In [82]:
```
svm=SVR(kernel="rbf")
mymodel(svm)
```

```
Accuracy :  0.8295754338218292
mae:  0.3246562551094187
mse:  0.1724059267635219
rmae:  0.5697861485763046
```

Out[82]:
```
SVR()
```

In [83]:
```
svm=SVR(kernel="sigmoid")
mymodel(svm)
```

```
Accuracy :  -255.00482261378215
mae:  11.377766428233503
mse:  258.98114155982216
rmae:  3.3730944884828684
```

Out[83]:
```
SVR(kernel='sigmoid')
```

In [84]:
```
svm=SVR(kernel="poly")
mymodel(svm)
```

```
Accuracy :  0.6684619992428139
mae:  0.4511034259844216
mse:  0.33539247046174525
rmae:  0.6716423348661262
```

Out[84]:
```
SVR(kernel='poly')
```

In [85]:
```
from sklearn.model_selection import GridSearchCV
```

In [86]:
```
parameters={'C':[1,100],'gamma':[0.1,0.01],'kernel':['rbf']}
```

In [87]:
```
grid=GridSearchCV(SVR(),parameters,verbose=2)
grid.fit(xtrain,ytrain)
```

```
Fitting 5 folds for each of 4 candidates, totalling 20 fits
[CV] END .....................C=1, gamma=0.1, kernel=rbf; total time=   0.3
s
[CV] END .....................C=1, gamma=0.1, kernel=rbf; total time=   0.3
s
[CV] END .....................C=1, gamma=0.1, kernel=rbf; total time=   0.3
s
[CV] END .....................C=1, gamma=0.1, kernel=rbf; total time=   0.3
s
[CV] END .....................C=1, gamma=0.1, kernel=rbf; total time=   0.3
s
[CV] END ....................C=1, gamma=0.01, kernel=rbf; total time=   0.3
s
[CV] END ....................C=1, gamma=0.01, kernel=rbf; total time=   0.3
s
[CV] END ....................C=1, gamma=0.01, kernel=rbf; total time=   0.3
s
[CV] END ....................C=1, gamma=0.01, kernel=rbf; total time=   0.3
s
[CV] END ....................C=1, gamma=0.01, kernel=rbf; total time=   0.3
s
[CV] END ...................C=100, gamma=0.1, kernel=rbf; total time=   3.4
s
[CV] END ...................C=100, gamma=0.1, kernel=rbf; total time=   3.5
s
[CV] END ...................C=100, gamma=0.1, kernel=rbf; total time=   3.0
s
[CV] END ...................C=100, gamma=0.1, kernel=rbf; total time=   3.4
s
[CV] END ...................C=100, gamma=0.1, kernel=rbf; total time=   3.2
s
[CV] END ..................C=100, gamma=0.01, kernel=rbf; total time=   0.7
s
[CV] END ..................C=100, gamma=0.01, kernel=rbf; total time=   0.7
s
[CV] END ..................C=100, gamma=0.01, kernel=rbf; total time=   0.7
s
[CV] END ..................C=100, gamma=0.01, kernel=rbf; total time=   0.6
s
[CV] END ..................C=100, gamma=0.01, kernel=rbf; total time=   0.7
s
```

Out[87]:
```
GridSearchCV(estimator=SVR(),
             param_grid={'C': [1, 100], 'gamma': [0.1, 0.01],
                         'kernel': ['rbf']},
             verbose=2)
```

In [88]:
```
grid.best_params_
```

Out[88]:
```
{'C': 100, 'gamma': 0.01, 'kernel': 'rbf'}
```

In [89]:
```
ypred=grid.predict(xtest)
```

In [90]:
```
ac=r2_score(ytest,ypred)
```

In [91]:
```
ac
```

Out[91]:
```
0.8354094963415167
```

The accuracy is 0.8349739241668995 i.e 84%

User Input

In [94]:
```
income = input("Enter your income: ")
house_age = input("Enter Area House Age ")
no_of_rooms = input("Enter Number of rooms: ")
no_of_bedrooms = input("Enter no. of bedrooms ")
population = input("Enter Area of population ")
state = input("Enter your state: ")

data = [income,house_age,no_of_rooms,no_of_bedrooms,population,state]

print((l2.predict([data])),end="")
```

```
Enter your income: 45888
Enter Area House Age 38
Enter Number of rooms: 23
Enter no. of bedrooms 23
Enter Area of population 34576
Enter your state: 23
[46136.81569977]
```