```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split, cross_val_score,cross_validate
from sklearn.preprocessing import normalize,StandardScaler
from sklearn.linear_model import LinearRegression,Lasso
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error
```

```
df=pd.read_csv("USA_Housing.csv")
```

```
#df=pd.read_csv("USA_Housing.csv",header=None) => it means my dataset didn't contain column names
#Otherwise this dataset considers 1st row is the header.
#So to explicitly provide column names to the dataset follow below instructions:
#col_list=['Avg. Area Income', 'Avg. Area House Age', 'Avg. Area Number of Rooms',
#       'Avg. Area Number of Bedrooms', 'Area Population', 'Price']
#df=pd.read_csv("USA_Housing.csv",names=col_list)
```

```
df.shape
```

```
(5000, 6)
```

```
df.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
0	79545.458574	5.682861	7.009188	4.09	23086.800503	1.059034e+06
1	79248.642455	6.002900	6.730821	3.09	40173.072174	1.505891e+06
2	61287.067179	5.865890	8.512727	5.13	36882.159400	1.058988e+06
3	63345.240046	7.188236	5.586729	3.26	34310.242831	1.260617e+06
4	59982.197226	5.040555	7.839388	4.23	26354.109472	6.309435e+05

```
df.columns
```

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

```
df.describe()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population	Price
count	5000.000000	5000.000000	5000.000000	5000.000000	5000.000000	5.000000e+03
mean	68583.108984	5.977222	6.987792	3.981330	36163.516039	1.232073e+06
std	10657.991214	0.991456	1.005833	1.234137	9925.650114	3.531176e+05
min	17796.631190	2.644304	3.236194	2.000000	172.610686	1.593866e+04
25%	61480.562388	5.322283	6.299250	3.140000	29403.928702	9.975771e+05
50%	68804.286404	5.970429	7.002902	4.050000	36199.406689	1.232669e+06
75%	75783.338666	6.650808	7.665871	4.490000	42861.290769	1.471210e+06
max	107701.748378	9.519088	10.759588	6.500000	69621.713378	2.469066e+06

```
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
```

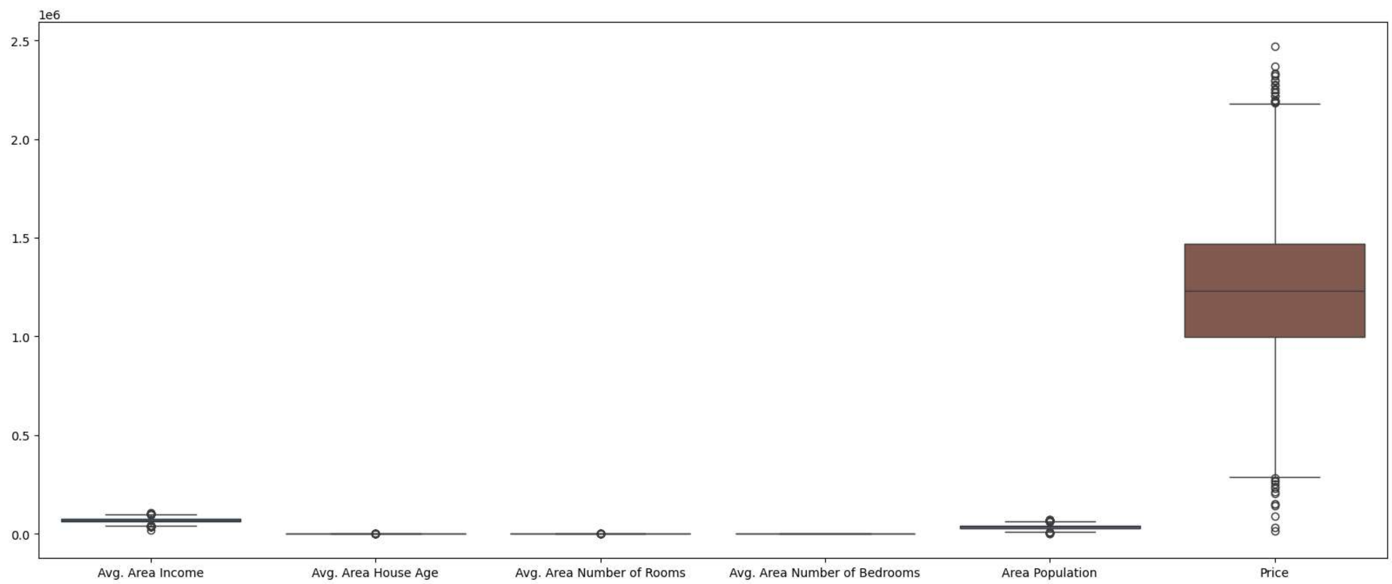
```
 2   Avg. Area Number of Rooms     5000 non-null   float64
 3   Avg. Area Number of Bedrooms  5000 non-null   float64
 4   Area Population              5000 non-null   float64
 5   Price                        5000 non-null   float64
dtypes: float64(6)
memory usage: 234.5 KB
```

```
#From the above interpretation i can say that
#No missing values
#Outliers in price
#No string datatype
#scaling required becoz area_population,area_income are in large range
```

```
df.isnull().sum()
```

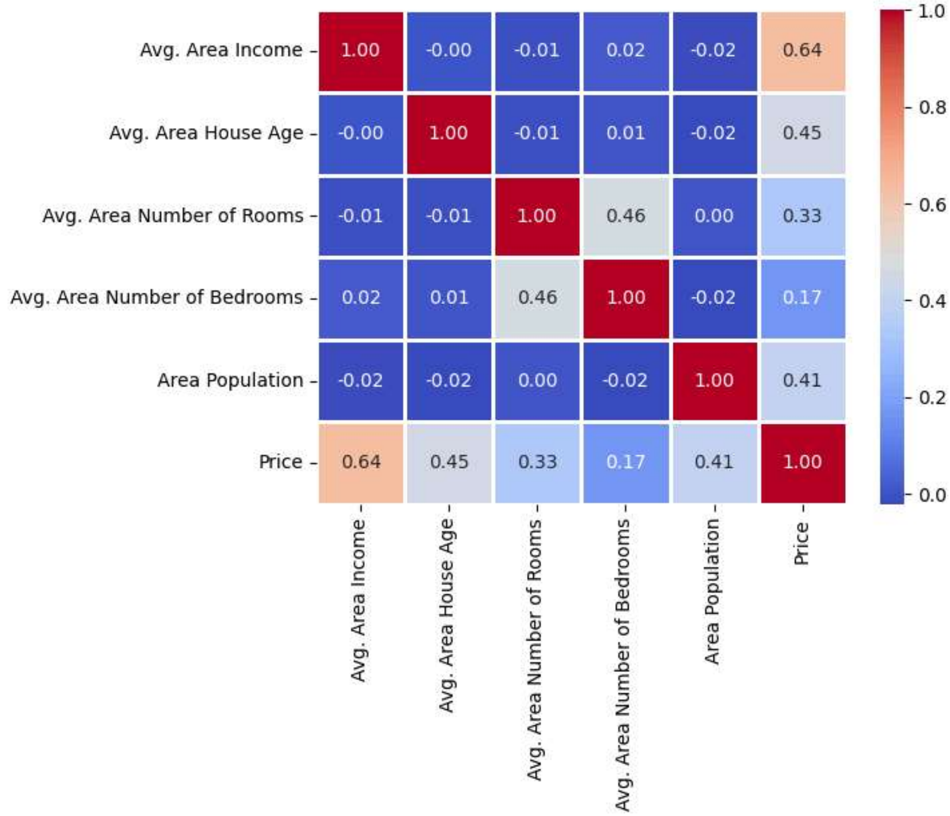
```
Avg. Area Income                0
Avg. Area House Age             0
Avg. Area Number of Rooms       0
Avg. Area Number of Bedrooms    0
Area Population                 0
Price                           0
dtype: int64
```

```
plt.figure(figsize =(20,8))
sns.boxplot(data=df,width=0.8)
plt.show()
```




```
#From the above plot we can say that there are outliers in the Area Income,Area Population,Price
```

```
sns.heatmap(df.corr(),annot=True,fmt='1.2f',annot_kws={'size':10},linewidth=1,cmap="coolwarm")
plt.show()
```




```
#After the basic EDA, before doing any manipulation, make sure to divide the data into train and test
```

```
train,test=train_test_split(df,test_size=0.2,random_state=1)
```

```
train.shape,test.shape
```

```
((4000, 6), (1000, 6))
```

```
#Treatment of Missing values
#Let's replace everything with Median.why not mean?
#if there are outliers,then impute the missing values with median,if not impute with mean.

#(df.isnull().sum()>0).index
#Taking out the names of the columns which have missing values
#missing_list=list(df.isnull().sum()[df.isnull().sum()>0].index)
#for col in missing_list:
#   median=train[col].median()
#   train[col].fillna(median,inplace=True)
#   test[col].fillna(median,inplace=True)
```

```
#Outlier treatment
df['Price'].min(),df['Price'].max()
```

```
(15938.6579232879, 2469065.5941747)
```

```
df['Price'].quantile(0.9)
```

```
1684620.9544020083
```

```
df['Price'].quantile(0.95)
```

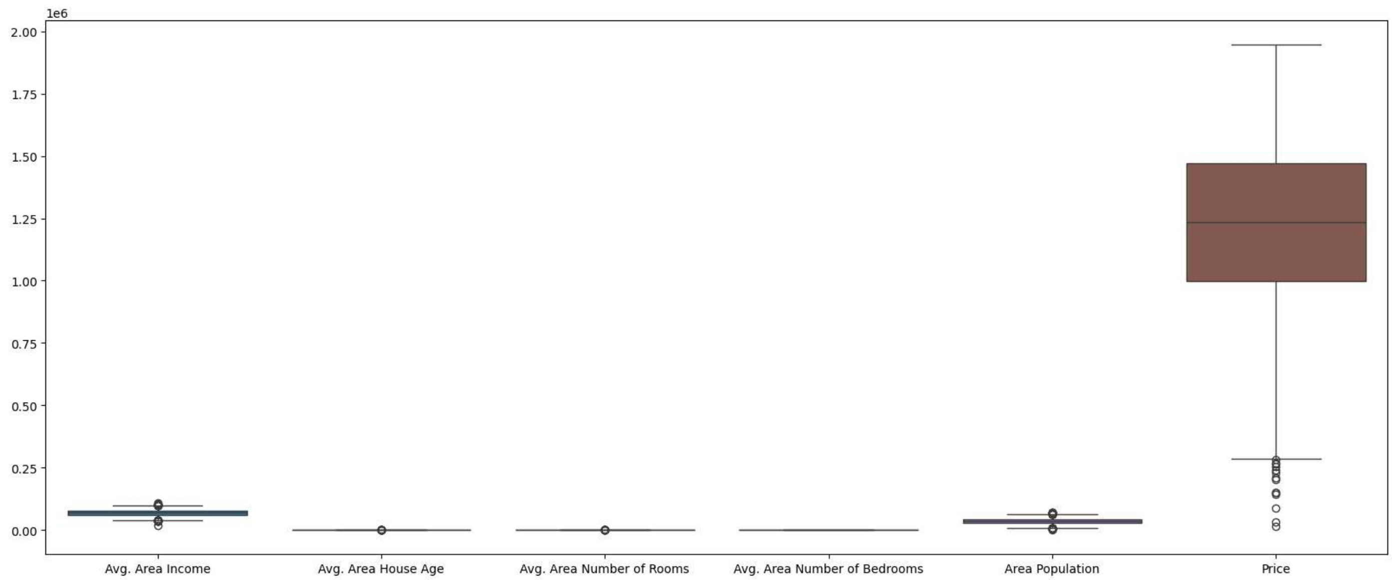
```
1813570.37914836
```

```
upper_limit=df['Price'].quantile(0.98)
upper_limit
```

```
1945653.0537544438
```

```
df['Price']=np.where(df['Price']>upper_limit,upper_limit,df['Price'])
```

```
plt.figure(figsize=(20,8))
sns.boxplot(data=df,width=0.8)
plt.show()
```




```
#If we observe in graph there are outliers in the Price which is my target ,but when i observe here there
#is no great difference between maximum and minimum value and 98% quantile value is also feasible.
#So,If there are potential outliers then cap the value at 98% quantile and replace the values which is greater
#than the cap value with the cap value.
#Otherwise that outliers may be useful insights for our data.So try to build a model in both scenarios
#Which gives best accuracy or least error choose that model.
```

```
#Just to keep it simple ,i'll not do Outliers treatment and removal of featurs based on Corr
```

```
#The range of features is different
#Let's Normalize all the features
```

```
#But i dont't want to Normalize my target
#therefore,i'll divide the data into features and target
```

```
#Formula of Normalization

#xnew=(xi-xmin)/(xmax-xmin)

#But if we have outliers, we should use Standardscaler instead =>why??

#xnew = xi-xmean/xstd
```

```
#Here i scaled both my features and target,if you want to scale only ur features,do scaling after completion of x_train and x_test
```

Start coding or generate with AI.

```
ss=StandardScaler()
train_scaled=ss.fit_transform(train)
test_scaled=ss.transform(test)
```

```
train_scaled=pd.DataFrame(train_scaled,columns=train.columns)
test_scaled=pd.DataFrame(test_scaled,columns=test.columns)
```

```
x_train=train_scaled.drop(['Price'],axis=1)
y_train=train_scaled['Price']
```

```
#ss=StandardScaler()
#x_train_scaled=ss.fit_transform(x_train)
#x_test_scaled=ss.transform(x_test)
#x_train_scaled=pd.DataFrame(x_train_scaled,columns=x_train.columns)
#x_test_scaled=pd.DataFrame(x_test_scaled,columns=x_test.columns)
```

```
x_train.head()
```

	Avg. Area Income	Avg. Area House Age	Avg. Area Number of Rooms	Avg. Area Number of Bedrooms	Area Population
0	-0.004591	0.340194	0.316584	-0.618734	-0.066558
1	0.004351	1.382290	1.800349	1.924455	0.776112
2	-1.012327	-0.355078	0.612859	-0.481046	-0.435302
3	0.022056	-1.123554	-0.072649	-0.562039	1.235740
4	-0.619123	0.716766	0.132689	-0.594436	1.001570

```
y_train.head()
```

```
0    0.293888
1    1.471125
2   -0.992134
3   -0.064477
4    0.409800
Name: Price, dtype: float64
```

```
x_train.shape,y_train.shape
```

```
((4000, 5), (4000,))
```

```
x_test=test_scaled.drop(['Price'],axis=1)
y_test=test_scaled['Price']
```

```
x_test.shape,y_test.shape
```

```
((1000, 5), (1000,))
```

```
#Building models
#First Model
lr=LinearRegression()
lr.fit(x_train,y_train)
pred=lr.predict(x_test)

error=mean_squared_error(y_test,pred)
```

```
print(error)
```

```
0.08627922454690747
```

```
lr.coef_
```

```
array([0.6576505 , 0.46974225, 0.34516644, 0.00544806, 0.42984357])
```

Start coding or generate with AI.

```
ls.coef_
```

```
array([0., 0., 0., 0., 0.])
```

Start coding or generate with AI.

```
#Cross-Validation
#Now ,we'll use train-val-test split instead of train-test split
```

```
linscores=cross_validate(knn,x_train,y_train,scoring="neg_mean_squared_error",cv=5,return_estimator=True)
```

```
linscores
```

```
{'fit_time': array([0.00813794, 0.00570035, 0.00695467, 0.00449061, 0.00476313]),
 'score_time': array([0.01209021, 0.01185918, 0.01132798, 0.01806164, 0.01129198]),
 'estimator': [KNeighborsRegressor(),
  KNeighborsRegressor(),
  KNeighborsRegressor(),
  KNeighborsRegressor(),
  KNeighborsRegressor()],
 'test_score': array([-0.15119307, -0.14977611, -0.13617985, -0.13068712, -0.1457427 ])}
```

```
linscores['test_score']
```

```
array([-0.15119307, -0.14977611, -0.13617985, -0.13068712, -0.1457427 ])
```

```
cv_score_of_model= -1*(linscores['test_score'].mean())
cv_score_of_model
```

```
0.142715772604106
```

```
#Choosing the Right value of K
score={}
for i in range(1,16):
  knncv=knn=KNeighborsRegressor(n_neighbors=i)
  linscores=cross_validate(knn,x_train,y_train,scoring="neg_mean_squared_error",cv=5,return_estimator=True)
  cv_score= -1*(linscores['test_score'].mean())
  score[i]=cv_score
```

```
score
```

```
{1: 0.23507005790263755,
 2: 0.17548191344850742,
 3: 0.15748454122894548,
 4: 0.14787041915636984,
 5: 0.142715772604106,
 6: 0.13921616877578602,
 7: 0.13715476181568878,
 8: 0.13498130264505798,
 9: 0.1338358999678834,
 10: 0.13231141478737468,
 11: 0.13119842735630943,
 12: 0.1320637220144672,
 13: 0.13170269956015987,
 14: 0.1317367193119779,
 15: 0.1324184704596564}
```

```
best_k=[key for (key,value) in score.items() if value==min(score.values())][0]
best_k
```

```
11
```

```
#Retrain the model with k=11
knnbest=KNeighborsRegressor(n_neighbors=11)
knnbest.fit(x_train,y_train)
pred=knnbest.predict(x_test)

error=mean_squared_error(y_test,pred)
print(error)
```

```
0.12879023831689335
```

Start coding or generate with AI.

Start coding or generate with AI.

Start coding or generate with AI.