In [66]:

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')
```

loading the dataset

In [67]:

```
house_df = pd.read_csv('C:/Users/lekshmi/Downloads/HousePricePrediction.xlsx - Sheet1.csv
```

In [68]:

```
house_df.shape
```

Out[68]:

```
(2919, 13)
```

2919 rows and 13 columns

In [69]:

```
house_df.head()
```

Out[69]:

	Id	MSSubClass	MSZoning	LotArea	LotConfig	BldgType	OverallCond	YearBuilt	YearRe
0	0	60	RL	8450	Inside	1Fam	5	2003	
1	1	20	RL	9600	FR2	1Fam	8	1976	
2	2	60	RL	11250	Inside	1Fam	5	2001	
3	3	70	RL	9550	Corner	1Fam	5	1915	
4	4	60	RL	14260	FR2	1Fam	5	2000	

Data Pre processing

In [70]:

```
house_df.drop_duplicates(inplace=True)
```

In [71]:

```
house_df.shape
```

Out[71]:

```
(2919, 13)
```

no duplicates are recorded

In [72]:

```
house_df.columns
```

Out[72]:

```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotArea', 'LotConfig', 'BldgType',
       'OverallCond', 'YearBuilt', 'YearRemodAdd', 'Exterior1st', 'BsmtFin
SF2',
       'TotalBsmtSF', 'SalePrice'],
      dtype='object')
```

In [73]:

```
house_df.drop(columns = ['Id'], inplace = True) # As 'id' column is unnecessary
```

checking for null values

In [74]:

```

house_df.isna().sum()
```

Out[74]:

```
MSSubClass         0
MSZoning           4
LotArea            0
LotConfig          0
BldgType           0
OverallCond        0
YearBuilt          0
YearRemodAdd       0
Exterior1st        1
BsmtFinSF2         1
TotalBsmtSF        1
SalePrice       1459
dtype: int64
```

some null values are recorded. Hence imputation technique is used for sales price column and for the rest, it is filled with zero

In [75]:

```
from sklearn.impute import SimpleImputer

imputer = SimpleImputer(strategy='mean')

imputer.fit(house_df[['SalePrice']])
```

Out[75]:

```
▾ SimpleImputer
SimpleImputer()
```

In [76]:

```
imputer.statistics_ # mean
```

Out[76]:

```
array([180921.19589041])
```

In [77]:

```
house_df['SalePrice'] = imputer.transform(house_df[['SalePrice']])
```

In [78]:

```
house_df = house_df.fillna(0)
```

In [79]:

```
house_df.isna().sum()
```

Out[79]:

```
MSSubClass      0
MSZoning        0
LotArea         0
LotConfig       0
BldgType        0
OverallCond     0
YearBuilt       0
YearRemodAdd    0
Exterior1st     0
BsmtFinSF2      0
TotalBsmtSF     0
SalePrice       0
dtype: int64
```

all the null values are handled

In [80]:

```
house_df.describe()
```

Out[80]:

	MSSubClass	LotArea	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF2
count	2919.000000	2919.000000	2919.000000	2919.000000	2919.000000	2919.000000
mean	57.137718	10168.114080	5.564577	1971.312778	1984.264474	49.565262
std	42.517628	7886.996359	1.113131	30.291442	20.894344	169.179104
min	20.000000	1300.000000	1.000000	1872.000000	1950.000000	0.000000
25%	20.000000	7478.000000	5.000000	1953.500000	1965.000000	0.000000
50%	50.000000	9453.000000	5.000000	1973.000000	1993.000000	0.000000
75%	70.000000	11570.000000	6.000000	2001.000000	2004.000000	0.000000
max	190.000000	215245.000000	9.000000	2010.000000	2010.000000	1526.000000

outlier detection

In [81]:

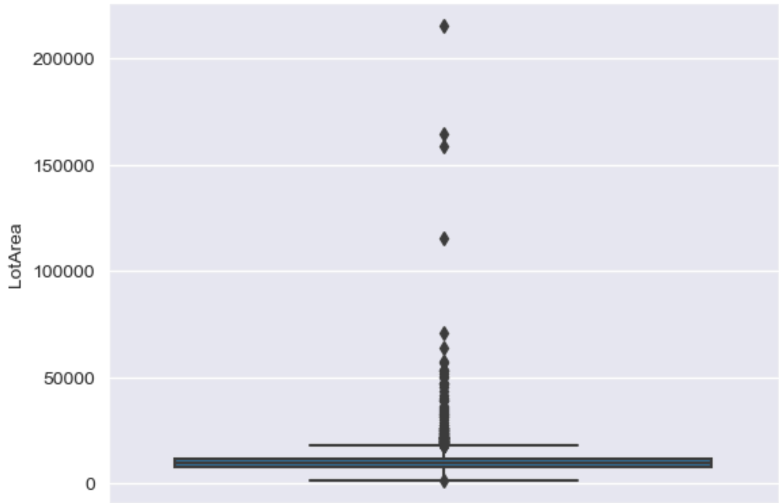
```
import matplotlib.pyplot as plt
import seaborn as sns
```

In [82]:

```
sns.set_style('darkgrid')
```

In [83]:

```
sns.boxplot(house_df, y = 'LotArea');
```



outliers are detected

outlier handling

In [84]:

```
import numpy as np

Q1 = np.percentile(house_df['LotArea'], 25, interpolation = 'midpoint') # first 25% of da
Q3 = np.percentile(house_df['LotArea'], 75, interpolation = 'midpoint') #75 % of data

IQR = Q3 - Q1
```

In [85]:

```
lowerBound = Q1 - 1.5 * IQR
upperBound = Q3 + 1.5 * IQR
```

In [86]:

```
df = house_df[(house_df.LotArea < upperBound) & (house_df.LotArea > lowerBound)]
```

In [87]:

```
df.shape
```

Out[87]:

```
(2791, 12)
```

In [88]:

```
df
```

Out[88]:

	MSSubClass	MSZoning	LotArea	LotConfig	BldgType	OverallCond	YearBuilt	YearRe
0	60	RL	8450	Inside	1Fam	5	2003	
1	20	RL	9600	FR2	1Fam	8	1976	
2	60	RL	11250	Inside	1Fam	5	2001	
3	70	RL	9550	Corner	1Fam	5	1915	
4	60	RL	14260	FR2	1Fam	5	2000	
...	...	...	...	...	...	...	...	
2913	160	RM	1526	Inside	Twnhs	5	1970	
2914	160	RM	1936	Inside	Twnhs	7	1970	
2915	160	RM	1894	Inside	TwnhsE	5	1970	
2917	85	RL	10441	Inside	1Fam	5	1992	
2918	60	RL	9627	Inside	1Fam	5	1993	

2791 rows × 12 columns

the outlier values have been removed

In [89]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 2791 entries, 0 to 2918
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   MSSubClass    2791 non-null   int64  
 1   MSZoning      2791 non-null   object 
 2   LotArea       2791 non-null   int64  
 3   LotConfig     2791 non-null   object 
 4   BldgType      2791 non-null   object 
 5   OverallCond   2791 non-null   int64  
 6   YearBuilt     2791 non-null   int64  
 7   YearRemodAdd  2791 non-null   int64  
 8   Exterior1st   2791 non-null   object 
 9   BsmtFinSF2    2791 non-null   float64
 10  TotalBsmtSF   2791 non-null   float64
 11  SalePrice     2791 non-null   float64
dtypes: float64(3), int64(5), object(4)
memory usage: 283.5+ KB
```

In [90]:

```
df.MSZoning.unique()
```

Out[90]:

```
array(['RL', 'RM', 'C (all)', 'FV', 'RH', 0], dtype=object)
```

as some columns contain categorical values and some numerical, we have to adjust the values

In [91]:

```
cat_cols = df.select_dtypes('object').columns.tolist()
```

In [92]:

```
cat_cols
```

Out[92]:

```
['MSZoning', 'LotConfig', 'BldgType', 'Exterior1st']
```

In [93]:

```
df[cat_cols] = df[cat_cols].astype(str)
```

OneHotEncoding

In [94]:

```
from sklearn.preprocessing import OneHotEncoder

encoder = OneHotEncoder(sparse= False, handle_unknown='ignore')

encoder.fit(df[cat_cols])
```

Out[94]:

```
▾ OneHotEncoder
OneHotEncoder(handle_unknown='ignore', sparse=False, sparse_output=False)
```

In [95]:

```
encoded_cols = encoder.get_feature_names_out(cat_cols)
```

In [96]:

```
encoded_cols
```

Out[96]:

```
array(['MSZoning_0', 'MSZoning_C (all)', 'MSZoning_FV', 'MSZoning_RH',
       'MSZoning_RL', 'MSZoning_RM', 'LotConfig_Corner',
       'LotConfig_CulDSac', 'LotConfig_FR2', 'LotConfig_FR3',
       'LotConfig_Inside', 'BldgType_1Fam', 'BldgType_2fmCon',
       'BldgType_Duplex', 'BldgType_Twnhs', 'BldgType_TwnhsE',
       'Exterior1st_AsbShng', 'Exterior1st_AsphShn',
       'Exterior1st_BrkComm', 'Exterior1st_BrkFace', 'Exterior1st_CBlock',
       'Exterior1st_CemntBd', 'Exterior1st_HdBoard',
       'Exterior1st_ImStucc', 'Exterior1st_MetalSd',
       'Exterior1st_Plywood', 'Exterior1st_Stone', 'Exterior1st_Stucco',
       'Exterior1st_VinylSd', 'Exterior1st_Wd Sdng',
       'Exterior1st_WdShing'], dtype=object)
```

In [97]:

```
df[encoded_cols] = encoder.transform(df[cat_cols])
```

In [98]:

```
df
```

Out[98]:

	MSSubClass	MSZoning	LotArea	LotConfig	BldgType	OverallCond	YearBuilt	YearRe
0	60	RL	8450	Inside	1Fam	5	2003	
1	20	RL	9600	FR2	1Fam	8	1976	
2	60	RL	11250	Inside	1Fam	5	2001	
3	70	RL	9550	Corner	1Fam	5	1915	
4	60	RL	14260	FR2	1Fam	5	2000	
...	...	...	...	...	...	...	...	
2913	160	RM	1526	Inside	Twnhs	5	1970	
2914	160	RM	1936	Inside	Twnhs	7	1970	
2915	160	RM	1894	Inside	TwnhsE	5	1970	
2917	85	RL	10441	Inside	1Fam	5	1992	
2918	60	RL	9627	Inside	1Fam	5	1993	

2791 rows × 43 columns

In [99]:

```
df.drop(columns=cat_cols, inplace=True)
```

In [100]:

```
df
```

Out[100]:

rd	Exterior1st_ImStucc	Exterior1st_MetalSd	Exterior1st_Plywood	Exterior1st_Stone	Exterior1st
.0	0.0	0.0	0.0	0.0	
.0	0.0	1.0	0.0	0.0	
.0	0.0	0.0	0.0	0.0	
.0	0.0	0.0	0.0	0.0	
.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	
.0	0.0	0.0	0.0	0.0	
.0	0.0	0.0	0.0	0.0	
.0	0.0	0.0	0.0	0.0	
.0	0.0	0.0	0.0	0.0	
.0	0.0	0.0	0.0	0.0	

Separation into independent and dependent variables

In [101]:

```
df.columns
```

Out[101]:

```
Index(['MSSubClass', 'LotArea', 'OverallCond', 'YearBuilt', 'YearRemodAd
d',
       'BsmtFinSF2', 'TotalBsmtSF', 'SalePrice', 'MSZoning_0',
       'MSZoning_C (all)', 'MSZoning_FV', 'MSZoning_RH', 'MSZoning_RL',
       'MSZoning_RM', 'LotConfig_Corner', 'LotConfig_CulDSac', 'LotConfig_
FR2',
       'LotConfig_FR3', 'LotConfig_Inside', 'BldgType_1Fam', 'BldgType_2fm
Con',
       'BldgType_Duplex', 'BldgType_Twnhs', 'BldgType_TwnhsE',
       'Exterior1st_AsbShng', 'Exterior1st_AsphShn', 'Exterior1st_BrkCom
m',
       'Exterior1st_BrkFace', 'Exterior1st_CBlock', 'Exterior1st_CemntBd',
       'Exterior1st_HdBoard', 'Exterior1st_ImStucc', 'Exterior1st_MetalS
d',
       'Exterior1st_Plywood', 'Exterior1st_Stone', 'Exterior1st_Stucco',
       'Exterior1st_VinylSd', 'Exterior1st_Wd Sdng', 'Exterior1st_WdShin
g'],
      dtype='object')
```

In [121]:

```
X = df.drop(columns = 'SalePrice')
y = df['SalePrice']
```

In [122]:

```
X
```

Out[122]:

	MSSubClass	LotArea	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF2	TotalBsmtS
0	60	8450	5	2003	2003	0.0	856
1	20	9600	8	1976	1976	0.0	1262
2	60	11250	5	2001	2002	0.0	920
3	70	9550	5	1915	1970	0.0	756
4	60	14260	5	2000	2000	0.0	1145
...	...	...	...	...	...	...	
2913	160	1526	5	1970	1970	0.0	546
2914	160	1936	7	1970	1970	0.0	546
2915	160	1894	5	1970	1970	0.0	546
2917	85	10441	5	1992	1992	0.0	912
2918	60	9627	5	1993	1994	0.0	996

2791 rows × 38 columns

as the values in different columns have different scales, it is necessary to bring them to a standard scaling

In [104]:

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

scaler.fit(X)
```

Out[104]:

```
▾ MinMaxScaler
MinMaxScaler()
```

In [105]:

```
X[:] = scaler.transform(X)
```

In [106]:

```
X
```

Out[106]:

	MSSubClass	LotArea	OverallCond	YearBuilt	YearRemodAdd	BsmtFinSF2	TotalBsmt
0	0.235294	0.430838	0.500	0.949275	0.883333	0.0	0.2669
1	0.000000	0.501821	0.875	0.753623	0.433333	0.0	0.3936
2	0.235294	0.603666	0.500	0.934783	0.866667	0.0	0.2869
3	0.294118	0.498735	0.500	0.311594	0.333333	0.0	0.2358
4	0.235294	0.789457	0.500	0.927536	0.833333	0.0	0.3571
...	...	...	...	...	...	...	
2913	0.823529	0.003457	0.500	0.710145	0.333333	0.0	0.1703
2914	0.823529	0.028764	0.750	0.710145	0.333333	0.0	0.1703
2915	0.823529	0.026171	0.500	0.710145	0.333333	0.0	0.1703
2917	0.382353	0.553731	0.500	0.869565	0.700000	0.0	0.2844
2918	0.235294	0.503487	0.500	0.876812	0.733333	0.0	0.3106

2791 rows × 38 columns

Splitting into training and test data

In [203]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=10
```

In [204]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[204]:

```
((2232, 38), (559, 38), (2232,), (559,))
```

importing linear regression model

In [205]:

```
from sklearn.linear_model import LinearRegression

model = LinearRegression()

model.fit(X_train, y_train)
```

Out[205]:

```
▾ LinearRegression
LinearRegression()
```

In [206]:

```
y_pred = model.predict(X_test)
```

In [207]:

```
y_test[:5]
```

Out[207]:

```
2272    180921.19589
1060    213500.00000
2182    180921.19589
1268    381000.00000
2782    180921.19589
Name: SalePrice, dtype: float64
```

In [208]:

```
y_pred[:5]
```

Out[208]:

```
array([182604.33207739, 189636.78144702, 202395.73183661, 196977.86382891,
       139023.10089291])
```

evaluation of model

In [209]:

```
from sklearn.metrics import mean_absolute_error
```

In [210]:

```
mean_absolute_error(y_test, y_pred)
```

Out[210]:

```
32690.00634743156
```

regularisation of model to improve the mean absolute error

In [259]:

```
from sklearn.linear_model import Lasso

lasso_reg = Lasso(alpha=20, max_iter=100, tol = 0.1)

lasso_reg.fit(X_train, y_train)
```

Out[259]:

```
▾ Lasso
Lasso(alpha=20, max_iter=200, tol=0.1)
```

In [260]:

```
lasso_pred = lasso_reg.predict(X_test)
```

In [261]:

```
mean_absolute_error(y_test, lasso_pred)
```

Out[261]:

32573.215872117675

In [265]:

```
from sklearn.linear_model import Ridge

ridge_reg = Ridge(alpha=50, max_iter=100, tol = 0.1)

ridge_reg.fit(X_train, y_train)
```

Out[265]:

```
▾ Ridge
Ridge(alpha=50, max_iter=100, tol=0.1)
```

In [266]:

```
ridge_pred = ridge_reg.predict(X_test)
```

In [267]:

```
mean_absolute_error(y_test, ridge_pred)
```

Out[267]:

32784.054095458036

In [ ]:

```
```

In [ ]:

```
```