Chennai House Price Prediction

dataset link- https://www.kaggle.com/code/kunwarakash/chennai-house-price-prediction/input

In [180…

```
# importing necessary libraries

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

from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Lasso,LassoCV,Ridge,RidgeCV,ElasticNet,ElasticNetCV
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score,mean_absolute_error
```

In [2]:

```
os.chdir('F:\Data Science\Chennai House Price Predicton')
```

In [3]:

```
# read csv file

df = pd.read_csv("Chennai houseing sale.csv")
```

In [4]:

```
# top 5 rows

df.head()
```

Out[4]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	...	U
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes	...	
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No	...	
2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes	...	
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No	...	
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes	...	

5 rows × 22 columns

In [5]:

```
#shape of data

df.shape
```

Out[5]:

```
(7109, 22)
```

In [6]:

```
# Checking basic dataset information

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7109 entries, 0 to 7108
Data columns (total 22 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   PRT_ID         7109 non-null   object
 1   AREA           7109 non-null   object
 2   INT_SQFT       7109 non-null   int64
 3   DATE_SALE      7109 non-null   object
 4   DIST_MAINROAD  7109 non-null   int64
 5   N_BEDROOM      7108 non-null   float64
 6   N_BATHROOM     7104 non-null   float64
 7   N_ROOM         7109 non-null   int64
 8   SALE_COND      7109 non-null   object
 9   PARK_FACIL     7109 non-null   object
 10  DATE_BUILD     7109 non-null   object
 11  BUILDTYPE      7109 non-null   object
 12  UTILITY_AVAIL  7109 non-null   object
 13  STREET         7109 non-null   object
 14  MZZONE         7109 non-null   object
 15  QS_ROOMS       7109 non-null   float64
 16  QS_BATHROOM    7109 non-null   float64
 17  QS_BEDROOM     7109 non-null   float64
 18  QS_OVERALL     7061 non-null   float64
 19  REG_FEE        7109 non-null   int64
 20  COMMIS         7109 non-null   int64
 21  SALES_PRICE    7109 non-null   int64
dtypes: float64(6), int64(6), object(10)
memory usage: 1.2+ MB
```

In [7]:
```
# column names in dataset


df.columns
```

Out[7]:
```
Index(['PRT_ID', 'AREA', 'INT_SQFT', 'DATE_SALE', 'DIST_MAINROAD', 'N_BEDROOM',
       'N_BATHROOM', 'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'DATE_BUILD',
       'BUILDTYPE', 'UTILITY_AVAIL', 'STREET', 'MZZONE', 'QS_ROOMS',
       'QS_BATHROOM', 'QS_BEDROOM', 'QS_OVERALL', 'REG_FEE', 'COMMIS',
       'SALES_PRICE'],
      dtype='object')
```

About Data

- PRT_ID- Id of house
- AREA - In which area house is located in Chennai
- INT_SQFT- Area in sqft
- DATE_SALE- When house was sold
- DIST_MAINROAD- Distance of house from main road
- N_BEDROOM- Number of Bedrooms
- N_BATHROOM- Number of Bathrooms
- N_ROOM- Number of Rooms
- SALE_COND- Sale condition
- PARK_FACIL- Is parking available or not
- DATE_BUILD- Date house was built
- BUILDTYPE- Purpose of house
- UTILITY_AVAIL- Facilities available there
- STREET- How is street outside that house
- MZZONE- Which zone it is in
- QS_ROOMS- It is masked data
- QS_BATHROOM- It is masked data
- QS_BEDROOM- It is masked data
- QS_OVERALL- It is masked data
- REG_FEE- Registration fee after sales
- COMMIS- Commission fee after sales
- SALES_PRICE- Sale price of house

1. How many numerical columns and catgorical columns are there in data ?

In [8]:
```
# All numeric columns are :


print(df.select_dtypes(include=np.number).columns)
```

```
print("No. of Numerical columns are :",len(df.select_dtypes(include=np.number).columns))
```

```
Index(['INT_SQFT', 'DIST_MAINROAD', 'N_BEDROOM', 'N_BATHROOM', 'N_ROOM',
       'QS_ROOMS', 'QS_BATHROOM', 'QS_BEDROOM', 'QS_OVERALL', 'REG_FEE',
       'COMMIS', 'SALES_PRICE'],
      dtype='object')
No. of Numerical columns are : 12
```

In [9]:
```
# All categorical columns are :

print(df.select_dtypes(include=object).columns)
print(" No. of categorical columns are :",len(df.select_dtypes(include=object).columns))
```

```
Index(['PRT_ID', 'AREA', 'DATE_SALE', 'SALE_COND', 'PARK_FACIL', 'DATE_BUILD',
       'BUILDTYPE', 'UTILITY_AVAIL', 'STREET', 'MZZONE'],
      dtype='object')
 No. of categorical columns are : 10
```

In [10]:
```
# 'MZZONE' Value_counts

df['MZZONE'].value_counts()
```

Out[10]:
```
RL    1858
RH    1822
RM    1817
C      550
A      537
I      525
Name: MZZONE, dtype: int64
```

DATA CLEANING

In [11]:
```
# as per general understanding dropping non-relevant columns from data- Masked data(removing)

df = df.drop(['QS_ROOMS','QS_BATHROOM', 'QS_BEDROOM', 'QS_OVERALL'], axis=1)
```

In [12]:
```
# checking dataset after removing columns

df.head()
```

Out[12]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	DATE
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes	15
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No	22
2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes	09
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No	18
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes	13

2. which column has missing values present ?

In [13]:
```
# checking null values in dataset

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

Out[13]:
```
PRT_ID           0
AREA             0
INT_SQFT         0
DATE_SALE        0
DIST_MAINROAD    0
N_BEDROOM        1
N_BATHROOM       5
N_ROOM           0
SALE_COND        0
PARK_FACIL       0
DATE_BUILD       0
BUILDTYPE        0
UTILITY_AVAIL    0
STREET           0
MZZONE           0
REG_FEE          0
COMMIS           0
SALES_PRICE      0
dtype: int64
```

N_BEDROOM, N_BATHROOM columns has missing values

In [14]:
```
#checking no. of bathrooms in each house


df['N_BATHROOM'].value_counts()
```

Out[14]:
```
1.0    5589
2.0    1515
Name: N_BATHROOM, dtype: int64
```

In [15]:
```
#filling Number of bathroom null value using mode value of bathrooms


df['N_BATHROOM'] = df['N_BATHROOM'].fillna(df['N_BATHROOM'].mode()[0])
```

In [16]:
```
#checking number of bedrooms available in houses


df['N_BEDROOM'].value_counts()
```

Out[16]:
```
1.0    3795
2.0    2352
3.0     707
4.0     254
Name: N_BEDROOM, dtype: int64
```

In [17]:
```
# dropping null Number of bedroom row


df.dropna(inplace= True)
```

In [18]:
```
# checking null value in dataset after null values operation


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

Out[18]:
```
PRT_ID           0
AREA             0
INT_SQFT         0
DATE_SALE        0
DIST_MAINROAD    0
N_BEDROOM        0
N_BATHROOM       0
N_ROOM           0
SALE_COND        0
PARK_FACIL       0
DATE_BUILD       0
BUILDTYPE        0
UTILITY_AVAIL    0
STREET           0
MZZONE           0
REG_FEE          0
COMMIS           0
SALES_PRICE      0
dtype: int64
```

In [19]:
```
#checking dataset after
```

```
df.head()
```

Out[19]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	DATE
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes	15-
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No	22-
2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes	09-
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No	18-
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes	13-

#

Feature Engineering

In [20]:
```
#adding Commision , registration fee and sales price column to create 'Total sales price column'

df['TOTAL_SALES_PRICE'] = df['SALES_PRICE'] + df['COMMIS'] + df['REG_FEE']
```

In [21]:
```
#checking dataset now

df.head()
```

Out[21]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	DATE
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes	15-
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No	22-
2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes	09-
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No	18-
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes	13-

In [22]:
```
#checking unique values in INT_SQFTcolumn

df['INT_SQFT'].value_counts()
```

Out[22]:
```
1781    18
1538    15
1505    13
1514    13
1655    12
        ..
559      1
2479     1
1330     1
880      1
598      1
Name: INT_SQFT, Length: 1699, dtype: int64
```

In [23]:
```
df['DATE_BUILD']
```

Out[23]:
```
0       15-05-1967
1       22-12-1995
2       09-02-1992
3       18-03-1988
4       13-10-1979
           ...
7104    15-01-1962
7105    11-04-1995
7106    01-09-1978
7107    11-08-1977
7108    24-07-1961
Name: DATE_BUILD, Length: 7108, dtype: object
```

In [24]:
```
# function to create  BUILD YEAR Column of dataset from DATE BUILD column

def convert(x):
```

```
    num = x.split('-')[2]
    return num
```

In [25]:
```
#creating copy of dataset

df2 = df.copy()
```

In [26]:
```
# Creatig 'BUILD YEAR' Column from 'DATE BUILD'

df2['BUILD_YEAR'] = df2['DATE_BUILD'].apply(convert)
```

In [27]:
```
# Creatig 'SALE_YEAR' Column from 'DATE_SALE'

df2['SALE_YEAR'] = df['DATE_SALE'].apply(convert)
```

In [28]:
```
df2.head()
```

Out[28]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	...	B
0	P03210	Karapakkam	1004	04-05-2011	131	1.0	1.0	3	AbNormal	Yes	...	
1	P09411	Anna Nagar	1986	19-12-2006	26	2.0	1.0	5	AbNormal	No	...	
2	P01812	Adyar	909	04-02-2012	70	1.0	1.0	3	AbNormal	Yes	...	
3	P05346	Velachery	1855	13-03-2010	14	3.0	2.0	5	Family	No	...	
4	P06210	Karapakkam	1226	05-10-2009	84	1.0	1.0	3	AbNormal	Yes	...	

5 rows × 21 columns

In [29]:
```
# Converting DATE sale in datetime format

df2['DATE_SALE'] = pd.to_datetime(df2['DATE_SALE'])
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_9156\2330758434.py:3: UserWarning: Parsing dates in DD/MM/YYYY format when dayfirst=False (the default) was specified. This may lead to inconsistently parsed dates! Specify a format to ensure consistent parsing.
  df2['DATE_SALE'] = pd.to_datetime(df2['DATE_SALE'])
```

In [30]:
```
#checking datatype of df2
df2.dtypes
```

Out[30]:
```
PRT_ID                object
AREA                  object
INT_SQFT               int64
DATE_SALE     datetime64[ns]
DIST_MAINROAD          int64
N_BEDROOM            float64
N_BATHROOM           float64
N_ROOM                 int64
SALE_COND             object
PARK_FACIL            object
DATE_BUILD            object
BUILDTYPE             object
UTILITY_AVAIL         object
STREET                object
MZZONE                object
REG_FEE                int64
COMMIS                 int64
SALES_PRICE            int64
TOTAL_SALES_PRICE      int64
BUILD_YEAR            object
SALE_YEAR             object
dtype: object
```

In [31]:
```
#checking details of datset

df2.describe(include='all')
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_9156\1016192461.py:3: FutureWarning: Treating datetime data as categor
ical rather than numeric in `.describe` is deprecated and will be removed in a future version of pandas. Specif
y `datetime_is_numeric=True` to silence this warning and adopt the future behavior now.
  df2.describe(include='all')
```

Out[31]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACI
count	7108	7108	7108.000000	7108	7108.000000	7108.000000	7108.000000	7108.000000	7108	710
unique	7108	17	NaN	2798	NaN	NaN	NaN	NaN	9	
top	P03210	Chrompet	NaN	2009-06-10 00:00:00	NaN	NaN	NaN	NaN	AdjLand	Ye
freq	1	1681	NaN	12	NaN	NaN	NaN	NaN	1433	358
first	NaN	NaN	NaN	2004-01-02 00:00:00	NaN	NaN	NaN	NaN	NaN	Nal
last	NaN	NaN	NaN	2015-12-02 00:00:00	NaN	NaN	NaN	NaN	NaN	Nal
mean	NaN	NaN	1382.048537	NaN	99.591728	1.637029	1.213140	3.688661	NaN	Nal
std	NaN	NaN	457.438429	NaN	57.399027	0.802902	0.409555	1.019164	NaN	Nal
min	NaN	NaN	500.000000	NaN	0.000000	1.000000	1.000000	2.000000	NaN	Nal
25%	NaN	NaN	993.000000	NaN	50.000000	1.000000	1.000000	3.000000	NaN	Nal
50%	NaN	NaN	1373.000000	NaN	99.000000	1.000000	1.000000	4.000000	NaN	Nal
75%	NaN	NaN	1744.000000	NaN	148.000000	2.000000	1.000000	4.000000	NaN	Nal
max	NaN	NaN	2500.000000	NaN	200.000000	4.000000	2.000000	6.000000	NaN	Nal

13 rows × 21 columns

3. Are there misspelled data ?

In [32]:
```
#checking uniqueness in SALE_COND column

df2['SALE_COND'].unique()
```

Out[32]:
```
array(['AbNormal', 'Family', 'Partial', 'AdjLand', 'Normal Sale',
       'Ab Normal', 'Partiall', 'Adj Land', 'PartiaLl'], dtype=object)
```

In [33]:
```
# as Abnormal = Ab normal, Partial = PartialL and adjLAND = Adj Land
# we will clean this column by merging duplicate values

df2['SALE_COND'] = df2['SALE_COND'].replace({'Ab Normal': 'AbNormal', 'Partiall':
'Partial','PartiaLl':'Partial','Adj Land':'AdjLand'})
```

In [34]:
```
df2['SALE_COND'].unique()
```

Out[34]:
```
array(['AbNormal', 'Family', 'Partial', 'AdjLand', 'Normal Sale'],
      dtype=object)
```

In [35]:
```
# Checking new df2

df2.head()
```

Out[35]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	...	B
0	P03210	Karapakkam	1004	2011-04-05	131	1.0	1.0	3	AbNormal	Yes	...	
1	P09411	Anna Nagar	1986	2006-12-19	26	2.0	1.0	5	AbNormal	No	...	
2	P01812	Adyar	909	2012-04-02	70	1.0	1.0	3	AbNormal	Yes	...	
3	P05346	Velachery	1855	2010-03-13	14	3.0	2.0	5	Family	No	...	
4	P06210	Karapakkam	1226	2009-05-10	84	1.0	1.0	3	AbNormal	Yes	...	

5 rows × 21 columns

In [36]:
```
#Checking unique values

df2['PARK_FACIL'].value_counts()
```

Out[36]:
```
Yes    3587
No     3519
Noo       2
Name: PARK_FACIL, dtype: int64
```

In [37]:
```
# replacing 'Noo' by No in dataframe

df2['PARK_FACIL'] = df2['PARK_FACIL'].replace('Noo','No')
```

In [38]:
```
df2['PARK_FACIL'].value_counts()
```

Out[38]:
```
Yes    3587
No     3521
Name: PARK_FACIL, dtype: int64
```

In [39]:
```
#Checking unique values

df2['BUILDTYPE'].value_counts()
```

Out[39]:
```
House         2443
Commercial    2325
Others        2310
Other           26
Comercial        4
Name: BUILDTYPE, dtype: int64
```

In [40]:
```
# replacing comercial by Commercial and Other by Others

df2['BUILDTYPE'] = df2['BUILDTYPE'].replace({'Comercial':'Commercial','Other':'Others'})
```

In [41]:
```
df2['BUILDTYPE'].value_counts()
```

Out[41]:
```
House         2443
Others        2336
Commercial    2329
Name: BUILDTYPE, dtype: int64
```

In [42]:
```
# creating copy of dataframe df3

df3 = df2.copy()
```

In [43]:
```
df3.head()
```

Out[43]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	...	B
0	P03210	Karapakkam	1004	2011-04-05	131	1.0	1.0	3	AbNormal	Yes	...	(
1	P09411	Anna Nagar	1986	2006-12-19	26	2.0	1.0	5	AbNormal	No	...	(
2	P01812	Adyar	909	2012-04-02	70	1.0	1.0	3	AbNormal	Yes	...	(
3	P05346	Velachery	1855	2010-03-13	14	3.0	2.0	5	Family	No	...	
4	P06210	Karapakkam	1226	2009-05-10	84	1.0	1.0	3	AbNormal	Yes	...	

5 rows × 21 columns

In [44]:
```
#Checking unique values

df3['UTILITY_AVAIL'].value_counts()
```

Out[44]:
```
AllPub     1886
NoSeWa     1871
NoSewr     1828
ELO        1522
All Pub       1
Name: UTILITY_AVAIL, dtype: int64
```

In [45]:
```
# replacing All Pub  by AllPub, NoSewr by NoSewa

df3['UTILITY'] = df3['UTILITY_AVAIL'].replace({'All Pub':'AllPub','NoSewr
':'NoSewa','NoSeWa':'NoSewa'})
```

In [46]:
```
#Checking unique values

df3['UTILITY'].value_counts()
```

Out[46]:
```
NoSewa    3699
AllPub    1887
ELO          1522
Name: UTILITY, dtype: int64
```

In [47]:
```
# Checking unique values in STREET Column

df3['STREET'].unique()
```

Out[47]:
```
array(['Paved', 'Gravel', 'No Access', 'Pavd', 'NoAccess'], dtype=object)
```

In [48]:
```
# removing duplicate names in street columns

df3['STREET'] = df3['STREET'].replace({'Pavd':'Paved','No Access':'NoAccess'})
```

In [49]:
```
# Checking unique values in STREET Column after operation

df3['STREET'].unique()
```

Out[49]:
```
array(['Paved', 'Gravel', 'NoAccess'], dtype=object)
```

In [50]:
```
# Checking unique values

df3['MZZONE'].value_counts()
```

Out[50]:
```
RL    1858
RH    1822
RM    1816
C      550
A      537
I      525
Name: MZZONE, dtype: int64
```

In [51]:
```
# checking unique values in AREA column

df3['AREA'].unique()
```

Out[51]:
```
array(['Karapakkam', 'Anna Nagar', 'Adyar', 'Velachery', 'Chrompet',
       'KK Nagar', 'TNagar', 'T Nagar', 'Chrompt', 'Chrmpet', 'Karapakam',
       'Ana Nagar', 'Chormpet', 'Adyr', 'Velchery', 'Ann Nagar',
       'KKNagar'], dtype=object)
```

In [52]:
```
# removing duplicate names in Area columns

df3['AREA'] = df3['AREA'].replace({'Karapakam':'Karapakkam','Ana Nagar':'Anna Nagar','Ann
Nagar':'Anna Nagar',

'Adyr':'Adyar','Velchery':'Velachery','Chormpet':'Chrompet','Chrompt':'Chrompet'
                                    ,'Chrmpet':'Chrompet','KKNagar':'KK Nagar','TNagar':'T
Nagar'})
```

In [53]:
```
# checking unique values in AREA column

df3['AREA'].unique()
```

Out[53]:
```
array(['Karapakkam', 'Anna Nagar', 'Adyar', 'Velachery', 'Chrompet',
       'KK Nagar', 'T Nagar'], dtype=object)
```

In [54]:
```
df3.head()
```

Out[54]:

	PRT_ID	AREA	INT_SQFT	DATE_SALE	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	...	U
0	P03210	Karapakkam	1004	2011-04-05	131	1.0	1.0	3	AbNormal	Yes	...	
1	P09411	Anna Nagar	1986	2006-12-19	26	2.0	1.0	5	AbNormal	No	...	
2	P01812	Adyar	909	2012-04-02	70	1.0	1.0	3	AbNormal	Yes	...	
3	P05346	Velachery	1855	2010-03-13	14	3.0	2.0	5	Family	No	...	
4	P06210	Karapakkam	1226	2009-05-10	84	1.0	1.0	3	AbNormal	Yes	...	

5 rows × 22 columns

In [55]:
```
df3.columns
```

Out[55]:
```
Index(['PRT_ID', 'AREA', 'INT_SQFT', 'DATE_SALE', 'DIST_MAINROAD', 'N_BEDROOM',
       'N_BATHROOM', 'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'DATE_BUILD',
       'BUILDTYPE', 'UTILITY_AVAIL', 'STREET', 'MZZONE', 'REG_FEE', 'COMMIS',
       'SALES_PRICE', 'TOTAL_SALES_PRICE', 'BUILD YEAR', 'SALE_YEAR',
       'UTILITY'],
      dtype='object')
```

In [56]:
```
# Dropping excess columns

df3.drop(columns=['UTILITY_AVAIL','DATE_BUILD','DATE_SALE'],axis=1,inplace = True)
```

In [57]:
```
# creating new dataframe

df4 = df3.copy()
```

In [58]:
```
# Now dropping more unnecessary columns

df4.drop(['PRT_ID'],axis = 1, inplace =True)
```

In [59]:
```
df4.head()
```

Out[59]:

	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	STREET	MZZ
0	Karapakkam	1004	131	1.0	1.0	3	AbNormal	Yes	Commercial	Paved	
1	Anna Nagar	1986	26	2.0	1.0	5	AbNormal	No	Commercial	Gravel	
2	Adyar	909	70	1.0	1.0	3	AbNormal	Yes	Commercial	Gravel	
3	Velachery	1855	14	3.0	2.0	5	Family	No	Others	Paved	
4	Karapakkam	1226	84	1.0	1.0	3	AbNormal	Yes	Others	Gravel	

In [60]:
```
# checking unique values

df4['N_ROOM'].value_counts()
```

Out[60]:
```
4    2562
3    2125
5    1246
2     921
6     254
Name: N_ROOM, dtype: int64
```

In [61]:
```
# checking unique values

df4['N_BEDROOM'].value_counts()
```

Out[61]:
```
1.0    3795
2.0    2352
3.0     707
4.0     254
Name: N_BEDROOM, dtype: int64
```

In [62]:
```
# checking unique values

df4['N_BATHROOM'].value_counts()
```

Out[62]:
```
1.0    5593
2.0    1515
Name: N_BATHROOM, dtype: int64
```

In [63]:
```
# changing dtype of columns

df4[['N_BATHROOM', 'N_BEDROOM']] = df4[['N_BATHROOM', 'N_BEDROOM']].astype(int)
```

In [64]:
```
df4.columns
```

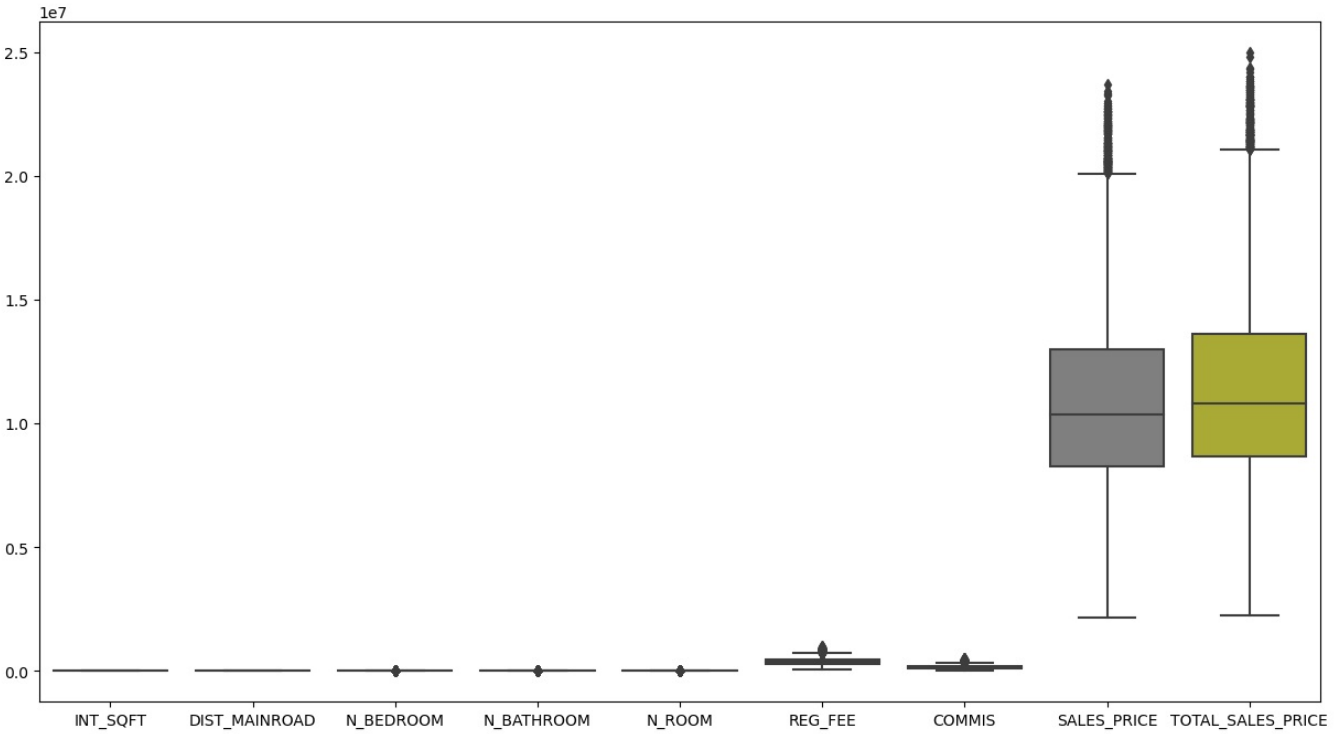
Out[64]:
```
Index(['AREA', 'INT_SQFT', 'DIST_MAINROAD', 'N_BEDROOM', 'N_BATHROOM',
       'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'BUILDTYPE', 'STREET', 'MZZONE',
       'REG_FEE', 'COMMIS', 'SALES_PRICE', 'TOTAL_SALES_PRICE', 'BUILD YEAR',
       'SALE_YEAR', 'UTILITY'],
      dtype='object')
```

4. Are there outliers in dataset ? if yes, which column has outlier ?

In [65]:
```
# Methods to find outliers
'''
1. Z-SCORE
2. BOX-PLOT
3. IQR
4. SCATTER PLOTS '''
```

Out[65]:
```
' \n1. Z-SCORE\n2. BOX-PLOT\n3. IQR\n4. SCATTER PLOTS '
```

In [66]:
```
# outliers finding using boxplot

plt.figure(figsize =(15,8))
sns.boxplot(data = df4)
plt.show()
```



It seems there are some outliers present in SALES_PRICE Column

In [67]:
```
# For understanding property rate we will create one column price/sqft

df4['PRICE_PER_SQ_FT'] = df4['TOTAL_SALES_PRICE']/df4['INT_SQFT']
```

In [68]:
```
# maximum price per square foot in chennai

df4['PRICE_PER_SQ_FT'].max()
```

Out[68]:
```
19452.509493670885
```

In [69]:
```
# minimum price per square ffot in chennai

df4['PRICE_PER_SQ_FT'].min()
```

Out[69]:
```
3138.9220779220777
```

it shows we have rate as per market "if you cross check in google"

In [70]:
```
## finding outliers in TOTAL _SALES_PRICE COLUMN using IQR Method

q1,q3 = np.percentile(df4['TOTAL_SALES_PRICE'],[25,75])
IQR = q3-q1
lower_val = q1 - (1.5*IQR)
upper_val = q3 + (1.5*IQR)
```

In [71]:
```
print('lower value :',lower_val)
print('upper value :',upper_val)
```

```
lower value : 1192439.0
upper value : 21067637.0
```

In [72]:
```
# Checing no. of outliers present above Upper bound

df4['TOTAL_SALES_PRICE'][df4['TOTAL_SALES_PRICE']>upper_val].count()
```

Out[72]:
```
227
```

In [73]:
```
# Checing no. of outliers present below lower bound

df4['TOTAL_SALES_PRICE'][df4['TOTAL_SALES_PRICE']<lower_val].count()
```

Out[73]:
```
0
```

In [74]:
```
df4.shape
```

Out[74]:
```
(7108, 19)
```

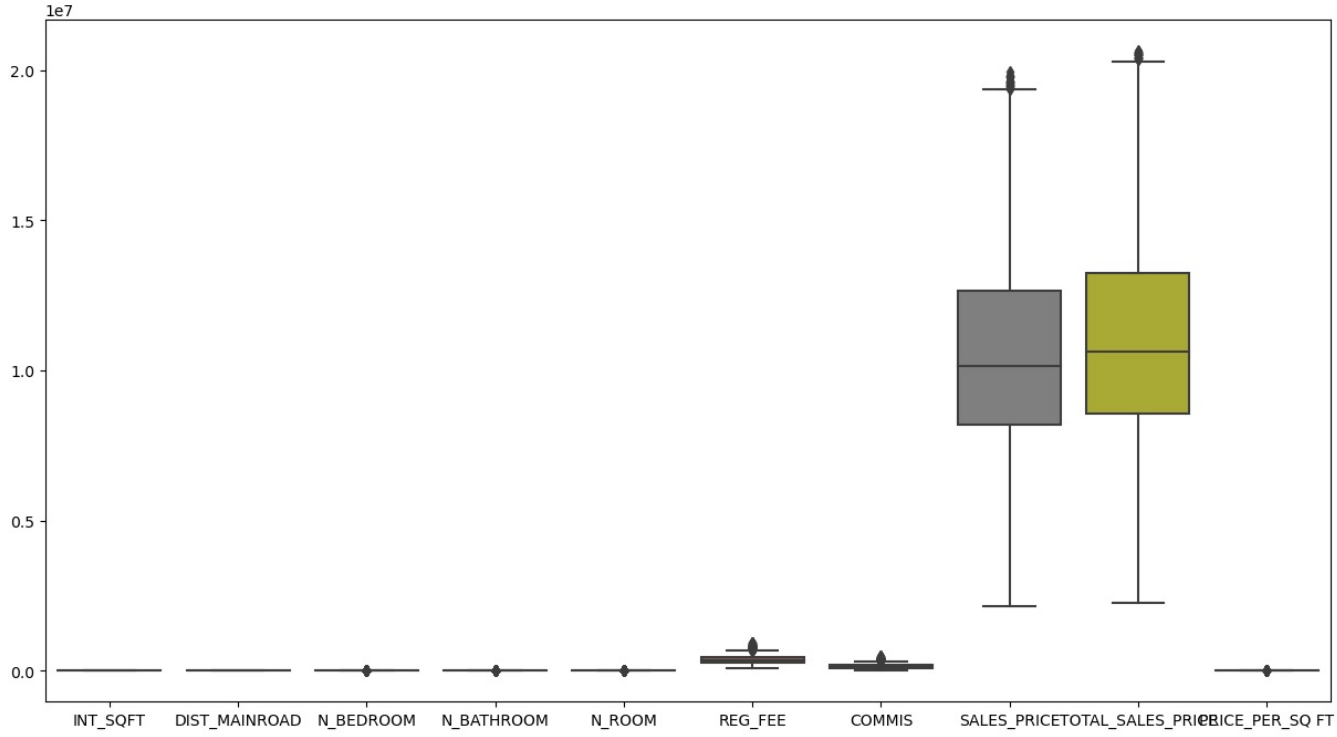
In [75]:
```
# creating new dataframe

df5 = df4.copy()
```

In [76]:
```
# using clip method to remove outlier taking 99 % of upper value

df5=
df5[df5['TOTAL_SALES_PRICE']==df5['TOTAL_SALES_PRICE'].clip(lower=1192439.0,upper=21067637.0*0.9
```

In [77]:
```
# shape of data after outlier treatment

df5.shape
```

Out[77]:
```
(6839, 19)
```

In [78]:
```
# checking outliers after removing extra rows that are out of upper bound value

fig = plt.figure(figsize =(15,8))
sns.boxplot(data = df5)
plt.show()
```

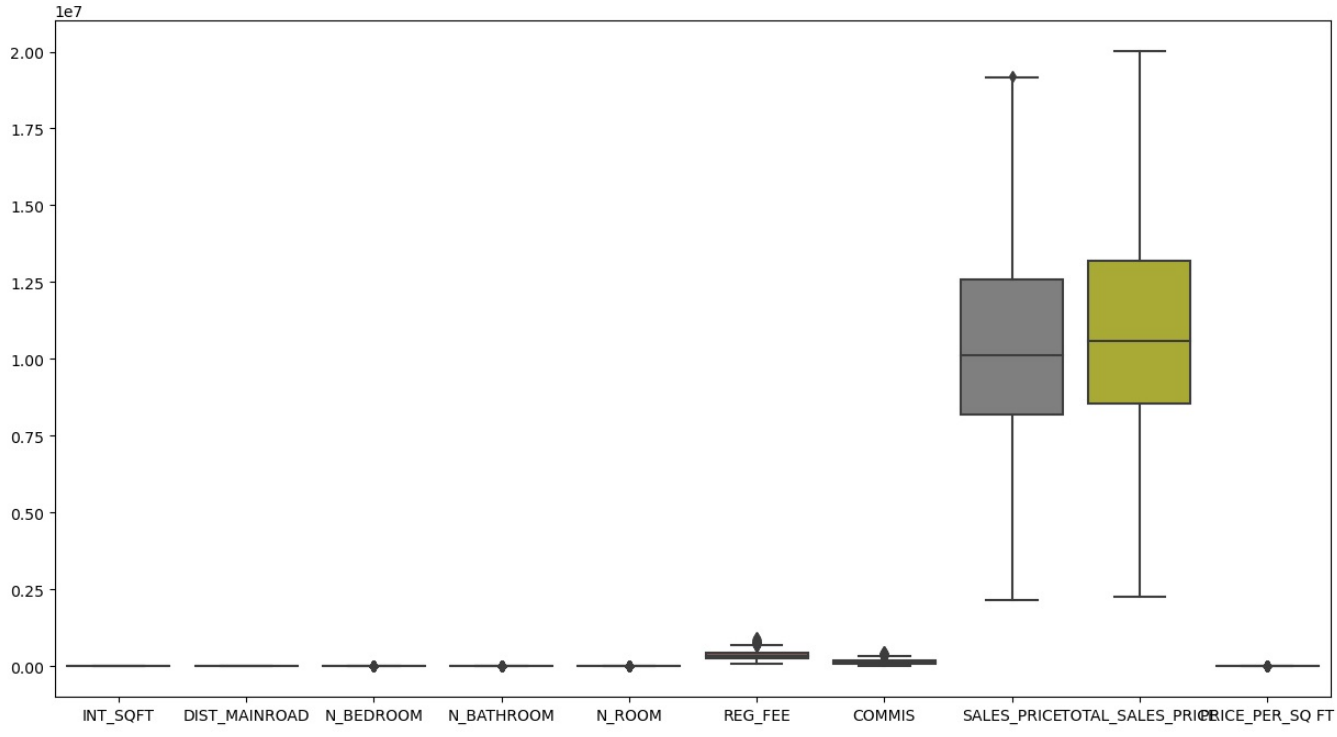


In [79]:

```
# As outliers are present even now, let's take 95% of upper value to clip from data

df5=
df5[df5['TOTAL_SALES_PRICE']==df5['TOTAL_SALES_PRICE'].clip(lower=1192439.0,upper=21067637.0*0.9
```

In [80]:

```
# checking outliers after removing extra rows that are out of upper bound value

fig = plt.figure(figsize =(15,8))
sns.boxplot(data = df5)
plt.show()
```



Outliers removed

In [81]:

```
# checking dataset again

df5
```

Out[81]:

	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	STREET
0	Karapakkam	1004	131	1	1	3	AbNormal	Yes	Commercial	Paved
2	Adyar	909	70	1	1	3	AbNormal	Yes	Commercial	Gravel
3	Velachery	1855	14	3	2	5	Family	No	Others	Paved
4	Karapakkam	1226	84	1	1	3	AbNormal	Yes	Others	Gravel
5	Chrompet	1220	36	2	1	4	Partial	No	Commercial	NoAccess
...	...	...	...	...	...	...	...	...	...	...
7104	Karapakkam	598	51	1	1	2	AdjLand	No	Others	NoAccess
7105	Velachery	1897	52	3	2	5	Family	Yes	Others	NoAccess
7106	Velachery	1614	152	2	1	4	Normal Sale	No	House	Gravel
7107	Karapakkam	787	40	1	1	2	Partial	Yes	Commercial	Paved
7108	Velachery	1896	156	3	2	5	Partial	Yes	Others	Paved

6782 rows × 19 columns

In [82]:
```
# Removing COMMIS and REG_FEE, SALES_PRICE column as these are not required

df5.drop(columns=['REG_FEE', 'COMMIS', 'SALES_PRICE'], inplace= True)
```

In [83]:
```
# checking columns

df5.columns
```

Out[83]:
```
Index(['AREA', 'INT_SQFT', 'DIST_MAINROAD', 'N_BEDROOM', 'N_BATHROOM',
       'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'BUILDTYPE', 'STREET', 'MZZONE',
       'TOTAL_SALES_PRICE', 'BUILD YEAR', 'SALE_YEAR', 'UTILITY',
       'PRICE_PER_SQ FT'],
      dtype='object')
```

In [84]:
```
df5.head()
```

Out[84]:

	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	STREET	MZ
0	Karapakkam	1004	131	1	1	3	AbNormal	Yes	Commercial	Paved	
2	Adyar	909	70	1	1	3	AbNormal	Yes	Commercial	Gravel	
3	Velachery	1855	14	3	2	5	Family	No	Others	Paved	
4	Karapakkam	1226	84	1	1	3	AbNormal	Yes	Others	Gravel	
5	Chrompet	1220	36	2	1	4	Partial	No	Commercial	NoAccess	

#

Data Visualization

In [85]:
```
from pandas_profiling import ProfileReport as pf
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_9156\927421620.py:1: DeprecationWarning: `import pandas_profiling` is going to be deprecated by April 1st. Please use `import ydata_profiling` instead.
  from pandas_profiling import ProfileReport as pf
```

In [86]:
```
pf(df5)
```

```
Summarize dataset:   0%|          | 0/5 [00:00<?, ?it/s]
Generate report structure:   0%|          | 0/1 [00:00<?, ?it/s]
Render HTML:   0%|          | 0/1 [00:00<?, ?it/s]
```

Overview

Dataset statistics

Number of variables	16
Number of observations	6782
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	0
Duplicate rows (%)	0.0%
Total size in memory	1.1 MiB
Average record size in memory	167.0 B

Variable types

Categorical	9
Numeric	6
Boolean	1

Alerts

`INT_SQFT` is highly overall correlated with `TOTAL_SALES_PRICE` and 4 other fields (TOTAL_SALES_PRICE, PRICE_PER_SQ FT, N_BEDROOM, N_BATHROOM, N_ROOM)	**High correlation**
`TOTAL_SALES_PRICE` is highly overall correlated with `INT_SQFT`	**High correlation**
`PRICE_PER_SQ FT` is highly overall correlated with `INT_SQFT`	**High correlation**
`AREA` is highly overall correlated with `N_BATHROOM`	**High correlation**
`N_BEDROOM` is highly overall correlated with `INT_SQFT` and 2 other fields (INT_SQFT, N_BATHROOM, N_ROOM)	**High correlation**

Out[86]:

In [87]:
```
#Correlation in numerical columns of data (only numerical columns participate)

df5.corr()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_9156\3116386517.py:3: FutureWarning: The default value of numeric_only
in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or sp
ecify the value of numeric_only to silence this warning.
  df5.corr()
```

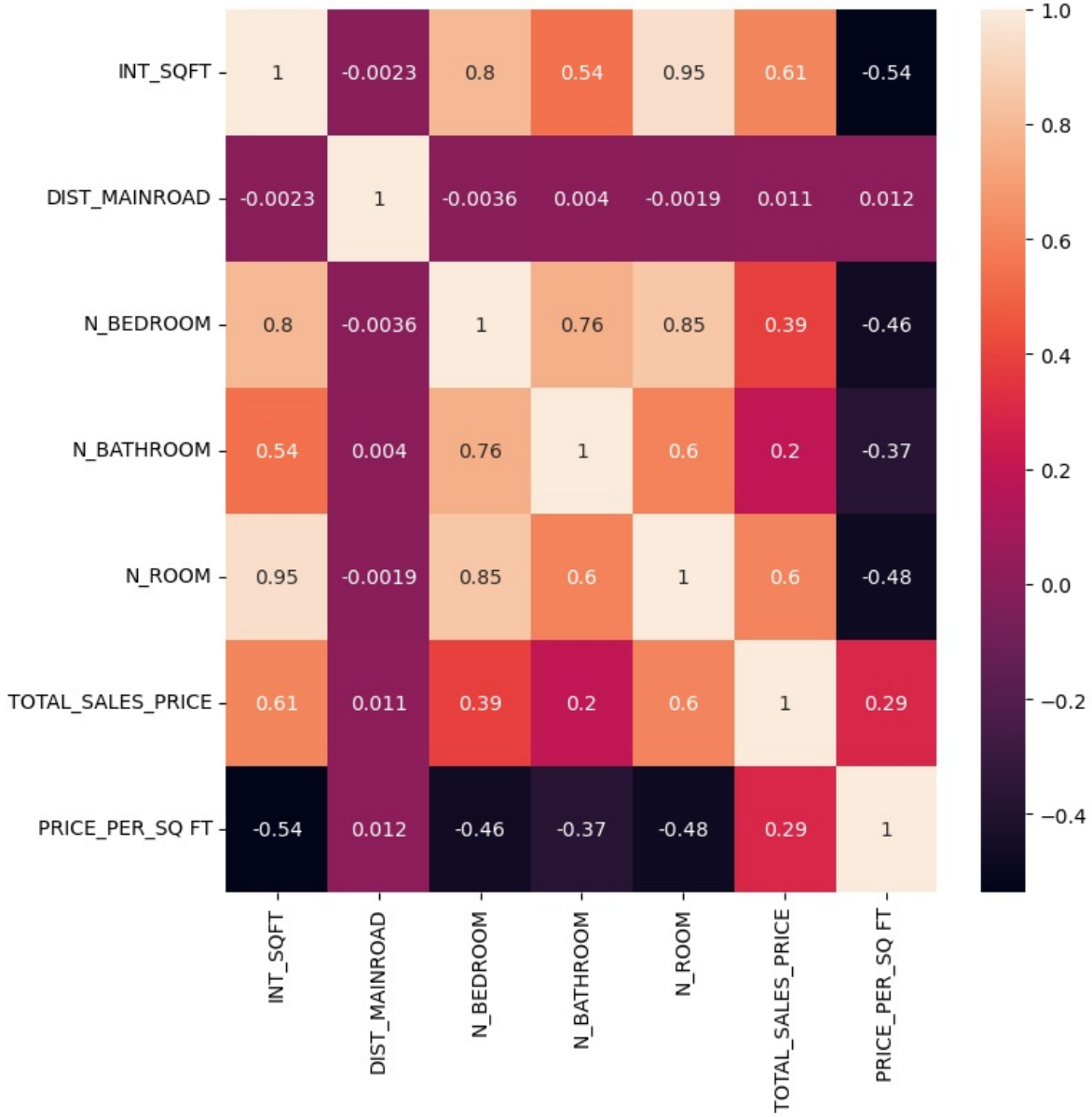
Out[87]:

	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	TOTAL_SALES_PRICE	PRICE_PER_SQ FT
INT_SQFT	1.000000	-0.002273	0.800027	0.543732	0.949781	0.611010	-0.535176
DIST_MAINROAD	-0.002273	1.000000	-0.003555	0.003993	-0.001870	0.010762	0.012120
N_BEDROOM	0.800027	-0.003555	1.000000	0.760998	0.851397	0.387391	-0.463757
N_BATHROOM	0.543732	0.003993	0.760998	1.000000	0.599007	0.195451	-0.371684
N_ROOM	0.949781	-0.001870	0.851397	0.599007	1.000000	0.604707	-0.478621
TOTAL_SALES_PRICE	0.611010	0.010762	0.387391	0.195451	0.604707	1.000000	0.292963
PRICE_PER_SQ FT	-0.535176	0.012120	-0.463757	-0.371684	-0.478621	0.292963	1.000000

In [88]:

```
# correlation representation using heatmap

plt.figure(figsize=(8,8))
sns.heatmap(df5.corr(),annot =True)
plt.show()
```

```
C:\Users\HP\AppData\Local\Temp\ipykernel_9156\4012904402.py:4: FutureWarning: The default value of numeric_only
in DataFrame.corr is deprecated. In a future version, it will default to False. Select only valid columns or sp
ecify the value of numeric_only to silence this warning.
  sns.heatmap(df5.corr(),annot =True)
```



1. N_Room and INT_SQFT are highly correlated to each other

2. N_Bedroom, N_ROOM and INT_SQFT are correalted to each other

3. N_BATHROOM is Correlated with N_BEDROOM

4. TOTAL_SALES_PRICE is correlated to INT_SQFT,N_ROOM mostly then comes N_BEDROOM and N_BATHROOM

5. Price/SQFT is -ve correlated to N_room, N_Bathroom, N_Bedroom and INT_SQFT

#

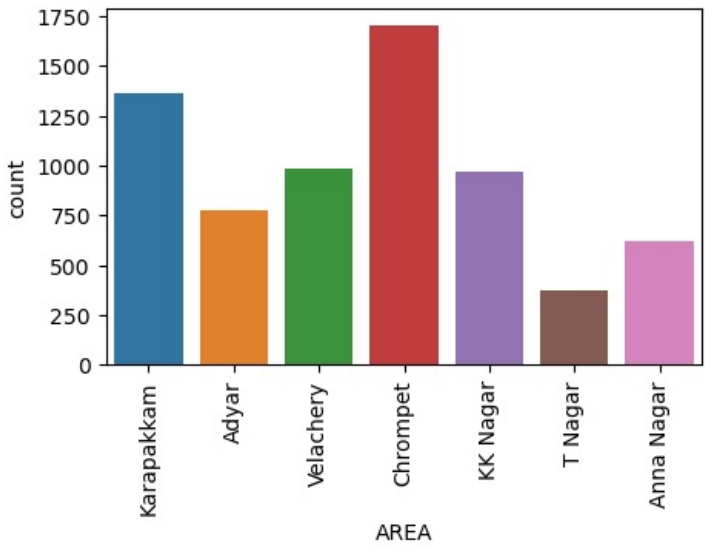
#

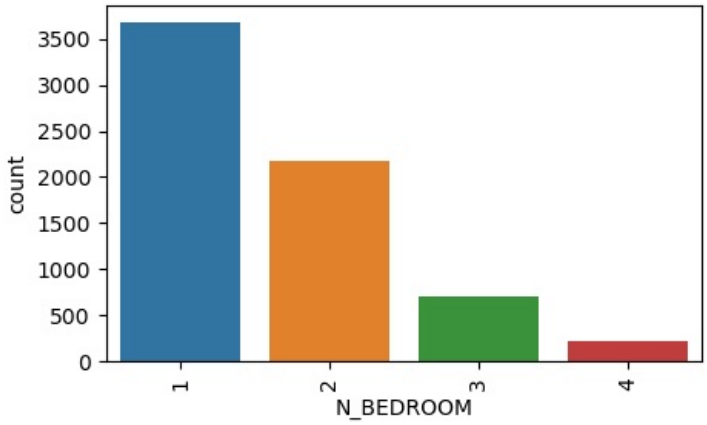
------ UNIVARIATE ANALYSIS------

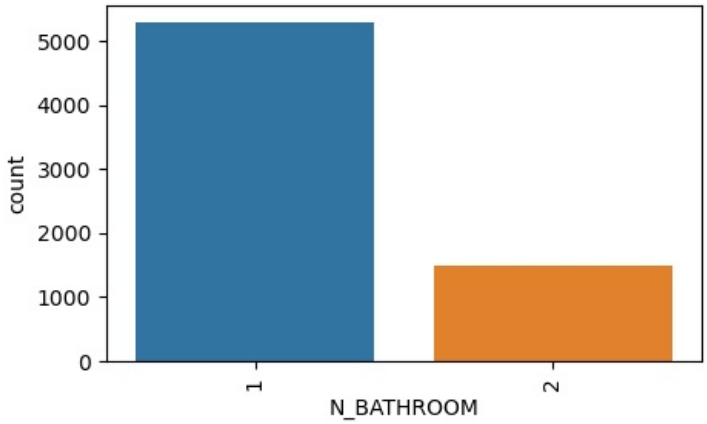
In [89]:
```
df5.info()
```

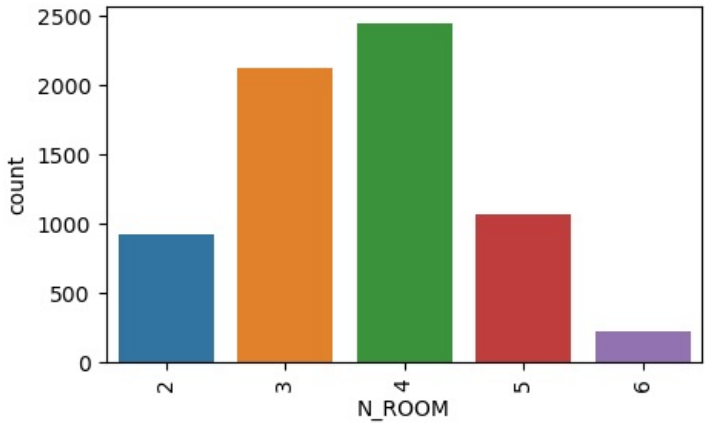
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6782 entries, 0 to 7108
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   AREA               6782 non-null   object 
 1   INT_SQFT           6782 non-null   int64  
 2   DIST_MAINROAD      6782 non-null   int64  
 3   N_BEDROOM          6782 non-null   int32  
 4   N_BATHROOM         6782 non-null   int32  
 5   N_ROOM             6782 non-null   int64  
 6   SALE_COND          6782 non-null   object 
 7   PARK_FACIL         6782 non-null   object 
 8   BUILDTYPE          6782 non-null   object 
 9   STREET             6782 non-null   object 
 10  MZZONE             6782 non-null   object 
 11  TOTAL_SALES_PRICE  6782 non-null   int64  
 12  BUILD YEAR         6782 non-null   object 
 13  SALE_YEAR          6782 non-null   object 
 14  UTILITY            6782 non-null   object 
 15  PRICE_PER_SQ FT    6782 non-null   float64
dtypes: float64(1), int32(2), int64(4), object(9)
memory usage: 1.1+ MB
```

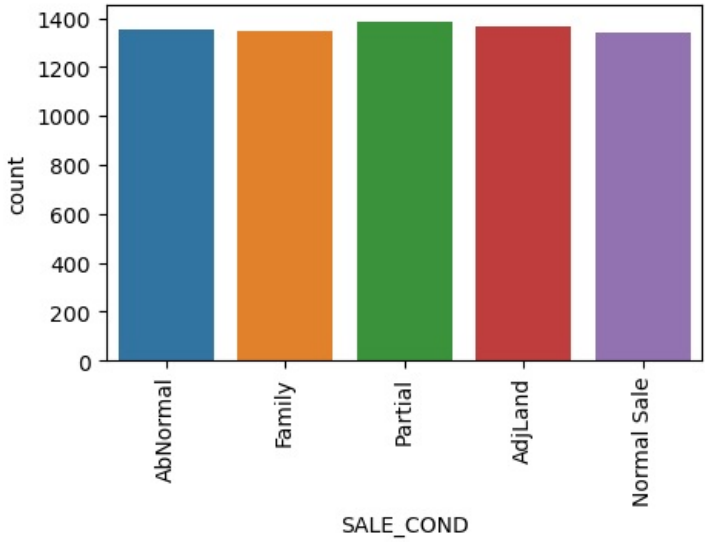
In [90]:
```
# Countplots of categorical columns

col = ['AREA', 'N_BEDROOM', 'N_BATHROOM',
       'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'BUILDTYPE', 'STREET', 'MZZONE', 'BUILD YEAR',
'SALE_YEAR', 'UTILITY']
for i in col:
    plt.figure(figsize=(5,3))
    sns.countplot(x=df5[i],data = df5)
    plt.xticks(rotation = 90)
    plt.show()
```

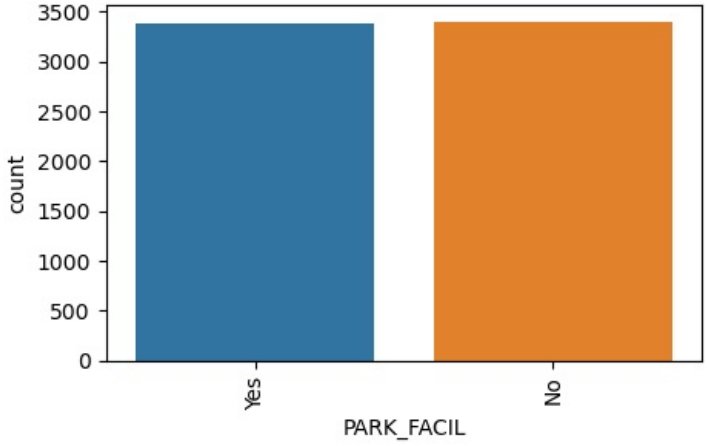


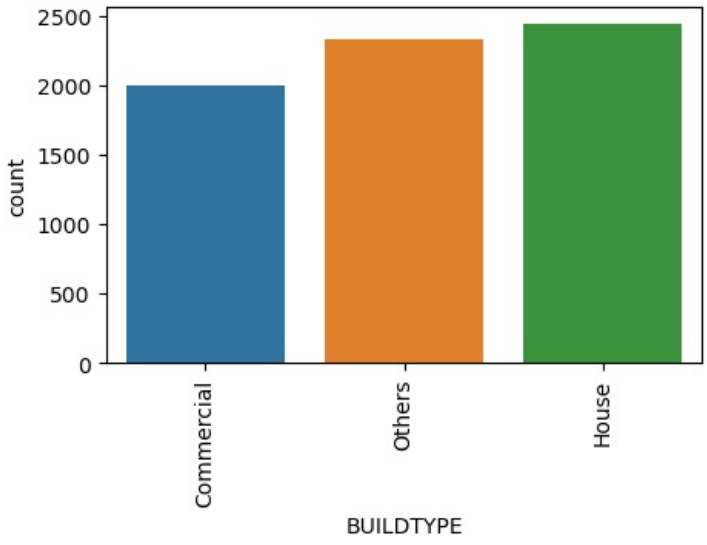

3500
3000
2500
2000
1500
1000
500
0
count
1
2
3
4
N_BEDROOM

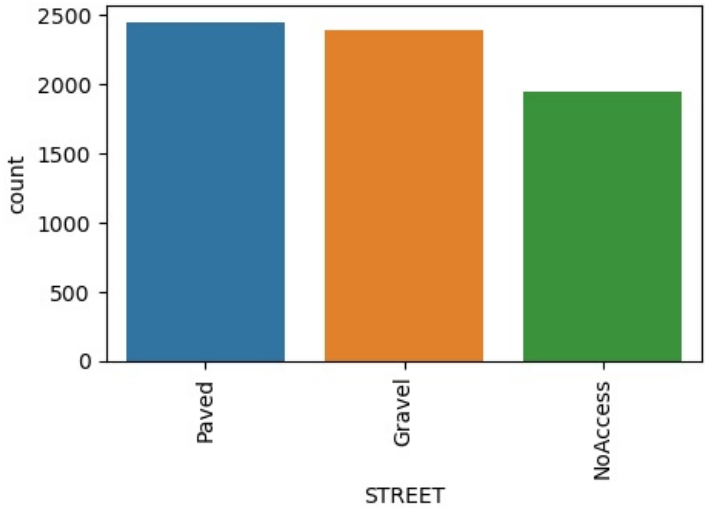

5000
4000
3000
2000
1000
0
count
1
2
N_BATHROOM

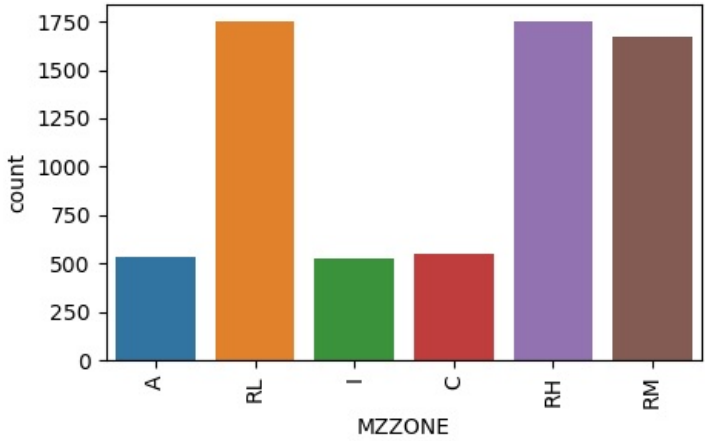

2500
2000
1500
1000
500
0
count
2
3
4
5
6
N_ROOM

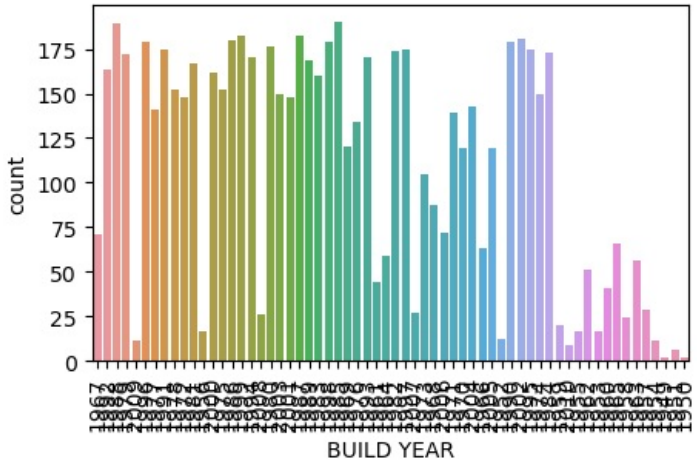

1400
1200
1000
800
600
400
200
0
count
AbNormal
Family
Partial
AdjLand
Normal Sale
SALE_COND

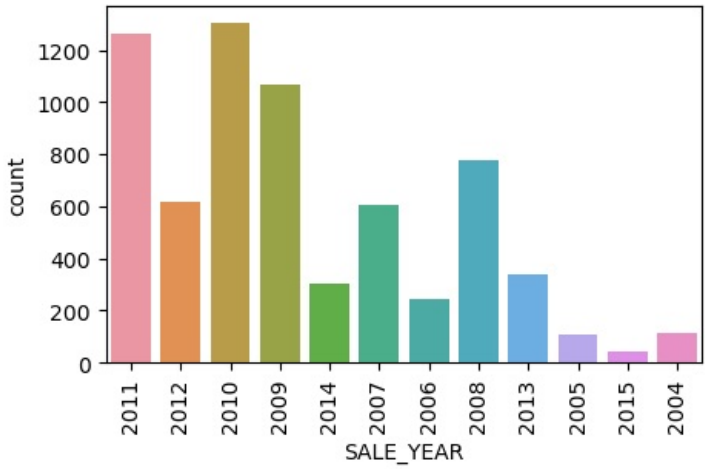

3500
3000
2500
2000
1500
1000
500
0
count
Yes
No
PARK_FACIL

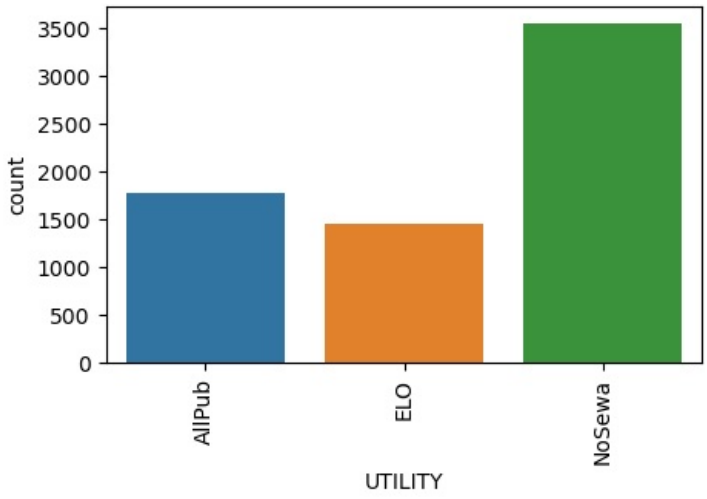

2500
2000
1500
1000
500
0
count
Commercial
Others
House
BUILDTYPE


2500
2000
1500
1000
500
0
count
Paved
Gravel
NoAccess
STREET


1750
1500
1250
1000
750
500
250
0
count
A
RL
I
C
RH
RM
MZZONE


175
150
125
100
75
50
25
0
count
BUILD YEAR


1200
1000
800
600
400
200
0
count
2011
2012
2010
2009
2014
2007
2006
2008
2013
2005
2015
2004
SALE_YEAR



1. AREA- Chrompet AREA has max houses and T Nagar has min houses

2. N_BEDROOM - There are max houses with 1 bedroom (3500+) and 4 bedroom houses(250 approx) -minimum

3. N_BATHROOM- 5000+ Houses has only 1 bathroom and 1500 approx houses has 2 bathrooms

4. No. of rooms - 2500+ houses has 4 rooms(max) and 200 houses has 6 rooms(min)

5. SALE Condition - condition-wise houses are equally divides- approx 1300 each

6. Park Facility - 3200 houses has parking and similar number for 'no parking'

7. BUILDTYPE- Max houses(2400) were built commercial are less

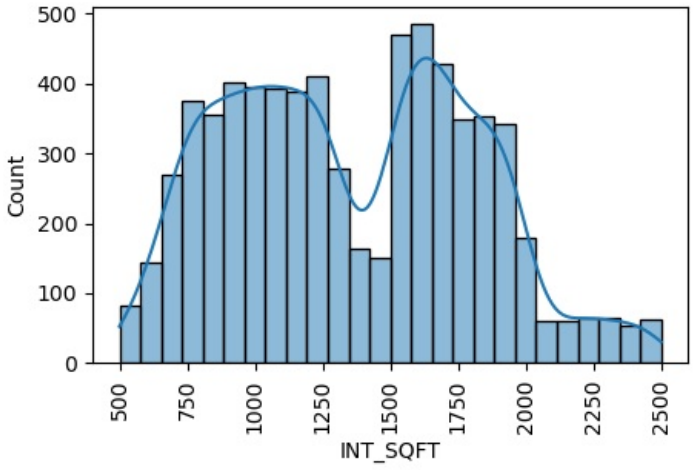
8. STREET - Paved streets max (2300 apprx)

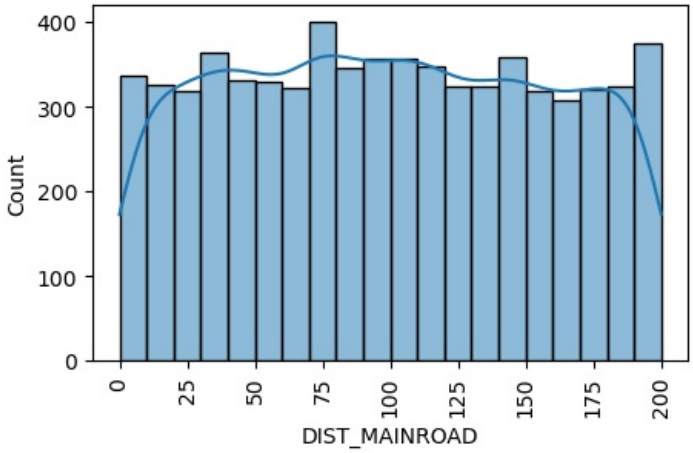
9. MZZONE - RH ZONE has max houses(1700), I Zone has min houses(500)

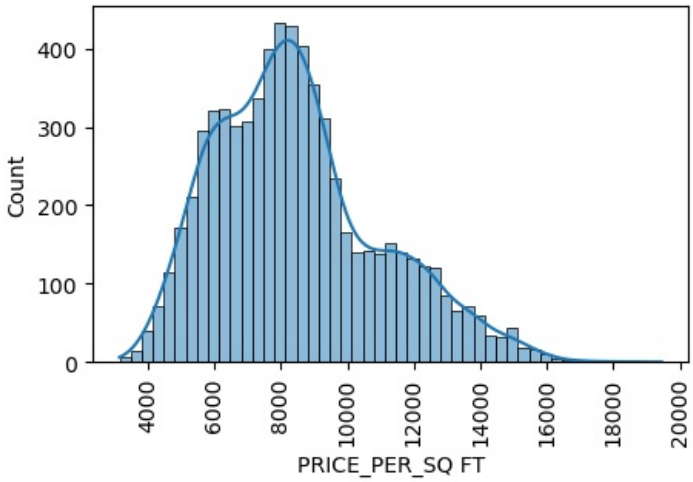
10.SALE_ YEAR -In 2010 year max houses(1300 apprx) were sold, in 2015 min houses were sold(75 apprx)

11. UTILITY- NOSewa category houses are max(3500) , with ELO min houses(1300)

In [91]:
```
# Distribution - histogram plot

col = [ 'INT_SQFT', 'DIST_MAINROAD',
       'TOTAL_SALES_PRICE',
       'PRICE_PER_SQ FT']
for i in col:
    plt.figure(figsize=(5,3))
    sns.histplot(x=df5[i],data = df5,kde= True)
    plt.xticks(rotation = 90)
    plt.show()
```







1.Total Sales price is approximate normally distributed

2.Price Per Square foot is somwhat right skewed

#

---- BIVARIATE ANALYSIS----

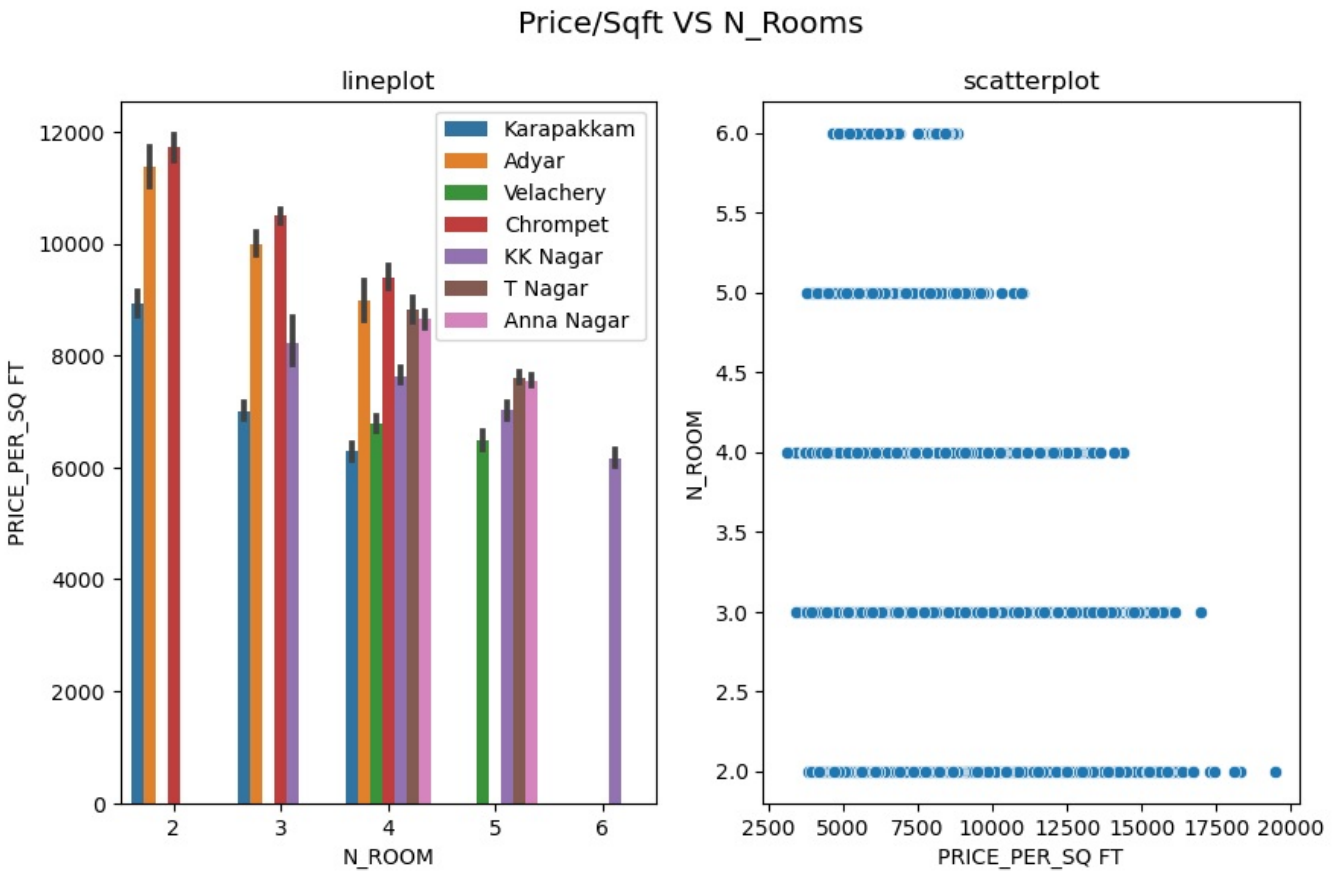
In [92]:
```
df5.columns
```

Out[92]:
```
Index(['AREA', 'INT_SQFT', 'DIST_MAINROAD', 'N_BEDROOM', 'N_BATHROOM',
       'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'BUILDTYPE', 'STREET', 'MZZONE',
       'TOTAL_SALES_PRICE', 'BUILD YEAR', 'SALE_YEAR', 'UTILITY',
       'PRICE_PER_SQ FT'],
      dtype='object')
```

In [93]:
```
# drawing barplot and scatterplot for 'PRICE/SQFT' and 'Number of Rooms'


fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,6))
sns.barplot(x="N_ROOM",y="PRICE_PER_SQ FT", data  = df5, ax =ax[0], hue='AREA')
```

```
sns.scatterplot(x="PRICE_PER_SQ FT", y="N_ROOM", data  = df5, ax = ax[1])
plt.suptitle('Price/Sqft VS N_Rooms',fontsize="x-large")
ax[0].legend(loc='upper right')
ax[0].set_title('lineplot')
ax[1].set_title('scatterplot')
```

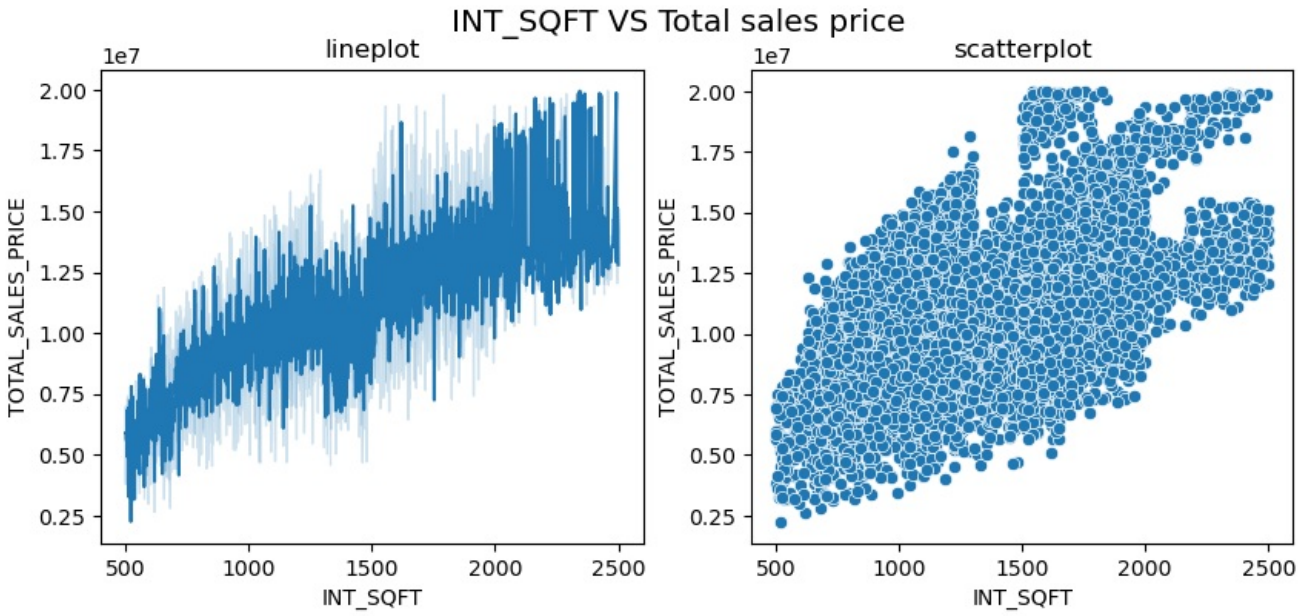
Out[93]:
```
Text(0.5, 1.0, 'scatterplot')
```



1. Price per sq foot is higher for 2 room houses
2. Price per sq foot is lesser for 6 room houses
3. KK Nagar Area has 6 room houses
4. Chrompet Area has 2,3 and 4 room houses and expensive of all Area - considering Price Per square foot cost

In [94]:
```
# drawing lineplot and scatterplot for 'INT_SQFT' and 'TOTAL_SALES_PRICE'

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
sns.lineplot(x="INT_SQFT", y="TOTAL_SALES_PRICE", data  = df5, ax =ax[0])
sns.scatterplot(x="INT_SQFT", y="TOTAL_SALES_PRICE", data  = df5, ax = ax[1])
plt.suptitle('INT_SQFT VS Total sales price',fontsize="x-large")
ax[0].set_title('lineplot')
ax[1].set_title('scatterplot')
```

Out[94]:
```
Text(0.5, 1.0, 'scatterplot')
```



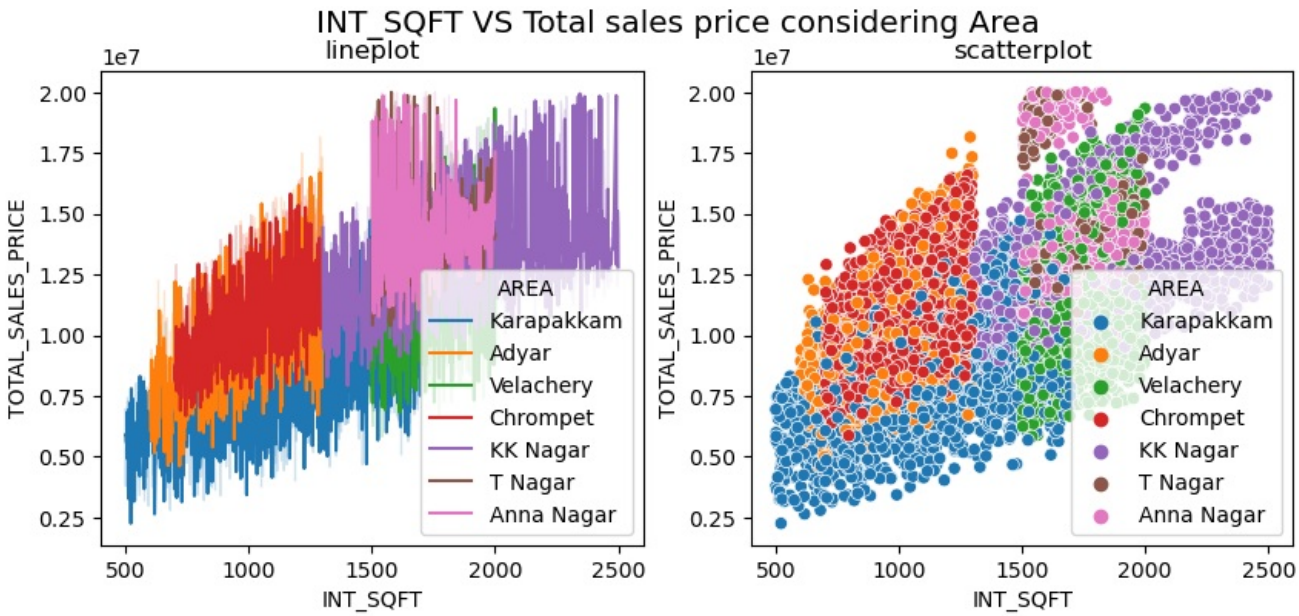
As Int Sqft increasing Total_Sale price also increasing

In [95]:

```
# drawing lineplot and scatterplot for 'INT_SQFT' and 'TOTAL_SALES_PRICE' in reference to AREA

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
sns.lineplot(x="INT_SQFT", y="TOTAL_SALES_PRICE", data  = df5, ax =ax[0], hue='AREA')
sns.scatterplot(x="INT_SQFT", y="TOTAL_SALES_PRICE", data  = df5, ax = ax[1], hue='AREA')
plt.suptitle('INT_SQFT VS Total sales price considering Area',fontsize="x-large")
ax[0].set_title('lineplot')
ax[1].set_title('scatterplot')
```

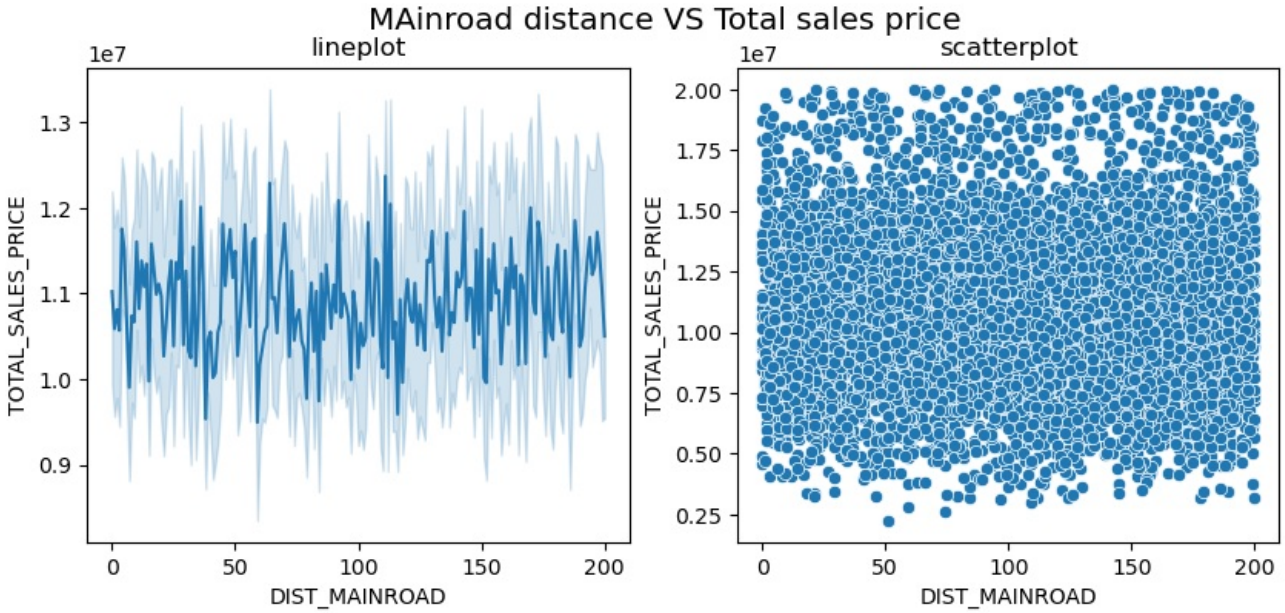
Out[95]:

```
Text(0.5, 1.0, 'scatterplot')
```



In [96]:
```
# drawing lineplot and scatterplot for 'DIST_MAINROAD' and 'TOTAL_SALES_PRICE'

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
sns.lineplot(x="DIST_MAINROAD", y="TOTAL_SALES_PRICE", data  = df5, ax =ax[0])
sns.scatterplot(x="DIST_MAINROAD", y="TOTAL_SALES_PRICE", data  = df5, ax = ax[1])
plt.suptitle('MAinroad distance VS Total sales price',fontsize="x-large")
ax[0].set_title('lineplot')
ax[1].set_title('scatterplot')
```

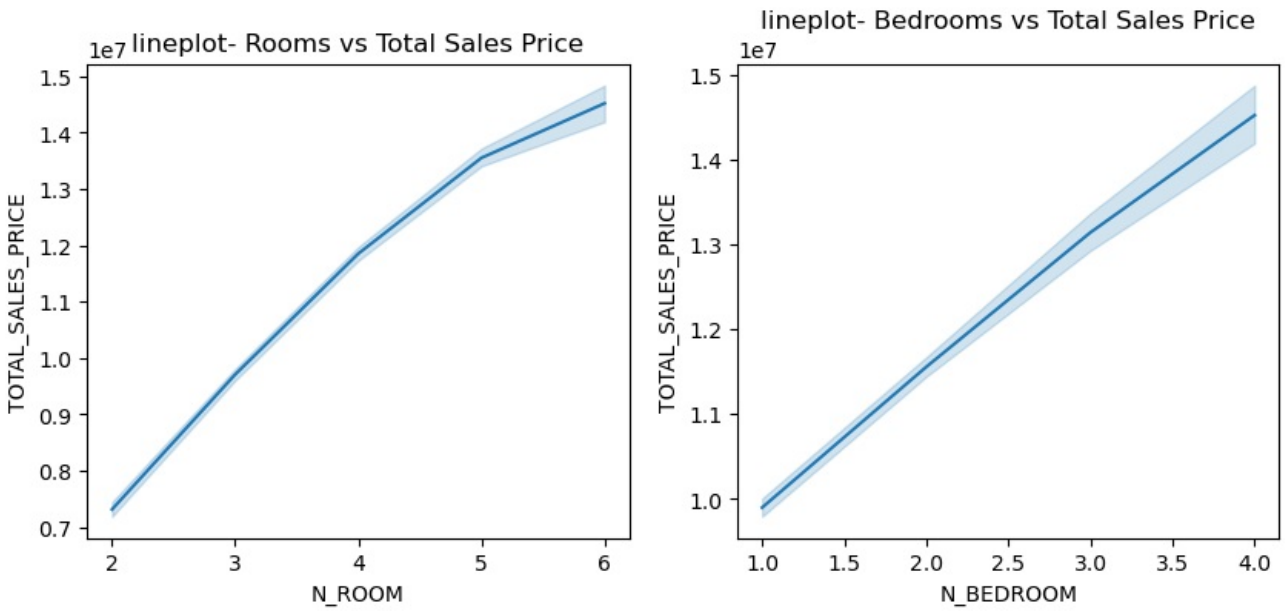
Out[96]:
```
Text(0.5, 1.0, 'scatterplot')
```



It shows distance from mainroad does not affect sales price of property

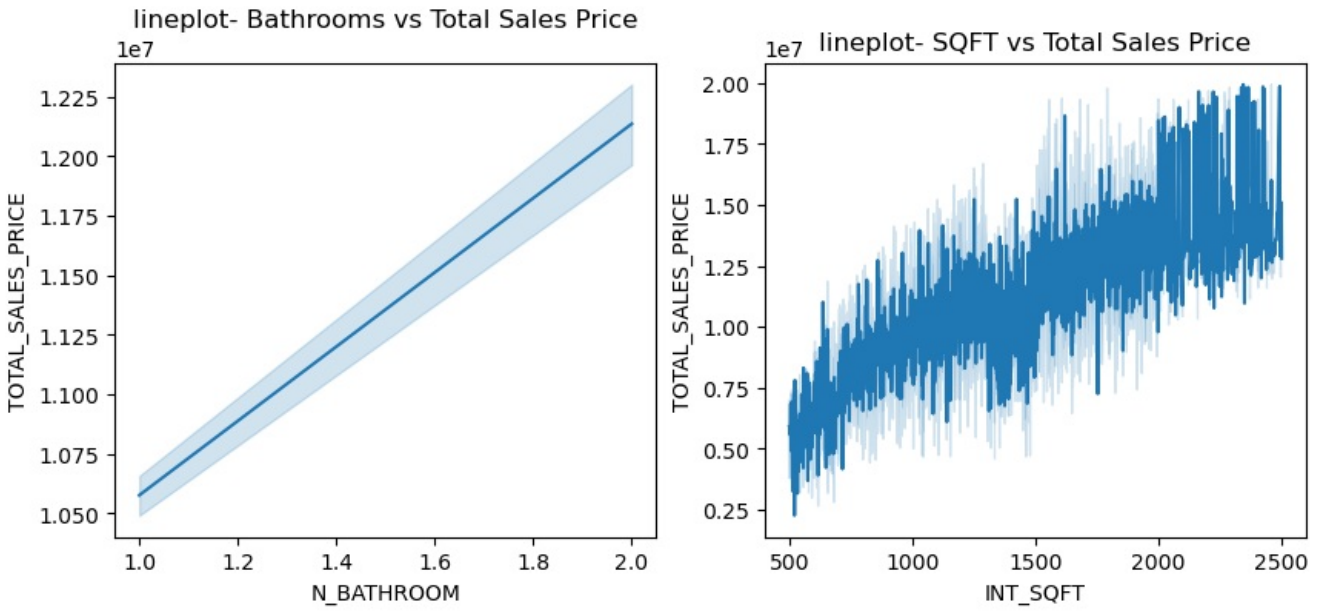
In [97]:
```
# Number of rooms and bedrooms affect on Total sales price

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
sns.lineplot(x="N_ROOM", y="TOTAL_SALES_PRICE", data  = df5, ax=ax[0])

sns.lineplot(x="N_BEDROOM", y="TOTAL_SALES_PRICE", data = df5, ax=ax[1])


ax[0].set_title('lineplot- Rooms vs Total Sales Price')
ax[1].set_title('lineplot- Bedrooms vs Total Sales Price')
```

Out[97]:
```
Text(0.5, 1.0, 'lineplot- Bedrooms vs Total Sales Price')
```



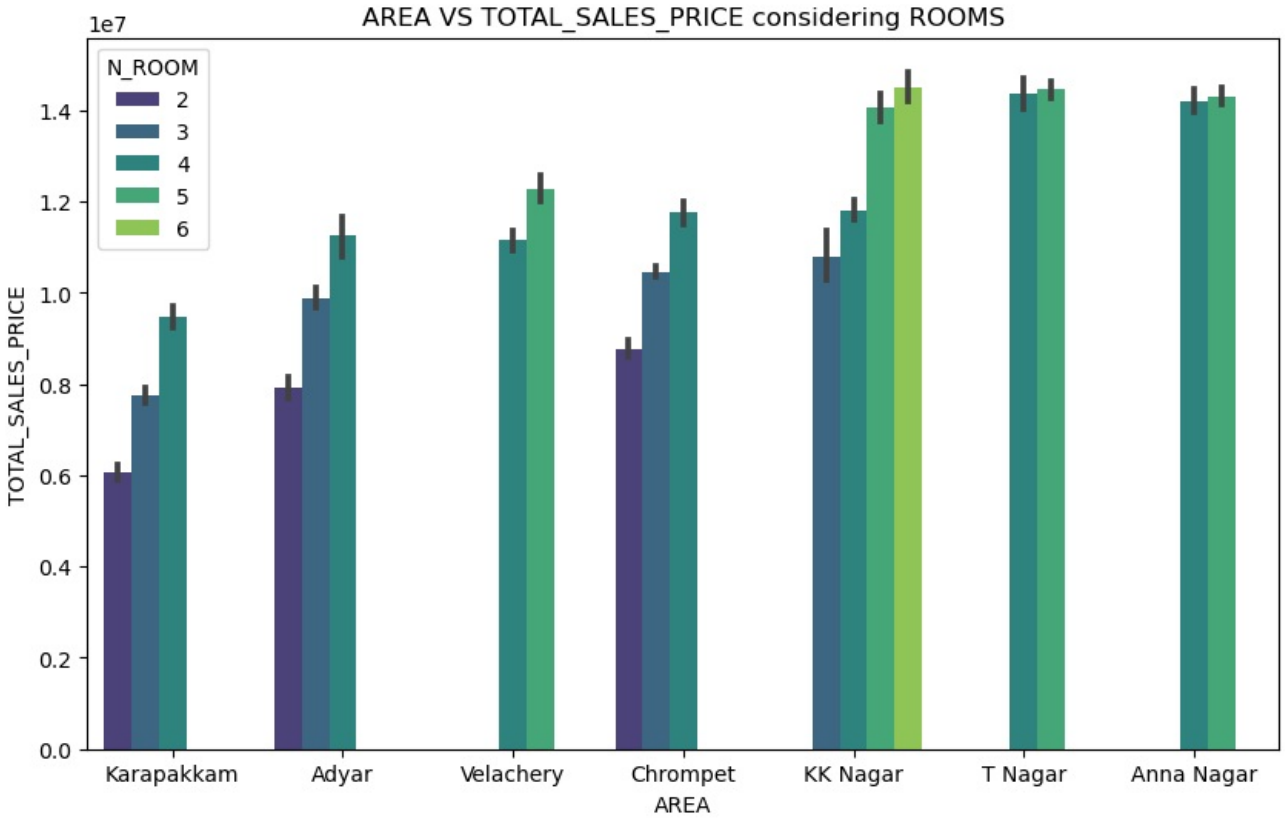
In [98]:
```
# Number of Bathrooms and INT_SQFT affect on Total sales price

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(10,4))
sns.lineplot(x="N_BATHROOM", y="TOTAL_SALES_PRICE", data  = df5,ax =ax[0])
sns.lineplot(x="INT_SQFT", y="TOTAL_SALES_PRICE", data  = df5, ax =ax[1])


ax[0].set_title('lineplot- Bathrooms vs Total Sales Price')
ax[1].set_title('lineplot- SQFT vs Total Sales Price')
```

Out[98]:
```
Text(0.5, 1.0, 'lineplot- SQFT vs Total Sales Price')
```



In [99]:
```
# AREA VS SALES_PRICE considering rooms

fig = plt.figure(figsize =(10,6))
sns.barplot(x='AREA',y= 'TOTAL_SALES_PRICE', data = df5 ,hue ='N_ROOM',color ='b', palette =
"viridis")
plt.title('AREA VS TOTAL_SALES_PRICE considering ROOMS')
```

Out[99]:
```
Text(0.5, 1.0, 'AREA VS TOTAL_SALES_PRICE considering ROOMS')
```



1. 6 Room house is available only in KK NAGAR

2. karapakkam is society having low Sales price in Chennai which has 2,3,4 Rooms houses available

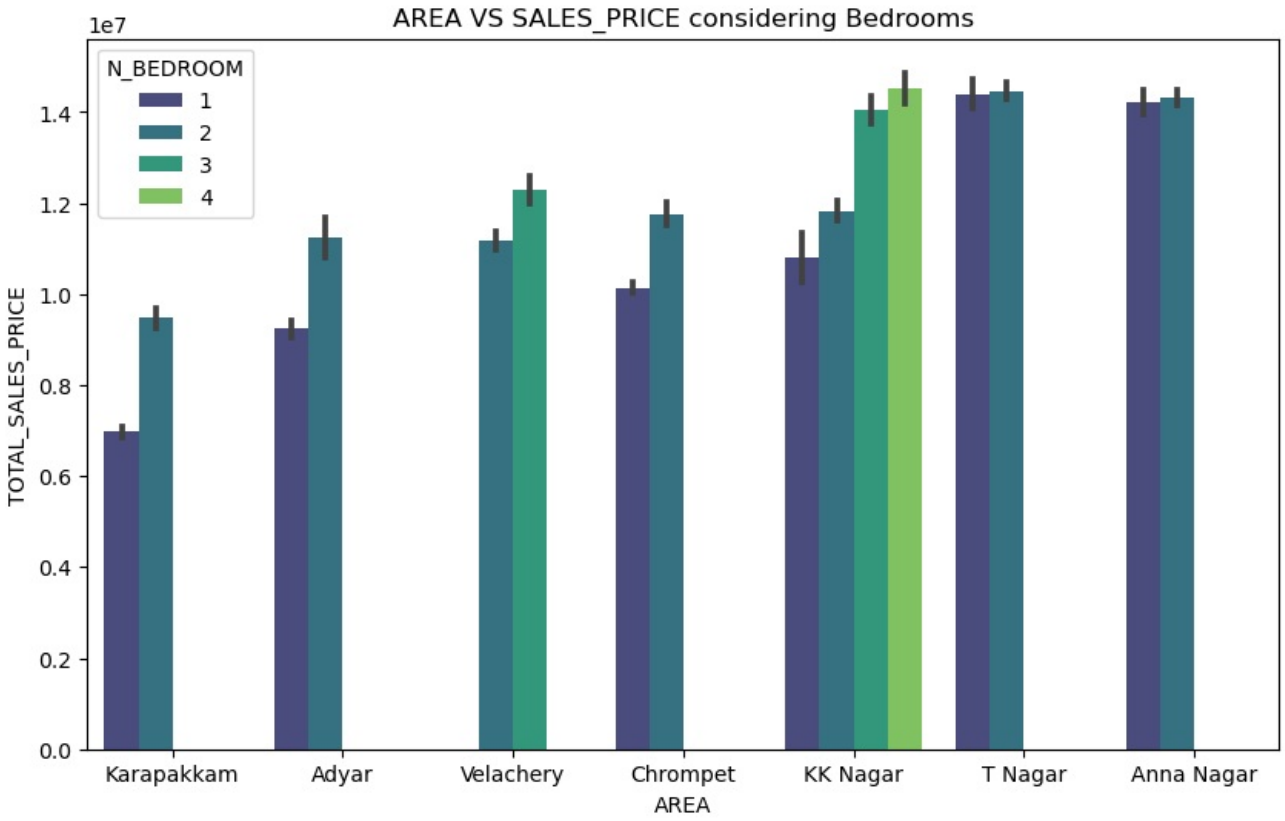
3. T NAGAR AND ANNA NAGAR has high sales price among all locations

In [100…

```
# AREA VS SALES_PRICE considering Bedrooms

fig = plt.figure(figsize =(10,6))
sns.barplot(x='AREA',y= 'TOTAL_SALES_PRICE', data = df5 ,hue ='N_BEDROOM',color ='b', palette =
"viridis")
plt.title('AREA VS SALES_PRICE considering Bedrooms')
```

Out[100]:

```
Text(0.5, 1.0, 'AREA VS SALES_PRICE considering Bedrooms')
```



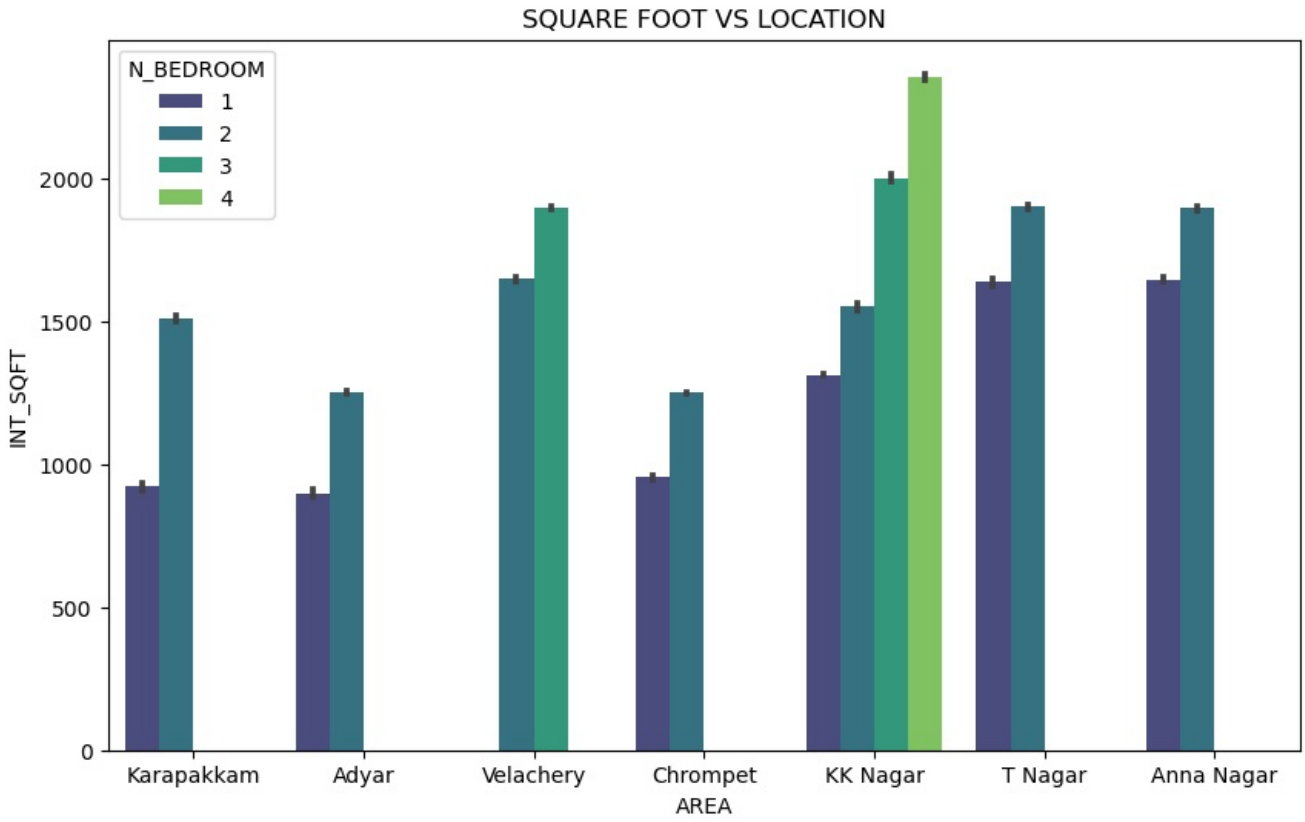
KKAGAR Society is versatile which has 1-4 Bedrooms Houses available

In [101…

```
# SQUARE FOOT VS LOCATION

fig = plt.figure(figsize =(10,6))
sns.barplot(x='AREA',y= 'INT_SQFT', data = df5 ,hue ='N_BEDROOM',color ='b', palette =
"viridis")
plt.title('SQUARE FOOT VS LOCATION')
```

Out[101]:

```
Text(0.5, 1.0, 'SQUARE FOOT VS LOCATION')
```



1. KK NAGAR Area has 1200-2500 SQ FT Range houses available

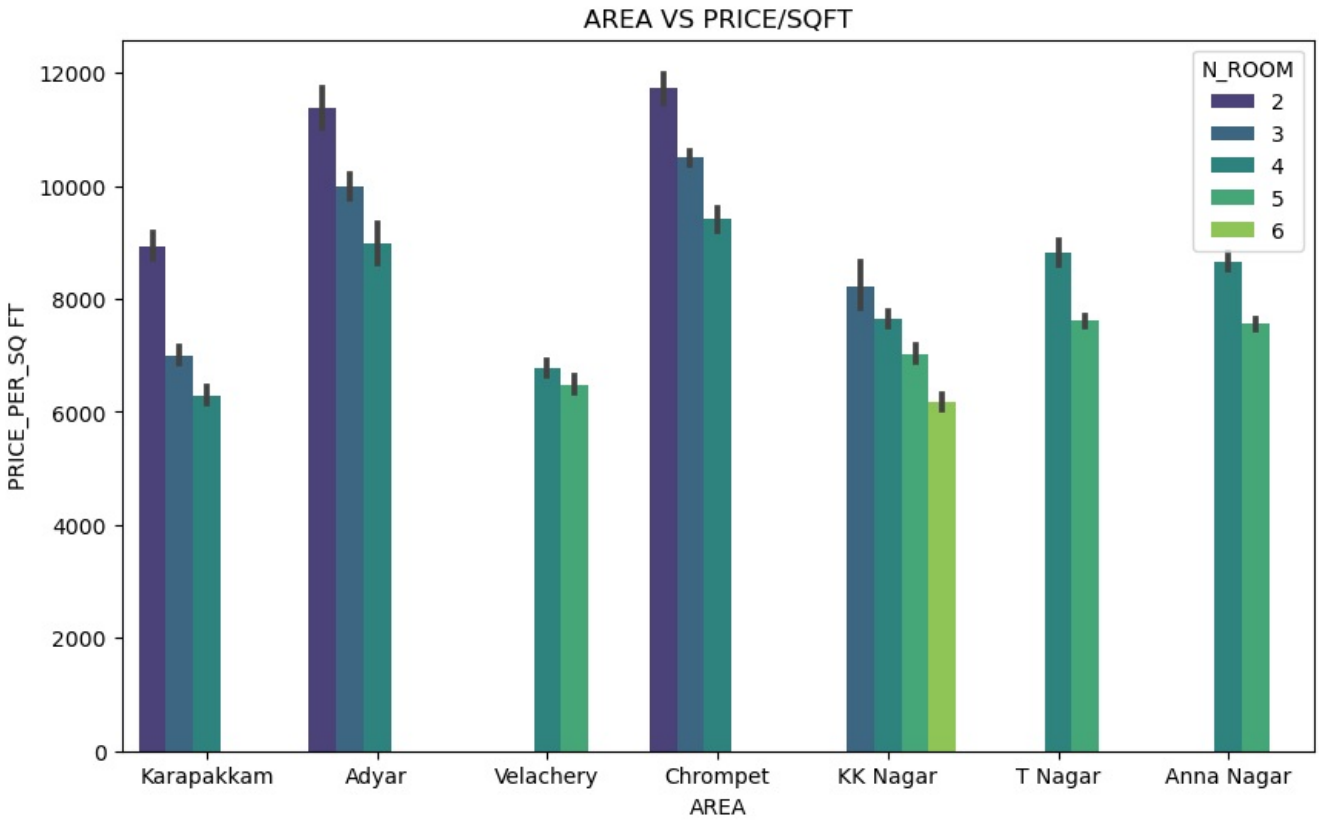
2. Chrompet and Adyar area has 900-1200 SQ ft range area (minimum among all AREA)

In [102…

```
# AREA VS PRICE/SQFT

fig = plt.figure(figsize =(10,6))
sns.barplot(x='AREA',y= 'PRICE_PER_SQ FT', data = df5 ,hue = 'N_ROOM',color ='b', palette =
"viridis")
plt.title('AREA VS PRICE/SQFT')
```

Out[102]:

```
Text(0.5, 1.0, 'AREA VS PRICE/SQFT')
```



Chrompet and Adyar area is expensive one even 2-4 Rooms available only

but Price/SQFT is reaching around 11000/SQFT

In [103…

```
df5.columns
```

Out[103]:

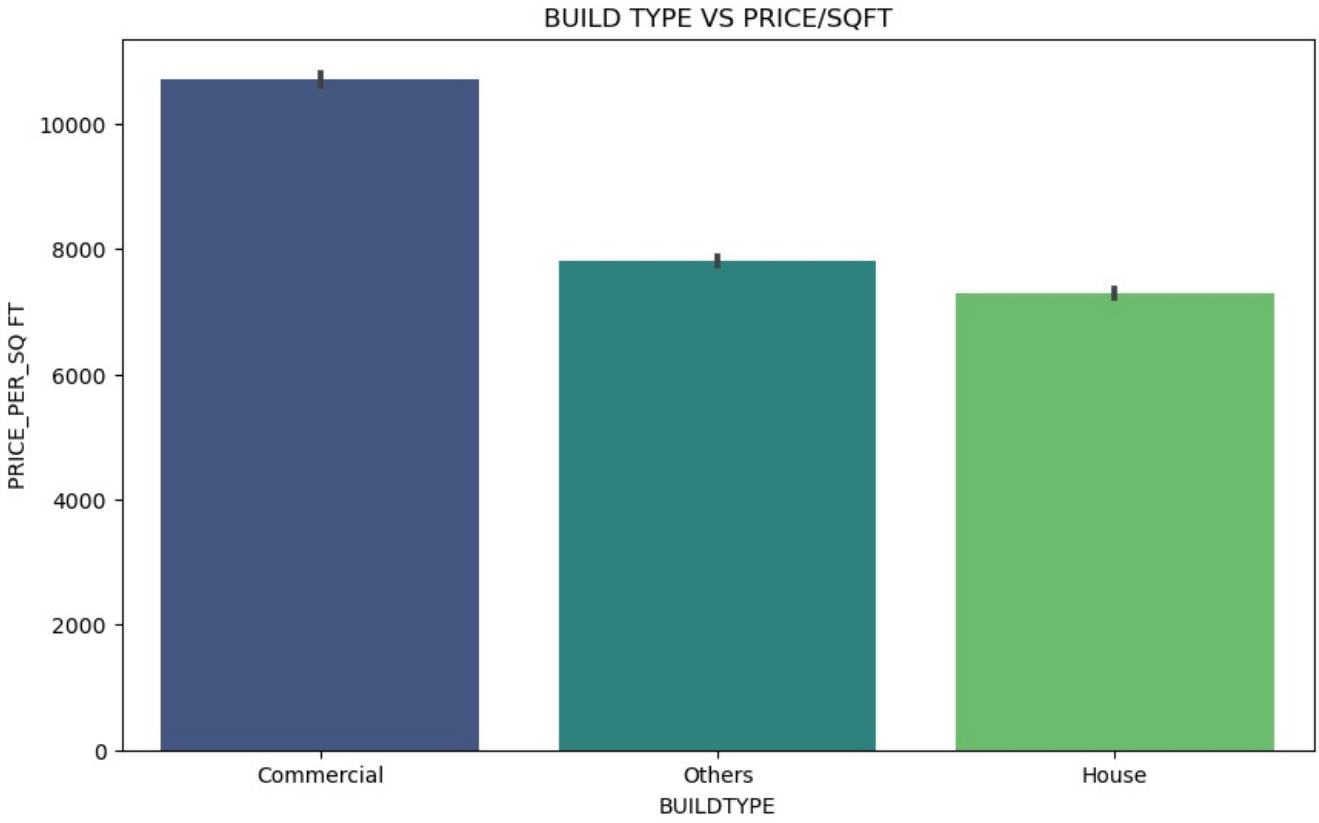
```
Index(['AREA', 'INT_SQFT', 'DIST_MAINROAD', 'N_BEDROOM', 'N_BATHROOM',
       'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'BUILDTYPE', 'STREET', 'MZZONE',
       'TOTAL_SALES_PRICE', 'BUILD YEAR', 'SALE_YEAR', 'UTILITY',
       'PRICE_PER_SQ FT'],
      dtype='object')
```

In [104…

```
# BUILD TYPE VS PRICE/SQFT

fig = plt.figure(figsize =(10,6))
sns.barplot(x='BUILDTYPE',y= 'PRICE_PER_SQ FT', data = df5 ,color ='b', palette = "viridis")
plt.title('BUILD TYPE VS PRICE/SQFT')
```

Out[104]:

```
Text(0.5, 1.0, 'BUILD TYPE VS PRICE/SQFT')
```



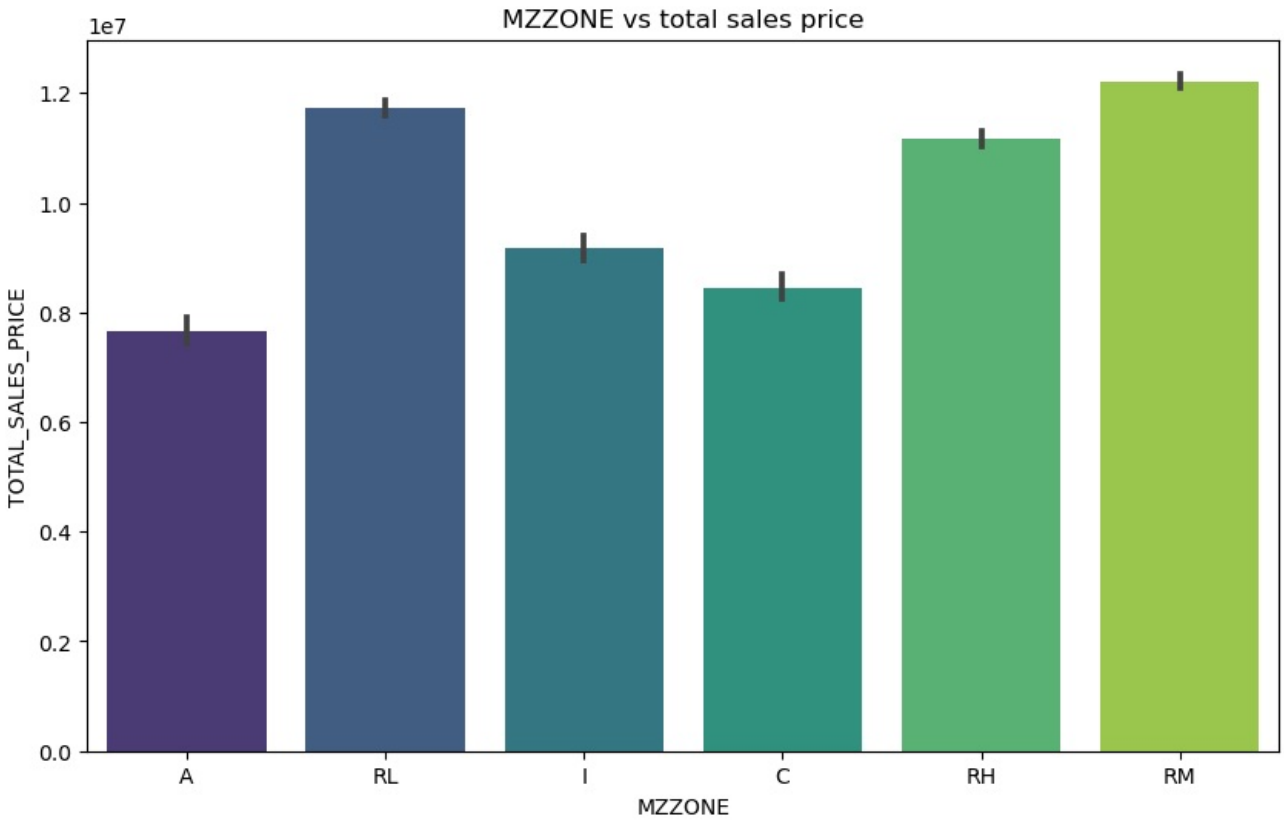
Commercial houses are expensive and simple Houses are cheaper

In [105…

```
# MZZONE vs total sales price

fig = plt.figure(figsize =(10,6))
sns.barplot(x='MZZONE',y= 'TOTAL_SALES_PRICE', data = df5 , color ='b', palette = "viridis")
plt.title('MZZONE vs total sales price')
```

Out[105]:

```
Text(0.5, 1.0, 'MZZONE vs total sales price')
```

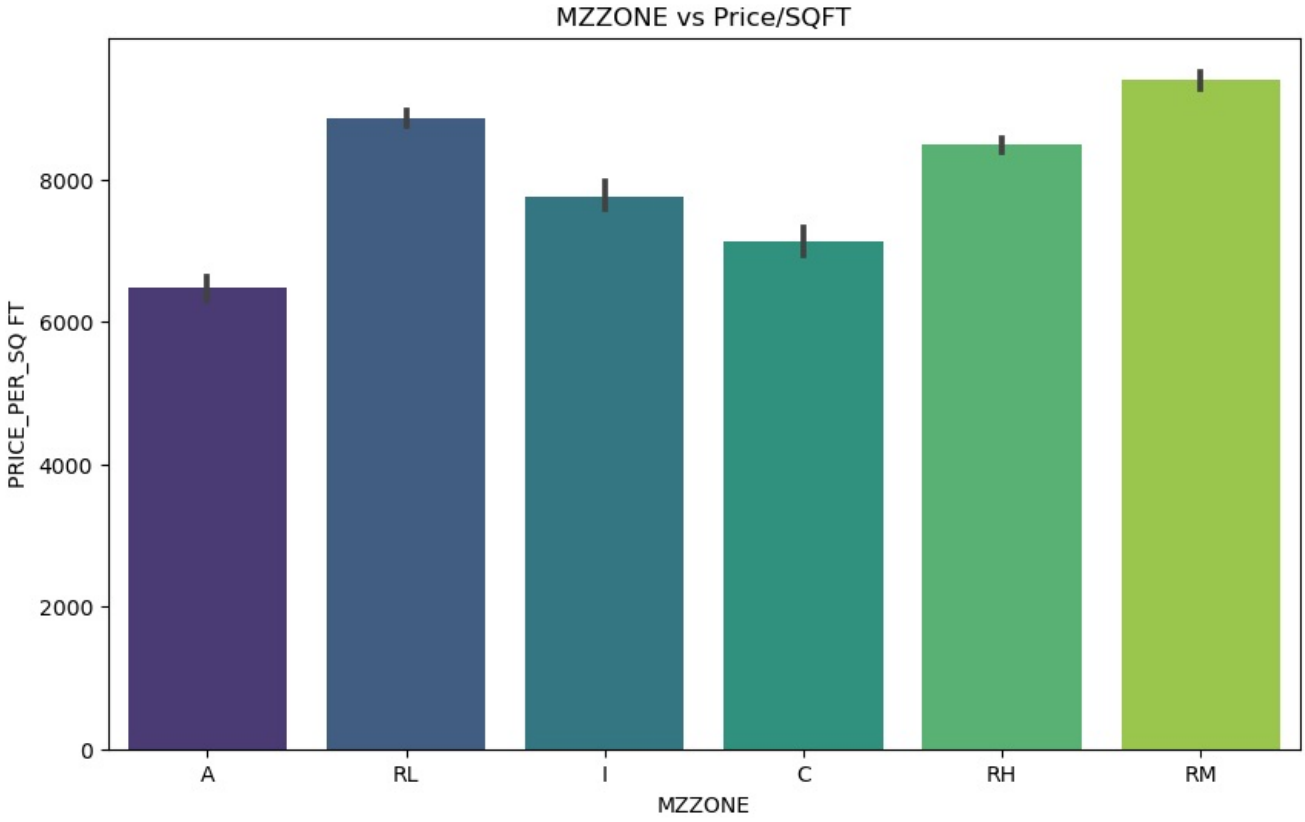


In [106…

```
# MZZONE vs Price/SQFT

fig = plt.figure(figsize =(10,6))
sns.barplot(x='MZZONE',y= 'PRICE_PER_SQ FT', data = df5 , color ='b', palette = "viridis")
plt.title('MZZONE vs Price/SQFT')
```

Out[106]:

```
Text(0.5, 1.0, 'MZZONE vs Price/SQFT')
```



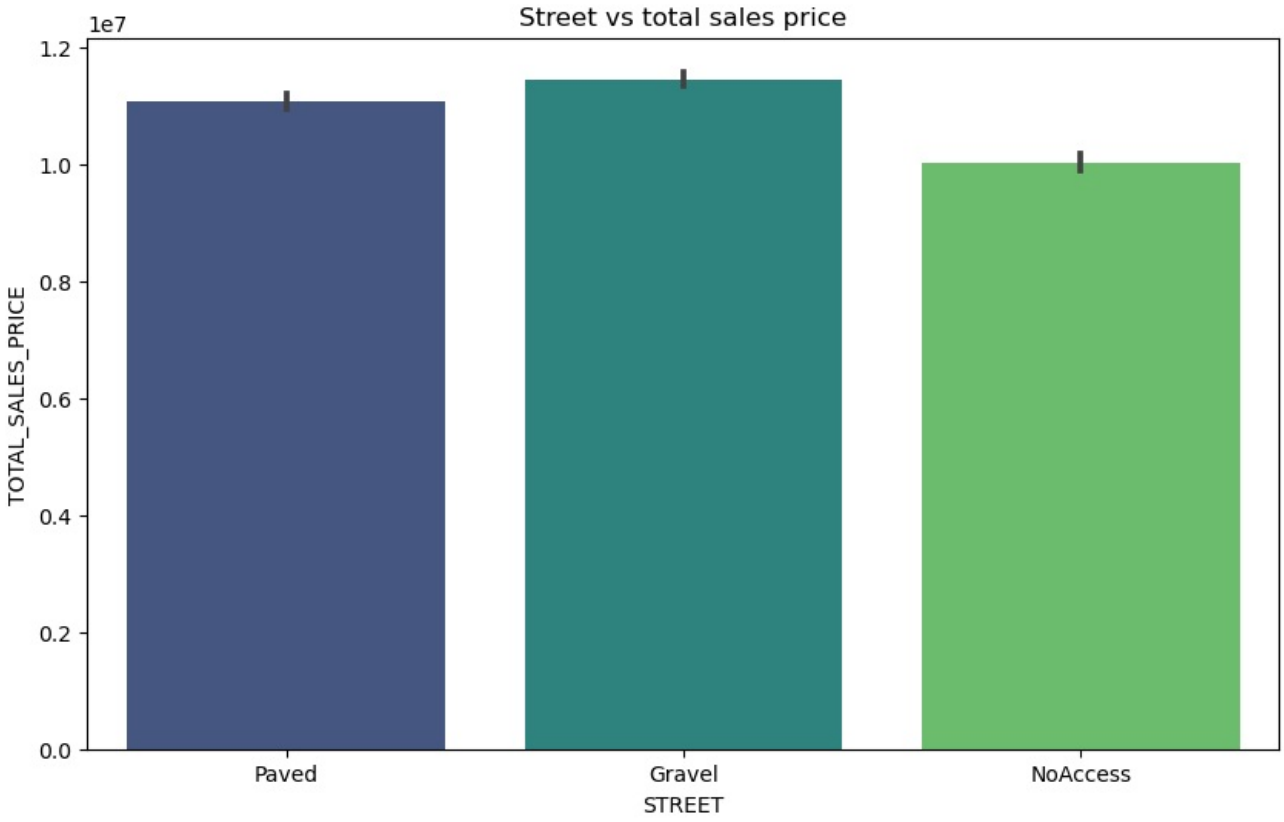
MZZONE tells about the status of society - A is the cheaper,RL, RH and RM comes under expensive

In [107…

```
# Street vs total sales price

fig = plt.figure(figsize =(10,6))
sns.barplot(x='STREET',y= 'TOTAL_SALES_PRICE', data = df5 , color ='b', palette = "viridis")
plt.title('Street vs total sales price')
```

Out[107]:

```
Text(0.5, 1.0, 'Street vs total sales price')
```

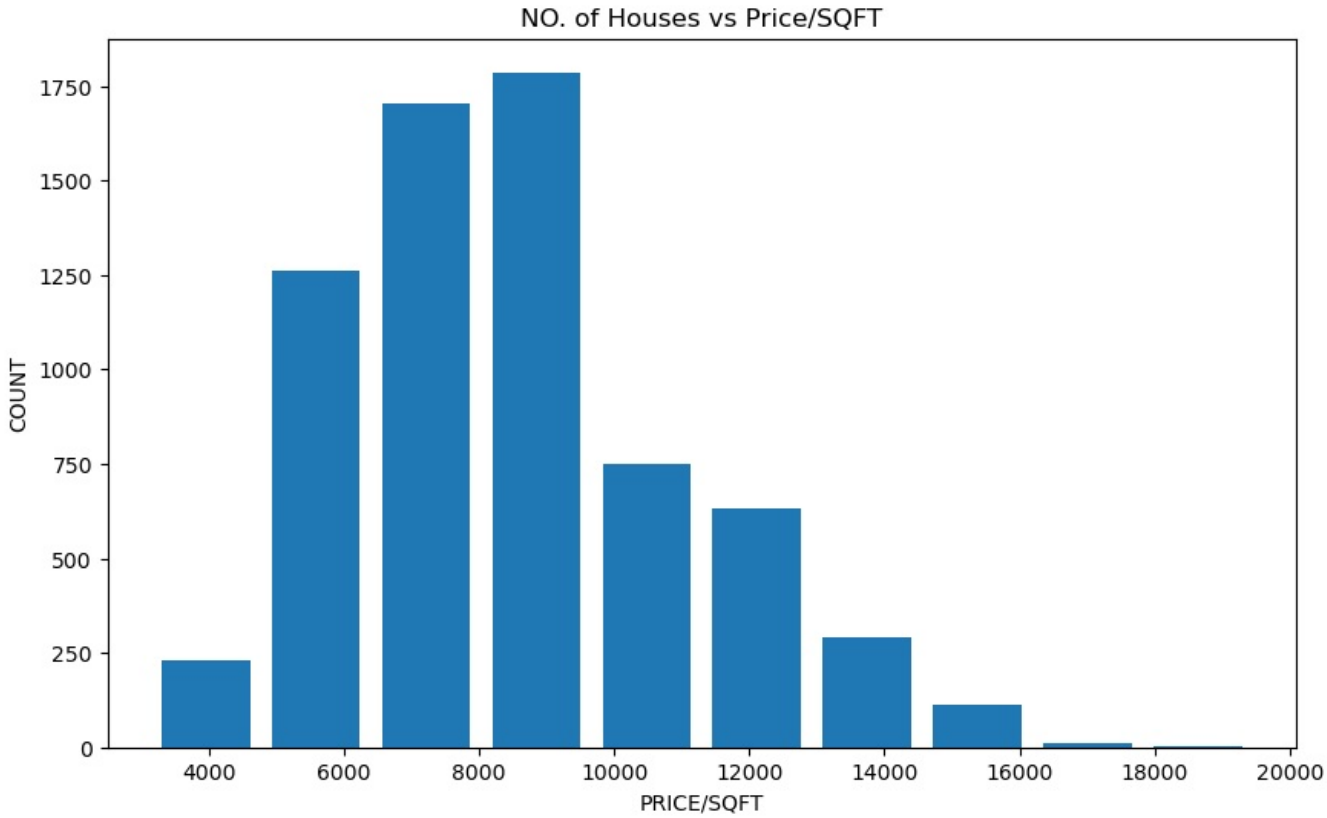


In [108…

```
# NO. of Houses vs Price/SQFT

fig = plt.figure(figsize =(10,6))
plt.hist('PRICE_PER_SQ_FT', data = df5, rwidth = 0.8)
plt.xlabel('PRICE/SQFT')
plt.ylabel('COUNT')
plt.title('NO. of Houses vs Price/SQFT')
```

Out[108]:

```
Text(0.5, 1.0, 'NO. of Houses vs Price/SQFT')
```

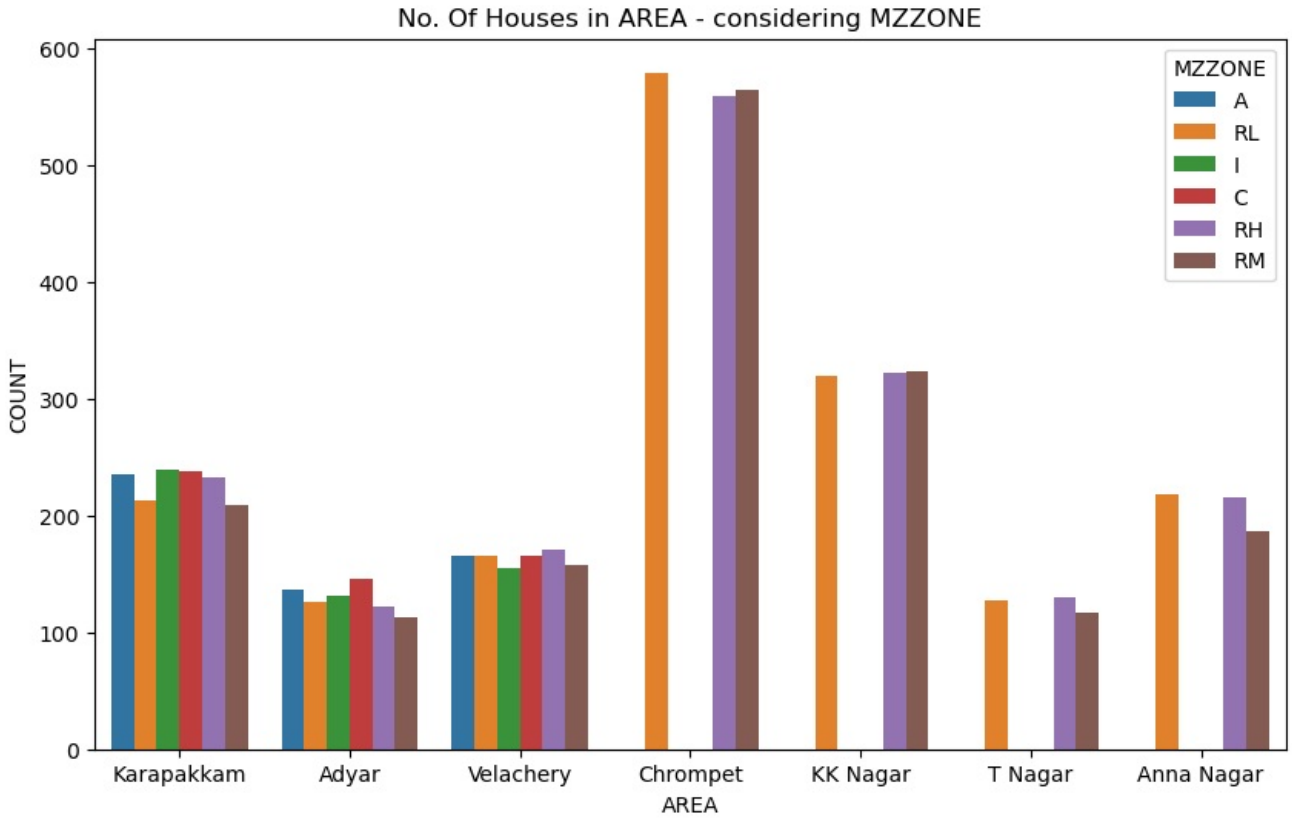


1. Around 4700 houses cost is in range of 5000-9000 Rs/ SQFT

2. 100-150 houses cost is in range of 14000+ RS/SQFT

In [109…

```
# No. Of Houses in AREA - considering MZZONE

plt.figure(figsize =(10,6))
sns.countplot(x='AREA', data = df5, hue = 'MZZONE')
plt.xlabel('AREA')
plt.ylabel('COUNT')
plt.title('No. Of Houses in AREA - considering MZZONE')
plt.show()
```



In [110…
```
df5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6782 entries, 0 to 7108
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   AREA               6782 non-null   object
 1   INT_SQFT           6782 non-null   int64
 2   DIST_MAINROAD      6782 non-null   int64
 3   N_BEDROOM          6782 non-null   int32
 4   N_BATHROOM         6782 non-null   int32
 5   N_ROOM             6782 non-null   int64
 6   SALE_COND          6782 non-null   object
 7   PARK_FACIL         6782 non-null   object
 8   BUILDTYPE          6782 non-null   object
 9   STREET             6782 non-null   object
 10  MZZONE             6782 non-null   object
 11  TOTAL_SALES_PRICE  6782 non-null   int64
 12  BUILD YEAR         6782 non-null   object
 13  SALE_YEAR          6782 non-null   object
 14  UTILITY            6782 non-null   object
 15  PRICE_PER_SQ FT    6782 non-null   float64
dtypes: float64(1), int32(2), int64(4), object(9)
memory usage: 1.1+ MB
```

In [111…
```
# Converting numeric values in to int data type
```

In [112…
```
df5[['BUILD YEAR', 'SALE_YEAR']] = df5[['BUILD YEAR', 'SALE_YEAR']].apply(pd.to_numeric)
```

In [113…
```
df5[['N_BATHROOM', 'N_BEDROOM']] = df5[['N_BATHROOM', 'N_BEDROOM']].astype(int)
```

In [114…
```
df5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 6782 entries, 0 to 7108
Data columns (total 16 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   AREA               6782 non-null   object 
 1   INT_SQFT           6782 non-null   int64  
 2   DIST_MAINROAD      6782 non-null   int64  
 3   N_BEDROOM          6782 non-null   int32  
 4   N_BATHROOM         6782 non-null   int32  
 5   N_ROOM             6782 non-null   int64  
 6   SALE_COND          6782 non-null   object 
 7   PARK_FACIL         6782 non-null   object 
 8   BUILDTYPE          6782 non-null   object 
 9   STREET             6782 non-null   object 
 10  MZZONE             6782 non-null   object 
 11  TOTAL_SALES_PRICE  6782 non-null   int64  
 12  BUILD YEAR         6782 non-null   int64  
 13  SALE_YEAR          6782 non-null   int64  
 14  UTILITY            6782 non-null   object 
 15  PRICE_PER_SQ FT    6782 non-null   float64
dtypes: float64(1), int32(2), int64(6), object(7)
memory usage: 1.1+ MB
```

In [115…

```
df5['PROP_AGE'] = df5['SALE_YEAR']-df5['BUILD YEAR']
```

In [116…

```
df5['PROP_AGE']
```

Out[116]:

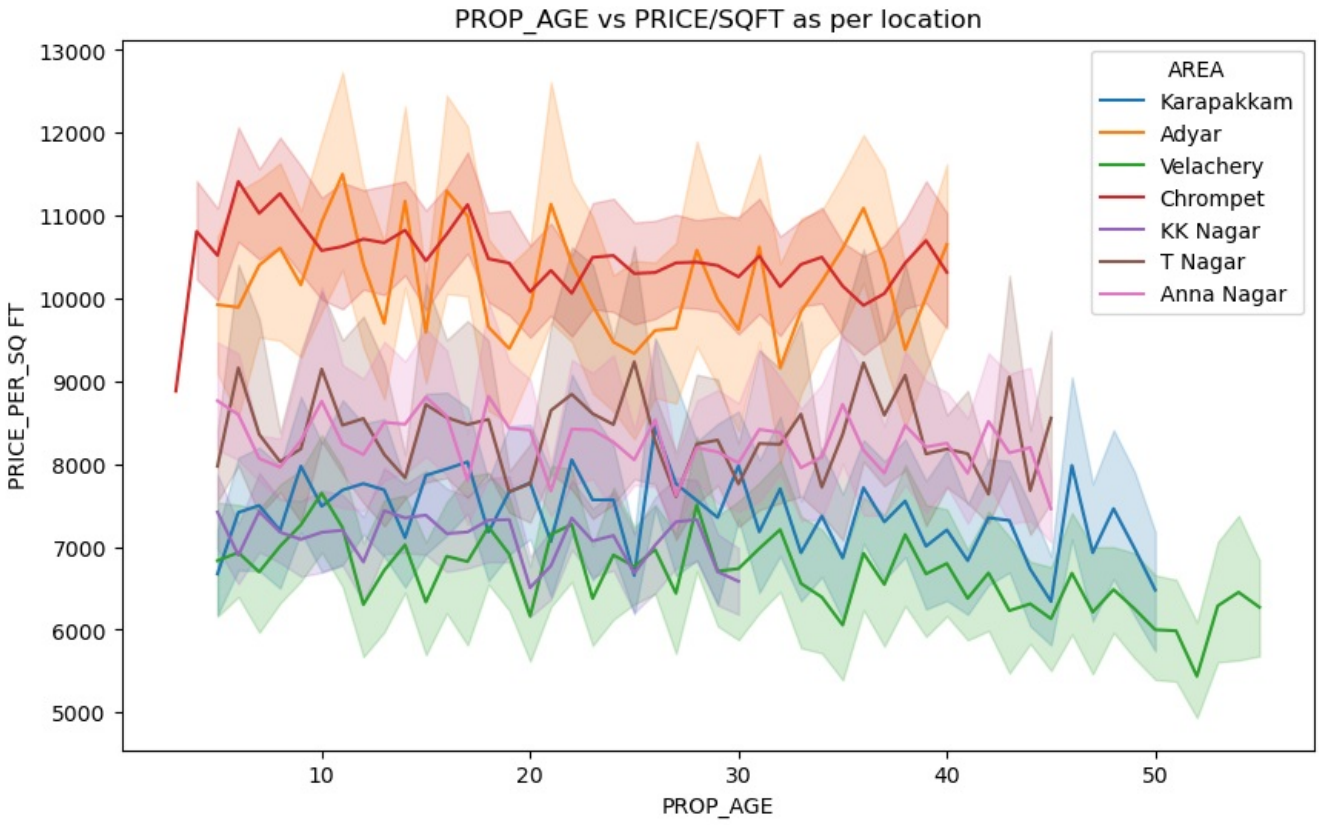
```
0       44
2       20
3       22
4       30
5        5
        ..
7104    49
7105     9
7106    28
7107    32
7108    44
Name: PROP_AGE, Length: 6782, dtype: int64
```

In [117…

```
# Property age vs Price/sqft as per location

fig = plt.figure(figsize =(10,6))
sns.lineplot(x='PROP_AGE',y='PRICE_PER_SQ FT', data = df5, hue = 'AREA')
plt.title('PROP_AGE vs PRICE/SQFT as per location')
```

Out[117]:

```
Text(0.5, 1.0, 'PROP_AGE vs PRICE/SQFT as per location')
```

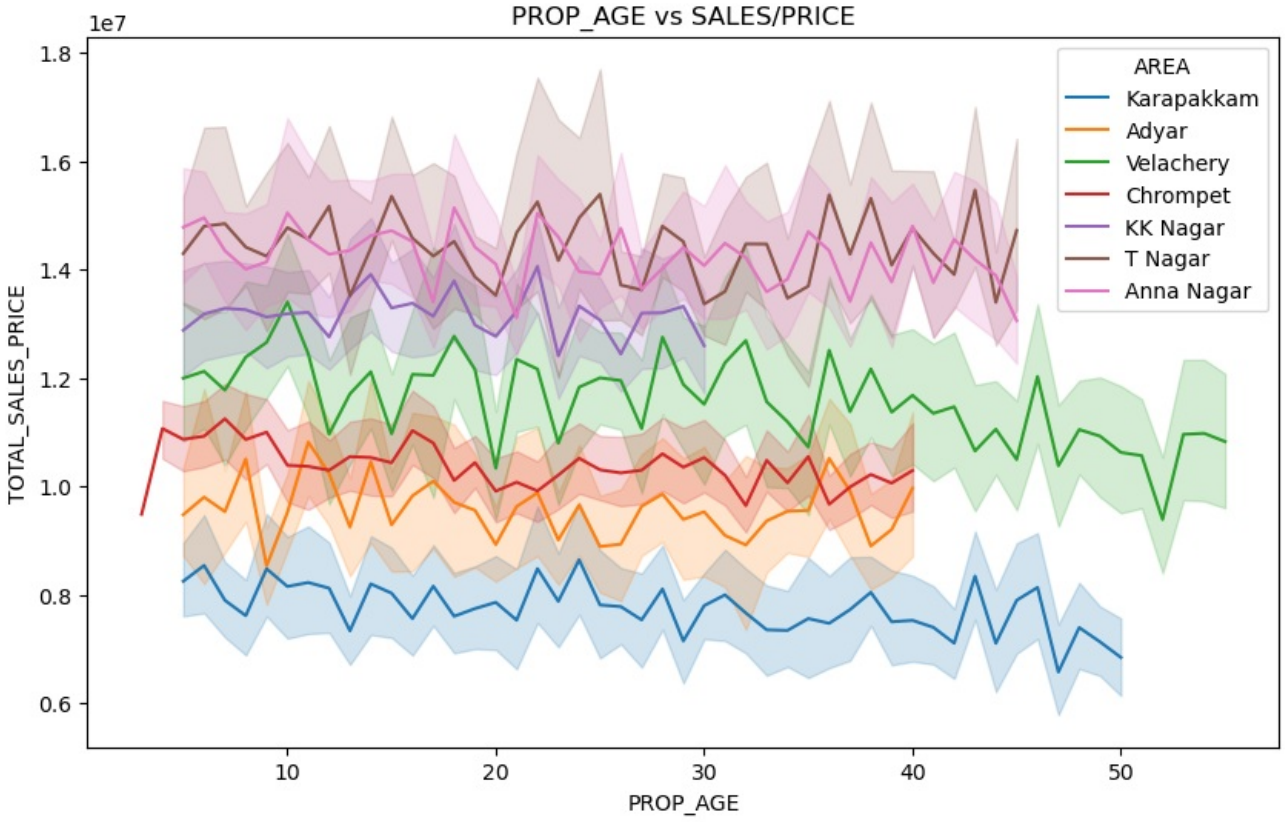


In [118…

```
fig = plt.figure(figsize =(10,6))
sns.lineplot(x='PROP_AGE',y='TOTAL_SALES_PRICE', data =df5, hue = 'AREA')
plt.title('PROP_AGE vs SALES/PRICE')
```

Out[118]:

```
Text(0.5, 1.0, 'PROP_AGE vs SALES/PRICE')
```



In [119…
```
# preparing data for model creation now

df6 = df5.copy()
```

In [120…
```
df6.head()
```

Out[120]:

	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	STREET	MZ
0	Karapakkam	1004	131	1	1	3	AbNormal	Yes	Commercial	Paved	
2	Adyar	909	70	1	1	3	AbNormal	Yes	Commercial	Gravel	
3	Velachery	1855	14	3	2	5	Family	No	Others	Paved	
4	Karapakkam	1226	84	1	1	3	AbNormal	Yes	Others	Gravel	
5	Chrompet	1220	36	2	1	4	Partial	No	Commercial	NoAccess	

In [121…
```
# Dropping irreleveant columns
df6.drop(['DIST_MAINROAD','MZZONE','BUILD YEAR','SALE_YEAR'],axis =1, inplace =True)
```

In [122…
```
# Checking shape of column
df6.shape
```

Out[122]:
```
(6782, 13)
```

In [123…
```
# checking dataframe after removing columns
df6.head()
```

Out[123]:

	AREA	INT_SQFT	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	STREET	TOTAL_SALES_PRICE
0	Karapakkam	1004	1	1	3	AbNormal	Yes	Commercial	Paved	8124400
2	Adyar	909	1	1	3	AbNormal	Yes	Commercial	Gravel	1367240
3	Velachery	1855	3	2	5	Family	No	Others	Paved	1006365
4	Karapakkam	1226	1	1	3	AbNormal	Yes	Others	Gravel	7717313
5	Chrompet	1220	2	1	4	Partial	No	Commercial	NoAccess	1300209

Convert categorical columns in numerical data

- Methods
- 1. replace method
- 1. Label Encoding(Binary caegories)
- 1. ONE HOT ENCODING(pd.get_dummies)
- 1. Count /Frequency Encoding m

In [124…

```
# creating dataframe copy in to df7

df7 = df6.copy()
```

In [125…

```
df7.head()
```

Out[125]:

	AREA	INT_SQFT	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	STREET	TOTAL_SALES_PRICE
0	Karapakkam	1004	1	1	3	AbNormal	Yes	Commercial	Paved	8124400
2	Adyar	909	1	1	3	AbNormal	Yes	Commercial	Gravel	1367240
3	Velachery	1855	3	2	5	Family	No	Others	Paved	1006365
4	Karapakkam	1226	1	1	3	AbNormal	Yes	Others	Gravel	7717313
5	Chrompet	1220	2	1	4	Partial	No	Commercial	NoAccess	1300209

In [126…

```
dummies = pd.get_dummies(data=
df7[['PARK_FACIL','AREA','SALE_COND','BUILDTYPE','STREET','UTILITY']], drop_first =True)
```

In [127…

```
dummies
```

Out[127]:

	PARK_FACIL_Yes	AREA_Anna Nagar	AREA_Chrompet	AREA_KK Nagar	AREA_Karapakkam	AREA_T Nagar	AREA_Velachery	SALE_COND_AdjLand	SA
0	1	0	0	0	1	0	0	0	
2	1	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	1	0	
4	1	0	0	0	1	0	0	0	
5	0	0	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	
7104	0	0	0	0	1	0	0	1	
7105	1	0	0	0	0	0	1	0	
7106	0	0	0	0	0	0	1	0	
7107	1	0	0	0	1	0	0	0	
7108	1	0	0	0	0	0	1	0	

6782 rows × 17 columns

In [128…

```
dummies.columns
```

Out[128]:

```
Index(['PARK_FACIL_Yes', 'AREA_Anna Nagar', 'AREA_Chrompet', 'AREA_KK Nagar',
       'AREA_Karapakkam', 'AREA_T Nagar', 'AREA_Velachery',
       'SALE_COND_AdjLand', 'SALE_COND_Family', 'SALE_COND_Normal Sale',
       'SALE_COND_Partial', 'BUILDTYPE_House', 'BUILDTYPE_Others',
       'STREET_NoAccess', 'STREET_Paved', 'UTILITY_ELO', 'UTILITY_NoSewa'],
      dtype='object')
```

In [129…

```
dummies.shape
```

Out[129]:
```
(6782, 17)
```

In [130…
```
dummies.columns
```

Out[130]:
```
Index(['PARK_FACIL_Yes', 'AREA_Anna Nagar', 'AREA_Chrompet', 'AREA_KK Nagar',
       'AREA_Karapakkam', 'AREA_T Nagar', 'AREA_Velachery',
       'SALE_COND_AdjLand', 'SALE_COND_Family', 'SALE_COND_Normal Sale',
       'SALE_COND_Partial', 'BUILDTYPE_House', 'BUILDTYPE_Others',
       'STREET_NoAccess', 'STREET_Paved', 'UTILITY_ELO', 'UTILITY_NoSewa'],
      dtype='object')
```

In [131…
```
dummies
```

Out[131]:

	PARK_FACIL_Yes	AREA_Anna Nagar	AREA_Chrompet	AREA_KK Nagar	AREA_Karapakkam	AREA_T Nagar	AREA_Velachery	SALE_COND_AdjLand	SA
0	1	0	0	0	1	0	0	0	
2	1	0	0	0	0	0	0	0	
3	0	0	0	0	0	0	1	0	
4	1	0	0	0	1	0	0	0	
5	0	0	1	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	
7104	0	0	0	0	1	0	0	1	
7105	1	0	0	0	0	0	1	0	
7106	0	0	0	0	0	0	1	0	
7107	1	0	0	0	1	0	0	0	
7108	1	0	0	0	0	0	1	0	

6782 rows × 17 columns

In [132…
```
# inserting dummies in to new dataframe

df8 = pd.concat([df7,dummies],axis = 1)
```

In [133…
```
df8.head()
```

Out[133]:

	AREA	INT_SQFT	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE	STREET	TOTAL_SALES_PRICE
0	Karapakkam	1004	1	1	3	AbNormal	Yes	Commercial	Paved	8124400
2	Adyar	909	1	1	3	AbNormal	Yes	Commercial	Gravel	13672408
3	Velachery	1855	3	2	5	Family	No	Others	Paved	10063653
4	Karapakkam	1226	1	1	3	AbNormal	Yes	Others	Gravel	7717313
5	Chrompet	1220	2	1	4	Partial	No	Commercial	NoAccess	13002093

5 rows × 30 columns

In [134…
```
df9 = df8.copy()
```

In [135…
```
# dropping duplicate columns

df9 = df8.drop(['PARK_FACIL','AREA','SALE_COND','BUILDTYPE','STREET','UTILITY','PRICE_PER_SQ
FT'],axis = 1)
```

In [136…
```
df9.columns
```

Out[136]:
```
Index(['INT_SQFT', 'N_BEDROOM', 'N_BATHROOM', 'N_ROOM', 'TOTAL_SALES_PRICE',
       'PROP_AGE', 'PARK_FACIL_Yes', 'AREA_Anna Nagar', 'AREA_Chrompet',
       'AREA_KK Nagar', 'AREA_Karapakkam', 'AREA_T Nagar', 'AREA_Velachery',
       'SALE_COND_AdjLand', 'SALE_COND_Family', 'SALE_COND_Normal Sale',
       'SALE_COND_Partial', 'BUILDTYPE_House', 'BUILDTYPE_Others',
       'STREET_NoAccess', 'STREET_Paved', 'UTILITY_ELO', 'UTILITY_NoSewa'],
      dtype='object')
```

In [137…
```
df9.shape
```

Out[137]:
```
(6782, 23)
```

In [138…
```
# resetting index of dataframe df9
df9 = df9.reset_index(drop =True)
```

In [139…
```
df9
```

Out[139]:

	INT_SQFT	N_BEDROOM	N_BATHROOM	N_ROOM	TOTAL_SALES_PRICE	PROP_AGE	PARK_FACIL_Yes	AREA_Anna Nagar	AREA_Chromp
0	1004	1	1	3	8124400	44	1	0	
1	909	1	1	3	13672408	20	1	0	
2	1855	3	2	5	10063653	22	0	0	
3	1226	1	1	3	7717313	30	1	0	
4	1220	2	1	4	13002093	5	0	0	
...	...	...	...	...	...	...	...	...	
6777	598	1	1	2	5668827	49	0	0	
6778	1897	3	2	5	11370222	9	1	0	
6779	1614	2	1	4	8835792	28	0	0	
6780	787	1	1	2	9051448	32	1	0	
6781	1896	3	2	5	10405469	44	1	0	

6782 rows × 23 columns

In [140…
```
# Now dropping more unnecessary columns

df10 = df9.copy()
df10
```

Out[140]:

	INT_SQFT	N_BEDROOM	N_BATHROOM	N_ROOM	TOTAL_SALES_PRICE	PROP_AGE	PARK_FACIL_Yes	AREA_Anna Nagar	AREA_Chromp
0	1004	1	1	3	8124400	44	1	0	
1	909	1	1	3	13672408	20	1	0	
2	1855	3	2	5	10063653	22	0	0	
3	1226	1	1	3	7717313	30	1	0	
4	1220	2	1	4	13002093	5	0	0	
...	...	...	...	...	...	...	...	...	
6777	598	1	1	2	5668827	49	0	0	
6778	1897	3	2	5	11370222	9	1	0	
6779	1614	2	1	4	8835792	28	0	0	
6780	787	1	1	2	9051448	32	1	0	
6781	1896	3	2	5	10405469	44	1	0	

6782 rows × 23 columns

In [141…
```
df10.columns
```

Out[141]:
```
Index(['INT_SQFT', 'N_BEDROOM', 'N_BATHROOM', 'N_ROOM', 'TOTAL_SALES_PRICE',
       'PROP_AGE', 'PARK_FACIL_Yes', 'AREA_Anna Nagar', 'AREA_Chrompet',
       'AREA_KK Nagar', 'AREA_Karapakkam', 'AREA_T Nagar', 'AREA_Velachery',
       'SALE_COND_AdjLand', 'SALE_COND_Family', 'SALE_COND_Normal Sale',
       'SALE_COND_Partial', 'BUILDTYPE_House', 'BUILDTYPE_Others',
       'STREET_NoAccess', 'STREET_Paved', 'UTILITY_ELO', 'UTILITY_NoSewa'],
      dtype='object')
```

In [142…
```
df10.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6782 entries, 0 to 6781
Data columns (total 23 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   INT_SQFT              6782 non-null   int64
 1   N_BEDROOM             6782 non-null   int32
 2   N_BATHROOM            6782 non-null   int32
 3   N_ROOM                6782 non-null   int64
 4   TOTAL_SALES_PRICE     6782 non-null   int64
 5   PROP_AGE              6782 non-null   int64
 6   PARK_FACIL_Yes        6782 non-null   uint8
 7   AREA_Anna Nagar       6782 non-null   uint8
 8   AREA_Chrompet         6782 non-null   uint8
 9   AREA_KK Nagar         6782 non-null   uint8
 10  AREA_Karapakkam       6782 non-null   uint8
 11  AREA_T Nagar          6782 non-null   uint8
 12  AREA_Velachery        6782 non-null   uint8
 13  SALE_COND_AdjLand     6782 non-null   uint8
 14  SALE_COND_Family      6782 non-null   uint8
 15  SALE_COND_Normal Sale 6782 non-null   uint8
 16  SALE_COND_Partial     6782 non-null   uint8
 17  BUILDTYPE_House       6782 non-null   uint8
 18  BUILDTYPE_Others      6782 non-null   uint8
 19  STREET_NoAccess       6782 non-null   uint8
 20  STREET_Paved          6782 non-null   uint8
 21  UTILITY_ELO           6782 non-null   uint8
 22  UTILITY_NoSewa        6782 non-null   uint8
dtypes: int32(2), int64(4), uint8(17)
memory usage: 377.6 KB
```

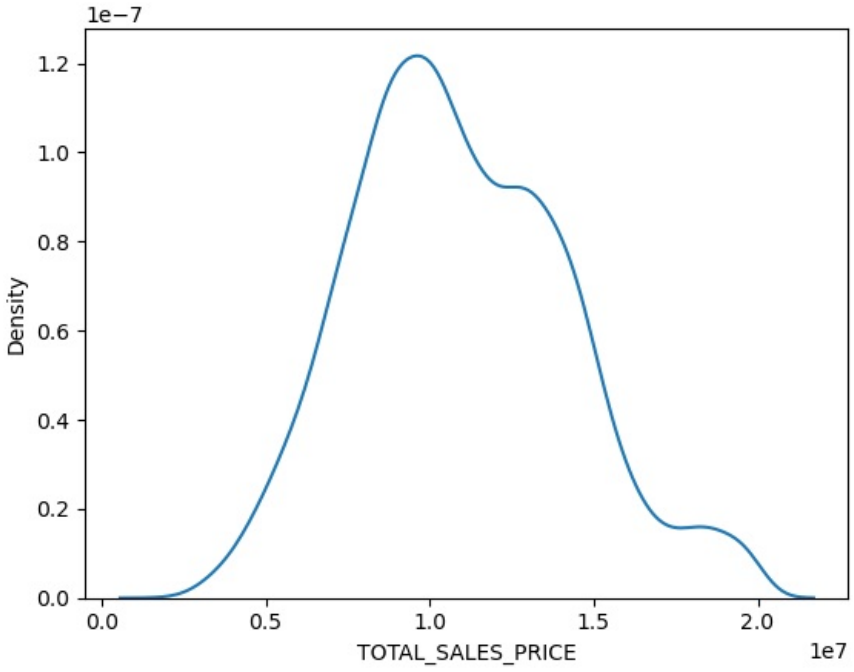
Model building

In [144…

```
# checking distribution of Sales price
sns.kdeplot(df10['TOTAL_SALES_PRICE'])
```

Out[144]:

```
<Axes: xlabel='TOTAL_SALES_PRICE', ylabel='Density'>
```

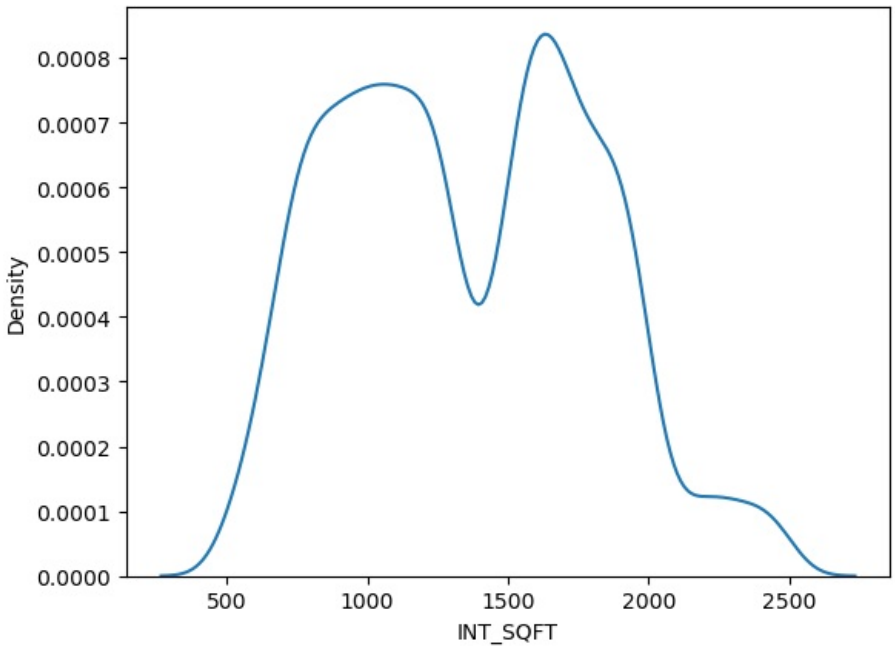


In [145…

```
# checking distribution of INT_SQFT
sns.kdeplot(df10['INT_SQFT'])
```

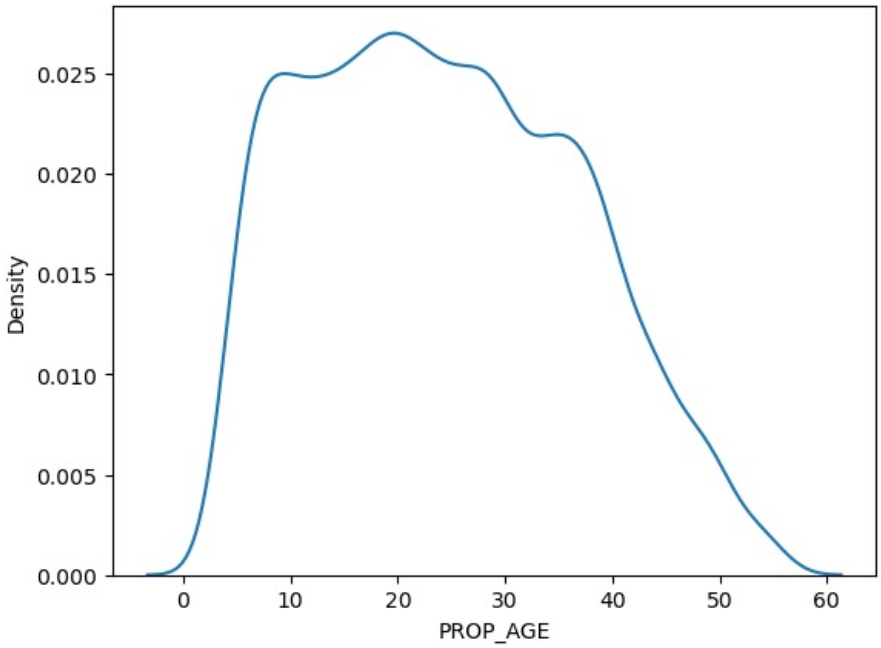
Out[145]:

```
<Axes: xlabel='INT_SQFT', ylabel='Density'>
```



In [146…
```
# checking distribution of Property Age
sns.kdeplot(df10['PROP_AGE'])
```

Out[146]:
```
<Axes: xlabel='PROP_AGE', ylabel='Density'>
```



In [147…
```
# To get better accuracy we will standardize our data
```

In [148…
```
X = df10.drop(['TOTAL_SALES_PRICE'], axis = 1)
```

In [149…
```
y = df10['TOTAL_SALES_PRICE']
```

In [150…
```
# Splittig data using train test split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

In [151…
```
# General split
```

```
X_train,X_test,y_train,y_test
```

Out[151]:

```
(      INT_SQFT  N_BEDROOM  N_BATHROOM  N_ROOM  PROP_AGE  PARK_FACIL_Yes  \
6586      1269          2           1       4        25               0
1886      1543          1           1       4        39               1
5017      1645          1           1       4        39               0
148        928          1           1       3         4               1
6232       787          1           1       2        39               0
...        ...        ...         ...     ...       ...             ...
1775      1338          2           2       4         8               1
5764       889          1           1       3        23               1
6095       956          1           1       3        29               0
4605      1900          3           2       5        21               0
5316      1555          2           1       4        26               0

      AREA_Anna Nagar  AREA_Chrompet  AREA_KK Nagar  AREA_Karapakkam  ...  \
6586                0              1              0                0  ...
1886                0              0              0                0  ...
5017                1              0              0                0  ...
148                 0              1              0                0  ...
6232                0              0              0                0  ...
...               ...            ...            ...              ...  ...
1775                0              0              0                1  ...
5764                0              1              0                0  ...
6095                0              1              0                0  ...
4605                0              0              0                0  ...
5316                0              0              0                0  ...

      SALE_COND_AdjLand  SALE_COND_Family  SALE_COND_Normal Sale  \
6586                  0                 0                      0
1886                  0                 0                      0
5017                  0                 1                      0
148                   0                 0                      0
6232                  1                 0                      0
...                 ...               ...                    ...
1775                  0                 1                      0
5764                  0                 0                      1
6095                  1                 0                      0
4605                  0                 0                      1
5316                  0                 0                      1

      SALE_COND_Partial  BUILDTYPE_House  BUILDTYPE_Others  STREET_NoAccess  \
6586                  1                0                 0                0
1886                  1                0                 0                0
5017                  0                1                 0                1
148                   1                0                 0                0
6232                  0                0                 1                0
...                 ...              ...               ...              ...
1775                  0                0                 1                0
5764                  0                0                 0                0
6095                  0                0                 1                0
4605                  0                1                 0                1
5316                  0                0                 0                1

      STREET_Paved  UTILITY_ELO  UTILITY_NoSewa
6586             1            0               1
1886             0            0               1
5017             0            0               0
148              0            0               0
6232             0            0               1
...            ...          ...             ...
1775             1            0               1
5764             1            0               1
6095             0            0               0
4605             0            1               0
5316             0            0               0

[5086 rows x 22 columns],
      INT_SQFT  N_BEDROOM  N_BATHROOM  N_ROOM  PROP_AGE  PARK_FACIL_Yes  \
2257       961          1           1       3        33               0
5097      1592          2           2       4         7               0
1276      1485          2           2       4        32               1
4         1220          2           1       4         5               0
4996       844          1           1       2        12               0
...        ...        ...         ...     ...       ...             ...
346        863          1           1       3         7               1
2648       826          1           1       3        12               0
6698      1617          1           1       4         7               0
1015      1231          2           1       4        22               0
```

```
1792      1040          1           1       3        20               1

      AREA_Anna Nagar  AREA_Chrompet  AREA_KK Nagar  AREA_Karapakkam  ...  \
2257                0              1              0                0  ...
5097                0              0              0                1  ...
1276                0              0              0                1  ...
4                   0              1              0                0  ...
4996                0              0              0                1  ...
...               ...            ...            ...              ...  ...
346                 0              0              0                0  ...
2648                0              1              0                0  ...
6698                1              0              0                0  ...
1015                0              1              0                0  ...
1792                0              0              0                1  ...

      SALE_COND_AdjLand  SALE_COND_Family  SALE_COND_Normal Sale  \
2257                  0                 0                      1
5097                  1                 0                      0
1276                  0                 0                      0
4                     0                 0                      0
4996                  0                 1                      0
...                 ...               ...                    ...
346                   0                 0                      0
2648                  0                 1                      0
6698                  0                 0                      0
1015                  0                 0                      0
1792                  0                 0                      1

      SALE_COND_Partial  BUILDTYPE_House  BUILDTYPE_Others  STREET_NoAccess  \
2257                  0                0                 1                0
5097                  0                0                 1                0
1276                  0                1                 0                1
4                     1                0                 0                1
4996                  0                0                 0                0
...                 ...              ...               ...              ...
346                   0                1                 0                0
2648                  0                1                 0                0
6698                  1                0                 1                0
1015                  1                0                 1                0
1792                  0                0                 0                0

      STREET_Paved  UTILITY_ELO  UTILITY_NoSewa
2257             1            0               1
5097             0            1               0
1276             0            1               0
4                0            0               1
4996             0            0               1
...            ...          ...             ...
346              1            0               1
2648             0            0               0
6698             1            0               0
1015             0            0               0
1792             0            1               0

[1696 rows x 22 columns],
6586    14162276
1886    19884934
5017    10470291
148     12895904
6232     7829085
          ...
1775     7761373
5764    12054702
6095    10377411
4605    10403223
5316    13678371
Name: TOTAL_SALES_PRICE, Length: 5086, dtype: int64,
2257     8704803
5097     8619678
1276     8485690
4       13002093
4996     6997612
          ...
346      6895848
2648     7804314
6698    13578437
1015     9995874
1792    10064484
Name: TOTAL_SALES_PRICE, Length: 1696, dtype: int64)
```

In [152…
```
# initializing scaler object

scaler = StandardScaler()
```

In [153…
```
# standardize scaling split data

x_train_std = scaler.fit_transform(X_train)
x_test_std =  scaler.transform(X_test)
```

In [154…
```
# creating scaled dataframe of x_train_std array

x_train_std_df = pd.DataFrame(x_train_std, columns = X_train.columns)
```

In [155…
```
x_train_std_df
```

Out[155]:

	INT_SQFT	N_BEDROOM	N_BATHROOM	N_ROOM	PROP_AGE	PARK_FACIL_Yes	AREA_Anna Nagar	AREA_Chrompet	AREA_KK Nagar	AREA_K
0	-0.203720	0.455018	-0.529892	0.352164	0.050691	-0.987495	-0.316091	1.714521	-0.414595	
1	0.400631	-0.785852	-0.529892	0.352164	1.178628	1.012664	-0.316091	-0.583253	-0.414595	
2	0.625608	-0.785852	-0.529892	0.352164	1.178628	-0.987495	3.163646	-0.583253	-0.414595	
3	-0.955850	-0.785852	-0.529892	-0.637398	-1.641215	1.012664	-0.316091	1.714521	-0.414595	
4	-1.266848	-0.785852	-0.529892	-1.626960	1.178628	-0.987495	-0.316091	-0.583253	-0.414595	
...	...	...	...	...	...	...	...	...	...	
5081	-0.051529	0.455018	1.887176	0.352164	-1.318947	1.012664	-0.316091	-0.583253	-0.414595	
5082	-1.041870	-0.785852	-0.529892	-0.637398	-0.110443	1.012664	-0.316091	1.714521	-0.414595	
5083	-0.894091	-0.785852	-0.529892	-0.637398	0.372959	-0.987495	-0.316091	1.714521	-0.414595	
5084	1.188051	1.695888	1.887176	1.341726	-0.271577	-0.987495	-0.316091	-0.583253	-0.414595	
5085	0.427099	0.455018	-0.529892	0.352164	0.131258	-0.987495	-0.316091	-0.583253	-0.414595	

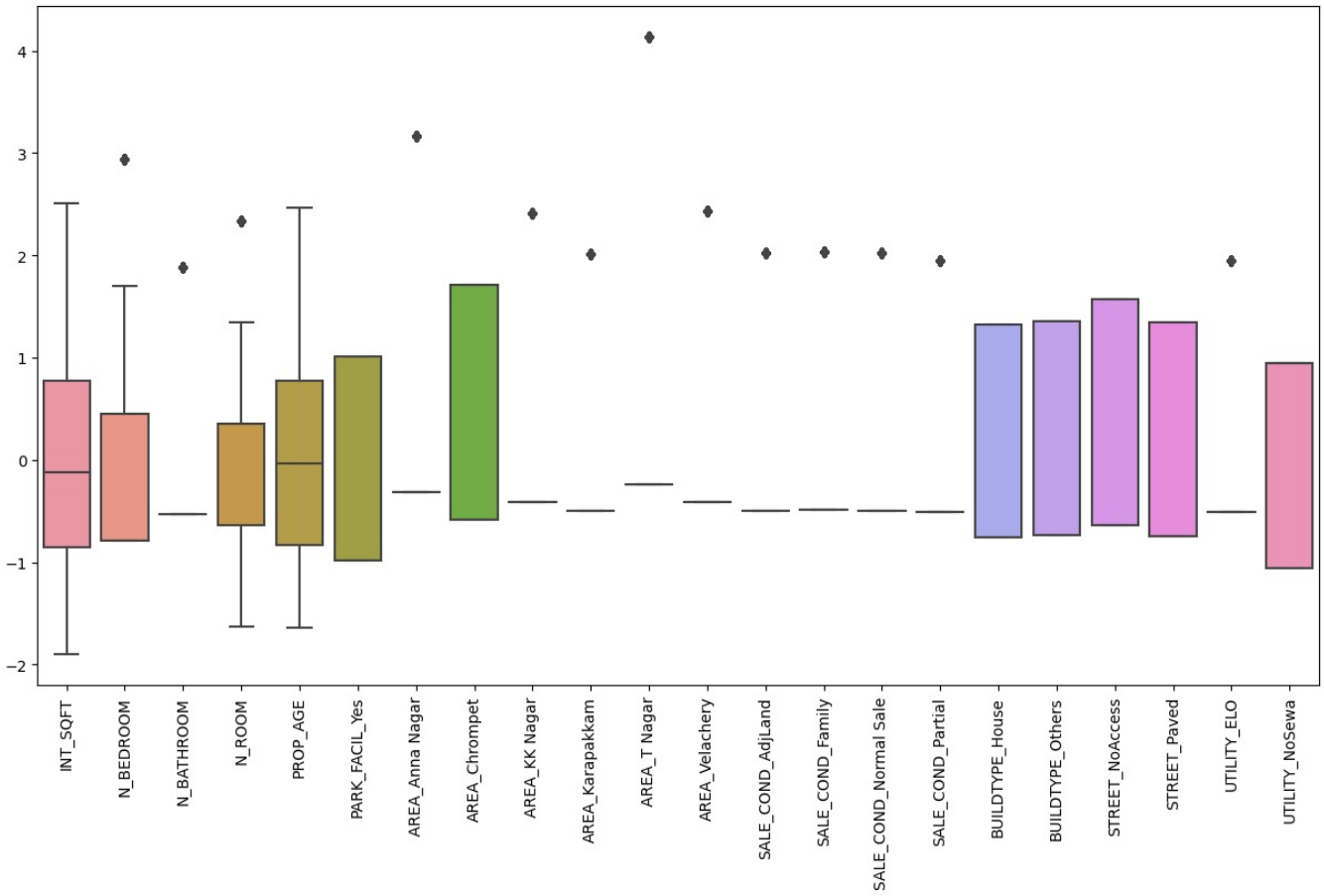
5086 rows × 22 columns

In [156…
```
x_train_std_df.columns
```

Out[156]:
```
Index(['INT_SQFT', 'N_BEDROOM', 'N_BATHROOM', 'N_ROOM', 'PROP_AGE',
       'PARK_FACIL_Yes', 'AREA_Anna Nagar', 'AREA_Chrompet', 'AREA_KK Nagar',
       'AREA_Karapakkam', 'AREA_T Nagar', 'AREA_Velachery',
       'SALE_COND_AdjLand', 'SALE_COND_Family', 'SALE_COND_Normal Sale',
       'SALE_COND_Partial', 'BUILDTYPE_House', 'BUILDTYPE_Others',
       'STREET_NoAccess', 'STREET_Paved', 'UTILITY_ELO', 'UTILITY_NoSewa'],
      dtype='object')
```

In [157…
```
# checking outliers in standard scaled data

fig = plt.figure(figsize =(15,8))
plt.xticks(rotation=90)
sns.boxplot(data = x_train_std_df)
```

Out[157]:
```
<Axes: >
```



Model without scaling

In [158…

```
# linear model object

lr = LinearRegression()
# training model
lr.fit(X_train, y_train)
y_pred = lr.predict(X_test)
print(f"Linear Regr-R2 Score without Standard scaling ={r2_score(y_test,y_pred)}")
```

```
Linear Regr-R2 Score without Standard scaling =0.9085312140334688
```

Model after standardscaling

In [159…

```
# linear model object

lr2 = LinearRegression()
# training model
lr2.fit(x_train_std, y_train)
y_pred_ss = lr2.predict(x_test_std)
print(y_pred_ss,'\n')
```

```
print(f"Linear Regr-R2 Score after Standard scaling : {r2_score(y_test,y_pred_ss)}")
```

```
[ 8599361.24764946  9641225.13993502  7538217.37619138 ...
 13364801.02803941 10392013.47079709 11509225.44225706]

Linear Regr-R2 Score after Standard scaling : 0.9085312140334687
```

In case of Linear Regression - all scores are same , we did not get any benefit afte scaling even

#

Model building using Lasso Reg without scaling

In [160…
```
# finding alpha value of lasso equation

lassocv = LassoCV(cv=5, random_state=0, max_iter  = 1000)
lassocv.fit(X_train, y_train)
lassocv.alpha_
```

Out[160]:
```
901224.7223429865
```

In [161…
```
# fitting lasso to data

lasso_G = Lasso(alpha=lassocv.alpha_)
lasso_G.fit(X_train, y_train)
y_pred = lasso_G.predict(X_test)
model_score = r2_score(y_test,y_pred)
print(' Lasso model score :', model_score)
```

```
 Lasso model score : 0.40146541011920533
```

Model building using Lasso Reg with Standardscaling

In [162…
```
# finding alpha value of lasso equation

lassocv = LassoCV(cv=5, random_state=0, max_iter  = 1000)
lassocv.fit(x_train_std, y_train)
lassocv.alpha_
```

Out[162]:
```
1987.7946309608358
```

In [163…
```
# fitting lasso to data

lasso_SS = Lasso(alpha=lassocv.alpha_)
lasso_SS.fit(x_train_std, y_train)
y_pred = lasso_SS.predict(x_test_std)
lasso_SS_score = r2_score(y_test,y_pred)
print(' Lasso model score after standard scaling :', lasso_SS_score)
```

```
 Lasso model score after standard scaling : 0.9085817705935827
```

Model building using Ridge Reg without scaling

In [164…
```
# finding alpha value of ridge equation

ridgecv =RidgeCV( alphas= np.random.uniform(0,10,50), cv= 10)
ridgecv.fit(X_train,y_train)
ridgecv.alpha_
```

Out[164]:
```
0.8074073530237569
```

In [165…
```
# fitting ridge to data

ridge_G = Ridge(alpha = ridgecv.alpha_)
ridge_G.fit(X_train,y_train)
```

```
y_pred = ridge_G.predict(X_test)
ridge_score = r2_score(y_test,y_pred)
print(' ridge model score before standard scaling :', ridge_score)
```

```
ridge model score before standard scaling : 0.9085576687381396
```

Model building using Ridge Reg with Standard scaling

In [166…

```
# finding alpha value of ridge equation

ridgecv =RidgeCV( alphas= np.random.uniform(0,10,50), cv= 10)
ridgecv.fit(x_train_std,y_train)
ridgecv.alpha_
```

Out[166]:

```
3.3276591941795317
```

In [167…

```
# fitting ridge to data

ridge_SS = Ridge(alpha = ridgecv.alpha_)
ridge_SS.fit(x_train_std,y_train)
y_pred = ridge_SS.predict(x_test_std)
ridge_SS_score = r2_score(y_test,y_pred)
print(' ridge model score after standard scaling :', ridge_score)
```

```
ridge model score after standard scaling : 0.9085576687381396
```

Model building using Elasticnet Reg without scaling

In [168…

```
# finding alpha and l1_ration value of elasticnet equation

elastic=ElasticNetCV(l1_ratio = 0.5, cv = 10, max_iter = 10000, random_state= 100)
elastic.fit(X_train,y_train)
elastic.alpha_, elastic.l1_ratio
```

Out[168]:

```
(1802449.4446859732, 0.5)
```

In [169…

```
# fitting elasticnet to data

elastic_G = ElasticNet(alpha=elastic.alpha_,l1_ratio = elastic.l1_ratio)
elastic_G.fit(X_train, y_train)
y_pred = elastic_G.predict(X_test)
elastic_score = r2_score(y_test,y_pred)
print(' elastic model score before standard scaling :',elastic_score)
```

```
elastic model score before standard scaling : 0.1217902941342065
```

Model building using Elasticnet Reg with Standard scaling

In [170…

```
# finding alpha and l1_ration value of elasticnet equation

elastic=ElasticNetCV(l1_ratio = 0.5, cv = 10, max_iter = 10000)
elastic.fit(x_train_std,y_train)
elastic.alpha_, elastic.l1_ratio
```

Out[170]:

```
(3975.589261921672, 0.5)
```

In [171…

```
# fitting elasticnet to data

elastic_SS = ElasticNet(alpha=elastic.alpha_,l1_ratio = elastic.l1_ratio)
elastic_SS.fit(x_train_std, y_train)
y_pred = elastic_SS.predict(x_test_std)
elastic_SS_score = r2_score(y_test,y_pred)
print(' elastic model score after standard scaling :',elastic_SS_score)
```

```
elastic model score after standard scaling : 0.0006295647893912459
```

Lasso, ridge and Linear regression with standard scaling is giving accuracy score of 90.4%

#

Model building Using Decision tree withou scaling

In [172…
```
from sklearn.tree import DecisionTreeRegressor
reg = DecisionTreeRegressor()
reg.fit(X_train, y_train)
y_pred = reg.predict(X_test)
print('DT Score : ', r2_score(y_test,y_pred))
```

```
DT Score :  0.8592234429844348
```

Model building Using Decision tree after scaling

In [173…
```
reg = DecisionTreeRegressor()
reg.fit(x_train_std,y_train)
y_pred = reg.predict(x_test_std)
print('DT Score after scaling : ', r2_score(y_test,y_pred))
```

```
DT Score after scaling :  0.8552945088877152
```

Model building Using Random Forest without scaling

In [174…
```
from sklearn.ensemble import RandomForestRegressor
reg = RandomForestRegressor(n_estimators=100,max_depth=15)
reg.fit(X_train,y_train)
y_pred = reg.predict(X_test)
print('RF score : ',r2_score(y_test,y_pred))
```

```
RF score :  0.9217300394571311
```

Model building Using Random Forest after scaling

In [175…
```
reg = RandomForestRegressor(n_estimators=100,max_depth=15)
reg.fit(x_train_std,y_train)
y_pred = reg.predict(x_test_std)
print('RF score afer scaling : ',r2_score(y_test,y_pred))
```

```
RF score afer scaling :  0.9214134542512772
```

In [176…
```
# checking predicted value as per random forest model

reg.predict(X_test[5:8])
```

```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\base.py:413: UserWarning: X has feature names, but RandomForest
Regressor was fitted without feature names
  warnings.warn(
```

Out[176]:
```
array([18749374.69      , 19371255.54      , 15660019.69430952])
```

In [177…
```
# actual value of data

y_test[5:8]
```

Out[177]:
```
3028    12807413
363     15234515
3805    10382747
Name: TOTAL_SALES_PRICE, dtype: int64
```

In [179…
```
filename = 'Chennai_House_Price_model.pickle'
pickle.dump(reg, open(filename, 'wb'))
```

91.7% Score from Random Forest model - hence we will choose this

In [ ]: