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

```
## Importing Perth Housing Price dataset
df = pd.read_csv('PerthHousing.csv')
df.head()
```

	ADDRESS	SUBURB	PRICE	BEDROOMS	BATHROOMS	GARAGE	LAND_AREA	FLOOR_AREA	BUILD_YEAR	CBD_DIST	NEAREST_STN	NEAREST_STN_DIST	DATE_SOLD	POSTCODE	LATITUDE	LONGITUDE	NEAREST_SCH	NEAREST_SCH_DIST	NEA
0	1 Acorn Place	South Lake	565000	4	2	2.0	600	160	2003.0	18300	Cockburn Central Station	1800	09-2018\r	6164	-32.115900	115.842450	LAKELAND SENIOR HIGH SCHOOL	0.828339	
1	1 Addis Way	Wandi	365000	3	2	2.0	351	139	2013.0	26900	Kwinana Station	4900	02-2019\r	6167	-32.193470	115.859554	ATWELL COLLEGE	5.524324	
2	1 Ainsley Court	Camillo	287000	3	1	1.0	719	86	1979.0	22600	Challis Station	1900	06-2015\r	6111	-32.120578	115.993579	KELMSCOTT SENIOR HIGH SCHOOL	1.649178	

Next steps: Generate code with df | View recommended plots | New interactive sheet

Observing the dataset with respect to :-

- What are the columns available
- Data typee of those columns
- Total Number of Rows
- Not Null Rows

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33656 entries, 0 to 33655
Data columns (total 19 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   ADDRESS           33656 non-null  object 
 1   SUBURB            33656 non-null  object 
 2   PRICE             33656 non-null  int64  
 3   BEDROOMS          33656 non-null  int64  
 4   BATHROOMS         33656 non-null  int64  
 5   GARAGE            31178 non-null  float64
 6   LAND_AREA         33656 non-null  int64  
 7   FLOOR_AREA        33656 non-null  int64  
 8   BUILD_YEAR        30501 non-null  float64
 9   CBD_DIST          33656 non-null  int64  
 10  NEAREST_STN       33656 non-null  object 
 11  NEAREST_STN_DIST  33656 non-null  int64  
 12  DATE_SOLD         33656 non-null  object 
 13  POSTCODE          33656 non-null  int64  
 14  LATITUDE          33656 non-null  float64
 15  LONGITUDE         33656 non-null  float64
 16  NEAREST_SCH       33656 non-null  object 
 17  NEAREST_SCH_DIST  33656 non-null  float64
 18  NEAREST_SCH_RANK  22704 non-null  float64
dtypes: float64(6), int64(8), object(5)
memory usage: 4.9+ MB
```

Looking standard stats w.r.t to each column

```
## Lets see the columns names
df.columns
```

```
Index(['ADDRESS', 'SUBURB', 'PRICE', 'BEDROOMS', 'BATHROOMS', 'GARAGE',
       'LAND_AREA', 'FLOOR_AREA', 'BUILD_YEAR', 'CBD_DIST', 'NEAREST_STN',
       'NEAREST_STN_DIST', 'DATE_SOLD', 'POSTCODE', 'LATITUDE', 'LONGITUDE',
       'NEAREST_SCH', 'NEAREST_SCH_DIST', 'NEAREST_SCH_RANK'],
      dtype='object')
```

- address - Physical address of the property
- suburb - Specific locality in Perth
- price - Price of the property
- bedrooms - number of bedrooms
- bathrooms - number of bathrooms
- garage - garage place (number)
- land_area - area in square meter
- floor_area - floor area in square meter
- build_year - Year property was built
- CBD_dist - distance from center of perth
- nearest_stn - nearest public transport station
- nearest_stn_dst - nearest public transport station distance
- date_sold - Month & year in which property got sold
- nearest_sch - nearest school
- nearest_sch_dist - nearest school distance

```
df.describe()
```

	PRICE	BEDROOMS	BATHROOMS	GARAGE	LAND_AREA	FLOOR_AREA	BUILD_YEAR	CBD_DIST	NEAREST_STN_DIST	POSTCODE	LATITUDE	LONGITUDE	NEAREST_SCH_DIST	NEAREST_SCH_RANK
count	3.365600e+04	33656.000000	33656.000000	31178.000000	33656.000000	33656.000000	30501.000000	33656.000000	33656.000000	33656.000000	33656.000000	33656.000000	33656.000000	22704.000000
mean	6.370720e+05	3.659110	1.823063	2.199917	2740.644016	183.501545	1989.706436	19777.374465	4523.371494	6089.420074	-31.960664	115.879265	1.815268	72.672569
std	3.558256e+05	0.752038	0.587427	1.365225	16693.513215	72.102982	20.964330	11364.415413	4495.064024	62.167921	0.177780	0.118137	1.746000	40.639795
min	5.100000e+04	1.000000	1.000000	1.000000	61.000000	1.000000	1868.000000	681.000000	46.000000	6003.000000	-32.472979	115.582730	0.070912	1.000000
25%	4.100000e+05	3.000000	1.000000	2.000000	503.000000	130.000000	1978.000000	11200.000000	1800.000000	6050.000000	-32.068437	115.789763	0.880568	39.000000
50%	5.355000e+05	4.000000	2.000000	2.000000	682.000000	172.000000	1995.000000	17500.000000	3200.000000	6069.000000	-31.933231	115.854198	1.345520	68.000000
75%	7.600000e+05	4.000000	2.000000	2.000000	838.000000	222.250000	2005.000000	26600.000000	5300.000000	6150.000000	-31.843818	115.970722	2.097225	105.000000
max	2.440000e+06	10.000000	16.000000	99.000000	999999.000000	870.000000	2017.000000	59800.000000	35500.000000	6558.000000	-31.457450	116.343201	23.254372	139.000000

Find the percentage of null value in the dataset with respect to particular column

```
#check for duplicate rows
df.loc[df.duplicated()]
```

ADDRESS	SUBURB	PRICE	BEDROOMS	BATHROOMS	GARAGE	LAND_AREA	FLOOR_AREA	BUILD_YEAR	CBD_DIST	NEAREST_STN	NEAREST_STN_DIST	DATE_SOLD	POSTCODE	LATITUDE	LONGITUDE	NEAREST_SCH	NEAREST_SCH_DIST	NEAREST_S

```
df.isna().sum()/df.shape[0]*100
```

	0
ADDRESS	0.000000
SUBURB	0.000000
PRICE	0.000000
BEDROOMS	0.000000
BATHROOMS	0.000000
GARAGE	7.362729
LAND_AREA	0.000000
FLOOR_AREA	0.000000
BUILD_YEAR	9.374257
CBD_DIST	0.000000
NEAREST_STN	0.000000
NEAREST_STN_DIST	0.000000
DATE_SOLD	0.000000
POSTCODE	0.000000
LATITUDE	0.000000
LONGITUDE	0.000000
NEAREST_SCH	0.000000
NEAREST_SCH_DIST	0.000000
NEAREST_SCH_RANK	32.541003

Lets drop Nearest_SCH_RANK it doesn't effect much and there are a lot of null values

```
df.drop('NEAREST_SCH_RANK', axis=1, inplace=True)
```

Adding mean value in build year and in garage inplace of null value. Can use other option as well for example :- For Garage we can look at the maximum occurence of grouped by the land_area

```
df['BUILD_YEAR'] = df['BUILD_YEAR'].fillna(df['BUILD_YEAR'].mean())
df['GARAGE'] = df['GARAGE'].fillna(df['GARAGE'].mean())
```

Convert date_sold to date type

```
#convert to datatime with just year
df['DATE_SOLD'] = pd.to_datetime(df['DATE_SOLD']).dt.strftime('%Y')
df['DATE_SOLD']
```

```
<ipython-input-176-c37ca62bb39a>:2: UserWarning: Could not infer format, so each element will be parsed individually, falling back to `dateutil`. To ensure parsing is consistent and as-expected, please specify
  df['DATE_SOLD'] = pd.to_datetime(df['DATE_SOLD']).dt.strftime('%Y')
```

	DATE_SOLD
0	2018
1	2019
2	2015
3	2018
4	2016
...	...
33651	2016
33652	2017
33653	2017
33654	2016
33655	2016

33656 rows × 1 columns

```
## Converting Build_Year to int
df['BUILD_YEAR'] = df['BUILD_YEAR'].astype(int)
#df['BUILD_YEAR'] = pd.to_datetime(df['BUILD_YEAR'].astype(int)).dt.year
df['BUILD_YEAR']
```

	BUILD_YEAR
0	2003
1	2013
2	1979
3	1953
4	1998
...	...
33651	2013
33652	1989
33653	1989
33654	1974
33655	1989

33656 rows × 1 columns

Removing LATITUDE and LONGITUDE won't be using it

```
df.drop(['LATITUDE','LONGITUDE'], axis=1, inplace=True)
```

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

ADDRESS	count
123 Fairway	3
68 Margaret Street	2
20 Third Avenue	2
5 William Street	2
34 Halcyon Way	2
...	...
20 Metroliner Drive	1
20 Mentor Street	1
20 Melvich Green	1
20 Melrose Crescent	1
9E Margaret Street	1

33566 rows × 1 columns

Address for everyone is different so removing as it won't make sense in the prediction

```
df.drop('ADDRESS', axis=1, inplace=True)
```

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

SUBURB	count
Bertram	231
Iluka	212
Bennett Springs	211
Mindarie	209
Carramar	208
...	...
Munster	1
Kwinana Beach	1
Welshpool	1
Wangara	1
Naval Base	1

321 rows × 1 columns

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

NEAREST_STN	count
Midland Station	4141
Warwick Station	1696
Cockburn Central Station	1640
Armadale Station	1372
Butler Station	1178
...	...
East Perth Station	35
Mosman Park Station	33
McIver Station	23
City West Station	16
Esplanade Station	3

68 rows × 1 columns

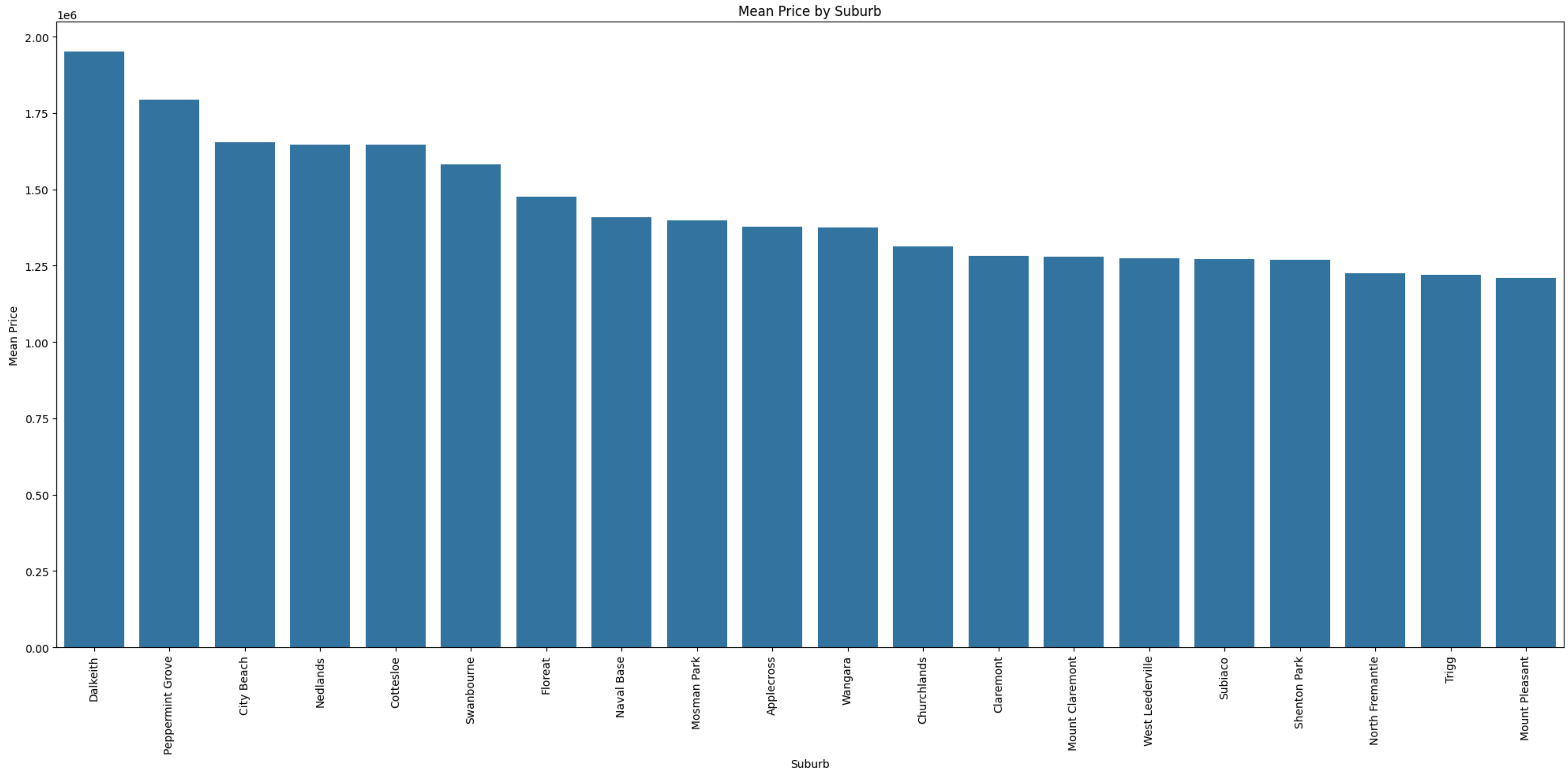
```
df['NEAREST_SCH'].value_counts()
```

NEAREST_SCH	count
SWAN VIEW SENIOR HIGH SCHOOL	895
KIARA COLLEGE	756
ATWELL COLLEGE	696
JOSEPH BANKS SECONDARY COLLEGE	687
SWAN VALLEY ANGLICAN COMMUNITY SCHOOL	567
...	...
ST GEORGE'S ANGLICAN GRAMMAR SCHOOL	15
METHODIST LADIES' COLLEGE	10
MERCEDES COLLEGE	8
FAIRBRIDGE COLLEGE	7
SOUTH METROPOLITAN YOUTH LINK COMMUNITY COLLEGE	3

160 rows × 1 columns

Lets visualise the Price with respect to particular suburb. Most of the time people have prefernces of the suburb where they want to buy the house. Lets see top 20 by price

```
mean_price_by_suburb = df.groupby('SUBURB')['PRICE'].mean()
top_20_mean_price_by_suburb = mean_price_by_suburb.nlargest(20)
plt.figure(figsize=(20,10))
sns.barplot(x=top_20_mean_price_by_suburb.index, y=top_20_mean_price_by_suburb.values)
plt.xlabel('Suburb')
plt.ylabel('Mean Price')
plt.title('Mean Price by Suburb')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



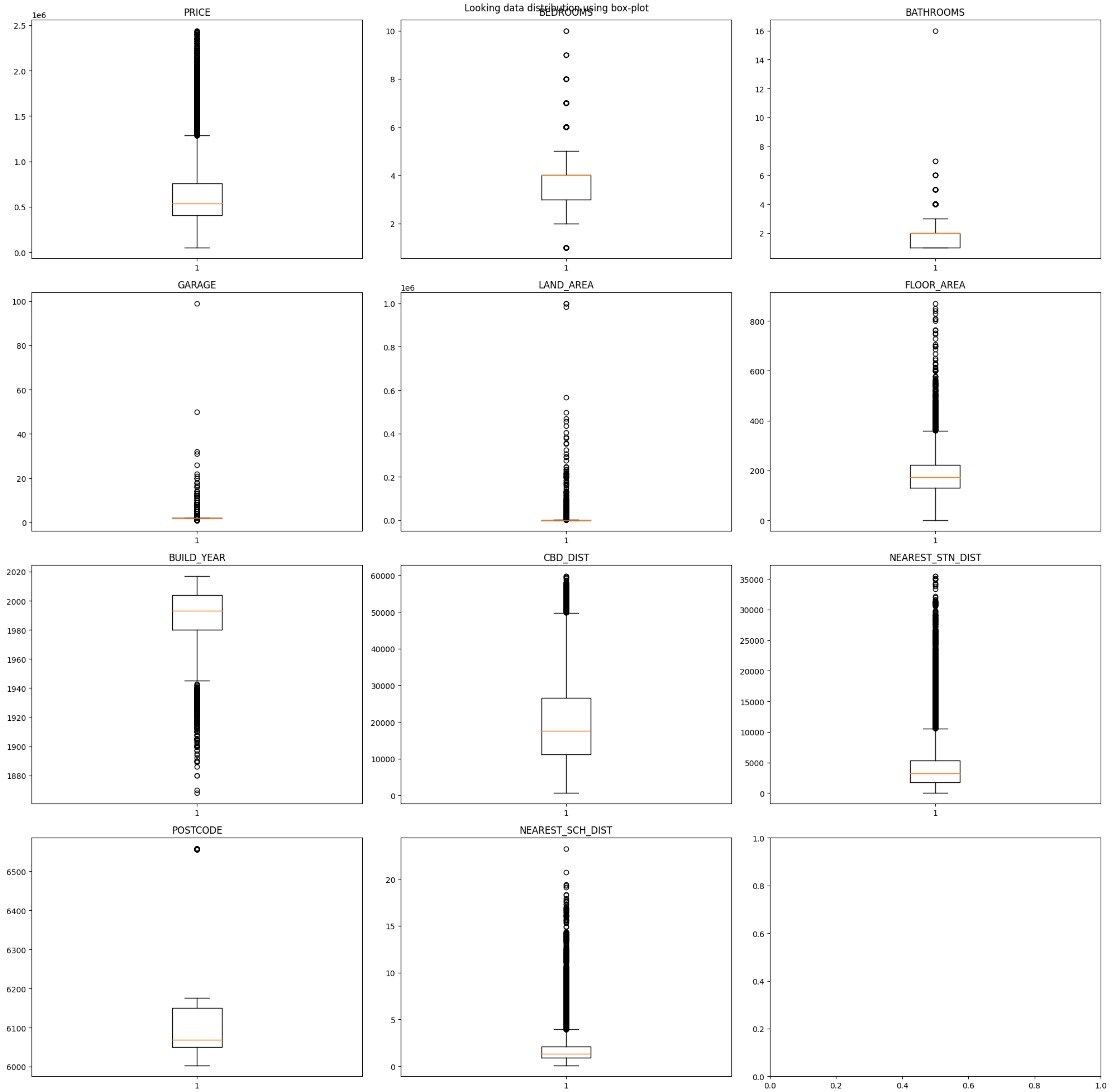
Lets see to box plot to see the distribution of the datal, can give the idea about the outliers too

```
columns_as_type_numeric = df.select_dtypes(include=[np.number]).columns
columns_as_type_numeric
```

```
Index(['PRICE', 'BEDROOMS', 'BATHROOMS', 'GARAGE', 'LAND_AREA', 'FLOOR_AREA',
       'BUILD_YEAR', 'CBD_DIST', 'NEAREST_STN_DIST', 'POSTCODE',
       'NEAREST_SCH_DIST'],
      dtype='object')
```

```
columns_as_type_numeric = df.select_dtypes(include=[np.number]).columns
fig, axs = plt.subplots(4,3, figsize=(20,20))
plt.subplots_adjust(hspace=0.5, wspace=0.5)
counts=0
len=columns_as_type_numeric.shape[0]
fig.suptitle("Looking data distribution using box-plot")
for x in range(4):
  for y in range(3):
    axs[x][y].boxplot(df[columns_as_type_numeric[counts]])
    axs[x][y].set_title(columns_as_type_numeric[counts])
    counts+=1
    if counts>=len:
      break

plt.tight_layout()
plt.show()
```



Performing Target Encoding for these three features

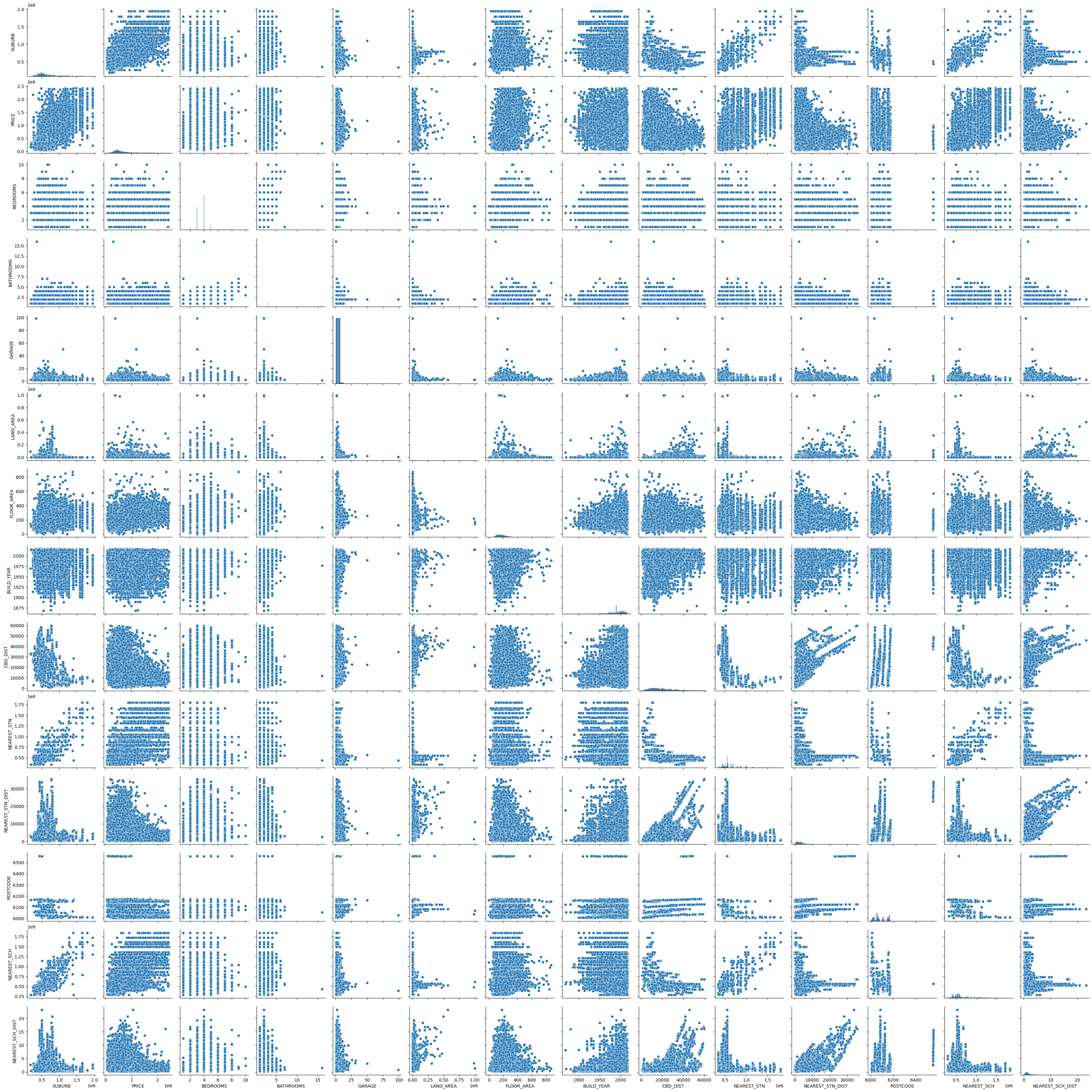
```
df['SUBURB']=df.groupby('SUBURB')['PRICE'].transform('mean')
```

Similarly, performing for NEAREST_STN and NEAREST_SCH

```
df['NEAREST_STN']=df.groupby('NEAREST_STN')['PRICE'].transform('mean')
df['NEAREST_SCH']=df.groupby('NEAREST_SCH')['PRICE'].transform('mean')
```

```
sns.pairplot(df)
```

<seaborn.axisgrid.PairGrid at 0x7c55d2e35f00>



Lets create a new column which gives the year difference between build_year and date_sold. Dropping build_year and date_sold to remove correlation

```
df['PROPERTY_AGE'] = df['DATE_SOLD'].astype(int) - df['BUILD_YEAR']
df.drop(['BUILD_YEAR','DATE_SOLD'], axis=1, inplace=True)
```

```
df['PROPERTY_AGE']
```

	PROPERTY_AGE
0	15
1	6
2	36
3	65
4	18
...	...
33651	3
33652	28
33653	28
33654	42
33655	27

33656 rows × 1 columns

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 33656 entries, 0 to 33655
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   SUBURB             33656 non-null  float64
 1   PRICE              33656 non-null  int64  
 2   BEDROOMS           33656 non-null  int64  
 3   BATHROOMS          33656 non-null  int64  
 4   GARAGE             33656 non-null  float64
 5   LAND_AREA          33656 non-null  int64  
 6   FLOOR_AREA         33656 non-null  int64  
 7   CBD_DIST           33656 non-null  int64  
 8   NEAREST_STN        33656 non-null  float64
 9   NEAREST_STN_DIST   33656 non-null  int64  
 10  POSTCODE           33656 non-null  int64  
 11  NEAREST_SCH        33656 non-null  float64
 12  NEAREST_SCH_DIST   33656 non-null  float64
 13  PROPERTY_AGE       33656 non-null  int64  
dtypes: float64(5), int64(9)
memory usage: 3.6 MB
```

Performing min-max scaling (Normalization) since range for the features differs a lot

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
df = pd.DataFrame(scaler.fit_transform(df), columns=df.columns)
df.head()
## Or you can just write your function for min-max scaling

# def min_max_normalize(x):
#   return ( (x-np.min(x))/(max(x)-min(x)) )

# df = df.apply(min_max_normalize)
```

	SUBURB	PRICE	BEDROOMS	BATHROOMS	GARAGE	LAND_AREA	FLOOR_AREA	CBD_DIST	NEAREST_STN	NEAREST_STN_DIST	POSTCODE	NEAREST_SCH	NEAREST_SCH_DIST	PROPERTY_AGE
0	0.151573	0.215153	0.333333	0.066667	0.010204	0.000539	0.182969	0.298026	0.155479	0.049473	0.290090	0.191065	0.032671	0.223529
1	0.216133	0.131436	0.222222	0.066667	0.010204	0.000290	0.158803	0.443495	0.066156	0.136910	0.295495	0.195674	0.235229	0.170588
2	0.073514	0.098786	0.222222	0.000000	0.000000	0.000658	0.097814	0.370761	0.059629	0.052293	0.194595	0.076420	0.068077	0.347059
3	0.111617	0.085391	0.111111	0.000000	0.010204	0.000590	0.066743	0.291260	0.142283	0.100243	0.095495	0.092800	0.064722	0.517647
4	0.103837	0.114692	0.333333	0.000000	0.010204	0.000405	0.149597	0.177929	0.076344	0.055114	0.091892	0.092751	0.062286	0.241176

Dividing dataset into train and test

```
from sklearn.model_selection import train_test_split
X = df.drop('PRICE', axis=1)
y = df['PRICE']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3, random_state=15)
```

```
print("X_train shape :- ",X_train.shape)
print("X_test shape :- ",X_test.shape)
print("y_train shape :- ",y_train.shape)
print("y_test shape :- ",y_test.shape)
```

```
X_train shape :-  (23559, 13)
X_test shape :-  (10097, 13)
y_train shape :-  (23559,)
y_test shape :-  (10097,)
```

Lets just predict using sklearn library

```
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
## Print the coeff_values
print("Coefficient/Weights are :- ",model.coef_)
```

```
Coefficient/Weights are :-  [ 0.43466133  0.07273627  0.35266344  0.47772461  0.47635351  0.53636285
 -0.02125304  0.06033943 -0.04534635  0.0069452   0.11781586  0.01482866
  0.1428876 ]
```

```
## Printing intercept value
print("Intercept is :- ",model.intercept_)
```

```
Intercept is :-  -0.09975612062067021
```

```
## Predicting y_pred for test dataset
y_pred = model.predict(X_test)
print("Predicted Values are :-", y_pred)
print(y_pred.shape)
```

```
Predicted Values are :- [0.06089156 0.09854722 0.2125603  ... 0.224197   0.22278046 0.18558698]
(10097,)
```

```
print("R2 score is :-", model.score(X_train, y_train))
```

```
R2 score is :- 0.7502977184299421
```

```
## R2 score value
print("R2 score is :- ", model.score(X_test,y_test))
```

```
R2 score is :-  0.7624677834025194
```

Lets re-write Linear Regression from Scratch

```
class ScratchLinearRegression:
   def __init__(self, learning_rate=0.1, n_iters=1000):
    self.learning_rate = learning_rate
    self.n_iters = n_iters
```

```
def predict(self, X):
    return np.dot(X, self.W) + self.b
ScratchLinearRegression.predict = predict
```

```
def r2_score(self, X, y):
    y_pred = predict(self, X)
    ss_res = np.sum((y-y_pred)**2)
    ss_tot = np.sum((y-y.mean())**2)
    score = (1 - ss_res/ss_tot)
    return score
ScratchLinearRegression.r2_score = r2_score
```

```
def update_weights(self):
    y_pred = self.predict(self.X)
    dW = - (2*(self.X.T).dot(self.y - y_pred))/self.m
    db = - 2*np.sum(self.y - y_pred)/self.m
    self.W = self.W - self.learning_rate*dW
    self.b = self.b - self.learning_rate*db
    return self
ScratchLinearRegression.update_weights = update_weights
```

```
def fit(self, X, y):
    self.m, self.n = X.shape
    self.W = np.zeros(self.n)
    self.b = 0
    self.X = X
    self.y = y
    self.error_list = []
    for i in range(self.n_iters):
      self.update_weights()
      y_pred = X.dot(self.W)+self.b
      error = np.square(np.subtract(y,y_pred)).mean()
      self.error_list.append(error)
    return self
ScratchLinearRegression.fit = fit
```

```
lr = ScratchLinearRegression(n_iters=1000)
```

```
X_train.shape
```

```
(23559, 13)
```

```
y_train.shape
```

```
(23559,)
```

```
lr.fit(X_train,y_train)
```

```
<__main__.ScratchLinearRegression at 0x7c55fdacb430>
```

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

```
lr.W #Coefficients/Weights
```

	0
SUBURB	0.370898
BEDROOMS	0.178295
BATHROOMS	0.136275
GARAGE	0.027867
LAND_AREA	0.029731
FLOOR_AREA	0.402187
CBD_DIST	-0.020853
NEAREST_STN	0.078796
NEAREST_STN_DIST	-0.018083
POSTCODE	-0.007065
NEAREST_SCH	0.200327
NEAREST_SCH_DIST	0.038270
PROPERTY_AGE	0.070450

dtype: float64

```
lr.b #interpect
```

```
-0.07133638188940825
```

```
lr.r2_score(X_train, y_train)
```

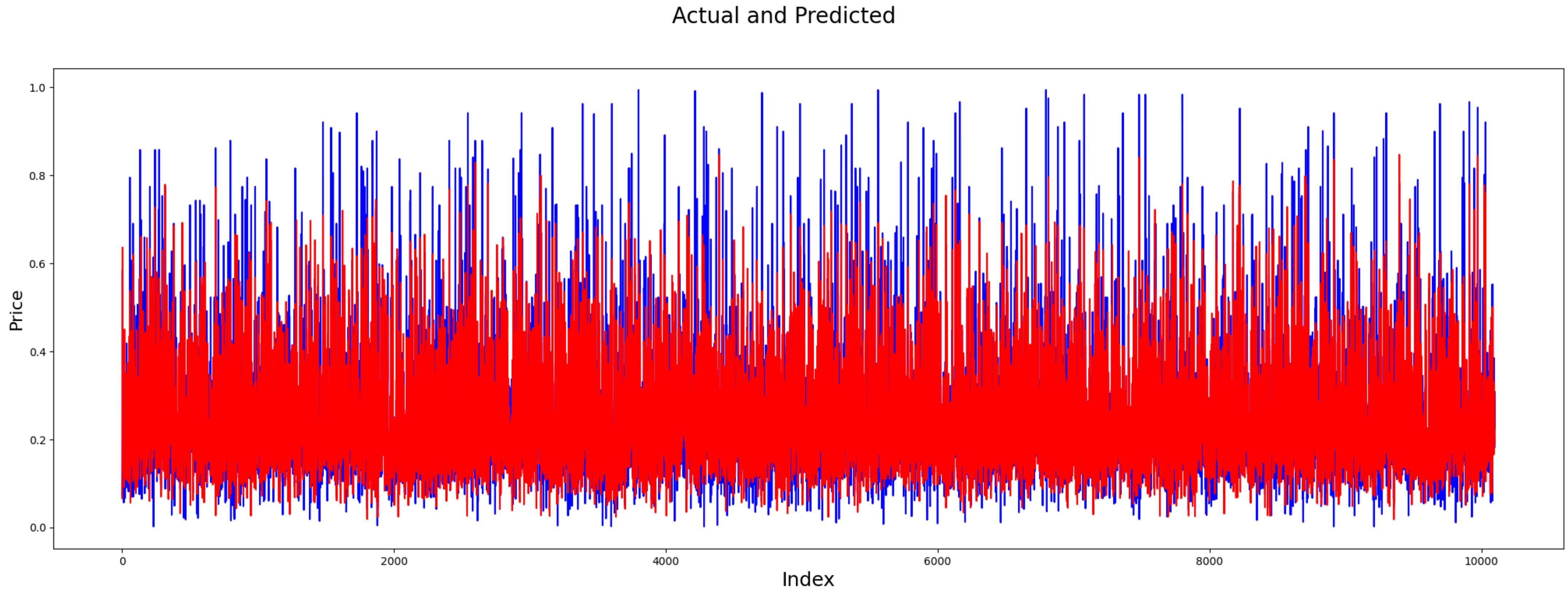
```
0.7352425943325134
```

```
lr.r2_score(X_test, y_test)
```

```
0.7474710479875291
```

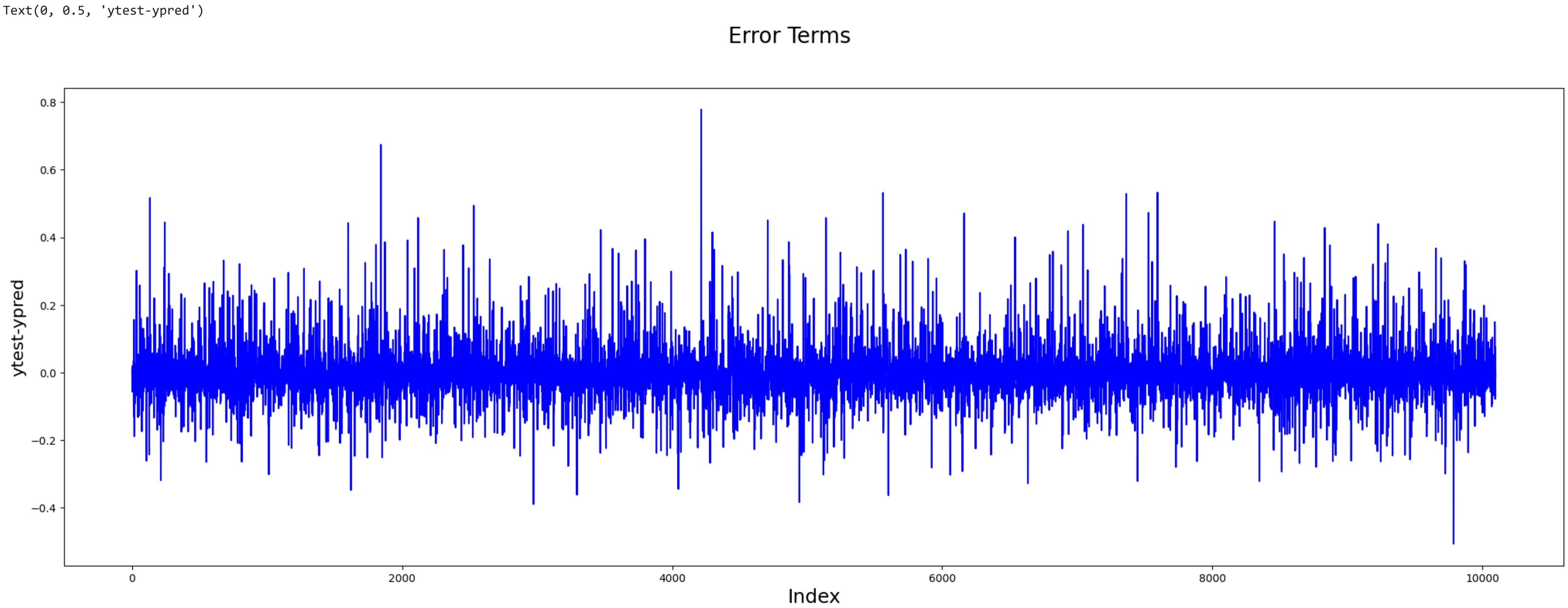
```
# Actual and Predicted
import matplotlib.pyplot as plt
c = [i for i in range(1,10098,1)] # generating index
fig = plt.figure(figsize=(25,8))
plt.plot(c,y_test, color="blue", linewidth=1.5, linestyle="-") #Plotting Actual
plt.plot(c,y_pred, color="red",  linewidth=1.5, linestyle="-") #Plotting predicted
fig.suptitle('Actual and Predicted', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                               # X-label
plt.ylabel('Price', fontsize=16)                       # Y-label
```

```
Text(0, 0.5, 'Price')
```




```
# Error terms
c = [i for i in range(1,10098,1)]
fig = plt.figure(figsize=(25,8))
plt.plot(c,y_test-y_pred, color="blue", linewidth=1.5, linestyle="-")
fig.suptitle('Error Terms', fontsize=20)              # Plot heading
plt.xlabel('Index', fontsize=18)                      # X-label
plt.ylabel('ytest-ypred', fontsize=16)                # Y-label
```

```
Text(0, 0.5, 'ytest-ypred')
```




```
# Plotting y_test and y_pred to understand the spread.
fig = plt.figure(figsize=(20,7))
plt.scatter(y_test,y_pred)
fig.suptitle('y_test vs y_pred', fontsize=20)              # Plot heading
plt.xlabel('y_test', fontsize=18)                          # X-label
plt.ylabel('y_pred', fontsize=16)                          # Y-label
```