In [351]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

In [352]:

```
df=pd.read_excel("DS - Assignment Part 1 data set.xlsx")
```

In [353]:

```
df
```

Out[353]:

	Transaction date	House Age	Distance from nearest Metro station (km)	Number of convenience stores	latitude	longitude	Number of bedrooms	House size (sqft)	Ho p of
0	2012.916667	32.0	84.87882	10	24.98298	121.54024	1	575	
1	2012.916667	19.5	306.59470	9	24.98034	121.53951	2	1240	
2	2013.583333	13.3	561.98450	5	24.98746	121.54391	3	1060	
3	2013.500000	13.3	561.98450	5	24.98746	121.54391	2	875	
4	2012.833333	5.0	390.56840	5	24.97937	121.54245	1	491	
...	...	...	...	...	...	...	...	...	
409	2013.000000	13.7	4082.01500	0	24.94155	121.50381	3	803	
410	2012.666667	5.6	90.45606	9	24.97433	121.54310	2	1278	
411	2013.250000	18.8	390.96960	7	24.97923	121.53986	1	503	
412	2013.000000	8.1	104.81010	5	24.96674	121.54067	1	597	
413	2013.500000	6.5	90.45606	9	24.97433	121.54310	2	1097	

414 rows × 9 columns

In [354]:

```
print('We have {} rows.'.format(df.shape[0]))
print('We have {} columns.'.format(df.shape[1]))
```

```
We have 414 rows.
We have 9 columns.
```

In [355]:

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

Out[355]:

```
Transaction date                              0
House Age                                     0
Distance from nearest Metro station (km)      0
Number of convenience stores                  0
latitude                                      0
longitude                                     0
Number of bedrooms                            0
House size (sqft)                             0
House price of unit area                      0
dtype: int64
```

In [356]:

```
df.dtypes
```

Out[356]:

```
Transaction date                              float64
House Age                                     float64
Distance from nearest Metro station (km)      float64
Number of convenience stores                    int64
latitude                                      float64
longitude                                     float64
Number of bedrooms                              int64
House size (sqft)                               int64
House price of unit area                      float64
dtype: object
```

In [357]:

```
#df = df.iloc[:,1:]
df_norm = (df - df.mean()) / (df.max()-df.min())
df_norm.head()
```

Out[357]:

	Transaction date	House Age	Distance from nearest Metro station (km)	Number of convenience stores	latitude	longitude	Number of bedrooms	House size (sqft)	(
0	-0.253404	0.326197	-0.154534	0.59058	0.169049	0.074174	-0.493961	-0.324659	-(
1	-0.253404	0.040809	-0.120237	0.49058	0.137057	0.066303	0.006039	0.280987	(
2	0.473869	-0.100743	-0.080732	0.09058	0.223339	0.113747	0.506039	0.117053	(
3	0.382960	-0.100743	-0.080732	0.09058	0.223339	0.113747	0.006039	-0.051435	(
4	-0.344313	-0.290241	-0.107248	0.09058	0.125302	0.098004	-0.493961	-0.401162	(

In [358]:

```
df_norm.corr()
```

Out[358]:

	Transaction date	House Age	Distance from nearest Metro station (km)	Number of convenience stores	latitude	longitude	Number of bedrooms
Transaction date	1.000000	0.017542	0.060880	0.009544	0.035016	-0.041065	0.061985
House Age	0.017542	1.000000	0.025622	0.049593	0.054420	-0.048520	-0.008756
Distance from nearest Metro station (km)	0.060880	0.025622	1.000000	-0.602519	-0.591067	-0.806317	-0.046856
Number of convenience stores	0.009544	0.049593	-0.602519	1.000000	0.444143	0.449099	0.043638
latitude	0.035016	0.054420	-0.591067	0.444143	1.000000	0.412924	0.043921
longitude	-0.041065	-0.048520	-0.806317	0.449099	0.412924	1.000000	0.041680
Number of bedrooms	0.061985	-0.008756	-0.046856	0.043638	0.043921	0.041680	1.000000
House size (sqft)	0.068405	-0.060361	0.001795	0.033286	0.031696	0.009322	0.752276
House price of unit area	0.087529	-0.210567	-0.673613	0.571005	0.546307	0.523287	0.050265

Target Feature

In [359]:

```
df["House price of unit area"].describe()[1:]
```

Out[359]:

```
mean      37.980193
std       13.606488
min        7.600000
25%       27.700000
50%       38.450000
75%       46.600000
max      117.500000
Name: House price of unit area, dtype: float64
```

In [360]:

```
df["House price of unit area"].median()
```

Out[360]:

38.45

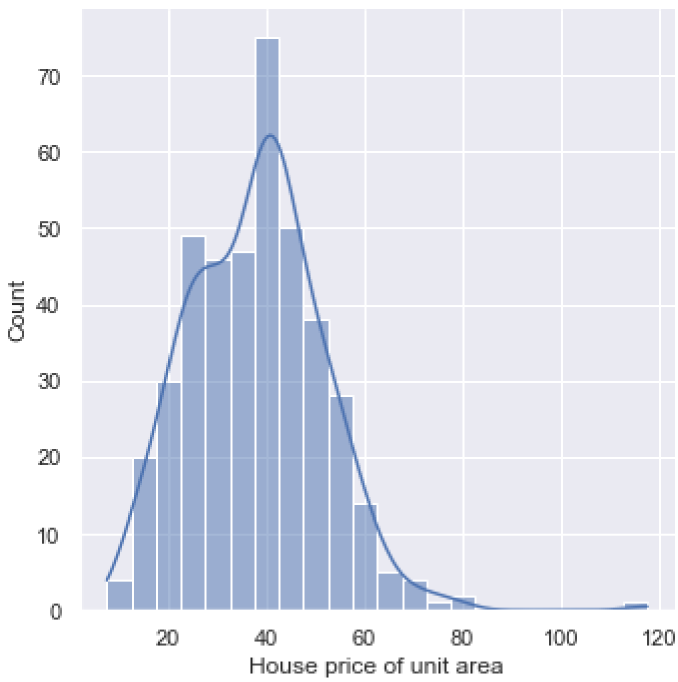
In [361]:

```
plt.figure(figsize=(7,5))
sns.displot(df["House price of unit area"], kde=True)
```

Out[361]:

<seaborn.axisgrid.FacetGrid at 0x23ef1917760>

<Figure size 504x360 with 0 Axes>

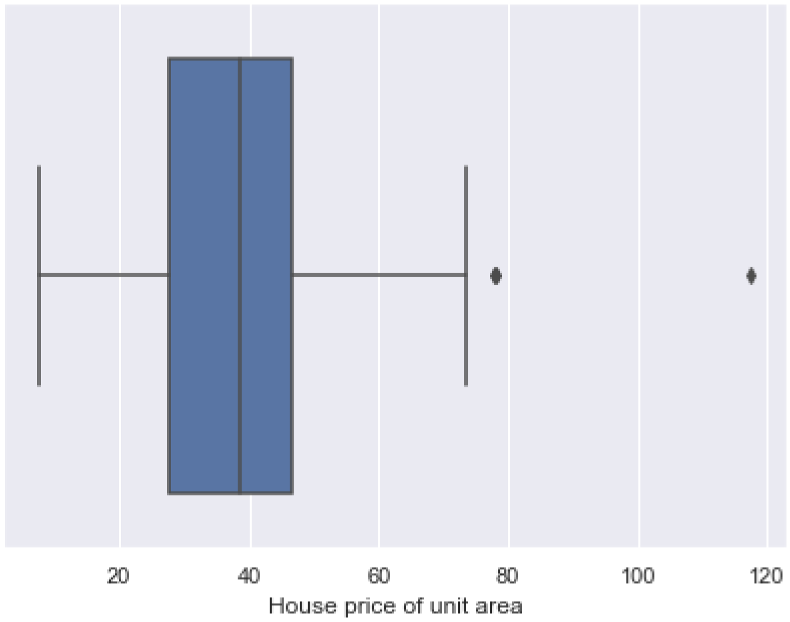


In [362]:

```
plt.figure(figsize=(7,5))
sns.boxplot(df["House price of unit area"])
```

Out[362]:

```
<AxesSubplot:xlabel='House price of unit area'>
```



-We can observe the outliers.

-Detecting the outlier using IQR and removing them.

In [363]:

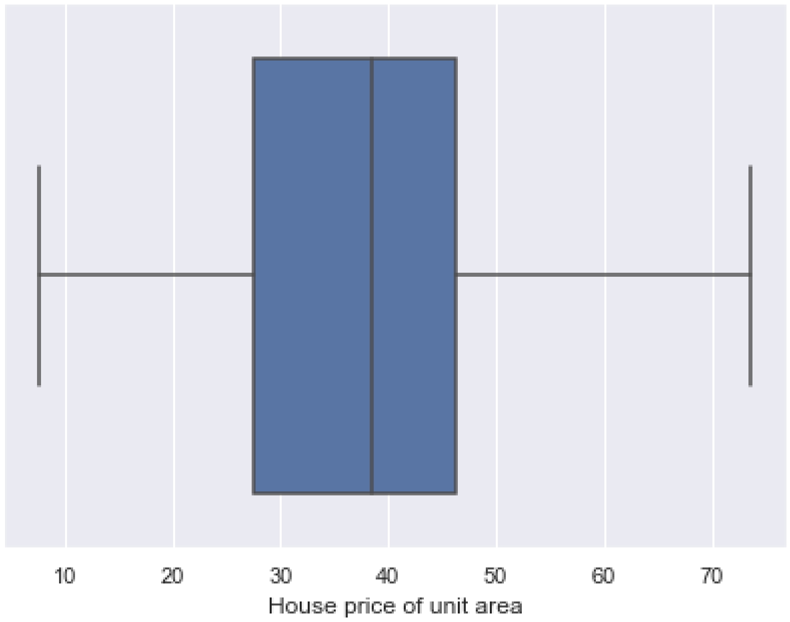
```
q1=np.percentile(df["House price of unit area"],25,interpolation='midpoint')
q3=np.percentile(df["House price of unit area"],75,interpolation='midpoint')
iqr= q3-q1
maximum=q3+1.5*iqr
df=df[df["House price of unit area"]<=maximum]
```

In [364]:

```
plt.figure(figsize=(7,5))
sns.boxplot(df["House price of unit area"])
```

Out[364]:

```
<AxesSubplot:xlabel='House price of unit area'>
```



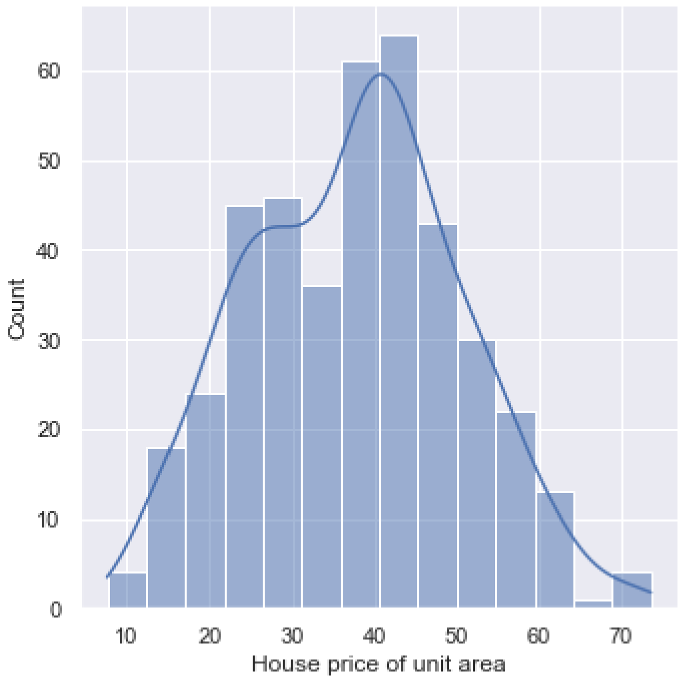
In [365]:

```
plt.figure(figsize=(7,5))
sns.displot(df["House price of unit area"], kde=True)
```

Out[365]:

```
<seaborn.axisgrid.FacetGrid at 0x23ef1a23130>
```

```
<Figure size 504x360 with 0 Axes>
```



-Outliers are removed.

-Lets check Skew and Kurtosis for the feature.

In [366]:

```
print("Skewness : {}".format(df["House price of unit area"].skew()))
print("Kurtosis : {}".format(df["House price of unit area"].kurtosis()))
```

```
Skewness : 0.07840924835348974
Kurtosis : -0.4699660583196885
```

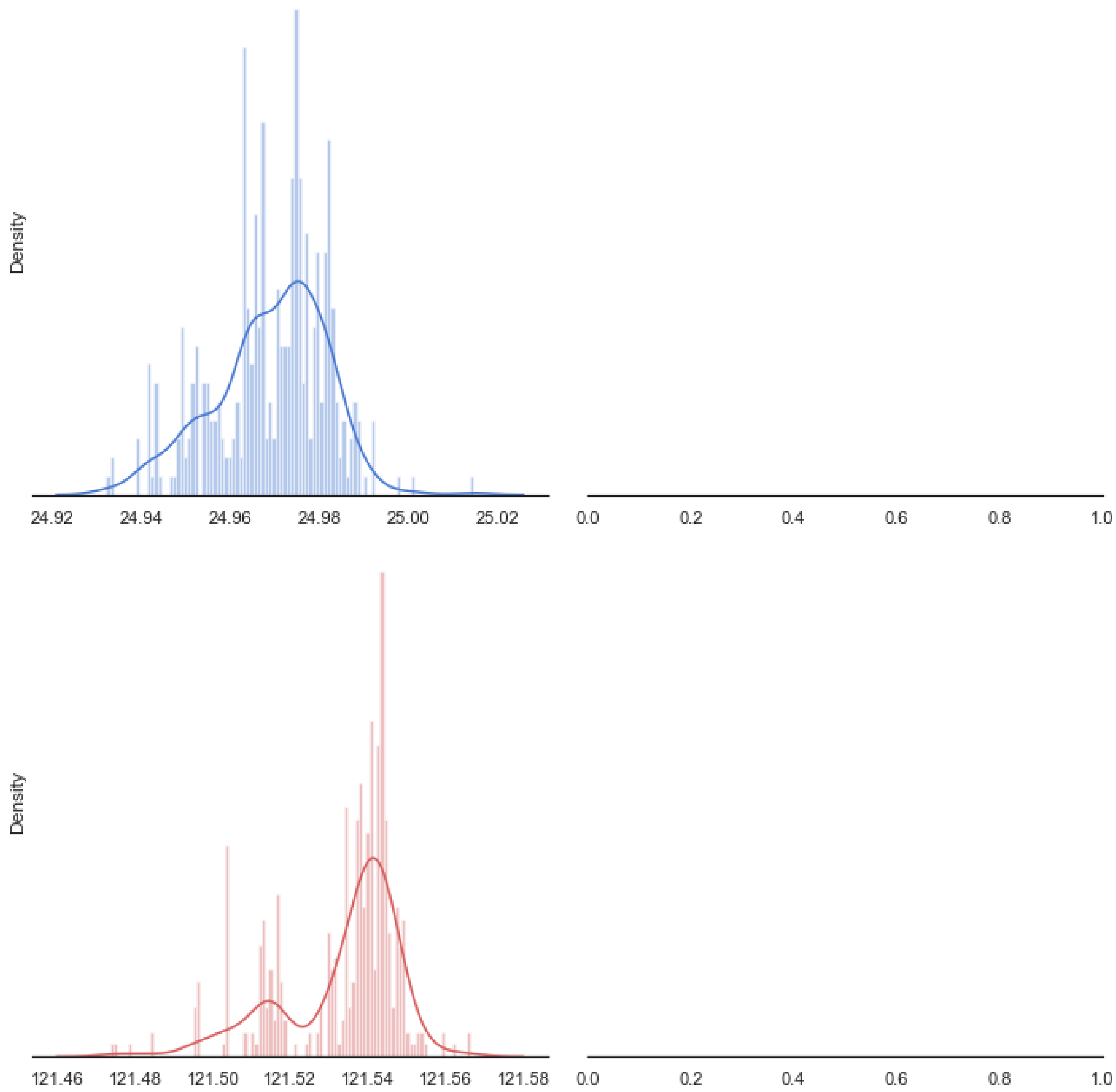
- -0.5 to 0.5 skew is considered for symmetric distribution.

- Negetive Kurtosis is refers as 'playkurtis' distribution will have thinner tails than a normal distribution means that there are few extreme events.

Other Features

In [367]:

```
sns.set(style="white", palette="muted", color_codes=True)
f, axes = plt.subplots(2,2,figsize=(10, 10), sharex=False, sharey = False)
sns.despine(left=True)
sns.distplot(df['latitude'].values, label = 'pickup_latitude',color="b",bins = 100, ax=axes
sns.distplot(df['longitude'].values, label = 'pickup_longitude',color="r",bins =100, ax=axe
plt.setp(axes, yticks=[])
plt.tight_layout()
plt.show()
```



In [368]:

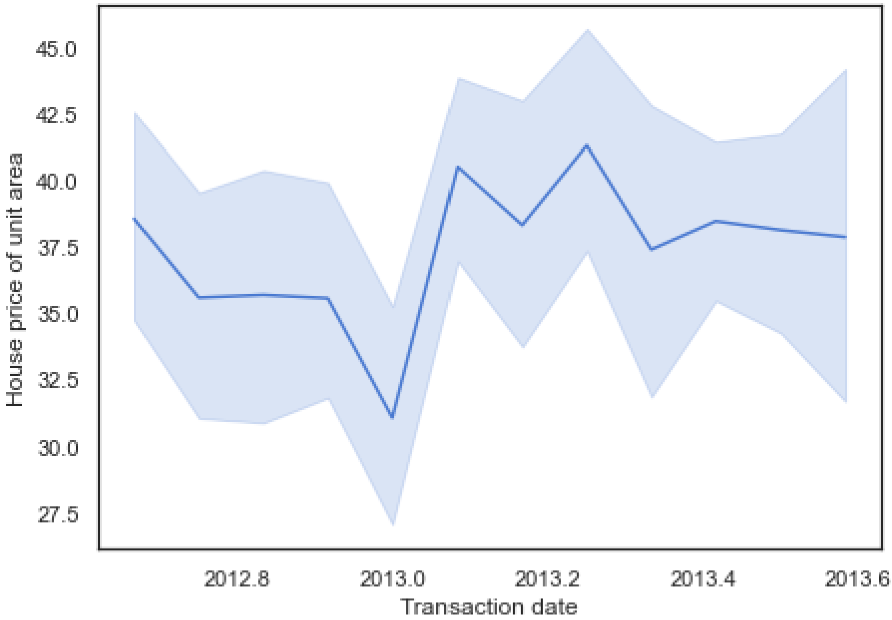
```
feature=["Transaction date","House Age","Distance from nearest Metro station (km)","Number
```

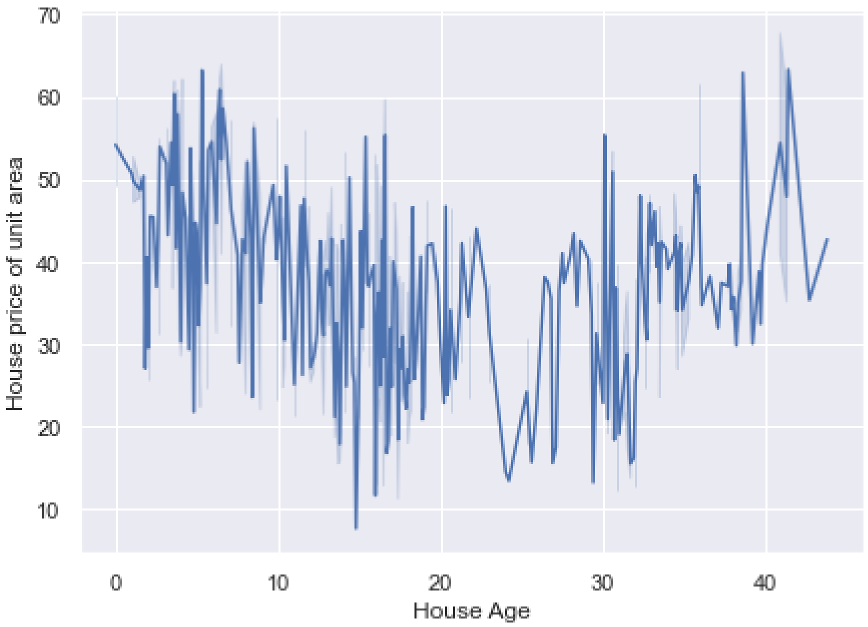
In [369]:

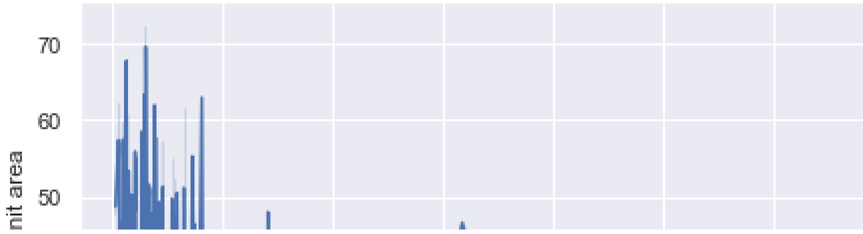
```
def visual(df, group):
    size=len(group)

    for j, i in enumerate(group):
        fig, ax = plt.subplots(figsize=(7, 5))
        sns.set()
        sns.lineplot(data=df,x=df[i], y='House price of unit area')
        plt.show()
```

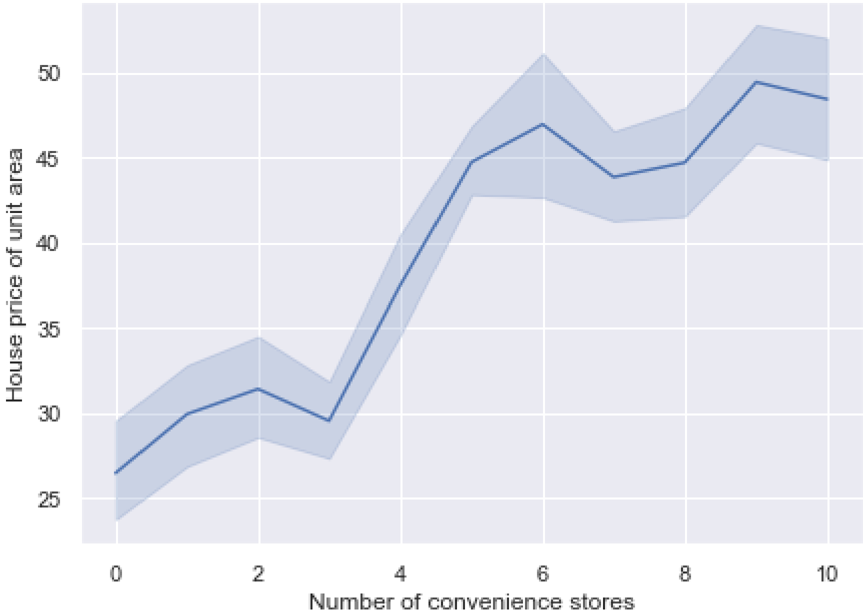
In [370]:

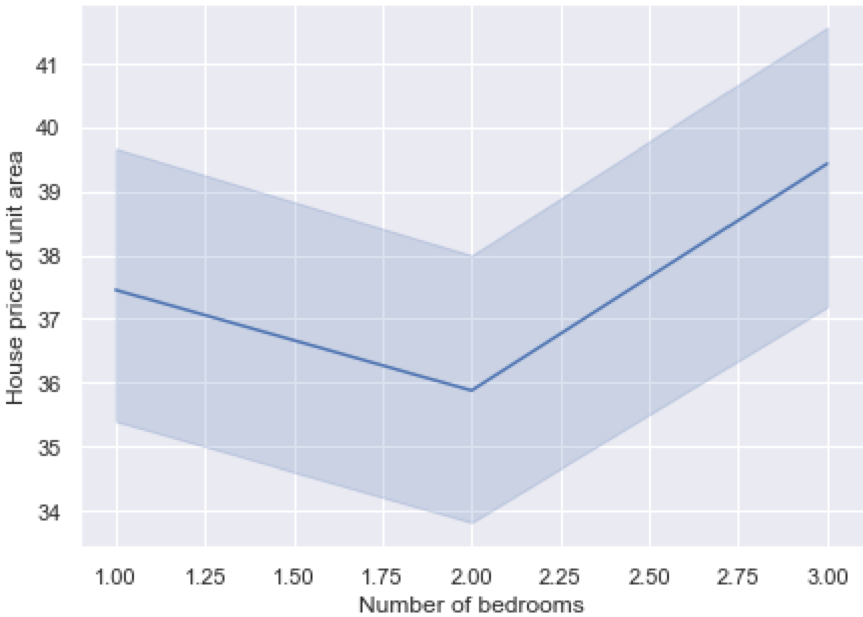
```
visual(df, feature)
```

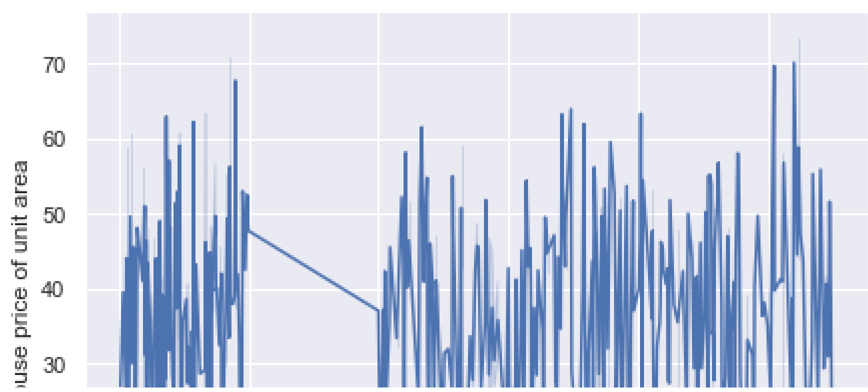





70
60
50
nit area


House price of unit area
50
45
40
35
30
25
0
2
4
6
8
10
Number of convenience stores


House price of unit area
41
40
39
38
37
36
35
34
1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Number of bedrooms

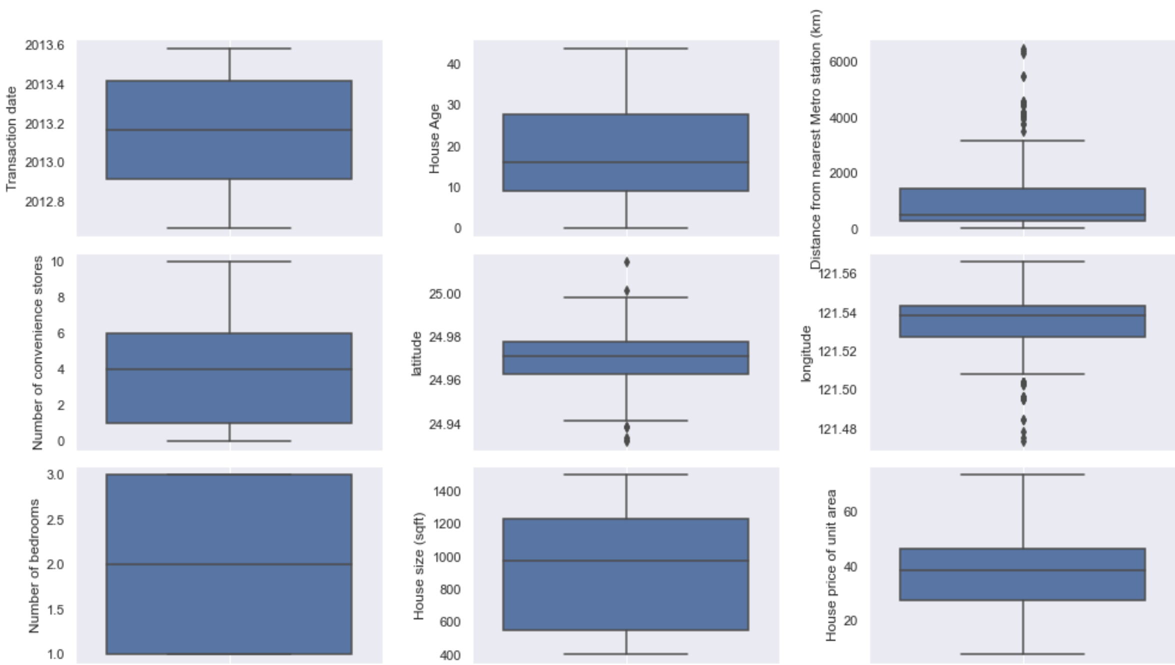


- House price are affected by number of convenience stores. Price are higher if the number of convenience store is more.
- We can observe that closer the house to the nearest MRT station, higher the price.
- Hosuse age doesn't have that much affect.

Checking Outliers

In [371]:

```
fig = plt.figure(figsize=(14,15))
for index,col in enumerate(df):
    plt.subplot(6,3,index+1)
    sns.boxplot(y=col, data=df.dropna())
    plt.grid()
fig.tight_layout(pad=1.0)
```



- Distance to the nearest MRT station have outliers

In [372]:

```
df = df[df['Distance from nearest Metro station (km)']<3000] #removing outliers
```

In [373]:

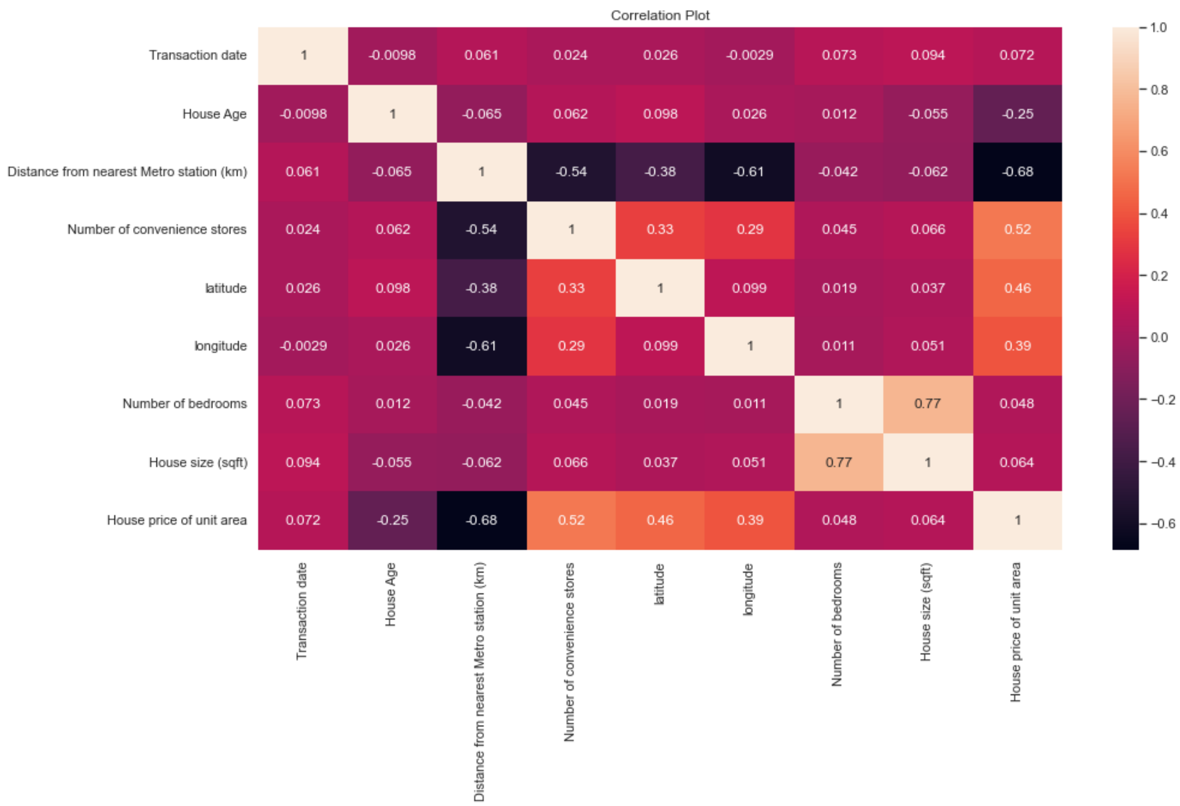
```
df['date'] = pd.to_datetime(df['Transaction date'], format='%Y') #Extracting Date from tran
```

In [374]:

```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(), annot=True)
plt.title('Correlation Plot')
```

Out[374]:

```
Text(0.5, 1.0, 'Correlation Plot')
```



In [375]:

```
#converting back the normalized price to real value of price
y_mean = df['House price of unit area'].mean()
y_std = df['House price of unit area'].std()
def convert_label_value(pred):
    return int(pred * y_std + y_mean)
#testing the function
print(convert_label_value(0.12))
```

```
40
```

In [376]:

```
#input features
x = df_norm.iloc[:, :6]
x.head()
```

Out[376]:

	Transaction date	House Age	Distance from nearest Metro station (km)	Number of convenience stores	latitude	longitude
0	-0.253404	0.326197	-0.154534	0.59058	0.169049	0.074174
1	-0.253404	0.040809	-0.120237	0.49058	0.137057	0.066303
2	0.473869	-0.100743	-0.080732	0.09058	0.223339	0.113747
3	0.382960	-0.100743	-0.080732	0.09058	0.223339	0.113747
4	-0.344313	-0.290241	-0.107248	0.09058	0.125302	0.098004

In [377]:

```
y = df_norm.iloc[:, -1]
y.head()
```

Out[377]:

```
0   -0.000730
1    0.038397
2    0.084803
3    0.153046
4    0.046586
Name: House price of unit area, dtype: float64
```

In [378]:

```
X_arr = x.values
y_arr = y.values
```

In [379]:

```
X_train, X_test, y_train, y_test = train_test_split(X_arr, y_arr, test_size = 0.2, shuffle
print('X_train shape: ', X_train.shape)
print('y_train shape: ', y_train.shape)
print('X_test shape: ', X_test.shape)
print('y_test shape: ', y_test.shape)
```

```
X_train shape:  (331, 6)
y_train shape:  (331,)
X_test shape:  (83, 6)
y_test shape:  (83,)
```

Neural Network Model

In [380]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.optimizers import Adam
import keras.losses
from tensorflow.keras.callbacks import EarlyStopping, LambdaCallback
```

In [381]:

```
def get_model():
    model = Sequential([
        Dense(5, input_shape = (6,), activation = 'relu'),
        Dense(12, activation = 'sigmoid'),
        Dense(6, activation = 'relu'),
        Dense(1)
    ])
    model.compile(
        loss='mse',
        optimizer='adadelta'
    )
    return model
model = get_model()
model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_28 (Dense)            (None, 5)                 35

 dense_29 (Dense)            (None, 12)                72

 dense_30 (Dense)            (None, 6)                 78

 dense_31 (Dense)            (None, 1)                 7

=================================================================
Total params: 192
Trainable params: 192
Non-trainable params: 0
_________________________________________________________________
```

In [382]:

```
early_stopping = EarlyStopping(monitor='accuracy', patience = 5)
history = model.fit(
    X_train, y_train,
    validation_data = (X_test, y_test),
    epochs = 100,
    callbacks = [early_stopping]
)
```

```
sorflow:Early stopping conditioned on metric `accuracy` which is not avail
able. Available metrics are: loss,val_loss
11/11 [==============================] - 0s 10ms/step - loss: 1.2363 - val
_loss: 1.2334
Epoch 4/100
 1/11 [=>............................] - ETA: 0s - loss: 1.2172WARNING:ten
sorflow:Early stopping conditioned on metric `accuracy` which is not avail
able. Available metrics are: loss,val_loss
11/11 [==============================] - 0s 8ms/step - loss: 1.2356 - val_
loss: 1.2327
Epoch 5/100
 1/11 [=>............................] - ETA: 0s - loss: 1.2175WARNING:ten
sorflow:Early stopping conditioned on metric `accuracy` which is not avail
able. Available metrics are: loss,val_loss
11/11 [==============================] - 0s 10ms/step - loss: 1.2349 - val
_loss: 1.2320
Epoch 6/100
 1/11 [=>............................] - ETA: 0s - loss: 1.2142WARNING:ten
sorflow:Early stopping conditioned on metric `accuracy` which is not avail
able. Available metrics are: loss.val loss
```

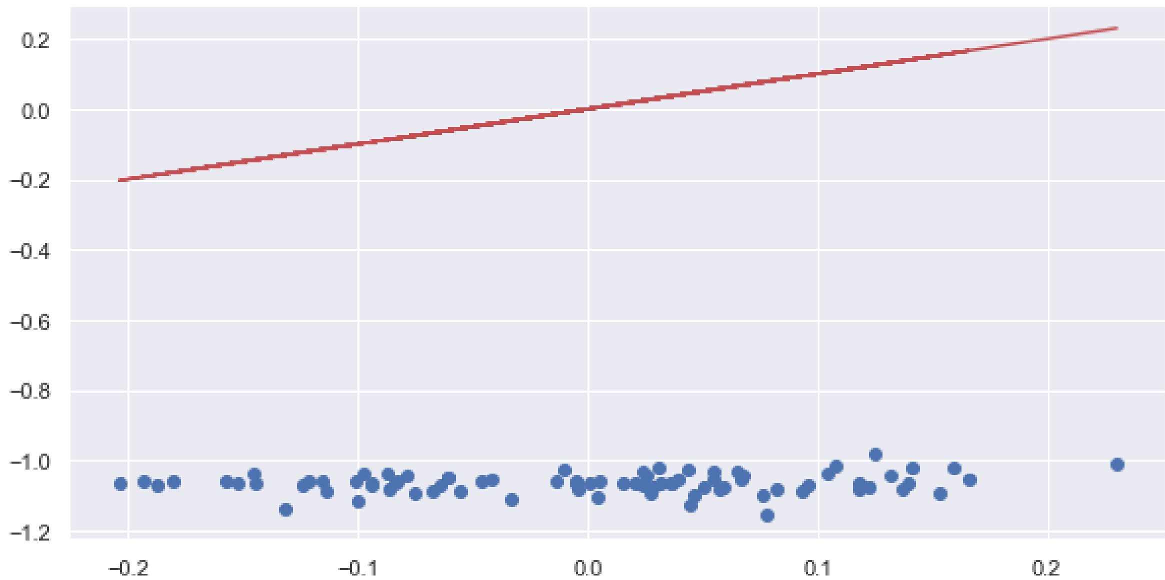
In [383]:

```
y_pred = model.predict(X_test)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
# Visualizing Our predictions
fig = plt.figure(figsize=(10,5))
plt.scatter(y_test,y_pred)
plt.plot(y_test,y_test,'r')
```

```
3/3 [==============================] - 0s 6ms/step
MAE: 1.064330265463306
MSE: 1.1421269297322645
RMSE: 1.0687033871623428
VarScore: 0.020213871380122095
```

Out[383]:

```
[<matplotlib.lines.Line2D at 0x23efaee9c10>]
```



Multiple Liner Regression

In [384]:

```
X=df.drop(['House price of unit area','Number of bedrooms',"House size (sqft)",'date'],axis
y=df['House price of unit area']
```

In [385]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=101)
```

In [386]:

```
#standardization scaler - fit&transform on train, fit only on test
from sklearn.preprocessing import StandardScaler
s_scaler = StandardScaler()
X_train = s_scaler.fit_transform(X_train.astype(np.float))
X_test = s_scaler.transform(X_test.astype(np.float))
```

In [387]:

```
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_train, y_train)
#evaluate the model (intercept and slope)
print(regressor.intercept_)
print(regressor.coef_)
#predicting the test set result
y_pred = regressor.predict(X_test)
#put results as a DataFrame
coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
coeff_df
```

```
39.475675675676136
[ 1.10673148 -3.50014834 -5.961365    2.32551228  2.71433804  0.12059698]
```

Out[387]:

	Coefficient
Transaction date	1.106731
House Age	-3.500148
Distance from nearest Metro station (km)	-5.961365
Number of convenience stores	2.325512
latitude	2.714338
longitude	0.120597

In [388]:

```
# visualizing residuals
fig = plt.figure(figsize=(10,5))
residuals = (y_test- y_pred)
sns.distplot(residuals)
```

Out[388]:

```
<AxesSubplot:xlabel='House price of unit area', ylabel='Density'>
```

In [389]:

```
#compare actual output values with predicted values
y_pred = regressor.predict(X_test)
results = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
df1 = results.head(10)
print(df1)
print("")
# evaluate the performance of the algorithm (MAE - MSE - RMSE)
from sklearn import metrics
print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
print('MSE:', metrics.mean_squared_error(y_test, y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
print('VarScore:',metrics.explained_variance_score(y_test,y_pred))
```

```
     Actual  Predicted
239    29.7  34.172000
97     34.6  37.740485
203    46.1  45.900690
284    34.4  43.722221
323    42.5  44.993842
193    49.3  49.583272
267    41.1  36.890390
211    43.5  46.836964
195    34.6  39.182445
152    28.9  28.331134

MAE: 4.380618762798454
MSE: 29.06728253626447
RMSE: 5.391408214582205
VarScore: 0.7325918361246666
```

Regularized Ridge and Lasso Model

In [390]:

```
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=.3)
ridge.fit(X_train,y_train)
print ("Ridge model:", (ridge.coef_))
ridge_pred=ridge.predict(X_test)
```

```
Ridge model: [ 1.10538315 -3.49632458 -5.9511888   2.32683654  2.71446437
0.12595251]
```

In [391]:

```
print('Train score: ', ridge.score(X_train, y_train))
print('Test score: ',ridge.score(X_test, y_test))
print('MAE:', metrics.mean_absolute_error(y_test, ridge_pred))
print('MSE:', metrics.mean_squared_error(y_test, ridge_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, ridge_pred)))
```

```
Train score:  0.6355177115370148
Test score:  0.7245455295324895
MAE: 4.3805767457438565
MSE: 29.07893367358207
RMSE: 5.3924886345343435
```

In [392]:

```
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(X_train,y_train)
print ("Lasso model:", (lasso.coef_))
lasso_pred = lasso.predict(X_test)
```

```
Lasso model: [ 1.01611141 -3.38964076 -5.95843127  2.27549704  2.6363103
0.04386526]
```

In [393]:

```
print('Train score: ',lasso.score(X_train, y_train))
print('Test score: ',lasso.score(X_test, y_test))
print('MAE:', metrics.mean_absolute_error(y_test, lasso_pred))
print('MSE:', metrics.mean_squared_error(y_test, lasso_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, lasso_pred)))
```

```
Train score:  0.6352329928014662
Test score:  0.7229710817372395
MAE: 4.375266445114411
MSE: 29.245143584544454
RMSE: 5.407877918790739
```

-A metric that tells us the mean absolute difference between the predicted values and the actual values in a dataset. The lower the MAE, the better a model fits a dataset.

-1.0 can be considered best possible score and it can be negative because the model can be arbitrarily worse

-RMSLE (Root Mean Squared Logarithmic Error)

-It is a ratio between true value and predicted value

-RMSLE indicate better fit with lesser LOSS if it has lower values

Taking Sample Values

In [394]:

```
mean_val=df.mean()
std_val=df.std()
# year of sale
date=2012.500
date=(date- mean_val[0] )/ std_val[0]
```

In [395]:

```
# house age in years

age= 23

age= (age- mean_val[1] )/ std_val[1]
```

In [396]:

```
# for Distance to nearest metro staion

mrt= 1200
mrt= (mrt- mean_val[2] )/ std_val[2]
```

In [397]:

```
# for number of stores in the Locality

stores=5

stores=(stores- mean_val[3] )/ std_val[3]
```

In [398]:

```
# for Latitude

latitude=24.97

latitude=(latitude- mean_val[4] )/ std_val[4]
```

In [399]:

```
# for Longitude
longitude=121.53

longitude=(longitude- mean_val[5] )/ std_val[5]
```

Prediction

In [400]:

```
test_input= np.array( [[ date, age, mrt, stores, latitude, longitude]] )
```

In [401]:

```
val= model.predict(test_input)
res=val[0][0]
print("The predicted price is=",convert_label_value(res))
```

```
1/1 [==============================] - 0s 31ms/step
The predicted price is= 27
```

In [ ]:

In [ ]:

In [ ]: