Australian Housing Prices prediction By Abdul Wasay

This dataset can be used to predict hosing prices in Australia. This dataset can be used to find relationships between housing prices and location. This dataset can be used to find relationships between housing prices and features such as size, number of bedrooms, and number of bathrooms

Hint: RealEstateAU_1000_Samples.csv file

Instructions:

1. Use Lifecycle of Data Sciece
2. Use necessary data Preprocess techniques
3. Use various Regression and Classification techniques for comparision
4. Use metrics for regression and classification when needed.
5. Use various Pipeline/Hyperparametr tuning techniques for improving performance

In [2]:

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

FileName = 'E:\DSPP\Assignments\JNTUH ML DL assignment 2\RealEstateAU_1000_Samples.xls'
dataset = pd.read_excel(FileName)

# Printing first 5 records of the dataset
dataset.head()
```

Out[2]:

	index	TID	breadcrumb	category_name	property_type	building_size	land_size	preferred_size	open_date	listing_agency	...	state	zip_code
0	0	1350988	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	NaN	Added 2 hours ago	Professionals - DARWIN CITY	...	NT	800
1	1	1350989	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Apartment	171m²|	NaN	171m²|	Added 7 hours ago	Nick Mousellis Real Estate - Eview Group Member	...	NT	800
2	2	1350990	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Unit	NaN	NaN	NaN	Added 22 hours ago	Habitat Real Estate - THE GARDENS	...	NT	800

	index	TID	breadcrumb	category_name	property_type	building_size	land_size	preferred_size	open_date	listing_agency	...	state	zip_code
3	3	1350991	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	House	NaN	NaN	NaN	Added yesterday	Ray White - NIGHTCLIFF	...	NT	800
4	4	1350992	Buy>NT>DARWIN CITY	Real Estate & Property for sale in DARWIN CITY...	Unit	201m²Ĩ	NaN	201m²Ĩ	Added yesterday	Carol Need Real Estate - Fannie Bay	...	NT	800

5 rows × 27 columns

In [3]:
```
dataset.shape
```

Out[3]:
```
(1000, 27)
```

In [ ]:
```
# Data Preprocessing
Categorize the features depending on their datatype (int, float, object) and then calculate the number of them.
```

In [4]:
```
obj = (dataset.dtypes == 'object')
object_cols = list(obj[obj].index)
print("Categorical variables:",len(object_cols))

int_ = (dataset.dtypes == 'int')
num_cols = list(int_[int_].index)
print("Integer variables:",len(num_cols))

fl = (dataset.dtypes == 'float')
fl_cols = list(fl[fl].index)
print("Float variables:",len(fl_cols))
```

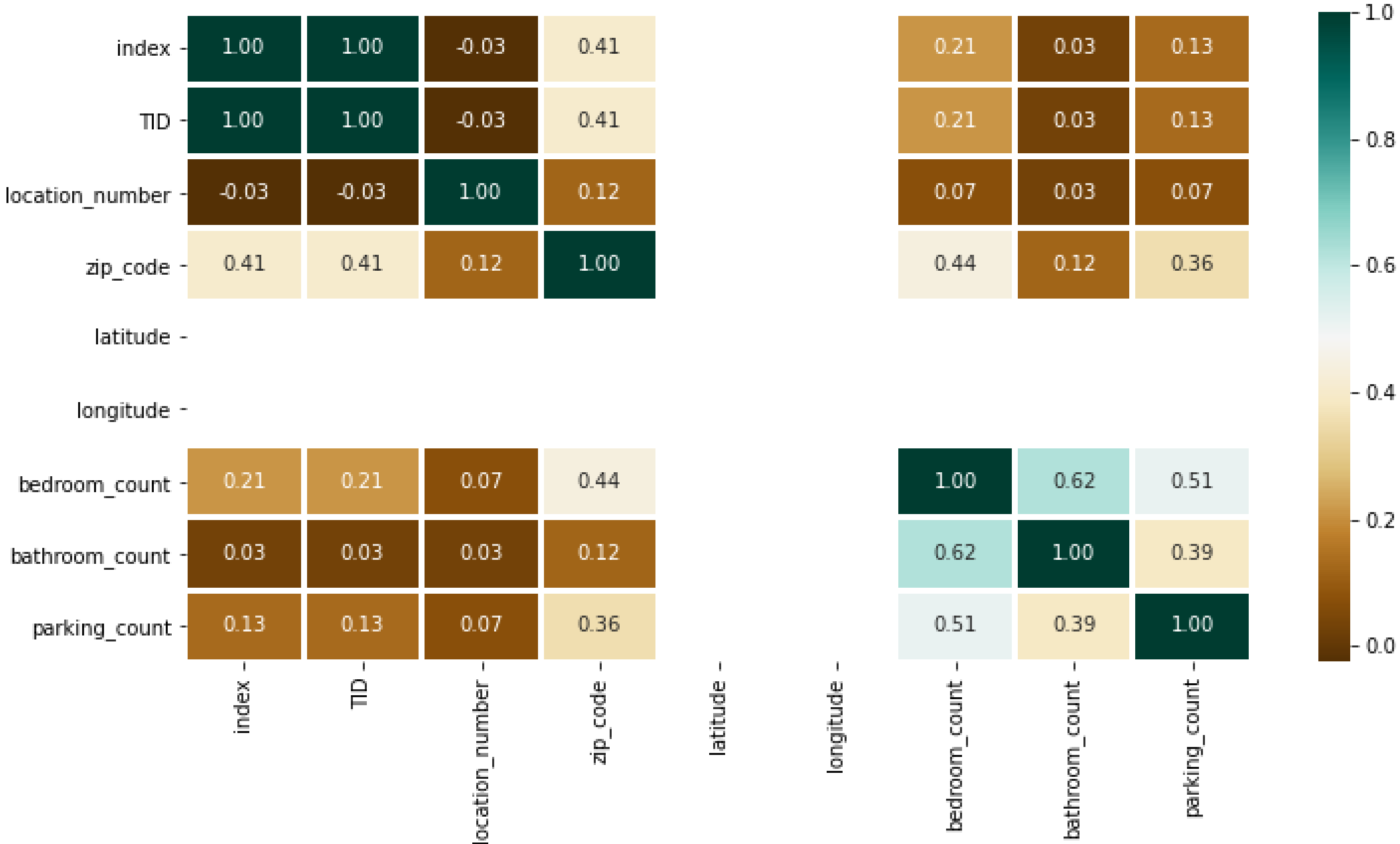
```
Categorical variables: 17
Integer variables: 0
Float variables: 5
```

In [ ]:
```
# Exploratory Data Analysis (EDA)
EDA refers to the deep analysis of data so as to discover different patterns and spot anomalies.
Before making inferences from data it is essential to examine all your variables.

Now let’s make a heatmap using seaborn library.
```

In [5]:
```
plt.figure(figsize=(12, 6))
sns.heatmap(dataset.corr(), cmap = 'BrBG', fmt = '.2f', linewidths = 2, annot = True)
```

Out[5]: <AxesSubplot:>

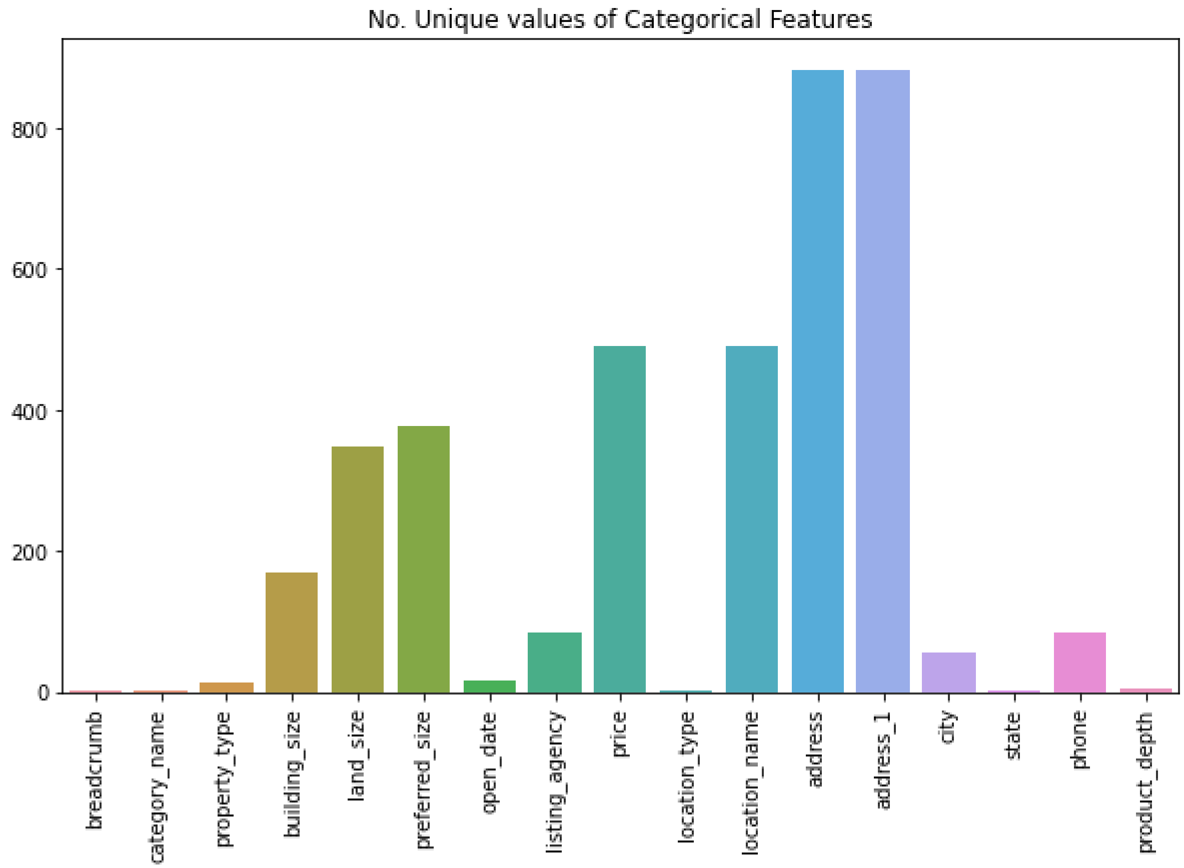


In [ ]: To analyze the different categorical features. Let’s draw the barplot.

In [6]:

```
unique_values = []
for col in object_cols:
    unique_values.append(dataset[col].unique().size)
plt.figure(figsize=(10,6))
plt.title('No. Unique values of Categorical Features')
plt.xticks(rotation=90)
sns.barplot(x=object_cols,y=unique_values)
```

Out[6]: <AxesSubplot:title={'center':'No. Unique values of Categorical Features'}>



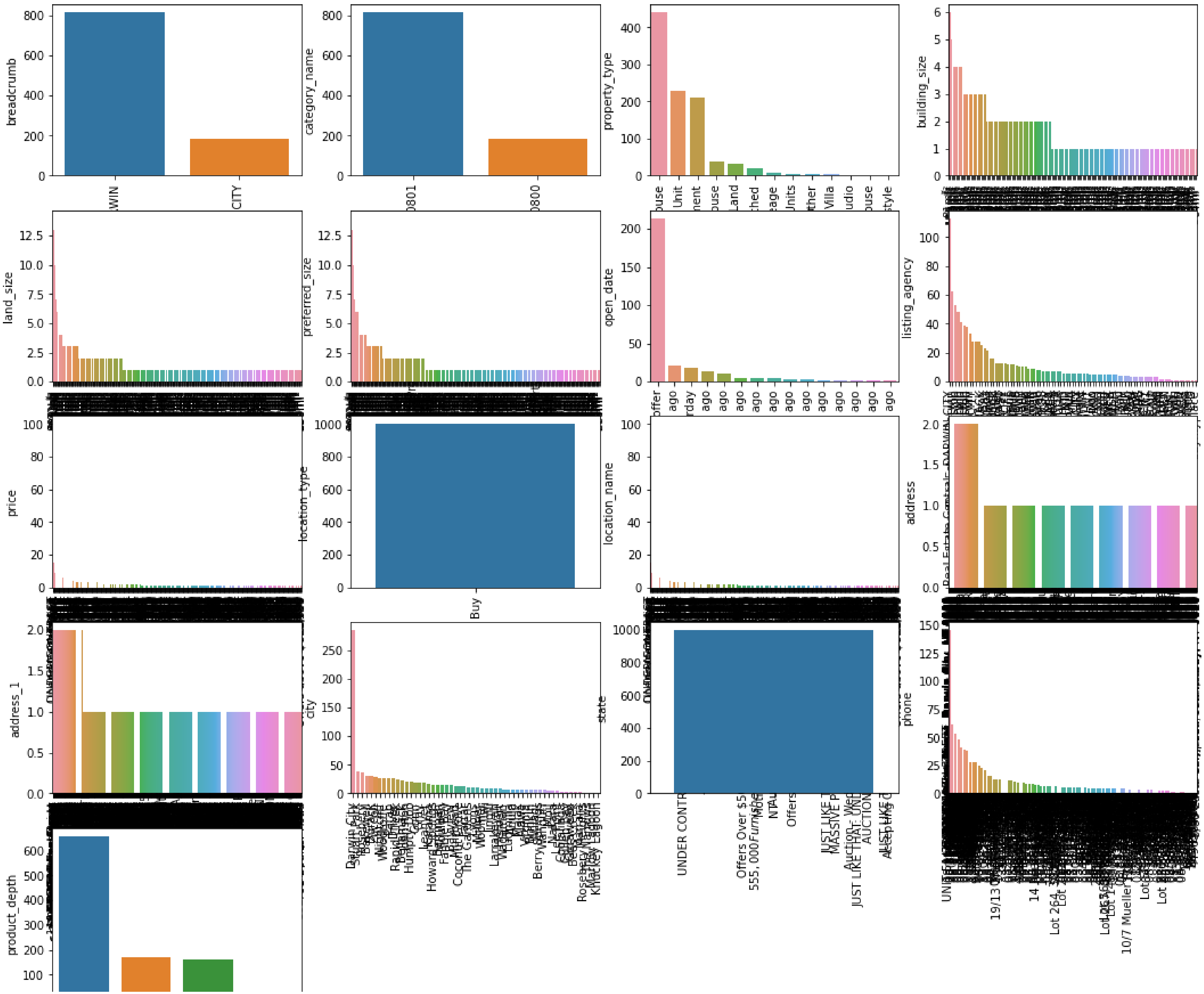
In [ ]:

```
The plot shows that Exterior1st has around 16 unique categories and other features have around  6 unique categories.
To findout the actual count of each category we can plot the bargraph of each four features separately.
```

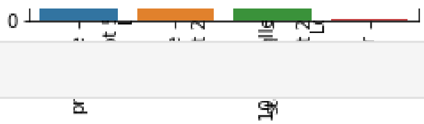
In [7]:

```
plt.figure(figsize=(18, 36))
plt.title('Categorical Features: Distribution')
plt.xticks(rotation=90)
index = 1

for col in object_cols:
    y = dataset[col].value_counts()
    plt.subplot(11, 4, index)
    plt.xticks(rotation=90)
    sns.barplot(x=list(y.index), y=y)
    index += 1
```



breadcrumb
category_name
property_type
building_size
land_size
preferred_size
open_date
listing_agency
price
location_type
Buy
location_name
address
address_1
city
state
phone
product_depth



In [ ]:

Data Cleaning

Data Cleaning is the way to improvise the data or remove incorrect, corrupted or irrelevant data.

As in our dataset, there are some columns that are not important and irrelevant for the model training. So, we can drop that column before training. There are 2 approaches to dealing with empty/null values

We can easily delete the column/row (if the feature or record is not much important). Filling the empty slots with mean/mode/0/NA/etc. (depending on the dataset requirement). As Id Column will not be participating in any prediction. So we can Drop it.

In [9]:
```
dataset.drop(['index'],
             axis=1,
             inplace=True)
```

Replacing SalePrice empty values with their mean values to make the data distribution symmetric.

In [ ]:
```
dataset['price'] = dataset['price'].fillna(
    dataset['price'].mean())
```

Drop records with null values (as the empty records are very less).

In [10]:
```
new_dataset = dataset.dropna()
```

Checking features which have null values in the new dataframe (if there are still any).

In [11]:
```
new_dataset.isnull().sum()
```

Out[11]:
```
TID                 0
breadcrumb          0
category_name       0
property_type       0
building_size       0
land_size           0
preferred_size      0
open_date           0
listing_agency      0
price               0
location_number     0
location_type       0
location_name       0
address             0
```

```
address_1          0
city               0
state              0
zip_code           0
phone              0
latitude           0
longitude          0
product_depth      0
bedroom_count      0
bathroom_count     0
parking_count      0
RunDate            0
dtype: int64
```

OneHotEncoder – For Label categorical features: One hot Encoding is the best way to convert categorical data into binary vectors. This maps the values to integer values. By using OneHotEncoder, we can easily convert object data into int. So for that, firstly we have to collect all the features which have the object datatype. To do so, we will make a loop.

In [13]:
```
from sklearn.preprocessing import OneHotEncoder

s = (new_dataset.dtypes == 'object')
object_cols = list(s[s].index)
print("Categorical variables:")
print(object_cols)
print('No. of. categorical features: ',
      len(object_cols))
```

```
Categorical variables:
['breadcrumb', 'category_name', 'property_type', 'building_size', 'land_size', 'preferred_size', 'open_date', 'listing_agency', 'pri
ce', 'location_type', 'location_name', 'address', 'address_1', 'city', 'state', 'phone', 'product_depth']
No. of. categorical features:  17
```

In [ ]:
```
# Once we have a list of all the features. We can apply OneHotEncoding to the whole list.
```

In [ ]:
```
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols = pd.DataFrame(OH_encoder.fit_transform(new_dataset[object_cols]))
OH_cols.index = new_dataset.index
OH_cols.columns = OH_encoder.get_feature_names()
df_final = new_dataset.drop(object_cols, axis=1)
df_final = pd.concat([df_final, OH_cols], axis=1)
```

Splitting Dataset into Training and Testing X and Y splitting (i.e. Y is the Price column and the rest of the other columns are X)

In [ ]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split

X = df_final.drop(['price'], axis=1)
Y = df_final['price']
```

```
# Split the training set into
# training and validation set
X_train, X_valid, Y_train, Y_valid = train_test_split(
    X, Y, train_size=0.8, test_size=0.2, random_state=0)
```

Model and Accuracy As we have to train the model to determine the continuous values, so we will be using these regression models.

- SVM-Support Vector Machine
- Random Forest Regressor
- Linear Regressor And To calculate loss we will be using the mean_absolute_percentage_error module. It can easily be imported by using sklearn library. The formula for Mean Absolute

SVM – Support vector Machine SVM can be used for both regression and classification model. It finds the hyperplane in the n-dimensional plane.

In [ ]:

```
from sklearn import svm
from sklearn.svm import SVC
from sklearn.metrics import mean_absolute_percentage_error

model_SVR = svm.SVR()
model_SVR.fit(X_train,Y_train)
Y_pred = model_SVR.predict(X_valid)

print(mean_absolute_percentage_error(Y_valid, Y_pred))
```

In [ ]:

```
0.18705129
```

Random Forest Regression Random Forest is an ensemble technique that uses multiple of decision trees and can be used for both regression and classification tasks.

In [ ]:

```
from sklearn.ensemble import RandomForestRegressor

model_RFR = RandomForestRegressor(n_estimators=10)
model_RFR.fit(X_train, Y_train)
Y_pred = model_RFR.predict(X_valid)

mean_absolute_percentage_error(Y_valid, Y_pred)
```

In [ ]:

```
0.1929469
```

Linear Regression Linear Regression predicts the final output-dependent value based on the given independent features. Like, here we have to predict Price depending on features like building_size, land_size, preferred_size, product_depth, bedroom_count, bathroom_count, parking_count etc.

In [ ]:

```
from sklearn.linear_model import LinearRegression

model_LR = LinearRegression()
model_LR.fit(X_train, Y_train)
Y_pred = model_LR.predict(X_valid)

print(mean_absolute_percentage_error(Y_valid, Y_pred))
```

In [ ]:

```
0.187416838
```

Conclusion Clearly, SVM model is giving better accuracy as the mean absolute error is the least among all the other regressor models i.e. 0.18 approx. To get much better results ensemble learning techniques like Bagging and Boosting can also be used.

In [ ]: