In [1]:
```python
import os
print(os.getcwd())
```

```
C:\Users\Lenovo
```

In [2]:
```python
import pandas as pd
```

In [3]:
```python
# save filepath to variable for easier access
me_file_path = './immodata.csv'
# read the data and store data in DataFrame titled me_data
me_data = pd.read_csv(me_file_path)
# print a summary of the data in me data
me_data.describe()
```

Out[3]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000

In [4]:
```python
import pandas as pd

me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)
me_data.columns
```

Out[4]:
```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitud
e',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

Ci-dessous suppression des lignes avec des valeurs manquantes

In [5]:
```python
# dropna drops missing values (think of na as "not available")
me_data = me_data.dropna(axis=0)
```

In [6]:
```python
y = me_data.Price
```

In [7]:
```python
me_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
```

In [8]:
```python
X = me_data[me_features]
```

In [9]:
```python
X.describe()
```

Out[9]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

In [10]:
```python
X.head()
```

Out[10]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
1	2	1.0	156.0	-37.8079	144.9934
2	3	2.0	134.0	-37.8093	144.9944
4	4	1.0	120.0	-37.8072	144.9941
6	3	2.0	245.0	-37.8024	144.9993
7	2	1.0	256.0	-37.8060	144.9954

In [11]:
```python
from sklearn.tree import DecisionTreeRegressor

# Define model. Specify a number for random_state to ensure same results ea
me_model = DecisionTreeRegressor(random_state=1)

# Entrainement du modèle
me_model.fit(X, y)
```

Out[11]:
```
▾        DecisionTreeRegressor
DecisionTreeRegressor(random_state=1)
```

In [12]:
```python
print("Making predictions for the following 5 houses:")
print(X.head())
print("The predictions are")
print(me_model.predict(X.head()))
```

```
Making predictions for the following 5 houses:
   Rooms  Bathroom  Landsize  Lattitude  Longtitude
1      2       1.0     156.0   -37.8079    144.9934
2      3       2.0     134.0   -37.8093    144.9944
4      4       1.0     120.0   -37.8072    144.9941
6      3       2.0     245.0   -37.8024    144.9993
7      2       1.0     256.0   -37.8060    144.9954
The predictions are
[1035000. 1465000. 1600000. 1876000. 1636000.]
```

In [13]:
```
# Filter rows with missing price values
filtered_me_data = me_data.dropna(axis=0)
# Choose target and features
y = filtered_me_data.Price
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = filtered_me_data[fme_features]

from sklearn.tree import DecisionTreeRegressor
# Define model
me_model = DecisionTreeRegressor()
# Fit model
me_model.fit(X, y)
```

Out[13]:
```
▾ DecisionTreeRegressor

DecisionTreeRegressor()
```

In [14]:
```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = me_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

Out[14]:
```
434.71594577146544
```

Ci-dessus l'erreur est faible car le model a déja vu les données lors de l'entrainement. Ci-dessous on garde une petite partie des données pour la validation seulement, le reste des données (la plus grosse partie) étant utilisée pour entrainer le modèle.

In [15]:
```
from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and targe
# The split is based on a random number generator. Supplying a numeric valu
# the random_state argument guarantees we get the same split every time we
# run this script.
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)
# Define model
me_model = DecisionTreeRegressor()
# Fit model
me_model.fit(train_X, train_y)

# get predicted prices on validation data
val_predictions = me_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))
```

```
258669.91542930924
```

In [16]:
```
# Data Loading Code Runs At This Point
import pandas as pd

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)
# Filter rows with missing values
filtered_me_data = me_data.dropna(axis=0)
# Choose target and features
y = filtered_me_data.Price
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = filtered_me_data[fme_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and targe
train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

Ci-dessous étude du paramètre nombre de feuilles

In [17]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor

def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_sta
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)
```

In [18]:
```
# compare MAE with differing values of max_leaf_nodes
for max_leaf_nodes in [5, 50, 150, 250, 500, 1500, 5000]:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes: %d  \t\t Mean Absolute Error:  %d" %(max_leaf_no
```

```
Max leaf nodes: 5                Mean Absolute Error:  347380
Max leaf nodes: 50               Mean Absolute Error:  258171
Max leaf nodes: 150              Mean Absolute Error:  253766
Max leaf nodes: 250              Mean Absolute Error:  247206
Max leaf nodes: 500              Mean Absolute Error:  243495
Max leaf nodes: 1500             Mean Absolute Error:  252130
Max leaf nodes: 5000             Mean Absolute Error:  255575
```

In [19]:
```
import pandas as pd

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)
# Filter rows with missing values
me_data = me_data.dropna(axis=0)
# Choose target and features
y = me_data.Price
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = me_data[fme_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and targe
# The split is based on a random number generator. Supplying a numeric valu
# the random_state argument guarantees we get the same split every time we

train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

Ci-dessous on remplace l'arbre de décision par une forêt aléatoire

In [20]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, preds))
```

```
191669.7536453626
```

In [21]:
```
# Récupération du nombre de feuilles dans chaque arbre de la forêt aléatoir
n_leaves_per_tree = [tree.get_n_leaves() for tree in forest_model.estimator

# Afficher les résultats
for i, n_leaves in enumerate(n_leaves_per_tree):
    print(f"Arbre {i+1}: {n_leaves} feuilles")

# Nombre total de feuilles dans la forêt (si pertinent)
print(f"Nombre total de feuilles dans tous les arbres : {sum(n_leaves_per_t
```

```
Arbre 1: 2850 feuilles
Arbre 2: 2870 feuilles
Arbre 3: 2868 feuilles
Arbre 4: 2908 feuilles
Arbre 5: 2900 feuilles
Arbre 6: 2889 feuilles
Arbre 7: 2845 feuilles
Arbre 8: 2845 feuilles
Arbre 9: 2844 feuilles
Arbre 10: 2854 feuilles
Arbre 11: 2893 feuilles
Arbre 12: 2908 feuilles
Arbre 13: 2835 feuilles
Arbre 14: 2869 feuilles
Arbre 15: 2857 feuilles
Arbre 16: 2916 feuilles
Arbre 17: 2853 feuilles
Arbre 18: 2888 feuilles
Arbre 19: 2889 feuilles
Arbre 20: 2850 feuilles
Arbre 21: 2889 feuilles
Arbre 22: 2876 feuilles
Arbre 23: 2830 feuilles
Arbre 24: 2876 feuilles
Arbre 25: 2855 feuilles
Arbre 26: 2846 feuilles
Arbre 27: 2896 feuilles
Arbre 28: 2881 feuilles
Arbre 29: 2875 feuilles
Arbre 30: 2867 feuilles
Arbre 31: 2884 feuilles
Arbre 32: 2844 feuilles
Arbre 33: 2882 feuilles
Arbre 34: 2903 feuilles
Arbre 35: 2838 feuilles
Arbre 36: 2868 feuilles
Arbre 37: 2906 feuilles
Arbre 38: 2882 feuilles
Arbre 39: 2869 feuilles
Arbre 40: 2853 feuilles
Arbre 41: 2846 feuilles
Arbre 42: 2859 feuilles
Arbre 43: 2882 feuilles
Arbre 44: 2860 feuilles
Arbre 45: 2868 feuilles
Arbre 46: 2858 feuilles
Arbre 47: 2893 feuilles
Arbre 48: 2867 feuilles
Arbre 49: 2884 feuilles
Arbre 50: 2824 feuilles
Arbre 51: 2905 feuilles
Arbre 52: 2852 feuilles
Arbre 53: 2882 feuilles
Arbre 54: 2865 feuilles
Arbre 55: 2892 feuilles
Arbre 56: 2832 feuilles
Arbre 57: 2865 feuilles
Arbre 58: 2856 feuilles
Arbre 59: 2884 feuilles
Arbre 60: 2866 feuilles
```

```
Arbre 61: 2879 feuilles
Arbre 62: 2883 feuilles
Arbre 63: 2879 feuilles
Arbre 64: 2854 feuilles
Arbre 65: 2888 feuilles
Arbre 66: 2855 feuilles
Arbre 67: 2879 feuilles
Arbre 68: 2896 feuilles
Arbre 69: 2875 feuilles
Arbre 70: 2873 feuilles
Arbre 71: 2840 feuilles
Arbre 72: 2835 feuilles
Arbre 73: 2891 feuilles
Arbre 74: 2864 feuilles
Arbre 75: 2866 feuilles
Arbre 76: 2863 feuilles
Arbre 77: 2869 feuilles
Arbre 78: 2854 feuilles
Arbre 79: 2841 feuilles
Arbre 80: 2872 feuilles
Arbre 81: 2898 feuilles
Arbre 82: 2915 feuilles
Arbre 83: 2853 feuilles
Arbre 84: 2896 feuilles
Arbre 85: 2890 feuilles
Arbre 86: 2903 feuilles
Arbre 87: 2925 feuilles
Arbre 88: 2882 feuilles
Arbre 89: 2853 feuilles
Arbre 90: 2864 feuilles
Arbre 91: 2858 feuilles
Arbre 92: 2877 feuilles
Arbre 93: 2869 feuilles
Arbre 94: 2891 feuilles
Arbre 95: 2933 feuilles
Arbre 96: 2912 feuilles
Arbre 97: 2859 feuilles
Arbre 98: 2863 feuilles
Arbre 99: 2880 feuilles
Arbre 100: 2863 feuilles
Nombre total de feuilles dans tous les arbres : 287231
```

On continue l'optimisation du paramètres nombre de feuilles

In [22]:
```
import pandas as pd

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)
# Filter rows with missing values
me_data = me_data.dropna(axis=0)
# Choose target and features
y = me_data.Price
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = me_data[fme_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and targe
# The split is based on a random number generator. Supplying a numeric valu
# the random_state argument guarantees we get the same split every time we

train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

In [23]:
```
def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = RandomForestRegressor(max_leaf_nodes=max_leaf_nodes, random_sta
    model.fit(train_X, train_y)
    preds = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds)
    return(mae)
```

In [24]:
```
# compare MAE with differing values of max_leaf_nodes
for max_leaf_nodes in [2700, 2750, 2800, 2850, 2900, 2950]:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes: %d  \t\t Mean Absolute Error:  %d" %(max_leaf_no
```

```
Max leaf nodes: 2700          Mean Absolute Error:  192499
Max leaf nodes: 2750          Mean Absolute Error:  192496
Max leaf nodes: 2800          Mean Absolute Error:  192496
Max leaf nodes: 2850          Mean Absolute Error:  192497
Max leaf nodes: 2900          Mean Absolute Error:  192497
Max leaf nodes: 2950          Mean Absolute Error:  192497
```

le nombre de feuilles est déja bien optimisé par le modèle

Ci dessous on fixe le paramètre max_depth à 20

In [25]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(max_depth=20, random_state=1)
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, preds))
```

```
191945.7841081549
```

Ci-dessous on remplace (impute) les valeurs manquantes par la moyenne de la colonne plutôt que de supprimer la ligne

In [26]:
```python
import pandas as pd

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)
# Filter rows with missing values
#me_data = me_data.dropna(axis=0)
me_data = me_data.fillna(me_data.mean(numeric_only=True))
# Choose target and features
y = me_data.Price
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = me_data[fme_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and targe
# The split is based on a random number generator. Supplying a numeric valu
# the random_state argument guarantees we get the same split every time we

train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

In [27]:
```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(max_depth=20, random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

```
175294.6787475501
```

In [28]:
```python
import gc
gc.collect()
```

Out[28]:
```
130
```

ci-dessous on recherche les meilleures paramètres pour cette forêt aléatoire

In [29]:
```
import pandas as pd

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)
# Filter rows with missing values
#me_data = me_data.dropna(axis=0)
me_data = me_data.fillna(me_data.mean(numeric_only=True))
# Choose target and features
y = me_data.Price
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = me_data[fme_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and targe
# The split is based on a random number generator. Supplying a numeric valu
# the random_state argument guarantees we get the same split every time we

train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

In [30]:

```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import mean_squared_error

df = pd.DataFrame(me_data)

# Préparation des données
X = df.drop(columns=['Price'])
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = me_data[fme_features]
y = df['Price']

param_grid = {
    # 'n_estimators': Nombre d'arbres dans la forêt.
    'n_estimators': [300, 500],

    # 'max_depth': Profondeur maximale des arbres. None signifie aucune lim
    'max_depth': [20],

    # 'min_samples_split': Nombre minimum d'échantillons requis pour divise
    'min_samples_split': [2, 5, 10, 20],

    # 'min_samples_leaf': Nombre minimum d'échantillons requis pour former
    'min_samples_leaf': [1, 2, 5, 10],

    # 'max_features': Nombre de caractéristiques à considérer pour la reche
    # 'sqrt': racine carrée du nombre total de caractéristiques,
    # 'log2': logarithme base 2,
    # None: toutes les caractéristiques,
    # ou une fraction du total (exemple : 0.5 ou 0.8).
    #'max_features': ['sqrt', 'log2', None, 0.5, 0.8],

    # 'bootstrap': Indique si les échantillons sont tirés avec remplacement
    #'bootstrap': [True, False],


    # 'oob_score': Utiliser ou non des échantillons hors-sacs pour estimer
    #'oob_score': [True, False]

}


model = RandomForestRegressor(random_state=1)

# GridSearchCV
grid_search = GridSearchCV(
    estimator=model,
    param_grid=param_grid,
    cv=3,  # 3-fold cross-validation
    scoring='neg_mean_squared_error',  # Minimize MAE
    n_jobs=-1,  # Utiliser tous les cœurs disponibles
    verbose=1
)

# Entraînement
grid_search.fit(X, y)
```

```
# Meilleurs paramètres
print("Meilleurs paramètres :")
print(grid_search.best_params_)
```

```
Fitting 3 folds for each of 32 candidates, totalling 96 fits
Meilleurs paramètres :
{'max_depth': 20, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estima
tors': 500}
```

In [31]:
```
import gc
gc.collect()
```

Out[31]: 559

In [32]:
```
import pandas as pd

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)
# Filter rows with missing values
#me_data = me_data.dropna(axis=0)
me_data = me_data.fillna(me_data.mean(numeric_only=True))
# Choose target and features
y = me_data.Price
fme_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                        'YearBuilt', 'Lattitude', 'Longtitude']
X = me_data[fme_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and targe
# The split is based on a random number generator. Supplying a numeric valu
# the random_state argument guarantees we get the same split every time we

train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

In [34]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(max_depth=20, min_samples_leaf=2, min_
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

```
173575.73027928325
```

In [35]:
```
print(me_data.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         13580 non-null  object 
 1   Address        13580 non-null  object 
 2   Rooms          13580 non-null  int64  
 3   Type           13580 non-null  object 
 4   Price          13580 non-null  float64
 5   Method         13580 non-null  object 
 6   SellerG        13580 non-null  object 
 7   Date           13580 non-null  object 
 8   Distance       13580 non-null  float64
 9   Postcode       13580 non-null  float64
 10  Bedroom2       13580 non-null  float64
 11  Bathroom       13580 non-null  float64
 12  Car            13580 non-null  float64
 13  Landsize       13580 non-null  float64
 14  BuildingArea   13580 non-null  float64
 15  YearBuilt      13580 non-null  float64
 16  CouncilArea    12211 non-null  object 
 17  Lattitude      13580 non-null  float64
 18  Longtitude     13580 non-null  float64
 19  Regionname     13580 non-null  object 
 20  Propertycount  13580 non-null  float64
dtypes: float64(12), int64(1), object(8)
memory usage: 2.2+ MB
None
```

Ci-dessous, pour les colonnes contenant des valeurs de types objet ou catégorie, on impute les valeurs manquantes , puis on encode le tout pour n'avoir que des chiffres. De ce fait on peut à présent utiliser toutes les colonnes.

In [36]:
```
import gc
gc.collect()
```

Out[36]:
```
26
```

In [37]:
```python
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)

# Separate target and features
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Identify categorical columns (if you have them)
categorical_cols = me_data.select_dtypes(include=['object', 'category']).co

# Impute missing values in categorical columns with the most frequent value
categorical_imputer = SimpleImputer(strategy='most_frequent')
me_data[categorical_cols] = categorical_imputer.fit_transform(me_data[categ

# Encode categorical columns using OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
me_data[categorical_cols] = ordinal_encoder.fit_transform(me_data[categoric

# Impute missing values in numerical columns with mean
me_data = me_data.fillna(me_data.mean(numeric_only=True))

# Re-select target and features (after handling categorical data)
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Split data into training and validation sets
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=0)

# Display the first few rows of the processed training set
print("Processed Training Data:")
print(train_X.head())
```

```
Processed Training Data:
       Suburb  Address  Rooms  Type  Method  SellerG  Date  Distance  \
664      22.0  12798.0      3   0.0     1.0     73.0  24.0       9.2
3270    152.0  13306.0      2   0.0     1.0    146.0  38.0      10.5
3873    185.0  12896.0      2   0.0     1.0    135.0  26.0      11.2
13170   117.0   4148.0      3   0.0     1.0    260.0  28.0      19.6
1730     63.0   8640.0      4   0.0     1.0     82.0   1.0      11.4

       Postcode  Bedroom2  Bathroom  Car  Landsize  BuildingArea    YearBu
ilt  \
664      3104.0       3.0       2.0  2.0     368.0     177.00000  2009.000
000
3270     3081.0       2.0       1.0  2.0     586.0      80.00000  1955.000
000
3873     3145.0       2.0       1.0  1.0     348.0     151.96765  1964.684
217
13170    3076.0       3.0       1.0  1.0     521.0     151.96765  1964.684
217
1730     3163.0       3.0       2.0  2.0     687.0     237.00000  1983.000
000

       CouncilArea  Lattitude  Longtitude  Regionname  Propertycount
664            2.0  -37.78460   145.09350         5.0         7809.0
3270           0.0  -37.74350   145.04860         0.0         2947.0
3873          26.0  -37.86720   145.04320         5.0         8801.0
13170         23.0  -37.63854   145.05179         2.0        10926.0
1730           8.0  -37.89310   145.04790         5.0         7822.0
```

In [38]:

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(max_depth=20, min_samples_leaf=2, min_
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, preds))
```

```
166659.591164046
```

Ci-dessous on détermine l'ordre d'importance des colonnes (caractéristiques ou features)

In [39]:
```python
from sklearn.feature_selection import mutual_info_classif
from sklearn.feature_selection import mutual_info_regression
from sklearn.preprocessing import LabelEncoder

df = pd.DataFrame(me_data)

# Separate features and target
X = df.drop(columns=['Price'])
y = df['Price']

# Compute Mutual Information scores (use mutual_info_classif for classifica
mi_scores = mutual_info_classif(X, y, random_state=1)

# Create a DataFrame to display the MI scores
mi_df = pd.DataFrame({
    'Feature': X.columns,
    'MI Score': mi_scores
}).sort_values(by='MI Score', ascending=False)

# Display the results
print("Mutual Information Scores:")
print(mi_df)
```

```
Mutual Information Scores:
          Feature  MI Score
4          Method  1.061313
10       Bathroom  0.700808
2           Rooms  0.494255
9        Bedroom2  0.488549
11            Car  0.428901
13   BuildingArea  0.370939
18     Regionname  0.345288
3            Type  0.288458
14      YearBuilt  0.185663
8        Postcode  0.151054
12       Landsize  0.130984
15    CouncilArea  0.119861
17     Longtitude  0.113064
16      Lattitude  0.111982
5         SellerG  0.097315
7        Distance  0.087107
0          Suburb  0.085528
19  Propertycount  0.080805
1         Address  0.043487
6            Date  0.012298
```

Ci-dessous on rajoute des colonnes créée à partir des autres et on supprime quelques colonnes ayant un MI score faible

In [40]:
```python
import gc
gc.collect()
```

Out[40]:
```
2106
```

```
In [41]: import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)

# Separate target and features
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Drop specific columns
columns_to_drop = ['SellerG', 'Distance', 'Suburb', 'Propertycount', 'Addre
X = X.drop(columns=columns_to_drop)

# Identify categorical columns (if you have them)
categorical_cols = me_data.select_dtypes(include=['object', 'category']).co

# Impute missing values in categorical columns with the most frequent value
categorical_imputer = SimpleImputer(strategy='most_frequent')
me_data[categorical_cols] = categorical_imputer.fit_transform(me_data[categ

# Encode categorical columns using OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
me_data[categorical_cols] = ordinal_encoder.fit_transform(me_data[categoric

# Impute missing values in numerical columns with mean
me_data = me_data.fillna(me_data.mean(numeric_only=True))

# Re-select target and features (after handling categorical data)
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

X['Metbath'] = X['Method'] + X['Bathroom']
X['Batroo'] = X['Bathroom'] + X['Rooms']
X['Roometh'] = X['Rooms'] + X['Method']


# Split data into training and validation sets
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=0)

# Initialize Random Forest model
forest_model = RandomForestRegressor(max_depth=20, min_samples_leaf=2, min_

# Fit model on training data and evaluate on validation set
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)
print("Validation MAE:", mean_absolute_error(val_y, preds))
```

```
Validation MAE: 165189.922322392
```

In [43]:
```python
print(X.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13580 entries, 0 to 13579
Data columns (total 23 columns):
 #   Column         Non-Null Count  Dtype  
---  ------         --------------  -----  
 0   Suburb         13580 non-null  float64
 1   Address        13580 non-null  float64
 2   Rooms          13580 non-null  int64  
 3   Type           13580 non-null  float64
 4   Method         13580 non-null  float64
 5   SellerG        13580 non-null  float64
 6   Date           13580 non-null  float64
 7   Distance       13580 non-null  float64
 8   Postcode       13580 non-null  float64
 9   Bedroom2       13580 non-null  float64
 10  Bathroom       13580 non-null  float64
 11  Car            13580 non-null  float64
 12  Landsize       13580 non-null  float64
 13  BuildingArea   13580 non-null  float64
 14  YearBuilt      13580 non-null  float64
 15  CouncilArea    13580 non-null  float64
 16  Lattitude      13580 non-null  float64
 17  Longtitude     13580 non-null  float64
 18  Regionname     13580 non-null  float64
 19  Propertycount  13580 non-null  float64
 20  Metbath        13580 non-null  float64
 21  Batroo         13580 non-null  float64
 22  Roometh        13580 non-null  float64
dtypes: float64(22), int64(1)
memory usage: 2.4 MB
None
```

Ci-dessous, pour utiliser toutes les données pour l'entrainement mais aussi pour la validation on utilise la validation croisée

In [44]:
```python
import gc
gc.collect()
```

Out[44]:
```
26
```

In [45]:

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Load data
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)

# Separate target and features
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Drop specific columns
columns_to_drop = ['SellerG', 'Distance', 'Suburb', 'Propertycount', 'Addre
X = X.drop(columns=columns_to_drop)

# Identify categorical columns (if you have them)
categorical_cols = me_data.select_dtypes(include=['object', 'category']).co

# Impute missing values in categorical columns with the most frequent value
categorical_imputer = SimpleImputer(strategy='most_frequent')
me_data[categorical_cols] = categorical_imputer.fit_transform(me_data[categ

# Encode categorical columns using OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
me_data[categorical_cols] = ordinal_encoder.fit_transform(me_data[categoric

# Impute missing values in numerical columns with mean
me_data = me_data.fillna(me_data.mean(numeric_only=True))

# Re-select target and features (after handling categorical data)
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

X['Metbath'] = X['Method'] + X['Bathroom']
X['Batroo'] = X['Bathroom'] + X['Rooms']
X['Roometh'] = X['Rooms'] + X['Method']


# Split data into training and validation sets
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=0)

# Initialize Random Forest model
forest_model = RandomForestRegressor(max_depth=20, min_samples_leaf=2, min_

# Perform 5-fold cross-validation
cv_scores = cross_val_score(forest_model, train_X, train_y, cv=5, scoring='

# Convert negative MAE to positive
cv_mae_scores = -cv_scores

# Print Cross-Validation Results
print("Cross-Validation MAE Scores:", cv_mae_scores)
print("Average MAE:", cv_mae_scores.mean())

# Fit model on training data and evaluate on validation set
```

```python
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)
print("Validation MAE:", mean_absolute_error(val_y, preds))
```

```
Cross-Validation MAE Scores: [161306.79200785 165336.71922253 169892.31592
179 169055.39766353
 165392.54275476]
Average MAE: 166196.7535140908
Validation MAE: 165189.922322392
```

Ci-dessous on utilise PCA pour rajouter des caractéristiques

In [46]:
```python
import gc
gc.collect()
```

Out[46]:
```
1845
```

In [47]:
```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.decomposition import PCA
from sklearn.metrics import mean_absolute_error

# Charger les données
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)

# Séparer la cible (Price) et les caractéristiques (features)
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Drop specific columns
columns_to_drop = ['SellerG', 'Distance', 'Suburb', 'Propertycount', 'Addre
X = X.drop(columns=columns_to_drop)

# Identifier les colonnes catégoriques (si elles existent)
categorical_cols = me_data.select_dtypes(include=['object', 'category']).co

# Imputation des valeurs manquantes dans les colonnes catégoriques avec la 
categorical_imputer = SimpleImputer(strategy='most_frequent')
me_data[categorical_cols] = categorical_imputer.fit_transform(me_data[categ

# Encoder les colonnes catégoriques avec OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
me_data[categorical_cols] = ordinal_encoder.fit_transform(me_data[categoric

# Imputation des valeurs manquantes dans les colonnes numériques avec la mo
me_data = me_data.fillna(me_data.mean(numeric_only=True))

# Re-sélectionner la cible (Price) et les caractéristiques (features) après
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Ajout de nouvelles combinaisons de colonnes (caractéristiques dérivées)
X['Metbath'] = X['Method'] + X['Bathroom']
X['Batroo'] = X['Bathroom'] + X['Rooms']
X['Roometh'] = X['Rooms'] + X['Method']

# Normalisation des données avant d'appliquer PCA
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Application de PCA pour générer de nouvelles caractéristiques
pca = PCA(n_components=3)  # Choisir 3 composantes principales
X_pca = pca.fit_transform(X_scaled)

# Ajouter les composantes principales comme nouvelles colonnes
pca_columns = [f'PCA_{i+1}' for i in range(X_pca.shape[1])]
X_pca_df = pd.DataFrame(X_pca, columns=pca_columns, index=X.index)
X = pd.concat([X, X_pca_df], axis=1)

# Division des données en ensembles d'entraînement et de validation
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=0)
```

```python
# Initialisation du modèle Random Forest
forest_model = RandomForestRegressor(
    max_depth=20,
    min_samples_leaf=2,
    min_samples_split=2,
    n_estimators=500,
    random_state=1,
    n_jobs=-1
)

# Validation croisée (5-fold)
cv_scores = cross_val_score(forest_model, train_X, train_y, cv=5, scoring='

# Convertir MAE négatif en positif
cv_mae_scores = -cv_scores

# Résultats de la validation croisée
print("Scores MAE de la validation croisée :", cv_mae_scores)
print("MAE moyen :", cv_mae_scores.mean())

# Entraîner le modèle sur les données d'entraînement et évaluer sur l'ensem
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)
print("MAE de validation :", mean_absolute_error(val_y, preds))
```

```
Scores MAE de la validation croisée : [163760.54014799 166817.25215536 171
912.8976703  170162.97164856
 167501.31992147]
MAE moyen : 168030.9963087337
MAE de validation : 166108.84406032477
```

ci-dessous on utilise KMeans en plus de PCA pour rajouter des caractéristiques

In [50]:
```python
import gc
gc.collect()
```

Out[50]: 1059

In [51]:
```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.ensemble import RandomForestRegressor
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import mean_absolute_error

# Charger les données
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)

# Séparer la cible (Price) et les caractéristiques (features)
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Drop specific columns
columns_to_drop = ['SellerG', 'Distance', 'Suburb', 'Propertycount', 'Addre
X = X.drop(columns=columns_to_drop)

# Identifier les colonnes catégoriques (si elles existent)
categorical_cols = me_data.select_dtypes(include=['object', 'category']).co

# Imputation des valeurs manquantes dans les colonnes catégoriques avec la
categorical_imputer = SimpleImputer(strategy='most_frequent')
me_data[categorical_cols] = categorical_imputer.fit_transform(me_data[categ

# Encoder les colonnes catégoriques avec OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
me_data[categorical_cols] = ordinal_encoder.fit_transform(me_data[categoric

# Imputation des valeurs manquantes dans les colonnes numériques avec la mo
me_data = me_data.fillna(me_data.mean(numeric_only=True))

# Re-sélectionner la cible (Price) et les caractéristiques (features) après
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Ajout de nouvelles combinaisons de colonnes (caractéristiques dérivées)
X['Metbath'] = X['Method'] + X['Bathroom']
X['Batroo'] = X['Bathroom'] + X['Rooms']
X['Roometh'] = X['Rooms'] + X['Method']

# Normalisation des données avant d'appliquer PCA et KMeans
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Application de PCA pour générer de nouvelles caractéristiques
pca = PCA(n_components=3)  # Choisir 3 composantes principales
X_pca = pca.fit_transform(X_scaled)

# Ajouter les composantes principales comme nouvelles colonnes
pca_columns = [f'PCA_{i+1}' for i in range(X_pca.shape[1])]
X_pca_df = pd.DataFrame(X_pca, columns=pca_columns, index=X.index)
X = pd.concat([X, X_pca_df], axis=1)

# Application de KMeans pour ajouter des caractéristiques basées sur les cl
kmeans = KMeans(n_clusters=5, random_state=0, n_init=10)  # Choisir un nomb
```

```python
X['Cluster'] = kmeans.fit_predict(X_scaled)  # Ajouter les labels des clust

# Ajouter la distance à chaque centroïde comme caractéristiques
cluster_distances = kmeans.transform(X_scaled)  # Distances aux centroïdes
distance_columns = [f'Distance_to_cluster_{i+1}' for i in range(cluster_dis
cluster_distances_df = pd.DataFrame(cluster_distances, columns=distance_col
X = pd.concat([X, cluster_distances_df], axis=1)

# Division des données en ensembles d'entraînement et de validation
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=0)

# Initialisation du modèle Random Forest
forest_model = RandomForestRegressor(
    max_depth=20,
    min_samples_leaf=2,
    min_samples_split=2,
    n_estimators=500,
    random_state=1,
    n_jobs=-1
)

# Validation croisée (5-fold)
cv_scores = cross_val_score(forest_model, train_X, train_y, cv=5, scoring='

# Convertir MAE négatif en positif
cv_mae_scores = -cv_scores

# Résultats de la validation croisée
print("Scores MAE de la validation croisée :", cv_mae_scores)
print("MAE moyen :", cv_mae_scores.mean())

# Entraîner le modèle sur les données d'entraînement et évaluer sur l'ensem
forest_model.fit(train_X, train_y)
preds = forest_model.predict(val_X)
print("MAE de validation :", mean_absolute_error(val_y, preds))
```

```
Scores MAE de la validation croisée : [164994.49269142 168607.4667327  172
135.29435762 168174.69198906
 169647.09272245]
MAE moyen : 168711.80769865023
MAE de validation : 167105.6643718445
```

Ci-desous on remplace RandomForest par XGBoost

In [52]:
```python
import gc
gc.collect()
```

Out[52]:
```
2389
```

In [54]:
```
!pip install xgboost
```

```
Collecting xgboost
  Downloading xgboost-2.1.3-py3-none-win_amd64.whl (124.9 MB)
     -------------------------------------- 124.9/124.9 MB 4.2 MB/s eta
0:00:00
Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\lib\sit
e-packages (from xgboost) (1.10.0)
Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\lib\sit
e-packages (from xgboost) (1.23.5)
Installing collected packages: xgboost
Successfully installed xgboost-2.1.3
```

In [55]:
```python
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import mean_absolute_error
from xgboost import XGBRegressor  # Importation de XGBoost

# Charger les données
me_file_path = './immodata.csv'
me_data = pd.read_csv(me_file_path)

# Séparer la cible (Price) et les caractéristiques (features)
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Identifier les colonnes catégoriques (si elles existent)
categorical_cols = me_data.select_dtypes(include=['object', 'category']).co

# Imputation des valeurs manquantes dans les colonnes catégoriques avec la 
categorical_imputer = SimpleImputer(strategy='most_frequent')
me_data[categorical_cols] = categorical_imputer.fit_transform(me_data[categ

# Encoder les colonnes catégoriques avec OrdinalEncoder
ordinal_encoder = OrdinalEncoder()
me_data[categorical_cols] = ordinal_encoder.fit_transform(me_data[categoric

# Imputation des valeurs manquantes dans les colonnes numériques avec la mo
me_data = me_data.fillna(me_data.mean(numeric_only=True))

# Re-sélectionner la cible (Price) et les caractéristiques (features) après
y = me_data.Price
X = me_data.drop(['Price'], axis=1)

# Ajout de nouvelles combinaisons de colonnes (caractéristiques dérivées)
X['Metbath'] = X['Method'] + X['Bathroom']
X['Batroo'] = X['Bathroom'] + X['Rooms']
X['Roometh'] = X['Rooms'] + X['Method']

# Normalisation des données avant d'appliquer PCA et KMeans
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Application de PCA pour générer de nouvelles caractéristiques
pca = PCA(n_components=3)  # Choisir 3 composantes principales
X_pca = pca.fit_transform(X_scaled)

# Ajouter les composantes principales comme nouvelles colonnes
pca_columns = [f'PCA_{i+1}' for i in range(X_pca.shape[1])]
X_pca_df = pd.DataFrame(X_pca, columns=pca_columns, index=X.index)
X = pd.concat([X, X_pca_df], axis=1)

# Application de KMeans pour ajouter des caractéristiques basées sur les cl
kmeans = KMeans(n_clusters=5, random_state=0, n_init=10)  # Choisir un nomb
X['Cluster'] = kmeans.fit_predict(X_scaled)  # Ajouter les labels des clust

# Ajouter la distance à chaque centroïde comme caractéristiques
cluster_distances = kmeans.transform(X_scaled)  # Distances aux centroïdes
```

```python
distance_columns = [f'Distance_to_cluster_{i+1}' for i in range(cluster_dis
cluster_distances_df = pd.DataFrame(cluster_distances, columns=distance_colu
X = pd.concat([X, cluster_distances_df], axis=1)

# Division des données en ensembles d'entraînement et de validation
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=0)

# Initialisation du modèle XGBoost
xgb_model = XGBRegressor(
    max_depth=6,  # Profondeur maximale de chaque arbre
    learning_rate=0.01,  # Taux d'apprentissage
    n_estimators=1000,  # Nombre d'arbres
    subsample=0.7,  # Fraction des échantillons utilisés par arbre
    colsample_bytree=0.7,  # Fraction des caractéristiques utilisées par ar
    random_state=1,  # Reproductibilité
    min_child_weight=1, # minimum number of houses in a leaf
    reg_alpha=0.5, # L1 regularization (like LASSO)
    reg_lambda=1.0, # L2 regularization (like Ridge)
    num_parallel_tree=1,
    n_jobs=-1,  # Utiliser tous les cœurs disponibles
)

# Validation croisée (5-fold)
cv_scores = cross_val_score(xgb_model, train_X, train_y, cv=5, scoring='neg_

# Convertir MAE négatif en positif
cv_mae_scores = -cv_scores

# Résultats de la validation croisée
print("Scores MAE de la validation croisée :", cv_mae_scores)
print("MAE moyen :", cv_mae_scores.mean())

# Entraîner le modèle sur les données d'entraînement et évaluer sur l'ensem
xgb_model.fit(train_X, train_y)
preds = xgb_model.predict(val_X)
print("MAE de validation :", mean_absolute_error(val_y, preds))
```

```
Scores MAE de la validation croisée : [158390.59402614 160178.04180474 165
607.48838672 161238.49320385
 160729.57923724]
MAE moyen : 161228.83933173786
MAE de validation : 159301.39090574373
```