projectles prévisions de prix immobiliers à Paris

January 13, 2023

Contents

1 Description du projet

Le projet s'intéresse au prix de l'immobilier sur Paris. Est-on en mesure d'avoir une bonne prédiction sur la valeur mobilière d'un bien

2 Lecture des jeux de données

Les jeux de données sont disponibles sur https://www.data.gouv.fr/fr/datasets/demandes-de-valeurs-foncieres-geolocalisees/

[1]:
```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

[2]:
```
# Fonction pour lire les donnnées en fonction du fichier
def ReadFile(nomFile, delimiter = '|'):
    # lecture du fichier excel
    df = pd.read_csv(nomFile, delimiter = delimiter, low_memory = False)
    print("taille du jeu de donnees :", df.shape)
    return df
```

[3]:
```
# Fonction pour extraire les données à partir d'un numéro de département
def ExtractDepartement(df, numDep):
    df['code_departement'].astype(str)
    df['Validation'] = (df['code_departement'] == numDep )
    dfDep = df[df['Validation']==True]
    dfDep = dfDep.drop('Validation', axis=1)
    print("Departement : {0}".format(numDep))
    print("Taille du jeu de donnees", dfDep.shape)
    return dfDep
```

[4]:
```
df1 = ReadFile("../input/AvecCoordonneesGeo/full.csv", ',')
```

```
taille du jeu de donnees : (1429093, 40)
```

[5]:
```
df2 = ReadFile("../input/AvecCoordonneesGeo/full2021.csv", ',')
```

```
taille du jeu de donnees : (4375223, 40)
```

2.1 Concaténation des données

[6]:
```
#Concaténation des deux jeux de données
df = pd.concat([df1,df2])
#,keys=['2022','2021'])
#, names = ['FileInput', 'RowId'])
print("taille suite à union :", df.shape)
```

```
taille suite à union : (5804316, 40)
```

2.2 Réduction à un département

[7]:
```
dfDep = ExtractDepartement(df,'75')
dfDepIni = dfDep
```

```
Departement : 75
Taille du jeu de donnees (130696, 40)
```

3 Nettoyage du jeu de données

3.1 Nettoyage

[8]:
```
#Suppression des lignes en doublon
dfDep.drop_duplicates(inplace=True)
dfDep.shape
```

[8]:
```
(119224, 40)
```

[9]:
```
#Suppression des longitudes et latitudes null
dfDep.drop(dfDep[(dfDep['longitude'].isnull()) | (dfDep['latitude'].isnull())].
 ↪index, inplace=True)
dfDep.shape
```

[9]:
```
(119182, 40)
```

[10]:
```
#Suppression des données où la valeur foncière est null
dfDep.drop(dfDep[dfDep['valeur_fonciere'].isnull() ].index, inplace=True)
dfDep.shape
```

[10]:
```
(118329, 40)
```

[11]:
```
#Suppression des valeurs foncières < 50KE et >3000KE
dfDep.drop(dfDep[dfDep['valeur_fonciere']<50000 ].index, inplace=True)
dfDep.drop(dfDep[dfDep['valeur_fonciere']>3000000 ].index, inplace=True)
dfDep.shape
```

[11]: (99469, 40)

[12]:
```
#Suppression des variables qui sont nulles pour 80% des valeurs
listVariables = dfDep.isnull().sum() > (dfDep.shape[0]*0.8)
listResultatsVarDrop = []
for colname, serie in listVariables.iteritems():
    if(serie == True):
            listResultatsVarDrop.append(colname)
listResultatsVarDrop
dfDep.drop(listResultatsVarDrop, inplace=True, axis=1)
dfDep.shape
```

[12]: (99469, 23)

[13]:
```
#Conversion des objets en string
dfDep['adresse_nom_voie'] = dfDep['adresse_nom_voie'].astype("string")
dfDep['adresse_numero'] = dfDep['adresse_numero'].astype("string")
dfDep['nom_commune'] = dfDep['nom_commune'].astype("string")
dfDep['adresse_complete']=dfDep['adresse_numero']+'␣
↪'+dfDep['adresse_nom_voie']+' , '+dfDep['nom_commune']
```

[14]:
```
#Suppression des données où le type de local est une dépendance
dfDep.drop(dfDep[dfDep['type_local']== 'Dépendance' ].index, inplace=True)
dfDep.shape
```

[14]: (59687, 24)

[15]:
```
dfDep.head(10)
```

[15]:

	id_mutation	date_mutation	numero_disposition	nature_mutation	\
1379722	2022-514000	2022-01-04	1	Vente	
1379723	2022-514000	2022-01-04	1	Vente	
1379725	2022-514001	2022-01-06	1	Vente	
1379732	2022-514004	2022-01-05	1	Vente	
1379734	2022-514005	2022-01-05	1	Vente	
1379736	2022-514006	2022-01-07	1	Vente	
1379738	2022-514007	2022-01-06	1	Vente	
1379739	2022-514008	2022-01-04	1	Vente	
1379740	2022-514009	2022-01-04	1	Vente	
1379742	2022-514010	2022-01-06	1	Vente	

	valeur_fonciere	adresse_numero	adresse_nom_voie	\
1379722	580000.0	13.0	RUE DE SOFIA	
1379723	580000.0	13.0	RUE DE SOFIA	
1379725	605000.0	51.0	RUE CHARLOT	
1379732	716250.0	6.0	RUE PAUL ESCUDIER	
1379734	320000.0	4.0	RUE DU CHATEAU LANDON	
1379736	320000.0	134.0	AV GAMBETTA	

```
1379738         220000.0            9.0  RUE ELYSEE MENILMONTANT
1379739         280000.0           18.0            RUE DES HAIES
1379740         200000.0          195.0            RUE DE CRIMEE
1379742         677500.0           79.0      RUE DES GRAVILLIERS

        adresse_code_voie  code_postal code_commune  ... lot1_surface_carrez  \
1379722              9002      75018.0        75118  ...                 NaN
1379723              9002      75018.0        75118  ...               61.00
1379725              1880      75003.0        75103  ...               40.66
1379732              7155      75009.0        75109  ...                 NaN
1379734              1924      75010.0        75110  ...                 NaN
1379736              3933      75020.0        75120  ...               32.52
1379738              3192      75020.0        75120  ...               34.53
1379739              4452      75020.0        75120  ...               24.59
1379740              2443      75019.0        75119  ...               27.64
1379742              4302      75003.0        75103  ...               57.12

        lot2_numero nombre_lots code_type_local   type_local  \
1379722          56           2             2.0  Appartement
1379723          58           3             2.0  Appartement
1379725         NaN           1             2.0  Appartement
1379732           3           3             2.0  Appartement
1379734          92           2             2.0  Appartement
1379736          50           2             2.0  Appartement
1379738          24           2             2.0  Appartement
1379739          83           2             2.0  Appartement
1379740          54           2             2.0  Appartement
1379742           4           2             2.0  Appartement

        surface_reelle_bati  nombre_pieces_principales  longitude   latitude  \
1379722                20.0                        2.0   2.348168  48.884490
1379723                25.0                        2.0   2.348168  48.884490
1379725                42.0                        3.0   2.362871  48.863374
1379732                69.0                        3.0   2.332324  48.880353
1379734                33.0                        2.0   2.362613  48.879658
1379736                29.0                        1.0   2.405513  48.872782
1379738                36.0                        2.0   2.386648  48.869335
1379739                28.0                        2.0   2.400622  48.852508
1379740                27.0                        2.0   2.375845  48.891167
1379742                58.0                        3.0   2.353479  48.864674

                                      adresse_complete
1379722       13.0 RUE DE SOFIA , Paris 18e Arrondissement
1379723       13.0 RUE DE SOFIA , Paris 18e Arrondissement
1379725         51.0 RUE CHARLOT , Paris 3e Arrondissement
1379732    6.0 RUE PAUL ESCUDIER , Paris 9e Arrondissement
1379734  4.0 RUE DU CHATEAU LANDON , Paris 10e Arrondis...
```

```
1379736       134.0 AV GAMBETTA , Paris 20e Arrondissement
1379738  9.0 RUE ELYSEE MENILMONTANT , Paris 20e Arrond...
1379739      18.0 RUE DES HAIES , Paris 20e Arrondissement
1379740     195.0 RUE DE CRIMEE , Paris 19e Arrondissement
1379742  79.0 RUE DES GRAVILLIERS , Paris 3e Arrondisse...

[10 rows x 24 columns]
```

[16]:
```python
def AggregationSimilarData(df):

    # Construction d'un dictionnaire
    # où la clé est la chaine de caractère qui permet d'indiquer que deux lignes␣
↪sont similaires
    # où la valeur est l'index dans le dataframe initial
    dict_similarData = {}
    for index,series in df.iterrows():
        keyRow =␣
↪str(series['date_mutation'])+'_'+str(series['valeur_fonciere'])+'_'+series['adresse_complete'
        if keyRow in dict_similarData:
            listIndexSimilaire = dict_similarData[keyRow]
            listIndexSimilaire.append(index)
        else:
            listKeyRow = list();
            listKeyRow.append(index)
            dict_similarData[keyRow] = listKeyRow

    #Suppression des valeurs dupliquées en prenant comme surface_reelle_bati le␣
↪cumulé des surfaces
    listIndexASupprimer = []
    for cle,listIndex in dict_similarData.items():
        if(len(listIndex)>1):
            valSurfaceAgregee = df.at[listIndex[0],"surface_reelle_bati"]
            val = 1
            while (val != len(listIndex)):
                listIndexASupprimer.append(listIndex[val])
                valSurfaceAgregee += df.at[listIndex[val],"surface_reelle_bati"]
                val += 1
            df.at[listIndex[0],"surface_reelle_bati"] = valSurfaceAgregee
    #print(listIndexASupprimer)
    df.drop(listIndexASupprimer, inplace = True, axis = 0)
    print(df.shape)
```

3.2 Gestion des doublons

[17]:
```python
AggregationSimilarData(dfDep)
```

```
(54289, 24)
```

3.3 Gestion des variables catégorielles

On regarde les valeurs uniques pour identifier les variables catégorielles

[18]:
```
for colname, serie in dfDep.iteritems():
    print(colname + " has " + str(serie.drop_duplicates().shape[0]) + " unique␣
 ↪values.")
```

```
id_mutation has 54084 unique values.
date_mutation has 453 unique values.
numero_disposition has 3 unique values.
nature_mutation has 5 unique values.
valeur_fonciere has 15118 unique values.
adresse_numero has 376 unique values.
adresse_nom_voie has 3429 unique values.
adresse_code_voie has 3419 unique values.
code_postal has 21 unique values.
code_commune has 20 unique values.
nom_commune has 20 unique values.
code_departement has 1 unique values.
id_parcelle has 25386 unique values.
lot1_numero has 1872 unique values.
lot1_surface_carrez has 10423 unique values.
lot2_numero has 1875 unique values.
nombre_lots has 18 unique values.
code_type_local has 4 unique values.
type_local has 4 unique values.
surface_reelle_bati has 497 unique values.
nombre_pieces_principales has 16 unique values.
longitude has 22986 unique values.
latitude has 21254 unique values.
adresse_complete has 26739 unique values.
```

[19]:
```
dfDep["nature_mutation"] = pd.Categorical(dfDep["nature_mutation"],␣
 ↪ordered=False)
dfDep["type_local"] = pd.Categorical(dfDep["type_local"], ordered=False)
dfDep["nombre_pieces_principales"] = pd.
 ↪Categorical(dfDep["nombre_pieces_principales"], ordered=False)
dfDep["nom_commune"] = pd.Categorical(dfDep["nom_commune"], ordered=False)
```

[20]:
```
#Suppression des variables qui semblent inutiles
dfDep.drop(['code_departement', 'code_postal', 'adresse_code_voie',␣
 ↪'code_commune', 'id_parcelle','lot1_numero','lot2_numero', 'code_type_local'],␣
 ↪inplace=True, axis=1)
dfDep.shape
```

[20]: (54289, 16)

3.4 Typage des variables

[21]:
```
dfDep['date_mutation'] = pd.to_datetime(dfDep['date_mutation'], format='%Y/%m/
↪%d')
```

[22]:
```
dfDep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 54289 entries, 1379722 to 4375222
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   id_mutation                54289 non-null  object
 1   date_mutation              54289 non-null  datetime64[ns]
 2   numero_disposition         54289 non-null  int64
 3   nature_mutation            54289 non-null  category
 4   valeur_fonciere            54289 non-null  float64
 5   adresse_numero             54288 non-null  string
 6   adresse_nom_voie           54288 non-null  string
 7   nom_commune                54289 non-null  category
 8   lot1_surface_carrez        32599 non-null  float64
 9   nombre_lots                54289 non-null  int64
 10  type_local                 53930 non-null  category
 11  surface_reelle_bati        53912 non-null  float64
 12  nombre_pieces_principales  53929 non-null  category
 13  longitude                  54289 non-null  float64
 14  latitude                   54289 non-null  float64
 15  adresse_complete           54288 non-null  string
dtypes: category(4), datetime64[ns](1), float64(5), int64(2), object(1),
string(3)
memory usage: 5.6+ MB
```

[23]:
```
#Conversion des objets en string
dfDep['id_mutation'] = dfDep['id_mutation'].astype("string")
dfDep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 54289 entries, 1379722 to 4375222
Data columns (total 16 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   id_mutation                54289 non-null  string
 1   date_mutation              54289 non-null  datetime64[ns]
 2   numero_disposition         54289 non-null  int64
 3   nature_mutation            54289 non-null  category
 4   valeur_fonciere            54289 non-null  float64
 5   adresse_numero             54288 non-null  string
 6   adresse_nom_voie           54288 non-null  string
 7   nom_commune                54289 non-null  category
```

```
 8   lot1_surface_carrez        32599 non-null  float64
 9   nombre_lots                54289 non-null  int64
 10  type_local                 53930 non-null  category
 11  surface_reelle_bati        53912 non-null  float64
 12  nombre_pieces_principales  53929 non-null  category
 13  longitude                  54289 non-null  float64
 14  latitude                   54289 non-null  float64
 15  adresse_complete           54288 non-null  string
dtypes: category(4), datetime64[ns](1), float64(5), int64(2), string(4)
memory usage: 5.6 MB
```

4 Exploration

4.1 Description univariée

4.1.1 La variable de temps

On effectue du feature engineering

[24]:
```
dfDep['month']=dfDep["date_mutation"].apply(lambda x: x.month)
dfDep['day'] = dfDep["date_mutation"].apply(lambda x: x.day)
dfDep['year'] = dfDep["date_mutation"].apply(lambda x: x.year)
dfDep["month"] = pd.Categorical(dfDep["month"], ordered=True)
dfDep["day"] = pd.Categorical(dfDep["day"], ordered=True)
dfDep["year"]= pd.Categorical(dfDep["year"], ordered=True)
```

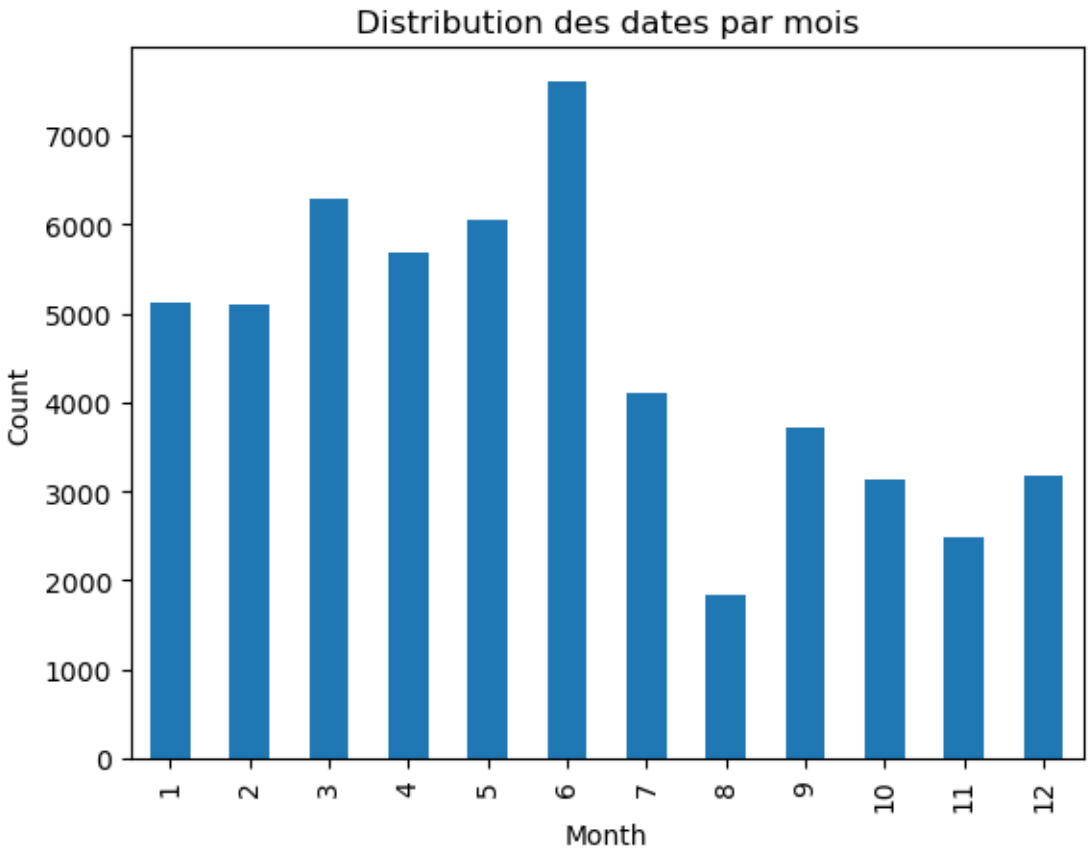
On va représenter les variables catégorielles à travers des tableaux de contingence ou des bars plots

[25]:
```
dfDep["year"].value_counts()
```

[25]:
```
2021    35393
2022    18896
Name: year, dtype: int64
```

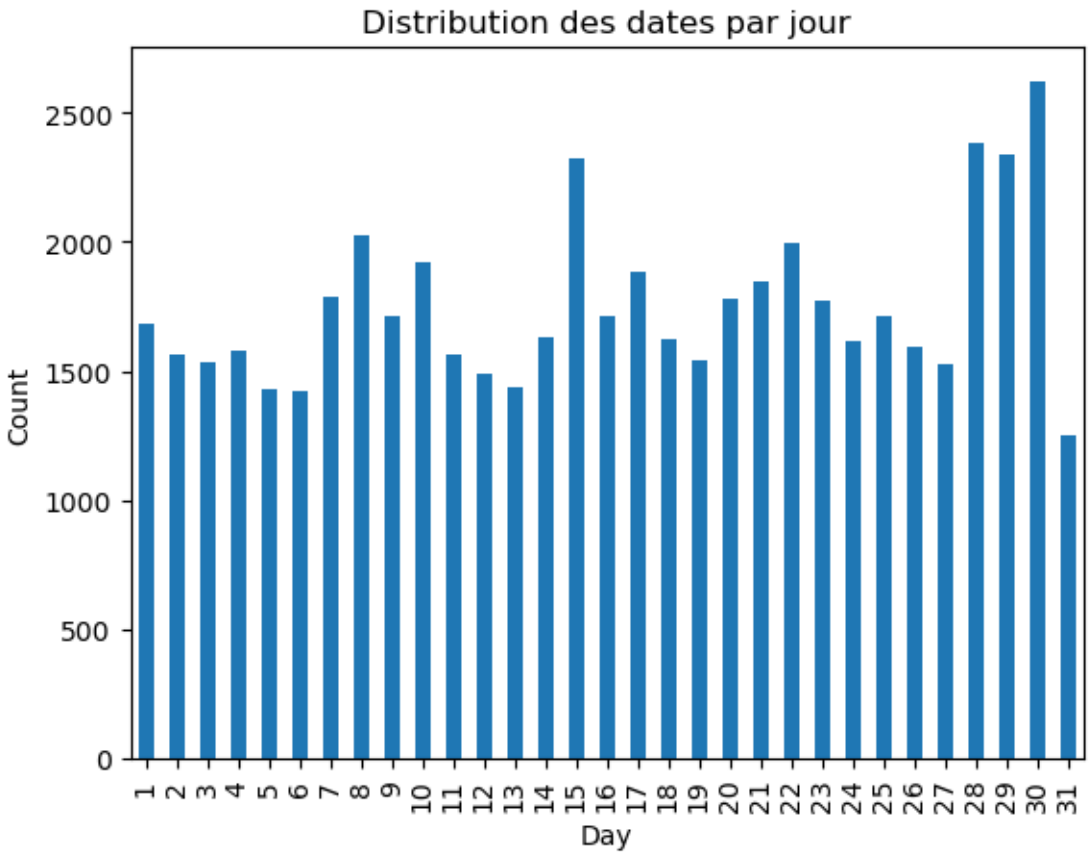
On observe qu'on a pratiquement le double de données entre 2021 et 2022 ce qui est normal car on a une vision partielle de 2022 avec des données jusqu'à Juin

[26]:
```
dfDep["month"].value_counts().sort_index().plot(kind="bar")
plt.title("Distribution des dates par mois")
plt.xlabel("Month")
plt.ylabel("Count")
plt.show()
```



On retrouve les données partielles. Par contre, il apparait difficile de faire des conclusions sur les mois hormi que le mois d'août semble relativement faible ce qui peut s'expliquer car les personnes sont en vacances

[27]:
```
dfDep["day"].value_counts().sort_index().plot(kind="bar")
plt.title("Distribution des dates par jour")
plt.xlabel("Day")
plt.ylabel("Count")
plt.show()
```



Il semblerait qu'il y ait plus de ventes en milieu et fin de mois

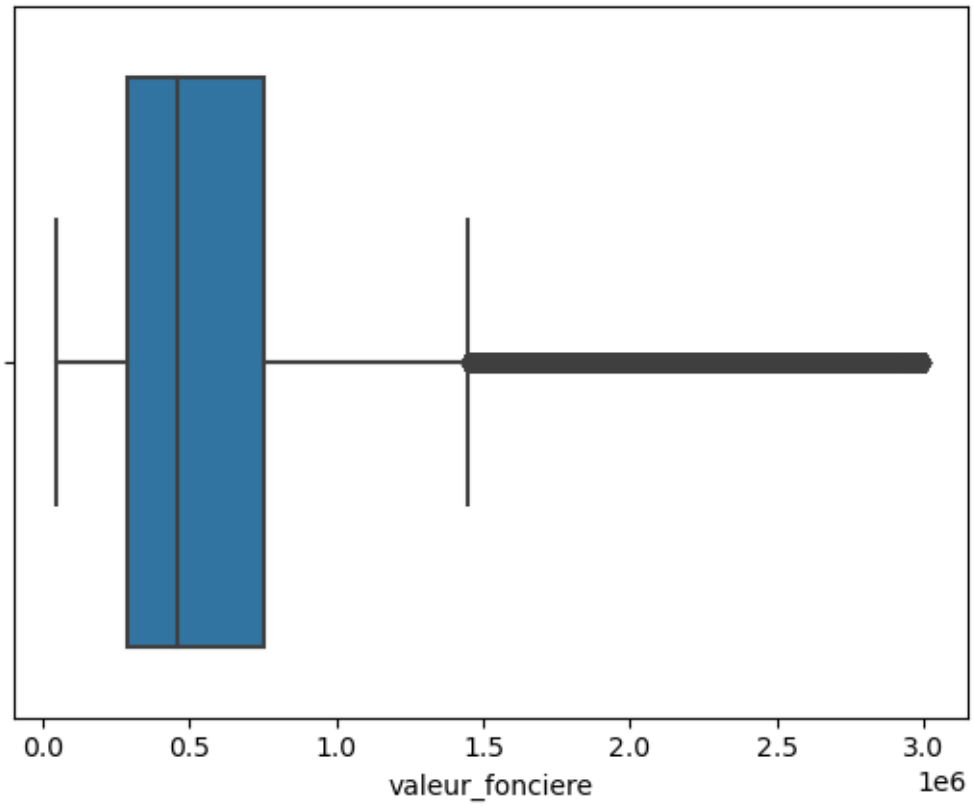
4.1.2 La cible de notre modèle

[28]:
```
dfDep["valeur_fonciere"].describe()
```

[28]:
```
count    5.428900e+04
mean     6.115892e+05
std      4.902127e+05
min      5.000000e+04
25%      2.920000e+05
50%      4.585697e+05
75%      7.550000e+05
max      3.000000e+06
Name: valeur_fonciere, dtype: float64
```

[29]:
```
sns.boxplot(x=dfDep['valeur_fonciere'])
```

[29]:
```
<AxesSubplot:xlabel='valeur_fonciere'>
```



On a clairement des problèmes d'échelle avec des outliers à supprimer

[30]:
```
#Methode Remove outliers pour une loi biaisée
def removeOutliers(variable):
    print("avant ", dfDep.shape)
    Q1 = dfDep[variable].quantile(0.25)
    Q3 = dfDep[variable].quantile(0.75)
    IQR = Q3 - Q1
    dfDep.drop(dfDep[(dfDep[variable]<Q1 - 1.5*IQR) | (dfDep[variable]>Q3 + 1.
 ↪5*IQR)].index, inplace=True)
    print("après ",dfDep.shape)
```

[31]:
```
removeOutliers('valeur_fonciere')
```

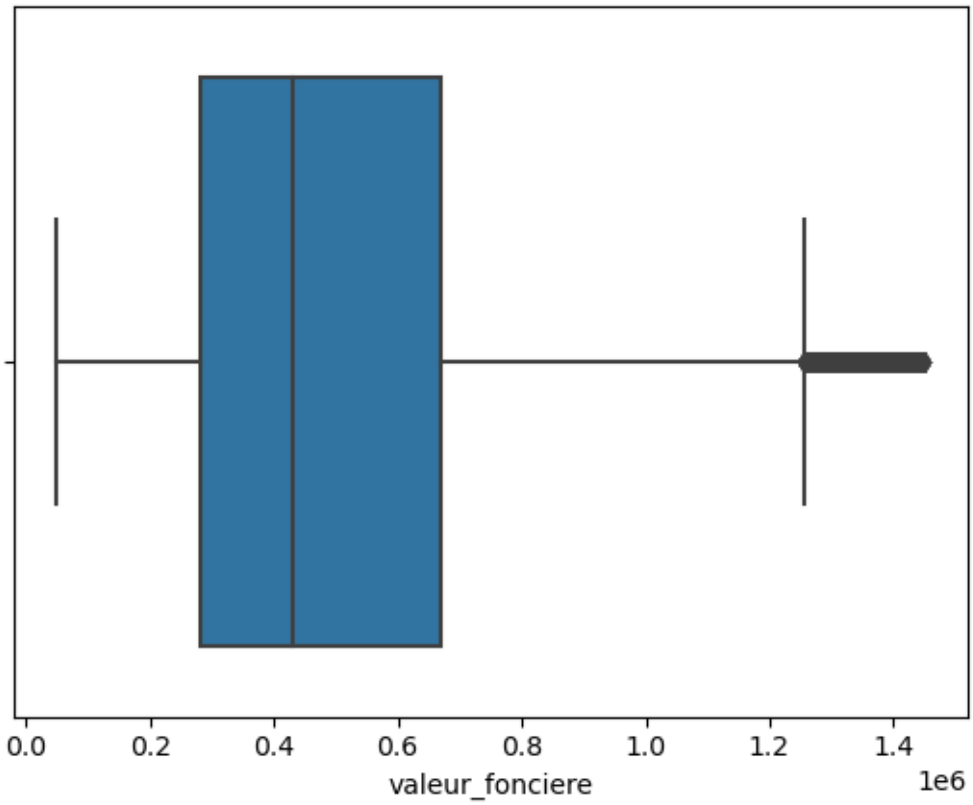
```
avant  (54289, 19)
après  (50446, 19)
```

[32]:
```
#Methode Remove outliers par une loi normale
#Mean = dfDep['valeur_fonciere'].mean()
#StandardDeviation = dfDep['valeur_fonciere'].std()
#dfDep.drop(dfDep[(dfDep['valeur_fonciere']<Mean - 3*StandardDeviation) |␣
 ↪(dfDep['valeur_fonciere']>Mean + 3*StandardDeviation)].index, inplace=True)
```

```
#dfDep.shape
```

[33]:
```
sns.boxplot(x=dfDep['valeur_fonciere'])
dfDep['valeur_fonciere'].describe()
```

[33]:
```
count    5.044600e+04
mean     5.078709e+05
std      3.049878e+05
min      5.000000e+04
25%      2.800000e+05
50%      4.300000e+05
75%      6.700000e+05
max      1.449300e+06
Name: valeur_fonciere, dtype: float64
```



4.1.3 Les autres variables quantitatives

Regardons les variables abérrantes sur les max sur les m2 et mettons une valeur max à 500 m2

[34]:
```
#dfDep.drop(dfDep[dfDep['surface_reelle_bati']>500].index, inplace=True, axis=0)
#dfDep.drop(dfDep[dfDep['lot1_surface_carrez']>500].index, inplace=True, axis=0)
```

```
#dfDep.shape
#removeOutliers('surface_reelle_bati')
#dfDep.describe()
```

[35]:
```
#removeOutliers('lot1_surface_carrez')
```

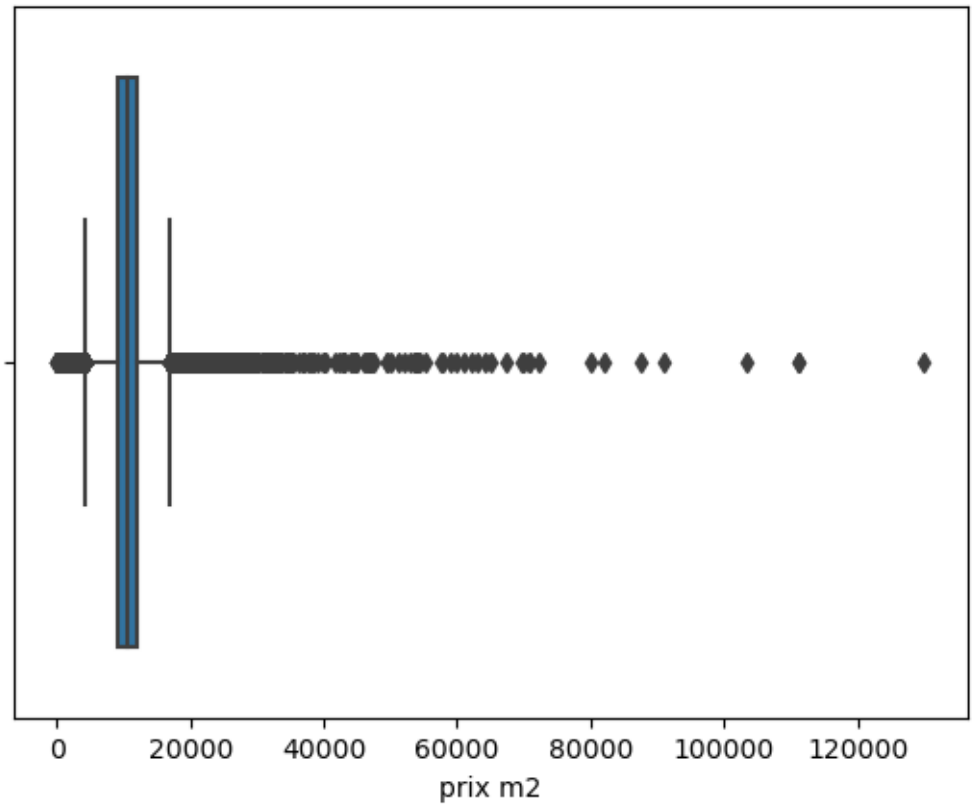
[36]:
```
dfDep.describe()
```

[36]:
```
       numero_disposition  valeur_fonciere  lot1_surface_carrez   nombre_lots  \
count        50446.000000     5.044600e+04         30741.000000  50446.000000
mean             1.002795     5.078709e+05            45.776034      1.675534
std              0.053541     3.049878e+05            61.569145      0.824916
min              1.000000     5.000000e+04             1.000000      0.000000
25%              1.000000     2.800000e+05            25.580000      1.000000
50%              1.000000     4.300000e+05            38.750000      2.000000
75%              1.000000     6.700000e+05            59.270000      2.000000
max              3.000000     1.449300e+06          7392.000000     34.000000

       surface_reelle_bati     longitude      latitude
count         50112.000000  50446.000000  50446.000000
mean             51.898567      2.342636     48.862166
std              66.661178      0.037447      0.020222
min               2.000000      2.255896     48.818759
25%              28.000000      2.316022     48.845022
50%              43.000000      2.345814     48.862759
75%              65.000000      2.373166     48.880174
max            7842.000000      2.412825     48.900565
```

[37]:
```
dfDep["prix m2"]=dfDep["valeur_fonciere"]/dfDep["surface_reelle_bati"]
sns.boxplot(x=dfDep['prix m2'])
dfDep["prix m2"].describe()
```

[37]:
```
count     50112.000000
mean      10639.642826
std        3691.344154
min          24.323322
25%        9000.000000
50%       10557.634033
75%       12117.835294
max      130000.000000
Name: prix m2, dtype: float64
```



[38]:
```
removeOutliers("prix m2")
```

```
avant  (50446, 20)
après  (47154, 20)
```

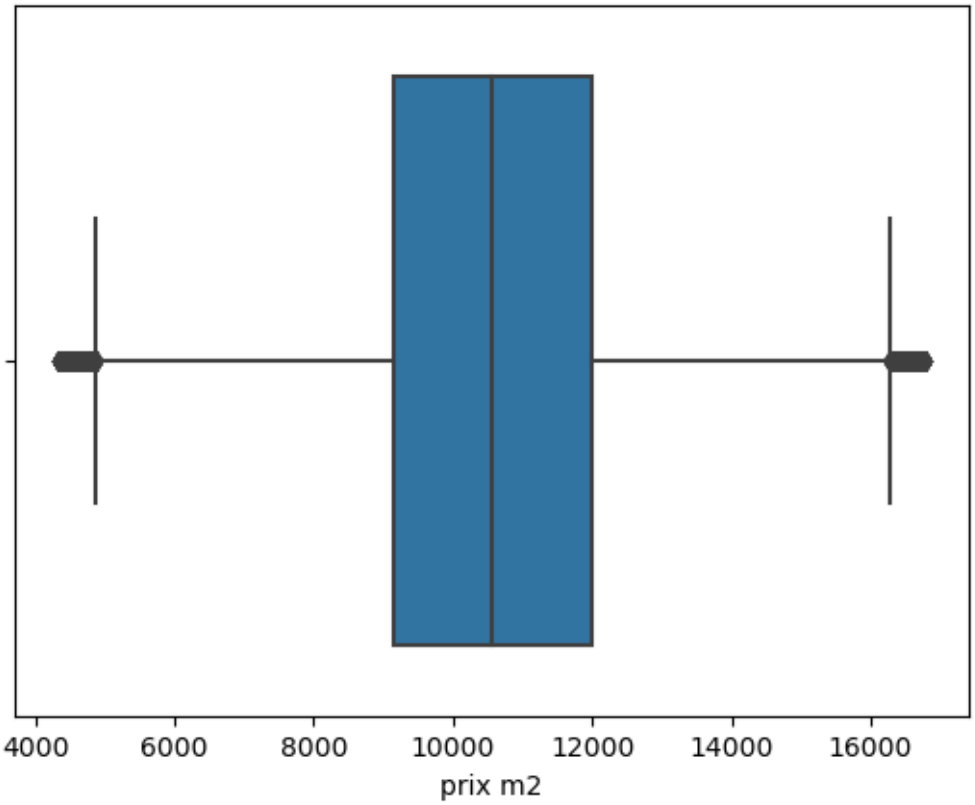
[39]:
```
dfDep["prix m2"]=dfDep["valeur_fonciere"]/dfDep["surface_reelle_bati"]
sns.boxplot(x=dfDep['prix m2'])
dfDep.describe()
```

[39]:

	numero_disposition	valeur_fonciere	lot1_surface_carrez	nombre_lots	\
count	47154.000000	4.715400e+04	29010.000000	47154.000000	
mean	1.002312	5.095385e+05	45.737030	1.679751	
std	0.048899	2.988223e+05	62.355009	0.779361	
min	1.000000	5.000000e+04	1.000000	0.000000	
25%	1.000000	2.870000e+05	25.840000	1.000000	
50%	1.000000	4.330000e+05	38.940000	2.000000	
75%	1.000000	6.686625e+05	59.220000	2.000000	
max	3.000000	1.449300e+06	7392.000000	16.000000	

	surface_reelle_bati	longitude	latitude	prix m2
count	46820.000000	47154.000000	47154.000000	46820.000000

mean	49.006279	2.343005	48.862179	10560.249890
std	28.651969	0.037685	0.020387	2317.756634
min	4.000000	2.255896	48.818759	4326.923077
25%	28.000000	2.316104	48.844722	9146.073718
50%	42.000000	2.346336	48.862890	10566.666667
75%	64.000000	2.373824	48.880446	12000.000000
max	300.000000	2.412825	48.900565	16791.044776



[40]:
```
removeOutliers("lot1_surface_carrez")
sns.boxplot(x=dfDep['lot1_surface_carrez'])
dfDep.describe()
```

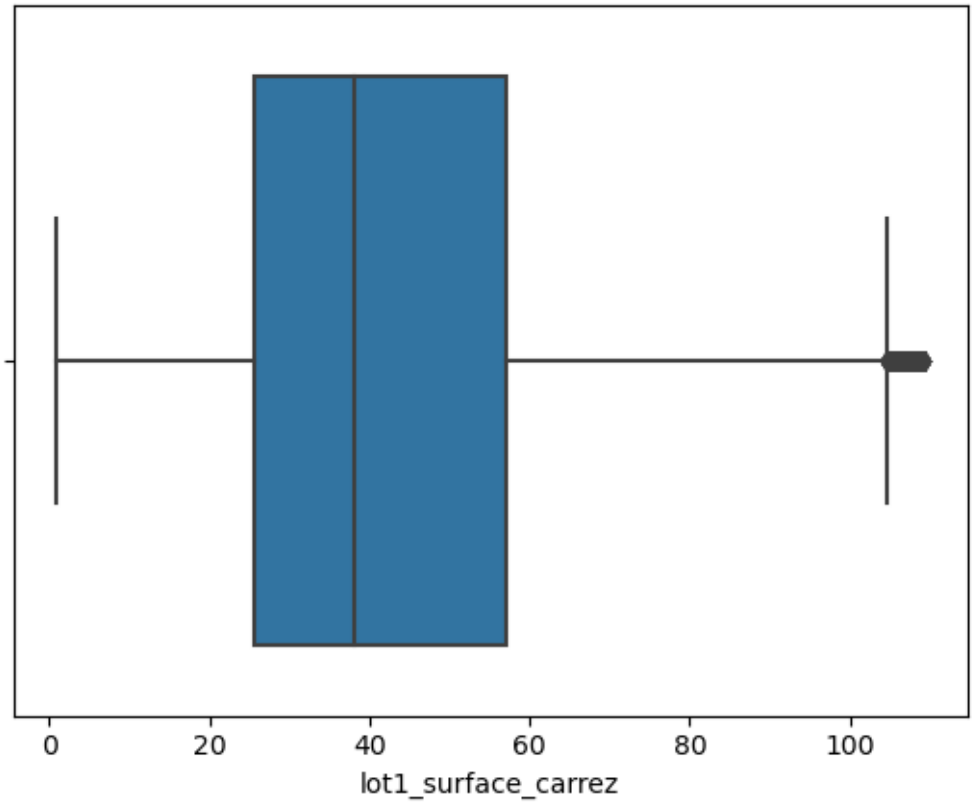
```
avant  (47154, 20)
après  (46482, 20)
```

[40]:

	numero_disposition	valeur_fonciere	lot1_surface_carrez	nombre_lots	\
count	46482.000000	4.648200e+04	28338.000000	46482.000000	
mean	1.002302	4.996812e+05	43.209783	1.676864	
std	0.048814	2.883507e+05	23.095905	0.774179	
min	1.000000	5.000000e+04	1.000000	0.000000	
25%	1.000000	2.850000e+05	25.530000	1.000000	

50%	1.000000	4.290000e+05	38.160000	2.000000
75%	1.000000	6.530150e+05	57.170000	2.000000
max	3.000000	1.449200e+06	109.270000	16.000000

	surface_reelle_bati	longitude	latitude	prix m2
count	46149.000000	46482.000000	46482.000000	46149.000000
mean	47.905610	2.343216	48.862159	10570.283401
std	27.152457	0.037587	0.020415	2317.846738
min	4.000000	2.255896	48.818759	4326.923077
25%	28.000000	2.316626	48.844628	9155.172414
50%	42.000000	2.346446	48.862869	10573.333333
75%	63.000000	2.373929	48.880462	12000.000000
max	300.000000	2.412825	48.900565	16791.044776

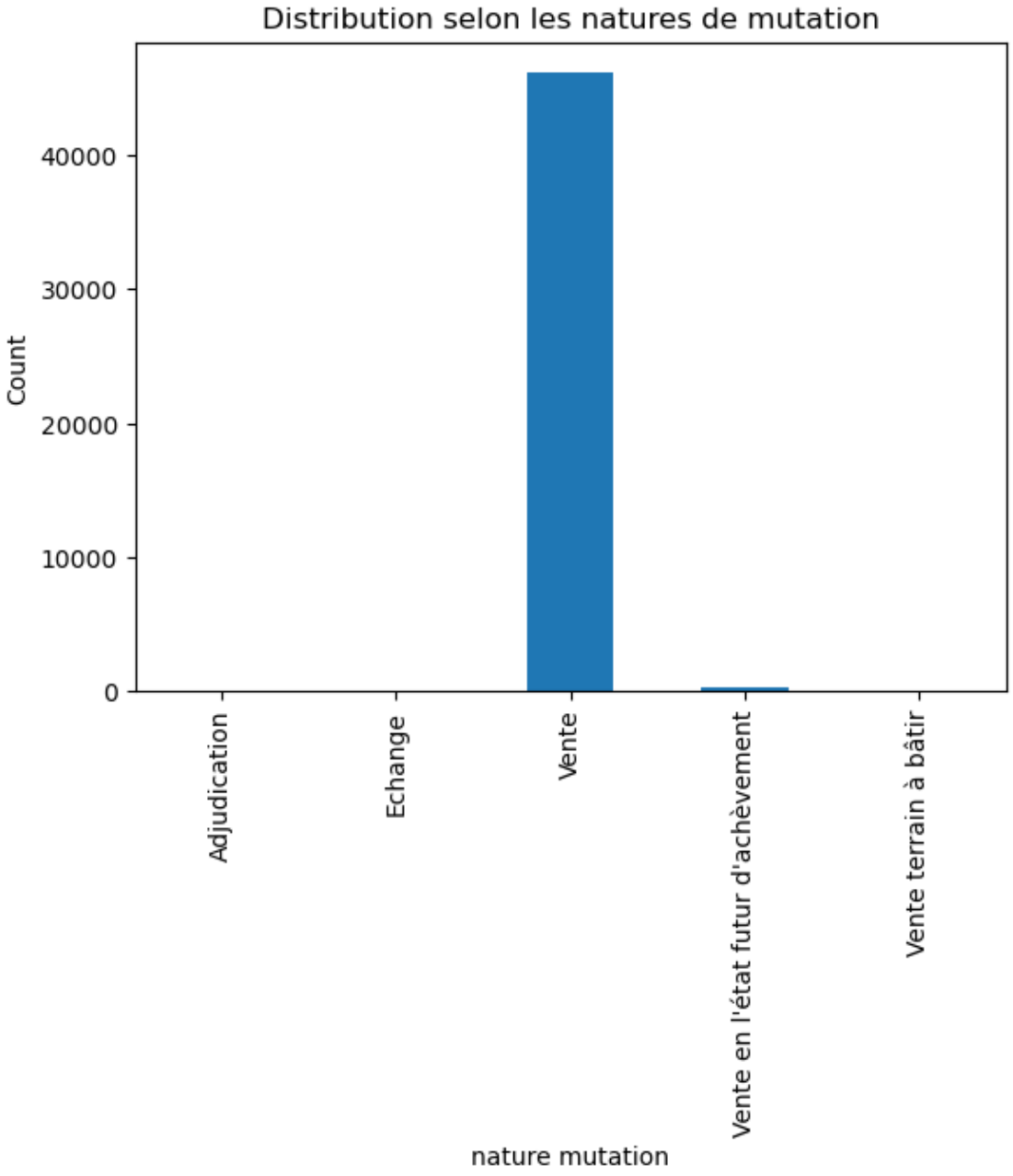


Les données sur les variables quantitatives semblent cohérentes en terme de grandeur suite à différentes suppressions des données atypiques.

[41]:
```
dfDep.drop(["prix m2"],inplace=True,axis=1)
```

4.1.4 Les variables catégorielles

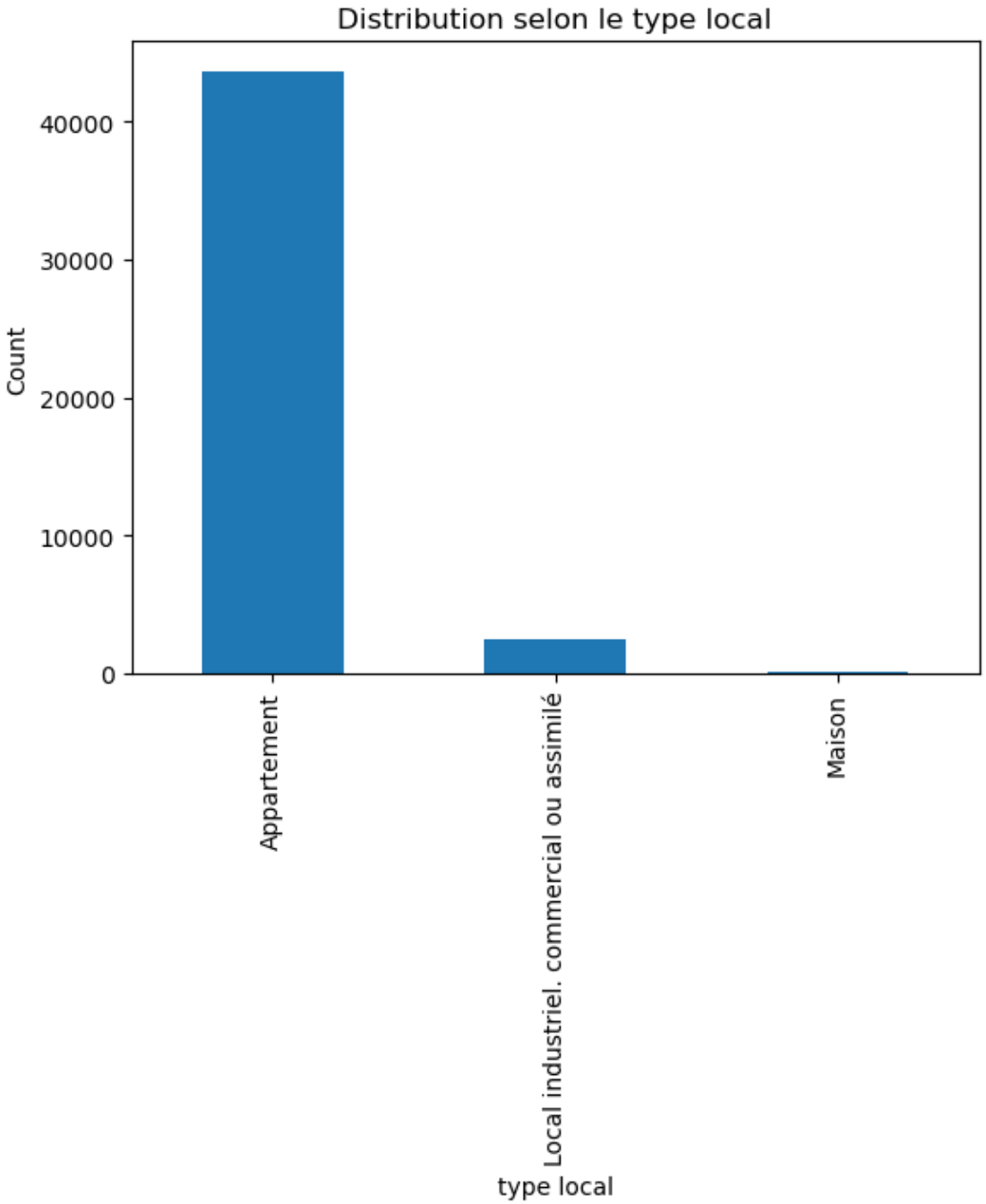
[42]:
```
dfDep["nature_mutation"].value_counts().sort_index().plot(kind="bar")
plt.title("Distribution selon les natures de mutation")
plt.xlabel("nature mutation")
plt.ylabel("Count")
plt.show()
```



La plupart des biens sont des ventes. On va pouvoir supprimer cette variable

[43]:
```
dfDep.drop(['nature_mutation'], inplace=True, axis=1)
```

[44]:
```
dfDep["type_local"].value_counts().sort_index().plot(kind="bar")
plt.title("Distribution selon le type local")
plt.xlabel("type local")
plt.ylabel("Count")
plt.show()
```



On peut se poser la question de la pertinence de cette variable car on a essentiellement deux modalités qui jouent

[45]:
```
dfDep["nombre_pieces_principales"].value_counts()
```

[45]:
```
2.0     16205
1.0     12097
3.0     10428
4.0      3958
0.0      2512
5.0       846
6.0        89
7.0        13
8.0         6
9.0         1
10.0        1
13.0        1
15.0        1
17.0        1
11.0        0
Name: nombre_pieces_principales, dtype: int64
```

Ce champs apparait mal instancié avec des valeurs abberantes

[46]:
```
dfDep.drop(['nombre_pieces_principales'], inplace=True, axis=1)
```

[47]:
```
dfDep["nom_commune"].value_counts()
```

[47]:
```
Paris 18e Arrondissement    5247
Paris 15e Arrondissement    5173
Paris 11e Arrondissement    4162
Paris 17e Arrondissement    3960
Paris 20e Arrondissement    3318
Paris 16e Arrondissement    3209
Paris 19e Arrondissement    2845
Paris 12e Arrondissement    2785
Paris 10e Arrondissement    2691
Paris 14e Arrondissement    2493
Paris 13e Arrondissement    2410
Paris 9e Arrondissement     1811
Paris 5e Arrondissement     1181
Paris 3e Arrondissement      956
Paris 7e Arrondissement      882
Paris 6e Arrondissement      798
Paris 8e Arrondissement      743
Paris 2e Arrondissement      740
Paris 4e Arrondissement      676
```

```
Paris 1er Arrondissement     402
Name: nom_commune, dtype: int64
```

4.1.5 Conclusion sur l'analyse univariée

On a pu voir que : * le jeu de données est constituée d'un an et demi d'historique * la target a été retravaillé pour éliminer les valeurs aberrantes. Il reste néanmoins des points atypiques sur le boxplot * on a rationnalisé certaines variables en enlevant des valeurs aberrantes ou en les éliminant de l'analyse pour certaines varaibles catégorielles. * on a introduit du feature ingeenering sur les dates

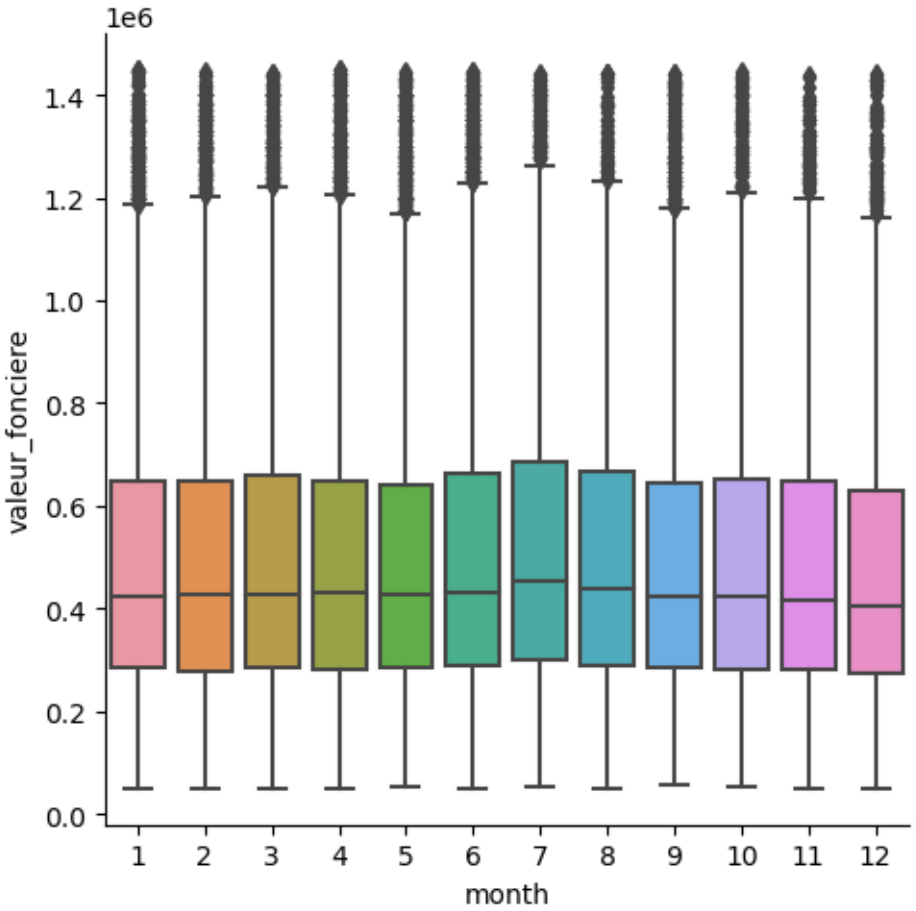
4.2 Analyse bivariée

L'analyse bivariée va consister à regarder l'influence de différentes variables sur la variable cible.

4.2.1 Les variables catégorielles

[48]:
```
sns.catplot(x="month", y="valeur_fonciere", kind="box", data=dfDep)
```

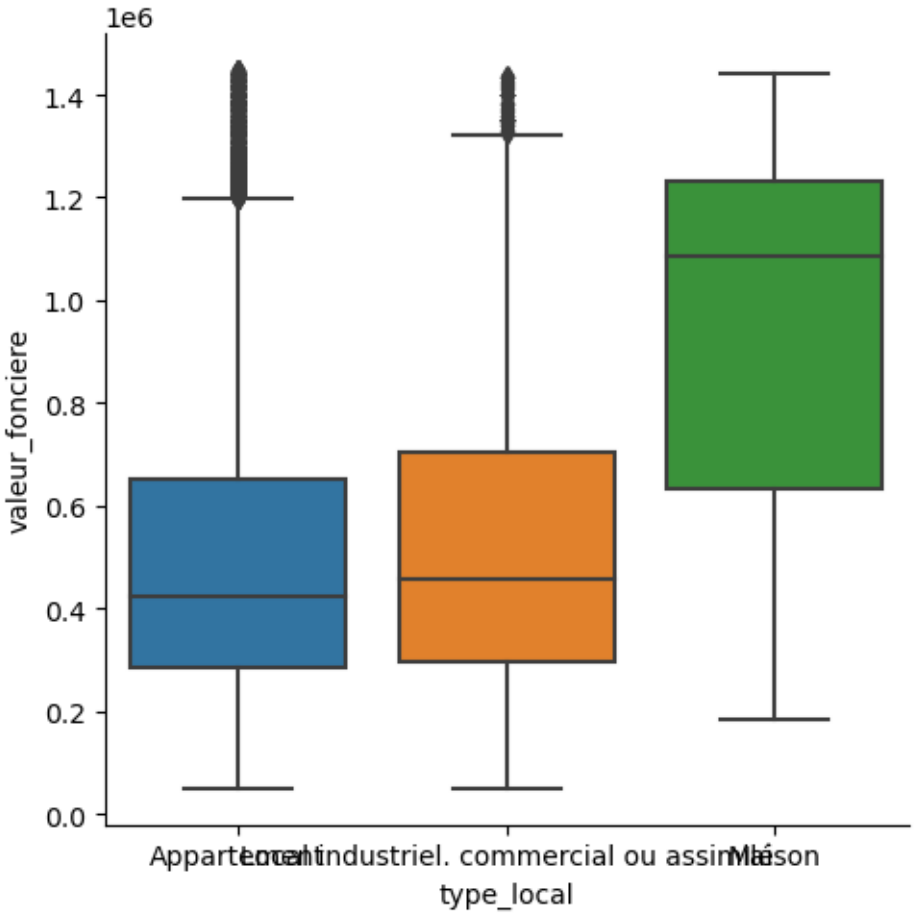
[48]:
```
<seaborn.axisgrid.FacetGrid at 0x16f219ac430>
```



Que cela soit en comparant month, year, day, les niveaux semblent relativement identiques mais on a beaucoup de points atypiques

[49]:
```
sns.catplot(x="type_local", y="valeur_fonciere", kind="box", data=dfDep,␣
↪orient="v")
```

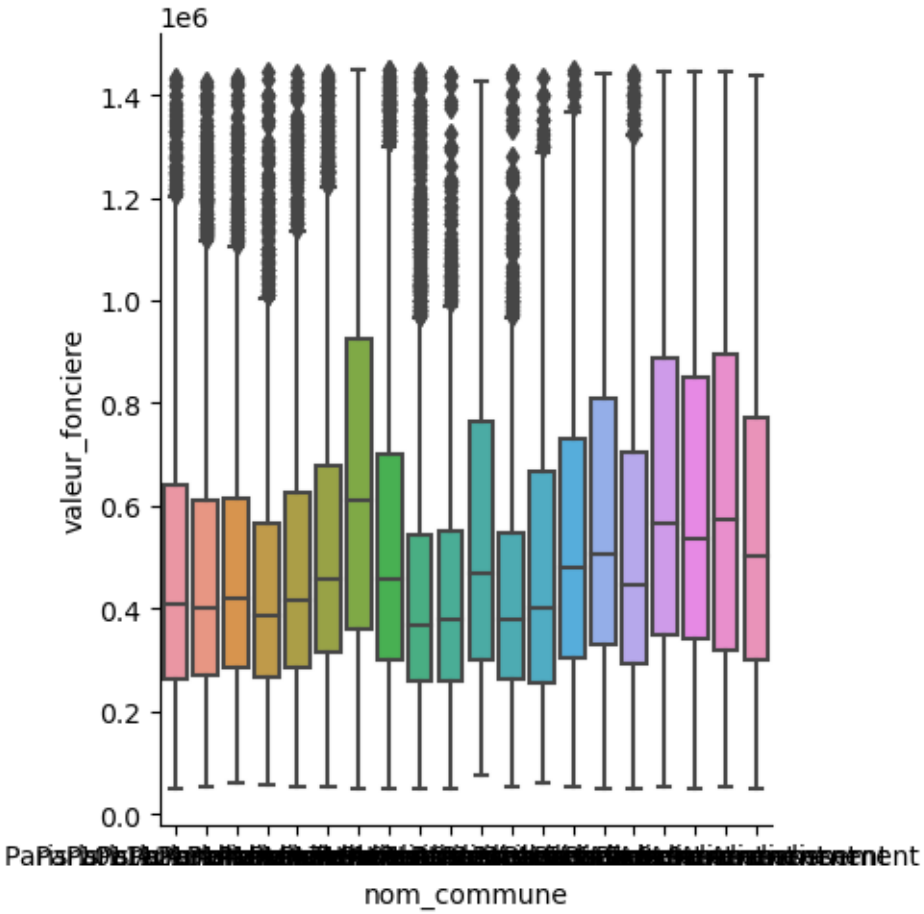
[49]: <seaborn.axisgrid.FacetGrid at 0x16f1f866ca0>



On peut constater que le prix des maisons est plus élevé que locaux industriels qui sont eux-mêmes à un niveau équivalent par rapport aux appartements en moyenne

[50]:
```
sns.catplot(x="nom_commune", y="valeur_fonciere", kind="box",␣
↪orient="v",data=dfDep)
```

[50]: <seaborn.axisgrid.FacetGrid at 0x16f234aa040>



Les arrondissements semblent avoir un effet sur les prix

4.2.2 Les variables quantitatives

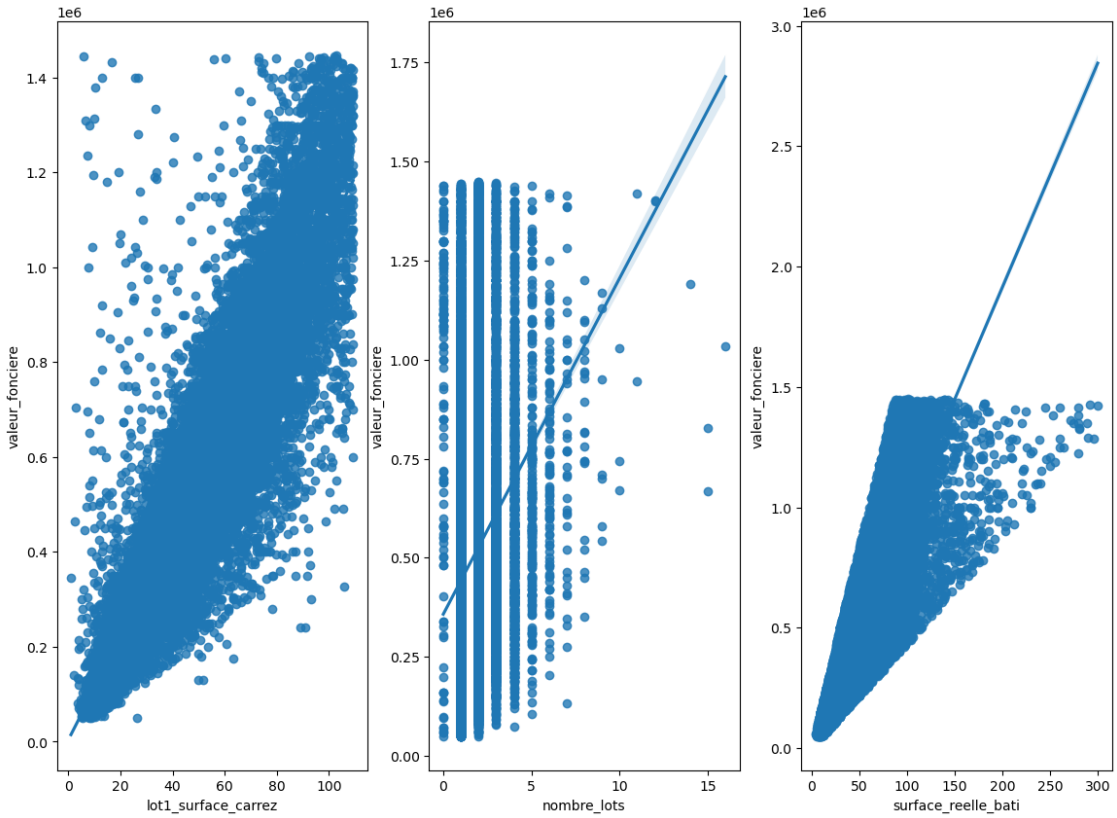
[51]:
```
dfDep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46482 entries, 1379722 to 4375222
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id_mutation          46482 non-null  string
 1   date_mutation        46482 non-null  datetime64[ns]
 2   numero_disposition   46482 non-null  int64
 3   valeur_fonciere      46482 non-null  float64
 4   adresse_numero       46481 non-null  string
 5   adresse_nom_voie     46481 non-null  string
 6   nom_commune          46482 non-null  category
 7   lot1_surface_carrez  28338 non-null  float64
```

```
 8   nombre_lots           46482 non-null  int64
 9   type_local            46159 non-null  category
 10  surface_reelle_bati   46149 non-null  float64
 11  longitude             46482 non-null  float64
 12  latitude              46482 non-null  float64
 13  adresse_complete      46481 non-null  string
 14  month                 46482 non-null  category
 15  day                   46482 non-null  category
 16  year                  46482 non-null  category
dtypes: category(5), datetime64[ns](1), float64(5), int64(2), string(4)
memory usage: 4.8 MB
```

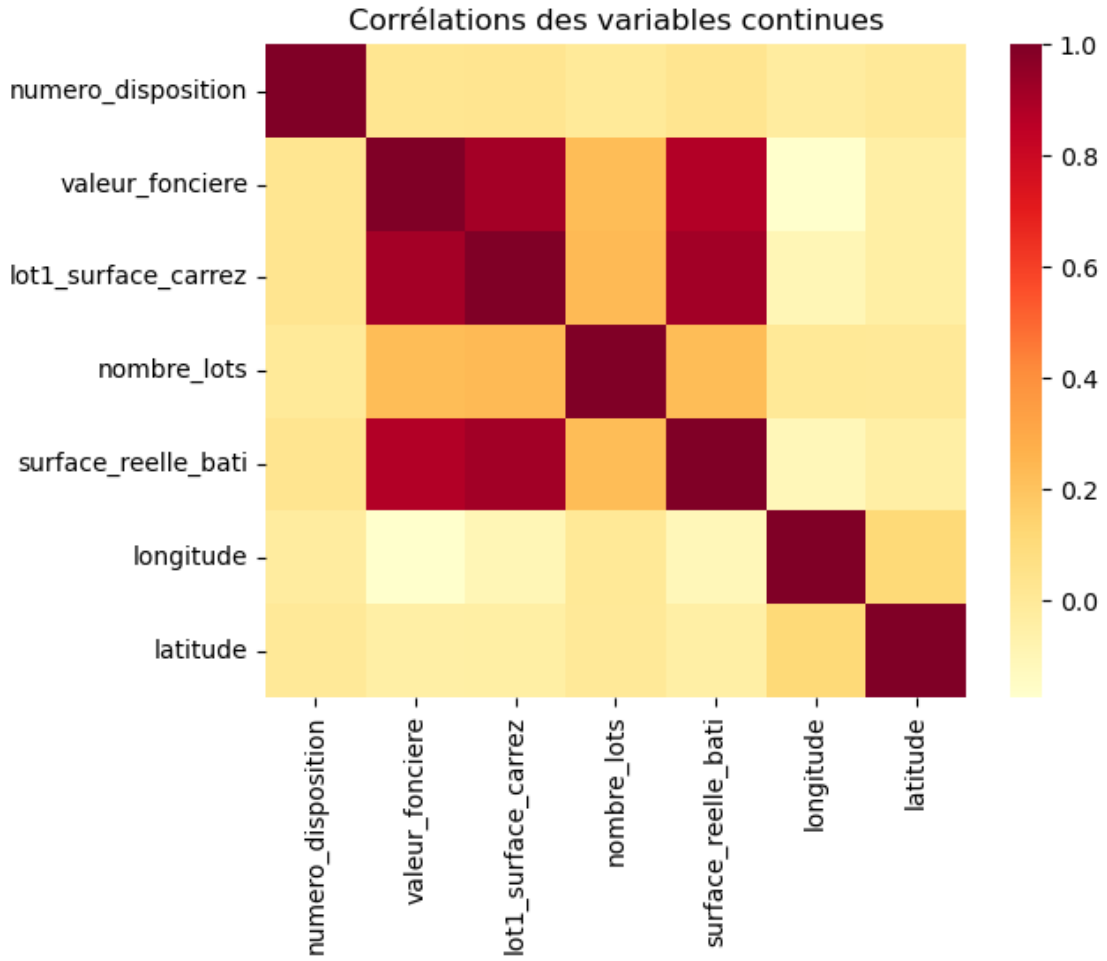
[52]:
```
fig,(ax1,ax2,ax3) = plt.subplots(ncols=3)
fig.set_size_inches(14,10)
sns.regplot(x="lot1_surface_carrez",y="valeur_fonciere",data=dfDep, ax=ax1)
sns.regplot(x="nombre_lots",y="valeur_fonciere",data=dfDep,ax=ax2)
sns.regplot(x="surface_reelle_bati",y="valeur_fonciere",data=dfDep,ax=ax3)
#sns.regplot(x="Prix m2",y="valeur_fonciere",data=dfDep,ax=ax4)
```

[52]:
```
<AxesSubplot:xlabel='surface_reelle_bati', ylabel='valeur_fonciere'>
```



La valeur foncière augmente avec la surface, le nombre de lots, la surface réelle.

[53]:
```
sns.heatmap(dfDep.corr(), cmap="YlOrRd")
plt.title("Corrélations des variables continues")
plt.show()
dfDep.corr()
```



[53]:

	numero_disposition	valeur_fonciere	lot1_surface_carrez	\
numero_disposition	1.000000	0.024541	0.031832	
valeur_fonciere	0.024541	1.000000	0.909703	
lot1_surface_carrez	0.031832	0.909703	1.000000	
nombre_lots	-0.000242	0.227670	0.235166	
surface_reelle_bati	0.031681	0.876837	0.915587	
longitude	-0.017176	-0.172863	-0.098811	
latitude	0.001994	-0.042408	-0.036007	

	nombre_lots	surface_reelle_bati	longitude	latitude
numero_disposition	-0.000242	0.031681	-0.017176	0.001994
valeur_fonciere	0.227670	0.876837	-0.172863	-0.042408

lot1_surface_carrez	0.235166	0.915587	-0.098811	-0.036007
nombre_lots	1.000000	0.227450	0.009932	0.003212
surface_reelle_bati	0.227450	1.000000	-0.106381	-0.041606
longitude	0.009932	-0.106381	1.000000	0.113772
latitude	0.003212	-0.041606	0.113772	1.000000

La valeur foncière est fortement corrélée à la surface carrez ou réelle, faiblement au nombre de lots et aux coordonnées gps

4.2.3 Conclusion sur l'analyse bivariée

La target est sensible : * à la surface réelle ou carrez * au prix du m2 * à la commune Par contre, elle ne semble pas tellement sensible au mois, jour, année. La difficulté vient surtout du nombre de points atypiques importants.

4.3 Analyse multivariée

[54]:
```
#sns.relplot(x="lot1_surface_carrez", y="surface_reelle_bati",␣
↪size="valeur_fonciere", sizes=(15, 100), data=dfDep);
#sns.catplot(x="lot1_surface_carrez", y="valeur_fonciere",␣
↪hue="type_local",kind="bar",data=dfDep);
```

5 Modelisation

5.1 Preprocessing pour scikit-learn¶

[55]:
```
dfDep.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 46482 entries, 1379722 to 4375222
Data columns (total 17 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   id_mutation          46482 non-null  string
 1   date_mutation        46482 non-null  datetime64[ns]
 2   numero_disposition   46482 non-null  int64
 3   valeur_fonciere      46482 non-null  float64
 4   adresse_numero       46481 non-null  string
 5   adresse_nom_voie     46481 non-null  string
 6   nom_commune          46482 non-null  category
 7   lot1_surface_carrez  28338 non-null  float64
 8   nombre_lots          46482 non-null  int64
 9   type_local           46159 non-null  category
 10  surface_reelle_bati  46149 non-null  float64
 11  longitude            46482 non-null  float64
 12  latitude             46482 non-null  float64
 13  adresse_complete     46481 non-null  string
 14  month                46482 non-null  category
```

```
 15  day                 46482 non-null  category
 16  year                46482 non-null  category
dtypes: category(5), datetime64[ns](1), float64(5), int64(2), string(4)
memory usage: 4.8 MB
```

5.1.1 Gestion des données manquantes

Les méthodes numériques d'apprentissage ne gèrent pas les NaN ou null sur les valeurs numériques

[56]:
```
dfDep.isna().sum()
```

[56]:
```
id_mutation               0
date_mutation             0
numero_disposition        0
valeur_fonciere           0
adresse_numero            1
adresse_nom_voie          1
nom_commune               0
lot1_surface_carrez    18144
nombre_lots               0
type_local              323
surface_reelle_bati     333
longitude                 0
latitude                  0
adresse_complete          1
month                     0
day                       0
year                      0
dtype: int64
```

[57]:
```
dfDep.drop(['lot1_surface_carrez'],inplace=True, axis=1)
dfDep.drop(dfDep[dfDep['surface_reelle_bati'].isna()].index, inplace=True,␣
↪axis=0)
```

[58]:
```
dfDep.isna().sum()
```

[58]:
```
id_mutation           0
date_mutation         0
numero_disposition    0
valeur_fonciere       0
adresse_numero        0
adresse_nom_voie      0
nom_commune           0
nombre_lots           0
type_local            0
surface_reelle_bati   0
longitude             0
latitude              0
```

```
adresse_complete       0
month                  0
day                    0
year                   0
dtype: int64
```

5.1.2 Construction des ensembles X et y à partir du dataframe

On peut enlever l'id qui est un champ purement technique ainsi que la date et tous les éléments de type string

[59]:
```
dfDep["id_mutation"].drop_duplicates(inplace=True)
print(dfDep.shape)
```

```
(46149, 16)
```

[60]:
```
X = dfDep.drop(["valeur_fonciere","id_mutation", "date_mutation",␣
↪"numero_disposition", "adresse_numero",␣
↪"adresse_nom_voie","adresse_complete"], axis = 1)
y = dfDep["valeur_fonciere"]
print(f"Shape de X : {X.shape}")
print(f"Shape de y : {y.shape}")
X.head(5)
```

```
Shape de X : (46149, 9)
Shape de y : (46149,)
```

[60]:

	nom_commune	nombre_lots	type_local	surface_reelle_bati	longitude	latitude	month	day	year
1379722	Paris 18e Arrondissement	2	Appartement	45.0	2.348168	48.884490	1	4	2022
1379725	Paris 3e Arrondissement	1	Appartement	42.0	2.362871	48.863374	1	6	2022
1379732	Paris 9e Arrondissement	3	Appartement	69.0	2.332324	48.880353	1	5	2022
1379734	Paris 10e Arrondissement	2	Appartement	33.0	2.362613	48.879658	1	5	2022
1379736	Paris 20e Arrondissement	2	Appartement	29.0	2.405513	48.872782	1	7	2022

5.1.3 Preprocessing sur les variables catégorielles

[61]:
```
categorical_features = X.columns[X.dtypes == "category"].tolist()
print(categorical_features)
```

```
['nom_commune', 'type_local', 'month', 'day', 'year']
```

Scikit-learn ne reconnait pas les objets de type DataFrame directement, notamment les types catégoriels. Il faut donc préparer nos données afin que les méthodes de scikit-learn puissent les inter-

préter. Scikit learn requiert un encodage numérique des ces variables. Nous allons donc devoir encoder nos variables explicatives catégorielles à l'aide de variables indicatrices.

[62]:
```
df_dummies =  pd.get_dummies(X[categorical_features], drop_first=True)
X = pd.concat([X.drop(categorical_features, axis=1), df_dummies], axis=1)
X.head(5)
```

[62]:

	nombre_lots	surface_reelle_bati	longitude	latitude	\
1379722	2	45.0	2.348168	48.884490	
1379725	1	42.0	2.362871	48.863374	
1379732	3	69.0	2.332324	48.880353	
1379734	2	33.0	2.362613	48.879658	
1379736	2	29.0	2.405513	48.872782	

	nom_commune_Paris 11e Arrondissement	\
1379722	0	
1379725	0	
1379732	0	
1379734	0	
1379736	0	

	nom_commune_Paris 12e Arrondissement	\
1379722	0	
1379725	0	
1379732	0	
1379734	0	
1379736	0	

	nom_commune_Paris 13e Arrondissement	\
1379722	0	
1379725	0	
1379732	0	
1379734	0	
1379736	0	

	nom_commune_Paris 14e Arrondissement	\
1379722	0	
1379725	0	
1379732	0	
1379734	0	
1379736	0	

	nom_commune_Paris 15e Arrondissement	\
1379722	0	
1379725	0	
1379732	0	
1379734	0	

```
1379736                                    0

         nom_commune_Paris 16e Arrondissement  ...  day_23  day_24  day_25  \
1379722                                    0  ...       0       0       0
1379725                                    0  ...       0       0       0
1379732                                    0  ...       0       0       0
1379734                                    0  ...       0       0       0
1379736                                    0  ...       0       0       0

         day_26  day_27  day_28  day_29  day_30  day_31  year_2022
1379722       0       0       0       0       0       0          1
1379725       0       0       0       0       0       0          1
1379732       0       0       0       0       0       0          1
1379734       0       0       0       0       0       0          1
1379736       0       0       0       0       0       0          1

[5 rows x 67 columns]
```

5.2 Train, Test

Nous utilisons scikit-learn pour faire le traitement et étant donné la volumétrie du jeu de données, nous allons prendre 80% pour le train et 20% pour le test

[63]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=777)
print(f"Shape du X_train : {X_train.shape}")
print(f"Shape du y_train : {y_train.shape}")
print(f"Shape du X_test : {X_test.shape}")
print(f"Shape du y_test : {y_test.shape}")
```

```
Shape du X_train : (36919, 67)
Shape du y_train : (36919,)
Shape du X_test : (9230, 67)
Shape du y_test : (9230,)
```

5.3 Preprocessing sur les variables numériques

[64]:
```
numerical_features = dfDep.columns[(dfDep.dtypes == "int64")].tolist() + dfDep.
 ↪columns[(dfDep.dtypes == "float64")].tolist()
print(numerical_features)
```

```
['numero_disposition', 'nombre_lots', 'valeur_fonciere', 'surface_reelle_bati',
'longitude', 'latitude']
```

Certaines méthodes d'apprentissage sont sensibles aux problèmes d'échelle sur les valeurs numériques. En preprocessing, on standardise les variables numériques en retranchant leur moyenne et en divisant par l'écart type via Scikit-learn. On réalise ce traitement sur l'ensemble d'apprentissage et on applique cette standardisation sur l'ensemble de test.

[65]:
```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

5.4 Un modèle simple : la régression linéaire

Un premier modèle qui nous servira de *baseline*.

Nous allons aussi introduire l'instanciation sur les données *train*, et nous validerons **ENSUITE** sur les données *test*.

5.4.1 Modèle de regression sur Train/Test

$$y = \sum_{i=1}^{n} a_i \times x_i + b$$

[66]:
```
from sklearn import linear_model
reg = linear_model.LinearRegression()

reg.fit(X_train_scaled, y_train)
y_trainPred = reg.predict(X_train_scaled)
y_testPred = reg.predict(X_test_scaled)
print(f"Score sur le train : {reg.score(X_train_scaled,y_train)}")
print(f"Score sur le test : {reg.score(X_test_scaled,y_test)}")
```

```
Score sur le train : 0.8017175728977979
Score sur le test : 0.8011631121444063
```

La régression linéaire donne des résultats et il n'y a pas de phénomène de sur-apprentissage.

5.5 Coefficients de la régression linéaire

Un des avantages de la régression linéaire est que nous pouvons obtenir les coefficients associés à chacune des variables. Nous pouvons voir les coefficients qui ont un impact sur le nombre de vélos loués.

Regardons ces coefficients :

[67]:
```
coefficients = pd.Series(reg.coef_.flatten(), index=X.columns).
 ↪sort_values(ascending=False)
coefficients
```

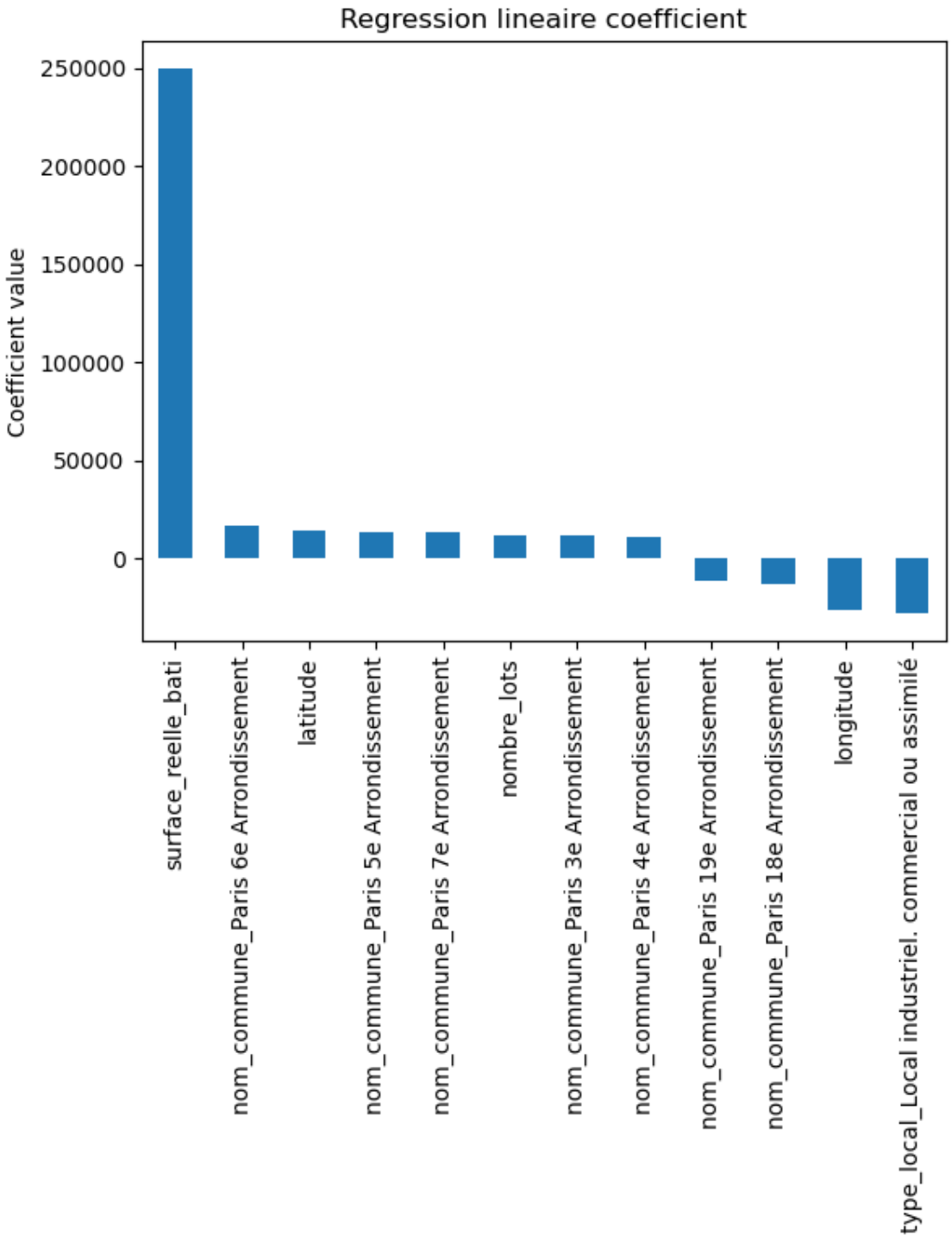
[67]:
```
surface_reelle_bati                          249572.870033
nom_commune_Paris 6e Arrondissement           16466.302201
latitude                                      14105.244694
nom_commune_Paris 5e Arrondissement           13676.283050
nom_commune_Paris 7e Arrondissement           13343.601169
                                                  ...
```

```
nom_commune_Paris 15e Arrondissement                    -5217.255155
nom_commune_Paris 19e Arrondissement                   -11347.416919
nom_commune_Paris 18e Arrondissement                   -12725.454998
longitude                                              -26494.427558
type_local_Local industriel. commercial ou assimilé    -27947.820593
Length: 67, dtype: float64
```

[68]:
```
print(f"ordonnee à l'origine : {reg.intercept_}")
```

```
ordonnee à l'origine : 499534.7393496575
```

[69]:
```
coefficients[np.abs(coefficients)>10000].plot(kind="bar")
plt.title("Regression lineaire coefficient")
plt.ylabel("Coefficient value")
plt.show()
```



On retrouve des éléments de l'exploration. Certaines communes tirent le prix vers le base comme le 18ème, 19ème au contraire du 6ème, ... L'élément le plus prépondérant est la surface réellement bati. La lattitude et la longitude s'opposent en termes d'effet.

5.5.1 Evaluation de la régression avec différentes métriques

Nous allons regarder quelques métriques associées aux problématiques de régression : * L'erreur maximum entre la prédiction et la réalité * La moyenne des erreurs absolus entre la prédiction et la réalité * La moyenne des erreurs au carré entre la prédiction et la réalité (MSE) * Le score R2 qui est le coefficient de détermination en comparant MSE et la variance. Fonction renvoyée par la méthode score de Scikit Learn

[70]:
```
from sklearn import metrics


def regression_metrics(y, y_pred):
    return pd.DataFrame(
        {
            "max_error": metrics.max_error(y_true=y, y_pred=y_pred),
            "mean_absolute_error": metrics.mean_absolute_error(y_true=y,␣
 ↪y_pred=y_pred),
            "mean_squared_error": metrics.mean_squared_error(y_true=y,␣
 ↪y_pred=y_pred),
            "r2_score": metrics.r2_score(y_true=y, y_pred=y_pred)
        },
        index=[0])
```

[71]:
```
print("Regression metrics for train data")
print(regression_metrics(y_train, y_trainPred))
print("Regression metrics for test data")
print(regression_metrics(y_test, y_testPred))
```

```
Regression metrics for train data
      max_error  mean_absolute_error  mean_squared_error  r2_score
0  1.596970e+06         84687.636312        1.653548e+10  0.801718
Regression metrics for test data
      max_error  mean_absolute_error  mean_squared_error  r2_score
0  1.321491e+06         85090.635792        1.620885e+10  0.801163
```

Le modèle de regression linéaire n'est pas très bon quelque soit la métrique retenue.

5.6 Arbre de décision et visions ensemblistes

5.6.1 Arbre de décision

[72]:
```
from sklearn.tree import DecisionTreeRegressor
decisionTree = DecisionTreeRegressor()
decisionTree.fit(X_train_scaled, y_train)
y_trainPred = decisionTree.predict(X_train_scaled)
y_testPred = decisionTree.predict(X_test_scaled)
print(f"Score sur le train de l'arbre de décision : {decisionTree.
 ↪score(X_train_scaled,y_train)}")
```

```
print(f"Score sur le test de l'arbre de décision : {decisionTree.
 ↪score(X_test_scaled,y_test)}")
```

```
Score sur le train de l'arbre de décision : 0.9998805881649797
Score sur le test de l'arbre de décision : 0.7432610064158165
```

[73]:
```
print("Regression metrics with Decision Tree for train data")
print(regression_metrics(y_train, y_trainPred))
print("Regression metrics with Decision Tree for test data")
print(regression_metrics(y_test, y_testPred))
```

```
Regression metrics with Decision Tree for train data
   max_error  mean_absolute_error  mean_squared_error  r2_score
0   300000.0            64.969443        9.958180e+06  0.999881
Regression metrics with Decision Tree for test data
   max_error  mean_absolute_error  mean_squared_error  r2_score
0   946000.0         97475.687469        2.092894e+10  0.743261
```

On est dans un cas de surapprentissage puisque l'arbre de décision "fit" à l'ensemble de train mais ne se généralise pas bien sur l'ensemble de test. Néanmoins la performance est moins bonne que la régression linéaire

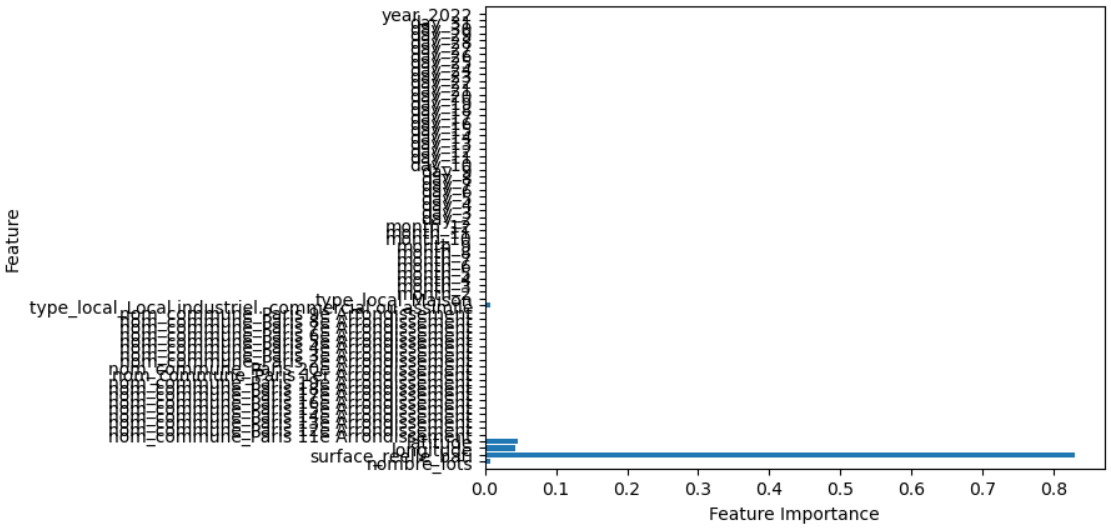
[74]:
```
print("Feature importances : \n{}".format(decisionTree.feature_importances_))
```

```
Feature importances :
[7.24203099e-03 8.29786055e-01 4.18925665e-02 4.66348159e-02
 6.67695212e-04 1.18150928e-04 7.76163497e-04 4.08262587e-04
 7.84889388e-04 7.01459265e-04 3.81124735e-04 2.63807564e-04
 7.69301917e-05 4.66695105e-04 8.91297788e-04 2.30626244e-04
 2.90281190e-04 4.79320086e-04 8.02850151e-04 1.13558807e-03
 8.98383096e-04 5.95663078e-04 3.73737532e-04 6.87450761e-03
 8.14283687e-04 1.81425646e-03 1.50775843e-03 1.97679818e-03
 1.64040197e-03 2.27642287e-03 1.86109849e-03 7.31854835e-04
 1.45719587e-03 1.88363398e-03 1.12424481e-03 1.54502644e-03
 1.16679892e-03 9.22287403e-04 8.28115801e-04 1.32674090e-03
 1.16443121e-03 1.12699776e-03 1.36499948e-03 1.37794852e-03
 1.42391302e-03 1.10168192e-03 8.97735692e-04 1.07600888e-03
 1.22417400e-03 1.40406454e-03 1.57954947e-03 1.38401471e-03
 1.04567914e-03 1.10180426e-03 1.35720554e-03 9.28337909e-04
 1.38411118e-03 1.30660494e-03 1.13217292e-03 9.89857433e-04
 1.01727895e-03 1.20337627e-03 1.38172305e-03 1.43790008e-03
 8.86376639e-04 1.46483530e-03 2.58739624e-03]
```

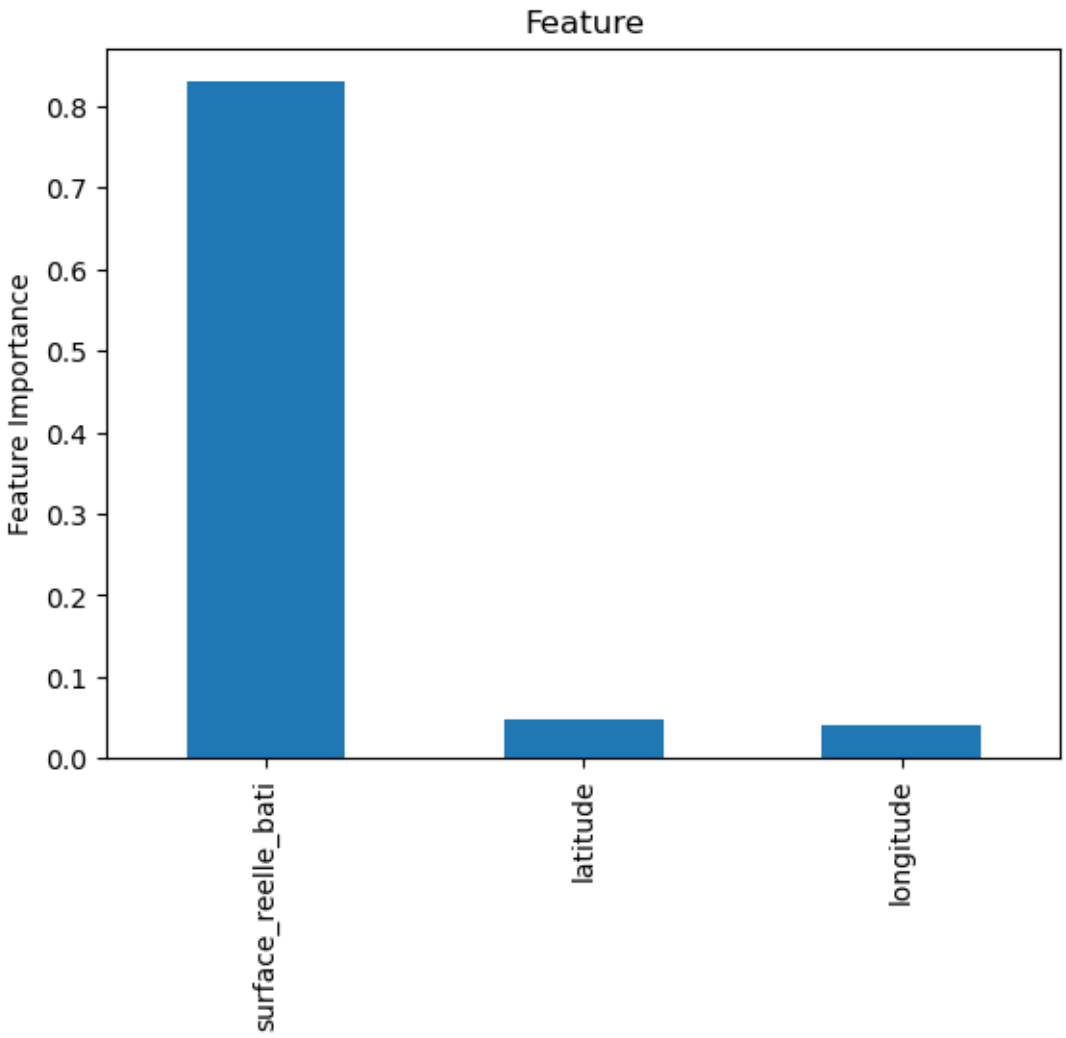
[75]:
```
def plot_feature_importances(model):
    n_features = X.shape[1]
    plt.barh(range(n_features), model.feature_importances_, align = 'center')
    plt.yticks(np.arange(n_features), X.columns)
    plt.xlabel("Feature Importance")
    plt.ylabel("Feature")
```

```
    plt.ylim(-1,n_features)
```

[76]:
```
plot_feature_importances(decisionTree)
```



[77]:
```
featuresImportance = pd.Series(decisionTree.feature_importances_.flatten(),␣
↪index=X.columns).sort_values(ascending=False)
featuresImportance[(featuresImportance)>0.03].plot(kind="bar")
plt.title("Feature")
plt.ylabel("Feature Importance")
plt.show()
```



On retrouve la surface réelle et la position géographique du bien.

[78]:
```
for depth in range(5,20):
    decisionTreeMaxDepth = DecisionTreeRegressor(max_depth=depth)
    decisionTreeMaxDepth.fit(X_train_scaled, y_train)
    print(f"Max depth : {depth}")
    print(f"Score sur le train de l'arbre de décision : {decisionTreeMaxDepth.
↪score(X_train_scaled,y_train)}")
    print(f"Score sur le test de l'arbre de décision : {decisionTreeMaxDepth.
↪score(X_test_scaled,y_test)}")
```

```
Max depth : 5
Score sur le train de l'arbre de décision : 0.8287761133628597
Score sur le test de l'arbre de décision : 0.8216097745011889
Max depth : 6
Score sur le train de l'arbre de décision : 0.8375911045161
```

```
Score sur le test de l'arbre de décision : 0.8298781113397137
Max depth : 7
Score sur le train de l'arbre de décision : 0.8475715134258025
Score sur le test de l'arbre de décision : 0.8345602372823613
Max depth : 8
Score sur le train de l'arbre de décision : 0.8571520065388156
Score sur le test de l'arbre de décision : 0.839275099223095
Max depth : 9
Score sur le train de l'arbre de décision : 0.8667514157812605
Score sur le test de l'arbre de décision : 0.8379927811502408
Max depth : 10
Score sur le train de l'arbre de décision : 0.8771788808029217
Score sur le test de l'arbre de décision : 0.8378138986698369
Max depth : 11
Score sur le train de l'arbre de décision : 0.8884427238026404
Score sur le test de l'arbre de décision : 0.8332184373980339
Max depth : 12
Score sur le train de l'arbre de décision : 0.9002835113213346
Score sur le test de l'arbre de décision : 0.827860871670689
Max depth : 13
Score sur le train de l'arbre de décision : 0.9126784232754481
Score sur le test de l'arbre de décision : 0.8184371899148659
Max depth : 14
Score sur le train de l'arbre de décision : 0.9248986124917595
Score sur le test de l'arbre de décision : 0.8073907282473716
Max depth : 15
Score sur le train de l'arbre de décision : 0.9360622025943398
Score sur le test de l'arbre de décision : 0.8011019844075465
Max depth : 16
Score sur le train de l'arbre de décision : 0.9465575116161347
Score sur le test de l'arbre de décision : 0.7943883184687182
Max depth : 17
Score sur le train de l'arbre de décision : 0.9559007041620089
Score sur le test de l'arbre de décision : 0.7843571997819065
Max depth : 18
Score sur le train de l'arbre de décision : 0.9638673313927707
Score sur le test de l'arbre de décision : 0.7741319430495943
Max depth : 19
Score sur le train de l'arbre de décision : 0.9706319172890191
Score sur le test de l'arbre de décision : 0.7741276562077122
```

On observe assez vite le surapprentissage lorsqu'on augmente la profondeur de l'arbre

Avantages : * On peut contrôler la complexité de l'arbre en jouant sur des paramètres avec la profondeur ou des stratégies d'élagage * Interprétabilité des décisions * Pas de problématique de prise en compte des échelles différentes entre les variables (même si dans notre cas, nous travaillons sur des données standardisées)

Inconvénient majeur : * Même en jouant sur la complexité de l'arbre, un arbre tend au surappren-

tissage et fournit de piètre performance de généralisation

5.6.2 Random Forest

[79]:
```
from sklearn.ensemble import RandomForestRegressor
nbTree = 100
print(f"Nombre d'arbres considérés : {nbTree}")
for depth in [5,10,15,20,30, 40]:
    randomForest = RandomForestRegressor(n_estimators=nbTree, random_state=2,␣
 ↪max_depth=depth)
    randomForest.fit(X_train_scaled, y_train)
    print(f"--- Max depth : {depth}")
    print(f"---------Score sur le train avec RandomForest : {randomForest.
 ↪score(X_train_scaled,y_train)}")
    print(f"---------Score sur le test avec RandomForest : {randomForest.
 ↪score(X_test_scaled,y_test)}")
```

```
Nombre d'arbres considérés : 100
--- Max depth : 5
---------Score sur le train avec RandomForest : 0.8359706488563999
---------Score sur le test avec RandomForest : 0.8289525406733299
--- Max depth : 10
---------Score sur le train avec RandomForest : 0.8896458202070148
---------Score sur le test avec RandomForest : 0.8595626102602516
--- Max depth : 15
---------Score sur le train avec RandomForest : 0.9407613378235083
---------Score sur le test avec RandomForest : 0.86188618552186
--- Max depth : 20
---------Score sur le train avec RandomForest : 0.9681499184443108
---------Score sur le test avec RandomForest : 0.8596451915853982
--- Max depth : 30
---------Score sur le train avec RandomForest : 0.979733060427418
---------Score sur le test avec RandomForest : 0.858242085415685
--- Max depth : 40
---------Score sur le train avec RandomForest : 0.9803220072381146
---------Score sur le test avec RandomForest : 0.8581958142919872
```

On observe avec Random Forest une amélioration du score fonction de la profondeur considérées avec un surapprentissage de plus en plus important.

5.6.3 GridSearch et Validation croisée

Nous allons creuser un peu plus loin afin d'améliorer RandomForest en optimisant les hyper-paramètres du modèle. Pour ce faire nous allons procéder par validation croisée avec 5 plis sur l'ensemble d'apprentissage. A l'aide de celle-ci, nous allons chercher quel(s) paramètre(s) nous donne(nt) le meilleur score et enfin nous évaluerons la qualité du modèle sur le jeu de données test.

Les paramètres que nous allons chercher à optimiser dans RandomForest sont : * le paramètre max_depth qui correspond à la profondeur de l'arbre * le nombre d'arbres à considérer dans la

forêt * le nombre de features maximale à considérer

[80]:
```
from sklearn.model_selection import GridSearchCV
# grille de valeurs
params = [{"max_depth": [10,15,20], "n_estimators": [100,200,300,500],␣
 ↪"max_features": [12, 15, 20, 25]}]

gridSearchCV = GridSearchCV(
    RandomForestRegressor(),
    params,
    cv=5,
    n_jobs=-1,
    return_train_score=True)
gridSearchCV.fit(X_train_scaled, y_train)
```

[80]:
```
GridSearchCV(cv=5, estimator=RandomForestRegressor(), n_jobs=-1,
             param_grid=[{'max_depth': [10, 15, 20],
                          'max_features': [12, 15, 20, 25],
                          'n_estimators': [100, 200, 300, 500]}],
             return_train_score=True)
```

[81]:
```
print("Score sur le test : {:.2f}".format(gridSearchCV.
 ↪score(X_test_scaled,y_test)))
```

```
Score sur le test : 0.86
```

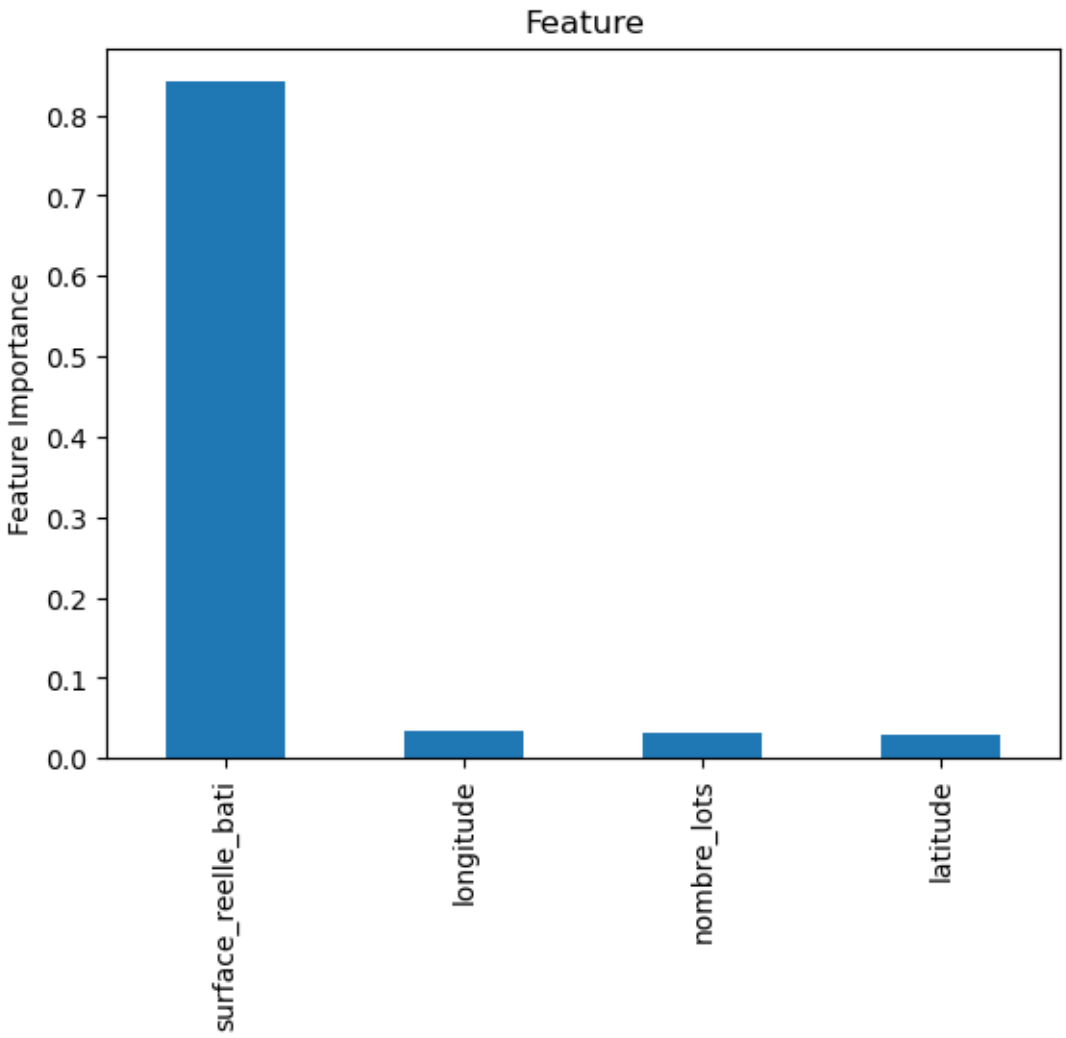
[82]:
```
print("Best parameters : {}".format(gridSearchCV.best_params_))
print("Best cross-validation score : {:.2f}".format(gridSearchCV.best_score_))
```

```
Best parameters : {'max_depth': 15, 'max_features': 25, 'n_estimators': 300}
Best cross-validation score : 0.86
```

[83]:
```
print("Best estimator:\n{}".format(gridSearchCV.best_estimator_))
```

```
Best estimator:
RandomForestRegressor(max_depth=15, max_features=25, n_estimators=300)
```

[84]:
```
featuresImportance = pd.Series(gridSearchCV.best_estimator_.feature_importances_.
 ↪flatten(), index=X.columns).sort_values(ascending=False)
featuresImportance[(featuresImportance)>0.03].plot(kind="bar")
plt.title("Feature")
plt.ylabel("Feature Importance")
plt.show()
```



A travers une validation croisée et un grid search, on obtient un paramétrage via Random Forest et on peut visualiser les variables qui ont de l'importance. On retrouve des variables explicatives en lien avec notre analyse exploratoire. On est aussi dans un cas où il n'y a pas de surapprentissage.

[85]:
```
y_testPred = gridSearchCV.best_estimator_.predict(X_test_scaled)
print("Regression metrics pour la forêt aléatoire optimisée for test data")
print(regression_metrics(y_test, y_testPred))
```

```
Regression metrics pour la forêt aléatoire optimisée for test data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  667391.637514         71635.656938        1.122499e+10  0.862301
```

[86]:
```
y_trainPred = gridSearchCV.best_estimator_.predict(X_train_scaled)
print("Regression metrics pour la forêt aléatoire optimisée for train data")
print(regression_metrics(y_train, y_trainPred))
```

```
Regression metrics pour la forêt aléatoire optimisée for train data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  535465.091525          54852.90272        6.189316e+09  0.925782
```

5.7 Sauvegarde du modèle

[87]:
```
#from joblib import dump, load
```

[88]:
```
#dump(gridSearchCV.best_estimator_.predict, 'sauvegardeModele.joblib')
```

[89]:
```
#clf = load('sauvegardeModele.joblib')
```

[90]:
```
#clf.predict(X_test_scaled)
```

[91]:
```
#dfDepIni.to_csv('../input/AvecCoordonneesGeo/dep75.csv')
```

6 Conclusion

6.1 Sur le travail réalisé

- L'analyse univariée et multivariée ont permis de mettre en évidence des liens entre les variables explicatives et à expliquer
- Le featuring Ingeenering a été un travail réalisé sur les dates pour essayer de voir les liens avec la variable à prédire.
- Les modèles linéaires donne des résultats pas très intéressants sur certaines métriques
- Un modèle basé sur des arbres de décision permet d'obtenir des meilleurs résultats par rapport à la regression linéaire. Une optimisation des paramètres a pu être mise en oeuvre via validation croisée et grille de recherche

6.2 Sur les perspectives

- Sur le code : la mise en place de Pipe avec l'utilisation de OneHotEncoder et StandardScaler.
- Sur les modèles : tester d'autres modèles pour améliorer la prévision. On peut penser à une régression polynomiale ou boosting d'arbres de régression, ou des modèles traitant spécifiquement de séries temporelles.
- Un traitement des points atypiques a été réalisé avec aussi de l'imputation mais il faudrait passer plus de temps sur la compréhension des données.