Project House Prices : Régression et Deep Learning

Anthony Moisan

03/01/2020

Contents

1 Description du projet

Le projet consiste à prévoir le prix de maisons en fonction d'un certain nombre de caractéristiques. Ce projet est issu d'un défi Kaggle que l'on peut retrouver ici.

L'objectif de ce projet sera : * de comprendre les variables explicatives et la variable cible en faisant une analyse exploratoire des données * de définir un premier modèle qui sera dans le cas présent une régression linéaire * de mettre en place un réseau de neurones * de faire une analyse comparative entre les deux modèles

2 Type de problème

On est typiquement dans une problématique de régression dans le cas d'un apprentissage supervisé.

3 Librairies

[0]:
```
import pandas as pd
import numpy as np

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

#Les librairies propres à Tensorflow ou Scikit-Learn seront importées au moment␣
 ↪de leur utilisation
```

4 Lecture du jeu de données

4.1 Taille du jeu de données

[2]:
```
df = pd.read_csv("https://raw.githubusercontent.com/anthonymoisan/
 ↪DeepLearningPredictHousePrices/master/input/train.csv")
print("taille du jeu de donnees :", df.shape)
df.head(10)
```

```
taille du jeu de donnees : (1460, 81)
```

[2]:

	Id	MSSubClass	MSZoning	...	SaleType	SaleCondition	SalePrice
0	1	60	RL	...	WD	Normal	208500
1	2	20	RL	...	WD	Normal	181500
2	3	60	RL	...	WD	Normal	223500
3	4	70	RL	...	WD	Abnorml	140000
4	5	60	RL	...	WD	Normal	250000
5	6	50	RL	...	WD	Normal	143000
6	7	20	RL	...	WD	Normal	307000

```
7   8          60       RL  ...        WD         Normal    200000
8   9          50       RM  ...        WD        Abnorml    129900
9  10         190       RL  ...        WD         Normal    118000

[10 rows x 81 columns]
```

On regarde les informations assez rapidement sur les variables

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null int64
MSZoning         1460 non-null object
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null object
Alley            91 non-null object
LotShape         1460 non-null object
LandContour      1460 non-null object
Utilities        1460 non-null object
LotConfig        1460 non-null object
LandSlope        1460 non-null object
Neighborhood     1460 non-null object
Condition1       1460 non-null object
Condition2       1460 non-null object
BldgType         1460 non-null object
HouseStyle       1460 non-null object
OverallQual      1460 non-null int64
OverallCond      1460 non-null int64
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null object
RoofMatl         1460 non-null object
Exterior1st      1460 non-null object
Exterior2nd      1460 non-null object
MasVnrType       1452 non-null object
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null object
ExterCond        1460 non-null object
Foundation       1460 non-null object
BsmtQual         1423 non-null object
BsmtCond         1423 non-null object
BsmtExposure     1422 non-null object
BsmtFinType1     1423 non-null object
BsmtFinSF1       1460 non-null int64
```

```
BsmtFinType2     1422 non-null object
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null object
HeatingQC        1460 non-null object
CentralAir       1460 non-null object
Electrical       1459 non-null object
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null object
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null object
Fireplaces       1460 non-null int64
FireplaceQu      770 non-null object
GarageType       1379 non-null object
GarageYrBlt      1379 non-null float64
GarageFinish     1379 non-null object
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1379 non-null object
GarageCond       1379 non-null object
PavedDrive       1460 non-null object
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           7 non-null object
Fence            281 non-null object
MiscFeature      54 non-null object
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null object
SaleCondition    1460 non-null object
SalePrice        1460 non-null int64
dtypes: float64(3), int64(35), object(43)
memory usage: 924.0+ KB
```

4.2 Gestion des variables catégorielles

On regarde les valeurs uniques pour identifier les variables catégorielles

[4]:
```
for colname, serie in df.iteritems():
    print(colname + " has " + str(serie.drop_duplicates().shape[0]) + " unique␣
 ↪values.")
```

```
Id has 1460 unique values.
MSSubClass has 15 unique values.
MSZoning has 5 unique values.
LotFrontage has 111 unique values.
LotArea has 1073 unique values.
Street has 2 unique values.
Alley has 3 unique values.
LotShape has 4 unique values.
LandContour has 4 unique values.
Utilities has 2 unique values.
LotConfig has 5 unique values.
LandSlope has 3 unique values.
Neighborhood has 25 unique values.
Condition1 has 9 unique values.
Condition2 has 8 unique values.
BldgType has 5 unique values.
HouseStyle has 8 unique values.
OverallQual has 10 unique values.
OverallCond has 9 unique values.
YearBuilt has 112 unique values.
YearRemodAdd has 61 unique values.
RoofStyle has 6 unique values.
RoofMatl has 8 unique values.
Exterior1st has 15 unique values.
Exterior2nd has 16 unique values.
MasVnrType has 5 unique values.
MasVnrArea has 328 unique values.
ExterQual has 4 unique values.
ExterCond has 5 unique values.
Foundation has 6 unique values.
BsmtQual has 5 unique values.
BsmtCond has 5 unique values.
BsmtExposure has 5 unique values.
BsmtFinType1 has 7 unique values.
BsmtFinSF1 has 637 unique values.
BsmtFinType2 has 7 unique values.
BsmtFinSF2 has 144 unique values.
BsmtUnfSF has 780 unique values.
TotalBsmtSF has 721 unique values.
Heating has 6 unique values.
```

```
HeatingQC has 5 unique values.
CentralAir has 2 unique values.
Electrical has 6 unique values.
1stFlrSF has 753 unique values.
2ndFlrSF has 417 unique values.
LowQualFinSF has 24 unique values.
GrLivArea has 861 unique values.
BsmtFullBath has 4 unique values.
BsmtHalfBath has 3 unique values.
FullBath has 4 unique values.
HalfBath has 3 unique values.
BedroomAbvGr has 8 unique values.
KitchenAbvGr has 4 unique values.
KitchenQual has 4 unique values.
TotRmsAbvGrd has 12 unique values.
Functional has 7 unique values.
Fireplaces has 4 unique values.
FireplaceQu has 6 unique values.
GarageType has 7 unique values.
GarageYrBlt has 98 unique values.
GarageFinish has 4 unique values.
GarageCars has 5 unique values.
GarageArea has 441 unique values.
GarageQual has 6 unique values.
GarageCond has 6 unique values.
PavedDrive has 3 unique values.
WoodDeckSF has 274 unique values.
OpenPorchSF has 202 unique values.
EnclosedPorch has 120 unique values.
3SsnPorch has 20 unique values.
ScreenPorch has 76 unique values.
PoolArea has 8 unique values.
PoolQC has 4 unique values.
Fence has 5 unique values.
MiscFeature has 5 unique values.
MiscVal has 21 unique values.
MoSold has 12 unique values.
YrSold has 5 unique values.
SaleType has 9 unique values.
SaleCondition has 6 unique values.
SalePrice has 663 unique values.
```

A la vue du fichier de description, qui est aussi confirmée par le nombre de modalités, un certain nombre de variables peuvent être redéfinies en variables catégorielles.

Initialement : on avait décidé de remettre les noms longs des modalités mais les graphiques n'étaient plus par la suite lisibles. On s'est aussi aperçu que les modalités NA des dictionnaires n'étaient pas bien pris en compte initialement et donc on a utilisé la fonction fillna avec le code suivant :

[0]:
```
def DefineCategoricalVariableAndDefineNa(myDf):
    myDf["MSSubClass"] = pd.Categorical(myDf["MSSubClass"], ordered=False)
    myDf["MSZoning"] = pd.Categorical(myDf["MSZoning"], ordered=False)
    myDf["Street"] = pd.Categorical(myDf["Street"], ordered=False).
↪rename_categories({'Grvl':'Gravel','Pave':'Paved'})
    myDf["Alley"].fillna('No alley access', inplace = True)
    myDf["Alley"] = pd.Categorical(myDf["Alley"], ordered=False).
↪rename_categories({'Grvl':'Gravel','Pave':'Paved'})
    myDf["LotShape"] = pd.Categorical(myDf["LotShape"], ordered=False)
    myDf["LandContour"] = pd.Categorical(myDf["LandContour"], ordered=False)
    myDf["Utilities"] = pd.Categorical(myDf["Utilities"], ordered=False)
    myDf["LotConfig"] = pd.Categorical(myDf["LotConfig"], ordered=False)
    myDf["LandSlope"] = pd.Categorical(myDf["LandSlope"], ordered=False)
    myDf["Neighborhood"] = pd.Categorical(myDf["Neighborhood"], ordered=False)
    myDf["Condition1"] = pd.Categorical(myDf["Condition1"], ordered=False)
    myDf["Condition2"] = pd.Categorical(myDf["Condition2"], ordered=False)
    myDf["BldgType"] = pd.Categorical(myDf["BldgType"], ordered=False)
    myDf["HouseStyle"] = pd.Categorical(myDf["HouseStyle"], ordered=False)
    myDf["OverallQual"] = pd.Categorical(myDf["OverallQual"], ordered=True)
    myDf["OverallCond"] = pd.Categorical(myDf["OverallCond"], ordered=True)
    myDf["RoofStyle"] = pd.Categorical(myDf["RoofStyle"], ordered=False)
    myDf["RoofMatl"] = pd.Categorical(myDf["RoofMatl"], ordered=False)
    myDf["Exterior1st"] = pd.Categorical(myDf["Exterior1st"], ordered=False)
    myDf["Exterior2nd"] = pd.Categorical(myDf["Exterior2nd"], ordered=False)
    myDf["MasVnrType"] = pd.Categorical(myDf["MasVnrType"], ordered=False)
    myDf["ExterQual"] = pd.Categorical(myDf["ExterQual"], ordered=True)
    myDf["ExterCond"] = pd.Categorical(myDf["ExterCond"], ordered=True)
    myDf["Foundation"] = pd.Categorical(myDf["Foundation"], ordered=False)
    myDf["BsmtQual"].fillna("No Basement", inplace=True)
    myDf["BsmtQual"] = pd.Categorical(myDf["BsmtQual"], ordered=True)
    myDf["BsmtCond"].fillna("No Basement", inplace=True)
    myDf["BsmtCond"] = pd.Categorical(myDf["BsmtCond"], ordered=True)
    myDf["BsmtExposure"].fillna("No Basement", inplace=True)
    myDf["BsmtExposure"] = pd.Categorical(myDf["BsmtExposure"], ordered=True)
    myDf["BsmtFinType1"].fillna("No Basement", inplace=True)
    myDf["BsmtFinType2"].fillna("No Basement", inplace=True)
    myDf["BsmtFinType1"] = pd.Categorical(myDf["BsmtFinType1"], ordered=True)
    myDf["BsmtFinType2"] = pd.Categorical(myDf["BsmtFinType2"], ordered=True)
    myDf["Heating"] = pd.Categorical(myDf["Heating"], ordered=False)
    myDf["HeatingQC"] = pd.Categorical(myDf["HeatingQC"], ordered=True)
    myDf["CentralAir"] = pd.Categorical(myDf["CentralAir"], ordered=False).
↪rename_categories({'N': 'No', 'Y':'Yes'})
    myDf["Electrical"] = pd.Categorical(myDf["Electrical"], ordered=False)
    myDf["KitchenQual"] = pd.Categorical(myDf["KitchenQual"], ordered=True)
    myDf["Functional"] = pd.Categorical(myDf["Functional"], ordered=True)
    myDf["FireplaceQu"].fillna("No Fireplace", inplace=True)
    myDf["FireplaceQu"] = pd.Categorical(myDf["FireplaceQu"], ordered=True)
```

```
    myDf["GarageType"].fillna("No Garage", inplace=True)
    myDf["GarageType"] = pd.Categorical(myDf["GarageType"], ordered=False)
    myDf["GarageFinish"].fillna("No Garage", inplace=True)
    myDf["GarageFinish"] = pd.Categorical(myDf["GarageFinish"], ordered=False)
    myDf["GarageQual"].fillna("No Garage", inplace=True)
    myDf["GarageCond"].fillna("No Garage", inplace=True)
    myDf["GarageQual"] = pd.Categorical(myDf["GarageQual"], ordered=True)
    myDf["GarageCond"] = pd.Categorical(myDf["GarageCond"], ordered=True)
    myDf["PavedDrive"] = pd.Categorical(myDf["PavedDrive"], ordered=False).
↪rename_categories({'Y':'Paved','P':'Partial Pavement','N':'Dirt/Gravel'})
    myDf["PoolQC"].fillna("No Pool", inplace=True)
    myDf["PoolQC"] = pd.Categorical(myDf["PoolQC"], ordered=True)
    myDf["Fence"].fillna("No Fence", inplace = True)
    myDf["Fence"] = pd.Categorical(myDf["Fence"], ordered=False)
    myDf["MiscFeature"].fillna('None', inplace = True)
    myDf["MiscFeature"] = pd.Categorical(myDf["MiscFeature"], ordered=False)
    myDf["SaleType"] = pd.Categorical(myDf["SaleType"], ordered=False)
    myDf["SaleCondition"] = pd.Categorical(myDf["SaleCondition"], ordered=False)
```

[6]:
```
DefineCategoricalVariableAndDefineNa(df)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1460 entries, 0 to 1459
Data columns (total 81 columns):
Id               1460 non-null int64
MSSubClass       1460 non-null category
MSZoning         1460 non-null category
LotFrontage      1201 non-null float64
LotArea          1460 non-null int64
Street           1460 non-null category
Alley            1460 non-null category
LotShape         1460 non-null category
LandContour      1460 non-null category
Utilities        1460 non-null category
LotConfig        1460 non-null category
LandSlope        1460 non-null category
Neighborhood     1460 non-null category
Condition1       1460 non-null category
Condition2       1460 non-null category
BldgType         1460 non-null category
HouseStyle       1460 non-null category
OverallQual      1460 non-null category
OverallCond      1460 non-null category
YearBuilt        1460 non-null int64
YearRemodAdd     1460 non-null int64
RoofStyle        1460 non-null category
```

```
RoofMatl         1460 non-null category
Exterior1st      1460 non-null category
Exterior2nd      1460 non-null category
MasVnrType       1452 non-null category
MasVnrArea       1452 non-null float64
ExterQual        1460 non-null category
ExterCond        1460 non-null category
Foundation       1460 non-null category
BsmtQual         1460 non-null category
BsmtCond         1460 non-null category
BsmtExposure     1460 non-null category
BsmtFinType1     1460 non-null category
BsmtFinSF1       1460 non-null int64
BsmtFinType2     1460 non-null category
BsmtFinSF2       1460 non-null int64
BsmtUnfSF        1460 non-null int64
TotalBsmtSF      1460 non-null int64
Heating          1460 non-null category
HeatingQC        1460 non-null category
CentralAir       1460 non-null category
Electrical       1459 non-null category
1stFlrSF         1460 non-null int64
2ndFlrSF         1460 non-null int64
LowQualFinSF     1460 non-null int64
GrLivArea        1460 non-null int64
BsmtFullBath     1460 non-null int64
BsmtHalfBath     1460 non-null int64
FullBath         1460 non-null int64
HalfBath         1460 non-null int64
BedroomAbvGr     1460 non-null int64
KitchenAbvGr     1460 non-null int64
KitchenQual      1460 non-null category
TotRmsAbvGrd     1460 non-null int64
Functional       1460 non-null category
Fireplaces       1460 non-null int64
FireplaceQu      1460 non-null category
GarageType       1460 non-null category
GarageYrBlt      1379 non-null float64
GarageFinish     1460 non-null category
GarageCars       1460 non-null int64
GarageArea       1460 non-null int64
GarageQual       1460 non-null category
GarageCond       1460 non-null category
PavedDrive       1460 non-null category
WoodDeckSF       1460 non-null int64
OpenPorchSF      1460 non-null int64
EnclosedPorch    1460 non-null int64
3SsnPorch        1460 non-null int64
```

```
ScreenPorch      1460 non-null int64
PoolArea         1460 non-null int64
PoolQC           1460 non-null category
Fence            1460 non-null category
MiscFeature      1460 non-null category
MiscVal          1460 non-null int64
MoSold           1460 non-null int64
YrSold           1460 non-null int64
SaleType         1460 non-null category
SaleCondition    1460 non-null category
SalePrice        1460 non-null int64
dtypes: category(46), float64(3), int64(32)
memory usage: 477.6 KB
```

Les données semblent correctement typées.

L'analyse sommaire des données permet de voir les éléments suivants :

- le premier champs est un identifiant numérique
- on a ensuite des champs permettant de caractériser **la localisation et les caractéristiques de la propriété**
- on a des champs définissant ensuite des **informations générales sur la construction**
- on a des champs décrivant **la toiture, l'emprise au sol, le sous-sol**
- on a des champs décrivant les **accès aux commodités** (électrique, chauffage, air conditioné…)
- on a des champs décrivant la **maison au-dessus du sous-sol**
- on a des champs décrivant des **commodités spéciales (piscines, vérandas) et le garage**
- on a enfin des champs décrivant les **caractéristiques de la vente**

5 Exploration

5.1 Valeurs manquantes

[7]:
```
# Nombre de valeurs manquantes par variable
total = df.isnull().sum().sort_values(ascending=False)
percent = (df.isnull().sum()/df.isnull().count()).sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(10)
```

[7]:

	Total	Percent
LotFrontage	259	0.177397
GarageYrBlt	81	0.055479
MasVnrType	8	0.005479
MasVnrArea	8	0.005479
Electrical	1	0.000685
SalePrice	0	0.000000
ExterCond	0	0.000000
RoofStyle	0	0.000000
RoofMatl	0	0.000000
Exterior1st	0	0.000000

On peut observer qu'il y a peu de données manquantes sur ce dataset : un petit problème avec la variable LotFrontage avec une complétion à 83%.

5.2 Exploration univariée

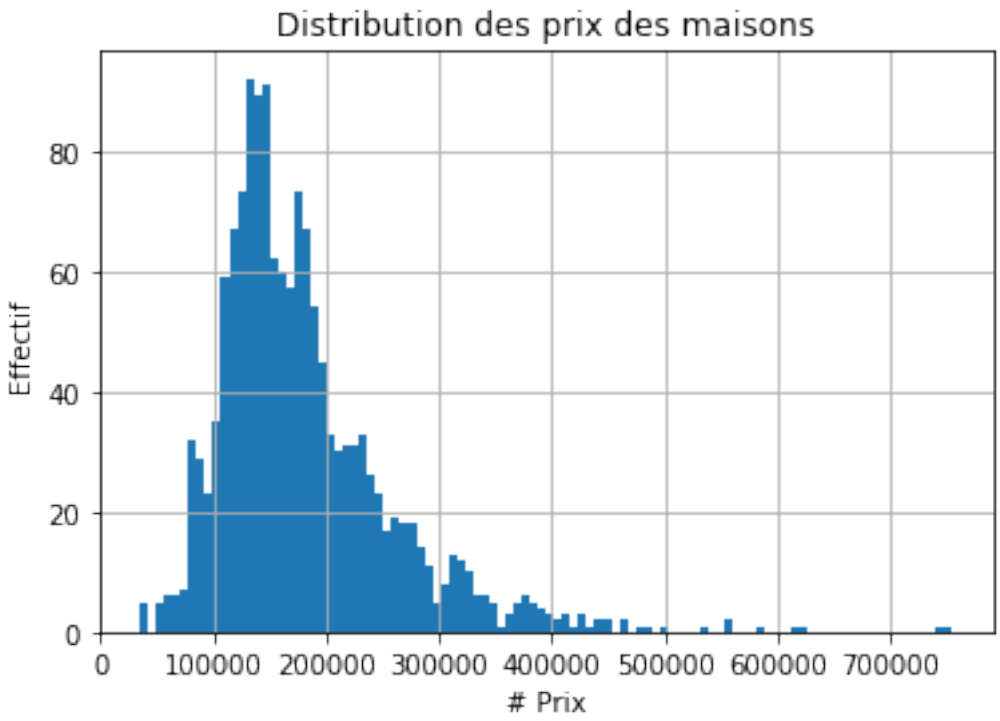
5.2.1 La variable cible

On va regarder la distribution de notre variable à expliquer à savoir le prix des logements.

[8]:
```
df['SalePrice'].describe()
```

[8]:
```
count      1460.000000
mean     180921.195890
std       79442.502883
min       34900.000000
25%      129975.000000
50%      163000.000000
75%      214000.000000
max      755000.000000
Name: SalePrice, dtype: float64
```

[9]:
```
df["SalePrice"].hist(bins=100)
plt.title("Distribution des prix des maisons")
plt.xlabel("# Prix")
plt.ylabel("Effectif")
plt.show()
```

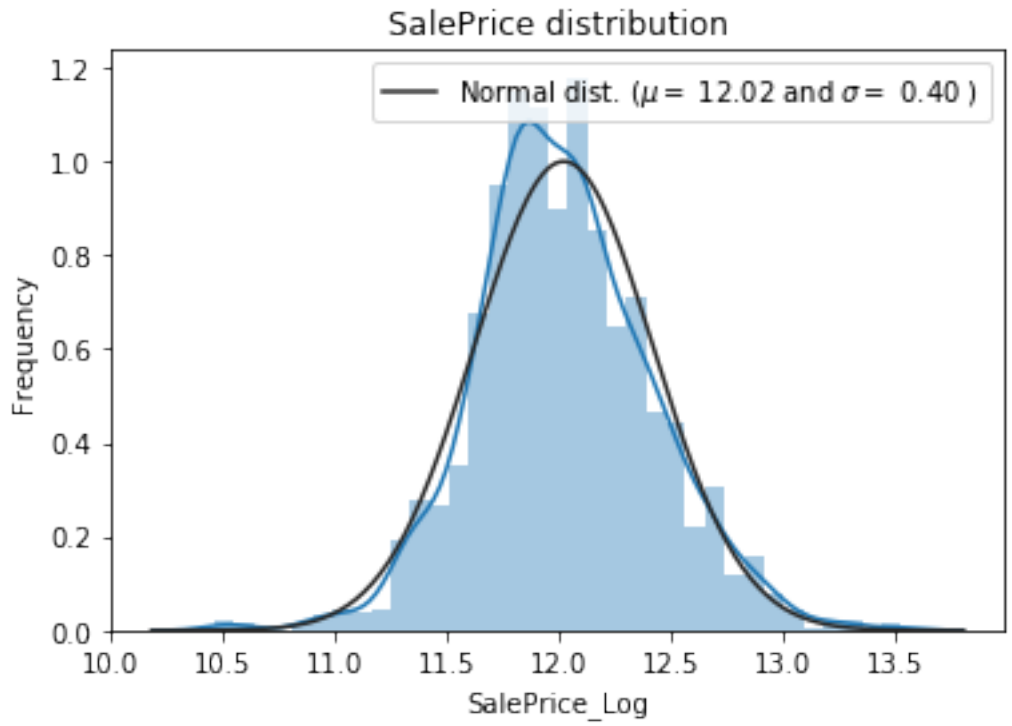


La cible de notre modèle s'apparente à une log-normale. Elle a pour moyenne 181 K et un écart type de l'ordre de 80 K avec une plage de valeurs compris entre 35 K et 755 K. Pour vérifier l'hypothèse de log-normalité, appliquons la transformation et calculons les paramètres $\mu =$ et $\$\sigma$

[10]:
```
df["SalePrice_Log"] = np.log1p(df["SalePrice"])

from scipy.stats import norm
#Check the new distribution
sns.distplot(df['SalePrice_Log'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(df['SalePrice_Log'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu,␣
 ↪sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('SalePrice distribution')
```

```
 mu = 12.02 and sigma = 0.40
```

[10]:
```
Text(0.5, 1.0, 'SalePrice distribution')
```



L'hypothèse de log-normalité est validée.

[0]:
```
df = df.drop("SalePrice_Log", axis =1)
```

5.2.2 Les autres variables numériques

[12]:
```
df_num = df.select_dtypes(include = ['float64', 'int64'])
df_num = df_num.drop(["SalePrice", "Id"], axis = 1)
df_num.describe()
```

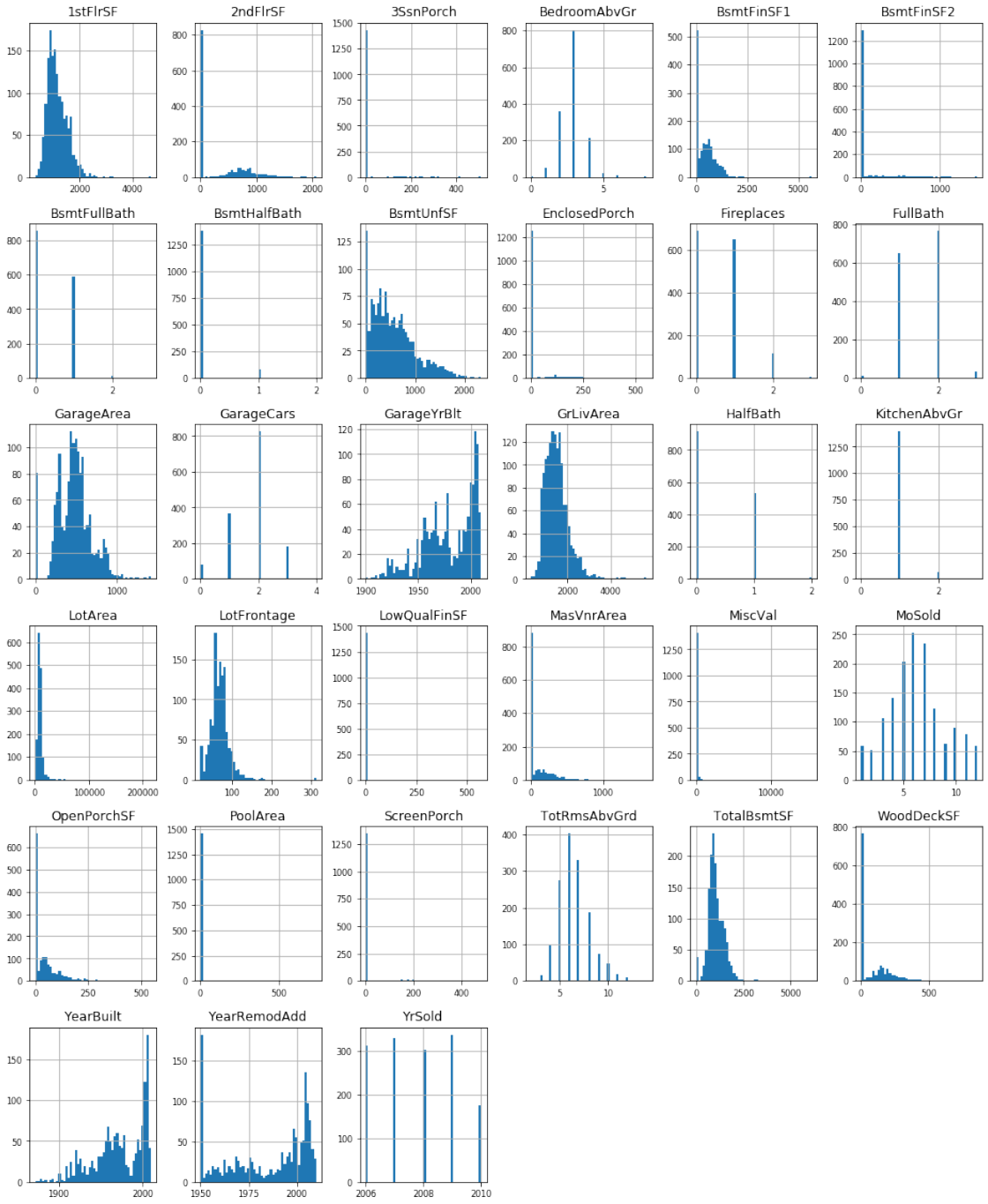
[12]:

	LotFrontage	LotArea	...	MoSold	YrSold
count	1201.000000	1460.000000	...	1460.000000	1460.000000
mean	70.049958	10516.828082	...	6.321918	2007.815753
std	24.284752	9981.264932	...	2.703626	1.328095
min	21.000000	1300.000000	...	1.000000	2006.000000
25%	59.000000	7553.500000	...	5.000000	2007.000000
50%	69.000000	9478.500000	...	6.000000	2008.000000
75%	80.000000	11601.500000	...	8.000000	2009.000000
max	313.000000	215245.000000	...	12.000000	2010.000000

[8 rows x 33 columns]

[13]:
```
df_num.hist(figsize=(16, 20), bins=50, xlabelsize=8, ylabelsize=8)
```

[13]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1e294a8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1d8ad30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1db8e80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1d66fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1d22160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1cd32b0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1c83400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1cb2828>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1cb2860>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1c1c390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1bc6940>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1bfbef0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1bb74e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1b68a90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1b28080>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1ad7630>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1a89be0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1a461d0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1a77780>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1a2ad30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b19e7320>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b19998d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1949e80>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1906470>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1938a20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b18ebfd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b18a65c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1858b70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1817160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b17c4710>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f71b17f7cc0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b17b72b0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1765860>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b1719e10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b16d5400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f71b16869b0>]],
      dtype=object)
```



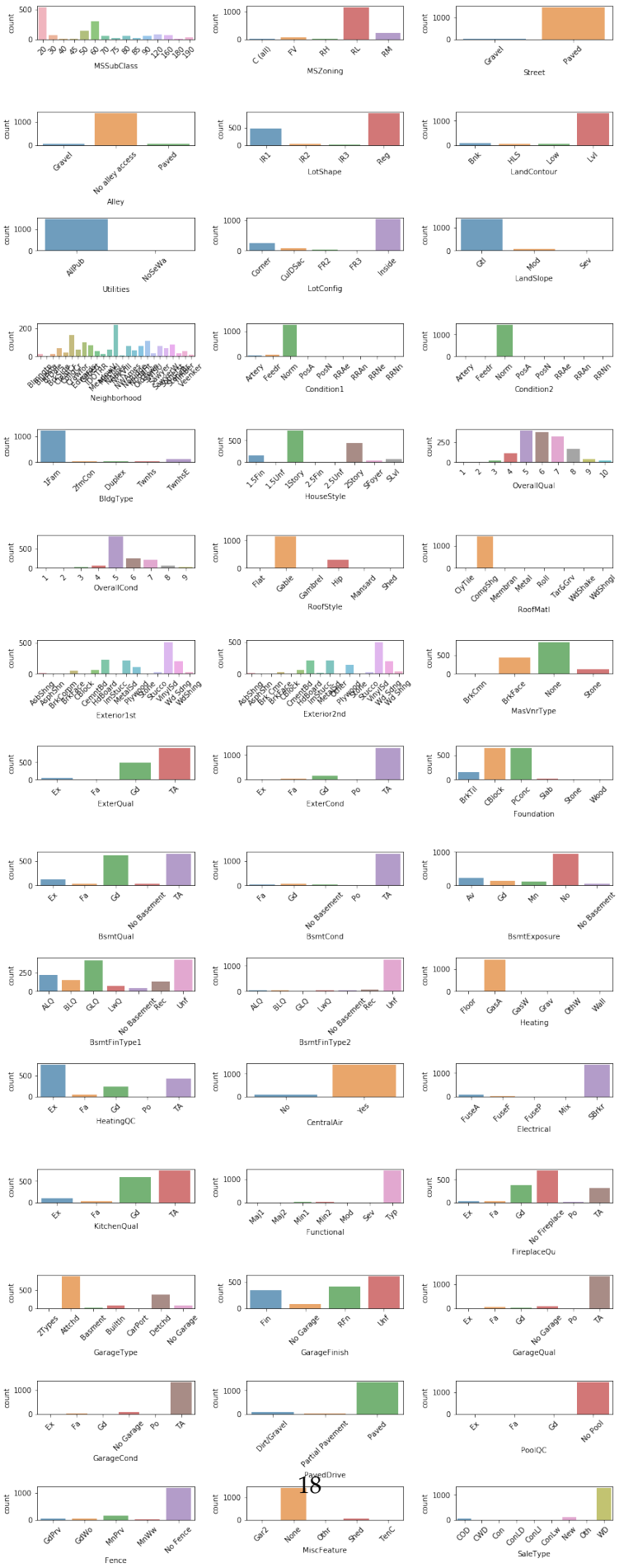
On peut observer : * sur la **la localisation et les caractéristiques de la propriété** * que les variables LotFrontage et LotArea ont une distribution de type LogNormal comme la cible * que la variable LotArea a des valeurs extrêmes supérieures * sur les **informations générales sur la construction** * des années de construction entre 1872 et 2010 * des années de rénovation entre 1950 et 2010 * sur la **la toiture, l'emprise au sol, le sous-sol** * que la valeur surface de maçonnerie a des valeurs extrêmes de même que les variables BsmtFinSF1 et BsmtFinSF2 * que les variables TotalBsmtSF et

BsmtUnfSF ont aussi une distribution de type LogNormal * sur les caractéristiques de la **maison au-dessus du sous-sol** * que la surface habitable a une distribution de type LogNormal * que la majorité des maisons ont un étage * que le nombre de chambres le plus important en terme de modalités est 3 * des informations sur les salles de bains, douches, cheminées avec des modalités comprises entre 0 et 3 * sur les **commodités spéciales (piscines, vérandas) et le garage** * 2 places de parking est la modalité la plus présente * une surface du garage qui suit aussi une loi normale * sur les **caractéristiques des ventes** : * que les années de vente sont comprises entre 2006 et 2010, * que les 12 mois sont représentés avec une gaussienne avec des ventes plus importantes sur l'été

5.2.3 Les variables catégorielles

[0]:
```
df_cat = df.select_dtypes(include = ['category'])
```

[15]:
```
fig, axes = plt.subplots(round(len(df_cat.columns) / 3), 3, figsize=(12, 30))

for i, ax in enumerate(fig.axes):
    if i < len(df_cat.columns):
        ax.set_xticklabels(ax.xaxis.get_majorticklabels(), rotation=45)
        sns.countplot(x=df_cat.columns[i], alpha=0.7, data=df_cat, ax=ax)

fig.tight_layout()
```


count
MSSubClass
MSZoning
Street
Alley
LotShape
LandContour
Utilities
LotConfig
LandSlope
Neighborhood
Condition1
Condition2
BldgType
HouseStyle
OverallQual
OverallCond
RoofStyle
RoofMatl
Exterior1st
Exterior2nd
MasVnrType
ExterQual
ExterCond
Foundation
BsmtQual
BsmtCond
BsmtExposure
BsmtFinType1
BsmtFinType2
Heating
HeatingQC
CentralAir
Electrical
KitchenQual
Functional
FireplaceQu
GarageType
GarageFinish
GarageQual
GarageCond
PavedDrive
PoolQC
Fence
MiscFeature
SaleType

Pour un certain nombre de variables catégorielles, une modalité représente la très grande majorité de l'information. Par conséquent, les variables n'ont pas à être prises en compte par la suite dans le modèle. On pourra ne pas considérer les variables suivantes MSZoning, Street, Alley, LandContour, Utilities, LandSlope, Condition1, Condition2, BldgType, RoofMatl, BsmtCond, Heating, BsmtFinType2, CentralAir, Functional, GarageQual, GarageCond, PoolQC,MiscFeature. On peut observer aussi que la qualité de la finition de la maison s'apparente à une gaussienne.

5.2.4 Conclusion sur l'analyse univariée

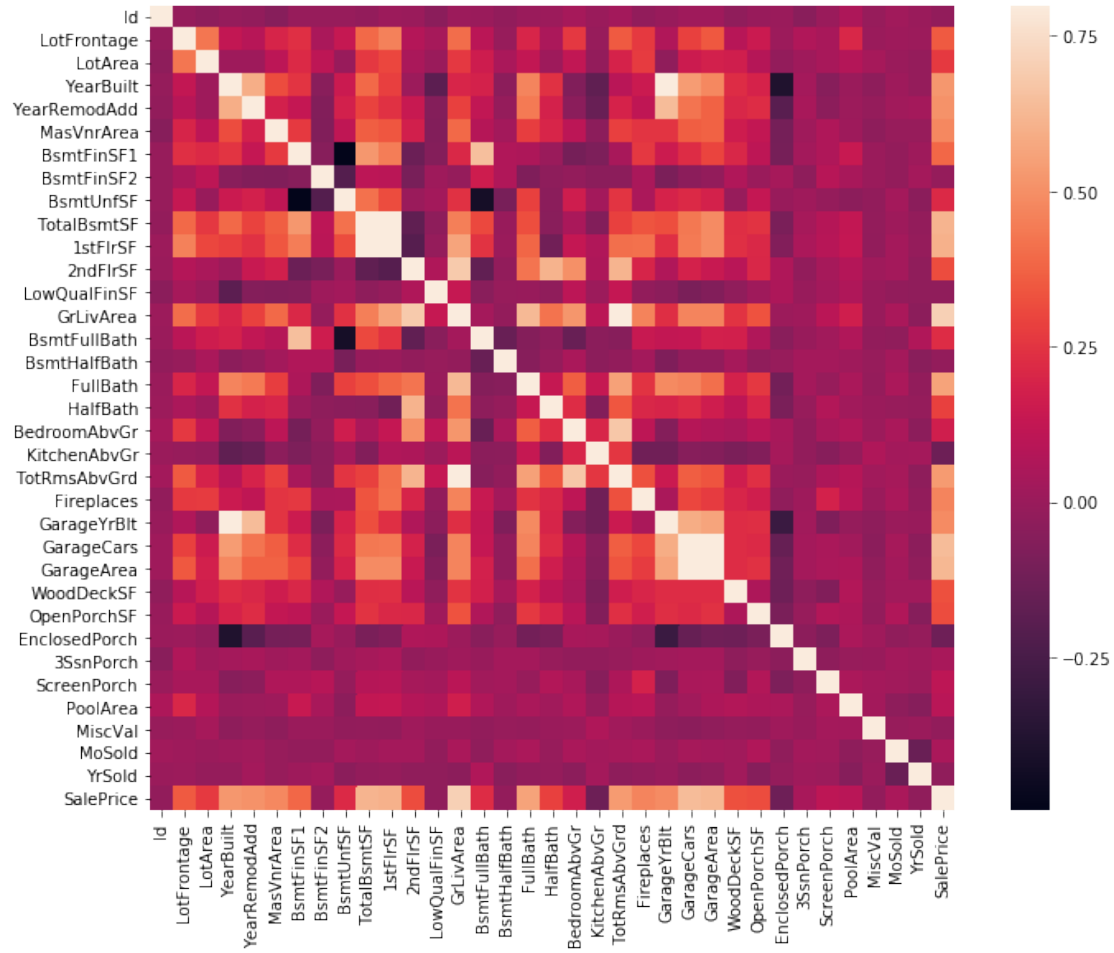
On a pu voir que : * notre cible de prix de vente suit une loi LogNormal * un certain nombre de variables numériques ont aussi un comportement assez similaire à notre target en termes de distribution * une élimination naturelle des variables catégorielles n'apportant pas d'informations.

5.3 Analyse bivariée

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

5.3.1 Les variables quantitatives

[16]:
```
corr = df.corr()
f, ax = plt.subplots(figsize=(12, 9))
sns.heatmap(corr, vmax=.8, square=True);
```

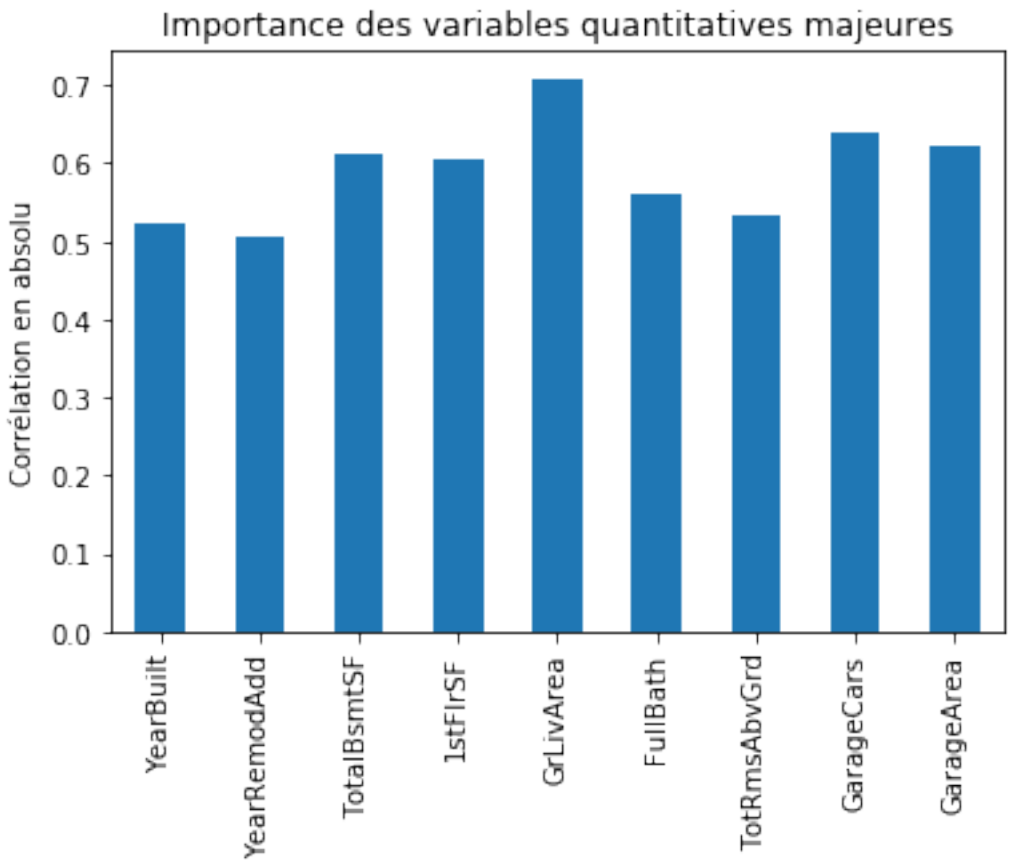


La matrice de corrélation entre les variables quantitatives avec de nombreuses variables permet néanmoins d'extraire des informations intéressantes : * les variables qui sont très fortement corrélées entre elles : on pourra citer YearBuilt et GarageYrBlt, GarageCars et GarageArea et TotalBsmtSF et 1FstFlrSF. Une des deux variables pourra être ignorée dans le cas de la prédiction car elles transmettent une information identique. * un focus spécifique sur la variable à prédire : on peut observer qu'elle est fortement liée positivement à GrLivArea, TotalBsmtSF, GarageCars pour les plus significatives. Elle a l'air corrélée négativement avec KitchenAbvGr et EnclosedPorch.

[0]:
```
dfSalePrice = corr[np.abs(corr['SalePrice'])>0.5]['SalePrice']
dfSalePrice = dfSalePrice.drop('SalePrice')
```

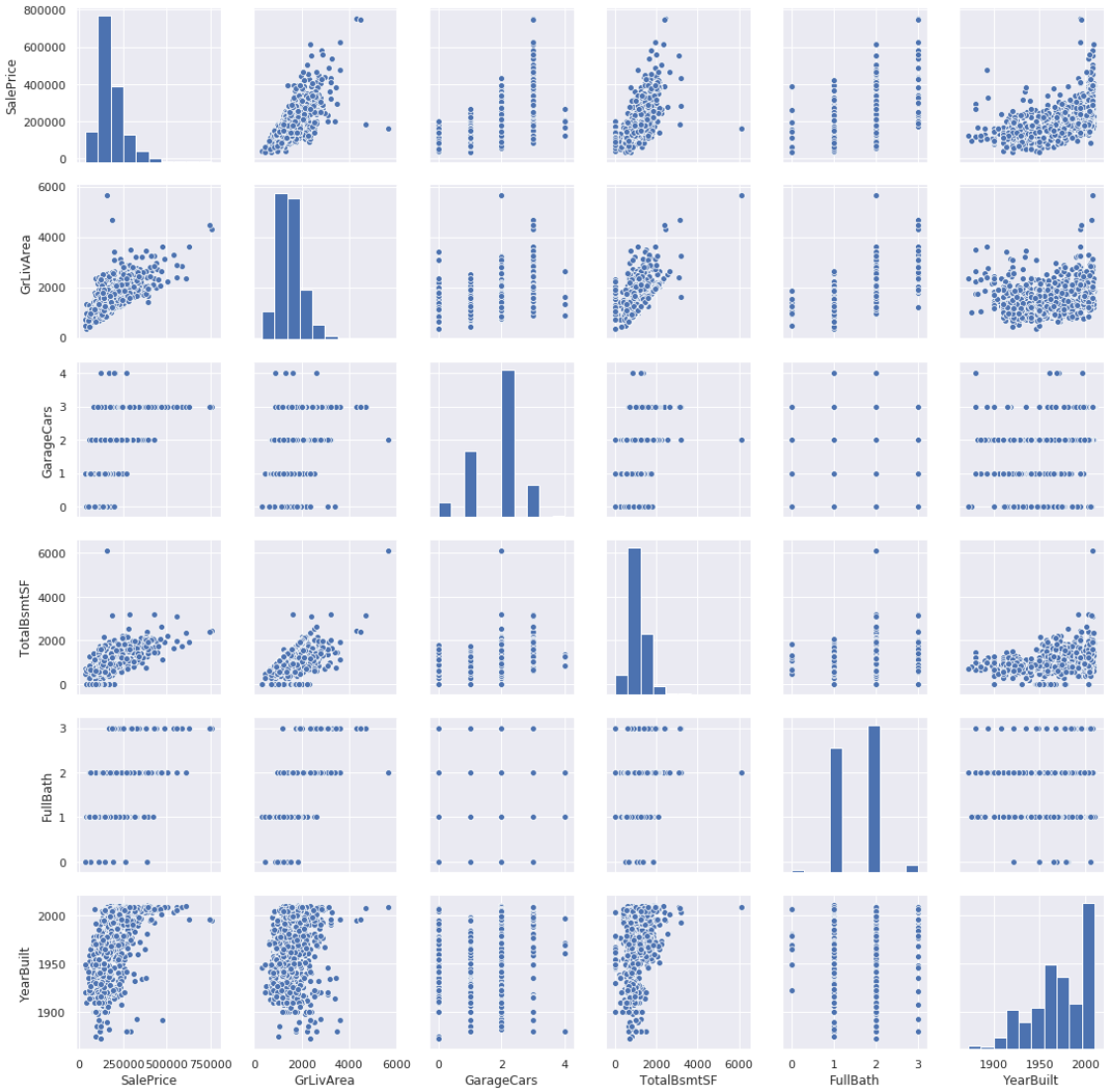
[18]:
```
impSalePrice = dfSalePrice.plot(kind="bar")
impSalePrice.set_title("Importance des variables quantitatives majeures")
impSalePrice.set_ylabel("Corrélation en absolu")
```

[18]:
```
Text(0, 0.5, 'Corrélation en absolu')
```



On regarde à travers un scatterplot pour visualiser la relation entre les variables numériques les plus significatives (en gardant uniquement une variable si problème de colinéarité) avec la variable cible.

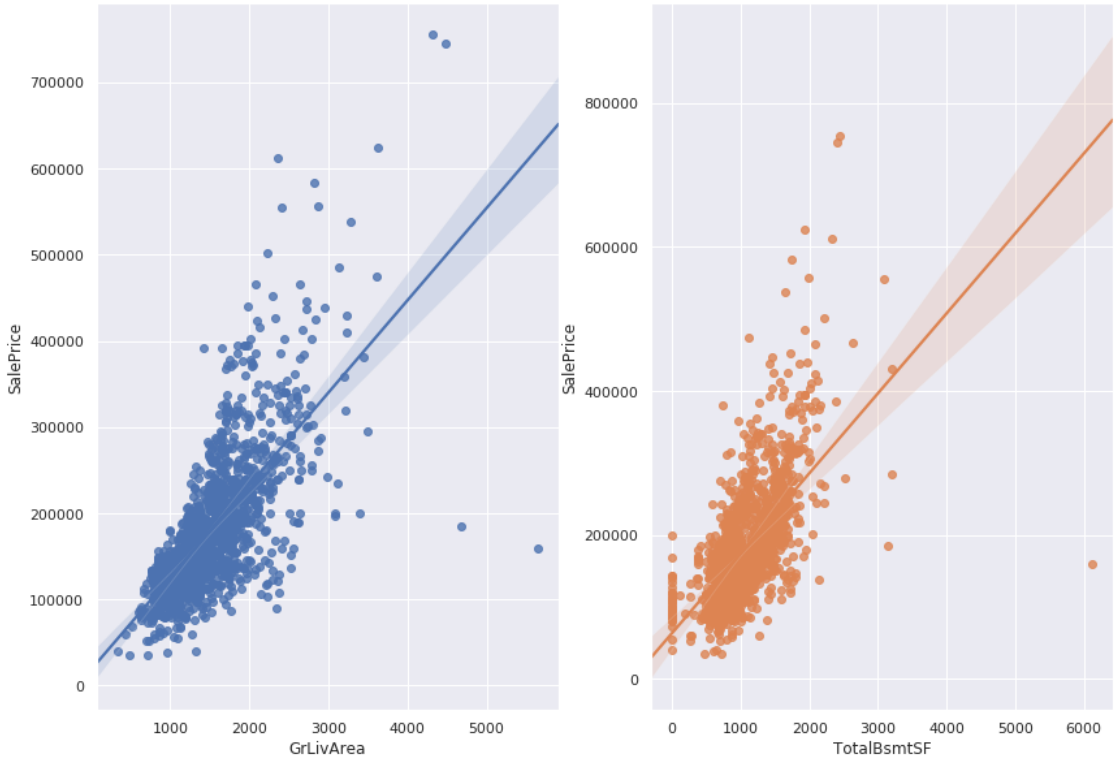
[19]:
```
sns.set()
cols = ['SalePrice', 'GrLivArea', 'GarageCars', 'TotalBsmtSF', 'FullBath',␣
 ↪'YearBuilt']
sns.pairplot(df[cols], height = 2.5)
plt.show();
```



Le prix de vente semble en tendance évoluer positivement fonction du nombre de places de parkings et de salle de bains (même s'il existe une certaine disparité). Faisons un focus sur des liens entre certaines variables avec la variable cible.

[20]:
```
fig,(ax1,ax2) = plt.subplots(ncols=2)
fig.set_size_inches(14,10)
sns.regplot(x="GrLivArea",y="SalePrice",data=df, ax=ax1)
sns.regplot(x="TotalBsmtSF",y="SalePrice",data=df,ax=ax2)
```

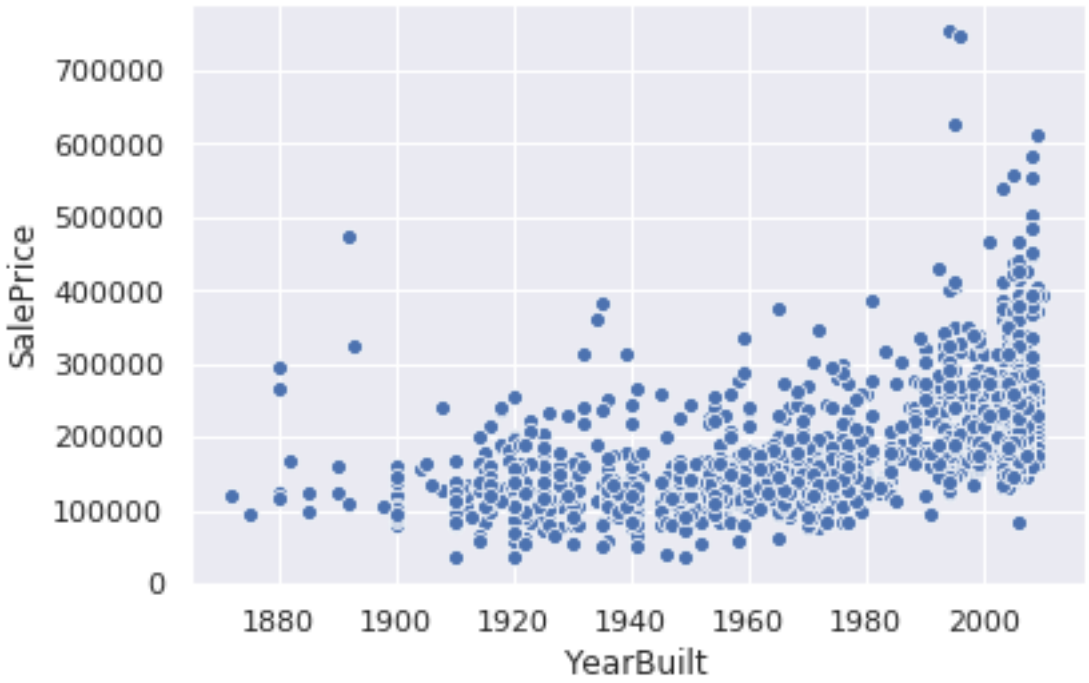
[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71aa41d898>
```



Le prix de vente évolue positivement fonction de la surface totale habitable et de l'emprise au sol.

[21]:
```
sns.scatterplot(x="YearBuilt",y="SalePrice",data=df)
```

[21]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71aa4274e0>
```



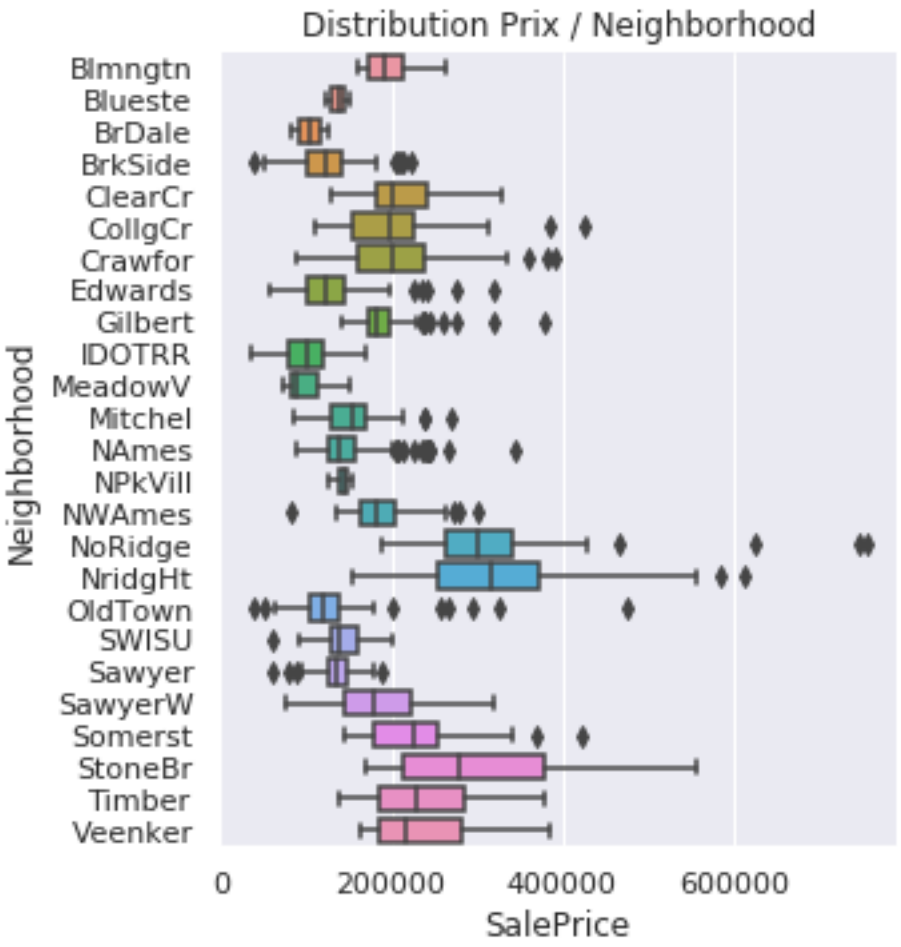
Le prix de vente semble augmenter en fonction de l'année de construction. Peut-être à prendre avec précaution car nous ne savons pas si les prix de vente sont en prix constant. Dans le cas contraire, cela reflète l'inflation de 1880 à 2010.

5.3.2 Les variables catégorielles

En croisant les variables catégorielles, il est possible de trouver des relations logiques avec notre cible. Pour certaines analyses, il est nécessaire d'avoir une vision métier supplémentaire par exemple pour la logique géographique au niveau des districts (neighborhood) qui peut faire sens pour une personne connaissant le marché immobilier ou qui nécessiterait d'introduire du featuring engeenering avec le niveau de vie moyen par district.

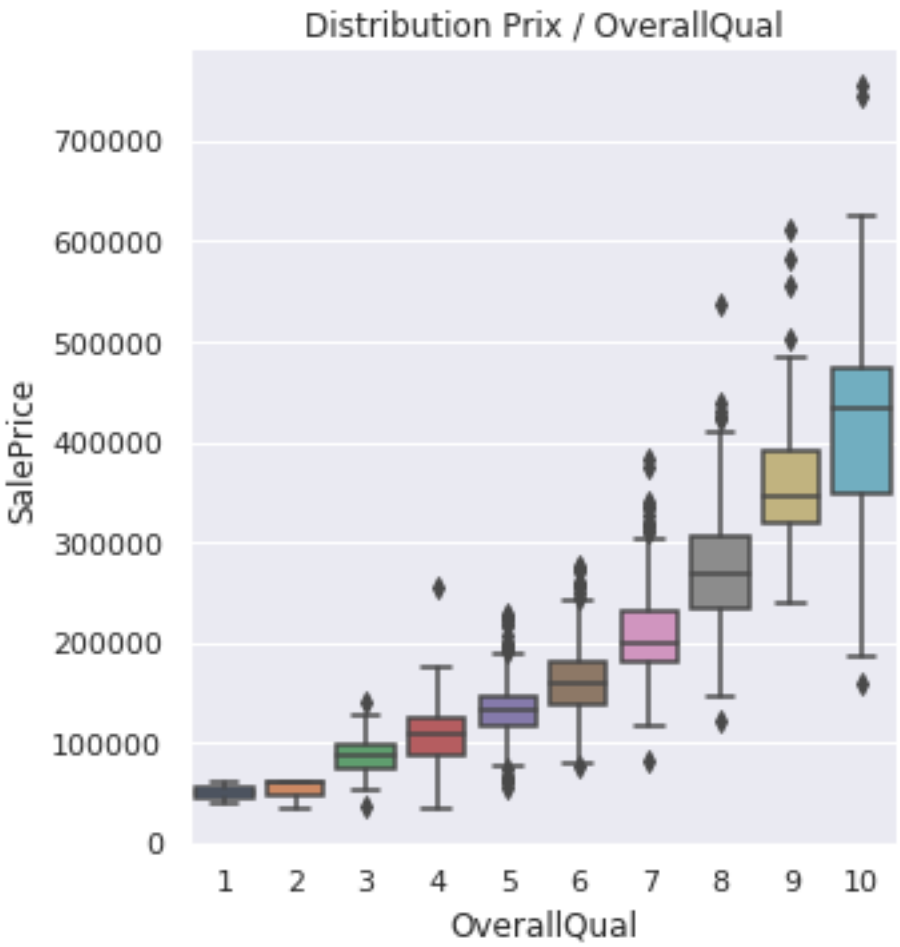
[22]:
```
sns.catplot(x="SalePrice", y="Neighborhood", kind="box",data=df)
plt.title("Distribution Prix / Neighborhood")
```

[22]:
```
Text(0.5, 1, 'Distribution Prix / Neighborhood')
```



[23]:
```
sns.catplot(x="OverallQual", y="SalePrice", kind="box", data=df)
plt.title("Distribution Prix / OverallQual")
```

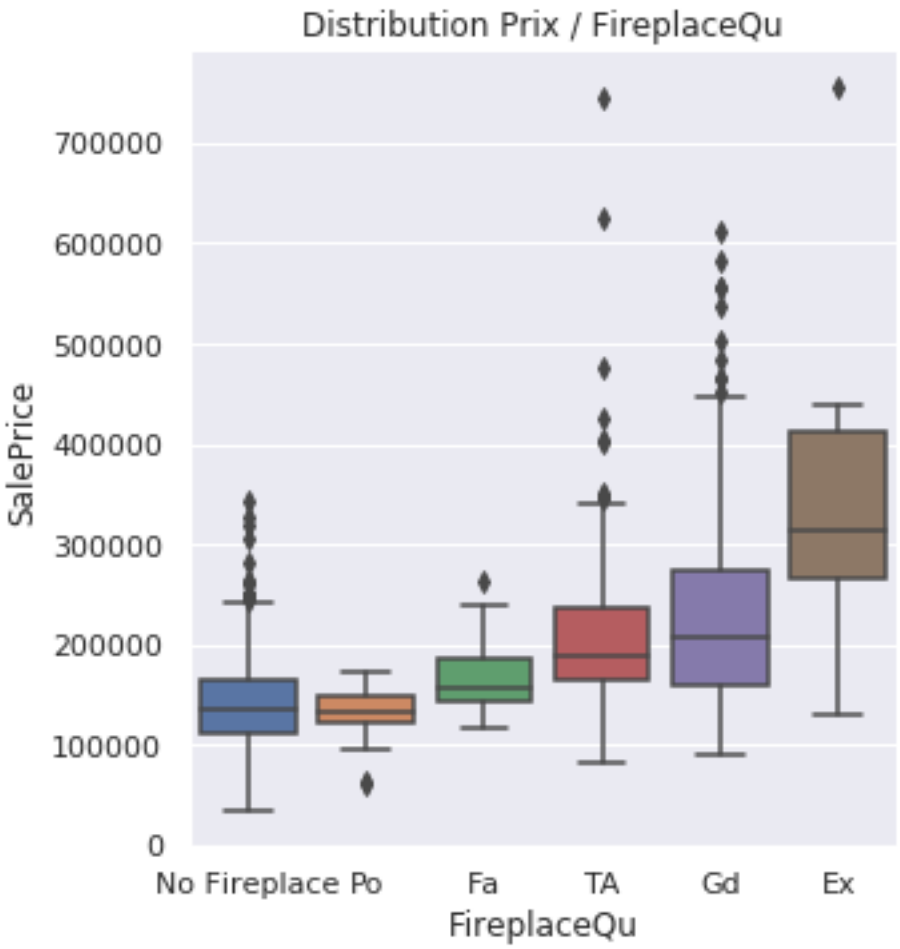
[23]:
```
Text(0.5, 1, 'Distribution Prix / OverallQual')
```



On observe de manière naturelle que le prix de vente dépend de l'évaluation de la qualité des matériaux et de la finition de la maison.

[24]:
```
sns.catplot(x="FireplaceQu", y="SalePrice", kind="box", order=["No Fireplace",␣
↪"Po", "Fa", "TA", "Gd", "Ex"], data=df)
plt.title("Distribution Prix / FireplaceQu")
```

[24]:
```
Text(0.5, 1, 'Distribution Prix / FireplaceQu')
```



Les variables avec une échelle de valeur comme FirePlaceQu (KitchenQual, HeatingQC, BsmtExposure,BsmtCond, BsmtQual, ...) ont un prix qui en tendance évolue positivement fonction d'une note élevée fonction du critère étudiée ce qui apparaît logique avec néanmoins des points atypiques pour certaines modalités.

5.3.3 Conclusion sur l'analyse bivariée

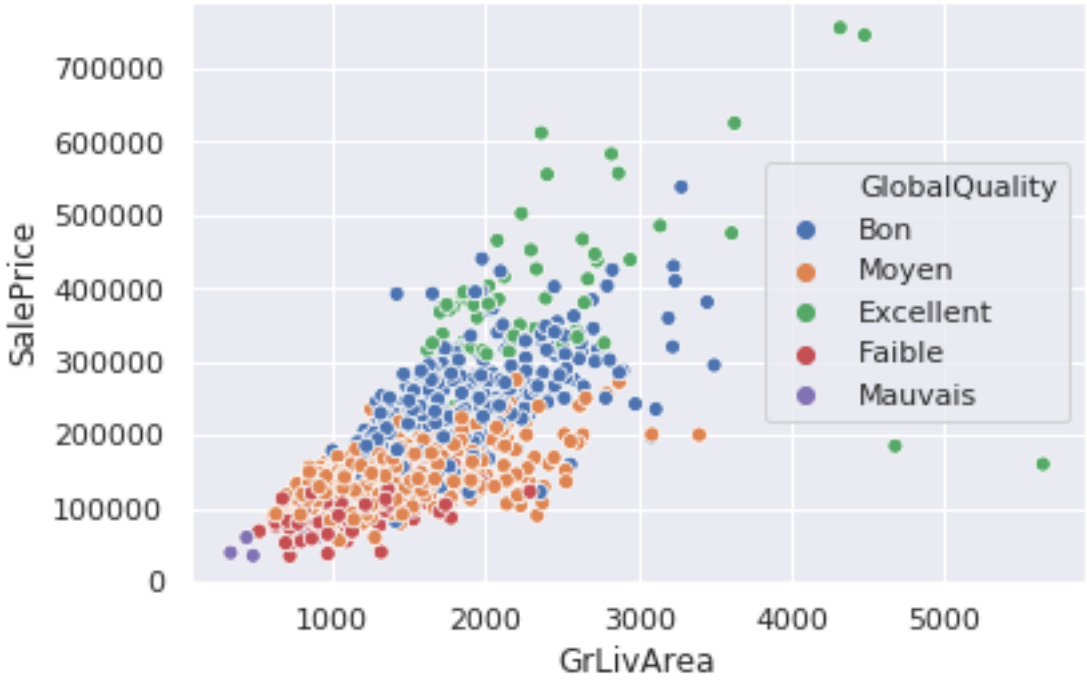
La target est sensible : * à des variables numériques les plus discriminantes comme la surface, le nombre de places de garage/salle de bain... * à des variables catégorielles comme des échelles de valeurs allant de mauvais à excellent... * certaines variables catégorielles peuvent faire sens mais nécessite une expertise métier.

5.4 Analyse multivariée

[0]:
```
df["GlobalQuality"] =df["OverallQual"].replace({1:"Mauvais",2:"Mauvais",3:
↪"Faible",4:"Faible",5:"Moyen",6:"Moyen",7:"Bon",8:"Bon",9:"Excellent",10:
↪"Excellent"})
```

[26]:
```
sns.scatterplot(x="GrLivArea", y="SalePrice", hue="GlobalQuality", data=df)
```

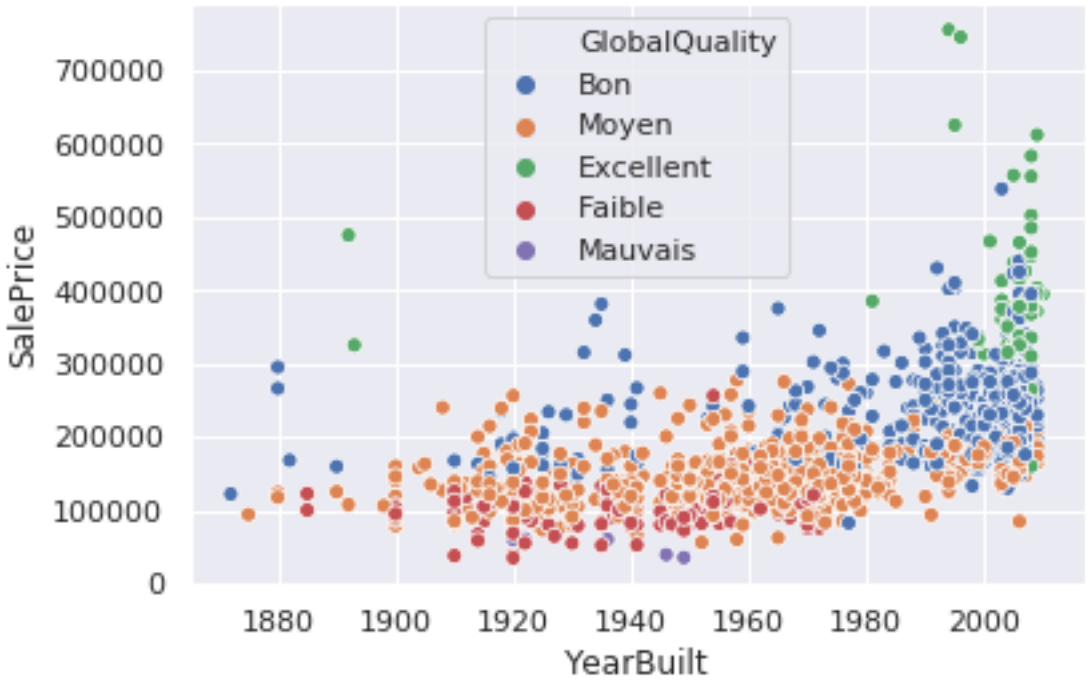
[26]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71ab09ed30>
```



On peut observer que le prix dépend fortement de la surface et de la qualité de la maison. Les maisons les plus chères sont les plus grandes avec une très bonne notation et inversement.

[27]:
```
sns.scatterplot(x="YearBuilt", y="SalePrice", hue="GlobalQuality", data=df)
```

[27]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f71a9e2e3c8>
```



On avait pu observer que le prix de vente (moyennant le fait qu'on ne sait si on raisonne en prix constant) augmente en tendance fonction de l'année de construction. On observe aussi que l'appréciation de qualité est plus élevée (bon, excellent) si la maison est récente.

5.4.1 Conclusion sur l'analyse multivariée

D'autres analyses multivariées auraient pu être réalisées aux vues du nombre de variables dans le jeu de données. Nous avons isolé quelques analyses multivariées permettant de voir des tendances entre certaines variables et la variable cible.

L'analyse exploratoire nous a permis : * de voir la distribution de notre variable cible * d'éliminer certaines variables catégorielles ne présentant pas d'information utile dans le cas de sur-représentation d'une modalité majoritaire * d'éliminer les variables numériques qui ont une corrélation très importantes entre elles : elles expriment le même type d'information * les liens à travers les variables catégorielles et numériques significatives par rapport à notre cible à travers des analyses bivariées ou multivariées.

6 Preprocessing pour scikit-learn

6.1 Lecture des données

[28]:
```
df_train = pd.read_csv("https://raw.githubusercontent.com/anthonymoisan/
 ↪DeepLearningPredictHousePrices/master/input/train.csv")
print("taille du jeu de donnees train :", df_train.shape)
```

```
df_test = pd.read_csv("https://raw.githubusercontent.com/anthonymoisan/
 ↪DeepLearningPredictHousePrices/master/input/test.csv")
print("taille du jeu de donnees test :", df_test.shape)
```

```
taille du jeu de donnees train : (1460, 81)
taille du jeu de donnees test : (1459, 80)
```

L'ensemble de train est l'ensemble d'apprentissage sur lequel on va construire et valider notre modèle. L'ensemble de test a le même nombre de variables explicatives et doit permettre d'inférer le prix de ventes avec le modèle.

6.2 Prise en compte de l'analyse exploratoire

Définition des variables catégorielles et des champs NA

[0]:
```
DefineCategoricalVariableAndDefineNa(df_train)
```

L'analyse exploratoire a permis d'identifier des variables numériques et catégorielles à enlever car n'apportant pas d'informations.

[30]:
```
listDropNumerical = ["Id", "GarageYrBlt",  "GarageCars", "1stFlrSF"]
listDropCategorical = ["MSZoning", "Street", "Alley", "LandContour",␣
 ↪"Utilities", "LandSlope", "Condition1", "Condition2", "BldgType", "RoofMatl",␣
 ↪"BsmtCond", "Heating", "BsmtFinType2", "CentralAir", "Functional",␣
 ↪"GarageQual", "GarageCond", "PoolQC","MiscFeature"]
print("Nombre de variables catégoriques à supprimer : " +␣
 ↪str(len(listDropCategorical)))
print("Nombre de variables numériques à supprimer : " +␣
 ↪str(len(listDropNumerical)))
```

```
Nombre de variables catégoriques à supprimer : 19
Nombre de variables numériques à supprimer : 4
```

[0]:
```
df_train = df_train.drop(listDropNumerical, axis = 1)
df_train = df_train.drop(listDropCategorical, axis = 1)
```

[32]:
```
print("Taille du dataset suite à prétraitements : ", df_train.shape)
```

```
Taille du dataset suite à prétraitements :  (1460, 58)
```

6.3 Construction des ensembles X et y à partir du dataframe

On construit l'ensemble X, y sur le dataframe résultant :

[0]:
```
X = df_train.drop(["SalePrice"], axis = 1)
y = df_train["SalePrice"]
```

6.4 Preprocessing sur les variables catégorielles

[0]:
```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
```

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 interpréter. Scikit learn requiert un encodage numérique des ces variables. Nous allons donc devoir encoder nos variables explicatives catégorielles à l'aide de variables indicatrices et nous utilison pour cela OneHotEncooder et on impute les modalités manquantes.

[35]:
```
categorical_features = X.columns[X.dtypes == "category"].tolist()
print(categorical_features)
```

```
['MSSubClass', 'LotShape', 'LotConfig', 'Neighborhood', 'HouseStyle',
'OverallQual', 'OverallCond', 'RoofStyle', 'Exterior1st', 'Exterior2nd',
'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
'BsmtExposure', 'BsmtFinType1', 'HeatingQC', 'Electrical', 'KitchenQual',
'FireplaceQu', 'GarageType', 'GarageFinish', 'PavedDrive', 'Fence', 'SaleType',
'SaleCondition']
```

[0]:
```
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))])
```

6.5 Preprocessing sur les variables numériques

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 et la méthode StandardScaler. Dans le cas présent, on mélange des unités différentes donc la standardisation semble appropriée. On a pu aussi voir qu'il y avait des données manquantes sur certaines variables numériques comme LotFrontage que l'on impute par la moyenne.

[37]:
```
numerical_features = X.columns[X.dtypes == "int64"].tolist()+X.columns[X.dtypes␣
 ↪== "float64"].tolist()
print(categorical_features)
```

```
['MSSubClass', 'LotShape', 'LotConfig', 'Neighborhood', 'HouseStyle',
'OverallQual', 'OverallCond', 'RoofStyle', 'Exterior1st', 'Exterior2nd',
'MasVnrType', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
'BsmtExposure', 'BsmtFinType1', 'HeatingQC', 'Electrical', 'KitchenQual',
'FireplaceQu', 'GarageType', 'GarageFinish', 'PavedDrive', 'Fence', 'SaleType',
'SaleCondition']
```

[0]:
```
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler())])
```

[0]:
```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)])
```

6.6 Train, Val

Nous allons prendre sur le training 70% pour le train et 30% pour la validation

[40]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.3,␣
 ↪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 : (1022, 57)
Shape du y_train : (1022,)
Shape du X_test : (438, 57)
Shape du y_test : (438,)
```

Nous standardisons aussi notre cible.

[0]:
```
Yscaler = StandardScaler()
y_train_standardise = Yscaler.fit_transform(y_train[:, None])[:, 0]
y_test_standardise = Yscaler.transform(y_test[:, None])[:, 0]
```

7 Des modèles basées sur la régression linéaire

7.1 Un modèle simple : la régression linéaire

Un premier modèle qui nous servira de *baseline*.

Nous allons aussi introduire l'imputation sur les données sur les données *train*, que nous appliquerons **ENSUITE** sur les données *test*.

7.1.1 Regression linéaire

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

[42]:
```
from sklearn.linear_model import LinearRegression
```

```
clfRegLinear = Pipeline(steps=[('preprocessor', preprocessor),␣
 ↪('LinearRegression', LinearRegression())])
clfRegLinear.fit(X_train,y_train_standardise)
y_trainPredict = clfRegLinear.predict(X_train)
y_testPredict = clfRegLinear.predict(X_test)
print("model score sur le train :  %.3f" % clfRegLinear.score(X_train,␣
 ↪y_train_standardise))
print("model score le test : %.3f" % clfRegLinear.score(X_test,␣
 ↪y_test_standardise))
```

```
model score sur le train :  0.891
model score le test : 0.853
```

On observe à priori un modèle avec un coefficient de détermination correct et on a une bonne généralisation sur l'ensemble de validation

7.1.2 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 prix de vente.

Regardons ces coefficients :

[0]:
```
ohe = (clfRegLinear.named_steps['preprocessor']
         .named_transformers_['cat']
         .named_steps['onehot'])
feature_names = ohe.get_feature_names(input_features=categorical_features)
feature_names = np.r_[numerical_features, feature_names]
```

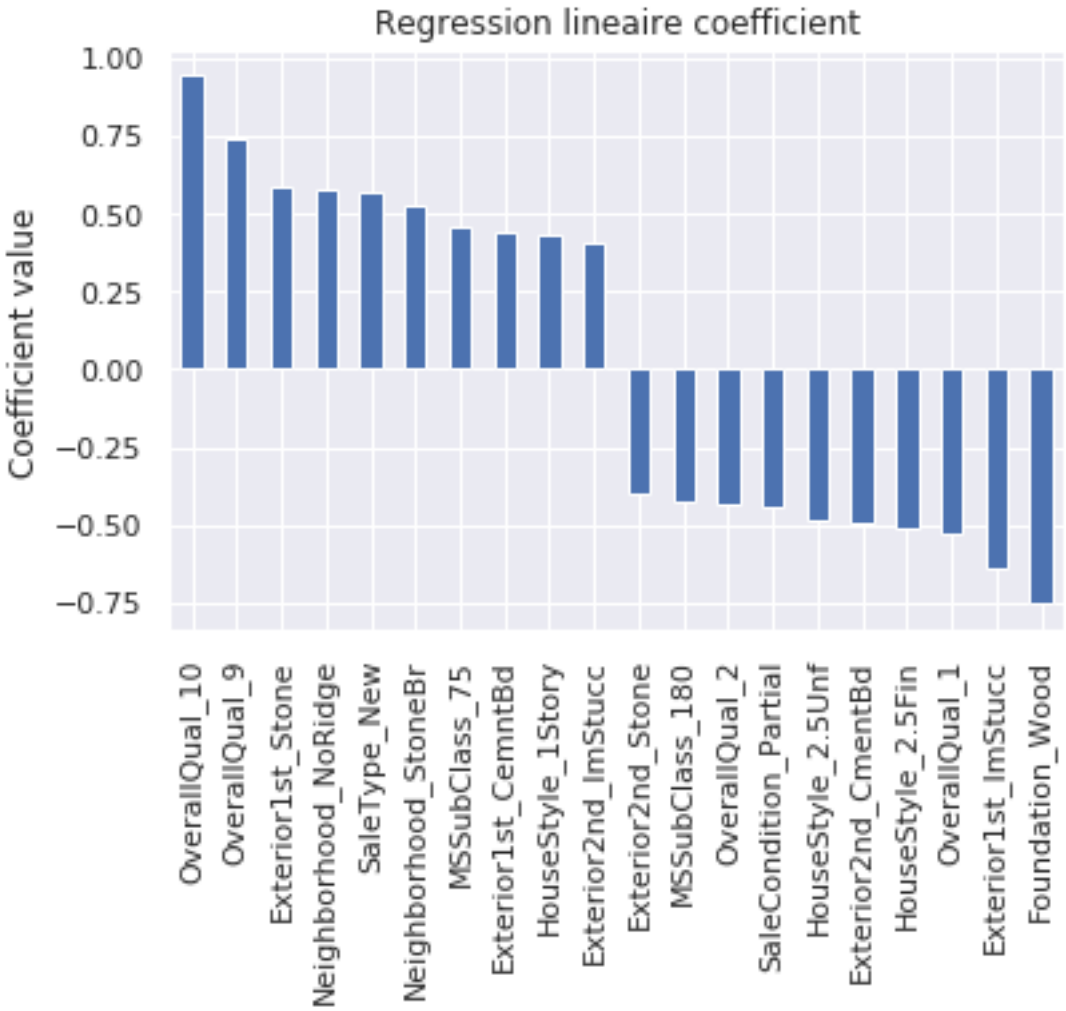
[44]:
```
coefficients = pd.Series(clfRegLinear.named_steps["LinearRegression"].coef_.
 ↪flatten(), index=feature_names).sort_values(ascending=False)
coefficients
```

[44]:
```
OverallQual_10          0.940247
OverallQual_9           0.742062
Exterior1st_Stone       0.582463
Neighborhood_NoRidge    0.574580
SaleType_New            0.570265
                             ...
Exterior2nd_CmentBd    -0.490410
HouseStyle_2.5Fin      -0.510151
OverallQual_1          -0.528082
Exterior1st_ImStucc    -0.640948
Foundation_Wood        -0.752301
Length: 234, dtype: float64
```

[45]:
```
print("ordonnee à l'origine : " + str(clfRegLinear.
 ↪named_steps["LinearRegression"].intercept_))
```

```
ordonnee à l'origine : -0.19533366583177567
```

[46]:
```
coefficients[np.abs(coefficients)>0.4].plot(kind="bar")
plt.title("Regression lineaire coefficient")
plt.ylabel("Coefficient value")
plt.show()
```



On observe une cohérence par rapport notamment aux échelles de notation (OverallQual) qui joue de manière positive ou négative sur le prix. On observe aussi qu'être dans les districts NoRidge, StoneBr, NridgHt à un impact positif, des fondations en pierre ou un extérieur en pierre joue de manière positive dans le prix contrairement à des fondations en bois

7.1.3 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

[0]:
```
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])
```

[48]:
```
print("Regression metrics for train data")
print(regression_metrics(Yscaler.inverse_transform(y_train_standardise), Yscaler.
 ↪inverse_transform(y_trainPredict)))
print("Regression metrics for test data")
print(regression_metrics(Yscaler.inverse_transform(y_test_standardise), Yscaler.
 ↪inverse_transform(y_testPredict)))
```

```
Regression metrics for train data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  268648.274025         16821.411538        7.489736e+08  0.890839
Regression metrics for test data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  170230.331078          19058.74088        7.289663e+08  0.852876
```

On retrouve la bonne généralisation quelque soit la métrique utilisée. Si on regarde en moyenne l'erreur absolu, on se trompe de l'ordre entre 17000 et 19000. Néanmoins, on peut aussi constater qu'avec le modèle de régression linéaire généralisée, on peut se tromper fortement sur l'estimation du prix > 170000.

7.2 Un modèle linéaire Lasso avec une régularisation

7.2.1 Regression Lasso

Ce modèle intègre en plus un terme de régularisation L1 sur la régression linéaire et force par conséquent un certain nombre de coefficients à être à 0.

[49]:
```
from sklearn.linear_model import Lasso
bestScore = 0
bestAlpha = 0
for alpha in [0.00001, 0.0001, 0.001,0.01, 0.1, 1, 10, 100, 1000 ]:
```

```
    print("Lasso avec regul : " + str(alpha))
    clfLasso = Pipeline(steps=[('preprocessor', preprocessor),␣
 ↪('LassoRegression', Lasso(alpha=alpha))])
    clfLasso.fit(X_train,y_train_standardise)
    #y_trainPredict = clfLasso.predict(X_train)
    #y_testPredict = clfLasso.predict(X_test)
    scoreTest = clfLasso.score(X_test, y_test_standardise)
    print("model score sur le train :  %.3f" % clfLasso.score(X_train,␣
 ↪y_train_standardise))
    print("model score le test : %.3f" % scoreTest)
    if( scoreTest > bestScore):
        bestScore = scoreTest
        bestAlpha = alpha
print("\n\n\n-----------------------------------------------\nLasso best score :␣
 ↪%.3f" % bestScore)
print("Lasso best regularization : " + str(bestAlpha))
```

```
Lasso avec regul : 1e-05
```

```
/usr/local/lib/python3.6/dist-
packages/sklearn/linear_model/coordinate_descent.py:459: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 10.973060312336798, tolerance: 0.10220000000000003
  max_iter, tol, rng, random, positive)
```

```
model score sur le train :  0.891
model score le test : 0.855
Lasso avec regul : 0.0001
model score sur le train :  0.890
model score le test : 0.864
Lasso avec regul : 0.001
model score sur le train :  0.882
model score le test : 0.887
Lasso avec regul : 0.01
model score sur le train :  0.826
model score le test : 0.871
Lasso avec regul : 0.1
model score sur le train :  0.707
model score le test : 0.775
Lasso avec regul : 1
model score sur le train :  0.000
model score le test : -0.017
Lasso avec regul : 10
model score sur le train :  0.000
model score le test : -0.017
Lasso avec regul : 100
model score sur le train :  0.000
model score le test : -0.017
```

```
Lasso avec regul : 1000
model score sur le train :  0.000
model score le test : -0.017
```

```
------------------------------------------------
Lasso best score : 0.887
Lasso best regularization : 0.001
```

Une régularisation importante conduit à des résultats surprenants qui peuvent s'expliquer peut-être par la standardisation de la variable y (sans la standardisation de celle-ci, les coefficients étaient beaucoup plus grands et le paramètre de régularisation optimale était de 100).

7.2.2 Coefficients de la régression Lasso

[50]:
```
clfLasso = Pipeline(steps=[('preprocessor', preprocessor), ('LassoRegression',␣
 ↪Lasso(alpha=0.001))])
clfLasso.fit(X_train,y_train_standardise)
y_trainPredictLasso = clfLasso.predict(X_train)
y_testPredictLasso = clfLasso.predict(X_test)
print("model score sur le train :  %.3f" % clfLasso.score(X_train,␣
 ↪y_train_standardise))
print("model score le test : %.3f" % clfLasso.score(X_test, y_test_standardise))

coefficientsLasso = pd.Series(clfLasso.named_steps["LassoRegression"].coef_.
 ↪flatten(), index=feature_names).sort_values(ascending=False)
print("\nCoefficients totaux : "+ str(len(coefficientsLasso)))
print("Dont coefficients nuls : "+ str(sum(coefficientsLasso == 0)))
```

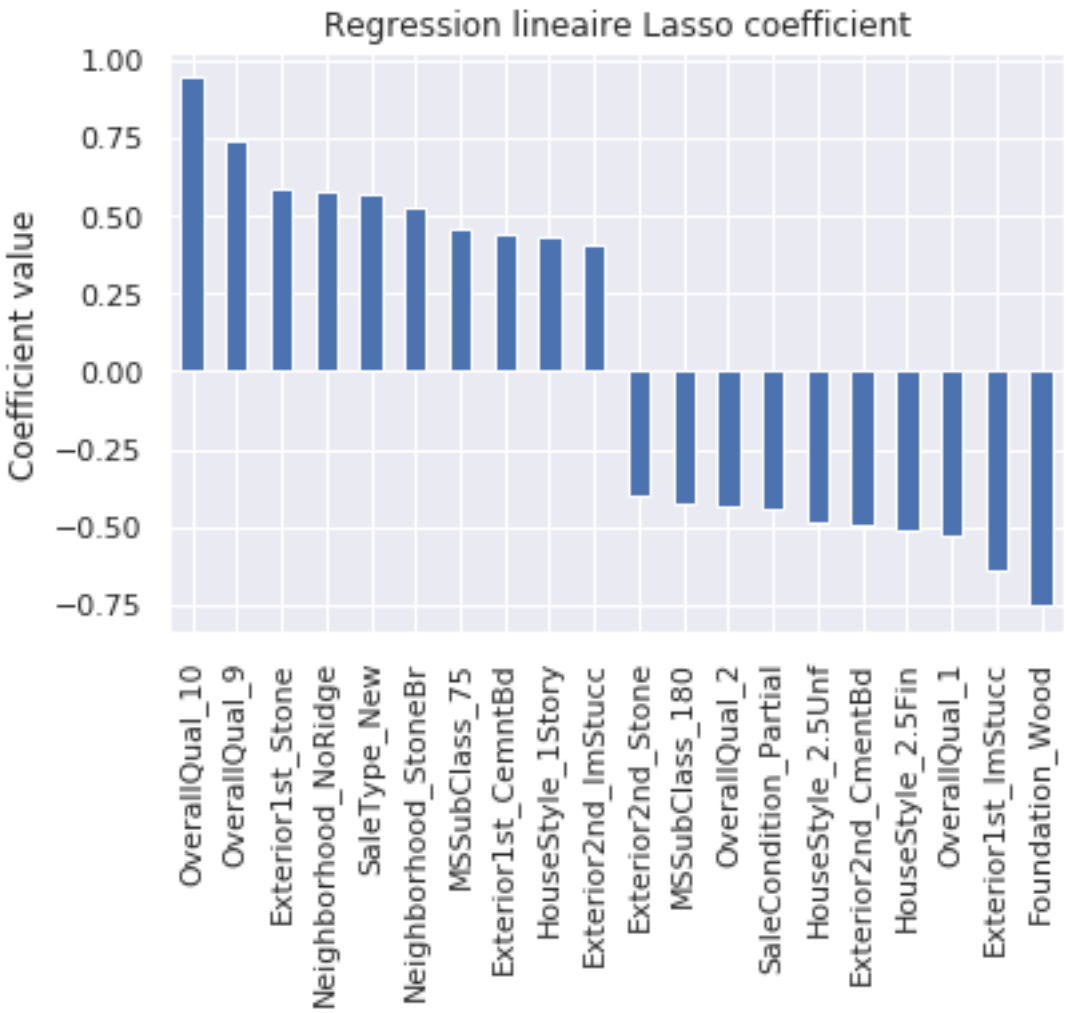
```
model score sur le train :  0.882
model score le test : 0.887

Coefficients totaux : 234
Dont coefficients nuls : 111
```

[51]:
```
coefficientsLasso
```

[51]:
```
OverallQual_10          0.900954
OverallQual_9           0.781774
Neighborhood_NoRidge    0.573501
Neighborhood_StoneBr    0.479460
Neighborhood_NridgHt    0.364574
                            ...
MSSubClass_120         -0.146227
BsmtQual_No Basement   -0.151211
Neighborhood_Edwards   -0.250958
MSSubClass_160         -0.257290
```

```
LotShape_IR3            -0.350952
Length: 234, dtype: float64
```

[52]:
```
coefficients[np.abs(coefficients)>0.4].plot(kind="bar")
plt.title("Regression lineaire Lasso coefficient")
plt.ylabel("Coefficient value")
plt.show()
```



7.2.3 Evaluation de la régression Lasso avec différentes métriques

[53]:
```
print("Regression metrics for train data")
print(regression_metrics(Yscaler.inverse_transform(y_train_standardise), Yscaler.
 ↪inverse_transform(y_trainPredictLasso)))
print("Regression metrics for test data")
```

```
print(regression_metrics(Yscaler.inverse_transform(y_test_standardise), Yscaler.
 ↪inverse_transform(y_testPredictLasso)))
```

```
Regression metrics for train data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  309318.171409         16624.623402        8.089878e+08  0.882092
Regression metrics for test data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  150555.331076         16747.844669        5.614283e+08  0.886689
```

7.3 Transformation log-normale de la variable cible et regression Lasso

- L'analyse exploratoire de la variable cible avait permis d'identifier une distribution de type log-normale.
- La référence suivante : https://scikit-learn.org/stable/auto_examples/compose/plot_transformed_target. expose des bénéfices dans le cadre d'une régression linéaire en effectuant une transformation préalable sur la cible.

[54]:
```
from sklearn.linear_model import Lasso
bestScore = 0
bestAlpha = 0
for alpha in [0.00001, 0.0001, 0.001,0.01, 0.1, 1, 10, 100, 1000 ]:
    print("Lasso avec regul : " + str(alpha))
    clfLassoTrans = Pipeline(steps=[('preprocessor', preprocessor),␣
 ↪('LassoRegression', Lasso(alpha=alpha))])
    clfLassoTrans.fit(X_train,np.log1p(y_train))
    scoreTest = clfLassoTrans.score(X_test, np.log1p(y_test))
    print("model score sur le train :  %.3f" % clfLassoTrans.score(X_train, np.
 ↪log1p(y_train)))
    print("model score le test : %.3f" % scoreTest)
    if( scoreTest > bestScore):
        bestScore = scoreTest
        bestAlpha = alpha
print("\n\n\n-----------------------------------------------\nLasso best score :␣
 ↪%.3f" % bestScore)
print("Lasso best regularization : " + str(bestAlpha))
```

```
Lasso avec regul : 1e-05
```

```
/usr/local/lib/python3.6/dist-
packages/sklearn/linear_model/coordinate_descent.py:459: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 0.04932461910538066, tolerance: 0.01695086798037297
  max_iter, tol, rng, random, positive)
```

```
model score sur le train :  0.917
model score le test : 0.879
Lasso avec regul : 0.0001
```

```
model score sur le train :  0.915
model score le test : 0.886
Lasso avec regul : 0.001
model score sur le train :  0.898
model score le test : 0.890
Lasso avec regul : 0.01
model score sur le train :  0.807
model score le test : 0.828
Lasso avec regul : 0.1
model score sur le train :  0.653
model score le test : 0.662
Lasso avec regul : 1
model score sur le train :  0.000
model score le test : -0.010
Lasso avec regul : 10
model score sur le train :  0.000
model score le test : -0.010
Lasso avec regul : 100
model score sur le train :  0.000
model score le test : -0.010
Lasso avec regul : 1000
model score sur le train :  0.000
model score le test : -0.010
```

```
------------------------------------------------
Lasso best score : 0.890
Lasso best regularization : 0.001
```

Dans le cas présent, nous n'observons pas un apport significatif sur le R2.

7.4 Conclusion sur les régressions linéaires

- Les régressions linéaires généralisées et Lasso réalisées mettent en avant les mêmes variables explicatives et ont des scores assez similaires avec des coefficients de détermination correctes.
- Suivant la métrique à considérer, un modèle pourra être mis en avant. Néanmoins, la régression Lasso met presque la moitié des coefficients à nul ce qui facilite aussi l'interprétation des résultats.
- Néanmoins il apparait assez surprenant de ne pas retrouver dans les variables prépondérantes les variables numériques comme GrLivArea

[55]:
```
("Reg Linéaire : ", coefficients["GrLivArea"], "Reg Lasso : ",␣
↪coefficientsLasso["GrLivArea"])
```

[55]:
```
('Reg Linéaire : ', 0.16389762960965606, 'Reg Lasso : ', 0.1889702781769011)
```

#Réseau de neurones

7.5 Librairies nécessaires

[56]:
```
!pip install git+https://github.com/tensorflow/docs
%tensorflow_version 2.x
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import tensorflow_docs.modeling

print(tf.__version__)
```

```
Collecting git+https://github.com/tensorflow/docs
  Cloning https://github.com/tensorflow/docs to /tmp/pip-req-build-zxjfuv2t
  Running command git clone -q https://github.com/tensorflow/docs /tmp/pip-req-
build-zxjfuv2t
Requirement already satisfied (use --upgrade to upgrade): tensorflow-docs==0.0.0
from git+https://github.com/tensorflow/docs in /usr/local/lib/python3.6/dist-
packages
Requirement already satisfied: astor in /usr/local/lib/python3.6/dist-packages
(from tensorflow-docs==0.0.0) (0.8.1)
Requirement already satisfied: absl-py in /usr/local/lib/python3.6/dist-packages
(from tensorflow-docs==0.0.0) (0.8.1)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from tensorflow-docs==0.0.0) (1.12.0)
Requirement already satisfied: pathlib2 in /usr/local/lib/python3.6/dist-
packages (from tensorflow-docs==0.0.0) (2.3.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.6/dist-packages
(from tensorflow-docs==0.0.0) (3.13)
Building wheels for collected packages: tensorflow-docs
  Building wheel for tensorflow-docs (setup.py) ... done
  Created wheel for tensorflow-docs: filename=tensorflow_docs-0.0.0-cp36-none-
any.whl size=80660
sha256=9dca8779937cfdf278dfb67e1e0207cb0b7b851dab1e8ad5808f8c33ed49652a
  Stored in directory: /tmp/pip-ephem-wheel-cache-6dy3tygn/wheels/eb/1b/35/fce87
697be00d2fc63e0b4b395b0d9c7e391a10e98d9a0d97f
Successfully built tensorflow-docs
TensorFlow 2.x selected.
2.1.0-rc1
```

7.6 Préprocessing sur les données

On réutilise les transformations précédentes sur les variables numériques et catégorielles : * pour les variables catégorielles : OneHotEncoding et imputation des valeurs manquantes * pour les variables numériques : recalibration entre 0 et 1 et imputation des valeurs manquantes par la moyenne.

[57]:
```
print("shape de X_train : " + str(X_train.shape))
print("shape de X_test : " + str(X_test.shape))
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)
print("shape de X_train_transformed : "+ str(X_train_transformed.shape))
print("shape de X_test_transformed : "+ str(X_test_transformed.shape))
```

```
shape de X_train : (1022, 57)
shape de X_test : (438, 57)
shape de X_train_transformed : (1022, 234)
shape de X_test_transformed : (438, 234)
```

La target à savoir le prix de vente est aussi entre 0 et 1 car nous avions déjà appliqué un processus de standardisation et cela peut avoir de l'importance car non seulement, cela facilite la phase d'apprentissage mais peut aussi donner des résultats meilleurs dans le modèle de régression avec un réseau de neurones.

[58]:
```
dimInput = X_train_transformed.shape[1]
print("Dimension de la couche visible du réseau :", dimInput)
```

```
Dimension de la couche visible du réseau : 234
```

7.7 Architectures du réseau de neurones

[0]:
```
def build_model(unitsLayer1 =64, unitsLayer2 = 64, dropOut=False):
  model = keras.Sequential()
  model.add(layers.Dense(unitsLayer1, activation='relu', input_shape=[dimInput]))
  if(dropOut):
    model.add(layers.Dropout(0.3))
  model.add(layers.Dense(unitsLayer2, activation='relu'))
  if(dropOut):
    model.add(layers.Dropout(0.3))
  model.add(layers.Dense(1, activation="linear"))



  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
  return model
```

[0]:
```
def build_model_4Layers(unitsLayer1 = 32, unitsLayer2 = 16, unitsLayer3 = 8,␣
↪unitsLayer4 = 4, dropOut=False):
  model = keras.Sequential()
  model.add(layers.Dense(unitsLayer1, activation='relu', input_shape=[dimInput]))
  if(dropOut):
```

```
    model.add(layers.Dropout(0.05))
  model.add(layers.Dense(unitsLayer2, activation='relu'))
  if(dropOut):
    model.add(layers.Dropout(0.05))
  model.add(layers.Dense(unitsLayer3, activation='relu'))
  if(dropOut):
    model.add(layers.Dropout(0.05))
  model.add(layers.Dense(unitsLayer4, activation='relu'))
  if(dropOut):
    model.add(layers.Dropout(0.05))
  model.add(layers.Dense(1, activation="linear"))

  optimizer = tf.keras.optimizers.RMSprop(0.001)

  model.compile(loss='mse',
                optimizer=optimizer,
                metrics=['mae', 'mse'])
  return model
```

7.7.1 Modèle avec 2 couches cachées et 64 neurones

[0]:
```
model_64_64_1 = build_model()
```

[62]:
```
model_64_64_1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 64)                15040
_________________________________________________________________
dense_1 (Dense)              (None, 64)                4160
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 19,265
Trainable params: 19,265
Non-trainable params: 0
_________________________________________________________________
```

La fonction summary permet d'inspecter le réseau de neurones et de voir le nombre de paramètres associés à chaque couche. Par exemple 15040 correspond au 234 features + 1 pour le biais multipliés par 64 (le nombre de neurones de la première couche cachée).

[63]:
```
model_64_64_1_DropOut = build_model(dropOut=True)
model_64_64_1_DropOut.summary()
```

```
Model: "sequential_1"
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_3 (Dense)              (None, 64)                15040
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_4 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_5 (Dense)              (None, 1)                 65
=================================================================
Total params: 19,265
Trainable params: 19,265
Non-trainable params: 0
_________________________________________________________________
```

Initialement : on s'est rendu compte après la phase d'entrainement d'un overfitting important. On initialise la même architecture mais en utilisant la fonctionnalité dropOut pour désactiver 30% des neurones pendant la phase d'entrainement de manière aléatoire dans les couches cachées.

7.7.2 Modèle avec 2 couches cachées et 32 neurones

[64]:
```
model_32_32_1 = build_model(unitsLayer1=32, unitsLayer2 =32, dropOut=False)
model_32_32_1.summary()
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_6 (Dense)              (None, 32)                7520
_________________________________________________________________
dense_7 (Dense)              (None, 32)                1056
_________________________________________________________________
dense_8 (Dense)              (None, 1)                 33
=================================================================
Total params: 8,609
Trainable params: 8,609
Non-trainable params: 0
_________________________________________________________________
```

[65]:
```
model_32_32_1_DropOut = build_model(unitsLayer1=32, unitsLayer2 =32,dropOut=True)
model_32_32_1_DropOut.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

```
=================================================================
dense_9 (Dense)              (None, 32)                7520
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_10 (Dense)             (None, 32)                1056
_________________________________________________________________
dropout_3 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 33
=================================================================
Total params: 8,609
Trainable params: 8,609
Non-trainable params: 0
_________________________________________________________________
```

7.7.3 Modèle avec 4 couches cachées

[66]:
```
model_32_16_8_4_1 = build_model_4Layers()
model_32_16_8_4_1.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 32)                7520
_________________________________________________________________
dense_13 (Dense)             (None, 16)                528
_________________________________________________________________
dense_14 (Dense)             (None, 8)                 136
_________________________________________________________________
dense_15 (Dense)             (None, 4)                 36
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 5
=================================================================
Total params: 8,225
Trainable params: 8,225
Non-trainable params: 0
_________________________________________________________________
```

[67]:
```
model_32_16_8_4_1_DropOut = build_model_4Layers(dropOut=True)
model_32_16_8_4_1_DropOut.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_17 (Dense)             (None, 32)                7520
```

```
_________________________________________________________________
dropout_4 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_18 (Dense)             (None, 16)                528
_________________________________________________________________
dropout_5 (Dropout)          (None, 16)                0
_________________________________________________________________
dense_19 (Dense)             (None, 8)                 136
_________________________________________________________________
dropout_6 (Dropout)          (None, 8)                 0
_________________________________________________________________
dense_20 (Dense)             (None, 4)                 36
_________________________________________________________________
dropout_7 (Dropout)          (None, 4)                 0
_________________________________________________________________
dense_21 (Dense)             (None, 1)                 5
=================================================================
Total params: 8,225
Trainable params: 8,225
Non-trainable params: 0
_________________________________________________________________
```

7.8 Entrainement et évaluation des réseaux de neurones

[0]:
```
def TrainingNetwork(model, networkName, callback=False):
  # train the model
  import time
  start_time = time.time()
  if(callback):
    print("Training model network " + networkName + " with Callback")
  else :
    print("Training model network " + networkName)

  if(callback):
    # The patience parameter is the amount of epochs to check for improvement
    early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)
    model.fit(X_train_transformed.toarray(), y_train_standardise,␣
↪validation_data=(X_test_transformed.toarray(), y_test_standardise),␣
↪epochs=200, batch_size=32, callbacks=[early_stop])
  else:
    model.fit(X_train_transformed.toarray(), y_train_standardise,␣
↪validation_data=(X_test_transformed.toarray(), y_test_standardise),␣
↪epochs=200, batch_size=32)
  print("Temps d'entrainement %s seconds ---" % (time.time() - start_time))
```

[0]:
```
def PredictionAndEvaluationNetwork(model, networkName, callback=False):
  # make predictions on the testing data
```

```
  if(callback):
    print("Predicting house prices model network "+ networkName + " with␣
↪Callback")
  else :
    print("Predicting house prices model network "+ networkName)
  y_testPredict = model.predict(X_test_transformed.toarray())
  y_trainPredict = model.predict(X_train_transformed.toarray())
  print("Regression metrics for train data")
  print(regression_metrics(Yscaler.inverse_transform(y_train_standardise),␣
↪Yscaler.inverse_transform(y_trainPredict)))
  print("Regression metrics for test data")
  print(regression_metrics(Yscaler.inverse_transform(y_test_standardise),␣
↪Yscaler.inverse_transform(y_testPredict)))
```

7.8.1 Modèle avec 2 couches cachées et 64 neurones

[70]:
```
TrainingNetwork(model_64_64_1, str(64) +" * " + str(64) + " * 1")
PredictionAndEvaluationNetwork(model_64_64_1, str(64) +" * " + str(64) + " * 1")
```

```
Training model network 64 * 64 * 1
Train on 1022 samples, validate on 438 samples
Epoch 1/200
1022/1022 [==============================] - 1s 949us/sample - loss: 0.3757 -
mae: 0.3772 - mse: 0.3757 - val_loss: 0.1200 - val_mae: 0.2604 - val_mse: 0.1200
Epoch 2/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.2314 -
mae: 0.2646 - mse: 0.2314 - val_loss: 0.0944 - val_mae: 0.2266 - val_mse: 0.0944
Epoch 3/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.1791 -
mae: 0.2256 - mse: 0.1791 - val_loss: 0.1134 - val_mae: 0.2311 - val_mse: 0.1134
Epoch 4/200
1022/1022 [==============================] - 0s 160us/sample - loss: 0.1393 -
mae: 0.2064 - mse: 0.1393 - val_loss: 0.0797 - val_mae: 0.2035 - val_mse: 0.0797
Epoch 5/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.1221 -
mae: 0.1861 - mse: 0.1221 - val_loss: 0.0872 - val_mae: 0.2135 - val_mse: 0.0872
Epoch 6/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0863 -
mae: 0.1648 - mse: 0.0863 - val_loss: 0.1516 - val_mae: 0.2921 - val_mse: 0.1516
Epoch 7/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0817 -
mae: 0.1675 - mse: 0.0817 - val_loss: 0.0945 - val_mae: 0.2217 - val_mse: 0.0945
Epoch 8/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0683 -
mae: 0.1477 - mse: 0.0683 - val_loss: 0.0866 - val_mae: 0.2051 - val_mse: 0.0866
Epoch 9/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0519 -
```

```
mae: 0.1423 - mse: 0.0519 - val_loss: 0.0917 - val_mae: 0.2164 - val_mse: 0.0917
Epoch 10/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0411 -
mae: 0.1337 - mse: 0.0411 - val_loss: 0.0848 - val_mae: 0.2009 - val_mse: 0.0848
Epoch 11/200
1022/1022 [==============================] - 0s 192us/sample - loss: 0.0390 -
mae: 0.1292 - mse: 0.0390 - val_loss: 0.0936 - val_mae: 0.2139 - val_mse: 0.0936
Epoch 12/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.0288 -
mae: 0.1173 - mse: 0.0288 - val_loss: 0.1016 - val_mae: 0.2188 - val_mse: 0.1016
Epoch 13/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0267 -
mae: 0.1121 - mse: 0.0267 - val_loss: 0.0925 - val_mae: 0.2091 - val_mse: 0.0925
Epoch 14/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0246 -
mae: 0.1103 - mse: 0.0246 - val_loss: 0.0917 - val_mae: 0.2087 - val_mse: 0.0917
Epoch 15/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0234 -
mae: 0.1047 - mse: 0.0234 - val_loss: 0.0863 - val_mae: 0.2011 - val_mse: 0.0863
Epoch 16/200
1022/1022 [==============================] - 0s 182us/sample - loss: 0.0204 -
mae: 0.0970 - mse: 0.0204 - val_loss: 0.1111 - val_mae: 0.2390 - val_mse: 0.1111
Epoch 17/200
1022/1022 [==============================] - 0s 162us/sample - loss: 0.0235 -
mae: 0.1086 - mse: 0.0235 - val_loss: 0.0843 - val_mae: 0.1998 - val_mse: 0.0843
Epoch 18/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0192 -
mae: 0.0986 - mse: 0.0192 - val_loss: 0.0998 - val_mae: 0.2145 - val_mse: 0.0998
Epoch 19/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0183 -
mae: 0.0961 - mse: 0.0183 - val_loss: 0.0870 - val_mae: 0.2007 - val_mse: 0.0870
Epoch 20/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0131 -
mae: 0.0831 - mse: 0.0131 - val_loss: 0.1186 - val_mae: 0.2461 - val_mse: 0.1186
Epoch 21/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0201 -
mae: 0.0941 - mse: 0.0201 - val_loss: 0.0975 - val_mae: 0.2135 - val_mse: 0.0975
Epoch 22/200
1022/1022 [==============================] - 0s 182us/sample - loss: 0.0133 -
mae: 0.0832 - mse: 0.0133 - val_loss: 0.0972 - val_mae: 0.2178 - val_mse: 0.0972
Epoch 23/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.0153 -
mae: 0.0869 - mse: 0.0153 - val_loss: 0.0971 - val_mae: 0.2064 - val_mse: 0.0971
Epoch 24/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0123 -
mae: 0.0815 - mse: 0.0123 - val_loss: 0.0971 - val_mae: 0.2169 - val_mse: 0.0971
Epoch 25/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0162 -
```

```
mae: 0.0878 - mse: 0.0162 - val_loss: 0.1293 - val_mae: 0.2523 - val_mse: 0.1293
Epoch 26/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0126 -
mae: 0.0797 - mse: 0.0126 - val_loss: 0.0976 - val_mae: 0.2135 - val_mse: 0.0976
Epoch 27/200
1022/1022 [==============================] - 0s 161us/sample - loss: 0.0120 -
mae: 0.0732 - mse: 0.0120 - val_loss: 0.1187 - val_mae: 0.2530 - val_mse: 0.1187
Epoch 28/200
1022/1022 [==============================] - 0s 188us/sample - loss: 0.0126 -
mae: 0.0811 - mse: 0.0126 - val_loss: 0.0932 - val_mae: 0.2085 - val_mse: 0.0932
Epoch 29/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0103 -
mae: 0.0724 - mse: 0.0103 - val_loss: 0.0938 - val_mae: 0.2119 - val_mse: 0.0938
Epoch 30/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0126 -
mae: 0.0773 - mse: 0.0126 - val_loss: 0.0965 - val_mae: 0.2060 - val_mse: 0.0965
Epoch 31/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0107 -
mae: 0.0745 - mse: 0.0107 - val_loss: 0.0886 - val_mae: 0.2014 - val_mse: 0.0886
Epoch 32/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0125 -
mae: 0.0774 - mse: 0.0125 - val_loss: 0.0899 - val_mae: 0.2024 - val_mse: 0.0899
Epoch 33/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0097 -
mae: 0.0699 - mse: 0.0097 - val_loss: 0.0955 - val_mae: 0.2114 - val_mse: 0.0955
Epoch 34/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0097 -
mae: 0.0732 - mse: 0.0097 - val_loss: 0.0988 - val_mae: 0.2123 - val_mse: 0.0988
Epoch 35/200
1022/1022 [==============================] - 0s 162us/sample - loss: 0.0104 -
mae: 0.0731 - mse: 0.0104 - val_loss: 0.0985 - val_mae: 0.2133 - val_mse: 0.0985
Epoch 36/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0087 -
mae: 0.0682 - mse: 0.0087 - val_loss: 0.0978 - val_mae: 0.2067 - val_mse: 0.0978
Epoch 37/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0114 -
mae: 0.0722 - mse: 0.0114 - val_loss: 0.0980 - val_mae: 0.2096 - val_mse: 0.0980
Epoch 38/200
1022/1022 [==============================] - 0s 181us/sample - loss: 0.0099 -
mae: 0.0729 - mse: 0.0099 - val_loss: 0.0900 - val_mae: 0.2056 - val_mse: 0.0900
Epoch 39/200
1022/1022 [==============================] - 0s 189us/sample - loss: 0.0101 -
mae: 0.0714 - mse: 0.0101 - val_loss: 0.1044 - val_mae: 0.2124 - val_mse: 0.1044
Epoch 40/200
1022/1022 [==============================] - 0s 191us/sample - loss: 0.0082 -
mae: 0.0648 - mse: 0.0082 - val_loss: 0.0889 - val_mae: 0.2069 - val_mse: 0.0889
Epoch 41/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0107 -
```

```
mae: 0.0671 - mse: 0.0107 - val_loss: 0.0989 - val_mae: 0.2127 - val_mse: 0.0989
Epoch 42/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.0095 -
mae: 0.0684 - mse: 0.0095 - val_loss: 0.0917 - val_mae: 0.2044 - val_mse: 0.0917
Epoch 43/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0106 -
mae: 0.0690 - mse: 0.0106 - val_loss: 0.0950 - val_mae: 0.2060 - val_mse: 0.0950
Epoch 44/200
1022/1022 [==============================] - 0s 161us/sample - loss: 0.0077 -
mae: 0.0629 - mse: 0.0077 - val_loss: 0.0930 - val_mae: 0.2096 - val_mse: 0.0930
Epoch 45/200
1022/1022 [==============================] - 0s 181us/sample - loss: 0.0095 -
mae: 0.0668 - mse: 0.0095 - val_loss: 0.0909 - val_mae: 0.2037 - val_mse: 0.0909
Epoch 46/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0088 -
mae: 0.0659 - mse: 0.0088 - val_loss: 0.1014 - val_mae: 0.2213 - val_mse: 0.1014
Epoch 47/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0088 -
mae: 0.0624 - mse: 0.0088 - val_loss: 0.1027 - val_mae: 0.2179 - val_mse: 0.1027
Epoch 48/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0084 -
mae: 0.0658 - mse: 0.0084 - val_loss: 0.1007 - val_mae: 0.2181 - val_mse: 0.1007
Epoch 49/200
1022/1022 [==============================] - 0s 161us/sample - loss: 0.0059 -
mae: 0.0558 - mse: 0.0059 - val_loss: 0.0917 - val_mae: 0.2061 - val_mse: 0.0917
Epoch 50/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0100 -
mae: 0.0625 - mse: 0.0100 - val_loss: 0.0938 - val_mae: 0.2061 - val_mse: 0.0938
Epoch 51/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.0077 -
mae: 0.0628 - mse: 0.0077 - val_loss: 0.0980 - val_mae: 0.2092 - val_mse: 0.0980
Epoch 52/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0105 -
mae: 0.0697 - mse: 0.0105 - val_loss: 0.0993 - val_mae: 0.2130 - val_mse: 0.0993
Epoch 53/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0061 -
mae: 0.0536 - mse: 0.0061 - val_loss: 0.0980 - val_mae: 0.2092 - val_mse: 0.0980
Epoch 54/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0089 -
mae: 0.0679 - mse: 0.0089 - val_loss: 0.1000 - val_mae: 0.2149 - val_mse: 0.1000
Epoch 55/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0064 -
mae: 0.0546 - mse: 0.0064 - val_loss: 0.1020 - val_mae: 0.2201 - val_mse: 0.1020
Epoch 56/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0073 -
mae: 0.0608 - mse: 0.0073 - val_loss: 0.1061 - val_mae: 0.2176 - val_mse: 0.1061
Epoch 57/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.0093 -
```

```
mae: 0.0633 - mse: 0.0093 - val_loss: 0.1001 - val_mae: 0.2093 - val_mse: 0.1001
Epoch 58/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0068 -
mae: 0.0585 - mse: 0.0068 - val_loss: 0.0884 - val_mae: 0.2052 - val_mse: 0.0884
Epoch 59/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0073 -
mae: 0.0574 - mse: 0.0073 - val_loss: 0.0973 - val_mae: 0.2125 - val_mse: 0.0973
Epoch 60/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0075 -
mae: 0.0600 - mse: 0.0075 - val_loss: 0.0930 - val_mae: 0.2060 - val_mse: 0.0930
Epoch 61/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0079 -
mae: 0.0561 - mse: 0.0079 - val_loss: 0.0957 - val_mae: 0.2066 - val_mse: 0.0957
Epoch 62/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0060 -
mae: 0.0543 - mse: 0.0060 - val_loss: 0.0976 - val_mae: 0.2113 - val_mse: 0.0976
Epoch 63/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.0086 -
mae: 0.0582 - mse: 0.0086 - val_loss: 0.1011 - val_mae: 0.2132 - val_mse: 0.1011
Epoch 64/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0057 -
mae: 0.0526 - mse: 0.0057 - val_loss: 0.0941 - val_mae: 0.2086 - val_mse: 0.0941
Epoch 65/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0080 -
mae: 0.0587 - mse: 0.0080 - val_loss: 0.0974 - val_mae: 0.2082 - val_mse: 0.0974
Epoch 66/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0052 -
mae: 0.0530 - mse: 0.0052 - val_loss: 0.0892 - val_mae: 0.2036 - val_mse: 0.0892
Epoch 67/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0074 -
mae: 0.0597 - mse: 0.0074 - val_loss: 0.0974 - val_mae: 0.2120 - val_mse: 0.0974
Epoch 68/200
1022/1022 [==============================] - 0s 185us/sample - loss: 0.0069 -
mae: 0.0580 - mse: 0.0069 - val_loss: 0.0926 - val_mae: 0.2041 - val_mse: 0.0926
Epoch 69/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.0068 -
mae: 0.0544 - mse: 0.0068 - val_loss: 0.0949 - val_mae: 0.2059 - val_mse: 0.0949
Epoch 70/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0073 -
mae: 0.0558 - mse: 0.0073 - val_loss: 0.0924 - val_mae: 0.2035 - val_mse: 0.0924
Epoch 71/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0057 -
mae: 0.0539 - mse: 0.0057 - val_loss: 0.1041 - val_mae: 0.2156 - val_mse: 0.1041
Epoch 72/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0072 -
mae: 0.0585 - mse: 0.0072 - val_loss: 0.0965 - val_mae: 0.2085 - val_mse: 0.0965
Epoch 73/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0058 -
```

```
mae: 0.0518 - mse: 0.0058 - val_loss: 0.1004 - val_mae: 0.2147 - val_mse: 0.1004
Epoch 74/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0055 -
mae: 0.0520 - mse: 0.0055 - val_loss: 0.0970 - val_mae: 0.2110 - val_mse: 0.0970
Epoch 75/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0069 -
mae: 0.0547 - mse: 0.0069 - val_loss: 0.1084 - val_mae: 0.2150 - val_mse: 0.1084
Epoch 76/200
1022/1022 [==============================] - 0s 161us/sample - loss: 0.0069 -
mae: 0.0586 - mse: 0.0069 - val_loss: 0.0977 - val_mae: 0.2069 - val_mse: 0.0977
Epoch 77/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0059 -
mae: 0.0527 - mse: 0.0059 - val_loss: 0.0983 - val_mae: 0.2146 - val_mse: 0.0983
Epoch 78/200
1022/1022 [==============================] - 0s 162us/sample - loss: 0.0070 -
mae: 0.0543 - mse: 0.0070 - val_loss: 0.0993 - val_mae: 0.2088 - val_mse: 0.0993
Epoch 79/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0055 -
mae: 0.0488 - mse: 0.0055 - val_loss: 0.0949 - val_mae: 0.2066 - val_mse: 0.0949
Epoch 80/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0056 -
mae: 0.0477 - mse: 0.0056 - val_loss: 0.0977 - val_mae: 0.2101 - val_mse: 0.0977
Epoch 81/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0057 -
mae: 0.0537 - mse: 0.0057 - val_loss: 0.0919 - val_mae: 0.2065 - val_mse: 0.0919
Epoch 82/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0066 -
mae: 0.0567 - mse: 0.0066 - val_loss: 0.0923 - val_mae: 0.2049 - val_mse: 0.0923
Epoch 83/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0069 -
mae: 0.0498 - mse: 0.0069 - val_loss: 0.0907 - val_mae: 0.2031 - val_mse: 0.0907
Epoch 84/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0054 -
mae: 0.0482 - mse: 0.0054 - val_loss: 0.0958 - val_mae: 0.2088 - val_mse: 0.0958
Epoch 85/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0047 -
mae: 0.0484 - mse: 0.0047 - val_loss: 0.0966 - val_mae: 0.2087 - val_mse: 0.0966
Epoch 86/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.0075 -
mae: 0.0560 - mse: 0.0075 - val_loss: 0.0936 - val_mae: 0.2041 - val_mse: 0.0936
Epoch 87/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0041 -
mae: 0.0455 - mse: 0.0041 - val_loss: 0.1054 - val_mae: 0.2179 - val_mse: 0.1054
Epoch 88/200
1022/1022 [==============================] - 0s 162us/sample - loss: 0.0048 -
mae: 0.0487 - mse: 0.0048 - val_loss: 0.1044 - val_mae: 0.2206 - val_mse: 0.1044
Epoch 89/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0051 -
```

```
mae: 0.0510 - mse: 0.0051 - val_loss: 0.1049 - val_mae: 0.2115 - val_mse: 0.1049
Epoch 90/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0058 -
mae: 0.0506 - mse: 0.0058 - val_loss: 0.0947 - val_mae: 0.2069 - val_mse: 0.0947
Epoch 91/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0072 -
mae: 0.0490 - mse: 0.0072 - val_loss: 0.1000 - val_mae: 0.2105 - val_mse: 0.1000
Epoch 92/200
1022/1022 [==============================] - 0s 186us/sample - loss: 0.0049 -
mae: 0.0452 - mse: 0.0049 - val_loss: 0.1060 - val_mae: 0.2194 - val_mse: 0.1060
Epoch 93/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0056 -
mae: 0.0493 - mse: 0.0056 - val_loss: 0.1014 - val_mae: 0.2147 - val_mse: 0.1014
Epoch 94/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.0055 -
mae: 0.0502 - mse: 0.0055 - val_loss: 0.1010 - val_mae: 0.2138 - val_mse: 0.1010
Epoch 95/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0052 -
mae: 0.0496 - mse: 0.0052 - val_loss: 0.1004 - val_mae: 0.2090 - val_mse: 0.1004
Epoch 96/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.0049 -
mae: 0.0497 - mse: 0.0049 - val_loss: 0.0927 - val_mae: 0.2064 - val_mse: 0.0927
Epoch 97/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0047 -
mae: 0.0478 - mse: 0.0047 - val_loss: 0.0928 - val_mae: 0.2029 - val_mse: 0.0928
Epoch 98/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0049 -
mae: 0.0516 - mse: 0.0049 - val_loss: 0.0967 - val_mae: 0.2049 - val_mse: 0.0967
Epoch 99/200
1022/1022 [==============================] - 0s 158us/sample - loss: 0.0050 -
mae: 0.0492 - mse: 0.0050 - val_loss: 0.1050 - val_mae: 0.2159 - val_mse: 0.1050
Epoch 100/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0064 -
mae: 0.0482 - mse: 0.0064 - val_loss: 0.0958 - val_mae: 0.2070 - val_mse: 0.0958
Epoch 101/200
1022/1022 [==============================] - 0s 192us/sample - loss: 0.0037 -
mae: 0.0436 - mse: 0.0037 - val_loss: 0.0957 - val_mae: 0.2052 - val_mse: 0.0957
Epoch 102/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.0048 -
mae: 0.0474 - mse: 0.0048 - val_loss: 0.0949 - val_mae: 0.2080 - val_mse: 0.0949
Epoch 103/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0049 -
mae: 0.0479 - mse: 0.0049 - val_loss: 0.1022 - val_mae: 0.2125 - val_mse: 0.1022
Epoch 104/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0041 -
mae: 0.0469 - mse: 0.0041 - val_loss: 0.0920 - val_mae: 0.2018 - val_mse: 0.0920
Epoch 105/200
1022/1022 [==============================] - 0s 161us/sample - loss: 0.0052 -
```

```
mae: 0.0490 - mse: 0.0052 - val_loss: 0.0952 - val_mae: 0.2048 - val_mse: 0.0952
Epoch 106/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0055 -
mae: 0.0487 - mse: 0.0055 - val_loss: 0.0961 - val_mae: 0.2047 - val_mse: 0.0961
Epoch 107/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0038 -
mae: 0.0440 - mse: 0.0038 - val_loss: 0.0970 - val_mae: 0.2044 - val_mse: 0.0970
Epoch 108/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0043 -
mae: 0.0455 - mse: 0.0043 - val_loss: 0.0934 - val_mae: 0.2056 - val_mse: 0.0934
Epoch 109/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.0046 -
mae: 0.0478 - mse: 0.0046 - val_loss: 0.0926 - val_mae: 0.2069 - val_mse: 0.0926
Epoch 110/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0040 -
mae: 0.0425 - mse: 0.0040 - val_loss: 0.1162 - val_mae: 0.2340 - val_mse: 0.1162
Epoch 111/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0049 -
mae: 0.0485 - mse: 0.0049 - val_loss: 0.0920 - val_mae: 0.2012 - val_mse: 0.0920
Epoch 112/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0055 -
mae: 0.0482 - mse: 0.0055 - val_loss: 0.0963 - val_mae: 0.2074 - val_mse: 0.0963
Epoch 113/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.0043 -
mae: 0.0465 - mse: 0.0043 - val_loss: 0.0964 - val_mae: 0.2071 - val_mse: 0.0964
Epoch 114/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0049 -
mae: 0.0437 - mse: 0.0049 - val_loss: 0.0931 - val_mae: 0.2050 - val_mse: 0.0931
Epoch 115/200
1022/1022 [==============================] - 0s 189us/sample - loss: 0.0037 -
mae: 0.0443 - mse: 0.0037 - val_loss: 0.0928 - val_mae: 0.2108 - val_mse: 0.0928
Epoch 116/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0055 -
mae: 0.0472 - mse: 0.0055 - val_loss: 0.0935 - val_mae: 0.2048 - val_mse: 0.0935
Epoch 117/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0037 -
mae: 0.0427 - mse: 0.0037 - val_loss: 0.1007 - val_mae: 0.2136 - val_mse: 0.1007
Epoch 118/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.0042 -
mae: 0.0466 - mse: 0.0042 - val_loss: 0.0892 - val_mae: 0.1998 - val_mse: 0.0892
Epoch 119/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.0062 -
mae: 0.0481 - mse: 0.0062 - val_loss: 0.1053 - val_mae: 0.2135 - val_mse: 0.1053
Epoch 120/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0032 -
mae: 0.0396 - mse: 0.0032 - val_loss: 0.0914 - val_mae: 0.2043 - val_mse: 0.0914
Epoch 121/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0039 -
```

```
mae: 0.0447 - mse: 0.0039 - val_loss: 0.0918 - val_mae: 0.1988 - val_mse: 0.0918
Epoch 122/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.0050 -
mae: 0.0458 - mse: 0.0050 - val_loss: 0.0921 - val_mae: 0.2011 - val_mse: 0.0921
Epoch 123/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0052 -
mae: 0.0459 - mse: 0.0052 - val_loss: 0.0935 - val_mae: 0.2009 - val_mse: 0.0935
Epoch 124/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.0037 -
mae: 0.0429 - mse: 0.0037 - val_loss: 0.0909 - val_mae: 0.2017 - val_mse: 0.0909
Epoch 125/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0039 -
mae: 0.0428 - mse: 0.0039 - val_loss: 0.0999 - val_mae: 0.2092 - val_mse: 0.0999
Epoch 126/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0039 -
mae: 0.0425 - mse: 0.0039 - val_loss: 0.0886 - val_mae: 0.1995 - val_mse: 0.0886
Epoch 127/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0045 -
mae: 0.0486 - mse: 0.0045 - val_loss: 0.0996 - val_mae: 0.2137 - val_mse: 0.0996
Epoch 128/200
1022/1022 [==============================] - 0s 162us/sample - loss: 0.0038 -
mae: 0.0417 - mse: 0.0038 - val_loss: 0.0907 - val_mae: 0.2006 - val_mse: 0.0907
Epoch 129/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0040 -
mae: 0.0424 - mse: 0.0040 - val_loss: 0.0897 - val_mae: 0.2023 - val_mse: 0.0897
Epoch 130/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0053 -
mae: 0.0428 - mse: 0.0053 - val_loss: 0.0947 - val_mae: 0.2043 - val_mse: 0.0947
Epoch 131/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0040 -
mae: 0.0419 - mse: 0.0040 - val_loss: 0.0903 - val_mae: 0.2017 - val_mse: 0.0903
Epoch 132/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.0039 -
mae: 0.0378 - mse: 0.0039 - val_loss: 0.0932 - val_mae: 0.2019 - val_mse: 0.0932
Epoch 133/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0042 -
mae: 0.0413 - mse: 0.0042 - val_loss: 0.0915 - val_mae: 0.2041 - val_mse: 0.0915
Epoch 134/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0035 -
mae: 0.0415 - mse: 0.0035 - val_loss: 0.0991 - val_mae: 0.2062 - val_mse: 0.0991
Epoch 135/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0039 -
mae: 0.0408 - mse: 0.0039 - val_loss: 0.0904 - val_mae: 0.1999 - val_mse: 0.0904
Epoch 136/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0049 -
mae: 0.0445 - mse: 0.0049 - val_loss: 0.0947 - val_mae: 0.2081 - val_mse: 0.0947
Epoch 137/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0040 -
```

```
mae: 0.0436 - mse: 0.0040 - val_loss: 0.0876 - val_mae: 0.1993 - val_mse: 0.0876
Epoch 138/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.0035 -
mae: 0.0379 - mse: 0.0035 - val_loss: 0.0873 - val_mae: 0.1982 - val_mse: 0.0873
Epoch 139/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.0035 -
mae: 0.0418 - mse: 0.0035 - val_loss: 0.0917 - val_mae: 0.2042 - val_mse: 0.0917
Epoch 140/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0040 -
mae: 0.0424 - mse: 0.0040 - val_loss: 0.0957 - val_mae: 0.2053 - val_mse: 0.0957
Epoch 141/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0039 -
mae: 0.0431 - mse: 0.0039 - val_loss: 0.0901 - val_mae: 0.2036 - val_mse: 0.0901
Epoch 142/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0046 -
mae: 0.0458 - mse: 0.0046 - val_loss: 0.0895 - val_mae: 0.2020 - val_mse: 0.0895
Epoch 143/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0033 -
mae: 0.0395 - mse: 0.0033 - val_loss: 0.0902 - val_mae: 0.2017 - val_mse: 0.0902
Epoch 144/200
1022/1022 [==============================] - 0s 190us/sample - loss: 0.0036 -
mae: 0.0377 - mse: 0.0036 - val_loss: 0.0912 - val_mae: 0.2023 - val_mse: 0.0912
Epoch 145/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0042 -
mae: 0.0430 - mse: 0.0042 - val_loss: 0.0926 - val_mae: 0.2035 - val_mse: 0.0926
Epoch 146/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0049 -
mae: 0.0406 - mse: 0.0049 - val_loss: 0.0875 - val_mae: 0.1976 - val_mse: 0.0875
Epoch 147/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0037 -
mae: 0.0420 - mse: 0.0037 - val_loss: 0.0930 - val_mae: 0.2063 - val_mse: 0.0930
Epoch 148/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0046 -
mae: 0.0393 - mse: 0.0046 - val_loss: 0.0915 - val_mae: 0.2028 - val_mse: 0.0915
Epoch 149/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0038 -
mae: 0.0409 - mse: 0.0038 - val_loss: 0.0914 - val_mae: 0.2047 - val_mse: 0.0914
Epoch 150/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.0034 -
mae: 0.0421 - mse: 0.0034 - val_loss: 0.0901 - val_mae: 0.2048 - val_mse: 0.0901
Epoch 151/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0042 -
mae: 0.0419 - mse: 0.0042 - val_loss: 0.0931 - val_mae: 0.2025 - val_mse: 0.0931
Epoch 152/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0038 -
mae: 0.0414 - mse: 0.0038 - val_loss: 0.0883 - val_mae: 0.1988 - val_mse: 0.0883
Epoch 153/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0034 -
```

```
mae: 0.0408 - mse: 0.0034 - val_loss: 0.0866 - val_mae: 0.2016 - val_mse: 0.0866
Epoch 154/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0040 -
mae: 0.0421 - mse: 0.0040 - val_loss: 0.0856 - val_mae: 0.2016 - val_mse: 0.0856
Epoch 155/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.0032 -
mae: 0.0349 - mse: 0.0032 - val_loss: 0.0877 - val_mae: 0.1993 - val_mse: 0.0877
Epoch 156/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.0040 -
mae: 0.0453 - mse: 0.0040 - val_loss: 0.0895 - val_mae: 0.2038 - val_mse: 0.0895
Epoch 157/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0031 -
mae: 0.0383 - mse: 0.0031 - val_loss: 0.0931 - val_mae: 0.2087 - val_mse: 0.0931
Epoch 158/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0047 -
mae: 0.0424 - mse: 0.0047 - val_loss: 0.0872 - val_mae: 0.1970 - val_mse: 0.0872
Epoch 159/200
1022/1022 [==============================] - 0s 195us/sample - loss: 0.0039 -
mae: 0.0394 - mse: 0.0039 - val_loss: 0.0920 - val_mae: 0.2000 - val_mse: 0.0920
Epoch 160/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0034 -
mae: 0.0376 - mse: 0.0034 - val_loss: 0.0949 - val_mae: 0.2077 - val_mse: 0.0949
Epoch 161/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.0033 -
mae: 0.0400 - mse: 0.0033 - val_loss: 0.0878 - val_mae: 0.1971 - val_mse: 0.0878
Epoch 162/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0039 -
mae: 0.0401 - mse: 0.0039 - val_loss: 0.0871 - val_mae: 0.1980 - val_mse: 0.0871
Epoch 163/200
1022/1022 [==============================] - 0s 165us/sample - loss: 0.0036 -
mae: 0.0377 - mse: 0.0036 - val_loss: 0.0886 - val_mae: 0.1997 - val_mse: 0.0886
Epoch 164/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0033 -
mae: 0.0369 - mse: 0.0033 - val_loss: 0.0929 - val_mae: 0.2049 - val_mse: 0.0929
Epoch 165/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.0034 -
mae: 0.0406 - mse: 0.0034 - val_loss: 0.0867 - val_mae: 0.1989 - val_mse: 0.0867
Epoch 166/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.0044 -
mae: 0.0366 - mse: 0.0044 - val_loss: 0.0924 - val_mae: 0.1998 - val_mse: 0.0924
Epoch 167/200
1022/1022 [==============================] - 0s 192us/sample - loss: 0.0036 -
mae: 0.0391 - mse: 0.0036 - val_loss: 0.0893 - val_mae: 0.2012 - val_mse: 0.0893
Epoch 168/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.0031 -
mae: 0.0367 - mse: 0.0031 - val_loss: 0.0928 - val_mae: 0.2027 - val_mse: 0.0928
Epoch 169/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.0026 -
```

```
mae: 0.0364 - mse: 0.0026 - val_loss: 0.0817 - val_mae: 0.1918 - val_mse: 0.0817
Epoch 170/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0039 -
mae: 0.0407 - mse: 0.0039 - val_loss: 0.0946 - val_mae: 0.2011 - val_mse: 0.0946
Epoch 171/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.0035 -
mae: 0.0389 - mse: 0.0035 - val_loss: 0.0891 - val_mae: 0.1983 - val_mse: 0.0891
Epoch 172/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.0040 -
mae: 0.0362 - mse: 0.0040 - val_loss: 0.0923 - val_mae: 0.2029 - val_mse: 0.0923
Epoch 173/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0031 -
mae: 0.0390 - mse: 0.0031 - val_loss: 0.0856 - val_mae: 0.1936 - val_mse: 0.0856
Epoch 174/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0031 -
mae: 0.0393 - mse: 0.0031 - val_loss: 0.0915 - val_mae: 0.2015 - val_mse: 0.0915
Epoch 175/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.0036 -
mae: 0.0382 - mse: 0.0036 - val_loss: 0.0901 - val_mae: 0.2045 - val_mse: 0.0901
Epoch 176/200
1022/1022 [==============================] - 0s 159us/sample - loss: 0.0036 -
mae: 0.0378 - mse: 0.0036 - val_loss: 0.0914 - val_mae: 0.2032 - val_mse: 0.0914
Epoch 177/200
1022/1022 [==============================] - 0s 162us/sample - loss: 0.0042 -
mae: 0.0386 - mse: 0.0042 - val_loss: 0.0925 - val_mae: 0.1999 - val_mse: 0.0925
Epoch 178/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.0029 -
mae: 0.0356 - mse: 0.0029 - val_loss: 0.1031 - val_mae: 0.2068 - val_mse: 0.1031
Epoch 179/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0033 -
mae: 0.0386 - mse: 0.0033 - val_loss: 0.0865 - val_mae: 0.1942 - val_mse: 0.0865
Epoch 180/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0026 -
mae: 0.0359 - mse: 0.0026 - val_loss: 0.0883 - val_mae: 0.1995 - val_mse: 0.0883
Epoch 181/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.0038 -
mae: 0.0395 - mse: 0.0038 - val_loss: 0.0882 - val_mae: 0.1996 - val_mse: 0.0882
Epoch 182/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0035 -
mae: 0.0405 - mse: 0.0035 - val_loss: 0.0921 - val_mae: 0.1987 - val_mse: 0.0921
Epoch 183/200
1022/1022 [==============================] - 0s 161us/sample - loss: 0.0039 -
mae: 0.0376 - mse: 0.0039 - val_loss: 0.0864 - val_mae: 0.1964 - val_mse: 0.0864
Epoch 184/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0030 -
mae: 0.0381 - mse: 0.0030 - val_loss: 0.0906 - val_mae: 0.1974 - val_mse: 0.0906
Epoch 185/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0034 -
```

```
mae: 0.0361 - mse: 0.0034 - val_loss: 0.0910 - val_mae: 0.2021 - val_mse: 0.0910
Epoch 186/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0034 -
mae: 0.0386 - mse: 0.0034 - val_loss: 0.0910 - val_mae: 0.1988 - val_mse: 0.0910
Epoch 187/200
1022/1022 [==============================] - 0s 168us/sample - loss: 0.0031 -
mae: 0.0351 - mse: 0.0031 - val_loss: 0.0911 - val_mae: 0.2012 - val_mse: 0.0911
Epoch 188/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0031 -
mae: 0.0368 - mse: 0.0031 - val_loss: 0.0953 - val_mae: 0.2058 - val_mse: 0.0953
Epoch 189/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0029 -
mae: 0.0387 - mse: 0.0029 - val_loss: 0.0879 - val_mae: 0.1985 - val_mse: 0.0879
Epoch 190/200
1022/1022 [==============================] - 0s 185us/sample - loss: 0.0032 -
mae: 0.0387 - mse: 0.0032 - val_loss: 0.0849 - val_mae: 0.1955 - val_mse: 0.0849
Epoch 191/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0040 -
mae: 0.0360 - mse: 0.0040 - val_loss: 0.0838 - val_mae: 0.1949 - val_mse: 0.0838
Epoch 192/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0026 -
mae: 0.0340 - mse: 0.0026 - val_loss: 0.0936 - val_mae: 0.2011 - val_mse: 0.0936
Epoch 193/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0037 -
mae: 0.0396 - mse: 0.0037 - val_loss: 0.0842 - val_mae: 0.1913 - val_mse: 0.0842
Epoch 194/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0028 -
mae: 0.0329 - mse: 0.0028 - val_loss: 0.0933 - val_mae: 0.2082 - val_mse: 0.0933
Epoch 195/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0031 -
mae: 0.0386 - mse: 0.0031 - val_loss: 0.0887 - val_mae: 0.1969 - val_mse: 0.0887
Epoch 196/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0034 -
mae: 0.0380 - mse: 0.0034 - val_loss: 0.0863 - val_mae: 0.1960 - val_mse: 0.0863
Epoch 197/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0030 -
mae: 0.0374 - mse: 0.0030 - val_loss: 0.0908 - val_mae: 0.1970 - val_mse: 0.0908
Epoch 198/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0028 -
mae: 0.0360 - mse: 0.0028 - val_loss: 0.0918 - val_mae: 0.2052 - val_mse: 0.0918
Epoch 199/200
1022/1022 [==============================] - 0s 166us/sample - loss: 0.0038 -
mae: 0.0385 - mse: 0.0038 - val_loss: 0.0887 - val_mae: 0.1998 - val_mse: 0.0887
Epoch 200/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.0027 -
mae: 0.0340 - mse: 0.0027 - val_loss: 0.0940 - val_mae: 0.2060 - val_mse: 0.0940
Temps d'entrainement 36.28351807594299 seconds ---
Predicting house prices model network 64 * 64 * 1
```

```
Regression metrics for train data
   max_error  mean_absolute_error  mean_squared_error  r2_score
0    31057.0          3945.486966        2.788137e+07  0.995936
Regression metrics for test data
      max_error  mean_absolute_error  mean_squared_error  r2_score
0  192880.96875         17065.007393        6.447108e+08  0.869881
```

On observe dans le cas présent un overfitting important sur le train par rapport à l'ensemble de validation. Une petite optimisation pour éviter de réaliser tous epoch est de mettre en place un callback et regarder si la fonction val_loss ne s'améliore pas de manière significative pendant une certaine période.

[71]:
```
TrainingNetwork(model_64_64_1, str(64) +" * " + str(64) + " * 1", callback=True)
PredictionAndEvaluationNetwork(model_64_64_1, str(64) +" * " + str(64) + " * 1",␣
 ↪callback=True)
```

```
Training model network 64 * 64 * 1 with Callback
Train on 1022 samples, validate on 438 samples
Epoch 1/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0034 -
mae: 0.0390 - mse: 0.0034 - val_loss: 0.0915 - val_mae: 0.1959 - val_mse: 0.0915
Epoch 2/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0029 -
mae: 0.0366 - mse: 0.0029 - val_loss: 0.0872 - val_mae: 0.1994 - val_mse: 0.0872
Epoch 3/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.0027 -
mae: 0.0351 - mse: 0.0027 - val_loss: 0.0955 - val_mae: 0.2024 - val_mse: 0.0955
Epoch 4/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0024 -
mae: 0.0343 - mse: 0.0024 - val_loss: 0.0843 - val_mae: 0.1931 - val_mse: 0.0843
Epoch 5/200
1022/1022 [==============================] - 0s 163us/sample - loss: 0.0033 -
mae: 0.0382 - mse: 0.0033 - val_loss: 0.0883 - val_mae: 0.1962 - val_mse: 0.0883
Epoch 6/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.0025 -
mae: 0.0332 - mse: 0.0025 - val_loss: 0.0856 - val_mae: 0.1953 - val_mse: 0.0856
Epoch 7/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0037 -
mae: 0.0396 - mse: 0.0037 - val_loss: 0.0918 - val_mae: 0.2022 - val_mse: 0.0918
Epoch 8/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0027 -
mae: 0.0333 - mse: 0.0027 - val_loss: 0.0849 - val_mae: 0.1925 - val_mse: 0.0849
Epoch 9/200
1022/1022 [==============================] - 0s 167us/sample - loss: 0.0039 -
mae: 0.0359 - mse: 0.0039 - val_loss: 0.0870 - val_mae: 0.1938 - val_mse: 0.0870
Epoch 10/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.0025 -
mae: 0.0339 - mse: 0.0025 - val_loss: 0.0864 - val_mae: 0.1951 - val_mse: 0.0864
```

```
Epoch 11/200
1022/1022 [==============================] - 0s 182us/sample - loss: 0.0033 -
mae: 0.0359 - mse: 0.0033 - val_loss: 0.0872 - val_mae: 0.1950 - val_mse: 0.0872
Epoch 12/200
1022/1022 [==============================] - 0s 192us/sample - loss: 0.0028 -
mae: 0.0334 - mse: 0.0028 - val_loss: 0.0883 - val_mae: 0.1969 - val_mse: 0.0883
Epoch 13/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.0032 -
mae: 0.0354 - mse: 0.0032 - val_loss: 0.0978 - val_mae: 0.1994 - val_mse: 0.0978
Epoch 14/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.0030 -
mae: 0.0347 - mse: 0.0030 - val_loss: 0.0895 - val_mae: 0.1966 - val_mse: 0.0895
Temps d'entrainement 2.5942599773406982 seconds ---
Predicting house prices model network 64 * 64 * 1 with Callback
Regression metrics for train data
   max_error  mean_absolute_error  mean_squared_error  r2_score
0    62764.5          3073.917415        2.455586e+07  0.996421
Regression metrics for test data
      max_error  mean_absolute_error  mean_squared_error  r2_score
0  189215.03125         16285.356967        6.143990e+08  0.875998
```

On observe toujours un overfitting important avec cette architecture malgré la mise en place du callback qui a juste limité le nombre d'epoch dans le cas présent mais aurait pu dans d'autres cas limiter le surapprentissage.

[72]:
```
print("Prise en compte du dropout dans le training dans les couches cachées avec␣
 ↪rate = 30%")
TrainingNetwork(model_64_64_1_DropOut, str(64) +" * " + str(64) + " * 1",␣
 ↪callback=True)
PredictionAndEvaluationNetwork(model_64_64_1_DropOut, str(64) +" * " + str(64) +␣
 ↪" * 1", callback=True)
```

```
Prise en compte du dropout dans le training dans les couches cachées avec rate =
30%
Training model network 64 * 64 * 1 with Callback
Train on 1022 samples, validate on 438 samples
Epoch 1/200
1022/1022 [==============================] - 1s 773us/sample - loss: 0.7352 -
mae: 0.5811 - mse: 0.7352 - val_loss: 0.2076 - val_mae: 0.3024 - val_mse: 0.2076
Epoch 2/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.4011 -
mae: 0.4478 - mse: 0.4011 - val_loss: 0.1332 - val_mae: 0.2491 - val_mse: 0.1332
Epoch 3/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.3434 -
mae: 0.3867 - mse: 0.3434 - val_loss: 0.1065 - val_mae: 0.2265 - val_mse: 0.1065
Epoch 4/200
1022/1022 [==============================] - 0s 194us/sample - loss: 0.3045 -
mae: 0.3535 - mse: 0.3045 - val_loss: 0.0919 - val_mae: 0.2106 - val_mse: 0.0919
```

```
Epoch 5/200
1022/1022 [==============================] - 0s 190us/sample - loss: 0.2610 -
mae: 0.3227 - mse: 0.2610 - val_loss: 0.0974 - val_mae: 0.2069 - val_mse: 0.0974
Epoch 6/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.2269 -
mae: 0.3131 - mse: 0.2269 - val_loss: 0.0887 - val_mae: 0.2029 - val_mse: 0.0887
Epoch 7/200
1022/1022 [==============================] - 0s 196us/sample - loss: 0.2098 -
mae: 0.2882 - mse: 0.2098 - val_loss: 0.0922 - val_mae: 0.2208 - val_mse: 0.0922
Epoch 8/200
1022/1022 [==============================] - 0s 181us/sample - loss: 0.2163 -
mae: 0.2831 - mse: 0.2163 - val_loss: 0.0957 - val_mae: 0.2237 - val_mse: 0.0957
Epoch 9/200
1022/1022 [==============================] - 0s 202us/sample - loss: 0.2979 -
mae: 0.2850 - mse: 0.2979 - val_loss: 0.0870 - val_mae: 0.2098 - val_mse: 0.0870
Epoch 10/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.1909 -
mae: 0.2693 - mse: 0.1909 - val_loss: 0.1099 - val_mae: 0.2303 - val_mse: 0.1099
Epoch 11/200
1022/1022 [==============================] - 0s 182us/sample - loss: 0.2174 -
mae: 0.2643 - mse: 0.2174 - val_loss: 0.0974 - val_mae: 0.2183 - val_mse: 0.0974
Epoch 12/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.2203 -
mae: 0.2630 - mse: 0.2203 - val_loss: 0.0885 - val_mae: 0.2094 - val_mse: 0.0885
Epoch 13/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.1978 -
mae: 0.2517 - mse: 0.1978 - val_loss: 0.0958 - val_mae: 0.2191 - val_mse: 0.0958
Epoch 14/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.1602 -
mae: 0.2390 - mse: 0.1602 - val_loss: 0.0881 - val_mae: 0.2014 - val_mse: 0.0881
Epoch 15/200
1022/1022 [==============================] - 0s 190us/sample - loss: 0.2045 -
mae: 0.2486 - mse: 0.2045 - val_loss: 0.0958 - val_mae: 0.2250 - val_mse: 0.0958
Epoch 16/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.1321 -
mae: 0.2308 - mse: 0.1321 - val_loss: 0.0960 - val_mae: 0.2229 - val_mse: 0.0960
Epoch 17/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.1378 -
mae: 0.2221 - mse: 0.1378 - val_loss: 0.0920 - val_mae: 0.2032 - val_mse: 0.0920
Epoch 18/200
1022/1022 [==============================] - 0s 187us/sample - loss: 0.1667 -
mae: 0.2339 - mse: 0.1667 - val_loss: 0.0941 - val_mae: 0.2151 - val_mse: 0.0941
Epoch 19/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.1599 -
mae: 0.2194 - mse: 0.1599 - val_loss: 0.1014 - val_mae: 0.2198 - val_mse: 0.1014
Temps d'entrainement 4.274690389633179 seconds ---
Predicting house prices model network 64 * 64 * 1 with Callback
Regression metrics for train data
```

```
      max_error  mean_absolute_error  mean_squared_error  r2_score
0  299434.15625         15739.528215        6.489125e+08  0.905422
Regression metrics for test data
     max_error  mean_absolute_error  mean_squared_error  r2_score
0  191663.4375         18206.392159        6.957507e+08  0.859579
```

On observe toujours un peu de surapprentissage mais de manière moindre grâce au dropout. Je me suis inspiré pour partie : https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/

7.8.2 Modèle avec 2 couches cachées et 32 neurones

[73]:
```
TrainingNetwork(model_32_32_1, str(32) +" * " + str(32) + " * 1", callback=True)
PredictionAndEvaluationNetwork(model_32_32_1, str(32) +" * " + str(32) + " * 1",␣
 ↪callback=True)
```

```
Training model network 32 * 32 * 1 with Callback
Train on 1022 samples, validate on 438 samples
Epoch 1/200
1022/1022 [==============================] - 1s 678us/sample - loss: 0.4628 -
mae: 0.4572 - mse: 0.4628 - val_loss: 0.1318 - val_mae: 0.2716 - val_mse: 0.1318
Epoch 2/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.2513 -
mae: 0.2847 - mse: 0.2513 - val_loss: 0.1061 - val_mae: 0.2377 - val_mse: 0.1061
Epoch 3/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.2014 -
mae: 0.2475 - mse: 0.2014 - val_loss: 0.0937 - val_mae: 0.2185 - val_mse: 0.0937
Epoch 4/200
1022/1022 [==============================] - 0s 193us/sample - loss: 0.1624 -
mae: 0.2177 - mse: 0.1624 - val_loss: 0.0930 - val_mae: 0.2138 - val_mse: 0.0930
Epoch 5/200
1022/1022 [==============================] - 0s 171us/sample - loss: 0.1440 -
mae: 0.2009 - mse: 0.1440 - val_loss: 0.0890 - val_mae: 0.2163 - val_mse: 0.0890
Epoch 6/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.1180 -
mae: 0.1878 - mse: 0.1180 - val_loss: 0.0849 - val_mae: 0.2074 - val_mse: 0.0849
Epoch 7/200
1022/1022 [==============================] - 0s 189us/sample - loss: 0.1004 -
mae: 0.1761 - mse: 0.1004 - val_loss: 0.1043 - val_mae: 0.2165 - val_mse: 0.1043
Epoch 8/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0873 -
mae: 0.1629 - mse: 0.0873 - val_loss: 0.0871 - val_mae: 0.2010 - val_mse: 0.0871
Epoch 9/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.0654 -
mae: 0.1520 - mse: 0.0654 - val_loss: 0.0885 - val_mae: 0.2109 - val_mse: 0.0885
Epoch 10/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.0679 -
mae: 0.1468 - mse: 0.0679 - val_loss: 0.0889 - val_mae: 0.2108 - val_mse: 0.0889
```

```
Epoch 11/200
1022/1022 [==============================] - 0s 173us/sample - loss: 0.0522 -
mae: 0.1371 - mse: 0.0522 - val_loss: 0.0831 - val_mae: 0.1991 - val_mse: 0.0831
Epoch 12/200
1022/1022 [==============================] - 0s 193us/sample - loss: 0.0454 -
mae: 0.1328 - mse: 0.0454 - val_loss: 0.0836 - val_mae: 0.2024 - val_mse: 0.0836
Epoch 13/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.0416 -
mae: 0.1249 - mse: 0.0416 - val_loss: 0.0858 - val_mae: 0.2003 - val_mse: 0.0858
Epoch 14/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.0314 -
mae: 0.1185 - mse: 0.0314 - val_loss: 0.0894 - val_mae: 0.2106 - val_mse: 0.0894
Epoch 15/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.0307 -
mae: 0.1130 - mse: 0.0307 - val_loss: 0.0889 - val_mae: 0.2041 - val_mse: 0.0889
Epoch 16/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.0232 -
mae: 0.1015 - mse: 0.0232 - val_loss: 0.0873 - val_mae: 0.1998 - val_mse: 0.0873
Epoch 17/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.0231 -
mae: 0.1037 - mse: 0.0231 - val_loss: 0.0945 - val_mae: 0.2105 - val_mse: 0.0945
Epoch 18/200
1022/1022 [==============================] - 0s 207us/sample - loss: 0.0223 -
mae: 0.0989 - mse: 0.0223 - val_loss: 0.0921 - val_mae: 0.2107 - val_mse: 0.0921
Epoch 19/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.0195 -
mae: 0.0993 - mse: 0.0195 - val_loss: 0.0893 - val_mae: 0.2064 - val_mse: 0.0893
Epoch 20/200
1022/1022 [==============================] - 0s 181us/sample - loss: 0.0178 -
mae: 0.0893 - mse: 0.0178 - val_loss: 0.0907 - val_mae: 0.2015 - val_mse: 0.0907
Epoch 21/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.0178 -
mae: 0.0887 - mse: 0.0178 - val_loss: 0.0961 - val_mae: 0.2080 - val_mse: 0.0961
Temps d'entrainement 4.498281717300415 seconds ---
Predicting house prices model network 32 * 32 * 1 with Callback
Regression metrics for train data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  118479.363281          6370.498869        9.677748e+07  0.985895
Regression metrics for test data
       max_error  mean_absolute_error  mean_squared_error  r2_score
0  184567.796875         17229.033765        6.590638e+08  0.866984
```

On a toujours un surpapprentissage et des différences non significatives par rapport à l'architecture 64 * 64 sur l'ensemble de validation. A noter que le modèle a deux fois moins de paramètres par rapport au réseau avec 32 neurones.

[74]:
```
print("Prise en compte du dropout dans le training dans les couches cachées avec␣
 ↪rate = 30%")
```

```
TrainingNetwork(model_32_32_1_DropOut, str(32) +" * " + str(32) + " *␣
 ↪1",callback=True)
PredictionAndEvaluationNetwork(model_32_32_1_DropOut, str(32) +" * " + str(32) +␣
 ↪" * 1",callback=True)
```

```
Prise en compte du dropout dans le training dans les couches cachées avec rate =
30%
Training model network 32 * 32 * 1 with Callback
Train on 1022 samples, validate on 438 samples
Epoch 1/200
1022/1022 [==============================] - 1s 729us/sample - loss: 0.8166 -
mae: 0.6107 - mse: 0.8166 - val_loss: 0.2369 - val_mae: 0.3411 - val_mse: 0.2369
Epoch 2/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.5648 -
mae: 0.4788 - mse: 0.5648 - val_loss: 0.1683 - val_mae: 0.2781 - val_mse: 0.1683
Epoch 3/200
1022/1022 [==============================] - 0s 185us/sample - loss: 0.4306 -
mae: 0.4169 - mse: 0.4306 - val_loss: 0.1523 - val_mae: 0.2660 - val_mse: 0.1523
Epoch 4/200
1022/1022 [==============================] - 0s 192us/sample - loss: 0.4249 -
mae: 0.3921 - mse: 0.4249 - val_loss: 0.1379 - val_mae: 0.2545 - val_mse: 0.1379
Epoch 5/200
1022/1022 [==============================] - 0s 176us/sample - loss: 0.3384 -
mae: 0.3740 - mse: 0.3384 - val_loss: 0.1266 - val_mae: 0.2527 - val_mse: 0.1266
Epoch 6/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.3152 -
mae: 0.3481 - mse: 0.3152 - val_loss: 0.1125 - val_mae: 0.2412 - val_mse: 0.1125
Epoch 7/200
1022/1022 [==============================] - 0s 164us/sample - loss: 0.3020 -
mae: 0.3350 - mse: 0.3020 - val_loss: 0.1173 - val_mae: 0.2408 - val_mse: 0.1173
Epoch 8/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.2684 -
mae: 0.3390 - mse: 0.2684 - val_loss: 0.1184 - val_mae: 0.2464 - val_mse: 0.1184
Epoch 9/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.2717 -
mae: 0.3197 - mse: 0.2717 - val_loss: 0.1130 - val_mae: 0.2433 - val_mse: 0.1130
Epoch 10/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.2174 -
mae: 0.3016 - mse: 0.2174 - val_loss: 0.1072 - val_mae: 0.2428 - val_mse: 0.1072
Epoch 11/200
1022/1022 [==============================] - 0s 174us/sample - loss: 0.3030 -
mae: 0.3076 - mse: 0.3030 - val_loss: 0.1241 - val_mae: 0.2346 - val_mse: 0.1241
Epoch 12/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.2332 -
mae: 0.3031 - mse: 0.2332 - val_loss: 0.1224 - val_mae: 0.2395 - val_mse: 0.1224
Epoch 13/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.2496 -
```

```
mae: 0.2926 - mse: 0.2496 - val_loss: 0.1257 - val_mae: 0.2439 - val_mse: 0.1257
Epoch 14/200
1022/1022 [==============================] - 0s 175us/sample - loss: 0.2533 -
mae: 0.2920 - mse: 0.2533 - val_loss: 0.1069 - val_mae: 0.2263 - val_mse: 0.1069
Epoch 15/200
1022/1022 [==============================] - 0s 196us/sample - loss: 0.2446 -
mae: 0.2996 - mse: 0.2446 - val_loss: 0.1021 - val_mae: 0.2230 - val_mse: 0.1021
Epoch 16/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.1916 -
mae: 0.2847 - mse: 0.1916 - val_loss: 0.1038 - val_mae: 0.2350 - val_mse: 0.1038
Epoch 17/200
1022/1022 [==============================] - 0s 172us/sample - loss: 0.2118 -
mae: 0.2746 - mse: 0.2118 - val_loss: 0.1141 - val_mae: 0.2446 - val_mse: 0.1141
Epoch 18/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.2006 -
mae: 0.2732 - mse: 0.2006 - val_loss: 0.1121 - val_mae: 0.2448 - val_mse: 0.1121
Epoch 19/200
1022/1022 [==============================] - 0s 178us/sample - loss: 0.1790 -
mae: 0.2611 - mse: 0.1790 - val_loss: 0.1022 - val_mae: 0.2241 - val_mse: 0.1022
Epoch 20/200
1022/1022 [==============================] - 0s 189us/sample - loss: 0.2167 -
mae: 0.2640 - mse: 0.2167 - val_loss: 0.1090 - val_mae: 0.2339 - val_mse: 0.1090
Epoch 21/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.1929 -
mae: 0.2728 - mse: 0.1929 - val_loss: 0.1133 - val_mae: 0.2340 - val_mse: 0.1133
Epoch 22/200
1022/1022 [==============================] - 0s 169us/sample - loss: 0.1831 -
mae: 0.2672 - mse: 0.1831 - val_loss: 0.1098 - val_mae: 0.2339 - val_mse: 0.1098
Epoch 23/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.1614 -
mae: 0.2585 - mse: 0.1614 - val_loss: 0.1025 - val_mae: 0.2249 - val_mse: 0.1025
Epoch 24/200
1022/1022 [==============================] - 0s 180us/sample - loss: 0.1986 -
mae: 0.2609 - mse: 0.1986 - val_loss: 0.1129 - val_mae: 0.2440 - val_mse: 0.1129
Epoch 25/200
1022/1022 [==============================] - 0s 170us/sample - loss: 0.1961 -
mae: 0.2583 - mse: 0.1961 - val_loss: 0.1085 - val_mae: 0.2335 - val_mse: 0.1085
Temps d'entrainement 5.230571269989014 seconds ---
Predicting house prices model network 32 * 32 * 1 with Callback
Regression metrics for train data
    max_error  mean_absolute_error  mean_squared_error  r2_score
0  382412.125          16899.94266        7.894516e+08  0.884939
Regression metrics for test data
    max_error  mean_absolute_error  mean_squared_error  r2_score
0  214109.625         19345.023277        7.442062e+08    0.8498
```

On arrive à une conclusion identique que l'architecture 64 par 64 avec une limitation du surapprentissage grâce aux paramètres DropOut sur les couches intermédiaires.

7.8.3 Modèle avec 4 couches cachées

[75]:
```
TrainingNetwork(model_32_16_8_4_1, str(32) +" * " + str(16) + " * " + str(8) + "␣
↪* " + str(4)  + " * 1", callback=True)
PredictionAndEvaluationNetwork(model_32_16_8_4_1,str(32) +" * " + str(16) + " *␣
↪" + str(8) + " * " + str(4)  + " * 1", callback=True)
```

```
Training model network 32 * 16 * 8 * 4 * 1 with Callback
Train on 1022 samples, validate on 438 samples
Epoch 1/200
1022/1022 [==============================] - 1s 838us/sample - loss: 0.7428 -
mae: 0.5262 - mse: 0.7428 - val_loss: 0.3609 - val_mae: 0.3688 - val_mse: 0.3609
Epoch 2/200
1022/1022 [==============================] - 0s 201us/sample - loss: 0.4517 -
mae: 0.3885 - mse: 0.4517 - val_loss: 0.2189 - val_mae: 0.3105 - val_mse: 0.2189
Epoch 3/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.2895 -
mae: 0.3112 - mse: 0.2895 - val_loss: 0.1377 - val_mae: 0.2553 - val_mse: 0.1377
Epoch 4/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.2169 -
mae: 0.2647 - mse: 0.2169 - val_loss: 0.1092 - val_mae: 0.2248 - val_mse: 0.1092
Epoch 5/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.1925 -
mae: 0.2360 - mse: 0.1925 - val_loss: 0.0930 - val_mae: 0.2068 - val_mse: 0.0930
Epoch 6/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.1695 -
mae: 0.2166 - mse: 0.1695 - val_loss: 0.0879 - val_mae: 0.2041 - val_mse: 0.0879
Epoch 7/200
1022/1022 [==============================] - 0s 199us/sample - loss: 0.1571 -
mae: 0.2017 - mse: 0.1571 - val_loss: 0.0839 - val_mae: 0.1946 - val_mse: 0.0839
Epoch 8/200
1022/1022 [==============================] - 0s 187us/sample - loss: 0.1467 -
mae: 0.1877 - mse: 0.1467 - val_loss: 0.0801 - val_mae: 0.1931 - val_mse: 0.0801
Epoch 9/200
1022/1022 [==============================] - 0s 177us/sample - loss: 0.1388 -
mae: 0.1798 - mse: 0.1388 - val_loss: 0.0787 - val_mae: 0.1892 - val_mse: 0.0787
Epoch 10/200
1022/1022 [==============================] - 0s 185us/sample - loss: 0.1299 -
mae: 0.1687 - mse: 0.1299 - val_loss: 0.0782 - val_mae: 0.1888 - val_mse: 0.0782
Epoch 11/200
1022/1022 [==============================] - 0s 189us/sample - loss: 0.1153 -
mae: 0.1655 - mse: 0.1153 - val_loss: 0.0782 - val_mae: 0.1870 - val_mse: 0.0782
Epoch 12/200
1022/1022 [==============================] - 0s 197us/sample - loss: 0.1095 -
mae: 0.1566 - mse: 0.1095 - val_loss: 0.0826 - val_mae: 0.1893 - val_mse: 0.0826
Epoch 13/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.1034 -
mae: 0.1510 - mse: 0.1034 - val_loss: 0.0827 - val_mae: 0.1888 - val_mse: 0.0827
```

```
Epoch 14/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.0936 -
mae: 0.1452 - mse: 0.0936 - val_loss: 0.0823 - val_mae: 0.1889 - val_mse: 0.0823
Epoch 15/200
1022/1022 [==============================] - 0s 183us/sample - loss: 0.0857 -
mae: 0.1400 - mse: 0.0857 - val_loss: 0.0843 - val_mae: 0.1885 - val_mse: 0.0843
Epoch 16/200
1022/1022 [==============================] - 0s 179us/sample - loss: 0.0827 -
mae: 0.1355 - mse: 0.0827 - val_loss: 0.0875 - val_mae: 0.1930 - val_mse: 0.0875
Epoch 17/200
1022/1022 [==============================] - 0s 191us/sample - loss: 0.0780 -
mae: 0.1292 - mse: 0.0780 - val_loss: 0.0876 - val_mae: 0.1910 - val_mse: 0.0876
Epoch 18/200
1022/1022 [==============================] - 0s 188us/sample - loss: 0.0677 -
mae: 0.1248 - mse: 0.0677 - val_loss: 0.0872 - val_mae: 0.1914 - val_mse: 0.0872
Epoch 19/200
1022/1022 [==============================] - 0s 188us/sample - loss: 0.0589 -
mae: 0.1215 - mse: 0.0589 - val_loss: 0.0913 - val_mae: 0.1966 - val_mse: 0.0913
Epoch 20/200
1022/1022 [==============================] - 0s 184us/sample - loss: 0.0543 -
mae: 0.1157 - mse: 0.0543 - val_loss: 0.0929 - val_mae: 0.1992 - val_mse: 0.0929
Temps d'entrainement 4.575250864028931 seconds ---
Predicting house prices model network 32 * 16 * 8 * 4 * 1 with Callback
Regression metrics for train data
     max_error  mean_absolute_error  mean_squared_error  r2_score
0  228293.4375          9134.514341        2.923508e+08   0.95739
Regression metrics for test data
    max_error  mean_absolute_error  mean_squared_error  r2_score
0  151578.875         16503.548034        6.372141e+08  0.871394
```

Le résultat de cette architecture 4 couches expose toujours du surapprentissage mais dans une mesure moindre que les archictectures 2 couches avec 32 ou 64 neurones sans dropout.

[76]:
```
print("Prise en compte du dropout dans le training dans les couches cachées avec␣
 ↪rate = 5%")
TrainingNetwork(model_32_16_8_4_1_DropOut, str(32) +" * " + str(16) + " * " +␣
 ↪str(8) + " * " + str(4)  + " * 1", callback=True)
PredictionAndEvaluationNetwork(model_32_16_8_4_1_DropOut,str(32) +" * " +␣
 ↪str(16) + " * " + str(8) + " * " + str(4)  + " * 1", callback=True)
```

```
Prise en compte du dropout dans le training dans les couches cachées avec rate =
5%
Training model network 32 * 16 * 8 * 4 * 1 with Callback
Train on 1022 samples, validate on 438 samples
Epoch 1/200
1022/1022 [==============================] - 1s 953us/sample - loss: 0.9866 -
mae: 0.6270 - mse: 0.9866 - val_loss: 0.4963 - val_mae: 0.4382 - val_mse: 0.4963
Epoch 2/200
```

```
1022/1022 [==============================] - 0s 204us/sample - loss: 0.7883 -
mae: 0.5196 - mse: 0.7883 - val_loss: 0.4576 - val_mae: 0.4133 - val_mse: 0.4576
Epoch 3/200
1022/1022 [==============================] - 0s 203us/sample - loss: 0.7355 -
mae: 0.4951 - mse: 0.7355 - val_loss: 0.4173 - val_mae: 0.3740 - val_mse: 0.4173
Epoch 4/200
1022/1022 [==============================] - 0s 204us/sample - loss: 0.6920 -
mae: 0.4667 - mse: 0.6920 - val_loss: 0.3953 - val_mae: 0.3585 - val_mse: 0.3953
Epoch 5/200
1022/1022 [==============================] - 0s 192us/sample - loss: 0.6659 -
mae: 0.4556 - mse: 0.6659 - val_loss: 0.3792 - val_mae: 0.3566 - val_mse: 0.3792
Epoch 6/200
1022/1022 [==============================] - 0s 205us/sample - loss: 0.6486 -
mae: 0.4496 - mse: 0.6486 - val_loss: 0.3636 - val_mae: 0.3518 - val_mse: 0.3636
Epoch 7/200
1022/1022 [==============================] - 0s 202us/sample - loss: 0.6139 -
mae: 0.4313 - mse: 0.6139 - val_loss: 0.3490 - val_mae: 0.3494 - val_mse: 0.3490
Epoch 8/200
1022/1022 [==============================] - 0s 205us/sample - loss: 0.5890 -
mae: 0.4197 - mse: 0.5890 - val_loss: 0.3160 - val_mae: 0.3107 - val_mse: 0.3160
Epoch 9/200
1022/1022 [==============================] - 0s 197us/sample - loss: 0.5580 -
mae: 0.4055 - mse: 0.5580 - val_loss: 0.2971 - val_mae: 0.3134 - val_mse: 0.2971
Epoch 10/200
1022/1022 [==============================] - 0s 201us/sample - loss: 0.5303 -
mae: 0.4045 - mse: 0.5303 - val_loss: 0.2649 - val_mae: 0.2979 - val_mse: 0.2649
Epoch 11/200
1022/1022 [==============================] - 0s 192us/sample - loss: 0.4602 -
mae: 0.3712 - mse: 0.4602 - val_loss: 0.2225 - val_mae: 0.2816 - val_mse: 0.2225
Epoch 12/200
1022/1022 [==============================] - 0s 202us/sample - loss: 0.3851 -
mae: 0.3598 - mse: 0.3851 - val_loss: 0.1623 - val_mae: 0.2439 - val_mse: 0.1623
Epoch 13/200
1022/1022 [==============================] - 0s 201us/sample - loss: 0.3388 -
mae: 0.3284 - mse: 0.3388 - val_loss: 0.1439 - val_mae: 0.2522 - val_mse: 0.1439
Epoch 14/200
1022/1022 [==============================] - 0s 200us/sample - loss: 0.2772 -
mae: 0.3122 - mse: 0.2772 - val_loss: 0.1326 - val_mae: 0.2498 - val_mse: 0.1326
Epoch 15/200
1022/1022 [==============================] - 0s 202us/sample - loss: 0.2655 -
mae: 0.3033 - mse: 0.2655 - val_loss: 0.1014 - val_mae: 0.2131 - val_mse: 0.1014
Epoch 16/200
1022/1022 [==============================] - 0s 188us/sample - loss: 0.2215 -
mae: 0.2785 - mse: 0.2215 - val_loss: 0.0968 - val_mae: 0.2174 - val_mse: 0.0968
Epoch 17/200
1022/1022 [==============================] - 0s 198us/sample - loss: 0.3018 -
mae: 0.2904 - mse: 0.3018 - val_loss: 0.0915 - val_mae: 0.1995 - val_mse: 0.0915
Epoch 18/200
```

```
1022/1022 [==============================] - 0s 199us/sample - loss: 0.2235 -
mae: 0.2674 - mse: 0.2235 - val_loss: 0.0972 - val_mae: 0.2224 - val_mse: 0.0972
Epoch 19/200
1022/1022 [==============================] - 0s 204us/sample - loss: 0.2413 -
mae: 0.2705 - mse: 0.2413 - val_loss: 0.1003 - val_mae: 0.2307 - val_mse: 0.1003
Epoch 20/200
1022/1022 [==============================] - 0s 194us/sample - loss: 0.3387 -
mae: 0.2759 - mse: 0.3387 - val_loss: 0.0975 - val_mae: 0.2223 - val_mse: 0.0975
Epoch 21/200
1022/1022 [==============================] - 0s 195us/sample - loss: 0.2756 -
mae: 0.2606 - mse: 0.2756 - val_loss: 0.1048 - val_mae: 0.2297 - val_mse: 0.1048
Epoch 22/200
1022/1022 [==============================] - 0s 209us/sample - loss: 0.2275 -
mae: 0.2545 - mse: 0.2275 - val_loss: 0.0856 - val_mae: 0.2007 - val_mse: 0.0856
Epoch 23/200
1022/1022 [==============================] - 0s 196us/sample - loss: 0.2182 -
mae: 0.2555 - mse: 0.2182 - val_loss: 0.0914 - val_mae: 0.2100 - val_mse: 0.0914
Epoch 24/200
1022/1022 [==============================] - 0s 195us/sample - loss: 0.2215 -
mae: 0.2459 - mse: 0.2215 - val_loss: 0.0951 - val_mae: 0.2144 - val_mse: 0.0951
Epoch 25/200
1022/1022 [==============================] - 0s 206us/sample - loss: 0.2030 -
mae: 0.2388 - mse: 0.2030 - val_loss: 0.0864 - val_mae: 0.2034 - val_mse: 0.0864
Epoch 26/200
1022/1022 [==============================] - 0s 189us/sample - loss: 0.1909 -
mae: 0.2307 - mse: 0.1909 - val_loss: 0.0907 - val_mae: 0.2067 - val_mse: 0.0907
Epoch 27/200
1022/1022 [==============================] - 0s 217us/sample - loss: 0.1992 -
mae: 0.2443 - mse: 0.1992 - val_loss: 0.0926 - val_mae: 0.2111 - val_mse: 0.0926
Epoch 28/200
1022/1022 [==============================] - 0s 197us/sample - loss: 0.1967 -
mae: 0.2309 - mse: 0.1967 - val_loss: 0.0906 - val_mae: 0.2096 - val_mse: 0.0906
Epoch 29/200
1022/1022 [==============================] - 0s 195us/sample - loss: 0.2219 -
mae: 0.2350 - mse: 0.2219 - val_loss: 0.0871 - val_mae: 0.1947 - val_mse: 0.0871
Epoch 30/200
1022/1022 [==============================] - 0s 194us/sample - loss: 0.1837 -
mae: 0.2218 - mse: 0.1837 - val_loss: 0.0866 - val_mae: 0.1995 - val_mse: 0.0866
Epoch 31/200
1022/1022 [==============================] - 0s 199us/sample - loss: 0.2162 -
mae: 0.2255 - mse: 0.2162 - val_loss: 0.0933 - val_mae: 0.2088 - val_mse: 0.0933
Epoch 32/200
1022/1022 [==============================] - 0s 216us/sample - loss: 0.1984 -
mae: 0.2113 - mse: 0.1984 - val_loss: 0.0968 - val_mae: 0.2135 - val_mse: 0.0968
Temps d'entrainement 7.440790176391602 seconds ---
Predicting house prices model network 32 * 16 * 8 * 4 * 1 with Callback
Regression metrics for train data
     max_error  mean_absolute_error  mean_squared_error  r2_score
```

```
0  356210.6875         15248.142926        8.047401e+08  0.882711
Regression metrics for test data
   max_error  mean_absolute_error  mean_squared_error  r2_score
0   184584.0         17684.972977        6.638654e+08  0.866015
```

Le dropout avec 5% sur les couches intermédiaires diminue le surapprentissage.

7.9 Conclusion sur les réseaux de neurones

- Différentes architectures peuvent être testées avec plus ou moins de couches cachées, le nombre de neurones par couche, du tuning fin sur des paramètres comme le dropout ou encore introduire des régularisations sur les données non réalisées dans le cas présent.
- Dans les différentes architectures testées, ma recommandation serait :
- Réseau 32/32 équivalent au réseau 64/64 mais préférence pour le réseau 32/32 car deux fois moins de paramètres
- Réseau 4 couches avec un dropOut de 5% a un résultat meilleur que le réseau en deux couches 32/32 et a globalement le même nombre de paramètres. Avec ce paramétrage, on a limité l'overfitting et on obtient une meilleure généralisation.

8 Comparaison des modèles

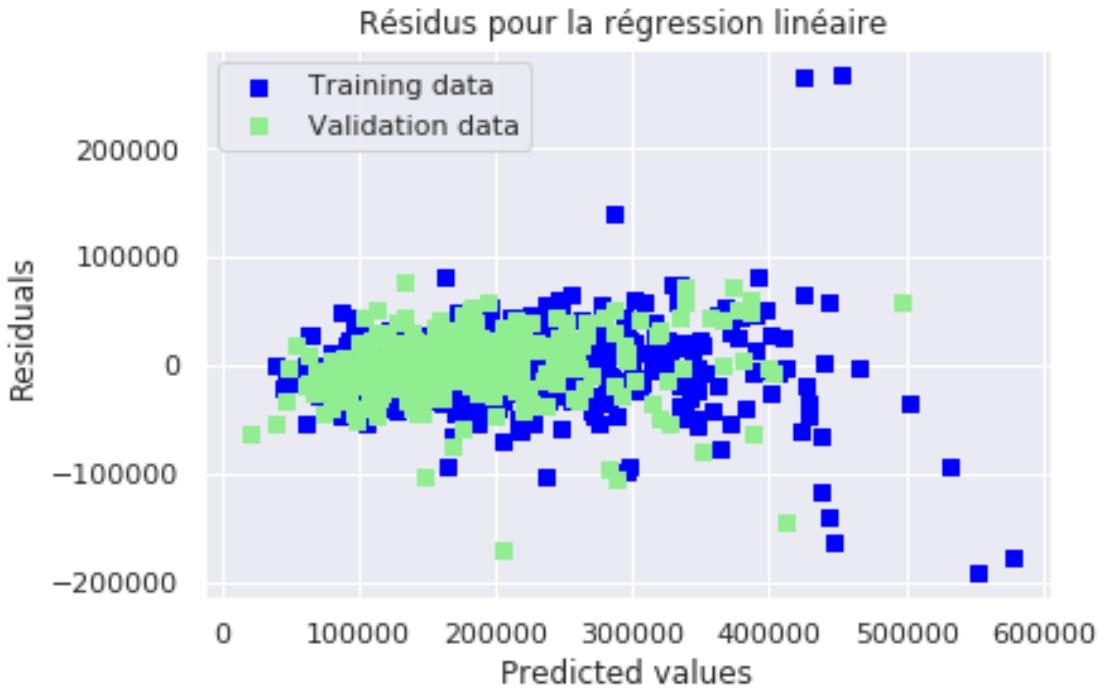
8.1 Rappel des prédictions pour les différents modèles

[0]:
```
#Regression linéaire
y_trainPredict_linear = np.around(Yscaler.inverse_transform(clfRegLinear.
 ↪predict(X_train)), decimals = 1)
y_testPredict_linear = np.around(Yscaler.inverse_transform(clfRegLinear.
 ↪predict(X_test)), decimals = 1)

#Regression lasso
y_trainPredict_lasso = np.around(Yscaler.inverse_transform(clfLasso.
 ↪predict(X_train)), decimals = 1)
y_testPredict_lasso = np.around(Yscaler.inverse_transform(clfLasso.
 ↪predict(X_test)), decimals = 1)

#Réseau de neurones 4 couches avec dropOut=5%
y_trainPredict_network = np.around(Yscaler.
 ↪inverse_transform(model_32_16_8_4_1_DropOut.predict(X_train_transformed.
 ↪toarray())), decimals =1)
y_testPredict_network = np.around(Yscaler.
 ↪inverse_transform(model_32_16_8_4_1_DropOut.predict(X_test_transformed.
 ↪toarray())), decimals =1)
```

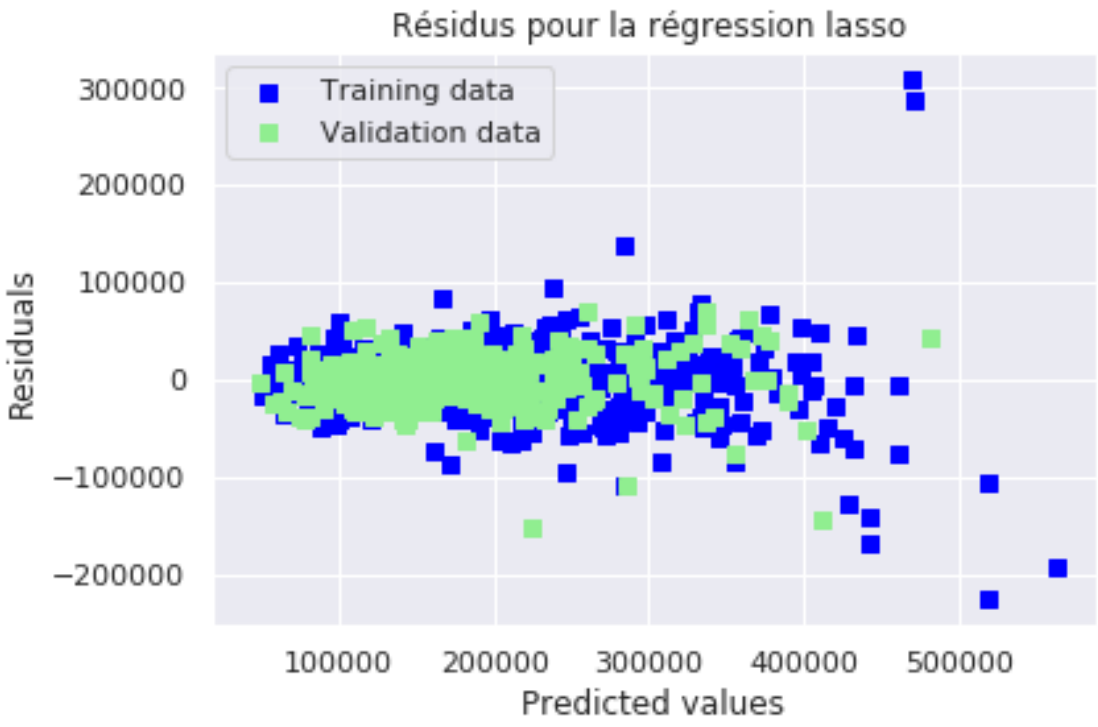
8.2 Graphique des résidus

[0]:
```
def PlotResidual(y_trainReal, y_trainPredicted, y_testReal, y_testPredicted,␣
↪title):
 plt.scatter(y_trainPredicted, y_trainPredicted - y_trainReal, c = "blue",␣
↪marker = "s", label = "Training data")
 plt.scatter(y_testPredicted, y_testPredicted - y_testReal, c = "lightgreen",␣
↪marker = "s", label = "Validation data")
 plt.title(title)
 plt.xlabel("Predicted values")
 plt.ylabel("Residuals")
 plt.legend(loc = "best")
 plt.show()
```

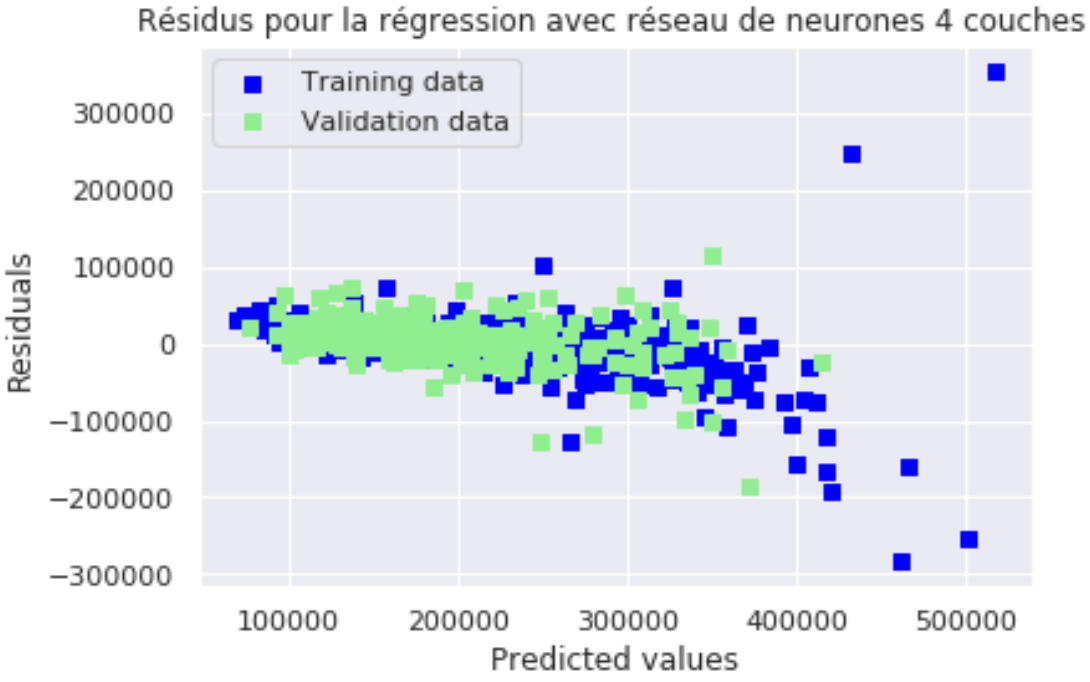
[79]:
```
PlotResidual(y_train, y_trainPredict_linear, y_test,␣
↪y_testPredict_linear,"Résidus pour la régression linéaire ")
```



[80]:
```
PlotResidual(y_train, y_trainPredict_lasso, y_test, y_testPredict_lasso,"Résidus␣
↪pour la régression lasso ")
```



[81]:
```
PlotResidual(y_train, y_trainPredict_network.reshape(-1), y_test,␣
↪y_testPredict_network.reshape(-1),"Résidus pour la régression avec réseau de␣
↪neurones 4 couches ")
```



- Le graphique des résidus exposent des formes similaires entre la régression linéaire et la régression de type lasso ce qui apparait logique par construction. Les points les moins bien prédits semblent en tendance ceux dont le prix de vente est élevé.
- Le graphique des résidus avec un réseau de neurones est bien différent en ayant plutôt une bonne prédiction sur les prix élevés et en se trompant plutôt sur le ventre mou des prix.
- On peut observer aussi le surapprentissage dans le cas présent beaucoup plus important dans le cas du réseau de neurone visible à travers le nuage bleu sur le training.

8.3 Comparaison des métriques

On regarde les métriques uniquement sur l'ensemble de validation.

[82]:
```
dataDict = {
    'Linear Regression' : [
        metrics.max_error(y_true=y_test, y_pred=y_testPredict_linear),
        metrics.mean_absolute_error(y_true=y_test, y_pred=y_testPredict_linear),
        metrics.mean_squared_error(y_true=y_test, y_pred=y_testPredict_linear),
        metrics.r2_score(y_true=y_test, y_pred=y_testPredict_linear)
    ],
    'Lasso Regression' : [
        metrics.max_error(y_true=y_test, y_pred=y_testPredict_lasso),
        metrics.mean_absolute_error(y_true=y_test, y_pred=y_testPredict_lasso),
        metrics.mean_squared_error(y_true=y_test, y_pred=y_testPredict_lasso),
        metrics.r2_score(y_true=y_test, y_pred=y_testPredict_lasso)
    ],
    'Network Regression' : [
        metrics.max_error(y_true=y_test, y_pred=y_testPredict_network),
        metrics.mean_absolute_error(y_true=y_test, y_pred=y_testPredict_network),
        metrics.mean_squared_error(y_true=y_test, y_pred=y_testPredict_network),
        metrics.r2_score(y_true=y_test, y_pred=y_testPredict_network)
    ],

}
metricsComparaison = pd.DataFrame.from_dict(dataDict, orient='index',␣
 ↪columns=["max_error", "mean_absolute_error", "mean_squared_error", "r2_score"])
metricsComparaison
```

[82]:
```
                   max_error  ...  r2_score
Linear Regression   170230.3  ...  0.852876
Lasso Regression    150555.3  ...  0.886689
Network Regression  184584.0  ...  0.866015

[3 rows x 4 columns]
```

- Le modèle Lasso domine la régression linéaire et offre aussi une facilité dans l'interprétation en imposant des coefficients nuls.
- Le réseau de neurones suivant certaines métriques pourrait être préféré au modèle Lasso.

#Soumission sur le test

On garde le modèle Lasso pour son interprétabilité en arrondissant les résultats avec uniquement une décimale.

[0]:
```
predictionWithLassoRegression = np.around(Yscaler.inverse_transform(clfLasso.
↪predict(df_test)), decimals = 1)
```

On construit un dataframe avec l'id et les résultats

[84]:
```
submission = pd.DataFrame({'Id':df_test['Id'], 'Sales Price':
↪predictionWithLassoRegression})

#Visualize the first 5 rows
submission.head(10)
```

[84]:

	Id	Sales Price
0	1461	135235.7
1	1462	142465.8
2	1463	172334.5
3	1464	186485.0
4	1465	212879.4
5	1466	162479.6
6	1467	172909.3
7	1468	152326.8
8	1469	201835.9
9	1470	127526.2

[0]:
```
#fonctionne uniquement en local
#filename = '../Output/HousePricesPrediction.csv'
#submission.to_csv(filename,index=False)
#print('Sauvegarde du fichier : ' + filename)
```

9 Perspectives

- une validation croisée pourrait être mise en oeuvre pour définir plus précisément le modèle en étant robuste à la détermination de l'ensemble d'apprentissage et de validation
- du featuring engineering pourrait être réalisé pour agréger éventuellement les données sur les surfaces afin qu'elle ressorte plus distinctement dans la régression comme un facteur explicatif du prix.
- une analyse en composante principale pourrait peut-être aussi agréger les valeurs numériques de manière intéressante
- un traitement des outliers pourrait aussi améliorer les résultats