H4221-ML-Notebook

April 30, 2023

1 H4221 - Rapport Machine Learning

Au cours du TP de Machine Learning décomposé en 2 séances de 4h, nous allons entrainer de multiples modèles/algorithme de Machine Learning sur un jeu de données sur les ventes de maisons à Seattle (USA) afin de pouvoir prédire les prix d'autres maisons.

Auteurs: - DUBILLOT Elise - FLANDRE Corentin - THOMAS Colin

1.1 Introduction

Le TP de Machine Learning va se diviser en plusieurs étapes clés: - Récupération de données - Nettoyage du jeu de données - Exploration du jeu de données

1.2 Récupération du jeu de données

Notre jeu de données est disponible à l'url suivante : https://www.kaggle.com/datasets/harlfoxem/housesalespredic

[ ]:
```
import pandas as pd
df = pd.read_csv("kc_house_data.csv")
```

Il s'avère que le jeu de données est très large et propre qui comporte des mesures numériques (hormis la date qui sera géré plus tard). Du fait du jeu de données propre, aucun nettoyage n'est nécessaire.

1.3 Exploration du jeu de données

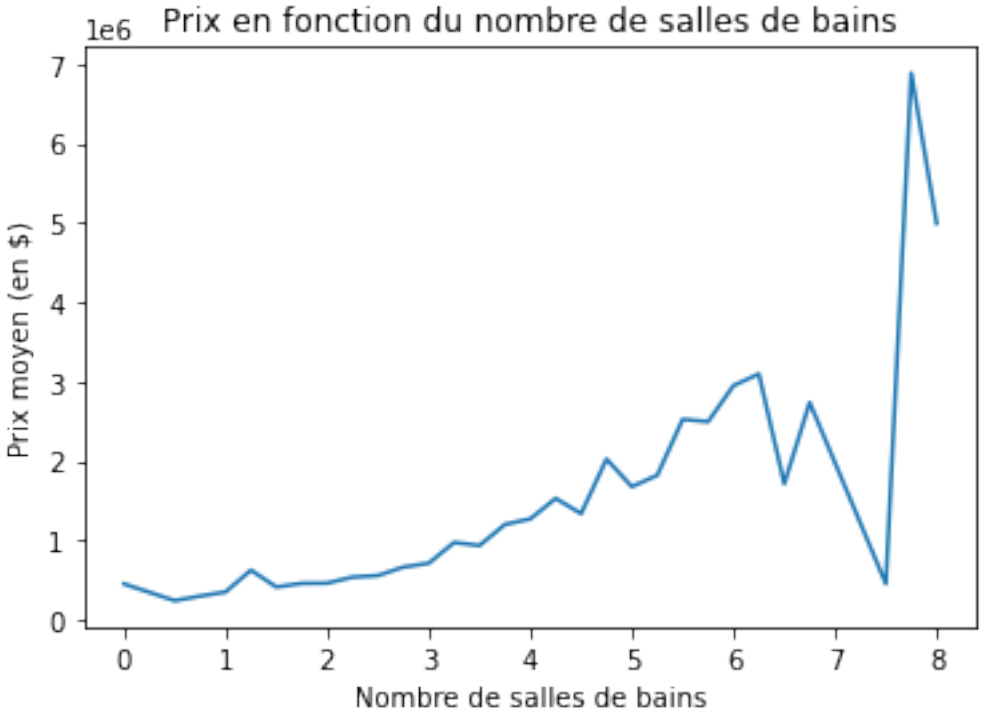
[ ]:
```
import matplotlib.pyplot as plt


dfsorted = df[["price", "bathrooms"]]
dfsorted.sort_values(by=["bathrooms"], inplace=True)

dfpbyb = dfsorted.groupby("bathrooms")["price"].mean()

dfpbyb.to_numpy()
plt.plot(dfpbyb[:])

plt.xlabel('Nombre de salles de bains')
plt.ylabel('Prix moyen (en $)')
```

```
plt.title('Prix en fonction du nombre de salles de bains')
plt.show()

dfsorted = df[["price", "floors"]]
dfsorted.sort_values(by=["floors"], inplace=True)

dfpbyb = dfsorted.groupby("floors")["price"].mean()

dfpbyb.to_numpy()
plt.plot(dfpbyb[:])

plt.xlabel("Nombre d'étages")
plt.ylabel('Prix moyen (en $)')
plt.title('Prix en fonction du nombre d\'étages')
plt.show()


dfsorted = df[["price", "sqft_living"]]
dfsorted.sort_values(by=["sqft_living"], inplace=True)

dfpbyb = dfsorted.groupby("sqft_living")["price"].mean()

dfpbyb.to_numpy()
plt.plot(dfpbyb[:])

plt.xlabel("Nombre de mètres carrés vivables")
plt.ylabel('Prix moyen (en $)')
plt.title('Prix en fonction du nombre de mètres carrés vivables')
plt.show()


dfsorted = df[["price", "bedrooms"]]
dfsorted.sort_values(by=["bedrooms"], inplace=True)

dfpbyb = dfsorted.groupby("bedrooms")["price"].mean()

dfpbyb.to_numpy()
plt.plot(dfpbyb[:])

plt.xlabel("Nombre de chambres")
plt.ylabel('Prix moyen (en $)')
plt.title('Prix en fonction du nombre de chambres')
plt.show()


dfsorted = df[["price", "sqft_lot"]]
dfsorted.sort_values(by=["sqft_lot"], inplace=True)
```

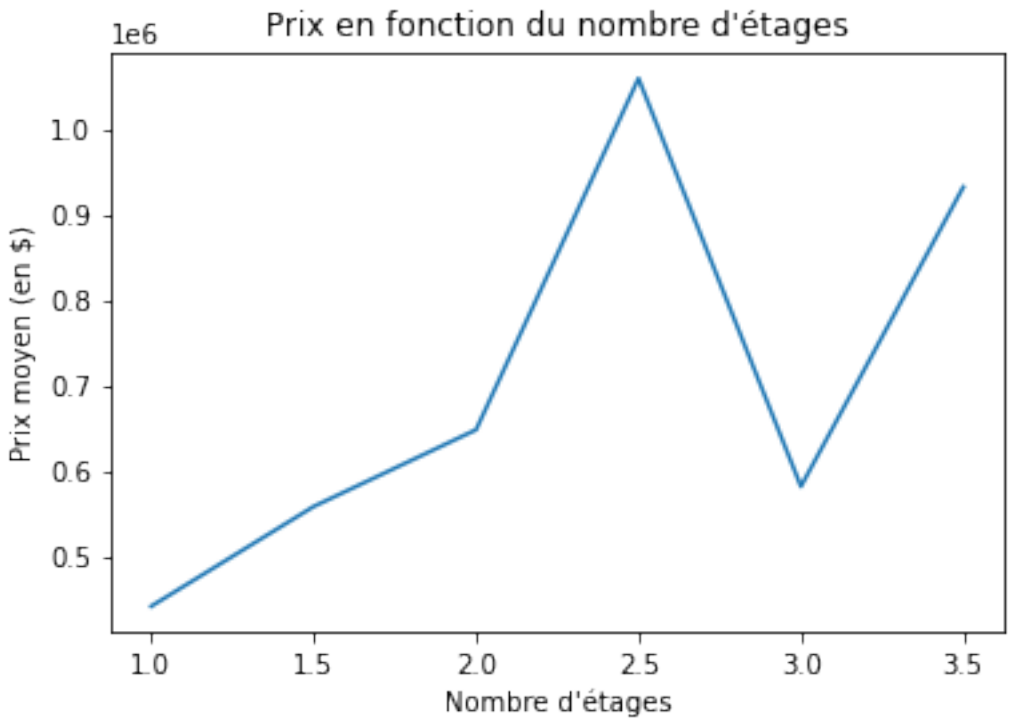
```
dfpbyb = dfsorted.groupby("sqft_lot")["price"].mean()

dfpbyb.to_numpy()
plt.plot(dfpbyb[:])

plt.xlabel("Taille du terrain en sqrft")
plt.ylabel('Prix moyen (en $)')
plt.title('Prix en fonction de la taille du terrain')
plt.show()

dfsorted = df[["price", "zipcode"]]
dfsorted.sort_values(by=["zipcode"], inplace=True)

dfpbyb = dfsorted.groupby("zipcode")["price"].mean()

dfpbyb.to_numpy()
plt.plot(dfpbyb[:])

plt.xlabel("ZIPCODE")
plt.ylabel('Prix moyen (en $)')
plt.title('Prix en fonction du zipcode')
plt.show()


dfsorted = df[["price", "lat"]]
dfsorted.sort_values(by=["lat"], inplace=True)

dfpbyb = dfsorted.groupby("lat")["price"].mean()

dfpbyb.to_numpy()
plt.hist(df["lat"], bins = 30)

plt.xlabel("Latitude")
plt.title('Répartition des données sur la latitude')
plt.show()

dfsorted = df[["price", "long"]]
dfsorted.sort_values(by=["long"], inplace=True)

dfpbyb = dfsorted.groupby("long")["price"].mean()

dfpbyb.to_numpy()
plt.hist(df["long"])

plt.xlabel("Longitude")
plt.title('Répartition des données sur la longitude')
```

```
plt.show()
```

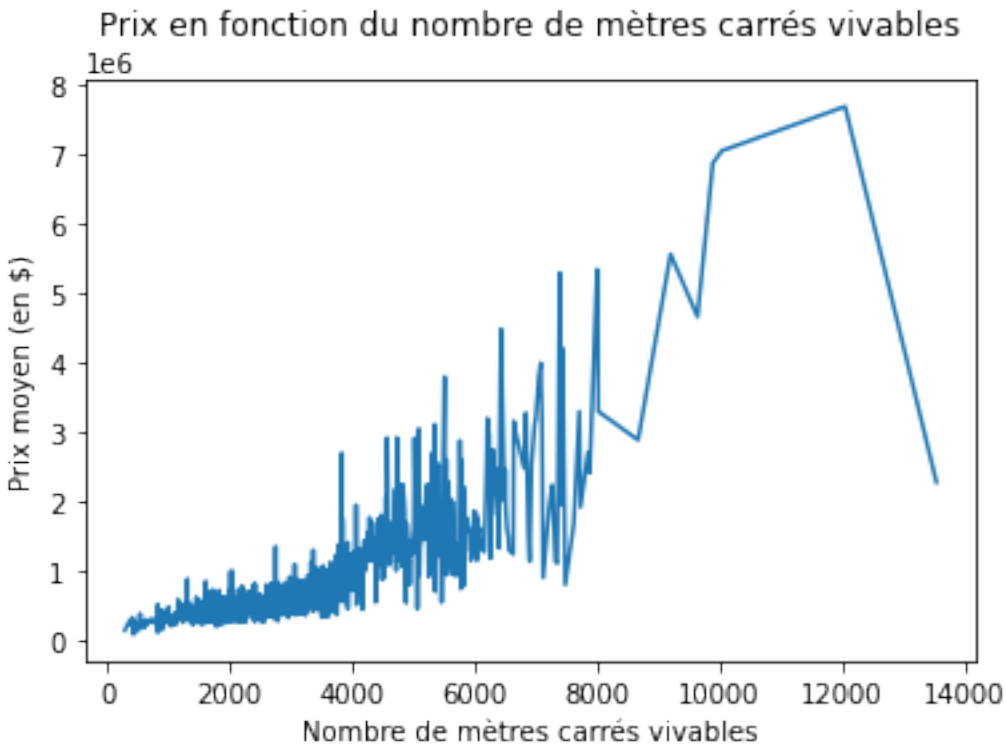
```
/tmp/ipykernel_1950/3840609229.py:5: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["bathrooms"], inplace=True)
```



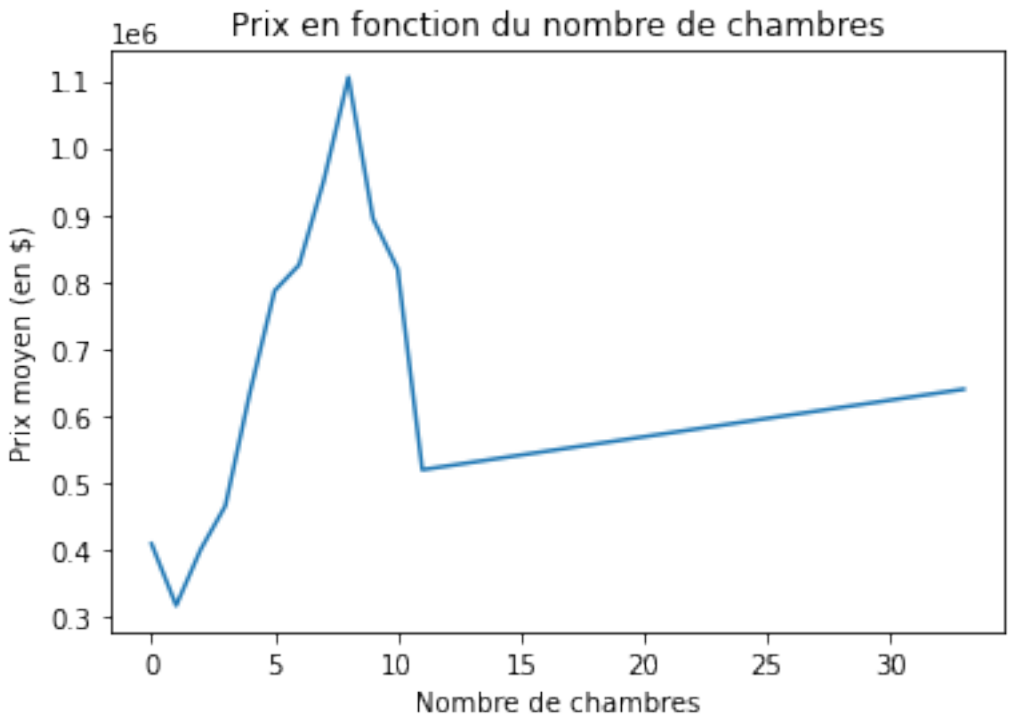

```
/tmp/ipykernel_1950/3840609229.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["floors"], inplace=True)
```



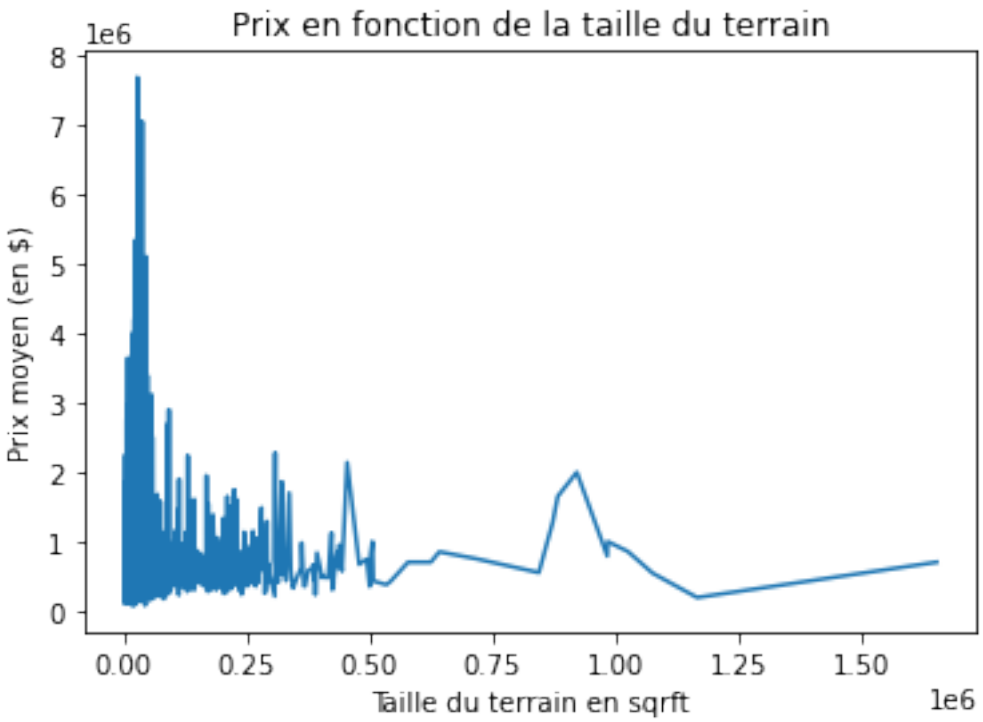

```
/tmp/ipykernel_1950/3840609229.py:32: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["sqft_living"], inplace=True)
```



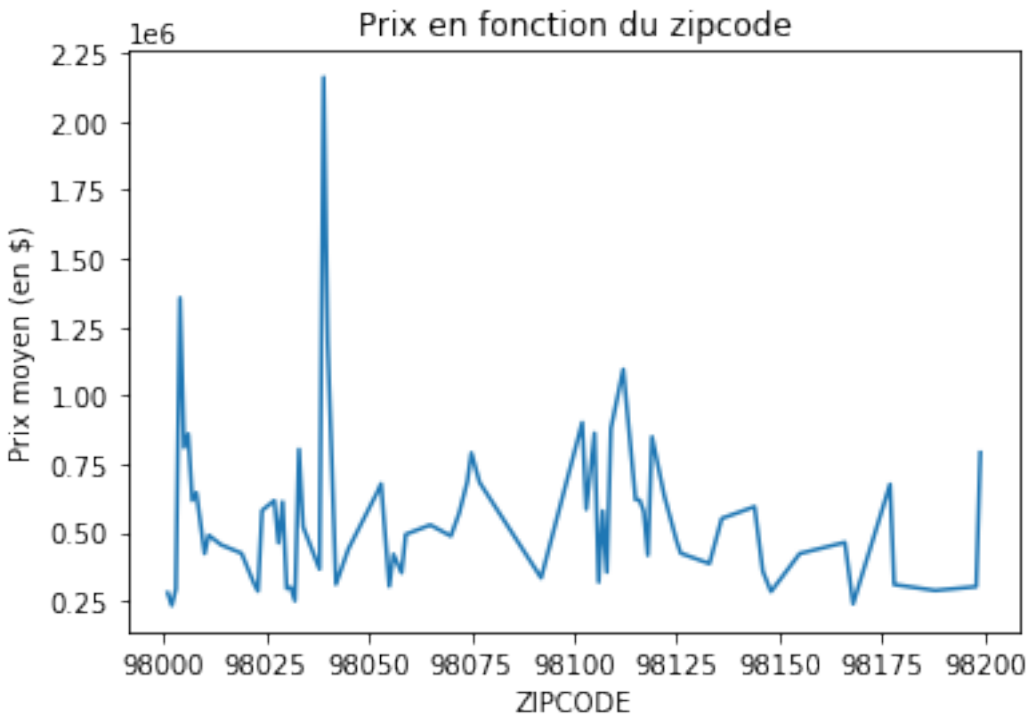

```
/tmp/ipykernel_1950/3840609229.py:46: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["bedrooms"], inplace=True)
```



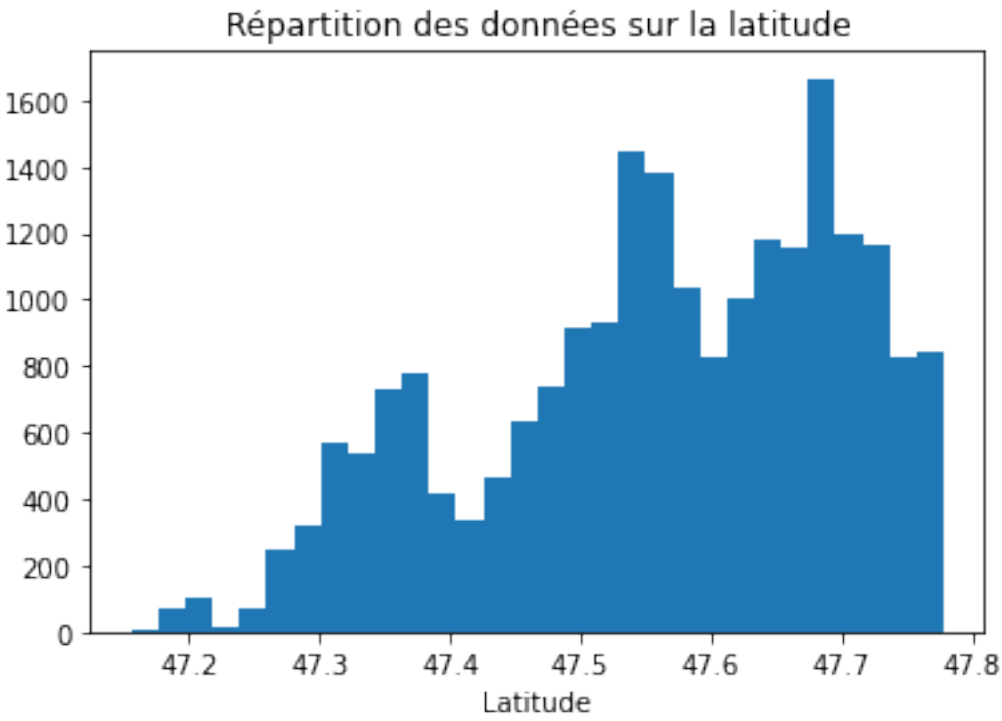

```
/tmp/ipykernel_1950/3840609229.py:60: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["sqft_lot"], inplace=True)
```



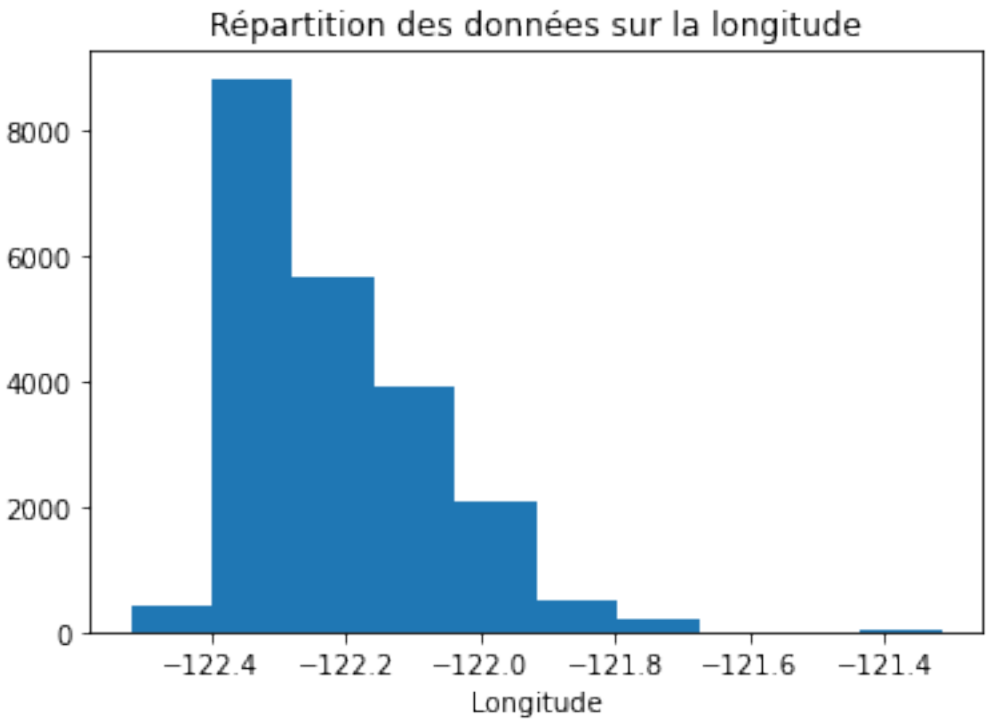

```
/tmp/ipykernel_1950/3840609229.py:73: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["zipcode"], inplace=True)
```




```
/tmp/ipykernel_1950/3840609229.py:87: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["lat"], inplace=True)
```




```
/tmp/ipykernel_1950/3840609229.py:99: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  dfsorted.sort_values(by=["long"], inplace=True)
```



[ ]:
```
#import plotly.express as px

#fig = px.density_mapbox(df, lat='lat', lon='long',
#                         mapbox_style="stamen-terrain")

#fig
```

On remarque ici plusieurs cas aberrants, que l'on traitera dans la partie traitement des données. Par exemple, dans le cas du nombre de chambre, on remarque une donnée qui représente une maison de 33 chambres, coûtant moins qu'une maison moyenne de 7 chambres et n'ayant que 1.25 salles de bain. On remarque également que le prix et la taille du terrain ne semblent pas avoir un lien au delà de environ 0.05 sqft. On peut supposer que cela est du à la multiplicité de facteurs influençant le prix.

1.4 Traitement du jeu de données

Suite à la phase d'exploration de données, nous allons les traiter afin d'avoir de meilleurs modèles. Ce traitement concerne principalement les valeurs aberrantes. Ces données abberrantes ne semblent pas pertinentes pour nos modèles afin de mieux travailler sur la variable cible du prix (et mieux estimer les prix des prochaines maisons). L'étape de traitement du jeu de données concerne aussi les dates qui seront traités afin de devenir une donnée numérique. Il aurait été possible d'utiliser un "encodage one-hot" mais

[ ]:
```
# taille du jeu de données initial
size_dataset_raws = len(df)
index_treatement = []

# suppression de la données à plus de 11 chambres
# print(f"before treatment of nb of bedrooms: {len(df.
 ↪loc[df['bedrooms']==33])}")
indexNames = df[ df['bedrooms'] >= 11 ].index
for value in indexNames.values:
    index_treatement.append(int(value))
# print(f"I: {index_treatement}")
df.drop(indexNames , inplace=True)
# print(f"after treatment of nb of bedrooms: {len(df.loc[df['bedrooms']==33])}")

# suppression de la données à 13540 m² de surface habitable
# print(f"before treatment of sqft_living: {len(df.
 ↪loc[df['sqft_living']>=12000])}")
indexNames2 = df[ df['sqft_living'] >= 12000 ].index
for value in indexNames2.values:
    index_treatement.append(int(value))
df.drop(indexNames2 , inplace=True)
# print(f"after treatment of sqft_living: {len(df.
 ↪loc[df['sqft_living']>=12000])}")


# suppression des données à 7 salles de bains
indexNames3 = df[df['bathrooms'] >=7].index
for value in indexNames3.values:
    index_treatement.append(int(value))
df.drop(indexNames3 , inplace=True)
```

Pour ce qui est des dates, nous faisons le choix d'identifier uniquement le numéro de mois. Le jeu de données prend en compte des dates entre Mai 2014 et Mai 2015.

[ ]:
```
new_dates = []
for i in range(size_dataset_raws):
    if i not in index_treatement:
        year = int(df["date"][i][0:4])
        month = int(df["date"][i][4:6])
        # day = int(df["date"][i][6:8]) # donnee non utilise pour notre␣
 ↪traitement
        new_date = 0
        if year==2014:
            if month==5:
                new_date = 1
            elif month==6:
                new_date = 2
```

```
            elif month==7:
                new_date = 3
            elif month==8:
                new_date = 4
            elif month==9:
                new_date = 5
            elif month==10:
                new_date = 6
            elif month==11:
                new_date = 7
            elif month==12:
                new_date = 8
        elif year==2015:
            if month==1:
                new_date = 9
            elif month==2:
                new_date = 10
            elif month==3:
                new_date = 11
            elif month==4:
                new_date = 12
            elif month==5:
                new_date = 13
        new_dates.append(new_date)
df.date = new_dates

# Gestion des erreurs de dates
print(f"Il y a {len(df.loc[df['date'] == 0])} lignes de données avec des dates␣
 ↪comprises entre Mai-2014 et Mai-2015")
```

```
Il y a 0 lignes de données avec des dates comprises entre Mai-2014 et Mai-2015
```

2 Modèle de régression linéaire

Le premier modèle que nous allons appliqué n'est

[ ]:
```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import numpy as np

# Données d'entrées
```

```
X = np.matrix([df['date'].values, df['sqft_lot'].values ,df['bedrooms'].values,␣
 ↪df['bathrooms'].values, df['sqft_living'].values, df['floors'].values,␣
 ↪df['waterfront'].values, df['view'].values, df['condition'].values,␣
 ↪df['grade'].values, df['sqft_above'].values, df['sqft_basement'].values,␣
 ↪df['yr_built'].values, df['yr_renovated'].values,df['zipcode'].
 ↪values,df['lat'].values,df['long'].values, df['sqft_living15'].values]).T

# Données de sortie
y = np.matrix(df['price']).T


# Standardisation
X = np.asarray(X)
y = np.asarray(y)
scaler = StandardScaler().fit(X)
X = np.asarray(scaler.transform(X))

# Division en training et testing set
X_training_set, X_testing_set, y_training_set, y_testing_set =␣
 ↪train_test_split(X, y, train_size=0.8)

# Application du modèle
reg = LinearRegression().fit(X_training_set, y_training_set)

print(f"Coefficient estimés du problème: {reg.coef_[0]}")
```

```
Coefficient estimés du problème: [ 13346.44695524    468.54084865
-31193.76792204  33414.15852298
  72549.39532276   5666.85023666  46397.24023474  37489.91435983
  18739.99112468 117496.62754608  66869.77119124  24961.25094805
 -76886.10964352   8484.36453057 -28669.30893924  83666.88443317
 -29852.49273553  22037.31689776]
```

On va donc tester notre modèle sur notre jeu de données d'entrainement:

[ ]:
```
y_predict = reg.predict(X_testing_set)
score = reg.score(X_testing_set, y_testing_set)
print(f"Coefficient de determination (R2): {score}")
sum_error = 0
for i in range(len(y_predict)):
    sum_error += abs(y_predict[i]-y_testing_set[i])
mean_error = sum_error/len(y_predict)
print(f"Moyenne d'erreur: {mean_error[0]}$")
```

```
Coefficient de determination (R2): 0.7012015369179463
Moyenne d'erreur: 124382.52516266327$
```

Le modèle de régression linéaire (avec méthode des moindres carrés) n'est pas spécialement bon,

on va donc essayer de l'entrainer avec un modèle de régression ridge (qui prend en compte de la régularisation) dans l'étape suivante

3 Modèle de régression ridge

Dans cette partie, nous allons utiliser un modèle de regression ridge pour prédire les prix d'autres maisons. Pour rappel, il s'agit dans notre cas d'un apprentissage supervisé puisque nous connaissons la sortie des données (c'est à dire le prix) et c'est un modèle linéaire paramétrique. Nous utiliserons la bibliothèque libre Scikit-Learn.

Un mélange des données sera nécessaire, nous l'effectuons de suite. Nous réduisons les dimensions des données aux dimensions que nous jugeons intéressantes pour les données d'entrées de notre algorithme (en utilisant l'étape d'exploration). Nous utilisons la dimension du prix pour les données de sorties.

[ ]:
```
# début code ridge regression
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge
import matplotlib.pyplot as plt



# "Shuffle" des données
df = df.iloc[np.random.permutation(len(df))]

# On décompose le dataset et on le transforme en matrices pour pouvoir␣
 ↪effectuer notre calcul
X = np.matrix([np.ones(df.shape[0]), df['date'].values, df['sqft_lot'].values␣
 ↪,df['bedrooms'].values, df['bathrooms'].values, df['sqft_living'].values,␣
 ↪df['floors'].values, df['waterfront'].values, df['view'].values,␣
 ↪df['condition'].values, df['grade'].values, df['sqft_above'].values,␣
 ↪df['sqft_basement'].values, df['yr_built'].values, df['yr_renovated'].
 ↪values,df['zipcode'].values,df['lat'].values,df['long'].values,␣
 ↪df['sqft_living15'].values]).T
y = np.matrix(df['price']).T
```

Nous normalisons les données d'entrées:

[ ]:
```
X = np.asarray(X)
scaler = StandardScaler().fit(X)
X = scaler.transform(X)
```

Pour voir l'efficacité de ce modèle, nous allons diviser notre jeu de données en deux: un "training test" et un "testing set" Le "training set" va permettre d'apprendre pour répondre à notre tâche de prédiction de prix. Le "testing set" va permet de mesurer l'erreur de prédiction de prix des maisons sur des données jamais vues par le modèle final. Nous allons repartir notre jeu de données en 80%

de données pour le training set et 20% pour le “testing set”. Il n’y a pas besoin de réduire le jeu d’entrée de données puisque le problème se résout en temps raisonnable sans cette réduction.

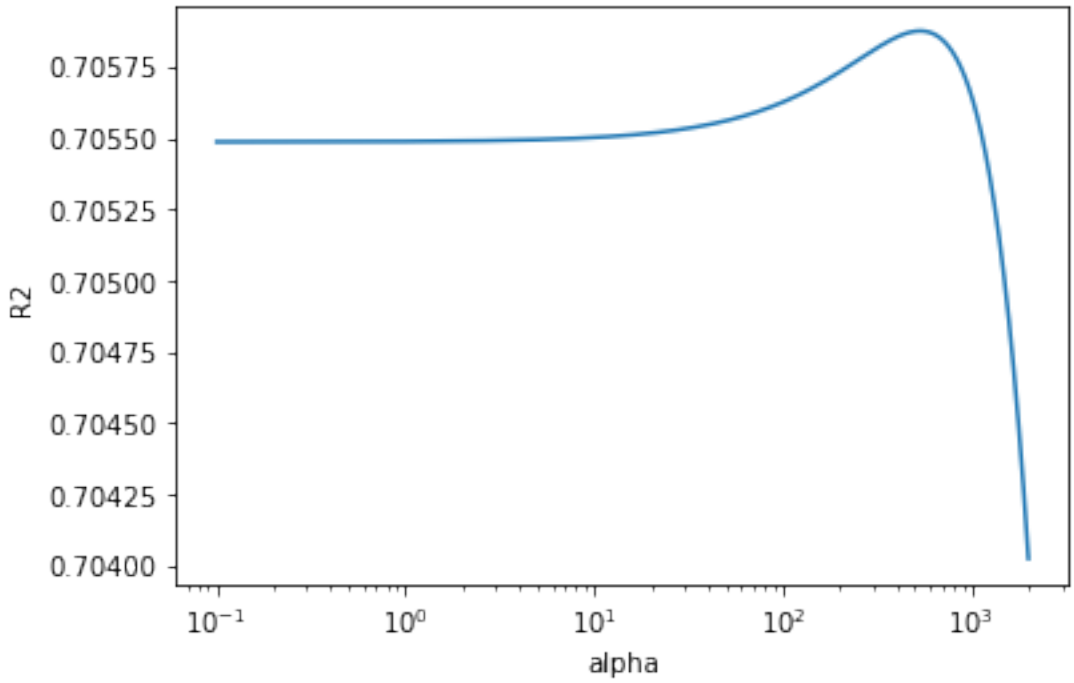
Le mélange précédent des données permet de biaiser au minimum notre modèle.

[ ]:
```
# Training set (80% des valeurs)
# Testing set (20% des valeurs)
X_training_set, X_testing_set, y_training_set, y_testing_set =␣
 ↪train_test_split(X, y, train_size=0.8)
```

Nous devons trouver un **coefficient de régularisation** adapté. Nous appellons ce coefficient alpha, nous allons en tester un certain nombre afin de trouver celui qui est optimal.

[ ]:
```
n_alphas = 1000
alphas = np.logspace(-1, 3.3, n_alphas)
R2_alphas = []
for i in range(len(alphas)):
    clf = Ridge(alpha=alphas[i])
    clf.fit(np.asarray(X_training_set), np.asarray(y_training_set))
    y_predict = clf.predict(np.asarray(X_testing_set))
    R2_alphas.append(clf.score(np.asarray(X_testing_set),np.
 ↪asarray(y_testing_set)))

ax = plt.gca()
ax.plot(alphas, R2_alphas)
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('R2')
plt.axis('tight')
plt.show()

# ax2 = plt.gca()
# ax2.plot(alphas, R2_alphas)
# # ax2.set_xscale('log')
# plt.xlabel('alpha')
# plt.ylabel('R2')
# plt.axis('tight')
# plt.show()
print(f"Meilleure valeur de coefficient de determination (R2):␣
 ↪{max(R2_alphas)}")
print(f"Meilleure valeur de l'hyperparamètre (alpha): {alphas[R2_alphas.
 ↪index(max(R2_alphas))]}")
# print(R2_alphas.)

sum_error = 0
clf = Ridge(alphas[R2_alphas.index(max(R2_alphas))])
clf.fit(np.asarray(X_training_set), np.asarray(y_training_set))
y_predict = clf.predict(np.asarray(X_testing_set))
```

```python
for i in range(len(y_predict)):
    sum_error += abs(y_predict[i]-y_testing_set[i])
mean_error = sum_error / len(y_predict)
print(f"Moyenne d'écarts d'erreur: {mean_error[0]}")

# fin code ridge regression
```




```
Meilleure valeur de coefficient de determination (R2): 0.7058754489237664
Meilleure valeur de l'hyperparamètre (alpha): 533.9831187409262
Moyenne d'écarts d'erreur: [[124676.31320091]]
```

Au vu du modèle de ridge regression, il existe bien un alpha qui maximise le coefficient de détermination. Cette valeur de alpha permet d'avoir un modèle avec un compromis entre biais et variance. Le modèle de Ridge Regression ne semble pas forcément beaucoup mieux que le modèle de régression linéaire. Nous allons donc entrainer notre jeu de données à un modèle de regression ridge à noyau dans l'étape suivante.

4 Modèle de régression ridge à noyau

Nous effectuons une Régression Ridge à noyau Gaussien.

Nous séparons tout d'abord un training set et un testing set. Nous décidons ici de réduire la taille de nos training et testing set, car les temps de calculs étaient trop long. Il sera envisageable de commenter les lignes correspondantes à la réduction de taille des set et de laisser tourner le programme pendant toute une journée.

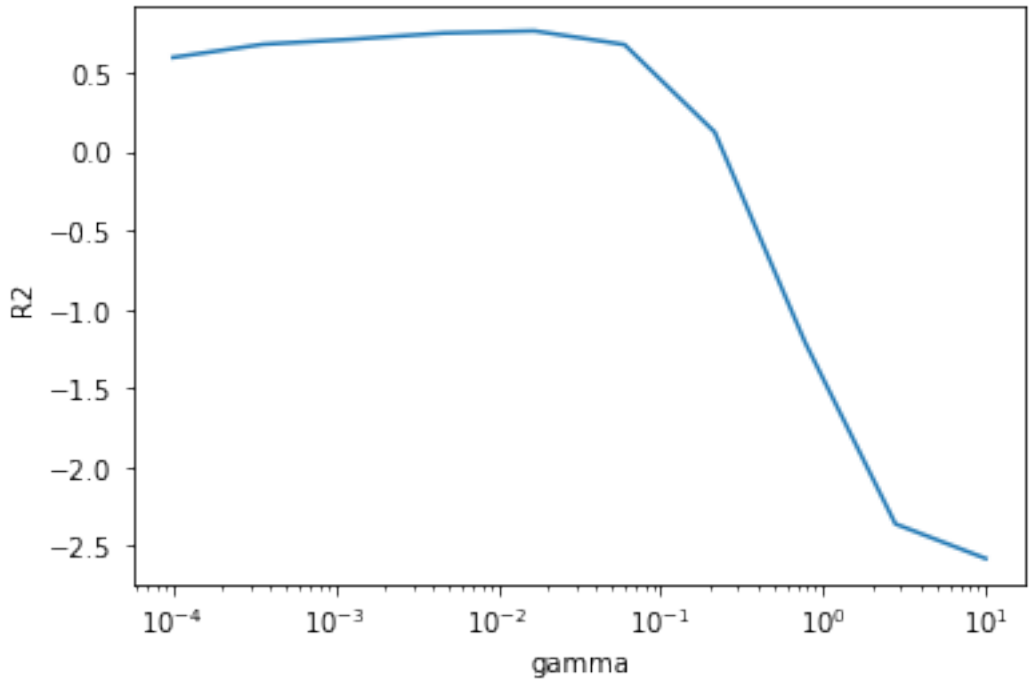
[ ]:
```
import matplotlib.pyplot as plt
from sklearn.linear_model import Ridge
from sklearn.kernel_ridge import KernelRidge
from sklearn.model_selection import train_test_split
from sklearn import preprocessing
import numpy as np
import pandas as pd


# Shuffle des données
df = df.iloc[np.random.permutation(len(df))]

# On décompose le dataset et on le transforme en matrices pour pouvoir␣
 ↪effectuer notre calcul
X = np.matrix([np.ones(df.shape[0]), df['date'].values,  df['bedrooms'].values,␣
 ↪df['bathrooms'].values, df['sqft_living'].values, df['floors'].values,␣
 ↪df['waterfront'].values, df['view'].values, df['condition'].values,
              df['grade'].values, df['sqft_above'].values, df['sqft_basement'].
 ↪values, df['yr_built'].values, df['yr_renovated'].values, df['zipcode'].
 ↪values, df['lat'].values, df['long'].values, df['sqft_living15'].values]).T
y = np.matrix(df['price']).T


# Training set (80% des valeurs)
# Testing set (20% des valeurs)
X_training_set, X_testing_set, y_training_set, y_testing_set = train_test_split(
    X, y, train_size=0.8)

X_training_set = X_training_set[1:4000]
y_training_set = y_training_set[1:4000]


X_testing_set = X_testing_set[1:1000]
y_testing_set = y_testing_set[1:1000]


X_training_set_norm = np.asarray(X_training_set)
scaler = preprocessing.StandardScaler().fit(X_training_set_norm)
X_training_set_norm = scaler.transform(X_training_set_norm)
X_training_set = np.asmatrix(X_training_set_norm)


X_testing_set_norm = np.asarray(X_testing_set)
scaler = preprocessing.StandardScaler().fit(X_testing_set_norm)
X_testing_set_norm = scaler.transform(X_testing_set_norm)
X_testing_set = np.asmatrix(X_testing_set_norm)
```

Maintenant, nous décidons de régler nos hyper-paramètres. Dans un premier temps, nous allons essayer de les trouver par essais, en utilisant la cross-validation. Nous trouvons d'abord le meilleur Gamma (hyper-paramètre représentant l'inverse du rayon d'influence des échantillons), en fixant Alpha (hyper-paramètre de régularisation) à 1.

[ ]:
```
from sklearn.model_selection import cross_val_score

n_gammas = 10
gammas = np.logspace(-4, 1, n_gammas)
R2_gammas = []
for i in range(len(gammas)):

    clf = KernelRidge(alpha=1, kernel="rbf", gamma = gammas[i])
    scores = cross_val_score(clf, np.asarray(X_training_set), np.
 ↪asarray(y_training_set), cv=5)
    print(scores)
    print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(),␣
 ↪scores.std()))
    R2_gammas.append(scores.mean())


for i in range(len(gammas)):
    print(f"{gammas[i]}: {R2_gammas[i]}")
ax = plt.gca()
ax.plot(gammas, R2_gammas)
ax.set_xscale('log')
plt.xlabel('gamma')
plt.ylabel('R2')
plt.axis('tight')
plt.show()
# fin code ridge regression



maximum = max(R2_gammas)
GammaMax = gammas[R2_gammas.index(maximum)]
print("MAX : "+str(maximum)+", at : "+str(GammaMax))
```

```
[0.60076937 0.55791312 0.57038309 0.6429502  0.6246555 ]
0.60 accuracy with a standard deviation of 0.03
[0.68671478 0.65022429 0.6714335  0.71342507 0.69192329]
0.68 accuracy with a standard deviation of 0.02
[0.71911123 0.69285952 0.71687695 0.73529172 0.7131779 ]
0.72 accuracy with a standard deviation of 0.01
[0.75862478 0.73441022 0.76370042 0.76498343 0.74819342]
0.75 accuracy with a standard deviation of 0.01
[0.78012984 0.72587585 0.79060713 0.79242822 0.74943623]
```

```
0.77 accuracy with a standard deviation of 0.03
[0.69229467 0.58172131 0.69041871 0.75793323 0.68228329]
0.68 accuracy with a standard deviation of 0.06
[0.10123709 0.11839926 0.05446459 0.18128175 0.17054045]
0.13 accuracy with a standard deviation of 0.05
[-1.1880819  -1.0146767  -1.17281199 -1.39725294 -1.26528148]
-1.21 accuracy with a standard deviation of 0.12
[-2.38199947 -2.01454409 -2.00561925 -2.85288857 -2.53903396]
-2.36 accuracy with a standard deviation of 0.32
[-2.62314378 -2.1862493  -2.16454844 -3.14571357 -2.77201823]
-2.58 accuracy with a standard deviation of 0.37
9.999999999999999e-05: 0.5993342560269173
0.00035938136638046257: 0.6827441884266268
0.001291549665014884: 0.7154634639935802
0.004641588833612782: 0.7539824529475492
0.01668100537200059: 0.7676954552868399
0.05994842503189409: 0.6809302435167337
0.21544346900318845: 0.12518462825209037
0.7742636826811277: -1.2076209993679523
2.782559402207126: -2.3588170671303423
10.0: -2.57833466716407
```



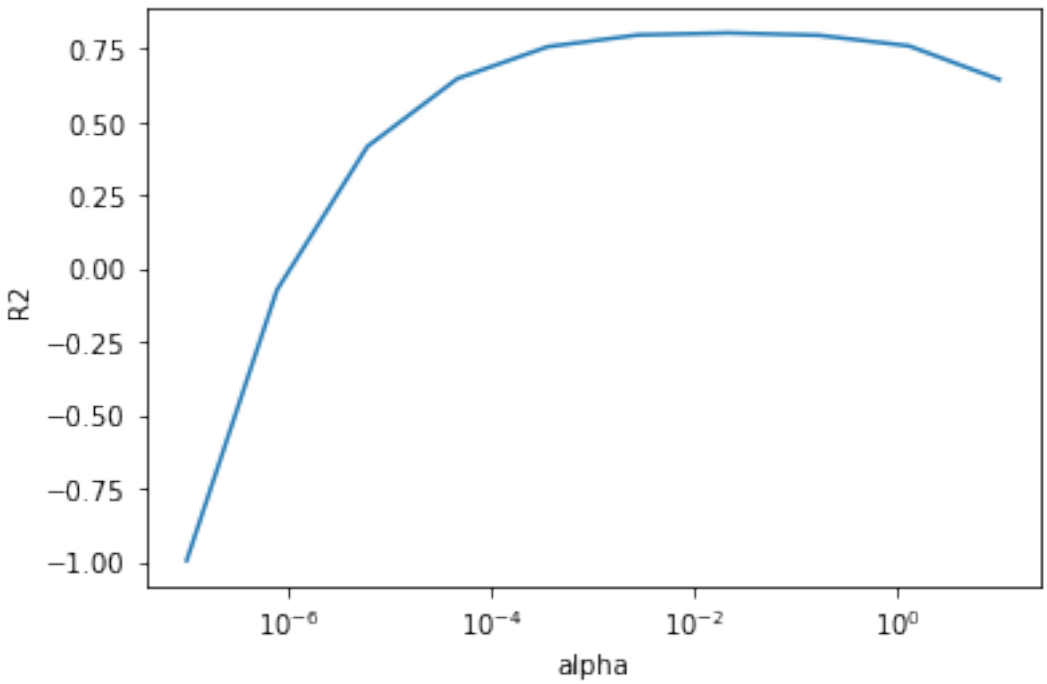

```
MAX : 0.7676954552868399, at : 0.01668100537200059
```

Nous réglons maintenant Alpha, en utilisant le Gamma optimal trouvé à l'étape précédente.

[ ]:
```
n_alphas = 10
alphas = np.logspace(-7, 1, n_alphas)
R2_alphas = []
for i in range(len(alphas)):
    clf = KernelRidge(alpha=alphas[i], kernel="rbf", gamma = GammaMax)
    scores = cross_val_score(clf, np.asarray(X_training_set), np.
 ↪asarray(y_training_set), cv=5)
    print(scores)
    print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(),␣
 ↪scores.std()))
    R2_alphas.append(scores.mean())


for i in range(len(alphas)):
    print(f"{alphas[i]}: {R2_alphas[i]}")
print(max(R2_alphas))
ax = plt.gca()
ax.plot(alphas, R2_alphas)
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('R2')
plt.axis('tight')
plt.show()


maximum = max(R2_alphas)
valuemax = alphas[R2_alphas.index(maximum)]
print("MAXIMUM : "+str(maximum)+", at : "+str(valuemax))




clf = KernelRidge(alpha=valuemax, kernel="rbf", gamma = GammaMax)
scores = cross_val_score(clf, np.asarray(X_training_set), np.
 ↪asarray(y_training_set), cv=5)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(),␣
 ↪scores.std()))
R2_alphas.append(scores.mean())
```

```
[-0.61773967 -0.08124243 -1.59576445 -1.39059053 -1.29307682]
-1.00 accuracy with a standard deviation of 0.56
[ 0.05605206  0.35103321 -0.34477481 -0.26054688 -0.16476459]
-0.07 accuracy with a standard deviation of 0.25
[0.48818873 0.58514869 0.31669604 0.32517401 0.36784252]
```

```
0.42 accuracy with a standard deviation of 0.10
[0.67413687 0.71635143 0.66293097 0.56346773 0.62819524]
0.65 accuracy with a standard deviation of 0.05
[0.77605444 0.79034569 0.79241367 0.68812885 0.74071194]
0.76 accuracy with a standard deviation of 0.04
[0.80214231 0.81197614 0.82793386 0.76153698 0.78370226]
0.80 accuracy with a standard deviation of 0.02
[0.80736028 0.80674048 0.8319795  0.78639675 0.7905015 ]
0.80 accuracy with a standard deviation of 0.02
[0.80121348 0.78245027 0.82346797 0.79589207 0.7789377 ]
0.80 accuracy with a standard deviation of 0.02
[0.77353894 0.71271537 0.78176649 0.78907799 0.74377476]
0.76 accuracy with a standard deviation of 0.03
[0.64788671 0.54758866 0.64694822 0.71313885 0.67150019]
0.65 accuracy with a standard deviation of 0.05
1e-07: -0.9956827823550084
7.742636826811278e-07: -0.07260020175073803
5.994842503189409e-06: 0.41660999753718536
4.641588833612773e-05: 0.649016447813802
0.00035938136638046257: 0.7575309185163308
0.002782559402207126: 0.797458308673402
0.021544346900318822: 0.80459569987815
0.16681005372000557: 0.796392299497052
1.2915496650148826: 0.7601747092482485
10.0: 0.6454125273175719
0.80459569987815
```



```
MAXIMUM : 0.80459569987815, at : 0.021544346900318822
[0.80736028 0.80674048 0.8319795  0.78639675 0.7905015 ]
0.80 accuracy with a standard deviation of 0.02
```

Nous trouvons ici une précision d'environ 0.80. Cette valeur est bien meilleure que pour les modèles de regression linéaire et régression ridge.

Nous décidons, maintenant que nous avons un ordre d'idée des hyper-paramètres optimaux, de réaliser une GridSearch plus précise sur le jeu de données, en parcourant les paramètres Alpha et Gamma, avec une 5-fold cross-validation.

[ ]:
```
from sklearn.model_selection import GridSearchCV
n_gammas = 3
gammas = GammaMax*np.logspace(-0.1, +0.1, n_gammas)
n_alphas = 3
alphas = valuemax*np.logspace(-0.1, +0.1, n_alphas)
parameters = {'gamma': gammas, 'alpha':alphas}
grid = GridSearchCV(clf, parameters, verbose=2, return_train_score=True)
grid.fit(np.asarray(X_training_set), np.asarray(y_training_set))
grid.best_params_

clf = KernelRidge(alpha=grid.best_params_['alpha'], kernel="rbf", gamma = grid.
 ↪best_params_['gamma'])
scores = cross_val_score(clf, np.asarray(X_training_set), np.
 ↪asarray(y_training_set), cv=5)
print(scores)
print("%0.2f accuracy with a standard deviation of %0.2f" % (scores.mean(),␣
 ↪scores.std()))
```

```
Fitting 5 folds for each of 9 candidates, totalling 45 fits
[CV] END alpha=0.017113283041617796, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.01668100537200059; total time=
0.5s
[CV] END alpha=0.017113283041617796, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.01668100537200059; total time=
```

```
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.017113283041617796, gamma=0.021000141557086554; total time=
0.5s
[CV] END alpha=0.021544346900318822, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.013250193550567484; total time=
0.5s
[CV] END alpha=0.021544346900318822, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.021544346900318822, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.013250193550567484; total time=
```

```
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.013250193550567484; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.013250193550567484; total time=
0.5s
[CV] END alpha=0.02712272579332026, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.01668100537200059; total time=
0.5s
[CV] END alpha=0.02712272579332026, gamma=0.01668100537200059; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.021000141557086554; total time=
0.4s
[CV] END alpha=0.02712272579332026, gamma=0.021000141557086554; total time=
0.4s
[0.80740568 0.8164716  0.83200617 0.78323188 0.79092298]
0.81 accuracy with a standard deviation of 0.02
```

Nous avons pu réaliser ici, avec une Régression Ridge à noyau Gaussien, faire une première recherche d'ordre de grandeur des hyper-paramètres Alpha et Gamma un à un, puis effectuer une recherche plus précise, en effectuant une GridSearch. Nous avons calculés tous les résultats avec une 5-fold Cross-Validation.

5 Modèle Random Forest

Idem, que précédemment, nous allons appliquer le modèle Random Forest (non vu en cours). Nous allons essayer d'avoir un modèle performant en analysant le résultat avec différentes profondeurs d'algorithmes. Nous analyserons le coefficient de détermination, la moyenne d'erreur sur le training test et le temps d'exécution pour chaque profondeur.

[ ]:
```python
from sklearn.ensemble import RandomForestRegressor
from datetime import datetime
import timeit
import time

# "Shuffle" des données
df = df.iloc[np.random.permutation(len(df))]
```

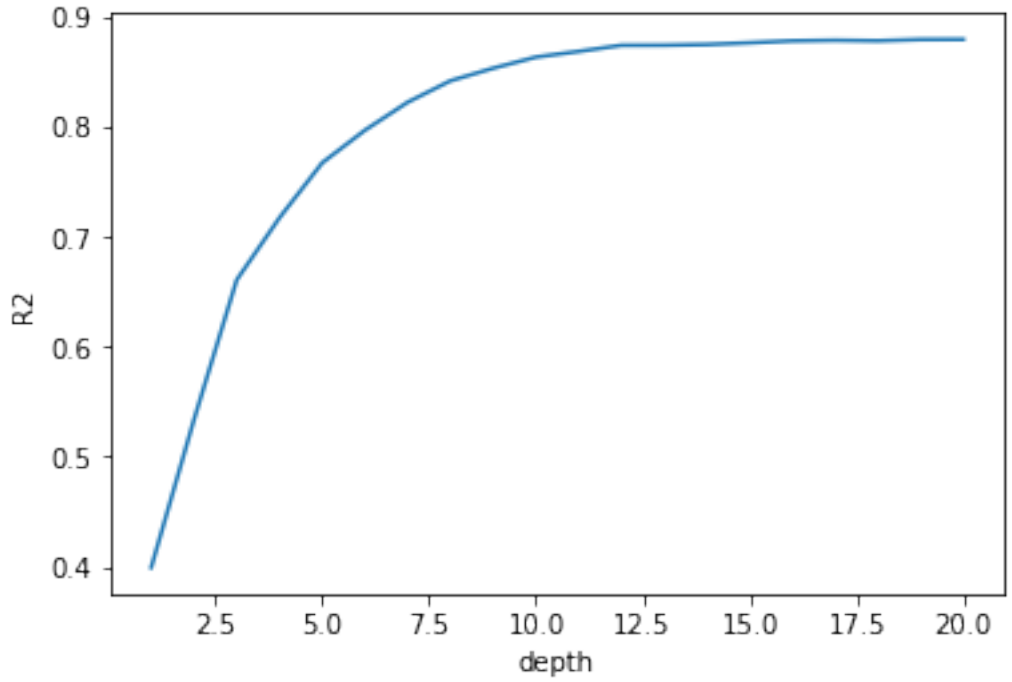
```
# On décompose le dataset et on le transforme en matrices pour pouvoir␣
 ↪effectuer notre calcul
X = np.matrix([np.ones(df.shape[0]), df['date'].values, df['sqft_lot'].values␣
 ↪,df['bedrooms'].values, df['bathrooms'].values, df['sqft_living'].values,␣
 ↪df['floors'].values, df['waterfront'].values, df['view'].values,␣
 ↪df['condition'].values, df['grade'].values, df['sqft_above'].values,␣
 ↪df['sqft_basement'].values, df['yr_built'].values, df['yr_renovated'].
 ↪values,df['zipcode'].values,df['lat'].values,df['long'].values,␣
 ↪df['sqft_living15'].values]).T
y = np.matrix(df['price']).T


X = np.asarray(X)
scaler = StandardScaler().fit(X)
X = scaler.transform(X)


# Training set (80% des valeurs)
# Testing set (20% des valeurs)
X_training_set, X_testing_set, y_training_set, y_testing_set =␣
 ↪train_test_split(X, y, train_size=0.8)

nb_depths = 20
depths = []
scores = []
means = []
clocks = []
for i in range(nb_depths):
    depths.append(i+1)
    start = timeit.default_timer()
    regr = RandomForestRegressor(max_depth=i+1, random_state=0)
    regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))

    # Score of the model
    scores.append(regr.score(np.asarray(X_testing_set), np.
 ↪asarray(y_testing_set)))
    # print(score)

    # moyenne d'écarts
    sum_error = 0
    y_predict = regr.predict(np.asarray(X_testing_set))
    for i in range(len(y_predict)):
        sum_error += int(abs(y_predict[i]-y_testing_set[i]))
    means.append(sum_error / len(y_predict))
    end = timeit.default_timer()
    clocks.append(end-start)
ax = plt.gca()
```

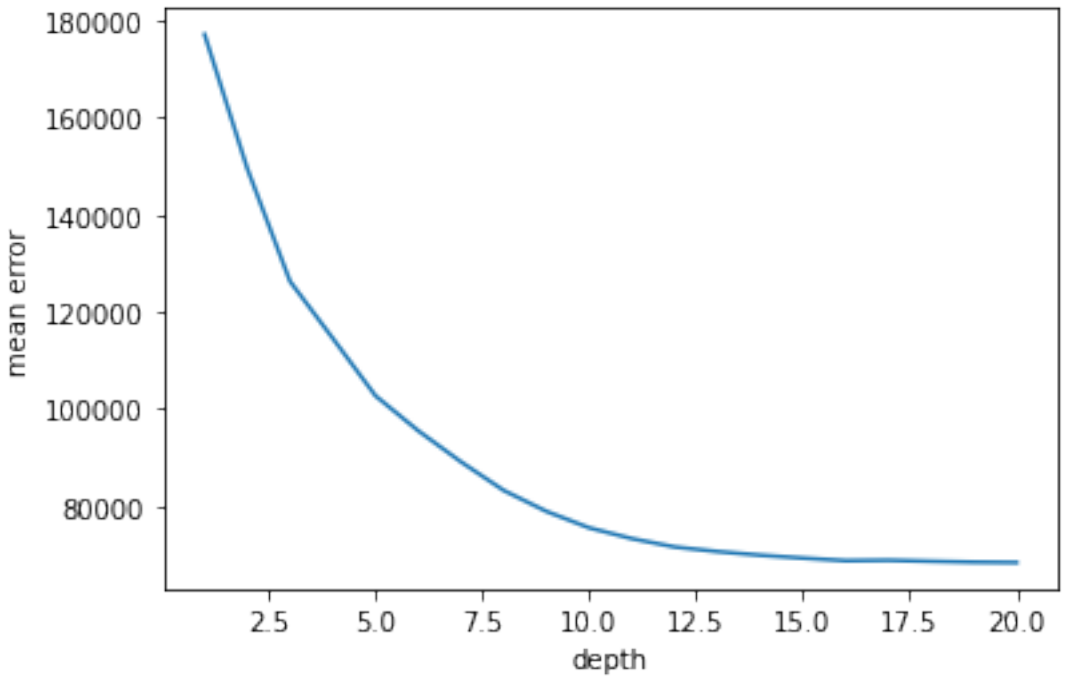
```python
ax.plot(depths, scores)
plt.xlabel('depth')
plt.ylabel('R2')
plt.axis('tight')
plt.show()


ax = plt.gca()
ax.plot(depths, means)
plt.xlabel('depth')
plt.ylabel('mean error')
plt.axis('tight')
plt.show()

ax = plt.gca()
ax.plot(depths, clocks)
plt.xlabel('depth')
plt.ylabel('time (s)')
plt.axis('tight')
plt.show()

print(f"Avec la plus grande profondeur ({nb_depths}) : \nR2 =␣
 ↪{scores[nb_depths-1]}\nmean_error = {means[nb_depths-1]}$\ntime_learning =␣
 ↪{clocks[nb_depths-1]}s")
```

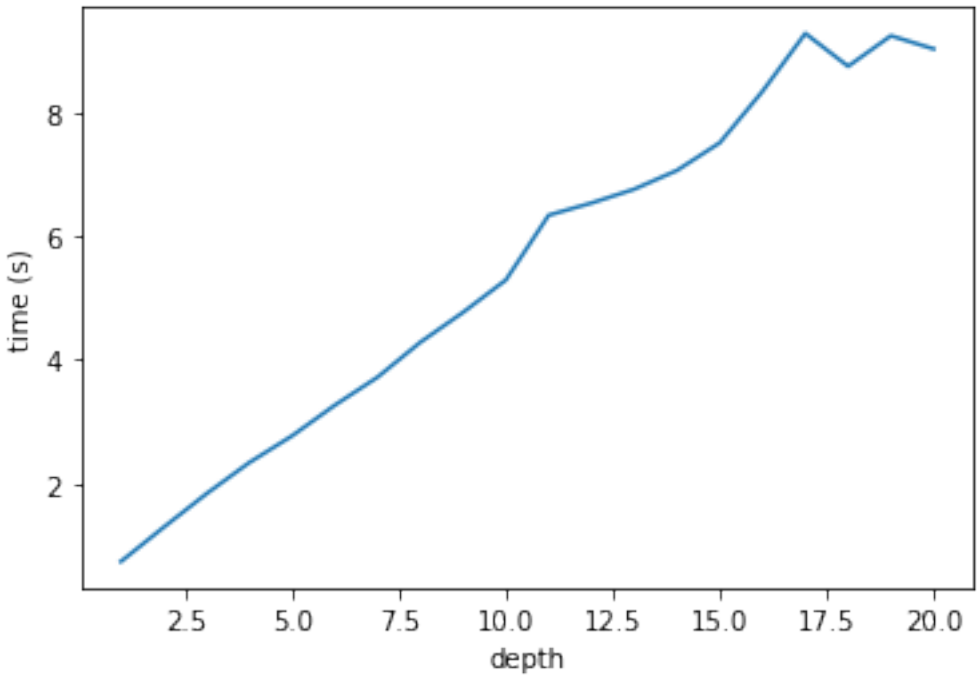
```
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
```

```
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
```

```
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
/tmp/ipykernel_1950/245567927.py:32: DataConversionWarning: A column-vector y
was passed when a 1d array was expected. Please change the shape of y to
(n_samples,), for example using ravel().
  regr.fit(np.asarray(X_training_set), np.asarray(y_training_set))
```








```
Avec la plus grande profondeur (20) :
R2 = 0.8790918716388102
mean_error = 68365.75474317446$
```

```
time_learning = 9.03159774300002s
```

De toute évidence, plus la profondeur de l'algorithme Random Forest permet d'avoir une meilleure prédiction dans notre cas. On arrive à un coefficient de détermination acceptable. L'un des grand avantage de cette algorithme c'est qu'il est linéaire en temps d'apprentissage. Ce modèle d'entrainement donne le meilleur score en tant que modèle de prédiction. Cela signifie que pour la prédiction de prix des maisons, il est largement préférable d'utiliser ce modèle parmi tout ceux testés.

6 Conclusion

Pour conclure, durant ce TP de Machine Learning de deux séances, nous avons appliquer une démarche scientifique en datascience (récupération, nettoyage, exploration, modélisation, évaluation et interprétation). Nous avons aussi appris à utiliser différents algorithmes d'apprentissage (apprentissage supervisé) appliqués pour de la régression. Il a été ainsi possible de faire de la prédiction. Parmi les modèles appliqués, nous avons travaillé sur des modèles paramétriques et non-paramétriques. Nous avons globalement divisé notre jeu de données, pour évaluer nos modèles, en jeux de données d'entrainement et de test. De plus, nous avons aussi utilisé la méthode de Cross-Validation dans le cas du modèle de Ridge Regression à noyau.

Pour ce qui est des outils utilisés pour ce TP de Machine Learning, nous avons appris à manier les notebooks Python (Jupyter). Nous avons aussi appris à utiliser les bibliothèques Python de l'écosystème Spicy dont pandas (pour tableaux et Dataframes), numpy (matrice), matplotlib (pour les graphes), iPython (feuilles de calcul). La bibliothèque pour utiliser les algorithmes est Scikit-learn (même si l'outil Tensorflow est omniprésent dans le monde du Machine Learning).