H4221-ML-Notebook

April 28, 2023

1 H4221 - Rapport Machine Learning

Au cours du TP de Machine Learning, 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

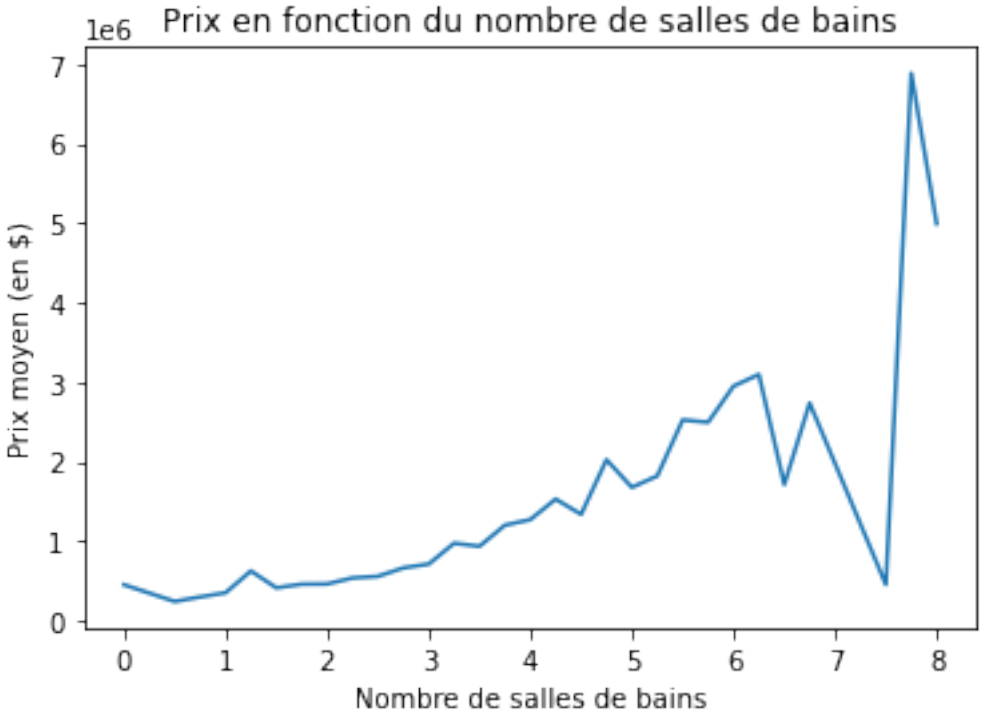
[ ]:
```
import matplotlib.pyplot as plt


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)
```

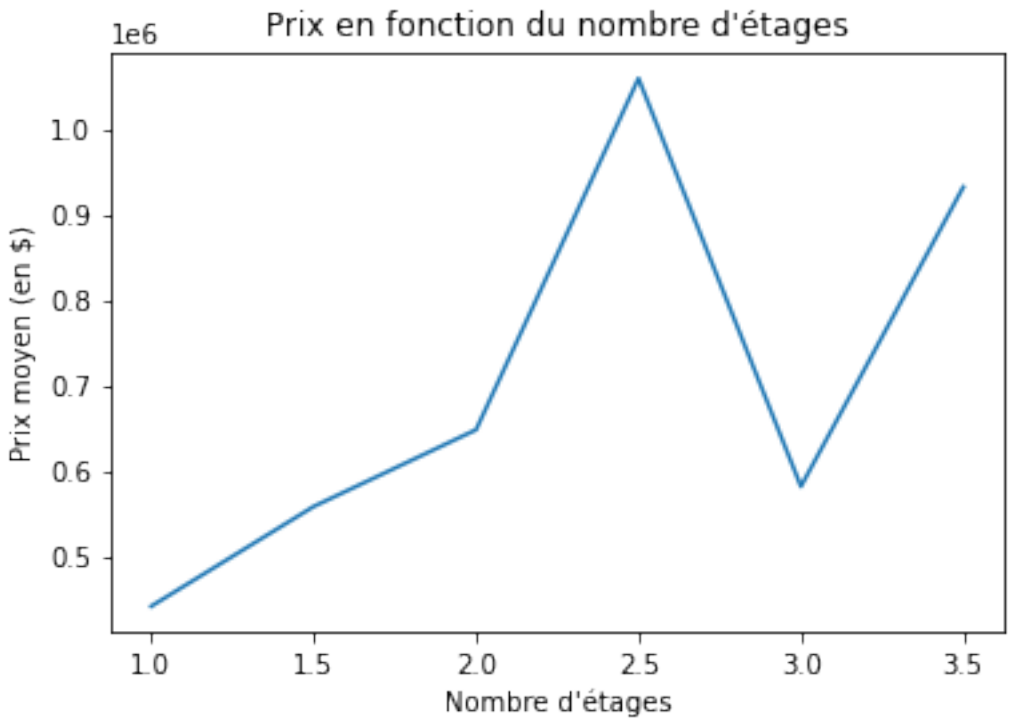
```
dfpbyb = dfsorted.groupby("sqft_lot")["price"].mean()

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()
```

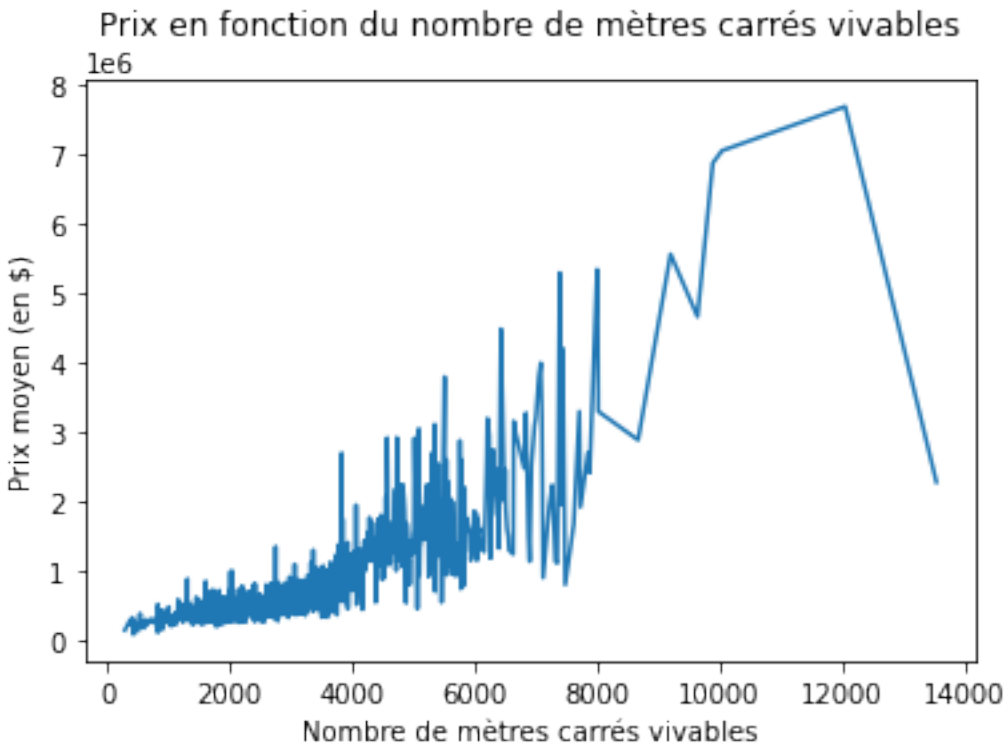
```
/tmp/ipykernel_414/3840609229.py:5: SettingWithCopyWarning:


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
```



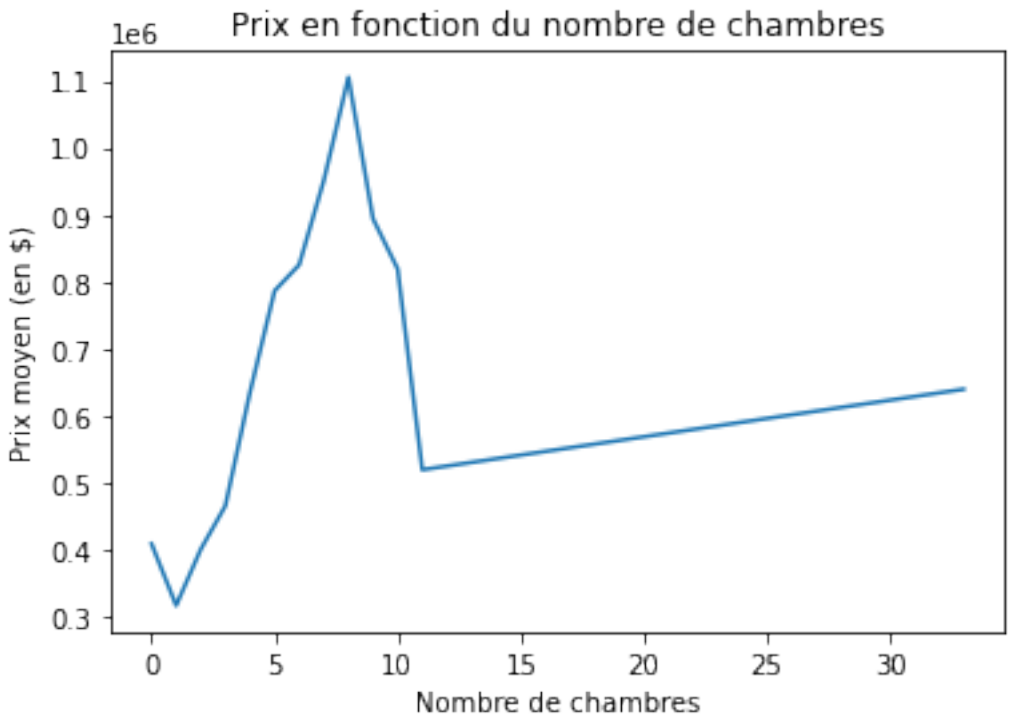

```
/tmp/ipykernel_414/3840609229.py:18: SettingWithCopyWarning:


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
```



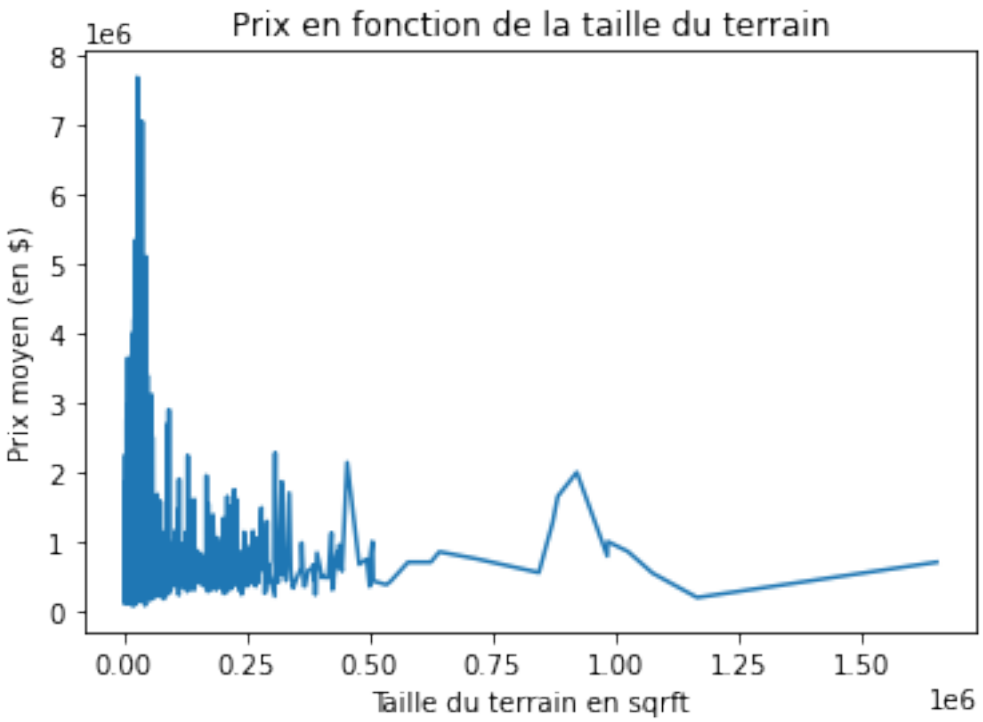

```
/tmp/ipykernel_414/3840609229.py:32: SettingWithCopyWarning:


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
```



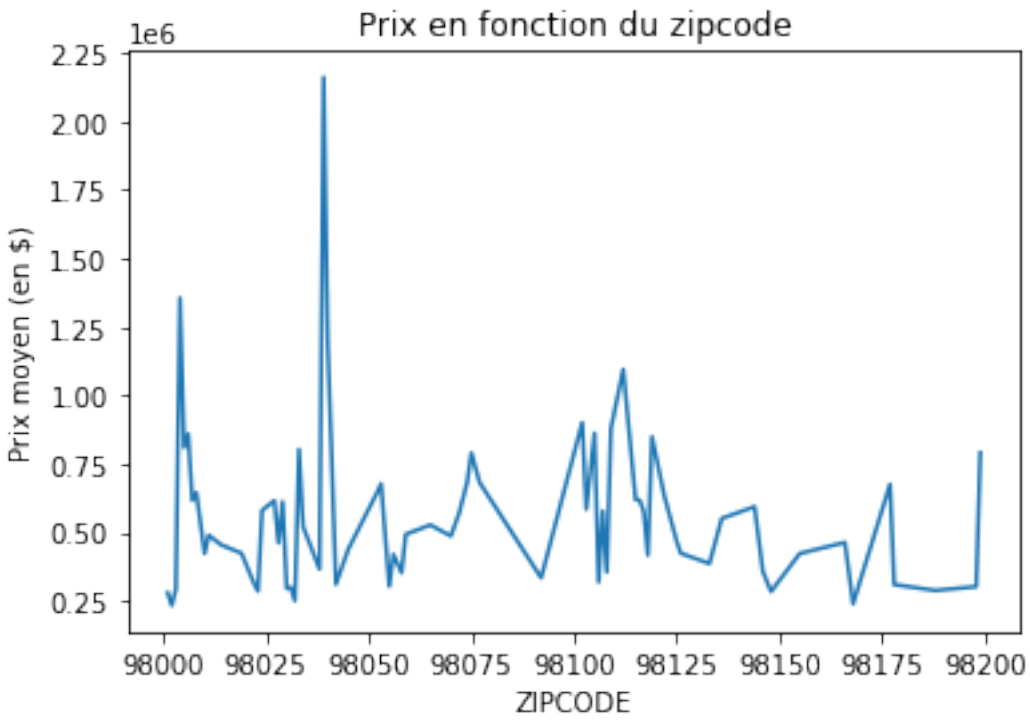

```
/tmp/ipykernel_414/3840609229.py:46: SettingWithCopyWarning:


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
```



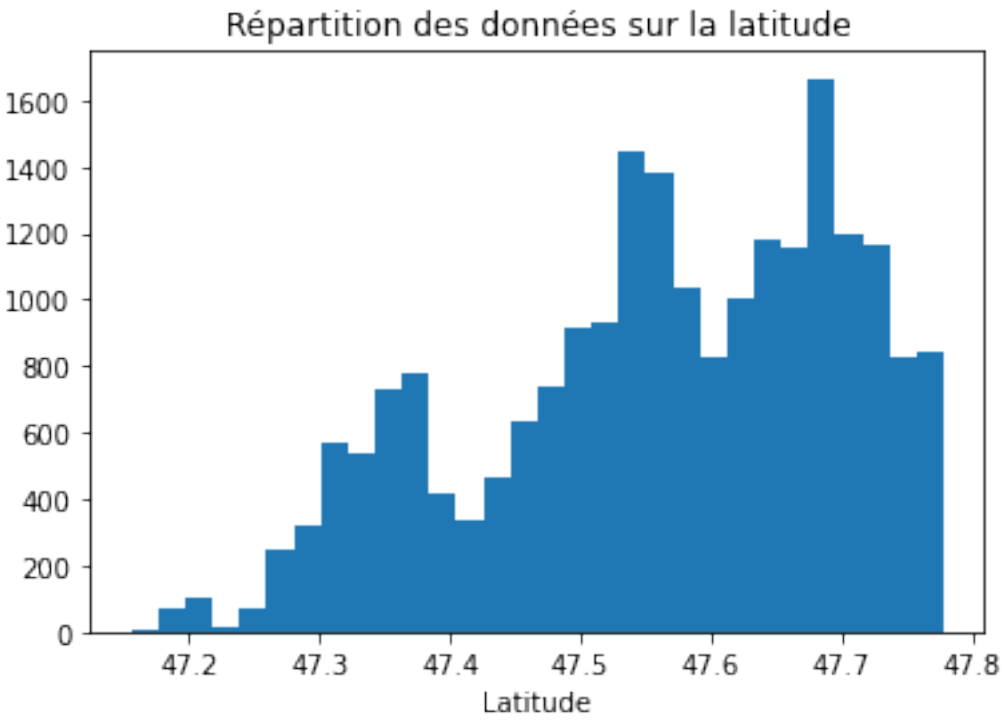

```
/tmp/ipykernel_414/3840609229.py:60: SettingWithCopyWarning:


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
```



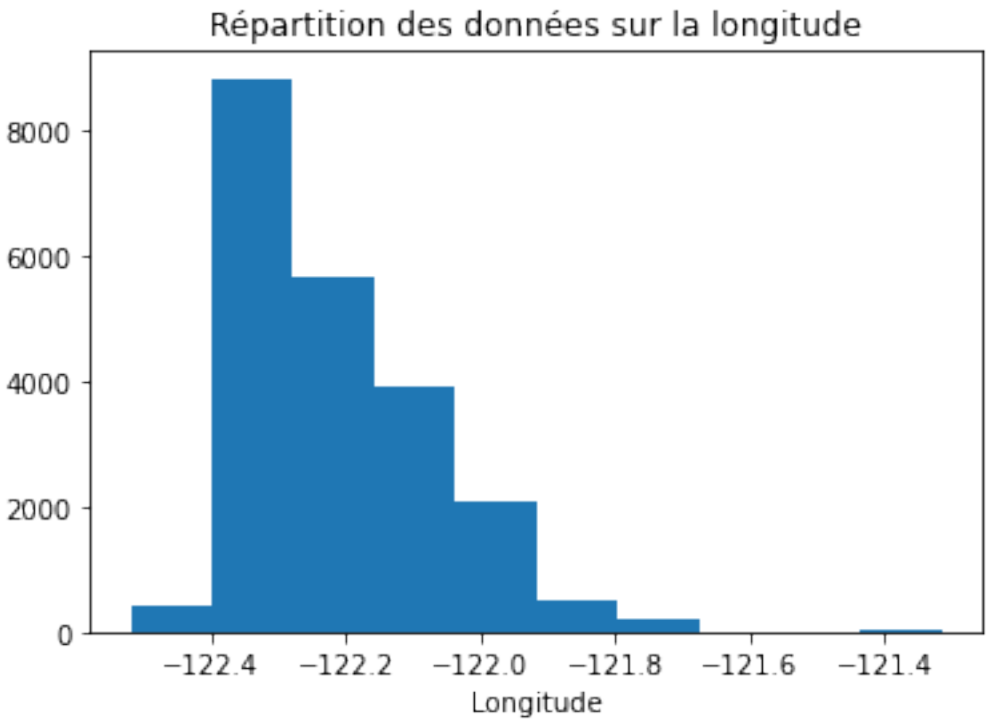

```
/tmp/ipykernel_414/3840609229.py:73: SettingWithCopyWarning:


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
```




```
/tmp/ipykernel_414/3840609229.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
```




```
/tmp/ipykernel_414/3840609229.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
```



[ ]:
```
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"There is {len(df.loc[df['date'] == 0])} raws with date not between␣
 ↪May-2014 and May-2015")
```

```
There is 0 raws with date not between May-2014 and May-2015
```

2 Modèle de régression linéaire

[blabla à compléter]

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)
```

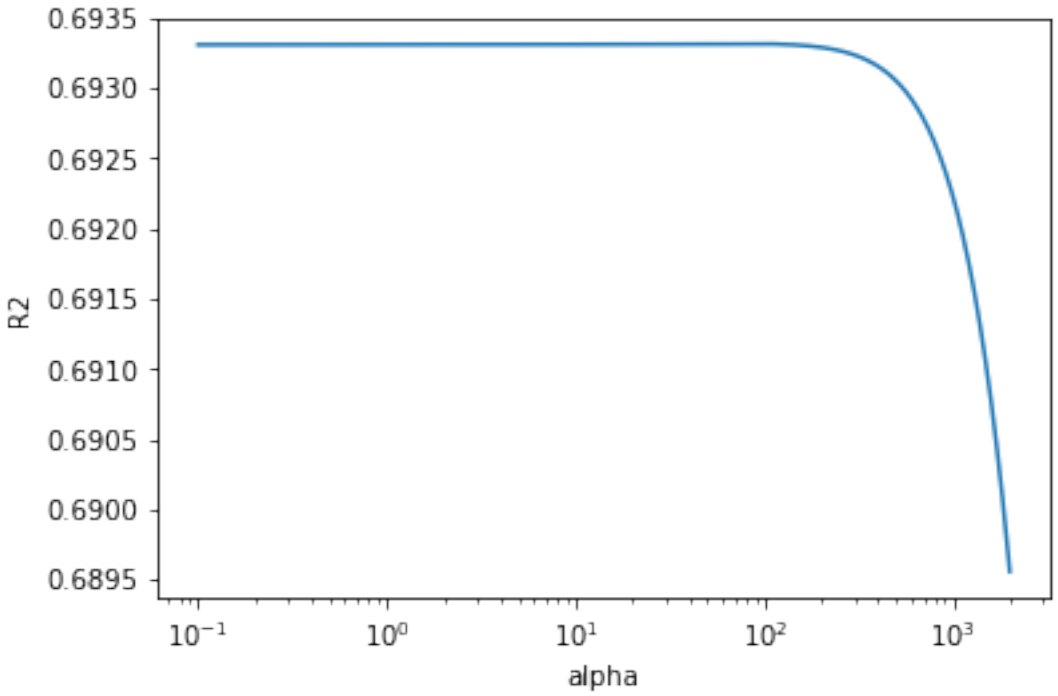
Pour voir l'efficacité de ce modèle, nous allons diviser notre jeu de données en deux: un "training test" et un "testing set" Le "training set" va permettre d'apprendre pour répondre à notre tâche de prédiction de prix. Le "testing set" va permet de mesurer l'erreur de prédiction de prix des maisons 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 alpha: {max(R2_alphas)}")
print(f"Meilleure valeur de 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))
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}")

# fin code ridge regression
```




```
Meilleure valeur de alpha: 0.6933191244726663
Meilleure valeur de alpha: 70.00759286182006
Moyenne d'écarts d'erreur: [[123388.10144538]]
```

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.

4 Modèle de régression ridge à noyau

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.

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

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

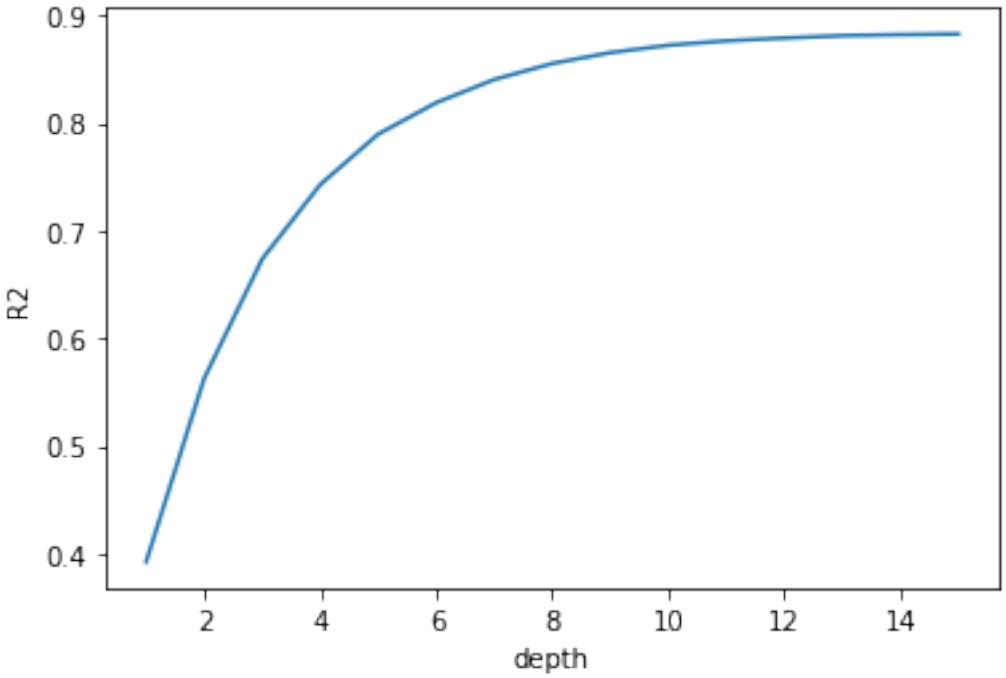
```
# On décompose le dataset et on le transforme en matrices pour pouvoir␣
 ↪effectuer notre calcul
X = np.matrix([np.ones(df.shape[0]), df['date'].values, df['sqft_lot'].values␣
 ↪,df['bedrooms'].values, df['bathrooms'].values, df['sqft_living'].values,␣
 ↪df['floors'].values, df['waterfront'].values, 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 = 15
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()
```

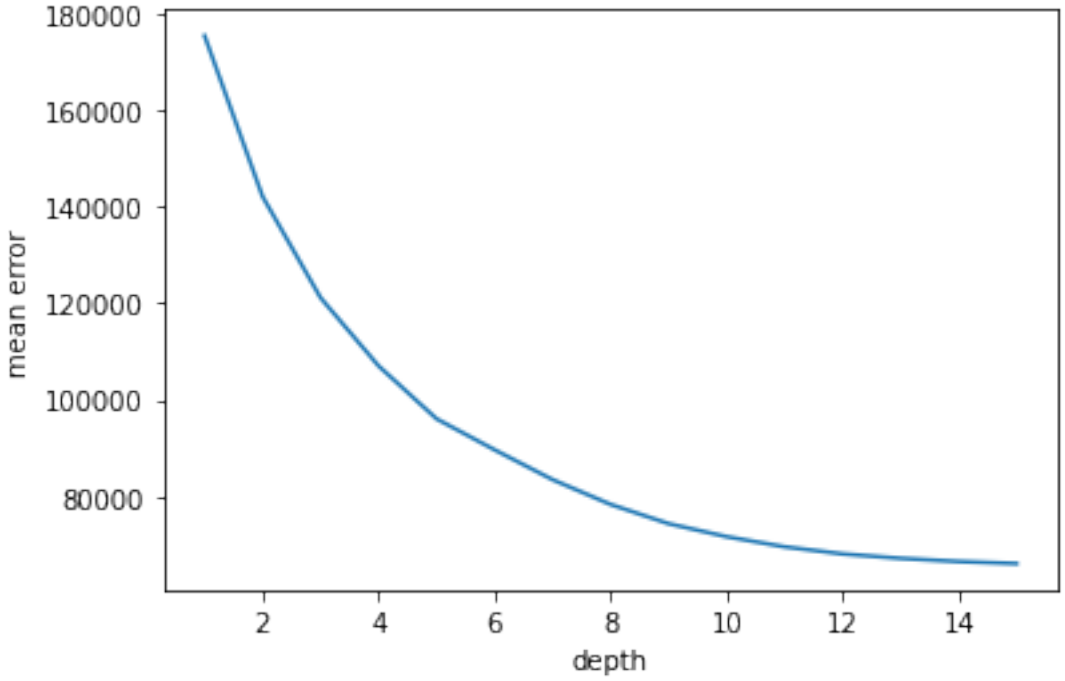
```
ax.plot(depths, scores)
plt.xlabel('depth')
plt.ylabel('R2')
plt.axis('tight')
plt.show()


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")
```

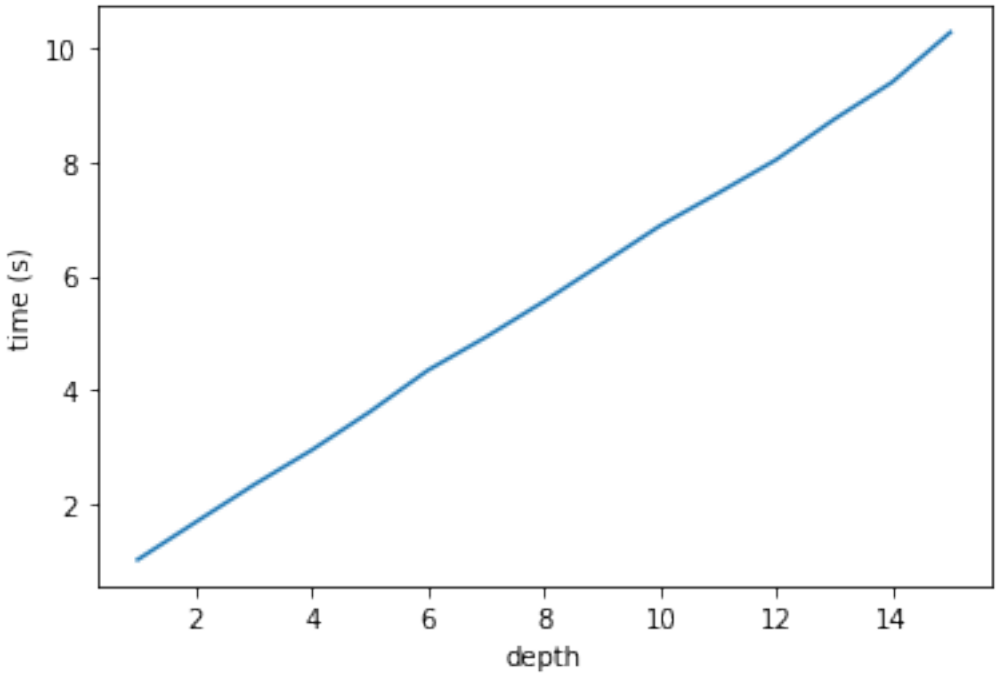
```
/tmp/ipykernel_414/469520333.py:32: DataConversionWarning:

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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().

/tmp/ipykernel_414/469520333.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().
```

```
/tmp/ipykernel_414/469520333.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().
```








```
Avec la plus grande profondeur (15) :
R2 = 0.8832149559509259
mean_error = 65958.54558074965$
time_learning = 10.287508613000227s
```

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. Un grand avantage de cette algorithme c'est qu'il est linéaire en temps d'apprentissage.

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 aussi utiliser la méthode de Cross-Validation dans le cas du modèle de Ridge Regression.

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).