Devoir_Python_caleb

February 5, 2021

1 Devoir de Python

Caleb KASHALA ILUNGA

1.1 Table de matiére

- Partie scrapping
- Partie nettoyage
- Partie descriptive
- partie apprentissage machine learning

1.1.1 Problématique: Comment estimer le meilleur prix possible en fonction de différentes variables présentées?

[1]:
```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
from time import sleep, time
from selenium.common.exceptions import StaleElementReferenceException
import pandas as pd

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import scipy.stats
import seaborn as sns

from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet
from sklearn.neighbors import KNeighborsRegressor
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor
from sklearn.neural_network import MLPRegressor

from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import LabelBinarizer
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import OneHotEncoder

from sklearn.impute import SimpleImputer
from sklearn.impute import MissingIndicator

from sklearn.linear_model import SGDClassifier

from sklearn.compose import make_column_transformer
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer

from sklearn.model_selection import cross_val_score

import matplotlib.pyplot as plt
import numpy as np

from sklearn.datasets import fetch_openml
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.inspection import permutation_importance
from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder

from sklearn import model_selection as ms
```

2 PARTIE SCRAPPING

[ ]:
```
driver = webdriver.Firefox()
```

[ ]:
```
page=driver.get("https://fr.shopping.rakuten.com/nav/
 ↪Tel-PDA_Telephones-mobiles#xtatc=PUB-[PMC]-[E]-[smartphones-tablettes-objets-connectes]-[so
```

[ ]:
```
i=1
carac=[]
n=1
pk=[]
page=driver.get("https://fr.shopping.rakuten.com/nav/
 ↪Tel-PDA_Telephones-mobiles#xtatc=PUB-[PMC]-[E]-[smartphones-tablettes-objets-connectes]-[so
sleep(5)
driver.find_element_by_link_text("Plus tard").click()
sleep(5)
```

```
python_button = driver.find_elements_by_xpath("/html/body/div[2]/div/div/div/
 ↪div/div[2]/span")[0]
python_button.click()
while n <30:
    pk.append([driver.
 ↪find_elements_by_class_name("productList_layoutContent_NC_.pl-8.f14")])
    sleep(4)
    driver.find_element_by_link_text("Suivant").click()
    sleep(10)
    p=driver.find_elements_by_class_name("productList_layoutContent_NC_.pl-8.
 ↪f14")
    sleep(4)
    for i in range(len(p)):
        pa=p[i].find_element_by_class_name("silver.description_caption_3J5")

        if len(pa.find_elements_by_class_name("f12.description_bulletPoints_Vnz␣
 ↪li"))>1:
            try:
                reseau=([pa.find_elements_by_class_name("f12.
 ↪description_bulletPoints_Vnz li")[1].text])
            except:
                reseau="NA"
            try:
                ecran=([pa.find_elements_by_class_name("f12.
 ↪description_bulletPoints_Vnz li")[0].text])
            except:
                ecran="NA"
            try:
                exploit=([pa.find_elements_by_class_name("f12.
 ↪description_bulletPoints_Vnz li")[2].text])
            except:
                exploit="NA"
            try:
                prix=([p[i].find_element_by_class_name("f20.b").text])
            except:
                prix="NA"
            m=driver.find_elements_by_class_name("lh-title")
            marque=m[i].text
        else:
            pass

        Resultat={"Marque":marque,"reseau":reseau,"exploit":exploit,"prix":prix,
                  "écran":ecran}

        carac.append(Resultat)
```

```
    n=n+1
```

[ ]:
```
Data=pd.DataFrame(carac)
Data.to_csv("Data_Scrapping.csv")
```

- Après avoir lancé le scrapping nous obtenons une base de données avec différentes variables et modalités associées aux variables.
- Notre base de données est une base contenant des informations sur différentes marques et modèles de téléphone, dans cette base de données nous avons 1305 observations et 5 variables:
  - Marque
  - Réseau
  - Exploit
  - prix
  - Ècran

[2]:
```
#import re
df1=pd.read_csv("Data_Scrapping.csv")#df['Marque'] = df['Header'].str.
↪extract('([A-S]\w{0,})')
df1[1:10]
```

[2]:
```
   Unnamed: 0                                             Marque  \
1           1    OnePlus Nord N100 Dual SIM 64 Go Midnight Frost
2           2  Huawei P40 lite 128 Go Double SIM Vert crush (…
3           3             Xiaomi Poco X3 NFC Dual-SIM 64 Go Bleu
4           4                 Xiaomi Mi 10T Dual-SIM 128 Go Noir
5           5  Huawei P30 lite 128 Go (RAM 4 Go) Double SIM B…
6           6        Samsung Galaxy Note9 128 Go Double SIM Noir
7           7            Samsung Galaxy S9 64 Go Double SIM Noir
8           8       Huawei P20 lite 64 Go Double SIM Noir minuit
9           9         Samsung Galaxy A51 128 Go Double SIM Blanc

          reseau                         exploit          prix  \
1  ['Réseau 4G']   ['Mémoire 64 Go / RAM 4 Go']  ['172,58 €']
2  ['Réseau 4G']  ['Mémoire 128 Go / RAM 6 Go']  ['196,59 €']
3    ['Android']                 ['Double SIM']  ['203,79 €']
4    ['Android']                 ['Double SIM']  ['409,99 €']
5  ['Réseau 4G']  ['Mémoire 128 Go / RAM 4 Go']  ['214,50 €']
6  ['Réseau 4G']  ['Mémoire 128 Go / RAM 6 Go']     ['374 €']
7  ['Réseau 4G']   ['Mémoire 64 Go / RAM 4 Go']     ['270 €']
8  ['Réseau 4G']   ['Mémoire 64 Go / RAM 4 Go']     ['189 €']
9  ['Réseau 4G']  ['Mémoire 128 Go / RAM 4 Go']  ['289,99 €']

                         écran
1                ['Ecran 6.52"']
2                 ['Ecran 6.4"']
3   ['Mémoire 64 Go / RAM 6 Go']
```

```
4  ['Mémoire 128 Go / RAM 8 Go']
5                ['Ecran 6.15"']
6                 ['Ecran 6.4"']
7                 ['Ecran 5.8"']
8                ['Ecran 5.84"']
9                 ['Ecran 6.5"']
```

3 Partie Nettoyage

Après avoir créé notre base de données provenant du scrapping nous allons commencer par nettoyer notre base de données et apporter quelques modifications. Dans un premier temps nous commençons par transformer toutes nos variables en type "str" pour pouvoir facilement utiliser les fonctions extract etreplace.

[3]:
```
#str
df1['écran']=df1['écran'].astype(str)
df1['reseau']=df1['reseau'].astype(str)
df1['Marque']=df1['Marque'].astype(str)
df1['exploit']=df1['exploit'].astype(str)
df1['prix']=df1['prix'].astype(str)
```

[4]:
```
#str
#df['sexe'].astype('int')
df1.dropna ( axis = 0 ,  how = 'all' )
```

[4]:
```
      Unnamed: 0                                             Marque  \
0              0       Samsung Galaxy Note10 256 Go Double SIM Noir
1              1    OnePlus Nord N100 Dual SIM 64 Go Midnight Frost
2              2  Huawei P40 lite 128 Go Double SIM Vert crush (…
3              3             Xiaomi Poco X3 NFC Dual-SIM 64 Go Bleu
4              4                 Xiaomi Mi 10T Dual-SIM 128 Go Noir
…            …                                                   …
1300        1300                      Apple iPhone 6 Plus 16 Go Or
1301        1301   Samsung Galaxy A5 (2017) Dual SIM 32 Go Or sable
1302        1302                Samsung Galaxy Xcover 4 16 Go Noir
1303        1303                 Samsung Galaxy A5 (2017) 32 Go Or
1304        1304                    Google Pixel 3A XL 64 Go Blanc

                            reseau                         exploit  \
0                  ['Réseau 4G']  ['Mémoire 256 Go / RAM 8 Go']
1                  ['Réseau 4G']   ['Mémoire 64 Go / RAM 4 Go']
2                  ['Réseau 4G']  ['Mémoire 128 Go / RAM 6 Go']
3                    ['Android']                ['Double SIM']
4                    ['Android']                ['Double SIM']
…                              …                             …
1300               ['Réseau 4G']                    ['iOS 8']
```

```
1301  ['Android 6.0 (Marshmallow)']                ['Double SIM']
1302                  ['Réseau 4G']   ['Mémoire 16 Go / RAM 2 Go']
1303                  ['Réseau 4G']   ['Mémoire 32 Go / RAM 3 Go']
1304    ['Single-SIM (SIM unique)']                            nan

              prix                          écran
0     ['546,88 €']                ['Ecran 6.3"']
1     ['172,58 €']               ['Ecran 6.52"']
2     ['196,59 €']                ['Ecran 6.4"']
3     ['203,79 €']   ['Mémoire 64 Go / RAM 6 Go']
4     ['409,99 €']  ['Mémoire 128 Go / RAM 8 Go']
…            …                             …
1300  ['164,90 €']                ['Ecran 5.5"']
1301  ['159,98 €']                ['Ecran 5.2"']
1302  ['159,99 €']                  ['Ecran 5"']
1303  ['149,99 €']                ['Ecran 5.2"']
1304  ['280,60 €']            ['Android 9.0 Pie']

[1305 rows x 6 columns]
```

[5]:
```
import Nettoyage
```

Dans un premier temps nous commençons par nettoyer la variable marque, pour si faire nous avons fait en sorte de ne garder que les marquent des téléphones et non pas les modèles associés aux marques.
Ensuite nous avons mis toutes les variables en minuscule, pour facilement pouvoir séparer les données par marques.

[6]:
```
Nettoyage.nettoyage_marque(df1)
```

```
C:\Users\caleb\Desktop\Devoir python\Nettoyage.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
  fichier['Marque'][i]="Inconnu"
```

[7]:
```
df1.Marque.value_counts()
```

[7]:
```
samsung      389
apple        275
xiaomi       138
inconnu      135
huawei       111
oneplus       43
nokia         41
sony          38
```

```
google          26
honor           21
asus            17
oppo            16
realme          16
lg              13
blackberry      10
nubia            9
alcatel          3
oukitel          2
cubot            2
Name: Marque, dtype: int64
```

Dans la cellule suivante nous avons nettoyé les données provenant de la variable écran. Nous avons enlevé la chaine de caractère "['Écran" au début et nous avons enlevé les 3 dernières chaine de caractère

[8]:
```
Nettoyage.nettoyage_ecran(df1)
```

[9]:
```
del df1['Unnamed: 0']
```

[10]:
```
#str
df1['écran']=df1['écran'].astype(str)
df1['reseau']=df1['reseau'].astype(str)
df1['Marque']=df1['Marque'].astype(str)
df1['exploit']=df1['exploit'].astype(str)
df1['prix']=df1['prix'].astype(str)
```

Pour le nettoyage de du variable réseau nous avons transformé les modalités: - 5G si le téléphone est doté d'un réseau 5G - 4G si le téléphone est doté d'un réseau 4G - 3G si le téléphone est doté d'un réseau 3G

[11]:
```
Nettoyage.nettoyage_reseau(df1)
```

En ce qui concerne le nettoyage des données provenant de la variable prix nous avons remplacé les virgules par des points pour pouvoir ensuite les transformer en type numérique, ensuite nous avons enlevé le signe euro et le "nan" présent dans cette colonne.

[12]:
```
Nettoyage.nettoyage_prix(df1)
```

En ce qui concerne le nettoyage des données provenant de la variable exploit nous avons effacé les chaines de caractère "['" et "']" présent.

[13]:
```
Nettoyage.nettoyage_exploit(df1)
```

À partir de le variable exploit nous avons créé deux variables la variable RAM et la variable mémoire. Dans le cas où l'information sur le RAM et la mémoire serait inexistante dans le variable

exploit nous avons défini la modalité “inconnu”. Et nous avons pour la fin supprimée la colonne associée à la variable exploit.

[14]:
```
Nettoyage.col_Ram(df1)
```

[15]:
```
Nettoyage.cor_exploit(df1)
```

[16]:
```
Nettoyage.col_memoire(df1)
```

[17]:
```
df1.pop('exploit')
df1[1:10]
```

[17]:

	Marque	reseau	prix	écran	Ram	Mémoire
1	oneplus	4G	172.58	6.52	RAM 4 Go	64
2	huawei	4G	196.59	6.4	RAM 6 Go	128
5	huawei	4G	214.50	6.15	RAM 4 Go	128
6	samsung	4G	374	6.4	RAM 6 Go	128
7	samsung	4G	270	5.8	RAM 4 Go	64
8	huawei	4G	189	5.84	RAM 4 Go	64
9	samsung	4G	289.99	6.5	RAM 4 Go	128
10	inconnu	5G	320.88	6.6	RAM 4 Go	128
11	google	5G	825	6	RAM 8 Go	128

[18]:
```
print(df1['Ram'].value_counts())
```

```
Inconnu        330
 RAM 4 Go      181
 RAM 6 Go      157
 RAM 8 Go      124
 RAM 3 Go       84
 RAM 2 Go       61
 RAM 1 Go       18
 RAM 12 Go       9
 RAM 512 Mo      7
 Inconnu         3
 RAM 16 Go       3
Name: Ram, dtype: int64
```

[19]:
```
Nettoyage.nettoyage_prix(df1)
```

[20]:
```
print(df1.Marque.value_counts())
```

```
samsung       267
apple         258
xiaomi        114
huawei         93
```

```
inconnu        64
sony           29
oneplus        29
nokia          26
google         21
honor          21
asus           15
oppo           14
lg              9
blackberry      8
realme          8
alcatel         1
Name: Marque, dtype: int64
```

[21]:
```
df1=df1.reset_index()
del df1['index']
```

- Àprès avoir nettoyé toute notre base de données et ajouté deux variables nous nous retrouvons avec une base de données de 977 observations et 6 variables.

[22]:
```
df1[1:10]
```

[22]:

	Marque	reseau	prix	écran	Ram	Mémoire
1	oneplus	4G	172.58	6.52	RAM 4 Go	64
2	huawei	4G	196.59	6.4	RAM 6 Go	128
3	huawei	4G	214.50	6.15	RAM 4 Go	128
4	samsung	4G	374	6.4	RAM 6 Go	128
5	samsung	4G	270	5.8	RAM 4 Go	64
6	huawei	4G	189	5.84	RAM 4 Go	64
7	samsung	4G	289.99	6.5	RAM 4 Go	128
8	inconnu	5G	320.88	6.6	RAM 4 Go	128
9	google	5G	825	6	RAM 8 Go	128

4 PARTIE DESCRIPTIVE

Pour commencer cette partie nous transformons toutes nos variables en type “category” sauf la variable correspondant au prix.

[23]:
```
#str
df1.Ram=df1.Ram.astype("category")
df1.reseau=df1.reseau.astype("category")
df1.Marque=df1.Marque.astype("category")
df1['écran']=df1['écran'].astype("category")
df1['Mémoire']=df1['Mémoire'].astype("category")
df1.prix=pd.to_numeric(df1["prix"])
```

[24]:
```
complet=df1
```

Avec cette commande nous montrons le nombre de modalités par variable, modalité la plus représentée pour chaque variable et leur nombre d'apparition.

[25]:
```
price = complet.set_index('prix') # transformer la variable en
print(price.describe())
```

```
         Marque reseau écran      Ram  Mémoire
count       977    977   977      977      977
unique       16      3    72       11        9
top     samsung     4G   6.1  Inconnu  Inconnu
freq        267    784    83      330      333
```

Ici nous montrons quelques informations sur le variable prix, notamment la moyenne, le minimum et le maximum.

[26]:
```
df1.describe().round(2)
```

[26]:
```
           prix
count   977.00
mean    370.50
std     321.79
min      15.00
25%     159.40
50%     255.00
75%     469.99
max    2099.00
```

[27]:
```
import seaborn as sns
import numpy
import pandas
```

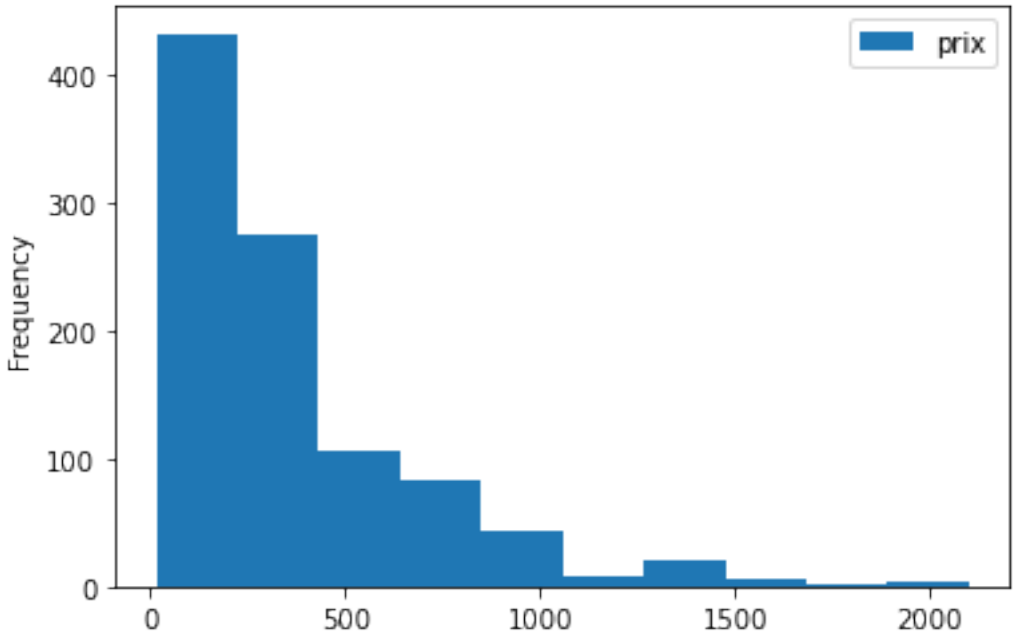
[4]:
```

```

[4]:
```
DeprecationWarning
```

Nous remarquons grace à ces deux graphiques que la majorité des téléphones ont un prix compris entre 15 et 500 euros.

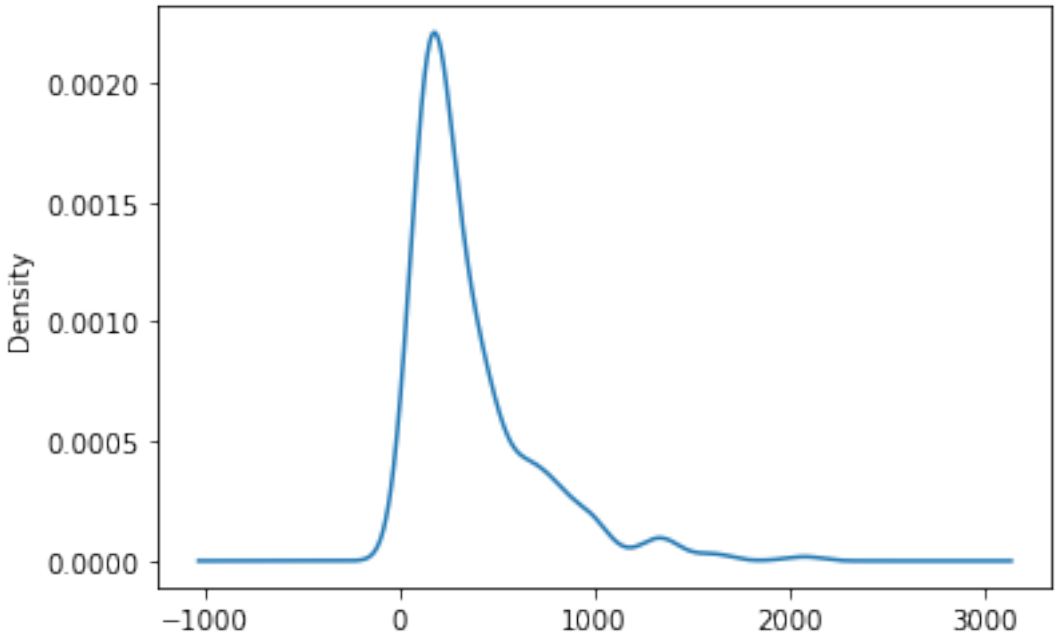
[28]:
```
df1.plot.hist()
```

[28]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20f5a8f91c0>
```



[29]:
```
df1.prix.plot(kind = "kde")
```

[29]: <matplotlib.axes._subplots.AxesSubplot at 0x20f5b0af760>

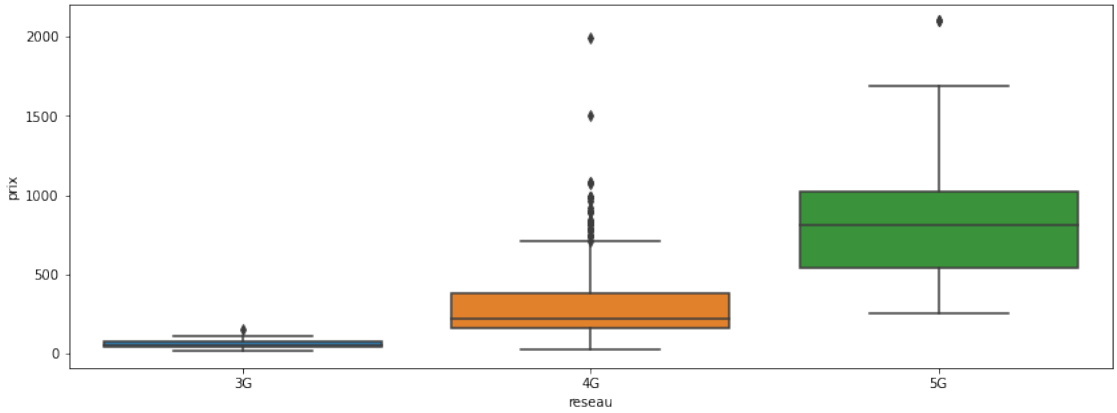


Grâce à ce boxplot nous constatons que le prix du téléphone dépend assez grandement de sa

technologie réseau.
Plus la technologie réseau est élevée plus le prix en moyenne est élevé.

[30]:
```
plt.figure(figsize=(14, 5))
sns.boxplot(x="reseau", y="prix", data=df1)
```

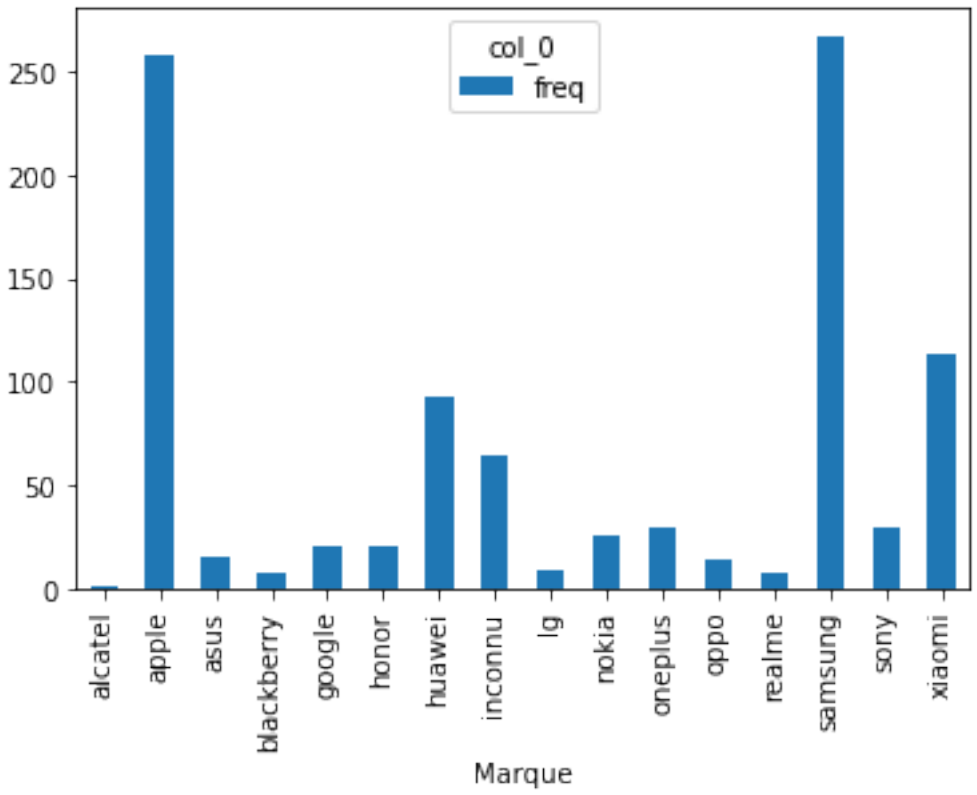
[30]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20f5b10fd90>
```



Grâce à ce diagramme en bâton nous observons que les téléphones de marque Samsung et apple sont les plus représentés ensuite viennent les marques telles que xiaomi, huawei et la variable “inconnu”.

[31]:
```
t = pandas.crosstab(df1.Marque, "freq")
t.plot.bar()
```

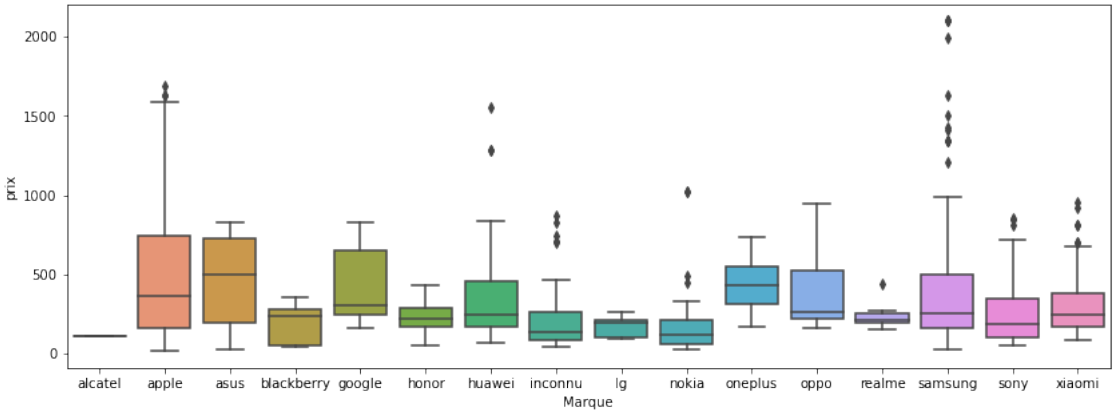
[31]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20f5b18c7c0>
```



Dans ce graphique nous représentons des boxplot qui nous montrent les prix par marques.

[32]:
```
plt.figure(figsize=(14, 5))
sns.boxplot(x="Marque", y="prix", data=df1)
```

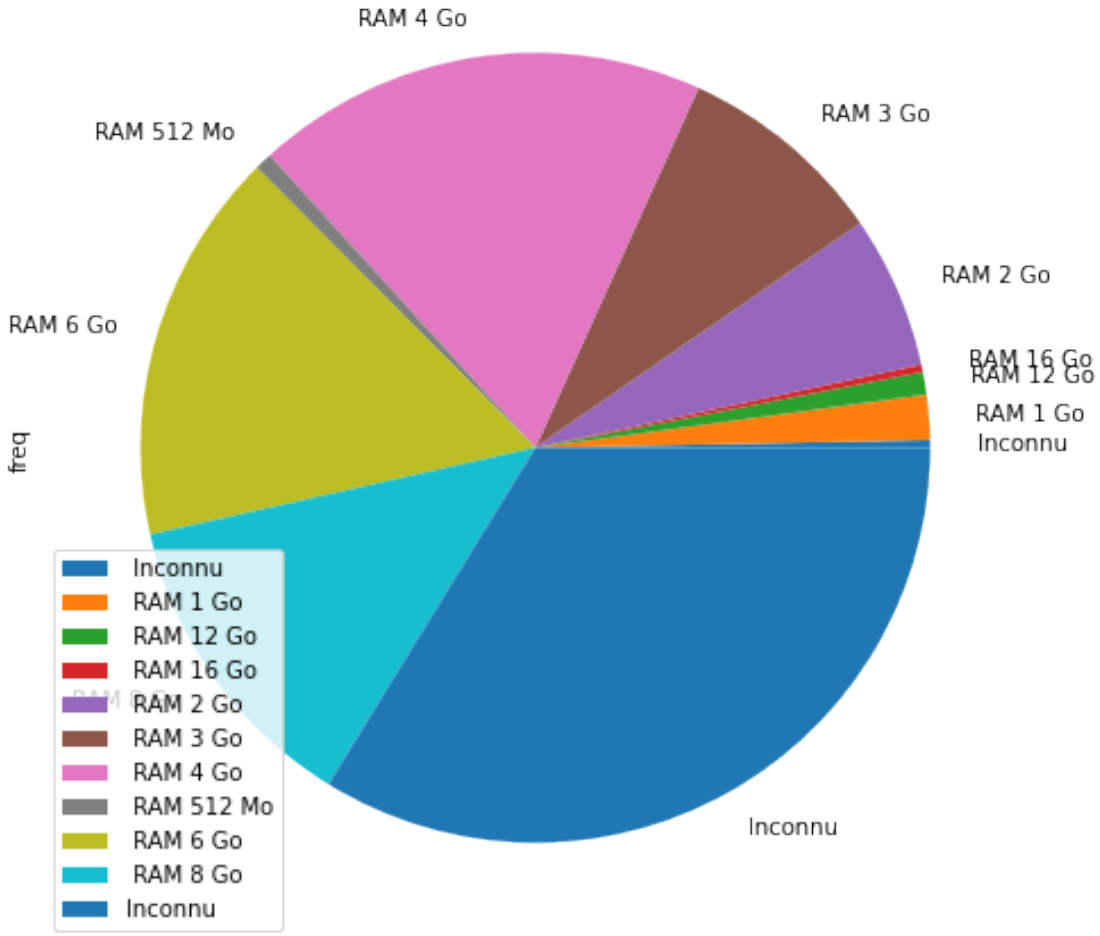
[32]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20f5b17cee0>
```



Ici nous représentons un camembert qui repreneuse l'effectif des téléphones pour chaque RAM disponiple.

[33]:
```
t = pandas.crosstab(df1.Ram, "freq")
t.plot.pie(subplots=True, figsize = (8,8))
```

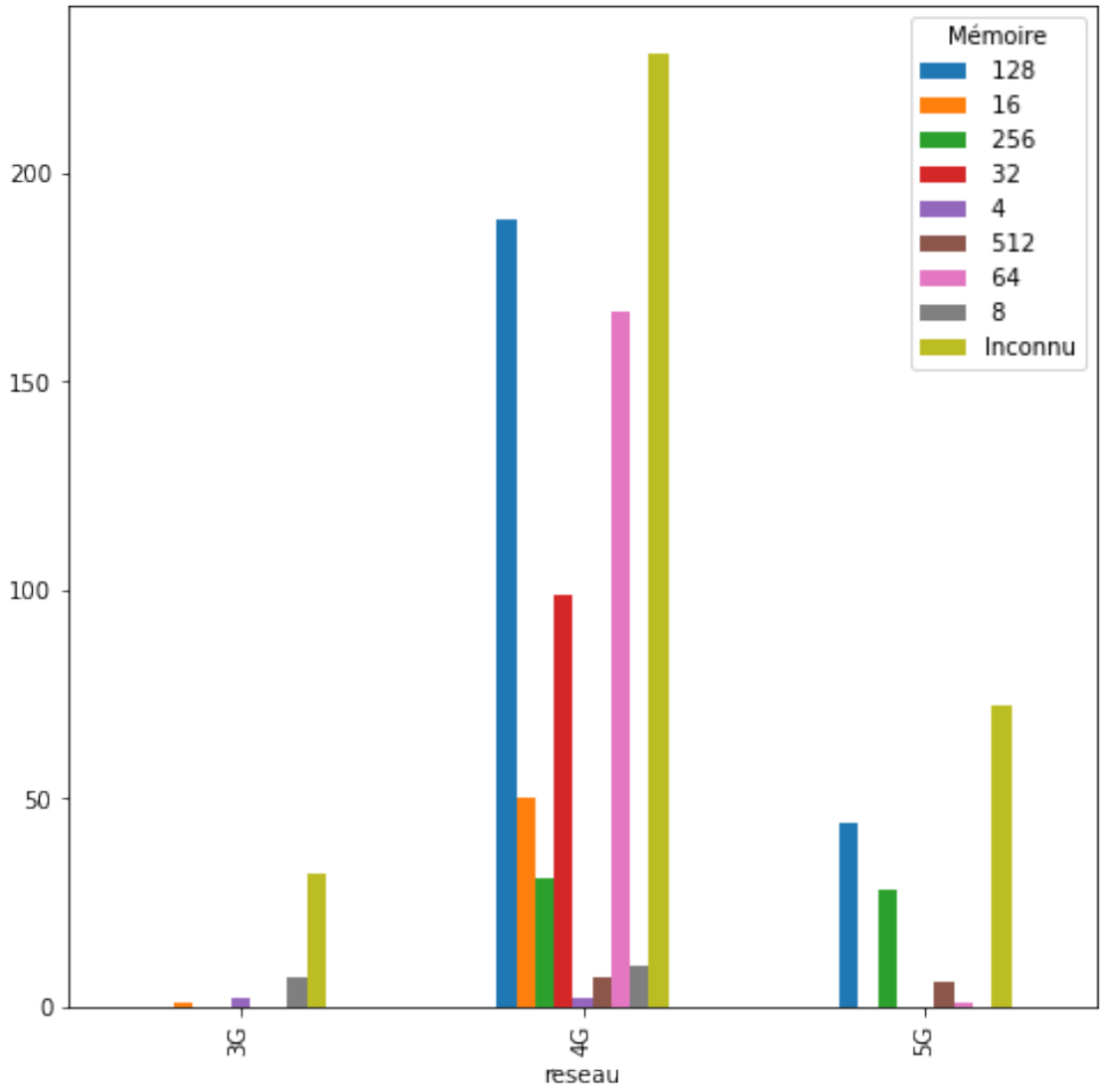
[33]:
```
array([<matplotlib.axes._subplots.AxesSubplot object at 0x0000020F5B1B2580>],
      dtype=object)
```



Ce graphique nous montre la quantité de téléphone en fonction de sa technologie réseau et de sa mémoire.

[34]:
```
t = pd.crosstab(df1.reseau, df1.Mémoire)
t.plot.bar( figsize = (8,8))
```

[34]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x20f5b565b50>
```



5 Partie Apprentissage

Pour commencer la partie apprentissage nous commençons par définir les variables explicatives et la variable à expliquer.
En ce qui nous concerne la variable à expliquer est le prix en fonction des variables marques, réseau, RAM, écran, mémoire.

[35]:
```
X, y = df1[["Marque","reseau","Ram","écran","Mémoire"]], df1["prix"]
```

Nous vérifions que nous avons bien le même nombre de lignes dans les matrices associé aux variables explicatives et à la variable à expliquer.

[36]:
```
X.shape, y.shape
```

[36]:
```
((977, 5), (977,))
```

[37]:
```
from sklearn.model_selection import train_test_split
```

- Séparation de la base de données en données d'entrainement et données d'apprentissage

[38]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y)
```

[39]:
```
quali = ["Marque","reseau","Ram","écran","Mémoire"]

qualitrans=Pipeline(steps=[('imputer',OneHotEncoder(handle_unknown="ignore"))])

transformer = ColumnTransformer(transformers=[('cat',qualitrans,quali)])
```

- Grâce à cette fonction nous allons pouvoir modéliser plusieurs modèles avec différents paramètres.
- Les modèles qui sont modélisés sont les suivants:*LinearRegression, Lasso, Ridge, ElasticNet, KNeighborsRegressor, SVR, RandomForestRegressor, MLPRegressor.*

[40]:
```
modeles = []
modeles.append(LinearRegression())

for val_alpha in (1e-3, 1e-2, 1e-1, 1):
    modeles.append(Lasso(alpha=val_alpha))

for val_alpha in (1e-3, 1e-2, 1e-1, 1):
    modeles.append(Ridge(alpha=val_alpha))

for val_alpha in (1e-3, 1e-2, 1e-1, 1):
    for val_l1 in (0.25, 0.5, 0.75):
        modeles.append(ElasticNet(alpha=val_alpha, l1_ratio=val_l1))

for nb_voisins in range(3, 10):
    modeles.append(KNeighborsRegressor(n_neighbors=nb_voisins))



for val_epsilon in (10 ** n for n in range(-3, 1)):
    for val_C in (10 ** n for n in range(-3, 4)):
        modeles.append(SVR(epsilon=val_epsilon, C=val_C))


for nb_estimateurs in (50, 100, 150, 200):
    modeles.append(RandomForestRegressor(n_estimators=nb_estimateurs))

for nb_neurones in  ((100,), (50, 50), (25, 50, 25)):
    modeles.append(MLPRegressor(hidden_layer_sizes=nb_neurones))
```

```
i=0
Classificateur=[]
for i in range(len(modeles)):
    liste= Pipeline(steps=[("processor",transformer),
                           ("classi",modeles[i])])
    Classificateur.append(liste)
    i=i+1

i=0
resultats=[]
for i in range(len(Classificateur)):
    cr=cross_val_score(Classificateur[i],X_train,y_train,cv=10)
    resultats.append(cr)
    i=i+1
DeprecationWarning
```

```
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1858602.4993796684, tolerance: 6767.307471662021
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1948954.9703600053, tolerance: 6772.872292048161
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1509948.71864085, tolerance: 7108.9050141169355
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2043054.927793432, tolerance: 7116.687170477754
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1585327.2739005527, tolerance: 7392.553895035389
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2194330.353223826, tolerance: 7276.594141967224
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
```

```
  warnings.warn(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

- Performance des modèles sur les données d'entrainement suite à une cross-validation.

[41]:
```
Dico={}
i=0
for i in range(len(resultats)):
    Dico[modeles[i]]=[resultats[i].mean(),resultats[i].std()]
    i=i+1
```

[42]:
```
for key, value in (sorted(Dico.items(), key=lambda x: x[1],reverse=True)) :
    print(f'{repr(key):50} : {value}')
```

```
RandomForestRegressor(n_estimators=200)            : [0.8500670896710559,
0.0452415438631804]
RandomForestRegressor(n_estimators=50)             : [0.8459837252851836,
0.0425471041385668]
RandomForestRegressor(n_estimators=150)            : [0.8447658919372051,
0.04981255906222098]
RandomForestRegressor()                            : [0.8445608626103738,
0.04700879066574336]
SVR(C=1000, epsilon=1)                             : [0.8098230198250691,
0.07335370999561991]
```

```
SVR(C=1000)                                          : [0.8095997135884161,
0.07344033867810665]
SVR(C=1000, epsilon=0.01)                            : [0.8095796211923375,
0.07345039282427256]
SVR(C=1000, epsilon=0.001)                           : [0.8095776309920767,
0.07345137517969176]
MLPRegressor(hidden_layer_sizes=(25, 50, 25))        : [0.7880671546626665,
0.06643525081167745]
KNeighborsRegressor(n_neighbors=4)                   : [0.7842880696893514,
0.05980865008094562]
ElasticNet(alpha=0.001, l1_ratio=0.75)               : [0.7702063318411752,
0.0636469239994682]
ElasticNet(alpha=0.001)                              : [0.7698429991044152,
0.06275959493907071]
Lasso(alpha=0.1)                                     : [0.7698412062902722,
0.06869858426347178]
Ridge(alpha=0.1)                                     : [0.7691111185490316,
0.06462507252305312]
KNeighborsRegressor(n_neighbors=7)                   : [0.7683026191659096,
0.07402266643600348]
ElasticNet(alpha=0.001, l1_ratio=0.25)               : [0.7677688774982762,
0.062388596575485004]
Lasso(alpha=0.01)                                    : [0.767522895219372,
0.06756790678095531]
KNeighborsRegressor(n_neighbors=6)                   : [0.7671818537071479,
0.0690334039167989]
Ridge(alpha=0.01)                                    : [0.7661004514614943,
0.06642412961329465]
LinearRegression()                                   : [0.7657426570448935,
0.06694350285629931]
Ridge(alpha=0.001)                                   : [0.7655698753785873,
0.06676112138715604]
Lasso(alpha=0.001)                                   : [0.7651284465200551,
0.06603228815959183]
KNeighborsRegressor()                                : [0.7636269476248141,
0.06695161885488107]
KNeighborsRegressor(n_neighbors=8)                   : [0.763155832677435,
0.07793604646104177]
Ridge(alpha=1)                                       : [0.7588592883490233,
0.06169738933320693]
KNeighborsRegressor(n_neighbors=9)                   : [0.758750668803495,
0.08357634932684442]
ElasticNet(alpha=0.01, l1_ratio=0.75)                : [0.7478867150412051,
0.060729492495686335]
KNeighborsRegressor(n_neighbors=3)                   : [0.7474435584855071,
0.06864788697767678]
MLPRegressor(hidden_layer_sizes=(50, 50))            : [0.73706482468824,
0.05882102735694653]
```

```
Lasso(alpha=1)                                  : [0.7340164644619082,
0.060741042801707626]
ElasticNet(alpha=0.01)                          : [0.7287063858786114,
0.059124176651952895]
ElasticNet(alpha=0.01, l1_ratio=0.25)           : [0.7159576895084248,
0.05826421897785568]
SVR(C=100, epsilon=1)                           : [0.695473566875257,
0.10161151403774481]
SVR(C=100)                                      : [0.6953442974371866,
0.1017880481651834]
SVR(C=100, epsilon=0.01)                        : [0.6953336521047896,
0.10181214582099625]
SVR(C=100, epsilon=0.001)                       : [0.6953324961986436,
0.10181461403612185]
ElasticNet(alpha=0.1, l1_ratio=0.75)            : [0.6630105553011121,
0.05942113080609634]
ElasticNet(alpha=0.1)                           : [0.6161218792365035,
0.061516868709634125]
ElasticNet(alpha=0.1, l1_ratio=0.25)            : [0.580505321422616,
0.06161192520668646]
ElasticNet(alpha=1, l1_ratio=0.75)              : [0.42373936097910914,
0.05597026354933814]
ElasticNet(alpha=1)                             : [0.32128102448027007,
0.04685775993289569]
SVR(C=10, epsilon=1)                            : [0.2901406494417049,
0.0822939287545958]
SVR(C=10)                                       : [0.290014799651673,
0.08233369251084098]
SVR(C=10, epsilon=0.01)                         : [0.29001463506434344,
0.08233397622092271]
SVR(C=10, epsilon=0.001)                        : [0.2900145370556694,
0.08233410130008821]
ElasticNet(alpha=1, l1_ratio=0.25)              : [0.26037999933823075,
0.04012567108874257]
MLPRegressor()                                  : [0.1908685637289355,
0.059739153956505385]
SVR(C=1, epsilon=0.001)                         : [-0.06895972375212187,
0.06804987049732593]
SVR(C=1, epsilon=0.01)                          : [-0.0689672466627875,
0.06804994946110285]
SVR(C=1)                                        : [-0.06905284784031904,
0.0680620869930629]
SVR(C=1, epsilon=1)                             : [-0.06941555101937173,
0.06819889112851797]
SVR(C=0.1)                                      : [-0.13321677654780975,
0.07529650064894326]
SVR(C=0.1, epsilon=0.01)                        : [-0.13323051784243803,
0.07535330871458967]
```

```
SVR(C=0.1, epsilon=0.001)                          : [-0.13323146768701105,
0.07535751627066044]
SVR(C=0.1, epsilon=1)                              : [-0.13348751915409646,
0.0751763786681212]
SVR(C=0.01, epsilon=1)                             : [-0.13965112664052926,
0.07587679947450317]
SVR(C=0.01, epsilon=0.001)                         : [-0.1396785483240938,
0.07659882899870603]
SVR(C=0.01, epsilon=0.01)                          : [-0.13968015837997966,
0.07659730792898009]
SVR(C=0.01)                                        : [-0.1396848956008851,
0.07655911278723776]
SVR(C=0.001, epsilon=1)                            : [-0.14024969239578106,
0.07598076725192401]
SVR(C=0.001, epsilon=0.001)                        : [-0.14027052355811656,
0.07668965903353325]
SVR(C=0.001, epsilon=0.01)                         : [-0.14027213390387971,
0.07668814061742786]
SVR(C=0.001)                                       : [-0.1402817053415087,
0.07663298789779514]
```

[43]:
```
modfit=[]
for i in range(len(Classificateur)):
    L = Classificateur[i].fit(X_train, y_train)
    modfit.append(L)
```

```
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 2163020.204727728, tolerance: 7907.571891470534
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 84049.36738547683, tolerance: 7907.571891470534
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 210658.7577155307, tolerance: 7907.571891470534
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\linear_model\_coordinate_descent.py:512: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 4539450.678364381, tolerance: 7907.571891470534
  model = cd_fast.sparse_enet_coordinate_descent(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
```

```
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
C:\Users\caleb\Anaconda3\lib\site-
packages\sklearn\neural_network\_multilayer_perceptron.py:582:
ConvergenceWarning: Stochastic Optimizer: Maximum iterations (200) reached and
the optimization hasn't converged yet.
  warnings.warn(
```

- Performance des modèles sur les données de test suite à une cross-validation.

[44]:
```
Dicoscoretest={}
i=0
for i in range(len(modeles)):
    Dicoscoretest[modeles[i]]=[modfit[i].score(X_test, y_test)]
    i=i+1
```

[45]:
```
for key, value in (sorted(Dicoscoretest.items(), key=lambda x:␣
 ↪x[1],reverse=True)) :
    print(f'{repr(key):50} : {value}')
```

```
RandomForestRegressor(n_estimators=200)            : [0.8241934501947241]
RandomForestRegressor(n_estimators=50)             : [0.8230916624617003]
RandomForestRegressor(n_estimators=150)            : [0.8226891585261457]
RandomForestRegressor()                            : [0.8225567096535018]
MLPRegressor(hidden_layer_sizes=(25, 50, 25))      : [0.8199349473220139]
SVR(C=1000, epsilon=1)                             : [0.8061313813667815]
SVR(C=1000)                                        : [0.8060535127113734]
SVR(C=1000, epsilon=0.01)                          : [0.8060404785478497]
SVR(C=1000, epsilon=0.001)                         : [0.8060391439345816]
KNeighborsRegressor(n_neighbors=9)                 : [0.7997880650492766]
KNeighborsRegressor(n_neighbors=8)                 : [0.7977441366148311]
KNeighborsRegressor(n_neighbors=6)                 : [0.7890954586828117]
KNeighborsRegressor(n_neighbors=7)                 : [0.7835892341103244]
KNeighborsRegressor()                              : [0.7824742624418706]
SVR(C=100, epsilon=1)                              : [0.7798069749108093]
SVR(C=100)                                         : [0.7795511266850292]
SVR(C=100, epsilon=0.01)                           : [0.7795199543309009]
SVR(C=100, epsilon=0.001)                          : [0.7795167615145757]
MLPRegressor(hidden_layer_sizes=(50, 50))          : [0.7758662735787456]
KNeighborsRegressor(n_neighbors=4)                 : [0.7553108015022969]
Lasso(alpha=0.01)                                  : [0.7468772179063484]
LinearRegression()                                 : [0.7468680817479408]
```

```
ElasticNet(alpha=0.001, l1_ratio=0.75)            : [0.7464079915366889]
Ridge(alpha=0.01)                                 : [0.7463450932926271]
Ridge(alpha=0.1)                                  : [0.7462453911340905]
Ridge(alpha=0.001)                                : [0.7460690213617304]
Lasso(alpha=0.001)                                : [0.7459852488514643]
ElasticNet(alpha=0.001)                           : [0.745786676012393]
ElasticNet(alpha=0.001, l1_ratio=0.25)            : [0.7450137827610144]
Ridge(alpha=1)                                    : [0.7424028894442701]
Lasso(alpha=0.1)                                  : [0.7417129320910988]
ElasticNet(alpha=0.01, l1_ratio=0.75)             : [0.7374514304010316]
KNeighborsRegressor(n_neighbors=3)                : [0.7343349939837853]
ElasticNet(alpha=0.01)                            : [0.7280178121118541]
ElasticNet(alpha=0.01, l1_ratio=0.25)             : [0.7201684283338825]
Lasso(alpha=1)                                    : [0.7081107205870469]
ElasticNet(alpha=0.1, l1_ratio=0.75)              : [0.6797373086550054]
ElasticNet(alpha=0.1)                             : [0.640329807886225]
ElasticNet(alpha=0.1, l1_ratio=0.25)              : [0.6084014280888481]
ElasticNet(alpha=1, l1_ratio=0.75)                : [0.45390505225860756]
SVR(C=10, epsilon=0.001)                          : [0.40044649646477726]
SVR(C=10, epsilon=0.01)                           : [0.4004394419302848]
SVR(C=10)                                         : [0.40036797658152656]
SVR(C=10, epsilon=1)                              : [0.400053671580084]
ElasticNet(alpha=1)                               : [0.347710388577246]
ElasticNet(alpha=1, l1_ratio=0.25)                : [0.28345319563502747]
MLPRegressor()                                    : [0.26164906361977336]
SVR(C=1)                                          : [-0.009079485324649195]
SVR(C=1, epsilon=0.01)                            : [-0.009087247140311838]
SVR(C=1, epsilon=0.001)                           : [-0.009088023496207409]
SVR(C=1, epsilon=1)                               : [-0.00965578734455641]
SVR(C=0.1, epsilon=1)                             : [-0.08578905232147926]
SVR(C=0.1, epsilon=0.001)                         : [-0.08583309877427703]
SVR(C=0.1, epsilon=0.01)                          : [-0.08583309877427703]
SVR(C=0.1)                                        : [-0.08583309877427703]
SVR(C=0.01, epsilon=1)                            : [-0.0927329407541233]
SVR(C=0.01, epsilon=0.001)                        : [-0.0930006957785301]
SVR(C=0.01, epsilon=0.01)                         : [-0.0930006957785301]
SVR(C=0.01)                                       : [-0.0930006957785301]
SVR(C=0.001, epsilon=1)                           : [-0.09323254078863497]
SVR(C=0.001, epsilon=0.001)                       : [-0.09372002390543743]
SVR(C=0.001, epsilon=0.01)                        : [-0.09372002390543743]
SVR(C=0.001)                                      : [-0.09372002390543743]
```

- Àprès avoir testé et ordonné tous les modèles sur les données d'entrainement et les données de test du meilleur au plus mauvais, nous constatons que les modèles sont les Random Forest. Et plus précisement celle avec un n_estimators de défaut c'est à 200(dans mon cas).

[46]:
```
import machine
```

- Nous utilisons le modfit[57] car elle correspond au modèle random forest avec un n_estimators par défaut.

[47]:
```
machine.prediction(df1,modfit[57],1)
```

[47]:
```
     predicted    true  difference
1  188.416029  172.58   15.836029
```

La fonction prediction provenant de machine.py fait une estimation du prix par rapports aux différentes variables explicatives.Elle estime le meilleur prix possible en fonction des données.

Ici les données proviennent de la base de données déjà existante que nous avons mais nous aurions bien évidemment pu modifier la fonction pour que nous-mêmes entrons les valeurs des variables explicatives et non pas qu'elles proviennent de la base de données déjà existante ou fictive.Et faire bien plus encore

Nous avons laissé certaine fonction comme celle du scrraping et celle sur les modèles(cross-validation) pour montrer la complexité.

6 fin