Evolution du prix des matières énergétiques sur le marché international: cas du prix du pétrole

RAZAFIMANANTENA Andy (DS2E) - DIN Kapo (SE) - DJAFON K. Joseph (SE)

2023-11-09

Import all packages we will need

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error, r2_score
from sklearn import metrics
from sklearn.linear_model import Ridge, RidgeCV, Lasso, LassoCV
```

Import our databse

```
path= r'C:\Users\AN\Documents\Master_SE\Cours\Master 2\Machine learning\data1.csv'
data = pd.read_csv(path, delimiter=';')
data
```

```
##            Date  crude_oil    gas  cobalt    uranium     charbon  bau_alu
## 0    01/04/2010  84.497727  2.900  39.125  41.325000  104.962500    152.1
## 1    01/05/2010  73.842381  2.890  38.445  41.300000  107.935150    155.5
## 2    01/06/2010  75.349091  2.785  38.100  40.777778  104.483766    143.4
## 3    01/07/2010  76.177273  2.782  41.650  41.944444  102.861526    137.9
## 4    01/08/2010  76.618182  2.783  38.780  46.062500   96.846939    142.9
## ..          ...        ...    ...     ...        ...         ...      ...
## 157  01/05/2023  71.673478  3.666  33.074  43.462666  179.262321    208.9
## 158  01/06/2023  70.306818  3.684  33.049  45.700483  138.631656    204.2
## 159  01/07/2023  75.766667  3.712  33.055  45.247534  141.029592    204.1
## 160  01/08/2023  81.372609  3.954  33.071  46.377181  152.792532    200.2
## 161  01/09/2023  89.240952  3.958  33.059  53.202742  168.479082    196.2
##
## [162 rows x 7 columns]
```

Checking dta

check 1

```
data.info()
```

```
## <class 'pandas.core.frame.DataFrame'>
## RangeIndex: 162 entries, 0 to 161
## Data columns (total 7 columns):
##  #   Column     Non-Null Count  Dtype
## ---  ------     --------------  -----
##  0   Date       162 non-null    object
##  1   crude_oil  162 non-null    float64
##  2   gas        162 non-null    float64
##  3   cobalt     162 non-null    float64
##  4   uranium    162 non-null    float64
##  5   charbon    162 non-null    float64
##  6   bau_alu    162 non-null    float64
## dtypes: float64(6), object(1)
## memory usage: 9.0+ KB
```

check 2

```
data.describe()
```

```
##         crude_oil         gas      cobalt     uranium     charbon     bau_alu
## count  162.000000  162.000000  162.000000  162.000000  162.000000  162.000000
## mean    71.474718    3.066938   40.101963   35.899715  121.574343  154.422222
## std     22.341630    0.626032   16.216917   10.628474   87.630467   34.555159
## min     16.975000    1.872000   21.820000   18.568182   51.382500  115.000000
## 25%     51.642267    2.536250   29.532500   27.391378   74.090631  131.625000
## 50%     71.456136    2.944500   33.052000   35.189016   96.157962  143.500000
## 75%     91.952795    3.609750   50.115000   41.696513  123.773011  162.200000
## max    114.675909    5.032000   93.550000   65.000000  467.783673  269.400000
```

check 3

```
data.isna().sum()
```

```
## Date         0
## crude_oil    0
## gas          0
## cobalt       0
## uranium      0
## charbon      0
## bau_alu      0
## dtype: int64
```

check 4

```
data.duplicated().sum()
```

```
## 0
```

Chek 5: Check for many outliers or not

```
# Var list
var = ['crude_oil','gas','cobalt','uranium','charbon','bau_alu']

# A function to detect outliers
def detect_outliers(columns):
    outlier_indices = []

    for column in columns:
        # 1st quartile
        Q1 = np.percentile(data[column], 25)
        # 3st quartile
        Q3 = np.percentile(data[column], 75)
        # IQR
        IQR = Q3 - Q1
        # Outlier Step
        outlier_step = IQR * 1.5
        # detect outlier and their indeces
        outlier_list_col = data[(data[column] < Q1 - outlier_step)
                                | (data[column] > Q3 + outlier_step)].index
        # store indeces
        outlier_indices.extend(outlier_list_col)
        return outlier_indices

len(detect_outliers(var))
```

```
## 0
```

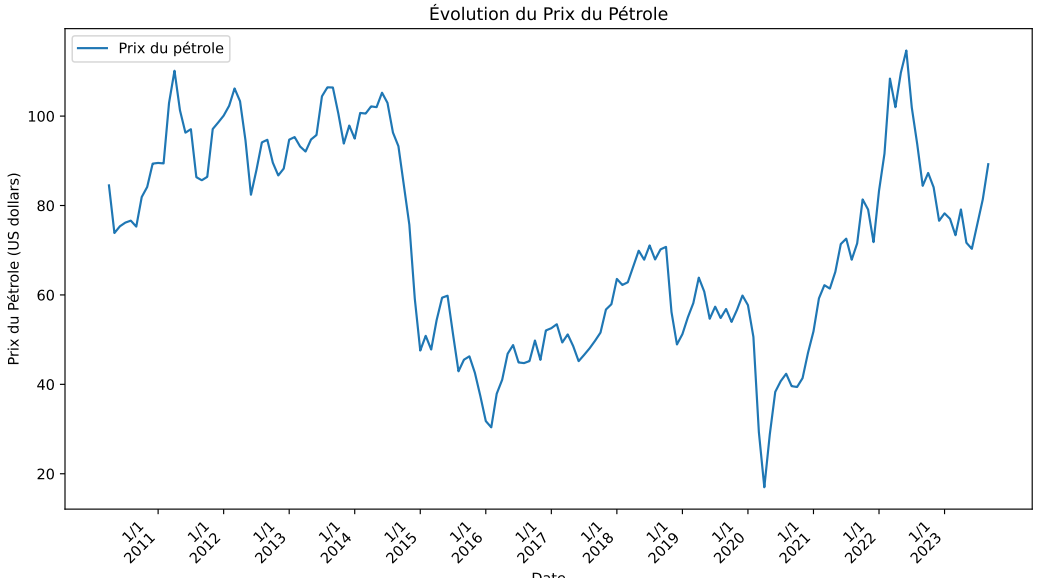
Plotting oil crude price

```
 # Put data column in datatime format
data['Date'] = pd.to_datetime(data['Date'])

# List for X labels
xticks_labels = [f"{x.month}/{x.day}\n{x.year}"
if i % 12 == 9 else '' for i, x in enumerate(data['Date'])]

# Not empty
filtered_xticks_labels = list(filter(lambda x: x != '', xticks_labels))

# Personalize the place of the x labels
custom_xticks_locations = [i for i, x in enumerate(data['Date']) if i % 12 == 9]

# PLot orude oil price
plt.figure(figsize=(12, 6))
plt.plot(data['crude_oil'], label='Prix du pétrole')
plt.xlabel('Date')
plt.ylabel('Prix du Pétrole (US dollars)')
```

```
plt.title('Évolution du Prix du Pétrole')

# Put x labels
plt.xticks(custom_xticks_locations, filtered_xticks_labels, rotation=45, ha='right')
```

```
## ([<matplotlib.axis.XTick object at 0x000000000B00D3A0>, <matplotlib.axis.XTick object at 0x000000000
```

```
plt.legend()
plt.show()
```



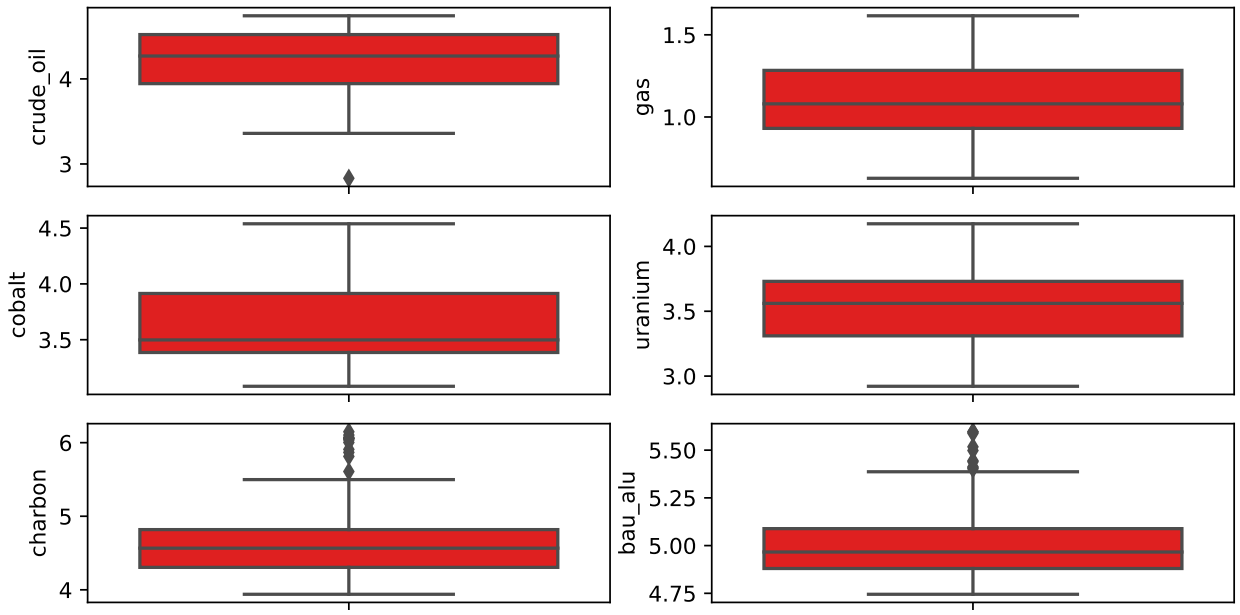
Transform price in log

```
# Selct our variables
colonnes_numeriques = ['crude_oil', 'gas', 'cobalt', 'uranium', 'charbon', 'bau_alu']

# Apply the log
data[colonnes_numeriques] = np.log(data[colonnes_numeriques])
```

Descriptive statistics for the varaibles

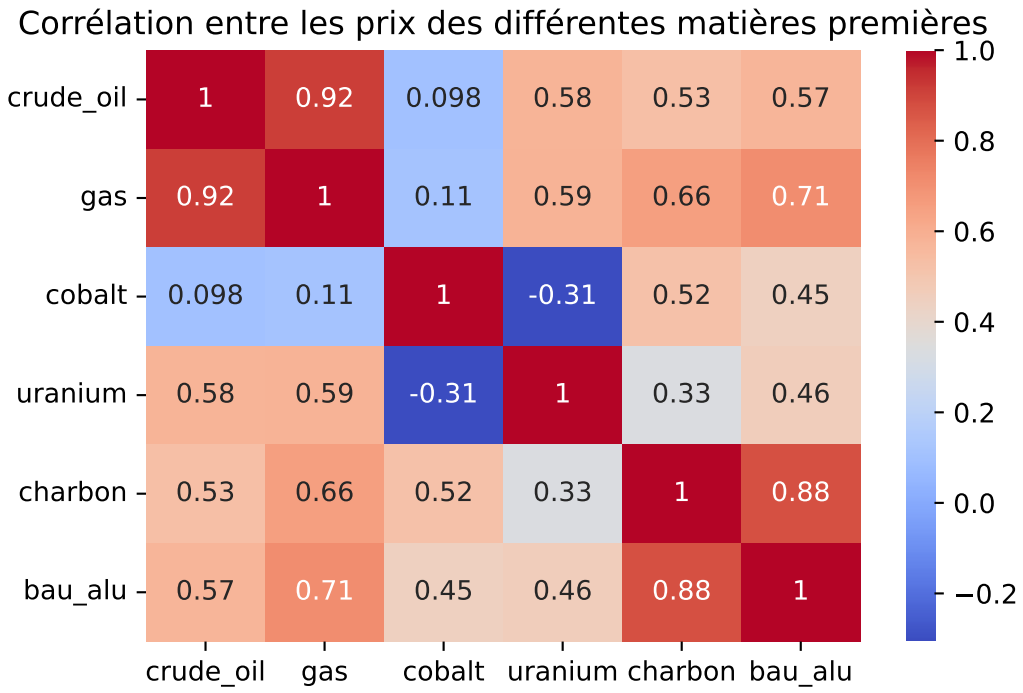
```
# Var list
var = ['crude_oil','gas','cobalt','uranium','charbon','bau_alu']

#Figure
fig, ax = plt.subplots(3, 2, figsize=(8, 4))
index = 0
ax = ax.flatten()

for col in var:
    sns.boxplot(y=col, data=data, color='r', ax=ax[index])
    index += 1
```

```
plt.tight_layout(pad=0.5, w_pad=1, h_pad=1.0)
plt.show()
```



Correlation matrix

```
## Correlation matrix
plt.figure(figsize=(6, 4))

# Matrice de corrélation entre les prix des différentes matières premières
correlation_matrix = data[['crude_oil', 'gas', 'cobalt', 'uranium', 'charbon',
'bau_alu']].corr()

# Plot de la heatmap avec les nouvelles étiquettes pour les axes x et y
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Corrélation entre les prix des différentes matières premières')

plt.show()
```



Separate variables and data

```
# Dinstinguish X vector and Y vector
X = data.drop(['Date','crude_oil'],axis=1)
Y = data['crude_oil']

# Separate the data in training data and test data

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state=2)
```

Estimate with Ridge regression

#Ridge(Step 1)

```
# Instantiate the model
ridge = Ridge()

# Fit the model on the training data
ridge = ridge.fit(X_train, Y_train)

# Visualize coefficients
print(pd.Series(ridge.coef_, index = X.columns))
```

```
## gas        1.170361
## cobalt     0.101749
## uranium    0.228927
## charbon    0.004743
## bau_alu   -0.058502
```

```
## dtype: float64
```

#Ridge(Step 2)

```
# Check the performance of the model
print('MSE (training): %.2f' % mean_squared_error(Y_train, ridge.predict(X_train)),
      'MSE (test): %.2f' % mean_squared_error(Y_test, ridge.predict(X_test)), sep='\n')
```

```
## MSE (training): 0.02
## MSE (test): 0.02
```

#Ridge(Step 3)

```
#Set manually some values for alpha
alphas = 10**np.linspace(5,-5,100)*0.5

coefs = []
for a in alphas:
    ridge = Ridge(alpha=a, fit_intercept=False)
    ridge.fit(X_train, Y_train)
    coefs.append(ridge.coef_)
```

#Ridge(Step 4)

```
# We use cross-validation to choose the tuning parameter
ridgecv = RidgeCV(alphas = alphas, cv = 10, scoring = 'neg_mean_squared_error')
fit1 = ridgecv.fit(X_train, Y_train)
```

#Ridge(Step 5)

```
# Print the turning parameter
print(ridgecv.alpha_)
```

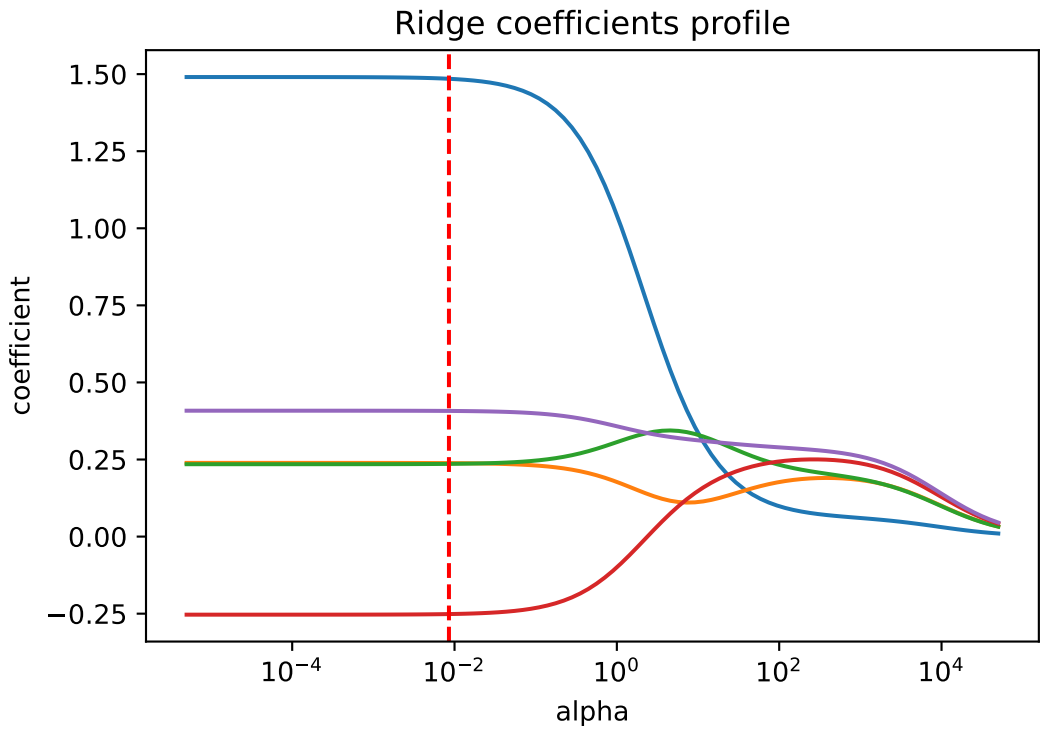
```
## 0.00853676323735346
```

#Ridge (Step 6)

```
#Plot ridge coefficients as a function of the regularization
ax = plt.gca()
ax.plot(alphas, coefs)
ax.set_xscale('log')
plt.axvline(x=ridgecv.alpha_, color='r', linestyle='--')
plt.xlabel('alpha')
plt.ylabel('coefficient')
plt.title('Ridge coefficients profile')
plt.axis('tight')
```

```
## (1.5811388300841894e-06, 158113.88300841895, -0.3404194087757555, 1.5777205180237415)
```

```
plt.show()
```



#Ridge(Step 7)

```
# Print min MSE obtain with the optimal turning parameter
print('MSE (training): %.2f' % mean_squared_error(Y_train, ridgecv.predict(X_train)),
      'MSE (test): %.2f' % mean_squared_error(Y_test, ridgecv.predict(X_test)), sep='\n')
```

```
## MSE (training): 0.02
## MSE (test): 0.01
```

#Ridge(Step8)

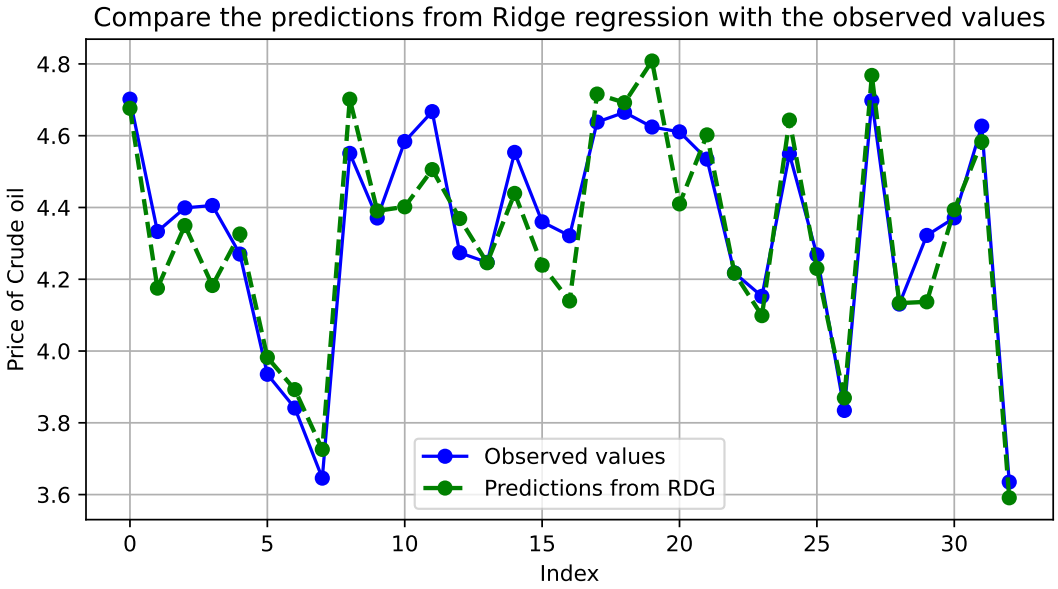
```
# Compute the Prediction and print the coef for Ridge regression
Predict_Ridge = ridgecv.predict(X_test)
print(pd.Series(ridgecv.coef_, index = X.columns))
```

```
## gas        1.738490
## cobalt     0.194334
## uranium    0.198552
## charbon   -0.056719
## bau_alu   -0.413865
## dtype: float64
```

#Ridge(Step 9)

```
# Crate a graph to compare Prediction from Ridge regression and Y test
plt.figure(figsize=(8, 4))
plt.plot(Y_test.values, color='blue', marker='o', label='Observed values')
plt.plot(Predict_Ridge, color='green', marker='o', linestyle='dashed',
linewidth=2, markersize=6, label='Predictions from RDG')
plt.xlabel('Index')
plt.ylabel('Price of Crude oil')
plt.title('Compare the predictions from Ridge regression with the observed values')
```

```
plt.legend()
plt.grid(True)
plt.show()
```



LASSO

#Lasso (Step 1)

```
#Set manually some values for alpha
alphas2 = 10**np.linspace(5,-5,100)*0.5

coefs2 = []
for a in alphas2 :
    lasso = Lasso(alpha=a, fit_intercept=False)
    lasso.fit(X_train, Y_train)
    coefs2.append(lasso.coef_)
```

#Lasso (Step 2)

```
#Estimate
lassocv = LassoCV(alphas = None, cv = 10, max_iter = 100000)
fit2 = lassocv.fit(X_train, Y_train)
```

#Lasso (Step 3)

```
# Print the turning parameter
print(lassocv.alpha_)
```

```
## 9.400163267094516e-05
```

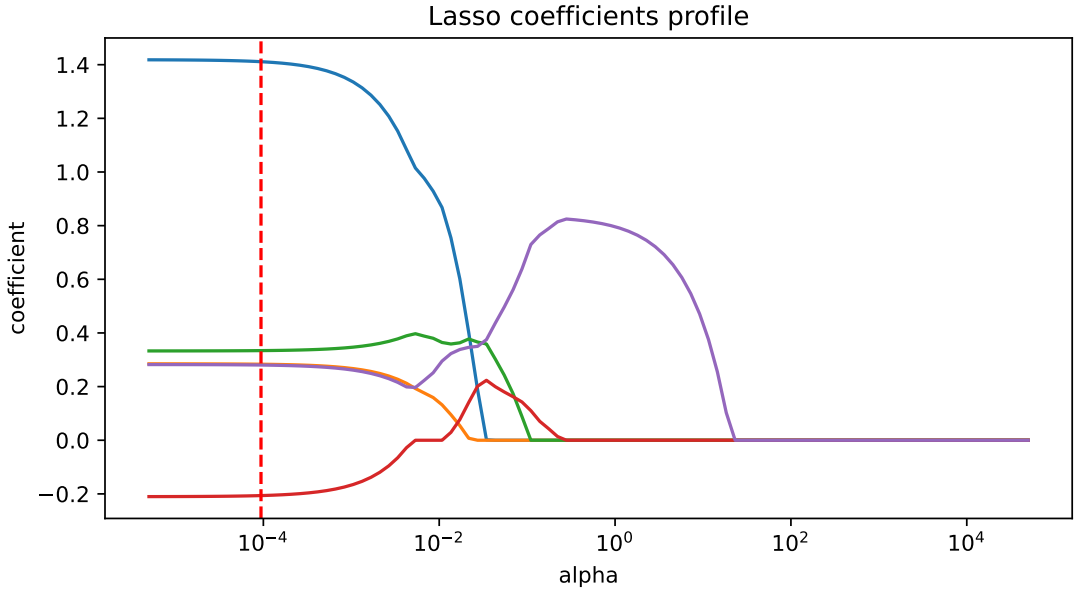
#Lasso (Step 4)

```
#Plot ridge coefficients as a function of the regularization
ax = plt.gca()
ax.plot(alphas2, coefs2)
ax.set_xscale('log')
```

```
plt.axvline(x=lassocv.alpha_, color='r', linestyle='--')
plt.xlabel('alpha')
plt.ylabel('coefficient')
plt.title('Lasso coefficients profile')
plt.axis('tight')
```

```
## (1.5811388300841894e-06, 158113.88300841895, -0.2915373663137017, 1.4996497528022912)
```

```
plt.show()
```



#Lasso (Step 5)

```
# How many coeffiecient are set to zero?
print('Number of features used:', np.sum(lassocv.coef_ != 0))
```

```
## Number of features used: 5
```

#Lasso (Step 6)

```
# Compute the Prediction and print the coef LASSO
Predict_Lasso = lassocv.predict(X_test)
print(pd.Series(lassocv.coef_, index = X.columns))
```

```
## gas        1.737734
## cobalt     0.190009
## uranium    0.193627
## charbon   -0.056576
## bau_alu   -0.404833
## dtype: float64
```

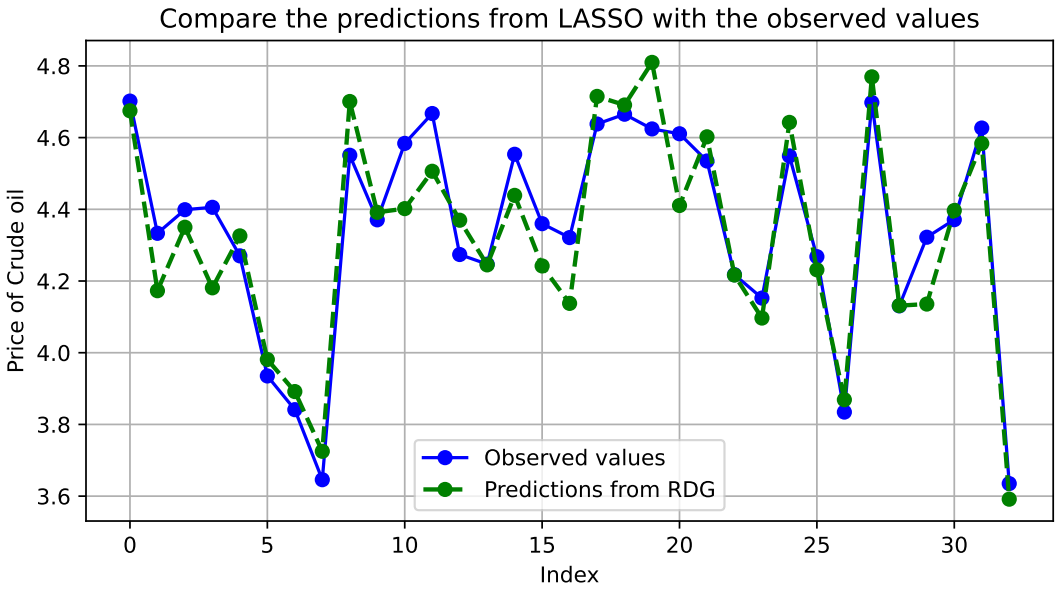
#Lasso (Step 7)

```
#Computethe Prediction and print the coef for Ridge regression
print('MSE (training): %.2f' % mean_squared_error(Y_train, lassocv.predict(X_train)),
      'MSE (test): %.2f' % mean_squared_error(Y_test, lassocv.predict(X_test)), sep='\n')
```

```
## MSE (training): 0.02
## MSE (test): 0.01
```

#Lasso (Step 8)

```
# Crate a graph to compare Prediction from Ridge regression and Y test
plt.figure(figsize=(8, 4))
plt.plot(Y_test.values, color='blue', marker='o', label='Observed values')
plt.plot(Predict_Lasso, color='green', marker='o', linestyle='dashed',
linewidth=2, markersize=6, label='Predictions from RDG')
plt.xlabel('Index')
plt.ylabel('Price of Crude oil')
plt.title('Compare the predictions from LASSO with the observed values')
plt.legend()
plt.grid(True)
plt.show()
```



Random Forest Regressor

#RF Step 1)

```
# Find the optimal tree
n_trees_list = [10, 50, 100, 150, 200]

# Initialiser une liste pour stocker les MSE correspondantes
mse_list = []

# Boucle sur les différents nombres d'arbres
for n_trees in n_trees_list:
    # Initialiser le modèle Random Forest avec le nombre d'arbres actuel
    model = RandomForestRegressor(n_estimators=n_trees, random_state = 200)

    # Entraîner le modèle sur les données d'entraînement
    model.fit(X_train, Y_train)

    # Faire des prédictions sur les données d'entraînement
    y_pred = model.predict(X_test)
```

```
    # Calculer la MSE et l'ajouter à la liste
    mse = mean_squared_error(Y_test, y_pred)
    mse_list.append(mse)

# Trouver le nombre d'arbres qui donne le minimum MSE
```

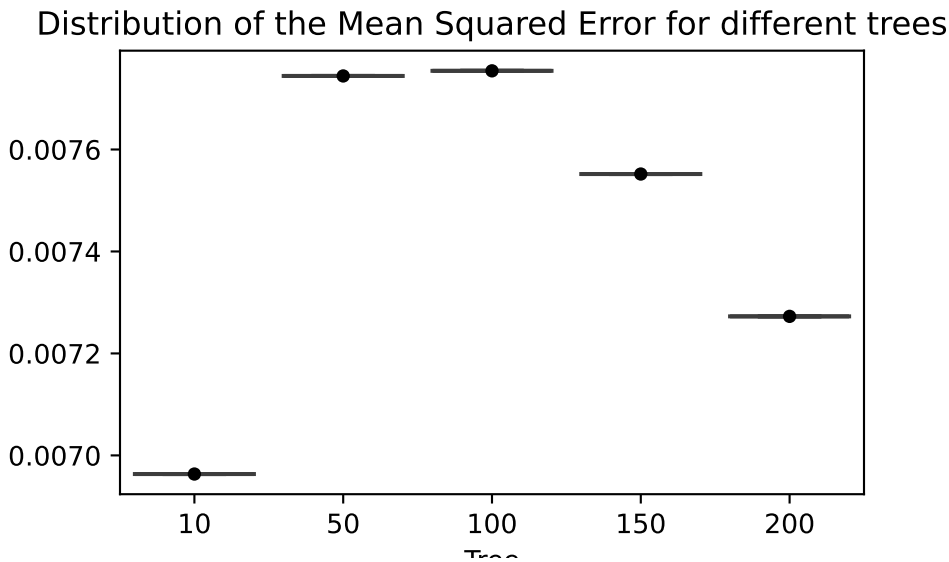
```
## RandomForestRegressor(n_estimators=10, random_state=200)
## RandomForestRegressor(n_estimators=50, random_state=200)
## RandomForestRegressor(random_state=200)
## RandomForestRegressor(n_estimators=150, random_state=200)
## RandomForestRegressor(n_estimators=200, random_state=200)
```

```
best_n_trees = n_trees_list[np.argmin(mse_list)]

# Afficher le résultat
print(f"Le nombre optimal d'arbres est : {best_n_trees}")
```

```
## Le nombre optimal d'arbres est : 10
```

#RF (Step 2)

```
# Box plot
table = pd.DataFrame({'Tree': n_trees_list, 'MSE': mse_list})
plt.figure(figsize=(5, 3))
sns.boxplot(x='Tree', y='MSE', data=table, showfliers=False)
sns.stripplot(x='Tree', y='MSE', data=table, color='black', jitter=0.2)

# Legend
plt.title('Distribution of the Mean Squared Error for different trees')
plt.xlabel('Tree')
plt.ylabel('Mean Squared Error (MSE)')

# Afficher le graphique
plt.show()
```



#RF (Step 3)

```
# estimate the RF with tree = 10
RF = RandomForestRegressor(n_estimators=10, random_state=100)
fit3 = RF.fit(X_train,Y_train)
```

#RF (Step 4)

```
# Compute the Predictionfrom RF
Predict_RF = RF.predict(X_test)
```

#RF (Step 5)

```
# Print feature importances
feature_importances = pd.Series(RF.feature_importances_, index=X_train.columns)
print("Feature Importances:")
```

```
## Feature Importances:
```

```
print(feature_importances)
```

```
## gas        0.916639
## cobalt     0.015486
## uranium    0.026289
## charbon    0.020575
## bau_alu    0.021012
## dtype: float64
```

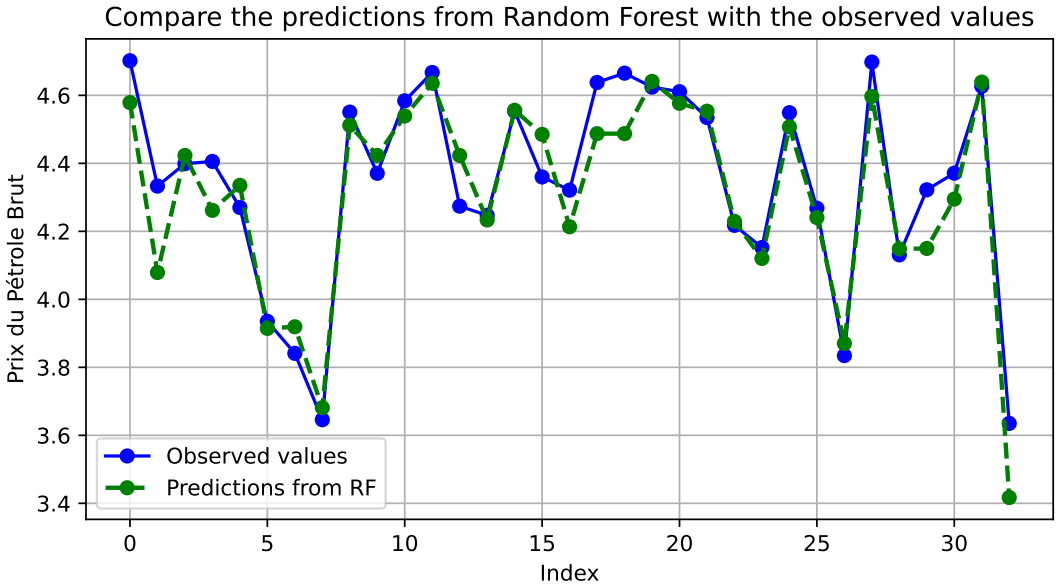
#RF (Step 6)

```
#Computethe Prediction and print the coef for Ridge regression
print('MSE (training): %.2f' % mean_squared_error(Y_train, RF.predict(X_train)),
      'MSE (test): %.2f' % mean_squared_error(Y_test, RF.predict(X_test)), sep='\n')
```

```
## MSE (training): 0.00
## MSE (test): 0.01
```

#RF (Step 7)

```
# Create a graph to compare predictions from RF and observed values
plt.figure(figsize=(8, 4))
plt.plot(Y_test.values, color='blue', marker='o', label='Observed values')
plt.plot(Predict_RF, color='green', marker='o', linestyle='dashed',
linewidth=2, markersize=6, label='Predictions from RF')
plt.xlabel('Index')
plt.ylabel('Prix du Pétrole Brut')
plt.title ('Compare the predictions from Random Forest with the observed values')
plt.legend()
plt.grid(True)
plt.show()
```


Compare the predictions from Random Forest with the observed values
Prix du Pétrole Brut
Index
Observed values
Predictions from RF
4.6
4.4
4.2
4.0
3.8
3.6
3.4
0
5
10
15
20
25
30