Prédiction des flux migratoires

February 8, 2026

Ce notebook constitue le support technique des travaux sur les flux migratoires présentés en partie II.B de ma note de synthèse. Le rapport d'étape joint détaille la phase exploratoire (ou revue de littérature) menée jusqu'à mi-décembre, tandis que ce support présente les avancées réalisées début janvier concernant l'optimisation de la prédiction (de manière assez informelle pour le moment). Il expose notamment la mise en œuvre de l'algorithme de Random Forest et la détection des interactions non linéaires (synergie de proximité, seuil de richesse), ayant permis de réduire l'erreur de prédiction (MAPE) d'un facteur 4 par rapport au modèle de gravité classique.

[8]:
```
import pandas as pd
import statsmodels.formula.api as smf
import os
import numpy as np
import statsmodels.formula.api as smf
import geopandas as gpd
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import r2_score
from sklearn.inspection import PartialDependenceDisplay
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error
from cmdstanpy import CmdStanModel
```

Quelques analyses descriptives simples, et présentation visuelle rapide de la base de données.

[9]:
```
df_main = pd.read_csv("data/data_prediction.csv")
#print(gravity_data.head())
df_main.columns
```

[9]:
```
Index(['iso3_o', 'iso3_d', 'year', 'migrantCount', 'distcap', 'contig',
       'comlang_off', 'col_dep_ever', 'pop_o', 'pop_15_64_pct_x',
       'pop_65_plus_pct_x', 'IMR_o', 'urban_o', 'LA_o', 'PSR_o', 'pop_d',
       'pop_15_64_pct_y', 'pop_65_plus_pct_y', 'IMR_d', 'urban_d', 'LA_d',
       'PSR_d', 'log_migrantCount', 'log_distcap', 'log_pop_o', 'log_pop_d',
       'log_PSR_o', 'log_PSR_d', 'log_IMR_o', 'log_IMR_d', 'log_urban_o',
       'log_urban_d', 'log_LA_o', 'log_LA_d', 't_centered', 't_centered_sq',
       'landlocked_o', 'landlocked_d', 'gdp_o', 'gdp_d', 'gdpcap_o',
       'gdpcap_d', 'gdp_o_lag', 'gdp_d_lag', 'gdpcap_o_lag', 'gdpcap_d_lag',
       'log_gdp_o', 'log_gdp_d', 'log_gdpcap_o', 'log_gdpcap_d',
       'log_gdp_o_lag', 'log_gdp_d_lag', 'log_gdpcap_o_lag',
       'log_gdpcap_d_lag'],
```

```
      dtype='object')
```

[10]:
```
df_main.describe()
```

[10]:
```
                year  migrantCount        distcap         contig  \
count  105646.000000  1.056460e+05  105646.000000  105646.000000
mean     2000.088787  5.212587e+03    6974.511722       0.033565
std         7.055936  4.314272e+04    4454.808329       0.180107
min      1990.000000  1.000000e+00      59.617230       0.000000
25%      1995.000000  6.000000e+00    3283.508000       0.000000
50%      2000.000000  5.200000e+01    6403.348000       0.000000
75%      2005.000000  6.570000e+02    9853.350000       0.000000
max      2010.000000  3.817502e+06   19812.040000       1.000000

        comlang_off   col_dep_ever         pop_o  pop_15_64_pct_x  \
count  105646.00000  105646.000000  1.056460e+05    105646.000000
mean        0.18403       0.022831  5.144637e+07        62.200044
std         0.38751       0.149365  1.474113e+08         6.719394
min         0.00000       0.000000  6.275300e+04        45.776877
25%         0.00000       0.000000  4.133900e+06        56.703262
50%         0.00000       0.000000  1.057310e+07        64.400281
75%         0.00000       0.000000  4.286928e+07        67.293320
max         1.00000       1.000000  1.337705e+09        85.249494

       pop_65_plus_pct_x          IMR_o  …  gdpcap_o_lag  gdpcap_d_lag  \
count      105646.000000  105646.000000  …  76880.000000  77674.000000
mean            7.967482      31.449067  …     10.534350     10.840615
std             5.226991      31.461213  …     13.298655     13.401745
min             0.862465       1.700000  …      0.065000      0.065000
25%             3.455495       6.800000  …      0.879000      0.999000
50%             5.569668      17.700000  …      3.604000      3.863000
75%            12.536309      48.000000  …     18.784000     19.787000
max            23.098026     169.600000  …     79.594000     79.594000

          log_gdp_o     log_gdp_d  log_gdpcap_o  log_gdpcap_d  log_gdp_o_lag  \
count  96187.000000  97156.000000  96187.000000  97156.000000   76880.000000
mean      17.546220     17.636564      1.422055      1.498213      17.351932
std        2.464610      2.416170      1.681087      1.657421       2.463418
min       10.254632     10.254632     -2.733368     -2.733368      10.254632
25%       15.673549     15.787011     -0.001001      0.183155      15.484220
50%       17.541375     17.647761      1.474992      1.558145      17.252137
75%       19.393575     19.420061      3.025485      3.039845      19.284989
max       23.428940     23.428940      4.633340      4.633340      23.295397

       log_gdp_d_lag  log_gdpcap_o_lag  log_gdpcap_d_lag
count   77674.000000      76880.000000      77674.000000
mean       17.451352          1.267002          1.351153
```

```
std         2.407656          1.685979          1.648312
min        10.254632         -2.733368         -2.733368
25%        15.599287         -0.128970         -0.001001
50%        17.427066          1.282044          1.351444
75%        19.325108          2.933005          2.985025
max        23.295397          4.376939          4.376939

[8 rows x 52 columns]
```

[11]:
```
df_main.head(10)
```

[11]:

	iso3_o	iso3_d	year	migrantCount	distcap	contig	comlang_off	\
0	ABW	AGO	2005	1	9516.913	0.0	0.0	
1	ABW	ARE	2005	1	12735.010	0.0	0.0	
2	ABW	ARE	2010	1	12735.010	0.0	0.0	
3	ABW	ARG	1995	1	5396.220	0.0	1.0	
4	ABW	ARG	2000	1	5396.220	0.0	1.0	
5	ABW	ARG	2005	1	5396.220	0.0	1.0	
6	ABW	ARG	2010	1	5396.220	0.0	1.0	
7	ABW	ATG	1990	1	1024.995	0.0	0.0	
8	ABW	ATG	1995	2	1024.995	0.0	0.0	
9	ABW	ATG	2000	1	1024.995	0.0	0.0	

	col_dep_ever	pop_o	pop_15_64_pct_x	…	gdpcap_o_lag	gdpcap_d_lag	\
0	0.0	97635.0	70.108246	…	20.620	0.656	
1	0.0	97635.0	70.108246	…	20.620	34.476	
2	0.0	101838.0	69.709380	…	23.303	43.534	
3	0.0	79805.0	67.362813	…	NaN	4.333	
4	0.0	90588.0	69.774621	…	16.441	8.973	
5	0.0	97635.0	70.108246	…	20.620	9.329	
6	0.0	101838.0	69.709380	…	23.303	5.768	
7	0.0	62753.0	67.390157	…	NaN	NaN	
8	0.0	79805.0	67.362813	…	NaN	6.325	
9	0.0	90588.0	69.774621	…	16.441	7.230	

	log_gdp_o	log_gdp_d	log_gdpcap_o	log_gdpcap_d	log_gdp_o_lag	\
0	14.661810	17.156027	3.148582	0.534737	14.443294	
1	14.661810	19.011889	3.148582	3.773542	14.443294	
2	14.718799	19.471675	3.190024	3.523002	14.661810	
3	14.093650	19.560336	2.799778	2.194220	NaN	
4	14.443294	19.656945	3.026261	2.233128	14.093650	
5	14.661810	19.222268	3.148582	1.752325	14.443294	
6	14.718799	19.952598	3.190024	2.438863	14.661810	
7	NaN	12.877921	NaN	1.844510	NaN	
8	14.093650	13.110666	2.799778	1.978239	NaN	
9	14.443294	13.571957	3.026261	2.312040	14.093650	

```
   log_gdp_d_lag  log_gdpcap_o_lag  log_gdpcap_d_lag
0      16.027032          3.026261         -0.421594
1      18.463140          3.026261          3.540263
2      19.011889          3.148582          3.773542
3      18.766766               NaN          1.466260
4      19.560336          2.799778          2.194220
5      19.656945          3.026261          2.233128
6      19.222268          3.148582          1.752325
7            NaN               NaN               NaN
8      12.877921               NaN          1.844510
9      13.110666          2.799778          1.978239

[10 rows x 54 columns]
```

Régression du modèle de gravité de Welch & raftery répliqué à l'identique (donc sans variable de richesse PIB)

[12]:
```
formula = """
log_migrantCount ~ log_pop_o + log_pop_d + log_distcap
+ log_PSR_o + log_PSR_d + log_IMR_o + log_IMR_d
+ log_urban_o + log_urban_d + log_LA_o + log_LA_d
+ landlocked_o + landlocked_d
+ contig + comlang_off + col_dep_ever
+ t_centered + t_centered_sq
"""


# cov_type='HC1' heteroskedasticity
results = smf.ols(formula, data=df_main.dropna()).fit(cov_type='HC1')


print(results.summary())
print(f"R-squared précis : {results.rsquared:.6f}")
R2_previous = results.rsquared
print(f"Adj. R-squared   : {results.rsquared_adj:.6f}")
R2_adj_previous = results.rsquared_adj
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_migrantCount   R-squared:                       0.549
Model:                           OLS   Adj. R-squared:                  0.549
Method:                Least Squares   F-statistic:                     5434.
Date:               Fri, 06 Feb 2026   Prob (F-statistic):               0.00
Time:                       20:03:10   Log-Likelihood:            -1.4888e+05
No. Observations:              69554   AIC:                         2.978e+05
Df Residuals:                  69535   BIC:                         2.980e+05
Df Model:                         18
Covariance Type:                 HC1
```

```
==============================================================================
=
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept      -2.3023      0.213    -10.821      0.000      -2.719
-1.885
log_pop_o       0.5778      0.007     82.811      0.000       0.564
0.592
log_pop_d       0.4945      0.007     71.836      0.000       0.481
0.508
log_distcap    -1.2510      0.010   -119.696      0.000      -1.271
-1.230
log_PSR_o       0.1198      0.019      6.146      0.000       0.082
0.158
log_PSR_d      -0.3043      0.022    -13.914      0.000      -0.347
-0.261
log_IMR_o      -0.7149      0.014    -50.623      0.000      -0.743
-0.687
log_IMR_d      -0.7979      0.015    -53.294      0.000      -0.827
-0.769
log_urban_o    -0.1656      0.024     -6.865      0.000      -0.213
-0.118
log_urban_d     0.2380      0.026      9.236      0.000       0.188
0.289
log_LA_o        0.1414      0.005     26.278      0.000       0.131
0.152
log_LA_d        0.2078      0.005     38.147      0.000       0.197
0.218
landlocked_o   -0.4170      0.022    -18.556      0.000      -0.461
-0.373
landlocked_d   -0.1170      0.022     -5.391      0.000      -0.160
-0.074
contig          1.6968      0.056     30.247      0.000       1.587
1.807
comlang_off     1.7930      0.021     83.808      0.000       1.751
1.835
col_dep_ever    1.6725      0.050     33.265      0.000       1.574
1.771
t_centered     -0.0628      0.002    -28.009      0.000      -0.067
-0.058
t_centered_sq   0.0005      0.000      1.560      0.119      -0.000
0.001
==============================================================================
Omnibus:                      284.507   Durbin-Watson:                   0.595
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              287.913
Skew:                           0.157   Prob(JB):                    3.02e-63
```

```
Kurtosis:                       2.977   Cond. No.                         1.63e+03
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 1.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
R-squared précis : 0.549046
Adj. R-squared   : 0.548929
```

Test de l'ajout de la variable non linéaire log_distance*contig dans le modèle de gravité

[13]:
```
formula_2 = """
log_migrantCount ~ log_pop_o + log_pop_d
+ log_PSR_o + log_PSR_d + log_IMR_o + log_IMR_d
+ log_urban_o + log_urban_d + log_LA_o + log_LA_d
+ landlocked_o + landlocked_d
+ comlang_off + col_dep_ever
+ t_centered + t_centered_sq + log_distcap*contig
"""


# cov_type='HC1' heteroskedasticity
results2 = smf.ols(formula_2, data=df_main.dropna()).fit(cov_type='HC1')


print(results2.summary())

print(f"R-squared précis : {results2.rsquared:.6f}")
print(f"Adj. R-squared   : {results2.rsquared_adj:.6f}")

R2_new = results2.rsquared
R2_adj_new = results2.rsquared_adj
print(f"Changement R-squared : {R2_new - R2_previous:.6f}")
print(f"Changement Adj. R-squared : {R2_adj_new - R2_adj_previous:.6f}")
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_migrantCount   R-squared:                       0.549
Model:                           OLS   Adj. R-squared:                  0.549
Method:                Least Squares   F-statistic:                     5195.
Date:               Fri, 06 Feb 2026   Prob (F-statistic):               0.00
Time:                       20:03:10   Log-Likelihood:            -1.4888e+05
No. Observations:              69554   AIC:                         2.978e+05
Df Residuals:                  69534   BIC:                         2.980e+05
Df Model:                         19
Covariance Type:                 HC1
==============================================================================
======
```

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-2.2520	0.214	-10.515	0.000	-2.672	-1.832
log_pop_o	0.5778	0.007	82.791	0.000	0.564	0.591
log_pop_d	0.4945	0.007	71.818	0.000	0.481	0.508
log_PSR_o	0.1196	0.019	6.140	0.000	0.081	0.158
log_PSR_d	-0.3045	0.022	-13.927	0.000	-0.347	-0.262
log_IMR_o	-0.7155	0.014	-50.650	0.000	-0.743	-0.688
log_IMR_d	-0.7983	0.015	-53.308	0.000	-0.828	-0.769
log_urban_o	-0.1665	0.024	-6.898	0.000	-0.214	-0.119
log_urban_d	0.2376	0.026	9.216	0.000	0.187	0.288
log_LA_o	0.1411	0.005	26.214	0.000	0.131	0.152
log_LA_d	0.2075	0.005	38.082	0.000	0.197	0.218
landlocked_o	-0.4173	0.022	-18.577	0.000	-0.461	-0.373
landlocked_d	-0.1172	0.022	-5.402	0.000	-0.160	-0.075
comlang_off	1.7933	0.021	83.825	0.000	1.751	1.835
col_dep_ever	1.6725	0.050	33.304	0.000	1.574	1.771
t_centered	-0.0628	0.002	-28.037	0.000	-0.067	-0.058
t_centered_sq	0.0005	0.000	1.561	0.119	-0.000	0.001
log_distcap	-1.2547	0.011	-119.042	0.000	-1.275	-1.234
contig	0.8583	0.437	1.964	0.050	0.002	1.715
log_distcap:contig	0.1249	0.064	1.939	0.052	-0.001	0.251

Omnibus:	284.815	Durbin-Watson:	0.595
Prob(Omnibus):	0.000	Jarque-Bera (JB):	288.235
Skew:	0.157	Prob(JB):	2.57e-63

```
Kurtosis:                       2.978   Cond. No.                         2.91e+03
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 2.91e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
R-squared précis : 0.549078
Adj. R-squared   : 0.548955
Changement R-squared : 0.000033
Changement Adj. R-squared : 0.000026
```

Ajout de la variable de richesse PIB (GDP) : tout d'abord, il paraît plus pertinent d'intégrer le PIB par tête plutôt que le PIB nominal. On regarde quand même par curiosité la différence entre une regression avec GDP et un regression avec GDP per capita.

[14]:
```
formula = """
log_migrantCount ~ log_pop_o + log_pop_d + log_distcap
+ log_PSR_o + log_PSR_d + log_IMR_o + log_IMR_d
+ log_urban_o + log_urban_d + log_LA_o + log_LA_d
+ landlocked_o + landlocked_d
+ contig + comlang_off + col_dep_ever
+ t_centered + t_centered_sq + log_gdp_o + log_gdp_d
"""

# cov_type='HC1' pour heteroskedascticity
results1 = smf.ols(formula, data=df_main.dropna()).fit(cov_type='HC1')


print(results1.summary())

formula= """log_migrantCount ~ log_pop_o + log_pop_d + log_distcap
+ log_PSR_o + log_PSR_d + log_IMR_o + log_IMR_d
+ log_urban_o + log_urban_d + log_LA_o + log_LA_d
+ landlocked_o + landlocked_d
+ contig + comlang_off + col_dep_ever
+ t_centered + t_centered_sq + log_gdpcap_o + log_gdpcap_d"""
results2 = smf.ols(formula, data=df_main.dropna()).fit(cov_type='HC1')
print(results2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_migrantCount   R-squared:                       0.568
Model:                           OLS   Adj. R-squared:                  0.568
Method:                Least Squares   F-statistic:                     5222.
Date:               Fri, 06 Feb 2026   Prob (F-statistic):               0.00
Time:                       20:03:11   Log-Likelihood:            -1.4741e+05
No. Observations:              69554   AIC:                         2.949e+05
Df Residuals:                  69533   BIC:                         2.950e+05
```

```
Df Model:                           20
Covariance Type:                   HC1
==============================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept        -2.9794      0.208    -14.310      0.000      -3.387
-2.571
log_pop_o         0.2363      0.014     16.756      0.000       0.209
0.264
log_pop_d        -0.0775      0.014     -5.535      0.000      -0.105
-0.050
log_distcap      -1.3047      0.010   -127.166      0.000      -1.325
-1.285
log_PSR_o         0.0292      0.019      1.512      0.130      -0.009
0.067
log_PSR_d        -0.4569      0.022    -21.157      0.000      -0.499
-0.415
log_IMR_o        -0.2788      0.021    -13.254      0.000      -0.320
-0.238
log_IMR_d        -0.0724      0.021     -3.434      0.001      -0.114
-0.031
log_urban_o      -0.4726      0.026    -17.988      0.000      -0.524
-0.421
log_urban_d      -0.3054      0.028    -10.853      0.000      -0.361
-0.250
log_LA_o          0.1315      0.005     24.746      0.000       0.121
0.142
log_LA_d          0.1836      0.005     34.175      0.000       0.173
0.194
landlocked_o     -0.4088      0.022    -18.505      0.000      -0.452
-0.366
landlocked_d     -0.1134      0.021     -5.306      0.000      -0.155
-0.072
contig            1.6809      0.057     29.633      0.000       1.570
1.792
comlang_off       1.6289      0.021     76.036      0.000       1.587
1.671
col_dep_ever      1.6069      0.051     31.295      0.000       1.506
1.708
t_centered       -0.0516      0.002    -23.209      0.000      -0.056
-0.047
t_centered_sq    -0.0036      0.000    -11.359      0.000      -0.004
-0.003
log_gdp_o         0.3648      0.013     27.885      0.000       0.339
0.390
```

```
log_gdp_d         0.6073      0.013     46.582      0.000       0.582
0.633
==============================================================================
Omnibus:                      440.859   Durbin-Watson:                   0.609
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              450.022
Skew:                           0.192   Prob(JB):                     1.90e-98
Kurtosis:                       3.085   Cond. No.                     1.73e+03
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 1.73e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_migrantCount   R-squared:                       0.568
Model:                           OLS   Adj. R-squared:                  0.568
Method:                Least Squares   F-statistic:                     5222.
Date:               Fri, 06 Feb 2026   Prob (F-statistic):               0.00
Time:                       20:03:11   Log-Likelihood:            -1.4740e+05
No. Observations:              69554   AIC:                         2.948e+05
Df Residuals:                  69533   BIC:                         2.950e+05
Df Model:                         20
Covariance Type:                 HC1
==============================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept        -2.8610      0.208    -13.747      0.000      -3.269
-2.453
log_pop_o         0.5998      0.007     87.218      0.000       0.586
0.613
log_pop_d         0.5280      0.007     77.881      0.000       0.515
0.541
log_distcap      -1.3032      0.010   -127.165      0.000      -1.323
-1.283
log_PSR_o         0.0319      0.019      1.656      0.098      -0.006
0.070
log_PSR_d        -0.4537      0.022    -21.018      0.000      -0.496
-0.411
log_IMR_o        -0.2930      0.021    -14.015      0.000      -0.334
-0.252
log_IMR_d        -0.0866      0.021     -4.140      0.000      -0.128
-0.046
log_urban_o      -0.4743      0.026    -18.043      0.000      -0.526
-0.423
```

```
log_urban_d      -0.3199      0.028    -11.341      0.000      -0.375
-0.265
log_LA_o          0.1332      0.005     25.087      0.000       0.123
0.144
log_LA_d          0.1862      0.005     34.666      0.000       0.176
0.197
landlocked_o     -0.4087      0.022    -18.487      0.000      -0.452
-0.365
landlocked_d     -0.1139      0.021     -5.325      0.000      -0.156
-0.072
contig            1.6876      0.057     29.774      0.000       1.576
1.799
comlang_off       1.6298      0.021     76.162      0.000       1.588
1.672
col_dep_ever      1.6029      0.051     31.141      0.000       1.502
1.704
t_centered       -0.0518      0.002    -23.308      0.000      -0.056
-0.047
t_centered_sq    -0.0036      0.000    -11.279      0.000      -0.004
-0.003
log_gdpcap_o      0.3587      0.013     27.501      0.000       0.333
0.384
log_gdpcap_d      0.6058      0.013     46.695      0.000       0.580
0.631
==============================================================================
Omnibus:                      440.837   Durbin-Watson:                   0.610
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              449.976
Skew:                           0.192   Prob(JB):                     1.95e-98
Kurtosis:                       3.085   Cond. No.                     1.63e+03
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 1.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

On regarde la différence dans les résultats de l'OLS entre le PIB par tête retardé (t-5) et le PIB par tête courant.

[15]:
```
# TEST GDP RETARDÉ vs GDP COURANT

formula = """
log_migrantCount ~ log_pop_o + log_pop_d + log_distcap
+ log_PSR_o + log_PSR_d + log_IMR_o + log_IMR_d
+ log_urban_o + log_urban_d + log_LA_o + log_LA_d
+ landlocked_o + landlocked_d
+ contig + comlang_off + col_dep_ever
+ t_centered + t_centered_sq + log_gdpcap_o_lag + log_gdpcap_d_lag
```

```
"""


# cov_type='HC1' pour heteroskedascticity
results1 = smf.ols(formula, data=df_main.dropna()).fit(cov_type='HC1')


print(results1.summary())

formula= """log_migrantCount ~ log_pop_o + log_pop_d + log_distcap
+ log_PSR_o + log_PSR_d + log_IMR_o + log_IMR_d
+ log_urban_o + log_urban_d + log_LA_o + log_LA_d
+ landlocked_o + landlocked_d
+ contig + comlang_off + col_dep_ever
+ t_centered + t_centered_sq + log_gdpcap_o + log_gdpcap_d"""
results2 = smf.ols(formula, data=df_main.dropna()).fit(cov_type='HC1')
print(results2.summary())
```

OLS Regression Results

Dep. Variable:	log_migrantCount	R-squared:	0.569
Model:	OLS	Adj. R-squared:	0.569
Method:	Least Squares	F-statistic:	5215.
Date:	Fri, 06 Feb 2026	Prob (F-statistic):	0.00
Time:	20:03:12	Log-Likelihood:	-1.4734e+05
No. Observations:	69554	AIC:	2.947e+05
Df Residuals:	69533	BIC:	2.949e+05
Df Model:	20		
Covariance Type:	HC1		

	coef	std err	z	P>|z|	[0.025	0.975]
Intercept	-3.0329	0.209	-14.540	0.000	-3.442	-2.624
log_pop_o	0.6037	0.007	87.594	0.000	0.590	0.617
log_pop_d	0.5340	0.007	78.578	0.000	0.521	0.547
log_distcap	-1.3050	0.010	-127.612	0.000	-1.325	-1.285
log_PSR_o	0.0383	0.019	1.997	0.046	0.001	0.076
log_PSR_d	-0.4388	0.022	-20.327	0.000	-0.481	-0.397
log_IMR_o	-0.3057	0.020	-15.109	0.000	-0.345	

```
-0.266
log_IMR_d           -0.1171      0.020     -5.752      0.000      -0.157
-0.077
log_urban_o         -0.4413      0.026    -17.242      0.000      -0.491
-0.391
log_urban_d         -0.2513      0.028     -9.104      0.000      -0.305
-0.197
log_LA_o             0.1311      0.005     24.607      0.000       0.121
0.142
log_LA_d             0.1828      0.005     33.810      0.000       0.172
0.193
landlocked_o        -0.4047      0.022    -18.313      0.000      -0.448
-0.361
landlocked_d        -0.0992      0.021     -4.627      0.000      -0.141
-0.057
contig               1.6984      0.056     30.145      0.000       1.588
1.809
comlang_off          1.6056      0.021     74.992      0.000       1.564
1.648
col_dep_ever         1.5950      0.051     31.010      0.000       1.494
1.696
t_centered          -0.0376      0.002    -16.631      0.000      -0.042
-0.033
t_centered_sq       -0.0025      0.000     -7.977      0.000      -0.003
-0.002
log_gdpcap_o_lag     0.3443      0.012     28.514      0.000       0.321
0.368
log_gdpcap_d_lag     0.5712      0.012     47.773      0.000       0.548
0.595
==============================================================================
Omnibus:                      457.992   Durbin-Watson:                   0.611
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              467.994
Skew:                           0.196   Prob(JB):                    2.38e-102
Kurtosis:                       3.090   Cond. No.                     1.63e+03
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 1.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
                            OLS Regression Results
==============================================================================
Dep. Variable:       log_migrantCount   R-squared:                       0.568
Model:                           OLS   Adj. R-squared:                  0.568
Method:                Least Squares   F-statistic:                     5222.
Date:               Fri, 06 Feb 2026   Prob (F-statistic):               0.00
Time:                       20:03:12   Log-Likelihood:            -1.4740e+05
No. Observations:              69554   AIC:                         2.948e+05
```

```
Df Residuals:                    69533   BIC:                          2.950e+05
Df Model:                           20
Covariance Type:                   HC1
==============================================================================
=
                    coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-
Intercept        -2.8610      0.208    -13.747      0.000      -3.269
-2.453
log_pop_o         0.5998      0.007     87.218      0.000       0.586
0.613
log_pop_d         0.5280      0.007     77.881      0.000       0.515
0.541
log_distcap      -1.3032      0.010   -127.165      0.000      -1.323
-1.283
log_PSR_o         0.0319      0.019      1.656      0.098      -0.006
0.070
log_PSR_d        -0.4537      0.022    -21.018      0.000      -0.496
-0.411
log_IMR_o        -0.2930      0.021    -14.015      0.000      -0.334
-0.252
log_IMR_d        -0.0866      0.021     -4.140      0.000      -0.128
-0.046
log_urban_o      -0.4743      0.026    -18.043      0.000      -0.526
-0.423
log_urban_d      -0.3199      0.028    -11.341      0.000      -0.375
-0.265
log_LA_o          0.1332      0.005     25.087      0.000       0.123
0.144
log_LA_d          0.1862      0.005     34.666      0.000       0.176
0.197
landlocked_o     -0.4087      0.022    -18.487      0.000      -0.452
-0.365
landlocked_d     -0.1139      0.021     -5.325      0.000      -0.156
-0.072
contig            1.6876      0.057     29.774      0.000       1.576
1.799
comlang_off       1.6298      0.021     76.162      0.000       1.588
1.672
col_dep_ever      1.6029      0.051     31.141      0.000       1.502
1.704
t_centered       -0.0518      0.002    -23.308      0.000      -0.056
-0.047
t_centered_sq    -0.0036      0.000    -11.279      0.000      -0.004
-0.003
log_gdpcap_o      0.3587      0.013     27.501      0.000       0.333
```

```
0.384
log_gdpcap_d      0.6058      0.013      46.695      0.000       0.580
0.631
==============================================================================
Omnibus:                      440.837   Durbin-Watson:                   0.610
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              449.976
Skew:                           0.192   Prob(JB):                     1.95e-98
Kurtosis:                       3.085   Cond. No.                     1.63e+03
==============================================================================

Notes:
[1] Standard Errors are heteroscedasticity robust (HC1)
[2] The condition number is large, 1.63e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

On compare la performance prédictive du modèle de gravité avec celui de Welch & Raftery (qui obtiennent une erreur MAPE de 1500%)

[16]:
```python
#  Liste EXHAUSTIVE des variables du modèle

all_features = [
    'log_migrantCount',
    'log_pop_o', 'log_pop_d', 'log_distcap',
    'log_PSR_o', 'log_PSR_d', 'log_IMR_o', 'log_IMR_d',
    'log_urban_o', 'log_urban_d', 'log_LA_o', 'log_LA_d',
    'landlocked_o', 'landlocked_d',
    'contig', 'comlang_off', 'col_dep_ever',
    't_centered', 't_centered_sq'
]

#
# On ne garde que les lignes où TOUTES ces variables sont présentes
df_clean = df_main.dropna(subset=all_features).copy()

# Split Train/Test (Test = 2010)
train_df = df_clean[df_clean['year'] < 2010]
test_df  = df_clean[df_clean['year'] == 2010]

print(f"Lignes d'entraînement : {len(train_df)}")
print(f"Lignes de test (2010): {len(test_df)}")

# Train
formula = """
log_migrantCount ~ log_pop_o + log_pop_d + log_distcap
+ log_PSR_o + log_PSR_d + log_IMR_o + log_IMR_d
+ log_urban_o + log_urban_d + log_LA_o + log_LA_d
+ landlocked_o + landlocked_d
+ contig + comlang_off + col_dep_ever
```

```
+ t_centered + t_centered_sq

"""


model = smf.ols(formula, data=train_df).fit()

# Test
pred_log = model.predict(test_df)

y_pred_real = np.exp(pred_log)
y_true_real = np.exp(test_df['log_migrantCount'])

mse = mean_squared_error(y_true_real, y_pred_real)
mape = mean_absolute_percentage_error(y_true_real, y_pred_real)

print(f"\n--- PERFORMANCE OLS ---")
print(f"MSE  : {mse:,.0f}")
print(f"MAPE : {mape:.2%}")
```

```
Lignes d'entraînement : 82963
Lignes de test (2010): 20750

--- PERFORMANCE OLS ---
MSE  : 4,178,612,349
MAPE : 1463.77%
```

On challenge la linéarité du modèle de gravité avec un algorithme Random Forest. Par exemple, le modèle de gravité (linéaire) prédit une augmentation linéaire du flux migratoire à mesure que le PIB augmente. Pourtant, on peut s'attendre à observer une "cloch": lorsque'un pays pauvre commence à avoir les moyens de migrer, la migration se met à augmenter, puis atteint un plateau, puis redescend à mesure que le pays s'enrichit (le pays devient stable, riche, les habitants y restent). Le modèle Random Forest, lui, est capable de détecter ces effets non linéaires. D'où son utilisation. Aussi, l'effet de la distance est très mal jaugé par l'OLS (l'estimateur des moindres carrés, la régression linéaire): la distance influe t-elle de la même manière en fonction de contig = 1 ou 0? On a besoin de dist*contig, que Random Forest gère tout seul.

[17]:
```
# RANDOM FOREST



#PREPARATION DES DONNEES ET SPLIT TEMPOREL


# On reprend tes variables du "Modèle Lagged" (le plus performant␣
 ↪économétriquement)
features = [
    'log_pop_o', 'log_pop_d', 'log_distcap',
```

```
    'log_PSR_o', 'log_PSR_d', 'log_IMR_o', 'log_IMR_d',
    'log_urban_o', 'log_urban_d', 'log_LA_o', 'log_LA_d',
    'landlocked_o', 'landlocked_d',
    'contig', 'comlang_off', 'col_dep_ever',
    't_centered', 't_centered_sq',
    'log_gdpcap_o_lag', 'log_gdpcap_d_lag'
]
target = 'log_migrantCount'

# Nettoyage strict (le RF ne gère pas les NaN)
df_ml = df_main.dropna(subset=features + [target]).copy()

# Split Temporel
# Train : 1990-2005 ; Test : 2010 (Année de prédiction) (il manque l'année 2015)
train_df = df_ml[df_ml['year'] < 2010]
test_df  = df_ml[df_ml['year'] == 2010].copy()

X_train, y_train = train_df[features], train_df[target]
X_test, y_test   = test_df[features], test_df[target]




# FEATURE IMPORTANCE RF


# Choix des arguments :
# - n_estimators=100 : Suffisant pour stabiliser la variance sans exploser le␣
 ↪temps de calcul (nbre de jurés)
# - max_depth=20 : Plus profond que projet elections car les flux migratoires
#   ont une variance plus complexe. Il faut capter des interactions fines.

rf = RandomForestRegressor(n_estimators=100, max_depth=20, random_state=42,␣
 ↪n_jobs=-1)
rf.fit(X_train, y_train)

# Calcul des R^2
r2_train = r2_score(y_train, rf.predict(X_train))
r2_test  = r2_score(y_test, rf.predict(X_test))

print(f"R^2 Random Forest (Entraînement 1990-2005) : {r2_train:.4f}")
print(f"R^2 Random Forest (Prédiction 2010)       : {r2_test:.4f}")

# Hiérarchie des variables (feature importance)
importances = pd.Series(rf.feature_importances_, index=features).
 ↪sort_values(ascending=False)
print("\nImportance des variables ")
```

```
print(importances.head(10))
```

```
R^2 Random Forest (Entraînement 1990-2005) : 0.9778
R^2 Random Forest (Prédiction 2010)       : 0.8010

Importance des variables
log_pop_o           0.177082
log_distcap         0.161586
log_pop_d           0.157749
log_gdpcap_d_lag    0.100145
log_gdpcap_o_lag    0.069036
contig              0.040304
log_LA_d            0.035027
log_IMR_d           0.034333
log_PSR_d           0.034297
log_LA_o            0.033638
dtype: float64
```

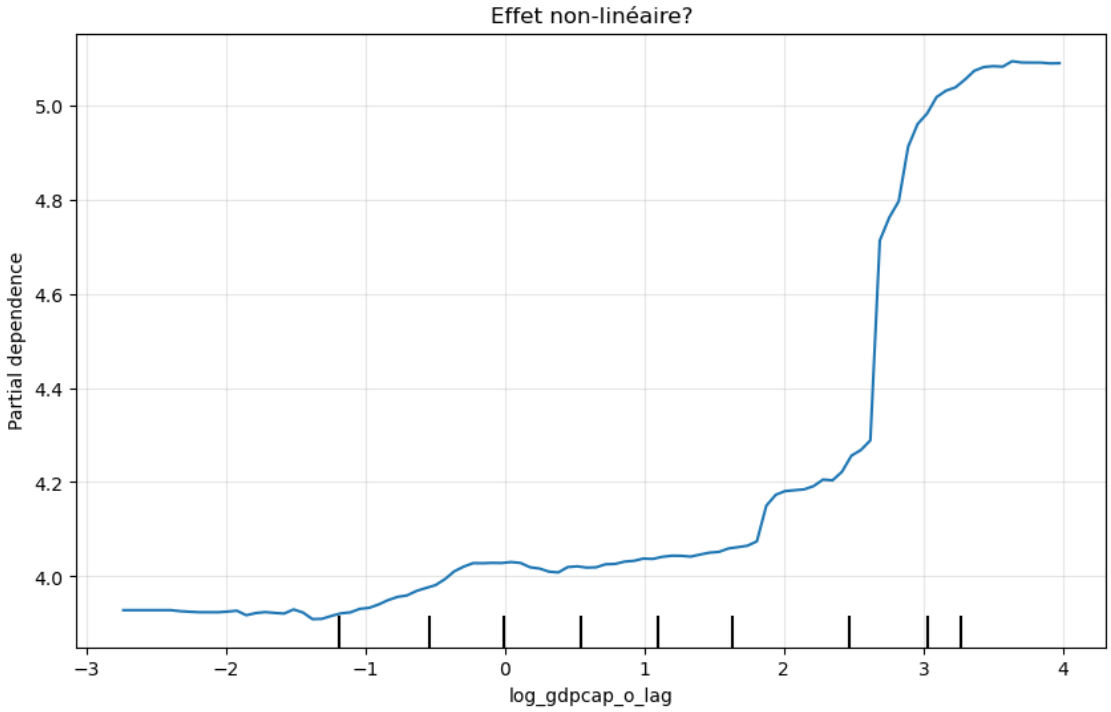
[18]:
```
# Erreurs MSE et MAPE

from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# Prédiction sur le Test (en Log)
y_pred_log = rf.predict(X_test)

#  Retour à l'échelle réelle (Exponentielle)
# On veut comparer des humains, pas des logs d'humains
y_pred_real = np.exp(y_pred_log)
y_true_real = np.exp(y_test)

# calcul des erreurs
mse_rf = mean_squared_error(y_true_real, y_pred_real) #somme des ecarts au carré
mape_rf = mean_absolute_percentage_error(y_true_real, y_pred_real) # somme des␣
 ↪écarts relatifs

print(f"\n PERFORMANCE PRÉDICTIVE (Test 2010) ")
print(f"MSE (Random Forest)  : {mse_rf:,.0f}")
print(f"MAPE (Random Forest) : {mape_rf:.2%}")
```

```
 PERFORMANCE PRÉDICTIVE (Test 2010)
MSE (Random Forest)  : 796,974,241
MAPE (Random Forest) : 396.44%
```

Amélioration de l'erreur MAPE d'un facteur 4 par rapport au modèle de gravité (1600 > 400%) C'est énorme. Le random Forest a bien mieux compris la dynamique du système que le modèle de gravité.

[19]:
```
# PARTIAL DEPENDENCE PLOT (pour gdpcap)


import matplotlib.pyplot as plt
from sklearn.inspection import PartialDependenceDisplay

# Vérif de l'étendue réelle des données
print("Min/Max de la variable :")
print(X_train['log_gdpcap_o_lag'].describe()[['min', 'max']])

fig, ax = plt.subplots(figsize=(10, 6))


PartialDependenceDisplay.from_estimator(
    rf,
    X_train,
    ['log_gdpcap_o_lag'],
    kind="average",
    percentiles=(0, 1),
    ax=ax
)

plt.title("Effet non-linéaire?")
plt.grid(True, alpha=0.3)
plt.show()
```

```
Min/Max de la variable :
min   -2.733368
max    3.973626
Name: log_gdpcap_o_lag, dtype: float64
```



Comment cette courbe est tracée: Pour chaque pays, on fixe le log du PIB par tête d'origine à une certaine valeur, puis on calcule la prédiction du log du nombre de migrants. On répète le processus pour balayer toutes les valeurs possibles de log(gdp_cap_o). C'est une expérience de pensée : que se passerait il si pour le pays en question, le PIB par tête valait X ? et ce, pour tout X.

Ensuite, on fait la moyenne sur tous les pays pour tracer ce graphique. (C'est le point criticable! il y a surtout des effets hétérogènes du PIB entre les pays). => c'est pourquoi nous avons l'idée de diviser la liste des pays en deux, en fixant arbitrairement la limite entre "pays pauvre" et "pays riche" à 12k$ GDP per capita

Log = -3 : 50$ par habitant. (Extrême pauvreté ou erreur de donnée).

Log = 2.5 : 12,100$ par habitant. (point de bascule).

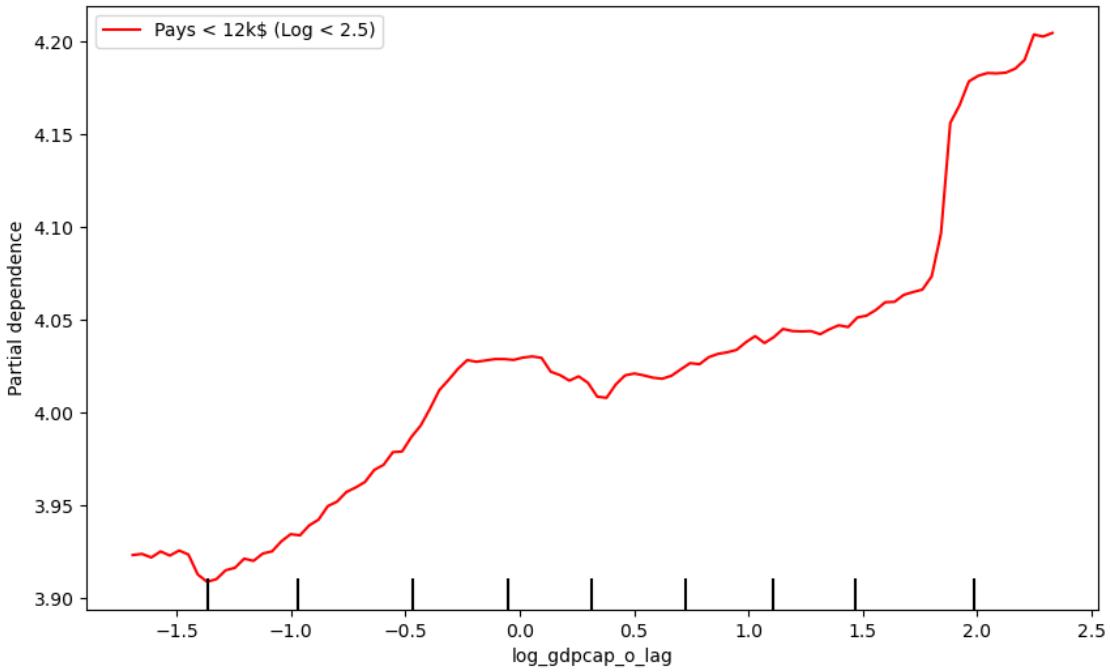
Log = 3.0 : 20 000$ par habitant. (Entrée dans le club des pays occidentaux, c'est le seuil attendu)

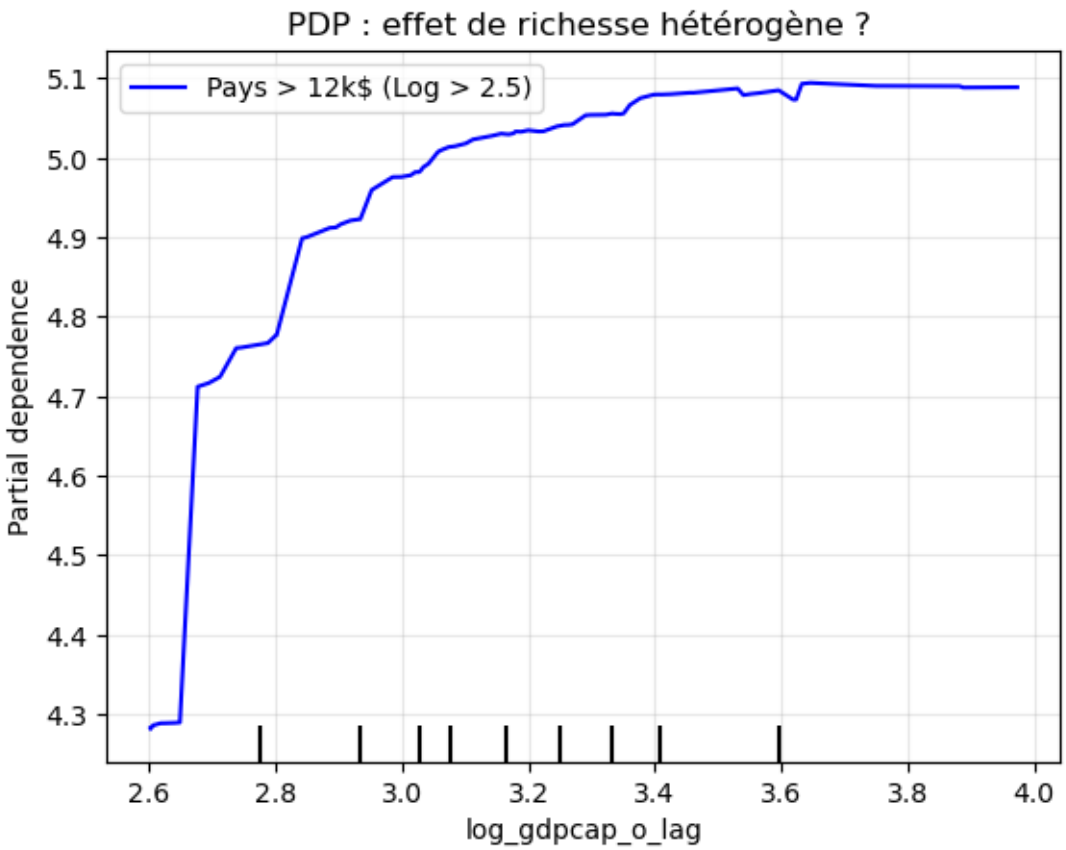
Log = 4.4 : 80 000$ par habitant. (max dans le monde, qatar suisse etc)

Nouvelles courbes (PDP) en divisant en deux groupes selon le critère >12k$ GDP/capita

[20]:
```
# Seuils log ~ 2.5 qui correspond à 12k$ GDP/capita
mask_pauvres = X_train['log_gdpcap_o_lag'] < 2.6
mask_riches  = X_train['log_gdpcap_o_lag'] >= 2.6

fig, ax = plt.subplots(figsize=(10, 6))

#  PDP Pays < Seuil (rouge)
```

```
PartialDependenceDisplay.from_estimator(
    rf,
    X_train[mask_pauvres],
    ['log_gdpcap_o_lag'],
    kind="average",
    line_kw={"color": "red", "label": "Pays < 12k$ (Log < 2.5)"},
    ax=ax
)

#  PDP Pays > Seuil (Bleu)
PartialDependenceDisplay.from_estimator(
    rf,
    X_train[mask_riches],
    ['log_gdpcap_o_lag'],
    kind="average",
    line_kw={"color": "blue", "label": "Pays > 12k$ (Log > 2.5)"},

)

plt.title("PDP : effet de richesse hétérogène ? ")
plt.grid(True, alpha=0.3)
plt.legend()
plt.show()
```

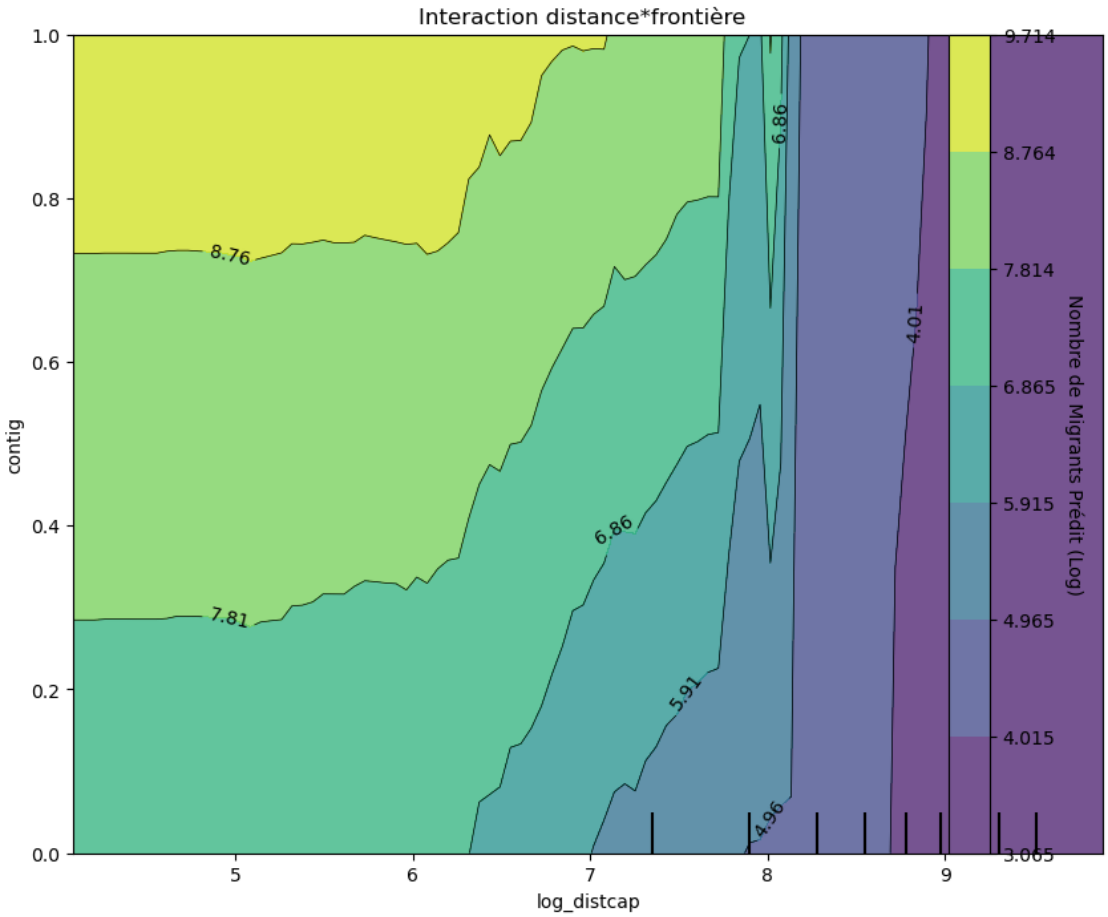




Il existe clairement un seuil de richesse à partir duquel la migration augmente brusquement. L'effet du PIB/tete pour les valeurs <= 12k$ est quasi-linéaire et quasi-nul: augmenter légèrement la richesse chez un pays pauvre n'a PAS d'effet sur la migration. (noter la différence d'échelle entre les deux graphiques /!!! La courbe rouge évolue sur une gamme très étroite de l'ordonnée, signe que la courbe est quasi plate. En revanche, la courbe bleue présente une augmentation nette sur une échelle significative)

[21]:
```
import matplotlib.pyplot as plt
from sklearn.inspection import PartialDependenceDisplay

# PDP 2D

features_inter = [('log_distcap', 'contig')]
# ------------------

fig, ax = plt.subplots(figsize=(10, 8))

graph= PartialDependenceDisplay.from_estimator(
    rf,
    X_train,
```

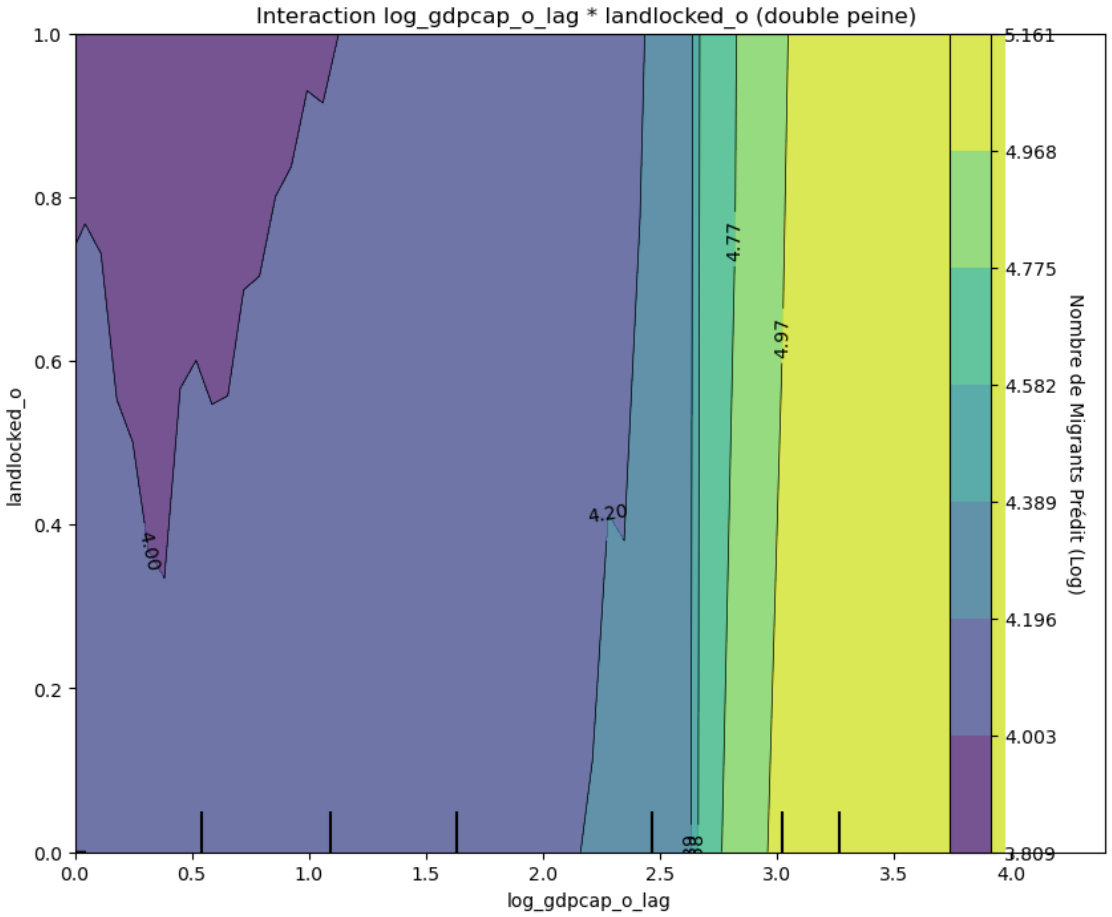
```
    features_inter,
    kind="average",
    percentiles=(0, 1),    # Gamme complète
    n_cols=1,
    ax=ax
)
cbar = fig.colorbar(graph.contours_[0, 0], ax=ax)
cbar.set_label('Nombre de Migrants Prédit (Log)', rotation=270, labelpad=15)
plt.title("Interaction distance*frontière  ")

plt.show()
```



Interprétation plus délicate, graphe d'un Partial Dependance Plot en 3D: on remarque que dans la zone contig=1 et log_distance faible (en haut à gauche) le nombre de migrant prédit est élevé. Au contraire, à contig=0 et log_distance élevée (double peine, mais logique qu'on ne soit pas voisin si la distance est élevée): le nombre de migrant prédit est beaucoup plus faible. Le gradient de couleur n'est pas linéaire et vertical de gauche à droite (contrairement au prochain graphe), c'est le signe que les deux variables contig et distance intéragissent.

Note: ce graphique a pour but de visualiser une possible interaction grossièrement, mais n'est pas rigoureusement interprétable tel quel puisque la variable contig ne prend que les valeurs 0 et 1 (les valeurs intermédiaires n'ont pas de sens)

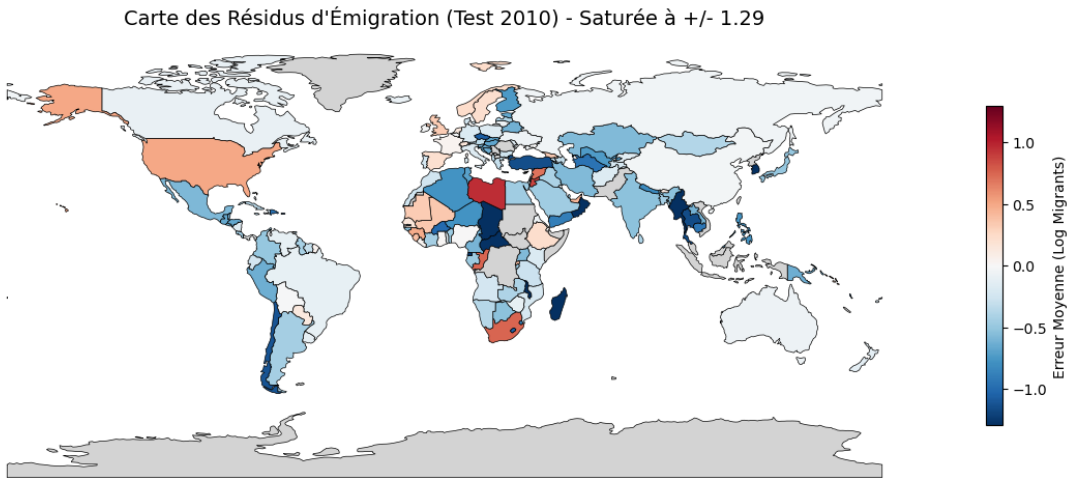
[22]:
```
import matplotlib.pyplot as plt
from sklearn.inspection import PartialDependenceDisplay

# PDP 2D

features_inter = [('log_gdpcap_o_lag', 'landlocked_o')]
# ------------------

fig, ax = plt.subplots(figsize=(10, 8))

graph= PartialDependenceDisplay.from_estimator(
    rf,
    X_train,
    features_inter,
    kind="average",
    percentiles=(0, 1),   # Gamme complète
    n_cols=1,
    ax=ax
)
cbar = fig.colorbar(graph.contours_[0, 0], ax=ax)
cbar.set_label('Nombre de Migrants Prédit (Log)', rotation=270, labelpad=15)
plt.title("Interaction log_gdpcap_o_lag * landlocked_o (double peine) ")
plt.xlim(0,4.4)
plt.show()
```



Ici, nous n'avons pas de preuve forte pour une interaction entre la variable "landlocked" et "log richesse": le gradient de couleur est plutôt linéaire et vertical de gauche à droite (la valeur de la prédiction de migrant pour une abscisse donnée n'est pas corrélée à la valeur de l'ordonnée)

[23]:
```
# Carte des Résidus


# 1) Calcul des résidus
test_df['residuals'] = test_df[target] - rf.predict(X_test)
resid_by_country = test_df.groupby('iso3_o')['residuals'].mean().reset_index()

# 2) Chargement du fond de carte (url directe natural earth)

url_world = "https://naturalearth.s3.amazonaws.com/110m_cultural/
 ↪ne_110m_admin_0_countries.zip"
world = gpd.read_file(url_world)
```

```python
# 3) Harmonisation des codes ISO (Natural Earth utilise souvent 'ADM0_A3' ou␣
 ↪'ISO_A3')
# On s'assure d'avoir une colonne 'iso_a3' minuscule pour votre fusion
col_iso = 'ADM0_A3' if 'ADM0_A3' in world.columns else 'ISO_A3'
world['iso_a3'] = world[col_iso]

# 4) Fusion
world_map = world.merge(resid_by_country, left_on='iso_a3', right_on='iso3_o',␣
 ↪how='left')

# 5) Vizu
v_max = world_map['residuals'].abs().quantile(0.95)

fig, ax = plt.subplots(figsize=(15, 8))
world_map.plot(
    column='residuals',
    cmap='RdBu_r',
    linewidth=0.5,
    edgecolor='black',
    ax=ax,
    legend=True,
    vmin=-v_max,
    vmax=v_max,
    missing_kwds={'color': 'lightgrey', 'label': 'Pas de données'},
    legend_kwds={'label': "Erreur Moyenne (Log Migrants)", 'shrink': 0.5}
)
ax.set_title(f"Carte des Résidus d'Émigration (Test 2010) - Saturée à +/-␣
 ↪{v_max:.2f}", fontsize=14)
ax.axis('off')
plt.show()
```



Commentaires: (PREDICTIONS FAITES SUR L'ANNEE 2010)

ZONES BLEUES (le modèle sur-estime les flux "réels")

- Niger Tchad Burkina Faso RépCentrafricaine: TRAPPE A PAUVRETé que le modèle a réussi à capter (cf courbe à seuil pour gdpcap_o), mais il en sous estime encore la puissance (il prédit un plus faible nombre de migrants, alors qu'en réalité ce nombre est vraiment proche du néant)
- Arabie Saoudite : le modèle voit un pays riche et connecté à l'international, il prédit une migration comparable aux pays occidentaux. En réalité, le pays est stabilisé par sa "rente pétrolière" qui achète et sédentarise les nationaux.
- Chili : isolement géographique par les Andes ? stabilité culturelle, nationaux attachés peut être ?
- Myanmar: dictature qui ferme le pays en 2010.
- madagascar: combinaison de la trappe à pauvreté avec l'isolement insulaire, les billets d'avions sont infiniment chers pour les locaux.
- Algérie : le modèle capte la proxmité coloniale, mais en réalité l'obtention d'un visa Schenghen refroidit le potentiel de proximité coloniale. Le désir de migrer peut être là ou pas, la possiblité légale ne l'est pas vraiment facilement?.
- Kazakhstan: le modèle a appris sur le départ des soviétiques après la chute de l'URSS en 90-2000, et a anticipté des trop gros flux en 2010 alors que l'économie s'était stabilisée.

ZONES ROUGES (le modèle sous-estime les flux "réels")

- Libye: problème ici ? Khadafi règnait jusqu'en 2011 et fermait le pays.
- Israel : diaspora, israeliens ont souvent double nationalité, circulent facilement, hyper-mobilité que le modèle n'a pas capté.
- Rép du Congo Brazzaville: plus stable que le Congo Kinshasa, assez pauvre mais avec liens coloniaux français plus forts que le modèle pensait (variable gdpcap * dummy liens coloniaux ?) Le modèle prédit bien pour Congo belge.
- Afrique du Sud: pays du BRICS très émergent, le modèle prévoit une stabilité et sédentarité. Or la criminalité assez fortement toujours présente pousse les classes moyennes/supérieures à fuir

ATTENTION avec gdpcap à t-5 on se trompe sur les Etats unis (on sous estime) la prédiction. (piste à explorer)

Prise en main de la syntaxe STAN ("boîte noire" pour lancer des simulations de chaînes de Markov par méthode Monte Carlo) Simulation basique, on implémentera un modèle hiérarchique avancé par la suite.

[24]:
```
# liste de variables
features = [
    'log_pop_o', 'log_pop_d', 'log_distcap',
```

```
    'log_PSR_o', 'log_PSR_d', 'log_IMR_o', 'log_IMR_d',
    'log_urban_o', 'log_urban_d', 'log_LA_o', 'log_LA_d',
    'landlocked_o', 'landlocked_d',
    'contig', 'comlang_off', 'col_dep_ever',
    't_centered', 't_centered_sq',
    'log_gdpcap_o_lag', 'log_gdpcap_d_lag'
]

# 1. Préparation des données pour Stan
# Stan ne lit pas les DataFrame, il veut un dictionnaire avec des matrices numpy
stan_data = {
    'N': len(train_df),
    'K': len(features),
    'X': train_df[features].values,            # On passe tout le bloc X d'un␣
 ↪coup
    'y': train_df['log_migrantCount'].values   # La cible
}

# 2. Compilation et Sampling

model = CmdStanModel(stan_file='MCMC.stan')

# On lance 4 chaînes en parallèle

fit = model.sample(data=stan_data, chains=4, iter_sampling=1000)

# 3. Résultats
print(fit.summary())
```

```
20:03:39 - cmdstanpy - INFO - CmdStan start processing
chain 1:   0%|          | 0/2000 [00:00<?, ?it/s, (Warmup)]
chain 2:   0%|          | 0/2000 [00:00<?, ?it/s, (Warmup)]
chain 3:   0%|          | 0/2000 [00:00<?, ?it/s, (Warmup)]
chain 4:   0%|          | 0/2000 [00:00<?, ?it/s, (Warmup)]


22:10:33 - cmdstanpy - INFO - CmdStan done processing.

                  Mean      MCSE    StdDev       MAD            5%  \
lp__     -62609.300000  0.076557  3.275630  3.137920 -62615.200000
alpha        -2.555420  0.004888  0.240302  0.237117     -2.955020
beta[1]       0.592906  0.000148  0.007739  0.007797      0.580163
beta[2]       0.519027  0.000134  0.007863  0.007898      0.506296
beta[3]      -1.294680  0.000210  0.011911  0.011818     -1.314350
beta[4]       0.064468  0.000398  0.022529  0.022022      0.027424
```

beta[5]	-0.422939	0.000371	0.022278	0.022016	-0.459920
beta[6]	-0.352739	0.000469	0.023161	0.024102	-0.390888
beta[7]	-0.193320	0.000474	0.023564	0.023198	-0.230735
beta[8]	-0.475511	0.000492	0.028556	0.028226	-0.522007
beta[9]	-0.187447	0.000512	0.030138	0.030250	-0.237736
beta[10]	0.131098	0.000108	0.006138	0.006238	0.121019
beta[11]	0.182159	0.000110	0.006213	0.006263	0.171821
beta[12]	-0.405546	0.000436	0.026097	0.025940	-0.449002
beta[13]	-0.132531	0.000427	0.026205	0.025775	-0.176047
beta[14]	1.732820	0.000987	0.056523	0.056225	1.639790
beta[15]	1.591970	0.000369	0.023703	0.023415	1.552540
beta[16]	1.597770	0.000897	0.059132	0.059765	1.500640
beta[17]	-0.032056	0.000030	0.002419	0.002368	-0.036048
beta[18]	0.003936	0.000011	0.000730	0.000724	0.002717
beta[19]	0.320674	0.000263	0.014048	0.014045	0.297290
beta[20]	0.511714	0.000282	0.014058	0.013736	0.488709
sigma	2.013080	0.000072	0.006212	0.006164	2.002870

	50%	95%	ESS_bulk	ESS_tail	ESS_bulk/s	R_hat
lp__	-62609.000000	-62604.500000	1835.94	2605.21	0.631801	1.000870
alpha	-2.557580	-2.160550	2435.59	2716.51	0.838156	1.000480
beta[1]	0.592872	0.605603	2769.03	2875.34	0.952903	1.001540
beta[2]	0.518978	0.531962	3480.27	3107.39	1.197660	1.000740
beta[3]	-1.294770	-1.275050	3289.11	2610.84	1.131880	0.999641
beta[4]	0.064903	0.102095	3218.89	2988.61	1.107720	0.999639
beta[5]	-0.423211	-0.386758	3624.12	2872.12	1.247170	1.000540
beta[6]	-0.352545	-0.315046	2467.08	2900.20	0.848994	1.000950
beta[7]	-0.194027	-0.152794	2481.64	2615.01	0.854006	1.000760
beta[8]	-0.475421	-0.428781	3377.37	3015.50	1.162250	1.001620
beta[9]	-0.186865	-0.138353	3485.92	2953.34	1.199610	1.001430
beta[10]	0.130988	0.141181	3251.43	3075.25	1.118910	1.000410
beta[11]	0.182251	0.192352	3231.53	2969.97	1.112060	1.001090
beta[12]	-0.405455	-0.363135	3608.91	3095.93	1.241930	0.999817
beta[13]	-0.132684	-0.088682	3798.35	2874.07	1.307120	1.001140
beta[14]	1.731850	1.827920	3378.50	3242.38	1.162640	0.999478
beta[15]	1.592840	1.629680	4180.83	2714.69	1.438750	1.000640
beta[16]	1.598150	1.695300	4342.01	3032.77	1.494210	1.000980
beta[17]	-0.032068	-0.028015	6443.61	2972.23	2.217430	1.000950
beta[18]	0.003943	0.005131	4278.67	2694.96	1.472420	1.001640
beta[19]	0.320637	0.344212	2871.50	2965.56	0.988168	1.000770
beta[20]	0.511653	0.534556	2495.34	2373.72	0.858719	1.001940
sigma	2.013020	2.023460	7466.58	3038.48	2.569470	1.000170

[25]:
```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_percentage_error

# 1. Extraction des 4000 simulations (Posterior)
```

```
alpha_samples = fit.stan_variable('alpha')  # shape: (4000,)
beta_samples  = fit.stan_variable('beta')   # shape: (4000, 20)

# 2. Données de Test
X_test = test_df[features].values
y_true_real = np.exp(test_df['log_migrantCount']) # On veut comparer en "vrais" migrants


y_log_pred_samples = alpha_samples + np.dot(X_test, beta_samples.T)

# 4. Aggrégation : On prend la moyenne des prédictions (en log) puis on exponentie
y_pred_mean = np.exp(y_log_pred_samples.mean(axis=1))

# 5. Calcul des erreurs
mse_bayes = mean_squared_error(y_true_real, y_pred_mean)
mape_bayes = mean_absolute_percentage_error(y_true_real, y_pred_mean)

print(f"RÉSULTATS STAN (Test sur 2010) ")
print(f"MSE  : {mse_bayes:,.0f}")
print(f"MAPE : {mape_bayes:.2%}")
```

```
RÉSULTATS STAN (Test sur 2010)
MSE  : 97,836,127,326
MAPE : 1152.20%
```

Meilleur résultats que l'OLS (modèle Stan avec comme prior "beta" entre -10 et 10 permet d'éviter de dire n'importe quoi pour les petits pays, contrairement à OLS), moins bon que Random Forest. Welch & Raftery obtiennent MAPE de 76%, grâce à leur modèle hiérarchique. Notre modèle n'est pas hiérarchique Random Forest qui capte les seuils notamment les trappes à pauvreté, est bien meilleur que notre modèle STAN, mais toujours moins bon que le modèle bayésien hiéararchique de W&R

NOTES DE DERNIERE REUNION : PROJET FLUX MIGRATOIRES

1. TRAITEMENT DES DONNEES ET STATISTIQUES DESCRIPTIVES

Identifier précisément les pays outliers potentiels, notamment le Mexique et les États-Unis. L'objectif est d'inférer les données manquantes si le timing le permet, tout en documentant le biais de sélection dans le rapport final. Comparer le décile supérieur des pays riches au décile le plus bas en pourcentage. Ce résultat, bien que surprenant au premier abord, est validé par les premières observations. Revoir les statistiques descriptives et la corrélation temporelle du taux de départ.

Le projet s'appuie sur le modèle AR(1) de Welch et Raftery pour rendre compte de la dynamique des séries :

$$\log(\delta_{it}) = \mu(1-\phi) + \phi \log(\delta_{it-1}) + \epsilon$$

2. MODELISATION ET AMELIORATION DU MAPE

Tester XGBoost via Scikit-learn en approche “boîte noire” pour comparer les scores MAPE avec les résultats obtenus par Random Forest. Segmenter le jeu de données pour affiner les prédictions, par exemple via la variable de contiguïté. L'idée est d'utiliser des variables indicatrices pour capturer des seuils de rupture non linéaires. Effectuer des prédictions “Out of sample” sur l'année 2020 avec les données disponibles pour valider la robustesse du modèle face aux prédictions “In sample”.

3. CALCULS ET METHODES MCMC Utiliser le serveur de calcul Onyxia pour les traitements lourds. Réaliser une estimation ad hoc par régression sur les indicateurs d'outflow. Ce travail vise à mieux initialiser les paramètres (priors) des algorithmes de Monte-Carlo par chaînes de Markov (MCMC). Analyser les coefficients kappa les plus significatifs pour les flux sortants, principalement pour le Mexique et les USA.
4. STRUCTURE ET REDACTION DU RAPPORT

Le rapport sera structuré en trois temps : Régression linéaire classique servant de baseline. Application du Random Forest pour l'innovation prédictive. Mise en oeuvre des algorithmes MCMC (échantillonnage de Gibbs) et présentation des résultats finaux basés sur Raftery et Welch.

Commencer la rédaction des statistiques descriptives pour validation par le professeur via GitHub.