Modèle de Probabilité de Défaut (PD) par Régression Logistique

In [2]:
```
import numpy as np
import pandas as pd
```

In [4]:
```
x_test = pd.read_csv('C:/Users/IDEAPAD5/Desktop/Python Udemy/2 Model PD/x_test.csv', index_col = 0)
```

In [5]:
```
y_test = pd.read_csv('C:/Users/IDEAPAD5/Desktop/Python Udemy/2 Model PD/y_test.csv', index_col = 0)
```

In [3]:
```
x_train = pd.read_csv('C:/Users/IDEAPAD5/Desktop/Python Udemy/2 Model PD/x_train.csv', index_col = 0)
```

In [6]:
```
y_train = pd.read_csv('C:/Users/IDEAPAD5/Desktop/Python Udemy/2 Model PD/y_train.csv', index_col = 0)
```

In [7]:
```
x_train.head()
```

Out[7]:

	Unnamed: 0	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	...	dti_19.9-20.8	dti_2
344531	344531	21200994	23503865	10000	10000	10000.0	36 months	11.67	330.57	B	...	0	
328300	328300	23904923	26277659	11000	11000	11000.0	36 months	12.49	367.94	B	...	0	
299890	299890	28603222	31126369	10950	10950	10950.0	60 months	18.99	283.99	E	...	0	
439226	439226	12325666	14317820	35000	35000	35000.0	36 months	15.31	1218.61	C	...	0	
167889	167889	3294813	4066999	8000	8000	7975.0	36 months	12.12	266.18	B	...	0	

5 rows × 286 columns

In [8]:
```
y_train.head()
```

Out[8]:

	Bonus_malus
344531	1
328300	1
299890	1
439226	1
167889	0

In [14]:
```
# Sélectionnons les variables indépendantes que nous utiliserons dans notre modèle.
```

In [16]:
```
x_train_model = x_train.loc[:, [
'Home_MORTGAGE',
'Home_RENT_ANY_OTHER_NONE',
'Home_OWN',
'Adresse_ND_NE_IA_NV',
'Adresse_FL', 'Adresse_AL_HI_MO_NM',
'Adresse_CA', 'Adresse_NC_ID_NJ',
'Adresse_NY',
'Adresse_KY_LA_MD',
'Adresse_MI_AR_AZ_VA_OK_DE_OH',
'Adresse_MN_PA_UT_MA_RI_WA_TN_IN',
'Adresse_CT_IL',
'Adresse_TX',
'Adresse_NH_AK_MT_MS_WY_WV_DC_ME',
'Verification_Not Verified',
'Verification_Source Verified',
'Verification_Verified',
'purpose_ed_pyme_enerren_moving',
'purpose_house_other_wedding_medical_vacation',
'Purpose_major_purchase_improvement_car',
'Purpose_debt_consolidation',
'Purpose_credit_card',
'Grade_A',
'Grade_B',
'Grade_C',
'Grade_D',
'Grade_E',
'Grades_F_G',
'initial_list_status_f',
'initial_list_status_w',
'echeance_36',
'echeance_60',
```

```
'ancianite_<_1',
'ancianite_1a4',
'ancianite_5a6',
'ancianite_7',
'ancianite_8a9',
'ancianite_10+',
'nb_mois_1er_credit_<87',
'nb_mois_1er_credit_87a89',
'nb_mois_1er_credit_89a90',
'nb_mois_1er_credit_90a98',
'nb_mois_1er_credit_98a101',
'nb_mois_1er_credit_101a110',
'nb_mois_1er_credit_110a126',
'nb_mois_1er_credit_126a155',
'nb_mois_1er_credit_>155',
'Revenu_<20K',
'Revenu_20K-30K',
'Revenu_30K-40K',
'Revenu_40K-50K',
'Revenu_50K-60K',
'Revenu_60K-70K',
'Revenu_70K-80K',
'Revenu_80K-90K',
'Revenu_90K-100K',
'Revenu_100K-126K',
'Revenu_126K-152K',
'Revenu_152K-227K',
'Revenu_>227K',
'mths_since_last_delinq_null',
'mths_since_last_delinq_0-4',
'mths_since_last_delinq_4-30',
'mths_since_last_delinq_30-60',
'mths_since_last_delinq_60-83',
'mths_since_last_delinq_83+',
'impaye_2ans_0',
'impaye_2ans_1-4',
'impaye_2ans_>=5',
'total_acc_<=6',
'total_acc_6-22',
'total_acc_22-50',
'total_acc_>50',
'dti<=3.2',
'dti_3.2-8.8',
'dti_8.8-10.4',
'dti_10.4-13.6',
'dti_13.6-16.0',
'dti_16.0-16.7',
'dti_16.7-19.9',
'dti_19.9-20.8',
'dti_20.8-23.2',
'dti_23.2-35.2',
'dti>35.2',
'mths_since_last_record_null',
'mths_since_last_record_0-3',
'mths_since_last_record_3-21',
'mths_since_last_record_21-31',
'mths_since_last_record_31-85',
'mths_since_last_record>85'
]]
```

In [17]:
```
# Pour rappel. pour une regresion logistique on utilise n-1 variables de chaque categorie.
#Générons un dataframe avec les catégories de référence (celles avec le < PdE)
```

In [18]:
```
ref = ['Home_RENT_ANY_OTHER_NONE',
'Adresse_ND_NE_IA_NV',
'Verification_Verified',
'purpose_ed_pyme_enerren_moving',
'Grades_F_G',
'initial_list_status_f',
'echeance_60',
'ancianite_<_1',
'nb_mois_1er_credit_>155',
'Revenu_<20K',
'mths_since_last_delinq_83+',
'impaye_2ans_>=5',
'total_acc_<=6',
'dti_23.2-35.2',
'mths_since_last_record_0-3'
]
```

In [19]:
```
#Enfin, générons le dataframe avec lequel nous ajusterons le modèle, c'est-à-dire
#retirons la catégorie de référence pour chacun des ensembles de variables fictives.
```

In [20]:
```
ind_train = x_train_model.drop(ref, axis = 1) #on retire du df x_train les variables references du df ref
```

In [21]:
```
ind_train.shape
```

Out[21]: (419656, 76)

In [22]:
```
ind_train.head()
```

Out[22]:

	Home_MORTGAGE	Home_OWN	Adresse_FL	Adresse_AL_HI_MO_NM	Adresse_CA	Adresse_NC_ID_NJ	Adresse_NY	Adresse_KY_LA
344531	1	0	0	0	0	0	0	
328300	1	0	0	0	0	0	0	
299890	1	0	0	0	0	0	0	
439226	1	0	0	0	0	0	0	
167889	1	0	0	0	0	0	0	

5 rows × 76 columns

Estimation du Modèle de Probabilité de Défaut

In [23]:
```
# Importons le modèle de régression logistique.
```

In [24]:
```
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
```

In [25]:
```
# Créons un objet de régression logistique à partir de la classe LogisticRegression.
```

In [24]:
```
#regresion = LogisticRegression()  # va contenir l'intercept et les coefficients de la régression
```

In [26]:
```
regresion = LogisticRegression(solver='newton-cg')  # solver='newton-cg' hace que nuestro modelo converga, evit
```

In [27]:
```
# Voyons tous les résultats (coefficients)
```

In [28]:
```
pd.options.display.max_rows = None
```

In [29]:
```
# Nous estimons le modèle en appliquant la méthode fit à l'objet de régression avec les variables
# indépendantes et dépendantes en tant que paramètres
# fit estime le modèle et sauvegarde les résultats dans l'objet de régression. Pour convertir de df à array .va
# regresion.fit(variables_independientes_entrenamiento, variable_dependiente_entrenamiento.values.ravel())
```

In [30]:
```
regresion.fit(ind_train, y_train.values.ravel())
```

Out[30]:
```
LogisticRegression
LogisticRegression(solver='newton-cg')
```

In [31]:
```
# Pour obtenir l'intercept de la régression
```

In [32]:
```
regresion.intercept_
```

Out[32]:
```
array([-0.76185989])
```

In [33]:
```
# Le méthode .coef_ nous donne les coefficients dans l'ordre du dataframe des variables indépendantes.
```

In [34]:
```
regresion.coef_
```

Out[34]:
```
array([[ 0.09928318,  0.08326017, -0.20774375, -0.15624847, -0.13755956,
        -0.14041358, -0.11685257, -0.12350589, -0.07955274, -0.07562235,
         0.04617939,  0.05656821,  0.28085473,  0.09547648, -0.00518066,
         0.39002955,  0.45597356,  0.40049189,  0.53159442,  2.00789053,
         1.32357042,  0.87675559,  0.55347262,  0.2592499 ,  0.06358482,
         0.03483883,  0.05445688,  0.03783859,  0.02564303,  0.00273019,
         0.09790182,  1.81366324,  1.48687054,  1.30527661,  0.99330569,
         0.69440974,  0.39447583,  0.07264291,  0.23205195, -0.11119003,
        -0.05099183,  0.02022245,  0.12720578,  0.16768695,  0.27377485,
         0.33555075,  0.40002367,  0.48026373,  0.54870401,  0.56478929,
         0.60314653,  0.30560026,  0.29584531,  0.38964762,  0.40678849,
         0.35039215,  0.08964278,  0.08464655, -0.0629663 , -0.12549971,
        -0.13010805,  0.26215739,  0.30131099,  0.2269338 ,  0.18768667,
         0.1831951 ,  0.13675437,  0.10600716,  0.09459712,  0.03877119,
        -0.09086395,  0.19210038,  0.31818031,  0.31147842,  0.38315392,
         0.11603228]])
```

creation d'un tableau pour presenter les resultats

In [35]:
```
# Nous créons un format standard similaire à ce que nous obtiendrions avec un logiciel d'économétrie tel que SP
```

```
# Nous créons un tableau avec les noms des variables indépendantes
```

In [36]:
```
nom_var_ind = ind_train.columns.values
nom_var_ind
```

Out[36]:
```
array(['Home_MORTGAGE', 'Home_OWN', 'Adresse_FL', 'Adresse_AL_HI_MO_NM',
       'Adresse_CA', 'Adresse_NC_ID_NJ', 'Adresse_NY', 'Adresse_KY_LA_MD',
       'Adresse_MI_AR_AZ_VA_OK_DE_OH', 'Adresse_MN_PA_UT_MA_RI_WA_TN_IN',
       'Adresse_CT_IL', 'Adresse_TX', 'Adresse_NH_AK_MT_MS_WY_WV_DC_ME',
       'Verification_Not Verified', 'Verification_Source Verified',
       'purpose_house_other_wedding_medical_vacation',
       'Purpose_major_purchase_improvement_car',
       'Purpose_debt_consolidation', 'Purpose_credit_card', 'Grade_A',
       'Grade_B', 'Grade_C', 'Grade_D', 'Grade_E',
       'initial_list_status_w', 'echeance_36', 'ancianite_1a4',
       'ancianite_5a6', 'ancianite_7', 'ancianite_8a9', 'ancianite_10+',
       'nb_mois_1er_credit_<87', 'nb_mois_1er_credit_87a89',
       'nb_mois_1er_credit_89a90', 'nb_mois_1er_credit_90a98',
       'nb_mois_1er_credit_98a101', 'nb_mois_1er_credit_101a110',
       'nb_mois_1er_credit_110a126', 'nb_mois_1er_credit_126a155',
       'Revenu_20K-30K', 'Revenu_30K-40K', 'Revenu_40K-50K',
       'Revenu_50K-60K', 'Revenu_60K-70K', 'Revenu_70K-80K',
       'Revenu_80K-90K', 'Revenu_90K-100K', 'Revenu_100K-126K',
       'Revenu_126K-152K', 'Revenu_152K-227K', 'Revenu_>227K',
       'mths_since_last_delinq_null', 'mths_since_last_delinq_0-4',
       'mths_since_last_delinq_4-30', 'mths_since_last_delinq_30-60',
       'mths_since_last_delinq_60-83', 'impaye_2ans_0', 'impaye_2ans_1-4',
       'total_acc_6-22', 'total_acc_22-50', 'total_acc_>50', 'dti<=3.2',
       'dti_3.2-8.8', 'dti_8.8-10.4', 'dti_10.4-13.6', 'dti_13.6-16.0',
       'dti_16.0-16.7', 'dti_16.7-19.9', 'dti_19.9-20.8', 'dti_20.8-23.2',
       'dti>35.2', 'mths_since_last_record_null',
       'mths_since_last_record_3-21', 'mths_since_last_record_21-31',
       'mths_since_last_record_31-85', 'mths_since_last_record>85'],
      dtype=object)
```

In [37]:
```
# Tableau récapitulatif avec les résultats de notre régression logistique
# Nous créons une colonne avec les noms des variables indépendantes
```

In [38]:
```
tableau = pd.DataFrame(columns = ['Variable Independante'], data = nom_var_ind)
```

In [39]:
```
# Deuxième colonne avec les coefficients pour chaque variable indépendante
# Transpose pour transposer (verticalement) les coefficients
```

In [40]:
```
tableau['Coeficientes'] = np.transpose(regresion.coef_)
```

In [41]:
```
# Nous plaçons l'intercept dans la première ligne de notre tableau
```

In [42]:
```
tableau.index = tableau.index + 1
tableau
```

Out[42]:

	Variable Independante	Coeficientes
1	Home_MORTGAGE	0.099283
2	Home_OWN	0.083260
3	Adresse_FL	-0.207744
4	Adresse_AL_HI_MO_NM	-0.156248
5	Adresse_CA	-0.137560
6	Adresse_NC_ID_NJ	-0.140414
7	Adresse_NY	-0.116853
8	Adresse_KY_LA_MD	-0.123506
9	Adresse_MI_AR_AZ_VA_OK_DE_OH	-0.079553
10	Adresse_MN_PA_UT_MA_RI_WA_TN_IN	-0.075622
11	Adresse_CT_IL	0.046179
12	Adresse_TX	0.056568
13	Adresse_NH_AK_MT_MS_WY_WV_DC_ME	0.280855
14	Verification_Not Verified	0.095476
15	Verification_Source Verified	-0.005181
16	purpose_house_other_wedding_medical_vacation	0.390030
17	Purpose_major_purchase_improvement_car	0.455974
18	Purpose_debt_consolidation	0.400492
19	Purpose_credit_card	0.531594
20	Grade_A	2.007891
21	Grade_B	1.323570
22	Grade_C	0.876756

23	Grade_D	0.553473
24	Grade_E	0.259250
25	initial_list_status_w	0.063585
26	echeance_36	0.034839
27	ancianite_1a4	0.054457
28	ancianite_5a6	0.037839
29	ancianite_7	0.025643
30	ancianite_8a9	0.002730
31	ancianite_10+	0.097902
32	nb_mois_1er_credit_<87	1.813663
33	nb_mois_1er_credit_87a89	1.486871
34	nb_mois_1er_credit_89a90	1.305277
35	nb_mois_1er_credit_90a98	0.993306
36	nb_mois_1er_credit_98a101	0.694410
37	nb_mois_1er_credit_101a110	0.394476
38	nb_mois_1er_credit_110a126	0.072643
39	nb_mois_1er_credit_126a155	0.232052
40	Revenu_20K-30K	-0.111190
41	Revenu_30K-40K	-0.050992
42	Revenu_40K-50K	0.020222
43	Revenu_50K-60K	0.127206
44	Revenu_60K-70K	0.167687
45	Revenu_70K-80K	0.273775
46	Revenu_80K-90K	0.335551
47	Revenu_90K-100K	0.400024
48	Revenu_100K-126K	0.480264
49	Revenu_126K-152K	0.548704
50	Revenu_152K-227K	0.564789
51	Revenu_>227K	0.603147
52	mths_since_last_delinq_null	0.305600
53	mths_since_last_delinq_0-4	0.295845
54	mths_since_last_delinq_4-30	0.389648
55	mths_since_last_delinq_30-60	0.406788
56	mths_since_last_delinq_60-83	0.350392
57	impaye_2ans_0	0.089643
58	impaye_2ans_1-4	0.084647
59	total_acc_6-22	-0.062966
60	total_acc_22-50	-0.125500
61	total_acc_>50	-0.130108
62	dti<=3.2	0.262157
63	dti_3.2-8.8	0.301311
64	dti_8.8-10.4	0.226934
65	dti_10.4-13.6	0.187687
66	dti_13.6-16.0	0.183195
67	dti_16.0-16.7	0.136754
68	dti_16.7-19.9	0.106007
69	dti_19.9-20.8	0.094597
70	dti_20.8-23.2	0.038771
71	dti>35.2	-0.090864
72	mths_since_last_record_null	0.192100
73	mths_since_last_record_3-21	0.318180
74	mths_since_last_record_21-31	0.311478
75	mths_since_last_record_31-85	0.383154
76	mths_since_last_record>85	0.116032

In [43]:
```
# Maintenant, remplissons la ligne 1 avec le nom de l'intercept et sa valeur respective (qui est restée à la fi
```

In [44]:
```
tableau.loc[0] = ['Intercepto', regresion.intercept_[0]]
tableau
```

Out[44]:

	Variable Independante	Coeficientes
1	Home_MORTGAGE	0.099283
2	Home_OWN	0.083260
3	Adresse_FL	-0.207744
4	Adresse_AL_HI_MO_NM	-0.156248
5	Adresse_CA	-0.137560
6	Adresse_NC_ID_NJ	-0.140414
7	Adresse_NY	-0.116853
8	Adresse_KY_LA_MD	-0.123506
9	Adresse_MI_AR_AZ_VA_OK_DE_OH	-0.079553
10	Adresse_MN_PA_UT_MA_RI_WA_TN_IN	-0.075622
11	Adresse_CT_IL	0.046179
12	Adresse_TX	0.056568
13	Adresse_NH_AK_MT_MS_WY_WV_DC_ME	0.280855
14	Verification_Not Verified	0.095476
15	Verification_Source Verified	-0.005181
16	purpose_house_other_wedding_medical_vacation	0.390030
17	Purpose_major_purchase_improvement_car	0.455974
18	Purpose_debt_consolidation	0.400492
19	Purpose_credit_card	0.531594
20	Grade_A	2.007891
21	Grade_B	1.323570
22	Grade_C	0.876756
23	Grade_D	0.553473
24	Grade_E	0.259250
25	initial_list_status_w	0.063585
26	echeance_36	0.034839
27	ancianite_1a4	0.054457
28	ancianite_5a6	0.037839
29	ancianite_7	0.025643
30	ancianite_8a9	0.002730
31	ancianite_10+	0.097902
32	nb_mois_1er_credit_<87	1.813663
33	nb_mois_1er_credit_87a89	1.486871
34	nb_mois_1er_credit_89a90	1.305277
35	nb_mois_1er_credit_90a98	0.993306
36	nb_mois_1er_credit_98a101	0.694410
37	nb_mois_1er_credit_101a110	0.394476
38	nb_mois_1er_credit_110a126	0.072643
39	nb_mois_1er_credit_126a155	0.232052
40	Revenu_20K-30K	-0.111190
41	Revenu_30K-40K	-0.050992
42	Revenu_40K-50K	0.020222
43	Revenu_50K-60K	0.127206
44	Revenu_60K-70K	0.167687
45	Revenu_70K-80K	0.273775
46	Revenu_80K-90K	0.335551
47	Revenu_90K-100K	0.400024
48	Revenu_100K-126K	0.480264
49	Revenu_126K-152K	0.548704
50	Revenu_152K-227K	0.564789

51	Revenu_>227K	0.603147
52	mths_since_last_delinq_null	0.305600
53	mths_since_last_delinq_0-4	0.295845
54	mths_since_last_delinq_4-30	0.389648
55	mths_since_last_delinq_30-60	0.406788
56	mths_since_last_delinq_60-83	0.350392
57	impaye_2ans_0	0.089643
58	impaye_2ans_1-4	0.084647
59	total_acc_6-22	-0.062966
60	total_acc_22-50	-0.125500
61	total_acc_>50	-0.130108
62	dti<=3.2	0.262157
63	dti_3.2-8.8	0.301311
64	dti_8.8-10.4	0.226934
65	dti_10.4-13.6	0.187687
66	dti_13.6-16.0	0.183195
67	dti_16.0-16.7	0.136754
68	dti_16.7-19.9	0.106007
69	dti_19.9-20.8	0.094597
70	dti_20.8-23.2	0.038771
71	dti>35.2	-0.090864
72	mths_since_last_record_null	0.192100
73	mths_since_last_record_3-21	0.318180
74	mths_since_last_record_21-31	0.311478
75	mths_since_last_record_31-85	0.383154
76	mths_since_last_record>85	0.116032
0	Intercepto	-0.761860

In [45]:
```
# Trier le tableau par numéro d'index
```

In [46]:
```
tableau = tableau.sort_index()
tableau
```

Out[46]:

	Variable Independante	Coeficientes
0	Intercepto	-0.761860
1	Home_MORTGAGE	0.099283
2	Home_OWN	0.083260
3	Adresse_FL	-0.207744
4	Adresse_AL_HI_MO_NM	-0.156248
5	Adresse_CA	-0.137560
6	Adresse_NC_ID_NJ	-0.140414
7	Adresse_NY	-0.116853
8	Adresse_KY_LA_MD	-0.123506
9	Adresse_MI_AR_AZ_VA_OK_DE_OH	-0.079553
10	Adresse_MN_PA_UT_MA_RI_WA_TN_IN	-0.075622
11	Adresse_CT_IL	0.046179
12	Adresse_TX	0.056568
13	Adresse_NH_AK_MT_MS_WY_WV_DC_ME	0.280855
14	Verification_Not Verified	0.095476
15	Verification_Source Verified	-0.005181
16	purpose_house_other_wedding_medical_vacation	0.390030
17	Purpose_major_purchase_improvement_car	0.455974
18	Purpose_debt_consolidation	0.400492
19	Purpose_credit_card	0.531594
20	Grade_A	2.007891
21	Grade_B	1.323570
22	Grade_C	0.876756

23	Grade_D	0.553473
24	Grade_E	0.259250
25	initial_list_status_w	0.063585
26	echeance_36	0.034839
27	ancianite_1a4	0.054457
28	ancianite_5a6	0.037839
29	ancianite_7	0.025643
30	ancianite_8a9	0.002730
31	ancianite_10+	0.097902
32	nb_mois_1er_credit_<87	1.813663
33	nb_mois_1er_credit_87a89	1.486871
34	nb_mois_1er_credit_89a90	1.305277
35	nb_mois_1er_credit_90a98	0.993306
36	nb_mois_1er_credit_98a101	0.694410
37	nb_mois_1er_credit_101a110	0.394476
38	nb_mois_1er_credit_110a126	0.072643
39	nb_mois_1er_credit_126a155	0.232052
40	Revenu_20K-30K	-0.111190
41	Revenu_30K-40K	-0.050992
42	Revenu_40K-50K	0.020222
43	Revenu_50K-60K	0.127206
44	Revenu_60K-70K	0.167687
45	Revenu_70K-80K	0.273775
46	Revenu_80K-90K	0.335551
47	Revenu_90K-100K	0.400024
48	Revenu_100K-126K	0.480264
49	Revenu_126K-152K	0.548704
50	Revenu_152K-227K	0.564789
51	Revenu_>227K	0.603147
52	mths_since_last_delinq_null	0.305600
53	mths_since_last_delinq_0-4	0.295845
54	mths_since_last_delinq_4-30	0.389648
55	mths_since_last_delinq_30-60	0.406788
56	mths_since_last_delinq_60-83	0.350392
57	impaye_2ans_0	0.089643
58	impaye_2ans_1-4	0.084647
59	total_acc_6-22	-0.062966
60	total_acc_22-50	-0.125500
61	total_acc_>50	-0.130108
62	dti<=3.2	0.262157
63	dti_3.2-8.8	0.301311
64	dti_8.8-10.4	0.226934
65	dti_10.4-13.6	0.187687
66	dti_13.6-16.0	0.183195
67	dti_16.0-16.7	0.136754
68	dti_16.7-19.9	0.106007
69	dti_19.9-20.8	0.094597
70	dti_20.8-23.2	0.038771
71	dti>35.2	-0.090864
72	mths_since_last_record_null	0.192100
73	mths_since_last_record_3-21	0.318180
74	mths_since_last_record_21-31	0.311478
75	mths_since_last_record_31-85	0.383154
76	mths_since_last_record>85	0.116032

P-values pour la Regression Logistique

In [47]:
```
# Les P-values permettent de visualiser les variables statistiquement significatives
# Les régressions logistiques en Python est qu'elles n'estiment pas les
# valeurs p multivariable de manière directe. Cependant, le code suivant a déjà résolu ce problème :
# https://gist.github.com/rspeare/77061e6e317896be29c6de9a85db301d
```

In [48]:
```
from sklearn import linear_model
import numpy as np
import scipy.stats as stat

class RegresionLogistica_con_p_values:
    """
    Wrapper Class for Logistic Regression which has the usual sklearn instance
    in an attribute self.model, and pvalues, z scores and estimated
    errors for each coefficient in

    self.z_scores
    self.p_values
    self.sigma_estimates

    as well as the negative hessian of the log Likelihood (Fisher information)

    self.F_ij
    """

    def __init__(self,*args,**kwargs):#,**kwargs):
        self.model = linear_model.LogisticRegression(*args,**kwargs, solver = 'newton-cg')#,**args)

    def fit(self,X,y):
        self.model.fit(X,y)
        #### Get p-values for the fitted model ####
        denom = (2.0*(1.0+np.cosh(self.model.decision_function(X))))
        denom = np.tile(denom,(X.shape[1],1)).T
        F_ij = np.dot((X/denom).T,X) ## Fisher Information Matrix
        Cramer_Rao = np.linalg.inv(F_ij) ## Inverse Information Matrix
        sigma_estimates = np.sqrt(np.diagonal(Cramer_Rao))
        z_scores = self.model.coef_[0]/sigma_estimates # z-score for eaach model coefficient
        p_values = [stat.norm.sf(abs(x))*2 for x in z_scores] ### two tailed test for p-values

        self.z_scores = z_scores
        self.p_values = p_values
        self.sigma_estimates = sigma_estimates
        self.F_ij = F_ij
```

In [49]:
```
# Nous allons créer un objet de la classe RegresionLogistica_con_p_values
```

In [50]:
```
reg_log_p_values = RegresionLogistica_con_p_values()
```

In [51]:
```
# Une exigence de la fonction que nous avons définie est que la variable dépendante soit un tableau
# au lieu d'un dataframe, ce n'est pas un problème et pour éviter une erreur, nous utilisons .values.ravel()
```

In [52]:
```
reg_log_p_values.fit(ind_train, y_train.values.ravel()) #.ravel pour obtenir une matrice
```

In [53]:
```
ind_train.corr() #matriz de correlation.
```

Out[53]:

	Home_MORTGAGE	Home_OWN	Adresse_FL	Adresse_AL_HI_MO_NM	Adresse_CA	Adresse_
Home_MORTGAGE	1.000000	-0.316638	-0.008751	0.037809	-0.137074	
Home_OWN	-0.316638	1.000000	0.013075	0.011641	-0.041781	
Adresse_FL	-0.008751	0.013075	1.000000	-0.054846	-0.114913	
Adresse_AL_HI_MO_NM	0.037809	0.011641	-0.054846	1.000000	-0.086432	
Adresse_CA	-0.137074	-0.041781	-0.114913	-0.086432	1.000000	
Adresse_NC_ID_NJ	0.000585	0.003938	-0.071681	-0.053915	-0.112962	
Adresse_NY	-0.119382	0.012143	-0.083039	-0.062458	-0.130862	
Adresse_KY_LA_MD	0.028450	0.012426	-0.058295	-0.043847	-0.091868	
Adresse_MI_AR_AZ_VA_OK_DE_OH	0.064833	0.004977	-0.104386	-0.078514	-0.164503	
Adresse_MN_PA_UT_MA_RI_WA_TN_IN	0.040091	-0.008133	-0.107951	-0.081196	-0.170121	
Adresse_CT_IL	0.011089	-0.001181	-0.065426	-0.049210	-0.103105	
Adresse_TX	0.046377	0.025941	-0.078687	-0.059185	-0.124004	
Adresse_NH_AK_MT_MS_WY_WV_DC_ME	0.016282	0.011119	-0.042132	-0.031689	-0.066395	
Verification_Not Verified	-0.064561	-0.004892	0.006709	0.000313	-0.004219	
Verification_Source Verified	-0.020449	0.015197	0.000299	0.001534	-0.004506	

purpose_house_other_wedding_medical_vacation	-0.064431	0.012584	0.005511	-0.008563	0.013111
Purpose_major_purchase_improvement_car	0.096584	0.040852	0.015992	0.001966	-0.014718
Purpose_debt_consolidation	-0.001388	-0.026810	-0.010970	0.007261	0.002071
Purpose_credit_card	-0.010325	-0.001945	-0.005518	-0.002921	-0.003986
Grade_A	0.065750	-0.006066	-0.005070	-0.006999	0.004983
Grade_B	0.005397	-0.000998	0.000876	-0.002665	0.009454
Grade_C	-0.015595	-0.000058	0.004101	0.001486	-0.005146
Grade_D	-0.035120	0.003808	0.002383	0.005422	-0.004746
Grade_E	-0.015290	0.002609	-0.004614	0.002146	-0.005147
initial_list_status_w	0.025427	0.008708	-0.003385	0.002619	-0.017128
echeance_36	-0.119039	0.017476	0.013456	-0.010034	0.041205
ancianite_1a4	-0.112089	-0.017090	0.007217	-0.008705	0.001452
ancianite_5a6	-0.016578	-0.003240	0.005902	-0.000624	0.008068
ancianite_7	0.009942	-0.005162	0.001979	-0.001953	0.006696
ancianite_8a9	0.017656	-0.007209	0.002409	0.001048	0.005288
ancianite_10+	0.163750	0.009722	-0.024341	0.014176	-0.018117
nb_mois_1er_credit_<87	-0.005991	0.016466	0.003242	0.001135	-0.010343
nb_mois_1er_credit_87a89	-0.007469	0.013804	0.000302	0.001062	-0.009174
nb_mois_1er_credit_89a90	0.000408	0.007518	0.001211	-0.001224	-0.009917
nb_mois_1er_credit_90a98	0.018624	0.009755	-0.008345	0.001902	-0.020153
nb_mois_1er_credit_98a101	0.020658	0.002901	-0.001964	-0.001000	-0.000883
nb_mois_1er_credit_101a110	0.025320	-0.020555	-0.005034	-0.000642	0.016624
nb_mois_1er_credit_110a126	-0.037509	-0.013676	0.012537	-0.000966	0.021615
nb_mois_1er_credit_126a155	-0.030498	-0.010362	0.003658	-0.001895	0.014906
Revenu_20K-30K	-0.137299	0.055193	0.023553	0.008513	-0.002136
Revenu_30K-40K	-0.140818	0.041732	0.025055	0.009752	-0.012501
Revenu_40K-50K	-0.091032	0.016352	0.014673	0.010012	-0.018244
Revenu_50K-60K	-0.024786	-0.000969	-0.000196	0.006048	-0.015487
Revenu_60K-70K	0.020779	-0.014551	-0.003305	0.000102	-0.004041
Revenu_70K-80K	0.049357	-0.018877	-0.007833	-0.001307	0.000216
Revenu_80K-90K	0.061510	-0.022065	-0.011271	-0.001940	0.006762
Revenu_90K-100K	0.071104	-0.021625	-0.012720	-0.004218	0.004177
Revenu_100K-126K	0.107981	-0.026145	-0.017018	-0.011597	0.022703
Revenu_126K-152K	0.077084	-0.014392	-0.010429	-0.012200	0.016945
Revenu_152K-227K	0.076136	-0.011659	-0.008332	-0.012380	0.017133
Revenu_>227K	0.044911	-0.000052	0.000232	-0.009373	0.006419
mths_since_last_delinq_null	-0.057444	0.001280	0.000100	-0.002349	0.004491
mths_since_last_delinq_0-4	0.025539	0.000693	-0.001312	-0.001935	-0.010085
mths_since_last_delinq_4-30	0.052512	-0.003739	0.000748	-0.000446	-0.003962
mths_since_last_delinq_30-60	0.008362	0.001409	0.001707	0.000081	0.003431
mths_since_last_delinq_60-83	0.004609	0.000926	-0.003197	0.005877	-0.002580
impaye_2ans_0	-0.061007	0.002624	-0.000042	0.002217	0.007726
impaye_2ans_1-4	0.059253	-0.002250	0.000801	-0.002345	-0.006372
total_acc_6-22	-0.183978	0.019332	0.018193	-0.014785	0.050656
total_acc_22-50	0.188546	-0.021740	-0.019114	0.012698	-0.045717
total_acc_>50	0.057774	0.000639	-0.004610	0.009546	-0.028662
dti<=3.2	-0.014790	0.006629	-0.000813	-0.007656	0.020351
dti_3.2-8.8	-0.013571	-0.005339	-0.005417	-0.019913	0.043960
dti_8.8-10.4	0.002136	-0.004528	0.000598	-0.010345	0.018047
dti_10.4-13.6	0.010869	-0.011850	-0.005993	-0.008838	0.016317
dti_13.6-16.0	0.010815	-0.013317	-0.002072	-0.006552	0.006987
dti_16.0-16.7	0.006780	-0.006030	0.000142	-0.000812	-0.001196
dti_16.7-19.9	0.014412	-0.006422	0.001612	0.003016	-0.008250
dti_19.9-20.8	0.004283	0.000150	0.002341	0.003349	-0.009245
dti_20.8-23.2	-0.001197	0.002826	0.002744	0.008487	-0.016621

dti>35.2	-0.013931	0.018359	0.001770	0.008183	-0.012144
mths_since_last_record_null	0.003207	-0.003668	0.009441	-0.004468	0.016581
mths_since_last_record_3-21	0.005091	0.004111	-0.007535	-0.003477	-0.005972
mths_since_last_record_21-31	-0.000796	0.002372	-0.009379	-0.000967	-0.002496
mths_since_last_record_31-85	-0.014011	0.003332	-0.002361	-0.000975	0.002819
mths_since_last_record>85	0.012344	-0.000241	-0.007732	0.008619	-0.023724

76 rows × 76 columns

In [86]:
```
# Une fois que nous avons généré la régression logistique, nous pouvons ajouter les valeurs p à notre tableau d
# Voyons les valeurs p
```

In [54]:
```
p_values = reg_log_p_values.p_values #p-values de las variables
p_values
```

Out[54]:
```
[4.77282793842003e-15,
 3.9893984650824146e-05,
 3.2283190790214957e-16,
 3.662117494573817e-07,
 1.4530342983220621e-10,
 9.72246210126704e-08,
 2.0393338085599344e-06,
 3.99350799188039e-05,
 0.0003239710901581531,
 0.0005777907268581233,
 0.1162135314325683,
 0.03205995962146847,
 1.8453188193492042e-10,
 5.168320947089321e-11,
 0.7035361524940518,
 2.1272965033491847e-33,
 5.4704589244364896e-42,
 2.166857395310302e-44,
 1.413566015323589e-66,
 0.0,
 0.0,
 1.485550240838152e-290,
 2.3401599396957834e-116,
 3.2528123282229273e-24,
 2.5169530532279 81e-06,
 0.012846157982132,
 0.0003132900482188978,
 0.09691919099228748,
 0.30060864713422,
 0.9186651593264036,
 1.5215246731125824e-11,
 3.439488022764563e-148,
 2.8932473423735515e-110,
 3.7296502755453023e-76,
 2.330429540602936e-55,
 7.073298157013933e-27,
 4.034352939301394e-10,
 0.2522739352494888,
 0.0003271199148322525,
 0.00403158509840865,
 0.14929488136683317,
 0.5615543666617577,
 0.0003135179764434612,
 2.8063474150813904e-06,
 1.4725864928978926e-13,
 4.498613734143315e-18,
 1.77164652337778e-22,
 1.6814607674459077e-35,
 8.682272401539748e-32,
 1.6766878904336955e-29,
 1.0720419316553086e-17,
 0.0034629090296822056,
 0.003624931881116301,
 0.00011712073947791432,
 0.00011338962577754569,
 0.0009846082601805213,
 0.2089595368846846,
 0.22552971062731209,
 0.09474379421666261,
 0.0013142639403616626,
 0.010773330744870665,
 2.922375995879908e-11,
 5.221812202085968e-46,
 7.215171770879303e-17,
 5.524567360235546e-22,
 2.3025973746546147e-19,
 2.8077730718240692e-05,
 3.5765071506800306e-09,
 0.0011702765641137829,
 0.053448799254099115,
 0.19290054952091384,
 0.039570481933775825,
 0.015016908797481238,
 0.014536089712109821,
 7.394165403013126e-05,
 0.2253321446708081]
```

In [55]:
```
# L'intercept n'a jamais une valeur p, mais nous avons besoin d'une valeur NaN pour avoir une colonne de la mêm
```

In [56]:
```
p_values = np.append(np.nan, np.array(p_values))
```

In [57]:
```
p_values
```

Out[57]:
```
array([           nan, 4.77282794e-015, 3.98939847e-005, 3.22831908e-016,
       3.66211749e-007, 1.45303430e-010, 9.72246210e-008, 2.03933381e-006,
       3.99350799e-005, 3.23971090e-004, 5.77790727e-004, 1.16213531e-001,
       3.20599596e-002, 1.84531882e-010, 5.16832095e-011, 7.03536152e-001,
       2.12729650e-033, 5.47045892e-042, 2.16685740e-044, 1.41356602e-066,
       0.00000000e+000, 0.00000000e+000, 1.48555024e-290, 2.34015994e-116,
       3.25281233e-024, 2.51695305e-006, 1.28461580e-002, 3.13290048e-004,
       9.69191910e-002, 3.00608647e-001, 9.18665159e-001, 1.52152467e-011,
       3.43948802e-148, 2.89324734e-110, 3.72965028e-076, 2.33042954e-055,
       7.07329816e-027, 4.03435294e-010, 2.52273935e-001, 3.27119915e-004,
       4.03158510e-003, 1.49294881e-001, 5.61554367e-001, 3.13517976e-004,
       2.80634742e-006, 1.47258649e-013, 4.49861373e-018, 1.77164652e-022,
       1.68146077e-035, 8.68227240e-032, 1.67668789e-029, 1.07204193e-017,
       3.46290903e-003, 3.62493188e-003, 1.17120739e-004, 1.13389626e-004,
       9.84608260e-004, 2.08959537e-001, 2.25529711e-001, 9.47437942e-002,
       1.31426394e-003, 1.07733307e-002, 2.92237600e-011, 5.22181220e-046,
       7.21517177e-017, 5.52456736e-022, 2.30259737e-019, 2.80777307e-005,
       3.57650715e-009, 1.17027656e-003, 5.34487993e-002, 1.92900550e-001,
       3.95704819e-002, 1.50169088e-002, 1.45360897e-002, 7.39416540e-005,
       2.25332145e-001])
```

In [60]:
```
tableau['p_values'] = p_values
```

In [61]:
```
tableau
#il va mieux voir sur excel
```

Out[61]:

	Variable Independante	Coeficientes	p_values
0	Intercepto	-0.761860	NaN
1	Home_MORTGAGE	0.099283	4.772828e-15
2	Home_OWN	0.083260	3.989398e-05
3	Adresse_FL	-0.207744	3.228319e-16
4	Adresse_AL_HI_MO_NM	-0.156248	3.662117e-07
5	Adresse_CA	-0.137560	1.453034e-10
6	Adresse_NC_ID_NJ	-0.140414	9.722462e-08
7	Adresse_NY	-0.116853	2.039334e-06
8	Adresse_KY_LA_MD	-0.123506	3.993508e-05
9	Adresse_MI_AR_AZ_VA_OK_DE_OH	-0.079553	3.239711e-04
10	Adresse_MN_PA_UT_MA_RI_WA_TN_IN	-0.075622	5.777907e-04
11	Adresse_CT_IL	0.046179	1.162135e-01
12	Adresse_TX	0.056568	3.205996e-02
13	Adresse_NH_AK_MT_MS_WY_WV_DC_ME	0.280855	1.845319e-10
14	Verification_Not Verified	0.095476	5.168321e-11
15	Verification_Source Verified	-0.005181	7.035362e-01
16	purpose_house_other_wedding_medical_vacation	0.390030	2.127297e-33
17	Purpose_major_purchase_improvement_car	0.455974	5.470459e-42
18	Purpose_debt_consolidation	0.400492	2.166857e-44
19	Purpose_credit_card	0.531594	1.413566e-66
20	Grade_A	2.007891	0.000000e+00
21	Grade_B	1.323570	0.000000e+00
22	Grade_C	0.876756	1.485550e-290
23	Grade_D	0.553473	2.340160e-116
24	Grade_E	0.259250	3.252812e-24
25	initial_list_status_w	0.063585	2.516953e-06
26	echeance_36	0.034839	1.284616e-02
27	ancianite_1a4	0.054457	3.132900e-04
28	ancianite_5a6	0.037839	9.691919e-02
29	ancianite_7	0.025643	3.006086e-01
30	ancianite_8a9	0.002730	9.186652e-01
31	ancianite_10+	0.097902	1.521525e-11
32	nb_mois_1er_credit_<87	1.813663	3.439488e-148
33	nb_mois_1er_credit_87a89	1.486871	2.893247e-110
34	nb_mois_1er_credit_89a90	1.305277	3.729650e-76
35	nb_mois_1er_credit_90a98	0.993306	2.330430e-55
36	nb_mois_1er_credit_98a101	0.694410	7.073298e-27
37	nb_mois_1er_credit_101a110	0.394476	4.034353e-10

38	nb_mois_1er_credit_110a126	0.072643	2.522739e-01
39	nb_mois_1er_credit_126a155	0.232052	3.271199e-04
40	Revenu_20K-30K	-0.111190	4.031585e-03
41	Revenu_30K-40K	-0.050992	1.492949e-01
42	Revenu_40K-50K	0.020222	5.615544e-01
43	Revenu_50K-60K	0.127206	3.135180e-04
44	Revenu_60K-70K	0.167687	2.806347e-06
45	Revenu_70K-80K	0.273775	1.472586e-13
46	Revenu_80K-90K	0.335551	4.498614e-18
47	Revenu_90K-100K	0.400024	1.771647e-22
48	Revenu_100K-126K	0.480264	1.681461e-35
49	Revenu_126K-152K	0.548704	8.682272e-32
50	Revenu_152K-227K	0.564789	1.676688e-29
51	Revenu_>227K	0.603147	1.072042e-17
52	mths_since_last_delinq_null	0.305600	3.462909e-03
53	mths_since_last_delinq_0-4	0.295845	3.624932e-03
54	mths_since_last_delinq_4-30	0.389648	1.171207e-04
55	mths_since_last_delinq_30-60	0.406788	1.133896e-04
56	mths_since_last_delinq_60-83	0.350392	9.846083e-04
57	impaye_2ans_0	0.089643	2.089595e-01
58	impaye_2ans_1-4	0.084647	2.255297e-01
59	total_acc_6-22	-0.062966	9.474379e-02
60	total_acc_22-50	-0.125500	1.314264e-03
61	total_acc_>50	-0.130108	1.077333e-02
62	dti<=3.2	0.262157	2.922376e-11
63	dti_3.2-8.8	0.301311	5.221812e-46
64	dti_8.8-10.4	0.226934	7.215172e-17
65	dti_10.4-13.6	0.187687	5.524567e-22
66	dti_13.6-16.0	0.183195	2.302597e-19
67	dti_16.0-16.7	0.136754	2.807773e-05
68	dti_16.7-19.9	0.106007	3.576507e-09
69	dti_19.9-20.8	0.094597	1.170277e-03
70	dti_20.8-23.2	0.038771	5.344880e-02
71	dti>35.2	-0.090864	1.929005e-01
72	mths_since_last_record_null	0.192100	3.957048e-02
73	mths_since_last_record_3-21	0.318180	1.501691e-02
74	mths_since_last_record_21-31	0.311478	1.453609e-02
75	mths_since_last_record_31-85	0.383154	7.394165e-05
76	mths_since_last_record>85	0.116032	2.253321e-01

Sélection de la variable avec les valeurs p

H0 : Non significatif H1 : Significatif

RDD : Si la valeur p < alpha (0,05) : elle est significative et H0 est rejetée

Une petite valeur p signifie que la probabilité que les résultats obtenus soient dus au hasard est faible.

In [95]:
```
# Nous allons retirer du modèle les variables indicatrices liées à la variable d'incidents de retard
# au cours des 2 dernières années, car en tant que membre du même groupe (morosidad_2anios_0, morosidad_2anios_
# aucune n'est statistiquement significative selon sa valeur p.
```

In [96]:
```
# Supprimons les variables indicatrices liées à morosidad_2anios du dataframe des variables indépendantes origi
```

In [64]:
```
x_train_model = x_train.loc[:, ['Home_MORTGAGE',
'Home_RENT_ANY_OTHER_NONE',
'Home_OWN',
'Adresse_ND_NE_IA_NV',
```

```
'Adresse_FL', 'Adresse_AL_HI_MO_NM',
'Adresse_CA', 'Adresse_NC_ID_NJ',
'Adresse_NY',
'Adresse_KY_LA_MD',
'Adresse_MI_AR_AZ_VA_OK_DE_OH',
'Adresse_MN_PA_UT_MA_RI_WA_TN_IN',
'Adresse_CT_IL',
'Adresse_TX',
'Adresse_NH_AK_MT_MS_WY_WV_DC_ME',
'Verification_Not Verified',
'Verification_Source Verified',
'Verification_Verified',
'purpose_ed_pyme_enerren_moving',
'purpose_house_other_wedding_medical_vacation',
'Purpose_major_purchase_improvement_car',
'Purpose_debt_consolidation',
'Purpose_credit_card',
'Grade_A',
'Grade_B',
'Grade_C',
'Grade_D',
'Grade_E',
'Grades_F_G',
'initial_list_status_f',
'initial_list_status_w',
'echeance_36',
'echeance_60',
'ancianite_<_1',
'ancianite_1a4',
'ancianite_5a6',
'ancianite_7',
'ancianite_8a9',
'ancianite_10+',
'nb_mois_1er_credit_<87',
'nb_mois_1er_credit_87a89',
'nb_mois_1er_credit_89a90',
'nb_mois_1er_credit_90a98',
'nb_mois_1er_credit_98a101',
'nb_mois_1er_credit_101a110',
'nb_mois_1er_credit_110a126',
'nb_mois_1er_credit_126a155',
'nb_mois_1er_credit_>155',
'Revenu_<20K',
'Revenu_20K-30K',
'Revenu_30K-40K',
'Revenu_40K-50K',
'Revenu_50K-60K',
'Revenu_60K-70K',
'Revenu_70K-80K',
'Revenu_80K-90K',
'Revenu_90K-100K',
'Revenu_100K-126K',
'Revenu_126K-152K',
'Revenu_152K-227K',
'Revenu_>227K',
'mths_since_last_delinq_null',
'mths_since_last_delinq_0-4',
'mths_since_last_delinq_4-30',
'mths_since_last_delinq_30-60',
'mths_since_last_delinq_60-83',
'mths_since_last_delinq_83+',
#'impaye_2ans_0',
#'impaye_2ans_1-4',
#'impaye_2ans_>=5',
'total_acc_<=6',
'total_acc_6-22',
'total_acc_22-50',
'total_acc_>50',
'dti<=3.2',
'dti_3.2-8.8',
'dti_8.8-10.4',
'dti_10.4-13.6',
'dti_13.6-16.0',
'dti_16.0-16.7',
'dti_16.7-19.9',
'dti_19.9-20.8',
'dti_20.8-23.2',
'dti_23.2-35.2',
'dti>35.2',
'mths_since_last_record_null',
'mths_since_last_record_0-3',
'mths_since_last_record_3-21',
'mths_since_last_record_21-31',
'mths_since_last_record_31-85',
'mths_since_last_record>85']]
```

In [98]:
```
# Faisons de même avec les catégories de référence
```

In [ ]:
```
ref = ['Home_RENT_ANY_OTHER_NONE',
```

In [65]:
```
ref = ['Home_RENT_ANY_OTHER_NONE',
'Adresse_ND_NE_IA_NV',
'Verification_Verified',
'purpose_ed_pyme_enerren_moving',
'Grades_F_G',
'initial_list_status_f',
'echeance_60',
'ancianite_<_1',
'nb_mois_1er_credit_>155',
'Revenu_<20K',
'mths_since_last_delinq_83+',
#'impaye_2ans_>=5',
'total_acc_<=6',
'dti_23.2-35.2',
'mths_since_last_record_0-3']
```

In [100…
```
# Supprimer les variables de référence des variables indépendantes du modèle
```

In [68]:
```
ind_train = x_train_model.drop(ref, axis = 1)
```

In [69]:
```
ind_train.shape
```

Out[69]:
```
(419656, 74)
```

In [70]:
```
# Définissons les objets regresion2 et regresion2_p_values
```

In [71]:
```
regresion2 = regresion = LogisticRegression(solver = 'newton-cg')
regresion2_p_values = RegresionLogistica_con_p_values() # Le solveur est déjà défini dans la classe RegresionLo
```

In [72]:
```
# Ajustons le modèle
```

In [73]:
```
regresion2.fit(ind_train, y_train.values.ravel())
regresion2_p_values.fit(ind_train, y_train.values.ravel())
```

In [106…
```
# tableau
```

In [75]:
```
nom_var_ind = ind_train.columns.values
tableau = pd.DataFrame(columns = ['Variable Independante'], data = nom_var_ind)
tableau['Coeficientes'] = np.transpose(regresion2.coef_)
tableau.index = tableau.index + 1
tableau.loc[0] = ['Intercepto', regresion2.intercept_[0]]
tableau = tableau.sort_index()
p_values = regresion2_p_values.p_values
p_values = np.append(np.nan, np.array(p_values))
tableau['p_values'] = p_values
```

In [76]:
```
tableau
```

Out[76]:

	Variable Independante	Coeficientes	p_values
0	Intercepto	-0.667983	NaN
1	Home_MORTGAGE	0.099254	4.826906e-15
2	Home_OWN	0.083285	3.966229e-05
3	Adresse_FL	-0.207613	3.308666e-16
4	Adresse_AL_HI_MO_NM	-0.156218	3.667267e-07
5	Adresse_CA	-0.137447	1.484883e-10
6	Adresse_NC_ID_NJ	-0.140432	9.642841e-08
7	Adresse_NY	-0.116864	2.028481e-06
8	Adresse_KY_LA_MD	-0.123486	3.996527e-05
9	Adresse_MI_AR_AZ_VA_OK_DE_OH	-0.079535	3.239793e-04
10	Adresse_MN_PA_UT_MA_RI_WA_TN_IN	-0.075609	5.771834e-04
11	Adresse_CT_IL	0.046172	1.162430e-01
12	Adresse_TX	0.056687	3.165762e-02
13	Adresse_NH_AK_MT_MS_WY_WV_DC_ME	0.280750	1.870772e-10
14	Verification_Not Verified	0.095455	5.203965e-11
15	Verification_Source Verified	-0.005208	7.020393e-01
16	purpose_house_other_wedding_medical_vacation	0.389900	2.034026e-33
17	Purpose_major_purchase_improvement_car	0.455846	5.272720e-42
18	Purpose_debt_consolidation	0.400467	1.846630e-44
19	Purpose_credit_card	0.531613	1.113748e-66
20	Grade_A	2.008301	0.000000e+00
21	Grade_B	1.323835	0.000000e+00

22	Grade_C	0.876908	8.340840e-291
23	Grade_D	0.553558	1.856190e-116
24	Grade_E	0.259348	3.012767e-24
25	initial_list_status_w	0.063553	2.545807e-06
26	echeance_36	0.034634	1.338126e-02
27	ancianite_1a4	0.054371	3.200932e-04
28	ancianite_5a6	0.037888	9.647531e-02
29	ancianite_7	0.025677	2.999673e-01
30	ancianite_8a9	0.002770	9.174683e-01
31	ancianite_10+	0.097862	1.549211e-11
32	nb_mois_1er_credit_<87	1.813029	4.383749e-148
33	nb_mois_1er_credit_87a89	1.486218	3.580371e-110
34	nb_mois_1er_credit_89a90	1.304625	4.417178e-76
35	nb_mois_1er_credit_90a98	0.992666	2.705136e-55
36	nb_mois_1er_credit_98a101	0.693929	7.641018e-27
37	nb_mois_1er_credit_101a110	0.394082	4.202029e-10
38	nb_mois_1er_credit_110a126	0.072298	2.545465e-01
39	nb_mois_1er_credit_126a155	0.231783	3.324138e-04
40	Revenu_20K-30K	-0.111294	3.967872e-03
41	Revenu_30K-40K	-0.051059	1.484027e-01
42	Revenu_40K-50K	0.020129	5.630304e-01
43	Revenu_50K-60K	0.127166	3.112796e-04
44	Revenu_60K-70K	0.167666	2.762332e-06
45	Revenu_70K-80K	0.273690	1.439970e-13
46	Revenu_80K-90K	0.335496	4.341269e-18
47	Revenu_90K-100K	0.399993	1.689195e-22
48	Revenu_100K-126K	0.480193	1.566924e-35
49	Revenu_126K-152K	0.548754	8.051498e-32
50	Revenu_152K-227K	0.564586	1.691447e-29
51	Revenu_>227K	0.602758	1.113899e-17
52	mths_since_last_delinq_null	0.305900	2.822434e-04
53	mths_since_last_delinq_0-4	0.289052	6.600199e-04
54	mths_since_last_delinq_4-30	0.384151	6.219555e-06
55	mths_since_last_delinq_30-60	0.407221	1.783825e-06
56	mths_since_last_delinq_60-83	0.350782	4.972709e-05
57	total_acc_6-22	-0.063092	9.368739e-02
58	total_acc_22-50	-0.126080	1.237349e-03
59	total_acc_>50	-0.131218	1.009810e-02
60	dti<=3.2	0.261988	2.977315e-11
61	dti_3.2-8.8	0.301200	5.464292e-46
62	dti_8.8-10.4	0.226738	7.638654e-17
63	dti_10.4-13.6	0.187562	5.852196e-22
64	dti_13.6-16.0	0.183184	2.298727e-19
65	dti_16.0-16.7	0.136840	2.773811e-05
66	dti_16.7-19.9	0.105962	3.623664e-09
67	dti_19.9-20.8	0.094623	1.166047e-03
68	dti_20.8-23.2	0.038756	5.353195e-02
69	dti>35.2	-0.091007	1.921946e-01
70	mths_since_last_record_null	0.188450	4.183036e-02
71	mths_since_last_record_3-21	0.312915	1.638101e-02
72	mths_since_last_record_21-31	0.305707	1.605904e-02
73	mths_since_last_record_31-85	0.379676	7.596651e-05
74	mths_since_last_record>85	0.112414	2.366746e-01

In [ ]:
```
#interpretation: voir l'excel Ejemplo+Interpretación+de+los+Coeficientes
```

Validation du Modèle PD

avec la base test

In [110…]:
```
# Commençons par sélectionner les variables de notre modèle (elles doivent être les mêmes que celles de l'ensem
```

In [77]:
```
x_test_model = x_test.loc[:, ['Home_MORTGAGE',
'Home_RENT_ANY_OTHER_NONE',
'Home_OWN',
'Adresse_ND_NE_IA_NV',
'Adresse_FL', 'Adresse_AL_HI_MO_NM',
'Adresse_CA', 'Adresse_NC_ID_NJ',
'Adresse_NY',
'Adresse_KY_LA_MD',
'Adresse_MI_AR_AZ_VA_OK_DE_OH',
'Adresse_MN_PA_UT_MA_RI_WA_TN_IN',
'Adresse_CT_IL',
'Adresse_TX',
'Adresse_NH_AK_MT_MS_WY_WV_DC_ME',
'Verification_Not Verified',
'Verification_Source Verified',
'Verification_Verified',
'purpose_ed_pyme_enerren_moving',
'purpose_house_other_wedding_medical_vacation',
'Purpose_major_purchase_improvement_car',
'Purpose_debt_consolidation',
'Purpose_credit_card',
'Grade_A',
'Grade_B',
'Grade_C',
'Grade_D',
'Grade_E',
'Grades_F_G',
'initial_list_status_f',
'initial_list_status_w',
'echeance_36',
'echeance_60',
'ancianite_<_1',
'ancianite_1a4',
'ancianite_5a6',
'ancianite_7',
'ancianite_8a9',
'ancianite_10+',
'nb_mois_1er_credit_<87',
'nb_mois_1er_credit_87a89',
'nb_mois_1er_credit_89a90',
'nb_mois_1er_credit_90a98',
'nb_mois_1er_credit_98a101',
'nb_mois_1er_credit_101a110',
'nb_mois_1er_credit_110a126',
'nb_mois_1er_credit_126a155',
'nb_mois_1er_credit_>155',
'Revenu_<20K',
'Revenu_20K-30K',
'Revenu_30K-40K',
'Revenu_40K-50K',
'Revenu_50K-60K',
'Revenu_60K-70K',
'Revenu_70K-80K',
'Revenu_80K-90K',
'Revenu_90K-100K',
'Revenu_100K-126K',
'Revenu_126K-152K',
'Revenu_152K-227K',
'Revenu_>227K',
'mths_since_last_delinq_null',
'mths_since_last_delinq_0-4',
'mths_since_last_delinq_4-30',
'mths_since_last_delinq_30-60',
'mths_since_last_delinq_60-83',
'mths_since_last_delinq_83+',
#'impaye_2ans_0',
#'impaye_2ans_1-4',
#'impaye_2ans_>=5',
'total_acc_<=6',
'total_acc_6-22',
'total_acc_22-50',
'total_acc_>50',
'dti<=3.2',
'dti_3.2-8.8',
'dti_8.8-10.4',
'dti_10.4-13.6',
'dti_13.6-16.0',
'dti_16.0-16.7',
```

```
'dti_16.7-19.9',
'dti_19.9-20.8',
'dti_20.8-23.2',
'dti_23.2-35.2',
'dti>35.2',
'mths_since_last_record_null',
'mths_since_last_record_0-3',
'mths_since_last_record_3-21',
'mths_since_last_record_21-31',
'mths_since_last_record_31-85',
'mths_since_last_record>85']]
```

In [78]:
```
ref_test = ['Home_RENT_ANY_OTHER_NONE',
'Adresse_ND_NE_IA_NV',
'Verification_Verified',
'purpose_ed_pyme_enerren_moving',
'Grades_F_G',
'initial_list_status_f',
'echeance_60',
'ancianite_<_1',
'nb_mois_1er_credit_>155',
'Revenu_<20K',
'mths_since_last_delinq_83+',
#'impaye_2ans_>=5',
'total_acc_<=6',
'dti_23.2-35.2',
'mths_since_last_record_0-3']
```

In [113…
```
# Supprimons les variables de référence de la base de données d'indicateurs de l'évaluation
```

In [79]:
```
ind_test = x_test_model.drop(ref_test, axis = 1)
```

In [80]:
```
ind_test.shape
```

Out[80]:
```
(46629, 74)
```

In [81]:
```
ind_train.shape
```

Out[81]:
```
(419656, 74)
```

In [117…
```
# Nous avons maintenant une base d'évaluation avec les mêmes 74 variables indicatrices que celle de l'ensemble
# Nous pouvons maintenant appliquer le modèle PD que nous avons déjà sur la base d'évaluation
```

In [118…
```
#regresion c'est le modele qu'on a deja fait
prediccion_y = regresion2.predict(ind_test)
```

In [119…
```
# Comment fonctionne le modèle :
# 1. Les valeurs des variables indicatrices (1-0) sont multipliées par leur coefficient respectif (Beta)
# Le résultat est le logarithme des probabilités (odds) d'être un bon client (ne pas faire défaut)
# 2. L'exponentielle (e) du résultat ci-dessus est estimée pour obtenir la probabilité d'être un bon client
# 3. Enfin, un seuil est déterminé pour catégoriser les probabilités d'être bon ou mauvais
# Le seuil par défaut est de 0,5. Mais si les coûts d'être bon ou mauvais sont connus, il est possible de
# déterminer le seuil qui optimise la fonction d'utilité.
```

In [120…
```
prediccion_y
```

Out[120]:
```
array([1, 1, 1, ..., 1, 1, 1], dtype=int64)
```

In [121…
```
# Les prédictions précédentes ont été catégorisées avec un seuil de 0,5.
# Comme nous souhaitons effectuer cette catégorisation, nous avons besoin des probabilités.
```

In [82]:
```
proba_y = regresion2.predict_proba(ind_test)
```

In [83]:
```
proba_y # les probabilités de défault au lieu des catégories (1 et 0)
```

Out[83]:
```
array([[0.09044552, 0.90955448],
       [0.03819745, 0.96180255],
       [0.0632368 , 0.9367632 ],
       ...,
       [0.06364772, 0.93635228],
       [0.11792685, 0.88207315],
       [0.01646695, 0.98353305]])
```

In [125…
```
# Nous avons obtenu un tableau de tableaux. Pour chacune des 46 629 observations, nous avons deux valeurs.
# La probabilité d'être un mauvais client (PD) et la probabilité d'être un bon client (1-PD).
# Générons un tableau avec la probabilité d'être un bon client.
```

In [84]:
```
proba_bon = proba_y[:,1]
```

In [85]:
```
proba_bon
```

Out[85]:
```
array([0.90955448, 0.96180255, 0.9367632 , ..., 0.93635228, 0.88207315,
       0.98353305])
```

In [86]:
```
# Maintenant, générons un df avec deux colonnes :
# 1. Avec les valeurs réelles de bon-mauvais de la base d'évaluation
# 2. Avec les probabilités que nous venons d'estimer
# Nous ne pouvons pas simplement concaténer un df avec un tableau ayant des indices différents.
# Nous devons d'abord extraire les valeurs réelles et supprimer les indices.
```

In [87]:
```
valeur_reel= y_test
```

In [88]:
```
valeur_reel.head()
```

Out[88]:

	Bonus_malus
89223	1
158835	0
108909	1
270155	1
23971	1

In [89]:
```
valeur_reel.reset_index(drop = True, inplace = True)
```

In [90]:
```
valeur_reel.head()
```

Out[90]:

	Bonus_malus
0	1
1	0
2	1
3	1
4	1

In [91]:
```
# concat
```

In [92]:
```
df_predict = pd.concat([valeur_reel, pd.DataFrame(proba_bon)], axis = 1)
```

In [93]:
```
df_predict.head()
```

Out[93]:

	Bonus_malus	0
0	1	0.909554
1	0	0.961803
2	1	0.936763
3	1	0.886823
4	1	0.854169

In [94]:
```
df_predict.columns = ['Real', 'Proba_Preditcion']
```

In [97]:
```
df_predict.head()
```

Out[97]:

	Real	Proba_Preditcion
0	1	0.909554
1	0	0.961803
2	1	0.936763
3	1	0.886823
4	1	0.854169

In [139…
```
# Mettons les indices de la base d'évaluation pour savoir qui est chaque client
```

In [98]:
```
df_predict.index = ind_test.index
```

In [99]:
```
df_predict.head()
```

Out[99]:

	Real	Proba_Preditcion
89223	1	0.909554
158835	0	0.961803
108909	1	0.936763
270155	1	0.886823
23971	1	0.854169

Matrice de confusion

In [205…
```
# Définissons la prédiction à partir des probabilités et d'un seuil.
```

In [103…
```
seuil = 0.5
```

In [104…
```
df_predict['Prediction'] = np.where(df_predict['Proba_Preditcion'] > seuil, 1, 0)
```

In [105…
```
df_predict.head()
```

Out[105]:

	Real	Proba_Preditcion	Prediction
89223	1	0.909554	1
158835	0	0.961803	1
108909	1	0.936763	1
270155	1	0.886823	1
23971	1	0.854169	1

In [146…
```
#Generons la matrice de confusion avec la méthode .crosstab de pandas.
#Paramètres : (valeurs réelles, prédictions, nom des lignes, nom des colonnes)
```

In [106…
```
pd.crosstab(df_predict['Real'], df_predict['Prediction'],
            rownames = ['Reel'], colnames = ['PredicT'])
```

Out[106]:

PredicT	0	1
Reel		
0	1	4381
1	1	42246

In [148…
```
#Nous pourrions également estimer la matrice de confusion en pourcentages.
#Divisons chacune des valeurs par le total des observations (nombre de lignes).
```

In [107…
```
pd.crosstab(df_predict['Real'], df_predict['Prediction'],
            rownames = ['Reel'], colnames = ['PredicT']) / df_predict.shape[0]
```

Out[107]:

PredicT	0	1
Reel		
0	0.000021	0.093954
1	0.000021	0.906003

In [150…
```
#Métrique d'exactitude (Accuracy)
#(Vrais positifs + Vrais négatifs) / Total
```

In [108…
```
VP = (pd.crosstab(df_predict['Real'], df_predict['Prediction'],
                  rownames = ['Reel'], colnames = ['PredicT']) / df_predict.shape[0]).iloc[1,1]
VP
```

Out[108]:
```
0.9060027021810462
```

In [110…
```
VN = (pd.crosstab(df_predict['Real'], df_predict['Prediction'],
                  rownames = ['Reel'], colnames = ['PredicT']) / df_predict.shape[0]).iloc[0,0]
VN
```

Out[110]:
```
2.1445881318492783e-05
```

In [111…
```
Accuracy = VP + VN
```

In [112…
```
Accuracy
```

Out[112]:
```
0.9060241480623646
```

Le modèle a une précision relativement élevée de 90 % et fait un excellent travail pour prédire les clients bons. Cependant, il a une

Le modèle a une précision relativement élevée de 90 % et fait un excellent travail pour prédire les clients bons. Cependant, il a une mauvaise performance pour prédire les clients mauvais. Cela s'explique par le fait que la majorité des observations dans la base sont des clients bons. Par conséquent, le modèle avec un seuil de coupure de 0,5 aura tendance à prédire que la plupart sont des clients bons, générant ainsi un grand nombre de faux positifs. Imaginons maintenant que nous utilisons ce modèle et ce seuil de coupure pour accorder des crédits : nous accorderions ainsi du crédit à de nombreux clients mauvais. Pour cette raison, il est nécessaire d'établir un seuil de coupure plus conservateur, disons 0,85. Recalculons la matrice de confusion avec un seuil. Avec ce nouveau seuil de coupure, nous réduirions considérablement le nombre de défauts, mais aussi le nombre de demandes approuvées. Alors, il faut trouver un seuil optime qui minimise le cout financiere.

ROC y AUC

In [113…
```
from sklearn.metrics import roc_curve, roc_auc_score
```

In [161…
```
#Définissons la courbe ROC. Nous avons besoin de deux arguments :
#Les valeurs réelles
#Les probabilités de notre modèle de prédiction
```

In [115…
```
roc_curve(df_predict['Real'], df_predict['Proba_Preditcion'])
```

Out[115]:
```
(array([0.        , 0.        , 0.        , ..., 0.99977179, 0.99977179,
        1.        ]),
 array([0.00000000e+00, 2.36703198e-05, 1.68059270e-03, ...,
        9.99905319e-01, 1.00000000e+00, 1.00000000e+00]),
 array([       inf, 0.99566605, 0.99349452, ..., 0.53623792, 0.48965534,
        0.470524  ]))
```

En résultat, nous obtenons trois tableaux :

1 tableau. Taux de faux positifs

2 tableau. Taux de vrais positifs

3 tableau. Les seuil

Extrayons chacun de ces tableaux dans des variables :

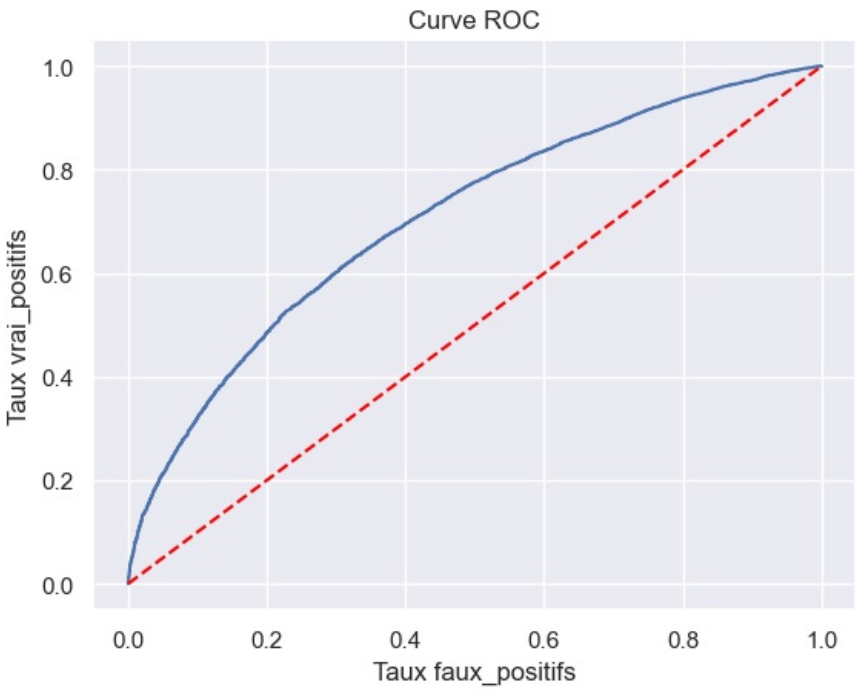
In [116…
```
faux_positifs, vrai_positifs, seuil = roc_curve(df_predict['Real'], df_predict['Proba_Preditcion'])
```

In [117…
```
# graph ROC
```

In [118…
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

In [120…
```
plt.plot(faux_positifs, vrai_positifs) # Nous définissons les données du graphique, avec des valeurs de x ficti
plt.plot(faux_positifs, faux_positifs, linestyle = '--', color = 'red') # Benchmark (Predictor 50/50) la diagon
plt.xlabel('Taux faux_positifs')
plt.ylabel('Taux vrai_positifs')
plt.title('Curve ROC')
```

Out[120]:
```
Text(0.5, 1.0, 'Curve ROC')
```



In [170…
```
# la métrique AUC (aire sous la courbe) nous sert à évaluer la qualité du modèle."
```

In [123…
```
AUC = roc_auc_score(df_predict['Real'], df_predict['Proba_Preditcion'])
```

In [124…
```
AUC #interpretation voir pdf
```

Out[124]:
```
0.7069446363103981
```

Coefficients de Gini y Kolgomorov-Smirnov

Gini : Mesure de l'inégalité entre les emprunteurs bons et mauvais. Et pourcentage cumulé des mauvais. Axe des X : Cumul total. Plus grande est l'aire sous la courbe, meilleur est le modèle.

Kolmogorov-Smirnov : Mesure à quel point le modèle sépare bien les bons et les mauvais. Plus ils sont séparés, meilleur est le modèle.

In [173…
```
# Trier le dataframe par les probabilités par ordre croissant.
```

In [137…
```
df_predict = df_predict.sort_values('Proba_Preditcion')
```

In [138…
```
df_predict.head()
```

Out[138]:

	index	Real	Proba_Preditcion	Prediction	Clients_cumulés
0	42319	0	0.470524	0	0
1	18781	1	0.489655	0	1
2	42113	1	0.513947	1	2
3	42481	1	0.520316	1	3
4	42433	1	0.521947	1	4

In [139…
```
df_predict.tail()
```

Out[139]:

	index	Real	Proba_Preditcion	Prediction	Clients_cumulés
46624	245306	1	0.994991	1	42243
46625	255463	1	0.995022	1	42244
46626	261086	1	0.995187	1	42245
46627	232642	1	0.995230	1	42246
46628	256952	1	0.995666	1	42247

In [140…
```
#Pour calculer la proportion cumulative, nous devons réindexer le dataframe selon
#l'ordre croissant des probabilités que nous avons. Nous voulons que l'observation
#avec la plus faible probabilité ait un indice de 0, la suivante 1, et ainsi de suite.
```

In [141…
```
df_predict= df_predict.reset_index()
```

In [142…
```
df_predict.head()
```

Out[142]:

	level_0	index	Real	Proba_Preditcion	Prediction	Clients_cumulés
0	0	42319	0	0.470524	0	0
1	1	18781	1	0.489655	0	1
2	2	42113	1	0.513947	1	2
3	3	42481	1	0.520316	1	3
4	4	42433	1	0.521947	1	4

In [143…
```
#Remarquons que, en écrivant par-dessus l'indice sans utiliser drop = True
#une colonne index a été générée automatiquement, préservant ainsi les indices d'origine pour identifier chaque
```

Pour créer les graphiques de performance de notre modèle, nous avons besoin de:

1. Le pourcentage cumulatif du total des clients
2. Le pourcentage cumulatif des clients bons
3. Le pourcentage cumulatif des clients mauvais Avant d'estimer le pourcentage, calculons le nombre (n) correspondant.

In [144…
```
# Cumulatif du total des clients : somme des indices.
```

In [145…
```
df_predict['Clients_cumulés'] = df_predict.index + 1
```

In [146…
```
#Cumulatif du total des clients bons : comme 'Real' pour les bons-mauvais est une variable binaire
```

```
#prenant la valeur de 1 lorsque le client est bon, tout ce que nous avons à faire est de réaliser
#la somme cumulative de cette variable en utilisant la méthode .cumsum.
```

In [147…
```
df_predict['Bons_Clients_cumulés'] = df_predict['Real'].cumsum()
```

In [148…
```
#Cumulatif du total des clients mauvais : Nous ne pouvons pas simplement sommer les zéros
#cumulés (ce serait toujours zéro). Cependant, pour chaque ligne, nous connaissons le nombre
#total de clients cumulés et le total de clients bons cumulés. Le cumulatif des clients mauvais
#devrait être la différence entre ces deux nombres.
```

In [149…
```
df_predict['Mauvais_Clients_cumulés'] = df_predict['Clients_cumulés'] - df_predict['Bons_Clients_cumulés']
```

In [186…
```
df_prediccion.head()
```

Out[186]:

	index	Real	Proba_Prediccion	Prediccion	Monto	Número Acumulado de Clientes	Número Acumulado de Clientes Buenos	Número Acumulado de Clientes Malos
0	42319	0	0.470524	0	8400	1	0	1
1	18781	1	0.489655	0	1200	2	1	1
2	42113	1	0.513947	1	6625	3	2	1
3	42481	1	0.520316	1	1000	4	3	1
4	42433	1	0.521947	1	3000	5	4	1

In [150…
```
df_predict.tail()
```

Out[150]:

	level_0	index	Real	Proba_Preditcion	Prediction	Clients_cumulés	Bons_Clients_cumulés	Mauvais_Clients_cumulés
46624	46624	245306	1	0.994991	1	46625	42243	4382
46625	46625	255463	1	0.995022	1	46626	42244	4382
46626	46626	261086	1	0.995187	1	46627	42245	4382
46627	46627	232642	1	0.995230	1	46628	42246	4382
46628	46628	256952	1	0.995666	1	46629	42247	4382

In [151…
```
#Une fois que nous avons estimé le nombre cumulatif, nous devons estimer la proportion cumulée.
#Pour le % Cumulatif de Clients, nous divisons le Nombre Cumulatif de Clients par le Total de Clients.
```

In [154…
```
df_predict['%_Clients_Cumulés'] = df_predict['Clients_cumulés'] / df_predict.shape[0] #0 total de filas
```

In [155…
```
#Pour le % Cumulatif de Clients Bons, nous divisons le Nombre Cumulatif de Clients
#Bons par le Total de Clients Bons.
```

In [159…
```
df_predict['%_Clients_Bons_Cumulés'] = df_predict['Bons_Clients_cumulés'] / df_predict['Real'].sum()
```

In [160…
```
#Pour le % Cumulatif de Clients Mauvais, nous divisons le Nombre Cumulatif de Clients Mauvais par le Total de C
```

In [161…
```
df_predict['%_Clients_Mauvais_Cumulés'] = df_predict['Mauvais_Clients_cumulés'] / (df_predict.shape[0] - df_pre
```

In [162…
```
df_predict.head()
```

Out[162]:

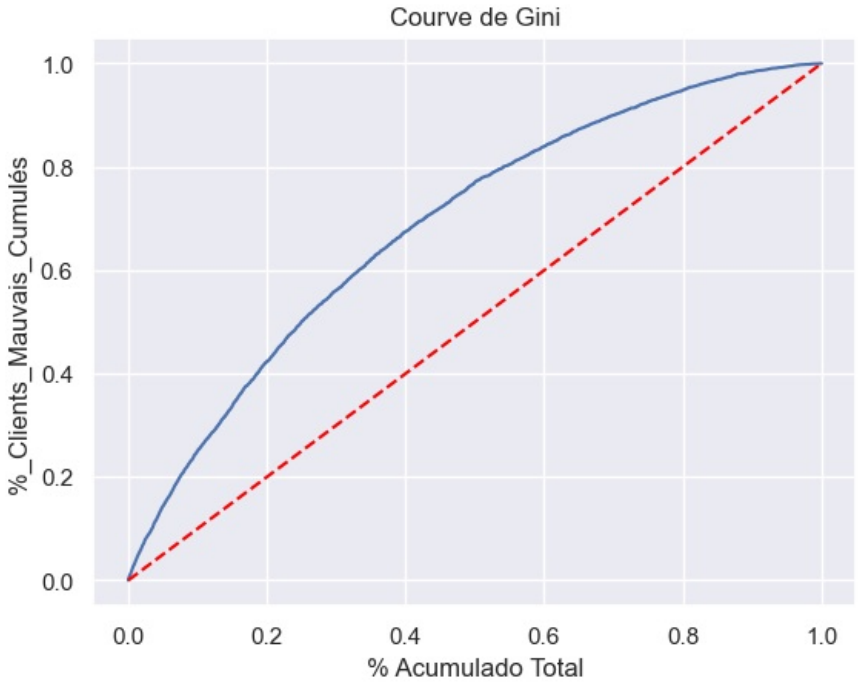
	level_0	index	Real	Proba_Preditcion	Prediction	Clients_cumulés	Bons_Clients_cumulés	Mauvais_Clients_cumulés	%_Clients_Cumulés
0	0	42319	0	0.470524	0	1	0	1	0.000021
1	1	18781	1	0.489655	0	2	1	1	0.000043
2	2	42113	1	0.513947	1	3	2	1	0.000064
3	3	42481	1	0.520316	1	4	3	1	0.000086
4	4	42433	1	0.521947	1	5	4	1	0.000107

In [163…
```
# Coeff de Gini y Kolgomorov-Smirnov.
```

In [164…
```
#Nous avons calculé les pourcentages cumulés, ce qui est tout ce dont nous avons besoin
#pour créer les graphiques et estimer les coefficients de Gini et de K-S.
#Commençons par le graphique de Gini, qui représente le % Cumulatif de Mauvais (axe Y) en fonction du % Cumulat
```

In [166…
```
plt.plot(df_predict['%_Clients_Cumulés'], df_predict['%_Clients_Mauvais_Cumulés'])
plt.plot(df_predict['%_Clients_Cumulés'], df_predict['%_Clients_Cumulés'], linestyle = '--', color = 'red') # B
plt.xlabel('% Acumulado Total')
plt.ylabel('%_Clients_Mauvais_Cumulés')
plt.title('Courve de Gini')
```

Out[166]:
```
Text(0.5, 1.0, 'Courve de Gini')
```



In [167…
```
#Il y a des similitudes entre le graphique de Gini et l'AUC.
#En fait, le coefficient de Gini peut être exprimé comme :
# GINI = AUC*2-1
```

In [168…
```
Coef_Gini = AUC*2-1
Coef_Gini
```

Out[168]:
```
0.4138892726207961
```

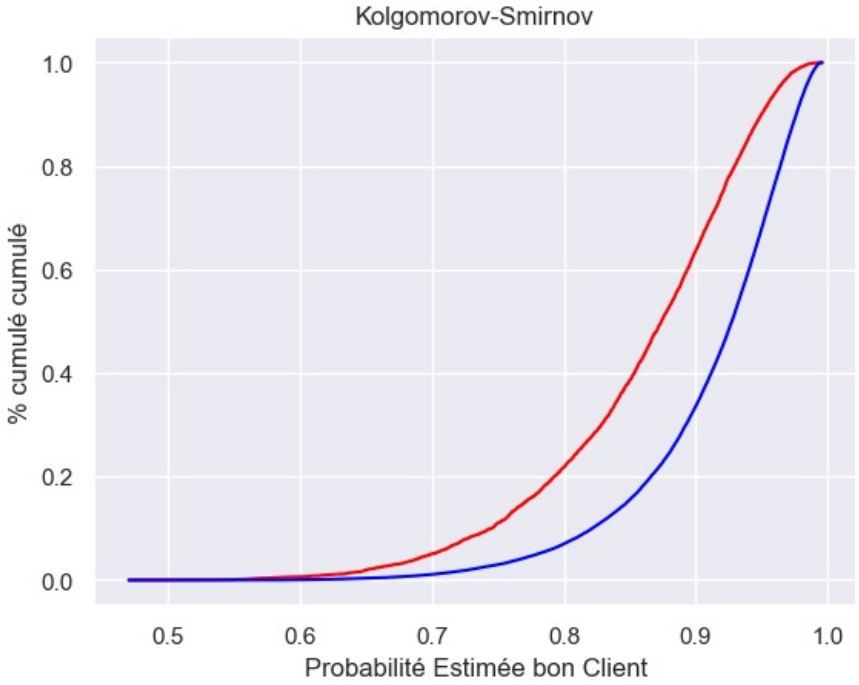
In [169…
```
#Un indice de Gini de 0 indiquerait que les emprunteurs à risque sont répartis de manière équitable
#sur l'ensemble de la gamme de notation ; en d'autres termes, le score de crédit n'a pas attribué
#de scores plus bas à davantage d'emprunteurs en défaut, comme le ferait un indicateur avec un pouvoir
#prédictif plus fort. Un indice de coefficient de 100 % indiquerait que tous les emprunteurs en défaut
#se sont vu attribuer avec succès les scores les plus bas possibles. Les coefficients de Gini de 45 %
#ou plus sont considérés comme des indicateurs d'une forte précision dans l'évaluation du crédit.
#Un coefficient de 41,4 % est jugé satisfaisant.
```

In [170…
```
# graph K-S
#Rappelons que la statistique de K-S mesure la distance (sur l'axe des Y)
#entre deux fonctions de distribution cumulée. Plus la distance est grande,
#meilleure est la caractéristique qui les distingue. Dans notre cas spécifique, les fonctions sont :

#% Cumulatif de Clients Bons en fonction de la Probabilité d'être bon (Proba_Prediccion)
#% Cumulatif de Clients Mauvais en fonction de la Probabilité d'être bon
#Si le modèle était parfait, la distance maximale serait égale à 1. Pour un modèle de prédiction aléatoire, la
```

In [172…
```
plt.plot(df_predict['Proba_Preditcion'], df_predict['%_Clients_Mauvais_Cumulés'], color = 'red')
plt.plot(df_predict['Proba_Preditcion'], df_predict['%_Clients_Bons_Cumulés'], color = 'blue')
plt.xlabel('Probabilité Estimée bon Client')
plt.ylabel('% cumulé cumulé')
plt.title('Kolgomorov-Smirnov')
```

Out[172]:
```
Text(0.5, 1.0, 'Kolgomorov-Smirnov')
```



In [173…
```
#Le coefficient K-S est la distance (verticale) maximale entre la courbe rouge et la courbe bleue.
#Nous pouvons le calculer avec les données du dataframe en prenant le maximum de la différence entre
#le % cumulatif des mauvais et le % cumulatif des bons :
```

In [175…
```
Coef_KS = max(df_predict['%_Clients_Mauvais_Cumulés']-df_predict['%_Clients_Bons_Cumulés'])
```

In [176…
```
Coef_KS
```

Out[176]: 0.3045755171087095

In [177…
```
#Il ne s'approche pas de 1, mais il est significativement plus élevé que zéro.
#Les deux distributions cumulatives sont suffisamment éloignées.
#Nous pouvons affirmer que le modèle a un pouvoir de prédiction satisfaisant.
```

In [178…
```
x_test.head()
```

Out[178]:

	Unnamed: 0	id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installment	grade	...	dti_19.9-20.8	dti_
89223	89223	7073644	8735123	9600	9600	9600.0	36 months	15.10	333.26	C	...	0	
158835	158835	3640390	4592970	14000	14000	14000.0	36 months	7.62	436.26	A	...	0	
108909	108909	6527485	8079529	16750	16750	16750.0	36 months	8.90	531.87	A	...	0	
270155	270155	32419070	35032306	19750	19750	19750.0	60 months	20.20	525.46	E	...	0	
23971	23971	606796	778433	15000	9475	9450.0	60 months	8.88	196.14	B	...	1	

5 rows × 286 columns

In [ ]: