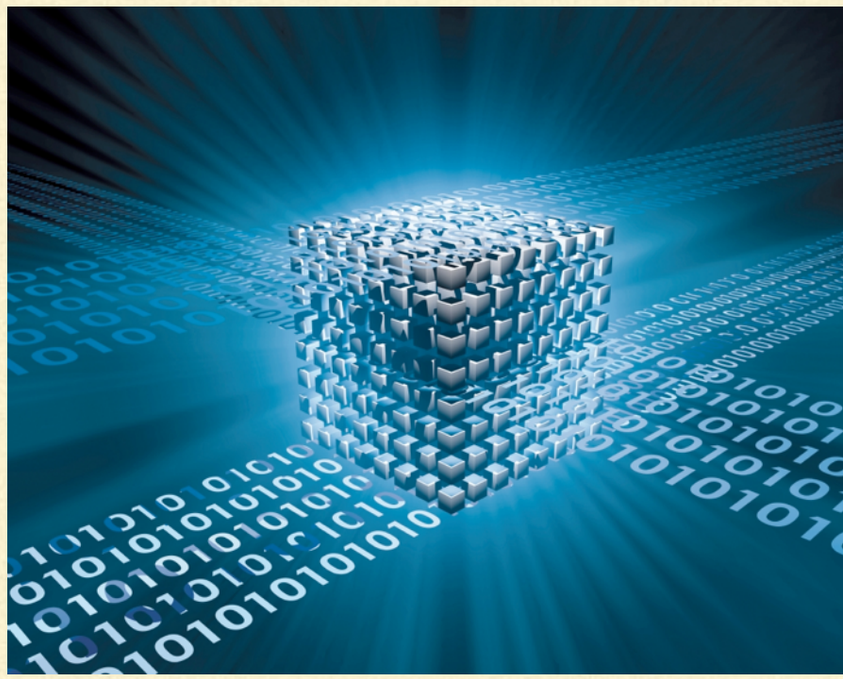
Accorder un crédit bancaire



La régression logistique : un outil dans la prise de décision d'un accord de crédit bancaire ?

Plan du TIPE

1. Théorie de la régression logistique
2. Choix des paramètres pris en compte
3. Régression logistique sous Python
4. Exploitation des résultats

I. Théorie de la régression logistique

1. Objectif de la régression logistique

On va chercher à expliquer la survenue d'un évènement
Ici, l'accord de crédit bancaire, noté Y

Pour cela, on cherche la probabilité du succès

La probabilité d'obtenir son crédit

2. Notations

Y prendra les valeurs {0,1}. On considèrera :

- 1 si le crédit est accordé
- 0 si le crédit est refusé

$X=(X_1,X_2,\ldots,X_i)$: descripteurs, variables explicatives

Pour un individu ω, on note la probabilité d'avoir Y=1, c'est-à-dire, d'avoir son crédit accordé

$$P[Y(\omega) = 1] = p(\omega) = p.$$

On note la probabilité d'avoir Y=1, sachant les valeurs prises par les variables explicatives est notée :

$$P[Y(\omega) = 1/X(\omega)] = \pi(\omega) = \pi.$$

3. Fonction logistique

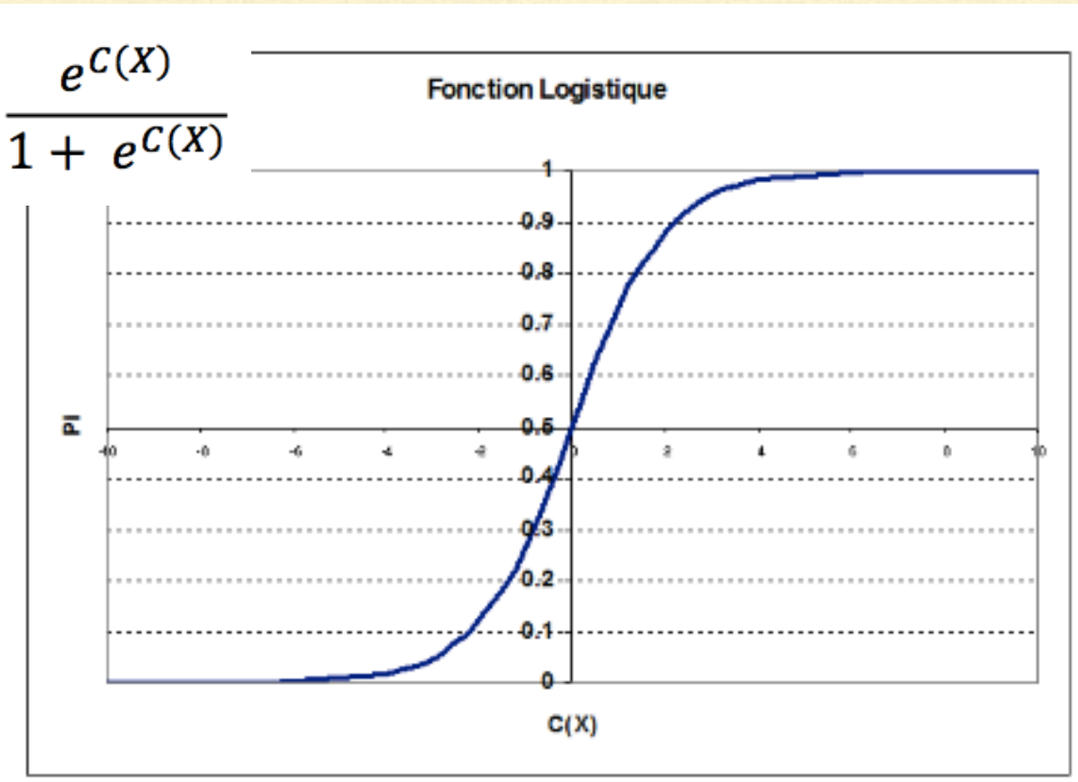
$$\ln\left(\frac{P(Y(\omega)=1/X(\omega)}{P(Y(\omega)=0/X(\omega)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_i X_i$$

$$\ln\left(\frac{\pi(\omega)}{1-\pi(\omega)}\right) = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_i X_i$$

3. Fonction logistique

$$\pi(\omega) = \frac{e^{\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_iX_i}}{1+e^{\beta_0+\beta_1X_1+\beta_2X_2+\cdots+\beta_iX_i}}$$

$$\frac{e^{C(X)}}{1+e^{C(X)}}$$



Règle de décision :

* Si π ≥ 0,5 alors : Y = 1
* Si π < 0,5 alors : Y = 0

4. Maximum de la vraisemblance

★ $$P[Y(\omega)/X(\omega)] = \pi(\omega)^{\gamma(\omega)} \times (1 - \pi(\omega))^{1-\gamma(\omega)}$$

★ Vraisemblance

$$L = \prod_{\omega} (1 - \pi(\omega))^{1-\gamma(\omega)} \times \pi(\omega)^{\gamma(\omega)}$$

★ Log - Vraisemblance

$$LL = \sum_{\omega} \gamma(\omega) \times \ln(\pi(\omega)) + (1 - \gamma(\omega)) \times \ln(1 - \pi(\omega))$$

5. Méthode de Newton Raphson

★ Variables explicatives

$$X = \begin{pmatrix} 1 & X_1(\omega_1) & \dots & X_i(\omega_1) \\ \vdots & \vdots & \ddots & \vdots \\ 1 & X_1(\omega_n) & \dots & X_i(\omega_n) \end{pmatrix}$$

★ Poids

$$\beta = \begin{pmatrix} \beta_0 \\ \vdots \\ \beta_i \end{pmatrix}$$

n : nombre d'individus
i : nombre de paramètres pris en compte

5. Méthode de Newton Raphson

$$\beta^{k+1} = \beta^{k} - \left(\frac{\Delta LL}{\Delta \beta}\right)^{-1} \times \left(\frac{\partial LL}{\partial \beta}\right)$$

★ Gradient

$$\frac{\partial LL}{\partial \beta} = X^{T} \begin{pmatrix} \gamma(\omega_1) - \pi(\omega_1) \\ \vdots \\ \gamma(\omega_n) - \pi(\omega_n) \end{pmatrix}$$

★ Matrice hessienne

$$H(\beta) = \frac{\Delta LL}{\Delta \beta} = -X^{T} \begin{pmatrix} \pi(\omega_1)\times(1-\pi(\omega_1)) & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & \pi(\omega_n)\times(1-\pi(\omega_n)) \end{pmatrix} X$$

II. Choix des paramètres pris en compte

1. Les paramètres choisis

* Solde total sur les cartes de crédit
* L'âge du client
* Ratio d'endettement
* Salaire mensuel
* Nombre de crédits bancaires en cours pour l'emprunteur.
* Nombre de prêts immobiliers.
* Nombre de personnes à charge, incluant lui-même : enfants, conjoint,..

2. Les données fournies par Kaggle

- Téléchargement des données sur le site

- Importation de la bibliothèque pandas

```
import pandas as pd
```

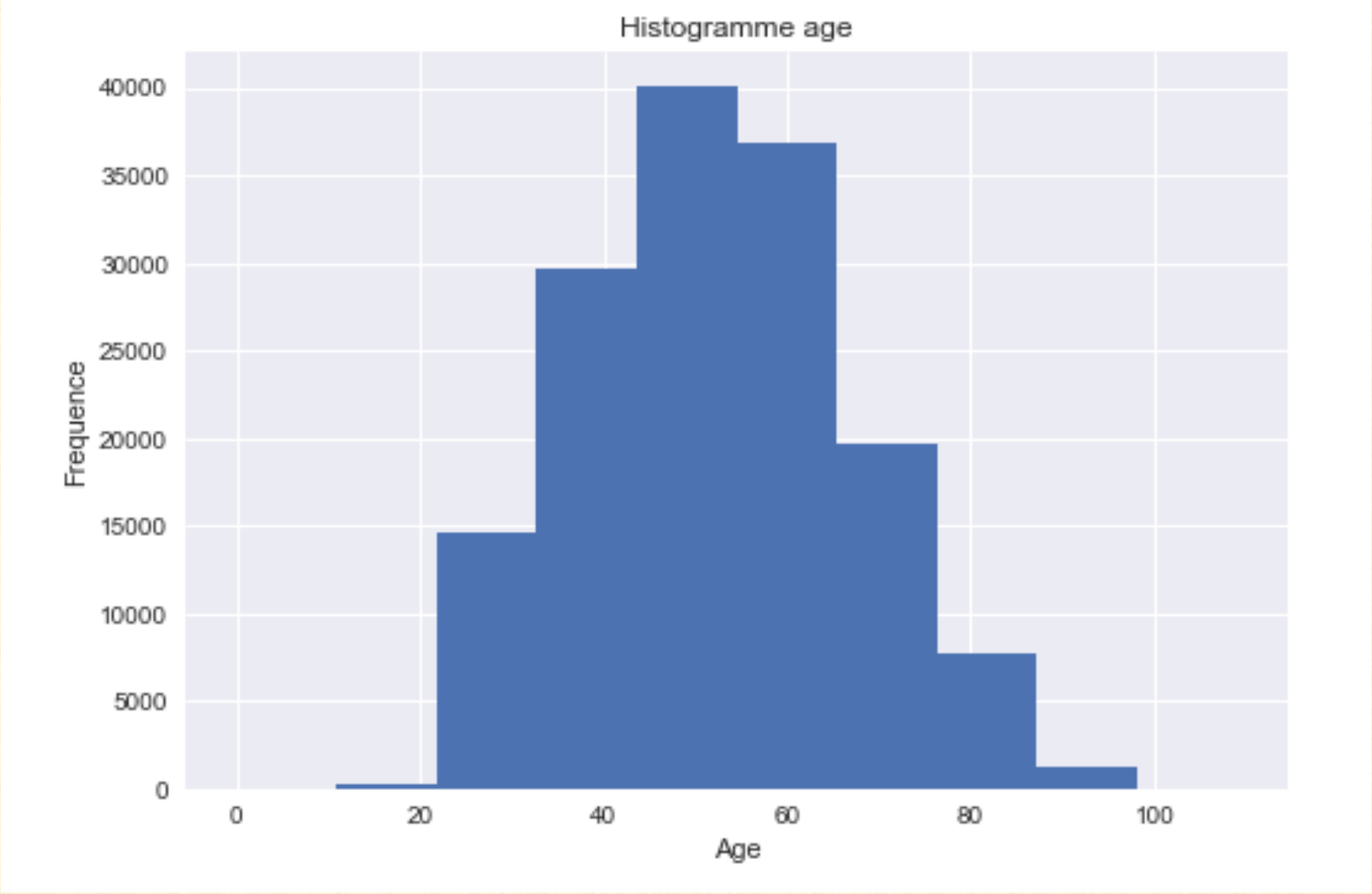
- Lecture des données grâce à pandas

```
train = pd.read_csv('cs-train.csv')
```

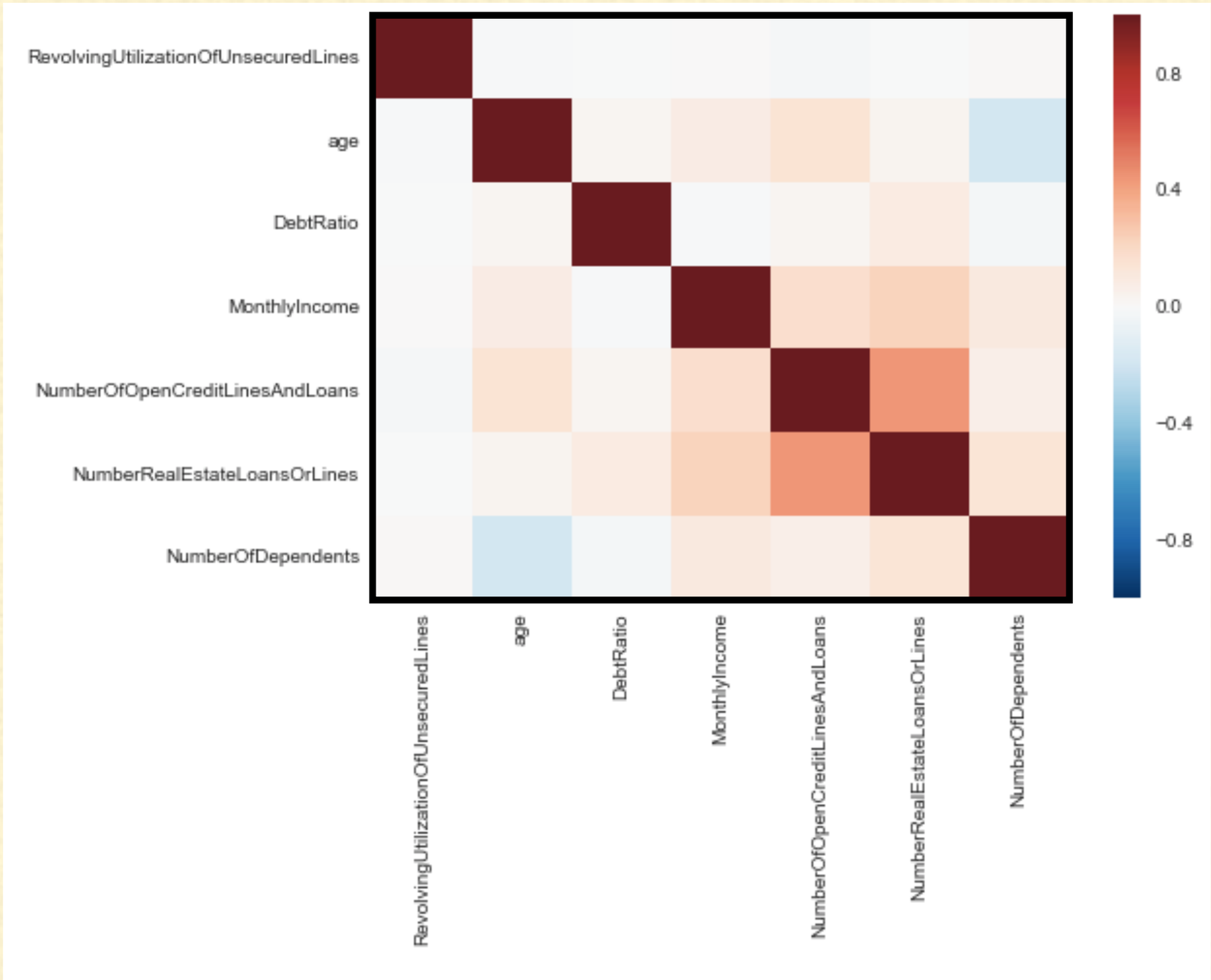
	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoan
1	1	0.766127	45	2	0.802982	9120.0	13
2	0	0.957151	40	0	0.121876	2600.0	4
3	0	0.658180	38	1	0.085113	3042.0	2
4	0	0.233810	30	0	0.036050	3300.0	5
5	0	0.907239	49	1	0.024926	63588.0	7
6	0	0.213179	74	0	0.375607		3
7	0	0.305682	57	0	5710.000 00	NaN	8

- Remplacement des données manquantes par la moyenne de toutes les autres

3. Etude descriptive



3. Etude descriptive



III. Régression Logistique sous Python

1. Le premier algorithme utilisant sklearn

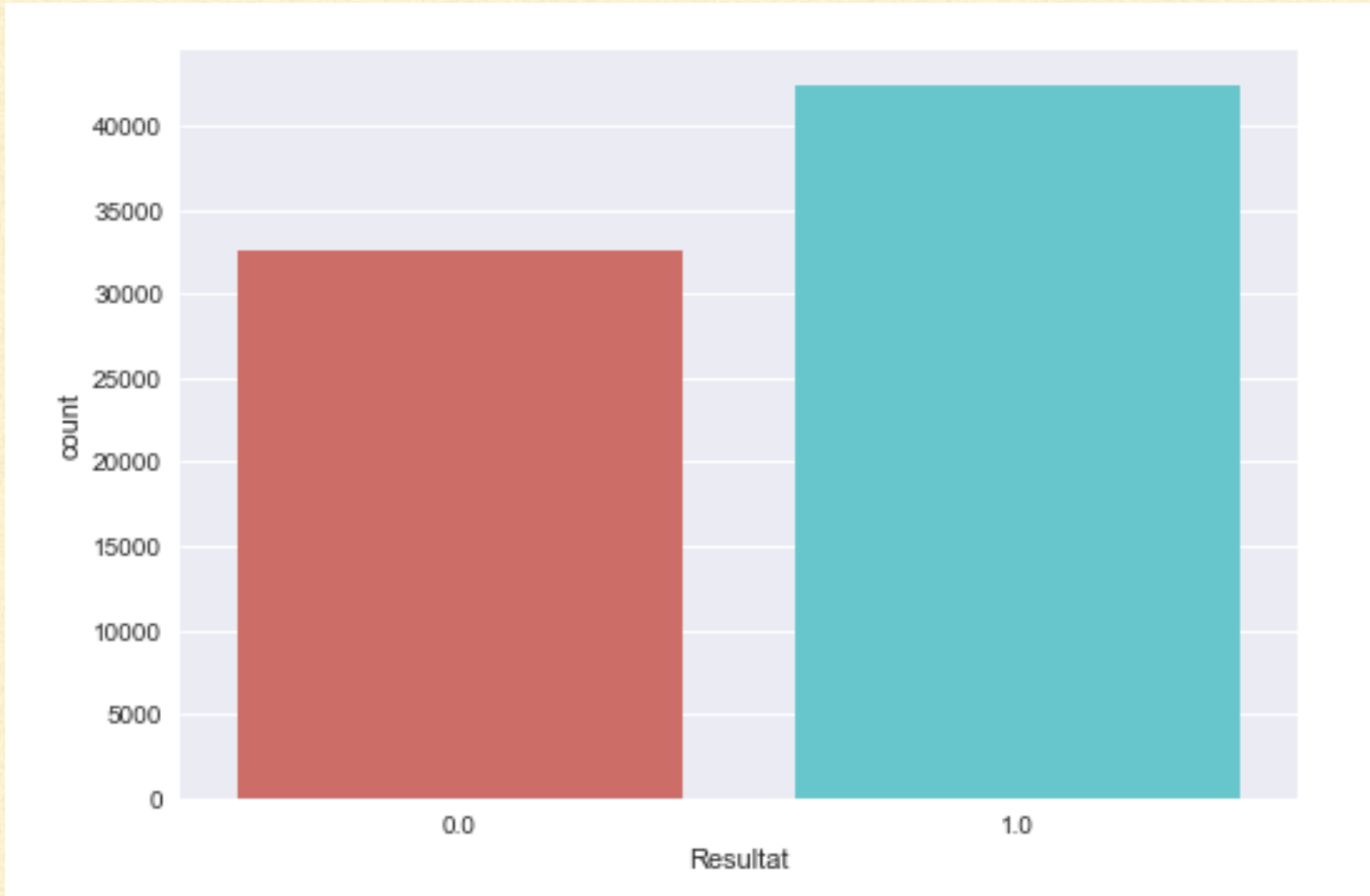
Logit Regression Results

Dep. Variable:	y	No. Observations:	75000
Model:	Logit	Df Residuals:	74993
Method:	MLE	Df Model:	6
Date:	Wed, 06 Jun 2018	Pseudo R-squ.:	0.02425
Time:	12:43:47	Log-Likelihood:	-18035.
converged:	True	LL-Null:	-18483.
		LLR p-value:	2.061e-190

	coef
x1	5.598e-05
x2	0.0476
x3	2.1e-05
x4	4.834e-05
x5	0.0141
x6	-0.0635
x7	-0.0753

← Les 7 coefficients obtenus

1. Le premier algorithme utilisant sklearn



2. Le deuxième algorithme

In [47]:
```python
# Fonction logistique

# f_logistique
def f_logistique(z):
    return 1 / (1 + np.exp(-z))
```

In [48]:
```python
# Vraisemblance logarithmique

# f_ll
# descripteurs : matrice (n, p)
# objectifs    : vecteur y (n)
# poids        : vecteur (p)
# return       : réel
def f_ll(descripteurs, objectifs, poids):
    predictions = f_logistique(np.dot(descripteurs, poids))
    return np.sum(objectifs * np.log(predictions) + (1 - objectifs) * np.log(1 - predictions))
```

In [49]:
```python
# Gradient de la vraisemblance logarithmique

# gradient_ll
# descripteurs : matrice (n, p)
# objectifs    : vecteur (n)
# poids        : vecteur (p)
# return       : vecteur (p)
def gradient_ll(descripteurs, objectifs, poids):
    predictions = f_logistique(np.dot(descripteurs, poids))
    return np.dot(descripteurs.T, objectifs - predictions)
```

In [50]:
```python
# Matrice hessienne de la vraisemblance logarithmique

# hessian_ll
# descripteurs : matrice (n, p)
# objectifs    : vecteur (n)
# poids        : vecteur (p)
# return       : matrice (p, p)
def hessian_ll(descripteurs, objectifs, poids):
    predictions = f_logistique(np.dot(descripteurs, poids))
    diagonale = np.diag([predictions[i, 0]*(1-predictions[i, 0]) for i in range (np.shape(predictions)[0])])
    return -np.dot(np.dot(descripteurs.T, diagonale), descripteurs)
```

** descripteurs = variables explicatives*

** objectifs = accord crédit : accordé ou refusé (0 ou 1)*

** poids = paramètres de la fonction logistique*

2. Le deuxième algorithme

```
Précision de la régression : 93.276 %
[[  1.13295830e+00]
 [  3.67773620e-05]
 [  2.80345634e-02]
 [  1.46334297e-05]
 [  2.08284984e-05]
 [  5.14470694e-03]
 [ -2.23641813e-02]
 [ -1.04944123e-01]]
```

Les coefficients obtenus

IV. Exploitation des résultats

1. Les différents résultats obtenus

° Les coefficients obtenus avec le 1er algorithme

```
[[   4.17112775e-05    3.27700523e-02    1.33593395e-05    2.73526026e-05
     1.13146309e-03   -3.90825909e-02   -1.12145758e-01]]
```

° Les coefficients obtenus avec le 2ème algorithme

```
[   3.67773620e-05]
[   2.80345634e-02]
[   1.46334297e-05]
[   2.08284984e-05]
[   5.14470694e-03]
[  -2.23641813e-02]
[  -1.04944123e-01]]
```

Les coefficients obtenus sont cohérents

° Probabilité d'obtenir son crédit avec le 1er algorithme

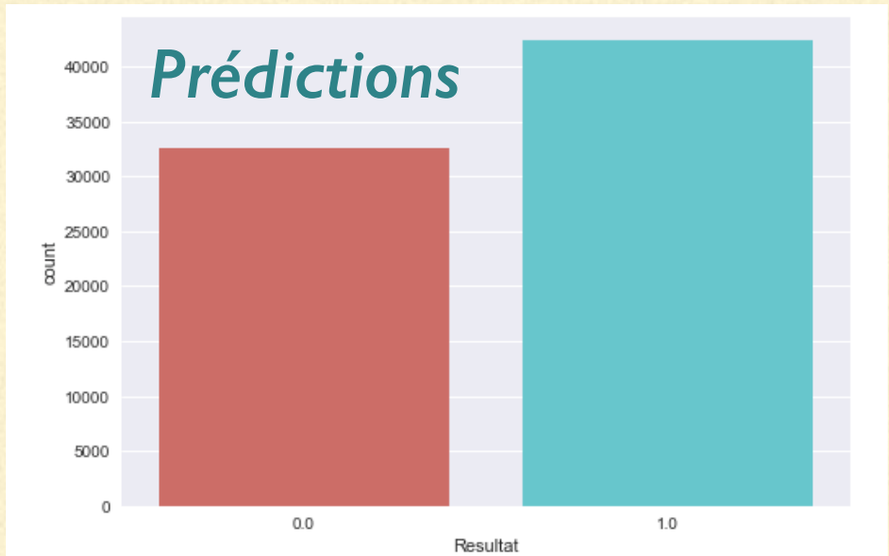
```
[ 0.55845347  0.45842711  0.4624171  ...,  0.61348928  0.62488068
  0.57787508]
```

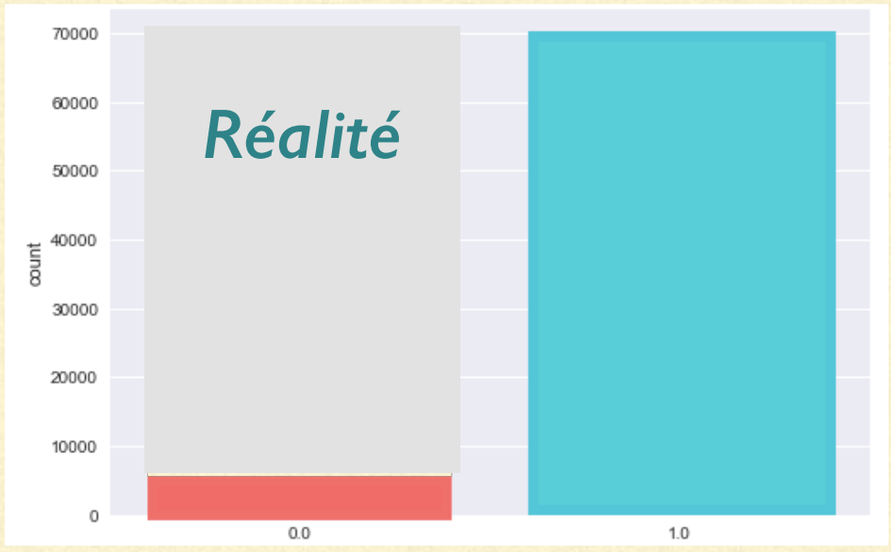
° Probabilité d'obtenir son crédit avec le 2ème algorithme

```
[[ 0.84197543]
 [ 0.78688457]
 [ 0.78927118]
 ...,
 [ 0.86028663]
 [ 0.86799667]
 [ 0.84919524]]
```

1. Les différents résultats obtenus

° Avec sklearn





56,53 % des prêts sont accordés
2,52% des prêts accordés ne le sont pas en réalité
39,35 % des prêts non accordés le sont pour les vraies valeurs

° Avec Newton

Avec une règle de décision à 0.5, l'algorithme de Newton refuse le crédit à seulement 1 personne
Règle de décision à 0.703 : 4971 crédits ne sont pas accordés => 7% de l'échantillon => même pourcentage que les vraies valeurs

2. Conclusion

Les résultats obtenus montrent que le module sklearn est basé sur une étude plus approfondie de la méthode de régression logistique programmée ici, qui utilise simplement l'algorithme de Newton-Raphson.

Elle permet à une banque, selon qu'elle souhaite être prudente ou bien se développer, de corriger la règle de décision à une valeur qui correspond à ses intérêts.

Algorithmes

1. Importations

° Importation des bibliothèques

In [1]:
```
import pandas as pd
import numpy as np
import scipy as sc
import csv
from __future__ import division
```

In [2]:
```
import seaborn as sns
import matplotlib.pyplot as plt
```

° Importation des données

In [3]:
```
train = pd.read_csv('cs-training2.csv')
```

In [4]:
```
train
```

Out[4]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCre
1	1	0.766127	45	2	0.802982	9120.0	13
2	0	0.957151	40	0	0.121876	2600.0	4
3	0	0.658180	38	1	0.085113	3042.0	2
4	0	0.233810	30	0	0.036050	3300.0	5
5	0	0.907239	49	1	0.024926	63588.0	7
6	0	0.213179	74	0	0.375607	3500.0	3
7	0	0.305682	57	0	5710.000000	NaN	8
8	0	0.754464	39	0	0.209940	3500.0	8
9	0	0.116951	27	0	46.000000	NaN	2
10	0	0.189169	57	0	0.606291	23684.0	9

2. Etude descriptive & préparation des données

In [5]:
```
train.describe()
```

Out[5]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	Numb
count	150000.000000	150000.000000	150000.000000	150000.000000	150000.000000	1.202690e+05	1500
mean	0.066840	6.048438	52.295207	0.421033	353.005076	6.670221e+03	8.452
std	0.249746	249.755371	14.771866	4.192781	2037.818523	1.438467e+04	5.145
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000e+00	0.000
25%	0.000000	0.029867	41.000000	0.000000	0.175074	3.400000e+03	5.000
50%	0.000000	0.154181	52.000000	0.000000	0.366508	5.400000e+03	8.000
75%	0.000000	0.559046	63.000000	0.000000	0.868254	8.249000e+03	11.00
max	1.000000	50708.000000	109.000000	98.000000	329664.000000	3.008750e+06	58.00

° Imputation des valeurs manquantes

In [6]:
```
from sklearn.preprocessing import Imputer
imp = Imputer(missing_values=np.nan, strategy="mean", axis=0)
train_filtered = pd.DataFrame(imp.fit_transform(train), columns=train.columns)
```

In [7]:
```
train_filtered
```

Out[7]:

	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCre
0	1.0	0.766127	45.0	2.0	0.802982	9120.000000	13.0
1	0.0	0.957151	40.0	0.0	0.121876	2600.000000	4.0
2	0.0	0.658180	38.0	1.0	0.085113	3042.000000	2.0
3	0.0	0.233810	30.0	0.0	0.036050	3300.000000	5.0
4	0.0	0.907239	49.0	1.0	0.024926	63588.000000	7.0
5	0.0	0.213179	74.0	0.0	0.375607	3500.000000	3.0

Annexes

° Séparation en 2 des données d'apprentissage

- xtrain = valeurs explicatives
- ytrain = accord ou refus de crédit

In [8]:
```
xtrain = train_filtered.drop(['SeriousDlqin2yrs', 'NumberOfTime30-59DaysPastDueNotWorse', 'NumberOfTime60-89DaysPastDu
xtrain
```

Out[8]:

	RevolvingUtilizationOfUnsecuredLines	age	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberRealEstateLoans(
0	0.766127	45.0	0.802982	9120.000000	13.0	6.0
1	0.957151	40.0	0.121876	2600.000000	4.0	0.0
2	0.658180	38.0	0.085113	3042.000000	2.0	0.0
3	0.233810	30.0	0.036050	3300.000000	5.0	0.0
4	0.907239	49.0	0.024926	63588.000000	7.0	1.0
5	0.213179	74.0	0.375607	3500.000000	3.0	1.0
6	0.305682	57.0	5710.000000	6670.221237	8.0	3.0
7	0.754464	39.0	0.209940	3500.000000	8.0	0.0
8	0.116951	27.0	46.000000	6670.221237	2.0	0.0
9	0.189169	57.0	0.606291	23684.000000	9.0	4.0

In [9]:
```
ytrain = train_filtered.drop(['RevolvingUtilizationOfUnsecuredLines','age','NumberOfTime30-59DaysPastDueNotWorse','Deb
ytrain
```

Out[9]:

	SeriousDlqin2yrs
0	1.0
1	0.0
2	0.0
3	0.0
4	0.0
5	0.0

Annexes

In [10]:
```
from sklearn import model_selection
```

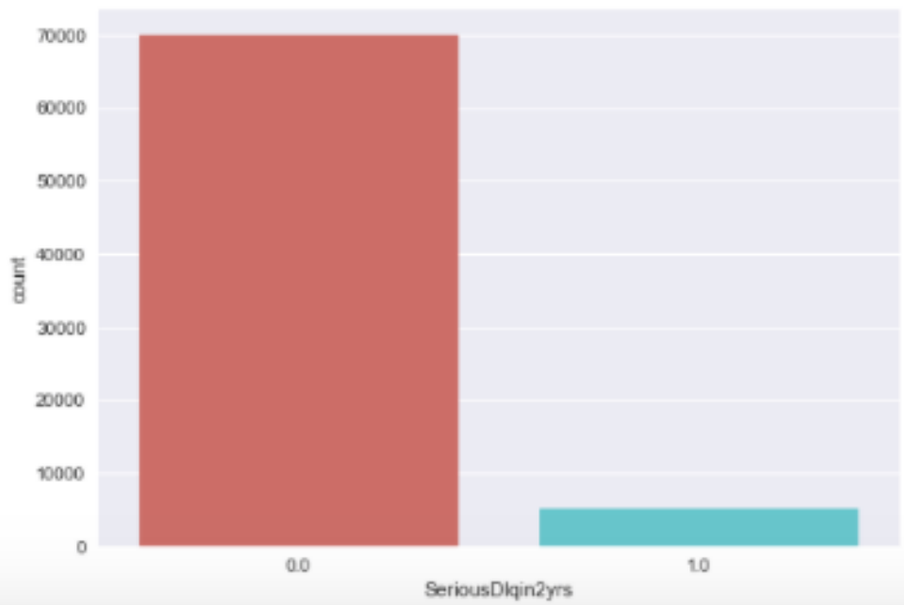
In [11]:
```
X_app,X_test,y_app,y_test = model_selection.train_test_split(xtrain,ytrain,test_size = 75000,random_state=0)
print(X_app.shape,X_test.shape,y_app.shape,y_test.shape)
X_app
```

```
((75000, 7), (75000, 7), (75000, 1), (75000, 1))
```

Out[11]:

	RevolvingUtilizationOfUnsecuredLines	age	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberRealEstateL
28047	0.027239	60.0	0.319106	3982.000000	10.0	1.0
35766	1.000000	49.0	0.023732	15000.000000	1.0	1.0
63982	0.013803	60.0	0.294570	6500.000000	6.0	1.0
106436	0.515596	55.0	4366.000000	6670.221237	11.0	2.0
92039	0.038281	28.0	2857.000000	6670.221237	10.0	1.0
18323	0.000000	91.0	0.000000	6670.221237	3.0	0.0
95740	0.275536	31.0	0.096981	5000.000000	4.0	0.0
11116	0.213512	39.0	0.189581	10000.000000	9.0	0.0
51495	0.033366	50.0	0.167823	4033.000000	10.0	0.0

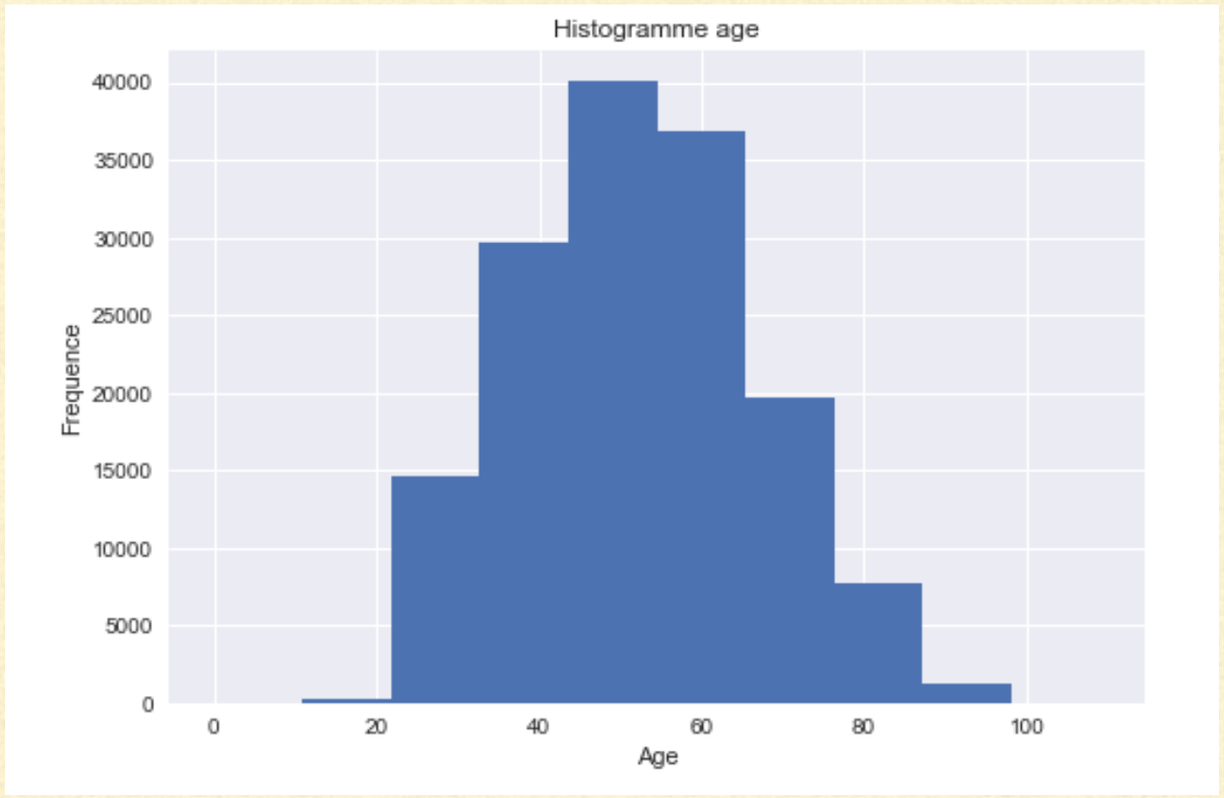
In [12]:
```
sns.countplot(x='SeriousDlqin2yrs', data=y_app, palette='hls')
plt.show()
sns.countplot(x='SeriousDlqin2yrs', data=y_test, palette='hls')
plt.show()
```

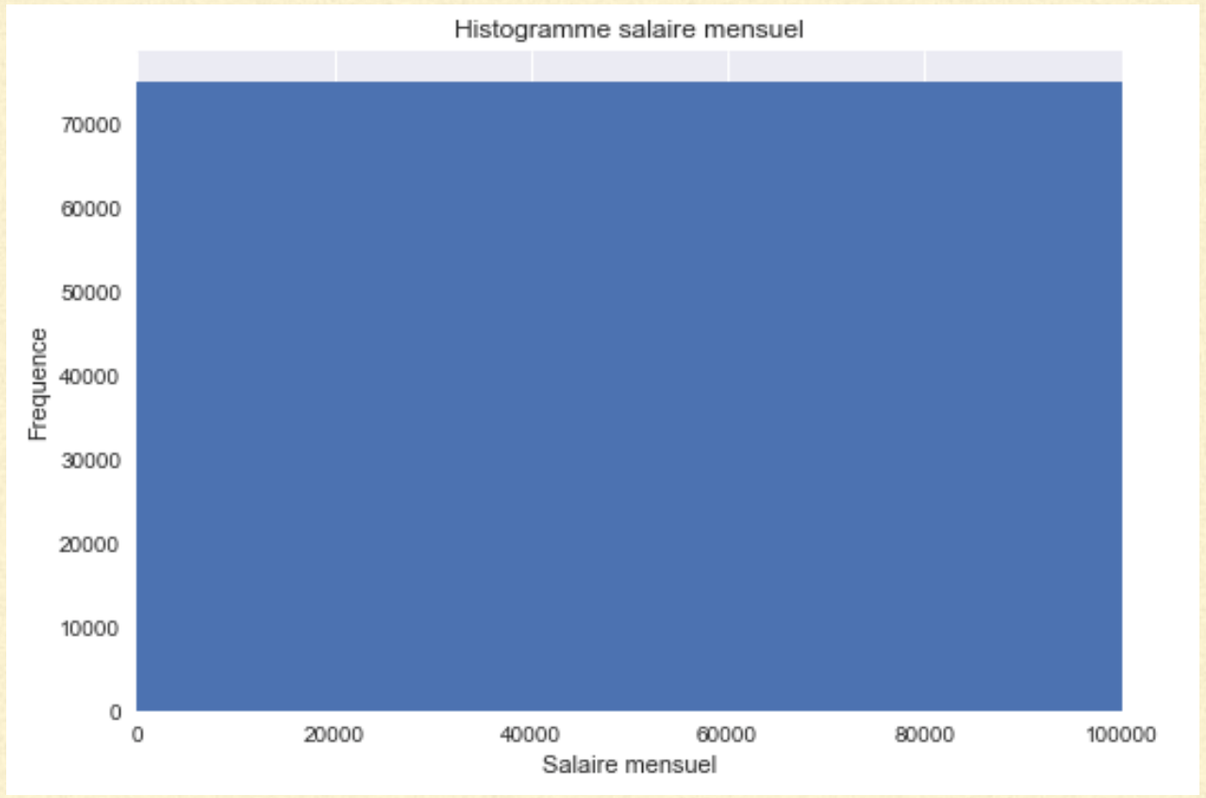


Annexes

° Quelques représentations graphiques

```
In [13]: xtrain.age.hist()
         plt.title('Histogramme age')
         plt.xlabel('Age')
         plt.ylabel('Frequence')
         plt.savefig('hist_age')
         plt.show()
```



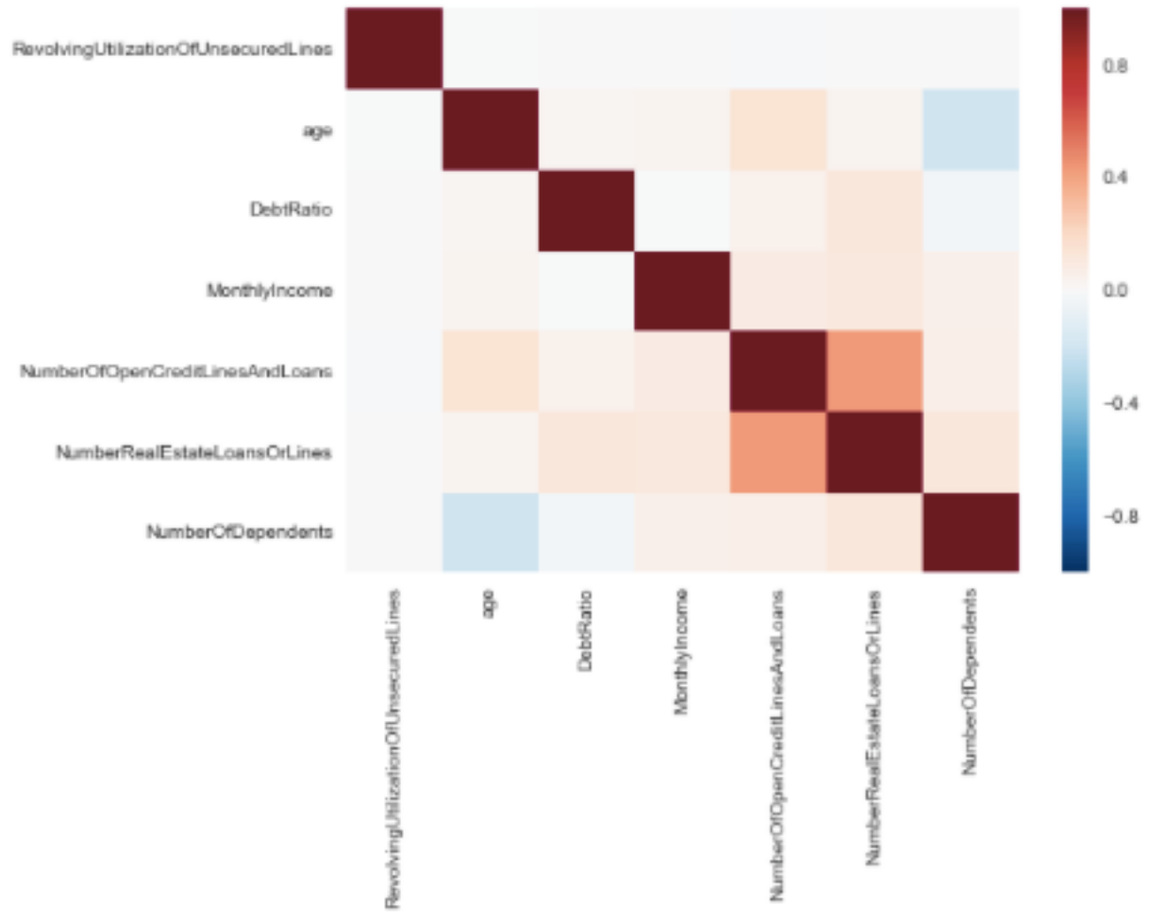



```
In [14]: X_app.MonthlyIncome.hist()
         plt.title('Histogramme salaire mensuel')
         plt.xlabel('Salaire mensuel')
         plt.xlim([0, 100000])
         plt.ylabel('Frequence')
         plt.savefig('hist_age')
         plt.show()
```

Annexes

° Indépendance des variables explicatives

In [15]:
```
sns.heatmap(xtrain.corr())
plt.show()
```



° Convertion en matrice

In [16]:
```
X_appnp = np.array(X_app)
X_testnp = np.array(X_test)
y_appn = np.array(y_app)
y_testn = np.array(y_test)

A = np.ones((75000,1))
y_appnp = A - y_appn
y_testnp = A - y_testn
```

3. Régression Logistique utilisant sklearn

° Régression logistique sur les données d'apprentissage

In [17]:
```
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
logit=LogisticRegression(class_weight='balanced')
logit.fit(X_appnp,y_appnp)
#les coefficients
print(logit.score(X_appnp,y_appnp))
print(logit.coef_)
print(logit.intercept_)
```

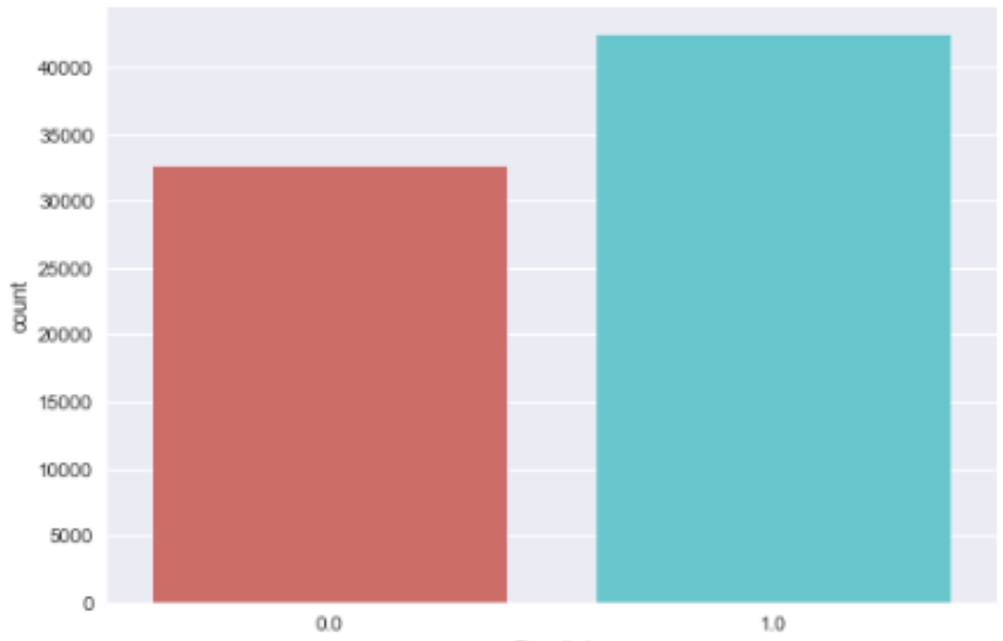
```
/Users/laurinebonin/anaconda/lib/python2.7/site-packages/sklearn/utils/validation.py:526: DataConversionWarning: A c
olumn-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example u
sing ravel().
  y = column_or_1d(y, warn=True)

0.581893333333
[[  4.17112775e-05   3.27700523e-02   1.33593395e-05   2.73526026e-05
    1.13146309e-03  -3.90825909e-02  -1.12145758e-01]]
[-1.65745345]
```

° Résultat sur les mêmes données

In [18]:
```
resultat_apprentissage = logit.predict(X_appnp)
print(resultat_apprentissage)
probabilite = logit.predict_proba(X_appnp)
p = probabilite[:,1]
print(p)
numero = X_app.index.values
res1 = pd.DataFrame({'Id': numero, 'Resultat': resultat_apprentissage})
l=res1.to_csv('pred_GiveMeSomeCredit.csv', index=False, sep=',')
tab_accord = pd.read_csv('pred_GiveMeSomeCredit.csv')
tab_accord
sns.countplot(x='Resultat', data=tab_accord, palette='hls')
plt.show()
```

Annexes

```
[ 1.  0.  1. ...,  1.  1.  0.]
[ 0.59625987  0.49605553  0.61164978 ...,  0.619836    0.69734529
  0.44853889]
```



In [19]:
```
probabilite_credit =pd.DataFrame({'Id': numero, 'Prob_credit': p})
l2 = probabilite_credit.to_csv('pred_prob.csv', index=False, sep=',')
tab_prob = pd.read_csv('pred_prob.csv')
tab_prob
```

Out[19]:

	Id	Prob_credit
0	28047	0.596260
1	35766	0.496056
2	63982	0.611650
3	106436	0.579319
4	92039	0.347066
5	18323	0.819130
6	95740	0.377475
7	11116	0.448225

Annexes

° Résultat sur les données test

In [20]:
```
resultat_test = logit.predict(X_testnp)
print(resultat_test)
probability = logit.predict_proba(X_testnp)
p = probability[:,1]
print(p)
numero = X_app.index.values
accord_2 = pd.DataFrame({'Id': numero, 'Resultat': resultat_test})
l=accord_2.to_csv('pred_GiveMeSomeCredit.csv', index=False, sep=',')
tab2_accord = pd.read_csv('pred_GiveMeSomeCredit.csv')
tab2_accord
```

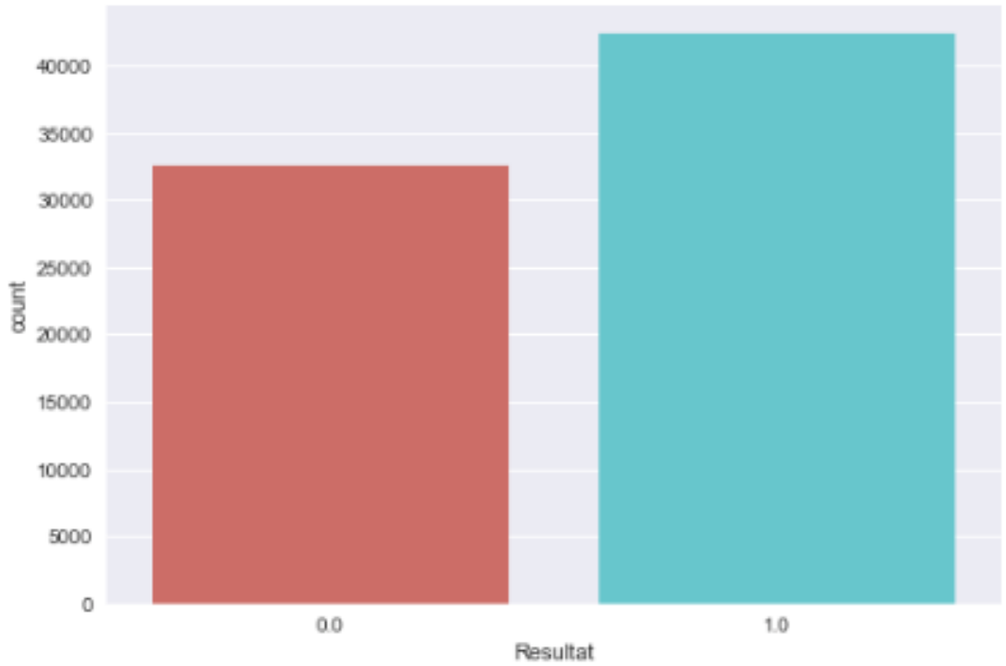
```
[ 1.  0.  0. ...,  1.  1.  1.]
[ 0.55845347  0.45842711  0.4624171  ...,  0.61348928  0.62488068
  0.57787508]
```

Out[20]:

	Id	Resultat
0	28047	1.0
1	35766	0.0
2	63982	0.0
3	106436	1.0
4	92039	0.0
5	18323	1.0
6	95740	0.0
7	11116	0.0

° Bilan des résultats

In [21]:
```
sns.countplot(x='Resultat', data=tab2_accord, palette='hls')
plt.show()
```



In [22]:
```
resultat1 = tab2_accord.drop(['Id'],1)
resultat1.describe()
```

Out[22]:

	Resultat
count	75000.000000
mean	0.565253
std	0.495727
min	0.000000
25%	0.000000
50%	1.000000
75%	1.000000
max	1.000000

In [23]:
```
import statsmodels.api as sm
logit_model=sm.Logit(y_appnp,X_appnp)
result=logit_model.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.240463
         Iterations 7
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                75000
Model:                          Logit   Df Residuals:                    74993
Method:                           MLE   Df Model:                            6
Date:                Sun, 10 Jun 2018   Pseudo R-squ.:                0.02425
Time:                        10:29:17   Log-Likelihood:                -18035.
converged:                       True   LL-Null:                       -18483.
                                        LLR p-value:                2.061e-190
==============================================================================
                 coef    std err          z      P>|z|      [95.0% Conf. Int.]
------------------------------------------------------------------------------
x1          5.598e-05   9.63e-05      0.581      0.561      -0.000       0.000
x2             0.0476      0.001     65.417      0.000       0.046       0.049
x3            2.1e-05   1.46e-05      1.436      0.151   -7.66e-06    4.97e-05
x4          4.834e-05   4.45e-06     10.861      0.000    3.96e-05    5.71e-05
x5             0.0141      0.003      4.059      0.000       0.007       0.021
x6            -0.0635      0.015     -4.321      0.000      -0.092      -0.035
x7            -0.0753      0.012     -6.203      0.000      -0.099      -0.051
==============================================================================
```

4. Un second algorithme utilisant la régression logistique

° Création des fonctions nécessaires

In [24]:
```
# Fonction logistique

# f_logistique
def f_logistique(z):
    return 1 / (1 + np.exp(-z))
```

In [25]:
```
# Vraisemblance logarithmique

# f_ll
# descripteurs : matrice (n, p)
# objectifs    : vecteur y (n)
# poids        : vecteur (p)
# return       : réel
def f_ll(descripteurs, objectifs, poids):
    predictions = f_logistique(np.dot(descripteurs, poids))
    return np.sum(objectifs * np.log(predictions) + (1 - objectifs) * np.log(1 - predictions))
```

In [26]:
```
# Gradient de la vraisemblance logarithmique

# gradient_ll
# descripteurs : matrice (n, p)
# objectifs    : vecteur (n)
# poids        : vecteur (p)
# return       : vecteur (p)
def gradient_ll(descripteurs, objectifs, poids):
    predictions = f_logistique(np.dot(descripteurs, poids))
    return np.dot(descripteurs.T, objectifs - predictions)
```

In [27]:
```
# Matrice hessienne de la vraisemblance logarithmique

# hessian_ll
# descripteurs : matrice (n, p)
# objectifs    : vecteur (n)
# poids        : vecteur (p)
# return       : matrice (p, p)
def hessian_ll(descripteurs, objectifs, poids):
    predictions = f_logistique(np.dot(descripteurs, poids))
    diagonale = np.diag([predictions[i, 0]*(1-predictions[i, 0]) for i in range (np.shape(predictions)[0])])
    return -np.dot(np.dot(descripteurs.T, diagonale), descripteurs)
```

Annexes

° Algorithme de Newton-Raphson

In [28]:
```
import time # Surveiller le temps nécessaire à la régression logistique

# Régression logistique (avec méthode de Newton)

# regression_logistique
# descripteurs : matrice (n, p)
# objectifs    : vecteur (n)
# delta        : réel, différence
# max_iters    : entier, nombre maximum d'itérations possibles
# x0           : booléen, ajoute aux descripteurs l'ordonnée à l'origine (intercept)
# return       : vecteur p
def regression_logistique(descripteurs, objectifs, delta = 1e-15, max_iters = 50, x0 = True):
    if x0:
        descripteurs = np.hstack((np.ones((descripteurs.shape[0], 1)), descripteurs))
    poids = np.array([np.zeros(descripteurs.shape[1], dtype = np.float64)]).T
    delta_ll = np.Infinity
    ll = f_ll(descripteurs, objectifs, poids)
    i = 0

    tps = time.time()
    liste_ll = [ll]
    liste_delta_ll = []

    while np.abs(delta_ll) > delta and i < max_iters:
        i += 1
        gradient = gradient_ll(descripteurs, objectifs, poids)
        hessian = hessian_ll(descripteurs, objectifs, poids)
        try:
            hessian_inv = np.linalg.inv(hessian)
        except np.linalg.linalg.LinAlgError:
            print("Oups ! La matrice hessian n'est pas inversible !")
            break
        delta_poids = -np.dot(hessian_inv, gradient)
        poids += delta_poids
        ll_nouveau = f_ll(descripteurs, objectifs, poids)
        delta_ll = ll - ll_nouveau
        ll = ll_nouveau

        liste_ll.append(ll)
        liste_delta_ll.append(delta_ll)
```

```
delta_tps = np.int(time.time() - tps)
plt.plot(range(len(liste_ll)), liste_ll, label = "ll")
plt.plot(range(1, len(liste_delta_ll) + 1), liste_delta_ll, label = "delta ll")
plt.legend()
plt.show()
print("Régression logistique effectuée en {} secondes avec {} itérations.".format(delta_tps, i))

scores = np.dot(descripteurs, poids)
resultats = np.round(f_logistique(scores))
score = (resultats == objectifs).sum().astype(float) / len(scores) * 100
print("Précision de la régression : {} % (delta_ll = {})".format(score, delta_ll))

return poids
```

° Résultat sur les données d'apprentissage

In [29]:
```
coefficients=regression_logistique(X_appnp, y_appnp)
print(coefficients)
resultat_app_1 = f_logistique(np.dot(X_appnp,coefficients[1:]))
print(resultat_app_1)
resultat_app_1.shape
```

```
Précision de la régression : 93.276 %
[[  1.13295830e+00]
 [  3.67773620e-05]
 [  2.80345634e-02]
 [  1.46334297e-05]
 [  2.08284984e-05]
 [  5.14470694e-03]
 [ -2.23641813e-02]
 [ -1.04944123e-01]]
[[ 0.85742939]
 [ 0.79480016]
 [ 0.86128069]
 ...,
 [ 0.87060075]
 [ 0.89778136]
 [ 0.78154739]]
```

Out[29]: (75000, 1)

° Résultat pour la deuxième partie des données d'apprentissage

In [30]:
```
coefficients=regression_logistique(X_appnp, y_appnp)
print(coefficients)
resultat_test_2 = f_logistique(np.dot(X_testnp,coefficients[1:]))
print(resultat_test_2)
resultat_test_2.shape
```

```
Précision de la régression : 93.276 %
[[  1.13295830e+00]
 [  3.67773620e-05]
 [  2.80345634e-02]
 [  1.46334297e-05]
 [  2.08284984e-05]
 [  5.14470694e-03]
 [ -2.23641813e-02]
 [ -1.04944123e-01]]
[[ 0.84197543]
 [ 0.78688457]
 [ 0.78927118]
 ...,
 [ 0.86028663]
 [ 0.86799667]
 [ 0.84919524]]
```

5. Exploitation des résultats

A) Pour sklearn

In [31]:
```
from sklearn.metrics import confusion_matrix
```

° Sur les données d'apprentissage

In [32]:
```
cm = confusion_matrix(y_appnp, tab_accord['Resultat'])
print('Matrice de confusion')
print(cm)
```

```
Matrice de confusion
[[ 3149  1894]
 [29464 40493]]
```

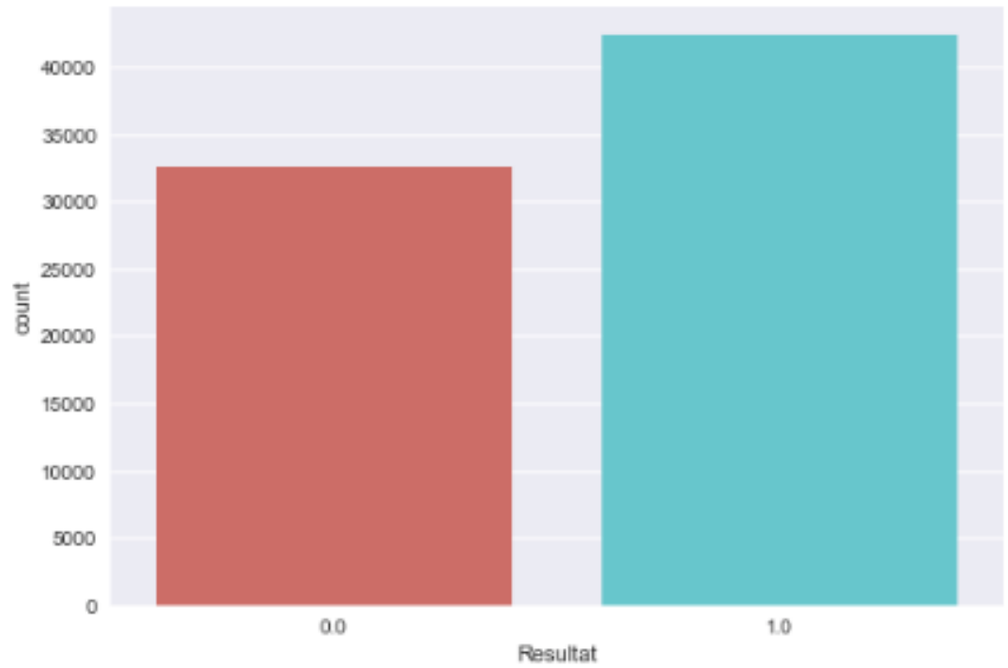
In [33]:
```
tn, fp, fn, tp = cm.ravel()

a = (fp + tp)/750
b = fp/750
c = fn/750
print('% de prets accordes:',round(a,2),'%')
print('% de prets accordes a des mauvais payeurs: ', round(b,2), '%')
print('% de prets non accordes a des bons payeurs: ', round(c,2), '%')
```

```
('% de prets accordes:', 56.52, '%')
('% de prets accordes a des mauvais payeurs: ', 2.53, '%')
('% de prets non accordes a des bons payeurs: ', 39.29, '%')
```

Annexes

° Sur les données test

In [34]:
```
sns.countplot(x='Resultat', data=tab2_accord, palette='hls')
plt.show()
```



In [35]:
```
cm2 = confusion_matrix(y_testnp, tab2_accord['Resultat'])
print('Matrice de confusion')
print(cm2)
```

```
Matrice de confusion
[[ 3093  1890]
 [29513 40504]]
```

In [36]:
```
tn, fp, fn, tp = cm2.ravel()
a = (fp + tp)/750
b = fp/750
c = fn/750
print("% de prets accordes: ", round(a,2), "%")
print('% de prets accordes a des mauvais payeurs: ', round(b,2), '%')
print('% de prets non accordes a des bons payeurs: ', round(c,2), '%')
```

```
('% de prets accordes: ', 56.53, '%')
('% de prets accordes a des mauvais payeurs: ', 2.52, '%')
('% de prets non accordes a des bons payeurs: ', 39.35, '%')
```

B) Avec Newton

In [38]:
```
def valeurs_inferieures(matrice_colonne, valeur_limite):
    L=[]
    for nombre in matrice_colonne :
        if nombre[0] < valeur_limite :
            L.append(nombre[0])
    return(L)
# Règle de décision à 0,7
a = valeurs_inferieures(resultat_test_2, 0.70)
print('il y a', len(a), 'credits non accordes')
# Règle de décision à 0,61
b = valeurs_inferieures(resultat_test_2, 0.61)
print('il y a', len(b), 'credits non accordes')
# Règle de décision à 0,5
c = valeurs_inferieures(resultat_test_2, 0.5)
print('il y a', len(c), 'credits non accordes')

# Recherche d'une règle de décision sur données d'apprentissage
print(len(valeurs_inferieures(resultat_app_1, 0.703)))

# Application de cette règle sur les données test
print(len(valeurs_inferieures(resultat_test_2, 0.703)))
```

```
('il y a', 4495, 'credits non accordes')
('il y a', 22, 'credits non accordes')
('il y a', 1, 'credits non accordes')
4975
4971
```