Lending Club Data Base

Les données traitées ici concernent la société américaine Lending Club, qui accordait des prêts personnels en ligne, sans garantie, d'un montant minimum de 1 000 dollars jusqu'à 40 000 dollars. Cette base de données est traitée sous la direction du PhD Carlos M. disponible sur la page kaggel https://www.kaggle.com/datasets/adarshsng/lending-club-loan-data-csv

Import Bibliothèques

In [3]:

```
import numpy as np
import pandas as pd
```

Exploration Data

In [10]:

```
df_copy.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 466285 entries, 0 to 466284
Data columns (total 75 columns):
 #   Column                       Non-Null Count   Dtype  
---  ------                       --------------   -----  
 0   Unnamed: 0                   466285 non-null  int64  
 1   id                           466285 non-null  int64  
 2   member_id                    466285 non-null  int64  
 3   loan_amnt                    466285 non-null  int64  
 4   funded_amnt                  466285 non-null  int64  
 5   funded_amnt_inv              466285 non-null  float64
 6   term                         466285 non-null  object 
 7   int_rate                     466285 non-null  float64
 8   installment                  466285 non-null  float64
 9   grade                        466285 non-null  object 
 10  sub_grade                    466285 non-null  object 
 11  emp_title                    438697 non-null  object 
 12  emp_length                   445277 non-null  object 
 13  home_ownership               466285 non-null  object 
 14  annual_inc                   466281 non-null  float64
 15  verification_status          466285 non-null  object 
 16  issue_d                      466285 non-null  object 
 17  loan_status                  466285 non-null  object 
 18  pymnt_plan                   466285 non-null  object 
 19  url                          466285 non-null  object 
 20  desc                         125983 non-null  object 
 21  purpose                      466285 non-null  object 
 22  title                        466265 non-null  object 
 23  zip_code                     466285 non-null  object 
 24  addr_state                   466285 non-null  object 
 25  dti                          466285 non-null  float64
 26  delinq_2yrs                  466256 non-null  float64
 27  earliest_cr_line             466256 non-null  object 
 28  inq_last_6mths               466256 non-null  float64
 29  mths_since_last_delinq       215934 non-null  float64
 30  mths_since_last_record       62638 non-null   float64
 31  open_acc                     466256 non-null  float64
 32  pub_rec                      466256 non-null  float64
 33  revol_bal                    466285 non-null  int64  
 34  revol_util                   465945 non-null  float64
 35  total_acc                    466256 non-null  float64
 36  initial_list_status          466285 non-null  object 
 37  out_prncp                    466285 non-null  float64
 38  out_prncp_inv                466285 non-null  float64
 39  total_pymnt                  466285 non-null  float64
 40  total_pymnt_inv              466285 non-null  float64
 41  total_rec_prncp              466285 non-null  float64
 42  total_rec_int                466285 non-null  float64
 43  total_rec_late_fee           466285 non-null  float64
 44  recoveries                   466285 non-null  float64
 45  collection_recovery_fee      466285 non-null  float64
 46  last_pymnt_d                 465909 non-null  object 
 47  last_pymnt_amnt              466285 non-null  float64
 48  next_pymnt_d                 239071 non-null  object 
 49  last_credit_pull_d           466243 non-null  object 
 50  collections_12_mths_ex_med   466140 non-null  float64
 51  mths_since_last_major_derog  98974 non-null   float64
 52  policy_code                  466285 non-null  int64  
 53  application_type             466285 non-null  object 
 54  annual_inc_joint             0 non-null       float64
 55  dti_joint                    0 non-null       float64
 56  verification_status_joint    0 non-null       float64
 57  acc_now_delinq               466256 non-null  float64
 58  tot_coll_amt                 396009 non-null  float64
 59  tot_cur_bal                  396009 non-null  float64
 60  open_acc_6m                  0 non-null       float64
 61  open_il_6m                   0 non-null       float64
 62  open_il_12m                  0 non-null       float64
 63  open_il_24m                  0 non-null       float64
 64  mths_since_rcnt_il           0 non-null       float64
 65  total_bal_il                 0 non-null       float64
 66  il_util                      0 non-null       float64
 67  open_rv_12m                  0 non-null       float64
 68  open_rv_24m                  0 non-null       float64
 69  max_bal_bc                   0 non-null       float64
 70  all_util                     0 non-null       float64
 71  total_rev_hi_lim             396009 non-null  float64
 72  inq_fi                       0 non-null       float64
 73  total_cu_tl                  0 non-null       float64
 74  inq_last_12m                 0 non-null       float64
dtypes: float64(46), int64(7), object(22)
memory usage: 266.8+ MB
```

Variables pour le Modèle PD

Définition de la Variable Dépendante : Bons et des Mauvais (ceux qui ont un retard de paiement de plus de 120 jours)

In [51]:
```
df_concat['loan_status'].unique()
#Mauvais:'Charged Off','Default',Does not meet the credit policy. Status:Charged Off'
```

Out[51]:
```
array(['Fully Paid', 'Charged Off', 'Current', 'Default',
       'Late (31-120 days)', 'In Grace Period', 'Late (16-30 days)',
       'Does not meet the credit policy. Status:Fully Paid',
       'Does not meet the credit policy. Status:Charged Off'],
      dtype=object)
```

In [52]:
```
df_concat['loan_status'].value_counts()
```

Out[52]:
```
Current                                                224226
Fully Paid                                             184739
Charged Off                                             42475
Late (31-120 days)                                       6900
In Grace Period                                          3146
Does not meet the credit policy. Status:Fully Paid       1988
Late (16-30 days)                                        1218
Default                                                   832
Does not meet the credit policy. Status:Charged Off       761
Name: loan_status, dtype: int64
```

In [53]:
```
df_concat['loan_status'].value_counts() / df_concat['loan_status'].count()
```

Out[53]:
```
Current                                                0.480878
Fully Paid                                             0.396193
Charged Off                                            0.091092
Late (31-120 days)                                     0.014798
In Grace Period                                        0.006747
Does not meet the credit policy. Status:Fully Paid     0.004263
Late (16-30 days)                                      0.002612
Default                                                0.001784
Does not meet the credit policy. Status:Charged Off    0.001632
Name: loan_status, dtype: float64
```

Charged Off: 0.091092

Default: 0.001784

Does not meet the credit policy. Status:Charged Off: 0.001632

Notre variable dépendante sera binaire

In [54]:
```
#Nous attribuerons 1 à Bon et 0 à Mauvais
df_concat['Bonus_malus'] = np.where(df_concat['loan_status'].isin(['Charged Off', 'Default', 'Does not meet the
```

In [55]:
```
df_concat['Bonus_malus']
```

Out[55]:
```
0         1
1         0
2         1
3         1
4         1
         ..
466280    1
466281    0
466282    1
466283    1
466284    1
Name: Bonus_malus, Length: 466285, dtype: int32
```

Partition de la base

In [56]:
```
from sklearn.model_selection import train_test_split
```

train_test_split(df_concat.drop('Bonus_malus', axis = 1), df_concat['Bonus_malus'])

In [58]:
```
x_train, x_test, y_train, y_test = train_test_split(df_concat.drop('Bonus_malus', axis = 1), df_concat['Bonus_m
```

In [59]:
```
x_train.shape
```

Out[59]:
```
(419656, 207)
```

In [60]:
```
x_test.shape
```

Out[60]:
```
(46629, 207)
```

In [61]:
```
y_train.shape
```

Out[61]:
```
(419656,)
```

In [62]:
```
y_test.shape
```

Out[62]: (46629,)

Traitement des variables catégorielles

Poids de l'évidence (PdE)

Nous calculons le poids de l'évidence dans le but de regrouper des catégories PdE similaires.

Pour ce faire, nous commencerons par estimer le poids de la preuve (PDE) PdE = ln(%Bons / %Mauvais).

Le poids de l'évidence **indique la puissance prédictive d'une variable indépendante (home_ownership) par rapport à la variable dépendante (bons_mauvais)**.

PdE est décrit comme une mesure de la séparation entre les bons et les mauvais clients.

"Clients mauvais" se réfère aux clients qui ont fait défaut sur un prêt, et "Bons clients" se réfère aux clients qui ont remboursé le prêt.

Tout cela est dû au fait que les catégories avec un PdE similaire ont presque la même proportion de bons et de mauvais. En d'autres termes, le comportement des deux catégories est le même.

In [63]:
```
# La variable home_ownership indica le type de logement que possède la personne
```

In [64]:
```
x_train['home_ownership'].unique()
```

Out[64]:
```
array(['MORTGAGE', 'RENT', 'OWN', 'NONE', 'OTHER', 'ANY'], dtype=object)
```

In [65]:
```
# df notre variable indépendante et dépendante
```

In [66]:
```
df = pd.concat([x_train['home_ownership'], y_train], axis = 1)
```

In [67]:
```
df.head()
```

Out[67]:

	home_ownership	Bonus_malus
344531	MORTGAGE	1
328300	MORTGAGE	1
299890	MORTGAGE	1
439226	MORTGAGE	1
167889	MORTGAGE	0

In [68]:
```
# les observations par catégorie
```

In [69]:
```
df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].count() #values[0]=home_owership. comm
```

Out[69]:

	home_ownership	Bonus_malus
0	ANY	1
1	MORTGAGE	212191
2	NONE	44
3	OTHER	168
4	OWN	37465
5	RENT	169787

In [70]:
```
# Les observations par type de logement seront ensuite utilisées pour calculer la moyenne (mean)
```

In [71]:
```
df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].mean()
```

Out[71]:

	home_ownership	Bonus_malus
0	ANY	1.000000
1	MORTGAGE	0.917315
2	NONE	0.818182
3	OTHER	0.779762
4	OWN	0.907754
5	RENT	0.890215

MORTGAGE 0.917315-->91% sont bons (1)

In [72]:
```
# Concatenation des 2 tableaux
```

In [73]:
```
df1 = pd.concat([df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].count(),
                 df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].mean()], axis=1)
```

In [74]:
```
df1
```

Out[74]:

	home_ownership	Bonus_malus	home_ownership	Bonus_malus
0	ANY	1	ANY	1.000000
1	MORTGAGE	212191	MORTGAGE	0.917315
2	NONE	44	NONE	0.818182
3	OTHER	168	OTHER	0.779762
4	OWN	37465	OWN	0.907754
5	RENT	169787	RENT	0.890215

In [75]:
```
df1 = df1.iloc[:, [0,1,3]] #toutes les lignes et les colonnes 0,1,3
```

In [76]:
```
df1
```

Out[76]:

	home_ownership	Bonus_malus	Bonus_malus
0	ANY	1	1.000000
1	MORTGAGE	212191	0.917315
2	NONE	44	0.818182
3	OTHER	168	0.779762
4	OWN	37465	0.907754
5	RENT	169787	0.890215

In [77]:
```
df1.columns = [df1.columns.values[0], 'observations', '%_bon']
```

In [78]:
```
df1
```

Out[78]:

	home_ownership	observations	%_bon
0	ANY	1	1.000000
1	MORTGAGE	212191	0.917315
2	NONE	44	0.818182
3	OTHER	168	0.779762
4	OWN	37465	0.907754
5	RENT	169787	0.890215

Estimons la proportion de chaque catégorie dans la base de données

C'est-à-dire, le nombre d'observations de chaque catégorie divisé par le total des observations

In [79]:
```
df1['%_obs'] = df1['observations']/df1['observations'].sum()
```

In [80]:
```
df1
```

Out[80]:

	home_ownership	observations	%_bon	%_obs
0	ANY	1	1.000000	0.000002
1	MORTGAGE	212191	0.917315	0.505631
2	NONE	44	0.818182	0.000105
3	OTHER	168	0.779762	0.000400
4	OWN	37465	0.907754	0.089276
5	RENT	169787	0.890215	0.404586

Estimons le nombre de bons et de mauvais par catégorie

Bons = prop_bueno *observations Malos = (1 - prop_bueno)* observations

In [81]:
```
df1['n_bons'] = df1['%_bon']*df1['observations']
```

In [82]:
```
df1['n_malus'] = (1-df1['%_bon'])*df1['observations']
df1
```

Out[82]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus
0	ANY	1	1.000000	0.000002	1.0	0.0
1	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0
2	NONE	44	0.818182	0.000105	36.0	8.0
3	OTHER	168	0.779762	0.000400	131.0	37.0
4	OWN	37465	0.907754	0.089276	34009.0	3456.0
5	RENT	169787	0.890215	0.404586	151147.0	18640.0

In [83]:
```
# proportion de bons et malus par catégorie
```

In [84]:
```
df1['%_n_bons'] = df1['n_bons']/df1['n_bons'].sum()
```

In [85]:
```
df1['%_n_malus'] = df1['n_malus']/df1['n_malus'].sum()
```

In [86]:
```
df1
```

Out[86]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus
0	ANY	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000
1	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0	0.512267	0.442095
2	NONE	44	0.818182	0.000105	36.0	8.0	0.000095	0.000202
3	OTHER	168	0.779762	0.000400	131.0	37.0	0.000345	0.000932
4	OWN	37465	0.907754	0.089276	34009.0	3456.0	0.089504	0.087084
5	RENT	169787	0.890215	0.404586	151147.0	18640.0	0.397787	0.469687

In [87]:
```
# PdE = ln(%Bons/%Malus) par catégorie
```

In [88]:
```
df1['PdE'] = np.log(df1['%_n_bons']/df1['%_n_malus'])
```

In [89]:
```
df1
```

Out[89]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE
0	ANY	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	inf
1	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0	0.512267	0.442095	0.147320
2	NONE	44	0.818182	0.000105	36.0	8.0	0.000095	0.000202	-0.755016
3	OTHER	168	0.779762	0.000400	131.0	37.0	0.000345	0.000932	-0.994814
4	OWN	37465	0.907754	0.089276	34009.0	3456.0	0.089504	0.087084	0.027420
5	RENT	169787	0.890215	0.404586	151147.0	18640.0	0.397787	0.469687	-0.166151

In [90]:
```
# classons-le en fonction du PdE du plus bas au plus élevé

# Reset les indices
```

In [91]:
```
df1 = df1.sort_values(['PdE'])
df1 = df1.reset_index(drop = True)
df1
```

Out[91]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE
0	OTHER	168	0.779762	0.000400	131.0	37.0	0.000345	0.000932	-0.994814
1	NONE	44	0.818182	0.000105	36.0	8.0	0.000095	0.000202	-0.755016
2	RENT	169787	0.890215	0.404586	151147.0	18640.0	0.397787	0.469687	-0.166151
3	OWN	37465	0.907754	0.089276	34009.0	3456.0	0.089504	0.087084	0.027420
4	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0	0.512267	0.442095	0.147320
5	ANY	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	inf

Catégorie Other avec la majeur probabilité de defaut

Valeur de la Information (VI)

L'analyse de la valeur d'information est une technique d'exploration de données qui aide à déterminer **quelles variables d'un ensemble de données ont un pouvoir prédictif ou une influence** sur la valeur d'une variable dépendante spécifique.

Notons que nous **n'évaluons pas le poids des catégories contenues dans la variable, mais plutôt de la variable elle-même**. cela veut dire la valeur globale de la variable

<2% le variable n'est pas un bon predicteur

2% à 10% predicteur faible

10% à 30% moyen

30% à 50% très fort

plus de 50% c'est pas normal

VI = Somme (PdE * (proportion des bons dans la catégorie - proportion des mauvais dans la catégorie))

In [92]:
```
df1['VI'] = df1['PdE']*(df1['%_n_bons'] - df1['%_n_malus'])
df1['somme_VI'] = df1['VI'].sum()
```

In [93]:
```
df1
```

Out[93]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	VI	somme_VI
0	OTHER	168	0.779762	0.000400	131.0	37.0	0.000345	0.000932	-0.994814	0.000585	inf
1	NONE	44	0.818182	0.000105	36.0	8.0	0.000095	0.000202	-0.755016	0.000081	inf
2	RENT	169787	0.890215	0.404586	151147.0	18640.0	0.397787	0.469687	-0.166151	0.011946	inf
3	OWN	37465	0.907754	0.089276	34009.0	3456.0	0.089504	0.087084	0.027420	0.000066	inf
4	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0	0.512267	0.442095	0.147320	0.010338	inf
5	ANY	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	inf	inf	inf

In [94]:
```
# On enleve la catégorie ANY pour calculer la somme de la VI
```

In [95]:
```
df2 = df1.loc[[0,1,2,3,4], :]
```

In [96]:
```
df2
```

Out[96]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	VI	somme_VI
0	OTHER	168	0.779762	0.000400	131.0	37.0	0.000345	0.000932	-0.994814	0.000585	inf
1	NONE	44	0.818182	0.000105	36.0	8.0	0.000095	0.000202	-0.755016	0.000081	inf
2	RENT	169787	0.890215	0.404586	151147.0	18640.0	0.397787	0.469687	-0.166151	0.011946	inf
3	OWN	37465	0.907754	0.089276	34009.0	3456.0	0.089504	0.087084	0.027420	0.000066	inf
4	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0	0.512267	0.442095	0.147320	0.010338	inf

In [97]:
```
# Calcul de la VI
```

In [98]:
```
df2['sommeVI'] = df2['VI'].sum()
```

In [99]:
```
df2
```

Out[99]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	VI	somme_VI	sommeVI
0	OTHER	168	0.779762	0.000400	131.0	37.0	0.000345	0.000932	-0.994814	0.000585	inf	0.023015
1	NONE	44	0.818182	0.000105	36.0	8.0	0.000095	0.000202	-0.755016	0.000081	inf	0.023015
2	RENT	169787	0.890215	0.404586	151147.0	18640.0	0.397787	0.469687	-0.166151	0.011946	inf	0.023015
3	OWN	37465	0.907754	0.089276	34009.0	3456.0	0.089504	0.087084	0.027420	0.000066	inf	0.023015
4	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0	0.512267	0.442095	0.147320	0.010338	inf	0.023015

Automatización du Preprocessus des Variables Catégoriques

Définissons la fonction PdE_categorica qui automatisera le processus précédent pour calculer le PdE et VI pour toute variable dépendante (catégorique) et indépendante.

In [100…
```
#tous les pas faites avant, on les met dans cette focntion
def PdE_catégorique(df, var_categ, df_var_y):
    df = pd.concat([df[var_categ], df_var_y], axis = 1)
    df = pd.concat([df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].count(),
                    df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].mean()], axis=1)
    df = df.iloc[:, [0,1,3]]
    df.columns = [df.columns.values[0], 'observations', '%_bon']
    df['%_obs'] = df['observations']/df['observations'].sum()
    df['n_bons'] = df['%_bon']*df['observations']
```

```
    df['n_malus'] = (1-df['%_bon'])*df['observations']
    df['%_n_bons'] = df['n_bons']/df['n_bons'].sum()
    df['%_n_malus'] = df['n_malus']/df['n_malus'].sum()
    df['PdE'] = np.log(df['%_n_bons']/df['%_n_malus'])
    df = df.sort_values(['PdE'])
    df = df.reset_index(drop = True)
    df['delta_%_n_bons'] = df['%_n_bons'].diff().abs()
    df['delta_PdE'] = df['PdE'].diff().abs() #Diferencia Absoluta entre categorías
    df['VI'] = (df['%_n_bons'] - df['%_n_malus']) * df['PdE']
    df.replace([np.inf, -np.inf], np.nan, inplace=True) #np ignore les nan mais pas les inf, alors comme ca on
    df['sommeVI'] = df['VI'].sum()
    return df
```

In [101…
```
df_PdE = PdE_catégorique(x_train, 'home_ownership', y_train)
```

In [102…
```
df_PdE
```

Out[102]:

	home_ownership	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	
0	OTHER	168	0.779762	0.000400	131.0	37.0	0.000345	0.000932	-0.994814	NaN	NaN	0
1	NONE	44	0.818182	0.000105	36.0	8.0	0.000095	0.000202	-0.755016	0.000250	0.239798	0
2	RENT	169787	0.890215	0.404586	151147.0	18640.0	0.397787	0.469687	-0.166151	0.397692	0.588866	0
3	OWN	37465	0.907754	0.089276	34009.0	3456.0	0.089504	0.087084	0.027420	0.308282	0.193570	0
4	MORTGAGE	212191	0.917315	0.505631	194646.0	17545.0	0.512267	0.442095	0.147320	0.422762	0.119900	0
5	ANY	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.512264	NaN	

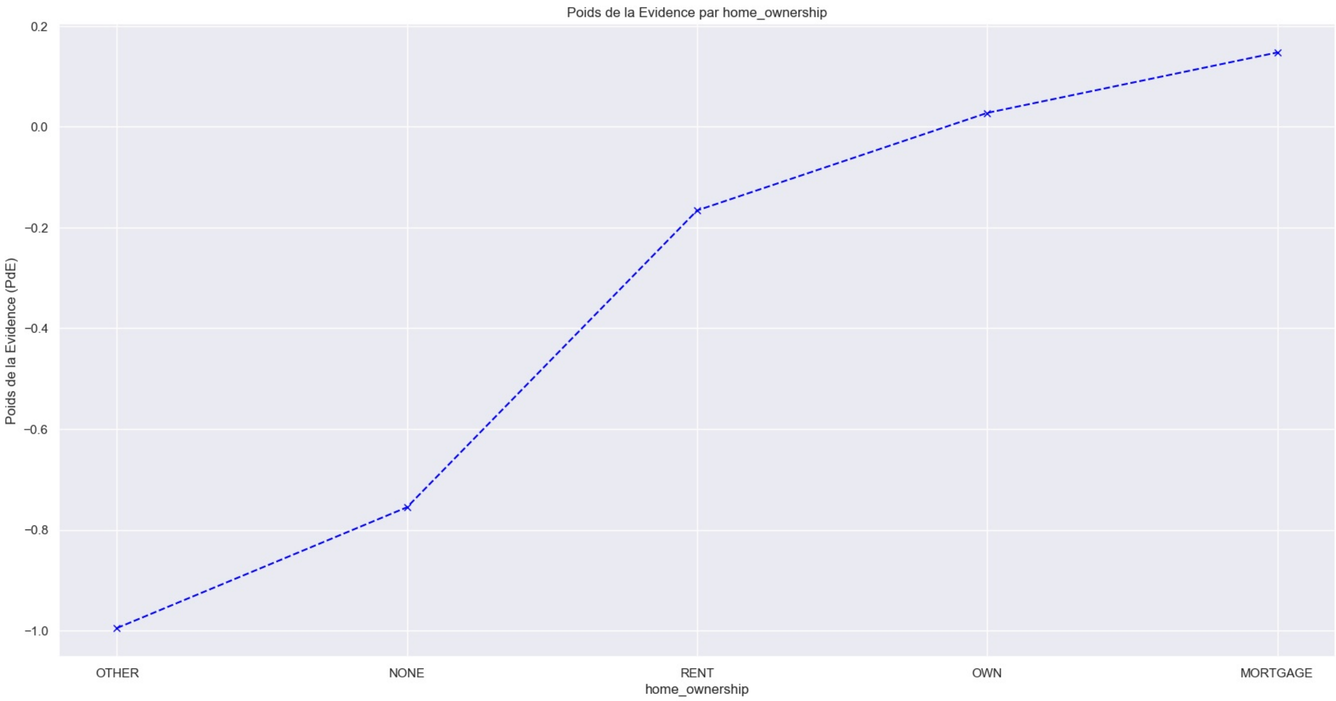
In [103…
```
# Graph la variable Catégorique Indépendante vs. PdE
```

In [104…
```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
```

In [105…
```
# X (la variable catégorique indépendante) et Y (PdE)
```

In [106…
```
def graph_PdE(df):
    x = np.array(df.iloc[:,0].apply(str)) #0=   home_ownership
    y = df['PdE']
    plt.figure(figsize = (20,10))
    plt.plot(x, y, marker = 'x', linestyle = '--', color = 'blue')
    plt.xlabel(df.columns[0])
    plt.ylabel('Poids de la Evidence (PdE)')
    plt.title(str('Poids de la Evidence par ' + df.columns[0]))
    #plt.xticks(rotation=90)
```

In [107…
```
graph_PdE(df1)
```



Le poids de l'évidence est utilisé pour définir les catégories à regrouper ayant un niveau prédictif similaire. Dans cette logique, nous allons regrouper les catégories Other, None, Rent et Any qui possèdent un niveau prédictif similaire.

Creation des Dummies

In [110…
```
# variable dummie Casa_RENT_ANY_OTHER_NONE possèdent un niveau prédictif similaire
```

In [111…
```
x_train['Home_RENT_ANY_OTHER_NONE'] = sum([x_train['Home_RENT'],
                                            x_train['Home_ANY'],
                                            x_train['Home_OTHER'],
                                            x_train['Home_NONE']])
```

In [112…
```
x_train['Home_RENT_ANY_OTHER_NONE']
```

Out[112]:
```
344531    0
328300    0
299890    0
439226    0
167889    0
         ..
239305    0
167080    0
177337    1
23587     1
160385    0
Name: Home_RENT_ANY_OTHER_NONE, Length: 419656, dtype: uint8
```

In [114…
```
# catégories de 'addr_state'
```

In [115…
```
x_train['addr_state'].unique()
```

Out[115]:
```
array(['RI', 'TN', 'IL', 'WV', 'PA', 'FL', 'KS', 'OR', 'GA', 'TX', 'NY',
       'CT', 'MA', 'VA', 'AZ', 'OH', 'AR', 'CA', 'CO', 'NC', 'MN', 'MO',
       'MI', 'NJ', 'WA', 'DE', 'OK', 'NV', 'IN', 'SC', 'UT', 'LA', 'KY',
       'NM', 'AL', 'MT', 'MD', 'MS', 'HI', 'NH', 'DC', 'WI', 'AK', 'SD',
       'WY', 'VT', 'IA', 'ME', 'NE', 'ID'], dtype=object)
```

In [116…
```
# tableau avec fonction PdE_categorica para la variable addr_state
```

In [117…
```
df = PdE_catégorique(x_train, 'addr_state', y_train)
```

In [118…
```
df
```

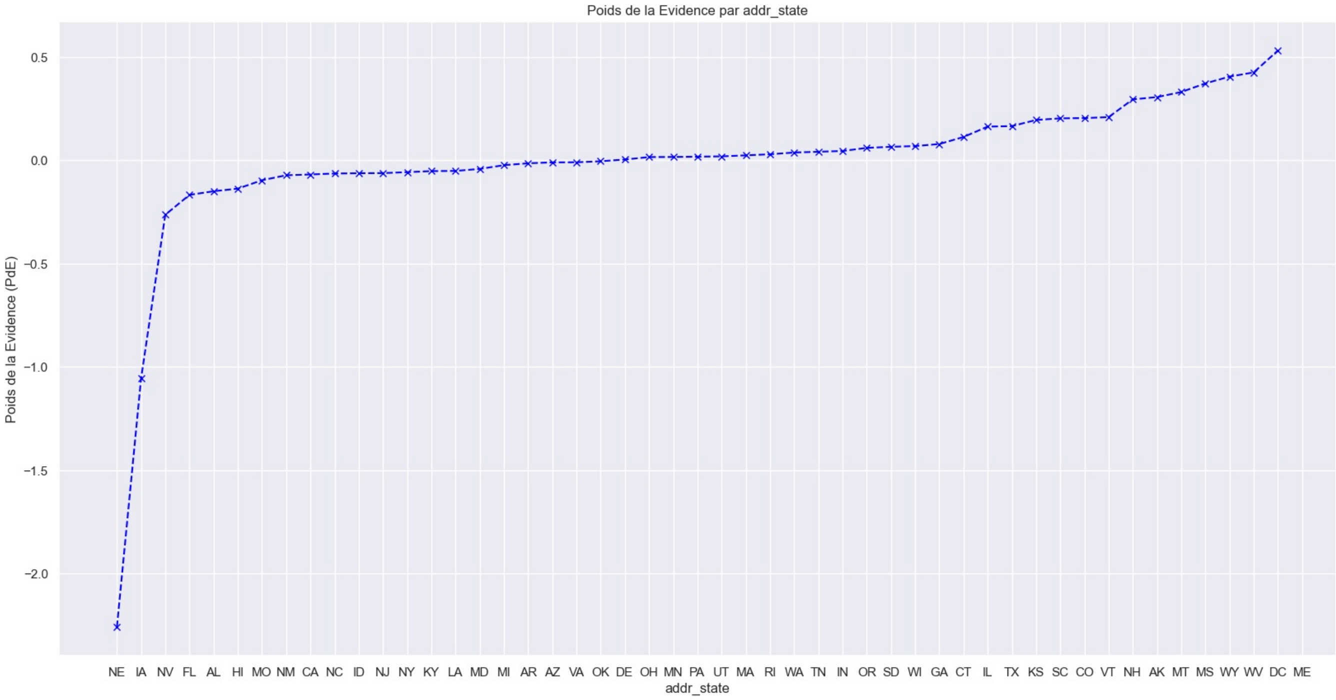
Out[118]:

	addr_state	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	
0	NE	12	0.500000	0.000029	6.0	6.0	0.000016	0.000151	-2.259094	NaN	NaN	3.05872
1	IA	13	0.769231	0.000031	10.0	3.0	0.000026	0.000076	-1.055121	0.000011	1.203973	5.19916
2	NV	5843	0.880370	0.013923	5144.0	699.0	0.013538	0.017613	-0.263158	0.013512	0.791963	1.07246
3	FL	28521	0.890151	0.067963	25388.0	3133.0	0.066816	0.078945	-0.166808	0.053278	0.096350	2.02320
4	AL	5262	0.891866	0.012539	4693.0	569.0	0.012351	0.014338	-0.149147	0.054465	0.017661	2.96291
5	HI	2254	0.893079	0.005371	2013.0	241.0	0.005298	0.006073	-0.136509	0.007053	0.012638	1.05778
6	MO	6761	0.896909	0.016111	6064.0	697.0	0.015959	0.017563	-0.095754	0.010661	0.040755	1.53562
7	NM	2349	0.899106	0.005597	2112.0	237.0	0.005558	0.005972	-0.071763	0.010401	0.023991	2.96774
8	CA	64344	0.899447	0.153326	57874.0	6470.0	0.152312	0.163030	-0.068002	0.146754	0.003762	7.28825
9	NC	11405	0.899868	0.027177	10263.0	1142.0	0.027010	0.028776	-0.063330	0.125302	0.004672	1.11831
10	ID	10	0.900000	0.000024	9.0	1.0	0.000024	0.000025	-0.061869	0.026986	0.001460	9.35291
11	NJ	16209	0.900056	0.038624	14589.0	1620.0	0.038395	0.040820	-0.061252	0.038371	0.000617	1.48555
12	NY	36256	0.900458	0.086395	32647.0	3609.0	0.085920	0.090939	-0.056772	0.047525	0.004481	2.84932
13	KY	3956	0.900910	0.009427	3564.0	392.0	0.009380	0.009878	-0.051717	0.076540	0.005055	2.57472
14	LA	4889	0.901002	0.011650	4405.0	484.0	0.011593	0.012196	-0.050683	0.002213	0.001034	3.05475
15	MD	9842	0.901849	0.023453	8876.0	966.0	0.023360	0.024341	-0.041151	0.011767	0.009532	4.03834
16	MI	10347	0.903450	0.024656	9348.0	999.0	0.024602	0.025173	-0.022931	0.001242	0.018220	1.30858
												1.42657

17	AR	3133	0.904245	0.007466	2833.0	300.0	0.007456	0.007559	-0.013785	0.017146	0.009146	
18	AZ	9688	0.904624	0.023086	8764.0	924.0	0.023065	0.023283	-0.009398	0.015609	0.004387	2.046841
19	VA	12840	0.904673	0.030596	11616.0	1224.0	0.030571	0.030842	-0.008835	0.007506	0.000564	2.396601
20	OK	3709	0.905096	0.008838	3357.0	352.0	0.008835	0.008870	-0.003922	0.021736	0.004913	1.361662
21	DE	1136	0.905810	0.002707	1029.0	107.0	0.002708	0.002696	0.004420	0.006127	0.008342	5.279184
22	OH	13705	0.906822	0.032658	12428.0	1277.0	0.032708	0.032178	0.016345	0.030000	0.011925	8.666741
23	MN	7366	0.906869	0.017552	6680.0	686.0	0.017580	0.017286	0.016902	0.015128	0.000557	4.979995
24	PA	14793	0.906983	0.035250	13417.0	1376.0	0.035311	0.034672	0.018248	0.017730	0.001346	1.165145
25	UT	3111	0.907104	0.007413	2822.0	289.0	0.007427	0.007282	0.019681	0.027884	0.001433	2.848517
26	MA	9991	0.907517	0.023808	9067.0	924.0	0.023862	0.023283	0.024591	0.016436	0.004910	1.425384
27	RI	1858	0.907966	0.004427	1687.0	171.0	0.004440	0.004309	0.029950	0.019423	0.005359	3.923409
28	WA	9449	0.908668	0.022516	8586.0	863.0	0.022597	0.021746	0.038380	0.018157	0.008430	3.265413
29	TN	5394	0.908973	0.012853	4903.0	491.0	0.012904	0.012372	0.042065	0.009693	0.003685	2.235856
30	IN	5864	0.909277	0.013973	5332.0	532.0	0.014033	0.013405	0.045744	0.001129	0.003680	2.870257
31	OR	5353	0.910517	0.012756	4874.0	479.0	0.012827	0.012070	0.060876	0.001205	0.015131	4.611836
32	SD	887	0.910936	0.002114	808.0	79.0	0.002126	0.001991	0.066020	0.010701	0.005145	8.969348
33	WI	5328	0.911224	0.012696	4855.0	473.0	0.012777	0.011919	0.069575	0.010651	0.003555	5.974864
34	GA	13423	0.912017	0.031986	12242.0	1181.0	0.032218	0.029759	0.079417	0.019441	0.009842	1.953450
35	CT	6508	0.914720	0.015508	5953.0	555.0	0.015667	0.013985	0.113589	0.016551	0.034171	1.910840
36	IL	16761	0.918561	0.039940	15396.0	1365.0	0.040519	0.034395	0.163859	0.024852	0.050271	1.003473
37	TX	32845	0.918740	0.078266	30176.0	2669.0	0.079417	0.067253	0.166249	0.038898	0.002390	2.022232
38	KS	3769	0.920934	0.008981	3471.0	298.0	0.009135	0.007509	0.196011	0.070282	0.029761	3.187108
39	SC	5019	0.921498	0.011960	4625.0	394.0	0.012172	0.009928	0.203787	0.003037	0.007776	4.573140
40	CO	8673	0.921596	0.020667	7993.0	680.0	0.021036	0.017135	0.205135	0.008864	0.001348	8.003059
41	VT	807	0.921933	0.001923	744.0	63.0	0.001958	0.001587	0.209812	0.019078	0.004678	7.775394
42	NH	2025	0.927901	0.004825	1879.0	146.0	0.004945	0.003679	0.295795	0.002987	0.085982	3.745493
43	AK	1135	0.928634	0.002705	1054.0	81.0	0.002774	0.002041	0.306805	0.002171	0.011010	2.248514
44	MT	1247	0.930233	0.002971	1160.0	87.0	0.003053	0.002192	0.331173	0.000279	0.024369	2.850289
45	MS	1103	0.932910	0.002628	1029.0	74.0	0.002708	0.001865	0.373184	0.000345	0.042010	3.147698
46	WY	999	0.934935	0.002381	934.0	65.0	0.002458	0.001638	0.405995	0.000250	0.032812	3.330102
47	WV	2162	0.936170	0.005152	2024.0	138.0	0.005327	0.003477	0.426484	0.002869	0.020488	7.887555
48	DC	1298	0.942219	0.003093	1223.0	75.0	0.003219	0.001890	0.532480	0.002108	0.105997	7.075810
49	ME	4	1.000000	0.000010	4.0	0.0	0.000011	0.000000	NaN	0.003208	NaN	N

In [119…

```
graph_PdE(df)
```



In [121…
```python
if ['Adresse_ND'] in x_train.columns.values:
    pass
else:
    x_train['Adresse_ND'] = 0
```

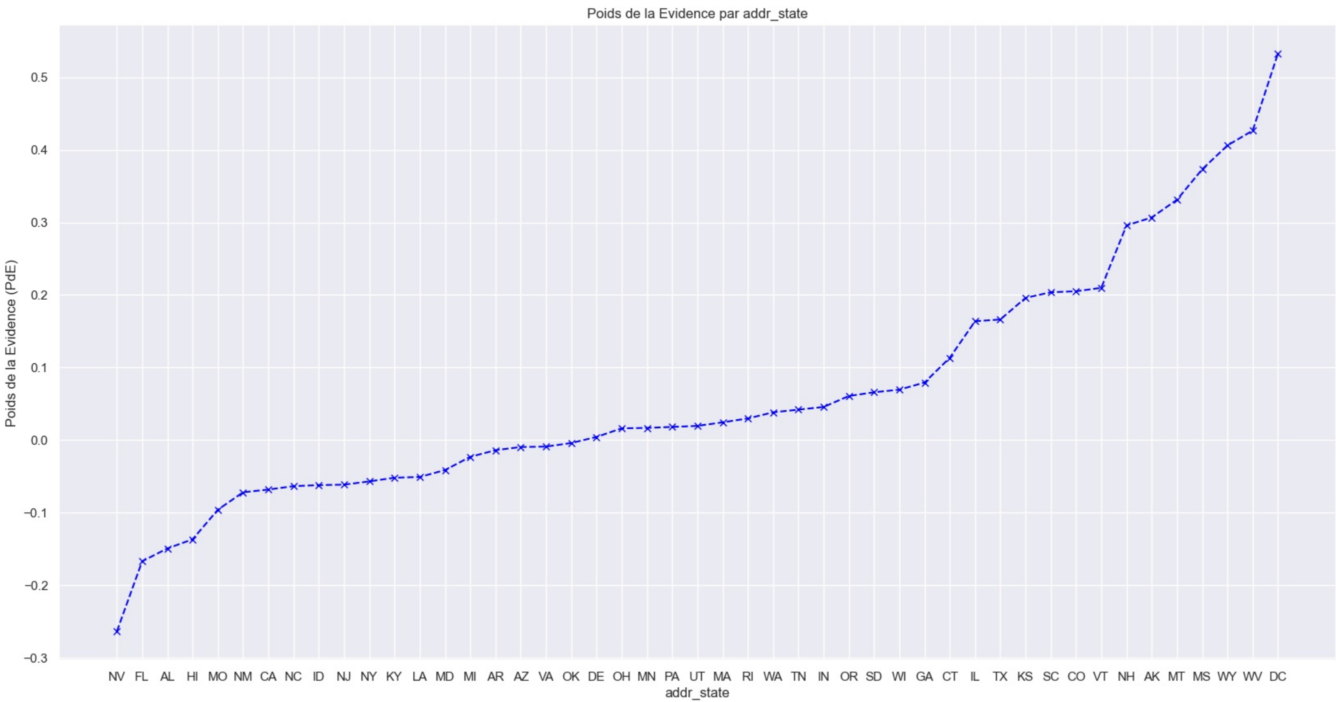
Notons que les deux premières observations ont des valeurs très basses de PdE. Remarquons également que la dernière observation a une valeur Inf de PdE.

Dans les deux cas, le nombre d'observations est très faible, il n'aurait donc pas de sens de les laisser dans des catégories séparées.

Nous les inclurons donc dans les catégories les plus basses et les plus élevées respectivement. Et pour avoir une meilleure perspective des états restants, nous les exclurons de notre graphique.

Graphique suivant: En enlevant la dernière observation (ME) car elle n'a pas de valeur PdE En enlevant les deux premières observations (NE et IA) car elles ont un PdE très bas

In [122…
```python
graph_PdE(df.iloc[2:-1,:])
```



L'idée est de rechercher les États qui ont une proportion d'observations pertinente.

S'il y a environ 50 États, alors en moyenne la proportion d'observations devrait être de 2%. Par conséquent, les États qui ont une proportion significativement plus élevée que 2% sont des candidats pour rester comme catégories individuelles.

En examinant le tableau : nous voyons %_obs CA, NY, FL et TX remplissent cette condition. Ensuite, nous regroupons les catégories restantes en fonction du graphique PdE. Lorsqu'il y a un "saut" significatif, il y aura une coupure.

Les groupes finaux pourraient être :

In [123...
```
# grouper Direccion_ND_NE_IA_NV
```

In [124...
```
x_train['Adresse_ND_NE_IA_NV'] = sum([x_train['Adresse_ND'],
                                                         x_train['Adresse_NE'],
                                                         x_train['Adresse_IA'],
                                                         x_train['Adresse_NV']])
```

In [125...
```
# FL toute seule
# gruper AL HI MO NM
```

In [126...
```
x_train['Adresse_AL_HI_MO_NM'] = sum([x_train['Adresse_AL'],
                                                         x_train['Adresse_HI'],
                                                         x_train['Adresse_MO'],
                                                         x_train['Adresse_NM']])
```

In [127...
```
# California toute seule
# grouper NC ID NJ
```

In [128...
```
x_train['Adresse_NC_ID_NJ'] = sum([x_train['Adresse_NC'],
                                                         x_train['Adresse_ID'],
                                                         x_train['Adresse_NJ']])
```

In [129...
```
# NY toute seule
# grouper KY LA MD
```

In [130...
```
x_train['Adresse_KY_LA_MD'] = sum([x_train['Adresse_KY'],
                                                         x_train['Adresse_LA'],
                                                         x_train['Adresse_MD']])
```

In [131...
```
# grouper MI AR AZ VA OK DE OH
```

In [132...
```
x_train['Adresse_MI_AR_AZ_VA_OK_DE_OH'] = sum([x_train['Adresse_MI'],
                                                         x_train['Adresse_AR'],
                                                         x_train['Adresse_AZ'],
                                                         x_train['Adresse_VA'],
                                                         x_train['Adresse_OK'],
                                                         x_train['Adresse_DE'],
                                                         x_train['Adresse_OH']])
```

In [133...
```
# grouper MN PA UT MA RI WA TN IN
```

In [134...
```
x_train['Adresse_MN_PA_UT_MA_RI_WA_TN_IN'] = sum([x_train['Adresse_MN'],
                                                         x_train['Adresse_PA'],
                                                         x_train['Adresse_UT'],
                                                         x_train['Adresse_MA'],
                                                         x_train['Adresse_RI'],
                                                         x_train['Adresse_WA'],
                                                         x_train['Adresse_TN'],
                                                         x_train['Adresse_IN']])
```

In [135...
```
# grouper OR SD WI GA
```

In [136...
```
x_train['Adresse_OR_SD_WI_GA'] = sum([x_train['Adresse_OR'],
                                                         x_train['Adresse_SD'],
                                                         x_train['Adresse_WI'],
                                                         x_train['Adresse_GA']])
```

In [137...
```
# grouper CT IL
```

In [138...
```
x_train['Adresse_CT_IL'] = sum([x_train['Adresse_CT'],
                                                x_train['Adresse_IL']])
```

In [139...
```
# grouper NH AK MT MS WY WV DC ME
```

In [140...
```
x_train['Adresse_NH_AK_MT_MS_WY_WV_DC_ME'] = sum([x_train['Adresse_NH'],
                                                         x_train['Adresse_AK'],
                                                         x_train['Adresse_MT'],
                                                         x_train['Adresse_MS'],
                                                         x_train['Adresse_WY'],
                                                         x_train['Adresse_WV'],
                                                         x_train['Adresse_DC'],
                                                         x_train['Adresse_ME']])
```

PdE pour la variable verification_status

In [141...
```
x_train['verification_status'].unique()
```

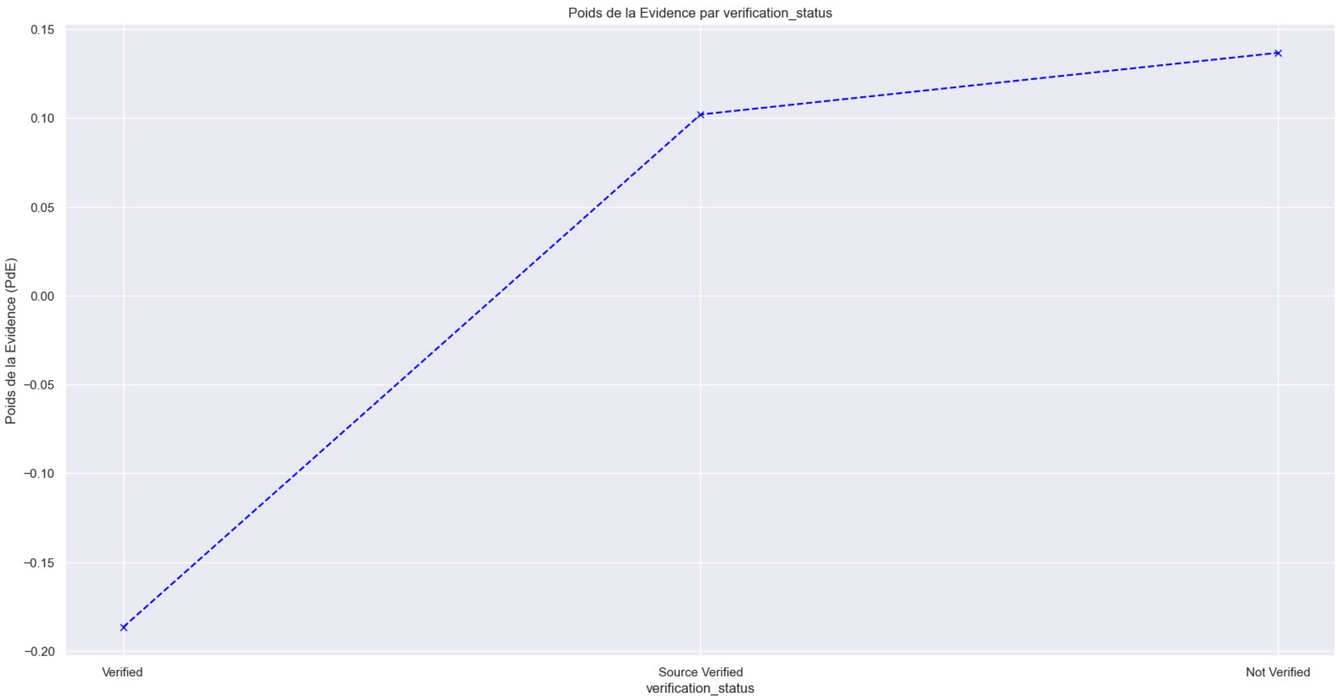
Out[141]: array(['Not Verified', 'Verified', 'Source Verified'], dtype=object)

In [142…
```
df = PdE_catégorique(x_train, 'verification_status', y_train)
df
```

Out[142]:

	verification_status	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE
0	Verified	151284	0.888197	0.360495	134370.0	16914.0	0.353633	0.426196	-0.186638	NaN	NaN
1	Source Verified	135048	0.913808	0.321806	123408.0	11640.0	0.324784	0.293302	0.101955	0.02885	0.288593
2	Not Verified	133324	0.916504	0.317698	122192.0	11132.0	0.321583	0.280502	0.136676	0.00320	0.034721

In [143…
```
graph_PdE(df)
```



La variable verification_status possède trois catégories. Les trois ont une proportion d'observations très similaire (30%) mais un PdE très différente. Pour cette raison, nous décidons de ne pas regrouper les catégories.

purpose

In [145…
```
x_train['purpose'].unique()
```

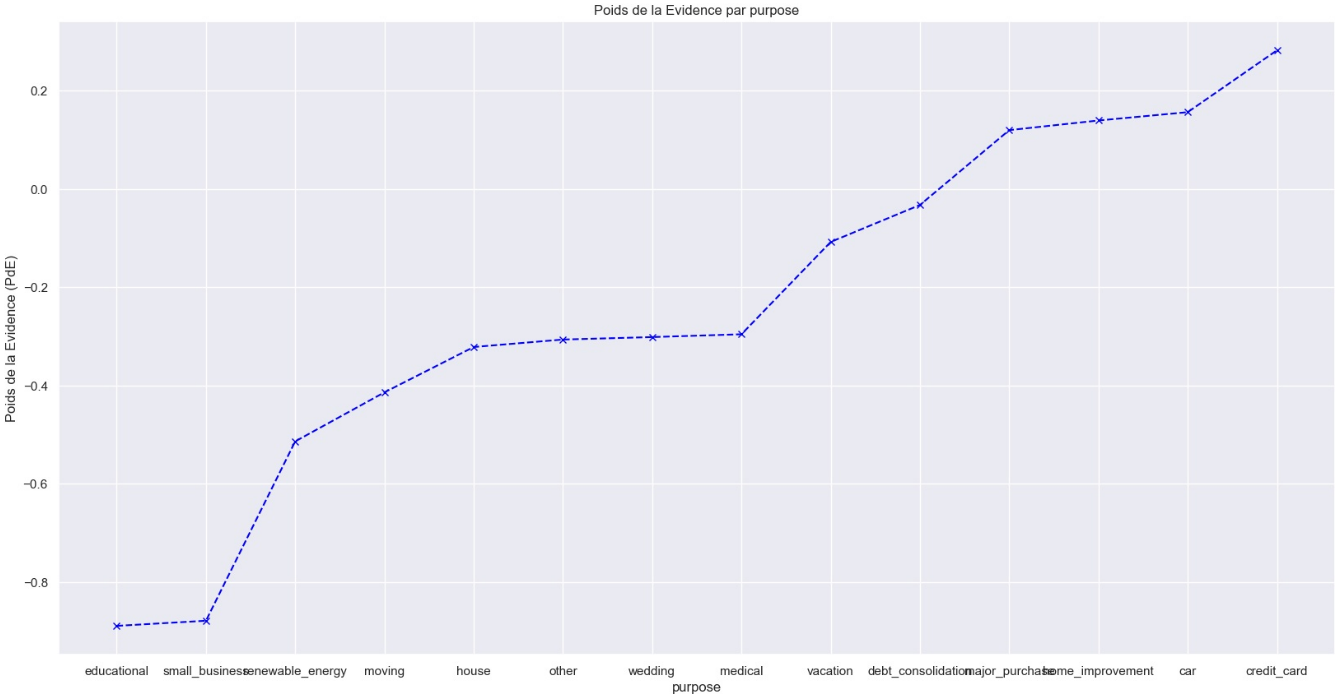
Out[145]:
```
array(['debt_consolidation', 'credit_card', 'other', 'home_improvement',
       'major_purchase', 'small_business', 'vacation', 'house', 'car',
       'medical', 'moving', 'wedding', 'educational', 'renewable_energy'],
      dtype=object)
```

In [146…
```
df = PdE_catégorique(x_train, 'purpose', y_train)
df
```

Out[146]:

	purpose	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE
0	educational	385	0.797403	0.000917	307.0	78.0	0.000808	0.001965	-0.888955	NaN	NaN
1	small_business	6326	0.799083	0.015074	5055.0	1271.0	0.013304	0.032026	-0.878520	0.012496	0.010435
2	renewable_energy	323	0.851393	0.000770	275.0	48.0	0.000724	0.001209	-0.513524	0.012580	0.364996
3	moving	2683	0.863586	0.006393	2317.0	366.0	0.006098	0.009222	-0.413699	0.005374	0.099825
4	house	2065	0.874092	0.004921	1805.0	260.0	0.004750	0.006551	-0.321460	0.001347	0.092239
5	other	21337	0.875756	0.050844	18686.0	2651.0	0.049178	0.066799	-0.306256	0.044427	0.015203
6	wedding	2118	0.876298	0.005047	1856.0	262.0	0.004885	0.006602	-0.301259	0.044293	0.004997
7	medical	4168	0.876919	0.009932	3655.0	513.0	0.009619	0.012926	-0.295518	0.004735	0.005741
8	vacation	2265	0.895806	0.005397	2029.0	236.0	0.005340	0.005947	-0.107627	0.004279	0.187891
9	debt_consolidation	246742	0.902623	0.587963	222715.0	24027.0	0.586138	0.605428	-0.032379	0.580798	0.075248
10	major_purchase	8810	0.915210	0.020993	8063.0	747.0	0.021220	0.018823	0.119882	0.564918	0.152261
11	home_improvement	23871	0.916719	0.056882	21883.0	1988.0	0.057591	0.050093	0.139487	0.036371	0.019605
12	car	4854	0.918006	0.011567	4456.0	398.0	0.011727	0.010029	0.156461	0.045864	0.016974
13	credit_card	93709	0.926997	0.223300	86868.0	6841.0	0.228618	0.172378	0.282362	0.216891	0.125901

In [147…
```
graph_PdE(df)
```



Du nombre d'observations, nous identifions deux catégories très importantes : Consolidation de la dette et Carte de crédit, qui devraient aller seules.

Deux autres groupes pertinents (5%) qui pourraient également être autonomes ou constituer le noyau d'un cluster.

Autres et Améliorations domiciliaires

En fonction du PdE et du poids des observations, les groupes pourraient être les suivants : Éducation - Petite entreprise - Énergie renouvelable - Déménagement

Maison - Autres - Mariage - Santé - Vacances

Consolidation de la dette

Grands achats - Améliorations - Automobile

Carte de crédit

In [148…
```
# Éducation - Petite entreprise - Énergie renouvelable - Déménagement
```

In [149…
```
x_train['purpose_ed_pyme_enerren_moving'] = sum([x_train['Purpose_educational'],
                                                  x_train['Purpose_small_business'],
                                                  x_train['Purpose_renewable_energy'],
                                                  x_train['Purpose_moving']])
```

In [150…
```
# Maison - Autres - Mariage - Santé - Vacances
```

In [151…
```
x_train['purpose_house_other_wedding_medical_vacation'] = sum([x_train['Purpose_house'],
                                                  x_train['Purpose_other'],
                                                  x_train['Purpose_wedding'],
                                                  x_train['Purpose_medical'],
                                                  x_train['Purpose_vacation']])
```

In [152…
```
# Consolidation de la dette
# Grands achats - Améliorations - Automobile
# Tarjeta de Crédito
```

In [153…
```
x_train['Purpose_major_purchase_improvement_car'] = sum([x_train['Purpose_major_purchase'],
                                                  x_train['Purpose_home_improvement'],
                                                  x_train['Purpose_car']])
```

grade

In [154…
```
x_train['grade'].unique()
```

Out[154]:
```
array(['B', 'E', 'C', 'D', 'A', 'G', 'F'], dtype=object)
```

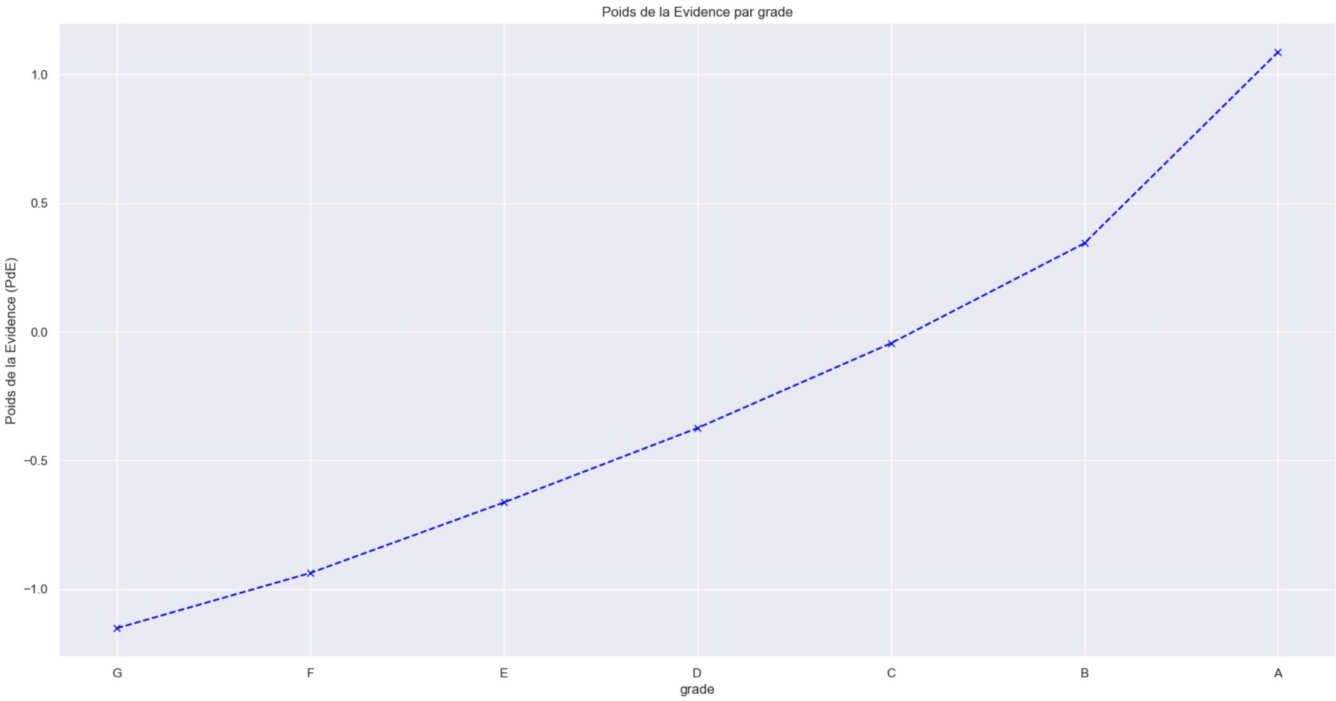
In [155…
```
df = PdE_catégorique(x_train, 'grade', y_train)
```

In [155]:
```
df = PdE_catégorique(x_train, 'grade', y_train)
df
```

Out[155]:

	grade	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	VI	sc
0	G	3000	0.751667	0.007149	2255.0	745.0	0.005935	0.018772	-1.151573	NaN	NaN	0.014784	0.
1	F	11912	0.789540	0.028385	9405.0	2507.0	0.024752	0.063171	-0.936939	0.018817	0.214634	0.035996	0.
2	E	32114	0.831475	0.076525	26702.0	5412.0	0.070274	0.136371	-0.662974	0.045522	0.273965	0.043820	0.
3	D	69267	0.868191	0.165057	60137.0	9130.0	0.158268	0.230056	-0.374034	0.087994	0.288940	0.026851	0.
4	C	112753	0.901546	0.268680	101652.0	11101.0	0.267526	0.279721	-0.044574	0.109259	0.329460	0.000544	0.
5	B	123286	0.931063	0.293779	114787.0	8499.0	0.302095	0.214156	0.344036	0.034569	0.388610	0.030254	0.
6	A	67324	0.965956	0.160427	65032.0	2292.0	0.171150	0.057753	1.086361	0.130945	0.742325	0.123190	0.

In [156…
```
graph_PdE(df)
```



La plupart des catégories ont un poids significatif dans le nombre d'observations à l'exception des catégories G et F,les deux ont un PdE similaire et sont candidates à être regroupées en une seule catégorie

In [157…
```
x_train['Grades_F_G'] = sum([x_train['Grade_F'],
                            x_train['Grade_G']])
```

initial_status

In [158…
```
x_train['initial_list_status'].unique()
```

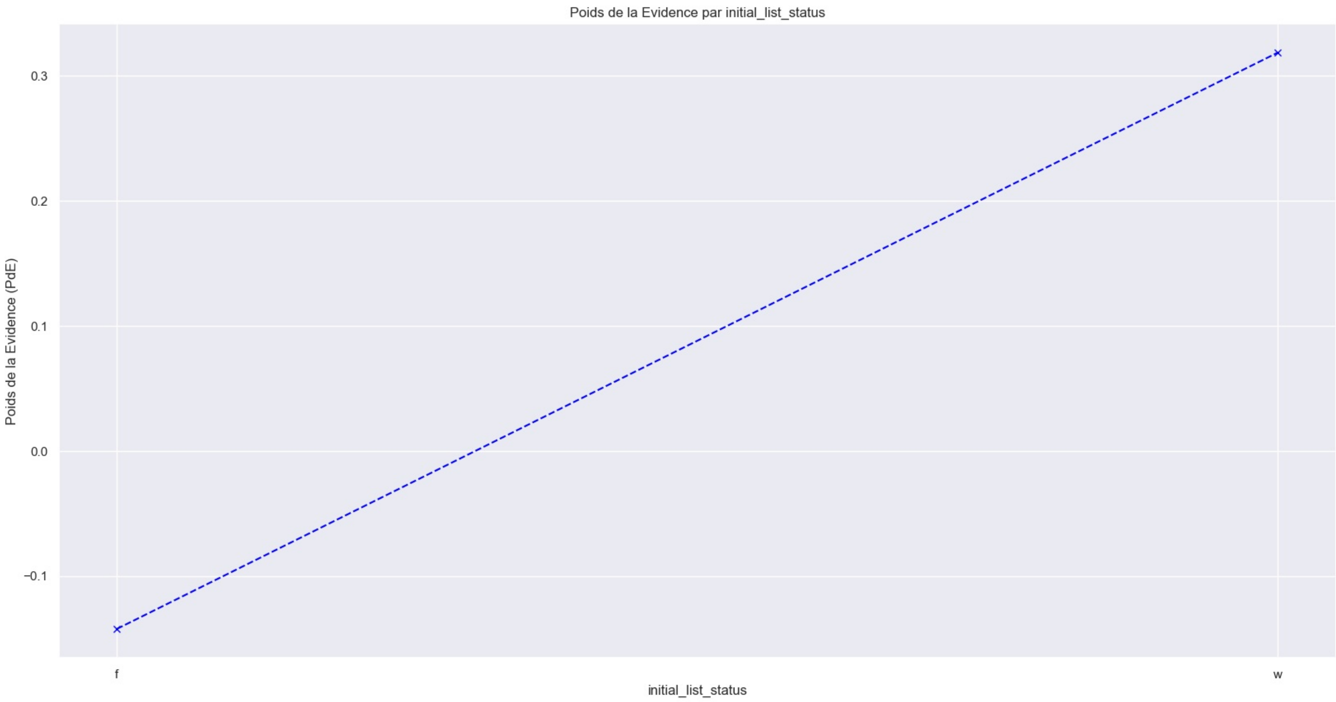
Out[158]:
```
array(['w', 'f'], dtype=object)
```

In [159…
```
df = PdE_catégorique(x_train, 'initial_list_status', y_train)
df
```

Out[159]:

	initial_list_status	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	
0	f	272685	0.892513	0.649782	243375.0	29310.0	0.640511	0.738548	-0.142419	NaN	NaN	0.
1	w	146971	0.929401	0.350218	136595.0	10376.0	0.359489	0.261452	0.318431	0.281022	0.46085	0.

In [160…
```
graph_PdE(df)
```



Il n'y a pas de sens à regrouper les deux catégories contenues dans la variable initial_status. Simplement, nous décidons de choisir la catégorie "f" comme la variable de référence.

Variables Continues

In [161…
```
# Nous copions et collons la fonction PdE_discrete
# Nous commentons les lignes de code où nous triions les valeurs par PdE et réinitialisions les indices
```

In [162…
```
def PdE_continue(df, var_categ, df_var_y):
    df = pd.concat([df[var_categ], df_var_y], axis = 1)
    df = pd.concat([df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].count(),
                    df.groupby(df.columns.values[0], as_index = False)[df.columns.values[1]].mean()], axis=1)
    df = df.iloc[:, [0,1,3]]
    df.columns = [df.columns.values[0], 'observations', '%_bon']
    df['%_obs'] = df['observations']/df['observations'].sum()
    df['n_bons'] = df['%_bon']*df['observations']
    df['n_malus'] = (1-df['%_bon'])*df['observations']
    df['%_n_bons'] = df['n_bons']/df['n_bons'].sum()
    df['%_n_malus'] = df['n_malus']/df['n_malus'].sum()
    df['PdE'] = np.log(df['%_n_bons']/df['%_n_malus'])
    #df = df.sort_values(['PdE'])
    #df = df.reset_index(drop = True)
    df['delta_%_n_bons'] = df['%_n_bons'].diff().abs()
    df['delta_PdE'] = df['PdE'].diff().abs() #Diferencia Absoluta entre categorías
    df['VI'] = (df['%_n_bons'] - df['%_n_malus']) * df['PdE']
    df.replace([np.inf, -np.inf], np.nan, inplace=True) #np ignore les nan mais pas les inf, alors comme ca on
    df['sommeVI'] = df['VI'].sum()
    return df
```

Comme le résultat sera un dataframe, nous n'avons pas besoin d'apporter de changements à notre fonction de graphique, la seule différence sera que le graphique ne sera pas trié du PdE le plus bas au plus élevé

variable term

In [163…
```
x_train['term_num'].unique()
```

Out[163]:
```
array([36, 60], dtype=int64)
```

Bien que ce soient des variables numériques, ce sont seulement deux nombres entiers, nous pouvons donc les traiter comme des variables catégoriques. Nous n'avons pas à faire de classification. Nous créons notre tableau avec la fonction PdE_continue.

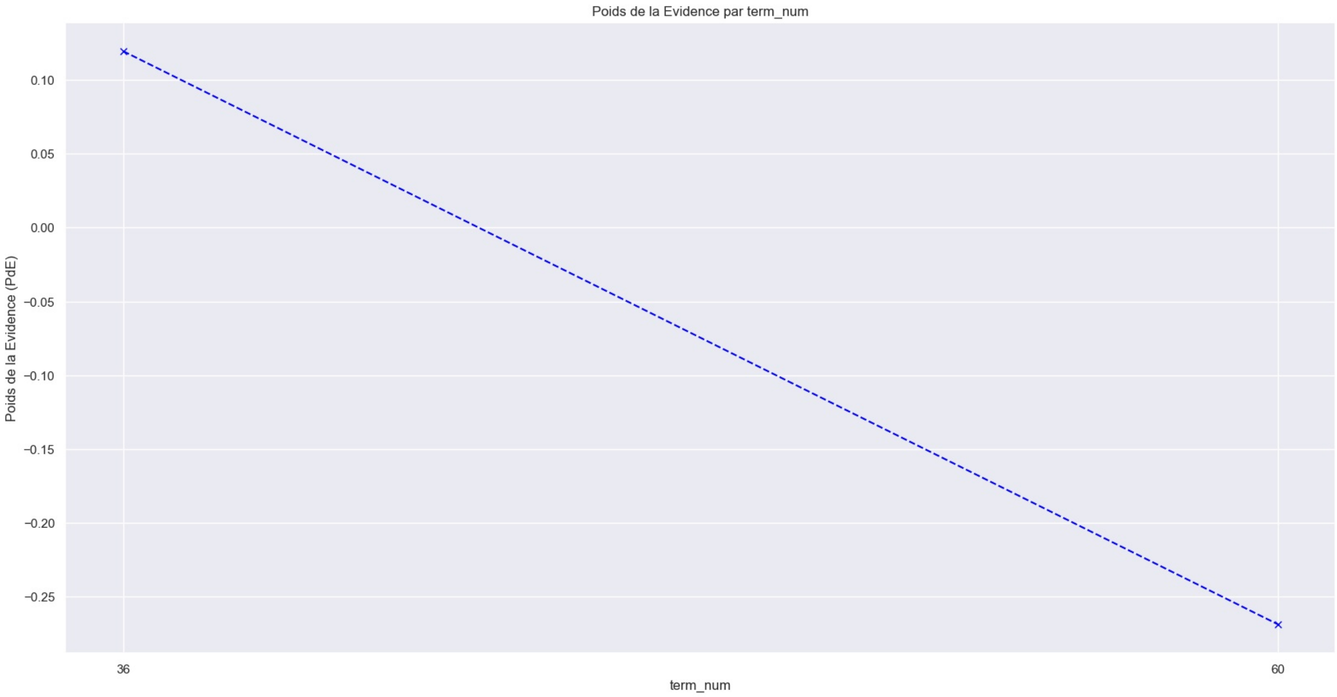
In [164…
```
df = PdE_continue(x_train, 'term_num', y_train)
```

In [165…
```
df
```

Out[165]:

	term_num	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	VI
0	36	304125	0.915170	0.724701	278326.0	25799.0	0.732495	0.650078	0.119364	NaN	NaN	0.009838
1	60	115531	0.879798	0.275299	101644.0	13887.0	0.267505	0.349922	-0.268570	0.464989	0.387934	0.022135

In [166…
```
graph_PdE(df)
```



Nous observons que la catégorie 36 a un PdE beaucoup plus élevé que celle de 60. Mais nous remarquons que notre graphique n'est plus trié de manière ascendante car nous avons utilisé la fonction PdE_continue. Nous créons maintenant nos variables dummy dans le dataframe x_train.

In [167…
```
x_train['échéance_36'] = np.where(x_train['term_num'] == 36, 1, 0)
```

In [168…
```
x_train['échéance_60'] = np.where(x_train['term_num'] == 60, 1, 0)
```

variable emp_length_num

In [170…
```
# Nombre d'années pendant lesquelles la personne qui demande le crédit a été employée
```

In [171…
```
x_train['emp_length_num'].unique()
```

Out[171]:
```
array([10,  0,  7,  3,  1,  2,  4,  5,  9,  6,  8], dtype=int64)
```

Ce n'est pas non plus une variable continue, elle nous donne les années sous forme de nombre entier. Il ne sera donc pas nécessaire de catégoriser la variable. Cependant, comme nous pouvons le voir, elle a plusieurs valeurs de 0 à 10, ce qui implique que nous devrons réduire le nombre de catégories. De plus, nous devons respecter l'ancienneté de l'emploi lors de la combinaison des catégories.

In [172…
```
df = PdE_continue(x_train, 'emp_length_num', y_train)
```

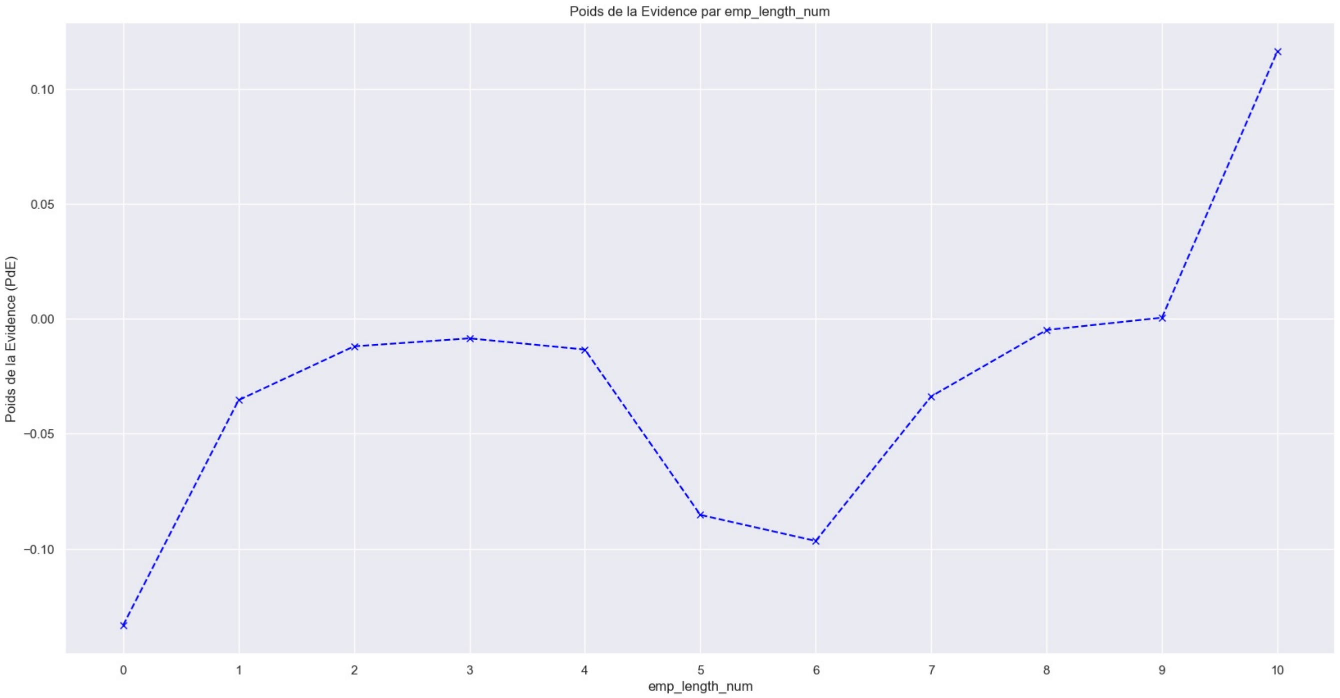
In [173…
```
df
```

Out[173]:

	emp_length_num	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE
0	0	51671	0.893383	0.123127	46162.0	5509.0	0.121489	0.138815	-0.133320	NaN	NaN
1	1	26660	0.902363	0.063528	24057.0	2603.0	0.063313	0.065590	-0.035332	0.058176	0.097988
2	2	37209	0.904405	0.088665	33652.0	3557.0	0.088565	0.089629	-0.011939	0.025252	0.023394
3	3	32961	0.904706	0.078543	29820.0	3141.0	0.078480	0.079146	-0.008456	0.010085	0.003483
4	4	25283	0.904284	0.060247	22863.0	2420.0	0.060171	0.060979	-0.013341	0.018309	0.004886
5	5	27681	0.897872	0.065961	24854.0	2827.0	0.065410	0.071234	-0.085291	0.005240	0.071950
6	6	23514	0.896827	0.056032	21088.0	2426.0	0.055499	0.061130	-0.096633	0.009911	0.011342
7	7	23570	0.902503	0.056165	21272.0	2298.0	0.055983	0.057905	-0.033741	0.000484	0.062892
8	8	20109	0.905018	0.047918	18199.0	1910.0	0.047896	0.048128	-0.004830	0.008087	0.028911
9	9	16060	0.905479	0.038269	14542.0	1518.0	0.038271	0.038250	0.000554	0.009624	0.005384
10	10	134938	0.914946	0.321544	123461.0	11477.0	0.324923	0.289195	0.116486	0.286652	0.115933

In [174…

```
graph_PdE(df)
```



Les catégories possibles sont ;

0 va seule

1-4 ont un PdE similaire et seront combinées

5-6 vont dans le même groupe

7 va seule

8-9 vont dans le même groupe

10 a un PdE beaucoup plus élevé que les autres, en plus son poids d'observations est proche d'1/3 du total, donc elle doit aller seule

In [175…

```
x_train['ancianité_<_1'] = np.where(x_train['emp_length_num'].isin([0]), 1, 0)
```

In [176…

```
#Generons maintenant la catégorie pour les employés ayant entre 1 et 4 ans d'ancienneté.
```

In [177…

```
x_train['ancianité_1a4'] = np.where(x_train['emp_length_num'].isin([range(1,4)]), 1, 0)
```

In [178…

```
# Generons maintenant les autres catégories.
```

In [179…
```
x_train['ancianité_5a6'] = np.where(x_train['emp_length_num'].isin(range(5,6)), 1, 0)
x_train['ancianité_7'] = np.where(x_train['emp_length_num'].isin([7]), 1, 0)
x_train['ancianité_8a9'] = np.where(x_train['emp_length_num'].isin(range(8,9)), 1, 0)
x_train['ancianité_10+'] = np.where(x_train['emp_length_num'].isin([10]), 1, 0)
```

In [180…
```
list(x_train.columns)
```

Out[180]:
```
['Unnamed: 0',
 'id',
 'member_id',
 'loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'term',
 'int_rate',
 'installment',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'annual_inc',
 'verification_status',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'url',
 'desc',
 'purpose',
 'title',
 'zip_code',
 'addr_state',
 'dti',
 'delinq_2yrs',
 'earliest_cr_line',
 'inq_last_6mths',
 'mths_since_last_delinq',
 'mths_since_last_record',
 'open_acc',
 'pub_rec',
 'revol_bal',
 'revol_util',
 'total_acc',
 'initial_list_status',
 'out_prncp',
 'out_prncp_inv',
 'total_pymnt',
 'total_pymnt_inv',
 'total_rec_prncp',
 'total_rec_int',
 'total_rec_late_fee',
 'recoveries',
 'collection_recovery_fee',
 'last_pymnt_d',
 'last_pymnt_amnt',
 'next_pymnt_d',
 'last_credit_pull_d',
 'collections_12_mths_ex_med',
 'mths_since_last_major_derog',
 'policy_code',
 'application_type',
 'annual_inc_joint',
 'dti_joint',
 'verification_status_joint',
 'acc_now_delinq',
 'tot_coll_amt',
 'tot_cur_bal',
 'open_acc_6m',
 'open_il_6m',
 'open_il_12m',
 'open_il_24m',
 'mths_since_rcnt_il',
 'total_bal_il',
 'il_util',
 'open_rv_12m',
 'open_rv_24m',
 'max_bal_bc',
 'all_util',
 'total_rev_hi_lim',
 'inq_fi',
 'total_cu_tl',
 'inq_last_12m',
 'term_num',
 'emp_length_num',
 'date',
 'nb_mois',
 'date_1er_credit',
 'nb_mois_1er_crédit',
 'Home_ANY',
```

```
'Home_MORTGAGE',
'Home_NONE',
'Home_OTHER',
'Home_OWN',
'Home_RENT',
'Purpose_car',
'Purpose_credit_card',
'Purpose_debt_consolidation',
'Purpose_educational',
'Purpose_home_improvement',
'Purpose_house',
'Purpose_major_purchase',
'Purpose_medical',
'Purpose_moving',
'Purpose_other',
'Purpose_renewable_energy',
'Purpose_small_business',
'Purpose_vacation',
'Purpose_wedding',
'Grade_A',
'Grade_B',
'Grade_C',
'Grade_D',
'Grade_E',
'Grade_F',
'Grade_G',
'initial_list_status_f',
'initial_list_status_w',
'Adresse_AK',
'Adresse_AL',
'Adresse_AR',
'Adresse_AZ',
'Adresse_CA',
'Adresse_CO',
'Adresse_CT',
'Adresse_DC',
'Adresse_DE',
'Adresse_FL',
'Adresse_GA',
'Adresse_HI',
'Adresse_IA',
'Adresse_ID',
'Adresse_IL',
'Adresse_IN',
'Adresse_KS',
'Adresse_KY',
'Adresse_LA',
'Adresse_MA',
'Adresse_MD',
'Adresse_ME',
'Adresse_MI',
'Adresse_MN',
'Adresse_MO',
'Adresse_MS',
'Adresse_MT',
'Adresse_NC',
'Adresse_NE',
'Adresse_NH',
'Adresse_NJ',
'Adresse_NM',
'Adresse_NV',
'Adresse_NY',
'Adresse_OH',
'Adresse_OK',
'Adresse_OR',
'Adresse_PA',
'Adresse_RI',
'Adresse_SC',
'Adresse_SD',
'Adresse_TN',
'Adresse_TX',
'Adresse_UT',
'Adresse_VA',
'Adresse_VT',
'Adresse_WA',
'Adresse_WI',
'Adresse_WV',
'Adresse_WY',
'loan_status_Charged Off',
'loan_status_Current',
'loan_status_Default',
'loan_status_Does not meet the credit policy. Status:Charged Off',
'loan_status_Does not meet the credit policy. Status:Fully Paid',
'loan_status_Fully Paid',
'loan_status_In Grace Period',
'loan_status_Late (16-30 days)',
'loan_status_Late (31-120 days)',
'Verification_Not Verified',
'Verification_Source Verified',
```

```
 'Verification_Verified',
 'Subgrade_A1',
 'Subgrade_A2',
 'Subgrade_A3',
 'Subgrade_A4',
 'Subgrade_A5',
 'Subgrade_B1',
 'Subgrade_B2',
 'Subgrade_B3',
 'Subgrade_B4',
 'Subgrade_B5',
 'Subgrade_C1',
 'Subgrade_C2',
 'Subgrade_C3',
 'Subgrade_C4',
 'Subgrade_C5',
 'Subgrade_D1',
 'Subgrade_D2',
 'Subgrade_D3',
 'Subgrade_D4',
 'Subgrade_D5',
 'Subgrade_E1',
 'Subgrade_E2',
 'Subgrade_E3',
 'Subgrade_E4',
 'Subgrade_E5',
 'Subgrade_F1',
 'Subgrade_F2',
 'Subgrade_F3',
 'Subgrade_F4',
 'Subgrade_F5',
 'Subgrade_G1',
 'Subgrade_G2',
 'Subgrade_G3',
 'Subgrade_G4',
 'Subgrade_G5',
 'Home_RENT_ANY_OTHER_NONE',
 'Adresse_ND',
 'Adresse_ND_NE_IA_NV',
 'Adresse_AL_HI_MO_NM',
 'Adresse_NC_ID_NJ',
 'Adresse_KY_LA_MD',
 'Adresse_MI_AR_AZ_VA_OK_DE_OH',
 'Adresse_MN_PA_UT_MA_RI_WA_TN_IN',
 'Adresse_OR_SD_WI_GA',
 'Adresse_CT_IL',
 'Adresse_NH_AK_MT_MS_WY_WV_DC_ME',
 'purpose_ed_pyme_enerren_moving',
 'purpose_house_other_wedding_medical_vacation',
 'Purpose_major_purchase_improvement_car',
 'Grades_F_G',
 'échéance_36',
 'échéance_60',
 'ancianité_<_1',
 'ancianité_1a4',
 'ancianité_5a6',
 'ancianité_7',
 'ancianité_8a9',
 'ancianité_10+']
```

nb_mois_1er_crédit

Correspond aux nombres de mois passés après l'octroi du prémier crédit

In [181…

```
x_train['nb_mois_1er_crédit'].unique()
```

Out[181]:
```
array([ 90.05523727,  89.03673587,  87.03258794,  94.98346989,
       106.97550258, 101.02876856, 102.04726996, 103.03291649,
        92.0593852 , 120.01889156,  93.04503173, 141.04601737,
        94.06353313, 142.06451878,  86.01408653,  96.00197129,
        99.02462063,  91.0408838 , 127.04983675, 138.05622292,
        97.0204727 , 107.99400398, 100.01026715, 122.02303949,
       119.00039015, 106.05556582, 117.02909711, 139.04186944,
       109.99815191,  98.00611922, 113.02080125, 130.99242284,
       104.05141789, 118.04759851, 124.02718742, 114.03930266,
       109.01250539, 125.04568882,  85.02844001, 112.00229984,
       130.07248609, 133.02942566, 105.03706442, 137.03772151,
       135.03357358, 149.0297542 , 148.0112528 , 132.01092425,
        88.01823446, 111.01665332, 116.04345058, 136.01922011,
       115.02494918, 129.05398468, 146.00710487, 171.04252654,
       151.03390213, 166.04858416, 126.06419023, 123.04154089,
       142.98445553, 160.00328549, 144.00295694, 154.05655147,
       152.05240354, 167.0013758 , 155.99498963, 169.03837861,
       121.03739296, 153.03805006, 157.01349104, 165.03008275,
       164.04443623, 147.02560628, 128.06833816, 140.06037085,
       168.0198772 , 175.0508224 , 145.02145835, 157.99913756,
       134.01507218, 154.97648822, 173.04667447, 150.04825561,
       162.0402883 , 170.02402513, 163.02593482, 159.01763897,
       174.06517588, 172.02817306, 161.0217869 ])
```

In [182…
```python
valeur_max, valeur_min = x_train['nb_mois_1er_crédit'].agg(['max', 'min'])
valeur_max,valeur_min
```

Out[182]:
```
(175.05082239881722, 85.0284400090351)
```

on va regrouper les nombres de mois par 50 catégories afin que chaque observation soit affectée à un intervalle

In [183…
```python
x_train['nb_mois_1er_crédit_intervalle']=pd.cut(x_train['nb_mois_1er_crédit'],50)
```

In [184…
```python
x_train['nb_mois_1er_crédit_intervalle']
#l'observation 344531 est entre >88.629 et <= 90.43
```

Out[184]:
```
344531       (88.629, 90.43]
328300       (88.629, 90.43]
299890      (86.829, 88.629]
439226      (94.031, 95.831]
167889    (106.634, 108.434]
                 ...
239305      (84.938, 86.829]
167080    (106.634, 108.434]
177337    (106.634, 108.434]
23587     (133.641, 135.441]
160385    (104.833, 106.634]
Name: nb_mois_1er_crédit_intervalle, Length: 419656, dtype: category
Categories (50, interval[float64, right]): [(84.938, 86.829] < (86.829, 88.629] < (88.629, 90.43] < (90.43, 92
.23] ... (167.849, 169.649] < (169.649, 171.45] < (171.45, 173.25] < (173.25, 175.051]]
```

In [185…
```python
df=PdE_continue(x_train,'nb_mois_1er_crédit_intervalle',y_train)
df
```

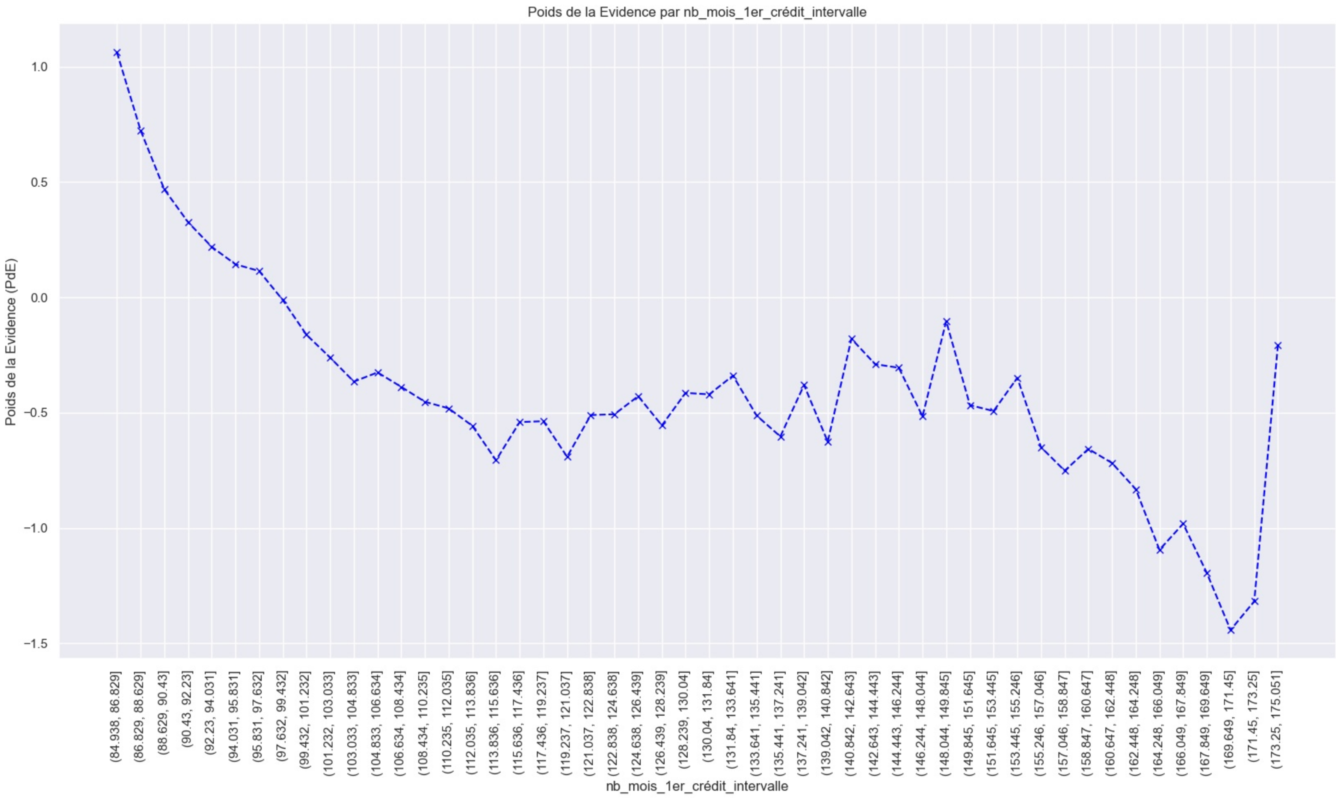
Out[185]:

	nb_mois_1er_crédit_intervalle	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons
0	(84.938, 86.829]	31889	0.965160	0.075988	30778.0	1111.0	0.081001	0.027995	1.062446	NaN
1	(86.829, 88.629]	44380	0.951803	0.105753	42241.0	2139.0	0.111169	0.053898	0.723959	0.030168
2	(88.629, 90.43]	43236	0.938616	0.103027	40582.0	2654.0	0.106803	0.066875	0.468163	0.004366
3	(90.43, 92.23]	32716	0.929912	0.077959	30423.0	2293.0	0.080067	0.057779	0.326244	0.026736
4	(92.23, 94.031]	17130	0.922592	0.040819	15804.0	1326.0	0.041593	0.033412	0.219002	0.038474
5	(94.031, 95.831]	28620	0.917016	0.068199	26245.0	2375.0	0.069071	0.059845	0.143384	0.027478
6	(95.831, 97.632]	27607	0.914768	0.065785	25254.0	2353.0	0.066463	0.059290	0.114200	0.002608
7	(97.632, 99.432]	25906	0.904424	0.061732	23430.0	2476.0	0.061663	0.062390	-0.011721	0.004800
8	(99.432, 101.232]	23096	0.890587	0.055036	20569.0	2527.0	0.054133	0.063675	-0.162342	0.007530
9	(101.232, 103.033]	20478	0.880555	0.048797	18032.0	2446.0	0.047456	0.061634	-0.261400	0.006677
10	(103.033, 104.833]	9348	0.869170	0.022275	8125.0	1223.0	0.021383	0.030817	-0.365455	0.026073
11	(104.833, 106.634]	15922	0.873634	0.037941	13910.0	2012.0	0.036608	0.050698	-0.325615	0.015225
12	(106.634, 108.434]	12971	0.866548	0.030909	11240.0	1731.0	0.029581	0.043617	-0.388314	0.007027
13	(108.434, 110.235]	11218	0.858888	0.026731	9635.0	1583.0	0.025357	0.039888	-0.453013	0.004224
14	(110.235, 112.035]	11156	0.855414	0.026584	9543.0	1613.0	0.025115	0.040644	-0.481382	0.000242
15	(112.035, 113.836]	4850	0.845773	0.011557	4102.0	748.0	0.010796	0.018848	-0.557267	0.014320
16	(113.836, 115.636]	7590	0.825296	0.018086	6264.0	1326.0	0.016486	0.033412	-0.706442	0.005690
17	(115.636, 117.436]	5950	0.847899	0.014178	5045.0	905.0	0.013277	0.022804	-0.540876	0.003208
18	(117.436, 119.237]	4920	0.848374	0.011724	4174.0	746.0	0.010985	0.018798	-0.537189	0.002292
19	(119.237, 121.037]	4387	0.827445	0.010454	3630.0	757.0	0.009553	0.019075	-0.691469	0.001432
20	(121.037, 122.838]	1997	0.851778	0.004759	1701.0	296.0	0.004477	0.007459	-0.510482	0.005077
21	(122.838, 124.638]	3756	0.852236	0.008950	3201.0	555.0	0.008424	0.013985	-0.506843	0.003948
22	(124.638, 126.439]	3466	0.861800	0.008259	2987.0	479.0	0.007861	0.012070	-0.428770	0.000563
23	(126.439, 128.239]	3195	0.846009	0.007613	2703.0	492.0	0.007114	0.012397	-0.555455	0.000747
24	(128.239, 130.04]	1414	0.863508	0.003369	1221.0	193.0	0.003213	0.004863	-0.414359	0.003900
25	(130.04, 131.84]	2521	0.862753	0.006007	2175.0	346.0	0.005724	0.008718	-0.420749	0.002511
26	(131.84, 133.641]	2439	0.872079	0.005812	2127.0	312.0	0.005598	0.007862	-0.339629	0.000126
27	(133.641, 135.441]	2215	0.851467	0.005278	1886.0	329.0	0.004964	0.008290	-0.512938	0.000634
28	(135.441, 137.241]	2139	0.839645	0.005097	1796.0	343.0	0.004727	0.008643	-0.603507	0.000237
29	(137.241, 139.042]	1094	0.867459	0.002607	949.0	145.0	0.002498	0.003654	-0.380419	0.002229
30	(139.042, 140.842]	1873	0.836626	0.004463	1567.0	306.0	0.004124	0.007711	-0.625761	0.001626
31	(140.842, 142.643]	1557	0.888889	0.003710	1384.0	173.0	0.003642	0.004359	-0.179652	0.000482
32	(142.643, 144.443]	1192	0.877517	0.002840	1046.0	146.0	0.002753	0.003679	-0.289972	0.000890
33	(144.443, 146.244]	1192	0.875839	0.002840	1044.0	148.0	0.002748	0.003729	-0.305491	0.000005
34	(146.244, 148.044]	987	0.851064	0.002352	840.0	147.0	0.002211	0.003704	-0.516125	0.000537
35	(148.044, 149.845]	395	0.896203	0.000941	354.0	41.0	0.000932	0.001033	-0.103369	0.001279
36	(149.845, 151.645]	742	0.857143	0.001768	636.0	106.0	0.001674	0.002671	-0.467334	0.000742
37	(151.645, 153.445]	623	0.853933	0.001485	532.0	91.0	0.001400	0.002293	-0.493310	0.000274
38	(153.445, 155.246]	566	0.871025	0.001349	493.0	73.0	0.001297	0.001839	-0.349044	0.000103
39	(155.246, 157.046]	468	0.833333	0.001115	390.0	78.0	0.001026	0.001965	-0.649656	0.000271
40	(157.046, 158.847]	193	0.818653	0.000460	158.0	35.0	0.000416	0.000882	-0.751847	0.000611
41	(158.847, 160.647]	155	0.832258	0.000369	129.0	26.0	0.000340	0.000655	-0.657378	0.000076
42	(160.647, 162.448]	221	0.823529	0.000527	182.0	39.0	0.000479	0.000983	-0.718649	0.000139
43	(162.448, 164.248]	217	0.806452	0.000517	175.0	42.0	0.000461	0.001058	-0.831977	0.000018
44	(164.248, 166.049]	227	0.762115	0.000541	173.0	54.0	0.000455	0.001361	-1.094786	0.000005
45	(166.049, 167.849]	633	0.781991	0.001508	495.0	138.0	0.001303	0.003477	-0.981790	0.000847
46	(167.849, 169.649]	417	0.743405	0.000994	310.0	107.0	0.000816	0.002696	-1.195350	0.000487
47	(169.649, 171.45]	199	0.693467	0.000474	138.0	61.0	0.000363	0.001537	-1.442714	0.000453
48	(171.45, 173.25]	114	0.719298	0.000272	82.0	32.0	0.000216	0.000806	-1.318110	0.000147
49	(173.25, 175.051]	79	0.886076	0.000188	70.0	9.0	0.000184	0.000227	-0.207823	0.000032

Dans la colonne %_obs, les dix premiers bins ont un poids compris entre 10% et 4%. Ensuite, le pourcentage commence à diminuer. Entre les bins 10 et 19, le poids est de 3% à 1%. Entre les bins 20 et 40, le pourcentage est compris entre 0.9% et 0.1%. Après le bin 40, le pourcentage est pratiquement nul. Cette analyse est utile pour effectuer une classification plus générale.

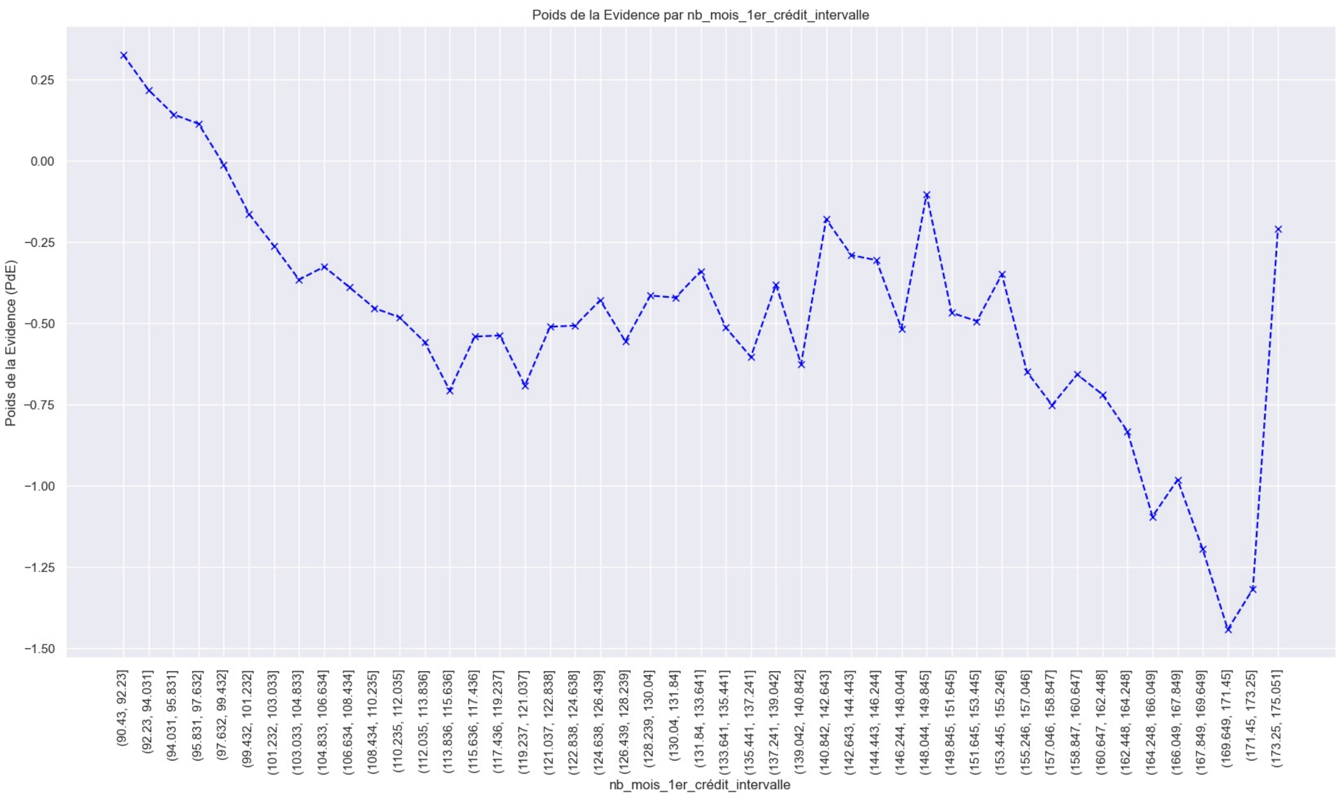
In [186…
```
def graph_PdE_V(df):
    x = np.array(df.iloc[:,0].apply(str)) #0=   home_ownership
    y = df['PdE']
    plt.figure(figsize = (20,10))
    plt.plot(x, y, marker = 'x', linestyle = '--', color = 'blue')
    plt.xlabel(df.columns[0])
    plt.ylabel('Poids de la Evidence (PdE)')
    plt.title(str('Poids de la Evidence par ' + df.columns[0]))
    plt.xticks(rotation=90)
```

In [187…
```
graph_PdE_V(df)
```



Les trois prèmieres catgéories ont un PdE très differente. Par conséqutce, nous ne les regrouperon pas et nous les retirerons pour rendre le graphique plus compréhensible.

In [188…
```
graph_PdE_V(df.iloc[3:,:])
```



In [189…
```
x_train['nb_mois_1er_crédit_<87'] = np.where(x_train['nb_mois_1er_crédit']<87,1,0)
```

In [190…
```
x_train['nb_mois_1er_crédit_87à89'] = np.where((x_train['nb_mois_1er_crédit']>=87) & (x_train['nb_mois_1er_créd
```

In [191…
```
x_train['nb_mois_1er_crédit_89à90']=np.where((x_train['nb_mois_1er_crédit']>=89)&(x_train['nb_mois_1er_crédit']
```

In [192…
```
x_train['nb_mois_1er_crédit_90à98']=np.where((x_train['nb_mois_1er_crédit']>=90)&(x_train['nb_mois_1er_crédit']
x_train['nb_mois_1er_crédit_98à101']=np.where((x_train['nb_mois_1er_crédit']>=98)&(x_train['nb_mois_1er_crédit'
x_train['nb_mois_1er_crédit_101à110']=np.where((x_train['nb_mois_1er_crédit']>=101)&(x_train['nb_mois_1er_crédi
x_train['nb_mois_1er_crédit_110à126']=np.where((x_train['nb_mois_1er_crédit']>=110)&(x_train['nb_mois_1er_crédi
x_train['nb_mois_1er_crédit_126à155']=np.where((x_train['nb_mois_1er_crédit']>=126)&(x_train['nb_mois_1er_crédi
x_train['nb_mois_1er_crédit_>155']=np.where(x_train['nb_mois_1er_crédit']>=155,1,0)
```

annual_inc

Classification de la variable de revenu annuel

In [193…
```
x_train['annual_inc_categ'] = pd.cut(x_train['annual_inc'], 50) #Classifions en 50 groupes.
```

In [194…
```
df = PdE_continue(x_train, 'annual_inc_categ', y_train)
```

In [195…
```
df
```

Out[195]:

	annual_inc_categ	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE
0	(-5602.104, 151858.08]	401270	0.903870	0.956188	362696.0	38574.0	0.954539	0.971980	-0.018107	NaN	NaN
1	(151858.08, 301820.16]	16542	0.939064	0.039418	15534.0	1008.0	0.040882	0.025399	0.475969	0.913656	0.494076
2	(301820.16, 451782.24]	1248	0.943109	0.002974	1177.0	71.0	0.003098	0.001789	0.548950	0.037785	0.072981
3	(451782.24, 601744.32]	337	0.946588	0.000803	319.0	18.0	0.000840	0.000454	0.615726	0.002258	0.066775
4	(601744.32, 751706.4]	107	0.925234	0.000255	99.0	8.0	0.000261	0.000202	0.256584	0.000579	0.359141
5	(751706.4, 901668.48]	68	0.911765	0.000162	62.0	6.0	0.000163	0.000151	0.076281	0.000097	0.180303
6	(901668.48, 1051630.56]	31	0.967742	0.000074	30.0	1.0	0.000079	0.000025	1.142104	0.000084	1.065822
7	(1051630.56, 1201592.64]	20	1.000000	0.000048	20.0	0.0	0.000053	0.000000	NaN	0.000026	NaN
8	(1201592.64, 1351554.72]	10	1.000000	0.000024	10.0	0.0	0.000026	0.000000	NaN	0.000026	NaN
9	(1351554.72, 1501516.8]	5	1.000000	0.000012	5.0	0.0	0.000013	0.000000	NaN	0.000013	NaN
10	(1501516.8, 1651478.88]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	0.000008	NaN
11	(1651478.88, 1801440.96]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000003	NaN
12	(1801440.96, 1951403.04]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	0.000003	NaN
13	(1951403.04, 2101365.12]	5	1.000000	0.000012	5.0	0.0	0.000013	0.000000	NaN	0.000008	NaN
14	(2101365.12, 2251327.2]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
15	(2251327.2, 2401289.28]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
16	(2401289.28, 2551251.36]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
17	(2551251.36, 2701213.44]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
18	(2701213.44, 2851175.52]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
19	(2851175.52, 3001137.6]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
20	(3001137.6, 3151099.68]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
21	(3151099.68, 3301061.76]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
22	(3301061.76, 3451023.84]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23	(3451023.84, 3600985.92]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN

24	(3600985.92, 3750948.0]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25	(3750948.0, 3900910.08]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
26	(3900910.08, 4050872.16]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
27	(4050872.16, 4200834.24]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
28	(4200834.24, 4350796.32]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29	(4350796.32, 4500758.4]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
30	(4500758.4, 4650720.48]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
31	(4650720.48, 4800682.56]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
32	(4800682.56, 4950644.64]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
33	(4950644.64, 5100606.72]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000000	NaN
34	(5100606.72, 5250568.8]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
35	(5250568.8, 5400530.88]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
36	(5400530.88, 5550492.96]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
37	(5550492.96, 5700455.04]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
38	(5700455.04, 5850417.12]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
39	(5850417.12, 6000379.2]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
40	(6000379.2, 6150341.28]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
41	(6150341.28, 6300303.36]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42	(6300303.36, 6450265.44]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
43	(6450265.44, 6600227.52]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
44	(6600227.52, 6750189.6]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
45	(6750189.6, 6900151.68]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
46	(6900151.68, 7050113.76]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
47	(7050113.76, 7200075.84]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
48	(7200075.84, 7350037.92]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
49	(7350037.92, 7500000.0]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	NaN	NaN

En créant les 50 groupes, le premier couvre plus de 95 % des observations. C'est pas normal. Nous créons 100 groupes pour voir si en augmentant le nombre de groupes, nous pouvons obtenir une classification plus fine.

In [196…
```
x_train['annual_inc_categ'] = pd.cut(x_train['annual_inc'], 100)
```

In [197…
```
df = PdE_continue(x_train, 'annual_inc_categ', y_train)
df
```

Out[197]:

	annual_inc_categ	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE
0	(-5602.104, 76877.04]	275903	0.892865	0.657450	246344.0	29559.0	0.648325	0.744822	-0.138753	NaN	NaN
1	(76877.04, 151858.08]	125367	0.928091	0.298738	116352.0	9015.0	0.306214	0.227158	0.298636	0.342111	0.437390
2	(151858.08, 226839.12]	13342	0.938840	0.031793	12526.0	816.0	0.032966	0.020561	0.472054	0.273248	0.173417
3	(226839.12, 301820.16]	3200	0.940000	0.007625	3008.0	192.0	0.007916	0.004838	0.492441	0.025049	0.020388

4	(301820.16, 376801.2]	792	0.936869	0.001887	742.0	50.0	0.001953	0.001260	0.438232	0.005964	0.054209
5	(376801.2, 451782.24]	456	0.953947	0.001087	435.0	21.0	0.001145	0.000529	0.771730	0.000808	0.333497
6	(451782.24, 526763.28]	208	0.951923	0.000496	198.0	10.0	0.000521	0.000252	0.726588	0.000624	0.045142
7	(526763.28, 601744.32]	129	0.937984	0.000307	121.0	8.0	0.000318	0.000202	0.457255	0.000203	0.269333
8	(601744.32, 676725.36]	52	0.923077	0.000124	48.0	4.0	0.000126	0.000101	0.225813	0.000192	0.231442
9	(676725.36, 751706.4]	55	0.927273	0.000131	51.0	4.0	0.000134	0.000101	0.286437	0.000008	0.060625
10	(751706.4, 826687.44]	27	0.851852	0.000064	23.0	4.0	0.000061	0.000101	-0.509894	0.000074	0.796331
11	(826687.44, 901668.48]	41	0.951220	0.000098	39.0	2.0	0.000103	0.000050	0.711321	0.000042	1.221215
12	(901668.48, 976649.52]	11	0.909091	0.000026	10.0	1.0	0.000026	0.000025	0.043491	0.000076	0.667829
13	(976649.52, 1051630.56]	20	1.000000	0.000048	20.0	0.0	0.000053	0.000000	NaN	0.000026	NaN
14	(1051630.56, 1126611.6]	9	1.000000	0.000021	9.0	0.0	0.000024	0.000000	NaN	0.000029	NaN
15	(1126611.6, 1201592.64]	11	1.000000	0.000026	11.0	0.0	0.000029	0.000000	NaN	0.000005	NaN
16	(1201592.64, 1276573.68]	6	1.000000	0.000014	6.0	0.0	0.000016	0.000000	NaN	0.000013	NaN
17	(1276573.68, 1351554.72]	4	1.000000	0.000010	4.0	0.0	0.000011	0.000000	NaN	0.000005	NaN
18	(1351554.72, 1426535.76]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	0.000005	NaN
19	(1426535.76, 1501516.8]	3	1.000000	0.000007	3.0	0.0	0.000008	0.000000	NaN	0.000003	NaN
20	(1501516.8, 1576497.84]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000005	NaN
21	(1576497.84, 1651478.88]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000000	NaN
22	(1651478.88, 1726459.92]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
23	(1726459.92, 1801440.96]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
24	(1801440.96, 1876422.0]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
25	(1876422.0, 1951403.04]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	NaN	NaN
26	(1951403.04, 2026384.08]	4	1.000000	0.000010	4.0	0.0	0.000011	0.000000	NaN	0.000005	NaN
27	(2026384.08, 2101365.12]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000008	NaN
28	(2101365.12, 2176346.16]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
29	(2176346.16, 2251327.2]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
30	(2251327.2, 2326308.24]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
31	(2326308.24, 2401289.28]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
32	(2401289.28, 2476270.32]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
33	(2476270.32, 2551251.36]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
34	(2551251.36, 2626232.4]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
35	(2626232.4, 2701213.44]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
36	(2701213.44, 2776194.48]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
37	(2776194.48, 2851175.52]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
38	(2851175.52, 2926156.56]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
39	(2926156.56,	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN

	3001137.6]										
40	(3001137.6, 3076118.64]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
41	(3076118.64, 3151099.68]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42	(3151099.68, 3226080.72]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
43	(3226080.72, 3301061.76]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
44	(3301061.76, 3376042.8]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
45	(3376042.8, 3451023.84]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
46	(3451023.84, 3526004.88]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
47	(3526004.88, 3600985.92]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
48	(3600985.92, 3675966.96]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
49	(3675966.96, 3750948.0]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
50	(3750948.0, 3825929.04]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
51	(3825929.04, 3900910.08]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
52	(3900910.08, 3975891.12]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
53	(3975891.12, 4050872.16]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
54	(4050872.16, 4125853.2]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
55	(4125853.2, 4200834.24]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
56	(4200834.24, 4275815.28]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
57	(4275815.28, 4350796.32]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
58	(4350796.32, 4425777.36]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
59	(4425777.36, 4500758.4]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
60	(4500758.4, 4575739.44]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
61	(4575739.44, 4650720.48]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
62	(4650720.48, 4725701.52]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
63	(4725701.52, 4800682.56]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
64	(4800682.56, 4875663.6]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
65	(4875663.6, 4950644.64]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
66	(4950644.64, 5025625.68]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000000	NaN
67	(5025625.68, 5100606.72]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
68	(5100606.72, 5175587.76]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
69	(5175587.76, 5250568.8]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
70	(5250568.8, 5325549.84]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
71	(5325549.84, 5400530.88]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
72	(5400530.88, 5475511.92]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
73	(5475511.92, 5550492.96]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
74	(5550492.96, 5625474.0]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN

75	(5625474.0, 5700455.04]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
76	(5700455.04, 5775436.08]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
77	(5775436.08, 5850417.12]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
78	(5850417.12, 5925398.16]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
79	(5925398.16, 6000379.2]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
80	(6000379.2, 6075360.24]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
81	(6075360.24, 6150341.28]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
82	(6150341.28, 6225322.32]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
83	(6225322.32, 6300303.36]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
84	(6300303.36, 6375284.4]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
85	(6375284.4, 6450265.44]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
86	(6450265.44, 6525246.48]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
87	(6525246.48, 6600227.52]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
88	(6600227.52, 6675208.56]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
89	(6675208.56, 6750189.6]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
90	(6750189.6, 6825170.64]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
91	(6825170.64, 6900151.68]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
92	(6900151.68, 6975132.72]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
93	(6975132.72, 7050113.76]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
94	(7050113.76, 7125094.8]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
95	(7125094.8, 7200075.84]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
96	(7200075.84, 7275056.88]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
97	(7275056.88, 7350037.92]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
98	(7350037.92, 7425018.96]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
99	(7425018.96, 7500000.0]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	NaN	NaN

Le premier groupe capture 66 % des observations, tandis que le deuxième en capture près de 30 %. Le troisième groupe représente 3 % et le 1 % restant est réparti entre les 97 groupes restants. Qu'est-ce qui se passe ? Rappelons-nous que nous analysons la variable du revenu annuel. Ainsi, ce résultat est assez intuitif. Il y a beaucoup de personnes ayant de faibles revenus qui se concentrent dans les catégories de revenus les plus bas. Et très peu de personnes ayant des revenus élevés, très dispersées. Dans cette logique, nous pourrions former : Un groupe pour la classe moyenne supérieure, qui pourrait être celui de l'indice 2 avec des revenus entre 152K et 227K. Un groupe pour la classe aisée, c'est-à-dire à partir de l'indice 3 avec des revenus supérieurs à 227K. Et nous pourrions approfondir l'étude des revenus des indices 0 et 1.

In [198…

```
df_temp = x_train.loc[x_train['annual_inc'] <= 152000, : ] #df des personnes avec les revenus plus bas <152k
```

Maintenant, générons la classification fine uniquement dans la base en excluant les observations avec des revenus élevés.

In [199…

```
df_temp['annual_inc_categ'] = pd.cut(df_temp['annual_inc'], 50) #class avec les personnes des revenus bass
df = PdE_continue(df_temp, 'annual_inc_categ', y_train)
df
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_21276\1138887146.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df_temp['annual_inc_categ'] = pd.cut(df_temp['annual_inc'], 50) #class avec les personnes des revenus bass
```

Out[199]:

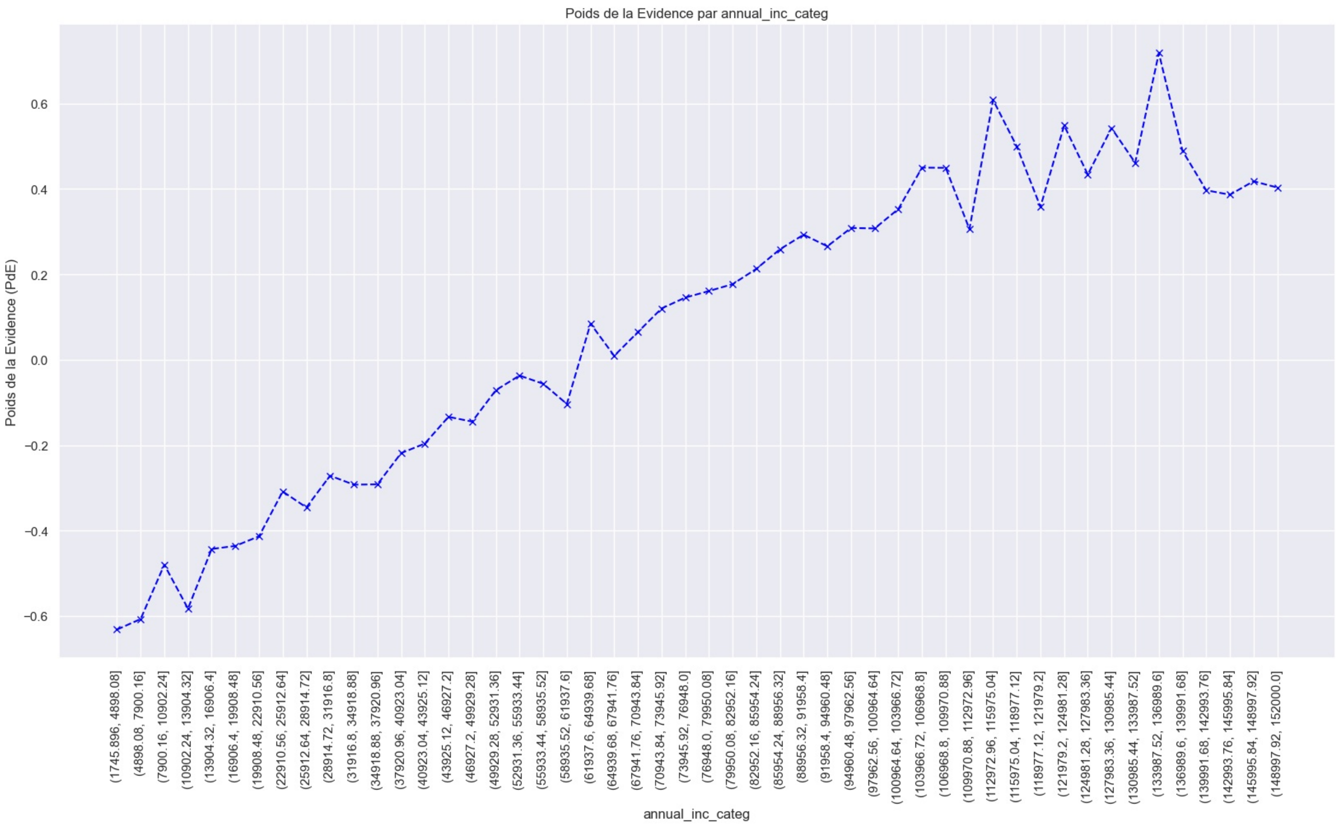
	annual_inc_categ	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	
0	(1745.896, 4898.08]	18	0.833333	0.000045	15.0	3.0	0.000041	0.000078	-0.631704	NaN	NaN	0
1	(4898.08, 7900.16]	49	0.836735	0.000122	41.0	8.0	0.000113	0.000207	-0.607011	0.000072	0.024693	0
2	(7900.16, 10902.24]	416	0.853365	0.001036	355.0	61.0	0.000978	0.001581	-0.479898	0.000865	0.127113	0
3	(10902.24, 13904.32]	794	0.840050	0.001978	667.0	127.0	0.001838	0.003292	-0.582539	0.000860	0.102641	0
4	(13904.32, 16906.4]	1654	0.857920	0.004120	1419.0	235.0	0.003911	0.006091	-0.443020	0.002073	0.139519	0
5	(16906.4, 19908.48]	2082	0.858790	0.005187	1788.0	294.0	0.004928	0.007620	-0.435869	0.001017	0.007151	0
6	(19908.48, 22910.56]	4130	0.861501	0.010288	3558.0	572.0	0.009806	0.014825	-0.413327	0.004878	0.022542	0
7	(22910.56, 25912.64]	7165	0.873552	0.017849	6259.0	906.0	0.017250	0.023482	-0.308406	0.007444	0.104922	0
8	(25912.64, 28914.72]	6767	0.869366	0.016858	5883.0	884.0	0.016214	0.022912	-0.345777	0.001036	0.037371	0
9	(28914.72, 31916.8]	11420	0.877583	0.028449	10022.0	1398.0	0.027621	0.036234	-0.271402	0.011407	0.074375	0
10	(31916.8, 34918.88]	10105	0.875309	0.025173	8845.0	1260.0	0.024377	0.032657	-0.292401	0.003244	0.020999	0
11	(34918.88, 37920.96]	16226	0.875385	0.040422	14204.0	2022.0	0.039147	0.052407	-0.291705	0.014770	0.000696	0
12	(37920.96, 40923.04]	20487	0.883145	0.051036	18093.0	2394.0	0.049865	0.062048	-0.218582	0.010718	0.073123	0
13	(40923.04, 43925.12]	14357	0.885422	0.035766	12712.0	1645.0	0.035035	0.042635	-0.196336	0.014830	0.022247	0
14	(43925.12, 46927.2]	18843	0.891631	0.046941	16801.0	2042.0	0.046305	0.052925	-0.133633	0.011270	0.062703	0
15	(46927.2, 49929.28]	13101	0.890543	0.032637	11667.0	1434.0	0.032155	0.037167	-0.144845	0.014150	0.011212	0
16	(49929.28, 52931.36]	25534	0.897509	0.063609	22917.0	2617.0	0.063161	0.067828	-0.071292	0.031006	0.073554	0
17	(52931.36, 55933.44]	18696	0.900620	0.046575	16838.0	1858.0	0.046407	0.048156	-0.037004	0.016754	0.034287	0
18	(55933.44, 58935.52]	11779	0.898888	0.029343	10588.0	1191.0	0.029181	0.030869	-0.056214	0.017225	0.019210	0
19	(58935.52, 61937.6]	21261	0.894502	0.052964	19018.0	2243.0	0.052415	0.058134	-0.103570	0.023234	0.047356	0
20	(61937.6, 64939.68]	10571	0.910983	0.026334	9630.0	941.0	0.026541	0.024389	0.084553	0.025874	0.188124	0
21	(64939.68, 67941.76]	18032	0.904669	0.044921	16313.0	1719.0	0.044960	0.044553	0.009078	0.018419	0.075476	0
22	(67941.76, 70943.84]	17796	0.909362	0.044333	16183.0	1613.0	0.044601	0.041806	0.064724	0.000358	0.055646	0
23	(70943.84, 73945.92]	9457	0.913820	0.023559	8642.0	815.0	0.023818	0.021123	0.120059	0.020783	0.055336	0
24	(73945.92, 76948.0]	15191	0.915805	0.037843	13912.0	1279.0	0.038342	0.033149	0.145531	0.014524	0.025472	0
25	(76948.0, 79950.08]	6432	0.916978	0.016023	5898.0	534.0	0.016255	0.013840	0.160831	0.022087	0.015299	0
26	(79950.08, 82952.16]	15317	0.918195	0.038157	14064.0	1253.0	0.038761	0.032475	0.176936	0.022506	0.016105	0
27	(82952.16, 85954.24]	11749	0.920844	0.029269	10819.0	930.0	0.029818	0.024104	0.212733	0.008943	0.035797	0
28	(85954.24, 88956.32]	5351	0.924126	0.013330	4945.0	406.0	0.013629	0.010523	0.258637	0.016189	0.045905	0
29	(88956.32, 91958.4]	11100	0.926486	0.027652	10284.0	816.0	0.028343	0.021149	0.292788	0.014715	0.034151	0
30	(91958.4, 94960.48]	5186	0.924605	0.012919	4795.0	391.0	0.013215	0.010134	0.265479	0.015128	0.027309	0
31	(94960.48, 97962.56]	7978	0.927551	0.019874	7400.0	578.0	0.020395	0.014981	0.308519	0.007180	0.043040	0

32	(97962.56, 100964.64]	11086	0.927476	0.027617	10282.0	804.0	0.028338	0.020838	0.307409	0.007943	0.001111	0
33	(100964.64, 103966.72]	3409	0.930478	0.008492	3172.0	237.0	0.008742	0.006143	0.352916	0.019596	0.045507	0
34	(103966.72, 106968.8]	5133	0.936489	0.012787	4807.0	326.0	0.013248	0.008449	0.449789	0.004506	0.096874	0
35	(106968.8, 109970.88]	2644	0.936460	0.006587	2476.0	168.0	0.006824	0.004354	0.449294	0.006424	0.000495	0
36	(109970.88, 112972.96]	6513	0.927376	0.016225	6040.0	473.0	0.016647	0.012259	0.305922	0.009823	0.143372	0
37	(112972.96, 115975.04]	3655	0.945280	0.009105	3455.0	200.0	0.009522	0.005184	0.608118	0.007124	0.302196	0
38	(115975.04, 118977.12]	1765	0.939377	0.004397	1658.0	107.0	0.004570	0.002773	0.499397	0.004953	0.108722	0
39	(118977.12, 121979.2]	6633	0.930801	0.016524	6174.0	459.0	0.017016	0.011896	0.357910	0.012446	0.141487	0
40	(121979.2, 124981.28]	1330	0.942105	0.003313	1253.0	77.0	0.003453	0.001996	0.548349	0.013563	0.190439	0
41	(124981.28, 127983.36]	3955	0.935525	0.009853	3700.0	255.0	0.010197	0.006609	0.433683	0.006744	0.114666	0
42	(127983.36, 130985.44]	3641	0.941774	0.009070	3429.0	212.0	0.009451	0.005495	0.542296	0.000747	0.108613	0
43	(130985.44, 133987.52]	923	0.937161	0.002299	865.0	58.0	0.002384	0.001503	0.461145	0.007067	0.081151	0
44	(133987.52, 136989.6]	2109	0.950688	0.005254	2005.0	104.0	0.005526	0.002695	0.717867	0.003142	0.256722	0
45	(136989.6, 139991.68]	735	0.938776	0.001831	690.0	45.0	0.001902	0.001166	0.488887	0.003624	0.228979	0
46	(139991.68, 142993.76]	2757	0.933261	0.006868	2573.0	184.0	0.007091	0.004769	0.396750	0.005190	0.092137	0
47	(142993.76, 145995.84]	1558	0.932606	0.003881	1453.0	105.0	0.004005	0.002721	0.386283	0.003087	0.010467	0
48	(145995.84, 148997.92]	596	0.934564	0.001485	557.0	39.0	0.001535	0.001011	0.417862	0.002469	0.031578	0
49	(148997.92, 152000.0]	3934	0.933655	0.009800	3673.0	261.0	0.010123	0.006765	0.403102	0.008588	0.014760	0

Maintenant, nous voyons que le poids des observations a plus de sens. Nous voyons plusieurs groupes avec des poids entre 1 et 6 %. Graphiquons les PdE.

In [200…

```
graph_PdE_V(df)
```



Dans le graphique, nous pouvons observer que le PdE augmente avec le revenu.

Analyse du graphique PdE du Revenu Annuel

À partir du graphique, nous pouvons observer une relation positive entre le revenu et le Poids de la Preuve (PdE). Autrement dit, plus le revenu est élevé, plus le PdE est élevé. Dans les cas où il y a des croissances ou des décroissances monotones, c'est-à-dire une ligne droite comme celle que nous observons, nous pouvons en toute sécurité diviser la variable en intervalles de taille égale, en tenant bien sûr compte du poids des observations. Par exemple, nous pourrions diviser les revenus de 0 à 152 000 en intervalles de 10 000 $. Cependant, les deux premiers intervalles auraient très peu d'observations. Ainsi, nous pourrions définir le premier intervalle de 0 à 20K, ou en d'autres termes, les revenus inférieurs à 20K. En suivant le même raisonnement, les quatre derniers intervalles auraient très peu d'observations. Par conséquent, nous pourrions définir deux intervalles de 26K, c'est-à-dire de 100K à 126K pour l'un et plus de 125K à 152K pour l'autre. Et les intervalles du milieu, qui ont le plus grand nombre d'observations, pourraient être regroupés par intervalles de 10K.

Résumé :

0-20K 20K-30K 30K-40K 40K-50K ... 80K-90K 90K-100K 100K-126K 126K-152K Nous devrions inclure dans nos catégories :

152K-227K 227K

In [201…
```
x_train['Revenu_<20K']= np.where(x_train['annual_inc'] <= 20000, 1, 0)
```

In [202…
```
x_train['Revenu_20K-30K']= np.where(x_train['annual_inc'].isin(range(20000,30000)), 1, 0)
```

In [203…
```
x_train['Revenu_30K-40K']= np.where(x_train['annual_inc'].isin(range(30000,40000)), 1, 0)
x_train['Revenu_40K-50K']= np.where(x_train['annual_inc'].isin(range(40000,50000)), 1, 0)
x_train['Revenu_50K-60K']= np.where(x_train['annual_inc'].isin(range(50000,60000)), 1, 0)
x_train['Revenu_60K-70K']= np.where(x_train['annual_inc'].isin(range(60000,70000)), 1, 0)
x_train['Revenu_70K-80K']= np.where(x_train['annual_inc'].isin(range(70000,80000)), 1, 0)
x_train['Revenu_80K-90K']= np.where(x_train['annual_inc'].isin(range(80000,90000)), 1, 0)
x_train['Revenu_90K-100K']= np.where(x_train['annual_inc'].isin(range(90000,100000)), 1, 0)
x_train['Revenu_100K-126K']= np.where(x_train['annual_inc'].isin(range(100000,126000)), 1, 0)
x_train['Revenu_126K-152K']= np.where(x_train['annual_inc'].isin(range(126000,152000)), 1, 0)
x_train['Revenu_152K-227K']= np.where(x_train['annual_inc'].isin(range(152000,227000)), 1, 0)
x_train['Revenu_>227K']= np.where(x_train['annual_inc'] > 227000, 1, 0)
```

mths_since_last_delinq

Mois du dernier impayé

La particularité de cette variable est que plusieurs clients n'ont pas de valeur numérique simplement parce qu'ils n'ont jamais été en défaut de paiement. Alors, la première chose que nous devons faire est de sélectionner les valeurs "not null", qui seront prétraitées comme une variable continue. Plus tard, nous inclurons toutes les valeurs null en tant que variable catégorique.

Créons un dataframe temporaire avec les variables indépendantes qui ont une valeur numérique dans la variable 'mths_since_last_delinq'.

In [204…
```
x_temp = x_train[pd.notnull(x_train['mths_since_last_delinq'])]
```

In [205…
```
x_temp['mths_since_last_delinq_categ'] = pd.cut(x_temp['mths_since_last_delinq'], 50)
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_21276\4149754453.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  x_temp['mths_since_last_delinq_categ'] = pd.cut(x_temp['mths_since_last_delinq'], 50)
```

Créons la table PdE_continue Nous utiliserons les variables indépendantes où 'mths_since_last_delinq' a des valeurs numériques Nous sélectionnerons également les valeurs de la variable dépendante uniquement pour les observations où 'mths_since_last_delinq' a une valeur numérique

In [206…
```
df = PdE_continue(x_temp, 'mths_since_last_delinq_categ', y_train[x_temp.index])
```

In [207…
```
df
```

Out[207]:

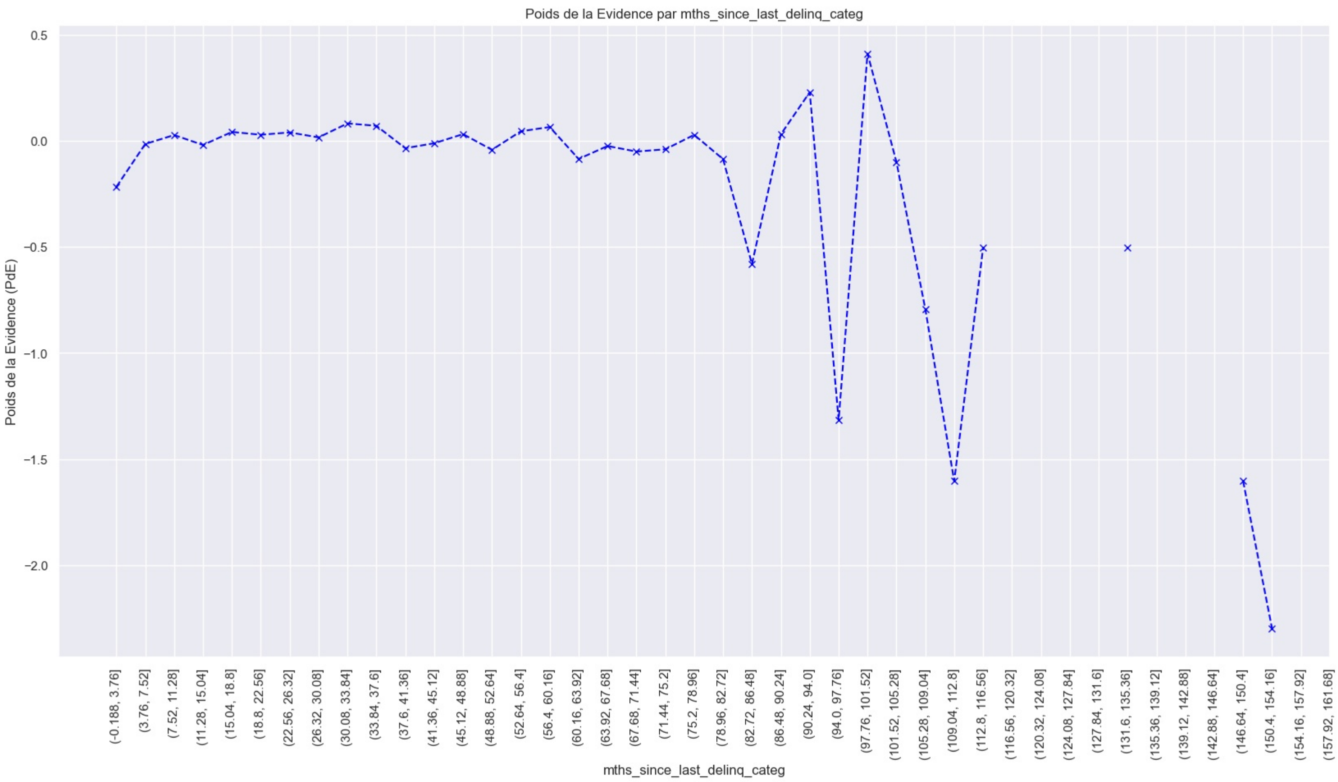
	mths_since_last_delinq_categ	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons
0	(-0.188, 3.76]	5950	0.888908	0.030630	5289.0	661.0	0.029964	0.037246	-0.217530	NaN
1	(3.76, 7.52]	12435	0.907358	0.064013	11283.0	1152.0	0.063923	0.064912	-0.015363	0.033958

2	(7.52, 11.28]	14984	0.910838	0.077135	13648.0	1336.0	0.077321	0.075280	0.026752	0.013399
3	(11.28, 15.04]	14940	0.906961	0.076908	13550.0	1390.0	0.076766	0.078323	-0.020078	0.000555
4	(15.04, 18.8]	10446	0.912024	0.053774	9527.0	919.0	0.053974	0.051783	0.041438	0.022792
5	(18.8, 22.56]	13104	0.910943	0.067457	11937.0	1167.0	0.067628	0.065758	0.028045	0.013654
6	(22.56, 26.32]	12306	0.911832	0.063349	11221.0	1085.0	0.063571	0.061137	0.039046	0.004056
7	(26.32, 30.08]	11890	0.909924	0.061208	10819.0	1071.0	0.061294	0.060348	0.015550	0.002277
8	(30.08, 33.84]	8536	0.915183	0.043942	7812.0	724.0	0.044258	0.040796	0.081464	0.017036
9	(33.84, 37.6]	10982	0.914314	0.056533	10041.0	941.0	0.056886	0.053023	0.070328	0.012628
10	(37.6, 41.36]	10484	0.905666	0.053970	9495.0	989.0	0.053793	0.055728	-0.035335	0.003093
11	(41.36, 45.12]	10318	0.907637	0.053115	9365.0	953.0	0.053056	0.053699	-0.012041	0.000737
12	(45.12, 48.88]	7319	0.911190	0.037677	6669.0	650.0	0.037783	0.036626	0.031092	0.015274
13	(48.88, 52.64]	7254	0.905018	0.037342	6565.0	689.0	0.037193	0.038823	-0.042895	0.000589
14	(52.64, 56.4]	6816	0.912265	0.035088	6218.0	598.0	0.035227	0.033696	0.044452	0.001966
15	(56.4, 60.16]	6582	0.913856	0.033883	6015.0	567.0	0.034077	0.031949	0.064491	0.001150
16	(60.16, 63.92]	4567	0.901248	0.023510	4116.0	451.0	0.023319	0.025413	-0.085991	0.010759
17	(63.92, 67.68]	5939	0.906550	0.030573	5384.0	555.0	0.030503	0.031273	-0.024942	0.007184
18	(67.68, 71.44]	5790	0.904318	0.029806	5236.0	554.0	0.029664	0.031217	-0.051013	0.000838
19	(71.44, 75.2]	5508	0.905229	0.028354	4986.0	522.0	0.028248	0.029413	-0.040439	0.001416
20	(75.2, 78.96]	3805	0.910907	0.019587	3466.0	339.0	0.019636	0.019102	0.027595	0.008611
21	(78.96, 82.72]	4017	0.901170	0.020679	3620.0	397.0	0.020509	0.022370	-0.086868	0.000872
22	(82.72, 86.48]	105	0.847619	0.000541	89.0	16.0	0.000504	0.000902	-0.581113	0.020005
23	(86.48, 90.24]	45	0.911111	0.000232	41.0	4.0	0.000232	0.000225	0.030117	0.000272
24	(90.24, 94.0]	27	0.925926	0.000139	25.0	2.0	0.000142	0.000113	0.228568	0.000091
25	(94.0, 97.76]	22	0.727273	0.000113	16.0	6.0	0.000091	0.000338	-1.316332	0.000051
26	(97.76, 101.52]	16	0.937500	0.000082	15.0	1.0	0.000085	0.000056	0.410889	0.000006
27	(101.52, 105.28]	10	0.900000	0.000051	9.0	1.0	0.000051	0.000056	-0.099936	0.000034
28	(105.28, 109.04]	11	0.818182	0.000057	9.0	2.0	0.000051	0.000113	-0.793084	0.000000
29	(109.04, 112.8]	3	0.666667	0.000015	2.0	1.0	0.000011	0.000056	-1.604014	0.000040
30	(112.8, 116.56]	7	0.857143	0.000036	6.0	1.0	0.000034	0.000056	-0.505402	0.000023
31	(116.56, 120.32]	4	1.000000	0.000021	4.0	0.0	0.000023	0.000000	NaN	0.000011
32	(120.32, 124.08]	4	1.000000	0.000021	4.0	0.0	0.000023	0.000000	NaN	0.000000
33	(124.08, 127.84]	2	1.000000	0.000010	2.0	0.0	0.000011	0.000000	NaN	0.000011
34	(127.84, 131.6]	5	1.000000	0.000026	5.0	0.0	0.000028	0.000000	NaN	0.000017
35	(131.6, 135.36]	7	0.857143	0.000036	6.0	1.0	0.000034	0.000056	-0.505402	0.000006
36	(135.36, 139.12]	2	1.000000	0.000010	2.0	0.0	0.000011	0.000000	NaN	0.000023
37	(139.12, 142.88]	3	1.000000	0.000015	3.0	0.0	0.000017	0.000000	NaN	0.000006
38	(142.88, 146.64]	2	1.000000	0.000010	2.0	0.0	0.000011	0.000000	NaN	0.000006
39	(146.64, 150.4]	3	0.666667	0.000015	2.0	1.0	0.000011	0.000056	-1.604014	0.000000

40	(150.4, 154.16]	2	0.500000	0.000010	1.0	1.0	0.000006	0.000056	-2.297161	0.000006
41	(154.16, 157.92]	1	1.000000	0.000005	1.0	0.0	0.000006	0.000000	NaN	0.000000
42	(157.92, 161.68]	1	1.000000	0.000005	1.0	0.0	0.000006	0.000000	NaN	0.000000
43	(161.68, 165.44]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
44	(165.44, 169.2]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
45	(169.2, 172.96]	1	1.000000	0.000005	1.0	0.0	0.000006	0.000000	NaN	NaN
46	(172.96, 176.72]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
47	(176.72, 180.48]	1	1.000000	0.000005	1.0	0.0	0.000006	0.000000	NaN	NaN
48	(180.48, 184.24]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
49	(184.24, 188.0]	1	1.000000	0.000005	1.0	0.0	0.000006	0.000000	NaN	NaN

In [208…
```
# Dans le tableau, nous remarquons qu'à partir du bin 22, la proportion d'observations est très basse.
```

In [209…
```
graph_PdE_V(df)
```



Maintenant, faisons la classification globale en fonction du graphique. Le premier groupe a un PdE beaucoup plus bas que les suivants, alors isolons-le. Ensuite, les sept groupes suivants ont un PdE similaire (jusqu'à 30.08). Nous pourrions faire la prochaine coupe ici à 60.16. Une autre coupe à 82.72. Ensuite, les derniers groupes commencent à faire un zigzag significatif en raison du faible poids d'observations, même avec plusieurs bins sans observations. Nous pourrions les regrouper tous. Résumé : Les manquants 0.188 - 3.76 3.76 - 30.08 30.80 - 60.16 60.16 - 82.74 82.74 jusqu'à la fin

In [210…
```
x_train['mths_since_last_delinq_null'] = np.where((x_train['mths_since_last_delinq'].isnull()), 1, 0)
x_train['mths_since_last_delinq_0-4'] = np.where(x_train['mths_since_last_delinq'].isin(range(0,4)), 1, 0)
x_train['mths_since_last_delinq_4-30'] = np.where(x_train['mths_since_last_delinq'].isin(range(4,30)), 1, 0)
x_train['mths_since_last_delinq_30-60'] = np.where(x_train['mths_since_last_delinq'].isin(range(30,60)), 1, 0)
x_train['mths_since_last_delinq_60-83'] = np.where(x_train['mths_since_last_delinq'].isin(range(60,83)), 1, 0)
x_train['mths_since_last_delinq_83+'] = np.where(x_train['mths_since_last_delinq'] > 83, 1, 0)
```

delinq_2yrs

Le nombre d'incidents de défaut de paiement de plus de 30 jours dans le dossier de crédit de l'emprunteur au cours des 2 dernières années

In [211…
```
x_train['delinq_2yrs'].unique()
```

Out[211]:
```
array([ 0.,  1.,  2.,  3.,  5., 10.,  8.,  6.,  4.,  7.,  9., 13., 12.,
       16., 11., 15., 17., 18., 14., 21., 22., 19., 29., 24.])
```

Comme ce sont des valeurs entières et qu'il n'y en a pas tant, nous n'avons pas besoin de faire de classification fine. Générons directement la table avec les poids de preuve.

In [212…
```
df = PdE_continue(x_train, 'delinq_2yrs', y_train)
```

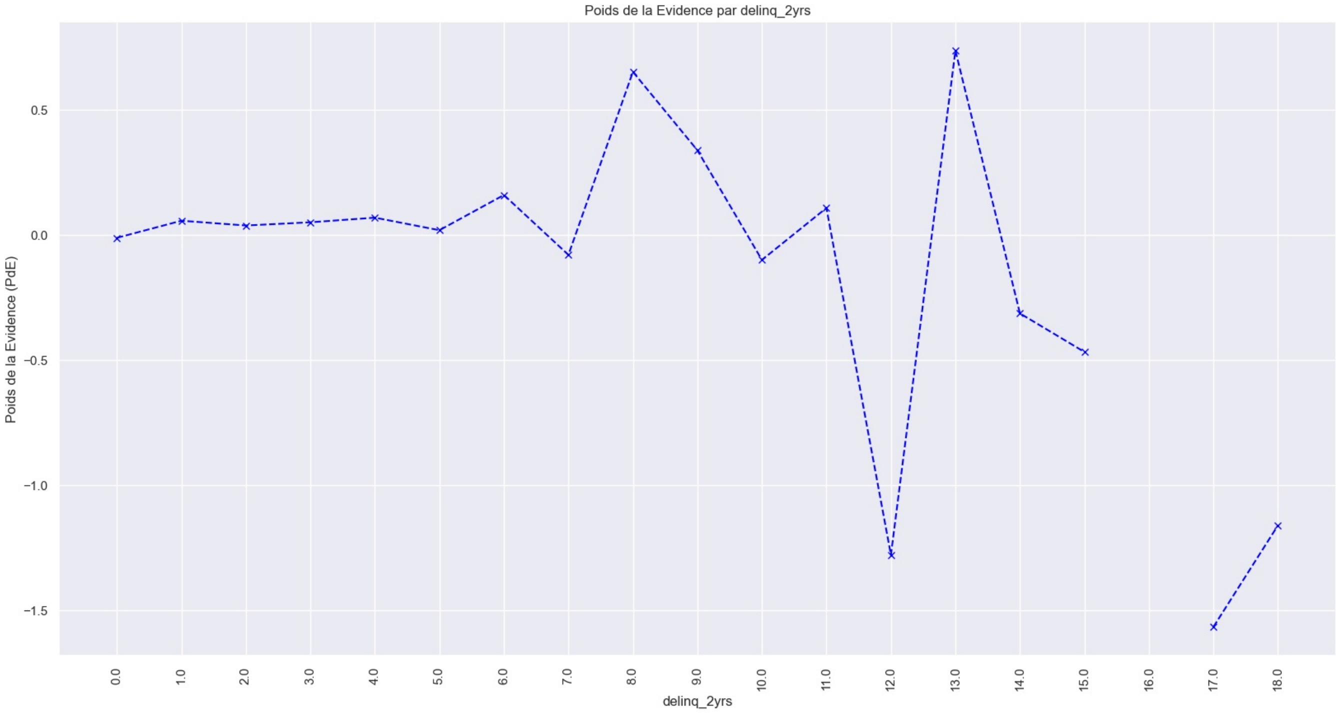
```
In [212]: df = PdE_continue(x_train, 'delinq_2yrs', y_train)
df
```

Out[212]:

	delinq_2yrs	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	
0	0.0	344728	0.904467	0.821454	311795.0	32933.0	0.820578	0.829839	-0.011223	NaN	NaN	1.0394
1	1.0	50548	0.910204	0.120451	46009.0	4539.0	0.121086	0.114373	0.057036	0.699492	0.068260	3.8288
2	2.0	14666	0.908632	0.034948	13326.0	1340.0	0.035071	0.033765	0.037954	0.086015	0.019083	4.9572
3	3.0	5116	0.909695	0.012191	4654.0	462.0	0.012248	0.011641	0.050824	0.022823	0.012870	3.0847
4	4.0	2151	0.911204	0.005126	1960.0	191.0	0.005158	0.004813	0.069333	0.007090	0.018509	2.3955
5	5.0	1119	0.907060	0.002666	1015.0	104.0	0.002671	0.002621	0.019159	0.002487	0.050173	9.7121
6	6.0	575	0.918261	0.001370	528.0	47.0	0.001390	0.001184	0.159855	0.001282	0.140696	3.2816
7	7.0	315	0.898413	0.000751	283.0	32.0	0.000745	0.000806	-0.079383	0.000645	0.239238	4.8847
8	8.0	155	0.948387	0.000369	147.0	8.0	0.000387	0.000202	0.651897	0.000358	0.731280	1.2079
9	9.0	101	0.930693	0.000241	94.0	7.0	0.000247	0.000176	0.338291	0.000139	0.313606	2.4019
10	10.0	58	0.896552	0.000138	52.0	6.0	0.000137	0.000151	-0.099610	0.000111	0.437900	1.4277
11	11.0	35	0.914286	0.000083	32.0	3.0	0.000084	0.000076	0.108030	0.000053	0.207639	9.3162
12	12.0	33	0.727273	0.000079	24.0	9.0	0.000063	0.000227	-1.278265	0.000021	1.386294	2.0914
13	13.0	21	0.952381	0.000050	20.0	1.0	0.000053	0.000025	0.736638	0.000011	2.014903	2.0211
14	14.0	8	0.875000	0.000019	7.0	1.0	0.000018	0.000025	-0.313184	0.000034	1.049822	2.1219
15	15.0	7	0.857143	0.000017	6.0	1.0	0.000016	0.000025	-0.467334	0.000003	0.154151	4.3962
16	16.0	5	1.000000	0.000012	5.0	0.0	0.000013	0.000000	NaN	0.000003	NaN	
17	17.0	3	0.666667	0.000007	2.0	1.0	0.000005	0.000025	-1.565947	0.000008	NaN	3.1215
18	18.0	4	0.750000	0.000010	3.0	1.0	0.000008	0.000025	-1.160482	0.000003	0.405465	2.0079
19	19.0	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	0.000003	NaN	
20	21.0	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	0.000000	NaN	
21	22.0	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	0.000000	NaN	
22	24.0	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000003	NaN	
23	29.0	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000000	NaN	

Nous notons que le groupe de 0 incidents de défaut de paiement a un poids significatif de 82%. À partir de 4 incidents de défaut de paiement, le poids des observations est inférieur à 1%. Traçons deux PdE.

```
In [213]: graph_PdE_V(df)
```



Groupes suggérés : 0 va seul en raison de sa proportion élevée 1-4 ont un PdE similaire 5 ou plus, commence à zigzaguer et la proportion d'observations est très faible

In [214…
```
x_train['impayé_2ans_0'] = np.where((x_train['delinq_2yrs'] == 0), 1, 0)
x_train['impayé_2ans_1-4'] = np.where((x_train['delinq_2yrs'] >= 1) & (x_train['delinq_2yrs'] <= 4), 1, 0)
x_train['impayé_2ans_>=5'] = np.where((x_train['delinq_2yrs'] >= 5), 1, 0)
```

total_acc

Le nombre total de lignes de crédit actuellement dans le dossier de crédit de l'emprunteur

In [215…
```
x_train['total_acc'].unique()
```

Out[215]:
```
array([ 27.,  21.,  39.,  63.,  26.,  22.,  38.,   9.,  32.,  20.,  12.,
        11.,  25.,  58.,  31.,  30.,  47.,  10.,  24.,  23.,  14.,  37.,
        34.,  18.,  35.,  13.,   8.,  29.,  54.,  48.,  16.,   7.,  45.,
        61.,  28.,   6.,  33.,  15.,  43.,  55.,  17.,  40.,  19.,  41.,
        46.,  60.,  49.,  42.,   5.,  56.,  44.,   4.,  36.,  57.,  50.,
        53.,  52.,  51.,  62.,   0.,  68.,  59.,   3.,  65.,  73.,  83.,
        70.,  67.,  76.,  93.,  81.,  90.,  77.,  64.,  74.,  69., 118.,
         2.,  94.,  80.,  72.,  66.,   1.,  71.,  78.,  82., 119., 124.,
        85., 101., 106.,  79.,  88.,  84.,  89.,  75.,  87., 105.,  95.,
        92.,  86.,  91., 150.,  97., 116., 102., 156.,  96.,  99.,  98.,
       121.])
```

Comme il y a de nombreuses valeurs uniques, commençons par faire une classification fine

In [216…
```
x_train['total_acc_categ'] = pd.cut(x_train['total_acc'], 50)
```

In [217…
```
df = PdE_continue(x_train, 'total_acc_categ', y_train)
df
```

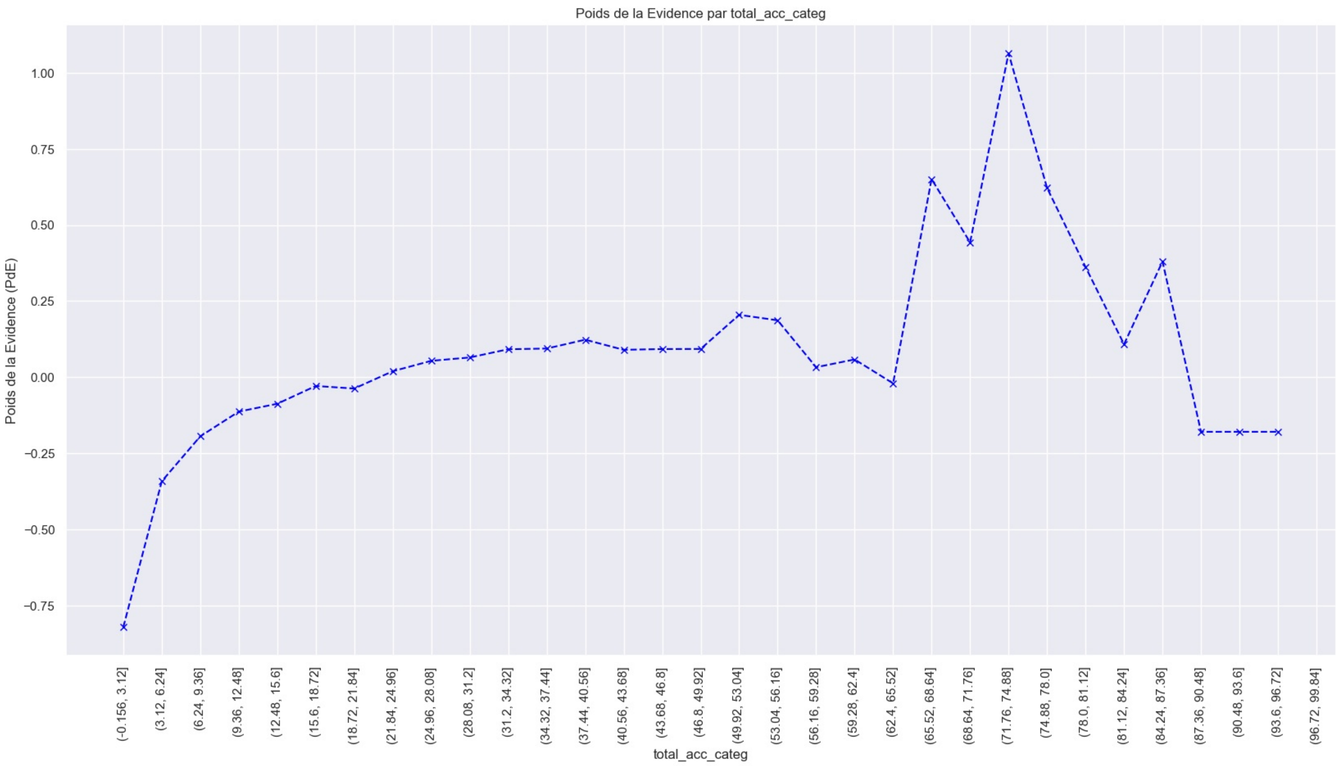
Out[217]:

	total_acc_categ	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE
0	(-0.156, 3.12]	516	0.808140	0.001230	417.0	99.0	0.001097	0.002495	-0.821127	NaN	NaN
1	(3.12, 6.24]	6763	0.871802	0.016116	5896.0	867.0	0.015517	0.021846	-0.342103	0.014420	4.790241e-01
2	(6.24, 9.36]	16957	0.887421	0.040407	15048.0	1909.0	0.039603	0.048103	-0.194428	0.024086	1.476751e-01
3	(9.36, 12.48]	28247	0.895281	0.067310	25289.0	2958.0	0.066555	0.074535	-0.113238	0.026952	8.119061e-02
4	(12.48, 15.6]	37867	0.897668	0.090233	33992.0	3875.0	0.089460	0.097641	-0.087514	0.022904	2.572339e-02
5	(15.6, 18.72]	43601	0.902915	0.103897	39368.0	4233.0	0.103608	0.106662	-0.029051	0.014148	5.846280e-02
6	(18.72, 21.84]	45796	0.902175	0.109127	41316.0	4480.0	0.108735	0.112886	-0.037467	0.005127	8.415554e-03
7	(21.84, 24.96]	45116	0.907084	0.107507	40924.0	4192.0	0.107703	0.105629	0.019445	0.001032	5.691195e-02

8	(24.96, 28.08]	53277	0.909924	0.126954	48478.0	4799.0	0.127584	0.120924	0.053609	0.019881	3.416378e-02
9	(28.08, 31.2]	33461	0.910822	0.079734	30477.0	2984.0	0.080209	0.075190	0.064614	0.047375	1.100509e-02
10	(31.2, 34.32]	27263	0.912996	0.064965	24891.0	2372.0	0.065508	0.059769	0.091679	0.014701	2.706519e-02
11	(34.32, 37.44]	21435	0.913179	0.051078	19574.0	1861.0	0.051515	0.046893	0.093994	0.013993	2.315366e-03
12	(37.44, 40.56]	16418	0.915459	0.039123	15030.0	1388.0	0.039556	0.034975	0.123091	0.011959	2.909616e-02
13	(40.56, 43.68]	12506	0.912842	0.029801	11416.0	1090.0	0.030044	0.027466	0.089744	0.009511	3.334616e-02
14	(43.68, 46.8]	9143	0.913048	0.021787	8348.0	795.0	0.021970	0.020032	0.092341	0.008074	2.596970e-03
15	(46.8, 49.92]	6534	0.913070	0.015570	5966.0	568.0	0.015701	0.014312	0.092617	0.006269	2.753928e-04
16	(49.92, 53.04]	5852	0.921565	0.013945	5393.0	459.0	0.014193	0.011566	0.204713	0.001508	1.120963e-01
17	(53.04, 56.16]	2809	0.920256	0.006694	2585.0	224.0	0.006803	0.005644	0.186741	0.007390	1.797215e-02
18	(56.16, 59.28]	2004	0.908184	0.004775	1820.0	184.0	0.004790	0.004636	0.032562	0.002013	1.541787e-01
19	(59.28, 62.4]	1661	0.910295	0.003958	1512.0	149.0	0.003979	0.003754	0.058148	0.000811	2.558623e-02
20	(62.4, 65.52]	1557	0.903661	0.003710	1407.0	150.0	0.003703	0.003780	-0.020514	0.000276	7.866249e-02
21	(65.52, 68.64]	271	0.948339	0.000646	257.0	14.0	0.000676	0.000353	0.650925	0.003027	6.714390e-01
22	(68.64, 71.76]	175	0.937143	0.000417	164.0	11.0	0.000432	0.000277	0.442877	0.000245	2.080476e-01
23	(71.76, 74.88]	115	0.965217	0.000274	111.0	4.0	0.000292	0.000101	1.064142	0.000139	6.212647e-01
24	(74.88, 78.0]	113	0.946903	0.000269	107.0	6.0	0.000282	0.000151	0.621976	0.000011	4.421665e-01
25	(78.0, 81.12]	59	0.932203	0.000141	55.0	4.0	0.000145	0.000101	0.361945	0.000137	2.600305e-01
26	(81.12, 84.24]	35	0.914286	0.000083	32.0	3.0	0.000084	0.000076	0.108030	0.000061	2.539152e-01
27	(84.24, 87.36]	30	0.933333	0.000071	28.0	2.0	0.000074	0.000050	0.379964	0.000011	2.719337e-01
28	(87.36, 90.48]	27	0.888889	0.000064	24.0	3.0	0.000063	0.000076	-0.179652	0.000011	5.596158e-01
29	(90.48, 93.6]	18	0.888889	0.000043	16.0	2.0	0.000042	0.000050	-0.179652	0.000021	1.387779e-16
30	(93.6, 96.72]	9	0.888889	0.000021	8.0	1.0	0.000021	0.000025	-0.179652	0.000021	0.000000e+00
31	(96.72, 99.84]	6	1.000000	0.000014	6.0	0.0	0.000016	0.000000	NaN	0.000005	NaN
32	(99.84, 102.96]	3	1.000000	0.000007	3.0	0.0	0.000008	0.000000	NaN	0.000008	NaN
33	(102.96, 106.08]	5	1.000000	0.000012	5.0	0.0	0.000013	0.000000	NaN	0.000005	NaN
34	(106.08, 109.2]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
35	(109.2, 112.32]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
36	(112.32, 115.44]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
37	(115.44, 118.56]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	NaN	NaN
38	(118.56, 121.68]	2	1.000000	0.000005	2.0	0.0	0.000005	0.000000	NaN	0.000000	NaN
39	(121.68, 124.8]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000003	NaN
40	(124.8, 127.92]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
41	(127.92, 131.04]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
42	(131.04, 134.16]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
43	(134.16, 137.28]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
44	(137.28, 140.4]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
45	(140.4, 143.52]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
46	(143.52, 146.64]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
47	(146.64, 149.76]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN	NaN
48	(149.76, 152.88]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	NaN	NaN
49	(152.88, 156.0]	1	1.000000	0.000002	1.0	0.0	0.000003	0.000000	NaN	0.000000	NaN

Les intervalles avec plus de 53 lignes de crédit ont un poids d'observations insignifiant.

In [218…
```
graph_PdE_V(df)
```



Les deux premières catégories entre 0 et 6.24 ont un PdE très bas et un poids d'observations très faible.

Entre 6.24 et 21.84.

Entre 21.84 et 49.92.

À partir de 49.92, commence à zigzaguer fortement à cause de la faible quantité d'observations

In [219…
```
x_train['total_acc_<=6'] = np.where((x_train['total_acc'] <= 6), 1, 0)
x_train['total_acc_6-22'] = np.where((x_train['total_acc'] > 6) & (x_train['total_acc'] <= 22), 1, 0)
x_train['total_acc_22-50'] = np.where((x_train['total_acc'] > 22) & (x_train['total_acc'] <= 50), 1, 0)
x_train['total_acc_>50'] = np.where((x_train['total_acc'] > 50), 1, 0)
```

dti

Ratio de dette sur le revenu Comme il s'agit d'un ratio, nous nous attendons à une variable continue, nous devons donc effectuer une classification fine

In [220…
```
x_train['dti'].unique()
```

Out[220]:
```
array([14.85, 16.5 , 21.89, ..., 38.21, 35.29, 38.39])
```

In [221…
```
x_train['dti_categ'] = pd.cut(x_train['dti'], 50)
```

In [222…
```
df = PdE_continue(x_train, 'dti_categ', y_train)
df
```

Out[222]:

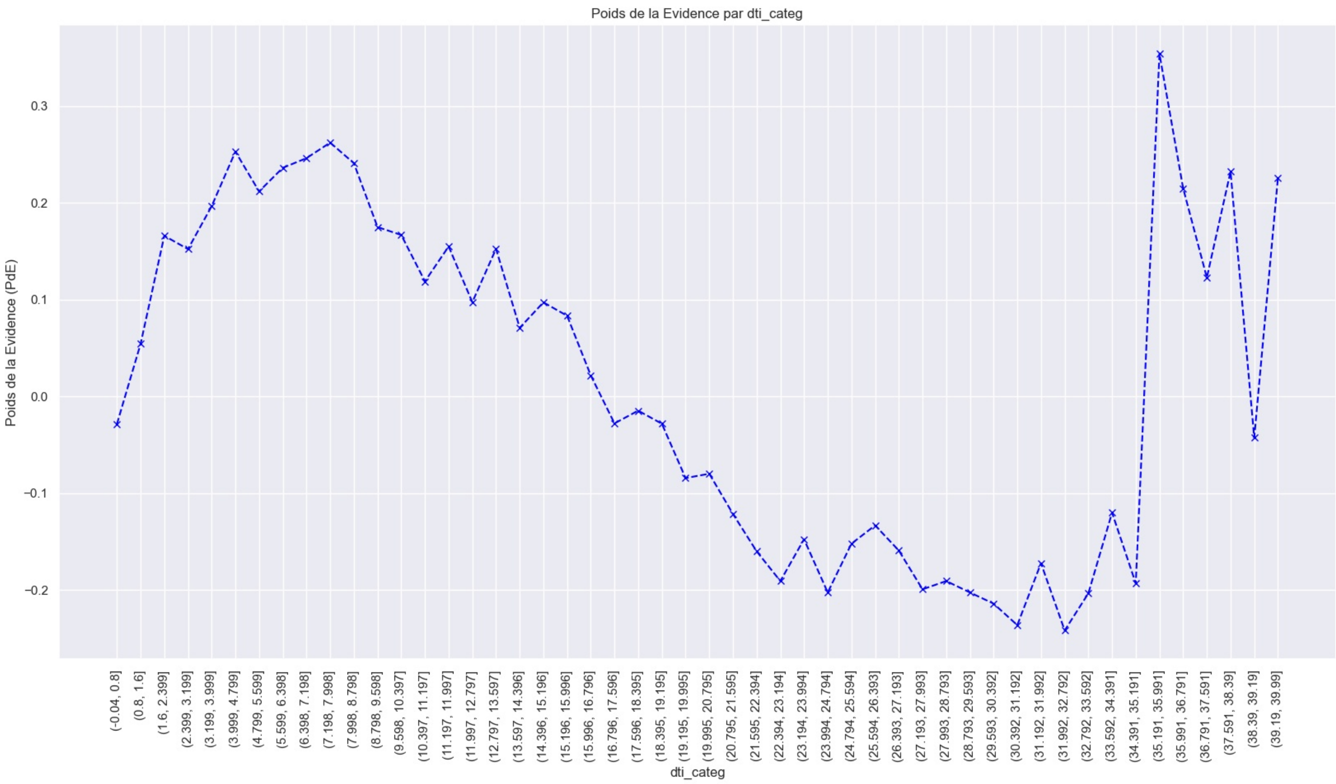
	dti_categ	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons	delta_PdE	VI
0	(-0.04, 0.8]	1752	0.902968	0.004175	1582.0	170.0	0.004163	0.004284	-0.028447	NaN	NaN	0.000003
1	(0.8, 1.6]	2211	0.909995	0.005269	2012.0	199.0	0.005295	0.005014	0.054486	0.001132	0.082933	0.000015
2	(1.6, 2.399]	2830	0.918728	0.006744	2600.0	230.0	0.006843	0.005795	0.166094	0.001547	0.111608	0.000174
3	(2.399, 3.199]	3657	0.917692	0.008714	3356.0	301.0	0.008832	0.007585	0.152301	0.001990	0.013793	0.000190
4	(3.199, 3.999]	4607	0.920990	0.010978	4243.0	364.0	0.011167	0.009172	0.196778	0.002334	0.044477	0.000393
5	(3.999, 4.799]	5666	0.924991	0.013502	5241.0	425.0	0.013793	0.010709	0.253085	0.002627	0.056306	0.000781
6	(4.799, 5.599]	6751	0.922086	0.016087	6225.0	526.0	0.016383	0.013254	0.211934	0.002590	0.041151	0.000663

7	(5.599, 6.398]	7666	0.923819	0.018267	7082.0	584.0	0.018638	0.014716	0.236317	0.002255	0.024383	0.000927
8	(6.398, 7.198]	8691	0.924520	0.020710	8035.0	656.0	0.021146	0.016530	0.246308	0.002508	0.009991	0.001137
9	(7.198, 7.998]	9897	0.925634	0.023584	9161.0	736.0	0.024110	0.018546	0.262387	0.002963	0.016079	0.001460
10	(7.998, 8.798]	10693	0.924156	0.025480	9882.0	811.0	0.026007	0.020435	0.241108	0.001898	0.021278	0.001343
11	(8.798, 9.598]	11563	0.919398	0.027554	10631.0	932.0	0.027979	0.023484	0.175103	0.001971	0.066005	0.000787
12	(9.598, 10.397]	12511	0.918791	0.029813	11495.0	1016.0	0.030252	0.025601	0.166945	0.002274	0.008158	0.000777
13	(10.397, 11.197]	13529	0.915145	0.032238	12381.0	1148.0	0.032584	0.028927	0.119048	0.002332	0.047897	0.000435
14	(11.197, 11.997]	14068	0.917899	0.033523	12913.0	1155.0	0.033984	0.029103	0.155040	0.001400	0.035992	0.000757
15	(11.997, 12.797]	14929	0.913457	0.035574	13637.0	1292.0	0.035890	0.032556	0.097501	0.001905	0.057539	0.000325
16	(12.797, 13.597]	15106	0.917715	0.035996	13863.0	1243.0	0.036484	0.031321	0.152602	0.000595	0.055100	0.000788
17	(13.597, 14.396]	15407	0.911339	0.036713	14041.0	1366.0	0.036953	0.034420	0.071001	0.000468	0.081601	0.000180
18	(14.396, 15.196]	15793	0.913443	0.037633	14426.0	1367.0	0.037966	0.034445	0.097320	0.001013	0.026319	0.000343
19	(15.196, 15.996]	15640	0.912340	0.037269	14269.0	1371.0	0.037553	0.034546	0.083455	0.000413	0.013865	0.000251
20	(15.996, 16.796]	15347	0.907278	0.036570	13924.0	1423.0	0.036645	0.035856	0.021753	0.000908	0.061702	0.000017
21	(16.796, 17.596]	15458	0.903028	0.036835	13959.0	1499.0	0.036737	0.037772	-0.027768	0.000092	0.049520	0.000029
22	(17.596, 18.395]	15278	0.904176	0.036406	13814.0	1464.0	0.036356	0.036890	-0.014584	0.000382	0.013184	0.000008
23	(18.395, 19.195]	14793	0.902995	0.035250	13358.0	1435.0	0.035155	0.036159	-0.028143	0.001200	0.013560	0.000028
24	(19.195, 19.995]	14711	0.897968	0.035055	13210.0	1501.0	0.034766	0.037822	-0.084251	0.000390	0.056108	0.000257
25	(19.995, 20.795]	14042	0.898376	0.033461	12615.0	1427.0	0.033200	0.035957	-0.079782	0.001566	0.004470	0.000220
26	(20.795, 21.595]	13348	0.894516	0.031807	11940.0	1408.0	0.031424	0.035479	-0.121370	0.001776	0.041588	0.000492
27	(21.595, 22.394]	12731	0.890818	0.030337	11341.0	1390.0	0.029847	0.035025	-0.159973	0.001576	0.038603	0.000828
28	(22.394, 23.194]	12281	0.887794	0.029264	10903.0	1378.0	0.028694	0.034723	-0.190689	0.001153	0.030716	0.001150
29	(23.194, 23.994]	11540	0.892028	0.027499	10294.0	1246.0	0.027092	0.031396	-0.147471	0.001603	0.043218	0.000635
30	(23.994, 24.794]	11042	0.886615	0.026312	9790.0	1252.0	0.025765	0.031548	-0.202475	0.001326	0.055004	0.001171
31	(24.794, 25.594]	9611	0.891583	0.022902	8569.0	1042.0	0.022552	0.026256	-0.152085	0.003213	0.050390	0.000563
32	(25.594, 26.393]	8743	0.893400	0.020834	7811.0	932.0	0.020557	0.023484	-0.133138	0.001995	0.018946	0.000390
33	(26.393, 27.193]	8130	0.890898	0.019373	7243.0	887.0	0.019062	0.022350	-0.159148	0.001495	0.026010	0.000523
34	(27.193, 27.993]	7536	0.886943	0.017958	6684.0	852.0	0.017591	0.021469	-0.199208	0.001471	0.040060	0.000772
35	(27.993, 28.793]	7130	0.887798	0.016990	6330.0	800.0	0.016659	0.020158	-0.190650	0.000932	0.008558	0.000667
36	(28.793, 29.593]	6597	0.886615	0.015720	5849.0	748.0	0.015393	0.018848	-0.202471	0.001266	0.011821	0.000699
37	(29.593, 30.392]	5334	0.885452	0.012710	4723.0	611.0	0.012430	0.015396	-0.213991	0.002963	0.011520	0.000635
38	(30.392, 31.192]	4357	0.883176	0.010382	3848.0	509.0	0.010127	0.012826	-0.236233	0.002303	0.022242	0.000637
39	(31.192, 31.992]	3950	0.889620	0.009412	3514.0	436.0	0.009248	0.010986	-0.172226	0.000879	0.064007	0.000299
40	(31.992, 32.792]	3433	0.882610	0.008181	3030.0	403.0	0.007974	0.010155	-0.241712	0.001274	0.069487	0.000527
41	(32.792, 33.592]	3199	0.886527	0.007623	2836.0	363.0	0.007464	0.009147	-0.203347	0.000511	0.038366	0.000342
42	(33.592,	2933	0.894647	0.006989	2624.0	309.0	0.006906	0.007786	-0.119980	0.000558	0.083367	0.000106

	34.391]												
43	(34.391, 35.191]	2081	0.887554	0.004959	1847.0	234.0	0.004861	0.005896	-0.193097	0.002045	0.073117	0.000200
44	(35.191, 35.991]	615	0.931707	0.001465	573.0	42.0	0.001508	0.001058	0.354122	0.003353	0.547219	0.000159
45	(35.991, 36.791]	605	0.922314	0.001442	558.0	47.0	0.001469	0.001184	0.215118	0.000039	0.139005	0.000061
46	(36.791, 37.591]	544	0.915441	0.001296	498.0	46.0	0.001311	0.001159	0.122865	0.000158	0.092253	0.000019
47	(37.591, 38.39]	497	0.923541	0.001184	459.0	38.0	0.001208	0.000958	0.232370	0.000103	0.109505	0.000058
48	(38.39, 39.19]	407	0.901720	0.000970	367.0	40.0	0.000966	0.001008	-0.042611	0.000242	0.274982	0.000002
49	(39.19, 39.99]	416	0.923077	0.000991	384.0	32.0	0.001011	0.000806	0.225813	0.000045	0.268424	0.000046

In [223…

```
graph_PdE_V(df)
```



Les quatre premiers groupes ont un faible poids d'observations : 0 - 3.2 3.2 - 8.8 8.8 - 10.39 10.39 - 13.6 13.6 - 16 16 - 16.7 16.7 - 19.9 19.9 - 20.8 20.8 - 23.19 23.19 - 35.19 À partir de 35.19, il y a une augmentation considérable du PdE et des catégories avec un poids d'observations très faible.

In [224…

```
x_train['dti<=3.2'] = np.where((x_train['dti'] <= 3.2), 1, 0)
x_train['dti_3.2-8.8'] = np.where((x_train['dti'] > 3.2) & (x_train['dti'] <= 8.8), 1, 0)
x_train['dti_8.8-10.4'] = np.where((x_train['dti'] > 8.8) & (x_train['dti'] <= 10.4), 1, 0)
x_train['dti_10.4-13.6'] = np.where((x_train['dti'] > 10.4) & (x_train['dti'] <= 13.6), 1, 0)
x_train['dti_13.6-16.0'] = np.where((x_train['dti'] > 13.6) & (x_train['dti'] <= 16.0), 1, 0)
x_train['dti_16.0-16.7'] = np.where((x_train['dti'] > 16.0) & (x_train['dti'] <= 16.7), 1, 0)
x_train['dti_16.7-19.9'] = np.where((x_train['dti'] > 16.7) & (x_train['dti'] <= 19.9), 1, 0)
x_train['dti_19.9-20.8'] = np.where((x_train['dti'] > 19.9) & (x_train['dti'] <= 20.8), 1, 0)
x_train['dti_20.8-23.2'] = np.where((x_train['dti'] > 20.8) & (x_train['dti'] <= 23.2), 1, 0)
x_train['dti_23.2-35.2'] = np.where((x_train['dti'] > 23.2) & (x_train['dti'] <= 35.2), 1, 0)
x_train['dti>35.2'] = np.where((x_train['dti'] > 35.2), 1, 0)
```

'mths_since_last_record'

Le nombre de mois depuis la dernière inscription publique

In [225…

```
x_train['mths_since_last_record'].unique()
```

Out[225]:
```
array([ nan,  48.,  96.,  81.,  99.,  69.,  45., 103.,  17.,  22., 107.,
        49.,  64., 111., 104.,  25.,  79., 112.,  61.,  35.,  65.,  62.,
        53.,  70.,  78.,  50.,   7.,   0.,  92.,  41.,  23.,  85., 117.,
        88.,  60.,  93.,  89.,  46.,  90.,  33.,  67.,  56.,  52.,  87.,
        47., 102.,  37.,  84., 114., 109.,  94.,  27.,  91.,  66., 101.,
        72., 113., 106.,  58.,  51.,  20.,  38.,  82., 115.,  95.,  80.,
       108.,  39.,  57., 118.,  40.,  75.,  97.,  59.,  42., 110.,  21.,
        44.,  30.,  12.,  24., 116.,  68., 100.,  98., 119.,  43.,  29.,
        71.,  32.,  18., 105.,  36.,  86.,  73.,  74.,  54.,  63.,  76.,
        14.,  77.,  55.,   6.,  34.,   8.,  83.,  26.,  31.,  15.,  19.,
         5.,  16.,  10.,  28.,  11.,  13.,   4.,   1.,   3.,   9., 121.,
         2., 120., 129.])
```

La première chose que nous remarquons est qu'il y a des valeurs "nan" avec lesquelles nous devons composer avant de faire la classification fine.

In [226…
```
df_temp = x_train[pd.notnull(x_train['mths_since_last_record'])]
```

Maintenant, effectuons la classification fine à partir des valeurs numériques dans df_temp.

In [227…
```
df_temp['mths_since_last_record_categ'] = pd.cut(df_temp['mths_since_last_record'], 50)
```

```
C:\Users\IDEAPAD5\AppData\Local\Temp\ipykernel_21276\228590877.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#ret
urning-a-view-versus-a-copy
  df_temp['mths_since_last_record_categ'] = pd.cut(df_temp['mths_since_last_record'], 50)
```

index sur df_temp pour sélectionner du dataframe y_train uniquement les observations avec des valeurs numériques dans la variable 'mths_since_last_record'.

In [228…
```
df = PdE_continue(df_temp, 'mths_since_last_record_categ', y_train[df_temp.index])
df
```

```
C:\Users\IDEAPAD5\anaconda3\lib\site-packages\pandas\core\arraylike.py:397: RuntimeWarning: divide by zero enco
untered in log
  result = getattr(ufunc, method)(*inputs, **kwargs)
```

Out[228]:

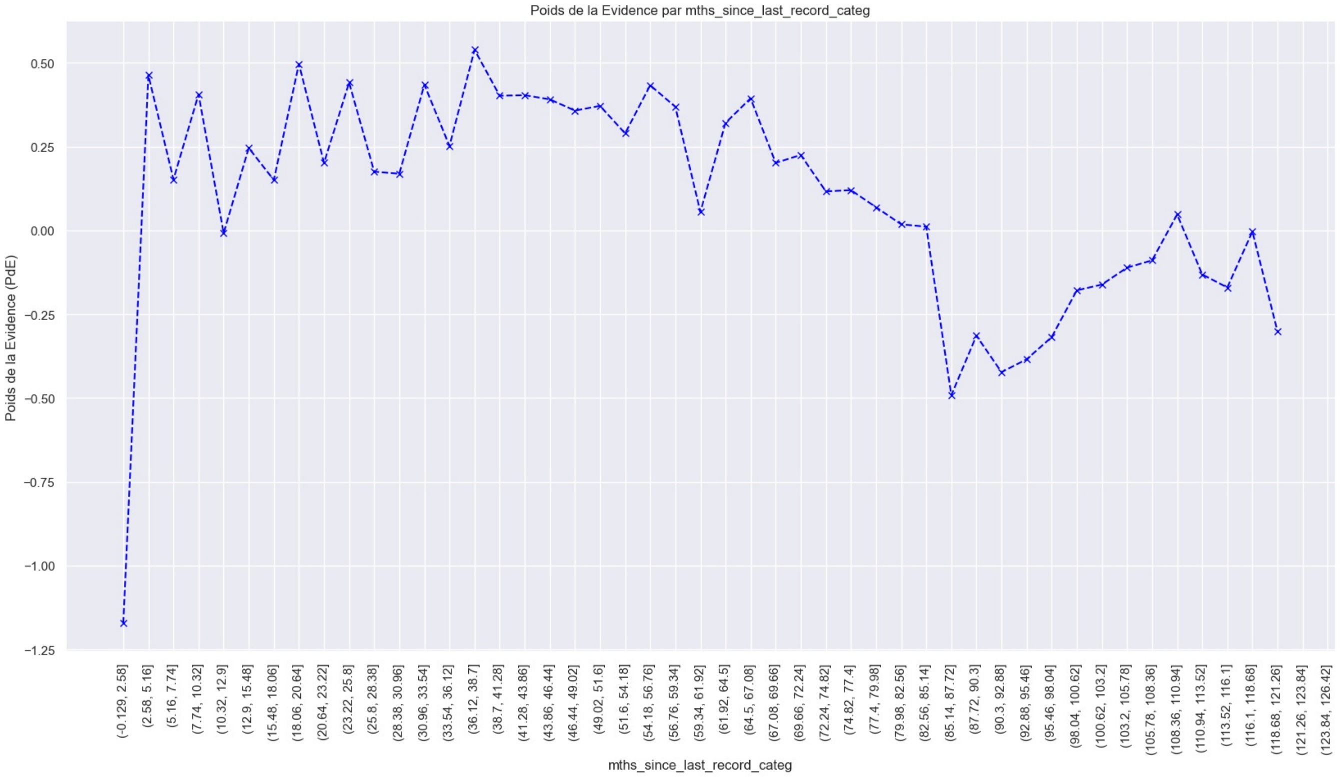
	mths_since_last_record_categ	observations	%_bon	%_obs	n_bons	n_malus	%_n_bons	%_n_malus	PdE	delta_%_n_bons
0	(-0.129, 2.58]	1180	0.762712	0.020979	900.0	280.0	0.017542	0.056657	-1.172413	NaN
1	(2.58, 5.16]	140	0.942857	0.002489	132.0	8.0	0.002573	0.001619	0.463342	0.014969
2	(5.16, 7.74]	144	0.923611	0.002560	133.0	11.0	0.002592	0.002226	0.152436	0.000019
3	(7.74, 10.32]	282	0.939716	0.005014	265.0	17.0	0.005165	0.003440	0.406498	0.002573
4	(10.32, 12.9]	226	0.911504	0.004018	206.0	20.0	0.004015	0.004047	-0.007874	0.001150
5	(12.9, 15.48]	357	0.929972	0.006347	332.0	25.0	0.006471	0.005059	0.246241	0.002456
6	(15.48, 18.06]	379	0.923483	0.006738	350.0	29.0	0.006822	0.005868	0.150619	0.000351
7	(18.06, 20.64]	271	0.944649	0.004818	256.0	15.0	0.004990	0.003035	0.497109	0.001832
8	(20.64, 23.22]	507	0.927022	0.009014	470.0	37.0	0.009161	0.007487	0.201797	0.004171
9	(23.22, 25.8]	360	0.941667	0.006400	339.0	21.0	0.006608	0.004249	0.441460	0.002553
10	(25.8, 28.38]	642	0.925234	0.011414	594.0	48.0	0.011578	0.009713	0.175660	0.004970
11	(28.38, 30.96]	492	0.924797	0.008747	455.0	37.0	0.008869	0.007487	0.169361	0.002709
12	(30.96, 33.54]	801	0.941323	0.014241	754.0	47.0	0.014696	0.009510	0.435227	0.005828
13	(33.54, 36.12]	876	0.930365	0.015574	815.0	61.0	0.015885	0.012343	0.252296	0.001189
14	(36.12, 38.7]	752	0.946809	0.013370	712.0	40.0	0.013878	0.008094	0.539180	0.002008
15	(38.7, 41.28]	1107	0.939476	0.019681	1040.0	67.0	0.020271	0.013557	0.402265	0.006393
16	(41.28, 43.86]	910	0.939560	0.016179	855.0	55.0	0.016665	0.011129	0.403750	0.003606
17	(43.86, 46.44]	1390	0.938849	0.024712	1305.0	85.0	0.025436	0.017200	0.391289	0.008771

18	(46.44, 49.02]	1553	0.936896	0.027610	1455.0	98.0	0.028360	0.019830	0.357776	0.002924
19	(49.02, 51.6]	1092	0.937729	0.019414	1024.0	68.0	0.019959	0.013760	0.371946	0.008401
20	(51.6, 54.18]	1844	0.932755	0.032784	1720.0	124.0	0.033525	0.025091	0.289780	0.013566
21	(54.18, 56.76]	1191	0.941226	0.021174	1121.0	70.0	0.021850	0.014164	0.433463	0.011675
22	(56.76, 59.34]	1826	0.937568	0.032464	1712.0	114.0	0.033369	0.023068	0.369201	0.011519
23	(59.34, 61.92]	1293	0.916473	0.022988	1185.0	108.0	0.023097	0.021854	0.055349	0.010272
24	(61.92, 64.5]	1868	0.934690	0.033211	1746.0	122.0	0.034032	0.024686	0.321044	0.010935
25	(64.5, 67.08]	1837	0.939031	0.032660	1725.0	112.0	0.033622	0.022663	0.394465	0.000409
26	(67.08, 69.66]	1151	0.927020	0.020463	1067.0	84.0	0.020797	0.016997	0.201771	0.012825
27	(69.66, 72.24]	1723	0.928613	0.030633	1600.0	123.0	0.031186	0.024889	0.225556	0.010389
28	(72.24, 74.82]	1026	0.921053	0.018241	945.0	81.0	0.018419	0.016390	0.116718	0.012767
29	(74.82, 77.4]	1627	0.921328	0.028926	1499.0	128.0	0.029217	0.025900	0.120505	0.010798
30	(77.4, 79.98]	1043	0.917546	0.018543	957.0	86.0	0.018653	0.017402	0.069438	0.010564
31	(79.98, 82.56]	1528	0.913613	0.027166	1396.0	132.0	0.027210	0.026710	0.018546	0.008557
32	(82.56, 85.14]	1288	0.913043	0.022899	1176.0	112.0	0.022922	0.022663	0.011357	0.004288
33	(85.14, 87.72]	1051	0.863939	0.018685	908.0	143.0	0.017698	0.028936	-0.491618	0.005224
34	(87.72, 90.3]	1425	0.883509	0.025335	1259.0	166.0	0.024540	0.033590	-0.313933	0.006841
35	(90.3, 92.88]	1069	0.871843	0.019005	932.0	137.0	0.018166	0.027722	-0.422666	0.006374
36	(92.88, 95.46]	1759	0.876066	0.031273	1541.0	218.0	0.030036	0.044112	-0.384326	0.011870
37	(95.46, 98.04]	1967	0.883071	0.034971	1737.0	230.0	0.033856	0.046540	-0.318183	0.003820
38	(98.04, 100.62]	1404	0.896724	0.024961	1259.0	145.0	0.024540	0.029340	-0.178679	0.009317
39	(100.62, 103.2]	2262	0.898320	0.040215	2032.0	230.0	0.039606	0.046540	-0.161322	0.015067
40	(103.2, 105.78]	1698	0.902827	0.030188	1533.0	165.0	0.029880	0.033387	-0.110982	0.009726
41	(105.78, 108.36]	2582	0.904725	0.045905	2336.0	246.0	0.045532	0.049777	-0.089154	0.015651
42	(108.36, 110.94]	1831	0.915893	0.032553	1677.0	154.0	0.032687	0.031161	0.047791	0.012845
43	(110.94, 113.52]	2385	0.901048	0.042402	2149.0	236.0	0.041887	0.047754	-0.131092	0.009200
44	(113.52, 116.1]	2303	0.897525	0.040944	2067.0	236.0	0.040288	0.047754	-0.169996	0.001598
45	(116.1, 118.68]	1396	0.911891	0.024819	1273.0	123.0	0.024812	0.024889	-0.003071	0.015476
46	(118.68, 121.26]	408	0.884804	0.007254	361.0	47.0	0.007036	0.009510	-0.301288	0.017776
47	(121.26, 123.84]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
48	(123.84, 126.42]	0	NaN	0.000000	NaN	NaN	NaN	NaN	NaN	NaN
49	(126.42, 129.0]	1	0.000000	0.000018	0.0	1.0	0.000000	0.000202	NaN	NaN

Nous remarquons qu'il n'y a pas beaucoup d'observations dans les intervalles 1-7 et 46-49.

In [229…

```
graph_PdE_V(df)
```



Définissons les groupes : Groupe "sans valeurs" 0-3 a le plus bas PdE Les groupes de 1 à 7 qui n'avaient pas beaucoup d'observations : 3 - 21 21-31 31-85 85+

In [230…

```
x_train['mths_since_last_record_null'] = np.where((x_train['mths_since_last_record'].isnull()), 1, 0)
x_train['mths_since_last_record_0-3'] = np.where((x_train['mths_since_last_record'] >= 0) & (x_train['mths_sinc
x_train['mths_since_last_record_3-21'] = np.where((x_train['mths_since_last_record'] > 3) & (x_train['mths_sinc
x_train['mths_since_last_record_21-31'] = np.where((x_train['mths_since_last_record'] > 21) & (x_train['mths_si
x_train['mths_since_last_record_31-85'] = np.where((x_train['mths_since_last_record'] > 31) & (x_train['mths_si
x_train['mths_since_last_record>85'] = np.where((x_train['mths_since_last_record'] > 85), 1, 0)
```

Prétraitement des données d'évaluation

In [232…

```
#Final_x_train = x_train
#Final_y_train = y_train

#une fois executé Ne pas Oublier de mettre comme comentaire
```

In [233…

```
#x_train=x_test
#y_train=y_test


#une fois executé Ne pas Oublier de mettre comme comentaire
```

In [235…

```
# Export *.csv pour les modèles
```

In [236…

```
#Final_x_train.to_csv('Final_x_train')
#Final_y_train.to_csv('Final_y_train')
#Final_x_test.to_csv('Final_x_test')
#Final_y_train.to_csv('Final_y_test')
```

Loading [MathJax]/extensions/Safe.js