1. Introduction

a. Import

In [1]:
```
import numpy as np
import pandas as pd
import sklearn
from sklearn.preprocessing import LabelEncoder
import os
import warnings
warnings.filterwarnings('ignore')
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import roc_curve
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
import plotly.graph_objs as go
import plotly.offline as py
```

b. Import des CSV

In [3]:
```
app_train = pd.read_csv('csv/application_train.csv')
print('Taille: ', app_train.shape)
app_train.head()
```

```
Taille:  (307511, 122)
```

Out[3]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OV
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	

5 rows × 122 columns

Le fichier contient 307511 lignes et 122 colonnes. Nous allons chercher à prédire la colonne "Target"

In [4]:
```
app_test = pd.read_csv('csv/application_test.csv')
print('Taille: ', app_test.shape)
app_test.head()
```

```
Taille:  (48744, 121)
```

Out[4]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALT
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	
3	100028	Cash loans	F	N	
4	100038	Cash loans	M	Y	

5 rows × 121 columns

Le fichier contient 48744 lignes et 121 colonnes. Il manque la colonne "Target"

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

1. Préprocessing

a. Conversion des données

In [5]:
```
app_train.dtypes.value_counts()
```

Out[5]:
```
float64    65
int64      41
object     16
dtype: int64
```

In [6]:
```
app_train.select_dtypes('object').apply(pd.Series.nunique, axis = 0)
```

Out[6]:
```
NAME_CONTRACT_TYPE              2
CODE_GENDER                     3
FLAG_OWN_CAR                    2
FLAG_OWN_REALTY                 2
NAME_TYPE_SUITE                 7
NAME_INCOME_TYPE                8
NAME_EDUCATION_TYPE             5
NAME_FAMILY_STATUS              6
NAME_HOUSING_TYPE               6
OCCUPATION_TYPE                18
WEEKDAY_APPR_PROCESS_START      7
ORGANIZATION_TYPE              58
FONDKAPREMONT_MODE              4
HOUSETYPE_MODE                  3
WALLSMATERIAL_MODE              7
EMERGENCYSTATE_MODE             2
dtype: int64
```

In [7]:
```
le = LabelEncoder()
le_count = 0

for col in app_train:
    if app_train[col].dtype == 'object':
        if len(list(app_train[col].unique())) <= 2:
            le.fit(app_train[col])
            app_train[col] = le.transform(app_train[col])
            app_test[col] = le.transform(app_test[col])
```

```
            le_count += 1

print('%d colonnes ont été encodées' % le_count)
```

```
3 colonnes ont été encodées
```

In [8]:
```
# one-hot encoding of categorical variables
app_train = pd.get_dummies(app_train)
app_test = pd.get_dummies(app_test)

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

```
Training Features shape:  (307511, 243)
Testing Features shape:  (48744, 239)
```

Nous allons comparer les deux jeux de données pour leur donner la même dimension

In [9]:
```
train_labels = app_train['TARGET']

app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)

app_train['TARGET'] = train_labels

print('Dimension app_train avant: ', app_train.shape)
print('Dimension app_test avant: ', app_test.shape)
```

```
Dimension app_train avant:  (307511, 240)
Dimension app_test avant:  (48744, 239)
```

b. Analyse des données manquantes

In [10]:
```
missing_values_count = app_train.isnull().sum()
missing_values_count
```

Out[10]:
```
SK_ID_CURR                           0
NAME_CONTRACT_TYPE                   0
FLAG_OWN_CAR                         0
FLAG_OWN_REALTY                      0
CNT_CHILDREN                         0
                                    ..
WALLSMATERIAL_MODE_Stone, brick      0
WALLSMATERIAL_MODE_Wooden            0
EMERGENCYSTATE_MODE_No               0
EMERGENCYSTATE_MODE_Yes              0
TARGET                               0
Length: 240, dtype: int64
```

In [11]:
```
total_cells = np.product(app_train.shape)
total_missing = missing_values_count.sum()
percent = (total_missing/total_cells) * 100
print('Pourcentage de données manquantes :',percent)
```

```
Pourcentage de données manquantes : 11.365574456415109
```

Et si on cherche plus en détail:

In [12]:
```
#Calculer les valeurs manquantes avec des pourcentages
count = app_train.isnull().sum().sort_values(ascending=False)
percentage = ((app_train.isnull().sum()/len(app_train)*100)).sort_values(asce
missing_app_train = pd.concat([count, percentage], axis=1, keys=['Count','Per
print('Total et pourcentage des données manquantes pour les 30 premières colo
missing_app_train.head(30)
```

Total et pourcentage des données manquantes pour les 30 premières colonnes du dataframe app_train

Out[12]:

	Count	Percentage
COMMONAREA_MEDI	214865	69.872297
COMMONAREA_AVG	214865	69.872297
COMMONAREA_MODE	214865	69.872297
NONLIVINGAPARTMENTS_AVG	213514	69.432963
NONLIVINGAPARTMENTS_MODE	213514	69.432963
NONLIVINGAPARTMENTS_MEDI	213514	69.432963
LIVINGAPARTMENTS_MEDI	210199	68.354953
LIVINGAPARTMENTS_AVG	210199	68.354953
LIVINGAPARTMENTS_MODE	210199	68.354953
FLOORSMIN_MEDI	208642	67.848630
FLOORSMIN_MODE	208642	67.848630
FLOORSMIN_AVG	208642	67.848630
YEARS_BUILD_MODE	204488	66.497784
YEARS_BUILD_MEDI	204488	66.497784
YEARS_BUILD_AVG	204488	66.497784
OWN_CAR_AGE	202929	65.990810
LANDAREA_AVG	182590	59.376738
LANDAREA_MODE	182590	59.376738
LANDAREA_MEDI	182590	59.376738
BASEMENTAREA_AVG	179943	58.515956
BASEMENTAREA_MEDI	179943	58.515956
BASEMENTAREA_MODE	179943	58.515956
EXT_SOURCE_1	173378	56.381073
NONLIVINGAREA_AVG	169682	55.179164
NONLIVINGAREA_MEDI	169682	55.179164
NONLIVINGAREA_MODE	169682	55.179164
ELEVATORS_MEDI	163891	53.295980
ELEVATORS_MODE	163891	53.295980
ELEVATORS_AVG	163891	53.295980
APARTMENTS_MODE	156061	50.749729

In [13]:

```
#Calculer les valeurs manquantes avec des pourcentages
count = app_test.isnull().sum().sort_values(ascending=False)
percentage = ((app_test.isnull().sum()/len(app_test)*100)).sort_values(ascend
missing_app_test = pd.concat([count, percentage], axis=1, keys=['Count','Perc
print('Total et pourcentage des données manquantes pour les 30 premières colo
missing_app_test.head(30)
```

Total et pourcentage des données manquantes pour les 30 premières colonnes du

dataframe app_train

Out[13]:

	Count	Percentage
COMMONAREA_MODE	33495	68.716150
COMMONAREA_MEDI	33495	68.716150
COMMONAREA_AVG	33495	68.716150
NONLIVINGAPARTMENTS_MEDI	33347	68.412523
NONLIVINGAPARTMENTS_AVG	33347	68.412523
NONLIVINGAPARTMENTS_MODE	33347	68.412523
LIVINGAPARTMENTS_MODE	32780	67.249302
LIVINGAPARTMENTS_AVG	32780	67.249302
LIVINGAPARTMENTS_MEDI	32780	67.249302
FLOORSMIN_AVG	32466	66.605121
FLOORSMIN_MEDI	32466	66.605121
FLOORSMIN_MODE	32466	66.605121
OWN_CAR_AGE	32312	66.289184
YEARS_BUILD_MODE	31818	65.275726
YEARS_BUILD_AVG	31818	65.275726
YEARS_BUILD_MEDI	31818	65.275726
LANDAREA_MODE	28254	57.964057
LANDAREA_MEDI	28254	57.964057
LANDAREA_AVG	28254	57.964057
BASEMENTAREA_MEDI	27641	56.706466
BASEMENTAREA_AVG	27641	56.706466
BASEMENTAREA_MODE	27641	56.706466
NONLIVINGAREA_MEDI	26084	53.512227
NONLIVINGAREA_MODE	26084	53.512227
NONLIVINGAREA_AVG	26084	53.512227
ELEVATORS_AVG	25189	51.676104
ELEVATORS_MEDI	25189	51.676104
ELEVATORS_MODE	25189	51.676104
APARTMENTS_AVG	23887	49.005006
APARTMENTS_MEDI	23887	49.005006

Sur les deux dataset, il y a beaucoup de données manquantes.

In [14]:

```
columns_with_na_dropped = app_train.dropna(axis=1)
print("Nombre de colonne à l'origine: %d \n" % app_train.shape[1])
print("Nombre de colonne qu'il nous resterait: %d" % columns_with_na_dropped.
```

```
Nombre de colonne à l'origine: 240

Nombre de colonne qu'il nous resterait: 179
```

In [ ]:

In [ ]:

In [15]:
```
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')


if 'TARGET' in app_train:
    train = app_train.drop(columns = ['TARGET'])
else:
    train = app_train.copy()


features = list(train.columns)
test = app_test.copy()
imputer = SimpleImputer(strategy = 'median')
scaler = MinMaxScaler(feature_range = (0, 1))
imputer.fit(train)
train = imputer.transform(train)
test = imputer.transform(app_test)
scaler.fit(train)
train = scaler.transform(train)
test = scaler.transform(test)

print('Dimension app_train après: ', train.shape)
print('Dimension app_test après; ', test.shape)
```

```
Dimension app_train après:  (307511, 239)
Dimension app_test après;  (48744, 239)
```

c. Correction des anomalies

In [16]:
```
app_train['DAYS_EMPLOYED_ANOM'] = app_train["DAYS_EMPLOYED"] == 365243
app_train['DAYS_EMPLOYED'].replace({365243: np.nan}, inplace = True)
app_test['DAYS_EMPLOYED_ANOM'] = app_test["DAYS_EMPLOYED"] == 365243
app_test["DAYS_EMPLOYED"].replace({365243: np.nan}, inplace = True)
```

In [17]:
```
print('There are %d anomalies in the test data out of %d entries' % (app_test
```

```
There are 9274 anomalies in the test data out of 48744 entries
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

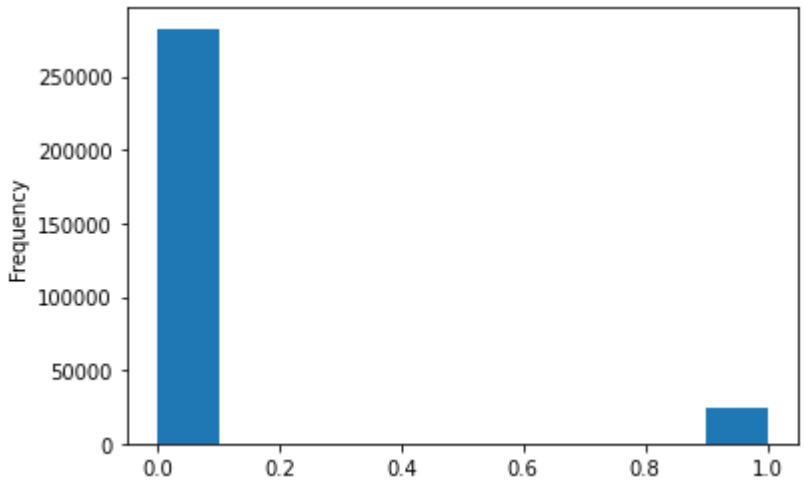
1. Analyse

Nous cherchons à prédire si un crédit peut être accordé ou non. Pour cela nous allons utiliser la variable "Target" qui permet de manière binaire, de préciser si le prêt a été payé à temps ou non. 0: le crédit a été payé à temps / 1: le payeur a eu des difficultés de paiement

In [18]:
```
app_train['TARGET'].value_counts()
```

Out[18]:
```
0    282686
1     24825
Name: TARGET, dtype: int64
```

In [19]:
```
app_train['TARGET'].astype(int).plot.hist();
```



Commencons par établir les corrélations. On affiche ici les 4 premières corrélations positives et les 4 corrélations négatives

In [20]:
```
correlations = app_train.corr()['TARGET'].sort_values()
print('Most Positive Correlations:\n', correlations.tail(4))
print('\nMost Negative Correlations:\n', correlations.head(4))
```

```
Most Positive Correlations:
 REGION_RATING_CLIENT_W_CITY    0.060893
DAYS_EMPLOYED                  0.074958
DAYS_BIRTH                     0.078239
TARGET                         1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                            -0.178919
EXT_SOURCE_2                            -0.160472
EXT_SOURCE_1                            -0.155317
NAME_EDUCATION_TYPE_Higher education    -0.056593
Name: TARGET, dtype: float64
```

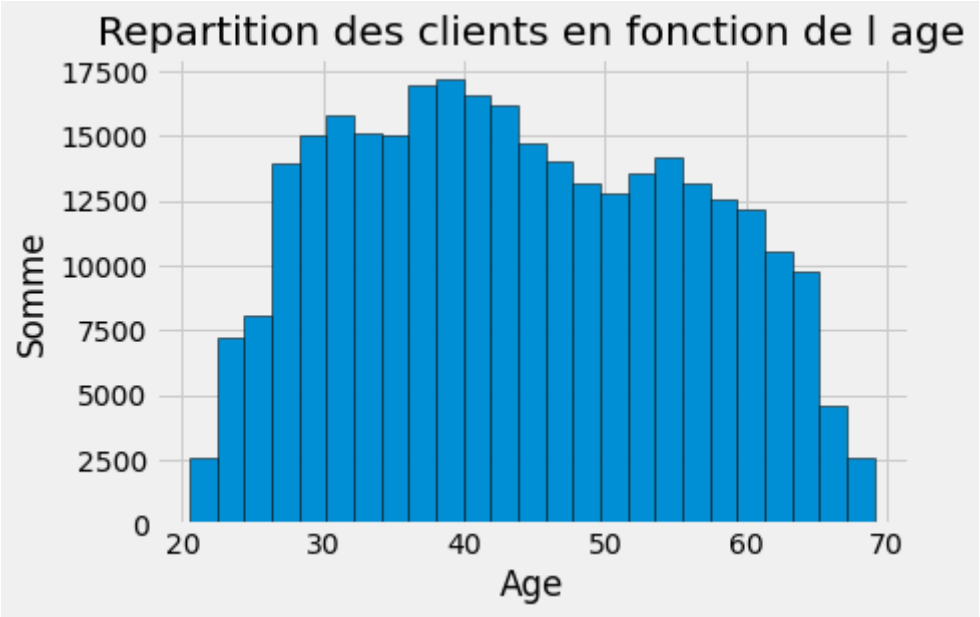
La colonne d'âge est la colonne la plus corrélée avec le paiement du crédit. Obeservons donc les montants accordés en fonction de l'âge

In [21]:
```
app_train['DAYS_BIRTH'] = abs(app_train['DAYS_BIRTH'])
app_train['DAYS_BIRTH'].corr(app_train['TARGET'])
```

Out[21]:
```
-0.07823930830982694
```

In [22]:
```
plt.style.use('fivethirtyeight')
plt.hist(app_train['DAYS_BIRTH'] / 365, edgecolor = 'k', bins = 25)
```

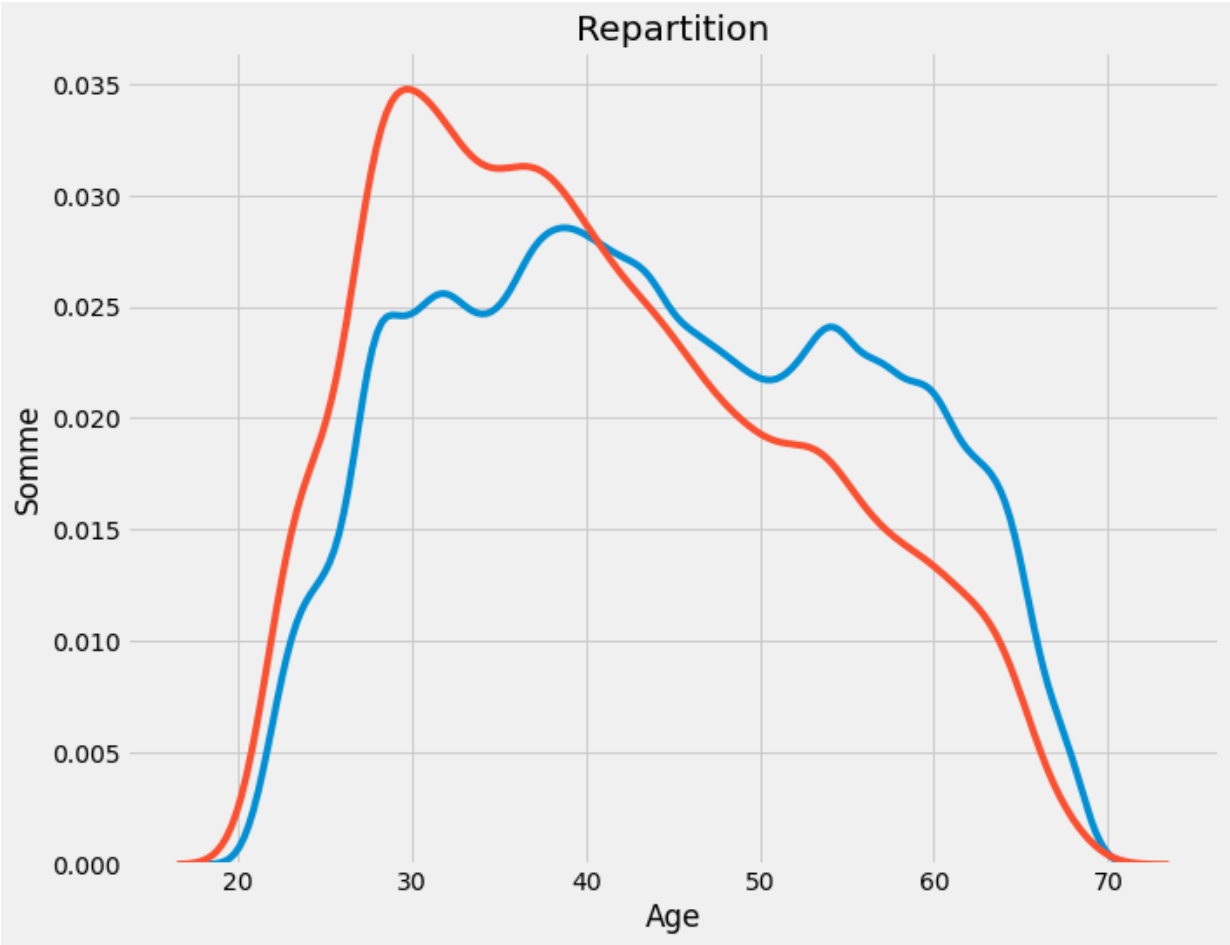
```
plt.title('Repartition des clients en fonction de l age');plt.xlabel('Age');
```



Calculons le KDE

In [23]:

```
plt.figure(figsize = (10, 8))
sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, 'DAYS_BIRTH'] / 365, labe
sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, 'DAYS_BIRTH'] / 365, labe
plt.xlabel('Age'); plt.ylabel('Somme'); plt.title('Repartition');
```



On va partager l'age en segment de 5 ans

In [24]:

```
age_data = app_train[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365
```

```
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace
age_data.head(10)
```

Out[24]:

	TARGET	DAYS_BIRTH	YEARS_BIRTH	YEARS_BINNED
0	1	9461	25.920548	(25.0, 30.0]
1	0	16765	45.931507	(45.0, 50.0]
2	0	19046	52.180822	(50.0, 55.0]
3	0	19005	52.068493	(50.0, 55.0]
4	0	19932	54.608219	(50.0, 55.0]
5	0	16941	46.413699	(45.0, 50.0]
6	0	13778	37.747945	(35.0, 40.0]
7	0	18850	51.643836	(50.0, 55.0]
8	0	20099	55.065753	(55.0, 60.0]
9	0	14469	39.641096	(35.0, 40.0]

In [25]:

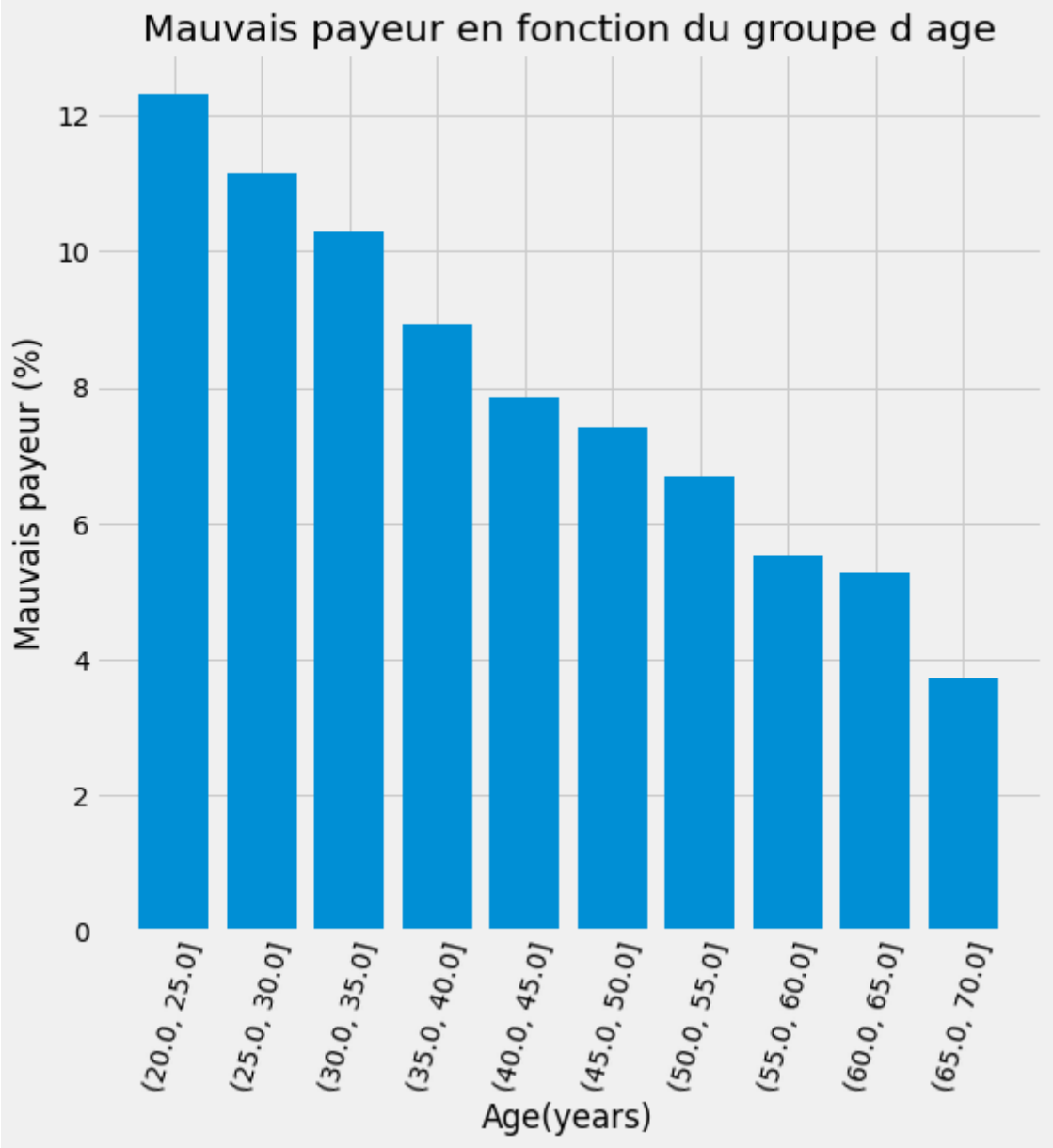
```
age_groups  = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

Out[25]:

YEARS_BINNED	TARGET	DAYS_BIRTH	YEARS_BIRTH
(20.0, 25.0]	0.123036	8532.795625	23.377522
(25.0, 30.0]	0.111436	10155.219250	27.822518
(30.0, 35.0]	0.102814	11854.848377	32.479037
(35.0, 40.0]	0.089414	13707.908253	37.555913
(40.0, 45.0]	0.078491	15497.661233	42.459346
(45.0, 50.0]	0.074171	17323.900441	47.462741
(50.0, 55.0]	0.066968	19196.494791	52.593136
(55.0, 60.0]	0.055314	20984.262742	57.491131
(60.0, 65.0]	0.052737	22780.547460	62.412459
(65.0, 70.0]	0.037270	24292.614340	66.555108

In [26]:

```
plt.figure(figsize = (8, 8))


plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])
plt.xticks(rotation = 75); plt.xlabel('Age(years)'); plt.ylabel('Mauvais paye
plt.title('Mauvais payeur en fonction du groupe d age');
```

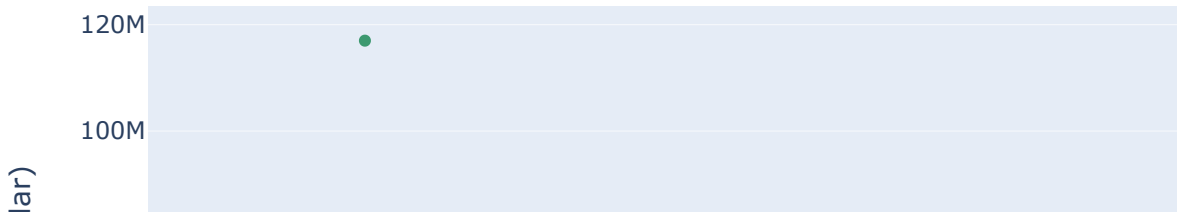


On découpe ces catégories

In [27]:
```
interval = (18, 25, 35, 60, 120)

cats = ['Student', 'Young', 'Adult', 'Senior']
app_train["Age_cat"] = pd.cut(age_data['YEARS_BIRTH'], interval, labels=cats)


df_good = app_train[app_train["TARGET"] == 1]
df_bad = app_train[app_train["TARGET"] == 0]
```

In [28]:
```
trace0 = go.Box(
    y=df_good["AMT_INCOME_TOTAL"],
    x=df_good["Age_cat"],
    name='Good credit',
    marker=dict(
        color='#3D9970'
    )
)

trace1 = go.Box(
    y=df_bad['AMT_INCOME_TOTAL'],
    x=df_bad['Age_cat'],
    name='Bad credit',
```

```
    marker=dict(
        color='#FF4136'
    )
)

data = [trace0, trace1]

layout = go.Layout(
    yaxis=dict(
        title='Credit Amount (US Dollar)',
        zeroline=False
    ),
    xaxis=dict(
        title='Age Categorical'
    ),
    boxmode='group'
)
fig = go.Figure(data=data, layout=layout)

py.iplot(fig)
```



In [ ]:

Autre corrélation: les sources extérieures Pour rappel des corrélations:

In [29]:

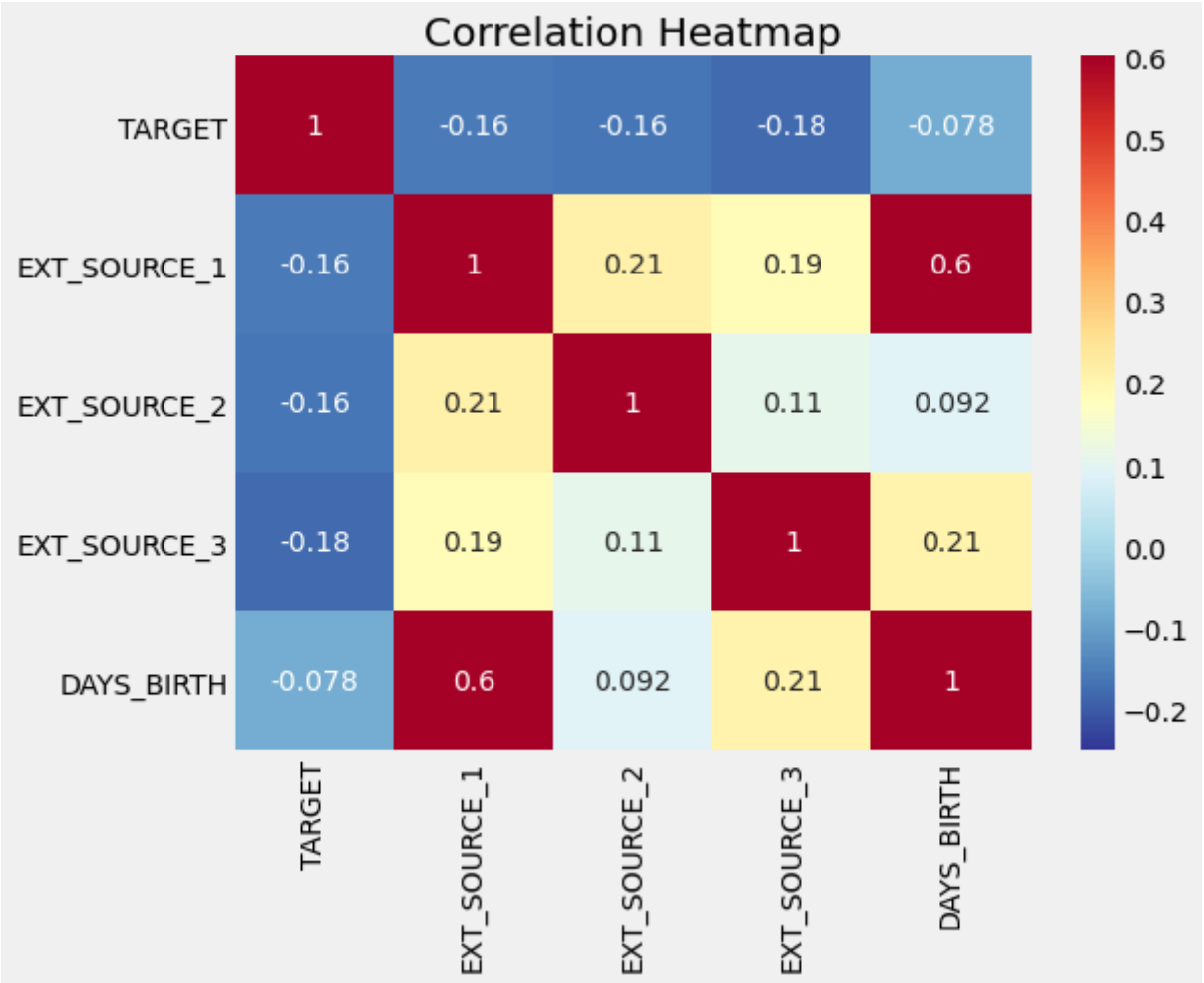
```
ext_data = app_train[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3
ext_data_corrs = ext_data.corr()
ext_data_corrs
```

Out[29]:

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155317	-0.160472	-0.178919	-0.078239
EXT_SOURCE_1	-0.155317	1.000000	0.213982	0.186846	0.600610
EXT_SOURCE_2	-0.160472	0.213982	1.000000	0.109167	0.091996
EXT_SOURCE_3	-0.178919	0.186846	0.109167	1.000000	0.205478
DAYS_BIRTH	-0.078239	0.600610	0.091996	0.205478	1.000000

In [30]:

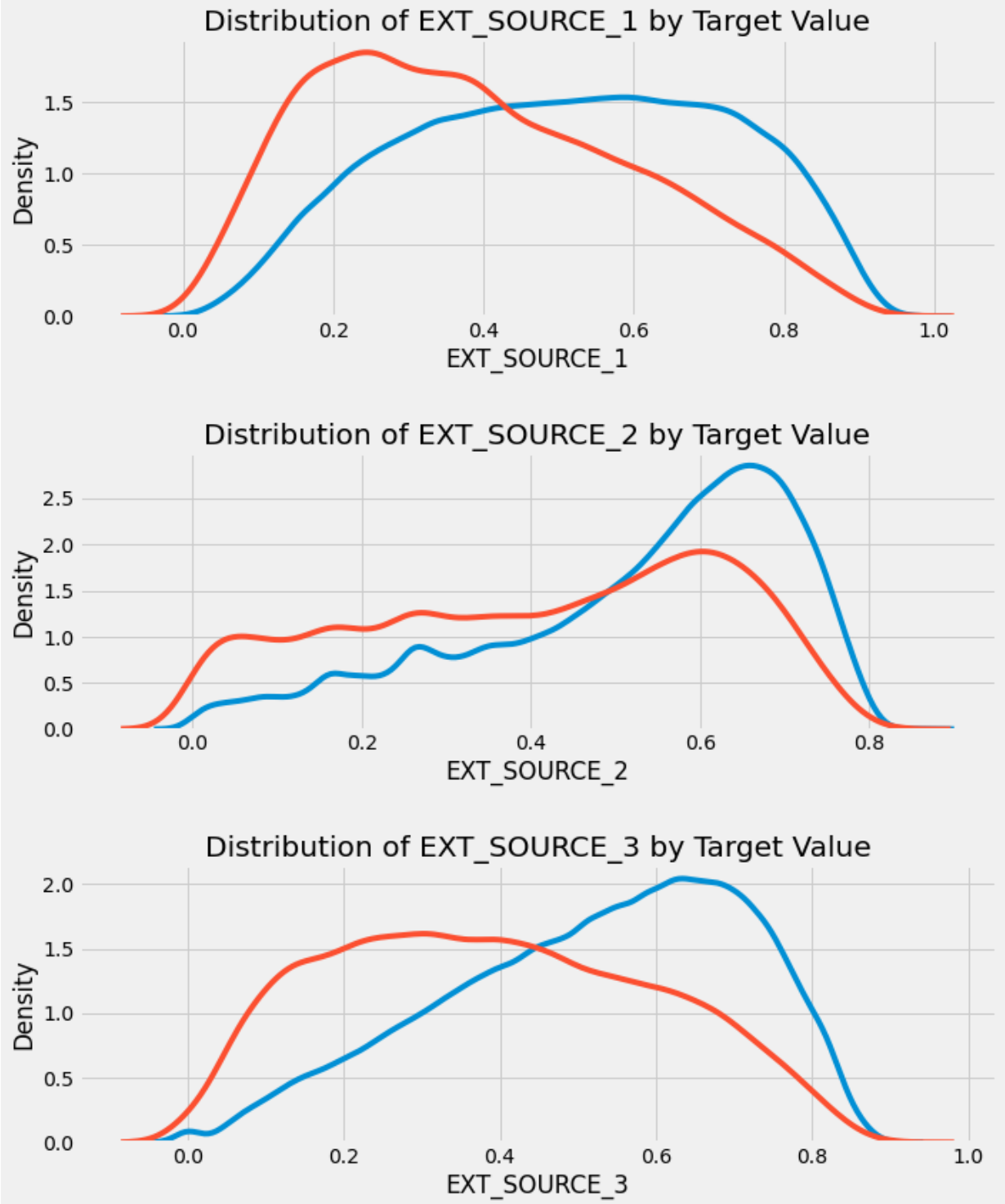
```
plt.figure(figsize = (8, 6))

sns.heatmap(ext_data_corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = Tru
plt.title('Correlation Heatmap');
```



In [31]:

```
plt.figure(figsize = (10, 12))


for i, source in enumerate(['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']):


    plt.subplot(3, 1, i + 1)
    sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, source], label = 'tar
    sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, source], label = 'tar

    plt.title('Distribution of %s by Target Value' % source)
    plt.xlabel('%s' % source); plt.ylabel('Density');
```

```
plt.tight_layout(h_pad = 2.5)
```

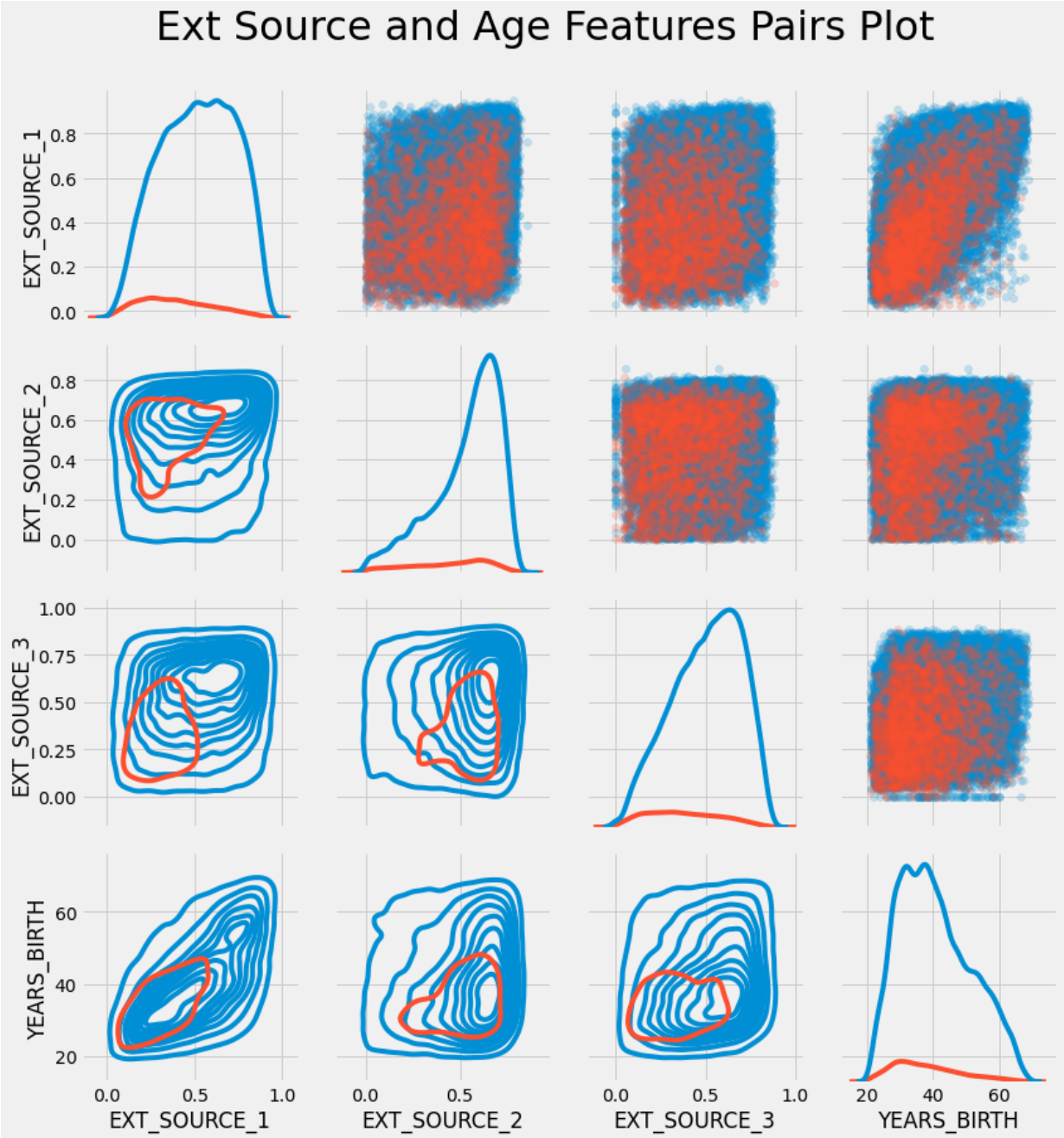


Pair plot

In [32]:

```
plot_data = ext_data.drop(columns = ['DAYS_BIRTH']).copy()

plot_data['YEARS_BIRTH'] = age_data['YEARS_BIRTH']


plot_data = plot_data.dropna().loc[:100000, :]

def corr_func(x, y, **kwargs):
    r = np.corrcoef(x, y)[0][1]
    ax = plt.gca()
    ax.annotate("r = {:.2f}".format(r),
                xy=(.2, .8), xycoords=ax.transAxes,
```

```
                    size = 20)

grid = sns.PairGrid(data = plot_data, size = 3, diag_sharey=False,
                        hue = 'TARGET',
                        vars = [x for x in list(plot_data.columns) if x != 'TARGE

grid.map_upper(plt.scatter, alpha = 0.2)

grid.map_diag(sns.kdeplot)

grid.map_lower(sns.kdeplot, cmap = plt.cm.OrRd_r);

plt.suptitle('Ext Source and Age Features Pairs Plot', size = 32, y = 1.05);
```



Regression logistique

In [33]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values=np.nan, strategy='mean')

if 'TARGET' in app_train:
    train = app_train.drop(columns = ['TARGET', 'Age_cat'])
```

```
else:
    train = app_train.copy()


features = list(train.columns)
test = app_test.copy()
imputer = SimpleImputer(strategy = 'median')
scaler = MinMaxScaler(feature_range = (0, 1))
imputer.fit(train)
train = imputer.transform(train)
test = imputer.transform(app_test)
scaler.fit(train)
train = scaler.transform(train)
test = scaler.transform(test)

print('Dimension app_train: ', train.shape)
print('Dimension app_test: ', test.shape)
```

```
Dimension app_train:  (307511, 240)
Dimension app_test:  (48744, 240)
```

In [34]:

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(C = 0.5, class_weight='balanced')
log_reg.fit(train, train_labels)
```

Out[34]:

```
LogisticRegression(C=0.5, class_weight='balanced')
```

In [35]:

```
pd.DataFrame(train)
```

Out[35]:

	0	1	2	3	4	5	6	7	8	9	...
0	0.000000	0.0	0.0	1.0	0.0	0.001512	0.090287	0.090032	0.077441	0.256321	...
1	0.000003	0.0	0.0	0.0	0.0	0.002089	0.311736	0.132924	0.271605	0.045016	...
2	0.000006	1.0	1.0	1.0	0.0	0.000358	0.022472	0.020025	0.023569	0.134897	...
3	0.000011	0.0	0.0	1.0	0.0	0.000935	0.066837	0.109477	0.063973	0.107023	...
4	0.000014	0.0	0.0	1.0	0.0	0.000819	0.116854	0.078975	0.117845	0.392880	...
...	...	...	...	...	...	...	...	...	...	...	...
307506	0.999989	0.0	0.0	0.0	0.0	0.001127	0.052360	0.101176	0.046016	0.446855	...
307507	0.999992	0.0	0.0	1.0	0.0	0.000396	0.056067	0.040505	0.046016	0.344429	...
307508	0.999994	0.0	0.0	1.0	0.0	0.001089	0.157969	0.110618	0.135802	0.065247	...
307509	0.999997	0.0	0.0	1.0	0.0	0.001243	0.081175	0.072499	0.069585	0.069553	...
307510	1.000000	0.0	0.0	0.0	0.0	0.001127	0.157303	0.185258	0.158249	0.635991	...

307511 rows × 240 columns

In [36]:

```
log_reg_pred = log_reg.predict_proba(test)[:, 1]
```

In [37]:

```
#sklearn.metrics.auc(test, log_reg_pred)
```

Score du model

In [38]:

```
from sklearn.linear_model import LogisticRegression
log_reg.score(train, train_labels)
```

Out[38]: 0.6876339382981422

In [39]:
```
submit = app_test[['SK_ID_CURR']]
submit['TARGET'] = log_reg_pred

submit.head()
```

Out[39]:

	SK_ID_CURR	TARGET
0	100001	0.369859
1	100005	0.750325
2	100013	0.308370
3	100028	0.310555
4	100038	0.618065

On sauvegarde

In [40]:
```
submit.to_csv('log_reg.csv', index = False)
```

In [ ]: