```python
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
from sklearn.preprocessing import StandardScaler

class Data_Cleaning:
    def __init__(self,csv_file, sep = ','):
        self.csv_file = csv_file
        self.df = None
        self.sep = sep
        self.cols = None
        self.x_num = None
        self.x_str = None
        self.x = None
        self.y = None
        self.target = None
        self.reply = None

    # Est ce que le df contient de header ou non
    def isHeader(self):
        if self.reply == 'yes':
            self.df = pd.read_csv(self.csv_file, sep = self.sep)
        else:
            self.df = pd.read_csv(self.csv_file, header = None)
            self.cols = [c.strip() for c in self.cols.split(',')]
            self.df.columns = self.cols

    # On fait la separation les colonnes numeriques et les colonnes catigoriales
    def separation_xnum_xstr(self):
        self.x_num = self.df.select_dtypes(include = [np.number])
        self.x_str = self.df.select_dtypes(exclude = [np.number])

        for i in self.x_str.columns:
            self.x_str[i] = self.x_str[i].fillna(self.x_str[i].mode()[0])

    # On fait l'encodeage des colonnes categoriale par LabelEncoder
    def encodage(self):
        encoder = LabelEncoder()

        for i in self.x_str.columns:
            self.x_str[i] = encoder.fit_transform(self.x_str[i])
        # On fait la concatination des deux colonnes encodées (initialement categoriale) avec les colonnes numerique
        self.x = pd.concat([self.x_num,self.x_str],axis = 1)

    # Choisir la methode de remplissage des valeurs manquantes
    def val_manq(self):
        if self.reply == 'intelligente':
            impute = IterativeImputer(random_state = 42)
            self.x = pd.DataFrame(impute.fit_transform(self.x), columns = self.x.columns)
        elif self.reply == 'moyenne':
            self.x = self.x.fillna(self.x.mean())
        else:
            self.x = self.x.fillna(self.x.median())

    # La suppression des observation doublons
    def duplication(self):
        self.x = self.x.drop_duplicates()

    # Pour traiter les valeurs aburantes, j'ai utilisé le "capping"
    def outlier(self,col):
        Q1 = col.quantile(0.25)
        Q3 = col.quantile(0.75)

        IQR = Q3 - Q1

        Fb = Q1 - 1.5 * IQR
        Fh = Q3 + 1.5 * IQR

        return col.clip(lower = Fb, upper = Fh)

    def remp_outlier(self):
        for col in self.x.columns :
            self.x[col] = self.outlier(self.x[col])

```

```
77      # Séparation des caracteristiques (x) et la variable cible (y)
78      def separation_x_y(self):
79          self.y = self.x[self.target]
80          self.x = self.x.drop(columns = self.target)
81
82      # Est ce que l'utilisateur souhaite de faire la standarisation ou non
83      def standarisation(self):
84          scale = StandardScaler()
85
86          if self.reply == 'yes':
87              self.x = pd.DataFrame(scale.fit_transform(self.x),columns = self.x.columns)
88
89      # Retourner la nouvelle dataframe nettoyée
90      def df_final(self):
91          self.df = pd.concat([self.x,self.y], axis = 1)
92          return self.df
```