Traitement des valeurs manquantes

Dans une analyse predictive de données, l'une des tâches que nous deverios effectuer avant la formatin de notre modèle d'apprentissage automatique est le preprocessing des données. Le nettoyage des données est un élément clé de la tâche de preprocessing des données et implique généralement la suppression des valeurs manquantes ou leur remplacement par la moyenne, la mediane, la mode ou une constante.

Pourquoi faut-il remplir les données manquantes ?

1. La plupart des modèles d'apprentissage automatique généreront une erreur si on leur transmet des valeurs NaN.
2. Le moyen le plus simple consiste simplement à les remplir avec 0, mais cela peut réduire considérablement la précision du modèle.
3. Pour remplir les valeurs manquantes, il existe de nombreuses méthodes disponibles.
4. Pour choisir la meilleure méthode, on doit comprendre tout d'abord le type de valeurs manquantes et leur signification.

On trouve généralement les valeurs manquantes sous forme de **NaN** ou **null** ou **None** dans le jeu de données.

Nous allons télecharchons notre jeu de données que nous avons récuperé sur Kaggle via lien ci-dessous :

https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset?select=test_Y3wMUE5_7gLdaTN.csv

In [1]:
```
import pandas as pd
import numpy as np

train_data = pd.read_csv("train_u6lujuX_CVtuZ9i.csv")
#test_data = pd.read_csv("test_Y3wMUE5_7gLdaTN.csv")
#data = pd.concat([train_data, test_data], ignore_index=True)
```

In [2]:
```
# Affichons l'entête de données
train_data.head()
```

Out[2]:

	Loan_ID	Gender	Married	Dependents	Education	Self_Employed	ApplicantIncome	Co
0	LP001002	Male	No	0	Graduate	No	5849	
1	LP001003	Male	Yes	1	Graduate	No	4583	
2	LP001005	Male	Yes	0	Graduate	Yes	3000	
3	LP001006	Male	Yes	0	Not Graduate	No	2583	
4	LP001008	Male	No	0	Graduate	No	6000	

In [3]:
```
#Afficher la forme de données
train_data.shape
```

Out[3]:
```
(614, 13)
```

In [4]:
```
train_data["Loan_Status"].value_counts()
```

Out[4]:
```
Y    422
N    192
Name: Loan_Status, dtype: int64
```

Nous avons 981 lignes et 13 colonnes dans notre jeu de données

In [5]:
```
# Pour avoir des informations sur notre jeu des données
train_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 614 entries, 0 to 613
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   Loan_ID            614 non-null    object 
 1   Gender             601 non-null    object 
 2   Married            611 non-null    object 
 3   Dependents         599 non-null    object 
 4   Education          614 non-null    object 
 5   Self_Employed      582 non-null    object 
 6   ApplicantIncome    614 non-null    int64  
 7   CoapplicantIncome  614 non-null    float64
 8   LoanAmount         592 non-null    float64
 9   Loan_Amount_Term   600 non-null    float64
 10  Credit_History     564 non-null    float64
 11  Property_Area      614 non-null    object 
 12  Loan_Status        614 non-null    object 
dtypes: float64(4), int64(1), object(8)
memory usage: 62.5+ KB
```

On constate qu'il y'a 7 colonnes qui contiennent des valeurs manquantes dont 5 des colonnes sont des caractéristiques catégorielles et 3 des caractéristiques numériques.

In [6]:
```
# Pour avoir le total de valeurs manquantes par colonne
train_data.isnull().sum()
```

Out[6]:
```
Loan_ID               0
Gender               13
Married               3
Dependents           15
Education             0
Self_Employed        32
ApplicantIncome       0
CoapplicantIncome     0
LoanAmount           22
Loan_Amount_Term     14
Credit_History       50
Property_Area         0
Loan_Status           0
dtype: int64
```

On constate que la caractéristique **Credit_History** est celle qui a le plus de valeurs manquantes (50)

Vérifieons qu'il existe également des valeurs catégorielles dans l'ensemble de données. Pour cela, nous devons utiliser **Label Encoding** ou **One Hot Encoding**.

In [7]:
```
train_data.columns
```

Out[7]:
```
Index(['Loan_ID', 'Gender', 'Married', 'Dependents', 'Education',
       'Self_Employed', 'ApplicantIncome', 'CoapplicantIncome', 'LoanAmou
nt',
       'Loan_Amount_Term', 'Credit_History', 'Property_Area', 'Loan_Statu
s'],
      dtype='object')
```

Les méthodes de gestion de valeurs manquantes

1. Suppression des colonnes avec des données manquantes
2. Suppression des lignes avec des données manquantes
3. Remplir les données manquantes avec une valeur : imputation
4. Remplir avec un modèle de régression

1. Suppression des colonnes avec des données manquantes

Dans ce cas, supprimons les colonnes avec des valeurs manqantes, puis ajustons le modèle et vérifions sa précision.

Mais il s'agit d'un cas extrême et ne doit être utilisé que lorsqu'il existe de nombreuses valeurs manqantes dans la colonne.

In [8]:
```
train_data_without_missing_axis1=train_data.dropna(axis=1)
train_data_without_missing_axis1.isnull().sum()
```

Out[8]:
```
Loan_ID              0
Education            0
ApplicantIncome      0
CoapplicantIncome    0
Property_Area        0
Loan_Status          0
dtype: int64
```

https://stackoverflow.com/questions/24458645/label-encoding-across-multiple-columns-in-scikit-learn

One Hot Encoding

Vérifions qu'il existe également des valeurs catégorielles dans l'ensemble de données. Pour cela, vous devons utiliser **Label Encoding** ou **One Hot Encoding**.

In [11]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

train_data_without_missing_axis1=train_data_without_missing_axis1.apply(L
print(train_data_without_missing_axis1.head())
print()
print(train_data_without_missing_axis1)
```

```
   Loan_ID  Education  ApplicantIncome  CoapplicantIncome  Property_Area
\
0        0          0              376                  0              2
1        1          0              306                 60              0
2        2          0              139                  0              2
```

```
3         3          1                90                160               2
4         4          0               381                  0               2

   Loan_Status
0            1
1            0
2            1
3            1
4            1

     Loan_ID  Education  ApplicantIncome  CoapplicantIncome  Property_Are
a  \
0          0          0              376                  0
2
1          1          0              306                 60
0
2          2          0              139                  0
2
3          3          1               90                160
2
4          4          0              381                  0
2
..       ...        ...              ...                ...
...
609      609          0              125                  0
0
610      610          0              275                  0
0
611      611          0              431                  3
2
612      612          0              422                  0
2
613      613          0              306                  0
1

     Loan_Status
0              1
1              0
2              1
3              1
4              1
..           ...
609            1
610            1
611            1
612            1
613            0

[614 rows x 6 columns]
```

In [12]:
```
X_without_missing_axis1 =train_data_without_missing_axis1.drop(["Loan_Sta

y=train_data_without_missing_axis1["Loan_Status"]
```

In [13]:
```
y.value_counts()
```

Out[13]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [14]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(X_without_missing_axis1
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(round(metrics.accuracy_score(pred,y_test), 4))
```

```
0.6486
```

Cette méthode de gestion des valeurs manquantes donne un classificateur moins précis. Nous avons une précision de **64.86%**. Nous allons utiliser d'autres méthodes pour y voir claire

2. Supprimer les lignes avec des données manquantes

S'il y a une certaine ligne avec des données manquantes, nous pouvez supprimer la ligne entière avec toutes les entités de cette ligne.

axis=1 : est utilisé pour supprimer la colonne avec les valeurs `NaN`.

axis=0 : est utilisé pour supprimer la ligne avec les valeurs `NaN`.

In [15]:
```
train_data_without_missing_axis0=train_data.dropna(axis=0)
train_data_without_missing_axis0.isnull().sum()
```

Out[15]:
```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

In [16]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

train_data_without_missing_axis0=train_data_without_missing_axis0.apply(L
print(train_data_without_missing_axis0.head())
print()
print(train_data_without_missing_axis0)
```

```
   Loan_ID  Gender  Married  Dependents  Education  Self_Employed  \
1        0       1        1           1          0              0
2        1       1        1           0          0              1
3        2       1        1           0          1              0
4        3       1        0           0          0              0
5        4       1        1           2          0              1

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
1              247                 50          76                 7
2              112                  0          23                 7
3               74                135          68                 7
4              305                  0          89                 7
5              281                196         159                 7
```

```
   Credit_History  Property_Area  Loan_Status
1               1              0            0
2               1              2            1
3               1              2            1
4               1              2            1
5               1              2            1

     Loan_ID  Gender  Married  Dependents  Education  Self_Employed  \
1          0       1        1           1          0              0
2          1       1        1           0          0              1
3          2       1        1           0          1              0
4          3       1        0           0          0              0
5          4       1        1           2          0              1
..       ...     ...      ...         ...        ...            ...
609      475       0        0           0          0              0
610      476       1        1           3          0              0
611      477       1        1           1          0              0
612      478       1        1           2          0              0
613      479       0        0           0          0              1

     ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
1                247                 50          76                 7
2                112                  0          23                 7
3                 74                135          68                 7
4                305                  0          89                 7
5                281                196         159                 7
..               ...                ...         ...               ...
609              101                  0          26                 7
610              219                  0           7                 4
611              344                  3         154                 7
612              336                  0         126                 7
613              247                  0          81                 7

     Credit_History  Property_Area  Loan_Status
1                 1              0            0
2                 1              2            1
3                 1              2            1
4                 1              2            1
5                 1              2            1
..              ...            ...          ...
609               1              0            1
610               1              0            1
611               1              2            1
612               1              2            1
613               0              1            0

[480 rows x 13 columns]
```

In [17]:
```
X_without_missing_axis0 =train_data_without_missing_axis0.drop(["Loan_Sta

y=train_data_without_missing_axis0["Loan_Status"]

y.value_counts()
```

Out[17]:
```
1    332
0    148
Name: Loan_Status, dtype: int64
```

In [18]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split
X_train, X_test,y_train,y_test = train_test_split(X_without_missing_axis0
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
```

```
pred = lr.predict(X_test)
print(round(metrics.accuracy_score(pred,y_test), 4))
```

```
0.7986
```

```
C:\Users\HP\anaconda3\envs\pyfinance\lib\site-packages\sklearn\linear_mod
el\_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression
  n_iter_i = _check_optimize_result(
```

Supprimer les lignes contenant des valeurs manquantes, c'est une meilleure façon de gérer les valeurs manquantes que de supprimer les colonnes avec des valeurs manquantes, car la précision de notre modéle est passée de 64.86% à 79.86%.

NB : Les colonnes suprimées ont beacoup plus d'informations que prévu.

3. Remplir les valeurs manquantes : imputation

Dans ce cas, nous remplirons les valeurs manquantes avec un certain nombre.

Les manières possibles de le faire sont :

1. Remplir les données manquantes avec la valeur moyenne ou médiane s'il s'agit d'une variable numérique.
2. Remplir les données manquantes avec la mode s'il s'agit d'une variable catégorielle.
3. Remplir la valeur numérique avec 0 ou -999, ou un autre nombre qui n'apparaîtra pas dans les données. Cela peut être fait pour que la machine puisse reconnaître que les données ne sont pas réelles ou sont différentes.
4. Remplir la variable catégorielle avec un nouveau type pour les valeurs manquantes.

Nous pouvons utiliser la fonction pandas **fillna()** pour remplir les valeurs manquantes dans l'ensemble de données.

Remplir les données manquantes avec la valeur moyenne ou médiane s'il s'agit d'une variable numérique et la mode s'il s'agit d'une variable catégorielle

In [19]:
```
# Fonction pour remplir les données manquantes
def transform_features(df):

    df =df.fillna(df.select_dtypes(include=['object']).mode().iloc[0], ax

    df = df.fillna(df.select_dtypes(include=['int', 'float']).mean(), axi

    return df
```

In [20]:
```
transform_df=transform_features(train_data)
```

```
print(transform_df.isnull().sum())
```

```
Loan_ID              0
Gender               0
Married              0
Dependents           0
Education            0
Self_Employed        0
ApplicantIncome      0
CoapplicantIncome    0
LoanAmount           0
Loan_Amount_Term     0
Credit_History       0
Property_Area        0
Loan_Status          0
dtype: int64
```

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

Out[21]:
```
(614, 13)
```

In [22]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

transform_df=transform_df.apply(LabelEncoder().fit_transform)
print(transform_df.head())
print()
print(transform_df.shape)
```

```
   Loan_ID  Gender  Married  Dependents  Education  Self_Employed  \
0        0       1        0           0          0              0
1        1       1        1           1          0              0
2        2       1        1           0          0              1
3        3       1        1           0          1              0
4        4       1        0           0          0              0

   ApplicantIncome  CoapplicantIncome  LoanAmount  Loan_Amount_Term  \
0              376                  0         100                 9
1              306                 60          81                 9
2              139                  0          26                 9
3               90                160          73                 9
4              381                  0          94                 9

   Credit_History  Property_Area  Loan_Status
0               2              2            1
1               2              0            0
2               2              2            1
3               2              2            1
4               2              2            1

(614, 13)
```

In [23]:
```
X=transform_df.drop(["Loan_Status","Loan_ID"], axis=1)

y=transform_df["Loan_Status"]

y.value_counts()
```

Out[23]:
```
1    422
0    192
Name: Loan_Status, dtype: int64
```

In [24]:
```
from sklearn import metrics
from sklearn.model_selection import train_test_split
```

```
X_train, X_test,y_train,y_test = train_test_split(X,y,test_size=0.3, rand
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,y_train)
pred = lr.predict(X_test)
print(round(metrics.accuracy_score(pred,y_test), 4))
```

```
0.7892
```

```
C:\Users\HP\anaconda3\envs\pyfinance\lib\site-packages\sklearn\linear_mod
el\_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (status
=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown i
n:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression
  n_iter_i = _check_optimize_result(
```

La valeur de précision ressort à 78,92%, ce qui est une réduction par rapport au cas précédent.

Cela ne se produira pas en général, dans ce cas, cela signifie que la moyenne et la mode n'ont pas pu remplir correctement les valeurs manquantes.

Références pour aller plus loin

https://towardsdatascience.com/imputing-missing-values-using-the-simpleimputer-class-in-sklearn-99706afaff46

https://vitalflux.com/pandas-impute-missing-values-mean-median-mode/

https://www.analyticsvidhya.com/blog/2021/05/dealing-with-missing-values-in-python-a-complete-guide/

https://towardsdatascience.com/pandas-tricks-for-imputing-missing-data-63da3d14c0d6

https://pandas.pydata.org/pandas-docs/stable/user_guide/missing_data.html

https://scikit-learn.org/stable/modules/generated/sklearn.impute.KNNImputer.html

In [ ]: