Chapitre 3 : SciKit-App

1-Introduction :

Dans ce chapitre nous allons enfin nous attaquer au vif du sujet, appliquer autant que possible de ce qui a été mentionné dans le chapitre précédent afin de donner vie à notre logiciel « SciKit-app ». Dans ce qui suit nous allons donc suivre le cycle de travail du data scientist afin de produire un modèle de Machine Learning et de le déployer.

C'est bien beau d'en parler mais il est temps de nous lancer, la première chose qu'il nous faut c'est un bon dataset et une problématique a résoudre, pour trouver cela il n'y a pas mieux que la communauté Kaggle qui mets a la disposition de tous un nombre énorme de datasets qui croit de jour en jour. Nous avons opté pour un dataset qui porte sur des patients touchés ou non par des accidents vasculaires cérébral (AVC), alors avant de nous lancer dans la description du dataset et de notre problème de **classification** on se posera la question suivante, qu'est-ce qu'un AVC ?

2-Accident vasculaire cérébral :

Les AVCs sont la deuxième cause de mortalité dans le monde, touts pays confondus. En effet, chaque 40 secondes, un américain fait un AVC et chaque quatre minutes un d'entre eux en meurt [1]. C'est l'une des maladies cardiovasculaires les plus pourfendeuses.

2-1-Causes :

Un AVC peut être causé par deux choses ; [2]

- Généralement c'est à cause de l'interruption de l'irrigation d'une partie du cerveau qui veut dire que le cerveau ne reçoit plus de sang, et pas de sang veut dire pas d'oxygène, ce qui est désastreux pour les cellules du cerveau, ceci est du a un caillot de graisse qui bloque une artère, on parles dans ce cas d'AVC Ischémique ou d'infarctus cérébral.
- Dans 20% des cas d'AVC c'est du a autre chose, une hémorragie cérébrale, c'est souvent associé à une paralysie du côté droit ou gauche.

2-2-Conséquences :

Si le taux de mortalité n'est pas assez pour faire peur, il faut savoir que même en survivre peut couter cher ; subir les séquelles d'un AVC peut vouloir dire être paralysé dans les meilleurs cas, dans d'autre bien plus triste c'est la démence, en effet, c'est la deuxième cause de démence en France après la maladie d'Alzheimer. Quoi qu'il en soit, que les patients subissent les séquelles ou pas, ils se voient vivre en sachant que plus de 30% de ceux qui y survivent revivent la même expérience en moins de cinq ans.[2]

3-Description du dataset :

Le dataset « Stroke Prediction Dataset – 11 clinical features for predicting stroke events » est un dataset assez connu dans la communauté de Kaggle, propriété d'un certain fedesoriano un des experts de datasets de la communauté. Le dataset est un fichier CSV mis a la disposition du publique en Février 2021, il est composé d'un total de 5110 lignes de 12 colonnes chaque, 10 paramètres

divers, un identifiant et la sortie résultante de ces paramètres. Le dictionnaire de données suivant est conforme a la description de la source.

Colonne	Type	Description
Id	Entier	Un identifiant unique, valeur vraisemblablement aléatoire comprise entre 67 et 72940.
gender	Chaine	Genre de la personne, chaine de caractères qui est soit 'Male', 'Female' ou 'Other'.
age	Entier	L'âge de la personne.
hypertension	Entier	Valeur booléenne qui est a 1 si le patient a de l'hypertension, 0 sinon.
heart_disease	Entier	Valeur booléenne qui est a 1 si le patient est atteint d'une cardiopathie, 0 sinon.
ever_married	Chaine	Chaine de caractères qui est a 'Yes' si le patient s'est déjà marié, 'No' sinon.
avg_glucose_level	Réel	La moyenne du glucose sanguin du patient.
Residence_type	Chaine	Chaine de caractères qui represente le type de residence du patient, soit 'Rural' soit 'Urban'.
work_type	Chaine	Chaine de caractères qui représente le type du travail du patient, prends une des valeurs 'children', 'Govt_job', 'Never_worked', 'Private' et 'Self-employed'.
bmi	Réel	La valeur de l'IMC (Indice de Masse Corporelle) du patient.
smoking_status	Chaine	Chaine de caractères qui la situation du patient par rapport à la cigarette, prends une des valeurs 'formerly_smoked', 'never_smoked', 'smokes' et 'Unknown'.
stroke	Entier	Valeur booléenne qui représente le résultat attendu de nos caractéristiques, 1 si le patient a eu un AVC, 0 sinon.

On va donc commencer notre travail par importer touts les modules que l'on va utiliser et par récupérer nos données dans un DataFrame Pandas, se référer à l'annexe C pour plus de détails.

```
1  import numpy as np
2  import pandas as pd
3  import seaborn as sns
4  import matplotlib.pyplot as plt
5  from sklearn.model_selection import train_test_split
6  from sklearn.metrics import classification_report
7  from sklearn.metrics import accuracy_score
8  from sklearn.metrics import recall_score
9  from sklearn.model_selection import GridSearchCV
10 from sklearn.linear_model import LogisticRegression
11 from sklearn.svm import SVC
12 from sklearn.ensemble import RandomForestClassifier
13 from sklearn.tree import DecisionTreeClassifier
14 from sklearn.neighbors import KNeighborsClassifier
15 from imblearn.over_sampling import SMOTE
16 import dill
17 avc_data = pd.read_csv("healthcare-dataset-stroke-data.csv",index_col='id')
18 avc_data.head(5)
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	Male	67.0	0	1	Yes	Private	Urban	228.69	36.6	formerly smoked	1
51676	Female	61.0	0	0	Yes	Self-employed	Rural	202.21	NaN	never smoked	1
31112	Male	80.0	0	1	Yes	Private	Rural	105.92	32.5	never smoked	1
60182	Female	49.0	0	0	Yes	Private	Urban	171.23	34.4	smokes	1
1665	Female	79.0	1	0	Yes	Self-employed	Rural	174.12	24.0	never smoked	1

Capture : Récupération des données.

4-Réalisation :

Nous allons maintenant suivre le cycle de travail du data scientist comme vu dans les deux chapitres précédents, mais en premier lieu vient la visualisation de nos données :

4-1-Visualisation :

Le but de cette partie est de comprendre nos données, on commencera donc par revérifier la forme du dataset et afficher son tableau de statistiques.

```
1 avc_data.shape
```

```
(5110, 11)
```

```
1 avc_data.describe()
```

	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

Capture : Forme et description statistique du dataset.

On déduit que la forme est bien conforme a celle donnée sur Kaggle, en remarquera aussi trois autres choses :

- Comme l'indique les moyennes très basses et leurs quartile, il y a beaucoup plus de valeurs à 0 que de valeurs à 1 pour les colonnes hypertension, heart_disease et stroke ce qui indique un déséquilibre dans nos données.
- Le count pour la colonne bmi est inferieur aux autre ce qui indique de valeurs nulles qu'il faudra impérativement nettoyer.
- Les valeurs maximum et de troisième quartile des colonnes bmi et avg_glucose_level sont très éloignées ce qui implique potentiellement la présence de valeurs aberrantes.

Ce que nous allons donc maintenant faire est de trouver quelles colonnes contiennent des valeurs nulles , confirmer graphiquement le déséquilibre de nos données et situer graphiquement les valeurs aberrantes.

```
1  avc_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 5110 entries, 9046 to 44679
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   gender             5110 non-null   object
 1   age                5110 non-null   float64
 2   hypertension       5110 non-null   int64
 3   heart_disease      5110 non-null   int64
 4   ever_married       5110 non-null   object
 5   work_type          5110 non-null   object
 6   Residence_type     5110 non-null   object
 7   avg_glucose_level  5110 non-null   float64
 8   bmi                4909 non-null   float64
 9   smoking_status     5110 non-null   object
 10  stroke             5110 non-null   int64
dtypes: float64(3), int64(3), object(5)
memory usage: 379.3+ KB
```

```
1  avc_data.isna().sum()
```

```
gender                 0
age                    0
hypertension           0
heart_disease          0
ever_married           0
work_type              0
Residence_type         0
avg_glucose_level      0
bmi                  201
smoking_status         0
stroke                 0
dtype: int64
```

On retrouve évidemment que la colonne bmi a des valeurs nulles, 201 pour être précis. Et heureusement pour nous, les autres colonnes sont consistantes et n'ont pas la moindre valeur à nulle. Ce qu'on remarquera par contre est que toute nos colonnes qui ont des valeurs de type chaine de caractères ont été chargées sous forme d'objet, chose qui pourrait nous poser des problèmes plus tard.

Capture : Colonnes qui ont des valeurs nulles.

Pour voir si nos données sont équilibrées on utilisera des countplots pour toutes nos colonnes catégoriques « voir la figure 1 ». Effectivement, le nombre de 1 est quasiment négligeable par rapport au nombre de 0 ce qui ne peut que nous mener vers des modèles souffrant d'overfitting, incapables de renvoyer un 1, seulement des 0. On remarquera par contre deux autres chose ; le nombre d'occurrences de 'other' dans la colonne gender n'est même pas visible car elle n'apparait qu'une seule fois « voir fig », on devra donc s'en débarrasser durant le nettoyage. La valeur 'Unknown' qui est sémantiquement l'équivalent de nul se répète très souvent (1544 fois « voir fig ») dans la colonne smoking_status, on la traitera comme les valeurs nulles.

```
1  avc_data['gender'].value_counts()
```

```
Female    2994
Male      2115
Other        1
Name: gender, dtype: int64
```

```
1  avc_data['smoking_status'].value_counts()
```

```
never smoked       1892
Unknown            1544
formerly smoked     885
smokes              789
Name: smoking_status, dtype: int64
```

Capture : Value counts des colonnes gender et smoking_status

Pour cibler les valeurs aberrantes dans nos colonnes continues il nous faudra des boites à moustaches (boxplot an anglais), on en profitera pour vérifier la répartition de leurs valeurs par rapport à la colonne cible stroke. Comme en le voit sur la figure les colonnes bmi et avg_glucose_level ont effectivement assez beaucoup de valeurs aberrantes (les points hors des

boites), la colonne age par contre n'a pas ce problème, ce qu'on peut déduire depuis cette dernière est que les personnes entre 60 et 80 ans sont plus susceptibles de faire un AVC que les plus jeunes, le taux de glucose ou IMC par contre non.

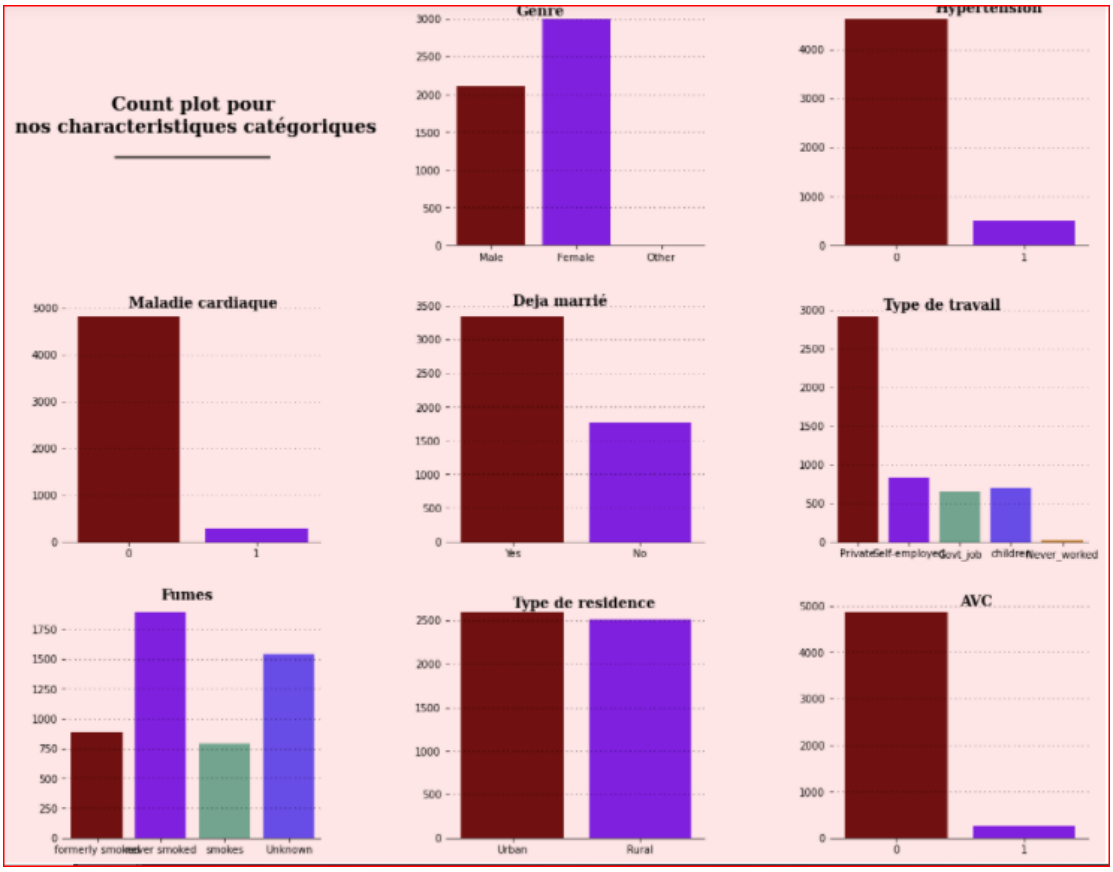


Figure Countplots pour les colonnes catégoriques du dataset.

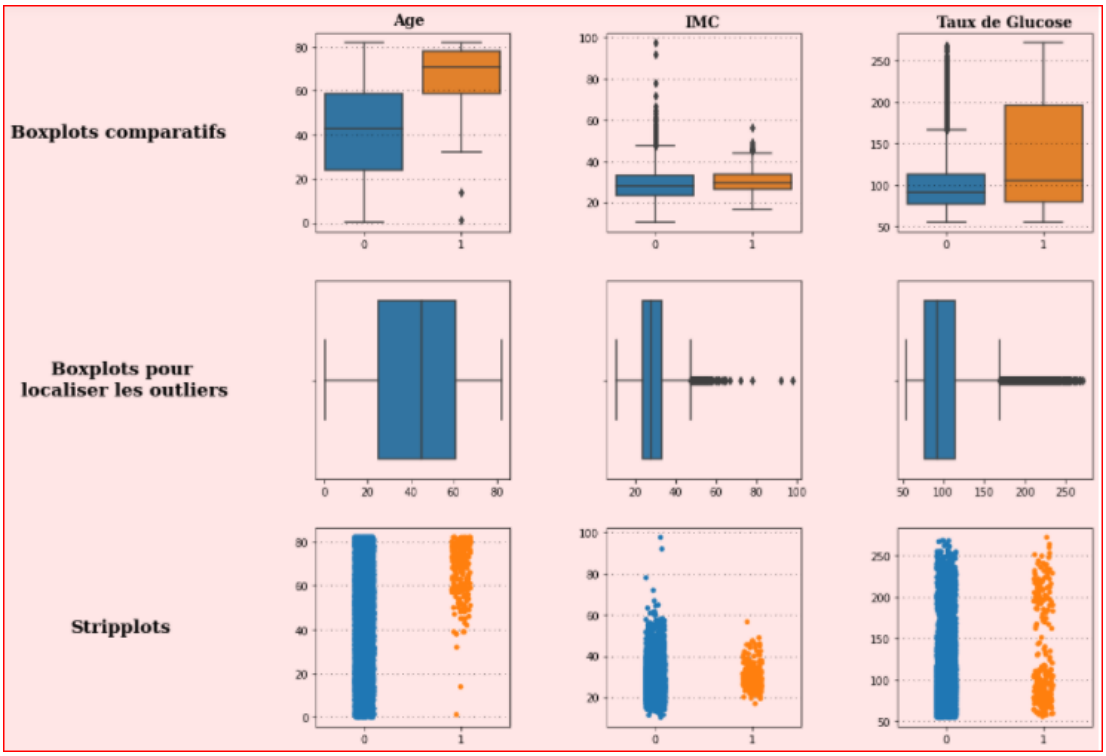


Figure Boxplots, boxplots comparatifs et stripplots pour nos colonnes continues.

Ceci conclut notre étape de visualisation, nous avons une idée assez claire des problèmes que l'on a rencontré avec nos données, nous allons donc nous atteler au nettoyage afin d'y palier du mieux possible.

4-2-Nettoyage et préparation:

Dans cette partie nous allons dans un premier temps résoudre les problèmes qu'on a trouvés durant la visualisation voir même plus si d'autre problèmes émergent en suivant la tactique problème=>stratégie=>execution, et en second lieu nous allons préparer nos données pour la modélisation.

4-2-1-Les valeurs nulles :

Problème : 201 lignes ont la valeur nulle pour la colonne bmi.

Stratégie : Remplir ces vides par la moyenne de l'IMC (28.49) tout en vérifiant que l'écart-type ne change pas trop.

Exécution :

```
1 avc_data['bmi'].std()
```

```
7.854066729680158
```

```
1 avc_data.fillna(value={'bmi':avc_data['bmi'].mean()},inplace=True)
```

```
1 avc_data['bmi'].std()
```

```
7.698017826857077
```

```
1 avc_data.isna().sum()
```

```
gender              0
age                 0
hypertension        0
heart_disease       0
ever_married        0
work_type           0
Residence_type      0
avg_glucose_level   0
bmi                 0
smoking_status      0
stroke              0
dtype: int64
```

Capture : Traiter les valeurs nulles.

La valeur de l'écart-type est passée d'environs 7.85 à 7.7, on dira que cet écart est négligeable et que la valeur est plus ou moins conservée. Et ainsi, nous n'avons plus de valeurs nulles dans nos données.

4-2-2-Traiter les outliers :

Problème : De nombreuses valeurs aberrantes dans les colonnes avg_glucose_level et bmi.

Stratégie : Définir une fonction qui renvoie les lignes avec des outliers pour les traiter ; toutes les valeurs inferieurs à la borne inferieure (on mettra Q1-1.5(Q3-Q1)) sera remplacé par la borne inferieure, de même pour les valeurs qui dépassent notre borne supérieure (Q1-1.5(Q3+Q1)) qui seront remplacées par la valeur de la borne sup.

Exécution : Dans un premier lieu on définit la fonction ;

```
def trouver_outliers(data,var_name):
    iqr = data[var_name].quantile(0.75) -  data[var_name].quantile(0.25)
    born_inf = data[var_name].quantile(0.25) - 1.5*iqr
    born_sup = data[var_name].quantile(0.75) + 1.5*iqr
    return data[(data[var_name]<born_inf) | (data[var_name]>born_sup)]
```

Capture : Fonction qui renvoie les valeurs aberrantes

Ensuite on traite les outliers de la colonne bmi :

```
outliers_bmi = trouver_outliers(avc_data,'bmi')
outliers_bmi.shape
```

```
(126, 11)
```

```
iqr = avc_data['bmi'].quantile(0.75) -  avc_data['bmi'].quantile(0.25)
avc_data.loc[(trouver_outliers(avc_data,'bmi').index , 'bmi')] = avc_data['bmi'].quantile(0.75) + 1.5*iqr
```

```
outliers_bmi = trouver_outliers(avc_data,'bmi')
outliers_bmi.shape
```

```
(0, 11)
```

Capture : Traitement des outliers de bmi

Suivi de ceux de la colonne avg_glucose_level :

```
outliers_gluc = trouver_outliers(avc_data,'avg_glucose_level')
outliers_gluc.shape
```

```
(627, 11)
```

```
iqr = avc_data['avg_glucose_level'].quantile(0.75) -  avc_data['avg_glucose_level'].quantile(0.25)
avc_data.loc[(trouver_outliers(avc_data,'avg_glucose_level').index , 'avg_glucose_level')]
                              = avc_data['avg_glucose_level'].quantile(0.75) + 1.5*iqr
```

```
outliers_gluc = trouver_outliers(avc_data,'avg_glucose_level')
outliers_gluc.shape
```

```
(0, 11)
```

Capture : traitement des outliers de avg_glucose_level

Il ne reste plus qu'à vérifier graphiquement que les valeurs aberrantes ne sont plus, voir la figure.

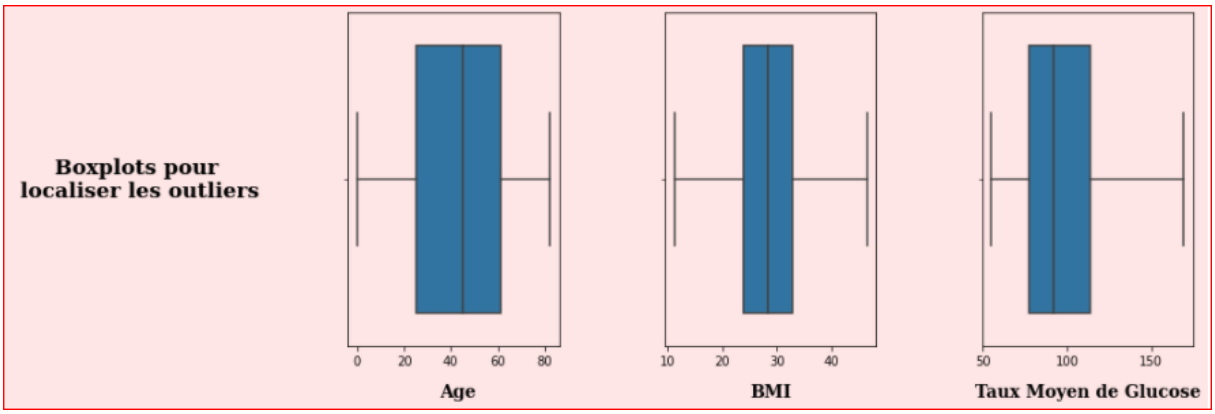


Figure boxplots après traitement des outliers.

4-2-3-Les valeurs ‘other’ et ‘Unknown’ :

Problème 1 : Une seule ligne a pour valeur ‘other’ pour la colonne gender.

Stratégie : Remplacer ‘other’ par ‘Male’.

Exécution :

```
1 avc_data[avc_data['gender']=='Other']
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
56156	Other	26.0	0	0	No	Private	Rural	143.33	22.4	formerly smoked	0

```
1 avc_data.replace('Other','Male',inplace=True)
```

```
1 avc_data[avc_data['gender']=='Other']
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke

Capture : Traitement de la valeur ‘other’

Problème 2 : 1544 lignes ont pour valeur ‘Unknown’ pour la colonne smoking_status.

Stratégie : Remplacer par les autres valeurs tout en préservant leur répartition i-e préserver la variance entre les autres valeurs i-e la valeur de la fraction nombre d’occurrence/nombre de lignes-nombre de ‘Unknown’ reste inchangée.

Exécution : Initialement la valeur de la fraction est 0.53, 0.25 et 0.22 pour ‘never smoked’, ‘formerly smoked’ et ‘smokes’ respectivement, pour préserver cette fraction il faudra donc répartir les 1544 lignes en 818, 386 et 340 pour ces valeurs.

La procédure est donc ; récupérer les n premières lignes avec la valeur ‘Unknown’=> les retirer de notre dataframe =>les traiter. Puis à la fin, concaténer le tout.

```
df1 = avc_data[avc_data['smoking_status'] == 'Unknown'].head(340)
df1["smoking_status"].replace({"Unknown":"smokes"},inplace=True)
avc_data.drop(avc_data.loc[avc_data['smoking_status']==1].head(340).index, inplace=True)
df1
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
27419	Female	59.00	0	0	Yes	Private	Rural	76.1500	28.893237	smokes	1
60491	Female	78.00	0	0	Yes	Private	Urban	58.5700	24.200000	smokes	1
8213	Male	78.00	0	1	Yes	Private	Urban	169.3575	28.893237	smokes	1
25226	Male	57.00	0	1	No	Govt_job	Urban	169.3575	28.893237	smokes	1
64778	Male	82.00	0	1	Yes	Private	Rural	169.3575	32.500000	smokes	1
...	...	...	...	...	...	...	...	...	...	...	...
533	Female	3.00	0	0	No	children	Rural	94.1200	21.400000	smokes	0
45554	Female	1.24	0	0	No	children	Urban	62.4000	22.100000	smokes	0
55744	Male	2.00	0	0	No	children	Urban	76.2500	20.100000	smokes	0
25767	Female	30.00	0	0	No	Private	Urban	96.4200	22.600000	smokes	0
71319	Male	15.00	0	0	No	Private	Rural	78.5900	25.100000	smokes	0

340 rows × 11 columns

Capture : Les nouvelles lignes avec smoking_status=smokes

```
df2 = avc_data[avc_data['smoking_status'] == 'Unknown'].head(386)
df2["smoking_status"].replace({"Unknown":"formerly smoked"},inplace=True)
avc_data.drop(avc_data.loc[avc_data['smoking_status']==1].head(386).index, inplace=True)
df2
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
27419	Female	59.0	0	0	Yes	Private	Rural	76.1500	28.893237	formerly smoked	1
60491	Female	78.0	0	0	Yes	Private	Urban	58.5700	24.200000	formerly smoked	1
8213	Male	78.0	0	1	Yes	Private	Urban	169.3575	28.893237	formerly smoked	1
25226	Male	57.0	0	1	No	Govt_job	Urban	169.3575	28.893237	formerly smoked	1
64778	Male	82.0	0	1	Yes	Private	Rural	169.3575	32.500000	formerly smoked	1
...	...	...	...	...	...	...	...	...	...	...	...
51959	Male	12.0	0	0	No	children	Rural	81.7400	28.300000	formerly smoked	0
3956	Male	13.0	0	0	No	children	Urban	65.5100	25.900000	formerly smoked	0
42703	Male	74.0	0	0	Yes	Self-employed	Urban	61.7800	25.800000	formerly smoked	0
34436	Female	2.0	0	0	No	children	Rural	109.5600	16.400000	formerly smoked	0
2772	Male	55.0	0	0	Yes	Private	Urban	87.7200	27.000000	formerly smoked	0

386 rows × 11 columns

Capture : Les nouvelles lignes avec smoking_status=formerly smoked

```
df3 = avc_data[avc_data['smoking_status'] == 'Unknown'].head(818)
df3["smoking_status"].replace({"Unknown":"never smoked"},inplace=True)
avc_data.drop(avc_data.loc[avc_data['smoking_status']==1].head(818).index, inplace=True)
df3
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
27419	Female	59.0	0	0	Yes	Private	Rural	76.1500	28.893237	never smoked	1
60491	Female	78.0	0	0	Yes	Private	Urban	58.5700	24.200000	never smoked	1
8213	Male	78.0	0	1	Yes	Private	Urban	169.3575	28.893237	never smoked	1
25226	Male	57.0	0	1	No	Govt_job	Urban	169.3575	28.893237	never smoked	1
64778	Male	82.0	0	1	Yes	Private	Rural	169.3575	32.500000	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...
60907	Male	48.0	0	0	Yes	Private	Rural	127.1300	35.000000	never smoked	0
12449	Female	34.0	0	0	Yes	Private	Rural	119.6100	26.400000	never smoked	0
2707	Male	10.0	0	0	No	children	Rural	68.9400	18.000000	never smoked	0
49120	Female	39.0	0	0	Yes	Govt_job	Rural	69.3800	22.100000	never smoked	0
7665	Female	73.0	0	0	Yes	Private	Rural	98.3400	30.900000	never smoked	0

818 rows × 11 columns

Capture : Les nouvelles lignes avec smoking_status=never smoked

```
avc_data.drop(avc_data.loc[avc_data['smoking_status']=='Unknown'].index, inplace=True)
avc_data[avc_data['smoking_status'] == 'Unknown']
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke

```
df4 = pd.concat([df2,df1])
df4 = pd.concat([df4,df3])
avc_data = pd.concat([avc_data,df4])
avc_data['smoking_status'].value_counts()
avc_data
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	Male	67.0	0	1	Yes	Private	Urban	169.3575	36.600000	formerly smoked	1
51676	Female	61.0	0	0	Yes	Self-employed	Rural	169.3575	28.893237	never smoked	1
31112	Male	80.0	0	1	Yes	Private	Rural	105.9200	32.500000	never smoked	1
60182	Female	49.0	0	0	Yes	Private	Urban	169.3575	34.400000	smokes	1
1665	Female	79.0	1	0	Yes	Self-employed	Rural	169.3575	24.000000	never smoked	1
...	...	...	...	...	...	...	...	...	...	...	...
60907	Male	48.0	0	0	Yes	Private	Rural	127.1300	35.000000	never smoked	0
12449	Female	34.0	0	0	Yes	Private	Rural	119.6100	26.400000	never smoked	0
2707	Male	10.0	0	0	No	children	Rural	68.9400	18.000000	never smoked	0
49120	Female	39.0	0	0	Yes	Govt_job	Rural	69.3800	22.100000	never smoked	0
7665	Female	73.0	0	0	Yes	Private	Rural	98.3400	30.900000	never smoked	0

5110 rows × 11 columns

Capture : Le dataframe complet après élimination de la valeur ‘Unknown’

Et ainsi, la valeur ‘Unknown’ n’est plus et la variance entre les autres valeurs est preservée.

4-2-4-Codifier les données :

Dans l’esprit de la préparation des données il est nécessaire de codifier les colonnes catégorique en colonnes à valeurs entières, on optera donc pour la codification du tableau, on en profitera aussi pour changer les types de colonnes depuis le type object à entier.

gender	0 pour Female. 1 pour Male.
hypertension	Pas de changement.
heart_disease	Pas de changement.
ever_married	0 pour No. 1 pour Yes.
Residence_type	0 pour Rural. 1 pour Urban.
work_type	0 pour Private. 1 pour Self_employed. 2 pour Govt_job. 3 pour children. 4 pour Never_worked.
smoking_status	0 pour never_smoked. 2 pour formerly_smoked. 3 pour smokes.

```
avc_data["gender"].replace({"Female":"0","Male":"1"},inplace=True)
avc_data["Residence_type"].replace({"Rural":"0","Urban":"1"},inplace=True)
avc_data["ever_married"].replace({"No":"0","Yes":"1"},inplace=True)
avc_data["smoking_status"].replace({"never smoked":"0","formerly smoked":"2","smokes":"3"},inplace=True)
avc_data["work_type"].replace({"Private":"0","Self-employed":"1","Govt_job":"2","children":"3","Never_worked":"4"},inplace=T
avc_data = avc_data.astype({"gender":"int64","ever_married":"int64","work_type":"int64","Residence_type":"int64","smoking_st
avc_data
```

id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	1	67.0	0	1	1	0	1	169.3575	36.600000	2	1
51676	0	61.0	0	0	1	1	0	169.3575	28.893237	0	1
31112	1	80.0	0	1	1	0	0	105.9200	32.500000	0	1
60182	0	49.0	0	0	1	0	1	169.3575	34.400000	3	1
1665	0	79.0	1	0	1	1	0	169.3575	24.000000	0	1
...	...	...	...	...	...	...	...	...	...	...	...
60907	1	48.0	0	0	1	0	0	127.1300	35.000000	0	0
12449	0	34.0	0	0	1	0	0	119.6100	26.400000	0	0
2707	1	10.0	0	0	0	3	0	68.9400	18.000000	0	0
49120	0	39.0	0	0	1	2	0	69.3800	22.100000	0	0
7665	0	73.0	0	0	1	0	0	98.3400	30.900000	0	0

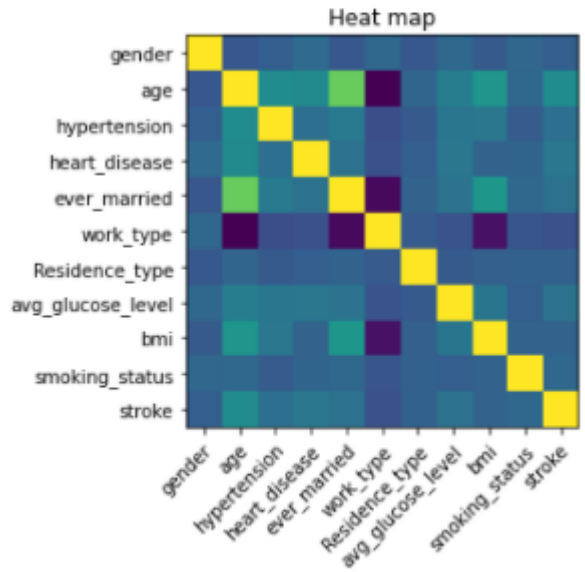
5110 rows × 11 columns

Capture : Codification des colonnes catégoriques et mise à jour des types.

On profitera de cette nouvelle codification pour consulter la table des corrélations de notre dataframe avec une Heatmap;

```
fig,ax = plt.subplots()
im = ax.imshow(avc_data.corr())
ax.set_xticks(np.arange(len(list(avc_data))))
ax.set_yticks(np.arange(len(list(avc_data))))
ax.set_xticklabels(list(avc_data))
ax.set_yticklabels(list(avc_data))
plt.setp(ax.get_xticklabels(), rotation=45, ha="right",rotation_mode="anchor")
ax.set_title("Heat map")
```

```
Text(0.5, 1.0, 'Heat map')
```



Capture : Heatmap de la table de corrélations.

On remarquera qu'il y a peu de valeurs de corrélation notables et elles ne sont pas en relation avec la colonne cible, on déduit donc que certaines de nos colonnes ne sont pas indépendantes entre-elles, on optera alors pour éliminer celles dont la valeur de corrélation surpasse 0.6 car elles sont superflues.

```python
def correlation(data,seuil):
    col_corr = set()
    corr_matrix = data.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):
            if abs(corr_matrix.iloc[i,j])>seuil :
                colname=corr_matrix.columns[i]
                col_corr.add(colname)
    return col_corr
```

```python
corr_features = correlation(avc_data,0.6)
corr_features
```

```
{'ever_married'}
```

```python
avc_data = avc_data.drop(corr_features,axis=1)
avc_data
```

id	gender	age	hypertension	heart_disease	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
9046	1	67.0	0	1	0	1	169.3575	36.600000	2	1
51676	0	61.0	0	0	1	0	169.3575	28.893237	0	1
31112	1	80.0	0	1	0	0	105.9200	32.500000	0	1
60182	0	49.0	0	0	0	1	169.3575	34.400000	3	1
1665	0	79.0	1	0	1	0	169.3575	24.000000	0	1
...	...	...	...	...	...	...	...	...	...	...
60907	1	48.0	0	0	0	0	127.1300	35.000000	0	0
12449	0	34.0	0	0	0	0	119.6100	26.400000	0	0
2707	1	10.0	0	0	3	0	68.9400	18.000000	0	0
49120	0	39.0	0	0	2	0	69.3800	22.100000	0	0
7665	0	73.0	0	0	0	0	98.3400	30.900000	0	0

5110 rows × 10 columns

Capture : Elimination des colonnes superflues.

La seule colonne touchée est celle de ever_married qui dépend trop de l'age.

4-2-5-Diviser les données :

On est maintenant à la dernière frontière qui nous sépare de l'étape de modélisation, il ne nous reste plus qu'à diviser notre dataset en dataset d'entrainement et dataset de test, on optera pour le ratio 80%-20%.

```
seed = 96
X = avc_data.drop(['stroke'],axis=1)
Y = avc_data['stroke']
X_train,X_test,Y_train,Y_test = train_test_split(X,Y,test_size=0.2,random_state=seed,stratify=Y)
print("La forme de X_train est ",X_train.shape)
print("La forme de Y_train est ",Y_train.shape)
print("La forme de X_test est ",X_test.shape)
print("La forme de Y_test est ",Y_test.shape)
```

```
La forme de X_train est  (4088, 9)
La forme de Y_train est  (4088,)
La forme de X_test est  (1022, 9)
La forme de Y_test est  (1022,)
```

Capture : Division du dataset en trainset et testset.

4-3-Modélisation :

Les modèles qu'on a opté en entrainer sont celui de la régression logistique, SVM, arbre décisionnel, forêt aléatoire et K-plus proche voisin. On les comparera tous dans le tableau après leur entrainement.

La procédure qu'on a suivi pour l'entrainement des modèles est plus ou moins exactement la même pour tout les modèles ; on instancie le modèle, instancie un GridSearchCV avec les hyper-paramètres qu'on veut et on lance l'entrainement avec la méthode fit. Les résultats de notre entrainement qu'on affiche sont ; le meilleur score rencontré durant l'entrainement, la meilleure combinaison d'hyper-paramètres, le score de précision et le rapport de classification.

```
lr = LogisticRegression(random_state = seed)
lr_hyp = {'C':[0.0001,0.001,0.01,0.1,1,10,100],'max_iter':[50,75,100,120,150]}
lr_cv = GridSearchCV(lr,lr_hyp,cv=5)
lr_cv.fit(X_train,Y_train)
print(lr_cv.best_score_)
print(lr_cv.best_estimator_)
print("Le score de précision du test du modèle de la regression logistique est ", accuracy_score(Y_test,lr_cv.predict(X_te
print(classification_report(Y_test,lr_cv.predict(X_test)))
```

```
0.9337085107720116
LogisticRegression(C=1, max_iter=150, random_state=96)
Le score de précision du test du modèle de la regression logistique est  0.9315068493150684
              precision    recall  f1-score   support

           0       0.93      1.00      0.96       953
           1       0.00      0.00      0.00        69

    accuracy                           0.93      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.87      0.93      0.90      1022
```

Capture : Entrainement avec LogisticRegression.

```
sv = SVC(random_state =seed)
sv_hyp = {'C':[0.001,0.01,0.1,1,10,100],'degree':[1,2,3,4,5,6]}
sv_cv = GridSearchCV(sv,sv_hyp,cv=5)
sv_cv.fit(X_train,Y_train)
print(sv_cv.best_score_)
print(sv_cv.best_estimator_)
print("Le score de précision du test du modèle SVM est ", accuracy_score(Y_test,sv_cv.predict(X_test)))
print(classification_report(Y_test,sv_cv.predict(X_test)))
```

```
0.9329747151753838
SVC(C=0.001, degree=1, random_state=96)
Le score de précision du test du modèle SVM est  0.9324853228962818
              precision    recall  f1-score   support

           0       0.93      1.00      0.97       953
           1       0.00      0.00      0.00        69

    accuracy                           0.93      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.87      0.93      0.90      1022
```

Capture : Entrainement avec SVC.

```
dt =DecisionTreeClassifier(random_state =seed)
dt_hyp = {'max_depth':[2,5,10,15,20]}
dt_cv = GridSearchCV(dt,dt_hyp,cv=5)
dt_cv.fit(X_train,Y_train)
print(dt_cv.best_score_)
print(dt_cv.best_estimator_)
print("Le score de précision du test du modèle de l'arbre aléatoire est ", accuracy_score(Y_test,dt_cv.predict(X_test)))
print(classification_report(Y_test,dt_cv.predict(X_test)))
```

```
0.9329747151753838
DecisionTreeClassifier(max_depth=2, random_state=96)
Le score de précision du test du modèle de l'arbre aléatoire est  0.9324853228962818
              precision    recall  f1-score   support

           0       0.93      1.00      0.97       953
           1       0.00      0.00      0.00        69

    accuracy                           0.93      1022
   macro avg       0.47      0.50      0.48      1022
weighted avg       0.87      0.93      0.90      1022
```

Capture : Entrainement avec DecisionTreeClassifier.

```
rf =RandomForestClassifier(random_state =seed)
rf_hyp = {'n_estimators':[5,10,20,50,100,200],'max_depth':[2,5,10,15,20]}
rf_cv = GridSearchCV(rf,rf_hyp,cv=5)
rf_cv.fit(X_train,Y_train)
print(rf_cv.best_score_)
print(rf_cv.best_estimator_)
print("Le score de précision du test du modèle de l'arbre aléatoire est ", accuracy_score(Y_test,rf_cv.predict(X_test)))
print(classification_report(Y_test,rf_cv.predict(X_test)))
```

```
0.9437368510831865
RandomForestClassifier(max_depth=20, random_state=96)
Le score de précision du test du modèle de l'arbre aléatoire est  0.9481409001956947
              precision    recall  f1-score   support

           0       0.95      1.00      0.97       953
           1       0.90      0.26      0.40        69

    accuracy                           0.95      1022
   macro avg       0.92      0.63      0.69      1022
weighted avg       0.95      0.95      0.93      1022
```

Capture : Entrainement avec RandomForestClassifier.

```
kn =KNeighborsClassifier(n_neighbors=5)
kn_hyp = {'n_neighbors':[1,2,3,4,5,10,15,20]}
kn_cv = GridSearchCV(kn,kn_hyp,cv=5)
kn_cv.fit(X_train,Y_train)
print(kn_cv.best_score_)
print(kn_cv.best_estimator_)
print("Le score de précision du test du modèle de l'arbre aléatoire est ", accuracy_score(Y_test,kn_cv.predict(X_test)))
print(classification_report(Y_test,kn_cv.predict(X_test)))
```

```
0.9322409195787559
KNeighborsClassifier(n_neighbors=20)
Le score de précision du test du modèle de l'arbre aléatoire est  0.9315068493150684
              precision    recall  f1-score   support

           0       0.93      1.00      0.96       953
           1       0.40      0.03      0.05        69

    accuracy                           0.93      1022
   macro avg       0.67      0.51      0.51      1022
weighted avg       0.90      0.93      0.90      1022
```

Capture : Entrainement avec KNeighboursClassifier.

La première chose que l'on remarque dans nos résultats et le score de précision qui frôle l'idéal pour un algorithme de Machine-Learning mais le rapport de classification n'est pas aussi positif, en effet, il nous montre un parti-pris indéniable vers la classe 0 pour tout nos modèles. Ce qui confirme l'une des peurs que nous avons cités précédemment ; le déséquilibre de nos données nous a mené vers l'overfitting, nos modèles ne peuvent pas reconnaitre efficacement les cas ou une personne peut être victime d'un AVC, plus spécialement les trois premiers modèles, le rapport nous montre qu'ils n'ont pas prédit la moindre fois la valeur 1, pour chacune des 1022 valeurs de test ils ont prédit un 0. Le modèle KNN n'a pas échoué aussi lamentablement car il a prédit un 1 environ 5 fois et n'a été correcte que deux fois. Le modèle de forêt aléatoire par contre a prouvé qu'il était effectivement le plus approprié pour palier à l'oversampling, il a prédit la valeur 1 environ 20 fois et a été correct 18 fois.

Le modèle de forêt aléatoire est donc bien mieux que nos autres modèles, mais rien n'empêche qu'ils sont toujours tous très peu fiables, d'où nous allons appliquer la technique d'upsampling sur notre dataframe « voir l'annexe pour plus de détails ».

```
# Upsampling
sm = SMOTE(random_state = 2)
X_train_res, Y_train_res = sm.fit_sample(X_train,Y_train.ravel())
```

Capture : Application de l'upsampling.

Après application de l'upsampling notre problème de déséquilibre de données n'est plus que de l'histoire ancienne, nous allons maintenant vérifier l'effet que cela a eu sur nos modèles avec les tables x et y.

On remarque une amélioration indéniable avec nos trois modèle les plus trouble-fêtes, ils sont passé de prédire la valeur 1 aucune fois à la prédire quelques centaines de fois, malgré cela ils restent toujours assez peu fiable mais toujours bien mieux qu'avant. Le modèle KNN lui a pu prédire

la valeur 1 113 fois et a été correcte une fois sur trois. Le modèle de forêt aléatoire par contre agis très différemment des autres après upsampling, il a prédit la valeur 1 le moins souvent (environs 87 fois) et a été correct prés de 33 fois, il reste néanmoins celui avec la meilleur valeur de f1-score i-e le meilleur alliage entre precision et recall et est donc notre meilleur modèle !

La dernière chose qu'il nous reste a faire maintenant est de sauvegarder nos modèles pour l'étape de déploiement qui suivra.

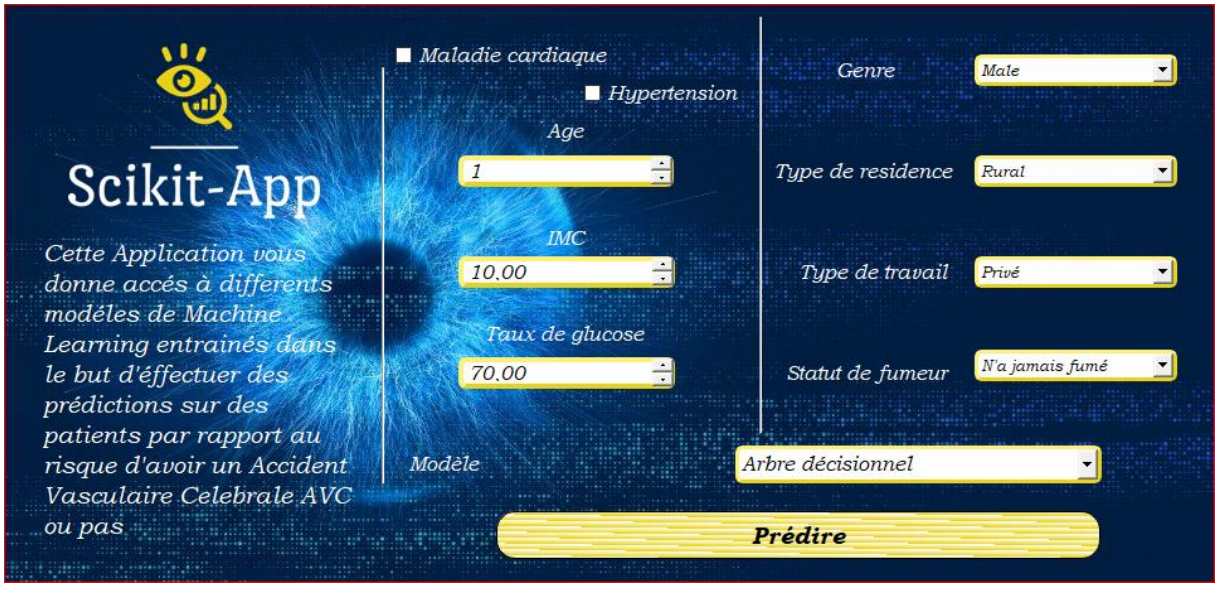
```
with open('Machine à vecteur support Finale.pk1','wb') as f:
    dill.dump(lr_cv, f)
with open('Regression logistique Finale.pk1','wb') as f:
    dill.dump(sv_cv, f)
with open('K-Plus proche voisin Finale.pk1','wb') as f:
    dill.dump(kn_cv, f)
with open('Arbre décisionnel Finale.pk1','wb') as f:
    dill.dump(dt_cv, f)
with open('Forêt aléatoire Finale.pk1','wb') as f:
    dill.dump(rf_cv, f)
```

Capture : Sauvegarde de modèles.

4-4-Déploiement :

Nous avons optés pour deux approches vis-à-vis du déploiement ;

-Une application de bureau qui donne accès aux modèles uniquement pour faire des prédictions, l'utilisateur n'est pas sensé savoir comment fonctionne le modèle voir même quoi que ce soit en Machine-Learning, l'utilisateur cible est un utilisateur lambda.



Capture : Formulaire de l'application de bureau.

-Une application web qui aborde le sujet d’une manière plus pédagogique et donne accès à plus d’informations sur le fonctionnement des prédictions, l’utilisateur cible est quiconque souhaite comprendre notre travail.