Diabete ML1

Import Pandas et le .csv

```
import pandas as pd

df = pd.read_csv("./data/test_without_class.csv")

df.head()
```

	ID	Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	d h
0	417	50	Female	No	No	No	Yes	No	No	Yes	Yes	No	Y
1	418	55	Male	No	Yes	No	Yes	No	Yes	No	No	Yes	Y
2	419	67	Male	Yes	Yes	No	Yes	Yes	Yes	No	Yes	Yes	N
3	420	45	Male	No	No	No	No	Yes	Yes	No	No	No	N
4	421	37	Male	No	No	No	No	No	No	No	No	No	N

Vérifie si il ya des données manquantes dans certains index

```
df.isna().sum()
```

```
ID                  0
Age                 0
Gender              0
Polyuria            0
Polydipsia          0
sudden weight loss  0
weakness            0
Polyphagia          0
Genital thrush      0
visual blurring     0
Itching             0
Irritability        0
delayed healing     0
partial paresis     0
muscle stiffness    0
Alopecia            0
Obesity             0
dtype: int64
```

Vérifie le type de données dans test_without_class.csv

```
df.dtypes
```

```
ID                   int64
Age                  int64
Gender              object
Polyuria            object
Polydipsia          object
sudden weight loss  object
weakness            object
Polyphagia          object
Genital thrush      object
visual blurring     object
Itching             object
Irritability        object
delayed healing     object
partial paresis     object
muscle stiffness    object
Alopecia            object
Obesity             object
dtype: object
```

Nettoyage des colonnes, transforme les object en 'boolean' (int)

```
for col in df.select_dtypes(include='object').columns:
    df[col] = df[col].map({"Yes":1,"No":0, "Male":1, "Female":0, "Positive":1, "Negative":0})
df.head()
```

Age	Gender	Polyuria	Polydipsia	sudden weight loss	weakness	Polyphagia	Genital thrush	visual blurring	Itching	Irritability	delayed healing
50	0	0	0	0	1	0	0	1	1	0	1
55	1	0	1	0	1	0	1	0	0	1	1
67	1	1	1	0	1	1	1	0	1	1	0
45	1	0	0	0	0	1	1	0	0	0	0
37	1	0	0	0	0	0	0	0	0	0	0

```
df['Age'].describe() # Détails la colonne age, pas d'incohérence
```

```
count    104.000000
mean      48.288462
std       12.263034
min       27.000000
25%       39.000000
50%       47.000000
75%       56.250000
```

```
max       90.000000
Name: Age, dtype: float64
```

Corrige les index, tout en minuscule et snake_case

```
df.columns = (df
                .columns
                .str
                .replace(' ', '_')
                .str
                .lower()
                .str
                .strip())
df.columns
```

```
Index(['id', 'age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
       'weakness', 'polyphagia', 'genital_thrush', 'visual_blurring',
       'itching', 'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity'],
      dtype='object')
```

id en majuscule

```
df.rename(columns={'id': 'ID'}, inplace=True)
df.columns
```

```
Index(['ID', 'age', 'gender', 'polyuria', 'polydipsia', 'sudden_weight_loss',
       'weakness', 'polyphagia', 'genital_thrush', 'visual_blurring',
       'itching', 'irritability', 'delayed_healing', 'partial_paresis',
       'muscle_stiffness', 'alopecia', 'obesity'],
      dtype='object')
```

Export en .csv vers ./data/test_clean.csv

```
df.to_csv('./data/test_clean.csv', index=False)
df_clean = pd.read_csv('./data/test_clean.csv')
df_clean.head()
```

	ID	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visual_blur
0	417	50	0	0	0	0	1	0	0	1
1	418	55	1	0	1	0	1	0	1	0
2	419	67	1	1	1	0	1	1	1	0
3	420	45	1	0	0	0	0	1	1	0

	ID	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visual_blur
4	421	37	1	0	0	0	0	0	0	0

Machine Learning

import les bibliothéques nécessaires

```
import pandas as pd
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
# from sklearn.preprocessing import LabelEncoder
# le = LabelEncoder()
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_val_score, GridSearchCV
```

Récuperer le csv 'train' (pour entrainer avec la colonne Class)

```
df_train = pd.read_csv('data/diabetes_clean.csv')
df_train.head()
```

	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visual_blurring
0	60	0	1	0	1	1	0	1	1
1	85	1	1	1	1	1	1	1	1
2	48	1	1	1	1	0	1	1	0
3	41	1	1	1	1	1	1	1	1
4	57	1	0	0	0	0	1	0	1

Définir le paramétre à déterminer pour le Model

```
y = df_train['class']
```

On abandonne la colonne Class pour entrainer le Model & Défini Train et Test

```
# Préparer X (train)
X_train = df_train.drop(columns=['class'], inplace=True)
# Définir les valeurs Train
X = df_train.copy()
# Définir les valeurs Test
X_test = df_clean.copy()

X.head()
```

	age	gender	polyuria	polydipsia	sudden_weight_loss	weakness	polyphagia	genital_thrush	visual_blurring
0	60	0	1	0	1	1	0	1	1
1	85	1	1	1	1	1	1	1	1
2	48	1	1	1	1	0	1	1	0
3	41	1	1	1	1	1	1	1	1
4	57	1	0	0	0	0	1	0	1

Création de nos Arbres et Lancement Model

```
def cross_val_boost(X, y):
    # Paramétres des arbres de décision
    param_grid = {
        'n_estimators': [100, 200, 300, 500],  # Ajouter 500
        'max_depth': [3, 4, 5, 6, 7],  # Tester 4 et 6
        'min_samples_split': [2, 5, 10, 20],

    }
    # Paramétre de Cross validations avec les paramétres des arbres
    grid_search = GridSearchCV(
        RandomForestClassifier(),
        param_grid,
        cv=5,
        scoring='accuracy',
        n_jobs=-1
    )
    # Montre le meilleurs score de nos paramétrages
    grid_search.fit(X, y) # Lancer le model
    print(f"Best parameters: {grid_search.best_params_}")
    print(f"Best cross-validation score: {grid_search.best_score_}")
    return grid_search.best_estimator_
```

Meilleur model

```
best_model = cross_val_boost(X,y)
```

```
Best parameters: {'max_depth': 7, 'min_samples_split': 2, 'n_estimators': 200}
Best cross-validation score: 0.9734939759036145
```

Utilise le meilleur Model pour creer les Prédictions (stocké en csv)

```
X_test_clean = X_test.drop(columns=['ID']) # Drop ID pour éviter les conflit de csv
predictions = best_model.predict(X_test_clean)

# Créer un DataFrame avec les IDs et les prédictions
result = pd.DataFrame({
    'ID': X_test['ID'],
    'class': predictions
})
```

```
result
result.to_csv('data/submission_f1.csv')
```

Sauvegarde du modéle en .pkl

```
#Sauvegarde du modèle

import joblib

joblib.dump(best_model,'model/diabeast.pkl')
```

```
['diabeast.pkl']
```