

PROJET MACHINE LEARNING



PYTHON & STREAMLIT

EQUIPE 1

Amandine ANDRE

Andréa LE MAREC

Anthony ALVES

21/02/2024

- Créer une application de Machine Learning :
 - Proposant 2 jeux de données, ou import d'une BDD au format CSV
 - Traitement de la BDD sélectionnée (data management)
 - Choix et entraînement d'un ou plusieurs types de modèles
 - Comparaison des performances des différents modèles
 - Enregistrement d'un modèle final

- I. Objectifs du projet
- II. Environnement technique
- III. Data Management
- IV. Modèles de régression
- V. Modèles de classification
- VI. Mise en application sur Streamlit
- VII. Conclusion et recommandations

I - Objectifs du projet

- Développer nos compétences techniques
 - Machine Learning
 - Python
 - Streamlit
- Développer nos compétences de travail en équipe
 - Communication
 - Répartition des tâches
 - Gestion du temps / Gestion de projet (et bonnes pratiques)
- Se préparer à la soutenance finale et aux questions techniques

■ Logiciel de code :

- Visual Studio Code

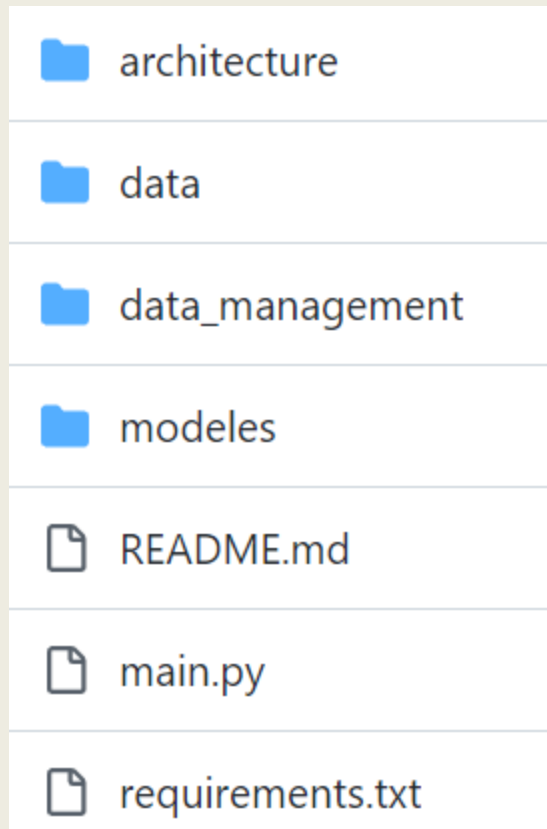
■ Langage :

- Python (version 3.11)

■ Bibliothèques additionnelles :

- Application web : Streamlit (framework)
- Analyse de données : Numpy & Pandas
- Machine Learning : Scikit-learn
- Rééquilibrage : Imbalanced-learn
- Visualisation de données : Matplotlib & Seaborn

■ Architecture de notre projet :



■ 3 fichiers source :

- README.md
- main.py :
- Requirements.txt

■ Un dossier 'architecture' :

- Comporte un fichier python pour chaque onglet de notre application

■ Un dossier 'data' :

- Comporte les deux bases de données proposées dans l'application

■ Un dossier 'data_management' :

- Comporte les fichiers python utiles au traitement/nettoyage des données

■ Un dossier 'modeles' :

- Comporte les fichiers python utiles aux différents modèles de machine Learning

DATA MANAGEMENT

- Rappel des étapes pour un "bon" Machine Learning :
 1. Chargement des données
 2. Data Management (traitement des données, des NaN, feature engineering, ...)
 3. Split du jeu de données
 4. Entraînement du modèle
 5. Validation du modèle

Donc pour avoir un « bon » modèle, il ne faut pas négliger l'étape de Data Management !

- Etapes de data management :
 1. *Visualisation des données*
 2. *Traitement des champs sans nom*
 3. *Traitement des NaN (valeurs manquantes)*
 4. *Préparation pour le modèle - Choix de la 'target'*
 5. *Préparation pour le modèle – Encodage*
 6. *Préparation pour le modèle – Standardisation*
 7. *Préparation pour le modèle – Features*
 8. *Préparation pour le modèle - Rééquilibrage*

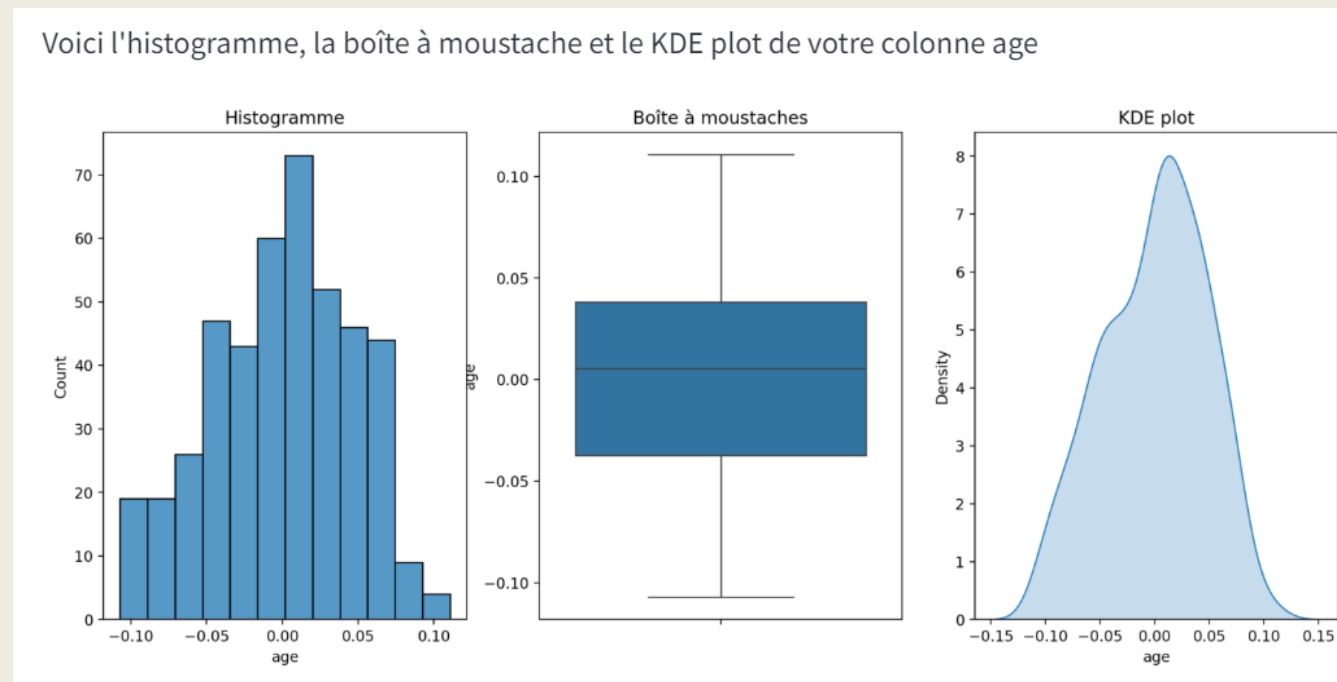
Allons voir sur Streamlit !

■ Première étape : Visualisation des données

- Affichage de quelques statistiques descriptives (tableaux et graphiques)

Intérêt : Voir le bon import des données, et s'il y aura beaucoup de retraitement à faire

Exemple :



■ Deuxième étape : Traitement des champs sans nom

- On a vu dans nos BDD d'exercice (diabète et vin) l'apparition de colonnes sans nom, qui débutaient par "Unnamed..."

Intérêt : Avoir des noms de colonnes qui sont tous compréhensibles

Traitement : Choix laissé à l'utilisateur :

- *Soit colonne(s) sans importance : on supprime*
- *Soit colonne(s) ayant de l'importance : on renomme la colonne (nom de colonne écrit par l'utilisateur)*

■ Troisième étape : Traitement des NaN (valeurs manquantes)

- Nos BDD d'exercice ne contiennent pas de valeurs nulles / manquantes mais nous avons développé cette partie dans le cas d'un import d'une nouvelle BDD

Intérêt : un modèle de Machine Learning ne doit pas comporter de valeurs manquantes

Traitement : Choix laissé à l'utilisateur :

- *Suppression de toutes les lignes (mais affichage d'un message d'alerte et demande de confirmation)*
- *Suppression de la colonne comportant des NaN (avec 2 options : directement sur toutes les colonnes concernées, ou choix de la (des) colonne(s))*
- *Remplacement des NaN : (avec les 2 options)*
 - # par la moyenne (ou médiane) si colonne de type numérique,*
 - # par la valeur la plus fréquente si colonne de type catégorielle*

- Quatrième étape : Préparation pour le modèle - Choix de la 'target'
 - Nos BDD d'exercice (diabète et vin) contiennent déjà un champ 'target'

Mais : Nous pouvons utiliser d'autres BDD qui n'ont pas forcément ce champ identifié

Intérêt : Choisir la target, c'est-à-dire la variable à estimer par le modèle

Solution : On propose à l'utilisateur de choisir une colonne comme "target"

■ Cinquième étape : Préparation pour le modèle - Encodage

- Si une variable catégorielle est intéressante pour le modèle, ou s'il s'agit de la target :

Intérêt : avoir des valeurs numériques pour les modèles (ne prennent que ces variables)

Choix laissé à l'utilisateur:

- *Pour la target : on la recode obligatoirement.*
- *Pour une autre variable : l'utilisateur choisit une variable (ou plusieurs) qu'il veut recoder et on applique l'encodage, en créant une nouvelle variable (nom rentré par l'utilisateur)*

- Sixième étape : Préparation pour le modèle - Standardisation
 - Un jeu de données seulement était standardisé, pas forcément les autres

Intérêt : avoir de meilleures prédictions

Choix laissé à l'utilisateur: Est-ce qu'il veut standardiser son jeu de données ?

Si oui, on applique une standardisation (StandardScaler() de Scikit-Learn)

■ Septième étape : Préparation pour le modèle - Features

- On sélectionne les variables de prédiction

Intérêt : choisir des variables intéressantes et significatives

Choix laissé à l'utilisateur:

- a) *On prend par défaut toutes les colonnes (numériques) de la BDD*
- b) *On sélectionne des colonnes (numériques) manuellement*
- c) *On sélectionne des colonnes (numériques) automatiquement, en fonction de leur corrélation avec la target*

■ Huitième étape : Préparation pour le modèle - Rééquilibrage

- Ce n'est pas réellement le cas dans nos 2 jeux d'exercice, mais on peut avoir des données déséquilibrées

Intérêt : avoir de meilleures prédictions

Choix laissé à l'utilisateur: Il décide s'il veut rééquilibrer le jeu de données. Si oui, on lui propose :

- a) Soit un suréchantillonnage SMOTE
- b) Soit un suréchantillonnage ROS
- c) Soit une attribution de poids de classes `CLASS_WEIGHT` (fonctionnalité en cours de développement)

- Le jeu de données est prêt pour le Machine Learning !



- Le modèle peut maintenant être choisi...

MODÈLES DE RÉGRESSION

Prédiction de valeurs quantitatives

- Préparation des données --> détection du type de *target*
si la *target* est quantitative --> modèles de régression
 - *Trois modèles de régression linéaire proposés :*
 - a) **LinearRegression** (simple régression linéaire, sans pénalité)
 - b) **Ridge** (régularisation L2 --> réduit effet des valeurs aberrantes, améliore la généralisation)
 - si valeurs aberrantes
 - si stabilité du modèle est cruciale
 - c) **Lasso** (régularisation L1 --> sélectionne des variables, sensible aux valeurs aberrantes)
 - si multicollinéarités entre variables
 - si interprétabilité du modèle est cruciale
 - *Futurs développements prévus, ajout des modèles suivants :*
 - a) **ElasticNet** (intermédiaire entre Ridge et Lasso)
 - b) **SVR** (permet de faire des régressions non-linéaires)

- Deux possibilités proposées à l'utilisateur :
 - *Choix d'un modèle parmi ceux proposés (avec paramétrage manuel)*
 - a) *LinearRegression* (aucun paramètre)
 - b) *Ridge* (*alpha*, *max_iter*, *tol*)
 - c) *Lasso* (*alpha*, *max_iter*, *tol*)
 - *Comparaison des modèles sur le JDD sélectionné*
 - a) *Méthode "manuelle"*
 - b) *Méthode utilisant la fonction GridSearchCV* (en cours de développement)

■ Choix d'un modèle de régression

- *Paramétrage de la validation croisée :*
 - *Nombre de "lots" (split) :*
valeur minimum 5 : correspond à 80% données d'entraînement / 20% données de test
- *Paramétrage des modèles (Ridge, Lasso) :*
 - **Alpha** : coefficient de la régularisation (resp. L2, L1)
valeur par défaut : 0.5
 - **Max_iter** : nombre maximum d'itérations autorisées lors de l'optimisation du modèle.
valeur par défaut : 1000
 - **Tol** : contrôle la convergence du modèle
valeur par défaut : 10^{-4}

- Choix d'un modèle de régression
 - *Paramétrage de la validation croisée*
 - *Paramétrage des modèles (Ridge, Lasso)*

Entrez un nombre de lots :

5 – +

Entrez une valeur de alpha (nombre décimal ou entier) :

0.5

Entrez un nombre d'itérations maximum :

1000

Entrez une valeur pour la tolérance :

0.0001

■ Résultats

- *Métriques :*
 - **Coefficient de détermination R^2**
 - **Racine carrée de l'erreur quadratique moyenne RMSE**
 - **Erreur absolue moyenne**
- *Infos complémentaires :*
 - **Tailles des échantillons d'entraînement et de test**

	R2	RMSE	MAE	train_sample_size	test_sample_size
0	0.4235	55.27	46.44	353	89
1	0.4323	59.51	49.62	353	89
2	0.3158	61.02	50.43	354	88
3	0.4951	58.29	50.6	354	88
4	0.3169	61.78	52.54	354	88

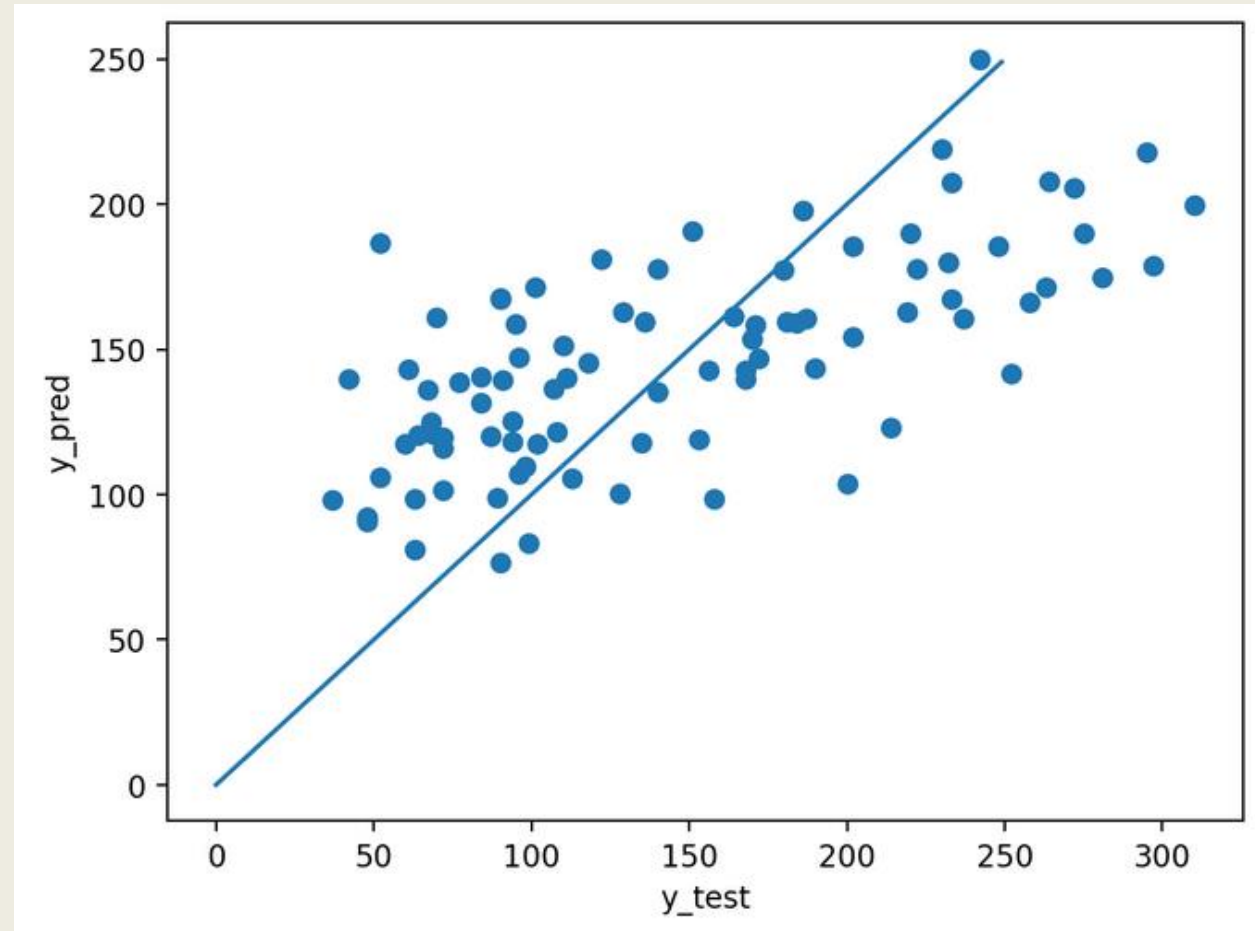
--> *Affichage des métriques pour chaque "lot" de la validation croisée (permet l'optimisation du paramétrage du modèle)*

■ Résultats

○ Graphiques :

*Nuages de points de y_{pred}
(prédiction de la target)
en fonction de y_{test}*

*(Pour chaque "lot" de la
validation croisée)*



■ Possibilité de sauvegarder le modèle entraîné :

Sauvegarde du modèle

--> Souhaitez-vous sauvegarder le modèle ?

☐ Non

☒ Oui

Entrez l'indice du modèle que vous souhaitez enregistrer :

☐ 0

☐ 1

☒ 2

☐ 3

☐ 4

--> Modèle Ridge (alpha : 0.5 ; max_iter : 1000 ; tol : 0.0001), entraînement n° 2 sauvegardé pour réaliser des prédictions.

■ Comparaison des modèles de régression

- *Méthode manuelle*
- *Méthode utilisant la fonction GridSearchCV*

Dans les deux cas :

Combine les paramètres suivants :

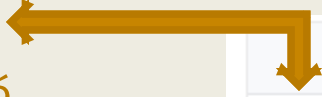
- *Régularisation (alpha) : 0,01 – 0,1 – 1 – 10*
- *Nombre max d'itérations (max_iter) : 100 – 1000 – 10 000*
- *Tolérance pour la convergence (tol) : 10^{-4} – 10^{-3} – 10^{-2}*

Validations croisées : nombre de lots fixé à 5 (80% / 20%)

■ Comparaison des modèles de régression - Méthode manuelle

○ Résultats :

Numéro du
modèle testé



	param	model	R2	RMSE	MAE
1	{}	LinearRegression()	0.3967	59.174	49.926
2	{'alpha': 0.01, 'max_iter': 100, 'tol': 0.0001}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
3	{'alpha': 0.01, 'max_iter': 100, 'tol': 0.001}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
4	{'alpha': 0.01, 'max_iter': 100, 'tol': 0.01}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
5	{'alpha': 0.01, 'max_iter': 1000, 'tol': 0.0001}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
6	{'alpha': 0.01, 'max_iter': 1000, 'tol': 0.001}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
7	{'alpha': 0.01, 'max_iter': 1000, 'tol': 0.01}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
8	{'alpha': 0.01, 'max_iter': 10000, 'tol': 0.0001}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
9	{'alpha': 0.01, 'max_iter': 10000, 'tol': 0.001}	Lasso(alpha=10.0, m	0.3967	59.174	49.926
10	{'alpha': 0.01, 'max_iter': 10000, 'tol': 0.01}	Lasso(alpha=10.0, m	0.3967	59.174	49.926

- Comparaison des modèles de régression - Méthode manuelle
 - *Sélection du meilleur modèle et sauvegarde :*

Meilleur modèle (R^2 le plus élevé) :

	1
param	{}
model	LinearRegression()
R2	0.39672
RMSE	59.174
MAE	49.926

Par défaut, sélection sur le R^2

Futurs développements :

- *choix de la métrique de sélection*
- *choix manuel du modèle parmi la liste de modèles entraînés (numéro du modèle)*

Sauvegarde du modèle

--> Souhaitez-vous sauvegarder le modèle ?

☐ Non

☒ Oui

--> Modèle LinearRegression entraînement n° 1 sauvegardé pour réaliser des prédictions.

MODÈLES DE CLASSIFICATION

Prédiction de valeurs catégorielles

V - Modèles de classification











- Utilisé sur « vin.csv »
 - “Petit” jeu de données
 - Pas de nettoyage à faire

👍 Tips de data exploration :

- skimpy → skimp(df)
- extension Data Wrangler (VS code)

skim(data)

Data Summary		Data Types	
dataframe	Values	Column Type	Count
Number of rows	178	float64	13
Number of columns	15	int32	1
		string	1

column_name	NA	NA %	mean	sd	p0	p25	p50	p75	p100	hist
Unnamed: 0	0	0	88	52	0	44	88	130	180	
alcohol	0	0	13	0.81	11	12	13	14	15	
malic_acid	0	0	2.3	1.1	0.74	1.6	1.9	3.1	5.8	
ash	0	0	2.4	0.27	1.4	2.2	2.4	2.6	3.2	
alcalinity_of_ash	0	0	19	3.3	11	17	20	22	30	
magnesium	0	0	100	14	70	88	98	110	160	
total_phenols	0	0	2.3	0.63	0.98	1.7	2.4	2.8	3.9	
flavanoids	0	0	2	1	0.34	1.2	2.1	2.9	5.1	
nonflavanoid_phenols	0	0	0.36	0.12	0.13	0.27	0.34	0.44	0.66	
proanthocyanins	0	0	1.6	0.57	0.41	1.2	1.6	1.9	3.6	
color_intensity	0	0	5.1	2.3	1.3	3.2	4.7	6.2	13	
hue	0	0	0.96	0.23	0.48	0.78	0.96	1.1	1.7	
od280/od315_of_diluted_wines	0	0	2.6	0.71	1.3	1.9	2.8	3.2	4	
proline	0	0	750	310	280	500	670	980	1700	

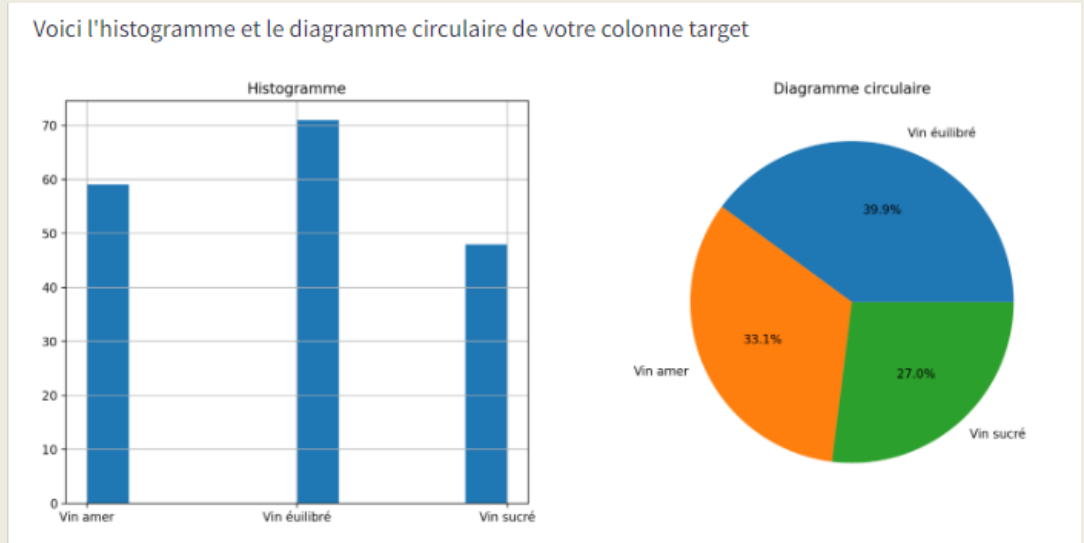
column_name	NA	NA %	words per row	total words
target	0	0	2	356

End

■ Data Management :

- Features : données quantitatives issues d'analyses des vins (*alcohol, flavanoids, tot_phenols, proline, hue, ash, etc.*)
- 'Target' : 3 catégories (=classes) de vins

➤ Léger déséquilibre mais choix de laisser en l'état



■ Data Management (suite) :

- Encodage de 'target' :

➔ Code

```
#Encodage de la target (cas d'une target non numérique)
def encodage(data, colonne: str, new_col: str): # prend le nom de la colonne
    i = 0
    labels = {}
    for _, val in enumerate(data[colonne].unique()):
        labels.update({val : i})
        i +=1
    data[new_col] = data[colonne].map(labels)
    st.write("La colonne", colonne, "a bien été encodée. Voici le résultat")
    new_col_res = data[new_col].value_counts()
    col_encod = data[new_col]
    return col_encod, new_col_res
```

- Recherche des corrélations et des colinéarités
➔ Heatmap + sélection des features : 'proline' et 'flavanoids' à écarter
- Possibilité de standardiser les valeurs et rééquilibrer la BDD

■ Modélisation :

- Train_test_split sur 25 % du jeu de données
- Différents modèles testés (dans un multiselect) :
 - ➔ LogisticRegression
 - ➔ DecisionTreeClassifier
 - ➔ RandomForestClassifier
 - ➔ KNeighborsClassifier
 - ➔ SVC
- Métriques de performance avec 'classification_report' :
 - accuracy, f1-score

■ Modélisation (suite) :

- Nombreux paramètres dans ces modèles :
 - ➔ Certains à fixer par défaut pour éviter des erreurs (taille du dataset ?)
 - ➔ D'autres laissés au choix de l'utilisateur
- Validation croisée avec 'StratifiedKFold'
 - ➔ 1 seule métrique possible : on garde accuracy
 - ➔ Comparaison des résultats :

Modèle	Scores moyens	
Logistic Regression	0,967	0,971
Decision Tree Clf	0,899	0,925
Random Forest Clf	0,977	0,983
SVC	0,955	0,962
KNeighbors Clf	/	/

IMPLEMENTATION SUR STREAMLIT

Démonstration !





■ Bilan global :

- Très formateur sur les aspects programmation Python et Machine Learning
- Utilisation du framework Streamlit intéressant mais chronophage
- Bonne mise en pratique des notions vues (Data Management, préparation du modèle et métriques de performances)
- Répartition des tâches et travail collaboratif tout au long du projet

■ Recommandations :

- Utilisation de jeux de données plus volumineux et/ou possibilité de validation du modèle avec un jeu de données dédié
- Développement de fonctionnalités supplémentaires si plus de temps