## **Étapes pour réaliser un projet de Machine Learning**

### **1. Collecte des données**

Si fichier csv :

df = pd.read\_csv()

### **2. Pré-traitement des données**

Comprendre et évaluer la qualité des données, sans encore les modifier.

**Objectif** : Identifier les problèmes à corriger lors de l'étape suivante.

Examiner les informations et statistiques du dataframe.

df.head(), df.tail(), df.sample()

df.shape

df.columns

df.info(), df.dtypes, df.describe()

Vérifier les doublons et les valeurs manquantes.

df.isna().sum()

df.duplicated().sum()

Analyser les distributions avec des histogrammes ou des graphiques.

df.boxplot() → outliers

df.corr(numeric\_only = True)

Repérer les corrélations avec des heatmaps.

sns.heatmap(df.corr(numeric\_only = True)) → plt.show()

### **3. Traitement des données**

* **But** : Modifier les données pour qu'elles soient prêtes à être utilisées.

Gérer les valeurs manquantes : les remplir (imputation) ou les supprimer.

df.fillna(), df.dropna(inplace = True)

*Le client choisit comment gérer les valeurs manquantes (par exemple remplir ou supprimer une colonne) ou les outliers car il connaît les valeurs importantes, le data scientist ne connaît pas les valeurs métiers.*

Supprimer les doublons.

df.drop\_duplicates(keep = ‘first’)

Transformer les variables catégoriques en numériques

pd.get\_dummies(df, columns=["nom\_de\_la\_colonne"])

pd.factorize()

### **4. Feature Engineering (Création et sélection de variables)**

* **But** : Identifier ou créer les variables explicatives les plus utiles pour le modèle.

Combiner des colonnes, créer des colonnes.

Supprimer les colonnes inutiles ou peu corrélées au problème.

df.drop(columns = [‘nom\_colonne\_à\_drop’], inplace = True)

Réduire les dimensions si nécessaire (ex. : PCA).

Définir les variables explicatives

var\_exp = [‘col1’,’col2’,’col3’]

var\_y = [‘col4’]

X = df[var\_exp]

y = df[var\_y]

### 

### 

### 

### **5. Définir le jeu d’entraînement et de test**

* **But** : Diviser les données pour évaluer la performance du modèle.

La stratégie du train\_test\_split() n’est pas obligatoire. Pas de train\_test\_split() quand il y a une série temporelle.

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size = 0.75, random\_state = 42)

Normaliser ou standardiser si nécessaire (on ne standardise pas les données catégorielles transformées avec fatorize() ou get\_dummies()).

scaler = StandardScaler().fit(X\_train) → on initialise le scaler et il apprend les paramètres des données d’entraînement

X\_train\_scaled = scaler.transform(X\_train) → on applique les paramètres appris (avec fit) pour transformer les données en les standardisant.

X\_test\_scaled = scaler.transform(X\_test)

### **6. Choisir l’algorithme ML (les étapes 5 et 6 sont interchangeables)**

* **But** : Identifier quel type de modèle est adapté au problème (régression, classification, clustering)

Pour une régression, les variables doivent être quantitatives.

Algorithmes de régression : Régression linéaire simple, régression linéaire multivariée, decision tree

Algorithmes de classification : KNN, Régression logistique, decision tree

### **7. Entraîner le modèle**

* **But** : Apprendre au modèle à trouver des motifs dans les données.

model = LinearRegression() → on initialise le modèle

model.fit(X,, y) → on entraîne le modèle

ou en cas de division et/ou de scaling :

si train\_test\_split a été utilisé :

model.fit(X\_train, y\_train)

si un scaler a été utilisé :

model.fit(X\_train\_scaled, y\_train)

### **8. Évaluer le modèle**

* **But** : Vérifier si le modèle fonctionne bien.

Vérifier le score de test

test\_score = model.score(X\_test\_scaled, y\_test)

Comparer le score de test avec le score d’entraînement (évaluer overfitting, underfiting ou robustesse)

train\_score = model.score(X\_train\_scaled, y\_train)

test\_score = model.score(X\_test\_scaled, y\_test)

*Si le score d'entraînement est très élevé et le score de test est faible → Overfitting*

*Si les deux scores sont faibles → Underfitting*

*Si les deux scores sont proches et suffisamment élevés → Modèle robuste.*

Utiliser des métriques d’évaluation

y\_pred = model.predict(X\_test\_scaled) → on fait des prédictions sur le jeu de test

accuracy = accuracy\_score(y\_test, y\_pred)

precision = precision\_score(y\_test, y\_pred)

recall = recall\_score(y\_test, y\_pred)

f1 = f1\_score(y\_test, y\_pred)

### **9. Amélioration du modèle (Fine-tuning)**

* **But** : Améliorer les performances du modèle.
* **Techniques possibles** :
  + Ajuster les hyperparamètres (ex. : nombre d’arbres dans un Random Forest).
  + Essayer d'autres algorithmes.
  + Ajouter ou retirer des variables.
  + Appliquer des méthodes comme la validation croisée pour affiner l’entraînement.

### **10. Prédictions**

* Une fois le modèle satisfaisant, on peut l’utiliser pour faire des prédictions sur de nouvelles données !

**LIBRAIRIES :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LinearRegression, LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, r2\_score

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import cross\_val\_score, GridSearchCV

from sklearn.decomposition import PCA

Pour étendre le nombre de lignes complètes à afficher ou le nombre de colonnes :

pd. set\_option(‘display.min\_rows’, *nbre de lignes*)

pd.set\_option(‘display.max\_columns’, *nbre de colonnes*)