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
- 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)
```