Ce projet est un problème de classification binaire, où mon objectif est de prédire si une transaction est frauduleuse (1) ou non (0). Voici comment je structure ce projet :

1. Prétraitement des données

Avant de commencer à modéliser, je m'assure de bien préparer mes données :

a. Nettoyage des données

- Je vérifie les valeurs manquantes et décide si je les remplace (par la moyenne, la médiane, ou une valeur spécifique) ou si je les supprime.
- Je vérifie aussi les doublons et les incohérences dans les données.

b. Encodage des variables catégorielles

Certaines colonnes comme `Nom_marchand`, `Sexe_titulaire`, ou `Nom_titulaire` sont des variables catégorielles. Je les transforme en variables numériques pour qu'elles puissent être utilisées par mon modèle :

- J'utilise un encodage comme `OneHotEncoding` ou `LabelEncoding` selon le type de variable. Par exemple, `Sexe_titulaire` peut être transformé avec un encodage binaire.
- Pour des variables comme `Nom_marchand` ou `Categorie_marchand`, je préfère utiliser un `OneHotEncoder` si le nombre de catégories n'est pas trop élevé.

c. Transformation des variables temporelles

Les variables comme `DateHeure_transaction` et `Heure_UNIX_transaction` peuvent être transformées pour extraire des informations utiles :

- Je peux extraire le jour de la semaine (est-ce un weekend ou un jour de semaine ?).
- Je vérifie l'heure de la journée (les fraudes se produisent-elles plus fréquemment à certaines heures ?).
- J'utilise aussi le mois ou l'année pour repérer des tendances saisonnières.

J'utilise `pd.to_datetime` pour convertir `DateHeure_transaction` en un format datetime.

d. Normalisation des variables numériques

Certaines de mes colonnes comme `Montant_transaction`, `Latitude_marchand`, etc., sont numériques. Pour des modèles comme les SVM ou les réseaux de neurones, je normalise ces variables pour les amener sur une échelle similaire :

- J'utilise `StandardScaler` ou `MinMaxScaler` pour normaliser ces variables.

e. Vérification de la balance des classes

Comme c'est un problème de classification binaire, je vérifie si mes classes sont équilibrées (i.e., si j'ai autant de transactions frauduleuses que non frauduleuses). Si ce n'est pas le cas, je peux utiliser des techniques comme le sous-échantillonnage (undersampling) ou le sur-échantillonnage (oversampling) pour rééquilibrer les classes.

2. Séparation des ensembles de données

Je divise mes données en trois ensembles :

- **Ensemble d'entraînement** : pour entraîner mon modèle.
- **Ensemble de validation** : pour ajuster les hyperparamètres de mon modèle.
- **Ensemble de test** : pour évaluer la performance du modèle final.

J'utilise souvent `train_test_split` de `sklearn` avec un pourcentage de 70%-80% pour l'entraînement et 20%-30% pour le test.

3. Choix du modèle

Pour une classification binaire, je commence par essayer plusieurs modèles et je compare leur performance :

- **Logistic Regression** : un bon point de départ pour comprendre la relation entre mes variables et la cible.
- **Random Forest / XGBoost / LightGBM** : ce sont des modèles puissants pour la classification et ils gèrent bien les interactions complexes entre les variables.
- **Support Vector Machine (SVM)** : utile si mes données sont complexes, mais je dois faire attention à la mise à l'échelle des données.
- **Réseaux de neurones (MLP)** : pour explorer des modèles plus complexes et des relations non linéaires.

4. Évaluation du modèle

Pour évaluer mes modèles, j'utilise plusieurs métriques. Comme c'est un problème de classification binaire, voici celles qui me semblent les plus pertinentes :

- **Precision / Recall / F1-score** : pour mieux comprendre les performances du modèle pour prédire les fraudes.
- **AUC-ROC** : pour visualiser la capacité du modèle à distinguer les classes positives (fraude) et négatives.

5. Optimisation des hyperparamètres

Une fois que j'ai un modèle de base qui fonctionne, je peux essayer d'optimiser ses hyperparamètres :

- J'utilise des méthodes comme la recherche par grille (`GridSearchCV`) ou la recherche aléatoire (`RandomizedSearchCV`) pour explorer différentes combinaisons d'hyperparamètres.

6. Interprétation du modèle

Pour un problème de fraude, il est important de comprendre pourquoi mon modèle prend certaines décisions. Je peux utiliser des techniques comme :

- **Feature Importance** : pour identifier les variables qui influencent le plus mon modèle (par exemple, `Montant_transaction` pourrait être un facteur clé).
- **SHAP (Shapley Additive Explanations)** : pour une explication plus détaillée des prédictions du modèle.

7. Mise en production

Quand mon modèle est prêt et que j'ai obtenu des résultats satisfaisants, je le déploie dans un environnement de production pour prédire les transactions en temps réel. Cela peut inclure la mise en place d'une API pour intégrer le modèle dans un système de gestion des transactions.

Résumé des étapes :

1. Prétraitement des données : nettoyage, encodage des variables, transformation des variables temporelles, normalisation.
2. Séparation des données en ensembles d'entraînement, validation et test.
3. Choix du modèle (logistic regression, random forest, XGBoost, SVM, MLP).
4. Évaluation du modèle avec des métriques adaptées (accuracy, precision, recall, F1-score, AUC-ROC).
5. Optimisation des hyperparamètres.
6. Interprétation du modèle.
7. Mise en production si nécessaire.