1️⃣ Type de modèle

Le modèle est un **modèle de régression supervisée** pour prédire des valeurs numériques ou catégorielles dans le contexte de la gestion de stock et des livraisons.

Points clés :

- **Objectif** : prédire la demande future ou la quantité de produit à préparer/livrer.
- **Entrée (features)** : les données de ta base :
  - `Produit` (café/cacao)
  - `Lot` (date, quantité)
  - `Emplacement` (magasin/étagère)
  - `Historique des ventes/livraisons`
  - `Utilisateur` (grossiste)
- **Sortie (target)** : quantité estimée à préparer/livrer ou niveau de stock futur.

> Exemple : prédire que demain, pour le produit **Café Robusta**, il faudra préparer **120 kg**, basé sur l'historique.

2️⃣ Format des fichiers et emplacement

Pour intégrer ce modèle à ton projet Django :

```
mokpokpo_supply/
├── core/
│   ├── models.py         # Base de données
│   ├── views.py          # Vues qui appelleront le modèle
│   ├── forms.py
│   ├── ai/
│   │   ├── model.pkl     # modèle entraîné (pickle)
```

```
|   |   └── ai_utils.py   # fonctions pour préparer les données et
prédire
|   ├── templates/
|   └── static/
├── manage.py
└── requirements.txt
```

- `model.pkl` → modèle entraîné sauvegardé avec **pickle** (ou `joblib`)
- `ai_utils.py` → fonctions pour :
    - Charger le modèle
    - Préparer les données de la base
    - Faire les prédictions

3️⃣ Données utilisées pour l'entraînement

Pour ce modèle, on utilise **l'historique réel ou simulé** des lots et des livraisons :

Feature	Type	Description
produit	Catégoriel	Café ou cacao
date_production	Date	Date du lot
quantite_initiale	Numérique	Quantité de lot entrée en stock
quantite_restante	Numérique	Quantité restante après mouvements/livraisons
emplacement	Catégoriel	Où est stocké le lot
statut_livraison	Catégoriel	Préparée / En cours / Livrée
utilisateur (grossiste)	Catégoriel	Pour identifier les commandes

> Le modèle transforme les catégories en **variables numériques** (one-hot encoding) pour pouvoir les utiliser.

4️⃣ Type de données et pré-traitement

1. **Catégories → numériques**
    - Produit, emplacement, statut, utilisateur
    - Utilisation de `pandas.get_dummies` ou `sklearn.OneHotEncoder`
2. **Dates → features numériques**
    - `date_production` peut être transformée en `jour_semaine`, `mois`, `jour_année`
    - Permet de capturer les **variations saisonnières**
3. **Normalisation / standardisation**
    - Pour les quantités, on peut utiliser `StandardScaler`
    - Optionnel selon le modèle choisi (ex : arbres décisionnels n'en ont pas besoin)

5️⃣ Type d'algorithme utilisé

- **RandomForestRegressor** ou **GradientBoostingRegressor** (de `sklearn`)
- Pourquoi ?
    - Gère bien les données tabulaires mixtes (numérique + catégoriel)
    - Résistant au sur-apprentissage si assez de données
    - Donne une interprétation via l'importance des features

Exemple de code pour entraîner :

```
from sklearn.ensemble import RandomForestRegressor
```

```
from sklearn.model_selection import train_test_split
import pandas as pd
import joblib

# Charger les données depuis la base ou CSV simulé
df = pd.read_csv("historique_lots.csv")

# Préparer features et target
X = pd.get_dummies(df[['produit','emplacement','jour_semaine']])
y = df['quantite_livraison']

# Séparer train/test
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)

# Créer le modèle
model = RandomForestRegressor(n_estimators=200, random_state=42)
model.fit(X_train, y_train)

# Sauvegarder
joblib.dump(model, 'core/ai/model.pkl')
```

6️⃣ Comment l’utiliser dans Django

Dans `core/ai/ai_utils.py` :

```
import joblib
import pandas as pd
from core.models import Lot, Produit, Emplacement

# Charger le modèle
model = joblib.load('core/ai/model.pkl')

def prepare_features(lot_id):
    lot = Lot.objects.get(id=lot_id)
    df = pd.DataFrame([{
        'produit': lot.produit.nom,
```

```
        'emplacement': lot.emplacement.code_emplacement,
        'jour_semaine': lot.date_production.weekday()
    }])
    return pd.get_dummies(df)

def predict_lot(lot_id):
    X = prepare_features(lot_id)
    return model.predict(X)[0]
```

Dans `views.py` :

```
from core.ai.ai_utils import predict_lot

@login_required
def dashboard_gerant(request):
    lots = Lot.objects.all()
    predictions = {lot.id: predict_lot(lot.id) for lot in lots}
    return render(request, 'dashboard_gerant.html', {
        'lots': lots,
        'predictions': predictions
    })
```

> Tu peux ensuite afficher la prédiction dans le tableau des lots du gérant.

7️⃣ Comment le modèle a été entraîné

- **Supervisé** : on a des exemples `entrée → sortie`
- **Historique réel ou simulé** des lots/livraisons
- **Algorithme** : RandomForest ou GradientBoosting
- **Sauvegarde** avec `joblib` pour réutilisation dans Django
- **Prédiction** : `model.predict(X)` où `X` est un DataFrame préparé avec les mêmes colonnes que l'entraînement

---

8️⃣ Limitations et conseils

- Si le nombre de lots ou grossistes change souvent → il faut **réentraîner régulièrement** le modèle
- Les données doivent être propres et cohérentes
- Pour les catégories non vues lors de l'entraînement → transformer en "inconnues" ou utiliser `OneHotEncoder(handle_unknown='ignore')`
- Pour de la vraie optimisation de routes → tu peux combiner ce modèle avec l'API Google Maps Distance Matrix