

Rapport Final – Projet IA : Prédiction de la Satisfaction Client & Déploiement Cloud

Ce projet complet de Machine Learning vise à prédire la satisfaction client d'un site e-commerce, de l'analyse des données au déploiement d'une API sur Google Cloud Run. L'objectif est d'améliorer la qualité de service et d'anticiper les risques d'insatisfaction.

Comprendre l'Insatisfaction Client : Un Enjeu Stratégique

Les entreprises d'e-commerce doivent constamment affiner l'expérience client. Une faible satisfaction peut engendrer des conséquences négatives:

- Augmentation des retours produits
- Baisse de la fidélité client
- Dégradation de la réputation en ligne

Notre défi est de **prédir automatiquement** la satisfaction client à partir de ses données d'achat et de livraison.



Dataset Olist : La Source de Nos Prédictions

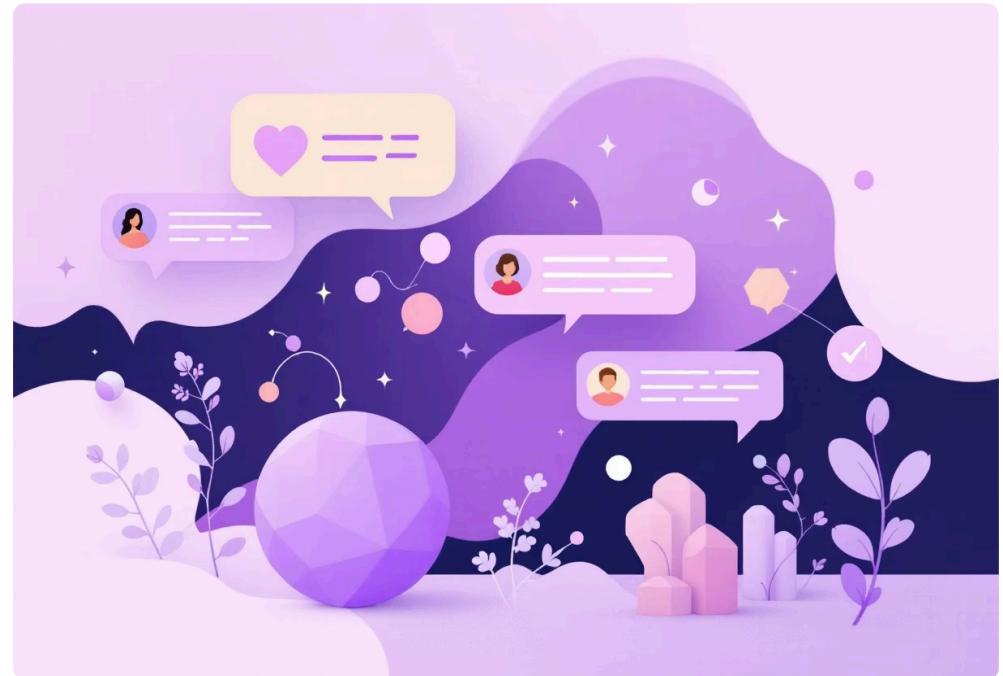
Le dataset provient de la plateforme brésilienne Olist, une mine d'informations sur les commandes, livraisons, paiements et avis clients.

Variables d'Entrée

- delai_livraison
- retard_livraison
- nb_versements
- valeur_paiement
- prix_total
- nb_articles
- longueur_commentaire
- product_category_encoded

Variable Cible

- satisfaction (0 = insatisfait, 1 = satisfait)



Préparation des Données : La Fondation du Modèle

01

Nettoyage Approfondi

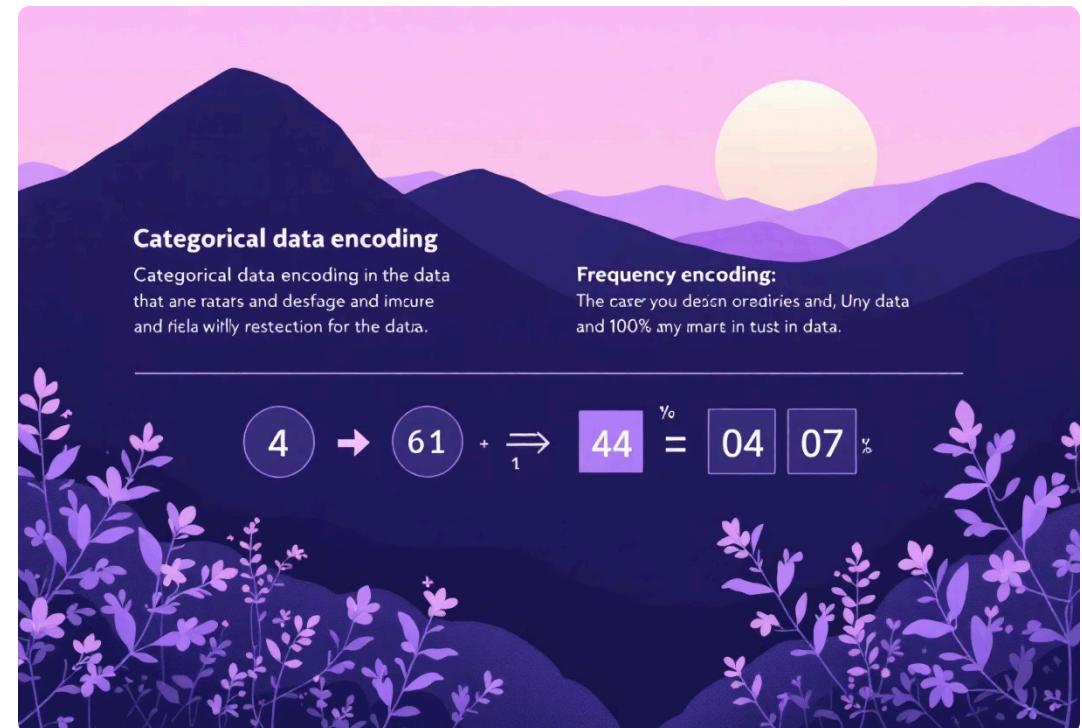
Nous avons vérifié les doublons, géré les valeurs manquantes et supprimé les colonnes non pertinentes.



02

Encodage des Catégories

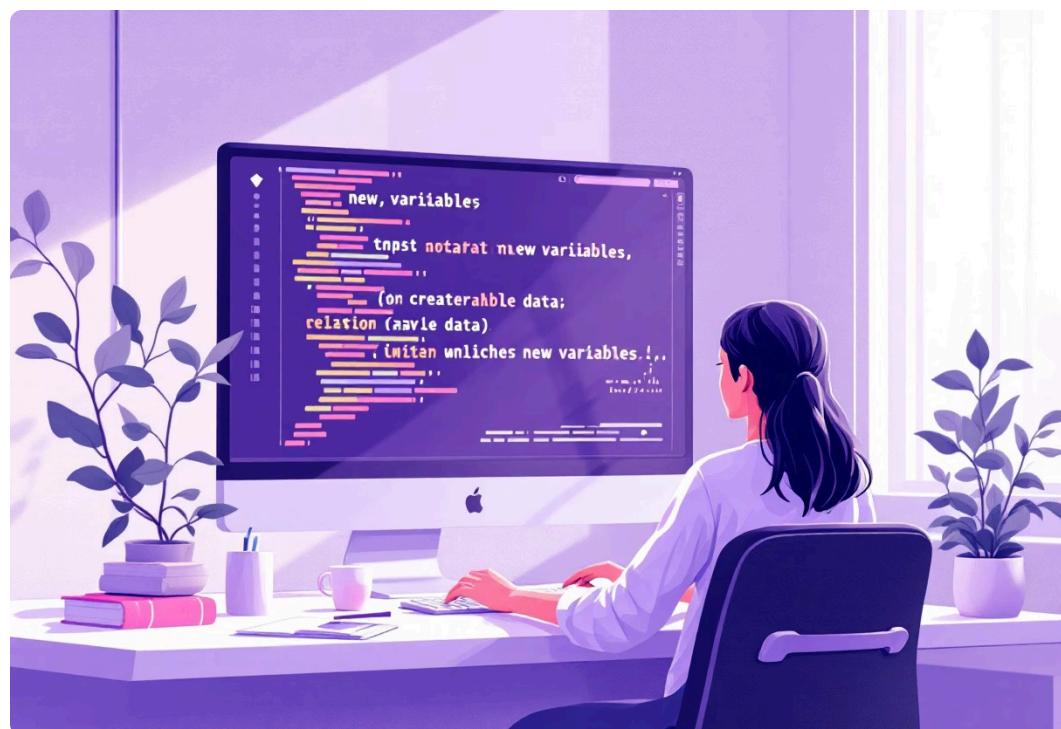
La colonne `product_category_name` (plus de 70 valeurs) a été encodée par fréquence pour une meilleure représentativité.



03

Feature Engineering

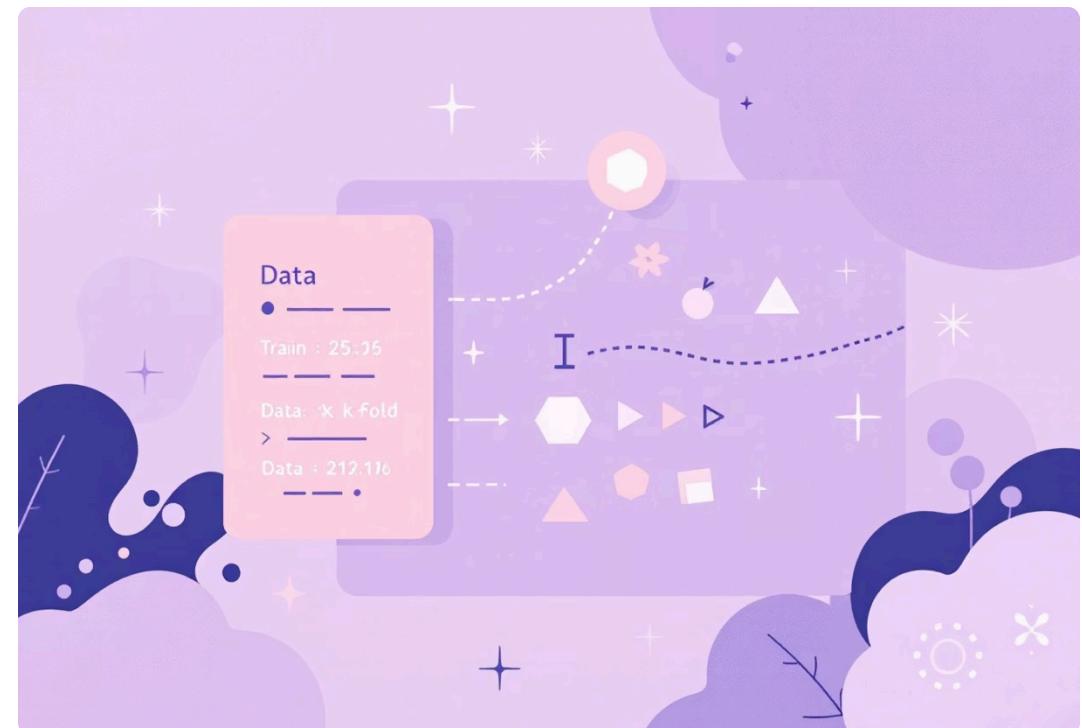
Création de nouvelles variables (délais, regroupements) pour enrichir le modèle et capturer des relations complexes.



04

Division et Validation

Les données ont été divisées en 80% pour l'entraînement et 20% pour le test, avec une validation croisée Stratified K-Fold pour gérer le déséquilibre des classes.



Modélisation et Sélection du Meilleur Modèle

Plusieurs algorithmes ont été évalués pour identifier le plus performant en matière de prédiction de satisfaction client.

KNN

F1-score: **0.881** \pm 0.001



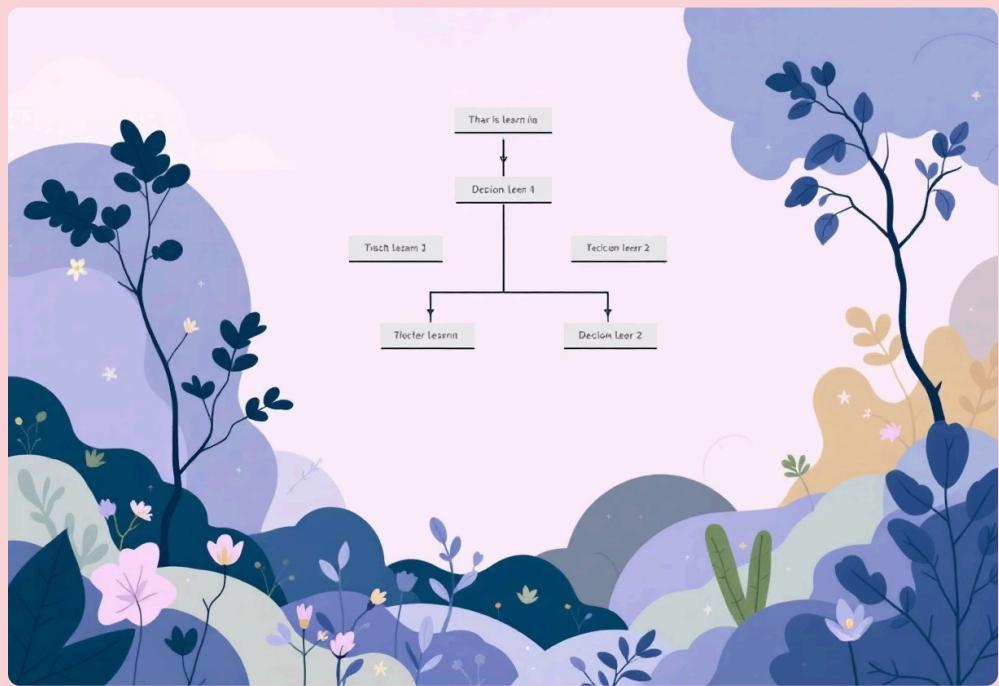
Régression Logistique

F1-score: **0.887** \pm 0.001



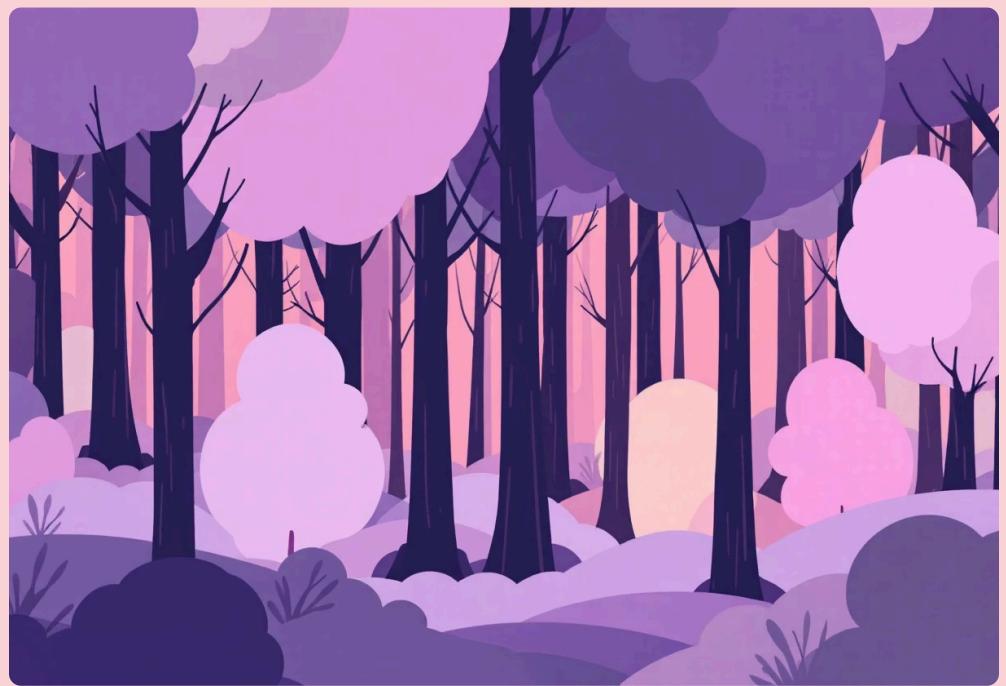
Arbre de Décision

F1-score: **0.839** \pm 0.004



Random Forest

F1-score: **0.902** \pm 0.001

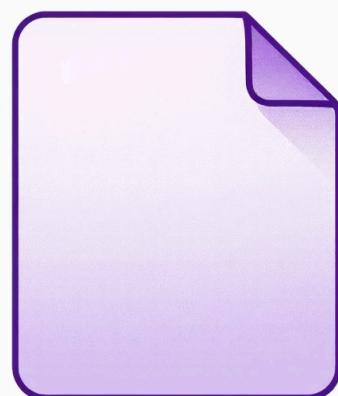
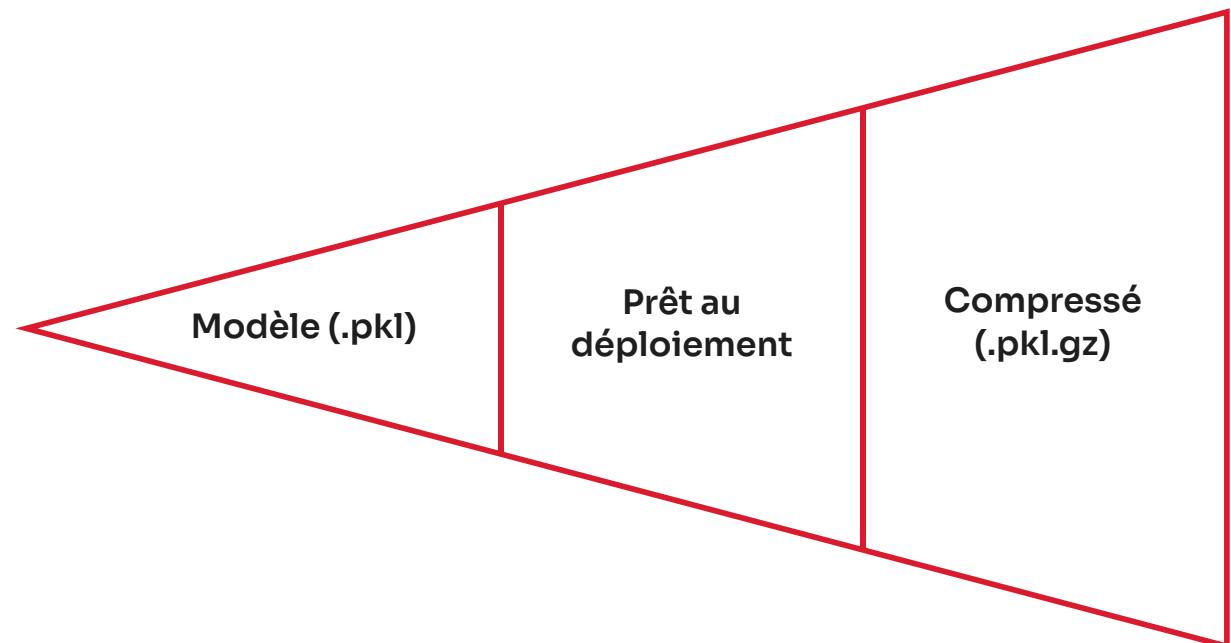


Le modèle **Random Forest** a été retenu pour son F1-score supérieur, sa faible variance et sa stabilité en validation croisée, tout en offrant une bonne interprétabilité des variables.

Préparation pour la Production : Exportation du Modèle

Pour faciliter son déploiement et son intégration, le modèle Random Forest a été exporté et optimisé.

- Sauvegarde sous format `.pkl`: Le modèle final est enregistré sous le nom `best_random_forest_model.pkl`.
- Compression: Le fichier `.pkl` a été compressé en `.pkl.gz` pour réduire sa taille, optimisant ainsi le conteneur Docker.



Développement de l'API : FastAPI pour l'Accessibilité

Une API robuste a été développée avec FastAPI pour interagir facilement avec le modèle de prédiction.

/health

Cette route permet de vérifier le statut de l'API et la bonne charge du modèle. Essentielle pour le monitoring et la maintenance.

/predict

Accepte un objet JSON contenant les caractéristiques du client et retourne la prédiction de satisfaction (0 ou 1) ainsi que les probabilités associées.

Exemple de requête JSON pour /predict:

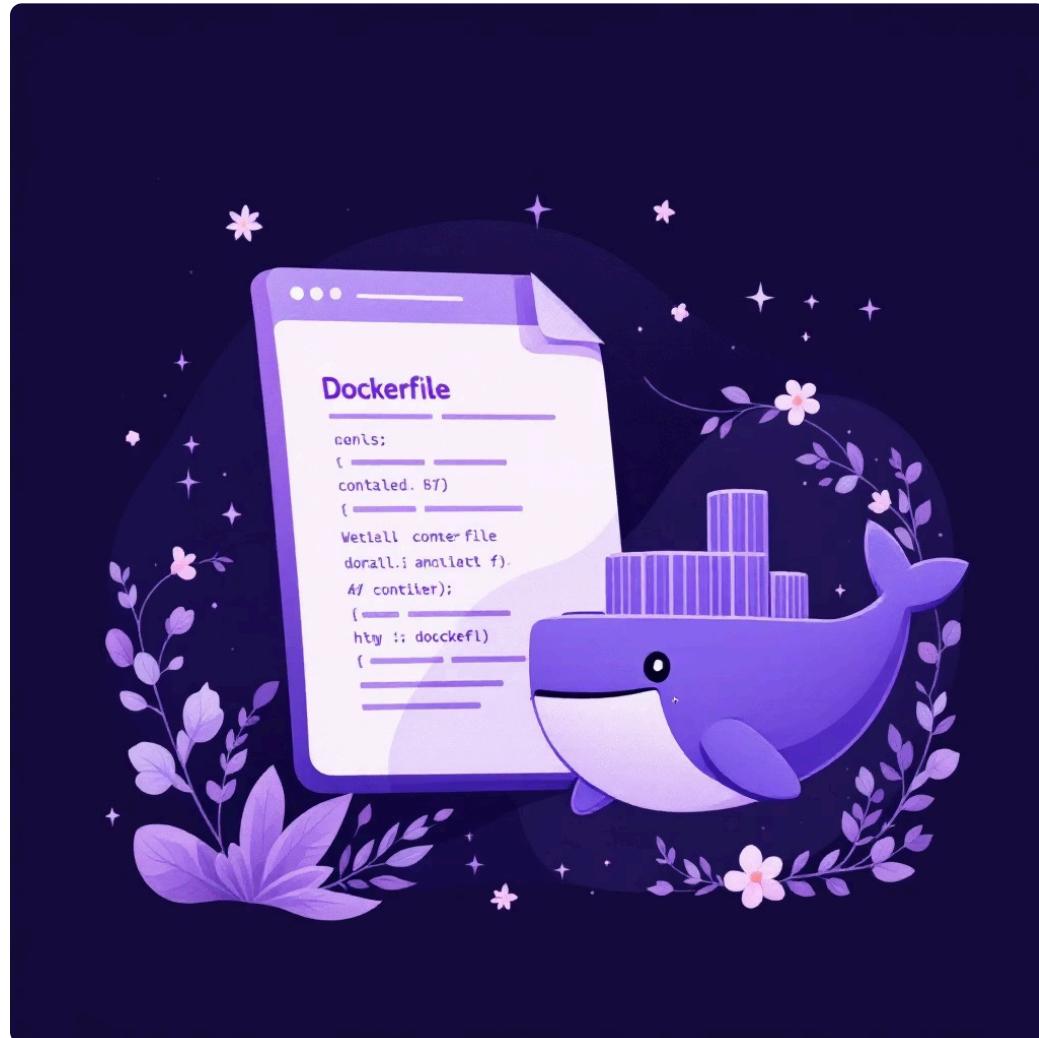
```
{  
    "delai_livraison": 2.0,  
    "retard_livraison": 0.0,  
    "nb_versements": 1,  
    "valeur_paiement": 100,  
    "prix_total": 150,  
    "nb_articles": 1,  
    "longueur_commentaire": 20,  
    "product_category_encoded": 3  
}
```

Dockerisation : Conteneuriser pour la Portabilité

La conteneurisation avec Docker assure un environnement reproductible et isolé pour notre API.

Dockerfile

- Installe Python et ses dépendances.
- Copie l'API et le modèle compressé.
- Lance Uvicorn sur le port dynamique \${PORT} requis par Cloud Run.



.dockerignore

Ce fichier crucial exclut les éléments non nécessaires du contexte de build, réduisant la taille de l'image Docker:

- Datasets bruts et intermédiaires
- Notebooks de développement
- Fichiers lourds non essentiels à l'API
- Dossiers `__pycache__`



Déploiement Serverless sur Google Cloud Run

Cloud Run offre une plateforme entièrement gérée pour déployer des applications conteneurisées à grande échelle.



Création du Service

Un service Cloud Run est configuré pour recevoir des requêtes HTTP.



Mise en Ligne de l'API

L'image Docker de notre API est déployée, rendant l'application accessible sur internet.



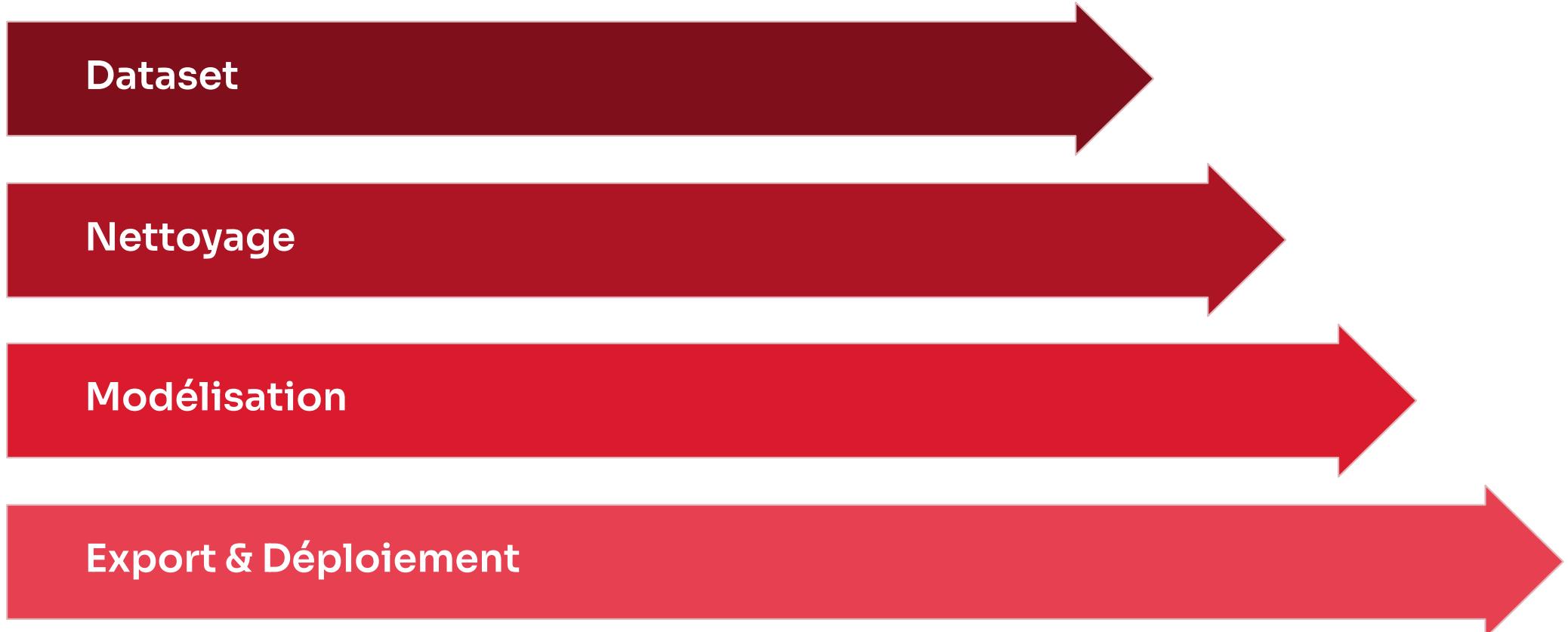
Permissions Publiques

Les permissions sont ajustées pour permettre un accès public à l'API sans authentification.

API Publique : <https://shopnow-api-223238462168.us-central1.run.app/>

Le Cycle Complet de l'IA en Production

Ce projet illustre une architecture moderne et scalable, de l'idée au déploiement en production.



Nous avons démontré la capacité à construire une solution IA performante, accessible en ligne et prête pour une intégration dans des applications réelles.

Problèmes Rencontrés & Solutions

Le chemin vers le déploiement en production est souvent jalonné d'obstacles. Voici les défis majeurs que nous avons surmontés et les solutions mises en œuvre :

1

Conteneur ne démarrait pas

Cause : L'API écoutait sur le port 8000.

Solution : Utilisation de la variable d'environnement `${PORT}` imposée par Cloud Run pour le port d'écoute.

2

Erreur "Method Not Allowed"

Cause : Utilisation incorrecte de la méthode HTTP (GET au lieu de POST).

Solution : Test correct de l'API avec la méthode POST appropriée.

3

Image Docker trop lourde

Cause : Inclusions inutiles (datasets, notebooks) dans le contexte de build.

Solution : Configuration précise du fichier `.dockerignore` pour exclure les éléments non essentiels.

4

Artifact Registry introuvable

Cause : Services Google Cloud non activés ou dépôt mal configuré.

Solution : Activation des services nécessaires et recréation du dépôt Artifact Registry.

5

Modèle trop volumineux

Cause : Fichier modèle sérialisé (`.pkl`) occupant trop d'espace.

Solution : Compression du modèle au format `.pkl.gz` pour réduire la taille.