**VII. Déploiement avec Streamlit**

Pour rendre notre modèle de prédiction de la dépression accessible et interactif, nous avons choisi de développer une application web simple en utilisant la bibliothèque **Streamlit**.

1. Objectif du Déploiement avec Streamlit

L'objectif principal était de :

* Fournir une interface utilisateur pour entrer les données d'un étudiant.
* Permettre d'obtenir une prédiction instantanée (dépression ou non-dépression) ainsi que la probabilité associée.
* Illustrer concrètement l'application pratique de notre modèle de Machine Learning.

2. Choix du Modèle pour le Déploiement Initial

Après avoir entraîné et évalué plusieurs modèles (Régression Logistique, K-Nearest Neighbors, Random Forest), nous avons initialement choisi de déployer le modèle de **Régression Logistique**.

* **Raison :** Ce modèle a montré d'excellentes performances, notamment un score ROC AUC de **0.9232** et un F1-score de **0.8742**. De plus, sa nature plus simple par rapport au Random Forest (non optimisé) le rendait idéal pour une première itération de déploiement, nous permettant de nous concentrer sur la mise en place du pipeline de déploiement.

3. Préparation pour le Déploiement : Sauvegarde des Artefacts

Avant de créer l'application Streamlit, il était crucial de sauvegarder tous les composants nécessaires pour reproduire les prédictions de manière cohérente avec l'entraînement :

* **Modèle Entraîné :** Le modèle de Régression Logistique (logistic\_regression\_dep\_model.joblib) a été sauvegardé en utilisant la bibliothèque joblib.
* **Préprocesseurs :**
  + L'objet StandardScaler (scaler\_dep.joblib), qui a été "fitté" sur les données d'entraînement numériques, a été sauvegardé pour assurer que les nouvelles données soient mises à l'échelle de la même manière.
  + Les **mappings pour le Label Encoding** (label\_mappings\_dep.joblib) des variables catégorielles binaires (Gender, Suicidal Thoughts, Family History) ont été conservés.
  + La **liste des colonnes finales attendues par le modèle** après tout le prétraitement (final\_training\_columns. joblib) a été sauvegardée pour garantir la cohérence de la structure des données d'entrée.
  + Les **noms des colonnes originales qui ont subi le One-Hot Encoding** (après regroupement, ex: City\_Processed, Dietary Habits, Degree\_Grouped) ont été stockés (original\_cols\_for\_dummies. joblib).
  + La **liste des colonnes numériques à mettre à l'échelle** (cols\_to\_scale.joblib) a été sauvegardée.
  + Le **mapping pour la transformation de Sleep Duration** en valeurs numériques (sleep\_mapping.joblib) a été conservé.
  + La liste des top\_30\_cities (top\_30\_cities.joblib) utilisée pour le regroupement de la feature 'City' a également été sauvegardée pour une reproduction fidèle du prétraitement.
  + **Raison :** La sauvegarde de ces artefacts est essentielle car le modèle ne peut faire des prédictions correctes que si les nouvelles données d'entrée subissent exactement les mêmes transformations (encodage, regroupement, mise à l'échelle, création de features dummy) que les données sur lesquelles il a été entraîné.

4. Création de l'Application Streamlit (streamlit\_app.py)

L'application a été développée dans un unique fichier Python :

* **Chargement des Ressources :** Au démarrage, l'application charge le modèle et tous les préprocesseurs sauvegardés. La fonction @st.cache\_resource de Streamlit a été utilisée pour mettre ces objets en cache et améliorer les performances de l'application en évitant les rechargements inutiles.
* **Fonctions de Prétraitement :**
  + La fonction group\_degree (identique à celle du notebook) a été intégrée pour regrouper les diplômes.
  + Une fonction preprocess\_input\_streamlit a été créée. Son rôle est de prendre les données brutes entrées par l'utilisateur (sous forme de dictionnaire), de les convertir en DataFrame Pandas, puis d'appliquer séquentiellement toutes les étapes de prétraitement :
    1. Transformation de Sleep Duration.
    2. Application des mappings de Label Encoding.
    3. Création de la feature Is\_Student à partir de Profession.
    4. Regroupement des City en City\_Processed (en utilisant top\_30\_cities si disponible).
    5. Regroupement des Degree en Degree\_Grouped.
    6. Application du One-Hot Encoding via pd.get\_dummies sur les colonnes appropriées.
    7. Mise à l'échelle des features numériques avec l'objet scaler chargé.
    8. Réindexation duDataFrame avec df.reindex(columns=final\_training\_columns, fill\_value=0) pour garantir que le DataFrame final ait exactement les mêmes colonnes (et dans le même ordre) que celles vues par le modèle pendant l'entraînement. Les colonnes dummy manquantes (catégories non présentes dans l'entrée utilisateur) sont ainsi créées et remplies avec 0.
  + **Raison :** La fidélité de cette fonction de prétraitement par rapport au notebook est la clé pour des prédictions fiables.
* **Interface Utilisateur (UI) :**
  + st.set\_pageb\_config() a été utilisé pour configurer le titre de la page et la mise en page.
  + Des titres (st.title), des descriptions (st.markdown), et des sous-titres (st.subheader) ont été utilisés pour structurer l'information.
  + Un formulaire (st.form) a été implémenté pour regrouper les champs d'entrée et ne déclencher la prédiction qu'après soumission.
  + Divers widgets Streamlit ont été employés pour la saisie des 16 features originales :
    1. st.number\_input pour l'âge.
    2. st.selectbox pour les variables catégorielles avec un nombre limité d'options (Gender, City, Profession, Sleep Duration, Dietary Habits, Degree).
    3. st.slider pour les variables numériques ordinales ou continues avec une plage définie (Academic Pressure, CGPA, etc.).
    4. st.radio pour les questions binaires (Pensées Suicidaires, Antécédents Familiaux).
  + Un bouton (st.form\_submit\_button) permet à l'utilisateur de lancer la prédiction.
* **Logique de Prédiction et Affichage :**
  + Lorsque l'utilisateur soumet le formulaire, les données des widgets sont collectées.
  + Elles sont passées à preprocess\_input\_streamlit.
  + Le DataFrame prétraité est ensuite fourni au model.predict() et model.predict\_proba().
  + Le résultat (prédiction de classe et probabilités) est affiché de manière claire à l'utilisateur, avec des messages conditionnels (st.success ou st.error) et des icônes.
  + Une barre latérale (st.sidebar) fournit des informations contextuelles sur l'application.

5. Lancement et Test de l'Application

L'application est lancée localement via la commande streamlit run streamlit\_app.py dans un terminal Anaconda. Elle devient alors accessible dans un navigateur web Des tests ont été effectués en entrant diverses combinaisons de valeurs pour s'assurer du bon fonctionnement du prétraitement et de la logique de prédiction.

**Flux des Données :**

1. **Utilisateur** -> Interagit avec les **Widgets Streamlit** (champs de saisie, menus déroulants, curseurs).
2. **Widgets Streamlit** -> Stockent les valeurs entrées dans des **variables Python** (dans notre cas, les clés du dictionnaire input\_data).
3. Quand submit\_button est vrai (cliqué) :
   * Le dictionnaire input\_data (contenant les **données brutes** de l'utilisateur) -> Est passé à la fonction preprocess\_input\_streamlit.
   * preprocess\_input\_streamlit -> Transforme input\_data en un **DataFrame Pandas (processed\_df)** avec le format exact attendu par le modèle (bonnes colonnes, encodage, scaling).
   * processed\_df -> Est passé à model.predict() et model.predict\_proba().
   * Les **résultats de la prédiction** -> Sont affichés à l'utilisateur via des commandes Streamlit (st.error, st.success, st.metric, st.markdown).