**🧾 Rapport de prétraitement du corpus DAIC-WOZ**

**⚙️ 1. Fusion et standardisation des métadonnées**

* Les fichiers train\_split\_Depression\_AVEC2017.csv, dev\_split\_Depression\_AVEC2017.csv et full\_test\_split.csv ont été chargés et concaténés pour former un seul tableau full\_df.
* La colonne PHQ\_Binary a été renommée en PHQ8\_Binary pour assurer la cohérence.
* Pour chaque participant, on a généré les chemins complets vers :
  + le fichier audio (\*\_AUDIO.wav)
  + le fichier de transcription (\*\_TRANSCRIPT.csv)
* Tous les chemins ont été sauvegardés dans un fichier daic\_paths.csv.

**🔉 2. Traitement des données audio**

Pour chaque fichier audio :

1. **Filtrage temporel** : seuls les intervalles où le locuteur est le **participant** sont extraits selon les timestamps de la transcription.
2. **Chargement** de l’audio à 16kHz via librosa.
3. **Segmentation** en tranches de 7.6 secondes.
4. **Augmentation** des segments audio par ajout de bruit gaussien (bruit blanc, noise\_factor=0.005).
5. **Extraction MFCC** :
   * 60 coefficients MFCC
   * n\_fft=1024, hop\_length=322, win\_length=960
   * Chaque segment est transformé en une matrice (378, 60).

**📝 3. Traitement des transcriptions textuelles**

Pour chaque fichier \*\_TRANSCRIPT.csv :

1. **Segmentation** temporelle du texte pour qu'elle corresponde aux segments audios (mêmes durées de 7.6 secondes).
2. **Tokenisation** du texte et transformation en vecteurs d’embedding :
   * Utilisation de GloVe 100 dimensions (glove.6B.100d.txt)
   * Pour chaque segment textuel, on prend jusqu'à 9 mots (max\_words=9)
   * Chaque segment est converti en matrice (378, 9), avec padding si nécessaire.

**🧩 4. Construction du dataset complet**

Pour chaque participant :

* Les MFCC audio (378, 60) et les embeddings texte (378, 9) sont appariés par segment.
* Chaque segment est étiqueté avec la classe binaire (PHQ8\_Binary).
* Les listes finales (X\_audio, X\_text, y, ids) sont sauvegardées dans :
  + processed\_dataset.npz (données brutes transformées, avant split)

**🧪 5. Création d’un split équilibré (Train / Dev / Test)**

* Les segments sont regroupés par classe (0 ou 1).
* À partir des plus petites classes, on extrait **10% pour dev** et **10% pour test**, de manière **équilibrée**.
* Le reste est utilisé pour train.
* Pour chaque split, on extrait :
  + Les segments audio (numpy arrays de shape (378, 60))
  + Les segments texte (numpy arrays de shape (378, 9))
  + Les étiquettes
  + Les identifiants des participants
* Sauvegarde finale dans :
  + dataset\_balanced.npz ✅
  + test\_ids.npy pour récupérer les IDs de test plus tard 🔖

**📦 Structure du fichier final dataset\_balanced.npz :**

* X\_audio\_train, X\_text\_train, y\_train, ids\_train
* X\_audio\_dev, X\_text\_dev, y\_dev, ids\_dev
* X\_audio\_test, X\_text\_test, y\_test, ids\_test

**📈 Contrôle qualité effectué**

* Vérification du **nombre d’exemples** dans chaque split.
* Vérification de l’**équilibre des classes** dans dev et test (égalité parfaite entre classes 0 et 1).
* Vérification des **shapes** :
  + audio segments → (378, 60)
  + texte segments → (378, 9)

**✅ Conclusion**

Le jeu de données est prêt pour l’entraînement d’un modèle multimodal basé sur l’audio et le texte. Il est équilibré, segmenté temporellement et formaté sous forme de tenseurs exploitables par un réseau de neurones. Le prétraitement respecte l’alignement entre audio et texte tout en incluant une légère augmentation des données (bruit) pour améliorer la robustesse du modèle.