**Analyse projet RTS**

Mon approche pour réaliser ce projet d'évaluation en tant qu'Analyste de Données pour la RTS sera structurée en quatre phases distinctes, visant à satisfaire les besoins stratégiques de la Spécialiste Distribution concernant la planification des productions de contenu pour l'année prochaine.

L'objectif principal est de déterminer le volume de contenu à produire pour les cinq thèmes principaux (Info, Sport, Musique, Société, Humour) en fonction de leur rôle stratégique (Acquisition ou Rétention/Fidélisation).

# Phase 0 : KPIs

In this data set, two files are provided:

* **Mesures\_contenu\_volume\_audio\_à\_commander.csv** presenting all metrics for audio shows and episodes (called segments) from 2024 to June 2025.
* **Correspondance\_show\_segment\_tag.csv** presenting a referential table with tags associated to each episode (segment).

In those tables you will find the following metrics:

* **segment ID** = unique ID of a segment, which is an episode
* **segment** = title of a segment (a show can have several segments)
* **show ID** = unique ID of a show
* **show** = title of a show
* **publication date** = format YYYY-MM-DD
* **app/site name** = platform on which the show was listened (rts.ch OR rts-app-play OR rts-app-info OR rts-app-sport)
* **device class** = device on which the show was listened (pc/laptop OR smartphone)
* **episode length** = length of a segment in seconds
* **assigned tags** = tags correspond to attributed themes of the content (available from January 2025, on the website rts.ch only)  
  We only want to focus on the 5 following main themes that you can identify as the following in the data set:
  + media\_radio:societe
  + media\_radio:humour
  + media\_radio:info
  + media\_radio:musique
  + media\_radio:sport
* **media views** = number of times a media has been listened
* **avg play duration** = Total Play duration / Media Views \* 100. Format hh:mm:ss, average consumption time generated by each media view of a segment.
* **visitors** = number of unique devices going through the page that generated a play of the segment.
* **new visit rate %** = new visits / visits
* **entries** = number of visits whose first page view corresponds to the play of the segment.
* **exits** = number of visits whose last page view corresponds to the play of the segment.
* **returning visits** = number of visits going through the page that generated a play of the segment when the visitor is already detected in the database.
* **bounces** = number of visits with only 1 page view
* total play duration = total volume of consumption time generated by all media views assigned to the segment. Format hh:mm:ss

| **KPI** | **Description / Interprétation** | **Remarques et points d’attention** |
| --- | --- | --- |
| **segment ID / segment / show ID / show** | Identifiants et libellés des émissions et épisodes. | Assure-toi que le lien entre *show ID* et *segment ID* est **unique et cohérent** (un show avec plusieurs segments). Si ce n’est pas le cas → suspicion de doublons. |
| **publication date** | Date de mise en ligne de l’épisode. | Peut être utilisée pour vérifier la cohérence temporelle (par exemple, si un segment a des écoutes avant sa publication → anomalie). |
| **app/site name** | Plateforme d’écoute. | Donnée utile pour segmenter l’audience. Attention : les écoutes “rts-app-\*” et “rts.ch” ne sont pas directement comparables (modes de tracking souvent différents). |
| **device class** | Type d’appareil utilisé. | Vérifie si les devices sont bien classés : parfois les tablettes sont confondues avec PC ou smartphones. |
| **episode length** | Durée totale du segment (en secondes). | Peut servir de base de normalisation (ex. comparer *avg play duration / episode length*). Vérifie s’il existe des durées nulles ou aberrantes (> 4h par exemple). |
| **assigned tags** | Thèmes attribués aux contenus (présents à partir de janv. 2025). | ⚠️ Attention : avant 2025 → pas de tag. Cela crée une **rupture temporelle** dans les analyses. Pour les mois 2024, il faudra traiter les tags comme “non disponibles”. |
| **media views** | Nombre de fois qu’un segment a été joué. | Vérifie si c’est “nombre de lectures totales” ou “nombre de clics sur play”. Dans certains trackers, une lecture interrompue puis relancée compte plusieurs fois. |
| **avg play duration** | Moyenne de la durée d’écoute par lecture : [Total Play Duration / Media Views]. | La formule donnée semble **fausse** : “*Total Play Duration / Media Views × 100*” → le ×100 est incohérent. Il ne faut pas multiplier par 100. Il s’agit simplement d’une moyenne en secondes ou au format hh:mm:ss. |
| **visitors** | Nombre d’appareils uniques ayant généré au moins une écoute. | Si *media views > visitors* → normal (plusieurs lectures par appareil). Si l’inverse se produit → anomalie. |
| **new visit rate %** | Pourcentage de nouveaux visiteurs : (nouveaux visiteurs / total visites). | Vérifie la cohérence avec “returning visits”. La somme new + returning ≈ total. Si >100% ou <80% → problème de tracking. |
| **entries** | Visites dont la première page vue est la page du segment. | Donnée utile pour savoir si le segment attire directement les visiteurs.  Si *entries > visits* → incohérence. |
| **exits** | Visites dont la dernière page vue est la page du segment. | Complémentaire à entries : peut montrer un contenu de “fin de parcours”. Attention si exits = visits → rebonds. |
| **returning visits** | Visites d’utilisateurs déjà connus. | Devrait être cohérent avec le “new visit rate %”. |
| **bounces** | Visites avec une seule page vue. | Important pour juger l’engagement : taux de rebond = bounces / visits. S’il est élevé (>70%), cela peut signifier un contenu consommé sans navigation supplémentaire. |
| **total play duration** | Temps total d’écoute cumulé (hh:mm:ss). | Doit être cohérent avec avg play duration × media views. |

# Phase 1 : Compréhension et Préparation des Données

## Nettoyage data Mesures

### Script Python

|  |
| --- |
| import pandas as pd  import io  # Nom du fichier initial  file\_path = "Mesures V0.csv"  # --- Fonction de conversion de durée ---  def convert\_duration\_to\_seconds(duration\_str):  if not isinstance(duration\_str, str):  return None  try:  parts = duration\_str.split(':')  if len(parts) == 3: # Format H:M:S  h, m, s = map(int, parts)  return (h \* 3600) + (m \* 60) + s  elif len(parts) == 2: # Format M:S  m, s = map(int, parts)  return (m \* 60) + s  else:  return None  except (ValueError, TypeError):  return None  try:  # --- Étape 0: Chargement Initial et Correction des Colonnes ---  print("--- Étape 0: Chargement Initial et Correction des Colonnes ---")  df = pd.read\_csv(file\_path, sep=';')    # \*\*\* CORRECTION \*\*\*: Nettoyer les noms de colonnes  df.columns = df.columns.str.strip()  print("Noms de colonnes nettoyés (suppression des espaces blancs).")    rows\_start = len(df)  print(f"Nombre de lignes initial: {rows\_start}")  # --- Étape 1: Suppression des valeurs manquantes ---  print("\n--- Étape 1: Suppression des valeurs manquantes ---")  print(f"Nombre de lignes avant: {len(df)}")  # Utilisation des noms de colonnes nettoyés  subset\_cols = ['Show ID', 'Show', 'Publication Date', 'App/Site Name', 'Device Class']  df\_step1 = df.dropna(subset=subset\_cols)  print(f"Nombre de lignes après: {len(df\_step1)}")    file\_step1 = "Mesures\_V1\_no\_missing.csv"  df\_step1.to\_csv(file\_step1, index=False, sep=';')  print(f"Fichier sauvegardé: {file\_step1}")  # --- Étape 2: Conversion de 'Publication Date' ---  print("\n--- Étape 2: Conversion de 'Publication Date' ---")  print(f"Nombre de lignes avant: {len(df\_step1)}")  df\_step2 = df\_step1.copy()  df\_step2['Publication Date'] = pd.to\_datetime(df\_step2['Publication Date'], format='%d.%m.%Y', errors='coerce')    print(f"Nombre de lignes après: {len(df\_step2)}")  file\_step2 = "Mesures\_V2\_dates\_converted.csv"  df\_step2.to\_csv(file\_step2, index=False, sep=';')  print(f"Fichier sauvegardé: {file\_step2}")  # --- Étape 3: Conversion de 'New Visit Rate %' ---  print("\n--- Étape 3: Conversion de 'New Visit Rate %' ---")  print(f"Nombre de lignes avant: {len(df\_step2)}")  df\_step3 = df\_step2.copy()    df\_step3['New Visit Rate'] = df\_step3['New Visit Rate %'].str.replace('%', '', regex=False).str.replace(',', '.', regex=False)  df\_step3['New Visit Rate'] = pd.to\_numeric(df\_step3['New Visit Rate'], errors='coerce') / 100  df\_step3 = df\_step3.drop(columns=['New Visit Rate %'])    print(f"Nombre de lignes après: {len(df\_step3)}")  file\_step3 = "Mesures\_V3\_percent\_converted.csv"  df\_step3.to\_csv(file\_step3, index=False, sep=';')  print(f"Fichier sauvegardé: {file\_step3}")  # --- Étape 4: Conversion des durées ---  print("\n--- Étape 4: Conversion des durées ('Avg Play Duration', 'Total Play Duration') ---")  print(f"Nombre de lignes avant: {len(df\_step3)}")  df\_step4 = df\_step3.copy()    df\_step4['Avg Play Seconds'] = df\_step4['Avg Play Duration'].apply(convert\_duration\_to\_seconds)  df\_step4['Total Play Seconds'] = df\_step4['Total Play Duration'].apply(convert\_duration\_to\_seconds)  df\_step4 = df\_step4.drop(columns=['Avg Play Duration', 'Total Play Duration'])    print(f"Nombre de lignes après: {len(df\_step4)}")  file\_step4 = "Mesures\_V4\_duration\_converted.csv"  df\_step4.to\_csv(file\_step4, index=False, sep=';')  print(f"Fichier sauvegardé: {file\_step4}")    # --- Étape 5: Vérification des colonnes catégorielles ---  print("\n--- Étape 5: Vérification des colonnes catégorielles ---")  print(f"Nombre de lignes du fichier final: {len(df\_step4)}")    print("\nValeurs uniques pour 'App/Site Name':")  print(df\_step4['App/Site Name'].unique())    print("\nValeurs uniques pour 'Device Class':")  print(df\_step4['Device Class'].unique())  # --- Nettoyage Terminé: Affichage du résultat final ---  print("\n--- Nettoyage Terminé ---")  print(f"Aperçu du fichier final ({file\_step4}):")  print(df\_step4.head().to\_markdown(index=False, numalign="left", stralign="left"))    print("\nInformations sur le fichier final:")  buffer = io.StringIO()  df\_step4.info(buf=buffer)  print(buffer.getvalue())  except Exception as e:  print(f"Une erreur est survenue lors du nettoyage : {e}") |

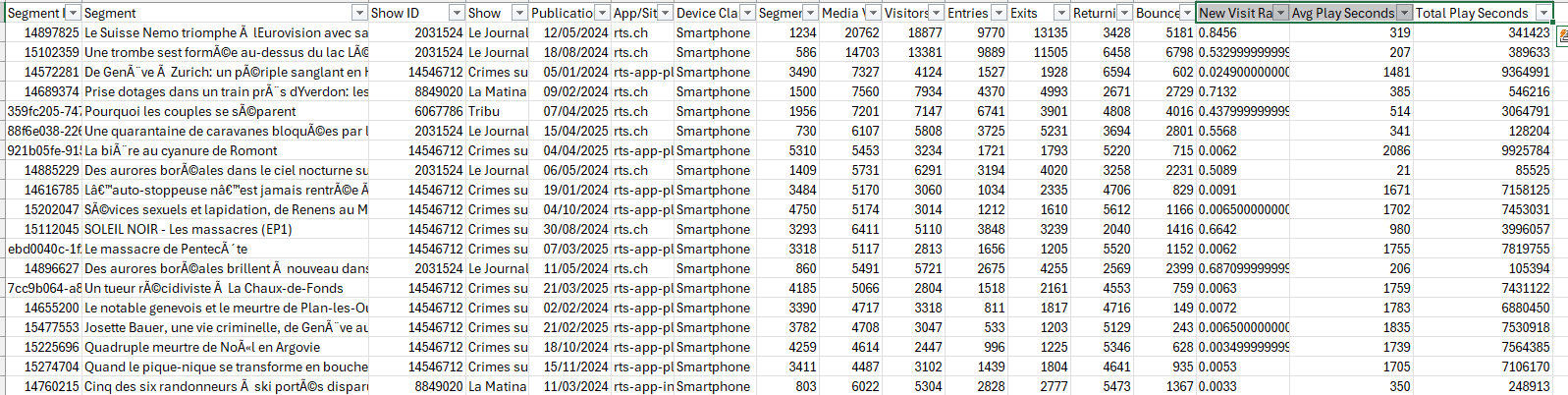
### Résultat

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

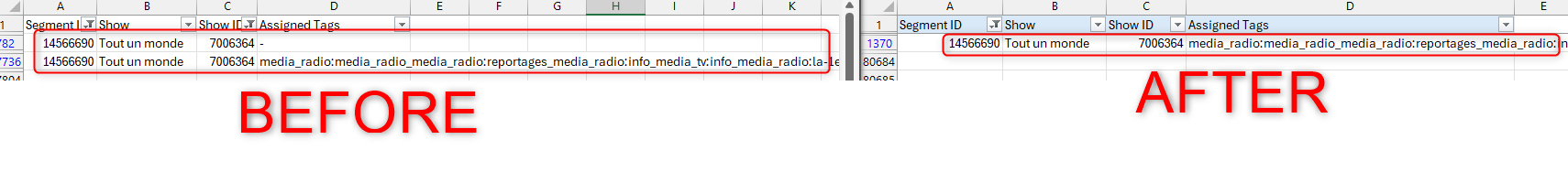
AI-generated content may be incorrect.



## Fichier « Correspondance\_show\_segment\_tag » suppression des doublons entre « Segment ID » et « Show ID »

Supprimer les doublons sur les champs « Segment ID » et « Show ID » avec comme critère de ne garder que les lignes où les champs « Assigned Tags » se sont pas vides.

Ex :



#### Script pour Supprimer les doublons sur les champs « Segment ID » et « Show ID »

Code

|  |
| --- |
| import pandas as pd  # --- Configuration ---  # REMPLACEZ par le nom de votre fichier d'entrée  fichier\_entree = 'Tags V0.1.csv'  # REMPLACEZ par le nom souhaité pour le fichier de sortie  fichier\_sortie = 'Tags nettoye V0.2.csv'  # ---------------------  try:  # --- 1. Lecture du fichier ---  # Ajout de sep=';' pour spécifier le séparateur point-virgule  try:  df = pd.read\_csv(fichier\_entree, sep=';')  except UnicodeDecodeError:  # Essayer avec un encodage différent si le premier échoue  # Ajout de sep=';' ici aussi  df = pd.read\_csv(fichier\_entree, sep=';', encoding='latin1')  except Exception as e:  print(f"Erreur lors de la lecture du fichier : {e}")  print("S'il s'agit d'un fichier Excel, utilisez pd.read\_excel('votre\_fichier.xlsx')")  exit()  print(f"Fichier initial chargé : {len(df)} lignes.")  # S'assurer que les colonnes de tags sont bien des chaînes de caractères  # .fillna('') remplace les cellules vides (NaN) par une chaîne vide  df['Assigned Tags'] = df['Assigned Tags'].fillna('').astype(str)  # --- 2. Identification des lignes à supprimer ---  # On crée une colonne temporaire.  # Elle sera 'True' si le tag, après avoir enlevé les espaces au début/fin,  # se termine par un '-'. C'est le critère de la ligne à \*supprimer\*.  df['est\_indesirable'] = df['Assigned Tags'].str.strip().str.endswith('-')  # --- 3. Tri des données ---  # On trie par les colonnes ID, puis par notre colonne 'est\_indesirable'.  # ascending=True fait que 'False' (à garder) vient AVANT 'True' (à supprimer).  df\_trie = df.sort\_values(  by=['Segment ID', 'Show ID', 'est\_indesirable'],  ascending=[True, True, True]  )  # --- 4. Suppression des doublons ---  # Pour chaque groupe ('Segment ID', 'Show ID'), on garde la première ligne ('first').  # Grâce au tri, la première ligne est toujours celle qu'on veut garder.  df\_propre = df\_trie.drop\_duplicates(  subset=['Segment ID', 'Show ID'],  keep='first'  )  # --- 5. Nettoyage final ---  # On supprime la colonne temporaire qui n'est plus nécessaire  df\_propre = df\_propre.drop(columns=['est\_indesirable'])  # --- 6. Sauvegarde du résultat ---  # On sauvegarde le fichier nettoyé avec le séparateur point-virgule  # Adaptez si vous voulez un fichier Excel (df\_propre.to\_excel(fichier\_sortie, index=False))  df\_propre.to\_csv(fichier\_sortie, index=False, sep=';')  print(f"Nettoyage terminé. {len(df\_propre)} lignes sauvegardées dans '{fichier\_sortie}'.")  except FileNotFoundError:  print(f"ERREUR : Le fichier '{fichier\_entree}' n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur inattendue est survenue : {e}") |

#### Résultat

*Fichier initial chargé : 107753 lignes.*

*Nettoyage terminé. 80682 lignes sauvegardées dans 'fichier\_nettoye.csv'.*

## Fichier « Correspondance\_show\_segment\_tag » comptage des mots dans la colonne Tag

#### Script pour compter les mots uniques dans la colonne « Assigned Tag »

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  from collections import Counter  import re  # --- 1. Configuration ---  NOM\_DU\_FICHIER = 'fichier\_nettoye.csv'  COLONNE\_TAGS = 'Assigned Tags'  TOP\_N\_TAGS = 20  # --- 2. Chargement des données ---  print(f"Tentative de chargement du fichier : {NOM\_DU\_FICHIER}...")  try:  # --- MODIFICATION ICI ---  # On spécifie que le séparateur est un point-virgule  df = pd.read\_csv(NOM\_DU\_FICHIER, sep=';')  # --- FIN DE LA MODIFICATION ---    print("Fichier chargé avec succès.")    except FileNotFoundError:  print(f"\n--- ERREUR ---")  print(f"Le fichier '{NOM\_DU\_FICHIER}' n'a pas été trouvé.")  print("Veuillez vous assurer que le fichier est dans le même dossier que ce script Python.")  exit()  except Exception as e:  print(f"\nUne erreur inattendue est survenue lors de la lecture du fichier : {e}")  print("\nCONSEIL : Si l'erreur persiste, ouvrez votre CSV dans un bloc-notes pour voir quel caractère sépare les colonnes (virgule, point-virgule, tabulation...).")  exit()  # --- 3. Vérification de la colonne ---  if COLONNE\_TAGS not in df.columns:  print(f"\n--- ERREUR ---")  print(f"La colonne '{COLONNE\_TAGS}' n'existe pas dans votre fichier CSV.")  # Affiche les colonnes trouvées, en retirant les espaces superflus  colonnes\_trouvees = [col.strip() for col in df.columns]  print(f"Les colonnes trouvées sont : {colonnes\_trouvees}")  print(f"Vérifiez que le nom '{COLONNE\_TAGS}' est exact (majuscules/minuscules/espaces).")  exit()  # --- 4. Nettoyage et comptage des tags ---  # Remplacer les cellules vides (NaN) par une chaîne vide  df[COLONNE\_TAGS] = df[COLONNE\_TAGS].fillna('')  # Créer un 'Counter' pour stocker les comptages  tag\_counter = Counter()  print("Analyse des tags...")  # Parcourir chaque ligne de la colonne  for entry in df[COLONNE\_TAGS]:    entry\_str = str(entry).strip() # Convertir en chaîne et enlever espaces début/fin  # Ignorer les lignes vides ou celles contenant juste ' -'  if not entry\_str or entry\_str == '-':  continue    # 1. Nettoyer la chaîne :  tag\_string = re.sub(r'^\d+\s\*', '', entry\_str).replace('\_', ':')    # 2. Séparer la chaîne par ':' pour obtenir les mots  tags = tag\_string.split(':')    # 3. Mettre à jour le compteur  tag\_counter.update(tag for tag in tags if tag) # 'if tag' ignore les chaînes vides  # --- 5. Affichage des résultats (Console) ---  print("\n--- Comptage des Tags ---")  if not tag\_counter:  print("Aucun tag valide n'a été trouvé après analyse.")  else:  print(f"Voici les {TOP\_N\_TAGS} tags les plus courants :")  for tag, count in tag\_counter.most\_common(TOP\_N\_TAGS):  print(f"{tag}: {count}")  # --- 6. Création du graphique ---  if tag\_counter:  common\_tags = tag\_counter.most\_common(TOP\_N\_TAGS)  common\_tags.reverse()    labels = [tag for tag, count in common\_tags]  counts = [count for tag, count in common\_tags]    plt.figure(figsize=(12, 8))  plt.barh(labels, counts, color='steelblue')  plt.title(f'Top {TOP\_N\_TAGS} des tags les plus courants')  plt.xlabel('Nombre d\'occurrences')  plt.ylabel('Tags')  plt.grid(axis='x', linestyle='--', alpha=0.7)  plt.tight\_layout()    print("\nAffichage du graphique...")  plt.show()  else:  print("\nAucune donnée à afficher dans le graphique.") |

#### Résultat

--- Comptage des Tags ---

Voici les 20 tags les plus courants :

media: 166868

radio: 154229

info: 64978

rts: 42874

la-1ere: 31901

tv: 12622

couleur3: 7876

monde: 4495

musique: 4265

culture: 4026

espace-2: 3635

sport: 3140

suisse: 3056

humour: 2491

reportages: 2191

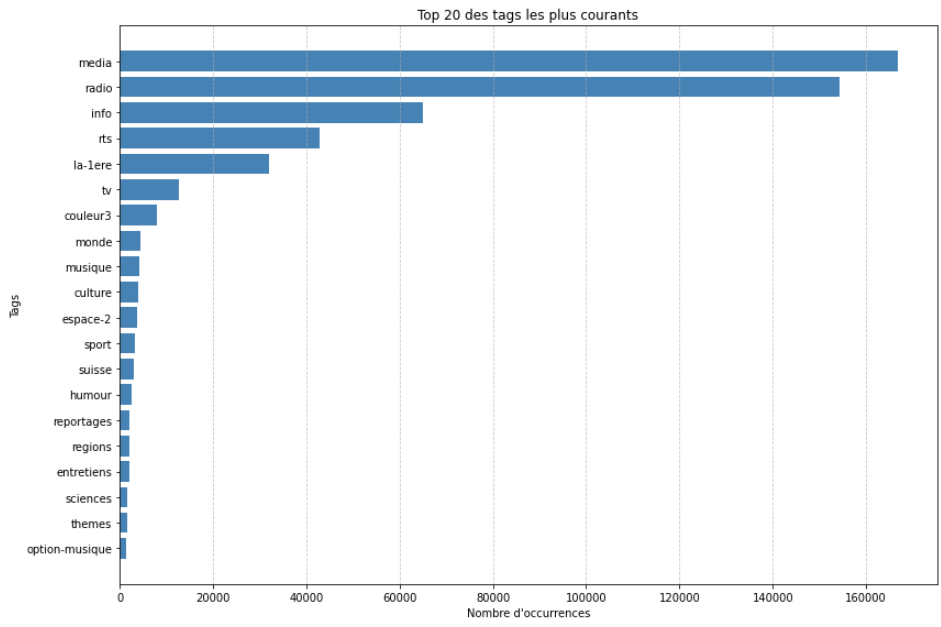
regions: 2172

entretiens: 2075

sciences: 1606

themes: 1500

option-musique: 1481

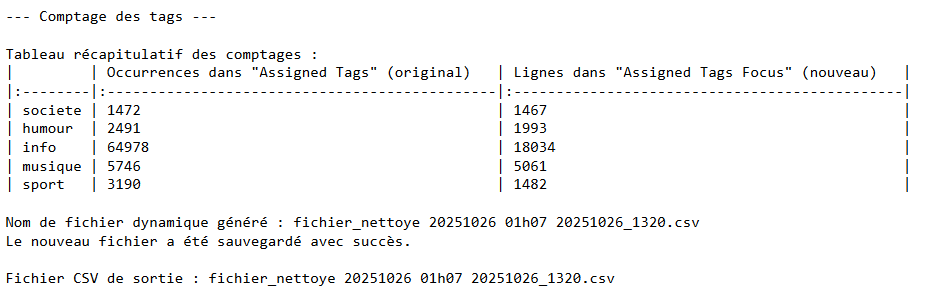


## Fichier « Correspondance\_show\_segment\_tag » on ne garde que les mots « societe, humour, info, musique, sport »

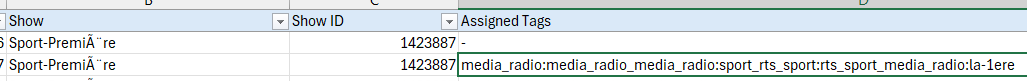
#### Script pour afficher seulement les mots « societe, humour, info, musique, sport »

|  |
| --- |
| import pandas as pd  from datetime import datetime  import re # Bibliothèque pour les expressions régulières (utilisée pour le comptage)  # 1. Définir la liste des mots-clés  focus\_tags = ["societe", "humour", "info", "musique", "sport"]  # 2. Charger le fichier CSV  file\_name = "fichier\_nettoye 20251026 01h07.csv"  try:  df = pd.read\_csv(file\_name, sep=';')  print(f"Fichier {file\_name} chargé avec succès.")  # 3. Définir la fonction d'extraction (inchangée)  def extract\_focus\_tags(tag\_string):  """  Extrait les mots-clés (focus\_tags) d'une chaîne de tags.  """  if not isinstance(tag\_string, str) or tag\_string == '-':  return ''    found\_tags = []  tag\_string\_lower = tag\_string.lower()    for tag in focus\_tags:  # Vérifie si le mot-clé est présent  if tag in tag\_string\_lower:  found\_tags.append(tag)    # Retourne les tags trouvés, séparés par une virgule  return ','.join(found\_tags)  # 4. Appliquer la fonction pour créer la nouvelle colonne  df['Assigned Tags Focus'] = df['Assigned Tags'].apply(extract\_focus\_tags)  print("Nouvelle colonne 'Assigned Tags Focus' créée.")  # 5. Compter les occurrences (Nouveau)  print("\n--- Comptage des tags ---")    # Préparation des colonnes pour le comptage (gestion des NaN et conversion en str/minuscules)  df\_tags\_original\_lower = df['Assigned Tags'].astype(str).str.lower()  df\_tags\_focus = df['Assigned Tags Focus'].astype(str) # Déjà en minuscules par la fonction  stats = {}  for tag in focus\_tags:  # Compte le nombre de fois que le tag (substring) apparaît dans la colonne originale  # .str.count() compte les occurrences de sous-chaînes (ex: "info" dans "rts\_info")  count\_original = df\_tags\_original\_lower.str.count(tag, flags=re.IGNORECASE).sum()    # Compte le nombre de lignes contenant le tag dans la nouvelle colonne  # .str.contains() vérifie si la ligne contient le tag (ex: "info" ou "info,societe")  count\_focus = df\_tags\_focus.str.contains(tag, na=False).sum()    stats[tag] = {  'Occurrences dans "Assigned Tags" (original)': int(count\_original),  'Lignes dans "Assigned Tags Focus" (nouveau)': int(count\_focus)  }    # Afficher les statistiques sous forme de tableau  stats\_df = pd.DataFrame(stats).T  print("\nTableau récapitulatif des comptages :")  print(stats\_df.to\_markdown(numalign="left", stralign="left"))  # 6. Générer le nom de fichier dynamique (Nouveau)  now = datetime.now()  # Format: YYYYMMDD\_HHMM (AnnéeMoisJour\_HeureMinute)  timestamp = now.strftime("%Y%m%d\_%H%M")    base\_name = "fichier\_nettoye 20251026 01h07"  output\_file\_name = f"{base\_name} {timestamp}.csv"  print(f"\nNom de fichier dynamique généré : {output\_file\_name}")  # 7. Sauvegarder le DataFrame modifié  df.to\_csv(output\_file\_name, sep=';', index=False, encoding='utf-8')  print(f"Le nouveau fichier a été sauvegardé avec succès.")  print(f"\nFichier CSV de sortie : {output\_file\_name}")  except Exception as e:  print(f"Une erreur est survenue : {e}") |

#### Résultat



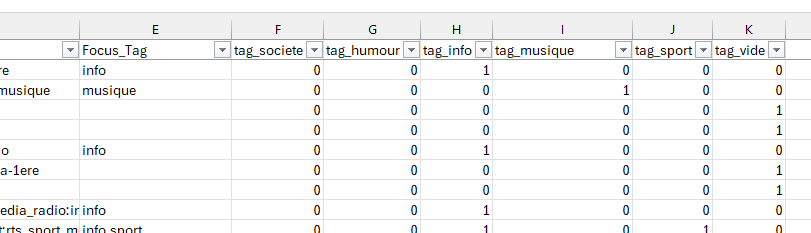
Le nombre d’occurrences diminue car sur une même cellule on peut avoir plusieurs fois le même mot. Exemple :



## Création des 5 colonnes tag + 1 colonne vide

#### Script Python

|  |
| --- |
| import pandas as pd  # Définir les noms de fichiers  input\_file = "fichier\_nettoye 20251026 01h07 20251028\_2143.csv"  output\_file = "fichier\_avec\_tags\_colonnes.csv"  try:  # Charger le fichier CSV avec le bon séparateur  df = pd.read\_csv(input\_file, sep=';')  # Liste des tags pour lesquels créer des colonnes  tags = ['societe', 'humour', 'info', 'musique', 'sport']  # Créer une colonne binaire (0 ou 1) pour chaque tag  for tag in tags:  col\_name = f"tag\_{tag}"  # .astype(str) gère les valeurs NaN (null) en les traitant comme la chaîne 'nan'  # .str.contains(tag, case=False, na=False) recherche le tag sans tenir compte de la casse  # et traite les NaN restants comme False (pas de correspondance)  # .astype(int) convertit True/False en 1/0  df[col\_name] = df['Focus\_Tag'].astype(str).str.contains(tag, case=False, na=False).astype(int)  # Créer la colonne 'tag\_vide'  # Elle est à 1 si 'Focus\_Tag' est nul (NaN) ou égal à '-' (basé sur l'aperçu du fichier)  # L'aperçu du fichier montrait 'NaN' pour les valeurs manquantes après lecture,  # donc isnull() est la vérification la plus fiable.  df['tag\_vide'] = (df['Focus\_Tag'].isnull() | (df['Focus\_Tag'] == '-')).astype(int)  # Sauvegarder le DataFrame modifié dans un nouveau fichier CSV  df.to\_csv(output\_file, sep=';', index=False)  print(f"Le fichier '{output\_file}' a été créé avec succès.")  # (Aperçu et infos affichés dans la sortie de code)  except FileNotFoundError:  print(f"Erreur : Le fichier '{input\_file}' n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur est survenue : {e}") |



## Harmonization si tag existe à 30% alors 1 partout

#### Script Python

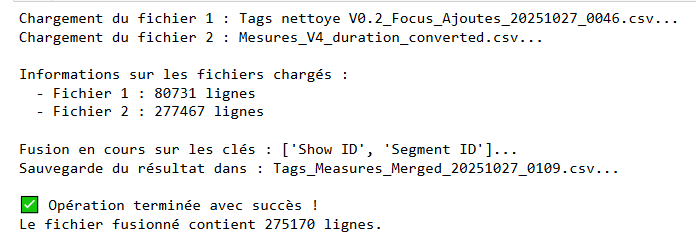
|  |
| --- |
| import pandas as pd  # Nom du fichier d'entrée  file\_name = 'fichier\_avec\_tags\_colonnes.csv'  try:  # Charger le fichier CSV avec le bon séparateur  df = pd.read\_csv(file\_name, sep=';')    print("Données initiales (5 premières lignes) :")  print(df.head())  print("\nInformations sur les colonnes et types de données :")  df.info()  # Identifier dynamiquement toutes les colonnes de tags  # (celles qui commencent par 'tag\_')  tag\_columns = [col for col in df.columns if col.startswith('tag\_')]    if not tag\_columns:  print("Erreur : Aucune colonne commençant par 'tag\_' n'a été trouvée.")  else:  print(f"\nColonnes de tags identifiées : {tag\_columns}")  # Définir le seuil (30%)  threshold = 0.3  # Créer une copie du DataFrame pour l'harmonisation  df\_harmonized = df.copy()  # Calculer les proportions en groupant par 'Show ID'  # .transform('mean') calcule la moyenne pour chaque groupe ('Show ID')  # et propage cette moyenne à toutes les lignes appartenant à ce groupe.  tag\_proportions = df.groupby('Show ID')[tag\_columns].transform('mean')  # Appliquer la règle :  # Si la proportion du groupe est >= 0.3, mettre 1  # Sinon, mettre 0  harmonized\_values = (tag\_proportions >= threshold).astype(int)  # Remplacer les anciennes valeurs de tags par les nouvelles valeurs harmonisées  df\_harmonized[tag\_columns] = harmonized\_values  # --- Section de Vérification (optionnelle mais utile) ---    # Trouver un 'Show ID' avec plusieurs entrées pour montrer la différence  show\_id\_counts = df['Show ID'].value\_counts()  # Filtrer pour les Show ID qui apparaissent plus d'une fois  multi\_segment\_shows = show\_id\_counts[show\_id\_counts > 1].index    if not multi\_segment\_shows.empty:  sample\_show\_id = multi\_segment\_shows[0] # Prendre le premier    print(f"\n--- Exemple de vérification pour Show ID : {sample\_show\_id} ---")    print("\nAvant Harmonisation :")  print(df[df['Show ID'] == sample\_show\_id][['Show ID'] + tag\_columns].head())    print("\nAprès Harmonisation :")  print(df\_harmonized[df\_harmonized['Show ID'] == sample\_show\_id][['Show ID'] + tag\_columns].head())  else:  print("\nPas de 'Show ID' avec plusieurs segments trouvé pour l'exemple de vérification.")    # --- Fin de la vérification ---  # Sauvegarder le fichier harmonisé  output\_file = 'fichier\_harmonise.csv'  df\_harmonized.to\_csv(output\_file, sep=';', index=False, encoding='utf-8')  print(f"\nL'harmonisation est terminée.")  print(f"Le nouveau fichier a été enregistré sous : {output\_file}")  except FileNotFoundError:  print(f"Erreur : Le fichier '{file\_name}' n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur est survenue : {e}") |

## Merge des 2 CSV

#### Script Python

|  |
| --- |
| import pandas as pd  from datetime import datetime  # --- 1. Définir les noms de fichiers et les clés ---  # Noms de vos fichiers d'entrée  fichier1 = "Tags nettoye V0.2\_Focus\_Ajoutes\_20251027\_0046.csv"  fichier2 = "Mesures\_V4\_duration\_converted.csv"  # Clés primaires pour la fusion  cles\_de\_fusion = ["Show ID", "Segment ID"]  # --- 2. Générer le nom du fichier de sortie ---  # Obtenir la date et l'heure actuelles  maintenant = datetime.now()  # Formater la date et l'heure (ex: 20251027\_1305)  timestamp = maintenant.strftime("%Y%m%d\_%H%M")  # Créer le nom de fichier complet  fichier\_sortie = f"Tags\_Measures\_Merged\_{timestamp}.csv"  # --- 3. Exécuter le processus de fusion ---  try:  # Charger les fichiers CSV dans des DataFrames Pandas  # NOTE : Si vos fichiers d'ENTRÉE utilisent aussi des ';',  # vous devrez ajouter sep=';' à pd.read\_csv() aussi.  print(f"Chargement du fichier 1 : {fichier1}...")  df1 = pd.read\_csv(fichier1, sep=';')    print(f"Chargement du fichier 2 : {fichier2}...")  df2 = pd.read\_csv(fichier2, sep=';')  print("\nInformations sur les fichiers chargés :")  print(f" - Fichier 1 : {len(df1)} lignes")  print(f" - Fichier 2 : {len(df2)} lignes")  # Fusionner (merge) les deux DataFrames  print(f"\nFusion en cours sur les clés : {cles\_de\_fusion}...")  df\_fusionne = pd.merge(df1, df2, on=cles\_de\_fusion)  # Sauvegarder le DataFrame fusionné dans un nouveau fichier CSV  print(f"Sauvegarde du résultat dans : {fichier\_sortie}...")    # --- MODIFICATION ICI ---  # Ajout de sep=';' pour que le fichier de sortie utilise  # des points-virgules.  df\_fusionne.to\_csv(  fichier\_sortie,  index=False,  encoding='utf-8-sig',  sep=';'  )  print("\n✅ Opération terminée avec succès !")  print(f"Le fichier fusionné contient {len(df\_fusionne)} lignes.")  except FileNotFoundError:  print("\n❌ ERREUR : Fichier non trouvé.")  print("Veuillez vérifier que les fichiers suivants sont dans le même dossier que le script :")  print(f" - {fichier1}")  print(f" - {fichier2}")  except KeyError as e:  print(f"\n❌ ERREUR : Colonne clé non trouvée.")  print(f"La colonne {e} n'a pas été trouvée dans l'un des fichiers.")  print("Vérifiez que les deux fichiers contiennent bien 'Show ID' et 'Segment ID'.")  except Exception as e:  print(f"\n❌ Une erreur inattendue est survenue : {e}") |

#### Résultat



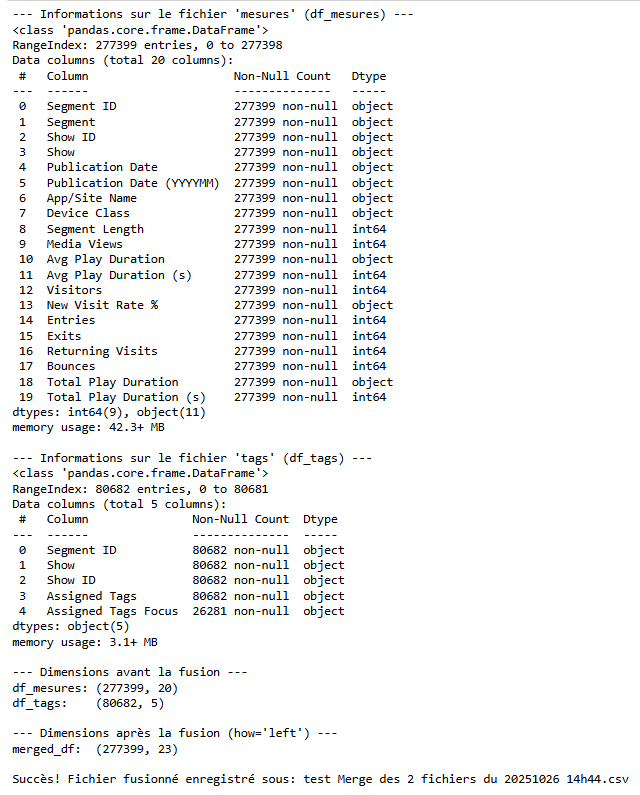
**IMPORTANT : ON PERD : 277467 – 275170 = 2297 lignes, soit environ 0.9 % des datas, ce qui est acceptable**

## Analyse du fichier Merge

#### Script Python

|  |
| --- |
| import pandas as pd  import matplotlib.pyplot as plt  import os  # --- Début de la fonction d'analyse par année ---  def analyze\_year(df, year\_to\_analyze):  """  Fonction pour filtrer, agréger et tracer les données pour une année spécifique.  Génère 3 graphiques : Ligne (évolution), Barres (Total), Circulaire (Répartition).  """    print(f"\n--- Début de l'analyse pour l'année : {year\_to\_analyze} ---")    # 1. Filtrer le DataFrame pour l'année spécifiée  df\_year = df[df['Year'] == str(year\_to\_analyze)].copy()    if df\_year.empty:  print(f"Aucune donnée trouvée pour l'année {year\_to\_analyze}. L'analyse pour cette année est sautée.")  return  print(f"{len(df\_year)} lignes trouvées pour {year\_to\_analyze}.")  # 2. Agrégation des données (par mois/tag)  print(f"Étape A ({year\_to\_analyze}) : Agrégation des 'Media Views'...")    df\_agg = df\_year.groupby(['Publication Date (YYYYMM)', 'Assigned Tags Focus'])['Media Views'].sum().reset\_index()    # 3. Sauvegarde des données agrégées  aggregated\_file = f'media\_views\_by\_tag\_{year\_to\_analyze}.csv'  df\_agg.to\_csv(aggregated\_file, index=False, sep=';', encoding='utf-8')  print(f"Données agrégées pour {year\_to\_analyze} sauvegardées dans : {aggregated\_file}")  # 4. Calculer le total des vues par tag pour CETTE année  total\_views\_per\_tag = df\_year.groupby('Assigned Tags Focus')['Media Views'].sum()    if total\_views\_per\_tag.empty:  print(f"Aucune donnée de tag à visualiser pour {year\_to\_analyze}.")  return  # --- NOUVEAU GRAPHIQUE 1 : Graphique à Barres (Top 15) ---  print(f"Étape B ({year\_to\_analyze}) : Création du graphique à barres (Top 15)...")    # Obtenir le Top 15  top\_15\_tags\_total = total\_views\_per\_tag.nlargest(15)    plt.figure(figsize=(12, 8))  # Trier par valeur pour un affichage horizontal propre (du plus petit au plus grand)  top\_15\_tags\_total.sort\_values().plot(kind='barh')    plt.title(f'Top 15 des Tags par "Media Views" (Total {year\_to\_analyze})')  plt.xlabel('Total Media Views')  plt.ylabel('Assigned Tags Focus')  plt.tight\_layout()    plot\_file\_bar = f'total\_views\_bar\_chart\_{year\_to\_analyze}.png'  plt.savefig(plot\_file\_bar)  plt.close() # Fermer la figure  print(f"Graphique à barres sauvegardé dans : {plot\_file\_bar}")  # --- NOUVEAU GRAPHIQUE 2 : Graphique Circulaire (Top 10 + Autres) ---  print(f"Étape C ({year\_to\_analyze}) : Création du graphique circulaire (Top 10)...")    top\_10\_tags\_series = total\_views\_per\_tag.nlargest(10)    # Calculer la part "Autres"  sum\_top\_10 = top\_10\_tags\_series.sum()  sum\_all = total\_views\_per\_tag.sum()  sum\_other = sum\_all - sum\_top\_10    # Créer les données pour le camembert  pie\_data = top\_10\_tags\_series.copy()  if sum\_other > 0:  pie\_data['Autres'] = sum\_other    plt.figure(figsize=(10, 10))  # `autopct` affiche les pourcentages, `startangle` fait démarrer la première tranche en haut  pie\_data.plot(kind='pie', autopct='%1.1f%%', startangle=90,  wedgeprops={'edgecolor': 'white'},  textprops={'color':"black"})    plt.title(f'Répartition des "Media Views" ({year\_to\_analyze} - Top 10 + Autres)', pad=20)  plt.ylabel('') # Enlève le label de l'axe Y (qui est "Media Views" par défaut)  plt.axis('equal') # Assure que le graphique est un cercle parfait    plot\_file\_pie = f'views\_pie\_chart\_{year\_to\_analyze}.png'  plt.savefig(plot\_file\_pie)  plt.close() # Fermer la figure  print(f"Graphique circulaire sauvegardé dans : {plot\_file\_pie}")  # --- GRAPHIQUE 3 : Évolution (Ligne) (Code existant) ---  print(f"Étape D ({year\_to\_analyze}) : Création du graphique d'évolution (Top 10)...")    top\_10\_tags\_index = top\_10\_tags\_series.index  df\_top\_10\_agg = df\_agg[df\_agg['Assigned Tags Focus'].isin(top\_10\_tags\_index)]  # 5. Pivoter les données  df\_pivot = df\_top\_10\_agg.pivot(  index='Publication Date (YYYYMM)',  columns='Assigned Tags Focus',  values='Media Views'  ).fillna(0)  df\_pivot = df\_pivot.sort\_index()  # 6. Création et sauvegarde du graphique  if not df\_pivot.empty:  plt.figure(figsize=(18, 10))    df\_pivot.plot(kind='line', marker='o', ax=plt.gca())    plt.title(f'Évolution des "Media Views" par "Assigned Tags Focus" ({year\_to\_analyze} - Top 10)')  plt.xlabel('Date de publication (YYYYMM)')  plt.ylabel('Total Media Views')  plt.grid(True, linestyle='--', alpha=0.6)    plt.legend(title='Assigned Tags Focus', bbox\_to\_anchor=(1.05, 1), loc='upper left')    plt.xticks(rotation=45)  plt.tight\_layout()    plot\_file\_line = f'evolution\_media\_views\_{year\_to\_analyze}.png'  plt.savefig(plot\_file\_line)  plt.close() # Fermer la figure  print(f"Graphique d'évolution sauvegardé dans : {plot\_file\_line}")    else:  print(f"Aucune donnée d'évolution à visualiser pour {year\_to\_analyze}.")    print(f"--- Fin de l'analyse pour {year\_to\_analyze} ---")  # --- Fin de la fonction ---  # --- Script principal ---  # Nom du fichier fourni par l'utilisateur  file\_name = 'test Merge des 2 fichiers du 20251026 14h44.csv'  print(f"Chargement du fichier : {file\_name}")  try:  # Charger le fichier CSV avec le délimiteur point-virgule  df = pd.read\_csv(file\_name, delimiter=';')  print("\nÉtape 1 : Nettoyage et filtrage des données...")  # Filtrer les lignes où 'Assigned Tags Focus' n'est pas vide  df\_filtered = df[df['Assigned Tags Focus'].notna() & (df['Assigned Tags Focus'].str.strip() != '')].copy()    if df\_filtered.empty:  print("Aucune donnée trouvée avec un 'Assigned Tags Focus' non vide. L'analyse est arrêtée.")  else:  # Convertir 'Media Views' en numérique  df\_filtered['Media Views'] = pd.to\_numeric(df\_filtered['Media Views'], errors='coerce')  # Nettoyer la colonne de date (ex: '202504,0' -> '202504')  df\_filtered['Publication Date (YYYYMM)'] = df\_filtered['Publication Date (YYYYMM)'].astype(str).str.split(',').str[0]  df\_filtered['Publication Date (YYYYMM)'] = df\_filtered['Publication Date (YYYYMM)'].str.split('.').str[0]  # Supprimer les lignes où 'Media Views' n'a pas pu être converti  df\_filtered = df\_filtered.dropna(subset=['Media Views'])    print("Nettoyage des dates et des vues terminé.")  # --- Étape 2 : Extraire l'année ---  print("\nÉtape 2 : Extraction de l'année...")  df\_filtered['Year'] = df\_filtered['Publication Date (YYYYMM)'].str[:4]  print(f"Années trouvées dans les données : {df\_filtered['Year'].unique()}")  # --- ÉTAPE 3 : Séparation et explosion des tags ---  print("\nÉtape 3 : Séparation des tags...")    df\_exploded = df\_filtered.assign(  \*\*{'Assigned Tags Focus': df\_filtered['Assigned Tags Focus'].str.split(',')}  ).explode('Assigned Tags Focus')    df\_exploded['Assigned Tags Focus'] = df\_exploded['Assigned Tags Focus'].str.strip()  df\_exploded = df\_exploded[df\_exploded['Assigned Tags Focus'] != '']    print(f"Le nombre total de lignes (tags individuels) à analyser est : {len(df\_exploded)}")  # --- ÉTAPE 4 : Exécution des analyses par année ---    # Lancer l'analyse pour 2024  analyze\_year(df\_exploded, 2024)    # Lancer l'analyse pour 2025  analyze\_year(df\_exploded, 2025)  except FileNotFoundError:  print(f"Erreur : Le fichier '{file\_name}' n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur est survenue : {e}") |

#### Résultat



A graph with lines and dots

AI-generated content may be incorrect.

A graph of a number of people

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

A graph with blue squares

AI-generated content may be incorrect.A pie chart with different colored circles

AI-generated content may be incorrect.

A pie chart with different colored circles

AI-generated content may be incorrect.

## 5 colonnes boolean 1 ou 0 pour les tags

#### Script Python

|  |
| --- |
| import pandas as pd  import datetime # Ajout de l'import pour la date et l'heure  # Nom du fichier d'origine  file\_name = "Tags\_Measures\_Merged\_20251027\_0109.csv"  try:  # Charger le fichier CSV  df = pd.read\_csv(file\_name, delimiter=';')  # Définir la liste des thèmes cibles  themes = ['info', 'sport', 'musique', 'societe', 'humour']  # Boucler sur chaque thème pour créer la colonne  for theme in themes:  df[theme] = df['Assigned Tags'].str.contains(theme, case=False, na=False).astype(int)  # --- Création du nom de fichier dynamique ---    # 1. Obtenir la date et l'heure actuelles  now = datetime.datetime.now()    # 2. Formater la date et l'heure (ex: 20251027\_013148)  timestamp = now.strftime("%Y%m%d\_%H%M%S")    # 3. Créer le nom de fichier dynamique en utilisant un f-string  output\_file\_name = f"Tags\_Measures\_Pivoted\_Themes\_{timestamp}.csv"    # --- Fin de la modification ---  # Enregistrer le DataFrame modifié dans le nouveau fichier CSV  df.to\_csv(output\_file\_name, sep=';', index=False)    print(f"Traitement terminé. Fichier enregistré sous : '{output\_file\_name}'.")  except FileNotFoundError:  print(f"Erreur : Le fichier '{file\_name}' n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur est survenue lors du traitement : {e}") |

## Analyse tags par Shows

#### Script Python

|  |
| --- |
| import pandas as pd  # Nom du fichier fourni par l'utilisateur  file\_name = "Tags\_Measures\_Pivoted\_Themes\_20251027\_013230.csv"  try:  # Lire le fichier CSV en spécifiant le séparateur point-virgule  df = pd.read\_csv(file\_name, sep=';')  # Vérifier si les colonnes nécessaires existent  required\_columns = ['Show ID', 'Show', 'Assigned Tags Focus']    if all(col in df.columns for col in required\_columns):    # Compter les occurrences en ajoutant la colonne 'Show'  tag\_counts = df.value\_counts(['Show ID', 'Show', 'Assigned Tags Focus'])    # Renommer la série résultante pour plus de clarté  tag\_counts.name = 'Count'    print("Décompte des 'Assigned Tags Focus' par 'Show ID' et 'Show':")  # Afficher les résultats complets  with pd.option\_context('display.max\_rows', None, 'display.max\_columns', None):  print(tag\_counts)    else:  missing = [col for col in required\_columns if col not in df.columns]  print(f"Erreur : Les colonnes suivantes sont introuvables : {missing}")  except FileNotFoundError:  print(f"Erreur : Le fichier '{file\_name}' n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur est survenue : {e}") |

#### Résultat

A screenshot of a computer

AI-generated content may be incorrect.

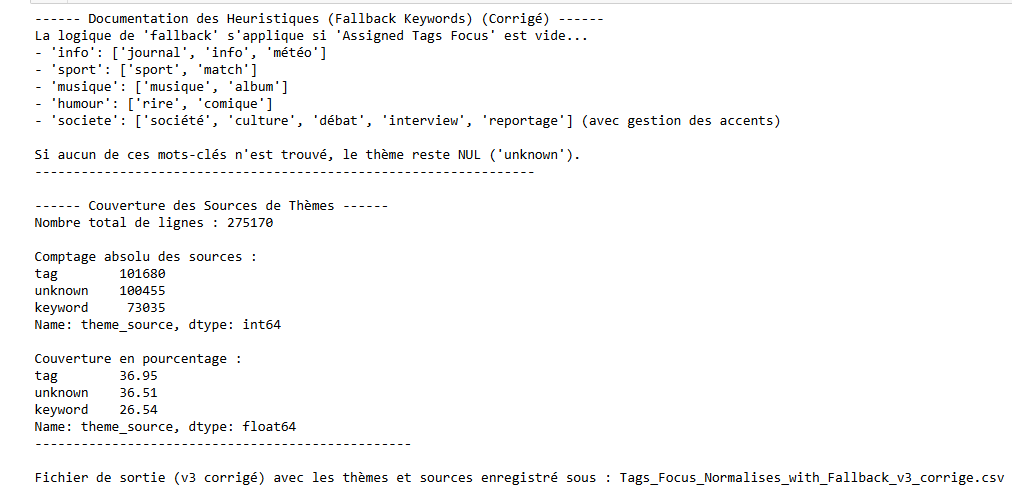
## Heuristique pour tags

#### KEYWORD\_HEURISTICS = { 'info': ['journal', 'info', 'météo', 'actualités', 'nouvelles', 'presse', 'politique', 'économie', 'faits divers', 'alerte', 'flash'], 'sport': ['sport', 'match', 'résultats', 'score', 'foot', 'tennis', 'basket', 'rugby', 'F1', 'championnat', 'coupe', 'JO', 'jeux olympiques', 'athlète', 'équipe'], 'musique': ['musique', 'album', 'chanson', 'clip', 'concert', 'live', 'artiste', 'chanteur', 'chanteuse', 'groupe', 'festival', 'playlist', 'single'], 'humour': ['rire', 'comique', 'drôle', 'blague', 'sketch', 'parodie', 'stand-up', 'humoriste', 'spectacle', 'bêtisier'], 'societe': ['société', 'societe', 'culture', 'débat', 'interview', 'reportage', 'documentaire', 'enquête', 'entretien', 'social', 'environnement', 'histoire', 'éducation', 'podcast'] }

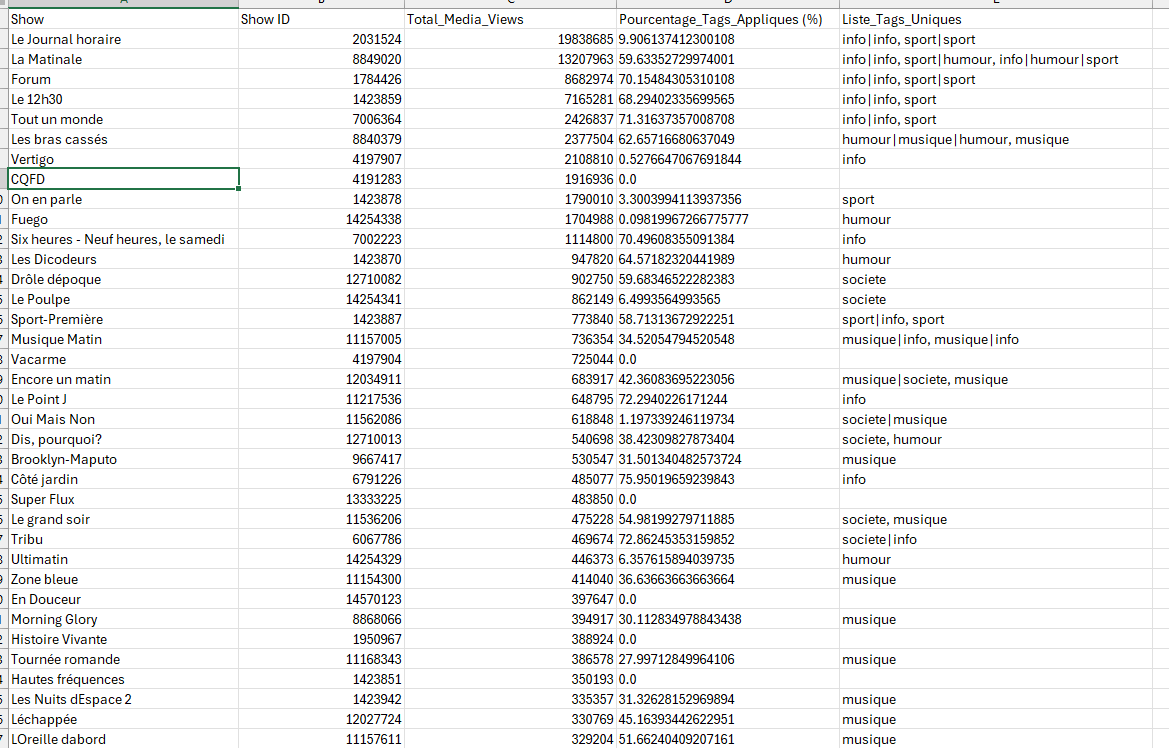
#### Script Python

|  |
| --- |
| import pandas as pd  import numpy as np  import re  # Fonction pour appliquer la logique de mots-clés (inchangée)  def apply\_keyword\_logic(row):  """  Applique une logique de mots-clés sur les colonnes 'Show' et 'Segment'.  Retourne un thème ('info', 'sport', 'musique', 'humour', 'societe') ou np.nan  """  show\_text = str(row['Show']).lower() if pd.notna(row['Show']) else ''  segment\_text = str(row['Segment']).lower() if pd.notna(row['Segment']) else ''  combined\_text = show\_text + " " + segment\_text  keywords = {  'info': [r'\bjournal\b', r'\binfo\b', 'météo'],  'sport': [r'\bsport\b', r'\bmatch\b'],  'musique': [r'\bmusique\b', r'\balbum\b'],  'humour': [r'\brire\b', r'\bcomique\b'],  'societe': [r'\bsoci[eé]t[eé]\b', r'\bculture\b', r'\bd[eé]bat\b', r'\binterview\b', r'\breportage\b']  }  for theme, kws in keywords.items():  if any(re.search(kw, combined\_text) for kw in kws):  return theme    return np.nan  # ----- Chargement des données -----  file\_name = "Tags\_Focus\_Normalises\_20251027\_231733.csv"  try:  df = pd.read\_csv(file\_name, sep=';')  # ----- Implémentation de la hiérarchie des thèmes -----  # 1. Initialiser les nouvelles colonnes  df['theme'] = np.nan  df['theme\_source'] = 'unknown'  # 2. Règle 1: "Assigned Tags Focus"  df['Assigned Tags Focus'] = df['Assigned Tags Focus'].replace('', np.nan)  condition\_tag = df['Assigned Tags Focus'].notna()  df.loc[condition\_tag, 'theme'] = df['Assigned Tags Focus']  df.loc[condition\_tag, 'theme\_source'] = 'tag'  # 3. Règle 2: "Show Mapping" (Espace réservé)  # ...  # 4. Règle 3: "Fallback Keywords"  # Sélectionner les lignes où le thème n'a pas encore été défini  condition\_keyword\_fallback = df['theme'].isna()    # Obtenir le sous-ensemble du DataFrame où le fallback s'applique  df\_fallback = df[condition\_keyword\_fallback]    if not df\_fallback.empty:  # Appliquer la fonction 'apply\_keyword\_logic' sur ce sous-ensemble  # keyword\_themes aura le MÊME index que df\_fallback (un sous-ensemble de l'index de df)  keyword\_themes = df\_fallback.apply(apply\_keyword\_logic, axis=1)    # Trouver les lignes (dans ce sous-ensemble) où un thème a été trouvé  condition\_keyword\_found = keyword\_themes.notna()    # Obtenir la série de thèmes qui ont été trouvés (non nuls)  # L'index de 'themes\_to\_apply' est un sous-ensemble de l'index de keyword\_themes  themes\_to\_apply = keyword\_themes[condition\_keyword\_found]    # Appliquer ces thèmes et la source au DataFrame original  # en utilisant l'index de 'themes\_to\_apply'  # C'est la correction clé : on utilise themes\_to\_apply.index  if not themes\_to\_apply.empty:  df.loc[themes\_to\_apply.index, 'theme'] = themes\_to\_apply  df.loc[themes\_to\_apply.index, 'theme\_source'] = 'keyword'    # Les lignes où keyword\_themes était np.nan garderont  # df['theme'] = np.nan et df['theme\_source'] = 'unknown' (défini à l'étape 1)  # ----- Documentation et Couverture -----    print("------ Documentation des Heuristiques (Fallback Keywords) (Corrigé) ------")  print("La logique de 'fallback' s'applique si 'Assigned Tags Focus' est vide...")  print("- 'info': ['journal', 'info', 'météo']")  print("- 'sport': ['sport', 'match']")  print("- 'musique': ['musique', 'album']")  print("- 'humour': ['rire', 'comique']")  print("- 'societe': ['société', 'culture', 'débat', 'interview', 'reportage'] (avec gestion des accents)")  print("\nSi aucun de ces mots-clés n'est trouvé, le thème reste NUL ('unknown').")  print("-----------------------------------------------------------------")    print("\n------ Couverture des Sources de Thèmes ------")  total\_rows = len(df)  print(f"Nombre total de lignes : {total\_rows}\n")    coverage\_counts = df['theme\_source'].value\_counts(dropna=False)  print("Comptage absolu des sources :")  print(coverage\_counts)    coverage\_perc = (df['theme\_source'].value\_counts(normalize=True, dropna=False) \* 100).round(2)  print("\nCouverture en pourcentage :")  print(coverage\_perc)  print("-------------------------------------------------")    # ----- Sauvegarde des résultats -----  output\_file = "Tags\_Focus\_Normalises\_with\_Fallback\_v3\_corrige.csv"  df.to\_csv(output\_file, sep=';', index=False, encoding='utf-8-sig')    print(f"\nFichier de sortie (v3 corrigé) avec les thèmes et sources enregistré sous : {output\_file}")  except FileNotFoundError:  print(f"Erreur : Le fichier {file\_name} n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur est survenue : {e}") |

#### Résultat



# Suppositions sur les tags



Tous les Show ID qui ont 30% du même tag, on met tout ce tag dans les autres show ID.

Code Python

|  |
| --- |
| import pandas as pd  import numpy as np  # Chemin du fichier d'entrée  file\_path = "Tags\_Focus\_Normalises\_with\_Fallback\_final (1).csv"  # Chemin du fichier de sortie  output\_file\_path = "Tags\_Focus\_Normalises\_with\_Fallback\_final\_updated.csv"  try:  # 1. Charger les données  # Assurez-vous que le séparateur est correct  df = pd.read\_csv(file\_path, sep=';')  print(f"Données chargées. Forme initiale : {df.shape}")  print(f"Colonnes détectées : {df.columns.tolist()}")  # Vérifier si les colonnes nécessaires existent  if 'Show ID' not in df.columns or 'theme\_source' not in df.columns:  print("Erreur : Les colonnes 'Show ID' ou 'theme\_source' sont introuvables.")  print("Veuillez vérifier les noms exacts des colonnes dans votre fichier.")  else:  # 2. Standardiser les valeurs 'unknown' en NaN pour un traitement facile  # Nous créons une copie temporaire pour l'analyse  df\_temp = df.copy()  df\_temp['theme\_source\_clean'] = df\_temp['theme\_source'].replace('unknown', np.nan)  # 3. Calculer le thème dominant pour chaque 'Show ID'  show\_theme\_map = {}  # Grouper par 'Show ID'  grouped = df\_temp.groupby('Show ID')  for show\_id, group in grouped:  total\_count = len(group)  if total\_count == 0:  continue  # 4. Compter les thèmes valides (non-NaN, non-'unknown')  valid\_theme\_counts = group['theme\_source\_clean'].value\_counts()  if not valid\_theme\_counts.empty:  # 5. Trouver le thème le plus fréquent  top\_theme = valid\_theme\_counts.idxmax()  top\_theme\_count = valid\_theme\_counts.max()  # 6. Vérifier si sa proportion atteint 30% du \*total\*  proportion = top\_theme\_count / total\_count  if proportion >= 0.30:  # Si oui, enregistrer ce thème pour ce Show ID  show\_theme\_map[show\_id] = top\_theme  print(f"Trouvé {len(show\_theme\_map)} 'Show ID' avec un thème dominant (>= 30%).")  # 7. Appliquer la correspondance au DataFrame original    # Créer une série de thèmes mappés (aura NaN si le Show ID n'est pas dans la map)  mapped\_themes = df['Show ID'].map(show\_theme\_map)  # 8. Identifier les lignes à remplir (NaN ou 'unknown')  is\_missing\_or\_unknown = df['theme\_source'].isnull() | (df['theme\_source'] == 'unknown')    # Masque pour les lignes qui sont à la fois "missing/unknown" ET dont le 'Show ID' a un thème dominant  rows\_to\_update\_mask = is\_missing\_or\_unknown & mapped\_themes.notnull()  num\_rows\_to\_update = rows\_to\_update\_mask.sum()    print(f"Mise à jour de {num\_rows\_to\_update} lignes où 'theme\_source' était 'unknown' ou manquant.")  # 9. Remplacer les valeurs  # Utiliser mask(): là où la condition (rows\_to\_update\_mask) est Vraie,  # remplacer la valeur de 'theme\_source' par la valeur de 'mapped\_themes'.  df['theme\_source'] = df['theme\_source'].mask(rows\_to\_update\_mask, mapped\_themes)  # 10. Sauvegarder le fichier modifié  df.to\_csv(output\_file\_path, sep=';', index=False)  print(f"Fichier mis à jour et sauvegardé sous : {output\_file\_path}")    # Afficher un aperçu des changements pour un Show ID affecté (si possible)  if not mapped\_themes.isnull().all():  example\_show\_id = mapped\_themes.dropna().index[0]  example\_show\_id = df.loc[example\_show\_id, 'Show ID']  print(f"\nAperçu des changements pour un Show ID affecté (par ex. '{example\_show\_id}'):")  # Montrer les anciennes valeurs "unknown" et les nouvelles  print(df[df['Show ID'] == example\_show\_id]['theme\_source'].value\_counts(dropna=False))  except FileNotFoundError:  print(f"Erreur : Le fichier '{file\_path}' n'a pas été trouvé.")  except Exception as e:  print(f"Une erreur est survenue : {e}") |

CQFD : société partout.

Crimes Suisse : dans aucun tag. J’ai vu que ça marchait bien mais je ne sais pas dans quel tag le mettre.

## Résultat final après configuration des tags

275171 lignes en tout

226467 lignes avec tag et 48705 sans tag (soit 17.7% sans tag)

# Phase 2 : Développement du Cadre Analytique (Acquisition, Rétention, Fidélisation)

Le succès du projet repose sur la capacité à relier les métriques brutes aux concepts commerciaux d'Acquisition, de Rétention et de Fidélisation.

J'utiliserai les indicateurs suivants pour définir la stratégie de chaque thème :

|  |  |  |
| --- | --- | --- |
| Concept Stratégique | Métrique Clé | Définition/Justification (basée sur les données) |
| **Acquisition** (Nouveaux Publics) | **New Visit Rate %** (Taux de nouvelles visites) | Représente la proportion de visites provenant de nouveaux utilisateurs. Un taux élevé indique une bonne performance en acquisition. |
|  | **Entries** (Entrées) | Mesure le nombre de visites dont la première page vue correspond à la lecture du segment. Un nombre élevé d'entrées suggère que le contenu est un point d'accès initial (acquisition). |
| **Rétention/Fidélisation** | **Returning Visits** (Visites de retour) | Indique le nombre de visites générées par des visiteurs déjà connus dans la base de données, signalant la fidélité et le retour du public. |
|  | **Avg Play Duration** (Durée moyenne de lecture) | Représente la qualité de l'engagement (temps de consommation moyen par vue média). Une durée élevée signale un contenu engageant qui retient l'utilisateur. |
|  | **Bounces** (Rebonds) | Le nombre de visites avec une seule page vue. Un faible taux de rebond est souvent un indicateur de rétention, car les utilisateurs continuent d'explorer. |

# Phase 3 : Exécution et Analyse

1. **Analyse des Tendances et Performances :** J'analyserais les performances passées (disponibles de 2024 à juin 2025) en agrégeant les métriques de la Phase 2 pour chacun des cinq thèmes.

2. **Cartographie Stratégique :** Je positionnerais chaque thème (Info, Sport, Musique, Société, Humour) sur une matrice bi-dimensionnelle (par exemple, Acquisition vs. Rétention/Engagement) pour visualiser son rôle stratégique.

    ◦ *Exemple d'axe Acquisition :* Taux de Nouvelles Visites ou Entrées.

    ◦ *Exemple d'axe Rétention/Engagement :* Durée Totale de Lecture ou Durée Moyenne de Lecture.

# Plan

## 2 — Nettoyage détaillé & transform classiques (scripts SQL / Python)

**Points importants**

* Convertis avg\_play\_seconds en fraction du segment\_length\_seconds pour calculer % d’écoute par média view :  
  pct\_consumed = avg\_play\_seconds / segment\_length\_seconds (clip entre 0 et 1).
* Si avg\_play\_seconds > segment\_length\_seconds, clip à 1 (données bruitées).
* Uniformiser App/Site Name (ex. rts-app-play / rts-app-info / rts.ch) en catégories fixes.

L'objectif ici est de s'assurer que les données sont fiables, complètes et dans un format **Nettoyer mesures\_df :**

* + **Dates :** Convertir Publication Date en objet datetime.
  + **Durées (Problème Majeur) :** Les colonnes Avg Play Duration et Total Play Duration sont au format hh:mm:ss. Elles sont inutilisables pour des calculs. Il faut les convertir en secondes (ou minutes).
  + **Pourcentages :** La colonne New Visit Rate % est un texte (ex: 84,56%). Il faut la nettoyer : remplacer la virgule par un point, enlever le % et convertir en *float* (ex: 0.8456).
  + **Nombres :** S'assurer que Media Views, Visitors, Entries, etc., sont bien des types numériques (int ou float).

## 3 — Construction du dataset final (modèle analytique)

But : table des épisodes (grain = Segment ID × publication\_date × app\_site) avec colonnes normalisées.

**Champs recommandés**

* segment\_id (string)
* show\_id, show\_title
* publication\_date (date)
* platform (rts.ch / rts-app-play / rts-app-info / rts-app-sport)
* device\_class (Smartphone / PC / ...)
* segment\_length\_s (int)
* media\_views (int)
* .03
* avg\_play\_s (float)
* total\_play\_s (float)
* visitors, entries, exits, returning\_visits, bounces (int)
* new\_visit\_rate (float)
* assigned\_tags (string) — from TagsV0 / assigned tags raw
* main\_theme (categorical) — one of the five themes (voir mapping)
* pct\_consumed = avg\_play\_s / segment\_length\_s
* views\_per\_visitor = media\_views / visitors (handle div0)
* entry\_rate = entries / visitors
* returning\_rate = returning\_visits / visits (or returning\_visits / visitors) — définir et documenter\_

## 4 — Attribution des *main themes* (règles pratiques)

1. **Fallback show-level mapping** : si une Show est majoritairement associée à un thème (ex. Le Journal horaire → info, Crimes suisses → societe ou humour?), construire un référentiel show\_id -> theme en agrégeant les épisodes taggés. Règle : si > 70% des épisodes taggés d’un show ont le même thème, assigner ce thème au show entier.
2. **Fallback keywords** : si pas de tags ni show mapping, appliquer une règle simple de mot-clé sur show ou segment (ex. mots « journal », « info », « météo » → info; « sport », « match » → sport; « musique », « album » → musique; « rire », « comique » → humour; else societe). Documenter ces heuristiques et leur couverture.
3. **Marquer les confiances** : theme\_source = tag / show\_mapping / keyword / unknown.

## 5 — Définir métriques KPI et mapping métier (Acquisition / Rétention / Loyalty)

Définitions proposées (pratiques et mesurables) — tu peux ajuster les pondérations.

* **5.1 Métriques brutes**
* **Media Views** (MV)
* **Visitors** (V)
* **New Visit Rate** (NVR) — proportion de visites nouvelles
* **Entries** (E) — proportion d’entrées directes (first page = play)
* **Returning Visits** (RV)
* **Avg Play Seconds** (APS)
* **Pct Consumed** (PC) = APS / segment\_length\_s
* **Total Play Seconds** (TPS)
* **Views per Visitor** (VPV) = MV / V
* **5.2 Construction de scores thème-par-période**

Calcule par thème (et par mois) les agrégats : sum(MV), sum(V), weighted avg(PC), sum(E), sum(RV), mean(NVR) pondérée par visites, etc.

1. **Score Acquisition (objectif : attirer de nouveaux utilisateurs)**

Proposition :

Acquisition\_score\_theme = w1 \* normalized(new\_visitors\_ratio) + w2 \* normalized(entries\_per\_visitor) + w3 \* normalized(mean\_new\_visit\_rate)

avec par exemple w1=0.5, w2=0.3, w3=0.2.

* new\_visitors\_ratio = sum(entries\_from\_new\_visitors) / sum(visitors) (approx via new\_visit\_rate \* visitors if granularité manque).
* Normalisation : min-max sur la période d’analyse (0..1).

1. **Score Retention (objectif : ramener les utilisateurs)**

Proposition :

Retention\_score\_theme = a1 \* normalized(returning\_visits\_rate) + a2 \* normalized(views\_per\_visitor) + a3 \* normalized(repeat\_sessions)

avec returning\_visits\_rate = sum(returning\_visits)/sum(visits).

1. **Score Loyalty (fidélisation / engagement profond)**

Proposition :

Loyalty\_score\_theme = b1 \* normalized(avg\_pct\_consumed) + b2 \* normalized(total\_play\_seconds\_per\_visitor) + b3 \* normalized(bounce\_inverse)

* total\_play\_seconds\_per\_visitor = sum(total\_play\_seconds) / sum(visitors).
* bounce\_inverse = 1 - (bounces / entries) pour favoriser faibles bounces.

1. **Scoring pratique (normalisation)**

* Pour chaque métrique, faire normalized = (x - x\_min) / (x\_max - x\_min).
* Calculer les scores par thème et pondérer comme ci-dessus. Documenter les pondérations.

Questions business en métriques chiffrées.

Proposition de framework :

1. **Thème d'Acquisition (Attirer de nouveaux visiteurs)**

Un thème performant en acquisition...

* ...a un **New Visit Rate % élevé** : C'est la métrique la plus directe.
* ...génère beaucoup d'**Entries** : Le contenu est la porte d'entrée principale sur la plateforme.
* ...a un nombre de **Visitors** (uniques) élevé, combiné à un New Visit Rate % élevé.

1. **Thème de Rétention & Fidélisation (Faire revenir et engager)**

Un thème performant en rétention...

* ...génère beaucoup de **Returning Visits** : C'est la métrique la porte-drapeau de la fidélisation.
* ...a un **Avg Play Duration** élevé. Pour le normaliser, je vous suggère de créer une métrique :
  + **Taux de Complétion** = Avg Play Duration (en sec) / Segment Length (en sec). Un taux élevé signifie que le contenu est engageant et apprécié par les fidèles.
* ...a un **Total Play Duration** élevé : Les utilisateurs fidèles consomment en volume.
* ...a un faible taux de **Bounces** (Rebonds) : Les utilisateurs ne viennent pas juste pour un contenu et repartent, ils restent sur la plateforme.

## 6 — Analyses recommandées (questions à produire / visuals)

Pour répondre aux besoins stratégiques, produire au moins les analyses suivantes :

1. **Vue synthétique par thème** (KPIs): tableau montrant pour chaque thème :
   * media\_views, visitors, new\_visit\_rate, entries, returning\_visits, avg\_pct\_consumed, Acquisition\_score, Retention\_score, Loyalty\_score, production\_minutes (si dispo).
   * Classement par score d’acquisition / rétention / fidélité.
2. **Tendance temporelle** : courbes mensuelles par thème pour media\_views, visitors, pct\_consumed, new\_visit\_rate.
3. **Funnel / Conversion** par thème : visitors → entries → media\_views → avg\_play\_seconds (visuel type funnel ou sankey).
4. **Cohorte** (si possible) : cohortes par mois de premières visites, suivi des returning\_rate 1,2,3 mois plus tard (utile pour mesurer rétention).
5. **Breakdown device / platform** : quels thèmes performent mieux sur rts.ch vs rts-app-play vs rts-app-info ; smartphone vs PC.
6. **Top shows / top episodes** : liste des meilleurs contributeurs à acquisition/retention/loyalty.
7. **Qualité données & coverage** : carte de chaleur montrant où les tags manquent (coverage tags par mois / platform).
8. **Construire le Dashboard (ex: Power BI) :**
   * **Page 1 : Vue d'ensemble (Réponse à la Priorisation)**
     + **Graphique Principal :** Diagramme en barres "Volume par Thème".
       - Vous avez deux options pour le "volume" :
         1. sum(Media Views) : Popularité (combien de clics).
         2. sum(Total Play Duration) : Engagement (combien de temps passé).
     + **KPIs Clés :** Cartes affichant le Total de Vues, Total de Visiteurs, Total de Temps de Lecture pour les 5 thèmes combinés.
   * **Page 2 : Analyse Stratégique (Acquisition vs. Rétention)**
     + **Graphique Principal : Matrice 2x2 (Scatter Plot)**
       - **Axe Y :** Métrique de Rétention (ex: Moyenne(Taux de Complétion) ou Sum(Returning Visits))
       - **Axe X :** Métrique d'Acquisition (ex: Moyenne(New Visit Rate %))
       - 5 thèmes sur ce graphique. Cela créera 4 quadrants :
         1. **Stars (Haut/Haut) :** Bons pour l'acquisition ET la rétention.
         2. **Moteurs d'Acquisition (Haut/Bas) :** Attire, mais ne retient pas.
         3. **Piliers de Fidélité (Bas/Haut) :** Contenu pour la base fidèle.
         4. **Sous-performants (Bas/Bas) :** A réévaluer.
     + **Graphiques secondaires :**
       - Barres "Taux de Nouveaux Visiteurs Moyen par Thème" (Acquisition).
       - Barres "Nombre de Visites Récurrentes par Thème" (Rétention).
   * **Page 3 : Contexte et Limites (Pour montrer votre rigueur analytique)**
     + Un graphique (ex: Donut) montrant la proportion des données que vous avez analysées :
       - Données Taguées (>= Jan 2025 & rts.ch) vs. Données Non-Taguées (le reste).
     + Cela montre que vous comprenez les limites de votre analyse et que vos conclusions sont basées sur un sous-ensemble spécifique.

## 7 — Power BI : modèle & pages (proposition de wireframe)

**Modèle de données**

* Table Episodes (fact table) : dataset nettoyé (cf §3).
* Dimensions : Dim\_Date (date, month, quarter), Dim\_Show (show\_id, title, default\_theme), Dim\_Platform, Dim\_Device, Dim\_Theme (5 thèmes).
* Relations : Episodes.show\_id → Dim\_Show ; Episodes.main\_theme → Dim\_Theme.

**Visuels clés**

* Cartes KPI, stacked area (trend), clustered bar (comparaisons), funnel chart, matrix table, scatter plot (acquisition\_score vs loyalty\_score), table with conditional formatting (top episodes).