Stage 1 - Data Exploration and Preprocessing *(Fixed)*

Diego Armando Salinas Lugo | Student ID: ds24353 / 2401168

Purpose (Fixing Stage 1):

The aim of this notebook is to preprocess the datasets provided for ADHD analysis. The objective is to create a well-structured and clean dataset that can be used in Stage 2 for developing predictive models for both ADHD outcome and Sex.

This process involves:

- Loading and inspecting datasets.
- Splitting the data into training and validation sets.
- Handling missing values appropriately.
- Performing feature selection and dimensionality reduction where required.
- Saving the prepared datasets for Stage 2 modeling.

In [20]:
```
# Importing necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import KNNImputer, IterativeImputer
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import KernelPCA
from sklearn.feature_selection import mutual_info_classif
import warnings
warnings.filterwarnings('ignore')
```

Step 1: Loading the Datasets

Datasets provided for analysis:

- `labels.xlsx` - Contains participant identifiers along with ADHD outcome and Sex.
- `metadata_a.xlsx` - Contains numerical metadata for each participant.
- `metadata_b.xlsx` - Contains categorical metadata for each participant.

- `connectome_matrices.csv` - Contains brain functional connectome matrices.

In [21]:
```
# Loading datasets
labels = pd.read_excel(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\DataScience\Exploration\ce888_data_2025\ce888_data_
metadata_a = pd.read_excel(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\DataScience\Exploration\ce888_data_2025\ce888_c
metadata_b = pd.read_excel(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\DataScience\Exploration\ce888_data_2025\ce888_c
connectome_matrices = pd.read_csv(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\DataScience\Exploration\ce888_data_2025\
```

In [22]:
```
# Confirming datasets are loaded correctly
print("Labels dataset shape:", labels.shape)
print("Metadata A dataset shape:", metadata_a.shape)
print("Metadata B dataset shape:", metadata_b.shape)
print("Connectome matrices dataset shape:", connectome_matrices.shape)
```

```
Labels dataset shape: (1213, 3)
Metadata A dataset shape: (1213, 19)
Metadata B dataset shape: (1213, 10)
Connectome matrices dataset shape: (1213, 19901)
```

Step 2: Initial Inspection and Dataset Alignment

The `participant_id` column is the common key across all datasets. This key will be used later for merging the datasets.

Before proceeding with splitting and cleaning, the datasets are inspected for consistency in `participant_id`.

In [23]:
```
# Checking common columns for merging later
common_key = set(labels.columns) & set(metadata_a.columns) & set(metadata_b.columns) & set(connectome_matrices.columr
print("Common key for merging:", common_key)

# Preview of participant IDs in each dataset
print("Unique participants in labels:", labels['participant_id'].nunique())
print("Unique participants in metadata_a:", metadata_a['participant_id'].nunique())
print("Unique participants in metadata_b:", metadata_b['participant_id'].nunique())
print("Unique participants in connectome_matrices:", connectome_matrices['participant_id'].nunique())
```

```
Common key for merging: {'participant_id'}
Unique participants in labels: 1213
Unique participants in metadata_a: 1213
Unique participants in metadata_b: 1213
Unique participants in connectome_matrices: 1213
```

Step 3: Splitting the Data into Training and Validation Sets

The datasets will be splitted into training and validation subsets to prepare for modeling.
Splitting will be stratified by both labels (`ADHD_Outcome` and `Sex_F`) to ensure balanced class representation.

Stratification Approach

As recommended in the assignment feedback, the two labels will be combined into a single categorical variable representing all four possible class combinations:

- `0_0` → No ADHD, Male
- `0_1` → No ADHD, Female
- `1_0` → ADHD, Male
- `1_1` → ADHD, Female

In [24]:
```
# Creating a combined label for stratification
labels['combined_label'] = labels['ADHD_Outcome'].astype(str) + "_" + labels['Sex_F'].astype(str)

# Performing the stratified split
train_ids, val_ids = train_test_split(
    labels['participant_id'],
    test_size=0.2,
    stratify=labels['combined_label'],
    random_state=42
)

# Confirming split sizes
print("Number of participants in training set:", len(train_ids))
print("Number of participants in validation set:", len(val_ids))
```

```
Number of participants in training set: 970
Number of participants in validation set: 243
```

In [25]:
```
# Creating training and validation sets for each dataset
train_labels = labels[labels['participant_id'].isin(train_ids)].reset_index(drop=True)
val_labels = labels[labels['participant_id'].isin(val_ids)].reset_index(drop=True)

train_metadata_a = metadata_a[metadata_a['participant_id'].isin(train_ids)].reset_index(drop=True)
val_metadata_a = metadata_a[metadata_a['participant_id'].isin(val_ids)].reset_index(drop=True)
```

```
train_metadata_b = metadata_b[metadata_b['participant_id'].isin(train_ids)].reset_index(drop=True)
val_metadata_b = metadata_b[metadata_b['participant_id'].isin(val_ids)].reset_index(drop=True)

train_connectome = connectome_matrices[connectome_matrices['participant_id'].isin(train_ids)].reset_index(drop=True)
val_connectome = connectome_matrices[connectome_matrices['participant_id'].isin(val_ids)].reset_index(drop=True)

# Confirming datasets are aligned after the split
print("Training labels shape:", train_labels.shape)
print("Validation labels shape:", val_labels.shape)

print("Training Metadata A shape:", train_metadata_a.shape)
print("Validation Metadata A shape:", val_metadata_a.shape)

print("Training Metadata B shape:", train_metadata_b.shape)
print("Validation Metadata B shape:", val_metadata_b.shape)

print("Training Connectome shape:", train_connectome.shape)
print("Validation Connectome shape:", val_connectome.shape)
```

```
Training labels shape: (970, 4)
Validation labels shape: (243, 4)
Training Metadata A shape: (970, 19)
Validation Metadata A shape: (243, 19)
Training Metadata B shape: (970, 10)
Validation Metadata B shape: (243, 10)
Training Connectome shape: (970, 19901)
Validation Connectome shape: (243, 19901)
```

Step 4: Preprocessing Metadata A (Numerical Data)

4.1 Handling Missing Values

The column `MRI_Track_Age_at_Scan` contains approximately 30% missing values. According to the feedback, an **Iterative Imputer** is recommended instead of using the mean or median. The imputer will be fit on the training data and then applied to the validation set to avoid data leakage.

In [ ]:
```
# Handling Missing Values with IterativeImputer
iterative_imputer = IterativeImputer(random_state=42)

# Selecting columns to impute (excluding participant_id)
```

```
columns_to_impute = train_metadata_a.drop(columns='participant_id').columns

# Fitting on training data
train_metadata_a_imputed = iterative_imputer.fit_transform(train_metadata_a[columns_to_impute])

# Transforming validation data
val_metadata_a_imputed = iterative_imputer.transform(val_metadata_a[columns_to_impute])

# Converting back to DataFrame and restoring participant_id
train_metadata_a_clean = pd.DataFrame(train_metadata_a_imputed, columns=columns_to_impute)
train_metadata_a_clean['participant_id'] = train_metadata_a['participant_id']

val_metadata_a_clean = pd.DataFrame(val_metadata_a_imputed, columns=columns_to_impute)
val_metadata_a_clean['participant_id'] = val_metadata_a['participant_id']

# Confirming good executing
print("Missing values in train_metadata_a_clean:", train_metadata_a_clean.isnull().sum().sum())
print("Missing values in val_metadata_a_clean:", val_metadata_a_clean.isnull().sum().sum())
# Confirming shapes
print("Training Metadata A after imputation:", train_metadata_a_clean.shape)
print("Validation Metadata A after imputation:", val_metadata_a_clean.shape)
```

```
Missing values in train_metadata_a_clean: 0
Missing values in val_metadata_a_clean: 0
Training Metadata A after imputation: (970, 19)
Validation Metadata A after imputation: (243, 19)
```

4.2 Feature Selection

Feature selection will be performed using **mutual information** between each numerical feature and the target label `ADHD_Outcome`. Only the features with a mutual information score above a defined threshold will be retained.

```
In [ ]: # Calculating mutual information scores between features and ADHD_Outcome in training set
mi_scores = mutual_info_classif(train_metadata_a_clean.drop(columns='participant_id'),
                                train_labels['ADHD_Outcome'],
                                discrete_features=False,
                                random_state=42)

# Creating a DataFrame to view scores
mi_scores_df = pd.DataFrame({
    'Feature': train_metadata_a_clean.drop(columns='participant_id').columns,
    'MI_Score': mi_scores
}).sort_values(by='MI_Score', ascending=False)

# Displaying mutual information scores
mi_scores_df
```

Out[ ]:

	Feature	MI_Score
13	SDQ_SDQ_Hyperactivity	0.163527
11	SDQ_SDQ_Externalizing	0.154282
9	SDQ_SDQ_Difficulties_Total	0.127550
12	SDQ_SDQ_Generating_Impact	0.112013
8	SDQ_SDQ_Conduct_Problems	0.045341
10	SDQ_SDQ_Emotional_Problems	0.028967
3	APQ_P_APQ_P_ID	0.024279
15	SDQ_SDQ_Peer_Problems	0.022525
14	SDQ_SDQ_Internalizing	0.018284
4	APQ_P_APQ_P_INV	0.014491
16	SDQ_SDQ_Prosocial	0.012048
5	APQ_P_APQ_P_OPD	0.005478
1	ColorVision_CV_Score	0.004532
7	APQ_P_APQ_P_PP	0.002890
0	EHQ_EHQ_Total	0.000000
2	APQ_P_APQ_P_CP	0.000000
6	APQ_P_APQ_P_PM	0.000000
17	MRI_Track_Age_at_Scan	0.000000

In [ ]:
```
# Selecting features with MI score above a threshold (MI > 0.01), to work with the more significant features
selected_features = mi_scores_df[mi_scores_df['MI_Score'] > 0.01]['Feature'].tolist()

print("Selected features based on mutual information:", selected_features)

# Extracting only the selected features (excluding participant_id)
X_train_a = train_metadata_a_clean[selected_features]
X_val_a = val_metadata_a_clean[selected_features]

# Standardising numerical features using StandardScaler
# This is important because selected Metadata A features are numerical and have different scales.
scaler_a = StandardScaler()
X_train_a_scaled = scaler_a.fit_transform(X_train_a)
X_val_a_scaled = scaler_a.transform(X_val_a)

# Reconverting scaled arrays to DataFrames with original feature names
X_train_a_scaled_df = pd.DataFrame(X_train_a_scaled, columns=selected_features, index=train_metadata_a_clean.index)
X_val_a_scaled_df = pd.DataFrame(X_val_a_scaled, columns=selected_features, index=val_metadata_a_clean.index)

# Reattaching participant_id
train_metadata_a_selected = pd.concat([X_train_a_scaled_df, train_metadata_a_clean[['participant_id']]], axis=1)
val_metadata_a_selected = pd.concat([X_val_a_scaled_df, val_metadata_a_clean[['participant_id']]], axis=1)

# Confirming final shapes after selection and scaling
print("Training Metadata A after feature selection + scaling:", train_metadata_a_selected.shape)
print("Validation Metadata A after feature selection + scaling:", val_metadata_a_selected.shape)
```

```
Selected features based on mutual information: ['SDQ_SDQ_Hyperactivity', 'SDQ_SDQ_Externalizing', 'SDQ_SDQ_Difficulti
es_Total', 'SDQ_SDQ_Generating_Impact', 'SDQ_SDQ_Conduct_Problems', 'SDQ_SDQ_Emotional_Problems', 'APQ_P_APQ_P_ID',
'SDQ_SDQ_Peer_Problems', 'SDQ_SDQ_Internalizing', 'APQ_P_APQ_P_INV', 'SDQ_SDQ_Prosocial']
Training Metadata A after feature selection + scaling: (970, 12)
Validation Metadata A after feature selection + scaling: (243, 12)
```

In [ ]:
```
train_metadata_a_selected.head()
```

Out[ ]:

	SDQ_SDQ_Hyperactivity	SDQ_SDQ_Externalizing	SDQ_SDQ_Difficulties_Total	SDQ_SDQ_Generating_Impact	SDQ_SDQ_Conduct_I
0	0.888748	0.112626	0.876890	1.043100	-
1	0.541133	0.112626	0.281660	0.336531	-
2	1.583979	1.996193	1.769735	1.749669	
3	1.583979	0.818963	0.876890	-0.016754	-
4	0.888748	1.054409	0.728083	1.749669	

In [31]:
```python
val_metadata_a_selected.head()
```

Out[31]:

	SDQ_SDQ_Hyperactivity	SDQ_SDQ_Externalizing	SDQ_SDQ_Difficulties_Total	SDQ_SDQ_Generating_Impact	SDQ_SDQ_Conduct_I
0	-0.154098	-0.593712	-0.908800	-1.429892	-
1	0.888748	0.818963	0.579275	0.689815	
2	0.888748	1.054409	0.728083	1.043100	
3	-0.501713	-0.829158	-0.313570	-0.016754	-
4	0.193518	0.583517	0.728083	0.336531	

Step 5: Preprocessing Metadata B (Categorical Data)

5.1 Handling Missing Values

The column `PreInt_Demos_Fam_Child_Ethnicity` contains missing values. According to the provided data dictionary and assignment feedback, missing values in this column should be filled with `3`, which represents 'Unknown'.

In [ ]:
```python
# Handling missing values
# Copies to avoid problems in original data
train_metadata_b_clean = train_metadata_b.copy()
val_metadata_b_clean = val_metadata_b.copy()
```

```
ethnicity_col = 'PreInt_Demos_Fam_Child_Ethnicity'

# Filling missing values with 3 ("Unknown")
train_metadata_b_clean[ethnicity_col] = train_metadata_b_clean[ethnicity_col].fillna(3)
val_metadata_b_clean[ethnicity_col] = val_metadata_b_clean[ethnicity_col].fillna(3)

# Ensuring all feature columns are treated as categorical
b_feature_columns = train_metadata_b_clean.drop(columns='participant_id').columns

for col in b_feature_columns:
    train_metadata_b_clean[col] = train_metadata_b_clean[col].astype('category')
    val_metadata_b_clean[col] = val_metadata_b_clean[col].astype('category')

# Confirming all columns are now categorical
print("Column data types (train):")
print(train_metadata_b_clean[b_feature_columns].dtypes)
```

```
Column data types (train):
Basic_Demos_Enroll_Year             category
Basic_Demos_Study_Site              category
PreInt_Demos_Fam_Child_Ethnicity    category
PreInt_Demos_Fam_Child_Race         category
MRI_Track_Scan_Location             category
Barratt_Barratt_P1_Edu              category
Barratt_Barratt_P1_Occ              category
Barratt_Barratt_P2_Edu              category
Barratt_Barratt_P2_Occ              category
dtype: object
```

5.2 Feature Selection

Feature selection for categorical data is performed using **Mutual Information**, which measures the dependency between each feature and the target `ADHD_Outcome`. All features are treated as discrete (categorical), as required for accurate analysis.

In [ ]:
```
# Calculating mutual information scores
mi_scores_b = mutual_info_classif(
    train_metadata_b_clean[b_feature_columns],
    train_labels['ADHD_Outcome'],
    discrete_features=True,
    random_state=42
)
```

```
# Creating DataFrame of MI scores
mi_scores_b_df = pd.DataFrame({
    'Feature': b_feature_columns,
    'MI_Score': mi_scores_b
}).sort_values(by='MI_Score', ascending=False)

# Displaying MI scores
mi_scores_b_df
```

Out[ ]:

	Feature	MI_Score
0	Basic_Demos_Enroll_Year	0.020455
4	MRI_Track_Scan_Location	0.014061
3	PreInt_Demos_Fam_Child_Race	0.005071
6	Barratt_Barratt_P1_Occ	0.004937
1	Basic_Demos_Study_Site	0.002598
8	Barratt_Barratt_P2_Occ	0.001855
5	Barratt_Barratt_P1_Edu	0.001620
7	Barratt_Barratt_P2_Edu	0.000938
2	PreInt_Demos_Fam_Child_Ethnicity	0.000814

In [34]:

```
# Selecting features with mutual information score above threshold
selected_features_b = mi_scores_b_df[mi_scores_b_df['MI_Score'] > 0.01]['Feature'].tolist()

print("Selected features from Metadata B:", selected_features_b)

# Creating reduced datasets
train_metadata_b_selected = train_metadata_b_clean[selected_features_b + ['participant_id']]
val_metadata_b_selected = val_metadata_b_clean[selected_features_b + ['participant_id']]

# Confirming final shapes
print("Training Metadata B after feature selection:", train_metadata_b_selected.shape)
print("Validation Metadata B after feature selection:", val_metadata_b_selected.shape)
```

```
Selected features from Metadata B: ['Basic_Demos_Enroll_Year', 'MRI_Track_Scan_Location']
Training Metadata B after feature selection: (970, 3)
Validation Metadata B after feature selection: (243, 3)
```

Step 6: Preprocessing Brain Connectome Data

The connectome dataset contains numerical values representing brain connectivity features for each participant.

6.1 Checking for Missing Values

According to the feedback, this dataset does not contain missing values. A final check is performed to confirm this.

In [ ]:
```
# Checking for missing values in training and validation connectome datasets
print("Missing values in training connectome:", train_connectome.isnull().sum().sum())
print("Missing values in validation connectome:", val_connectome.isnull().sum().sum())
```

```
Missing values in training connectome: 0
Missing values in validation connectome: 0
```

6.2 Standardisation

The connectome features must be standardised before applying dimensionality reduction.

In [36]:
```
# Dropping participant_id temporarily
connectome_features = train_connectome.drop(columns='participant_id').columns

# Initialising and fitting scaler on training data
scaler = StandardScaler()
train_connectome_scaled = scaler.fit_transform(train_connectome[connectome_features])
val_connectome_scaled = scaler.transform(val_connectome[connectome_features])
```

6.3 Dimensionality Reduction with KernelPCA

KernelPCA is used instead of traditional PCA, as the connectome features are typically uncorrelated and KernelPCA can better capture complex non-linear structures. A radial basis function (RBF) kernel is used (to capture complex patterns), and the number of components is initially set to 100 (arbitrary for now).

In [37]:
```
# Applying KernelPCA with RBF kernel
kernel_pca = KernelPCA(n_components=100, kernel='rbf', random_state=42)
```

```
# Fitting and transforming training data
train_connectome_reduced = kernel_pca.fit_transform(train_connectome_scaled)

# Transforming validation data using the same transformation
val_connectome_reduced = kernel_pca.transform(val_connectome_scaled)

# Converting to DataFrames and reattaching participant_id
train_connectome_reduced_df = pd.DataFrame(train_connectome_reduced,
                                           columns=[f'kpc_{i+1}' for i in range(train_connectome_reduced.shape[1])]
train_connectome_reduced_df['participant_id'] = train_connectome['participant_id'].values

val_connectome_reduced_df = pd.DataFrame(val_connectome_reduced,
                                         columns=[f'kpc_{i+1}' for i in range(val_connectome_reduced.shape[1])])
val_connectome_reduced_df['participant_id'] = val_connectome['participant_id'].values

# Confirming final shapes
print("Reduced training connectome shape:", train_connectome_reduced_df.shape)
print("Reduced validation connectome shape:", val_connectome_reduced_df.shape)
```

```
Reduced training connectome shape: (970, 101)
Reduced validation connectome shape: (243, 101)
```

Step 7: Merging All Cleaned Datasets and Saving Final Files

This final step merges the cleaned versions of all datasets into one final training and one final validation dataset, using `participant_id` as the join key.

Each of the following preprocessed components will be included:

- Metadata A (numerical features)
- Metadata B (categorical features)
- Brain Connectome (dimensionality-reduced features)
- Labels (`ADHD_Outcome` and `Sex_F`)

After merging, the resulting datasets will be saved as CSV files to be used in Stage 2 for model development.

In [38]:
```
# Merging all training components on 'participant_id'
train_merged = (
    train_labels
    .merge(train_metadata_a_selected, on='participant_id')
    .merge(train_metadata_b_selected, on='participant_id')
    .merge(train_connectome_reduced_df, on='participant_id')
)

# Merging all validation components
val_merged = (
    val_labels
    .merge(val_metadata_a_selected, on='participant_id')
    .merge(val_metadata_b_selected, on='participant_id')
    .merge(val_connectome_reduced_df, on='participant_id')
)

# Reordering columns: participant_id first, features in the middle, labels at the end (for easier management in Model
def reorder_columns(df):
    # Starting with participant_id
    cols = ['participant_id']

    # Finding label columns
    label_cols = ['ADHD_Outcome', 'Sex_F', 'combined_label']

    # Getting all other columns(features)
    feature_cols = [col for col in df.columns if col not in cols + label_cols]

    # Final column order
    return df[cols + feature_cols + label_cols]

# Applying reordering
train_merged = reorder_columns(train_merged)
val_merged = reorder_columns(val_merged)

# Confirming final structure
print("Final training dataset shape:", train_merged.shape)
print("Final validation dataset shape:", val_merged.shape)
train_merged.head()
```

```
Final training dataset shape: (970, 117)
Final validation dataset shape: (243, 117)
```

Out[38]:

	participant_id	SDQ_SDQ_Hyperactivity	SDQ_SDQ_Externalizing	SDQ_SDQ_Difficulties_Total	SDQ_SDQ_Generating_Impact	SDC
0	CPaeQkhcjg7d	0.888748	0.112626	0.876890	1.043100	
1	Nb4EetVPm3gs	0.541133	0.112626	0.281660	0.336531	
2	p4vPhVu91o4b	1.583979	1.996193	1.769735	1.749669	
3	M09PXs7arQ5E	1.583979	0.818963	0.876890	-0.016754	
4	tBGXkEdv2cp7	0.888748	1.054409	0.728083	1.749669	

5 rows × 117 columns

In [39]:

```
# Saving cleaned datasets to CSV
train_merged.to_csv('cleaned_train_data.csv', index=False)
val_merged.to_csv('cleaned_validation_data.csv', index=False)

print("Cleaned datasets saved as 'cleaned_train_data.csv' and 'cleaned_validation_data.csv'")
```

```
Cleaned datasets saved as 'cleaned_train_data.csv' and 'cleaned_validation_data.csv'
```

Step 8: Plan for Stage 2 (Model Development)

In Stage 2, the objective will be to train predictive models for the two labels: `ADHD_Outcome` and `Sex_F`, using the cleaned and preprocessed dataset prepared in Stage 1.

8.1 Target Variables

The final dataset includes two classification labels:

- `ADHD_Outcome` (binary classification)
- `Sex_F` (binary classification)

8.2 Candidate Models

A variety of classification models will be considered:

- **Logistic Regression**

- **Random Forest Classifier**
- **Support Vector Machines (SVM)**
- **K-Nearest Neighbors (KNN)**
- **Gradient Boosting (e.g., XGBoost)**
- **Multi-layer Perceptron (Neural Networks)**

8.3 Evaluation Metrics

Since both labels are binary, the following metrics will be used to evaluate model performance:

- Accuracy
- Precision
- Recall
- F1 Score
- Area Under the ROC Curve (AUC-ROC)

Metrics will be reported on both the training and validation sets.

8.4 Hyperparameter Tuning

- **Grid Search** and/or **Random Search**
- **Cross-Validation** (e.g., 5-fold stratified)

8.5 Bias and Fairness

The potential for model bias will be evaluated, especially regarding:

- Gender (`Sex_F`)
- Data imbalance

8.6 Interpretability

Model interpretability will be addressed considering the use of:

- **Feature importance** (tree-based models)
- **Permutation importance**

- **SHAP values** (for complex models like XGBoost or MLP)
- **LIME**

Interpretability is essential to ensure the models offer transparent, explainable predictions in a health-related context.