### Documentation: Synthetic Dataset Generation for ADHD Prediction using Serious Game Data

#### Purpose:

This Python script generates a synthetic dataset of users who play a serious game that collects various behavioral metrics. The dataset simulates data for players, with the goal of training machine learning models to predict whether a player has ADHD based on their in-game performance. The generated dataset includes user IDs, ADHD-related behavioral metrics (randomly generated), distraction metrics (thresholded values), and a binary label indicating whether a player has ADHD.

The dataset consists of four major parts:

1. **User ID**: A unique identifier for each player.
2. **ADHD Metrics**: Randomly generated metrics that represent behavioral traits relevant to ADHD (e.g., cognitive flexibility, error rates).
3. **Distraction Metrics**: Thresholded values representing a player's distraction levels during the game.
4. **Label**: A binary classification (has\_adhd), indicating whether the player has ADHD.

#### Key Components:

1. **Logging (**Log**)**:
   * A helper function to print log messages with timestamps, allowing tracking of the progress of the data generation process.
2. **Random Data Generation (**generate\_list**)**:
   * Generates lists of random values from different probability distributions (normal, beta, binomial, etc.) to simulate game performance metrics.
   * This function is used to create synthetic ADHD-related metrics and distraction metrics for each user.
3. **Thresholding (**thresholding**)**:
   * Applies a threshold to a list of values, converting continuous values into binary (1 or 0) to simulate binary distraction levels in the game.
4. **Main Process**:
   * **Step 1**: Generate random behavioral metrics (both ADHD-related and distraction-related) for a specified number of users (users\_num = 1000).
   * **Step 2**: Assign a label (has\_adhd) to each user. A certain percentage of users (adhd\_positive\_ratio = 0.15) are randomly assigned ADHD based on a predefined probability.
   * **Step 3**: For each user, the script creates rows of data containing:
     + **ADHD metrics** (randomly chosen based on the user’s ADHD status).
     + **Distraction metrics** (binary values based on thresholding).
   * **Step 4**: Append each user's data to a list.
   * **Step 5**: Convert the list into a pandas DataFrame and save it as a CSV file (dataset.csv).
5. **Columns in the Generated Dataset**:
   * **User ID** (user\_id): Unique identifier for each user.
   * **ADHD-related Metrics**: A set of metrics related to cognitive performance and motor control (e.g., cognitive flexibility, reaction time, processing speed).
   * **Distraction Metrics**: Binary distraction values (1 = distracted, 0 = not distracted).
   * **ADHD Label** (has\_adhd): Indicates whether the user has ADHD (1 = ADHD, 0 = no ADHD).
6. **Data Saving**:
   * The generated data is saved as a CSV file for further machine learning analysis.

#### Example Use Case:

The generated dataset is intended to be used to train machine learning models (e.g., Random Forest, Neural Networks) to classify players as having ADHD or not based on their in-game behavior.

#### CSV Output:

* The CSV file (dataset.csv) contains one row per user, with columns for the user ID, ADHD metrics, distraction metrics, and the ADHD label.

This synthetic dataset provides a controlled environment to test and develop machine learning algorithms aimed at ADHD detection.

### Documentation: Machine Learning Model Training for ADHD Prediction

#### Purpose:

This script is designed to train a machine learning model using a synthetic dataset that contains data about users playing a serious game. The dataset is pre-generated and includes features such as cognitive flexibility, reaction time, and other metrics, along with a label indicating whether the user has ADHD. The purpose of this script is to train a classifier that predicts whether a player has ADHD based on these metrics.

The key features of the script include:

* Menu-driven selection of machine learning algorithms (though currently only Random Forest is implemented).
* Optional handling of class imbalance through SMOTE (Synthetic Minority Over-sampling Technique).
* Model evaluation using multiple metrics such as accuracy, precision, recall, F1-score, and confusion matrix.
* Saving the trained model for future use.

#### Key Components:

1. **Logging (**Log**)**:
   * A helper function that prints timestamped log messages to track the progress of various tasks. This provides clear visibility into the execution flow of the script.
2. **Menu for Algorithm Selection (**Menu**)**:
   * A simple function that presents the user with a menu of different machine learning algorithms.
   * The user selects an algorithm by entering a number corresponding to their choice. Currently, only **Random Forest** is implemented, though the structure allows for easy expansion to include other models.
3. **Main Data Processing Flow (**main**)**:
   * The core logic of the script is contained within the main() function, ensuring that the code executes only when run directly (and not when imported as a module).
4. **Dataset Loading and Preprocessing**:
   * **Dataset**: The script loads the dataset (dataset.csv), which includes user data (excluding user\_id) and the ADHD classification (has\_adhd).
   * **Feature-Target Split**: The features (game performance metrics) are separated from the target labels (has\_adhd).
5. **Handling Class Imbalance (Optional) (**SMOTE**)**:
   * The script provides the option to apply **SMOTE**, which is useful when the dataset has an imbalance between ADHD and non-ADHD cases.
   * SMOTE generates synthetic samples of the minority class (ADHD cases) to create a more balanced dataset for training.
6. **Train-Test Split**:
   * After applying SMOTE (if selected), the script splits the dataset into training and testing sets (80% for training and 20% for testing).
7. **Random Forest Classifier**:
   * A **Random Forest** model is trained using the training data. Random Forest is an ensemble learning method that creates a collection of decision trees for classification.
   * The model is trained with 100 decision trees (n\_estimators=100), with a fixed random state (random\_state=42) for reproducibility.
8. **Model Evaluation**:
   * After training, the model's performance is evaluated using the test data. Multiple metrics are computed:
     + **Accuracy**: Measures overall correctness.
     + **Precision**: Focuses on the correctness of positive predictions (ADHD cases).
     + **Recall**: Measures how well the model captures actual positive cases.
     + **F1-Score**: Harmonic mean of precision and recall, balancing both.
     + **Confusion Matrix**: A matrix representation of true positives, true negatives, false positives, and false negatives.
     + **Classification Report**: A summary of precision, recall, and F1-score for both classes (ADHD and non-ADHD).
9. **Class Imbalance Metrics**:
   * If SMOTE is applied, the script reports the class distribution after resampling. Otherwise, it reports the original class distribution from the dataset. This helps understand the imbalance ratio between ADHD and non-ADHD cases.
10. **Saving the Model**:

* The trained Random Forest model is saved using joblib for later use. The model is stored in a file named adhd\_random\_forest\_model.pkl.

1. **Execution Control (**if \_\_name\_\_ == "\_\_main\_\_":**)**:

* The code is structured to run only when the script is executed directly. This ensures that functions like main() are not called if the script is imported as a module in another script.

#### Example Workflow:

1. **Start**:
   * The script begins with logging the start of the process.
   * The user is prompted to select a machine learning algorithm (currently only Random Forest is functional).
2. **Dataset Loading and Preprocessing**:
   * The dataset is loaded, features are separated from the labels, and class imbalance handling is applied (if selected).
3. **Model Training**:
   * The Random Forest classifier is trained on the dataset.
4. **Model Evaluation**:
   * Predictions are made on the test data, and various performance metrics are calculated and logged.
5. **Model Saving**:
   * The trained model is saved to disk, allowing it to be reused without retraining.

#### Logging:

Throughout the process, the Log function provides time-stamped updates on the following stages:

* Dataset loading.
* Application of SMOTE (if enabled).
* Train-test split.
* Model training.
* Prediction and evaluation.
* Model saving.

This logging helps track the progress of the script and provides transparency in the workflow.

#### Output:

* **Model File**: A saved Random Forest model (adhd\_random\_forest\_model.pkl).
* **Performance Metrics**: The script prints and logs metrics such as accuracy, precision, recall, F1-score, confusion matrix, and class imbalance ratio.

This structured process allows for easy extension, reproducibility, and reliable model training for ADHD prediction based on game performance data.