Given the considerations and the importance of nutritional data in the SmartChef project, let's revise Step 2 to include a more nuanced approach to handling missing data, blending imputation for some fields with the preservation of complete cases for critical analyses.

**Step 2: Prepare the Dataset (Revised)**

2.1 Load the Dataset

1. **Install Necessary Libraries** (if not already installed):

bashCopy code

pip install pandas scikit-learn

1. **Load Your CSV**:

pythonCopy code

import pandas as pd file\_path = '/mnt/data/epi\_r.csv' # Adjust path as necessary data = pd.read\_csv(file\_path)

2.2 Data Cleaning

1. **Identify Missing Values**:

pythonCopy code

missing\_values = data.isnull().sum() print(missing\_values[missing\_values > 0])

1. **Remove Duplicate Entries**:

pythonCopy code

data = data.drop\_duplicates()

2.3 Handling Missing Nutritional Data

Given the significant amount of missing nutritional data, we will employ a two-pronged approach:

* **Imputation for General Use**: For broad recommendations where exact nutritional values are less critical.
* **Preservation of Complete Cases**: For features or analyses where nutritional accuracy is paramount.

1. **Impute Missing Values Using Median**:

pythonCopy code

for column in ['calories', 'protein', 'fat', 'sodium']: data[column].fillna(data[column].median(), inplace=True)

1. **Create a Flag for Recipes with Originally Missing Information**: This step allows us to track which recipes have imputed values, should we need to filter them out for certain analyses or user features.

pythonCopy code

data['imputed'] = 0 data.loc[data[['calories', 'protein', 'fat', 'sodium']].isnull().any(axis=1), 'imputed'] = 1

1. **(Optional) Advanced Imputation Techniques**: For a more sophisticated approach, consider using predictive models or KNN for imputation. This requires a bit more setup:

pythonCopy code

from sklearn.impute import KNNImputer # Example using KNNImputer; adjust 'n\_neighbors' as needed imputer = KNNImputer(n\_neighbors=5) nutritional\_cols = ['calories', 'protein', 'fat', 'sodium'] data[nutritional\_cols] = imputer.fit\_transform(data[nutritional\_cols])

2.4 Data Splitting for ML Models

Before splitting the data, ensure your feature matrix (X) and target variable (y) are defined. Assuming 'rating' is your target:

1. **Define Features and Target**:

pythonCopy code

X = data.drop(['rating', 'title', 'imputed'], axis=1) # Exclude 'title' and 'imputed' from features y = data['rating']

1. **Split the Data**:

pythonCopy code

from sklearn.model\_selection import train\_test\_split X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

This revised step ensures that your dataset is clean and ready for both broad and precise nutritional analysis. It maintains the integrity of your data for critical dietary recommendations while also maximizing the use of available data for general features and insights.

Top of Form