To start modularizing your SmartChef project, let's outline the Python files you'll create, aligning with the suggested file structure. This approach will organize your project, making it easier to manage, understand, and expand.

**File Names to Create:**

1. **config.py**
   * For global settings and configurations.
2. **data\_preparation.py**
   * For data loading, cleaning, and preprocessing tasks.
3. **exploratory\_data\_analysis.py**
   * For EDA functions and visualization scripts.
4. **feature\_engineering.py**
   * Optional, for any complex feature engineering beyond basic preprocessing.
5. **model.py**
   * For model training, evaluation, and serialization.
6. **recommendation\_system.py**
   * For the recommendation logic based on user preferences.
7. **user\_interface.py**
   * Optional, for the CLI or web interface for user interaction.
8. **main.py**
   * The main script that orchestrates the project workflow.

**Starting with config.py:**

This file will store your project's configuration settings. Let's begin by setting up this file.

**config.py**

pythonCopy code

# config.py # Path to the dataset file DATASET\_PATH = 'path/to/your/dataset.csv' # Model parameters (example) MODEL\_PARAMS = { 'n\_estimators': 100, 'random\_state': 42, } # Any other

User

**You**

smart chef.txt

Document

this is the entire code. help me segregate them to more manageable parts like above I want each python file to start with """==================================================================== Code for Project: Smart Chef, prepared by: D. Cabreros, E.L. Dorji & S. Lhamo, for ICT619 Project of S1-2024 ======================================================================= """

ChatGPT

**ChatGPT**

Given the comprehensive script you've provided, let's start segregating it into more manageable parts as per the suggested file structure. We'll begin by creating the proposed Python files, starting with **config.py**, and then proceed to populate each file with relevant parts of the code from your script. Each file will start with the specified header comment.

**1. config.py**

This file will contain configuration settings like file paths and model parameters.

pythonCopy code

"""==================================================================== Code for Project: Smart Chef, prepared by: D. Cabreros, E.L. Dorji & S. Lhamo, for ICT619 Project of S1-2024 ======================================================================= """ # Path to the dataset file DATASET\_PATH = 'epi\_r.csv' # Model parameters MODEL\_PARAMS = { 'n\_estimators': 100, 'random\_state': 42, }

**2. data\_preparation.py**

This file will include functions for loading the dataset, handling missing values, and preparing the data for analysis and modeling.

pythonCopy code

"""==================================================================== Code for Project: Smart Chef, prepared by: D. Cabreros, E.L. Dorji & S. Lhamo, for ICT619 Project of S1-2024 ======================================================================= """ import pandas as pd from config import DATASET\_PATH def load\_and\_prepare\_data(file\_path): data = pd.read\_csv(file\_path) data.drop\_duplicates(inplace=True) # Flagging recipes with originally missing info data['imputed'] = 0 data.loc[data[['calories', 'protein', 'fat', 'sodium']].isnull().any(axis=1), 'imputed'] = 1 # Impute missing values using median for column in ['calories', 'protein', 'fat', 'sodium']: data[column].fillna(data[column].median(), inplace=True) return data

**3. exploratory\_data\_analysis.py**

Contains functions for performing EDA, including visualizations.

pythonCopy code

"""==================================================================== Code for Project: Smart Chef, prepared by: D. Cabreros, E.L. Dorji & S. Lhamo, for ICT619 Project of S1-2024 ======================================================================= """ import matplotlib.pyplot as plt import seaborn as sns def perform\_eda(data): # Distribution of ratings sns.histplot(data['rating'], bins=20, kde=True) plt.title('Distribution of Recipe Ratings') plt.xlabel('Rating') plt.ylabel('Frequency') plt.show() # Add more EDA functions as needed...

**Proceeding with Other Files:**

Continue creating additional files (**feature\_engineering.py**, **model.py**, **recommendation\_system.py**, **user\_interface.py**, and **main.py**) by similarly segregating the relevant parts of your script into these modules. For instance, model training and evaluation would go into **model.py**, and the logic for filtering recipes and generating recommendations based on user preferences would be placed in **recommendation\_system.py**.

**Final Step - main.py:**

The **main.py** file will serve as the entry point for your application, tying together functions and modules defined in other files.

After segregating the code, you can use imports in **main.py** to utilize the functionalities defined in other modules. This modular approach not only organizes your code better but also makes it easier to manage and extend your project.

Let's continue creating the remaining files. If you need specific help with any of these files or further detailed breakdowns, let me know!

User

**You**

continue with the next file. just one

ChatGPT

**ChatGPT**

Let's proceed with creating the **model.py** file, which will encompass model selection, training, and evaluation tasks.

**4. model.py**

This file includes the functions for training the machine learning model and evaluating its performance.

pythonCopy code

"""==================================================================== Code for Project: Smart Chef, prepared by: D. Cabreros, E.L. Dorji & S. Lhamo, for ICT619 Project of S1-2024 ======================================================================= """ from sklearn.ensemble import RandomForestRegressor from sklearn.metrics import mean\_squared\_error, r2\_score from config import MODEL\_PARAMS import pandas as pd def train\_model(X\_train, y\_train): """ Trains the Random Forest model on the training data. Parameters: - X\_train (pd.DataFrame): Training features. - y\_train (pd.Series): Target variable for training data. Returns: - model: Trained model. """ model = RandomForestRegressor(\*\*MODEL\_PARAMS) model.fit(X\_train, y\_train) return model def evaluate\_model(model, X\_test, y\_test): """ Evaluates the trained model on the test set. Parameters: - model: The trained model. - X\_test (pd.DataFrame): Testing features. - y\_test (pd.Series): True target variable for testing data. Returns: - dict: Dictionary containing RMSE and R² metrics. """ y\_pred = model.predict(X\_test) mse = mean\_squared\_error(y\_test, y\_pred) rmse = mse \*\* 0.5 r2 = r2\_score(y\_test, y\_pred) return {'RMSE': rmse, 'R²': r2}

In **model.py**, we define two primary functions: **train\_model** for training the Random Forest model using the training data, and **evaluate\_model** for assessing the model's performance with the test data, returning both RMSE and �2*R*2 metrics. The model's parameters are fetched from **config.py** to maintain a centralized configuration setup.

**Integration in main.py:**

When you're ready to tie everything together in **main.py**, you will import and utilize these functions to train and evaluate your model within the broader application workflow. Here’s a snippet illustrating how to incorporate the **model.py** functions:

pythonCopy code

from data\_preparation import load\_and\_prepare\_data from model import train\_model, evaluate\_model # Assuming you have a function to split your data in data\_preparation.py or elsewhere from data\_preparation import split\_data def main(): data = load\_and\_prepare\_data('path/to/dataset.csv') X\_train, X\_test, y\_train, y\_test = split\_data(data) model = train\_model(X\_train, y\_train) evaluation\_metrics = evaluate\_model(model, X\_test, y\_test) print(f"Evaluation Metrics: {evaluation\_metrics}") if \_\_name\_\_ == "\_\_main\_\_": main()

This modular approach makes your code cleaner, more maintainable, and easier to extend or modify. Next, you would proceed with creating and implementing the remaining files (**recommendation\_system.py**, **user\_interface.py**, etc.) similarly, focusing on separating concerns and encapsulating functionalities.

User

**You**

continue to recommendation\_system.py

ChatGPT

**ChatGPT**

Continuing with the modularization, let's create the **recommendation\_system.py** file. This module will handle generating recipe recommendations based on user preferences, incorporating the model's predictions.

**5. recommendation\_system.py**

This file implements the logic for filtering recipes according to user preferences and using the trained model to suggest the most suitable recipes.

pythonCopy code

"""==================================================================== Code for Project: Smart Chef, prepared by: D. Cabreros, E.L. Dorji & S. Lhamo, for ICT619 Project of S1-2024 ======================================================================= """ import pandas as pd def filter\_recipes\_by\_preferences(data, preferences): """ Filters recipes based on user preferences. Parameters: - data (pd.DataFrame): Dataset containing recipes. - preferences (dict): User preferences including dietary restrictions, disliked ingredients, etc. Returns: - pd.DataFrame: Filtered dataset based on preferences. """ filtered\_data = data.copy() # Example: Filtering based on dietary restriction (e.g., vegetarian) if 'dietary\_restriction' in preferences: filtered\_data = filtered\_data[filtered\_data[preferences['dietary\_restriction']] == 1] # Implement additional filtering based on other preferences... return filtered\_data def generate\_recommendations(model, filtered\_data): """ Generates recipe recommendations using the trained model. Parameters: - model: The trained machine learning model. - filtered\_data (pd.DataFrame): The filtered dataset based on user preferences. Returns: - pd.DataFrame: Top recommended recipes. """ # Assuming 'predicted\_rating' is to be determined by the model filtered\_data['predicted\_rating'] = model.predict(filtered\_data.drop(['rating', 'title'], axis=1)) # Sort by predicted rating to get top recommendations recommendations = filtered\_data.sort\_values(by='predicted\_rating', ascending=False) return recommendations.head(10) # Adjust the number of recommendations as needed

In **recommendation\_system.py**, two functions are defined: **filter\_recipes\_by\_preferences**, which filters the dataset based on a dictionary of user preferences; and **generate\_recommendations**, which applies the trained model to the filtered dataset to predict ratings and sort the recipes by their predicted ratings to recommend the top recipes.

**Integration in main.py:**

To utilize the recommendation system in your application, you'll import and call these functions in **main.py**, integrating them with the rest of your application's workflow.

pythonCopy code

from data\_preparation import load\_and\_prepare\_data from model import train\_model from recommendation\_system import filter\_recipes\_by\_preferences, generate\_recommendations # Import or define a method for collecting user preferences, possibly from user\_interface.py from user\_interface import collect\_user\_preferences def main(): data = load\_and\_prepare\_data('path/to/dataset.csv') user\_preferences = collect\_user\_preferences() filtered\_data = filter\_recipes\_by\_preferences(data, user\_preferences) model = train\_model(...) # Ensure you have the model trained with appropriate data top\_recommendations = generate\_recommendations(model, filtered\_data) print("Top Recommendations:", top\_recommendations) if \_\_name\_\_ == "\_\_main\_\_": main()

This structure allows for a clear separation of concerns: **data\_preparation.py** for initial data setup, **model.py** for all modeling activities, and **recommendation\_system.py** for applying those models to make meaningful recommendations based on user inputs. Next steps would include finalizing the **user\_interface.py** module for interacting with users and pulling everything together in **main.py**.

please recreate the entire code for main.py

ChatGPT

**ChatGPT**

Let's create the **main.py** file that orchestrates your SmartChef project by tying together the functionalities defined in the modularized files. This file serves as the entry point to your application, handling the workflow from data preparation to generating recommendations based on user preferences.

**main.py**

pythonCopy code

"""==================================================================== Code for Project: Smart Chef, prepared by: D. Cabreros, E.L. Dorji & S. Lhamo, for ICT619 Project of S1-2024 ======================================================================= """ from config import DATASET\_PATH from data\_preparation import load\_and\_prepare\_data from exploratory\_data\_analysis import perform\_eda # Import the function for splitting data if it's defined in data\_preparation.py or elsewhere from data\_preparation import split\_data from model import train\_model, evaluate\_model from recommendation\_system import filter\_recipes\_by\_preferences, generate\_recommendations from user\_interface import collect\_user\_preferences def main(): # Load and prepare the dataset data = load\_and\_prepare\_data(DATASET\_PATH) # Optional: Perform exploratory data analysis perform\_eda(data) # Split the dataset into training and testing sets X\_train, X\_test, y\_train, y\_test = split\_data(data) # Train the model on the training set model = train\_model(X\_train, y\_train) # Evaluate the model on the test set evaluation\_metrics = evaluate\_model(model, X\_test, y\_test) print(f"Evaluation Metrics: {evaluation\_metrics}") # Collect user preferences for recipe recommendations user\_preferences = collect\_user\_preferences() # Filter the recipes based on user preferences filtered\_data = filter\_recipes\_by\_preferences(data, user\_preferences) # Generate recommendations based on the filtered dataset and trained model recommendations = generate\_recommendations(model, filtered\_data) # Display the top recommendations to the user print("Top Recommendations:", recommendations) if \_\_name\_\_ == "\_\_main\_\_": main()

**Key Components of main.py:**

* **Data Preparation:** Loads and prepares the dataset using functions from **data\_preparation.py**.
* **Exploratory Data Analysis (EDA):** Optionally performs EDA on the prepared data to glean insights.
* **Model Training and Evaluation:** Trains a machine learning model with the prepared data and evaluates its performance using metrics.
* **User Preferences and Recommendations:** Collects user preferences (this part requires **collect\_user\_preferences** function implementation in **user\_interface.py**), filters recipes based on those preferences, and generates recipe recommendations.
* **Display Recommendations:** Prints the top recommendations based on the model's predictions and user preferences.

**Note:**

* Ensure that all the imported functions (**load\_and\_prepare\_data**, **perform\_eda**, **split\_data**, **train\_model**, **evaluate\_model**, **filter\_recipes\_by\_preferences**, **generate\_recommendations**, and **collect\_user\_preferences**) are correctly defined in their respective modules with the expected parameters and return values.
* The **split\_data** function is mentioned as part of the data preparation step. If it's not separately defined, you might need to adjust the script to include your data splitting logic directly in **main.py** or within **data\_preparation.py**.
* **collect\_user\_preferences** is assumed to be a part of **user\_interface.py**, which should collect and return user preferences in a format compatible with **filter\_recipes\_by\_preferences**.

This structured approach not only makes your code more readable and maintainable but also facilitates easier debugging and future enhancements to your SmartChef project.