Dietary Compass: Personalized Food Choice Navigator

BAX 422 Data Design and Representation: Project I

Team: Via Lin, Mahnoor Shahid, Jiyeon (Jenna) Woo, Kaylyn Nguyen

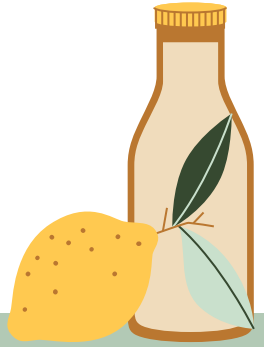
01

Project Scope

Motivation

- **Problem Overview:**
  - 60% of Americans have dietary restrictions
  - Growing demand for transparency in food choices
  - Increasing complexity of processed foods

- **Project Goals:**
  - Simplify healthy food selection
  - Automate dietary restriction checking
  - Educate users about processed foods



02

Implementation

API Used

Open Food Facts

- Provides access to crowdsourced database of food products from around the world
- Includes details such as ingredients, nutrition facts, labels, allergens, and sustainability scores
- Supports JSON, XML, CSV formats
- Free to use under open data license

User Flow

1. Input Phase
   - Product search
   - Dietary preferences setting
   - Restriction specification
2. Processing Phase
   - API data retrieval
   - Score calculation
   - Restriction checking
3. Output Phase
   - Results display
   - Alternative suggestions
   - Detailed explanations

Healthier Alternatives

1. How to find the healthiest food options?

Challenge:

- Users struggle to find healthier alternatives to favorite foods
- Difficult to make effective comparisons between similar products

Solution:

- Input-based recommendation system
- Example: User inputs "soda" → Display options sorted by:
  - Lowest calorie content
  - Lowest sugar content
  - Higher nutrition scores

Dietary Restrictions

2. Does this product align with my dietary restrictions?

Challenge:

- Time-consuming to check every ingredient
- Risk of missing important dietary restrictions
- Complex for those with multiple restrictions

Solution:

- Automated ingredient analysis
- Cross-reference with user dietary preferences
- Instant flagging system for:
  - Allergens
  - Diet incompatibilities
  - Restricted ingredients

Processing Awareness

3. How can I make informed decisions about processed foods?

Challenge:

- Difficulty understanding food processing levels
- Limited knowledge of additives and preservatives
- Complex ingredient lists

Solution:

- Custom "Processing Score" system based on:
  - Number of additives
  - Types of artificial ingredients
  - Processing methods
- Clear ingredient explanations
- Suggestions for less processed alternatives

03

Findings

Healthy Food Finder

```
===  Healthy Food Finder  ===

 Enter a food name (or 'quit' to exit): potato chips

 Searching for healthier alternatives

 Top 5 Most Healthy Options (Sorted by Low Calories & Sugar):

1. Oven Baked Sweet Potato Fries
    Nutri-Score: C
    Calories: 156.0 kcal per 100g
    Sugar: 13.0g per 100g

2. Hash Browns
    Nutri-Score: C
    Calories: 169.0 kcal per 100g
    Sugar: 0.5g per 100g

3. Crispy French Fries
    Nutri-Score: B
    Calories: 250.0 kcal per 100g
    Sugar: 0.5g per 100g

4. Sour Cream & Onion Potato Chips
    Nutri-Score: C
    Calories: 432.0 kcal per 100g
    Sugar: 4.4g per 100g

5. Barbeque Potato Chips
    Nutri-Score: C
    Calories: 435.0 kcal per 100g
    Sugar: 8.3g per 100g
```

Example input: potato chips

- Healthiest choices suggested based on calorie count and sugar level
- Top 5 relevant results are suggested
- Nutrition score indicates how healthy the similar item is overall

Functions/Libraries Used

Libraries

- Requests
- Time
- Json

Functions

- Fetch products by name
- Extract key nutritional information from a product dictionary
- Find healthier options(product_name)

Healthy Food Finder

DDR_Proj1_Code

File Edit View Insert Runtime Tools Help

+ Code + Text

QUESTION 1: How to find the healthiest food options (based on low calories and sugar)?

Tool: Healthy Food Finder

[7]

```python
import requests
import time
import json

# Cache to store API responses and avoid redundant calls
cache = {}

# API Base URL
SEARCH_URL = "https://world.openfoodfacts.org/cgi/search.pl"
SLEEP_TIME = 1  # Sleep time in seconds to avoid excessive requests

def fetch_products_by_name(product_name):
    # Fetch a list of products by name from Open Food Facts with caching and rate-limiting
    if product_name in cache:
        return cache[product_name]
    params = {
        "search_terms": product_name,
        "search_simple": 1,
        "action": "process",
        "json": 1,
    }
    response = requests.get(SEARCH_URL, params=params)
    time.sleep(SLEEP_TIME)  # Avoid overwhelming API requests

    if response.status_code == 200:
        data = response.json()
        products = data.get("products", [])
        cache[product_name] = products
        return products
    return []

def extract_nutritional_info(product):
```

18s completed at 8:56 AM

Dietary Compatibility Scanner

```
===  Dietary Compatibility Scanner  ===

Enter the food name (e.g., 'bread', 'milk', 'pasta'): lasagna

Available products:
1. Lasagne all'uovo (Barcode: 8076800376999)
2. Lasagne Platten (Barcode: 8076809523738)
3. Lasagna (Barcode: 8480000044877)
4. Lasagne Pasta (Barcode: 5000436101925)
5. Lasagnes à la bolognaise (Barcode: 3166352968591)
6. Lasagnes (Barcode: 20411978)
7. Beef Lasagne (Barcode: 00206310)
8. Lasagnes à la bolognaise (Barcode: 3302740047534)
9. Lasaña Fácil (Barcode: 8410173072025)
10.  (Barcode: 20163723)

 Select a product by number: 4

 Dietary Options Available: Vegan, Vegetarian, Gluten-Free, Lactose-Free
Do you have any dietary restrictions or allergens you would like to check for this product?
 (e.g., vegan, gluten-free, lactose-free, or allergens like milk, peanuts, etc.): gluten-free, milk

 Allergen and Dietary Information:
 Product Name: Lasagne Pasta
 Barcode: 5000436101925

 Allergens: Gluten
 Gluten-free: Not Suitable
```

Example input: lasagna

- Can explore up to 10 available products that match user input
- Barcode information is provided
- Can select a specific product for allergen information
- Can search for specific allergens

Functions/Libraries Used

Libraries

- Requests, Time

Functions

- fetch_product_details(barcode)
- check_allergens(product)
- check_dietary_restrictions(product, restrictions)

Dietary Compatibility Scanner

DDR_Proj1_Code

File Edit View Insert Runtime Tools Help

+ Code + Text

QUESTION 2: Does this product align with my dietary restrictions?

Tool: Dietary Compatibility Scanner

```python
import requests
import time

# Cache to store API responses
cache = {}

# API Base URL
SEARCH_URL = "https://world.openfoodfacts.org/cgi/search.pl"
PRODUCT_URL = "https://world.openfoodfacts.org/api/v0/product/{}.json"
SLEEP_TIME = 0.5  # Reduce sleep time to improve performance
ALLERGENS_LIST = ["Milk", "Eggs", "Peanuts", "Tree nuts", "Soy", "Wheat", "Fish", "Shellfish", "Sesame"]
DIETARY_OPTIONS = {
    "vegan": "en:vegan",
    "vegetarian": "en:vegetarian",
    "gluten-free": "en:gluten-free",
    "lactose-free": "en:lactose-free"
}

def fetch_product_by_name(product_name):
    # Fetch a list of products by name from Open Food Facts with caching and rate-limiting
    if product_name in cache:
        return cache[product_name]  # Return cached results
    params = {
        "search_terms": product_name,
        "search_simple": 1,
        "action": "process",
        "json": 1,
        "fields": "product_name,code"
    }
    response = requests.get(SEARCH_URL, params=params)
    time.sleep(SLEEP_TIME)
    if response.status_code == 200:
        data = response.json()
        products = data.get("products", [])
```

15s completed at 8:57 AM

Food Processing Analyzer

```
1. Organic - Lightly Salted - Wholegrain
    Processing Score: 55/100
    NOVA Score: 3/4
    Additives: 0
    Ingredients: 2

2. Nutri+ avoine chocolat
    Processing Score: 25/100
    NOVA Score: 4/4
    Additives: 3
    Ingredients: 21

3. Eyoo cover
    Processing Score: 10/100
    NOVA Score: 4/4
    Additives: 6
    Ingredients: 20

4. Momo black
    Processing Score: 0/100
    NOVA Score: 4/4
    Additives: 8
    Ingredients: 17

5. Merendino
    Processing Score: 0/100
    NOVA Score: 4/4
    Additives: 8
    Ingredients: 21
```

```
Enter a number to analyze product
Select #: 2

 Processing Analysis Report for Nutri+ avoine chocolat

1. Processing Classification
    NOVA Group: 4/4
    Classification: Ultra-processed foods
    Processing Score: 25/100

2. Ingredients Analysis
    Total Ingredients: 21
    Number of Additives: 3

3. Additives Found:
    e392
    e500
    e503

4. Full Ingredients List:
   Flocons d'avoine* 31%, farine de blé*, huile de colza
```

Example input: cake

- 5 of the least processed foods are selected
  - High score = less processed
- NOVA score:
  1. Unprocessed / minimally processed
  2. Processed ingredients
  3. Processed
  4. Ultra-Processed
- Lists number of ingredients and additives

Functions/Libraries Used

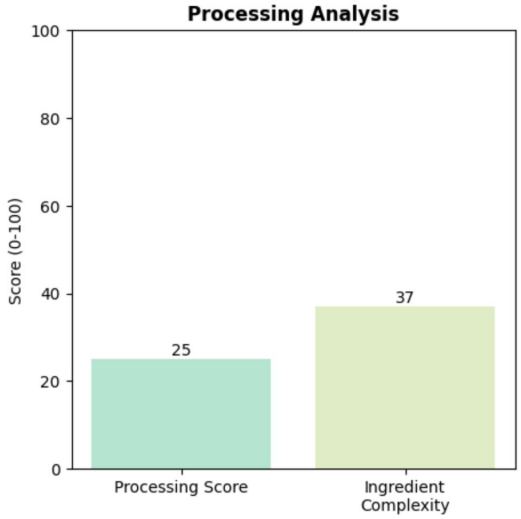
Functions

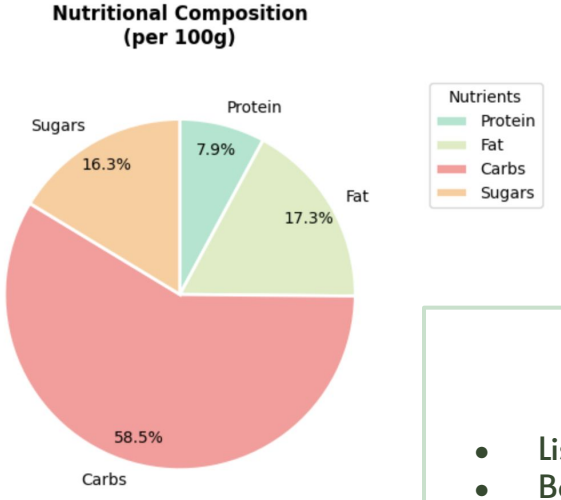
- Search products by name and barcode
- Calculate processing scores based on ingredients and additives
- Generate visualizations for processing levels and nutrition
- Rate-limited API request handling with caching

Libraries

- requests & requests_cache
- pandas
- matplotlib.pyplot
- time & typing

Food Processing Analyzer





Example input: cake

- Lists all ingredients
- Bar graph shows processing score and ingredient complexity
- Pie chart shows nutritional composition

Food Processing Analyzer

DDR_Proj1_Code Saving...

File Edit View Insert Runtime Tools Help

+ Code + Text

RAM Disk

QUESTION 3: How can I make informed decisions about processed foods?

Tool: Food Processing Analyzer

```
!pip install requests_cache
import requests
import requests_cache
import pandas as pd
import matplotlib.pyplot as plt
import time
from typing import Dict, Optional, List

class FoodAnalyzer:
    API_BASE = "https://world.openfoodfacts.org/api/v0/product/"
    SEARCH_ENDPOINT = "https://world.openfoodfacts.org/cgi/search.pl"

    def __init__(self):
        requests_cache.install_cache('food_cache', expire_after=6*3600)
        self.session = requests.Session()
        self.request_interval = 0.1
        self.last_request = 0

    def _rate_limited_get(self, url: str, params: dict = None) -> Optional[dict]:
        # Make rate-limited API requests
        elapsed = time.time() - self.last_request
        if elapsed < self.request_interval:
            time.sleep(self.request_interval - elapsed)
        try:
            response = self.session.get(url, params=params, timeout=10)
            response.raise_for_status()
            self.last_request = time.time()
            return response.json()
        except requests.exceptions.RequestException as e:
            print(f"Request Failed: {str(e)}")
            return None

    def search_by_name(self, query: str, max_results: int = 5) -> pd.DataFrame:
```

Executing (0s)

04

Limitations & Future Work

Limitations

- **Current Constraints:**
  - API data completeness varies by region
  - Limited to packaged foods
  - Processing score needs refinement
- **Technical Challenges:**
  - Ingredient disambiguation
  - Regional product variations
  - Real-time data updates

Future Work

- **Nutrition calculator**
  - Calculate calories, fats, carbohydrates, proteins in a meal
  - Plan balanced meals
- **Sustainability Analyzer**
  - Find products with smaller environmental impacts

Thanks!

Do you have any questions?