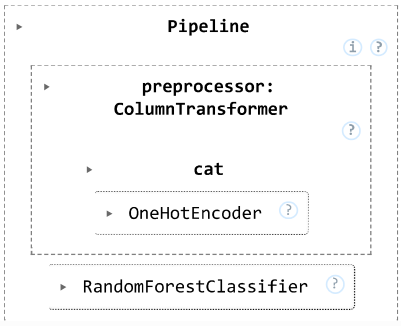
```
import pandas as pd

myfile = pd.read_csv('allergen_alternatives.csv')

myfile.head()
```

	Food Name	Ingredients	Detected Allergens	Risk Level	Notes	Alternative Suggestions
0	Peking Duck	Duck, Soy Sauce, Hoisin Sauce	Soy Sauce	Medium Risk	⚠️ Medium Risk: Contains Soy Sauce. Avoid if a...	NaN
1	Peanut Chikki	Peanuts, Jaggery	Peanuts	Medium Risk	⚠️ Medium Risk: Contains Peanuts. Avoid if all...	Sunflower Butter, Almond Butter
2	Shrimp Fried Rice	Rice, Shrimp, Eggs, Vegetables	Shrimp, Eggs	High Risk	⚠️ High Risk: Contains Shrimp, Eggs. Avoid if...	Chia Seeds, Flaxseeds

```
myfile.describe()
```

	Food Name	Ingredients	Detected Allergens	Risk Level	Notes	Alternative Suggestions
count	10000	10000	10000	10000	10000	2448
unique	31	28	431	2	431	6
top	Sichuan Hot Pot	Wheat, Vegetables, Soy Sauce	Vegetables	High Risk	⚠️ Medium Risk: Contains Vegetables. Avoid if ...	Gluten-Free Flour, Rice Flour

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report


myfile.columns = myfile.columns.str.strip()


# Separate features (X) and target (y)
X = myfile[["Food Name", "Ingredients","Detected Allergens"]]  # Features
y = myfile["Risk Level"]  # Target variable

# Encode the target variable (if it's categorical)
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Encode categorical features
categorical_features = ["Food Name", "Ingredients","Detected Allergens"]
preprocessor = ColumnTransformer(
    transformers=[
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ])


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Train the model
model.fit(X_train, y_train)
```




```
# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

```
print(y_pred)
```

```
[0 0 1 ... 1 0 0]
```

```
import joblib

# Save the model
joblib.dump(model, "allergen_classification_model.pkl")
```

```
['allergen_classification_model.pkl']
```

```
myfile.describe()
```

	Food Name	Ingredients	Detected Allergens	Risk Level	Notes	Alternative Suggestions
count	10000	10000	10000	10000	10000	2448
unique	31	28	431	2	431	6
top	Sichuan Hot Pot	Wheat, Vegetables, Soy Sauce	Vegetables	High Risk	⚠️ Medium Risk: Contains Vegetables. Avoid if ...	Gluten-Free Flour, Rice Flour

```
allergen_alternatives = pd.read_csv('allergen_alternatives.csv')
allergen_alternatives.head()
```

	Food Name	Ingredients	Detected Allergens	Risk Level	Notes	Alternative Suggestions
0	Peking Duck	Duck, Soy Sauce, Hoisin Sauce	Soy Sauce	Medium Risk	⚠️ Medium Risk: Contains Soy Sauce. Avoid if a...	NaN
1	Peanut Chikki	Peanuts, Jaggery	Peanuts	Medium Risk	⚠️ Medium Risk: Contains Peanuts. Avoid if all...	Sunflower Butter, Almond Butter
2	Shrimp Fried Rice	Rice, Shrimp, Eggs, Vegetables	Shrimp, Eggs	High Risk	⚠️ High Risk: Contains Shrimp, Eggs. Avoid if...	Chia Seeds, Flaxseeds

```
allergen_mapping = {
    "Cream": "Dairy",
    "Wheat": "Gluten",
    "Yogurt": "Dairy",
    "Butter": "Dairy",
    "Fish": "Fish",
    "Milk": "Dairy",
    "Coconut": "Coconut",
    "Shrimp": "Shellfish",
    "Paneer": "Dairy",
    "Bread": "Gluten",
    "Tofu": "Soy",
    "Peanuts": "Peanuts",
```

```
    "Soy Sauce": "Soy"
}



def detect_allergens(ingredients):
    detected = set()
    for ingredient, allergen in allergen_mapping.items():
        if ingredient.lower() in ingredients.lower():
            detected.add(allergen)
    return ", ".join(detected) if detected else "None"



alternative_mapping = {
    "Dairy": "Almond milk, Oat milk, Coconut milk, Vegan cheese, Cashew cheese",
    "Eggs": "Flaxseed egg, Chia egg, Applesauce, Mashed banana, Aquafaba",
    "Gluten": "Rice flour, Quinoa flour, Chickpea flour, Corn flour, Sorghum flour",
    "Soy": "Coconut aminos, Chickpeas, Lentils, Hemp seeds, Pea protein",
    "Peanuts": "Sunflower seed butter, Pumpkin seed butter, Tahini, Hemp seed butter",
    "Tree Nuts": "Sunflower seeds, Pumpkin seeds, Hemp seeds, Sesame seeds",
    "Fish": "Jackfruit, Plant-based seafood, Seaweed-based fish alternatives",
    "Shellfish": "Plant-based seafood, Mushrooms, King oyster mushrooms, Artichokes",
    "Sesame": "Sunflower butter, Pumpkin seeds, Poppy seeds, Flaxseeds",
    "Mustard": "Turmeric, Mustard-free dressings, Horseradish, Wasabi",
    "Lupin": "Rice flour, Chickpea flour, Fava bean flour, Lentil flour",
    "Corn": "Rice flour, Potato starch, Arrowroot flour, Tapioca flour",
    "Coconut": "Almond milk, Soy milk, Oat milk, Hemp milk",
    "Sulfites": "Fresh lemon juice, Apple cider vinegar, White vinegar",
    "Artificial Additives": "Natural sweeteners (Stevia, Monk fruit, Date syrup), Fresh herbs",
    "Food Dyes": "Turmeric powder, Beetroot powder, Annatto, Spirulina extract"
}


def suggest_alternatives(detected_allergens):
    allergens = detected_allergens.split(", ")
    alternatives = [alternative_mapping.get(a, "No alternative available") for a in allergens]
    return ", ".join(alternatives) if alternatives else "No alternatives available"



def generate_notes(detected_allergens):
    if detected_allergens == "None":
        return "✅ Safe to consume."
    else:
        return f"⚠️ Contains {detected_allergens}. Avoid if allergic."



# Function to process user input
def get_food_allergy_info(food_name):
    # Search for the food item in the dataset
    food_item = myfile[myfile["Food Name"].str.lower() == food_name.lower()]

    if food_item.empty:
        return f"❌ Food item '{food_name}' not found in the dataset."

    # Get ingredients
    ingredients = food_item["Ingredients"].values[0]

    # Detect allergens, get alternatives, and generate notes
    detected_allergens = detect_allergens(ingredients)
    alternatives = suggest_alternatives(detected_allergens)
    notes = generate_notes(detected_allergens)

    return f"""
    Food Name: {food_name}
    Ingredients: {ingredients}
    Detected Allergens: {detected_allergens}
    Alternative Suggestions: {alternatives}
    Notes: {notes}
    """

# Ask for user input
user_input = input("Enter a food item: ")
result = get_food_allergy_info(user_input)
```

```
print(result)
```

```
Enter a food item: Pav Bhaji

        Food Name: Pav Bhaji
        Ingredients: Vegetables, Butter, Bread
        Detected Allergens: Dairy, Gluten
        Alternative Suggestions: Almond milk, Oat milk, Coconut milk, Vegan cheese, Cashew cheese, Rice flour, Quinoa flour, Chickpea flour,
        Notes: ⚠️ Contains Dairy, Gluten. Avoid if allergic.
```

```
from flask import Flask, render_template, request, session
import pandas as pd

app = Flask(__name__)
app.secret_key = "mysecretkey"  # Required for sessions

# 1. Load Data
try:
    myfile = pd.read_csv('allergen_alternatives.csv')
    meal_plans = pd.read_csv('meal_plans.csv')  # Load meal plans
except FileNotFoundError:
    print("Error: CSV file not found. Please make sure the files are in the correct directory.")
    exit()

# 2. Allergen Mapping (as before)
allergen_mapping = {
    "Cream": "Dairy",
    "Wheat": "Gluten",
    "Yogurt": "Dairy",
    "Butter": "Dairy",
    "Fish": "Fish",
    "Milk": "Dairy",
    "Coconut": "Coconut",
    "Shrimp": "Shellfish",
    "Paneer": "Dairy",
    "Bread": "Gluten",
    "Tofu": "Soy",
    "Peanuts": "Peanuts",
    "Soy Sauce": "Soy"
}

# 3. Alternative Mapping (as before)
alternative_mapping = {
    "Dairy": "Almond milk, Oat milk, Coconut milk, Vegan cheese, Cashew cheese",
    "Eggs": "Flaxseed egg, Chia egg, Applesauce, Mashed banana, Aquafaba",
    "Gluten": "Rice flour, Quinoa flour, Chickpea flour, Corn flour, Sorghum flour",
    "Soy": "Coconut aminos, Chickpeas, Lentils, Hemp seeds, Pea protein",
    "Peanuts": "Sunflower seed butter, Pumpkin seed butter, Tahini, Hemp seed butter",
    "Tree Nuts": "Sunflower seeds, Pumpkin seeds, Hemp seeds, Sesame seeds",
    "Fish": "Jackfruit, Plant-based seafood, Seaweed-based fish alternatives",
    "Shellfish": "Plant-based seafood, Mushrooms, King oyster mushrooms, Artichokes",
    "Sesame": "Sunflower butter, Pumpkin seeds, Poppy seeds, Flaxseeds",
    "Mustard": "Turmeric, Mustard-free dressings, Horseradish, Wasabi",
    "Lupin": "Rice flour, Chickpea flour, Fava bean flour, Lentil flour",
    "Corn": "Rice flour, Potato starch, Arrowroot flour, Tapioca flour",
    "Coconut": "Almond milk, Soy milk, Oat milk, Hemp milk",
    "Sulfites": "Fresh lemon juice, Apple cider vinegar, White vinegar",
    "Artificial Additives": "Natural sweeteners (Stevia, Monk fruit, Date syrup), Fresh herbs",
    "Food Dyes": "Turmeric powder, Beetroot powder, Annatto, Spirulina extract"
}

# 4. Allergen Detection Function (as before)
def detect_allergens(ingredients):
    detected = set()
    for ingredient, allergen in allergen_mapping.items():
        if ingredient.lower() in ingredients.lower():
            detected.add(allergen)
    return ", ".join(detected) if detected else "None"

# 5. Alternative Suggestion Function (as before)
def suggest_alternatives(detected_allergens):
    allergens = detected_allergens.split(", ")
    alternatives = [alternative_mapping.get(a, "No alternative available") for a in allergens]
    return ", ".join(alternatives) if alternatives else "No alternatives available"
```

```
# 6. Notes Generation Function (as before)
def generate_notes(detected_allergens):
    if detected_allergens == "None":
        return "Safe to consume."
    else:
        return f"Contains {detected_allergens}. Avoid if allergic."

# 7. Food Allergy Information Retrieval Function (modified)
def get_food_allergy_info(food_name):
    food_item = myfile[myfile["Food Name"].str.lower() == food_name.lower()]
    if food_item.empty:
        return None, None, None, f"Food item '{food_name}' not found in the dataset."

    ingredients = food_item["Ingredients"].values[0]
    detected_allergens = detect_allergens(ingredients)
    alternatives = suggest_alternatives(detected_allergens)
    notes = generate_notes(detected_allergens)
    risk_level = food_item["Risk Level"].values[0] if "Risk Level" in food_item else "Not Available"
    return ingredients, detected_allergens, alternatives, notes

# 8. Chatbot Logic and Flask Routes
@app.route('/')
def home():
    session.clear()  # Clear session on new visit
    return render_template('chat.html')

@app.route('/get_response', methods=['POST'])
def get_response():
    user_input = request.form['user_input']
    response = process_input(user_input)
    return response

def process_input(user_input):
    if 'allergies' not in session:
        session['allergies'] =

    if "I am allergic to" in user_input.lower():
        allergens = user_input.lower().split("i am allergic to")[-1].strip().split(",")
        allergens = [allergen.strip() for allergen in allergens]
        session['allergies'].extend(allergens)
        return f"Okay, I have noted that you are allergic to {', '.join(session['allergies'])}. What else?"

    elif "suggest meal plan" in user_input.lower():
        if not session['allergies']:
            return "Please tell me your allergies first."
        else:
            suggested_plans = suggest_meal_plans(session['allergies'])
            if suggested_plans:
                return "Based on your allergies, I suggest these meal plans: " + ", ".join(suggested_plans)
            else:
                return "I couldn't find any meal plans that fit your allergies. Please check back later."

    elif "what can i eat" in user_input.lower():
         if not session['allergies']:
            return "Please tell me your allergies first."
         else:
            safe_foods = suggest_safe_foods(session['allergies'])
            if safe_foods:
                return "Based on your allergies, you can try these food: " + ", ".join(safe_foods)
            else:
                return "I couldn't find any meal plans that fit your allergies. Please check back later."

    elif "goodbye" in user_input.lower():
        session.clear()
        return "Goodbye! Have a healthy day!"

    else:
        return "I'm still learning. I can currently help with allergy information and meal plan suggestions. Try asking about your allergies o

def suggest_meal_plans(allergies):
    """Suggests meal plans based on user's allergies."""
    valid_plans =
    for index, plan in meal_plans.iterrows():
        plan_avoids = plan['Allergens to Avoid'].split(", ")
        safe_plan = True
        for allergen in allergies:
            if allergen in plan_avoids:
```

```
                safe_plan = False
                break
        if safe_plan:
            valid_plans.append(plan['Meal Plan Name'])
    return valid_plans

def suggest_safe_foods(allergies):
    """Suggests safe foods based on user's allergies."""
    safe_foods =
    for index, food in myfile.iterrows():
        food_allergens = food['Detected Allergens'].split(", ")
        safe_food = True
        for allergen in allergies:
            if allergen in food_allergens:
                safe_food = False
                break
        if safe_food:
            safe_foods.append(food['Food Name'])
    return safe_foods

if __name__ == '__main__':
    app.run(debug=True)
```

```
  File "<ipython-input-130-5785b83f2fd8>", line 100
    session['allergies'] =
                          ^
SyntaxError: invalid syntax
```