Culinary Heritage and AI

By Florent Cadet

Interactive application

Dataset (CSV File)

```
,Recipe_Name,Ingredients,Dish_Name
0,Einkorn Onion Rings,"onion,flour,baking,salt,egg,milk,bread,coconut,salt",onion_rings
1,Crispy Onion Rings,"onion,panko,cornmeal,paprika,onion,garlic,cayenne,salt,flour,egg,oil",onion_rings
2,Onion Rings,"onion,flour,baking,egg,bread,oil",onion_rings
3,Onion Rings_2,"oil,cake,onion",onion_rings
4,Onion Rings_3,"nut,onion,flour,egg,celery,baking,ale",onion_rings
5,Crispy Onion Rings_2,"onion,flour,salt,baking,cold",onion_rings
6,Spicy Buttermilk Onion Rings,"onion,butter,flour,salt,black",onion_rings
7,Best Ever Onion Rings,"egg,milk,flour,baking,onion,oil,onion",onion_rings
8,Onion Rings_4,"onion,flour,baking,egg,bread,oil",onion_rings
9,Classic Onion Rings,"onion,flour,soda,egg,cayenne",onion_rings
10,BACON WRAPPED ONION RINGS,"onion,bacon,black",onion_rings
11,Low Fat Baked Onion Rings,"onion,butter,bread,bread,corn,salt,baking",onion_rings
12,Homemade Onion Rings,"onion,flour,baking,salt,egg,milk,bread,oil,salt",onion_rings
13,Baked Onion Rings,"onion,flour,bread,egg,cajun",onion_rings
```

Dataset

```
def read_csv_and_save_to_db(file_path):
    with open(file_path, 'r') as file:
        reader = csv.DictReader(file)
        for row in reader:
            ingredient = row['Ingredients'].replace(",&", "")
            if ingredient != "":
                Recipe.objects.create(
                    recipe_name=row['Recipe_Name'],
                    ingredients=row['Ingredients'],
                    dish_name=row['Dish_Name']
                )


def clean_all_recipes():
    Recipe.objects.all().delete()
```

Task 1:

Create a graph to present the most popular ingredients

```
def get_bar_data(recipes):
    ingredients_count = Counter()
    for recipe in recipes:
        ingredients = recipe.ingredients.split(',')
        ingredients_count.update(ingredients)

    data = [{"label": ingredient, "y": count} for ingredient, count in
ingredients_count.most_common()]
    return data
```

Task 2:

Allow the user to select a Dish. Based on their selection, you should create a graph to present the most popular ingredients for that Dish.

```
selected_dish: str = request.GET.get('dish', None)
selected_recipe: str = request.GET.get('recipe', None)
if selected_dish:
    recipes = Recipe.objects.filter(dish_name=selected_dish)
    recipes_selector = recipes.order_by('recipe_name')
    if selected_recipe:
        recipes = recipes.filter(recipe_name=selected_recipe)
```

Task 3:

Tokenise ingredients to represent each “Ingredients” record with a binary vector (you can do that by using sklearn and the DictVectorizer function.)

```
def tokenize(recipes):
    ingredients_list = [recipe.ingredients.split(',') for recipe in recipes]
    ingredients_binary = []
    for ingredients in ingredients_list:
        binary_vector = {ingredient: 1 for ingredient in ingredients}
        ingredients_binary.append(binary_vector)

    vectorizer: DictVectorizer = DictVectorizer(sparse=False)
    X: np.ndarray = vectorizer.fit_transform(ingredients_binary)
    return X
```

Task 4:

Based on the binary representation of “Ingredients” cluster together, similar dishes and use any dimensionality reduction technique you like to plot the derived clusters onto a two dimensional space.

```
def get_centroid(recipes=None):
    centroids = []
    selected_dish = recipes[0].dish_name if recipes else None
    if not recipes:
        dishes = Recipe.objects.values_list('dish_name', flat=True).distinct()
    else:
        dishes = Recipe.objects.values_list('dish_name',
flat=True).distinct().filter(dish_name=selected_dish)
    for dish in dishes:
        dish_recipes = Recipe.objects.filter(dish_name=dish)

        X_reduced = get_PCA_data(dish_recipes)
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X_reduced)
        centroid = np.mean(X_scaled, axis=0)
        centroids.append({"dish": dish, "centroid": centroid})
    return centroids
```

Task 4:

Based on the binary representation of “Ingredients” cluster together, similar dishes and use any dimensionality reduction technique you like to plot the derived clusters onto a two dimensional space.

```
def find_N_closer_and_far_centroid_of_centroid (centroid_center, centroids, n=5):
    distances = []
    for dish_centroid in centroids:
        centroid = dish_centroid["centroid"]
        dish = dish_centroid["dish"]
        print(centroid_center["centroid"], centroid)
        if centroid_center["dish"] == dish:
            continue
        dist = distance.euclidean(centroid_center["centroid"], centroid)
        distances.append({
            "dish": dish,
            "distance": dist
        })
    distances.sort(key=lambda x: x['distance'])
    closer_recipes = distances[:n]
    far_recipes = distances[-n:]
    far_recipes.reverse()
    return closer_recipes, far_recipes
```

Task 4:

Based on the binary representation of “Ingredients” cluster together, similar dishes and use any dimensionality reduction technique you like to plot the derived clusters onto a two dimensional space.

```
def get_centroid_centroid_plot(centroids: list, selected_dish_centroid: dict = None, closer_centroid: list[dict] = None, far_centroid:
list[dict] = None):
    result = []
    i = 0
    for dish_centroid in centroids:
        dish = dish_centroid["dish"]
        centroid = dish_centroid["centroid"]
        is_nearest = True if closer_centroid and dish in [c_centroid['dish'] for c_centroid in closer_centroid] else False
        is_farthest = True if far_centroid and dish in [c_centroid['dish'] for c_centroid in far_centroid] else False
        nearest_or_farthest = "Nearest <br>" if is_nearest else "Farthest <br>" if is_farthest else ""
        if selected_dish_centroid:
            color = COLOR[1]["hex"] if selected_dish_centroid["dish"] == dish else COLOR[4]["hex"] if is_nearest else COLOR[0]["hex"] if
is_farthest else COLOR[2]["hex"]
        else:
            color = COLOR[i%len(COLOR)]["hex"]
        result.append({
            "x": centroid[0],
            "y": centroid[1],
            "label": f"{nearest_or_farthest} Dish: {dish}",
            "color": color
        })
        i+=1
    return result
```

Task 5 and 6 (Attempt 1):

Allow the user to select a Dish. Based on their selection, find present the most similar and the most dissimilar recipes (dynamic)

```
def get_similarity_recipes(recipes):
    similarity_matrix = cosine_similarity(tokenize(recipes))
    return similarity_matrix
```

Task 5 and 6 (Attempt 1):

Allow the user to select a Dish. Based on their selection, find present the most similar and the most dissimilar recipes (dynamic)

```
def find_similar_recipes(recipes, similarity_matrix):
    similar_recipes = []
    recipes_list = list(recipes)
    nb = 5
    for i, recipe in enumerate(recipes_list):
        similar_indices: int = np.argsort(similarity_matrix[i])[::-1][1:nb+1]
        similarities = [{"recipe": recipes_list[index], "similarity": similarity_matrix[i][index]}
               for index in similar_indices if recipes_list[index].recipe_name != recipe.recipe_name]
        avg_similarity = sum(similarity["similarity"] for similarity in similarities) / nb
        similar_recipes.append({"recipe": recipe, "avg_similarity": avg_similarity, "data":
similarities})
    similar_recipes.sort(key=lambda x: x['avg_similarity'], reverse=True)
    return similar_recipes
```

Task 5 and 6 (Attempt 1):

Allow the user to select a Dish. Based on their selection, find present the most similar and the most dissimilar recipes (dynamic)

```
def find_dissimilar_recipes(recipes, similarity_matrix):
    dissimilar_recipes = []
    recipes_list = list(recipes)
    nb = 5
    for i, recipe in enumerate(recipes):
        dissimilar_indices: int = np.argsort(similarity_matrix[i])[::-1][-nb:]
        dissimilarities = [{"recipe": recipes_list[index], "similarity": similarity_matrix[i][index]}
            for index in dissimilar_indices if recipes_list[index].recipe_name != recipe.recipe_name]
        avg_dissimilarity = sum(dissimilarity["similarity"] for dissimilarity in dissimilarities)/nb
        dissimilar_recipes.append({"recipe": recipe, "avg_dissimilarity": avg_dissimilarity, "data":
dissimilarities})

    dissimilar_recipes.sort(key=lambda x: x['avg_dissimilarity'], reverse=False)
    return dissimilar_recipes
```

Task 5 and 6 (Attempt 2):

Allow the user to select a Dish. Based on their selection, find present the most similar and the most dissimilar recipes (dynamic)

```
def get_centroid(recipes=None):
    centroids = []
    selected_dish = recipes[0].dish_name if recipes else None
    if not recipes:
        dishes = Recipe.objects.values_list('dish_name', flat=True).distinct()
    else:
        dishes = Recipe.objects.values_list('dish_name',
flat=True).distinct().filter(dish_name=selected_dish)
    for dish in dishes:
        dish_recipes = Recipe.objects.filter(dish_name=dish)

        X_reduced = get_PCA_data(dish_recipes)
        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X_reduced)
        centroid = np.mean(X_scaled, axis=0)
        centroids.append({"dish": dish, "centroid": centroid})
    return centroids
```

Task 5 and 6 (Attempt 2):

Allow the user to select a Dish. Based on their selection, find present the most similar and the most dissimilar recipes (dynamic)

```
def find_N_closer_and_far_recipe_of_centroid(centroid, recipes, n=5):
    distances = []
    recipes_point = get_PCA_data(recipes)
    for recipe_point, recipe in zip(recipes_point, recipes):
        dist = distance.euclidean(centroid, recipe_point)
        distances.append({
            "recipe": recipe,
            "distance": dist
        })

    distances.sort(key=lambda x: x['distance'])
    closer_recipes = distances[:n]
    far_recipes = distances[-n:]
    far_recipes.reverse()
    return closer_recipes, far_recipes
```

Task 5 and 6 (Attempt 2):

Allow the user to select a Dish. Based on their selection, find present the most similar and the most dissimilar recipes (dynamic)

```
def get_centroid_recipes_plot (centroids, recipes: list = None, closer_recipes: list[dict] = None, far_recipes: list[dict] = None):
    selected_dish = recipes[0].dish_name if recipes else None
    result = []
    for dish_centroid in centroids:
        color = COLOR[1]["hex"] if selected_dish == dish_centroid["dish"] else COLOR[3]["hex"]
        result.append({
            "x": dish_centroid["centroid"][0],
            "y": dish_centroid["centroid"][1],
            "label": f"Centroid <br> Dish: {dish_centroid["dish"]}",
            "color": color
        })
    if recipes:
        X_reduced = get_PCA_data(recipes)
        for i, recipe in enumerate(recipes):
            is_nearest = True if closer_recipes and recipe.recipe_name in [recipe['recipe'].recipe_name for recipe in closer_recipes] else False
            is_farthest = True if far_recipes and recipe.recipe_name in [recipe['recipe'].recipe_name for recipe in far_recipes] else False
            nearest_or_farthest = "Nearest <br>" if is_nearest else "Farthest <br>" if is_farthest else ""
            result.append({
                "x": X_reduced[i][0],
                "y": X_reduced[i][1],
                "label": f"{nearest_or_farthest} Recipe: {recipe.recipe_name} <br> Ingredients: {recipe.ingredients}",
                "color": COLOR[4]["hex"] if is_nearest else COLOR[0]["hex"] if is_farthest else COLOR[2]["hex"]
            })
    return result
```

Thank you

Now it's time for the live presentation