1 Room for new Information

1.1 Extracting Restaurants, Fast Food and Cafes from OpenStreetMap

For extracting the data from OpenStreetMap, we utilized the Overpass API. The used query is as follows:

```
[out:json][timeout:180];
area["name"="{name}"]["boundary"="administrative"]["admin_level"="4"]->.a;
(
  node["amenity"="restaurant"](area.a);
  node["amenity"="cafe"](area.a);
  node["amenity"="fast_food"](area.a);
);
out center;
```

1.1.1 Explanation of the query:

1.1.1.1 1. Query Output and Timeout Settings

`[out:json][timeout:180];`

- `out:json`: Sets the output format to **JSON**.
- `timeout:180`: Sets a **timeout limit of 180 seconds** for the query to run, useful for large or slow queries.

1.1.1.2 2. Select the Area

```
area["name"="{name}"]["boundary"="administrative"]["admin_level"="4"]->.a;
```

- This finds an **administrative area**:
  - With name `{name}` (e.g. "Germany" or "Zurich" – replace with the actual name).
  - With `boundary=administrative` (only administrative boundaries).
  - With `admin_level=4` (typically a region/state-level boundary).
- `->.a;`: Saves the matched area into a variable `.a`.

Note: The area is **not the same** as a polygon in the map. Internally, Overpass assigns IDs to areas derived from OSM relations.

1.1.1.3 3. Find Nodes Within That Area

```
(
  node["amenity"="restaurant"](area.a);
  node["amenity"="cafe"](area.a);
  node["amenity"="fast_food"](area.a);
);
```

- This block fetches **nodes** (points) that:
  - Have `amenity=restaurant`, `amenity=cafe`, or `amenity=fast_food`.
  - Are **located within the area `.a`** defined above.
- Parentheses group the different queries together so the result includes all three types.

1.1.1.4 4. Output the Results

```
out center;
```

- `out center`: Outputs each matching object with its **center coordinates**.
  - `center` is typically used for areas (ways/relations), but if only nodes are returned, the output is similar to `out body`.

1.1.2 Result of the API Query

For our example we extracted all the restaurants from all the cantons in Switzerland. This gives us the following list of JSON files:

```
cantons/
  restaurants_Aargau.json
  restaurants_Appenzell_Ausserrhoden.json
  restaurants_Appenzell_Innerrhoden.json
  restaurants_Basel-Landschaft.json
  restaurants_Basel-Stadt.json
  restaurants_Bern_Berne.json
  restaurants_Fribourg_Freiburg.json
  restaurants_Genève.json
  restaurants_Glarus.json
  restaurants_Graubünden_Grischun_Grigioni.json
  restaurants_Jura.json
  restaurants_Luzern.json
  restaurants_Neuchâtel.json
  restaurants_Nidwalden.json
  restaurants_Obwalden.json
  restaurants_Schaffhausen.json
  restaurants_Schwyz.json
  restaurants_Solothurn.json
  restaurants_St._Gallen.json
  restaurants_Thurgau.json
  restaurants_Ticino.json
  restaurants_Uri.json
  restaurants_Valais_Wallis.json
  restaurants_Vaud.json
  restaurants_Zug.json
  restaurants_Zürich.json
```

After combing all the JSON into one single JSON file, the data can be analyzed. The following statistics were generated:

```
Restaurant Statistics - restaurants_Switzerland.json

Total gathered: 20977
- Restaurants: 14917
- Fast food: 2710
- Cafes: 3350
```

```
Additional information:
- With URL: 8924 (42.54%)
- With cuisine type: 9440 (45.00%)
- With URL and cuisine type: 5064 (24.14%)
```

1.1.2.1 Conclusion of the Analysis

Based on the analysis it is visible that only 50% percent of all the restaurants e.g. have an URL or a cuisine type. And only 28% have both. Even with the missing data, the dataset can still be used to test or verify the FoodClassifier.

1.1.2.2 Extracting Cuisine Types

For the next steps, the cuisine types needs to be extracted from the JSON file. The FoodClassifier will use these cuisine types as labels for the training data and also will be used as result of the module.

The simplest way to extract the cuisine types is to use a set, which will automatically remove duplicates. The cuisine types are stored in the `cuisines` set.

```python
import json

with open("restaurants_Switzerland.json", "r", encoding="utf-8") as f:
    data = json.load(f)

# prepare set for the cuisine type
cuisines = set()

for element in data.get("elements", []):
    tags = element.get("tags", {})
    cuisine = tags.get("cuisine")
    if cuisine:
        # split multiple types
        types = [c.strip() for c in cuisine.split(";")]
        cuisines.update(types)
```

Now we take a look at the extracted cuisine types:

```
Central American
Gourmet
```

```
Grill
Pains
Pizza & Grill
Schnitzel
Southern_BBQ
Texas_Barbecue
afghan
african
american
...
homemade
hot_dog
https://labelfaitmaison.ch/de/restaurant/roba-buona-2/
ice_cream
indian
...
```

We see the cuisine types are not really normalized. For example: * Uppercase vs. lowercase * Spaces vs. underscores * Types seperated with '&' * URLs in the cuisine type

Now we need to update the python code to normalize the cuisine types, with the following rules: * Convert to lowercase * Replace spaces with underscores * Additionally, split types separated by '&' and add them as separate entries * Remove URLs from the cuisine type

This leads to the following updated code:

```python
import re
import json

with open("restaurants_Switzerland.json", "r", encoding="utf-8") as f:
    data = json.load(f)

# prepare set for the cuisine type
cuisines = set()

for element in data.get("elements", []):
    tags = element.get("tags", {})
    cuisine = tags.get("cuisine")
    if cuisine:
        # split multiple types
        types = [c.strip() for c in re.split(r'[;&]', cuisine)]

        cleaned_types = [
```

```
            t.replace(" ", "_").replace("-", "_").lower() # clean up the types
            for t in types
            if not t.strip().lower().startswith("http") # filter out URLs
        ]
        cuisines.update(cleaned_types)
```

This leads to the following normalized cuisine types with 386 unique entries:

```
afghan
african
alp
alpine_hut
american
arab
argentinian
asian
austrian
ayran
bacon
bagel
bakery
baklawa
balkan
bangladeshi
bar
bar_and_grill
...
```

We can still see some cuisine types doesn't really match our expectations, like "alp", "alpine_hut" isn't really a cuisine type.

1.2 Finalizing the Cuisine Types

Like mentioned before, the cuisine types are used as labels for the training data and also will be used as result of the module. So we need to finalize the cuisine types and currently the extracted cuisine types from the OpenStreetMap data are not really suitable for that. 1. Define what should be included in the cuisine types. 2. Gather all the cuisine types and save it to a file.

1.2.1 Defining the Cuisine Types

For simplicity, we will only include countries. Regions, cities or other types are at the current moment a little bit too specific and because of that we will limit ourselves only to the countries.

1.2.2 Gathering the Cuisine Types

The first step to get all the countries, like before we use the Overpass API to extract the countries from OpenStreetMap. That way we can use the country names an make an API call to `restcountries.com` to get the matching demonyms.

```
import requests

def get_demonym(country_name):
    try:
        response = requests.get(f"https://restcountries.com/v3.1/name/{country_name}")
        data = response.json()
        # include small random delay
        return data[0]["demonyms"]["eng"]["m"]
    except Exception as e:
        print(f"Error fetching demonym for {country_name}: {e}")
        return "Unknown"


countries = [] # is filled with the country names
# remove duplicates and sort the list
countries = sorted(set(countries))

# get the demonym for each country
countries_demonyms = [get_demonym(country) for country in countries]
```

After all the demonyms are gathered, we can use the following code to sanitize and normalize the demonyms:

```
# remove duplicates
countries_demonyms = list(set(countries_demonyms))

# remove the "Unkown" demonym
countries_demonyms = [d for d in countries_demonyms if d != "Unknown"]
```

```
# split the demonyms by comma
for cd in countries_demonyms:
    if "," in cd:
        parts = cd.split(",")
        countries_demonyms.remove(cd)
        countries_demonyms.extend([part.strip() for part in parts if part.strip()])

# normalize the demonyms
countries_demonyms = [d.replace(" ", "_").lower() for d in countries_demonyms]

# sort the demonyms
countries_demonyms = sorted(countries_demonyms)
```

The resulting list of demonyms is as follows:

```
afghan
albanian
algerian
american_islander
american_samoan
andorran
angolan
anguillian
antiguan
...
```