MongoDB Integration and Queries

Analysis and Ranking of Milan’s Neighborhoods

Edoardo Olivieri, Federica Romano, Francesca Verna

27/01/2025

Contents

```
from pymongo import MongoClient
import geopandas as gpd
from shapely.geometry import shape, mapping
import folium
import math
from pprint import pprint
```

Integration

Connection to MongoDB and loading the files

```
# Connection to MongoDB
client = MongoClient("mongodb://admin:DataMan2023!@localhost:27017/")
db = client["my_database"]
collection = db["neighborhoods"]
```

```
# Reading the files
gdf_combined = gpd.read_file("C:/Users/edoar/combined_quartieri.geojson")
PolyHomePrices = gpd.read_file("C:/Users/edoar/PolyHomePrices.geojson")
PolyRestaurants = gpd.read_file("C:/Users/edoar/PolyRestaurants.geojson")
PolyMuseums = gpd.read_file("C:/Users/edoar/PolyMuseums.geojson")
PolyNightlife = gpd.read_file("C:/Users/edoar/PolyNightlife.geojson")
PolyDogParks = gpd.read_file("C:/Users/edoar/PolyDogParks.geojson")
PolyPharmacy = gpd.read_file("C:/Users/edoar/PolyPharmacy.geojson")
PolyPlaygrounds = gpd.read_file("C:/Users/edoar/PolyPlaygrounds.geojson")
PolySportVenues = gpd.read_file("C:/Users/edoar/PolySportVenues.geojson")
PolySchools = gpd.read_file("C:/Users/edoar/PolySchools.geojson")
PolyUniversity = gpd.read_file("C:/Users/edoar/PolyUniversity.geojson")
PolyCoworking = gpd.read_file("C:/Users/edoar/PolyCoworking.geojson")
PolyLibraries = gpd.read_file("C:/Users/edoar/PolyLibraries.geojson")
PolySupermarkets = gpd.read_file("C:/Users/edoar/PolySupermarkets.geojson")
PolyTransport = gpd.read_file("C:/Users/edoar/PolyTransport.geojson")
```

Initializing the dictionary

```
# Create a dictionary to hold all neighborhood docs
neighborhood_docs = {}
# populating the base neighborhood documents
for idx, row in gdf_combined.iterrows():
```

```
    nb_name = row["Neighborhood"]
    # using shapely's "mapping function" from shapely.geometry to convert geometries
    # to a geojson-like dictionary to store them in MongoDB
    geo_json = mapping(row["geometry"])
    neighborhood_docs[nb_name] = {
        "_id": nb_name,
        "neighborhood_name": nb_name,
        "geometry": geo_json,
        "locations": {
            "restaurants": [],
            "museums": [],
            "nightlife": [],
            "dogparks": [],
            "pharmacies": [],
            "playgrounds": [],
            "sportvenues": [],
            "schools": [],
            "universities": [],
            "coworking": [],
            "libraries": [],
            "supermarkets": [],
            "transport": []
        },
        "home_prices": {
            "min_price": None,  # placeholder for the min price
            "max_price": None,  # placeholder for the max price
            "avg_price": None   # placeholder for the avg price
        }
    }
```

Function to append rows

```
# Function to append rows from a POI GeoDataFrame to the neighborhood_docs
def append_to_neighborhoods(field, gdf, poi_key, field_mappings=None):

    #field is the sub-document where the list will be appended to.
    #gdf is a GeoDataFrame with columns ["Neighborhood", ...#data columns...].
    #poi_key: e.g. "pharmacies", "restaurants", etc.
    #field_mappings is the dict of { "source_column": "destination_field_name", ... } used to pick and rename columns from the gdf row.

    if field_mappings is None:
        # if not supplied, just store all columns except geometrya and Neighborhood
        field_mappings = {
```

```
            col: col
            for col in gdf.columns
            if col not in ("Neighborhood", "geometry")
        }

    # group by Neighborhood to handle rows for each neighborhood
    grouped = gdf.groupby("Neighborhood")

    # for each neighborhood:
    for nb_name, group_df in grouped:
        # if neighborhood is not in the dict, skip it
        if nb_name not in neighborhood_docs:
            continue

        # convert each row to a dictionary with the needed fields
        for _, row in group_df.iterrows():
            poi_data = {}
            for src_col, dest_col in field_mappings.items():
                if src_col in row:
                    poi_data[dest_col] = row[src_col]

            # append to the correct list inside the correct field
            neighborhood_docs[nb_name][field][poi_key].append(poi_data)
```

Appending the GeoDataFrames

```
# Append each of the 13 POI DataFrames to the base neighborhood docs

# Restaurants
append_to_neighborhoods(
    field="locations",
    gdf=PolyRestaurants,
    poi_key="restaurants",
    field_mappings={
        "Business Name": "name",
        "Business Address": "address",
        "Categories": "category",
        "Average Star Rating": "avg_star_rating",
        "Review Count": "tot_ratings",
        "Price": "price"
    }
)
# Museums
append_to_neighborhoods(
    field="locations",
    gdf=PolyMuseums,
    poi_key="museums",
    field_mappings={
        "Museum Name": "name",
```

```
        "Museum Address": "address",
        "Categories": "category",
        "Average Star Rating": "avg_star_rating",
        "Review Count": "tot_ratings"
    }
)
# Nightlife
append_to_neighborhoods(
    field="locations",
    gdf=PolyNightlife,
    poi_key="nightlife",
    field_mappings={
        "Venue Name": "name",
        "Venue Address": "address",
        "Categories": "category",
        "Average Star Rating": "avg_star_rating",
        "Review Count": "tot_ratings"
    }
)
# Dog Parks
append_to_neighborhoods(
    field="locations",
    gdf=PolyDogParks,
    poi_key="dogparks",
    field_mappings={
        "località": "name",
        "area_mq": "area_mq",
        "perim_m": "perimeter_m",
        "obj_id": "park_id",
        "municipio": "municipality"
    }
)
# Pharmacies
append_to_neighborhoods(
    field="locations",
    gdf=PolyPharmacy,
    poi_key="pharmacies",
    field_mappings={
        "DESCRIZIONE_FARMACIA": "name",
        "INDIRIZZO": "address",
        "CODICE_FARMACIA": "pharmacy_id",
        "MUNICIPIO": "municipality"
    }
)
# Playgrounds
append_to_neighborhoods(
    field="locations",
    gdf=PolyPlaygrounds,
    poi_key="playgrounds",
    field_mappings={
        "località": "name",
```

```
        "area_mq": "area_mq",
        "perim_m": "perimeter_m",
        "obj_id": "park_id",
        "municipio": "municipality"
    }
)
# Sport Venues
append_to_neighborhoods(
    field="locations",
    gdf=PolySportVenues,
    poi_key="sportvenues",
    field_mappings={
        "Nome": "name",
        "Indirizzo": "address",
        "info": "category",
    }
)
# Schools
append_to_neighborhoods(
    field="locations",
    gdf=PolySchools,
    poi_key="schools",
    field_mappings={
        "DENOMINAZIONESCUOLA": "name",
        "INDIRIZZOSCUOLA": "address",
        "DESCRIZIONECARATTERISTICASCUOLA": "school_type",
        "DESCRIZIONETIPOLOGIAGRADOISTRUZIONESCUOLA": "educational_lvl",
        "MUNICIPIO": "municipality"
    }
)
# Universities
append_to_neighborhoods(
    field="locations",
    gdf=PolyUniversity,
    poi_key="universities",
    field_mappings={
        "DENOMINAZ": "name",
        "INDIRIZZO": "address",
        "FACOLTA": "faculty",
        "PROPRIETA": "ownership_type",
        "MUNICIPIO": "municipality"
    }
)
# Coworking
append_to_neighborhoods(
    field="locations",
    gdf=PolyCoworking,
    poi_key="coworking",
    field_mappings={
        "SPAZIO": "name",
        "Sede": "address",
```

```
126             "Orario di apertura": "opening_hrs",
127             "Numero postazioni": "tot_desks",
128             "MUNICIPIO": "municipality"
129         }
130 )
131 # Libraries
132 append_to_neighborhoods(
133         field="locations",
134         gdf=PolyLibraries,
135         poi_key="libraries",
136         field_mappings={
137             "Biblioteche - Sede": "name",
138             "Indirizzo": "address",
139             "MUNICIPIO": "municipality"
140         }
141 )
142
143 # Supermarkets
144 append_to_neighborhoods(
145         field="locations",
146         gdf=PolySupermarkets,
147         poi_key="supermarkets",
148         field_mappings={
149             "name": "name"
150         }
151 )
152
153 # Transport
154 append_to_neighborhoods(
155         field="locations",
156         gdf=PolyTransport,
157         poi_key="transport",
158         field_mappings={
159             "Nome": "name",
160             "Linee": "lines",
161             "Mezzo": "transport_type"
162         }
163 )
164
165 # Home Prices
166 for idx, row in PolyHomePrices.iterrows():
167         nb_name = row["Neighborhood"]
168         if nb_name in neighborhood_docs:
169             # updateing the home_prices container with actual values
170             neighborhood_docs[nb_name]["home_prices"]["min_price"] = row["
                Compr_min"]
171             neighborhood_docs[nb_name]["home_prices"]["max_price"] = row["
                Compr_max"]
172             neighborhood_docs[nb_name]["home_prices"]["avg_price"] = row["
                Compr_mean"]
```

Storing the data into MongoDB

```
# Insert the data into MongoDB
# (neighborhood_docs is a dictionary with neighborhood_name as keys and the final documents as values)
documents_to_insert = list(neighborhood_docs.values())  # conversion to list of dicts
collection.insert_many(documents_to_insert)
print("Data inserted into MongoDB")
```

```
Data inserted into MongoDB
```

Queries

Neighborhood (Example)

Here a basic query is presented to show the structure of the final DB.

```
# Query for the neighborhood "Tre Torri"
tretorri_data = collection.find_one({"neighborhood_name": "Tre Torri"})
pprint(tretorri_data)
```

```
{'_id': 'Tre Torri',
 'geometry': {'coordinates': [[[9.1598895, 45.4742524],
                               [9.1599665, 45.4752257],
                               [9.159981, 45.475719],
                               [9.1600586, 45.4772662],
                               [9.1600766, 45.4774377],
                               [9.1600869, 45.4779766],
                               [9.1601359, 45.4786687],
                               [9.1602194, 45.4800522],
                               [9.1602396, 45.4801317],
                               [9.1602753, 45.4803015],
                               [9.1600595, 45.4803093],
                               [9.1586847, 45.4803591],
                               [9.1559822, 45.4803917],
                               [9.1516378, 45.4804495],
                               [9.1516812, 45.4802959],
                               [9.1516687, 45.4800506],
                               [9.1515654, 45.4780585],
                               [9.1513698, 45.4744323],
                               [9.1511429, 45.4743509],
                               [9.1505715, 45.4739819],
                               [9.1515806, 45.4731998],
                               [9.1516232, 45.4736605],
                               [9.1527203, 45.4736241],
                               [9.1537328, 45.4736061],
```

```
                                  [9.1555551, 45.4735608],
                                  [9.1567836, 45.4735338],
                                  [9.1573631, 45.4735149],
                                  [9.158371, 45.4734854],
                                  [9.1598245, 45.4734469],
                                  [9.1598895, 45.4742524]]],
                'type': 'Polygon'},
 'home_prices': {'avg_price': 7687.5,
                 'max_price': 12600.0,
                 'min_price': 3500.0},
 'locations': {'coworking': [],
               'dogparks': [{'area_mq': 1007.6380499947832,
                             'municipality': 8,
                             'name': 'piazza Giulio Cesare',
                             'park_id': 25448,
                             'perimeter_m': 126.55090634925524}],
               'libraries': [],
               'museums': [],
               'nightlife': [{'address': 'Piazzale Arduino 1',
                              'avg_star_rating': 4.0,
                              'category': 'Cocktail Bars',
                              'name': 'GUD',
                              'tot_ratings': 4},
                             {'address': 'Piazza Tre Torri',
                              'avg_star_rating': 5.0,
                              'category': 'Wine Bars, Venues & Event
                                 Spaces, '
                                          'Cafes',
                              'name': 'Peck City Life',
                              'tot_ratings': 1},
                             {'address': 'Piazza Tre Torri 1L',
                              'avg_star_rating': 3.3,
                              'category': 'Beer Bar, Burgers, Pubs',
                              'name': 'East River',
                              'tot_ratings': 4},
                             {'address': 'Piazza Tre Torri 1L',
                              'avg_star_rating': 0.0,
                              'category': 'Bars, Italian, Cafes',
                              'name': 'Bistrot City Life',
                              'tot_ratings': 0}],
               'pharmacies': [],
               'playgrounds': [{'area_mq': 1266.72318686715,
                                'municipality': 8,
                                'name': 'via Demetrio Stratos',
                                'park_id': 140705,
                                'perimeter_m': 177.3132451489803},
                               {'area_mq': 752.8324000176437,
                                'municipality': 8,
                                'name': 'piazza Giulio Cesare',
                                'park_id': 28266,
                                'perimeter_m': 110.55382725754959}],
```

```
                'restaurants': [{'address': 'Viale Cassiodoro 5',
                                 'avg_star_rating': 4.6,
                                 'category': 'Dim Sum, Asian Fusion,
                                    Japanese',
                                 'name': 'Mi Cucina di Confine',
                                 'price': '€€',
                                 'tot_ratings': 8},
                                {'address': 'Piazza Tre Torri',
                                 'avg_star_rating': 5.0,
                                 'category': 'Wine Bars, Venues & Event
                                    Spaces, '
                                             'Cafes',
                                 'name': 'Peck City Life',
                                 'price': 'N/A',
                                 'tot_ratings': 1},
                                {'address': 'Piazza Tre Torri 1L',
                                 'avg_star_rating': 2.0,
                                 'category': 'Mexican',
                                 'name': 'Calavera',
                                 'price': 'N/A',
                                 'tot_ratings': 3},
                                {'address': 'Piazza Tre Torri 1L',
                                 'avg_star_rating': 4.0,
                                 'category': 'Sushi Bars, Brazilian,
                                    Asian '
                                             'Fusion',
                                 'name': 'Bomaki',
                                 'price': 'N/A',
                                 'tot_ratings': 1},
                                {'address': 'Piazza Tre Torri 1L',
                                 'avg_star_rating': 3.3,
                                 'category': 'Beer Bar, Burgers, Pubs',
                                 'name': 'East River',
                                 'price': 'N/A',
                                 'tot_ratings': 4}],
                'schools': [],
                'sportvenues': [],
                'supermarkets': [{'name': 'Carrefour Market'}],
                'transport': [{'lines': '5',
                               'name': 'TRE TORRI',
                               'transport_type': 'Metro'},
                              {'lines': '1',
                               'name': 'AMENDOLA',
                               'transport_type': 'Metro'},
                              {'lines': '151',
                               'name': 'P.za Amendola',
                               'transport_type': 'Bus'},
                              {'lines': '68',
                               'name': 'V.le Berengario, 8 dopo P.za
                                  Amendola',
                               'transport_type': 'Bus'},
```

```
                                {'lines': '1,19',
                                 'name': 'V.le Boezio altezza l.go
                                    Domodossola',
                                 'transport_type': 'Bus'}],
                'universities': []},
  'neighborhood_name': 'Tre Torri'}
```

Most diverse Neighborhoods (in terms of amenities)

With this query we wanted to show which neighborhoods where the most diverse in terms of the amount of different POIs contained.

```python
pipeline = [
    {
        "$addFields": {
            "diversity_score": {
                "$size": {
                    "$filter": {
                        "input": {"$objectToArray": "$locations"},  # Convert 'locations' sub-document to array
                        "as": "amenity",
                        "cond": {"$gt": [{"$size": "$$amenity.v"}, 0]}  # Count non-empty categories
                    }
                }
            }
        }
    },
    {"$sort": {"diversity_score": -1}},  # Sort neighborhoods by diversity score (highest first)
    {"$limit": 5},  # Return only the top 5 neighborhoods
    {"$project": {  # Project the fields to include in the output
        "neighborhood_name": 1,
        "diversity_score": 1
    }}
]

# Execute the query
results = list(collection.aggregate(pipeline))

# Output results
print("\n=== Top 5 Neighborhoods with the Most Diverse Amenities ===")
for i, result in enumerate(results, start=1):
    print(f"{i}. {result['neighborhood_name']} (Diversity Score: {result['diversity_score']})")
```

```
=== Top 5 Neighborhoods with the Most Diverse Amenities ===
1. Guastalla (Diversity Score: 13)
2. Bovisa (Diversity Score: 13)
```

```
3. Buenos Aires - Venezia (Diversity Score: 13)
4. Città Studi (Diversity Score: 12)
5. Bicocca (Diversity Score: 12)
```

Score calculation (Students, Singles/Couples, Families)

In this section we defined a custom scoring function and then used it with weights that simulate three different categories, Students, Singles/Couples and Families.

NaNs removals

Since some of the neighborhoods have NaN values for the avg_price of the homes, the choice was to either exlude them at all from the ranking, or to assign the global average as their average home prices. Since keeping all the neighborhoods for comparison was our main goal, we decided to proceed with the second option

```
neighborhoods_with_nan = collection.find({})
print("Neighborhoods with avg_price = NaN:")
for doc in neighborhoods_with_nan:
    avg_price = doc.get("home_prices", {}).get("avg_price")
    if avg_price is not None and math.isnan(avg_price):  # Check if
        avg_price is NaN
        print(doc["neighborhood_name"])
```

```
Neighborhoods with avg_price = NaN:
Chiaravalle
Quintosole
Ronchetto delle Rane
```

```
# Compute global minimum or average price
price_stats = collection.aggregate([
    {"$match": {"home_prices.avg_price": {"$not": {"$eq": float("NaN")
        }}}},  # Exclude NaN and missing values
    {"$group": {
        "_id": None,
        "avg_avg_price": {"$avg": "$home_prices.avg_price"}
    }}
])
price_stats = next(price_stats, None)

global_avg_price = price_stats["avg_avg_price"]

# Update neighborhoods with NaN avg_price to the global average price
collection.update_many(
    {"neighborhood_name": {"$in": ["Chiaravalle", "Quintosole", "
        Ronchetto delle Rane"]}},
```

```
    {"$set": {"home_prices.avg_price": global_avg_price}}
)
```

```
UpdateResult({'n': 3, 'nModified': 3, 'ok': 1.0, 'updatedExisting':
    True}, acknowledged=True)
```

Function to Compute Score:

The function takes into consideration the number of locations of each type and the average price of homes in each neighborhood. Further development should be focused on taking into consideration also specific attributes of each location, like the quality of restaurants, the square metres of parks and playgrounds, or the number of workstations in coworking spaces. For this project we opted for a three general scores for each neighborhood, since assigning weights to those location-specific attributes would require external knowledge from an expert of the field, or also questionaries from the public.

```
def compute_score(neighborhood_doc, weights, price_weight, collection):
    # Compute a score for a neighborhood, considering distinct POIs for
        certain categories.

    # takes in input
    #    neighborhood_doc: A MongoDB document with neighborhood data.
    #    weights: A dictionary of weights for each POI category.
    #    price_weight: Weight to apply to the normalized avg_price.
    #    collection: The MongoDB collection to query for global min and
        max avg_price.

    # Returns:
    #    The total score for the neighborhood.

    total_score = 0.0

    # Categories requiring distinct filtering
    distinct_categories = {"universities", "sportvenues", "schools"}

    # Retrieve global min and max for each POI category to normalize
       the count
    poi_stats = collection.aggregate([
        {"$project": {
            "poi_counts": {
                "$map": {
                    "input": {"$objectToArray": "$locations"},
                    "as": "poi",
                    "in": {"k": "$$poi.k", "v": {"$size": {"$ifNull": [
                       "$$poi.v", []]}}}
                }
            }
        }},
```

```
        {"$unwind": "$poi_counts"},
        {"$group": {
            "_id": "$poi_counts.k",
            "min_count": {"$min": "$poi_counts.v"},
            "max_count": {"$max": "$poi_counts.v"}
        }}
    ])

    # Convert the results into a dictionary
    global_poi_min_max = {stat["_id"]: {"min": stat["min_count"], "max"
        : stat["max_count"]} for stat in poi_stats}

    # Normalize the POI counts and compute the scores
    for category, weight in weights.items():
        pois = neighborhood_doc.get("locations", {}).get(category, [])

        if category in distinct_categories:
            # For distinct categories, filter unique entries by address
            unique_pois = {poi.get("address") for poi in pois if "
                address" in poi}
            count = len(unique_pois)
        else:
            # Regular count for other categories
            count = len(pois)

        global_min = global_poi_min_max.get(category, {}).get("min", 0)
        global_max = global_poi_min_max.get(category, {}).get("max", 1)
             # Avoid division by zero

        if global_max > global_min:
            normalized_count = (count - global_min) / (global_max -
                global_min)
        else:
            normalized_count = 0.0

        total_score += normalized_count * weight

    # Price influence
    avg_price = neighborhood_doc.get("home_prices", {}).get("avg_price"
        )

    # Retrieve global min and max prices from the database for
        normalization of the avg price
    price_stats = collection.aggregate([
        {"$group": {
            "_id": None,
            "min_avg_price": {"$min": "$home_prices.avg_price"},
            "max_avg_price": {"$max": "$home_prices.avg_price"}
        }}
    ])
    price_stats = next(price_stats, None)
```

```
    min_avg_price = price_stats["min_avg_price"]
    max_avg_price = price_stats["max_avg_price"]

    normalized_price = (avg_price - min_avg_price) / (max_avg_price -
        min_avg_price)
    total_score -= normalized_price * price_weight

    return total_score
```

Weights

```
# Example weighting dictionaries (tweak as you wish)

students_weights = {
    "restaurants": 2.0,
    "museums": 5.0,
    "nightlife": 8.0,
    "dogparks": 1.0,
    "pharmacies": 6.0,
    "playgrounds": 6.0,
    "sportvenues": 8.0,
    "schools": 1.0,
    "universities": 10.0,
    "coworking": 7.0,
    "libraries": 9.0,
    "supermarkets": 9.0,
    "transport": 10.0
}
# price weight
price_weight_students = 10.0


single_couples_weights = {
    "restaurants": 7.0,
    "museums": 5.0,
    "nightlife": 8.0,
    "dogparks": 5.0,
    "pharmacies": 6.0,
    "playgrounds": 1.0,
    "sportvenues": 7.0,
    "schools": 1.0,
    "universities": 1.0,
    "coworking": 8.0,
    "libraries": 5.0,
    "supermarkets": 10.0,
    "transport": 10.0
}
# price weight
price_weight_single_couples = 6.0
```

```

families_weights = {
    "restaurants": 1.0,
    "museums": 6.0,
    "nightlife": 1.0,
    "dogparks": 10.0,
    "pharmacies": 7.0,
    "playgrounds": 10.0,
    "sportvenues": 3.0,
    "schools": 10.0,
    "universities": 1.0,
    "coworking": 1.0,
    "libraries": 8.0,
    "supermarkets": 8.0,
    "transport": 4.0
}
# price weight
price_weight_families = 7.5
```

Scores

```
# Read all neighborhoods
all_neighborhoods = list(collection.find({}))

# --- Ranking for Students ---
print("=== Ranking for Students ===")
students_scores = []
for nb in all_neighborhoods:
    score = compute_score(nb, students_weights, price_weight=
        price_weight_students, collection=collection)
    students_scores.append({
        "neighborhood_name": nb["neighborhood_name"],
        "score": score
    })

# sort by score descending
students_scores.sort(key=lambda x: x["score"], reverse=True)

# transform scores into percentages to be able to interpret better the
    scoring
if students_scores:
    max_score_students = students_scores[0]["score"]
    for item in students_scores: # sort of scaling
        item["percentage"] = (item["score"] / max_score_students) * 100
            if max_score_students > 0 else 0

# showing the top 5 neighborhoods
for rank, item in enumerate(students_scores[:5], start=1):
```

```
    print(f"{rank}. {item['neighborhood_name']} => {item['percentage']:.2f}%")


# --- Ranking for Singles/Couples ---
print("\n=== Ranking for Singles/Couples ===")
single_couples_scores = []
for nb in all_neighborhoods:
    score = compute_score(nb, single_couples_weights, price_weight=price_weight_single_couples, collection=collection)
    single_couples_scores.append({
        "neighborhood_name": nb["neighborhood_name"],
        "score": score
    })

single_couples_scores.sort(key=lambda x: x["score"], reverse=True)

# transform scores into percentages to be able to interpret better the scoring
if single_couples_scores:
    max_score_single_couples = single_couples_scores[0]["score"]
    for item in single_couples_scores: # sort of scaling
        item["percentage"] = (item["score"] / max_score_single_couples) * 100 if max_score_single_couples > 0 else 0

# showing the top 5 neighborhoods
for rank, item in enumerate(single_couples_scores[:5], start=1):
    print(f"{rank}. {item['neighborhood_name']} => {item['percentage']:.2f}%")


# --- Ranking for Families ---
print("\n=== Ranking for Families ===")
families_scores = []
for nb in all_neighborhoods:
    score = compute_score(nb, families_weights, price_weight=price_weight_families, collection=collection)
    families_scores.append({
        "neighborhood_name": nb["neighborhood_name"],
        "score": score
    })

families_scores.sort(key=lambda x: x["score"], reverse=True)

# transform scores into percentages to be able to interpret better the scoring
if families_scores:
    max_score_families = families_scores[0]["score"]
    for item in families_scores: # sort of scaling
        item["percentage"] = (item["score"] / max_score_families) * 100 if max_score_families > 0 else 0
```

```
68
69  # showing the top 5 neighborhoods
70  for rank, item in enumerate(families_scores[:5], start=1):
71      print(f"{rank}. {item['neighborhood_name']} => {item['percentage
        ']:.2f}%")
```

```
1   === Ranking for Students ===
2   1. Buenos Aires - Venezia => 100.00%
3   2. Città Studi => 69.57%
4   3. Niguarda - Cà Granda => 65.02%
5   4. Duomo => 64.33%
6   5. Villapizzone => 63.95%
7
8   === Ranking for Singles/Couples ===
9   1. Buenos Aires - Venezia => 100.00%
10  2. Duomo => 72.00%
11  3. Città Studi => 57.49%
12  4. Niguarda - Cà Granda => 54.08%
13  5. Villapizzone => 52.21%
14
15  === Ranking for Families ===
16  1. Niguarda - Cà Granda => 100.00%
17  2. Buenos Aires - Venezia => 98.91%
18  3. Villapizzone => 81.92%
19  4. Stadera => 81.15%
20  5. Gallaratese => 72.33%
```

Cloropleth map (for Students)

```
1   # Fetch all neighborhood documents and calculate scores using the
      previously choosen students scores
2   neighborhood_data = [
3       {
4           "neighborhood_name": doc["neighborhood_name"],
5           "geometry": shape(doc["geometry"]),
6           "score": compute_score(doc, students_weights, price_weight=
              price_weight_students, collection=collection)
7       }
8       for doc in collection.find({})
9   ]
10
11  # find the maximum score
12  max_score = max([item["score"] for item in neighborhood_data], default
      =1)  #to avoid division by zero
13
14  # normalizing the scores to percentages
15  for item in neighborhood_data:
16      item["percentage"] = (item["score"] / max_score) * 100 if max_score
            > 0 else 0
```

```
17
18 # conversion to GeoDataFrame and ensuring CRS is set to EPSG:4326
19 gdf = gpd.GeoDataFrame(neighborhood_data, crs="EPSG:4326")
20
21 # generate an empty folium map
22 map1 = folium.Map(location=[45.4642, 9.1900], zoom_start=12)
23
24 # first add a choropleth layer
25 folium.Choropleth(
26     geo_data=gdf.to_json(),
27     name="choropleth",
28     data=gdf,
29     columns=["neighborhood_name", "percentage"],
30     key_on="feature.properties.neighborhood_name",
31     fill_color="YlOrRd",
32     fill_opacity=0.7,
33     line_opacity=0.2,
34     legend_name="Students Ranking Score (%)"
35 ).add_to(map1)
36
37 # add the tooltip layer with transparent polygons
38 folium.GeoJson(
39     gdf,
40     name="Neighborhoods",
41     tooltip=folium.GeoJsonTooltip(
42         fields=["neighborhood_name", "percentage"],
43         aliases=["Neighborhood:", "Score (%):"],
44         localize=True
45     ),
46     style_function=lambda x: {"fillColor": "transparent", "color": "
        transparent", "weight": 0}
47 ).add_to(map1)
48
49 map1
```

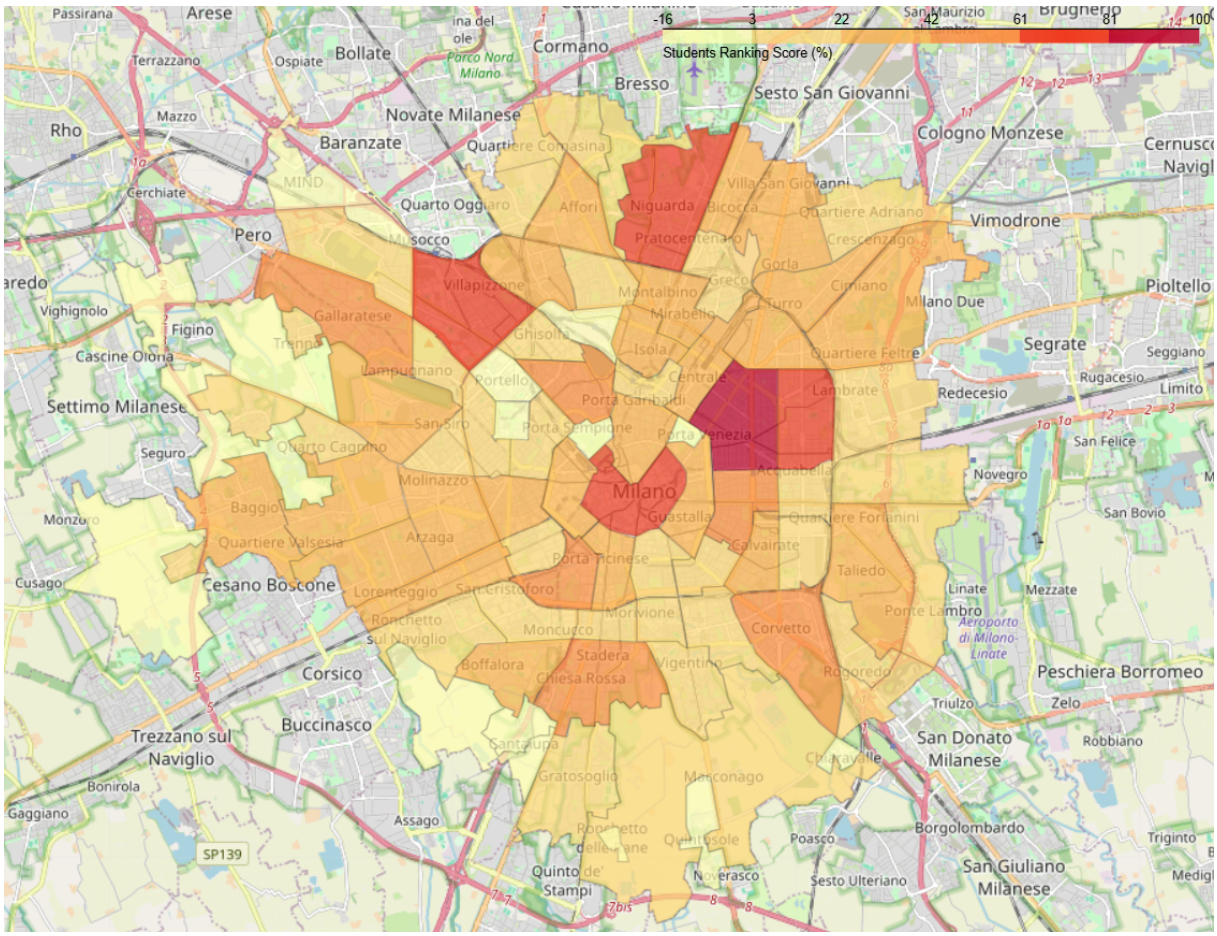

Figure 1: Custom Map

Suitable Neighborhoods

Here the user gets asked some questions in order to present the most suitable neighborhood depending on their needs. Further development could be focus on asking even more in depth questions, for example if it is important for restaurants to have excellent reviews, or if some combinations are preferred compared to others (for example a user could prefer to have lots of restaurants compared to the presence of a library, but ideally would want both to be in their neighborhood)

```
# Interactive user inputs
print("\nPlease specify your preferences:")

faculty = input("Enter the faculty you're looking for (e.g., Economia):
    ").strip().upper()  # Convert to uppercase
parks_required = input("Do you need a dog park? (yes/no): ").strip().
   lower() == "yes"
```

```
6  library_required = input("Do you need a library? (yes/no): ").strip().
       lower() == "yes"
7
8  restaurant_category = input("Enter the type of restaurant you prefer (e
       .g., Italian): ").strip()
9  min_restaurants = int(input("Enter the minimum number of restaurants
       you want: ").strip() or 0)
10
11 coworking_required = input("Do you need a coworking space? (yes/no): ")
       .strip().lower() == "yes"
12 sport_venue_required = input("Do you need a sport venue? (yes/no): ").
       strip().lower() == "yes"
13 sport_venue_category = input("Enter the sport venue category (e.g.,
       Piscina, Atletica) [optional]: ").strip().upper() if
       sport_venue_required else None
14 supermarket_required = input("Do you need a supermarket? (yes/no): ").
       strip().lower() == "yes"
15 museum_required = input("Do you need a museum? (yes/no): ").strip().
       lower() == "yes"
16 pharmacy_required = input("Do you need a pharmacy? (yes/no): ").strip()
       .lower() == "yes"
17 playground_required = input("Do you need a playground? (yes/no): ").
       strip().lower() == "yes"
18
19 transport_required = input("Do you need public transport? (yes/no): ").
       strip().lower() == "yes"
20 metro_required = train_required = bus_required = False
21 if transport_required:
22     metro_required = input("Do you need metro service? (yes/no): ").
           strip().lower() == "yes"
23     train_required = input("Do you need train service? (yes/no): ").
           strip().lower() == "yes"
24     bus_required = input("Do you need bus service? (yes/no): ").strip()
           .lower() == "yes"
25
26 budget = float(input("Enter your budget for home prices (price in euros
       per square meter): ").strip() or 0)
27
28 # Build the query dynamically based on inputs
29 query = {
30     "$addFields": {
31         "match_score": {
32             "$add": [
33                 # Check for faculty match in university
34                 {"$cond": [
35                     {"$in": [faculty, "$locations.universities.faculty"
                           ]}, 1, 0
36                 ]} if faculty else 0,
37                 # Check for restaurant category match
38                 {"$cond": [
39                     {"$in": [restaurant_category, "$locations.
```

```
                        restaurants.category"]}, 1, 0
40                  ]} if restaurant_category else 0,
41                  # Check for minimum number of restaurants
42                  {"$cond": [
43                      {"$gte": [{"$size": "$locations.restaurants"},
                            min_restaurants]}, 1, 0
44                  ]} if min_restaurants > 0 else 0,
45                  # Check for parks presence
46                  {"$cond": [
47                      {"$gt": [{"$size": "$locations.dogparks"}, 0]}, 1,
                            0
48                  ]} if parks_required else 0,
49                  # Check for libraries presence
50                  {"$cond": [
51                      {"$gt": [{"$size": "$locations.libraries"}, 0]}, 1,
                             0
52                  ]} if library_required else 0,
53                  # Check for coworking presence
54                  {"$cond": [
55                      {"$gt": [{"$size": "$locations.coworking"}, 0]}, 1,
                             0
56                  ]} if coworking_required else 0,
57                  # Check for sport venue presence
58                  {"$cond": [
59                      {"$gt": [{"$size": "$locations.sportvenues"}, 0]},
                            1, 0
60                  ]} if sport_venue_required else 0,
61                  # Check for specific sport venue category match
62                  {"$cond": [
63                      {"$in": [sport_venue_category, "$locations.
                            sportvenues.category"]}, 1, 0
64                  ]} if sport_venue_category else 0,
65                  # Check for supermarket presence
66                  {"$cond": [
67                      {"$gt": [{"$size": "$locations.supermarkets"}, 0]},
                             1, 0
68                  ]} if supermarket_required else 0,
69                  # Check for museum presence
70                  {"$cond": [
71                      {"$gt": [{"$size": "$locations.museums"}, 0]}, 1, 0
72                  ]} if museum_required else 0,
73                  # Check for pharmacy presence
74                  {"$cond": [
75                      {"$gt": [{"$size": "$locations.pharmacies"}, 0]},
                            1, 0
76                  ]} if pharmacy_required else 0,
77                  # Check for playground presence
78                  {"$cond": [
79                      {"$gt": [{"$size": "$locations.playgrounds"}, 0]},
                            1, 0
80                  ]} if playground_required else 0,
```

```
                # Check for public transport
                {"$cond": [
                    {"$or": [
                        {"$in": ["Metro", "$locations.transport.
                            transport_type"]} if metro_required else
                            False,
                        {"$in": ["Treno", "$locations.transport.
                            transport_type"]} if train_required else
                            False,
                        {"$in": ["Bus", "$locations.transport.
                            transport_type"]} if bus_required else False
                    ]}, 1, 0
                ]} if transport_required else 0,
                # Check for budget in home prices
                {"$cond": [
                    {"$lte": ["$home_prices.avg_price", budget]}, 1, 0
                ]} if budget > 0 else 0
            ]
        }
    }
}

# Run the aggregation pipeline
pipeline = [
    {
        "$match": {
            "home_prices.avg_price": {"$lte": 2 * budget}  # Keep
                neighborhoods within twice the budget
        }
    },
    query,  # Add the match_score field
    {"$sort": {
            "match_score": -1,  # Higher match scores first
            "home_prices.avg_price": 1  # Lower avg prices first
        }
    },
    {"$limit": 3},  # Limit to the top 3 neighborhoods
    {"$project": {  # Project only relevant fields for output
        "neighborhood_name": 1,
        "match_score": 1,
        "locations": 1,
        "home_prices.avg_price": 1
    }}
]

results = list(collection.aggregate(pipeline))

# Output the top neighborhoods
print("\n=== Top 3 Suitable Neighborhoods ===")
if results:
    for i, neighborhood in enumerate(results, start=1):
```

```
126             print(f"\n{i}. {neighborhood['neighborhood_name']} (Score: {
                    neighborhood['match_score']})")
127             print(f"   - Average Price: €{neighborhood.get('home_prices',
                    {}).get('avg_price', 'N/A')}")
128 
129             fulfilled = []
130             not_fulfilled = []
131 
132             # Faculty check
133             if faculty:
134                 if any(faculty in uni.get("faculty", []) for uni in
                        neighborhood["locations"].get("universities", [])):
135                     fulfilled.append("Faculty")
136                 else:
137                     not_fulfilled.append("Faculty")
138 
139             # Restaurant category check
140             if restaurant_category:
141                 if any(restaurant_category in rest.get("category", []) for
                        rest in neighborhood["locations"].get("restaurants", [])
                        ):
142                     fulfilled.append("Restaurant Category")
143                 else:
144                     not_fulfilled.append("Restaurant Category")
145 
146             # Minimum number of restaurants
147             if min_restaurants > 0:
148                 if len(neighborhood["locations"].get("restaurants", [])) >=
                        min_restaurants:
149                     fulfilled.append("Minimum Number of Restaurants")
150                 else:
151                     not_fulfilled.append("Minimum Number of Restaurants")
152 
153             # Parks check
154             if parks_required:
155                 if len(neighborhood["locations"].get("dogparks", [])) > 0:
156                     fulfilled.append("Dog Park")
157                 else:
158                     not_fulfilled.append("Dog Park")
159 
160             # Library check
161             if library_required:
162                 if len(neighborhood["locations"].get("libraries", [])) > 0:
163                     fulfilled.append("Library")
164                 else:
165                     not_fulfilled.append("Library")
166 
167             # Coworking check
168             if coworking_required:
169                 if len(neighborhood["locations"].get("coworking", [])) > 0:
170                     fulfilled.append("Coworking Space")
```

```
171                 else:
172                     not_fulfilled.append("Coworking Space")
173 
174             # Sport venue check
175             if sport_venue_required:
176                 if len(neighborhood["locations"].get("sportvenues", [])) >
                        0:
177                     fulfilled.append("Sport Venue")
178                 else:
179                     not_fulfilled.append("Sport Venue")
180 
181             # Sport venue category check
182             if sport_venue_category:
183                 if any(sport_venue_category in venue.get("category", [])
                        for venue in neighborhood["locations"].get("sportvenues"
                        , [])):
184                     fulfilled.append("Specific Sport Venue Category")
185                 else:
186                     not_fulfilled.append("Specific Sport Venue Category")
187 
188             # Supermarket check
189             if supermarket_required:
190                 if len(neighborhood["locations"].get("supermarkets", [])) >
                         0:
191                     fulfilled.append("Supermarket")
192                 else:
193                     not_fulfilled.append("Supermarket")
194 
195             # Museum check
196             if museum_required:
197                 if len(neighborhood["locations"].get("museums", [])) > 0:
198                     fulfilled.append("Museum")
199                 else:
200                     not_fulfilled.append("Museum")
201 
202             # Pharmacy check
203             if pharmacy_required:
204                 if len(neighborhood["locations"].get("pharmacies", [])) >
                        0:
205                     fulfilled.append("Pharmacy")
206                 else:
207                     not_fulfilled.append("Pharmacy")
208 
209             # Playground check
210             if playground_required:
211                 if len(neighborhood["locations"].get("playgrounds", [])) >
                        0:
212                     fulfilled.append("Playground")
213                 else:
214                     not_fulfilled.append("Playground")
215 
```

```
216         # Transport check
217         if transport_required:
218             transport_fulfilled = []
219             transport_list = neighborhood["locations"].get("transport",
                    [])
220 
221             # Iterate through transport_list to find the required
                    transport types
222             if metro_required and any("Metro" in transport.get("
                    transport_type", []) for transport in transport_list):
223                 transport_fulfilled.append("Metro")
224             if train_required and any("Treno" in transport.get("
                    transport_type", []) for transport in transport_list):
225                 transport_fulfilled.append("Train")
226             if bus_required and any("Bus" in transport.get("
                    transport_type", []) for transport in transport_list):
227                 transport_fulfilled.append("Bus")
228 
229             # If any transport types are fulfilled, add them to
                    fulfilled; otherwise, add to not_fulfilled
230             if transport_fulfilled:
231                 fulfilled.append(f"Transport ({', '.join(
                        transport_fulfilled)})")
232             else:
233                 not_fulfilled.append("Transport")
234 
235         # Budget check
236         if budget > 0:
237             avg_price = neighborhood.get("home_prices", {}).get("
                    avg_price", float("inf"))
238             if avg_price <= budget:
239                 fulfilled.append("Budget")
240             else:
241                 not_fulfilled.append("Budget")
242 
243         # Print fulfilled and not fulfilled
244         print("  - Fulfilled:", ", ".join(fulfilled) if fulfilled else
                "None")
245         print("  - Not Fulfilled:", ", ".join(not_fulfilled) if
                not_fulfilled else "None")
246 else:
247     print("No neighborhoods match your criteria.")
```

```
1 Please specify your preferences:
2 
3 
4 Enter the faculty you're looking for (e.g., Economia):  Fisica
5 Do you need a dog park? (yes/no):  no
6 Do you need a library? (yes/no):  yes
7 Enter the type of restaurant you prefer (e.g., Italian):  Seafood
8 Enter the minimum number of restaurants you want:  6
```

```
Do you need a coworking space? (yes/no):  yes
Do you need a sport venue? (yes/no):  yes
Enter the sport venue category (e.g., Piscina, Atletica) [optional]:
    Tennis
Do you need a supermarket? (yes/no):  yes
Do you need a museum? (yes/no):  no
Do you need a pharmacy? (yes/no):  yes
Do you need a playground? (yes/no):  no
Do you need public transport? (yes/no):  yes
Do you need metro service? (yes/no):  yes
Do you need train service? (yes/no):  no
Do you need bus service? (yes/no):  yes
Enter your budget for home prices (price in euros per square meter):
    3500



=== Top 3 Suitable Neighborhoods ===

1. Città Studi (Score: 10)
  - Average Price: €3947.5
  - Fulfilled: Faculty, Restaurant Category, Minimum Number of
      Restaurants, Library, Coworking Space, Sport Venue, Specific Sport
      Venue Category, Supermarket, Pharmacy, Transport (Metro, Bus)
  - Not Fulfilled: Budget

2. Stadera (Score: 9)
  - Average Price: €2990.625
  - Fulfilled: Minimum Number of Restaurants, Library, Coworking Space,
      Sport Venue, Specific Sport Venue Category, Supermarket, Pharmacy
     , Transport (Metro, Bus), Budget
  - Not Fulfilled: Faculty, Restaurant Category

3. Quarto Cagnino (Score: 9)
  - Average Price: €3086.4583333333335
  - Fulfilled: Restaurant Category, Minimum Number of Restaurants,
      Coworking Space, Sport Venue, Specific Sport Venue Category,
      Supermarket, Pharmacy, Transport (Bus), Budget
  - Not Fulfilled: Faculty, Library
```