
MongoDB Integration and Queries

Analysis and Ranking of Milan's Neighborhoods

Edoardo Olivieri, Federica Romano, Francesca Verna

27/01/2025



Contents

Integration	2
Connection to MongoDB and loading the files	2
Initializing the dictionary	2
Function to append rows	3
Appending the GeoDataFrames	4
Inserting the data into MongoDB	8
Queries	8
Neighborhood (Example)	8
Most diverse Neighborhoods (in terms of amenities)	11
Score calculation (Students, Singles/Couples, Families)	12
Function to Compute Score:	12
Weights	14
Scores	15
Cloropleth map (for Students)	17
Suitable Neighborhoods	19

```
1 from pymongo import MongoClient
2 import geopandas as gpd
3 from shapely.geometry import shape, mapping
4 import folium
5 import math
6 from pprint import pprint
```

Integration

Connection to MongoDB and loading the files

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

```
1 # Reading the files
2 gdf_combined = gpd.read_file("C:/Users/edoar/combined_quartieri.geojson")
3 PolyHomePrices = gpd.read_file("C:/Users/edoar/PolyHomePrices.geojson")
4 PolyRestaurants = gpd.read_file("C:/Users/edoar/PolyRestaurants.geojson")
5 PolyMuseums = gpd.read_file("C:/Users/edoar/PolyMuseums.geojson")
6 PolyNightlife = gpd.read_file("C:/Users/edoar/PolyNightlife.geojson")
7 PolyDogParks = gpd.read_file("C:/Users/edoar/PolyDogParks.geojson")
8 PolyPharmacy = gpd.read_file("C:/Users/edoar/PolyPharmacy.geojson")
9 PolyPlaygrounds = gpd.read_file("C:/Users/edoar/PolyPlaygrounds.geojson")
10 PolySportVenues = gpd.read_file("C:/Users/edoar/PolySportVenues.geojson")
11 PolySchools = gpd.read_file("C:/Users/edoar/PolySchools.geojson")
12 PolyUniversity = gpd.read_file("C:/Users/edoar/PolyUniversity.geojson")
13 PolyCoworking = gpd.read_file("C:/Users/edoar/PolyCoworking.geojson")
14 PolyLibraries = gpd.read_file("C:/Users/edoar/PolyLibraries.geojson")
15 PolySupermarkets = gpd.read_file("C:/Users/edoar/PolySupermarkets.
    geojson")
16 PolyTransport = gpd.read_file("C:/Users/edoar/PolyTransport.geojson")
```

Initializing the dictionary

```
1 # Create a dictionary to hold all neighborhood docs
2 neighborhood_docs = {}
3
4 # populating the base neighborhood documents
```

```

5 for idx, row in gdf_combined.iterrows():
6     nb_name = row["Neighborhood"]
7     # using shapely's "mapping function" from shapely.geometry to
        convert geometries
8     # to a geojson-like dictionary to store them in MongoDB
9     geo_json = mapping(row["geometry"])
10
11     neighborhood_docs[nb_name] = {
12         "_id": nb_name,
13         "neighborhood_name": nb_name,
14         "geometry": geo_json,
15         "locations": {
16             "restaurants": [],
17             "museums": [],
18             "nightlife": [],
19             "dogparks": [],
20             "pharmacies": [],
21             "playgrounds": [],
22             "sportvenues": [],
23             "schools": [],
24             "universities": [],
25             "coworking": [],
26             "libraries": [],
27             "supermarkets": [],
28             "transport": []
29         },
30         "home_prices": {
31             "min_price": None, # placeholder for the min price
32             "max_price": None, # placeholder for the max price
33             "avg_price": None # placeholder for the avg price
34         }
35     }

```

Function to append rows

```

1 # Function to append rows from a POI GeoDataFrame to the
        neighborhood_docs
2 def append_to_neighborhoods(field, gdf, poi_key, field_mappings=None):
3     """
4     field: the sub-document where the list will be appended to.
5     gdf: A GeoDataFrame with columns ["Neighborhood", ...data columns
        ...].
6     poi_key: e.g. "pharmacies", "restaurants", etc.
7     field_mappings: dict of { "source_column": "destination_field_name",
        ..., ... }
8                                     used to pick and rename columns from the gdf row.
9     """
10    if field_mappings is None:

```

```
11     # if not supplied, just store all columns except geometry and
    Neighborhood
12     field_mappings = {
13         col: col
14         for col in gdf.columns
15         if col not in ("Neighborhood", "geometry")
16     }
17
18     # group by Neighborhood to handle rows for each neighborhood
19     grouped = gdf.groupby("Neighborhood")
20
21     # for each neighborhood:
22     for nb_name, group_df in grouped:
23         # if neighborhood is not in the dict, skip it
24         if nb_name not in neighborhood_docs:
25             continue
26
27         # convert each row to a dictionary with the needed fields
28         for _, row in group_df.iterrows():
29             poi_data = {}
30             for src_col, dest_col in field_mappings.items():
31                 if src_col in row:
32                     poi_data[dest_col] = row[src_col]
33
34             # append to the correct list inside the correct field
35             neighborhood_docs[nb_name][field][poi_key].append(poi_data)
```

Appending the GeoDataFrames

```
1  # Append each of the 13 POI DataFrames to the base neighborhood docs
2
3  # Restaurants
4  append_to_neighborhoods(
5      field="locations",
6      gdf=PolyRestaurants,
7      poi_key="restaurants",
8      field_mappings={
9          "Business Name": "name",
10         "Business Address": "address",
11         "Categories": "category",
12         "Average Star Rating": "avg_star_rating",
13         "Review Count": "tot_ratings",
14         "Price": "price"
15     }
16 )
17
18 # Museums
19 append_to_neighborhoods(
20     field="locations",
```

```
21     gdf=PolyMuseums,
22     poi_key="museums",
23     field_mappings={
24         "Museum Name": "name",
25         "Museum Address": "address",
26         "Categories": "category",
27         "Average Star Rating": "avg_star_rating",
28         "Review Count": "tot_ratings"
29     }
30 )
31
32 # Nightlife
33 append_to_neighborhoods(
34     field="locations",
35     gdf=PolyNightlife,
36     poi_key="nightlife",
37     field_mappings={
38         "Venue Name": "name",
39         "Venue Address": "address",
40         "Categories": "category",
41         "Average Star Rating": "avg_star_rating",
42         "Review Count": "tot_ratings"
43     }
44 )
45
46 # Dog Parks
47 append_to_neighborhoods(
48     field="locations",
49     gdf=PolyDogParks,
50     poi_key="dogparks",
51     field_mappings={
52         "località": "name",
53         "area_mq": "area_mq",
54         "perim_m": "perimeter_m",
55         "obj_id": "park_id",
56         "municipio": "municipality"
57     }
58 )
59
60 # Pharmacies
61 append_to_neighborhoods(
62     field="locations",
63     gdf=PolyPharmacy,
64     poi_key="pharmacies",
65     field_mappings={
66         "DESCRIZIONE_FARMACIA": "name",
67         "INDIRIZZO": "address",
68         "CODICE_FARMACIA": "pharmacy_id",
69         "MUNICIPIO": "municipality"
70     }
71 )
```

```
72
73 # Playgrounds
74 append_to_neighborhoods(
75     field="locations",
76     gdf=PolyPlaygrounds,
77     poi_key="playgrounds",
78     field_mappings={
79         "località": "name",
80         "area_mq": "area_mq",
81         "perim_m": "perimeter_m",
82         "obj_id": "park_id",
83         "municipio": "municipality"
84     }
85 )
86
87 # Sport Venues
88 append_to_neighborhoods(
89     field="locations",
90     gdf=PolySportVenues,
91     poi_key="sportvenues",
92     field_mappings={
93         "Nome": "name",
94         "Indirizzo": "address",
95         "info": "category",
96     }
97 )
98
99 # Schools
100 append_to_neighborhoods(
101     field="locations",
102     gdf=PolySchools,
103     poi_key="schools",
104     field_mappings={
105         "DENOMINAZIONESCUELA": "name",
106         "INDIRIZZOSCUELA": "address",
107         "DESCRIZIONECARATTERISTICASCUELA": "school_type",
108         "DESCRIZIONETIPOLOGIAGRADOISTRUZIONEESCUELA": "educational_lvl",
109         "MUNICIPIO": "municipality"
110     }
111 )
112
113 # Universities
114 append_to_neighborhoods(
115     field="locations",
116     gdf=PolyUniversity,
117     poi_key="universities",
118     field_mappings={
119         "DENOMINAZ": "name",
120         "INDIRIZZO": "address",
121         "FACOLTA": "faculty",
122         "PROPRIETA": "ownership_type",
```

```
123         "MUNICIPIO": "municipality"
124     }
125 )
126
127 # Coworking
128 append_to_neighborhoods(
129     field="locations",
130     gdf=PolyCoworking,
131     poi_key="coworking",
132     field_mappings={
133         "SPAZIO": "name",
134         "Sede": "address",
135         "Orario di apertura": "opening_hrs",
136         "Numero postazioni": "tot_desks",
137         "MUNICIPIO": "municipality"
138     }
139 )
140
141 # Libraries
142 append_to_neighborhoods(
143     field="locations",
144     gdf=PolyLibraries,
145     poi_key="libraries",
146     field_mappings={
147         "Biblioteche - Sede": "name",
148         "Indirizzo": "address",
149         "MUNICIPIO": "municipality"
150     }
151 )
152
153 # Supermarkets
154 append_to_neighborhoods(
155     field="locations",
156     gdf=PolySupermarkets,
157     poi_key="supermarkets",
158     field_mappings={
159         "name": "name"
160     }
161 )
162
163 # Transport
164 append_to_neighborhoods(
165     field="locations",
166     gdf=PolyTransport,
167     poi_key="transport",
168     field_mappings={
169         "Nome": "name",
170         "Linee": "lines",
171         "Mezzo": "transport_type"
172     }
173 )
```



```
174
175 # Home Prices
176 for idx, row in PolyHomePrices.iterrows():
177     nb_name = row["Neighborhood"]
178     if nb_name in neighborhood_docs:
179         # updateing the home_prices container with actual values
180         neighborhood_docs[nb_name]["home_prices"]["min_price"] = row["
            Compr_min"]
181         neighborhood_docs[nb_name]["home_prices"]["max_price"] = row["
            Compr_max"]
182         neighborhood_docs[nb_name]["home_prices"]["avg_price"] = row["
            Compr_mean"]
```

Inserting the data into MongoDB

```
1 # Insert the data into MongoDB
2 # (neighborhood_docs is a dictionary with neighborhood_name as keys and
   the final documents as values)
3 documents_to_insert = list(neighborhood_docs.values()) # conversion to
   list of dicts
4
5 collection.insert_many(documents_to_insert)
6
7 print("Data inserted into MongoDB")
```

```
1 Data inserted into MongoDB
```

Queries

Neighborhood (Example)

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

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

```
1 {'_id': 'Tre Torri',
2  'geometry': {'coordinates': [[9.1598895, 45.4742524],
3                                [9.1599665, 45.4752257],
4                                [9.159981, 45.475719],
5                                [9.1600586, 45.4772662],
6                                [9.1600766, 45.4774377],
7                                [9.1600869, 45.4779766],
8                                [9.1601359, 45.4786687],
```

```

9      [9.1602194, 45.4800522],
10     [9.1602396, 45.4801317],
11     [9.1602753, 45.4803015],
12     [9.1600595, 45.4803093],
13     [9.1586847, 45.4803591],
14     [9.1559822, 45.4803917],
15     [9.1516378, 45.4804495],
16     [9.1516812, 45.4802959],
17     [9.1516687, 45.4800506],
18     [9.1515654, 45.4780585],
19     [9.1513698, 45.4744323],
20     [9.1511429, 45.4743509],
21     [9.1505715, 45.4739819],
22     [9.1515806, 45.4731998],
23     [9.1516232, 45.4736605],
24     [9.1527203, 45.4736241],
25     [9.1537328, 45.4736061],
26     [9.1555551, 45.4735608],
27     [9.1567836, 45.4735338],
28     [9.1573631, 45.4735149],
29     [9.158371, 45.4734854],
30     [9.1598245, 45.4734469],
31     [9.1598895, 45.4742524]]],
32     'type': 'Polygon'},
33     'home_prices': {'avg_price': 7687.5,
34                     'max_price': 12600.0,
35                     'min_price': 3500.0},
36     'locations': {'coworking': [],
37                   'dogparks': [{'area_sq': 1007.6380499947832,
38                                'municipality': 8,
39                                'name': 'piazza Giulio Cesare',
40                                'park_id': 25448,
41                                'perimeter_m': 126.55090634925524}],
42                   'libraries': [],
43                   'museums': [],
44                   'nightlife': [{'address': 'Piazzale Arduino 1',
45                                  'avg_star_rating': 4.0,
46                                  'category': 'Cocktail Bars',
47                                  'name': 'GUD',
48                                  'tot_ratings': 4},
49                                  {'address': 'Piazza Tre Torri',
50                                   'avg_star_rating': 5.0,
51                                   'category': 'Wine Bars, Venues & Event
52                                           Spaces, '
53                                           'Cafes',
54                                   'name': 'Peck City Life',
55                                   'tot_ratings': 1},
56                                  {'address': 'Piazza Tre Torri 1L',
57                                   'avg_star_rating': 3.3,
58                                   'category': 'Beer Bar, Burgers, Pubs',
59                                   'name': 'East River',

```

```

59         'tot_ratings': 4},
60     {'address': 'Piazza Tre Torri 1L',
61      'avg_star_rating': 0.0,
62      'category': 'Bars, Italian, Cafes',
63      'name': 'Bistrot City Life',
64      'tot_ratings': 0}],
65     'pharmacies': [],
66     'playgrounds': [{ 'area_mq': 1266.72318686715,
67                       'municipality': 8,
68                       'name': 'via Demetrio Stratos',
69                       'park_id': 140705,
70                       'perimeter_m': 177.3132451489803},
71                     { 'area_mq': 752.8324000176437,
72                       'municipality': 8,
73                       'name': 'piazza Giulio Cesare',
74                       'park_id': 28266,
75                       'perimeter_m': 110.55382725754959}],
76     'restaurants': [{ 'address': 'Viale Cassiodoro 5',
77                       'avg_star_rating': 4.6,
78                       'category': 'Dim Sum, Asian Fusion,
79                               Japanese',
80                       'name': 'Mi Cucina di Confine',
81                       'price': '€€',
82                       'tot_ratings': 8},
83                     { 'address': 'Piazza Tre Torri',
84                       'avg_star_rating': 5.0,
85                       'category': 'Wine Bars, Venues & Event
86                               Spaces, '
87                               Cafes',
88                       'name': 'Peck City Life',
89                       'price': 'N/A',
90                       'tot_ratings': 1},
91                     { 'address': 'Piazza Tre Torri 1L',
92                       'avg_star_rating': 2.0,
93                       'category': 'Mexican',
94                       'name': 'Calavera',
95                       'price': 'N/A',
96                       'tot_ratings': 3},
97                     { 'address': 'Piazza Tre Torri 1L',
98                       'avg_star_rating': 4.0,
99                       'category': 'Sushi Bars, Brazilian,
100                               Asian '
101                               Fusion',
102                       'name': 'Bomaki',
103                       'price': 'N/A',
104                       'tot_ratings': 1},
105                     { 'address': 'Piazza Tre Torri 1L',
106                       'avg_star_rating': 3.3,
107                       'category': 'Beer Bar, Burgers, Pubs',
108                       'name': 'East River',
109                       'price': 'N/A',

```

```

107         'tot_ratings': 4]],
108     'schools': [],
109     'sportvenues': [],
110     'supermarkets': [{'name': 'Carrefour Market'}],
111     'transport': [{'lines': '5',
112                    'name': 'TRE TORRI',
113                    'transport_type': 'Metro'},
114                   {'lines': '1',
115                    'name': 'AMENDOLA',
116                    'transport_type': 'Metro'},
117                   {'lines': '151',
118                    'name': 'P.za Amendola',
119                    'transport_type': 'Bus'},
120                   {'lines': '68',
121                    'name': 'V.le Berengario, 8 dopo P.za
122                        Amendola',
123                    'transport_type': 'Bus'},
124                   {'lines': '1,19',
125                    'name': 'V.le Boezio altezza l.go
126                        Domodossola',
127                    'transport_type': 'Bus'}],
126     'universities': [],
127     'neighborhood_name': 'Tre Torri'

```

Most diverse Neighborhoods (in terms of amenities)

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

```

1 pipeline = [
2     {
3         "$addField": {
4             "diversity_score": {
5                 "$size": {
6                     "$filter": {
7                         "input": {"$objectToArray": "$locations"}, #
8                             convert locations sub-document to array
9                         "as": "amenity",
10                        "cond": {"$gt": [{"size": "$$amenity.v"}, 0]}
11                        # count non-empty categories
12                    }
13                }
14            }
15        },
16        {"$sort": {"diversity_score": -1}}, # sort neighborhoods by
17        diversity score (highest first)
18    }, {"$limit": 5}, # show only the top 5 neighborhoods
19    {"$project": { # project the fields to include in the output

```

```

18         "neighborhood_name": 1,
19         "diversity_score": 1
20     }}
21 ]
22
23 # query
24 results = list(collection.aggregate(pipeline))
25
26 # results
27 print("\n=== Top 5 Neighborhoods with the Most Diverse Amenities ===")
28 for i, result in enumerate(results, start=1):
29     print(f"{i}. {result['neighborhood_name']} (Diversity Score: {
        result['diversity_score']})")

```

```

1 === Top 5 Neighborhoods with the Most Diverse Amenities ===
2 1. Guastalla (Diversity Score: 13)
3 2. Bovisa (Diversity Score: 13)
4 3. Buenos Aires - Venezia (Diversity Score: 13)
5 4. Città Studi (Diversity Score: 12)
6 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.

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 questionnaires from the public.

```

1 def compute_score(neighborhood_doc, weights, price_weight, collection):
2     # Compute a score for a neighborhood, considering distinct POIs for
      certain categories.
3
4     # takes in input
5     #   neighborhood_doc: A MongoDB document with neighborhood data.
6     #   weights: A dictionary of weights for each POI category.
7     #   price_weight: Weight to apply to the normalized avg_price.
8     #   collection: The MongoDB collection to query for global min and
      max avg_price.

```

```

9
10     # Returns:
11     #     The total score for the neighborhood.
12
13     total_score = 0.0
14
15     # Categories requiring distinct filtering
16     distinct_categories = {"universities", "sportvenues", "schools"}
17
18     # Retrieve global min and max for each POI category to normalize
19     # the count
20     poi_stats = collection.aggregate([
21         {"$project": {
22             "poi_counts": {
23                 "$map": {
24                     "input": {"$objectToArray": "$locations"},
25                     "as": "poi",
26                     "in": {"k": "$$poi.k", "v": {"$size": {"$ifNull": [
27                         "$$poi.v", []]}}}
28                 }
29             }
30         }},
31         {"$unwind": "$poi_counts"},
32         {"$group": {
33             "_id": "$poi_counts.k",
34             "min_count": {"$min": "$poi_counts.v"},
35             "max_count": {"$max": "$poi_counts.v"}
36         }}
37     ])
38
39     # Convert the results into a dictionary
40     global_poi_min_max = {stat["_id"]: {"min": stat["min_count"], "max":
41         : stat["max_count"]} for stat in poi_stats}
42
43     # Normalize the POI counts and compute the scores
44     for category, weight in weights.items():
45         pois = neighborhood_doc.get("locations", {}).get(category, [])
46
47         if category in distinct_categories:
48             # For distinct categories, filter unique entries by address
49             unique_pois = {poi.get("address") for poi in pois if "
50                 address" in poi}
51             count = len(unique_pois)
52         else:
53             # Regular count for other categories
54             count = len(pois)
55
56     global_min = global_poi_min_max.get(category, {}).get("min", 0)
57     global_max = global_poi_min_max.get(category, {}).get("max", 1)
58     # to avoid division by zero

```

```
55     if global_max > global_min:
56         normalized_count = (count - global_min) / (global_max -
57             global_min)
58     else:
59         normalized_count = 0.0
60     total_score += normalized_count * weight
61
62     # Price influence
63     avg_price = neighborhood_doc.get("home_prices", {}).get("avg_price"
64         )
65     # Retrieve global min and max prices from the database for
66     # normalization of the avg price
67     price_stats = collection.aggregate([
68         {"$group": {
69             "_id": None,
70             "min_avg_price": {"$min": "$home_prices.avg_price"},
71             "max_avg_price": {"$max": "$home_prices.avg_price"}
72         }}
73     ])
74     price_stats = next(price_stats, None)
75     min_avg_price = price_stats["min_avg_price"]
76     max_avg_price = price_stats["max_avg_price"]
77     normalized_price = (avg_price - min_avg_price) / (max_avg_price -
78         min_avg_price)
79     total_score += normalized_price * price_weight
80     return total_score
```

Weights

```
1  # Example weighting dictionaries (tweak as you wish)
2
3  students_weights = {
4      "restaurants": 2.0,
5      "museums": 5.0,
6      "nightlife": 8.0,
7      "dogparks": 1.0,
8      "pharmacies": 6.0,
9      "playgrounds": 6.0,
10     "sportvenues": 8.0,
11     "schools": 1.0,
12     "universities": 10.0,
13     "coworking": 7.0,
14     "libraries": 9.0,
15     "supermarkets": 9.0,
16     "transport": 10.0
```

```
17 }
18 # price weight
19 price_weight_students = 10.0
20
21
22 single_couples_weights = {
23     "restaurants": 7.0,
24     "museums": 5.0,
25     "nightlife": 8.0,
26     "dogparks": 5.0,
27     "pharmacies": 6.0,
28     "playgrounds": 1.0,
29     "sportvenues": 7.0,
30     "schools": 1.0,
31     "universities": 1.0,
32     "coworking": 8.0,
33     "libraries": 5.0,
34     "supermarkets": 10.0,
35     "transport": 10.0
36 }
37 # price weight
38 price_weight_single_couples = 6.0
39
40
41 families_weights = {
42     "restaurants": 1.0,
43     "museums": 6.0,
44     "nightlife": 1.0,
45     "dogparks": 10.0,
46     "pharmacies": 7.0,
47     "playgrounds": 10.0,
48     "sportvenues": 3.0,
49     "schools": 10.0,
50     "universities": 1.0,
51     "coworking": 1.0,
52     "libraries": 8.0,
53     "supermarkets": 8.0,
54     "transport": 4.0
55 }
56 # price weight
57 price_weight_families = 7.5
```

Scores

```
1 # Read all neighborhoods
2 all_neighborhoods = list(collection.find({}))
3
4 # --- Ranking for Students ---
5 print("=== Ranking for Students ===")
```



```
6 students_scores = []
7 for nb in all_neighborhoods:
8     score = compute_score(nb, students_weights, price_weight=
9         price_weight_students, collection=collection)
10    students_scores.append({
11        "neighborhood_name": nb["neighborhood_name"],
12        "score": score
13    })
14 # sort by score descending
15 students_scores.sort(key=lambda x: x["score"], reverse=True)
16
17 # transform scores into percentages
18 if students_scores:
19     max_score_students = students_scores[0]["score"]
20     for item in students_scores: # sort of scaling
21         item["percentage"] = (item["score"] / max_score_students) * 100
22         if max_score_students > 0 else 0
23
24 # showing the top 5 neighborhoods
25 for rank, item in enumerate(students_scores[:5], start=1):
26     print(f"{rank}. {item['neighborhood_name']} => {item['percentage']:.2f}%")
27
28 # --- Ranking for Singles/Couples ---
29 print("\n=== Ranking for Singles/Couples ===")
30 single_couples_scores = []
31 for nb in all_neighborhoods:
32     score = compute_score(nb, single_couples_weights, price_weight=
33         price_weight_single_couples, collection=collection)
34     single_couples_scores.append({
35         "neighborhood_name": nb["neighborhood_name"],
36         "score": score
37     })
38 single_couples_scores.sort(key=lambda x: x["score"], reverse=True)
39
40 # transform scores into percentages
41 if single_couples_scores:
42     max_score_single_couples = single_couples_scores[0]["score"]
43     for item in single_couples_scores: # sort of scaling
44         item["percentage"] = (item["score"] / max_score_single_couples)
45         * 100 if max_score_single_couples > 0 else 0
46
47 # showing the top 5 neighborhoods
48 for rank, item in enumerate(single_couples_scores[:5], start=1):
49     print(f"{rank}. {item['neighborhood_name']} => {item['percentage']:.2f}%")
50
```

```

51 # --- Ranking for Families ---
52 print("\n=== Ranking for Families ===")
53 families_scores = []
54 for nb in all_neighborhoods:
55     score = compute_score(nb, families_weights, price_weight=
56         price_weight_families, collection=collection)
57     families_scores.append({
58         "neighborhood_name": nb["neighborhood_name"],
59         "score": score
60     })
61 families_scores.sort(key=lambda x: x["score"], reverse=True)
62
63 # transform scores into percentages
64 if families_scores:
65     max_score_families = families_scores[0]["score"]
66     for item in families_scores: # sort of scaling
67         item["percentage"] = (item["score"] / max_score_families) * 100
68         if max_score_families > 0 else 0
69
68 # showing the top 5 neighborhoods
69 for rank, item in enumerate(families_scores[:5], start=1):
70     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 chosen 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

```

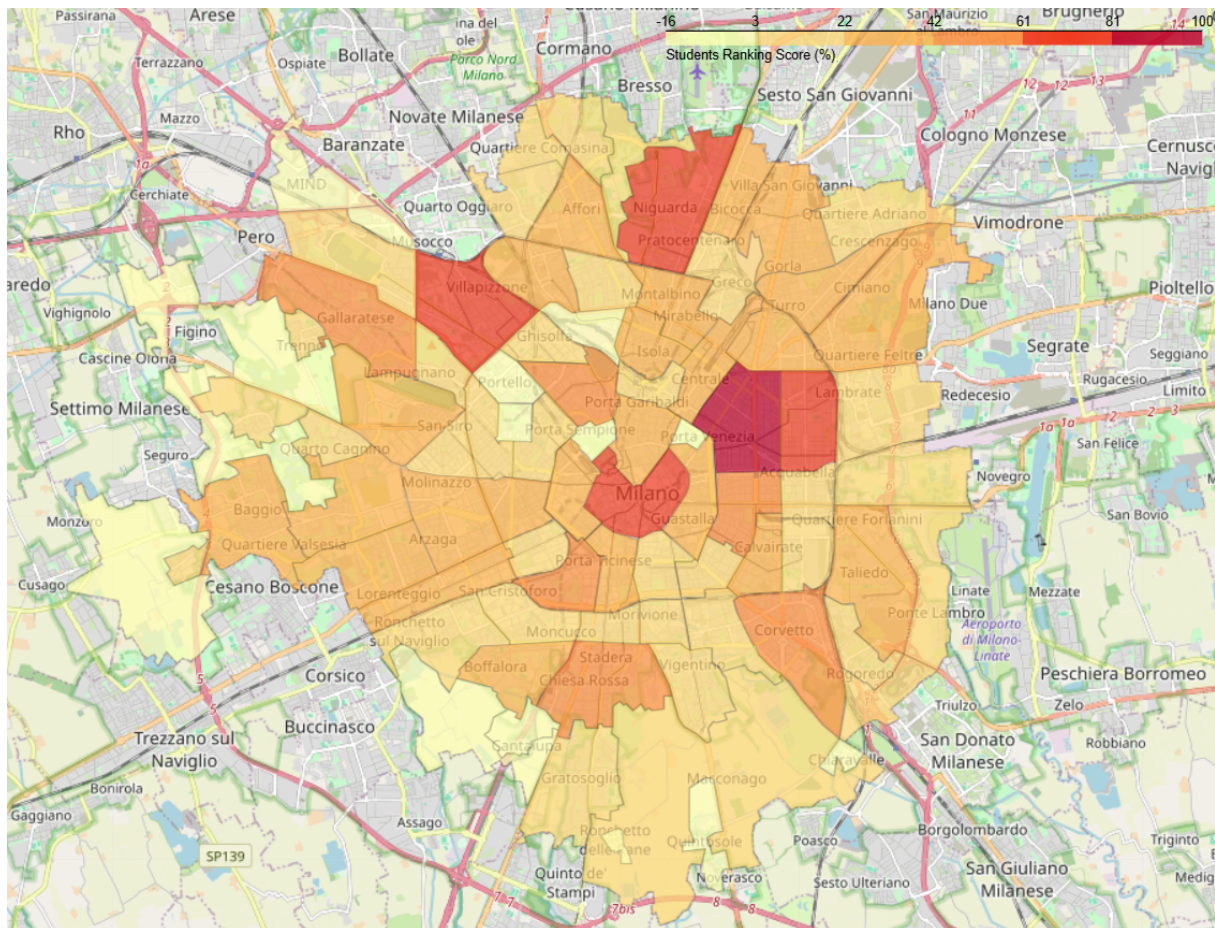


Figure 1: Custom Map

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).

```

1 # Interactive user inputs
2 print("\nPlease specify your preferences:")
3

```

```
4 faculty = input("Enter the faculty you're looking for (e.g., Economia):  
    ").strip().upper() # convert to uppercase  
5 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 # query  
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, "$locations.
40             restaurants.category"]}], 1, 0
41     ]} if restaurant_category else 0,
42     # check for minimum number of restaurants
43     {"$cond": [
44         {"$gte": [{"$size": "$locations.restaurants"},
45             min_restaurants]}], 1, 0
46     ]} if min_restaurants > 0 else 0,
47     # check for parks presence
48     {"$cond": [
49         {"$gt": [{"$size": "$locations.dogparks"}, 0]}], 1,
50         0
51     ]} if parks_required else 0,
52     # check for libraries presence
53     {"$cond": [
54         {"$gt": [{"$size": "$locations.libraries"}, 0]}], 1,
55         0
56     ]} if library_required else 0,
57     # check for coworking presence
58     {"$cond": [
59         {"$gt": [{"$size": "$locations.coworking"}, 0]}], 1,
60         0
61     ]} if coworking_required else 0,
62     # check for sport venue presence
63     {"$cond": [
64         {"$gt": [{"$size": "$locations.sportvenues"}, 0]}],
65         1, 0
66     ]} if sport_venue_required else 0,
67     # check for specific sport venue category match
68     {"$cond": [
69         {"$in": [sport_venue_category, "$locations.
70             sportvenues.category"]}], 1, 0
71     ]} if sport_venue_category else 0,
72     # check for supermarket presence
73     {"$cond": [
74         {"$gt": [{"$size": "$locations.supermarkets"}, 0]}],
75         1, 0
76     ]} if supermarket_required else 0,
77     # check for museum presence
78     {"$cond": [
79         {"$gt": [{"$size": "$locations.museums"}, 0]}], 1, 0
80     ]} if museum_required else 0,
81     # check for pharmacy presence
82     {"$cond": [
83         {"$gt": [{"$size": "$locations.pharmacies"}, 0]}],
84         1, 0
85     ]} if pharmacy_required else 0,
86     # check for playground presence
```



```
78         {"$cond": [  
79             {"$gt": [{"$size": "$locations.playgrounds"}, 0]},  
1, 0  
80         ]} if playground_required else 0,  
81         # check for public transport  
82         {"$cond": [  
83             {"$or": [  
84                 {"$in": ["Metro", "$locations.transport.  
transport_type"]} if metro_required else  
False,  
85                 {"$in": ["Treno", "$locations.transport.  
transport_type"]} if train_required else  
False,  
86                 {"$in": ["Bus", "$locations.transport.  
transport_type"]} if bus_required else False  
87             ]}, 1, 0  
88         ]} if transport_required else 0,  
89         # check for budget in home prices  
90         {"$cond": [  
91             {"$lte": ["$home_prices.avg_price", budget]}, 1, 0  
92         ]} if budget > 0 else 0  
93     ]  
94 }  
95 }  
96 }  
97  
98 # running the aggregation pipeline  
99 pipeline = [  
100     {  
101         "$match": {  
102             "home_prices.avg_price": {"$lte": 2 * budget} # keep  
neighborhoods within twice the budget  
103         }  
104     },  
105     query, # add the match_score field  
106     {"$sort": {  
107         "match_score": -1, # higher match scores first  
108         "home_prices.avg_price": 1 # lower avg prices first  
109     }  
110     },  
111     {"$limit": 3}, # limit to the top 3 neighborhoods  
112     {"$project": { # project only relevant fields for output  
113         "neighborhood_name": 1,  
114         "match_score": 1,  
115         "locations": 1,  
116         "home_prices.avg_price": 1  
117     }}  
118 ]  
119  
120 results = list(collection.aggregate(pipeline))  
121
```

```
122 # Output the top neighborhoods
123 print("\n=== Top 3 Suitable Neighborhoods ===")
124 if results:
125     for i, neighborhood in enumerate(results, start=1):
126         print(f"\n{i}. {neighborhood['neighborhood_name']} (Score: {
127             neighborhood['match_score']})")
128         print(f"    - Average Price: €{neighborhood.get('home_prices',
129             {}).get('avg_price', 'N/A')}")
129
130 fulfilled = []
131 not_fulfilled = []
132
133 # Faculty check
134 if faculty:
135     if any(faculty in uni.get("faculty", []) for uni in
136         neighborhood["locations"].get("universities", [])):
137         fulfilled.append("Faculty")
138     else:
139         not_fulfilled.append("Faculty")
140
141 # Restaurant category check
142 if restaurant_category:
143     if any(restaurant_category in rest.get("category", []) for
144         rest in neighborhood["locations"].get("restaurants", [])):
145         fulfilled.append("Restaurant Category")
146     else:
147         not_fulfilled.append("Restaurant Category")
148
149 # Minimum number of restaurants
150 if min_restaurants > 0:
151     if len(neighborhood["locations"].get("restaurants", [])) >=
152         min_restaurants:
153         fulfilled.append("Minimum Number of Restaurants")
154     else:
155         not_fulfilled.append("Minimum Number of Restaurants")
156
157 # Parks check
158 if parks_required:
159     if len(neighborhood["locations"].get("dogparks", [])) > 0:
160         fulfilled.append("Dog Park")
161     else:
162         not_fulfilled.append("Dog Park")
163
164 # Library check
165 if library_required:
166     if len(neighborhood["locations"].get("libraries", [])) > 0:
167         fulfilled.append("Library")
168     else:
169         not_fulfilled.append("Library")
```



```
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", [])) >
177             0:
178             fulfilled.append("Sport Venue")
179         else:
180             not_fulfilled.append("Sport Venue")
181
182     # Sport venue category check
183     if sport_venue_category:
184         if any(sport_venue_category in venue.get("category", [])
185             for venue in neighborhood["locations"].get("sportvenues", [])):
186             fulfilled.append("Specific Sport Venue Category")
187         else:
188             not_fulfilled.append("Specific Sport Venue Category")
189
190     # Supermarket check
191     if supermarket_required:
192         if len(neighborhood["locations"].get("supermarkets", [])) >
193             0:
194             fulfilled.append("Supermarket")
195         else:
196             not_fulfilled.append("Supermarket")
197
198     # Museum check
199     if museum_required:
200         if len(neighborhood["locations"].get("museums", [])) > 0:
201             fulfilled.append("Museum")
202         else:
203             not_fulfilled.append("Museum")
204
205     # Pharmacy check
206     if pharmacy_required:
207         if len(neighborhood["locations"].get("pharmacies", [])) >
208             0:
209             fulfilled.append("Pharmacy")
210         else:
211             not_fulfilled.append("Pharmacy")
212
213     # Playground check
214     if playground_required:
215         if len(neighborhood["locations"].get("playgrounds", [])) >
216             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
222         # Iterate through transport_list to find the required
223         # transport types
224         if metro_required and any("Metro" in transport.get("
225             transport_type", []) for transport in transport_list):
226             transport_fulfilled.append("Metro")
227         if train_required and any("Treno" in transport.get("
228             transport_type", []) for transport in transport_list):
229             transport_fulfilled.append("Train")
230         if bus_required and any("Bus" in transport.get("
231             transport_type", []) for transport in transport_list):
232             transport_fulfilled.append("Bus")
233
234         # If any transport types are fulfilled, add them to
235         # fulfilled; otherwise, add to not_fulfilled
236         if transport_fulfilled:
237             fulfilled.append(f"Transport ({', '.join(
238                 transport_fulfilled)})")
239         else:
240             not_fulfilled.append("Transport")
241
242     # Budget check
243     if budget > 0:
244         avg_price = neighborhood.get("home_prices", {}).get("
245             avg_price", float("inf"))
246         if avg_price <= budget:
247             fulfilled.append("Budget")
248         else:
249             not_fulfilled.append("Budget")
250
251     # Print fulfilled and not fulfilled
252     print(" - Fulfilled:", ", ".join(fulfilled) if fulfilled else
253         "None")
254     print(" - Not Fulfilled:", ", ".join(not_fulfilled) if
255         not_fulfilled else "None")
256 else:
257     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
9 Do you need a coworking space? (yes/no): yes
10 Do you need a sport venue? (yes/no): yes
11 Enter the sport venue category (e.g., Piscina, Atletica) [optional]:
    Tennis
12 Do you need a supermarket? (yes/no): yes
13 Do you need a museum? (yes/no): no
14 Do you need a pharmacy? (yes/no): yes
15 Do you need a playground? (yes/no): no
16 Do you need public transport? (yes/no): yes
17 Do you need metro service? (yes/no): yes
18 Do you need train service? (yes/no): no
19 Do you need bus service? (yes/no): yes
20 Enter your budget for home prices (price in euros per square meter):
    3200
21
22
23
24 === Top 3 Suitable Neighborhoods ===
25
26 1. Città Studi (Score: 10)
27   - Average Price: €3947.5
28   - Fulfilled: Faculty, Restaurant Category, Minimum Number of
    Restaurants, Library, Coworking Space, Sport Venue, Specific Sport
    Venue Category, Supermarket, Pharmacy, Transport (Metro, Bus)
29   - Not Fulfilled: Budget
30
31 2. Stadera (Score: 9)
32   - Average Price: €2990.625
33   - Fulfilled: Minimum Number of Restaurants, Library, Coworking Space,
    Sport Venue, Specific Sport Venue Category, Supermarket, Pharmacy
    , Transport (Metro, Bus), Budget
34   - Not Fulfilled: Faculty, Restaurant Category
35
36 3. Quarto Cagnino (Score: 9)
37   - Average Price: €3086.4583333333335
38   - Fulfilled: Restaurant Category, Minimum Number of Restaurants,
    Coworking Space, Sport Venue, Specific Sport Venue Category,
    Supermarket, Pharmacy, Transport (Bus), Budget
39   - Not Fulfilled: Faculty, Library
```