GIS Engineering – Individual Project Report

Waterdragen 24[redacted]

Steps to run the Python code

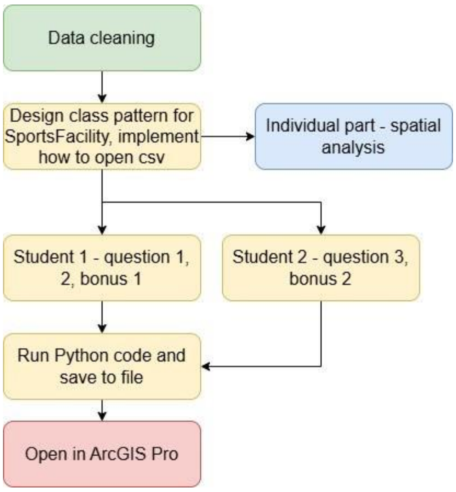
1. Open the Anaconda Navigator and use the arcgis environment with `arcpy`. Open the terminal and change the directory to the directory containing task2.py. Run `` `cd C:/your/path/goes/here` ``, replace the path with the actual path.

2. Ensure that the Python version is at least Python 3.7. Run `` `python -V` `` to check the version

3. Run `` `python task2.py` ``

4. Use Add Data function in ArcGIS Pro and add `sport_facilities` in `./Waterdragen.gdb`

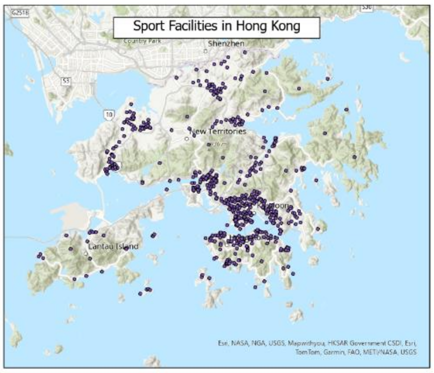
Workflow



This part has been explained in the group report. In short, my group aimed to be collaborative on the data structure and synchronize the method to open the CSV files. However, only the **logic** of `core.py` and `consts.py` are the same and the **implementation details** are **different** in this part. **Question 1** creates the point geometry and **Question 2** finds the nearest facility to the given point.

Question 1

This is a list of `PointGeometry` of all kinds of sport facilities in Hong Kong, imported into ArcGIS.



Question 2

Given a point, find the nearest facility to the point.

For this report, the selected point is Block Z, PolyU. The coordinates are (818630, 836500) in Hong Kong 1980 Grid. The nearest point is saved to a point feature in the database and requires ArcGIS Pro to open it.

```
Successfully found the nearest facility to point (818630, 836500)
Saved as nearest_facility
```

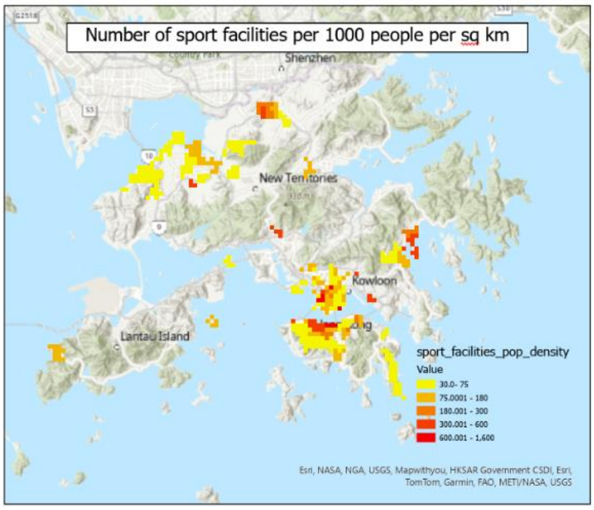
←Success message of the console log for saving to `nearest_facility`.

←Viewing the facility point in ArcGIS Pro with reference to the location of Block Z.

Bonus question 1

Number of sport facilities per 1000 people per square kilometer.

For this question, there are three factors affecting the final result, the number of sport facilities within the districts, the population, and the areas of the districts. The more facilities, lower population, and smaller area of the district will result in higher number of the final index. This spatial analysis aims to view the areas where sport facilities are most abundant, high availability, and locally accessible.



The population data is `Pop_Projection2023-2031`, which is from the CSDI portal. The formula for the spatial analysis is number of facilities within district / population within district area / district area and the unit is facilities per thousand people km^2.

From the figure, it appears that urban areas like Kowloon, Tuen Mun, Yuen Long, Fan Ling, and Northern Hong Kong Island have high number of sport facilities per 1000 people per sq km. In addition, areas like Sai Kung, Shek O, and Tai O also shows high sport facility availability. This is likely due to the low number of population, clustered living areas, with moderate number of sport facilities.

Bonus question 2

There are a total of two tasks printed to the console. The first task is to filter the facilities of a given point within a walkable distance. The second task is to count facilities by dataset and district.

```
Her are the facilities within 500 meters of (818940, 836861):
Basketball Courts in Ho Man Tin Park
Basketball Courts in King's Park High Level Service Reservoir Playground
Basketball Courts in Tsing Chau Street Playground
Basketball Courts in Wuhu Street Temporary Playground
Fitness Rooms in Ho Man Tin Sports Centre
Other Recreation & Sports Facilities in Hung Ling Street Sitting-out Area
Swimming Pools in Ho Man Tin Swimming Pool
```

Filter facilities of a given point within a radius

The point selected is Ho Man Tin station (818940, 836861). The walkable distance is set to 500 meters. The following console outputs the facilities within the walkable distance to Ho Man Tin station.

```
Here are the facilities count by dataset:
Basketball Courts: 305
Other Recreation & Sports Facilities: 144
Badminton Courts: 113
Fitness Rooms: 73
Parks, Zoos and Gardens: 66
Swimming Pools: 44
Sports Grounds: 28
Country Parks: 24
```

The facilities are grouped by dataset (facility type) and counted for each group.

```
Here are the facilities count by district:
KWAI TSING: 61
YUEN LONG: 56
NORTH: 54
KWUN TONG: 52
YAU TSIM MONG: 48
KOWLOON CITY: 46
SHA TIN: 43
EASTERN: 42
TUEN MUN: 42
SHAM SHUI PO: 41
CENTRAL AND WESTERN: 41
TSUEN WAN: 40
WONG TAI SIN: 39
ISLANDS: 37
TAI PO: 35
SAI KUNG: 34
SOUTHERN: 31
WAN CHAI: 30
None: 24
LEI YUE MUN: 1
```

Similarly, the facilities are also grouped by the districts, and counted for each group.

Appendix

From now on this part shall not be counted towards to page count.

Code Review

```python
class FacilityFeature:
    def __init__(self, workspace, csv_folder: str, feature_name: str):
        # Store the workspace for processing data
        self.workspace = workspace
        arcpy.env.workspace = self.workspace
        arcpy.env.overwriteOutput = True
        arcpy.CreateFileGDB_management("./", workspace)

        # Storages for facility and population data
        self.fac_list: list[SportFacility] = read_all_csvs(csv_folder)
        self.feature_name = feature_name
```

Firstly, this code defines a class `FacilityFeature` and the `__init__` constructor function, storing the necessary data such as workspace, facility list, and feature name. It also initializes the GDB, reading from the CSV files, and setting overwrite to true.

```python
    def point_to_feature_class(self):
        # Set the workspace for processing data
        arcpy.env.workspace = self.workspace

        # Allocate exact space for storing the points to avoid memory overhead
        crs_point_geom: list = [None] * len(self.fac_list)

        for index, facility in enumerate(self.fac_list):
            temp_point = arcpy.Point(X=facility.easting, Y=facility.northing, ID=index)

            geom = arcpy.PointGeometry(temp_point, spatial_reference=HK1980GRID)

            crs_point_geom[index] = geom

        print(f"The out feature name is `{self.feature_name}`")
        arcpy.CopyFeatures_management(crs_point_geom, self.feature_name)
        print("Successfully created a point feature class")
```

This function converts the point to feature class by first allocating a list for the `PointGeometry`, and for each item in the list, the program creates a point and then a `PointGeometry` from the Point. Finally the program saves the list of `PointGeometry` to the out feature name.

```python
def add_attribute(self):
    # Set the workspace for processing data
    arcpy.env.workspace = self.workspace
    (FieldManager(self.feature_name)
     .add_field("GMID", "TEXT")
     .add_field("Dataset", "TEXT")
     .add_field("FacilityName", "TEXT")
     .add_field("Address", "TEXT")
     .add_field("District", "TEXT")
     .add_field("Northing", "DOUBLE")
     .add_field("Easting", "DOUBLE")
     .add_field("Latitude", "DOUBLE")
     .add_field("Longitude", "DOUBLE")
     )

    feature_table = arcpy.UpdateCursor(self.feature_name)
    for row, facility in zip(feature_table, self.fac_list):
        row.setValue("GMID", facility.gmid)
        row.setValue("Dataset", facility.dataset)
        row.setValue("FacilityName", facility.fac_name)
        row.setValue("Address", facility.addr)
        row.setValue("District", facility.district)
        row.setValue("Northing", facility.northing)
        row.setValue("Easting", facility.easting)
        row.setValue("Latitude", facility.lat)
        row.setValue("Longitude", facility.lon)
        feature_table.updateRow(row)

    print(f"Successfully added attributes for '{self.feature_name}'")
```

This function adds the necessary attributes to the feature class. The `FieldManager` is a custom class that reduces the boilerplate of repeating the feature class name and allows method chaining.

```python
class FieldManager:
    def __init__(self, out_feature_name: str):
        self.out_feature_name = out_feature_name

    def add_field(self, field_name: str, field_type: str):
        """
        Add the fields to the feature class. Supports method chaining.
        Args:
            field_name: The new field name to be added
            field_type: The data type of the field (e.g. TEXT, DOUBLE)
        """

        # Add multiple fields and types to the same feature name
        arcpy.AddField_management(self.out_feature_name, field_name, field_type)
        return self  # allows us to chain methods for readability
```

Definition of `FieldManager` that helps adding attributes.

```python
# Compulsory Task
def nearest_facility(self, out_name: str, location: tuple[float, float]):
    """
    Find the location of the nearest facility with a given location.

    Args:
        out_name: the facility name to be saved
        location: (x, y) of the location
    """
    # Set the workspace for processing data
    arcpy.env.workspace = self.workspace

    loc_lat, loc_lon = location

    # Create a point object and its geometry
    point = arcpy.Point(X=loc_lon, Y=loc_lat, ID=1)
    point_geom = arcpy.PointGeometry(point, spatial_reference=HK1980GRID)

    arcpy.SpatialJoin_analysis(point_geom, self.feature_name, out_name,
                               join_operation="JOIN_ONE_TO_ONE",
                               match_option="CLOSEST_GEODESIC")
    print(f"Successfully found the nearest facility to point ({loc_lat}, {loc_lon})")
    print(f"Saved as {out_name}\n")
```

This part is for question 2. `nearest_facility()` requires the output point feature name and the location of the starting point. It uses `SpatialJoin_analysis` with one to one relationship matching the closest geodesic to find the nearest point feature to the given point. Then it saves to the database.

```
# Extra questions
def sport_fac_per_people_per_area(self):
    """
    Computes the sport facilities per 1000 people per square kilometer and saves the feature class.
    """

    print("Computing sport facilities per 1000 people per sq km...")
    population_fc = os.path.abspath("./HK_Population/Pop_Projection_2023to2031.shp")
    output_name = self.feature_name + "_pop_density"

    # Step 1: Read the shape file into the database
    population_fc = arcpy.conversion.FeatureClassToFeatureClass(
        population_fc,
        out_path=WORKSPACE,
        out_name="Hk_Pop_Projection"
    )

    field_name = "fac_density"
    arcpy.management.AddField(population_fc, field_name, "DOUBLE")

    with arcpy.da.UpdateCursor(population_fc,
                               ["SHAPE@", "Y2025", "Shape_Area", field_name]) as cursor:
        # Step 2: Iterate all the districts (shapes) in the shape file, reading its fields
        for row in cursor:
            # Skip if population or area is zero/null
            shape, population, area_sqm, _ = row
            if not all((population, area_sqm, population)):
                row[3] = 0
                cursor.updateRow(row)
                continue

            # Step 3: Select the facilities contained by the shape
            arcpy.management.SelectLayerByLocation(
                shape, "CONTAINS", self.feature_name
            )

            # Step 4: Count the facilities selected
            facility_count = int(arcpy.management.GetCount(self.feature_name)[0])
```

This part is for **Bonus question 1**, which computes the sport facilities per 1000 people per square kilometer. It reads the shape file of Pop_projection_2023to2031 and copies into the GDB database, the shape file is a population projection shape file retrieved from the CSDI portal. Next the program iterates through the districts in the shape file and read its fields. “Y2025” is the population field and `“SHAPE@”, “Shape_Area”, and “fac_density”` are the geometry, shape area, and the new field for facility density for the shape file respectively. The population and area should not be zero or null and should be skipped. Then the program uses **`SelectLayerByLocation`** to find the facility points contained by the districts. Then use **`GetCount`** to retrieve the selected points within the list of points.

```python
        # Step 5: Calculate density (facilities per 1000 people per sq km)
        # Formula: (facilities / (population/1000)) / (area_sqm/sqm_to_sqkm)
        area_sqkm = area_sqm / SQM_TO_SQKM
        pop_per_thousand = population / 1000

        if pop_per_thousand > 0 and area_sqkm > 0:
            density = (facility_count / pop_per_thousand) / area_sqkm
        else:
            density = 0

        row[3] = density
        cursor.updateRow(row)

# Step 6: Rasterize the polygons
arcpy.conversion.PolygonToRaster(
    in_features=population_fc,
    value_field=field_name,
    out_rasterdataset=output_name,
    cell_assignment="CELL_CENTER",
    priority_field="NONE",
    cellsize=500
)

# Step 7: High pass filter: >= 30 facilities per 1000 people per sq km
# Then save to file
filtered_raster = arcpy.sa.Con(arcpy.Raster(output_name) >= 30, arcpy.Raster(output_name))
filtered_raster.save(output_name + "_filtered")

# Clean up (replace filtered with just output name)
arcpy.Delete_management(output_name)
arcpy.Rename_management(output_name + "_filtered", output_name)
print(f"Successfully saved bonus question to {output_name}\n")
```

This part continues for **Bonus question 1**. Calculate the density (facility count / thousands of people in district / district in sq km) of individual shapes. Now for the whole shape, it needs to be rasterized and filtered with >= 30 density. Finally the raster is saved to the output feature name in the GDB.

```python
def filter_facility_within_radius(self, location: tuple[float, float], radius):
    """
    List all the facilities of the location within the given radius.

    Args:
        location: (x, y) of the current location
        radius: the radius in meters of the circle
    """

    print(f"Here are the facilities within {radius} meters of {location}:")
    northing, easting = location
    for fac in self.fac_list:
        if sqrt((fac.northing - northing) ** 2 + (fac.easting - easting) ** 2) < radius:
            print(f"{fac.dataset} in {fac.fac_name}")
    print()
```

This part is for Bonus question 2. The program finds the facilities with a given starting point and within the walkable distance. It iterates through the facilities list and calculates the Euclidean distance. If the distance is within the radius, the facility is printed to the console.

```python
def count_facility_by_district(self):
    self.count_facility_by_x("district")

def count_facility_by_dataset(self):
    self.count_facility_by_x("dataset")
```

This part continues **Bonus question 2**. The program counts the facilities grouped by district or dataset, which forwards the function to a helper function.

```python
# helper function
def count_facility_by_x(self, attr: str):
    """
    Count the facilities grouped by the given attribute.

    Args:
        attr: The attribute name
    """

    print(f"Here are the facilities count by {attr}:")
    # Get a list of tuples[item, occurence]
    counts = list(Counter(getattr(fac, attr) for fac in self.fac_list).items())

    # Sort by occurences in descending order
    counts.sort(key=lambda tup: tup[1], reverse=True)

    for field, count in counts:
        # Convert the integer representation back to a string
        print(f"{field}: {count}")
    print()
```

The helper function takes the attribute name and dynamically accesses the fields of `FacilityFeature`. The Counter is in a built-in `collections` module that counts an iterable of items and returns a `dict`-like object. The `dict` is converted into a list and sorted by its occurrence in descending order. Finally the sorted list is printed out, showing the district/type of facility and count pairs.

Appendix

The full console log output

Administrator: C:\WINDOWS\system32\cmd.exe

```
(arcgispro-py3-clone1) C:\[redacted]\Desktop\GIS Engineering\GroupProject>python task2.py
Successfully created geodatabase folder
The out feature name is `sport_facilities`
Successfully created a point feature class
Successfully added attributes for 'sport_facilities'
Successfully found the nearest facility to point (818630, 836500)
Saved as nearest_facility

Her are the facilities within 500 meters of (818940, 836861):
Basketball Courts in Ho Man Tin Park
Basketball Courts in King's Park High Level Service Reservoir Playground
Basketball Courts in Tsing Chau Street Playground
Basketball Courts in Wuhu Street Temporary Playground
Fitness Rooms in Ho Man Tin Sports Centre
Other Recreation & Sports Facilities in Hung Ling Street Sitting-out Area
Swimming Pools in Ho Man Tin Swimming Pool

Here are the facilities count by dataset:
Basketball Courts: 305
Other Recreation & Sports Facilities: 144
Badminton Courts: 113
Fitness Rooms: 73
Parks, Zoos and Gardens: 66
Swimming Pools: 44
Sports Grounds: 28
Country Parks: 24

Here are the facilities count by district:
KWAI TSING: 61
YUEN LONG: 56
NORTH: 54
KWUN TONG: 52
YAU TSIM MONG: 48
KOWLOON CITY: 46
SHA TIN: 43
EASTERN: 42
TUEN MUN: 42
SHAM SHUI PO: 41
CENTRAL AND WESTERN: 41
TSUEN WAN: 40
WONG TAI SIN: 39
ISLANDS: 37
TAI PO: 35
SAI KUNG: 34
SOUTHERN: 31
WAN CHAI: 30
None: 24
LEI YUE MUN: 1

Computing sport facilities per 1000 people per sq km...
Successfully saved bonus question to sport_facilities_pop_density


(arcgispro-py3-clone1) C:\[redacted]\Desktop\GIS Engineering\GroupProject>
```

Implementation of class structure for individual row in the CSV.

For both the individual and the group based task, the class structure is chosen for better readability and type safety. The logic is the same but the implementation details are different. the `@dataclass` decorator generates the __init__ function automatically to reduce boilerplate.

```python
from __future__ import annotations

from dataclasses import dataclass

from consts import FileNames
import csv
import os


def check_na_or_none(s: str):
    return None if not s or s == "N.A." else s

@dataclass
class SportFacility:
    gmid: str
    dataset: str
    fac_name: str
    addr: str | None
    district: str | None
    northing: float
    easting: float
    lat: float
    lon: float

    @classmethod
    def from_csv_row(cls, csv_row: list[str]):
        return cls(
            *(csv_row[i] for i in range(0, 3)),
            *(check_na_or_none(csv_row[i]) for i in range(3, 5)),
            *(float(csv_row[i]) for i in range(5, 9)),
        )
```

Implementation of reading all the CSV files into a list of `SportFacility`

```python
def read_all_csvs(folder: str) -> list[SportFacility]:
    """
    Args:
        folder: folder containing the csv files
    Returns:
        a list merging all the csv files in the directory
    """
    table = []
    for filename in FileNames:
        table.extend(read_csv(os.path.join(folder, filename)))
    return table


def read_csv(filename) -> list[SportFacility]:
    """
    Args:
        filename: path to the csv
    Returns:
        a list of csv rows
    """
    with open(filename, encoding='utf-8') as f:
        csv_reader = csv.reader(f)
        next(csv_reader)  # skip the header
        return [SportFacility.from_csv_row(row) for row in csv_reader]
```

Code to define the constants

```
FileNames = [
    "Badminton_court.csv",
    "Basketball_court.csv",
    "country_parks.csv",
    "fitness_center.csv",
    "other_recreation_sports_facilities.csv",
    "parks_gardens.csv",
    "sports_grounds.csv",
    "swimming_pools.csv",
]

FacilityTypes = [
    "Badminton Courts",
    "Basketball Courts",
    "Country Parks",
    "Fitness Rooms",
    "Other Recreation & Sports Facilities",
    "Parks, Zoos And Gardens",
    "Sports Grounds",
    "Swimming Pools",
]

# Define ArcGIS workspace (update this to your actual geodatabase path)
WORKSPACE = "./Waterdragen.gdb"
```