GIS Engineering – Group Project Report

Group 1

Students:

Waterdragen 24[redacted]D

[redacted] 22[redacted]D

Table of Contents

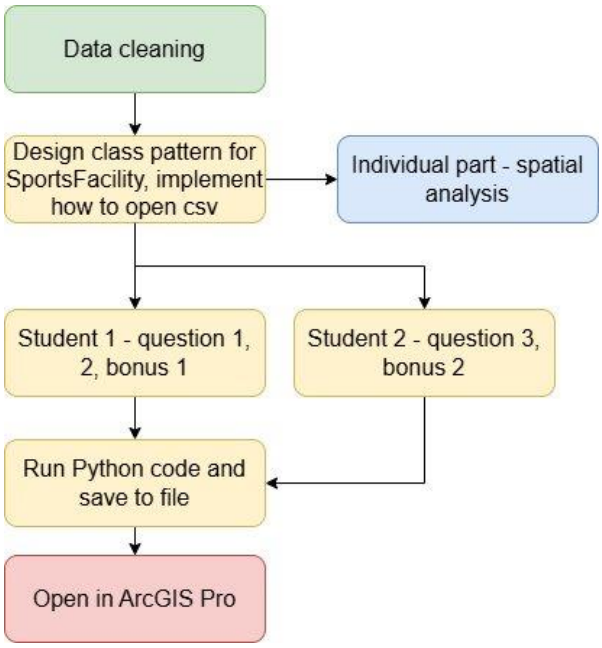
Abstract 2
Background 2
Introduction 2
Workflow 3
Task 1 - Data Cleaning 4
Setting up guidelines - reading the files 4
Task 3 – Group-based spatial analysis 7
Overview of the questions 7
Rationale behind the questions for spatial analysis 7
Question 1 7
Question 2 10
Question 3 13
Bonus question 1 15
Bonus question 2 17
Conclusion 19
Challenges 20

Abstract
Hong Kong has usually ranked among the world's longest-lived populations, which is typically linked to nutritious diets, regular physical activity, low smoking rates, and easy access to outdoor activities. This work uses Python scripting and ArcPy GIS tools to map, quantify, and analyze the spatial distribution of sports and outdoor facilities in Hong Kong. By applying Python scripting, dataclass modules and ArcPy GIS tools—to systematically collect, clean and analyze facility data for various recreation facilities across all 18 districts. After handling disparate CSV sources into a unified SportsFacility data structure, we implement buffer, service-area, network-intersection and focal-statistics analyses to assess (1) per-area facility density, (2) 500 m and 1 km walking-distance catchments, (3) multi-facility overlap zones (≥3 facility types), (4) proximity to a case study site (PolyU), and (5) flat-land accessibility (elevation SD < 20 m). The results reveal strong coverage in new towns and major urban districts—particularly Tsuen Wan, Tuen Mun, Yuen Long, Kowloon and Northern Hong Kong Island. These findings demonstrate the power of Python and ArcPy for urban-planning insights districts where recreational infrastructure could be strengthened.

Background
Hong Kong's high-density urban development and world-leading life expectancy emphasize the importance of accessible recreational spaces for maintaining public health. While diet and personal exercise habits are well documented, spatial disparities in facility distribution and thus practical access remain unexplored at the factor of public health.

Introduction
In this project, we adopt geospatial analysis to examine the distribution and accessibility of key recreational assets across Hong Kong. By developing Python modules and utilizing ArcPy, we automate the conversion of CSV data into ArcGIS feature classes, establish a standard and take in different facility records from multiple CSV sources into a Python dataclass-based structure, ensuring consistent data types and column schemas. Then automate spatial analyses with ArcPy to compute facility densities, walking distance buffers, facility-type intersections (more than 3 types), proximity to a sample urban site (PolyU), and flat-land suitability (elevation standard deviation).

Workflow



For this project we have split the work into multiple stages and among all the members. The first step is data cleaning, this part is done manually to edit the CSV files to be opened by the Python program with ease. Next, we design the class pattern for SportsFacility. We use a class instead of a list for more readable naming. We also implement how to open the CSV to ensure every member of the group is synchronized on this part for both individual and group projects, avoiding to reinvent the wheel. Furthermore, this also enhances the future group collaboration with a common data structure.

Moving on to the core part of the project, we have divided up the work among the two group members. One student shall do questions 1, 2 and bonus question 1, while the other shall do question 3 and bonus question 2. After all members have finished their code, we run the Python code and save the files to the .gdb database file. Finally, we can open the files in ArcGIS Pro and apply some final touch. For example, import the raster intot the new layout and add north arrows, scales, titles, and legends for readability of the maps.

Role of each member of the project

Waterdragen took up the tasks for Question 3, Bonus 2, Data Cleaning, Conclusion, and Challenges in this report, while [redacted] was in charge of Question 1, Question 2, and Bonus question 1, also the Abstract, Background, and Introduction in this report. All group members work collaboratively on designing the class structure for `SportsFacility` and implementing the function to open the CSV files. As for the group presentation, each member have also worked on their corresponding questions.

Task 1 - Data Cleaning

- Removed the Columns with Chinese characters (e.g.: 數據集, 設施名稱, 地址, 地區, 設施種類, 開放時間, 聯絡電話, 傳真號碼, 電郵地址, 網頁, 設施詳情)
- Removed the columns *Opening Hours* and *Last Update*
- Removed the columns involving contact information (e.g.: Telephone, Fax Number, Email Address, Website
- Removed the columns with very long strings (e.g. Facility Type, Facility Details)
- Added empty columns for *District* where necessary. For example, the country parks do not have districts.

Setting up guidelines - reading the files

1. Use CSV reading libraries, such as csv or pandas for reading the CSVs
2. For columns *Address* and *District* if the string is empty or **"N.A."**, the entry will be regarded as `None`
3. Write each row to a class, and store each CSV as a list of the class

This step ensures that every member of the group opens the CSV files the same way as the project diverges into **individual** and **group** tasks, and everyone are **synchronized** in future coding logics.

Data Structure after reading to the files

Field name	Column name	Data type in Python
`gmid`	GMID	`str`
`dataset`	Dataset	`str`
`fac_name`	Facility Name	`str`
`addr`	Address	`str | None`
`district`	District	`str | None`
`northing`	Northing	`float`
`easting`	Easting	`float`
`lat`	Latitude	`float`
`lon`	Longitude	`float`

Each CSV shall have the same columns as shown above.

Code Review for Task 1

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

Code excerpt for defining the structure of a single row in the CSV.

We chose a **class** over a **list**, because the data types are **heterogeneous** (`str, int, float, list, None`), thus using a list might introduce type errors. Besides, the attribute names provide meaning to the columns, whereas for a list, it just uses integer indices, which might affect the readability.

The `__init__`method is **automatically** generated by the `@dataclass` metaclass decorator, based on the type annotations. This approach can avoid some code repetition and increase the readability.

```python
@classmethod
def from_csv_row(cls, csv_row: list[str]):
    return cls(
        csv_row[0],
        csv_row[1],
        csv_row[2],
        check_na_or_empty(csv_row[3], None),
        check_na_or_empty(csv_row[4], None),
        float(csv_row[5]),
        float(csv_row[6]),
        float(csv_row[7]),
        float(csv_row[8]),
    )
```

`From_csv_row` is the factory method for parsing a CSV row. It calls the necessary transformation functions for each string.

`Check_na_or_empty` checks if the string is empty or is “N.A.” and may fall back to the default value.

`Float` converts the string to a float.

```python
def check_na_or_empty(s: str, default):
    if s == "" or s == "N.A.":
        return default
    return s
```

`Check_na_or_empty` returns the default value if the string is empty or “N.A.”, or else return the original string.

```python
def read_all_csvs() -> list[SportFacility]:
    """
    :return: a list merging all the csv files in the directory
    """
    table = []
    for filename in FileNames:
        table.extend(read_csv(filename))
    return table

def read_csv(filename) -> list[SportFacility]:
    """
    :param filename: path to the csv
    :return: 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]
```

`Read_csv` takes a file name and returns a list of rows, where each row is a sport facility. Note that the header of the CSV file must be skipped.

`Read_all_csvs` merges multiple lists of sport facilities into one list of sport facilities.

Task 3 – Group-based spatial analysis

Create python scripts/modules that analyze the spatial distribution of sports and outdoor facilities in Hong Kong.

Overview of the questions

Question 1: Find the areas in Hong Kong with good coverage of different facilities.

Question 2: Areas in Hong Kong to a facility type within walking distance.

Question 3: Find the areas with at least three types of different sports and outdoor facilities within walkable distance.

Bonus question 1: Find the facilities near PolyU.

Bonus question 2: Find the flat land areas with more than three sport facilities.

Rationale behind the questions for spatial analysis

We aim to analyze the areas in Hong Kong where the sport facilities are **accessible**, using various parameters such as number of facilities, flat land, distance, and urban areas. Here is how the questions can achieve our objective:

Question 1: we analyze the number of facilities

Question 2: we analyze the distance

Question 3 is the combination of distance and number of facilities.

Bonus question 2: we analyze the combination of flat land and number of facilities.

Finally we use our common knowledge for the urban areas. We also use a real example as in bonus question 1 – PolyU located in urban areas can serve as a great example of finding nearby facilities. By correlating the areas from the results in each question, we believe we can find out the areas in Hong Kong where the sport facilities are the most accessible.

Question 1

Find the areas in Hong Kong with good coverage of different facilities.

Code Review for Task 3.1

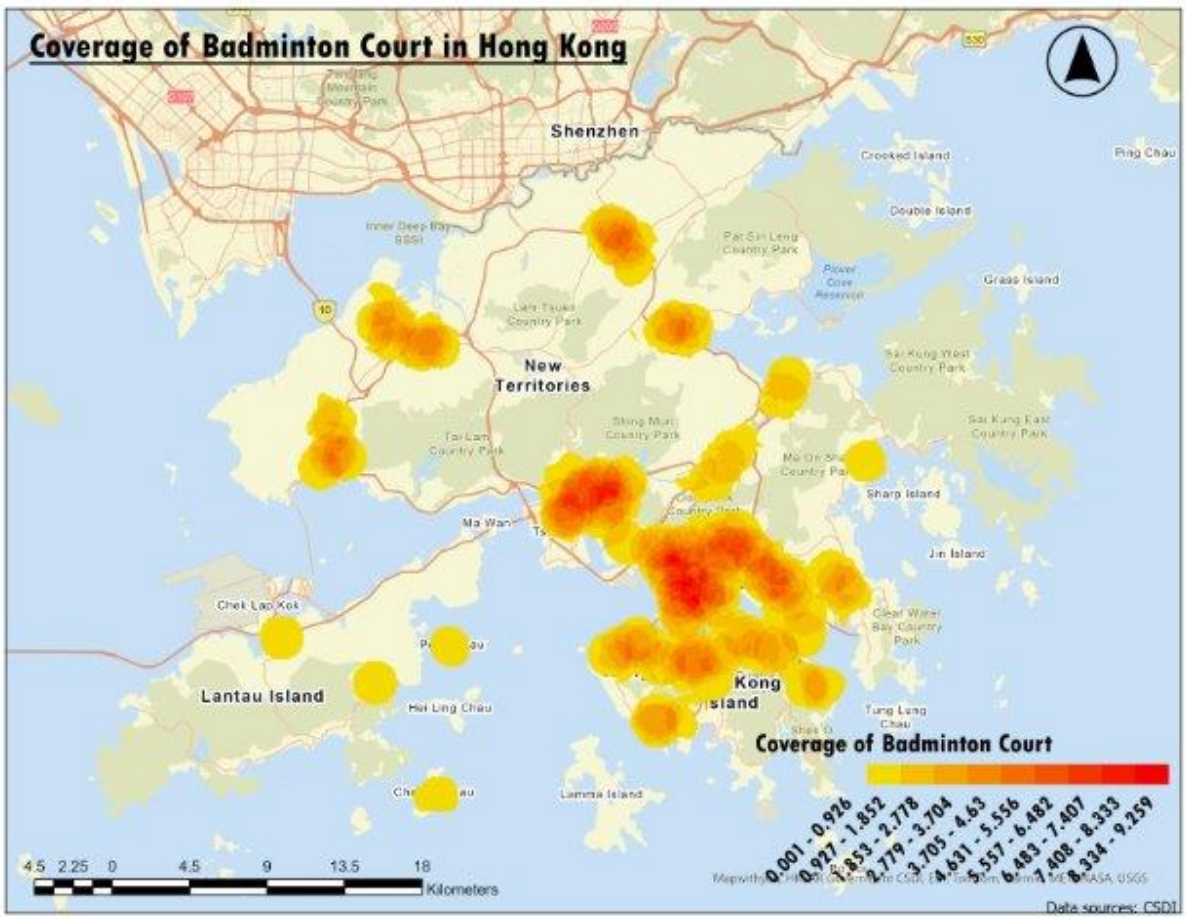
```python
def create_feature_class(sports_data, output_fc_name="facilities_list"):  5 usages
    """
    Convert a list of SportFacility objects into an ArcGIS point feature class.
    :param sports_data: List of SportFacility objects from read_all_csvs
    :param output_fc_name: Name of the output feature class
    :return: Path to the created feature class
    """
    try:
        # Create a new point feature class
        output_fc = arcpy.management.CreateFeatureclass(
            WORKSPACE, output_fc_name, "POINT", spatial_reference=SPATIAL_REFERENCE
        )

        # Add fields for facility attributes
        arcpy.management.AddField(output_fc, "Facility_Type", "TEXT", field_length=50)
        arcpy.management.AddField(output_fc, "Fac_Name", "TEXT", field_length=100)
        arcpy.management.AddField(output_fc, "District", "TEXT", field_length=50)
        # Insert data into the feature cl
                                    output_fc: Any = arcpy.management.CreateFeatureclass(...

        with arcpy.da.InsertCursor(output_fc, ["SHAPE@", "Facility_Type", "Fac_Name", "District"]) as cursor:

            for facility in sports_data:
                # Validate coordinates
                if not (facility.easting and facility.northing):
                    print(f"Warning: Skipping {facility.fac_name} due to missing coordinates.")

                    continue

                point = arcpy.Point(float(facility.easting), float(facility.northing))
                point_geom = arcpy.PointGeometry(point, SPATIAL_REFERENCE)
                facility_type = facility.dataset.replace(".csv", "").replace("_", " ").title()
                cursor.insertRow([point_geom, facility_type, facility.fac_name, facility.district])
        print(f"Feature class '{output_fc}' created successfully.")

        return output_fc

    except arcpy.ExecuteError:
        print(f"Error creating feature class: {arcpy.GetMessages()}")
        raise

    except Exception as e:
        print(f"Unexpected error in create_feature_class: {e}")
        raise
```

This function converts a list of facility objects (from CSV files) into a point feature class in the geodatabase. By adding a new point feature class (e.g., facilities_list) is created with fields. For each facility, easting and northing coordinates are used to create point geometries, to ensure they exist (skipping entries with missing coordinates). Then, Facility attributes (type, name, district) are populated, with the type come from the CSV file name (e.g., badminton_courts.csv becomes "Badminton Courts").

```python
def generate_density_map(facility_type, feature_class, output_file_name, cell_size=100):  1 usage
    """
    Generate a density heatmap for a specific facility type.
    :param facility_type: Type of facility (e.g., 'Badminton Court')
    :param feature_class: Input feature class with all facilities
    :param output_file_name: Name of the output raster
    :param cell_size: Raster cell size in meters (default: 100)
    """

    try:
        # Select facilities of the specified type
        query = f"Facility_Type = '{facility_type}'"
        arcpy.management.SelectLayerByAttribute(feature_class, "NEW_SELECTION", query)

        # Check if any features were selected
        if int(arcpy.management.GetCount(feature_class)[0]) == 0:
            print(f"Warning: No features found for '{facility_type}'. Skipping density map.")

            return
            # Calculate point density
        out_density = PointDensity(feature_class,  population_field: "NONE", cell_size)
        out_density.save(output_file_name)

        print(f"Density heatmap for '{facility_type}' saved as '{output_file_name}'.")

    except arcpy.ExecuteError:
        print(f"Error generating density map for '{facility_type}': {arcpy.GetMessages()}")

    except Exception as e:
        print(f"Unexpected error in generate_density_map: {e}")
```

This function first selects all point features of the specified facility type (e.g. “Badminton Court”) using arcpy.management.SelectLayerByAttribute, then feeds that selection into the Spatial Analyst PointDensity tool. The tool calculates, for each 100 m × 100 m cell, the number of selected points falling within its neighborhood, producing a continuous density raster. Finally, it saves the raster under the given output name, so that hot-spot and cold-spot areas are readily visible.

Results for Question 1

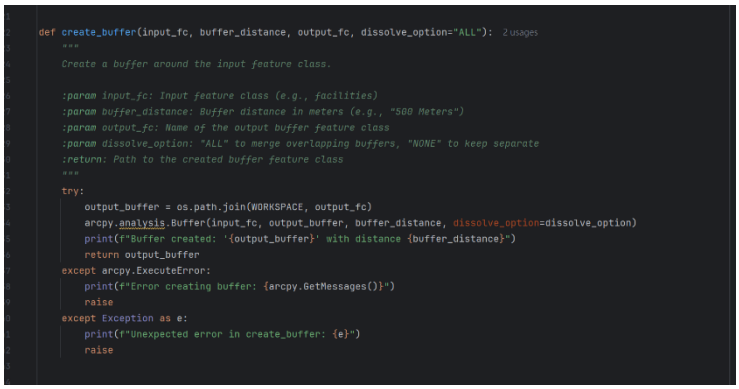


The unit for coverage is in occurrrence per square kilometer while the raster cell size is 100x100 (1 hectare). In the diagram, the areas with the most badminton courts coverage is Tsuen Wan and Kowloon, with over 6 courts per sq km. Other areas such as Tuen Mun, Yuen Long, Fan Ling, Tai Po, and Northern Hong Kong Island also have good coverage, with over 4 courts per sq km.

Question 2

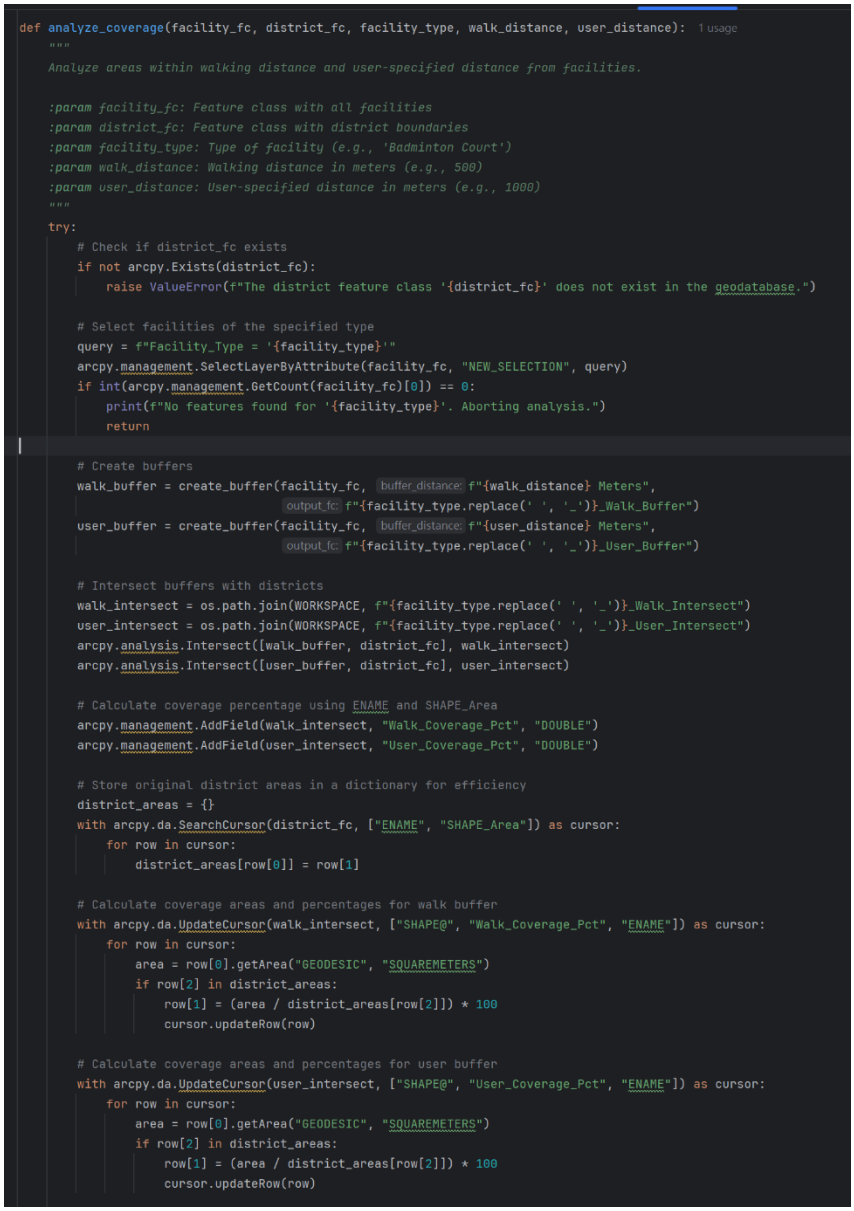
Areas in Hong Kong to a facility type within walking distance.

Code Review for Task 3.1



```python
def create_buffer(input_fc, buffer_distance, output_fc, dissolve_option="ALL"):  2 usages
    """
    Create a buffer around the input feature class.

    :param input_fc: Input feature class (e.g., facilities)
    :param buffer_distance: Buffer distance in meters (e.g., "500 Meters")
    :param output_fc: Name of the output buffer feature class
    :param dissolve_option: "ALL" to merge overlapping buffers, "NONE" to keep separate
    :return: Path to the created buffer feature class
    """
    try:
        output_buffer = os.path.join(WORKSPACE, output_fc)
        arcpy.analysis.Buffer(input_fc, output_buffer, buffer_distance, dissolve_option=dissolve_option)
        print(f"Buffer created: '{output_buffer}' with distance {buffer_distance}")
        return output_buffer
    except arcpy.ExecuteError:
        print(f"Error creating buffer: {arcpy.GetMessages()}")
        raise
    except Exception as e:
        print(f"Unexpected error in create_buffer: {e}")
        raise
```

This functions creates a buffer zone around input features (e.g., facility locations), distance is in meters (e.g., "500 Meters").

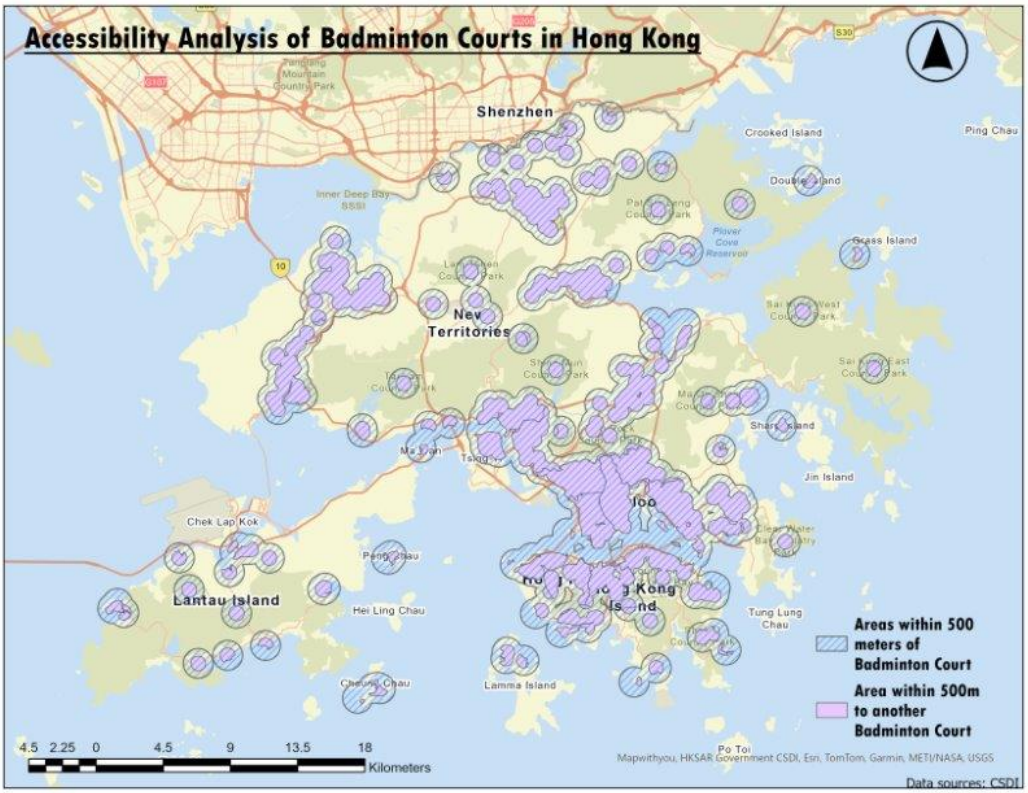


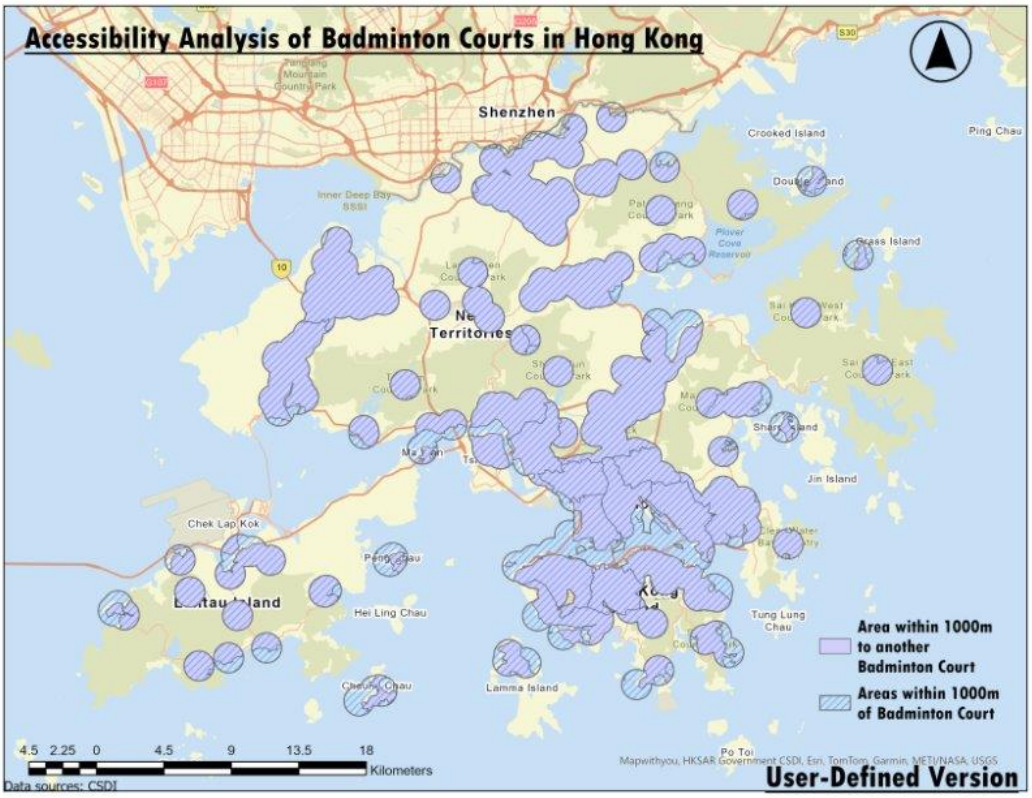
```python
def analyze_coverage(facility_fc, district_fc, facility_type, walk_distance, user_distance):  1 usage
    """
    Analyze areas within walking distance and user-specified distance from facilities.

    :param facility_fc: Feature class with all facilities
    :param district_fc: Feature class with district boundaries
    :param facility_type: Type of facility (e.g., 'Badminton Court')
    :param walk_distance: Walking distance in meters (e.g., 500)
    :param user_distance: User-specified distance in meters (e.g., 1000)
    """
    try:
        # Check if district_fc exists
        if not arcpy.Exists(district_fc):
            raise ValueError(f"The district feature class '{district_fc}' does not exist in the geodatabase.")

        # Select facilities of the specified type
        query = f"Facility_Type = '{facility_type}'"
        arcpy.management.SelectLayerByAttribute(facility_fc, "NEW_SELECTION", query)
        if int(arcpy.management.GetCount(facility_fc)[0]) == 0:
            print(f"No features found for '{facility_type}'. Aborting analysis.")
            return

        # Create buffers
        walk_buffer = create_buffer(facility_fc,  buffer_distance: f"{walk_distance} Meters",
                                    output_fc: f"{facility_type.replace(' ', '_')}_Walk_Buffer")
        user_buffer = create_buffer(facility_fc,  buffer_distance: f"{user_distance} Meters",
                                    output_fc: f"{facility_type.replace(' ', '_')}_User_Buffer")

        # Intersect buffers with districts
        walk_intersect = os.path.join(WORKSPACE, f"{facility_type.replace(' ', '_')}_Walk_Intersect")
        user_intersect = os.path.join(WORKSPACE, f"{facility_type.replace(' ', '_')}_User_Intersect")
        arcpy.analysis.Intersect([walk_buffer, district_fc], walk_intersect)
        arcpy.analysis.Intersect([user_buffer, district_fc], user_intersect)

        # Calculate coverage percentage using ENAME and SHAPE_Area
        arcpy.management.AddField(walk_intersect, "Walk_Coverage_Pct", "DOUBLE")
        arcpy.management.AddField(user_intersect, "User_Coverage_Pct", "DOUBLE")

        # Store original district areas in a dictionary for efficiency
        district_areas = {}
        with arcpy.da.SearchCursor(district_fc, ["ENAME", "SHAPE_Area"]) as cursor:
            for row in cursor:
                district_areas[row[0]] = row[1]

        # Calculate coverage areas and percentages for walk buffer
        with arcpy.da.UpdateCursor(walk_intersect, ["SHAPE@", "Walk_Coverage_Pct", "ENAME"]) as cursor:
            for row in cursor:
                area = row[0].getArea("GEODESIC", "SQUAREMETERS")
                if row[2] in district_areas:
                    row[1] = (area / district_areas[row[2]]) * 100
                    cursor.updateRow(row)

        # Calculate coverage areas and percentages for user buffer
        with arcpy.da.UpdateCursor(user_intersect, ["SHAPE@", "User_Coverage_Pct", "ENAME"]) as cursor:
            for row in cursor:
                area = row[0].getArea("GEODESIC", "SQUAREMETERS")
                if row[2] in district_areas:
                    row[1] = (area / district_areas[row[2]]) * 100
                    cursor.updateRow(row)
```

This functions analyzes the coverage of a specific facility type within districts at two distances. First, The function checks district data, verifies that district_fc exists in the geodatabase. Then, selects specified facility by using a query (e.g., "Facility_Type = 'Badminton Court'"). Followed by, creates Buffer with Walkable distane buffer (e.g., 500 meters) andUser-specified buffer (e.g., 1000 meters). Next, intersects with districts to find areas where buffers overlap district boundaries, then creates two new feature classes (e.g., "Badminton_Court_Walk_Intersect").

After that, This function calculates coverage by adding fields "Walk_Coverage_Pct" and "User_Coverage_Pct" to the intersected feature classes, and stores district areas in a dictionary "ENAME" (district name) and "SHAPE_Area". Afterward, updates the intersected features, and computes the intersected area with getArea("GEODESIC", "SQUAREMETERS"). Lastly, calculates the percentage: (intersected_area / district_area) * 100.

Results for Question 2





There are two areas in each diagram, one contains the water, while the other does not. The shape file of the 18 Districts of Hong Kong is used to trim the buffer generated from the points of the badminton courts, and the shape file can be retrieved from the CSDI portal.

We think that it is important to exclude the areas of the waterbodies since they are inaccessible. We have also set the walkable distance to 500m and the user-defined distance to 1000m. We think that this covers people with moderate to high accessibility for the spatial analysis. Furthermore, we have stacked the 500m on top of the 1000m buffer.

As for the results, it seems like the distribution of the two buffers are similar, with with urban areas like Tuen Mun, Yuen Long, Northern District, Tai Po, Sha Tin, Tsuen Wan, Kowloon, and Hong Kong Island having the most coverage. This shows that there is a positive correlation between the walkable distance and the accessible sports facilities. However, we believe that distance is not the sole factor for accessibility, therefore, we have included other spatial analyses questions to address other parameters for accessibility.

Question 3

Find the areas with at least three types of different sports and outdoor facilities.

Code Review for Task 3.3

```
def make_fac_type_layer(feature_class, facility_type):
    # Create a new feature layer for the current facility type
    fac_type_name = f"{replace_special_chars(facility_type)}_Layer"
    fac_type_layer = arcpy.management.SelectLayerByAttribute(
        feature_class, "NEW_SELECTION",
        f"Facility_Type = '{facility_type}'"
    )
    count = int(arcpy.GetCount_management(fac_type_layer)[0])
    print(f"Selected {count} features for Facility_Type = '{facility_type}'")

    print(f"Created layer {fac_type_name}")
    arcpy.CopyFeatures_management(fac_type_name, f"./shape/{fac_type_name}.shp")
```

This function first filters the buffer with the matching facility type, then saves to a .shp file.

```python
@exception_handler
def three_or_above_facilities(radius: float = 500.0, facilities_list="facilities_list"):
    facilities_buffer = "facilities_buffer"
    three_type_fac_intersect = "ThreeTypeIntersect"
    hk_shape = "./Hong_Kong_18_Districts/HKDistrict18.shp"
    arcpy.Buffer_analysis(facilities_list, facilities_buffer, f"{radius} Meters")

    for fac_type in FacilityTypes:
        make_fac_type_layer(facilities_buffer, fac_type)

    n = len(FacilityTypes)
    ncr = factorial(n) / (6 * factorial(n - 3))
    print(f"Total combinations: {ncr}")

    # Use multiprocessing for each combination of 3 for the intersection
    with mp.Pool() as pool:
        results = [pool.apply_async(process_combination,
                                    args=(index, fac_type_comb))
                   for index, fac_type_comb
                   in enumerate(combinations(FacilityTypes, 3))]
        fac_intersect_list = [res.get() for res in results]

    # Combine the interecting shapes
    arcpy.management.Merge(fac_intersect_list, three_type_fac_intersect)
    arcpy.Clip_analysis(three_type_fac_intersect, hk_shape)

    # Dissolve all the shapes into one shape
    arcpy.Dissolve_management(three_type_fac_intersect, three_type_fac_intersect + "_dissolve")
    three_type_fac_intersect += "_dissolve"
    arcpy.MultipartToSinglepart_management(three_type_fac_intersect, three_type_fac_intersect + "_union")
    three_type_fac_intersect += "_union"

    # Save to a shape file in "./final" folder
    arcpy.CopyFeatures_management(three_type_fac_intersect, "./final/ThreeTypeIntersectFinal.shp")
```

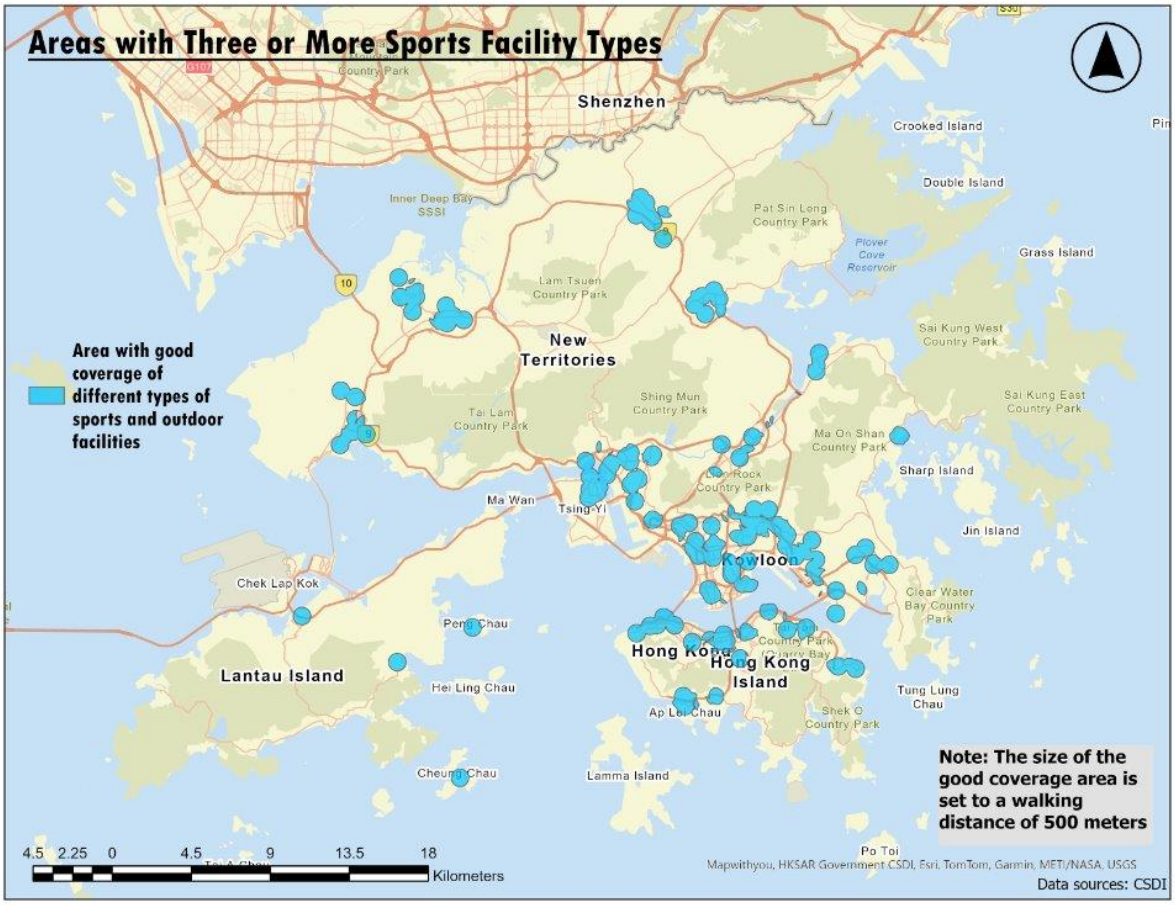
This function has a `@exception_handler` decorator, the decorator handles the exceptions such as `arcpy.ExecuteError` and `Exception`. This function first gives each facility type a buffer and saves to a .shp file. After that, the buffers are grouped in a group of 3, for each combination, the program runs the spatial intersection in parallel. Finally, the program dissolves the individual shapes in the intersecting area, forming one large area.

To reiterate, we have also set the reasonable distance to 500m here.

```python
def process_combination(index, fac_type_comb):
    print("index: ", index)
    fac_type_layers = []
    for fac_type in fac_type_comb:
        fac_type_name = f"./shape/{replace_special_chars(fac_type)}_Layer.shp"
        assert arcpy.Exists(fac_type_name), fac_type_name
        fac_type_layers.append(fac_type_name)

    out_name = f"./shape/FeatureCombination_{index}.shp"
    try:
        arcpy.Intersect_analysis(fac_type_layers, out_name, "ONLY_FID")
    except Exception as e:
        print(e)
        print(f"Warning: {out_name} is ignored")
    return out_name
```

This is the individual function for possessing the spatial intersection operation for each combination of 3 buffer feature classes. This part is run in parallel by the handler in `three_or_above_facilities`.

Results for Question 3



The areas in the figure show the areas that have at least 3 types of sports and outdoor facilities within 500m distance.

It appears that the areas with three or more sports facilities are Tuen Mun, Yuen Long, Fan Ling, Tai Po, Tsuen Wan, Kowloon, and Northern Hong Kong Island.

Bonus question 1

Sports Facilities within walking distance of PolyU.

Code Review for Bonus 2

```python
def filter_facilities_by_type_and_distance(sports_data, lat, lon, distance_km):  1 usage
    """
    Filter facilities by type and within a specified distance from a reference point.

    Args:
        sports_data (list): List of SportFacility objects from CSV files.
        lat (float): Latitude of the reference point (e.g., PolyU).
        lon (float): Longitude of the reference point (e.g., PolyU).
        distance_km (float): Distance threshold in kilometers.

    Returns:
        dict: Dictionary with facility types as keys and lists of facilities as values.
    """
    filtered_by_type = {}
    for facility in sports_data:
        facility_coords = (facility.lat, facility.lon)
        polyu_coords = (lat, lon)
        distance = geodesic(facility_coords, polyu_coords).kilometers
        if distance <= distance_km:
            # Extract and format facility type from dataset
            facility_type = facility.dataset.replace(".csv", "").replace("_", " ").title()
            if facility_type not in filtered_by_type:
                filtered_by_type[facility_type] = []
            filtered_by_type[facility_type].append(facility)
    return filtered_by_type
```

This function Filters facilities within the specified distance from PolyU and groups them by type. First, this function calculates the geodesic distance between PolyU and each facility using geodesic. Then, retains facilities within the threshold. Next, extracts the facility type from the dataset attribute (e.g., converts "basketball_court.csv" to "Basketball Court"). Lastly, groups facilities into a dictionary by type.

Objective

The function is designed to serve the PolyU's stakeholders, enabling students to enjoy recreational activities, faculty and staff to improve campus life, planners to make informed decisions, and local authorities to enhance urban planning.

Results of bonus question 1

We have set the location of PolyU to (22.301612, 114.184546). From the figure, it appears there are 7 basketball courts, 1 badminton court, 1 swimming pool, and 4 other recreational sports facilities within the walkable distance from PolyU. This could indicate that there is moderate accessibility to sports facilities from an urban area.

By the basemap provided in ArcGIS pro, the stakeholders can visually find the path to go to their desired facilities around the PolyU. It is simple but a useful function for the public too. If we change the coordinates in the code, we can investigate the facilities which are close to our desired place. For example, you can change the coordinate to your home, and you can get the thematic map for yourself.

Bonus question 2

Find the areas that are flat and with more than 3 facilities.

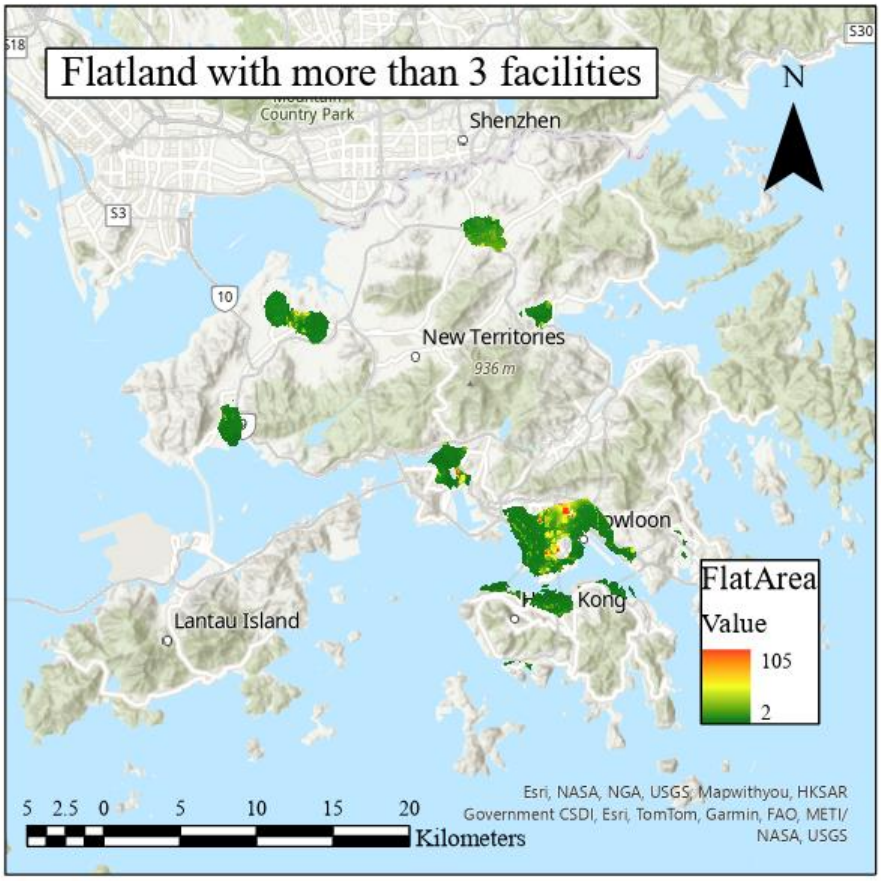
Code Review for Bonus 2

First we have defined “Flatness” - less than 20 meters of standard deviation in elevation within 500x500 meter cell size, for each neighbor circle has a “walkable distance” of 500m. We have also trimmed the buffer to exclude the waterbodies.

```
# Create a binary mask (1 for land, NoData for water)
land_mask = arcpy.sa.Con(arcpy.sa.IsNull(temp_land_raster) == 0, 1)

masked_raster = arcpy.sa.SetNull(land_mask == 0, dtm_raster)
masked_raster.save(land_raster)

arcpy.Delete_management(temp_land_raster)

output_suitable_areas = "FlatArea"
facilities_fc = "facilities_list"

neighborhood = arcpy.sa.NbrCircle(500, "MAP")  # 500m radius circle
elevation_std = arcpy.sa.FocalStatistics(
    land_raster,
    neighborhood,
    "STD",
    "DATA"
)

# Step 2: Identify low-variation areas (e.g., std < 2 meters)
low_variation = arcpy.sa.Con(elevation_std < 20, 1, 0)

# Step 3: Calculate facility density
facility_density = arcpy.sa.PointDensity(
    facilities_fc,
    population_field=None,  # No population field
    cell_size=100,  # Pixel size is 100x100 meters
)

# Step 4: Identify areas with facilities (density > 3, at least 4 facil
has_facilities = arcpy.sa.Con(facility_density > 3, 1, 0)

# Step 5: Combine low-variation and facility areas
suitable_areas = arcpy.sa.Con(
    (low_variation == 1) & (has_facilities == 1), 1, 0
)
```

This is a code except from `bonus2.py`. This shows **five** keys steps of the spatial analysis. First we have used `Con` to apply the condition for the land (set to 1 for land, 0 for waterbodies). Then we apply `FocalStatistic` to calculate the stardard deviation for each neighbor circle, conbined with a filter to only include the areas with SD < 20m. We move on to another spatial analysis – `PointDensity` for the facilities count, and filter with count > 3. Finally we can combine the two analyses with a condition that they both overlap (pixel set to true)

Results for Bonus Question 2



From the diagram, it appears that areas that Tuen Mun, Yuen Long, Fan Ling, Tai Po, Kwai Tsing, Kowloon, and Northern Hong Kong Island are flat areas with at least 4 facilities, and they are also urban areas. There are also some other interesting results, from the previous figures, Sha Tin are usually on the list for more facilities, and being an urban area, being an outlyer likely due to its nature of rugged terrain. There are also some high elevation points in Kowloon, for instance Kowloon Tsai Park in northern Kowloon, there are likely many parks clustered around the elevated land. The map also shows a hollow area in Quarry Hill, likely due to a large rugged land without many parks. Overall, this shows the accessibility of the parks can indeed be influenced by flatness if we account for it.

Conclusion

Overall, this report explores different spatial analysis to view different parameters for accessbility of the sports facilities. Among all the analyses, areas like Tuen Mun, Yuen Long, Fan Long, Tai Po, Kwai Tsing, Kowloon, and Northern Hong Kong Island stand out for their total number of sports facilities, distance, density, and flatness. These areas are also urban areas of Hong Kong. However, there are some outliers, for Sha Tin, the district fulfills most parameters for accessbility but not its flatness. This could affect the accessibility of sports facilities in Sha Tin. Overall, it is still reasonable to say that sports facilities in urban areas tend to be more accessible.

Challenges

We have encountered multiple challenges in the development of the group project. Though it is still manageable for 2 people to work on the code via OneDrive, it is not sizeable enough due to the complexities of the code, tracking differences and changes. One possible solution is to use Github for the group project. Another challenge of the project is the long processing time for arcpy, especially for computationally expensive spatial operations. In addition, the program might fail after a long time, greatly reducing the productivity. We have mitigated this issue by implementing multiprocessing in question 3. Moreover, the intermediate results could not be viewed directly as we progress in the project. This could reduce the productivity of the code, and harder to find out the steps that went wrong. We have mitigated this issue by saving temporary files and view in ArcGIS Pro. Another mitigation method is to use Jupiter Notebook to view the feature classes.

Overall, although there are multiple challenges such as managability, code performance, and viewing intermediate results, we were able to overcome them with some mitigation strategies.