

task1

December 17, 2024

```
[1]: import pandas as pd
import osmnx as ox
import shapely as shp
import numpy as np
import os
import requests
import fynesse
import geopandas as gpd
import yaml
import matplotlib.pyplot as plt
import seaborn as sns
from zipfile import ZipFile
import MySQLdb
import multiprocessing as mp
import re
import dask.dataframe as dd
import dask_geopandas as ddg

# set up database connection

%load_ext sql

with open("./credentials1.yaml") as file:
    credentials = yaml.safe_load(file)

username = credentials["username"]
password = credentials["password"]
url = credentials["url"]
port = credentials["port"]

%config SqlMagic.style = '_DEPRECATED_DEFAULT'

connection_string = f"mysql+pymysql://{username}:{password}@{url}:{port}/
↳ads_2024?local_infile=1"
%sql $connection_string
%sql use ads_2024;
```

```
conn = MySQLdb.connect(host=url, user=username, password=password,
↳database="ads_2024", local_infile=True)
```

```
* mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
0 rows affected.
```

```
[2]: # download data
```

```
for url in [
    # 2021 Census data
    # NS-SEC
    "https://static.ons.gov.uk/datasets/TS062-2021-5.csv",
    "https://www.nomisweb.co.uk/output/census/2021/census2021-ts062.zip",
    "https://www.nomisweb.co.uk/output/census/2021/census2021-ts062-extra.zip",

    # Industry by age categories
    # ("./RM062-2021-3-filtered-2024-11-26T15_05_33Z.csv", "https://static.ons.
↳gov.uk/datasets/3195f3da-ba62-4f47-b03a-51f26092371f/
↳RM062-2021-3-filtered-2024-11-26T15:05:33Z.csv#get-data"),
    "https://www.nomisweb.co.uk/output/census/2021/census2021-ts059.zip",

    # OSM data
    "https://download.openstreetmap.fr/extracts/europe/united_kingdom-latest.
↳osm.pbf",

    # Geographic data of census output areas
    ("./output_areas.csv", "https://open-geography-portalx-ons.hub.arcgis.com/
↳api/download/v1/items/6beafcfd9b9c4c9993a06b6b199d7e6d/csv?layers=0"),
    ("./output_areas.geojson", "https://open-geography-portalx-ons.hub.arcgis.
↳com/api/download/v1/items/6beafcfd9b9c4c9993a06b6b199d7e6d/geojson?
↳layers=0"),
    ("./counties.geojson", "https://open-geography-portalx-ons.hub.arcgis.com/
↳api/download/v1/items/5e0277da82884fd184ff3e1aa55bd414/geojson?layers=0"),
]:

    if isinstance(url, tuple):
        filename, url = url
    else:
        filename = f"./{url.split('/')[-1]}"

    if not os.path.exists(filename):
        print(f"Downloading {url}")
        r = requests.get(url)
        with open(filename, 'wb') as f:
            f.write(r.content)
        print(f"Downloaded {filename}")
```

```

else:
    print(f"Already downloaded {filename}")

    if filename.endswith('.zip') and not os.path.exists(filename.replace('.
↪zip', '')):
        with ZipFile(filename, 'r') as zip_ref:
            zip_ref.extractall()

```

```

Already downloaded ./TS062-2021-5.csv
Already downloaded ./census2021-ts062.zip
Already downloaded ./census2021-ts062-extra.zip
Already downloaded ./census2021-ts059.zip
Already downloaded ./united_kingdom-latest.osm.pbf
Already downloaded ./output_areas.csv
Already downloaded ./output_areas.geojson
Already downloaded ./counties.geojson

```

```

[3]: df = pd.read_csv("census2021-ts062-oa.csv")

# the values in "geography" and "geography code" columns are equal
assert (df['geography'] == df['geography code']).all()

# the values in "geography" column are less than 10 characters
assert (df["geography"].str.len() < 10).all()

```

```

[4]: %%sql
CREATE TABLE IF NOT EXISTS census_nssec (
    -- Year of the census
    year INT NOT NULL,

    -- Geography identifiers
    output_area VARCHAR(10) NOT NULL,

    -- Population counts by NS-SEC classification
    total_residents_16_and_over INT NOT NULL,
    higher_managerial_admin_professional INT NOT NULL,
    lower_managerial_admin_professional INT NOT NULL,
    intermediate_occupations INT NOT NULL,
    small_employers_own_account INT NOT NULL,
    lower_supervisory_technical INT NOT NULL,
    semi_routine_occupations INT NOT NULL,
    routine_occupations INT NOT NULL,
    never_worked_longterm_unemployed INT NOT NULL,
    full_time_students INT NOT NULL,

    -- Constraints

```

```

PRIMARY KEY (year, output_area),
CHECK (total_residents_16_and_over >= 0),
CHECK (higher_managerial_admin_professional >= 0),
CHECK (lower_managerial_admin_professional >= 0),
CHECK (intermediate_occupations >= 0),
CHECK (small_employers_own_account >= 0),
CHECK (lower_supervisory_technical >= 0),
CHECK (semi_routine_occupations >= 0),
CHECK (routine_occupations >= 0),
CHECK (never_worked_longterm_unemployed >= 0),
CHECK (full_time_students >= 0)
) DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1;

```

```

* mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
0 rows affected.

```

[4]: []

```

[5]: if pd.read_sql("SELECT * FROM census_nssec limit 1", conn).empty:
    command = """
    LOAD DATA LOCAL INFILE 'census2021-ts062-oa.csv' \
    INTO TABLE census_nssec \
    FIELDS TERMINATED BY ',' \
    ENCLOSED BY '"' \
    LINES TERMINATED BY '\n' \
    IGNORE 1 LINES \
    (year, output_area, @geocode, total_residents_16_and_over,␣
    ↪higher_managerial_admin_professional, lower_managerial_admin_professional,␣
    ↪intermediate_occupations, small_employers_own_account,␣
    ↪lower_supervisory_technical, semi_routine_occupations, routine_occupations,␣
    ↪never_worked_longterm_unemployed, full_time_students);"""

    %sql $command

```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\3308575725.py:1: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```
if pd.read_sql("SELECT * FROM census_nssec limit 1", conn).empty:
```

```

[6]: %%sql
CREATE TABLE IF NOT EXISTS oas (
    year INT NOT NULL,                -- Year of the census

    -- Area codes and names
    code VARCHAR(10) NOT NULL,        -- Output Area code
    lsoa_code VARCHAR(9) NOT NULL,    -- LSOA code

```

```

lsoa_name VARCHAR(100) NOT NULL,          -- LSOA name in English

-- Geographic coordinates
bng_easting INT NOT NULL,                  -- British National Grid Easting
bng_northing INT NOT NULL,                -- British National Grid Northing
latitude DECIMAL(10,8) NOT NULL,          -- Latitude coordinate
longitude DECIMAL(11,8) NOT NULL,         -- Longitude coordinate

-- Unique identifier
global_id VARCHAR(36) NOT NULL,

-- Geometry
geometry GEOMETRY NOT NULL,               -- Geometry of the output area in
↪WG84

-- Constraints
PRIMARY KEY (year, code)
) DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1;

```

```

* mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
0 rows affected.

```

[6]: []

```

[7]: if not os.path.exists("output_areas.csv"):
    output_areas_gdf = gpd.read_file("output_areas.geojson")

    # set the default geometry column
    output_areas_gdf.set_geometry("geometry", inplace=True)

    output_areas_gdf.geometry.set_crs(epsg=27700, inplace=True)
    output_areas_gdf.geometry = output_areas_gdf.geometry.to_crs(epsg=4326)
    output_areas_gdf.to_csv("output_areas.csv", index=False, sep="|")

if pd.read_sql("SELECT * FROM oas limit 1", conn).empty:
    command = """
    LOAD DATA LOCAL INFILE 'output_areas.csv' \
    INTO TABLE oas \
    FIELDS TERMINATED BY '|' \
    OPTIONALLY ENCLOSED BY '"' \
    LINES TERMINATED BY '\n' \
    IGNORE 1 LINES \
    (@fid, code, lsoa_code, lsoa_name, @welsh, bng_easting, bng_northing,
↪latitude, longitude, global_id, @geometry) \
    SET geometry = ST_GeomFromText(@geometry, 4326), year = 2021;"""

    %sql $command

```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\95361804.py:11: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```
if pd.read_sql("SELECT * FROM oas limit 1", conn).empty:
```

```
[8]: %%sql
CREATE TABLE IF NOT EXISTS hours_worked (
    -- Year of the census
    year INT NOT NULL,

    -- Geography identifiers
    output_area VARCHAR(10) NOT NULL,

    -- Population counts by hours worked
    total_employed_over_16 INT NOT NULL,
    part_time INT NOT NULL,
    worked_15_hours_or_less INT NOT NULL,
    worked_16_to_30_hours INT NOT NULL,
    full_time INT NOT NULL,
    worked_31_to_48_hours INT NOT NULL,
    worked_49_hours_or_more INT NOT NULL,

    -- Constraints
    PRIMARY KEY (year, output_area),
    CHECK (total_employed_over_16 >= 0),
    CHECK (part_time >= 0),
    CHECK (worked_15_hours_or_less >= 0),
    CHECK (worked_16_to_30_hours >= 0),
    CHECK (full_time >= 0),
    CHECK (worked_31_to_48_hours >= 0),
    CHECK (worked_49_hours_or_more >= 0)
) DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1;
```

```
* mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
0 rows affected.
```

```
[8]: []
```

```
[9]: if pd.read_sql("SELECT * FROM hours_worked limit 1", conn).empty:
    command = """LOAD DATA LOCAL INFILE './census2021-ts059-oa.csv' INTO TABLE_
↪`hours_worked` \
    FIELDS TERMINATED BY ',' \
    OPTIONALLY ENCLOSED BY '"' \
    LINES STARTING BY '' \
    TERMINATED BY '\\n' \
    IGNORE 1 LINES \
```

```
(year, output_area, @geography_code, \
total_employed_over_16, part_time, \
worked_15_hours_or_less, worked_16_to_30_hours, \
full_time, worked_31_to_48_hours, \
worked_49_hours_or_more);"""
```

```
%sql $command
```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\3280249861.py:1: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```
if pd.read_sql("SELECT * FROM hours_worked limit 1", conn).empty:
```

```
[10]: %%sql
CREATE TABLE IF NOT EXISTS osm_features (
    -- Unique identifier
    osmid INT NOT NULL,

    -- Area of the feature
    area DOUBLE,

    -- Tags
    amenity VARCHAR(255),
    building VARCHAR(255),
    building_use VARCHAR(255),
    building_levels INT,
    height FLOAT,
    shop VARCHAR(255),
    leisure VARCHAR(255),
    sport VARCHAR(255),
    landuse VARCHAR(255),
    office VARCHAR(255),
    railway VARCHAR(255),
    public_transport VARCHAR(255),
    highway VARCHAR(255),
    aeroway VARCHAR(255),
    waterway VARCHAR(255),
    man_made VARCHAR(255),

    -- Geometry
    geometry GEOMETRY NOT NULL,

    -- Constraints
    PRIMARY KEY (osmid),
    CHECK (osmid >= 0)
) DEFAULT CHARSET=utf8 COLLATE=utf8_bin AUTO_INCREMENT=1;
```

```
* mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
0 rows affected.
```

```
[10]: []
```

```
[11]: def parse_height_to_meters(height_series):
    def convert_to_meters(height):
        if pd.isnull(height):
            return None

        height_str = str(height).strip().lower()
        pure_number_pattern = r'^([\d,.\s*(m|meter|meters|metre|metres|ft|foot|feet)?$'
        feet_inches_pattern = r'^(\d+)\s*(\d+)?\s*$'

        # Try pure number with optional unit
        match = re.match(pure_number_pattern, height_str)

        if match:
            value, unit = match.groups()
            # Replace comma with dot for decimal conversion if necessary
            value = value.replace(',', '.')

            try:
                value = float(value)
            except ValueError:
                return None

            # Define conversion factors
            unit = unit.lower() if unit else 'm' # Assume meters if no unit
            provided

            if unit in ['m', 'meter', 'meters', 'metre', 'metres']:
                return value
            elif unit in ['ft', 'foot', 'feet']:
                return value * 0.3048 # 1 foot = 0.3048 meters
            else:
                return None

        # Try feet and inches pattern
        match = re.match(feet_inches_pattern, height_str)
        if match:
            feet, inches = match.groups()
            try:
                feet = int(feet)
                inches = int(inches) if inches else 0
            except ValueError:
```



```

        return np.nan # Unable to convert to integers

    total_meters = feet * 0.3048 + inches * 0.0254
    return round(total_meters, 4) # Rounded to 4 decimal places

# If no pattern matches, return None
return None

# Apply the conversion to each element in the Series
return height_series.apply(convert_to_meters)

```

```

[12]: headers = "osmid|area|amenity|building|building:use|building:
↳levels|height|shop|leisure|sport|landuse|office|railway|public_transport|highway|aeroway|wa
↳split("|")

# for every geojson file in the osm_features directory, save it to a csv

def process_file(file):
    if not file.endswith(".geojson"):
        return

    name = file.split(".")[0]

    if os.path.exists(f"osm_features/{name}.csv"):
        return

    print(f"Processing {name}")

    pois = gpd.read_file(f"osm_features/{file}")

    pois["height"] = parse_height_to_meters(pois["height"])
    pois["building:levels"] = pois["building:levels"].apply(lambda x: int(x) if
↳pd.notnull(x) and x.isdigit() else None)
    pois["area"] = pois["geometry"].set_crs(epsg=4326).to_crs(epsg=27700).area

    pois = pois.dropna(subset=["geometry"])

    to_save = pd.DataFrame(columns=headers)

    for col in headers:
        if col in pois.columns:
            to_save[col] = pois[col]

    to_save.to_csv(f"osm_features/{name}.csv", sep="|", index=False)

files = os.listdir("osm_features")

```

```
for file in files:
    process_file(file)
```

[13]: *# for each csv file in the osm_features directory, load it into the database*

```
if pd.read_sql("SELECT * FROM osm_features LIMIT 1", conn).empty:
    for file in os.listdir("osm_features"):
        if not file.endswith(".csv"):
            continue

        name = file.split(".")[0]
        print(f"Processing {name}")

        command = f"""
LOAD DATA LOCAL INFILE 'osm_features/{file}' \
INTO TABLE osm_features \
FIELDS TERMINATED BY '|' \
ENCLOSED BY '"' \
LINES TERMINATED BY '\\n' \
IGNORE 1 LINES \
(osmid, @area, @amenity, @building, @building_use, building_levels, \
↪@height, @shop, @leisure, @sport, @landuse, @office, @railway, \
↪@public_transport, @highway, @aeroway, @waterway, @man_made, @geo) \
SET geometry = ST_GeomFromText(@geo, 4326), \
area = NULLIF(@area, '0.0'), \
amenity = NULLIF(@amenity, ''), \
building = NULLIF(@building, ''), \
building_use = NULLIF(@building_use, ''), \
shop = NULLIF(@shop, ''), \
leisure = NULLIF(@leisure, ''), \
sport = NULLIF(@sport, ''), \
landuse = NULLIF(@landuse, ''), \
office = NULLIF(@office, ''), \
railway = NULLIF(@railway, ''), \
public_transport = NULLIF(@public_transport, ''), \
highway = NULLIF(@highway, ''), \
aeroway = NULLIF(@aeroway, ''), \
waterway = NULLIF(@waterway, ''), \
man_made = NULLIF(@man_made, ''), \
height = NULLIF(@height, 'NaN');"""

%sql $command
```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\3375045728.py:3: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```
if pd.read_sql("SELECT * FROM osm_features LIMIT 1", conn).empty:
```

```
[14]: # census nssec table
      %sql ALTER TABLE census_nssec ADD INDEX idx_output_area (output_area);
      # output area geometry table
      %sql ALTER TABLE oas ADD SPATIAL INDEX idx_geometry (geometry);
      # hours worked
      %sql ALTER TABLE hours_worked ADD INDEX idx_output_area (output_area);

      * mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
      (pymysql.err.OperationalError) (1061, "Duplicate key name 'idx_output_area'")
      [SQL: ALTER TABLE census_nssec ADD INDEX idx_output_area (output_area);]
      (Background on this error at: https://sqlalche.me/e/20/e3q8)
      * mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
      (pymysql.err.OperationalError) (1061, "Duplicate key name 'idx_geometry'")
      [SQL: ALTER TABLE oas ADD SPATIAL INDEX idx_geometry (geometry);]
      (Background on this error at: https://sqlalche.me/e/20/e3q8)
      * mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
      (pymysql.err.OperationalError) (1061, "Duplicate key name 'idx_output_area'")
      [SQL: ALTER TABLE hours_worked ADD INDEX idx_output_area (output_area);]
      (Background on this error at: https://sqlalche.me/e/20/e3q8)
```

```
[15]: %%sql
      ALTER TABLE osm_features ADD INDEX idx_amenity (amenity),
      ADD INDEX idx_building (building),
      ADD INDEX idx_building_use (building_use),
      ADD INDEX idx_shop (shop),
      ADD INDEX idx_leisure (leisure),
      ADD INDEX idx_sport (sport),
      ADD INDEX idx_landuse (landuse),
      ADD INDEX idx_office (office),
      ADD INDEX idx_railway (railway),
      ADD INDEX idx_public_transport (public_transport),
      ADD INDEX idx_highway (highway),
      ADD INDEX idx_aeroway (aeroway),
      ADD INDEX idx_waterway (waterway),
      ADD INDEX idx_man_made (man_made),
      ADD SPATIAL INDEX idx_geometry (geometry);
```

```
      * mysql+pymysql://root:***@localhost:3306/ads_2024?local_infile=1
      (pymysql.err.OperationalError) (1061, "Duplicate key name 'idx_amenity'")
      [SQL: ALTER TABLE osm_features ADD INDEX idx_amenity (amenity),
      ADD INDEX idx_building (building),
      ADD INDEX idx_building_use (building_use),
      ADD INDEX idx_shop (shop),
      ADD INDEX idx_leisure (leisure),
      ADD INDEX idx_sport (sport),
      ADD INDEX idx_landuse (landuse),
```

```

ADD INDEX idx_office (office),
ADD INDEX idx_railway (railway),
ADD INDEX idx_public_transport (public_transport),
ADD INDEX idx_highway (highway),
ADD INDEX idx_aeroway (aeroway),
ADD INDEX idx_waterway (waterway),
ADD INDEX idx_man_made (man_made),
ADD SPATIAL INDEX idx_geometry (geometry);]
(Background on this error at: https://sqlalche.me/e/20/e3q8)

```

```
[16]: available_parallelism = mp.cpu_count()
```

```
[17]: def load_geometry_from_wkt(df, geo_col="geometry", wkt_col="wkt", crs="EPSG:
      ↪4326"):
      df[geo_col] = df[wkt_col].apply(shp.wkt.loads)
      df.drop(columns=[wkt_col], inplace=True)
      gdf = gpd.GeoDataFrame(df, geometry=geo_col, crs=crs)
      return gdf
```

```
[18]: oas = load_geometry_from_wkt(pd.read_sql("SELECT *, ST_AsText(geometry) as wkt_
      ↪from oas", conn))
      oas = ddg.from_geopandas(oas, npartitions=available_parallelism)
```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\705512205.py:1: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```
oas = load_geometry_from_wkt(pd.read_sql("SELECT *, ST_AsText(geometry) as wkt
from oas", conn))
```

```
[19]: cutoffs = [
      (1000000, 100),
      (10000, 1_000),
      (5000, 10_000),
      (2000, 50_000),
      (1000, 1_000_000),
      (100, 100_000_000)
      ]

      for radius, _ in cutoffs:
          oas[f"{radius}m"] = oas["geometry"].to_crs(epsg=6933).buffer(radius).
          ↪to_crs(epsg=4326).simplify(radius/10)
```

```
[20]: import hashlib

      if not os.path.exists("oa_osm_joined"):
          os.mkdir("oa_osm_joined")
```

```

keys_to_keep = ["code", "osmid", "distance", "area", "building_levels"]

def collect_features_for_condition(condition, name):
    condition_hash = hashlib.md5(condition.encode()).hexdigest()
    filepath = f"oa_osm_joined/{name}_{condition_hash}.csv"

    if os.path.exists(filepath):
        print(f"Already processed {condition}, loading from file...")
        df = dd.read_csv(f"{filepath}/*.part", sep="|", index_col=False)
        if df.columns.equals(keys_to_keep):
            return df.set_index("code")
        else:
            df = df[keys_to_keep]
            df.to_csv(filepath, sep="|", index=False)
            return df.set_index("code")
    else:
        print(f"Processing {condition}...")
        gdf = ddg.from_geopandas(load_geometry_from_wkt(pd.read_sql(f"SELECT *,
↪ST_AsText(geometry) as wkt from osm_features where {condition}", conn)),
↪npartitions=available_parallelism)
        size = len(gdf)

        print(f"{condition}")
        print(f"Found {size} features")

        for r, cutoff in cutoffs:
            if size < cutoff:
                radius = r
                break

        print(f"Looking within {radius}m")

        oas_reset = oas.set_geometry(f"{radius}m").reset_index(drop=True)
        joined = gdf.sjoin(oas_reset, predicate="intersects")
        joined = joined.compute()

        print(f"Found {len(joined)} relationships, calculating distances...")
        joined["distance"] = gpd.GeoSeries(joined["geometry_left"], crs="EPSG:
↪4326").to_crs(epsg=6933).distance(
            gpd.GeoSeries(joined["geometry_right"], crs="EPSG:4326").
↪to_crs(epsg=6933)
        )

        joined["building_levels"] = pd.to_numeric(joined["building_levels"],
↪errors="coerce")

```

```

print(f"saving...")
df = joined[keys_to_keep]
df = dd.from_pandas(df, npartitions=available_parallelism)
df.to_csv(filepath, sep="|", index=False)
return df.set_index("code")

```

```

[21]: from IPython.display import display

osm_properties = pd.DataFrame()

for property in ["amenity", "building", "building_use", "shop", "leisure", "sport", "landuse", "office", "railway", "public_transport", "highway", "aeroway", "waterway", "man_made"]:
    new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features", conn)[property]
    osm_properties[property] = new_series

osm_properties

```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```

new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features", conn)[property]

```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```

new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features", conn)[property]

```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```

new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features", conn)[property]

```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```

new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features", conn)[property]

```

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```

new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.

```

```

new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\767566527.py:6: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.
new_series = pd.read_sql(f"SELECT DISTINCT {property} FROM osm_features",
conn)[property]

```

```

[21]:
amenity building building_use \
0 None None None
1 24hr Recovery * Car_sales
2 Airfield Centre_for_Ecology_and_Hydrology Library
3 Brewpub & Kitchen Clubhouse Recreation
4 Care Home Coffee_Van Scout Hut
.. ...
497 wildlife_hide social_facility NaN
498 workshop sorting_office NaN
499 yacht_club spa NaN
500 yard sport NaN
501 youth_centre sports NaN

shop leisure \
0 None None
1 3d_printing Entertainment_complex_and_restaurant
2 ==_closed_== MUGA
3 African_Goods_Shop Open_Access_Area
4 Bathroom_Fitters abseil_tower
.. ...
497 health NaN
498 health & wellness NaN
499 health_and_beauty NaN
500 health_care NaN
501 health_food NaN

sport landuse office \
0 None None None
1 * Community_Facility Airport_Administration
2 10pin Lifeboat_station_area Barkers_Farn_House
3 10pin;billiards Llwynderw_Fawr Bowerhouse
4 10pin;ice_skating aeroway Building_Consultants
.. ...
497 greyhound_racing NaN NaN
498 greyhound_racing NaN NaN
499 gym NaN NaN
500 gym;climbing;caving NaN NaN
501 gym;sauna;swimming NaN NaN

```


	railway	public_transport	highway	aeroway \
0	None	None	None	None
1	abandoned	pay_scale_area	abandoned	abandoned
2	bridge	platform	bridleway	aerodrome
3	coaling_facility	shelter;waiting_room	bus_guideway	aerodrome_marking
4	construction	station	bus_stop	airstrip
..
497	NaN	NaN	NaN	NaN
498	NaN	NaN	NaN	NaN
499	NaN	NaN	NaN	NaN
500	NaN	NaN	NaN	NaN
501	NaN	NaN	NaN	NaN

	waterway	man_made
0	None	None
1	*	2
2	Mill Creek	Dock
3	Swan Cutting	Environmental Services Depot
4	abandoned	Flood water offtake structures
..
497	NaN	NaN
498	NaN	NaN
499	NaN	NaN
500	NaN	NaN
501	NaN	NaN

[502 rows x 14 columns]

1 TODO: Kernel Density Estimation

2 TODO: Land Use Entropy

```
[22]: queries = {
    "higher_education": r"amenity like '%university%' or amenity like_
↪ '%research_institute%' or amenity like '%college%'",
    "other_education": r"amenity like '%school%' or amenity like_
↪ '%kindergarten%'",
    "public_working_space": r"amenity like '%library%' or amenity like '%cafe%'_
↪ or amenity like '%coffee_shop%' or amenity like '%coworking_space%'",
    "restaurants_and_fast_food": r"amenity like '%restaurant%' or amenity like_
↪ '%fast_food%'",
    "entertainment_venues": r"amenity like '%cinema%' or amenity like_
↪ '%theatre%' or amenity like '%bar%' or amenity like '%pub%' or amenity like_
↪ '%nightclub%' or amenity like '%music_venue%'",
```

```

    "gyms_and_sports_centres": r"leisure like '%gym%' or leisure like
↳ '%sports_centre%' or leisure like '%fitness%' or sport is not null",
    "green_spaces": r"leisure like '%park%' or leisure like '%playground%' or
↳ leisure like '%garden%' or landuse like '%recreation_ground%' or landuse
↳ like '%meadow%' or landuse like '%grass%'",
    "bookstores_and_copysshops": r"shop like '%bookstore%' or shop like
↳ '%copyshop%'",
    "public_transport": r"highway like '%bus%' or amenity like '%bus%' or
↳ public_transport like '%bus%' or railway like '%station%' or railway like
↳ '%subway%' or public_transport like '%train%' or public_transport like
↳ '%subway%' or public_transport like '%tram%' or railway like '%tram%' or
↳ public_transport like '%light_rail%' or railway like '%light_rail%'",
    "cycling_and_walking_infrastructure": r"amenity like '%cycle%' or highway
↳ like '%cycle%' or highway like '%foot%' or highway like '%path%' or highway
↳ like '%pedestrian%'",
    "healthcare_facilities": r"amenity like '%hospital%' or amenity like
↳ '%clinic%' or amenity like '%pharmacy%' or amenity like '%doctor%' or
↳ amenity like '%dentist%' or amenity like '%nursing_home%' or amenity like
↳ '%health_centre%' or amenity like '%veterinary%' or amenity like
↳ '%optician%'",
    "student_accommodation": r"building like '%dormitory%' or building like
↳ '%student_accommodation%' or amenity like '%student_accommodation%'",
    "community_centres_and_youth_clubs": r"amenity like '%community_centre%' or
↳ amenity like '%youth_club%'",
    "beaches": r"leisure like '%beach%'",
    "rivers": r"waterway like '%river%'",
}

features = {
    name: collect_features_for_condition(query, name)
    for name, query in queries.items()
}

```

Already processed amenity like '%university%' or amenity like '%research_institute%' or amenity like '%college%', loading from file...

Already processed amenity like '%school%' or amenity like '%kindergarten%', loading from file...

Already processed amenity like '%library%' or amenity like '%cafe%' or amenity like '%coffee_shop%' or amenity like '%coworking_space%', loading from file...

Already processed amenity like '%restaurant%' or amenity like '%fast_food%', loading from file...

Already processed amenity like '%cinema%' or amenity like '%theatre%' or amenity like '%bar%' or amenity like '%pub%' or amenity like '%nightclub%' or amenity like '%music_venue%', loading from file...

Already processed leisure like '%gym%' or leisure like '%sports_centre%' or leisure like '%fitness%' or sport is not null, loading from file...

Already processed leisure like '%park%' or leisure like '%playground%' or

leisure like '%garden%' or landuse like '%recreation_ground%' or landuse like '%meadow%' or landuse like '%grass%', loading from file...

Already processed shop like '%bookstore%' or shop like '%copyshop%', loading from file...

Already processed highway like '%bus%' or amenity like '%bus%' or public_transport like '%bus%' or railway like '%station%' or railway like '%subway%' or public_transport like '%train%' or public_transport like '%subway%' or public_transport like '%tram%' or railway like '%tram%' or public_transport like '%light_rail%' or railway like '%light_rail%', loading from file...

Already processed amenity like '%cycle%' or highway like '%cycle%' or highway like '%foot%' or highway like '%path%' or highway like '%pedestrian%', loading from file...

Already processed amenity like '%hospital%' or amenity like '%clinic%' or amenity like '%pharmacy%' or amenity like '%doctor%' or amenity like '%dentist%' or amenity like '%nursing_home%' or amenity like '%health_centre%' or amenity like '%veterinary%' or amenity like '%optician%', loading from file...

Already processed building like '%dormitory%' or building like '%student_accommodation%' or amenity like '%student_accommodation%', loading from file...

Already processed amenity like '%community_centre%' or amenity like '%youth_club%', loading from file...

Processing leisure like '%beach%'...

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\1283237874.py:23: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```
gdf = ddg.from_geopandas(load_geometry_from_wkt(pd.read_sql(f"SELECT *,
ST_AsText(geometry) as wkt from osm_features where {condition}", conn)),
npartitions=available_parallelism)
```

leisure like '%beach%'

Found 8 features

Looking within 1000000m

Found 1510911 relationships, calculating distances...

saving...

Processing waterway like '%river%'...

C:\Users\varun\AppData\Local\Temp\ipykernel_28252\1283237874.py:23: UserWarning: pandas only supports SQLAlchemy connectable (engine/connection) or database string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested. Please consider using SQLAlchemy.

```
gdf = ddg.from_geopandas(load_geometry_from_wkt(pd.read_sql(f"SELECT *,
ST_AsText(geometry) as wkt from osm_features where {condition}", conn)),
npartitions=available_parallelism)
```

waterway like '%river%'

Found 32396 features

Looking within 2000m

Found 810243 relationships, calculating distances...
saving...

```
[23]: oas_with_diameter = oas.join(oas["geometry"].to_crs(epsg=6933).apply(lambda x:
    ↪x.minimum_rotated_rectangle.length, meta=('geometry', 'float64'))).
    ↪rename("diameter"))
oas_with_diameter = oas_with_diameter.compute()
oas_with_diameter.set_index("code", inplace=True)
oas_with_diameter.head()
```

```
[23]:
```

	year	lsoa_code	lsoa_name	bng_easting	bng_northing	\
code						
E00000001	2021	E01000001	City of London 001A	532250	181864	
E00000003	2021	E01000001	City of London 001A	532171	181819	
E00000005	2021	E01000001	City of London 001A	532166	181722	
E00000007	2021	E01000001	City of London 001A	532088	181473	
E00000010	2021	E01000003	City of London 001C	532092	182114	

	latitude	longitude	global_id	\
code				
E00000001	51.52022	-0.09523	3a44dd3d-5082-4a09-9b9c-3a5fadc811ed	
E00000003	51.51983	-0.09638	f1216dc8-14d1-4857-9230-cab0641758fb	
E00000005	51.51896	-0.09649	44d6f70f-549c-4288-9b6d-de2adbf02582	
E00000007	51.51674	-0.09771	4dd683e1-9a5c-46cf-9e19-8465c8fbb6cb	
E00000010	51.52250	-0.09741	7476781f-8fe4-4c9b-bde1-0eecbd146dff	

	geometry	\
code		
E00000001	POLYGON ((-0.09450 51.51976, -0.09579 51.52007...	
E00000003	POLYGON ((-0.09579 51.52007, -0.09614 51.51962...	
E00000005	POLYGON ((-0.09630 51.51933, -0.09576 51.51879...	
E00000007	POLYGON ((-0.09603 51.51847, -0.09508 51.51824...	
E00000010	POLYGON ((-0.09691 51.52267, -0.09749 51.52217...	

	1000000m	\
code		
E00000001	POLYGON ((-2.12396 40.51086, -10.39970 52.8942...	
E00000003	POLYGON ((6.53525 62.49735, 8.66637 45.25950, ...	
E00000005	POLYGON ((6.51649 62.50678, 8.98142 45.83038, ...	
E00000007	POLYGON ((1.05215 66.55772, 10.21385 50.24435,...	
E00000010	POLYGON ((2.48881 66.07282, 10.13064 49.54280,...	

	10000m	\
code		
E00000001	POLYGON ((0.00793 51.50076, -0.12655 51.40082,...	
E00000003	POLYGON ((-0.02948 51.61619, -0.00817 51.45339...	
E00000005	POLYGON ((-0.03016 51.61555, -0.00499 51.45853...	

```
E00000007 POLYGON ((-0.08455 51.64281, 0.00789 51.50098,...
E00000010 POLYGON ((-0.07106 51.64383, 0.00536 51.50245,...
```

5000m \

code

```
E00000001 POLYGON ((-0.04329 51.51026, -0.11153 51.46049...
E00000003 POLYGON ((-0.06264 51.56811, -0.05198 51.48672...
E00000005 POLYGON ((-0.06322 51.56742, -0.05037 51.48865...
E00000007 POLYGON ((-0.12461 51.46207, -0.14684 51.54332...
E00000010 POLYGON ((-0.08398 51.58321, -0.04578 51.51256...
```

2000m \

code

```
E00000001 POLYGON ((-0.10032 51.49571, -0.11679 51.52547...
E00000003 POLYGON ((-0.10437 51.49634, -0.11698 51.52656...
E00000005 POLYGON ((-0.08285 51.53835, -0.07868 51.50463...
E00000007 POLYGON ((-0.10911 51.49449, -0.11763 51.52992...
E00000010 POLYGON ((-0.09174 51.54687, -0.07689 51.51621...
```

1000m \

code

```
E00000001 POLYGON ((-0.09812 51.50791, -0.10665 51.52287...
E00000003 POLYGON ((-0.10041 51.50787, -0.10701 51.52312...
E00000005 POLYGON ((-0.08938 51.52864, -0.08722 51.51171...
E00000007 POLYGON ((-0.10402 51.50540, -0.10812 51.52469...
E00000010 POLYGON ((-0.09433 51.53477, -0.08690 51.51944...
```

100m diameter

code

```
E00000001 POLYGON ((-0.09634 51.51901, -0.09715 51.52127... 542.417997
E00000003 POLYGON ((-0.09702 51.51843, -0.09797 51.52025... 349.721440
E00000005 POLYGON ((-0.09561 51.52026, -0.09504 51.51789... 448.004920
E00000007 POLYGON ((-0.09902 51.51545, -0.10023 51.51850... 1512.050021
E00000010 POLYGON ((-0.09666 51.52388, -0.09598 51.52211... 259.493756
```

```
[24]: def calculate_scores(df, name):
    cache_dir = "oa_scores"
    if not os.path.exists(cache_dir):
        os.makedirs(cache_dir)

    cache_file = os.path.join(cache_dir, f"{name}.csv")

    if os.path.exists(cache_file):
        print(f"Loading cached scores for {name} from {cache_file}")
        return pd.read_csv(cache_file).set_index("code")["score"]
```

```

    # Join the DataFrame with the oas_with_diameter DataFrame to include the
    ↪diameter column
    df_with_diameter = df.merge(oas_with_diameter[["diameter"]], how="left",
    ↪left_index=True, right_index=True)

    # calculate the mean area and building levels, but don't include zeros
    df["area"] = pd.to_numeric(df["area"], errors="coerce")

    mean_area = min(df[df["area"] > 0.0]["area"].drop_duplicates("osmid").
    ↪mean().compute(), 1.0)
    df["area"] = df["area"].fillna(mean_area).replace(0.0, mean_area)
    print(f"Mean area: {mean_area}")

    df["building_levels"] = pd.to_numeric(df["building_levels"],
    ↪errors="coerce")

    mean_building_levels = min(df[df["building_levels"] > 0]["building_levels"].
    ↪drop_duplicates("osmid").mean().compute(), 1.0)
    df["building_levels"] = df["building_levels"].fillna(mean_building_levels).
    ↪replace(0, mean_building_levels)
    print(f"Mean building levels: {mean_building_levels}")

    result = df_with_diameter.groupby(df_with_diameter.index).apply(
        lambda df: ((df["area"] * df["building_levels"]) / (df["distance"] +
    ↪df["diameter"]/4)).sum(),
        meta=('score', 'float64')
    ).compute()

    result.to_csv(cache_file, header=True)
    return result

scores = {
    name: calculate_scores(df, name)
    for name, df in features.items()
}

```

Loading cached scores for higher_education from oa_scores\higher_education.csv
 Loading cached scores for other_education from oa_scores\other_education.csv
 Loading cached scores for public_working_space from
 oa_scores\public_working_space.csv
 Loading cached scores for restaurants_and_fast_food from
 oa_scores\restaurants_and_fast_food.csv
 Loading cached scores for entertainment_venues from
 oa_scores\entertainment_venues.csv
 Loading cached scores for gyms_and_sports_centres from
 oa_scores\gyms_and_sports_centres.csv
 Loading cached scores for green_spaces from oa_scores\green_spaces.csv

```

Loading cached scores for bookstores_and_copyshops from
oa_scores\bookstores_and_copyshops.csv
Loading cached scores for public_transport from oa_scores\public_transport.csv
Loading cached scores for cycling_and_walking_infrastructure from
oa_scores\cycling_and_walking_infrastructure.csv
Loading cached scores for healthcare_facilities from
oa_scores\healthcare_facilities.csv
Loading cached scores for student_accommodation from
oa_scores\student_accommodation.csv
Loading cached scores for community_centres_and_youth_clubs from
oa_scores\community_centres_and_youth_clubs.csv
Mean area: 1.0
Mean building levels: nan
Mean area: 1.0
Mean building levels: nan

```

```

[25]: scores_df = pd.DataFrame()
scores_df.index = oas["code"]

for name, score in scores.items():
    scores_df = scores_df.join(score.rename(name), how="left")

scores_df = scores_df.fillna(0.0)
scores_df

```

```

[25]:
higher_education  other_education  public_working_space  \
E00000001        82.195642        83.847498        19.476666
E00000003        73.818493        89.024087        19.969984
E00000005        68.561107       100.726060        19.759186
E00000007        55.925248        34.155105        17.998577
E00000010        93.426792        41.005444        25.272895
...
W00010693         0.000000         0.000000         0.000000
W00010694         0.000000         0.000000         0.000000
W00010695         0.000000         0.000000         0.000000
W00010696         0.000000         0.000000         0.000000
W00010697         0.000000         0.000000         0.121761

restaurants_and_fast_food  entertainment_venues  \
E00000001        61.497269        59.640436
E00000003        59.550329        63.674065
E00000005        59.556343        62.234104
E00000007        47.374385        54.534089
E00000010        70.156278        63.538403
...
W00010693         0.000000         0.000000
W00010694         0.000000         0.000000

```

W00010695	0.000000	0.000000
W00010696	0.000000	0.000000
W00010697	0.400985	0.233411

	gyms_and_sports_centres	green_spaces	bookstores_and_copysshops	\
E00000001	11.955781	0.0	2.447050	
E00000003	11.408710	0.0	2.468116	
E00000005	10.131534	0.0	2.453923	
E00000007	7.104191	0.0	2.341451	
E00000010	13.901484	0.0	2.394846	
...	
W00010693	0.000000	0.0	0.000000	
W00010694	2.385027	0.0	0.000000	
W00010695	0.000000	0.0	0.000000	
W00010696	0.000000	0.0	0.000000	
W00010697	0.000000	0.0	0.000000	

	public_transport	cycling_and_walking_infrastructure	\
E00000001	2.742097	0.0	
E00000003	2.812897	0.0	
E00000005	2.793144	0.0	
E00000007	2.745562	0.0	
E00000010	2.920288	0.0	
...	
W00010693	0.000000	0.0	
W00010694	0.000000	0.0	
W00010695	0.000000	0.0	
W00010696	0.000000	0.0	
W00010697	0.000000	0.0	

	healthcare_facilities	student_accommodation	\
E00000001	7.271802	114.472598	
E00000003	7.554300	109.644535	
E00000005	7.813229	107.537653	
E00000007	9.155378	103.960092	
E00000010	6.488253	108.499954	
...	
W00010693	0.000000	0.000000	
W00010694	0.000000	0.000000	
W00010695	0.000000	0.000000	
W00010696	0.000000	0.000000	
W00010697	1.428474	0.000000	

	community_centres_and_youth_clubs	beaches	rivers
E00000001	2.582780	0.0	0.0
E00000003	2.751345	0.0	0.0
E00000005	2.771769	0.0	0.0

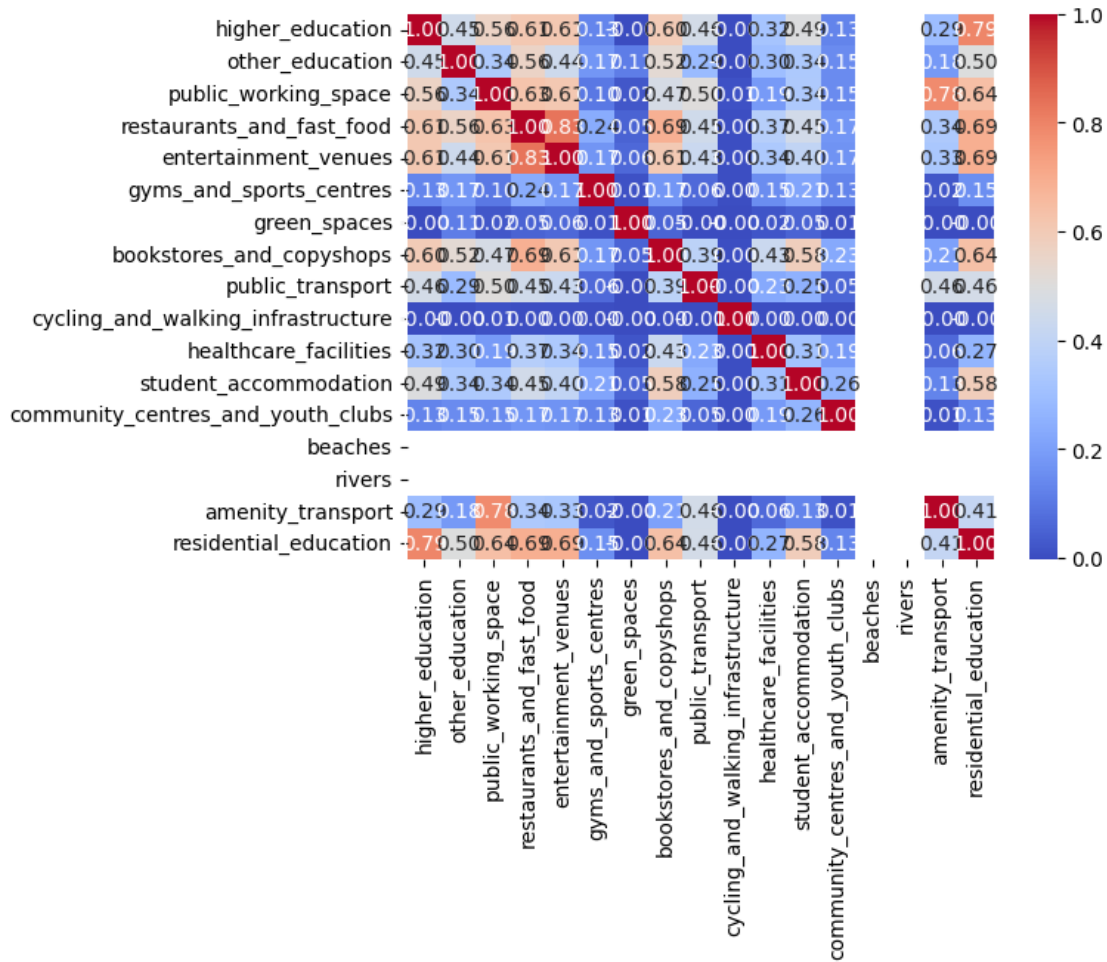
E00000007	2.854118	0.0	0.0
E00000010	3.302692	0.0	0.0
...
W00010693	0.000000	0.0	0.0
W00010694	0.000000	0.0	0.0
W00010695	0.000000	0.0	0.0
W00010696	0.000000	0.0	0.0
W00010697	2.709583	0.0	0.0

[188880 rows x 15 columns]

```
[26]: # Add interaction terms
scores_df['amenity_transport'] = scores_df['public_transport'] *
    ↪ scores_df['public_working_space']
scores_df['residential_education'] = scores_df['student_accommodation'] *
    ↪ scores_df['higher_education']
```

```
[27]: sns.heatmap(scores_df.corr(), annot=True, fmt=".2f", cmap="coolwarm")
```

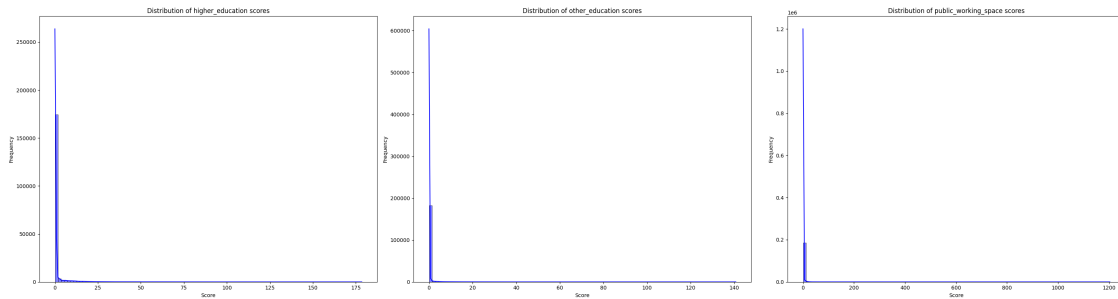
```
[27]: <Axes: >
```



```
[28]: fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(30, 8))

for ax, column in zip(axes, scores_df.columns):
    sns.histplot(scores_df[column], bins=100, kde=True, color='blue',
    label=column, alpha=0.5, ax=ax)
    ax.set_title(f"Distribution of {column} scores")
    ax.set_xlabel("Score")
    ax.set_ylabel("Frequency")

plt.tight_layout()
plt.show()
```



```
[29]: def plot_transformation(fn):
    scores_transformed = scores_df.apply(fn)

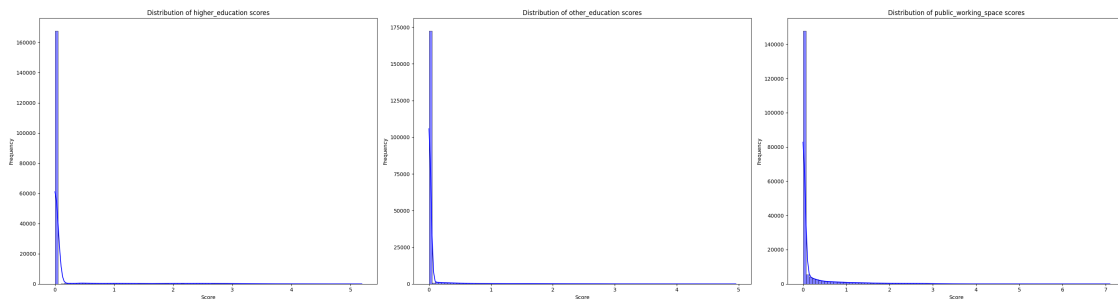
    # plot the first 5 transformed scores
    fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(30, 8))

    for ax, column in zip(axes, scores_transformed.columns):
        sns.histplot(scores_transformed[column], bins=100, kde=True,
        color='blue', label=column, alpha=0.5, ax=ax)
        ax.set_title(f"Distribution of {column} scores")
        ax.set_xlabel("Score")
        ax.set_ylabel("Frequency")

    plt.tight_layout()
    plt.show()

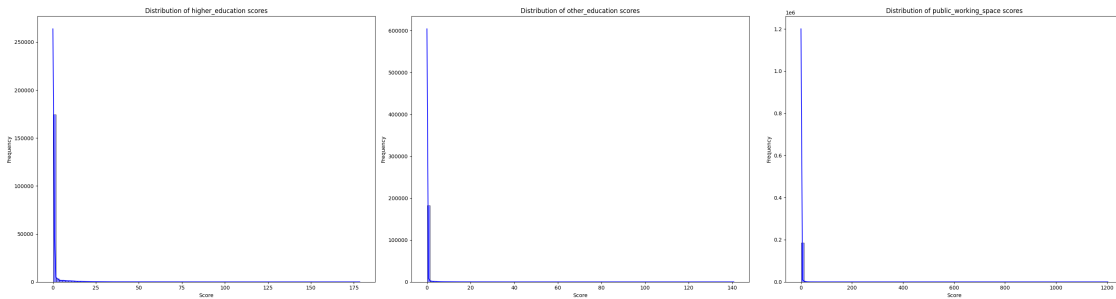
    return scores_transformed

scores_log1p = plot_transformation(np.log1p)
```



```
[30]: from scipy.stats import boxcox

scores_boxcox = plot_transformation(lambda x: boxcox(x + 1)[0] if (x > 0).all()
else x)
```



```
[31]: scores_with_geometry = oas.set_index("code")["geometry"].join(scores_log1p,
    ↪how="left").compute()
to_plot = scores_with_geometry
```

```
[32]: # cluster the output areas based on the scores, and then plot
from sklearn.cluster import KMeans
from sklearn.preprocessing import RobustScaler

numeric = scores_with_geometry.select_dtypes(include=[np.number, np.float64, np.
    ↪int64])

scaler = RobustScaler()
numeric = scaler.fit_transform(numeric)

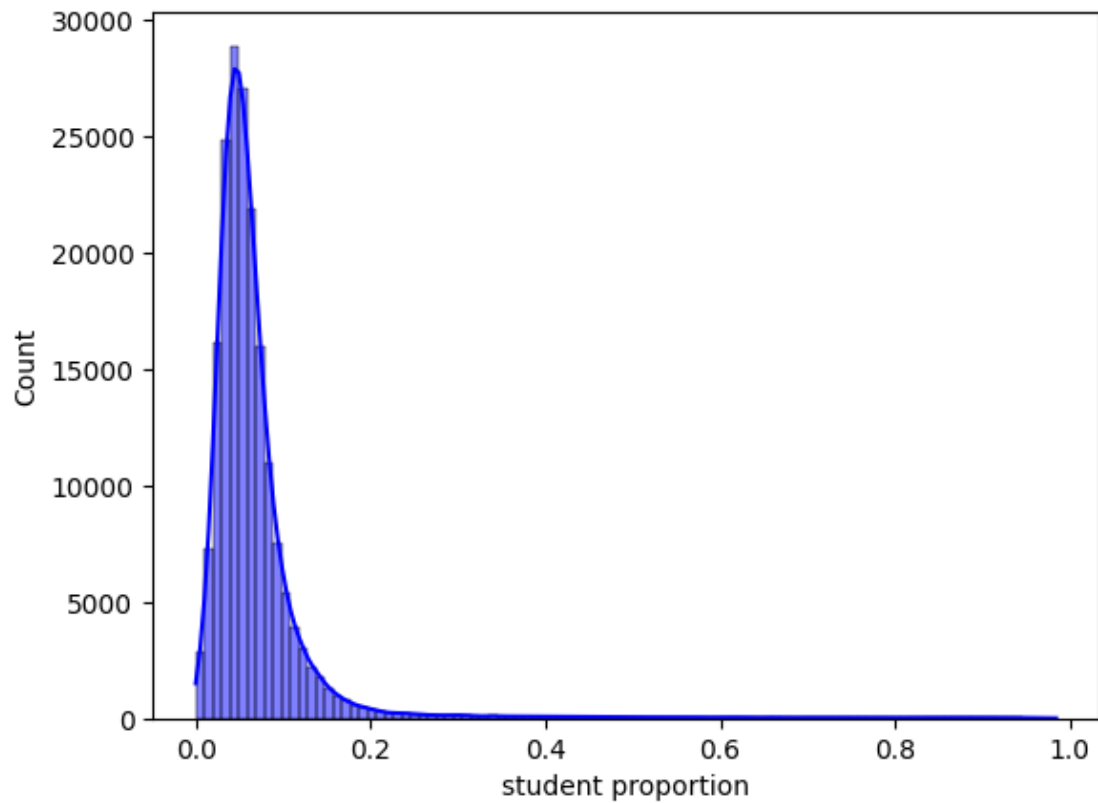
kmeans = KMeans(n_clusters=5, random_state=0).fit(numeric)
to_plot["cluster"] = kmeans.labels_
```

```
[33]: Y = pd.read_sql("SELECT full_time_students, total_residents_16_and_over,
    ↪output_area FROM census_nssec", conn).set_index("output_area")
Y["student proportion"] = Y["full_time_students"] /
    ↪Y["total_residents_16_and_over"]
Y = Y['student proportion']
# plot the distribution of student proportions
sns.histplot(Y, bins=100, kde=True, color='blue', label='Student Proportion',
    ↪alpha=0.5)
```

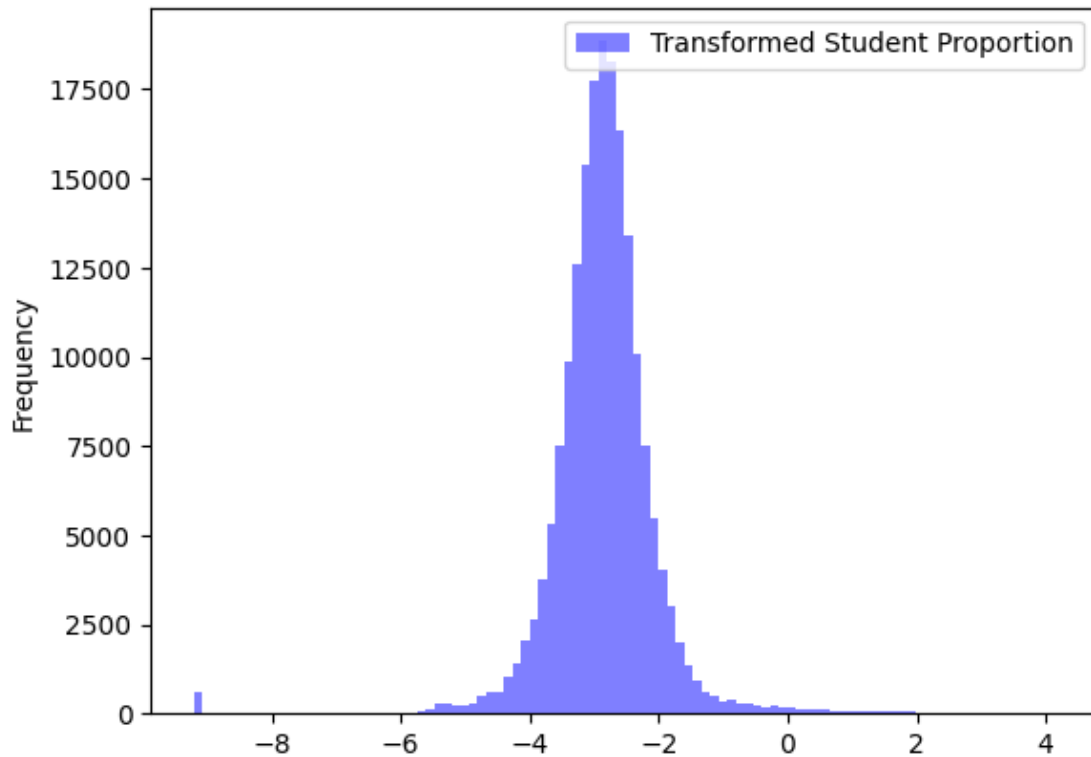
C:\Users\varun\AppData\Local\Temp\ipykernel_28252\1183412520.py:1: UserWarning:
pandas only supports SQLAlchemy connectable (engine/connection) or database
string URI or sqlite3 DBAPI2 connection. Other DBAPI2 objects are not tested.
Please consider using SQLAlchemy.

```
Y = pd.read_sql("SELECT full_time_students, total_residents_16_and_over,
output_area FROM census_nssec", conn).set_index("output_area")
```

```
[33]: <Axes: xlabel='student proportion', ylabel='Count'>
```



```
[34]: # as it's a proportion, we can use the logit transformation
clip = 1e-4
clipped = Y.clip(clip, 1 - clip)
Y_logit = np.log(clipped / (1 - clipped))
Y_logit.plot(kind="hist", bins=100, color='blue', label='Transformed Student_␣
↳Proportion', alpha=0.5)
plt.legend()
plt.show()
```



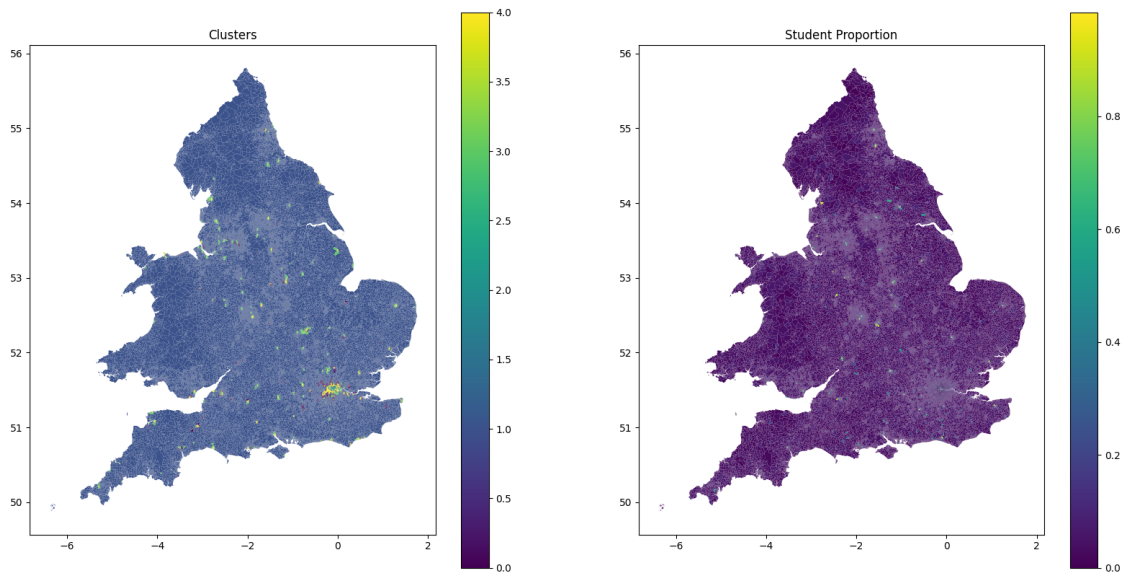
```
[35]: to_plot = to_plot.join(Y)
```

```
[36]: # Plot the clusters side-by-side with the student proportion
fig, axes = plt.subplots(1, 2, figsize=(20, 10))

to_plot.plot(column="cluster", cmap="viridis", legend=True, ax=axes[0])
to_plot.plot(column="student proportion", cmap="viridis", legend=True,
              ↪ax=axes[1])

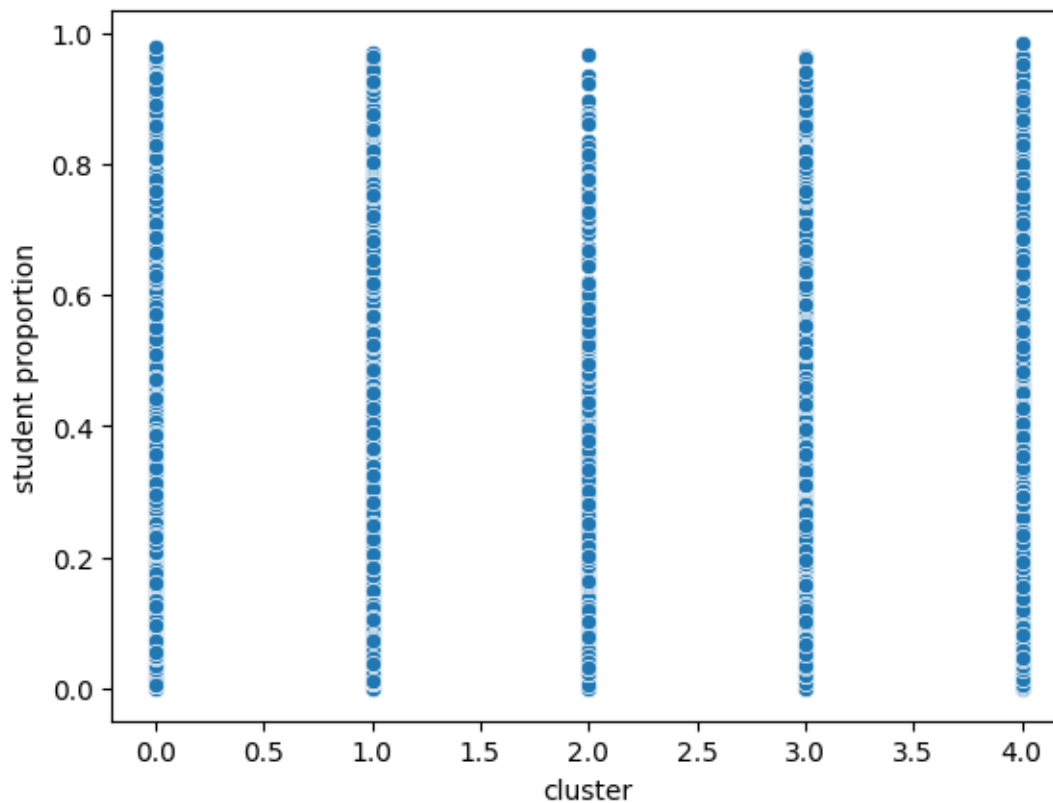
axes[0].set_title("Clusters")
axes[1].set_title("Student Proportion")

plt.show()
```



```
[37]: # plot cluster against student proportion
sns.scatterplot(data=to_plot, x="cluster", y="student proportion")
```

```
[37]: <Axes: xlabel='cluster', ylabel='student proportion'>
```



```
[38]: from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

# Prepare the data
X = scores_log1p

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, Y_logit, test_size=0.1,
    ↪ random_state=42)

# Initialize the linear regression model
model = LinearRegression()

# Train the model
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R^2 Score: {r2}")

# Plot the predicted vs actual values
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, alpha=0.3)
plt.plot([Y_logit.min(), Y_logit.max()], [Y_logit.min(), Y_logit.max()], 'r--',
    ↪ lw=2)
plt.xlabel("Actual Logit Transformed Student Proportion")
plt.ylabel("Predicted Logit Transformed Student Proportion")
plt.title("Actual vs Predicted Logit Transformed Student Proportion")
plt.show()
```

Mean Squared Error: 0.5855899703223785

R^2 Score: 0.11539999543534618

