task1

December 19, 2024

[2]:
```
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
import statsmodels.api as sm

# 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.
```

[3]:
```
# 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"),

    # UK Shapefile
    ("uk_shp.zip", "https://www.eea.europa.eu/data-and-maps/data/
 ↪eea-reference-grids-2/gis-files/great-britain-shapefile/at_download/file"),
]:

    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
Already downloaded uk_shp.zip
```

[4]:
```
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()
```

[5]:
```
%%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.
```

[5]:
```
[]
```

[6]:
```
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_22364\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:
```

[7]:
```
%%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.
```

[7]:
```
[]
```

[8]:
```
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_22364\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:
```

[9]:
```
%%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.
```

[9]:
```
[]
```

[10]:
```
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_22364\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:
```

[11]:
```
%%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.
```

[11]:
```
[]
```

[12]:
```
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)?$'
        feet_inches_pattern = r"^(\d+)'\s*(\d+)?\"?$"

        # 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)
```

[13]:
```
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)
```

[14]:
```
# 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_22364\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:
```

[15]:
```
# 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)
```

[16]:
```
%%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)
```

0.0.1 Reasoning for Using Dask GeoPandas Instead of MySQL Spatial Joins

1. **Scalability**: Dask GeoPandas allows for parallel processing and can handle larger-than-memory datasets by distributing the computation across multiple cores or even a cluster of machines. This is particularly useful when dealing with large geospatial datasets, which can be computationally intensive.
2. **Performance**: Dask GeoPandas can perform spatial operations more efficiently by leveraging Dask's parallel computing capabilities. This can result in significant performance improvements compared to MySQL spatial joins, which may not be as optimized for parallel processing.
3. **Flexibility**: Dask GeoPandas provides a more flexible and Pythonic interface for performing geospatial operations. It integrates seamlessly with other Python libraries such as Pandas, GeoPandas, and Dask, allowing for more complex and customized workflows.
4. **Memory Management**: Dask GeoPandas can handle out-of-core computations, meaning it can process data that does not fit into memory by breaking it into smaller chunks and processing them sequentially. This is particularly advantageous when working with large geospatial datasets that exceed the available memory.

[17]:
```
available_parallelism = mp.cpu_count()
```

[18]:
```
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
```

0.0.2 Reading PostGIS Data

I encountered difficulties getting the PostGIS reading functionality to work properly. As a workaround, I opted to deserialize WKT (Well-Known Text) representations of geometries instead. This approach involved extracting the WKT strings from the database and converting them into geometric objects using the `shapely` library. This method proved to be a reliable alternative, allowing me to proceed with the geospatial analysis without further issues.

[19]:
```
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)
oas["code"] = oas["code"].astype(str)
```

```
C:\Users\varun\AppData\Local\Temp\ipykernel_22364\2804205704.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))
```

1 Assess

The clear first potential feature to look when trying to predict student proportion in each output area is universities.

[20]:
```
university_buildings = load_geometry_from_wkt(pd.read_sql(r"SELECT *,␣
 ↪ST_AsText(geometry) as wkt from osm_features where amenity like␣
 ↪'%university%'", conn))
university_buildings.explore()
```

```
C:\Users\varun\AppData\Local\Temp\ipykernel_22364\98538524.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.
  university_buildings = load_geometry_from_wkt(pd.read_sql(r"SELECT *,
ST_AsText(geometry) as wkt from osm_features where amenity like '%university%'",
conn))
```

[20]:
```
<folium.folium.Map at 0x1a571bdad50>
```

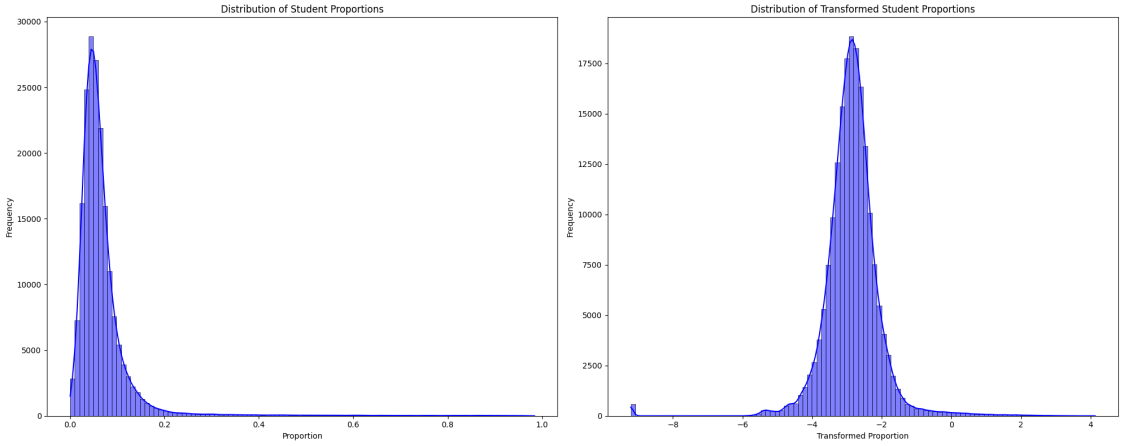
1.0.1 University Features

The university features in the dataset exhibit significant variability in size and scope, encompassing a wide range of entities from individual buildings to entire campuses. This diversity is evident in the representation of features such as West Cambridge, which is encapsulated as a single feature despite its extensive area and numerous facilities. This variability poses challenges in spatial analysis, as the granularity of the data can significantly impact the interpretation of spatial relationships and the accuracy of derived metrics. Consequently, it is crucial to account for these differences when analyzing and visualizing university-related features to ensure meaningful and accurate insights.

Lets compare it to student proportion within each output area

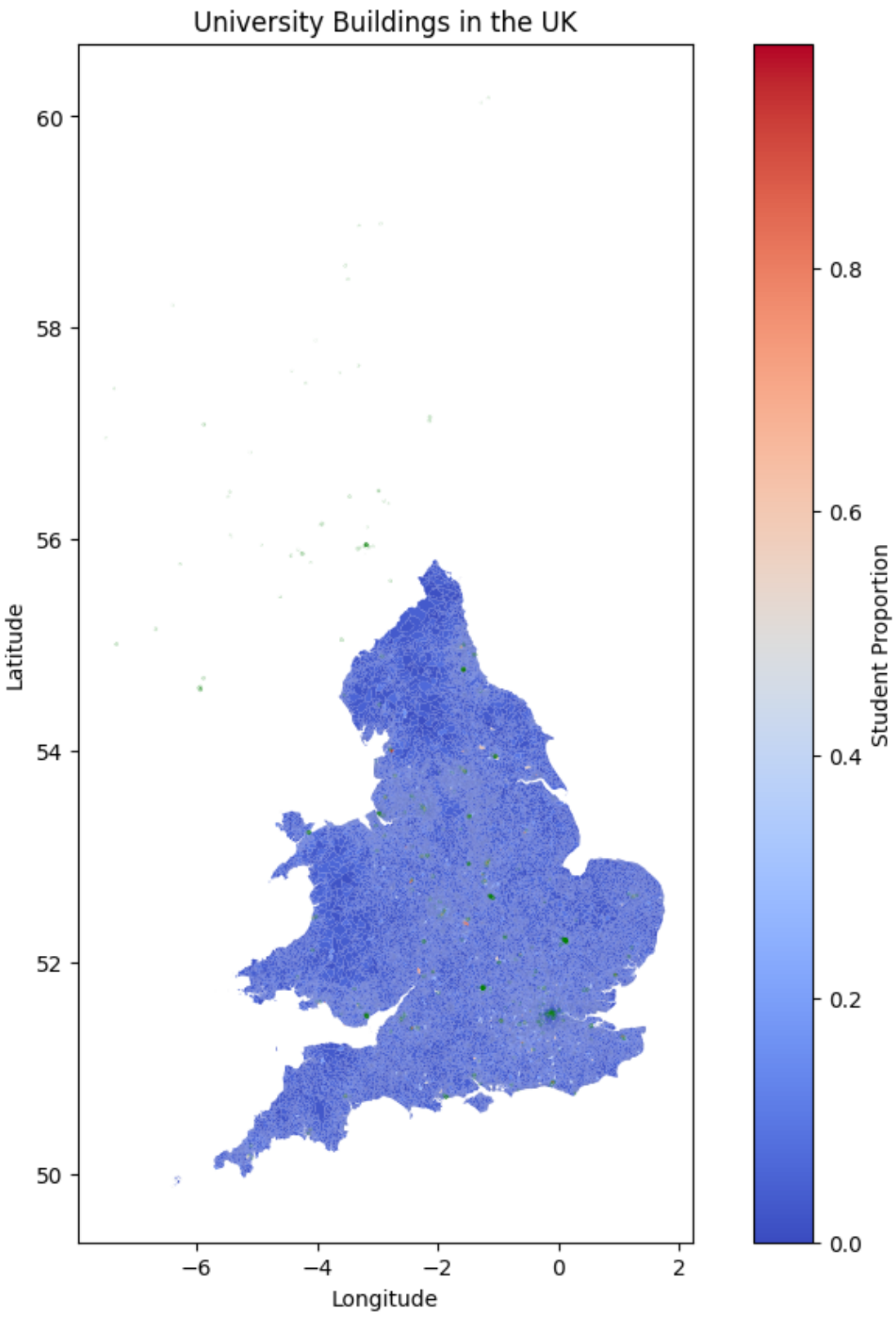
[21]:
```
Y = pd.read_sql("SELECT full_time_students, total_residents_16_and_over,␣
 ↪output_area FROM census_nssec", conn).set_index("output_area")
Y["student proportion"] = Y["full_time_students"] /␣
 ↪Y["total_residents_16_and_over"]
Y = Y['student proportion']

# Create subplots
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(20, 8))

# Plot the distribution of student proportions
sns.histplot(Y, bins=100, kde=True, color='blue', label='Student Proportion',␣
 ↪alpha=0.5, ax=axes[0])
axes[0].set_title("Distribution of Student Proportions")
axes[0].set_xlabel("Proportion")
axes[0].set_ylabel("Frequency")

# 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))
sns.histplot(Y_logit, bins=100, kde=True, color='blue', label='Transformed␣
 ↪Student Proportion', alpha=0.5, ax=axes[1])
axes[1].set_title("Distribution of Transformed Student Proportions")
axes[1].set_xlabel("Transformed Proportion")
axes[1].set_ylabel("Frequency")

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

```
C:\Users\varun\AppData\Local\Temp\ipykernel_22364\2805985166.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")
```



[22]:
```
oas_with_student_proportion = oas.set_geometry("geometry")[["code",␣
 ↪"geometry"]].compute()
oas_with_student_proportion = oas_with_student_proportion.merge(Y,␣
 ↪left_on="code", right_index=True, how="left")
```

[23]:
```
fig, ax = plt.subplots(1, 1, figsize=(10, 10))

oas_with_student_proportion.plot(column="student proportion", ax=ax,␣
 ↪legend=True, cmap="coolwarm", legend_kwds={"label": "Student Proportion"})

# Plot the university buildings
university_buildings.plot(ax=ax, color='green', markersize=5, alpha=0.2)

plt.title('University Buildings in the UK')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```


University Buildings in the UK
Latitude
60
58
56
54
52
50
−6
−4
−2
0
2
Longitude
Student Proportion
0.8
0.6
0.4
0.2
0.0

To perform a linear regression analysis to understand the impact of university buildings on various output areas, it is essential to calculate a score for each output area based on the proximity and characteristics of university buildings. This score will serve as a quantitative measure of the influence of university buildings on each output area. To achieve this, we need to determine the distance relationships between every university building feature and each output area. By calculating these distances, we can assess how the presence and proximity of university buildings affect the output areas. This involves spatially joining the university building features with the output areas and computing the distances between them. The resulting scores will then be used as input variables in the linear regression model, allowing us to analyze the relationship between university buildings and various socio-economic factors within the output areas.

1.0.2 Importance of Caching Spatial Join Results for Efficiency

Caching the results of the spatial join was vital for efficiency in this analysis due to the computationally intensive nature of spatial operations. Spatial joins, especially when dealing with large geospatial datasets, can be time-consuming and resource-intensive. By caching the results, we avoided redundant computations, significantly reducing the processing time for subsequent operations. This approach allowed us to quickly retrieve previously computed results, facilitating faster iterations and enabling more efficient data exploration and analysis. Additionally, caching helped in managing memory usage more effectively, as it prevented the need to repeatedly load and process large datasets, thereby optimizing the overall performance of the workflow.

[24]:
```
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)
```

1.0.3 Varying Search Distances Based on Feature Density

I chose to vary the distance at which I looked for features from each output area depending on the total number of features found for that query to balance computational efficiency and spatial relevance. By adjusting the search radius based on feature density, I ensured that areas with sparse features were examined over larger distances to capture meaningful data, while densely populated areas were restricted to smaller radii to maintain precision and relevance. The cutoffs were determined experimentally by iteratively testing different radius values and observing the distribution of feature counts. This process involved analyzing the spatial distribution of features and adjusting the radii to achieve a balance where the majority of output areas had a sufficient number of features for meaningful analysis without overwhelming the computational resources. The final cutoffs were set at 100, 1000, 2000, 5000, 10000, and 1000000 meters, ensuring a scalable and efficient approach to feature collection.

[25]:
```
import hashlib

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

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

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)
        )
```

```
        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")

university_building_distances = collect_features_for_condition("amenity like␣
↪'%university%'", "university")
```

```
Already processed amenity like '%university%', loading from file…
```

The formula used to calculate the relative density for each feature is:

$$\text{score} = \sum \left(\frac{\text{area}}{\text{distance} + \frac{\text{diameter}}{4}} \right)$$

1.0.4 Design Justification:

- **Area and Building Levels**: Using the product of the area and building levels would represent the volumetric presence of a feature but this wouldn't work great for predicting a proportion. For example, if 20% of buildings in an area are 10 stories tall but so are all the other buildings, the proportion would still be 20% despite a much higher score than if all the buildings were 1 story tall.
- **Distance and Diameter**: The denominator incorporates the distance to the feature and a fraction of the feature's diameter, ensuring that larger features have a proportionally reduced impact of distance. This helps in balancing the influence of proximity and feature size. Using diameter/4 is to approximate the distance from the center of an output area to the edge of the feature but ensures that features near the center of the output area don't disproportionately affect the score.
- **Summation**: By summing the contributions of all features, the formula aggregates the relative densities, providing a comprehensive measure of feature density within the specified context.

1.0.5 Tradeoffs:

- **Computational Complexity**: The formula involves multiple operations (multiplication, division, and summation) for each feature, which can be computationally intensive for large datasets.
- **Area may be zero**: If the area of a feature is zero, the score for that feature will be zero, which may not accurately reflect the density of the feature. For this, I just use the median area of the target features to replace the zero area.

[26]:
```
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)
```

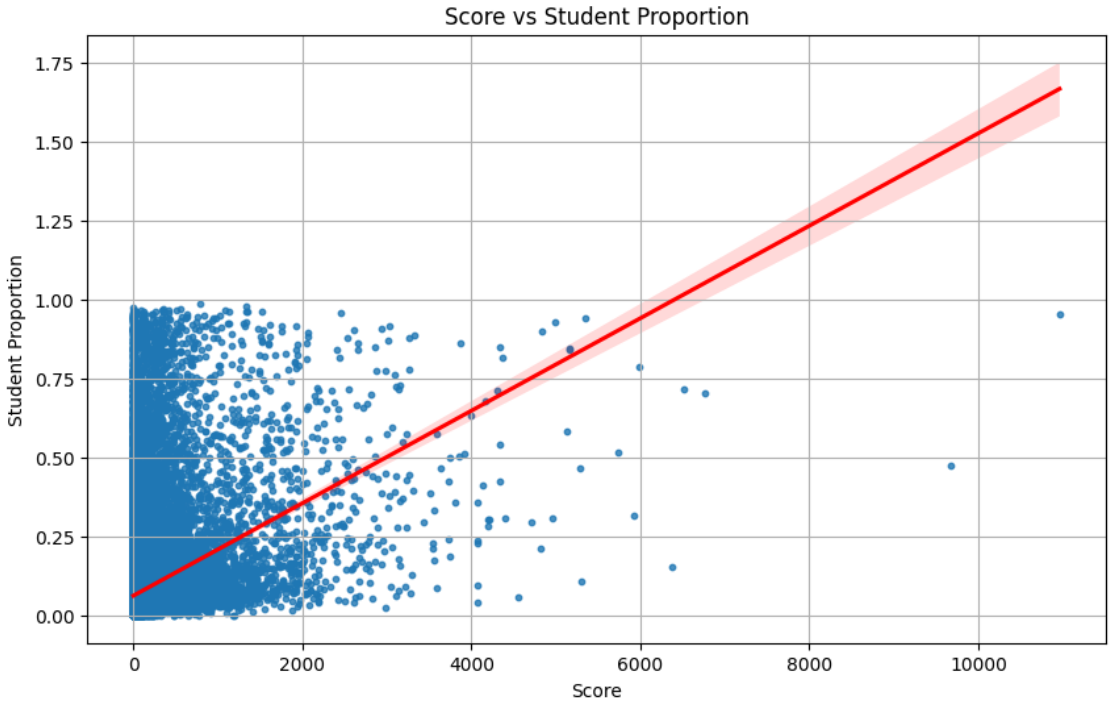
[27]:
```
def calculate_scores(df, name):
    cache_dir = "oa_scores"
    if not os.path.exists(cache_dir):
        os.makedirs(cache_dir)

    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 but don't include zeros
    df["area"] = pd.to_numeric(df["area"], errors="coerce")

    # media approximate instead for speeeeeeeed
    mean_area = min(df[df["area"] > 0.0].drop_duplicates("osmid")["area"].
 ↪compute().median(), 1.0)
    df["area"] = df["area"].fillna(mean_area).replace(0.0, mean_area)
    print(f"Median area: {mean_area}")

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

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

university_building_scores = calculate_scores(university_building_distances,␣
 ↪"university")
```

```
Loading cached scores for university from oa_scores\university.csv
```

[28]:
```
university_building_scores_df = university_building_scores.rename("score").
 ↪to_frame()
plottable = oas.merge(university_building_scores_df, how="left",␣
 ↪left_on="code", right_index=True).fillna(0).compute()
```

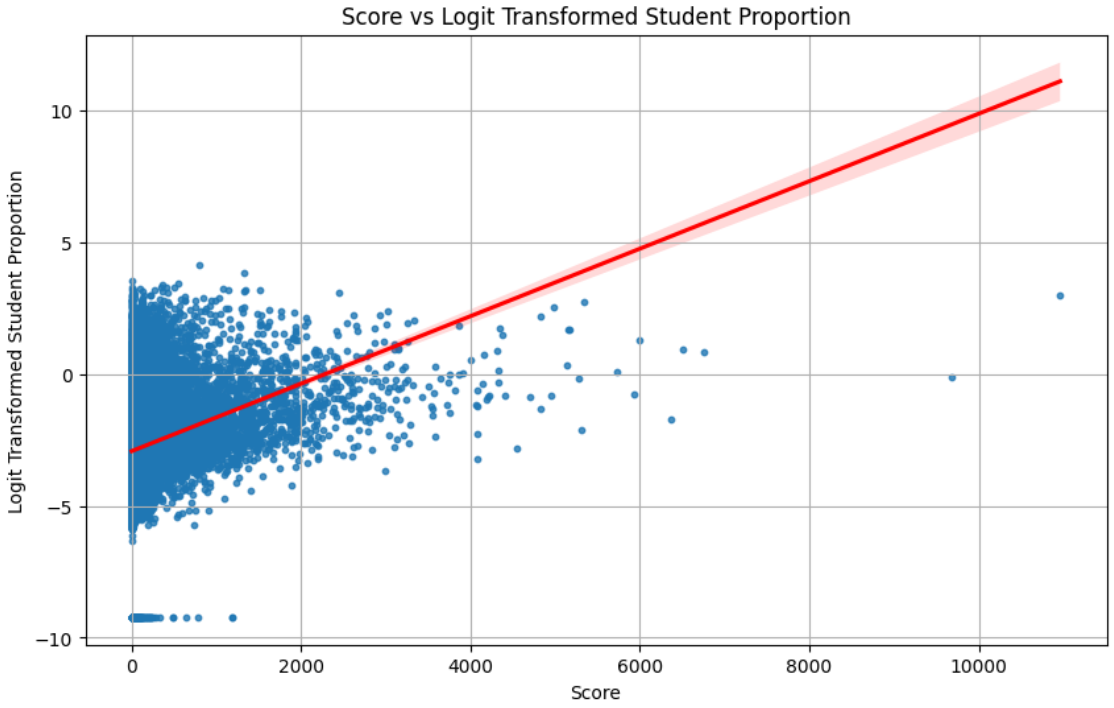
[29]:
```
plt.figure(figsize=(10, 6))
sns.regplot(x=plottable["score"], y=Y, scatter_kws={'s': 10}, line_kws={'color':
 ↪ 'red'})
```

```
plt.xlabel("Score")
plt.ylabel("Student Proportion")
plt.title("Score vs Student Proportion")
plt.grid(True)
plt.show()
```



Interestingly, the initial analysis reveals that there is a small correlation between the calculated scores for university buildings and the student proportion within each output area. This suggests that while the presence of university buildings does have some predictive power regarding student population density, it may not be the sole or most significant factor. To gain a deeper understanding, we will visualize the relationship using the logit-transformed student proportion. This transformation can help to stabilize variance and make the relationship more linear, potentially revealing patterns that were not apparent in the original scale.

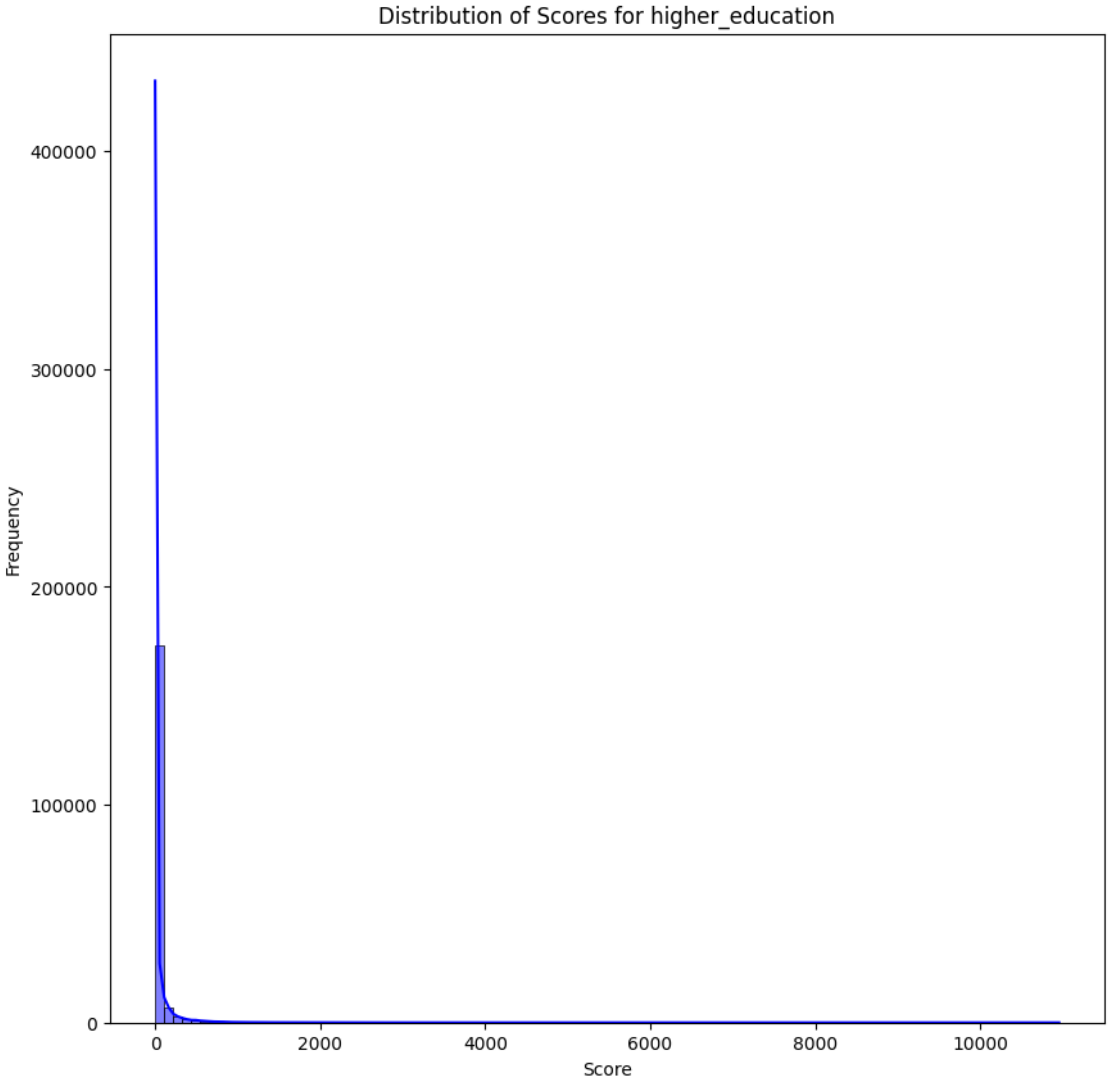
[30]:
```
plt.figure(figsize=(10, 6))
sns.regplot(x=plottable["score"], y=Y_logit, scatter_kws={'s': 10},␣
 ↪line_kws={'color': 'red'})
plt.xlabel("Score")
plt.ylabel("Logit Transformed Student Proportion")
plt.title("Score vs Logit Transformed Student Proportion")
plt.grid(True)
plt.show()
```

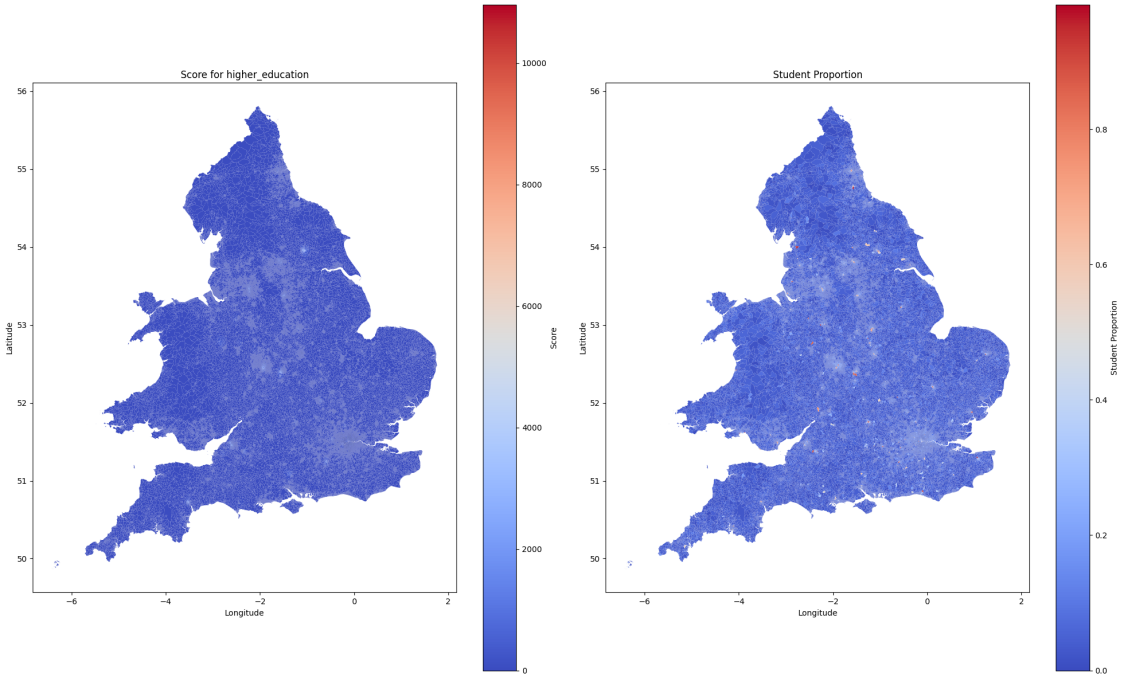


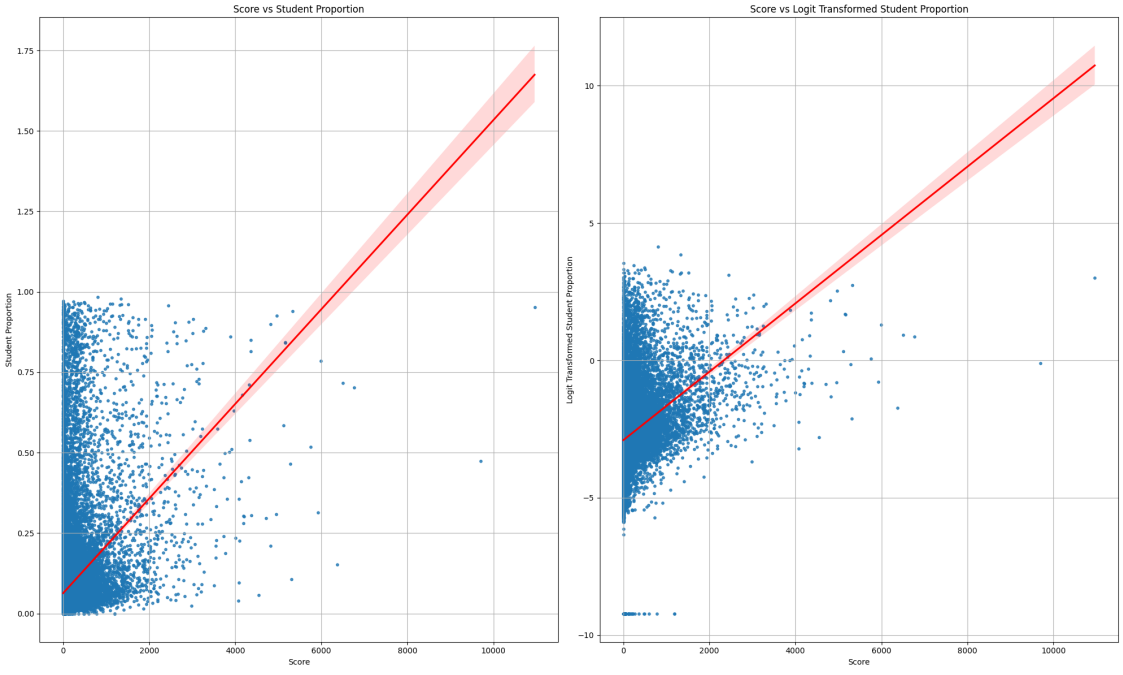
[31]:
```
from functools import lru_cache

@lru_cache(maxsize=None) # for quick iteration when building the graphs
def get_plottable(condition, name):
    features = collect_features_for_condition(condition, name)
    scores = calculate_scores(features, name)
    scores = scores.rename("score").to_frame()
    return oas.merge(scores, how="left", left_on="code", right_index=True).
 ↪fillna(0).compute()

def analyse_feature(condition, name):
    plottable = get_plottable(condition, name)

    fig, ax = plt.subplots(1, 1, figsize=(10, 10))

    # plot a histogram of the scores
    sns.histplot(plottable["score"], bins=100, kde=True, color='blue',␣
 ↪label='Score', alpha=0.5, ax=ax)
    ax.set_title(f"Distribution of Scores for {name}")
    ax.set_xlabel("Score")
    ax.set_ylabel("Frequency")
    plt.show()

    # plot the scores on a map side by side with the student proportion
```

```
    fig, axes = plt.subplots(1, 2, figsize=(20, 12))

    plottable.plot(column="score", ax=axes[0], legend=True, cmap="coolwarm",␣
 ↪legend_kwds={"label": "Score"})
    axes[0].set_title(f"Score for {name}")
    axes[0].set_xlabel("Longitude")
    axes[0].set_ylabel("Latitude")

    oas_with_student_proportion.plot(column="student proportion", ax=axes[1],␣
 ↪legend=True, cmap="coolwarm", legend_kwds={"label": "Student Proportion"})
    axes[1].set_title("Student Proportion")
    axes[1].set_xlabel("Longitude")
    axes[1].set_ylabel("Latitude")

    plt.tight_layout()
    plt.show()

    # plot the two scatter plots
    fig, axes = plt.subplots(1, 2, figsize=(20, 12))

    sns.regplot(x=plottable["score"], y=Y, scatter_kws={'s': 10},␣
 ↪line_kws={'color': 'red'}, ax=axes[0])
    axes[0].set_xlabel("Score")
    axes[0].set_ylabel("Student Proportion")
    axes[0].set_title("Score vs Student Proportion")
    axes[0].grid(True)

    sns.regplot(x=plottable["score"], y=Y_logit, scatter_kws={'s': 10},␣
 ↪line_kws={'color': 'red'}, ax=axes[1])
    axes[1].set_xlabel("Score")
    axes[1].set_ylabel("Logit Transformed Student Proportion")
    axes[1].set_title("Score vs Logit Transformed Student Proportion")
    axes[1].grid(True)

    plt.tight_layout()
    plt.show()

analyse_feature(r"amenity like '%university%' or amenity like␣
 ↪'%research_institute%' or amenity like '%college%'", "higher_education")
```

```
Already processed amenity like '%university%' or amenity like
'%research_institute%' or amenity like '%college%', loading from file…
Loading cached scores for higher_education from oa_scores\higher_education.csv
```


Distribution of Scores for higher_education
Frequency
400000
300000
200000
100000
0
0
2000
4000
6000
8000
10000
Score





1.0.6 Analysis of Student Populations in Relation to University Proximity

The analysis reveals that there are clearly areas with high student populations that are not in close proximity to universities. This observation suggests that factors other than the immediate presence

of university buildings significantly influence student residency patterns. These areas may include neighborhoods with affordable housing, good public transportation links, or amenities that cater to student needs, such as libraries, cafes, and recreational facilities. Additionally, some students may prefer to live in quieter residential areas away from the hustle and bustle of university campuses. This highlights the importance of considering a broader range of socio-economic and infrastructural factors when analyzing student population distributions.

The analysis of various features in relation to student populations reveals a multifaceted landscape where multiple factors contribute to the distribution and density of students within different output areas. Higher education institutions, such as universities and colleges, naturally attract a significant number of students, making them a primary feature of interest. However, other amenities like public working spaces (libraries, cafes, coworking spaces), entertainment venues (cinemas, theatres, bars, pubs), and student accommodations also play crucial roles in shaping student residency patterns. These amenities provide essential services and social opportunities that cater to student lifestyles, making certain areas more attractive for student living. Additionally, public transport infrastructure is vital for students who commute to their educational institutions, further influencing their choice of residence.

Exploring other geographical features such as rivers or beaches can also be valuable. These natural features often enhance the quality of life and recreational opportunities available in an area, potentially making them more appealing to students. For instance, areas near rivers may offer scenic views, outdoor activities, and a tranquil environment, while coastal regions with beaches can provide leisure and relaxation options. Understanding the impact of these natural features can offer a more comprehensive view of the factors that influence student populations, helping to identify areas that combine both educational and lifestyle benefits for students.

[32]:
```
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_copyshops": 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…
Already processed leisure like '%beach%', loading from file…
Already processed waterway like '%river%', loading from file…
```

[33]:
```
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
Loading cached scores for beaches from oa_scores\beaches.csv
Loading cached scores for rivers from oa_scores\rivers.csv
```

[34]:
```
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
```

[34]:

	higher_education	other_education	public_working_space	\
E00000001	228.704393	19.102732	11.212272	
E00000003	235.016824	22.291268	12.260396	
E00000005	234.351321	24.470269	12.111465	
E00000007	222.115700	10.955936	10.572545	
E00000010	247.666206	11.190529	12.820952	
…	…	…	…	
W00010693	0.513706	0.000000	0.500777	
W00010694	0.000000	1.161160	1.506790	
W00010695	0.000000	0.000000	0.722091	
W00010696	0.000000	0.036956	0.543361	
W00010697	0.000000	0.030031	0.131369	

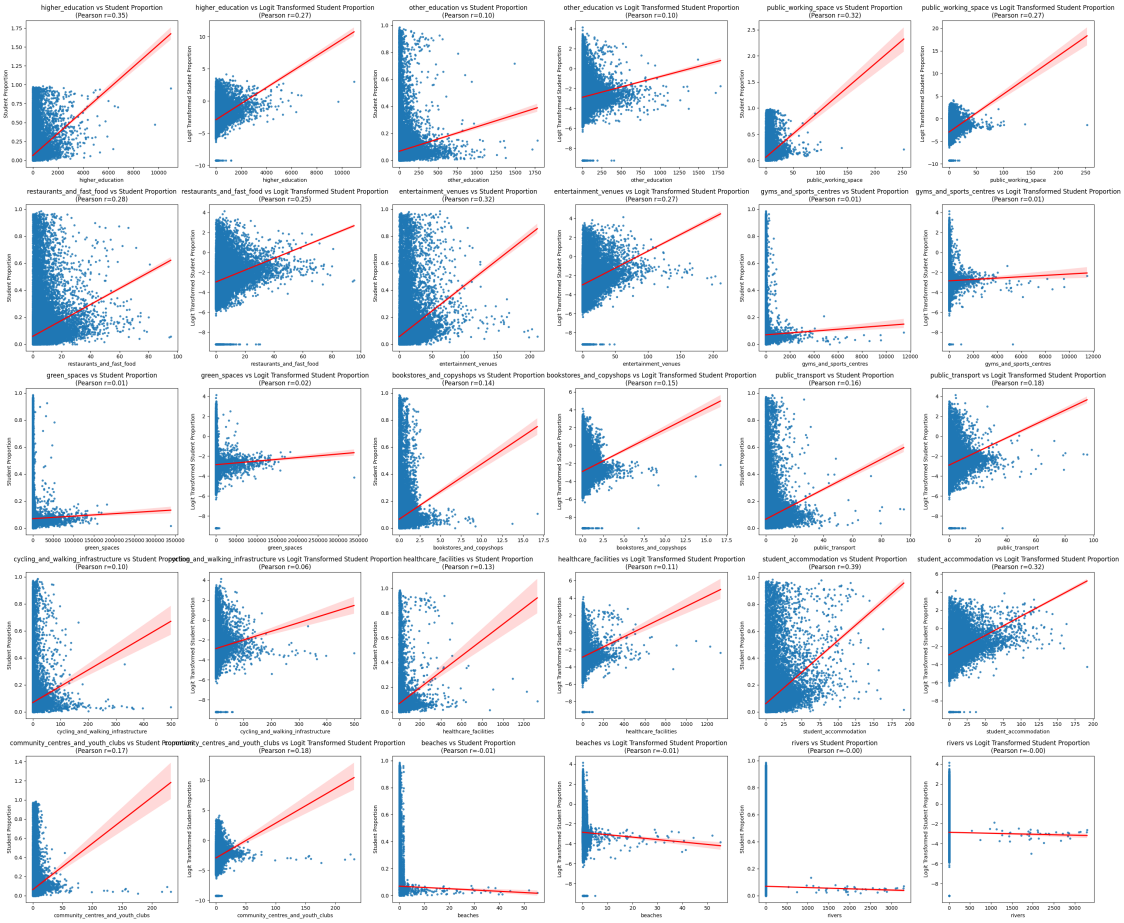
	restaurants_and_fast_food	entertainment_venues	\
E00000001	25.446743	33.383041	
E00000003	25.113051	34.245359	
E00000005	27.224627	34.765657	
E00000007	22.780747	33.775396	
E00000010	26.987618	33.180614	
…	…	…	
W00010693	0.765684	2.021278	
W00010694	0.665379	1.362841	
W00010695	2.108880	0.885610	
W00010696	1.481928	2.086869	
W00010697	3.288268	5.367231	

	gyms_and_sports_centres	green_spaces	bookstores_and_copyshops	\
E00000001	13.628192	36.279164	0.875279	
E00000003	14.289565	35.045224	0.870312	
E00000005	12.202488	33.945269	0.859745	
E00000007	7.496952	29.509889	0.806164	
E00000010	26.513742	45.001193	0.858849	
…	…	…	…	
W00010693	1.363833	36.747104	0.000000	
W00010694	8.121855	16.289179	0.000000	
W00010695	3.138311	0.000000	0.000000	
W00010696	1.184433	36.084193	0.291997	
W00010697	4.708822	94.027384	0.098656	

```
           public_transport  cycling_and_walking_infrastructure  \
E00000001          2.834653                          126.418510
E00000003          2.898433                          106.146306
E00000005          2.920085                           63.794949
E00000007          3.509561                           25.816608
E00000010          2.810414                           32.160475
…                     …                                  …
W00010693          0.000000                            0.000000
W00010694          0.000000                            0.000000
W00010695          0.000000                            0.000000
W00010696          0.669319                            0.000000
W00010697          0.992231                            0.000000

           healthcare_facilities  student_accommodation  \
E00000001               4.394881              26.978518
E00000003               4.406836              27.282454
E00000005               4.269479              26.664052
E00000007               6.019433              23.833159
E00000010               4.066524              33.605559
…                          …                     …
W00010693               0.315119               0.000000
W00010694               0.994017               0.000000
W00010695               0.100821               0.000000
W00010696               1.781111               0.000000
W00010697               2.242732               0.000000

           community_centres_and_youth_clubs   beaches  rivers
E00000001                          3.821318  0.203664     0.0
E00000003                          4.122265  0.203852     0.0
E00000005                          3.495183  0.203902     0.0
E00000007                          2.874737  0.203944     0.0
E00000010                          5.737512  0.203762     0.0
…                                     …       …     …
W00010693                          0.975861  0.072945     0.0
W00010694                          0.269510  0.080191     0.0
W00010695                          0.000000  0.086650     0.0
W00010696                          0.056849  0.085991     0.0
W00010697                          5.294341  0.084998     0.0

[188880 rows x 15 columns]
```

[35]:
```
fig, axes = plt.subplots(nrows=5, ncols=6, figsize=(30, 25))

# Flatten the axes array for easy iteration
axes = axes.flatten()
```

```
# Plot each column of scores_df against Y and Y_logit
for i, column in enumerate(scores_df.columns):
    # Calculate Pearson correlation coefficient
    corr_Y = scores_df[column].corr(Y)
    corr_Y_logit = scores_df[column].corr(Y_logit)

    sns.regplot(x=scores_df[column], y=Y, ax=axes[i*2], scatter_kws={'s': 10},␣
 ↪line_kws={'color': 'red'})
    axes[i*2].set_title(f"{column} vs Student Proportion\n(Pearson r={corr_Y:.
 ↪2f})")
    axes[i*2].set_xlabel(column)
    axes[i*2].set_ylabel("Student Proportion")

    sns.regplot(x=scores_df[column], y=Y_logit, ax=axes[i*2 + 1],␣
 ↪scatter_kws={'s': 10}, line_kws={'color': 'red'})
    axes[i*2 + 1].set_title(f"{column} vs Logit Transformed Student␣
 ↪Proportion\n(Pearson r={corr_Y_logit:.2f})")
    axes[i*2 + 1].set_xlabel(column)
    axes[i*2 + 1].set_ylabel("Logit Transformed Student Proportion")

# Adjust layout
plt.tight_layout()
plt.show()
```

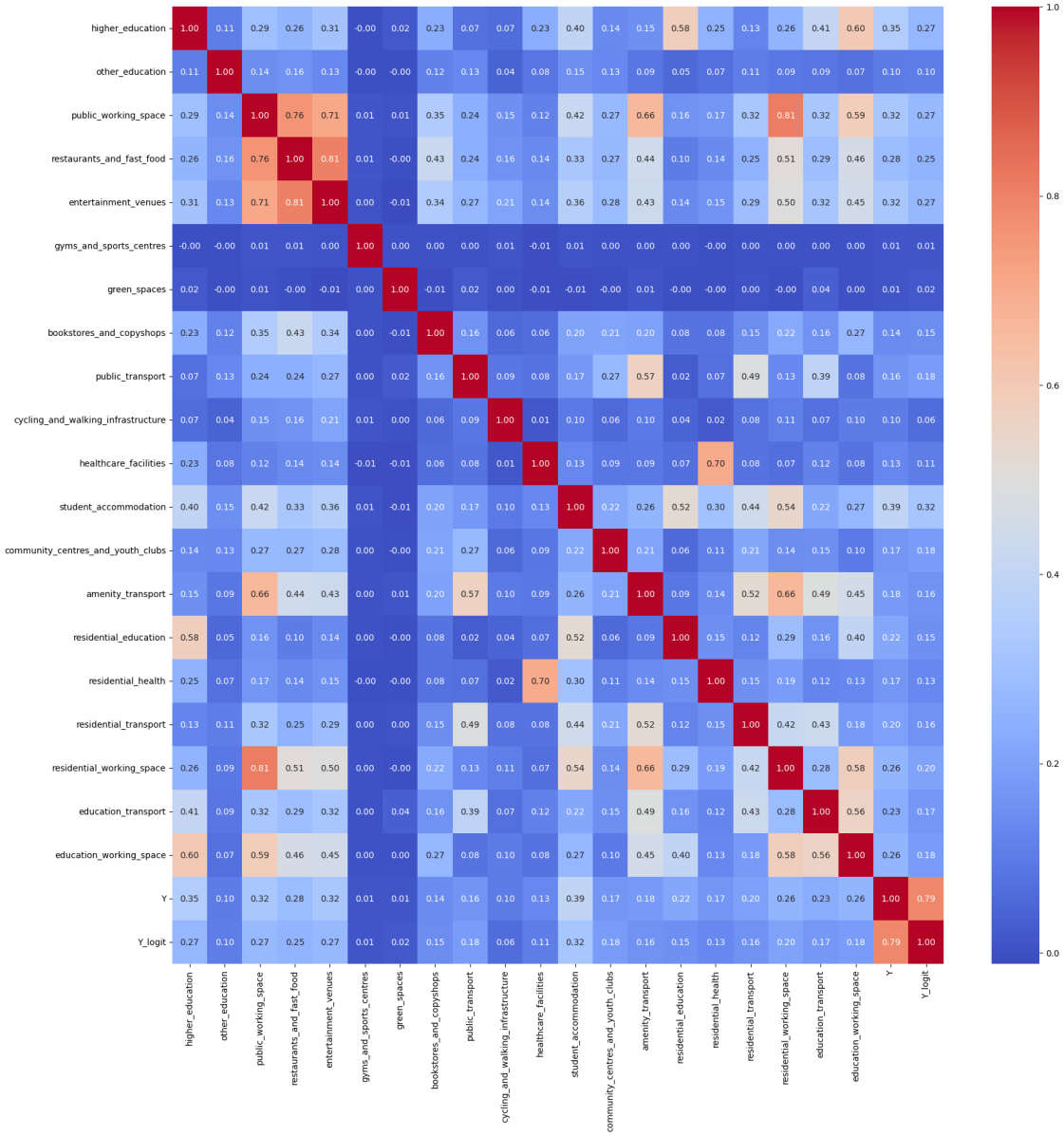


[40]:
```
# Add interaction terms and remove rivers and beaches
scores_df['amenity_transport'] = scores_df['public_transport'] *␣
 ↪scores_df['public_working_space']
scores_df['residential_education'] = scores_df['student_accommodation'] *␣
 ↪scores_df['higher_education']
scores_df['residential_health'] = scores_df['student_accommodation'] *␣
 ↪scores_df['healthcare_facilities']
scores_df['residential_transport'] = scores_df['student_accommodation'] *␣
 ↪scores_df['public_transport']
scores_df['residential_working_space'] = scores_df['student_accommodation'] *␣
 ↪scores_df['public_working_space']
scores_df['education_transport'] = scores_df['higher_education'] *␣
 ↪scores_df['public_transport']
scores_df['education_working_space'] = scores_df['higher_education'] *␣
 ↪scores_df['public_working_space']

scores_df = scores_df.drop(['rivers', 'beaches'], axis=1)
```

[41]:
```
scores_df.columns
```

[41]:
```
Index(['higher_education', 'other_education', 'public_working_space',
       'restaurants_and_fast_food', 'entertainment_venues',
       'gyms_and_sports_centres', 'green_spaces', 'bookstores_and_copyshops',
       'public_transport', 'cycling_and_walking_infrastructure',
       'healthcare_facilities', 'student_accommodation',
       'community_centres_and_youth_clubs', 'amenity_transport',
       'residential_education', 'residential_health', 'residential_transport',
       'residential_working_space', 'education_transport',
       'education_working_space'],
      dtype='object')
```

[44]:
```
scores_with_Y = scores_df
scores_with_Y["Y"] = Y
scores_with_Y["Y_logit"] = Y_logit

plt.figure(figsize=(20, 20))
sns.heatmap(scores_with_Y.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.show()
```



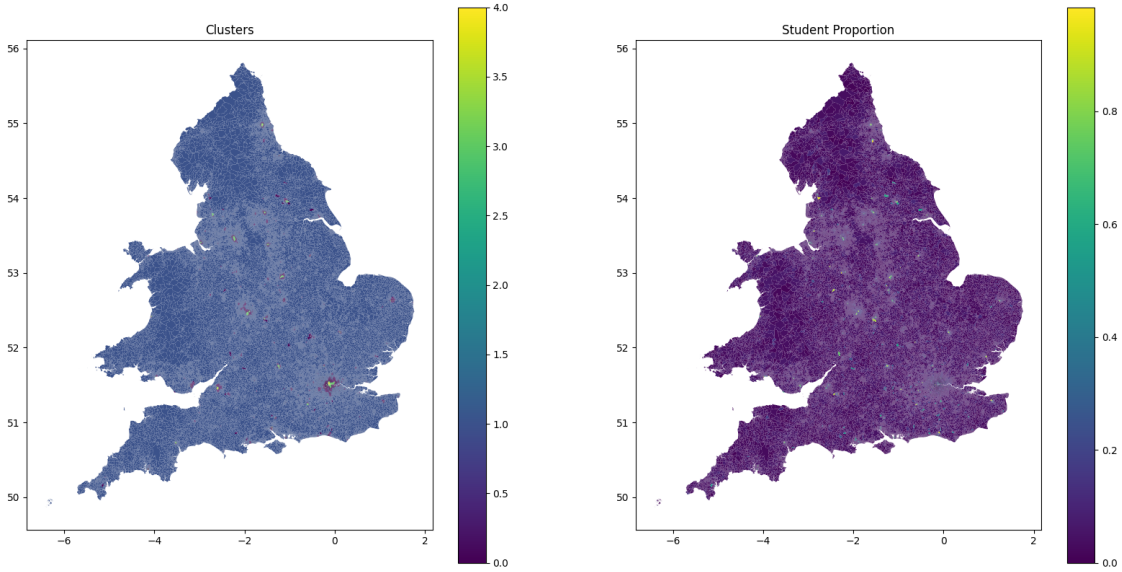
1.0.7 Correlation Analysis of Features

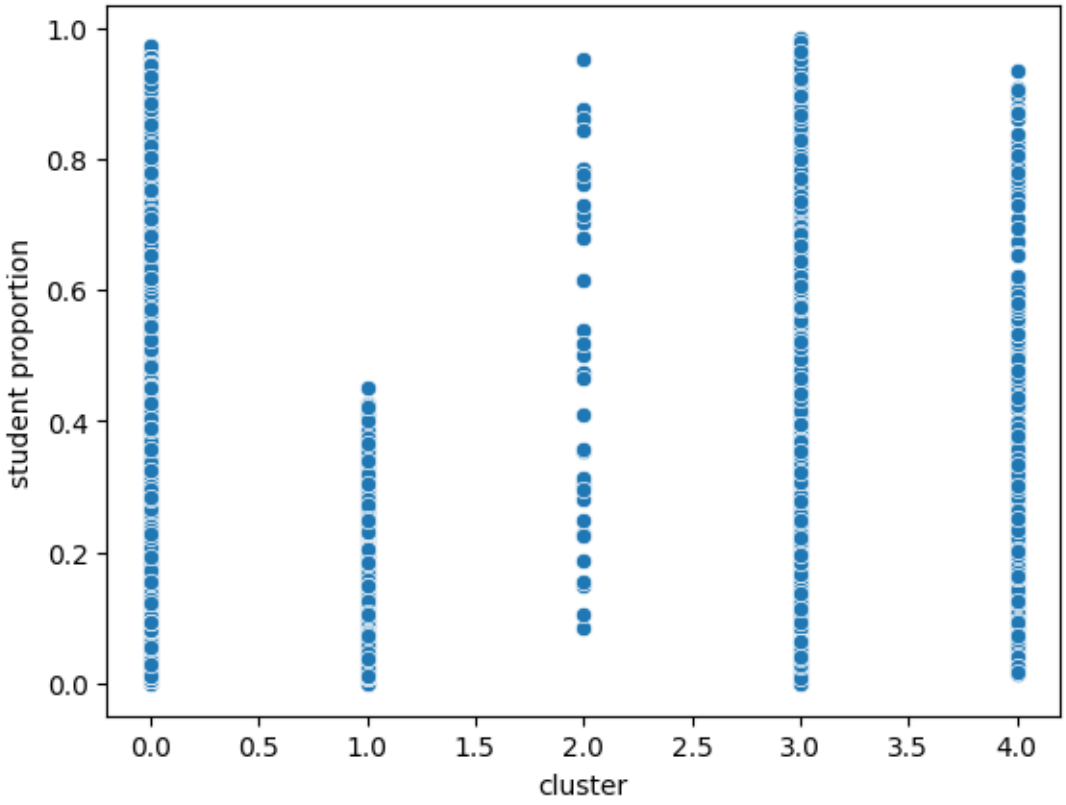
Upon examining the correlation between various features and the student proportion within each output area, it is evident that none of the features exhibit a strong correlation. The highest correlation observed is below 0.4, indicating that we do not have any robust predictors. The feature with the highest correlation is student accommodation, followed by higher education. Despite these being the best predictors in our dataset, their correlation values suggest that they alone are insufficient to accurately predict student proportions. This highlights the need for further exploration and potentially the inclusion of additional variables to improve the predictive power of our model.

1.0.8 Cluster Analysis

[45]:
```
scores_with_geometry = oas.set_index("code")[["geometry"]].join(scores_df,␣
 ↪how="left").compute()
to_plot = scores_with_geometry
```

[46]:
```
# cluster the output areas based on the scores, and then plot
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

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

scaler = StandardScaler()
numeric = scaler.fit_transform(numeric)

kmeans = KMeans(n_clusters=5, random_state=0).fit(numeric)


to_plot["cluster"] = kmeans.labels_
to_plot = to_plot.join(Y)

# 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()

# plot cluster against student proportion
sns.scatterplot(data=to_plot, x="cluster", y="student proportion")
plt.show()
```

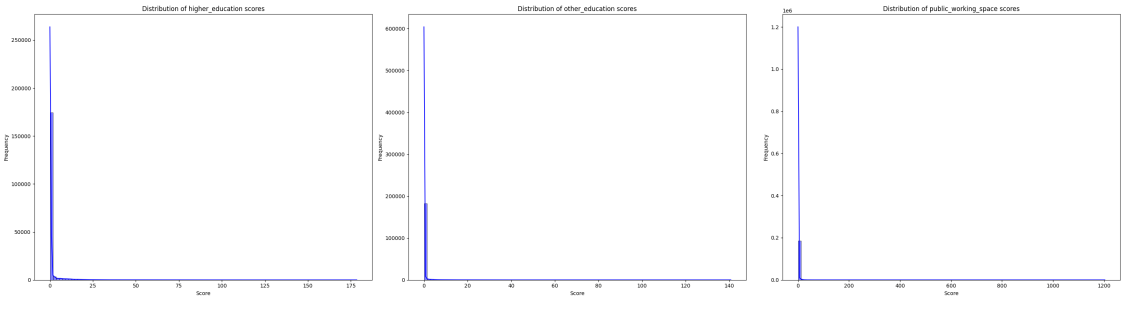

Clusters
Student Proportion


student proportion
cluster

1.0.9 Addressing Skewed Distribution of Scores

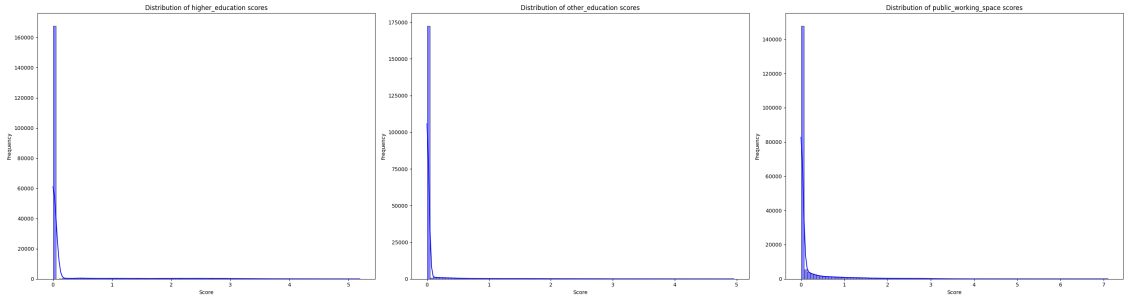
The distribution of scores for various features is highly skewed, which can adversely affect the performance and interpretability of our models. To address this issue, we can apply transformations to the scores to make their distribution more normal. Common transformations include the logarithmic transformation, which can reduce right skewness by compressing the range of high values, and the Box-Cox transformation, which can handle both positive and negative skewness by applying a power transformation. These transformations can help stabilize variance, make the data more normally distributed, and improve the robustness of statistical analyses and machine learning models. By transforming the scores, we can ensure that our models are better able to capture the underlying relationships between features and the target variable, leading to more accurate and reliable predictions.

[ ]:
```
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()
```

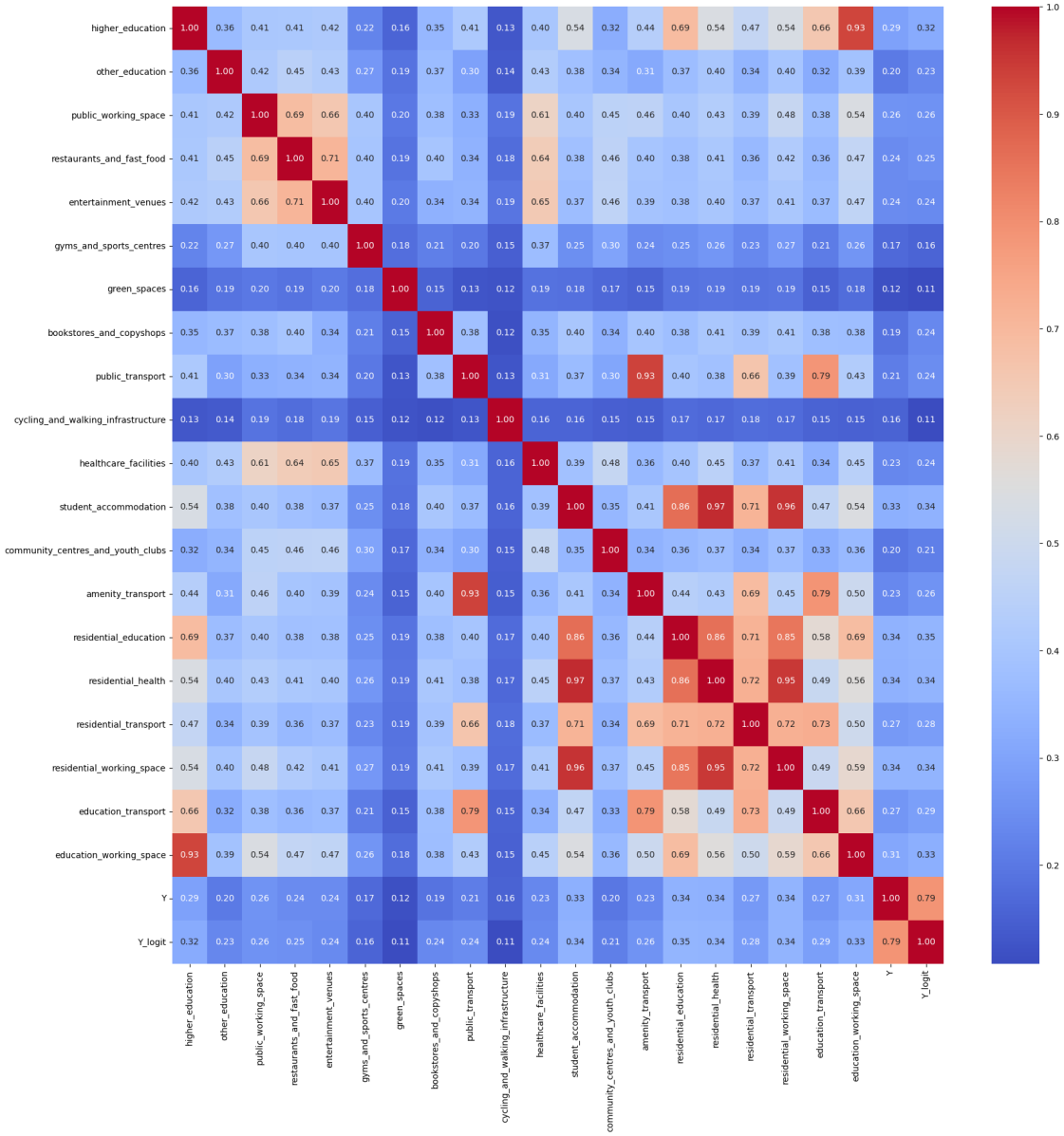


[ ]:
```
from scipy.stats import boxcox

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

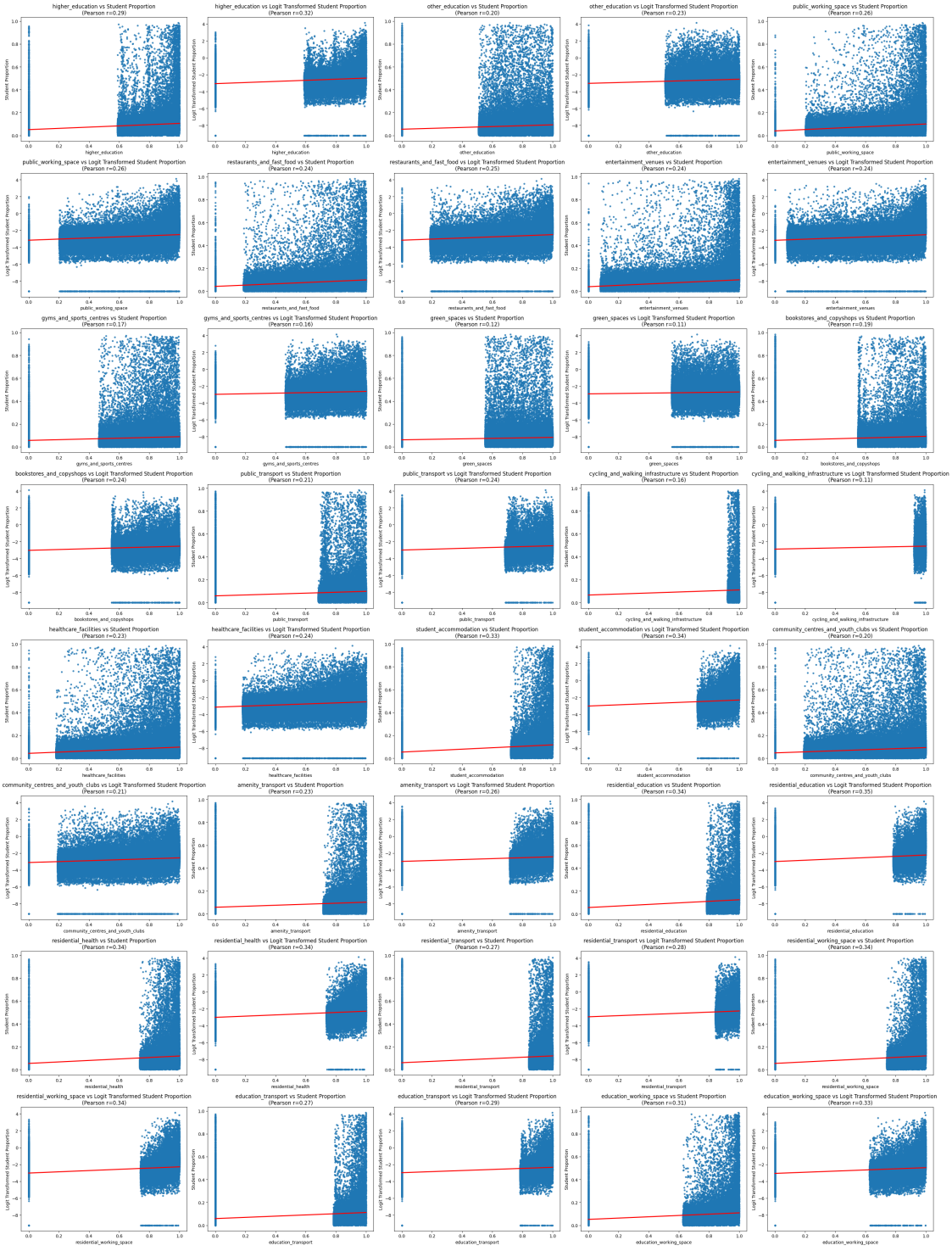


[55]:
```
ranked_scores = scores_df.rank(axis=0, method='min', pct=True)
ranked_scores_with_Y = ranked_scores.copy()
ranked_scores_with_Y["Y"] = Y
ranked_scores_with_Y["Y_logit"] = Y_logit

plt.figure(figsize=(20, 20))
sns.heatmap(ranked_scores_with_Y.corr(), annot=True, fmt=".2f", cmap="coolwarm")
plt.show()
```




```
[ ]: fig, axes = plt.subplots(ncols=5, nrows=8, figsize=(30, 40))

     # Flatten the axes array for easy iteration
     axes = axes.flatten()

     # Plot each column of ranked_scores against Y and Y_logit
     for i, column in enumerate(ranked_scores.columns):
         # Calculate Pearson correlation coefficient
         corr_Y = ranked_scores[column].corr(Y)
         corr_Y_logit = ranked_scores[column].corr(Y_logit)
```

```
    sns.regplot(x=ranked_scores[column], y=Y, ax=axes[i*2], scatter_kws={'s':␣
↪10}, line_kws={'color': 'red'})
    axes[i*2].set_title(f"{column} vs Student Proportion\n(Pearson r={corr_Y:.
↪2f})")
    axes[i*2].set_xlabel(column)
    axes[i*2].set_ylabel("Student Proportion")

    sns.regplot(x=ranked_scores[column], y=Y_logit, ax=axes[i*2 + 1],␣
↪scatter_kws={'s': 10}, line_kws={'color': 'red'})
    axes[i*2 + 1].set_title(f"{column} vs Logit Transformed Student␣
↪Proportion\n(Pearson r={corr_Y_logit:.2f})")
    axes[i*2 + 1].set_xlabel(column)
    axes[i*2 + 1].set_ylabel("Logit Transformed Student Proportion")

# Adjust layout
plt.tight_layout()
plt.show()
```

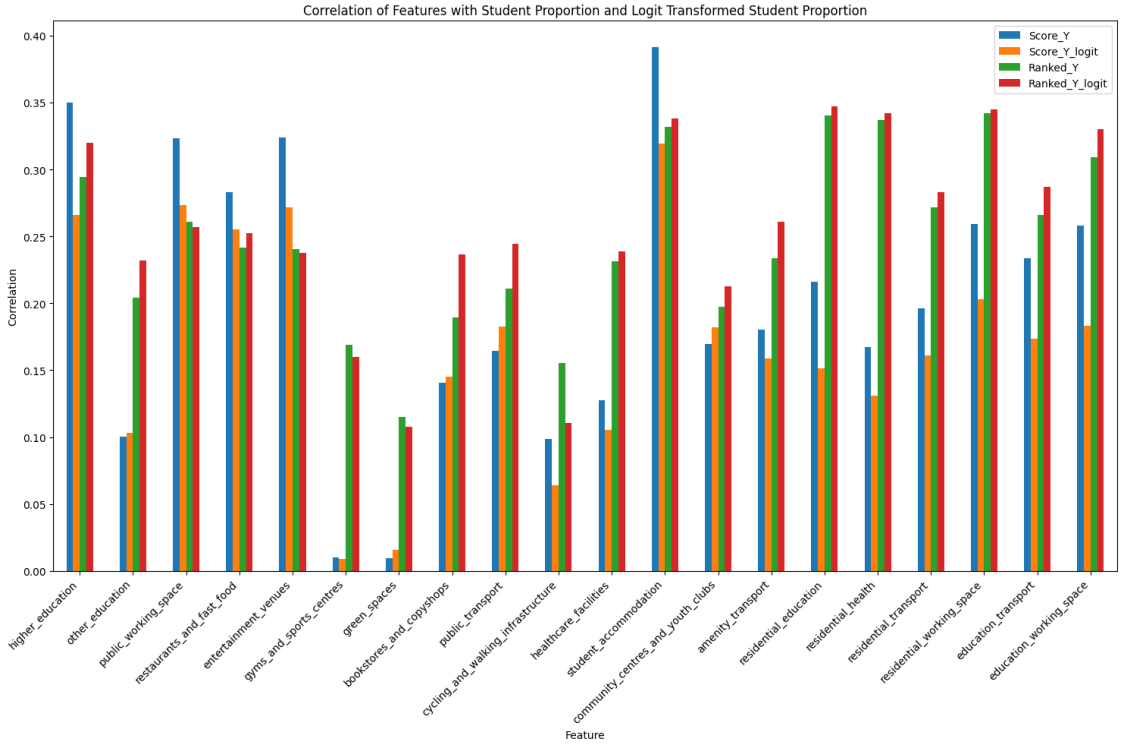


1.0.10 Ranking

Using percentile ranking significantly improves the correlations for a wide range of features. This method effectively addresses the skewed distribution of the scores by transforming them into a

uniform distribution. By ranking the scores and converting them into percentiles, the variance is stabilized, and the range of values is compressed. This enhances the robustness of statistical analyses and machine learning models. Consequently, the ranked scores provide a more accurate and reliable representation of the underlying relationships between features and the target variable, leading to improved predictive power and interpretability of the models.

[58]:
```
# Calculate the correlation of each feature with Y and Y_logit for both score␣
 ↪and ranked form
correlations = pd.DataFrame({
    'Feature': scores_df.columns,
    'Score_Y': [scores_df[col].corr(Y) for col in scores_df.columns],
    'Score_Y_logit': [scores_df[col].corr(Y_logit) for col in scores_df.
 ↪columns],
    'Ranked_Y': [ranked_scores[col].corr(Y) for col in ranked_scores.columns],
    'Ranked_Y_logit': [ranked_scores[col].corr(Y_logit) for col in␣
 ↪ranked_scores.columns]
})

# Plot the correlations
fig, ax = plt.subplots(figsize=(15, 10))
correlations.set_index('Feature').plot(kind='bar', ax=ax)
ax.set_title('Correlation of Features with Student Proportion and Logit␣
 ↪Transformed Student Proportion')
ax.set_ylabel('Correlation')
ax.set_xlabel('Feature')
plt.xticks(rotation=45, ha='right')
plt.tight_layout()
plt.show()
```



[59]:
```
correlations
```

[59]:

	Feature	Score_Y	Score_Y_logit	Ranked_Y	\
0	higher_education	0.349842	0.266228	0.294287	
1	other_education	0.100317	0.103104	0.204063	
2	public_working_space	0.323099	0.273082	0.260690	
3	restaurants_and_fast_food	0.283125	0.254941	0.241493	
4	entertainment_venues	0.323825	0.271544	0.240400	
5	gyms_and_sports_centres	0.009997	0.008927	0.168858	
6	green_spaces	0.009299	0.015997	0.115054	
7	bookstores_and_copyshops	0.140558	0.145320	0.189608	
8	public_transport	0.164196	0.182720	0.211098	
9	cycling_and_walking_infrastructure	0.098793	0.063808	0.155549	
10	healthcare_facilities	0.127797	0.105446	0.231177	
11	student_accommodation	0.391421	0.319531	0.331796	
12	community_centres_and_youth_clubs	0.169290	0.181871	0.197420	
13	amenity_transport	0.180277	0.158581	0.233636	
14	residential_education	0.215822	0.151210	0.340009	
15	residential_health	0.167099	0.131045	0.336820	
16	residential_transport	0.196328	0.161183	0.271898	
17	residential_working_space	0.259442	0.203211	0.341809	
18	education_transport	0.233900	0.173539	0.266196	
19	education_working_space	0.258073	0.183335	0.308857	

```
    Ranked_Y_logit
0         0.319601
1         0.232141
2         0.257104
3         0.252590
4         0.237535
5         0.159892
6         0.107893
7         0.236347
8         0.244327
9         0.110668
10        0.238967
11        0.338162
12        0.212633
13        0.260720
14        0.346880
15        0.341965
16        0.283199
17        0.344779
18        0.286870
19        0.330132
```

[63]:
```
# take the max of all four values and see if it's ranked or scored
correlations["Max"] = correlations[["Score_Y", "Score_Y_logit", "Ranked_Y",␣
 ↪"Ranked_Y_logit"]].abs().max(axis=1)
correlations["Type"] = correlations["Max"].apply(lambda x: "Score" if x in␣
 ↪correlations["Score_Y"].abs().values or x in correlations["Score_Y_logit"].
 ↪abs().values else "Ranked")
correlations["logit_better"] = ((correlations["Type"] == "Score") &␣
 ↪(correlations["Score_Y_logit"].abs() > correlations["Score_Y"].abs())) |␣
 ↪((correlations["Ranked_Y_logit"].abs() > correlations["Ranked_Y"].abs()))
correlations
```

[63]:
```
                              Feature   Score_Y  Score_Y_logit  Ranked_Y
Ranked_Y_logit  score_logit_better  ranked_logit_better       Max    Type
logit_better
0                     higher_education  0.349842       0.266228  0.294287
0.319601               False                 True  0.349842   Score
True
1                      other_education  0.100317       0.103104  0.204063
0.232141                True                 True  0.232141  Ranked
True
2                 public_working_space  0.323099       0.273082  0.260690
0.257104               False                False  0.323099   Score
False
3            restaurants_and_fast_food  0.283125       0.254941  0.241493
```

```
0.252590               False                True  0.283125   Score
True
4                entertainment_venues  0.323825       0.271544  0.240400
0.237535               False                False  0.323825   Score
False
5             gyms_and_sports_centres  0.009997       0.008927  0.168858
0.159892               False                False  0.168858  Ranked
False
6                        green_spaces  0.009299       0.015997  0.115054
0.107893                True                False  0.115054  Ranked
False
7            bookstores_and_copyshops  0.140558       0.145320  0.189608
0.236347                True                 True  0.236347  Ranked
True
8                    public_transport  0.164196       0.182720  0.211098
0.244327                True                 True  0.244327  Ranked
True
9   cycling_and_walking_infrastructure  0.098793       0.063808  0.155549
0.110668               False                False  0.155549  Ranked
False
10              healthcare_facilities  0.127797       0.105446  0.231177
0.238967               False                True  0.238967  Ranked
True
11              student_accommodation  0.391421       0.319531  0.331796
0.338162               False                True  0.391421   Score
True
12   community_centres_and_youth_clubs  0.169290       0.181871  0.197420
0.212633                True                 True  0.212633  Ranked
True
13                  amenity_transport  0.180277       0.158581  0.233636
0.260720               False                True  0.260720  Ranked
True
14              residential_education  0.215822       0.151210  0.340009
0.346880               False                True  0.346880  Ranked
True
15                 residential_health  0.167099       0.131045  0.336820
0.341965               False                True  0.341965  Ranked
True
16              residential_transport  0.196328       0.161183  0.271898
0.283199               False                True  0.283199  Ranked
True
17          residential_working_space  0.259442       0.203211  0.341809
0.344779               False                True  0.344779  Ranked
True
18                education_transport  0.233900       0.173539  0.266196
0.286870               False                True  0.286870  Ranked
True
```

```
19              education_working_space  0.258073       0.183335  0.308857
0.330132                False                  True  0.330132  Ranked
True
```

[64]:
```
# Initialize an empty DataFrame for the feature matrix
feature_matrix = pd.DataFrame(index=scores_df.index)

# Iterate over the features and select from the appropriate DataFrame
for feature in correlations["Feature"]:
    if correlations.loc[correlations["Feature"] == feature, "Type"].values[0]␣
 ↪== "Score":
        feature_matrix[feature] = scores_df[feature]
    else:
        feature_matrix[feature] = ranked_scores[feature]

# Display the feature matrix
feature_matrix
```

[64]:
```
           higher_education  other_education  public_working_space
restaurants_and_fast_food  entertainment_venues  gyms_and_sports_centres
green_spaces  bookstores_and_copyshops  public_transport
cycling_and_walking_infrastructure  healthcare_facilities  student_accommodation
community_centres_and_youth_clubs  amenity_transport  residential_education
residential_health  residential_transport  residential_working_space
education_transport  education_working_space
E00000001        228.704393         0.887982             11.212272
25.446743             33.383041                0.927033      0.788659
0.984265          0.925386                            0.999624
0.844208              26.978518                            0.916164
0.982936               0.984985           0.975265               0.987992
0.994473              0.977721                0.986934
E00000003        235.016824         0.897930             12.260396
25.113051             34.245359                0.930808      0.786420
0.984127          0.926784                            0.999396
0.844737              27.282454                            0.926440
0.985350               0.985589           0.975535               0.988681
0.995262              0.978918                0.988289
E00000005        234.351321         0.903579             12.111465
27.224627             34.765657                0.917921      0.784191
0.983741          0.927303                            0.998263
0.837950              26.664052                            0.903849
0.985250               0.985202           0.974259               0.988352
0.994965              0.979018                0.988077
E00000007        222.115700         0.850688             10.572545
22.780747             33.775396                0.864178      0.774783
0.981591          0.939946                            0.994510
0.901260              23.833159                            0.871241
```

```
0.986335              0.982873           0.979087              0.989660
0.992641            0.981914              0.985986
E00000010        247.666206        0.852165            12.820952
26.987618             33.180614                0.966158     0.803145
0.983693          0.924799                          0.995627
0.826853              33.605559                         0.960917
0.985668              0.988781           0.978155              0.991608
0.996797            0.979447              0.989125
…                      …               …                   …
…                 …                  …              …
…             …                                 …                     …
…                               …               …                     …
…                   …                        …                  …
…
W00010693          0.513706        0.000005              0.500777
0.765684              2.021278                 0.620537     0.789586
0.000005          0.000005                          0.000005
0.288098              0.000000                          0.614157
0.000005              0.000005           0.000005              0.000005
0.000005            0.000005              0.684747
W00010694          0.000000        0.692816              1.506790
0.665379              1.362841                 0.874333     0.736393
0.000005          0.000005                          0.000005
0.471998              0.000000                          0.341391
0.000005              0.000005           0.000005              0.000005
0.000005            0.000005              0.000005
W00010695          0.000000        0.000005              0.722091
2.108880              0.885610                 0.724439     0.000005
0.000005          0.000005                          0.000005
0.212241              0.000000                          0.000005
0.000005              0.000005           0.000005              0.000005
0.000005            0.000005              0.000005
W00010696          0.000000        0.512346              0.543361
1.481928              2.086869                 0.606766     0.788289
0.924232          0.809551                          0.000005
0.613908              0.000000                          0.205723
0.808519              0.000005           0.000005              0.000005
0.000005            0.000005              0.000005
W00010697          0.000000        0.511139              0.131369
3.288268              5.367231                 0.791492     0.851980
0.809376          0.842440                          0.000005
0.675874              0.000000                          0.953187
0.771522              0.000005           0.000005              0.000005
0.000005            0.000005              0.000005

[188880 rows x 20 columns]
```

[65]:
```
# run a VIF analysis to check for multicollinearity
from statsmodels.stats.outliers_influence import variance_inflation_factor

# Create a DataFrame to store the VIF results
vif_results = pd.DataFrame(index=feature_matrix.columns, columns=["VIF"])

# Calculate the VIF for each feature
for i, feature in enumerate(feature_matrix.columns):
    vif_results.loc[feature, "VIF"] = variance_inflation_factor(feature_matrix.
 ↪values, i)

# Display the VIF results
vif_results
```

[65]:

	VIF
higher_education	1.36879
other_education	2.706079
public_working_space	3.325968
restaurants_and_fast_food	4.753891
entertainment_venues	4.286037
gyms_and_sports_centres	2.490895
green_spaces	1.819342
bookstores_and_copyshops	2.509321
public_transport	12.988295
cycling_and_walking_infrastructure	1.195526
healthcare_facilities	5.416159
student_accommodation	1.748573
community_centres_and_youth_clubs	4.217851
amenity_transport	13.05175
residential_education	7.456409
residential_health	17.399424
residential_transport	5.096823
residential_working_space	15.958552
education_transport	6.070857
education_working_space	4.827489

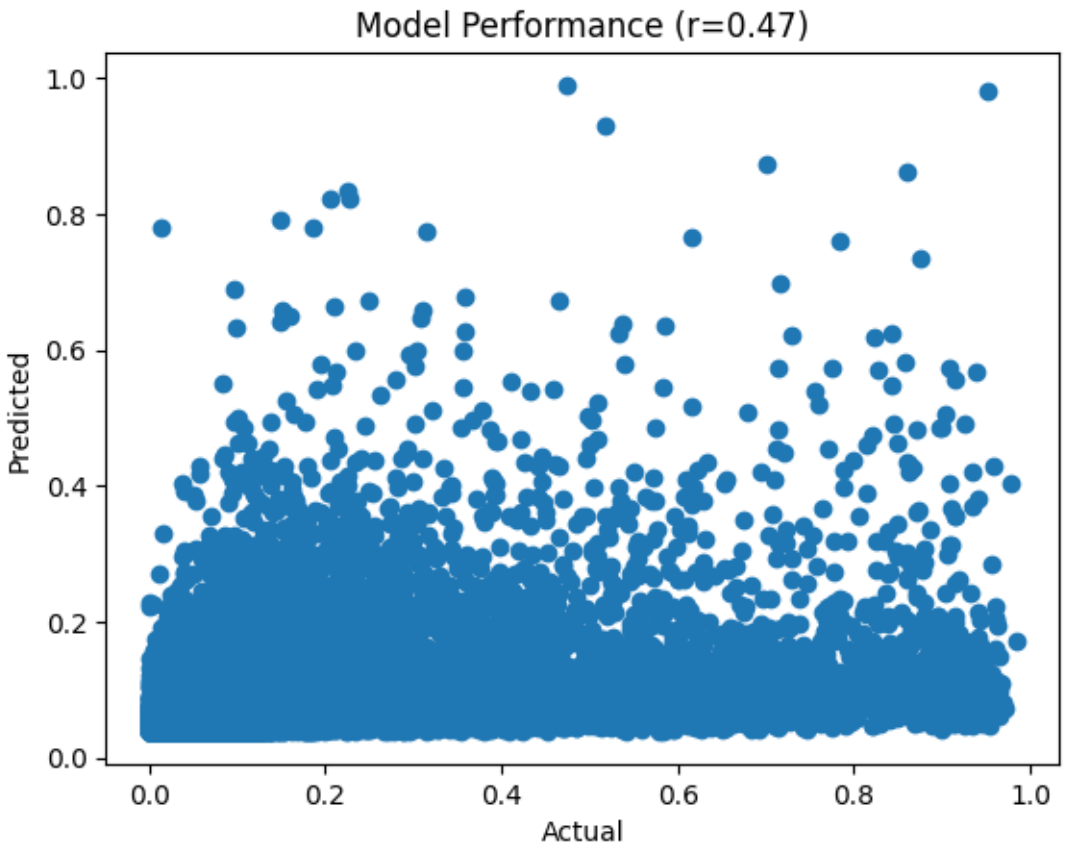
[66]:
```
feature_matrix.drop(columns=["residential_working_space", "residential_health",␣
 ↪"amenity_transport"], inplace=True)
```

[67]:
```
# re-run the VIF analysis
vif_results = pd.DataFrame(index=feature_matrix.columns, columns=["VIF"])

for i, feature in enumerate(feature_matrix.columns):
    vif_results.loc[feature, "VIF"] = variance_inflation_factor(feature_matrix.
 ↪values, i)

vif_results
```

[67]:

	VIF
higher_education	1.365598
other_education	2.679258
public_working_space	3.303607
restaurants_and_fast_food	4.74624
entertainment_venues	4.282967
gyms_and_sports_centres	2.48311
green_spaces	1.818978
bookstores_and_copyshops	2.485451
public_transport	4.449416
cycling_and_walking_infrastructure	1.195039
healthcare_facilities	5.167574
student_accommodation	1.701998
community_centres_and_youth_clubs	4.200968
residential_education	4.549357
residential_transport	4.299768
education_transport	5.781999
education_working_space	4.543645

[73]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
from scipy.special import expit

def train_model(X, Y):
    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2,␣
 ↪random_state=42)
    model = LinearRegression()
    model.fit(X_train, Y_train)
    Y_pred = model.predict(X_test)
    mse = mean_squared_error(Y_test, Y_pred)
    r2 = r2_score(Y_test, Y_pred)
    return model, mse, r2

def plot_model(model, X, Y):
    Y_pred = model.predict(X)
    Y_sig = expit(Y)
    Y_pred_sig = expit(Y_pred)
    corr = np.corrcoef(Y_sig, Y_pred_sig)[0, 1]
    plt.scatter(expit(Y), expit(Y_pred))
    plt.xlabel("Actual")
    plt.ylabel("Predicted")
    plt.title(f"Model Performance (r={corr:.2f})")
    plt.show()

model, mse, r2 = train_model(feature_matrix, Y_logit)
plot_model(model, feature_matrix, Y_logit)
```


Model Performance (r=0.47)
Predicted
Actual
0.0
0.2
0.4
0.6
0.8
1.0

[ ]: