In [1]:
```
from IPython.display import HTML
HTML('''
    <style> body {font-family: "Roboto Condensed Light", "Roboto Condensed";} h2 {padding:
    content: "@"; font-family:"Wingdings"; font-style:regular; margin-right: 4px;} .text_c
''')
```

Out[1]:

In [2]:
```
import pandas as pd #pandas
import numpy as np
import geopandas as gpd #geopandas
from shapely.geometry import Point, Polygon, MultiPolygon
from geoalchemy2 import Geometry, WKTElement
import matplotlib.pyplot as plt
from sqlalchemy import create_engine #sql
import psycopg2
import psycopg2.extras
import json
import os
from statistics import mean, median, mode, stdev #stats
import seaborn as sns
#read csv & txt
income = pd.read_csv('Income.csv')
population = pd.read_csv('Population.csv')
polling = pd.read_csv('PollingPlaces2019.csv')
business = pd.read_csv('Businesses.csv')
stops = pd.read_csv('Stops.csv')
#read shp
catchment_future = gpd.read_file("Catchments\catchments\catchments_future.shp")
catchment_secondary = gpd.read_file("Catchments\catchments\catchments_secondary.shp")
catchment_primary = gpd.read_file("Catchments\catchments\catchments_primary.shp")
sa2 = gpd.read_file("SA2_2021_AUST_SHP_GDA2020-20230510T075423Z-001\SA2_2021_AUST_SHP_GDA2
```

TASK 1: Original data sets

1.1. Cleaning

In [3]:
```
#1 income
# primary = sa2_code & sa2_name
# change np to numerical value
income = income.replace("np", 0)
# convert all numerical col to from obj to int
income['earners'] = income['earners'].astype('int64')
income['median_age'] = income['median_age'].astype('int64')
income['median_income'] = income['median_income'].astype('int64')
income['mean_income'] = income['mean_income'].astype('int64')
```

In [4]:
```
#2 polling
# primary = FID
# 140 rows are dropped contains NaN in all 3 location attributes
polling = polling.dropna(subset=['longitude', 'latitude', 'the_geom'])
# there are still na in other col, keep in sql as type NULL
# drop column state because all is NSW
polling = polling.drop('state', axis=1)
# create geom = (lon, lat)
polling['geom'] = gpd.points_from_xy(polling.longitude, polling.latitude)
```

```
polling = polling.drop(columns=['longitude', 'latitude', 'the_geom'])
polling['geom'] = polling['geom'].apply(lambda x: WKTElement(x.wkt, srid=4326))
```

In [5]:
```
#3 business
# no primary & no nan
business.isnull().sum().sum()

#4 population
# primary = sa2_code & sa2_name
# no nan val, clean
population.isnull().sum().sum()
```

Out[5]: 0

In [6]:
```
#5 stops
# primary = stop_id
# no NaN in important cols <stop_lat & stop_lon>
# create geom = (lon, lat)
stops['geom'] = gpd.points_from_xy(stops.stop_lon, stops.stop_lat)
stops = stops.drop(columns=['stop_lat', 'stop_lon'])
stops['geom'] = stops['geom'].apply(lambda x: WKTElement(x.wkt, srid=4326))
```

In [7]:
```
#6 Catchment
# primary = use_id
# no NaN in whole data set
catchment_future.isnull().sum().sum()
# no NaN in important column <geom> for other 2 catchment_*
# convert use_id to int
catchment_future['USE_ID'] = catchment_future['USE_ID'].astype('int64')
catchment_secondary['USE_ID'] = catchment_secondary['USE_ID'].astype('int64')
catchment_primary['USE_ID'] = catchment_primary['USE_ID'].astype('int64')

# CONVERT POLYGON TO MULTIPOLYGON
def create_wkt_element(geom, srid):
    if geom.geom_type == 'Polygon':
        geom = MultiPolygon([geom])
    return WKTElement(geom.wkt, srid)

catchment_future['geom'] = catchment_future['geometry'].apply(lambda x: create_wkt_element
catchment_future = catchment_future.drop(columns='geometry')
catchment_future = catchment_future.replace({0:'N', 2024:'Y', 2025:'Y', 2026:'Y', 2027:'Y'

catchment_secondary['geom'] = catchment_secondary['geometry'].apply(lambda x: create_wkt_e
catchment_secondary = catchment_secondary.drop(columns=['geometry', 'PRIORITY'])

catchment_primary['geom'] = catchment_primary['geometry'].apply(lambda x: create_wkt_eleme
catchment_primary = catchment_primary.drop(columns=['geometry', 'PRIORITY'])

# MERGE 3 CATCHMENT TO SCHOOL
catchment = [catchment_future, catchment_secondary, catchment_primary]
catchment = pd.concat(catchment) # exist duplicated students
```

In [8]:
```
#7 SA2
# primary = sa2_code21 & sa2_name_21
# Greater Sydney only
sa2 = sa2.drop(sa2[sa2['GCC_NAME21'] != 'Greater Sydney'].index)
# dropped row that has NaN in <geom>
sa2 = sa2.dropna(subset=['geometry'])
# convert to int
sa2['SA2_CODE21'] = sa2['SA2_CODE21'].astype('int64')
```

```
# CONVERT POLYGON TO MULTIPOLYGON
sa2['geom'] = sa2['geometry'].apply(lambda x: create_wkt_element(geom=x,srid=4326))
sa2 = sa2[['SA2_CODE21', 'SA2_NAME21', 'AREASQKM21', 'geom']]
```

1.2. SQL

In [9]:

```
# CONNECT TO SERVER
credentials = "Credentials.json"

def pgconnect(credential_filepath, db_schema="public"):
    with open(credential_filepath) as f:
        db_conn_dict = json.load(f)
        host       = db_conn_dict['host']
        db_user    = db_conn_dict['user']
        db_pw      = db_conn_dict['password']
        default_db = db_conn_dict['user']
        try:
            db = create_engine('postgresql+psycopg2://'+db_user+':'+db_pw+'@'+host+'/'+def
            conn = db.connect()
            print('Connected successfully.')
        except Exception as e:
            print("Unable to connect to the database.")
            print(e)
            db, conn = None, None
        return db,conn

# RETURN QUERY AS PANDAS DATA FRAME
def query(conn, sqlcmd, args=None, df=True):
    result = pd.DataFrame() if df else None
    try:
        if df:
            result = pd.read_sql_query(sqlcmd, conn, params=args)
        else:
            result = conn.execute(sqlcmd, args).fetchall()
            result = result[0] if len(result) == 1 else result
    except Exception as e:
        print("Error encountered: ", e, sep='\n')
    return result

db, conn = pgconnect(credentials)
```

```
Connected successfully.
```

In [10]:

```
query(conn, "select PostGIS_Version()")
```

Out[10]:

	postgis_version
0	3.3 USE_GEOS=1 USE_PROJ=1 USE_STATS=1

In [11]:

```
# LOADING INTO DATABASE
```

In [12]:

```
# CREATE SCHEMA: ED 878
# 1 income <csv>
conn.execute("""
DROP TABLE IF EXISTS income CASCADE;
CREATE TABLE income (
   sa2_code INTEGER PRIMARY KEY,
   sa2_name VARCHAR(50) NOT NULL,
   earners INTEGER,
```

```
   median_age INTEGER,
   median_income INTEGER,
   mean_income INTEGER
);""")
# 2 polling <csv>
conn.execute("""
DROP TABLE IF EXISTS polling CASCADE;
CREATE TABLE polling (
   fid VARCHAR(100) PRIMARY KEY,
   division_id INTEGER,
   division_name VARCHAR(20),
   polling_place_id INTEGER,
   polling_place_type_id INTEGER,
   polling_place_name VARCHAR(100),
   premises_name VARCHAR(100),
   premises_address_1 VARCHAR(100),
   premises_address_2 VARCHAR(100),
   premises_address_3 VARCHAR(100),
   premises_suburb VARCHAR(100),
   premises_state_abbreviation VARCHAR(10),
   premises_post_code FLOAT,
   geom GEOMETRY(POINT,4326)
);""")

# 3 business <csv>
conn.execute("""
DROP TABLE IF EXISTS business CASCADE;
CREATE TABLE business (
   industry_code VARCHAR(10),
   industry_name VARCHAR(100),
   sa2_code INTEGER,
   sa2_name VARCHAR(100),
   "0_to_50k_businesses" INTEGER,
   "50k_to_200k_businesses" INTEGER,
   "200k_to_2m_businesses" INTEGER,
   "2m_to_5m_businesses" INTEGER,
   "5m_to_10m_businesses" INTEGER,
   "10m_or_more_businesses" INTEGER,
   total_businesses INTEGER
);""")

# 4 population <csv>
conn.execute("""
DROP TABLE IF EXISTS population CASCADE;
CREATE TABLE population (
   sa2_code INTEGER PRIMARY KEY,
   sa2_name VARCHAR(50) NOT NULL,
   "0-4_people" INTEGER,
   "5-9_people" INTEGER,
   "10-14_people" INTEGER,
   "15-19_people" INTEGER,
   "20-24_people" INTEGER,
   "25-29_people" INTEGER,
   "30-34_people" INTEGER,
   "35-39_people" INTEGER,
   "40-44_people" INTEGER,
   "45-49_people" INTEGER,
   "50-54_people" INTEGER,
   "55-59_people" INTEGER,
   "60-64_people" INTEGER,
   "65-69_people" INTEGER,
   "70-74_people" INTEGER,
   "75-79_people" INTEGER,
   "80-84_people" INTEGER,
   "85-and-over_people" INTEGER,
   total_people INTEGER
```

```
);""")

# 5 stops <txt>
conn.execute("""
DROP TABLE IF EXISTS stops CASCADE;
CREATE TABLE stops (
   stop_id VARCHAR(100) PRIMARY KEY,
   stop_code VARCHAR(100),
   stop_name VARCHAR(100),
   location_type VARCHAR(100),
   parent_station VARCHAR(100),
   wheelchair_boarding INTEGER,
   platform_code VARCHAR(100),
   geom GEOMETRY(POINT,4326)
);""")

# 6 catchment <shp>
conn.execute("""
DROP TABLE IF EXISTS catchment CASCADE;
CREATE TABLE catchment (
    use_id INTEGER,
    catch_type VARCHAR(80),
    use_desc VARCHAR(80),
    add_date VARCHAR(80),
    kindergart VARCHAR(10),
    year1 VARCHAR(10),
    year2 VARCHAR(10),
    year3 VARCHAR(10),
    year4 VARCHAR(10),
    year5 VARCHAR(10),
    year6 VARCHAR(10),
    year7 VARCHAR(10),
    year8 VARCHAR(10),
    year9 VARCHAR(10),
    year10 VARCHAR(10),
    year11 VARCHAR(10),
    year12 VARCHAR(10),
    geom GEOMETRY(MULTIPOLYGON,4326)
);"""
)

# 7 sa2 <shp>
conn.execute("""
DROP TABLE IF EXISTS sa2 CASCADE;
CREATE TABLE sa2 (
    sa2_code21 INTEGER PRIMARY KEY,
    sa2_name21 VARCHAR(50) NOT NULL,
    areasqkm21 FLOAT,
    geom GEOMETRY(MULTIPOLYGON,4326)
);"""
)
```

Out[12]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc87865070>
```

In [13]:
```
# 1 income
income.columns = map(str.lower, income.columns)
income.to_sql("income", con=conn, if_exists='append', index=False)

# 2 polling
polling.columns = map(str.lower,polling.columns)
polling.to_sql('polling', conn, if_exists='append', index=False, dtype={'geom': Geometry(g

# 3 business
business.columns = map(str.lower, business.columns)
```

```
business.to_sql("business", con=conn, if_exists='append', index=False)

# 4 population
population.columns = map(str.lower, population.columns)
population.to_sql("population", con=conn, if_exists='append', index=False)

# 5 stops
stops.columns = map(str.lower, stops.columns)
stops.to_sql('stops', conn, if_exists='append', index=False, dtype={'geom': Geometry(geome

# 6 catchment
catchment.columns = map(str.lower, catchment.columns)
catchment.to_sql("catchment", conn, if_exists='append', index=False, dtype={'geom': Geomet

# 7 sa2
sa2.columns = map(str.lower, sa2.columns)
sa2.to_sql("sa2", conn, if_exists='append', index=False, dtype={'geom': Geometry('MULTIPOI
```

In [14]:

```
# CREATE INDEX ON SCHEMA : optimise queries
conn.execute("CREATE INDEX IF NOT EXISTS iid ON income(sa2_code)")
conn.execute("CREATE INDEX IF NOT EXISTS pgid ON polling USING GIST (geom)")
conn.execute("CREATE INDEX IF NOT EXISTS bid ON business(sa2_code)")
conn.execute("CREATE INDEX IF NOT EXISTS pid ON population(sa2_code)")
conn.execute("CREATE INDEX IF NOT EXISTS sgid ON stops USING GIST (geom)")
conn.execute("CREATE INDEX IF NOT EXISTS cgid ON catchment USING GIST (geom)")
conn.execute("CREATE INDEX IF NOT EXISTS sid ON sa2(sa2_code21)")
conn.execute("CREATE INDEX IF NOT EXISTS sgid ON sa2 USING GIST (geom)")
```

Out[14]:

```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc895880d0>
```

Task 2: Calculate score

In [15]:

```
# IGNORE ALL AREA WITH POPULATION < 100
```

In [16]:

```
# SA2 AREA & POPULATION < 100
conn.execute("""
DROP VIEW IF EXISTS sa100 CASCADE;
CREATE VIEW sa100 AS
    (SELECT sa2_code21, sa2_name21, areasqkm21, geom, total_people,
        CASE
            WHEN total_people < 100 THEN NULL
            ELSE total_people
        END AS "tot_p",
        CASE
            WHEN total_people < 100 THEN NULL
            ELSE "0-4_people"+"5-9_people"+"10-14_people"+"15-19_people"
        END AS "young_p",
        CASE
            WHEN total_people < 100 THEN NULL
            ELSE areasqkm21
        END AS "area"
        FROM sa2 JOIN population ON (sa2_code21 = sa2_code)
        ORDER BY sa2_code21)""")
```

Out[16]:

```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc8786f2b0>
```

2.1. Individual score

In [17]:
```
# 1.1 RETAIL TABLE
conn.execute("""
DROP VIEW IF EXISTS retail_1 CASCADE;
CREATE VIEW retail_1 AS
    (SELECT sa2_code21, total_businesses, tot_p
        FROM business RIGHT JOIN sa100 ON (sa2_code21 = sa2_code)
        WHERE industry_name = 'Retail Trade'
        GROUP BY sa2_code21, total_businesses, tot_p
        ORDER BY sa2_code21)""")
# 1.2 RETAIL SCORE
conn.execute("""
DROP VIEW IF EXISTS retail_2 CASCADE;
CREATE VIEW retail_2 AS
  (SELECT sa2_code21,
   CAST(total_businesses AS FLOAT)/CAST(tot_p AS FLOAT)*1000 AS retail_score
   FROM retail_1)""")
# 1.3 RETAIL ZSCORE
conn.execute("""
DROP VIEW IF EXISTS retail_3 CASCADE;
CREATE VIEW retail_3 AS
    (WITH r3 as
        (SELECT AVG(retail_score) AS mean, STDDEV(retail_score) AS sd
         from retail_2)
     SELECT sa2_code21, (retail_score - r3.mean) / r3.sd AS z_retail
     FROM r3, retail_2)""")
```

Out[17]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc89588400>
```

In [18]:
```
# 2.1 HEALTH TABLE
conn.execute("""
DROP VIEW IF EXISTS health_1 CASCADE;
CREATE VIEW health_1 AS
    (SELECT sa2_code21, total_businesses, tot_p
        FROM business RIGHT JOIN sa100 ON (sa2_code21 = sa2_code)
        WHERE industry_name = 'Health Care and Social Assistance'
        GROUP BY sa2_code21, total_businesses, tot_p
        ORDER BY sa2_code21)""")
# 2.2 HEALTH SCORE
conn.execute("""
DROP VIEW IF EXISTS health_2 CASCADE;
CREATE VIEW health_2 AS
  (SELECT sa2_code21,
   CAST(total_businesses AS FLOAT)/CAST(tot_p AS FLOAT)*1000 AS health_score
   FROM health_1)""")
# 2.3 HEALTH ZSCORE
conn.execute("""
DROP VIEW IF EXISTS health_3 CASCADE;
CREATE VIEW health_3 AS
    (WITH h3 AS
        (SELECT AVG(health_score) AS mean, stddev(health_score) AS sd
         FROM health_2)
     SELECT sa2_code21, (health_score - h3.mean) / h3.sd AS z_health
     FROM h3, health_2)""")
```

Out[18]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc8955d190>
```

In [19]:
```
# 3.1 STOPS TABLE
conn.execute("""
DROP VIEW IF EXISTS stops_1 CASCADE;
CREATE VIEW stops_1 AS
    (SELECT sa2_code21, COUNT(stop_id) AS stops, area
        FROM stops RIGHT JOIN sa100 ON ST_Contains(sa100.geom, stops.geom)
        GROUP BY sa2_code21, area
```

```
        ORDER BY sa2_code21)""")
# 3.2 STOPS SCORE
conn.execute("""
DROP VIEW IF EXISTS stops_2 CASCADE;
CREATE VIEW stops_2 AS
    (SELECT sa2_code21, CAST(stops AS float)/area AS stops_score
     FROM stops_1)""")
# 3.3 STOPS ZSCORE
conn.execute("""
DROP VIEW IF EXISTS stops_3 CASCADE;
CREATE VIEW stops_3 AS
    (WITH s3 AS
        (SELECT AVG(stops_score) AS mean, stddev(stops_score) AS sd
         FROM stops_2)
     SELECT sa2_code21, (stops_score - s3.mean) / s3.sd AS z_stops
     FROM s3, stops_2)
""")
```

Out[19]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc8955dc40>
```

In [20]:
```
# 4.1 POLLING TABLE
conn.execute("""
DROP VIEW IF EXISTS polls_1 CASCADE;
CREATE VIEW polls_1 AS
    (SELECT sa2_code21, COUNT(division_id) AS division, area
        FROM polling RIGHT JOIN sa100 ON ST_Contains(sa100.geom, polling.geom)
        GROUP BY sa2_code21, area
        ORDER BY sa2_code21)""")
# 4.2 POLLING SCORE
conn.execute("""
DROP VIEW IF EXISTS polls_2 CASCADE;
CREATE VIEW polls_2 AS
    (SELECT sa2_code21, CAST(division AS float)/area AS polls_score
     FROM polls_1)""")
# 4.3 POLLING ZSCORE
conn.execute("""
DROP VIEW IF EXISTS polls_3 CASCADE;
CREATE VIEW polls_3 AS
    (WITH p3 AS
        (SELECT AVG(polls_score) AS mean, stddev(polls_score) AS sd
         FROM polls_2)
     SELECT sa2_code21, (polls_score - p3.mean) / p3.sd AS z_polls
     FROM p3, polls_2)""")
```

Out[20]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc8955dbb0>
```

In [21]:
```
# 5.1 SCHOOL TABLE
conn.execute("""
DROP VIEW IF EXISTS schools_1 CASCADE;
CREATE VIEW schools_1 AS
    (SELECT sa2_code21, COUNT(use_id) AS schools, young_p
        FROM sa100 LEFT JOIN catchment ON ST_Overlaps(sa100.geom, catchment.geom)
        GROUP BY sa2_code21, young_p
        ORDER BY sa2_code21)""")
# 5.2 SCHOOL SCORE
conn.execute("""
DROP VIEW IF EXISTS schools_2 CASCADE;
CREATE VIEW schools_2 AS
    (SELECT sa2_code21,
     CAST(schools AS FLOAT)/CAST(young_p AS FLOAT)*1000 AS schools_score
     FROM schools_1)""")
# 5.3 SCHOOL ZSCORE
conn.execute("""
```

```
DROP VIEW IF EXISTS schools_3 CASCADE;
CREATE VIEW schools_3 AS
    (WITH sc3 AS
        (SELECT AVG(schools_score) AS mean, stddev(schools_score) AS sd
         FROM schools_2)
     SELECT sa2_code21, (schools_score - sc3.mean) / sc3.sd AS z_schools
     FROM sc3, schools_2)""")
```

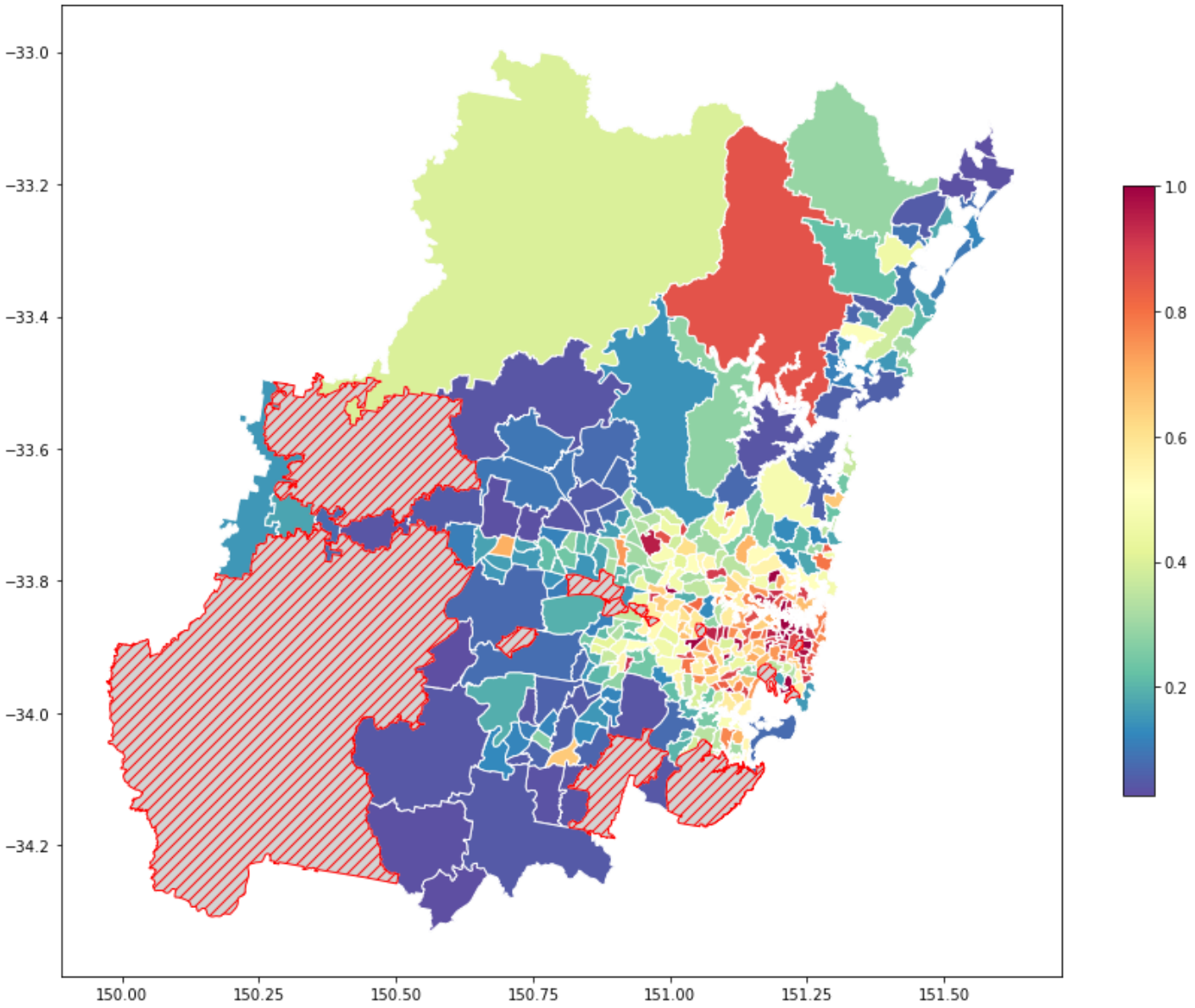
Out[21]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc894a0f10>
```

2.2. Total well-resourced score

In [22]:
```
#ORIGINAL SIGMOID
conn.execute("""
DROP VIEW IF EXISTS sigmoid_1 CASCADE;
CREATE VIEW sigmoid_1 AS
    (SELECT sa2_code21, sa2_name21, area, r.z_retail,
             h.z_health, s.z_stops, p.z_polls, sch.z_schools, geom,
            1/(1 + EXP(-(z_retail + z_health + z_stops +
               z_polls + z_schools))) AS "well_resourced_score"
     FROM sa100 JOIN retail_3 r USING (sa2_code21)
                JOIN health_3 h USING (sa2_code21)
                JOIN stops_3 s USING (sa2_code21)
                JOIN polls_3 p USING (sa2_code21)
                JOIN schools_3 sch USING (sa2_code21))""")
sa2_sigmoid = gpd.read_postgis("SELECT * FROM sigmoid_1", conn)
```

In [23]:
```
#GRAPH WELL-RESOURCED BY SA2 REGIONS
```

In [24]:
```
# TOTAL <WELL_SOURCED> SCORE
sa2_sigmoid.plot(
    column="well_resourced_score",
    legend=True,
    figsize=(15, 15),
    missing_kwds={"color": "lightgrey", "edgecolor": "red",
                  "hatch": "///", "label": "pop <100"},
    cmap='Spectral_r',
    edgecolor = 'white',
    legend_kwds={'shrink': 0.5}
);
```



Task 3: Extensions

3.1. Clean new data sets

In [25]:
```
# 3 DATA SETS
```

In [26]:
```
#1 toilet <json>
# primary = FacilityID
with open('toiletMap.json') as json_data:
    data = json.load(json_data)

fields = data.get('fields')              # heading <list of dictionaries>
heading = [i.get('id') for i in fields]
records = data.get('records')            # data rows <list of list>
toilet = pd.DataFrame(records)           # convert to df
toilet.columns = heading                 # update heading

toilet['geom'] = gpd.points_from_xy(toilet.Longitude, toilet.Latitude)
toilet = toilet.drop(columns=['Longitude', 'Latitude'])
toilet['geom'] = toilet['geom'].apply(lambda x: WKTElement(x.wkt, srid=4326))
# no NaN
toilet = toilet[['FacilityID', 'Name', 'FacilityType', 'Address1', 'Town', 'geom']]
```

In [27]:
```
#2 crime <shp>
# primary = OBJECTID, no NaN
crime = gpd.read_file("CrimeHotspots\DomesticAssault_JanToDec2021.shp")

crime['geom'] = crime['geometry'].apply(lambda x: create_wkt_element(geom=x,srid=4326))
crime = crime.drop(columns='geometry')
```

In [28]:
```
#3 employees <csv>
employees = pd.read_csv('employees.csv', thousands=',')
# drop row where sa2_code contains NaN, 'w'
employees['sa2_code'] = pd.to_numeric(employees['sa2_code'], errors='coerce')
employees = employees[employees['sa2_code'].notna()]
# only NaN left in numeric column
employees = employees.replace(np.nan, 0)
# covert float to int type
employees['sa2_code'] = employees['sa2_code'].astype('int64')
employees['non_employing'] = employees['non_employing'].astype('int64')
employees['1_to_4_employees'] = employees['1_to_4_employees'].astype('int64')
employees['5_to_19_employees'] = employees['5_to_19_employees'].astype('int64')
employees['20_to_199_employees'] = employees['20_to_199_employees'].astype('int64')
employees['200_to_more_employees'] = employees['200_to_more_employees'].astype('int64')
employees['total'] = employees['total'].astype('int64')
```

3.2. SQL

In [29]:
```
# CREATE SCHEMA
# 1 toilet <json>
conn.execute("""
DROP TABLE IF EXISTS toilet CASCADE;
CREATE TABLE toilet (
   facilityid VARCHAR(50) PRIMARY KEY,
   name VARCHAR(150),
   facilitytype VARCHAR(150),
   address1 VARCHAR(150),
   town VARCHAR(100),
   geom GEOMETRY(POINT,4326)
);""")

# 2 crime <shp>
conn.execute("""
DROP TABLE IF EXISTS crime CASCADE;
CREATE TABLE crime (
    objectid INTEGER PRIMARY KEY,
    contour FLOAT,
    density VARCHAR(50),
    orig_fid VARCHAR(50),
    shape_leng FLOAT,
    shape_area FLOAT,
    geom GEOMETRY(MULTIPOLYGON,4326)
);"""
)

#3 employees <csv>
conn.execute("""
DROP TABLE IF EXISTS employees CASCADE;
CREATE TABLE employees (
   industry_code VARCHAR(50),
   industry_name VARCHAR(100),
   sa2_code INTEGER,
   sa2_name VARCHAR(100),
```

```
    "non_employing" INTEGER,
    "1_to_4_employees" INTEGER,
    "5_to_19_employees" INTEGER,
    "20_to_199_employees" INTEGER,
    "200_to_more_employees" INTEGER,
    total INTEGER
);""")
```

Out[29]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc895a4820>
```

In [30]:
```
# INSERT DATA
# 1 toilet
toilet.columns = map(str.lower, toilet.columns)
toilet.to_sql('toilet', conn, if_exists='append', index=False, dtype={'geom': Geometry(geo

# 2 crime
crime.columns = map(str.lower, crime.columns)
crime.to_sql("crime", conn, if_exists='append', index=False, dtype={'geom': Geometry('MULT

#3 employees
employees.columns = map(str.lower, employees.columns)
employees.to_sql("employees", con=conn, if_exists='append', index=False)
```

In [31]:
```
# CREATE INDEX
conn.execute("CREATE INDEX IF NOT EXISTS tid ON toilet USING GIST (geom)")
conn.execute("CREATE INDEX IF NOT EXISTS cgid ON crime USING GIST (geom)")
conn.execute("CREATE INDEX IF NOT EXISTS eid ON employees(sa2_code)")
```

Out[31]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc895336d0>
```

3.3. Added score

In [32]:
```
# 1.1 TOILET TABLE +score
conn.execute("""
DROP VIEW IF EXISTS toilet_1 CASCADE;
CREATE VIEW toilet_1 AS
    (SELECT sa2_code21, COUNT(facilityid) AS toilet, area
        FROM toilet RIGHT JOIN sa100 ON ST_Contains(sa100.geom, toilet.geom)
        GROUP BY sa2_code21, area
        ORDER BY sa2_code21)""")
# 1.2 TOILET SCORE
conn.execute("""
DROP VIEW IF EXISTS toilet_2 CASCADE;
CREATE VIEW toilet_2 AS
    (SELECT sa2_code21, CAST(toilet AS FLOAT)/area AS toilet_score
     FROM toilet_1)""")
# 1.3 TOILET ZSCORE
conn.execute("""
DROP VIEW IF EXISTS toilet_3 CASCADE;
CREATE VIEW toilet_3 AS
    (WITH t3 AS
        (SELECT AVG(toilet_score) AS mean, stddev(toilet_score) AS sd
         FROM toilet_2)
     SELECT sa2_code21, (toilet_score - t3.mean) / t3.sd AS z_toilet
     FROM t3, toilet_2)""")
```

Out[32]:
```
<sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc80c0f640>
```

In [33]:
```
# 2.1 CRIME TABLE   -score
```

```
conn.execute("""
DROP VIEW IF EXISTS crime_1 CASCADE;
CREATE VIEW crime_1 AS
    (SELECT sa2_code21, SUM(shape_area) AS crime, area
        FROM sa100 LEFT JOIN crime ON ST_Overlaps(sa100.geom, crime.geom)
        GROUP BY sa2_code21, area
        ORDER BY sa2_code21)""")
# 2.2 CRIME SCORE
conn.execute("""
DROP VIEW IF EXISTS crime_2 CASCADE;
CREATE VIEW crime_2 AS
    (SELECT sa2_code21,
         CASE
            WHEN crime IS NULL THEN  CAST(0 AS FLOAT)/area
            ELSE CAST(crime AS FLOAT)/area
         END AS "crime_score"
     FROM crime_1)""")
# 2.3 CRIME ZSCORE
conn.execute("""
DROP VIEW IF EXISTS crime_3 CASCADE;
CREATE VIEW crime_3 AS
    (WITH c3 AS
        (SELECT AVG(crime_score) AS mean, stddev(crime_score) AS sd
         FROM crime_2)
     SELECT sa2_code21, (crime_score - c3.mean) / c3.sd AS z_crime
     FROM c3, crime_2)""")
```

Out[33]: <sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc97b3d1f0>

In [34]:

```
# 3.1 CONSTRUCTION_NON_EMPLOYING TABLE  -score
# non employing construction industry per 1000 people
# -score, cuz constraction need large amount of human resources
conn.execute("""
DROP VIEW IF EXISTS construct_1 CASCADE;
CREATE VIEW construct_1 AS
    (SELECT sa2_code21, non_employing AS construct, tot_p
        FROM employees RIGHT JOIN sa100 ON (sa2_code21 = sa2_code)
        WHERE industry_name = 'Construction'
        GROUP BY sa2_code21, construct, tot_p
        ORDER BY sa2_code21)""")
# 3.2 CONSTRUCTION_NON_EMPLOYING SCORE
conn.execute("""
DROP VIEW IF EXISTS construct_2 CASCADE;
CREATE VIEW construct_2 AS
    (SELECT sa2_code21,
     CAST(construct AS FLOAT)/CAST(tot_p AS FLOAT)*1000 AS construct_score
     FROM construct_1)""")
# 3.2 CONSTRUCTION_NON_EMPLOYING ZSCORE
conn.execute("""
DROP VIEW IF EXISTS construct_3 CASCADE;
CREATE VIEW construct_3 AS
    (WITH e3 AS
        (SELECT AVG(construct_score) AS mean, stddev(construct_score) AS sd
         FROM construct_2)
     SELECT sa2_code21, (construct_score - e3.mean) / e3.sd AS z_construct
     FROM e3, construct_2)""")
```

Out[34]: <sqlalchemy.engine.cursor.LegacyCursorResult at 0x1dc97b3d910>

3.4. New sigmoid

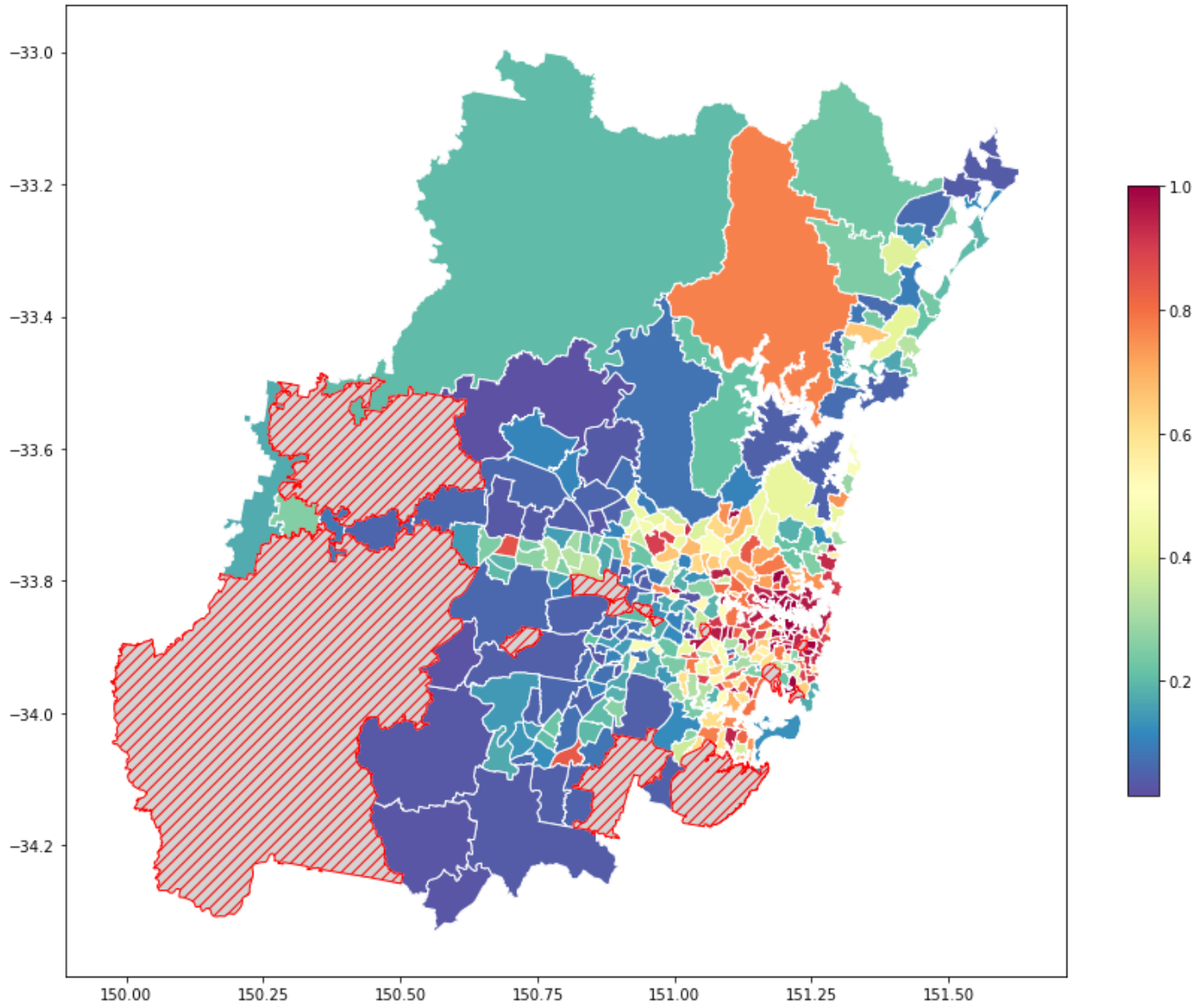
In [35]:

```
# NEW SIGMOID
```

```
conn.execute("""
DROP VIEW IF EXISTS sigmoid_2 CASCADE;
CREATE VIEW sigmoid_2 AS
    (SELECT sa2_code21, sa2_name21, area, r.z_retail, h.z_health, s.z_stops,
           p.z_polls, sch.z_schools, t.z_toilet, c.z_crime, e.z_construct, geom,
           1/(1 + EXP(-(z_retail + z_health + z_stops + z_polls + z_schools
           + z_toilet - z_crime - z_construct))) as "well_resourced_score"
     FROM sa100 JOIN retail_3 r USING (sa2_code21)
                JOIN health_3 h USING (sa2_code21)
                JOIN stops_3 s USING (sa2_code21)
                JOIN polls_3 p USING (sa2_code21)
                JOIN schools_3 sch USING (sa2_code21)
                JOIN toilet_3 t USING (sa2_code21)
                JOIN crime_3 c USING (sa2_code21)
                JOIN construct_3 e USING (sa2_code21))""")
sa2_sigmoid_new = gpd.read_postgis("SELECT * FROM sigmoid_2", conn)
```

In [36]:

```
# TOTAL <WELL_SOURCED> SCORE
sa2_sigmoid_new.plot(
    column="well_resourced_score",
    legend=True,
    figsize=(15, 15),
    missing_kwds={"color": "lightgrey", "edgecolor": "red",
                  "hatch": "///", "label": "pop <100"},
    cmap='Spectral_r',
    edgecolor = 'white',
    legend_kwds={'shrink': 0.5}
);
```



3.5. Corr

In [37]:
```
merged_income = query(conn, """SELECT * FROM sigmoid_2 LEFT JOIN income ON (sa2_code21 = s
merged_income[['well_resourced_score' ,'median_income']].corr()
```

Out[37]:

	well_resourced_score	median_income
well_resourced_score	1.000000	0.504201
median_income	0.504201	1.000000

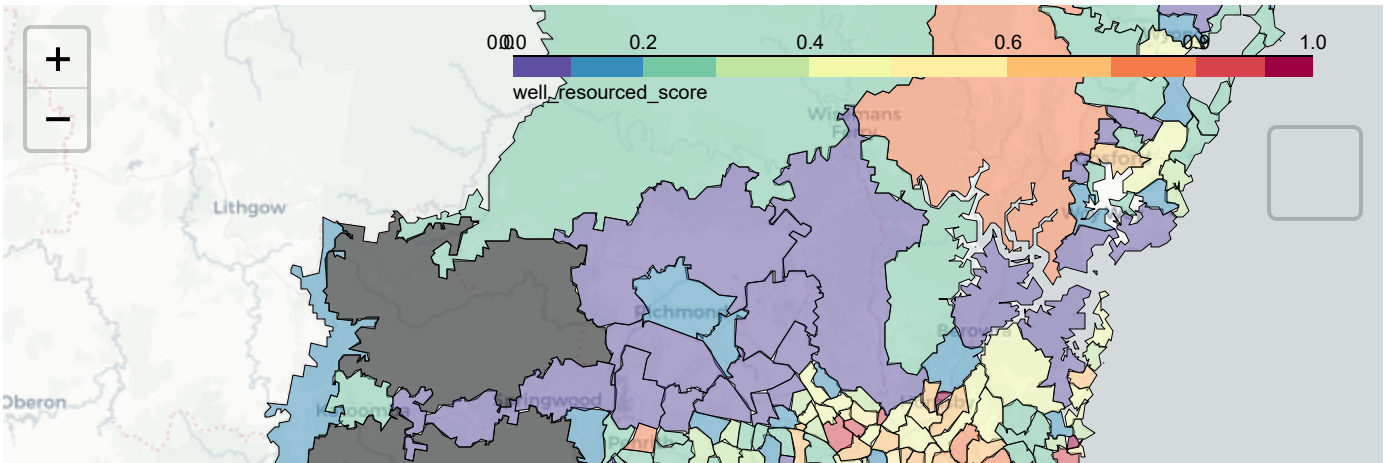
In [38]:
```
sns.regplot(x='median_income', y='well_resourced_score', data=merged_income)
plt.title('Correlation between Median Income and Well Resourced Score')
plt.show()

# Based on the plot, the median_income is positively related to well_resourced.
# As median_income increases, the well_resourced score will also increase accordingly.
```

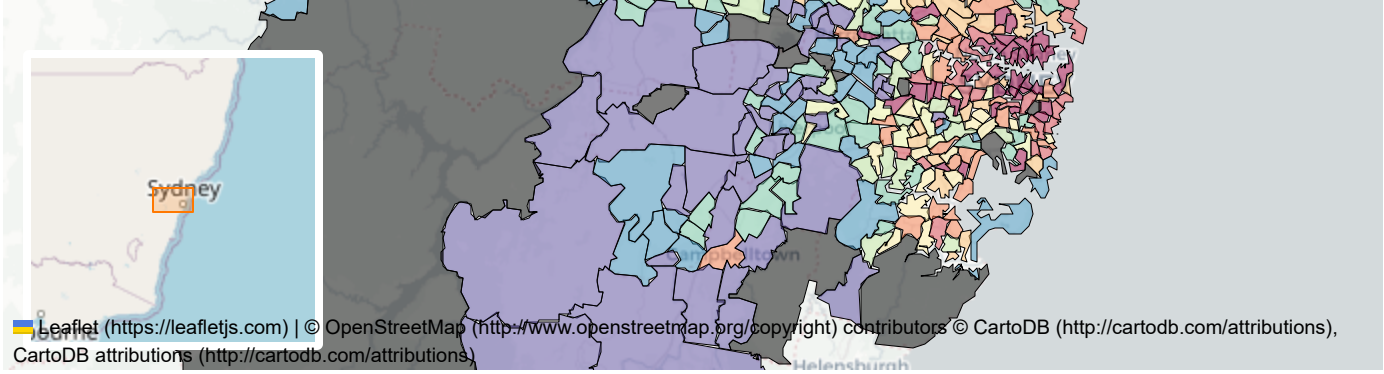
Extra: Interactive Map

In [39]:

```
# conda install -c conda-forge folium
import folium
from folium.plugins import MiniMap
from folium.plugins import GroupedLayerControl

m = folium.Map(location=(-33.7028, 150.7214),  tiles="CartoDB positron", zoom_start=9, cor
m1 = sa2_sigmoid_new.explore(
    m=m,
    show=True,
    column="well_resourced_score",
    scheme="naturalbreaks",  # use mapclassify's natural breaks scheme
    name='well_resourced_score',
    legend=True,  # show legend
    k=10,  # use 10 bins
    tooltip=False,  # hide tooltip
    popup=["sa2_name21", "sa2_code21", "area", 'well_resourced_score'],  # show popup (on-
    style_kwds=dict(weight=0.5, color="black"),
    cmap="Spectral_r",
    tiles="CartoDB positron",
)

minimap = MiniMap(toggle_display=True, position='bottomleft')
minimap.add_to(m)
folium.TileLayer('cartodbdark_matter').add_to(m)

folium.LayerControl().add_to(m)
m
```

Out[39]:





In [40]:

```
# please check the interactive map in current working dir
m.save('interactive_cc09_g2.html')
```

In [42]:

```
# conn.close()
# db.dispose()
```