DATA2001 ASSIGNMENT DOCUMENTATION

Dataset Description

Dataset Overview

- **Neighbourhoods**

  Census data on neighbourhoods in Greater Sydney. This includes land area, population, and economic information about the population on the certain area.
- **StatisticalAreas**

  Area identifier and its corrosponding parent area identifier.
- **BusinessStats**

  Business information on areas in Greater Sydney.
- **RFSNSW_BFPL**

  Geometric information of areas in Greater Sydney. This includes its area, vegetation category of the area, and the geometric position of the area.
- **SA2_2016_AUST**

  SA2 boundary data in Greater Sydney.

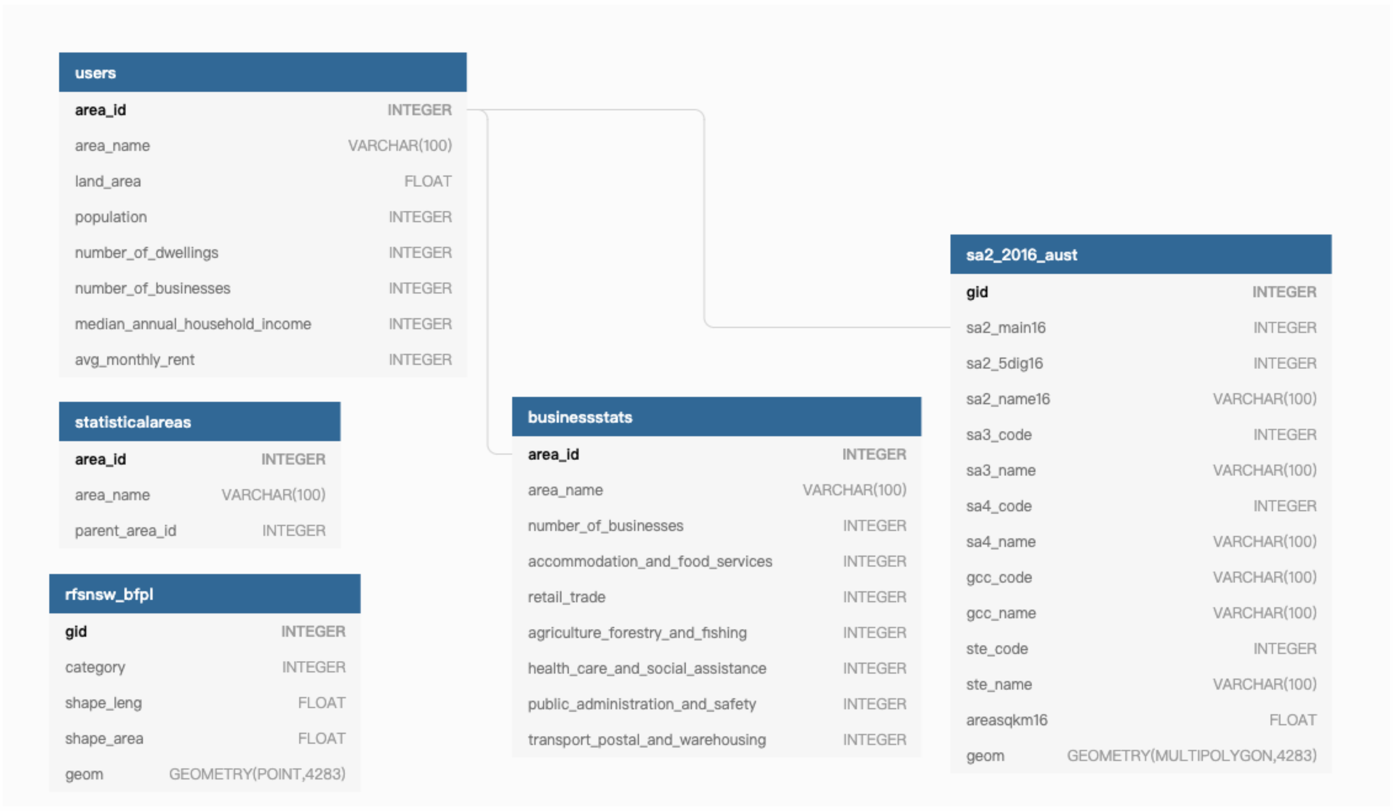
Dataset Cleaning

- **Neighbourhoods**
  - Through inspecting the dataset, the original data contains `NA` value. In order to ensure the validity of further calculations and processing, we need to neglect any rows in Neighbourhoods that contains one or more `NA` values. Use `dropna` to drop the rows containing `NA` value.
  - Some columns containing numeric data are of type `float`. However, by inspection, the numbers after the decimal point are zeros so it is deduced that these decimals are kept for precision. Hence, for the convenience of later processing, convert them to `int`.
  - See **Appendix 1a** for Neighbourhoods data cleaning code.
- **StatisticalAreas**
  - Through exploring the data set, it is found that there are some rows containing duplicated area identifiers. Since we intend to make `area_id` primary key for the table `statisticalareas`, it is neccessary to drop the rows with duplicated `area_id`.

    - See **Appendix 1b** for StatisticalAreas data cleaning code.
- **BusinessStats**
    - By comparing BusinessStats against Neighbourhoods, it is found that the `area_id` in some of rows in BusinessStats does not appear in Neighbourhoods. Hence, use a loop to iterate every row in BusinessStats, and if the row contains an `area_id` that is not found in Neighbourhoods, then drop it.
    - See **Appendix 1c** for BusinessStats data cleaning code.
- **RFSNSW_BFPL**
    - In the original dataset, the column `geometry` is of type `POINT`. Transform it to WKT values. This will be convenient for further analysis.
    - Change some of the orginal column names to lowercase in order to satisfy the requirements of assignment guidlines.
    - See **Appendix 1d** for RFSNSW_BFPL data cleaning code.
- **SA2_2016_AUST**
    - Some of columns are of type `object`, but the values are actually strings of integers. Hence, convert them to `Integer`.
    - Change some of the orginal column names to lowercase in order to satisfy the requirements of assignment guidlines.
    - Rename some of the columns to the names specified by the assignment guidlines.
    - By inspection, some of the rows contain `NA` values. Hence, drop those rows with one or more `NA` values.
    - `sa2_main16` in `SA2_2016_AUST` is corrosponding to `area_id` in `Neighbourhoods`. However, it is found that some of the rows in this table having an `area_id` that does not appear in Neighbourhoods. Hence, use a loop to iterate every row in SA2_2016_AUST, and if the row contains a `sa2_main16` that is not found in Neighbourhoods, then drop it.
    - See **Appendix 1e** for SA2_2016_AUST data cleaning code.

Database Description

Database Schema Diagram



users	
area_id	INTEGER
area_name	VARCHAR(100)
land_area	FLOAT
population	INTEGER
number_of_dwellings	INTEGER
number_of_businesses	INTEGER
median_annual_household_income	INTEGER
avg_monthly_rent	INTEGER

statisticalareas	
area_id	INTEGER
area_name	VARCHAR(100)
parent_area_id	INTEGER

rfsnsw_bfpl	
gid	INTEGER
category	INTEGER
shape_leng	FLOAT
shape_area	FLOAT
geom	GEOMETRY(POINT,4283)

businessstats	
area_id	INTEGER
area_name	VARCHAR(100)
number_of_businesses	INTEGER
accommodation_and_food_services	INTEGER
retail_trade	INTEGER
agriculture_forestry_and_fishing	INTEGER
health_care_and_social_assistance	INTEGER
public_administration_and_safety	INTEGER
transport_postal_and_warehousing	INTEGER

sa2_2016_aust	
gid	INTEGER
sa2_main16	INTEGER
sa2_5dig16	INTEGER
sa2_name16	VARCHAR(100)
sa3_code	INTEGER
sa3_name	VARCHAR(100)
sa4_code	INTEGER
sa4_name	VARCHAR(100)
gcc_code	VARCHAR(100)
gcc_name	VARCHAR(100)
ste_code	INTEGER
ste_name	VARCHAR(100)
areasqkm16	FLOAT
geom	GEOMETRY(MULTIPOLYGON,4283)

- **neighbourhoods**
  - `area_id` is chosen to be the primary key for this table as it is a unique identifier that can be used to differentiate between rows. And `neighbourhoods` is the main table that can be integrate and processed with other tables, so `area_id` can be set as foreign keys for other tables. See **Appendix 2.1.a** for the realtion schema creation code of this table.
  - Create index on neighbourhood`'area_id` since it is a primary key; furthermore, the table neighbourhoods is the main table so it will be queried frequently. Hence, adding an index will speed up queries. See **Appendix 2.1.b** for the index creation of this table.

- **businessstats**
  - `area_id` is selected as both the primary and foreign key for this table. As a primary key, it is used to distinguished between rows. As a foreign key, it references the column `area_id` in the table `neighbourhoods` so that the two tables are linked together. This ensures referential integrity of the data across the two tables and reject invalid entries of data (i.e. if an `area_id` that does not appear in `neighbourhoods` is inserted into `businessstats` , then this `area_id` is considered to be an invalid entry) during insertions or updates. See **Appendix 2.2.a** for the realtion schema creation code of this table.
  - Create index on the column `area_id` of the tables `businessstats` since it is a foreign key of the table. In addition, the data in this table will be frequently quried for the purpose of analysing the relation between fire risk score and economic information of the population, so creating an index will dramtically improve its performance. See **Appendix 2.2.b** for the index creation of this table.

- **sa2_2016_aust**
  - The column `gid` is chosen as the primary key and the column `sa2_main_16` is chosen as the foreign key for this table. The column `gid` is actually the index auto-generated by SQL that is used to distinguish between rows. The column `sa2_main_16` contains the same information as the column `area_id` in the table `neighbourhoods`. Hence, it is used to link the two tables together and ensure no invalid entry is inserted or updated in the table `sa2_2016_aust`. See **Appendix 2.3.a** for the realtion schema creation code of this table.
  - There are two indexes for this table:
    - Create an index on `sa2_main16` since it is a foreign key. See **Appendix 2.3.b** for the index creation of this table.
    - Create a spatial index on the column `geom` since it will be spatially joined with the table `rfsnsw_bfpl`. This will speed up both the join and queries. See **Appendix 2.3.c** for the index creation of this table.
- **statisticalareas**
  - The column `area_id` is selected as the primary key to differentiate rows. See **Appendix 2.4.a** for the realtion schema creation code of this table.
- **rfsnsw_bfpl**
  - The column `gid` is selected as the primary key to differentiate rows. It is identical to the index auto-generated by SQL. See **Appendix 2.5.a** for the index creation of this table.
  - Create a spatial index on the column `geom` since the table `sa2_2016_aust` will spatially join with this table using the index created by `geom`. See **Appendix 2.5.b** for the index creation of this table.

Fire Risk Score Analysis

Fire Risk Computation

See **Appendix 3** for the fire risk score computation code.

The formula used to calculate the fire risk score is

$$fire\ risk = S(z(population\ density) + z(dwelling\ and\ business\ density) + z(bfpl\ density) - z(assistive\ service\ density)$$

which is identical to the one given in the assignment guideline.

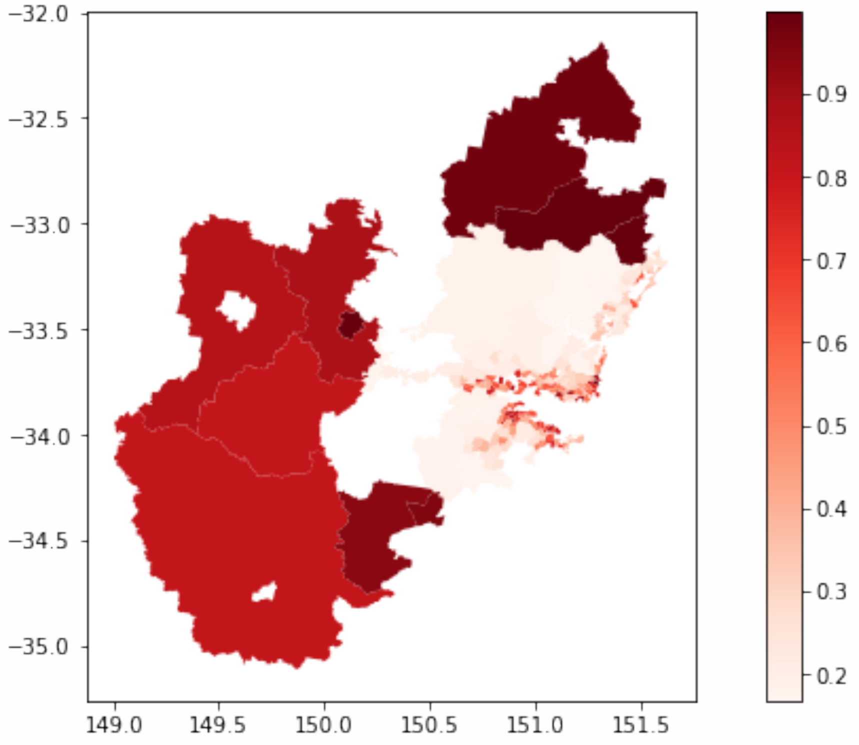
1. Join four tables `neighbourhoods`, `businessstats`, `sa2_2016_aust`, `rfsnsw_bfpl`.
2. Traverse the column `area_id` and calculate their densities as following.
3. Calculate densities for each `area_id` and store them into separate list.
   - **population density**

     Divide the column `population` by the column `land_area`.
   - **dewelling and business density**

     Take the sum of the column `number_of_dwellings` and the column `number_of_businesses` and divide the sum by `land_area`.
   - **assistive service density**

     Assistive service includes the column `health_care_and_social_assistance` and the column `public_administration_and_safety`. Take the sum of these two columns and divide the sum by `land_area`.
   - **bfpl density**

     The column `category` represents the vegetation categories in the certain areas in Greater Sydney. There are three categories: 1, 2 and 3 in an order of decreasing fire risk. Hence, if we treat these three categories as numeric values, we can divide the column `shape_area` by the column `category`. The result will follow such order that, say if there are two regions of the same area, then the region with the higher risk vegetation will have a larger result, and the region with the lower risk vegetation will have a smaller result. This is reasonable and logical in terms of computing the fire risk score. Then, take the sum of the results obtained from a certain area. Divide the sum by `land_area`.
4. Calculate the densities' average and standard deviation.
5. Calculate the z score using the average and the standard deviation.
6. Calculate the fire risk score using the formula given in the assignment guidelines.
7. Visualize the results onto a map.



Fire Risk Analysis

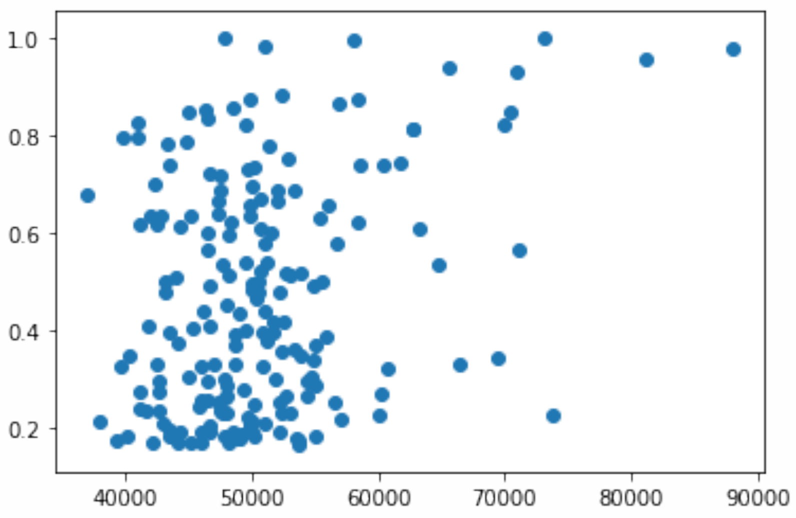
In the graph above, the horizontal axis represents longtitude and the vertical axis represents latitude, and the colours represent the fire risk score. The regions in deeper red will have a higher fire risk score and the regions in lighter red will have a lower fire risk. Hence, the region boundaried by the longtitude [150.5 to 151.5] and the latitude [-33.0 to -32.2] is the most risky area. The region boundaried by the longtitude [150.0 to 151.6] and the latitude [--34.7 to -33.0] is the least risky area. The rest of the areas are moderately risky.

Correlation Analysis

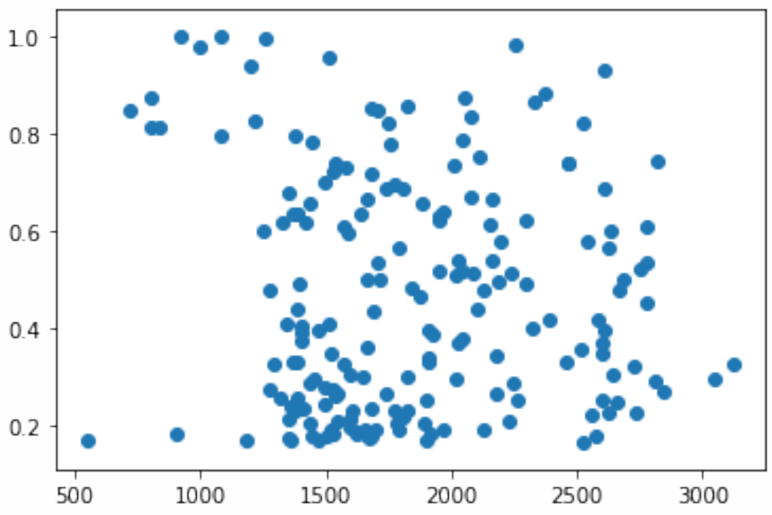
See **Appendix 4** for correlation computation code.

It is investigated how the fire risk score correlate to both the median household incomes and the rental prices of each region.

1. visualize their correlations using scatter plots.
   - The correlation between the score and the income. The horizontal axis is the median household incomes of each region and the vertical axis is the fire risk score.



- The correlation between the score and the rent. The horizontal axis is the rental prices of each region and the vertical axis is the fire risk score.



For the income graph, the points have slightly increasing trend which indicates only a low linear relationship between the income and the score. From the rent graph, the points are randomly scattered and do not have an apparent pattern. Hence, there is no linear correlation between the score and the rent.

2. Use Pearson's correlation to test their linear correlation.

- Pearson's correlation coefficient is the covariance of the two variables divided by the product of their standard deviations. It is commonly represented by the Greek letter ρ (rho). Given a pair of random variables (X, Y), the formula for ρ is:

$$\rho_{X,Y} = \frac{\operatorname{cov}(X, Y)}{\sigma_X \sigma_Y}$$

Reference: Pearson correlation coefficient

- Correlation results
  - Using Pearson's correlation formula, the result for the correaltion between the score and the income is 0.29455856183304896 which indicates they are only weakly correlated. The result for the correaltion between the score and the rent is -0.06830049136978633 which is close to zero indicating no linear correlation between them.

Appendix

1. Dataset Description

a.

```
"""
    clean and load neighbourhoods data
"""
neighbourhoods_data = pd.read_csv('Neighbourhoods.csv', thousands=",")
neighbourhoods_data2 = neighbourhoods_data.copy()
# drop the rows with NA value
neighbourhoods_data2 = neighbourhoods_data2.dropna()

# convert float to int
neighbourhoods_data2['population'] = neighbourhoods_data2['population'].astype(int)
neighbourhoods_data2['number_of_businesses'] =
neighbourhoods_data2['number_of_businesses'].astype(int)
neighbourhoods_data2['median_annual_household_income'] =
neighbourhoods_data2['median_annual_household_income'].astype(int)
neighbourhoods_data2['avg_monthly_rent'] =
neighbourhoods_data2['avg_monthly_rent'].astype(int)

neighbourhoods_data2.to_sql('neighbourhoods', con=conn, if_exists='append',index=False)
```

b.

```
"""
    clean and load statisticalareas data
"""
statisticalareas_data = pd.read_csv('StatisticalAreas.csv')
statisticalareas_data2 = statisticalareas_data.copy()
# drop the rows with same area_id
statisticalareas_data2.drop_duplicates(subset=['area_id'], keep='first', inplace=True)
statisticalareas_data2.to_sql('statisticalareas', con=conn, if_exists='append',index=False)
```

c.

```
# store all area_id in neighbourhoods_data2 into a list
pk_area_id = []
for index, row in neighbourhoods_data2.iterrows():
    pk_area_id.append(row['area_id'])

businessstats_data = pd.read_csv('BusinessStats.csv')
businessstats_data2 = businessstats_data.copy()
rows_delete = []
```

```python
# remove the rows having the area_id that do not appear in neighbourhoods
for index, row in businessstats_data2.iterrows():
    if not row['area_id'] in pk_area_id:
        rows_delete.append(index)
businessstats_data2 = businessstats_data2.drop(rows_delete)
businessstats_data2.to_sql('businessstats', con=conn, if_exists='append',index=False)
```

d.

```python
# Use GeoAlchemy's WKTElement to create a geom with SRID
def create_wkt_point_element(geom,srid):
    return WKTElement(geom.wkt, srid)

rfsnsw_bfpl_data2 = rfsnsw_bfpl_data.copy()
# change columns name to lower case
rfsnsw_bfpl_data2.columns = map(str.lower, rfsnsw_bfpl_data2.columns)
rfsnsw_bfpl_data2['geom'] = rfsnsw_bfpl_data2['geometry'].apply(lambda x:
create_wkt_point_element(x,4283))
rfsnsw_bfpl_data2["gid"] = rfsnsw_bfpl_data2.index
# delete the old column before insertion
rfsnsw_bfpl_data2 = rfsnsw_bfpl_data2.drop(columns="geometry")

rfsnsw_bfpl_data2.to_sql('rfsnsw_bfpl', conn, if_exists='append', index=False,
                         dtype={'geom': Geometry('POINT', 4283)})
```

e.

```python
# Use GeoAlchemy's WKTElement to create a geom with SRID
def create_wkt_element(geom,srid):
    if (geom.geom_type == 'Polygon'):
        geom = MultiPolygon([geom])
    return WKTElement(geom.wkt, srid)

sa2_2016_aust_data2 = sa2_2016_aust_data.copy()

# Convert object to int
sa2_2016_aust_data2['SA2_MAIN16'] = sa2_2016_aust_data2['SA2_MAIN16'].astype(int)

# Change column names to lower case
sa2_2016_aust_data2.columns = map(str.lower, sa2_2016_aust_data2.columns)

# Rename columns
sa2_2016_aust_data2.rename(columns={
    'sa3_code16': 'sa3_code',
    'sa3_name16': 'sa3_name',
    'sa4_code16': 'sa4_code',
    'sa4_name16': 'sa4_name',
    'gcc_code16': 'gcc_code',
    'gcc_name16': 'gcc_name',
    'ste_code16': 'ste_code',
    'ste_name16': 'ste_name',
}, inplace=True)

# drop rows with None value
sa2_2016_aust_data2 = sa2_2016_aust_data2.dropna()

rows_delete = []
# remove the rows having the area_id that do not appear on neighbourhoods
for index, row in sa2_2016_aust_data2.iterrows():
    if not row['sa2_main16'] in pk_area_id:
        rows_delete.append(index)
sa2_2016_aust_data2 = sa2_2016_aust_data2.drop(rows_delete)

sa2_2016_aust_data2["gid"] = sa2_2016_aust_data2.index

sa2_2016_aust_data2['geom'] = sa2_2016_aust_data2['geometry'].apply(lambda x:
create_wkt_element(x,4283))
#delete the old column before insertion
sa2_2016_aust_data2 = sa2_2016_aust_data2.drop(columns="geometry")

sa2_2016_aust_data2.to_sql('sa2_2016_aust', conn, if_exists='append', index=False,
                         dtype={'geom': Geometry('MULTIPOLYGON', 4283)})
```

2. Database Description

1.

a

```
conn.execute("DROP TABLE IF EXISTS neighbourhoods CASCADE")
neighbourhoos_schema = """
                         CREATE TABLE IF NOT EXISTS neighbourhoods (
                         area_id INTEGER PRIMARY KEY,
                         area_name VARCHAR(100),
                         land_area FLOAT,
                         population INTEGER,
                         number_of_dwellings INTEGER,
                         number_of_businesses INTEGER,
                         median_annual_household_income INTEGER,
                         avg_monthly_rent INTEGER
                   )"""
conn.execute(neighbourhoos_schema)
```

b

```
# create index on neighbourhoods's area_id since it is primary key
try:
    pd.read_sql_query("DROP INDEX IF EXISTS neig_idx;", conn)
except:
    pass

try:
    pd.read_sql_query("CREATE INDEX neig_idx ON neighbourhoods (area_id);", conn)
except:
    pass
```

2.

a

```
conn.execute("DROP TABLE IF EXISTS businessstats")
businessstats_schema = """
                         CREATE TABLE IF NOT EXISTS businessstats (
                         area_id INTEGER,
                         area_name VARCHAR(100),
                         number_of_businesses INTEGER,
                         accommodation_and_food_services INTEGER,
                         retail_trade INTEGER,
                         agriculture_forestry_and_fishing INTEGER,
                         health_care_and_social_assistance INTEGER,
                         public_administration_and_safety INTEGER,
                         transport_postal_and_warehousing INTEGER,

                         PRIMARY KEY(area_id),
                         CONSTRAINT fk_busi_neig
                         FOREIGN KEY(area_id)
                         REFERENCES neighbourhoods(area_id)
                   )"""
conn.execute(businessstats_schema)
```

b

```
try:
    pd.read_sql_query("DROP INDEX IF EXISTS busi_idx;", conn)
except:
    pass

try:
    pd.read_sql_query("CREATE INDEX busi_idx ON businessstats (area_id);", conn)
except:
    pass
```

3.

a

```
conn.execute("DROP TABLE IF EXISTS sa2_2016_aust")
sa2_2016_aust_schema = """CREATE TABLE IF NOT EXISTS sa2_2016_aust (
                         gid INTEGER PRIMARY KEY,
                         sa2_main16 INTEGER,
                         sa2_5dig16 INTEGER,
                         sa2_name16 VARCHAR(100),
                         sa3_code INTEGER,
                         sa3_name VARCHAR(100),
                         sa4_code INTEGER,
                         sa4_name VARCHAR(100),
                         gcc_code VARCHAR(100),
                         gcc_name VARCHAR(100),
                         ste_code INTEGER,
                         ste_name VARCHAR(100),
                         areasqkm16 FLOAT,
                         geom GEOMETRY(MULTIPOLYGON,4283),

                         CONSTRAINT fk_sa2_neig
                         FOREIGN KEY(sa2_main16)
                         REFERENCES neighbourhoods(area_id)
                   )"""
conn.execute(sa2_2016_aust_schema)
```

b

```
try:
    pd.read_sql_query("DROP INDEX IF EXISTS sa2_idx;", conn)
except:
    pass

try:
    pd.read_sql_query("CREATE INDEX sa2_idx ON sa2_2016_aust (sa2_main16);", conn)
except:
    pass
```

c

```
try:
    pd.read_sql_query("DROP INDEX IF EXISTS sa2_geom_idx;", conn)
except:
    pass

try:
    pd.read_sql_query("""
                CREATE INDEX sa2_geom_idx
                  ON sa2_2016_aust
                  USING GIST (geom);
                """, conn)
except:
    pass
```

4.

a.

```
conn.execute("DROP TABLE IF EXISTS statisticalareas")
statisticalareas_schema = """
                         CREATE TABLE IF NOT EXISTS statisticalareas (
                         area_id INTEGER PRIMARY KEY,
                         area_name VARCHAR(100),
                         parent_area_id INTEGER
                   )"""
conn.execute(statisticalareas_schema)
```

5.

a

```
conn.execute("DROP TABLE IF EXISTS rfsnsw_bfpl")
rfsnsw_bfpl_schema = """CREATE TABLE IF NOT EXISTS rfsnsw_bfpl (
                         gid INTEGER PRIMARY KEY,
                         category INTEGER,
                         shape_leng FLOAT,
                         shape_area FLOAT,
                         geom GEOMETRY(POINT,4283)
                   )"""
conn.execute(rfsnsw_bfpl_schema)
```

b

```
try:
    pd.read_sql_query("DROP INDEX IF EXISTS rfsnsw_bfpl_geom_idx;", conn)
except:
    pass

#create spatial index on rfsnsw_bfpl
try:
    pd.read_sql_query("""
                CREATE INDEX rfsnsw_bfpl_geom_idx
                  ON rfsnsw_bfpl
                  USING GIST (geom);
                """, conn)
except:
    pass
```

3.

```
import matplotlib as mpl
import matplotlib.pyplot as plt

# Join four tables neighbourhoods, businessstats, sa2_2016_aust, rfsnsw_bfpl
data = pd.read_sql_query("""
                SELECT neighbourhoods.area_id, land_area, population, number_of_dwellings,
neighbourhoods.number_of_businesses,
                health_care_and_social_assistance, public_administration_and_safety, category,
shape_area, sa2_2016_aust.geom AS sa2_geom,
                rfsnsw_bfpl.geom AS bfpl_geom
                FROM neighbourhoods, businessstats, sa2_2016_aust, rfsnsw_bfpl
                WHERE neighbourhoods.area_id = businessstats.area_id AND
                    neighbourhoods.area_id = sa2_2016_aust.sa2_main16 AND
                    ST_Contains(sa2_2016_aust.geom, rfsnsw_bfpl.geom)
                """, conn)

data.drop_duplicates(inplace=True)

# ----------------------------- Calculating densities ---------------------------

# all the area_id
neigh_area_ids = set(data['area_id'])
# map area_id to a list contains this neighbourhood's population density, dwelling_business
density, bfpl density, assistive service density
id_data = {}
# store all neighbourhoods's data to calculate avg and std
population_densities = []
dwelling_business_densities = []
bfpl_densities = []
assistive_densities = []

# Traverse all the area_id and calculte their densities
```

```
for i in neigh_area_ids:
    temp = []
    df = data[data['area_id'] == i]
    # population density
    pd = df['population'].iloc[0] / df['land_area'].iloc[0]
    population_densities.append(pd)
    temp.append(pd)
    # dwelling_business density
    dbd = (df['number_of_dwellings'].iloc[0] + df['number_of_businesses'].iloc[0]) /
df['land_area'].iloc[0]
    dwelling_business_densities.append(dbd)
    temp.append(dbd)
    # bfpl density
    # area and category is combined using area/category since 1 is the highest risk
    aux = df['shape_area'] / df['category']
    bd = aux.sum() / df['land_area'].iloc[0]
    bfpl_densities.append(bd)
    temp.append(bd)
    # assistive service density
    ad = (df['health_care_and_social_assistance'].iloc[0] +
df['public_administration_and_safety'].iloc[0]) / df['land_area'].iloc[0]
    assistive_densities.append(ad)
    temp.append(ad)
    id_data[i] = temp

# ------------------------------------------------------------------------------


# ----------------------------- Calculating avg and std ---------------------------
population_density_avg = sum(population_densities) / len(population_densities)
population_density_std = statistics.stdev(population_densities)

dwelling_business_density_avg = sum(dwelling_business_densities) /
len(dwelling_business_densities)
dwelling_business_density_std = statistics.stdev(dwelling_business_densities)

bfpl_density_avg = sum(bfpl_densities) / len(bfpl_densities)
bfpl_density_std = statistics.stdev(bfpl_densities)

assistive_density_avg = sum(assistive_densities) / len(assistive_densities)
assistive_density_std = statistics.stdev(assistive_densities)
# ------------------------------------------------------------------------------


# -------------------- Calculating z score and fire risk score --------------------
def z_population_density(x):
    return (x - population_density_avg) / population_density_std

def z_dwelling_business_density(x):
    return (x - dwelling_business_density_avg) / dwelling_business_density_std

def z_bfpl_density(x):
    return (x - bfpl_density_avg) / bfpl_density_std

def z_assistive_density(x):
    return (x - assistive_density_avg) / assistive_density_std

def s(x):
    return 1 / (1 + math.exp(-x))
```

```python
sa2_2016_aust_data3 = sa2_2016_aust_data.copy()
sa2_2016_aust_data3['SA2_MAIN16'] = sa2_2016_aust_data3['SA2_MAIN16'].astype(int)

neig_geoms = []
fire_risks = []
# calculate fire risk of each neighbourhoods
for i in neigh_area_ids:
    d = id_data[i]
    fr = s(z_population_density(d[0]) + z_dwelling_business_density(d[1]) +
z_bfpl_density(d[2]) - z_assistive_density(d[3]))
    g = sa2_2016_aust_data3[sa2_2016_aust_data3['SA2_MAIN16'] == i]['geometry'].iloc[0]
    neig_geoms.append(g)
    fire_risks.append(fr)
# -------------------------------------------------------------------------------


# ----------------------------- Visualize onto a map ----------------------------
%matplotlib inline
map_df = pd.DataFrame(
    {'fire_risk': fire_risks,
     'geom': neig_geoms
    })
gpd_map_df = gpd.GeoDataFrame(map_df, geometry='geom')

fig, ax = plt.subplots(1, figsize=(12,6))
gpd_map_df.plot(ax=ax, cmap='Reds', column='fire_risk', legend=True)
# -------------------------------------------------------------------------------
```

4.

```python
import matplotlib as mpl
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from scipy.stats import spearmanr

neig_income = []
neig_rent = []
for i in neigh_area_ids:
    d = id_data[i]
    income = neighbourhoods_data2[neighbourhoods_data2['area_id'] == i]
['median_annual_household_income'].iloc[0]
    rent = neighbourhoods_data2[neighbourhoods_data2['area_id'] == i]
['avg_monthly_rent'].iloc[0]
    neig_income.append(income)
    neig_rent.append(rent)


# ------------------------- income -------------------------
# scatter diagram
plt.scatter(neig_income, fire_risks)
plt.show()

# Pearson
corr, _ = pearsonr(neig_income, fire_risks)
print("Pearson's correlation: {}".format(corr))
# Pearson's correlation: 0.29455856183304896

# Spearman
corr, _ = spearmanr(neig_income, fire_risks)
print("Spearman's correlation: {}".format(corr))
# Spearman's correlation: 0.1869587524477832

# ------------------------- rent -------------------------
# scatter diagram
plt.scatter(neig_rent, fire_risks)
plt.show()

# Pearson
corr, _ = pearsonr(neig_rent, fire_risks)
print("Pearson's correlation: {}".format(corr))
# Pearson's correlation: -0.06830049136978633

# Spearman
corr, _ = spearmanr(neig_rent, fire_risks)
print("Spearman's correlation: {}".format(corr))
# Spearman's correlation: 0.012667883796248901
```