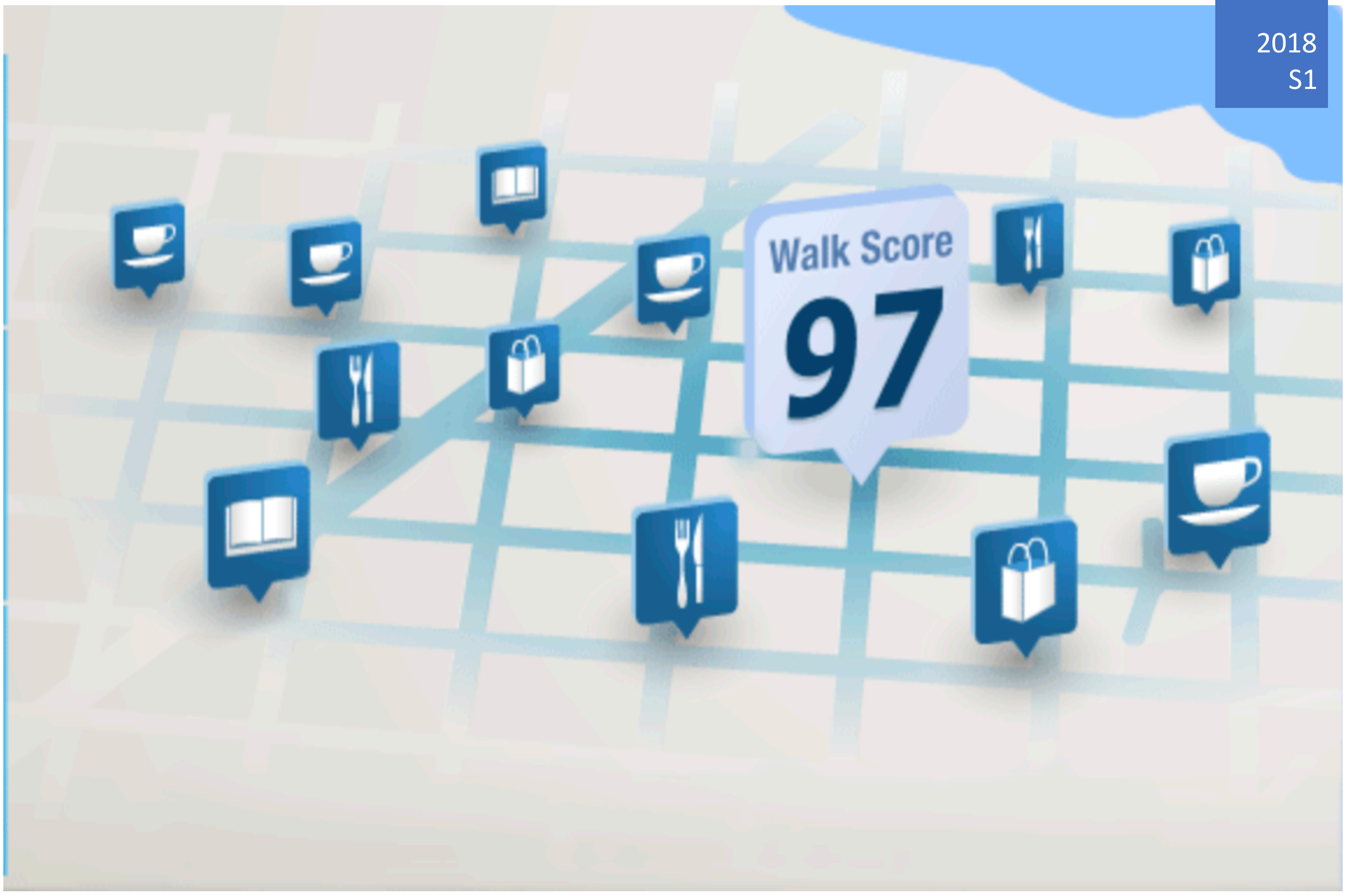
2018
S1



Sydney Walkability

DATA2001

YIRAN JING 460244129
HONGYU HE 460012791

Part 1: Dataset Description

To perform a data analysis of the walkability of 312 different neighbourhoods in Sydney, we mainly use four csv datasets of Statistical Area 2 (SA2), provided by Australia Bureau of Statistics (ABS), including *StatisticalAreas.csv, Neighbourhoods.csv, CensusStat.scv* and *BusinessStats.csv*. We also generate *Sydney2_2016_AUST.shp*, which including 312 areas in Sydney, based on *SA2_2016_AUST.shp* given by ABS.

To achieve 5D's measurements, in addition to *CarsharingPods.scv* given on Canvas, we add additional four datasets. We load *BusStops* table to DBMS by importing txt file from TransitFeeds website*,* and generate *TrainStation.csv, LibraryBookstore.csv* and *ParksPlayground.csv by web scraping tables* from websites (web links are given in the appendix).

- **Google Map API**

Besides the attributes given, the boundary of the neighbourhoods is added as an extra column in the *neighbourhoods* dataset (Approx. bounding box type scraping, see appendix code 4). And we add longitude and latitude of each record in *BusStops*, *TrainStation, LibraryBookstore* and *ParksPlayground* by Google Map API.

- **Precise spatial join with neighbourhood**

Car sharing pods dataset is added an extra column called area_id where those pods are located in, by performing the precise spatial join with Sydney2_2016_AUST.shp. We perform this by loop through a list of neighbourhoods (based on *shapely.geometry*, see appendix code 5 and spatial join_final.ipynb).

We generate the number of cars sharing pods for each neighbourhood by loop through car sharing pods (see appendix code 6 and exactly matching_final.ipynb). A similar method for *BusStops*, *TrainStation, LibraryBookstore* and *ParksPlayground* dataset.

- **Cleaning Data**

Before the analysis is conducted, data cleaning is performed as there are attributes where there were a few observations with missing values (such as the original BusinessStats.csv). An empty observation merely meant misrepresentation for that particular observation, and we replace the empty entry by 0. For instance, in BusinessStats dataset, missing entries occur along with a low number of total business and thus it is reasonable to infer that missing entries stands for misrepresentation of a generic type of business.

Part 2: Database Description

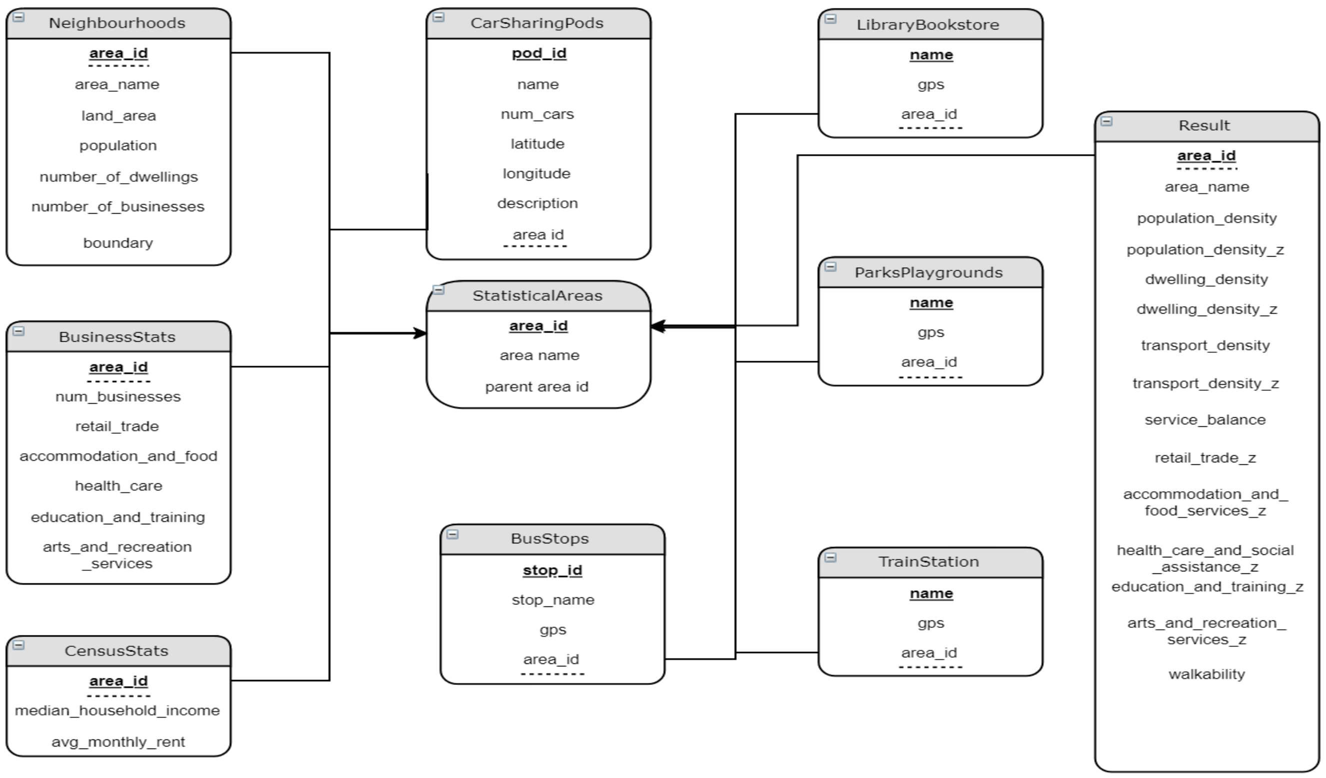
In order to speed up the queries on the result table, we introduce three indexes.

The first index is created on the *area_name* column from the *result* table. It is created because for users, it is possible that they want to know the walkability score for a specific area name by querying on an area_name.

The second index is created on the *walkability* column from *result* table. It is created because it is common for users to do a range query on walkability to see what the neighbourhoods are within a certain walkability scores range.

The third index is created using GIST on the *boundary* column from *neighbourhoods* table and *GPS* column from *carsharingpods* table. This index is useful when we spatial join neighbourhoods table and *carsharingpods* table. As it turns out, the performance is improved.

However, the improvement of efficiency is not significant. Due to the small size of table, the time spent on sequential scanning from database does not take large amount of time. In some cases, the time used to find entries using index takes longer than doing sequential scanning and system chooses to do sequential scanning to optimize query speed. That is, using index can speed up the query but performing index search requires time and once an index is found, fetching the data from database also takes time. And it turns out that the second index is not used by system and this is found by EXPLAIN statement at the start of a query.



Part 3: Walkability Analysis

$$\begin{aligned} ServiceBlance = {} & Z(retail\ trade\ density) + Z(accommodation\ and\ food\ density) \\ & + Z(healthcare\ and\ social\ assistance\ density) + Z(arts\ and\ recreation\ density) \end{aligned}$$

Where Z is z_score (assuming normal) and the density of selected business type is the number of business type divided by the total number of the business of each area (The detailed description of the meaning of each variable is included in appendix table 1).

Walkability calculation should also take people living, shopping, visiting and enjoying into consideration. We define $service\ balance$ by summing up z-scores of densities of four selected business types and give them equal weight. To cover design, diversity and destination accessibility of five D's, we use four additional datasets and add relevant new variables (number of busStop, trainStation, libraryBookstore, parkPlayground of each area) and rescale the number of bus stop by log transformation (plus one as some areas have no bus stop but log0 is undefined).

$$Walkability = 2 * Z(populationDensity) + 3 * Z(dwellingDensity) + 2 * Z(service\ balance) + 3 * Z(transportDensity) - 2 * Z(landArea) + 2 * \log(number\ of\ busStops + 1) + 0.01 * number\ of\ art\ and\ playground + 0.01 * number\ of\ library\ and\ bookstore + 0.01 * number\ of\ train\ station$$

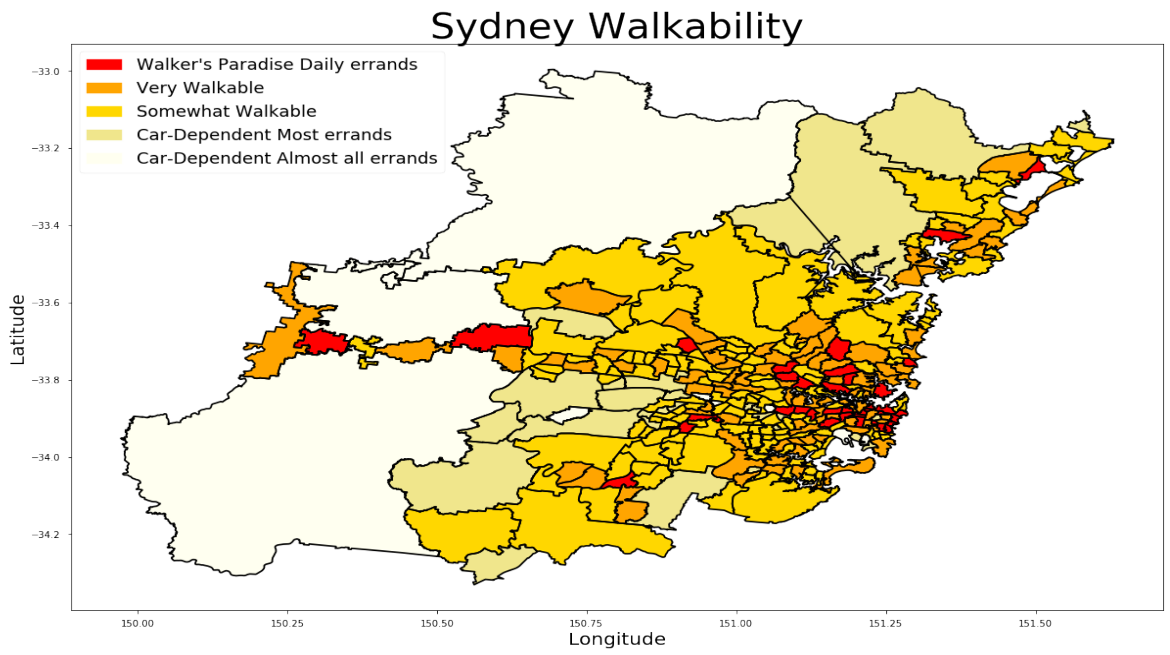
To rescale walkability score from 0 to 100:

$$Walkability\ score\ of\ area\ i = \frac{100 * (walkability_i - \min(walkability))}{\max(walkability) - \min\ (walkability)}$$

(the detailed calculation process/code shown in appendix code 1 and walkability score.ipynb)

The walkability scores we get are rounded to integer after calculation and are within the range 0 to 100 inclusively. We put higher weights for the z score of densities of population, dwelling and transport and service balance as they are the most important variables. And give a negative weight for z-score of land area, since the neighborhoods with quite large areas tend to have significant low walkability scores (such as blue mountain). The relevant calculation results saved in Walkability_final_result.csv and relevant table in DBMS. The graphical visualization is shown below. And we classify walkability scores into five classes (The description of each class in appendix table 2).

Figure 1 – walkability scores for 312 areas in Sydney

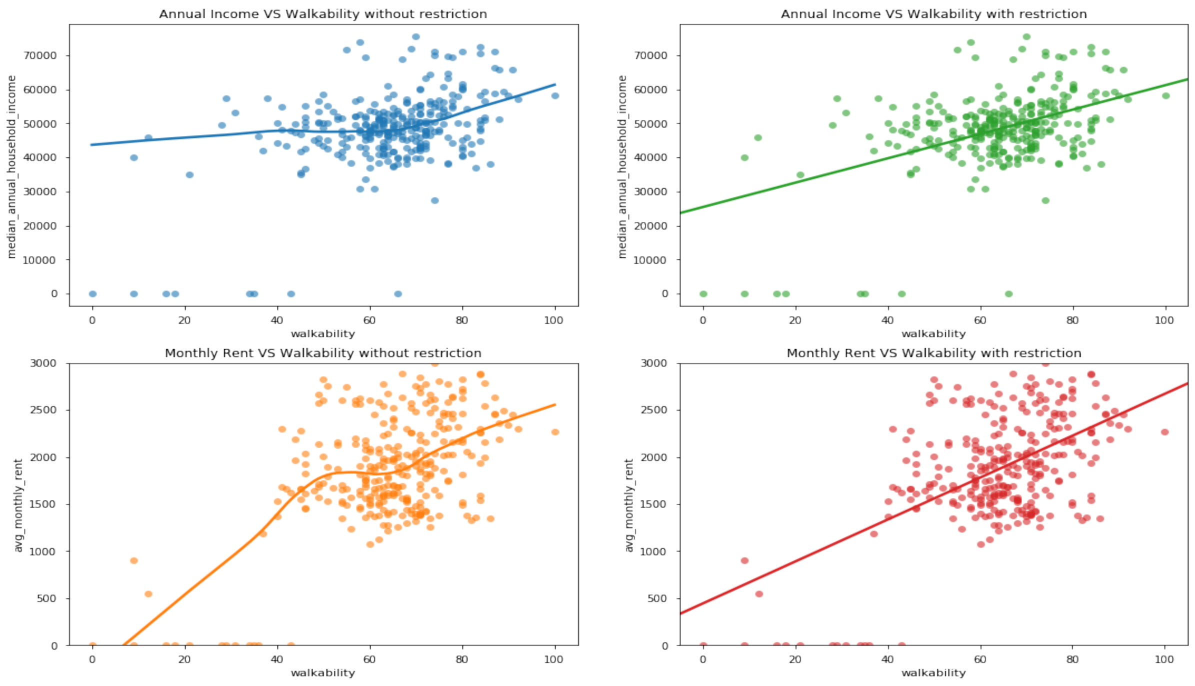


- Newtown - Camperdown – Darlington: Based on our calculation, its walkability score is 100, intuitively it is reasonable as it contains shopping centre, restaurants, schools, parks and bookstore, bustling with activity day and night.
- Blue Mountains South: Walkability is 0, which is also quite sensible as it is a mountainous area.

Part 4: Correlation Tests and Analysis

Plotting the median annual household income and the average monthly rent with walkability score, variables increase concurrently and we can verify moderate positive linear relationships between these two variables and the walkability score.

Figure 2 – the relationship between annual income and walkability; the relationship between average monthly rent and walkability



To verify the moderate positive linear relationships, we calculate the correlation value using correlation function of python dataframe (Appendix code 2 and Walkability score.ipynb).

Table 1 – the correlation between annual income and walkability; the correlation between average monthly rent and walkability

	Correlation value	**Coefficicent of determination**
Median annual income	0.464	0.215
Average monthly rent	0.543	0.3

Efficient size of the correlation: The coefficient of determination is the square of the coefficient. For the annual household income, the coefficient of determination is around 0.215, which means that 21.5% of the variation in mean can be predicted from the relationship between household income and walkability score. For the average monthly rent, the coefficient of determination is around 0.3, which means that 30% of the variation in monthly rent can be predicted from the relationship between monthly rent and walkability score.

Based on the plots and numerical value, we can say that median annual household income and walkability score tend to increase together. Similarly, monthly rent and walkability score also tend to increase together. We can expect that the area with higher walkability score tends to have higher median annual household income and higher average monthly rent, but without properly controlled experiments, the causal relationship cannot be determined.

Appendix

Table 1

Measure	Definition	Data Source
retail trade density	The number of retail trade divided by the total number of the business of that neighborhood	BusinessStats
accommodation and food density	The number of accommodation and food divided by the total number of the business of that neighborhood	BusinessStats
healthcare and social assistance density	Number of healthcare and social assistance divided by the total number of the business of that neighborhood	BusinessStats
arts and recreation density	The number of arts and recreation divided by the total number of the business of that neighborhood	BusinessStats

Appendix table 2

Class	Walkability Score
Walker's Paradise Daily errands	81-100
Very Walkable	70-80
Somewhat Walkable	50-69
Car-Dependent Most errands	31-49
Car-Dependent Almost all errands	0-30

Appendix code 1

```python
def z_score(df):
    df.columns = [x + "_zscore" for x in df.columns.tolist()]
    return ((df - df.mean())/df.std(ddof=0))
```

```python
def min_max_norm(dataset):

    if isinstance(dataset,list):
        norm_list = list()
        min_value = min(dataset)
        max_value = max(dataset)

        for value in dataset:
            tmp = (value - min_value)*100 / (max_value - min_value)
            norm_list.append(tmp)

    return norm_list
```

Appendix code 2

```python
calculate_balance()

# new column
data["walkability"] = 2*data["ZP"] + 3*data["ZD"] + 3*data["ZT"] + 2*data["buss_type_banace"] - 2*data['area_z_score']
data["walkability"] = data["walkability"] + 2*np.log(data['busstops_num']+1)
data["walkability"] = data["walkability"] + (data['librarybookstore_num']+data['parksplaygrounds_num'])/100
data["walkability"] = data["walkability"] + data['trainstation_num']/100

scaled_walkability = min_max_norm(list(data["walkability"])) # walkabilility should be integer

# round to Integer
for i in range(len(scaled_walkability)):
    scaled_walkability[i] = round(scaled_walkability[i])

data["walkability"]=scaled_walkability

corr_median_income = data_CensusStats["median_annual_household_income"].corr(data["walkability"])
corr_monthly_rent = data_CensusStats["avg_monthly_rent"].corr(data["walkability"])
print(corr_median_income)
print(corr_monthly_rent)
```

```
Please input weight for retail trade: 1
Please input weight for accommodation_and_food_services: 1
Please input weight for health_care_and_social_assistance: 1
Please input weight for education_and_training: 1
Please input weight for arts_and_recreation_services: 1
0.4638661046973436
0.5430275438601628
```

Appendix code 3

```python
def calculate_balance():
    w1 = float(input("Please input weight for retail trade: "))
    w2 = float(input("Please input weight for accommodation_and_food_services: "))
    w3 = float(input("Please input weight for health_care_and_social_assistance: "))
    w4 = float(input("Please input weight for education_and_training: "))
    w5 = float(input("Please input weight for arts_and_recreation_services: "))
    data["buss_type_banace"]=np.nan
    count = 0
    for j in data_BusinessStats['area_id']:
        for i in data["area_id"]:
            if i == j:
                sum_balance = (w1*data_BusinessStats['retail_trade_z'][count] + w2*data_BusinessStats['accommodatio
                data["buss_type_banace"][count] = sum_balance
        count += 1
    #return data["buss_type_banace"].head()
```

Appendix code 4

```python
def location_boundry(name1):
    try:
        name1 = name1.strip().replace('\n','')
        #name1 = name1 +" - Sydney"
        my_params= {'address': name1,'language':'en'}
        response = requests.get(base_url, params = my_params)
        results  = response.json()['results']
        nhood_bounds = results[0]['geometry']['viewport']
        x_geo    = results[0]['geometry']['location']
        if name1 in GPS.keys() or name1 in Location.keys():
            print("duplication !!! 🐳🐳🐳")
            return
        else:
            if x_geo['lng'] < 147 or x_geo['lng'] > 155 or x_geo['lat'] < -36 or x_geo['lat'] > -30: # wrong boundary
                name1 = name1 +" - Sydney"
                location_boundry(name1)
                my_params= {'address': name1,'language':'en'}
                response = requests.get(base_url, params = my_params)
                results  = response.json()['results']
                nhood_bounds = results[0]['geometry']['viewport']
                name1 = name1.replace(" - Sydney","")
                x_geo    = results[0]['geometry']['location']

            GPS[name1] = [x_geo['lng'],x_geo['lat']]
            Location[name1] = [json.dumps(nhood_bounds, indent=4)]

        print("{} GPS : {} {}".format(name1, x_geo['lng'], x_geo['lat']))
        print("Boundary: ", json.dumps(nhood_bounds, indent=4))
        print('🌀-----------------------------------------------------------------🌀')
    except IndexError as e:
        print(name1)
        time.sleep( 1 )
        location_boundry(name1) #recursive
        print('😂-----------------------------------------------------------------😂')
```

```python
# save location
count = 0
for j in data_Neighbourhoods['area_name']:
    for k,v in Location.items():
        if j == k:
            s = ast.literal_eval(str(v[0].replace('\n','').replace(" ","").strip()))
            for k,v in s.items():
                if k == "northeast":
                    for k1, v1 in v.items():
                        if k1 == 'lat':
                            data_Neighbourhoods['Location_northeast_lat'][count]=v1
                        elif k1 == "lng":
                            data_Neighbourhoods['Location_northeast_lng'][count]=v1
                elif k == "southwest":
                    for k1, v1 in v.items():
                        if k1 == 'lat':
                            data_Neighbourhoods['Location_southwest_lat'][count]=v1
                        elif k1 == "lng":
                            data_Neighbourhoods['Location_southwest_lng'][count]=v1

    count += 1
```

```python
data_Neighbourhoods.to_csv("new_Neighbourhoods.csv",index=False)
```

Appendix code 5

```python
data_CarSharingPods.to_csv("new_CarSharingPods.csv",index=False)
```

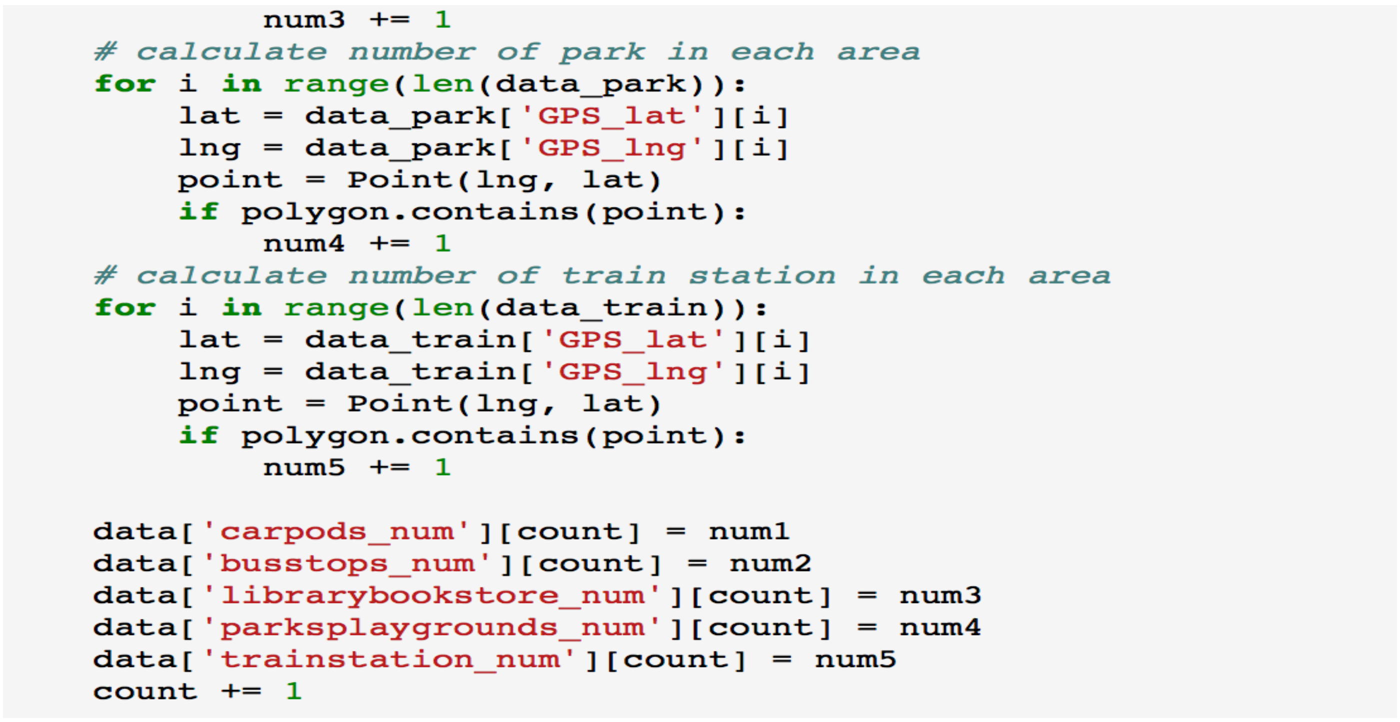
```python
sf = shp.Reader("Sydney2_2016_AUST.shp")
data_CarSharingPods['area_id']=0  # foreign key of another table

count = 0
for shape in sf.shapeRecords():
    polygon = Polygon([(i[0],i[1]) for i in shape.shape.points[:]])
    # calculate number of carpods in each area
    for i in range(len(data_CarSharingPods)):
        lat = data_CarSharingPods['latitude'][i]
        lng = data_CarSharingPods['longitude'][i]
        point = Point(lng, lat)
        if polygon.contains(point) and data_CarSharingPods['area_id'][i] == 0:
            data_CarSharingPods['area_id'][i] = data['area_id'][count]
        elif polygon.contains(point) and data_CarSharingPods['area_id'][i] != 0:
            print("wrong!!!")
            print(data_CarSharingPods['area_id'][i])
            print(data['area_id'][count])
    count += 1
```

Appendix code 6

```python
sf = shp.Reader("Sydney2_2016_AUST.shp")
data['carpods_num']=np.nan
data['busstops_num']=np.nan
data['librarybookstore_num']=np.nan
data['parksplaygrounds_num']=np.nan
data['trainstation_num']=np.nan
count = 0
for shape in sf.shapeRecords():
    num1 = 0
    num2 = 0
    num3 = 0
    num4 = 0
    num5 = 0
    polygon = Polygon([(i[0],i[1]) for i in shape.shape.points[:]])

    # calculate number of carpods in each area
    for i in range(len(data_CarSharingPods)):
        lat = data_CarSharingPods['latitude'][i]
        lng = data_CarSharingPods['longitude'][i]
        point = Point(lng, lat)
        if polygon.contains(point):
            num1 += 1
    # calculate number of busstop in each area
    for i in range(len(data_bus)):
        lat = data_bus['stop_lat'][i]
        lng = data_bus['stop_lon'][i]
        point = Point(lng, lat)
        if polygon.contains(point):
            num2 += 1
    # calculate number of lib in each area
    for i in range(len(data_lib)):
        lat = data_lib['GPS_lat'][i]
        lng = data_lib['GPS_lng'][i]
        point = Point(lng, lat)
        if polygon.contains(point):
```



```
                num3 += 1
        # calculate number of park in each area
        for i in range(len(data_park)):
            lat = data_park['GPS_lat'][i]
            lng = data_park['GPS_lng'][i]
            point = Point(lng, lat)
            if polygon.contains(point):
                num4 += 1
        # calculate number of train station in each area
        for i in range(len(data_train)):
            lat = data_train['GPS_lat'][i]
            lng = data_train['GPS_lng'][i]
            point = Point(lng, lat)
            if polygon.contains(point):
                num5 += 1

        data['carpods_num'][count] = num1
        data['busstops_num'][count] = num2
        data['librarybookstore_num'][count] = num3
        data['parksplaygrounds_num'][count] = num4
        data['trainstation_num'][count] = num5
        count += 1
```

"Walkability" URL: https://en.wikipedia.org/wiki/Walkability

"Australian Bureau of statistics, Statistical Area Level 2" URL: http://www.abs.gov.au/ausstats/abs@.nsf/Lookup/by%20Subject/1270.0.55.001~July%202016~Main%20Features~Statistical%20Area%20Level%202%20(SA2)~10014

"TransitFeeds" URL: http://transitfeeds.com/p/transport-for-nsw/
''City of Sydney'' URL: http://www.cityofsydney.nsw.gov.au/explore/facilities/parks/off-leash-parks

"List of Sydney Trains railway station" URL: https://en.wikipedia.org/wiki/List_of_Sydney_Trains_railway_stations

"Geo - Python - AutoGIS" URL: https://automating-gis-processes.github.io/2017/lessons/L3/point-in-polygon.html