Viral Vulnerability Analysis of Sydney

Prepared by F14A Group 16

Jeff Luo 470534496
Castiel Jiang 470343256

Part 1: Introduction

It is believed that the people's health is the national wealth of Australia. Despite NSW Health notifies the public of any locations where there has been a risk of infection, there are still concerns expressed about the risk of viral vulnerability in one's neighbourhood area. This report aims to present an investigation of the viral vulnerability of 312 different neighbourhoods in Sydney and share data integration and analysis skills to students and colleagues interested in Data2001. Key findings include a viral vulnerability analysis and a correlation analysis of vulnerability with covid-19 cases and tests.

Part 2: Dataset Description

- **Provided datasets**

Six main csv datasets are gathered and integrated, namely StatisticalAreas.csv, Neighbourhoods.csv, PopulationStats2016.scv, HealthServices.csv, NSW_Postcodes.csv, and COVID-19 Statistics. The detailed references of the sources are listed in the Bibliography.

- **Extra dataset on geographical boundary and number of commuters in each neighbourhood**

To integrate further data and refine parameters for analysis, an extra dataset Statistical Area 2 (SA2), provided by Australia Bureau of Statistics (ABS) is used to gauge the geographical boundary information about 312 neighbourhoods in Sydney, based on SA2_2016_AUST.shp given by ABS.

- **Use API to acquire extra dataset on number of commuters in each neighbourhood**

In addition, another extra dataset on commuters is acquired using scraping skills (Appendix Code 1). A trustworthy URL (http://soit-app-pro-4.ucc.usyd.edu.au:3000/) on university database is used to provide the number of commuters originating from each neighbourhood. This dataset serves to refine the computation of vulnerability scores of neighbourhoods.

- **Extra dataset on confirmed cases in each neighbourhood**

Another dataset on confirmed COVID19 cases is acquired from a csv file accessed from data.gov.au. This original csv file records all the test results done by the Australian government and is filtered to give the postcode_cases.csv.

- **Precise spatial join with neighbourhood**

We use geometry data to create polygon data of each neighbourhood. In each neighbourhood where health services are located are aggregated using a precise spatial join with the function *ST_Contains*. Similarly, number of COVID 19 tests are obtained using spatial join. This dataset is used with NSW_Postcodes.csv to associate number of confirmed cases with a location point. Location points are used with the geometry data in SA2_2016_AUST.shp to allow spatial join with neighbourhood to record the confirmed cases in each neighbourhood area. (Refer to Appendix Code 2 for spatial join of health service, COVID-19 tests and confirmed cases data with neighbourhood).

- **Cleaning Data**

Before the analysis is conducted, data cleaning is checked and performed as there are attributes where observations could be null values (such as num_beds in HealthServices.csv, test_capacity in COVID-19 Statistics and data in a military region). Such missing values are assumed to be replaceable with 0 for future aggregate computations. For instance, we use syntax like the following to replace any missing values as 0.

```
SUM(COALESCE(h.num_beds,0))*1.0/population*1000.0 AS "hospitalbed_density",
SUM(COALESCE(test_capacity,0)) AS "sum_test_capacity",
```

```
df = df.fillna(0)
```

Part 3: Database Description

- **Two indexes are created to speed up queries.**

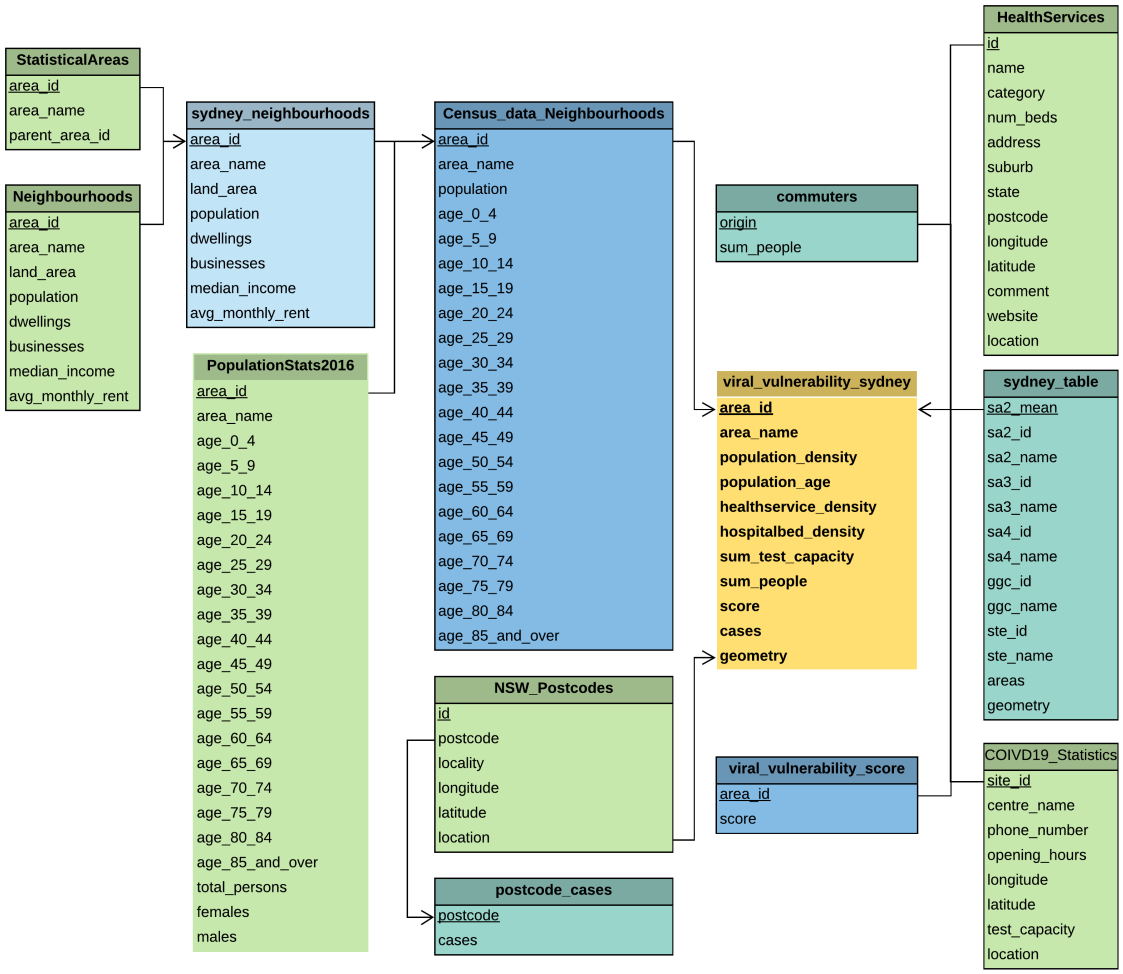
A multi-column index is created on the tuple of area_name and score column in the viral_vulnerability_sydney table. This index is created (as shown in Appendix Code 3) because these two attributes are most likely to be used by other further analyses. Moreover, query on an area_name and query on viral vulnerability score often come in pairs and could be the most common queries to explore the table to know which neighbourhoods have what scores.

The other index is created (as shown in Appendix Code 4) using GIST on the geometry column from the sydney_table obtained by creating SQL table from SA2_2016_AUST.shp, as shown below. This index is useful when we spatial join neighbourhoods table with HealthServices and COVID19_Statistics table. The performance is greatly improved to allow the join to be done within seconds.

Indexes are not created with other attributes where improvements in efficiency are negligible. Due to the relatively small size of table, sequential scanning from database may take comparable time with indexing. As such, the system could choose sequential scanning to optimize query speed as well, rather than using index search to find index of data, then fetch the data from the database.

- **Database Schema Diagram**

The database used for our analysis is stored on y20s1d2x01_zjia5829 with password 470343256 on soitpw11d59.shared.sydney.edu.au server. The following is the schema diagram for our database:



As shown in the schema, the primary keys in the above tables are underlined. They are area_id in various tables, id in nsw_postcodes and healthservices tables, origin in commuters, postcode in postcode_address, sa2_mean in sydney_table, and site_id in covid19_statistics table.

The foreign keys are postcode in nsw_postcodes and location in nsw_postcodes table.

Part 4: Viral Vulnerability Score Analysis

- **Computation of the z-score of parameters**

$$z(measure, x) = \frac{x - avg_{measure}}{stddev_{measure}}$$

Z-score is standardised score assuming normal distribution (computation is shown in Appendix Code 5).

- **Computation of the Viral Vulnerability Score**

$$vulnerability = S(z(population\_density) + z(population\_age) - z(healthservice\_density) - z(hospitalbed\_density) + z(sum\_people))$$

Where S denotes the Sigmoid function, $$S(x) = \frac{1}{1 + e^{-x}}$$

Viral vulnerability score of a specific neighbourhood is computed (as shown in Appendix Code 6) as the sigmoid value of the total z-scores of population density, percentage of people who is over age 70, density of health service within the area, number of beds available and total number of commuters originating from that area (as shown in Appendix Code 7).

- **Normalisation of the Viral Vulnerability Score**

$$normalize(x) = \frac{x - min}{max - min} * 100$$

Viral Vulnerability Score is normalized and rounded as a score from 0 to 100 (as shown in Appendix code 8). The calculation results are saved in final_table.csv and relevant table in DBMS (as shown in Appendix Code 9). The table is then analysed with python scripts for graphical visualization of the viral vulnerability scores in 312 neighbourhoods in Sydney (The code used is in task 3 visualisation.ipynb and in Appendix Code 10).

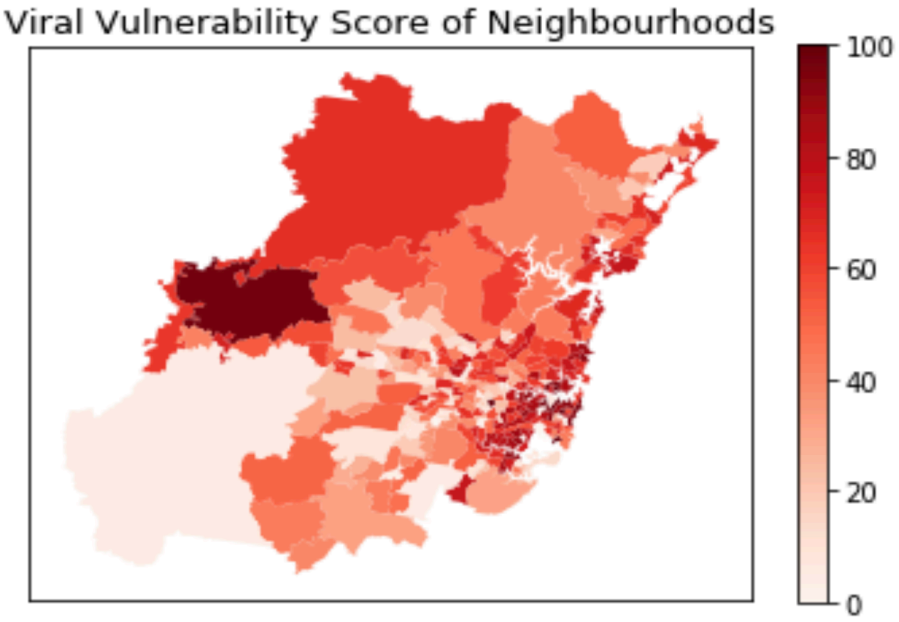


Figure 1 – Viral vulnerability scores for 312 areas in Sydney

As shown in Figure 1, the viral vulnerability scores are classified into 5 classes and values are explicitly shown by gradual changes from light red to dark red, denoting scores increasing from 0 to 100. The observations can be useful to inform the public about the risk of viral vulnerability in one's neighbourhood area and help prevent infections by using more stringent measures in areas with higher viral vulnerability scores. Take Bondi Beach for instance,

- Bondi Beach – North with score 96. This score is intuitively reasonable as Bondi Beach contains restaurants and high population density and commuters bustling with activity day and night. A travel ban should be issued here.

Nevertheless, some observations need more investigation to be justifiable, for instance,

- Sydney Airport with score 0. This observation is not expected. Further investigation (as shown in Appendix Code 11) shows that Sydney airport has very low population density, very low over-70-age percentages, high health services density and zero commuter originating from Sydney Airport (This observation is strange). It is speculated that deeper investigations are needed to see if this observation is an outlier.

Part 5: Correlation Tests and Analysis

Plotting the correlation between vulnerability score and test capacity, number of confirmed COVID19 cases in neighbourhoods, the results verify a negative correlation between vulnerability score and test capacity and a positive correlation between vulnerability score and number of confirmed cases.

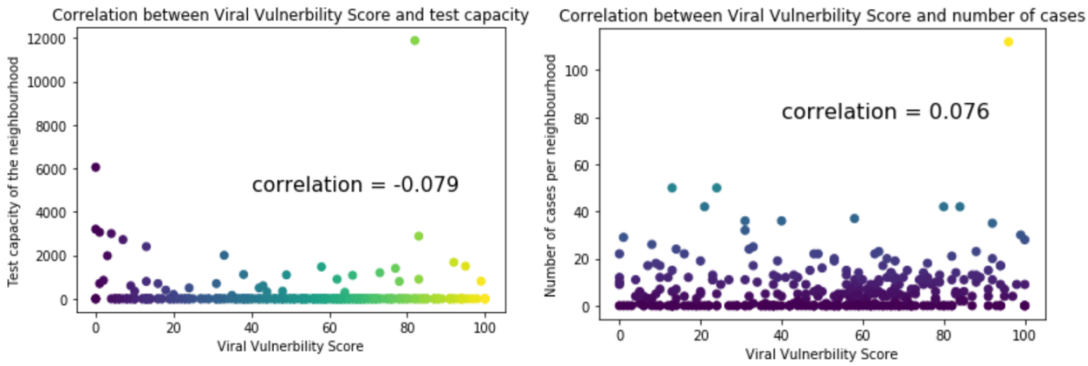


Figure 2 – the correlations between viral vulnerability score and test capacity /number of confirmed cases

To verify if there is a trend in the correlations between score and test capacity or number of confirmed cases in a neighbourhood, we calculate the correlation values using correlation function of python dataframe (as shown in Appendix Code 12 and task 3 correlation.ipynb).

Table 1 – *the correlation values between viral vulnerability score and test capacity /number of confirmed cases*

	Correlation value	Coefficient of determination
Test capacity	-0.079	0.006
Cases	0.076	0.006

It is observed in the figure that the viral vulnerability score of a neighbourhood has a very weak positive correlation with COIVID19 cases and a very weak negative correlation with test capacity of the neighbourhood. This observation is reasonable and implies that the area with higher vulnerability score tends to have less test capacity and more cases. This is a trend that can be further researched on with properly controlled experiments to investigate the causal relationship.

The coefficient of determination, which is the square of the coefficient is introduced to gauge the percentage of variations correlated to test capacity and cases respectively. For the test capacity, the coefficient of determination is around 0.006, which means that 0.6% of the variation in viral vulnerability score can be predicted from the variation in z-score of test capacity in a neighbourhood. For the number of COVID19 cases, the coefficient of determination is also around 0.006, which means that 0.6% of the variation in viral vulnerability score can be predicted from the variation in z-score of confirmed cases in a neighbourhood

It is noted the correlation values are quite small due to the presence of outliers and yet-to-improve score values. The score values such as the score of Industrial area and military area are high due to lack of health services and bed capacity, they are likely to be outliers as the information in these areas are not accurate due to missing field or secrecy. In addition, the z-score of parameters are added with equal ratio, further investigation may improve the accuracy of the score by using machine learning skills to adjust the weightage of each parameter and make the score a more complex and trustworthy estimation of the real viral vulnerability of each neighbourhood.

Bibliography

Abs.gov.au. 2020. 1270.0.55.001 - Australian Statistical Geography Standard (ASGS): Volume 1 - Main Structure and Greater Capital City Statistical Areas, July 2016. [online] Available at: <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> [Accessed 28 May 2020].

Automating-gis-processes.github.io. 2020. Point in Polygon & Intersect — Geo-Python - Autogis Documentation. [online] Available at: <https://automating-gis-processes.github.io/2017/lessons/L3/point-in-polygon.html> [Accessed 28 May 2020].

Appendix

- Code 1

Additional Table Destination_people

Change the limit first!!

jupyter notebook --NotebookApp.iopub_data_rate_limit=10000000

```
import requests
import json
base_url = 'http://soit-app-pro-4.ucc.usyd.edu.au:3000/api/v1/json'

my_params= {'data' :'origin, destinations','format':'json','limit':'1'}
response = requests.get(base_url, params = my_params)
```

- Code 2

FINAL TABLE WITH DATA CLEANING

```
conn.execute("""
       INSERT INTO viral_vulnerability_sydney
       SELECT sn.area_id, sn.area_name,
       COALESCE(population*1.0/land_area,0) AS "population_density",
       COALESCE((age_70_74 + age_75_79 + age_80_84 + age_85_and_over)*1.0/population,0) AS "population_age",
       COALESCE(COUNT(DISTINCT h.id)*1.0/population*1000,0) AS "healthservice_density",
       COALESCE(SUM(COALESCE(h.num_beds,0))*1.0/population*1000.0,0) AS "hospitalbed_density",
       SUM(COALESCE(test_capacity,0)) AS "sum_test_capacity",
       COALESCE(cm.sum_people*1.0/population*1000.0,0) AS "sum_people",
       sco.score,
       MAX(COALESCE(cases,0)) AS "cases",
       st.geometry
       FROM sydney_table st JOIN sydney_neighbourhoods sn ON st.SA2_mean = sn.area_id
                                       JOIN PopulationStats2016 USING(area_id)
                                       LEFT JOIN healthservices h ON (ST_Contains(st.geometry, h.location))
                                       LEFT JOIN covid19_statistics cs  ON (ST_Contains(st.geometry, cs.locatio
                                       LEFT JOIN commuters cm ON sn.area_id = cm.origin
                                       LEFT JOIN viral_vulnerability_score sco USING(area_id)
                                       LEFT JOIN (SELECT postcode,cases,location
                                                  FROM postcode_cases RIGHT JOIN
                                                 (SELECT DISTINCT postcode,location
                                                  FROM nsw_postcodes) AS postcode_dis USING(postcode))
                                                  AS post_cases ON (ST_Contains(st.geometry, post_cases.locatio
       GROUP BY sn.area_id,population_age,st.geometry,cm.origin,sco.area_id;
""")
```

- Code 3

Create Multicolumn Index for final table

```
index_multicolumn = "CREATE INDEX area_name_and_score ON viral_vulnerability_sydney (area_name,score);"
pgquery(conn, index_multicolumn, returntype='dict')
```

- Code 4

Create Index on GIST(geometry)

```
index_command = "CREATE INDEX areas_idx ON sydney_table USING GIST (geometry)"
pgquery(conn, index_command, returntype='dict')
```

- Code 5

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

```

```
🙈 ---------------------- the mean ------------------------ 🙈
population_density        29.288288
population_age             0.097765
healthservice_density      0.350541
hospitalbed_density        4.045346
sum_people               326.660781
dtype: float64
🙈 ---------------------- the std ------------------------ 🙈
population_density        26.188167
population_age             0.077563
healthservice_density      0.367075
hospitalbed_density       23.491719
sum_people               200.074565
dtype: float64
🙈 ---------------------- the z_scores ------------------------ 🙈
     population_density_zscore  population_age_zscore  \
0                    -0.668199              -0.161439
1                    -0.987636               0.660658
2                    -1.115972               0.214971
3                    -0.957504               1.545934
4                    -0.680690               0.328434
..                         ...                    ...
307                   0.070842               0.188247
308                   0.089832               0.625038
309                  -0.066258               1.130395
310                  -0.380701               0.097215
311                  -0.670308              -0.672045
```

- Code 6

```
# Add the extra parameter and compute sum of all 5 parameters
row_sum_of_z_scores = all_z_scores['population_density_zscore'] + all_z_scores['population_age_zscore']\
                      - all_z_scores['healthservice_density_zscore'] - all_z_scores['hospitalbed_density_zscore'
                      + all_z_scores['sum_people_zscore']
print("\n🙈 ---------------------- the sum of z_scores ------------------------ 🙈\n")
total_z_score_for_print = list(zip(table.iloc[:,2], row_sum_of_z_scores))
total_z_score_for_print.sort(key = lambda x:-x[-1])
print("{:<50}{:<8}".format('name','sum of first 5'))
for _ in total_z_score_for_print :
    print("{:<50}{:<8}".format(*_))
```

```
🙈 ---------------------- the sum of z_scores ------------------------ 🙈

name                                              sum of first 5
Port Botany Industrial                            10.309591361240548
Centennial Park                                   10.008985470291465
Rookwood Cemetery                                 7.1122703520011346
Pyrmont - Ultimo                                  5.463327624375231
Potts Point - Woolloomooloo                       5.313291305094316
Surry Hills                                       4.37795361812251
Neutral Bay - Kirribilli                          3.791705672019842
Blue Mountains - North                            3.328324141866923
Bondi Beach - North Bondi                         3.1166091190414322
Redfern - Chippendale                             3.010933886353115
```

- Code 7

Extra table of commuters to refine scores

```
conn.execute("drop table if exists commuters;")

commuters = """
create table commuters(
    origin varchar(200) primary key,
    sum_people integer
)
"""

conn.execute(commuters)

matches  = response.json()
num_matches = len(matches)

list = []
for i in range(0,num_matches):
    match_origin = matches[i]['origin']
    match_destinations = matches[i]['destinations']
    sum_people = 0
    for j in range(0,len(match_destinations)):
        sum_people += match_destinations[j]['people']
    list.append([match_origin,sum_people])

df = pd.DataFrame (list)
df.columns = ['origin','sum_people']
table_name = "commuters"
df.to_sql(table_name, con=conn, if_exists='append', index=False)
```

- Code 8

```
import math

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

def normalize(data):
    result = []
    min_val = min(data)
    max_val = max(data)
    for val in data:
        normalized_score =(val-min_val) * 100 / (max_val - min_val)
        result.append(round(normalized_score))
    return result

# compute vulnerability for each neighbourhood
vulnerability = []
for i in row_sum_of_z_scores:
    vulnerability.append(sigmoid(i))
scores = normalize(vulnerability)
print("🙈 ---------------------- the normalised scores ---------------------- 🙈")
print(scores)
```

- Code 9

Final version table to csv

```
final_data = pd.read_sql_table('viral_vulnerability_sydney',conn)
final_data.to_csv('./final_table.csv',encoding='gbk')
```

- Code 10

Merge score data into the GeoDataFrame

```
# change column name
areas = areas.rename(index = str, columns = {'SA2_MAIN16':'area_id'})

# merge data
areas = areas.merge(df, on = 'area_id', how = 'inner')
```

Plot the Visualisation

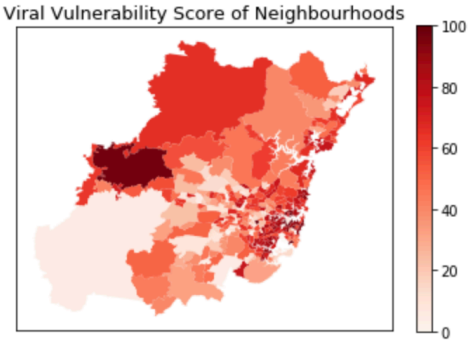
```

# alhpa = 1 is the opacity of color
# Possible cmap values are: Blues, Greens,  Pastel2, rainbow
areas.plot('score', k = 5, cmap = 'Reds', alpha = 1, legend=True)

# extra feature needs: conda install -c anaconda pysal
#areas.plot('score', k = 4, cmap = plt.cm.Greens, scheme = 'percentiles',alpha= 1, figsize = (9, 9), legend = Tr

plt.title('Viral Vulnerability Score of Neighbourhoods', fontsize=13)

# remove x-axis and y-axis
plt.gca().xaxis.set_major_locator(plt.NullLocator())
plt.gca().yaxis.set_major_locator(plt.NullLocator())
```



- Code 11

print the top 20 and bottom 20

```
table['score'] = pd.DataFrame(scores)
table.sort_values("score",inplace=True, ascending=False)
print("\n🐵 ---------------------- the top 20 ---------------------- 🐵\n")
print(table.iloc[0:20, [2,9,10]])
print("\n🐵 ---------------------- the middle 20 ---------------------- 🐵\n")
print(table.iloc[150:170, [2,9,10]])
print("\n🐵 ---------------------- the random 3 ---------------------- 🐵\n")
print(table.iloc[[38,77,238], [2,9,10]])
print("\n🐵 ---------------------- the safe 20 ---------------------- 🐵\n")
print(table.iloc[-20:, [2,9,10]])
```

	area_name	population_density	population_age	healthservice_density	hospitalbed_density	sum_people	score	cases
243	St Marys - North St Marys	13.055577	0.087699	0.714456	0.000000	0.000000	4	11
245	Auburn - Central	49.758907	0.059381	0.807537	19.930013	0.000000	4	4
212	Campbelltown - Woodbine	14.066773	0.096024	1.137598	24.256683	357.774445	3	0
190	Hornsby - East	48.921759	0.062858	1.352642	2.068746	0.000000	2	0
48	Blacktown (East) - Kings Park	22.405016	0.071748	0.951422	71.961048	395.567495	1	0
105	Randwick - South	42.211681	0.078522	0.983284	52.345835	0.000000	1	29
106	Kogarah	55.072072	0.082938	1.390479	120.644528	376.083756	0	10
303	Liverpool	46.819117	0.077438	1.081629	53.871895	0.000000	0	22
130	Bankstown - South	38.335971	0.077200	1.533381	22.983015	0.000000	0	0
83	Newtown - Camperdown - Darlington	83.663517	0.033807	0.400729	364.021858	473.224044	0	9
74	Sydney Airport	0.226686	0.113861	4.950495	0.000000	0.000000	0	12

- Code 12

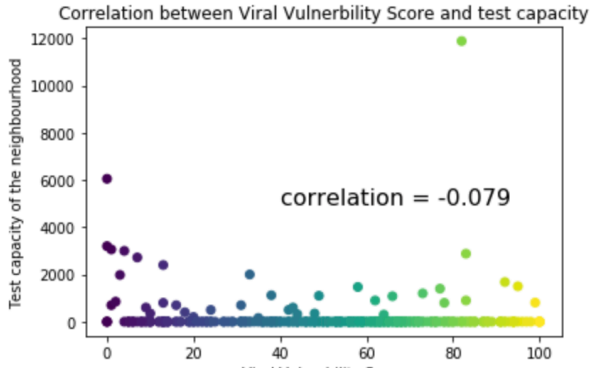
Correlation between Viral Vulnerbility Score and test_capacity

```
plt.scatter(scores,tests,c=scores)
plt.title('Correlation between Viral Vulnerbility Score and test capacity')
plt.xlabel('Viral Vulnerbility Score')
plt.ylabel('Test capacity of the neighbourhood')

#rounding to 3 decimal places
test_corr = round(scores.corr(tests),3)
print("coefficient of determination (the sqaure) is", round(test_corr**2, 3) )
plt.annotate('correlation = '+str(test_corr), xy = (20, 40), xytext=(40, 5000),fontsize=16)
```

```
coefficient of determination (the sqaure) is 0.006

Text(40, 5000, 'correlation = -0.079')
```



Correlation between Viral Vulnerbility Score and number of COIVD19 cases

```
plt.scatter(scores,cases,c=cases)
plt.title('Correlation between Viral Vulnerbility Score and number of cases')
plt.xlabel('Viral Vulnerbility Score')
plt.ylabel('Number of cases per neighbourhood')

#rounding to 3 decimal places
test_corr = round(scores.corr(cases),3)
print("coefficient of determination (the sqaure) is", round(test_corr**2, 3) )
plt.annotate('correlation = '+str(test_corr), xy = (20, 40), xytext=(40, 80),fontsize=16)
```

```
coefficient of determination (the sqaure) is 0.006

Text(40, 80, 'correlation = 0.076')
```