**FIT5137 S2 2023 Assignment 4: PTV Answer Sheet (Weight = 30%)**

**PLEASE SUBMIT ANSWER SHEET IN PDF FORMAT**

**Due date: Wednesday, 25 October 2023, 11:55pm**

Version: 2.0 – 25/09/2023

**Assignment Task list**

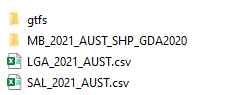
Your assignment consists of several parts. Always read the instruction one by one. Do not move to the step without completing the previous step:

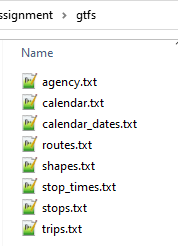
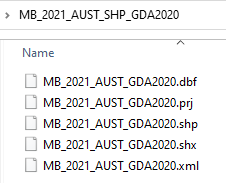
* Task 1: Data Restoration - Restore the data to the database. Monitor the success indicator to ensure successful restoration of the data.
* Task 2: Data Preprocessing - Perform necessary structure maintenance and create result tables for further processing.
* Task 3: Data Analytics - Develop SQL queries to analyze the data and evaluate performance.
* Task 4: Data Visualization - Create visualizations to present the results of the data analytics.

**For simplicity, all the data required for this assignment is readily available in the PostGIS Docker container. You can access these datasets within the container by navigating to the /data/adata folder.If you don’t know how to do it, refer to the labs 10 activities. As a data analyst, it is your responsibility to understand and explore these publicly available data.**



Verify your data before the restoration process.



**As a data analyst, it is your responsibility to understand and explore these publicly available data.**

**Do not edit or remove any content on the answer sheet, including the questions**. Please write your answers in the answer boxes provided. If necessary, you can adjust the size of the answer box to fit your answer.

## **Task 1: Data Restoration**

Before you can start the data analytic processes, the first thing you have to do is to restore the external data to your database. Make sure you prepare a destination schema to restore your data. The destination schema for your assignment is “**ptv**”.

Note:

* Before initiating the data restoration process, **it is essential to thoroughly explore the dataset**. This exploration involves identifying appropriate data types, determining field lengths, and making other relevant considerations that will inform the creation of the table structure.
* Ensure that you restore the data into the PTV schema using regular (local) tables. Do not utilize foreign tables, as the data must be stored directly within the PostgreSQL database – updated 27/09/2023
* Make sure all 8 GTFS tables are restored successfully
* Index or constraints can be added to the table after the data has been restored completely

(Note: This index or constraint requirement is **NOT** mandatory in this Task 1 – updated 25/09/2023)

* **No data cleaning required for this assignment**
* For more information, see the FAQ for Assignment 4.

### 1.1 PTV schema

Write the SQL script to create the destination schema named “**ptv**”.

| create schema ptv; |
| --- |

### 1.2 GTFS

Write the SQL script to restore **ALL** tables in GTFS files.

| --1.2  -- Create & Import Agency Table  create table ptv.agency(  agency\_id numeric,  agency\_name varchar,  agency\_url varchar,  agency\_timezone varchar,  agency\_lang varchar  );  COPY ptv.agency FROM '/data/adata/gtfs/agency.txt' DELIMITER ',' CSV HEADER;  -- Create & Import Routes Table  create table ptv.routes(  route\_id varchar,  agency\_id numeric,  route\_short\_name varchar,  route\_long\_name varchar,  route\_type numeric,  route\_color varchar,  route\_text\_color varchar  );  COPY ptv.routes FROM '/data/adata/gtfs/routes.txt' DELIMITER ',' CSV HEADER;  -- Create & Import Calendar Table  create table ptv.calendar(  service\_id varchar,  monday numeric,  tuesday numeric,  wednesday numeric,  thursday numeric,  friday numeric,  saturday numeric,  sunday numeric,  start\_date date,  end\_date date  );  COPY ptv.calendar FROM '/data/adata/gtfs/calendar.txt' DELIMITER ',' CSV HEADER;  -- Create & Import Calendar\_Dates Table create table ptv.calendar\_dates(  service\_id varchar,  date date,  exception\_type numeric  );  COPY ptv.calendar\_dates FROM '/data/adata/gtfs/calendar\_dates.txt' DELIMITER ',' CSV HEADER;  -- Create & Import Shapes Table create table ptv.shapes(  shape\_id varchar,  shape\_pt\_lat float,  shape\_pt\_lon float,  shape\_pt\_sequence numeric,  shape\_dist\_traveled float  );  COPY ptv.shapes FROM '/data/adata/gtfs/shapes.txt' DELIMITER ',' CSV HEADER;  -- Create & Import Stop\_Times Table create table ptv.stop\_times(  trip\_id varchar,  arrival\_time varchar,  departure\_time varchar,  stop\_id varchar,  stop\_sequence numeric,  stop\_headsign varchar,  pickup\_type numeric,  drop\_off\_type numeric,  shape\_dist\_travelerd varchar  );  COPY ptv.stop\_times FROM '/data/adata/gtfs/stop\_times.txt' DELIMITER ',' CSV HEADER;  -- Create & Import Stops Table create table ptv.stops(  stop\_id varchar,  stop\_name varchar,  stop\_lat float,  stop\_lon float  );  COPY ptv.stops FROM '/data/adata/gtfs/stops.txt' DELIMITER ',' CSV HEADER;  -- Create & Import Trips Table create table ptv.trips(  route\_id varchar,  service\_id varchar,  trip\_id varchar,  shape\_id varchar,  trip\_headsign varchar,  direction\_id numeric  );  COPY ptv.trips FROM '/data/adata/gtfs/trips.txt' DELIMITER ',' CSV HEADER; |
| --- |

### 1.3 ABS Mesh Blocks

Scripts to restore the Mesh Blocks files by using correct dataset file. Restore the file using ogr2ogr into table “**mb2021**”

| ogr2ogr PG:"dbname=gisdb user=postgres" "/data/adata/MB\_2021\_AUST\_SHP\_GDA2020/MB\_2021\_AUST\_GDA2020.shp" -nln ptv.mb2021 -overwrite -nlt MULTIPOLYGON |
| --- |

### 1.4 ABS Allocation Files

Write the SQL script to restore the LGA2021 Allocation file.

| create table ptv.lga2021(  mb\_code\_2021 varchar,  lga\_code\_2021 varchar,  lga\_name\_2021 varchar,  state\_code\_2021 varchar,  state\_name\_2021 varchar,  aus\_code\_2021 varchar,  aus\_name\_2021 varchar,  area\_albers\_sqkm float,  asgs\_loci\_uri\_2021 varchar  );  COPY ptv.lga2021 FROM '/data/adata/LGA\_2021\_AUST.csv' DELIMITER ',' CSV HEADER; |
| --- |

Write the SQL script to restore the SAL 2021 Allocation file for suburb2021.

| create table ptv.sal2021(  mb\_code\_2021 varchar,  sal\_code\_2021 varchar,  sal\_name\_2021 varchar,  state\_code\_2021 varchar,  state\_name\_2021 varchar,  aus\_code\_2021 varchar,  aus\_name\_2021 varchar,  area\_albers\_sqkm float,  asgs\_loci\_uri\_2021 varchar  );  COPY ptv.sal2021 FROM '/data/adata/SAL\_2021\_AUST.csv' DELIMITER ',' CSV HEADER; |
| --- |

The allocation tables have 1-N relationship with the mb2021 in mb\_code21 – mb\_code\_2021. Although there are no PK – FK defined in the table, the relationship rule still apply.



### 1.5 Data Verification

Verify your restoration by running this script. 1.Do not modify the verification script. 2.Make sure that the table name is consistent with the table we provided.

Output: Attach a screenshot of the results to include all tables you have restored in Task 1.

| **with** tbl **as**  (**select** table\_schema, TABLE\_NAME  **from** information\_schema.**tables**  **where** table\_schema **in** ('ptv'))  **select** table\_schema, TABLE\_NAME,  (**xpath**('/row/c/text()', **query\_to\_xml**(format('select count(\*) as c from %I.%I', table\_schema, TABLE\_NAME), **FALSE**, **TRUE**, '')))[1]::**text**::**int** **AS** rows\_n  **from** tbl  **order** **by** table\_name; |
| --- |

Screenshot:

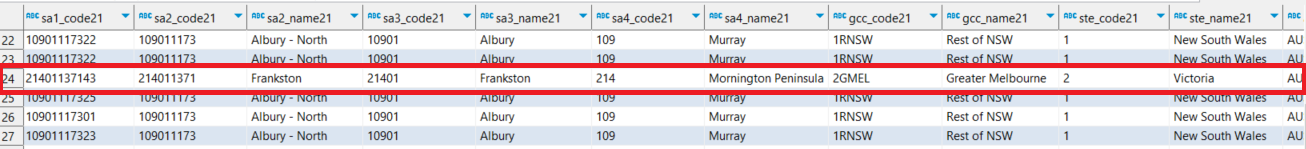
|  |
| --- |

## **Task 2: Data Preprocessing**

### 2.1 Filter Melbourne Metropolitan area

The mb2021 table contains whole mesh blocks in Australia. To minimise the query cost, we want to ensure that you only use the mesh blocks in Melbourne Metropolitan. The Melbourne Metropolitan’s mesh blocks can be identified from the gcc\_name21. If the column contains “Greater Melbourne”, this mesh block is located in Melbourne Metropolitan.

Create a table named “**mb2021\_mel**” that contains ONLY the mesh blocks in Melbourne Metropolitan.



Write the SQL script to do this.

| create table ptv.mb2021\_mel as  (  select \* from ptv.mb2021  where lower(gcc\_name21) = lower('Greater Melbourne')  ); |
| --- |

Attach a screenshot of the Spatial Map results.

Screenshot:

|  |
| --- |

### 2.2 Melbourne Metropolitan Boundary

Since the working area will be Melbourne Metropolitan, it is important to have a polygon for the boundary of our working area. Create a table, named “**melbourne**” for Melbourne Metropolitan boundary.   
Hint: aggregate all mesh blocks polygon to create one large polygon for Melbourne Metropolitan boundary.

Write the SQL script to do this.

| create table ptv.melbourne as  (  select st\_union(wkb\_geometry) from ptv.mb2021\_mel  ); |
| --- |

Attach a screenshot of the Spatial Map results.

|  |
| --- |

### 2.3 Add Geometry column to Stops table

Stops table does not have any geometry column. Add a geometry column by using the latitude and longitude value that are available in the table. Make sure you use GDA2020 (SRID:7844) for this column.

Write the SQL script to do this.

| -- Add Geometry Column to Stops table  -- Step 1 Alter new column  alter table ptv.stops  add geom geometry;  -- Update the geom with the geom value from their lon and lat  update ptv.stops  set geom = ST\_SetSRID(ST\_MakePoint(stop\_lon, stop\_lat), 7844); |
| --- |

Attach a screenshot of the Spatial Map results.

|  |
| --- |

### 2.4 Denormalise GTFS structure

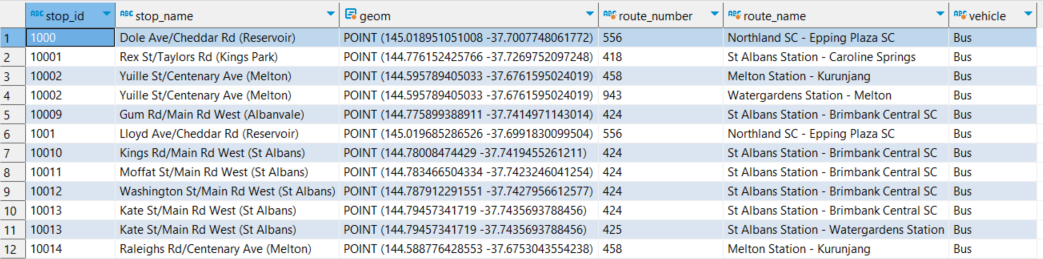
The ptv.stops table does not show direct information regarding the vehicle types, routes\_short\_name and routes\_long\_name. These information are stored in the routes table.

Create a table called "stops\_routes\_mel" to encompass the following attributes: stop\_id, stop\_name, coordinates, route number (derived from routes\_short\_name), route name (derived from routes\_long\_name), and vehicle type. This data set should encompass all stops within the Melbourne Metropolitan area.

The vehicle type is determined by the corresponding route type, where:

* 0 corresponds to tram
* 2 corresponds to train
* 3 corresponds to bus
* Any other route type is labeled as 'Unknown'.

Use this figure as an example of expected result. (Note:Data value is for demonstration purposes only.)



Make sure you remove any duplications in your result.

Write your SQL query here

| create table ptv.stops\_routes\_mel as  (  select distinct s.stop\_id, s.stop\_name, s.geom , r.route\_short\_name as "route\_number", r.route\_long\_name as "route\_name",  case  when r.route\_type = 0 then 'Tram'  when r.route\_type = 2 then 'Train'  when r.route\_type = 3 then 'Bus'  else 'Unknown'  end as Vehicle  from ptv.stops s  join ptv.stop\_times st on s.stop\_id = st.stop\_id  join ptv.trips t on st.trip\_id = t.trip\_id  join ptv.routes r on t.route\_id = r.route\_id  ); |
| --- |

Please complete the following statistics from **stops\_routes\_mel** table:

**Question 2.4.1**:How many rows do you have in the stops\_routes\_mel table?

Write the SQL script to do this and attach a screenshot of the query

| select count(\*) from ptv.stops\_routes\_mel; |
| --- |

Screenshot

|  |
| --- |

**Question 2.4.2**:How many unique stop\_ids do you have in the stops\_routes\_mel table?

Write the SQL script to do this and attach a screenshot of the query

| select count(distinct(stop\_id)) from ptv.stops\_routes\_mel; |
| --- |

Screenshot

|  |
| --- |

## **Task 3: Data Analytics**

### 3.1 Suburbs Accessibility

Create an SQL query to identify the **number of bus stops** in each Suburb. Your result should have the suburb name and the total bus stops in it.

Hint :

* identify the mesh block location of a bus stop. Then, aggregate the number in Suburb level.
* One suburb consists of multiple mesh blocks

Write your SQL query here

| WITH BusStops AS (  -- Filter out stops where the vehicle is a 'Bus'  SELECT stop\_id, geom  FROM ptv.stops\_routes\_mel  WHERE vehicle= 'Bus'  ),  BusStopsWithMB AS (  -- Join bus stops with mesh blocks based on spatial location  SELECT b.stop\_id, m.mb\_code21, m.sa2\_name21  FROM BusStops b  JOIN ptv.mb2021\_mel m ON ST\_Contains(m.wkb\_geometry, b.geom)  )  -- Join with sal2021 to get the suburb name and count the bus stops  SELECT s.sal\_name\_2021 AS suburb\_name, COUNT(b.stop\_id) AS total\_bus\_stops  FROM BusStopsWithMB b  JOIN ptv.sal2021 s ON b.mb\_code21 = s.mb\_code\_2021  GROUP BY s.sal\_name\_2021  ORDER BY total\_bus\_stops DESC; |
| --- |

Provide the screenshot of your **execution plan** and the real **execution time** for this query. You can get the real execution time under your result set.

Note:

* Execution plan can be found in SQL Editor -> Explain Execution Plan
* The execution time is shown in the screenshot below.(Note: the screenshot is for demonstration purposes only)  
  

| Execution plan : |
| --- |

| Execution time : |
| --- |

You are now tasked with devising an approach to enhance your query execution and minimize execution time. Provide a comprehensive explanation of your strategy, accompanied by the SQL script outlining the measures you've taken to optimize query performance. Additionally, include a screenshot showcasing the execution plan and execution time, effectively visualizing the enhancements achieved following the optimization.

| Strategy and SQL script:  **Strategy:** Due to the nature of the complexity with performing spatial operations, we have decided to modify some parts of the logic inside the queries that significantly improves our code in fetching the required query.   1. Firstly, we introduce what is called spatial indexing. This ensures that spatial indexes exist on the spatial columns which significantly speed up spatial queries that are essentially involved in the spatial operations. Though, it seems like the indexes created are redundant inside the queries. These indexes are implicitly used by the database engine when performing the spatial operations on the BoundingBox check logic. 2. Which comes on our second modification that is the BoundingBoxJoin. Instead of directly using ST\_Contains function to determine if a mesh block contains a bus stop which was used in BusStopsWithMB on the non-optimised queries, it performs a simpler bounding box check of (b.geom && m.wkb\_geometry). This checks if the bounding boxes of the two geometries overlap. Not only these checks are less computationally intensive than the detailed spatial checks, the spatial indexes that have been introduced earlier help the spatial operations to quickly narrow down potential matches, without the need of scanning the entire table which results in slow and inefficient operations.   **SQL Script:**  -- OPTIMIZED 3.1  -- Ensure spatial indexes exist on the spatial columns  CREATE INDEX IF NOT EXISTS idx\_stops\_routes\_mel\_geom ON ptv.stops\_routes\_mel USING GIST(geom);  CREATE INDEX IF NOT EXISTS idx\_mb2021\_mel\_wkb\_geometry ON ptv.mb2021\_mel USING GIST(wkb\_geometry);  -- Filter first: Create a temporary table with only bus stops  WITH BusStops AS (  SELECT stop\_id, geom  FROM ptv.stops\_routes\_mel  WHERE vehicle = 'Bus'  ),  -- Use bounding box checks before detailed spatial checks  BoundingBoxJoin AS (  SELECT b.stop\_id, m.mb\_code21, m.sa2\_name21  FROM BusStops b  JOIN ptv.mb2021\_mel m ON b.geom && m.wkb\_geometry -- Bounding box check  WHERE ST\_Contains(m.wkb\_geometry, b.geom) -- Detailed spatial check  )  -- Final aggregation  SELECT s.sal\_name\_2021 AS suburb\_name, COUNT(b.stop\_id) AS total\_bus\_stops  FROM BoundingBoxJoin b  JOIN ptv.sal2021 s ON b.mb\_code21 = s.mb\_code\_2021  GROUP BY s.sal\_name\_2021  ORDER BY total\_bus\_stops DESC; |
| --- |

| Improved Execution plan : |
| --- |

| Improved Execution Time : |
| --- |

**Question 3.1.1**: Provide a list of the five suburbs with the lowest count of stops. In case multiple suburbs share the same minimum number of stops in your findings, arrange them in ascending order based on their suburb names.

Write the SQL script to do this and attach a screenshot of the query

| -- 3.1.1  -- Ensure spatial indexes exist on the spatial columns  CREATE INDEX IF NOT EXISTS idx\_stops\_routes\_mel\_geom ON ptv.stops\_routes\_mel USING GIST(geom);  CREATE INDEX IF NOT EXISTS idx\_mb2021\_mel\_wkb\_geometry ON ptv.mb2021\_mel USING GIST(wkb\_geometry);  -- Filter first: Create a temporary table or CTE with only bus stops  WITH BusStops AS (  SELECT stop\_id, geom  FROM ptv.stops\_routes\_mel  WHERE vehicle = 'Bus'  ),  -- Use bounding box checks before detailed spatial checks  BoundingBoxJoin AS (  SELECT b.stop\_id, m.mb\_code21, m.sa2\_name21  FROM BusStops b  JOIN ptv.mb2021\_mel m ON b.geom && m.wkb\_geometry -- Bounding box check  WHERE ST\_Contains(m.wkb\_geometry, b.geom) -- Detailed spatial check  )  -- Final aggregation  SELECT s.sal\_name\_2021 AS suburb\_name, COUNT(b.stop\_id) AS total\_bus\_stops  FROM BoundingBoxJoin b  JOIN ptv.sal2021 s ON b.mb\_code21 = s.mb\_code\_2021  GROUP BY s.sal\_name\_2021  ORDER BY total\_bus\_stops ASC, s.sal\_name\_2021 ASC  LIMIT 5; |
| --- |

Screenshot

|  |
| --- |

**Question 3.1.2**: Average number of distinct stops in suburb

Write the SQL script to do this and attach a screenshot of the query

| -- 3.1.2  -- Ensure spatial indexes exist on the spatial columns  CREATE INDEX IF NOT EXISTS idx\_stops\_routes\_mel\_geom ON ptv.stops\_routes\_mel USING GIST(geom);  CREATE INDEX IF NOT EXISTS idx\_mb2021\_mel\_wkb\_geometry ON ptv.mb2021\_mel USING GIST(wkb\_geometry);  -- Filter first: Create a temporary table with only bus stops  WITH BusStops AS (  SELECT stop\_id, geom  FROM ptv.stops\_routes\_mel  WHERE vehicle = 'Bus'  ),  -- Use bounding box checks before detailed spatial checks  BoundingBoxJoin AS (  SELECT b.stop\_id, m.mb\_code21, m.sa2\_name21  FROM BusStops b  JOIN ptv.mb2021\_mel m ON b.geom && m.wkb\_geometry -- Bounding box check  WHERE ST\_Contains(m.wkb\_geometry, b.geom) -- Detailed spatial check  ),  -- Aggregation by suburb  SuburbStops AS (  SELECT s.sal\_name\_2021 AS suburb\_name, COUNT(DISTINCT b.stop\_id) AS distinct\_bus\_stops  FROM BoundingBoxJoin b  JOIN ptv.sal2021 s ON b.mb\_code21 = s.mb\_code\_2021  GROUP BY s.sal\_name\_2021  )  -- Calculate the average  SELECT AVG(distinct\_bus\_stops) AS average\_distinct\_stops\_per\_suburb  FROM SuburbStops; |
| --- |

Screenshot

|  |
| --- |

### 3.2 LGA Blankspot

The next step is to evaluate the **residential area** **without Bus Stops**. The residential area without any Bus Stops in it is considered as the **Blankspot**. Each mesh block has a distinct category. The category is defined in “**mb\_cat21**” column, mb2021\_mel table. In this task, your duty is to identified the percentage of blankspot in every city council or LGA. Below is the blankspot example in Kingsbury.



Let B as the number of residential blankspot, R as the total number of Residential Mesh Blocks, where B ⊆ R. The percentage of blankspot X in every LGA can be calculated using the following formula

Display the LGA name, total number of Residential Mesh Blocks, total number of residential blankspot, percentage of blankspot in Melbourne Metropolitan. Sort results in ascending order by total number of Residential Mesh Blocks.

Write the SQL script to do this and attach a screenshot of the query

| -- 3.2 LGA BLANKSPOT  -- Ensure spatial indexes exist on the spatial columns  CREATE INDEX IF NOT EXISTS idx\_stops\_routes\_mel\_geom ON ptv.stops\_routes\_mel USING GIST(geom);  CREATE INDEX IF NOT EXISTS idx\_mb2021\_mel\_wkb\_geometry ON ptv.mb2021\_mel USING GIST(wkb\_geometry);  -- Filter first: Create a temporary table or CTE with only bus stops  WITH BusStops AS (  SELECT stop\_id, geom  FROM ptv.stops\_routes\_mel  WHERE vehicle = 'Bus'  ),  -- Identify mesh blocks without bus stops  MeshBlocksWithoutStops AS (  SELECT m.mb\_code21  FROM ptv.mb2021\_mel m  LEFT JOIN BusStops b ON m.wkb\_geometry && b.geom  WHERE ST\_Contains(m.wkb\_geometry, b.geom) IS FALSE  AND m.mb\_cat21 = 'Residential'  ),  -- Calculate the blankspot percentage for each LGA  BlankspotStats AS (  SELECT  l.lga\_name\_2021 AS lga\_name,  COUNT(DISTINCT m.mb\_code21) AS total\_residential\_meshblocks,  COUNT(DISTINCT ms.mb\_code21) AS blankspots  FROM ptv.mb2021\_mel m  JOIN ptv.lga2021 l ON m.mb\_code21 = l.mb\_code\_2021  LEFT JOIN MeshBlocksWithoutStops ms ON m.mb\_code21 = ms.mb\_code21  WHERE m.mb\_cat21 = 'Residential'  AND l.lga\_name\_2021 LIKE 'Melbourne%'  GROUP BY l.lga\_name\_2021  )  -- Display the results  SELECT  lga\_name,  total\_residential\_meshblocks,  blankspots,  (blankspots::FLOAT / total\_residential\_meshblocks) \* 100 AS blankspot\_percentage  FROM BlankspotStats |
| --- |

Screenshot

|  |
| --- |

**Question 3.2.1**: Complete the following statistical data based on the result.

Note:

* The query and screenshot are not required for this section. You can write down your results directly
* If more than one suburb has the same percentage of blankspots, sort them by suburb name in ascending order.

| Criteria | Answer |
| --- | --- |
| Top 5 LGAs with the highest % of blankspots | 1. Yarra Ranges 2. Manningham 3. Bayside (Vic.) 4. Whitehorse 5. Greater Dandenong |
| Top 5 LGAs with the lowest % of blankspots | 1. Yarra 2. Melbourne 3. Stonnington 4. Port Phillip 5. Murrindindi |
| Average % of blankspots | 23.52455 % |

## **Task 4: Data Visualisation**

In this task, you are required to incorporate a heatmap visualization.

### 4.1 LGA Blankspot Analysis

In this task, you will put the blankspot percentage in the heatmap. Provide the segmentation as follow:

* + X ≤ 20 %
  + 20% < X ≤ 40%
  + 40% < X ≤ 60%
  + 60% < X ≤ 80%
  + X > 80%

Write an SQL query to create a table named 'lga\_blankspot' containing the blankspot percentages categorised by the LGA from the previous task 3.2 and sufficient spatial data. And attach a screenshot of the table contents.

Note: Ensure that this table is structured to facilitate the creation of a visual heat map specifically for the Melbourne region.

| CREATE TABLE ptv.lga\_blankspot AS  (  -- CTE (Common Table Expression) to filter out only bus stops from the stops\_routes\_mel table  WITH BusStops AS (  SELECT stop\_id, geom  FROM ptv.stops\_routes\_mel  WHERE vehicle = 'Bus'  ),  -- CTE to identify mesh blocks that are residential but don't have any bus stops  MeshBlocksWithoutStops AS (  SELECT m.mb\_code21  FROM ptv.mb2021\_mel m  LEFT JOIN BusStops b ON m.wkb\_geometry && b.geom  WHERE ST\_Contains(m.wkb\_geometry, b.geom) IS FALSE  AND m.mb\_cat21 = 'Residential'  ),  -- CTE to calculate blankspot statistics for each LGA  BlankspotStats AS (  SELECT  lga.lga\_name\_2021 AS lga\_name,  ST\_Union(mb.wkb\_geometry) AS lga\_geom, -- Aggregating the geometries of the mesh blocks for each LGA  COUNT(DISTINCT mb.mb\_code21) AS total\_residential\_meshblocks,  COUNT(DISTINCT CASE WHEN mws.mb\_code21 IS NOT NULL THEN mb.mb\_code21 ELSE NULL END) AS blankspots  FROM ptv.mb2021\_mel mb  JOIN ptv.lga2021 lga ON mb.mb\_code21 = lga.mb\_code\_2021  LEFT JOIN MeshBlocksWithoutStops mws ON mb.mb\_code21 = mws.mb\_code21  WHERE mb.mb\_cat21 = 'Residential'  GROUP BY lga.lga\_name\_2021  )  -- Selecting the final results from the BlankspotStats CTE, computing the blankspot percentage, and including the aggregated geometry  SELECT  lga\_name,  lga\_geom,  total\_residential\_meshblocks,  blankspots,  (blankspots::FLOAT / total\_residential\_meshblocks) \* 100 AS blankspot\_percentage  FROM BlankspotStats  ); |
| --- |

Screenshot

|  |
| --- |

Provide the screenshot for the heatmap by using QGIS. Please note that the heatmap is map-based and visually represents the distribution of blankspot percentages across different LGAs in the Melbourne region.

Remember to include appropriate labels, titles, and legends in your visualizations to make them easy to understand (Updated 05/10/2023)

|  |
| --- |

\*\*\*End\*\*\*