Assignment-1, GEO-1002

Ming-Chieh Hu, Dimitrios Lioumis, Neelabh Singh

October 2024

Project Description

This multi-part assignment we use GRASS GIS to perform 3 types of (geo)spatial analyses that are generally carried out using GIS software.

1. Solar analyses
2. Network analyses
3. Data integration

Each part is split into smaller exercises.

Assignment Group Members

In alphabetical order by surname:

SURNAME 1: Hu
NAME 1: Ming-Chieh
STUDENT ID 1: 6186416
STUDY PROGRAMME 1: Geomatics

SURNAME 2: Lioumis
NAME 2: Dimitrios
STUDENT ID 2: 6213944
STUDY PROGRAMME 2: Geomatics

SURNAME 3: Singh
NAME 3: Neelabh
STUDENT ID 3: 6052045
STUDY PROGRAMME 3: Geomatics

Fraud And Plagiarism Awareness

We confirm herewith that we have read and we are aware of the TU Delft rules (and consequences) regarding fraud and plagiarism, as written in https://www.tudelft.nl/en/student/my-study-me/rules-guidelines-and-participation/fraud-plagiarism

Work Allocation Summary

Part 1: Ming-Chieh Hu, Dimitrios Lioumis, approx. 15 hours
Part 2: Neelabh Singh, Ming-Chieh Hu, approx. 16 hours
Part 3: Dimitrios Lioumis, Neelabh Singh, approx. 14 hours
Report Assembling: Ming-Chieh Hu, approx. 4 hours

Contents

1 Part 1: Solar Analyses

1.1 Exercise 1.1

Load GRASS GIS, select GRASS location "trento" and GRASS mapset "dtm". Find the following datasets in folder /data/trento_data, load them into mapset "dtm" and name the resulting GRASS layers trento_extents and dtm, respectively:

- trento_extents.shp: A single polygon representing the area that will be used to set the region in GRASS.
- dtm_trento.asc: An ASCII grid file representing the DTM of the study area in Trento.

Once you are done with the import, be sure to set the working region to correspond to the trento_extents layer. Set the grid resolution to 1m, for now. Visualise the layers, for the dtm layer display the legend, check the values range, etc.

Import data

```
v.import -l input=/vector/filepath output=trento_extents
r.in.gdal -o -r input=/raster/filepath output=dtm
```

Set region to match trento_extents

```
g.region res=1
g.region vector=trento_extents@dtm
g.region -p raster=dtm@dtm
```

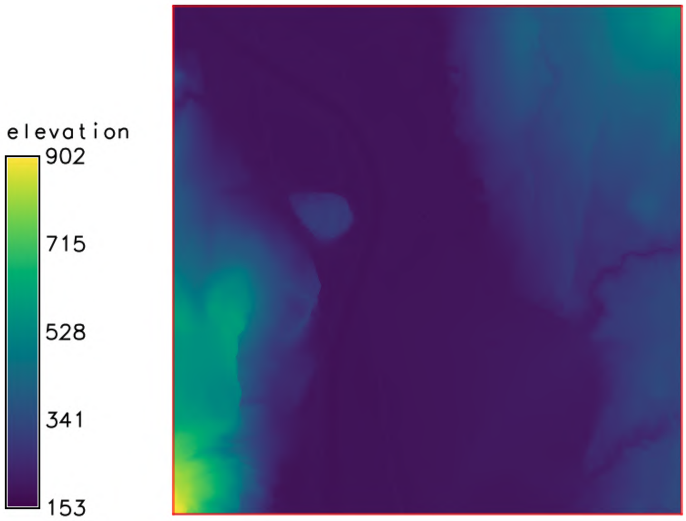


Figure 1: imported dtm raster map

1.2 Exercise 1.2

Create now a new mapset (g.mapset) named "solar_25", and start working in it. Set the grid resolution to 25m, the region extents need to correspond to the trento_extents layer. Resample (r.resample) the dtm layer and create a new raster dtm_25. Using dtm_25, compute the slope and aspect maps, called slope_25 and aspect_25, respectively. Add to the report screenshots of the three newly generated maps.

Resample dtm to resolution 25

```
g.mapset -c mapset=solar_25
g.region raster=dtm@dtm res=25
g.region -p
r.resample input=dtm@dtm output=dtm_25
```

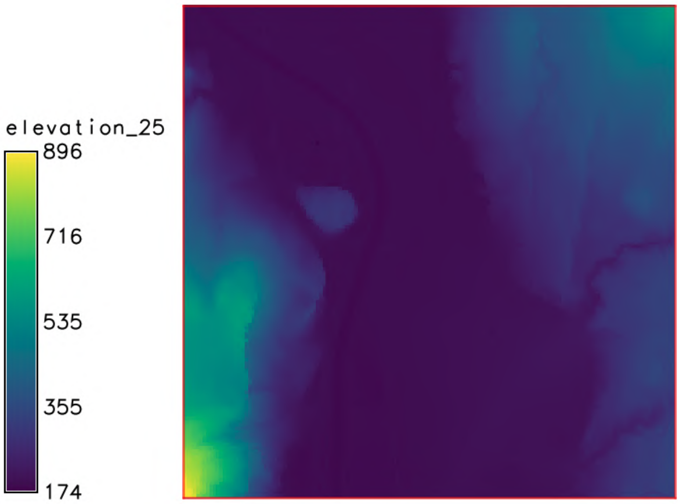


Figure 2: resampled dtm_25 raster map

Compute the slope and aspect maps

```
r.slope.aspect elevation=dtm_25 slope=slope_25 aspect=aspect_25
```

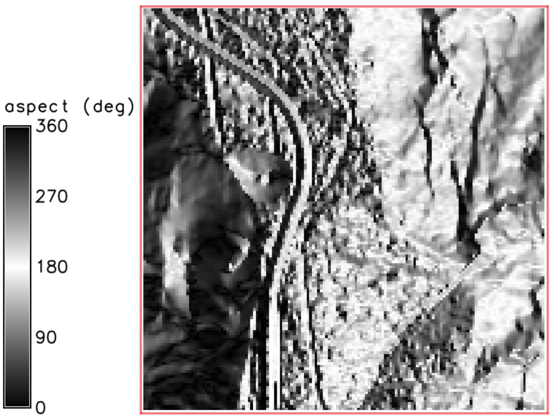


Figure 3: aspect map generated from dtm_25

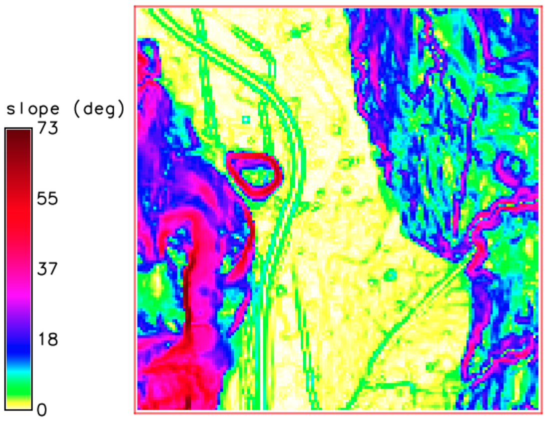


Figure 4: slope map generated from dtm_25

1.3 Exercise 1.3

Keep working at 25-m resolution and compute 8 horizon maps (r.horizon, read the documentation), one for each cardinal and intercardinal direction. Check that the maps are generated using a prefix (e.g. "horizon"). At the end, the layers should be like: horizon_000, horizon_045, . . . , etc.

Compute 8 horizon height maps

```
r.horizon elevation=dtm_25 step=45 start=0 end=360 output=horizon
```

Here we use the **step**, **start**, **end** parameter to generate 8 horizon maps at once.

Figure 5: horizon maps in 8 different directions

1.4 Exercise 1.4

Load the r.sun module and read the documentation carefully. First, you will compute the global solar irradiation for a single day: the 15th January. In order to do this, you will need the dtm, slope, aspect and horizon maps computed before (all at 25 m resolution). For the rest, you can use the default values for the Linke coefficient (3.0) and ground albedo (0.2). The output layer will be called irr_global_015 (15 is the index of the day with respect to a 365-day year). Once you are done, explore the map, understand what "numbers" are contained. What are the units of measure of map irr_global_015?

Compute global solar irradiation for 15th January

```
r.sun elevation=dtm_25 aspect=aspect_25 slope=slope_25 horizon_basename=horizon horizon_step=45
day=15 glob_rad=irr_global_015
```

the unit used in this map is Wh/m^2 (Watt-hour per square meter). Watt-hour is a unit of energy equal to one watt of power expended (used) for one hour.

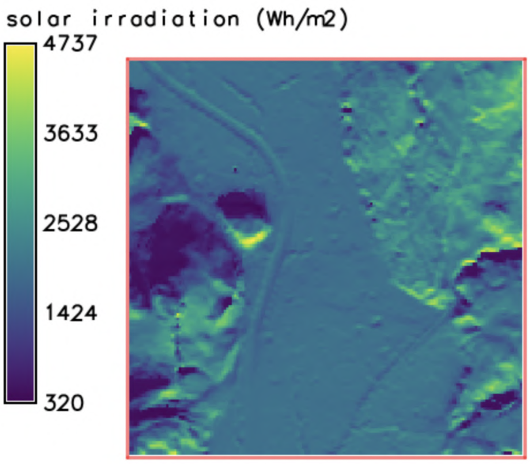


Figure 6: solar irradiation map of 15th January

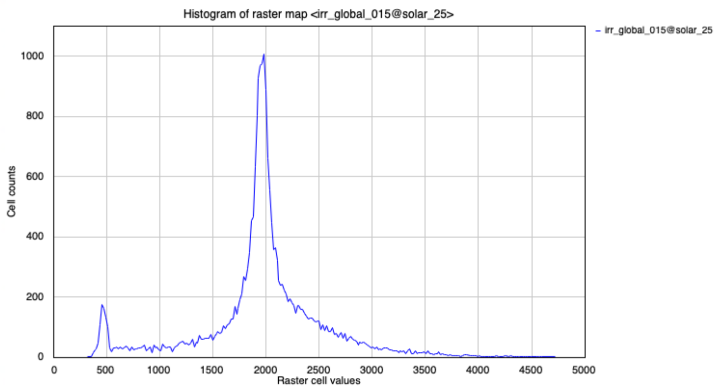


Figure 7: histogram of irr_global_015 values

In order to obtain the irradiation maps for the whole year (you will do this later), you will first run r.sun a number of times to compute the solar irradiation in different periods of the year. For the sake of the computational time needed, we will approximate this operation in this exercise by computing only 12 daily values. Run r.sun accordingly, and store for each day a raster layer named irr_global_xxx, where xxx is the index of the corresponding day. Visualise the maps and check the legends.

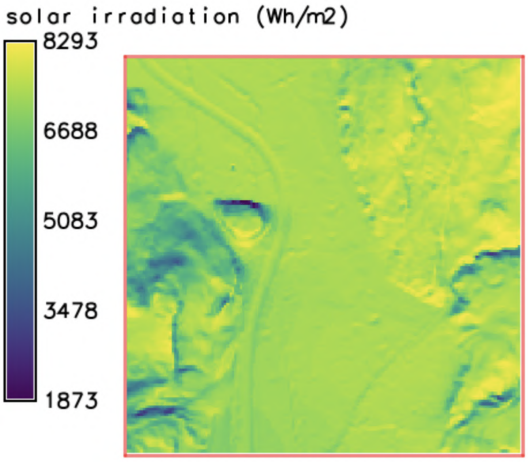


Figure 8: solar irradiation map of 15th August

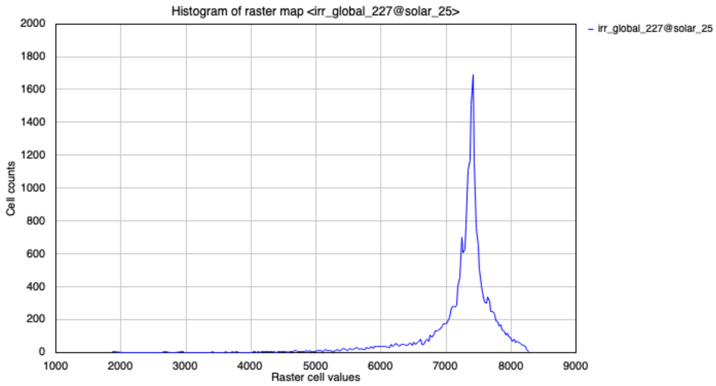


Figure 9: histogram of irr_global_227 values

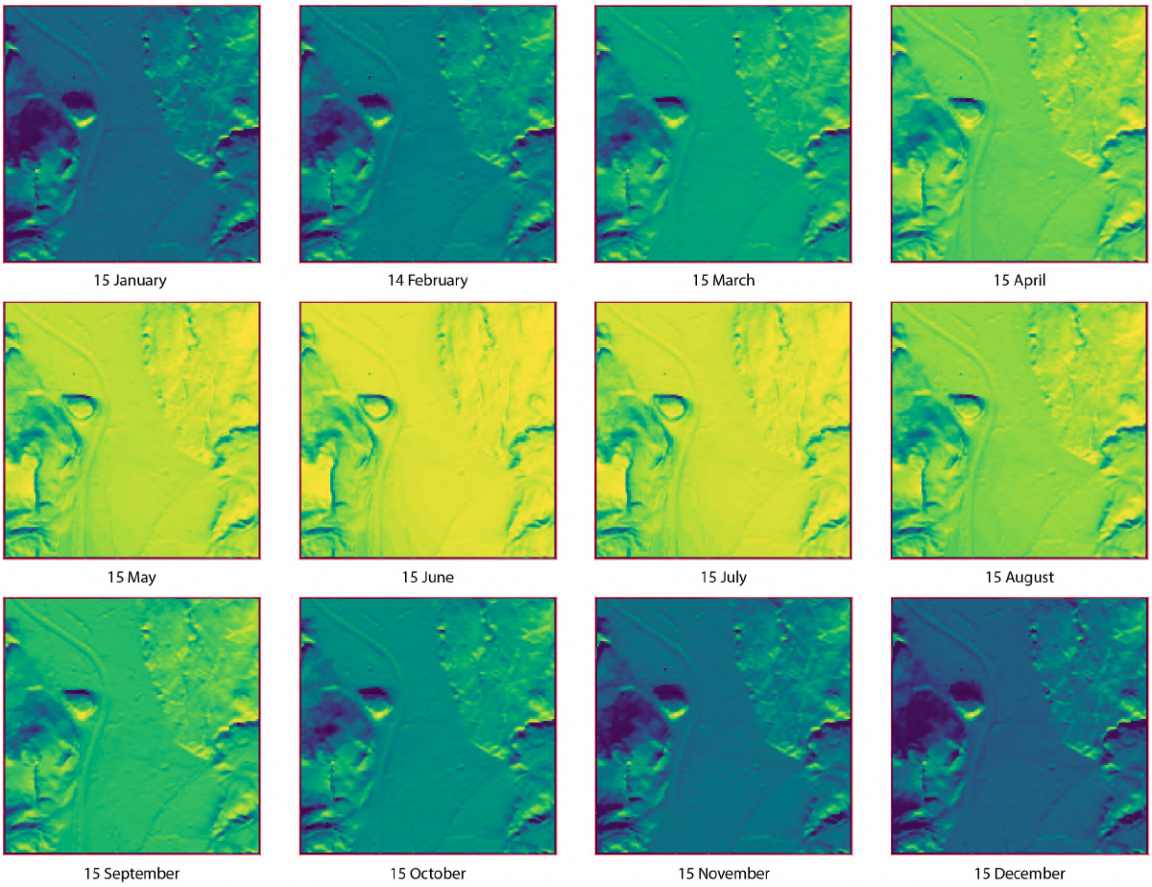


Figure 10: global solar irradiation map in 12 months

Comparing histograms and figure 6, 7, 8, 9 we can see that the solar irradiation values varies a lot, and they have different distribution as well. In figure 10, the irradiation maps for all 12 months are shown, but they are not directly comparable due to inconsistent color schemes. To ensure accurate visual comparisons, we need to standardize the value-to-color mapping across all maps.

1.5 Exercise 1.5

Visualise the maps (and check the legends). What do you notice? Compare, for example, December and June. Force all the irr_glob_xxx maps to use the same colour coding. For this, you need first to find out which the overall maximum and minimum values all over the year are (according to the maps you computed), and then define a file containing the colour rules to be assigned to all rasters.

Check the maximum and minimum values from raster maps

```
r.univar map=
irr_global_015,irr_global_045,irr_global_074,irr_global_105,irr_global_135,irr_global_166,
irr_global_196,irr_global_227,irr_global_258,irr_global_288,irr_global_319,irr_global_349
```

```
...
Of the non-null cells:
----------------------
n: 270000
minimum: 283.232
maximum: 9086.95
range: 8803.72
mean: 5188.18
mean of absolute values: 5188.18
standard deviation: 2545.42
variance: 6.47916e+06
variation coefficient: 49.0618%
sum: 1400809893.45132
```

The corresponding color rule file

```
200 violet
1500 blue
3200 cyan
4700 green
6200 yellow
7700 orange
9100 red
```

Remap all the irr_glob_xxx maps

```
r.colors map=
irr_global_015,irr_global_045,irr_global_074,irr_global_105,irr_global_135,irr_global_166,
irr_global_196,irr_global_227,irr_global_258,irr_global_288,irr_global_319,irr_global_349
rules=/color/rule/filepath
```

Plot the legend in scale of remapped value

```
d.legend title="solar irradiation (Wh/m2)" range=200,9100 raster=irr_global_015
```

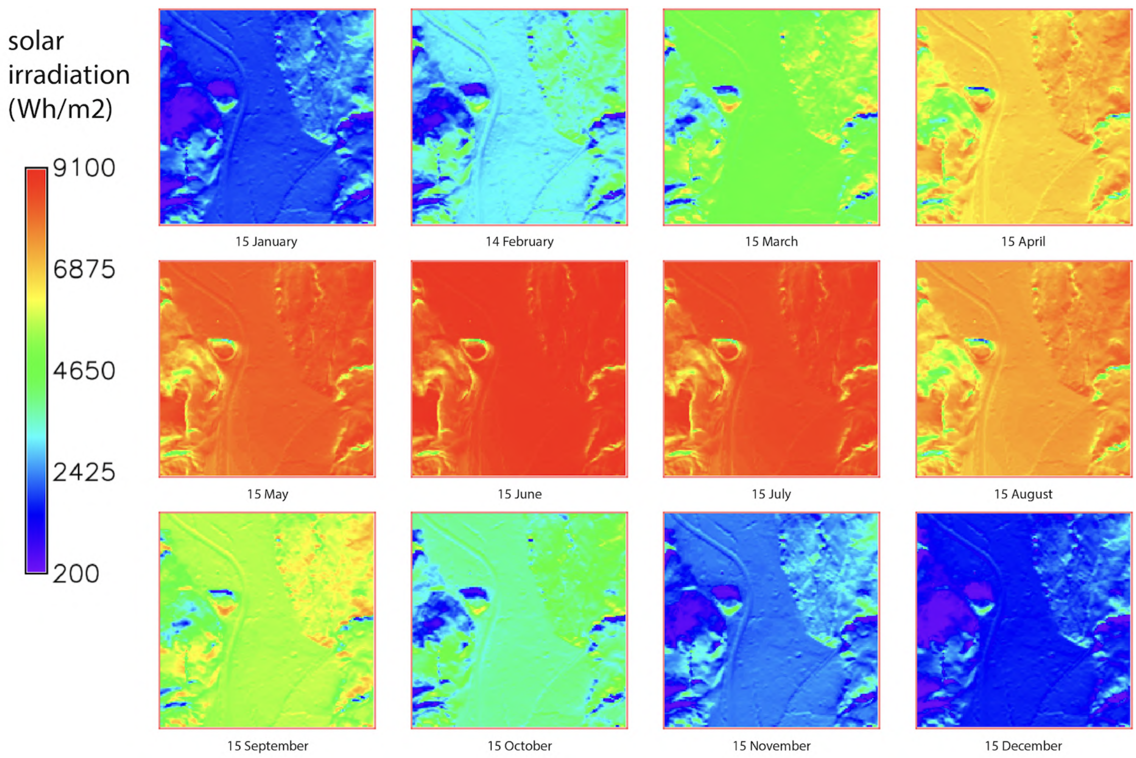


Figure 11: recolored global solar irradiation map in 12 months

1.6 Exercise 1.6

Using map algebra (r.mapcalc), compute the annual global solar irradiation map in $MWh/(m^2 * a)$ and the average daily global irradiation in $kWh/(m^2 * d)$. The new layers are called irr_glob_ and irr_glob_avg_day, respectively. Consider a standard year of 365 days. Add proper screenshots (with legend) of these maps.

Additionally, create a map called irr_glob__gt_19 which contains only those areas that have a value of annual global solar irradiation $\geq 1.9\ MWh/(m^2 * a)$. Add a screenshot of this map (with legend) to the report.

Finally, if you haven't already done, create a binary map, called irr_glob__gt_19_bin, that consists of 1 values for all cells of irr_glob__gt_19 that are not null, and 0 otherwise. You will need this map later.

For each month, the total solar irradiation is calculated by multiplying the daily irradiation by the number of days in that month, denoted as $I_{month} = I_{day} \times N_{day}$. Example for January: $I_{Jan} = I_{Jan\ 15th} \times 31$. The total annual global solar irradiation is the summation of the solar irradiation for each month, denoted as: $I_{year} = \Sigma_{i=1}^{12} I_{month\ i}$. Where Monthly Irradiation, is the irradiation map for the i^{th} month.

Calculate the annual global solar irradiation map

```
r.mapcalc expression="irr_glob_year = (
irr_global_015 * 31 + irr_global_045 * 28 + irr_global_074 * 31 +
irr_global_105 * 30 + irr_global_135 * 31 + irr_global_166 * 30 +
irr_global_196 * 31 + irr_global_227 * 31 + irr_global_258 * 30 +
irr_global_288 * 31 + irr_global_319 * 30 + irr_global_349 * 31) / 1000000"
```

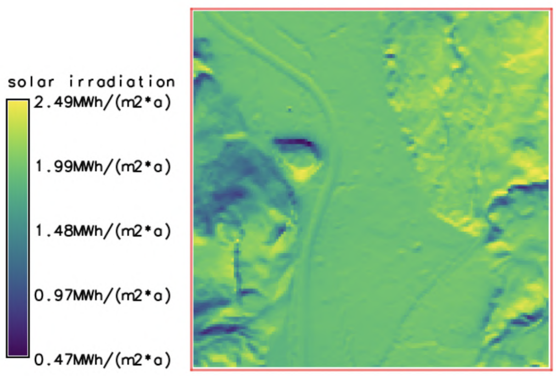


Figure 12: annual global solar irradiation map

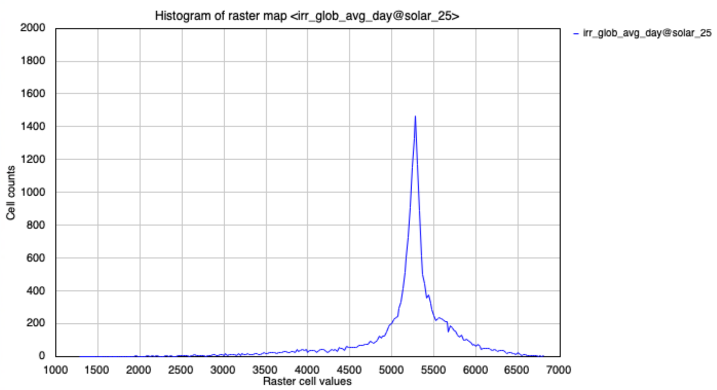


Figure 13: histogram of irr_glob_year values

Calculate the average daily global irradiation map

```
r.mapcalc expression="irr_glob_avg_day = irr_glob_year * 1000 / 365"
```

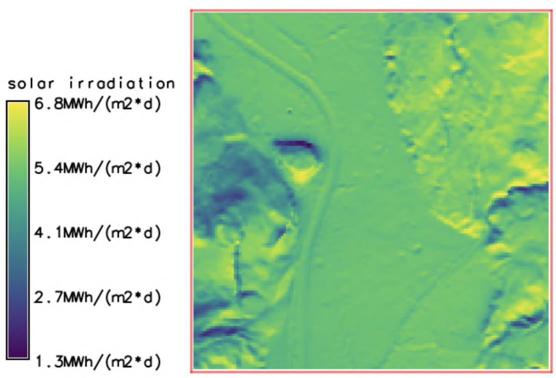


Figure 14: average daily global irradiation map

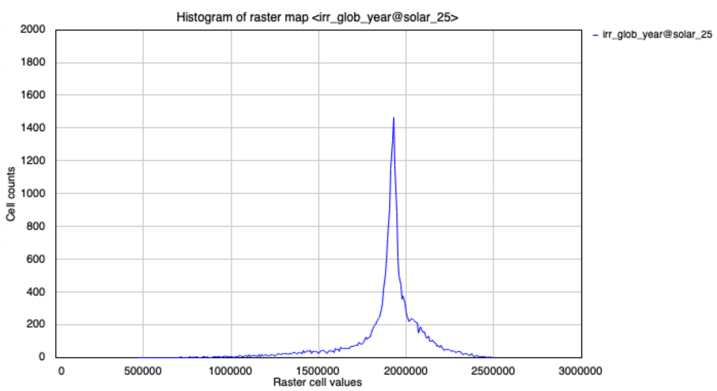


Figure 15: histogram of irr_glob_avg_day values

Filter values from annual global solar irradiation

```
r.mapcalc expression="if((irr_glob_year >= 1.9), irr_glob_year,null())"
```

Create binary map for filtered result

```
r.mapcalc expression="if((irr_glob_year >= 1.9),1,0)"
```

Show statistics

```
r.univar map=irr_glob_year_gt_19
```

```
total null and non-null cells: 23104
total null cells: 8277
Of the non-null cells...
```

What are the extents (in m^2) of the selected area(s)?
The statistics showed there are 23104 cells in this square area and 8277 of them are not selected (with NULL value). Thus, we conducted the selected area is $14827 * 25 * 25 = 9266875m^2$.

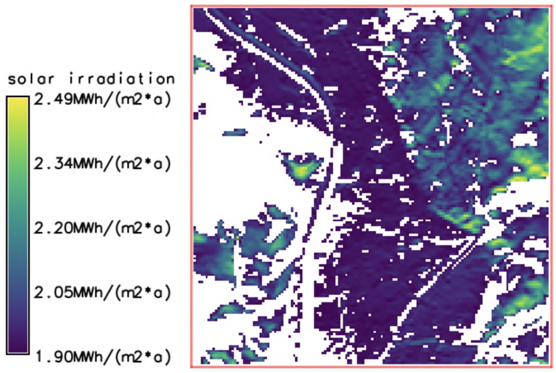


Figure 16: filtered annual global solar irradiation map

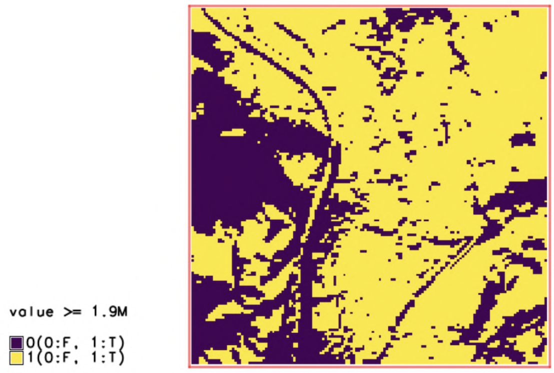


Figure 17: filtered binary map

1.7 Exercise 1.7

You will now partially repeat the same solar analyses done before, but investigating the role of the raster resolution. Create a new mapset called "solar_5", and set the raster resolution at 5 m.
Always working at 5 m resolution, repeat the steps carried out before to resample the dtm and to generate the 8 horizon maps. If you have created a script before, this exercise will be rather simple because you only need to slightly adapt the script for the new mapset and resolution, and to rename the new layers accordingly (e.g. dtm_5, slope_5, aspect_5, etc.)
When you are done:

- Add a new image representing layer dtm_5 (with legend)
- Add two representative screenshots of the new horizon maps, i.e. those computed for angle 00 and 90 (with legends).

Create nea mapset and resample dtm to resolution 5

```
g.mapset -c mapset=solar_25
g.region raster=dtm@dtm res=25
g.region -p
r.resample input=dtm@dtm output=dtm_25
```

Compute the slope and aspect maps

```
r.slope.aspect elevation=dtm_25 slope=slope_25 aspect=aspect_25
```

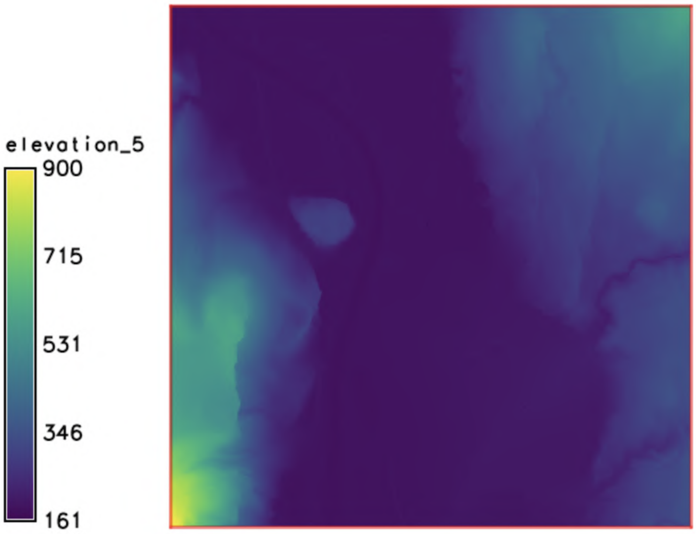


Figure 18: resampled dtm_5 raster map

Compute 8 horizon height maps

```
r.horizon elevation=dtm_5 step=45 start=0 end=360 output=horizon
```

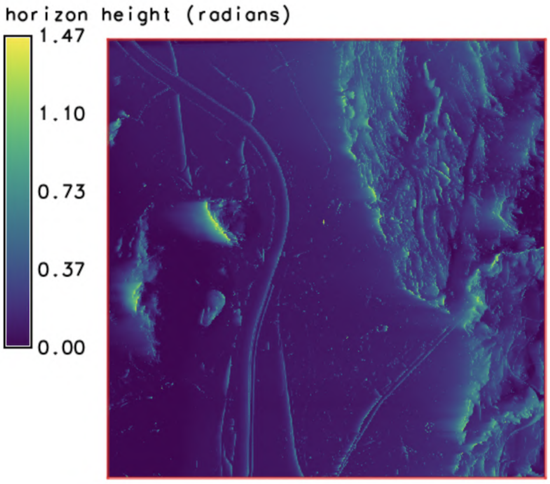


Figure 19: horizon height map from the east (0 degree)

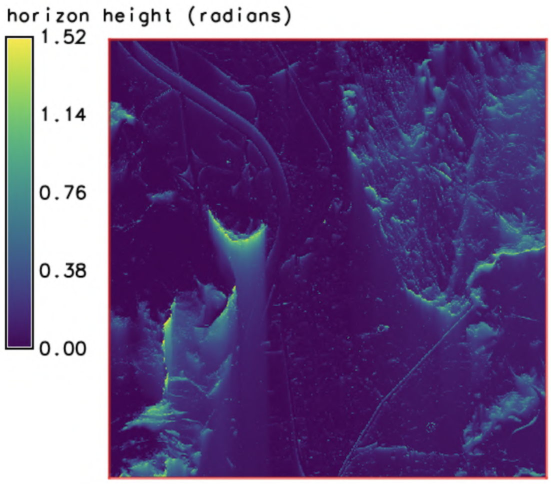


Figure 20: horizon height map from the north (90 drgree)

1.8 Exercise 1.8

Always working at 5m resolution, repeat now the necessary steps described before to compute the 12 monthly values of total solar irradiation, and to obtain the "new" irr_glob_year and irr_glob_avg_day, however at 5 m resolution. Again, this exercise can be carried out by adapting the script written for the previous exercises. Be aware that the computation time will be a bit longer due to the increased resolution. Therefore, check carefully the script before running it!

- Add to the report the screenshots of two "new" solar maps at 5m resolution, i.e. for June and December (with legends).
- Compute and add to the report the maximum and minimum values of daily global solar irradiation computed with the new maps at 5 m resolution, similarly as in Exercise 1.5 (you do not need to redo the colour styling).
- Add to the report the screenshots of the "new" irr_glob_year and irr_glob_avg_day maps at 5 m resolution (with legends).
- Compute and add to the report the screenshot of the "new" irr_glob_year_gt_19 layer at 5 m resolution (with legend).
- Finally, compute and add a screenshot to the report of the "new" binary map irr_glob_year_gt_19_bin, obtained from the irr_glob_year_gt_19 at 5 m resolution computed in this exercise.

Compute global solar irradiation for 12 months

```
r.sun elevation=dtm_5 aspect=aspect_5 slope=slope_5 horizon_basename=horizon horizon_step=45
day=15 glob_rad=irr_global_015
r.sun elevation=dtm_5 aspect=aspect_5 slope=slope_5 horizon_basename=horizon horizon_step=45
day=45 glob_rad=irr_global_045
...
r.sun elevation=dtm_5 aspect=aspect_5 slope=slope_5 horizon_basename=horizon horizon_step=45
day=349 glob_rad=irr_global_349
```

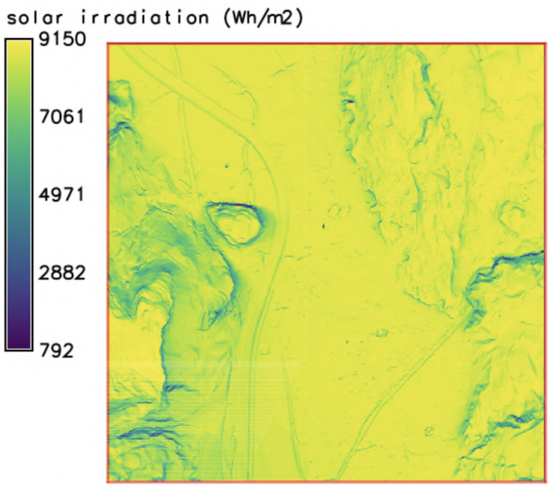


Figure 21: solar irradiation map of 15th June

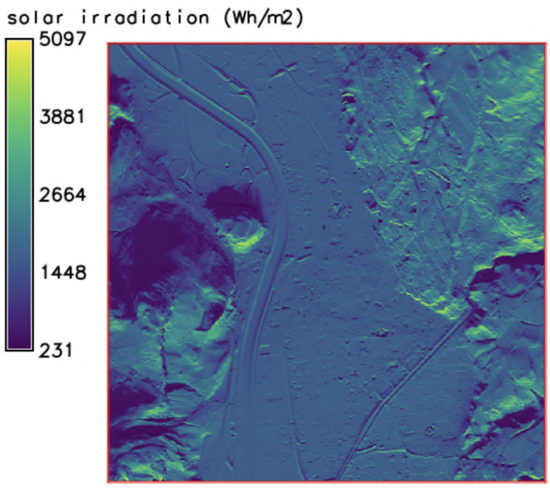


Figure 22: solar irradiation map of 15th December

Check the maximum and minimum values from raster maps

```
r.univar map=
irr_global_015,irr_global_045,irr_global_074,irr_global_105,irr_global_135,irr_global_166,
irr_global_196,irr_global_227,irr_global_258,irr_global_288,irr_global_319,irr_global_349
```

```
...
Of the non-null cells:
----------------------
n: 6894756
minimum: 230.861
maximum: 9150.4
range: 8919.54
mean: 5132.38
mean of absolute values: 5132.38
standard deviation: 2551.53
variance: 6.51028e+06
variation coefficient: 49.7143%
sum: 35386517311.7231
```

Calculate the annual global solar irradiation map

```
r.mapcalc expression="irr_glob_year = (
irr_global_015 * 31 + irr_global_045 * 28 + irr_global_074 * 31 +
irr_global_105 * 30 + irr_global_135 * 31 + irr_global_166 * 30 +
irr_global_196 * 31 + irr_global_227 * 31 + irr_global_258 * 30 +
irr_global_288 * 31 + irr_global_319 * 30 + irr_global_349 * 31) / 1000000"
```

Calculate the average daily global irradiation map

```
r.mapcalc expression="irr_glob_avg_day = irr_glob_year * 1000 / 365"
```

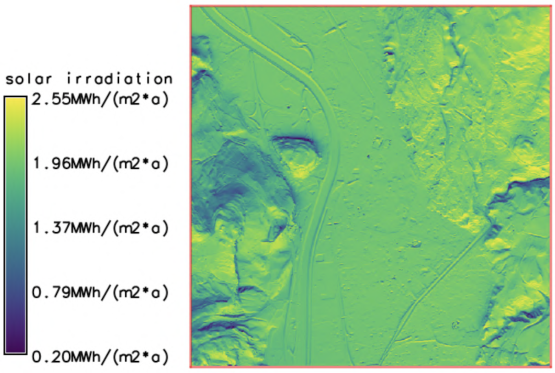


Figure 23: annual global solar irradiation map

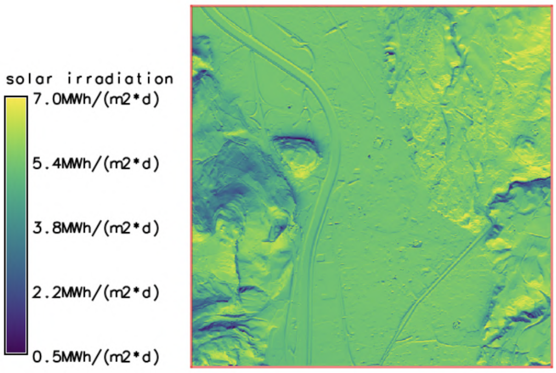


Figure 24: average daily global solar irradiation map

Filter values from annual global solar irradiation

```
r.mapcalc expression="if((irr_glob_year >= 1.9), irr_glob_year,null())"
```

Create binary map for filtered result

```
r.mapcalc expression="if((irr_glob_year >= 1.9),1,0)"
```

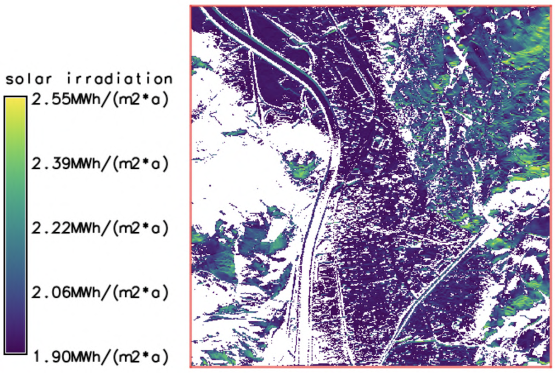


Figure 25: filtered annual global solar irradiation map

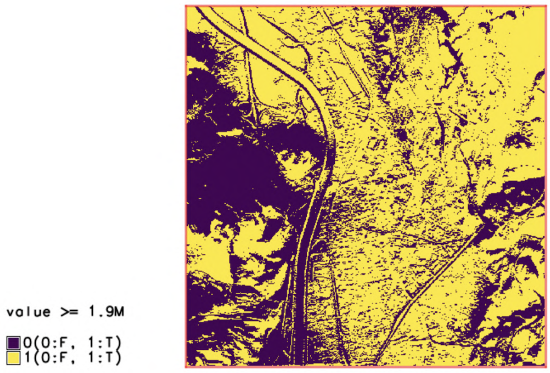


Figure 26: filtered binary map

1.9 Exercise 1.9

Working at 5 m resolution, compute a new map, called irr_glob_year_gt_19_bin_diff that represents the difference between irr_glob_year_gt_19_bin@solar_25 and irr_glob_year_gt_19_bin@solar_5. Add a screenshot (with legend) to the report.

Answer the following questions:

- What can you say about map irr_glob_year_gt_19_bin_diff?
- What can you say about the minimum and maximum values of daily global solar irradiation computed at 25 and at 5 m resolution, respectively? What could be the reason?
- What can you say about the irr_glob_year and irr_glob_avg_day maps computed at 25 and at 5 m resolution, respectively?
- What can you say about the values of total area with an annual global solar irradiation $\geq 1.9\ MWh/(m^2 * a)$ and computed at 25 and at 5 m resolution, respectively? How much is the difference, in %?

Calculate the difference between 2 binary maps

```
r.mapcalc expression="xor(irr_glob_year_gt_19_bin@solar_5, irr_glob_year_gt_19_bin@solar_25)"
```

The difference map irr_glob_year_gt_19_bin_diff (figure 27) is showing some similar to those observed in the aspect map (figure 3). Resampling at a coarser resolution leads to a slight loss of detail, particularly at the boundaries where elevation changes. This effect is most notable along the edges of distinct objects, textures, mountains, rivers, and other features.

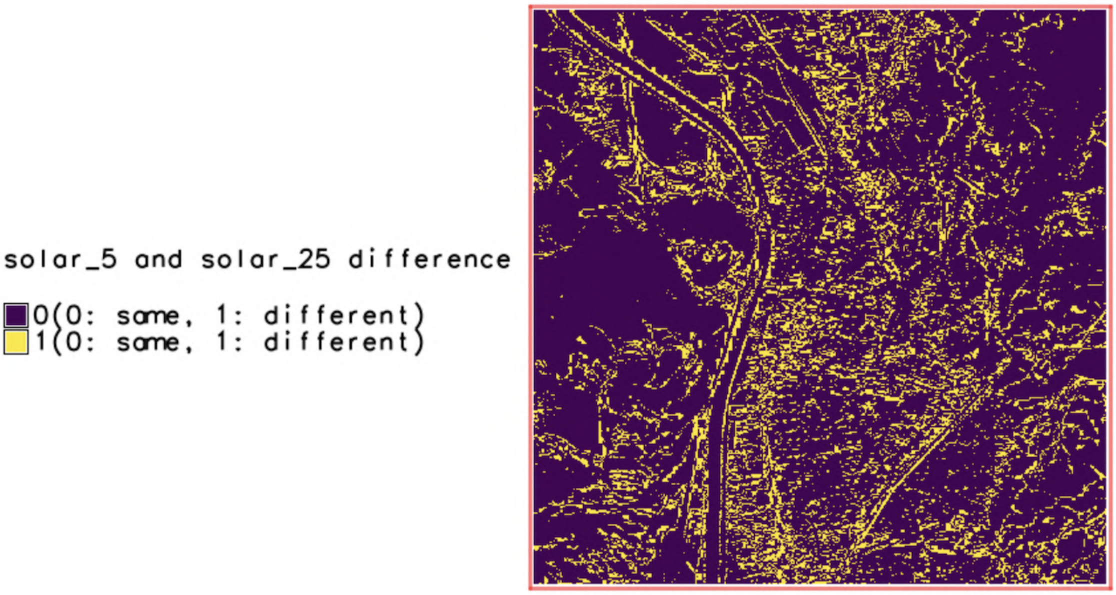


Figure 27: difference map

Check the maximum and minimum values from irr_glob_avg_day

```
r.univar map=irr_glob_avg_day@solar_5
```

```
...
Of the non-null cells:
----------------------
n: 574563
minimum: 0.549077
maximum: 6.97799
range: 6.42891
mean: 5.14233
mean of absolute values: 5.14233
standard deviation: 0.675842
variance: 0.456762
variation coefficient: 13.1427%
sum: 2954593.9249717
```

```
r.univar map=irr_glob_avg_day@solar_25
```

```
...
Of the non-null cells:
----------------------
n: 22500
minimum: 1.28107
maximum: 6.82602
range: 5.54495
mean: 5.19824
mean of absolute values: 5.19824
standard deviation: 0.579874
variance: 0.336253
variation coefficient: 11.1552%
sum: 116960.44225955
```

For 5m resolution the minimum value is lower and the maximum value is higher. We think the main reason of this phenomena is that the sampling method tends to reduce some of the extremes. Finer resolution (e.g., 5 m) tends to capture more extreme minimum and maximum values due to its ability to model smaller, more localized terrain and solar variations. Resample to a finer coarser resolution (e.g., 25 m), especially with nearest neighbor algorithm, tends to smooth out variations, leading to less extremes in the map.

Also, according to GRASS documentation of r.resample: *The method by which resampling is conducted is "nearest neighbor" (see r.neighbors). The resulting raster map layer will have the same resolution as the resolution of the current geographic region (set using g.region).*

Check the maximum and minimum values from irr_glob_year

```
r.univar map=irr_glob_year@solar_5
```

```
...
Of the non-null cells:
----------------------
n: 574563
minimum: 0.200413
maximum: 2.54697
range: 2.34655
mean: 1.87695
mean of absolute values: 1.87695
standard deviation: 0.246682
variance: 0.0608521
variation coefficient: 13.1427%
sum: 1078426.78259648
```

```
r.univar map=irr_glob_year@solar_25
```

```
...
Of the non-null cells:
----------------------
n: 22500
minimum: 0.467592
maximum: 2.4915
range: 2.02391
mean: 1.89736
mean of absolute values: 1.89736
standard deviation: 0.211654
variance: 0.0447974
variation coefficient: 11.1552%
sum: 42690.5614324212
```

According to the formula we used, irr_glob_avg_day has similar proportions as irr_glob_year but divided by 365 and used different unit. We can see the variation coefficient ($VC = \frac{\sigma}{\mu}$) is the same in both irr_glob_avg_day and irr_glob_year maps (in same resolution).

Check the values from irr_glob_year

```
r.univar map=irr_glob_year_gt_19@solar_5
```

```
total null and non-null cells: 577599
total null cells: 228451
Of the non-null cells:
----------------------
n: 349148
...
```

```
r.univar map=irr_glob_year_gt_19@solar_25
```

```
total null and non-null cells: 23104
total null cells: 8277
Of the non-null cells:
----------------------
n: 14827
...
```

Calculate area
Area with annual global solar irradiation $\geq$ 1.9 MWh/(m2*a) in 5m resolution: $area = 349148 * 5 * 5 = 8728700(m^2)$.
Area with annual global solar irradiation $\geq$ 1.9 MWh/(m2*a) in 25m resolution: $area = 14827 * 25 * 25 = 9266875(m^2)$.
The difference of these values calculated in different resolutions is $(9266875 - 8728700)/9266875 = 5.8\%$

At a 25m resolution, the averaging effect smooths out small-scale variations, resulting in more areas with solar irradiation values greater than 1.9 MWh/(m2*a). In contrast, the finer detail at 5m resolution captures these variations more accurately, showing more areas with lower irradiation values and reducing the total area that exceeds the threshold.

1.10 Exercise 1.10

Write a short reflection (max 200 words) on the type of solar analyses you carried out on this test site in Trento.

- What have you learned?
- What are, in your opinion, the shortcomings of this approach?
- How and in which parts could it be improved?

Through this exercise, We learned how to perform a solar analysis in GRASS GIS, including resampling, generating slope and aspect maps, and working with data manipulation and visualization. We also gained insight into how resolution affects the accuracy of irradiation values—coarser resolution maps reveal broader trends, while finer resolution maps capture more local variations.

However, the analysis has some shortcomings. First, calculating annual solar irradiation using only 12 monthly raster maps is efficient but not highly accurate. Second, resolution significantly impacts the results, as seen in the differences between 5m and 25m resolutions (which also affect computation time).

To improve this analysis, we need to carefully choose the appropriate resolution to balance insights into both trends and local variations without compromising performance. Additionally, incorporating more global irradiation or weather data could enhance the accuracy of the predictions.

2 Part 2: Network Analyses

2.1 Exercise 2.1

Launch GRASS GIS and create a new location “padova”. EPSG code is 3003. Create a new mapset called “geo1002”. Import the shapefiles street_line.shp and street_crossing.shp into vectors street_lines and street_crossing, using v.in.ogr from the GRASS command console. Upon import, pay attention that field “cat” is used to generate the categories in the GRASS layer (it is one of the import optional parameters!). Categories, in GRASS, correspond to IDs and are used to identify uniquely a feature.

Add a screenshot of each imported dataset to the report. For the street_crossing layer, create a map that shows not only the geometries, but also the associated cat values. In the latter case, if the density of crossings is too dense, you can zoom in and create a screenshot of a portion of the map, so that it is still readable.

Create location “pavoda” and mapset “geo1002” with EPSG 3003

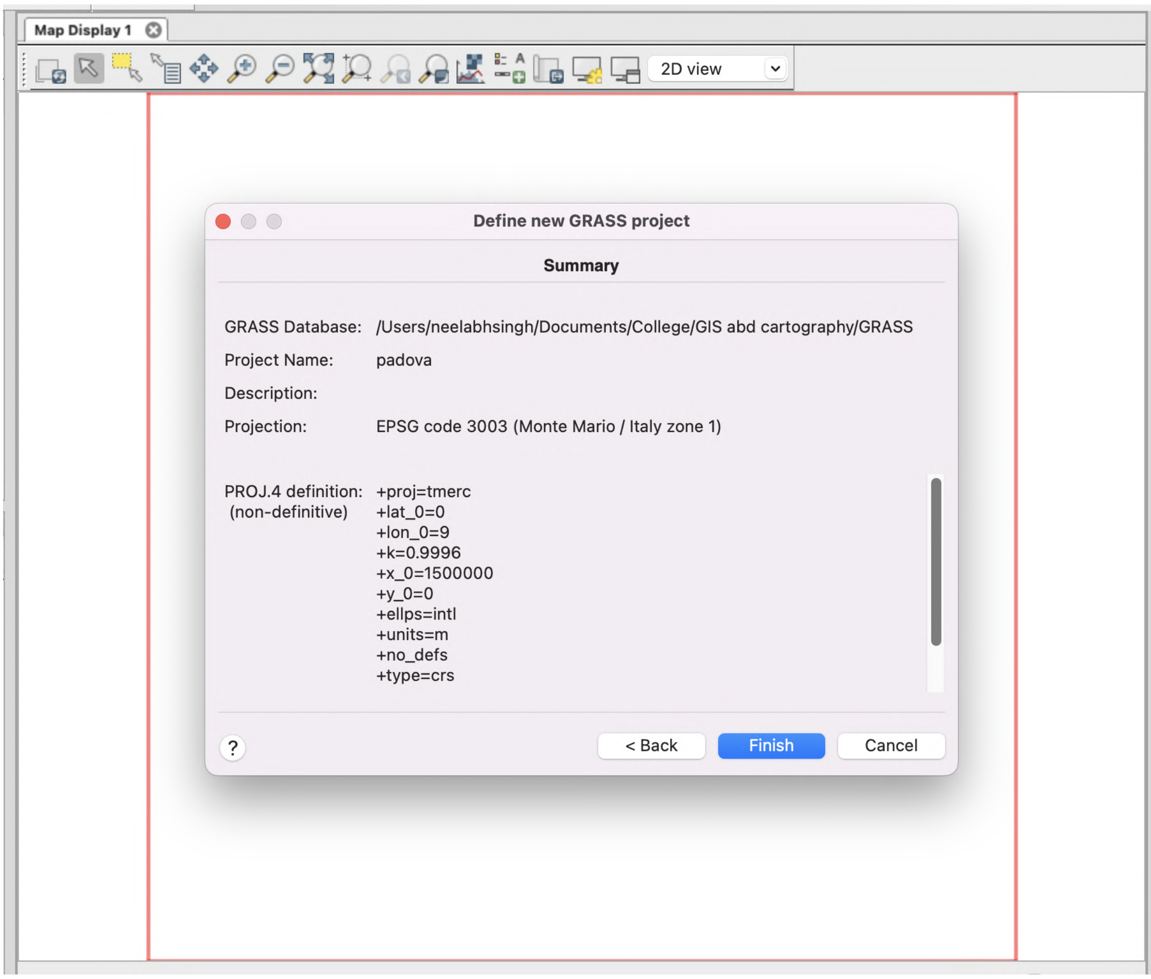


Figure 28: create new location “pavoda” with EPSG code 3003

Visualise street line layer and keep cat as key value

```
v.in.ogr input="/street_line/filepath" output=street_lines key="cat"
```

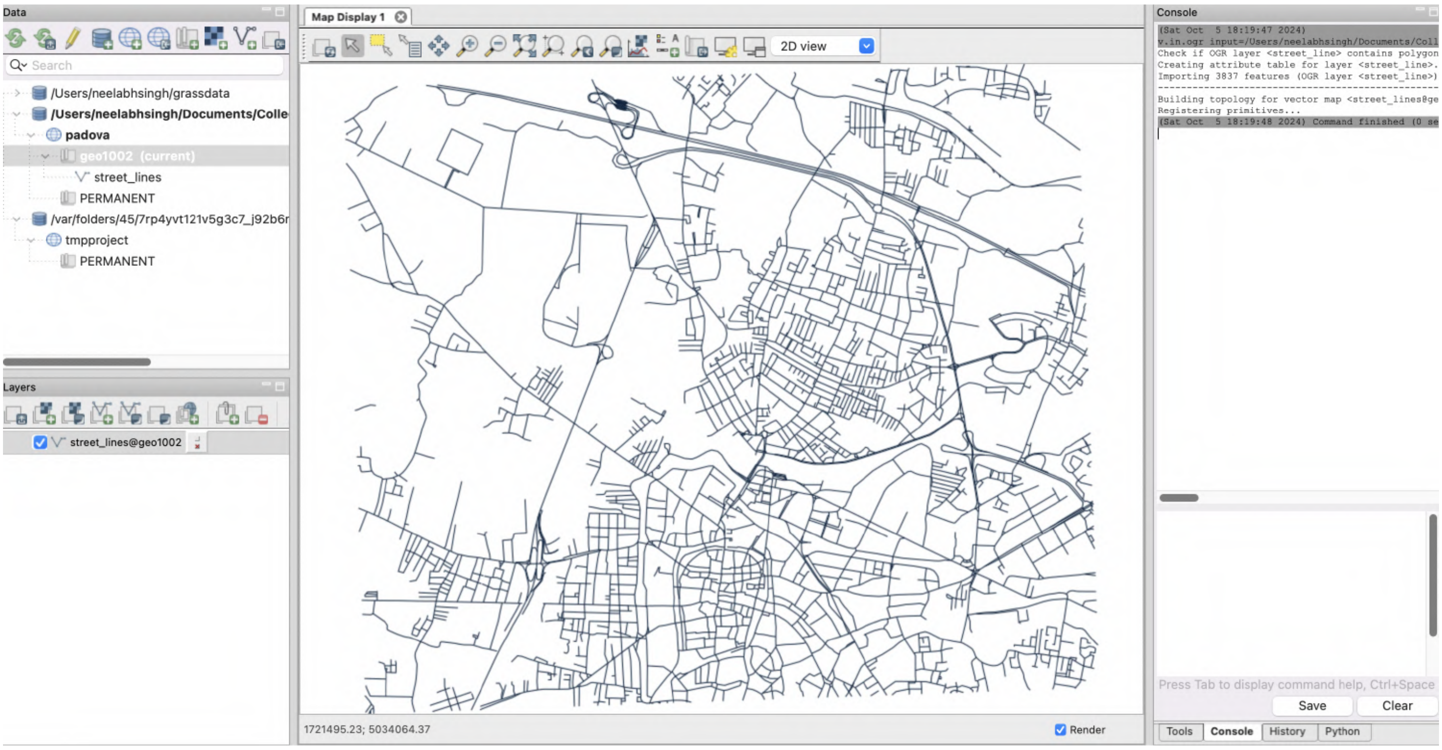


Figure 29: a new vector map added in the project named as street_line

Visualise street crossing layer and keep cat as key value

```
v.in.ogr input="/street_crossing/filepath" output=street_crossing key="cat"
```

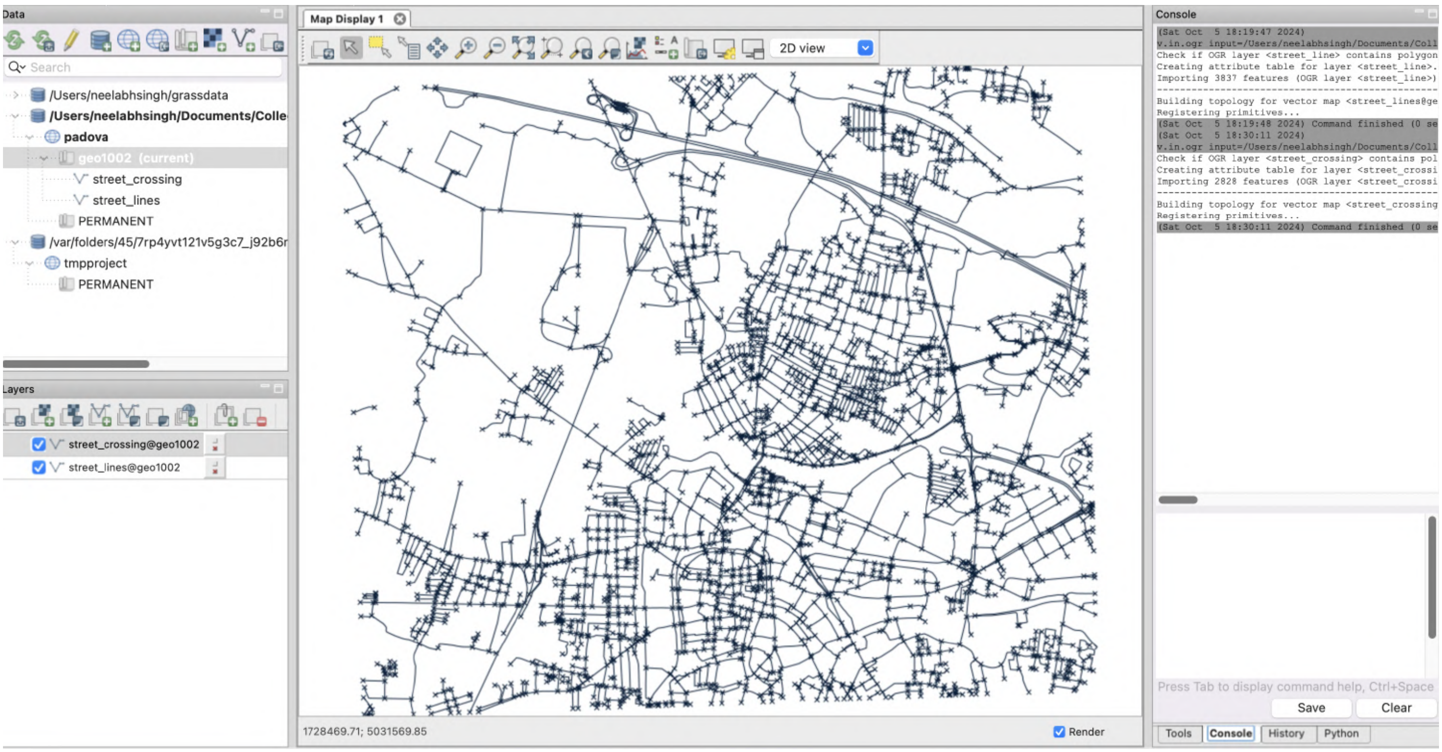


Figure 30: a new vector map added in the project named as street_crossing

Display street crossing layer cat labels

```
d.vect map=street_crossing@geo1002 attribute_column=cat label_color=255:26:23:255
```

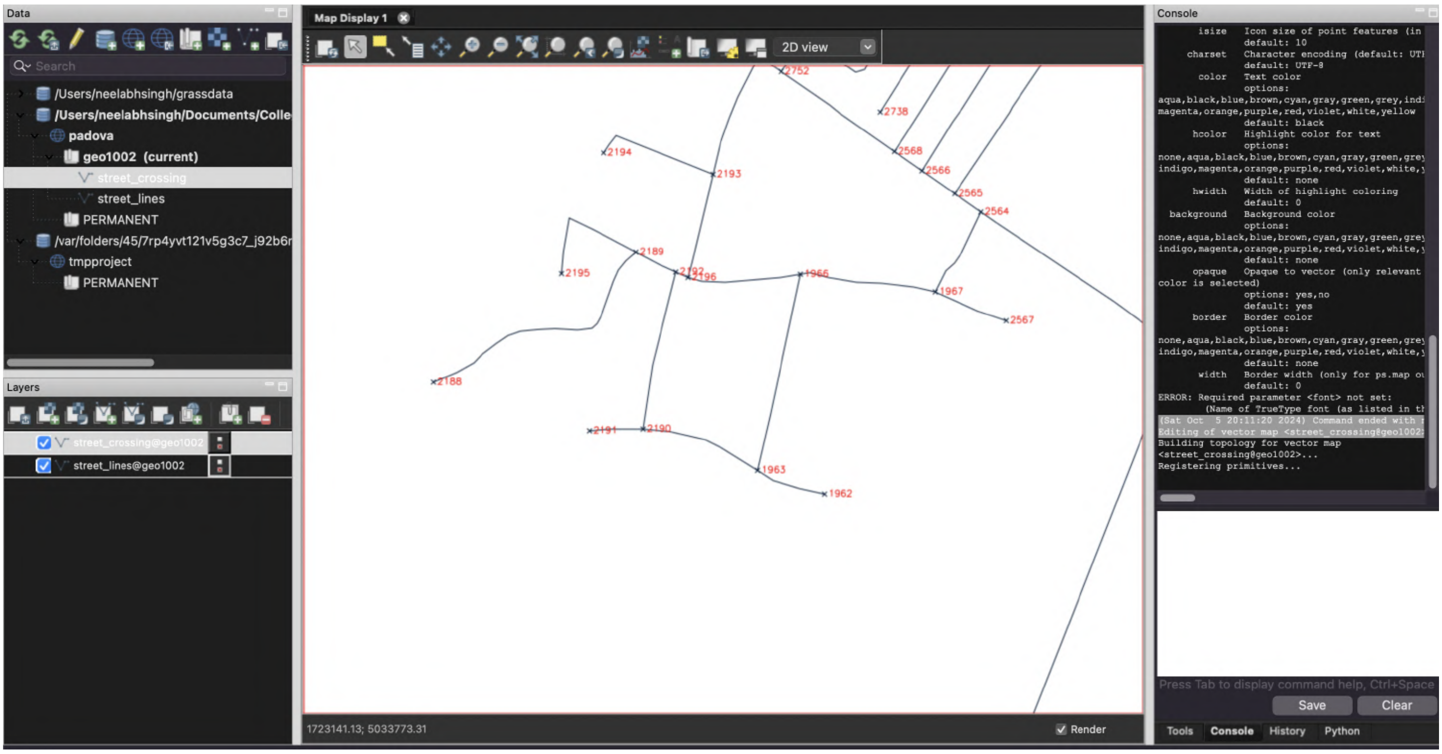


Figure 31: a close up of street_crossing with cat value labelled in red

2.2 Exercise 2.2

Before performing any network operations, you need to build a new vector layer that stores all topological information. In other words, in GRASS GIS you need to connect the nodes (points) to the current edges (the streets).

Use module v.net (read the manual), and using the two previous vector layers as input, create a new layer street_graph assigning the edges to layer 1, the nodes to layer 2. In GRASS, you end up having a vector file which contains different entities: in layer 1 the (poly)lines, in layer 2 the points. Additionally, each layer can be associated to a different table! Layer 1 to the table of vector street_lines, layer 2 to that of street_crossing. For the "threshold" parameter, please use a value of 0.25 (meters).

Create street_graph with polyline in Layer 1 and point in Layer 2

```
v.net input=street_lines@geo1002 points=street_crossing@geo1002 output=street_graph operation=connect
threshold=0.25
```

Visualise the existing layers in street_graph

```
v.category street_graph@geo1002 op=report
```

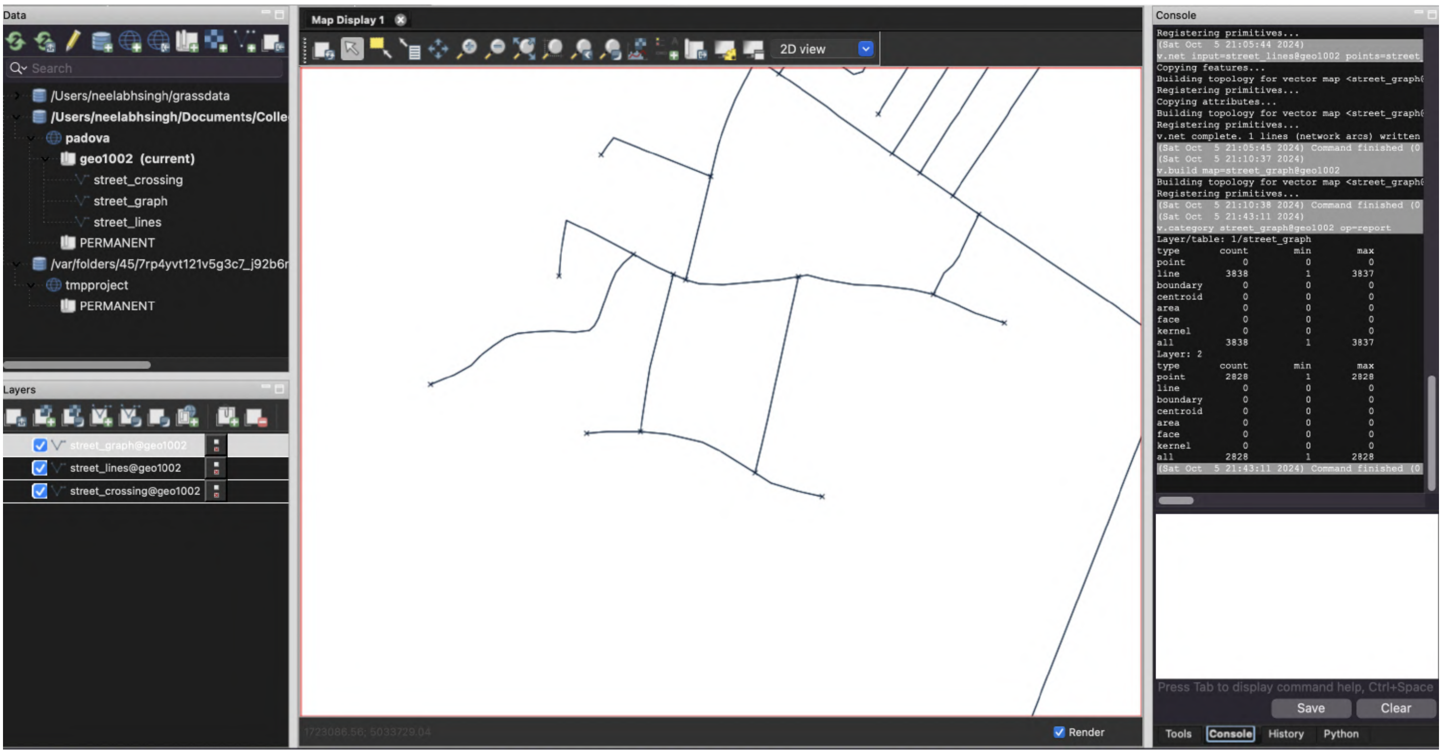


Figure 32: street_graph network map with polyline in layer 1 and points in Layer 2

Connect point layer street_crossing table to street_graph Layer2

```
v.db.connect map=street_graph table=street_crossing@geo1002 layer=2
```

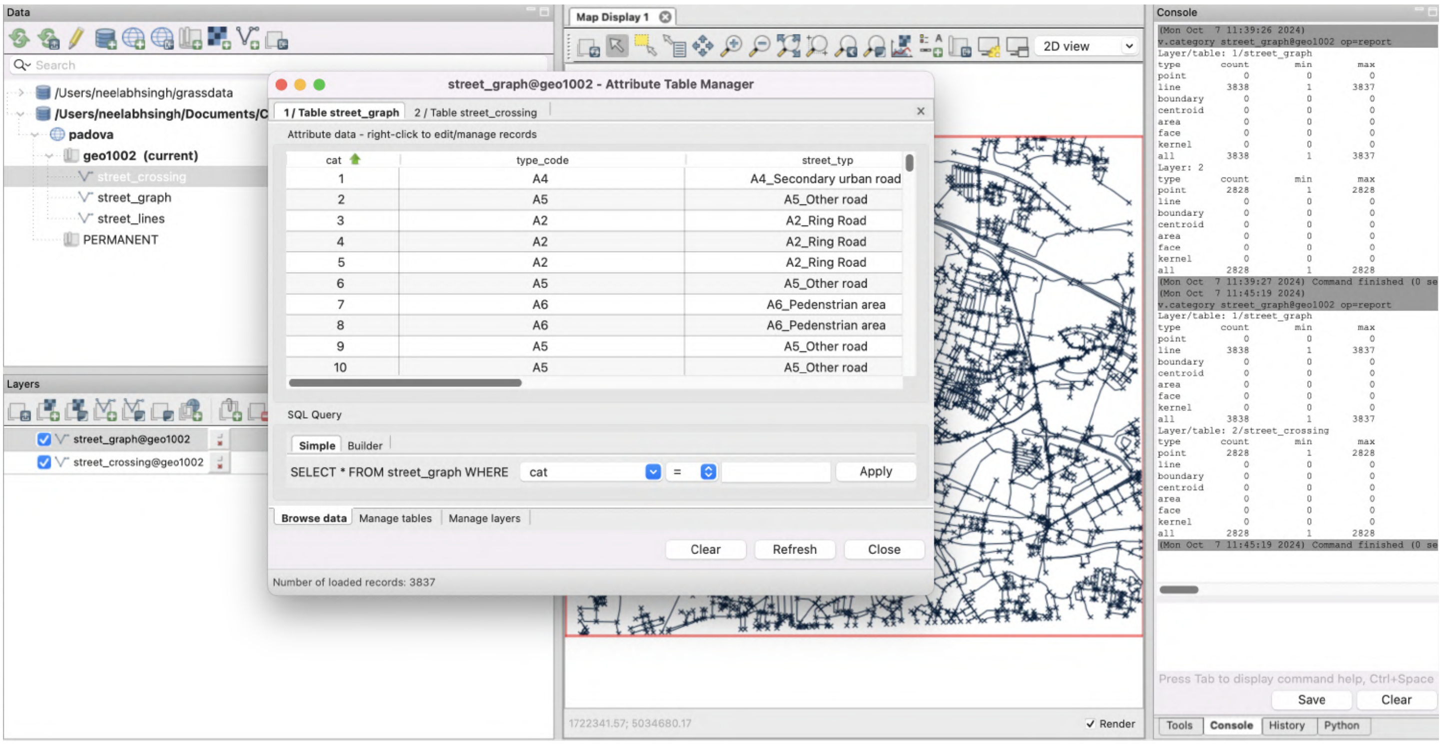


Figure 33: street_graph layer information

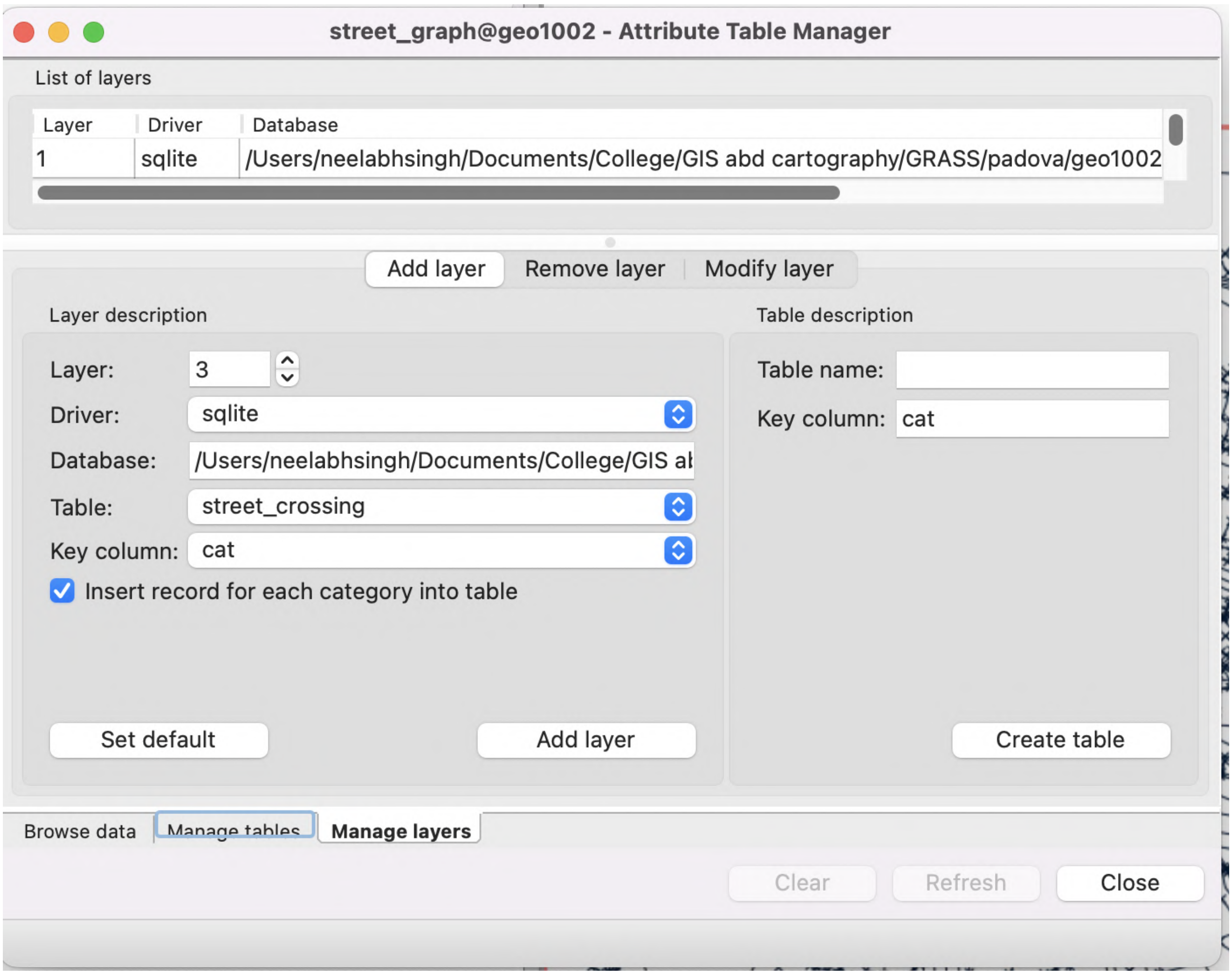


Figure 34: street_graph table information

2.3 Exercise 2.3

Using street_graph as input, create and store a vector layer named street_path where you compute the shortest path between nodes 396 and 369, i.e. from the "beginning" of the motorway to a place in the city centre. When you are done, properly visualise the street_graph and street_path layers.

Calculate a shortest path between node 369 and 396 using time as the cost

```
v.net.path input=street_graph@geo1002 output=street_path file="/points/.tmp/filepath" arc_column=tm_to
arc_backward_column=tm_from node_column=time
```

Visualize node 396 and 369

```
d.vect map=street_graph@geo1002 layer=2 display=shape,cat color=red size=20 icon=basic/cross2
where="cat IN (396,369)"
```

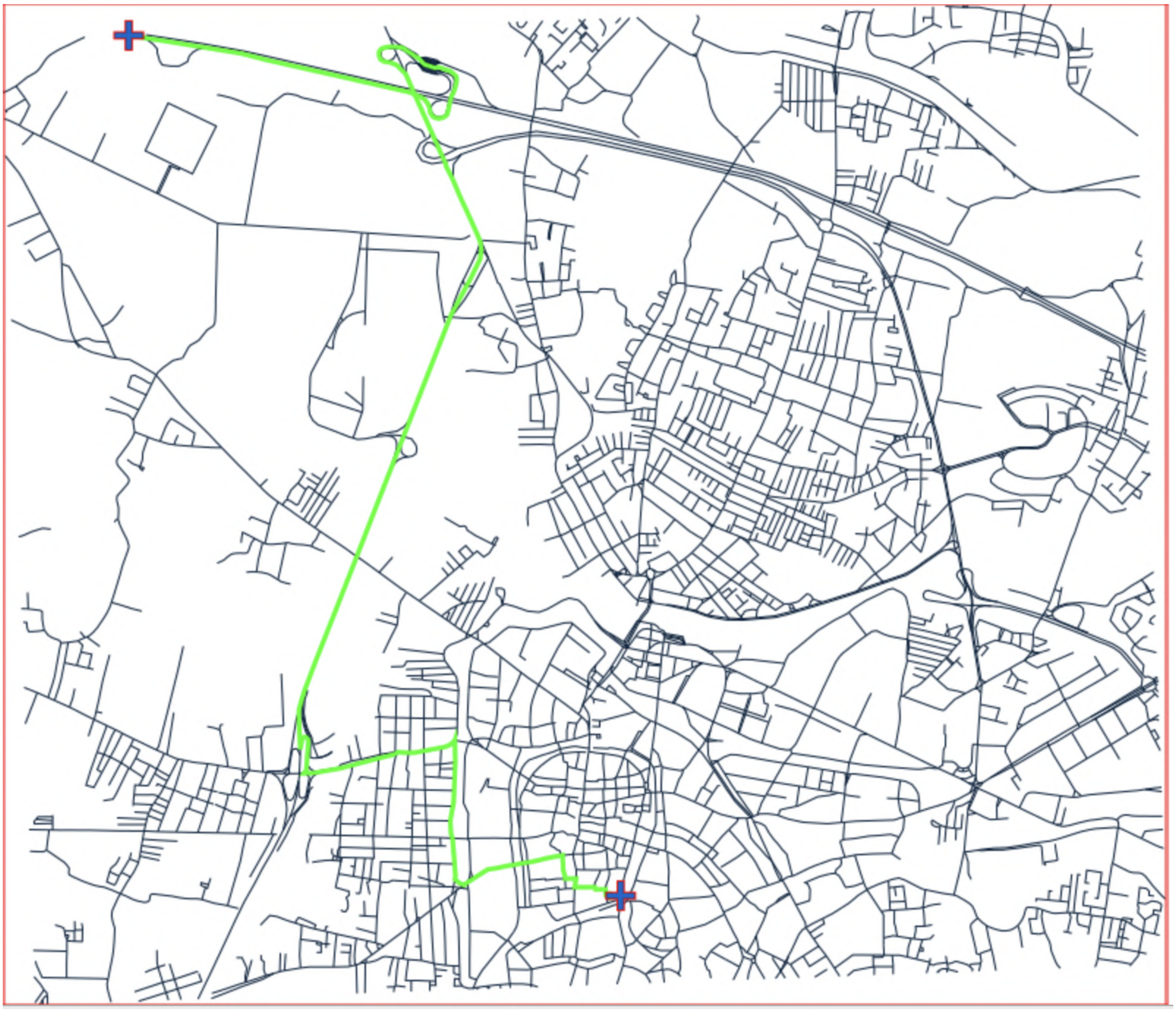


Figure 35: shortest path (green colour) between node 396 and 369 created from street_graph

2.4 Exercise 2.4

Using the traversal time as cost parameter, perform now a network allocation operation, in that you create subnetworks referring to the respective (sub)net centres. Using street_graph as input, create and store vector layer street_alloc where you split the whole network into 5 (sub)networks according to nodes with categories 1178, 2181, 935, 650, and 1619. Learn how to visualise the 5 (sub)networks in different colours and do not forget to show the result also in the report.

Create a network allocation model on 5 nodes, using traversal time as cost parameter

```
v.net.alloc input=street_graph@geo1002 output=street_alloc center_cats="1178, 2181, 935, 650, 1619"
arc_column=tm_from arc_backward_column=tm_to node_column=time
```

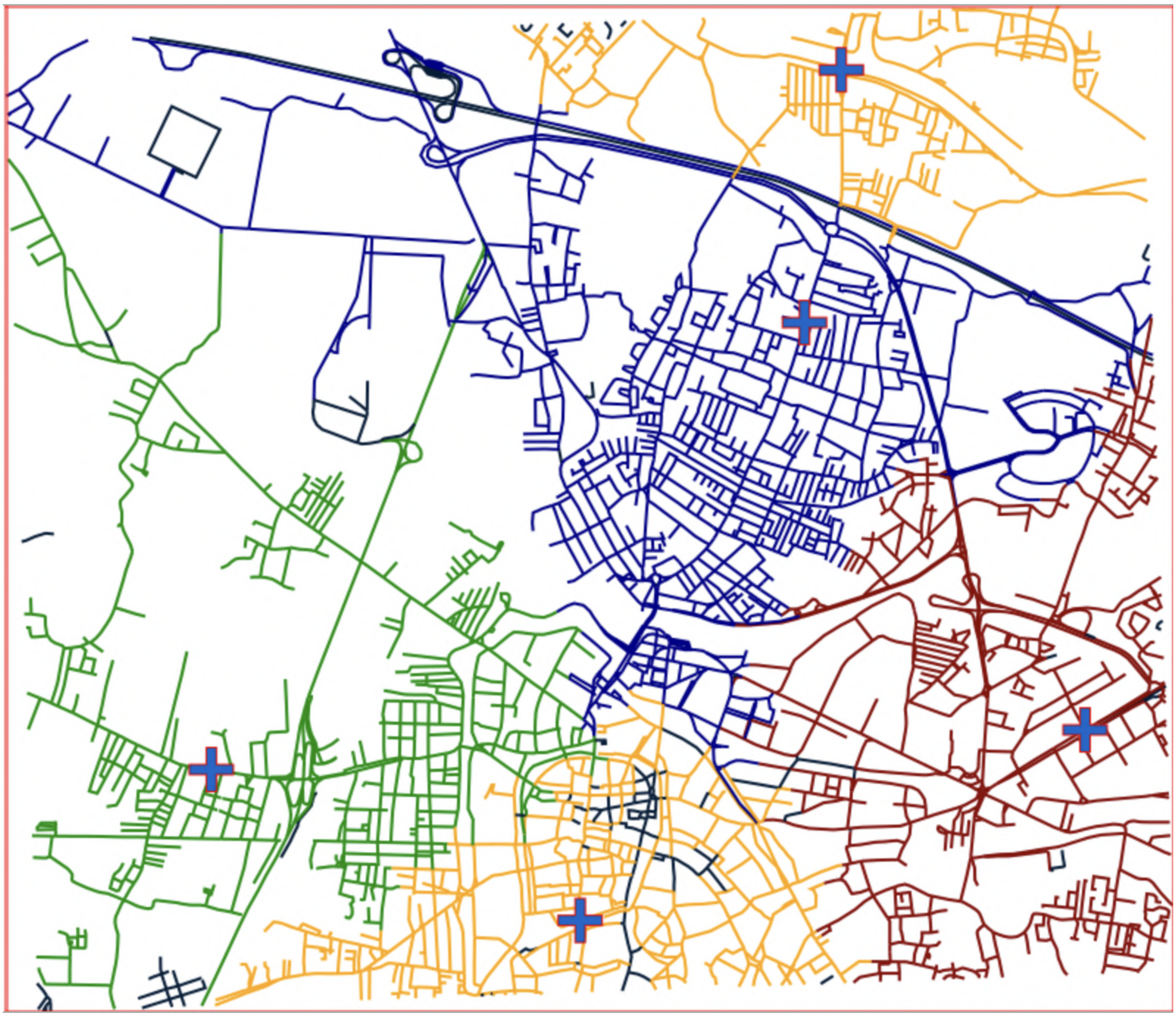


Figure 36: street_alloc: 5 subnetworks from street_graph in different colours, created with node 1178, 2181, 935, 650 and 1619

Look properly at the results, have all edges been used and allocated to the 5 subnetworks? Explain your findings (max 100 words).

We can see above, some edges have not been allocated to any subnetwork. This is very prominent where edges or streets are around the boundary of the map, as it is disconnected from the main network. The allocation process can only assign any street to a sub-network based on its connectivity and proximity to the sub-centers. To have a more refined result of sub-networks, it is important to introduce more nodes that connect all the edges to the network, so that no street or crossing is isolated from the system, topological correctness can also be done.

2.5 Exercise 2.5

Compute now isochrones from a node in the graph. Using street_graph as input, create and store vector layer street_iso. Starting from the train station (node cat=1426), create the (sub)networks corresponding the isochrones from the train station at 1-minute, 2-minute and 5-minute drive. Pay attention to how (i.e. in which units of measure) you express time!

Visualise the (sub)networks and provide screenshots of the results.

Create isochrones from node 1426 i.e the train station, by taking time as cost

```
v.net.iso -u input=street_graph@geo1002 output=street_iso center_cats=1426 costs="60, 120, 300"
arc_column=tm_from arc_backward_column=tm_to node_column=time
```

The isochrone map will show the spread for driving at the maximum speed limit, using time as the cost for values of 60s, 120s, and 300s, as illustrated in the following map. (The -u flag ensures that an attribute table is created.)

Visualize the isochrone map

```
v.db.addcolumn map=street_iso@geo1002 columns="isolbl_numeric INT"
```

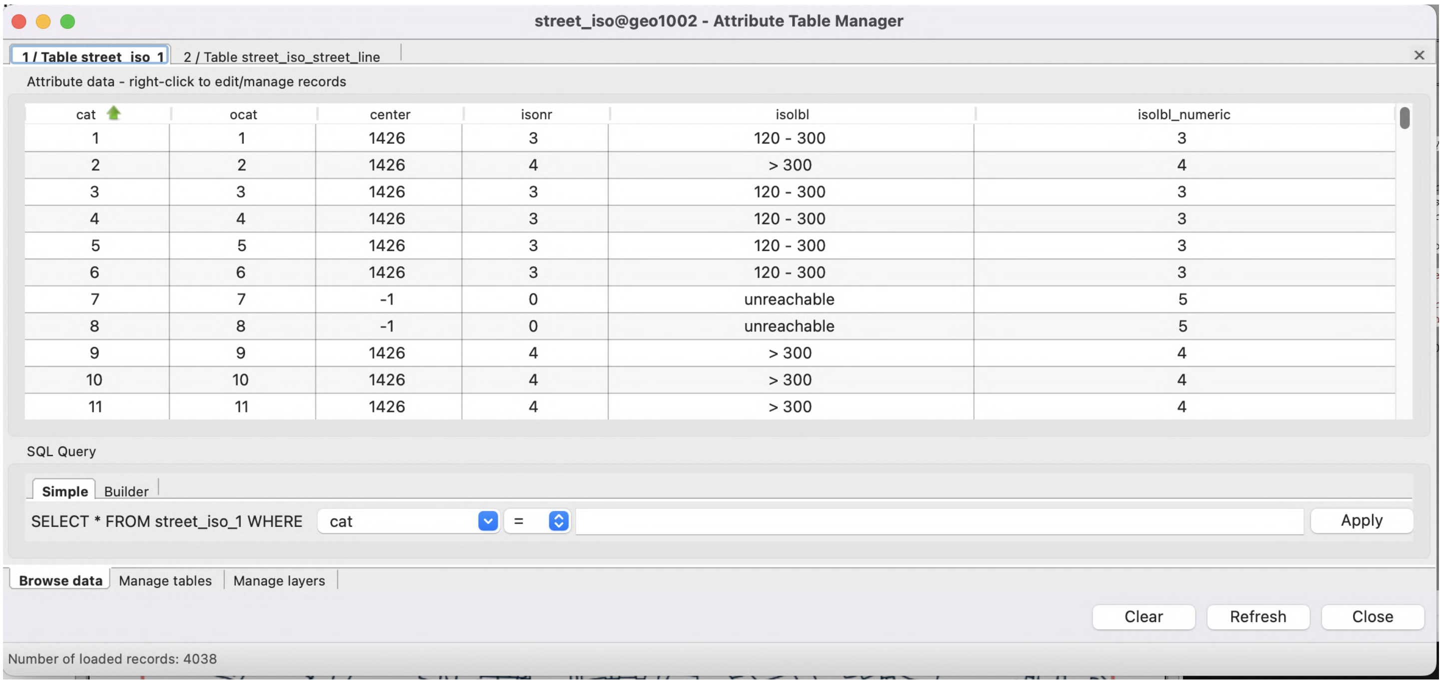
To visualize the isochrone map with random coloring, we need to add a numeric column that stores the values of the five cases and apply color based on those values.

Add numeric value for all 5 cases

```
v.db.update map=street_iso@geo1002 column=isolbl_numeric value=1 where="isolbl = '0 - 60'"
v.db.update map=street_iso@geo1002 column=isolbl_numeric value=2 where="isolbl = '60 - 120'"
v.db.update map=street_iso@geo1002 column=isolbl_numeric value=3 where="isolbl = '120 - 300'"
v.db.update map=street_iso@geo1002 column=isolbl_numeric value=4 where="isolbl = '> 300'"
v.db.update map=street_iso@geo1002 column=isolbl_numeric value=5 where="isolbl = 'unreachable'"
```

Colour according to the numeric value present in isolbl_numeric column

```
v.colors map=street_iso@geo1002 color=bcyr column=isolbl_numeric
```



cat	ocat	center	isonr	isolbl	isolbl_numeric
1	1	1426	3	120 - 300	3
2	2	1426	4	> 300	4
3	3	1426	3	120 - 300	3
4	4	1426	3	120 - 300	3
5	5	1426	3	120 - 300	3
6	6	1426	3	120 - 300	3
7	7	-1	0	unreachable	5
8	8	-1	0	unreachable	5
9	9	1426	4	> 300	4
10	10	1426	4	> 300	4
11	11	1426	4	> 300	4

Figure 37: table of street_iso with isolbl and isolbl_numeric colour

Visualise the node of train station with cat value 1426

```
d.vect map=street_graph@geo1002 layer=2 display=shape,cat color=red size=20 icon=basic/cross2
where="cat = 1426"
```

```
Value 1: Blue, 0 - 60
Value 2: Cyan, 60 - 120
Value 3: Yellow, 120 - 300
Value 4: Red, > 300
Value 5: Brown, unreachable
```

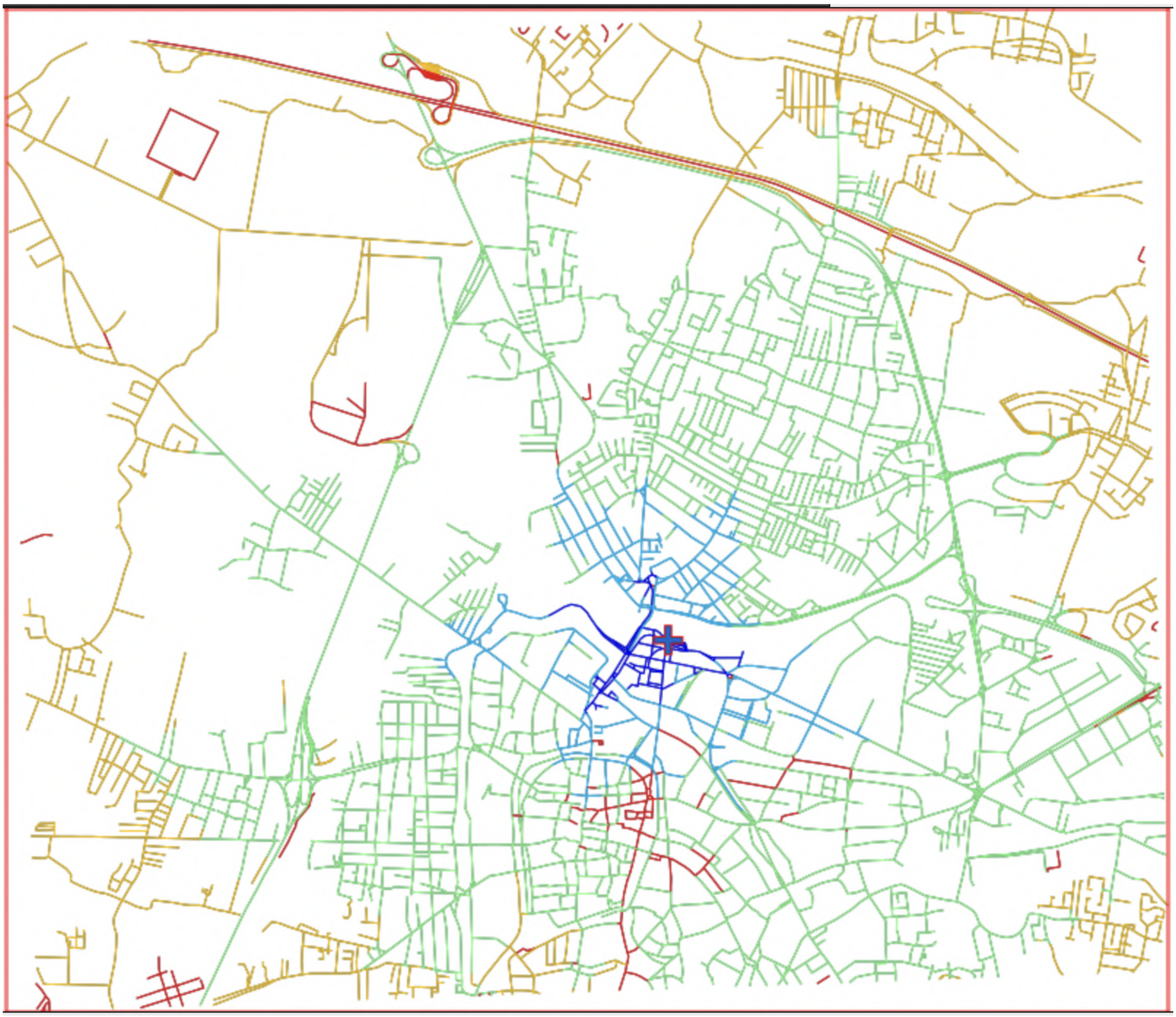


Figure 38: street_iso from node 1426 (train station) with time as cost parameter using values of 60s, 120s and 300s values, created from street_graph network

2.6 Exercise 2.6

Compute now the isochrones, however for pedestrians. Using street_graph as input, create and store vector layer street_iso_p. Here the new criteria:
a) The max walking speed of a person to 4 km/h
b) A person can walk on any street except those of class A1 and A2 (Motorways and Ring roads)
c) The traversal time is the same in both directions of an edge.
d) The traversal time for crossing remains the same as before.
Compute now the isochrones for walking distance up to 5 and 10 minutes from the main train station (node cat=1426). Visualise the (sub)networks and provide screenshots of the results.

Add a column time_walk with type double for values of walking time in seconds

```
v.db.addcolumn map=street_graph@geo1002 columns="tm_walk double precision"
```

To calculate the walking time in seconds and add it to the time_walk column, we exclude two roads that are restricted for pedestrian access by assigning a cost value of -1, and also assign -1 to edges with a value of 0 to achieve a more refined result. (Nodes and arcs can be closed using cost = -1.) To calculate time_walk in metres per second with an average speed of 4km/hr which is around 1.11m/s, simply divide the length by 1.11.

Calculate and add the walking time to the time_walk column

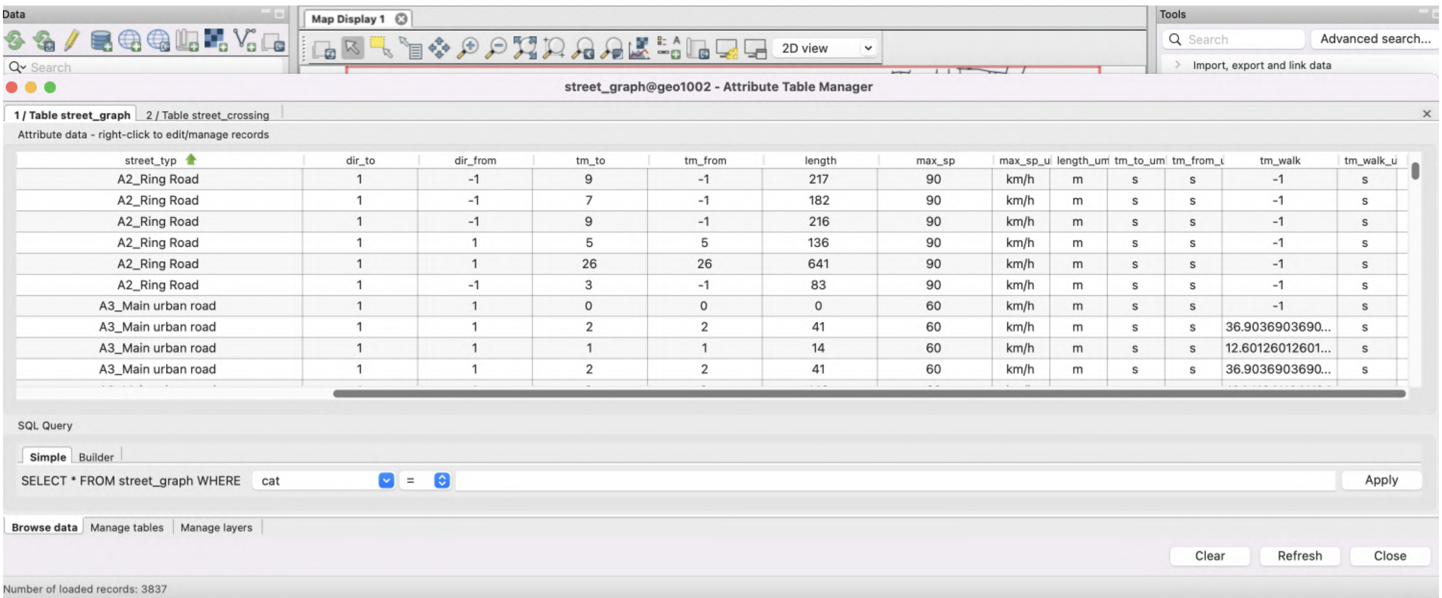
```
v.db.update map=street_graph@geo1002 column=tm_walk value="CASE WHEN street_typ IN ('A1_Motorway', 'A2_Ring R
```

Add the column for the units of time_walk

```
v.db.addcolumn map=street_graph@geo1002 columns="tm_walk_units varchar(10)"
```

Update the units column with value s

```
v.db.update map=street_graph@geo1002 column=tm_walk_units value='s'
```



street_typ	dir_to	dir_from	tm_to	tm_from	length	max_sp	max_sp_u	length_um	tm_to_um	tm_from_u	tm_walk	tm_walk_u
A2_Ring Road	1	-1	9	-1	217	90	km/h	m	s	s	-1	s
A2_Ring Road	1	-1	7	-1	182	90	km/h	m	s	s	-1	s
A2_Ring Road	1	-1	9	-1	216	90	km/h	m	s	s	-1	s
A2_Ring Road	1	1	5	5	136	90	km/h	m	s	s	-1	s
A2_Ring Road	1	1	26	26	641	90	km/h	m	s	s	-1	s
A2_Ring Road	1	-1	3	-1	83	90	km/h	m	s	s	-1	s
A3_Main urban road	1	1	0	0	0	60	km/h	m	s	s	-1	s
A3_Main urban road	1	1	2	2	41	60	km/h	m	s	s	36.9036903690...	s
A3_Main urban road	1	1	1	1	14	60	km/h	m	s	s	12.60126012601...	s
A3_Main urban road	1	1	2	2	41	60	km/h	m	s	s	36.9036903690...	s

Figure 39: the updated table of street_graph with a new time_walk field

The street_graph table (figure 39) includes a new time_walk field, which is calculated based on walking time along the edges at an average speed of 1.11 m/s, with -1 values assigned to A1 and A2 roads.

Create isochrones from node 1426 i.e the train station, by taking time as cost

```
v.net.iso -u input=street_graph@geo1002 output=street_iso_p center_cats=1426 costs="300, 600"
arc_column=tm_walk arc_backward_column=tm_walk node_column=time
```

The isochrone map shows the spread for walking at a maximum speed of 4 km/h (1.11 m/s), using time as the cost parameter for 300 and 600 seconds, as depicted in the following map.

Visualise the node of train station with cat value 1426

```
d.vect map=street_graph@geo1002 layer=2 display=shape,cat color=red size=20 icon=basic/cross2
where="cat = 1426"
```

```
Value 1: Grey, 0 - 300
Value 2: Red, 300-600
Value 3: Brown, <600
Value 4: Black, unreachable
```

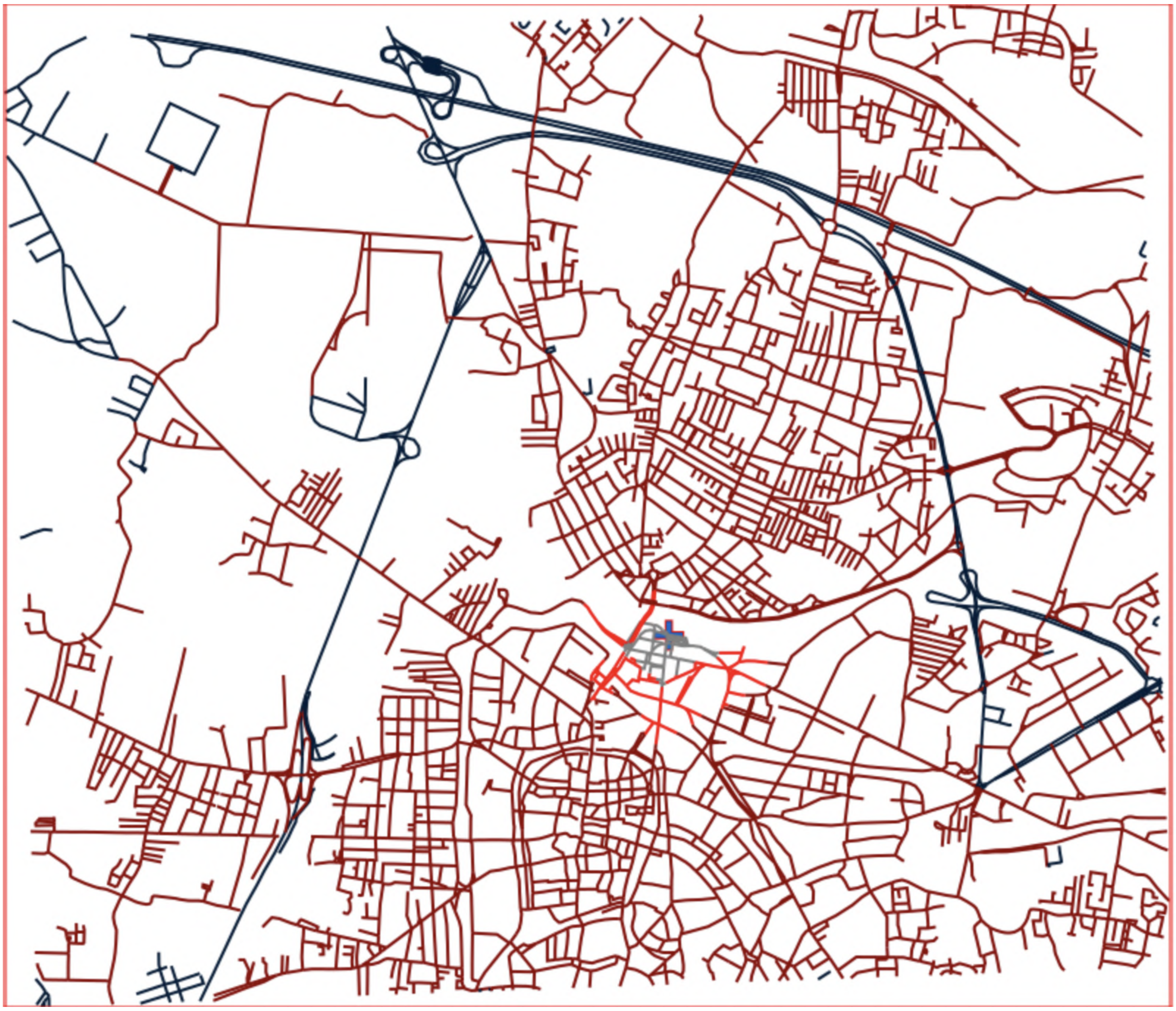


Figure 40: street_iso_p map, created from updated street_graph

How do these subnetworks differ from the ones of the previous Exercise 2.5? (max 100 words)

One of the main differences between isochrones reflecting subnetworks for walking and driving is the coverage, as the driving speed limit is higher in case of driving, and the cost, which is the traversal time, is significantly less. Hence, the subnetworks formed for driving are more widely spread than walking sub-networks, which has an average speed of 4 km/hr only. Additionally, as 2 classes of roads (A1 and A2) are restricted for pedestrians, it results in a more fragmented subnetwork for walking, whereas in the case of the subnetwork of driving, it is a more connected, accessible, and wider subnetwork.

3 Part 3: Data Integration

3.1 Exercise 3.1

Launch GRASS GIS and create a new location "corine". Create a new mapset called "geo1002". Import the shapefiles listed above into vector layers tile39 and tile40, respectively. Provide a simple screenshot of the imported datasets. For your convenience, set the GRASS region to comprise the extents of both datasets.

Open GRASS GIS
For Task 3, we began by launching GRASS GIS and creating a new database. While creating a database wasn't strictly necessary, it helped manage the workload more efficiently.

Create the location
Next, we created a new location named "corine" using the "Create new location" option. We set the EPSG code to 3035 for this location to ensure compatibility with our data in future steps.

Import the shapefiles and set the region

```
v.import input=/data/corine_data/100KME39N32.shp output=tile39
v.import input=/data/corine_data/100KME40N32.shp output=tile40
```

To import the required shapefiles, we used the command console and executed the v.import function. By default, v.import will reproject the data to match the location's Coordinate Reference System (CRS). However, since the shapefiles are already in EPSG:3035 (matching our location), GRASS GIS imported the data without needing reprojection.
Now that we have already set the CRS of our location, our data reproject on the fly to EPSG: 3035. Although, the data were already on EPSG: 3035. To check if the tiles are in the same projection as the locations CRS we used the following:

```
v.info map=tile39
v.info map=tile40
```

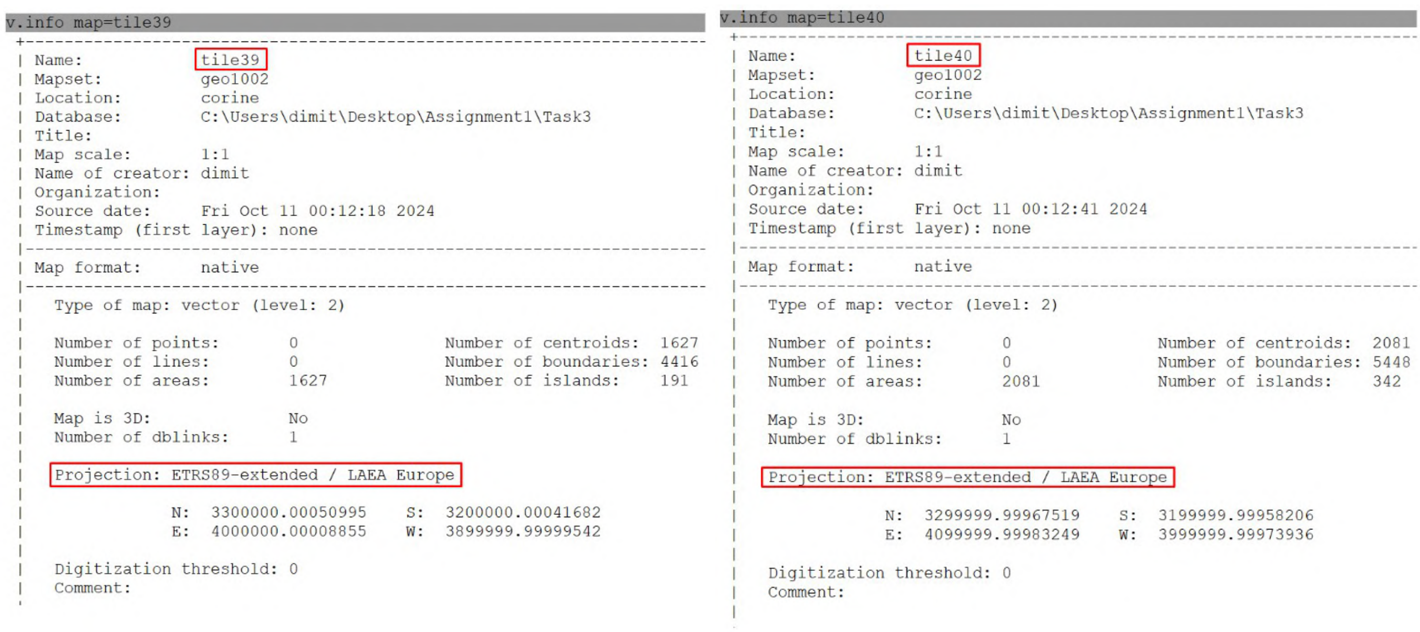


Figure 41: CRS validation for tiles 39,40

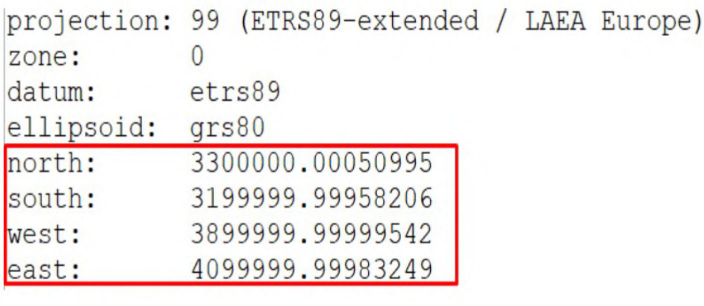
We can validate that TRS89 / LAEA Europe (commonly referred to as ETRS89-extended or ETRS89 Lambert Azimuthal Equal Area) corresponds to EPSG:3035. This projection is widely used for pan-European data, particularly in environmental and land cover datasets like CORINE, as it minimizes distortion across the European continent. To make sure that any spatial operation we perform will be done on the extent of the two tiles, we use the g.region function.

Set the region to tile39 and tile40

```
g.region vector=tile39,tile40
```

Confirm the region settings after running the command

```
g.region -p
```



```
projection: 99 (ETRS89-extended / LAEA Europe)
zone:       0
datum:      etrs89
ellipsoid:  grs80
north:      3300000.00050995
south:      3199999.99958206
west:       3899999.99999542
east:       4099999.99983249
```

Figure 42: region validation for tiles 39,40

We can validate from figure 3.1, 3.1 that the extent of the region matches the extent of the 2 tiles.

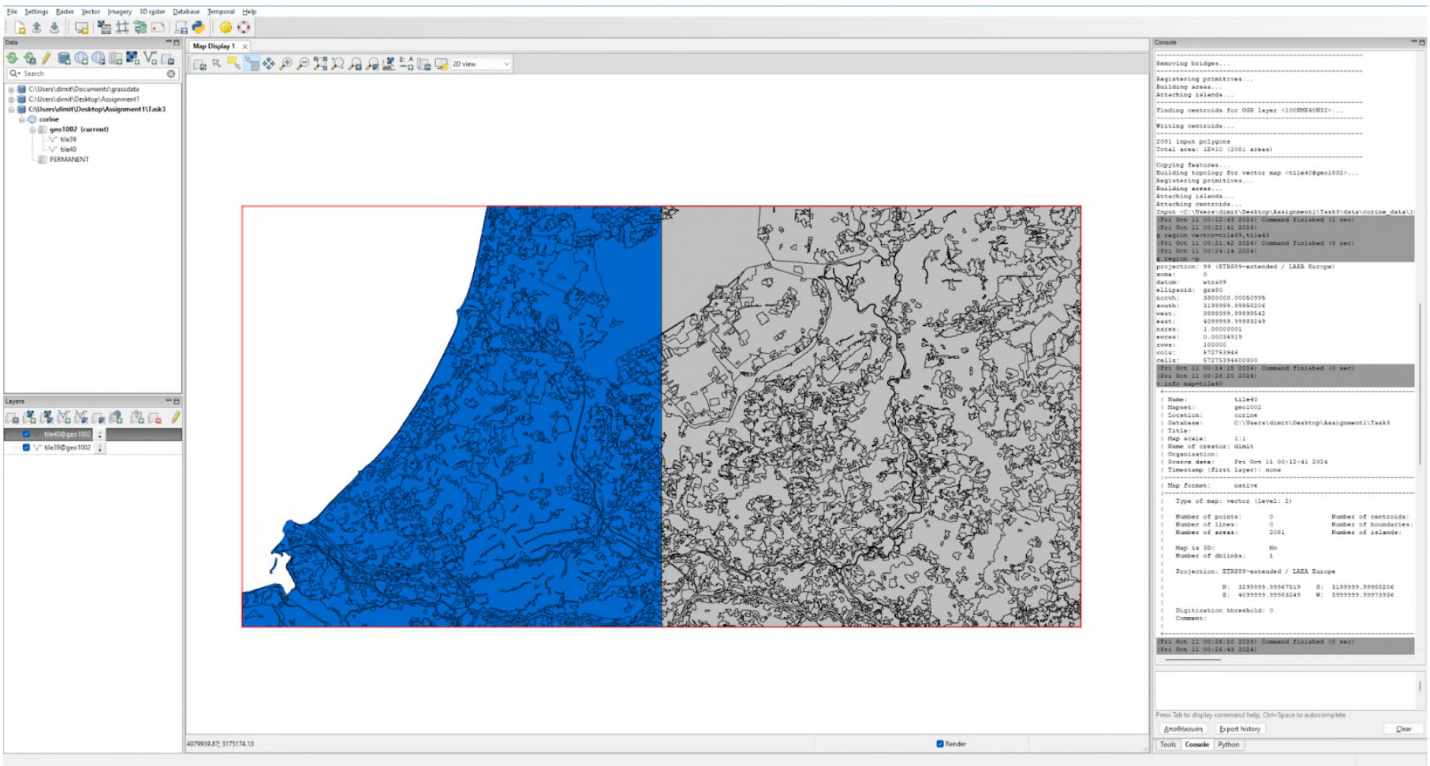


Figure 43: tile 39 and 40 are imported and the region is set to the extent of both

3.2 Exercise 3.2

Define a proper set of operations (mostly overlay ones!) with the goal of merging both tiles into one. The result must be a vector layer named tile39_40 with the polygons and the CODE_00 attribute from both tiles. Adjacent polygons having the same CODE_00 but originally split over the two datasets must be joined and dissolved in layer tile39_40. Provide evidence in the report – and a short explanation – of the steps and operations you carry out, as there are different ways to carry out this exercise. Whenever necessary, add a screenshot to illustrate the (intermediate) results.
To ensure a seamless integration of tile39 and tile40, this process involves an initial union without snapping to detect potential boundary misalignments. After identifying any misalignments, we proceed with a snapped union to align boundaries, followed by cleaning and dissolving steps to finalize the merged layer.

3.2.1 Step 1: Perform an initial union without snapping (misalignment check)

We run v.overlay without snapping to merge tile39 and tile40 initially, without attempting to align boundaries.

Create a union of the 2 tiles without snapping

```
v.overlay ainput=tile39 binput=tile40 operator=or output=tile39_40_union_no_snap
```

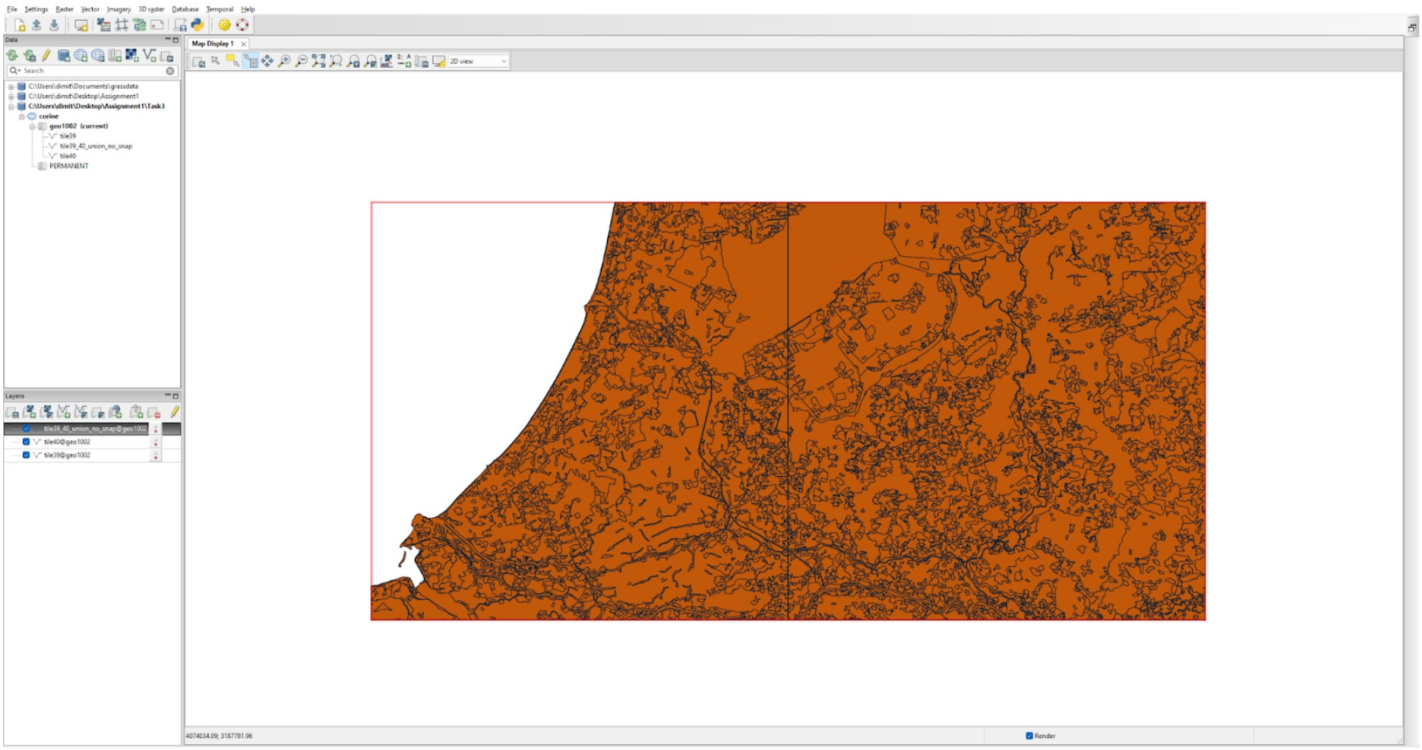


Figure 44: creating a Union for Tiles 39 and 40 without snapping parameters

This step helps us identify potential misalignments between the tiles along their shared boundary. By not snapping, we preserve any small gaps or overlaps, allowing us to analyze if the boundaries align as expected or if discrepancies are present.

Add a column to populate the corine code from both tiles

```
v.db.addcolumn map=tile39_40_union_no_snap column="CODE_00"
```

We add a CODE_00 column to hold the merged land cover code from a_CODE_00 (from tile39) and b_CODE_00 (from tile40):

tile39_40_union_no_snap@geo1002 - Attribute Table Manager

Attribute data - right-click to edit/manage records

cat	a_cat	a_CODE_00	a_AREA	b_cat	b_CODE_00	b_AREA	CODE_00
1	109	112	323923.449937				
2	137	121	480887.971523				
3	439	423	1446916.858676				
4	515	512	553579.647359				
5	435	243	934213.18061				
6	626	112	1169721.149877				
7	898	512	427791.639434				
8	912	512	259104.872169				
9	950	312	1777480.503627				
10	1011	411	282521.866613				
11	1021	231	326862.10173				

Figure 45: field CODE_00 creation

populate the column with the following conditions using the v.db.update function

Use a_CODE_00 where b_CODE_00 is NULL

```
v.db.update map=tile39_40_union_no_snap  column=CODE_00 query_column=a_CODE_00
where="b_CODE_00 IS NULL"
```

Use b_CODE_00 where a_CODE_00 is NULL

```
v.db.update map=tile39_40_union_no_snap  column=CODE_00 query_column=b_CODE_00
where="a_CODE_00 IS NULL"
```

Use either value where both are equal, indicating alignment

```
v.db.update map=tile39_40_union_no_snap  column=CODE_00 query_column=a_CODE_00
where="a_CODE_00 = b_CODE_00"
```

This step is crucial for confirming alignment. If there are records where a_CODE_00 and b_CODE_00 both have values but differ, it indicates a misalignment. By checking this, we can determine if snapping is necessary to achieve a seamless merge.

tile39_40_union_no_snap@geo1002 - Attribute Table Manager

Attribute data - right-click to edit/manage records

cat	a_cat	a_CODE_00	a_AREA	b_cat	b_CODE_00	b_AREA	CODE_00
1	109	112	323923.449937				112
2	137	121	480887.971523				121
3	439	423	1446916.858676				423
4	515	512	553579.647359				512
5	435	243	934213.18061				243
6	626	112	1169721.149877				112
7	898	512	427791.639434				512
8	912	512	259104.872169				512
9	950	312	1777480.503627				312

Figure 46: Use a_CODE_00 where b_CODE_00 is NULL

tile39_40_union_no_snap@geo1002 - Attribute Table Manager

Attribute data - right-click to edit/manage records

cat	a_cat	a_CODE_00	a_AREA	b_cat	b_CODE_00	b_AREA	CODE_00
1647				81	211	596799.724943	211
1648				101	211	272343.954716	211
1649				92	512	335342.957038	512
1650				216	112	251354.857619	112
1651				311	512	465558.61048	512
1652				313	512	278611.794202	512
1653				514	243	350121.107622	243
1654				521	112	746002.203498	112
1655				501	231	1779412.266795	231
1656				639	311	447851.135279	311
1657				661	311	711915.07397	311

Figure 47: Use b_CODE_00 where a_CODE_00 is NULL

tile39_40_union_no_snap@geo1002 - Attribute Table Manager

Attribute data - right-click to edit/manage records

cat	a_cat	a_COD...	a_AREA	b_cat	b_COD...	b_AREA	CODE_00
3801	868	313	3217451.521747	1013	313	8514254.260807	313

Figure 48: Use a_CODE_00 where a_CODE_00 = b_CODE_00

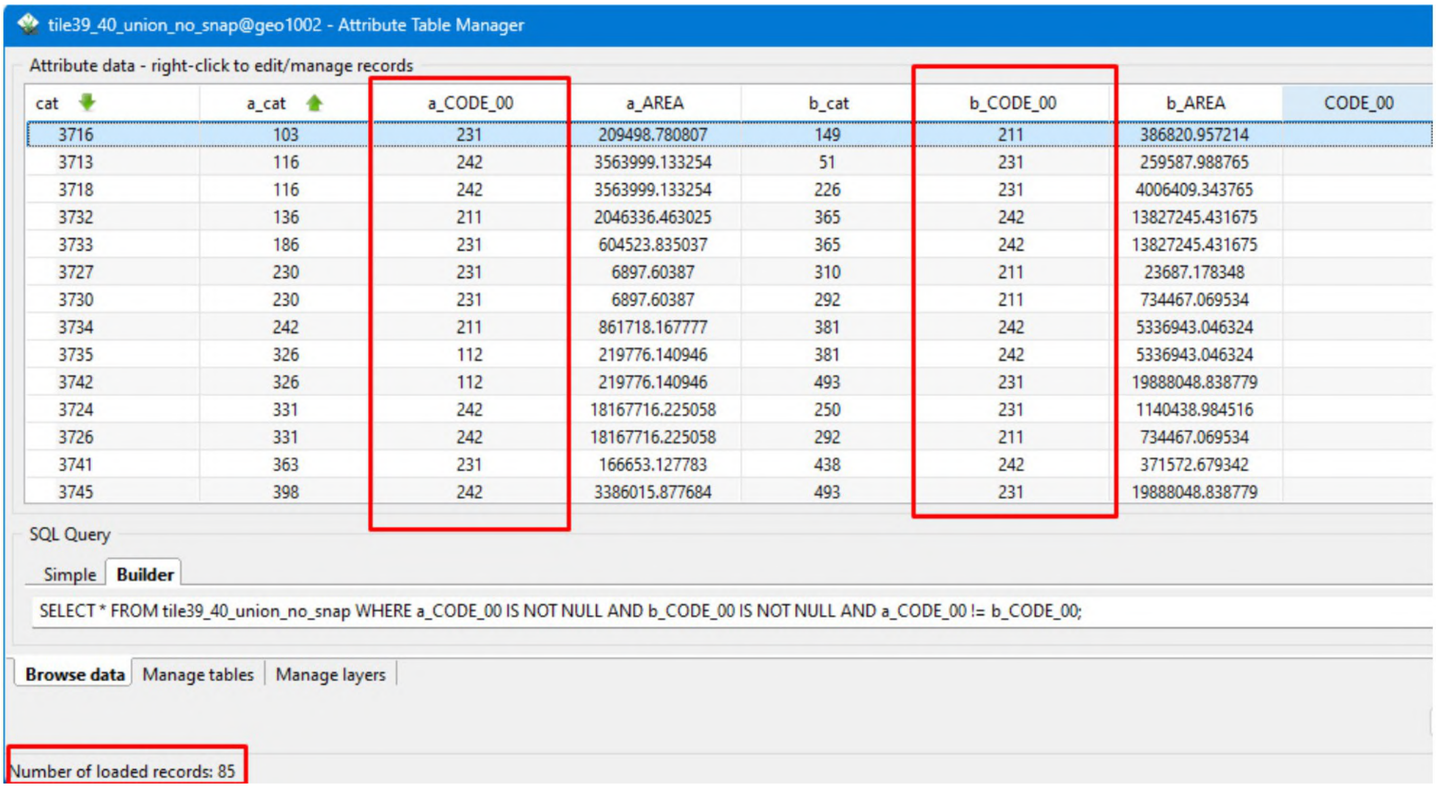
To spot misalignment, the key is to identify records where both a_CODE_00 and b_CODE_00 have values, but those values are not the same. After running the updates to populate CODE_00 in cases where a_CODE_00 or b_CODE_00 is null or where they are equal, any records where a_CODE_00 and b_CODE_00 differ will remain unpopulated in the CODE_00 column. This means that records with no value in CODE_00 indicate areas of misalignment between the tiles.
This means that:

- Both a_CODE_00 and b_CODE_00 have values
- a_CODE_00 and b_CODE_00 have different values
- The CODE_00 column is unpopulated for these records

In order to find the unpopulated records of CODE_00 where the above criteria are valid we wrote the following SQL code in the SQL Query Builder of the attribute table for the dataset of the tile39_40_union_no_snap.

SQL CODE

```sql
SELECT * FROM tile39_40_union_no_snap
WHERE a_CODE_00 IS NOT NULL AND b_CODE_00 IS NOT NULL AND a_CODE_00 != b_CODE_00;
```



cat	a_cat	a_CODE_00	a_AREA	b_cat	b_CODE_00	b_AREA	CODE_00
3716	103	231	209498.780807	149	211	386820.957214	
3713	116	242	3563999.133254	51	231	259587.988765	
3718	116	242	3563999.133254	226	231	4006409.343765	
3732	136	211	2046336.463025	365	242	13827245.431675	
3733	186	231	604523.835037	365	242	13827245.431675	
3727	230	231	6897.60387	310	211	23687.178348	
3730	230	231	6897.60387	292	211	734467.069534	
3734	242	211	861718.167777	381	242	5336943.046324	
3735	326	112	219776.140946	381	242	5336943.046324	
3742	326	112	219776.140946	493	231	19888048.838779	
3724	331	242	18167716.225058	250	231	1140438.984516	
3726	331	242	18167716.225058	292	211	734467.069534	
3741	363	231	166653.127783	438	242	371572.679342	
3745	398	242	3386015.877684	493	231	19888048.838779	

Figure 49: selected records with desired criteria before snapping

We understand from the selected records in Screenshot 3.9 that there are 85 records where a_CODE_00 and b_CODE_00 are populated but have different values. That means that these polygons overlap but are misaligned.

3.2.2 Step 2: Perform union with snapping

We now run v.overlay with snapping to correct the minor boundary misalignments. This aligns vertices within a defined tolerance, creating a unified layer that resolves any discrepancies.
Create a Union of the 2 tiles with snapping parameters

```
v.overlay ainput=tile39 binput=tile40 operator=or output=tile39_40_union snap=0.1
```

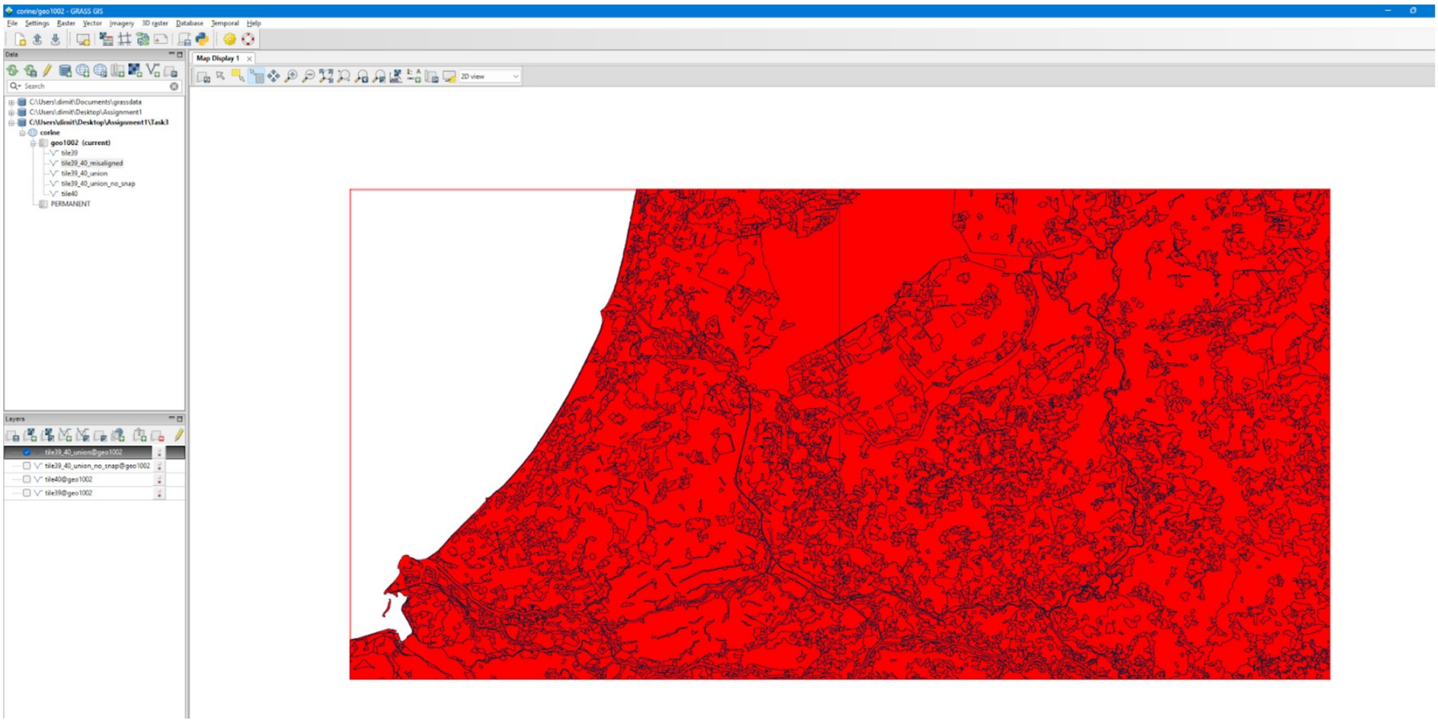
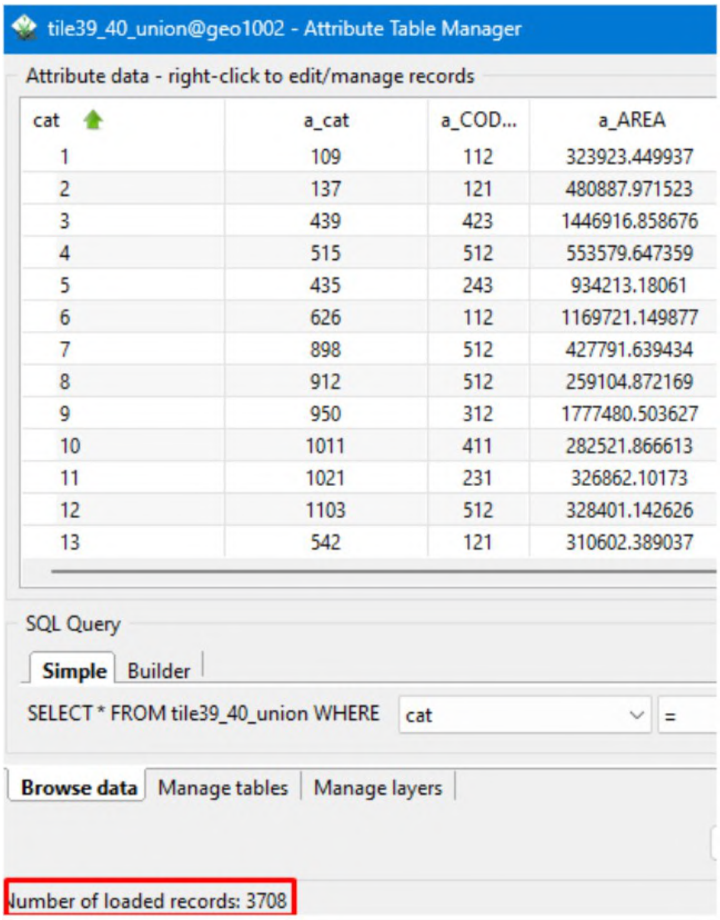


Figure 50: creating a Union for Tiles 39 and 40 with snapping parameters

We can already observe that the records in the attribute table are significantly less.



cat	a_cat	a_COD...	a_AREA
1	109	112	323923.449937
2	137	121	480887.971523
3	439	423	1446916.858676
4	515	512	553579.647359
5	435	243	934213.18061
6	626	112	1169721.149877
7	898	512	427791.639434
8	912	512	259104.872169
9	950	312	1777480.503627
10	1011	411	282521.866613
11	1021	231	326862.10173
12	1103	512	328401.142626
13	542	121	310602.389037

Figure 51: attribute table of the new union dataset

Using snapping at this stage corrects the minor misalignments identified in the previous step, ensuring that the boundaries of tile39 and tile40 match seamlessly. This step is essential for creating a continuous layer without gaps or overlaps. Next we perform all the steps of the misalignment check.

SQL CODE

```sql
SELECT * FROM tile39_40_union
WHERE a_CODE_00 IS NOT NULL AND b_CODE_00 IS NOT NULL AND a_CODE_00 != b_CODE_00;
```

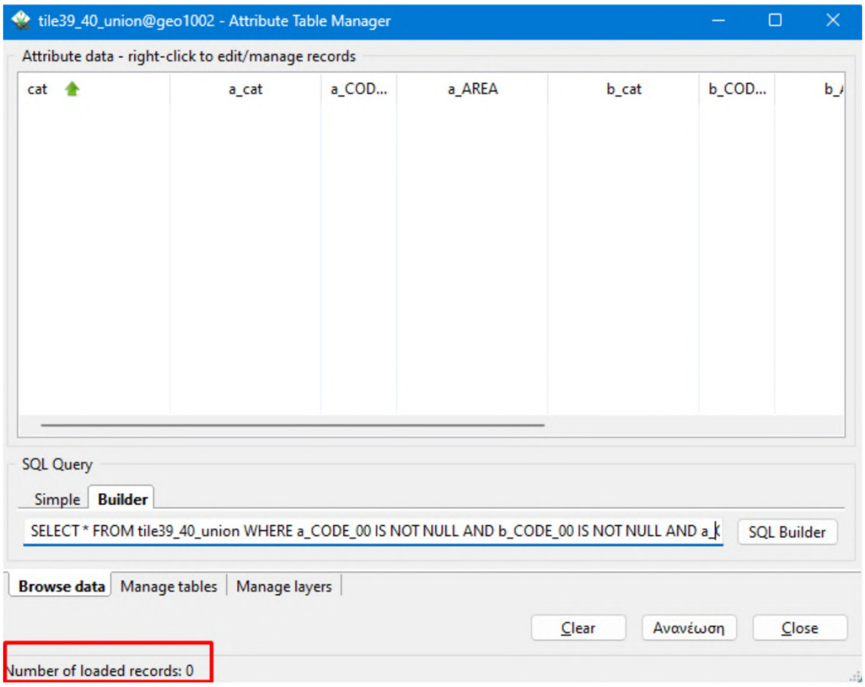


Figure 52: selected records with desired criteria after snapping

We can clearly understand that no record was selected where both a_CODE_00 and b_CODE_00 have values, a_CODE_00 and b_CODE_00 have different values, and the CODE_00 column is unpopulated for these records. That means that with snapping tolerance = 0.1 we have no misaligned geometries in the union.

3.2.3 Step 3: Clean the union layer

Now that our dataset is aligned we used v.clean to eliminate duplicate geometries and ensure a single, clean boundary along the union.

Remove duplicate boundaries

```
v.clean input=tile39_40_union output=tile39_40_clean tool=rmdupl,break
```

Removing duplicates and breaking lines at intersections ensures that the merged layer has a single, clean boundary.

3.2.4 Step 4: Dissolve based on CODE_00

In this final step we simplify the dataset by merging contiguous polygons with the same land cover classification (CODE_00). This removes unnecessary internal boundaries and creates larger, unified regions (v. dissolve was used).
Dissolve polygons with the same CODE_00

```
v.dissolve input=tile39_40_clean output=tile39_40_dissolved column=CODE_00
```

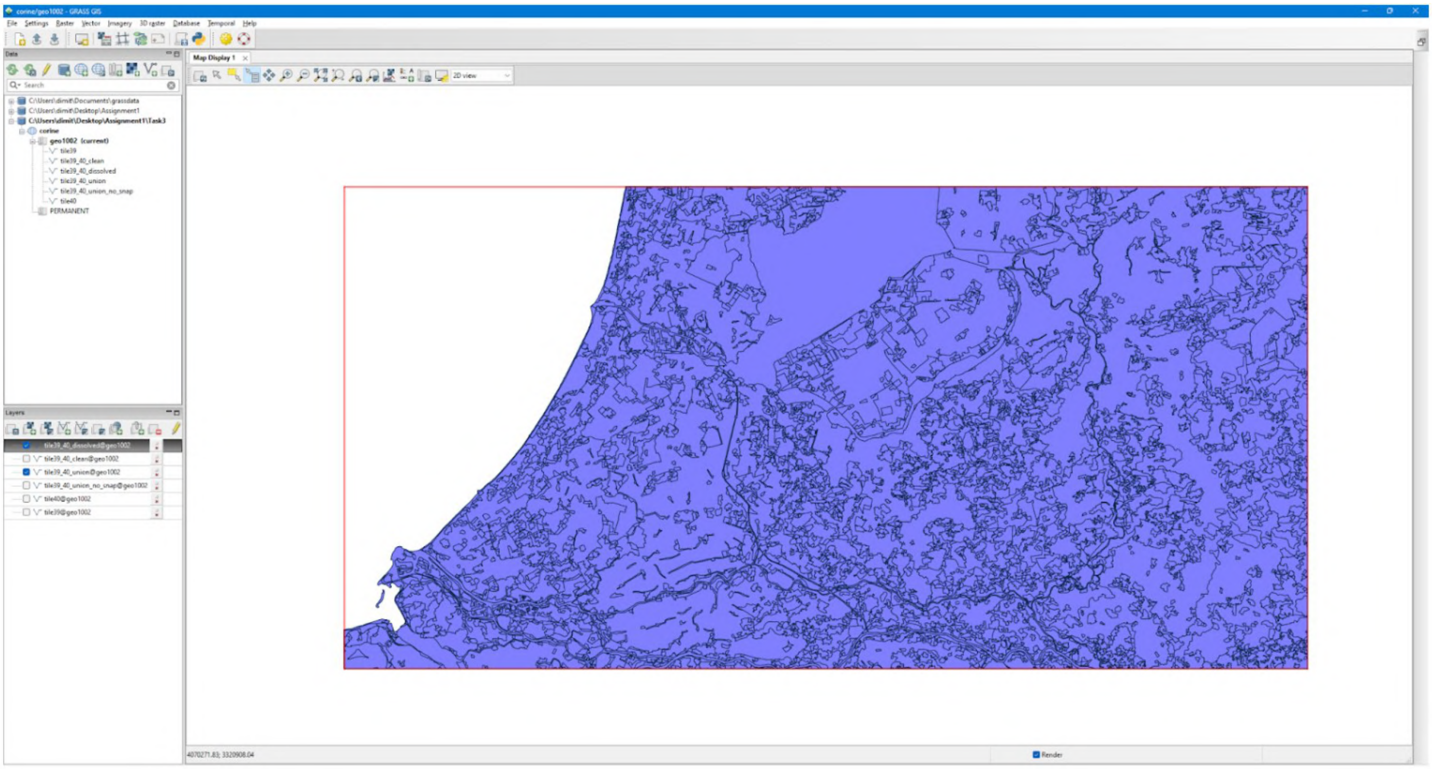


Figure 53: aligned and dissolved union of tile 39 and tile 40

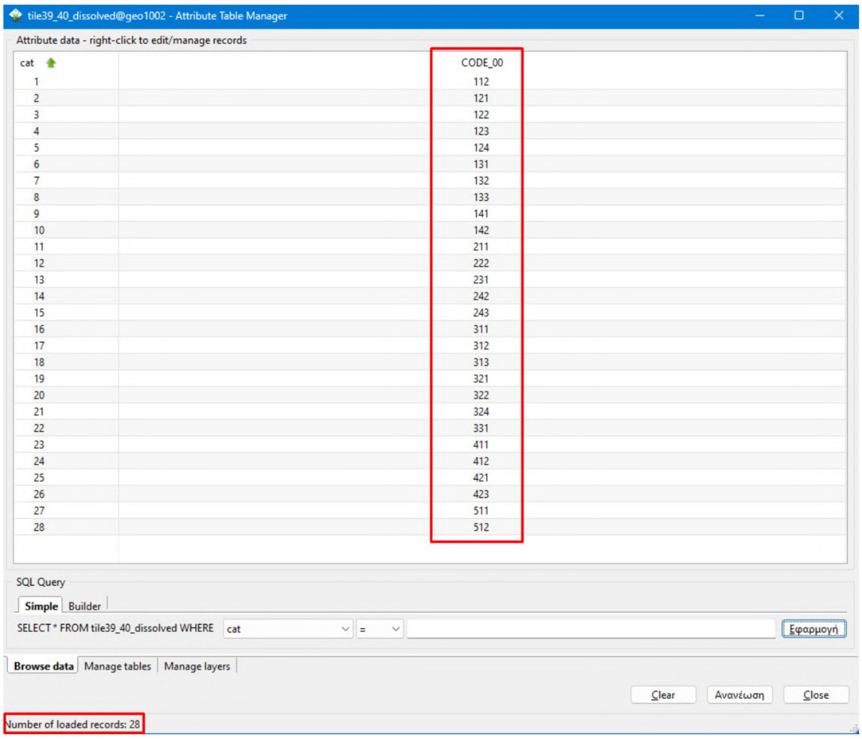


Figure 54: attribute table of the final aligned and dissolved union of tile 39 and tile 40

NOTE: v.dissolve drops the rest of the columns.

3.3 Exercise 3.3

Export the vector tile39_40 as a shapefile. Open it in QGIS to check that the export has been successful.

How many polygons are there in this datasets?
Export the tile39_40_dissolved layer to a shapefile

```
v.out.ogr input=tile39_40 format=ESRI_Shapefile output=data/corine_data/tile39_40.shp
```

Load the file in QGIS

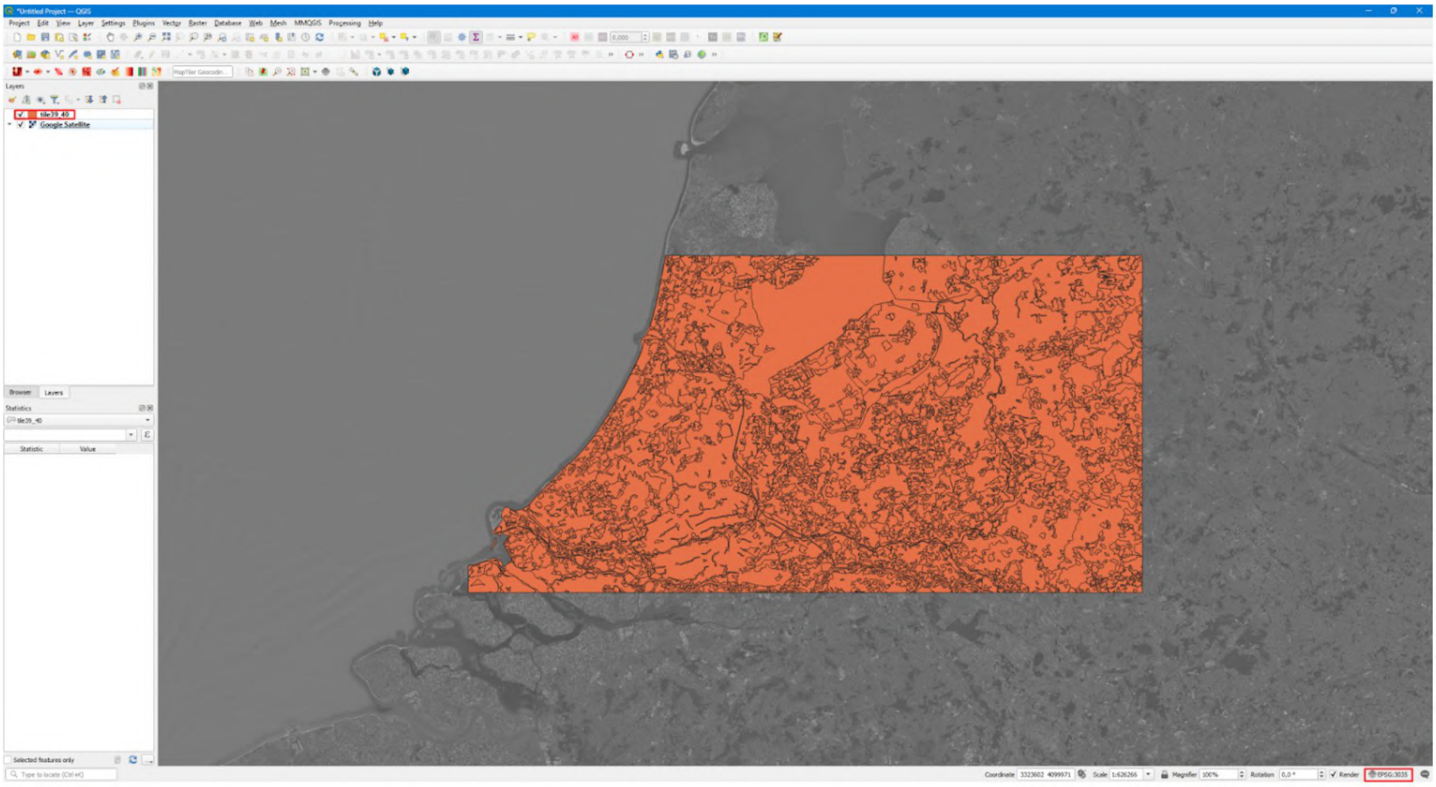


Figure 55: the exported shapefile in QGIS environment

Load the file in QGIS

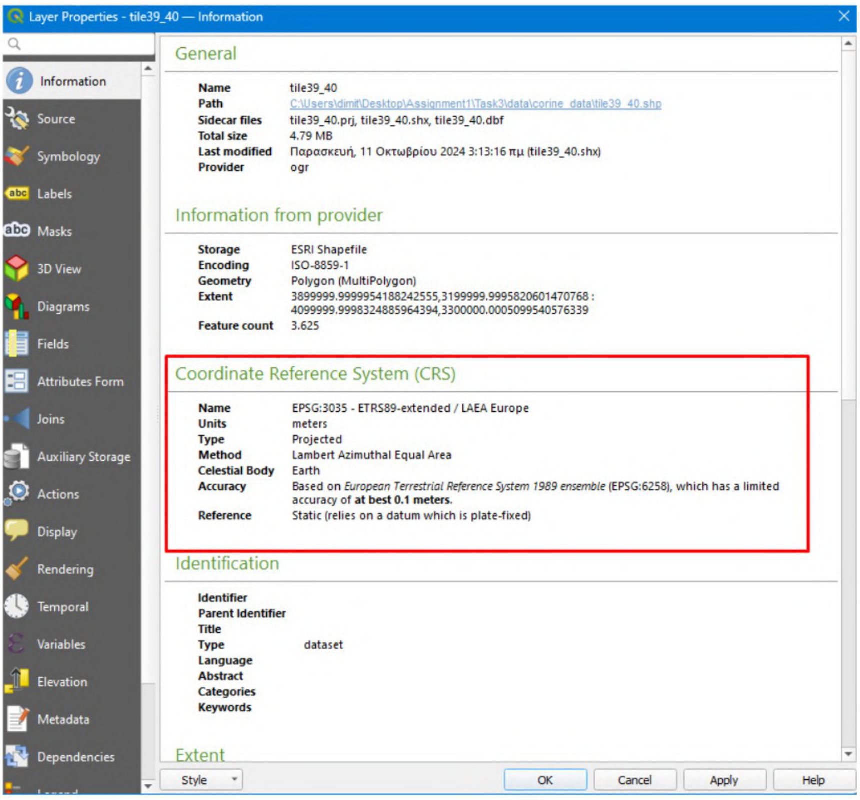
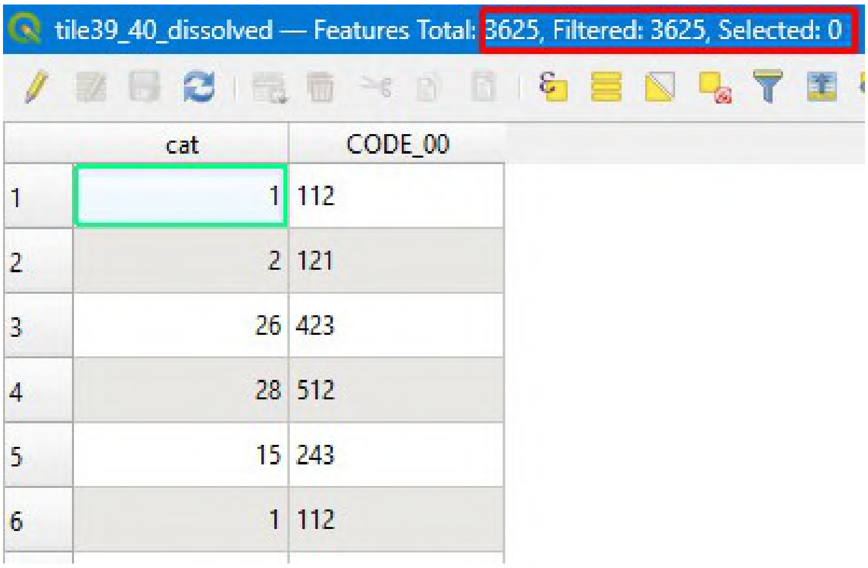


Figure 56: validating shapefiles CRS as QGIS reprojects on the fly

To count how many polygons we have in the dissolved file, we opened the attribute table of the file. We observed that, instead of having 28 polygons as we expected from the dissolved file from GRASS GIS, we now have 3625 polygons. Which means, when exporting the dissolved data, it turned the dissolved multiparts back into singleparts.



	cat	CODE_00
1	1	112
2	2	121
3	26	423
4	28	512
5	15	243
6	1	112

Figure 57: attribute table of the exported dissolved shapefile

When exporting multi-part features from GRASS GIS to formats like Shapefile or GeoPackage, our team noticed that multi-part geometries are often split into single-part features. This behavior appears to stem from how these formats handle geometry data, particularly the Shapefile format, which can separate multi-part features into individual single-part polygons during export to simplify structure.

Even though GeoPackage supports multi-part geometries more robustly, software like QGIS may still interpret these features as single-part. This is illustrated by the "Multipart to Singleparts" tool in QGIS, which explicitly converts multi-part features into single-part ones, highlighting how GIS tools often prioritize single-part handling for ease of processing.

To maintain the original multi-part structure, we found that it's beneficial to dissolve adjacent polygons using spatial criteria before export and verify the output directly in QGIS or another GIS software. In cases where single-part features are sufficient, however, managing the data in a split format can streamline further analysis

3.4 Exercise 3.4

Please write a short comment (max 200 words) about the major challenges that you have faced to solve this task. If you had more time, would you solve Exercise 3.2 differently?

The biggest challenge of this process was to seamlessly integrate the two tiles, tile39 and tile40, and at the same time ensure that only adjacent polygons with the same attribute would be merged. First, the process in fact grouped both adjacent and nonadjacent areas with the same attribute, which is not the requirement of the task-the requirement of the task is to dissolve only connected polygons. This involved more processes to isolate the non-adjacent features, checking their alignment; hence, it gave room for complexity and increased the time to be used.

Others included handling the geometry inconsistencies, such as very small gaps, overlaps, and self-intersecting boundaries that resulted from the merge process. These needed careful attention so as to deliver the integrity of the resultant dataset, especially upon export.

Given more time, I would try alternative approaches to improve the merging process, in greater detail paying more attention to those techniques which are mainly based on criteria of spatial adjacency. In this way, one could avoid a lot of post-processing and would have an easier path toward the results of the task at hand.