Multi-Criteria Analysis

2025-04-26

Per Halvorsen
pmhalvor@uio.no
GEO4460

1 Introduction

Can we use publicly available data to improve cycling conditions routes around Oslo?

The goal of this project was to produce a multi-criteria analysis to find potentially dangerous cycling routes around Oslo. To do so, we used handful of important criteria to consider when planning future cycling routes, including route popularity, traffic, and typical cycling speed.

This multi-criteria analysis was possible thanks to publicly available data sources like the Strava API, Kartverket's N50 dataset, and Statens Vegvesenet's Traffic Data Portal.

For reproducibility purposes, the majority of this project was developed as a Python workflow using Whitebox Tools, Rasterio, and Geopandas. The code is available on GitHub at pmhalvor/geo4460/mca.

NOTE: In this initial draft, I've kept all sections as bullet points for now. This was a temporary decision, to easier see which parts are neccessary for the final product, and which should parts could be cut (since we are already at roughly 12 pages).

2 Data

Data sources:

- Strava API
  - Segments
  - Activities
- Statens Vegvesen Traffic Data
- Kartverkets N50

3 Method

3.1 General Development

3.1.1 Development environment

- Chose to write everything in Python for quicker reproducibility.
- Cross checked feature layers in ArcGIS Pro to verfiy results.

- Leveraged Dask for parallel processing of independent feature layer generations.

3.1.2 Data processing

- Data came in many formats:
  - Strava: JSON w/ polylines (Google Polyline Encoder)
  - Statens Vegvesen: CSV w/ coordinates
  - Kartverket: Geo-Database
- Data was converted to GeoDataFrames using Geopandas for querying (join, intersect, filter, etc.)
- Feature layers saved to vectors, raster, and shapefiles for downstream use

3.1.3 Verification

- Each feature layer can be run on its own to verify results, with outputs:
  - Relevant layer file (`.gpkg`, `.tif`, or`.shp`)
  - A folium-map visualization of the layer (as an `.html` file)
- Final and intermediate layers loaded into ArcGIS Pro to during development to visually inspect for artifacts and errors.
- Raster layers were also evaluated with RMSE against train-test-split data to ensure generalizability.

3.2 Feature Layers

3.2.1 Strava Segments

- Strava segments were found based on a few methods for generating interesting geo-points:
  - **explore**:
    * Manually define coordinates for specific locations around Oslo
    * Convert coordinates to points
  - **segments from simple road diff**:
    * Build layer with road segments missing bike lanes (from Road feature)
    * Convert segments to randomly sampled points
  - **segments from activity diff**:
    * Find activities with sections not currently represented by downloaded segments
    * Convert segments to randomly sampled points
- For each set of points above, segments from the Strava API were downloaded by the following steps:
  - Build bounding boxes of sizes 2km, 5km, and 10km around each point
  - Send boxes into the explore segmetns API endpoint
  - Store the segments returned (max 10 per request), filtering to only include rides
- Segments, along with their details like effort count, star count, athlete count, and encoded polylines, were saved to a GeoDataFrame
- Polylines were decoded using the Google Polyline Decoder and added to the GeoDataFrame
- The following popularity metrics weer defined for each segment:
  - **effort count / age**: number of times the segment was ridden relative to when it was created
  - **star count / age**: number of times the segment was starred relative to when it was created

  - **athlete count /age**: number of unique athletes who have ridden the segment relative to when it was created
- A layer (gdf) was created for each metric, with the metric value as an attribute to each segment in the layer
- Rasters were visually verified using folium maps and ArcGIS Pro
- The layers were saved to a GeoPackage for downstream use

3.2.2 Average Speed Heatmap

- Personal activities of type `ride` were downloaded from Strava, stored as json, containing:
  - Activity ID
  - Start time
  - End time
  - Distance
  - Polyline (encoded)
- Activities details were then fetched to get average speeds per split (per 1 kilometer)
- Polylines were decoded using the Google Polyline Decoder, then converted to randomly sampled points, preserving their speed information
- The points were then converted to a raster layer using Inverse Distance Weighting (IDW), with parameters configurable from config.py
- Coordinate system had to be manually set and reprojected to EPSG:4326 to align with visualization method
- Results visually verified using Folium maps
- Paths to raster passed on to next steps

3.2.3 Roads

- N50 data for Oslo region downloaded from GeoNorge as GeoDatabase
- Road layer (samferdsel) was loaded as a GeoDataFrame
- Bike lanes filtered and stored as separate layer
- Difference between all roads and bike lanes was calculated and stored
- Difference between simple roads and bike lanes was calculated and stored
- Paths to layers passed on to next steps

3.2.4 Traffic

- All car stations in Oslo selected from the Traffikkdata Portalen
- Hourly data for the month of May 2024 was downloaded as a csv
- Station coordinates downloaded separately as a json, with columns:
  - id
  - name
  - location: coordinates: lat/lon
- Traffic data loaded extracting the relevant columns
  - Trafikkregistreringspunkt -> "station_id"
  - Dato -> "date_str"
  - Fra -> "time_str"
  - Fra -> "datetime_str"
  - Trafikkmengde -> "volume"

- Station coordinates were loaded with lan/lon converted to points, and merged with the traffic data
- Data then grouped by time of day:
  - morning: 00-08
  - daytime: 08-16
  - evening: 16-24
- An IDW raster was genereated for each time of day, with parameters configurable from config.py
- Coordinate system had to be manually set and reprojected to EPSG:4326 to align with visualization method
- Raster masked to only include Oslo region (using a unioned layer of all N50 land covers)
- Results visually verified using folium
- Paths to raster passed on to next steps

3.2.5 Elevation / Slope

- Same N50 data for Oslo region contained contour lines
- Contour lines were loaded as a GeoDataFrame
- Lines converted to points for interpolation
- points split into train/test sets
- An elevation DEM was generated using with following interpolation methods and their parameters configurable from config.py:
  - IDW
  - Natural Neighbors
  - Triangulated Irregular Network (TIN) gridding
- The DEM was then used to generate a slope raster layer
- Coordinate system had to be manually set and reprojected to EPSG:4326 to align with visualization method (double check)
- Raster masked to only include Oslo region (using a unioned layer of all N50 land covers)
- RSME was calculated on train/test data helping choose best interpolation method
- Results visually verified using Folium maps
- Paths to raster passed on to next steps

3.2.6 Cost

- Combine slope, speed, and roads to generate a cost layer
- Weights decided based on the following:
  - Slope: 0.5
  - Speed: 0.5 (speeds higher that 21.8 km/h were given negative weights as reward, "easier" to ride)
- Roads used as a binary mask. Non-roads got impossibly high cost (max value)
- Cost layer was generated using IDW interpolation with parameters configurable from config.py
- Coordinate system had to be manually set and reprojected to EPSG:4326 to align with visualization method (double check)
- Raster masked to only include Oslo region (using a unioned layer of all N50 land covers)
- Results visually verified using Folium maps
- Paths to raster passed on to next steps

3.3 Overlays

- Relevant feature layers combined as part of the multi-criteria analysis
- Each layer was reprojected to the same coordinate system (EPSG:4326)
- Each layer was reprojected to the same resolution (10m) (TODO! Could be problem with cost.)

3.3.1 Diagram

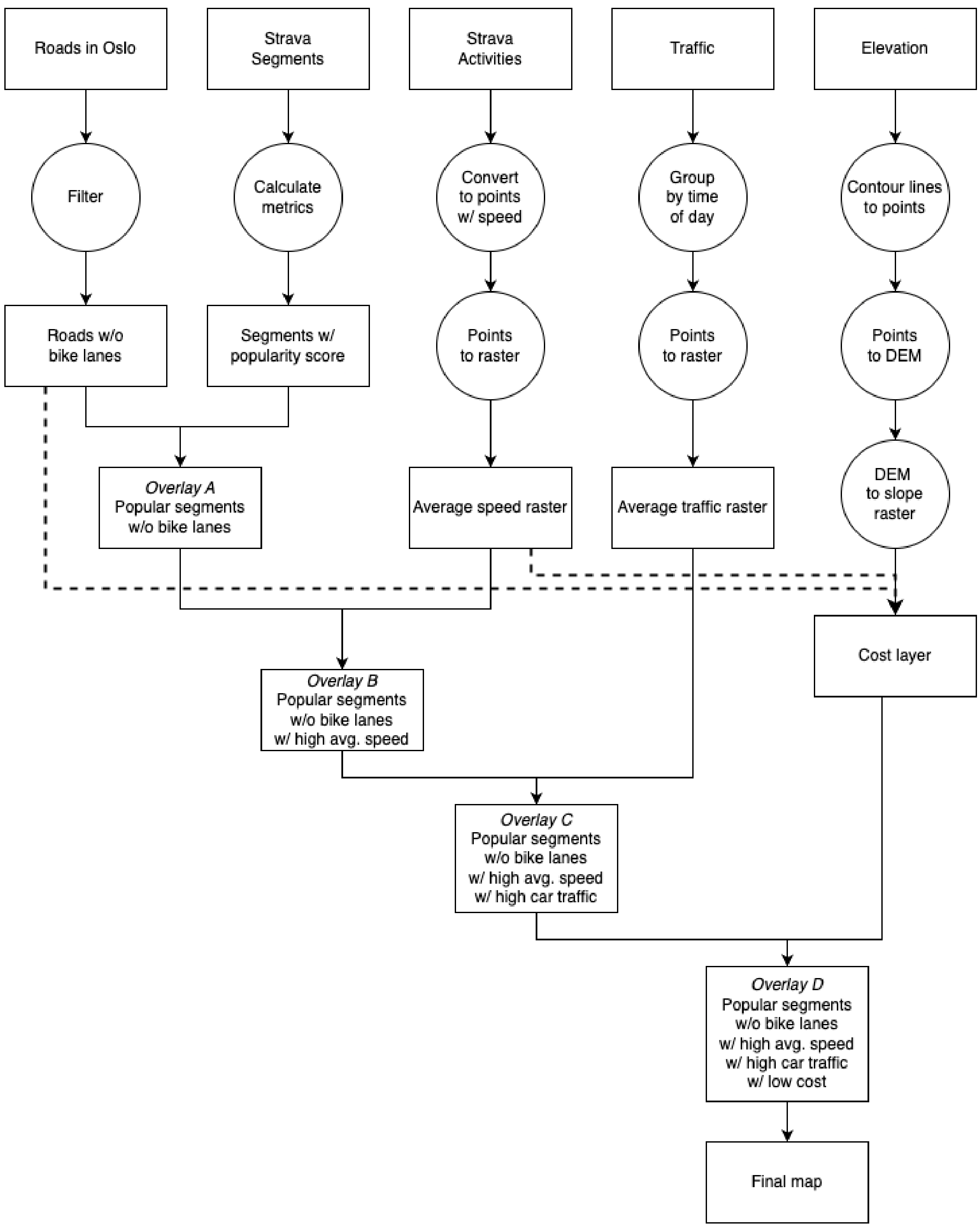


Figure 1: Workflow diagram showing the steps taken to generate the final cost layer.

3.3.2 Steps:

A. Popular segments on roads missing bike lanes

B. Overlay A w/ high speeds

C. Overlay B w/ traffic data (daytime)

D. Overlay C w/ cost

Final output is a GeoDataFrame with segments that:

- Are on roads missing bike lanes
- Are popular (effort count, star count, athlete count)
- Have high average speeds
- Have high traffic volumes
- Have low cost (easy to ride)

3.3.3 Weighting

Final ranking of segments required weighting of each layer based on their importance to the final output. The following weights were proposed during development:

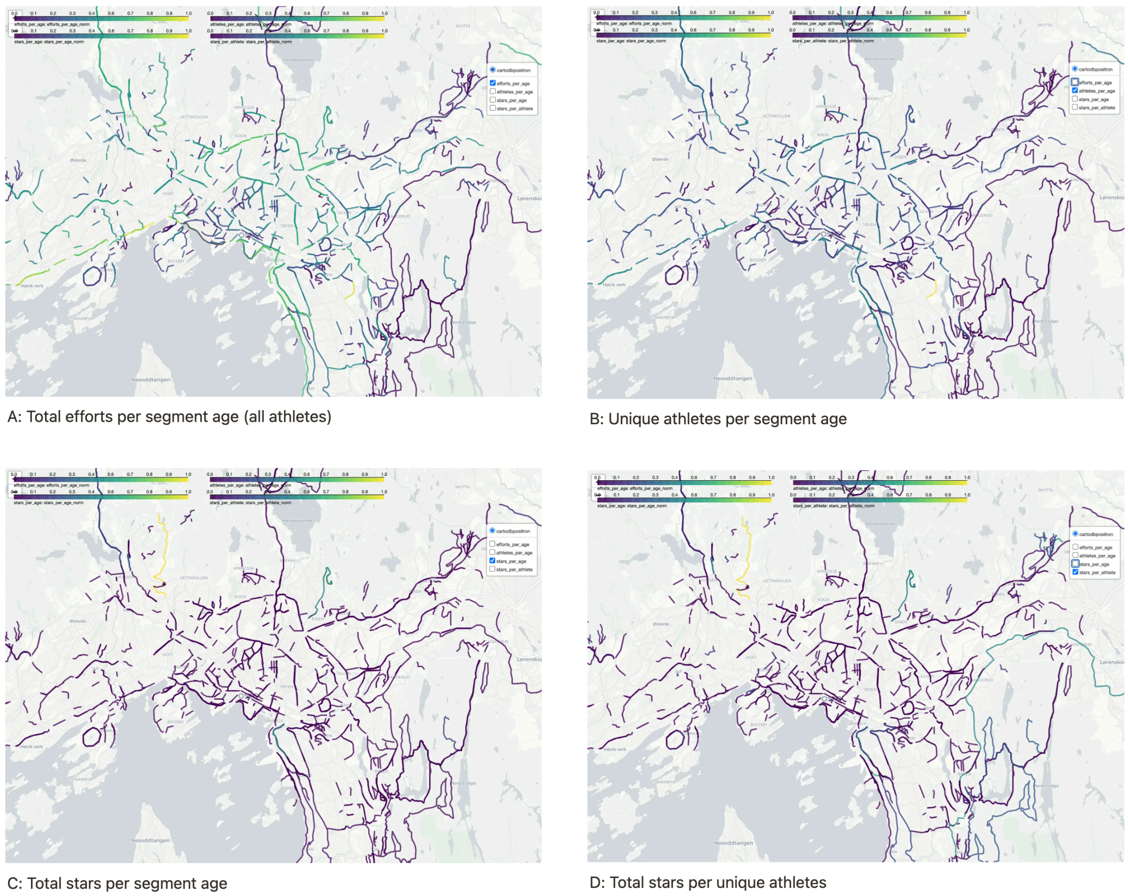
- Popularity: 0.60
- Speed: 0.25
- Traffic: 0.20
- Cost: 0.05

Popularity gets the highest weight as it highlights the most ridden segments, i.e. more people would benefit from bike lanes here. Speed gets the second highest weight as it is a good indicator of potentially dangerous roads, where cyclists have to share the road with cars. Traffic was assumed to also be a good indicator of dangerous roads, but we realized through development that relative traffic is hard to calculate, especially when some traffic stations are along the highway (with high volumes) and others are on smaller streets within the city center. Cost was considered the least important as we were not able to properly tune the thresholds to filter out a significant number of segments. Also, when planning new bike lanes, especially in a hilly city like Oslo, focusing too much on cost (measuring effort exerted by cyclists) would likely skew results to only building bike lanes on the flat areas of the city (less practical for communters).

4 Results

4.1 Feature Layers

4.1.1 Strava Segments



A: Total efforts per segment age (all athletes)

B: Unique athletes per segment age

C: Total stars per segment age

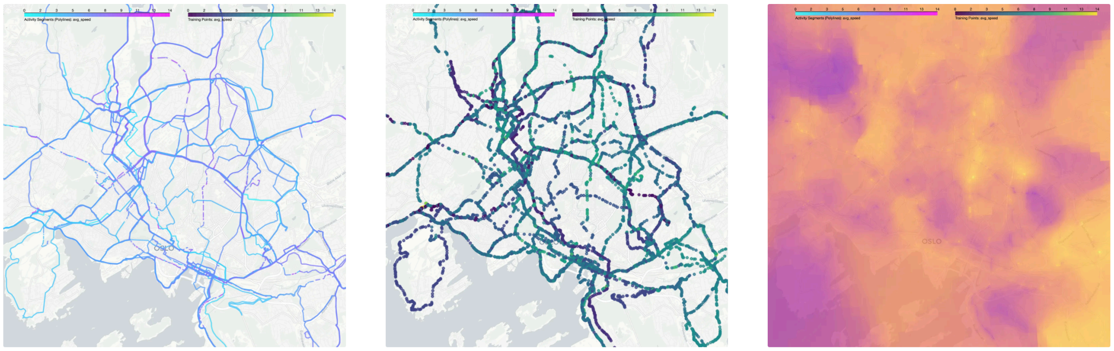
D: Total stars per unique athletes

Figure 2: Strava segments with popularity metrics (efforts per age, athlete per age, star per age, stars per athlete) overlaid on the Oslo region. The segments are color-coded based on their respective metric values, blue being low and yellow being high. For a larger view, view the raw html file here: segment_popularity_multi_layer_map.html (TODO: update after merge)

We decided to use the `efforts_per_age` as our default popularity metric. That layer gave the most variable metric values, providing us with the most informative perspective of the segments we are looking at.

In both of the `stars_per_*` layers, we see there is one segment in Vettakollen that has many stars relative to the amount of athletes and years its been present in Strava. This must be a part of some race around Nordmarka that was newly created, meaning many athletes recently transversed it after it was created. It's high value skews the rest of the segments, making everything else seem very unpopular.

4.1.2 Average Speed Heatmap



A: Activities as polylines B: Polylines to train points C: IDW interpolation raster

Figure 3: Show the steps take to generate the heatmap. From activity polylines to points to raster. For a larger view, view the raw html file here: average_speed_map.html (TODO: update with most recent version)

Speed was only available from my personal Strava user's activities, limiting the overall generalizability of this layer. This is likely due to data privacy restrictions enforced by Strava. Given this, the speed data available still provides a good representation of some of the bottle necks in the city.

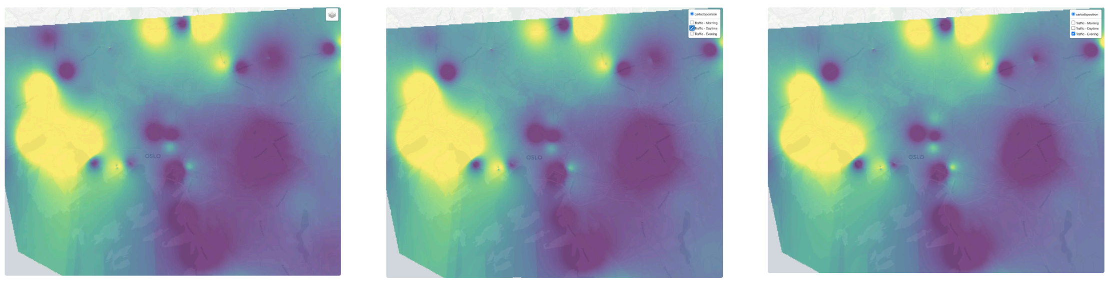
One consideration along the way was to potentially produce a slope-raster using a heatmap as an input "DEM". Those outputs should show where we observe sharp transitions from high-to-low speed, potentially pointing out dangerous areas. Such an analysis was however down-prioritized due to time restraints.

Direction of travel was also considered as an interesting caveat to this average speed raster. No directional information was included, which means points with speeds from trips up a hill could appear directly next to points with speeds from trip traveling down a hill, thus having vastly different speeds in close proximity. We acknowledge this as an over-simplification of the data, but had to make this limitation also due to time restraints.

As map (b) in Figure 3 shows, only a hand full of points were used for layer generation from the original activity data. This could have also introduced some bias into our output layer. We tried to, however, mitigate this bias as much as possible using an RMSE evaluation on an external test data set. Given that RMSE scores for the test set were generally higher than for the train set, with test set sizes 1/4 of the train set, we acknowledge that we were not entirely successful with this bias mitigation. However, there will always be a trade over with bias and variablility when producing raster from a subset of data, and we feel the final product from this layer is a good enough representation of average speeds around Oslo, especially when compared against the underlying road network.

A final comment on these results is how big of a challenge CRS alignment was. We noticed this due to the small yellow or purple dots, as can be seen in map C, not perfectly aligning with the underlying road network. After a lot of trail and error, we realized that interpoaltion outputs from Whitebox tools need to have the CRS first manually set, then be reprojected into the map coordinate system (4326). Once this was discoverd, we started producing heatmap plots that showed hotspot zones perfectly aligning with roads, meaning the heatmaps were finally correctly projected. Usually, such extreme points are deemed as unwanted artifacts in output rasters, or signal that the rasters may be slightly overfit. However, here they actually served as a good indicator on how well our layer represented the training data.

4.1.3 Traffic



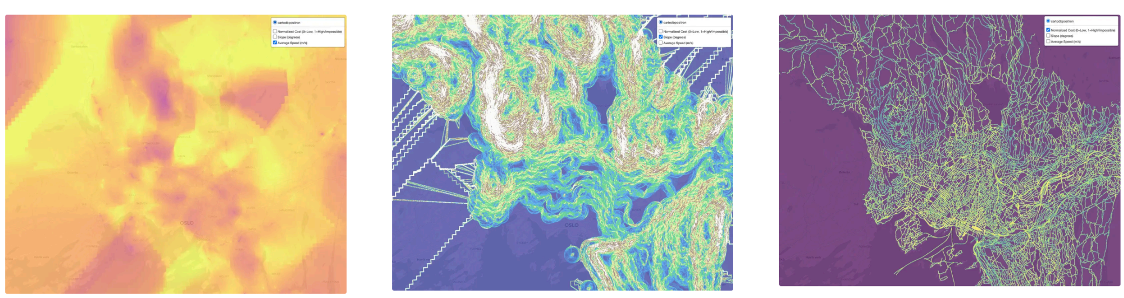
A: Traffic during morning hours (00h-08h) B: Traffic during daytime (08h-16h) C: Traffic during evening (16h-24h)

Figure 4: Traffic rasters for Oslo from the monthly volume averages during morning, daytime, and evenings in May 2024. For a larger view, view the raw html file here: traffic_density_daytime_map.html (TODO: update after merge)

We noticed there was not much variation between the 3 time-of-day in which our traffic data was grouped into. This is likely due to the over simplification of using the raw volume count values from the stations. Had we instead convert these volume counts to some relavtive metrics, based on each station's own max/min, then we might have observed a much more nuanced traffic raster. Given the little variation between rasters, we chose to just use the daytime raster for downstream processing.

The initial idea of build a raster for traffic instead of points with buffers was that areas of the city with many cars flowing through one point likely correlated to the number of cars traveling _around those points too (for example in side streets). We recognize this as a simplistic assumption, that prioritizes highway stations, as can be see by the high values around Skøyen/Bygdeøy.

4.1.4 Cost



A: Average speed heatmap B: Slope layer C: Cost layer

Figure 5: Cost layer (C) generated from the speed (A) and slope layers (B), confined to roads. For a larger view, view the raw html file here: cost_layer_visualization.html (TODO: update after merge)

The slope layer, as seen in Map B of Figure 5, has some undesired artifacts, seen as the step-like diagonals to the left and right of the raster. Similar artifacts are faintly observable in the average speed heatmap, shown in Map A. We assume the artifacts are due to missing data in these areas. Had we applied a mask clipping only the Oslo city limits, these artifacts would no longer be present.

Map C shows the final cost layer, with the inverse `vridirs` colormap, meaning yellows represent *low* values and indego as *high* values. To generate this map, we applied a buffer to the road network, and used this buffer layer as a mask. That allows us to mark cells that were not corresponding to any road, path, or other forms of transport as impossible to transverse. The coloring inside the map also seems reasonable, with the flatter regions of the city main yellow, and the sleeper regions more greenish/blue.

Average speed was used an an indicator here to again highlight bottle necks, like red lights or areas with many pedestrians. Using these data together with slope helps mitigate the aforementioned problem of loss of directional information in the speed plots and very steep hills. A steep hill might usually mean higher cost if traveling upwards, but if the average speed was usually high for this area, the two factors might cancel each other out.

4.2 Root-mean-square error (RMSE)

RMSE was calculated when finding the best elevation and heatmap interpolation methods. The below tables show some RMSE results from those experiments.

4.2.1 Elevation

Interpolation Method	Cell Size	Train RMSE	Test RMSE	Num Train Points	Num Test Points	DEM TIN Max Triangle Edge Length	DEM IDW Weight	DEM IDW Radius	DEM IDW Min Points
nn	15.0	1.1774	2.5437	113417	28355	100.0	1.0	500.0	5
tin	15.0	1.3550	2.4741	113417	28355	100.0	1.0	500.0	5
tin	15.0	1.1440	2.5677	113417	28355	1000.0	1.0	500.0	5
idw	15.0	10.8700	13.0014	113417	28355	1000.0	1.0	500.0	5
tin	15.0	1.3704	1.3618	113417	28355	50.0	1.0	500.0	5
tin	15.0	1.2366	2.6380	113417	28355	150.0	1.0	500.0	5
tin	15.0	1.1508	2.5751	113417	28355	350.0	1.0	500.0	5
tin	15.0	1.1439	2.5732	113417	28355	500.0	1.0	500.0	5
tin	15.0	1.1429	2.5686	113417	28355	750.0	1.0	500.0	5
tin	15.0	1.1442	2.5680	113417	28355	900.0	1.0	500.0	5
tin	15.0	1.1440	2.5677	113417	28355	1000.0	1.0	500.0	5

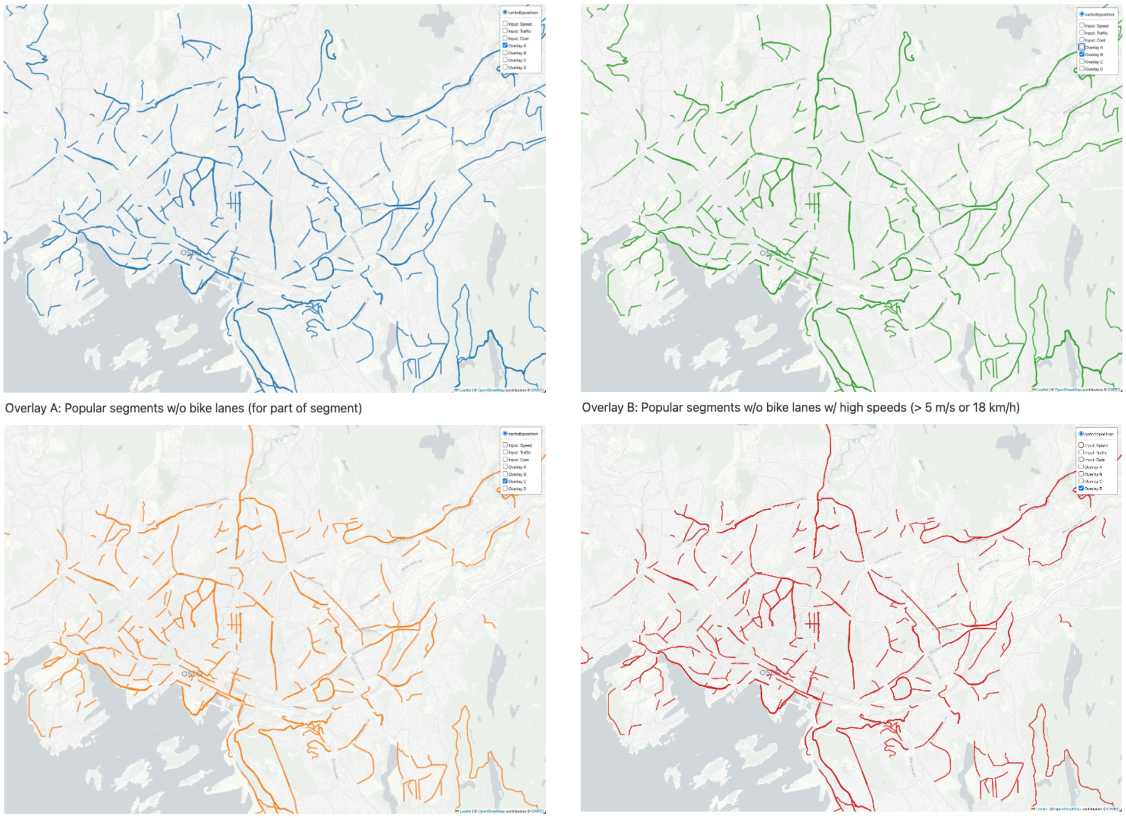
Table 1: RMSE results for the elevation interpolation methods. The best method was found to be TIN with a max triangle edge length of 1000m, with a RMSE of 2.5677 on the test set. The IDW method performed poorly, with a RMSE of 13.0014 on the test set. The number of train/test points is also shown, along with the parameters used for each method. The cell size was set to 15m for all methods.

4.2.2 Average heat map

Train RMSE	Test RMSE	Cell Size	Weight	Radius	Min Points	Train Points	Test Points
0.5701	1.5128	15	1.5	500	15	4136	1034
0.5841	1.5008	15	1.5	1000	15	4136	1034
0.5048	1.6118	15	2.3	1000	15	4136	1034
0.5154	1.5756	15	2.0	1000	25	4136	1034
0.5154	1.5781	15	2.0	1000	50	4136	1034
0.5147	1.5816	15	2.0	500	50	4136	1034
0.3907	1.5760	10	2.0	500	100	4136	1034
0.3908	1.5756	10	2.0	500	125	4136	1034
0.4123	1.5457	10	1.75	500	125	4136	1034
0.5559	1.4783	10	1.25	500	125	4136	1034
0.5631	1.4781	10	1.25	500	150	4136	1034
0.5652	1.4789	10	1.25	750	150	4136	1034
0.4133	1.5457	10	1.75	1000	150	4136	1034
0.4139	1.5449	10	1.75	1000	250	4136	1034
0.5889	1.4208	10	1.75	1000	250	8252	2063
0.8081	1.4012	10	1.75	1000	250	20670	5168
0.8042	1.4019	10	1.75	500	250	20670	5168
1.0872	1.3609	10	1.0	500	250	20670	5168
1.0640	1.3485	10	1.0	500	150	20670	5168
1.0640	1.3485	15	1.5	500	15	20670	5168
1.0640	1.3485	10.0	1.0	500.0	150	20670	5168
1.2937	1.3487	100.0	1.0	500.0	150	20670	5168
1.1913	1.3446	25.0	1.0	500.0	150	20670	5168
6.7490	6.7633	10.0	1.0	500.0	150	20670	5168
1.1868	1.3570	25.0	1.0	500.0	150	20680	5171
1.1821	1.3683	25.0	1.0	500.0	150	20655	5164
1.6717	1.7921	25.0	1.0	500.0	150	15313	3829
1.6133	1.7994	10.0	1.0	500.0	150	22971	5743

Table 2: RMSE results for the average speed interpolation methods. The best method was found to be IDW with a weight of 1.75, with a RMSE of 1.3485 on the test set. The number of train/test points is also shown, along with the parameters used for each method. The cell size varying during experimentation, for faster processing during other debugging.

4.3 Overlays



Overlay A: Popular segments w/o bike lanes (for part of segment)

Overlay B: Popular segments w/o bike lanes w/ high speeds (> 5 m/s or 18 km/h)

Overlay C: Popular segments w/o bike lanes w/ high speeds & high traffic (>100 car/h)

Overlay D: Pop. segments w/o bike lanes w/ high speeds, high traffic, & low cost

Figure 6: Overlay steps A, B, C, and D. The final output is a GeoDataFrame with segments that are on roads missing bike lanes, are popular (effort count, star count, athlete count), have high average speeds, have high traffic volumes, and have low cost (easy to ride). For a larger view, view the raw html file here: combined_overlays_map.html (TODO: update after merge)

The final outputs currently show many segments that could be potential candidates for future bike lanes. We recognize that there are a few segments that overlap with known bike lanes, meaning there is likely bug in out bike lane removal of roads which should be further investigated.

Given there are still many segments left, we could be even more strict with our thresholding, to ensure only a handful of final segments remain. The current status of the project is a great starting point for further investigation into answering our research question.

In conclusion, we see quite a few suggestions for future bike lanes. Though another iteration of this analysis needs to include roads with buffers and then filter away bike lanes, to ensure that only segments w/o any bike lanes are considered in the final overlay. With the current settings, we notice the most filtering happens when considering areas with high average speeds and high traffic. Thresholds for these criteria could be tuned a bit more, to be even more selective. A final ranking output based on the criteria weighting could also help find the most pressing routes that need a future bike lane.

5 Discussion

5.1 Experimentation

5.1.1 Inidivdual layers

- Each layer having its own callable `main` sections allowed for faster debugging when a single layer's results seemed off
  - Downside was keeping these in sync when updating code to bette suite higher level functions
- Displaying in Folium maps right away also helped inspect each layer, giving visual feedback typical to GUI based GIS software like ArcGIS Pro
  - Downside was that underlying CRS conversions and explicit CRS definitions not always behaving as expected
- Modular approach helps with scalability, and reusing code for later work
- Code can be refactored to be more generalizable for potentially future MCAs

5.1.2 Parameter controll through `config.py`

- Used together with version-control like `git` and evaluation checks (RMSE)
- Very helpful for keeping track of which parameters created which outputs
- Easily passed between tasks, to ensure the same settings and output files are used for a single execution
- Output directories with date and time for local organization of experiment results
- Using `pathlib.Path` module to ensure files exist before executing pipeline helps to ensure inputs are present before running

5.2 Limitations/Challenges

5.2.1 CRS management

- Biggest problem during development was often due to coordinate system management
- Originally chose to use default EPSG:25833
  - Good for Whiteboxtools (interpolation) and N50 data
  - Bad for visualization and Strava data
- Had to convert all Strava data from EPSG:4326 to EPSG:25833
- Had to reproject to EPSG:4326 for visualization with Folium
- All of this was discovered throughout development, so still echos in code of unnecessary reprojections and crs checks
- In ArcGIS Pro, this is often handled automatically, so not often something we've thought about previously

5.2.2 Rasters, points, or polylines?

- Every layer represented different data
- Needed to decide what made most sense to represent the layer: raster, points, or polylines
- For example, segment layers were orignally made as rasters, giving results hard to interpret
- Polylines work great but not always exact overlap, making hard to filter away
  - Could be solved with **buffers**, but then why not go all the way with raster?

5.2.3 Roads without bike lanes

- Polylines representing roads independent from bike lanes
- Can see in final results that some roads we know have bike lanes still show up
  - Is this due to the segment touching a part of road w/o bike lane?
  - Or is this rather due to bike lines represented as separate geomtries as roads?
- This relates to the above point of how to represent the data

5.2.4 Un-normalized traffic data

- Traffic volumes vary greatly depending on the road type the stations are measuring
- Highways will obviously have higher traffic than inner city streets
- No good way to normalize fairly
- Should have maybe taken into consideration road type, and weighted accordingly

6 Summary

In this MCA, we've gotten direct experience collecting data from external APIs, converting that to relevat geo-informed dataframes, and used it to produce this analysis on cycling infrastructure around Oslo. Though results still seem inconclusive, our Python based workflow enables us to quickly run further experimentation via paramter tuning, to find the utmost important routes to consider for new bike lanes.

We've discussed some decisions made along the way, like the modular approach to each feature layer and choices around which data formats best represent each data type. Some challenges we faced during the process provided useful experience for future projects, especially when working with data and software that use varying base coordinate systems and how different regularization techniques might enable us to get even more information out of our data.

In conclusion, we found many candidate routes to propse for future bike lanes. Though due to minimal fine tuning, we are hesitatnt to draw any hard conclusions about which are the most important to propose to the client. However, the work presented in this MCA enables future findings on this topic.

7 References

- **Strava API**: https://developers.strava.com/docs/reference/
- **Statens vegvesen API**: https://trafikkdata.atlas.vegvesen.no/
- **Luftkvalitet API**: https://luftkvalitet.nilu.no/historikk/
- **N50 Map Data**: https://www.geonorge.no/geonetworktest/srv/api/records/