Multi-Criteria Analysis

2025-04-26

Per Halvorsen
pmhalvor@uio.no
GEO4460

1 Introduction

In this paper, we will look into ….

Topics we will explore:

- …

2 Data

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

3 Methods

3.1 General Development

3.1.1 Development environment

- Chose to write everything in Pytohn 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 teh segments returned (max 10 per request), filtering to only include rides
- Segments, along with their details like effort count, star count, athelete 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 user-interface
- 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 20km/h were given negative weights as reward, "easier" to ride)
  - Roads: 1.0
- 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

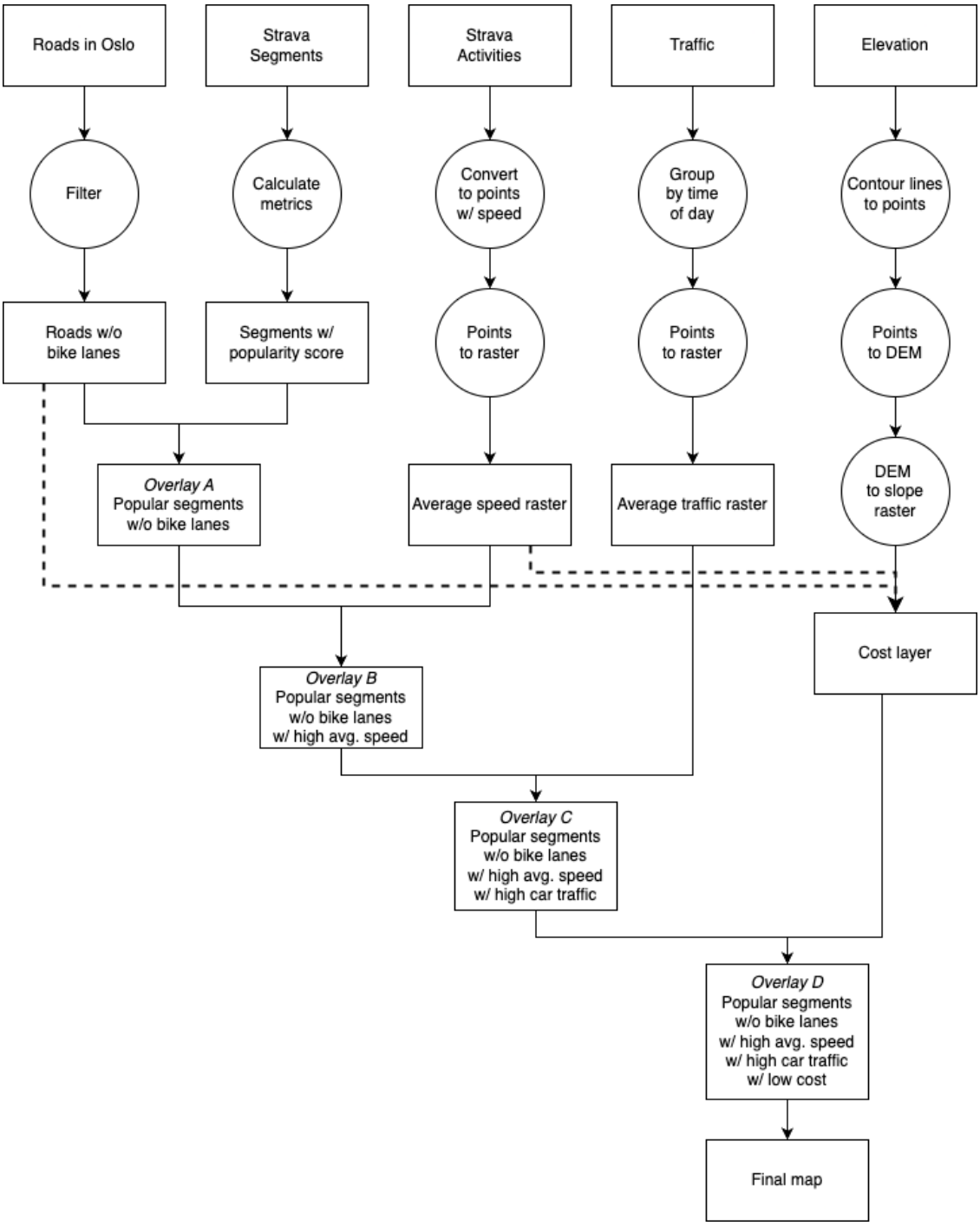


Figure 1: Workflow diagram showing the steps taken to generate the final cost layer.

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:

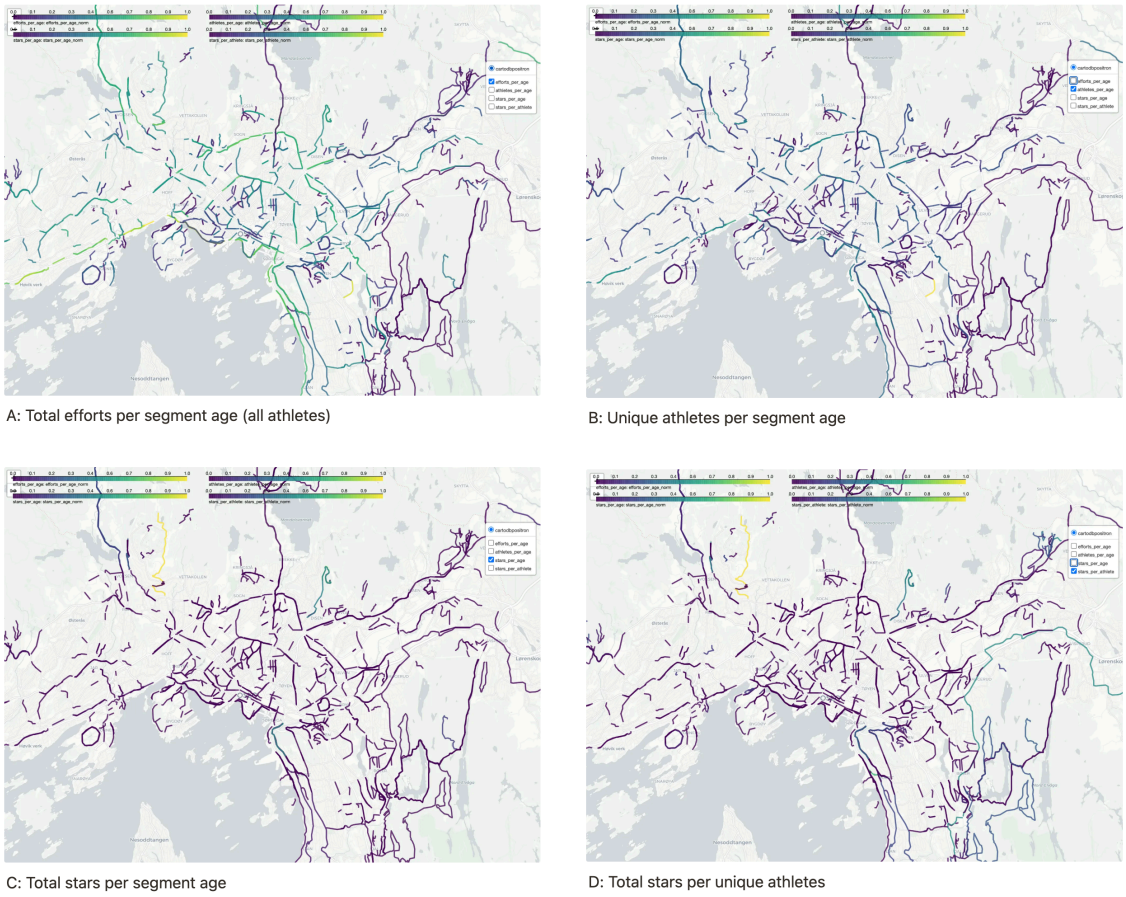
- Popularity: 0.60
- Speed: 0.25
- Traffic: 0.20
- Cost: 0.05

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)

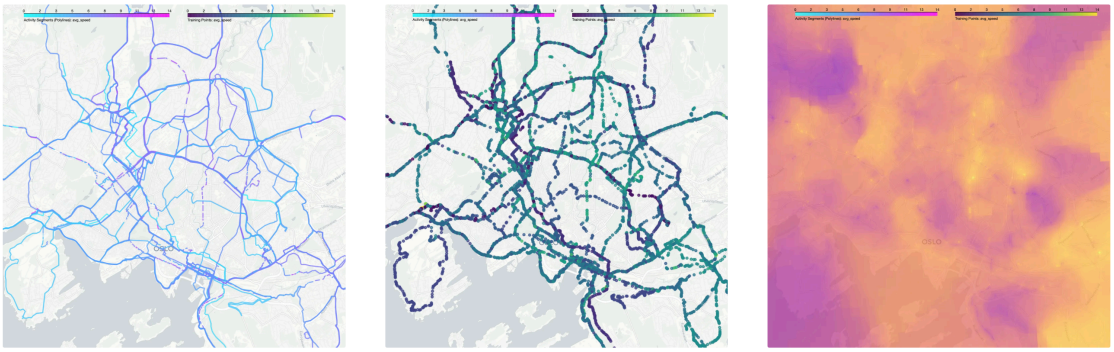
B: Unique athletes per segment age

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)

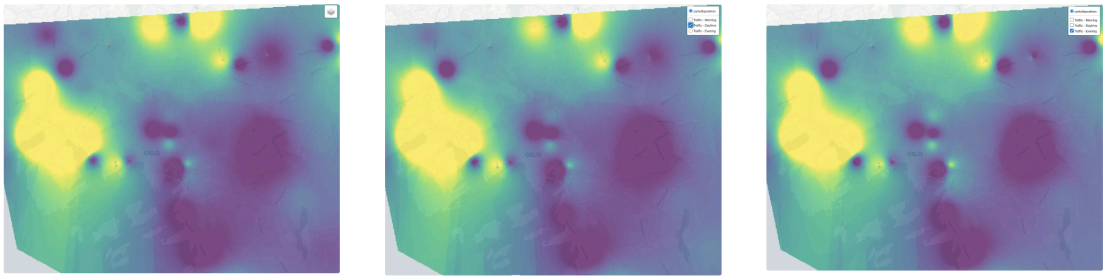
4.1.2 Average Speed Heatmap



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)

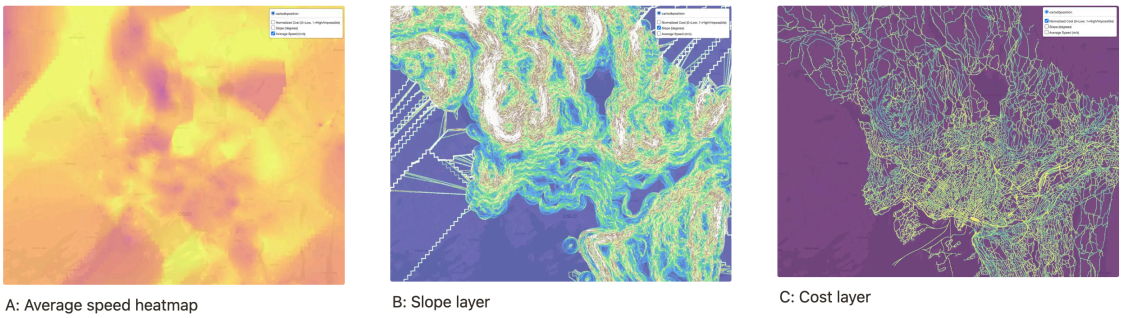
4.1.3 Traffic



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)

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)

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

Train RMSE	Test RMSE	Cell Size	Weight	Radius	Min Points	Train Points	Test Points
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)

5 Discussion

6 Conclusion/Summary

7 References

[ ]: