Per Halvorsen pmhalvor@uio.no GEO4460

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 a handful of important criteria to consider when planning future cycling routes, including route popularity, traffic, and typical cycling speed. Thresholds for each layer of interest were tuned along the way, giving us a final output layer containing only the most important roads in Oslo that should be considered as potential candidates when planning for future bike lanes.

The hypothetical client we had in mind was Oslo Kommune. Our analysis can serve as a baseline for planning of future cycling paths around Oslo. By mixing automotive traffic with real average speed calculations, we were able to find some of the most popular routes already traveled by many cyclists daily that currently serve as a threat to public safety.

We used Strava's public segment data as it is crowdsourced, with new segments being added and updated daily by all users, on any road. This is a cost-effective way of gathering data on roads and routes potentially overlooked historically, albeit being quite often traveled by many cyclists. Public segments are an informative and scalable data source that provide a novel perspective on cyclist traffic, not previously captured by traditional registration methods.

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. Both data collection and feature generation workflows can be found in the repository, along with some simple instructions on how to get started. Comments and doc-strings were also included to help guide future work on this codebase.

To ensure the results from each step in the process made sense, we displayed the output features on Folium maps. These maps enabled us to quickly zoom and pan over the observed region when jobs were finished. When more detailed verification was needed, we loaded the layers into ArcGIS Pro, which allowed us to analyze the values in each layer more thoroughly.

Data

The following datasources were used for this project:

- Strava API
  - Segments
  - Activities
- Statens Vegvesen Traffic Data
- Kartverkets N50
- Sykkel ruter i Oslo (Oslo Kommune)

Strava API

Two important data types were pulled from the Strava API, Segments (public data) and Activities (personal data). This was possible by creating an API key through my user account, providing authorization credentials for the API. A more detailed description of these data and how they were collected can be found in the Strava Segments and Activity Details sections in Method.

Traffic data

Statens Vegvesen provides an easy-to-use API to download data on traffic stations around the country. The user must select each station they are interested in, as well as a desired time frame and the frequency of data they want to download. For simplicity, we limited the traffic data to only the hourly data of May 2025 for all the car stations around Oslo.

N50 kartdata

These data are available for download from GeoNorge's data registry. The user must specify which region they are interested in, as well as the desired coordinate system. We'll mention later how the choice of coordinate system ended up being one of the biggest hurdles of this project.

Sykkelruter i Oslo

This dataset is an interactive Google Maps layer linked to from Oslo Kommune's official homepage. It has a range of bike path categorizations (separated bike paths, bike path & sidewalk, bike paths integrated with car traffic, etc.). The data can be downloaded as a KML-database, making loading and filtering desired layers with GeoPandas very straightforward. Here, we defined "bike paths" as those separate from cars and sidewalks. In Roads & Bike Paths, we'll talk more about which specific bike path categories we chose to use for our analysis and why.

Method

In this section, we outline the individual steps followed to build our MCA workflow. We start by setting up the development environment,

General Development

Development environment

- Chose to write everything in Python for quicker reproducibility.
- Cross-checked feature layers in ArcGIS Pro to verify results.
- Leveraged Dask for parallel processing of independent feature layer generations.

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.

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 during development to visually inspect for artifacts and errors.
- Raster layers were also evaluated with RMSE against train-test-split data to ensure generalizability.

Feature Layers

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 segments 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 were 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.

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)
- Activity 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.
- The coordinate system had to be manually set and reprojected to EPSG:4326 to align with the visualization method.
- Results were visually verified using Folium maps as well as ArcGIS Pro.
- Paths to the raster were passed on to the next steps.

Roads and Bike Lanes

- N50 data for the Oslo region was downloaded from GeoNorge as a GeoDatabase.
- The road layer (samferdsel) was loaded as a GeoDataFrame.
- Bike lanes were added as their own class from the KML data set.
- Three categories were used for bike lanes:
  - `Gang- og sykkelveier - vis hensyn (separate shared-use paths)`: After manual observation, we decided this category included many bike paths that were already built to remove cyclists from car traffic.
  - `Sykkelfelt (Bicycle lane)`: This is the typical bike lane on the side of the road, usually painted in red.

    - `Sykkelvei med fortau (Separate bicycle path)`: Another isolated bike lane away from traffic, which means there already exists a safe bike lane here.
    - The rest of the categories were deemed "unsafe", usually involving mixed bike and car traffic.
- A buffer was draw around the selected bike lanes from above to ensure roads running parallel to the bike paths were also excluded from the final results.
- The difference between all roads and bike lanes was calculated and stored.
- The difference between simple roads and bike lanes was calculated and stored.
- Paths to layers were passed on to the next steps.

Traffic

- All car stations in Oslo were selected from the Traffikkdata Portalen.
- Hourly data for the month of May 2024 was downloaded as a CSV.
- Station coordinates were downloaded separately as a JSON, with columns:
    - id
    - name
    - location: coordinates: lat/lon
- Traffic data was loaded, extracting the relevant columns:
    - Trafikkregistreringspunkt -> "station_id"
    - Dato -> "date_str"
    - Fra -> "time_str"
    - Fra -> "datetime_str"
    - Trafikkmengde -> "volume"
- Station coordinates were loaded with lat/lon converted to points and merged with the traffic data.
- Data was then grouped by time of day:
    - morning: 00-08
    - daytime: 08-16
    - evening: 16-24
- An IDW raster was generated for each time of day, with parameters configurable from config.py.
- The coordinate system had to be manually set and reprojected to EPSG:4326 to align with the visualization method.
- The raster was masked to only include the Oslo region (using a unioned layer of all N50 land covers).
- Results were visually verified using Folium and ArcGIS Pro.
- Paths to the raster were passed on to the next steps.

Elevation / Slope

- The same N50 data for the Oslo region contained contour lines.
- Contour lines were loaded as a GeoDataFrame.
- Lines were converted to points for interpolation.
- Points were split into train/test sets.

- An elevation DEM was generated using the following interpolation methods, with 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.
- The coordinate system had to be manually set and reprojected to EPSG:4326 to align with the visualization method (double-check).
- The raster was masked to only include the Oslo region (using a unioned layer of all N50 land covers).
- RMSE was calculated on train/test data, helping to choose the best interpolation method.
- Results were visually verified using Folium maps and ArcGIS Pro.
- Paths to the raster were passed on to the next steps.

Cost

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

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 a problem with cost).

Diagram

workflow_diagram

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

Steps:

A. Popular segments on roads missing bike lanes.

B. Overlay A with high speeds.

C. Overlay B with traffic data (daytime).

D. Overlay C with cost.

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.
- Have low cost (easy to ride).

Weighting

The 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 commuters).

Results

Feature Layers

Strava Segments

results_segments

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

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.

Average Speed Heatmap

results_heatmap

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

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.

Traffic

traffic results

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

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.

Cost

cost results

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

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.

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.

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.

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.

Overlays

results_overlay_ab results_overlay_cd

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

The final outputs currently show only a handful of segments that could be potential candidates for future bike lanes. Some parameter tuning was done during development, both of thresholds and of weights. Given our personal experience of biking in Oslo, the final routes do seem like good places to consider for future bike lane installments.

Given that there are so few segments left in the final result tells us we were maybe too strict during thresholding. We initally tuned thresholds to cover the median values of each previous overlay layer. This halfing of each overlay was eventually loosened to allow roughly the top 75% of segments through, which seemed good enough for this analysis. We point this out to highlight that a more rigorous hyperparameter tuning would be needed here after inital feedback from the client has been recieved.

In conclusion, we saw the best effects of filtering when using wide buffer sizes on our bike lane filtering and low filtering thresholds on the overlay combinations. A final ranking output based on the criteria weighting was produced by the workflow, and is available in the repository, including the segments from Overlay D, with their unique identifiers in the Strava API.

Discussion

Experimentation

Individual 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 better suit 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 did not always behave as expected.
- Modular approach helps with scalability and reusing code for later work.
- Code can be refactored to be more generalizable for potential future MCAs.

Parameter control 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 the `pathlib.Path` module to ensure files exist before executing the pipeline helps to ensure inputs are present before running.

Limitations/Challenges

CRS management

- The 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 there are still echoes in the code of unnecessary reprojections and CRS checks.
- In ArcGIS Pro, this is often handled automatically, so it is not often something we've thought about previously.

Rasters, points, or polylines?

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

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 the road without a bike lane?
  - Or is this rather due to bike lanes being represented as separate geometries from roads?
- This relates to the above point of how to represent the data.

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.

Summary

In this MCA, we've gained direct experience collecting data from external APIs, converting it to relevant geo-informed dataframes, and using 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 parameter tuning to find the most important routes to consider for new bike lanes.

We've discussed some decisions made along the way, such as 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 extract even more information from our data.

In conclusion, we found a handful of candidate routes to propose for future bike lanes. However, due to minimal fine-tuning, we recommend further refinement of the parameters used here after receiving initial feedback from the client.

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/
- **Sykkel ruter i Oslo (Oslo Kommune)**: https://www.google.com/maps/d/viewer?mid=1oO7FZepl8zXxKmx22yV91LUd1KZgYkU_&ll=59.92348788725151%2C10.730553549999948&z=11