Project report - Summer 2021

Scenarios of coastal response to Sea Level Rise: a machine learning based graphical user interface

Student: Ngoc Nhu Hoang
Mentor: Daiane G. Faller
Faculty supervisor: David M. Holland

Center for Global Sea Level Change
New York University Abu Dhabi
August 6, 2021

Contents

1 PROJECT DESCRIPTION

Future impacts caused by climate change are expected to widely affect coastal areas with landscapes encompassing a variety of responses to different sea-level rise (SLR). The landscape response is related to their geomorphology, geologic setting, ecology, and development level (Lentz et al., 2016 [1]). So, SLR can cause a range of threats to natural and built environments, and understanding SLR-induced hazards is essential for decision making. Knowing where available coastal habitats are likely to be resilient, transitioning to a new state, or requiring a buffer zone to accommodate landward translation is essential for developing management and resource allocation strategies that preserve the coastal system's intrinsic values.

The usual methodologies applied to identify the coastal response to SLR usually focus on one response. Simultaneously, alternative methods can provide a range of potential reactions like inundation or dynamic response in a specific landscape (Strauss et al., 2012 [2]). Besides inundation seems a straightforward evaluation accounting the vertical and horizontal movement of water-based in topography and projected sea level, a rigorous application requires accounting for technical and data uncertainties (Lentz et al., 2016 [1]). On the other hand, several regions will develop a dynamic response due to land use, ecological and morphological processes (i.e., erosion, deposition, tides, wave range, etc.). A technique that can make probabilistic assessments to communicate the uncertainty such as magnitudes of SLR, storminess, and extrapolation from cross-shore profiles is crucial to generate reliable information of a coastal system.

We aim to develop a probabilistic approach using a combination of Bayesian Network (BN) to calculate the probability of a long-term shoreline change given physical parameters, relative sea-level rise, and land use. BN is ideal to combine historical, current, and projections of phenomena by integrating observations to evaluate relationships between forcing factors and coastal responses. The assessment of that information can be expressed by numbers or percentages or by established likelihood terms. As such systems, in general, are multivariate and involve a comprehensive combination of variables, the use of an "adaptive and learning" process can make the task less complicated. The constructed response can cover as much of the system parameter space as possible. The platform consists of a user-friendly interface, a data-driven model using BN, and an interactive coastal response that considers different parameters in a region to determine both inundation and dynamic response by providing statistical analysis and a range of adaptation scenarios for SLR.

The Python scripts written for this project can be found on the team's GitHub repository: https://github.com/sashanksilwal/CRSLR.

2 PROJECT OBJECTIVES

The main feature of the Center for Sea Level Rise platform is to enable a user to create probabilistic scenarios of coastal response (inundation or adaptation) due to sea-level rise using an available dataset for a specific region. With this method, we will be able to:

- identify the coastal landscape dynamic response to SLR;
- guide to coastal resource management decisions;
- future SLR impacts and uncertainty against ecological targets and economic constraints.

The project is divided into four main stages:

1. Stage 1:
    - Read spatial data images as matrix
    - Develop a Bayesian Network based in examples
    - Develop visual aid for Bayesian inference
2. Stage 2:
    - Integrate spatial data (test case) and BN
    - Use real data to train the BN to create future scenarios
    - Apply the visualization
3. Stage 3:
    - Develop a user interface
    - Connect and troubleshooting
4. Stage 4:
    - Test case finalization and scientific disclosure
    - Launch the platform for user data

For the duration of my internship at the Center for Sea Level Rise, I was involved in the first two stages of the project, in which my team worked on reading the data as suitable formats, processing the datasets according to their specific nature and usage, discretizing the data into intervals and bins, building the Bayesian model with the pre-processed data, evaluating the model, as well as creating graphs and maps at various stages of the entire process to visualize and communicate the data effectively.

3 BACKGROUND

3.1 Bayesian Networks

A BN framework consists of an interactive, fast and efficient tool for modelling that evaluates the probability of an outcome based on variables causal relationship. Bayes' theorem relates the probability of one event R given the occurrence of another event O (Bayes, 1763; Gelman and others, 2004) as:

$$p(R_i|O_j) = \frac{p(O_j|R_i) \times p(R_i)}{p(O_j)}$$

The left side of the equation $p(R_i|O_j)$ represents the conditional probability of a particular response, R_i, given a set of observations O_j. For example, a distinct coastal response could be a joint occurrence of a specific sea-level rise rate and a particular coastal habitat type. The ith response scenarios refer to the number of inputs scenarios that can be considered, and the jth observations refer to the set of observations considered in each scenario. On the right side of the equation, $p(O_j|R_i)$ is the likelihood of a known response's observations. This term indicates the correlation between observation and response, such as the rate of sea-level rise and habitat type. The correlation is high if the observations are accurate and if response variables are sensitive to the observed variables. The numerator $p(R_i)$ term is the prior probability of the response, which is the probability of a particular response integrated overall expected observation scenarios. The denominator $p(O_j)$ is a normalization factor to account for the likelihood of the observations.

A BN consists of a qualitative part (a directed acyclic graph - DAG) and a quantitative part (parameters/variable specifying the conditional probability of each node given its parents described in DAG). A conditional probability table (CPT) quantifies the strength of influence between child nodes (variable) and parent nodes (Knochenhauer et al., 2013). The qualitative or quantitative value represents a state of a node. Each node must have, at least, two states and must represent all values that the node can take (Cain, 2001).

Using this procedure, BN establishes the relationships between variables and parameters using directed links so-called causal relationships which represent the conditional probabilities trained on observations, probabilistic or deterministic equations or expert (empirical). One of the main features of BNs is the robust consideration of uncertainty; BNs propagate the uncertainties in the relationships to provide predicted probability for each discrete outcome. In our model we test the ability of BNs to propagate uncertainty, perform inference and calculate conditional probabilities and structure the integration of stochastic, deterministic and expert (empirical) relationships.

3.2 Parameters and data

To make probabilistic scenarios, we need to identify shoreline phenomena and parameters that will affect the coastal response. The BN's capacity to correctly represent the processes strongly depends on the quality and completeness of input data. Accordingly, we selected predictor variables used to build the BN model (Figure 1) and will create parameters to predict the coastal response. The framework use information of SLR projections, vertical land movement and elevation to create an adjusted elevation that will be used along with land cover type to predict the coastal response based on the study conducted by Lentz et al. 2016 [1].

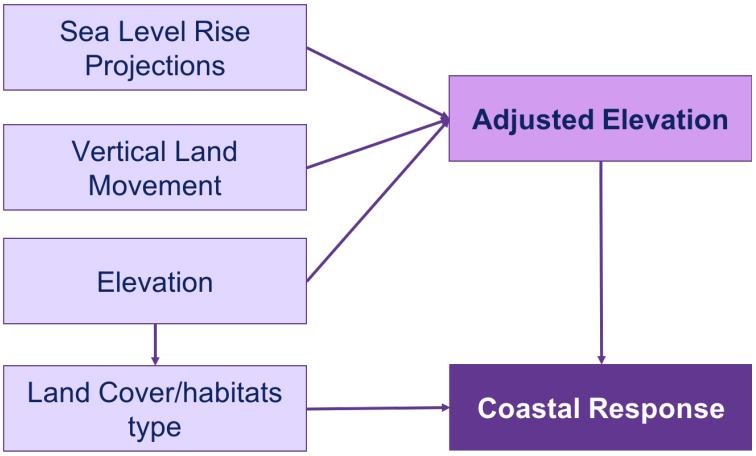

Figure 1: Schematic diagram showing the Bayesian network coastal response model.

The first step will be the adjusted elevation computation using vertical movement, elevation, and SLR projections:

$$\text{Adjusted elevation} = \text{Elevation} - \text{SLR projection} + \text{vertical land movement} + \text{uncertainties}$$

Sources of uncertainty include elevation data accuracy, vertical datum adjustments, and land vertical movement's interpolation from the original data point. These geospatially explicit input uncertainties are propagated through the model to produce a probability mass function $p(\text{adjusted elevation})$ for every grid cell. Once we determine the adjusted elevation we will combine it with the land cover/habitats and predict the coastal response.

3.3 Programming language usage

The scripts for the main program are written entirely in Python, making use of various Python packages and libraries like NumPy, SciPy, pandas, GeoPandas for data processing, pgmpy for probabilistic models-related functionalities, and Matplotlib and seaborn for data visualization.

4 DATA PROCESSING

4.1 Data reading

For the first step, we first familiarize ourselves with the data formats used throughout the project. The data is majorly stored in geospatial formats, namely GeoTIFF and Shapefile. Our first task is to research about Python libraries and packages suitable for reading these formats and ensuring the files' readiness for further processing, especially by storing them in data structures that are compatible with other Python libraries for data analysis and data manipulation.

The elevation base map is available in GeoTIFF format. We mainly use two Python libraries to process this format: Rasterio and GeoRasters. Rasterio's `open()` function was used in tandem with `read()` to return a NumPy array. All cells in the original raster file that are of null values are retained in the array, returned in the form of NumPy's `NaN`. GeoRasters provides two functions `from_file()` and `to_geopandas()` which returns a GeoPandas dataframe, compatible with further analysis using both GeoPandas and pandas. Unlike NumPy array format returned using Rasterio, this dataframe returns only original data cells that do not contain null values. Each of the eligible cell makes up a row in the dataframe.

Habitat data is available as a shapefile. GeoPandas's `read_file()` is particularly useful for this format, as it reads the file into a GeoPandas dataframe.

Throughout the processing and model training process, most of the data is structured in either NumPy arrays or GeoPandas dataframes. This way, we can more easily utilize the various data analysis and data visualization functionalities from many Python libraries that are compatible with these formats, namely NumPy, GeoPandas, pandas, matplotlib.

4.2 Elevation

The project focuses on the coastline of Abu Dhabi, capital city of United Arab Emirates, as the main case study. Figure 2 shows a rough estimation of the area of interest to the research.

The elevation map (Figure 3) serves as the base map for processing all other variables. As such, elevation data needs to be provided with high quality and accuracy. This map will be used to project the rises in sea level in different scenarios, from which we can estimate the levels of submergence for different land type across the region. Elevation data is provided in GeoTIFF format.

Figure 2: Map showing parts of Abu Dhabi coastline. Source: Google Earth.

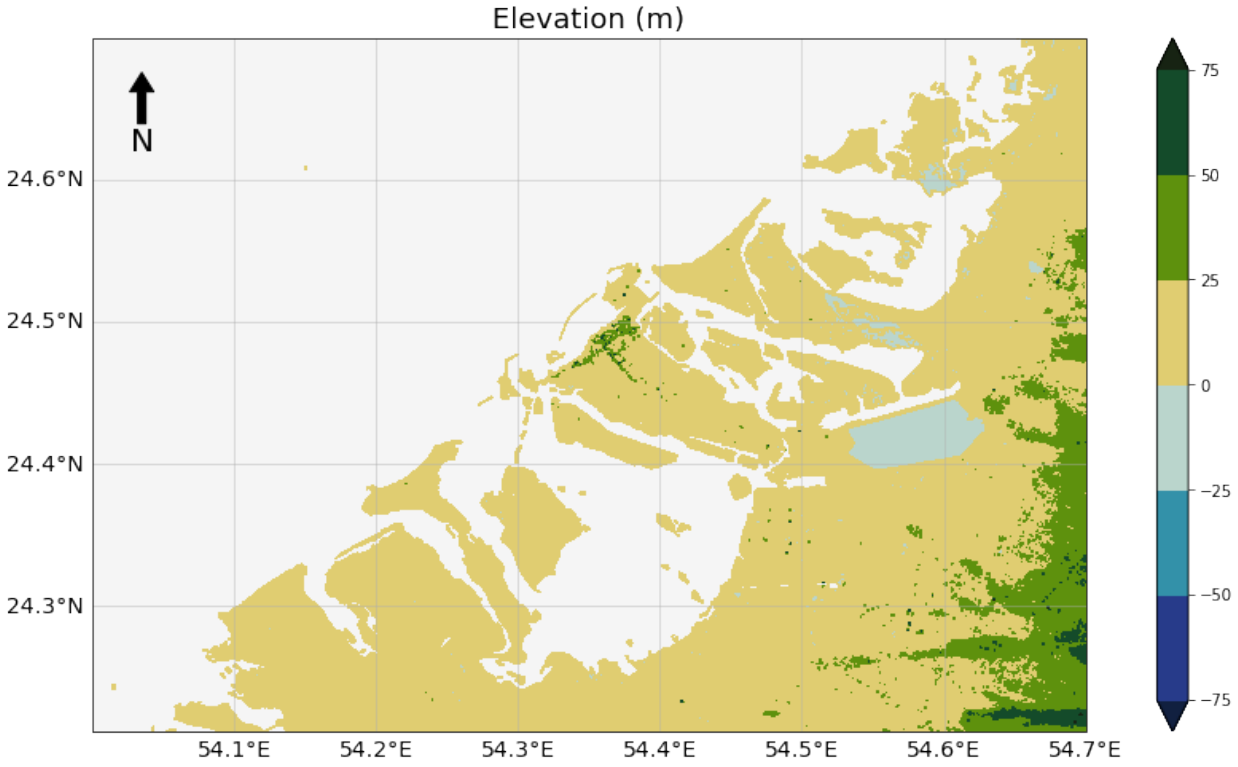

Figure 3: Elevation base map. Source: Xianwei Wang.

The elevation base map used in this project spans around 54.013°E to 54.700°E and 24.212°N to 24.700°N. When loaded as a NumPy array using the Rasterio library, the map is returned as an array of dimensions 1758 $\times$ 2521. As such, the elevation grid (the NumPy array) contains 4,431,918 data cells. When loaded with using the GeoRasters library, the resulting elevation dataframe contains 1,789,661 rows, corresponding to the number of grid cells with non-null values. Each non-null cell in the grid, and equivalently each row in the dataframe, contains an elevation value for the corresponding location on the map, measured in meters. Further inspection of the dataset shows that the max elevation value is 83m, while the min value is -89m, and the mean elevation is 7.1m with a standard deviation of about 10m.

4.3 Sea level rise projections

The project uses a wide range of sea level rise projections corresponding to different Representative Concentration Pathway (RCP) scenarios, which are associated with greenhouse gas concentration scenarios, as outlined by the Intergovernmental Panel on Climate Change (IPCC). Sea level rise scenarios used in this project are based on the study by Irani et al. [3] which examines the sea level rise level in the Persian Gulf and Oman Sea in future time windows.

Greenhouse Gas Emission					
Prediction window	Mitigation	Intermediate	Intermediate -High	High	Extreme*
	RCP 2.6	RCP 4.5	RCP 6	RCP 8.5	
2046-2065	0.18-0.34 (0.27)	0.2-0.37 (0.3)	0.2-0.36 (0.28)	0.25-0.43 (0.33)	
2081-2100	0.29-0.61 (0.42)	0.36-0.7 (0.52)	0.37-0.71 (0.54)	0.51-0.93 (0.7)	
2100	0.31-0.67 (0.49)	0.4-0.8 (0.6)	0.42-0.81 (0.61)	0.6-1.1 (0.83)	2.5

Table 1: Regional sea level rise projections. Unit: Meters. Scenarios used in this project. Source: Irani et al [3].

For each prediction time window, we choose the left end of the range in lowest scenario (Mitigation) and the right end of the range in highest scenario (High). Additionally, we added an Extreme scenario of 2.5m sea level rise as a hypothetical test case for the model. This number is identified as the average global sea level rise if the Thwaites Glacier melts completely. In total, we use seven sea level rise projections for the model. In each scenario, the sea level rise is applied to every data point using the deterministic equation to calculate adjusted elevation.

4.4 Vertical land movement

Vertical land movement is one of the key factors in studying sea level rise, as it signifies the average change of a surface over long-term periods of time. Vertical land movement may vary across the region due to isostasy or other ecological and morphological processes.

The vertical land movement data is retrieved from a global GPS vertical velocities dataset from the study of Schumacher et al. (2018) [2]. Out of over 12,000 data points in the dataset, 27 are selected to be relevant to this project. These data points are scattered across the area near the Persian Gulf and the coastline of the UAE, spanning 52.588°N to 54.960°N and 23.721°E to 24.797°E. Meanwhile, the elevation base map covers a narrower area with much higher density, containing over 1,500,000 data points, as previously discussed. The disparity between the two datasets is roughly visualized in Figure 4, with red and blue points representing the vertical land movement data and elevation data, respectively.

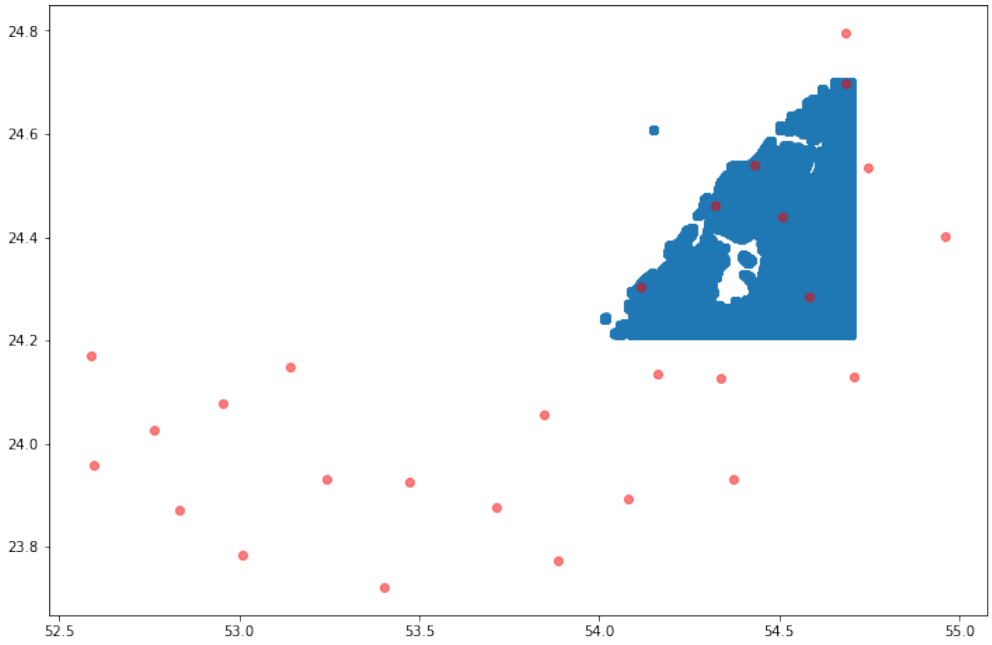

Figure 4: Comparison of vertical land movement data and elevation data.

To ensure the consistency between vertical land movement and elevation data, the vertical land movement data points need to undergo an interpolation process to generate a 2D grid containing continuous vertical land movement estimates. For this processing stage, we use two interpolation steps with two respective interpolation grids.

4.4.1 First interpolation step

This first step involves interpolating based on original vertical land movement data points. The goal of this step is to generate new data points in between, since the original data points

are relatively far apart. Using an intermediate grid before interpolating onto the elevation grid helps to create a smoother surface of estimates. For a skeleton grid to be constructed, we need to know the coordinates over which the grid span, and the dimensions of the grid. The grid boundary information is easily extracted from the coordinates of the points. To determine on grid dimensions, we must decide on the density of cells we want to put between points. We choose to base this density on the current distances among the points.

First, based on the original 27 data points, we locate the boundary points, that is, points that are southernmost, northernmost, easternmost, and westernmost. Original data points are plotted in Figure 5 and the detected boundary points are marked.

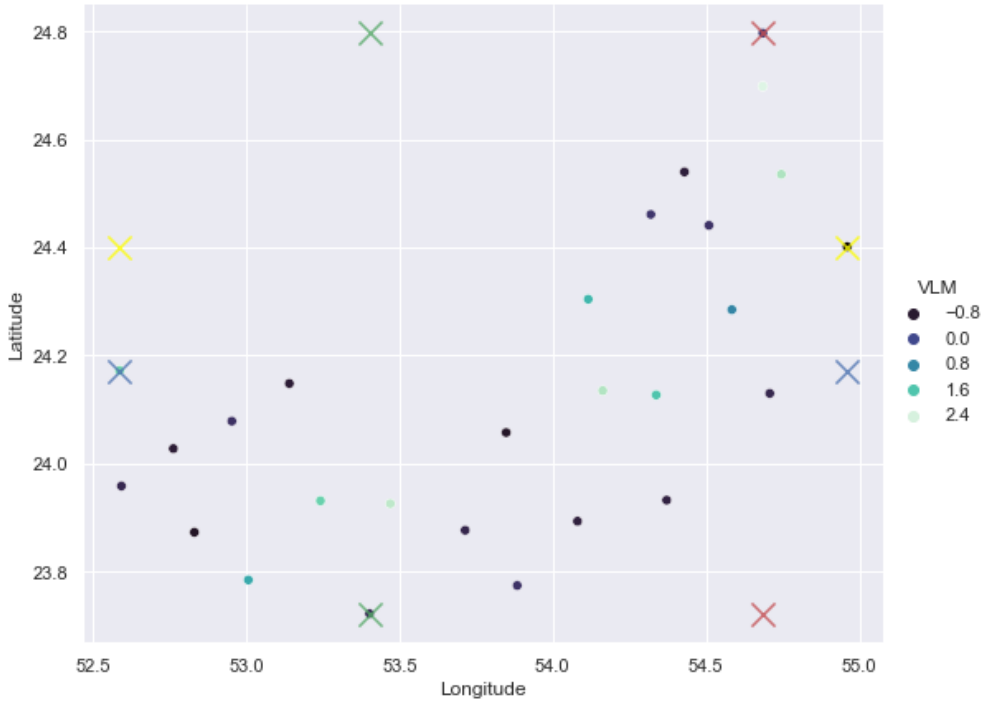

Figure 5: Boundary vertical land movement data points.

With the boundary points detected, we can generate pairs of points that span the area and calculate the distances among them. We use a function that takes in longitudes and latitudes of two points and returns the distance between them in meters. This function utilizes the Haversine formula:

$$d = 2r \arcsin\left[\sqrt{\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right]$$

in which d is the distance between the two points (in meters), φ_1, φ_2 are the latitude of point 1 and latitude of point 2 (in radians), λ_1, λ_2 are the longitude of point 1 and longitude of point 2 (in radians), and $r = 6,371,000$ is the radius of Earth (in meters).

Using the function described above, the North - South span of the vertical land movement data points is determined to be about 119,548m, and the East - West span is about 240,370m. With these estimates, we decide that generating a distance of 100m between two points will create a good density for the skeleton grid in this step. As such, the dimensions for the grid are determined to be 1196 $\times$ 2404 (1196 rows corresponding to the North - South span, and 2404 columns corresponding to the East - West span).

With the grid boundary coordinates and grid dimensions, the next step is to construct a coordinate matrix as input for the interpolation function.

We first use the `linspace()` function from NumPy to generate two 1D arrays as base horizontal and vertical coordinates:

```
x = np.linspace(x_min, x_max, nx)
y = np.linspace(y_max, y_min, ny)
```

In which `x_min, x_max` signify the span of longitudes (52.587928, 54.9595), `y_max, y_min` signify the span of latitudes (24.796595, 23.72148), and `nx, ny` signify the dimensions of the grid as discussed above. `x, y` are two arrays of dimensions (2404,) and (1196,) respectively.

The `linspace()` function returns evenly spaced numbers over a specified interval, with the first and second parameters representing the start and end values of the sequence. Since our data points lie entirely in the Northern hemisphere, the point coordinates follow a Cartesian coordinate system with lower left origin. This is the reason why we need to use `y_max` as the first parameter and `y_min` for the second parameter when calculating `y`, as opposed to the min - max order for `x`. Next, we use `x, y` as input coordinate vectors for the `meshgrid()` function:

```
xv, yv = np.meshgrid(x, y)
```

The above line of code returns two coordinate matrices, each of dimension 1196 $\times$ 2404. Each cell in the `xv` matrix contains the x-coordinate (or longitude) of the point at that location, whose y-coordinate (or latitude) is stored in the corresponding cell in the `yv` matrix. These two matrices, in turn, serve as inputs for the `griddata()` function from the SciPy library.

The `griddata()` function takes three main parameters: data point coordinates, data point values, and point coordinates at which to interpolate data. The first parameter, data points coordinates, is provided using the (x, y) pairs of the original data points (columns **Longitude** and **Latitude** in Figure 6) in the form of an array size (27, 2) (27 points, or 27 pairs of coordinates). The second parameter, data point values, is taken from the corresponding vertical land movement rates of the available data points (column **VLM** in Figure 6). Finally, point coordinates to interpolate data onto are the two arrays

	Station	Longitude	Latitude	VLM	VLM_std
0	NYWV	54.684594	24.796595	-0.207	0.223
1	NYWT	54.115618	24.303783	1.413	0.158
2	NYSM	53.848164	24.056855	-0.856	0.154
3	NYRH	53.141386	24.147625	-0.724	0.166
4	NYRB	52.587928	24.171598	1.812	0.274
5	NYQN	52.763306	24.027073	-0.724	0.135
6	NYPD	53.470797	23.925228	2.240	0.164
7	NYPB	54.162907	24.134528	2.164	0.162
8	NYON	54.429235	24.539444	-0.683	0.170
9	NYNB	54.319712	24.460688	-0.259	0.255

Figure 6: Sample vertical land movement data points.

`xv, yv` discussed previously. We also specify a `method='cubic'` parameter for the function. According to the SciPy library documentation, this directs the function to use the cubic interpolation method, which will "return the value determined from a piecewise cubic, continuously differentiable (C1), and approximately curvature-minimizing polynomial surface".

```
vlm_grid_1 = griddata(vlm_points, vlm_values, (xv, yv), method='cubic')
```

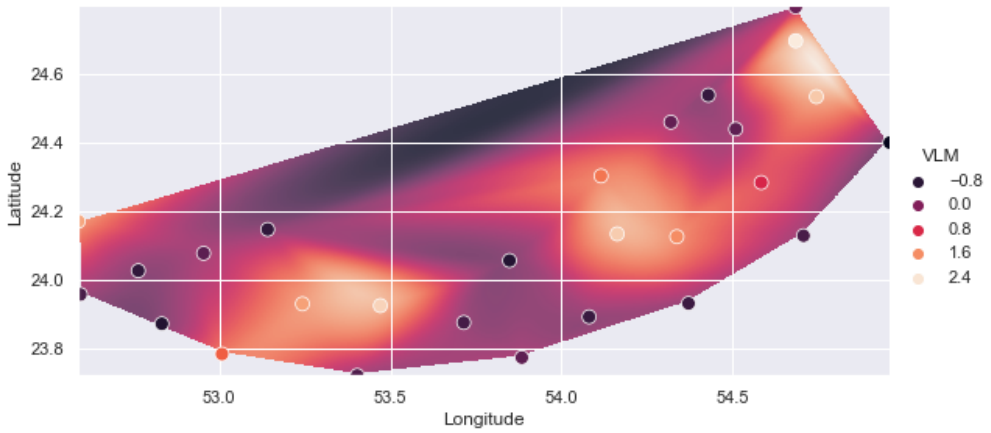

Figure 7: Result of the first interpolation step.

At the end of this interpolation step, we get an array of dimension 1196×2404

containing the interpolated vertical land movement values. The result is roughly visualized in Figure 7, with the original data points plotted on top of the interpolated values.

4.4.2 Second interpolation step

Similarly to the first step, we also use the `griddata()` function from SciPy to interpolate vertical land movement values onto a grid of the same boundary and dimensions as the elevation grid.

The first `griddata()` parameter for this step, data point coordinates, is produced by reshaping the two coordinate matrices `xv` and `yv` from dimensions (1196, 2404) to (2875184, 1), and stacking them horizontally. Similar to the first interpolation step, we have a new array of size (2875184, 2), or 2,875,184 pairs of coordinates.

```
vlm_inter_points = np.hstack((xv.reshape(-1, 1), yv.reshape(-1, 1)))
```

Corresponding to these point coordinates are the data point values, which are the results of the first interpolation step, currently stored in array `vlm_grid_1`. We flatten this array to get a vector of size (2875184,) named `vlm_inter_values`.

A skeleton grid is constructed for this step using information extracted from the elevation map. With the boundary coordinates of the elevation map available as are the dimensions of the grid, we follow the same technique detailed in the first interpolation step to produce two coordinate matrices:

```
xx = np.linspace(x_min_elev, x_max_elev, nxx)
yy = np.linspace(y_max_elev, y_min_elev, nyy)
xxv, yyv = np.meshgrid(xx, yy)
```

Finally, we are ready to run the second `griddata()` function with the linear method to get the final vertical land movement interpolation values:

```
vlm_grid_2 = griddata(vlm_inter_points, vlm_inter_values, (xxv, yyv),
             method='linear')
```

At the end of this step, we get an array of dimension (1758, 2521), the same as the elevation grid and spanning the same area. However, as we use the values from the first interpolation step as the base data point values for the second step, the newly interpolated vertical land movement values actually span a slightly larger area than the elevation data points, as visualized in Figure 8, with the interpolated vertical land movement values plotted in lower opacity. To make sure that the values between the two grids are consistent, we detect all the cells in the elevation grid that contain null values, and replace the values at the same positions in the interpolated vertical land movement grid with null values. The result of this is visualized in Figure 9.

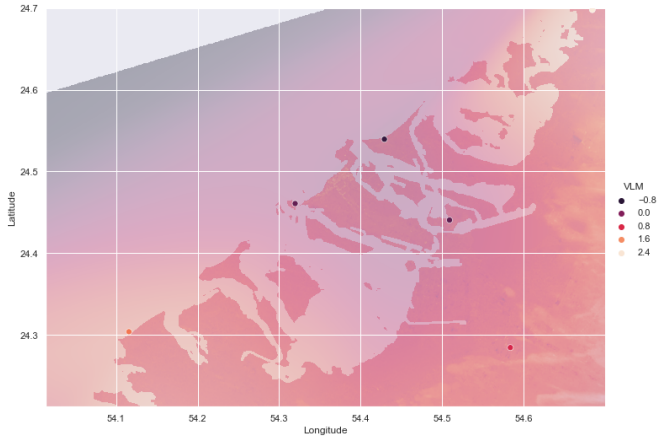

Figure 8: Result of the second interpolation step.

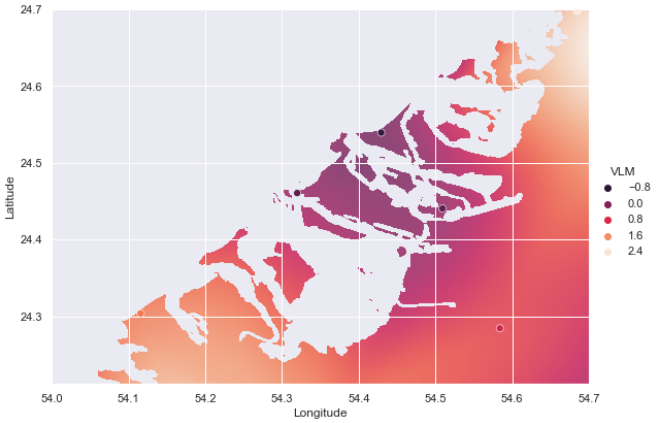

Figure 9: Interpolated vertical land movement grid after modifications.

An observation is clear when we plot the original data points on top of this grid. The elevation map spans an area that covers 6 out of the original 27 vertical land movement data points. This was also clear from Figure 4. We have the final vertical land movement map as Figure 10. Now, each cell on the base map has an elevation value and a vertical land movement estimate.

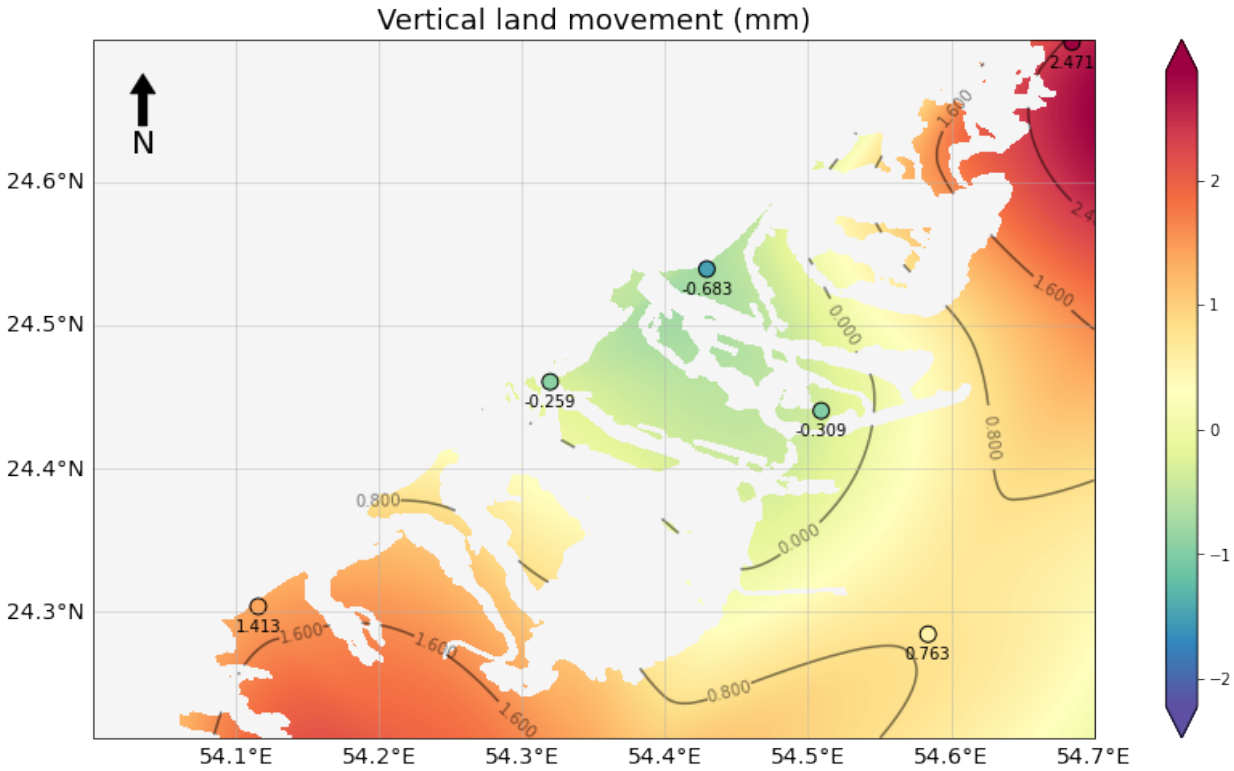

Figure 10: Interpolated vertical land movement map.

4.5 Adjusted elevation

With data on elevation and vertical land movement available, we can calculate the adjusted elevation corresponding to each of the sea level rise scenarios using a deterministic equation based on the study of Lentz et al. 2016 [1]:

$$AE = E - SLR + VLM + \text{uncertainties}$$

In which AE is the calculated adjusted elevation (in meters), E is the current elevation measures (in meters), SLR is the sea level rise projection for a particular scenario being considered (in meters), and VLM is the vertical land movement estimates (in meters). Uncertainties in this equation come from each of the boundary variables (elevation, sea level rise projections, vertical land movement) and are propagated through the network to result in the probability mass functions for the response variables (adjusted elevation and coastal response).

Figure 11 visualizes the calculated adjust elevation for one of the seven scenarios being considered in this project.

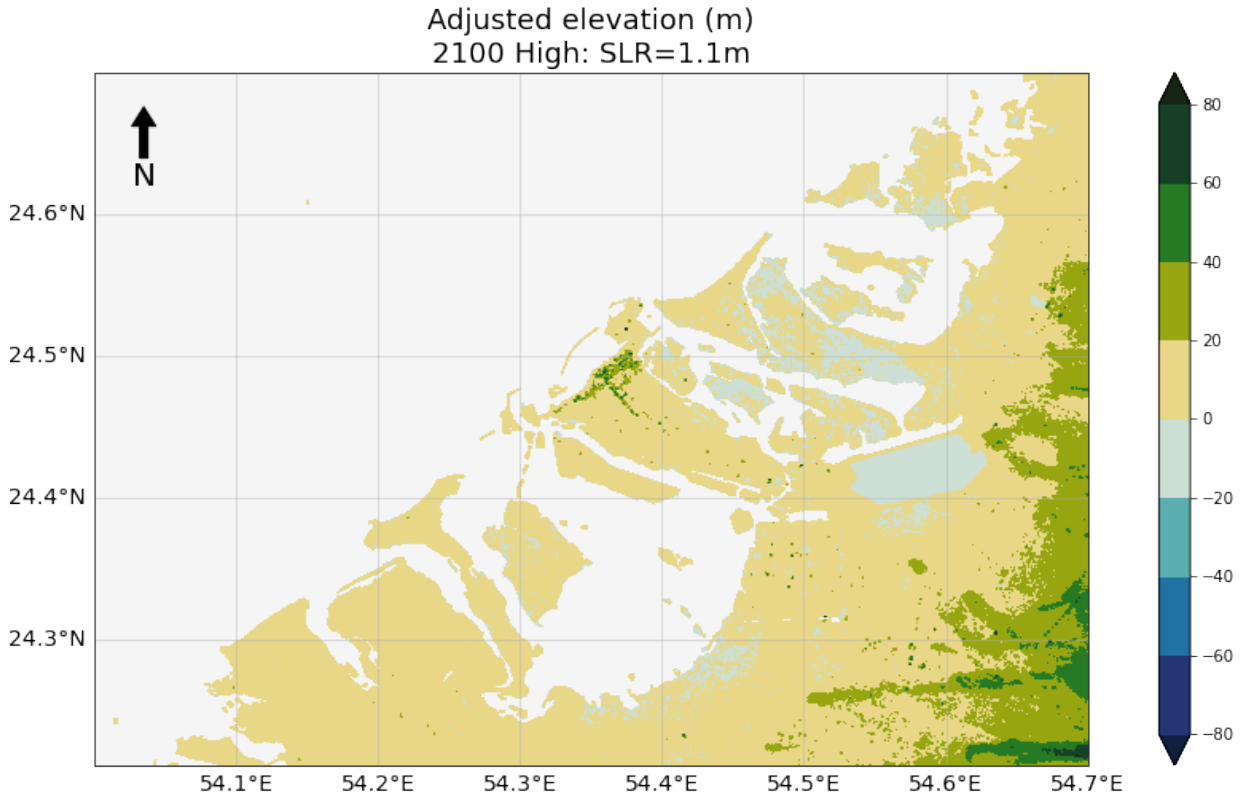

Figure 11: Adjusted elevation in the scenario of 1.1m sea level rise.

4.6 Land cover

The land cover dataset used in this project is stored in the form of a shapefile. We use GeoPandas' `read_file()` to read the shapefile into a geodataframe. The first 5 data points are shown in Figure 12. The dataset includes a total of 68,412 rows/data points. For each row, we are particularly interested in the habitat type and the polygon it encompasses, that is, information in the last two columns **Habitats** and **geometry**.

	OBJECTID	Id	HabitatTyp	HabitatT_1	HabitatSub	HabitatS_1	RuleID	Shape_Leng	Shape_Area	Habitats	geometry
0	1	1	1000	Intertidal Habitats	1010	Mudflats And Sand Exposed At Low Tide	14	7.524321e+03	2.115300e+06	Intertidal Habitats	POLYGON ((194075.975 2671979.549, 194105.053 2...
1	2	2	1000	Intertidal Habitats	1010	Mudflats And Sand Exposed At Low Tide	14	4.870493e+04	7.777595e+06	Intertidal Habitats	POLYGON ((202477.763 2674966.027, 203045.295 2...
2	3	3	1000	Intertidal Habitats	1010	Mudflats And Sand Exposed At Low Tide	14	2.721521e+03	1.367130e+05	Intertidal Habitats	POLYGON ((205824.217 2675669.812, 205805.696 2...
3	4	4	1000	Intertidal Habitats	1010	Mudflats And Sand Exposed At Low Tide	14	1.001023e+04	1.424318e+06	Intertidal Habitats	POLYGON ((199076.749 2676693.733, 199083.231 2...
4	5	5	1000	Intertidal Habitats	1010	Mudflats And Sand Exposed At Low Tide	14	6.023903e+04	1.222009e+07	Intertidal Habitats	POLYGON ((212836.986 2678391.436, 212806.558 2...

Figure 12: Sample land cover data points. Source: Jhon Mojica.

Similar to vertical land movement data, the land cover dataset covers a wider area than the base map that is the elevation data. This is visualized in Figure 13, where the elevation data points in blue are plotted on top of the land cover map.

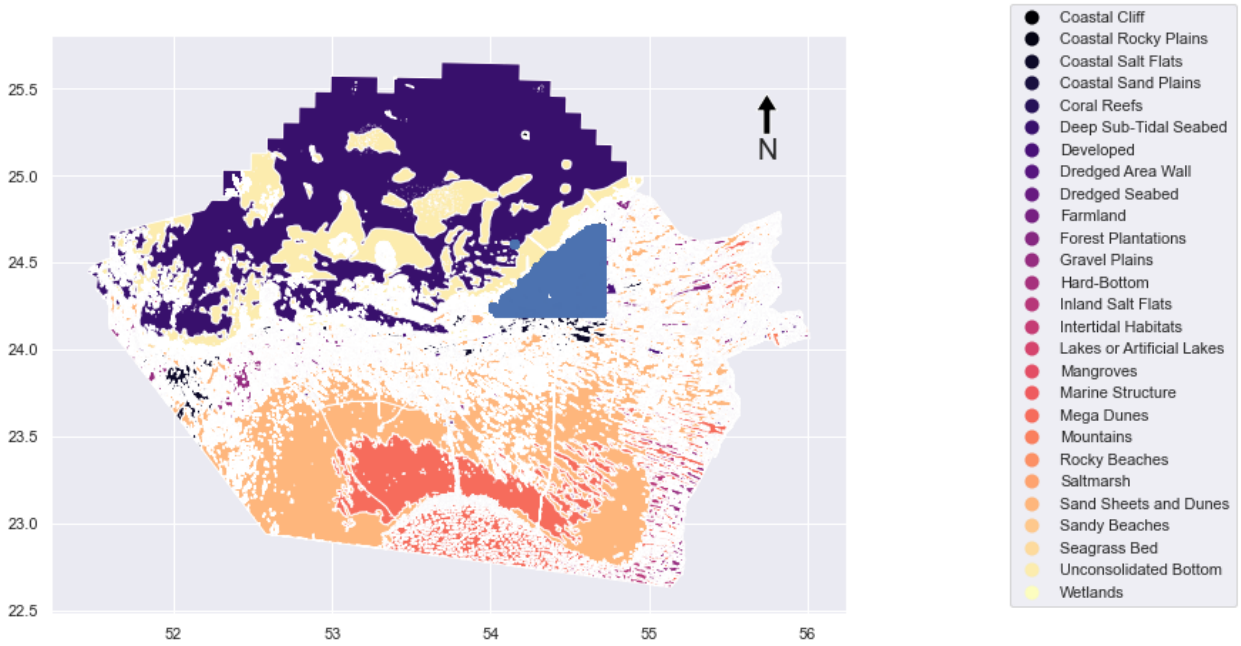

Figure 13: Comparison of land cover data and elevation data.

To ensure consistency between land cover data and elevation data, the next step is to intersect the land cover map with the elevation map so that we can have two maps of the same dimensions, and each data point in the base map can have a corresponding land cover type. The technique that we use for processing vertical land movement, however, can not be utilized for land cover. Land cover data is most effectively stored as a geodataframe, and each row in the dataframe corresponds to a particular polygon of a specific shape, not a single spatial point or grid cell. As such, it is not optimal to convert land cover data into a NumPy array for interpolation as we did with vertical land movement. Rather, we need to conduct this processing stage while treating the datasets as dataframes.

The first step is to load elevation data as dataframe. For this, we use GeoRasters' `from_file()` and `to_geopandas()`. A sample of the elevation dataframe is shown in Figure 14. This dataframe contains 1,789,661 rows, the same number of cells in the elevation grid with non-null values. Each row also has a corresponding geometry element, which is generated by GeoRasters to be a square.

	row	col	value	x	y	geometry
0	0	2347	1.0	54.651806	24.700139	POLYGON ((54.65181 24.70014, 54.65208 24.70014...
1	0	2348	0.0	54.652084	24.700139	POLYGON ((54.65208 24.70014, 54.65236 24.70014...
2	0	2349	1.0	54.652361	24.700139	POLYGON ((54.65236 24.70014, 54.65264 24.70014...
3	0	2350	1.0	54.652639	24.700139	POLYGON ((54.65264 24.70014, 54.65292 24.70014...
4	0	2351	2.0	54.652917	24.700139	POLYGON ((54.65292 24.70014, 54.65319 24.70014...
...	...	...	...	...	...	...
1789656	1757	2516	37.0	54.698750	24.212083	POLYGON ((54.69875 24.21208, 54.69903 24.21208...
1789657	1757	2517	36.0	54.699028	24.212083	POLYGON ((54.69903 24.21208, 54.69931 24.21208...
1789658	1757	2518	37.0	54.699306	24.212083	POLYGON ((54.69931 24.21208, 54.69958 24.21208...
1789659	1757	2519	38.0	54.699584	24.212083	POLYGON ((54.69958 24.21208, 54.69986 24.21208...
1789660	1757	2520	39.0	54.699861	24.212083	POLYGON ((54.69986 24.21208, 54.70014 24.21208...

Figure 14: Sample elevation data points.

Since we are dealing with two geodataframes containing of inconsistent geometries, we need to explore functions that allow us to find the overlapping components of the two maps and to retain all the attributes of those parts coming from both datasets. The two functions we experiment with are `sjoin()` and `overlay()` from GeoPandas.

According to GeoPandas' documentation, the `how` parameter of `sjoin` determines which rows from which dataframe are kept, ignored, or duplicated based on the spatial relationships among their geometries. The `overlay()` function creates new geometries

based on the spatial relationships among them, for example, to create new shapes where they overlap, or where they do not overlap. Even though these two functions do support our purpose in this step, we run into the problem of run time. Since both functions operate based on the differences and similarities between two datasets, their complexities also rise quickly. Because the functions have to compare every data row from one dataset against every data row from the other set, it can be estimated that their run time complexities are of $\mathcal{O}(m \times n)$. With our elevation dataset containing 1,789,661 points and land cover dataset containing 68,412 points, the number of necessary computations will be in the 100,000,000,000 range. On top of that, handling geodataframe further complicates the computations, adding on to the run time. Our tests show that each function takes over 6 hours to fully process the two datasets, returning the desired dataset of overlapping components. In order to reduce run time to a manageable level, we use a series of pre-processing steps applied on both datasets before running either `sjoin()` or `overlay()`. Our goal is to reduce the number of data points in either or both datasets to a selected few, such that either overlapping function will only have to process parts of the data without losing necessary information.

We seek to locate the area that the elevation map spans, and drop all rows in the land cover dataset that do not interest this area. First, by locating all corner coordinates of the elevation map, we can create a bounding box made up of a polygon object. We can then use GeoPandas' `GeoSeries.intersects()` function on the **geometry** column of the land cover dataframe. This function returns a series containing boolean values depending on whether a land cover row intersects with the elevation bounding box. Using this series, we keep only land cover rows that do intersect. At the end of this step, we retain 7057 rows in the land cover dataframe. Figure 15 visualizes these retained rows, and the elevation bounding box (in low opacity blue).

We further optimize the land cover dataset by next creating a polygon in the shape of the entirety of the elevation map. That is, a polygon more specific to the outline shape of the elevation map, rather than an overarching bounding box. Using the `cascaded_union()` function from the Shapely library, we are able to joint all separate geometries in the elevation dataframe into one single polygon for this step. After that, we use the same `GeoSeries.intersects()` function to find all rows from the land cover dataframe (having been reduced once) that intersect with the elevation polygon. After this step, the land cover dataset is further reduced to 4798 rows. The result is visualized in Figure 16, in which the elevation polygon is also plotted in low opacity red. With this, we have significantly reduced the land cover dataset from the original 68,412 rows while also ensuring that we retain all rows that overlap with the elevation map at some point.

Next, we use the `sjoin()` function on the elevation dataframe and the reduced land cover dataframe with the following set of parameters:

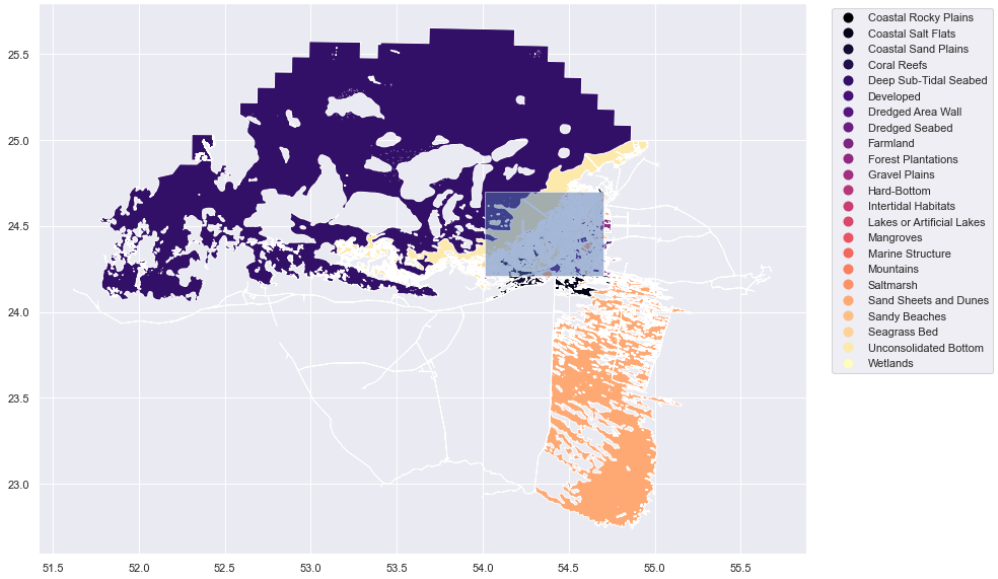

Figure 15: Land cover map after one reduction step using elevation bounding box.

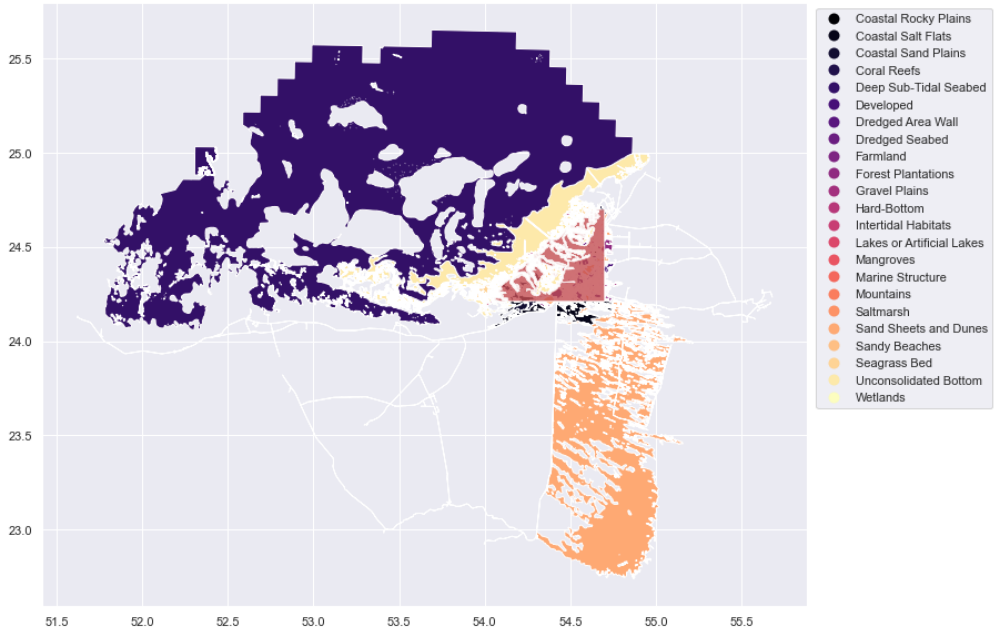

Figure 16: Land cover map after two reduction steps using elevation bounding box and elevation polygon.

```
final = gpd.sjoin(elevation_df, habitat_cut_cut, how="left", op="within")
```

Since we are interested in retaining the geometries of the elevation map and the land cover attribute, we specify the elevation dataframe first, then the land cover dataframe second, and set `how="left"`. According to GeoPandas' documentation, this will conduct a left outer join, in which "we keep all rows from the left and duplicate them if necessary to

represent multiple hits between the two dataframes," and we "retain attributes of the right if they intersect and lose right rows that don't intersect."

`op="within"` specifies the binary predicate we use to define the relationship that we want to examine among the geometries. The default value is `op="intersects"`, but we opt for `within` since this option speeds up the operation significantly, though with lower accuracy. With this settings, all locations at which a point from the elevation map coincides with the boundary of a polygon from the land cover map are considered ineligible according to the `sjoin()` function, and attributes from the land cover dataframe will not be retained for those points. Therefore, the returned dataframe `final` contains a total of 1,789,661 rows since all the shapes from the elevation dataframe are retained, but only 1,478,285 of them come with a specified land cover value. The remaining 311,376, which lie along boundaries of original land cover shapes, have null land cover values. We use a customized function to interpolate these missing rows with values from the nearest points.

With this series of data reduction steps and an interpolation step afterwards, we have significantly reduced the time complexity of processing land cover data. The entire process, including land cover dataframe reduction, map intersection, and missing values interpolation now take under 20 minutes altogether. Figure 17 visualizes the final land cover map ready for the next data processing and model building steps.

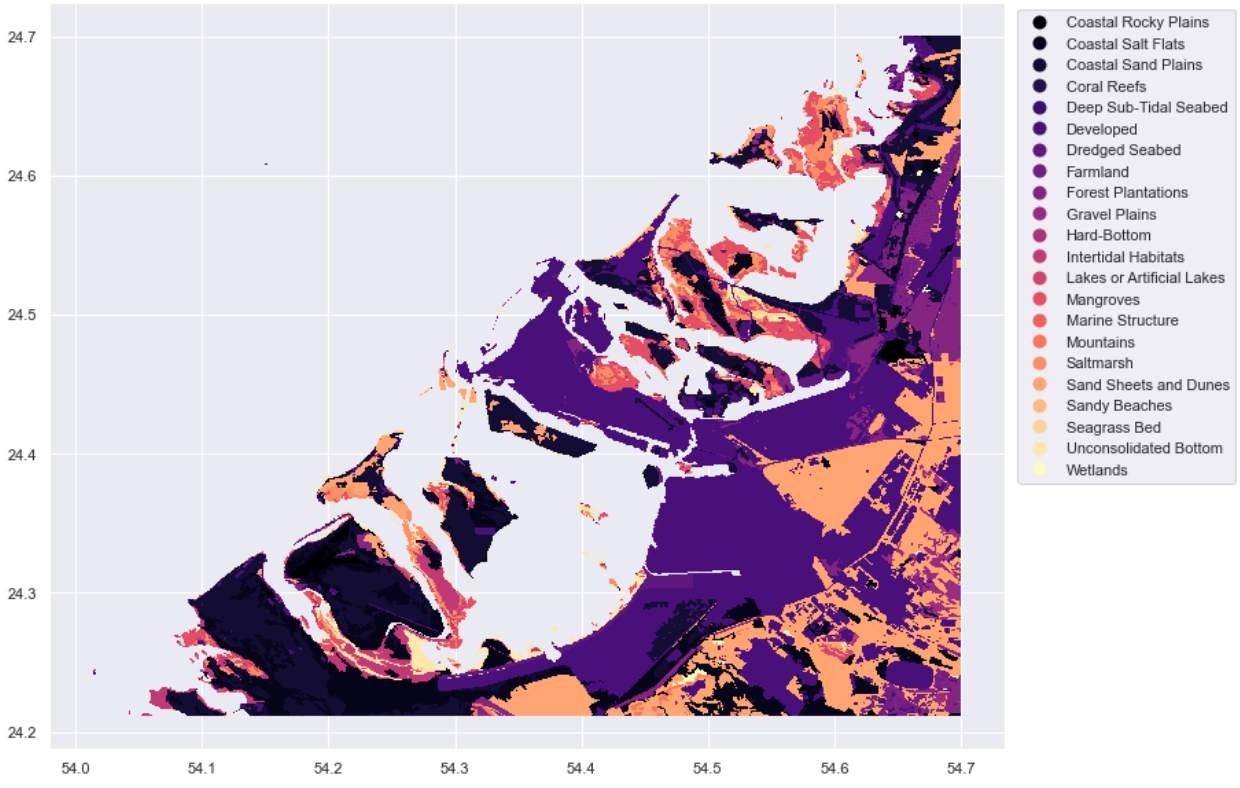

Figure 17: Land cover types map.

4.7 Coastal response

The coastal response variable is a child node of adjusted elevation and land cover type in our Bayesian Network, as established in section 3.2. The causal relationship between coastal response and the two parent nodes used in this project is based upon the study by Lentz et al. (2015) [4], specifically the likelihood estimates based on the particular combination of an adjusted elevation prediction and the corresponding land cover type. Table 2 describes the estimated coastal response probabilities for all such combinations as they are used in this project. The probabilities shown are dynamic response likelihood estimates, and since the coastal response variable takes up either the inundation state or the dynamic state, the probabilities for land inundation can be obtained by subtracting the corresponding dynamic response probabilities from 1.

Land cover type	Adjusted elevation ranges (meters)						
	<–12*	–12 to –1	–1 to 0	0 to 1	1 to 5	5 to 10	>10*
Subaqueous	1	0.9	0.7	0.5	0.1	0*	0
Rocky	0	0.05	0.1	0.5	0.9	1*	1
Marsh/Salt Flats	0.05	0.25	0.45	0.65	0.9	1*	1
Sandy	0.2	0.4	0.85	0.95	1	1	1
Forest	0	0.1	0.45	0.5	0.75	0.95*	1
Developed	0	0.05	0.25	0.5	0.75	0.95*	1

Table 2: Dynamic coastal response probability estimates for land cover type - adjusted elevation range combinations. Source: Lentz et al. 2015 [4] with modifications (*).

References for each of the land cover types are detailed in the study conducted by Lentz et al. 2015 [4]. For this project, we make further modifications and estimations for the probability table to be more suitable for our case study and software usage. Specifically, we first add two adjusted elevation ranges to include all areas and scenarios in our data that result in adjusted elevation measures of either below -12m or above 10m. The original coastal response estimates also do not include the data for elevation range 5-10m for areas of type Subaqueous, Rocky, Marsh/Salt Flats, Forest, and Developed. Missing values as such are not allowed in the later construction of the Bayesian Network with the specific Python library we use. Therefore, we decide to modify the coastal response estimates with our inferences for the missing values in the columns for < -12m, 5-10m, and > 10m adjusted elevation ranges. These inferences are based on the existing coastal response probabilities in other adjusted elevation ranges in the same land cover type. The final coastal response probability table encompasses relevant expert knowledge and inferences on the causal relationships among adjusted elevation predictions, land cover types, and

dynamic coastal response likelihood, and will be extensively utilized in building, training, and evaluating the Bayesian Network.

5 DATA DISCRETIZATION

In order to fit the Bayesian Network with our datasets, an additional step is necessary where we discretize all of the variables into predefined bins or intervals and encode them with digits.

5.1 Elevation

For elevation data, we define 7 intervals and divide the data accordingly. Like with coastal response likelihood estimates discussed in 4.7, we base our work on the study by Lentz et al. 2015 [4], with additions to ensure the model covers the entirety of our dataset. The elevation (in meters) groups, the specific intervals, and their numeric encodings are as followed:

- Interval 0: $[-89.0, -10.0)$ - all points below -10m (equivalent to interval $(-\infty, -10)$).
- Interval 1: $[-10.0, -1.0)$ - all points from -10m to below -1m.
- Interval 2: $[-1.0, 0.0)$ - all points from -1m to below 0m.
- Interval 3: $(-0.001, 1.0]$ - all points from 0m to 1m (equivalent to interval $[0, 1]$).
- Interval 4: $(1.0, 5.0]$ - all points from above 1m to 5m.
- Interval 5: $(5.0, 10.0]$ - all points from above 5m to 10m.
- Interval 6: $(10.0, 83.0]$ - all points above 10m (equivalent to interval $(10, +\infty)$).

We use the `cut()` function from pandas to divide all points into intervals with different ranges and endpoints inclusion. Since this function does not allow intervals closed on both sides (like $[0, 1]$), we opt to create a right-closed interval for interval 3, with modified left end value to make sure it includes values equal to 0. Figure 18 shows the data points distribution in all elevation intervals.

5.2 Vertical land movement

For vertical land movement, instead of dividing the data into predefined intervals like with elevation, we instead seek to divide the data into 3 classes in such a way that the classes have relatively the same number of data points, regardless of the actual intervals. For this, we use the function `qcut()` from pandas. For our specific dataset, the intervals, defined by the function's computations, are detailed as followed.

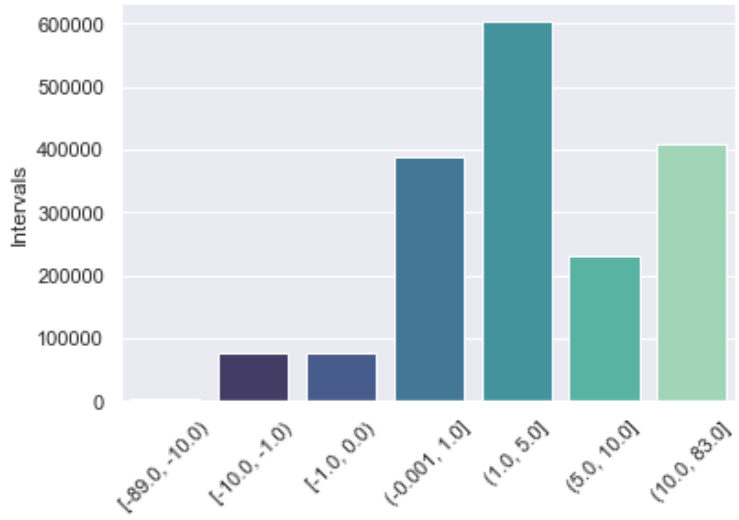

Figure 18: Distribution of discretized elevation data.

- Interval 0: $(-0.00322, 0.000517]$.
- Interval 1: $(0.000517, 0.00102]$.
- Interval 2: $(0.00102, 0.00288]$.

5.3 Adjusted elevation

For adjusted elevation, we use the same technique as we do with elevation, in which we define in advance the intervals, then use pandas' `cut()` function to divide the data accordingly. Overall, the intervals for adjusted elevation do not differ significantly from elevation intervals. The adjusted elevation (in meters) groups, the specific intervals for the first scenario (sea level rise projection determined to be 0.18m), and their numeric encodings are detailed as followed, and their data points distribution is visualized in Figure 19.

- Interval 0: $[-89.179, -12.0)$ - all points below -12m (equivalent to interval $(-\infty, -12)$).
- Interval 1: $[-12.0, -1.0)$ - all points from -12m to below -1m.
- Interval 2: $[-1.0, 0.0)$ - all points from -1m to below 0m.
- Interval 3: $(-0.001, 1.0]$ - all points from 0m to 1m (equivalent to interval $[0, 1]$).
- Interval 4: $(1.0, 5.0]$ - all points from above 1m to 5m.
- Interval 5: $(5.0, 10.0]$ - all points from above 5m to 10m.
- Interval 6: $(10.0, 82.82]$ - all points above 10m (equivalent to interval $(10, +\infty)$).

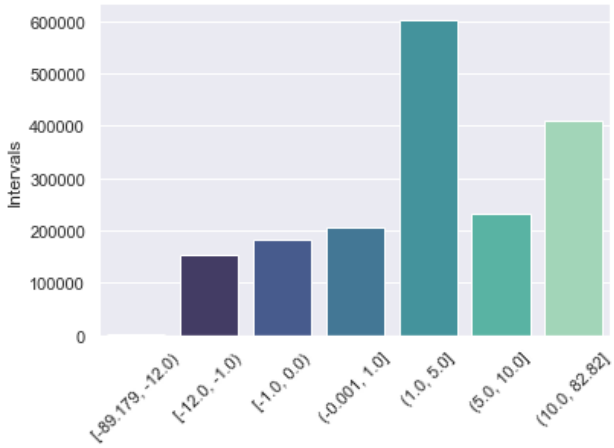

Figure 19: Distribution of discretized adjusted elevation for scenario 0.

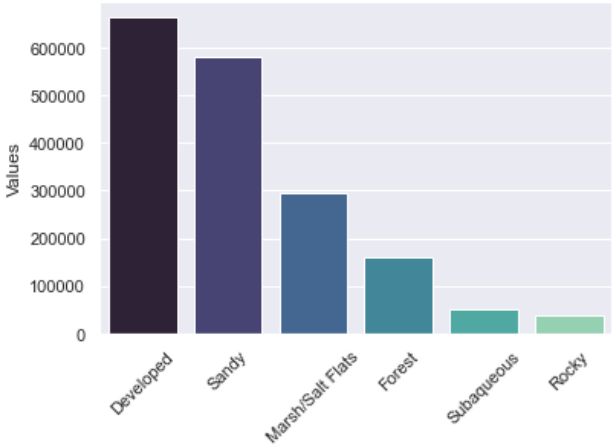

Figure 20: Distribution of discretized land cover type.

5.4 Land cover

The land cover type data, after being intersected with the elevation base map, is further clustered into six generalized groups, in which each group assumes a similar coastal response probability in all of its sub-groups (Lentz et al. 2015 [4]). The original land cover types as visualized in Figure 17 are grouped according to Table 3. The newly grouped land cover types are visualized in Figure 21, and their distribution in Figure 20.

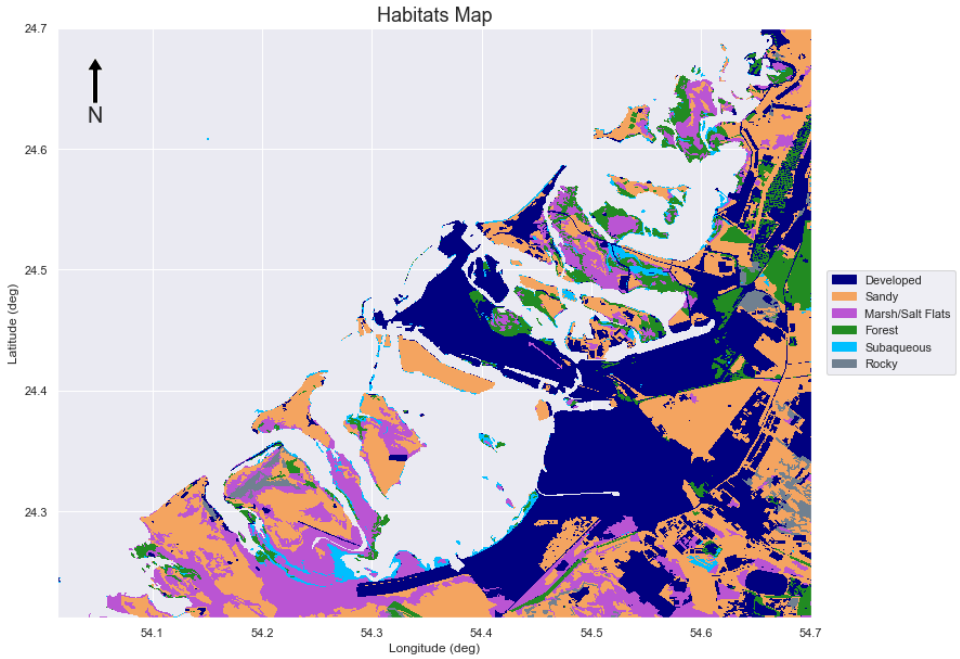

Figure 21: Generalized land cover groups map.

Land cover type	Group	Land cover type	Group
Mountains	Rocky	Hard-Bottom	Subaqueous
Mega Dunes	Sandy	Inland Salt Flats	Marsh/Salt Flats
Marine Structure	Developed	Intertidal Habitats	Marsh/Salt Flats
Coastal Cliff	Rocky	Lakes or Artificial Lakes	Subaqueous
Coastal Rocky Plains	Rocky	Rock Armouring/ Artificial Reef	Rocky
Coastal Salt Flats	Marsh/Salt Flats	Rocky Beaches	Rocky
Coastal Sand Plains	Sandy	Saltmarsh	Marsh/Salt Flats
Coral Reefs	Subaqueous	Sand Sheets and Dunes	Sandy
Deep Sub-Tidal Seabed	Subaqueous	Sandy Beaches	Sandy
Developed	Developed	Seagrass Bed	Subaqueous
Dredged Area Wall	Developed	Storm Beach Ridges	Rocky
Dredged Seabed	Developed	Unconsolidated Bottom	Subaqueous
Farmland	Developed	Wetlands	Marsh/Salt Flats
Forest Plantations	Forest	Mangroves	Forest
Gravel Plains	Rocky		

Table 3: Land cover types and corresponding land cover groups.

5.5 Coastal response

Since we do not have real data on coastal response, this step involves using the coastal response probability table (Table 2) to generate temporary labels for all the data points. In saying "temporary labels," we mean as in for a particular data point (a set of observed values for elevation, vertical land movement, and sea level rise projection), its coastal response probability is determined based on its adjusted elevation range and land cover type. In doing this, we are using the adjusted elevation range computed using the deterministic equation relating all three parent nodes of adjusted elevation. In other words, we

are assuming an uncertainty rate of zero for adjusted elevation, and use this value and land cover type to determine the data point's dynamic coastal response likelihood.

In retrieving temporary labels for coastal response, we retain two sets of labels. First is the set of coastal response probabilities, that is, the probability of inundation as well as the probability of dynamic response ($P(CR = CR_0)$ and $P(CR = CR_1)$, in which CR_0 denotes an inundation scenario, and CR_1 denotes a dynamic response, and $P(CR = CR_0) + P(CR = CR_1) = 1$). These labels are extracted through indexing the coastal response probability table. The second set of labels include binary coastal response classes, that is, whether the response is inundation (0) or dynamic (1). These labels are derived using coastal response probabilities, in which a $P(CR = CR_1) > 0.5$ means a dynamic response (1), and $P(CR = CR_1) \leq 0.5$ means inundation (0).

5.6 Training and testing data

With all datasets discretized and encoded in numeric form, we move on to merging them into one comprehensive dataset for model training and testing. For each sea level rise scenario (encoded from 0 to 6), the sea level rise projection (in meters) is used in the deterministic equation which produces the adjusted elevation dataset for that scenario. Consequently, temporary labels for coastal response are obtained. This way, in each scenario, a dataset including all variables (sea level rise projection, elevation, vertical land movement, adjusted elevation, land cover type, and coastal response) is produced, containing 1,789,661 rows/data points. We merge all 7 scenarios into one training dataset, containing 12,527,627 data points. A sample of this dataframe is shown in Figure 22.

	SLR	E	VLM	AE	LC	CR
0	0	3	2	3	5	1
1	0	3	2	2	5	0
2	0	3	2	3	5	1
3	0	3	2	3	5	1
4	0	4	2	4	5	1

Figure 22: A sample of the training set.

The testing dataset for the model is the same as the training dataset, sans the two columns of the two variables we want the model to predict: **AE** and **CR**, for adjusted elevation and coastal response.

6 BAYESIAN MODEL

For this project, my colleagues and I have developed two Bayesian models with different approaches to how we interpret and evaluate the coastal response variable. The first model, which treats temporary labels for coastal response as discrete classes for training and testing, will be further explored in my colleagues' reports for this project. In this paper, I will focus on the second model, which uses binary coastal response classes and the conditional probability distribution table of coastal response in building the model.

6.1 Model construction

We recall the architecture detailed in 3.2 that we are using for the Bayesian model:

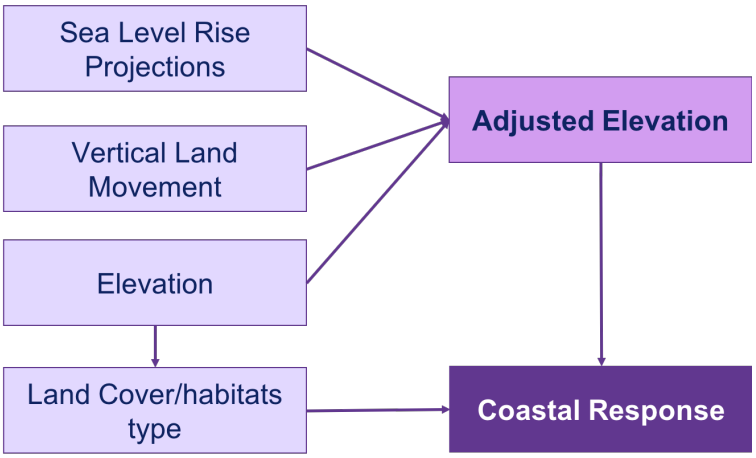

Figure 23: Structure of the Bayesian model.

Using the pgmpy library, we can build a Bayesian model with the structure describe in Figure 23 by providing directed edges representing the causal relationships among the variables.

```
from pgmpy.models import BayesianModel
model = BayesianModel([('SLR', 'AE'),
                       ('VLM', 'AE'),
                       ('E', 'AE'),
                       ('E', 'LC'),
                       ('LC', 'CR'),
                       ('AE', 'CR')])
```

6.2 Model training

We proceed to train the model on the `data` variable, which contains the training dataset we have prepared.

```
from pgmpy.estimators import BayesianEstimator
model.fit(data, estimator=BayesianEstimator, prior_type="BDeu")
```

In our experiments, training on 12,527,627 data points takes about 15 seconds.

After fitting the model with the training data, a conditional probability distribution able is generated for every variable/node in the network. These tables quantify the causal relationship between a node and its child node(s) and will later be used when making predictions for missing variables, given a set of values of the remaining variables.

The conditional probability distribution for coastal response is currently based solely on binary classes, 0 and 1, as labels. This method bypasses the important conditional relationships between coastal response and its parent nodes, adjusted elevation and land cover. To fully capture these relationships, coming from expert knowledge and inference as previously discussed in 4.7, we overwrite the existing coastal response conditional probability distribution table with our own input using the coastal response likelihood estimates table structured in a format acceptable to the pgmpy library. Figure 24 shows parts of the original conditional probability distribution table for coastal response computed by the model based on training data, and Figure 25 shows parts of the updated one.

AE	AE(0)	AE(0)	AE(0)	AE(0)
LC	LC(0)	LC(1)	LC(2)	LC(3)
CR(0)	0.001561524047470331	0.9719101123595506	0.9993807282635621	0.9998837695848248
CR(1)	0.9984384759525297	0.02808988764044944	0.0006192717364379489	0.0001162304151750429 8

Figure 24: Conditional probability distribution table for coastal response - Model computations.

AE	AE(0)	AE(0)	AE(0)	AE(0)	AE(0)	AE(0)	AE(1)	AE(1)	AE(1)	AE(1)	AE(1)	AE(1)	AE(2)	AE(2)	AE(2)
LC	LC(0)	LC(1)	LC(2)	LC(3)	LC(4)	LC(5)	LC(0)	LC(1)	LC(2)	LC(3)	LC(4)	LC(5)	LC(0)	LC(1)	LC(2)
CR(0)	0.0	1.0	0.95	0.8	1.0	1.0	0.1	0.95	0.75	0.6	0.9	0.95	0.3	0.9	0.55
CR(1)	1.0	0.0	0.05	0.2	0.0	0.0	0.9	0.05	0.25	0.4	0.1	0.05	0.7	0.1	0.45

Figure 25: Conditional probability distribution table for coastal response - Manual input.

6.3 Model evaluation

Now fitted, we can use the model to make predictions on two variables: adjusted elevation and coastal response.

6.3.1 Scenario testing

The first testing approach we take is to test the model on each scenario to compare the results across all scenarios. We use the function `model.predict()` from the pgmpy library on the same dataframe as the training data minus the two columns of variables we want to predict (adjusted elevation and coastal response). This function returns a dataframe with two columns, **AE** and **CR**, containing the class predictions for adjusted elevation and coastal response for all of the data points in the test set.

For adjusted elevation, we predict the class that the data is most likely to fall into. Since there is a finite number of such classes (7 intervals), we use accuracy score, the number of true predictions over the total number of predictions made, to evaluate model predictions of adjusted elevation. Table 4 summarizes the accuracy scores obtained when we test the model on all 7 sea level rise scenarios.

Scenario	AE accuracy (0-1)
0 (SLR=0.18m)	0.9063090719415576
1 (SLR=0.43m)	0.9063090719415576
2 (SLR=0.29m)	0.9063090719415576
3 (SLR=0.93m)	0.9063090719415576
4 (SLR=0.31m)	0.9063090719415576
5 (SLR=1.10m)	0.7416460435803205
6 (SLR=2.50m)	0.7154366106206707

Table 4: Accuracy scores for adjusted elevation in 7 scenarios

Adjusted elevation accuracy remains relatively the same for the first 5 scenarios, but seems to drop quite significantly as soon as we reach the 1.1m sea level rise threshold. The last two scenarios correspond to High scenario in time window 2100, and the hypothetical Extreme scenario. We see in later inspection of coastal response predictions how the uncertainties in adjusted elevation get propagated through the network, showing in the changes in errors as we move to more extreme scenarios.

For coastal response, pgmpy's `model.predict()` function returns a binary class prediction of whether the area will inundate (0) or respond dynamically (1), but this is not what we focus on. Instead of the state that the coastal response variable is most likely to take, we are more interested in the probabilities of all states of the variable, which the model uses to make a discrete class prediction. For this purpose, we use pgmpy's `model.predict_probabilty()` function, which returns the probabilities for all states of all missing variables. In the case of our models, for coastal response, `predict_probability()`

returns two probabilities CR_0 and CR_1, corresponding to the likelihood of predicting 0 (inundation) and 1 (dynamic response). Since the coastal response variable takes up either one of these values, $P(CR = CR_0) + P(CR = CR_1) = 1$. We are most interested in $P(CR = CR_1)$, which we interpret as the dynamic coastal response likelihood, and a key research point of this project.

The `predict_probability()` function, however, raises another run time complexity issue. Our tests show that predicting probabilities for one scenario takes over 9 hours. To counter this, we notice that there usually is a high level of repetition in the dataset: data points with similiar combinations of elevation, vertical land movement, and land cover classes. Meanwhile, the source code for `predict_probability()` suggests that the function conducts inference for every single data point without consideration of their repetition. From this observation, we think that using a cache can help to speeds up processing time. We implement a wrapper function for `predict_probability()`, in which we use a Python dictionary to store inferred probabilities for coastal response. Each element of this dictionary is a **(key, value)** pair, in which the **key** is a tuple made up of the sea level rise class, elevation class, vertical land movement class, and land cover class, and the **value** is $P(CR = CR_1)$ for that particular set of observations. For each new data point, we construct the **key** tuple and check whether it is present in the dictionary (time complexity $\mathcal{O}(1)$). If not, meaning the model has not made any prediction for that particular combination of observations, then we run `predict_probability()` for that point. Otherwise, we search the dictionary using the **keys** combination to get the dynamic response probability (time complexity $\mathcal{O}(1)$). So `predict_probability()`, which contributes most to the high run time complexity of the function, is only applied on a handful of points. Our modified predict function, as shown below, takes around 90 seconds to make probability predictions for one scenario, according to our tests.

```
def predict_prob(model, test):
    cache_dict = {}
    prob_list = []
    for _, data_point in test.iterrows():
        keys = (data_point.SLR, data_point.E, data_point.VLM, data_point.LC)
        if keys not in cache_dict:
            pred_df = model.predict_probability(pd.DataFrame([data_point]))
            pred_CR1 = pred_df.iloc[0].CR_1
            cache_dict[keys] = pred_CR1
        prob_list.append(cache_dict[keys])
    return pd.DataFrame(prob_list, columns=['CR_1'])
```

Following are coastal response maps of the temporary labels, of model predictions, and model errors, from 3 selected scenarios.

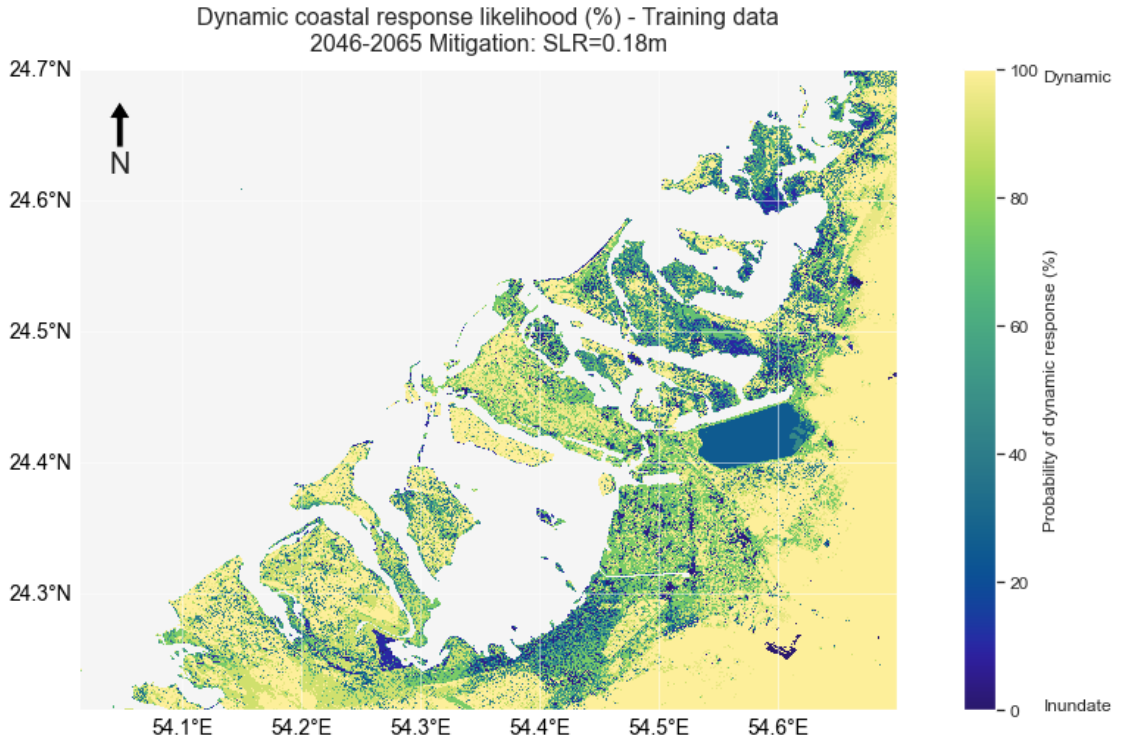

Figure 26: Dynamic coastal response likelihood map for scenario 0 - Labels.

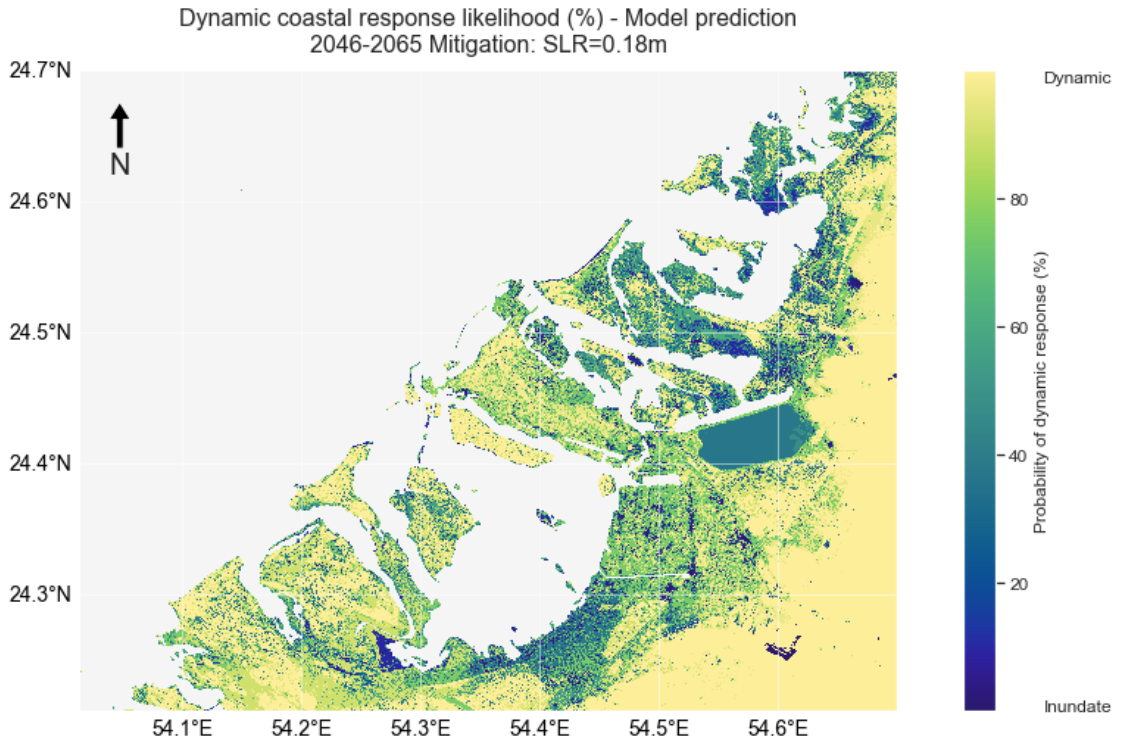

Figure 27: Dynamic coastal response likelihood map for scenario 0 - Predictions.

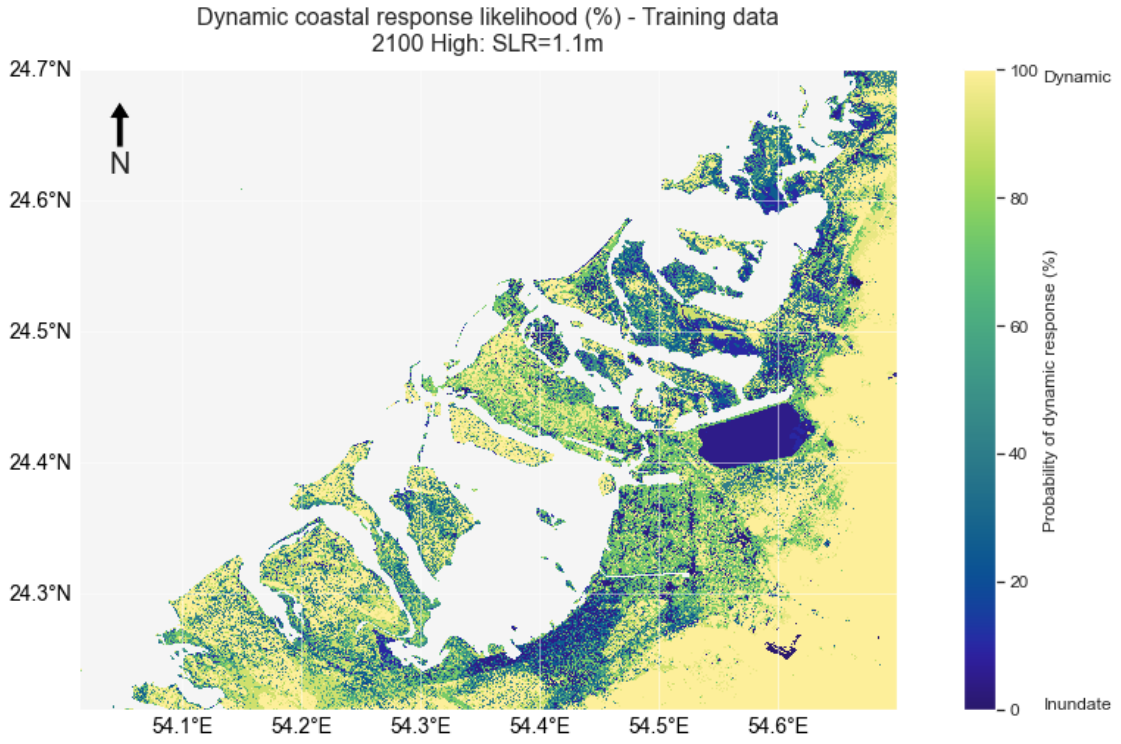

Figure 28: Dynamic coastal response likelihood map for scenario 5 - Labels.

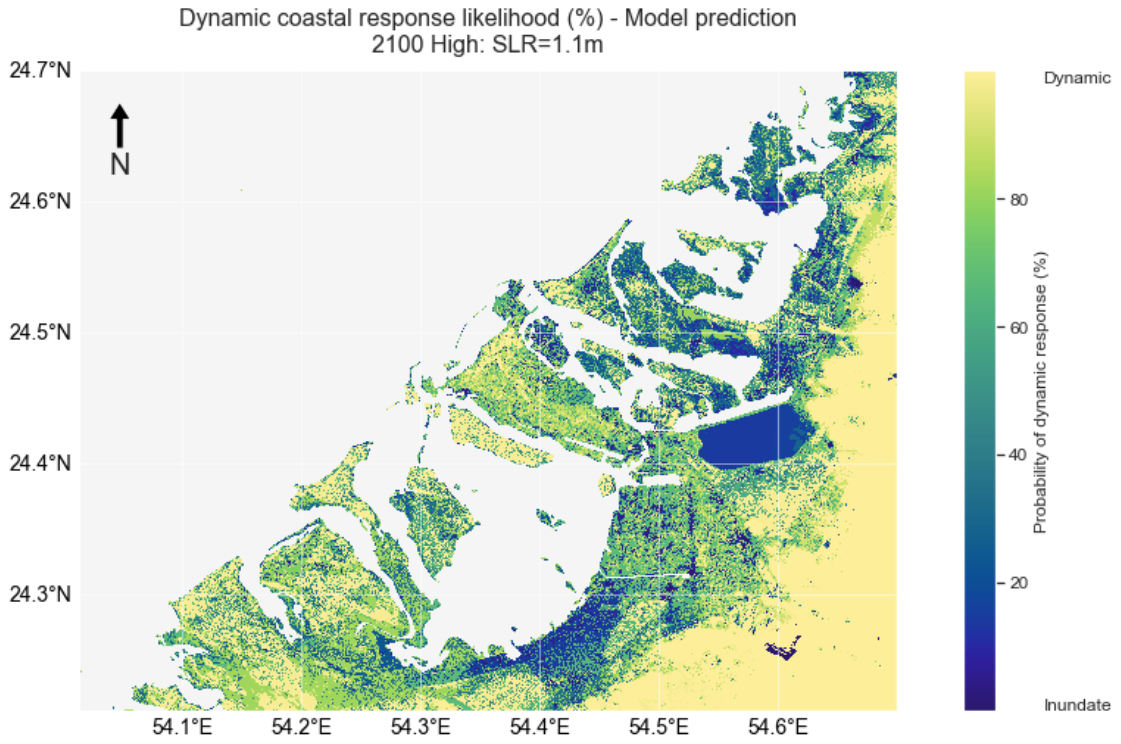

Figure 29: Dynamic coastal response likelihood map for scenario 5 - Predictions.

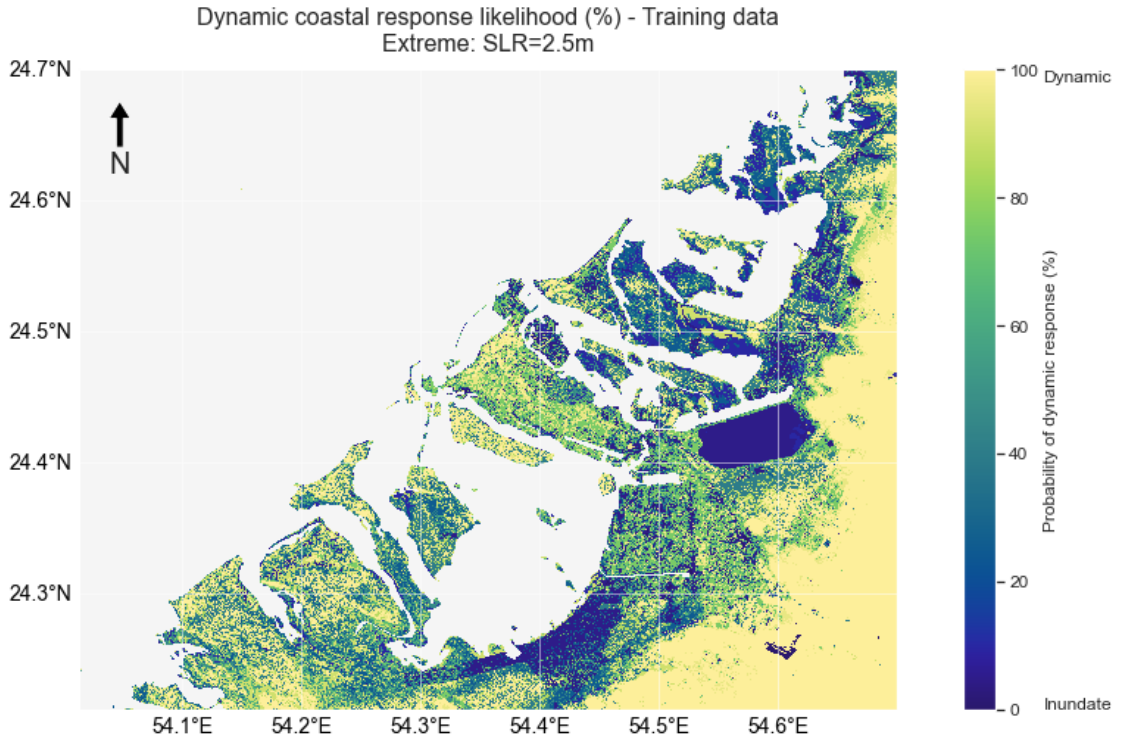

Figure 30: Dynamic coastal response likelihood map for scenario 6 - Labels.

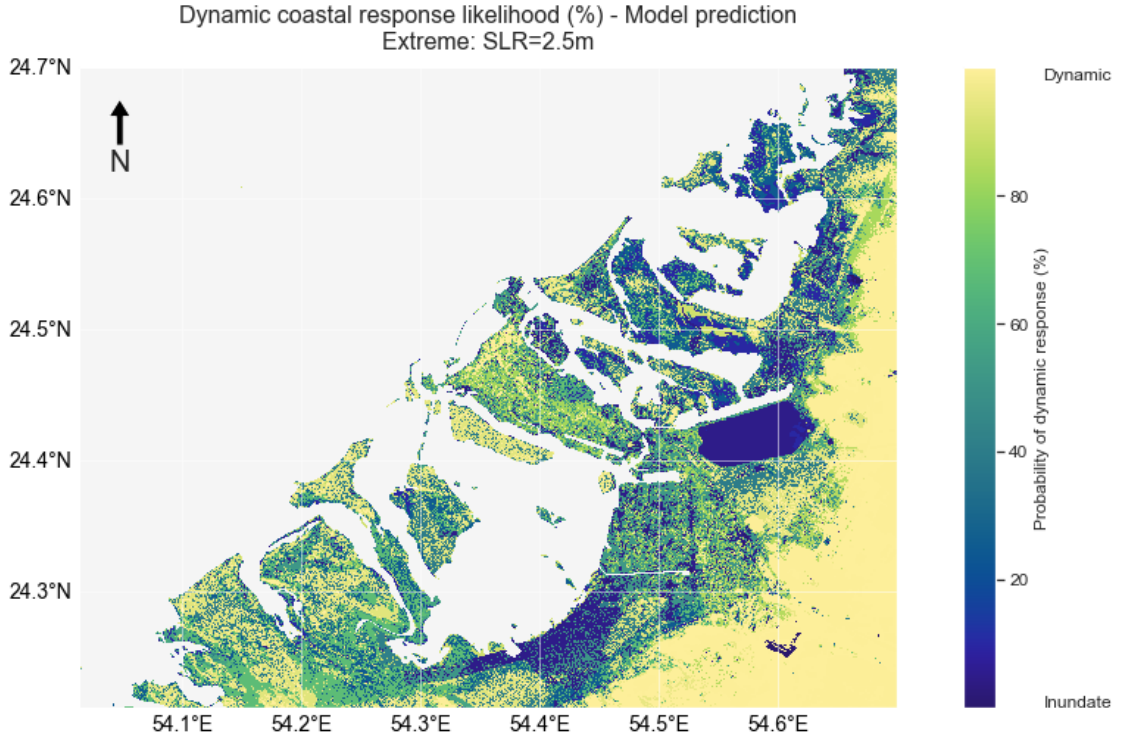

Figure 31: Dynamic coastal response likelihood map for scenario 6 - Predictions.

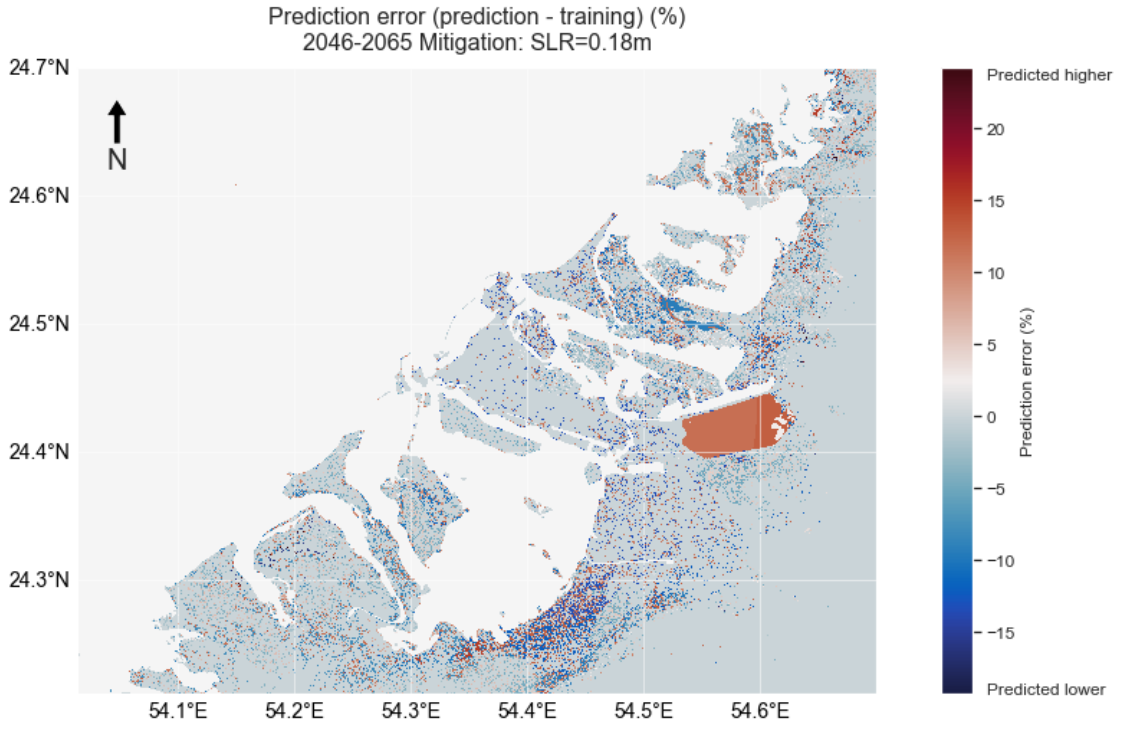

Figure 32: Dynamic coastal response likelihood map for scenario 0 - Model errors.

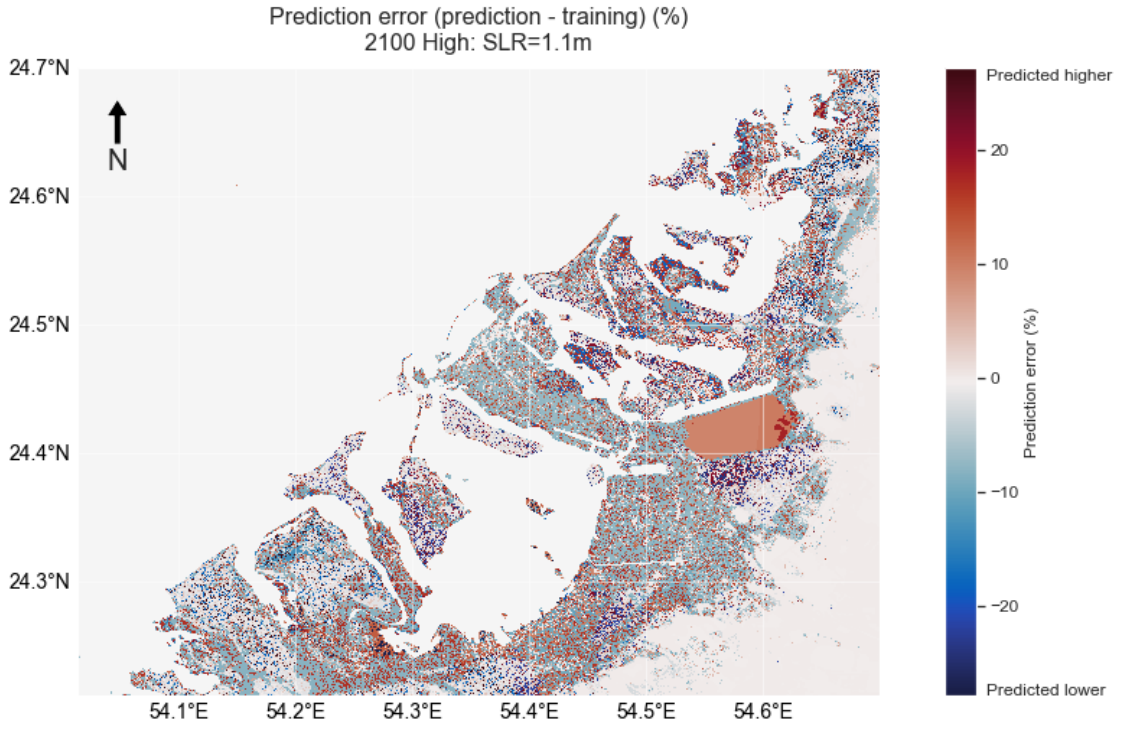

Figure 33: Dynamic coastal response likelihood map for scenario 5 - Model errors.

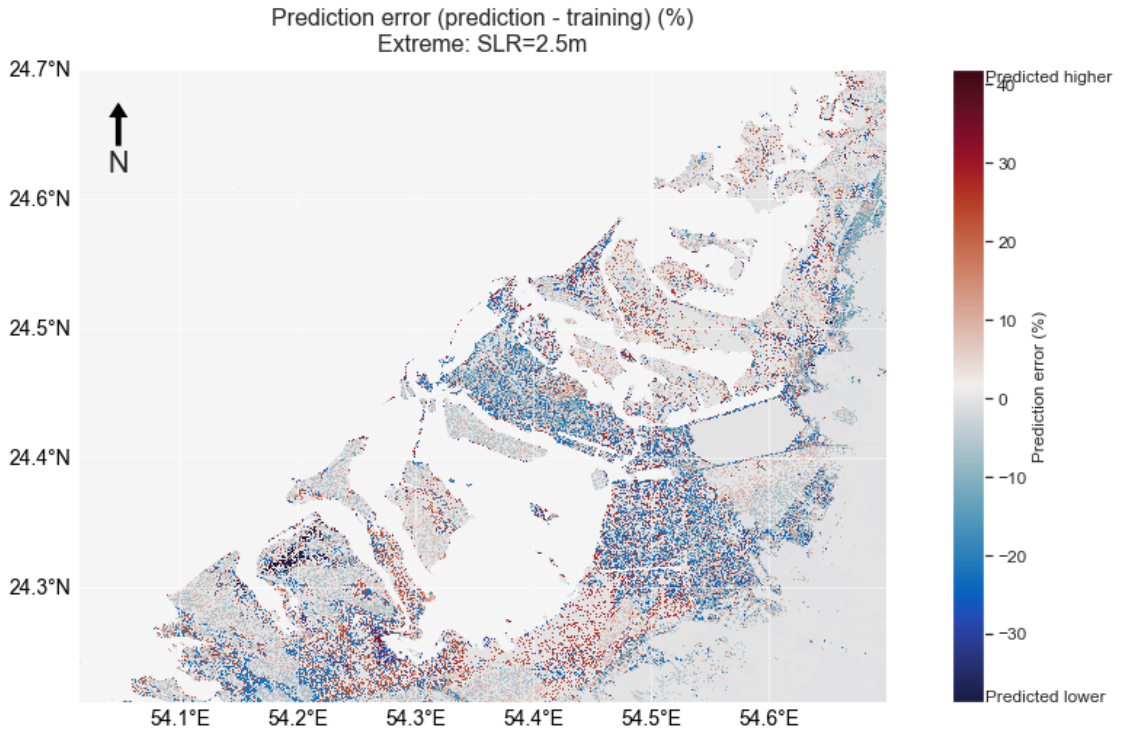

Figure 34: Dynamic coastal response likelihood map for scenario 6 - Model errors.

Considering the first prediction errors map, we observe more areas in pale white color (where predictions are close to labels) than areas in red or blue (where predictions are off by a level of errors). In the later two scenarios, however, we notice that there are more red and blue patches. This shows consistency with our earlier discussion on adjusted elevation predictions, because we see that accuracy for adjusted elevation drops as we move to more extreme scenarios. Since coastal response is a child node of adjusted elevation in our network, the error in adjusted elevation predictions gets propagated in the network, which leads to less accurate coastal response predictions as observed in these three scenarios.

Since we focus on the predicted probabilities for coastal response, and the outputs are in continuous form rather than discrete classes, we cannot use accuracy score to evaluate coastal response predictions. Rather, we choose to focus on the absolute error between model predictions and (temporary) labels. Table 5 summarizes the means and standard deviations of absolute errors observed in each of the 7 scenarios. Consistent with what we have discussed, the mean absolute error increases as we move to more extreme scenarios.

6.3.2 10-fold cross validation

The second testing approach we take is 10-fold cross validation, in which the entire dataset of 7 scenarios is randomly partitioned into 10 non-overlapping parts. In which fold, 9 of these parts are used to train a model, and the remaining part is used for testing that model.

Scenario	Means	Standard deviations
0	0.018453908176439764	0.03934033728310384
1	0.018453908176439764	0.03934033728310384
2	0.018453908176439764	0.03934033728310384
3	0.018453908176439764	0.03934033728310384
4	0.018453908176439764	0.03934033728310384
5	0.06610062305553042	0.06917224142318723
6	0.05514874573036009	0.08523793591422163

Table 5: Means and standard deviations of absolute errors in all scenarios.

The accuracy scores for adjusted elevation as well as means and standard deviations of absolute errors for coastal response are summarized in Table 6. Since the data is randomly partitioned, we observe that the accuracy scores and errors are more uniform across all folds. Overall, we get an accuracy of about 0.85 for adjusted elevation, and mean absolute error of about 0.03, standard deviation 0.056 for coastal response.

Fold	AE accuracy	CR abs errors means	CR abs errors standard deviations
0	0.855946	0.030475	0.056555
1	0.855274	0.030508	0.056552
2	0.855758	0.030525	0.056558
3	0.855717	0.030577	0.056655
4	0.855308	0.030467	0.056475
5	0.855369	0.030517	0.056568
6	0.855743	0.030482	0.056547
7	0.855161	0.030597	0.056689
8	0.855318	0.030500	0.056536
9	0.855589	0.030381	0.056417

Table 6: Accuracy scores and absolute errors from 10-fold cross validation.

7 FUTURE DEVELOPMENT

Our work on the project has been highly focused on our case study of the Abu Dhabi coastline, and many of our programming implementations and modifications are specific to the datasets we work with. Therefore, more test cases are necessary to further test the robustness of the code and the model, and to improve the quality and reusability of the Python scripts. Once the scripts have been refined, the model can be used on another location and/or geographic area with new datasets.

In the next stages of the project, a graphic user interface will be implemented and integrated with the Python scripts, allowing users to provide their own datasets and to tune various model parameters. It might potentially be a good choice to have a certain degree of data quality required from the users, for instance, a minimum number of original data points for vertical land movement. This can help to produce more accurate predictions, and cut down on run time of the program.

8 ACKNOWLEDGEMENT

I would like to thank Dr. Daiane G. Faller for her constant support and mentorship throughout the project, and for creating a very welcome and encouraging working environment where I have been able to significantly improve my skills in data analytics and problem solving. I would also like to thank Dr. Clare Eayrs and Nicola Freissmuth for welcoming us and coordinating our participation in the project, Dr. David Holland for the opportunity to work on this project, and Farhana Goha from New York University Abu Dhabi Office of Undergraduate Research for connecting me with the Center for Sea Level Rise. Lastly, I would like to thank Farah Ayyad and Sashank Silwal, my colleagues on this project, for their hard work and support for the past two months.

References

[1] E. Lentz, E. Thieler, N. Plant, S. Stippa, R. Horton, and D. Gesch, "Evaluation of dynamic coastal response to sea-level rise modifies inundation likelihood," *Nature Climate Change*, vol. 6, 03 2016.

[2] M. Schumacher, M. A. King, J. Rougier, Z. Sha, S. A. Khan, and J. L. Bamber, "A new global GPS data set for testing and improving modelled GIA uplift rates," *Geophysical Journal International*, vol. 214, pp. 2164–2176, 07 2018.

[3] M. Irani, A. Massah Bavani, A. Bohluly, and H. Alizadeh Katak Lahijani, "Sea level rise in persian gulf and oman sea due to climate change in the future periods," *Physical Geography Research Quarterly*, vol. 49, no. 4, pp. 603–614, 2017.

[4] E. Lentz, S. Stippa, E. Thieler, N. Plant, D. Gesch, and R. Horton, "Evaluating coastal landscape response to sea-level rise in the northeastern united states—approach and methods," 01 2015.