University of Stavanger

SIMEN ASHEIM ASKELAND
SUPERVISOR: ATLE ØGLEND

Risk-Based Decisions for Water Resource Management in Drammensvassdraget

Master thesis 2024
Industrial Economics
Faculty of Science and Technology

Preface

I became interested in hydropower during the Hans flood in Norway in 2023. The idea that empty reservoirs could have prevented this event stuck with me and guided my choice for a master's thesis. I discovered an abundance of data on water levels, reservoir volumes, and water flow in Norway's reservoirs, making this topic not only fascinating but also rich in open data. My prior interest in decision-making, decision support, and decision analysis at UiS further solidified my choice. After substantial preliminary work, I decided on this project.

The journey proved more challenging than anticipated, a thought likely shared by many who undertake a master's thesis. I aimed to keep things simple, believing that the foundation for assessing risk doesn't necessarily require the latest in computing power and statistical analysis. Consequently, I developed a decision-support formula to serve as a preliminary risk assessment tool based on given water levels during specific seasons. The principles of reservoir management have remained largely unchanged over the past century, despite the absence of modern computing power back then. This endeavor has highlighted the complexity of reservoir management and the multitude of factors that influence it. My attempt to simplify the process has often been incredibly challenging, revealing the necessity for complex solutions involving substantial data processing and simulation.

I would like to extend my gratitude to Atle Øglend from UiS for his guidance and to the Norwegian Water Resources and Energy Directorate (NVE) for their help.

Abstract

This thesis develops a decision-support formula for water resource management in the Drammensvassdraget region, aimed at balancing electricity generation and flood risk mitigation. By leveraging historical data and statistical analyses, the formula quantifies flood and energy shortage risks without relying on predictive forecasting. Sensitivity analyses reveal that flood risk is highly sensitive to seasonal and density adjustments, while energy shortage risk is moderately sensitive, especially to seasonal factors.

Key findings indicate the formula's practical utility in real-world scenarios, particularly in helping operators make informed water resource management decisions. By providing a quantitative basis for balancing electricity generation and flood risk.

The research underscores the formula's strengths, including its robust statistical foundation and practical applicability. However, limitations such as reliance on historical data and the exclusion of immediate weather forecasts are acknowledged. The research emphasizes the need for expert judgment in interpreting the risk values produced by the formula, particularly under extreme conditions.

Overall, this thesis is yet another tool to the field of water resource management, offering a comprehensive decision-support tool that integrates historical data, regulatory constraints, and seasonal variations.

Table of Contents

Abbreviations

LRW: Lowest Regulated Waterlevel

HRW: Highest Regulated Waterlevel

NVE: Norwegian Water Resources and Energy Directorate

KDE: Kernel Density Estimation

PDF: Probability Density Function

SVI: Seasonal Variability Index

C: Current Reservoir Capacity Adjustment

R: Regulatory Constraints Adjustment

S: Seasonal Trends Adjustment

ESR: Energy Shortage Risk

FR: Flood Risk

OWL: Observed Waterlevel

MF: Mean Flood

PT: Pre-Threshold

LT: Lower Threshold

UT: Upper Threshold

D: Density Adjustment

m.a.s.l.: meters above sea level

α: Scaling Factor for Energy Capacity Adjustment

β: Scaling Factor for Flood Capacity Adjustment

List of tables

List of figures

List of equations

1. Introduction

Hydropower has been a pillar of renewable energy, its legacy spanning centuries and continuously evolving with technological advancements. In Norway, hydropower has played a pivotal role since the late 19th century, shaping the country´s industrial and economic landscape. The nation's abundant water resources have made Norway a global leader in hydropower development, contributing significantly to the energy security and sustainability (International Hydropower Association, 2023). Globally, the history of hydropower is rich with milestones that highlight its transformative impact. Early developments, such as the invention of the Francis and Kaplan turbines, paved the way for large-scale projects like the Hoover Dam and the Three Gorges Dam, underscoring hydropower's capacity to meet substantial energy demands while fostering economic growth (Hydropower, 2024).

Norwegian hydropower historical evolution, from its starting stages in the late 1800s to its current sophisticated state, mirrors global advancements in the field. Projects like Norway's initial hydroelectric plants set a precedent for future developments, showcasing how technological innovation and natural resource management can work together to create robust energy systems (Regjeringen.no, 2016).

The Drammensvassdraget region, encompassing the interconnected lakes of Tyrifjorden, Randsfjorden, and Sperillen, is a critical area where the balance between water management and energy production is paramount. This balance was dramatically highlighted during "Ekstremværet Hans" in August 2023, an extreme weather event that brought record-breaking rainfall and severe flooding to the region. The storm caused extensive damage, leading to thousands of evacuations and significant disruptions to infrastructure and daily life (Ekstremværet Hans, 2024)

The thesis, “Risk-Based Decisions for Water Resource Management in Drammensvassdraget”, aims to develop a framework to assist in navigating these complex challenges. By integrating daily hydrology data from the Norwegian Water Resource and Energy Directorate, the framework will enable decision-makers to evaluate trade-offs between maximizing electricity generation and minimizing flood risks effectively. The research will develop into a framework displayed as a formula or a decision model. In developing this decision framework, research draws on the lessons from the past and recent climatic events

like “Ekstremværet Hans”. It will incorporate extensive statistical analysis and various techniques to provide a robust framework for decision-making.

Ultimately, this thesis seeks to pioneer a path forward, a starting point for a computational and AI driven management practice. It aims to be a starting point for a machine learning and data-driven approach to the challenges of energy security and environmental sustainability, ensuring that regions like Drammensvassdraget can thrive amidst the challenges posed by climate change and evolving resource demands. Through this novel approach, the research aspires to be a possibility study for the future of water management.

Importance of risk management in Water and Energy Management

Effective water and energy management is a cornerstone of sustainable development, especially in regions heavily reliant on hydropower like Norway. Decision models are indispensable tools in this context, providing a robust framework for optimizing resource use, enhancing sustainability, and mitigating risks associated with extreme weather events. These models empower policymakers and resource managers to make informed decisions based on comprehensive data analysis and predictive simulations.

In Norway, the history and evolution of hydropower underscore the critical role of decision models. The Norwegian Water Resources and Energy Directorate (NVE) uses sophisticated decision models to manage the country's extensive hydropower resources. These models integrate hydrological, climatological, and operational data to predict optimal water release schedules, ensuring that energy production is maximized during periods of high demand without compromising flood protection measures (Vassdragsregulanters ansvar og muligheter, 2023).

The practical application of decision models in Norwegian hydropower management provides a compelling case study. During the spring, when the risk of flooding increases due to snowmelt, decision models predict the timing and volume of snowmelt and coordinate the release of water from reservoirs to prevent downstream flooding. These models help maintain a delicate balance, ensuring that reservoirs do not overflow while preserving enough water for energy production (NVE - Vårflom, 2020).

In conclusion, decision models are essential for modern water and energy management. They provide the analytical foundation necessary for optimizing resource use, enhancing sustainability, and mitigating the risks associated with extreme weather events. The integration of these models into Norway's hydropower management exemplifies their critical role in ensuring the safe, efficient, and sustainable utilization of natural resources.

Research Objective and Scope

The objective of this thesis is to develop a novel decision-support framework for the management of water resources in the Drammensvassdraget, particularly focusing on Tyrifjorden, Randsfjord, and Sperillen. The research will be based on quantitative measures generated from historical statistics, excluding the use of weather forecasts and potential snow melting predictions. This exclusion means the model will not consider immediate weather warnings.

The goal is not to provide definitive decisions but to offer decision-support, recognizing that expert judgment, large computational models and qualitative assessments will always play a role. The model aims to assist decision-makers before the final decision stage, avoiding predictions. It can be viewed as a tool that aggregates knowledge and quantifies it, providing a procedure that converts history and statistics into numerical data.
By focusing on historical data, the research aims to provide a reliable framework for evaluating these trade-offs, ultimately aiding the operators in controlling the outflow of the lakes.

Introduction to Drammensvassdraget Region

Drammensvassdraget, one of Norway's most significant river systems, it encompasses a rainfall area of approximately 17,000 square kilometers, making it the country's third-largest watershed. Originating in the highlands and flowing through diverse landscapes, it integrates several major tributaries and lakes, including Tyrifjorden, Randsfjorden, and Sperillen, before emptying into Drammensfjorden (Thorsnæs, 2023).

The river system is renowned for its hydropower potential, with numerous dams and reservoirs harnessing the energy of water to produce a substantial portion of Norway's

electricity. Drammensvassdragets regulation capacity is significant, reflecting its crucial role in both energy production and flood management. Hydropower plants along the river, such as those at Tyrifjorden and Randsfjorden, are integral to the region's energy infrastructure, providing a reliable and renewable energy source while also contributing to flood control efforts (Drammensvassdraget, 2024).

Flood management in Drammensvassdraget is a vital aspect of its regulation, especially given the historical occurrences of severe flooding. Notable flood events, like those in 1927, 1967, and more recently, have demonstrated the importance of proactive and strategic water management. The river's regulation involves careful monitoring and control of water levels in its reservoirs to mitigate the risk of downstream flooding, particularly in densely populated areas. These measures are essential for protecting both human lives and property from the devastating impacts of floods.

In summary, Drammensvassdraget is a multifaceted river system with significant implications for energy production, flood management, and ecological conservation in Norway. The careful and integrated management of this river system is essential for ensuring its continued contribution to the region's sustainable development and environmental health.

Key Reservoirs

Tyrifjorden

Tyrifjorden, Norway’s fifth-largest lake, is part of the Drammensvassdraget system, serving as a natural regulator for downstream flow and a significant resource for hydropower generation. This lake, situated in the municipalities of Ringerike, Hole, Lier, and Modum in the county of Viken, spans nearly 137 square kilometers and has a reservoir volume of 134 million cubic meters (Tyrifjorden, 2024).

LRW	62 m.a.s.l.
HRW	63 m.a.s.l.
Area at HRW	136,56 km^2
Reservoir Volume	134 million m^3
Number of Hydropower plants	3
Mean Flood	64,2 m
5-Year Flood	64,7 m

10-Year Flood	64,9 m
20-Year Flood	65,1 m
50- Year Flood	65,2 m

Table 1-1 Tyrifjorden Factsheet (NVE Atlas, u.d.)

The water levels in Tyrifjorden are minorly regulated, with a low reference water level of 62 meters and a high reference water level of 63 meters (Holmqvist, 2000). This slight regulation helps maintain a balance between water conservation and flood prevention, crucial for both ecological stability and human activities. Tyrifjordens importance is highlighted by its use for hydropower, with three power plants: Geithusfoss, Gravfoss 1, and Gravfoss 2, which contribute significantly to the region's energy production (NVE - Tyrifjorden, 2024).

Sperillen

Sperillen, is within the Ådal valley in Ringerike municipality, Viken county, is a notable lake in Norway. Covering an area of about 37 square kilometers and stretching approximately 26 kilometers in length, Sperillen ranks as the 33rd largest lake in Norway. It lies at an elevation of 159 meters above sea level and is fed by the Begna and Urula rivers from the north, which contribute significantly to its volume and ecosystem (Lauritzen, 2023).

LRW	147,95 m.a.s.l.
HRW	150,25 m.a.s.l.
Area at HRW	37,32 km^2
Reservoir Volume	86,8 million m^3
Number of Hydropower plants	4
Mean Flood	151,1276 m
5-Year Flood	151,6132 m
10-Year Flood	152,0137 m
20-Year Flood	152,4 m
50-Year Flood	152,9034 m

Table 1-2 Sperillen Factsheet (NVE Atlas, u.d.)

The lake plays a crucial role within the Begnavassdraget, part of the larger Drammensvassdraget water system. With a substantial volume of 86.8 million cubic meters, Sperillen is integral to the region's regulated energy production. This is highlighted by its connection to four hydropower plants: Hensfoss, Begna, Hofsfoss, and Hønefoss, which utilize its waters for electricity generation. The careful regulation of Sperillens water levels,

maintained between 147.95 meters and 150.25 meters, ensures optimal conditions for both power production and flood management

Randsfjorden

Randsfjorden, the fourth largest lake in Norway, is a freshwater body spanning approximately 140 square kilometers at its highest regulated water level. Positioned within the counties of Innlandet and Viken, this lake plays a vital role in the local ecosystem and hydroelectric production. Randsfjorden has a substantial volume of more than 400 million cubic meters, making it an essential resource for energy generation and storage. The lake supports five hydropower plants: Bergerfoss, Kistefoss 1 and 2, Askerudfoss, and Viulfoss (Thorsnæs, Randsfjorden, 2023).

LRW	131,3 m.a.s.l.
HRW	134,5 m.a.s.l.
Area at HRW	140,75 km^2
Reservoir Volume	408,6 million m^3
Number of Hydropower plants	5
Mean flood	134,689 m
5-year flood	134,9159 m
10-year flood	135,1058 m
20-year flood	135,2902 m
50-year flood	135,5321 m

Table 1-3 Randsfjorden Factsheet (NVE Atlas, u.d.)

Randsfjordens hydrological significance is underscored by its contributions to the Drammensvassdraget system. The lake is fed by several rivers, including Etna, Dokka, Vigga, and Fallselva, and drains into Randselva at its southern end. This connectivity facilitates the management of water flow and energy production, highlighting the lake's integral role in regional water resource management (Randsfjorden, 2024).

Powerplants in Drammensvassdraget

The Drammensvassdraget system is home to several hydropower plants that play a vital role in Norway's renewable energy production. These plants harness the flow of water from significant lakes within the system, including Tyrifjorden, Sperillen, and Randsfjorden, each contributing to the region's energy supply and flood management capabilities.

These powerplants collectively underscore the Drammensvassdraget system's significance in Norway's renewable energy landscape, highlighting the integration of natural resources and technological advancements to meet energy demands sustainably.

Tyrifjorden			
Plant Name	Geithusfoss	Gravfoss 1	Gravfoss 2
Max Effect	13,5 MW	18,6 MW	30,2 MW
Gross Head	9,19 m	19,7 m	20 m
Energyequivelant	0,025 kWh/m^3	0,044 kWh/m^3	0,048 kWh/m^3

Sperillen				
Plant Name	Hensfoss	Begna	Hofsfoss	Hønefoss
Max Effect	18,3 MW	5,6 MW	27 MW	29,4 MW
Gross Head	24,4 m	8 m	26,79 m	21,5 m
Energyequivelant	0,055 kWh/m^3	0,018 kWh/m^3	0,061 kWh/m^3	0,051 kWh/m^3

Randsfjorden					
Plant Name	Bergerfoss	Kistefoss 1	Kistefoss 2	Askerudfoss	Viulfoss
Max Effect	3,3 MW	1,4 MW	4,2 MW	13,2 MW	12,5 MW
Gross Head	5,4 m	9 m	10,5 m	20,6 m	17,29 m
Energyequivelant	0,013 kWh/m^3	0,018 kWh/m^3	0,025 kWh/m^3	0,048 kWh/m^3	0,042 kWh/m^3

Table 1-4 Powerplants in key reservoirs (Vannkraftdatabase, 2024).

2. Theoretical Framework

Kernel Density Estimation (KDE):

Kernel Density Estimation (KDE) is a non-parametric way to estimate the probability density function (PDF) of a random variable. Unlike parametric methods, KDE does not assume a specific distribution model for the data. Instead, it uses a smooth function (kernel) to create a continuous estimate of the data's distribution.

Mathematical Representation:

The KDE estimate f(x) at point x is given by:

$$\hat{f}(x) = \frac{1}{nh}\sum_{i=1}^{n} K\left(\frac{x - x_i}{h}\right)$$

Where:

- *n* is the number of data points.
- *h* is the bandwidth parameter, controlling the smoothness of the estimate.
- K(·) is the kernel function, commonly a Gaussian function:

$$K(u) = \frac{1}{\sqrt{2\pi}} exp\left(-\frac{u^2}{2}\right)$$

KDE is particularly useful for visualizing the underlying distribution of data, identifying modes, and detecting the presence of multimodal distributions. In hydrology, it is used to analyze the distribution of water levels, helping in understanding the patterns and estimating probabilities of extreme events.

Volatility:

Volatility is best known as a statistical measure of the dispersion of returns for a given security or market index. It indicates the degree of variation of a financial instrument's price over time. In the context of hydrology, volatility can describe the variability in water levels, over time.

Mathematical Representation:

Volatility is often quantified using the standard deviation of returns σ:

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

where x_i represents individual observations, $\bar{x}$ is the mean of the observations, and N is the number of observations.

In hydrological studies, understanding the volatility of water levels helps in assessing the risks associated with high variability, which is crucial for flood risk management and reservoir operation strategies.

Energyequivelant:

The Energy Equivalent is the amount of energy that can be generated from a unit of water. It is a critical concept in hydropower engineering, linking reservoir levels to energy potential. Each Hydropower power plant has its own Energyequivelant, which can be seen in table 1-4. This equivalent can be multiplied by the reservoir volume to get the amount of energy that volume represents.

Hydropower Regulation and Operation:

Hydropower Regulation and Operation involve the rules, practices, and technical measures used to manage water flow and reservoir levels. This includes maintaining reservoir levels, controlling water discharge, and optimizing energy production while minimizing flood risks.

Key Elements:

- **Regulatory Guidelines**: Standards and rules set by authorities (NVE in Norway).
- **Operational Strategies**: Techniques for reservoir management, including flood control and energy production.
- **Technical Measures**: Use of gates, turbines, and other equipment to control water flow.

Manøvreringsreglementet (Norwegian Regulation):

- **Description**: Norwegian regulations governing the operation and management of hydropower plants.
- **Relevance**: Sets the legal and operational boundaries within which the decision-support model operates.

Kruskal-Wallis Test:

The Kruskal-Wallis Test is a non-parametric method for testing whether samples originate from the same distribution. It extends the Mann-Whitney U test to multiple groups. This test does not assume a normal distribution of the data, making it suitable for comparing more than two groups.

Mathematical Representation

$$H = \frac{12}{N(N+1)} \sum_{i=1}^{k} \frac{R_i^2}{n_i} - 3(N+1)$$

where:

- N is the total number of observations.
- k is the number of groups.
- N_i is the number of observations in the i-th group.
- R_i is the sum of ranks for the i-th group.

The Kruskal-Wallis Test is used to compare water levels across different seasons, identifying significant differences. This helps in understanding seasonal variations and their impact on water resource management.

P-Value:

The P-Value is the probability of obtaining test results at least as extreme as the observed results, assuming that the null hypothesis is correct. It provides a measure of the evidence against the null hypothesis. The P-Value is calculated based on the test statistic from a hypothesis test. For example, in the context of the Kruskal-Wallis Test, the P-Value is derived from the chi-square distribution.

In the context of the Kruskal-Wallis Test, the P-Value is used to determine the statistical significance of the observed differences between groups. A low P-Value indicates strong evidence against the null hypothesis, confirming significant seasonal differences in water levels.

Sensitivity Testing:

Sensitivity Testing involves analyzing how different values of an independent variable affect a particular dependent variable under a given set of assumptions. It assesses the impact of varying input parameters on model outputs.

Sensitivity Testing is essential for assessing the robustness of the decision-support framework. It identifies which parameters significantly influence the model's outcomes, guiding improvements and ensuring reliable predictions under varying conditions.

*Global and Local Variables**:***

Global Variables are parameters that affect the entire model universally, while Local Variables impact specific instances or parts of the model. This distinction helps in managing the complexity and scope of the model.

Differentiating between global and local variables is important for defining the scope and impact of different parameters within the decision-support model. This helps in managing the model efficiently and ensuring accurate predictions related to the dual objectives of energy production and flood risk management.

3. Research Design

This chapter is meant as an overview of the research, with the methodology for the decision framework presented in chapter 5. The reason for this separation is the nature of the research. Before any method and choices for the framework can be made, the data and statistical properties must be evaluated. The first step in the method will therefore be choice of path moving forward in the decision framework.

The research employs a quantitative approach, focusing on a thorough analysis of the statistical characteristics. It assumes that the reservoirs show strong seasonality and correlation in reservoir volume, water level and waterflow in each unique lake. This assumption is tested in the start of the Exploratory Data Analysis (EDA). By confirming the assumption, the statistical analysis can be done on water level, and not all the different types of water statistics. The primary steps of the research are:

1. **Data Collection:**
    - Gathering historical water data from Norwegian Water Resources and Energy Directorate.
2. **Data Cleaning and Preparation**
    - Ensuring the consistency and reliability of the data by aligning time series, removing inconsistencies such as zero-values, and ensuring all datasets cover the same period.
3. **Exploratory Data Analysis**
    - *Correlation Analysis:*
        - Analyze the correlation in water levels and reservoir levels to confirm assumption and to validate that water level data is a reliable indicator.
        - High correlations confirm the strong positive relationship, allowing the use of water level data for further analysis.
    - *Descriptive Statistics:*
        - Calculating key statistical measures (mean, standard deviation, skewness, kurtosis, etc.) to understand the central tendencies and variability of the water level.
        - Visualize the data in histograms, time-series and boxplots.
    - *Seasonal Analysis:*

        - Segmenting the data by seasons to capture the impact of seasonal variations.
        - Analyzing trends and cycles. Identify variability in a single season.
    - *Seasonal Analysis:*
        - Using histograms, Kernel Density Estimates (KDE), and the Kruskal-Wallis test to identify distinct seasonal modes in the water level data.
        - Confirming significant seasonal differences due to seasonal impacts.

4. **Decision Framework Development**
    - Methodology choice based on the insights from the EDA
    - Developing risk scores for flood and water shortage.
5. **Analysis**
    - Testing the robustness and responsiveness to various inputs to ensure it reliably responds to variations in water levels and seasonal factors.
    - Evaluating the performance and identifying areas for improvement by comparing the risk scores against historical events and expert assessments.

4. Exploratory Data Analysis

Dataset

The datasets are gathered from the Norwegian Water Resources and Energy Directorate (Sildre NVE, 2024). By using open-source data, the daily datapoints for Tyrifjorden, Sperillen and Randsfjorden were downloaded. This data was uncleaned, and not processed to fit with the study. The first action was to conduct a cleaning and preparation for the forthcoming research. The cleaning was done with python program *Data_Cleaner.py*, which can be found in appendix 1. The cleaning is intended to have the datasets align in time and datapoints. Below is the result of the cleaning of the daily waterlevels.

Tyrifjorden	
Uncleaned	
Range	1994 – 2024
Datapoints	10888
Cleaned	
Range	2004 – 2023
Datapoints	7305
Sperillen	
Uncleaned	
Range	1947 – 2024
Datapoints	28252
Cleaned	
Range	2004 - 2023
Datapoints	7305
Sperillen	
Uncleaned	
Range	1947 – 2024
Datapoints	28252
Cleaned	
Range	2004 - 2023
Datapoints	7305

Table 4-1 Cleaned Datasets

As we can see from the uncleaned data, Sperillen had a significant larger dataset for daily waterlevels. Moreover, due to changes in Randsfjord regulations of water, the data was best suited with the range 2004 throughout 2023 (Olje- og energidepartementet, 2022).

Additionally, the occurrences of zero-values were investigated in the cleaned data. If this occurred more research into the integrity of the data would have to be conducted. In the cleaned data there were no occurrences of zero-values. After the initial cleaning this is what the top and bottom of the dataset looks like, Tyrifjorden is used as example.

```
Top of Cleaned Dataset
        Date  Waterlevel
0 2004-01-01    62.59000
1 2004-01-02    62.57000
2 2004-01-03    62.56450
3 2004-01-04    62.54617
4 2004-01-05    62.53410

Bottom of Cleaned Dataset
           Date  Waterlevel
7300 2023-12-27    62.60221
7301 2023-12-28    62.58148
7302 2023-12-29    62.56809
7303 2023-12-30    62.55120
7304 2023-12-31    62.53333
Save? (yes/no): yes
```

Figure 4-1 Python Printout of the Cleaned Tyrifjorden Dataset

As showed in the printout above all datasets will have two columns, one for Date and one for Water level, Reservoir Volume or Waterflow. All datasets can be downloaded from NVE, see *chapter 13 – Dataset Downloading*.

Correlation

The first step of the EDA is to analyze the correlation between the possible datasets. As mentioned previously the assumption is that the datasets are highly correlated. Reservoir and Waterlevel almost or exactly perfect correlation, with waterflow possibly lagging slightly. The reason for the lag can be because of immediate weather changes, or the capacity. A watershed only has the possibility to let out that much water. At HRW the hatchets are open max, so the excess over there is the max waterflow. Therefore, there is a maximum the

waterflow can go, even though the water might still be rising. This is what causes a flood, and the overall theme of these research.

By doing this first step, the data analysis will be less extensive then if the thesis will need more than one statistical analysis. Given that if there is a good correlation the analysis and framework can rely on only one of the datasets for the most part. Below are the correlation matrices given from the correlation analysis.

Tyrifjorden			
	Waterlevel	Waterflow	Reservoir
Waterlevel	1	0,89595	0,999935
Waterflow	0,89595	1	0,89906
Reservoir	0,999935	0,89906	1
Sperillen			
	Waterlevel	Waterflow	Reservoir
Waterlevel	1	0,707879	0,999633
Waterflow	0,707879	1	0,721373
Reservoir	0,999633	0,721373	1
Randsfjord			
	Waterlevel	Waterflow	Reservoir
Waterlevel	1	0,287398	0,999956
Waterflow	0,287398	1	0,289755
Reservoir	0,999956	0,289755	1

Table 4-2 Correlation Analysis of all Reservoirs

As the table shows the assumption was correct, and there is correlation enough to rely on only waterlevel in the statistical analysis. The waterflow was slightly lower correlated, and even more in Randsfjord, this is assumed to be because of the mentioned capacities of the watersheds. Randsfjord stands out with a lower correlation on waterflow.

After the waterlevel datasets had shown to be a reliable set to analyze, the study moved on to perform a correlation analysis on the waterlevels across the three reservoirs. This is done to get an early indication of interconnection and gives the research more reliability that there can be made one framework that works sufficient across the three reservoirs. The correlation matrix is shown below.

	Tyrifjorden	Sperillen	Randsfjord
Tyrifjorden	1	0,761035	0,633054
Sperillen	0,761035	1	0,775634
Randsfjord	0,633054	0,775634	1

Table 4-3 Correlation Analysis of all Reservoirs

All lakes show significant positive correlation in water levels, indicating that changes in one lake's water level are likely to impact each other or have connecting events. The matrix implies that the relationship between water levels across these lakes is connected, where increases or decreases in one are reflected in the others.

The correlation analysis was done using *Correlation.py*, Appendix 3, and the complete analysis with heatmaps can be seen in Appendix 2.

Descriptive Statistics

The first step of this research is to get a general overview of the statistics for the three reservoirs. This section uses multiple python programs that will be included in the appendix. The second assumption made in this research is the seasonality plays a major part. For that reason, the analysis is divided into three parts, Statistical Analysis, Seasonal Analysis and a Multimodal Analysis.

Statistical Analysis

The statistical analysis seeks to understand the distribution of waterlevels across the range. Analyzing outliers, general statistical measurements, and variability.

The next sections provide a statistical overview of the waterlevels in Tyrifjorden, Sperillen and Randsfjord, employing data from the Python program, *Statistical_Analysis.py*, Appendix 4. The datasets comprise of 7305 datapoints for Sperillen and Tyrifjorden, and 7298 for Randsfjord, offering a robust basis for evaluating the waterlevel dynamics in the three reservoirs.

The complete statistical analysis for all reservoirs can be found in the appendix 5-8.

Tyrifjorden

Tyrifjorden is the last lake in the system, with Sperillen and Randsfjorden being upstream of Tyrifjorden, connected by Randselva and Begnavassdraget. The mean waterlevel across the dataset is recorded at approximately 62,92 meters, which is quite high in the regulation zone. That zone being between 62 and 63 meters, LRW and HRW respectively. With a standard deviation of 0,38 meters, indicating moderate variability around the mean. Suggesting that the waterlevel occasionally goes above HRW but not often closing in on LRW. Since the mean-flood level for Tyrifjorden is 64,2 meters, going slightly above HRW is not dramatic.

Statistic	Value
Mean	62,9181
Standard Deviation	0,377319
Min	62,02999
25th percentile	62,75689
Median	62,8723
75th percentile	62,99604
90th percentile	63,31896
95th percentile	63,63215
99th percentile	64,35287
Max	65,40757

Table 4-4 Tyrifjorden Descriptive Statistics

From the table above it can be noted that 75% of the waterlevels are below HRW of 63 meters.

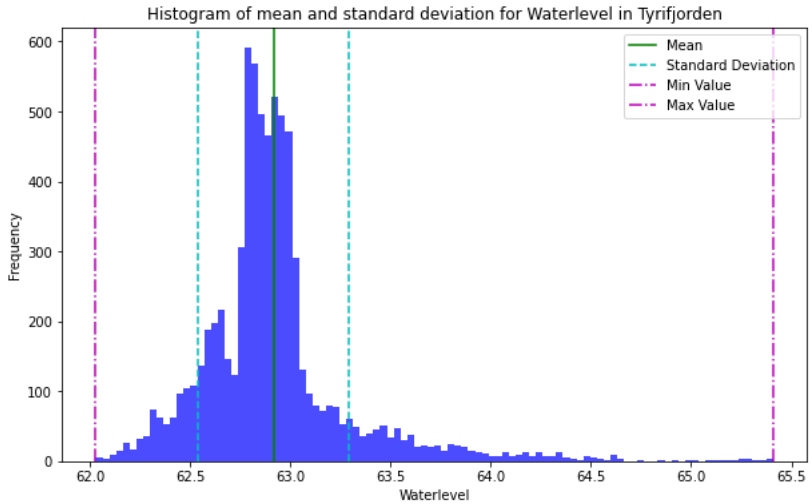


Table 4-5 Tyrifjorden Histogram Distribution of Waterlevels

There is a notable decrease in waterlevels directly above HRW, which corresponds closely to the 75th percentile. The distribution also highlights the rarity of extreme waterlevels, on either side. Below 62,5 meters and above approximately 63,3 meters there are not many recorded waterlevels.

Time-Series Analysis

A time series analysis over more than two decades shows consistent seasonal fluctuations, underscoring the assumed seasonality in the waterlevels.

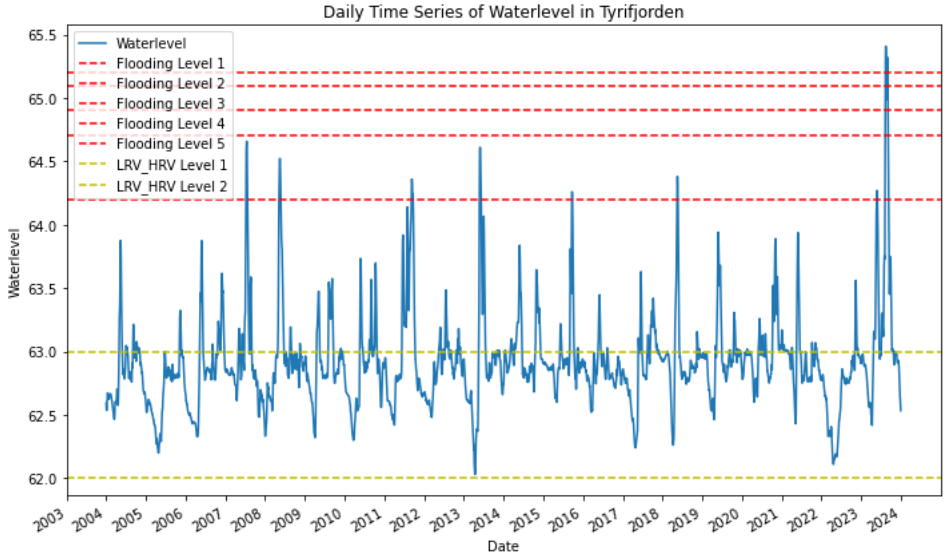


Figure 4-2 Time-Series Chart Tyrifjorden

The water levels generally remain within a defined range, with occasional spikes that exceed the flooding thresholds marked by the red dashed lines in the analysis.

Statistical Measures and Flood Incidence

78.81% of observed waterlevels fall within one standard deviation from the mean. This tight clustering is more pronounced than in a standard normal distribution, suggesting predictability in water level behaviors.

Frequency			**Standard Deviation Analysis**	
Condition	*Days*	*Percent*	*Number of Std Devs*	*Percentage Within Range*
Regulation Zone	5512	75,45517	1	78,80903
Caution Zone	1653	22,62834	2	90,63655
Mean to 5-Year Flood	86	1,177276	3	95,6742

5 to 10-Year Flood	2	0,027379	4	98,38467
10 to 20-Year Flood	6	0,082136	5	99,28816
20 to 50-Year Flood	4	0,054757	6	99,58932
50-Year Flood	15	0,205339	7	100
Total Flood Days	113	1,546886	8	100

Table 4-6 Frequency and Standard Deviation Analysis Tyrifjorden

The data categorizes 1.547% of the observation period as flood days, emphasizing the low but non-negligible risk of flooding. The 15 days of 50-Year Flood is the extreme weather in 2023. As we have seen in the former statistics a certain amount of the water level is recorded above HRW for Tyrifjorden. Tyrifjorden can probably do this due to the relatively big margin from HRW to mean-flood.

Sperillen

The mean water level in Sperillen is approximately 149.63 meters with a standard deviation of 0.71 meters, reflecting a moderate level of variability. Although, a significant increase from the variability in Tyrifjorden.

Statistic	Value
Mean	149,6315
Standard Deviation	0,71156
Min	148,1312
25%	149,0202
Median	149,7027
75%	150,1673
90%	150,3899
95%	150,5535
99%	151,6396
Max	154,023

Table 4-7 Sperillen Descriptive Statistics

The data is mainly centralized; the histogram below illustrates that most water levels are tightly clustered around the mean and median (149.70 meters). Moreover, there are sharper declines after one standard deviation from the mean.

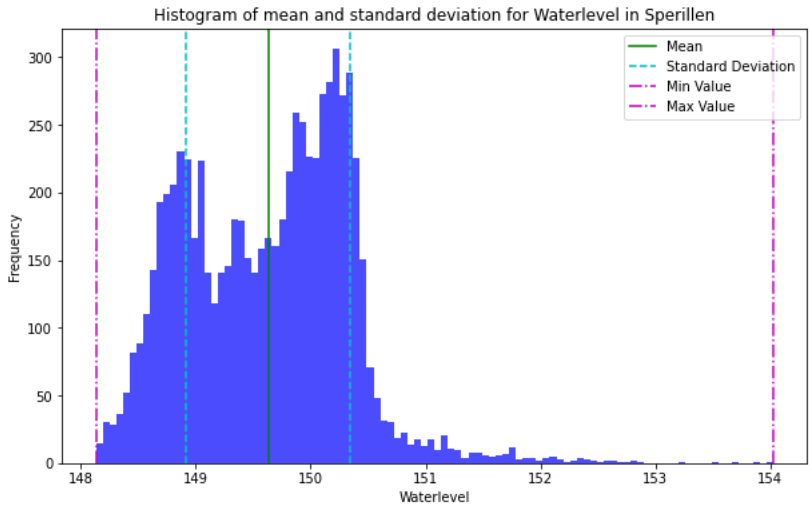


Figure 4-3 Sperillen Histogram Distribution of Waterlevels

Notably, the frequency of occurrences diminishes significantly for water levels above the 75th percentile, highlighting the infrequency of extremely high-water levels, which peak at a maximum of 154.02 meters. Much like Tyrifjorden.

Time-Series Analysis

The time-series for Sperillen resembles Tyrifjorden and reveals a pattern of season water level fluctuations.

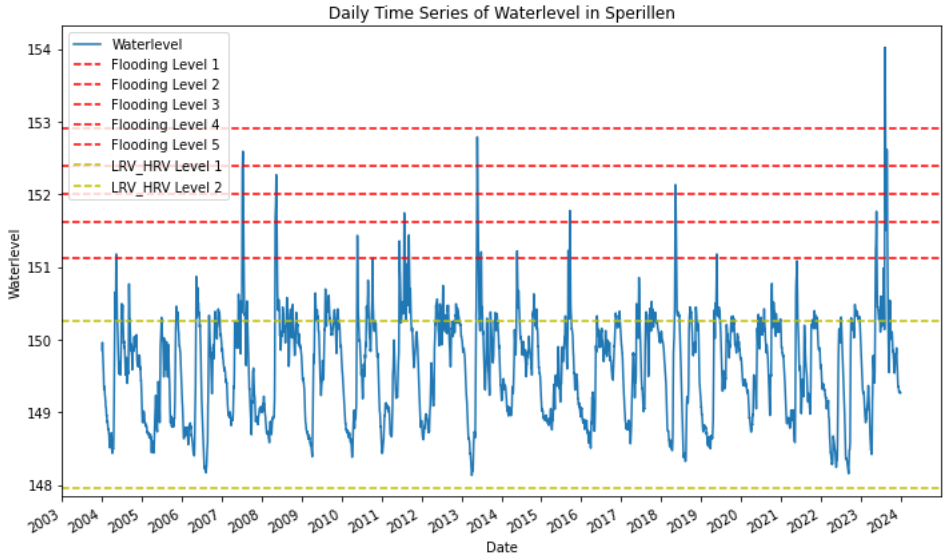


Figure 4-4 Time-Series Graph Sperillen

The water levels oscillate within the regulated range (LRW/HRW), marked by yellow dashed lines, suggesting consistent management and predictable behavior of the lake over time. Notably, critical flooding thresholds indicated by red dashed lines are seldom exceeded.

Statistical Measures and Flood Incidence

Much like Tyrifjorden, Sperillen shows approximately 73.85% of the data points are within one standard deviation from the mean, indicating less variability than a normal distribution might suggest.

Frequency			**Standard Deviation Analysis**	
Condition	*Days*	*Percent*	*Number of Std Devs*	*Percentage Within Range*
Regulation Zone	5919	81,02669	1	73,85352
Caution Zone	1233	16,87885	2	95,50992
Mean to 5-Year Flood	74	1,013005	3	98,4668
5 to 10-Year Flood	38	0,520192	4	99,56194
10 to 20-Year Flood	21	0,287474	5	99,86311
20 to 50-Year Flood	13	0,17796	6	99,97262
50-Year Flood	5	0,068446	7	100
Total Flood Days	151	2,067077	8	100

Table 4-8 Frequency and Standard Deviation Analysis Sperillen

The data also shows that 2.07% of the observation days fall under various flood conditions, underscoring the occasional but important flood risk.

Randsfjorden

The average water level of Randsfjord stands at approximately 133.50 meters with a standard deviation of 0.87 meters. This level of deviation suggests a moderate fluctuation around the mean, primarily staying within a predictable range.

Statistic	**Value**
Mean	133,5049
Standard Deviation	0,871948
Min	131,43
25%	132,84
Median	133,9
75%	134,15
90%	134,35
95%	134,4618
99%	134,66
Max	136,07

Table 4-9 Randsfjorden Descriptive Statistics

Most water levels are clustered around the median of 133.9 meters, and the frequency distribution decreases for levels beyond the 75th percentile, culminating at a maximum of 136.07 meters.

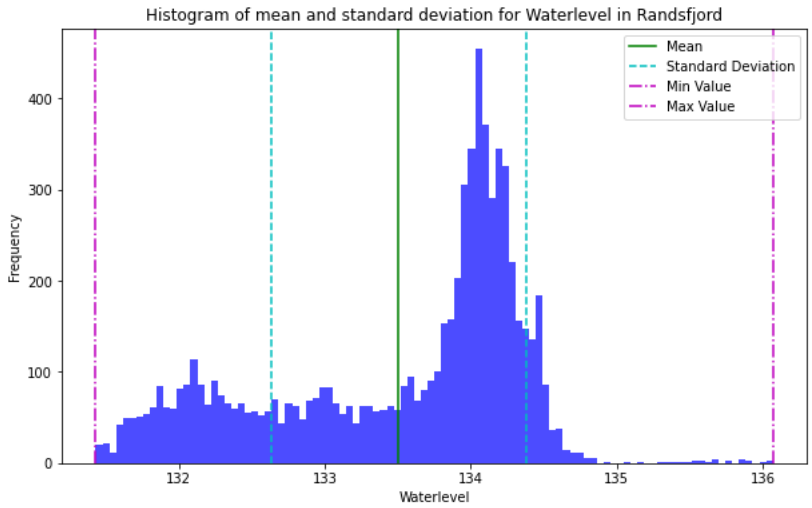


Figure 4-5 Randsfjorden Histogram Distribution of Waterlevels

Notably, there looks to be a higher frequency in the lower range toward LRW, than with the two other reservoirs.

Time-Series Analysis

The time series analysis spanning over two decades shows that Randsfjord maintains a stable water level with regular seasonal variations. These variations are well-contained within the established regulatory thresholds.

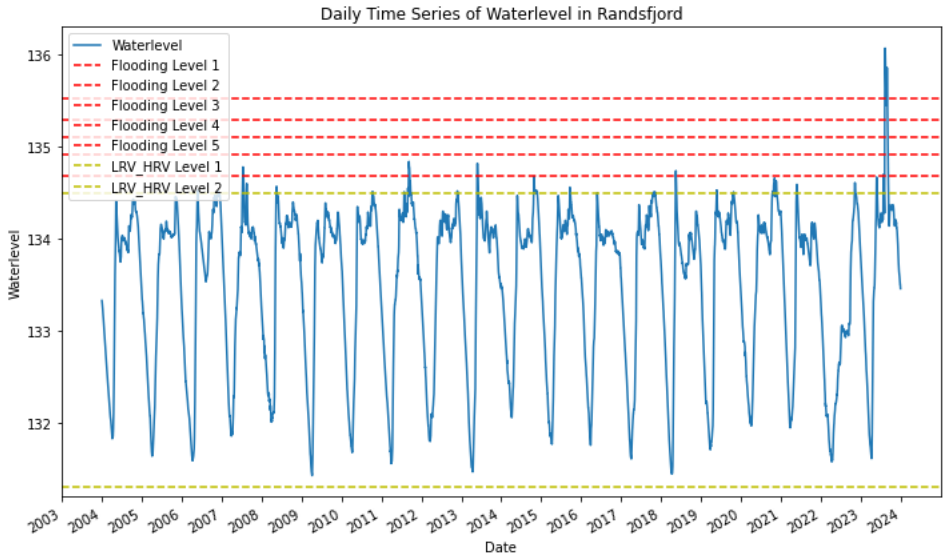


Figure 4-6 Time-Series Graph Randsfjorden

Statistical Measures and Flood Incidence

An impressive 82.65% of the data points lie within one standard deviation from the mean, emphasizing the lake's stability, uniformity and predictability.

Frequency			Standard Deviation Analysis	
Condition	*Days*	*Percent*	*Number of Std Devs*	*Percentage Within Range*
Regulation Zone	7063	96,77994	1	82,65278
Caution Zone	146	2,000548	2	99,26007
Mean to 5-Year Flood	34	0,465881	3	100
5 to 10-Year Flood	2	0,027405	4	100
10 to 20-Year Flood	2	0,027405	5	100
20 to 50-Year Flood	5	0,068512	6	100
50-Year Flood	21	0,28775	7	100
Total Flood Days	64	0,876953	8	100

Table 4-10 Frequency and Standard Deviation Analysis

The occurrence of days with flooding conditions is remarkably low (less than 1%), which reinforces the effectiveness of the existing water management strategies to handle high-water events. Randsfjorden stands out with its low flooding and highly regulated waterlevels between LRW and HRW.

Summary

The statistical analyses of water levels in Tyrifjorden, Sperillen, and Randsfjord provide insights into the hydrological stability and variability of these lakes. Each analysis, grounded in robust datasets and comprehensive statistical metrics, underscores both the individual characteristics and shared behaviors of these water reservoirs.

Across all three lakes, the analyses highlight a strong tendency toward central clustering of water levels around the mean, with water levels falling within a predictable range. The frequency of floods is shown to be rare, but noticeable for Sperillen and Tyrifjorden. Randsfjorden stands out with an impressive flooding percent below 1.
Despite similarities in management success, the reservoirs exhibit varying degrees of natural variability. For example, Randsfjord shows remarkable predictability with 82.65% of observations falling within one standard deviation from the mean, compared to 78.81% for Tyrifjorden and 73.85% for Sperillen.

The risk of flooding, while generally low across all reservoirs, is meticulously documented, with each lake experiencing rare but notable high-water. Randsfjord displays a very low incidence of flood days (0.88%).

Seasonal Analysis

Based on the statistical analysis the study observed seasonal patterns that needs to be analyzed. The seasonal analysis will separate the datasets in seasons, and perform the same statistical analysis done before. Furthermore, the trends for the seasons will be analyzed. The complete seasonal analysis can be found in the appendix 9-11. The analysis is done in the python program *Seasonal_Analysis_Waterlevel.py,* appendix 8. For the datasets to start at the start of a season, the datasets will be filtered to start 1st March 2004. The first season will then be spring 2004.

Tyrifjorden

The seasonal statistics table provides a comprehensive overview of the mean, standard deviation, minimum, and maximum water levels for each season. The data reveals that autumn has the highest mean water level, indicating generally stable conditions with occasional peaks. Spring, on the other hand, shows the lowest mean but the highest standard deviation, reflecting significant variability.

Season	mean	std	min	25%	50%	75%	max	SVI
Autumn	63,02	0,32	62,48	62,83	62,94	63,04	65,25	0,007203
Spring	62,79	0,45	62,03	62,48	62,73	62,97	64,61	0,006588
Summer	63,06	0,42	62,27	62,82	62,92	63,12	65,41	0,005076
Winter	62,81	0,17	62,33	62,66	62,84	62,94	63,61	0,00273

Table 4-11 Tyrifjorden Seasonal Statistics

Summer's water levels are comparable to autumn's, with considerable variability suggesting extreme weather events. Winter displays the lowest variability, indicating more consistent water levels likely due to freezing conditions.

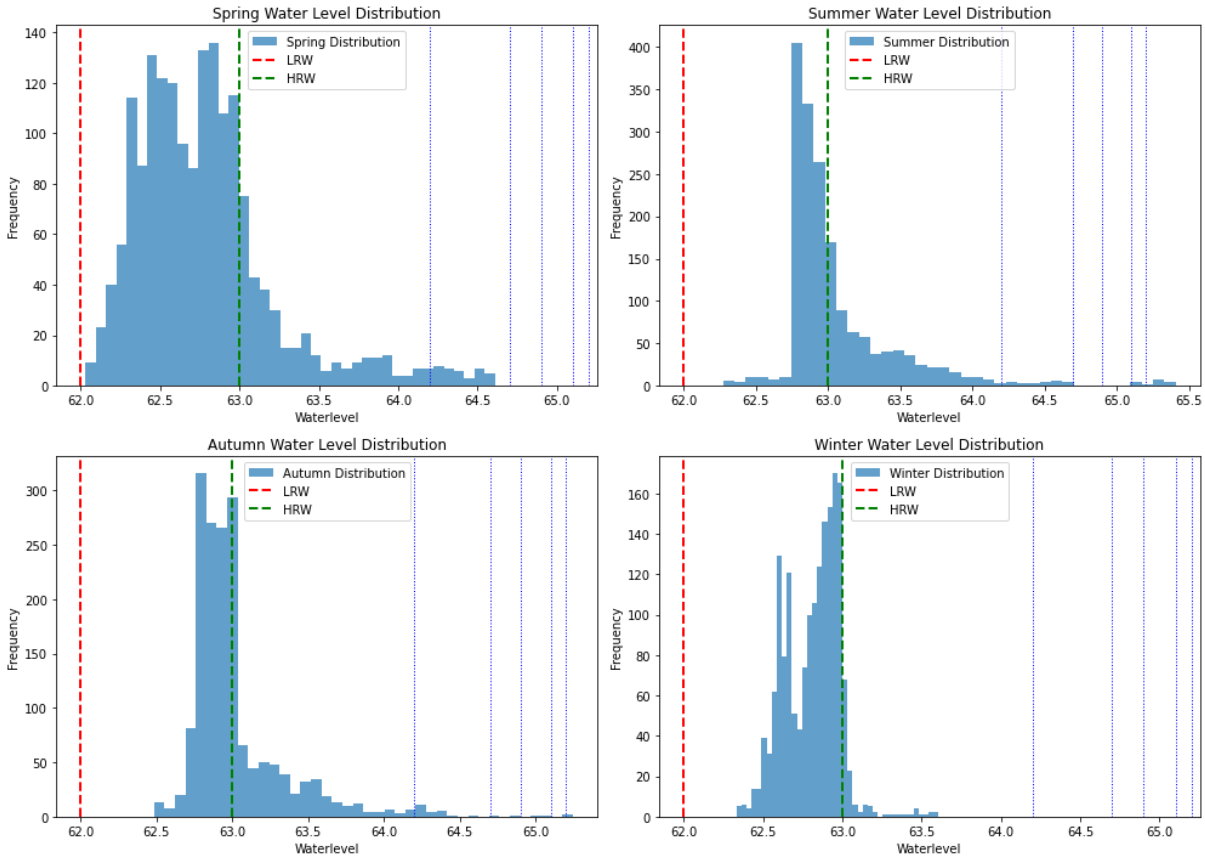


Figure 4-7 Tyrifjorden Seasonal Histograms

The seasonal changes are quite apparent with winter having no floods and stable waterlevels, before the waterlevel increase when spring comes. There are incidents of flood in the spring. As summer histogram shows the higher waterlevels seem to come from an increasing level throughout spring. The year ending with a declining waterlevel in autumn, moving into winter.

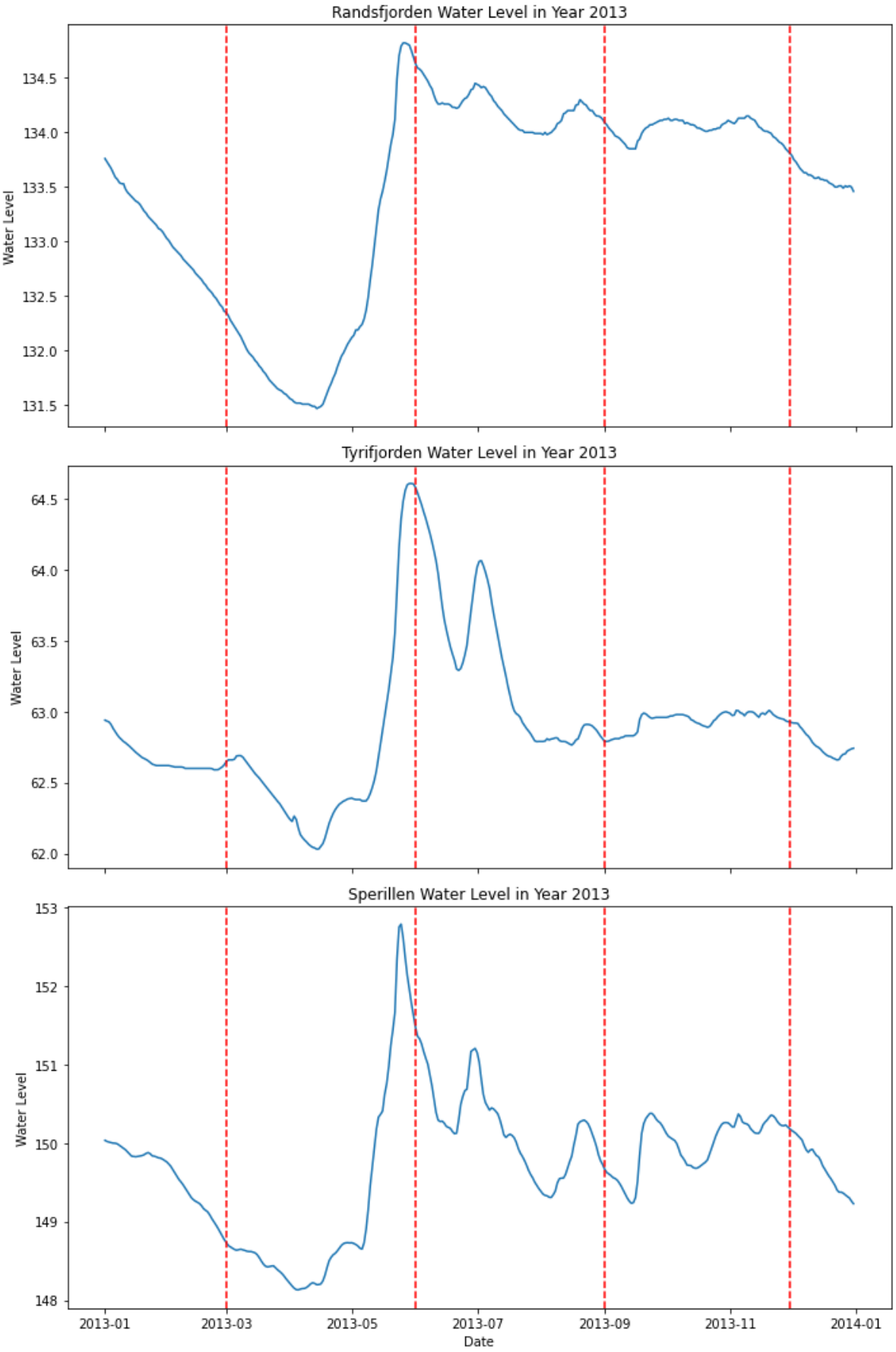


Figure 4-8 Yearly Plot Example All lakes

What we can take out from the time-series above, from 2013, is the seasonal changes connecting with the histogram. 2013 is taken as an example, and this graph is for all reservoirs. It will not be shown in the other reservoirs seasonal analysis. The plots can be made using python and the python program *Yearly_plots.py*. The graph shows and increasing waterlevel in spring, and large outflow during summer. Summer and autumn displaying more volatility due to changing weather. The spring smelt is what makes the higher waterlevels in summer.

Time-Series and Seasonal Fluctuations

The decomposed time-series analysis offers a clear visualization of the waterlevels over two decades, capturing both the observed values and the seasonal components. The actual observed water levels exhibit sharp peaks and downs, highlighting significant fluctuations and extreme events.

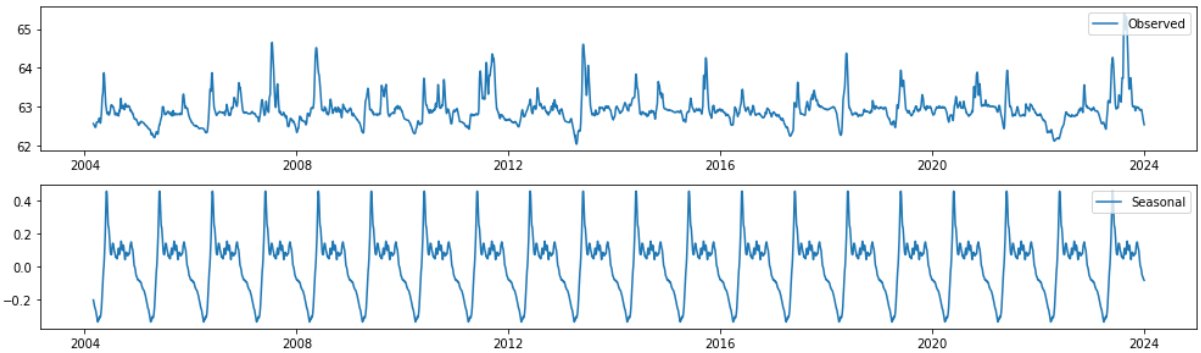


Figure 4-9 Decomposed Time Series with fluctuations

This pattern is particularly pronounced in spring and summer, where climatic factors such as precipitation and snowmelt probably contribute to the variability. The seasonal component of the time-series shows a consistent cyclical pattern, underscoring the strong influence of seasonal changes on water levels. This regular cycle suggests that despite yearly variations, the underlying seasonal trends remain stable, driven by predictable factors.

Flood Incidence

Spring and summer are marked by higher variability and a greater incidence of flood days. Summer experiences a high frequency of flood days. Autumn shows a reduction in flood days compared to summer, reflecting a transition to more stable water levels. Winter, with its low variability and absence of flood days, presents the most stable scenario, likely due to freezing conditions that limit water level fluctuations.

	Spring	Summer	Autumn	Winter
Below 62	0	0	0	0
62 to 63	1381	1123	1147	1633
63 to 64.2	329	578	535	131
64.2 to 64.7	38	26	22	0
64.7 to 64.9	0	0	2	0
64.9 to 65.1	0	4	2	0
65.1 to 65.2	0	3	1	0
Above 65.2	0	14	1	0
Total Flood Days	38	47	28	0

Table 4-12 Flood Frequencies Tyrifjorden

Box plots for each season provide additional insights into the distribution and spread of water levels. These plots reveal not only the central tendencies but also the range and presence of outliers. Spring and summer show higher mean levels and more pronounced spreads, as evidenced by the interquartile ranges, suggesting more substantial fluctuations in water levels during these periods. The box plots can be seen in the Seasonal Analysis Tyrifjorden appendix.

Seasonal Trend

The trend analysis across different seasons reveals distinct patterns in waterlevel changes. Spring shows a positive slope, suggesting an overall increase in water levels as the season progresses, which may be due to snowmelt and increased rainfall. This trend highlights the potential for increased flooding risks in spring, necessitating proactive water management strategies. In contrast, summer, autumn, and winter exhibit negative slopes, indicating a general decline in water levels throughout these seasons.

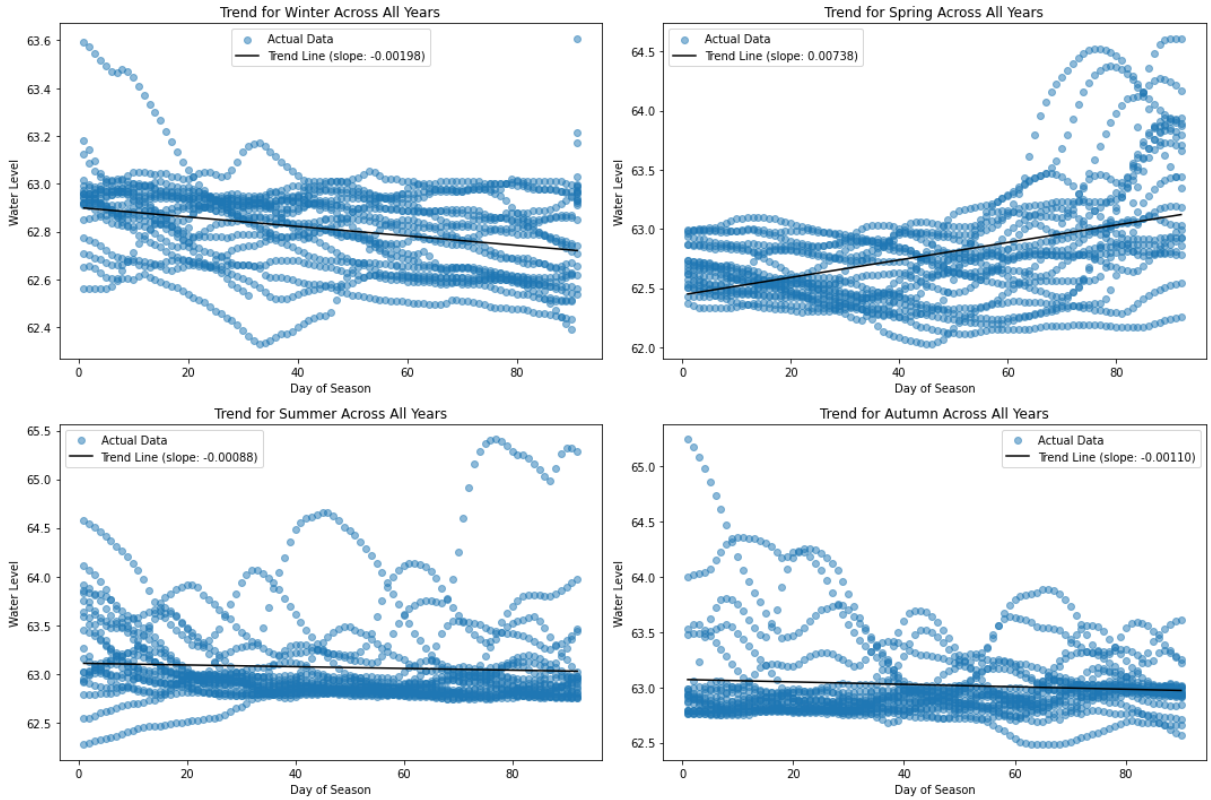


Figure 4-10 Seasonal Trend Tyrifjorden

The statistical significance of these trends is confirmed by p-values, found in appendix, well below the 0.05 threshold, indicating that these patterns are not due to random chance. However, the practical implications of these trends require careful consideration. Given that the trends, while significant, are possibly not practical, the trend is not large. The trend is not from a certain water level to a flooding level.

Sperillen

Autumn shows relatively stable water levels with a modest variability, indicating a balanced hydrological state. In contrast, spring displays the highest variability

Season	mean	std	min	25%	50%	75%	max
Autumn	149,96	0,5	148,15	149,74	150,09	150,28	151,87
Spring	149,27	0,81	148,13	148,69	148,98	149,6	152,79
Summer	149,95	0,69	148,17	149,62	150,01	150,31	154,02
Winter	149,36	0,5	148,44	148,92	149,29	149,81	150,42

Table 4-13 Seasonal Statistics Sperillen

Summer's water levels are like those in autumn but with increased variability. Winter, with the lowest variability, indicates consistent water levels. The histograms of seasonal water levels provide insights into the distribution across different times of the year.

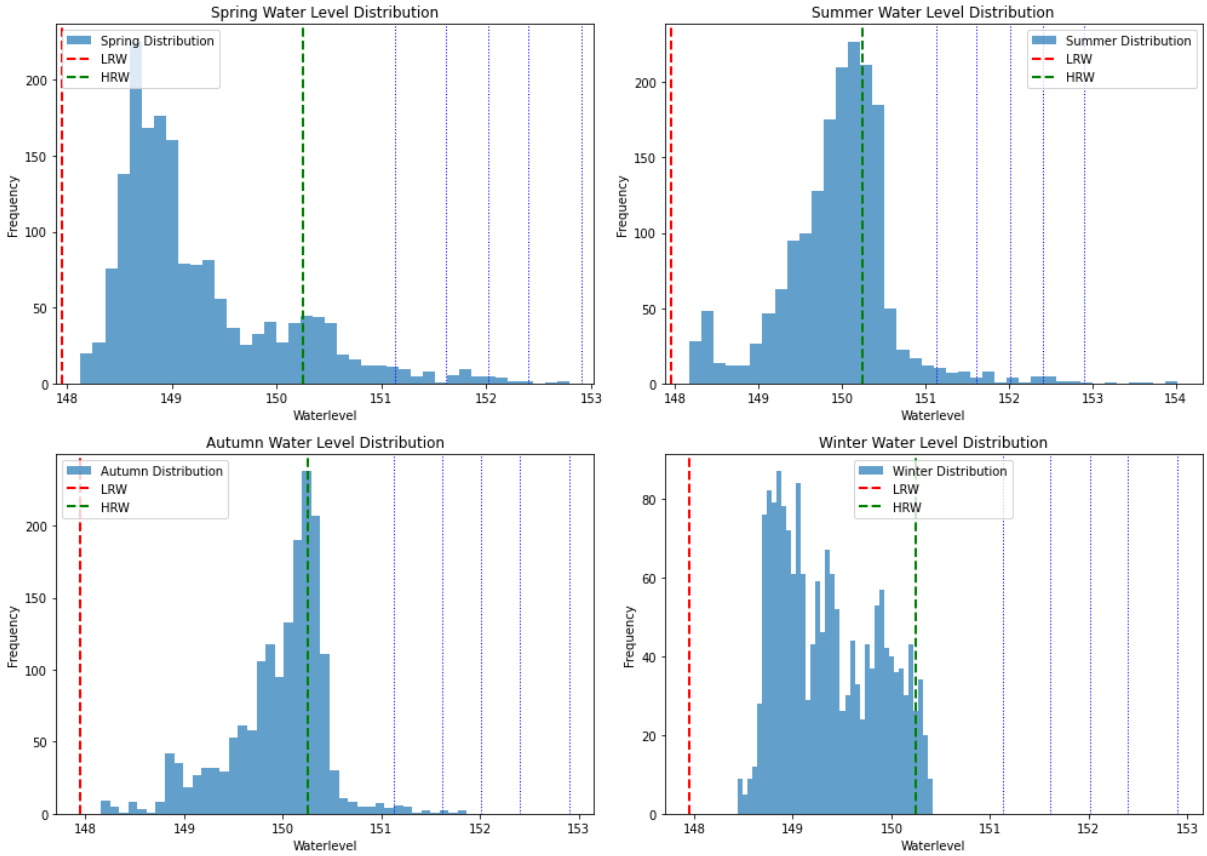


Figure 4-11 Seasonal Histograms Sperillen

Spring's histogram highlights a significant increase in water levels within the 149 to 150.25-meter range.

Time-Series and Seasonal Fluctuations

The decomposed time-series analysis of Sperillens water levels reveals significant seasonal fluctuations, characterized by peaks and downs. The actual observed water levels show substantial variability, particularly during spring and summer.

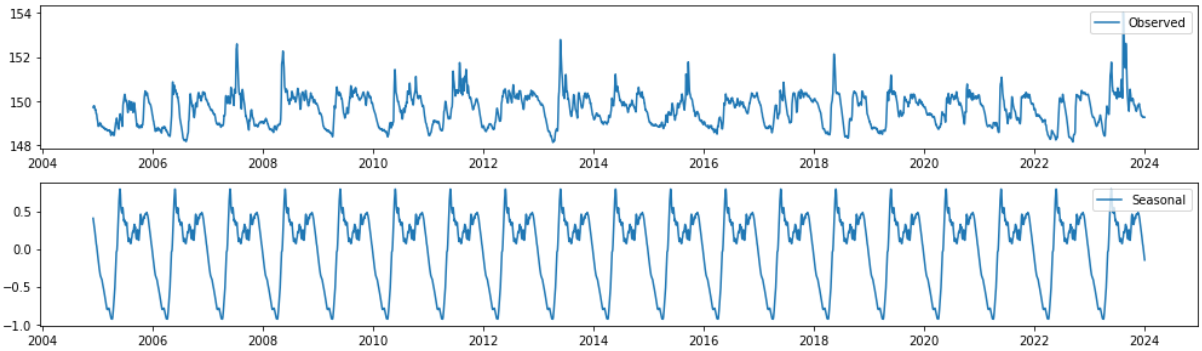


Figure 4-12 Time-Series with seasonal fluctuations Sperillen

The seasonal component of the time-series analysis showcases a predictable and repetitive pattern, underlining a seasonal effect on water levels. This consistent cycle indicates that despite inter-annual variations, the underlying seasonal trends remain stable, influenced by predictable climatic factors.

Variation and Flood Incidence

Like Tyrifjorden, spring and summer are marked by higher variability and a greater incidence of flood days.

Waterlevel	Spring	Summer	Autumn	Winter
Below 147.95	0	0	0	0
147.95 to 150.25	1494	1234	1216	1694
150.25 to 151.1276	188	452	474	70
151.1276 to 151.6132	29	26	16	0
151.6132 to 152.0137	23	11	4	0
152.0137 to 152.4	11	10	0	0
152.4 to 152.9034	3	10	0	0
Above 152.9034	0	5	0	0
Total Flood Days	66	62	20	0

Table 4-14 Frequency Sperillen

Summer experiences a high frequency of flood days. Autumn shows a reduction in flood days compared to summer, reflecting a transition to more stable water levels. Winter, with its low variability and absence of flood days, presents the most stable scenario.

Seasonal Trend

The trend analysis reveals distinct patterns in water level changes across different seasons. Spring shows a positive slope, suggesting an overall increase in water levels as the season progresses.

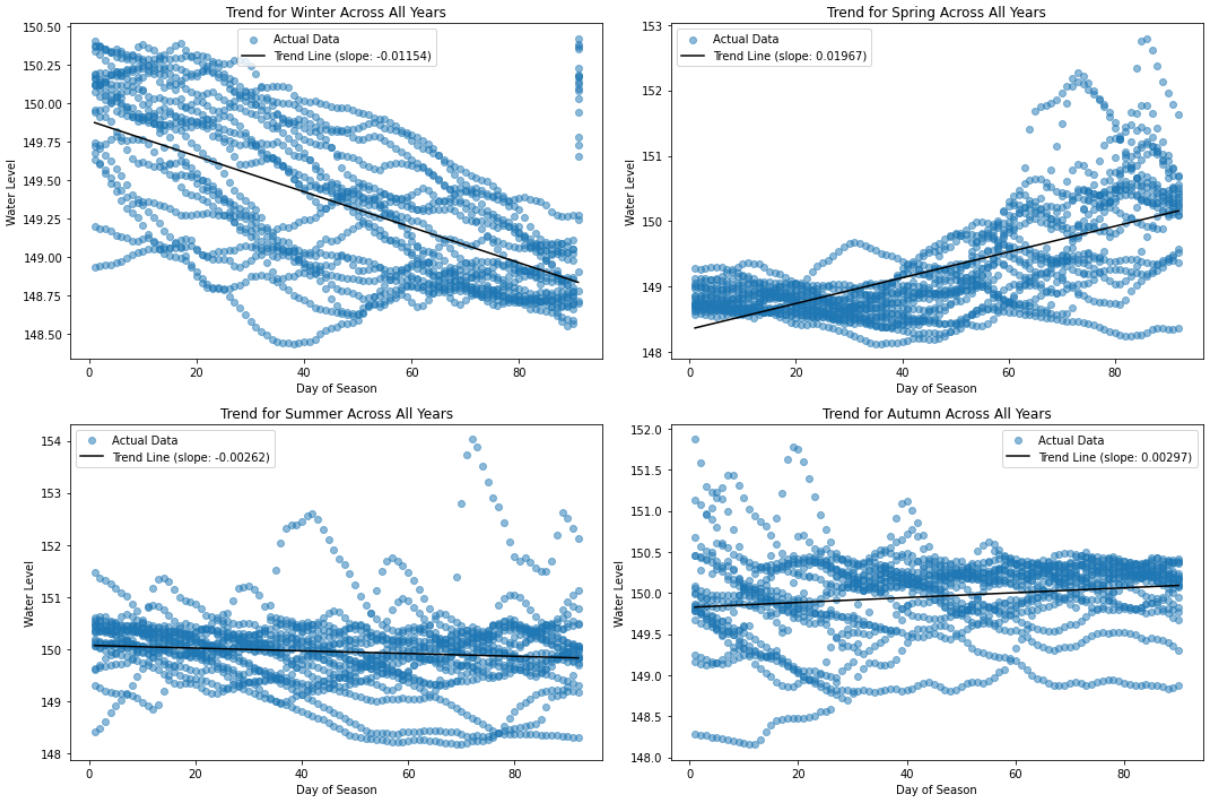


Figure 4-13 Seasonal Trend Sperillen

In contrast, summer, autumn, and winter exhibit negative slopes, indicating a general decline in water levels.

Randsfjorden

The seasonal statistics table for Randsfjord reveals a consistent pattern in water levels across different seasons. Autumn displays relatively stable water levels, evidenced by a low standard deviation, indicating less variability and fewer extreme fluctuations. In contrast, spring exhibits increased variability.

Season	mean	std	min	25%	50%	75%	max
Autumn	134,17	0,26	132,95	134,03	134,15	134,35	135,66
Spring	132,55	0,88	131,43	131,9	132,18	133,14	134,82

Summer	134,04	0,39	132,35	133,95	134,08	134,21	136,07
Winter	133,27	0,61	132,03	132,77	133,29	133,74	134,5

Table 4-15 Seasonal Statistics Randsfjorden

The histogram analysis highlights significant seasonal variance in water levels. Spring and summer show elevated water levels reaching into higher flood-risk categories, with spring having 12 and summer 33 total flood days, respectively.

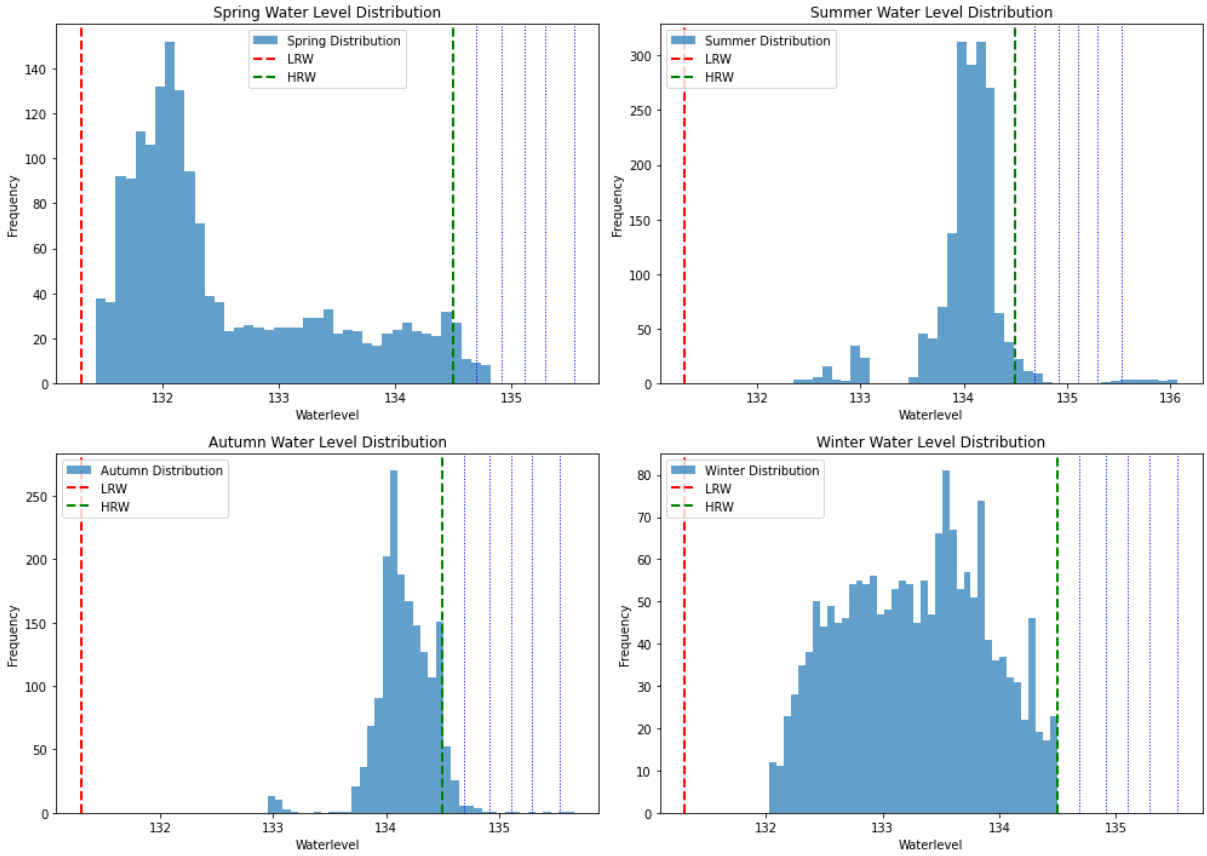


Figure 4-14 Seasonal Histograms Randsfjorden

Autumn's distribution, with negative skewness, suggests a tail of lower water levels, while the high kurtosis indicates a peaked distribution with potential for extreme high-water levels. Winter's symmetric distribution, with fewer outliers, aligns with no recorded flood days, reflecting stable water levels during this season.

Time-Series and Seasonal Fluctuations

The decomposed time-series analysis of Randsfjord waterlevels indicates a cyclical seasonal pattern with noticeable peaks, reflecting substantial fluctuations driven by environmental and climatic influences.

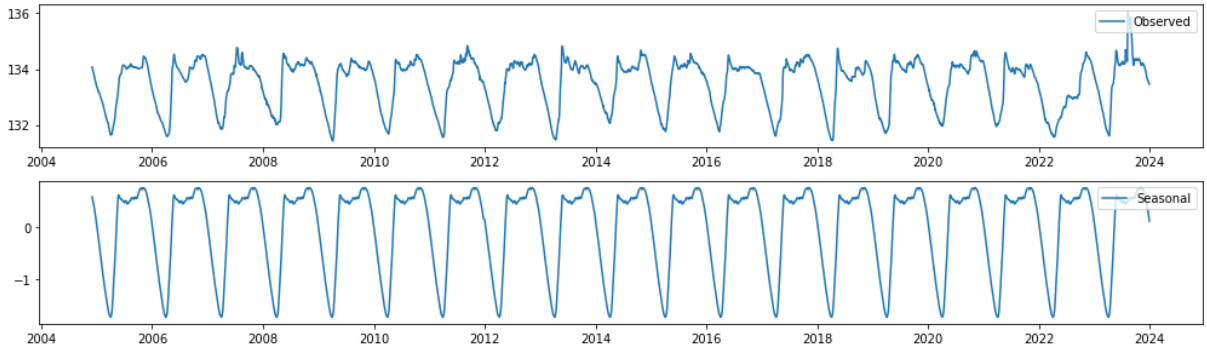


Figure 4-15 Time-Series and Seasonal Fluctuations Randsfjorden

The regularity in the seasonal component suggests that the lake's response to seasonal changes is consistent over the years. This pattern underscores the predictable nature of seasonal variations.

Flood Incidence

As with the two other reservoirs, Randsfjorden experience flooding in 3 out of 4 seasons. With summer with the highest frequency of flood.

Waterlevel	Spring	Summer	Autumn	Winter
Below 131.3	0	0	0	0
131.3 to 134.5	1702	1686	1608	1757
134.5 to 134.689	34	29	83	0
134.689 to 134.9159	12	10	12	0
134.9159 to 135.1058	0	0	2	0
135.1058 to 135.2902	0	0	2	0
135.2902 to 135.5321	0	4	1	0
Above 135.5321	0	19	2	0
Total Flood Days	12	33	19	0

Table 4-16 Frequency Randsfjorden

Seasonal Trend

The trend analysis reveals distinct patterns in water level changes across different seasons. Spring shows a significant increase in water levels

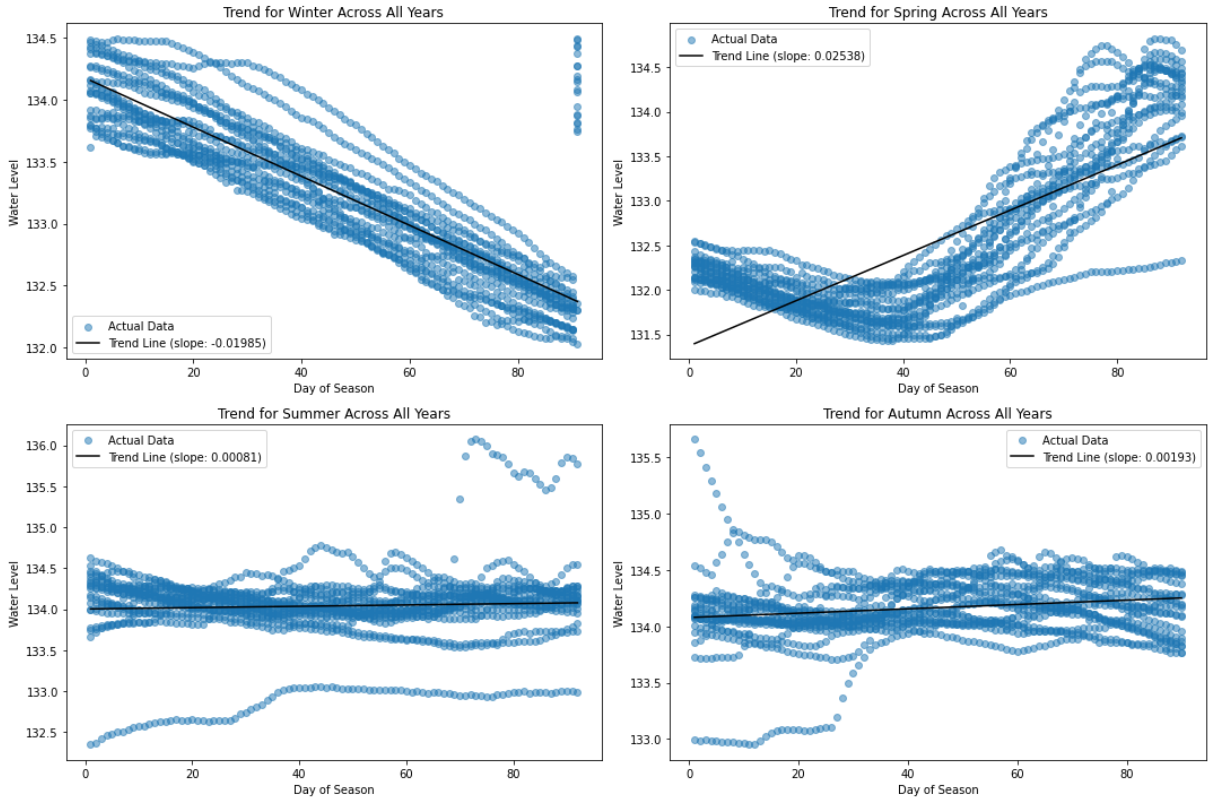


Figure 4-16 Seasonal Trend Randsfjorden

Winter shows a remarkable decline in waterlevels before going on to the increase in spring.

Multimodal Analysis

The multimodal analysis of water levels in three lakes, Tyrifjorden, Sperillen, and Randsfjord, was conducted to understand the impact of seasonal variations. Using histograms, Kernel Density Estimates (KDE), and the Kruskal-Wallis test, the analysis identified distinct seasonal modes in the water level data. The python program used for the multimodal analysis is *multimodal_analysis.py*, appendix 14. The complete multimodal analysis can be found in appendix 12.

In Tyrifjorden, the histograms and KDE plots showed varied peaks and distributions for different seasons, suggesting distinct modes. Monthly averages revealed water levels were lowest in early spring, peaked in June, and slightly declined towards autumn. The Kruskal-Wallis test, with an H-statistic of 1074.35 and a p-value near zero, confirmed significant differences across seasons, supporting the multimodal distribution due to seasonal variations.

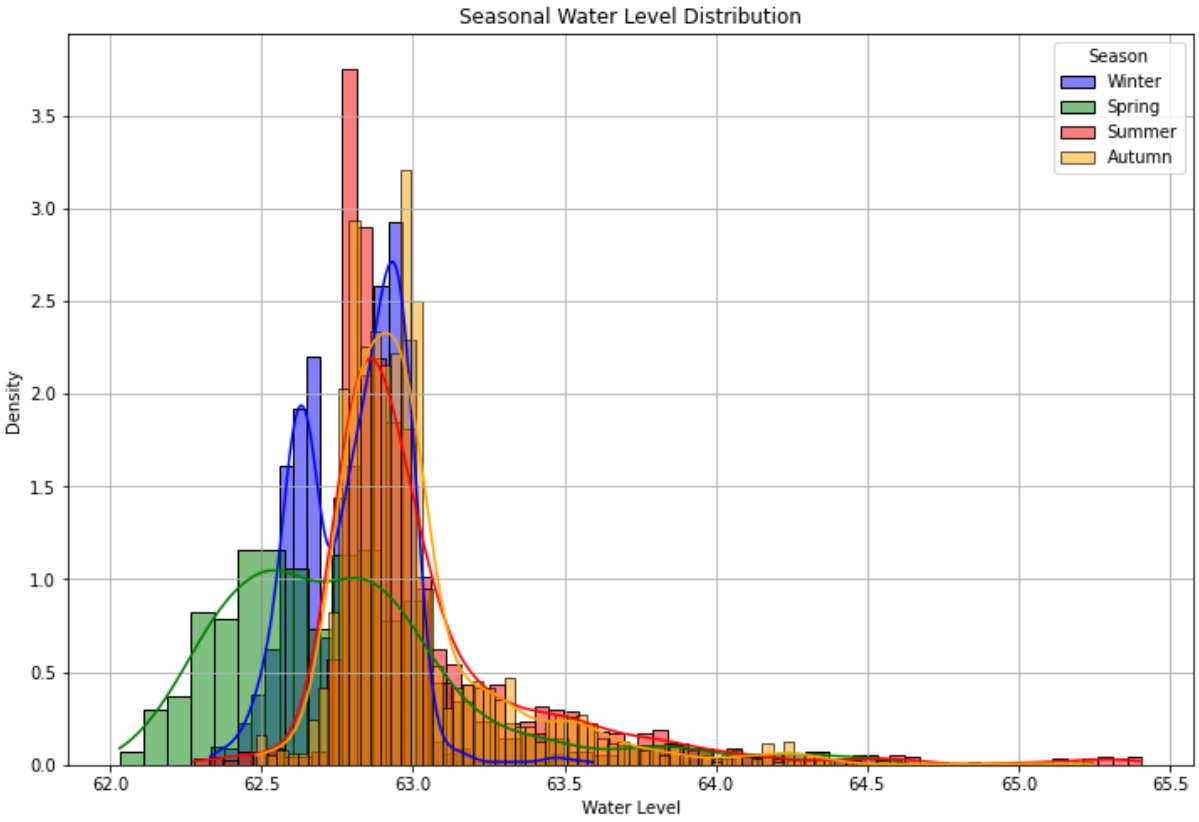


Figure 4-17 Multimodal Histogram Tyrifjorden

Sperillens water levels displayed similar seasonal fluctuations. Histograms and KDE plots indicated multiple modes corresponding to different times of the year. Monthly averages showed levels rising in March, peaking in May and June, and remaining high until November. The Kruskal-Wallis test, with an H-statistic of 1757s, confirmed significant seasonal differences, reinforcing the multimodal nature of the data due to seasonal impacts.

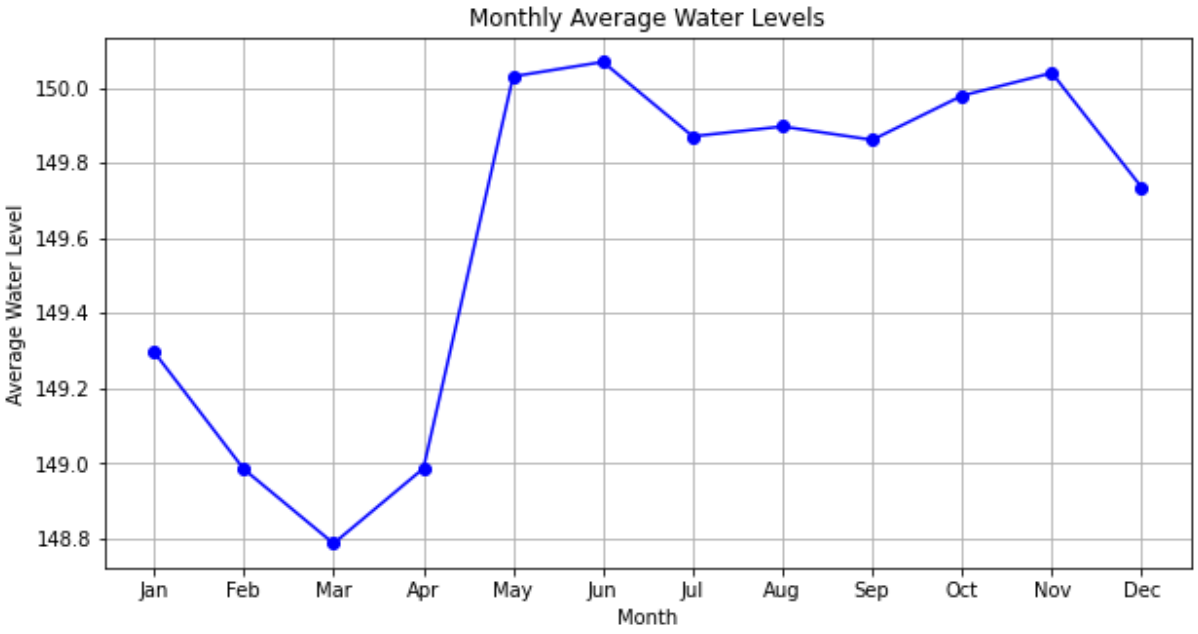


Figure 4-18 Monthly Averages Sperillen

Randsfjord exhibited distinct water level distributions for each season. Histograms and KDE plots indicated multiple modes, with levels rising dramatically from March to May, peaking in early summer, and stabilizing until a slight drop in December. The Kruskal-Wallis test, with an H-statistic of 3712.69 and a p-value of 0.0, confirmed significant seasonal differences, supporting the multimodal distribution driven by distinct environmental factors.

In conclusion, the multimodal analysis demonstrated that seasonal variations significantly influence water levels in Tyrifjorden, Sperillen, and Randsfjorden. This understanding is crucial for effective water resource management and risk assessment, especially in anticipating seasonal water availability and addressing potential flooding or drought conditions.

Summary of Exploratory Data Analysis

The exploratory data analysis (EDA) conducted for Tyrifjorden, Sperillen, and Randsfjorden reveals critical insights essential for the development of the decision-support framework. The correlation analysis confirmed a strong positive correlation between water levels and reservoir levels across all three lakes. This validation allowed the focus to remain solely on water level data for further analysis.

Initial data cleaning ensured the removal of inconsistencies and alignment of time series, resulting in datasets free from zero-values and suitable for robust analysis.
In Tyrifjorden, the mean water level is approximately 62.92 meters, with a standard deviation of 0.38 meters. This lake shows moderate variability with a pronounced central tendency around the mean. Seasonal fluctuations are notable, particularly in spring and summer, with occasional spikes exceeding flooding thresholds.

Sperillen has a mean water level of 149.63 meters and a standard deviation of 0.71 meters, indicating moderate variability. The water levels cluster significantly around the mean, with higher variability observed in spring and summer due to snowmelt and precipitation.
Randsfjordens mean water level is 133.50 meters, with a standard deviation of 0.87 meters. The data reflects moderate fluctuations around the mean, with a highly predictable range. The waterlevels are tightly regulated, and extreme values are infrequent. Seasonal analysis further demonstrates the hydrological dynamics of each lake. In Tyrifjorden, autumn presents the highest mean water level with stable conditions, while spring shows the lowest mean but the highest variability due to snowmelt and rainfall. Summer exhibits variability comparable to autumn, and winter shows the lowest variability, indicating consistent conditions.
Sperillens seasonal data reveals stable water levels in autumn with modest variability, while spring displays the highest variability driven by transitional weather patterns. Summer continues this trend with high water levels and increased variability, whereas winter is marked by the lowest variability and stable conditions.
Randsfjordens seasonal analysis highlights stable water levels in autumn, with low variability. Spring demonstrates increased variability and higher water levels, summer shows moderate spread and elevated flood risk, and winter remains stable with minimal extreme events.

The multimodal analysis, employing Kernel Density Estimates (KDE), identified distinct seasonal modes in the water level data for all three lakes. Significant seasonal differences were confirmed, highlighting the influence of seasonal impacts on water resource management.

In conclusion, the EDA provides a comprehensive understanding of the hydrological stability and variability of Tyrifjorden, Sperillen, and Randsfjorden. Each lake exhibits unique characteristics influenced by seasonal changes. These insights will inform the development of

a robust decision-support framework to enhance the management of water resources in Drammensvassdraget, aligning with the research objectives and scope.

5. Methodology

This chapter outlines the method used to develop the framework for risk-based decisions for managing water in Tyrifjorden, Randsfjorden and Sperillen. As mentioned, the primary goal is to add simplicity to the balancing of electricity generation and flood risk management by leveraging historical waterlevel data. This approach leans on the analysis already conducted in the former chapters. This chapter will outline the development and analysis of the performance.

Understanding the decision-support framework.

The framework developed in this study is a decision-support tool designed to convert historical quantitative data into a risk score for either flood or energy shortage. It focuses on historical data and statistical methods to provide a dimensionless risk score based on the current water level. The model is intended to support decision-making by quantifying the risk, but it is essential to understand its limitations and scope.

Before any development can be done it is imperative to choose what it should be able to do, and what is it not able to do. The question arises then to the design of a formula or a model. This is an important choice, given the advantages and constraints of both. Below is a table outlining the differences.

Aspect	**Decision Formula**	**Decision Model**
Complexity	Simple, direct calculations	Complex, involves multiple variables and scenarios
Flexibility	Rigid, fixed relationships	Flexible, can adapt to changes and incorporate uncertainty
Scope	Limited to specific well-defined situations	Broad, used for complex and strategic decisions
Nature	Deterministic	Often probabilistic and analytical

Tools Used	Basic mathematical expressions	Advanced tools like decision trees, simulations and optimization

Figure 5-1 Decision Formula vs. Decision Model

This model is not cut in stone, as there is grey area in between. The framework to be developed will use this grey area. As mentioned, numerous times, the complexity will be kept to a minimum. The scope is also well-defined flooding and waterlevels. This leans the framework toward a Decision Formula. However, the analysis done in the EDA will be used. Therefore, there are probabilistic and analytical elements to it. The use of probability forces the study to acknowledge the uncertainty in both probability and analysis. Finally, due to the simplicity and the lack of prediction in this framework, the choice is to make a formula rather than a model. This will enable the decision-support to not be time-sensitive and abstain from prediction. Formulas enables further development, and it might be included in a model on later stages. The lack of prediction and probability in the three decisions to be made, increase, decrease or maintain outflow is the biggest reason for choosing a formula.

- **What the framework is:**
    - Decision-Support Tool: It aids decision-makers by translating historical water level data into a risk score.
    - Quantitative Focus: The model relies on historical data and statistical methods to provide risk assessments.
    - Non-Time binding Risk: The risk score provided is dimensionless and not tied to a specific timeline. It indicates the current level of risk without predicting the exact timing of a flood or water shortage.
- **What the framework is Not:**
    - Dynamic Predictor: The model does not predict future events or provide a dynamic forecast. It updates based on historical data but does not account for real-time changes or future conditions.
    - Definitive Decision-Maker: While it provides valuable risk quantification, it does not replace expert judgment or operational decisions. The model highlights the risk, but the final decisions should consider qualitative assessments and other operational factors and methods.

Understanding these aspects ensures that operators can effectively integrate it into their decision-making processes, recognizing its strengths and limitations.

Formula

The formula will be designed to give the operator a baseline understanding of the risks associated with the current waterlevel in the reservoir. The formula integrates several factors from the EDA. It will use seasonal historical waterlevel density, seasonal trends, reservoir capacity and regulatory constraints to adjust the risk scores. While there are several more statistics that can be included in the formula, the EDA gave a thorough insight into the distribution of waterlevel and the frequency of outliers. This will all be included through the density and the trends. Aswell, the seasonal trends will encompass more of the analysis. The factors let out will represent some of the uncertainty of the formula and will be discussed in the appropriate chapter.

The formula will be split in to, Energy Shortage Risk and Flood Risk. The general formulas will look like this:

Energy Shortage Risk (ESR):

$$ESR = Baseline\ ESR \times D_{Energy} \times C_{Energy} \times R_{Energy} \times S_{Energy}$$

Equation 5-1 Energy Shortage Risk Formula

Flood Risk (FR):

$$FR = Baseline\ FR \times D_{Flood} \times C_{Flood} \times R_{Flood} \times S_{Flood}$$

Equation 5-2 Flood Risk Formula

- **Components:**
    - Each of the components (H, C, R, S) represents a different adjustment factor:
        - **Baseline:** Initial estimate of risk based on observed waterlevel
        - **D:** Density Adjustment
        - **C:** Current Reservoir Capacity Adjustment
        - **R:** Regulatory Constraints Adjustment
        - **S:** Seasonal Trends Adjustment
- These components will be thoroughly explained in the model development chapter, detailing how each factor is derived and integrated into the final risk score.

The model will require two inputs, apart from which lake the operator is analyzing. This will be the observed waterlevel and which season is current. This will be the starting point for the python program. The formula will then use seasonal waterlevel data to acquire the necessary statistics and regulations.

The final framework will in general be as the figure below.

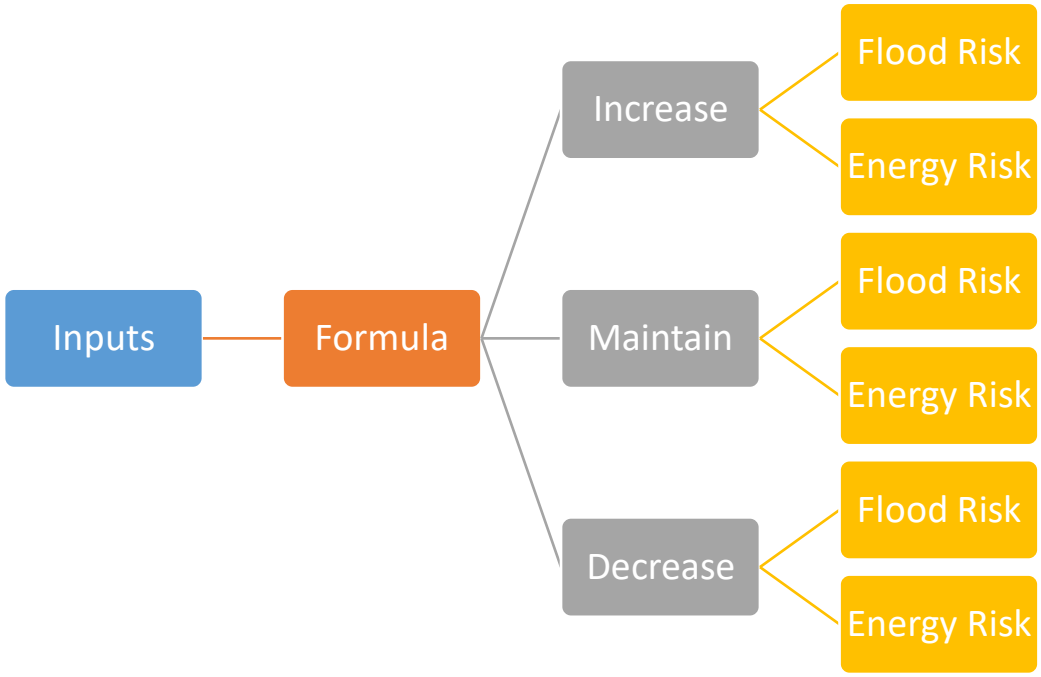


Figure 5-2 General Overview of Decision Framework

After the formula has calculated the risks at a certain waterlevel, a decision factor will be included to account for the adjustment based on what the operator will decide. This enables the operator to see what effect each decision will have on the risk.

Formula Preparation

For the formula python program to have the necessary data, several steps must be taken to prepare the datasets to work with a decision formula. The following python programs will be used to prepare for the formula:

- *Reservoir_to_DailyEnergy.py*
- *Data_Preperation.py*
- *States_constructor.py*

All these can be found in appendix.

Since the analysis has shown clear seasonality, the datasets will be categorized into seasons. This will ensure the formula only use waterlevels that have been present in the season that is current. Furthermore, waterflow, waterlevel, reservoir volume and energy has been combined into one set. The reason for this is the next step in the preparation, which is construction of waterlevel states. The reason for this is the probability density function which will be used in the density factors, this does not allow for one single point of density. As theory state, a continuous probability function does not have single definite probabilities. Only ranges. These ranges are the states. The states are constructed with percentiles, LRW, HRW, etc. The list can be seen in the picture below:

```
if name == 'Randsfjord':
    water_levels = [
        130.5718,  # Extended Low Water
        131.51356985708998,  # Extended Low Energy
        131.43,  # Low Observed Water
        131.6159,  # Low Observed Energy
        131.6623,  # Extended 1st percentile Energy
        131.68,  # 1st percentile Energy
        131.8302,  # Extended 5th percentile Energy
        131.88,  # 5th percentile Energy
        132.0365,  # Extended 10th percentile Energy
        132.08,  # 10th percentile Energy
        132.7427,  # Extended 25th percentile Energy
        132.85,  # 25th percentile Energy
        133.5323,  # Mean Energy
        133.5048011787279,  # Mean Water
        134.5,  # HRV
        134.689,  # Mean Flood
        134.9159,  # 5-Year Flood
        135.1058,  # 10-Year Flood
        135.2902,  # 20-Year Flood
        135.5321,  # 50-Year Flood
        136.07,  # High Observed Water
        136.9281,  # Extended High Water
    ]
```

Figure 5-3 State Limits for Randsfjord

These points are used to construct the states. However, the states above mean-flood are merged. Since the formula does not account for the flooding levels above mean flood. Given that if the waterlevel surpasses mean flood the reservoir is in a flooding state.

State	Lower Bound	Upper Bound
State 0	147,431	148,1312
State 1	148,1312	148,3988
State 2	148,3988	148,6157
State 3	148,6157	148,7402
State 4	148,7402	148,9442
State 5	148,9442	149,6315
State 6	149,6315	150,25
State 7	150,25	151,1276
State 8	151,1276	162,4594

Table 5-1 Sperillen Waterlevel States

Table 5-1 shows the states that have been defined. There will always be small fluctuations in the waterlevel, this is another positive point for the definition of states to be more useful. Figure 5-4 shows the state 6 highlighted and the densities on either side. These are the densities that will be used in that factor of the formula.

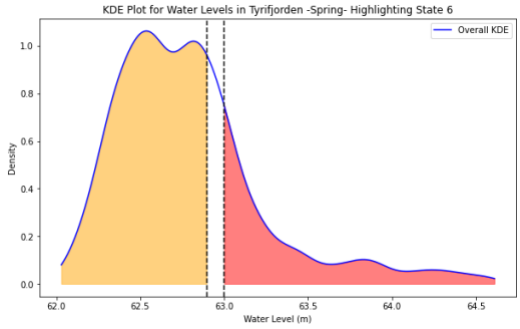


Figure 5-4 Example of States and Densities

To sum up, this methodology chapter has outlined the approach taken to develop a decision-support formula for water resource management in the Drammensvassdraget region. By leveraging historical data and statistical analysis, the development and formula aims to provide a tool for supporting the operators in the balancing of electricity generation and flood risk. The subsequent chapters will discuss the application of the model, its analysis, and the implications of the findings in detail.

6. Development and Design of the Formula

The formula is scripted in python, there are two python programs used for calculating the formula, *Single_Decision.py* and *Decision_for_loop.py*. The last one is made mostly for the analysis, which iterates over each waterlevel with a set increment. This enables a solid analysis of the formula's performance and limitations.

Before the risk values can be calculated, all the factors need to be produced. As shown before the two formulas and their factors looks like this. Equation 5-1 and 5-2.

Energy Shortage Risk (ESR):

$$ESR = Baseline\ ESR \times D_{Energy} \times C_{Energy} \times R_{Energy} \times S_{Energy}$$

Flood Risk (FR):

$$FR = Baseline\ FR \times D_{Flood} \times C_{Flood} \times R_{Flood} \times S_{Flood}$$

Historical and Extended Density Adjustment (D)

The historical density adjustment factor uses Kernel Density Estimation to smooth historical data and consider unobserved events. It combines historical and extended data; the extended data is made to provide a lower low and higher high. This combination ensures a comprehensive coverage of the complete range of waterlevels. The KDE will use a bandwidth of 0,2, this is a qualitative judgement, but provides enough smoothing to encompass unobserved events. While not destroying the integrity of the dataset. This component of the formula uses two python programs: *Historic_Risk_Factor.py* and *Extended_Risk_Factor.py*, appendix 18 and 19.

The extended data is generated through simulation. The start is by calculating the minimum and maximum observed waterlevel, and the standard deviation. This python program will then generate a set of synthetic waterlevels below minimum and above maximum. These values are uniformly distributed between:

$$[(Min\ Waterlevel - (3 \times Standard\ Deviations)) \quad , \quad Min\ \ Waterlevel]$$

$$[Max\ Waterlevel \quad , \quad (Max\ Waterlevel + (3 \times Standard\ Deviation))]$$

Equation 6-1 Extension Range Waterlevels

These simulated datapoints are then added into the original dataset, thus extending the low and high of the historical dataset.

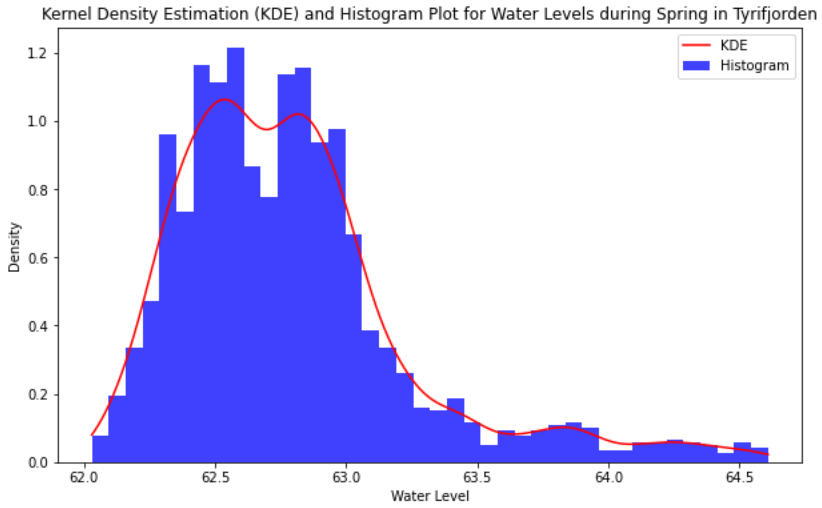


Figure 6-1 Histogram and KDE of Tyrifjorden Historic Waterlevels

The plot above shows the red line as the KDE, with bandwidth 0,2, for the historical waterlevels. The plot below shows the KDE for the extended waterlevels, here it is obvious that the waterlevels have lower and higher observations.

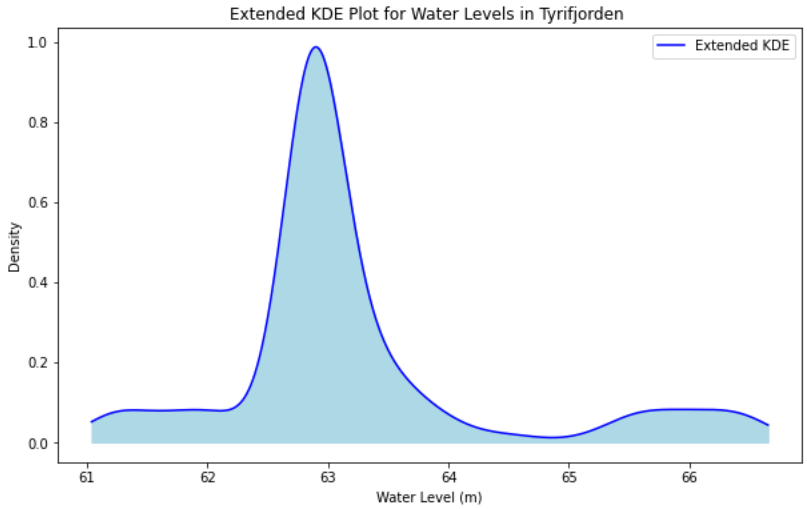


Figure 6-2 KDE plot for Extended waterlevels Tyrifjorden

The next table shows the product of the two python programs. The historic and extended densities for the current season are shown. The extension alters the densities in a way that the formula will account for possible unobserved events below and above highest and lowest historic.

			Historic		Extended	
State	Lower Bound	Upper Bound	Energy Density	Flood Density	Energy Density	Flood Density
State 0	61,6587	62,02999	0	0,993353	0,046527	0,906176
State 1	62,02999	62,1965	0	0,958392	0,076414	0,892932
State 2	62,1965	62,3788	0,034961	0,840824	0,089658	0,873002
State 3	62,3788	62,5273	0,152529	0,693116	0,109588	0,836318
State 4	62,5273	62,7795	0,300237	0,438424	0,146273	0,676999
State 5	62,7795	62,8956	0,554928	0,32169	0,305591	0,566333
State 6	62,8956	63	0,671663	0,231252	0,416257	0,46545
State 7	63	64,2	0,7621	0,018614	0,51714	0,122296
State 8	64,2	69,06785	0,974739	0	0,860294	0

Table 6-1 Densities Tyrifjorden

Since the formula is using two densities, and the extended has the sole purpose of extending the data outside of the original range there were chosen weights for the historical and the extended.

$$Density\ (Current\ state) = w_1 \times Density_{Historical} + w_2 \times Density_{Extended}$$

Equation 6-2 Density (Current State) Formula

Where:

- w_1 and w_2 are weights for the historical and extended, respectively.

The research will put a definite weight to both. The density factor for the formula will look like the formula below.

$$Density\ Adjustment\ Factor\ (H) = 1 + Density\ (Current\ State)$$

Equation 6-3 Density Adjustment Factor (H)

Current Reservoir Capacity (C)

The reservoir capacity is used to further incur a penalty if the waterlevel reach the lower part of the regulated zone, or the higher part of the regulated zone. This is to ensure the risks are properly shown when the waterlevel is at the ends of the regulated zone, between LRW and HRW.

This factor will represent the impact of the current reservoir capacity. This factor uses normalized reservoir levels, reservoir level are in cubic meters.

$$Normalized\ Reservoir\ Level = \frac{Current\ Reservoir\ Level}{Maximum\ Reservoir\ Capacity}$$

Equation 6-4 Normalized Reservoir Level

The maximum is calculated as the reservoir max at mean flood. The regulated zone is between LRW and HRW, the mean flood is used to further penalize a higher waterlevel than HRW.

The penalty will incur from L on the lower range, and from H at the higher range. The capacity adjustment is determined as:

$$Capacity\ Factor_{Energy} = 1 + \alpha \times (L - Normalized\ Reservoir\ Level)$$
$$Capacity\ Factor_{Flood} = 1 + \beta \times (Normalized\ Reservoir\ Level - H)$$

Equation 6-5 Capacity Factors (C)

Where:

- α and β are the scaling factors chosen.
- L and H are thresholds chosen to where the penalties will incur.

Regulatory Constraints (R)

There are several regulatory constraints that affect the risk. Regulatory constraints are made as mitigating measures to have a secure supply of energy, and low risk of severe flood. This factor will impose penalties if these thresholds are broken, or the waterlevel is closing in on them.

These thresholds are if the waterlevel goes above HRW there must be maximum output on the outflow. This will be a risk reducing act above HRW. Also, the rivers flowing out, Begna,

Randselva and Drammensvassdraget are not allowed to be dry, meaning that some outflow must always be on.

The regulatory factor is defined using various thresholds and zones.

$$Regulation\ Zone\ (RZ) = HRW - LRW$$
$$Pre\ Threshold\ (PT) = 0{,}2$$
$$Lower\ Threshold\ (LT) = LRW + PT \times RZ$$
$$Upper\ Threshold\ (UT) = HRW - PT \times RZ$$

Equation 6-6 Regulatory Thresholds and Zones

The upper threshold is where there is an imposed penalty for closing in on the HRW, when the HRW is passed the reservoir must have maximum outflow, regardless of energy need or weather. This is risk reducing from a statistical standpoint, but from a decision standpoint it might not be the optimal choice. For that reason, a penalty is imposed up until HRW, after that a flood risk reduction will be seen due to the maximum outflow.

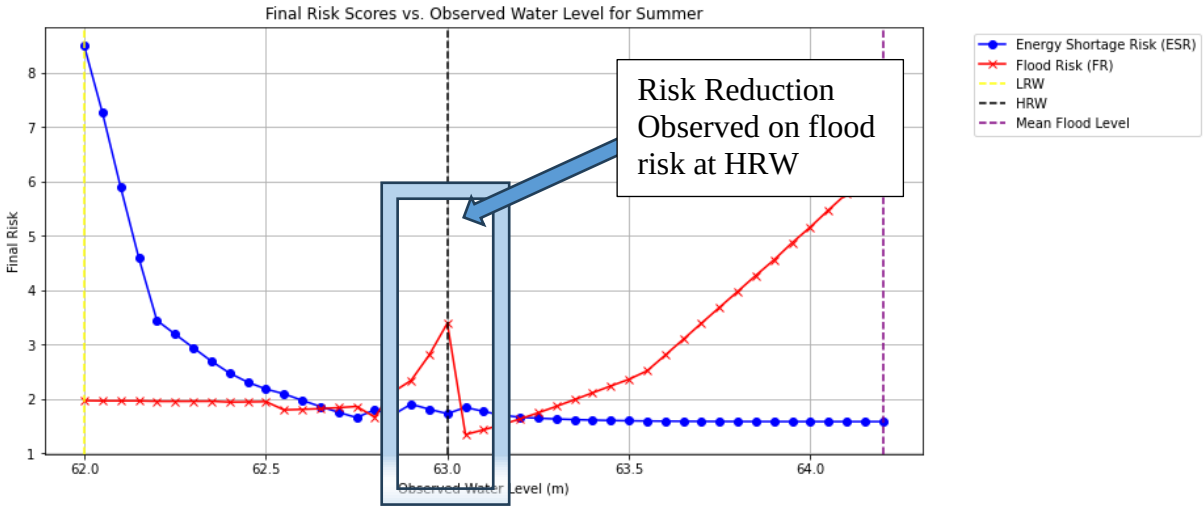


Figure 6-3 Example of Risk Reduction After HRW

Season Factor (S)

Given the big impact from seasons a seasonal factor is implemented. This is ensuring adjustments for variations in seasons, including deviation, trends and volatility.

$$S_{flood} = 1 + k \times (Seasonal\ Deviation + Seasonal\ Trend + Seasonal\ Volatility)$$
$$S_{energy} = 1 + k \times (Seasonal\ Volatility - Seasonal\ Trend - Seasonal\ Deviation)$$

Equation 6-7 Seasonal Factors (S)

Where:

- k is a scaling factor for seasonal adjustments

The seasonal adjustment factor (k) is to adjust the seasonal to a value that fits the other factors in the formula. For a seasonal adjustment of 1 the seasonality is the biggest contributor to the final risk formula. Although seasonality is a big factor, the adjustment is to have the possibility to adjust it down. For the formula at this stage, it is set to 0,8.
Below are the formulas for Seasonal Deviation and Volatility. The trend is calculated using python, and the numbers will be the average start and end waterlevel over the dataset in the current season.

$$Seasonal\ Deviation = \frac{Observed\ Waterlevel - Seasonal\ Average}{Seasonal\ Standard\ Deviation_{Mean}}$$

Equation 6-8 Seasonal Deviation

$$Seasonal\ Volatility = Seasonal\ Standard\ Deviation_{Mean}$$

Equation 6-9 Seasonal Volatility

Baseline Flood and Energy Shortage Score and Final Risk Scores

The baseline score are the initial estimates for flood risk (FR) and energy shortage risk (ESR) based on the observed waterlevel. It is an exponential formula developed through trial to get a meaningful baseline for the risks at a certain observed waterlevel.

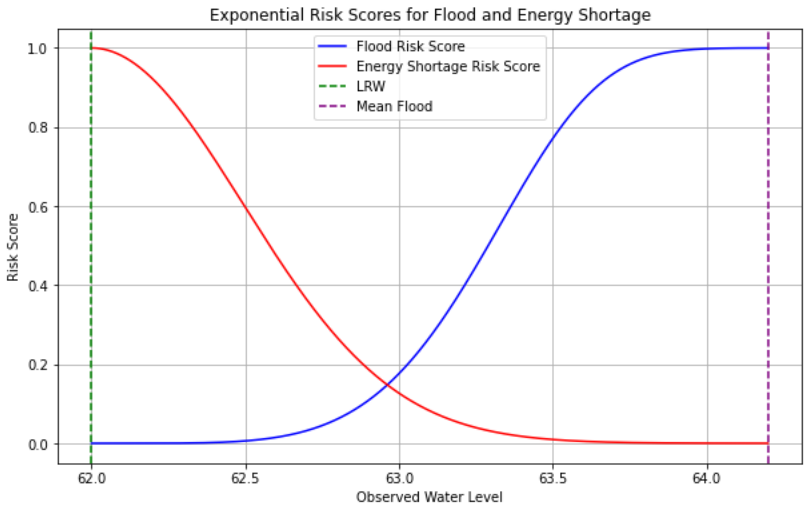


Figure 6-4 Example Baseline Flood and Energy Shortage Risk

The formula is made with certain conditions, it is based on the observed waterlevel (OWL), lowest regulated waterlevel (LRW) and mean flood level for the current reservoir (MF).

Condition 1: Observed Waterlevel below Lowest Regulated Waterlevel

$$if\ OWL < LRW : Energy\ Score = 1$$

Equation 6-10 Condition 1 Baseline Risks

Condition 2: Observed Waterlevel above Mean Flood Level

$$if\ OWL > MF : Flood\ Score = 1$$

Equation 6-11 Condition 2 Baseline Risks

Between LRW and MF the scores will be calculated using exponential formulas and a normalized waterlevel between 0 and 1.

$$Normalized\ Waterlevel = \frac{OWL - LRW}{MF - LRW}$$

Equation 6-12 Normalized Waterlevel

$$Flood\ Score = 1 - \ e^{-10 * \ normalized\ water\ level^5}$$

Equation 6-13 Baseline flood score

$$Energy\ Score = \ e^{-10 * \ normalized\ water\ level^2}$$

Equation 6-14 Baseline energy score

The calculation is done in python, to automate and iterate over multiple waterlevels.

```python
# Define the function to calculate risk scores with adjusted exponential scalings
def calculate_risk_scores(observed_waterlevel, LRW, mean_flood):
    # Initialize scores
    flood_score = 0
    energy_score = 0

    if observed_waterlevel < LRW:
        # Maximum water shortage risk when below LRW
        energy_score = 1  # Max water shortage risk
    elif observed_waterlevel > mean_flood:
        # Maximum flood risk when above Mean Flood
        flood_score = 1  # Max flood risk
    else:
        # Between LRW and Mean Flood: separate exponential scaling of risks
        normalized_level = (observed_waterlevel - LRW) / (mean_flood - LRW)


        flood_score = 1 - np.exp(-10 * (normalized_level**5))
        energy_score =  np.exp(-10 * (normalized_level**2))
    return flood_score, energy_score
```

Figure 6-5 Python print Baseline Risk Scores

The complete baseline ESR and FR then becomes:

$$FR = \begin{cases} 0 & OWL < LRW \\ 1 & OWL > MF \\ 1 - e^{-10 * normalized\ waterlevel^5} & LRW \leq OWL \leq MF \end{cases}$$

Figure 6-6 Baseline Flood Risk Compete

$$ESR = \begin{cases} 1 & OWL < LRW \\ 0 & OWL > MF \\ 1 - e^{-10 * normalized\ waterlevel^2} & LRW \leq OWL \leq MF \end{cases}$$

Figure 6-7 Baseline Energy Shortage Risk Complete

Decision Factor

The last part of the formula, after the final risk scores have been calculated using the formula. Is to account for the possible decisions the operator will have. This is where the decision factor comes in. These factors adjust the scores based on the impact of increasing, decreasing or maintaining the outflow.

$$ESR = Baseline\ ESR \times D_{Energy} \times C_{Energy} \times R_{Energy} \times S_{Energy}$$
$$ESR_{Final} = Decision\ Factor_{Energy} \times\ ESR$$

Equation 6-15 Final Energy Shortage Risk

$$FR = Baseline\ FR \times D_{Flood} \times C_{Flood} \times R_{Flood} \times S_{Flood}$$

$$FR_{Final} = Decision\ Factor_{Flood} \times FR$$

Equation 6-16 Final Flood Risk

The decision factor can take three values, increase, decrease or maintain outflow. Each of these will alter the risks from the formula.

```
# Define decision factors
ESR_increase = 1.2  # Increase outflow gives a 20% increase in Energy Shortage Risk
ESR_decrease = 0.8  # Decrease outflow gives 20% decrease in Energy Shortage Risk
ESR_maintain = 1  # Maintain outflow gives no change in final risk

FR_increase = 0.8  # Increase outflow gives a 20% decrease in Flood Risk
FR_decrease = 1.2  # Decrease outflow gives a 20% increase in Flood Risk
FR_maintain = 1  # Maintain outflow gives no change in final risk
```

Figure 6-8 Decision Factor Value

This factor will then multiply by the factor corresponding to the decision of interest. This enables the operator to look at all possible actions before moving on to other evaluations. However, after HRW is passed regulation demands max outflow, so the possibility of increase is removed after HRW.

The multiplication factors are based on this:

1. **Increase Outflow**:
   - Adjusts risks when outflow is increased
   - Typically decreases flood risk and increases energy shortage risk

2. **Decrease Outflow:**
   - Adjusts risks when outflow is decreased
   - Typically increases flood risk and decreases shortage risk

3. **Maintain Current Outflow:**
   - No change in the final risk scores.

Below is a printout of the python program Single_Decision.py. This is the entire calculation for Tyrifjorden at 62,99 meters during spring. A possible scaling of the factors, for example between 0 and 1 will be addressed in the coming chapters.

```
wdir= /Users/simen/Desktop/Complete Master/02 Python/03 Por
Observed Waterlevel : 62.99

Baseline Flood Risk:
1.1685034462700368

Baseline Energy Shortage Risk:
1.1319938431878298

Density Adjustment Factors (H):
Energy Density Adjustment : 1.6549818439996866
Flood Density Adjustment : 1.2541100078154495

Capacity Factor (C):
Flood Capacity Factor: 1
Energy Capacity Factor: 1

Regulatory Constraints Factor
Flood Penalty Factor for Tyrifjorden: 1.9500000000000106
Energy Penalty Factor for Tyrifjorden: 1

Seasonal Adjustment Factor for Flood (S_flood):
1.3901454190060676
Seasonal Adjustment Factor for Energy (S_energy):
1.3825146358540867

Final Scores
Final ESR, before decision factor : 2.5900433684157456
Final FR, before decision factor : 3.9724686213650817

Decision Risk Score
Observed Waterlevel : 62.99m in Spring
Increase:
Energy Shortage Rrisk : 3.1081
Flood Risk : 3.1780
Decrease:
Energy Shortage Risk : 2.0720
Flood Risk : 4.7670
Maintain:
Energy Shortage Risk : 2.5900
Flood Risk : 3.9725
```

Figure 6-9 Printout from Tyrifjorden Final Risk

Priming the variables

There are several flexible values that can be altered to enhance the presentation of the decisions. This comes at a cost; the approach must make sure that the data and presentation is not compromised. Also, the weights between the factors needs to be at a suitable level.

This part does not need a for-loop, but testing, trial and an assumption based on a qualitative assessment. For example: at 64,1 meters in Tyrifjorden the flood risk should be quite high.

There is no definitive solution to how the factors should be "weighted". However, the research has shown strong seasonality, therefore the seasons should be quite high compared to the others. Also, history should be respected and therefore, density should not be neglected. Based on these two assumptions the following variables and weights were used for the model. These are subject for changing, if necessary, after sensitivity analysis, and will be discussed at the end of the research.

Factor	Variable	Value
Density (H)	Historic Density w_1	0,7
	Extended Density w_2	0,3
Capacity (C)	L	0,2
	H	0,7
	Alpha	2
	Beta	3
Seasonal (S)	Seasonal k	0,8

Table 6-2 Variables for formula

These variables will be used in the continuing analysis and the final decision support formula, chapter 8.

Test runs

This section will go through two runs of the program, one with a single decision and finally run the for-loop python program to see the distribution of the risk scores, normalized between 0 and 1.

Single_Decision.py

Inputs:

Reservoir: Tyrifjorden

Season: Spring

Observed Waterlevel: 62,95 meters

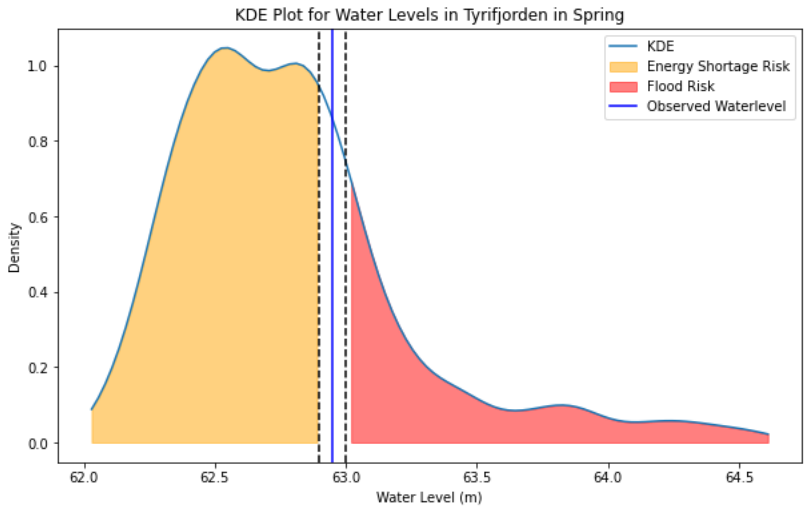
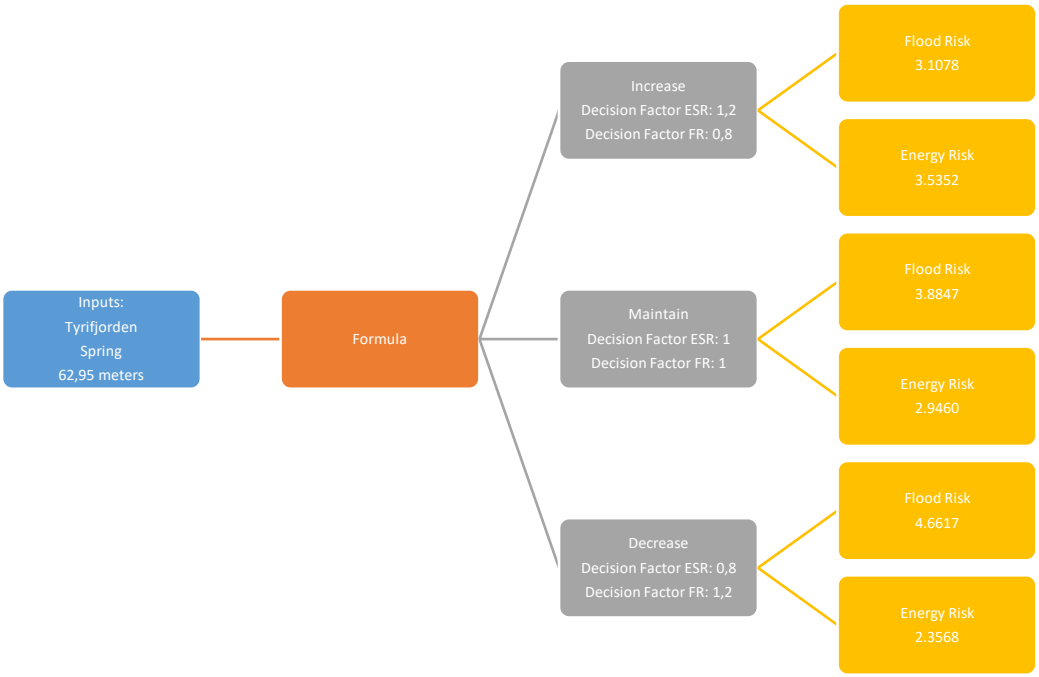


Figure 6-10 Test Run Single Decision

Observed Waterlevel	62.95
Baseline Flood Risk	1.1394149571039434
Baseline Energy Shortage Risk	1.154947423708174
Density Adjustment Factors (H)	
Energy Density Adjustment	1.6549818439996866
Flood Density Adjustment	1.2541100078154495
Capacity Factor (C)	
Flood Capacity Factor	1
Energy Capacity Factor	1
Regulatory Constraints Factor (R)	
Flood Penalty Factor for Tyrifjorden	1.7500000000000178
Energy Penalty Factor for Tyrifjorden	1
Seasonal Adjustment (S)	
Seasonal Adjustment Factor for Flood	1.5534804302075078
Seasonal Adjustment Factor for Energy	1.5412711771643386
Final Scores	
Final ESR, before decision factor	2.946011955860864
Final FR, before decision factor	3.8847348799706047
Decision Risk Score	
Increase	
Energy Shortage Risk	3.5352
Flood Risk	3.1078
Decrease	
Energy Shortage Risk	2.3568
Flood Risk	4.6617
Maintain	
Energy Shortage Risk	2.9460
Flood Risk	3.8847

Table 6-3 Test Run Factors Table



While the risk values make sense magnitude wise in this presentation, the need for a scaling is apparent. The formula lacks the ability to represent "how high" the risk is, or low. This will be handled in the for-loop program. With a scaling between 0 and 1 on risks.

Decision_for_loop.py

In this script the risk values have been normalized between 0 and 1. 1 will then represent the highest risk. This script is designed to iterate over waterlevels and calculate the risk. This will then be visualized in different plots.

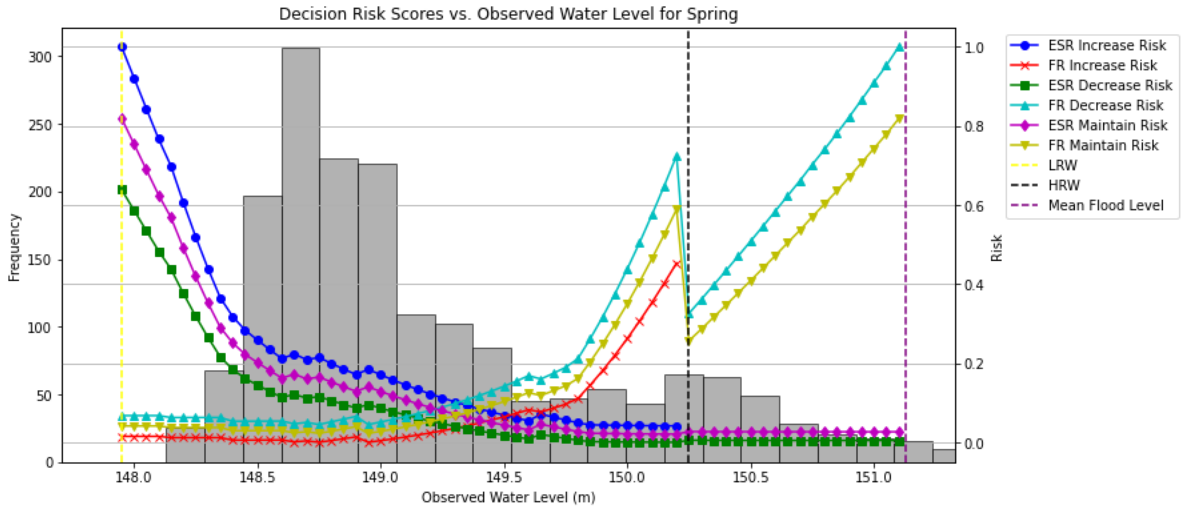


Figure 6-11 Complete Histogram and Risk Scores of Waterlevels

This is the most fulfilling plot the python program provides. The normalization has turned the values into a more suitable presentation, on the right y-axis. The script will also take one input of what the current water level is. This will then present the decision scores corresponding to that waterlevel.

```
# Find the row with the closest observed water level to the current water level
current_waterlevel = 149
closest_row = decision_scores_df.iloc[(decision_scores_df['Observed Water Level'] - current_waterlevel).abs().argmin()]

# Print the decision scores for the closest observed water level
print(f"Closest decision scores for observed water level {closest_row['Observed Water Level']}:")
print(closest_row)
```

Figure 6-12 Script example Complete Formula

```
Closest decision scores for observed water level 149.000
Observed Water Level    149.000000
ESR Increase Risk         0.170807
FR Increase Risk          0.004216
ESR Decrease Risk         0.086233
FR Decrease Risk          0.051321
ESR Maintain Risk         0.128520
FR Maintain Risk          0.027768
Name: 21, dtype: float64
```

Figure 6-13 Printout Decisions Factors Complete Formula

From the plot we can see that the regulation works quite efficiently, with the histogram in the background showing the most density in “low risk” territory.

Complete risk values can be found in appendix 22.

7. Sensitivity Analysis

In the context of water resource management, sensitivity analysis plays a crucial role in understanding how various factors influence the risk assessments of flood and energy shortages. This chapter delves into the sensitivity analysis of the decision-support framework, focusing on three key stages: seasonal adjustment factors, density adjustments, and global sensitivity analysis. The reason for choosing only these two factors is that the other is quite constant. Regulative changes are fairly set, as the mentioned Randsfjord adjustments have been going on since 1995, and the capacity is what it is.

The primary objective of this sensitivity analysis is to evaluate how changes in critical parameters impact the final risk scores for both flood and energy shortage scenarios. Aswell as the python program *Sensitivity_Analysis.py* in appendix.

Seasonal Adjustment

The plot provided shows the sensitivity analysis of risk factors focusing on seasonal adjustment factors for flood risk (FR) and energy shortage risk (ESR). Here is a detailed interpretation:

Axes

- X-Axis (Factor Value): Represents the varying values of the seasonal adjustment factors.
- Y-Axis (Average Risk): Represents the average risk values for flood and energy shortage.

Lines and Markers

- Blue Line (Average ESR - Seasonal Adjustment Factor for Flood): Shows how the average energy shortage risk changes with varying seasonal adjustment factors for flood.
- Orange Line (Average FR - Seasonal Adjustment Factor for Flood): Shows how the average flood risk changes with varying seasonal adjustment factors for flood.
- Green Line (Average ESR - Seasonal Adjustment Factor for Energy): Shows how the average energy shortage risk changes with varying seasonal adjustment factors for energy.
- Red Line (Average FR - Seasonal Adjustment Factor for Energy): Shows how the average flood risk changes with varying seasonal adjustment factors for energy.

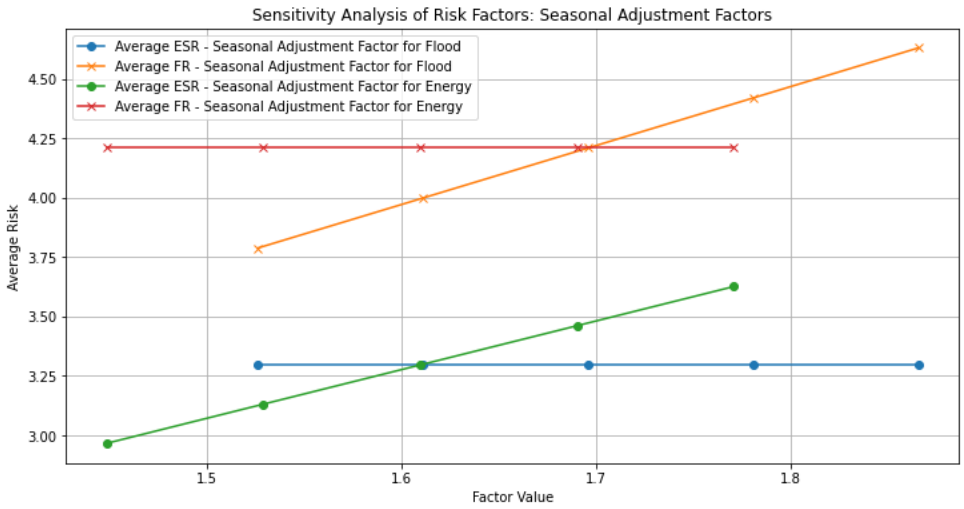


Figure 7-1 Sensitivity Analysis Seasonal Adjustment

Observations

The impact of the seasonal adjustment factor on flood risk, as represented by the orange line, shows a direct relationship where flood risk increases with an increase in the seasonal adjustment factor for flood. The steep slope of the orange line indicates that flood risk is highly sensitive to these changes.

In contrast, the impact on energy shortage risk, represented by the green line, also demonstrates a direct relationship with its respective seasonal adjustment factor. However, the green line's upward trend is less steep than the orange line, indicating a moderate sensitivity to changes in the seasonal adjustment factor for energy.

When examining the impact on flood risk for energy, depicted by the red line, the trend remains relatively flat. This flat trend suggests that the flood risk does not significantly vary with changes in the seasonal adjustment factor for energy, indicating low sensitivity.

Similarly, the blue line representing the impact on energy shortage risk for flood also shows a relatively flat trend. This indicates that energy shortage risk is not significantly influenced by variations in the seasonal adjustment factor for flood, suggesting low sensitivity.

In conclusion, flood risk is highly sensitive to changes in the seasonal adjustment factor for flood, while energy shortage risk shows moderate sensitivity to changes in the seasonal adjustment factor for energy. However, both flood risk and energy shortage risk are not significantly affected by changes in the seasonal adjustment factors for the other risk type.

Density Adjustment

This plot shows the sensitivity analysis of risk factors focusing on density adjustments for flood risk (FR) and energy shortage risk (ESR). A detailed interpretation:

Lines and Markers

- Blue Line (Average ESR - Flood Density Adjustment): Shows how the average energy shortage risk changes with varying density adjustment factors for flood.
- Orange Line (Average FR - Flood Density Adjustment): Shows how the average flood risk changes with varying density adjustment factors for flood.
- Green Line (Average ESR - Energy Density Adjustment): Shows how the average energy shortage risk changes with varying density adjustment factors for energy.
- Red Line (Average FR – Energy Density Adjustment): Shows how the average flood risk changes with varying density adjustment factors for energy.

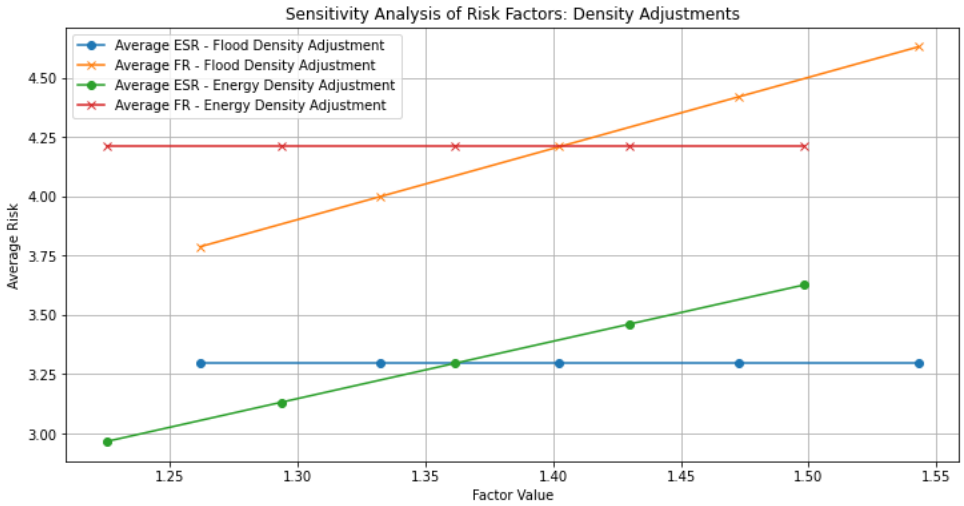


Figure 7-2 Sensitivity Analysis Density Adjustments

Observations

The impact of the density adjustment factor on flood risk, illustrated by the orange line, shows a direct relationship where flood risk increases as the density adjustment factor for flood rises. The steep slope of the orange line indicates that flood risk is highly sensitive to these changes.

In contrast, the effect on energy shortage risk, depicted by the green line, also reveals a direct relationship with its respective density adjustment factor. The green line shows an upward

trend, although it is less steep than the orange line, suggesting moderate sensitivity to changes in the density adjustment factor for energy.

Regarding the impact on flood risk for energy, represented by the red line, the trend remains relatively flat. This flat trend indicates that flood risk does not significantly change with varying density adjustment factors for energy, indicating low sensitivity.

Similarly, the blue line representing the impact on energy shortage risk for flood also shows a relatively flat trend. This suggests that energy shortage risk is not significantly influenced by changes in the density adjustment factor for flood, indicating low sensitivity.

In conclusion, flood risk is highly sensitive to changes in the density adjustment factor for flood, while energy shortage risk demonstrates moderate sensitivity to changes in the density adjustment factor for energy. Both flood risk and energy shortage risk are not significantly affected by changes in the density adjustment factors for the other risk type.

Global Sensitivity

This plot provides a global sensitivity analysis of the risk factors by varying multiple parameters simultaneously and displaying their impact on the average risk with standard deviation (Std Dev) error bars.

Axes

- X-Axis (Parameter Value): Represents the varying values of the parameters (seasonal adjustment factors and density adjustments).
- Y-Axis (Average Risk with Std Dev): Represents the average risk values for flood and energy shortage, along with the standard deviation.

Lines and Markers

- Blue Markers (ESR - Seasonal Adjustment Factor for Flood): Shows how the average energy shortage risk changes with varying seasonal adjustment factors for flood.
- Orange Markers (ESR - Seasonal Adjustment Factor for Energy): Shows how the average energy shortage risk changes with varying seasonal adjustment factors for energy.
- Green Markers (FR - Density Adjustment for Flood): Shows how the average flood risk changes with varying density adjustments for flood.

- Red Markers (FR – Density Adjustment for Energy): Shows how the average flood risk changes with varying density adjustments for energy.

Error Bars

- Error bars represent the standard deviation of the risk scores, indicating the variability in the risk assessments for each parameter value.

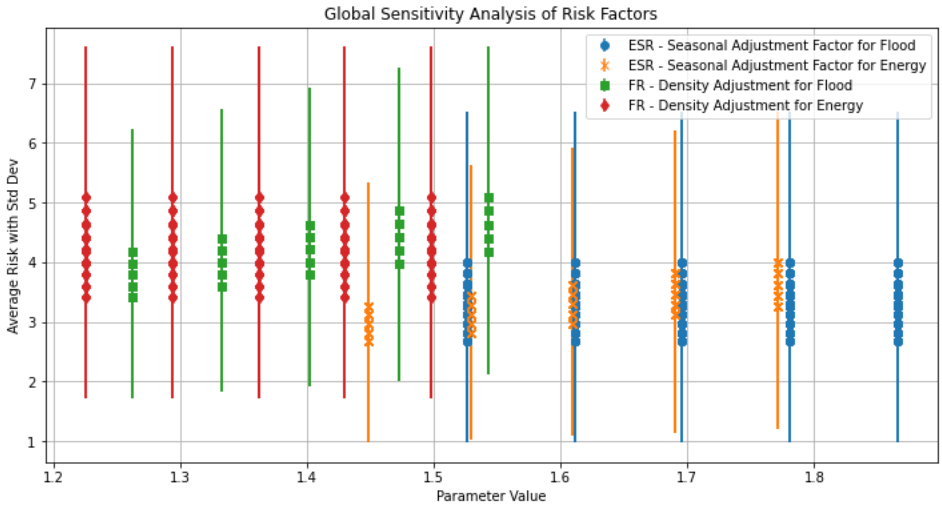


Figure 7-3 Global Sensitivity Analysis

Observations

The analysis of seasonal adjustment factors shows that the average flood risk, indicated by blue markers, varies significantly across different parameter values. This high variability and standard deviation suggest a high sensitivity and variability in flood risk. Conversely, the average energy shortage risk, shown by orange markers, exhibits moderate changes with increasing parameter values, and its variability is relatively low compared to flood risk.

In terms of density adjustments, the average flood risk, represented by green markers, significantly increases with parameter values. The error bars indicate substantial variability, implying high sensitivity. On the other hand, the average energy shortage risk, depicted by red markers, remains relatively constant across different parameter values, and its variability is lower compared to the density adjustment for flood.

The global sensitivity plot reveals that flood risk is generally more sensitive to changes in both seasonal adjustment factors and density adjustments compared to energy shortage risk. Energy shortage risk exhibits lower variability and sensitivity to these parameter changes.

In conclusion, flood risk is highly sensitive to changes in both seasonal adjustment factors and density adjustments, as indicated by the high variability and significant changes in average risk with different parameter values. In contrast, energy shortage risk is less sensitive to these parameters, particularly for density adjustments related to flood risk, with lower variability in risk scores indicating more stable risk assessments.

Key Findings

The sensitivity analysis reveals critical insights into the behavior of the risk model under different conditions:

- Flood Risk: Demonstrates high sensitivity to both seasonal adjustment factors and density adjustments, indicating a need for precise calibration in these areas.
- Energy Shortage Risk: Shows moderate sensitivity, particularly to seasonal factors, suggesting that energy risk assessments are relatively stable but still influenced by seasonal variations.

8. Final Decision-Support Formula

This chapter sum up the insights and methodologies developed in the previous chapters to present the final decision-support formula for managing water resources in the Drammensvassdraget region. The formula integrates various risk factors and adjustment parameters to provide a comprehensive tool for balancing electricity generation and flood risk management.

Summary of Development and Design

In Chapter 6, we detailed the creation and calibration of the decision-support formula. This formula was designed to convert historical water level data into risk scores for both flood and energy shortage scenarios. The key components of the formula include:

- **Historical and Extended Density Adjustment (D):** This component uses Kernel Density Estimation (KDE) to account for both observed and unobserved events, extending the range of historical data to cover extreme water levels.
- **Current Reservoir Capacity (C):** This factor adjusts risk scores based on the current reservoir level relative to its capacity, with penalties for levels near the lower and upper bounds of the regulated zone.
- **Regulatory Constraints (R):** This component imposes penalties based on regulatory thresholds, such as mandatory outflows when water levels exceed the highest regulated water level (HRW).
- **Seasonal Factors (S):** These factors account for seasonal variations in water levels, incorporating seasonal deviation, trends, and volatility into the risk assessments.

Final Risk Scores

The formula calculates two primary risk scores:

1. **Energy Shortage Risk (ESR):**

$$ESR = Baseline\ ESR\ \times\ D_{Energy} \times C_{Energy} \times R_{Energy} \times S_{Energy}$$

2. **Flood Risk (FR):**

$$FR = Baseline\ FR\ \times D_{Flood} \times C_{Flood} \times R_{Flood} \times S_{Flood}$$

Decision Factors

To provide actionable insights, the formula includes decision factors that adjust the risk scores based on potential management actions:

1. **Increase Outflow:** Typically decreases flood risk but increases energy shortage risk.
2. **Decrease Outflow:** Typically increases flood risk but decreases energy shortage risk.
3. **Maintain Current Outflow:** Maintains the current risk levels.

The final risk scores, incorporating decision factors, are given by:

$$ESR_{Final} = Decision\ Factor_{Energy} \times ESR$$
$$FR_{Final} = Decision\ Factor_{Flood} \times FR$$

Sensitivity Analysis

In Chapter 7, we conducted a comprehensive sensitivity analysis to evaluate the impact of different factors on the risk scores. The analysis included:

3. **Seasonal Adjustment Factors:** Examined the sensitivity of risk scores to variations in seasonal factors.
4. **Density Adjustments:** Assessed how changes in the density adjustment factors affect the risk scores.
5. **Global Sensitivity Analysis:** Evaluated the combined impact of varying multiple parameters simultaneously on the risk scores.

The sensitivity analysis demonstrated that flood risk is highly sensitive to changes in both seasonal adjustment factors and density adjustments, while energy shortage risk shows moderate sensitivity, particularly to seasonal factors.

Implementation and Application

The final decision-support formula provides a tool for managing water resources in the Drammensvassdraget region. By integrating historical data, regulatory constraints, and seasonal variations, the formula offers a quantitative basis for balancing electricity generation and flood risk. The decision factors further enhance its practical utility, allowing operators to

assess the impact of different management actions on risk levels. The formula culminates in the plot shown the final test run, from *Decision_for_loop.py*.

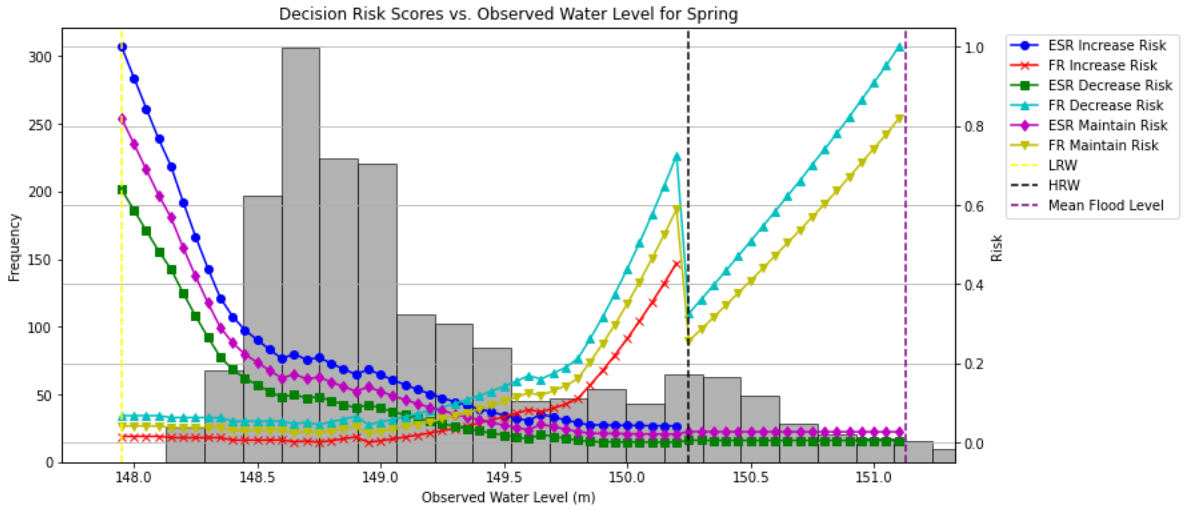


Figure 8-1 Final Decision Formula Result from for-loop

All seasons and all reservoirs can be found in appendix 22.

9. Summary and Discussion

The primary findings of this research highlight the challenges in water resource management and flood risk mitigation. The decision-support formula developed provides a valuable tool for managing water resources in the Drammensvassdraget region. By leveraging historical data and statistical analysis, the formula accurately quantifies flood and energy shortage risks, facilitating informed decision-making. The development of a dimensionless risk score based on historical water levels avoids the pitfalls of early-stage predictive guessing. Extensive sensitivity analyses demonstrated that flood risk is highly sensitive to seasonal and density adjustments, while energy shortage risk is moderately sensitive, particularly to seasonal factors. Furthermore, the formula's ability to integrate historical data, regulatory constraints, and seasonal variations provides a quantitative basis for balancing electricity generation and flood risk, with practical decision factors enhancing its applicability.

The formula's practical utility lies in its ability to help operators make informed decisions about water resource management by quantifying risks associated with both floods and energy shortages. Additionally, the inclusion of decision factors allows for adjustments based on potential management actions, enabling dynamic responses to changing water levels.

One of the key strengths of this study is the robust statistical analysis underlying the formula, ensuring reliability and robustness in risk assessment. Detailed sensitivity analyses provide insights into the formula's performance under various conditions, ensuring its reliability. However, the formula's reliance on historical data may limit its accuracy in scenarios where past patterns do not reflect future conditions. Additionally, the exclusion of immediate weather warnings means the formula does not account for real-time weather changes, making it less responsive to immediate risks. The timeless risk values also require expert interpretation and judgment, particularly under extreme conditions.

The sensitivity and robustness analysis conducted in this research provided crucial insights into the behavior of the decision-support formula under different conditions. The formula showed high sensitivity to seasonal adjustment factors, particularly for flood risk, highlighting the need for precise calibration of seasonal parameters to ensure accurate risk assessments. Changes in density adjustment factors significantly influenced flood risk scores, demonstrating the importance of incorporating density variations into the formula. The

formula's robustness was confirmed through various scenarios and parameter changes, ensuring its reliability in diverse conditions.

As this research reaches its conclusion, we reflect on the development of this robust decision-support formula for managing water resources in the Drammensvassdraget region. Designed to balance the objectives of electricity generation and flood risk mitigation.

The formula's cornerstone is its ability to accurately quantify flood and energy shortage risks by using historical data. This foundation ensures robust risk assessments grounded in past events, offering a dimensionless risk score that facilitates informed decision-making without predicting specific future occurrences. The formula's novelty is its biggest strength; it quantifies the current situation without trying to predict the future. This key distinction emphasized throughout the thesis allows the formula and its operator to use the risk values at their discretion. The use of densities is central, and apart from seasonal adjustments, the sensitivity analysis showed densities to be impactful. However, this reliance on densities can produce some intriguing results.

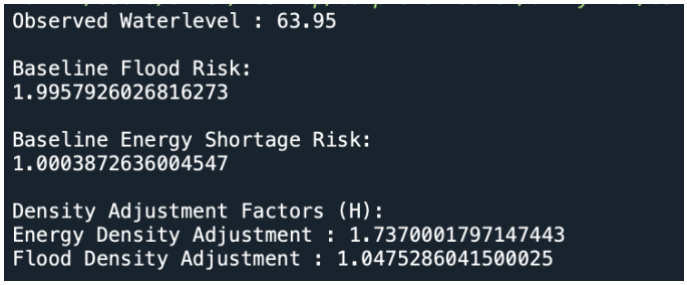


Figure 9-1 Printout from Python, Density Adjustment factor

Examining the Python printout, the density factor is a significant mitigating factor when nearing a flood level. At an observed water level of 63.95 meters in Tyrifjorden, close to a flooding level of 64.2 meters, the energy density adjustment is high, increasing the energy risk, but mitigating the flood risk. This is because, at 63.95 meters, most of the density is below this level. This implies that, historically, water levels are likely to decrease from this point, indicating a lower probability of flooding. However, this also reveals a potential weakness in the formula. Firstly, the formula lacks a component to adjust for immediate risks, such as sudden weather changes. This design choice makes the risk values dependent on expert judgment and additional analysis at both higher and lower levels.

The Energy Density Adjustment results in a higher Energy Shortage Risk at extreme water levels. At 63.95 meters, the energy risk is practically zero, but a time-independent risk value cannot be zero, although it is quite low.

The risk values are timeless; the primary goal was to quantify the risk at certain levels. This timelessness and avoidance of prediction mean the formula does not account for typical water level changes over a day, week, or month. The probabilities of weekly changes, while not part of the thesis, illustrate a major consequence of the chosen formula:

- Probability of no significant change (±5 cm): 0.8334
- Probability of decrease (> 5 cm): 0.0943
- Probability of increase (> 5 cm): 0.0723

These probabilities show the limitations of the formula's design. By not incorporating probability and prediction, the ability to account for expected changes over a week is lost. While the formula retains value, it misses some advantages offered by a model, decision tree, or a compounded decision-making process. For example, a 10% chance of the water level changing by 10 centimeters can nullify many risk values over a week. However, the risk values produced by the formula are time-independent, meaning the risk score at the highest regulated water level (HRW) should not be zero, even if flooding or energy shortage is not "expected" in the coming weeks. Although this represents limitations, it also grounds the formula in its reliance on long-term seasonal changes rather than short-term fluctuations.

The formula's performance is intrinsically linked to the quality and scope of historical data used. Assumptions, such as ignoring immediate weather forecasts, aim to streamline the focus on long-term trends rather than short-term fluctuations. Focusing on historical data and statistical measures helps avoid the noise of short-term weather variations, providing a clearer picture of current water levels. However, ignoring immediate weather forecasts may limit the framework's real-time responsiveness, potentially impacting its effectiveness in sudden change scenarios. Therefore, the formula is not standalone; expert judgment and other predictors are vital for the formula to account for fluctuations.

The inclusion of extended density estimations to account for unobserved events enhances the formula's comprehensiveness, addressing potential risks beyond the immediate scope of

historical data. This approach bridges gaps in historical data, offering a more holistic risk assessment.

Adaptability to future climatic changes is crucial, with rising temperatures and changing precipitation patterns potentially impacting water levels and necessitating adjustments to current strategies. The formula's framework allows for incorporating new data and trends, ensuring its continued relevance. Continuous monitoring and real-time data integration will be essential to address evolving climatic conditions and maintain the formula's effectiveness.

Balancing quantitative formula outputs with qualitative expert judgments is vital. While the formula provides a strong quantitative foundation, expert input is crucial for interpreting results and making final decisions. Integrating expert judgment contextualizes formula outputs within the broader decision-making framework, enhancing practical utility. In complex or ambiguous situations, expert judgment is indispensable, underscoring the need for a collaborative approach.

To sum up, the research has shown stable water levels, and the need for a formula might not be immediately apparent. However, the quantification of history, knowledge, and statistics provides valuable insights into current risks. For example, winter, across all reservoirs, shows no need for a flood risk formula, although it provides insights into energy shortage risks. While there are no flooding risks in winter, a dry spring with low winter water levels will affect energy production moving into spring. The much-mentioned time-independence offers insights into the magnitude of risk associated with a given water level. A very low winter water level with little snow, a dry spring, and a warm summer will impact the following year's water levels. This is the value of the timelessness approach.

Final thoughts and Future Directions

The decision-support formula has mostly shown the stability in the waterlevels, while still addressing the occurrences of extreme lows and highs. Analysis has shown that waterlevels mostly are regulated between low risk levels.

Its development reflects a meticulous and comprehensive approach, integrating historical data, regulatory requirements, and seasonal trends. Future research should focus on incorporating real-time weather data and expanding the formula into a model. Continuous

validation and updates will be crucial to maintaining the formulas relevance and effectiveness amidst dynamic and evolving climatic conditions.

10. Bibliography

Regjeringen.no. (2016, July 20). *The History of Norwegian Hydropower in 5 Minutes*. Retrieved from https://www.regjeringen.no/en/topics/energy/renewable-energy/the-history-of-norwegian-hydropower-in-5-minutes/id2346106/

International Hydropower Association. (2023). *A Brief History of Hydropower*. Retrieved from https://www.hydropower.org/iha/discover-history-of-hydropower

Energy Facts Norway. (2023). *Electricity Production*. Retrieved from Energy Facts Norway: https://energifaktanorge.no/en/norsk-energiforsyning/kraftproduksjon/#hydropower

Thorsnæs, G. (2023, December 31). *Drammensvassdraget*. Retrieved from Store Norske Leksikon: https://snl.no/Drammensvassdraget

Holmqvist, E. (2000). *Analyse av flomvannstander og tørrårstilsig i Tyrifjorden.* Oslo: Norwegian Water Resources and Energy Directorate.

Lauritzen, P. R. (2023, December 23). *Sperillen*. Retrieved from Store Norske Leksikon: https://snl.no/Sperillen

Drammensvassdraget. (2024, February 14). *Drammensvassdraget*. Retrieved from Wikipedia.org: https://no.wikipedia.org/wiki/Drammensvassdraget

Ekstremværet Hans. (2024, April 28). *Ekstremværet Hans*. Retrieved from https://no.wikipedia.org/wiki/Ekstremværet_Hans

Hydropower. (2024, May 20). *Hydropower*. Retrieved from Wikipedia.org: https://en.wikipedia.org/wiki/Hydropower

Randsfjorden. (2024, January 1). *Randsfjorden*. Retrieved from Wikipedia.org: https://no.wikipedia.org/wiki/Randsfjorden

Sperillen. (2024, February 14). *Sperillen*. Retrieved from Wikipedia.org: https://no.wikipedia.org/wiki/Sperillen

Tyrifjorden. (2024, January 1). *Tyrifjorden*. Retrieved from Wikipedia: https://no.wikipedia.org/wiki/Tyrifjorden

NVE - Drammensvassdraget. (2024). *Drammensvassdraget*. Retrieved from NVE Varsom.no: https://www.varsom.no/flom-og-jordskred/om-flom-og-jordskred/rad-og-forebygging/vassdragsregulanters-ansvar-og-muligheter/drammensvassdraget/

NVE - Tyrifjorden. (2024). *Tyrifjorden*. Retrieved from Norwegian Water Resources and Energy Directorate: https://www.nve.no/vann-og-vassdrag/vassdragsforvaltning/verneplan-for-vassdrag/viken/012-14-tyrifjorden/

NVE - Vannkraft. (2023). *Vannkraft*. Retrieved from Norwegian Water Resources and Energy Directorate: https://www.nve.no/energi/energisystem/vannkraft/

Vannkraftdatabase. (2024). *Vannkraftdatabase*. Retrieved from Norwegian Water Resources and Energy Directorate: https://www.nve.no/energi/energisystem/vannkraft/vannkraftdatabase/

Vassdragsregulanters ansvar og muligheter. (2023). *Vassdragsregulanters ansvar og muligheter*. Retrieved from NVE Varsom.no: https://www.varsom.no/flom-og-jordskred/om-flom-og-jordskred/rad-og-forebygging/vassdragsregulanters-ansvar-og-muligheter/

NVE - Vårflom. (2020). *Vårflom*. Retrieved from NVE Varsom.no: https://www.varsom.no/flom-og-jordskred/om-flom-og-jordskred/varflom/

NVE - Begnavassdraget med Sperillen. (2024). *Begnavassdraget med Sperillen*. Retrieved from NVE Varsom.no: https://www.varsom.no/flom-og-jordskred/om-flom-og-jordskred/rad-og-forebygging/vassdragsregulanters-ansvar-og-muligheter/drammensvassdraget/begnavassdraget-med-sperillen/

Thorsnæs, G. (2023, October 3). *Randsfjorden*. Retrieved from Store norske leksikon: https://snl.no/Randsfjorden

NVE Atlas. (n.d.). *NVE Atlas*. Retrieved from https://atlas.nve.no/html5Viewer/?viewer=nveatlas

Sildre NVE. (2024). *Sildre*. Retrieved from Sildre: https://sildre.nve.no/map?x=380400&y=7228000&zoom=4

Reservoar (Hydrologi). (2022, 09 1). *Reservoar (Hydrologi)*. Retrieved from https://no.wikipedia.org/w/index.php?title=Reservoar_(hydrologi)&oldid=22926530

NVE - Flom. (2022, 05 13). *Flom*. Retrieved from NVE: https://www.nve.no/naturfare/laer-om-naturfare/flom/

NVE - Tørke. (2020, 01 14). *Tørke*. Retrieved from NVE.no: https://www.nve.no/naturfare/laer-om-naturfare/toerke/

NVE - Regnflom. (2022, 05 13). *Regnflom*. Retrieved from NVE.no: https://www.nve.no/naturfare/laer-om-naturfare/flom/regnflom/

NVE - Ordliste. (2024, 06 01). *Ordliste for flom*. Retrieved from NVE.no: https://www.varsom.no/flom-og-jordskred/ordliste/ordliste-for-flom/

Rosvold, K. A. (2020, 06 9). *Manøvreringsreglementet*. Retrieved from Store Norske Leksikon: https://snl.no/manøvreringsreglement

Energidepartementet. (2024, 06 07). *Lov om regulering og kraftutbygging i vassdrag (vassdragsreguleringsloven)*. Retrieved from Lovdata.no: https://lovdata.no/dokument/NL/lov/1917-12-14-17

Sperillen. (1926). *Manøvrerling av reguleringsdammen for Sperillen.* Oslo: Norwegian State.

Olje- og energidepartementet. (2022). *Endelig fastsettelse av manøvreringsreglement for Randsfjorden.* Oslo: Olje- og energidepartementet.

Nærings- og energidepartementet. (1995). *Tillatelse for Foreningen av Randsfjord Regulering til fortsatt regulering av Randsfjorden.* Oslo: Nærings- og energidepartementet.

Nærings- og energidepartementet. (1994). *Foreningen til Tyrifjords regulering. Revisjon av manøvreringsregment for Tyrifjorden.* Oslo: Nærings- og energidepartementet.

11. AI Disclosure

ChatGPT was used in the start as a helper and "companion" in the definition of the thesis proposal and scope. ChatGPT has been used to proofread and provide a certain flow to the text where I felt this was needed.

ChatGPT has been used as a coding expert. While python programming is not a new thing to me, certain errors in the code cannot be effectively found and fixed with the extent of the programming code and number of lines in this coding amount. ChatGPT programming help enabled the master thesis to be more extensive, more thorough. Since I did not have to use half my time finding spelling errors in the python code.

Simen Askeland, 14. June 2024.

12. Python Note

Python version used: 5.5.1

Necessary modules are represented at the top of each python script.

For the python programs to work it is important to look at the file imports and file savings for the program to work. The file path is important to change to get it working on your computer. Also, be careful when naming the downloaded csv files, they have to match the name in the script. For help I advise asking ChatGPT.

13. Dataset Downloading

The datasets can be downloaded on:

Tyrifjorden: https://sildre.nve.no/station/12.65.0

Sperillen: https://sildre.nve.no/station/12.83.0

Randsfjorden: https://sildre.nve.no/station/12.69.0

14. Appendix

1. *Data_Cleaner.py*
2. Correlation Analysis
3. *Correlation.py*
4. *Statistical_Analysis.py*
5. Statistical Analysis Tyrifjorden
6. Statistical Analysis Sperillen
7. Statistical Analysis Randsfjorden
8. *Seasonal_Analysis.py*
9. Seasonal Analysis Tyrifjorden
10. Seasonal Analysis Sperillen
11. Seasonal Analysis Randsfjorden
12. Complete Multimodal Analysis
13. *Yearly_plots.py*
14. *Multimodal_analysis.py*
15. *Reservoir_to_DailyEnergy.py*
16. *Data_Preperation.py*
17. *States_constructor.py*
18. *Historic_Risk_Factor.py*
19. *Extended_Risk_Factor.py*
20. *Decision_Single.py*
21. *Decision_for_loop.py*
22. Complete Risk Values
23. Sensitivity_Analysis.py
24. External Reports and Sources

Appendix 1:

Data_Cleaner.py

```
import pandas as pd
import sys
import matplotlib.pyplot as plt
import numpy as np
import warnings
import matplotlib.dates as mdates
warnings.simplefilter("ignore", category=UserWarning)
warnings.simplefilter("ignore", category=FutureWarning)
# Path to the CSV file
name = "Randsfjord"
data_type = "Waterflow"
# Input complete local filepath
file_path = f'/Users/simen/Desktop/Complete Master/01 Data/Uncleaned Data/{name}_{data_type}_Daily.csv'
df = pd.read_csv(file_path, delimiter=';',header=1)
print("First 15 Rows before cleaning")
print(df.head(15))
print("--------------------------------------------------------------------------")
print("\nLast 15 Rows before cleaning")
print(df.tail(15))
# Option to exit or move on
continue_choice = input("Move on? (yes/no): ").lower()
if continue_choice != 'yes':
    print("Exiting.")
    sys.exit()
Remove_1 = 'Korrigert'
Remove_2 = 'Kontrollert'
df = df.drop([Remove_1, Remove_2], axis=1)
Rename_1 = 'Tidspunkt'
if data_type == 'Waterlevel':
    Rename_2 = 'Vannstand (m)'
elif data_type == 'Reservoir':
    Rename_2 = 'Magasinvolum (millioner m³)'
else:
    Rename_2 = 'Vannføring (m³/s)'
df = df.rename(columns={Rename_1: 'Date', Rename_2: data_type})
df['Date'] = pd.to_datetime(df['Date']).dt.date
df = df.dropna()
# Replace commas with dots in the entire column
df[data_type] = df[data_type].str.replace(',', '.')
# Convert the column to numeric
df[data_type] = pd.to_numeric(df[data_type])
```

```
# Convert 'data_type' column to numeric (assuming 'data_type' is a column name)
df[data_type] = pd.to_numeric(df[data_type], errors='coerce')
# Reset index after manipulation
df.reset_index(drop=True, inplace=True)
# Convert 'Date' column to datetime objects
df['Date'] = pd.to_datetime(df['Date'])
# Define the date range to keep
start_date = '2004-01-01'
end_date = '2023-12-31'
# Create a boolean mask to filter rows based on the date range
mask = (df['Date'] >= start_date) & (df['Date'] <= end_date)
# Apply the mask to filter rows within the specified date range
filtered_df = df[mask]
# Keep rows based on the boolean mask
df = df[mask]
# Reset the index
df.reset_index(drop=True, inplace=True)
# Filter rows where Waterlevel is 0
zero_data_type_rows = df[df[data_type] <= 0]
print('number of rows below 0 : ',len(zero_data_type_rows))
#print(zero_data_type_rows)  # Print rows where Waterlevel is 0
df.loc[zero_data_type_rows.index, data_type] = 0
# Filter rows where Waterlevel is 0
zero_data_type_rows = df[df[data_type] < 0]
print('Zero-Removal')
print('number of rows below 0 : ',len(zero_data_type_rows))
#print(zero_data_type_rows)  # Print rows where Waterlevel is 0
print(len(df[df[data_type] < 0]))
print('Final Statistics')
print(df.describe())
# Option to exit or move on
continue_choice = input("Move on? (yes/no): ").lower()
if continue_choice != 'yes':
    print("Exiting.")
    sys.exit()
# Calculate statistical values
mean_value = df.mean()
std_dev = df[data_type].std()
min_value = df[data_type].min()
max_value = df[data_type].max()
# Histogram
plt.figure(figsize=(10, 6))
plt.hist(df[data_type], bins=50, alpha=0.7, color='blue')
# Mean
plt.axvline(x=mean_value[data_type], color='g', linestyle='-', label='Mean')
```

```
# Standard Deviation (both sides)
plt.axvline(x=mean_value[data_type] - std_dev, color='c', linestyle='--', label='Standard Deviation')
plt.axvline(x=mean_value[data_type] + std_dev, color='c', linestyle='--')
# Min
plt.axvline(x=min_value, color='m', linestyle='-.', label='Min Value')
# Max
plt.axvline(x=max_value, color='m', linestyle='-.', label='Max Value')
plt.title(f'Histogram of mean and standard deviation for {data_type} in {name}')
plt.xlabel(data_type)
plt.ylabel('Frequency')
plt.legend()
plt.savefig(f'/Users/simen/Desktop/Complete Master/04 Plots/{name}_{data_type}_histogram_plot.png')
plt.show()
# Set 'Date' as the DataFrame index
df.set_index('Date', inplace=True)
# Time-series plot
plt.figure(figsize=(10, 6))
plt.plot(df.index, df[data_type], label=data_type)
# Format the x-axis to show years and months/dates
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.gcf().autofmt_xdate()  # Auto-rotate dates for better spacing
# Add vertical lines for each year
for year in pd.date_range(start=df.index.min(), end=df.index.max(), freq='YS'):
    plt.axvline(x=year, color='gray', linestyle=':', linewidth=0.5)
plt.title(f'Daily Time Series of {data_type} in {name}')
plt.xlabel('Date')
plt.ylabel(data_type)
plt.legend()
plt.tight_layout()
plt.savefig(f'/Users/simen/Desktop/Complete Master/04 Plots/{name}_{data_type}_time_series_plot.png')
plt.show()
# Reset index after manipulation
df.reset_index(inplace=True)
print()
print('Top of Cleaned Dataset')
print(df.head())
print()
print('Bottom of Cleaned Dataset')
print(df.tail())
# Option to exit or Save
continue_choice = input("Save? (yes/no): ").lower()
if continue_choice != 'yes':
    print("Exiting.")
    sys.exit()
```

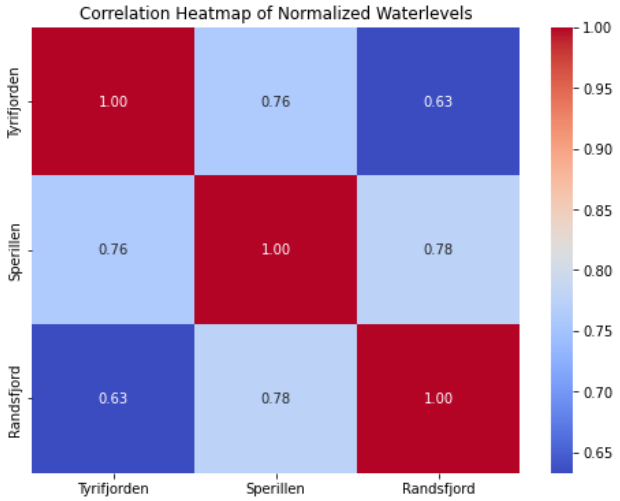
```
# Where to save the cleaned dataset
output_file_path = '/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/'
filename_csv = f'Cleaned_{name}_{data_type}.csv'
# Save the cleaned DataFrame to the new CSV file
df.to_csv(output_file_path + filename_csv, index=False)
# Save the cleaned DataFrame to excel file for Appendix.
df.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{name}/Cleaned_{name}_{data_type}.xlsx', index=False)
```

Appendix 2:

Correlation Analysis

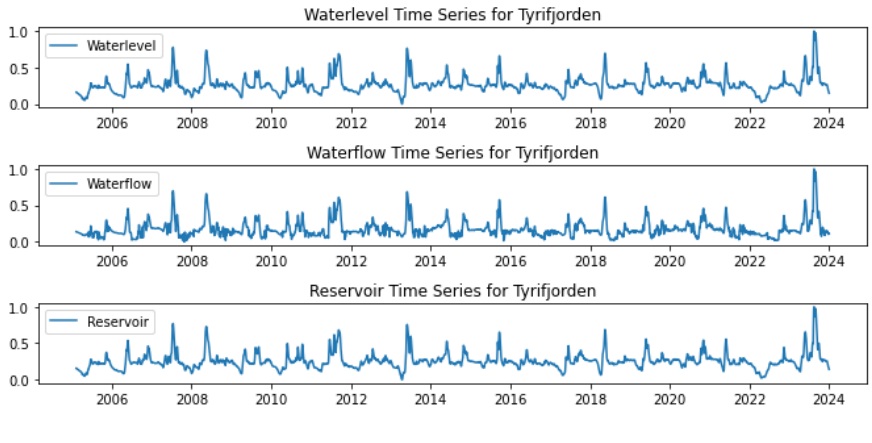
All Lakes:

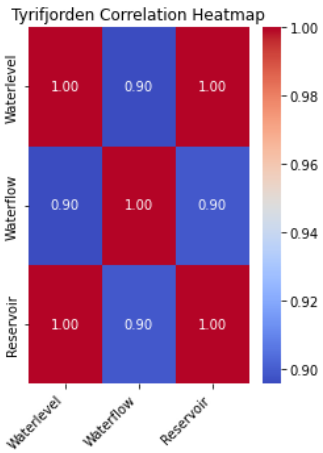
	Tyrifjorden	Sperillen	Randsfjord
Tyrifjorden	1	0,761035	0,633054
Sperillen	0,761035	1	0,775634
Randsfjord	0,633054	0,775634	1



Tyrifjorden:

	Waterlevel	Waterflow	Reservoir
Waterlevel	1	0,89595	0,999935
Waterflow	0,89595	1	0,89906
Reservoir	0,999935	0,89906	1



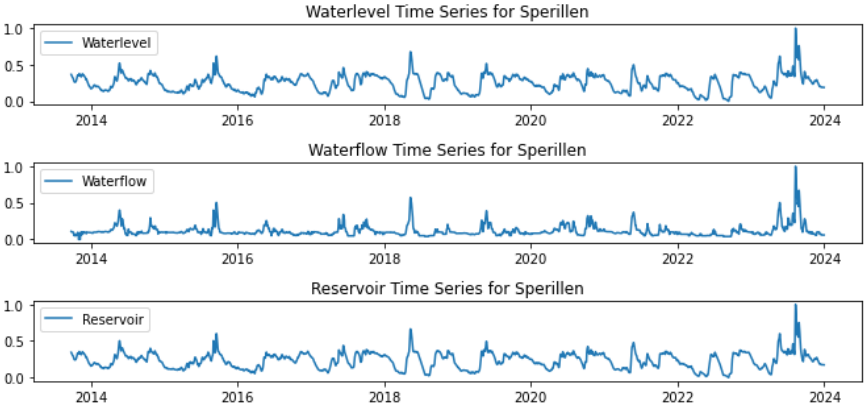


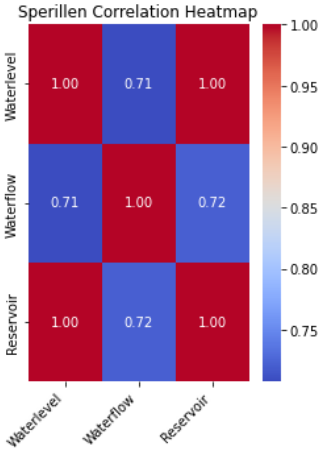
Appendix 2:

Correlation Analysis

Sperillen:

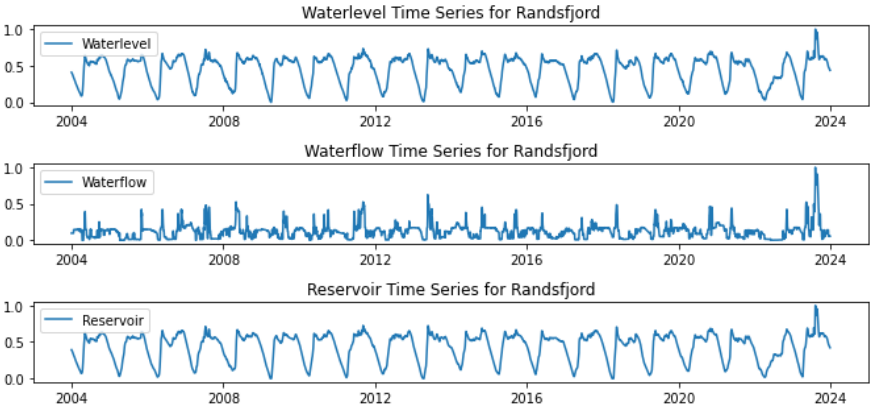
	Waterlevel	Waterflow	Reservoir
Waterlevel	1	0,707879	0,999633
Waterflow	0,707879	1	0,721373
Reservoir	0,999633	0,721373	1

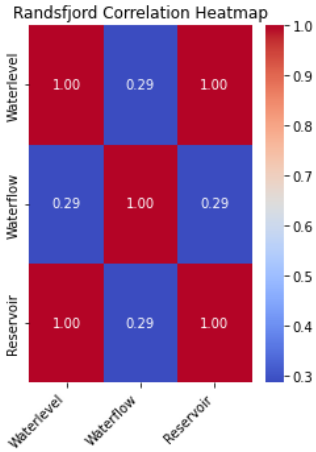




Randsfjorden:

	Waterlevel	Waterflow	Reservoir
Waterlevel	1	0,287398	0,999956
Waterflow	0,287398	1	0,289755
Reservoir	0,999956	0,289755	1





Appendix 3:

Correlation.py

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
from sklearn.preprocessing import MinMaxScaler
import matplotlib.gridspec as gridspec
lake_names = ["Tyrifjorden", "Sperillen", "Randsfjord"]
for name in lake_names:
    # Load the datasets with complete local filepath
    waterlevel_data = pd.read_csv(f"/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/Cleaned_{name}_Waterlevel.csv")
    waterflow_data = pd.read_csv(f"/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/Cleaned_{name}_Waterflow.csv")
    reservoirlevel_data = pd.read_csv(f"/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/Cleaned_{name}_Reservoir.csv")
    # Merge the datasets on 'Date'
    merged_data = pd.merge(waterlevel_data, waterflow_data, on='Date')
    merged_data = pd.merge(merged_data, reservoirlevel_data, on='Date')
    # Ensures the 'Date' column is a datetime type
    merged_data['Date'] = pd.to_datetime(merged_data['Date'])
    merged_data_without_date = merged_data.drop(columns=['Date'])
    # Calculate the correlation
    correlation_matrix = merged_data_without_date.corr()
    # Normalize the data
    scaler = MinMaxScaler()
    merged_data[['Waterlevel', 'Waterflow', 'Reservoir']] = scaler.fit_transform(
        merged_data[['Waterlevel', 'Waterflow', 'Reservoir']])
    fig = plt.figure(figsize=(13, 5))
    gs = gridspec.GridSpec(3, 2, width_ratios=[3, 1])
    # Create time-series subplots in the first column of the grid
    time_series_axes = []
    for i in range(3):
        ax = fig.add_subplot(gs[i, 0])
        time_series_axes.append(ax)
        ax.plot(merged_data['Date'], merged_data.iloc[:, i+1], label=merged_data.columns[i+1])
        ax.legend()
        ax.set_title(f"{merged_data.columns[i+1]} Time Series for {name}")
    heatmap_ax = fig.add_subplot(gs[:, 1])
    # Plot the heatmap
    sns.heatmap(correlation_matrix, ax=heatmap_ax, annot=True, cmap='coolwarm', fmt=".2f")
    heatmap_ax.set_title(f'{name} Correlation Heatmap')
    heatmap_ax.set_aspect('auto')
```

```
    for label in heatmap_ax.get_xticklabels():
        label.set_rotation(45)  # Rotate labels to 45 degrees
        label.set_ha('right')
    plt.tight_layout()
    plt.savefig(f'/Users/simen/Desktop/Complete Master/04 Plots/{name}_correlation_heatmap.png')
    plt.show()
    correlation_matrix_save = correlation_matrix
    correlation_matrix_save.to_excel(f"/Users/simen/Desktop/Complete Master/03 Excel Products/01
Correlation/correlation_matrix_{name}.xlsx")
# Load the datasets for Waterlevel for all lakes
tyrifjorden_data = pd.read_csv('/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/Cleaned_Tyrifjorden_Waterlevel.csv')
sperillen_data = pd.read_csv('/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/Cleaned_Sperillen_Waterlevel.csv')
randsfjord_data = pd.read_csv('/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/Cleaned_Randsfjord_Waterlevel.csv')
tyrifjorden_data['Date'] = pd.to_datetime(tyrifjorden_data['Date'])
sperillen_data['Date'] = pd.to_datetime(sperillen_data['Date'])
randsfjord_data['Date'] = pd.to_datetime(randsfjord_data['Date'])
# Normalize the datasets
scaler = MinMaxScaler()
tyrifjorden_data['Normalized'] = scaler.fit_transform(tyrifjorden_data[['Waterlevel']])
sperillen_data['Normalized'] = scaler.fit_transform(sperillen_data[['Waterlevel']])
randsfjord_data['Normalized'] = scaler.fit_transform(randsfjord_data[['Waterlevel']])
# Merge the datasets on a common date column
combined_data = pd.DataFrame()
combined_data['Date'] = tyrifjorden_data['Date']  # Assuming all datasets have the same date range
combined_data = combined_data.merge(tyrifjorden_data[['Date', 'Normalized']], on='Date', how='left')
combined_data = combined_data.merge(sperillen_data[['Date', 'Normalized']], on='Date', how='left', suffixes=(' Tyrifjorden', '
Sperillen'))
combined_data = combined_data.merge(randsfjord_data[['Date', 'Normalized']], on='Date', how='left')
combined_data.rename(columns={'Normalized': 'Randsfjord'}, inplace=True)
combined_data.rename(columns={'Normalized Tyrifjorden': 'Tyrifjorden'}, inplace=True)
combined_data.rename(columns={'Normalized Sperillen': 'Sperillen'}, inplace=True)
combined_data_without_date = combined_data.drop(columns=['Date'])
# Calculate the correlation
correlation_matrix = combined_data_without_date.corr()
plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Normalized Waterlevels')
plt.savefig('/Users/simen/Desktop/Complete Master/04 Plots/all_lakes_correlation_heatmap.png')
plt.show()
# Save the correlation matrix to an Excel file
correlation_matrix.to_excel("/Users/simen/Desktop/Complete Master/03 Excel Products/01
Correlation/correlation_matrix_all.xlsx")
```

Appendix 4:

Statistical_Analysis.py

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.dates as mdates
# Path to the CSV file
name = "Randsfjord"
data_type = "Waterlevel"
file_path = f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_{name}_Waterlevel.csv'
# Read the CSV file
df = pd.read_csv(file_path)
df[data_type] = pd.to_numeric(df[data_type], errors='coerce') # Convert data to numeric
df.reset_index(drop=True, inplace=True)
if name == 'Randsfjord':
    if data_type == 'Waterlevel':
        # Define flooding levels
        mean_flood = 134.689
        five_year_flood = 134.9159
        ten_year_flood = 135.1058
        twenty_year_flood = 135.2902
        fifty_year_flood = 135.5321
        LRV = 131.3
        HRV = 134.5
        LRV_HRV = [LRV, HRV]
        flooding_levels = [mean_flood, five_year_flood, ten_year_flood, twenty_year_flood, fifty_year_flood]
    else:
        # Handle the case when data_type is not 'Waterlevel'
        flooding_levels = None
elif name == 'Tyrifjorden':
    if data_type == 'Waterlevel':
        # Define flooding levels
        mean_flood = 64.2
        five_year_flood = 64.7
        ten_year_flood = 64.9
        twenty_year_flood = 65.1
        fifty_year_flood = 65.2
        LRV = 62
        HRV = 63
        LRV_HRV = [LRV, HRV]
        flooding_levels = [mean_flood, five_year_flood, ten_year_flood, twenty_year_flood, fifty_year_flood]
    else:
        # Handle the case when data_type is not 'Waterlevel'
        flooding_levels = None
```

```
elif name == 'Sperillen':
    if data_type == 'Waterlevel':
        # Define flooding levels
        mean_flood = 151.1276
        five_year_flood = 151.6132
        ten_year_flood = 152.0137
        twenty_year_flood = 152.4
        fifty_year_flood = 152.9034
        LRV = 147.95
        HRV = 150.25
        LRV_HRV = [LRV, HRV]
        flooding_levels = [mean_flood, five_year_flood, ten_year_flood, twenty_year_flood, fifty_year_flood]
    else:
        # Handle the case when data_type is not 'Waterlevel'
        flooding_levels = None
# Calculate and print basic statistics
mean_value = df[data_type].mean()
std_dev = df[data_type].std()
min_value = df[data_type].min()
quantile_25 = df[data_type].quantile(0.25)
median_value = df[data_type].median()
quantile_75 = df[data_type].quantile(0.75)
quantile_90 = df[data_type].quantile(0.90)
quantile_95 = df[data_type].quantile(0.95)
quantile_99 = df[data_type].quantile(0.99)
max_value = df[data_type].max()
total_data_points = len(df[data_type])
print('Total Data Points =', total_data_points)
print('Mean Value =', mean_value)
print('Standard Deviation =', std_dev)
print('Min Value =', min_value)
print('50% / Median =', median_value)
print('75% =', quantile_75)
print('90% =', quantile_90)
print('95% =', quantile_95)
print('99% =', quantile_99)
print('Max Value =', quantile_90)
# Save statistics into a DataFrame
statistics_df = pd.DataFrame({
    'Statistic': ['Total Data Points', 'Mean', 'Standard Deviation', 'Min', '25%', 'Median', '75%', '90%', '95%', '99%', 'Max'],
    'Value': [total_data_points, mean_value, std_dev, min_value, quantile_25, median_value, quantile_75, quantile_90,
quantile_95, quantile_99, max_value]
})
print(df.head())
print(statistics_df.head())
```

```
# Histogram
plt.figure(figsize=(10, 6))
plt.hist(df[data_type], bins=100, alpha=0.7, color='blue')
# Mean
plt.axvline(x=mean_value, color='g', linestyle='-', label='Mean')
# Standard Deviation (both sides)
plt.axvline(x=mean_value - std_dev, color='c', linestyle='--', label='Standard Deviation')
plt.axvline(x=mean_value + std_dev, color='c', linestyle='--')
# Min
plt.axvline(x=min_value, color='m', linestyle='-.', label='Min Value')
# Max
plt.axvline(x=max_value, color='m', linestyle='-.', label='Max Value')
plt.title(f'Histogram of mean and standard deviation for {data_type} in {name}')
plt.xlabel(data_type)
plt.ylabel('Frequency')
plt.legend()
plt.show()
# Histogram
plt.figure(figsize=(10, 6))
plt.hist(df[data_type], bins=100, alpha=0.7, color='blue')
# 25th Percentile
plt.axvline(x=quantile_25, color='y', linestyle=':', label='25% Percentile')
# Median
plt.axvline(x=median_value, color='k', linestyle='-', label='Median (50% Percentile)')
# 75th Percentile
plt.axvline(x=quantile_75, color='y', linestyle=':', label='75% Percentile')
# 90th Percentile
plt.axvline(x=quantile_90, color='y', linestyle=':', label='90% Percentile')
# 95th Percentile
plt.axvline(x=quantile_95, color='y', linestyle=':', label='95% Percentile')
# 99th Percentile
plt.axvline(x=quantile_99, color='y', linestyle=':', label='99% Percentile')
plt.title(f'Histogram of percentiles for {data_type} in {name}')
plt.xlabel(data_type)
plt.ylabel('Frequency')
plt.legend()
plt.show()
# Histogram
plt.figure(figsize=(10, 6))
plt.hist(df[data_type], bins=100, alpha=0.7, color='blue')
for value in flooding_levels:
    plt.axvline(x=value, color='r', linestyle='--', label='Flooding Level' if 'Flooding Level' not in
plt.gca().get_legend_handles_labels()[1] else "_nolegend_")
for value in LRV_HRV:
```

```
    plt.axvline(x=value, color='y', linestyle='--', label='LRW/HRW' if 'LRW/HRW' not in
plt.gca().get_legend_handles_labels()[1] else "_nolegend_")
plt.title(f'Histogram of {data_type} in {name}')
plt.xlabel(data_type)
plt.ylabel('Frequency')
plt.legend()
if data_type == 'Waterlevel':
    # Add vertical lines at specified x-axis values
    for value in flooding_levels:
        plt.axvline(x=value, color='r', linestyle='--')
else:
    # Handle the case when data_type is not 'Waterlevel'
    # For example, set flooding_levels to None or print a message
    flooding_levels = None
if data_type == 'Waterlevel':
    # Add vertical lines at specified x-axis values
    for value in LRV_HRV:
        plt.axvline(x=value, color='y', linestyle='--')
else:
    # Handle the case when data_type is not 'Waterlevel'
    # For example, set flooding_levels to None or print a message
    LRV_HRV = None
plt.show()
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
# Time-series plot setup
plt.figure(figsize=(10, 6))
plt.plot(df.index, df[data_type], label=data_type)
# Format the x-axis to show years
plt.gca().xaxis.set_major_locator(mdates.YearLocator())
plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%Y'))
plt.gcf().autofmt_xdate()  # Auto-rotate dates for better spacing
plt.title(f'Daily Time Series of {data_type} in {name}')
plt.xlabel('Date')
plt.ylabel(data_type)
# Plotting flooding levels with labels
if data_type == 'Waterlevel':
    for index, level in enumerate(flooding_levels):
        plt.axhline(y=level, color='r', linestyle='--', label=f'Flooding Level {index + 1}')
# Plotting LRV_HRV levels with labels
if data_type == 'Waterlevel':
    for index, level in enumerate(LRV_HRV):
        plt.axhline(y=level, color='y', linestyle='--', label=f'LRV_HRV Level {index + 1}')
plt.legend()
plt.tight_layout()
```

```
plt.show()
# Number of days within the regulation zone 62-63 meters
regulation_zone_days = len(df[(df[data_type] > LRV) & (df[data_type] < HRV)])
regulation_zone_percent = (regulation_zone_days / total_data_points) * 100
print(f"Number of days within the regulation zone 62-63 meters: {regulation_zone_days}
({regulation_zone_percent:.2f}%)")
# Number of days within the caution zone, above HRV and below mean-flood
caution_zone_days = len(df[(df[data_type] > HRV) & (df[data_type] < mean_flood)])
caution_zone_percent = (caution_zone_days / total_data_points) * 100
print(f"Number of days within the caution zone HRW to mean-flood: {caution_zone_days} ({caution_zone_percent:.2f}%)")
# Number of days with mean flood to 5-year flood
meanflood_5year_days = len(df[(df[data_type] > mean_flood) & (df[data_type] < five_year_flood)])
meanflood_5year_percent = (meanflood_5year_days / total_data_points) * 100
print(f"Number of days with mean-flood to 5-year flood: {meanflood_5year_days} ({meanflood_5year_percent:.2f}%)")
# Number of days with 5 to 10-year flood
five_10year_days = len(df[(df[data_type] > five_year_flood) & (df[data_type] < ten_year_flood)])
five_10year_percent = (five_10year_days / total_data_points) * 100
print(f"Number of days with 5 to 10-year flood: {five_10year_days} ({five_10year_percent:.2f}%)")
# Number of days with 10 to 20-year flood
ten_20year_days = len(df[(df[data_type] > ten_year_flood) & (df[data_type] < twenty_year_flood)])
ten_20year_percent = (ten_20year_days / total_data_points) * 100
print(f"Number of days with 10 to 20-year flood: {ten_20year_days} ({ten_20year_percent:.2f}%)")
# Number of days with 20 to 50-year flood
twenty_50year_days = len(df[(df[data_type] > twenty_year_flood) & (df[data_type] < fifty_year_flood)])
twenty_50year_percent = (twenty_50year_days / total_data_points) * 100
print(f"Number of days with 20 to 50-year flood: {twenty_50year_days} ({twenty_50year_percent:.2f}%)")
# Number of days with 50-year flood
fifty_year_days = len(df[df[data_type] > fifty_year_flood])
fifty_year_percent = (fifty_year_days / total_data_points) * 100
print(f"Number of days with 50-year flood: {fifty_year_days} ({fifty_year_percent:.2f}%)")
# Total number of days with flood
total_flood_days = len(df[df[data_type] > mean_flood])
total_flood_percent = (total_flood_days / total_data_points) * 100
print(f"Number of days with flood: {total_flood_days} ({total_flood_percent:.2f}%)")
# Save to DataFrame
frequency_days_df = pd.DataFrame({
    'Condition': ['Regulation Zone', 'Caution Zone', 'Mean to 5-Year Flood', '5 to 10-Year Flood', '10 to 20-Year Flood', '20 to
50-Year Flood', '50-Year Flood', 'Total Flood Days'],
    'Days': [regulation_zone_days, caution_zone_days, meanflood_5year_days, five_10year_days, ten_20year_days,
twenty_50year_days, fifty_year_days, total_flood_days],
    'Percent': [regulation_zone_percent, caution_zone_percent, meanflood_5year_percent, five_10year_percent,
ten_20year_percent, twenty_50year_percent, fifty_year_percent, total_flood_percent]
})
# Assuming mean_value, std_dev, df, data_type, and total_data_points are already defined
# Example of std_devs, which needs to be defined:
```

```
std_devs = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
# Calculate probabilities for values greater than or equal to the specified number of standard deviations from the mean
probabilities = []
percentage_within_ranges = []
for num_std_devs in std_devs:
    # Calculate the value at the given number of standard deviations from the mean
    value_at_std_devs = mean_value + num_std_devs * std_dev
    # Calculate the probability of observing a value at least as extreme as value_at_std_devs
    probability = len(df[df[data_type] >= value_at_std_devs]) / total_data_points
    # For both sides of the distribution, multiply the probability by 2
    # This accounts for both tails assuming a normal distribution
    adjusted_probability = min(probability * 2, 1)  # Ensure probability does not exceed 100%
    probabilities.append(adjusted_probability)
    # Calculate the percentage falling within the range of the specified number of standard deviations from the mean
    percentage_within_range = (1 - adjusted_probability) * 100
    percentage_within_ranges.append(percentage_within_range)
    print(f"{percentage_within_range:.2f}% falls within {num_std_devs} standard deviation{'s' if num_std_devs > 1 else ''}
from the mean.")

# Save to DataFrame
std_dev_analysis_df = pd.DataFrame({
    'Number of Std Devs': std_devs,
    'Percentage Within Range': percentage_within_ranges
})
level_names = ['Mean', 'Five-year', 'Ten-year', 'Twenty-year', 'Fifty-year']
# Calculate how many standard deviations each flooding level is from the mean
std_devs_from_mean = {}
std_dev_data = []
for level_name, level_value in zip(level_names, flooding_levels):
    num_std_devs_from_mean = (level_value - mean_value) / std_dev
    std_devs_from_mean[level_name] = num_std_devs_from_mean
    std_dev_data.append(num_std_devs_from_mean)  # This line was missing; now it appends each computed std dev
# Print the results for each flood level's standard deviations from the mean
for level_name, num_std_devs_from_mean in std_devs_from_mean.items():
    print(f"{level_name} flooding level is {num_std_devs_from_mean:.2f} standard deviations from the mean.")
# Printing to verify the content and length of std_dev_data
print(std_dev_data)  # This will show the list of standard deviations computed
print(len(level_names))  # This prints the length of level_names, which should be 5
print(len(std_dev_data))  # This now should also print 5, confirming entries are made to the list
# Create DataFrame
flood_levels_std_dev_df = pd.DataFrame({
    'Flood Level': level_names,
    'Std Devs from Mean': std_dev_data
})
# Reset the index to make sure 'Date' is a column, not the index, to avoid issues
```

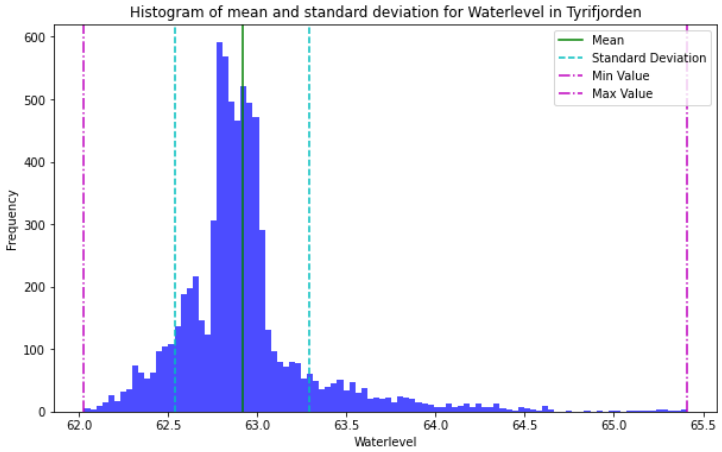
```
df.reset_index(inplace=True)
# Ensure 'Date' column is in datetime format
df['Date'] = pd.to_datetime(df['Date'])
# Now, set 'Date' as the index again, this time for the purpose of resampling
df.set_index('Date', inplace=True)
# Resample the data to get annual statistics. 'A' stands for 'Annual'.
annual_data = df.resample('A').agg(['mean', 'std'])
# Calculate the Yearly Variability Index for each year
# YVI = standard deviation / mean for each year
annual_data['YVI'] = annual_data[(data_type, 'std')] / annual_data[(data_type, 'mean')]
# Calculate the average of the Yearly Variability Index across all years
average_yvi = annual_data['YVI'].mean()
print("Yearly Variability Index (YVI) for each year:\n", annual_data['YVI'])
print("\nAverage Yearly Variability Index (YVI) across all years:", average_yvi)
# Creating DataFrame to hold this information
yearly_variability_index_df = annual_data[[('YVI', '')]].copy()
yearly_variability_index_df.columns = ['Yearly Variability Index']  # Rename the columns for clarity
print()
print(statistics_df)
print()
print(frequency_days_df)
print()
print(std_dev_analysis_df)
print()
print(flood_levels_std_dev_df)
print()
print(yearly_variability_index_df)
# Define the save path
save_path = f'/Users/simen/Desktop/Complete Master/03 Excel Products/02 Lakes/{name}/'
# File name based on a variable 'name'
file_name = f'Combined_Statistical_Data_{name}.xlsx'
full_path = save_path + file_name
# Save all DataFrames to an Excel file with each DataFrame as a separate sheet
with pd.ExcelWriter(full_path, engine='xlsxwriter') as writer:
    statistics_df.to_excel(writer, sheet_name='Statistics', index=False)
    frequency_days_df.to_excel(writer, sheet_name='Frequency Days', index=False)
    std_dev_analysis_df.to_excel(writer, sheet_name='Standard Deviation Analysis', index=False)
    flood_levels_std_dev_df.to_excel(writer, sheet_name='Flood Levels Std Dev', index=False)
    yearly_variability_index_df.to_excel(writer, sheet_name='Yearly Variability Index', index=False)
print(f"All DataFrames have been saved as an excel file at {full_path}.")
df.reset_index(inplace=True)
df.set_index('Date', inplace=True)
plt.figure(figsize=(10, 6))
plt.boxplot(df[data_type].dropna(), vert=True)  # Ensure there are no NaN values
plt.title(f'Boxplot of {data_type} for {name}')
```

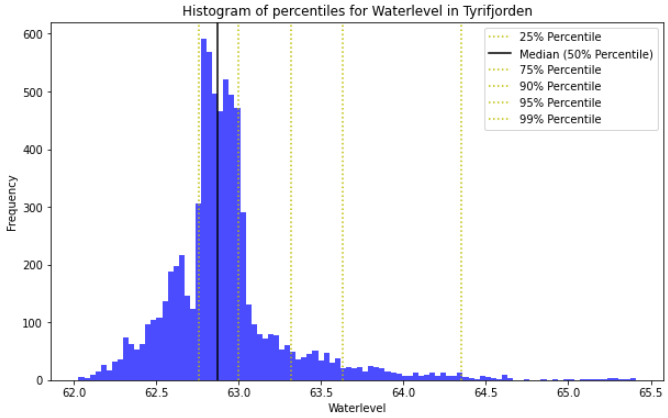
```
plt.ylabel(data_type)
plt.xticks([1], [data_type])  # Set a custom x-axis label
plt.grid(True)
plt.show()
```

Appendix 5:

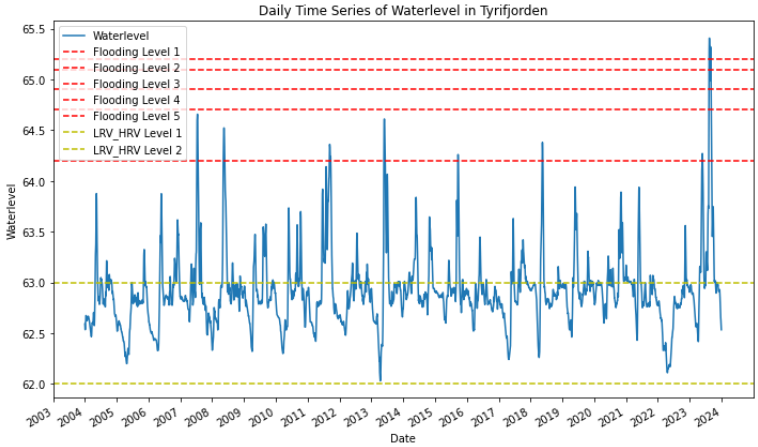
Statistical Analysis Tyrifjorden

Statistic	Value
Total Data Points	7305
Mean	62,9181
Standard Deviation	0,377319
Min	62,02999
25%	62,75689
Median	62,8723
75%	62,99604
90%	63,31896
95%	63,63215
99%	64,35287
Max	65,40757





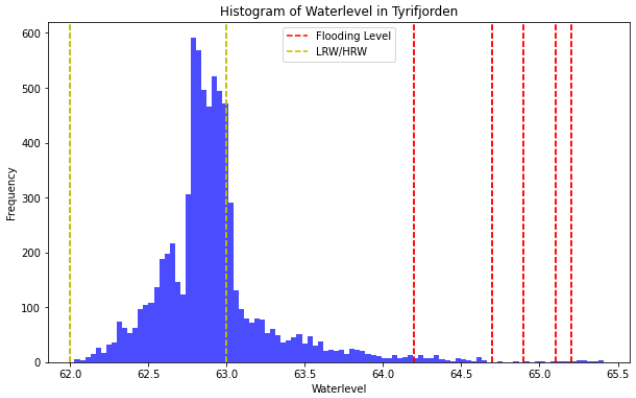
Condition	Days	Percent
Regulation Zone	5512	75,45517
Caution Zone	1653	22,62834
Mean to 5-Year Flood	86	1,177276
5 to 10-Year Flood	2	0,027379
10 to 20-Year Flood	6	0,082136
20 to 50-Year Flood	4	0,054757
50-Year Flood	15	0,205339
Total Flood Days	113	1,546886

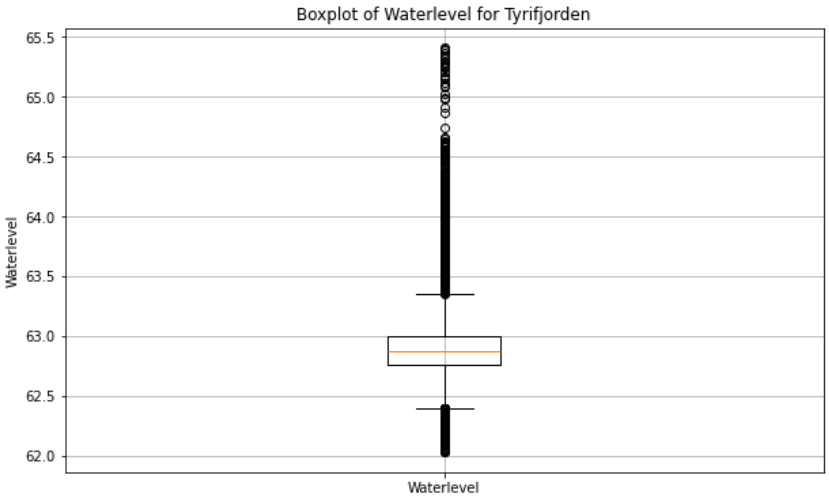


Appendix 5:

Statistical Analysis Tyrifjorden

Flood Level	Std Devs from Mean	Number of Std Devs	Percentage Within Range
Mean	3,39738879	1	78,80903491
Five-year	4,722526974	2	90,63655031
Ten-year	5,252582248	3	95,67419576
Twenty-year	5,782637522	4	98,38466804
Fifty-year	6,047665158	5	99,2881588
		6	99,58932238
		7	100
		8	100
		9	100
		10	100

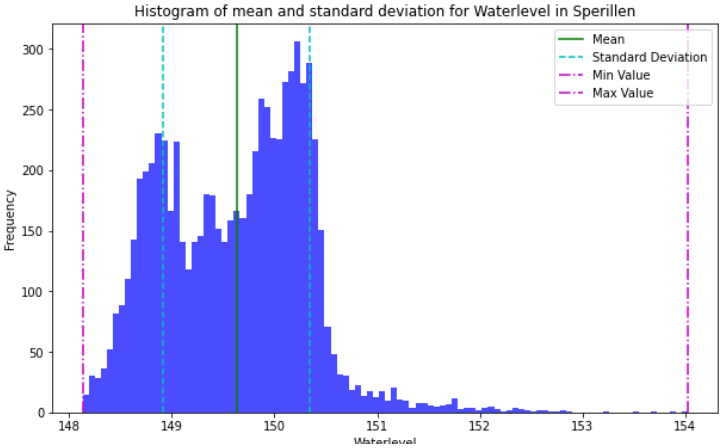


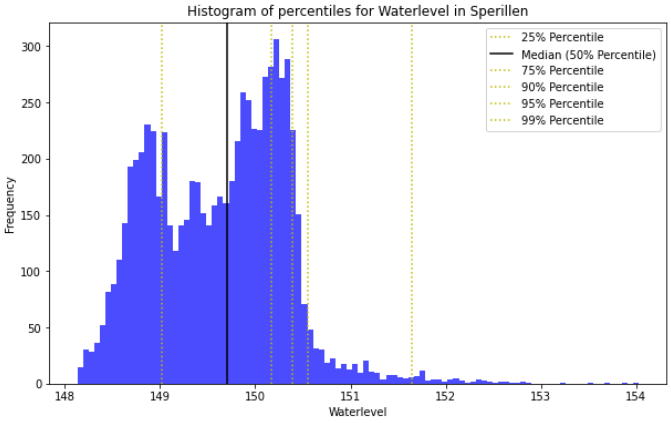


Appendix 6:

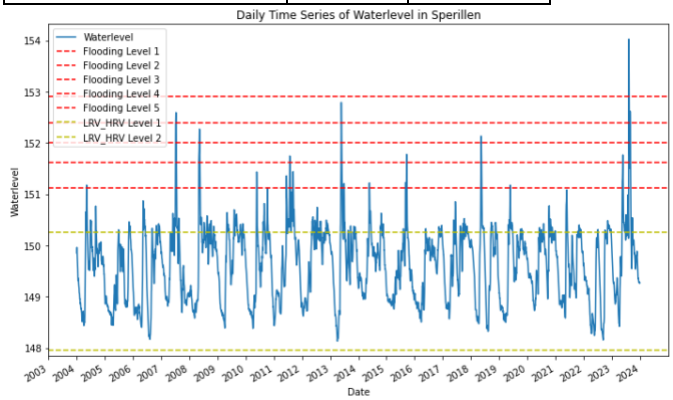
Statistical Analysis Sperillen

Statistic	Value
Total Data Points	7305
Mean	149,6315
Standard Deviation	0,71156
Min	148,1312
25%	149,0202
Median	149,7027
75%	150,1673
90%	150,3899
95%	150,5535
99%	151,6396
Max	154,023





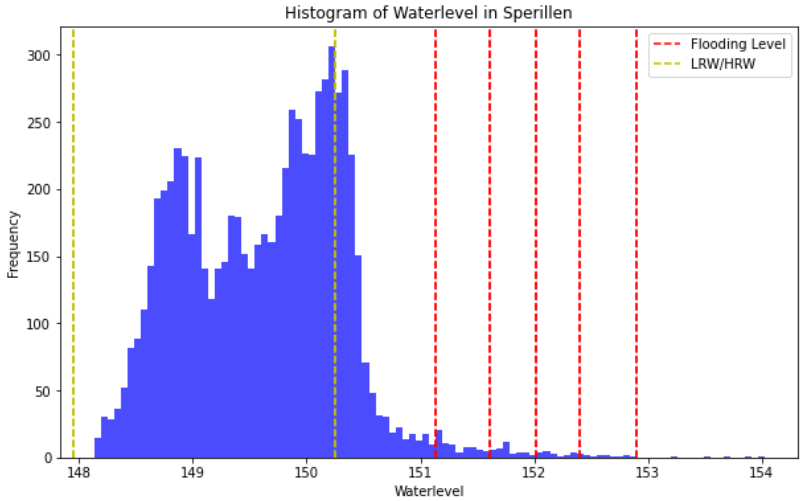
Condition	Days	Percent
Regulation Zone	5919	81,02669
Caution Zone	1233	16,87885
Mean to 5-Year Flood	74	1,013005
5 to 10-Year Flood	38	0,520192
10 to 20-Year Flood	21	0,287474
20 to 50-Year Flood	13	0,17796
50-Year Flood	5	0,068446
Total Flood Days	151	2,067077

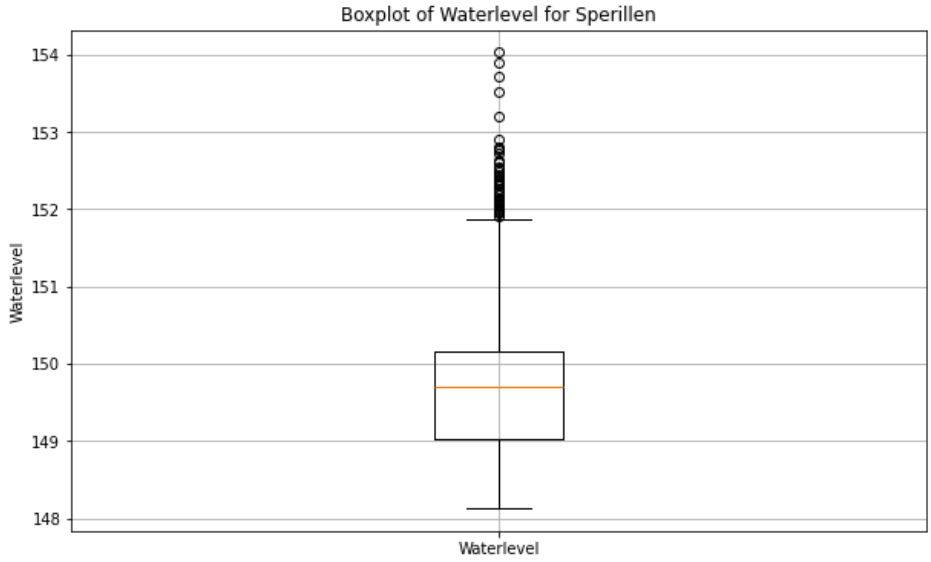


Appendix 6:

Statistical Analysis Sperillen

Flood Level	Std Devs from Mean	Number of Std Devs	Percentage Within Range
Mean	2,102542	1	73,85352
Five-year	2,784986	2	95,50992
Ten-year	3,347834	3	98,4668
Twenty-year	3,890726	4	99,56194
Fifty-year	4,598186	5	99,86311
		6	99,97262
		7	100
		8	100
		9	100
		10	100

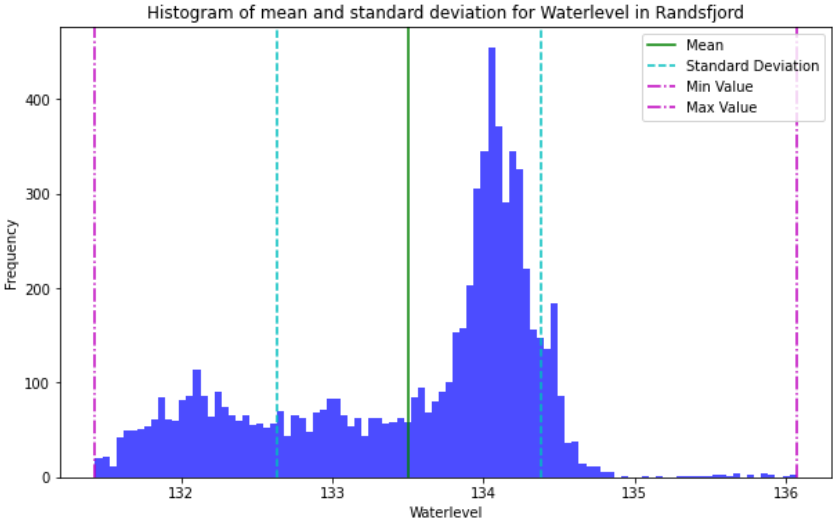


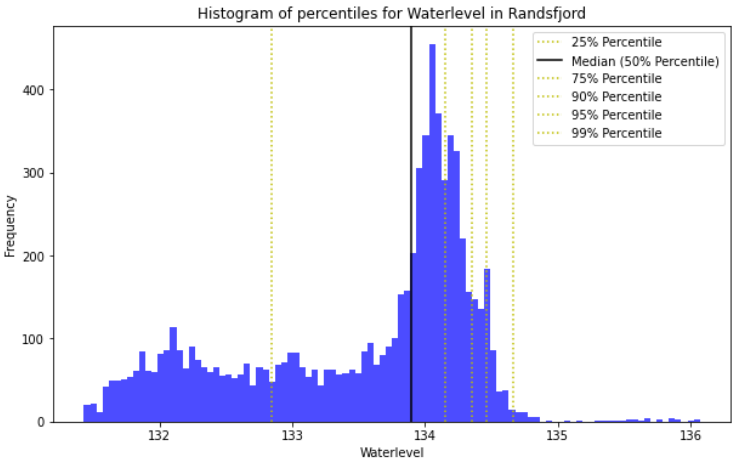


Appendix 7

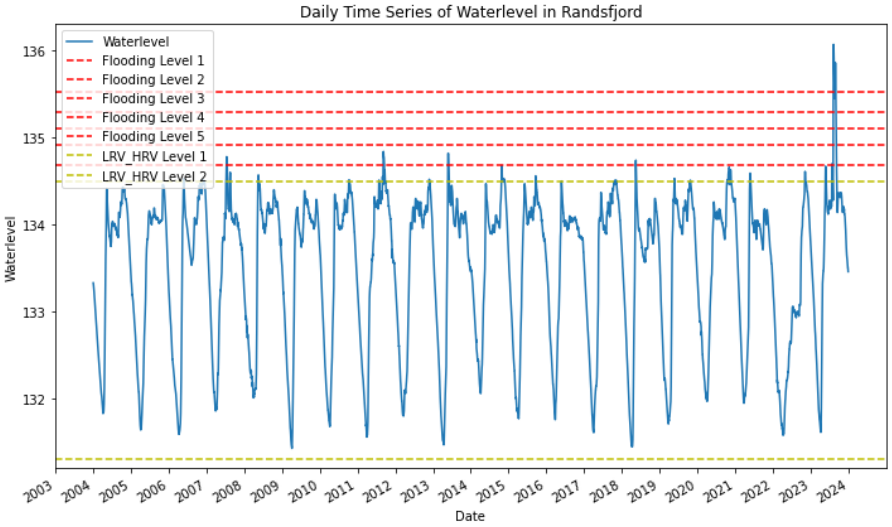
Statistical Analysis Randsfjorden

Statistic	Value
Total Data Points	7298
Mean	133,5049
Standard Deviation	0,871948
Min	131,43
25%	132,84
Median	133,9
75%	134,15
90%	134,35
95%	134,4618
99%	136,66
Max	136,07





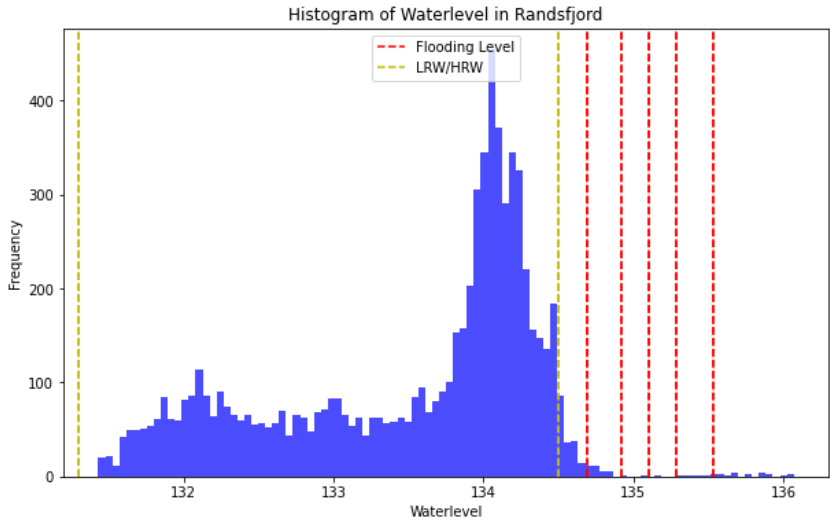
Condition	Days	Percent
Regulation Zone	7063	96,77994
Caution Zone	146	2,000548
Mean to 5-Year Flood	34	0,465881
5 to 10-Year Flood	2	0,027405
10 to 20-Year Flood	2	0,027405
20 to 50-Year Flood	5	0,068512
50-Year Flood	21	0,28775
Total Flood Days	64	0,876953

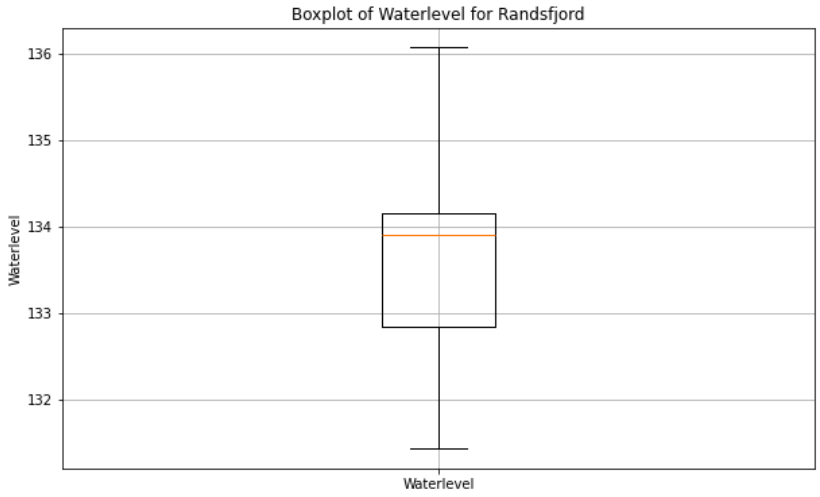


Appendix 7

Statistical Analysis Randsfjorden

Flood Level	Std Devs from Mean	Number of Std Devs	Percentage Within Range
Mean	1,358027	1	82,65278
Five-year	1,618249	2	99,26007
Ten-year	1,836038	3	100
Twenty-year	2,047518	4	100
Fifty-year	2,324943	5	100
		6	100
		7	100
		8	100
		9	100
		10	100





Appendix 8:

Seasonal_Analysis.py

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.linear_model import LinearRegression
import statsmodels.api as sm
from datetime import datetime
import sys
# Path to the CSV file
name = "Randsfjord"
data_type = "Waterlevel"
file_path = f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_{name}_Waterlevel.csv'
# Read the CSV file
df = pd.read_csv(file_path)
df[data_type] = pd.to_numeric(df[data_type], errors='coerce')  # Convert data to numeric, ensuring all data is correctly formatted
df['Date'] = pd.to_datetime(df['Date'])  # Ensure the Date column is in datetime format
# Define the date range to keep. The Range has been altered to match the start of a season. Given that the dataset begins mid Winter season, originally.
start_date = '2004-12-01'
end_date = '2023-12-31'
# Create a boolean mask to filter rows based on the date range
mask = (df['Date'] >= start_date) & (df['Date'] <= end_date)
# Apply the mask to filter rows within the specified date range
filtered_df = df[mask]
# Keep rows based on the boolean mask
df = df[mask]
df.set_index('Date', inplace=True)  # Set the Date column as the index for easier time series analysis
if name == 'Randsfjord':
    if data_type == 'Waterlevel':
        # Define flooding levels
        mean_flood = 134.689
        five_year_flood = 134.9159
        ten_year_flood = 135.1058
        twenty_year_flood = 135.2902
        fifty_year_flood = 135.5321
        LRV = 131.3
        HRV = 134.5
        LRV_HRV = [LRV, HRV]
        flooding_levels = [mean_flood, five_year_flood, ten_year_flood, twenty_year_flood, fifty_year_flood]
    else:
```

```
        # Handle the case when data_type is not 'Waterlevel'
        flooding_levels = None
elif name == 'Tyrifjorden':
    if data_type == 'Waterlevel':
        # Define flooding levels
        mean_flood = 64.2
        five_year_flood = 64.7
        ten_year_flood = 64.9
        twenty_year_flood = 65.1
        fifty_year_flood = 65.2
        LRV = 62
        HRV = 63
        LRV_HRV = [LRV, HRV]
        flooding_levels = [mean_flood, five_year_flood, ten_year_flood, twenty_year_flood, fifty_year_flood]
    else:
        # Handle the case when data_type is not 'Waterlevel'
        flooding_levels = None
elif name == 'Sperillen':
    if data_type == 'Waterlevel':
        # Define flooding levels
        mean_flood = 151.1276
        five_year_flood = 151.6132
        ten_year_flood = 152.0137
        twenty_year_flood = 152.4
        fifty_year_flood = 152.9034
        LRV = 147.95
        HRV = 150.25
        LRV_HRV = [LRV, HRV]
        flooding_levels = [mean_flood, five_year_flood, ten_year_flood, twenty_year_flood, fifty_year_flood]
df_reset = df.reset_index(inplace=False)
df_reset['Date'] = pd.to_datetime(df_reset['Date'])
df_reset['Date'] = df_reset['Date'].dt.strftime('%m/%d/%Y')
df_reset.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{name}/{name}_season_grouped.xlsx', index=True)
df_reset.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_season_grouped.csv', index=True)
decomposition = seasonal_decompose(df[data_type], model='additive', period=365)  # Using a period of 365 to account for
yearly seasonality
# Plotting the decomposition results
plt.figure(figsize=(14, 8))
plt.subplot(412)
plt.plot(decomposition.observed, label='Observed')
plt.legend(loc='upper right')
plt.subplot(413)
plt.plot(decomposition.seasonal,label='Seasonal')
plt.legend(loc='upper right')
```

```
plt.tight_layout()
plt.show()
# Define a function to assign seasons and handle crossover for winter
def assign_season(date):
    year = date.year
    spring_start = pd.Timestamp(year=year, month=3, day=1)
    summer_start = pd.Timestamp(year=year, month=6, day=1)
    autumn_start = pd.Timestamp(year=year, month=9, day=1)
    winter_start = pd.Timestamp(year=year, month=11, day=30)
    if date >= spring_start and date < summer_start:
        return 'Spring'
    elif date >= summer_start and date < autumn_start:
        return 'Summer'
    elif date >= autumn_start and date < winter_start:
        return 'Autumn'
    else:
        return 'Winter'
# Apply the season function to each date
df['Season'] = df.index.map(assign_season)
# Handle winter crossover: If it's January or February, assign it to the previous year's winter
df['Year'] = df.index.year
# Define the start of the winter season
winter_start_month = 12
winter_start_day = 1
# Custom function to calculate DayOfSeason
def calculate_day_of_season(row):
    # If the month is December, January, or February, it's winter
    if row.name.month == 12 or row.name.month <= 2:
        # Winter starts on December 1st
        season_start = pd.Timestamp(year=row.name.year if row.name.month == 12 else row.name.year-1,
                                    month=winter_start_month, day=winter_start_day)
    elif row.name.month >= 3 and row.name.month <= 5:
        # Spring starts on March 1st
        season_start = pd.Timestamp(year=row.name.year, month=3, day=1)
    elif row.name.month >= 6 and row.name.month <= 8:
        # Summer starts on June 1st
        season_start = pd.Timestamp(year=row.name.year, month=6, day=1)
    else:
        # Autumn starts on September 1st
        season_start = pd.Timestamp(year=row.name.year, month=9, day=1)
    # Calculate the DayOfSeason
    return (row.name - season_start).days + 1
# Apply the custom function to calculate DayOfSeason
df['DayOfSeason'] = df.apply(calculate_day_of_season, axis=1)
# Continue with the rest of your analysis...
```

```
# Print the first few rows of the dataframe to verify
print(df.head())
# Option to exit or move on
continue_choice = input("Move on? (yes/no): ").lower()
if continue_choice != 'yes':
    print("Exiting.")
    sys.exit()
# Group by season and calculate statistical summaries
statistics_seasonal_df = df.groupby('Season')['Waterlevel'].describe()
# If you want to round the statistics for cleaner presentation
statistics_seasonal_df = statistics_seasonal_df.round(2)
df['Season'] = df.index.map(assign_season)
# Plot histograms for each season with LRW, HRW, and flooding levels
fig, axes = plt.subplots(2, 2, figsize=(14, 10), tight_layout=True)
seasons = ['Spring', 'Summer', 'Autumn', 'Winter']
for ax, season in zip(axes.flatten(), seasons):
    season_data = df[df['Season'] == season][data_type]
    ax.hist(season_data, bins=40, alpha=0.7, label=f'{season} Distribution')
    ax.axvline(LRV, color='r', linestyle='dashed', linewidth=2, label='LRW')
    ax.axvline(HRV, color='g', linestyle='dashed', linewidth=2, label='HRW')
    # Add flooding levels
    for level in flooding_levels:
        ax.axvline(level, color='b', linestyle='dotted', linewidth=1)
    ax.set_title(f'{season} Water Level Distribution')
    ax.set_xlabel(data_type)
    ax.set_ylabel('Frequency')
    ax.legend()
plt.show()
# Define a function to calculate skewness and kurtosis
def calculate_skewness_kurtosis(data):
    skewness = data.skew()
    kurtosis = data.kurtosis()
    return pd.Series({'Skewness': skewness, 'Kurtosis': kurtosis})
# Calculate skewness and kurtosis for each season
skewness_kurtosis_seasonal = df.groupby('Season')['Waterlevel'].apply(calculate_skewness_kurtosis)
def count_days_within_ranges(df, season, levels, LRV):
    # Filter the dataframe for the specified season
    season_data = df[df['Season'] == season]
    # Initialize a dictionary to store counts
    counts = {}
    # Count days below the Lowest Reference Value (LRV)
    count_below_LRV = season_data[season_data[data_type] < LRV].shape[0]
    counts[f'Below {LRV}'] = count_below_LRV
    # Loop through the levels and count days within each range and above the last specified level
    for i in range(len(levels) - 1):
```

```
        lower_bound = levels[i]
        upper_bound = levels[i + 1]
        count = season_data[(season_data[data_type] > lower_bound) & (season_data[data_type] <= upper_bound)].shape[0]
        counts[f'{lower_bound} to {upper_bound}'] = count

    # Add count for days above the highest level specified
    highest_level = levels[-1]
    count_above_highest = season_data[season_data[data_type] > highest_level].shape[0]
    counts[f'Above {highest_level}'] = count_above_highest
    # Count total days above the first flood level (mean flood level)
    total_flood_days = season_data[season_data[data_type] > levels[2]].shape[0]  # Assuming levels[0] is the mean flood level
    counts['Total Flood Days'] = total_flood_days
    return counts
# Levels including HRW and flooding levels, ordered from lowest to highest criticality
levels = LRV_HRV + flooding_levels
levels.sort()
# Use the function
season_counts = {season: count_days_within_ranges(df, season, levels, LRV) for season in seasons}
# Convert the dictionary to a DataFrame for display
frequency_seasonal_df = pd.DataFrame(season_counts)  # Transpose for better readability
print(frequency_seasonal_df)
# Add a function to calculate the Seasonal Variability Index for each season
def calculate_SVI(season_data):
    mean_level = season_data.mean()
    std_dev = season_data.std()
    svi = std_dev / mean_level
    return svi
# Calculate the SVI for each season
risk_indicators_by_season = {}
for season in seasons:
    season_data = df[df['Season'] == season][data_type]
    svi = calculate_SVI(season_data)
    risk_indicators_by_season[season] = svi
# Convert the risk indicators dictionary to a DataFrame
SVI_df = pd.DataFrame(list(risk_indicators_by_season.items()), columns=['Season', 'SVI'])
# Create a boxplot for each season
plt.figure(figsize=(14, 6))  # Set the figure size (width, height) as desired
for i, season in enumerate(seasons):
    plt.subplot(1, len(seasons), i+1)  # Create subplots for each season
    seasonal_data = df[df['Season'] == season]
    plt.boxplot(seasonal_data[data_type])
    plt.title(season)
    plt.xlabel('Season')
    plt.ylabel(data_type)
plt.tight_layout()  # Adjust subplots to fit in the figure area
```

```
plt.show()
# Ensure the index is in datetime format, if it's not already
df.index = pd.to_datetime(df.index)
# Add a column for the year directly from the index
df['Year'] = df.index.year
# Initialize a linear regression model
model = LinearRegression()
seasons = df['Season'].unique()
# List to store the slope for each season
slopes = []
for season in seasons:
    # Extract all data points for the season across all years
    seasonal_data = df[df['Season'] == season]
    # The independent variable is the day of the season
    X = seasonal_data['DayOfSeason'].values.reshape(-1, 1)
    # The dependent variable is the water level
    y = seasonal_data['Waterlevel'].values
    # Fit the regression model
    model.fit(X, y)
    # Calculate the slope (coefficient)
    slope = model.coef_[0]
    slopes.append((season, slope))
    # Generate a sequence of day numbers for predictions
    X_pred = np.arange(1, seasonal_data['DayOfSeason'].max() + 1).reshape(-1, 1)
    y_pred = model.predict(X_pred)


# Initialize your model outside the loop
model = LinearRegression()
# Create a figure and a grid of subplots
fig, axs = plt.subplots(2, 2, figsize=(15, 10))
# Flatten the array of axes, for easy iteration
axs = axs.flatten()
# Iterate through each season and plot
for i, season in enumerate(['Winter', 'Spring', 'Summer', 'Autumn']):
    # Select the subplot where you want to plot the current season's trend
    ax = axs[i]
    # Extract all data points for the season across all years
    seasonal_data = df[df['Season'] == season]
    # The independent variable is the day of the season
    X = seasonal_data['DayOfSeason'].values.reshape(-1, 1)
    # The dependent variable is the water level
    y = seasonal_data['Waterlevel'].values
    # Fit the regression model
    model.fit(X, y)
```

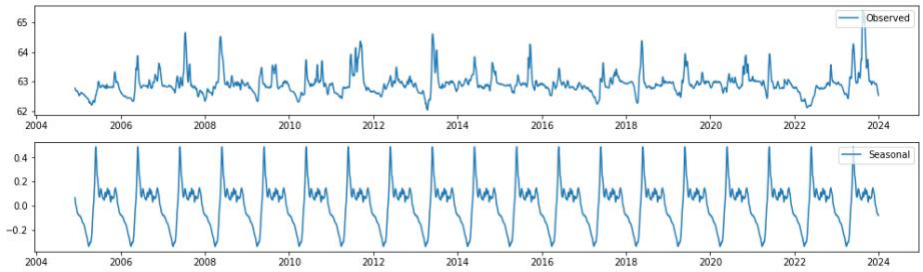
```
    # Calculate the slope (coefficient)
    slope = model.coef_[0]
    # Generate a sequence of day numbers for predictions
    X_pred = np.arange(1, seasonal_data['DayOfSeason'].max() + 1).reshape(-1, 1)
    y_pred = model.predict(X_pred)
    # Plot the actual data points and the regression line on the current subplot
    ax.scatter(seasonal_data['DayOfSeason'], y, alpha=0.5, label='Actual Data')
    ax.plot(X_pred, y_pred, color='black', label=f'Trend Line (slope: {slope:.5f})')
    ax.set_title(f'Trend for {season} Across All Years')
    ax.set_xlabel('Day of Season')
    ax.set_ylabel('Water Level')
    ax.legend()

# Adjust the layout so that all subplots fit into the figure neatly
plt.tight_layout()
plt.show()
# Convert the list of slopes to a DataFrame
slope_df = pd.DataFrame(slopes, columns=['Season', 'Slope'])
# Assuming 'DayOfSeason' and 'Waterlevel' are columns in your DataFrame, df.
seasons = df['Season'].unique()
slope_results = []
for season in seasons:
    # Extract all data points for the season across all years
    seasonal_data = df[df['Season'] == season]
    # The independent variable is the day of the season (add a constant term for intercept)
    X = sm.add_constant(seasonal_data['DayOfSeason'].values)
    # The dependent variable is the water level
    y = seasonal_data['Waterlevel'].values
    # Fit the regression model using OLS (Ordinary Least Squares)
    model = sm.OLS(y, X).fit()
    # Store the season, slope, p-value, and whether it's significant at alpha=0.05
    slope, p_value = model.params[1], model.pvalues[1]
    slope_results.append({
        'Season': season,
        'Slope': slope,
        'p-value': p_value,
        'Significant (p<0.05)': p_value < 0.05
    })
# Convert the results to a DataFrame
slope_results_df = pd.DataFrame(slope_results)
print(statistics_seasonal_df)
print(frequency_seasonal_df)
print(SVI_df)
print(slope_results_df)
print(skewness_kurtosis_seasonal)
```

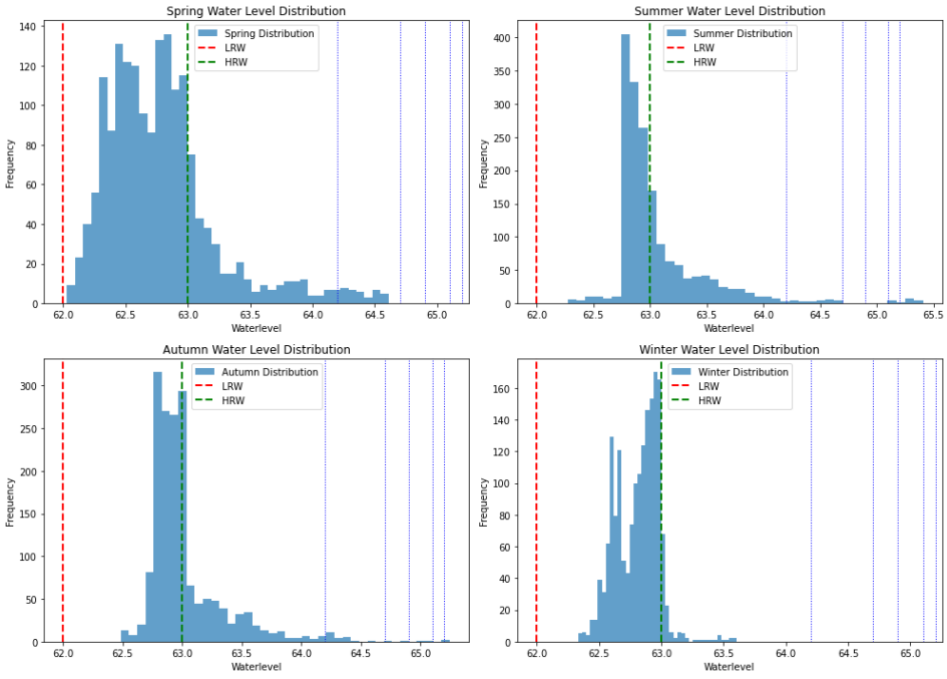
```
print()
# Define the save path
save_path = f'/Users/simen/Desktop/Complete Master/03 Excel Products/02 Lakes/{name}/'  # Update this path as needed
# File name based on a variable 'name'
file_name = f'Combined__Seasonal_Statistical_Data_{name}.xlsx'
# Full path including file name
full_path = save_path + file_name
# Save all DataFrames to an Excel file with each DataFrame as a separate sheet
with pd.ExcelWriter(full_path, engine='xlsxwriter') as writer:
    statistics_seasonal_df.to_excel(writer, sheet_name='Statistics', index=True)
    frequency_seasonal_df.to_excel(writer, sheet_name='Frequency Days', index=True)
    SVI_df.to_excel(writer, sheet_name='SVI', index=False)
    slope_results_df.to_excel(writer, sheet_name='Theortical Significance', index=False)
    skewness_kurtosis_seasonal.to_excel(writer, sheet_name='Skewness and Kurtosis', index=True)
print("All DataFrames have been saved as an excel file.")
# Save the statistical summary as CSV
statistics_seasonal_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_seasonal_statistics.csv', index=True)
# Save the frequency of flooding days as CSV
frequency_seasonal_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_seasonal_flooding_frequency.csv', index=True)
# Save the Seasonal Variability Index as CSV
SVI_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_seasonal_SVI.csv', index=False)
# Save the slopes of the trend analysis as CSV
slope_results_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_seasonal_trend_slopes.csv',
index=False)
# Save the skewness and kurtosis as CSV
skewness_kurtosis_seasonal.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_seasonal_skewness_kurtosis.csv', index=True)
# Print a message to confirm that files are saved
print("CSV files have been saved.")
```

Appendix 9

Seasonal Analysis Tyrifjorden

Season	count	mean	std	min	25%	50%	75%	max	SVI
Autumn	1710	63,02	0,33	62,48	62,83	62,94	63,05	65,25	0,005196
Spring	1748	62,79	0,45	62,03	62,47	62,73	62,97	64,61	0,007219
Summer	1748	63,07	0,42	62,27	62,82	62,92	63,14	65,41	0,006715
Winter	1764	62,81	0,17	62,33	62,66	62,84	62,94	63,61	0,002731

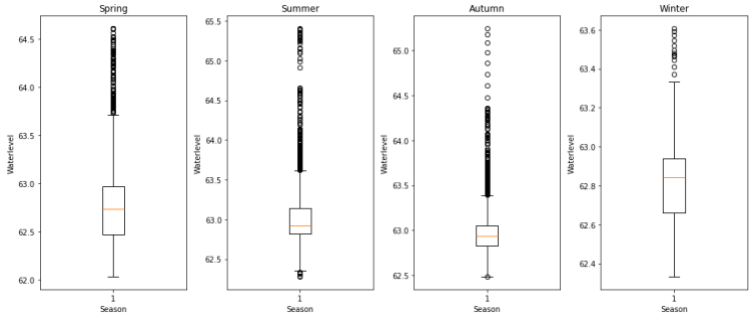


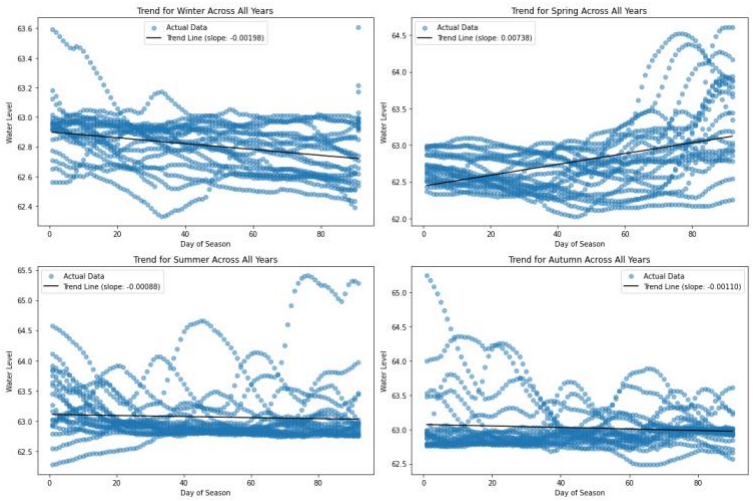


Appendix 9

Seasonal Analysis Tyrifjorden

Season		Waterlevel		Spring	Summer	Autumn	Winter
Autumn	**Skewness**	2,409307	**Below 62**	0	0	0	0
	Kurtosis	7,880773	**62 to 63**	1381	1123	1147	1633
Spring	**Skewness**	1,460868	**63 to 64.2**	329	578	535	131
	Kurtosis	2,754842	**64.2 to 64.7**	38	26	22	0
Summer	**Skewness**	2,546273	**64.7 to 64.9**	0	0	2	0
	Kurtosis	8,420227	**64.9 to 65.1**	0	4	2	0
Winter	**Skewness**	0,090666	**65.1 to 65.2**	0	3	1	0
	Kurtosis	0,906967	**Above 65.2**	0	14	1	0
			Total Flood Days	38	47	28	0



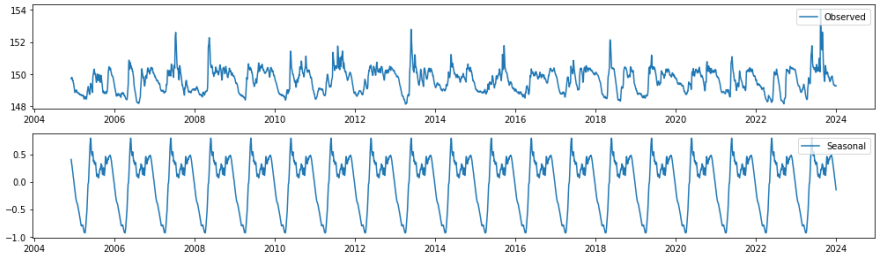


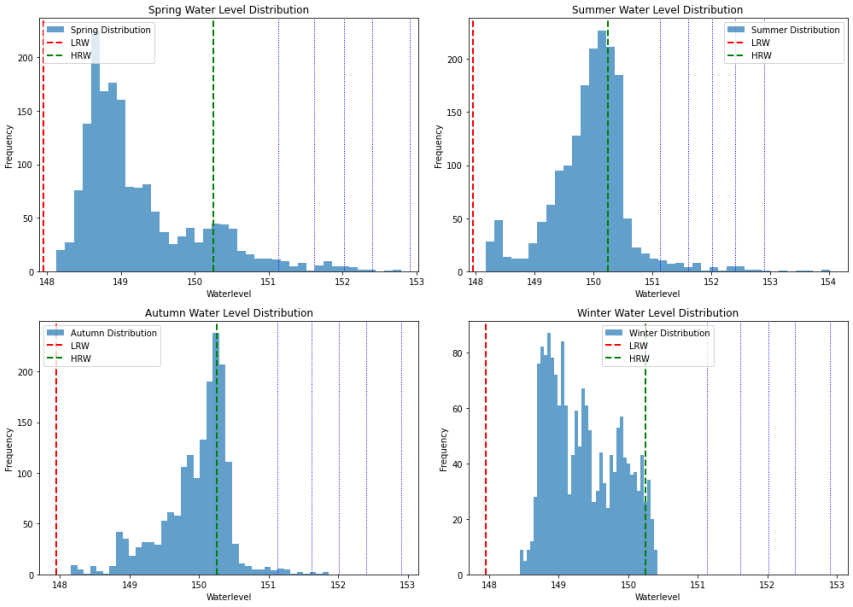
Season	Slope	p-value	Significant (p<0.05)
Winter	-0,00198	3,08E-39	TRUE
Spring	0,007381	1,18E-80	TRUE
Summer	-0,00088	0,02041	TRUE
Autumn	-0,0011	0,00029	TRUE

Appendix 10

Seasonal Analysis Sperillen

Season	count	mean	std	min	25%	50%	75%	max	SVI
Autumn	1710	149,96	0,5	148,15	149,74	150,09	150,28	151,87	0,003343
Spring	1748	149,27	0,81	148,13	148,69	148,98	149,6	152,79	0,005432
Summer	1748	149,95	0,69	148,17	149,62	150,01	150,31	154,02	0,004573
Winter	1764	149,36	0,5	148,44	148,92	149,29	149,81	150,42	0,003361

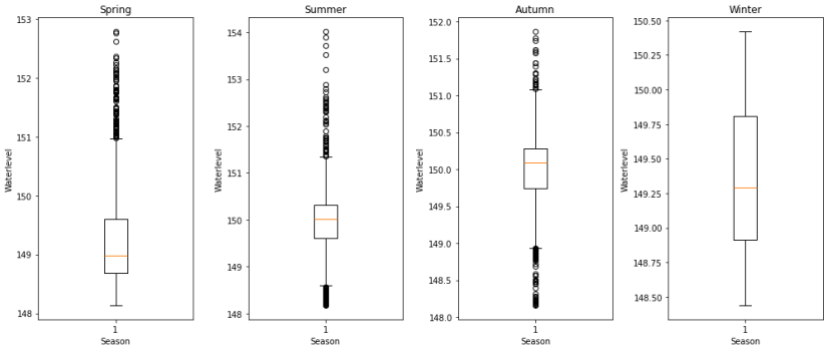




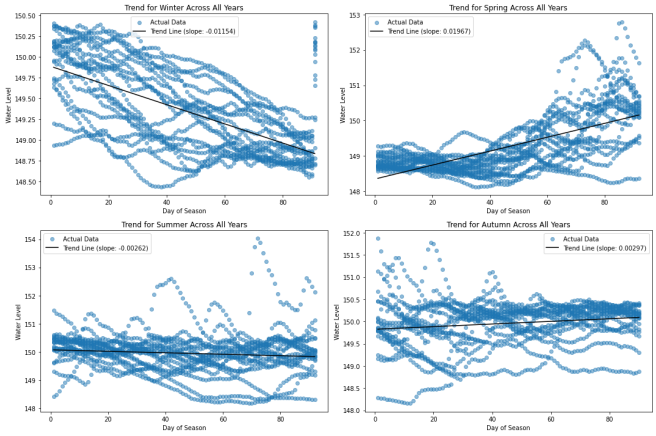
Season		Waterlevel
Autumn	**Skewness**	-0,75645
	Kurtosis	1,499028
Spring	**Skewness**	1,397661
	Kurtosis	1,703976
Summer	**Skewness**	0,512221
	Kurtosis	4,160338
Winter	**Skewness**	0,335115
	Kurtosis	-1,09509

Appendix 10

Seasonal Analysis Sperillen



	Spring	Summer	Autumn	Winter
Below 147.95	0	0	0	0
147.95 to 150.25	1494	1234	1216	1694
150.25 to 151.1276	188	452	474	70
151.1276 to 151.6132	29	26	16	0
151.6132 to 152.0137	23	11	4	0
152.0137 to 152.4	11	10	0	0
152.4 to 152.9034	3	10	0	0
Above 152.9034	0	5	0	0
Total Flood Days	66	62	20	0

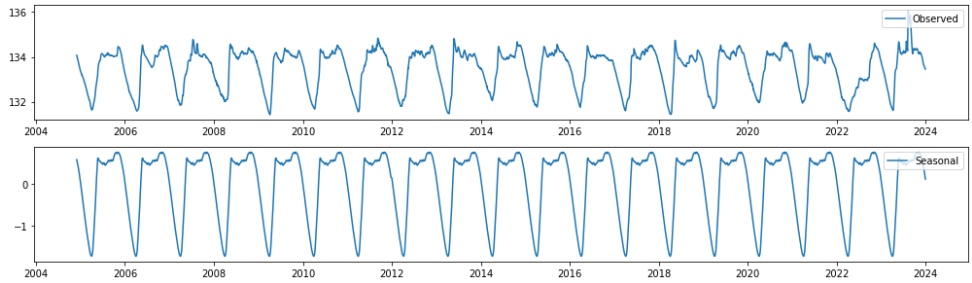


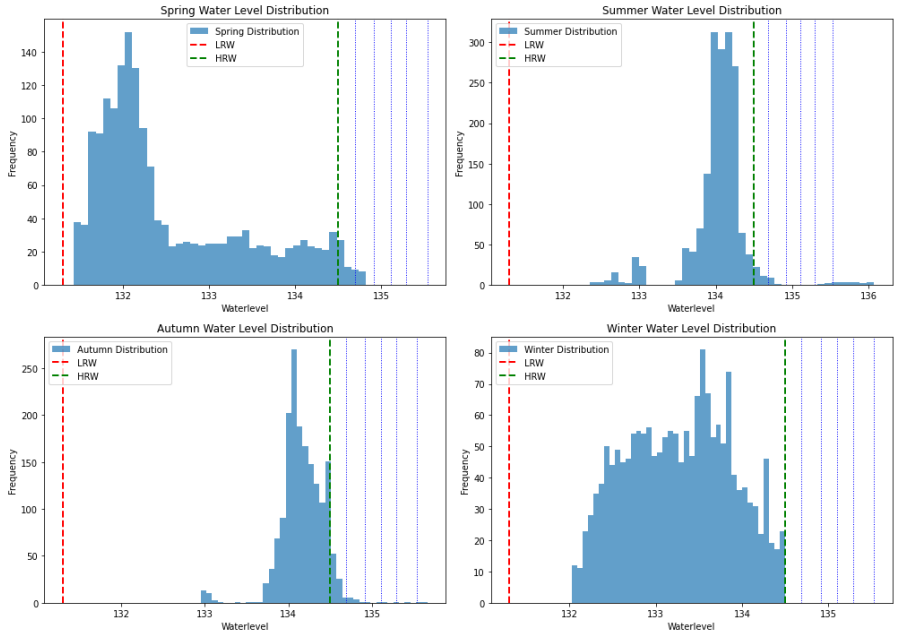
Season	Slope	p-value	Significant (p<0.05)
Winter	-0,01154	2,1E-178	TRUE
Spring	0,019665	1,3E-205	TRUE
Summer	-0,00262	2,18E-05	TRUE
Autumn	0,002967	1,62E-10	TRUE

Appendix 11

Seasonal Analysis Randsfjorden

Season	count	mean	std	min	25%	50%	75%	max	SVI
Autumn	1710	134,17	0,26	132,95	134,03	134,15	134,35	135,66	0,001963
Spring	1748	132,55	0,88	131,43	131,9	132,18	133,14	134,82	0,00664
Summer	1748	134,04	0,39	132,35	133,95	134,08	134,21	136,07	0,002912
Winter	1757	133,27	0,61	132,03	132,77	133,29	133,74	134,5	0,004553

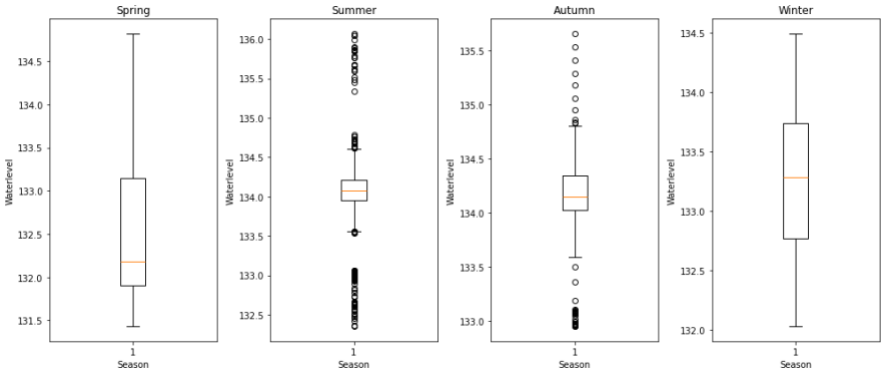




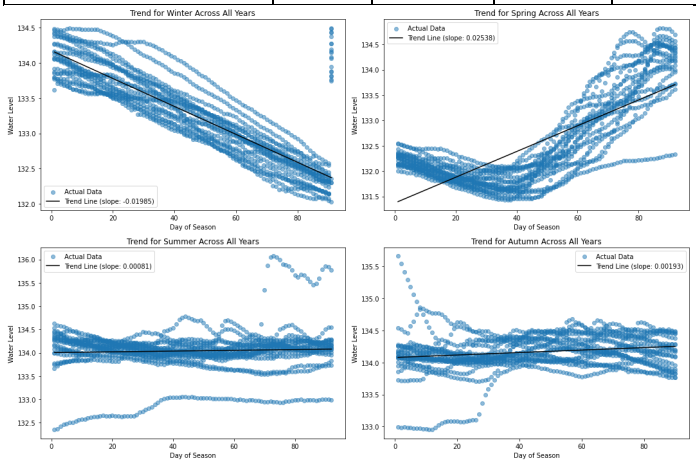
Season		Waterlevel
Autumn	**Skewness**	-0,70886
	Kurtosis	5,481842
Spring	**Skewness**	0,954235
	Kurtosis	-0,31941
Summer	**Skewness**	-0,36848
	Kurtosis	7,803884
Winter	**Skewness**	-0,00381
	Kurtosis	-0,9609

Appendix 11

Seasonal Analysis Randsfjorden



	Spring	Summer	Autumn	Winter
Below 131.3	0	0	0	0
131.3 to 134.5	1702	1686	1608	1757
134.5 to 134.689	34	29	83	0
134.689 to 134.9159	12	10	12	0
134.9159 to 135.1058	0	0	2	0
135.1058 to 135.2902	0	0	2	0
135.2902 to 135.5321	0	4	1	0
Above 135.5321	0	19	2	0
Total Flood Days	12	33	19	0



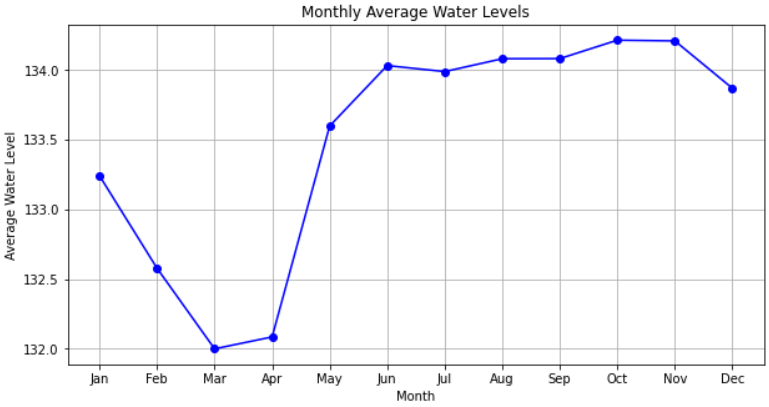
Season	Slope	p-value	Significant (p<0.05)
Winter	-0,01985	0	TRUE
Spring	0,025381	0	TRUE
Summer	0,000814	0,020487	TRUE
Autumn	0,001929	2,12E-15	TRUE

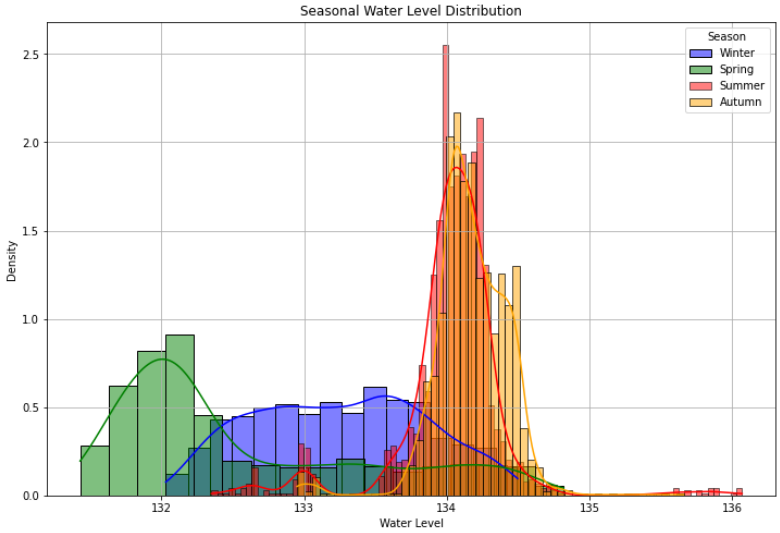
Appendix 14

Complete Multimodal Analysis

Randsfjorden:

Month	Waterlevel
1	133,2422
2	132,5779
3	131,9997
4	132,0852
5	133,5997
6	134,0334
7	133,9904
8	134,0827
9	134,0835
10	134,216
11	134,2108
12	133,8706



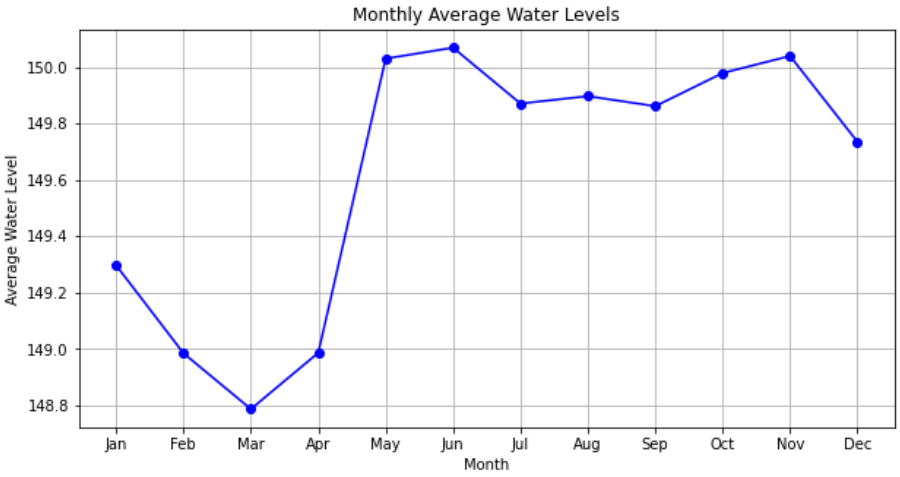


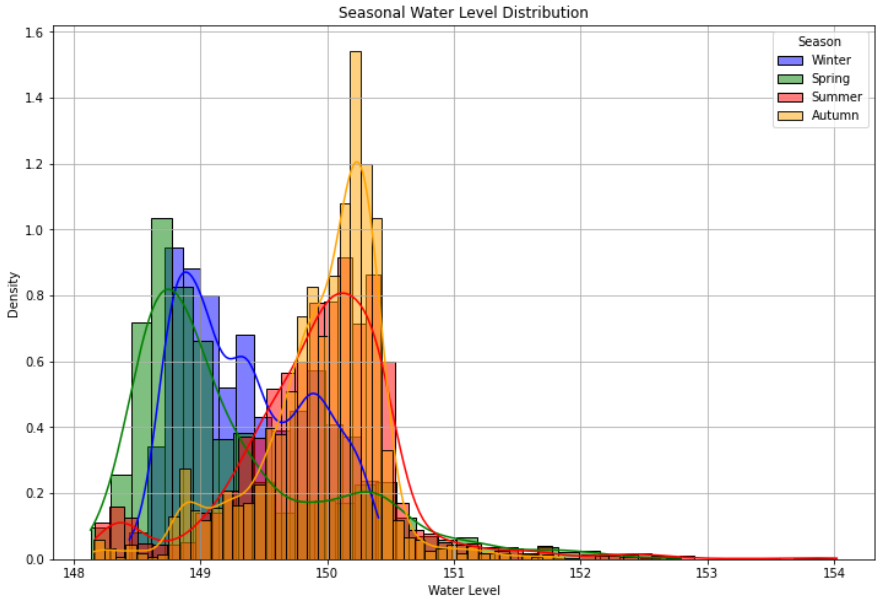
Appendix 14

Complete Multimodal Analysis

Sperillen:

Month	Waterlevel
1	149,297111
2	148,985516
3	148,787791
4	148,987182
5	150,029516
6	150,068158
7	149,870261
8	149,89644
9	149,86087
10	149,97837
11	150,038774
12	149,734727



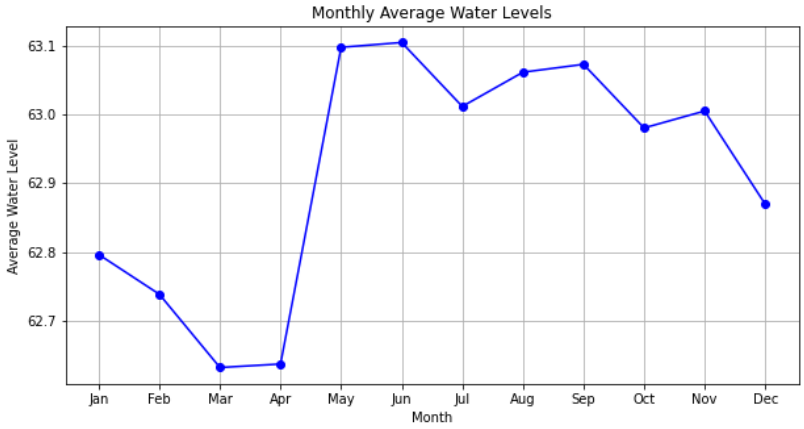


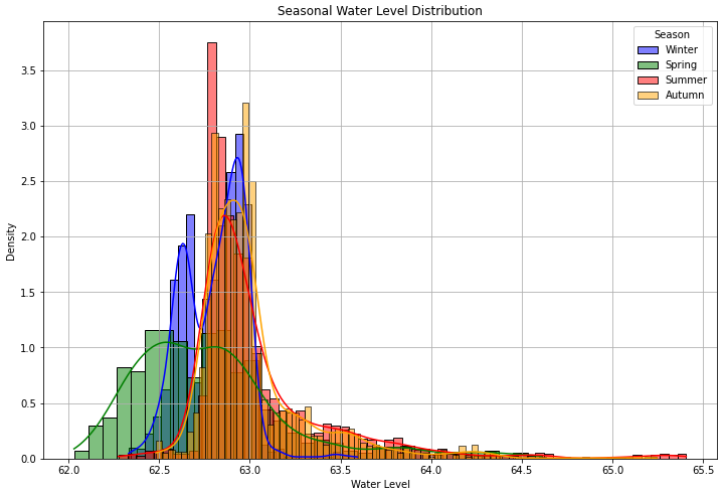
Appendix 14

Complete Multimodal Analysis

Tyrifjorden:

Month	Waterlevel
1	62,7961491
2	62,7383216
3	62,631753
4	62,6368141
5	63,0976748
6	63,1046604
7	63,0116149
8	63,0613906
9	63,0729041
10	62,980334
11	63,0050141
12	62,8693652





Appendix 13:

Yearly_plots.py

```
import pandas as pd
import matplotlib.pyplot as plt
# Load the CSV files
file1 = '/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_Randsfjord_Waterlevel.csv'
file2 = '/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_Tyrifjorden_Waterlevel.csv'
file3 = '/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_Sperillen_Waterlevel.csv'
# Importing CSV files
df1 = pd.read_csv(file1, parse_dates=['Date'])
df2 = pd.read_csv(file2, parse_dates=['Date'])
df3 = pd.read_csv(file3, parse_dates=['Date'])
# Extracting the year from the Date column
df1['Year'] = df1['Date'].dt.year
df2['Year'] = df2['Date'].dt.year
df3['Year'] = df3['Date'].dt.year
# Extracting unique years for plotting
years = sorted(set(df1['Year']).union(set(df2['Year'])).union(set(df3['Year'])))
# Creating plots for each year
for year in years:
    fig, axes = plt.subplots(3, 1, figsize=(10, 15), sharex=True)
    # Plot for lake1
    lake1_data = df1[df1['Year'] == year]
    axes[0].plot(lake1_data['Date'], lake1_data['Waterlevel'])
    axes[0].set_title(f'Randsfjorden Water Level in Year {year}')
    axes[0].set_ylabel('Water Level')
    # Plot for lake2
    lake2_data = df2[df2['Year'] == year]
    axes[1].plot(lake2_data['Date'], lake2_data['Waterlevel'])
    axes[1].set_title(f'Tyrifjorden Water Level in Year {year}')
    axes[1].set_ylabel('Water Level')
    # Plot for lake3
    lake3_data = df3[df3['Year'] == year]
    axes[2].plot(lake3_data['Date'], lake3_data['Waterlevel'])
    axes[2].set_title(f'Sperillen Water Level in Year {year}')
    axes[2].set_ylabel('Water Level')
    axes[2].set_xlabel('Date')
    dates_to_mark = [f'{year}-11-30', f'{year}-03-01', f'{year}-06-01', f'{year}-09-01']
    for ax in axes:
        for date in dates_to_mark:
            ax.axvline(pd.to_datetime(date), color='r', linestyle='--')
    plt.tight_layout()
    plt.show()
```

Appendix 14

Multimodal_analysis.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import kruskal
from sklearn.mixture import GaussianMixture
import warnings
from scipy.stats import gaussian_kde
from scipy.integrate import quad
# Suppress specific sklearn UserWarnings
warnings.simplefilter("ignore", category=UserWarning)
warnings.simplefilter("ignore", category=FutureWarning)
name = 'Randsfjord'
# Modify this line to match the exact file name shown in the uploaded.keys()
file_path =  f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_{name}_Waterlevel.csv'
data = pd.read_csv(file_path)
# Convert the 'Date' column to datetime format for easier manipulation
data['Date'] = pd.to_datetime(data['Date'])
print("Date column converted to datetime.")
# Create additional columns for analysis
data['Month'] = data['Date'].dt.month
data['Season'] = data['Month'].apply(lambda x: 'Winter' if x in [12, 1, 2] else
                                     'Spring' if x in [3, 4, 5] else
                                     'Summer' if x in [6, 7, 8] else 'Autumn')
# Calculate monthly average water levels to see seasonal variations
monthly_averages = data.groupby('Month')['Waterlevel'].mean()
print("Monthly averages of water levels:")
print(monthly_averages)
# Compute descriptive statistics for each season
seasonal_stats = data.groupby('Season')['Waterlevel'].describe()
# Convert the descriptive statistics into a DataFrame
stats_df = pd.DataFrame(seasonal_stats)
# Save the statistics DataFrame to an Excel file
excel_path = f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{name}/{name}_Seasonal_Waterlevel_Stats.xlsx'
stats_df.to_excel(excel_path)
# Make sure monthly_averages is a DataFrame
monthly_averages_df = pd.DataFrame(monthly_averages).reset_index()
# Save the monthly averages DataFrame to an Excel file
monthly_averages_excel_path = f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{name}/{name}_Monthly_Averages_Waterlevel.xlsx'
```

```
monthly_averages_df.to_excel(monthly_averages_excel_path, index=False)
# Assuming you already have monthly_averages calculated from your groupby operation
monthly_averages_df = pd.DataFrame(monthly_averages).reset_index()
# Plot the monthly averages
plt.figure(figsize=(10, 5))
plt.plot(monthly_averages_df['Month'], monthly_averages_df['Waterlevel'], marker='o', linestyle='-', color='b')
plt.title('Monthly Average Water Levels')
plt.xlabel('Month')
plt.ylabel('Average Water Level')
plt.xticks(monthly_averages_df['Month'], ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'])
plt.grid(True)
plt.savefig(f'/Users/simen/Desktop/Complete Master/04 Plots/{name}_monthly_avg.png')
plt.show()
# Define colors for each season for consistency
season_colors = {
    'Winter': 'blue',
    'Spring': 'green',
    'Summer': 'red',
    'Autumn': 'orange'
}
# Plot histogram using Seaborn
plt.figure(figsize=(12, 8))
for season, color in season_colors.items():
    # Select the season
    season_data = data[data['Season'] == season]
    # Plot the data with the season-specific color
    sns.histplot(season_data, x="Waterlevel", stat="density", kde=True, color=color, label=season)
plt.title('Seasonal Water Level Distribution')
plt.xlabel('Water Level')
plt.ylabel('Density')
# Create the legend with the defined colors
plt.legend(title='Season')
plt.grid(True)
plt.savefig(f'/Users/simen/Desktop/Complete Master/04 Plots/{name}_multimodal_histogram.png')
plt.show()
# Kruskal-Wallis Test across seasons
winter_levels = data[data['Season'] == 'Winter']['Waterlevel']
spring_levels = data[data['Season'] == 'Spring']['Waterlevel']
summer_levels = data[data['Season'] == 'Summer']['Waterlevel']
autumn_levels = data[data['Season'] == 'Autumn']['Waterlevel']
kruskal_result = kruskal(winter_levels, spring_levels, summer_levels, autumn_levels)
print(f"Kruskal-Wallis test result: H-statistic = {kruskal_result.statistic}, p-value = {kruskal_result.pvalue}")
```

Appendix 15:

Reservoir_to_DailyEnergy.py

```
import pandas as pd
import matplotlib.pyplot as plt
# Define the energy equivalents for each power station (in kWh/m^3).
energy_equivalents = {
    'Tyrifjorden_Geithusfoss': 0.025,
    'Tyrifjorden_Gravfoss_one': 0.044,
    'Tyrifjorden_Gravfoss_two': 0.048,
    'Sperillen_Hensfoss': 0.055,
    'Sperillen_Begna': 0.018,
    'Sperillen_Hofsfoss': 0.061,
    'Sperillen_Hoenefoss': 0.051,
    'Randsfjord_Bergerfoss': 0.013,
    'Randsfjord_Kistefoss_one': 0.018,
    'Randsfjord_Kistefoss_two': 0.025,
    'Randsfjord_Askerudfoss': 0.048,
    'Randsfjord_Viulfoss': 0.042
}

def read_and_prepare_data(file_path):
    df = pd.read_csv(file_path)
    df['Reservoir'] = pd.to_numeric(df['Reservoir'], errors='coerce')
    df['Date'] = pd.to_datetime(df['Date'])

    # Count the number of negative values
    negative_count = (df['Reservoir'] < 0).sum()

    # Remove negative 'Reservoir' values or set them to zero
    df.loc[df['Reservoir'] < 0, 'Reservoir'] = 0

    df.dropna(subset=['Reservoir'], inplace=True)
    df.reset_index(drop=True, inplace=True)

    return df, negative_count
# File paths, has to be adjust if another computer is used.
file_paths = {
    'Tyrifjorden': '/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_Tyrifjorden_Reservoir.csv',
    'Sperillen': '/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_Sperillen_Reservoir.csv',
    'Randsfjord': '/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_Randsfjord_Reservoir.csv',
}
# Process each dataset and plot
for title, path in file_paths.items():
```

```
    df, negative_count = read_and_prepare_data(path)
    print(f"Number of negative entries removed/set to zero for {title}: {negative_count}")
# Process the Randsfjord dataset and create a new dataframe with daily energy equivalent calculations
df_Tyrifjorden, _ = read_and_prepare_data(file_paths['Tyrifjorden'])
# Calculate the daily energy equivalent for all Energy in Randsfjord
df_energy_Tyrifjorden_calc = df_Tyrifjorden.copy()
df_energy_Tyrifjorden_calc['Geithusfoss [GWh]'] = df_energy_Tyrifjorden_calc['Reservoir'] *
energy_equivalents['Tyrifjorden_Geithusfoss']
df_energy_Tyrifjorden_calc['Gravfoss 1 [GWh]'] = df_energy_Tyrifjorden_calc['Reservoir'] *
energy_equivalents['Tyrifjorden_Gravfoss_one']
df_energy_Tyrifjorden_calc['Gravfoss 2 [GWh]'] = df_energy_Tyrifjorden_calc['Reservoir'] *
energy_equivalents['Tyrifjorden_Gravfoss_two']
df_energy_Tyrifjorden_calc['DailyEnergy [GWh]'] = df_energy_Tyrifjorden_calc['Geithusfoss [GWh]'] +
df_energy_Tyrifjorden_calc['Gravfoss 1 [GWh]'] + df_energy_Tyrifjorden_calc['Gravfoss 2 [GWh]']
# Process the Tyrifjorden dataset and create a new dataframe with daily energy equivalent calculations
df_Tyrifjorden, _ = read_and_prepare_data(file_paths['Tyrifjorden'])
# Calculate the daily energy equivalent for Tyrifjorden / Geithusfoss Kraftverk
df_energy_Tyrifjorden = df_Tyrifjorden.copy()
df_energy_Tyrifjorden['DailyEnergy [GWh]'] = df_energy_Tyrifjorden_calc['DailyEnergy [GWh]']
# Process the Randsfjord dataset and create a new dataframe with daily energy equivalent calculations
df_Sperillen, _ = read_and_prepare_data(file_paths['Sperillen'])
# Calculate the daily energy equivalent for all Energy in Randsfjord
df_energy_Sperillen_calc = df_Sperillen.copy()
df_energy_Sperillen_calc['Hensfoss [GWh]'] = df_energy_Sperillen_calc['Reservoir'] *
energy_equivalents['Sperillen_Hensfoss']
df_energy_Sperillen_calc['Begna [GWh]'] = df_energy_Sperillen_calc['Reservoir'] * energy_equivalents['Sperillen_Begna']
df_energy_Sperillen_calc['Hofsfoss [GWh]'] = df_energy_Sperillen_calc['Reservoir'] *
energy_equivalents['Sperillen_Hofsfoss']
df_energy_Sperillen_calc['Hoenefoss [GWh]'] = df_energy_Sperillen_calc['Reservoir'] *
energy_equivalents['Sperillen_Hoenefoss']
df_energy_Sperillen_calc['DailyEnergy [GWh]'] = df_energy_Sperillen_calc['Hensfoss [GWh]'] +
df_energy_Sperillen_calc['Begna [GWh]'] + df_energy_Sperillen_calc['Hofsfoss [GWh]'] +
df_energy_Sperillen_calc['Hoenefoss [GWh]']
# Process the Sperillen dataset and create a new dataframe with daily energy equivalent calculations
df_Sperillen, _ = read_and_prepare_data(file_paths['Sperillen'])
# Calculate the daily energy equivalent for Sperillen / Hensfoss Kraftverk
df_energy_Sperillen = df_Sperillen.copy()
df_energy_Sperillen['DailyEnergy [GWh]'] = df_energy_Sperillen_calc['DailyEnergy [GWh]']
# Process the Randsfjord dataset and create a new dataframe with daily energy equivalent calculations
df_Randsfjord, _ = read_and_prepare_data(file_paths['Randsfjord'])
# Calculate the daily energy equivalent for all Energy in Randsfjord
df_energy_Randsfjord_calc = df_Randsfjord.copy()
df_energy_Randsfjord_calc['Bergerfoss [GWh]'] = df_energy_Randsfjord_calc['Reservoir'] *
energy_equivalents['Randsfjord_Bergerfoss']
```

```
df_energy_Randsfjord_calc['Kistefoss 1 [GWh]'] = df_energy_Randsfjord_calc['Reservoir'] *
energy_equivalents['Randsfjord_Kistefoss_one']
df_energy_Randsfjord_calc['Kistefoss 2 [GWh]'] = df_energy_Randsfjord_calc['Reservoir'] *
energy_equivalents['Randsfjord_Kistefoss_two']
df_energy_Randsfjord_calc['Askerudfoss [GWh]'] = df_energy_Randsfjord_calc['Reservoir'] *
energy_equivalents['Randsfjord_Askerudfoss']
df_energy_Randsfjord_calc['Viulfoss [GWh]'] = df_energy_Randsfjord_calc['Reservoir'] *
energy_equivalents['Randsfjord_Viulfoss']
df_energy_Randsfjord_calc['DailyEnergy [GWh]'] = df_energy_Randsfjord_calc['Viulfoss [GWh]'] +
df_energy_Randsfjord_calc['Askerudfoss [GWh]'] + df_energy_Randsfjord_calc['Kistefoss 2 [GWh]'] +
df_energy_Randsfjord_calc['Kistefoss 1 [GWh]'] + df_energy_Randsfjord_calc['Bergerfoss [GWh]']
# Process the Sperillen dataset and create a new dataframe with daily energy equivalent calculations
df_Randsfjord, _ = read_and_prepare_data(file_paths['Randsfjord'])
# Calculate the daily energy equivalent for all Energy in Randsfjord
df_energy_Randsfjord = df_Randsfjord.copy()
df_energy_Randsfjord['DailyEnergy [GWh]'] = df_energy_Randsfjord_calc['DailyEnergy [GWh]']
print(df_energy_Randsfjord.head())
print(df_energy_Sperillen.head())
print(df_energy_Tyrifjorden.head())
# Drop 'Reservoir' column and rename 'DailyEnergy [GWh]' to 'Energy_GWh' for df_energy_Randsfjord
df_energy_Randsfjord = df_energy_Randsfjord.drop(['Reservoir'], axis=1)
df_energy_Randsfjord = df_energy_Randsfjord.rename(columns={'DailyEnergy [GWh]': 'Energy'})
print('-------------------------')
print(df_energy_Randsfjord.head())
df_energy_Randsfjord.to_csv('/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Randsfjord_Energy_Daily.csv',
index=False)
# Drop 'Reservoir' column and rename 'DailyEnergy [GWh]' to 'Energy_GWh' for df_energy_Sperillen
df_energy_Sperillen = df_energy_Sperillen.drop(['Reservoir'], axis=1)
df_energy_Sperillen = df_energy_Sperillen.rename(columns={'DailyEnergy [GWh]': 'Energy'})
print('-------------------------')
print(df_energy_Sperillen.head())
df_energy_Sperillen.to_csv('/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Sperillen_Energy_Daily.csv',
index=False)
# Drop 'Reservoir' column and rename 'DailyEnergy [GWh]' to 'Energy_GWh' for df_energy_Tyrifjorden
df_energy_Tyrifjorden = df_energy_Tyrifjorden.drop(['Reservoir'], axis=1)
df_energy_Tyrifjorden = df_energy_Tyrifjorden.rename(columns={'DailyEnergy [GWh]': 'Energy'})
print('-------------------------')
print(df_energy_Tyrifjorden.head())
df_energy_Tyrifjorden.to_csv('/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/Tyrifjorden_Energy_Daily.csv', index=False)
```

Appendix 16:

Data_Preperation.py

```
import pandas as pd
import pandas as pd
# Path to your CSV files
name = "Randsfjord"
file_path_reservoirlevel = f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_{name}_Reservoir.csv'
file_path_waterlevel = f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_{name}_Waterlevel.csv'
file_path_energy = f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_Energy_Daily.csv'
file_path_waterflow = f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_{name}_Waterflow.csv'

# Read the CSV files
df_reservoirlevel = pd.read_csv(file_path_reservoirlevel, delimiter=',', header=0, parse_dates=['Date'])
df_waterlevel = pd.read_csv(file_path_waterlevel, delimiter=',', header=0, parse_dates=['Date'])
df_energy = pd.read_csv(file_path_energy, delimiter=',', header=0, parse_dates=['Date'])
df_waterflow = pd.read_csv(file_path_waterflow, delimiter=',', header=0, parse_dates=['Date'])

print(df_reservoirlevel.head())
print(df_energy.head())
print(df_waterlevel.head())
print(df_waterflow.head())

# Merge df_reservoirlevel and df_waterlevel
combined_df = pd.merge(df_reservoirlevel, df_waterlevel, on='Date', how='outer', suffixes=('_reservoir', '_water'))
# Merge the result with df_energy
combined_df = pd.merge(combined_df, df_energy, on='Date', how='outer')
combined_df = pd.merge(combined_df, df_waterflow, on='Date', how='outer')
df_total = combined_df

# Now, combined_df contains all the combined information. You can print the head to check
print(df_total.head())
print(df_total.tail())
df_total.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_Total_Daily.csv', index=False)

# Load your dataset
df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/Cleaned_{name}_Waterlevel.csv')

# Ensure the 'Date' column is in datetime format
df['Date'] = pd.to_datetime(df['Date'])

# Define your start date for the dataset, in order to get full seasons. the dataset starts mid winter, january 1st
start_date = '2004-03-01'
# Filter the dataset to start from the first spring season of 2004
```

```
df_filtered = df[df['Date'] >= pd.to_datetime(start_date)]
df = df_filtered

# Define a function to categorize dates into seasons
def get_season(date):
    if date.month in [12, 1, 2]:
        return 'Winter'
    elif date.month in [3, 4, 5]:
        return 'Spring'
    elif date.month in [6, 7, 8]:
        return 'Summer'
    elif date.month in [9, 10, 11]:
        return 'Autumn'

# Apply the function to create a 'Season' column
df['Season'] = df['Date'].apply(get_season)

# Split the data into seasons
winter_df = df[df['Season'] == 'Winter']
spring_df = df[df['Season'] == 'Spring']
summer_df = df[df['Season'] == 'Summer']
autumn_df = df[df['Season'] == 'Autumn']

# Save each season's data to a new CSV file
winter_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_winter_waterlevel_df.csv',
index=False)
spring_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_spring_waterlevel_df.csv',
index=False)
summer_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_summer_waterlevel_df.csv',
index=False)
autumn_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_autumn_waterlevel_df.csv',
index=False)
print("Datasets have been split into seasons and saved as separate CSV files.")
slope_file_path = f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_seasonal_adjustment.csv'
# Read the CSV file
df_slope = pd.read_csv(slope_file_path)
df_slope = df_slope.drop(['Skewness', 'Kurtosis','SVI'], axis=1)
# Save the slopes of the trend analysis as CSV
df_slope.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_seasonal_trend.csv', index=False)
```

Appendix 17:

States_Constructor.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
# Suppress specific sklearn UserWarnings
warnings.simplefilter("ignore", category=UserWarning)
# Load the dataset
name = 'Tyrifjorden'
df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_Total_Daily.csv')
print(df.head())
df.dropna(inplace=True)  # Drop NA values
df['Waterlevel'] = pd.to_numeric(df['Waterlevel'], errors='coerce')
# Convert 'Date' to datetime if needed
df['Date'] = pd.to_datetime(df['Date'])
# Setting the 'Date' column as the index for easier plotting
df.set_index('Date', inplace=True)
print(name)
if name == 'Randsfjord':
    water_levels = [
        130.5718,  # Extended Low Water
        131.51356985708998,  # Extended Low Energy
        131.43,  # Low Observed Water
        131.6159,  # Low Observed Energy
        131.6623,  # Extended 1st percentile Energy
        131.68,  # 1st percentile Energy
        131.8302,  # Extended 5th percentile Energy
        131.88,  # 5th percentile Energy
        132.0365,  # Extended 10th percentile Energy
        132.08,  # 10th percentile Energy
        132.7427,  # Extended 25th percentile Energy
        132.85,  # 25th percentile Energy
        133.5323,  # Mean Energy
        133.5048011787279,  # Mean Water
        134.5,  # HRV
        134.689,  # Mean Flood
        134.9159,  # 5-Year Flood
        135.1058,  # 10-Year Flood
        135.2902,  # 20-Year Flood
        135.5321,  # 50-Year Flood
        136.07,  # High Observed Water
```

```
        136.9281,  # Extended High Water
    ]
elif name == 'Tyrifjorden':
    water_levels = [
        61.6587,  # Extended Low Water
        62.016221473098916,  # Extended Low Energy
        62.02999,  # Low Observed Water
        62.1460,  # Low Observed Energy
        62.1965,  # Extended 1st percentile Energy
        62.29,  # 1st percentile Energy
        62.3788,  # Extended 5th percentile Energy
        62.46,  # 5th percentile Energy
        62.5273,  # Extended 10th percentile Energy
        62.57,  # 10th percentile Energy
        62.7795,  # Extended 25th percentile Energy
        62.80,  # 25th percentile Energy
        62.8956,  # Mean Energy
        62.9181001834363,  # Mean Water
        63.00,  # HRV
        64.2,  # Mean Flood
        64.7,  # 5-Year Flood
        64.9,  # 10-Year Flood
        65.1,  # 20-Year Flood
        65.2,  # 50-Year Flood
        65.40757,  # High Observed Water
        65.7789,  # Extended High Water
    ]
elif name == 'Sperillen':
    water_levels = [
        147.4310,  # Extended Low Water
        148.0985862129438,  # Extended Low Energy
        148.1312,  # Low Observed Water
        148.3388,  # Low Observed Energy
        148.3988,  # Extended 1st percentile Energy
        148.47,  # 1st percentile Energy
        148.6157,  # Extended 5th percentile Energy
        148.67,  # 5th percentile Energy
        148.7402,  # Extended 10th percentile Energy
        148.76,  # 10th percentile Energy
        148.9442,  # Extended 25th percentile Energy
        148.98,  # 25th percentile Energy
        149.6641,  # Mean Energy
        149.63151556468176,  # Mean Water
        150.25,  # HRV
        151.1276,  # Mean Flood
```

```
        151.6132,  # 5-Year Flood
        152.0137,  # 10-Year Flood
        152.4,  # 20-Year Flood
        152.9034,  # 50-Year Flood
        154.023,  # High Observed Water
        154.7232,  # Extended High Water
    ]
# Sort the water levels in ascending order
water_levels = sorted(water_levels)
# Initialize an empty list to hold the state definitions
states = []
# Iterate over the sorted water levels to create states
for i in range(len(water_levels)-1):
    lower_bound = water_levels[i]
    upper_bound = water_levels[i+1]
    states.append((f"State {i}", lower_bound, upper_bound))
# Add a final state for the upper bound
upper_bound = water_levels[21]*1.05 # 5% increase from extended high, allows integration.
if name == 'Randsfjord':
    chosen_upper_level = upper_bound # to allow integration a sensible upper level is chosen
elif name == 'Tyrifjorden':
    chosen_upper_level = upper_bound # to allow integration a sensible upper level is chosen
elif name == 'Sperillen':
    chosen_upper_level = upper_bound # to allow integration a sensible upper level is chosen
states.append((f"State {len(water_levels)}", water_levels[-1], chosen_upper_level))

# Convert the states list into a DataFrame
states_df = pd.DataFrame(states, columns=['State', 'Lower_Bound', 'Upper_Bound'])
print(states_df.head(30))
# Choose a specific state to highlight
# Replace 'state_number' with the actual number of the state you want to highlight
state_number = 5  # for example, to highlight State 5
# Set the range to +/- 1m on each side of the bounds for the KDE plot
plot_lower_bound = lower_bound - 0.5
plot_upper_bound = upper_bound + 0.5 if upper_bound else lower_bound + 2  # Add 2m to the upper bound if it's the last
state
# Filter the DataFrame for the water levels within the specified plot range
filtered_df = df[(df['Waterlevel'] >= plot_lower_bound) & (df['Waterlevel'] <= plot_upper_bound)]
# Calculate the meter range within each state
states_df['Range_Meters'] = states_df['Upper_Bound'] - states_df['Lower_Bound']
def merge_multiple_states(states_df, merge_pairs):
    new_states_list = []
    skip_indices = []
    # Sort the merge pairs to ensure we process them in order
    merge_pairs.sort(key=lambda x: x[0])
```

```
    for i, row in states_df.iterrows():
        # Check if this index is part of a pair to merge
        merge_pair = next((pair for pair in merge_pairs if i in pair), None)
        if merge_pair:
            # Skip if this index is the second part of a merge pair, as it's already processed
            if i == merge_pair[1] or i in skip_indices:
                continue
            first_state_idx, second_state_idx = merge_pair
            new_lower_bound = states_df.iloc[first_state_idx]['Lower_Bound']
            new_upper_bound = states_df.iloc[second_state_idx]['Upper_Bound']
            new_states_list.append([f"Merged State {first_state_idx}-{second_state_idx}", new_lower_bound,
new_upper_bound])
            # Mark indices to skip in the next iteration
            skip_indices.extend([first_state_idx, second_state_idx])
        else:
            # Add the state as is if it's not part of a merge pair
            new_states_list.append([row['State'], row['Lower_Bound'], row['Upper_Bound']])
    # Create a new DataFrame from the list of new and merged states
    merged_states_df = pd.DataFrame(new_states_list, columns=['State', 'Lower_Bound', 'Upper_Bound'])
    # Optional: Reset state names to reflect their new order
    merged_states_df['State'] = merged_states_df.index.map(lambda x: f"State {x}")
    return merged_states_df
# Define the pairs of state indices to merge
merge_pairs = [(0, 1), (2, 3), (4,5),(6,7),(8,9),(10,11),(12,13)]  # Example: Merge states at indices 0 and 1, and states at
indices 3 and 4
# Create the new DataFrame with merged states
merged_states_df = merge_multiple_states(states_df, merge_pairs)
# Calculate the meter range within each state
merged_states_df['Range_Meters'] = states_df['Upper_Bound'] - states_df['Lower_Bound']
directory = ''
# Display the updated
print()
print('merged states df')
print(merged_states_df)
# Save the Merged States DataFrame
merged_states_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_merged_States.csv',
index=False)
merged_states_df.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{name}/{name}_merged_States.xlsx', index=False)
```

Appendix 18:

Histric_Risk_Factor.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from scipy.stats import gaussian_kde
from scipy.integrate import quad
# Suppress specific sklearn UserWarnings
warnings.simplefilter("ignore", category=UserWarning)
warnings.simplefilter("ignore", category=FutureWarning)
# Load the dataset
name = 'Tyrifjorden'
season = 'Spring'
df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_{season}_waterlevel_df.csv')
df.dropna(inplace=True)
observed_waterlevel = 63
bandwidth = 0.2
df['Waterlevel'] = pd.to_numeric(df['Waterlevel'], errors='coerce')
# Convert 'Date' to datetime if needed
df['Date'] = pd.to_datetime(df['Date'])
# Setting the 'Date' column as the index for easier plotting
df.set_index('Date', inplace=True)
df_states = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_merged_States.csv')
df_states = df_states.drop(['Range_Meters'], axis=1)
current_state = df_states[(df_states['Lower_Bound'] <= observed_waterlevel) & (df_states['Upper_Bound'] >=
observed_waterlevel)]
if not current_state.empty:
    print("Current State based on the observed water level:")
    print(current_state[['State']])
else:
    print("The observed water level does not match any defined state.")

# Assuming 'df' is your DataFrame
data = {
    'State': ['State 8-14'],
    'Lower_Bound': [df_states.loc[8, 'Lower_Bound']],  # Lower bound from State 8
    'Upper_Bound': [df_states.loc[14, 'Upper_Bound']]  # Upper bound from State 14
}
# Creating a new DataFrame with the combined information
new_df = pd.DataFrame(data)
result = pd.concat([df_states, new_df], ignore_index=True)
```

```
# Indices of rows to remove (you need to adjust these based on your DataFrame)
indices_to_remove = list(range(8, 15))  # This would remove rows for State 8 to State 14
# Removing the specified rows
df_final = result.drop(indices_to_remove)
df_final = df_final.reset_index(drop=True)
# Rename 'State 8-14' to 'State 8' at index 8
df_final.at[8, 'State'] = 'State 8'
df_states = df_final
print(df_states)
df_states.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_final_states.csv', index=False)
df_states.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02 Lakes/{name}/{name}_final_states.xlsx',
index=False)
# KDE and Histogram Water Levels with custom bandwidth
plt.figure(figsize=(10, 6))
# Plot histogram with kernel density estimation
sns.histplot(df['Waterlevel'], color='blue', stat='density', linewidth=0, bins=40)
# Calculate KDE with custom bandwidth
# Calculate the KDE for the water levels
water_level_kde = gaussian_kde(df['Waterlevel'], bw_method=bandwidth)
# Plot KDE curve
x_grid = np.linspace(df['Waterlevel'].min(), df['Waterlevel'].max(), 1000)
plt.plot(x_grid, water_level_kde(x_grid), color='red')
plt.title(f'Kernel Density Estimation (KDE) and Histogram Plot for Water Levels during {season} in {name}')
plt.xlabel('Water Level')
plt.ylabel('Density')
plt.legend(['KDE', 'Histogram'])
plt.show()
risk_values_df = pd.DataFrame(columns=['State', 'Energy_Shortage_Risk', 'Flood_Risk'])
# Loop through each state to calculate and plot
for index, state_row in df_states.iterrows():
    plt.figure(figsize=(10, 6))
    # Calculate the KDE for the water levels
    x_grid = np.linspace(df['Waterlevel'].min(), df['Waterlevel'].max(), 1000)
    y_dens = water_level_kde(x_grid)
    plt.plot(x_grid, y_dens, label='Overall KDE', color='blue')
    # Energy Shortage Risk: integrate from min water level to the lower bound
    energy_shortage_risk = quad(water_level_kde, df['Waterlevel'].min(), state_row['Lower_Bound'])[0]
    energy_shortage_risk = max(energy_shortage_risk, 0)  # Ensure non-negative
    # Flood Risk: integrate from the upper bound to max water level
    flood_risk = quad(water_level_kde, state_row['Upper_Bound'], df['Waterlevel'].max())[0]
    flood_risk = max(flood_risk, 0)  # Ensure non-negative
    # Append the risks to the DataFrame
    new_row = pd.DataFrame({
        'State': [state_row['State']],
        'Energy_Shortage_Risk': [energy_shortage_risk],
```

```
        'Flood_Risk': [flood_risk]
    })
    risk_values_df = pd.concat([risk_values_df, new_row], ignore_index=True)
    # Shade outside the state's range
    plt.fill_between(x_grid, y_dens, where=(x_grid < state_row['Lower_Bound']), color='orange', alpha=0.5)
    plt.fill_between(x_grid, y_dens, where=(x_grid > state_row['Upper_Bound']), color='red', alpha=0.5)
    # Highlight the state range
    plt.axvline(x=state_row['Lower_Bound'], color='black', linestyle='--')
    plt.axvline(x=state_row['Upper_Bound'], color='black', linestyle='--')
    # Adding title and labels
    plt.title(f'KDE Plot for Water Levels in {name} -{season}- Highlighting State {index}')
    plt.xlabel('Water Level (m)')
    plt.ylabel('Density')
    plt.legend()
    plt.show()
# Merging the dataframes on the "State" column
historic_factor_df = pd.merge(df_states, risk_values_df, on="State")
historic_factor_df = historic_factor_df.rename(columns={'Lower_Bound': 'Lower Bound', 'Upper_Bound': 'Upper Bound',
'Energy_Shortage_Risk':'Energy Density','Flood_Risk':'Flood Density'})
print(historic_factor_df)
historic_factor_df.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{name}/{name}_{season}_historic_factor.xlsx', index=False)
# Save the DataFrame to a CSV file
historic_factor_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_{season}_historic_factor.csv', index=False)
```

Appendix 19:

Extended_Risk_Factor.py

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from scipy.stats import gaussian_kde
from scipy.integrate import quad
# Suppress specific sklearn UserWarnings
warnings.simplefilter("ignore", category=UserWarning)
warnings.simplefilter("ignore", category=FutureWarning)
# Load the dataset
name = 'Tyrifjorden'
season = 'summer'
df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_{season}_waterlevel_df.csv')
df.dropna(inplace=True)
observed_waterlevel = 63
df['Waterlevel'] = pd.to_numeric(df['Waterlevel'], errors='coerce')
df['Date'] = pd.to_datetime(df['Date'])
df.set_index('Date', inplace=True)
df_states = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_merged_States.csv')
df_states = df_states.drop(['Range_Meters'], axis=1)
current_state = df_states[(df_states['Lower_Bound'] <= observed_waterlevel) & (df_states['Upper_Bound'] >=
observed_waterlevel)]
if not current_state.empty:
    print("Current State based on the observed water level:")
    print(current_state[['State']])
else:
    print("The observed water level does not match any defined state.")

data = {
    'State': ['State 8-14'],
    'Lower_Bound': [df_states.loc[8, 'Lower_Bound']],  # Lower bound from State 8
    'Upper_Bound': [df_states.loc[14, 'Upper_Bound']]  # Upper bound from State 14
}

new_df = pd.DataFrame(data)
result = pd.concat([df_states, new_df], ignore_index=True)
indices_to_remove = list(range(8, 15))  # This would remove rows for State 8 to State 14
df_final = result.drop(indices_to_remove)
df_final = df_final.reset_index(drop=True)
df_final.at[8, 'State'] = 'State 8'
```

```
df_states = df_final
min_value = df['Waterlevel'].min()
max_value = df['Waterlevel'].max()
min_std = df['Waterlevel'].std()
max_std = min_std
synthetic_lower = np.random.uniform(min_value - 3*min_std, min_value, 250)
synthetic_higher = np.random.uniform(max_value, max_value + 3*max_std, 250)
combined_data = np.concatenate([synthetic_lower, df['Waterlevel'].values, synthetic_higher])
# Set the bandwidth here
bandwidth = 0.2
# Calculate extended KDE with custom bandwidth
kde_extended = gaussian_kde(combined_data, bw_method=bandwidth)
x_dens = np.linspace(combined_data.min(), combined_data.max(), 1000)
y_dens = kde_extended(x_dens)
risk_values_extended_df = pd.DataFrame(columns=['State', 'Energy_Shortage_Risk', 'Flood_Risk'])
plt.figure(figsize=(10, 6))
plt.plot(x_dens, y_dens, label='Extended KDE', color='blue')
# Fill the area under the KDE curve
plt.fill_between(x_dens, y_dens, color='lightblue')
plt.title(f'Extended KDE Plot for Water Levels in {name} ')
plt.xlabel('Water Level (m)')
plt.ylabel('Density')
plt.legend()
plt.show()
for index, state_row in df_states.iterrows():
    energy_shortage_risk = quad(kde_extended, combined_data.min(), state_row['Lower_Bound'])[0]
    flood_risk = quad(kde_extended, state_row['Upper_Bound'], combined_data.max())[0]

    energy_shortage_risk = max(energy_shortage_risk, 0)
    flood_risk = max(flood_risk, 0)

    new_row = pd.DataFrame({
        'State': [state_row['State']],
        'Energy_Shortage_Risk': [energy_shortage_risk],
        'Flood_Risk': [flood_risk]
    })
    risk_values_extended_df = pd.concat([risk_values_extended_df, new_row], ignore_index=True)
    plt.figure(figsize=(10, 6))
    plt.plot(x_dens, y_dens, label='Extended KDE', color='blue')
    plt.fill_between(x_dens, y_dens, where=(x_dens < state_row['Lower_Bound']), color='orange', alpha=0.5, label='Energy
Shortage Risk')
    plt.fill_between(x_dens, y_dens, where=(x_dens > state_row['Upper_Bound']), color='red', alpha=0.5, label='Flood Risk')
    plt.axvline(state_row['Lower_Bound'], color='black', linestyle='--')
    plt.axvline(state_row['Upper_Bound'], color='black', linestyle='--')
    plt.title(f'Extended KDE Plot for Water Levels in {name} - Highlighting State {index}')
```

```
    plt.xlabel('Water Level (m)')
    plt.ylabel('Density')
    plt.legend()
    plt.show()
extended_factor_df = pd.merge(df_states, risk_values_extended_df, on="State")
extended_factor_df = extended_factor_df.rename(columns={'Lower_Bound': 'Lower Bound', 'Upper_Bound': 'Upper Bound',
'Energy_Shortage_Risk':'Energy Density','Flood_Risk':'Flood Density'})
print(extended_factor_df)
extended_factor_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_{season}_extended_factor.csv', index=False)
extended_factor_df.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{name}/{name}_{season}_extended_factor.xlsx', index=False)
```

Appendix 20:

Decision_Single.py

```
import pandas as pd
from sklearn.preprocessing import MinMaxScaler
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
from scipy.stats import gaussian_kde
from scipy.integrate import quad
import sys
import math
# Path to the CSV file
name = "Tyrifjorden"
season = "Spring" #Capital First letter
season_water = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_{season}_waterlevel_df.csv')
observed_waterlevel = 63.95
print(f'Observed Waterlevel : {observed_waterlevel}')
print()
if name == 'Randsfjord':
    mean_flood = 134.689
    LRW = 131.3
    HRW = 134.5
elif name == 'Tyrifjorden':
    mean_flood = 64.2
    LRW = 62
    HRW = 63
elif name == 'Sperillen':
    mean_flood = 151.1276
    LRW = 147.95
    HRW = 150.25
if observed_waterlevel > mean_flood:
    print('We are already in a flood state above Mean Flood')
    sys.exit()
if observed_waterlevel < LRW:
    print('We are already in a Water Shortage state below Lowest Regulated Water Level')
    sys.exit()
# Define the function to calculate risk scores with adjusted exponential scalings
def calculate_risk_scores(observed_waterlevel, LRW, mean_flood):
    # Initialize scores
    flood_score = 0
    energy_score = 0
```

```
    if observed_waterlevel < LRW:
        # Maximum water shortage risk when below LRW
        energy_score = 1  # Max water shortage risk
    elif observed_waterlevel > mean_flood:
        # Maximum flood risk when above Mean Flood
        flood_score = 1  # Max flood risk
    else:
        # Between LRW and Mean Flood: separate exponential scaling of risks
        normalized_level = (observed_waterlevel - LRW) / (mean_flood - LRW)
        flood_score = 1 - np.exp(-10 * (normalized_level**5))
        energy_score =  np.exp(-10 * (normalized_level**2))
    return flood_score, energy_score
# Baseline Risk Scores (Baseline ESR/FRS):
baseline_FR, baseline_ESR = calculate_risk_scores(observed_waterlevel, LRW, mean_flood)
baseline_ESR =  1 + baseline_ESR
baseline_FR =  1 + baseline_FR
print('Baseline Flood Risk:') #Flood Risk Score
print(baseline_FR)
print()
print('Baseline Energy Shortage Risk:') # Energy Shortage Risk Score
print(baseline_ESR)
print()
# Historical Seasonal Density Adjustment (H):
Rename_1 = 'Energy Risk'
Rename_2 = 'Flood Risk'
# Historical Density
historic_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_{season}_historic_factor.csv')
historic_df = historic_df.rename(columns={Rename_1: 'Energy Density', Rename_2: 'Flood Density'})
current_state_historic = historic_df[(historic_df['Lower Bound'] <= observed_waterlevel) & (historic_df['Upper Bound'] >=
observed_waterlevel)]
w_historic = 0.7
historic_energy_density = current_state_historic['Energy Density'].values[0]
historic_flood_density = current_state_historic['Flood Density'].values[0]
# Extended Density
extended_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_{season}_extended_factor.csv')
extended_df = extended_df.rename(columns={Rename_1: 'Energy Density', Rename_2: 'Flood Density'})
current_state_extended = extended_df[(extended_df['Lower Bound'] <= observed_waterlevel) & (extended_df['Upper
Bound'] >= observed_waterlevel)]
w_extended = 0.3
extended_energy_density = current_state_extended['Energy Density'].values[0]
extended_flood_density = current_state_extended['Flood Density'].values[0]
# Density Adjustment Factor calculation (H):
seasonal_density_adjustment_energy = 1 + w_historic * historic_energy_density + w_extended * extended_energy_density
```

```
seasonal_density_adjustment_flood = 1 + w_historic * historic_flood_density + w_extended * extended_flood_density
print('Density Adjustment Factors (H):')
print(f'Energy Density Adjustment : {seasonal_density_adjustment_energy}')
print(f'Flood Density Adjustment : {seasonal_density_adjustment_flood}')
# Current Reservoir Capacity (C)
Remove_1 = 'Waterflow'
print("\nCapacity Factor (C):")
# Load capacity data
capacity_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{name}_Total_Daily.csv')
capacity_df = capacity_df.drop([Remove_1], axis=1)
# Filter the dataset between water levels 62 meters and 64.2 meters
filtered_df = capacity_df[(capacity_df['Waterlevel'] >= LRW) & (capacity_df['Waterlevel'] <= mean_flood)]
# Calculate Normalized Reservoir Level
# Find the maximum value
max_reservoir_value = filtered_df['Reservoir'].max()
# Print the entire row(s) where the reservoir is at its maximum capacity
max_reservoir_rows = filtered_df[filtered_df['Reservoir'] == max_reservoir_value]
# Print the entire row(s) where the reservoir is at its minimum capacity
min_reservoir_value = filtered_df['Reservoir'].min()
min_reservoir_rows = filtered_df[filtered_df['Reservoir'] == min_reservoir_value]
# Find the row corresponding most closely to the observed water level
closest_row = filtered_df.iloc[(filtered_df['Waterlevel'] - observed_waterlevel).abs().argsort()[:1]]
# Calculate normalized reservoir level for the closest row
normalized_reservoir = closest_row['Reservoir'].values[0] / max_reservoir_value
# Define thresholds and scaling factors
L = 0.2  # Low threshold approx 5% percentile
H = 0.7  # High threshold approx 95% percentile
alpha = 2  # Scaling factor for low reservoir levels
beta = 3  # Scaling factor for high reservoir levels
# Calculate the capacity adjustment factors for flood and energy risks
def calculate_capacity_factors(normalized_reservoir, L, H, alpha, beta):
    if normalized_reservoir <= L:
        energy_capacity_factor = 1 + alpha * (L - normalized_reservoir)
    else:
        energy_capacity_factor = 1
    if normalized_reservoir >= H:
        flood_capacity_factor = 1 + beta * (normalized_reservoir - H)
    else:
        flood_capacity_factor = 1
    return flood_capacity_factor, energy_capacity_factor
flood_capacity_factor, energy_capacity_factor = calculate_capacity_factors(normalized_reservoir, L, H, alpha, beta)
# Print results
print(f"Flood Capacity Factor: {flood_capacity_factor}")
print(f"Energy Capacity Factor: {energy_capacity_factor}")
# Regulatory Constraints (R):
```

```
print()
print('Regulatory Constraints Factor')
# Define a function to calculate the penalty factors for flood and energy risks with pre-threshold adjustments
def calculate_penalty_factors(observed_level, HRW, LRW, pre_threshold=0.2):
    flood_penalty_factor = 1
    energy_penalty_factor = 1
    regulation_zone = HRW - LRW
    lower_threshold = LRW + pre_threshold * regulation_zone
    upper_threshold = HRW - pre_threshold * regulation_zone
    # Energy penalty factor increases as the water level gets closer to LRW
    if observed_level < lower_threshold:
        energy_penalty_factor += (lower_threshold - observed_level) / (lower_threshold - LRW)
    # Flood penalty factor increases as the water level exceeds upper threshold
    if observed_level > HRW:
        flood_penalty_factor = 1  # Regulatory measures mitigate flood risk above HRW
    elif observed_level > upper_threshold:
        flood_penalty_factor += (observed_level - upper_threshold) / (HRW - upper_threshold)
    return flood_penalty_factor, energy_penalty_factor
# Calculate the penalty factors
flood_penalty_factor, energy_penalty_factor = calculate_penalty_factors(observed_waterlevel, HRW, LRW)
# Print results
print(f"Flood Penalty Factor for {name}: {flood_penalty_factor}")
print(f"Energy Penalty Factor for {name}: {energy_penalty_factor}")
# Seasonal Trends  (S):
# Load seasonal trend data
trend_analysis_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{name}_seasonal_trend.csv')
seasonal_trend = trend_analysis_df.loc[trend_analysis_df['Season'] == season, 'Slope'].values[0]
# Calculate Seasonal Deviation
mean_water_season = season_water['Waterlevel'].mean()
std_water_season = season_water['Waterlevel'].std()
seasonal_deviation = (observed_waterlevel - mean_water_season) / std_water_season
# Calculate Seasonal Volatility
# Convert Date column to datetime
season_water['Date'] = pd.to_datetime(season_water['Date'])
# Extract year from the Date column
season_water['Year'] = season_water['Date'].dt.year
# Calculate the standard deviation for each year
yearly_volatility = season_water.groupby('Year')['Waterlevel'].std()
# Calculate the average volatility across all years
seasonal_volatility = yearly_volatility.mean()
# Seasonal Scaling Factor
seasonal_k = 0.8
# Calculate the Seasonal Adjustment Factors
def calculate_seasonal_adjustment_factor(seasonal_deviation, seasonal_trend, seasonal_volatility, seasonal_k, type='flood'):
```

```
    if type == 'flood':
        adjustment_factor = 1 + seasonal_k * (seasonal_deviation + seasonal_trend + seasonal_volatility)
        adjustment_factor = max(adjustment_factor, 1.1)  # Set a minimum value to avoid negative risks
    elif type == 'energy':
        # Ensure the adjustment factor remains positive by using an absolute value
        adjustment_factor = 1 + seasonal_k * (seasonal_deviation - seasonal_trend + seasonal_volatility)
        adjustment_factor = max(adjustment_factor, 1.1)  # Set a minimum value to avoid negative risks
    return adjustment_factor
seasonal_adjustment_factor_flood = calculate_seasonal_adjustment_factor(seasonal_deviation, seasonal_trend,
seasonal_volatility, seasonal_k, type='flood')
seasonal_adjustment_factor_energy = calculate_seasonal_adjustment_factor(seasonal_deviation, seasonal_trend,
seasonal_volatility, seasonal_k, type='energy')
# Print results
print()
print('Seasonal Adjustment Factor for Flood (S_flood):')
print(seasonal_adjustment_factor_flood)
print('Seasonal Adjustment Factor for Energy (S_energy):')
print(seasonal_adjustment_factor_energy)
# Final Risk before decision factor
print()
print('Final Scores')
final_ESR = baseline_ESR * seasonal_density_adjustment_energy * energy_capacity_factor  * energy_penalty_factor *
seasonal_adjustment_factor_energy
print(f'Final ESR, before decision factor : {final_ESR}')

final_FR = baseline_FR * seasonal_density_adjustment_flood * flood_capacity_factor * flood_penalty_factor *
seasonal_adjustment_factor_flood
print(f'Final FR, before decision factor : {final_FR}')
# Decision Risk Score (D):
print()
print('Decision Risk Score')
# Energy Shortage Factors
ESR_increase = 1.2  # Increase outflow gives a 20% increase in Energy Shortage Risk
ESR_decrease = 0.8  # Decrease outflow gives 20% decrease in Energy Shortage Risk
ESR_maintain = 1    # Maintain outflow gives no change in final risk
# Flood Factors
FR_increase = 0.8  # Increase outflow gives a 20% decrease in Flood Risk
FR_decrease = 1.2  # Decrease outflow gives a 20% increase in Flood Risk
FR_maintain = 1    # Maintain outflow gives no change in final risk
# Calculate final risks
def calculate_decision_risks(final_ESR, final_FR):
    # Increase
    ESR_increase_risk = ESR_increase * final_ESR
    FR_increase_risk = FR_increase * final_FR
```

```
    # Decrease
    ESR_decrease_risk = ESR_decrease * final_ESR
    FR_decrease_risk = FR_decrease * final_FR

    # Maintain
    ESR_maintain_risk = ESR_maintain * final_ESR
    FR_maintain_risk = FR_maintain * final_FR
    return {
        'Increase': {'ESR': ESR_increase_risk, 'FR': FR_increase_risk},
        'Decrease': {'ESR': ESR_decrease_risk, 'FR': FR_decrease_risk},
        'Maintain': {'ESR': ESR_maintain_risk, 'FR': FR_maintain_risk}
    }
# Get the decision risks
decision_risks = calculate_decision_risks(final_ESR, final_FR)
if observed_waterlevel < HRW:
    print(f'Observed Waterlevel : {observed_waterlevel}m in {season}')
    # Print the results with 4 decimal places
    print('Increase:')
    print(f'Energy Shortage Rrisk : {decision_risks["Increase"]["ESR"]:.4f}')
    print(f'Flood Risk : {decision_risks["Increase"]["FR"]:.4f}')
    print('Decrease:')
    print(f'Energy Shortage Risk : {decision_risks["Decrease"]["ESR"]:.4f}')
    print(f'Flood Risk : {decision_risks["Decrease"]["FR"]:.4f}')
    print('Maintain:')
    print(f'Energy Shortage Risk : {decision_risks["Maintain"]["ESR"]:.4f}')
    print(f'Flood Risk : {decision_risks["Maintain"]["FR"]:.4f}')
else:
    print(f'Observed Waterlevel : {observed_waterlevel}m in {season}')
    # Print the results with 4 decimal places
    print('Decrease:')
    print(f'Energy Shortage Risk : {decision_risks["Decrease"]["ESR"]:.4f}')
    print(f'Flood Risk : {decision_risks["Decrease"]["FR"]:.4f}')
    print('Maintain:')
    print(f'Energy Shortage Risk : {decision_risks["Maintain"]["ESR"]:.4f}')
    print(f'Flood Risk : {decision_risks["Maintain"]["FR"]:.4f}')



# Calculate KDE for the water levels
water_level_kde = gaussian_kde(season_water['Waterlevel'])
x_dens = np.linspace(season_water['Waterlevel'].min(), season_water['Waterlevel'].max(), 100)
y_dens = water_level_kde(x_dens)

# Extract lower and upper bounds from current_state_historic
lower_bound = current_state_historic['Lower Bound'].values[0]
upper_bound = current_state_historic['Upper Bound'].values[0]
```

```
# Create arrays for lower and upper bounds with the same length as x_dens
lower_bound_array = np.full_like(x_dens, lower_bound)
upper_bound_array = np.full_like(x_dens, upper_bound)

# KDE plot for the entire range
plt.figure(figsize=(10, 6))
plt.plot(x_dens, y_dens, label='KDE')

# Shade the outside of the selected state
plt.fill_between(x_dens, y_dens, where=(x_dens <= lower_bound_array), alpha=0.5, color='orange', label='Energy Shortage Risk')
plt.fill_between(x_dens, y_dens, where=(x_dens >= upper_bound_array), alpha=0.5, color='red', label='Flood Risk')

# Highlight the selected state (leave it unshaded)
plt.axvline(lower_bound, color='black', linestyle='--')
plt.axvline(upper_bound, color='black', linestyle='--')
plt.axvline(observed_waterlevel, color='blue', label='Observed Waterlevel')
plt.title(f'KDE Plot for Water Levels in {name} in {season}')
plt.xlabel('Water Level (m)')
plt.ylabel('Density')
plt.legend()
plt.show()

if observed_waterlevel < HRW:

    # Decision points and their corresponding risk scores
    decision_points = ['Increase', 'Decrease', 'Maintain']
    ESR_scores = [decision_risks[point]['ESR'] for point in decision_points]
    FR_scores = [decision_risks[point]['FR'] for point in decision_points]

    # Plot the decision points on the KDE plot
    plt.scatter(ESR_scores, FR_scores, color='black', label='Decision Points')

    # Annotate the decision points with text
    for point, ESR, FR in zip(decision_points, ESR_scores, FR_scores):
        plt.text(ESR, FR, point, ha='left')

    plt.xlabel('Energy Shortage Risk')
    plt.ylabel('Flood Risk')

    plt.show()

else:
    # Decision points and their corresponding risk scores
```

```
decision_points = ['Decrease', 'Maintain']
ESR_scores = [decision_risks[point]['ESR'] for point in decision_points]
FR_scores = [decision_risks[point]['FR'] for point in decision_points]

# Plot the decision points on the KDE plot
plt.scatter(ESR_scores, FR_scores, color='black', label='Decision Points')

# Annotate the decision points with text
for point, ESR, FR in zip(decision_points, ESR_scores, FR_scores):
    plt.text(ESR, FR, point, ha='left')

plt.xlabel('Energy Shortage Risk')
plt.ylabel('Flood Risk')

plt.show()
```

Appendix 21:

Decision_for_loop.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.pyplot as plt
import warnings
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import MinMaxScaler
# Suppress specific sklearn UserWarnings
warnings.simplefilter("ignore", category=UserWarning)
warnings.simplefilter("ignore", category=FutureWarning)
# Example lake name and current conditions
lake_name = "Sperillen"
season = "Summer"
current_waterlevel = 151
# Load seasonal trend data
trend_analysis_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{lake_name}_seasonal_trend.csv')
seasonal_trend = trend_analysis_df.loc[trend_analysis_df['Season'] == season, 'Slope'].values[0]
# Load seasonal water level data
season_water = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{lake_name}_{season}_waterlevel_df.csv')
# Load density data
historic_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{lake_name}_{season}_historic_factor.csv')
extended_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/{lake_name}_{season}_extended_factor.csv')
historic_df = historic_df.rename(columns={'Energy Risk': 'Energy Density', 'Flood Risk': 'Flood Density'})
extended_df = extended_df.rename(columns={'Energy Risk': 'Energy Density', 'Flood Risk': 'Flood Density'})

# Load capacity data
capacity_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned Data/{lake_name}_Total_Daily.csv')
capacity_df = capacity_df.drop(['Waterflow'], axis=1)

# Define thresholds and scaling factors
L = 0.2  # Low threshold approx 5% percentile
H = 0.7  # High threshold approx 95% percentile
alpha = 2  # Scaling factor for low reservoir levels
beta = 3  # Scaling factor for high reservoir levels
```

```
# Define reservoir regulation parameters
if lake_name == 'Randsfjord':
    mean_flood = 134.689
    LRW = 131.3
    HRW = 134.5
elif lake_name == 'Tyrifjorden':
    mean_flood = 64.2
    LRW = 62
    HRW = 63
elif lake_name == 'Sperillen':
    mean_flood = 151.1276
    LRW = 147.95
    HRW = 150.25

# Define the function to calculate risk scores with adjusted exponential scalings
def calculate_risk_scores(observed_waterlevel, LRW, mean_flood):
    # Initialize scores
    flood_score = 0
    energy_score = 0

    if observed_waterlevel < LRW:
        # Maximum water shortage risk when below LRW
        energy_score = 1  # Max water shortage risk
    elif observed_waterlevel > mean_flood:
        # Maximum flood risk when above Mean Flood
        flood_score = 1  # Max flood risk
    else:
        # Between LRW and Mean Flood: separate exponential scaling of risks
        normalized_level = (observed_waterlevel - LRW) / (mean_flood - LRW)



        flood_score = 1 - np.exp(-10 * (normalized_level**5))
        energy_score =  np.exp(-10 * (normalized_level**2))
    return flood_score, energy_score

# Define the parameters
water_levels = np.linspace(LRW, mean_flood, 3000)

# Calculate the scores for each water level
flood_scores = []
energy_scores = []

for level in water_levels:
    flood_score, energy_score = calculate_risk_scores(level, LRW, mean_flood)
    flood_scores.append(flood_score)
```

```
    energy_scores.append(energy_score)

# Plotting the results
plt.figure(figsize=(10, 6))
plt.plot(water_levels, flood_scores, label='Flood Risk Score', color='blue')
plt.plot(water_levels, energy_scores, label='Energy Shortage Risk Score', color='red')
plt.axvline(LRW, color='green', linestyle='--', label='LRW')
plt.axvline(mean_flood, color='purple', linestyle='--', label='Mean Flood')
plt.xlabel('Observed Water Level')
plt.ylabel('Risk Score')
plt.title('Exponential Risk Scores for Flood and Energy Shortage')
plt.legend()
plt.grid(True)
plt.show()

# Function to calculate capacity factors
def calculate_capacity_factors(normalized_reservoir, L, H, alpha, beta):
    if normalized_reservoir <= L:
        energy_capacity_factor = 1 + alpha * (L - normalized_reservoir)
    else:
        energy_capacity_factor = 1

    if normalized_reservoir >= H:
        flood_capacity_factor = 1 + beta * (normalized_reservoir - H)
    else:
        flood_capacity_factor = 1

    return flood_capacity_factor, energy_capacity_factor

# Function to calculate penalty factors
def calculate_penalty_factors(observed_level, HRW, LRW, pre_threshold=0.2):
    flood_penalty_factor = 1
    energy_penalty_factor = 1
    regulation_zone = HRW - LRW
    lower_threshold = LRW + pre_threshold * regulation_zone
    upper_threshold = HRW - pre_threshold * regulation_zone

    if observed_level < lower_threshold:
        energy_penalty_factor += (lower_threshold - observed_level) / (lower_threshold - LRW)

    if observed_level > HRW:
        flood_penalty_factor = 1
    elif observed_level > upper_threshold:
        flood_penalty_factor += (observed_level - upper_threshold) / (HRW - upper_threshold)
```

```
    return flood_penalty_factor, energy_penalty_factor

# Function to calculate seasonal adjustment factors
def calculate_seasonal_adjustment_factor(seasonal_deviation, seasonal_trend, seasonal_volatility, seasonal_k, type='flood'):
    if type == 'flood':
        adjustment_factor = 1 + seasonal_k * (seasonal_deviation + seasonal_trend + seasonal_volatility)
        adjustment_factor = max(adjustment_factor, 1)  # Set a minimum value to avoid negative risks
    elif type == 'energy':
        adjustment_factor = 1 + seasonal_k * (-1*seasonal_deviation - seasonal_trend + seasonal_volatility)
        adjustment_factor = max(adjustment_factor, 1)
    return adjustment_factor

# Define decision factors
ESR_increase = 1.2  # Increase outflow gives a 20% increase in Energy Shortage Risk
ESR_decrease = 0.8  # Decrease outflow gives 20% decrease in Energy Shortage Risk
ESR_maintain = 1  # Maintain outflow gives no change in final risk
FR_increase = 0.8  # Increase outflow gives a 20% decrease in Flood Risk
FR_decrease = 1.2  # Decrease outflow gives a 20% increase in Flood Risk
FR_maintain = 1  # Maintain outflow gives no change in final risk
# Iterate through observed water levels from 62 to 64.2 in increments of 0.05
results = []
combined = []
decision_scores = []
for observed_waterlevel in np.arange(LRW, mean_flood, 0.05):
    baseline_FR, baseline_ESR = calculate_risk_scores(observed_waterlevel, LRW, mean_flood)
    baseline_FR = 1 + baseline_FR
    baseline_ESR = 1 + baseline_ESR
    # Historical Density
    current_state_historic = historic_df[(historic_df['Lower Bound'] <= observed_waterlevel) & (historic_df['Upper Bound']
>= observed_waterlevel)]
    historic_energy_density = current_state_historic['Energy Density'].values[0]
    historic_flood_density = current_state_historic['Flood Density'].values[0]
    # Extended Density
    current_state_extended = extended_df[(extended_df['Lower Bound'] <= observed_waterlevel) & (extended_df['Upper
Bound'] >= observed_waterlevel)]
    extended_energy_density = current_state_extended['Energy Density'].values[0]
    extended_flood_density = current_state_extended['Flood Density'].values[0]
    # Density Adjustment Factor calculation (H)
    w_historic = 0.7
    w_extended = 0.3
    seasonal_density_adjustment_energy = 1 + w_historic * historic_energy_density + w_extended *
extended_energy_density
    seasonal_density_adjustment_flood = 1 + w_historic * historic_flood_density + w_extended * extended_flood_density
    # Capacity Factor (C)
    filtered_df = capacity_df[(capacity_df['Waterlevel'] >= LRW) & (capacity_df['Waterlevel'] <= mean_flood)]
```

```
    max_reservoir_value = filtered_df['Reservoir'].max()
    closest_row = filtered_df.iloc[(filtered_df['Waterlevel'] - observed_waterlevel).abs().argsort()[:1]]
    normalized_reservoir = closest_row['Reservoir'].values[0] / max_reservoir_value
    flood_capacity_factor, energy_capacity_factor = calculate_capacity_factors(normalized_reservoir, L, H, alpha, beta)
    # Regulatory Constraints (R)
    flood_penalty_factor, energy_penalty_factor = calculate_penalty_factors(observed_waterlevel, HRW, LRW)
    # Seasonal Trends (S)
    mean_water_season = season_water['Waterlevel'].mean()
    std_water_season = season_water['Waterlevel'].std()
    seasonal_deviation = (observed_waterlevel - mean_water_season) / std_water_season
    # Convert Date column to datetime
    season_water['Date'] = pd.to_datetime(season_water['Date'])
    # Extract year from the Date column
    season_water['Year'] = season_water['Date'].dt.year
    yearly_volatility = season_water.groupby('Year')['Waterlevel'].std()
    seasonal_volatility = yearly_volatility.mean()
    seasonal_k = 0.8
    seasonal_adjustment_factor_flood = calculate_seasonal_adjustment_factor(seasonal_deviation, seasonal_trend,
seasonal_volatility, seasonal_k, type='flood')
    seasonal_adjustment_factor_energy = calculate_seasonal_adjustment_factor(seasonal_deviation, seasonal_trend,
seasonal_volatility, seasonal_k, type='energy')
    # Final Risk
    final_ESR = baseline_ESR * seasonal_density_adjustment_energy * energy_capacity_factor * energy_penalty_factor *
seasonal_adjustment_factor_energy
    final_FR = baseline_FR * seasonal_density_adjustment_flood * flood_capacity_factor * flood_penalty_factor *
seasonal_adjustment_factor_flood
        # Decision Risk Score (D)
    ESR_increase_risk = ESR_increase * final_ESR
    FR_increase_risk = FR_increase * final_FR

    ESR_decrease_risk = ESR_decrease * final_ESR
    FR_decrease_risk = FR_decrease * final_FR

    ESR_maintain_risk = ESR_maintain * final_ESR
    FR_maintain_risk = FR_maintain * final_FR

    # Append the decision scores
    decision_scores.append({
        'Observed Water Level': observed_waterlevel,
        'ESR Increase Risk': ESR_increase_risk,
        'FR Increase Risk': FR_increase_risk,
        'ESR Decrease Risk': ESR_decrease_risk,
        'FR Decrease Risk': FR_decrease_risk,
        'ESR Maintain Risk': ESR_maintain_risk,
        'FR Maintain Risk': FR_maintain_risk
```

```
    })
    # Store results
    results.append({
        'Observed Water Level': observed_waterlevel,
        'Final Energy Shortage Risk (ESR)': final_ESR,
        'Final Flood Risk (FR)': final_FR
    })

    # Append the results to the results list
    combined.append({
        'Observed Water Level': observed_waterlevel,
        'Baseline Flood Risk': baseline_FR,
        'Baseline Energy Shortage Risk': baseline_ESR,
        'Energy Density Adjustment (H)': seasonal_density_adjustment_energy,
        'Flood Density Adjustment (H)': seasonal_density_adjustment_flood,
        'Flood Capacity Factor (C)': flood_capacity_factor,
        'Energy Capacity Factor (C)': energy_capacity_factor,
        'Flood Penalty Factor (R)': flood_penalty_factor,
        'Energy Penalty Factor (R)': energy_penalty_factor,
        'Seasonal Adjustment Factor for Flood (S_flood)': seasonal_adjustment_factor_flood,
        'Seasonal Adjustment Factor for Energy (S_energy)': seasonal_adjustment_factor_energy,
        'Final Energy Shortage Risk (ESR)': final_ESR,
        'Final Flood Risk (FR)': final_FR,
    })
combined_df = pd.DataFrame(combined)
# Save the slopes of the trend analysis as CSV
combined_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/combined_df_{lake_name}_{season}.csv', index=False)
combined_df.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{lake_name}/combined_df_{lake_name}_{season}.xlsx', index=False)
# Convert results to DataFrame for further analysis or plotting
results_df = pd.DataFrame(results)
print(results_df)
# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(results_df['Observed Water Level'], results_df['Final Energy Shortage Risk (ESR)'], label='Energy Shortage Risk
(ESR)', color='b', marker='o')
plt.plot(results_df['Observed Water Level'], results_df['Final Flood Risk (FR)'], label='Flood Risk (FR)', color='r', marker='x')
# Adding vertical lines for LRW, HRW, and mean_flood
plt.axvline(x=LRW, color='yellow', linestyle='--', label='LRW')
plt.axvline(x=HRW, color='black', linestyle='--', label='HRW')
plt.axvline(x=mean_flood, color='purple', linestyle='--', label='Mean Flood Level')
plt.xlabel('Observed Water Level (m)')
plt.ylabel('Final Risk')
plt.title(f'Final Risk Scores vs. Observed Water Level for {season}')
```

```
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
decision_scores_df = pd.DataFrame(decision_scores)
print(decision_scores_df)
# Convert decision scores to DataFrame
decision_scores_df = pd.DataFrame(decision_scores)
decision_scores_df.to_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/decision_risk_df_{lake_name}_{season}.csv', index=False)
x_cutoff = HRW  # Replace this with the desired x-axis value
# Create masks for the points up to the cutoff
esr_mask = np.where(decision_scores_df['Observed Water Level'] <= x_cutoff, decision_scores_df['ESR Increase Risk'],
np.nan)
fr_mask = np.where(decision_scores_df['Observed Water Level'] <= x_cutoff, decision_scores_df['FR Increase Risk'],
np.nan)
# Plotting decision scores
plt.figure(figsize=(12, 6))
plt.plot(decision_scores_df['Observed Water Level'], esr_mask, label='ESR Increase Risk', color='b', marker='o')
plt.plot(decision_scores_df['Observed Water Level'], fr_mask, label='FR Increase Risk', color='r', marker='x')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Decrease Risk'], label='ESR Decrease Risk',
color='g', marker='s')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Decrease Risk'], label='FR Decrease Risk',
color='c', marker='^')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Maintain Risk'], label='ESR Maintain Risk',
color='m', marker='d')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Maintain Risk'], label='FR Maintain Risk',
color='y', marker='v')
# Adding vertical lines for LRW, HRW, and mean_flood
plt.axvline(x=LRW, color='yellow', linestyle='--', label='LRW')
plt.axvline(x=HRW, color='black', linestyle='--', label='HRW')
plt.axvline(x=mean_flood, color='purple', linestyle='--', label='Mean Flood Level')
plt.xlabel('Observed Water Level (m)')
plt.ylabel('Risk')
plt.title(f'Decision Risk Scores vs. Observed Water Level for {season}')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
# Plotting decision scores with histogram in the background
fig, ax1 = plt.subplots(figsize=(12, 6))
# Plot the histogram of the seasonal water level data
ax1.hist(season_water['Waterlevel'], bins=30, color='gray', alpha=0.6, edgecolor='black')
ax1.set_xlabel('Observed Water Level (m)')
ax1.set_ylabel('Frequency')
ax1.set_title(f'Decision Risk Scores vs. Observed Water Level for {season}')
# Create a secondary y-axis for the risk scores
```

```
ax2 = ax1.twinx()
# Plot the decision risk scores
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Increase Risk'], label='ESR Increase Risk',
color='b', marker='o')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Increase Risk'], label='FR Increase Risk',
color='r', marker='x')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Decrease Risk'], label='ESR Decrease Risk',
color='g', marker='s')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Decrease Risk'], label='FR Decrease Risk',
color='c', marker='^')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Maintain Risk'], label='ESR Maintain Risk',
color='m', marker='d')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Maintain Risk'], label='FR Maintain Risk',
color='y', marker='v')
# Adding vertical lines for LRW, HRW, and mean_flood
ax2.axvline(x=LRW, color='yellow', linestyle='--', label='LRW')
ax2.axvline(x=HRW, color='black', linestyle='--', label='HRW')
ax2.axvline(x=mean_flood, color='purple', linestyle='--', label='Mean Flood Level')
ax2.set_ylabel('Risk')
# Combine legends from both axes
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
# Function to normalize a single column in a DataFrame
def normalize_column(df, column_name):
    scaler = MinMaxScaler()
    df[column_name] = scaler.fit_transform(df[[column_name]])
    return df
# Function to normalize selected columns in a DataFrame together
def normalize_columns_together(df, columns):
    min_val = df[columns].min().min()
    max_val = df[columns].max().max()
    df[columns] = (df[columns] - min_val) / (max_val - min_val)
    return df
# List of columns to normalize in each DataFrame
results_columns_to_normalize = ['Final Energy Shortage Risk (ESR)', 'Final Flood Risk (FR)']
# Normalize each column in results_df individually
for col in results_columns_to_normalize:
    results_df = normalize_column(results_df, col)
# List of columns to normalize together in decision_scores_df
esr_columns_to_normalize = ['ESR Increase Risk', 'ESR Decrease Risk', 'ESR Maintain Risk']
fr_columns_to_normalize = ['FR Increase Risk', 'FR Decrease Risk', 'FR Maintain Risk']
# Normalize each set of columns in decision_scores_df together
```

```
decision_scores_df = normalize_columns_together(decision_scores_df, esr_columns_to_normalize)
decision_scores_df = normalize_columns_together(decision_scores_df, fr_columns_to_normalize)
# Verify normalization
print("Normalized decision_scores_df:")
for col_set in [esr_columns_to_normalize, fr_columns_to_normalize]:
    for col in col_set:
        print(f"{col}: min = {decision_scores_df[col].min()}, max = {decision_scores_df[col].max()}")
# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(results_df['Observed Water Level'], results_df['Final Energy Shortage Risk (ESR)'], label='Energy Shortage Risk (ESR)', color='b', marker='o')
plt.plot(results_df['Observed Water Level'], results_df['Final Flood Risk (FR)'], label='Flood Risk (FR)', color='r', marker='x')
# Adding vertical lines for LRW, HRW, and mean_flood
plt.axvline(x=LRW, color='yellow', linestyle='--', label='LRW')
plt.axvline(x=HRW, color='black', linestyle='--', label='HRW')
plt.axvline(x=mean_flood, color='purple', linestyle='--', label='Mean Flood Level')
plt.xlabel('Observed Water Level (m)')
plt.ylabel('Final Risk')
plt.title(f'Final Risk Scores vs. Observed Water Level for {season}')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
# Mask the data points for ESR and FR increase risks beyond HRW
x_cutoff = HRW  # Replace this with the desired x-axis value
esr_mask = np.where(decision_scores_df['Observed Water Level'] <= x_cutoff, decision_scores_df['ESR Increase Risk'], np.nan)
fr_mask = np.where(decision_scores_df['Observed Water Level'] <= x_cutoff, decision_scores_df['FR Increase Risk'], np.nan)
# Plotting decision scores
plt.figure(figsize=(12, 6))
plt.plot(decision_scores_df['Observed Water Level'], esr_mask, label='ESR Increase Risk', color='b', marker='o')
plt.plot(decision_scores_df['Observed Water Level'], fr_mask, label='FR Increase Risk', color='r', marker='x')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Decrease Risk'], label='ESR Decrease Risk', color='g', marker='s')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Decrease Risk'], label='FR Decrease Risk', color='c', marker='^')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Maintain Risk'], label='ESR Maintain Risk', color='m', marker='d')
plt.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Maintain Risk'], label='FR Maintain Risk', color='y', marker='v')
# Adding vertical lines for LRW, HRW, and mean_flood
plt.axvline(x=LRW, color='yellow', linestyle='--', label='LRW')
plt.axvline(x=HRW, color='black', linestyle='--', label='HRW')
plt.axvline(x=mean_flood, color='purple', linestyle='--', label='Mean Flood Level')
plt.xlabel('Observed Water Level (m)')
```

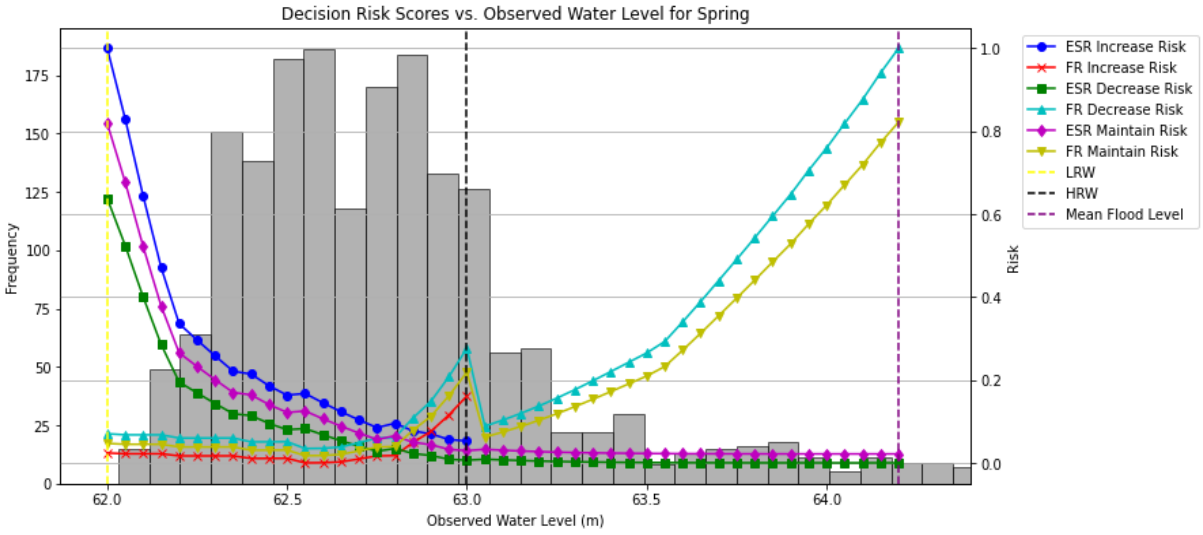
```
plt.ylabel('Risk')
plt.title(f'Decision Risk Scores vs. Observed Water Level for {season}')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
# Plotting decision scores with histogram in the background
fig, ax1 = plt.subplots(figsize=(12, 6))
# Plot the histogram of the seasonal water level data
ax1.hist(season_water['Waterlevel'], bins=30, color='gray', alpha=0.6, edgecolor='black')
ax1.set_xlabel('Observed Water Level (m)')
ax1.set_ylabel('Frequency')
ax1.set_title(f'Decision Risk Scores vs. Observed Water Level for {season}')
# Create a secondary y-axis for the risk scores
ax2 = ax1.twinx()
# Plot the decision risk scores
ax2.plot(decision_scores_df['Observed Water Level'], esr_mask, label='ESR Increase Risk', color='b', marker='o')
ax2.plot(decision_scores_df['Observed Water Level'], fr_mask, label='FR Increase Risk', color='r', marker='x')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Decrease Risk'], label='ESR Decrease Risk',
color='g', marker='s')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Decrease Risk'], label='FR Decrease Risk',
color='c', marker='^')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['ESR Maintain Risk'], label='ESR Maintain Risk',
color='m', marker='d')
ax2.plot(decision_scores_df['Observed Water Level'], decision_scores_df['FR Maintain Risk'], label='FR Maintain Risk',
color='y', marker='v')
# Adding vertical lines for LRW, HRW, and mean_flood
ax2.axvline(x=LRW, color='yellow', linestyle='--', label='LRW')
ax2.axvline(x=HRW, color='black', linestyle='--', label='HRW')
ax2.axvline(x=mean_flood, color='purple', linestyle='--', label='Mean Flood Level')
ax2.set_ylabel('Risk')
ax2.set_xlim(left=ax1.get_xlim()[0], right=mean_flood + 0.2)
# Combine legends from both axes
lines, labels = ax1.get_legend_handles_labels()
lines2, labels2 = ax2.get_legend_handles_labels()
ax2.legend(lines + lines2, labels + labels2, bbox_to_anchor=(1.05, 1), loc='upper left')
plt.grid(True)
plt.show()
print()
print()
norm_decision_scores = decision_scores_df
norm_results = results_df
norm_decision_scores.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{lake_name}/norm_risk_dec_values_{lake_name}_{season}.xlsx', index=False)
#results_df.to_excel(f'/Users/simen/Desktop/Complete Master/03 Excel Products/02
Lakes/{lake_name}/norm_results_values_{lake_name}_{season}.xlsx', index=False)
```

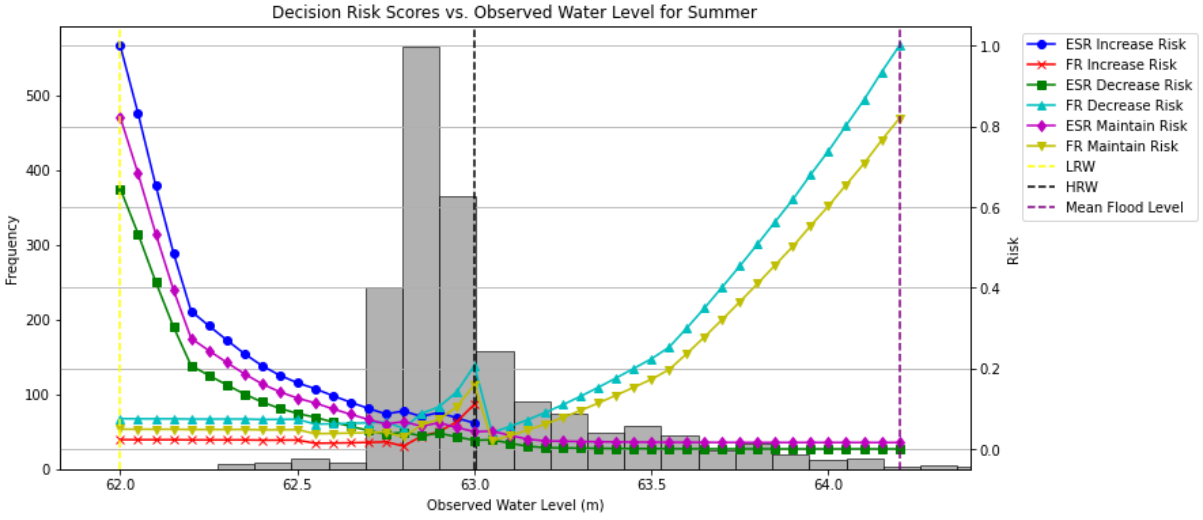
```
# Find the row with the closest observed water level to the current water level
closest_row = decision_scores_df.iloc[(decision_scores_df['Observed Water Level'] - current_waterlevel).abs().argmin()]
# Print the decision scores for the closest observed water level
print(f"Closest decision scores for observed water level {closest_row['Observed Water Level']}:")
print(closest_row)
```

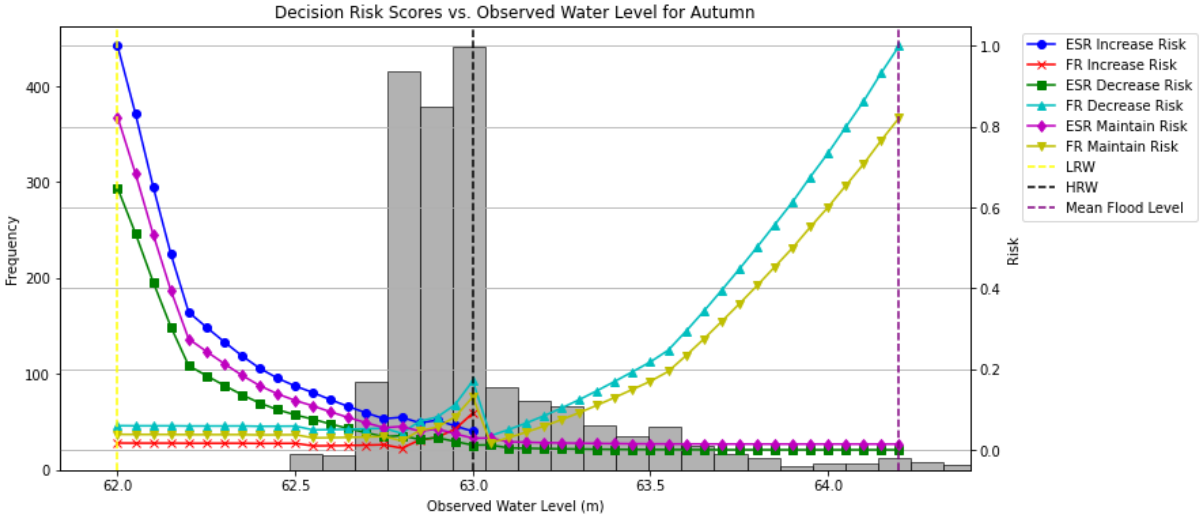
Appendix 22:

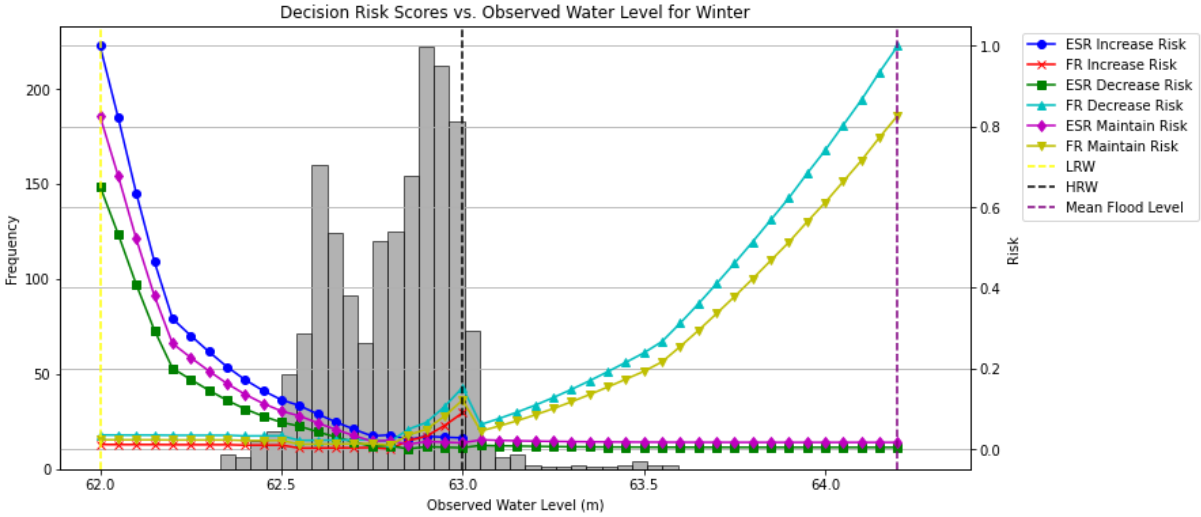
Complete Risk Values

Tyrifjorden:

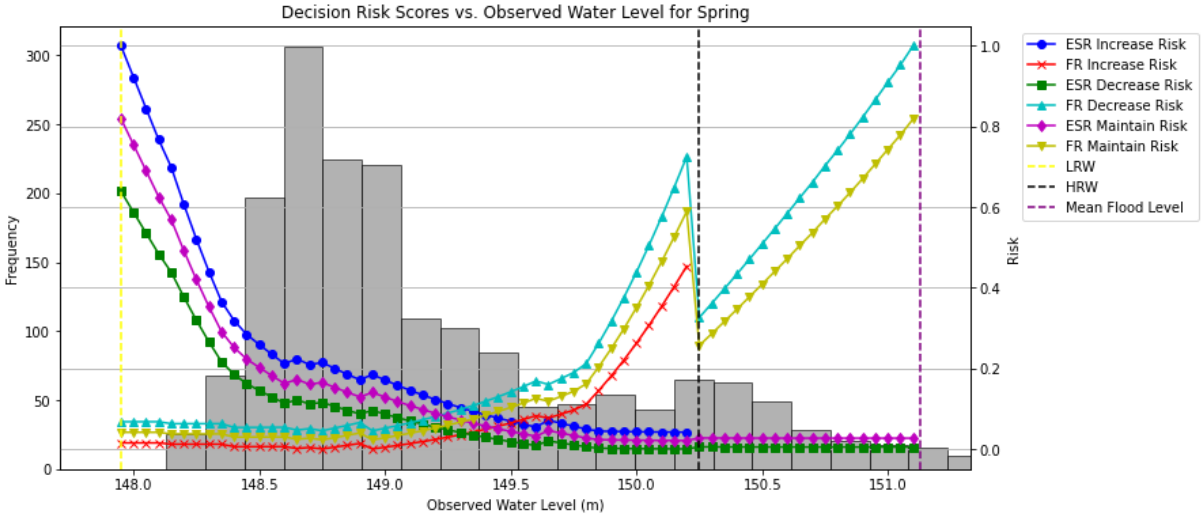


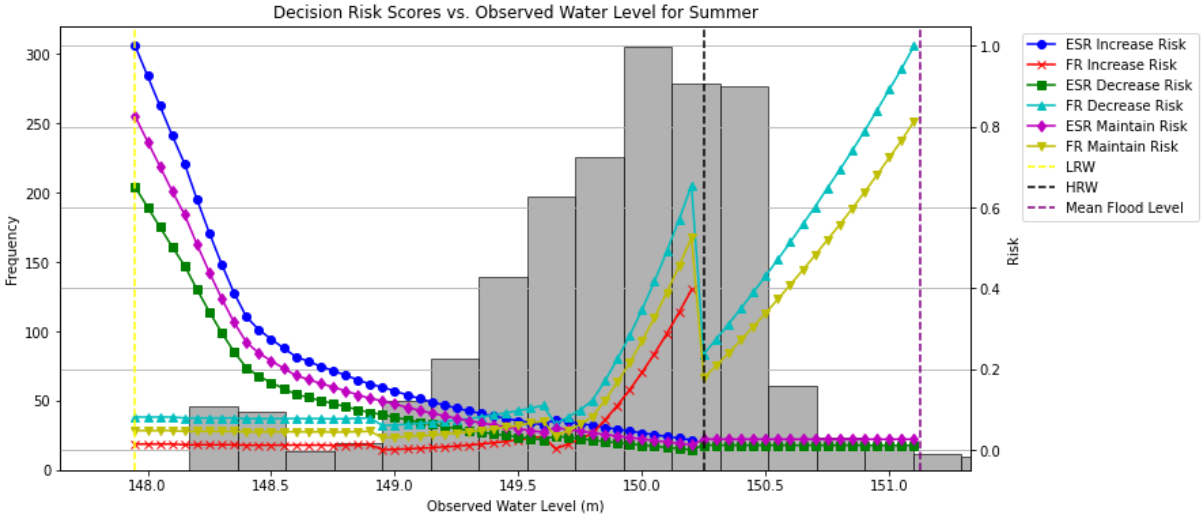


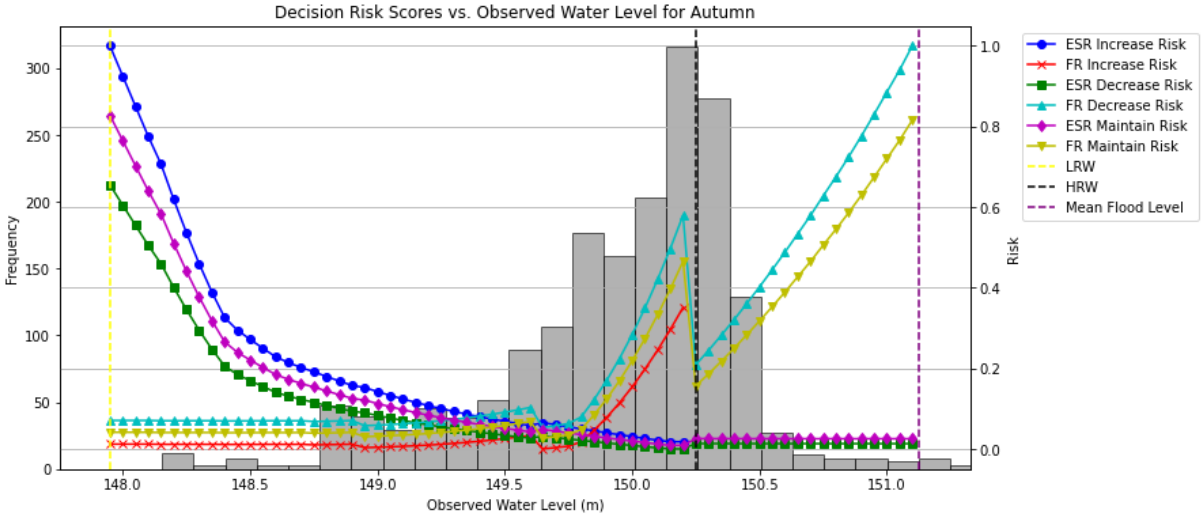

Decision Risk Scores vs. Observed Water Level for Autumn
Frequency
Risk
Observed Water Level (m)
ESR Increase Risk
FR Increase Risk
ESR Decrease Risk
FR Decrease Risk
ESR Maintain Risk
FR Maintain Risk
LRW
HRW
Mean Flood Level

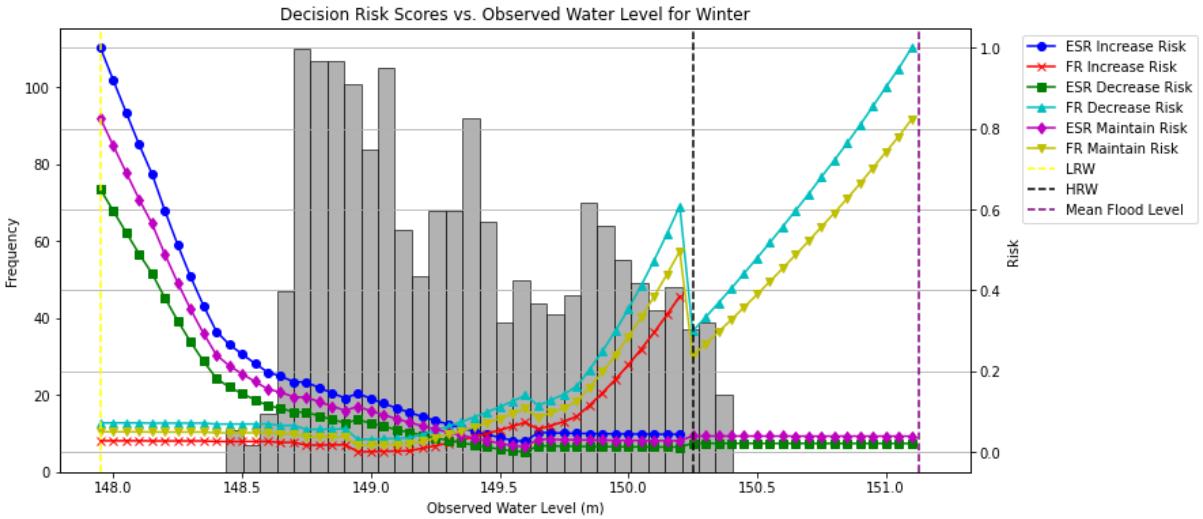

Decision Risk Scores vs. Observed Water Level for Winter
Frequency
Risk
Observed Water Level (m)
ESR Increase Risk
FR Increase Risk
ESR Decrease Risk
FR Decrease Risk
ESR Maintain Risk
FR Maintain Risk
LRW
HRW
Mean Flood Level

Sperillen:

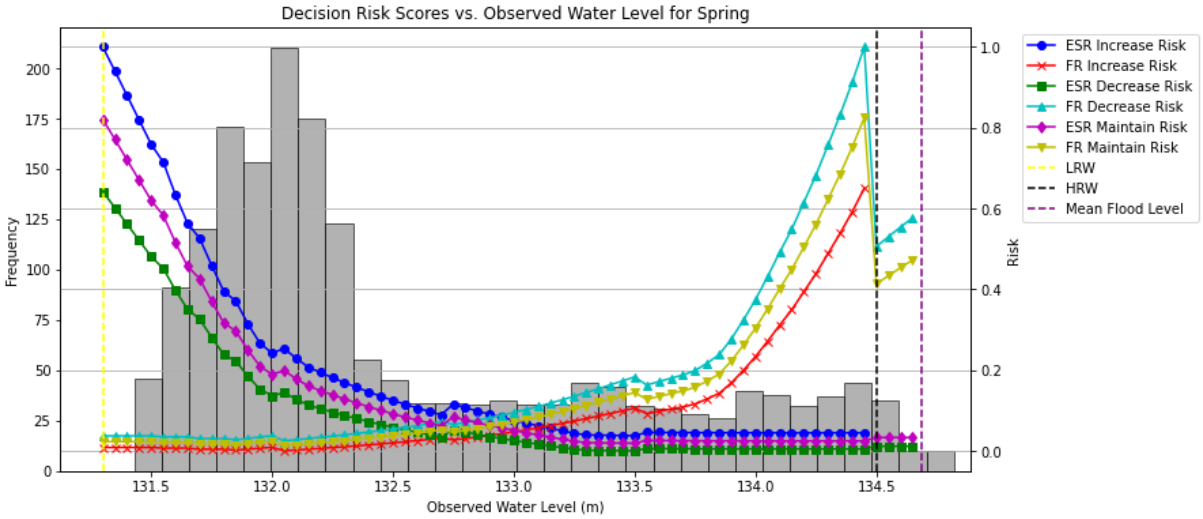

Decision Risk Scores vs. Observed Water Level for Spring
Frequency
Risk
Observed Water Level (m)
ESR Increase Risk
FR Increase Risk
ESR Decrease Risk
FR Decrease Risk
ESR Maintain Risk
FR Maintain Risk
LRW
HRW
Mean Flood Level

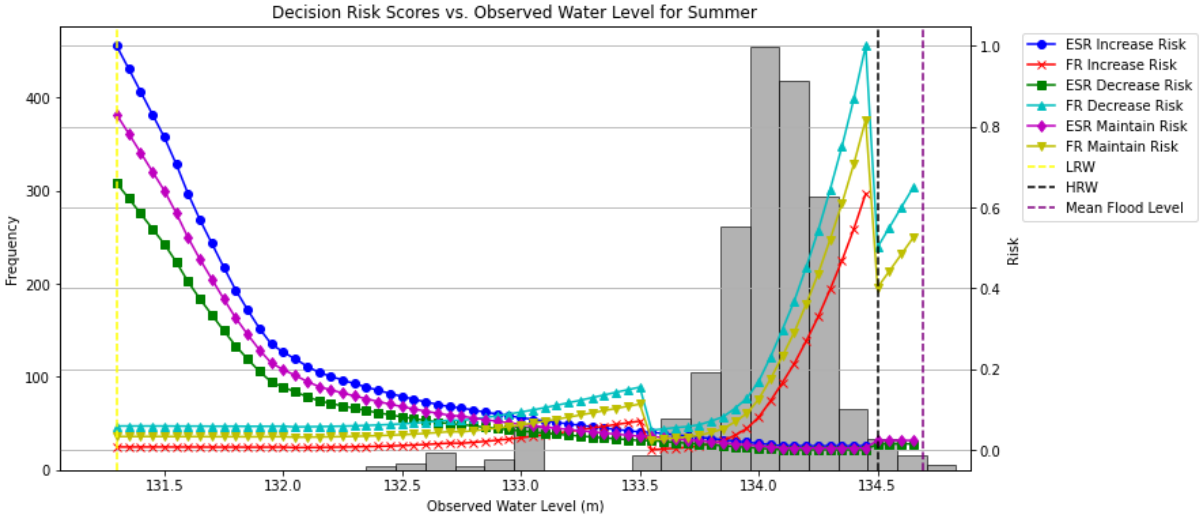

Decision Risk Scores vs. Observed Water Level for Summer
Frequency
Risk
Observed Water Level (m)
ESR Increase Risk
FR Increase Risk
ESR Decrease Risk
FR Decrease Risk
ESR Maintain Risk
FR Maintain Risk
LRW
HRW
Mean Flood Level

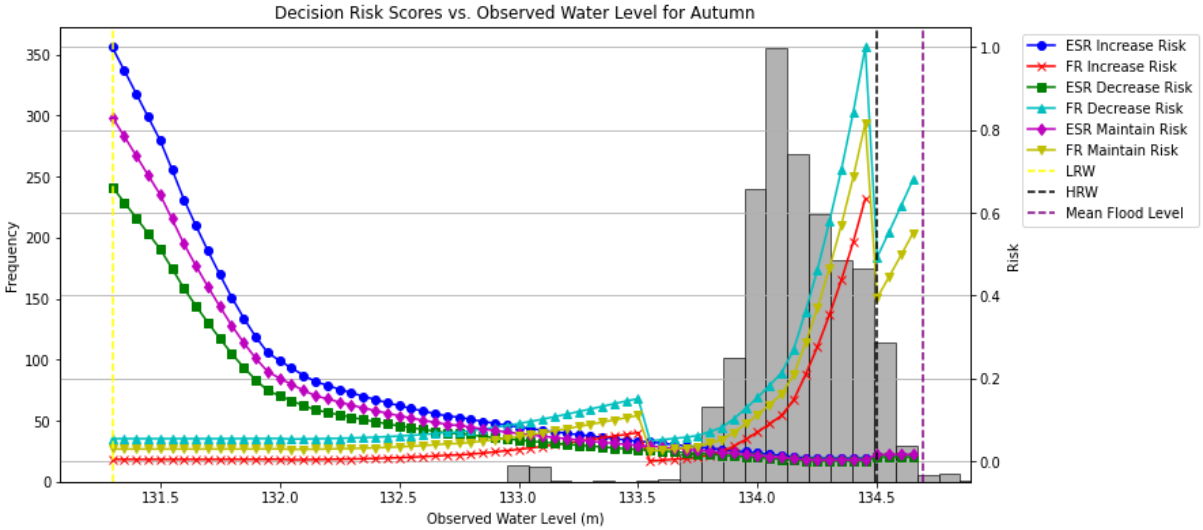

Decision Risk Scores vs. Observed Water Level for Autumn
Frequency
Risk
Observed Water Level (m)
ESR Increase Risk
FR Increase Risk
ESR Decrease Risk
FR Decrease Risk
ESR Maintain Risk
FR Maintain Risk
LRW
HRW
Mean Flood Level

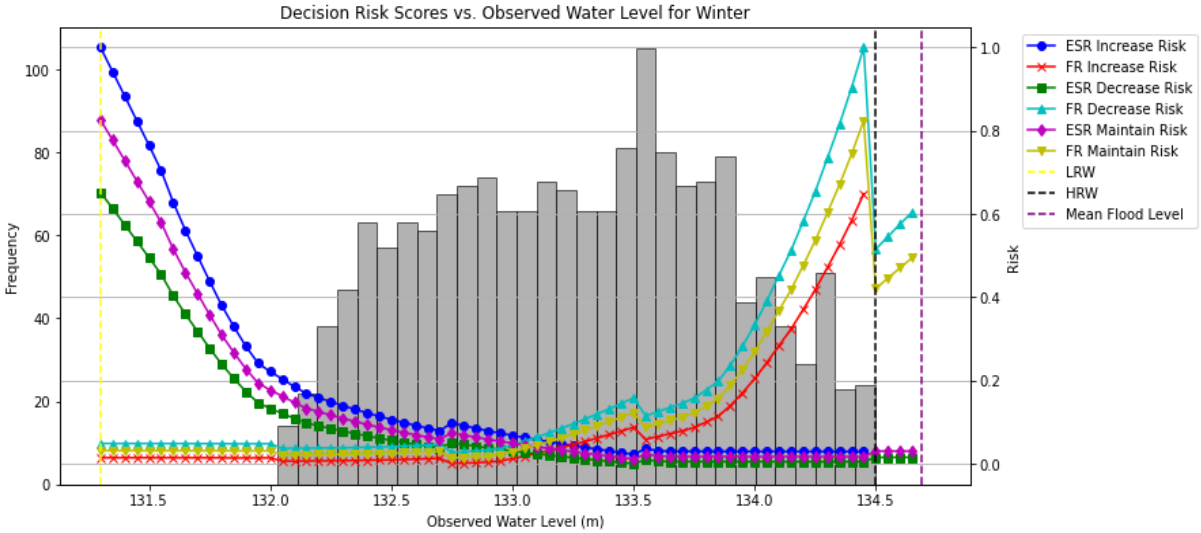

Decision Risk Scores vs. Observed Water Level for Winter
Frequency
Risk
Observed Water Level (m)
ESR Increase Risk
FR Increase Risk
ESR Decrease Risk
FR Decrease Risk
ESR Maintain Risk
FR Maintain Risk
LRW
HRW
Mean Flood Level

Randsfjorden:








Decision Risk Scores vs. Observed Water Level for Winter
Frequency
Risk
Observed Water Level (m)
ESR Increase Risk
FR Increase Risk
ESR Decrease Risk
FR Decrease Risk
ESR Maintain Risk
FR Maintain Risk
LRW
HRW
Mean Flood Level

Appendix 23:

Sensitivity_Analysis.py

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
# Example lake name and current conditions
lake_name = "Randsfjord"
season = "Autumn"
# Load combined dataset
combined_df = pd.read_csv(f'/Users/simen/Desktop/Complete Master/01 Data/Cleaned
Data/combined_df_{lake_name}_{season}.csv')
seasonal_adjustment_flood_range = np.linspace(combined_df['Seasonal Adjustment Factor for Flood (S_flood)'].mean() *
0.9,
                                              combined_df['Seasonal Adjustment Factor for Flood (S_flood)'].mean() * 1.1, 5)
seasonal_adjustment_energy_range = np.linspace(combined_df['Seasonal Adjustment Factor for Energy (S_energy)'].mean()
* 0.9,
                                               combined_df['Seasonal Adjustment Factor for Energy (S_energy)'].mean() * 1.1, 5)
density_adjustment_flood_range = np.linspace(combined_df['Flood Density Adjustment (H)'].mean() * 0.9,
                                             combined_df['Flood Density Adjustment (H)'].mean() * 1.1, 5)
density_adjustment_energy_range = np.linspace(combined_df['Energy Density Adjustment (H)'].mean() * 0.9,
                                              combined_df['Energy Density Adjustment (H)'].mean() * 1.1, 5)

# Initialize results list
sensitivity_results = []

# Perform sensitivity analysis
for seasonal_adjustment_flood in seasonal_adjustment_flood_range:
    for seasonal_adjustment_energy in seasonal_adjustment_energy_range:
        for density_adjustment_flood in density_adjustment_flood_range:
            for density_adjustment_energy in density_adjustment_energy_range:
                for index, row in combined_df.iterrows():
                    # Extract baseline values
                    baseline_ESR = row['Baseline Energy Shortage Risk']
                    baseline_FR = row['Baseline Flood Risk']
                    energy_density_adjustment = density_adjustment_energy
                    flood_density_adjustment = density_adjustment_flood
                    energy_capacity_factor = row['Energy Capacity Factor (C)']
                    flood_capacity_factor = row['Flood Capacity Factor (C)']
                    energy_penalty_factor = row['Energy Penalty Factor (R)']
                    flood_penalty_factor = row['Flood Penalty Factor (R)']
                    seasonal_adjustment_factor_energy = seasonal_adjustment_energy
                    seasonal_adjustment_factor_flood = seasonal_adjustment_flood
```

```
            # Calculate final risk scores
            final_ESR = baseline_ESR * energy_density_adjustment * energy_capacity_factor * energy_penalty_factor *
seasonal_adjustment_factor_energy
            final_FR = baseline_FR * flood_density_adjustment * flood_capacity_factor * flood_penalty_factor *
seasonal_adjustment_factor_flood

            sensitivity_results.append({
                'Observed Water Level': row['Observed Water Level'],
                'Seasonal Adjustment Factor for Flood': seasonal_adjustment_flood,
                'Seasonal Adjustment Factor for Energy': seasonal_adjustment_energy,
                'Flood Density Adjustment': density_adjustment_flood,
                'Energy Density Adjustment': density_adjustment_energy,
                'Final Energy Shortage Risk (ESR)': final_ESR,
                'Final Flood Risk (FR)': final_FR
            })

# Convert results to DataFrame
sensitivity_df = pd.DataFrame(sensitivity_results)
# Plot sensitivity analysis results for seasonal adjustment factors
plt.figure(figsize=(12, 6))
for factor in ['Seasonal Adjustment Factor for Flood', 'Seasonal Adjustment Factor for Energy']:
    factor_df = sensitivity_df.groupby(factor).mean().reset_index()
    plt.plot(factor_df[factor], factor_df['Final Energy Shortage Risk (ESR)'], label=f'Average ESR - {factor}', marker='o')
    plt.plot(factor_df[factor], factor_df['Final Flood Risk (FR)'], label=f'Average FR - {factor}', marker='x')
plt.xlabel('Factor Value')
plt.ylabel('Average Risk')
plt.title('Sensitivity Analysis of Risk Factors: Seasonal Adjustment Factors')
plt.legend()
plt.grid(True)
plt.show()
# Plot sensitivity analysis results for density adjustments
plt.figure(figsize=(12, 6))
for factor in ['Flood Density Adjustment', 'Energy Density Adjustment']:
    factor_df = sensitivity_df.groupby(factor).mean().reset_index()
    plt.plot(factor_df[factor], factor_df['Final Energy Shortage Risk (ESR)'], label=f'Average ESR - {factor}', marker='o')
    plt.plot(factor_df[factor], factor_df['Final Flood Risk (FR)'], label=f'Average FR - {factor}', marker='x')
plt.xlabel('Factor Value')
plt.ylabel('Average Risk')
plt.title('Sensitivity Analysis of Risk Factors: Density Adjustments')
plt.legend()
plt.grid(True)
plt.show()
# Local Sensitivity Analysis: Plot each factor separately
for factor in ['Seasonal Adjustment Factor for Flood', 'Seasonal Adjustment Factor for Energy', 'Flood Density Adjustment',
'Energy Density Adjustment']:
```

```
    plt.figure(figsize=(12, 6))
    factor_df = sensitivity_df.groupby(factor).mean().reset_index()
    plt.plot(factor_df[factor], factor_df['Final Energy Shortage Risk (ESR)'], label=f'Average ESR - {factor}', marker='o')
    plt.plot(factor_df[factor], factor_df['Final Flood Risk (FR)'], label=f'Average FR - {factor}', marker='x')
    plt.xlabel(f'{factor} Value')
    plt.ylabel('Average Risk')
    plt.title(f'Local Sensitivity Analysis of {factor}')
    plt.legend()
    plt.grid(True)
    plt.show()
# Global Sensitivity Analysis: Simultaneously vary multiple parameters
# Compute mean and standard deviation of final risks for each combination
grouped_sensitivity_df = sensitivity_df.groupby(['Seasonal Adjustment Factor for Flood', 'Seasonal Adjustment Factor for Energy', 'Flood Density Adjustment', 'Energy Density Adjustment']).agg({'Final Energy Shortage Risk (ESR)': ['mean', 'std'], 'Final Flood Risk (FR)': ['mean', 'std']}).reset_index()
# Plot global sensitivity analysis results
plt.figure(figsize=(12, 6))
plt.errorbar(grouped_sensitivity_df['Seasonal Adjustment Factor for Flood'], grouped_sensitivity_df['Final Energy Shortage Risk (ESR)']['mean'], yerr=grouped_sensitivity_df['Final Energy Shortage Risk (ESR)']['std'], label='ESR - Seasonal Adjustment Factor for Flood', fmt='o')
plt.errorbar(grouped_sensitivity_df['Seasonal Adjustment Factor for Energy'], grouped_sensitivity_df['Final Energy Shortage Risk (ESR)']['mean'], yerr=grouped_sensitivity_df['Final Energy Shortage Risk (ESR)']['std'], label='ESR - Seasonal Adjustment Factor for Energy', fmt='x')
plt.errorbar(grouped_sensitivity_df['Flood Density Adjustment'], grouped_sensitivity_df['Final Flood Risk (FR)']['mean'], yerr=grouped_sensitivity_df['Final Flood Risk (FR)']['std'], label='FR - Density Adjustment for Flood', fmt='s')
plt.errorbar(grouped_sensitivity_df['Energy Density Adjustment'], grouped_sensitivity_df['Final Flood Risk (FR)']['mean'], yerr=grouped_sensitivity_df['Final Flood Risk (FR)']['std'], label='FR - Density Adjustment for Energy', fmt='d')
plt.xlabel('Parameter Value')
plt.ylabel('Average Risk with Std Dev')
plt.title('Global Sensitivity Analysis of Risk Factors')
plt.legend()
plt.grid(True)
plt.show()
```

Appendix 24 External Reports and Sources

012.E2 . KOB 577 TBR

Tillatelse

for

Foreningen til Randsfjords Regulering

TIL FORTSATT REGULERING AV RANDSFJORDEN

(MEDDELT VED REGJERINGENS RESOLUSJON AV 12. JANUAR 1995)

Ved regjeringens resolusjon av 12. januar 1995 er bestemt:

«1. I medhold av vassdragsreguleringsloven av 14. desember 1917 nr. 17 gis Foreningen til Randsfjords Regulering konsesjon for fortsatt regulering av Randsfjorden på de vilkår som er tatt inn i Nærings- og energidepartementets foredrag av 12. januar 1995.
2. Det fastsettes manøvreringsreglement i samsvar med det inntatte utkast i foredraget.»

NVE
Norges vassdrags- og energidirektorat

Olje- og energidepartementet
Postboks 8148 Dep
0033 OSLO

Vår dato: 14.12.2022
Vår ref.: 200800384-42 Oppgis ved henvendelse
Deres ref.:

Foreningen til Randsfjords Reguleringen - Endelig fastsettelse av manøvreringsreglement for Randsfjorden – NVEs innstilling

NVE anbefaler at manøvreringsreglement for reguleringen av Randsfjorden justeres i tråd med regulantens forslag. Regulant er Foreningen til Randsfjords Regulering (FRR). NVE har i hovedsak sett på forholdene til storørret samt flomproblematikk i vår vurdering, og balansert dette opp mot landskap, naturmangfold, friluftsliv, landbruk og ev. ulemper for regulant. Vi anbefaler at minstevannføringen i Randselva på 20 m^3/s skal gjelde for hele året og at det innføres en bestemmelse om maksimal vannføringsreduksjon (maks 4 m^3/s hver time) ved vannføringer under 32 m^3/s. For å tilpasse manøvreringsreglementet i forhold til flomproblematikk anbefaler vi å forskyve tidspunktet for å nå LRV fra 10. april til 1. april, samt ta i bruk den definerte «selvreguleringskurven» (vassdraget naturlige flomvannføring) ved avledning av flommer.

Sammendrag

Tillatelse til fortsatt regulering av Randsfjorden ble gitt til Foreningen til Randsfjords regulering (FRR) ved kgl.res. av 12.januar 1995. Med konsesjonen fulgte et prøvereglement, som skulle tas opp til ny vurdering etter en driftstid på 5 år. I den forbindelse har Foreningen til Randsfjords Regulering utarbeidet forslag til nytt manøvreringsreglement. Forslaget har vært kunngjort og dokumentet med vedlegg vært ute på offentlig høring. Etter første høringsrunde ble det ytret sterke ønsker om ytterlige undersøkelser og kunnskap om vassdraget. Dette ble gjennomført og forslag om nytt manøvreringsreglement ble justert litt og sendt ut på ny offentlig høring med tilleggsundersøkelsene.

FRR har i hovedsak foreslått fire endringer fra dagens reglement. For å bedre forholdene til storørret, har FRR foreslått at minstevannføringen i Randselva på 20 m^3/s skal gjelde for hele året og at det innføres en bestemmelse om maksimal vannføringsreduksjon (maks 4 m^3/s hver time) ved vannføringer under 32 m^3/s. For å tilpasse manøvreringsreglementet i forhold til flomproblematikk foreslår FRR å forskyve tidspunktet for å nå LRV fra 10. april til 1. april, samt ta i bruk den definerte «selvreguleringskurven» (vassdraget naturlige flomvannføring) ved avledning av flommer.

Reglement

for

manøvrering av reguleringsdammen for Sperillen.

Approbert ved kgl. resolusjon av 7de mai 1926.

§ 1.

Når vannet om våren begynner å stige, skal alle nåler være optatt og bukkene holdes nedlagt, inntil vårflommen er avløpet, og vannstanden er gått ned til 4,70 m. ovenfor dammen (154,70 m. o. h.).

Når vannstanden synker under denne høide, reises hurtigst bukkene fra slusen utover til midtpillaren, og nålene settes litt efter litt og så tett som mulig under iakttagelse av, at vannstanden i Sperillen ikke stiger over regulert vannstand, motsvarende 4,70 m. ovenfor dammen.

Eftersom vannstanden viser tilbøielighet til å ville synke, reises bukkene også fra østre landkar — en og en — og nålene settes tett efter hvert, det siste av hensyn til tømmeret, som ellers kan ville gå under lensen.

§ 2.

Tapning utover sommeren og høsten foregår efter brukseierforeningens bestemmelse under hensyntagen til bedriftene og fløtningen, til hvis fremme til Hønefoss avgis det nødvendige vann, idet den for dampskibstrafikken til Sørum nødvendige minimumsvannstand søkes opretholdt, så lenge sådan trafikk ikke hindres av is eller for liten vannføring i Bægna elv. Tapningen må imidlertid ikke skje i sådan utstrekning, at slusen ikke kan passeres av fartøier som de hittil i Ådalsvassdraget almindelig benyttede grundtgående motorbåter og føringsbåter.

§ 3.

Vannstanden må ikke overstige den tillatte reguleringshøide 4,70 m. ovenfor dammen, uten at den hele reguleringsdam er nedlagt.

§ 4.

Tapningen skjer såvidt mulig jevnt utover vinteren efter brukseierforeningens bestemmelse. Herunder bør såvidt mulig iakttas, at alt inndemmet vann er uttappet innen utgangen av april.

§ 5.

På søn- og helligdager vil den regulerte vannføring i tiden fra 1ste desember til 1ste mai kunne reduceres, hvis man ikke derved påfører de i vassdraget interesserte skade eller ulempe.

§ 6.

Dam- og slusevokteren, som må være norsk statsborger, ansettes av brukseierforeningens bestyrelse. Ansettelsen må approberes av Arbeidsdepartementet. Han har forøvrig å foreta observasjoner over vannstanden ovenfor og nedenfor dammen, nedbør og temperatur, hvilke daglig innføres i en dertil innrettet journal. I denne blir også å innføre, hvordan dammen manøvreres.

Hver 1ste og 15de i måneden tilstiller han vassdrags- og fløtningsdirektøren og formannen i Foreningen til Bægnavassdragets regulering en bekreftet utskrift av journalen.

Ennvidere har han å påse, at sluse- og damanlegg til enhver tid er i forsvarlig stand, spesielt, at der ikke finnes lekasjer i grunndammen, eller finner utgravning sted. Inntreffer sådant, har han uopholdelig å underrette brukseierforeningens formann eller andre foresatte herom, samt på egen hånd efter beste evne søke å utbedre skaden og forebygge videre sådan. Tilfeller av den art blir å anmerke i journalen. Endelig har han uopholdelig å anmelde overtredelse av gjeldende damreglement for vassdrags- og fløtningsdirektøren.

Ved falsk journalførsel eller ved grovt brudd på reglementet kan dam- og slusevokteren avskjediges av Arbeidsdepartementet eller av brukseierforeningens bestyrelse.

Dam- og slusevokteren må ikke være fraværende uten efter permisjon. Sådan meddeles for inntil 14 dager av brukseierforeningens formann. Permisjon for lengere tid meddeles av den samme, men approbasjon fra vassdrags- og fløtningsdirektøren, eller den han dertil bemyndiger, blir da å innhente. Ved hver permisjon må godkjent stedfortreder stilles.

§ 7.

Forandringer i dette reglement kan kun foretas av Kongen, efterat fellesfløtningsforeningene, brukseierforeningen og statsbanene samt de interesserte kommuner har hatt anledning til å uttale sig.

for

manøvrering av reguleringsdammen for Sperillen.

Approbert ved kgl. resolusjon av 7de mai 1926.

§ 1.

Når vannet om våren begynner å stige, skal alle nåler være optatt og bukkene holdes nedlagt, inntil vårflommen er avløpet, og vannstanden er gått ned til 4,70 m. ovenfor dammen (154,70 m. o. h.).

Når vannstanden synker under denne høide, reises hurtigst bukkene fra slusen utover til midtpillaren, og nålene settes litt efter litt og så tett som mulig under iakttagelse av, at vannstanden i Sperillen ikke stiger over regulert vannstand, motsvarende 4,70 m. ovenfor dammen.

Eftersom vannstanden viser tilbøielighet til å ville synke, reises bukkene også fra østre landkar — en og en — og nålene settes tett efter hvert, det siste av hensyn til tømmeret, som ellers kan ville gå under lensen.

§ 2.

Tapning utover sommeren og høsten foregår efter brukseierforeningens bestemmelse under hensyntagen til bedriftene og fløtningen, til hvis fremme til Hønefoss avgis det nødvendige vann, idet den for dampskibstrafikken til Sørum nødvendige minimumsvannstand søkes opretholdt, så lenge sådan trafikk ikke hindres av is eller for liten vannføring i Bægna elv. Tapningen må imidlertid ikke skje i sådan utstrekning, at slusen ikke kan passeres av fartøier som de hittil i Ådalsvassdraget almindelig benyttede grundtgående motorbåter og føringsbåter.

§ 3.

Vannstanden må ikke overstige den tillatte reguleringshøide 4,70 m. ovenfor dammen, uten at den hele reguleringsdam er nedlagt.

§ 4.

Tapningen skjer såvidt mulig jevnt utover vinteren efter brukseierforeningens bestemmelse. Herunder bør såvidt mulig iakttas, at alt innsamlet vann er uttappet innen utgangen av april.

§ 5.

På søn- og helligdager vil den regulerte vannføring i tiden fra 1ste desember til 1ste mai kunne reduceres, hvis man ikke derved påfører de i vassdraget interesserte skade eller ulempe.

§ 6.

Dam- og slusevokteren, som må være norsk statsborger, ansettes av brukseierforeningens bestyrelse. Ansettelsen må approberes av Arbeidsdepartementet. Han har forøvrig å foreta observasjoner over vannstanden ovenfor og nedenfor dammen, nedbør og temperatur, hvilke daglig innføres i en dertil innrettet journal. I denne blir også å innføre, hvordan dammen manøvreres.

Hver 1ste og 15de i måneden tilstiller han vassdrags- og fløtningsdirektøren og formannen i Foreningen til Bægnavassdragets regulering en bekreftet utskrift av journalen.

Ennvidere har han å påse, at sluse- og damanlegg til enhver tid er i forsvarlig stand, spesielt, at der ikke finnes lekasjer i grunndammen, eller finner utgravning sted. Inntreffer sådant, har han uopholdelig å underrette brukseierforeningens formann eller andre foresatte herom, samt på egen hånd efter beste evne søke å utbedre skaden og forebygge videre sådan. Tilfeller av den art blir å anmerke i journalen. Endelig har han uopholdelig å anmelde overtredelse av gjeldende damreglement for vassdrags- og fløtningsdirektøren.

Ved falsk journalførsel eller ved grovt brudd på reglementet kan dam- og slusevokteren avskjediges av Arbeidsdepartementet eller av brukseierforeningens bestyrelse.

Dam- og slusevokteren må ikke være fraværende uten efter permisjon. Sådan meddeles for inntil 14 dager av brukseierforeningens formann. Permisjon for lengere tid meddeles av den samme, men approbasjon fra vassdrags- og fløtningsdirektøren, eller den han dertil bemyndiger, blir da å innhente. Ved hver permisjon må godkjent stedfortreder stilles.

§ 7.

Forandringer i dette reglement kan kun foretas av Kongen, efterat fellesfløtningsforeningene, brukseierforeningen og statsbanene samt de interesserte kommuner har hatt anledning til å uttale sig.

012.D0 , KDB: 1103

Kopi:

DET KONGELIGE NÆRINGS- OG ENERGIDEPARTEMENT

KONTOR: PLØENS GT. 8 - TLF. 22 34 90 90 - TELEFAX 22 34 95 25/65
POSTADRESSE: POSTBOKS 8148 DEP., 0033 OSLO - TELEKS 21486 OEDER

NVE
REG.NR. 94 12420 - 1
AVD./SAKSBEH.
VK/KGA 22.APR1994
ARKIVNR.
HENLEGGES

Norges vassdrags- og energiverk
Postboks 5091 Majorstua
0301 Oslo

Deres ref

Vår ref. (bes oppgitt ved svar)
NOE 93/10254 EV ØJ

Dato 21 APR. 1994

FORENINGEN TIL TYRIFJORDS REGULERING. REVISJON AV MANØVRERINGSREGLEMENT FOR TYRIFJORDEN

Ved kongelig resolusjon av 8. april 1994 ble bestemt:

" I medhold av lov om vassdragsreguleringer av 14. desember 1917 nr. 17 endres manøvreringsreglementet for reguleringsdammen for Tyrifjorden i henhold til forslag inntatt i Nærings- og energidepartementets foredrag av 08.04.1994. "

//. Vedlagt følger kopi av departementets bemerkninger, samt det oppdaterte manøvreringsreglement.

/. Saksdokumentene returneres vedlagt.

Etter fullmakt

P. H. Høisveen

Harald Solli

(FTRNVE.SAM)

012.DO , KDB: 1103

Kopi:

DET KONGELIGE NÆRINGS- OG ENERGIDEPARTEMENT

KONTOR: PLØENS GT. 8 - TLF. 22 34 90 90 - TELEFAX 22 34 95 25/65
POSTADRESSE: POSTBOKS 8148 DEP., 0033 OSLO - TELEKS 21486 OEDER N

NVE
REG.NR. 94 12420 - 1
AVD./SAKSBEH.
VK/KGA 22.APR1994
ARKIVNR.
HENLEGGES

Norges vassdrags- og energiverk
Postboks 5091 Majorstua
0301 Oslo

Deres ref

Vår ref. (bes oppgitt ved svar)
NOE 93/10254 EV ØJ

Dato 21 APR. 1994

FORENINGEN TIL TYRIFJORDS REGULERING. REVISJON AV MANØVRERINGSREGLEMENT FOR TYRIFJORDEN

Ved kongelig resolusjon av 8. april 1994 ble bestemt:

" I medhold av lov om vassdragsreguleringer av 14. desember 1917 nr. 17 endres manøvreringsreglementet for reguleringsdammen for Tyrifjorden i henhold til forslag inntatt i Nærings- og energidepartementets foredrag av 08.04.1994. "

//. Vedlagt følger kopi av departementets bemerkninger, samt det oppdaterte manøvreringsreglement.

/. Saksdokumentene returneres vedlagt.

Etter fullmakt

P. H. Høisveen

Harald Solli

(FTRNVE.SAM)