WIRTSCHAFTS
UNIVERSITÄT
WIEN VIENNA
UNIVERSITY OF
ECONOMICS
AND BUSINESS

BACHELOR'S THESIS

The Evolution of Weather Warnings:
Insights from GeoSphere Austria's Forecasts

submitted by

Lisa Schuster

intended degree

Bachelor of Science (WU), BSc (WU)

Student ID number:	12107156
Degree program:	Business, Economics and Social Sciences
Supervisor:	Dipl.-Ing. Dr. Amin Anjomshoaa
Co-Supervisor:	Dipl.-Ing. Hannah Schuster

Vienna, April 2025

Contents

List of Figures

List of Tables

Abstract

Accurate weather warnings are vital for minimizing the impacts of extreme weather events on public safety and infrastructure. This thesis focuses on the analysis of changed weather warnings issued by GeoSphere Austria, to examine how these warnings evolve over time to improve forecasting accuracy and reliability. With GeoSphere's historical dataset, which tracks the progression of warnings from their initial issuance to their final state, the study identifies patterns in geographic scope, severity levels, numerical adjustments, and cancellations.

The methodology involves pairing the first and last entries of each changed warning to evaluate key modifications, while accounting for unique cases like cancellations. Municipality-specific vectors are constructed to capture localized warning dynamics to provide insights into how these changes vary across regions. The analysis also explores lead times, to distinguish between proactive and reactive warning adjustments and evaluates the frequency and nature of same day updates.

The analysis revealed a nuanced approach in GeoSphere's initial warnings. While warnings often start with a broader geographic scope and are refined to focus on fewer areas as forecasts become more precise, there were also cases where the scope of warnings expanded due to updated information. Cancellations, geographic scope adjustments, and severity level changes reveal areas for improvement in the warning system. While the overall system demonstrates reliability, the frequent modifications underline the need for enhanced forecasting precision to maintain public trust.

This research contributes to a deeper understanding of GeoSphere's forecasting practices and identifies opportunities to optimize warning issuance. The results lay the groundwork for improving the timeliness, accuracy, and clarity of weather warnings, to support better preparedness for extreme weather events in Austria.

1 Introduction

Climate change is driving an increase in the intensity of extreme weather events in Central Europe, including Austria. A study by World Weather Attribution found that human-induced climate change has doubled the likelihood of intense rainfall, such as that caused by Storm Boris in September [1]. This rain lead to deadly floods in central Europe and caused significant damage in Austria. In this context, timely weather alerts play a crucial role in enabling communities to prepare for extreme weather events effectively. GeoSphere, Austria's national weather forecasting organization, informs the public about potential weather risks and issues timely warnings to mitigate their impact.

This thesis examines GeoSphere's historical dataset to conduct an in-depth analysis of extreme weather events and specifically focuses on changed weather warnings. The study aims to evaluate the accuracy and reliability of GeoSphere's forecasting system. Key factors, such as the affected municipalities, severity levels, and timing of warnings will be considered to understand the dynamics of these adjustments.

This analysis focuses on comparing the initial warnings with the final entries in the dataset to provide information on how the scope, severity, and timing evolved over time. The final update is assumed to most accurately represent the actual event, as it is the closest in time to the occurrence of the weather event. Through this analysis, the thesis seeks to identify patterns and reasons for changes in those warnings that could help improve the reliability of GeoSphere's warning system to enhance public safety and preparedness.

1.1 Importance of Weather Warnings

Weather warnings are essential tools for minimizing the impact of extreme weather events on life, property, and infrastructure [8]. They serve as early warnings that allow people, communities, and organizations to prepare and mitigate the potential consequences of severe weather. As described by NOAA's National Weather Service, "A warning is issued when a hazardous weather or hydrological event is occurring, imminent, or likely. A warning means weather conditions pose a threat to life or property. People in the path of the storm need to take protective action" [13]. This highlights the critical role of warnings for timely and decisive action to reduce harm.

The effectiveness of weather warnings depends on their clarity, accuracy, and timeliness, therefore clear communication is vital to ensure the public acts upon warnings appropriately [14]. However, challenges such as public perception, varying levels of trust in the warning system, and the inherent uncertainty in weather forecasts can affect the overall impact of warnings on safety and preparedness [14]. These challenges underline the need for continuous refinement in technical precision and public communication. This thesis investigates changed weather warnings in Austria to assess whether skepticism toward the warning system is justified.

Warnings are especially important due to the increasing climate variability and extreme weather events. The frequency and severity of events such as floods, hurricanes, and heat waves are on the rise and need systems that can predict and communicate risks with high reliability [14]. GeoSphere Austria, as the national meteorological authority, plays a pivotal role in issuing warnings that help save lives and property. However, as this thesis focuses on changed warnings, the dynamics of updates and cancellations become particularly relevant. Frequent modifications to warnings, while often necessary to reflect the evolving nature of weather conditions, can sometimes lead to confusion or skepticism among the public [14]. Evaluating the effectiveness of warnings requires an understanding of their evolution and the public's response to them. Research suggests that timely updates can enhance credibility, but inconsistent or frequent changes might diminish trust [14].

In conclusion, weather warnings are a cornerstone of public safety during extreme weather events. Their design and implementation must balance accuracy with clarity and consistency to ensure timely action. By studying how warnings evolve, including cancellations and updates, this research contributes to the broader understanding of effective warning systems.

1.2 Uncertainty in Weather Warnings

City and county emergency managers face numerous time-sensitive decisions related to hazardous weather preparations. They determine the optimal timing and usefulness of sharing weather forecasts with other local officials, such as fire department captains, public works supervisors, and school safety officials [9]. Kox et al. investigate in [6] how emergency services in Germany perceive and respond to extreme weather warnings through a survey conducted in 2012 with over 160 emergency workers. Their study examines the uncertainty associated with extreme weather forecasts and how emergency personnel interpret and act on such information. The concept of uncertainty, as outlined in the article, relates to the probability of occurrence, but also links to gaps in knowledge or limited observations.

Germany's weather warning system, which forms the basis of Kox et al.'s study, operates in three distinct stages: early warnings issued days in advance, updates as the event approaches, and final alerts on the day of the event [6]. This stepwise approach aligns closely with GeoSphere's process, which similarly includes early warnings and updates leading up to the actual weather event. Given the geographic similarities between Germany and Austria, the findings of this study are equally relevant for GeoSphere's practices.

For the interviewed emergency workers, the most critical weather extremes are strong winds, heavy rainfall, and thunderstorms. The study found that weather warnings are often underestimated, particularly when significant time passes between the initial forecast and the event itself [6]. This observation parallels a key aspect of this thesis, where it will be analyzed how late warnings are changed, as well as how many are updated or cancelled on the same day or during the event itself. Therefore, this study aims to assess whether GeoSphere's initial warnings are reliable or whether emergency personnel should prioritize later updates.

The findings of Kox et al. provide a foundational context for analyzing GeoSphere's weather warnings, particularly with respect to uncertainties and their potential perception. The observed tendency to underestimate warnings underscores the importance of optimizing the communication of weather alerts. This thesis will evaluate the accuracy of GeoSphere's changed warnings to determine whether the associated uncertainties are justified and if adjustments could enhance the reliability.

1.3 Research Objective

The objective of this thesis is to identify patterns in how GeoSphere Austria's weather warnings are adjusted over time. The analysis compares the initial forecasts to the final outcomes to identify how the warning changed over time. This involves evaluating how the number of affected municipalities changes over time, whether there are shifts in the warning levels, and identifying if the warnings tend to adopt a more cautious approach as a safety measure to minimize risk.

The goal is to gain a deeper understanding of how initial warnings evolve. The analysis compares the first warning with the last recorded update, based on the assumption that the final warning, being closest in time to the actual event, provides the most accurate representation. This evaluation also investigates whether GeoSphere prioritizes a more conservative approach to ensure public safety and how such an approach impacts the precision of the warnings. Finally, the analysis seeks to identify patterns and areas for improvement to provide a foundation for enhancing the accuracy and timeliness of GeoSphere's warning system. This can lead to more effective communication of weather risks and better preparedness for extreme weather events.

2 Background

Analyzing historical weather warnings offers insights into the reliability and evolution of GeoSphere's forecasting practices. This thesis examines the adjustments made to warnings and assesses whether these updates prioritize public safety through a cautious approach, which means that initial warnings often cover larger areas or have higher severity levels. This strategy ensures that potential risks are addressed early, even if subsequent updates refine or downgrade the warnings based on new data. The goal is to identify areas where improvements could enhance the overall effectiveness of the warning system.

2.1 The Impact of Climate Change on Extreme Weather Events

The Intergovernmental Panel on Climate Change (IPCC) has reported an increase in the frequency and intensity of extreme weather events due to climate change [10], which has led to an increase in extreme weather events such as floods, droughts, and heat waves. Disasters are becoming increasingly costly since their frequency and intensity continue to rise. Extreme weather events have effects on public health and cause issues such as heat exhaustion, respiratory problems, and injuries related to flooding [2]. Health care systems experience problems due to damaged infrastructure and the growing demand for medical services, like the need for additional staff to manage these crises effectively [2]. Therefore, precise weather forecasts are more important than ever to inform the public about potential risks and to help managing their impact [12]. This underscores the necessity for timely, accurate, and consistent weather warnings to mitigate the impacts of such events.

In this context, it becomes particularly interesting to examine whether GeoSphere effectively fulfills its role in providing accurate and timely warnings. Early and reliable forecasts could reduce pressure on emergency services, since they help communities to prepare in advance. Contrarily, the uncertainty associated with frequent changes to weather warnings could contribute to public skepticism about their reliability [7]. If warnings are perceived as inconsistent or overly cautious, they might not be taken seriously, which leads to situations where emergency services face increased strain due to a lack of public preparedness. Consequently, it is important to analyze how often GeoSphere's weather warnings were changed and how early they are issued to give insights into their impact on public preparedness and the demands placed on emergency services.

2.2 GeoSphere Austria: Mission and Contributions

Since 2023, GeoSphere Austria has combined the Central Institute for Meteorology and Geodynamics (ZAMG) and the Geological Survey of Austria (GBA), creating the national authority for meteorology, climatology, and geology and combines over 300 years of expertise to tackle challenges like climate change, natural hazards, and sustainable resource management [4]. GeoSphere's network of monitoring stations enables accurate data collection, modeling, and forecasting to enhance public safety and resilience [3].

In 2023, GeoSphere issued 200 red-level and 30,000 orange-level warnings, which indicates their active monitoring in response to observed extreme weather events. GeoSphere also supports global initiatives, such as the United Nations SOFF program, to establish robust weather monitoring networks, to provide universal access to reliable warnings by 2027 [4].

Beyond forecasting, GeoSphere addresses geological risks like landslides and earthquakes and conducts climatological research on impacts of global warming and strategies for mitigation and adaptation [5]. GeoSphere's goal is to ensure that Austria and its global partners are better prepared for environmental challenges to contribute to a safer and more sustainable future [5]. GeoSphere's website (`https://data.hub.geosphere.at/`) offers direct access to data with detailed records of past weather warnings. This thesis focuses specifically on analyzing changes to these warnings to gain deeper insights into GeoSphere's practices and to identify areas where adjustments have been made.

3 Methodology

In this section, we present a structured methodology to analyze the evolution of weather warnings over time. The analysis focused on identifying changes in geographic scope, severity levels, numerical values, and time, with separate consideration for cancelled and changed warnings. The aim was to identify trends and areas for improvement in GeoSphere's forecasting practices. The process included data preprocessing, analysis of distinct pairs of cancelled and changed warnings, and the creation of warning-type-specific vectors to provide an overview of the observed changes.

3.1 Analysis Approach

The analysis seeks to uncover whether warnings become more or less severe over time, while distinguishing between changed and cancelled warnings for a more precise evaluation.

Geographic changes involved assessing the number of impacted municipalities to determine whether warnings initially cover a broad area, reflecting a cautious approach, or whether their geographic scope expands as more data becomes available. Changes in severity levels, values, status, and warning types were analyzed by comparing initial and final attributes for each sequence to identify whether these levels increased, decreased, or remained stable. By separately analyzing changed and cancelled warnings, the study provided a clearer understanding of how each type of warning evolves and impacts the overall system.

Timing was another critical focus of the analysis. The lead time between the issuance of warnings and the predicted weather events was evaluated to assess GeoSphere's effectiveness in providing timely preparation. The frequency of same-day issuances and same-day modifications was also analyzed to evaluate responsiveness. Furthermore, the analysis examined whether the warning timeframes changed to provide further insights into how the forecasts were adjusted to meet evolving conditions.

The primary goal was to gain a deeper understanding of GeoSphere's forecasting methods. An evaluation of how both changed and cancelled warnings evolve helped to determine whether the system has areas for improvement to create a more reliable and effective warning system to offer clearer communication to the public and enhance overall preparedness.

3.2 Description of the Weather Warning Data

The dataset from GeoSphere contains weather warning data from 2011 to 2023 and forms the basis for analyzing the progression and evolution of these weather warnings. Each warning is uniquely identified by a *Warning-ID*, represented by an integer that distinguishes individual weather events. Within each *Warning-ID*, the *Sequence-ID* distinguishes individual warnings that belong to the same weather event. Each *Sequence-ID* represents a separate warning issued under the same *Warning-ID*, meaning multiple warnings can exist within a single weather event, each with its own *Sequence-ID*. If a warning for a particular region is updated, it retains the same *Sequence-ID*.

The *Change-ID* indicates if a change was made to a warning. This parameter helps to differentiate between unchanged warnings and those that were updated and reflects adjustments in scope, severity, or other attributes during the warning period. The *Status* field indicates whether a warning is currently active ("aktiv") or has been cancelled ("aufgehoben").

The dataset includes the start and end times of each weather event, specifying when the event is predicted to begin and conclude. The *Created* and *Created File* columns both record the timestamps of when each warning was issued, but they differ in precision and origin. While the *Created File* column provides exact timestamps down to the second, extracted directly from the file name, the *Created* column contains rounded timestamps for a broader reference. Essentially, one timestamp comes from the file's metadata, while the other is derived from the file name itself.

Furthermore, the *Municipality* field lists the affected municipalities as integers that represent different geographic regions in Austria. This allows for an analysis of the geographic scope of warnings and any subsequent changes over time.

The *Warning Type* categorizes the warnings into specific weather phenomena with the following values:

- 1: Storm,
- 2: Rain,
- 3: Snow,
- 4: Black ice,
- 5: Thunderstorm,
- 6: Heat,
- 7: Cold.

The *Value* column provides a numerical measure that quantifies the expected intensity of the weather event, based on the specific type of warning. It does not indicate severity but rather the forecasted amount or magnitude of the phenomenon. Finally, the *Warning Level* field captures the severity of each warning on a scale of one to three. A value of 1 represents low severity (yellow), 2 indicates moderate severity (orange), and 3 corresponds to high severity (red). This analysis will focus on changes in status, times, affected municipalities, value, and warning level.

The analysis began with preprocessing the provided data set to ensure consistency and readability. Since the original dataset contained fields in German, the column names were renamed in English for clarity. Table 1 shows the original column names along with the new names.

Original Column Name	**Renamed Column Name**
warnid	Warning-ID
chgid	Change-ID
verlaufid	Sequence-ID
status	Status
begin	Start Time
ende	End Time
erstellt	Created
created_file	File Created
gemeinden	Municipalities
warntyp	Warning Type
wert	Value
warnstufe	Warning Level

It was found that missing values only occurred in rows where the warning status was "cancelled". In these cases, all values were missing except for *Status*, *Warning-ID*, *Change-ID*, and *File Created*. This made it easy to address the missing values without the need for complex manipulation. For the missing *Sequence-ID*, the appropriate values from the warnings were inserted. Empty numeric fields, such as *Warning Type*, *Value*, and *Warning Level*, were filled with 0. Similarly, empty entries in the *Municipalities* field were replaced with an empty list ([]), and missing timestamps in the *Start Time* and *End Time* fields were assigned placeholder values of 1970-01-01 00:00:00. Since the missing values only occurred in cancelled warnings, this approach ensured a consistent and error-free dataset without the need for additional precautions.

Data types were standardized, with numerical columns converted to integers and date-related fields reformatted into the datetime64format. This step ensured that the dataset was clean, structured, and ready for analysis to enable efficient handling of temporal, numerical, and categorical data.

3.3 Pairing First and Last Warnings

The aim of this analysis was to compare the initial and final warnings for each sequence to identify changes in scope, severity, and other attributes over time. Weather warnings without any updates consisted of only a single row, which simplified filtering out unchanged warnings. Changed warnings were grouped by their Sequence-ID within each unique Warning-ID to get a clear distinction between the start and end states of a warning. Cancelled warnings were treated separately to address their unique characteristics.

For changed warnings, the first entry represented the initial forecast, while the last entry reflected the final state of the warning. In contrast, cancelled warnings posed a unique challenge, as they were issued at the Warning-ID level and applied universally to all associated Sequence-IDs. To address this, additional rows were added to pair cancellations with the appropriate Sequence-ID. This adjustment ensured that cancellations could be accurately paired with the corresponding Sequence-IDs for analysis, allowing a separate and focused evaluation of cancelled warnings.

The dataset was then sorted by Warning-ID and Sequence-ID to maintain a logical order and provide a structured foundation for further analysis. This preprocessing step ensured that all changes, including cancellations, could be analyzed systematically and provided an overview of how GeoSphere's warnings evolved over time. By separating changed and cancelled warnings, the analysis was able to provide a more detailed and precise understanding of the dynamics of both updated and discontinued warnings. While this sorting step was not strictly necessary for the analysis itself, as grouping and aggregation inherently handle the data structure, its inclusion provided a logical and intuitive sequence for following changes in warnings, particularly during initial data checks and visualizations.

3.4 Municipality Analysis Approach

Changes in the *Municipalities* field were analyzed by comparing the first and last warnings to investigate how the list of affected areas evolved over time. The analysis focused on the actual municipalities impacted, rather than just the numeric changes in their counts. To achieve this, the lists of municipalities in the first and last warnings were transformed into sets. This transformation allowed for the calculation of the symmetric difference between the two sets, mathematically represented as:

$$\Delta(A, B) = (A \setminus B) \cup (B \setminus A).$$

In this context, A represents the set of municipalities affected by the initial warning and B the set of municipalities affected by the final warning. $(A \setminus B)$ is the set of elements in A but not in B, capturing municipalities removed from the warning scope. $(B \setminus A)$ is the set of elements in B but not in A and captures municipalities newly added to the warning scope. The union $(A \setminus B) \cup (B \setminus A)$ then provides the total set of changes in the affected municipalities, regardless of whether they were added or removed.

With that, we capture the elements present in one set but not in the other. This approach ensured that any changes were detected in the actual municipalities, even in cases where the total number of affected municipalities remained the same.

In addition to analyzing the changes of specific municipalities, the changes in the number of affected municipalities were also examined. Instead of focusing on the identities of the municipalities, this analysis compared the total counts of affected municipalities in the first and last changed warnings. This provided a simpler but complementary metric for evaluating the evolution of the changed warnings.

3.5 Changes in Weather Warning Timeframes

The analysis of temporal changes in weather warnings focused on capturing adjustments to the *Start Time* and *End Time* fields separately for changed and cancelled warnings. To ensure uniformity, missing values in the cancelled dataset were replaced with a placeholder timestamp of 1970-01-01 00:00:00. This placeholder facilitated comparisons during the analysis while avoiding errors from null or empty values.

Then, each pair of rows was analyzed to compare the start times and end times, categorizing changes into three groups: identical changes for both start and end times, no changes for either field, or differing changes where one field was updated while the other remained unchanged. This distinction allowed for a better understanding of how the timing of warnings evolved.

In the analysis of cancelled warnings, placeholder timestamps in the *Created* column were often found, which made this column unsuitable for analysis. The accurate creation timestamp from the *File Created* column was used instead. The *File Created* column represents the exact time the file was created, rather than a timestamp extracted directly from the dataset. For cancelled warnings, the lead time was calculated by comparing the *File Created* timestamp with the *Start Time* of the preceding row. This adjustment ensured proper inclusion of cancelled warnings in the lead time analysis.

The lead time evaluation for changed warnings also relied on precise timestamps from the *File Created* and *Start Time* columns, this time for the same rows. The *File Created* column provided more reliable timestamps and resolved inconsistencies in the abstracted *Created* field. This ensured accurate calculations and meaningful comparisons.

Furthermore, created warnings were analyzed using the first row of each pair to evaluate how far in advance changed warnings were created before the predicted event's start time. Secondly, modified warnings were analyzed using the second row of each pair to evaluate how far in advance warnings were changed before the event's start time. For cancelled warnings, the lead time was calculated using the *File Created* timestamp compared to the *Start Time* of the preceding row. This adjustment addressed the placeholder issue, as cancelled warnings lack an actual start time.

The resulting lead times for changed and cancelled warnings were then aggregated and analyzed separately. This approach ensured the accurate treatment of each category while providing a comprehensive view of temporal adjustments in GeoSphere's weather warnings.

3.6 Evolution Vectors

To analyze how weather warnings evolved for individual municipalities and warnings, municipality-specific and warning-specific evolution vectors were created. The municipality-specific vectors captured key changes for different areas in attributes such as the first and last warning levels, differences in values, and the count of changed or cancelled warnings for each municipality. Similarly, the warning-specific vectors focused on changes within each warning and records shifts in warning levels, differences in numerical values, and the changes of timeframes.

3.6.1 Municipality-Specific Evolution Vectors

Although the list of affected municipalities could change during updates, the analysis focused on the municipalities listed in the first row. Changes for the affected municipalities during the warning lifecycle appear in the changed warnings count, which is included in the evolution vector. The separation of changed and cancelled warnings allows updates to ongoing warnings and cancellations to be tracked independently.

Each municipality has recorded counts of specific warning level transitions, value changes within predefined ranges, and the total number of changes. The dataset also includes the frequency of different warning types, such as rain, storms, or snow, to highlight which types of warnings undergo the most adjustments. Additionally, the total number of modified warnings per municipality is included, with a focus on cases where at least one attribute has changed. In the case of cancelled warnings, the change count directly corresponds to cancellations to ensure a structured comparison between evolving warnings and discontinued ones.

The final vector included a summary of the warning evolution for each municipality with details of the frequency and scale of changes in the warnings they received. This provides a framework for analyzing patterns in geographic expansion, severity upgrades or downgrades, numerical refinements, and cancellations.

3.6.2 Warning-Specific Evolution Vectors

The last part of the code was to examines how individual warnings evolve over time to reveal patterns and trends in updates and adjustments to attributes such as warning levels, numerical values, geographic coverage, and timing. The findings offer valuable insights into how warnings are managed and refined.

Initially, two separate CSV files were created to distinguish changed and cancelled warnings. This separation allowed for a focused analysis of each category. Subsequently, these datasets were merged into a single CSV file to enable a comprehensive evaluation of overall trends and comparisons between the two categories.

The final combined dataset includes key columns such as Warning-ID, Sequence-ID, Difference in Municipality Numbers, Difference in Warning Type, Difference in Warning Level, Start Time Changes, End Time Changes, Created Same Day as Start, and Modified Same Day as Start. These columns encapsulate the essential changes for each warning. The consolidated dataset is stored in a single CSV file to facilitate streamlined analysis and ensure easy access for future research.

4 Results

The following section presents a detailed analysis of the evolution of changed and cancelled weather warnings issued by GeoSphere Austria. The findings are structured to address key areas of interest, including changes in the geographic scope of warnings, warning severity, and other relevant characteristics. Each subsection highlights specific patterns and trends observed in the data and provides insight into how these warnings were adjusted over time. It becomes visible that out of 6,283 warnings, 4,418 were either changed or cancelled. This corresponds to 70.32% and shows that GeoSphere changes a majority of those warnings.

4.1 Cancelled Warnings

Cancelled warnings are a significant aspect of GeoSphere's forecasting system. They represent cases where the perceived risk level of a weather event diminished to the point that no further warnings were deemed necessary. The decision to cancel a weather warning could also stem from a meteorologist's assessment that the warning was no longer necessary. This is evident from the time elapsed between the issuance and cancellation of the warning. This status is marked as "aufgehoben" in the dataset and plays a critical role to understand the evolution of weather warnings over time.

Out of 6,283 distinct warnings in the dataset, 875 of them were identified as cancellations, which corresponds to approximately 13.93%. These cancellations were detected by comparing the status field across paired rows for each changed warning. Specifically, if the status of a warning transitioned to "aufgehoben", it was recorded as a cancellation.

Cancellations have significant implications for interpreting other metrics. For instance, when changes in warning levels, values, or affected municipalities result in a reduction to zero, this often coincides with a cancelled warning. This context is crucial to accurately understand downward trends in these metrics, as they frequently reflect a reduction in the perceived risk level rather than an adjustment in forecasting methodology. Additionally, weather forecasts generally improve in accuracy as the event nears. This refinement process contributes to both modifications and cancellations, ensuring that warnings remain aligned with the latest meteorological assessments.

The distribution of cancelled warnings underscores GeoSphere's approach that they often release a weather warning and then cancel it. Approximately 19.81% of the changed warnings were issued preemptively to prioritize public safety but were later cancelled when the risks diminished. Those changes lay the foundation to interpret other metrics.

4.2 Analysis of Municipality Changes

The analysis of changed municipalities highlights trends in how the geographic scope of weather warnings changes over time. GeoSphere begins with broad initial warnings to ensure readiness and then refines them as more precise data becomes available. The evaluation recorded 2,742 changes in the list of affected municipalities, accounting for 77.59% of the total changed warnings and 43.64% of all warnings. Changes based solely on the number of affected municipalities totaled 2,731, representing 77.28% of changed warnings and 43.47% of all warnings. The average magnitude of these changes, measured as the absolute difference in the number of municipalities between the first and last warnings, reached approximately 59.86 municipalities per warning.

The analysis revealed 1,380 warnings where the number of affected municipalities increased, which reflects updates that expanded the scope to include additional areas. In contrast, 1,351 warnings showed a reduction in scope, which often occurred as forecasts refined the focus on risk areas. Another 803 warnings retained the same number of affected municipalities between their initial and final states.

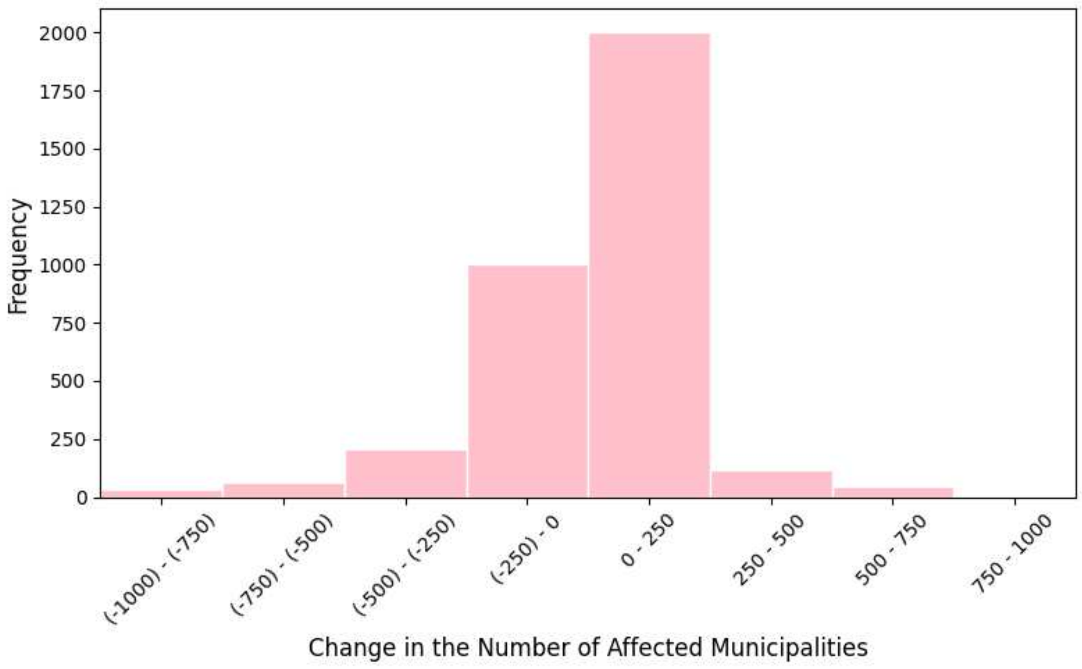


Figure 1 illustrates the distribution of these changes, with most adjustments clustered around zero. The numbers on the y-axis represent the count of warnings experiencing specific changes in the number of affected municipalities on the x-axis. The histogram shows a near-equal distribution of increases and decreases, though decreases include significant outliers where forecasts excluded large numbers of municipalities.

These extreme changes, with increases or decreases exceeding 500 municipalities, reflect substantial forecast updates influenced by unexpected developments or changing weather patterns. Notably, such warnings often have a high Change-ID, which indicates that they have been revised multiple times. This pattern suggests rapidly changing weather conditions that necessitate frequent updates and highlights the uncertainty in forecasting under such dynamic circumstances. The high frequency of changes underscores the challenges faced in accurately predicting weather events with rapidly evolving impacts. Furthermore, given that Austria comprises 2,093 municipalities [11], these warnings typically pertain to weather events that impact a significant portion of the country and often cover almost the entire nation. This widespread scope further emphasizes the scale and complexity of these extreme weather situations. An example is the warning with Warning-ID 1093 and Sequence-ID 1, which underwent 13 changes. Each of these changes was a cancellation that was later reactivated. Initially, there were only three Sequence-IDs for this warning, but later, there were nine. This subdivision could explain the reduction of 1,339 affected municipalities, as the original large-scale warning may have been split into multiple smaller warnings, each covering a more specific region. Such an extreme decrease could also indicate a significant adjustment in the forecast, suggesting that the initially affected area was overestimated. Subsequent updates could also indicate that more precise data became available, leading to a refinement of the warning. Given the magnitude of the municipality change, this warning probably concerned a weather event that initially appeared to affect almost the entire country.

GeoSphere's approach ensures flexibility and responsiveness in adapting warnings as new data emerges. The balance between increases and decreases in Figure 1 in the number of affected municipalities highlights the dynamics of weather forecasting, where updates continuously improve the precision of warnings while addressing emerging risks. These findings underline the importance of adaptive forecasting techniques and provide a basis for further optimizing warning accuracy.

The analysis of cancelled warnings revealed significant shifts in the number of affected municipalities. Across all 875 cancelled warnings, changes were observed in the geographic scope, with every instance involving a reduction in the number of impacted municipalities, which represents 13.93% of the total warnings.

The average magnitude of these changes, calculated as the absolute difference in the number of municipalities between the first and last states of the warnings, was approximately 136.4 municipalities per warning. This substantial reduction underscores the role of cancellations in refining forecasts by excluding areas initially predicted to be at risk.

Warning-ID	**Sequence-ID**	**Municipality Changes**	**Warning Type**	**Warning Level**
543	1	-1014	Rain	3
543	2	-1365	Rain	2
604	1	-1015	Storm	1
697	1	-1404	Storm	1
881	1	-1096	Thunderstorm	1

Table 2 presents five examples of cancelled warnings, in which the number of affected municipalities decreased by more than 1,000. A rain warning saw a reduction of 1,014 affected municipalities at warning level 3, while another rain warning experienced a larger decrease of 1,365 municipalities at warning level 2. A storm warning resulted in 1,015 fewer affected municipalities at level 1, and another storm warning resulted in a reduction of 1,404 municipalities, also at level 1. Additionally, a thunderstorm warning excluded 1,096 municipalities at warning level 1. These cases illustrate substantial adjustments in the geographic scope of weather warnings across different types and severity levels.

4.3 Warning Type Analysis

This part of the analysis focuses on evaluating changes in the type of warnings issued over time. Specifically, the first and last warnings for each event were compared to determine whether the type of weather event remained consistent or changed during the warning evolution. "Warning Type" refers to categories such as rain, storms, snow, heat, black ice, or thunderstorms, each represented by a specific numerical value.

The analysis shows that changes in warning type without cancellations were extremely rare. Out of 7,068 warning type rows, only 2 instances involved changes in the type of warning, representing just 0.06% of the cases where warnings changed, and 0.03% of all warnings. These two changes occurred where a rain warning shifted to a storm warning. This outcome aligns with expectations, as the type of extreme weather event is typically determined with high confidence when the warning is issued. Changes in warning type are inherently unlikely since the underlying conditions leading to these events are distinct and unlikely to evolve into entirely different phenomena.

These findings confirm the stability of GeoSphere's weather classifications. Once a warning is issued, the type of event remains consistent in nearly all cases, reflecting the system's accuracy in identifying and categorizing extreme weather events. The negligible number of changes underscores the reliability of GeoSphere's initial classifications.

The analysis of warning type changes for canceled warnings revealed that such changes occurred in all 875 cases, accounting for 13.93% of the total 6,283 warnings. These transitions exclusively involve cancellations, as the warning type is removed when a warning is no longer active. The frequency of these cancellations for each initial warning type is summarized in Table 3.

Initial Warning Type	**Final Warning Type**	**Frequency**
Rain	Cancelled	4.71%
Storm	Cancelled	4.55%
Thunderstorms	Cancelled	2.23%
Snow	Cancelled	1.85%
Black ice	Cancelled	0.59%

The findings indicate that warning cancellations are highly consistent across different warning types. Rain warnings were the most frequently cancelled, followed by storm and thunderstorm warnings.

4.4 Warning Level Analysis

The next part of the analysis focuses on adjustments made to the *Warning Level*. The results show that out of a total of 3,534 changed warnings, 523 experienced a change in the warning level. This corresponds to 14.80% of those warnings and to 8.32% of the total warnings.

Initial Warning Level	**Final Warning Level**	**Frequency**
Yellow	Orange	5.28%
Orange	Yellow	2.42%
Orange	Red	0.32%
Red	Orange	0.17%
Yellow	Red	0.06%
Red	Yellow	0.06%
Yellow	Cancelled	10.55%
Orange	Cancelled	3.07%
Red	Cancelled	0.30%

Table 4 provides a breakdown of the warning level changes and reveals several key patterns. The most frequent change from level 1 to level 2 occurred 332 times, which represents 5.28% of all cases. This shows that warnings that are issued at the lowest severity level were often escalated to moderate severity, which indicates that GeoSphere occasionally underestimates the initial severity of weather events and later adjusts the warnings upward as more data becomes available. These upward adjustments highlight GeoSphere's responsiveness to evolving weather conditions but also point to opportunities to improve the accuracy of initial forecasts.

The second most frequent change, from level 2 to level 1, corresponding to 152 occurrences and 2.42% of all cases, demonstrates that moderate-level warnings were often revised downward. Transitions from level 2 to level 3 occurred 20 times, accounting for 0. 32% of all 6,283 warnings, while transitions from level 3 to level 2 occurred 11 times, representing 0.18%. Rare changes, such as shifts between levels 1 and 3 or 3 and 1, with 4 occurrences each, representing 0.06% of all weather warnings, indicate that extreme adjustments in severity are uncommon. These cases likely represent extraordinary weather scenarios or rapidly evolving conditions that necessitate such significant changes.

While cancellations of warning changes to a lower severity are less important in this context, the substantial number of upward adjustments, particularly from level 1 to level 2, suggests room for improvement in identifying higher-severity weather risks earlier in the warning lifecycle for better public preparedness.

The analysis of warning level changes of cancelled warnings shows that all of the 875 cancelled warnings experienced changes in their warning levels. This result aligns with expectations, as cancellations inherently involve a reduction in severity or a transition to an inactive status.

The most frequent change occurred when yellow warnings were cancelled, with 663 instances which represents 10.55% of all warnings. This reflects the tendency for less severe warnings to be reevaluated and cancelled more frequently as updated information reduces the perceived risk. Orange warnings were cancelled 193 times, accounting for 3.07% of all warnings, while red warnings experienced only 19 cancellations, corresponding to 0.30%. These results indicate that higher-severity warnings, such as red alerts, are less likely to be withdrawn, as they often represent critical weather events where risks remain significant.

These changes, also summarized in Table 4, demonstrate the high frequency of cancellations at lower severity levels. They suggest that GeoSphere's early warnings aim to ensure preparedness by including potential risks. As updated data becomes available, GeoSphere adjusts its forecasts, often withdrawing warnings when the likelihood of impact diminishes. This practice reflects an effort to balance early caution with accurate targeting.

4.5 Value Analysis

In this part of the analysis, changes in the *Value* field are examined, which represent specific numerical measures associated with different types of weather warnings. Since each warning type uses a different unit of measurement, such as wind speed for storms and temperature for heat warnings, these values are not directly comparable. Therefore, we analyze each warning type separately to better understand how their value severity evolves over time.

Out of the 3,534 changed warnings in the dataset, 886 exhibited changes in their values, corresponding to 25.07% of all changed warnings and 14.10% of all warnings. This suggests that a significant portion of the warnings underwent adjustments in their forecasted severity during their lifecycle.

Furthermore, the average percentage change across all warnings is 105.51%, meaning that on average, the forecasted values of warnings increased by 5% from their initial to final entries. This indicates that, in many cases, the severity of weather events was updated upwards as more data became available. To analyze the direction of these changes, the differences were calculated as the second value minus the first value for each pair. This approach provides insights into whether the severity of warnings became more or less pronounced over time.

The average value changes for each warning type are as follows:

- Storm: 1.03 km/h,
- Rain: 2.96 mm/h,
- Snow: 1.93 cm,
- Black ice: 0.03,
- Thunderstorm: 0.04 strikes/km^2.

In absolute terms, the average changes per warning type are:

- Storm: 3.45 km/h,
- Rain: 9.39 mm/h,
- Snow: 5.53 cm,
- Black ice: 0.06,
- Thunderstorm: 0.07 strikes/km^2.

These values for changed warnings reveal, that while many of the changes in values are relatively small, some warning types, particularly storms, rain, and snow warnings, exhibit more pronounced adjustments in forecasted severity. For example, storm warnings show a noticeable increase in value, with an average change of 3.45 km/h. This suggests that storms were typically forecasted to become more severe over time. For black ice warnings, no clear unit of measurement could be determined, unlike other warning types, where changes in severity could be quantified.

Rain warnings, with an average value change of 9.39 mm/h, and snow warnings, showing an average change of 5.53 cm, also indicate relatively high variability in forecasted severity. On the other hand, black ice warnings, with an average change of just 0.06, and thunderstorm warnings, showing an average change of 0.07 strikes/km^2, indicate relatively minor adjustments in their values. This suggests that the severity of these events, in terms of the values used for warnings, tended to remain stable over time, with smaller variations or less need for drastic updates.

For the cancelled warnings, the average value change per warning type is as follows:

- Storm: -75.31 km/h,
- Rain: -50.03 mm/h,
- Snow: -28.38 cm,
- Black ice: -1.00,
- Thunderstorm: -1.11 strikes/km^2.

These values show that the magnitude of change in these warnings is substantial, particularly for storm, rain, and snow warnings. For instance, storm warnings had a significant average change of -75.31 km/h, reflecting the fact that severe weather events, such as storms, were called off due to the diminishing threat. Similarly, rain and snow warnings also exhibited notable reductions in their value, with averages of -50.03 mm/h and -28.38 cm.

In contrast, black ice and thunderstorm warnings showed smaller reductions, with changes of -1.00 and -1.11 strikes/km^2, respectively. The reductions in these warnings suggest that the initial forecasts of black ice or thunderstorms, while still relevant, did not require the same magnitude of adjustment as the larger-scale weather events like storms or heavy precipitation.

4.6 Temporal Adjustments in Weather Warnings

The analysis of temporal adjustments in weather warnings provides a deeper understanding of how GeoSphere adapts its forecasts over time. These adjustments focus on changes to the predicted start and end times of warnings.

For the changed warnings, a total of 973 start time changes, corresponding to 15.49% of the total warnings, and 1,335 end time changes, corresponding to 21.25%, were identified in the dataset. End time changes occurred more frequently, which suggests an approach to ensure warnings remain relevant as weather events often last longer or shorter than initially predicted. In contrast, start time changes were less common, indicating that initial forecasts of when events would begin were generally more reliable.

Among these adjustments, 659 instances, which make up 10.49% of all warnings, showed overlapping changes in both start and end times. Overlapping changes refer to cases where both the start and end times were updated for the same warning. This indicates that the forecast was thoroughly revised, likely to account for significant changes in the weather conditions. Such revisions suggest a proactive approach to ensure the warnings accurately reflect updated predictions for both the onset and duration of weather events.

The distribution of start and end time changes, including their overlap, provides a clearer picture of the extent to which adjustments were made to one or both temporal aspects of the warnings. The differences in the frequency of start and end time changes suggest distinct patterns. Start times were less likely to be updated, in contrast, end times were adjusted more frequently, which demonstrates the uncertainty involved in predicting the duration of weather events.

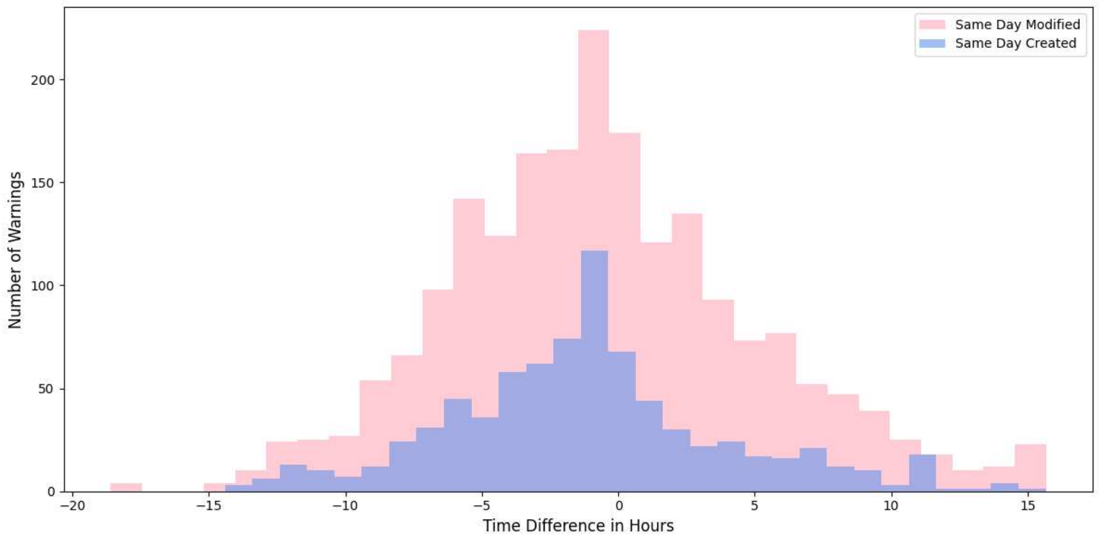
The frequent changes in end times demonstrate a focus on keeping warnings relevant, while the less frequent changes in start times suggest confidence in initial predictions unless substantial new data necessitated updates.

For the cancelled warnings, the analysis revealed that all 875 cases experienced modifications in both their start and end times. This consistent adjustment ensures that once a warning is cancelled, its entire temporal scope is updated accordingly. The identical changes in start and end times across all cancelled warnings reflect a standardized approach, where both timestamps are uniformly revised to accurately mark the warning as inactive in all relevant records.

4.7 Lead Time Analysis of Warning Creation

Building on the chapter before which analyzed temporal adjustments in changed warnings, this section specifically examines the lead time of warnings that underwent modifications. The analysis focuses on the time difference between when these warnings were initially created or subsequently modified and their respective start times. The results provide insights into how much lead time GeoSphere typically gives before issuing these warnings, or, in some cases, whether they were created or altered after the predicted start time of the event. This is essential to understand the proactive versus reactive nature of the system in responding to evolving weather risks.

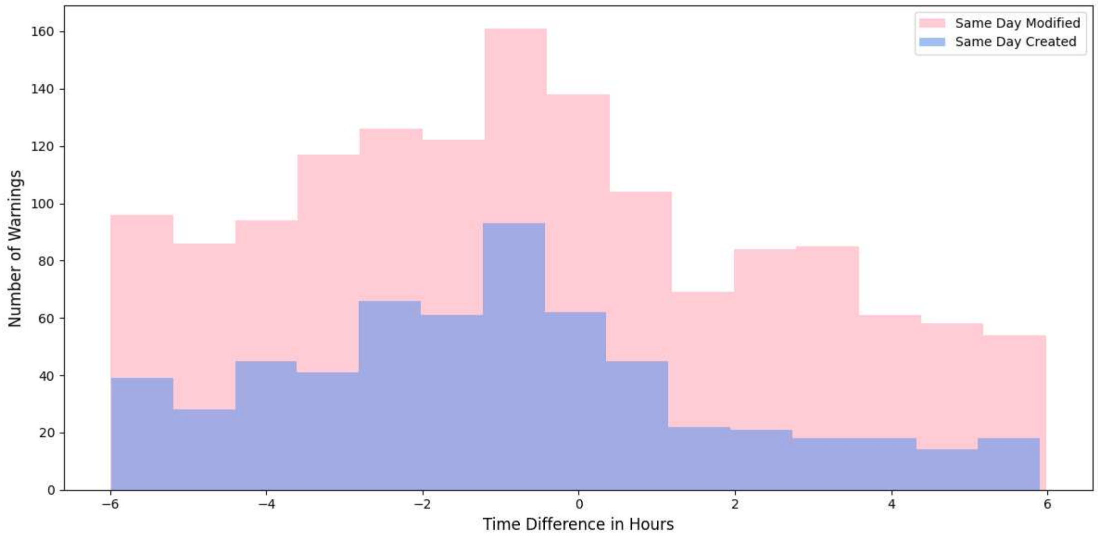
The histogram in Figure 2 shows the distribution of time differences for changed warnings created or modified on the same day as the predicted weather event. Negative values indicate warnings or modifications made after the event started, while positive values represent actions taken before the event's start time.



For same-day created warnings, represented by the blue bars, the number totaled 1,254 warnings, corresponding to 19.96% in total. The mean time difference was -1.15 hours, with a median of -1.25 hours. This indicates that the majority of warnings were issued close to the start of the event, often with a slight delay. The minimum time difference of -14.41 hours highlights cases where warnings were issued significantly after the event began, while the maximum lead time extended to 15.67 hours.

Same-day modified warnings, shown in pink, amounted to 2,031, which makes up 32.33% of the total warnings. These warnings refer to cases where a warning was issued and then modified on the same day of the predicted start time of the event. These modifications had an average time difference of -0.60 hours and a median of -0.97 hours. The minimum time difference of -18.61 hours shows substantial delays in some cases, while the maximum of 15.67 hours reflects proactive adjustments made in advance.

The histogram in Figure 2 reveals that a significant portion of warnings, particularly modifications, clustered around 0 hours, emphasizing real-time adjustments made during or shortly after the event began. The broader distribution of modified warnings indicates frequent updates in response to evolving conditions, compared to created warnings which show a slightly narrower spread.



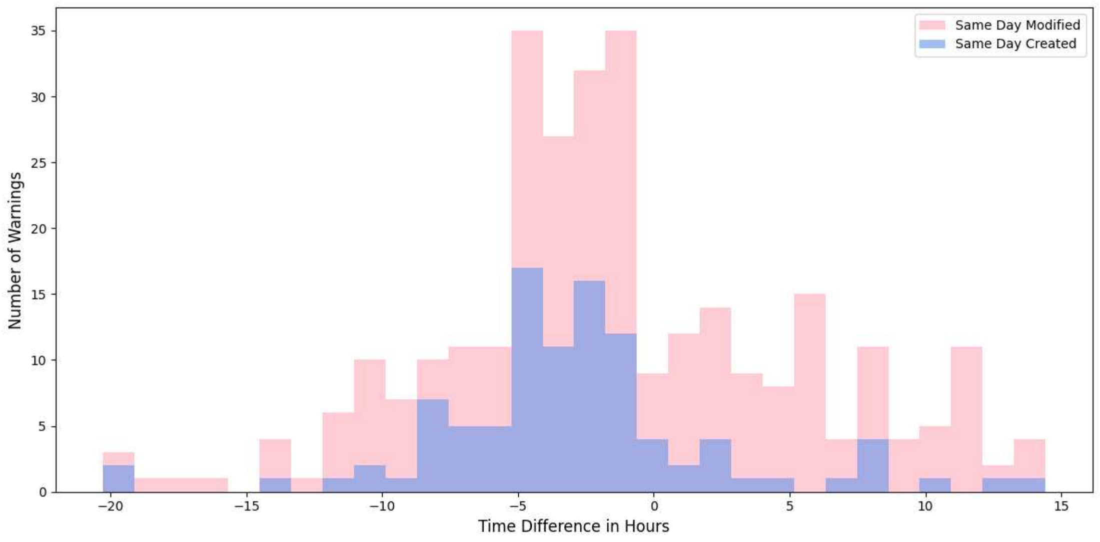
The histogram in Figure 3 illustrates the distribution of time differences for warnings that were created or modified within a 6-hour window relative to the start time of the weather event. The blue bars represent warnings created on the same day as the event, while the pink bars indicate warnings modified on the same day.

A total of 591 warnings, which makes up 9.41% of the total warnings, were created within the 6-hour window, which are scenarios where forecasts were issued either shortly before or after the event's onset. The clustering of blue bars around the 0 to -6 hour range indicates that many warnings were created in a reactive manner, often as events were developing or after their start.

Conversely, the pink bars demonstrate a significantly higher count of same-day modified warnings, with 1,455 cases, which correspond to 23.16% of the warnings, falling within the 6-hour window. This suggests that modifications are more frequent and often necessary as weather events unfold and more data becomes available.

Overall, the chart highlights the importance of real-time adjustments in forecasting and provides insights into GeoSphere's operational behavior in issuing and refining weather warnings within critical timeframes. While the system manages to provide lead times for many events, the reliance on last-minute changes suggests potential for improving initial predictions to reduce the need for reactive adjustments.

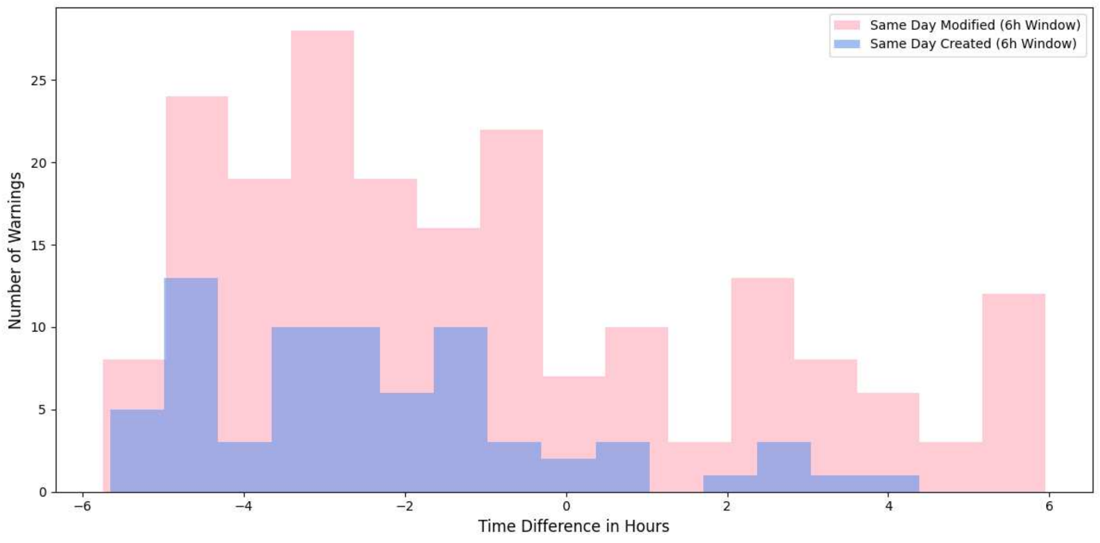
The histogram in Figure 4 illustrates the distribution of time differences for cancelled warnings that were created or modified on the same day as the predicted weather event. Negative values represent warnings or modifications made after the event began, while positive values indicate actions taken in advance of the event's start time.



For same-day created warnings, represented by the blue bars, the number totaled 270 warnings, which corresponds to 4.30% of the total warnings. The mean time difference was -2.92 hours, with a median of -3.11 hours. This demonstrates that the majority of these warnings were issued shortly after the event had already started. The minimum time difference of -20.29 hours highlights instances where warnings were issued long after the event had begun, while the maximum lead time reached 14.42 hours.

Same-day modified warnings are shown in pink and amounted to 303, which makes up for 4.82% of the total warnings. These modifications had an average time difference of -1.31 hours and a median of -2.13 hours, indicating a tendency for adjustments to occur close to or shortly after the event's start. The minimum time difference of -20.29 hours reveals substantial delays in a few cases, while the maximum of 14.42 hours reflects proactive modifications made well in advance.

The histogram in Figure 4 shows a noticeable clustering of warnings, especially modifications, around 0 hours, that again emphasize real-time adjustments that are made during or immediately after the event started. The broader distribution for modifications compared to created warnings suggests that frequent updates were made to refine the warnings.



The histogram in Figure 5 visualizes the distribution of time differences for cancelled warnings that were created or modified within a 6-hour window around the event's start time. The blue bars represent warnings created within the window, while the pink bars denote modified warnings.

A total of 71 warnings, which makes up 1.13% of the total warnings, were created within this timeframe, which indicates instances where forecasts were issued in close proximity to the event's start. These cases often represent last-minute forecasts or adjustments made as new data became available. The distribution of created warnings is relatively narrow, clustering around the -6 to 0-hour range, which highlights a tendency for these warnings to be issued and later cancelled reactively. On the other hand, 198 modified warnings, which corresponds to 3.15%, were identified in the same window, showing that adjustments for cancelled warnings were more frequent than new issuances.

This comparison demonstrates the importance of flexibility in weather warning systems, as a significant number of cancellations rely on modifications made shortly before or after an event's onset. It highlights GeoSphere's adaptive approach in handling cancelled warnings, where real-time updates play a crucial role in maintaining the accuracy and relevance of warnings, but also show room for improvement.

4.8 Localized Assessment of Weather Warning Changes

The municipality-specific evolution vectors provide an overview of the changes made to weather warnings across 2,750 different municipalities in Austria. These vectors summarize key adjustments, such as transitions in warning levels, change counts, and the distribution of warning types. This enables a better analysis of how warnings evolved for each municipality and helps to identify trends in the reliability of GeoSphere's warning system on a local level.

A major advantage of these vectors lies in their ability to make data easily searchable for each municipality. For a specific location, such as the municipality 60101 (Graz), it is possible to review a summary of all warnings that underwent changes or cancellations and analyze the adjustments, as shown in Table 5 and 6. This allows insights into the consistency and accuracy of forecasts at the municipal level to help identify areas where the warning system was effective and areas that require refinement.

Municipalities with fewer changes in their warning vectors generally indicate more stable and precise forecasting, while locations with frequent modifications suggest higher levels of uncertainty or rapidly changing weather conditions. The inclusion of cancellation counts reflect GeoSphere's approach in refining its warnings for different areas. The warning type distributions across all municipalities help identify regions frequently impacted by specific weather events and offer input for targeted improvements in forecasting.

Municipality Number	**Changed Warnings Count**	**Warning Level Changes (Initial → Final: Count)**	**Warning Type Counts (Type: Count)**
60101	400	1 → 2: 43 2 → 1: 7 2 → 3: 1 3 → 2: 1	Storm: 54 Rain: 54 Snow: 21 Black ice: 19 Thunderstorm: 252

Municipality Number	**Cancelled Warnings Count**	**Warning Level Changes (Initial → Final: Count)**	**Warning Type Counts (Type: Count)**
60101	93	1 → 0: 82 2 → 0: 11	Storm: 15 Rain: 28 Snow: 9 Black ice: 3 Thunderstorm: 38

As illustrated in Tables 5 and 6, Graz experienced 400 changed warnings and 93 cancelled warnings. The vectors for Graz include statistics on warning level transitions, and warning type distributions. For example, in changed warnings, the most frequent level transition was from 1 to 2 with 43 times. Similarly, for cancelled warnings, the majority of transitions involved warnings being downgraded from 1 to 0, consistent with the discontinuation of weather risks.

This example demonstrates how the municipality-specific vectors enable targeted analysis of GeoSphere's warning system at a local level. By making changes visible at specific municipalities, these vectors provide a framework for evaluating the performance and reliability of weather warnings in various regions.

4.9 Warning-Specific Datasets for Analyzing Weather Warning Adjustments

The ability to analyze and improve weather warnings relies on the availability of structured and comprehensive datasets. To achieve this, new unified datasets were created, combining key attributes of changed and cancelled warnings. These datasets serve as a standardized foundation for evaluating the evolution of changed and cancelled warnings, identifying patterns, and optimizing forecasting processes. Consolidating different aspects of warnings into a single format enables for a better analysis and provides insights into the changes made by GeoSphere in their weather warnings.

The datasets integrate a range of information that captures different aspects of warning adjustments. Each warning is assigned a unique identifier, making it traceable throughout its lifecycle. Changes in the number of affected municipalities, transitions between different warning types, and shifts in warning levels are documented to provide insights into the geographic and risk-related adjustments made during the warning. Numerical adjustments, such as intensity or magnitude changes, are also included, along with indicators of modifications to the start and end times of warnings. These attributes allow researchers to assess the accuracy of initial forecasts and the rationale behind updates.

The datasets also highlight whether warnings were issued or modified on the same day as the predicted event. This information sheds light on the lead time available to affected communities and the extent to which real-time adjustments were necessary. Finally, the inclusion of a cancellation indicator enables an analysis of warnings that were ultimately withdrawn and offer insights into potential false positives.

By capturing a broad spectrum of attributes in a standardized format, the datasets support a wide range of analytical objectives. Researchers can identify patterns in how warnings are adjusted, examine the timing of updates, and evaluate the consistency of warnings across different municipalities. The data also provides opportunities to study the accuracy of initial predictions and the effectiveness of updates on the same day. These datasets are not only a foundation for understanding past warning behaviors, but also a tool for shaping the future prediction technique of GeoSphere's weather warning systems.

5 Discussion

The analysis of GeoSphere's weather warnings reveals key insights into their processes and patterns. First, the results highlight the overall stability of the warning system in terms of warning types and the rare modifications to them. This consistency suggests that GeoSphere's classification of weather events is robust and reliable from the outset. However, changes such as cancellations or upward adjustments in warning levels reveal interesting dynamics.

The most frequent adjustment is the cancellation of lower severity warnings, indicated by a warning level change from 1 to 0, and reflects a reassessment of risks as conditions become less severe than expected. This behavior balances the need to avoid over-alarming the public while maintaining credibility. In contrast, upward adjustments, such as changes from level 1 to 2, occur only half as often and indicate situations where initial forecasts underestimated the severity of the event.

The analysis of value changes further supports this duality of stability and responsiveness. While most warnings experience minimal adjustments, a considerable proportion shows increases or decreases in severity. Positive changes suggest that new data highlighted greater risks than initially predicted, while negative changes reflect either overestimated initial forecasts or cancellations.

The analysis of temporal adjustments highlights GeoSphere's strategy for maintaining relevance in its warnings. The higher frequency of end time changes compared to start time changes highlights the uncertainty in predicting the duration of weather events. This is because forecasting the end time is more challenging, as it is further in the future. The overlap of start and end time changes reveals situations where forecasts were thoroughly revised and shows GeoSphere's responsiveness to evolving conditions. This approach ensures that warnings remain up to date.

The lead time analysis reveals critical insights into the timing of warnings. Many warnings are issued just hours before the event's predicted start, with a significant portion of same-day warnings being created or modified in real time. This reactive approach, particularly for modifications, reflects the challenges of forecasting rapidly developing weather systems. Although GeoSphere demonstrates the ability to issue early alerts, the high frequency of last-minute changes highlights the need for continuous updates to ensure accuracy.

The municipality-specific vectors developed in this thesis provide insights into how warnings evolve across different regions. Municipalities with fewer changes benefit from more stable forecasting, while those with frequent modifications may experience higher uncertainty or rapidly changing conditions. This data can be instrumental in targeting areas for further refinement in the forecasting process.

Warning-specific vectors provide a detailed perspective on how individual warnings evolve over time. Warnings with minimal changes indicate a higher degree of forecasting accuracy and stability, while those with frequent adjustments highlight events with greater uncertainty or dynamic conditions. This helps identifying patterns in warning behaviors. By understanding the evolution of warnings, targeted improvements can be made to enhance the precision and responsiveness of warning systems.

The findings of this analysis offer valuable insights into GeoSphere's weather warning system and suggest several areas for improvement. However, the most crucial recommendation is to prioritize the communication of uncertainties in weather warnings. Enhancing the way probabilities and potential variations are communicated will help the public better understand the inherent unpredictability of weather events and the rationale behind warning updates.

One key finding is the stability in warning types, reflecting a reliable classification process. However, the system occasionally issues low-severity warnings, which can lead to unnecessary cancellations. Upward adjustments in warning levels indicate the system's adaptability, though they also suggest that the initial forecasts may sometimes underestimate the severity of events. The frequent updates to end times, compared to start times, highlight the challenge of predicting event duration. While real-time updates are crucial, increasing the lead time for warnings would provide more time for preparation.

Clear and consistent messaging about the reasons for warning updates is essential. By emphasizing the uncertainty inherent in weather predictions and highlighting the potential for changes in severity or timing, GeoSphere can help the public make more informed decisions. This transparent communication will not only manage expectations, but also reduce confusion, especially when warnings evolve in real-time. Ultimately, a clearer understanding of how forecasts change will foster greater trust in the system and improve public preparedness.

In conclusion, focusing on clear communication of uncertainties would be an impactful change GeoSphere could make. By consistently informing the public about the rationale behind warning updates, whether due to new data, changing conditions, or evolving forecasts, GeoSphere can strengthen its warning system. This approach will ensure that the public is better equipped to respond to weather events, to enhance confidence and effectiveness in weather preparedness.

6 Conclusion and Future Work

This study examined how weather warnings issued by GeoSphere Austria have evolved over time, specifically with respect to changes and cancellations. The study reveals an efficient system that is proficient at properly classifying weather phenomena, maintaining consistency in warning types, and responding to evolving conditions through timely updates. The system's strength lies in its ability to adapt to new information and refine initial warnings as more data becomes available. This prioritization of safety is crucial, although frequent adjustments can lead to public uncertainty regarding the severity and timing of events.

The results reveal a balance between stability and adaptability. While most warnings are relatively consistent throughout their lifecycle, the system also shows flexibility by modifying severity levels, values, affected municipalities, and start or end times when necessary. This adaptability is critical for responding to changing weather conditions, but requires clear communication to ensure public understanding and trust. Miscommunications or frequent changes in warnings can undermine the credibility of the system, even if the adjustments themselves are justified [14].

Future research could delve deeper into how the timing of updates impacts the effectiveness of warnings in reducing risks and aiding preparation. Examining whether earlier updates lead to better outcomes could provide insights into optimizing lead times for more proactive warnings. Furthermore, exploring the relationship between the frequency of updates and public trust could guide improvements in messaging strategies. Frequent updates may be necessary for accuracy but could also lead to confusion or skepticism if not communicated effectively [14].

In conclusion, GeoSphere's weather warning system is effective in delivering accurate and timely warnings. However, to maximize its impact, it is crucial to focus on the communication of uncertainties and adjustments. Transparent messaging will foster better public understanding, improve preparation, and build trust in the system. Ultimately, refining how uncertainties are conveyed will strengthen the overall effectiveness of the system in safeguarding the public.

References

[1] Deadly Flooding in Central Europe made twice as likely by Climate Change. Section: Climate.

[2] Kristie L. Ebi, Jennifer Vanos, Jane W. Baldwin, Jesse E. Bell, David M. Hondula, Nicole A. Errett, Katie Hayes, Colleen E. Reid, Shubhayu Saha, June Spector, and Peter Berry. Extreme Weather and Climate Change: Population Health and Health System Implications. 42(1):293–315, 2021.

[3] GeoSphere Austria. Jahresbericht 2023. `https://www.geosphere.at/de/ueber-uns/downloads`, 2023.

[4] GeoSphere Austria. GeoSphere. `https://www.geosphere.at/de`, 2024.

[5] GeoSphere Austria. Leistungsvereinbarung 2024-2026. `https://www.geosphere.at/de/ueber-uns/downloads`, 2024.

[6] Thomas Kox, Lars Gerhold, and Uwe Ulbrich. Perception and Use of Uncertainty in Severe Weather Warnings by Emergency Services in Germany. 158-159:292–301, 2015.

[7] Joy Losee and Susan Joslyn. The need to trust: How features of the forecasted weather influence forecast trust. 30.

[8] D.S. Mileti and J.H. Sorensen. Communication of Emergency Public Warnings: A Social Science Perspective and State-of-the-Art Assessment.

[9] Jillian R. Olson, Claire M. Doyle, Daphne S. LaDue, and Alex N. Marmo. End-User Threat Perception: Building Confidence to Make Decisions Ahead of Severe Weather. pages 95–109.

[10] Sonia I. et al. Seneviratne. Managing the Risks of Extreme Events and Disasters to Advance Climate Change Adaptation. pages 109–230. Cambridge University Press, 1 edition.

[11] Oesterreichischer Staedtebund. Allgemein - Oesterreichischer Staedtebund. `https://www.staedtebund.gv.at/services/faq/allgemein`.

[12] Peter Stott. How Climate Change affects Extreme Weather Events. 352(6293):1517–1518, 2016. Publisher: American Association for the Advancement of Science.

[13] NOAA US Department of Commerce. Watch Warning Advisory Explained. Publisher: NOAA's National Weather Service.

[14] Qinghong Zhang, Liye Li, Beth Ebert, Brian Golding, David Johnston, Brian Mills, Shannon Panchuk, Sally Potter, Michael Riemer, Juanzhen Sun, Andrea Taylor, Sarah Jones, Paolo Ruti, and Julia Keller. Increasing the Value of Weather-Related Warnings. 64(10):647–649.

Appendix

This analysis examines changes in warning messages, focusing on start and end times, affected municipalities, warning levels, and other relevant attributes. The goal is to identify patterns and gain a better understanding of how these warnings evolve and are modified over time. Both changed and cancelled warnings are considered in the study.

[ ]:
```
# Importing necessary libraries
import pandas as pd  # Pandas for data manipulation
import ast  # Abstract Syntax Tree for handling Python expressions
import matplotlib.pyplot as plt  # Matplotlib for plotting graphs
import numpy as np  # Numpy for numerical operations
import dataframe_image as dfi  # Library for exporting DataFrames
as images
from collections import defaultdict  # Default dictionary for
handling default values
from matplotlib_venn import venn2  # Venn diagram for 2 sets

# Defining the file path for the warnings data
file_path = "data/warnings.csv"

# Reading the CSV file into a pandas DataFrame
warnings = pd.read_csv(file_path)
```

[ ]:
```
# Counting unique warnings by 'Warning-ID' and 'Sequence-ID'
unique_warnings = warnings[['warnid', 'verlaufid']].dropna().
 ↪drop_duplicates()
num_unique_warnings = unique_warnings.shape[0]

print(f"Number of unique warnings: {num_unique_warnings}")
```

In this step, we clean and structure the warning dataset to prepare it for further analysis. This includes renaming columns for clarity, handling missing values, and converting data types. We also identify and separate active and cancelled warnings, ensuring that we can analyze changes over time. Finally, we filter out duplicate or redundant entries and create two structured datasets: one for changed warnings and one for cancelled warnings.

[ ]:
```
# Renaming the columns in the warnings DataFrame to match
meaningful names
```

```
warnings.columns = ['Warning-ID', 'Change-ID', 'Sequence-ID',
'Status', 'Start Time', 'End Time', 'Created',
                    'File Created', 'Municipalities', 'Warning
Type', 'Value', 'Warning Level']

# Filling missing values in specific columns with default values
warnings['Sequence-ID'] = warnings['Sequence-ID'].fillna(0)
warnings['Municipalities'] = warnings['Municipalities'].
 ↪fillna('[]')
warnings['Warning Type'] = warnings['Warning Type'].fillna(0)
warnings['Value'] = warnings['Value'].fillna(0)
warnings['Warning Level'] = warnings['Warning Level'].fillna(0)
warnings['Start Time'] = warnings['Start Time'].fillna(0)
warnings['End Time'] = warnings['End Time'].fillna(0)
warnings['Created'] = warnings['Created'].fillna(0)

# Converting columns to appropriate data types (integers for ID/
 ↪level columns, datetime for time-related columns)
warnings = warnings.astype({
    'Sequence-ID': 'int',
    'Warning Type': 'int',
    'Value': 'int',
    'Warning Level': 'int',
    'Start Time': 'datetime64[ns, UTC]',
    'End Time': 'datetime64[ns, UTC]',
    'Created': 'datetime64[ns, UTC]',
    'File Created': 'datetime64[ns, UTC]',
})

# Identifying rows where the status is 'aufgehoben' (cancelled) by
grouping by 'Warning-ID' and getting the last row
last_rows = warnings.groupby('Warning-ID').last().reset_index()
aufgehoben_rows = last_rows[last_rows['Status'] == 'aufgehoben']

# For each 'Warning-ID', adding rows for the cancelled warnings
(where Sequence-ID > 0)
rows_to_add = []
for warn_id, group in warnings.groupby('Warning-ID'):
    aufgehoben_row = aufgehoben_rows[aufgehoben_rows['Warning-ID']
== warn_id]
    if not aufgehoben_row.empty:
        for _, row in group.iterrows():
            if row['Sequence-ID'] > 0:
                new_row = aufgehoben_row.iloc[0].copy()
                new_row['Sequence-ID'] = row['Sequence-ID']
                rows_to_add.append(new_row)

# Concatenating the new rows with the original DataFrame and
sorting them by 'Warning-ID' and 'Sequence-ID'
warnings_with_aufgehoben = pd.concat([warnings, pd.
 ↪DataFrame(rows_to_add)], ignore_index = True)
warnings_with_aufgehoben =
warnings_with_aufgehoben[warnings_with_aufgehoben['Sequence-ID'] >
0]
warnings_with_aufgehoben = warnings_with_aufgehoben.sort_values(by
= ['Warning-ID', 'Sequence-ID']).reset_index(drop = True)

# Filtering out warnings that have more than one entry for the
same Warning-ID and Sequence-ID (combined warnings)
group_counts = warnings_with_aufgehoben.groupby(['Warning-ID',
'Sequence-ID']).size().reset_index(name = 'Count')
multiple_entries = group_counts[group_counts['Count'] > 1]

# Merging warnings with multiple entries to find all combined
warnings
all_combined_warnings = warnings_with_aufgehoben.
 ↪merge(multiple_entries[['Warning-ID', 'Sequence-ID']],
                                                       on =
['Warning-ID', 'Sequence-ID'],
                                                       how =
'inner')

# Sorting the combined warnings and identifying the first and last
occurrences of each Warning-ID and Sequence-ID combination
all_combined_warnings = all_combined_warnings.sort_values(by =
['Warning-ID', 'Sequence-ID']).reset_index(drop = True)
```

```
first_warnings = all_combined_warnings.groupby(['Warning-ID',
'Sequence-ID']).first().reset_index()
last_warnings = all_combined_warnings.groupby(['Warning-ID',
'Sequence-ID']).last().reset_index()

# Concatenating the first and last warnings and sorting them by
'Warning-ID' and 'Sequence-ID'
combined_warnings = pd.concat([first_warnings, last_warnings],
ignore_index = True)
combined_warnings = combined_warnings.
 ↪sort_values(by=['Warning-ID', 'Sequence-ID']).reset_index(drop
= True)

# Extracting rows where the status is 'aufgehoben' (cancelled)
cancelled_rows = combined_warnings[combined_warnings['Status'] ==
'aufgehoben']

# Initializing a list to store the rows that need to be removed or
cancelled
all_cancelled_rows = []

# Adding the previous row if its status is 'aktiv' (active) when a
cancelled row is found
removed_indices = []
for index, row in cancelled_rows.iterrows():
    if index > 0:
        previous_row = combined_warnings.iloc[index - 1]
        if previous_row['Status'] == 'aktiv':
            all_cancelled_rows.append(previous_row)
            all_cancelled_rows.append(row)
            removed_indices.append(index - 1)
            removed_indices.append(index)

# Creating a final DataFrame for the cancelled warnings and for
the changed warnings after removal
changed_warnings_final = pd.DataFrame(combined_warnings)
cancelled_warnings_final = pd.DataFrame(all_cancelled_rows)
```

```
cancelled_warnings_final['Warning-Sequence'] =
cancelled_warnings_final['Warning-ID'].astype(str) + "_" +
cancelled_warnings_final['Sequence-ID'].astype(str)
changed_warnings_final['Warning-Sequence'] =
changed_warnings_final['Warning-ID'].astype(str) + "_" +
changed_warnings_final['Sequence-ID'].astype(str)

# Removing the cancelled warnings from the final changed warnings
DataFrame
changed_warnings_final =
changed_warnings_final[~changed_warnings_final['Warning-Sequence'].
 ↪isin(cancelled_warnings_final['Warning-Sequence'])]

# Resetting the index of the changed warnings DataFrame after
removal
changed_warnings_final = changed_warnings_final.reset_index(drop =
True)

# Ensuring that the cancelled warnings DataFrame has the first
1750 rows if necessary
cancelled_warnings_final = cancelled_warnings_final.iloc[:1750].
 ↪reset_index(drop = True)

# Displaying the final DataFrames
cancelled_warnings_final.head(10)
changed_warnings_final.head(10)
```

Before proceeding with further analysis, we check both the `changed_warnings_final` and `cancelled_warnings_final` datasets for missing values. This ensures data completeness and helps identify any gaps that may require further cleaning or imputation. The results will indicate which columns contain null values and how many entries are affected.

[ ]:
```
# Checking for missing (null) values in the final DataFrames:
'changed_warnings_final' and 'cancelled_warnings_final'
missing_values_in_changed = changed_warnings_final.isnull().sum()
# Counting missing values in 'changed_warnings_final'
missing_values_in_cancelled = cancelled_warnings_final.isnull().
 ↪sum()  # Counting missing values in 'cancelled_warnings_final'
```

```
# Printing the result to show the number of missing values for
each column in both DataFrames
print("Missing values in changed_warnings_final:")
print(missing_values_in_changed)  # Displaying missing values in
'changed_warnings_final'

print("\nMissing values in cancelled_warnings_final:")
print(missing_values_in_cancelled)  # Displaying missing values in
'cancelled_warnings_final'
```

[ ]:
```
cancelled_warnings_final.info()
changed_warnings_final.info()
print(len(cancelled_warnings_final))
```

In this step, we analyze changes in the affected municipalities for each warning. We compare municipality lists between consecutive warning entries to detect additions or removals. This helps in understanding how warnings evolve in terms of geographic coverage and whether certain areas are repeatedly affected or removed from alerts over time.

[ ]:
```
# Function to calculate municipality changes by comparing two sets
of municipalities
def municipality_changes(first, last):
    first_set = set(ast.literal_eval(first))
    last_set = set(ast.literal_eval(last))
    symmetric_difference = first_set.
 ↪symmetric_difference(last_set)
    return len(symmetric_difference)

# Creating a new column 'Municipality Changes' and calculating
changes between consecutive rows
changed_warnings_final['Municipality Changes'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_municipalities = changed_warnings_final.
 ↪iloc[i]['Municipalities']
        second_municipalities = changed_warnings_final.iloc[i +
1]['Municipalities']
```

```
        municipality_change =
municipality_changes(first_municipalities, second_municipalities)
        changed_warnings_final.at[i, 'Municipality Changes'] = 0
        changed_warnings_final.at[i + 1, 'Municipality Changes'] =
municipality_change

# Creating a new column 'Municipality Number Changes' and
comparing the number of municipalities between consecutive rows
changed_warnings_final['Municipality Number Changes'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_municipality_number = changed_warnings_final.
 ↪iloc[i]['Municipalities']
        second_municipality_number = changed_warnings_final.iloc[i
+ 1]['Municipalities']
        first_length = len(set(ast.
 ↪literal_eval(first_municipality_number)))
        second_length = len(set(ast.
 ↪literal_eval(second_municipality_number)))
        if first_length != second_length:
            changed_warnings_final.at[i, 'Municipality Number
Changes'] = 0
            changed_warnings_final.at[i + 1, 'Municipality Number
Changes'] = 1
        else:
            changed_warnings_final.at[i, 'Municipality Number
Changes'] = 0
            changed_warnings_final.at[i + 1, 'Municipality Number
Changes'] = 0

# Printing statistics about municipality rows and changes,
including percentage of changes and average number of changes
municipality_rows = len(changed_warnings_final['Municipalities'])
print("Number of Municipality Rows:", municipality_rows)
print()
```

```
municipality_changes =
changed_warnings_final[changed_warnings_final['Municipality
Changes'] != 0]
print("Number of Municipality Changes:",
len(municipality_changes))
print("Percentage of Municipality Changes:",
round(len(municipality_changes) / (municipality_rows/2) * 100, 2),
"%")

# Calculating and displaying the average number of municipality
changes
average_municipality_changes =
changed_warnings_final['Municipality Changes'].mean()
print("Average Municipality Changes: ",
round(average_municipality_changes, 2))
```

[ ]:
```
# Function to calculate municipality changes by comparing two sets
of municipalities.
def municipality_changes(first, last):
    first_set = set(ast.literal_eval(first))
    last_set = set(ast.literal_eval(last))
    symmetric_difference = first_set.
 ↪symmetric_difference(last_set)
    return len(symmetric_difference)

# Creating a new column 'Municipality Changes' in the cancelled
warnings DataFrame and calculating changes.
cancelled_warnings_final['Municipality Changes'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_municipalities = cancelled_warnings_final.
 ↪iloc[i]['Municipalities']
        second_municipalities = cancelled_warnings_final.iloc[i +
1]['Municipalities']
        municipality_change =
municipality_changes(first_municipalities, second_municipalities)
        cancelled_warnings_final.at[i, 'Municipality Changes'] = 0
        cancelled_warnings_final.at[i + 1, 'Municipality Changes']
= municipality_change

# Creating a new column 'Municipality Number Changes' and
comparing the number of municipalities between consecutive rows.
cancelled_warnings_final['Municipality Number Changes'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_municipality_number = cancelled_warnings_final.
 ↪iloc[i]['Municipalities']
        second_municipality_number = cancelled_warnings_final.
 ↪iloc[i + 1]['Municipalities']
        first_length = len(set(ast.
 ↪literal_eval(first_municipality_number)))
        second_length = len(set(ast.
 ↪literal_eval(second_municipality_number)))
        if first_length != second_length:
            cancelled_warnings_final.at[i, 'Municipality Number
Changes'] = 0
            cancelled_warnings_final.at[i + 1, 'Municipality
Number Changes'] = 1
        else:
            cancelled_warnings_final.at[i, 'Municipality Number
Changes'] = 0
            cancelled_warnings_final.at[i + 1, 'Municipality
Number Changes'] = 0

# Printing statistics about municipality rows and changes,
including percentage of changes and average number of changes.
municipality_rows =
len(cancelled_warnings_final['Municipalities'])
print("Number of Municipality Rows:", municipality_rows)
print()

municipality_changes =
cancelled_warnings_final[cancelled_warnings_final['Municipality
Changes'] != 0]
```

```
print("Number of Municipality Changes:",
len(municipality_changes))
print("Percentage of Municipality Changes:",
round(len(municipality_changes) / (municipality_rows/2) * 100, 2),
"%")

# Calculating and displaying the average number of municipality
changes.
average_municipality_changes =
cancelled_warnings_final['Municipality Changes'].mean()
print("Average Municipality Changes: ",
round(average_municipality_changes, 2))
```

This step examines how the number of affected municipalities changes between consecutive warnings. By comparing the count of municipalities in each warning pair, we can identify whether the coverage area has increased, decreased, or remained the same. This helps in assessing the scale and impact of warning modifications.

[ ]:
```
## Municipality Number Comparison: Counting and displaying the
number of municipality number changes
# and their percentage in the dataset.
municipality_rows = len(changed_warnings_final['Municipalities'])
municipality_number_changes =
changed_warnings_final[changed_warnings_final['Municipality Number
Changes'] == 1]
print("Number of Municipality Number Changes:",
int(len(municipality_number_changes)))
print("Percentage of Municipality Number Changes:",
round(len(municipality_number_changes) / (municipality_rows/2) *
100, 2), "%")

# Calculating the difference in the number of municipalities
between consecutive rows and storing the result.
changed_warnings_final['Difference in Municipality Numbers'] = 0

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_municipalities = set(ast.
 ↪literal_eval(changed_warnings_final.iloc[i]['Municipalities']))
```

```
        second_municipalities = set(ast.
↪literal_eval(changed_warnings_final.iloc[i +
1]['Municipalities']))

        first_length = len(first_municipalities)
        second_length = len(second_municipalities)
        difference = second_length - first_length

        changed_warnings_final.at[i + 1, 'Difference in
Municipality Numbers'] = difference
        changed_warnings_final.at[i, 'Difference in Municipality
Numbers'] = 0

# Extracting the differences in municipality numbers from every
second row for further analysis.
differences = changed_warnings_final.iloc[1::2]['Difference in
Municipality Numbers']

# Calculating the average absolute difference in municipality
numbers.
absolute_differences = changed_warnings_final['Difference in
Municipality Numbers'].abs()
average_municipality_number_change = absolute_differences.mean()
print("Average Municipality Number Changes (absolute values): ",
round(average_municipality_number_change, 2))
print()

# Counting the number of increases, decreases, and cases where the
number of municipalities remained unchanged.
positive_changes = differences > 0
negative_changes = differences < 0
no_changes = differences == 0

print(f"Number of Increases: {positive_changes.sum()}")
print(f"Number of Decreases: {negative_changes.sum()}")
print(f"Number of No Changes: {no_changes.sum()}")
print()
```

```
# Visualizing the distribution of municipality number changes
using a histogram.
plt.figure(figsize = (8, 5))
bins = np.arange(-1000, 1001, 250)
plt.hist(differences, bins = bins, color = 'pink', alpha = 0.8,
edgecolor = 'white')
labels = [
    f"({bins[i]}) - ({bins[i + 1]})" if bins[i] < 0 and bins[i +
1] < 0
    else f"({bins[i]}) - {bins[i + 1]}" if bins[i] < 0
    else f"{bins[i]} - ({bins[i + 1]})" if bins[i + 1] < 0
    else f"{bins[i]} - {bins[i + 1]}"
    for i in range(len(bins) - 1)
]
plt.xticks(ticks = (bins[:-1] + bins[1:]) / 2, labels = labels,
rotation = 45)
plt.xlabel("Change in the Number of Affected Municipalities",
fontsize = 12)
plt.xlim(-1000, 1000)
plt.ylabel("Frequency", fontsize = 12)
plt.tight_layout()
plt.savefig('plots/
 ↪changed_municipality_number_changes_distribution.png')
plt.show()
```

[ ]:
```
warnings_with_minus_1000 =
changed_warnings_final[changed_warnings_final['Difference in
Municipality Numbers'] <= -1000]
warnings_with_minus_1000[['Warning-ID', 'Sequence-ID',
'Change-ID', 'Difference in Municipality Numbers', 'Warning Type',
'Warning Level']].head(5)
```

[ ]:
```
warning_1093 = warnings[warnings['Warning-ID'] == 1093]
warning_1093
```

[ ]:
```
# Municipality Number Comparison: Counting and displaying the
number of municipality number changes
# and their percentage in the cancelled warnings dataset.
municipality_rows =
len(cancelled_warnings_final['Municipalities'])
```

```
municipality_number_changes =
cancelled_warnings_final[cancelled_warnings_final['Municipality
Number Changes'] == 1]
print("Number of Municipality Number Changes:",
int(len(municipality_number_changes)))
print("Percentage of Municipality Number Changes:",
round(len(municipality_number_changes) / (municipality_rows/2) *
100, 2), "%")

# Calculating the difference in the number of municipalities
between consecutive rows and storing the result.
cancelled_warnings_final['Difference in Municipality Numbers'] = 0

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_municipalities = set(ast.
 ↪literal_eval(cancelled_warnings_final.
 ↪iloc[i]['Municipalities']))
        second_municipalities = set(ast.
 ↪literal_eval(cancelled_warnings_final.iloc[i +
1]['Municipalities']))

        first_length = len(first_municipalities)
        second_length = len(second_municipalities)
        difference = second_length - first_length

        cancelled_warnings_final.at[i + 1, 'Difference in
Municipality Numbers'] = difference
        cancelled_warnings_final.at[i, 'Difference in Municipality
Numbers'] = 0

# Extracting the differences in municipality numbers from every
second row for further analysis.
differences = cancelled_warnings_final.iloc[1::2]['Difference in
Municipality Numbers']

# Calculating the average absolute difference in municipality
numbers.
```

```
absolute_differences = cancelled_warnings_final['Difference in
Municipality Numbers'].abs()
average_municipality_number_change = absolute_differences.mean()
print("Average Municipality Number Changes (absolute values): ",
round(average_municipality_number_change, 2))
print()

# Counting the number of increases, decreases, and cases where the
number of municipalities remained unchanged.
positive_changes = differences > 0
negative_changes = differences < 0
no_changes = differences == 0

print(f"Number of Increases: {positive_changes.sum()}")
print(f"Number of Decreases: {negative_changes.sum()}")
print(f"Number of No Changes: {no_changes.sum()}")
print()

# Visualizing the distribution of municipality number changes
using a histogram.
plt.figure(figsize = (8, 5))
plt.hist(differences, bins = range(-1500, 1, 250), color = 'pink',
alpha = 0.8, edgecolor = 'white')
plt.xlabel("Change in the Number of Affected Municipalities",
fontsize = 12)
plt.xlim(-1500, 0)
plt.ylabel("Frequency", fontsize = 12)
bins = plt.hist(differences, bins = range(-1500, 1, 250), color =
'pink', edgecolor = 'white')[1]
labels = [
    f"({int(bins[i])}) - ({int(bins[i + 1])})" if bins[i] < 0 and
bins[i + 1] < 0
    else f"({int(bins[i])}) - {int(bins[i + 1])}" if bins[i] < 0
    else f"{int(bins[i])} - ({int(bins[i + 1])})" if bins[i + 1] <
0
    else f"{int(bins[i])} - {int(bins[i + 1])}"
    for i in range(len(bins) - 1)
]
```

```
plt.xticks(bins[:-1] + (bins[1] - bins[0]) / 2, labels, rotation =
45)
plt.tight_layout()
plt.savefig('plots/
 ↪cancelled_municipality_number_changes_distribution.png')
plt.show()
```

[ ]:
```
warnings_with_minus_1000 =
cancelled_warnings_final[cancelled_warnings_final['Difference in
Municipality Numbers'] <= -1000]
warnings_with_minus_1000[['Warning-ID', 'Sequence-ID',
'Change-ID', 'Difference in Municipality Numbers']].head(5)
```

[ ]:
```
warning_543_1 =
cancelled_warnings_final[(cancelled_warnings_final['Warning-ID']
== 543) & (cancelled_warnings_final['Sequence-ID'] == 1)]
warning_543_1[['Warning-ID', 'Sequence-ID', 'Change-ID', 'Warning
Type', 'Warning Level']].head(1)
```

[ ]:
```
warning_543_2 =
cancelled_warnings_final[(cancelled_warnings_final['Warning-ID']
== 543) & (cancelled_warnings_final['Sequence-ID'] == 2)]
warning_543_2[['Warning-ID', 'Sequence-ID', 'Change-ID', 'Warning
Type', 'Warning Level']].head(1)
```

[ ]:
```
warning_604 =
cancelled_warnings_final[(cancelled_warnings_final['Warning-ID']
== 604) & (cancelled_warnings_final['Sequence-ID'] == 1)]
warning_604[['Warning-ID', 'Sequence-ID', 'Change-ID', 'Warning
Type', 'Warning Level']].head(1)
```

[ ]:
```
warning_697 =
cancelled_warnings_final[(cancelled_warnings_final['Warning-ID']
== 697) & (cancelled_warnings_final['Sequence-ID'] == 1)]
warning_697[['Warning-ID', 'Sequence-ID', 'Change-ID', 'Warning
Type', 'Warning Level']].head(1)
```

[ ]:
```
warning_881 =
cancelled_warnings_final[(cancelled_warnings_final['Warning-ID']
== 881) & (cancelled_warnings_final['Sequence-ID'] == 1)]
```

```
warning_881[['Warning-ID', 'Sequence-ID', 'Change-ID', 'Warning
Type', 'Warning Level']].head(1)
```

Here, we analyze changes in the type of warnings issued. By comparing consecutive entries, we identify whether a warning has been modified to a different type, such as shifting from one hazard category to another. This helps in understanding how warnings evolve in response to new information or changing conditions.

```
[ ]: # Warning Type Changes: This block creates a new column to track
changes in warning types between consecutive rows.
changed_warnings_final['Warning Type Changes'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_warning_type = changed_warnings_final.
 ↪iloc[i]['Warning Type']
        second_warning_type = changed_warnings_final.iloc[i +
1]['Warning Type']
        if first_warning_type != second_warning_type:
            changed_warnings_final.at[i, 'Warning Type Changes'] =
0
            changed_warnings_final.at[i + 1, 'Warning Type
Changes'] = 1
        else:
            changed_warnings_final.at[i, 'Warning Type Changes'] =
0
            changed_warnings_final.at[i + 1, 'Warning Type
Changes'] = 0

# Warning Type Comparison: This section calculates and prints the
number and percentage of warning type changes.
warning_type_rows = len(changed_warnings_final['Warning Type'])
print("Number of Warning Type Rows:", warning_type_rows)

warning_type_changes =
changed_warnings_final[changed_warnings_final['Warning Type
Changes'] != 0]
print("Number of Warning Type Changes:",
len(warning_type_changes))
```

```
percentage_warning_type_changes = len(warning_type_changes) /
(warning_type_rows / 2) * 100
print("Percentage of Warning Type Changes:",
round(percentage_warning_type_changes, 2), "%")
print()

# Difference in Warning Type: This block records the specific
warning types that changed between consecutive rows.
changed_warnings_final['Difference in Warning Type'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_warning_type = changed_warnings_final.
 ↪iloc[i]['Warning Type']
        second_warning_type = changed_warnings_final.iloc[i +
1]['Warning Type']
        if first_warning_type != second_warning_type:
            changed_warnings_final.at[i, 'Difference in Warning
Type'] = first_warning_type
            changed_warnings_final.at[i + 1, 'Difference in
Warning Type'] = second_warning_type
        else:
            changed_warnings_final.at[i, 'Difference in Warning
Type'] = 0
            changed_warnings_final.at[i + 1, 'Difference in
Warning Type'] = 0

# Counting the different Changes: This section counts the
frequency of each warning type change pair.
pair_change_frequencies = {}

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_warning_type_change = changed_warnings_final.
 ↪iloc[i]['Difference in Warning Type']
        second_warning_type_change = changed_warnings_final.iloc[i
+ 1]['Difference in Warning Type']
        if first_warning_type_change !=
second_warning_type_change:
```

```
            pair = (first_warning_type_change,
second_warning_type_change)
            if pair in pair_change_frequencies:
                pair_change_frequencies[pair] += 1
            else:
                pair_change_frequencies[pair] = 1

# Creating a DataFrame to display the frequency of warning type
changes.
pair_change_frequencies_df = pd.DataFrame(
    list(pair_change_frequencies.items()),
    columns = ['Warning Type Change', 'Frequency']
).sort_values(by = 'Frequency', ascending = False)

print(pair_change_frequencies_df)
```

[ ]:
```
# Warning Type Changes: This section creates a new column to track
changes in warning types between consecutive rows.
cancelled_warnings_final['Warning Type Changes'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_warning_type = cancelled_warnings_final.
 ↪iloc[i]['Warning Type']
        second_warning_type = cancelled_warnings_final.iloc[i +
1]['Warning Type']
        if first_warning_type != second_warning_type:
            cancelled_warnings_final.at[i, 'Warning Type Changes']
= 0
            cancelled_warnings_final.at[i + 1, 'Warning Type
Changes'] = 1
        else:
            cancelled_warnings_final.at[i, 'Warning Type Changes']
= 0
            cancelled_warnings_final.at[i + 1, 'Warning Type
Changes'] = 0

# Warning Type Comparison: This part calculates and displays the
number and percentage of warning type changes.
```

```
warning_type_rows = len(cancelled_warnings_final['Warning Type'])
print("Number of Warning Type Rows:", warning_type_rows)

warning_type_changes =
cancelled_warnings_final[cancelled_warnings_final['Warning Type
Changes'] != 0]
print("Number of Warning Type Changes:",
len(warning_type_changes))

percentage_warning_type_changes = len(warning_type_changes) /
(warning_type_rows / 2) * 100
print("Percentage of Warning Type Changes:",
round(percentage_warning_type_changes, 2), "%")
print()

# Difference in Warning Type: This section stores the specific
warning types that changed between consecutive rows.
cancelled_warnings_final['Difference in Warning Type'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_warning_type = cancelled_warnings_final.
 ↪iloc[i]['Warning Type']
        second_warning_type = cancelled_warnings_final.iloc[i +
1]['Warning Type']
        if first_warning_type != second_warning_type:
            cancelled_warnings_final.at[i, 'Difference in Warning
Type'] = first_warning_type
            cancelled_warnings_final.at[i + 1, 'Difference in
Warning Type'] = second_warning_type
        else:
            cancelled_warnings_final.at[i, 'Difference in Warning
Type'] = 0
            cancelled_warnings_final.at[i + 1, 'Difference in
Warning Type'] = 0

# Counting the different Changes: This section counts the
frequency of each warning type change pair.
pair_change_frequencies = {}
```

```
for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_warning_type_change = cancelled_warnings_final.
 ↪iloc[i]['Difference in Warning Type']
        second_warning_type_change = cancelled_warnings_final.
 ↪iloc[i + 1]['Difference in Warning Type']
        if first_warning_type_change !=
second_warning_type_change:
            pair = (first_warning_type_change,
second_warning_type_change)
            if pair in pair_change_frequencies:
                pair_change_frequencies[pair] += 1
            else:
                pair_change_frequencies[pair] = 1

# Creating a DataFrame to display the frequency of warning type
changes.
pair_change_frequencies_df = pd.DataFrame(
    list(pair_change_frequencies.items()),
    columns = ['Warning Type Change', 'Frequency']
).sort_values(by = 'Frequency', ascending = False)

print(pair_change_frequencies_df)
```

This step examines how the severity of warnings changes over time. By comparing the warning levels in consecutive entries, we can detect whether a warning has been upgraded, downgraded, or remained the same. This analysis provides insights into how risk assessments evolve as situations develop.

[ ]:
```
# Warning Level Changes: This section creates a new column to
track changes in warning levels between consecutive rows.
changed_warnings_final['Warning Level Changes'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_warning_level = changed_warnings_final.
 ↪iloc[i]['Warning Level']
        second_warning_level = changed_warnings_final.iloc[i +
1]['Warning Level']
```

```
        if first_warning_level != second_warning_level:
            changed_warnings_final.at[i, 'Warning Level Changes']
= 0
            changed_warnings_final.at[i + 1, 'Warning Level
Changes'] = 1
        else:
            changed_warnings_final.at[i, 'Warning Level Changes']
= 0
            changed_warnings_final.at[i + 1, 'Warning Level
Changes'] = 0

# Warning Level Comparison: This part calculates and displays the
number and percentage of warning level changes.
warning_level_rows = len(changed_warnings_final['Warning Level'])
print("Number of Warning Level Rows:", warning_level_rows)

warning_level_changes =
changed_warnings_final[changed_warnings_final['Warning Level
Changes'] != 0]
print("Number of Warning Level Changes:",
len(warning_level_changes))

percentage_warning_level_changes = len(warning_level_changes) /
(warning_level_rows / 2) * 100
print("Percentage of Warning Level Changes:",
round(percentage_warning_level_changes, 2), "%")
print()

# Difference in Warning Level: This section stores the specific
warning levels that changed between consecutive rows.
changed_warnings_final['Difference in Warning Level'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_warning_level = changed_warnings_final.
 ↪iloc[i]['Warning Level']
        second_warning_level = changed_warnings_final.iloc[i +
1]['Warning Level']
        if first_warning_level != second_warning_level:
            changed_warnings_final.at[i, 'Difference in Warning
Level'] = first_warning_level
            changed_warnings_final.at[i + 1, 'Difference in
Warning Level'] = second_warning_level
        else:
            changed_warnings_final.at[i, 'Difference in Warning
Level'] = 0
            changed_warnings_final.at[i + 1, 'Difference in
Warning Level'] = 0

# Counting the different Changes: This section counts the
frequency of each warning level change pair.
pair_change_frequencies = {}

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_warning_level_change = changed_warnings_final.
 ↪iloc[i]['Difference in Warning Level']
        second_warning_level_change = changed_warnings_final.
 ↪iloc[i + 1]['Difference in Warning Level']
        if first_warning_level_change !=
second_warning_level_change:
            pair = (first_warning_level_change,
second_warning_level_change)
            if pair in pair_change_frequencies:
                pair_change_frequencies[pair] += 1
            else:
                pair_change_frequencies[pair] = 1

# Creating a DataFrame to display the frequency of warning level
changes.
pair_change_frequencies_df = pd.DataFrame(
    list(pair_change_frequencies.items()),
    columns = ['Warning Level Change', 'Frequency']
).sort_values(by = 'Frequency', ascending = False)

print(pair_change_frequencies_df)
```

```
[ ]: # Warning Level Changes: This section creates a new column to
track changes in warning levels between consecutive rows.
cancelled_warnings_final['Warning Level Changes'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_warning_level = cancelled_warnings_final.
 ↪iloc[i]['Warning Level']
        second_warning_level = cancelled_warnings_final.iloc[i +
1]['Warning Level']
        if first_warning_level != second_warning_level:
            cancelled_warnings_final.at[i, 'Warning Level
Changes'] = 0
            cancelled_warnings_final.at[i + 1, 'Warning Level
Changes'] = 1
        else:
            cancelled_warnings_final.at[i, 'Warning Level
Changes'] = 0
            cancelled_warnings_final.at[i + 1, 'Warning Level
Changes'] = 0

# Warning Level Comparison: This part calculates and displays the
number and percentage of warning level changes.
warning_level_rows = len(cancelled_warnings_final['Warning
Level'])
print("Number of Warning Level Rows:", warning_level_rows)

warning_level_changes =
cancelled_warnings_final[cancelled_warnings_final['Warning Level
Changes'] != 0]
print("Number of Warning Level Changes:",
len(warning_level_changes))

percentage_warning_level_changes = len(warning_level_changes) /
(warning_level_rows / 2) * 100
print("Percentage of Warning Level Changes:",
round(percentage_warning_level_changes, 2), "%")
print()
```

```
# Difference in Warning Level: This section stores the specific
warning levels that changed between consecutive rows.
cancelled_warnings_final['Difference in Warning Level'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_warning_level = cancelled_warnings_final.
 ↪iloc[i]['Warning Level']
        second_warning_level = cancelled_warnings_final.iloc[i +
1]['Warning Level']
        if first_warning_level != second_warning_level:
            cancelled_warnings_final.at[i, 'Difference in Warning
Level'] = first_warning_level
            cancelled_warnings_final.at[i + 1, 'Difference in
Warning Level'] = second_warning_level
        else:
            cancelled_warnings_final.at[i, 'Difference in Warning
Level'] = 0
            cancelled_warnings_final.at[i + 1, 'Difference in
Warning Level'] = 0

# Counting the different Changes: This section counts the
frequency of each warning level change pair.
pair_change_frequencies = {}

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_warning_level_change = cancelled_warnings_final.
 ↪iloc[i]['Difference in Warning Level']
        second_warning_level_change = cancelled_warnings_final.
 ↪iloc[i + 1]['Difference in Warning Level']
        if first_warning_level_change !=
second_warning_level_change:
            pair = (first_warning_level_change,
second_warning_level_change)
            if pair in pair_change_frequencies:
                pair_change_frequencies[pair] += 1
            else:
                pair_change_frequencies[pair] = 1
```

```
# Creating a DataFrame to display the frequency of warning level
changes.
pair_change_frequencies_df = pd.DataFrame(
    list(pair_change_frequencies.items()),
    columns = ['Warning Level Change', 'Frequency']
).sort_values(by = 'Frequency', ascending = False)

print(pair_change_frequencies_df)
```

This analysis focuses on changes in the `Value` field of warnings, which could represent measurements like temperature, wind speed, or water levels. By comparing values in consecutive warnings, we can track how key warning parameters fluctuate over time and assess the severity of these changes.

```
[ ]: # Value Changes: This section creates a new column to track
changes in the 'Value' column between consecutive rows.
changed_warnings_final['Value Changes'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_value = changed_warnings_final.iloc[i]['Value']
        second_value = changed_warnings_final.iloc[i + 1]['Value']
        if first_value != second_value:
            changed_warnings_final.at[i, 'Value Changes'] = 0
            changed_warnings_final.at[i + 1, 'Value Changes'] = 1
        else:
            changed_warnings_final.at[i, 'Value Changes'] = 0
            changed_warnings_final.at[i + 1, 'Value Changes'] = 0

# Value Comparison: This part calculates and displays the number
and percentage of value changes.
value_rows = len(changed_warnings_final['Value'])
print("Number of Value Rows:", value_rows)

value_changes =
changed_warnings_final[changed_warnings_final['Value Changes'] !=
0]
print("Number of Value Changes:", (len(value_changes)))
print("Percentage of Value Changes:", round((len(value_changes) /
((value_rows / 2)) * 100), 2), "%")

# Value Difference Changes: This section calculates the numerical
difference in values between consecutive rows.
changed_warnings_final['Difference in Value Numbers'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_value = changed_warnings_final.iloc[i]['Value']
        second_value = changed_warnings_final.iloc[i + 1]['Value']
        difference = second_value - first_value

        changed_warnings_final.at[i + 1, 'Difference in Value
Numbers'] = difference
        changed_warnings_final.at[i, 'Difference in Value
Numbers'] = 0  # Set first row of each pair to 0

# Extract every second row (differences only) for analysis.
differences_values = changed_warnings_final.iloc[1::2]['Difference
in Value Numbers']

# Calculating the absolute average change in values.
absolute_differences_values = differences_values.abs()
average_value_number_change = absolute_differences_values.mean()
print("Average Value Number Changes (absolute values): ",
round(average_value_number_change, 2))
```

[ ]:
```
# Extract every second row (differences only) for analysis,
including the warning type
differences_values_with_type = changed_warnings_final.iloc[1::
 ↪2][['Warning Type', 'Difference in Value Numbers']]

# Calculate the average difference per warning type
average_changes_per_type = differences_values_with_type.
 ↪groupby('Warning Type', as_index=False)['Difference in Value
Numbers'].mean()

# Display the average change for each warning type
```

```
print("Average Value Change per Warning Type:")
print(average_changes_per_type)

# Calculate the absolute average change per warning type
absolute_average_changes_per_type = differences_values_with_type.
 ↪groupby('Warning Type', as_index=False)['Difference in Value
Numbers'].apply(lambda x: x.abs().mean())

# Display the absolute average change for each warning type
print("Absolute Average Value Change per Warning Type:")
print(absolute_average_changes_per_type)
```

[ ]:
```
# Add a new column for percentage change in value
changed_warnings_final['Percentage Value Change'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_value = changed_warnings_final.iloc[i]['Value']
        second_value = changed_warnings_final.iloc[i + 1]['Value']

        if first_value != 0:  # Avoid division by zero
            percentage_change = (second_value / first_value) * 100
        else:
            percentage_change = None  # Set to None if first value
is zero to avoid errors

        changed_warnings_final.at[i + 1, 'Percentage Value
Change'] = percentage_change
        changed_warnings_final.at[i, 'Percentage Value Change'] =
0  # Set first row of each pair to 0

# Extract every second row (percentage changes only) for analysis.
percentage_changes = changed_warnings_final.iloc[1::2]['Percentage
Value Change']

# Calculate the average percentage change (excluding None values)
average_percentage_change = percentage_changes.dropna().mean()
print("Average Percentage Value Change: ",
round(average_percentage_change, 2), "%")
```

[ ]:
```
# Value Changes: This section creates a new column to track
changes in the ’Value’ column between consecutive rows.
cancelled_warnings_final[’Value Changes’] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_value = cancelled_warnings_final.iloc[i][’Value’]
        second_value = cancelled_warnings_final.iloc[i +
1][’Value’]
        if first_value != second_value:
            cancelled_warnings_final.at[i, ’Value Changes’] = 0
            cancelled_warnings_final.at[i + 1, ’Value Changes’] =
1
        else:
            cancelled_warnings_final.at[i, ’Value Changes’] = 0
            cancelled_warnings_final.at[i + 1, ’Value Changes’] =
0

# Value Comparison: This part calculates and displays the number
and percentage of value changes.
value_rows = len(cancelled_warnings_final[’Value’])
print("Number of Value Rows:", value_rows)

value_changes =
cancelled_warnings_final[cancelled_warnings_final[’Value Changes’]
!= 0]
print("Number of Value Changes:", (len(value_changes)))
print("Percentage of Value Changes:", round((len(value_changes) /
((value_rows / 2)) * 100), 2), "%")

# Value Difference Changes: This section calculates the numerical
difference in values between consecutive rows.
cancelled_warnings_final[’Difference in Value Numbers’] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_value = cancelled_warnings_final.iloc[i][’Value’]
        second_value = cancelled_warnings_final.iloc[i +
1][’Value’]
```

```
        difference = second_value - first_value

        cancelled_warnings_final.at[i + 1, 'Difference in Value
Numbers'] = difference
        cancelled_warnings_final.at[i, 'Difference in Value
Numbers'] = 0  # Set first row of each pair to 0

# Extract every second row (differences only) for analysis.
differences_values = cancelled_warnings_final.iloc[1::
 ↪2]['Difference in Value Numbers']

# Calculating the absolute average change in values.
absolute_differences_values = differences_values.abs()
average_value_number_change = absolute_differences_values.mean()
print("Average Value Number Changes (absolute values): ",
round(average_value_number_change, 2))
```

[ ]:
```
# Extract every second row (differences only) for analysis,
including the warning type
differences_values_with_type = cancelled_warnings_final.iloc[1::
 ↪2].copy()  # Create a copy for modification

# Assign the 'Warning Type' from the previous row (i.e., the first
row of the pair)
differences_values_with_type['Warning Type'] =
cancelled_warnings_final.iloc[::2]['Warning Type'].values

# Calculate the average difference per warning type
average_changes_per_type = differences_values_with_type.
 ↪groupby('Warning Type')['Difference in Value Numbers'].mean()

# Calculate the absolute average change per warning type
absolute_average_changes_per_type = differences_values_with_type.
 ↪groupby('Warning Type')['Difference in Value Numbers'].
 ↪apply(lambda x: x.abs().mean())

# Display the results with Warning Type
print("Average Value Change per Warning Type:")
for warning_type, value in average_changes_per_type.items():
    print(f"Warning Type {warning_type}: {value:.2f}")

print("\nAbsolute Average Value Change per Warning Type:")
for warning_type, value in absolute_average_changes_per_type.
↪items():
    print(f"Warning Type {warning_type}: {value:.2f}")
```

This analysis examines modifications in the start and end times of warnings. By comparing timestamps between consecutive entries, we can identify delays, extensions, or rescheduling of warnings. Understanding these time-based changes helps in assessing how warnings are adjusted in response to evolving situations.

```
[ ]: # Changes in Start Time: This section tracks changes in the 'Start
Time' column between consecutive rows.
changed_warnings_final['Start Time Changes'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_start_time = changed_warnings_final.iloc[i]['Start
Time']
        second_start_time = changed_warnings_final.iloc[i +
1]['Start Time']
        if first_start_time != second_start_time:
            changed_warnings_final.at[i, 'Start Time Changes'] =
first_start_time
            changed_warnings_final.at[i + 1, 'Start Time Changes']
= second_start_time
        else:
            changed_warnings_final.at[i, 'Start Time Changes'] = 1
            changed_warnings_final.at[i + 1, 'Start Time Changes']
= 1

# Changes in End Time: This section tracks changes in the 'End
Time' column between consecutive rows.
changed_warnings_final['End Time Changes'] = None

for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_end_time = changed_warnings_final.iloc[i]['End
Time']
```

```
        second_end_time = changed_warnings_final.iloc[i + 1]['End
Time']
        if first_end_time != second_end_time:
            changed_warnings_final.at[i, 'End Time Changes'] =
first_end_time
            changed_warnings_final.at[i + 1, 'End Time Changes'] =
second_end_time
        else:
            changed_warnings_final.at[i, 'End Time Changes'] = 1
            changed_warnings_final.at[i + 1, 'End Time Changes'] =
1

# Count rows where 'Start Time Changes' is not equal to 1
start_time_changes_count = int((changed_warnings_final['Start Time
Changes'] != 1).sum()/2)

# Count rows where 'End Time Changes' is not equal to 1
end_time_changes_count = int((changed_warnings_final['End Time
Changes'] != 1).sum()/2)

print(f"Number of rows with changes in Start Time:
{start_time_changes_count}")
print(f"Number of rows with changes in End Time:
{end_time_changes_count}")
```

[ ]:
```
# Changes in Start Time: This section tracks changes in the 'Start
Time' column between consecutive rows.
cancelled_warnings_final['Start Time Changes'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_start_time = cancelled_warnings_final.iloc[i]['Start
Time']
        second_start_time = cancelled_warnings_final.iloc[i +
1]['Start Time']
        if first_start_time != second_start_time:
            cancelled_warnings_final.at[i, 'Start Time Changes'] =
first_start_time
```

```
            cancelled_warnings_final.at[i + 1, 'Start Time
Changes'] = second_start_time
        else:
            cancelled_warnings_final.at[i, 'Start Time Changes'] =
1
            cancelled_warnings_final.at[i + 1, 'Start Time
Changes'] = 1

# Changes in End Time: This section tracks changes in the 'End
Time' column between consecutive rows.
cancelled_warnings_final['End Time Changes'] = None

for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_end_time = cancelled_warnings_final.iloc[i]['End
Time']
        second_end_time = cancelled_warnings_final.iloc[i +
1]['End Time']
        if first_end_time != second_end_time:
            cancelled_warnings_final.at[i, 'End Time Changes'] =
first_end_time
            cancelled_warnings_final.at[i + 1, 'End Time Changes']
= second_end_time
        else:
            cancelled_warnings_final.at[i, 'End Time Changes'] = 1
            cancelled_warnings_final.at[i + 1, 'End Time Changes']
= 1

# Count rows where 'Start Time Changes' is not equal to 1
start_time_changes_count = int((cancelled_warnings_final['Start
Time Changes'] != 1).sum()/2)

# Count rows where 'End Time Changes' is not equal to 1
end_time_changes_count = int((cancelled_warnings_final['End Time
Changes'] != 1).sum()/2)

print(f"Number of rows with changes in Start Time:
{start_time_changes_count}")
```

```
print(f"Number of rows with changes in End Time:
{end_time_changes_count}")
```

This step analyzes whether the `Start Time` and `End Time` of warnings were modified together or independently. By comparing changes in these timestamps across warning pairs, we can determine if adjustments are typically made simultaneously or if only one time component is altered. This provides insights into how warnings are updated over time.

[ ]:
```
# Start Time Comparison: This section counts and displays the
number of start time changes in the dataset.
start_time_rows =
changed_warnings_final[changed_warnings_final['Start Time'].
 ↪notnull()]
print("Number of Start Time Rows:", len(start_time_rows))

# Filtering and counting rows where start time has changed.
start_time_changes =
changed_warnings_final[changed_warnings_final['Start Time
Changes'].apply(lambda x: isinstance(x, pd.Timestamp))]
print("Number of Start Time Changes:", int(len(start_time_changes)
/ 2))
print()

# End Time Comparison: This section counts and displays the number
of end time changes in the dataset.
end_time_rows = changed_warnings_final[changed_warnings_final['End
Time'].notnull()]
print("Number of End Time Rows:", len(end_time_rows))

# Filtering and counting rows where end time has changed.
end_time_changes =
changed_warnings_final[changed_warnings_final['End Time Changes'].
 ↪apply(lambda x: isinstance(x, pd.Timestamp))]
print("Number of End Time Changes:", int(len(end_time_changes) /
2))
print()

# Initializing counters and lists to categorize time changes.
same_time_changes = 0
```

```
no_time_changes = 0
different_time_changes = 0

same_change_rows = []
no_change_rows = []
different_change_rows = []

# Iterating through the dataset in pairs to classify start and end
time changes.
for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        start_time_change_1 = changed_warnings_final.
 ↪iloc[i]['Start Time Changes']
        start_time_change_2 = changed_warnings_final.iloc[i +
1]['Start Time Changes']
        end_time_change_1 = changed_warnings_final.iloc[i]['End
Time Changes']
        end_time_change_2 = changed_warnings_final.iloc[i +
1]['End Time Changes']

        # Categorizing rows where both start and end times have
changed.
        if (isinstance(start_time_change_1, pd.Timestamp) and
isinstance(start_time_change_2, pd.Timestamp) and
           isinstance(end_time_change_1, pd.Timestamp) and
isinstance(end_time_change_2, pd.Timestamp)):
            same_time_changes += 1
            same_change_rows.append((i, i + 1))

        # Categorizing rows where no changes occurred.
        elif (start_time_change_1 == 1 and start_time_change_2 ==
1 and end_time_change_1 == 1 and end_time_change_2 == 1):
            no_time_changes += 1
            no_change_rows.append((i, i + 1))

        # Categorizing rows where changes are inconsistent (one
row changed, the other didn't).
        else:
            different_time_changes += 1
```

```
                different_change_rows.append((i, i + 1))

# Displaying counts of categorized changes.
print(f"Number of same changes (both datetime in both rows):
{same_time_changes}")
print(f"Number of no changes (all 1s): {no_time_changes}")
print(f"Number of different changes (one datetime, one 1 or null
in rows): {different_time_changes}")
```

```
[ ]: # Start Time Comparison: This section counts and displays the
number of start time rows and changes.
start_time_rows =
cancelled_warnings_final[cancelled_warnings_final['Start Time'].
 ↪notnull()]
print("Number of Start Time Rows:", len(start_time_rows))

start_time_changes =
cancelled_warnings_final[cancelled_warnings_final['Start Time
Changes'].apply(lambda x: isinstance(x, pd.Timestamp))]
print("Number of Start Time Changes:", int(len(start_time_changes)
/ 2))
print()

# End Time Comparison: This section counts and displays the number
of end time rows and changes.
end_time_rows =
cancelled_warnings_final[cancelled_warnings_final['End Time'].
 ↪notnull()]
print("Number of End Time Rows:", len(end_time_rows))

end_time_changes =
cancelled_warnings_final[cancelled_warnings_final['End Time
Changes'].apply(lambda x: isinstance(x, pd.Timestamp))]
print("Number of End Time Changes:", int(len(end_time_changes) /
2))
print()

# Initializing counters for time changes
same_time_changes = 0
```

```
no_time_changes = 0
different_time_changes = 0

# Lists to store row indices of changes for potential debugging or
further analysis
same_change_rows = []
no_change_rows = []
different_change_rows = []

# Iterating through cancelled warnings in pairs to categorize
changes in start and end times.
for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        start_time_change_1 = cancelled_warnings_final.
 ↪iloc[i]['Start Time Changes']
        start_time_change_2 = cancelled_warnings_final.iloc[i +
1]['Start Time Changes']
        end_time_change_1 = cancelled_warnings_final.iloc[i]['End
Time Changes']
        end_time_change_2 = cancelled_warnings_final.iloc[i +
1]['End Time Changes']

        if (isinstance(start_time_change_1, pd.Timestamp) and
isinstance(start_time_change_2, pd.Timestamp) and
            isinstance(end_time_change_1, pd.Timestamp) and
isinstance(end_time_change_2, pd.Timestamp)):
            same_time_changes += 1
            same_change_rows.append((i, i + 1))
        elif (start_time_change_1 == 1 and start_time_change_2 ==
1 and end_time_change_1 == 1 and end_time_change_2 == 1):
            no_time_changes += 1
            no_change_rows.append((i, i + 1))
        else:
            different_time_changes += 1
            different_change_rows.append((i, i + 1))

# Displaying results of time change analysis.
print(f"Number of same changes (both datetime in both rows):
{same_time_changes}")
```

```
print(f"Number of no changes (all 0): {no_time_changes}")
print(f"Number of different changes (one datetime, one 0 or null
in rows): {different_time_changes}")
```

This step examines whether warnings were created or modified on the same day as their start time. By analyzing these patterns, we can assess how quickly warnings are issued and updated in response to evolving conditions. This helps in understanding the timeliness of warning systems and their adaptability to real-time events.

```
[ ]: # Analyze the first rows of the pairs (Created Warnings):
# This section checks whether the 'File Created' date is the same
as the 'Start Time' date for the first row in each warning pair.
first_rows = changed_warnings_final.iloc[::2].copy()
first_rows['Created Same Day as Start'] = first_rows.apply(
    lambda row: row['File Created'].date() == row['Start Time'].
 ↪date(), axis = 1
)

# Analyze the second rows of the pairs (Modified Warnings):
# This section checks if the 'Created Same Day as Start' flag for
the first row is carried over to the second row.
second_rows = changed_warnings_final.iloc[1::2].copy()
second_rows['Created Same Day as Start'] = second_rows.apply(
    lambda row: first_rows.loc[row.name - 1, 'Created Same Day as
Start'] if row.name - 1 in first_rows.index else False,
    axis = 1
)

# Check whether the second row (modified warnings) was modified on
the same day as 'Start Time'.
second_rows['Modified Same Day as Start'] = second_rows.apply(
    lambda row: (row['File Created'].date() == first_rows.loc[row.
 ↪name - 1, 'Start Time'].date())
    if row['Status'] == 'aufgehoben'
    else row['File Created'].date() == row['Start Time'].date(),
    axis = 1
)

# Initialize new columns with False to avoid missing values.
```

```
changed_warnings_final['Created Same Day as Start'] = False
changed_warnings_final['Modified Same Day as Start'] = False

# Assign values only to the second row of each warning pair.
changed_warnings_final.loc[second_rows.index, 'Created Same Day as
Start'] = second_rows['Created Same Day as Start']
changed_warnings_final.loc[second_rows.index, 'Modified Same Day
as Start'] = second_rows['Modified Same Day as Start']

# Convert to boolean type for consistency.
changed_warnings_final['Created Same Day as Start'] =
changed_warnings_final['Created Same Day as Start'].astype(bool)
changed_warnings_final['Modified Same Day as Start'] =
changed_warnings_final['Modified Same Day as Start'].astype(bool)

# Count the number of warnings where 'Created Same Day as Start'
and 'Modified Same Day as Start' are True (only considering second
rows).
same_day_created_count =
changed_warnings_final[(changed_warnings_final.index % 2 == 1) &
(changed_warnings_final['Created Same Day as Start'] == True)].
 ↪shape[0]
same_day_modified_count =
changed_warnings_final[(changed_warnings_final.index % 2 == 1) &
(changed_warnings_final['Modified Same Day as Start'] == True)].
 ↪shape[0]

# Print results.
print(f"Number of Same Day Created Warnings:
{same_day_created_count}")
print(f"Number of Same Day Modified Warnings:
{same_day_modified_count}")

# Display the first 10 rows of the modified DataFrame for
verification.
#changed_warnings_final.head(10)
```

```
[ ]: # Define categories (fixing duplicates): This section defines the
categories for the visualization.
categories = ['Created Same Day as Start', 'Not Created Same Day
as Start',
              'Modified Same Day as Start', 'Not Modified Same Day
as Start']

# Compute values directly from `changed_warnings_final`: This
calculates the number of warnings for each category.
values = [
    changed_warnings_final[(changed_warnings_final.index % 2 == 1)
& (changed_warnings_final['Created Same Day as Start'] == True)].
 ↪shape[0],
    changed_warnings_final[(changed_warnings_final.index % 2 == 1)
& (changed_warnings_final['Created Same Day as Start'] == False)].
 ↪shape[0],
    changed_warnings_final[(changed_warnings_final.index % 2 == 1)
& (changed_warnings_final['Modified Same Day as Start'] == True)].
 ↪shape[0],
    changed_warnings_final[(changed_warnings_final.index % 2 == 1)
& (changed_warnings_final['Modified Same Day as Start'] ==
False)].shape[0]
]

print(values)

# Create bar chart to visualize the number of warnings created and
modified on the same or different day.
plt.figure(figsize = (10, 6))
plt.bar(categories, values, color = ['skyblue', 'lightpink',
'lightgreen', 'orange'])

# Add titles and labels to the chart.
plt.title("Warnings Created and Modified on the Same or Different
Day", fontsize = 14)
plt.xlabel("Categories", fontsize = 12)
plt.ylabel("Number of Warnings", fontsize = 12)

# Save and show the chart.
plt.tight_layout()
plt.savefig('plots/
↪changed_warnings_same_day_created_modified_warnings.png')
plt.show()
```

[ ]:
```
# Extract warnings that were created on the same day as the start
time.
same_day_created_warnings =
changed_warnings_final[(changed_warnings_final.index % 2 == 1) &
(changed_warnings_final['Created Same Day as Start'] == True)].
↪copy()

# Calculate the time difference in hours for same-day created
warnings.
same_day_created_warnings.loc[:, 'Time Difference in Hours'] =
same_day_created_warnings.apply(
    lambda row: (row['Start Time'] - row['File Created']).
↪total_seconds() / 3600
    if row['Start Time'].date() == row['File Created'].date()
    else None, axis = 1
)

# Extract warnings that were modified on the same day as the start
time.
same_day_modified_warnings =
changed_warnings_final[(changed_warnings_final.index % 2 == 1) &
(changed_warnings_final['Modified Same Day as Start'] == True)].
↪copy()

# Calculate the time difference in hours for same-day modified
warnings.
same_day_modified_warnings.loc[:, 'Time Difference in Hours'] =
same_day_modified_warnings.apply(
    lambda row: (row['Start Time'] - row['File Created']).
↪total_seconds() / 3600
    if row['Start Time'].date() == row['File Created'].date()
    else None, axis = 1
)
```

```python
# Compute descriptive statistics for time differences in same-day
created warnings.
created_mean = same_day_created_warnings['Time Difference in
Hours'].mean()
created_median = same_day_created_warnings['Time Difference in
Hours'].median()
created_min = same_day_created_warnings['Time Difference in
Hours'].min()
created_max = same_day_created_warnings['Time Difference in
Hours'].max()

# Compute descriptive statistics for time differences in same-day
modified warnings.
modified_mean = same_day_modified_warnings['Time Difference in
Hours'].mean()
modified_median = same_day_modified_warnings['Time Difference in
Hours'].median()
modified_min = same_day_modified_warnings['Time Difference in
Hours'].min()
modified_max = same_day_modified_warnings['Time Difference in
Hours'].max()

# Print statistics for same-day created warnings.
print("Time Difference of Same Day Created Warnings in Hours:")
print(f"Mean: {created_mean:.2f}, Median: {created_median:.2f},
Min: {created_min:.2f}, Max: {created_max:.2f}\n")

# Print statistics for same-day modified warnings.
print("Time Difference of Same Day Modified Warnings in Hours:")
print(f"Mean: {modified_mean:.2f}, Median: {modified_median:.2f},
Min: {modified_min:.2f}, Max: {modified_max:.2f}\n")

# Visualization: Histogram showing the distribution of time
differences for same-day created and modified warnings.
plt.figure(figsize = (12, 6))
plt.hist(same_day_modified_warnings['Time Difference in Hours'],
bins = 30, alpha = 0.8, label = 'Same Day Modified', color =
'pink')
```

```
plt.hist(same_day_created_warnings['Time Difference in Hours'],
bins = 30, alpha = 0.6, label = 'Same Day Created', color =
'cornflowerblue')

# Add titles and labels.
plt.xlabel("Time Difference in Hours", fontsize = 12)
plt.ylabel("Number of Warnings", fontsize = 12)
plt.legend()

# Save and display the plot.
plt.tight_layout()
plt.savefig('plots/
 ↪changed_time_difference_created_modified_warnings.png')
plt.show()
```

```
[ ]: # Define a time window (6 hours in seconds)
time_window_seconds = 6 * 3600

# Filter same-day created warnings within the 6-hour window
same_day_created_in_window = same_day_created_warnings[
    same_day_created_warnings['Time Difference in Hours'].apply(
        lambda x: -6 <= x <= 6 if x is not None else False
    )
]

# Filter same-day modified warnings within the 6-hour window
same_day_modified_in_window = same_day_modified_warnings[
    same_day_modified_warnings['Time Difference in Hours'].apply(
        lambda x: -6 <= x <= 6 if x is not None else False
    )
]

# Count the number of warnings in the 6-hour window
created_count_in_window = same_day_created_in_window.shape[0]
modified_count_in_window = same_day_modified_in_window.shape[0]

# Print counts and display filtered data
print(f"Number of Same Day Created Warnings in 6-hour window:
{created_count_in_window}")
```

```
print(f"Number of Same Day Modified Warnings in 6-hour window:
{modified_count_in_window}")

# Visualization: Histogram for 6-hour window
plt.figure(figsize = (12, 6))
plt.hist(
    same_day_modified_in_window['Time Difference in Hours'],
    bins = 15,
    alpha = 0.8,
    label = 'Same Day Modified',
    color='pink'
)
plt.hist(
    same_day_created_in_window['Time Difference in Hours'],
    bins = 15,
    alpha = 0.6,
    label = 'Same Day Created',
    color = 'cornflowerblue'
)

# Add labels
plt.xlabel("Time Difference in Hours", fontsize = 12)
plt.ylabel("Number of Warnings", fontsize = 12)
plt.legend()

# Save and display the plot
plt.tight_layout()
plt.savefig('plots/changed_same_day_created_modified_6h_window.
 ↪png')
plt.show()
```

```
[ ]: # Analyze the first rows of the pairs (Created Warnings):
# This section checks whether the 'File Created' date is the same
as the 'Start Time' date for the first row in each warning pair.
first_rows = cancelled_warnings_final.iloc[::2].copy()
first_rows['Created Same Day as Start'] = first_rows.apply(
    lambda row: row['File Created'].date() == row['Start Time'].
 ↪date(), axis = 1
)
```

```
# Analyze the second rows of the pairs (Modified Warnings):
# This section checks if the 'Created Same Day as Start' flag for
the first row is carried over to the second row.
second_rows = cancelled_warnings_final.iloc[1::2].copy()
second_rows['Created Same Day as Start'] = second_rows.apply(
    lambda row: first_rows.loc[row.name - 1, 'Created Same Day as
Start'] if row.name - 1 in first_rows.index else False,
    axis = 1
)

# Check whether the second row (modified warnings) was modified on
the same day as 'Start Time'.
second_rows['Modified Same Day as Start'] = second_rows.apply(
    lambda row: (row['File Created'].date() == first_rows.loc[row.
↪name - 1, 'Start Time'].date())
    if row['Status'] == 'aufgehoben'
    else row['File Created'].date() == row['Start Time'].date(),
    axis = 1
)

# Initialize new columns with False to avoid missing values.
cancelled_warnings_final['Created Same Day as Start'] = False
cancelled_warnings_final['Modified Same Day as Start'] = False

# Assign values only to the second row of each warning pair.
cancelled_warnings_final.loc[second_rows.index, 'Created Same Day
as Start'] = second_rows['Created Same Day as Start']
cancelled_warnings_final.loc[second_rows.index, 'Modified Same Day
as Start'] = second_rows['Modified Same Day as Start']

# Convert to boolean type for consistency.
cancelled_warnings_final['Created Same Day as Start'] =
cancelled_warnings_final['Created Same Day as Start'].astype(bool)
cancelled_warnings_final['Modified Same Day as Start'] =
cancelled_warnings_final['Modified Same Day as Start'].
↪astype(bool)
```

```
# Count the number of warnings where 'Created Same Day as Start'
and 'Modified Same Day as Start' are True (only considering second
rows).
same_day_created_count =
cancelled_warnings_final[(cancelled_warnings_final.index % 2 == 1)
& (cancelled_warnings_final['Created Same Day as Start'] ==
True)].shape[0]
same_day_modified_count =
cancelled_warnings_final[(cancelled_warnings_final.index % 2 == 1)
& (cancelled_warnings_final['Modified Same Day as Start'] ==
True)].shape[0]

# Print results.
print(f"Number of Same Day Created Warnings:
{same_day_created_count}")
print(f"Number of Same Day Modified Warnings:
{same_day_modified_count}")

# Display the first 10 rows of the modified DataFrame for
verification.
#cancelled_warnings_final.head(10)
```

```
[ ]: # Define categories (fixing duplicates): This section defines the
categories for the visualization.
categories = ['Created Same Day as Start', 'Not Created Same Day
as Start',
              'Modified Same Day as Start', 'Not Modified Same Day
as Start']

# Compute values directly from `cancelled_warnings_final`: This
calculates the number of warnings for each category.
values = [
    cancelled_warnings_final[(cancelled_warnings_final.index % 2
== 1) & (cancelled_warnings_final['Created Same Day as Start'] ==
True)].shape[0],
    cancelled_warnings_final[(cancelled_warnings_final.index % 2
== 1) & (cancelled_warnings_final['Created Same Day as Start'] ==
False)].shape[0],
    cancelled_warnings_final[(cancelled_warnings_final.index % 2
== 1) & (cancelled_warnings_final['Modified Same Day as Start'] ==
True)].shape[0],
    cancelled_warnings_final[(cancelled_warnings_final.index % 2
== 1) & (cancelled_warnings_final['Modified Same Day as Start'] ==
False)].shape[0]
]

print(values)

# Create bar chart to visualize the number of warnings created and
modified on the same or different day.
plt.figure(figsize = (10, 6))
plt.bar(categories, values, color = ['skyblue', 'lightpink',
'lightgreen', 'orange'])

# Add titles and labels to the chart.
plt.title("Warnings Created and Modified on the Same or Different
Day", fontsize = 14)
plt.xlabel("Categories", fontsize = 12)
plt.ylabel("Number of Warnings", fontsize = 12)

# Save and show the chart.
plt.tight_layout()
plt.savefig('plots/
 ↪cancelled_warnings_same_day_created_modified_warnings.png')
plt.show()
```

[ ]:
```
# Iterate through all row pairs (first and second row) to compute
the time difference for same-day created warnings.
for i in range(0, len(cancelled_warnings_final), 2):
    # Check if the second row satisfies the 'Created Same Day as
Start' condition
    if cancelled_warnings_final.iloc[i + 1]['Created Same Day as
Start'] == True:
        # Calculate the time difference between 'Start Time' and
'File Created' for both rows
        cancelled_warnings_final.at[i, 'Created Time Difference in
Hours'] = (
            (cancelled_warnings_final.at[i, 'Start Time'] -
cancelled_warnings_final.at[i + 1, 'File Created']).
 ↪total_seconds() / 3600
            if cancelled_warnings_final.at[i, 'Start Time'].date()
== cancelled_warnings_final.at[i + 1, 'File Created'].date()
            else None
        )

        cancelled_warnings_final.at[i + 1, 'Created Time
Difference in Hours'] = (
            (cancelled_warnings_final.at[i + 1, 'Start Time'] -
cancelled_warnings_final.at[i, 'File Created']).total_seconds() /
3600
            if cancelled_warnings_final.at[i + 1, 'Start Time'].
 ↪date() == cancelled_warnings_final.at[i, 'File Created'].date()
            else None
        )

# Compute statistics for 'Same Day Created Warnings'
created_mean = cancelled_warnings_final['Created Time Difference
in Hours'].mean()
created_median = cancelled_warnings_final['Created Time Difference
in Hours'].median()
created_min = cancelled_warnings_final['Created Time Difference in
Hours'].min()
created_max = cancelled_warnings_final['Created Time Difference in
Hours'].max()

# Print summary statistics for same-day created warnings.
print("Time Difference of Same Day Created Warnings in Hours:")
print(f"Mean: {created_mean:.2f}, Median: {created_median:.2f},
Min: {created_min:.2f}, Max: {created_max:.2f}\n")

# Compute the time difference for same-day modified warnings.
for i in range(0, len(cancelled_warnings_final), 2):
    # Check if the second row satisfies the 'Modified Same Day as
Start' condition
    if cancelled_warnings_final.iloc[i + 1]['Modified Same Day as
Start'] == True:
        # Calculate the time difference between 'Start Time' and
'File Created' for modified warnings
        cancelled_warnings_final.at[i, 'Modified Time Difference
in Hours'] = (
            (cancelled_warnings_final.at[i, 'Start Time'] -
cancelled_warnings_final.at[i + 1, 'File Created']).
↪total_seconds() / 3600
            if cancelled_warnings_final.at[i, 'Start Time'].date()
== cancelled_warnings_final.at[i + 1, 'File Created'].date()
            else None
        )

        cancelled_warnings_final.at[i + 1, 'Modified Time
Difference in Hours'] = (
            (cancelled_warnings_final.at[i + 1, 'Start Time'] -
cancelled_warnings_final.at[i, 'File Created']).total_seconds() /
3600
            if cancelled_warnings_final.at[i + 1, 'Start Time'].
↪date() == cancelled_warnings_final.at[i, 'File Created'].date()
            else None
        )

# Compute statistics for 'Same Day Modified Warnings'
modified_mean = cancelled_warnings_final['Modified Time Difference
in Hours'].mean()
modified_median = cancelled_warnings_final['Modified Time
Difference in Hours'].median()
modified_min = cancelled_warnings_final['Modified Time Difference
in Hours'].min()
modified_max = cancelled_warnings_final['Modified Time Difference
in Hours'].max()

# Print summary statistics for same-day modified warnings.
print("Time Difference of Same Day Modified Warnings in Hours:")
print(f"Mean: {modified_mean:.2f}, Median: {modified_median:.2f},
Min: {modified_min:.2f}, Max: {modified_max:.2f}\n")

# Visualization: Histogram of time differences for same-day
created and modified warnings.
```

```
plt.figure(figsize = (12, 6))
plt.hist(cancelled_warnings_final['Modified Time Difference in
Hours'], bins = 30, alpha = 0.8, label = 'Same Day Modified',
color = 'pink')
plt.hist(cancelled_warnings_final['Created Time Difference in
Hours'], bins = 30, alpha = 0.6, label = 'Same Day Created', color
= 'cornflowerblue')

# Add titles and labels to the chart.
plt.xlabel("Time Difference in Hours", fontsize = 12)
plt.ylabel("Number of Warnings", fontsize = 12)
plt.legend()

# Save and show the histogram.
plt.tight_layout()
plt.savefig('plots/
 ↪cancelled_time_difference_created_modified_warnings.png')
plt.show()
```

```
[ ]: # Define a time window (6 hours in seconds)
time_window_seconds = 6 * 3600

# Filter same-day created warnings within the 6-hour window
same_day_created_in_window = cancelled_warnings_final[
    (cancelled_warnings_final['Created Time Difference in Hours']
>= -6) &
    (cancelled_warnings_final['Created Time Difference in Hours']
<= 6)
].copy()

# Filter same-day modified warnings within the 6-hour window
same_day_modified_in_window = cancelled_warnings_final[
    (cancelled_warnings_final['Modified Time Difference in Hours']
>= -6) &
    (cancelled_warnings_final['Modified Time Difference in Hours']
<= 6)
].copy()

# Count the number of warnings in the 6-hour window
```

```
created_count_in_window = same_day_created_in_window.shape[0]
modified_count_in_window = same_day_modified_in_window.shape[0]

# Print counts and display filtered data
print(f"Number of Same Day Created Warnings in 6-hour window:
{created_count_in_window}")
print(f"Number of Same Day Modified Warnings in 6-hour window:
{modified_count_in_window}")

# Visualization: Histogram showing the distribution of time
differences for same-day created and modified warnings within the
6-hour window.
plt.figure(figsize = (12, 6))
plt.hist(
    same_day_modified_in_window['Modified Time Difference in
Hours'],
    bins = 15,
    alpha = 0.8,
    label = 'Same Day Modified (6h Window)',
    color = 'pink'
)
plt.hist(
    same_day_created_in_window['Created Time Difference in
Hours'],
    bins = 15,
    alpha = 0.6,
    label = 'Same Day Created (6h Window)',
    color = 'cornflowerblue'
)

# Add labels
plt.xlabel("Time Difference in Hours", fontsize = 12)
plt.ylabel("Number of Warnings", fontsize = 12)
plt.legend()

# Save and display the plot
plt.tight_layout()
plt.savefig('plots/cancelled_same_day_created_modified_6h_window.
 ↪png')
```

```
plt.show()
```

In this step, we generate vectors for each municipality to summarize how warnings have changed over time in different locations. These vectors capture information such as warning level changes, value fluctuations, and frequency of modifications. This approach allows for a municipality-level analysis of warning trends and risk patterns.

```
[ ]: # Initializing a dictionary to store various changes at the
municipality level.
municipality_changes = defaultdict(lambda: {
    'Warning Level Changes Count': defaultdict(int),  # Dictionary
to count specific warning level changes
    'Cancelled Warnings': 0,  # Count of cancelled warnings per
municipality
    'Warning Type Counts': defaultdict(int),  # Dictionary to
count warning types
    'Changed Warnings Count': 0  # Count of total changed warnings
per municipality
})

# Iterating through changed warnings in pairs to track changes in
warning levels, values, and types.
for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        first_row = changed_warnings_final.iloc[i]
        second_row = changed_warnings_final.iloc[i + 1]

        # Extract relevant fields from both rows in the pair.
        warning_level_start = first_row['Warning Level']
        warning_level_end = second_row['Warning Level']
        diff_value_numbers = second_row['Difference in Value
Numbers']
        warning_type = first_row['Warning Type']

        # Processing each municipality in pairs.
        first_row_municipalities = ast.
 ↪literal_eval(first_row['Municipalities'])
        for municipality in first_row_municipalities:
            municipality_changes[municipality]['Changed Warnings
Count'] += 1  # Increment warning count

            if warning_level_start != warning_level_end:
                warning_level_change = (warning_level_start,
warning_level_end)
                municipality_changes[municipality]['Warning Level
Changes Count'][warning_level_change] += 1

            municipality_changes[municipality]['Warning Type
Counts'][warning_type] += 1

# Converting the dictionary into a DataFrame, sorting the
municipalities.
municipality_vectors_df = pd.DataFrame([
    {
        'Municipality': municipality,
        'Changed Warnings Count': changes['Changed Warnings
Count'],  # Include total warning count
        'Warning Level Changes Count':
dict(sorted(changes['Warning Level Changes Count'].items())),
        'Warning Type Counts': dict(sorted(changes['Warning Type
Counts'].items()))
    }
    for municipality, changes in sorted(municipality_changes.
 ↪items())
])

# Ensuring the 'Municipality' column is treated as an integer for
consistency.
municipality_vectors_df['Municipality'] =
municipality_vectors_df['Municipality'].astype(int)

# Displaying the first few rows of the DataFrame.
#municipality_vectors_df.head()

# Filter the DataFrame for the municipality Graz (60101)
```

```
graz_data =
municipality_vectors_df[municipality_vectors_df['Municipality'] ==
60101]
graz_data
```

[ ]:
```
# Initializing a dictionary to store various changes at the
municipality level.
municipality_changes = defaultdict(lambda: {
    'Warning Level Changes Count': defaultdict(int),  # Dictionary
to count specific warning level changes
    'Cancelled Warnings': 0,  # Count of cancelled warnings per
municipality
    'Warning Type Counts': defaultdict(int),  # Dictionary to
count warning types
    'Cancelled Warnings Count': 0  # Count of total changed
warnings per municipality
})

# Iterating through cancelled warnings in pairs to track changes
in warning levels, values, and types.
for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        first_row = cancelled_warnings_final.iloc[i]
        second_row = cancelled_warnings_final.iloc[i + 1]

        # Extract relevant fields from both rows in the pair.
        warning_level_start = first_row['Warning Level']
        warning_level_end = second_row['Warning Level']
        diff_value_numbers = second_row['Difference in Value
Numbers']
        warning_type = first_row['Warning Type']

        # Processing each municipality in pairs.
        first_row_municipalities = ast.
 ↪literal_eval(first_row['Municipalities'])
        for municipality in first_row_municipalities:
            municipality_changes[municipality]['Cancelled Warnings
Count'] += 1  # Increment warning count
```

```
            if warning_level_start != warning_level_end:
                warning_level_change = (warning_level_start,
warning_level_end)
                municipality_changes[municipality]['Warning Level
Changes Count'][warning_level_change] += 1

            municipality_changes[municipality]['Warning Type
Counts'][warning_type] += 1

# Converting the dictionary into a DataFrame, sorting the
municipalities.
municipality_vectors_df = pd.DataFrame([
    {
        'Municipality': municipality,
        'Cancelled Warnings Count': changes['Cancelled Warnings
Count'],  # Include total warning count
        'Warning Level Changes Count':
dict(sorted(changes['Warning Level Changes Count'].items())),
        'Warning Type Counts': dict(sorted(changes['Warning Type
Counts'].items()))
    }
    for municipality, changes in sorted(municipality_changes.
 ↪items())
])

# Ensuring the 'Municipality' column is treated as an integer for
consistency.
municipality_vectors_df['Municipality'] =
municipality_vectors_df['Municipality'].astype(int)

# Displaying the first few rows of the DataFrame.
municipality_vectors_df.head()

# Filter the DataFrame for the municipality Graz (60101)
graz_data =
municipality_vectors_df[municipality_vectors_df['Municipality'] ==
60101]
graz_data
```

In this step, we create vectors that summarize changes for each individual warning.

These vectors capture key modifications, such as changes in warning type, level, affected municipalities, and timing. By structuring warnings this way, we can analyze trends, detect patterns, and better understand how warnings evolve over time.

[ ]:
```
# Iterate through each pair of rows in the dataframe to correct
'Start Time Changes'
for i in range(0, len(changed_warnings_final), 2):
    # Ensure the second row exists within range
    if i + 1 < len(changed_warnings_final):
        # Check if the first row contains a timestamp in 'Start
Time Changes'
        if isinstance(changed_warnings_final.at[i, 'Start Time
Changes'], pd.Timestamp):
            # Assign False to the first row and True to the second
row
            changed_warnings_final.at[i, 'Start Time Changes'] =
False
            changed_warnings_final.at[i + 1, 'Start Time Changes']
= True
        else:
            changed_warnings_final.at[i, 'Start Time Changes'] =
False
            changed_warnings_final.at[i + 1, 'Start Time Changes']
= False

# Iterate through each pair of rows in the dataframe to correct
'End Time Changes'
for i in range(0, len(changed_warnings_final), 2):
    # Ensure the second row exists within range
    if i + 1 < len(changed_warnings_final):
        # Check if the first row contains a timestamp in 'End Time
Changes'
        if isinstance(changed_warnings_final.at[i, 'End Time
Changes'], pd.Timestamp):
            # Assign False to the first row and True to the second
row
            changed_warnings_final.at[i, 'End Time Changes'] =
False
```

```
            changed_warnings_final.at[i + 1, 'End Time Changes'] =
True
        else:
            changed_warnings_final.at[i, 'End Time Changes'] =
False
            changed_warnings_final.at[i + 1, 'End Time Changes'] =
False

# Iterate through each pair of rows to combine 'Difference in
Warning Type' and 'Difference in Warning Level' into a tuple for
the second row
for i in range(0, len(changed_warnings_final), 2):
    if i + 1 < len(changed_warnings_final):
        # Create a tuple of 'Difference in Warning Type' and
'Difference in Warning Level' from the first row and assign it to
the second row
        warning_type_tuple = (int(changed_warnings_final.at[i,
'Difference in Warning Type']), int(changed_warnings_final.at[i +
1, 'Difference in Warning Type']))
        warning_level_tuple = (int(changed_warnings_final.at[i,
'Difference in Warning Level']), int(changed_warnings_final.at[i +
1, 'Difference in Warning Level']))

        # Assign these tuples to the second row
        changed_warnings_final.at[i + 1, 'Difference in Warning
Type'] = warning_type_tuple
        changed_warnings_final.at[i + 1, 'Difference in Warning
Level'] = warning_level_tuple
        changed_warnings_final.at[i, 'Difference in Warning Type']
= 0
        changed_warnings_final.at[i, 'Difference in Warning
Level'] = 0

# Display the first few rows of the modified dataframe for
verification
changed_warnings_final.head()
```

```
[ ]: # Relevant columns for the vector
     vector_columns = [
         'Difference in Municipality Numbers', 'Difference in Warning
     Type', 'Difference in Warning Level',
         'Difference in Value Numbers', 'Start Time Changes', 'End Time
     Changes',
         'Created Same Day as Start', 'Modified Same Day as Start'
     ]

     # Extract every second row (only consider modified rows)
     changed_filtered = changed_warnings_final.iloc[1::2].copy().
      ↪reset_index(drop = True)

     # Select only the desired columns for the final vector
     changed_vectors = changed_filtered[['Warning-ID', 'Sequence-ID'] +
     vector_columns]

     # Add the 'Cancelled' column and set all values to False
     changed_vectors.loc[:, 'Cancelled'] = False

     # Save the result to a CSV file
     changed_vectors.to_csv("changed_warning_vectors.csv", index =
     False)
```

[ ]:
```
# Load the CSV file
file_path = "changed_warning_vectors.csv"
changed_vectors = pd.read_csv(file_path)

# Convert True/False columns to numeric (if needed)
changed_vectors['Created Same Day as Start'] =
changed_vectors['Created Same Day as Start'].astype(bool)
changed_vectors['Modified Same Day as Start'] =
changed_vectors['Modified Same Day as Start'].astype(bool)

# Count how many warnings were Created Same Day as Start
created_same_day_count = changed_vectors['Created Same Day as
Start'].sum()

# Count how many warnings were Modified Same Day as Start
modified_same_day_count = changed_vectors['Modified Same Day as
Start'].sum()
```

```
# Print results
print(f"Number of warnings Created Same Day as Start:
{created_same_day_count}")
print(f"Number of warnings Modified Same Day as Start:
{modified_same_day_count}")
```

[ ]:
```
# Iterate through each pair of rows in the dataframe to correct
'Start Time Changes'
for i in range(0, len(cancelled_warnings_final), 2):
    # Ensure the second row exists within range
    if i + 1 < len(cancelled_warnings_final):
        # Check if the first row contains a timestamp in 'Start
Time Changes'
        if isinstance(cancelled_warnings_final.at[i, 'Start Time
Changes'], pd.Timestamp):
            # Assign False to the first row and True to the second
row
            cancelled_warnings_final.at[i, 'Start Time Changes'] =
False
            cancelled_warnings_final.at[i + 1, 'Start Time
Changes'] = True
        else:
            cancelled_warnings_final.at[i, 'Start Time Changes'] =
False
            cancelled_warnings_final.at[i + 1, 'Start Time
Changes'] = False

# Iterate through each pair of rows in the dataframe to correct
'End Time Changes'
for i in range(0, len(cancelled_warnings_final), 2):
    # Ensure the second row exists within range
    if i + 1 < len(cancelled_warnings_final):
        # Check if the first row contains a timestamp in 'End Time
Changes'
        if isinstance(cancelled_warnings_final.at[i, 'End Time
Changes'], pd.Timestamp):
            # Assign False to the first row and True to the second
row
```

```
            cancelled_warnings_final.at[i, 'End Time Changes'] =
False
            cancelled_warnings_final.at[i + 1, 'End Time Changes']
= True
        else:
            cancelled_warnings_final.at[i, 'End Time Changes'] =
False
            cancelled_warnings_final.at[i + 1, 'End Time Changes']
= False

# Iterate through each pair of rows to combine 'Difference in
Warning Type' and 'Difference in Warning Level' into a tuple for
the second row
for i in range(0, len(cancelled_warnings_final), 2):
    if i + 1 < len(cancelled_warnings_final):
        # Create a tuple of 'Difference in Warning Type' and
'Difference in Warning Level' from the first row and assign it to
the second row
        warning_type_tuple = (int(cancelled_warnings_final.at[i,
'Difference in Warning Type']), int(cancelled_warnings_final.at[i
+ 1, 'Difference in Warning Type']))
        warning_level_tuple = (int(cancelled_warnings_final.at[i,
'Difference in Warning Level']), int(cancelled_warnings_final.at[i
+ 1, 'Difference in Warning Level']))

        # Assign these tuples to the second row
        cancelled_warnings_final.at[i + 1, 'Difference in Warning
Type'] = warning_type_tuple
        cancelled_warnings_final.at[i + 1, 'Difference in Warning
Level'] = warning_level_tuple
        cancelled_warnings_final.at[i, 'Difference in Warning
Type'] = 0
        cancelled_warnings_final.at[i, 'Difference in Warning
Level'] = 0

# Display the first few rows of the modified dataframe for
verification
cancelled_warnings_final.head()
```

```
[ ]: # Relevant columns for the vector
vector_columns = [
    'Difference in Municipality Numbers', 'Difference in Warning
Type', 'Difference in Warning Level',
    'Difference in Value Numbers', 'Start Time Changes', 'End Time
Changes',
    'Created Same Day as Start', 'Modified Same Day as Start'
]

# Extract every second row (only consider modified rows)
cancelled_filtered = cancelled_warnings_final.iloc[1::2].copy().
 ↪reset_index(drop = True)

# Select only the desired columns for the final vector
cancelled_vectors = cancelled_filtered[['Warning-ID',
'Sequence-ID'] + vector_columns]

# Add the 'Cancelled' column and set all values to True
cancelled_vectors.loc[:, 'Cancelled'] = True

# Save the result to a CSV file
cancelled_vectors.to_csv("cancelled_warning_vectors.csv", index =
False)
```

```
[ ]: # Load the CSV file
file_path = "cancelled_warning_vectors.csv"
cancelled_vectors = pd.read_csv(file_path)

# Convert True/False columns to numeric (if needed)
cancelled_vectors['Created Same Day as Start'] =
cancelled_vectors['Created Same Day as Start'].astype(bool)
cancelled_vectors['Modified Same Day as Start'] =
cancelled_vectors['Modified Same Day as Start'].astype(bool)

# Count how many warnings were Created Same Day as Start
created_same_day_count = cancelled_vectors['Created Same Day as
Start'].sum()

# Count how many warnings were Modified Same Day as Start
```

```
modified_same_day_count = cancelled_vectors['Modified Same Day as
Start'].sum()

# Print results
print(f"Number of warnings Created Same Day as Start:
{created_same_day_count}")
print(f"Number of warnings Modified Same Day as Start:
{modified_same_day_count}")
```

```
[ ]: # Load the two CSV files into DataFrames
changed_warning_vectors = pd.read_csv('changed_warning_vectors.
 ↪csv')
cancelled_warning_vectors = pd.
 ↪read_csv('cancelled_warning_vectors.csv')

# Merge the two DataFrames based on 'Warning-ID' and 'Sequence-ID'
merged_vectors = pd.merge(changed_warning_vectors,
cancelled_warning_vectors, on = ['Warning-ID', 'Sequence-ID',
    'Difference in Municipality Numbers', 'Difference in Warning
Type', 'Difference in Warning Level',
    'Difference in Value Numbers', 'Start Time Changes', 'End Time
Changes', 'Created Same Day as Start',
    'Modified Same Day as Start', 'Cancelled'], how = 'outer')

# Sort the resulting DataFrame by 'Warning-ID' and 'Sequence-ID'
merged_vectors_sorted = merged_vectors.sort_values(by =
['Warning-ID', 'Sequence-ID'])

# Save the merged and sorted DataFrame to a CSV file
merged_vectors_sorted.to_csv("warning_vectors.csv", index = False)
```