### **Data Preprocessing and Merging Documentation**

#### **Overview**

This document outlines the preprocessing and merging steps undertaken to integrate four datasets—disasters, events, economic data, and weather conditions—into a unified dataset. The primary objective was to align data temporally and spatially to facilitate comprehensive analysis.

#### **1. Data Sources**

* **Disasters Data**: Included event details such as type, severity, casualties, economic impact, and timestamps.
* **Events Data**: Contained event specifics like event type and location details.
* **Economic Data**: Comprised business metrics such as revenue and transactions.
* **Weather Data**: Provided environmental data including temperature, humidity, and precipitation.

#### **2. Preprocessing Steps**

##### **Disasters Data**

* Parsed the date field as a datetime object.
* Added 13 years to align the disaster dataset temporally with other datasets:  
  disasters["timestamp"] = disasters["date"] + pd.DateOffset(years=13)
* Dropped the original date column after timestamp alignment.

##### **Events and Economic Data**

Converted the date fields to datetime objects and renamed them to timestamp for consistency.  
events["timestamp"] = pd.to\_datetime(events["date"])

* economic["timestamp"] = pd.to\_datetime(economic["date"])
* Dropped the original date columns.

##### **Weather Data**

* Ensured the timestamp field was properly parsed as datetime during import.

#### **3. Merging Strategy**

Employed an outer join on the unified timestamp column to merge datasets:  
merged = disasters.merge(events, on="timestamp", how="outer", suffixes=('\_disaster', '\_event'))

merged = merged.merge(economic, on="timestamp", how="outer")

* merged = merged.merge(weather, on="timestamp", how="outer")

#### **4. Column Cleanup**

Dropped unnecessary columns to streamline the dataset:

columns\_to\_drop = [

"location\_disaster", "event\_id\_event", "name", "is\_major\_event",

"hour\_x", "event\_day", "hour\_y", "date"

]

merged.drop(columns=columns\_to\_drop, inplace=True, errors="ignore")

#### **5. Geospatial Zone Assignment**

* Filtered rows with valid geographic coordinates (latitude and longitude).
* Applied KMeans clustering to spatially group data points into 50 distinct geographic zones:

from sklearn.cluster import KMeans

coords = merged.loc[merged["latitude"].notna() & merged["longitude"].notna(), ["latitude", "longitude"]].to\_numpy()

kmeans = KMeans(n\_clusters=50, random\_state=42)

merged.loc[merged["latitude"].notna() & merged["longitude"].notna(), "zone\_id"] = kmeans.fit\_predict(coords)

#### **6. Final Output**

* Generated the finalized merged dataset with zone identifiers.
* Exported the dataset as a CSV file (final\_merged\_with\_zones.csv) for further analysis.

#### **Conclusion**

The resulting merged dataset, enriched with geographic zoning, provides a robust foundation for in-depth temporal and spatial analysis.