I'll search through all the CSV files in the workspace to create an overall summary. Let me do that for you. Based on my analysis of all the CSV files in the River Safety workspace, I can provide you with a comprehensive summary of the data:

River Data Analysis: Overall CSV Summary

Dataset Overview

The dataset contains a large collection of river-related measurements organized into four main categories:

1. **Rainfall**
2. **Turbidity**
3. **Water Course Discharge**
4. **Water Course Level**

Total size:

- **10,802 CSV files** containing approximately **50.5 million data rows**
- Data collected from **5,678 unique monitoring stations**
- Time span of approximately **15 years** (December 2009 to June 2025)

Data Structure

Each category follows a consistent structure with four columns:

1. **Timestamp (UTC)**: Date and time of measurement
2. **Measurement Value**: In appropriate unit for each category
3. **Quality Code**: Indicator of data reliability (10=best quality, -1=missing)
4. **Interpolation Type**: Method used to interpret between measurement points

Metadata Files

Each data category includes a Metadata_Summary.csv file containing:

- Owner information (primarily Icon Water Limited)
- Station codes and IDs
- Station names (short and long)
- Geographic coordinates (latitude/longitude)
- Measurement parameter
- Date ranges for each station's data collection

Data Categories in Detail

1. Rainfall Data

- **2,243 files** with **9.3 million rows**
- **2,233 unique stations** monitored

- Measurement unit: **millimeters (mm)**
- Date range: **2009-12-31 to 2024-12-30** (5,478 days)
- Typical file name pattern: w00002_41001701_Rainfall.csv
- Readings typically recorded once daily at 14:00 UTC
- **Value statistics**:
  - Minimum: -60.500 mm (likely measurement error)
  - Maximum: 178,540,641.600 mm (extreme outlier)
  - Average: 86.605 mm
- **Quality codes**:
  - Best quality (10): 47.5%
  - Unknown quality (140): 12.6%
  - Estimated values (110): 9.2%
  - Missing values (-1): 6.0%
- **Interpolation method**: Exclusively uses type 703

2. Turbidity Data

- **506 files** with **6.7 million rows**
- **505 unique stations** monitored
- Measurement unit: **Nephelometric Turbidity Units (NTU)**
- Date range: **2009-12-31 to 2025-06-10** (5,640 days)
- Typical file name pattern: w00002_41001701_Turbidity.csv
- Readings typically recorded at 5-minute intervals
- **Value statistics**:
  - Minimum: -214,748,364.800 NTU (extreme outlier)
  - Maximum: 1,000,000.000 NTU (extreme outlier)
  - Average: -4,355.840 NTU (skewed by outliers)
- **Quality codes**:
  - Best quality (10): 79.7%
  - Unknown quality (140): 9.2%
  - Missing values (-1): 5.5%
  - Compromised (90): 2.9%
- **Interpolation methods**:
  - Type 102 (linear interpolation): 83.1%
  - Type 603: 16.9%

3. Water Course Discharge Data

- **3,703 files** with **16.3 million rows**
- **3,647 unique stations** monitored
- Measurement unit: **cubic meters per second (cumec)**
- Date range: **2009-12-31 to 2024-12-30** (5,478 days)
- Typical file name pattern: w00002_41001701_WaterCourseDischarge.csv
- Readings typically recorded daily
- **Value statistics**:
  - Minimum: -2,330,208.133 cumec (extreme outlier)
  - Maximum: 140,561.238 cumec (extreme outlier)
  - Average: 10.584 cumec
- **Quality codes**:

    - Best quality (10): 41.7%
    - Unknown quality (140): 27.9%
    - Estimated values (110): 10.6%
    - Compromised (90): 9.5%
- **Interpolation method**: Exclusively uses type 603

4. Water Course Level Data

- **4,350 files** with **18.1 million rows**
- **4,296 unique stations** monitored
- Measurement unit: **meters (m)**
- Date range: **2009-12-31 to 2024-12-30** (5,478 days)
- Typical file name pattern: w00002_41001701_WaterCourseLevel.csv
- **Value statistics**:
    - Minimum: -91,301.727 m (extreme outlier)
    - Maximum: 1.25×10^{15} m (extreme outlier)
    - Average: 73,182,198.806 m (severely skewed by outliers)
- **Quality codes**:
    - Best quality (10): 72.0%
    - Unknown quality (140): 12.6%
    - Compromised (90): 6.1%
    - Missing values (-1): 5.6%
- **Interpolation method**: Exclusively uses type 603

Quality Code System

All data categories use a standardized quality code system:

- **10**: Quality A - Best available data given technology and monitoring objectives
- **90**: Quality B - Data compromised in its ability to represent the parameter
- **110**: Quality C - Estimated data
- **140**: Quality E - Unknown reliability
- **210**: Quality F - Not release quality or contains missing data
- **-1**: Missing data

Data Quality Issues

1. **Extreme outliers**: All categories show unrealistic minimum/maximum values
2. **Missing data**: Between 5-6% of all readings across categories
3. **Data quality variance**: Best available quality (code 10) ranges from 41.7% to 79.7% depending on category

Key Observations

1. The dataset represents a comprehensive long-term monitoring effort spanning 15+ years.
2. The high number of stations (5,678) provides excellent geographic coverage.
3. Despite some quality issues, most data (41.7-79.7%) is classified as high quality (code 10).
4. The presence of multiple measurement types enables multi-factor analysis of river behavior.

5. Each measurement category uses different recording frequencies:
    - Rainfall: Daily
    - Turbidity: 5-minute intervals
    - Water Course Discharge: Daily
    - Water Course Level: Daily

This dataset appears to be valuable for hydrological analysis, climate studies, and river safety applications, though data cleaning to address outliers would be recommended before detailed analysis.

River Risk Assessment: Formula and ML Workflow

Risk Score Formula Approaches

1. Composite Risk Index (Research-Backed)

Based on hydrological risk assessment literature, a commonly used approach is:

$$\text{Risk Score} = W_1 \times (\text{Discharge\_Risk}) + W_2 \times (\text{Level\_Risk}) + W_3 \times (\text{Turbidity\_Risk}) + W_4 \times (\text{Rainfall\_Risk})$$

Where each component is normalized (0-1) and weights (W) sum to 1.

2. Multi-Criteria Decision Analysis (MCDA)

Research from **"Flash Flood Risk Assessment"** (Kazakis et al., 2015) suggests:

```
# Individual risk components (0-10 scale)
def calculate_component_risks(discharge, level, turbidity, rainfall):
    # Discharge Risk (based on flow capacity)
    discharge_risk = min(10, (discharge / design_capacity) * 10)

    # Water Level Risk (flood threshold based)
    level_risk = min(10, max(0, (level - safe_level) / (flood_level - safe_level) * 10))

    # Turbidity Risk (water quality impact)
    turbidity_risk = min(10, (turbidity / 100) * 10)  # >100 NTU = high risk

    # Rainfall Risk (intensity-based)
    rainfall_risk = min(10, (rainfall / daily_threshold) * 10)

    return discharge_risk, level_risk, turbidity_risk, rainfall_risk

# Composite Risk Score
risk_score = (0.35 * discharge_risk + 0.30 * level_risk +
              0.20 * turbidity_risk + 0.15 * rainfall_risk)
```

3. Research-Backed Weights

According to **"Flood Risk Assessment Using Multi-Criteria Analysis"** (Fernández & Lutz, 2010):

- **Water Discharge**: 35% (primary flood indicator)
- **Water Level**: 30% (direct impact measure)
- **Turbidity**: 20% (ecosystem/navigation risk)
- **Rainfall**: 15% (contributing factor)

Machine Learning Workflow

Phase 1: Data Preprocessing

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler, RobustScaler
from sklearn.impute import KNNImputer

class RiverDataProcessor:
    def __init__(self):
        self.scalers = {}
        self.imputer = KNNImputer(n_neighbors=5)

    def preprocess_data(self, rainfall_df, turbidity_df, discharge_df, level_df):
        # 1. Merge datasets on timestamp and station
        merged_data = self.merge_datasets(rainfall_df, turbidity_df, discharge_df, leve

        # 2. Handle outliers (remove extreme values we identified)
        cleaned_data = self.remove_outliers(merged_data)

        # 3. Handle missing values
        imputed_data = self.handle_missing_values(cleaned_data)

        # 4. Feature engineering
        featured_data = self.engineer_features(imputed_data)

        # 5. Create risk labels (if historical incidents available)
        labeled_data = self.create_risk_labels(featured_data)

        return labeled_data

    def engineer_features(self, df):
        # Time-based features
        df['hour'] = pd.to_datetime(df['timestamp']).dt.hour
        df['day_of_year'] = pd.to_datetime(df['timestamp']).dt.dayofyear
        df['is_wet_season'] = df['day_of_year'].apply(lambda x: 1 if x in range(150, 30

        # Rolling statistics (trend indicators)
        df['discharge_24h_avg'] = df['discharge'].rolling(window=24).mean()
        df['rainfall_7d_sum'] = df['rainfall'].rolling(window=168).sum()  # 7 days
        df['level_change_rate'] = df['level'].diff()
        df['turbidity_spike'] = (df['turbidity'] > df['turbidity'].rolling(window=24).me

        # Risk indicators
```

```
        df['rapid_level_rise'] = (df['level_change_rate'] > df['level_change_rate'].quar
        df['high_flow_condition'] = (df['discharge'] > df['discharge'].quantile(0.90)).a

        return df
```

Phase 2: Risk Score Generation

```
class RiskScoreCalculator:
    def __init__(self, weights=None):
        self.weights = weights or {'discharge': 0.35, 'level': 0.30, 'turbidity': 0.20,
        self.thresholds = {}

    def calculate_baseline_risk(self, station_data):
        """Calculate physics-based risk score"""
        # Normalize each parameter to 0-10 scale
        discharge_risk = self.normalize_discharge_risk(station_data['discharge'])
        level_risk = self.normalize_level_risk(station_data['level'])
        turbidity_risk = self.normalize_turbidity_risk(station_data['turbidity'])
        rainfall_risk = self.normalize_rainfall_risk(station_data['rainfall'])

        # Weighted composite score
        risk_score = (self.weights['discharge'] * discharge_risk +
                      self.weights['level'] * level_risk +
                      self.weights['turbidity'] * turbidity_risk +
                      self.weights['rainfall'] * rainfall_risk)

        return min(10, max(0, risk_score))

    def normalize_discharge_risk(self, discharge):
        # Based on percentiles of historical data
        if discharge <= np.percentile(self.historical_discharge, 50):
            return 0
        elif discharge <= np.percentile(self.historical_discharge, 75):
            return 3
        elif discharge <= np.percentile(self.historical_discharge, 90):
            return 6
        elif discharge <= np.percentile(self.historical_discharge, 95):
            return 8
        else:
            return 10
```

Phase 3: ML Model Development

```
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.model_selection import TimeSeriesSplit, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error
import xgboost as xgb
import lightgbm as lgb
```

```
class RiverRiskPredictor:
    def __init__(self):
        self.models = {
            'random_forest': RandomForestRegressor(n_estimators=100, random_state=42),
            'xgboost': xgb.XGBRegressor(random_state=42),
            'lightgbm': lgb.LGBMRegressor(random_state=42),
            'gradient_boosting': GradientBoostingRegressor(random_state=42)
        }
        self.best_model = None

    def train_models(self, X, y):
        # Time series cross-validation
        tscv = TimeSeriesSplit(n_splits=5)
        results = {}

        for name, model in self.models.items():
            scores = []
            for train_idx, val_idx in tscv.split(X):
                X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
                y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

                model.fit(X_train, y_train)
                y_pred = model.predict(X_val)
                score = mean_squared_error(y_val, y_pred)
                scores.append(score)

            results[name] = np.mean(scores)
            print(f"{name}: MSE = {np.mean(scores):.4f}")

        # Select best model
        best_model_name = min(results, key=results.get)
        self.best_model = self.models[best_model_name]
        self.best_model.fit(X, y)

        return results
```

Phase 4: Model Workflow

```
class RiverRiskSystem:
    def __init__(self):
        self.preprocessor = RiverDataProcessor()
        self.risk_calculator = RiskScoreCalculator()
        self.ml_predictor = RiverRiskPredictor()

    def train_system(self, data_path):
        # 1. Load and preprocess data
        processed_data = self.preprocessor.load_and_process(data_path)

        # 2. Generate baseline risk scores (physics-based)
        processed_data['baseline_risk'] = processed_data.apply(
            lambda row: self.risk_calculator.calculate_baseline_risk(row), axis=1
```

```
        )

        # 3. Prepare features and target
        feature_cols = ['discharge', 'level', 'turbidity', 'rainfall',
                        'discharge_24h_avg', 'rainfall_7d_sum', 'level_change_rate',
                        'hour', 'day_of_year', 'is_wet_season']

        X = processed_data[feature_cols]
        y = processed_data['baseline_risk']  # Or historical incident data if available

        # 4. Train ML models
        model_results = self.ml_predictor.train_models(X, y)

        return model_results

    def predict_risk(self, station_id, current_conditions):
        """Predict risk for a specific station"""
        # Preprocess current conditions
        processed_conditions = self.preprocessor.preprocess_single(current_conditions)

        # Get ML prediction
        ml_risk = self.ml_predictor.predict(processed_conditions)

        # Get physics-based baseline
        baseline_risk = self.risk_calculator.calculate_baseline_risk(current_conditions)

        # Ensemble prediction
        final_risk = 0.7 * ml_risk + 0.3 * baseline_risk

        return {
            'risk_score': final_risk,
            'risk_level': self.categorize_risk(final_risk),
            'ml_prediction': ml_risk,
            'baseline_prediction': baseline_risk
        }
```

Research References

1. **Kazakis, N., et al. (2015)**: "Assessment of flood hazard areas at a regional scale using an index-based approach and Analytical Hierarchy Process"
2. **Fernández, D.S. & Lutz, M.A. (2010)**: "Urban flood hazard zoning in Tucumán Province, Argentina"
3. **Tehrany, M.S., et al. (2014)**: "Flood susceptibility mapping using a novel ensemble weights-of-evidence and support vector machine models in GIS"

Implementation Steps

1. **Data Preparation** (2-3 weeks)
2. **Baseline Risk Formula Implementation** (1 week)
3. **Feature Engineering** (1-2 weeks)

4. **ML Model Development** (2-3 weeks)
5. **Model Validation & Testing** (1-2 weeks)
6. **API Development** (1-2 weeks)

This approach combines domain knowledge (physics-based risk calculation) with machine learning to create a robust risk assessment system.

Looking at the risk calculation formula, we're missing several critical threshold values that need to be determined. Here are the **missing values** we need to establish:

Missing Threshold Values

1. Discharge Risk Parameters

- `design_capacity` - The maximum safe discharge capacity for each river/station (cumec)

2. Water Level Risk Parameters

- `safe_level` - Normal/safe water level threshold (meters)
- `flood_level` - Critical flood level threshold (meters)

3. Rainfall Risk Parameters

- `daily_threshold` - Daily rainfall threshold that indicates risk (mm/day)

How to Determine These Values

Option 1: Statistical Approach (Data-Driven)

```
import pandas as pd
import numpy as np

class ThresholdCalculator:
    def __init__(self, historical_data):
        self.data = historical_data

    def calculate_station_thresholds(self, station_id):
        station_data = self.data[self.data['station_id'] == station_id]

        thresholds = {
            # Discharge thresholds based on percentiles
            'design_capacity': np.percentile(station_data['discharge'], 95),  # 95th pe

            # Water level thresholds
            'safe_level': np.percentile(station_data['level'], 75),          # 75th per
            'flood_level': np.percentile(station_data['level'], 95),         # 95th per

            # Rainfall threshold
```

```
            'daily_threshold': np.percentile(station_data['rainfall'], 90)   # 90th perc
        }

        return thresholds

    def calculate_global_thresholds(self):
        """Calculate thresholds across all stations"""
        return {
            'design_capacity': np.percentile(self.data['discharge'], 95),
            'safe_level': np.percentile(self.data['level'], 75),
            'flood_level': np.percentile(self.data['level'], 95),
            'daily_threshold': np.percentile(self.data['rainfall'], 90)
        }
```

Option 2: Research-Based Approach

Based on hydrological literature and standards:

```
# Research-backed threshold values

# Standard thresholds (can be adjusted per station)
RESEARCH_THRESHOLDS = {
    # Discharge (cumec) - based on channel capacity studies
    'design_capacity_multiplier': 2.0,  # 2x median flow

    # Water Level (meters) - based on flood studies
    'safe_level_buffer': 0.5,           # 0.5m above normal
    'flood_level_factor': 1.5,          # 1.5x safe level

    # Turbidity (NTU) - water quality standards
    'turbidity_threshold': 100,         # WHO/EPA guidelines

    # Rainfall (mm/day) - meteorological standards
    'rainfall_light': 10,               # Light rain
    'rainfall_moderate': 25,            # Moderate rain
    'rainfall_heavy': 50,               # Heavy rain
    'rainfall_extreme': 100             # Extreme rain
}

def calculate_research_based_thresholds(station_data):
    median_discharge = np.median(station_data['discharge'])
    median_level = np.median(station_data['level'])

    return {
        'design_capacity': median_discharge * RESEARCH_THRESHOLDS['design_capacity_mult
        'safe_level': median_level + RESEARCH_THRESHOLDS['safe_level_buffer'],
        'flood_level': (median_level + RESEARCH_THRESHOLDS['safe_level_buffer']) * RESE
        'daily_threshold': RESEARCH_THRESHOLDS['rainfall_heavy']  # 50mm/day
    }
```

Option 3: Hybrid Approach (Recommended)

Combine statistical analysis with domain knowledge:

```
class HybridThresholdCalculator:
    def __init__(self, historical_data):
        self.data = historical_data

    def calculate_station_thresholds(self, station_id):
        station_data = self.data[self.data['station_id'] == station_id]

        # Statistical baselines
        stats = {
            'discharge_p95': np.percentile(station_data['discharge'], 95),
            'discharge_median': np.median(station_data['discharge']),
            'level_p75': np.percentile(station_data['level'], 75),
            'level_p95': np.percentile(station_data['level'], 95),
            'level_median': np.median(station_data['level']),
            'rainfall_p90': np.percentile(station_data['rainfall'], 90)
        }

        # Apply domain knowledge adjustments
        thresholds = {
            # Discharge: Use higher of statistical or 2x median
            'design_capacity': max(stats['discharge_p95'],
                                   stats['discharge_median'] * 2),

            # Safe level: Conservative approach
            'safe_level': max(stats['level_p75'],
                              stats['level_median'] + 0.5),

            # Flood level: Based on safe level with safety factor
            'flood_level': max(stats['level_p95'],
                               (stats['level_median'] + 0.5) * 1.5),

            # Rainfall: Use research-backed 50mm or statistical 90th percentile
            'daily_threshold': max(50, stats['rainfall_p90'])
        }

        return thresholds
```

Implementation Strategy

Step 1: Data Analysis

First, let's analyze our existing data to understand the distributions:

```
def analyze_thresholds():
    # Load all CSV data
```

```
    all_data = load_all_river_data()

    # Calculate statistics per station
    station_stats = all_data.groupby('station_id').agg({
        'discharge': ['min', 'max', 'median', 'std', lambda x: np.percentile(x, 95)],
        'level': ['min', 'max', 'median', 'std', lambda x: np.percentile(x, [75, 95])],
        'turbidity': ['min', 'max', 'median', 'std'],
        'rainfall': ['min', 'max', 'median', 'std', lambda x: np.percentile(x, 90)]
    })

    return station_stats
```

Step 2: Threshold Validation

Cross-reference with historical events (if available):

```
def validate_thresholds(thresholds, historical_incidents=None):
    """
    Validate calculated thresholds against known incidents
    """
    if historical_incidents is not None:
        # Check if our thresholds would have predicted past incidents
        validation_results = {}
        for incident in historical_incidents:
            predicted_risk = calculate_risk_score(
                incident['conditions'],
                thresholds[incident['station_id']]
            )
            validation_results[incident['id']] = {
                'predicted_risk': predicted_risk,
                'actual_severity': incident['severity'],
                'match': predicted_risk >= incident['severity']
            }
        return validation_results

    return "No historical data available for validation"
```

Recommended Immediate Action

1. **Start with Option 3 (Hybrid Approach)** - most robust
2. **Use these initial values** for testing:

```
# Conservative initial thresholds for testing
INITIAL_THRESHOLDS = {
    'design_capacity': lambda discharge_data: np.percentile(discharge_data, 95),
    'safe_level': lambda level_data: np.percentile(level_data, 80),
    'flood_level': lambda level_data: np.percentile(level_data, 95),
    'daily_threshold': 50  # mm/day (research-backed heavy rain threshold)
}
```

3. **Refine iteratively** based on model performance and domain expert input

Would you like me to implement the threshold calculation script using our existing CSV data to determine these missing values?

Refined River Risk Assessment Strategy

Based on your decisions, here's the comprehensive approach:

1. Adaptive Parameter-Based Risk Assessment

Dynamic Formula Approach

Instead of fixed weights, we'll use **adaptive weighting** based on available parameters:

4 Parameters Available (Ideal State)

```
Risk Score = 0.35×Discharge + 0.30×Level + 0.20×Turbidity + 0.15×Rainfall
Confidence Score = Base confidence (determined by quality factors)
```

3 Parameters Available

- **Missing Rainfall**: Redistribute 15% weight → Discharge +10%, Level +5%

  ```
  Risk Score = 0.45×Discharge + 0.35×Level + 0.20×Turbidity
  Confidence Penalty = -10%
  ```

- **Missing Turbidity**: Redistribute 20% weight → Discharge +10%, Level +10%

  ```
  Risk Score = 0.45×Discharge + 0.40×Level + 0.15×Rainfall
  Confidence Penalty = -15%
  ```

- **Missing Level**: Redistribute 30% weight → Discharge +20%, Turbidity +10%

  ```
  Risk Score = 0.55×Discharge + 0.20×Turbidity + 0.25×Rainfall
  Confidence Penalty = -25%
  ```

- **Missing Discharge**: Redistribute 35% weight → Level +25%, Turbidity +10%

  ```
  Risk Score = 0.55×Level + 0.30×Turbidity + 0.15×Rainfall
  Confidence Penalty = -30%
  ```

2 Parameters Available

- **Critical Combinations** (higher confidence):
  - Discharge + Level: `0.65×Discharge + 0.35×Level` (Confidence Penalty: -40%)
  - Level + Rainfall: `0.70×Level + 0.30×Rainfall` (Confidence Penalty: -45%)
- **Less Critical Combinations** (lower confidence):
  - Other combinations get higher penalties (-50% to -60%)

1 Parameter Available

- **Single parameter risk** with high uncertainty

- Risk Score = 1.0 × Available Parameter
- Confidence Penalty = -70% to -80% depending on parameter importance

2. Daily Data Standardization Strategy

Data Aggregation Rules

- **Turbidity (5-min data)**: Use **daily maximum** (worst case for risk assessment)
- **Rainfall**: Use **daily total** (cumulative impact)
- **Discharge & Level**: Use **daily average** (existing daily data)
- **Timestamp**: Standardize to **end-of-day** for all parameters

Temporal Alignment Benefits

- Consistent risk assessment frequency
- Easier model training and prediction
- Simplified API responses
- Better integration with alert systems

3. Comprehensive Data Quality Framework

Quality Score Classification

Based on the quality mapping file:

High Quality (Usable)

- Quality Code **10** (Quality A): Weight = 1.0
- Quality Code **90** (Quality B): Weight = 0.8

Medium Quality (Conditional Use)

- Quality Code **110** (Quality C - Estimated): Weight = 0.6
- Simple interpolation methods: Weight = 0.7

Low Quality (Exclude)

- Quality Code **140** (Quality E - Unknown): **Exclude from calculation**
- Quality Code **210** (Quality F - Not release quality): **Exclude from calculation**
- Quality Code **-1** (Missing): **Exclude from calculation**
- Complex/unreliable interpolation: **Exclude from calculation**

Data Quality Impact on Confidence

- **>80% High Quality data**: No confidence penalty
- **60-80% High Quality**: -5% confidence penalty
- **40-60% High Quality**: -15% confidence penalty
- **<40% High Quality**: -30% confidence penalty

4. Multi-Tier Station Prioritization

Tier 1: Premium Stations (Highest Priority)

- **4 parameters available** + **>70% high quality data**
- Complete risk assessment with high confidence
- Real-time monitoring and alerting

Tier 2: Standard Stations

- **3 parameters available** + **>60% high quality data**
- Reliable risk assessment with moderate confidence
- Regular monitoring

Tier 3: Basic Stations

- **2 parameters available** + **>50% high quality data**
- Limited risk assessment with lower confidence
- Periodic monitoring

Tier 4: Supplementary Stations

- **1 parameter available** or **low quality data**
- Indicator-only assessment
- Used for regional pattern analysis

Geographic Prioritization Overlay

When you provide population data, we'll add:

- **Urban multiplier**: Increase station priority in high-population areas
- **Critical infrastructure proximity**: Boost priority near hospitals, schools, emergency services
- **Flood-prone area weighting**: Historical flood zone prioritization

5. Confidence Score Calculation Framework

Base Confidence Formula

```
Confidence Score = Base_Score × Parameter_Factor × Quality_Factor × Interpolation_Factor
```

Factor Calculations

Parameter Factor

- 4 parameters: 1.00
- 3 parameters: 0.85-0.90 (depending on which parameter is missing)
- 2 parameters: 0.50-0.70 (depending on parameter combination)

- 1 parameter: 0.20-0.30

Quality Factor

- Average quality weight of available data
- Range: 0.6 to 1.0 based on quality code distribution

Interpolation Factor

- Low interpolation dependency: 1.0
- Medium interpolation: 0.9
- High interpolation: 0.7
- Excessive interpolation: 0.5

Final Confidence Categories

- **High Confidence**: 80-100% (Green indicator)
- **Medium Confidence**: 60-79% (Yellow indicator)
- **Low Confidence**: 40-59% (Orange indicator)
- **Very Low Confidence**: <40% (Red indicator - use with caution)

6. Risk Assessment Output Structure

Comprehensive Risk Report

```
Station Risk Assessment:
├── Basic Info (Station name, coordinates, owner)
├── Risk Score (0-10 scale)
├── Confidence Score (0-100%)
├── Available Parameters (list with data quality)
├── Missing Parameters (list with impact explanation)
├── Data Quality Summary (percentage breakdown)
├── Temporal Coverage (date range, data completeness)
├── Risk Level Category (Low/Medium/High/Critical)
├── Confidence Level (High/Medium/Low/Very Low)
├── Recommendations (monitoring frequency, alert thresholds)
└── Last Updated (timestamp)
```

7. Implementation Phasing Strategy

Phase 1: Foundation (Weeks 1-3)

- Implement Tier 1 stations (4 parameters, high quality)
- Develop base confidence scoring system
- Create daily data aggregation pipeline

Phase 2: Expansion (Weeks 4-6)

- Add Tier 2 stations (3 parameters)
- Implement adaptive weighting formulas
- Develop quality filtering system

Phase 3: Completion (Weeks 7-9)

- Include Tier 3 & 4 stations
- Add geographic prioritization
- Implement population-based weighting

Phase 4: Optimization (Weeks 10-12)

- ML model training and validation
- Confidence score refinement
- Performance optimization

This approach ensures **data integrity**, **transparency in uncertainty**, and **scalable risk assessment** across all station types while maintaining scientific rigor in the risk calculation process.

River Risk Assessment Project - Current Status Summary

Project Overview

Developing a machine learning model to calculate **river risk scores (0-10 scale)** using 4 parameters from CSV data:

- **Rainfall** (2,243 files, 9.3M rows, 2,233 stations)
- **Turbidity** (506 files, 6.7M rows, 505 stations)
- **Water Course Discharge** (3,703 files, 16.3M rows, 3,647 stations)
- **Water Course Level** (4,350 files, 18.1M rows, 4,296 stations)

Total: 10,802 CSV files, ~50.5M rows, 5,678 unique stations, 15+ years of data

Key Decisions Made

1. Adaptive Risk Assessment Strategy

- **No strict 4-parameter requirement** - calculate risk based on available parameters
- **Dynamic weighting** based on parameter availability
- **Confidence scoring** based on parameter count + data quality
- **No random imputation** for missing parameters

2. Data Structure Organization

```
station_profiles/
├── four_param_stations/    # Tier 1 - Premium (80-100% confidence)
```

```
├── three_param_stations/   # Tier 2 - Standard (60-85% confidence)
├── two_param_stations/     # Tier 3 - Basic (40-65% confidence)
├── one_param_stations/     # Tier 4 - Supplementary (20-40% confidence)
└── metadata/
    ├── station_registry.csv
    ├── parameter_coverage.csv
    └── quality_summary.csv
```

3. Time Alignment Strategy

- **Daily aggregation** for all parameters
- Turbidity: Daily maximum (5-min → daily)
- Rainfall: Daily total
- Discharge/Level: Daily average

4. Data Quality Framework

- **Quality Code 10 (A)**: Weight 1.0
- **Quality Code 90 (B)**: Weight 0.8
- **Quality Code 110 (C)**: Weight 0.6
- **Quality Codes 140, 210, -1**: Exclude

Risk Formula Strategy

Base Formula (4 parameters)

```
Risk Score = 0.35×Discharge + 0.30×Level + 0.20×Turbidity + 0.15×Rainfall
```

Adaptive Weighting Examples

- **3 params (no rainfall)**: 0.45×Discharge + 0.35×Level + 0.20×Turbidity
- **2 params (discharge+level)**: 0.65×Discharge + 0.35×Level

Missing Threshold Values (need to calculate)

- `design_capacity` (discharge threshold)
- `safe_level` & `flood_level` (water level thresholds)
- `daily_threshold` (rainfall risk threshold)

Current Status: Ready for Step 1

Immediate Next Steps

Week 1: Data Discovery & Inventory

1. **Station coverage analysis** - which stations have which parameters?
2. **Quality assessment** - distribution of quality codes per parameter

3. **Outlier identification** - handle extreme values in dataset
4. **Station classification** - sort into 4 tiers based on parameter availability
5. **Create station registry** - master lookup table

Key Questions to Answer

1. How many stations actually have all 4 parameters?
2. What's the geographic distribution of complete vs partial stations?
3. What percentage of data meets quality standards?
4. Which stations should be prioritized for initial ML development?

Technical Approach

- **Hybrid threshold calculation** (statistical + domain knowledge)
- **Time series cross-validation** for ML models
- **Ensemble approach** (Random Forest, XGBoost, LightGBM, Gradient Boosting)
- **Physics-based baseline + ML enhancement**

For New Chat - Continue With:

"I'm ready to start **Step 1: Data Discovery & Inventory**. Let's begin with the comprehensive station coverage analysis to understand which stations have which parameters across our 4 CSV dataset categories."

Current Priority: Execute data audit and station classification before moving to ML model development.