Predicting and Mapping Waterborne Risk

Charlie Serafin and Kayla Vo

Introduction

Personal Motivation

- Apply machine learning to a real public health problem: identifying unsafe or high-risk water sources before they cause harm
- Practice classification, model evaluation, and interpretability on an environmental dataset with meaningful stakeholder impact
- Help communities and health officials make data-driven decisions about water safety

Dataset Description

Structure & Size:

- Source: Kaggle "Water Pollution and Disease" CSV dataset
- 3,000 instances (country–region–year–source combinations)
- ~10 attributes per record

Data Types:

- Categorical: Country, Region, Year, Water Source Type (River, Well, Tap, Lake, Spring, Pond)
- Continuous: Contaminant Level (ppm), pH, Turbidity (NTU), Dissolved Oxygen (mg/L), Nitrate (mg/L), Lead Concentration (µg/L)

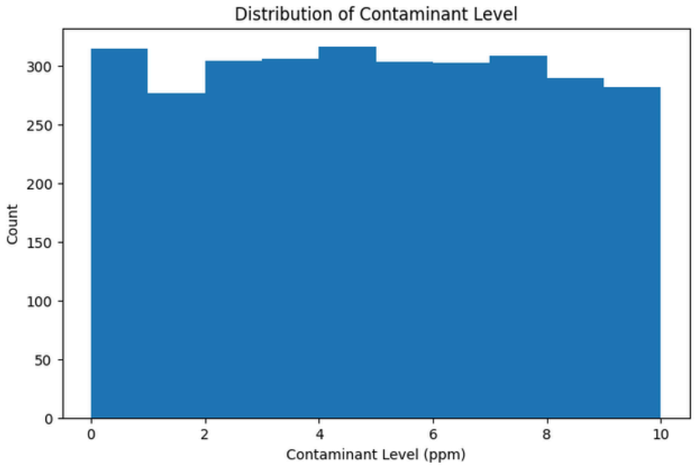
Class Labels (Project-Defined):

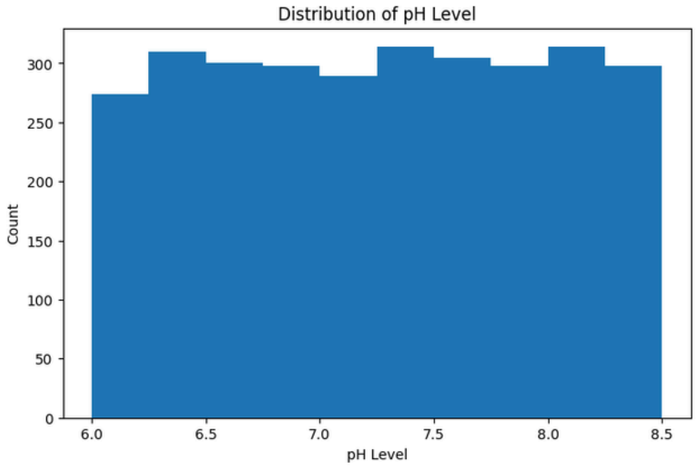
- "Contaminated" (Yes/No) – binary classification using threshold-inspired rules
- "Risk" (Low/Medium/High) – three-class risk categorization from combined measurements

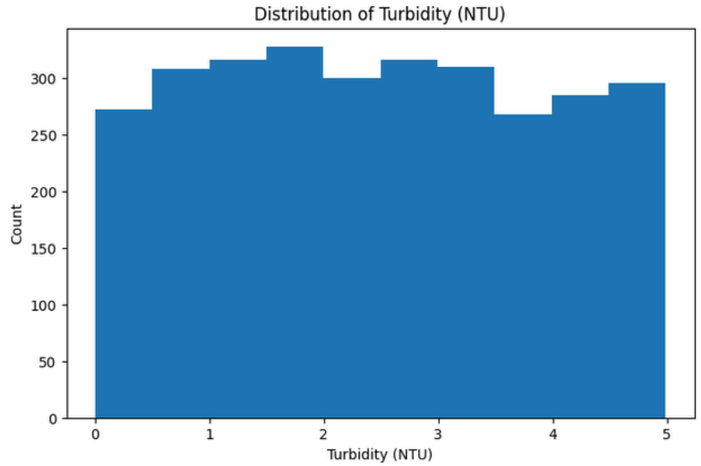
Key Challenges:

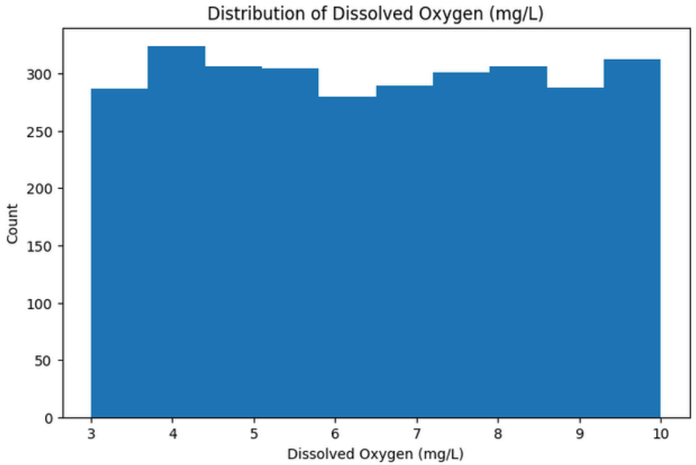
- Missing values in some water-quality measurements
- Possible class imbalance (more safe/low-risk samples than unsafe/high-risk)
- Continuous attributes on different measurement scales → require normalization for KNN and distance-based methods

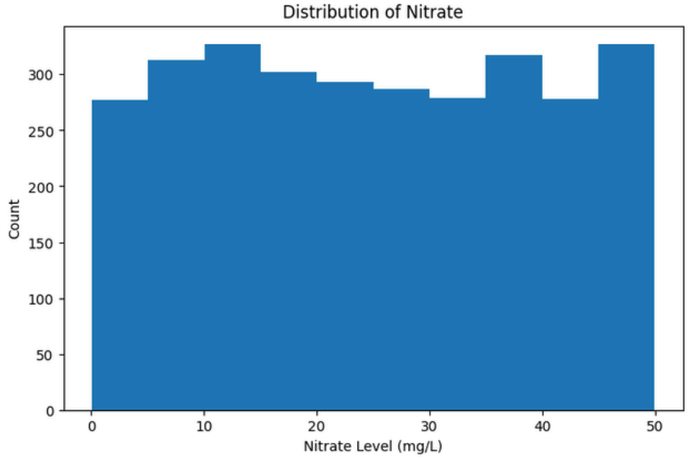
Pre-label EDA

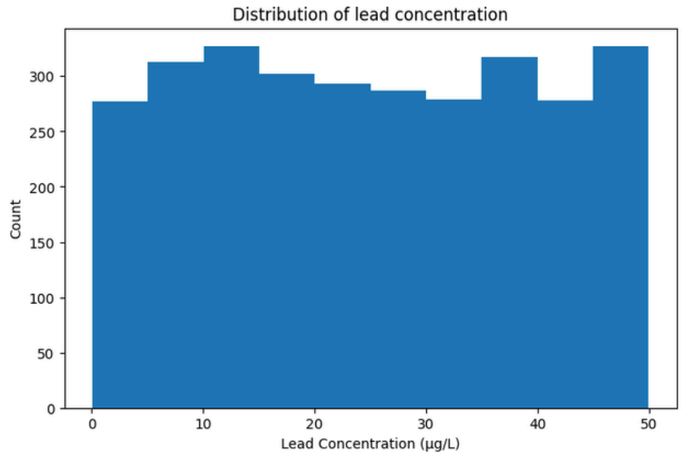




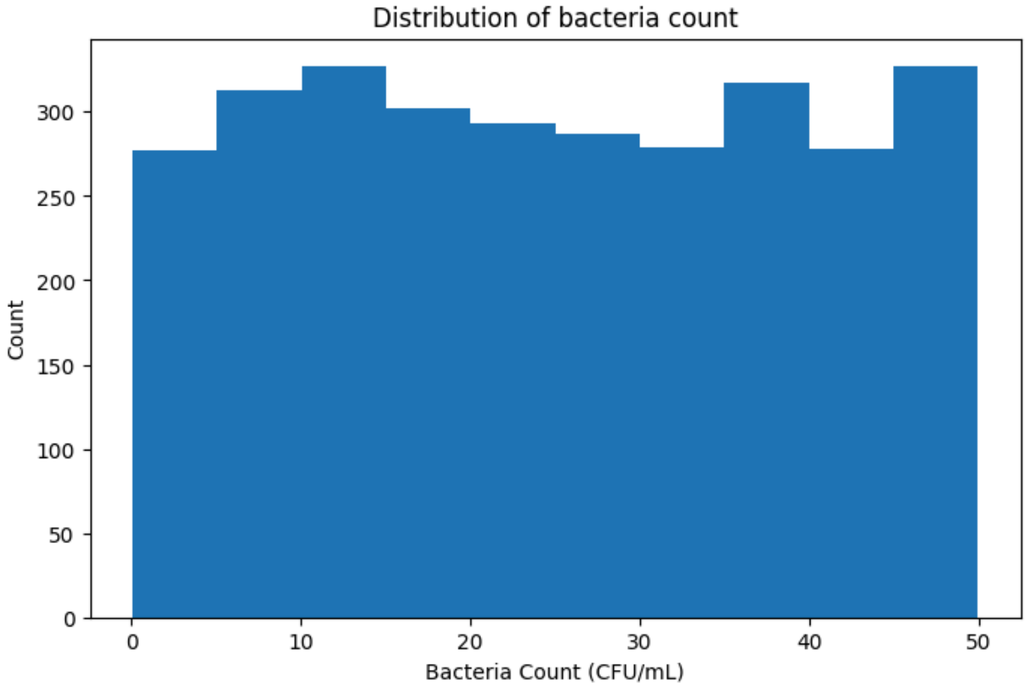


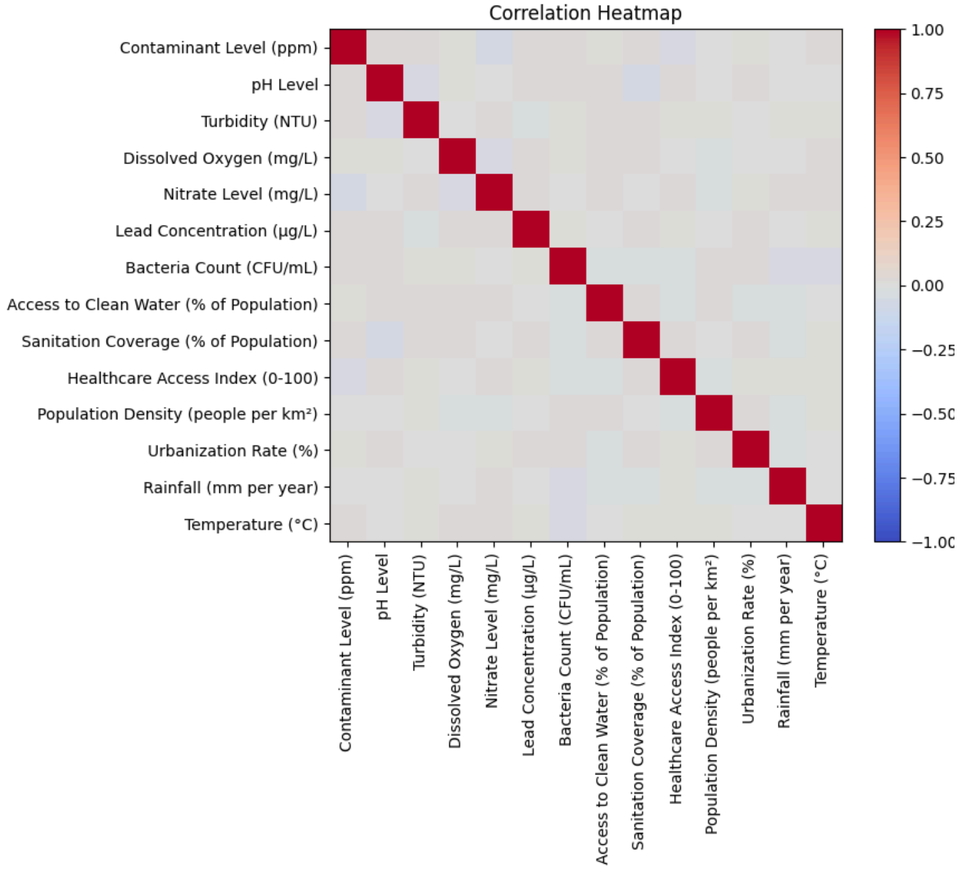






Pre-label EDA



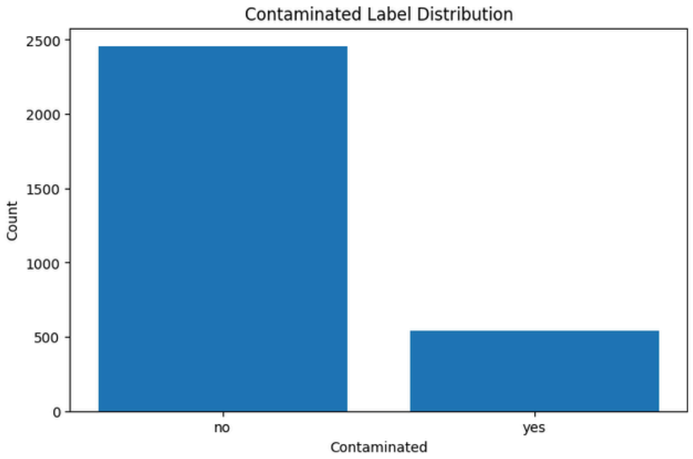


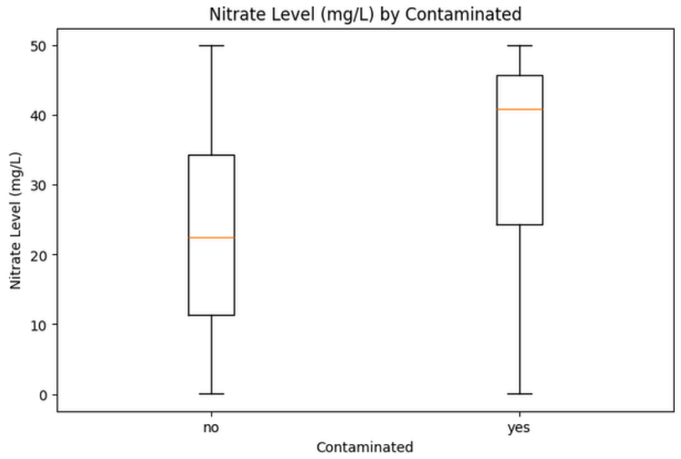
Conclusion:

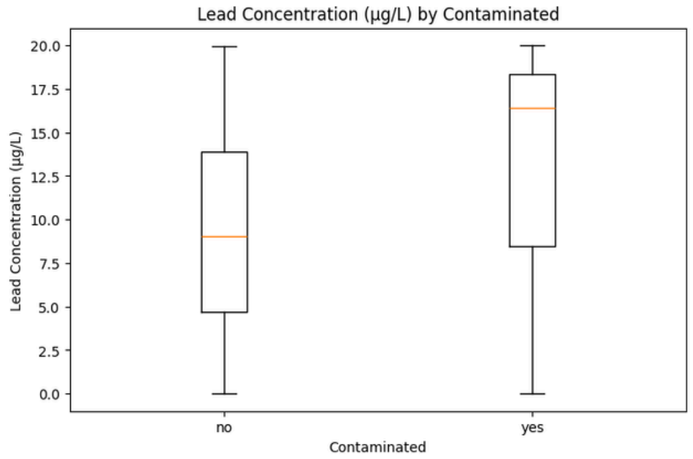
The pre-label EDA shows that the chosen water-quality and risk-related features have diverse, roughly uniform distributions across their ranges. There is no strong linear redundancy among these features, so none need to be removed solely due to high correlation with another feature. With multicollinearity not a concern, the next step is to define Pollution and Risk labels using domain guidelines.

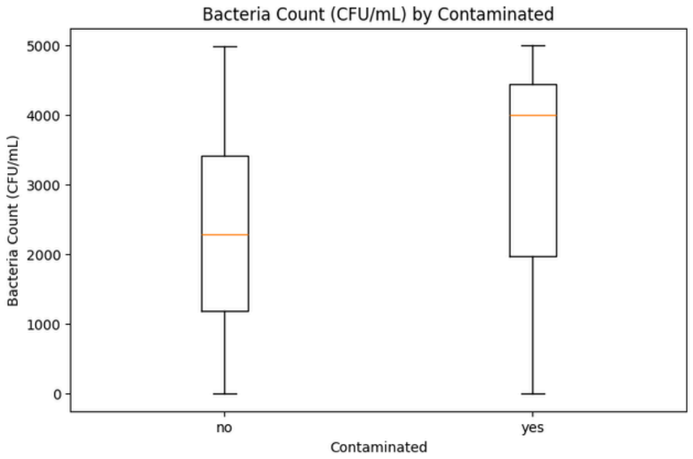
Post-label EDA

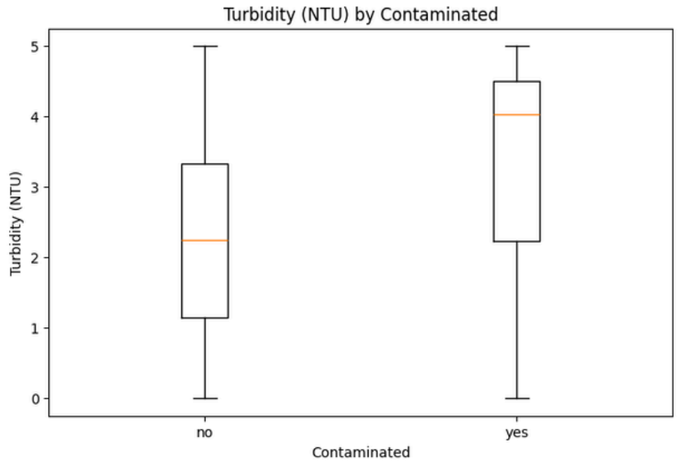
Contaminated Label

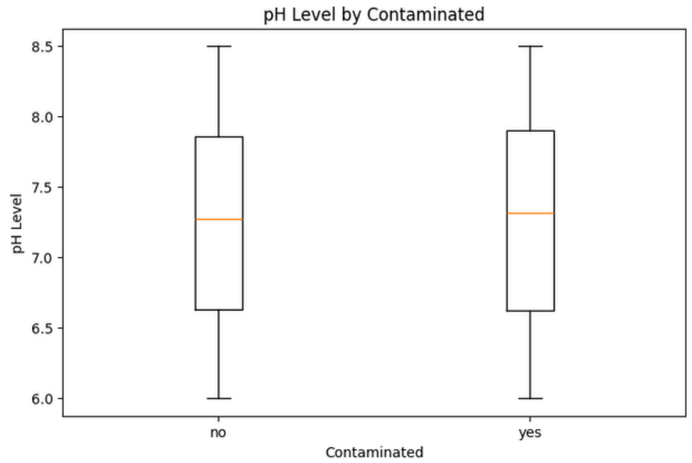






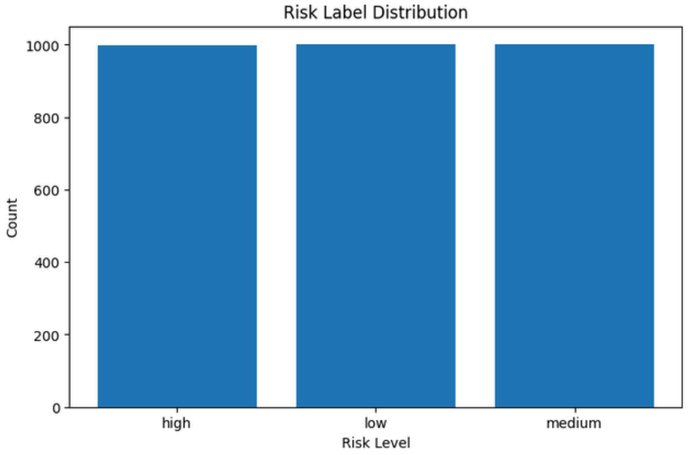


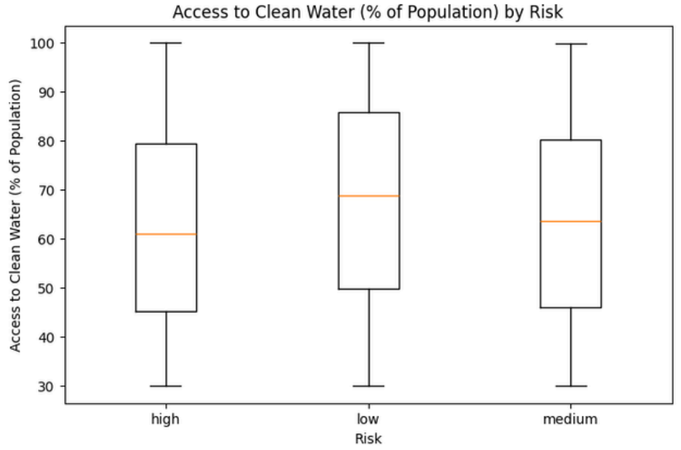


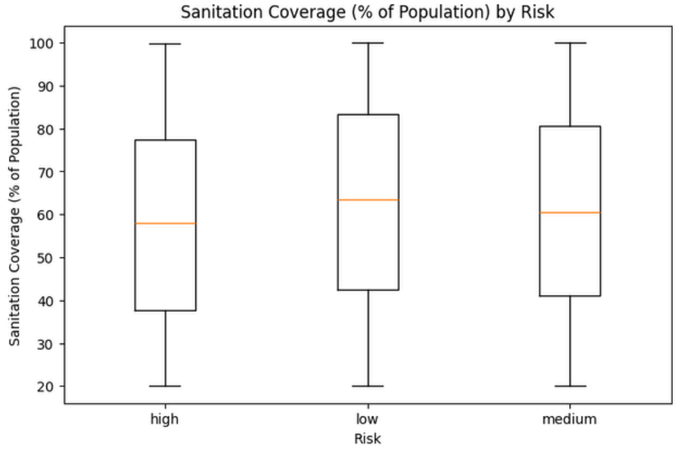


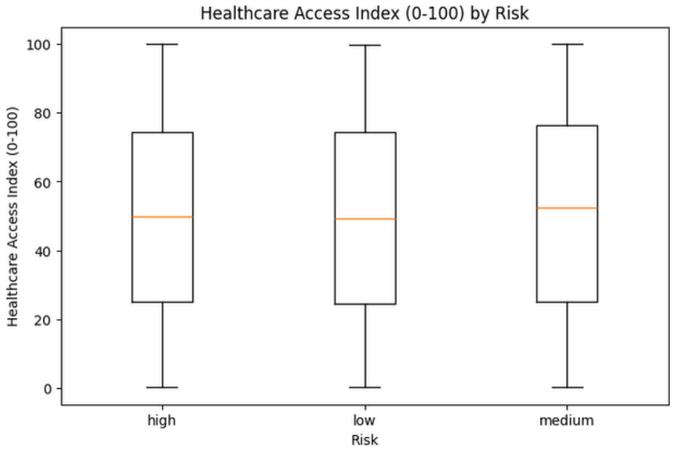
Post-label EDA

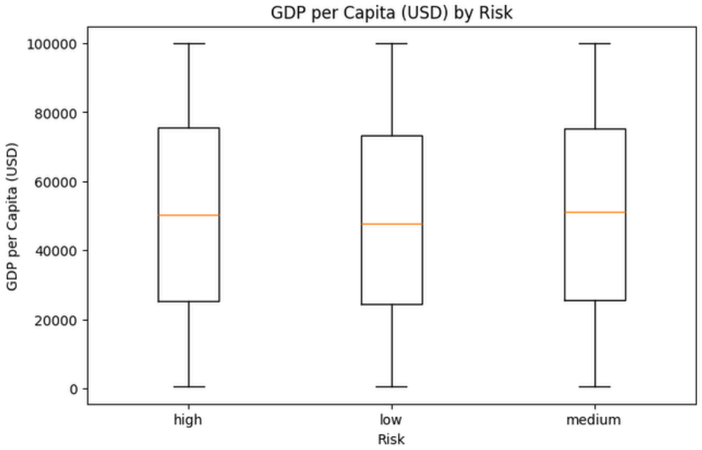
Risk Label











Post-label EDA

Contamination Label

For *Contaminated*, the classes are clearly imbalanced: about 2,450 “no” vs 550 “yes”, so only roughly 18–20% of samples are labeled contaminated. This means a naive model that always predicts “no” would already get around 80% accuracy, so we will have to report metrics like precision, recall, and use stratified splits or class-balanced evaluation to show value beyond that baseline.
These boxplots show that the contamination label is doing what we intended and that pH contributes less than the other indicators.
- For nitrate, lead, bacteria, and turbidity, the “yes” box is clearly shifted higher than the “no” box, and the medians for contaminated samples are above those for non-contaminated samples. This supports that contaminated sites tend to have systematically higher levels on all four indicators, not just random noise.
- The spreads overlap but “yes” tends to occupy the upper part of each variable’s range, which matches our rule of flagging rows with multiple unusually high values.
- For pH, the distributions for “yes” and “no” are very similar, with overlapping medians and ranges. That suggests pH is less discriminative for our contamination label in this dataset, probably because most samples fall within our relatively wide acceptable pH band.

Risk Label

For *Risk*, the classes are perfectly balanced at about 1,000 each for “low”, “medium”, and “high”, so a trivial baseline is 33% accuracy. This is ideal for our risk classifier, because models are not biased toward any one class and performance comparisons (accuracy, per-class recall) will be more informative.
The boxplots show that risk is not driven by any single context variable
- Access to clean water, sanitation coverage, healthcare index, and GDP all have very similar, overlapping distributions across low, medium, and high risk.
- This means our Risk label reflects a composite of disease burden and WASH factors rather than simply “low access = high risk” or “low GDP = high risk”.

Classification Results

Contaminated Label

```
========================================
Predictive Accuracy – Contaminated (kNN)
========================================

Stratified 10-Fold Cross Validation
kNN: accuracy = 0.89, error rate = 0.11
Precision (pos=yes) = 0.87
Recall    (pos=yes) = 0.47
F1        (pos=yes) = 0.61
----------------------------------------
========================================
Confusion Matrix
========================================
kNN (Stratified 10-Fold Cross Validation Results):
label     yes    no    Total    Recognition (%)
  yes     254   290     544              47
  no       39  2417    2456              98
```

kNN

```
========================================
Predictive Accuracy – Contaminated
========================================

Stratified 10-Fold Cross Validation
Decision Tree: accuracy = 0.75, error rate = 0.25
Precision (pos=yes) = 0.21
Recall    (pos=yes) = 0.14
F1        (pos=yes) = 0.17
----------------------------------------
========================================
Confusion Matrix
========================================
Decision Tree (Stratified 10-Fold Cross Validation Results):
label     yes    no    Total    Recognition (%)
  yes      78   466     544              14
  no      288  2168    2456              88
```

Decision Tree

```
========================================
Predictive Accuracy – Contaminated (RF)
========================================

Stratified 10-Fold Cross Validation
Random Forest: accuracy = 0.82, error rate = 0.18
Precision (pos=yes) = 0.37
Recall    (pos=yes) = 0.01
F1        (pos=yes) = 0.02
----------------------------------------
========================================
Confusion Matrix
========================================
Random Forest (Stratified 10-Fold Cross Validation Results):
label     yes    no    Total    Recognition (%)
  yes       7   537     544               1
  no       12  2444    2456             100
```

Custom Random Forest

Classification Results

Risk Label

==========
Predictive Accuracy – Risk (kNN)
==========

Stratified 10-Fold Cross Validation
kNN: accuracy = 0.33, error rate = 0.67
Precision (pos=high) = 0.32
Recall (pos=high) = 0.32
F1 (pos=high) = 0.32

==========
Confusion Matrix
==========
kNN (Stratified 10-Fold Cross Validation Results):

label	high	low	medium	Total	Recognition (%)
high	319	341	339	999	32
low	343	328	330	1001	33
medium	344	314	342	1000	34

kNN

==========
Predictive Accuracy – Risk
==========

Stratified 10-Fold Cross Validation
Decision Tree: accuracy = 0.33, error rate = 0.67
Precision (pos=high) = 0.33
Recall (pos=high) = 0.84
F1 (pos=high) = 0.47

==========
Confusion Matrix
==========
Decision Tree (Stratified 10-Fold Cross Validation Results):

label	high	low	medium	Total	Recognition (%)
high	841	72	86	999	84
low	854	83	64	1001	8
medium	865	68	67	1000	7

Decision Tree

==========
Predictive Accuracy – Risk (RF)
==========

Stratified 10-Fold Cross Validation
Random Forest: accuracy = 0.34, error rate = 0.66
Precision (pos=high) = 0.33
Recall (pos=high) = 0.71
F1 (pos=high) = 0.45

==========
Confusion Matrix
==========
Random Forest (Stratified 10-Fold Cross Validation Results):

label	high	low	medium	Total	Recognition (%)
high	713	243	43	999	71
low	702	262	37	1001	26
medium	721	248	31	1000	3

Custom Random Forest

Classification Results

Overall

kNN

Contaminated Label

kNN

Risk Label

Decision Tree

Conclusion

Summary

- Add feature engineering (e.g., ratios like contaminants per turbidity, interaction terms between WASH indicators) to capture non-linear relationships.
- Try more expressive models (e.g., gradient boosted trees or tuned random forests) with systematic hyperparameter search to better handle complex decision boundaries.
- Use techniques for class imbalance (re-weighting classes, SMOTE, threshold tuning) to improve detection of contaminated and high-risk cases.

Challenges

- Potential class imbalance makes it harder for standard classifiers to correctly identify the minority “contaminated” or “high-risk” label.
- Correlated or noisy features (e.g., overlapping WASH indicators) can confuse models and lead to overfitting without careful regularization or feature selection.
- Limited sample size and derived labels restrict how complex the model can be before it starts memorizing rather than generalizing.

Key components of the code

```python
def evaluate_knn_cv(X, y, feature_names, positive_label,
                    n_splits=10, n_neighbors=10,
                    stratify=True, random_state=0, shuffle=True):
    """Evaluate kNN with stratified k-fold CV.

    Returns dict with: feature_names, accuracy, error_rate,
    precision, recall, f1, labels, confusion_matrix.
    """
    # choose fold generator
    if stratify:
        folds = myevaluation.stratified_kfold_split(
            X, y, n_splits=n_splits,
            random_state=random_state, shuffle=shuffle
        )
    else:
        folds = myevaluation.kfold_split(
            X, n_splits=n_splits,
            random_state=random_state, shuffle=shuffle
        )

    all_y_true = []
    all_y_pred = []

    for train_idx, test_idx in folds:
        X_train = [X[i] for i in train_idx]
        y_train = [y[i] for i in train_idx]
        X_test  = [X[i] for i in test_idx]
        y_test  = [y[i] for i in test_idx]

        # normalize using train stats
        X_train_norm, min_vals, max_vals = min_max_normalize(X_train)
        X_test_norm, _, _ = min_max_normalize(X_test, min_vals, max_vals)

        knn = myclassifiers.MyKNeighborsClassifier(n_neighbors=n_neighbors)
        knn.fit(X_train_norm, y_train)
        y_pred = knn.predict(X_test_norm)

        all_y_true.extend(y_test)
        all_y_pred.extend(y_pred)

    all_y_true = np.array(all_y_true)
    all_y_pred = np.array(all_y_pred)
```

```python
def evaluate_random_forest_cv(X, y, feature_names, positive_label,
                              n_splits=10, n_trees=100,
                              max_features=None,
                              stratify=True, random_state=0, shuffle=True):
    """Run k-fold CV for a custom random forest on (X, y).

    Returns dict with: feature_names, accuracy, error_rate,
    precision, recall, f1, labels, confusion_matrix.
    """
    # choose folds
    if stratify:
        folds = myevaluation.stratified_kfold_split(
            X, y, n_splits=n_splits,
            random_state=random_state, shuffle=shuffle
        )
    else:
        folds = myevaluation.kfold_split(
            X, n_splits=n_splits,
            random_state=random_state, shuffle=shuffle
        )

    all_y_true = []
    all_y_pred = []

    for train_idx, test_idx in folds:
        X_train = [X[i] for i in train_idx]
        y_train = [y[i] for i in train_idx]
        X_test  = [X[i] for i in test_idx]
        y_test  = [y[i] for i in test_idx]

        rf = myclassifiers.MyRandomForestClassifier(
            n_trees=n_trees,
            max_features=max_features,
            random_state=random_state
        )
        rf.fit(X_train, y_train)
        y_pred = rf.predict(X_test)

        all_y_true.extend(y_test)
        all_y_pred.extend(y_pred)

    all_y_true = np.array(all_y_true)
    all_y_pred = np.array(all_y_pred)
```

```python
def evaluate_decision_tree_cv(X, y, feature_names, positive_label,
                              n_splits=10, stratify=True,
                              random_state=0, shuffle=True):
    """Run k-fold CV for a decision tree on (X, y).

    Args:
        X (list of list of obj): feature matrix (already subsetted).
        y (list of obj): labels.
        feature_names (list of str): names of the features in X (same order as columns).
        positive_label (obj): label to treat as the positive class for precision/recall/F1
        n_splits (int): number of folds.
        stratify (bool): whether to use stratified k-fold.
        random_state (int): seed for reproducibility.
        shuffle (bool): whether to shuffle before splitting.

    Returns:
        dict with keys:
            "feature_names", "accuracy", "error_rate",
            "precision", "recall", "f1",
            "labels", "confusion_matrix"
    """
    # choose fold generator
    if stratify:
        folds = myevaluation.stratified_kfold_split(
            X, y, n_splits=n_splits,
            random_state=random_state, shuffle=shuffle
        )
    else:
        folds = myevaluation.kfold_split(
            X, n_splits=n_splits,
            random_state=random_state, shuffle=shuffle
        )

    all_y_true = []
    all_y_pred = []

    for train_idx, test_idx in folds:
        X_train = [X[i] for i in train_idx]
        y_train = [y[i] for i in train_idx]
        X_test  = [X[i] for i in test_idx]
        y_test  = [y[i] for i in test_idx]

        dt = myclassifiers.MyDecisionTreeClassifier()
        dt.fit(X_train, y_train)
```

```python
# Load csv into table
water_pollution_disease_table = MyPyTable().load_from_file("input_data/water_pollution_disease.csv")
water_pollution_disease_table.pretty_print()
```

✓ 0.4s Python

```python
def stratified_kfold_split(X, y, n_splits=10, random_state=None, shuffle=False):
    """Split dataset into stratified cross validation folds.

    Args:
        X(list of list of obj): The list of instances (samples).
            The shape of X is (n_samples, n_features)
        y(list of obj): The target y values (parallel to X).
            The shape of y is n_samples
        n_splits(int): Number of folds.
        random_state(int): integer used for seeding a random number generator for reproducible results
        shuffle(bool): whether or not to randomize the order of the instances before creating folds

    Returns:
        folds(list of 2-item tuples): The list of folds where each fold is defined as a 2-item tuple
            The first item in the tuple is the list of training set indices for the fold
            The second item in the tuple is the list of testing set indices for the fold

    Notes:
        Loosely based on sklearn's StratifiedKFold split():
            https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html#sklearn.mode
    """
    n_samples = len(X)
    if n_samples != len(y):
        raise ValueError("X and y must have the same length")

    if n_splits < 2:
        raise ValueError("n_splits must be at least 2")

    indices = np.arange(n_samples)

    # Shuffle if needed
    if shuffle:
        rng = np.random.RandomState(random_state)
        rng.shuffle(indices)

    # find unique class labels
    unique_labels = []
    for label in y:
        if label not in unique_labels:
            unique_labels.append(label)

    # build per-class index lists
    label_indices = []
```

Contribution

Charlie Serafin

- Provide the dataset
- **Developed the first draft of the presentation slides and code**
- **Created repository on Github**

Kayla Vo

- Completed Project Porposal
- Designed all final source code
- Wrote technical report
- Made final presentation slides
- Performed interpretation and analysis