AI7101-001 Final Project - pump it

Team & Contribution

- **Qinrong Cui** & **Maosheng Li** : data preprocessing, visualization and final report.
- **Shengjie Zhu** & **Yuhang Liu**: model configuration and training.

Question Definition

- Goal
  - We are working on a project that uses data from Taarifa and the Tanzanian Ministry of Water to apply machine learning techniques for predicting whether a water pump is operational or not.
- significance
  - This is important because reliable access to water is critical, while manual inspections are costly and inefficient. A machine learning model can enable proactive decision-making by identifying high-risk pumps before they completely fail.

Structure of Project

- `notebook`
  - `pump_it_final_1.ipynb`: This is the main Jupyter Notebook for the project. It integrates data loading, preprocessing, feature engineering, model training, and evaluation.
- `src`
  - `data`
    - `train.csv`: Training dataset containing both features.
    - `train_label.csv`: Contains the target labels corresponding to `train.csv`.
  - `utils`
    - `utils.py`: Contains utility functions for data preprocessing and cleaning.
    - `encode_feature.py`: Defines preprocessing pipelines for categorical and numerical features.
- `requirements.txt`: Contains main libraries for whole project.

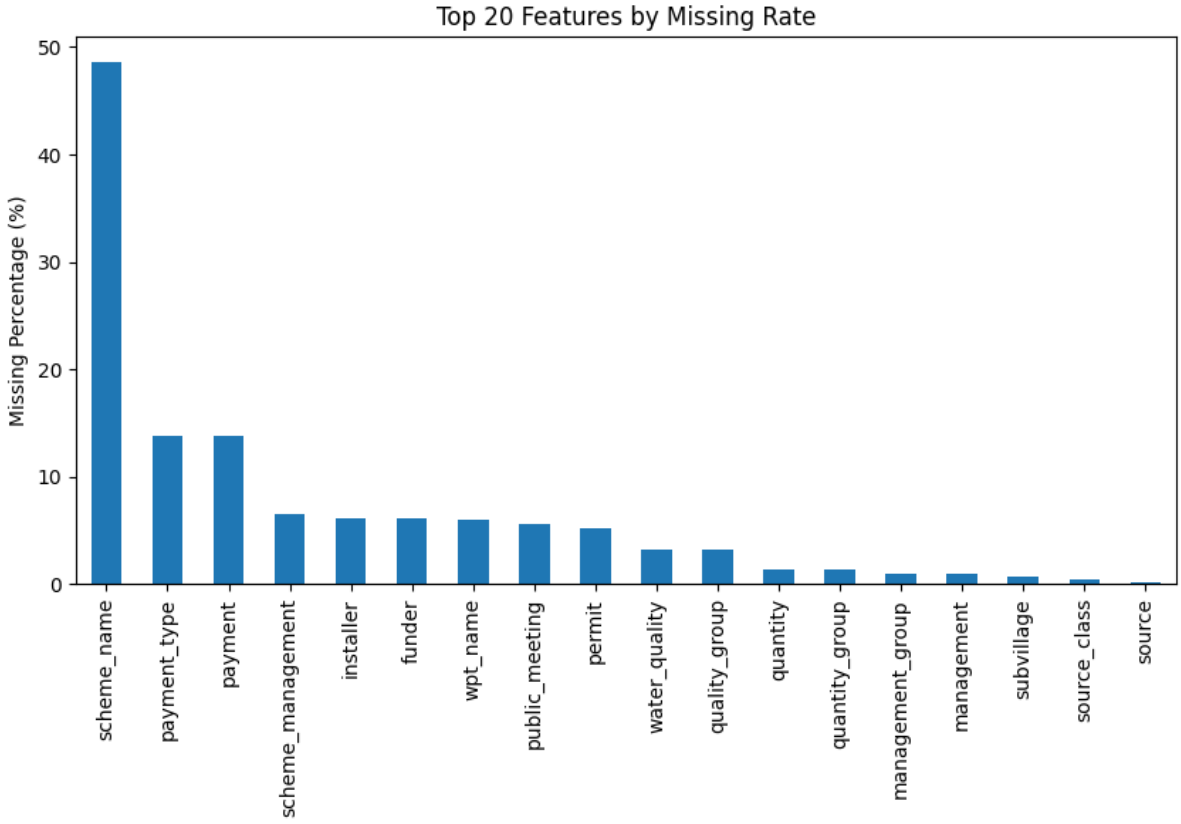
Work Flow

load data and libraries, and set seeds

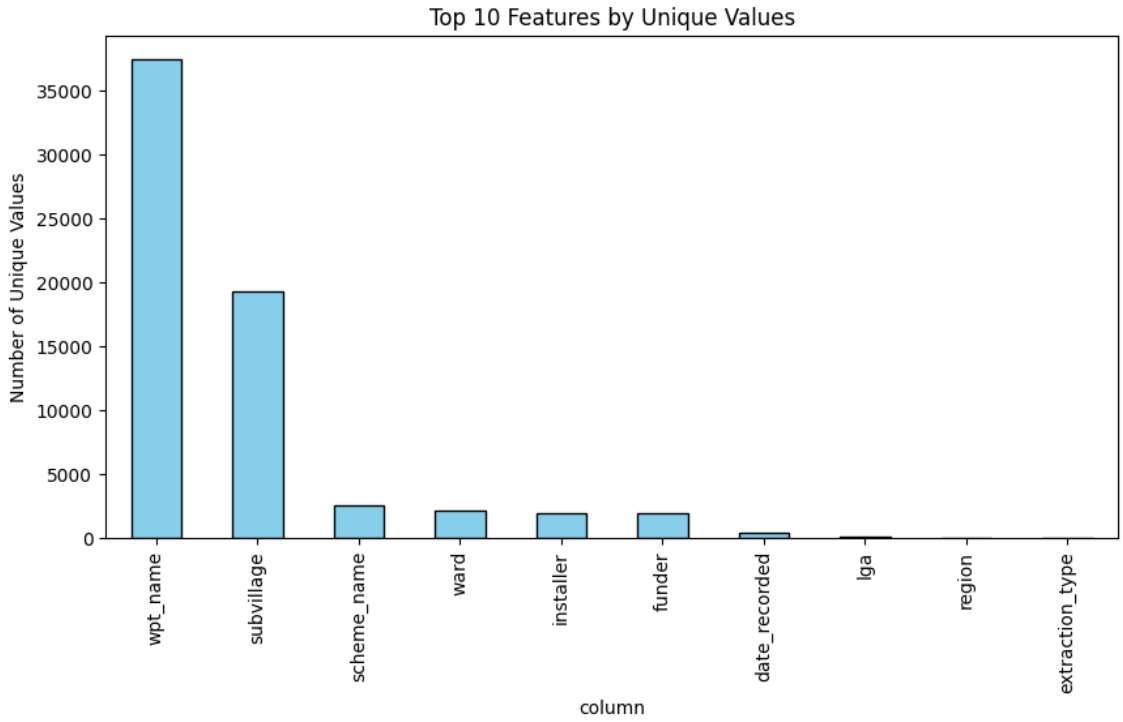
data processing

- **handle missing values**
  - normalize the different ways ‘missing’ can appear in text-like empty strings
  - different strategies to handle missing values
    - For **categorical columns** (of type `object` or `category`), if there are missing values, we replace them with `"Unknown"`

- For **numeric columns**, if there are missing values, we replace them with the `median` of that column.
- specially for `construction_year` **column**, if the value is `0`, we treat it as missing. Then we create a new indicator column, `construction_year_missing`, which is set to `1` if the year is `0` and `0` otherwise. Then we replace the `0` values in `construction_year` with `NaN`.

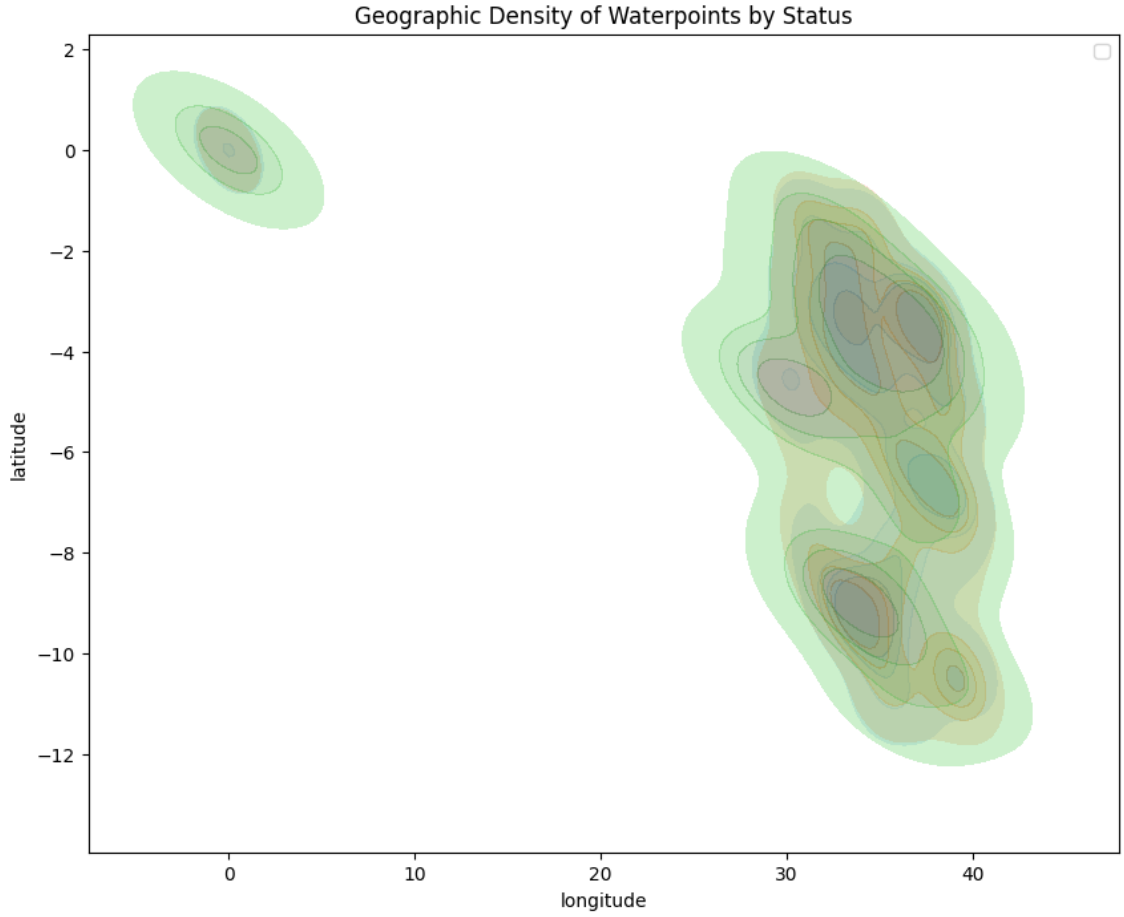


- **count unique values in each column to serve the subsequent feature encoding**
  - To avoid exploding the feature space, we first check the **cardinality** which is how many unique values live in each categorical column.



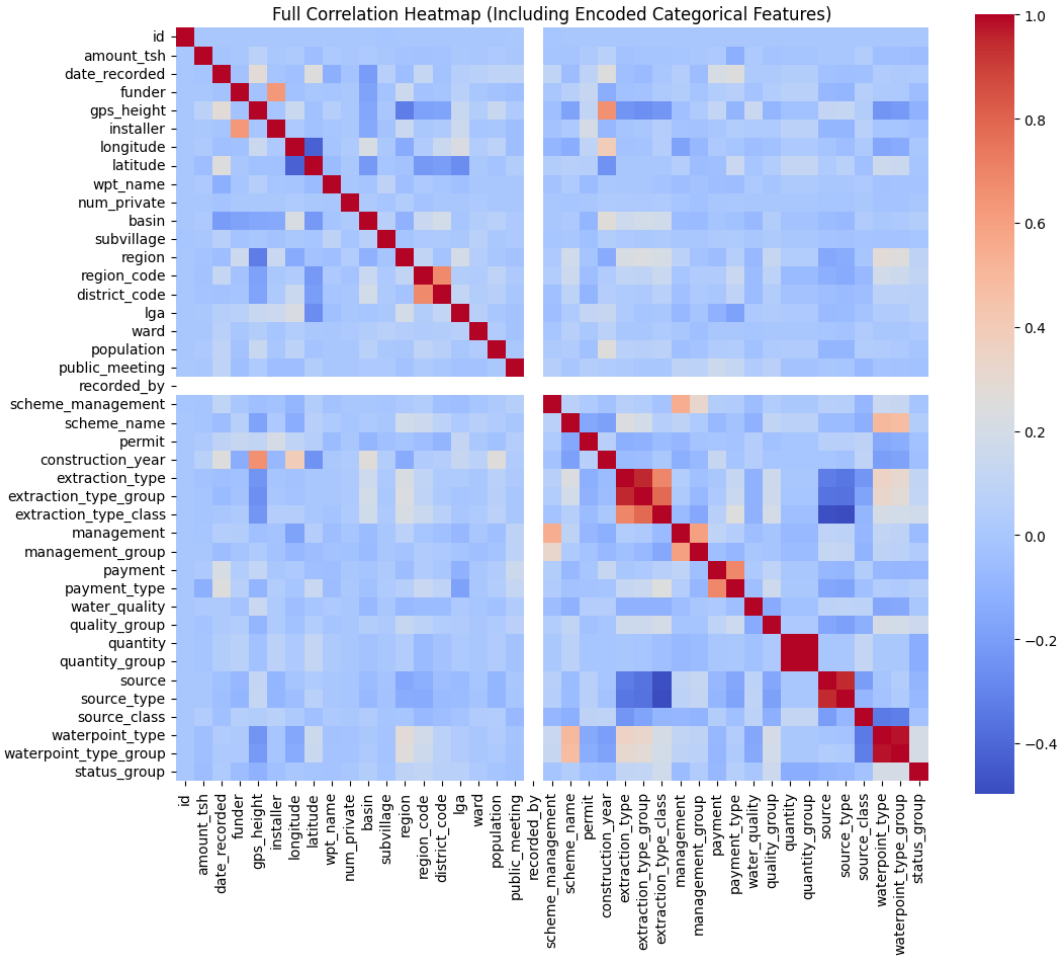
Geographic Density Plot

- To complement the scatterplot of longitude and latitude, we also plotted a geographic density map. This shows the density "hot zones" of waterpoints by status group.
- The KDE density plot shows that waterpoints are not randomly scattered, they actually cluster in specific geographic regions. Within these regions, the distribution of functional vs. non-functional wells is not uniform. This indicates that location plays a role in waterpoint condition, and therefore geographic features can provide valuable signals to a machine learning model.



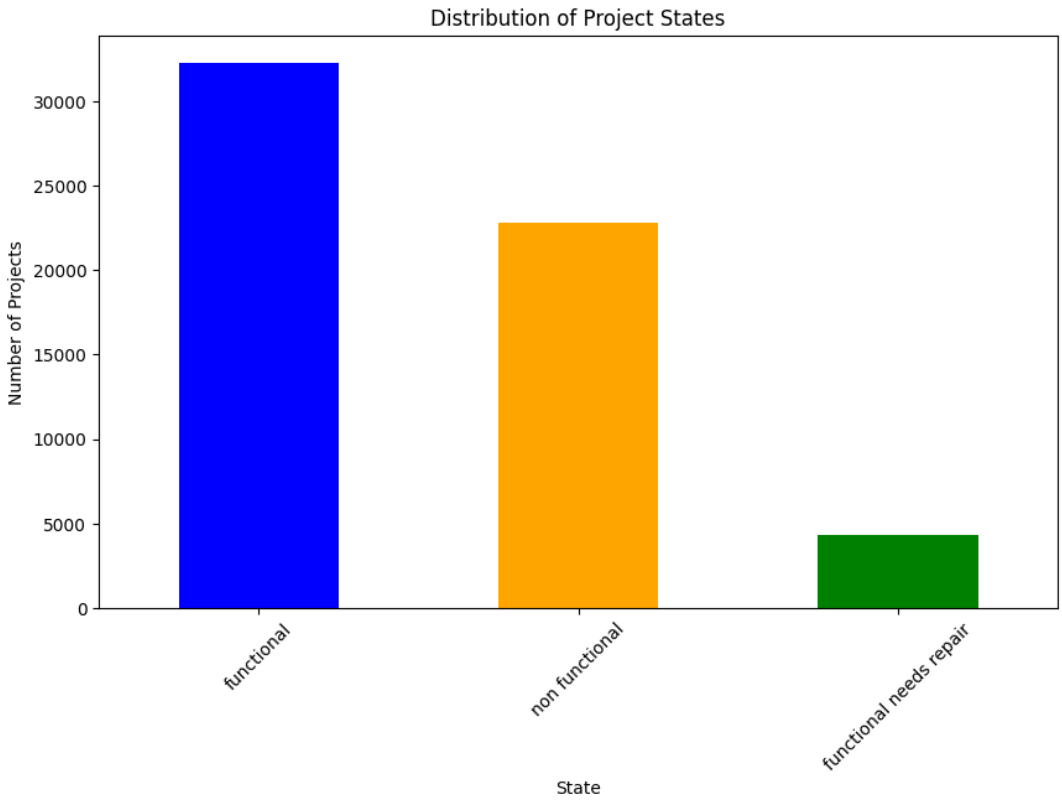
feature selection and encoding

- **feature selection**
  - **manual selection**: we remove columns that are useless for prediction or problematic distribution like `id` and recorded_by, and very high-cardinality text like `wpt_name` and `subvillage` , and we remove redundant 'grouped' variants—like `extraction_type_group` , `quantity_group`, and `waterpoint_type_group` —when the base columns already carry the signal.



  - `SelectKBest` : Besides relying on manual selection, we incorporated `SelectKBest` as an automated feature selection method to ensure that the most informative predictors were retained

- **feature encoding**
  - **date & bool:** we transform `date_recorded` into clean **year, month, and day** features and drop the original timestamp, and we also encoded bools into integer.
  - **labels**: map the label `status_group` to integers. We then take a quick look at the class distribution. That preview of imbalance is what motivates the dual metrics we'll use later, especially **F1**, so we don't just optimize for the majority class



- **object features**: we build a **single preprocessing object** designed for re-fitting inside cross-validation. It splits columns into three branches, in this way we avoid dimension explosion that could occur if we one-hot encode all features.
  - **Numeric branch** with a `StandardScaler` for models that benefit from comparable scales.
  - **Low and medium-cardinality categories** go through a *rare-category grouper* that keeps only the top `100` values and change the rest into `Other`, followed by a One-Hot Encoder.
  - **High-cardinality categories** use the same rare-category merge first, but then apply a simple *frequency encoding*, turning each category into its prevalence in the training data.

Model configuration and training

- **model configuration**
  - Logistic Regression

    Used as a simple and interpretable baseline. Showed relatively low accuracy and F1, indicating that linear decision boundaries were insufficient.
  - K-Nearest Neighbors

    Tested to capture nonlinear boundaries. Improved performance compared to Logistic Regression, but was sensitive to feature scaling and inefficient for larger datasets.
  - Gradient Boosting

    Strong ensemble model able to capture feature interactions. Performed better than Logistic Regression and KNN, but required careful hyperparameter tuning and still did not outperform Random Forest.
  - Random Forest

Delivered the best balance of accuracy and stability across folds. Became the main candidate model for further tuning.

- **evaluation metrics**
  - F1 score & accuracy

    Both were monitored under stratified 5-fold cross-validation to ensure consistent and unbiased performance estimation.

Final Results

model	accuracy_mean	accuracy_std	f1_mean	f1_std
LogisticRegression	0.721818	0.004697	0.513352	0.007256
RandomForest	0.793165	0.002464	0.660585	0.004970
GradientBoosting	0.782828	0.004046	0.644632	0.004866
KNN	0.776313	0.004718	0.661483	0.004638

Although Logistic Regression is the simplest and most interpretable model, the results clearly show that ensemble methods (Random Forest, Gradient Boosting) and KNN outperform it in both F1-score and accuracy. Therefore, if the goal is highest predictive performance, Random Forest would be the best choice due to its balance of F1-score (0.661) and high accuracy (0.793) with low variance.

Most importantly, in a real-life context, the model's ability to capture both functional and non-functional waterpoints (as reflected in its F1-score) is highly valuable. By identifying wells that are likely to fail, the model can help governments and NGOs prioritize maintenance, reduce unnecessary field inspections, and better allocate limited resources.