Pump it Up

MMAI 869 – Team Adelaide

Business Context

- 25 million people in Tanzania (43% of the country's population) do not have access to clean, portable water.

- Problem statement:

How can we optimize resources to provide clean, portable water to all communities?

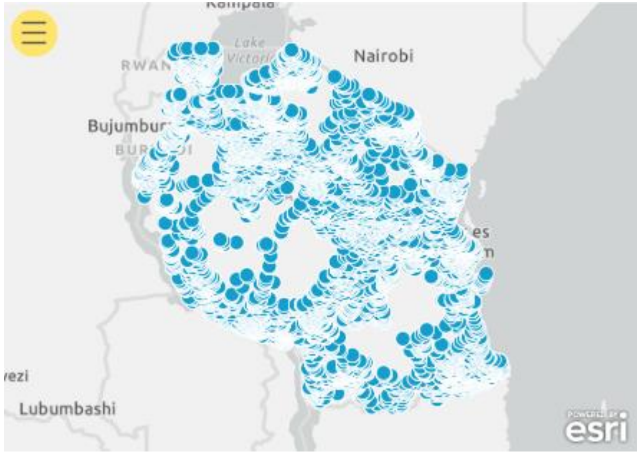
- Solution:

Classify water pumps as:

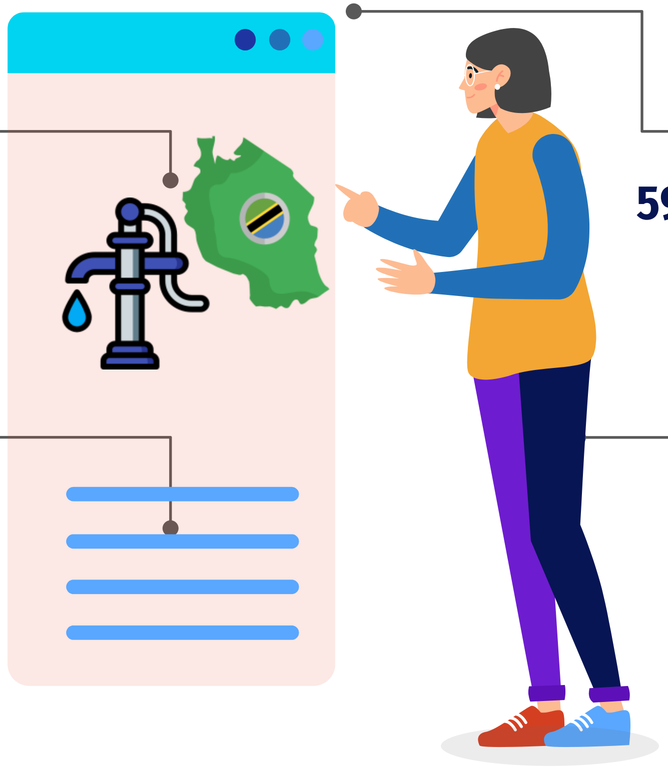
1) Functional
2) Non-functional
3) Functional but needs repair

This would help the government allocate resources effectively.

Non-Functional Waterpoints in Tanzania



Water pumps in Tanzania Profile



Multi-class problem

3 statuses: Functional, Function Needs Repair (FNR), Non-Functional

Missing Values

10 features contain missing values

Data Sets

59,400 instances for training, with inconsistent format in some features

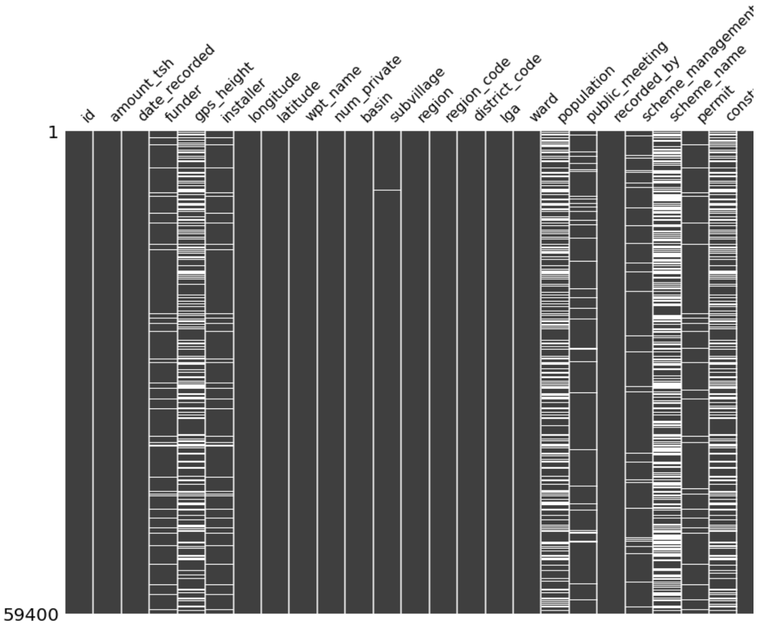
Features

39 features consisting of geographical, date/time, other attributes about each water pump

Key Observation

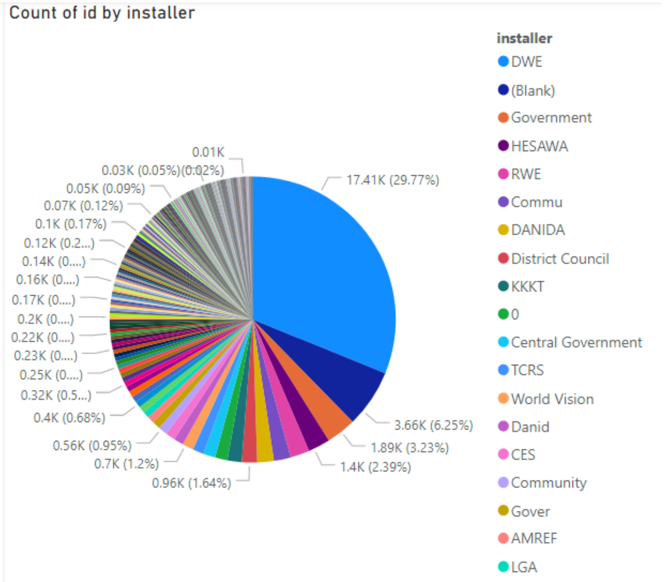
01

Missing Values

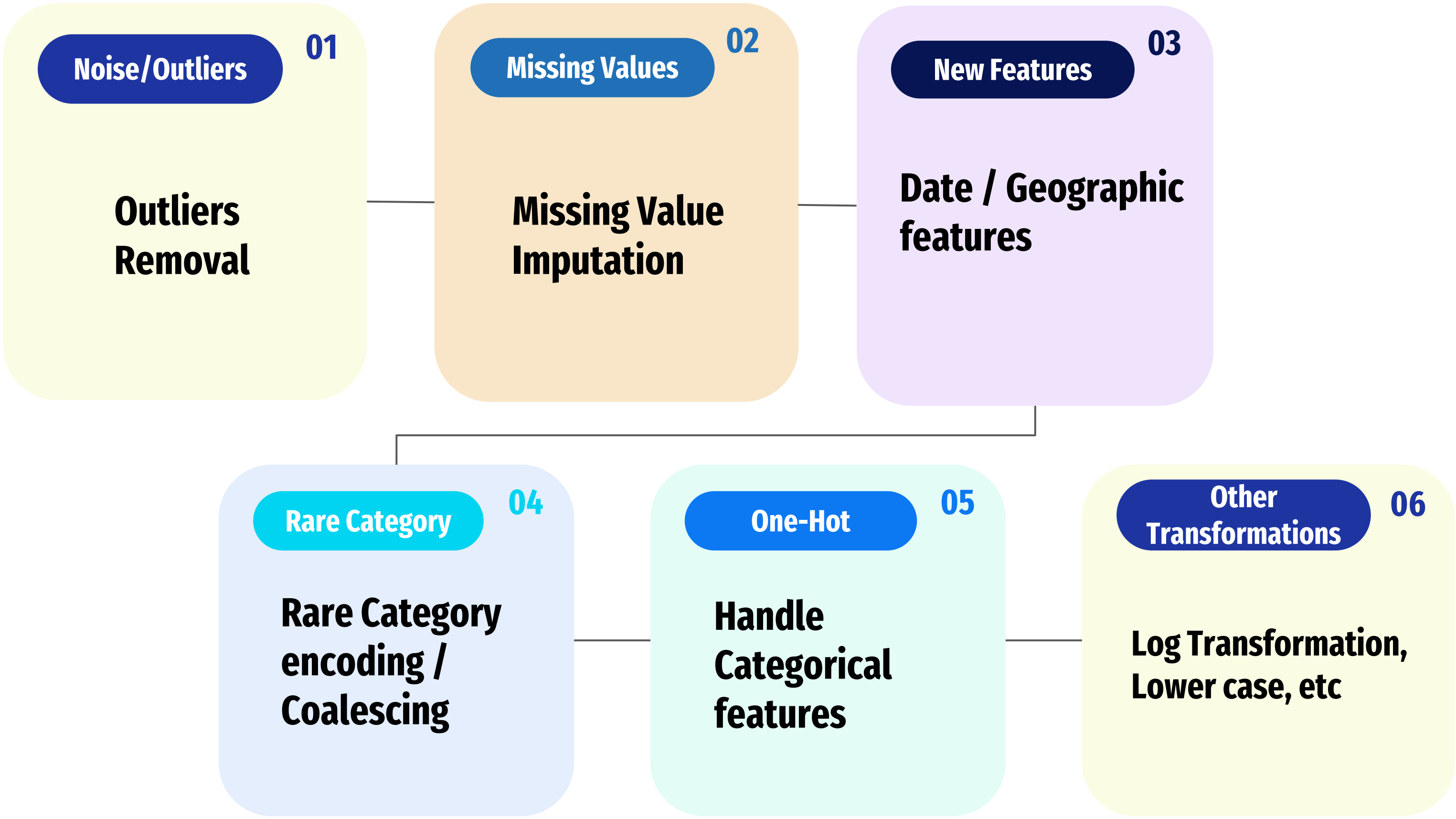


02

Lots of Categories



Pre-processing & Feature Engineering


Noise/Outliers
01
Outliers Removal
Missing Values
02
Missing Value Imputation
New Features
03
Date / Geographic features
Rare Category
04
Rare Category encoding / Coalescing
One-Hot
05
Handle Categorical features
Other Transformations
06
Log Transformation, Lower case, etc

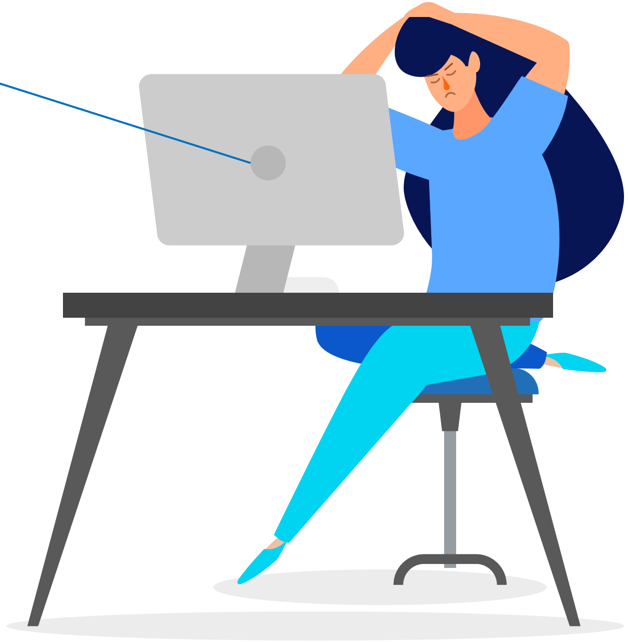
Key Steps

Class-based outliers

You won't be able to tell who's outliers until you include the class labels

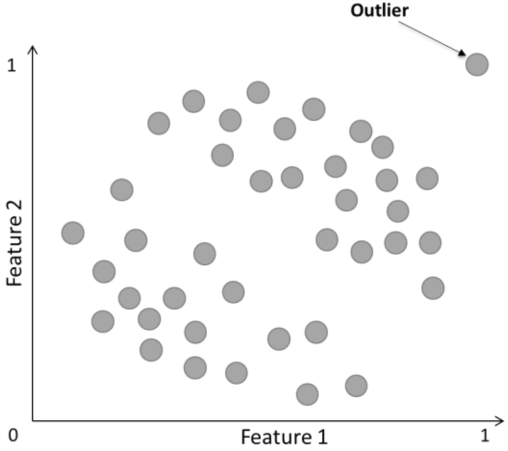
01

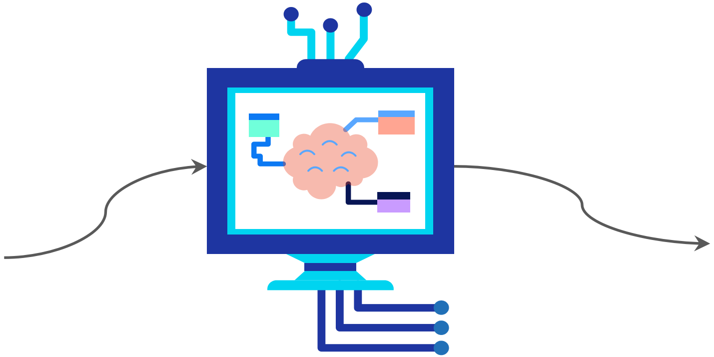
What we learned...



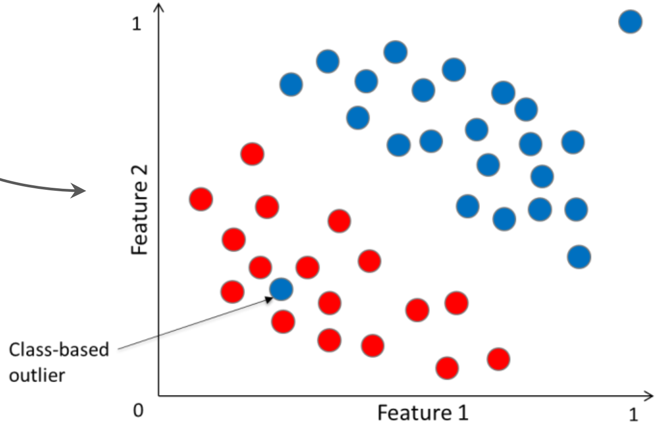
Reducing Outliers...

Before





After



Boosting algorithms are robust to general outliers. However, class-based outliers can cause them to overfit.

Class	Count of Class Outliers	% of Class Outliers
Functional	250	0.77%
Functional Needs Repair	443	10.26%
Non-Functional	1021	4.47%
TOTAL	1714	2.88%

Accuracy

Before: 80.02
After: 81.33

Key Steps

What we learned...

01

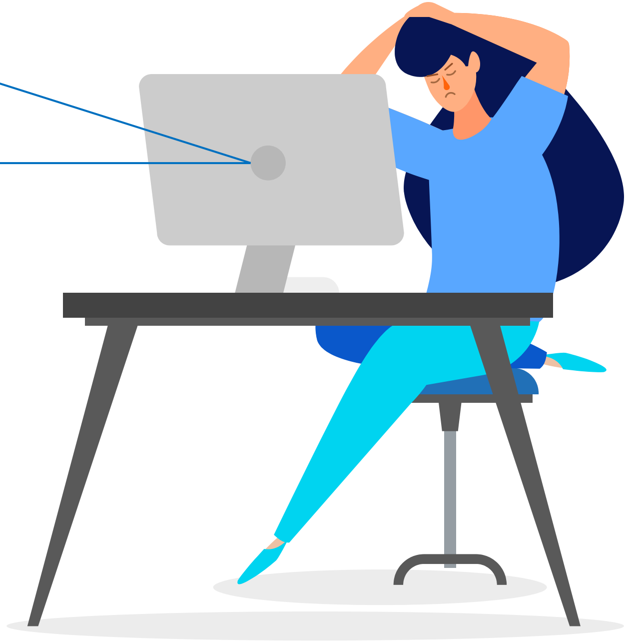
Class-based outliers

You won't be able to tell who's outliers until you include the class labels

02

Geocoding to fill missing values

Reversed geocoder library will take you longitude and latitude data and return the nearest town/city/sub-village



Cleaning Missing Values...

Before

Longitude	Latitude	Sub-village
-11.36	35.44	NaN
-10.34	-34.02	NaN
83.04	-79.11	NaN
24.78	40.22	NaN

After

Longitude	Latitude	Sub-village
-11.36	35.44	Subvillage1
-10.34	-34.02	Subvillage4
83.04	-79.11	Subvillage2
24.78	40.22	Subvillage3

Key Steps

What we learned...

01

Class-based outliers

You won't be able to tell who's outliers until you include the class labels

02

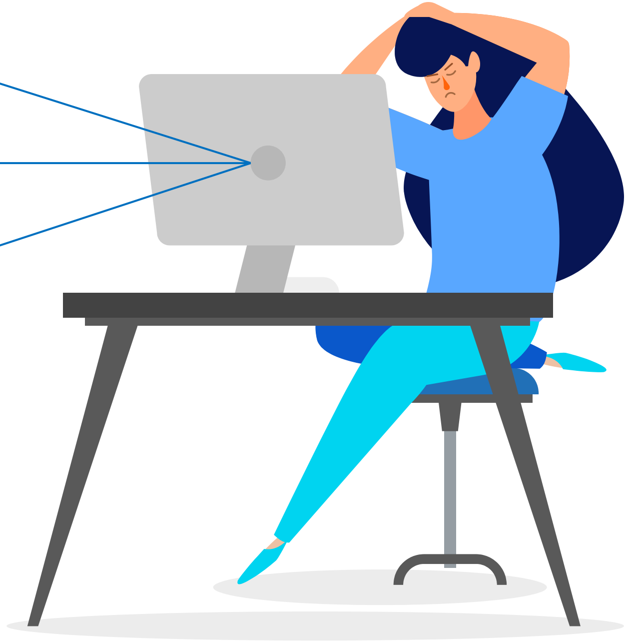
Geocoding to fill missing values

Reversed geocoder library will take you longitude and latitude data and return the nearest town/city/sub-village

03

Time and Date feature

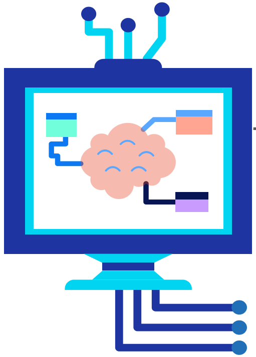
We created new features like Age / date recorded month from construction to recorded



Feature Creation...

Before

Date_recorded	Construction_year
2013-12-01	1990
2002-08-08	1992
2001-01-08	1998
1999-05-04	1986
2012-11-05	2001

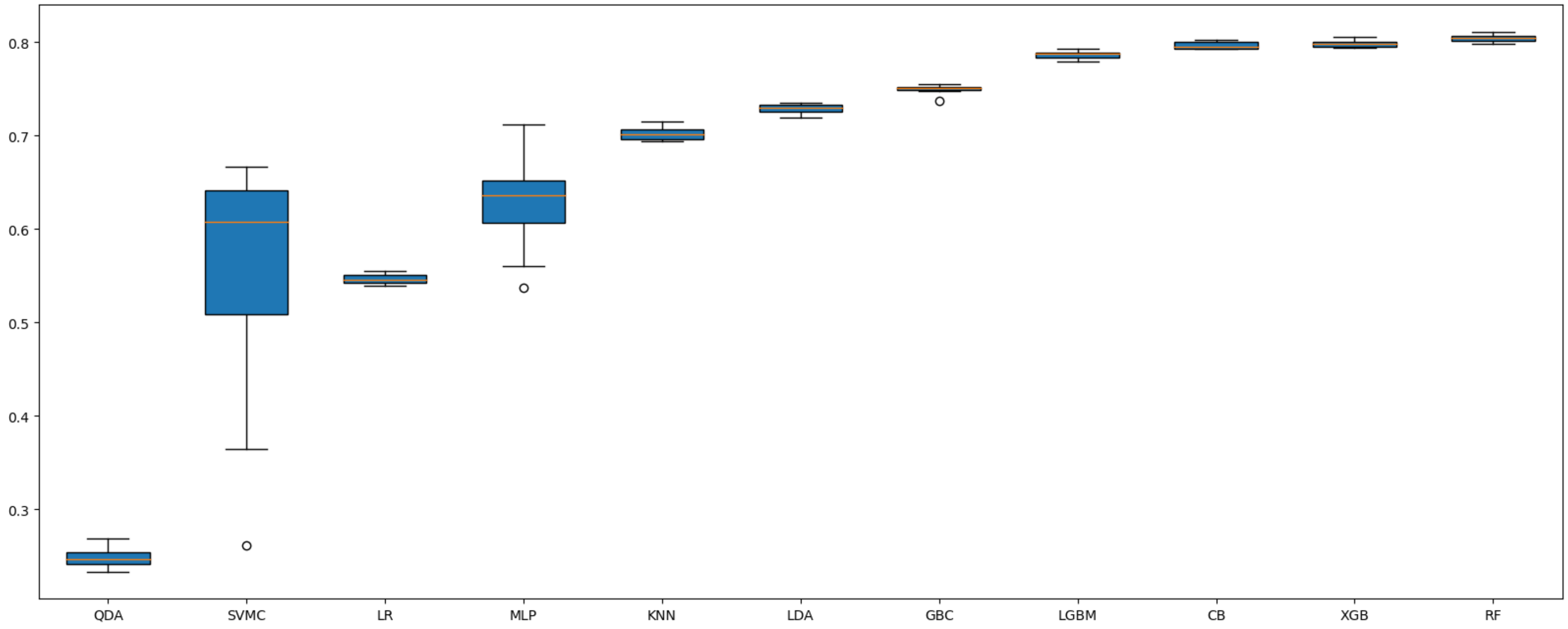


After

Age = date_recorded – construction_year
23
10
3
13
11

Modeling

Algorithm Comparison

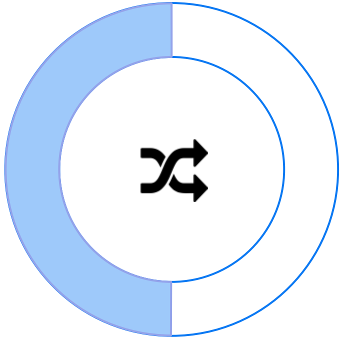


*RBF Support Vector Classification: Best Cross-validation score = 0.6193

Hyperparameter Tuning

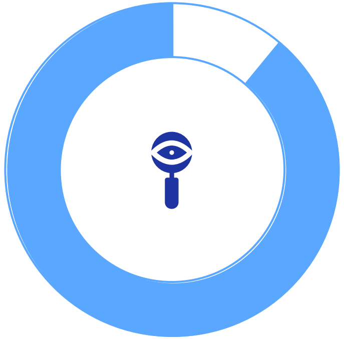
Initially we tried to start with Grid Search

- Took too much time & processing power



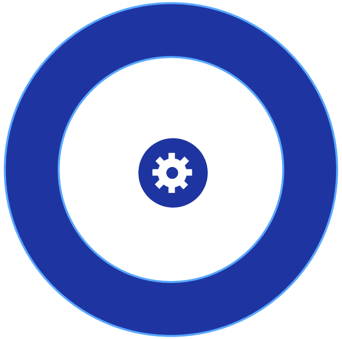
RandomizedSearchCV

Used random search to get a "better-than-default" set of hyperparameters



GridSearchCV

Used grid search and added +/- bounds to the hyperparameters found in random search



Manual Tuning

Varied one hyperparameter at a time and compare cross validation score

Accuracy	RandomizedSearchCV	GridSearchCV	Manual Tuning
Accuracy	0.7865	0.8022	0.8093

Ensemble

To improve results, we applied *StackingClassifier*

- Proof of Concept:

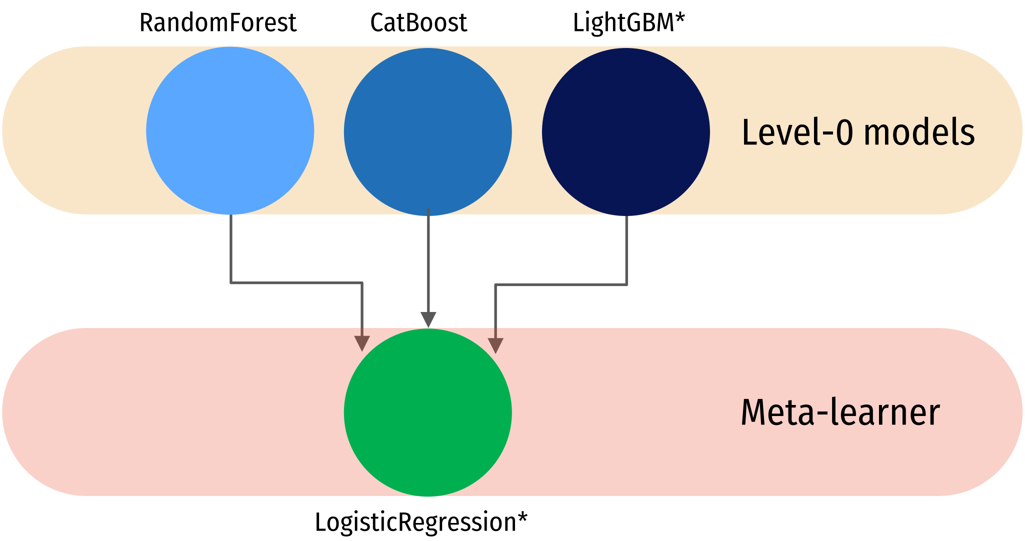
Models Stacked	*Results*
Random forest + Gradient boost	0.7075
RF + GB + LightGBM	0.7125

- Observations:

Additional models can increase accuracy

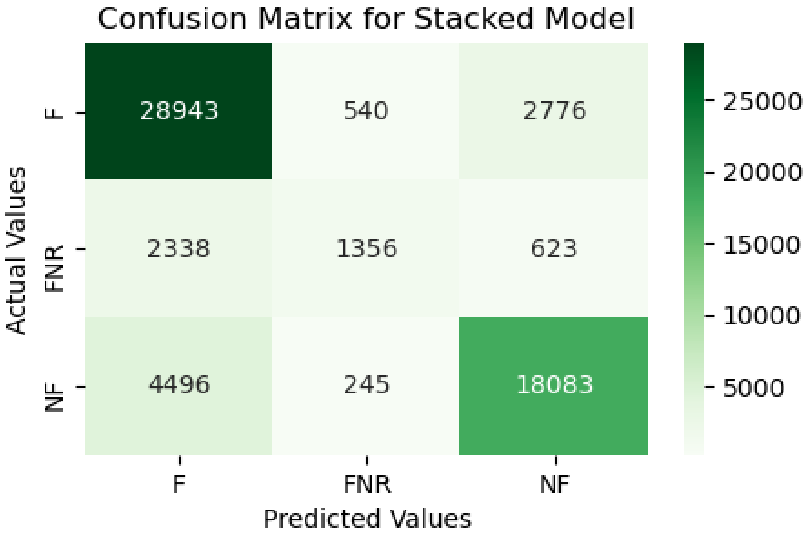
Takes very long with diminishing returns

- Construct stacked model using top performing models for accuracy gains



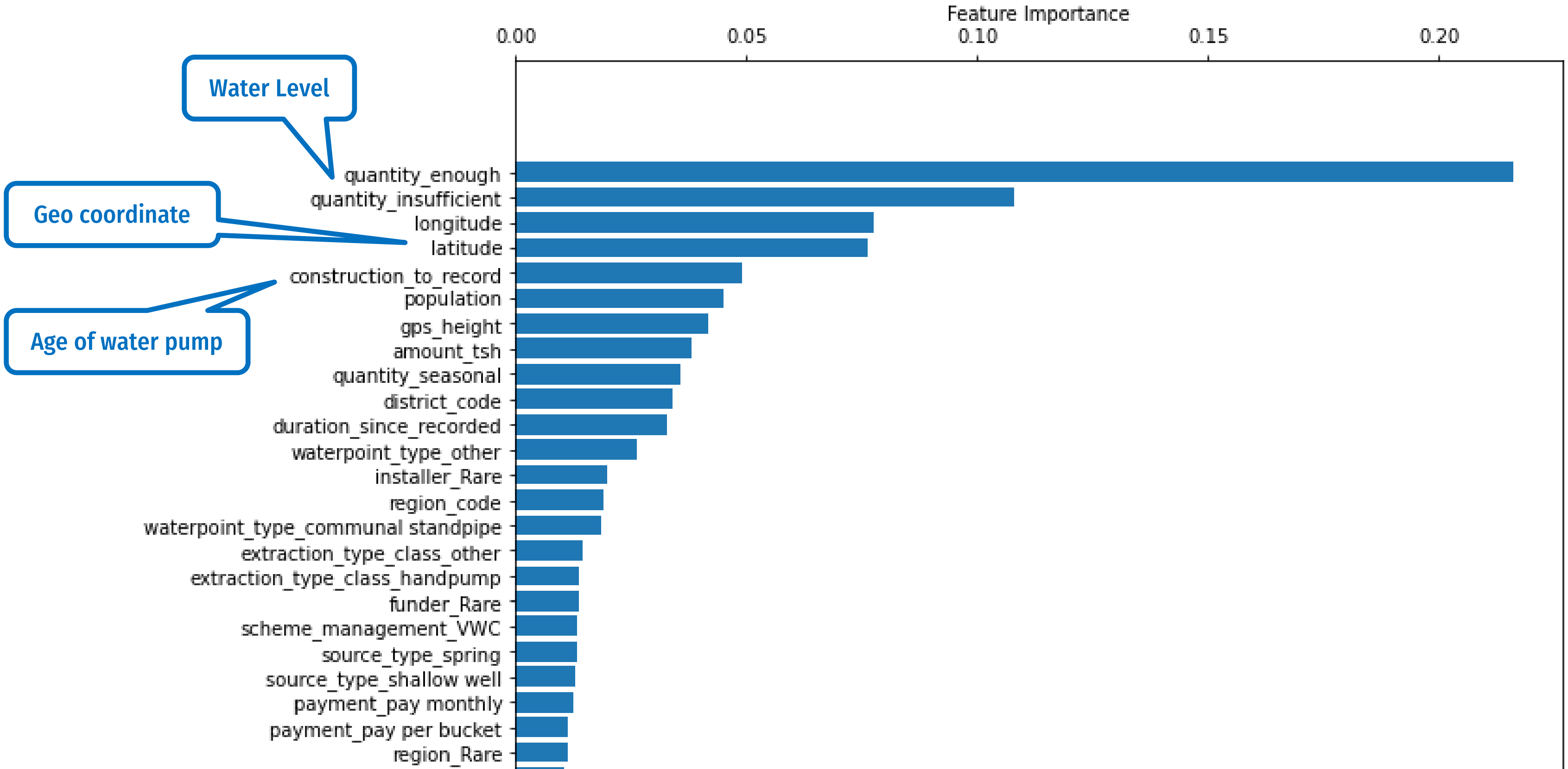
DrivenData Results... ***0.8191***

Confusion Matrix & Metrics



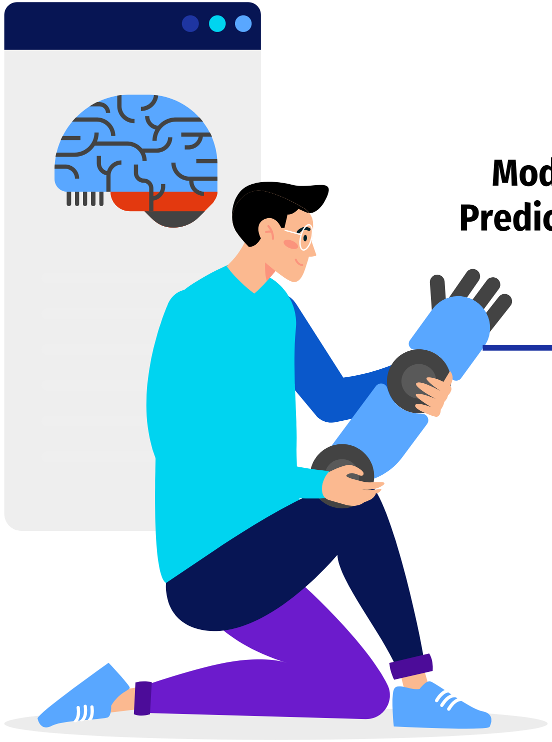
	precision	recall	f1-score	support
functional	0.81	**0.90**	0.85	32259
funct. needs repair	0.63	**0.31**	0.42	4317
non functional	**0.84**	0.79	0.82	22824
accuracy			0.81	59400
macro avg	0.76	0.67	0.70	59400
weighted avg	0.81	0.81	0.81	59400

Feature Importance



Application

Model Use



Model Prediction

Functional
Improve access of clean water to communities

Functional Needs Repair
Plan Maintenance and Reduce labor cost

Non-Functional
Repair or remove waterpoint, Reallocate Resources

OBJECTIVE

More people with access to clean water (>>43%)

More functional waterpoints

Lower labor cost

Financial Impact

Cost Matrix (USD cost)

Actual \ Predicted	F	FNR	NF
	0	200	200
FNR	6200	200	0
NF	6200	0	200

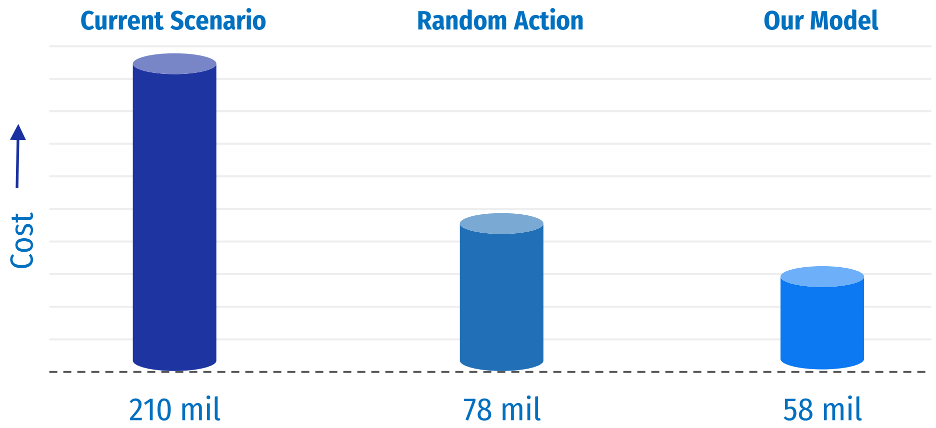
Labor Cost = 200 USD / wp

Govt Healthcare cost = 6200 USD / wp

Cost Matrix (Lives lost)

Actual \ Predicted	F	FNR	NF
F	0	0	0
FNR	0.7	0	0
NF	0.7	0	0

Avg Infant deaths per NF wp = 0.7



Current scenario

- Costs USD $210 million
- Costs 23,900 lives of infants

Random action

- Save USD $132 million
- Save 15,994 lives

Our model

- **Saves USD $152 million**
- **Saves 17,929 lives**

Lessons Learned & Next Steps

Lessons Learned

Approach

- How to effectively code as a group: balance between many approaches vs splitting sections

- Prioritize objectives in big projects

ML Knowledge

- Significance of Feature Engineering over other techniques

- Effectiveness of Boosting models

Next Steps

Pre-processing

- Seasonality, math transformations
- Over-sampling by SMOTE
- Cleaning of categorical features
- Class distributions for better coalescing
- Supervised techniques for missing value imputation
- Feature Selection using Recursive Feature Elimination

Modeling

- Explore models in the H2O package
- Use packages such as Optuna for tuning
- Experiment with a 2-Class model

Code Refactoring

- Create functions for all steps to help sharing and reusing of code among team members

Questions?

Appendix

Noise/Outliers 01

Outliers Removal

Such as:
- IQR capping
- Outliers removal
- Z-score

Missing values 02

Missing Value Imputation

Also:
- Categorical: mode/mean/median
- Numeric: mean/median

Normalization 03

Not Required

All models are tree-based

New features 04

Date / geographic features

Also:
- KMeans clustering to classified data based on longitude and latitude data
- Calculate haversine distance

Rare category 05

Handle categorical features with so many unique values

- Funder
- installer
- Region
- Sub-village

One-hot 06

Increase dimensionality for good

- Payment
- Funder
- Installer

Other Transformation 07

Log Transformation , Lower case ,etc

Logarithm (population + 1)

- The **installer** feature - spelling mistakes, wrong abbreviations and inconsistent capitalization

Converted data to lowercase. Reduced number of mistakes and applied grouping through simple rules

- Many outlier values of **latitude** and **longitude**

Replaced with the median values of the corresponding region_code

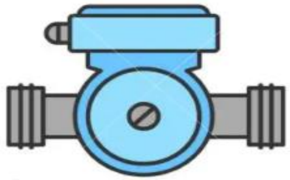
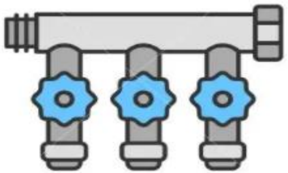
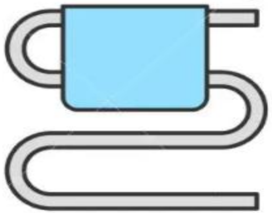
- Data contains features with very similar categories

Delete *scheme_management, quantity_group, water_quality, payment_type, extraction_type, waterpoint_type_group, region_code*

- Missing values in **public_meeting** and **permit**

Replaced with median values. Adopted similar technique for **subvillage** and **scheme_name**

- The features **date_recorded** and **recorded_by** were deleted because their values were meaningless.

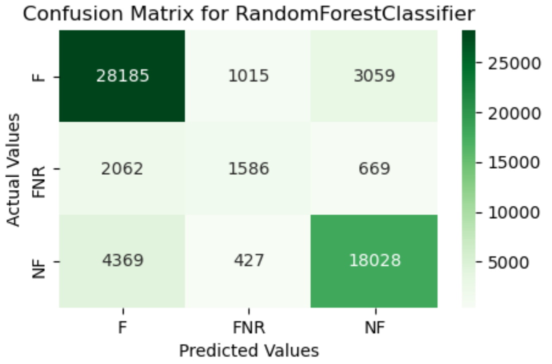
Reverse Geocoding

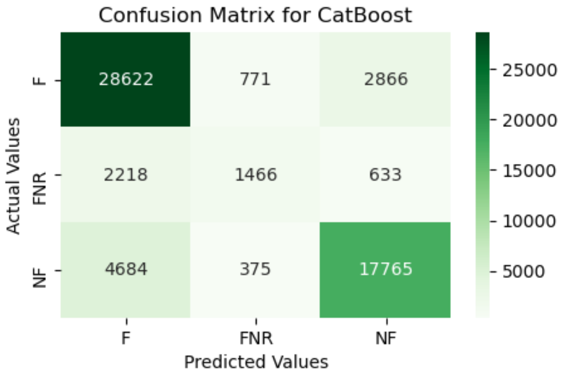
```
import reverse_geocoder as rg
def geocoder(data):
  '''
  input: dataframe containing Latitude(x) and Longitude(y) coordinates
  output: JSON data containing info on available building or street names.
  '''
  coordinates = tuple(data[['latitude','longitude']].values[3])
  #data[['latitude','longitude']].values
  results = rg.search(coordinates) # default mode = 2
  return results
```

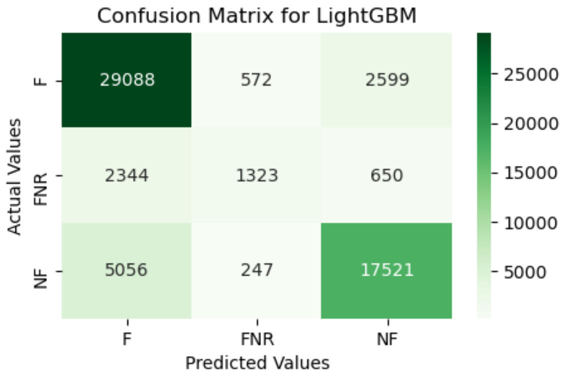
```
[ ] geocoder(data_x)
```

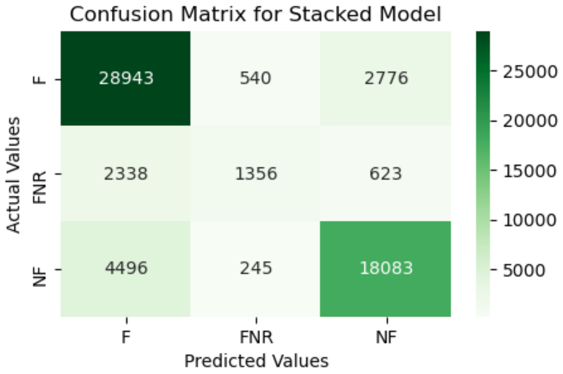

```
[{'lat': '-11.36667',
  'lon': '38.41667',
  'name': 'Masuguru',
  'admin1': 'Mtwara',
  'admin2': '',
  'cc': 'TZ'}]
```

Confusion Matrices (cross-val-predict)

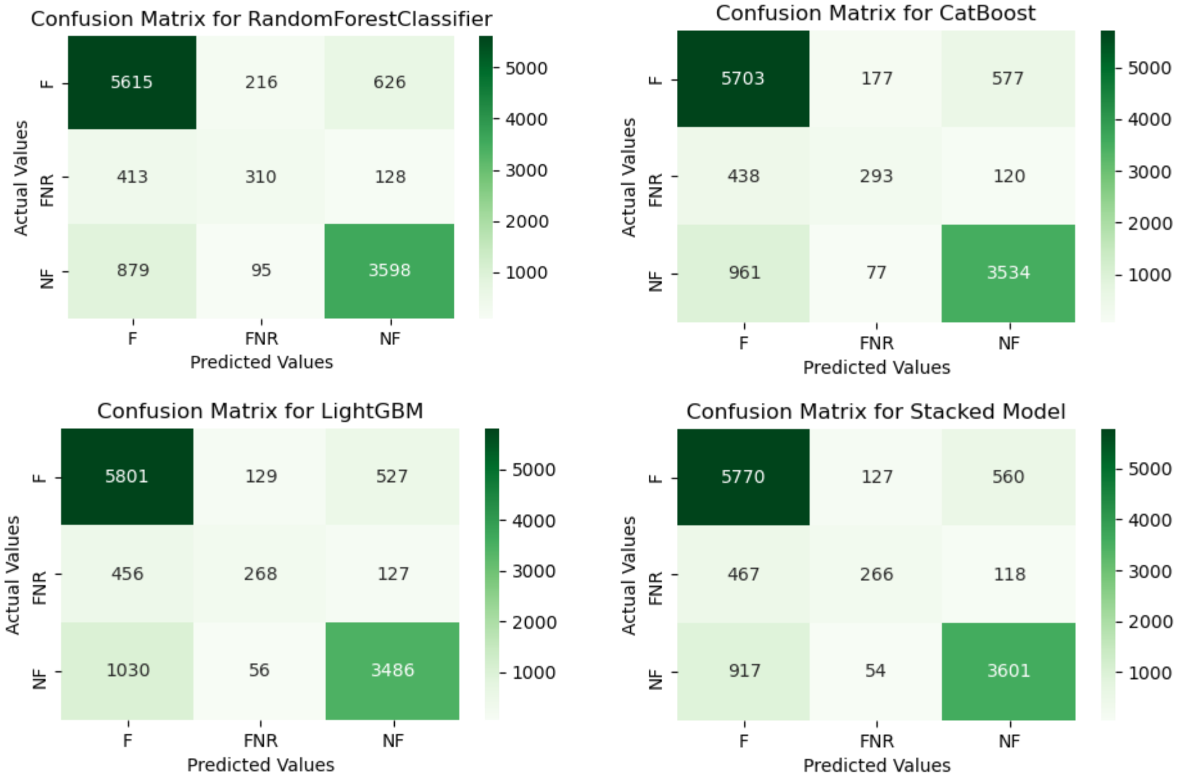








Confusion Matrices (split-train-test)



Example Scores

Model	Test Accuracy
RandomForest (default)	80.96
XGBoost (default)	80.02
Stack (RF + Gradient Boost)	70.75
Stack (RF, Gradient boosting, LightGBM)	71.25
CatBoost (tuned)	81.12
Stack (RF, LightGBM, CatBoost – all tuned)	81.91
RandomForest – H2O module	81.95
RandomForest – H2O module, tuned	82.01

Feature Descriptions

- amount_tsh – Total static head (amount water available to waterpoint)
- date_recorded – The date the row was entered
- funder – Who funded the well
- gps_height – Altitude of the well
- installer – Organization that installed the well
- longitude – GPS coordinate
- latitude – GPS coordinate
- wpt_name – Name of the waterpoint if there is one
- public_meeting – True/False
- recorded_by – Group entering this row of data
- scheme_management – Who operates the waterpoint

- num_private – No information
- basin – Geographic water basin
- subvillage – Geographic location
- region – Geographic location
- region_code – Geographic location (coded)
- district_code – Geographic location (coded)
- lga – Geographic location
- ward – Geographic location
- population – Population around the well