Tanzanian Water Well Project

Problem Statement:

Tanzania, as a developing country, struggles with providing clean water to its population of over 57,000,000. There are many waterpoints already established in the country, but some are in need of repair while others have failed altogether.

For this project I need to build a classifier to predict the condition of a water well, using information about the sort of pump, when it was installed, etc. The final model will classify the condition of water wells into 3 categories namely

functional - the waterpoint is operational and there are no repairs needed

functional needs repair - the waterpoint is operational but needs repairs

non-functional - the waterpoint is not operational

By predicting the pumps which are functional but needs repair, decreases the overall cost for the Tanzanian Ministry of Water. Which can improve the maintenance operations of the water pumps and make sure that clean, potable water is available to communities across Tanzania.

Plan:

1. Understanding Data
1. Cleaning and Exploring Data
1. Preparing Data to Modeling
1. Ternary Target Modeling
1. Visualizations
1. Conclusions
1. Future work

Data

The data for this project comes from the Taarifa waterpoints dashboard, which aggregates data from the Tanzania Ministry of Water.

You may find and download the data here https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/23/ (https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/page/23/) after signing up for the competition.

In this project, I will use train set and train label set (contains the status of every Pump Observatin in the training dataset).

Data Preprocessing

In [1]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
import random
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

# Sklearn packages
from sklearn.preprocessing import OneHotEncoder, StandardScaler, MinMaxS
caler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.compose import ColumnTransformer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix, classification_report
from xgboost import XGBClassifier
from catboost import CatBoostClassifier, Pool
```

In [2]:
```
# Importing the training set values and set id as an index to train set
X_train_val = pd.read_csv('./data/train.csv', index_col='id')

# Importing the training set labels data
y_train_val = pd.read_csv('./data/train_val_target.csv', index_col='id')

# Declare target
TARGET = 'status_group'

# Merge X_train_val and y_train_val
train_val = pd.merge(X_train_val, y_train_val, left_index = True, right_
index = True)
train_val.head()
```

Out[2]:

	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_na
id								
69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	n
8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Zaha
34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	k Mahu
67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Zaha Nanyun
19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Shu

5 rows × 40 columns

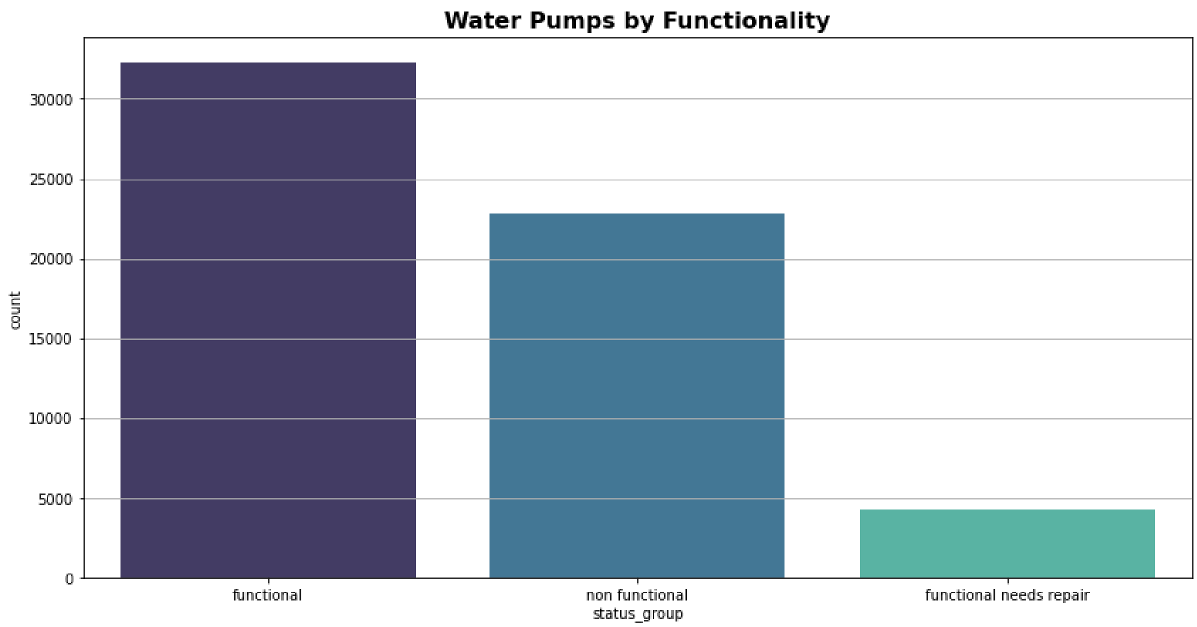
Explore target variable

In [3]:
```
train_val.status_group.value_counts(normalize = True)
```

Out[3]:
```
functional                 0.543081
non functional             0.384242
functional needs repair    0.072677
Name: status_group, dtype: float64
```

Exploratory analysis indicates that `functional` has the most of the records in `status_group` . The distribution of the functionality of the water pumps can be seen below:

In [4]:
```
plt.figure(figsize = (14,7))
plt.title("Water Pumps by Functionality",fontsize = 16, fontweight = 'bo
ld')
plt.grid(True)
sns.countplot(x = y_train_val['status_group'], data = X_train_val, palet
te = "mako");
```



By looking at the distribution of water pumps I can see that there is a clear class imbalance, however, for this research no changes to handle the class imbalance will not be made.

For the future work I might use reasampling technique to deal with this unbalanced dataset. This technique consists of removing samples from the majority class (under-sampling) and/or adding more examples from the minority class (over-sampling).

In [5]:
```
# Check attributes of all features
train_val.info();
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59400 entries, 69572 to 26348
Data columns (total 40 columns):
amount_tsh               59400 non-null float64
date_recorded            59400 non-null object
funder                   55765 non-null object
gps_height               59400 non-null int64
installer                55745 non-null object
longitude                59400 non-null float64
latitude                 59400 non-null float64
wpt_name                 59400 non-null object
num_private              59400 non-null int64
basin                    59400 non-null object
subvillage               59029 non-null object
region                   59400 non-null object
region_code              59400 non-null int64
district_code            59400 non-null int64
lga                      59400 non-null object
ward                     59400 non-null object
population               59400 non-null int64
public_meeting           56066 non-null object
recorded_by              59400 non-null object
scheme_management        55523 non-null object
scheme_name              31234 non-null object
permit                   56344 non-null object
construction_year        59400 non-null int64
extraction_type          59400 non-null object
extraction_type_group    59400 non-null object
extraction_type_class    59400 non-null object
management               59400 non-null object
management_group         59400 non-null object
payment                  59400 non-null object
payment_type             59400 non-null object
water_quality            59400 non-null object
quality_group            59400 non-null object
quantity                 59400 non-null object
quantity_group           59400 non-null object
source                   59400 non-null object
source_type              59400 non-null object
source_class             59400 non-null object
waterpoint_type          59400 non-null object
waterpoint_type_group    59400 non-null object
status_group             59400 non-null object
dtypes: float64(3), int64(6), object(31)
memory usage: 21.1+ MB
```

There are null values in the features which are needed to be handled for better training. Also, I will need to drop some of the columns so I can run my models easier.

In [6]:
```
# Get a feel for the unique values
train_val.nunique()
```

Out[6]:
```
amount_tsh                  98
date_recorded              356
funder                    1897
gps_height                2428
installer                 2145
longitude                57516
latitude                 57517
wpt_name                 37400
num_private                 65
basin                        9
subvillage               19287
region                      21
region_code                 27
district_code               20
lga                        125
ward                      2092
population                1049
public_meeting               2
recorded_by                  1
scheme_management           12
scheme_name               2696
permit                       2
construction_year           55
extraction_type             18
extraction_type_group       13
extraction_type_class        7
management                  12
management_group             5
payment                      7
payment_type                 7
water_quality                8
quality_group                6
quantity                     5
quantity_group               5
source                      10
source_type                  7
source_class                 3
waterpoint_type              7
waterpoint_type_group        6
status_group                 3
dtype: int64
```

Some features have a lot of unique values.

For this project, I will drop all the columns with more than 21 unique values.

However, firstly, I want to look closer at some of the columns to see which of them can be dropped because of irrelevant information they contain.

Feature Exploration

To decide which columns to drop, I will need to check which features have similar representation of data and check number of unique values they contain. Columns with repetitive information and more than 21 unique values will be dropped.

Also, I will fill Null/NaN values for categirical and numerical columns.

For my future work,however, I will need to make a further feature exploration to find unnecessary or wrong values.

In [7]:
```
train_val['amount_tsh'].value_counts()
```

Out[7]:
```
0.0         41639
500.0        3102
50.0         2472
1000.0       1488
20.0         1463
            ...
8500.0          1
6300.0          1
220.0           1
138000.0        1
12.0            1
Name: amount_tsh, Length: 98, dtype: int64
```

Total static head (`amount_tsh`) column mostly consists of zero values so I will drop it. Also, there is another column `num_private` which consists mostly of zeroes so, I will drop it too.

In [8]:
```
train_val['recorded_by'].value_counts()
```

Out[8]:
```
GeoData Consultants Ltd    59400
Name: recorded_by, dtype: int64
```

`recorded_by` column has only 1 unique value so I will drop this column; `wpt_name` and `scheme_name` coulmns are irrelevent for my research so I can drop both of them.

Also, I will drop `source_type` , `source_class` , `quantity_group` , `quality_group` , `extraction_type` , `extraction_type_class` , `waterpoint_type_group` , `region_code` , `payment_type` columns because they keep the same or similar information to other column(s) and usually have less detailed information compare to others.

In [9]:
```
train_val['management'].value_counts()
```

Out[9]:
```
vwc                 40507
wug                  6515
water board          2933
wua                  2535
private operator     1971
parastatal           1768
water authority       904
other                 844
company               685
unknown               561
other - school         99
trust                  78
Name: management, dtype: int64
```

In [10]:
```
train_val['management_group'].value_counts()
```

Out[10]:
```
user-group    52490
commercial     3638
parastatal     1768
other           943
unknown         561
Name: management_group, dtype: int64
```

In [11]:
```
train_val['scheme_management'].value_counts()
```

Out[11]:
```
VWC                 36793
WUG                  5206
Water authority      3153
WUA                  2883
Water Board          2748
Parastatal           1680
Private operator     1063
Company              1061
Other                 766
SWC                    97
Trust                  72
None                    1
Name: scheme_management, dtype: int64
```

There are more null values in `scheme_management` column; `management_group` column is similar to `management` column so I will keep `management` column only.

In [12]:
```
train_val['construction_year'].value_counts()
```

Out[12]:

```
0       20709
2010     2645
2008     2613
2009     2533
2000     2091
2007     1587
2006     1471
2003     1286
2011     1256
2004     1123
2012     1084
2002     1075
1978     1037
1995     1014
2005     1011
1999      979
1998      966
1990      954
1985      945
1980      811
1996      811
1984      779
1982      744
1994      738
1972      708
1974      676
1997      644
1992      640
1993      608
2001      540
1988      521
1983      488
1975      437
1986      434
1976      414
1970      411
1991      324
1989      316
1987      302
1981      238
1977      202
1979      192
1973      184
2013      176
1971      145
1960      102
1967       88
1963       85
1968       77
1969       59
1964       40
1962       30
1961       21
1965       19
1966       17
Name: construction_year, dtype: int64
```

`construction_year` column is in integer format but not continuous data do not make sense for model. So, I divided them in decades and assumed every decade as categorical value.

In [13]:
```
# Create new column
train_val['decade'] = train_val['construction_year']
```

In [14]:
```
# Dividing the column decades

train_val['decade'].replace(to_replace = (1960,1961,1962,1963,1964,1965,
1966,1967,1968,1969),
                        value ='60s' , inplace = True)
train_val['decade'].replace(to_replace = (1970,1971,1972,1973,1974,1975,
1976,1977,1978,1979),
                        value ='70s' , inplace = True)
train_val['decade'].replace(to_replace = (1980,1981,1982,1983,1984,1985,
1986,1987,1988,1989),
                        value ='80s' , inplace = True)
train_val['decade'].replace(to_replace = (1990,1991,1992,1993,1994,1995,
1996,1997,1998,1999),
                        value ='90s' , inplace = True)
train_val['decade'].replace(to_replace = (2000,2001,2002,2003,2004,2005,
2006,2007,2008,2009),
                        value ='00s' , inplace = True)
train_val['decade'].replace(to_replace = (2010,2011,2012,2013),
                        value ='10s' , inplace = True)
train_val['decade'][train_val['decade'] == 0] = 'unknown'
```

```
/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-
packages/ipykernel_launcher.py:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-d
ocs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  from ipykernel import kernelapp as app
```

In [15]:
```
train_val['decade'].value_counts()
```

Out[15]:
```
unknown    20709
00s        15330
90s         7678
80s         5578
10s         5161
70s         4406
60s          538
Name: decade, dtype: int64
```

In [16]:
```
# Dropping columns with the same or irrelevant information
drop_cols = ['amount_tsh',
             'recorded_by',
             'quantity_group',
             'source_type',
             'source_class',
             'num_private',
             'quality_group',
             'extraction_type',
             'extraction_type_class',
             'payment_type',
             'waterpoint_type_group',
             'management_group',
             'scheme_management',
             'wpt_name',
             'region_code',
             'scheme_name',
             'construction_year', ]

train_val = train_val.drop(columns = drop_cols)
```

In [17]:
```
# Exploring numerical columns
train_val.describe()
```

Out[17]:

	gps_height	longitude	latitude	district_code	population
count	59400.000000	59400.000000	5.940000e+04	59400.000000	59400.000000
mean	668.297239	34.077427	-5.706033e+00	5.629747	179.909983
std	693.116350	6.567432	2.946019e+00	9.633649	471.482176
min	-90.000000	0.000000	-1.164944e+01	0.000000	0.000000
25%	0.000000	33.090347	-8.540621e+00	2.000000	0.000000
50%	369.000000	34.908743	-5.021597e+00	3.000000	25.000000
75%	1319.250000	37.178387	-3.326156e+00	5.000000	215.000000
max	2770.000000	40.345193	-2.000000e-08	80.000000	30500.000000

In [18]:
```
# Exploring object columns
train_val[[c for c in train_val.columns if train_val[c].dtype == 'objec
t']].describe()
```

Out[18]:

	date_recorded	funder	installer	basin	subvillage	region	lga	ward	public_r
count	59400	55765	55745	59400	59029	59400	59400	59400	
unique	356	1897	2145	9	19287	21	125	2092	
top	2011-03-15	Government Of Tanzania	DWE	Lake Victoria	Madukani	Iringa	Njombe	Igosi	
freq	572	9084	17402	10248	508	5294	2503	307	

In [19]:
```
# Get a Null/Nan Report
na_sum = train_val.isna().sum()
na_sum
```

Out[19]:
```
date_recorded               0
funder                   3635
gps_height                  0
installer                3655
longitude                   0
latitude                    0
basin                       0
subvillage                371
region                      0
district_code               0
lga                         0
ward                        0
population                  0
public_meeting           3334
permit                   3056
extraction_type_group       0
management                  0
payment                     0
water_quality               0
quantity                    0
source                      0
waterpoint_type             0
status_group                0
decade                      0
dtype: int64
```

In [20]:
```
# changing from bool to int
train_val['permit'] = train_val['permit'].astype(bool).astype(int)
```

In [21]:
```
# changing from bool to int
train_val['public_meeting'] = train_val['public_meeting'].astype(bool).a
stype(int)
```

In [22]:
```
# filling 0 and null values in funder column with unknown
train_val['funder'].fillna(value='Unknown',inplace=True)
train_val['funder'].replace(to_replace = '0', value ='Unknown' , inplace
=True)
```

In [23]:
```
train_val['funder'].value_counts().head(10)
```

Out[23]:
```
Government Of Tanzania    9084
Unknown                   4416
Danida                    3114
Hesawa                    2202
Rwssp                     1374
World Bank                1349
Kkkt                      1287
World Vision              1246
Unicef                    1057
Tasaf                      877
Name: funder, dtype: int64
```

In [24]:
```
# 10 most common funders
train_val1 = train_val.loc[train_val['funder']== 'Government Of Tanzani
a']
train_val2 = train_val.loc[train_val['funder']== 'Unknown']
train_val3 = train_val.loc[train_val['funder']== 'Danida']
train_val4 = train_val.loc[train_val['funder']== 'Hesawa']
train_val5 = train_val.loc[train_val['funder']== 'World Bank']
train_val6 = train_val.loc[train_val['funder']== 'Rwssp']
train_val7 = train_val.loc[train_val['funder']== 'Kkkt']
train_val8 = train_val.loc[train_val['funder']== 'World Vision']
train_val9 = train_val.loc[train_val['funder']== 'Unicef']
train_val10 = train_val.loc[train_val['funder']== 'Tasaf']

df_funder = pd.concat([train_val1,train_val2,train_val3,train_val4,train
_val5,train_val6,train_val7,train_val8,train_val9,train_val10], ignore_i
ndex=True)
```

In [25]:
```
# creating new column
train_val['funder_col'] = train_val['funder']

c_fund = ['Government Of Tanzania','Unknown','Danida','Hesawa','Rwssp',
'World Bank','Kkkt', 'World Vision',
         'Unicef', 'Tasaf']

# converting the values which have less than 600 value counts to others
train_val.loc[~train_val["funder_col"].isin(c_fund), "funder_col"] = "Ot
hers"
```

In [26]:
```
train_val['funder_col'].nunique()
```

Out[26]:
```
11
```

In [27]:
```
# filling 0 and null values in installer column with unknown
train_val['installer'].fillna(value='Unknown',inplace=True)
train_val['installer'].replace(to_replace = '0', value ='Unknown' , inpl
ace=True)
```

In [28]:
```
train_val['installer'].value_counts().head(10)
```

Out[28]:
```
DWE                   17402
Unknown                4435
Government             1825
RWE                    1206
Commu                  1060
DANIDA                 1050
KKKT                    898
Hesawa                  840
TCRS                    707
Central government      622
Name: installer, dtype: int64
```

In [29]:
```
# 10 most common installers
train_val1 = train_val.loc[train_val['installer']== 'DWE']
train_val2 = train_val.loc[train_val['installer']== 'Unknown']
train_val3 = train_val.loc[train_val['installer']== 'Government']
train_val4 = train_val.loc[train_val['installer']== 'RWE']
train_val5 = train_val.loc[train_val['installer']== 'Commu']
train_val6 = train_val.loc[train_val['installer']== 'DANIDA']
train_val7 = train_val.loc[train_val['installer']== 'KKKT']
train_val8 = train_val.loc[train_val['installer']== 'Hesawa']
train_val9 = train_val.loc[train_val['installer']== 'TCRS']
train_val10 = train_val.loc[train_val['installer']== 'Central governmen
t']

df_installer = pd.concat([train_val1,train_val2,train_val3,train_val4,tr
ain_val5,train_val6,train_val7,train_val8,train_val9,train_val10], ignor
e_index=True)
```

In [30]:
```
# creating new column
train_val['installer_col'] = train_val['installer']

c_install = ['DWE','Unknown','Government','RWE','Commu','DANIDA','KKT',
'Hesawa',
         'TCRS', 'Central government']

# converting the values which have less than  500 value counts to others
train_val.loc[~train_val["installer_col"].isin(c_fund), "installer_col"]
= "Others"
```

In [31]:
```
train_val['installer_col'].nunique()
```

Out[31]: 8

In [32]:
```
# Addressing Nulls/NaNs for categorical columns

na_columns = na_sum[na_sum != 0].index

def fill_cat_feat(s: pd.Series):
    random_cat = random.choice(s.dropna().values)
    return s.fillna(random_cat)

for c in na_columns:
    train_val[c] = fill_cat_feat(train_val[c])
```

In [33]:
```
# Fill NA/NaN with mean for numerical columns
numerical_columns = train_val.describe().columns
train_val[numerical_columns] = train_val[numerical_columns].fillna(train_val[numerical_columns].mean())
```

In [34]:
```
na_sum = train_val.isna().sum()
na_sum
```

Out[34]:
```
date_recorded            0
funder                   0
gps_height               0
installer                0
longitude                0
latitude                 0
basin                    0
subvillage               0
region                   0
district_code            0
lga                      0
ward                     0
population               0
public_meeting           0
permit                   0
extraction_type_group    0
management               0
payment                  0
water_quality            0
quantity                 0
source                   0
waterpoint_type          0
status_group             0
decade                   0
funder_col               0
installer_col            0
dtype: int64
```

None of the columns have the null values.

In [35]:
```
# assign numerical columns
numerical_columns = train_val.describe().columns
numerical_columns = numerical_columns[numerical_columns != 'id']

# assign categorical columns
categorical_columns = train_val[[c for c in train_val.columns if train_v
al[c].dtype == 'object']].describe().columns
```

In [36]:
```
# declare a treshold for unique values
mask = (train_val[categorical_columns].nunique() < 21).values

# transform categorical and numerical columns to list and assign them to
train_val
train_val = train_val[categorical_columns[mask].tolist() + numerical_col
umns.tolist()]
```

In [37]:
```
numerical_columns = train_val.describe().columns
numerical_columns = numerical_columns[numerical_columns != 'id']
categorical_columns = train_val[[c for c in train_val.columns if train_v
al[c].dtype == 'object']].describe().columns
categorical_columns = categorical_columns[categorical_columns != TARGET]
```

In [38]:
```
# check how many columns with unique values left
train_val.nunique()
```

Out[38]:
```
basin                      9
extraction_type_group     13
management                12
payment                    7
water_quality              8
quantity                   5
source                    10
waterpoint_type            7
status_group               3
decade                     7
funder_col                11
installer_col              8
gps_height              2428
longitude              57516
latitude               57517
district_code             20
population              1049
public_meeting             2
permit                     2
dtype: int64
```

Model Development

I will creat 6 classifiers models and I will use accuracy score as my primary metric to pick the best model.

1. Logistic Regression.Uses an equation as the representation where input values are combined linearly using weights or coefficient values to predict an output value such as a multiclass value. .

1. K-Neighbors. Finds closest in distance of the new point to predefined number of training points and predict the label according to this.

1. Random Forrest. Reduces overfitting risk.

1. Support vector machines. Looks at creating planes as boundaries that differentiate the categories, trying to maximise the distance between the classes.

1. XGBoost. A decision tree algorithm that uses gradient boosting.

1. CatBoost.Requires minimal data preparation,easy to implement,converts categorical values into numbers using various statistics on combinations of categorical features and combinations of categorical and numerical features.It reduces the need for extensive hyper-parameter tuning and lower the chances of overfitting.

The main reason why I chose these models is because they have different computational approaches.

In [39]:
```
X_train_val, y_train_val = train_val.drop(columns=[TARGET]), train_val[TARGET]
```

In [40]:
```
# converting the labels into numeric form so as to convert it into the machine-readable form
le = LabelEncoder()
le.fit(y_train_val)

y_train_val = le.transform(y_train_val)
```

In [41]:
```
# choosing train-validation splits
X_train, X_valid, y_train, y_valid = train_test_split(X_train_val, y_train_val,
                                                      test_size = 0.2,
                                                      random_state = 179
)

# getting together scaler and encoder with column transformer
column_transformer = ColumnTransformer([
    ('ohe', OneHotEncoder(handle_unknown = "ignore"), categorical_columns),
    ('scaling', MinMaxScaler(), numerical_columns)
])

X_train = column_transformer.fit_transform(X_train)
X_valid = column_transformer.transform(X_valid)
```

In [42]:
```
# execution of Logistic Regression

lr = LogisticRegression(max_iter=100)

lr.fit(X_train, y_train)
y_preds = lr.predict(X_valid)

# predictions on train set
lr.predict(X_valid)

print(accuracy_score(y_valid, y_preds))
```

```
0.7390572390572391

/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-
packages/sklearn/linear_model/_logistic.py:764: ConvergenceWarning: lbf
gs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown
in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [43]:
```
# execution of K-Nearesrt Neighbors

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

# predictions on train set
y_preds = knn.predict(X_valid)


print(accuracy_score(y_valid, y_preds))
```

```
0.7765993265993266
```

In [44]:
```
# execution of Random Forrest Classifier

rf = RandomForestClassifier(n_estimators = 100, max_depth = 5, verbose=1
0)
rf.fit(X_train, y_train)

# predictions on train set
y_preds = rf.predict(X_valid)

print("accuracy:", accuracy_score(y_valid, y_preds))
```

```
building tree 1 of 100
building tree 2 of 100
building tree 3 of 100

[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.1s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.2s remaining:
0.0s

building tree 4 of 100
building tree 5 of 100
building tree 6 of 100
building tree 7 of 100
building tree 8 of 100
building tree 9 of 100

[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.2s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.2s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.3s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.3s remaining:
0.0s
```

```
building tree 10 of 100
building tree 11 of 100
building tree 12 of 100
building tree 13 of 100
building tree 14 of 100
building tree 15 of 100
building tree 16 of 100
building tree 17 of 100
building tree 18 of 100
building tree 19 of 100
building tree 20 of 100
building tree 21 of 100
building tree 22 of 100
building tree 23 of 100
building tree 24 of 100
building tree 25 of 100
building tree 26 of 100
building tree 27 of 100
building tree 28 of 100
building tree 29 of 100
building tree 30 of 100
building tree 31 of 100
building tree 32 of 100
building tree 33 of 100
building tree 34 of 100
building tree 35 of 100
building tree 36 of 100
building tree 37 of 100
building tree 38 of 100
building tree 39 of 100
building tree 40 of 100
building tree 41 of 100
building tree 42 of 100
building tree 43 of 100
building tree 44 of 100
building tree 45 of 100
building tree 46 of 100
building tree 47 of 100
building tree 48 of 100
building tree 49 of 100
building tree 50 of 100
building tree 51 of 100
building tree 52 of 100
building tree 53 of 100
building tree 54 of 100
building tree 55 of 100
building tree 56 of 100
building tree 57 of 100
building tree 58 of 100
building tree 59 of 100
building tree 60 of 100
building tree 61 of 100
building tree 62 of 100
building tree 63 of 100
building tree 64 of 100
building tree 65 of 100
building tree 66 of 100
```

```
building tree 67 of 100
building tree 68 of 100
building tree 69 of 100
building tree 70 of 100
building tree 71 of 100
building tree 72 of 100
building tree 73 of 100
building tree 74 of 100
building tree 75 of 100
building tree 76 of 100
building tree 77 of 100
building tree 78 of 100
building tree 79 of 100
building tree 80 of 100
building tree 81 of 100
building tree 82 of 100
building tree 83 of 100
building tree 84 of 100
building tree 85 of 100
building tree 86 of 100
building tree 87 of 100
building tree 88 of 100
building tree 89 of 100
building tree 90 of 100
building tree 91 of 100
building tree 92 of 100
building tree 93 of 100
building tree 94 of 100
building tree 95 of 100
building tree 96 of 100
building tree 97 of 100
building tree 98 of 100
building tree 99 of 100
building tree 100 of 100
accuracy: 0.7202861952861953
```

```
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    2.8s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent
workers.
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   2 out of   2 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   3 out of   3 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   4 out of   4 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   5 out of   5 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   6 out of   6 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   7 out of   7 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   8 out of   8 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done   9 out of   9 | elapsed:    0.0s remaining:
0.0s
[Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    0.1s finished
```

In [45]:

```python
# execution of Support Vector Machine Classifier
clf = SVC(C = 0.1, kernel= 'rbf', max_iter = 1000)

clf.fit(X_train, y_train)
```

```
/Users/alinakorsunenko/opt/anaconda3/envs/learn-env/lib/python3.6/site-
packages/sklearn/svm/_base.py:249: ConvergenceWarning: Solver terminate
d early (max_iter=1000).  Consider pre-processing your data with Standa
rdScaler or MinMaxScaler.
  % self.max_iter, ConvergenceWarning)
```

Out[45]:

```
SVC(C=0.1, max_iter=1000)
```

In [46]:

```python
y_preds = clf.predict(X_valid)

print("accuracy:", accuracy_score(y_valid, y_preds))
```

```
accuracy: 0.5418350168350168
```

In [47]:

```python
# execution of XGBoost Classifier
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train)

score = xgb_clf.score(X_valid, y_valid)
print(score)
```

```
0.7473063973063973
```

In [48]:
```
# execution of CatBooster

model = CatBoostClassifier(iterations=3000,
          eval_metric='Accuracy')

eval_dataset = Pool(X_valid,
                    y_valid)

model.fit(X_train,
          y_train,
          eval_set=eval_dataset,
          verbose=False,
             plot=True)
```

Out[48]: <catboost.core.CatBoostClassifier at 0x1a2610dd68>

In [49]:
```
y_preds = model.predict(X_valid)

print("accuracy:", accuracy_score(y_valid, y_preds))
```

```
accuracy: 0.8026936026936027
```

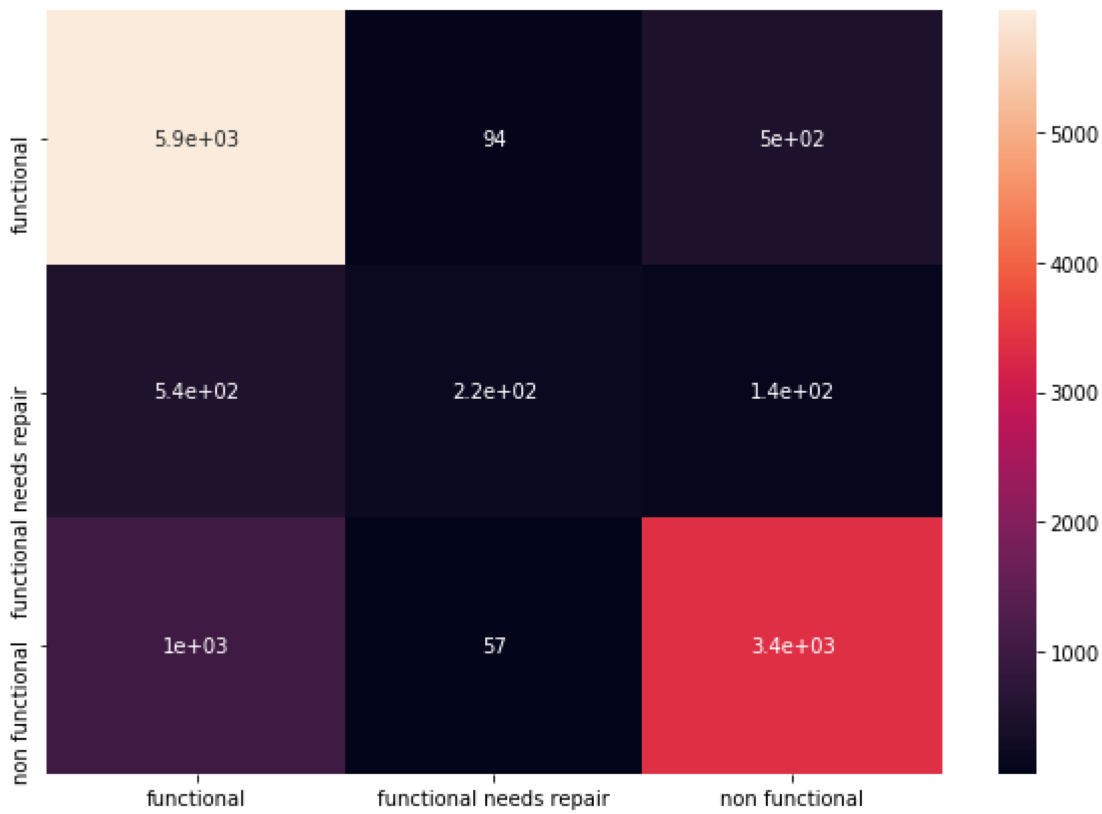
All of these 6 models, CatBoostClassifier showed the best result 80%. KNeighborsClassifier showed also a good result which is 77%.

Classification accuracy alone can be misleading in my situatuin because I have three classes in my dataset.

Computing a confusion matrix can give me a better idea of what my classification model is getting right and what types of errors it is making.

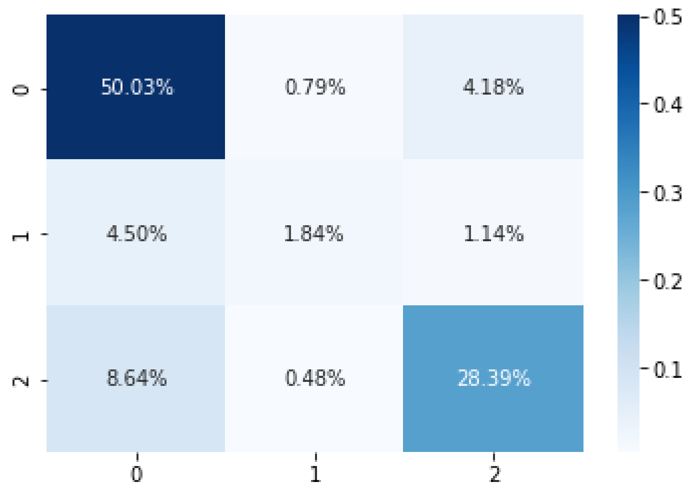
In [50]:
```
# Compute confusion matrix for performance measurement
cm = confusion_matrix(y_valid, y_preds)
df_cm = pd.DataFrame(cm, index = [i for i in np.unique(le.inverse_transf
orm(y_train))],
                     columns = [i for i in np.unique(le.inverse_transform(y
_train))])
plt.figure(figsize = (10,7))
sns.heatmap(df_cm, annot=True);

print("Classification Report")
print(classification_report(y_valid, y_preds))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.79      0.91      0.85      6534
           1       0.59      0.25      0.35       890
           2       0.84      0.76      0.80      4456

    accuracy                           0.80     11880
   macro avg       0.74      0.64      0.66     11880
weighted avg       0.80      0.80      0.79     11880
```



In [51]:
```
# check what percentage of my data is represented in each quadrant
sns.heatmap(cm/np.sum(cm), annot=True,
            fmt='.2%', cmap='Blues')
```

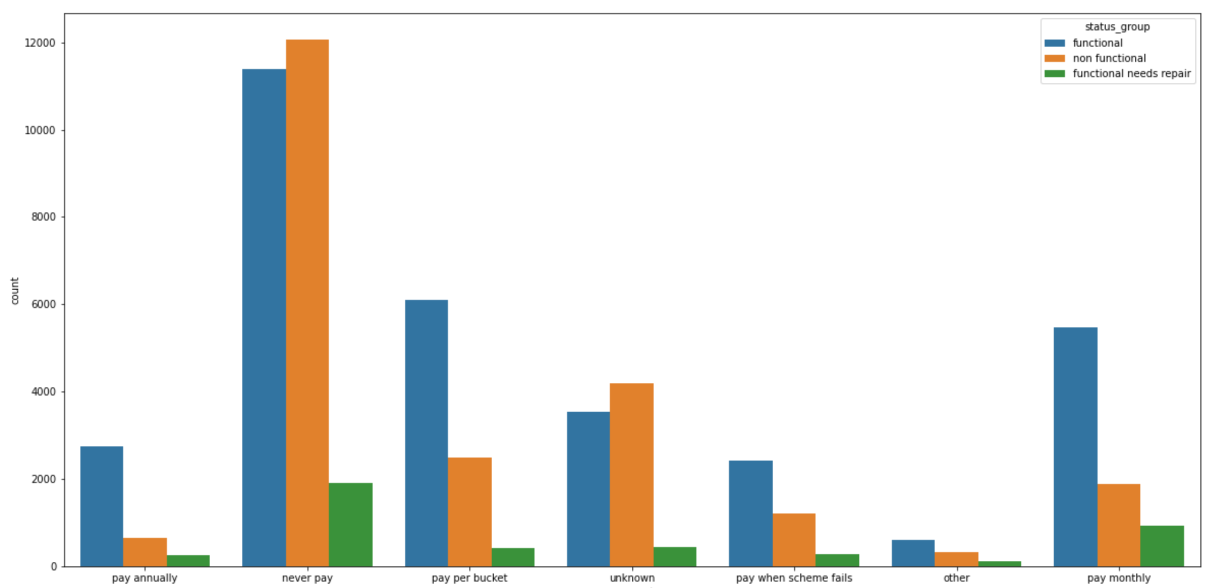
Out[51]: <AxesSubplot:>



Because of the class imbalance and low amount of values in `functional needs repair` my classification model is confused when it makes predictions.

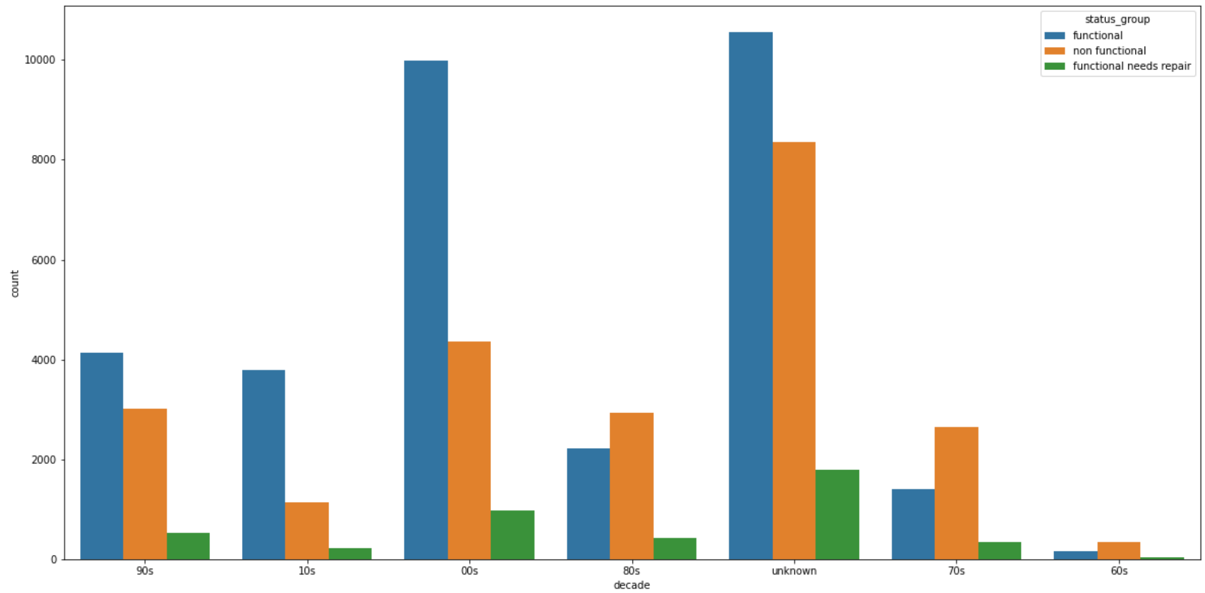
Visualizations

In [52]:
```
# check if `payment` relates to water well functionality
plt.figure(figsize=(20,10))
ax = sns.countplot(x='payment', hue="status_group", data=train_val)
```



According to the above visualization `never pay` water well pumps have the highest number of non-functioning pumps.So, it is safe to assume that in order to have more functional pumps, Tanzania Ministry of Water and Irrigation should try to find a solution how to motivate/help citizens pay for water.

In [53]:
```
plt.figure(figsize=(20,10))
ax = sns.countplot(x='decade', hue="status_group", data=train_val)
```



This plot proves that more `non functional` pumps apear during the time that is why it is important to maintain the water wells.

Conclusions

My model can predict the functionality of the water wells with 80% accuracy. That number is high enough but my model can use some hyperparameter optimization to get that number even better.

Based on my research I can recommend :

1. Make functional wells one of the State priorities (including budgeting, financial incentives).
2. Create a strategy plan on preventing of emerging new non functional wells.

Future Work:

1. Solve Imbalanced target problem by using SMOTE oversampling technique to create balanced data.

1. Use Grid Search to Optimise CatBoost Parameters and/or KNN

1. Create a visual representation of how different values affect predictions (feature importance with catboost)

In [ ]: