AHMET16 / Tanzania-Ministry-of-Water Public

Code Issues Pull requests Actions Projects Wiki Security

main

Tanzania-Ministry-of-Water / Untitled.ipynb

AHMET16 Add files via upload History

1 contributor

1772 lines (1772 sloc) | 169 KB

1. Business Understanding

There are two facts about Tanzania we want the reader to know before continuing.

Three out of ten people do not have access to basic drinking water. Four million people lack access to an improved source of safe water.

The Tanzania Ministry of Water has tasked our team with identifying which wells have a proclivity to be functional, while also identifying which wells are in need of repair. All of this will be achieved through predictive modeling from basic information we acquired about each well.

The process for a Tanzanian to procure water is very different compared to those of us who reside in a developed nation. Only 3% of homes in this East African country have an access point within their home while 81% of people are required to travel outside of their village or compound to reach a water well [3].

Our data scientist, Andrew, lived in Tanzania for over a year and shared that during his participation during an excursion on water collecting where simply accessing the well took a 45 minute walk. This highlights the need for the Ministry of Water to be able to anticipate which wells are in need of service and thus allow a chance to be proactive with repairs.

Due to the risk of dehydration it's imperative to ensure that the wells communities depend on remain in service. Failing to find the best models or a high rate of error would directly affect citizens' ability to receive the water needed for survival. As our binary target was split into wells needing repair (0) and functional wells (1), a false positive would mean marking a well as functioning when it is not. Every false positive means a community is not labeled as in need of aid. This can lead to a community to take drastic measures such as relocating to regain reliable access to water or risk dying of thirst. It can also waste valuable resouces that could be used on non-functional wells. Because of these risks we decided that precision as our evaluation metric was of the most importance.

2. Data Understanding

Our team obtained data sets from DrivenData who collected data from the Tanzania Ministry of Water and Taarifa, a Rwandan news provider, in addition to conducting some initial data cleaning. This initial data included a training data set which contained each of these wells' functional status. The data was limited by data from 2011 - 2013 and also had some unusable data points. The

by date from 2011 - 2013 and also had some unuseable data points. The private_num was unspecificed as to what it was and population also had a lot of 0's which we could not identify as null, erroneously data or if it was truly 0.

Importing the packages/libraries and our datasets

In [1]:
```
# import required packages
# Import required packages
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
import geopandas as gpd
import contextily as cx

from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.dummy import DummyClassifier
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.ensemble import BaggingClassifier, RandomForestClassifie
ExtraTreesClassifier, VotingClassifier, StackingRegressor

from sklearn.compose import ColumnTransformer
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, OneHotEncoder, Func
from sklearn.metrics import confusion_matrix, plot_confusion_matrix,\
    precision_score, accuracy_score, log_loss, make_scorer
```

In [2]:
```
# Importing CSV's
df_test = pd.read_csv('test_set_values.csv')
df_train = pd.read_csv('training_set_values.csv')
df_train_label = pd.read_csv('training_set_labels.csv')
```

In [3]:
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14850 entries, 0 to 14849
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     14850 non-null  int64  
 1   amount_tsh             14850 non-null  float64
 2   date_recorded          14850 non-null  object 
 3   funder                 13981 non-null  object 
 4   gps_height             14850 non-null  int64  
 5   installer              13973 non-null  object 
 6   longitude              14850 non-null  float64
 7   latitude               14850 non-null  float64
```

```
 8   wpt_name               14850 non-null  object
 9   num_private            14850 non-null  int64
 10  basin                  14850 non-null  object
 11  subvillage             14751 non-null  object
 12  region                 14850 non-null  object
 13  region_code            14850 non-null  int64
 14  district_code          14850 non-null  int64
 15  lga                    14850 non-null  object
 16  ward                   14850 non-null  object
 17  population             14850 non-null  int64
 18  public_meeting         14029 non-null  object
 19  recorded_by            14850 non-null  object
 20  scheme_management      13881 non-null  object
 21  scheme_name            7758 non-null   object
 22  permit                 14113 non-null  object
 23  construction_year      14850 non-null  int64
 24  extraction_type        14850 non-null  object
 25  extraction_type_group  14850 non-null  object
 26  extraction_type_class  14850 non-null  object
 27  management             14850 non-null  object
 28  management_group       14850 non-null  object
 29  payment                14850 non-null  object
 30  payment_type           14850 non-null  object
 31  water_quality          14850 non-null  object
 32  quality_group          14850 non-null  object
 33  quantity               14850 non-null  object
 34  quantity_group         14850 non-null  object
 35  source                 14850 non-null  object
 36  source_type            14850 non-null  object
 37  source_class           14850 non-null  object
 38  waterpoint_type        14850 non-null  object
 39  waterpoint_type_group  14850 non-null  object
dtypes: float64(3), int64(7), object(30)
memory usage: 4.5+ MB
```

In [4]:

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 40 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   id                     59400 non-null  int64
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object
 3   funder                 55765 non-null  object
 4   gps_height             59400 non-null  int64
 5   installer              55745 non-null  object
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59400 non-null  object
 9   num_private            59400 non-null  int64
 10  basin                  59400 non-null  object
 11  subvillage             59029 non-null  object
 12  region                 59400 non-null  object
 13  region_code            59400 non-null  int64
 14  district_code          59400 non-null  int64
 15  lga                    59400 non-null  object
 16  ward                   59400 non-null  object
 17  population             59400 non-null  int64
```

```
 18  public_meeting          56066 non-null  object
 19  recorded_by             59400 non-null  object
 20  scheme_management       55523 non-null  object
 21  scheme_name             31234 non-null  object
 22  permit                  56344 non-null  object
 23  construction_year       59400 non-null  int64
 24  extraction_type         59400 non-null  object
 25  extraction_type_group   59400 non-null  object
 26  extraction_type_class   59400 non-null  object
 27  management              59400 non-null  object
 28  management_group        59400 non-null  object
 29  payment                 59400 non-null  object
 30  payment_type            59400 non-null  object
 31  water_quality           59400 non-null  object
 32  quality_group           59400 non-null  object
 33  quantity                59400 non-null  object
 34  quantity_group          59400 non-null  object
 35  source                  59400 non-null  object
 36  source_type             59400 non-null  object
 37  source_class            59400 non-null  object
 38  waterpoint_type         59400 non-null  object
 39  waterpoint_type_group   59400 non-null  object
dtypes: float64(3), int64(7), object(30)
memory usage: 18.1+ MB
```

In [5]:
```
df_train_label.columns
```

Out[5]:
```
Index(['id', 'status_group'], dtype='object')
```

In [6]:
```
df_train_label.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 2 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   id            59400 non-null  int64
 1   status_group  59400 non-null  object
dtypes: int64(1), object(1)
memory usage: 928.2+ KB
```

- we found a total of 41 columns
- we found that only 32,259 water wells out of a total of 59,400 in the country have been recorded as functional and set to work deciphering which columns could be eliminated to gain a clearer picture of what we were working with.

Data Preparation and EDA

We continued with additional cleaning ourselves, changing the three initial well status targets into binary target of wells that are completely functional and wells in need of repair. This weights wellsthat are completely broken equally wth wells that are technically functional, but are unreliable or have contaminated water due

malfunction.

In [7]:
```
# Reassigning targets to establish bionomial targets
target = df_train_label.replace({'status_group': {'functional' : 1,
                                 'non functional' : 0,
                                 'functional needs repair' : 0}})
df = pd.concat([df_train, target], axis = 1)
```

In [8]:
```
# Reassigning targets to establish bionomial targets
target = df_train_label.replace({'status_group': {'functional' : 1,
                                 'non functional' : 0,
                                 'functional needs repair' : 0}})
df = pd.concat([df_train, target], axis = 1)
```

In [9]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 42 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   id                     59400 non-null  int64  
 1   amount_tsh             59400 non-null  float64
 2   date_recorded          59400 non-null  object 
 3   funder                 55765 non-null  object 
 4   gps_height             59400 non-null  int64  
 5   installer              55745 non-null  object 
 6   longitude              59400 non-null  float64
 7   latitude               59400 non-null  float64
 8   wpt_name               59400 non-null  object 
 9   num_private            59400 non-null  int64  
 10  basin                  59400 non-null  object 
 11  subvillage             59029 non-null  object 
 12  region                 59400 non-null  object 
 13  region_code            59400 non-null  int64  
 14  district_code          59400 non-null  int64  
 15  lga                    59400 non-null  object 
 16  ward                   59400 non-null  object 
 17  population             59400 non-null  int64  
 18  public_meeting         56066 non-null  object 
 19  recorded_by            59400 non-null  object 
 20  scheme_management      55523 non-null  object 
 21  scheme_name            31234 non-null  object 
 22  permit                 56344 non-null  object 
 23  construction_year      59400 non-null  int64  
 24  extraction_type        59400 non-null  object 
 25  extraction_type_group  59400 non-null  object 
 26  extraction_type_class  59400 non-null  object 
 27  management             59400 non-null  object 
 28  management_group       59400 non-null  object 
 29  payment                59400 non-null  object 
 30  payment_type           59400 non-null  object 
 31  water_quality          59400 non-null  object 
 32  quality_group          59400 non-null  object 
 33  quantity               59400 non-null  object 
 34  quantity_group         59400 non-null  object 
```

```
 35  source                 59400 non-null  object
 36  source_type            59400 non-null  object
 37  source_class           59400 non-null  object
 38  waterpoint_type        59400 non-null  object
 39  waterpoint_type_group  59400 non-null  object
 40  id                     59400 non-null  int64
 41  status_group           59400 non-null  int64
dtypes: float64(3), int64(9), object(30)
memory usage: 19.0+ MB
```

In [10]:
```
df.describe()
```

Out[10]:

	id	amount_tsh	gps_height	longitude	latitude
count	59400.000000	59400.000000	59400.000000	59400.000000	5.940000e+04
mean	37115.131768	317.650385	668.297239	34.077427	-5.706033e+00
std	21453.128371	2997.574558	693.116350	6.567432	2.946019e+00
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01
25%	18519.750000	0.000000	0.000000	33.090347	-8.540621e+00
50%	37061.500000	0.000000	369.000000	34.908743	-5.021597e+00
75%	55656.500000	20.000000	1319.250000	37.178387	-3.326156e+00
max	74247.000000	350000.000000	2770.000000	40.345193	-2.000000e-08

Eliminating features that we found to be redudant or unuseable for our models.

Many of the features had the same information provided by similar features in the dataset so we trimmed them. In addition to this some features like 'num_private' had no feature descriptions explaining what it was so we chose to not utilize it.

In [11]:
```
col_to_delete = ['id', 'recorded_by', 'funder', 'public_meeting',
                 'lga', 'ward', 'region_code', 'district_code',
                  'wpt_name','scheme_name', 'extraction_type', 'extrac
                  'payment', 'quality_group', 'source_type', 'quantity
                  'waterpoint_type_group', 'subvillage', 'num_private'
```

We created a new feature 'year_recorded' to capture only the year that the data was recorded to transform the 'date_recorded'. This was because we believed that this may have been important to keep in our data, but keeping it in its original state may have cause our model to be too complex with the sheer number of unique values.

We applied the same thought processes to the following features. In addition we dropped any data that was entered before 2005 as it only comprised 0.06% of our data and created additional problems when attempting to train test split our

data.

In [12]:

```
def data_cleaning(df_to_clean):
    # Removing columns that are non-factors for our model
    col_to_delete = ['id', 'recorded_by', 'funder', 'public_meeting',
                'lga', 'ward', 'region_code', 'district_code',
                 'wpt_name','scheme_name', 'extraction_type', 'extrac
                 'payment', 'quality_group', 'source_type', 'quantity
                 'waterpoint_type_group', 'subvillage', 'num_private'

    # Remove duplicated data entries and null values
    dfn = df_to_clean.drop(col_to_delete, axis = 1)
    dfn = dfn.dropna(axis = 0)

    # Pulling the year off and type casting to int
    dfn['year_recorded'] = [int(val[0:4]) for val in dfn['date_record
    dfn['year_recorded'].astype(np.int64)
    dfn.drop(['date_recorded'], axis = 'columns', inplace = True)

    # Binning the years into decades
    dfn['construction_year'] = ['unknown' if val == 0
                            else str((val // 10) * 10) for val in dfn[

    # Binning the unique values
    scheme_management_list = ['SWC', 'Trust', 'None']
    dfn['scheme_management'].replace(scheme_management_list, 'Other',

    # Binning unique values
    installer_list = ['DWE', 'Government', 'Commu', 'DANIDA',
                  'RWE', 'KKKT', 'TCRS']
    dfn['installer'] = ['Other' if val not in installer_list
                            else val for val in dfn['installer']]

    dfn.drop(dfn.index[dfn['year_recorded'] < 2005], inplace=True)
    dfn.reset_index(inplace=True, drop=True)
    return dfn
```

4.Modeling

The cleaned training data was combined with the target data and split into a 75% train/25% testing set for us to train and evaluate the effectiveness of our models before we attempted to use them on our true testing data, a similar list of wells with no functional status provided.

We also designed functions for :

- One Hot Encoding and Scaling our data
- Creating a dataframe with scaled numerics and one hot encoded categoricals
- Printing the accuracy, precision score as well as a confusion matrix for the model

In [13]:

In [13]:
```
def num_encoder(df_to_encode):
    ss = StandardScaler()

    ss.fit(df_to_encode)
    nums_df = pd.DataFrame(ss.transform(df_to_encode),
                           columns = df_to_encode.columns,
                          index = df_to_encode.index)
    return nums_df

def cat_encoder(df_to_encode):
    ohe = OneHotEncoder(
        drop = 'first',
        sparse = False)

    dums = ohe.fit_transform(df_to_encode)
    dums_df = pd.DataFrame(dums,
                           columns = ohe.get_feature_names(),
                           index = df_to_encode.index)
    return dums_df
```

In [14]:
```
def split_join(split):
    categories = split.select_dtypes('object')
    numerics = split.select_dtypes(['float64', 'int64'])

    joined = pd.concat([num_encoder(numerics), cat_encoder(categories
    return joined
```

In [15]:
```
def score_maxtrix_printer(model, X_train, y_train, X_test, y_test):

    train_pred = model.predict(X_train)
    test_pred = model.predict(X_test)

    ascore_train = accuracy_score(y_train, train_pred)
    pscore_train = precision_score(y_train, train_pred)

    ascore_test = accuracy_score(y_test, test_pred)
    pscore_test = precision_score(y_test, test_pred)

    conf_mat = plot_confusion_matrix(model, X_test, y_test)

    print(f"""
    Train Accuracy: {ascore_train}
    Train Precision: {pscore_train}
    Test Accuracy: {ascore_test}
    Test Precision: {pscore_test}
    """)
```

Establishing the Baseline (Dummy) Model

In [16]:
```
df2 = data_cleaning(df)
```

In [17]:
```
y = df2.status_group
```

```
y = df2.status_group
X = df2.drop('status_group', axis = 1)

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size = 0.25,
```

In [18]:

```
X_train_cat = X_train.select_dtypes('object')
X_train_nums = X_train.select_dtypes(['float64', 'int64'])

cont_pipeline = Pipeline(steps=[
    ('ss', StandardScaler())
])

cat_pipeline = Pipeline(steps=[
    ('ohe', OneHotEncoder(drop = 'first'))
])

trans = ColumnTransformer(transformers=[
    ('continuous', cont_pipeline, X_train_nums.columns),
    ('categorical', cat_pipeline, X_train_cat.columns)
])

dummy = Pipeline(steps=[
    ('trans', trans),
    ('dummy', DummyClassifier(random_state = 69, strategy = 'most_fre
])

#Fitting and checking the score
dummy.fit(X_train, y_train)
dummy.score(X_train, y_train)
```

Out[18]:

```
0.546180041072032
```

Our dummy model predictably produces a score of 54% because it is based on the majority target. This establishes our baseline.

Model 1 (Decision Tree Classifier)

We decided to use a decision tree as our first model for feature selection. For the first iteration we did not specify any parameters except for the random state.

In [19]:

```
X_train_clean = split_join(X_train)
X_test_clean = split_join(X_test)
```

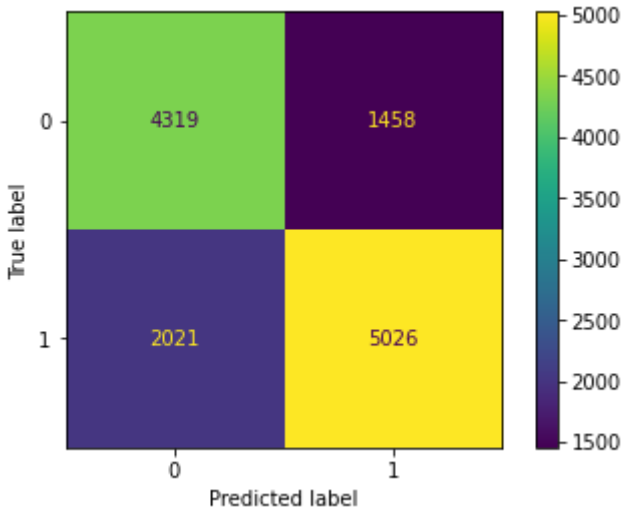
```
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function get_feature_names is d
eprecated; get_feature_names is deprecated in 1.0 and will be removed
in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function get_feature_names is d
eprecated; get_feature_names is deprecated in 1.0 and will be removed
in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

In [20]:
```
decision_tree = DecisionTreeClassifier(random_state = 69)
decision_tree.fit(X_train_clean, y_train)

score_maxtrix_printer(decision_tree, X_train_clean, y_train, X_test_c
```

```
    Train Accuracy: 0.9955028724427462
    Train Precision: 0.996568487274807
    Test Accuracy: 0.7287117903930131
    Test Precision: 0.7751388032078964

/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix
is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0
and will be removed in 1.2. Use one of the class methods: ConfusionMa
trixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimato
r.
  warnings.warn(msg, category=FutureWarning)
```



Unsurprisingly the model is severly overfit with an accuracy score of 99% and precision score of 99% on the training set in comparison to the accuracy score of 70% and precision score of 74% on our testing set.

Grid Search for Model 2 Optimal Parameters

We utilized Grid Search to find the optimal parameters for our Decision Tree model. This was done to solve the overfitting that was present in the previous iteration.

In [21]:
```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train_clean, y_train)
```

Out[21]:
```
DecisionTreeClassifier()
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [22]:

```
param_dict = {
    "criterion":['gini', 'entropy'],
    "max_depth":[1, 10, 25, 50],
    "min_samples_split":range(1, 10),
    "min_samples_leaf":range(1, 10)
}
```

In [23]:

```
tw = GridSearchCV(estimator=decision_tree,
                  param_grid=param_dict,
                  cv=5,
                  verbose=1,
                  n_jobs=-1)
tw.fit(X_train_clean, y_train)
```

```
Fitting 5 folds for each of 648 candidates, totalling 3240 fits
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/mo
del_selection/_validation.py:378: FitFailedWarning:
360 fits failed out of a total of 3240.
The score on these train-test partitions for these parameters will be
set to nan.
If these failures are not expected, you can try to debug them by sett
ing error_score='raise'.

Below are more details about the failures:
----------------------------------------------------------------------
-----------
360 fits failed with the following error:
Traceback (most recent call last):
  File "/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sk
learn/model_selection/_validation.py", line 686, in _fit_and_score
    estimator.fit(X_train, y_train, **fit_params)
  File "/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sk
learn/tree/_classes.py", line 969, in fit
    super().fit(
  File "/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sk
learn/tree/_classes.py", line 265, in fit
    check_scalar(
  File "/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sk
learn/utils/validation.py", line 1480, in check_scalar
    raise ValueError(
ValueError: min_samples_split == 1, must be >= 2.

  warnings.warn(some_fits_failed_message, FitFailedWarning)
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/mo
del_selection/_search.py:953: UserWarning: One or more of the test sc
ores are non-finite: [       nan 0.61873192 0.61873192 0.61873192 0.6
1873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
```

```
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.76666914 0.76659115
 0.76703306 0.76716302 0.7667731  0.76640917 0.76695505 0.76659115
        nan 0.76612323 0.76609726 0.76609725 0.76578531 0.76638317
 0.76599325 0.76617523 0.76635717        nan 0.76666912 0.76700705
 0.76713702 0.76664312 0.76711102 0.76705903 0.76661712 0.767267
        nan 0.76672108 0.76646114 0.76659111 0.76635717 0.76630517
 0.76690306 0.76646114 0.76635718        nan 0.76646115 0.76653912
 0.76630517 0.76677308 0.76620119 0.76643517 0.76643516 0.76674709
        nan 0.76716298 0.76705902 0.76679905 0.76718897 0.7666171
 0.76692903 0.76666907 0.76695501        nan 0.76625317 0.76651313
 0.76612317 0.76630516 0.76625315 0.76651314 0.76638316 0.76653911
        nan 0.76638317 0.76651314 0.76596726 0.76622721 0.76648715
 0.76599325 0.76633119 0.76651315        nan 0.7662532  0.7658113
 0.76591528 0.76617521 0.76599325 0.76594126 0.76599325 0.76599325
        nan 0.782838   0.77974466 0.77979659 0.77982259 0.77982264
 0.77886083 0.77969272 0.78068045        nan 0.77480552 0.77381772
 0.77496147 0.77376578 0.77542947 0.7751435  0.77631338 0.77691122
        nan 0.77964057 0.77938061 0.77938065 0.77990054 0.77953658
 0.77896473 0.77995257 0.78065438        nan 0.77641722 0.77600131
 0.77602726 0.77662513 0.77652118 0.77618329 0.77561134 0.77680713
        nan 0.77930263 0.77914675 0.77909469 0.7796666  0.7797186
 0.77956263 0.77878278 0.77927665        nan 0.77940664 0.77964066
 0.77878276 0.77927666 0.77906877 0.77878275 0.77984856 0.77948462
        nan 0.7809663  0.78135622 0.78104428 0.78138219 0.7812003
 0.78029041 0.78133023 0.78088832        nan 0.7818241  0.78205806
 0.78096628 0.78198005 0.78107023 0.78159015 0.78205805 0.782266
        nan 0.7808103  0.78094027 0.7807583  0.78104426 0.78088827
 0.78062835 0.78091431 0.78094029        nan 0.77826287 0.77412973
 0.77340181 0.77384373 0.77438965 0.77350585 0.77553351 0.77457166
        nan 0.76781288 0.76760494 0.76841075 0.76934658 0.76924262
 0.7702045  0.77171217 0.77236212        nan 0.77605328 0.77644321
 0.77602726 0.77659915 0.77535145 0.77475357 0.77688513 0.77665119
        nan 0.77192014 0.77231004 0.77262195 0.77280394 0.77251796
 0.77106225 0.77295993 0.77316782        nan 0.77776894 0.77711909
 0.77748303 0.77636524 0.77753499 0.77735299 0.77680714 0.77758701
        nan 0.77743107 0.7769631  0.77771697 0.77693713 0.77654718
 0.77730107 0.77743103 0.77766499        nan 0.7793546  0.78062834
 0.77979652 0.78049839 0.7797705  0.77964052 0.77945859 0.77969253
        nan 0.7803684  0.7797965  0.78083629 0.78091425 0.7806283
 0.7802384  0.7800824  0.78049835        nan 0.77979652 0.78008248
 0.78081028 0.77997847 0.78005645 0.77990048 0.77984852 0.7802904
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
        nan 0.61873192 0.61873192 0.61873192 0.61873192 0.61873192
 0.61873192 0.61873192 0.61873192        nan 0.76448558 0.76458953
 0.76500548 0.76492748 0.7644596  0.76516145 0.76482352 0.76503148
        nan 0.76482351 0.76432961 0.76479751 0.76461557 0.76516146
 0.76547339 0.76495351 0.76477155        nan 0.76479752 0.76518744
```

```
 0.7649015  0.76500548 0.76453759 0.76505749 0.76477155 0.76503148
        nan 0.76531742 0.76523944 0.76552537 0.76547338 0.7653954
 0.7653434  0.76521344 0.76529143        nan 0.76497951 0.7650575
 0.76484954 0.76500551 0.76479754 0.76471957 0.76495351 0.76482354
        nan 0.76414764 0.76425162 0.76427761 0.76427762 0.76422562
 0.7643816  0.76440759 0.76393968        nan 0.76354975 0.76344578
 0.76349777 0.76354976 0.76354977 0.76362774 0.76349776 0.76373172
        nan 0.76347174 0.76349773 0.76349774 0.76326378 0.76347174
 0.7636277  0.76339375 0.76354972        nan 0.76276986 0.76279586
 0.76289984 0.76292583 0.7630818  0.76300382 0.76276986 0.76282186
        nan 0.78023848 0.77940659 0.77810692 0.77802897 0.77579337
 0.77691116 0.77711912 0.77750908        nan 0.77277798 0.77308985
 0.77376575 0.77503952 0.77503957 0.77498758 0.77561141 0.77737904
        nan 0.78026445 0.77927669 0.77784689 0.77922467 0.77945864
 0.777821   0.77919867 0.7792247         nan 0.77633931 0.77792498
 0.77696317 0.77607939 0.7756894  0.77685921 0.77758704 0.77722313
        nan 0.78086233 0.78068032 0.78018644 0.77992648 0.77974452
 0.78003048 0.78008247 0.78044641        nan 0.77787287 0.77873072
 0.77844475 0.77789889 0.77808085 0.77880868 0.77789887 0.77831483
        nan 0.77891273 0.77906868 0.77943263 0.77990051 0.77930261
 0.7799525  0.7799785  0.7794326         nan 0.78008252 0.78052442
 0.78003054 0.77987458 0.78023846 0.78034248 0.77995258 0.77953663
        nan 0.78120029 0.78034246 0.78086236 0.77995256 0.78013449
 0.78055041 0.78060239 0.7804464         nan 0.77535149 0.77337581
 0.77225806 0.77264805 0.77283    0.772674   0.77293402 0.77368776
        nan 0.76934663 0.7688787  0.76869669 0.77082829 0.77030843
 0.77147817 0.77259603 0.77550738        nan 0.77545536 0.77444159
 0.77600119 0.77607925 0.77467551 0.77418167 0.7749615  0.77537743
        nan 0.772258   0.77272594 0.77314186 0.77275189 0.77376568
 0.77360976 0.77272593 0.77418162        nan 0.77782089 0.77717098
 0.77688505 0.77839276 0.77818483 0.77735298 0.77818481 0.77841872
        nan 0.7762091  0.77649503 0.77516935 0.77633911 0.77644308
 0.77678101 0.77709289 0.77620912        nan 0.77813284 0.77745699
 0.77914662 0.77823682 0.77909459 0.77800284 0.77776886 0.77854873
        nan 0.77932866 0.77927664 0.78013449 0.77995253 0.77977063
 0.77922466 0.7797706  0.7797966         nan 0.77961455 0.77969252
 0.77971854 0.7798745  0.77932864 0.77935459 0.77964054 0.77990049]
  warnings.warn(
```

Out[23]:

```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(), n_jobs=
-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [1, 10, 25, 50],
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(1, 10)},
             verbose=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [24]:

```
print(tw.best_params_)
print(tw.best_estimator_)
print(tw.best_score_)
```

```
{'criterion': 'gini', 'max_depth': 25, 'min_samples_leaf': 1, 'min_sa
mples_split': 2}
DecisionTreeClassifier(max_depth=25)
```

```
0.7828379963609358
```

Here we found that the best parameters for our Decision Tree model are criterion = 'gini', max_depth = 25, min_samples_leaf = 7

Kept getting different min_samples_split value without results changing so it seems criterion, max_depth, & and min_samples_leaf give us a consistent enough result that min_samples_split does not need to be manipulated by us.

Model 2 (Decision Tree) with Optimized Parameters

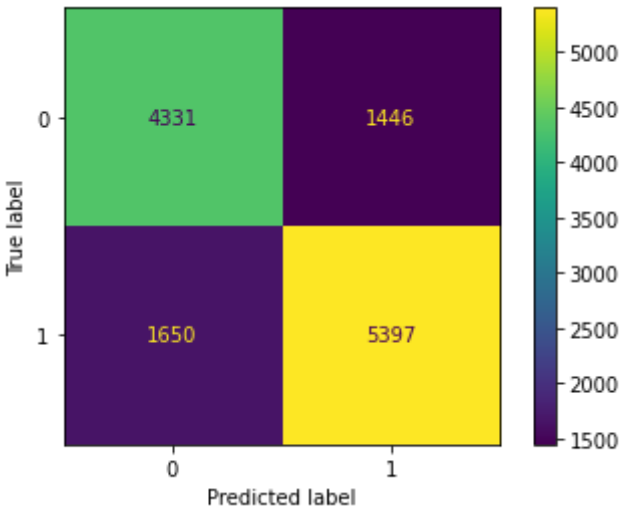
Now that we have our optimal parameters we are building the second model utilizing those parameters.

In [25]:
```
decision_tree_better = DecisionTreeClassifier(random_state = 69, crit
                                              max_depth = 25, min_sam
decision_tree_better.fit(X_train_clean, y_train)

score_maxtrix_printer(decision_tree_better, X_train_clean, y_train, X
```

```
    Train Accuracy: 0.8749382619771764
    Train Precision: 0.8799249530956847
    Test Accuracy: 0.7585776668746101
    Test Precision: 0.7886891714160456

/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix
is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0
and will be removed in 1.2. Use one of the class methods: ConfusionMa
trixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimato
r.
  warnings.warn(msg, category=FutureWarning)
```



With our optimized parameters, this decision tree performed much better than our first attempt, with new training accuracy and precion scores of nearly 88% and a test accuracy and precion scores of 74%. The problem with our overfitted first model has now been resolved!

first model has now been resolved!

Feature Importance & Model 3 (Logistic Regression) Using Top Predictors

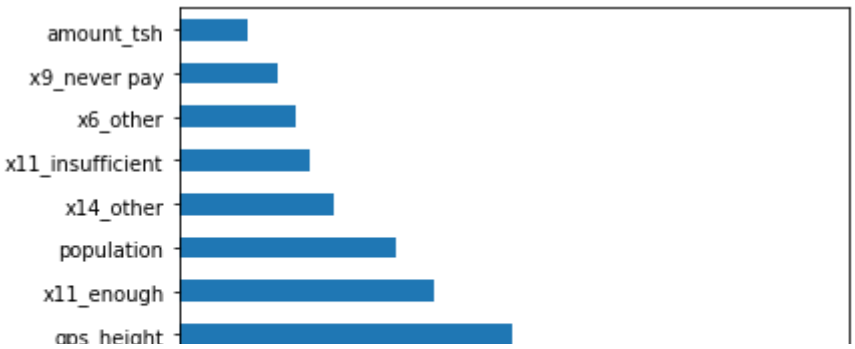
Next, we used the ExtraTreesClassifier() to identify the features with the highest importance.

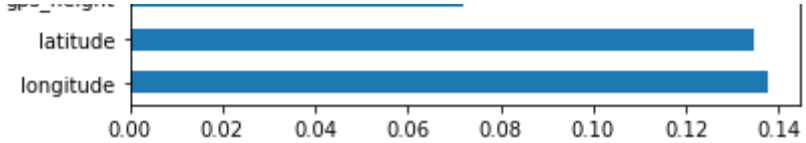
In [26]:
```
model = ExtraTreesClassifier()
model.fit(X_train_clean, y_train)

print(model.feature_importances_) #use inbuilt class feature_importar

#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index = X_tr
feat_importances.nlargest(10).plot(kind = 'barh')
plt.show()
```

```
[1.48556864e-02 7.18152849e-02 1.37875265e-01 1.34827466e-01
 4.68684010e-02 1.02738217e-02 1.34864404e-03 1.04510529e-02
 5.63635630e-03 8.09445884e-04 1.15772634e-02 2.71823471e-03
 1.23470673e-03 5.13953742e-03 3.31974088e-03 4.66951464e-03
 4.78047420e-03 5.96480929e-03 4.45252070e-03 3.90971045e-03
 4.60611287e-03 7.33967236e-04 2.58015323e-03 1.07528461e-02
 2.41840152e-03 3.33341632e-03 3.44864682e-03 1.54511255e-03
 2.83862312e-03 1.95972354e-03 2.62996510e-03 2.35973098e-03
 2.19113230e-03 2.65672227e-03 3.15727644e-03 2.26675281e-03
 2.56711141e-03 2.48475540e-03 1.48387341e-03 1.78931954e-03
 2.80148148e-03 1.30114188e-03 1.51079679e-03 1.39377871e-03
 6.88897360e-03 2.03708758e-03 2.65733538e-03 5.36179660e-03
 3.63055346e-03 1.37635314e-02 9.62245920e-03 6.96104266e-03
 7.04389592e-03 1.23727024e-02 1.14867125e-02 5.96402937e-03
 8.52243017e-03 3.96374729e-03 2.53615147e-02 8.31681753e-04
 7.70533657e-03 4.50581035e-04 8.28231302e-04 1.91494889e-04
 1.49440058e-03 3.00052138e-03 2.61231936e-04 3.02007015e-04
 7.36195134e-03 1.37512771e-03 2.96854842e-03 1.63097622e-03
 3.25408704e-03 8.95042869e-04 1.37892905e-03 3.18920415e-04
 4.73353688e-03 6.61128945e-03 2.12546301e-02 4.44021794e-03
 1.58380818e-03 9.32992298e-03 7.83174751e-03 7.51807470e-04
 5.57960810e-05 1.19164037e-03 5.63903183e-03 1.30866117e-03
 8.44351744e-03 4.43025151e-03 5.51502831e-02 2.82862286e-02
 1.38074658e-02 1.71548082e-03 1.48707464e-03 1.79762949e-03
 8.10748400e-03 6.42178313e-04 3.97113850e-03 5.13044596e-03
 8.15553628e-03 9.08996610e-03 1.26989495e-04 6.48112467e-03
 5.84978289e-04 1.42546691e-02 1.21722217e-02 3.17458890e-05
 8.14796549e-03 2.48587527e-03 3.35419993e-02]
```





In [27]:
```
# gatheringtop feature together

importance_col = ['longitude', 'latitude', 'gps_height', 'quantity',
                  'status_group', 'population', 'waterpoint_type',
                 'extraction_type_class', 'payment_type', 'amount_tsh
df_importance = df2[importance_col]
```

As expected the location of the well is significant. This is shown by the latitude longitude and gps_height being the three highest ranked features. We then used our top 10 most important features to predict our model using logistic regression.

In [28]:
```
y_Ext = df_importance.status_group
X_Ext = df_importance.drop('status_group', axis = 1)

X_train_Ext, X_test_Ext, y_train_Ext, y_test_Ext = train_test_split(X
                                                    test_size = 0
```

In [29]:
```
X_train_clean_Ext = split_join(X_train_Ext)
X_test_clean_Ext = split_join(X_test_Ext)
```

```
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function get_feature_names is d
eprecated; get_feature_names is deprecated in 1.0 and will be removed
in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function get_feature_names is d
eprecated; get_feature_names is deprecated in 1.0 and will be removed
in 1.2. Please use get_feature_names_out instead.
  warnings.warn(msg, category=FutureWarning)
```

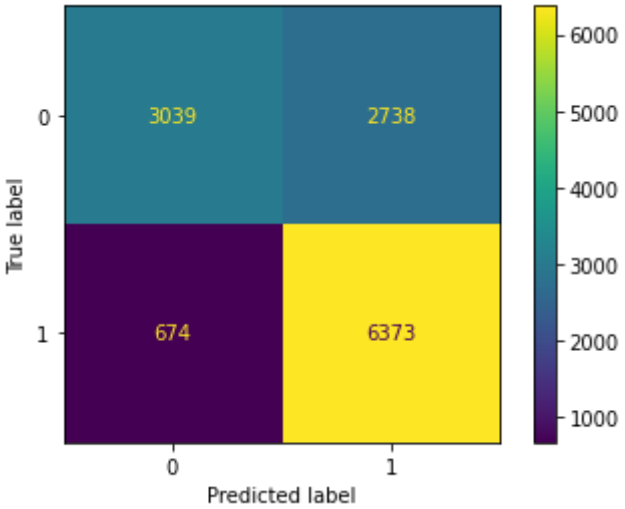
In [30]:
```
logreg_clf_Ext = LogisticRegression(random_state = 69, max_iter = 100
logreg_model_Ext = logreg_clf_Ext.fit(X_train_clean_Ext, y_train_Ext)

score_maxtrix_printer(logreg_model_Ext, X_train_clean_Ext, y_train_Ex
```

```
Train Accuracy: 0.7301983415217448
Train Precision: 0.6905649555491826
Test Accuracy: 0.7339363693075484
Test Precision: 0.6994841400504884

/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix
is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0
and will be removed in 1.2. Use one of the class methods: ConfusionMa
trixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimato
r.
```

```
warnings.warn(msg, category=FutureWarning)
```



We recorded that the top 10 predictors we recived from our ExtraTreesClassifier() had a 73% accuracy score and 69% precision score of our test data when using a LogisticRegression(). This was with only 10 variables which shows how significant they are, however we can see that using just these yieled lower scores than our decision tree model.

Model 4 (Logistic Regression) Utilizing All Predictors

Next, we ran a new Logistic Regression model which utilized all predictor columns available. We decided to make a logistic regression model next to see how the accuracy and precision would compare with our decision tree model.

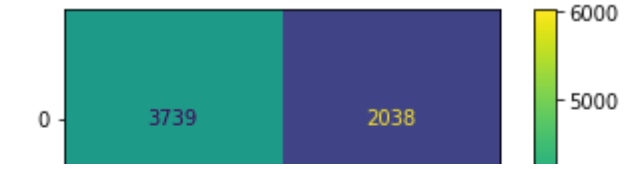
In [31]:

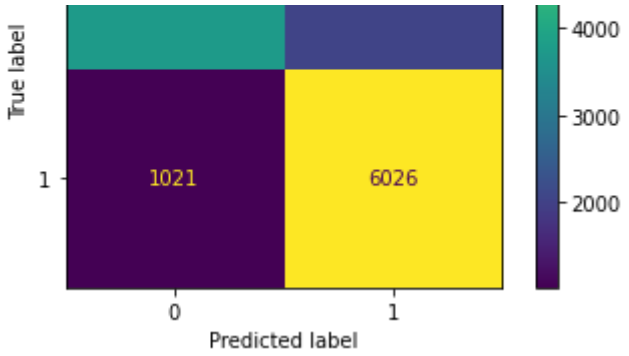
```
# All predictors were utilized in this model
logreg_clf = LogisticRegression(random_state = 69, max_iter = 1000)
logreg_model = logreg_clf.fit(X_train_clean, y_train)

score_maxtrix_printer(logreg_model, X_train_clean, y_train, X_test_cl
```

```
Train Accuracy: 0.7608723907561933
Train Precision: 0.743065272861964
Test Accuracy: 0.7614628820960698
Test Precision: 0.7472718253968254
```

```
/Users/karaoglan/opt/anaconda3/lib/python3.8/site-packages/sklearn/ut
ils/deprecation.py:87: FutureWarning: Function plot_confusion_matrix
is deprecated; Function `plot_confusion_matrix` is deprecated in 1.0
and will be removed in 1.2. Use one of the class methods: ConfusionMa
trixDisplay.from_predictions or ConfusionMatrixDisplay.from_estimato
r.
  warnings.warn(msg, category=FutureWarning)
```





With all predictors, both our training and testing sets' metrics improved. Test Accuracy had gone up to 76% and precision is slightly lower at 74%.

Grid Search for Model 5 Optimal Parameters

For our final individual model, we again utilized Grid Search to find the optimal parameters for a kNN model. We too wanted to compare this model to see how the score would change from our decision tree as well as our logistic regression model.

In [32]:
```
knn_model = KNeighborsClassifier()
knn_model.fit(X_train_clean,y_train)
grid = {

    'n_neighbors': range(1,15),
    'metric':['manhattan','minkowski'],
    'weights' :['distance','uniform']

}
```

In [33]:
```
gs = GridSearchCV(estimator= knn_model,param_grid=grid,cv=5,verbose=1
gs.fit(X_train_clean,y_train)
```

```
Fitting 5 folds for each of 56 candidates, totalling 280 fits
```

Out[33]:
```
GridSearchCV(cv=5, estimator=KNeighborsClassifier(), n_jobs=-
1,
             param_grid={'metric': ['manhattan', 'minkowski'],
                         'n_neighbors': range(1, 15),
                         'weights': ['distance', 'uniform']},
             verbose=1)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [34]:
```
gs.best_params_
```

Out[34]:
```
{'metric': 'manhattan', 'n_neighbors': 13, 'weights': 'distance'}
```

Here we found that the best parameters for our kNN model are n_neighbors = 14, metric = 'manhattan', weights = 'distance'

Model 5 (K Nearest Neighbors) with Optimized