Final Project Submission

Please fill out:

- Student name: Abigail Campbell
- Student pace: Flex
- Scheduled project review date/time:
- Instructor name: Morgan Jones
- Blog post URL:

Business Problem

Tanzania has water pumps installed throughout the country in order to make pottable water accessible to it's residents. Maintaining these water pumps can prove to be a challenge, since many are in remote areas that are difficult to monitor.

The goal of this project is to build a model that can predict which water pumps are in need of repair in order to efficiently dispatch technicians to the sites most in need and provide as many people as possible with clean water.

The Data

A dataset of Tanzania water pumps is maintained by Taarifa in a waterpoints dashboard by aggregating data from the Tanzania Ministry of Water. This data set keeps track of information regarding the location of the water pump, information about the pump (contruction year, extraction type, etc.), the people involved in the pump (installers, management, population), as well as the information regarding the source of water feeding the pump.

Additionally, the training test set provided has another test set containing the id nubmer of every pump and wheter it is functional, functional but in need of repair, or non-functional. This will allow us to train and validate our model.

Prior to cleaning, there are 59,400 water pumps in the training data set.

Setup

Import relevant packages

In [1]:
```
import sqlite3
import pandas as pd
import numpy as np
import scipy.stats as stats
import statsmodels.api as sm
import json

from sklearn.preprocessing import OneHotEncoder
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import f1_score, mean_squared_error, r2_score, roc_curv
from sklearn.ensemble import RandomForestClassifier

import seaborn as sns
import matplotlib.pyplot as plt
# plt.style.use('seaborn-v0_8-whitegrid')
%matplotlib inline
```

Load and Clean Data

Training values set

Preview Data

In [2]:
```
training_values = pd.read_csv('data/Training_set_values.csv')
print(f'number of rows: {len(training_values)}')
training_values.head()
```

```
number of rows: 59400
```

Out[2]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	none
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Zahanati
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	Kwa Mahundi
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Zahanati Ya Nanyumbu
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	Shuleni

5 rows × 40 columns

Investigate Missing Data

In [3]:
```
training_values.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
```

Clean data

Investigate funder, installer, subvillage, public_meeting, scheme management, scheme_name, permit

- funder:
  - 3635 missing values (6.1%)
  - Not Known, Unknown, categories
  - 0 category (same as none?)
- installer:
  - 3655 missing values (6.1%)
  - uknown, Unknown, Not known categories
- subvillage:
  - 371 missing values (0.6%)
  - relatively few missing values
  - 19287 unique values - too many to consolidate inot meaningful categories
    - enough other location based categories
    - drop column
- public_meeting:
  - 3334 missing values (5.6%)
  - binary true/false
  - drop missing rows - want to keep it binary
- scheme management:
  - 3877 missing values (6.5%)
  - will likeley create an "unknown" column
  - one occurance of None - remove (unable to have an example in both test and training sets)
- scheme name:
  - 28166 missing values (47.4%)
  - too much missing data - drop this
- permit:
  - 3056 missing values (5.1%)
  - binary true/false
  - drop missing rows - want to keep it binary

After cleaning, 53,277 data points (89% of original set)

permit, and public_meeting columns:

- remove NaN values

In [4]:
```
training_values = training_values.dropna(axis=0, subset=['permit', 'public_m
```

Drop unnecessary columns:

columns with majority unique values or data that is not relevant or redundant

- scheme_name, wpt_name, date_recorded, subvillage, ward
- quantity_group and quantity are the same, drop one
- extraction_type, extraction_type_group, extraction_type_class are varying hierarchies of the same feature. take the middle one.
- source, source_class, source_type are varying hierarchies of the same feature. take the middle one.
- waterpoint_type and waterpoint_type_group: virtually the same, except waterpoint_type splits communal standpipe into communal standpipe and communal statepipe multiple. keep waterpoint_type.
- payment and payment_type are identical. drop payment_type.

In [5]:
```
drop_cols = ['scheme_name', 'wpt_name', 'date_recorded', 'subvillage', 'ward
training_values = training_values.drop(columns=drop_cols, axis=1)
```

Manage "Unknown" and "Other" Categories

- Remove NaN values or convert to "unknown".
- Combine any "not known" variations into one "unknown" column
- Combine categories with low value counts into a single "other" column

In [6]:
```
## clean up permit, and public_meeting columns
training_values = training_values.dropna(axis=0, subset=['permit', 'public_m
```

In [7]:
```
## clean up funder column
# combine Not Known, Unknown, and Nan categories into one 'Unknown'
training_values['funder'] = training_values['funder'].fillna('Unknown')
training_values['funder'] = training_values['funder'].map(lambda x: 'Unknown

# create an "other" column for all funders with less than 10 wells
#     Note: 156 was chosen as it combined sufficient small categories withou
small_funders = []
for key, value in training_values['funder'].value_counts().to_dict().items()
    if value < 16:
        small_funders.append(key)

training_values['funder'] = training_values['funder'].map(lambda x: 'Other'
```

In [8]:
```
ean up installer column
bine uknown, Unknown, Not known, and Nan categories into one 'Unknown'
ing_values['installer'] = training_values['installer'].fillna('Unknown')
ing_values['installer'] = training_values['installer'].map(lambda x: 'Unknown
ing_values['installer'] = training_values['installer'].map(lambda x: 'Unknown

ate an "other" column for all installers with less than 15 wells
 Note: 15 was chosen as it combined sufficient small categories without makin
_installers = []
ey, value in training_values['installer'].value_counts().to_dict().items():
f value < 15:
   small_installers.append(key)

ing_values['installer'] = training_values['installer'].map(lambda x: 'Other'
```

In [9]:
```
## clean up scheme_management column
# convert Nan values into one 'Unknown'
training_values['scheme_management'] = training_values['scheme_management'].
```

Remove categories with only 1 or 2 occurances

A few columns have categories with very few examples, not enough to be reliably split between the training and test data set, so those categories will be removed

In [10]:
```
# Scheme management: remove one row with "None"
training_values = training_values[training_values['scheme_management'] != 'N
```

In [11]:
```
# lga: remove one row with "Nyamagana"
training_values = training_values[training_values['lga'] != 'Nyamagana']
```

Investigate value counts of final columns

Visually inspect the number of categories in each column to confirm that no column will create a large number of columns

- any categorical columns should have a "managable" number of columns
- any numberical columns do not need to have low value counts, as they will not be encoded

In [12]:
```
for col in list(training_values.columns):
    print(f'{col}: {len(training_values[col].value_counts())}')
# not dummy variables
# id, gps_height, population, construction_year
```

```
id: 53279
amount_tsh: 91
funder: 302
gps_height: 2426
installer: 304
longitude: 51779
latitude: 51781
num_private: 61
basin: 9
region: 21
region_code: 27
district_code: 20
lga: 121
population: 1006
public_meeting: 2
recorded_by: 1
scheme_management: 12
permit: 2
construction_year: 55
extraction_type_group: 13
management: 12
management_group: 5
payment: 7
water_quality: 8
quality_group: 6
quantity_group: 5
source: 10
source_type: 7
waterpoint_type: 7
```

Training Set Labels

Load the training set lables and restrict to only contain ids that are included in the cleaned training values dataset

In [13]:
```
# load data
training_labels = pd.read_csv('data/Training_set_labels.csv')

# investigate values
training_labels.status_group.value_counts()
```

Out[13]:
```
functional                 32259
non functional             22824
functional needs repair     4317
Name: status_group, dtype: int64
```

In [14]:
```
# create a mask for ids present in the filtered training values set
mask = training_labels['id'].isin(list(training_values.id))

# apply the mask
training_labels = training_labels[mask]
print(len(training_values))
```

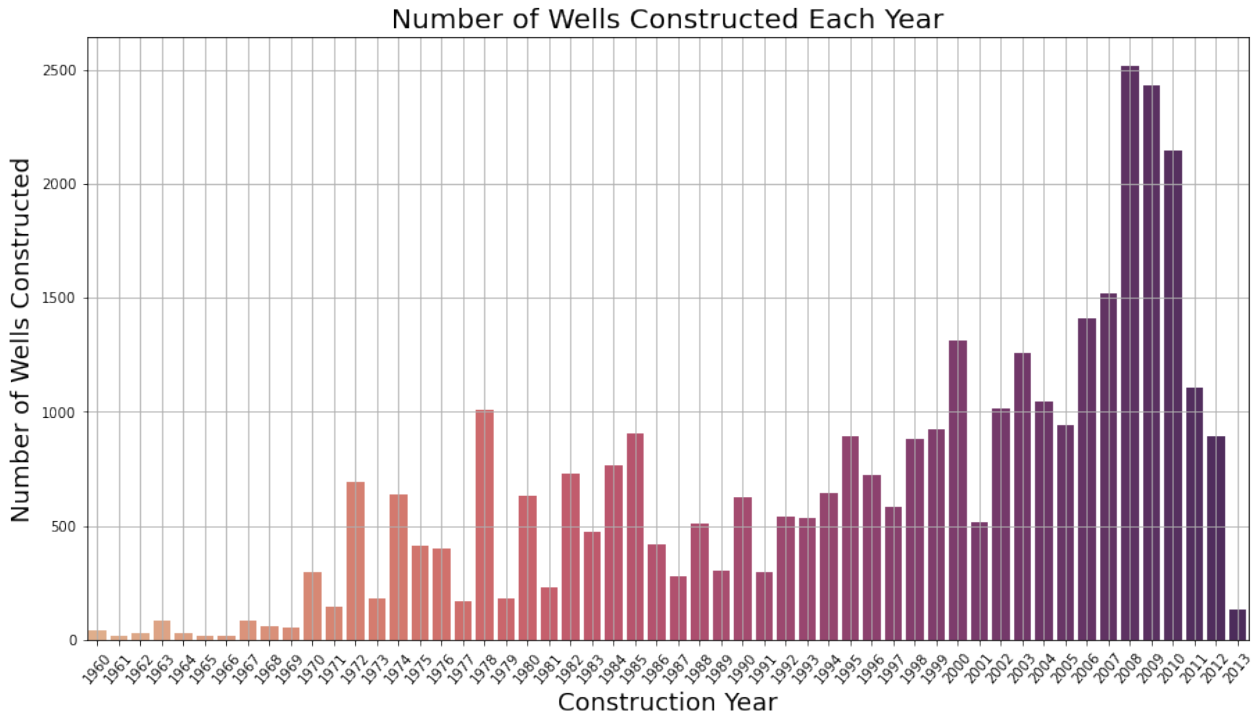
```
53279
```

Full Data Set Visualizations

Initial visualizations to get a look into the data set

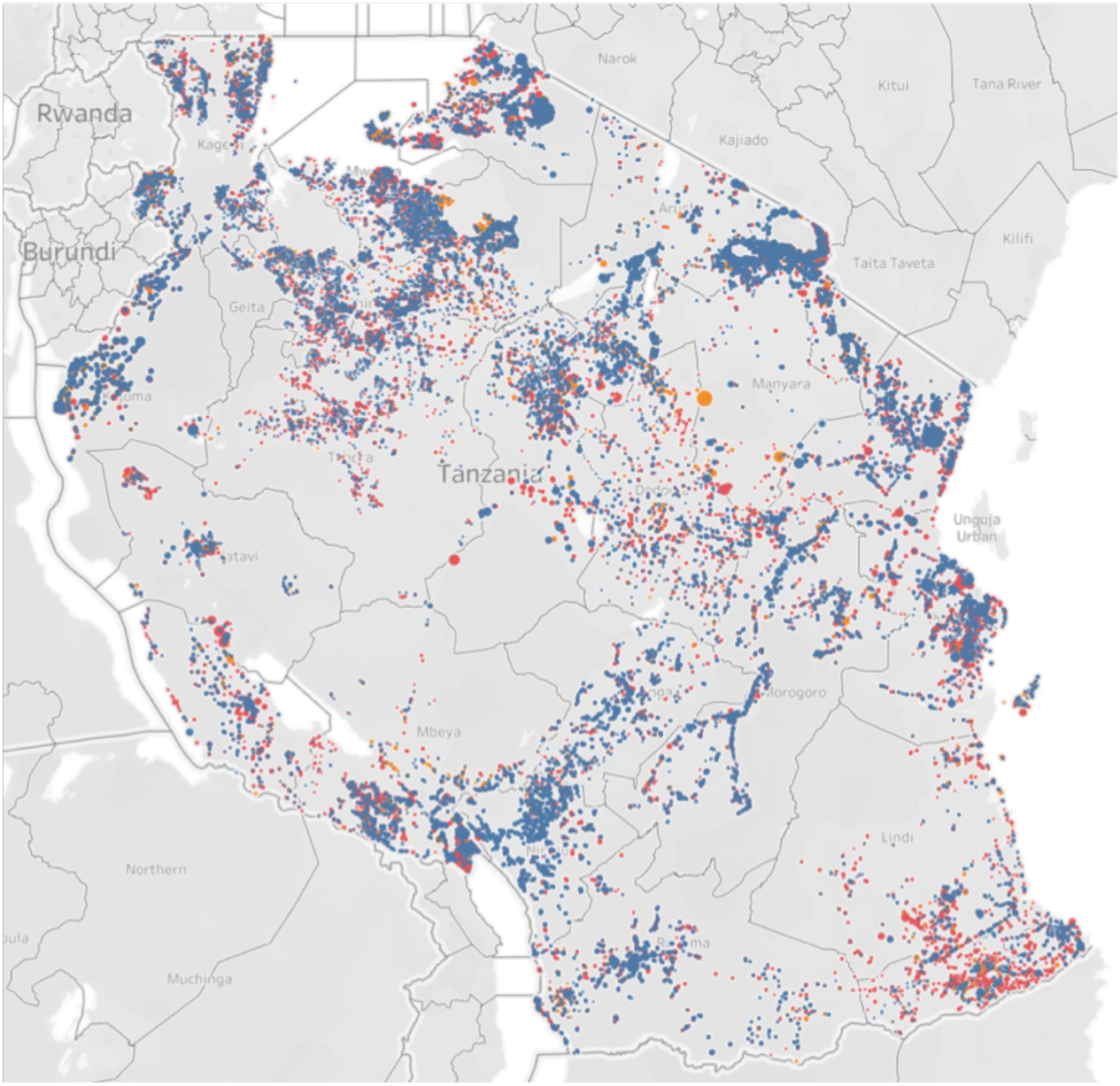
View the number of wells constructed per year

In [15]:
```
# group by year constucted
years = training_values.construction_year.value_counts().index[1:]
year_counts = training_values.construction_year.value_counts().values[1:]

# count the number of wells in each year
year_data = {
    'year': training_values.construction_year.value_counts().index[1:],
    'count': training_values.construction_year.value_counts().values[1:]
}
year_df = pd.DataFrame(year_data)

plt.figure(figsize=(15,8))
sns.barplot(data=year_df, x='year', y='count', palette='flare')
plt.xticks(rotation=50)
plt.xlabel('Construction Year', fontsize=18)
plt.ylabel('Number of Wells Constructed', fontsize=18)
plt.title('Number of Wells Constructed Each Year', fontsize=20)
plt.grid()
```



Locations of wells across Tanzania

- map generated using Tableau using well coordinates and inserted into this notbook
  - blue: functional
  - orange: functional, needs repair
  - red: non functional



Data Prep

Prepare the data to be compatible with classification

Split the given training set into a train set and a test set

In [16]:
```
# create X and y variables
X = training_values.drop(columns=['id'], axis=1)
y = training_labels['status_group']
```

In [17]:
```
# pick a random seed to standardize the random selection
SEED = 20
```

In [18]:
```
# Split X and y into test and train sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, ra
```

Encode categorical data as binary values

In [19]:
```
# identify the non categorical columns that will not be included in the one
non_categorical_cols = ['amount_tsh', 'gps_height', 'population', 'construct
```

In [20]:
```
# initialize the One Hot Encoder
ohe = OneHotEncoder()
```

In [21]:
```
# create a function to perform the one hot encoding of the X data
def encode_X(X, non_categorical_cols):
    """
    function to encode a given X data set
        Inputs:
            - X: dataframe to encode
            - non_categorical_cols: list of column names to exclude from enc
        Outputs:
            - X_ohe_final: final encoded dataframe containing both encoded c
    """

    # create two dataframes: categorical and non categorical
    categorical_df = X.drop(columns=non_categorical_cols, axis=1)
    non_categorical_df = X[non_categorical_cols]

    # encode the categorical dataframe and convert back to a dataframe
    X_ohe_categorical = ohe.fit_transform(categorical_df).toarray()
    ohe_df = pd.DataFrame(X_ohe_categorical, columns=ohe.get_feature_names(c

    # combine the encoded categorical dataframe and the non categorical data
    X_ohe = pd.concat([ohe_df, non_categorical_df], axis=1)

    # reset the index
    ohe_df['ind'] = categorical_df.index
    ohe_df = ohe_df.set_index('ind')
    ohe_df.head()

    # combine the encoded categorical dataframe and the non categorical data
    X_ohe_final = pd.concat([ohe_df, non_categorical_df], axis=1)

    # check the number of columns in train and test sets to ensure they are
    print(len(X_ohe_final.columns))

    return X_ohe_final
```

In [22]:
```
# encode the train and test data sets
X_train_ohe = encode_X(X_train, non_categorical_cols)
X_test_ohe = encode_X(X_test, non_categorical_cols)
```

```
882
882
```

Functions for Evaluating Model Performance

Model evaluation will occur several times throughout this notebook. These functions allow the model perfromance evaluation to be standareized easily applied for each model iteration

4 functions are defined:

- create_y_ohe: encodes the data labels to allow metrics to be calculated for each label (there are 3 labels)
- performance_metrics: calcualtes performance metrics between a data set of test data labels and their predicted values
- evaluate_sbuset_performance: evaluates the performance of each subset of labels
- evaluate_model_performance: evaluate the performance of a model

In [23]:
```
tion to perform one hot encoding on the y results
eate_y_ohe(y, columns):
"
nction to encode a given y data set
  Inputs:
      - y: dataframe to encode
      - columns: list of column names included in the encoding
  Outputs:
      - y_ohe_df: final encoded dataframe
"
initialize the encoder
e = OneHotEncoder()

create a y dataframe
df = pd.DataFrame(y, columns=columns)

encode the dataframe
ohe = ohe.fit_transform(y_df).toarray()

converte the encoded y back into a dataframe
ohe_df = pd.DataFrame(y_ohe, columns=ohe.get_feature_names(y_df.columns))

turn y_ohe_df


tional subset accuracy
rformance_metrics(y_test, y_pred):
"
nction to calcualte performance metrics between a data set of test data labe
  Inputs:
      - y_test: known labels for the test data set
      - y_pred: predicted labels for the test data set
  Outputs:
      - results: dictionary containing performance metircs including:
          - accuracy, precision, recall, mean-squared_error, f1, and auc
"

calculate metrics
curacy = accuracy_score(y_test, y_pred)
```

```
ecision = precision_score(y_test, y_pred)
call = recall_score(y_test, y_pred)
e = mean_squared_error(y_test, y_pred)
 = f1_score(y_test, y_pred)

lse_positive_rate, true_positive_rate, thresholds = roc_curve(y_test, y_pred)
c_auc = auc(false_positive_rate, true_positive_rate)

organize metrics into a dictionary
sults = {
  'accuracy': round(accuracy, 4),
  'precision': round(precision, 4),
  'recall': round(recall ,4),
  'mse': round(mse, 4),
  'f1': round(f1, 4),
  'auc': round(roc_auc, 4)


turn results


aluate_sbuset_performance(y_test_ohe_df, y_pred_ohe_df):
"
nction to evaluate the performance of each subset of labels
  Inputs:
      - y_test_ohe_df: encoded dataframe of known labels for the test data se
      - y_pred_ohe_dt: encoded dataframe of predicted labels for the test dat
  Outputs:
      - subset_results: dictionary containing performance metircs for each su
          - each set of results within a key named for the subset
              'subset': {
                  accuracy: val,
                  precision: val,
                  recall: val,
                  mse: val,
                  auc: val
              }
"
define the subsets withing the labels
bsets = ['status_group_functional', 'status_group_functional needs repair',

bset_results = {}

loop through each subset label and create
r subset in subsets:

  # if one subset is not present in the predicted labels given, create a colu
  try:
      y_pred = y_pred_ohe_df[subset]
  except KeyError:
      y_pred_ohe_df[subset] = np.zeros(len(y_pred_ohe_df))

  y_test = y_test_ohe_df[subset]
```

```
  # calculate the subset results
  subset_results[subset] = performance_metrics(y_test, y_pred)

turn subset_results


aluate_model_performance(y_test, y_pred):
"
nction to evaluate the performance of a model
  Inputs:
      - y_test: array of known labels for the test data set
      - y_pred: array of predicted labels for the test data set
  Outputs:
      - results: dictionary containing performance metircs for each subset ar
          - each set of results within a key named for the subset
              'subset': {
                  accuracy: val,
                  precision: val,
                  recall: val,
                  mse: val,
                  auc: val
              }
"
convert the label arrays into encoded dataframes in order to calculate the me
test_ohe_df = create_y_ohe(y_test, columns=['status_group'])
pred_ohe_df = create_y_ohe(y_pred, columns=['status_group'])

calculate the performance for each label subset
bset_results = evaluate_sbuset_performance(y_test_ohe_df, y_pred_ohe_df)

average the subset results together to get an evaluation for the whole model
curacy = []
ecision = []
call = []
e = []
 = []
c = []
r subset in subset_results.keys():

  accuracy.append(subset_results[subset]['accuracy'])
  precision.append(subset_results[subset]['precision'])
  recall.append(subset_results[subset]['recall'])
  mse.append(subset_results[subset]['mse'])
  f1.append(subset_results[subset]['f1'])
  auc.append(subset_results[subset]['auc'])

g_accuracy = round(np.array(accuracy).mean(), 4)
g_precision = round(np.array(precision).mean(), 4)
g_recall = round(np.array(recall).mean(), 4)
g_mse = round(np.array(mse).mean(), 4)
g_f1 = round(np.array(f1).mean(), 4)
g_auc = round(np.array(auc).mean(), 4)

sults = {
```

```
    'accuracy': avg_accuracy,
    'precision': avg_precision,
    'recall': avg_recall,
    'auc': avg_auc,
    'mse': avg_mse,
    'f1': avg_f1,
    'subsets': subset_results


turn results
```

Base Model

Train the tree

- the criterion "gini" was chosen, as it is typically faster than "entropy"
- the same random state is used in the decision tree as the train/test split

In [24]:
```
criterion='gini'

# intialize and fit the tree
clf_base = DecisionTreeClassifier(criterion=criterion, random_state=SEED)
clf_base.fit(X_train_ohe, y_train)
```

Out[24]:
```
DecisionTreeClassifier(random_state=20)
```

Use the tree to predict labels for the test data set

In [25]:
```
y_pred_base = clf_base.predict(X_test_ohe)
```

In [26]:
```
X_test_ohe.head()
```

Out[26]:

	funder_0	funder_Aar	funder_Abasia	funder_Acra	funder_Adb	funder_Adp	funder_Adra	funder
47562	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
29566	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
10873	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
31529	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
5826	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

5 rows × 882 columns

Evaluate the base model performance

Which value will be our primary metric?

- the goal of this model is to identify which water wells are not functioning as intended, and therefore need to be repaired or fixed.
- We want to avoid identifying a well as functioning when it is not functioning (false positives)
  - We would rather visit a well that is working rather than not visit a well that isn't, assuming we have the man power available
- Therefore, the metric we will be prioritizing is precision, as it prioritizes minimizing false positives rather than false negatives

In [27]:
```
results = evaluate_model_performance(y_test.values, y_pred_base)

print(f'Model 1 Performance - Average')
print(f'  accuracy = {results["accuracy"]}')
print(f' ** precision = {results["precision"]} **')
print(f' ** f1 = {results["f1"]} **')
print(f'  recall = {results["recall"]}')
print(f'  auc = {results["auc"]}')

print(f'Model 1 Performance - Functional')
print(f'  accuracy = {results["subsets"]["status_group_functional"]["accurac
print(f' ** precision = {results["subsets"]["status_group_functional"]["prec
print(f' ** f1 = {results["subsets"]["status_group_functional"]["f1"]} **')
print(f'  recall = {results["subsets"]["status_group_functional"]["recall"]}
print(f'  auc = {results["subsets"]["status_group_functional"]["auc"]}')
```

```
Model 1 Performance - Average
  accuracy = 0.8359
 ** precision = 0.6419 **
 ** f1 = 0.6404 **
  recall = 0.639
  auc = 0.7464
Model 1 Performance - Functional
  accuracy = 0.7809
 ** precision = 0.7947 **
 ** f1 = 0.7998 **
  recall = 0.805
  auc = 0.7786
```

We have an precision of 80% with no model tailoring

Hyperparemeter Tuning

Tune the model by determining the optimal value for 4 model parameters:

- max_features
- min_samples_split
- min_samples_leaf
- max_features

Tune the model based on functional precision

- this is the precision in identifying whether or not a well is functional
- combines non-functional and functional-needs-repair into one category
  - both pumps will need to be visited
  - not many examples of functional-needs-repair, difficult to train to that metric

Function to perform the tuning

Grid search major - use precision as the metric to define what the best model 4 nested for loops - maybe try another algorigthms - use random forest, XG Boost

Include a classification report (call to scikit learn) - compare to my function

- get a confustion matrix in there (communicate false positives)

Feature Importance:

- at the end of the analysis, calculate feature importance of the best performing model (decision tree vs forest, etc)
- areas for the NGO to focus on to repair and help with longevity
- kinda like analyzing coefficients in linear regression

3 recs:

- model, then 2 most important features

Function to perform the tuning

Define a function to investigate a range of values for a given parameter on the effect they have on the model.

- This function will be used to perform tuning on the base model and determine the best values for each parameter

In [28]:
```
def investigate_parameter(param_name, param_values
```

In [28]:
```
def investigate_parameter(param_name, param_values,
                          max_depth=None, min_samples_leaf=1, min_samples_sp
    """
    function to investigate a range of values for a given parameter on the e

        Inputs:
            - param_name: name of the parameter associated with the range of
            - param_values: range of values to test
            - max_depth: option to set a specific value for the max depth (d
            - max_samples_leaf: option to set a specific value for the max s
            - min_samples_split: option to set a specific value for the max
            - max_features: option to set a specific value for the max featu
    """
    # initialize metric storage
    train_auc = []
    test_auc = []

    train_accuracy = []
    test_accuracy = []

    train_precision = []
    test_precision = []

    train_recall = []
    test_recall = []

    train_f1 = []
    test_f1 = []

    # loop through each value and calculate the performance metrics for the
    for value in param_values:

#         print(value)

        # select the decision tree that uses the range of values specified
        if param_name == 'max_depth':
            dt = DecisionTreeClassifier(criterion=criterion,
                                        max_depth=value,
                                        min_samples_leaf=min_samples_leaf,
                                        min_samples_split=min_samples_split,
                                        max_features=max_features,
                                        random_state=SEED)
        elif param_name == 'min_samples_leaf':
            dt = DecisionTreeClassifier(criterion=criterion,
                                        max_depth=max_depth,
                                        min_samples_leaf=value,
                                        min_samples_split=min_samples_split,
                                        max_features=max_features,
                                        random_state=SEED)
        elif param_name == 'min_samples_split':
            dt = DecisionTreeClassifier(criterion=criterion,
                                        max_depth=max_depth,
                                        min_samples_leaf=min_samples_leaf,
                                        min_samples_split=value,
                                        max_features=max_features
```

```
                                        max_features=max_features,
                                        random_state=SEED)
        elif param_name == 'max_features':
            dt = DecisionTreeClassifier(criterion=criterion,
                                        max_depth=max_depth,
                                        min_samples_leaf=min_samples_leaf,
                                        min_samples_split=min_samples_split,
                                        max_features=value,
                                        random_state=SEED)
        # fit the tree
        dt.fit(X_train_ohe, y_train)

        # predict the train and test labels
        train_pred = dt.predict(X_train_ohe)
        test_pred = dt.predict(X_test_ohe)

        # evaluate model performance for both the train and test data sets
        test_performance = evaluate_model_performance(y_test, test_pred)
        train_performance = evaluate_model_performance(y_train, train_pred)

        # accumulate metrics for each parameter value to plot
        train_auc.append(train_performance["subsets"]["status_group_function
        test_auc.append(test_performance["subsets"]["status_group_functional

        train_precision.append(train_performance["subsets"]["status_group_fu
        test_precision.append(test_performance["subsets"]["status_group_func

        train_recall.append(train_performance["subsets"]["status_group_funct
        test_recall.append(test_performance["subsets"]["status_group_functio

        train_accuracy.append(train_performance["subsets"]["status_group_fun
        test_accuracy.append(test_performance["subsets"]["status_group_funct

        train_f1.append(train_performance["subsets"]["status_group_functiona
        test_f1.append(test_performance["subsets"]["status_group_functional"

    # determine the parameter value with the hightest precision on the test
    max_precision = max(test_precision)
    max_value = param_values[test_precision.index(max_precision)]

    # plot the auc value for train and test results
    plt.figure(figsize=(15, 8))
    plt.plot(param_values, train_auc, label='train auc')
    plt.plot(param_values, test_auc, label='test auc')
    plt.grid()
    plt.xlabel(param_name)
    plt.ylabel('auc')
    plt.legend()
    plt.show()

    # plot the precision, recall, and accuracy for the test results
    plt.figure(figsize=(15, 8))
    plt.plot(param_values, test_precision, label='test precision')
    plt.plot(param_values, test_recall, label='test recall')
    plt.plot(param_values, test_accuracy, label='test accuracy')
```

```
    plt.plot(param_values, test_f1, label='test f1')
    plt.grid()
    plt.xlabel(param_name)
    plt.ylabel('auc')
    plt.legend()
    plt.show()

    return max_value, max_precision
```

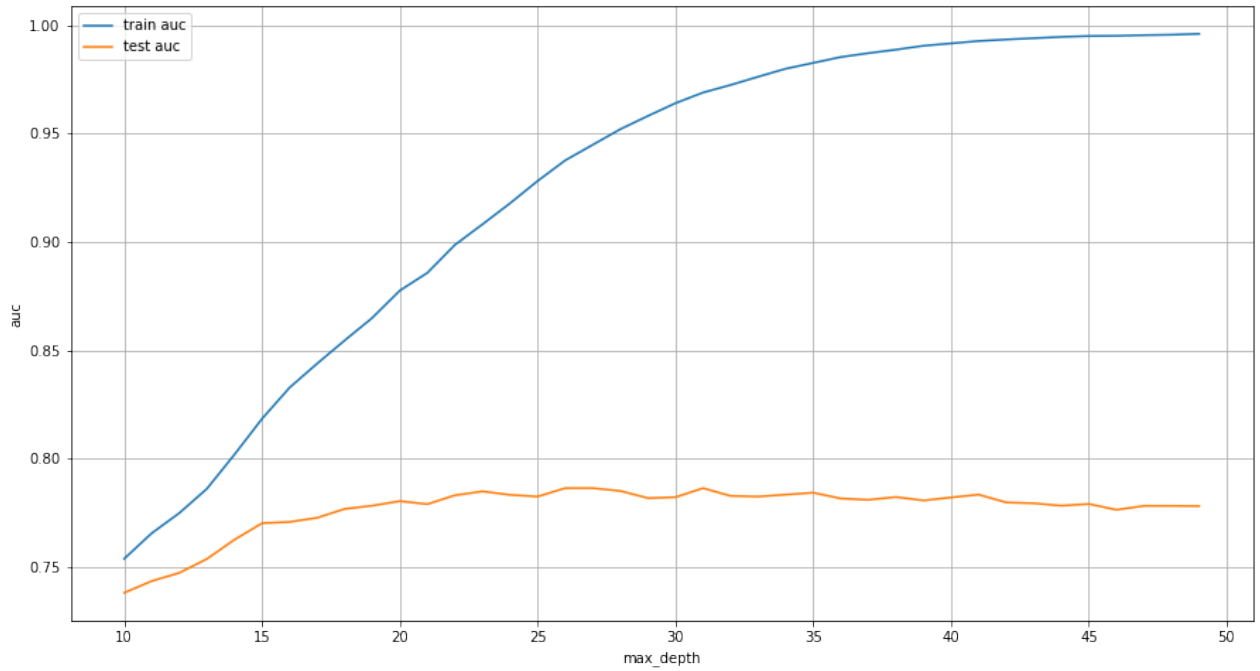
Investigate parameters

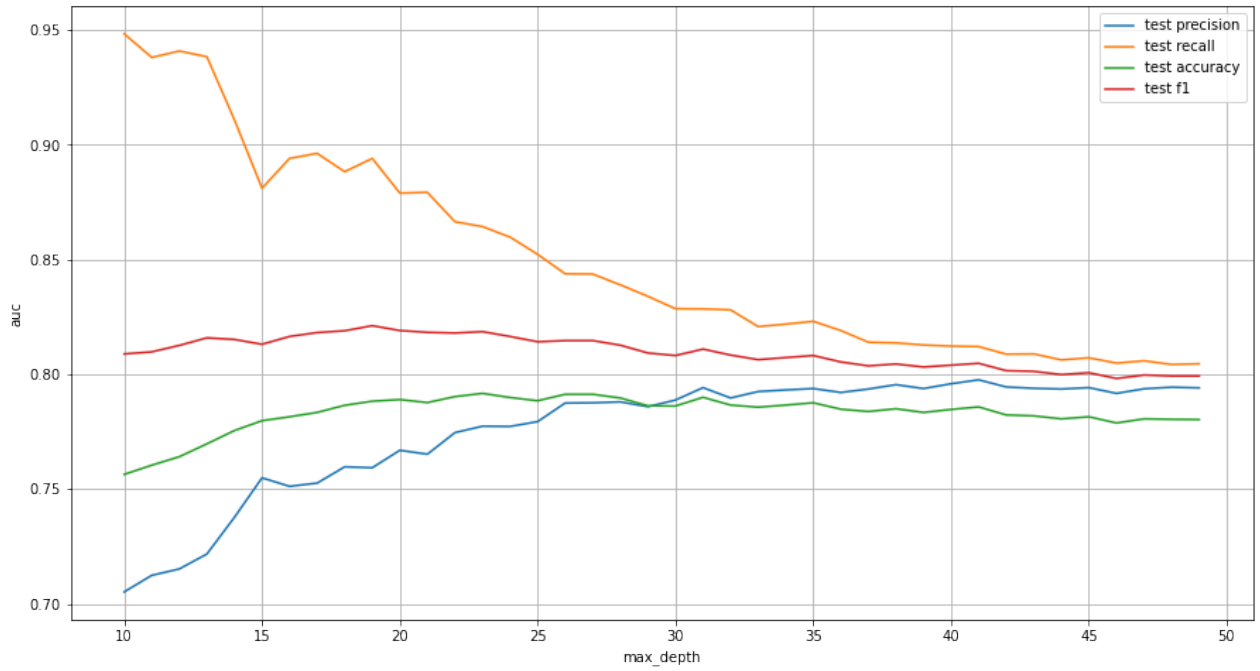
Investigate ranges for each parameter.

- select the value that returns the highest precision
- select a range of values including that value to be used in a grid search

Max Depth

In [31]:
```
# initialize a list of parameters and submit to the investigate parameters f
max_depths = list(range(10,50))
max_depth, max_precision = investigate_parameter(param_name='max_depth', par
```





Results

In [30]:
```
# select the max depth value with the highest precision
selected_max_depth = max_depth  # 19
# define a range of max depth values to in put in the grid search via inspec
selected_max_depth_range = list(range(30, 46))
selected_max_depth_range.extend([None])  # include the default value in the
```

In [31]:
```
# create, fit, and predict the model iteration for this parameter tuning
clf2 = DecisionTreeClassifier(criterion=criterion,
                              max_depth=selected_max_depth,
                              random_state=SEED)
clf2.fit(X_train_ohe, y_train)
y_pred_2 = clf2.predict(X_test_ohe)
```

In [32]:
```
# evaluate the new model and print key metrics
results2 = evaluate_model_performance(y_test.values, y_pred_2)

print(f'Model 2 Performance - Average')
print(f'  accuracy = {results2["accuracy"]}')
print(f'  precision = {results2["precision"]}')
print(f'  f1 = {results2["f1"]}')
print(f'  recall = {results2["recall"]}')
print(f'  auc = {results2["auc"]}')

print(f'Model 2 Performance - Functional')
print(f'  accuracy = {results2["subsets"]["status_group_functional"]["accura
print(f'  precision = {results2["subsets"]["status_group_functional"]["preci
print(f'  f1 = {results2["subsets"]["status_group_functional"]["f1"]}')
print(f'  recall = {results2["subsets"]["status_group_functional"]["recall"]
print(f'  auc = {results2["subsets"]["status_group_functional"]["auc"]}')
```

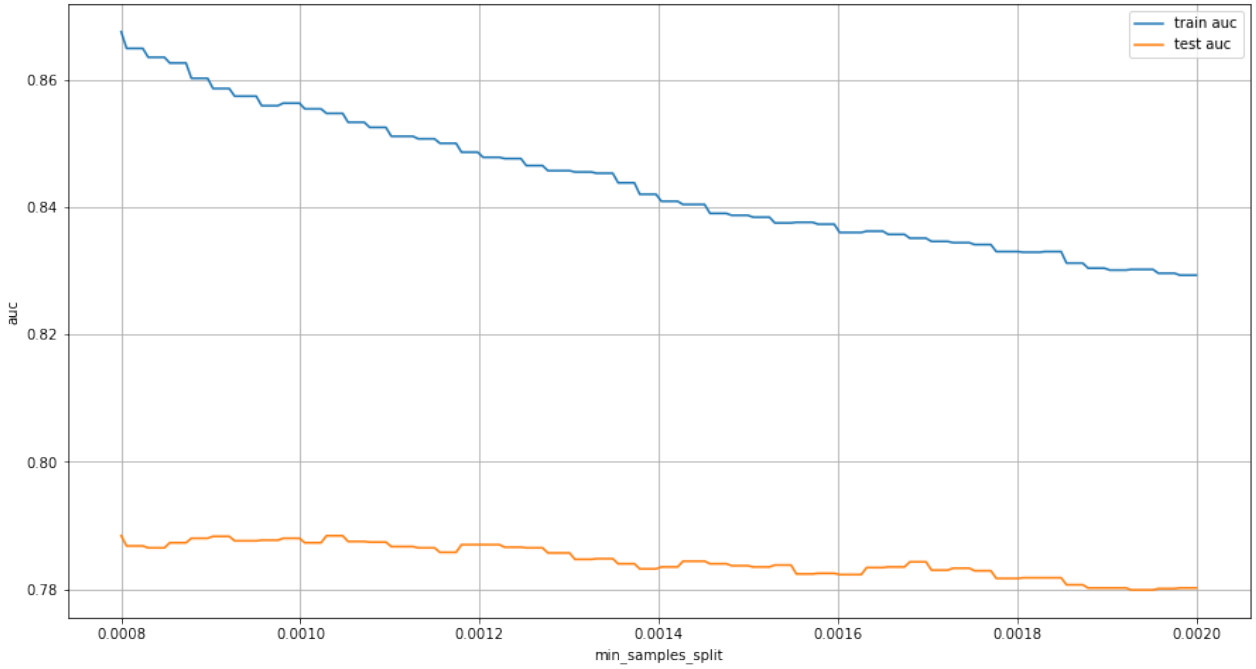
```
Model 2 Performance - Average
  accuracy = 0.8421
  precision = 0.6535
  f1 = 0.6456
  recall = 0.6392
  auc = 0.7481
Model 2 Performance - Functional
  accuracy = 0.79
  precision = 0.7942
  f1 = 0.811
  recall = 0.8285
  auc = 0.7863
```

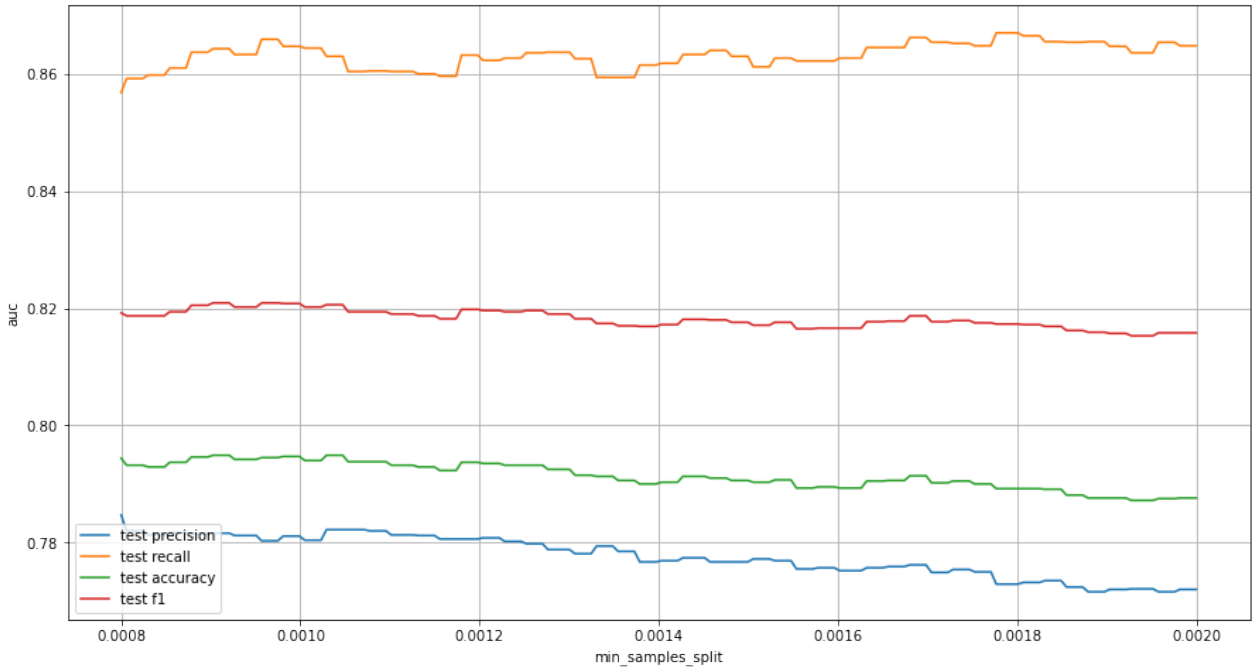
Interpret Results

- precision increased as the max depth increased, until it plateaued around 30.
- the average and function precision values both increased
- the highest precision found with a max depth at 35
- recall started high then fell as max depth increased until it plateaued around 30

Min Samples Split

In [35]:
```
# initialize a list of parameters and submit to the investigate parameters f
min_samples_splits = np.linspace(0.0008, 0.002, 200, endpoint=True)
min_samples_split, max_precision_split = investigate_parameter(param_name='m
```





Results

In [33]:
```
# select the min samples split value with the highest precision

selected_min_samples_split = min_samples_split  # 0.0008
print(selected_min_samples_split)

selected_min_samples_split_range = list(np.linspace(0.0008, 0.0014, 7, endpo
selected_min_samples_split_range.append(2)
```

```
0.0008
```

In [34]:
```
# create, fit, and predict the model iteration for this parameter tuning
clf3 = DecisionTreeClassifier(criterion=criterion,
                              max_depth=selected_max_depth,
                              min_samples_split=selected_min_samples_split,
                              random_state=SEED)
clf3.fit(X_train_ohe, y_train)
y_pred_3 = clf3.predict(X_test_ohe)
```

In [35]:
```
# evaluate the new model and print key metrics
results3 = evaluate_model_performance(y_test.values, y_pred_3)

print(f'Model 3 Performance - Average')
print(f'  accuracy = {results3["accuracy"]}')
print(f'  precision = {results3["precision"]}')
print(f'  precision = {results3["precision"]}')
print(f'  recall = {results3["recall"]}')
print(f'  auc = {results3["auc"]}')

print(f'Model 3 Performance - Functional')
print(f'  accuracy = {results3["subsets"]["status_group_functional"]["accura
print(f'  precision = {results3["subsets"]["status_group_functional"]["preci
print(f'  f1 = {results3["subsets"]["status_group_functional"]["f1"]}')
print(f'  recall = {results3["subsets"]["status_group_functional"]["recall"]
print(f'  auc = {results3["subsets"]["status_group_functional"]["auc"]}')
```

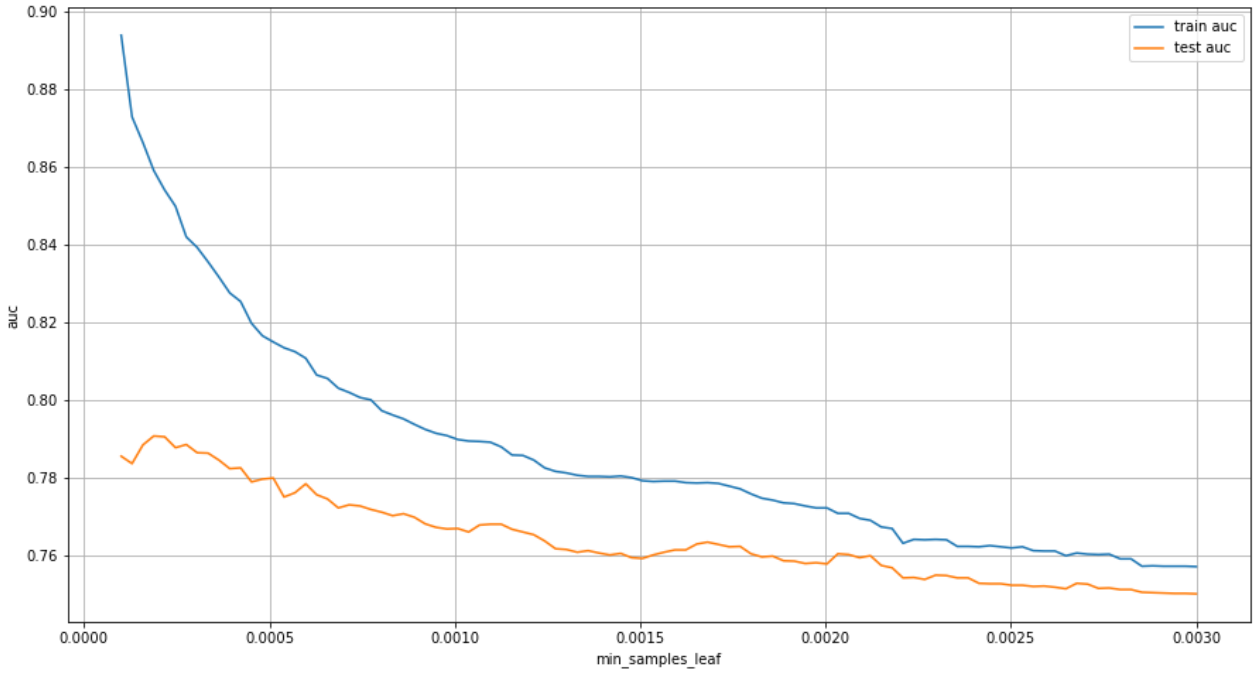
```
Model 3 Performance - Average
  accuracy = 0.8496
  precision = 0.6937
  precision = 0.6937
  recall = 0.6385
  auc = 0.7481
Model 3 Performance - Functional
  accuracy = 0.7955
  precision = 0.7835
  f1 = 0.8209
  recall = 0.8621
  auc = 0.7891
```

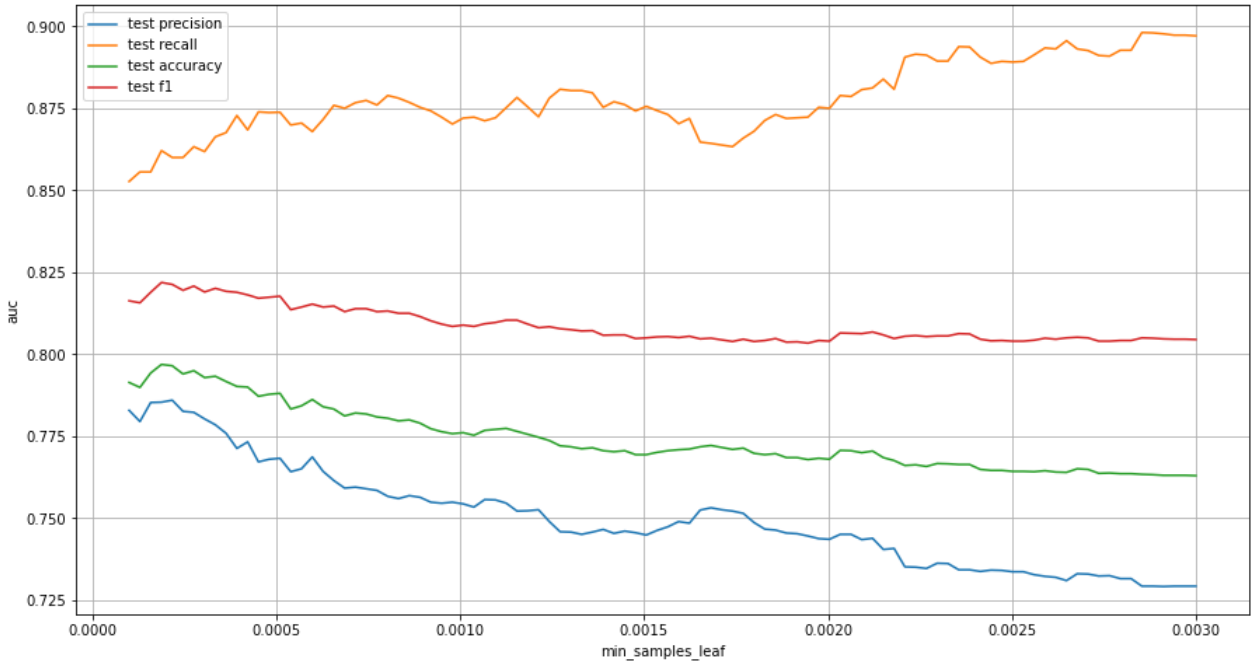
Interpret Results

- there is not a reliable increase throughout the range of min samples split given
  - the optimal value results in an increase in average precision, but not functional precision
- seeing as there is no disntinct increase in precision, it is likely that the default value will be the most accurate, however, a range of values will still be submitted to the grid search to investigate how they are affected by the other parameters

Min Samples Leaf

In [65]:
```
# initialize a list of parameters and submit to the investigate parameters f
min_samples_leafs = np.linspace(10e-5, 300e-5, 100, endpoint=True)
min_samples_leaf, max_precision_leaf = investigate_parameter(param_name='min
                                                             param_values=
                                                             max_depth=sel
                                                             min_samples_s
```





Results

In [66]:
```
# select the min samples leaf value with the highest precision
selected_min_samples_leaf = min_samples_leaf
selected_min_samples_leaf_range = list(np.linspace(13e-5, 26e-5, 14, endpoin
selected_min_samples_leaf_range.append(1)
```

In [67]:
```
# create, fit, and predict the model iteration for this parameter tuning
clf4 = DecisionTreeClassifier(criterion=criterion,
                              max_depth=selected_max_depth,
                              min_samples_split=selected_min_samples_split,
                              min_samples_leaf=selected_min_samples_leaf,
                              random_state=SEED)
clf4.fit(X_train_ohe, y_train)
y_pred_4 = clf4.predict(X_test_ohe)
```

In [68]:
```
# evaluate the new model and print key metrics
results4 = evaluate_model_performance(y_test.values, y_pred_4)

print(f'Model 4 Performance - Average')
print(f'  accuracy = {results4["accuracy"]}')
print(f'  precision = {results4["precision"]}')
print(f'  f1 = {results4["f1"]}')
print(f'  recall = {results4["recall"]}')
print(f'  auc = {results4["auc"]}')

print(f'Model 4 Performance - Functional')
print(f'  accuracy = {results4["subsets"]["status_group_functional"]["accura
print(f'  precision = {results4["subsets"]["status_group_functional"]["preci
print(f'  f1 = {results4["subsets"]["status_group_functional"]["f1"]}')
print(f'  recall = {results4["subsets"]["status_group_functional"]["recall"]
print(f'  auc = {results4["subsets"]["status_group_functional"]["auc"]}')
```

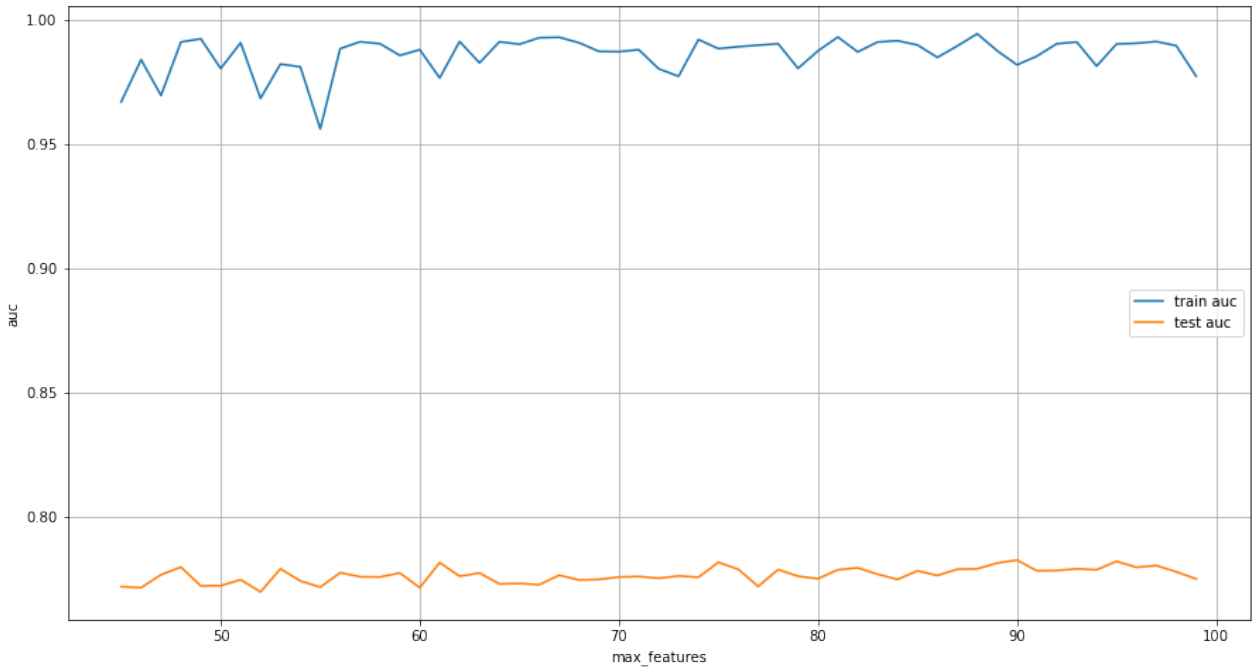
```
Model 4 Performance - Average
  accuracy = 0.851
  precision = 0.7065
  f1 = 0.6552
  recall = 0.6323
  auc = 0.7448
Model 4 Performance - Functional
  accuracy = 0.7967
  precision = 0.7818
  f1 = 0.8228
  recall = 0.8684
  auc = 0.7898
```

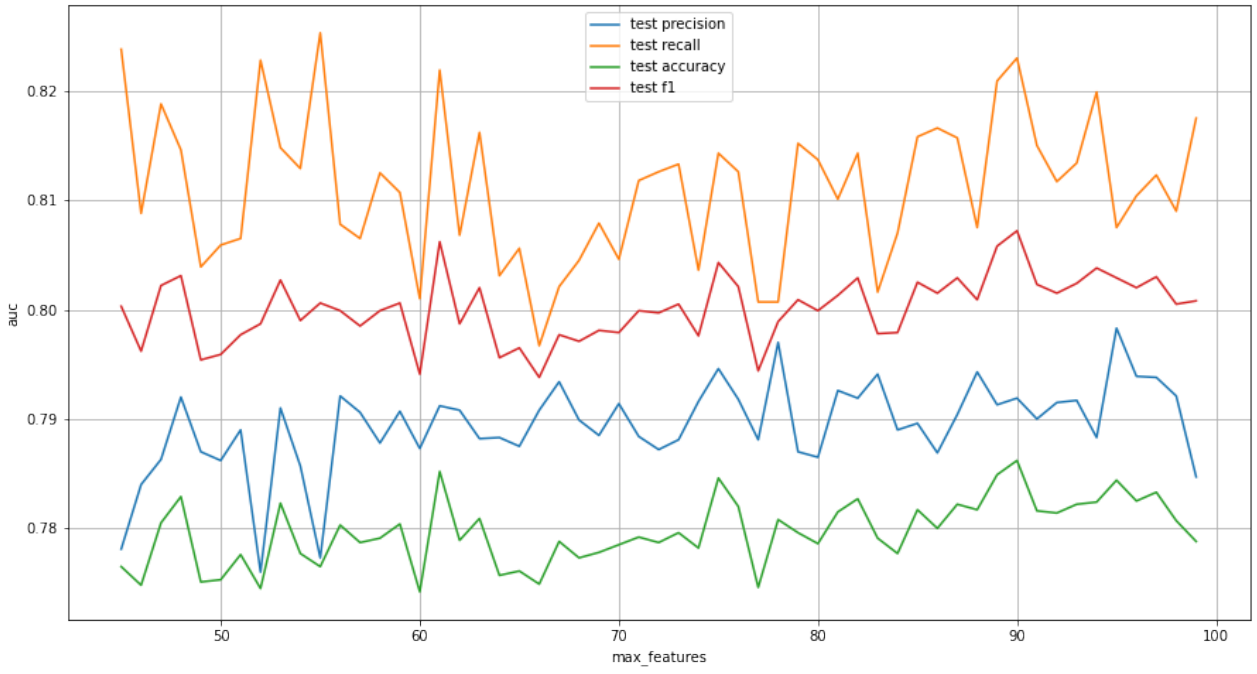
Interpretation

- there is a decrease in precision as min_samples_leaf increases, indicating that smaller values will return higher precision
- with the optimal value selected, average precision increased, while functional precision decreased

Max Features

In [53]:
```
# initialize a list of parameters and submit to the investigate parameters f
max_features = list(range(45, 100))
max_feature, max_precision_features = investigate_parameter(param_name='max_
                                                            max_depth=selec
```





Results

In [39]:
```
# select the max features value with the highest precision
selected_max_features = max_feature
selected_max_features_range = list(range(73, 100))
selected_max_features_range.extend([None])
```

In [40]:
```
# create, fit, and predict the model iteration for this parameter tuning
clf5 = DecisionTreeClassifier(criterion=criterion,
                              max_depth=selected_max_depth,
                              max_features=selected_max_features,
                              random_state=SEED)
clf5.fit(X_train_ohe, y_train)
y_pred_5 = clf5.predict(X_test_ohe)
```

In [41]:
```
# evaluate the new model and print key metrics
results5 = evaluate_model_performance(y_test.values, y_pred_5)

print(f'Model 5 Performance - Average')
print(f'  accuracy = {results5["accuracy"]}')
print(f'  precision = {results5["precision"]}')
print(f'  f1 = {results5["f1"]}')
print(f'  recall = {results5["recall"]}')
print(f'  auc = {results5["auc"]}')

print(f'Model 5 Performance - Functional')
print(f'  accuracy = {results5["subsets"]["status_group_functional"]["accura
print(f'  precision = {results5["subsets"]["status_group_functional"]["preci
print(f'  f1 = {results5["subsets"]["status_group_functional"]["f1"]}')
print(f'  recall = {results5["subsets"]["status_group_functional"]["recall"]
print(f'  auc = {results5["subsets"]["status_group_functional"]["auc"]}')
```

```
Model 5 Performance - Average
  accuracy = 0.8388
  precision = 0.6541
  f1 = 0.6481
  recall = 0.6432
  auc = 0.7485
Model 5 Performance - Functional
  accuracy = 0.783
  precision = 0.7867
  f1 = 0.8051
  recall = 0.8244
  auc = 0.779
```

Interpret Results

- there is a large difference between precision as max features increase without a clear trend.
- the optimal value selection seems more random rather than a distinct peak.

Grid Search

Using the ranges of values selected in the inital parameter tuning iterations, iterate through all possible combinations in a search for the combination of paramters that results in the highest functional precision.

In [47]:

```
hat iterates between all possible parameter combinations
oles_leafs, max_features):

keep track of progress


current min samples split to keep track of progress




urrent combination of paramters
on=criterion,
epth,
f=min_samples_leaf,
lt=min_samples_split,
_feature,
ED)




he)


erformance(y_test.values, test_pred)
_performance(y_train, train_pred)


hance)

performance['subsets']['status_group_functional']['precision']
pend(functional_binary_precision)

est_performance['subsets']['status_group_non functional']['precision']
l.append(non_functional_binary_precision)
```

```
e['precision']
ge_precision)

.it, min_samples_leaf, max_feature]


verage precision

.sion_grid.index(max_avg_precision)]

nctional precision
ary_precision_grid)
][functional_binary_precision_grid.index(max_functional_binary_precision)]

n-functional precision
.onal_binary_precision_grid)
_grid[non_functional_binary_precision_grid.index(max_non_functional_binary_pr

 2)}%: params = {max_avg_precision_params}')
_binary_precision*100, 2)}%: params = {max_functional_binary_precision_params
nctional_binary_precision*100, 2)}%: params = {max_non_functional_binary_prec



arams,
al_binary_precision,
nctional_binary_precision_params,
unctional_binary_precision,
ax_non_functional_binary_precision_params,




.ected_min_samples_split_range, selected_min_samples_leaf_range, selected_max
```

```
30
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
31
    0.0008
    0.0009
    0.001
```

```
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
32
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
33
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
34
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
35
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
36
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
37
    0.0008
    0.0009
    0.001
```

```
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
38
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
39
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
40
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
41
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
42
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
43
    0.0008
    0.0009
    0.001
```

```
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
44
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
45
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
None
    0.0008
    0.0009
    0.001
    0.0011
    0.0012000000000000001
    0.0013
    0.0014
    2
avg precision: 72.27%: params = [38, 0.0014, 0.00020999999999999998, 95]
functional precision: 80.07%: params = [42, 2, 1, 89]
non functional precision: 82.56%: params = [32, 2, 0.00015999999999999999,
91]
```

Interpret Results

- prior to hyperparameter tuning, the model functional precision was 80.25%.
- after hyperparameter tuning, the model functional precision is 80.69%
- there is an increase in functional precision of 0.44%

Final Model - Decision Tree

Run the final model chosen to maximize functional precision

In [48]:
```
# import results
with open("data/grid_results_precision.json", "r") as f:
    grid_results = json.load(f)
```

In [49]:
```
# select the final model parameters
chosen_final_model_params = grid_results['max_functional_binary_precision_pa

final_max_depth = chosen_final_model_params[0]
final_min_samples_split = chosen_final_model_params[1]
final_min_samples_leaf = chosen_final_model_params[2]
final_max_features = chosen_final_model_params[3]
```

In [50]:
```
# define and train a the decision tree, and predict the test labels
clf_final = DecisionTreeClassifier(criterion=criterion,
                                   max_depth=final_max_depth,
                                   min_samples_split=final_min_samples_split
                                   min_samples_leaf=final_min_samples_leaf,
                                   max_features=final_max_features,
                                   random_state=SEED)
clf_final.fit(X_train_ohe, y_train)
y_pred_final = clf_final.predict(X_test_ohe)
```

In [51]:
```
final_results = evaluate_model_performance(y_test.values, y_pred_final)

print(f'Final Model Performance - Average')
print(f'  accuracy = {final_results["accuracy"]}')
print(f'  precision = {final_results["precision"]}')
print(f'  f1 = {final_results["f1"]}')
print(f'  recall = {final_results["recall"]}')
print(f'  auc = {final_results["auc"]}')

print(f'Final Model Performance - Functional')
print(f'  accuracy = {final_results["subsets"]["status_group_functional"]["a
print(f'  precision = {final_results["subsets"]["status_group_functional"]["
print(f'  f1 = {final_results["subsets"]["status_group_functional"]["f1"]}')
print(f'  recall = {final_results["subsets"]["status_group_functional"]["rec
print(f'  auc = {final_results["subsets"]["status_group_functional"]["auc"]}
```

```
Final Model Performance - Average
  accuracy = 0.8395
  precision = 0.6531
  f1 = 0.6514
  recall = 0.6499
  auc = 0.7534
Final Model Performance - Functional
  accuracy = 0.7878
  precision = 0.8007
  f1 = 0.8061
  recall = 0.8117
  auc = 0.7855
```

Random Forest Using Final Decision Tree Parameters

Input the final decision tree parameters into a random forest.

In [52]:
```
# define random forest parameters
n_estimators=100
n_jobs=None

# define the random forest and train it
forest = RandomForestClassifier(n_estimators=n_estimators, criterion=criteri
                                max_depth=final_max_depth, max_features=final
                                min_samples_split=final_min_samples_split, mi
                                n_jobs=n_jobs, random_state=SEED)
forest.fit(X_train_ohe, y_train)

# use the forest to predict the test labels
y_forest = forest.predict(X_test_ohe)
```

In [53]:
```
forest_results = evaluate_model_performance(y_test.values, y_forest)

print(f'Final Model Performance - Average')
print(f'  accuracy = {forest_results["accuracy"]}')
print(f'  precision = {forest_results["precision"]}')
print(f'  f1 = {forest_results["f1"]}')
print(f'  recall = {forest_results["recall"]}')
print(f'  auc = {forest_results["auc"]}')

print(f'Final Model Performance - Functional')
print(f'  accuracy = {forest_results["subsets"]["status_group_functional"]["
print(f'  precision = {forest_results["subsets"]["status_group_functional"][
print(f'  f1 = {forest_results["subsets"]["status_group_functional"]["f1"]}'
print(f'  recall = {forest_results["subsets"]["status_group_functional"]["re
print(f'  auc = {forest_results["subsets"]["status_group_functional"]["auc"]
```

```
Final Model Performance - Average
  accuracy = 0.8686
  precision = 0.7221
  f1 = 0.6893
  recall = 0.6698
  auc = 0.7727
Final Model Performance - Functional
  accuracy = 0.8204
  precision = 0.805
  f1 = 0.8426
  recall = 0.8839
  auc = 0.8143
```

Interpretaion

- after hyperparameter tuning, the model functional precision is 80.69%
- using the same optimized hyperparameters, the model functional precision is increased to 80.86%
- there is an increase in functional precision of 0.61% from the orignal base model

Model Improvements

In [69]:
```
# functional
precision_increase = forest_results["subsets"]["status_group_functional"]["pr
f1_increase = forest_results["subsets"]["status_group_functional"]["f1"] - re
recall_increase = forest_results["subsets"]["status_group_functional"]["recal
accuracy_increase = forest_results["subsets"]["status_group_functional"]["acc

print(f'increase in precision from base model to final model: {round(precisio
print(f'increase in f1 score from base model to final model: {round(f1_increa
print(f'increase in recall from base model to final model: {round(recall_incr
print(f'increase in accuracy from base model to final model: {round(accuracy_
```

```
increase in precision from base model to final model: 1.03%
increase in f1 score from base model to final model: 4.28%
increase in recall from base model to final model: 7.89%
increase in accuracy from base model to final model: 3.95%
```

The optimized model generated an increase in all major metrics, with the largest increae in recall.

In [55]:
```
# final model classification report
final_report = classification_report(y_test, y_forest)
print(final_report)
```

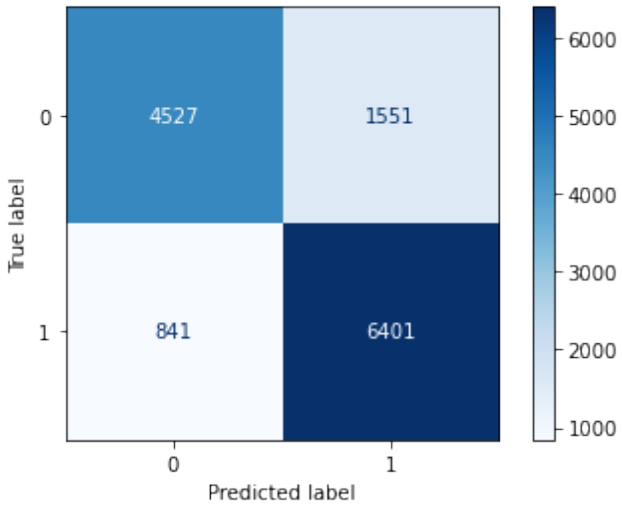
```
                         precision    recall  f1-score   support

             functional       0.80      0.88      0.84      7242
functional needs repair       0.52      0.35      0.42       984
         non functional       0.84      0.78      0.81      5094

               accuracy                           0.80     13320
              macro avg       0.72      0.67      0.69     13320
           weighted avg       0.80      0.80      0.80     13320
```

Confusion Matrix

Generate a confusion matrix to illustrate the numbers of true postives, true negaties, false positives, and fasle negatives

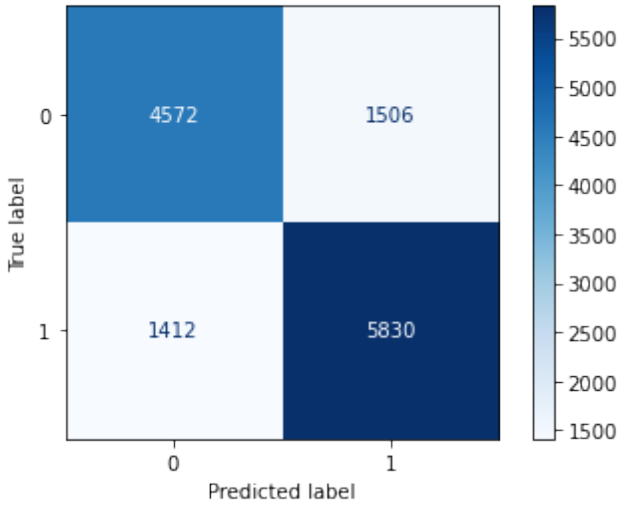
Final Model

In [56]:
```
y_test_ohe_df = create_y_ohe(y_test, columns=['status_group'])
y_pred_ohe_df = create_y_ohe(y_forest, columns=['status_group'])

y_test_functional = y_test_ohe_df['status_group_functional'].values
y_pred_functional = y_pred_ohe_df['status_group_functional'].values

cfm = confusion_matrix(y_test_functional, y_pred_functional)
disp = ConfusionMatrixDisplay(confusion_matrix=cfm)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



Base Model

In [57]:
```
y_test_ohe_df = create_y_ohe(y_test, columns=['status_group'])
y_pred_ohe_df = create_y_ohe(y_pred_base, columns=['status_group'])

y_test_functional = y_test_ohe_df['status_group_functional'].values
y_pred_functional = y_pred_ohe_df['status_group_functional'].values

cfm = confusion_matrix(y_test_functional, y_pred_functional)
disp = ConfusionMatrixDisplay(confusion_matrix=cfm)
disp.plot(cmap=plt.cm.Blues)
plt.show()
```



Interpretation

Our goal was to minimize the number of false positives by using precision as out primary metric, while the number of false positives increased, the number of true positives increased at a greater rate, which in turn raised the precision by 1%.

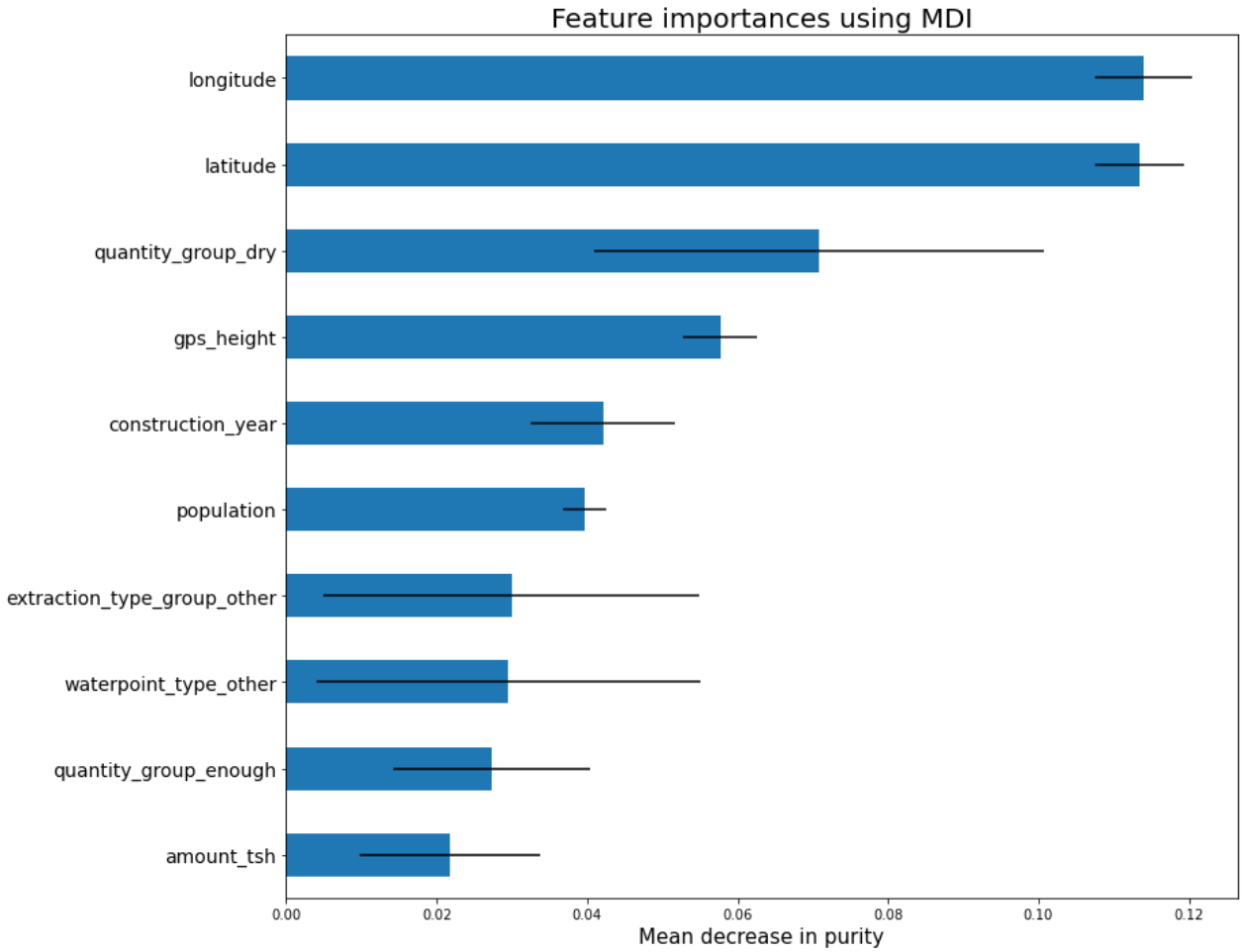
Random Forest Feature Importance

Investigate the features to determine the which characteristics of the pumps have the largest effect on their functionality.

In [58]:
```
# define the importances and their standard deivations
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axi
```

In [59]:
```
# define feature names, importances, and their standard deviations
feature_names = np.array(X_train_ohe.columns)
importances = forest.feature_importances_
std = np.std([tree.feature_importances_ for tree in forest.estimators_], axi

# create a data frame for all the importances
data={
    'feature': feature_names,
    'importance': importances,
    'std': std
}
forest_importances = pd.DataFrame(data)

# filter the importances to only include the top features, then sort
forest_importances_filtered = forest_importances[forest_importances['importa
forest_importances_filtered = forest_importances_filtered.set_index('feature
forest_importances_filtered = forest_importances_filtered.sort_values(by='im
```

In [60]:
```
# plot feature importances
fig, ax = plt.subplots(figsize=(13,10))
forest_importances_filtered['importance'].plot.barh(xerr=forest_importances_f
ax.set_title("Feature importances using MDI", fontsize=20)
ax.set_xlabel("Mean decrease in purity", fontsize=15)
ax.set_ylabel('', )
plt.yticks(fontsize=14)
fig.tight_layout()
```



Interpretaion

The top features that are most important in determining if a water pump is functional:

1. location (latitude/longitude/elevation)
2. water quantity
3. construction year

As we develop a plan to make maintenance visits to water pumps, we should prioritize the pumps by location first, identifying regions with higher rates of non-functioning pumps. After that, focus on older pumps with "dry" water quantities.