ModelResults

March 7, 2025

[3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

When we look at the data, we realize that the target columns are actually stored in a separate file. To fix this, we can just merge the two since they share ids into one dataframe. We will combine functional with functional needs to repairs so this will be a binary problem as opposed to a ternary one.

[4]:
```
df_labels = pd.read_csv('../data/training_set_labels.csv', header=0)
df_labels.replace({'functional needs repair': 'functional'}, inplace=True)
df_labels['status_group'].value_counts(normalize=True)
```

[4]:
```
functional        0.615758
non functional    0.384242
Name: status_group, dtype: float64
```

[5]:
```
df_values = pd.read_csv('../data/training_set_values.csv', header=0)
df_values
```

[5]:
```
          id  amount_tsh date_recorded          funder  gps_height  \
0      69572      6000.0    2011-03-14           Roman        1390
1       8776         0.0    2013-03-06         Grumeti        1399
2      34310        25.0    2013-02-25    Lottery Club         686
3      67743         0.0    2013-01-28          Unicef         263
4      19728         0.0    2011-07-13      Action In A           0
…        …           …           …               …          …
59395  60739        10.0    2013-05-03  Germany Republi        1210
59396  27263      4700.0    2011-05-07     Cefa-njombe        1212
59397  37057         0.0    2011-04-11             NaN           0
59398  31282         0.0    2011-03-08           Malec           0
59399  26348         0.0    2011-03-23      World Bank         191

          installer  longitude   latitude              wpt_name  num_private  \
0             Roman  34.938093  -9.856322                  none            0
1           GRUMETI  34.698766  -2.147466              Zahanati            0
```

```
2      World vision  37.460664  -3.821329          Kwa Mahundi            0
3            UNICEF  38.486161 -11.155298  Zahanati Ya Nanyumbu            0
4           Artisan  31.130847  -1.825359              Shuleni            0
…                 …          …          …                    …          …
59395           CES  37.169807  -3.253847   Area Three Namba 27            0
59396          Cefa  35.249991  -9.070629    Kwa Yahona Kuvala            0
59397           NaN  34.017087  -8.750434              Mashine            0
59398          Musa  35.861315  -6.378573               Mshoro            0
59399         World  38.104048  -6.747464       Kwa Mzee Lugawa            0

       … payment_type water_quality quality_group      quantity  \
0      …     annually          soft          good        enough
1      …    never pay          soft          good  insufficient
2      …   per bucket          soft          good        enough
3      …    never pay          soft          good           dry
4      …    never pay          soft          good      seasonal
…    …            …             …             …             …
59395  …   per bucket          soft          good        enough
59396  …     annually          soft          good        enough
59397  …      monthly      fluoride      fluoride        enough
59398  …    never pay          soft          good  insufficient
59399  …   on failure         salty         salty        enough

       quantity_group                source           source_type  \
0              enough                spring                spring
1        insufficient  rainwater harvesting  rainwater harvesting
2              enough                   dam                   dam
3                 dry           machine dbh              borehole
4            seasonal  rainwater harvesting  rainwater harvesting
…                   …                     …                     …
59395          enough                spring                spring
59396          enough                 river            river/lake
59397          enough           machine dbh              borehole
59398    insufficient          shallow well          shallow well
59399          enough          shallow well          shallow well

       source_class              waterpoint_type waterpoint_type_group
0       groundwater           communal standpipe    communal standpipe
1           surface           communal standpipe    communal standpipe
2           surface  communal standpipe multiple    communal standpipe
3       groundwater  communal standpipe multiple    communal standpipe
4           surface           communal standpipe    communal standpipe
…                 …                            …                     …
59395   groundwater           communal standpipe    communal standpipe
59396       surface           communal standpipe    communal standpipe
59397   groundwater                    hand pump             hand pump
59398   groundwater                    hand pump             hand pump
```

```
59399   groundwater                         hand pump            hand pump

[59400 rows x 40 columns]
```

Since the data is stored in two separate csv, we will combine them into one. Then we'll combine the target columns with the dataframes.

[6]:
```
df_val_test = pd.read_csv('../data/test_set.csv', header=0)
df_val_test
df_values = pd.concat([df_values, df_val_test], ignore_index=True)
df_values = df_values.merge(df_labels, left_on='id', right_on='id')
```

[7]:
```
df_values
```

[7]:
```
          id  amount_tsh date_recorded           funder  gps_height  \
0      69572      6000.0    2011-03-14            Roman        1390
1       8776         0.0    2013-03-06          Grumeti        1399
2      34310        25.0    2013-02-25     Lottery Club         686
3      67743         0.0    2013-01-28           Unicef         263
4      19728         0.0    2011-07-13      Action In A           0
…          …           …             …                …           …
59395  60739        10.0    2013-05-03  Germany Republi        1210
59396  27263      4700.0    2011-05-07      Cefa-njombe        1212
59397  37057         0.0    2011-04-11              NaN           0
59398  31282         0.0    2011-03-08            Malec           0
59399  26348         0.0    2011-03-23       World Bank         191

          installer  longitude   latitude              wpt_name  num_private  \
0             Roman  34.938093  -9.856322                  none            0
1           GRUMETI  34.698766  -2.147466              Zahanati            0
2      World vision  37.460664  -3.821329           Kwa Mahundi            0
3            UNICEF  38.486161 -11.155298  Zahanati Ya Nanyumbu            0
4           Artisan  31.130847  -1.825359               Shuleni            0
…               …          …          …                   …          …
59395           CES  37.169807  -3.253847   Area Three Namba 27            0
59396          Cefa  35.249991  -9.070629     Kwa Yahona Kuvala            0
59397           NaN  34.017087  -8.750434               Mashine            0
59398          Musa  35.861315  -6.378573                Mshoro            0
59399         World  38.104048  -6.747464       Kwa Mzee Lugawa            0

       … water_quality quality_group      quantity  quantity_group  \
0      …          soft          good        enough          enough
1      …          soft          good  insufficient    insufficient
2      …          soft          good        enough          enough
3      …          soft          good           dry             dry
4      …          soft          good      seasonal        seasonal
…     …             …             …             …               …
59395  …          soft          good        enough          enough
```

```
59396  …          soft          good         enough           enough
59397  …      fluoride      fluoride         enough           enough
59398  …          soft          good   insufficient     insufficient
59399  …         salty         salty         enough           enough

                     source           source_type source_class  \
0                    spring                spring  groundwater
1       rainwater harvesting  rainwater harvesting      surface
2                       dam                   dam      surface
3               machine dbh              borehole  groundwater
4       rainwater harvesting  rainwater harvesting      surface
…                       …                     …           …
59395                spring                spring  groundwater
59396                 river            river/lake      surface
59397           machine dbh              borehole  groundwater
59398          shallow well          shallow well  groundwater
59399          shallow well          shallow well  groundwater

                  waterpoint_type waterpoint_type_group    status_group
0              communal standpipe    communal standpipe      functional
1              communal standpipe    communal standpipe      functional
2     communal standpipe multiple    communal standpipe      functional
3     communal standpipe multiple    communal standpipe  non functional
4              communal standpipe    communal standpipe      functional
…                             …                     …             …
59395          communal standpipe    communal standpipe      functional
59396          communal standpipe    communal standpipe      functional
59397                   hand pump             hand pump      functional
59398                   hand pump             hand pump      functional
59399                   hand pump             hand pump      functional

[59400 rows x 41 columns]
```

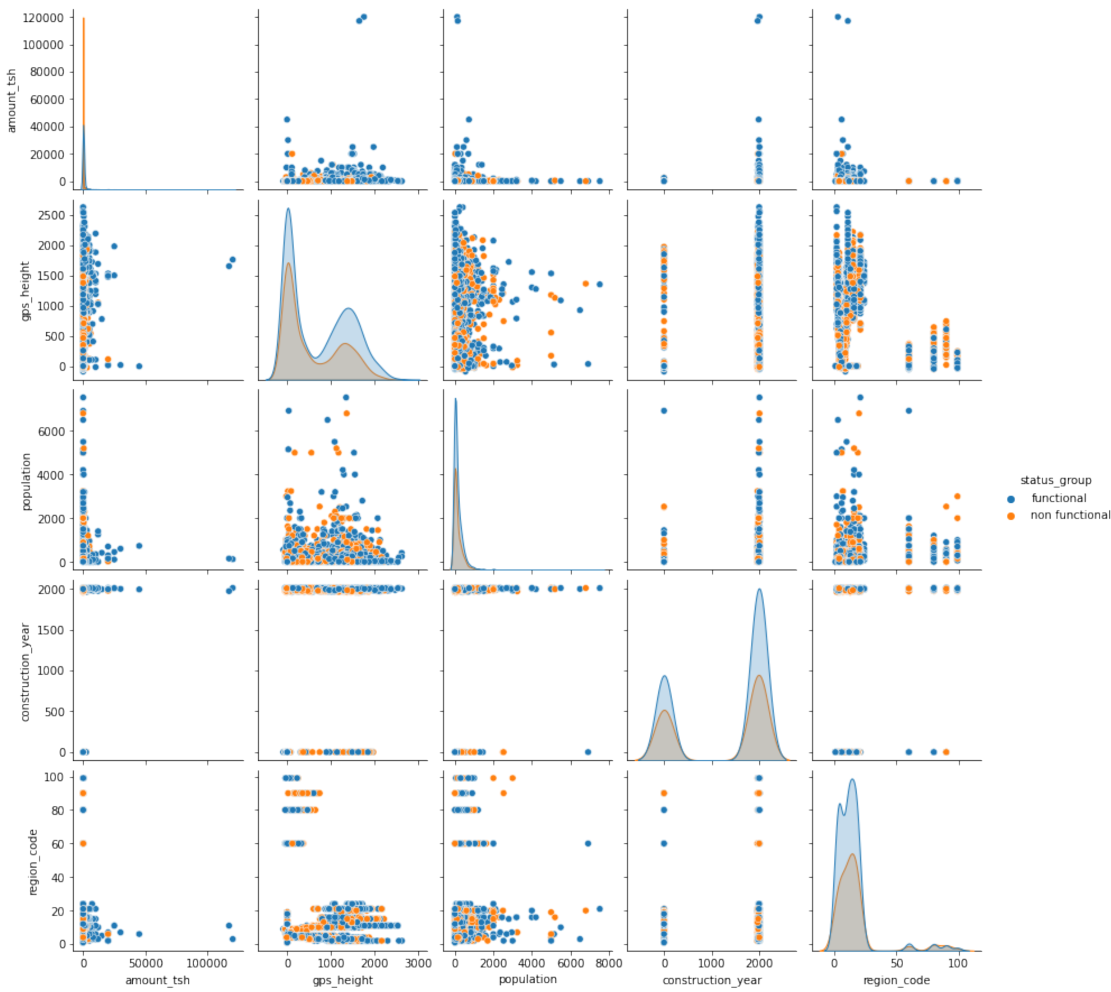
There are a lot of columns with non numerical entries. This means we might have to one hot encode them, however with how many columns and distinct entries there are, it might be too many factors. We will likely need to drop some of these columns. First lets plot of graphs with only the numeric columns

We also should take a sample of the data for this graphing since there is a lot of data and it would take a long time to graph each time.

[8]:
```
df_numeric = df_values.select_dtypes(include=np.number).merge(df_labels,␣
 ↪left_on='id', right_on='id')
#Take a sample of a quarter of the data, random state for reproducability
df_num_sample = df_numeric.sample(frac=0.10, random_state= 5)
df_num_sample.drop(columns=['id'], inplace=True)
```

[9]:
```
df_small = df_num_sample[['amount_tsh', 'gps_height', 'population',␣
↪'construction_year', 'region_code', 'status_group']]
sns.pairplot(hue='status_group', data=df_small)
#sns.pairplot(hue = 'status_group', data= df_num_sample)
```

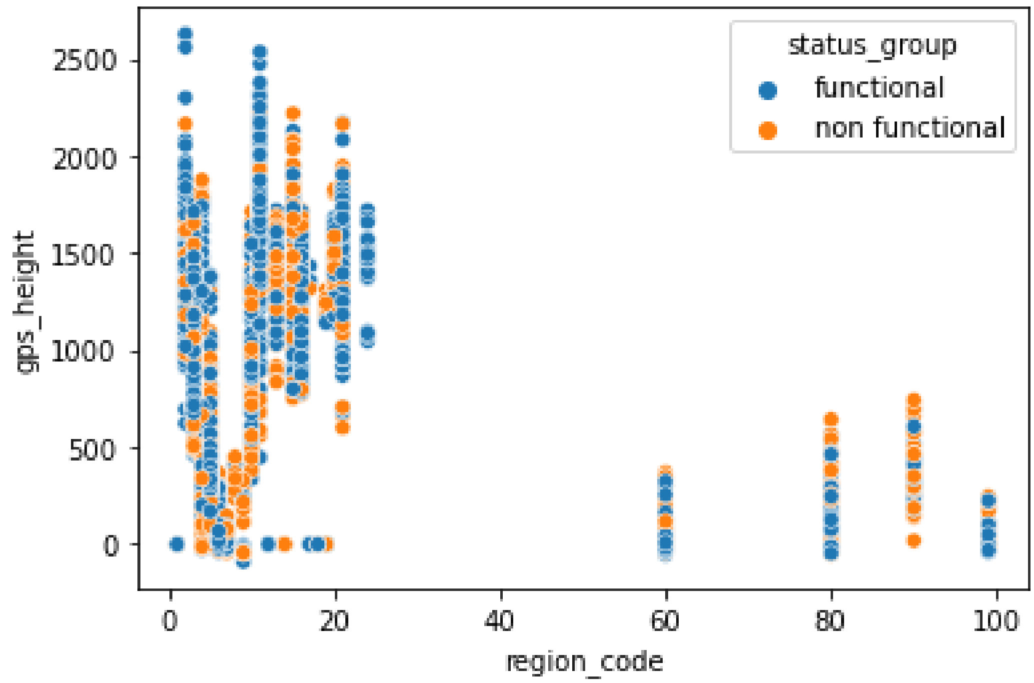
[9]: <seaborn.axisgrid.PairGrid at 0x29cb4816460>



Here we'll look more closely at some specific examples

[10]:
```
sns.scatterplot(data= df_num_sample, x = 'region_code', y='gps_height', hue =␣
↪'status_group')
```

[10]: <AxesSubplot:xlabel='region_code', ylabel='gps_height'>



[11]:
```
df_values.columns
```

[11]:
```
Index(['id', 'amount_tsh', 'date_recorded', 'funder', 'gps_height',
       'installer', 'longitude', 'latitude', 'wpt_name', 'num_private',
       'basin', 'subvillage', 'region', 'region_code', 'district_code', 'lga',
       'ward', 'population', 'public_meeting', 'recorded_by',
       'scheme_management', 'scheme_name', 'permit', 'construction_year',
       'extraction_type', 'extraction_type_group', 'extraction_type_class',
       'management', 'management_group', 'payment', 'payment_type',
       'water_quality', 'quality_group', 'quantity', 'quantity_group',
       'source', 'source_type', 'source_class', 'waterpoint_type',
       'waterpoint_type_group', 'status_group'],
      dtype='object')
```

[12]:
```
good_qual_df = df_values[(df_values['quality_group']=='good') &␣
 ↪(df_values['amount_tsh']<100)]
good_qual_df
```

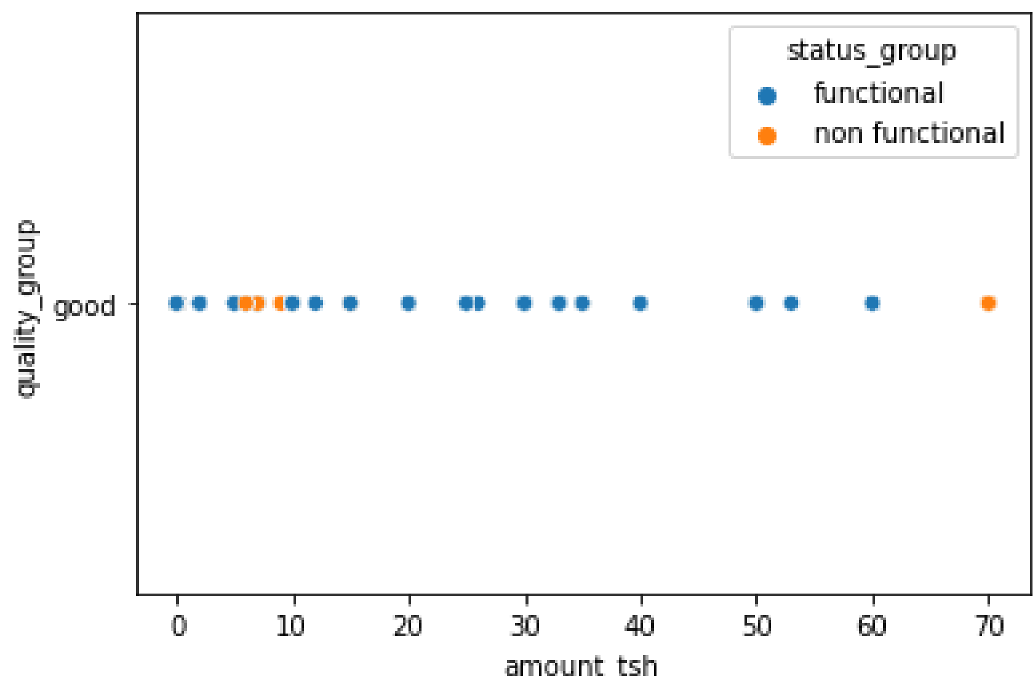
[12]:
```
          id  amount_tsh date_recorded                  funder  gps_height  \
1       8776         0.0    2013-03-06                 Grumeti        1399
2      34310        25.0    2013-02-25            Lottery Club         686
3      67743         0.0    2013-01-28                  Unicef         263
4      19728         0.0    2011-07-13            Action In A           0
6      19816         0.0    2012-10-01                    Dwsp           0
```

```
…      …         …          …                       …        …
59391  44885       0.0    2013-08-03  Government Of Tanzania       540
59392  40607       0.0    2011-04-15  Government Of Tanzania         0
59393  48348       0.0    2012-10-27                 Private         0
59395  60739      10.0    2013-05-03         Germany Republi      1210
59398  31282       0.0    2011-03-08                   Malec         0

          installer  longitude   latitude              wpt_name  num_private  \
1           GRUMETI  34.698766  -2.147466              Zahanati            0
2      World vision  37.460664  -3.821329           Kwa Mahundi            0
3            UNICEF  38.486161 -11.155298  Zahanati Ya Nanyumbu            0
4           Artisan  31.130847  -1.825359               Shuleni            0
6              DWSP  33.362410  -3.766365            Kwa Ngomho            0
…               …         …          …                    …           …
59391    Government  38.044070  -4.272218                   Kwa            0
59392    Government  33.009440  -8.520888        Benard Charles            0
59393       Private  33.866852  -4.287410             Kwa Peter            0
59395           CES  37.169807  -3.253847   Area Three Namba 27            0
59398          Musa  35.861315  -6.378573                Mshoro            0

       … water_quality quality_group      quantity  quantity_group  \
1      …          soft          good  insufficient    insufficient
2      …          soft          good        enough          enough
3      …          soft          good           dry             dry
4      …          soft          good      seasonal        seasonal
6      …          soft          good        enough          enough
…    …           …             …             …              …
59391  …          soft          good        enough          enough
59392  …          soft          good        enough          enough
59393  …          soft          good  insufficient    insufficient
59395  …          soft          good        enough          enough
59398  …          soft          good  insufficient    insufficient

                     source           source_type source_class  \
1      rainwater harvesting  rainwater harvesting      surface
2                       dam                   dam      surface
3               machine dbh              borehole  groundwater
4      rainwater harvesting  rainwater harvesting      surface
6               machine dbh              borehole  groundwater
…                       …                    …          …
59391                 river            river/lake      surface
59392                spring                spring  groundwater
59393                   dam                   dam      surface
59395                spring                spring  groundwater
59398          shallow well          shallow well  groundwater

                   waterpoint_type waterpoint_type_group    status_group
```

```
1                 communal standpipe    communal standpipe      functional
2        communal standpipe multiple    communal standpipe      functional
3        communal standpipe multiple    communal standpipe  non functional
4                 communal standpipe    communal standpipe      functional
6                          hand pump             hand pump  non functional
…                                  …                     …               …
59391             communal standpipe    communal standpipe  non functional
59392             communal standpipe    communal standpipe  non functional
59393                          other                 other      functional
59395             communal standpipe    communal standpipe      functional
59398                          hand pump             hand pump      functional

[40493 rows x 41 columns]
```

[13]:
```
sns.scatterplot(data=good_qual_df, x = 'amount_tsh', y='quality_group', hue =␣
↪'status_group')
```

[13]:
```
<AxesSubplot:xlabel='amount_tsh', ylabel='quality_group'>
```



We can see that in most of the cases, the functional and non-functional had no clear separations. So the numerical columns don't seem to be good at determining whether a well is functional or not.

1 Baseline Model

With just the numerical categories, there does not seem to be any columns that do a particularly good job at correlating to a well being either function or non functional.
What we'll do for now is just to create a baseline model without any preprocessing to see what the accuracy score will be.

[14]:
```
#Drop the id column now that we don't need it
df_values.drop(columns=['id'], inplace=True)
df_values['status_group'].replace({'functional': 1, 'non functional': 0},␣
 ↪inplace=True)
```

[15]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split, cross_validate
from sklearn.metrics import accuracy_score, recall_score

X = df_numeric.drop(columns=['status_group'])
y = df_values['status_group']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state=42)

model = LogisticRegression(max_iter=10000, solver='liblinear')
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
print(accuracy_score(y_test, y_pred))
print(f"Specificity(Recall) Score: {recall_score(y_test, y_pred, pos_label=0)}")
print(f"Recall Score: {recall_score(y_test, y_pred)}")
```

```
0.6174410774410775
Specificity(Recall) Score: 0.10936949630151462
Recall Score: 0.9319668556476232
```

Here we are creating lists to store the accuracy, specificity and recall score so we can graph a line chart at the end that measures how they change

[16]:
```
list_acc = []
list_spec = []
list_rec = []
list_acc.append(accuracy_score(y_test, y_pred))
list_spec.append(recall_score(y_test, y_pred, pos_label=0))
list_rec.append(recall_score(y_test, y_pred))
```

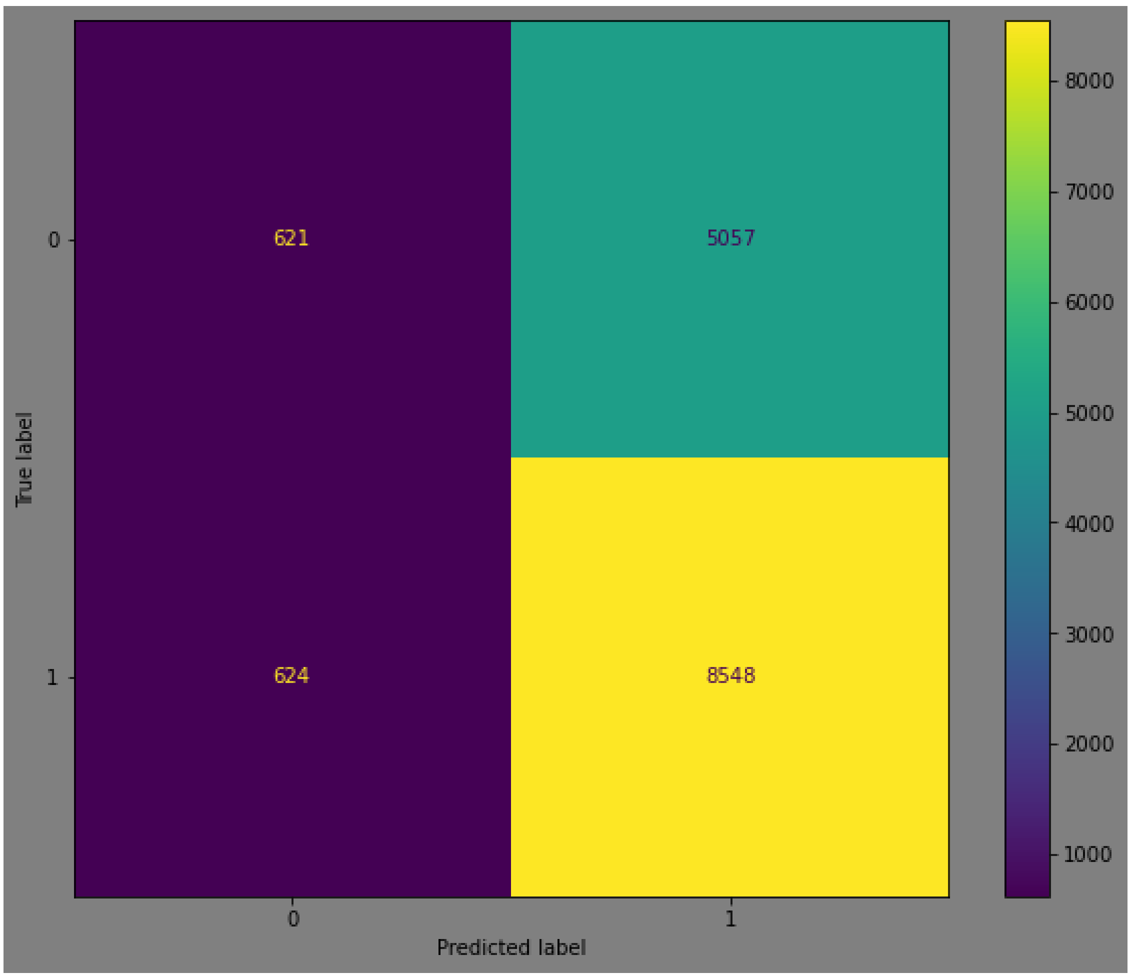
[17]:
```
df_values['status_group'].value_counts()
```

[17]:
```
1    36576
0    22824
Name: status_group, dtype: int64
```

Without any preproccessing done, the model does a pretty bad job at predicting the results, with a large amount of false positives and a not so small false negative as well.

[18]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay,␣
    ↪recall_score, precision_score
fig, ax = plt.subplots(figsize=(10,8))
fig.set_facecolor('grey')
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot(ax=ax)
```

[18]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29cb7b4abe0>



Next, we'll try to look at the coeeficients to see which columns have a high correlation.

[19]:
```
coefficients = model.coef_[0]
odds_ratios = np.exp(coefficients)
```

```
feature_importance = pd.DataFrame({'Feature': X.columns, 'Coefficients':␣
↪coefficients, 'Odds Ratios': odds_ratios})
feature_importance
```

[19]:

	Feature	Coefficients	Odds Ratios
0	id	0.000002	1.000002
1	amount_tsh	0.000267	1.000267
2	gps_height	0.000386	1.000386
3	longitude	0.010821	1.010879
4	latitude	-0.000632	0.999368
5	num_private	0.000817	1.000818
6	region_code	-0.006937	0.993087
7	district_code	-0.000543	0.999457
8	population	0.000102	1.000102
9	construction_year	-0.000157	0.999843

Here we can see that most of the numerical data has a negative coefficient, aside from longitude, num private, region code, and construction year, most of which have a near 0 coefficient.
We'll also look at the categorical columns and see if we can drop any that are not particularly useful.

[20]:
```
df_values['extraction_type_group'].value_counts()
```

[20]:
```
gravity            26780
nira/tanira         8154
other               6430
submersible         6179
swn 80              3670
mono                2865
india mark ii       2400
afridev             1770
rope pump            451
other handpump       364
other motorpump      122
wind-powered         117
india mark iii        98
Name: extraction_type_group, dtype: int64
```

[21]:
```
#A fair amount of these columns are essentially duplicates, have the same data␣
↪as other columns or similar
#PUT BACK water_quality, management, source, district_code, extraction_type
non_important_columns = ['wpt_name', 'region_code', 'subvillage',␣
↪'extraction_type','payment', 'quantity', 'waterpoint_type', 'source_type',␣
↪'scheme_name', 'lga', 'ward']
prep_df_values = df_values.drop(columns=non_important_columns)
prep_df_values
```

```
[21]:        amount_tsh date_recorded           funder  gps_height     installer  \
0          6000.0    2011-03-14            Roman        1390         Roman
1             0.0    2013-03-06          Grumeti        1399       GRUMETI
2            25.0    2013-02-25     Lottery Club         686  World vision
3             0.0    2013-01-28           Unicef         263        UNICEF
4             0.0    2011-07-13      Action In A           0       Artisan
…             …          …              …        …          …
59395        10.0    2013-05-03  Germany Republi        1210           CES
59396      4700.0    2011-05-07      Cefa-njombe        1212          Cefa
59397         0.0    2011-04-11              NaN           0           NaN
59398         0.0    2011-03-08            Malec           0          Musa
59399         0.0    2011-03-23       World Bank         191         World

       longitude   latitude  num_private                    basin  \
0      34.938093  -9.856322            0               Lake Nyasa
1      34.698766  -2.147466            0           Lake Victoria
2      37.460664  -3.821329            0                  Pangani
3      38.486161 -11.155298            0  Ruvuma / Southern Coast
4      31.130847  -1.825359            0           Lake Victoria
…            …          …            …                      …
59395  37.169807  -3.253847            0                  Pangani
59396  35.249991  -9.070629            0                   Rufiji
59397  34.017087  -8.750434            0                   Rufiji
59398  35.861315  -6.378573            0                   Rufiji
59399  38.104048  -6.747464            0             Wami / Ruvu

            region  …   management  management_group payment_type  \
0           Iringa  …          vwc        user-group     annually
1             Mara  …          wug        user-group    never pay
2          Manyara  …          vwc        user-group   per bucket
3           Mtwara  …          vwc        user-group    never pay
4           Kagera  …        other             other    never pay
…             …  …          …                …          …
59395  Kilimanjaro  …  water board        user-group   per bucket
59396       Iringa  …          vwc        user-group     annually
59397        Mbeya  …          vwc        user-group      monthly
59398       Dodoma  …          vwc        user-group    never pay
59399     Morogoro  …          vwc        user-group   on failure

      water_quality quality_group quantity_group                source  \
0              soft          good         enough                spring
1              soft          good   insufficient  rainwater harvesting
2              soft          good         enough                   dam
3              soft          good            dry           machine dbh
4              soft          good       seasonal  rainwater harvesting
…                …            …            …                   …
59395          soft          good         enough                spring
```

```
59396          soft          good         enough                  river
59397      fluoride      fluoride         enough            machine dbh
59398          soft          good   insufficient           shallow well
59399         salty         salty         enough           shallow well

      source_class waterpoint_type_group status_group
0      groundwater    communal standpipe            1
1          surface    communal standpipe            1
2          surface    communal standpipe            1
3      groundwater    communal standpipe            0
4          surface    communal standpipe            1
…              …                    …           …
59395  groundwater    communal standpipe            1
59396      surface    communal standpipe            1
59397  groundwater             hand pump            1
59398  groundwater             hand pump            1
59399  groundwater             hand pump            1

[59400 rows x 29 columns]
```

We can also combine date_recorded with construction year by subtracting the two and storing them as a year

[22]:
```
from sklearn.preprocessing import OneHotEncoder
from datetime import datetime
prep_df_values['date_recorded'] = prep_df_values['date_recorded'].apply(lambda␣
  ↪row :row[:4])
prep_df_values['years_active'] = prep_df_values['date_recorded'].astype(int) -␣
  ↪prep_df_values['construction_year'].astype(int)

#Convert years with 0 as construction year to simply 0
prep_df_values['years_active'] = prep_df_values['years_active'].apply(lambda␣
  ↪year: year if year < 100 else np.nan)
#TEMP TRYING
prep_df_values[['amount_tsh', 'gps_height', 'longitude', 'latitude',␣
  ↪'population']] = prep_df_values[['amount_tsh', 'gps_height', 'longitude',␣
  ↪'latitude', 'population']].replace(0, np.nan)
prep_df_values.replace('unknown', np.nan)
#Now drop the two other columns
prep_df_values.drop(columns=['date_recorded', 'construction_year'],␣
  ↪inplace=True)

prep_df_values
```

[22]:
```
      amount_tsh          funder  gps_height     installer  longitude  \
0         6000.0           Roman      1390.0         Roman  34.938093
1            NaN         Grumeti      1399.0       GRUMETI  34.698766
```

```
2            25.0     Lottery Club       686.0  World vision  37.460664
3             NaN           Unicef       263.0        UNICEF  38.486161
4             NaN      Action In A         NaN       Artisan  31.130847
…           …              …       …           …         …
59395        10.0  Germany Republi      1210.0           CES  37.169807
59396      4700.0      Cefa-njombe      1212.0          Cefa  35.249991
59397         NaN              NaN         NaN           NaN  34.017087
59398         NaN            Malec         NaN          Musa  35.861315
59399         NaN       World Bank       191.0         World  38.104048

        latitude  num_private                    basin       region  \
0      -9.856322            0               Lake Nyasa       Iringa
1      -2.147466            0            Lake Victoria         Mara
2      -3.821329            0                  Pangani      Manyara
3     -11.155298            0  Ruvuma / Southern Coast       Mtwara
4      -1.825359            0            Lake Victoria       Kagera
…          …           …                      …          …
59395  -3.253847            0                  Pangani  Kilimanjaro
59396  -9.070629            0                   Rufiji       Iringa
59397  -8.750434            0                   Rufiji        Mbeya
59398  -6.378573            0                   Rufiji       Dodoma
59399  -6.747464            0              Wami / Ruvu     Morogoro

       district_code  …  management_group payment_type water_quality  \
0                  5  …        user-group     annually          soft
1                  2  …        user-group    never pay          soft
2                  4  …        user-group   per bucket          soft
3                 63  …        user-group    never pay          soft
4                  1  …             other    never pay          soft
…               …  …              …          …             …
59395              5  …        user-group   per bucket          soft
59396              4  …        user-group     annually          soft
59397              7  …        user-group      monthly      fluoride
59398              4  …        user-group    never pay          soft
59399              2  …        user-group   on failure         salty

      quality_group quantity_group                source source_class  \
0              good         enough                spring  groundwater
1              good   insufficient  rainwater harvesting      surface
2              good         enough                   dam      surface
3              good            dry           machine dbh  groundwater
4              good       seasonal  rainwater harvesting      surface
…              …             …                   …          …
59395          good         enough                spring  groundwater
59396          good         enough                 river      surface
59397      fluoride         enough           machine dbh  groundwater
59398          good   insufficient          shallow well  groundwater
```

```
59399          salty         enough          shallow well  groundwater

      waterpoint_type_group status_group years_active
0        communal standpipe            1         12.0
1        communal standpipe            1          3.0
2        communal standpipe            1          4.0
3        communal standpipe            0         27.0
4        communal standpipe            1          NaN
…                       …          …          …
59395    communal standpipe            1         14.0
59396    communal standpipe            1         15.0
59397             hand pump            1          NaN
59398             hand pump            1          NaN
59399             hand pump            1          9.0

[59400 rows x 28 columns]
```

Here we will create a dataframe for specifically the categorical columns

[23]:
```python
df_categoricals = prep_df_values.select_dtypes(exclude=np.number)
df_categoricals
```

[23]:
```
                funder     installer                    basin       region  \
0                Roman         Roman               Lake Nyasa       Iringa
1              Grumeti       GRUMETI           Lake Victoria         Mara
2         Lottery Club  World vision                  Pangani      Manyara
3               Unicef        UNICEF  Ruvuma / Southern Coast       Mtwara
4            Action In A       Artisan           Lake Victoria       Kagera
…                    …             …                        …          …
59395  Germany Republi           CES                  Pangani  Kilimanjaro
59396      Cefa-njombe          Cefa                   Rufiji       Iringa
59397              NaN           NaN                   Rufiji        Mbeya
59398            Malec          Musa                   Rufiji       Dodoma
59399       World Bank         World              Wami / Ruvu     Morogoro

      public_meeting              recorded_by scheme_management permit  \
0               True  GeoData Consultants Ltd               VWC  False
1                NaN  GeoData Consultants Ltd             Other   True
2               True  GeoData Consultants Ltd               VWC   True
3               True  GeoData Consultants Ltd               VWC   True
4               True  GeoData Consultants Ltd               NaN   True
…                  …                        …                 …     …
59395           True  GeoData Consultants Ltd       Water Board   True
59396           True  GeoData Consultants Ltd               VWC   True
59397           True  GeoData Consultants Ltd               VWC  False
59398           True  GeoData Consultants Ltd               VWC   True
59399           True  GeoData Consultants Ltd               VWC   True
```

```
      extraction_type_group extraction_type_class   management  \
0                   gravity               gravity          vwc
1                   gravity               gravity          wug
2                   gravity               gravity          vwc
3               submersible           submersible          vwc
4                   gravity               gravity        other
…                       …                     …          …
59395               gravity               gravity  water board
59396               gravity               gravity          vwc
59397                swn 80              handpump          vwc
59398           nira/tanira              handpump          vwc
59399           nira/tanira              handpump          vwc

      management_group payment_type water_quality quality_group  \
0           user-group     annually          soft          good
1           user-group    never pay          soft          good
2           user-group   per bucket          soft          good
3           user-group    never pay          soft          good
4                other    never pay          soft          good
…                  …            …             …            …
59395       user-group   per bucket          soft          good
59396       user-group     annually          soft          good
59397       user-group      monthly      fluoride      fluoride
59398       user-group    never pay          soft          good
59399       user-group   on failure         salty         salty

      quantity_group                source source_class waterpoint_type_group
0             enough                spring  groundwater    communal standpipe
1       insufficient  rainwater harvesting      surface    communal standpipe
2             enough                   dam      surface    communal standpipe
3                dry           machine dbh  groundwater    communal standpipe
4           seasonal  rainwater harvesting      surface    communal standpipe
…                …                    …           …                   …
59395         enough                spring  groundwater    communal standpipe
59396         enough                 river      surface    communal standpipe
59397         enough           machine dbh  groundwater             hand pump
59398   insufficient          shallow well  groundwater             hand pump
59399         enough          shallow well  groundwater             hand pump

[59400 rows x 19 columns]
```

2 Building up Logistic Regression Model

Now is the time to one hot encode the categorical columns to see if there is a correlation. For any columns that end up being encoded into hundreds or even thousands of columns, they aren't worth considering.

[24]:
```
y = prep_df_values['status_group']
X = prep_df_values[['amount_tsh', 'gps_height', 'longitude', 'latitude',␣
↪'num_private', 'population', 'years_active', 'district_code',
                    'basin', 'region', 'management_group', 'quality_group',␣
↪'permit', 'water_quality', 'management', 'source', 'extraction_type_class',␣
↪'extraction_type_group',
                    'waterpoint_type_group', 'quantity_group', 'payment_type']]
X
```

[24]:
```
       amount_tsh  gps_height  longitude   latitude  num_private  population  \
0          6000.0      1390.0  34.938093  -9.856322            0       109.0
1             NaN      1399.0  34.698766  -2.147466            0       280.0
2            25.0       686.0  37.460664  -3.821329            0       250.0
3             NaN       263.0  38.486161 -11.155298            0        58.0
4             NaN         NaN  31.130847  -1.825359            0         NaN
…               …           …          …          …            …
59395        10.0      1210.0  37.169807  -3.253847            0       125.0
59396      4700.0      1212.0  35.249991  -9.070629            0        56.0
59397         NaN         NaN  34.017087  -8.750434            0         NaN
59398         NaN         NaN  35.861315  -6.378573            0         NaN
59399         NaN       191.0  38.104048  -6.747464            0       150.0

       years_active  district_code                    basin       region  …  \
0              12.0              5               Lake Nyasa       Iringa  …
1               3.0              2            Lake Victoria         Mara  …
2               4.0              4                  Pangani      Manyara  …
3              27.0             63  Ruvuma / Southern Coast       Mtwara  …
4               NaN              1            Lake Victoria       Kagera  …
…                 …              …                        …          …  …
59395          14.0              5                  Pangani  Kilimanjaro  …
59396          15.0              4                   Rufiji       Iringa  …
59397           NaN              7                   Rufiji        Mbeya  …
59398           NaN              4                   Rufiji       Dodoma  …
59399           9.0              2              Wami / Ruvu     Morogoro  …

      quality_group permit water_quality   management               source  \
0              good  False          soft          vwc               spring
1              good   True          soft          wug  rainwater harvesting
2              good   True          soft          vwc                  dam
3              good   True          soft          vwc          machine dbh
4              good   True          soft        other  rainwater harvesting
…                 …    …             …            …                    …
59395          good   True          soft  water board               spring
59396          good   True          soft          vwc                river
59397      fluoride  False      fluoride          vwc          machine dbh
59398          good   True          soft          vwc         shallow well
59399         salty   True         salty          vwc         shallow well
```

```
      extraction_type_class extraction_type_group waterpoint_type_group  \
0                   gravity               gravity    communal standpipe
1                   gravity               gravity    communal standpipe
2                   gravity               gravity    communal standpipe
3               submersible           submersible    communal standpipe
4                   gravity               gravity    communal standpipe
…                     …                     …                     …
59395               gravity               gravity    communal standpipe
59396               gravity               gravity    communal standpipe
59397              handpump                swn 80             hand pump
59398              handpump           nira/tanira             hand pump
59399              handpump           nira/tanira             hand pump

      quantity_group payment_type
0             enough     annually
1       insufficient    never pay
2             enough   per bucket
3                dry    never pay
4           seasonal    never pay
…                …            …
59395         enough   per bucket
59396         enough     annually
59397         enough      monthly
59398   insufficient    never pay
59399         enough   on failure

[59400 rows x 21 columns]
```

Here we see that with the new relevant numerical columns and the one hot encoded columns that are worth exploring, we actually increased our accuracy metric to 78 from around 65.

[25]:
```python
from sklearn.preprocessing import StandardScaler
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import SimpleImputer, IterativeImputer
from imblearn.over_sampling import SMOTE
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state=41)

num_imp = SimpleImputer(strategy='median')
cat_imp = SimpleImputer(strategy='most_frequent')
#Impute numeric and categorical columns for train and test set
X_train_num = X_train[['amount_tsh', 'gps_height', 'longitude', 'latitude',␣
 ↪'num_private', 'population' ,'years_active', 'district_code']]
```

```
X_train_cat = X_train[['basin', 'region' , 'management_group', 'quality_group', 'permit', 'waterpoint_type_group', 'quantity_group' ,'payment_type', 'water_quality', 'management', 'source', 'extraction_type_group', 'extraction_type_class']]
X_test_num = X_test[['amount_tsh', 'gps_height', 'longitude', 'latitude', 'num_private', 'population' ,'years_active', 'district_code']]
X_test_cat = X_test[['basin', 'region' , 'management_group', 'quality_group', 'permit', 'waterpoint_type_group', 'quantity_group' ,'payment_type', 'water_quality', 'management', 'source', 'extraction_type_group', 'extraction_type_class']]


#Fit transform for train, transform test
num_arr = num_imp.fit_transform(X_train_num)
cat_arr = cat_imp.fit_transform(X_train_cat)
num_t_arr = num_imp.transform(X_test_num)
cat_t_arr = cat_imp.transform(X_test_cat)

#One Hot Encode Train
X_train_imp = pd.concat([pd.DataFrame(num_arr, columns=X_train_num.columns, index= X_train_num.index), pd.DataFrame(cat_arr, columns=X_train_cat.columns, index=X_train_cat.index)], axis=1)
X_train_imp = pd.get_dummies(X_train_imp,
                             columns=['basin', 'region', 'management_group', 'quality_group', 'permit', 'waterpoint_type_group', 'quantity_group', 'payment_type', 'water_quality', 'management', 'source','extraction_type_group', 'extraction_type_class'], drop_first=True, dtype=int)
#Test
X_test_imp = pd.concat([pd.DataFrame(num_t_arr, columns=X_test_num.columns, index= X_test_num.index), pd.DataFrame(cat_t_arr, columns=X_test_cat.columns, index=X_test_cat.index)], axis=1)
X_test_imp = pd.get_dummies(X_test_imp,
                            columns=['basin', 'region', 'management_group', 'quality_group', 'permit', 'waterpoint_type_group', 'quantity_group', 'payment_type', 'water_quality', 'management', 'source','extraction_type_group', 'extraction_type_class'], drop_first=True, dtype=int)


#Oversample
smote = SMOTE(random_state=42, sampling_strategy= 1)
X_train_samp, y_train_samp = smote.fit_resample(X_train_imp, y_train)
#X_train_samp, y_train_samp = X_train_imp, y_train

#Fit model
```

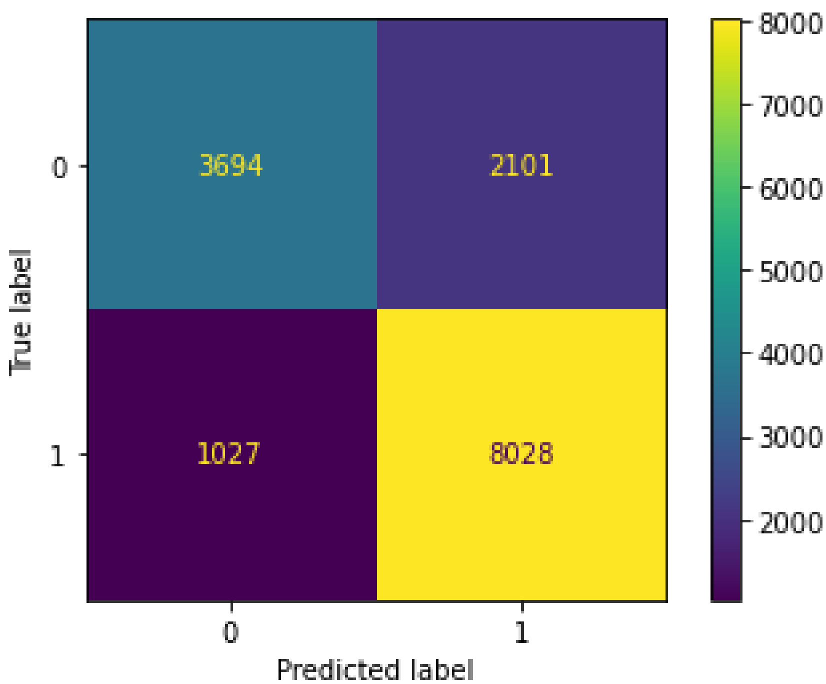
```
model = LogisticRegression(solver='liblinear', max_iter=10000)
model.fit(X_train_samp, y_train_samp)

y_train_pred = model.predict(X_train_samp)
y_pred = model.predict(X_test_imp)

print(f"Train Acc: {accuracy_score(y_train_samp, y_train_pred)}\nTest Acc:␣
↪{accuracy_score(y_test, y_pred)}")
print(f"Specificity(Recall) Score: {recall_score(y_test, y_pred, pos_label=0)}")
print(f"Recall Score: {recall_score(y_test, y_pred)}")
```

```
Train Acc: 0.8042949020747793
Test Acc: 0.7893602693602694
Specificity(Recall) Score: 0.6374460742018981
Recall Score: 0.8865819988956378
```

We can see that we have increased the number of true negatives, but we still have a large amount of the false metrics. In fact, we got a little more false negatives.

[26]:
```
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(cm)
disp.plot()
```

[26]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29cb649a7c0>

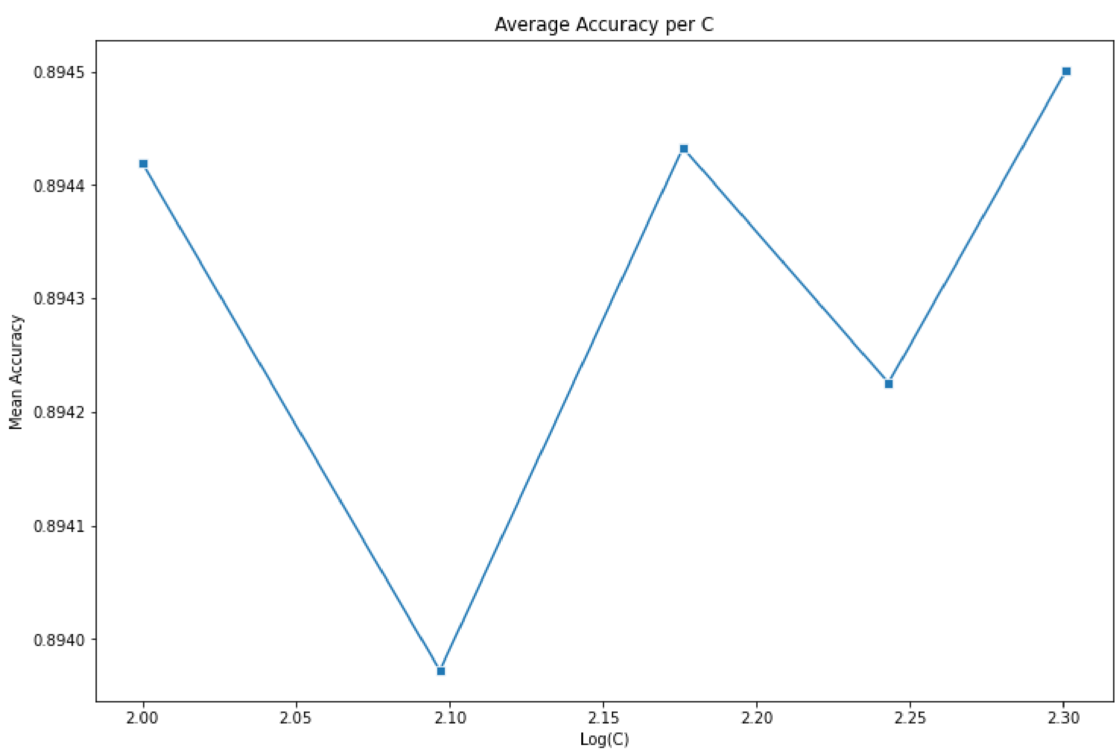


3 Hyperparameter Tuning (RAW)

Now that our accuracy is a good bit higher than where we started, we can begin tuning the hyperparameters to see if we can get a better accuracy

[27]:
```
C_list = [100, 125, 150, 175, 200]
cv_scores = []

for c in C_list:
    logreg = LogisticRegression(C = c, solver='liblinear', max_iter=10000)
    cv_loop_results = cross_validate(X = X_train_samp,y = y_train_samp,␣
 ↪estimator=logreg, cv=10)
    cv_scores.append(np.mean(np.sqrt(np.abs(cv_loop_results['test_score']))))
    print(f"Done with C = {c}")

fig, ax = plt.subplots(figsize=(12, 8))
sns.lineplot(x = np.log10(C_list), y = cv_scores, marker = 's')
ax.set_xlabel('Log(C)')
ax.set_ylabel('Mean Accuracy')
ax.set_title('Average Accuracy per C');
```

```
Done with C = 100
Done with C = 125
Done with C = 150
Done with C = 175
Done with C = 200
```



[28]:
```
print(cv_scores)
index = cv_scores.index(max(cv_scores))
print(f"A C of {C_list[index]} is the best performing c with a score of:␣
↪{cv_scores[index]}")
```

```
[0.8944189106752531, 0.8939721736670734, 0.8944327842262105, 0.8942255197636193,
0.8945013355816519]
A C of 200 is the best performing c with a score of: 0.8945013355816519
```

With the cross validation testing finding that a c of 100 performs the best, we can put this back into our model and see if this reflects.

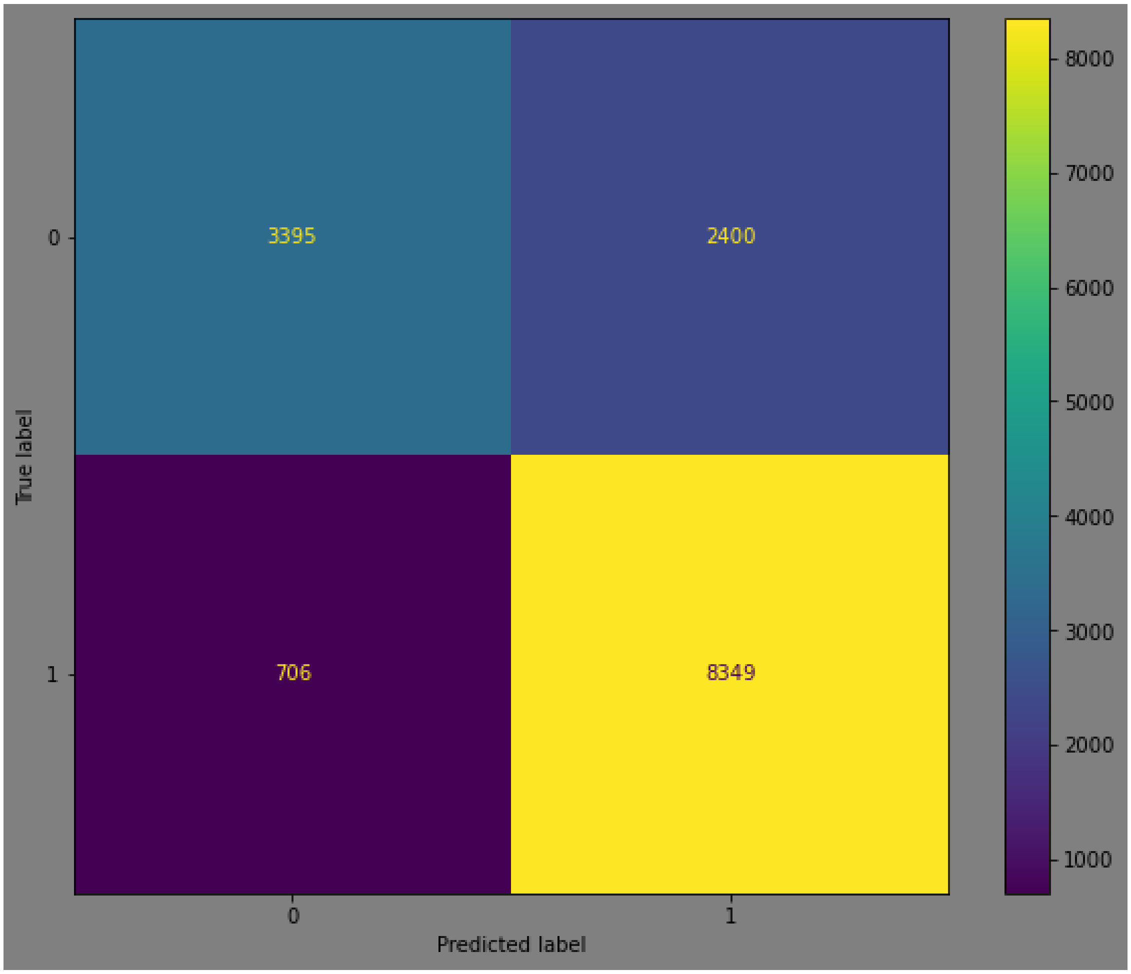
[29]:
```
h_model = LogisticRegression(solver='liblinear', C=200, max_iter=10000)
h_model.fit(X_train_imp, y_train)

y_htrain_pred = h_model.predict(X_train_imp)
y_hpred = h_model.predict(X_test_imp)
print(f"Train Acc: {accuracy_score(y_train, y_htrain_pred)}\nTest Acc:␣
↪{accuracy_score(y_test, y_hpred)}")
print(f"Specificity(Recall) Score: {recall_score(y_test, y_hpred,␣
↪pos_label=0)}")
print(f"Recall Score: {recall_score(y_test, y_hpred)}")
```

```
Train Acc: 0.790280583613917
Test Acc: 0.7908417508417508
Specificity(Recall) Score: 0.5858498705780846
Recall Score: 0.9220320265046935
```

[30]:
```
list_acc.append(accuracy_score(y_test, y_hpred))
list_spec.append(recall_score(y_test, y_hpred, pos_label=0))
list_rec.append(recall_score(y_test, y_hpred))
```

[31]:
```
fig, ax = plt.subplots(figsize=(10,8))
fig.set_facecolor('grey')
cm = confusion_matrix(y_test, y_hpred)
disp = ConfusionMatrixDisplay(cm)
disp.plot(ax=ax)
```

[31]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29cbd1feaf0>
```



We may be approaching the limit of what can be done with a logistic regression model. We got a peak test accuracy of around 79 percent. We will now move on to our second model to see if we can improve our score

4 Building a Random Forest classifier

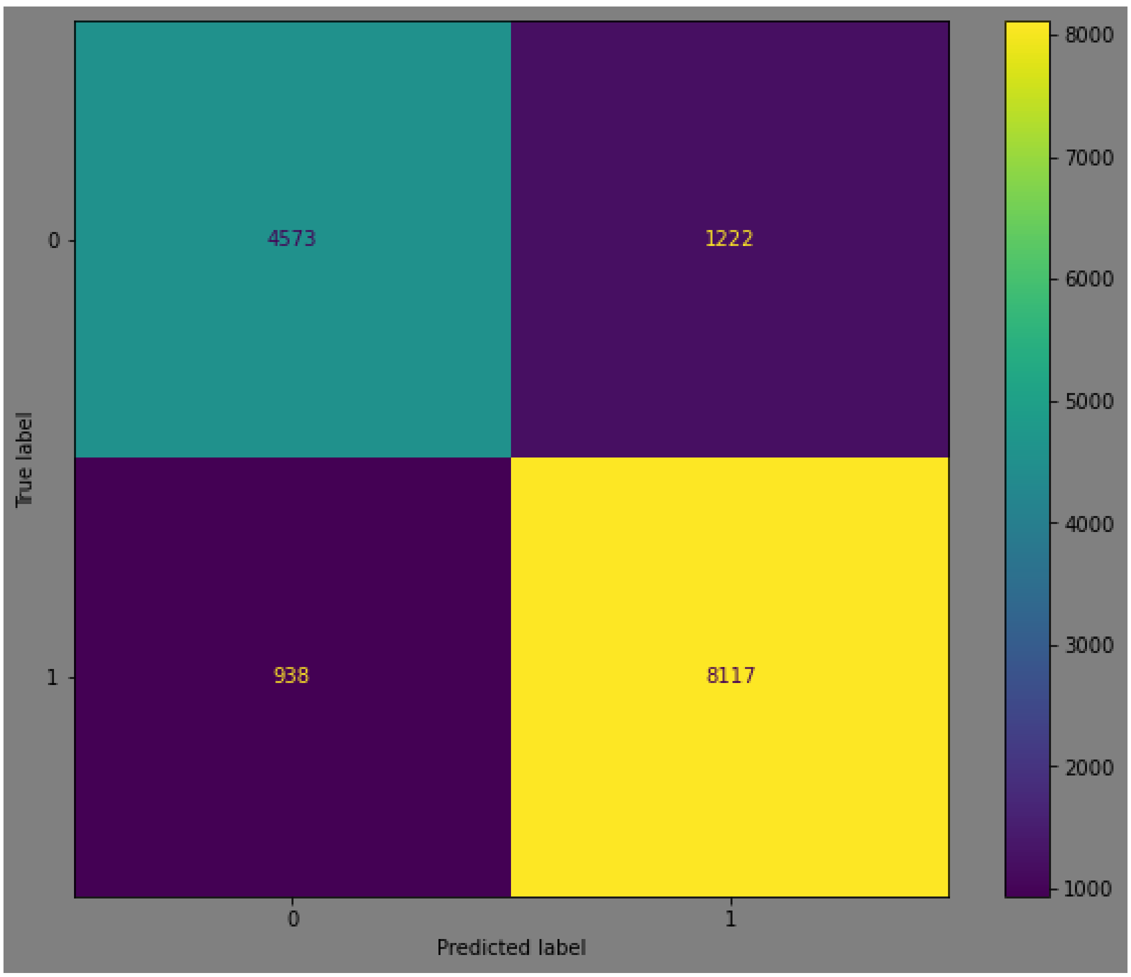
[32]:
```
from sklearn.ensemble import RandomForestClassifier

clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_samp, y_train_samp)

train_tree_pred = clf.predict(X_train_samp)
test_tree_pred = clf.predict(X_test_imp)
print(f"Train Acc: {accuracy_score(y_train_samp, train_tree_pred)}\nTest Acc:␣
↪{accuracy_score(y_test, test_tree_pred)}")
print(f"Specificity(Recall) Score: {recall_score(y_test, test_tree_pred,␣
↪pos_label=0)}")
```

```
print(f"Recall Score: {recall_score(y_test, test_tree_pred)}")
```

```
Train Acc: 0.9984738926637841
Test Acc: 0.8545454545454545
Specificity(Recall) Score: 0.7891285591026748
Recall Score: 0.896410822749862
```

[33]:
```
fig, ax = plt.subplots(figsize=(10,8))
cm = confusion_matrix(y_test, test_tree_pred)
fig.set_facecolor('grey')
disp = ConfusionMatrixDisplay(cm)
disp.plot(ax=ax)
```

[33]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29cbd295b80>



We can see that this is already considerably better than the previous logistic regression model. However, there is a fair amount of overfitting so we can now tune some hyperparameters to see if we can improve it.

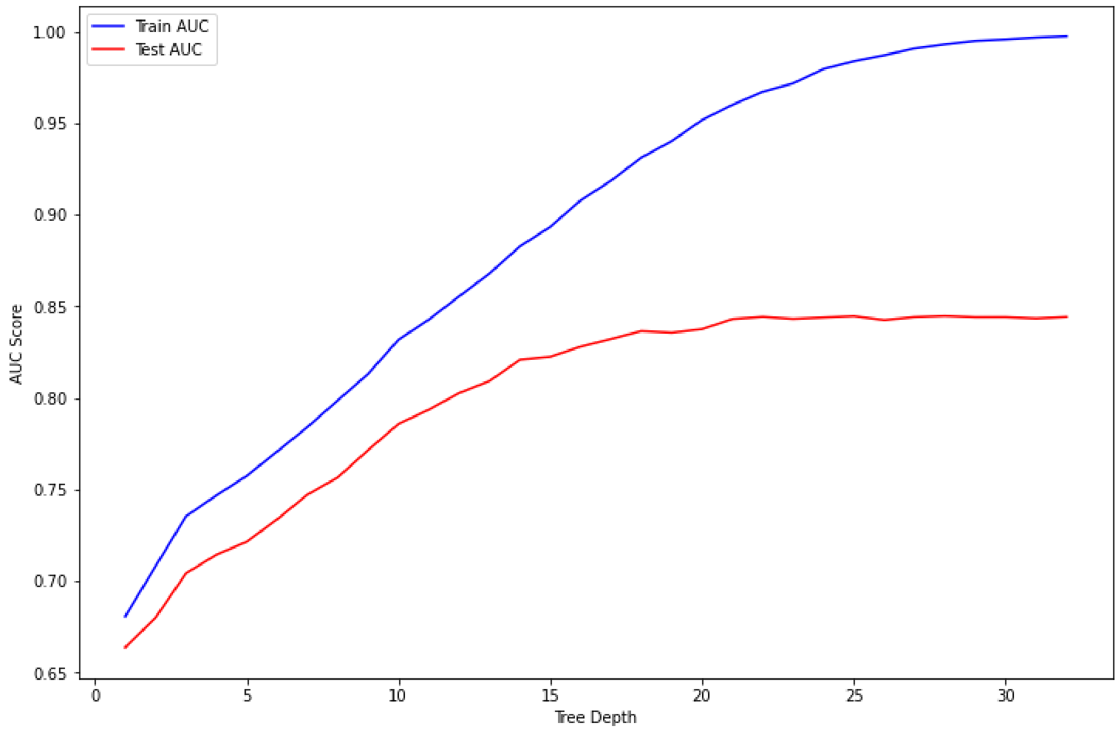
5 Max Tree Depth (RAW)

[34]:

```
# Identify the optimal tree depth for given data
from sklearn.metrics import auc, roc_curve
SEED = 42
depths = np.arange(1, 33)
train_results = []
test_results = []
for depth in depths:
    dt_d = RandomForestClassifier(criterion='entropy', random_state=SEED,␣
 ↪max_depth=depth)
    dt_d.fit(X_train_samp, y_train_samp)

    y_train_pred = dt_d.predict(X_train_samp)
    fpr, tpr, thresholds = roc_curve(y_train_samp, y_train_pred)
    roc_auc = auc(fpr, tpr)
    train_results.append(roc_auc)

    y_test_pred = dt_d.predict(X_test_imp)
    fpr, tpr, thresholds = roc_curve(y_test, y_test_pred)
    roc_auc = auc(fpr, tpr)
    test_results.append(roc_auc)

fig, ax = plt.subplots(figsize=(12, 8))
plt.plot(depths, train_results, 'b', label='Train AUC')
plt.plot(depths, test_results, 'r', label='Test AUC')
plt.legend()
plt.xlabel('Tree Depth')
plt.ylabel('AUC Score')
```

[34]:

```
Text(0, 0.5, 'AUC Score')
```



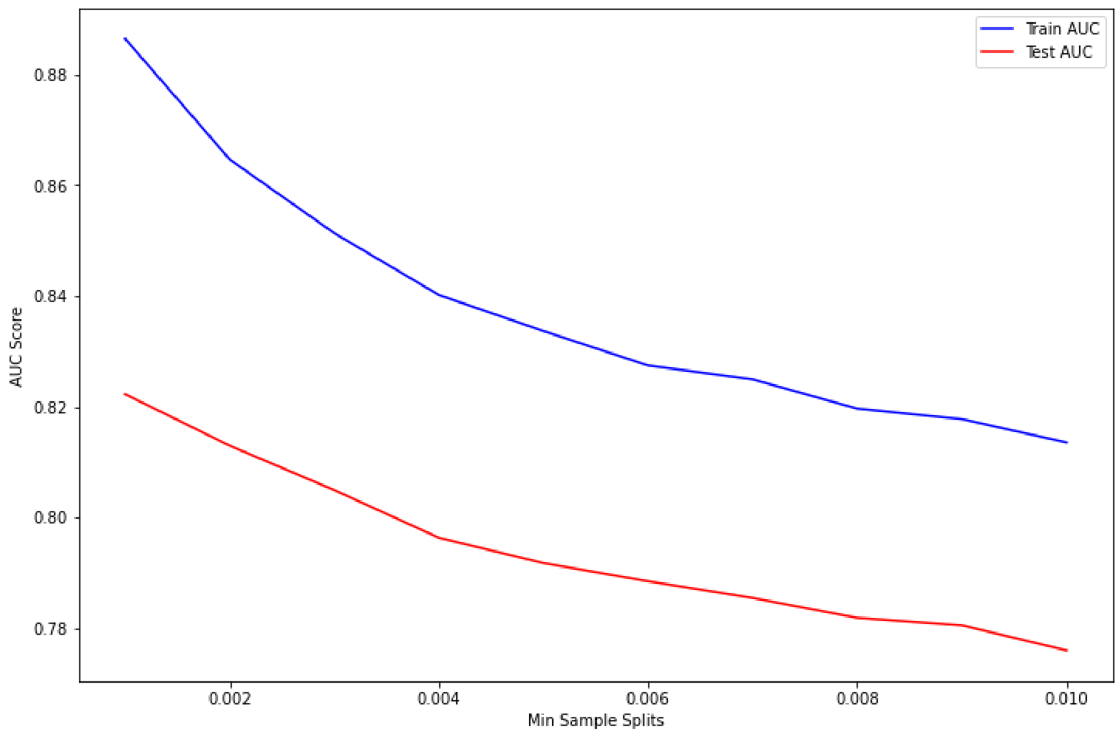
Best max tree depth is around 12-17

6 Minimum Sample Split (RAW)

[35]:
```python
# Identify the optimal min-samples-split for given data
min_sample_splits = np.linspace(0.001, 0.01, 10)
train_results = []
test_results = []
for sample in min_sample_splits:
    dt_d = RandomForestClassifier(criterion='entropy', random_state=SEED,␣
 ↪min_samples_split=sample)
    dt_d.fit(X_train_samp, y_train_samp)

    y_train_pred = dt_d.predict(X_train_samp)
    fpr, tpr, thresholds = roc_curve(y_train_samp, y_train_pred)
    roc_auc = auc(fpr, tpr)
    train_results.append(roc_auc)

    y_test_pred = dt_d.predict(X_test_imp)
    fpr, tpr, thresholds = roc_curve(y_test, y_test_pred)
    roc_auc = auc(fpr, tpr)
    test_results.append(roc_auc)
```

```
fig, ax = plt.subplots(figsize=(12, 8))
plt.plot(min_sample_splits, train_results, 'b', label='Train AUC')
plt.plot(min_sample_splits, test_results, 'r', label='Test AUC')
plt.legend()
plt.xlabel('Min Sample Splits')
plt.ylabel('AUC Score')
```

[35]: Text(0, 0.5, 'AUC Score')



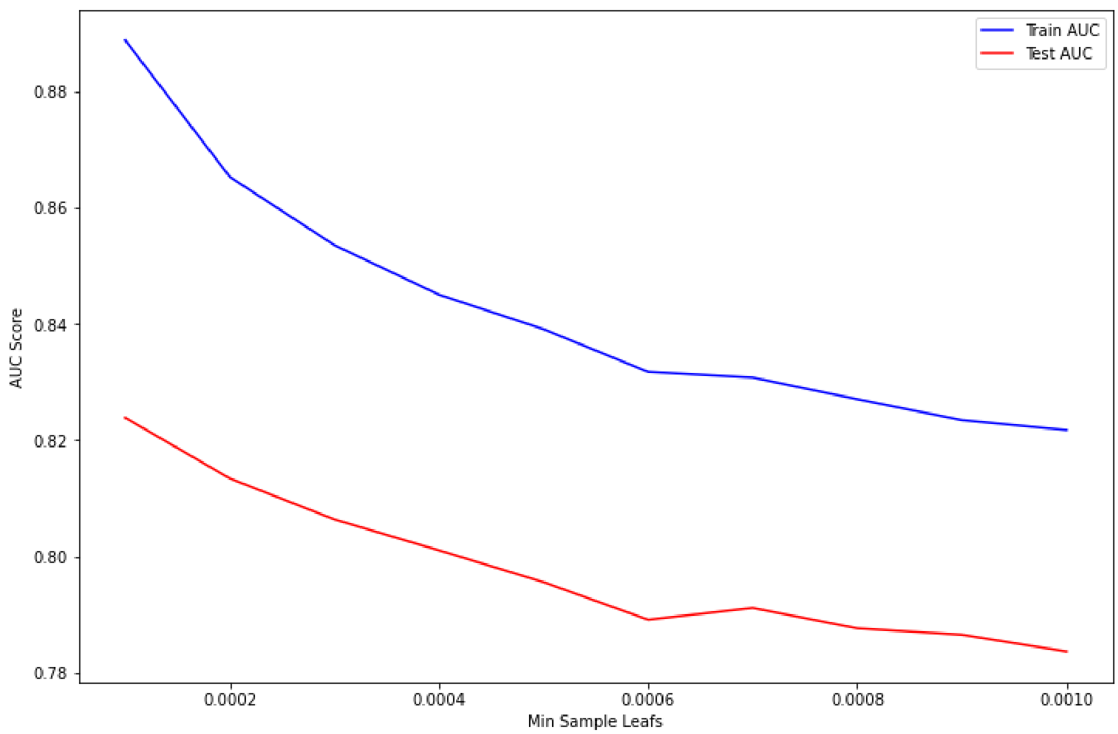
Best minimum sample split is around 0.1

7 Minimum Sample Leafs (RAW)

[36]:
```
# Calculate the optimal value for minimum sample leafs
min_sample_leafs = np.linspace(0.0001, 0.001, 10)
train_results = []
test_results = []
for leafs in min_sample_leafs:
    dt_d = RandomForestClassifier(criterion='entropy', random_state=SEED,␣
↪min_samples_leaf=leafs)
    dt_d.fit(X_train_samp, y_train_samp)

    y_train_pred = dt_d.predict(X_train_samp)
```

```
    fpr, tpr, thresholds = roc_curve(y_train_samp, y_train_pred)
    roc_auc = auc(fpr, tpr)
    train_results.append(roc_auc)

    y_test_pred = dt_d.predict(X_test_imp)
    fpr, tpr, thresholds = roc_curve(y_test, y_test_pred)
    roc_auc = auc(fpr, tpr)
    test_results.append(roc_auc)

fig, ax = plt.subplots(figsize=(12, 8))
plt.plot(min_sample_leafs, train_results, 'b', label='Train AUC')
plt.plot(min_sample_leafs, test_results, 'r', label='Test AUC')
plt.legend()
plt.xlabel('Min Sample Leafs')
plt.ylabel('AUC Score')
```

[36]: Text(0, 0.5, 'AUC Score')

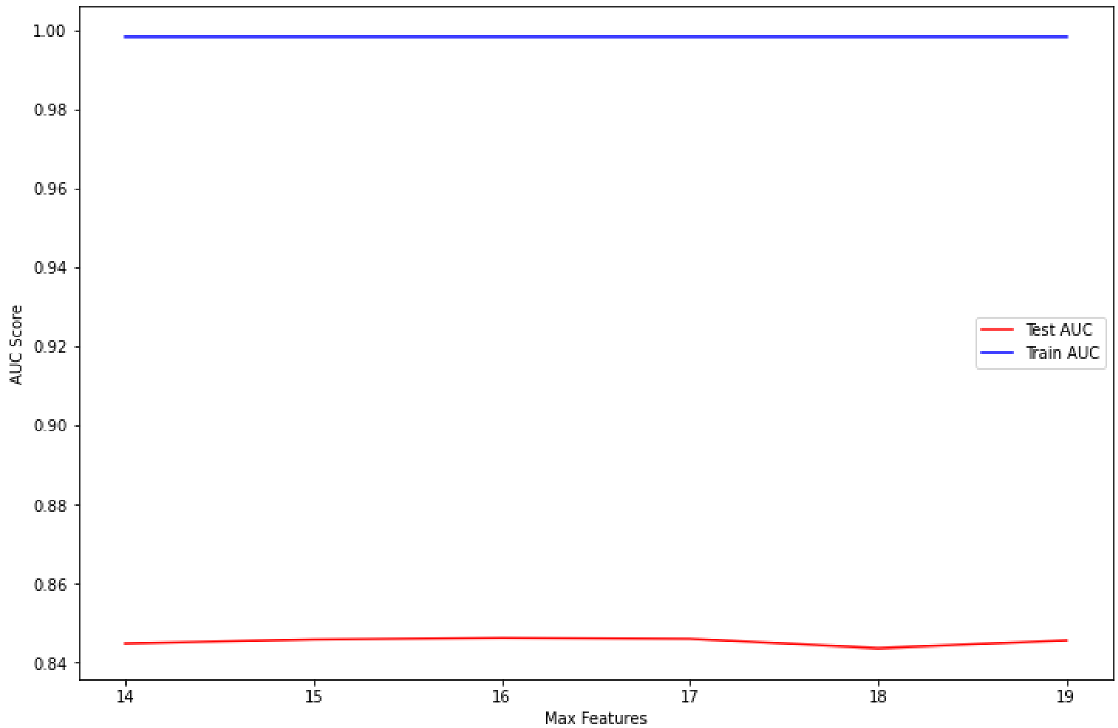


The best minimum sample leaf is around 0.1

8 Maximum Features (RAW)

[37]:
```
# Find the best value for optimal maximum feature size
max_features = np.arange(14, 20)
test_results = []
train_results = []

for feature in max_features:
    dt = RandomForestClassifier(criterion='entropy', random_state=SEED,␣
 ↪max_features=feature)
    dt.fit(X_train_samp, y_train_samp)

    y_train_pred = dt.predict(X_train_samp)
    fpr, tpr, thresholds = roc_curve(y_train_samp, y_train_pred)
    auc_score = auc(fpr, tpr)
    train_results.append(auc_score)

    y_test_pred = dt.predict(X_test_imp)
    fpr, tpr, thresholds = roc_curve(y_test, y_test_pred)
    auc_score = auc(fpr, tpr)
    test_results.append(auc_score)

fig, ax = plt.subplots(figsize=(12, 8))
plt.plot(max_features, test_results, 'r', label='Test AUC')
plt.plot(max_features, train_results, 'b', label='Train AUC')
plt.legend()
plt.ylabel('AUC Score')
plt.xlabel('Max Features')
```

[37]:
```
Text(0.5, 0, 'Max Features')
```



The best maximum features count is around 19

With all the features optimized, we took a slight hit to accuracy, but managed to bring the test and train accuracy more inline with eachother. However, the specificity score it considerably lower than the recall score. In this case, we deemed it to be more beneficial to get a false negative than a false positive: a broken well being predicted as functional would be worse than a functioning well being predicted as broken. We will try to increase specificity, even at the cost of recall and overall accuracy to a limit.

[38]:
```
from sklearn.ensemble import RandomForestClassifier

#clf = RandomForestClassifier(random_state=42, min_samples_leaf=15,␣
↪max_depth=20, max_features=19, min_samples_split=0.001, class_weight={0: 1,␣
↪1: 0.75})
clf = RandomForestClassifier(random_state=42, min_samples_leaf=20,␣
↪max_depth=15, max_features=19, min_samples_split=0.001)
#clf = RandomForestClassifier(random_state=42)
clf.fit(X_train_samp, y_train_samp)

train_tree_pred = clf.predict(X_train_samp)
test_tree_pred = clf.predict(X_test_imp)
print(f"Train Acc: {accuracy_score(y_train_samp, train_tree_pred)}\nTest Acc:␣
↪{accuracy_score(y_test, test_tree_pred)}")
```

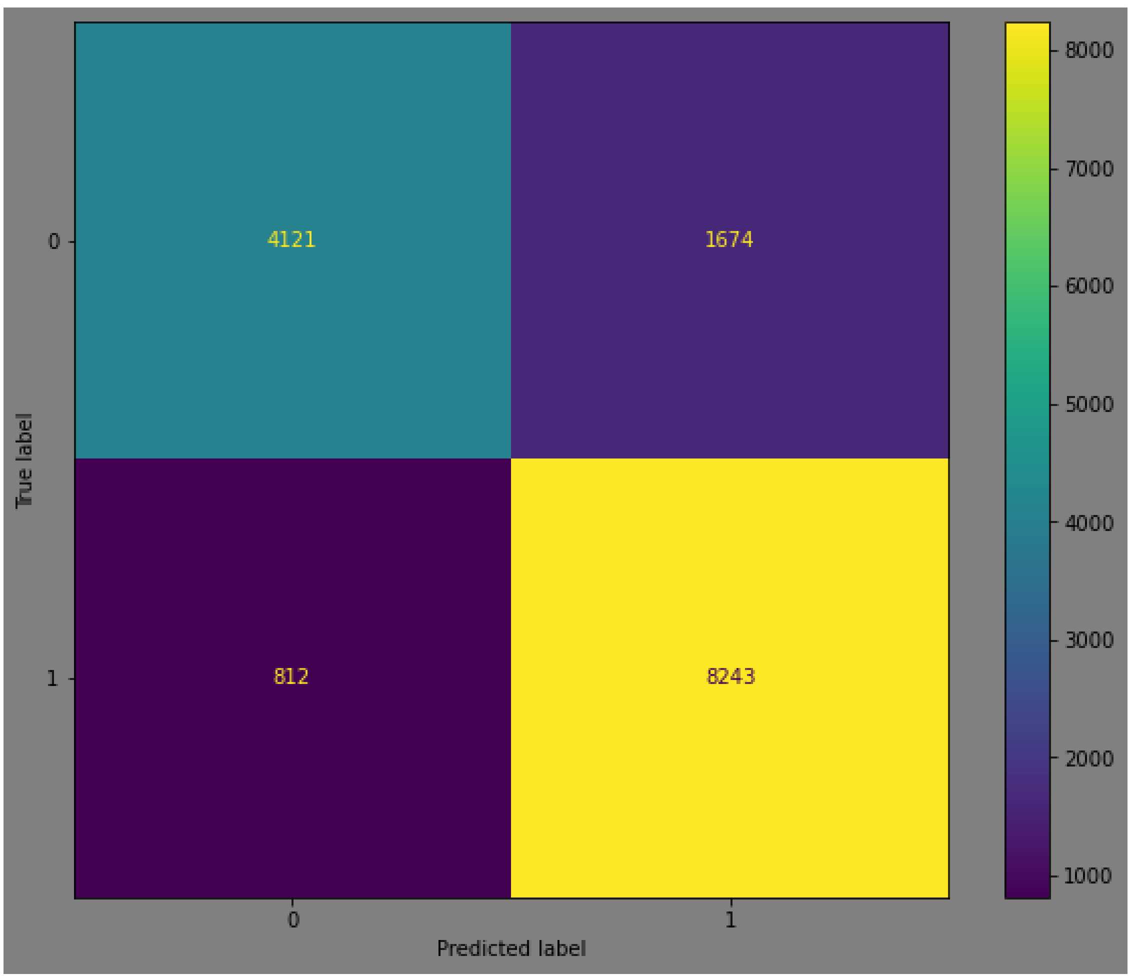
```
print(f"Specificity(Recall) Score: {recall_score(y_test, test_tree_pred,␣
↪pos_label=0)}")
print(f"Recall Score: {recall_score(y_test, test_tree_pred)}")
```

```
Train Acc: 0.8546019403364703
Test Acc: 0.8325925925925926
Specificity(Recall) Score: 0.7111302847282139
Recall Score: 0.9103257868580894
```

[39]:
```
list_acc.append(accuracy_score(y_test, test_tree_pred))
list_spec.append(recall_score(y_test, test_tree_pred, pos_label=0))
list_rec.append(recall_score(y_test, test_tree_pred))
```

[40]:
```
fig, ax = plt.subplots(figsize=(10,8))
cm = confusion_matrix(y_test, test_tree_pred)
fig.set_facecolor('grey')
disp = ConfusionMatrixDisplay(cm)
disp.plot(ax=ax)
```

[40]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29cba232400>
```



We can specify class weights to make the model less likely to predict positive.

[41]:
```
clf = RandomForestClassifier(random_state=42, min_samples_leaf=20,␣
↪max_depth=15, max_features=19, min_samples_split=0.001, class_weight={0: 1,␣
↪1: 0.65})
clf.fit(X_train_samp, y_train_samp)

train_tree_pred = clf.predict(X_train_samp)
test_tree_pred = clf.predict(X_test_imp)
print(f"Train Acc: {accuracy_score(y_train_samp, train_tree_pred)}\nTest Acc:␣
↪{accuracy_score(y_test, test_tree_pred)}")
print(f"Specificity(Recall) Score: {recall_score(y_test, test_tree_pred,␣
↪pos_label=0)}")
print(f"Recall Score: {recall_score(y_test, test_tree_pred)}")
```

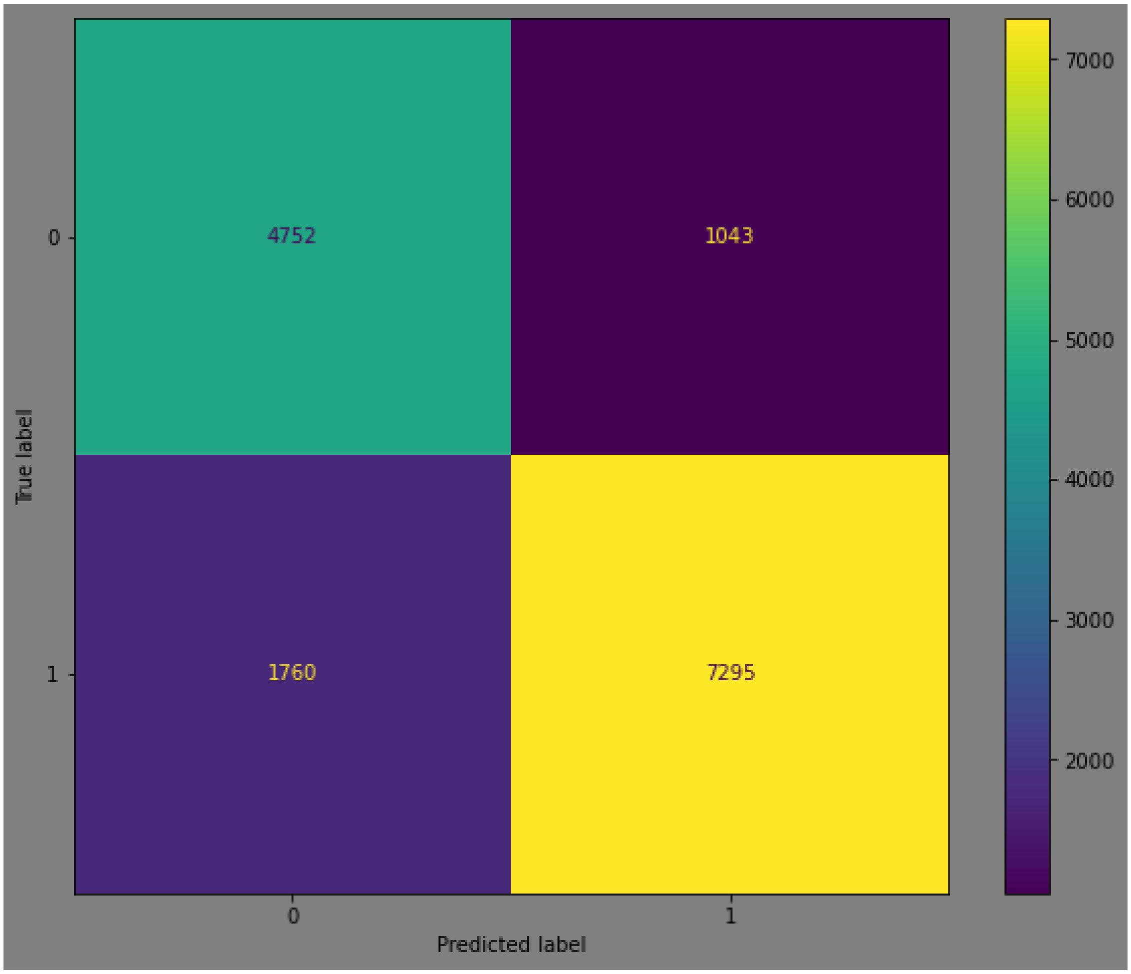
```
Train Acc: 0.8514043821082082
Test Acc: 0.8112457912457912
Specificity(Recall) Score: 0.8200172562553926
Recall Score: 0.8056322473771397
```

[42]:
```
list_acc.append(accuracy_score(y_test, test_tree_pred))
list_spec.append(recall_score(y_test, test_tree_pred, pos_label=0))
list_rec.append(recall_score(y_test, test_tree_pred))
```

This is around the best we can do with train and test accuracy are more in line with eachother. We did take a hit to both training and test acc, but since they are closer together, there should be less variance when tested on other data.

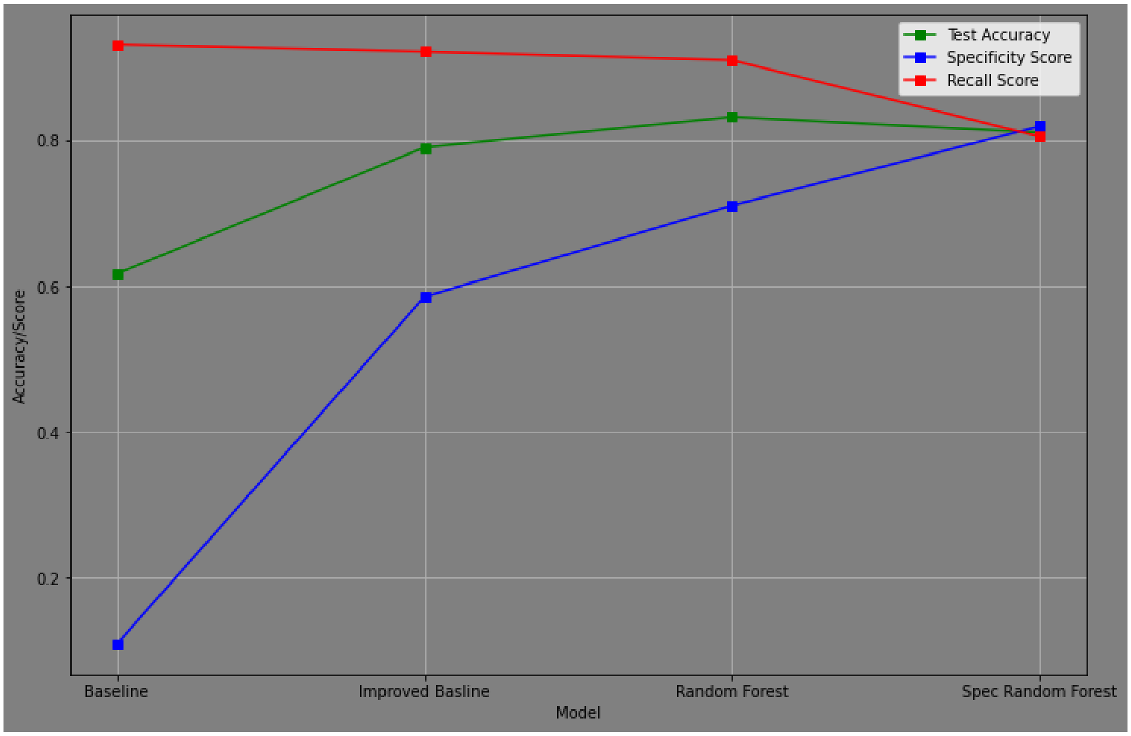
[43]:
```
fig, ax = plt.subplots(figsize=(10,8))
cm = confusion_matrix(y_test, test_tree_pred)
fig.set_facecolor('grey')
disp = ConfusionMatrixDisplay(cm)
disp.plot(ax=ax)
```

[43]:
```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x29cc26f8e50>
```



9 Line chart of accuracy over time

[44]:
```
fig, ax = plt.subplots(figsize=(12, 8))
x_arr = ['Baseline', 'Improved Basline', 'Random Forest', 'Spec Random Forest']
plt.plot(x_arr, list_acc, label='Test Accuracy', c='g', marker='s')
plt.plot(x_arr, list_spec, label='Specificity Score', c='b', marker='s')
plt.plot(x_arr, list_rec, label='Recall Score', c='r', marker='s')
ax.set_facecolor('grey')
fig.set_facecolor('grey')

plt.legend()
plt.grid()
plt.xlabel('Model')
plt.ylabel('Accuracy/Score');
```

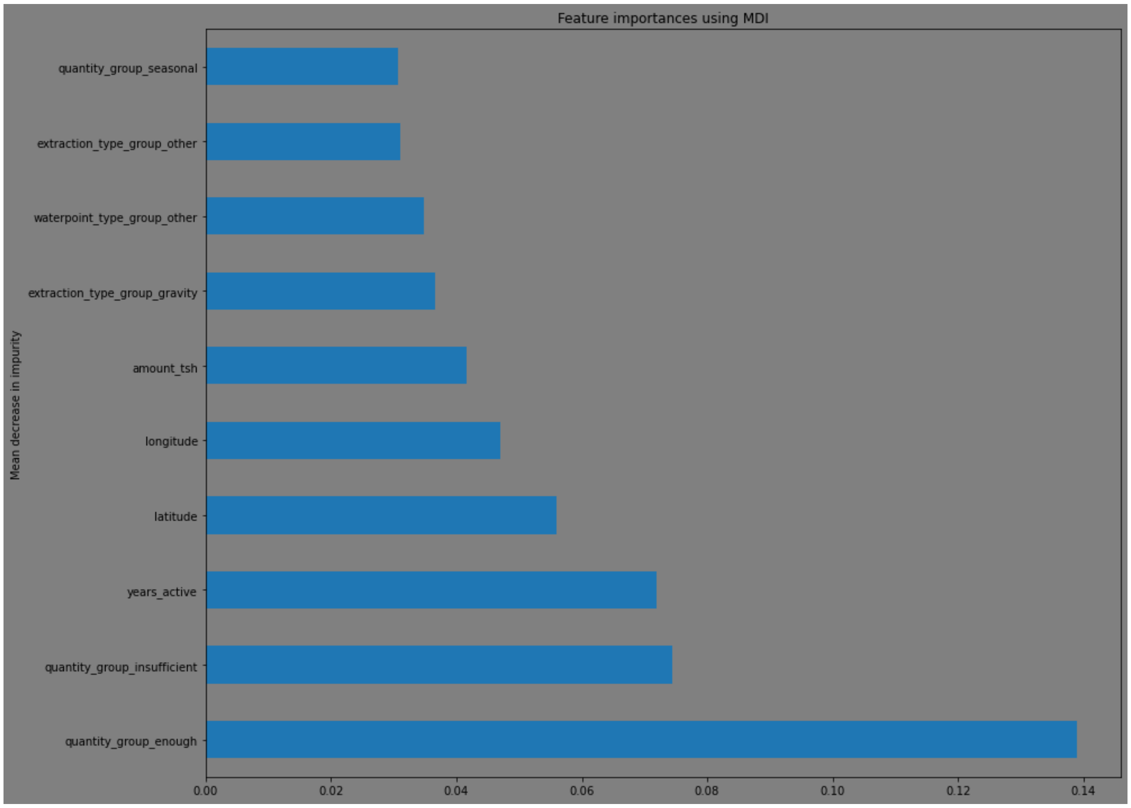


10 Feature Importance

Next we will try to determine which features played a bigger role in determining the target

[45]:
```
#Mean decrease in impurity
importances = clf.feature_importances_
names = X_train_samp.columns
mdi = {k:v for (k,v) in zip(names, importances)}

#sort by importance values
#top_20_mdi = sorted(mdi.items(), key=lambda item: item[1], reverse=True)[:20]
mdi_ser = pd.Series(mdi)
top_20_mdi = mdi_ser.sort_values(ascending=False)[:10]
```

[46]:
```
fig, ax = plt.subplots(figsize=(14, 10))
top_20_mdi.plot.barh(ax=ax)
fig.set_facecolor('grey')
ax.set_facecolor('grey')
ax.set_title("Feature importances using MDI")
ax.set_ylabel("Mean decrease in impurity")
fig.tight_layout()
#fig.autofmt_xdate()
```



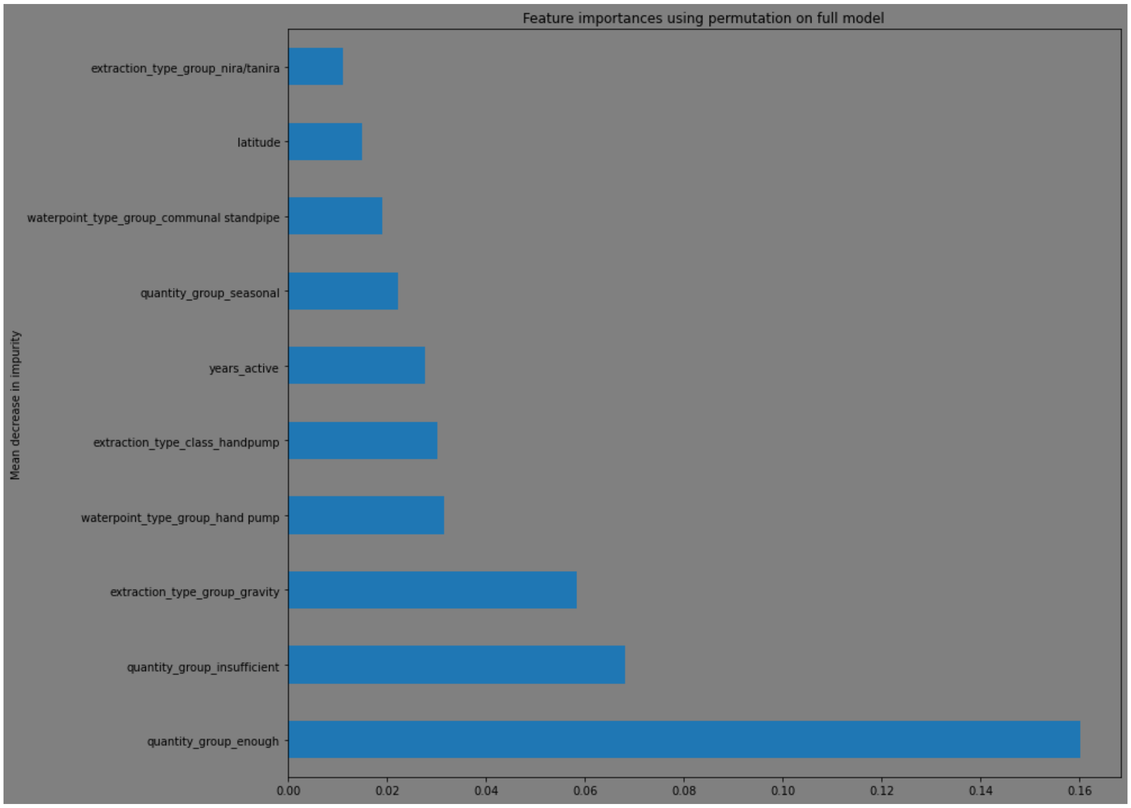
Here when we sort by the importance values, we see some interesting categories. Since we one hot encoded a lot of the categories, we will choose specific attributes of the column that are deemed important.

[47]:
```
#Feature permutation
from sklearn.inspection import permutation_importance

result = permutation_importance(clf, X_test_imp, y_test, n_repeats=5,␣
↪random_state=42, n_jobs=2, scoring='accuracy')
forest_importances = pd.Series(result.importances_mean, index=names)
```

[48]:
```
top_20_imp = forest_importances.sort_values(ascending=False)[:10]
```

[49]:
```
from matplotlib import rcParams

fig, ax = plt.subplots(figsize=(14, 10))
top_20_imp.plot.barh(ax=ax)

rcParams['font.weight'] = 'bold'
fig.set_facecolor('grey')
ax.set_facecolor('grey')
ax.set_title("Feature importances using permutation on full model")
```

```
ax.set_ylabel("Mean decrease in impurity")
#fig.autofmt_xdate()
fig.tight_layout()
```



Here we see that within the top 10 of mean difference in impurity and permutation feature importance, quantity_group_enough insufficient and seasonal, years_active, latitude, and extraction_type_group_gravity all appear. Thus, we will deem that these factors are important in deciding whether or not a well is functional.