Data Cleaning and EDA Prep Notebook

In [1]:
```python
# Import the required libraries
import pandas as pd
import numpy as np

# set up pandas to display floats in a more human friendly way
pd.options.display.float_format = '{:,.2f}'.format
```

Original training data files downloaded from data source were split between values and labels.

Step 1: Conduct data cleaning USING FUNCTIONS on the Training and Test Files

Step 2: Combine the Training values and labels into a single file for use in model training.

In [18]:
```python
# TRAINING DATA
train_values_raw = pd.read_csv('../data/original/TrainingSetValues/4910797b-ee55-40a7-8668-10efd5c

print(train_values_raw.shape)
train_values_raw.head(3)
```

(59400, 40)

Out[18]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_private	...	payment_type	water_
0	69572	6,000.00	2011-03-14	Roman	1390	Roman	34.94	-9.86	none	0	...	annually	
1	8776	0.00	2013-03-06	Grumeti	1399	GRUMETI	34.70	-2.15	Zahanati	0	...	never pay	
2	34310	25.00	2013-02-25	Lottery Club	686	World vision	37.46	-3.82	Kwa Mahundi	0	...	per bucket	

3 rows × 40 columns

In [19]:
```python
# TRAIN TARGET
train_targets = pd.read_csv('../data/original/TrainingSetLabels/0bf8bc6e-30d0-4c50-956a-603fc693d9

print(train_targets.shape)
train_targets.head(3)
```

(59400, 2)

Out[19]:

	id	status_group
0	69572	functional
1	8776	functional
2	34310	functional

In [20]:
```python
# VALIDATION DATA
test_values_raw = pd.read_csv('../data/original/TestSetValues/702ddfc5-68cd-4d1d-a0de-f5f566f76d91

print(test_values_raw.shape)
test_values_raw.head(3)
```

(14850, 40)

Out[20]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt_name	num_private	...	payment_type	wat
0	50785	0.00	2013-02-04	Dmdd	1996	DMDD	35.29	-4.06	Dinamu Secondary School	0	...	never pay	
1	51630	0.00	2013-02-04	Government Of Tanzania	1569	DWE	36.66	-3.31	Kimnyak	0	...	never pay	
2	17168	0.00	2013-02-01	NaN	1567	NaN	34.77	-5.00	Puma Secondary	0	...	never pay	

3 rows × 40 columns

Column and Row Information

- 40 columns/features in the raw data TRAINING
- 59,400 rows in the TRAINING data
    - TRAINING labels:
        - functional 32,259
        - non functional 22,824
        - functional needs repair 4,317

In [21]:
```
#train_values_raw.dtypes
```

In [22]:
```
'''

for var in train_values_raw.columns:
    # print the first 20 unique values in the cols
    unique_vals = train_values_raw[var].unique()
    print(var, unique_vals.size, unique_vals[0:20], '\n')
    '''
```

Out[22]:
```
"\n\nfor var in train_values_raw.columns:\n    # print the first 20 unique values in the cols\n    uni
in_values_raw[var].unique()\n    print(var, unique_vals.size, unique_vals[0:20], '\n')\n    "
```

Original Data Column Descriptions

- id - Numeric identifer for the waterpoint
- amount_tsh - Total static head (amount water available to waterpoint)
- date_recorded - The date the row was entered
- funder - Who funded the well
- gps_height - Altitude of the well
- installer - Organization that installed the well
- longitude - GPS coordinate
- latitude - GPS coordinate
- wpt_name - Name of the waterpoint if there is one
- num_private -
- basin - Geographic water basin
- subvillage - Geographic location
- region - Geographic location, NOTE: Hierarchy is Region > LGA > Ward
- region_code - Geographic location (coded)
- district_code - Geographic location (coded)
- lga - Geographic location
- ward - Geographic location
- population - Population around the well
- public_meeting - True/False
- recorded_by - Group entering this row of data
- scheme_management - Who operates the waterpoint
- scheme_name - Who operates the waterpoint
- permit - True/False, If the waterpoint is permitted
- construction_year - Year the waterpoint was constructed
- extraction_type - The kind of extraction the waterpoint uses
- extraction_type_group - The kind of extraction the waterpoint uses
- extraction_type_class - The kind of extraction the waterpoint uses
- management - How the waterpoint is managed
- management_group - How the waterpoint is managed
- payment - What the water costs
- payment_type - What the water costs
- water_quality - The quality of the water
- quality_group - The quality of the water
- quantity - The quantity of water
- quantity_group - The quantity of water
- source - The source of the water
- source_type - The source of the water
- source_class - The source of the water
- waterpoint_type - The kind of waterpoint
- waterpoint_type_group - The kind of waterpoint

Data cleaning steps:

- **Duplicate check**
- **Address Null Values**
- **Address Zeros in Numeric Values**
- **String type normalization**
- **Note:** No Data type conversions needed

Duplicate Check

Spoiler Alert: There are no exact duplicate rows nor duplicate identifers in the Training or Test dataset

In [23]:

```
# Functions for Duplicate checks
def has_exact_dups(df):
    dups = df[df.duplicated()]
    return len(dups) > 0

def has_identifier_dups(df, col_name='id'):
    num_rows = df.shape[0]
    num_ids = len(df[col_name].unique())
    return num_ids != num_rows
```

In [24]:

```
# Dup checking
dup1 = has_exact_dups(train_values_raw)
dup2 = has_identifier_dups(train_values_raw)
dup3 = has_exact_dups(test_values_raw)
dup4 = has_identifier_dups(test_values_raw)
print(dup1, dup2, dup3, dup4)
```

```
False False False False
```

In [25]:

```
# Make a deep copy before any data cleaning (Deep copy has own copy of data and index)
train_values_processed = train_values_raw.copy(deep=True)
test_values_processed = test_values_raw.copy(deep=True)
```

Handling Null Values

TRAINING and TEST/VALIDATION columns with Null values:

- funder : set to 'unknown' when null
- installer : set to 'unknown' when null
- subvillage : set to 'unknown' when null
- public_meeting : set to True when null
    - Training dataset: only 5055 out of 59,400 were False. 51,011 out of 59,4000 were True.
    - Test dataset: only 1291 out of 14,850 were False, 12,738 out of 14,850 were True.
    - As the vast majority of both Training and Test waterpoints have public_meeting of True, use True to replace all nulls.
- scheme_management : set to 'unknown' when null
- scheme_name : set to 'unknown' when null
- permit : set to True when nulls
    - Training dataset: 38,852 out of 59,4000 were True. 17,492 out of 59,4000 where False.
    - Test dataset: 9754 out of 14,850 were True. 4359 out of 14,850 were False
    - As the majority of both Training and Test waterpoints have permit populated as True, use True to replace all nulls.

In [26]:

```
# Null handler functions
def handle_nulls_inplace(df, cols_to_fill):
    for item in cols_to_fill:
        for key, value in item.items():
            df[key].fillna(value, inplace=True)
```

See what percentage of data is missing

In [27]:
```
train_values_processed.isnull().mean()
```

Out[27]:
```
id                       0.00
amount_tsh               0.00
date_recorded            0.00
funder                   0.06
gps_height               0.00
installer                0.06
longitude                0.00
latitude                 0.00
wpt_name                 0.00
num_private              0.00
basin                    0.00
subvillage               0.01
region                   0.00
region_code              0.00
district_code            0.00
lga                      0.00
ward                     0.00
population               0.00
public_meeting           0.06
recorded_by              0.00
scheme_management        0.07
scheme_name              0.47
permit                   0.05
construction_year        0.00
extraction_type          0.00
extraction_type_group    0.00
extraction_type_class    0.00
management               0.00
management_group         0.00
payment                  0.00
payment_type             0.00
water_quality            0.00
quality_group            0.00
quantity                 0.00
quantity_group           0.00
source                   0.00
source_type              0.00
source_class             0.00
waterpoint_type          0.00
waterpoint_type_group    0.00
dtype: float64
```

In [28]:
```
cols_to_fill = [{'funder': 'unknown'}, {'installer': 'unknown'}, {'subvillage': 'unknown'}, {'publ

handle_nulls_inplace(train_values_processed, cols_to_fill)
handle_nulls_inplace(test_values_processed, cols_to_fill)
```

Handing Zeros in Numeric Columns

- No change needed
  - latitude : No zeros
  - region_code : No zeros
- **Drop** data
  - num_private : ~98 of Train, **DROP this COLUMN** from Train and Test dataset.
- Replace Zeros
  - construction_year : ~35% of Train and ~35% of Test - Update zeros with the Average Construction Year.
- Do nothing. These 0 values seem in line with data used on the Offical Tanzanian Water Point Mapping System (WPMS) [http://wpm.maj (http://wpm.maji.go.tz/%5D). I don't have enough context to know what to replace the zero values with.
  - amount_tsh
  - gps_height
  - population

In [29]:
```
# Functions for handling Zeros in Numeric columns
def count_zeros(df, col_name):
    return df[df[col_name]==0][col_name].count()
```

In [30]:
```
numeric_col_names = ['amount_tsh', 'gps_height', 'latitude', 'longitude', 'num_private', 'region_c

for col_name in numeric_col_names:
    the_train_count = count_zeros(train_values_processed, col_name)
    the_test_count = count_zeros(test_values_processed, col_name)
    if(the_test_count + the_train_count > 0):
        print('TRAIN:', col_name, the_train_count)
        print('TEST:', col_name, the_test_count)
```

```
TRAIN: amount_tsh 41639
TEST: amount_tsh 10410
TRAIN: gps_height 20438
TEST: gps_height 5211
TRAIN: longitude 1812
TEST: longitude 457
TRAIN: num_private 58643
TEST: num_private 14656
TRAIN: district_code 23
TEST: district_code 4
TRAIN: population 21381
TEST: population 5453
TRAIN: construction_year 20709
TEST: construction_year 5260
```

In [31]:
```
# Drop the num_private COLUMN
train_values_processed.drop('num_private', axis=1, inplace=True)
test_values_processed.drop('num_private', axis=1, inplace=True)
```

In [32]:
```
# Drop the rows with 0 longitude from TRAIN
train_indices_long = train_values_processed[train_values_processed['longitude'] == 0 ].index
train_values_processed.drop(train_indices_long, inplace=True)

# drop the rows with 0 district_code from TRAIN
train_indices_distric_code = train_values_processed[train_values_processed['district_code'] == 0 ]
train_values_processed.drop(train_indices_distric_code, inplace=True)
```

In [33]:
```
# Get the average construction year for TRAIN and TEST/VALIDATION
known_const_year_rows = train_values_processed[train_values_processed['construction_year']>0]
avg_counstruction_year = int(known_const_year_rows['construction_year'].mean().round())
print(avg_counstruction_year)

test_known_const_year_rows = test_values_processed[test_values_processed['construction_year']>0]
test_avg_counstruction_year = int(test_known_const_year_rows['construction_year'].mean().round())
print(test_avg_counstruction_year)
```

```
1997
1997
```

In [34]:
```
# Set construction_year to the average construction year where that value is 0
train_values_processed['construction_year'] = train_values_processed.apply(lambda row: avg_constru
test_values_processed['construction_year'] = test_values_processed.apply(lambda row: test_avg_coun
```

Normalize String columns - all to lower case

- funder - Who funded the well
- installer - Organization that installed the well
- wpt_name - Name of the waterpoint if there is one
- basin - Geographic water basin
- subvillage - Geographic location
- region - Geographic location
- lga - Geographic location
- ward - Geographic location
- recorded_by - Group entering this row of data
- scheme_management - Who operates the waterpoint
- scheme_name - Who operates the waterpoint
- extraction_type - The kind of extraction the waterpoint uses
- extraction_type_group - The kind of extraction the waterpoint uses
- extraction_type_class - The kind of extraction the waterpoint uses
- management - How the waterpoint is managed
- management_group - How the waterpoint is managed
- payment_type - What the water costs
- payment_type - What the water costs
- water_quality - The quality of the water
- quality_group - The quality of the water
- quantity - The quantity of water
- quantity_group - The quantity of water
- source - The source of the water
- source_type - The source of the water
- source_class - The source of the water
- waterpoint_type - The kind of waterpoint
- waterpoint_type_group - The kind of waterpoin

In [36]:
```
# Normalize String values function(s)
def normalize_strings(df, col_name):
    df[col_name] = df.apply(lambda row: row[col_name].lower(), axis=1)
```

In [37]:
```
string_col_names = ['funder', 'installer', 'wpt_name', 'basin', 'subvillage', 'region', 'lga', 'wa

for col_name in string_col_names:
    normalize_strings(train_values_processed, col_name)
    normalize_strings(test_values_processed, col_name)
```

New Columns/Feature creation for EDA - TRAINING dataset ONLY

- recorded_year - Pulling out the year from date_recorded
- waterpoint_age - Calculate as recorded_year - construction_year
- recorded_good_quality - True if quality_group == 'good', False if anything other than 'good'
- recorded_good_quantity - True if quanity_group == 'sufficient', False if anythign other than 'sufficient'

In [39]:
```
# Functions for creating new features
def get_recorded_year(recorded_date_string):
    year = 0
    date_segs = recorded_date_string.split('-')
    if((len(date_segs) == 3) & (len(date_segs[0]) == 4)):
        try:
            year = int(date_segs[0])
        except:
            print("Not a valid year format.")
    return year


def get_waterpoint_age(recorded_year, constructed_year):
    age = 0
    is_logical_year = constructed_year > 0
    is_logical_age = recorded_year > constructed_year
    if (is_logical_year & is_logical_age):
        age = recorded_year - constructed_year
    return age


def get_recorded_good_quality(quality_group):
    result = False
    if ('good' == quality_group):
        result = True
    return result


def get_recorded_good_quanity(quanity_group):
    result = False
    if ('enough' == quanity_group):
        result = True
    return result
```

In [40]:
```
# recorded_year
train_values_processed['recorded_year'] = train_values_processed.apply(lambda row: get_recorded_ye
```

In [41]:
```
# waterpoint_age
train_values_processed['waterpoint_age'] = train_values_processed.apply(lambda row: get_waterpoint_
```

In [42]:
```
# recorded_good_quality
train_values_processed['recorded_good_quality'] = train_values_processed.apply(lambda row: get_rec
```

In [43]:
```
# recorded_good_quanity
train_values_processed['recorded_good_quantity'] = train_values_processed.apply(lambda row: get_re
```

Final Prep

- Add the class labels to the TRAINING dataset
- Save both cleaned TRAINING and TEST to file for use in EDA and Classifer Modeling

In [44]:
```
train_values_processed_and_labeled = pd.merge(train_values_processed, train_targets, on='id')
```

In [45]:
```
train_values_processed_and_labeled.to_csv('../data/train_processed_labeled.csv', index=False)
test_values_processed.to_csv('../data/test_processed.csv', index=False)
```

In [46]:
```
train_values_processed_and_labeled.shape
```

Out[46]: (57565, 44)

In [47]:
```
test_values_processed.shape
```

Out[47]: (14850, 39)

In [ ]:
```

```