Introduction

Final Project Submission

- Student Name: Adam Marianacci
- Student Pace: Flex
- Scheduled project review date/time: TBD
- Instructor Name: Mark Barbour

Business Understanding

It is my job to help the WWFA (Water Wells For Africa) locate wells that need to be repaired in Tanzania.

Data Understanding

Data Preperation

In [1]:

```
# Importing the necessary libraries
import pandas as pd
from datetime import datetime
import numpy as np
import seaborn as sns
import statsmodels as sm
import sklearn
import sklearn.preprocessing as preprocessing
import matplotlib.pyplot as plt
from scipy import stats
from sklearn import linear_model
from sklearn.linear_model import LogisticRegression
from sklearn.feature_selection import RFE
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
import warnings
warnings.filterwarnings('ignore')
```

In [2]:

```
# Set display options to show all rows and columns
pd.set_option('display.max_rows', None)
pd.set_option('display.max_columns', None)
```

In [3]:

```
# Importing the dataframes
df_x = pd.read_csv('data/training_set_values.csv')
df_y = pd.read_csv('data/training_set_labels.csv')
```

In [4]:

```
# Combining the 2 dataframes into 1 new dataframe
Waterwells_df = pd.concat([df_y, df_x], axis=1)
```

In [5]:
```python
# Previewing the dataframe
Waterwells_df.head()
```

Out[5]:

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

In [6]:
```python
#dropping the missing values from the 'construction_year' column and creating a new df
Construction_Year_df = Waterwells_df[Waterwells_df['construction_year'] != 0]

# Calculate the current year
current_year = datetime.now().year

# Create a new column 'age' by subtracting construction year from the current year
Construction_Year_df['age'] = current_year - Waterwells_df['construction_year']
```

In [7]:
```
Construction_Year_df['construction_year'].value_counts()
```

Out[7]:
```
2010    2645
2008    2613
2009    2533
2000    2091
2007    1587
2006    1471
2003    1286
2011    1256
2004    1123
2012    1084
2002    1075
1978    1037
1995    1014
2005    1011
1999     979
1998     966
1990     954
1985     945
1996     811
1980     811
1984     779
1982     744
1994     738
1972     708
1974     676
1997     644
1992     640
1993     608
2001     540
1988     521
1983     488
1975     437
1986     434
1976     414
1970     411
1991     324
1989     316
1987     302
1981     238
```

```
1977     202
1979     192
1973     184
2013     176
1971     145
1960     102
1967      88
1963      85
1968      77
1969      59
1964      40
1962      30
1961      21
1965      19
1966      17
Name: construction_year, dtype: int64
```

Dropping columns that are not directly related to the business problem and also have high cardinality, making them difficult to one hot encode.

In [8]:

```python
# Dropping irrelevant columns from the dataframe, also columns with large amounts of missing da
columns_to_drop = [
    'id', 'scheme_management', 'region', 'payment', 'public_meeting', 'district_code', 'populat
    'num_private', 'basin', 'construction_year',
    'waterpoint_type_group', 'source_class', 'payment_type', 'management_group', 'recorded_by',
    'extraction_type', 'management',
    'source_type', 'extraction_type_group', 'permit', 'funder',
    'date_recorded', 'installer', 'ward', 'scheme_name', 'wpt_name', 'lga', 'subvillage'
]

Waterwells_df = Waterwells_df.drop(columns_to_drop, axis=1, errors='ignore')

```

In [9]:

```
# Examining the dimensions of the dataframe
Waterwells_df.head()
```

Out[9]:

	status_group	amount_tsh	gps_height	longitude	latitude	region_code	extraction_type_class	water_quality	quality_group	qua
0	functional	6000.0	1390	34.938093	-9.856322	11	gravity	soft	good	er
1	functional	0.0	1399	34.698766	-2.147466	20	gravity	soft	good	insuff
2	functional	25.0	686	37.460664	-3.821329	21	gravity	soft	good	er
3	non functional	0.0	263	38.486161	-11.155298	90	submersible	soft	good	
4	functional	0.0	0	31.130847	-1.825359	18	gravity	soft	good	sea

In [10]:

```
# Checking for missing values and learning about the datatypes of the columns
Waterwells_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 59400 entries, 0 to 59399
Data columns (total 13 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   status_group           59400 non-null  object 
 1   amount_tsh             59400 non-null  float64
 2   gps_height             59400 non-null  int64  
 3   longitude              59400 non-null  float64
 4   latitude               59400 non-null  float64
 5   region_code            59400 non-null  int64  
 6   extraction_type_class  59400 non-null  object 
 7   water_quality          59400 non-null  object 
 8   quality_group          59400 non-null  object 
 9   quantity               59400 non-null  object 
 10  quantity_group         59400 non-null  object 
 11  source                 59400 non-null  object 
 12  waterpoint_type        59400 non-null  object 
dtypes: float64(3), int64(2), object(8)
memory usage: 5.9+ MB
```

In [11]:

```python
# Create a new column 'needs_repair' by merging the two categories
Waterwells_df['needs_repair'] = Waterwells_df['status_group'].replace(
    {'functional': 0, 'non functional': 1,
     'functional but needs repair': 1})

# Drop the original 'status_group' column
Waterwells_df.drop('status_group', axis=1, inplace=True)

#Display the updated DataFrame
Waterwells_df.head()


```

Out[11]:

	amount_tsh	gps_height	longitude	latitude	region_code	extraction_type_class	water_quality	quality_group	quantity	quantity
0	6000.0	1390	34.938093	-9.856322	11	gravity	soft	good	enough	
1	0.0	1399	34.698766	-2.147466	20	gravity	soft	good	insufficient	ins
2	25.0	686	37.460664	-3.821329	21	gravity	soft	good	enough	
3	0.0	263	38.486161	-11.155298	90	submersible	soft	good	dry	
4	0.0	0	31.130847	-1.825359	18	gravity	soft	good	seasonal	s

In [12]:

```python
# Defining X and y variables
y = Waterwells_df["needs_repair"]
X = Waterwells_df.drop("needs_repair", axis=1)
```

In [13]:

```python
# Performing a train, test, split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state=42)
```

In [14]:

```
# Looking at the number of missing values in each column
X_train.isna().sum()
```

Out[14]:

```
amount_tsh               0
gps_height               0
longitude                0
latitude                 0
region_code              0
extraction_type_class    0
water_quality            0
quality_group            0
quantity                 0
quantity_group           0
source                   0
waterpoint_type          0
dtype: int64
```

In [15]:

```
#Defining categorical df
X_train_categorical = X_train.select_dtypes(include='object').copy()
X_train_categorical.head()

```

Out[15]:

	extraction_type_class	water_quality	quality_group	quantity	quantity_group	source	waterpoint_type
3607	gravity	soft	good	insufficient	insufficient	spring	communal standpipe
50870	handpump	soft	good	enough	enough	shallow well	hand pump
20413	other	soft	good	enough	enough	shallow well	other
52806	gravity	soft	good	insufficient	insufficient	river	communal standpipe
50091	other	salty	salty	enough	enough	shallow well	other

In [16]:

```
#Inspecting the unique values of source
X_train_categorical['quality_group'].unique()
```

Out[16]:

```
array(['good', 'salty', 'unknown', 'colored', 'fluoride', 'milky'],
      dtype=object)
```

In [17]:
```
X_train_categorical['quality_group'].value_counts()
```

Out[17]:
```
good        40633
salty        4173
unknown      1490
milky         650
colored       395
fluoride      179
Name: quality_group, dtype: int64
```

In [18]:
```
# Removing 'other' and 'unknown' from the source column
X_train_categorical = X_train_categorical[~X_train_categorical['source'].isin(['other', 'unknown

# Display the updated counts
print(X_train_categorical['source'].value_counts())

```

```
shallow well            13540
spring                  13537
machine dbh              8849
river                    7719
rainwater harvesting     1829
hand dtw                  701
lake                      606
dam                       505
Name: source, dtype: int64
```

cite this

In [19]:

```
ohe = OneHotEncoder(handle_unknown="ignore", sparse=False, drop='first').set_output(transform='p
# column_to_encode = X_train_categorical['source']
X_train_categorical = ohe.fit_transform(X_train_categorical.values.reshape(-1,1))
X_train_categorical.head()
```

Out[19]:

	x0_colored	x0_coloured	x0_communal standpipe	x0_communal standpipe multiple	x0_dam	x0_dry	x0_enough	x0_fluoride	x0_fluoride abandoned	x0_good	x0_gravity
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

In [20]:

```
type(X_train_categorical)
```

Out[20]:

```
pandas.core.frame.DataFrame
```

In [21]:
```
X_train_categorical.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 331002 entries, 0 to 331001
Data columns (total 33 columns):
 #   Column                         Non-Null Count   Dtype  
---  ------                         --------------   -----  
 0   x0_colored                     331002 non-null  float64
 1   x0_coloured                    331002 non-null  float64
 2   x0_communal standpipe          331002 non-null  float64
 3   x0_communal standpipe multiple 331002 non-null  float64
 4   x0_dam                         331002 non-null  float64
 5   x0_dry                         331002 non-null  float64
 6   x0_enough                      331002 non-null  float64
 7   x0_fluoride                    331002 non-null  float64
 8   x0_fluoride abandoned          331002 non-null  float64
 9   x0_good                        331002 non-null  float64
 10  x0_gravity                     331002 non-null  float64
 11  x0_hand dtw                    331002 non-null  float64
 12  x0_hand pump                   331002 non-null  float64
 13  x0_handpump                    331002 non-null  float64
 14  x0_improved spring             331002 non-null  float64
 15  x0_insufficient                331002 non-null  float64
 16  x0_lake                        331002 non-null  float64
 17  x0_machine dbh                 331002 non-null  float64
 18  x0_milky                       331002 non-null  float64
 19  x0_motorpump                   331002 non-null  float64
 20  x0_other                       331002 non-null  float64
 21  x0_rainwater harvesting        331002 non-null  float64
 22  x0_river                       331002 non-null  float64
 23  x0_rope pump                   331002 non-null  float64
 24  x0_salty                       331002 non-null  float64
 25  x0_salty abandoned             331002 non-null  float64
 26  x0_seasonal                    331002 non-null  float64
 27  x0_shallow well                331002 non-null  float64
 28  x0_soft                        331002 non-null  float64
 29  x0_spring                      331002 non-null  float64
 30  x0_submersible                 331002 non-null  float64
 31  x0_unknown                     331002 non-null  float64
 32  x0_wind-powered                331002 non-null  float64
dtypes: float64(33)
```

```
memory usage: 83.3 MB
```

In [23]:

```python
#Defining numerical df
X_train_numerical = X_train[['amount_tsh', 'gps_height']].copy()
X_train_numerical.head()
```

Out[23]:

	amount_tsh	gps_height
3607	50.0	2092
50870	0.0	0
20413	0.0	0
52806	0.0	0
50091	300.0	1023

In [24]:

```python

# Initialize MinMaxScaler
scaler = MinMaxScaler()

# Scale the selected column
X_train_numerical = scaler.fit_transform(X_train_numerical)

column_names = ['amount_tsh']
# Converting X_train_numerical back into a df
X_train_numerical = pd.DataFrame(X_train_numerical, columns=column_names)

# Display the array
X_train_numerical
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-24-a113bd572cb9> in <module>
      6
      7 # Converting X_train_numerical back into a df
----> 8 X_train_numerical = pd.DataFrame(X_train_numerical, columns=column_names)
      9
     10 # Display the array

NameError: name 'column_names' is not defined
```

In [ ]:
```
X_train_numerical.shape
```

In [ ]:
```
# X_train_numerical is a NumPy array



# Convert the NumPy array to a pandas DataFrame with specified column names
X_train_numerical = pd.DataFrame(X_train_numerical, columns=column_names)

# Display the DataFrame


print(X_train_numerical)
```

In [ ]:
```
X_train_full = pd.concat([X_train_numerical, X_train_categorical], axis=1)
X_train_full.head()
```

In [ ]:
```
X_train_full.info()
```

In [ ]:
```
missing_values = X_train_full.isnull().sum()
print(missing_values)
```

In [ ]:
```
X_train_full.info()
```

In [ ]:
```
# Creating a heatmap from the initial dataframe
fig, ax = plt.subplots(figsize=(10,10))
cor = Waterwells_df.corr()
sns.heatmap(cor,cmap="Blues",annot=True)
```

I wanted to create a function so I could easily evaluate each of models with an r2 score, root mean squared error, and mean absolute error.

In [ ]:
```
def evaluate_model(y_test, y_pred, lr):
    # R-squared (R2)
    r2 = r2_score(y_test, y_pred)

    # Root Mean Squared Error (RMSE)
    rmse = mean_squared_error(y_test, y_pred, squared=False)

    # Mean Absolute Error (MAE)
    mae = mean_absolute_error(y_test, y_pred)

    # Intercept
    #intercept = lr.intercept_

    # Printing the results
    print("R2 score: ", r2)
    print("Root Mean Squared Error: ", rmse)
    print("Mean Absolute Error: ", mae)
    #print("Intercept: ", intercept)

    # Returning the results as a dictionary
    results_model = {
        'r2': r2,
        'rmse': rmse,
        'mae': mae,
        #'intercept': intercept
    }

    return results_model
```

Modeling

In [ ]:
```
logreg = LogisticRegression(fit_intercept=False, C=1e12, solver='liblinear')
model_log = logreg.fit(X_train_full, y_train)
```

Evaluation

Conclusion

Recommendations

Limitations

Next Steps

In [ ]:
```
 
```