TANZANIA WATER WELLS PROJECT.

Final Project Submission

Students Name: Dennis Mwenda Kimathi

Student pace: Full time Hybrid

Instructor name: Maryann Mwikali

INTRODUCTION

Tanzania, as a developing country, struggles with providing clean water to its population of over 57,000,000. There are many water points already established in the country, but some are in need of repair while others have failed altogether. According to the World Health Organization, access to safe and clean drinking water is essential for health, development, and well-being.

Business Understanding

The goal is to ensure that water wells in Tanzania are functional, providing reliable access to clean water for communities. Ensuring well functionality involves understanding various factors influencing the wells' status, such as management practices, payment methods, and water quality. Effective management and maintenance strategies can be developed by identifying key determinants of well functionality, leading to improved water access for the population.

Problem Statement

The primary problem is the high rate of non-functional water wells in Tanzania, which undermines efforts to provide safe and reliable water access. This project aims to identify the critical factors affecting well functionality, predict the status of wells using these factors, and recommend strategies to improve well management and maintenance.

DATA UNDERSTANDING

The main data used from the project is from two datasets which have been merged.

The first dataset contained wells infomation and the second on contains information of wells condition

Main data has 27813 rows × 27 columns

Our data gives information about the Tanzania wells and its features.

link: https://www.drivendata.org/competitions/7/pump-it-up-data-mining-the-water-table/data/

Columns

1. **id**: Unique identifier, .
2. **amount_tsh**: .
3. **date_recorded**: .
4. **funder**: funder of the well.
5. **gps_height**: heiht of the well .
6. **installer**: installer of the well.
7. **longitude, latitude**: geographical location of the well.
8. **wpt_name**: Name of the water point, .
9. **num_private**: .
10. **basin**: source of water.
11. **subvillage**: .
12. **region**: region the well is located.
13. **region_code**: Can be redundant with `region` .
14. **district_code**: .
15. **lga**: Local government area.
16. **ward**: .
17. **population**: population of the area.
18. **public_meeting**:
19. **recorded_by**:
20. **scheme_management**:management schemes.
21. **scheme_name**:.
22. **permit**: permit status .
23. **construction_year**:.
24. **extraction_type**: How the water is sourced.
25. **extraction_type_group**: Can be redundant with `extraction_type` .
26. **extraction_type_class**: Can be redundant with `extraction_type` .
27. **management**: who manages the well.
28. **management_group**: Can be redundant with `management` .
29. **payment**: Relevant payment types.
30. **payment_type**: Can be redundant with `payment` .
31. **water_quality**:
32. **quality_group**: Can be redundant with `water_quality` .
33. **quantity**: Relevant for quantity analysis.
34. **quantity_group**: Can be redundant with `quantity` .
35. **source**: source of the well water.
36. **source_type**: Can be redundant with `source` .
37. **source_class**: Can be redundant with `source` .
38. **waterpoint_type**:
39. **waterpoint_type_group**: Can be redundant with `waterpoint_type` .
40. **status_group**: The target variable, important for analysis.

DATA PREPARATION

In [53]:
```
# Importing necessary libraries for data analysis and visualization

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt  # for data visualization.
%matplotlib inline
import seaborn as sns  # for enhanced data visualization.
from pandas.api.types import is_numeric_dtype  # Used to check if a data type
```

Reading the Datasets

In [54]:
```
df1 = pd.read_csv('files/Tz water wells1.csv')
df2 = pd.read_csv('files/wells.cond.csv')
```

In [55]:
```
df1.head()
```

Out[55]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude	wpt
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322	
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466	Z
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329	M
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298	Z Nar
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359	

5 rows × 40 columns

In [56]:
```
df2.head()
```

Out[56]:

	id	status_group
0	69572	functional
1	8776	functional
2	34310	functional
3	67743	non functional
4	19728	functional

In [57]:
```
df1.describe()
```

Out[57]:

	id	amount_tsh	gps_height	longitude	latitude	num_private	r
count	59400.000000	59400.000000	59400.000000	59400.000000	5.940000e+04	59400.000000	59
mean	37115.131768	317.650385	668.297239	34.077427	-5.706033e+00	0.474141	
std	21453.128371	2997.574558	693.116350	6.567432	2.946019e+00	12.236230	
min	0.000000	0.000000	-90.000000	0.000000	-1.164944e+01	0.000000	
25%	18519.750000	0.000000	0.000000	33.090347	-8.540621e+00	0.000000	
50%	37061.500000	0.000000	369.000000	34.908743	-5.021597e+00	0.000000	
75%	55656.500000	20.000000	1319.250000	37.178387	-3.326156e+00	0.000000	
max	74247.000000	350000.000000	2770.000000	40.345193	-2.000000e-08	1776.000000	

In [58]:
```
# Merging target data frame with our main dataframe
data = pd.merge(df1,df2, on = 'id', how = 'inner')
data
```

Out[58]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latitude
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856322
1	8776	0.0	2013-03-06	Grumeti	1399	GRUMETI	34.698766	-2.147466
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821329
3	67743	0.0	2013-01-28	Unicef	263	UNICEF	38.486161	-11.155298
4	19728	0.0	2011-07-13	Action In A	0	Artisan	31.130847	-1.825359
...	...	...	...	...	...	...	...	...
59395	60739	10.0	2013-05-03	Germany Republi	1210	CES	37.169807	-3.253847
59396	27263	4700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.249991	-9.070629
59397	37057	0.0	2011-04-11	NaN	0	NaN	34.017087	-8.750434
59398	31282	0.0	2011-03-08	Malec	0	Musa	35.861315	-6.378573
59399	26348	0.0	2011-03-23	World Bank	191	World	38.104048	-6.747464

59400 rows × 41 columns

In [59]:
```
def inspect_data(data):
    """
    Inspect a Pandas DataFrame and print its head, tail, description, and shape

    """
    print("Head:")
    print(data.head())
    print("\nTail:")
    print(data.tail())
    print("\nDescription:")
    print(data.describe())
    print("\nShape:")
    print(data.shape)

inspect_data(data)
```

```
Head:
      id  amount_tsh date_recorded        funder  gps_height     installer
\
0  69572      6000.0    2011-03-14         Roman        1390         Roman
1   8776         0.0    2013-03-06       Grumeti        1399       GRUMETI
2  34310        25.0    2013-02-25  Lottery Club         686  World vision
3  67743         0.0    2013-01-28        Unicef         263        UNICEF
4  19728         0.0    2011-07-13   Action In A           0       Artisan

   longitude   latitude              wpt_name  num_private  ... water_quali
ty  \
0  34.938093  -9.856322                  none            0  ...          so
ft
1  34.698766  -2.147466              Zahanati            0  ...          so
ft
2  37.460664  -3.821329           Kwa Mahundi            0  ...          so
ft
3  38.486161 -11.155298  Zahanati Ya Nanyumbu            0  ...          so
ft
```

In [60]:
```
# creating a function to check for duplicates.
def has_duplicates(df):
    num_duplicates = df.duplicated().sum()
    if num_duplicates == 0:
        print("There are no duplicate rows in the DataFrame.")
    else:
        print(f"There are {num_duplicates} duplicate rows in the DataFrame.")

has_duplicates(data)
```

```
There are no duplicate rows in the DataFrame.
```

In [61]:
```
# Creating function to check counts of missing values
def has_missing_values(df):
    missing_values = df.isnull().sum()
    num_missing_values = missing_values[missing_values > 0].count()
    if num_missing_values == 0:
        print("There are no missing values in the DataFrame.")
    else:
        print(f"There are {num_missing_values} columns with missing values.")
        print(missing_values[missing_values > 0])

has_missing_values(data)
```

```
There are 7 columns with missing values.
funder                3635
installer             3655
subvillage             371
public_meeting        3334
scheme_management     3877
scheme_name          28166
permit                3056
dtype: int64
```

In [62]:
```
# drop all rows with missing values
data.dropna(inplace = True )
data
```

Out[62]:

	id	amount_tsh	date_recorded	funder	gps_height	installer	longitude	latit
0	69572	6000.0	2011-03-14	Roman	1390	Roman	34.938093	-9.856
2	34310	25.0	2013-02-25	Lottery Club	686	World vision	37.460664	-3.821
5	9944	20.0	2011-03-13	Mkinga Distric Coun	0	DWE	39.172796	-4.765
13	50495	0.0	2013-03-15	Lawatefuka Water Supply	1368	Lawatefuka water sup	37.092574	-3.181
14	53752	0.0	2012-10-20	Biore	0	WEDECO	34.364073	-3.629
...	...	...	...	...	...	...	...	
59381	67885	0.0	2011-03-16	Mkinga Distric Coun	0	DWE	38.835001	-4.880
59382	47002	6.0	2013-08-03	Ces(gmbh)	1383	DWE	37.454759	-3.323
59391	44885	0.0	2013-08-03	Government Of Tanzania	540	Government	38.044070	-4.272
59395	60739	10.0	2013-05-03	Germany Republi	1210	CES	37.169807	-3.253
59396	27263	4700.0	2011-05-07	Cefa-njombe	1212	Cefa	35.249991	-9.070

27813 rows × 41 columns

In [63]:
```
# check column types
data.dtypes
```

Out[63]:
```
id                         int64
amount_tsh               float64
date_recorded             object
funder                    object
gps_height                 int64
installer                 object
longitude                float64
latitude                 float64
wpt_name                  object
num_private                int64
basin                     object
subvillage                object
region                    object
region_code                int64
district_code              int64
lga                       object
ward                      object
population                 int64
public_meeting            object
recorded_by               object
scheme_management         object
scheme_name               object
permit                    object
construction_year          int64
extraction_type           object
extraction_type_group     object
extraction_type_class     object
management                object
management_group          object
payment                   object
payment_type              object
water_quality             object
quality_group             object
quantity                  object
quantity_group            object
source                    object
source_type               object
source_class              object
waterpoint_type           object
waterpoint_type_group     object
status_group              object
dtype: object
```

In [64]:
```
# 1. Data Type Conversion
data['date_recorded'] = pd.to_datetime(data['date_recorded'])

# 2. Outliers Detection and Handling
for column in ['amount_tsh', 'gps_height', 'longitude', 'latitude']:
    q1 = data[column].quantile(0.25)
    q3 = data[column].quantile(0.75)
    iqr = q3 - q1
    lower_bound = q1 - 1.5 * iqr
    upper_bound = q3 + 1.5 * iqr
    data[column] = np.where(data[column] < lower_bound, q1, data[column])
    data[column] = np.where(data[column] > upper_bound, q3, data[column])

# 3. Data Consistency
categorical_columns = ['funder', 'installer', 'basin', 'region', 'lga', 'ward'
for column in categorical_columns:
    data[column] = data[column].str.lower()

# 4. Feature Engineering
data['recorded_year'] = data['date_recorded'].dt.year
data['recorded_month'] = data['date_recorded'].dt.month

# 5. Dropping Irrelevant or Redundant Columns
columns_to_drop = ['id', 'wpt_name', 'num_private', 'subvillage', 'region_code
data.drop(columns=columns_to_drop, inplace=True)

# 6. Normalization/Standardization
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
data[['amount_tsh', 'gps_height', 'population']] = scaler.fit_transform(data[[

# Display the cleaned data
data.info(), data.head()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 27813 entries, 0 to 59396
Data columns (total 27 columns):
 #   Column             Non-Null Count  Dtype         
---  ------             --------------  -----         
 0   amount_tsh         27813 non-null  float64       
 1   date_recorded      27813 non-null  datetime64[ns]
 2   funder             27813 non-null  object        
 3   gps_height         27813 non-null  float64       
 4   installer          27813 non-null  object        
 5   longitude          27813 non-null  float64       
 6   latitude           27813 non-null  float64       
 7   basin              27813 non-null  object        
 8   region             27813 non-null  object        
 9   district_code      27813 non-null  int64         
 10  lga                27813 non-null  object        
 11  ward               27813 non-null  object        
 12  population         27813 non-null  float64       
 13  public_meeting     27813 non-null  object        
 14  scheme_management  27813 non-null  object        
 15  permit             27813 non-null  object        
 16  construction_year  27813 non-null  int64         
 17  extraction_type    27813 non-null  object        
 18  management         27813 non-null  object        
 19  payment            27813 non-null  object        
 20  water_quality      27813 non-null  object        
 21  quantity           27813 non-null  object        
 22  source             27813 non-null  object        
 23  waterpoint_type    27813 non-null  object        
 24  status_group       27813 non-null  object        
 25  recorded_year      27813 non-null  int64         
 26  recorded_month     27813 non-null  int64         
dtypes: datetime64[ns](1), float64(5), int64(4), object(17)
memory usage: 5.9+ MB
```

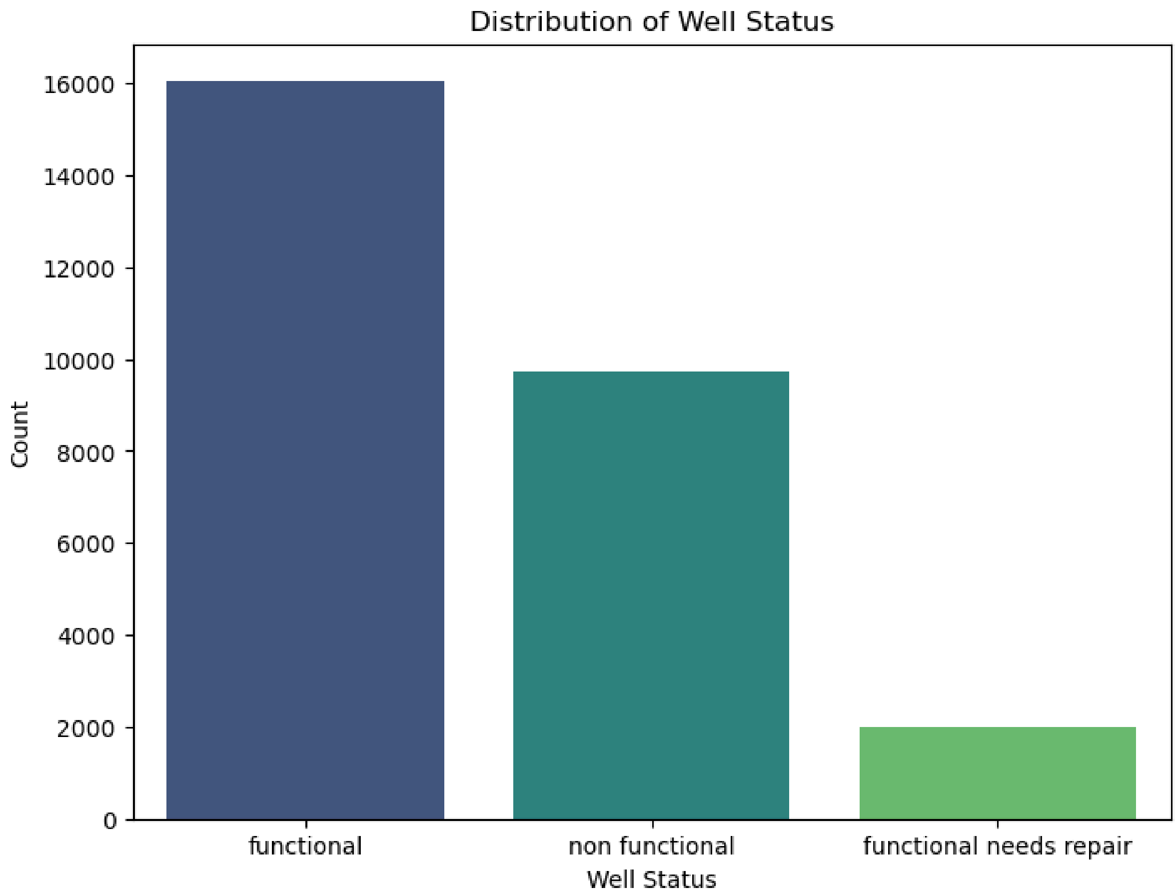
Out[64]:
```
(None,
    amount_tsh date_recorded                    funder  gps_height  \
0     0.500000    2011-03-14                     roman    0.544518
2     0.083333    2013-02-25              lottery club    0.285504
5     0.066667    2011-03-13        mkinga distric coun    0.033113
13    0.000000    2013-03-15  lawatefuka water supply    0.536424
14    0.000000    2012-10-20                     biore    0.033113

              installer  longitude  latitude       basin       region  \
0                 roman  34.938093 -9.856322  lake nyasa       iringa
2          world vision  37.460664 -3.821329     pangani      manyara
5                   dwe  39.172796 -4.765587     pangani        tanga
13  lawatefuka water sup  37.092574 -3.181783     pangani  kilimanjaro
14               wedeco  34.364073 -3.629333    internal    shinyanga

    district_code  ... extraction_type   management         payment  \
0               5  ...         gravity          vwc    pay annually
2               4  ...         gravity          vwc  pay per bucket
5               8  ...     submersible          vwc  pay per bucket
13              7  ...         gravity  water board     pay monthly
14              6  ...     nira/tanira          wug       never pay

   water_quality quantity        source              waterpoint_type  \
0           soft   enough        spring           communal standpipe
2           soft   enough           dam  communal standpipe multiple
5          salty   enough         other  communal standpipe multiple
13          soft   enough        spring           communal standpipe
14          soft   enough  shallow well                    hand pump

   status_group recorded_year recorded_month
0    functional          2011              3
2    functional          2013              2
5    functional          2011              3
13   functional          2013              3
14   functional          2012             10

[5 rows x 27 columns])
```

EXPLORATORY DATA ANALYSIS

a) Univariate Analysis

1. Distribution of Well Status (status_group)

In [65]:
```
# Distribution of Well Status
plt.figure(figsize=(8, 6))
sns.countplot(data=data, x='status_group', palette='viridis')
plt.title('Distribution of Well Status')
plt.xlabel('Well Status')
plt.ylabel('Count')
plt.show()
```

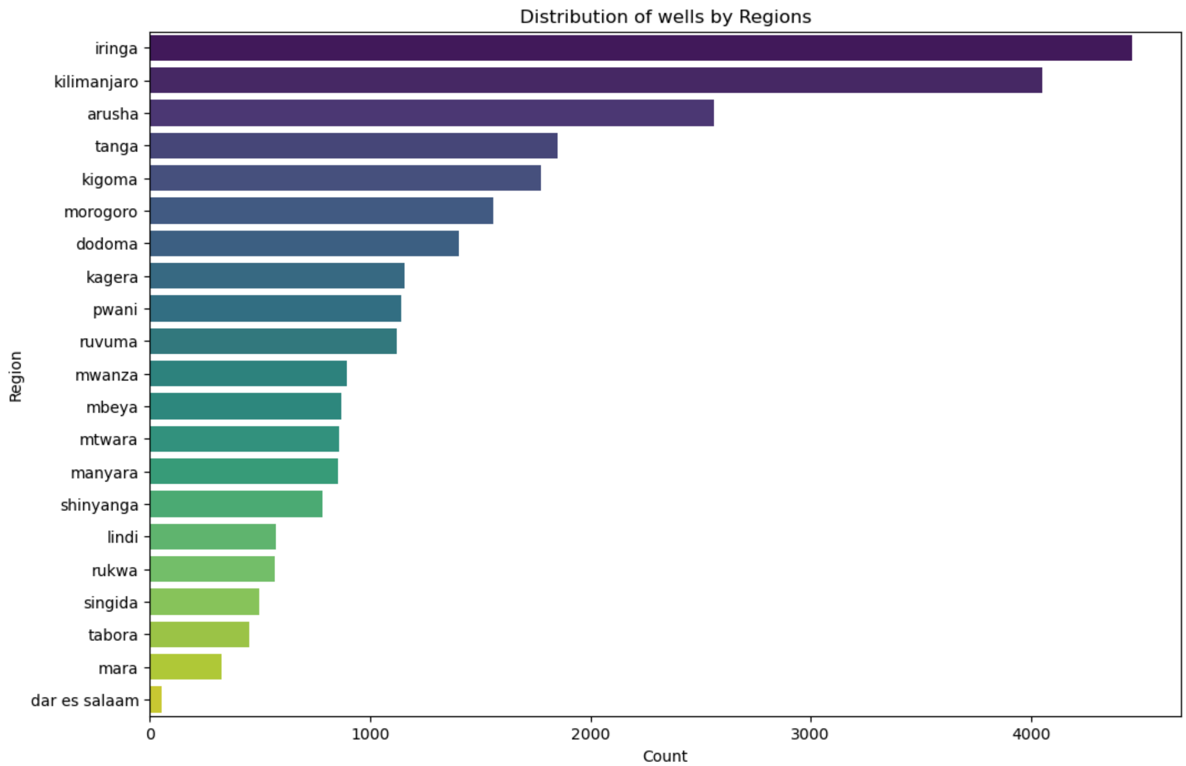


This plot shows the count of wells categorized by their status. It indicates the proportion of functional and non-functional wells.

Insight: There are more functional wells compared to non-functional ones.

2. Distribution of Regions (region)

In [66]:
```
# Distributions of regions
plt.figure(figsize=(12,8))
sns.countplot(data=data, y='region', palette = 'viridis', order=data['region']
plt.title('Distribution of wells by Regions')
plt.xlabel('Count')
plt.ylabel('Region')
plt.show()
```

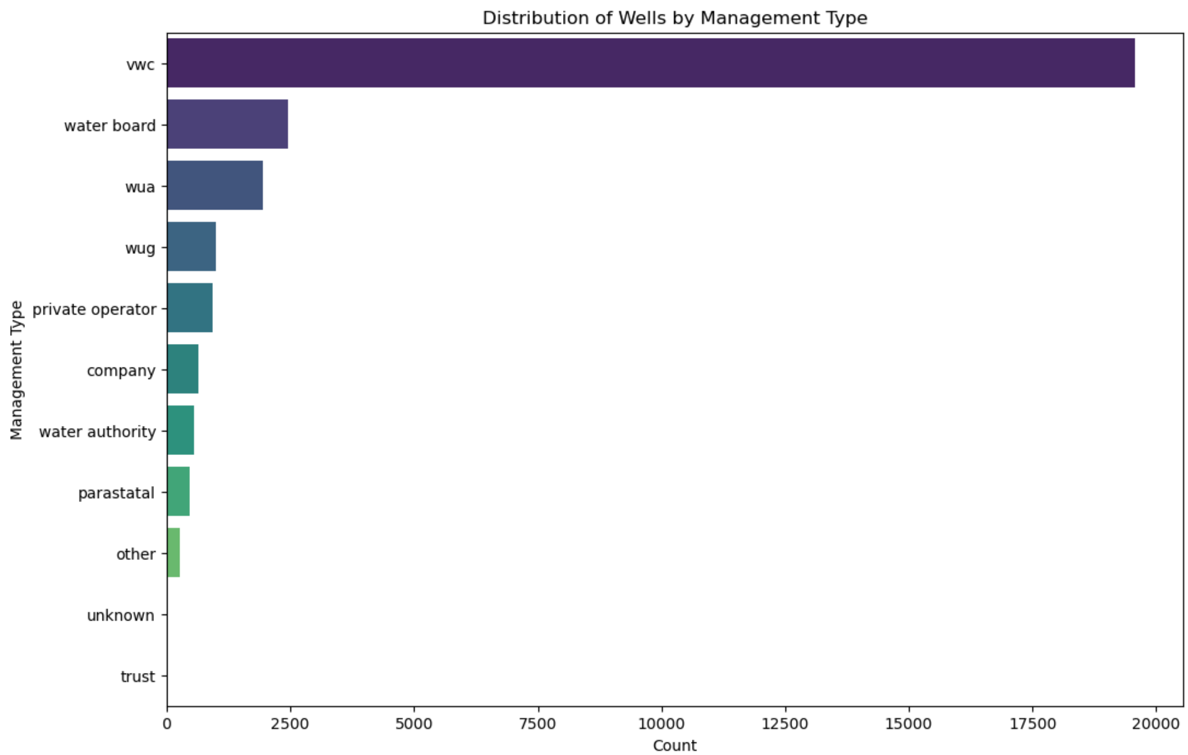


This plot displays the number of wells in each region. It helps understand the geographical distribution of wells.

Insight : The regions with the highest number of wells are shown, with Iringa, Manyara, and Tanga being prominent.

Distribution of Management Types

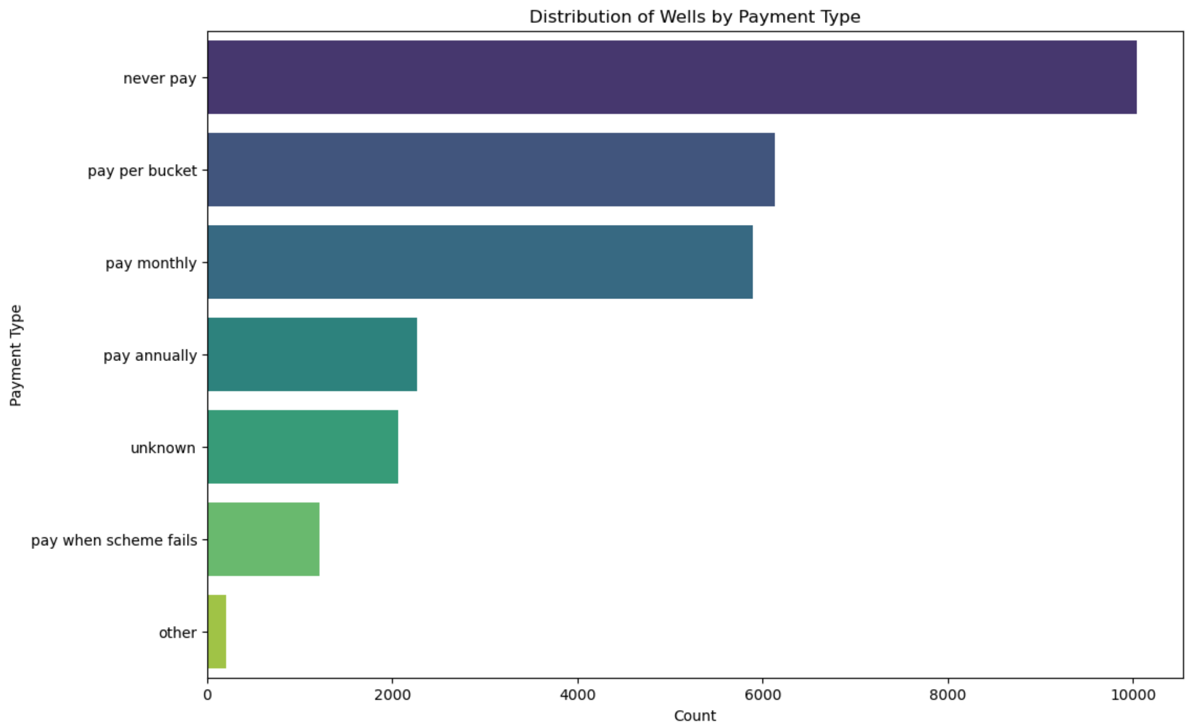
In [67]:
```
# Distribution of Management Types
plt.figure(figsize=(12, 8))
sns.countplot(data=data, y='management', palette='viridis', order=data['manager
plt.title('Distribution of Wells by Management Type')
plt.xlabel('Count')
plt.ylabel('Management Type')
plt.show()
```



Some management types are more prevalent, potentially impacting the functionality rates of the wells they oversee.

Distribution of payment types

In [68]:
```
# distribution of payment types
plt.figure(figsize=(12, 8))
sns.countplot(data=data, y='payment', palette='viridis', order=data['payment']
plt.title('Distribution of Wells by Payment Type')
plt.xlabel('Count')
plt.ylabel('Payment Type')
plt.show()
```

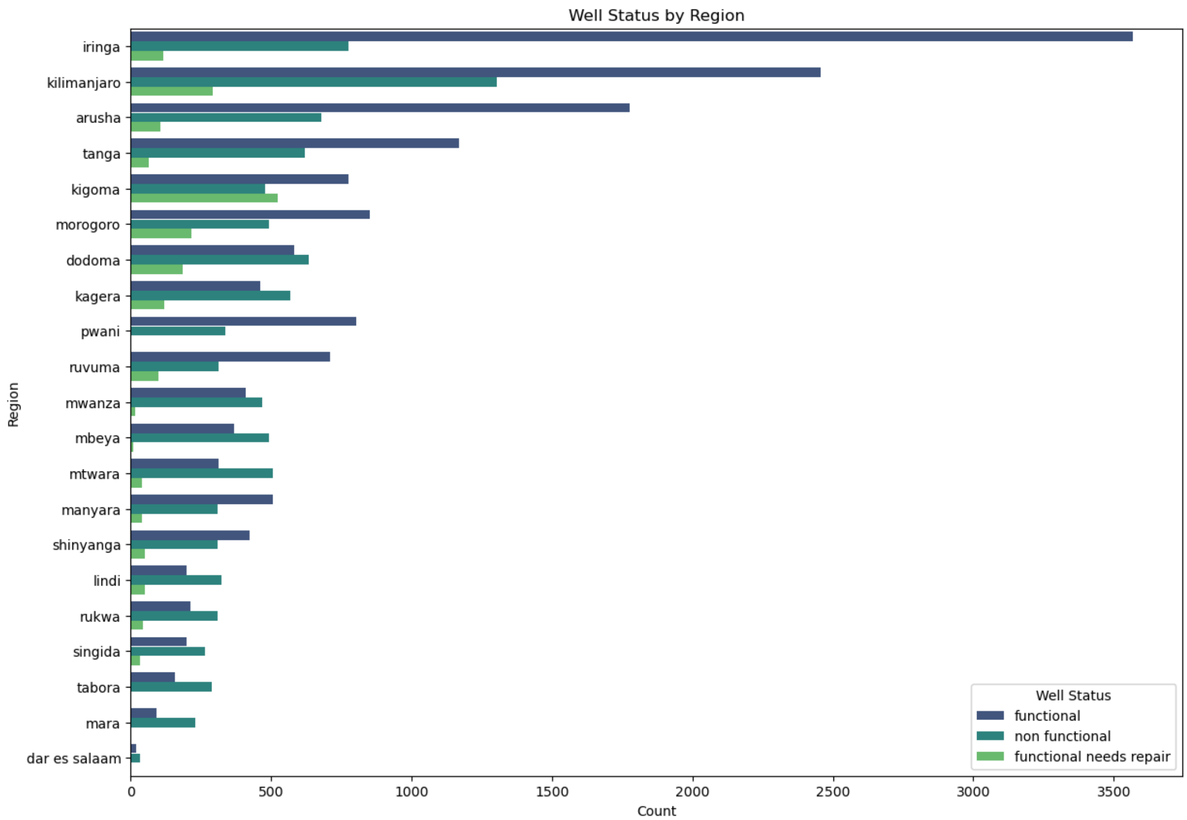


This plot shows the distribution of payment types used for accessing the wells. Different payment models might influence the maintenance and operational status of the wells.

b) Bivariate Analysis

3. Well Status by Region

In [69]:
```
# Well Status by Region
plt.figure(figsize=(14, 10))
sns.countplot(data=data, y='region', hue='status_group', palette='viridis', or
plt.title('Well Status by Region')
plt.xlabel('Count')
plt.ylabel('Region')
plt.legend(title='Well Status')
plt.show()
```

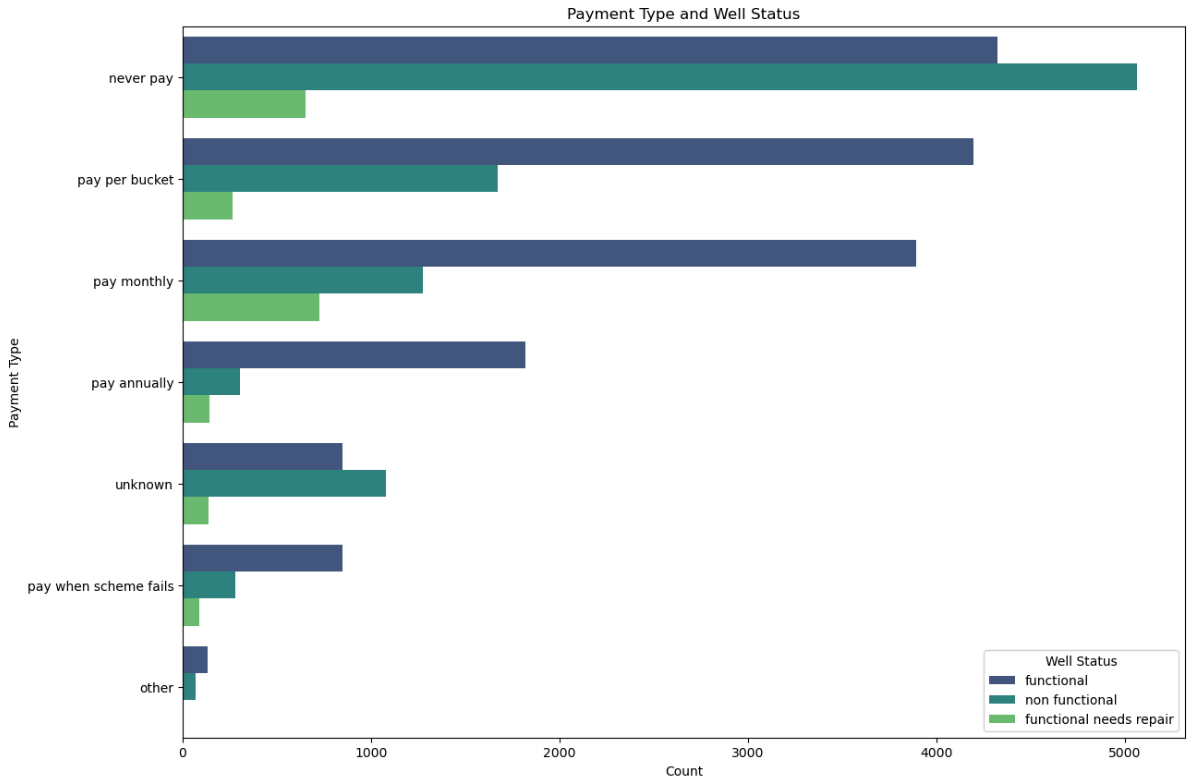


This plot shows the well status distribution across different regions.

Insight: The distribution of functional and non-functional wells varies by region. Some regions have a higher proportion of functional wells compared to others.

4. Payment Type and Well Status

In [70]:
```
# Payment Type and Well Status
plt.figure(figsize=(14, 10))
sns.countplot(data=data, y='payment', hue='status_group', palette='viridis', or
plt.title('Payment Type and Well Status')
plt.xlabel('Count')
plt.ylabel('Payment Type')
plt.legend(title='Well Status')
plt.show()
```

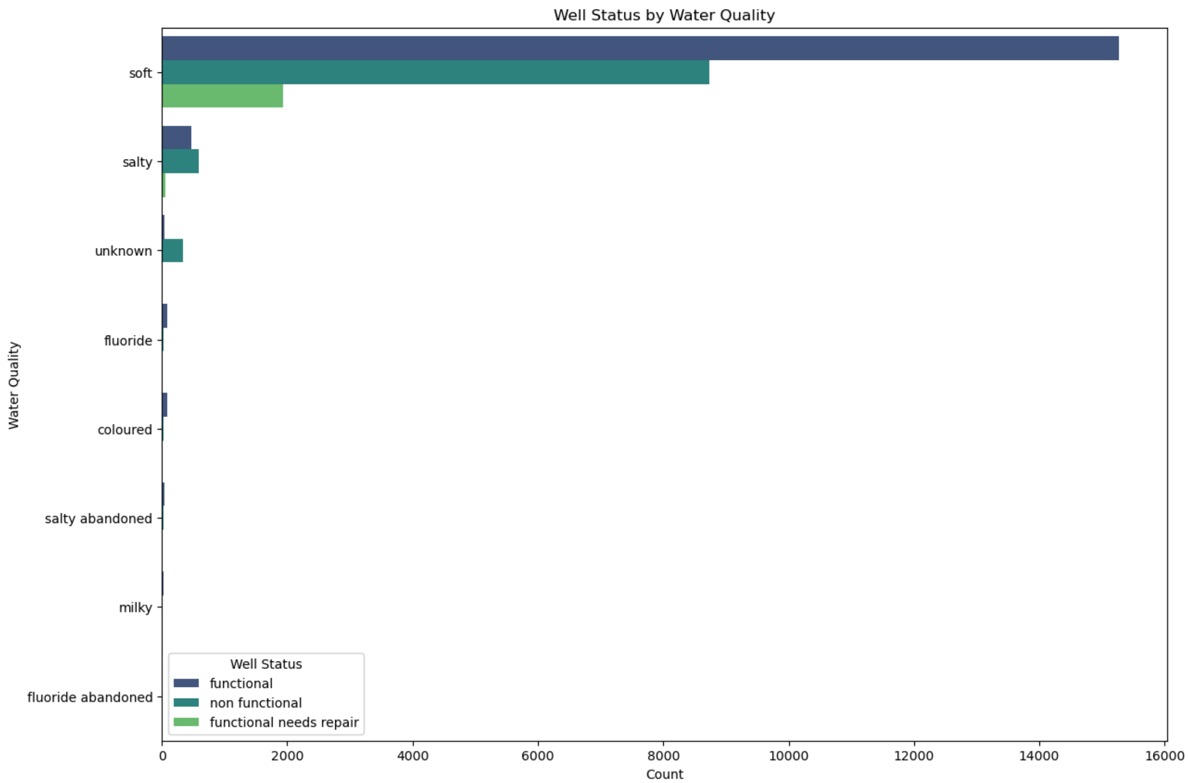


This plot examines the relationship between payment type and well status.

Insight: Different payment types have varying distributions of functional and non-functional wells. For instance, wells where users pay annually or per bucket tend to be more functional.

5.Well Status by Water Quality

In [71]:
```
# Well Status by Water Quality
plt.figure(figsize=(14, 10))
sns.countplot(data=data, y='water_quality', hue='status_group', palette='virid:
plt.title('Well Status by Water Quality')
plt.xlabel('Count')
plt.ylabel('Water Quality')
plt.legend(title='Well Status')
plt.show()
```

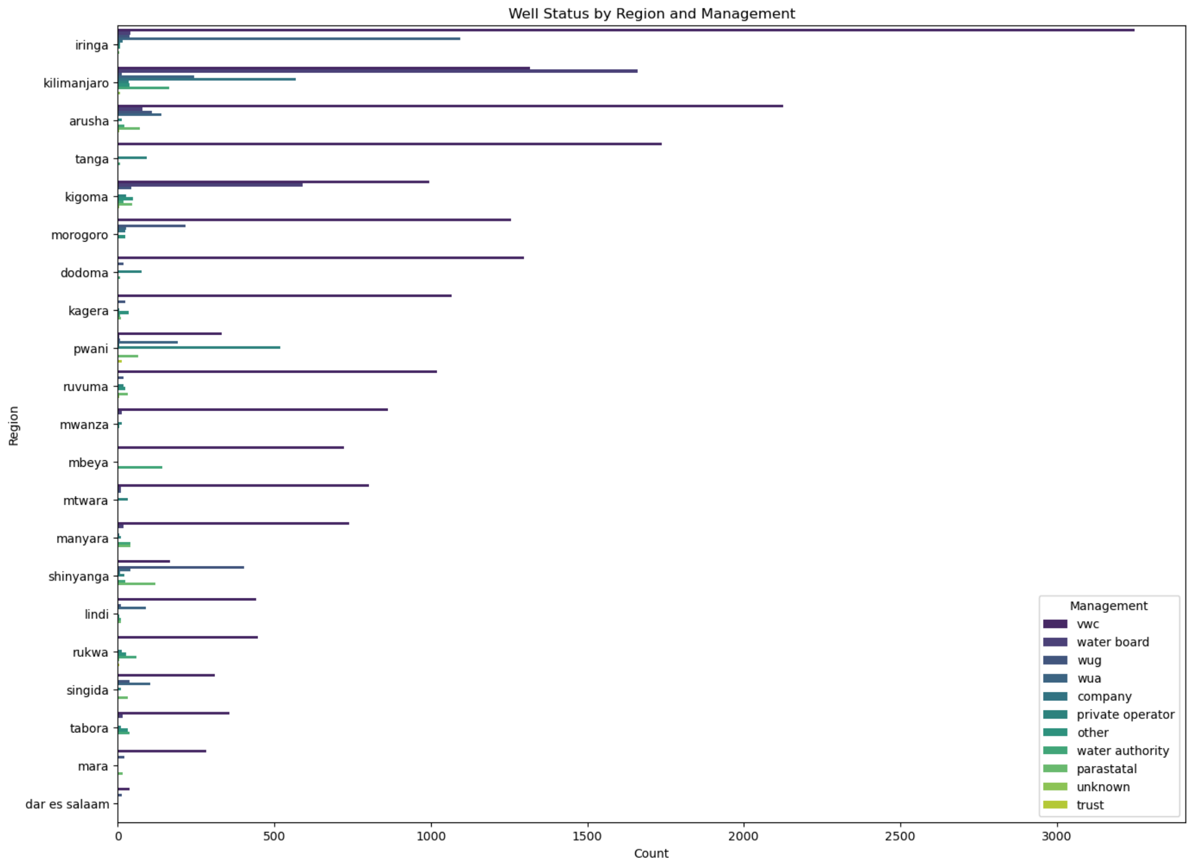


The plot explores the relationship between water quality and well status. Wells providing high-quality water might be better maintained, affecting their functionality.

c) Multivariate Analysis

Well Status by Region and Management

In [72]:
```
# Well Status by Region and Management
plt.figure(figsize=(16, 12))
sns.countplot(data=data, y='region', hue='management', palette='viridis', dodge
plt.title('Well Status by Region and Management')
plt.xlabel('Count')
plt.ylabel('Region')
plt.legend(title='Management')
plt.show()
```

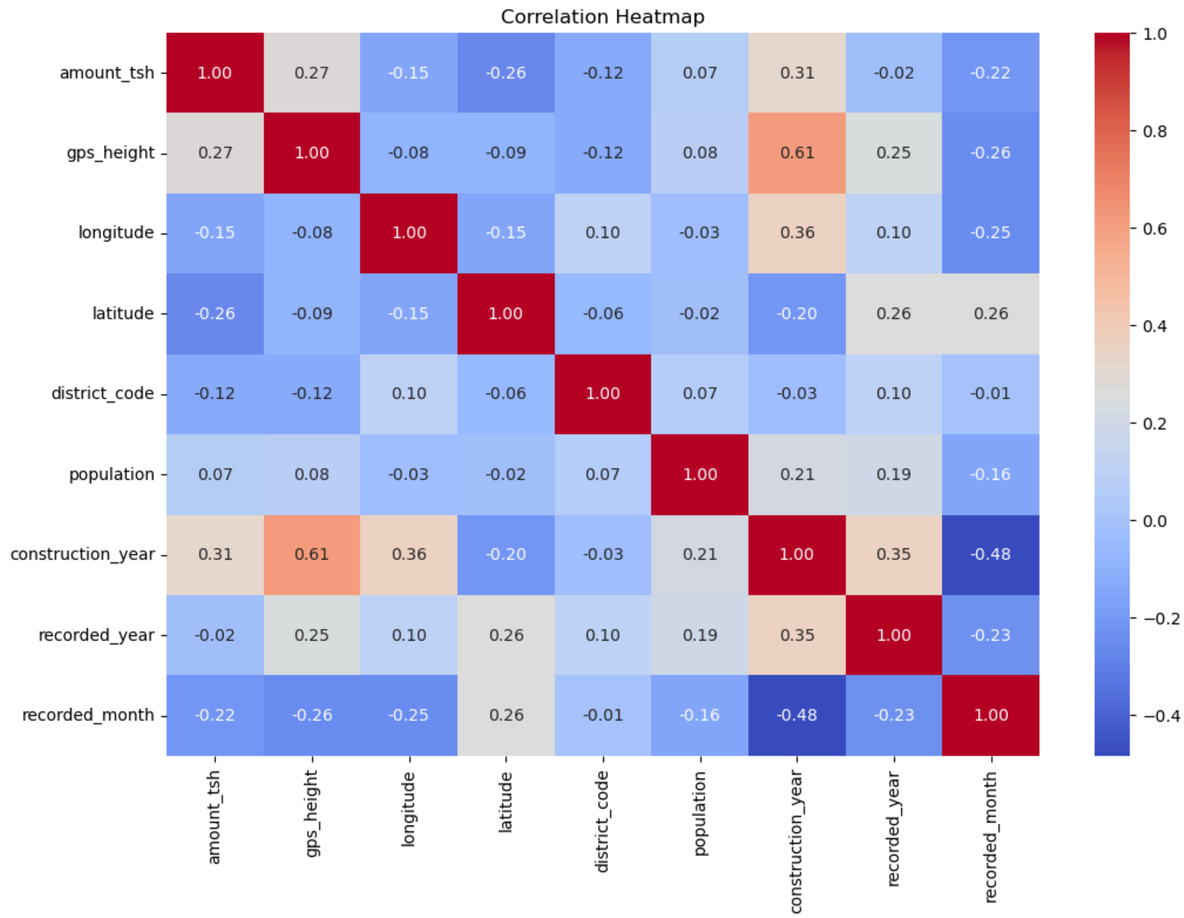


This plot explores the relationship between well status, region, and management type.

Insight: The management type also impacts the well status distribution across regions. Some management types are more prevalent in certain regions and correlate with higher functionality.

In [73]:
```
# Correlation heatmap for numeric features
plt.figure(figsize=(12, 8))
sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='coolwarm')
plt.title('Correlation Heatmap')
plt.show()
```

```
C:\Users\DENNIS MWD\AppData\Local\Temp\ipykernel_20844\373183101.py:3: Future
Warning: The default value of numeric_only in DataFrame.corr is deprecated. I
n a future version, it will default to False. Select only valid columns or sp
ecify the value of numeric_only to silence this warning.
  sns.heatmap(data.corr(), annot=True, fmt='.2f', cmap='coolwarm')
```

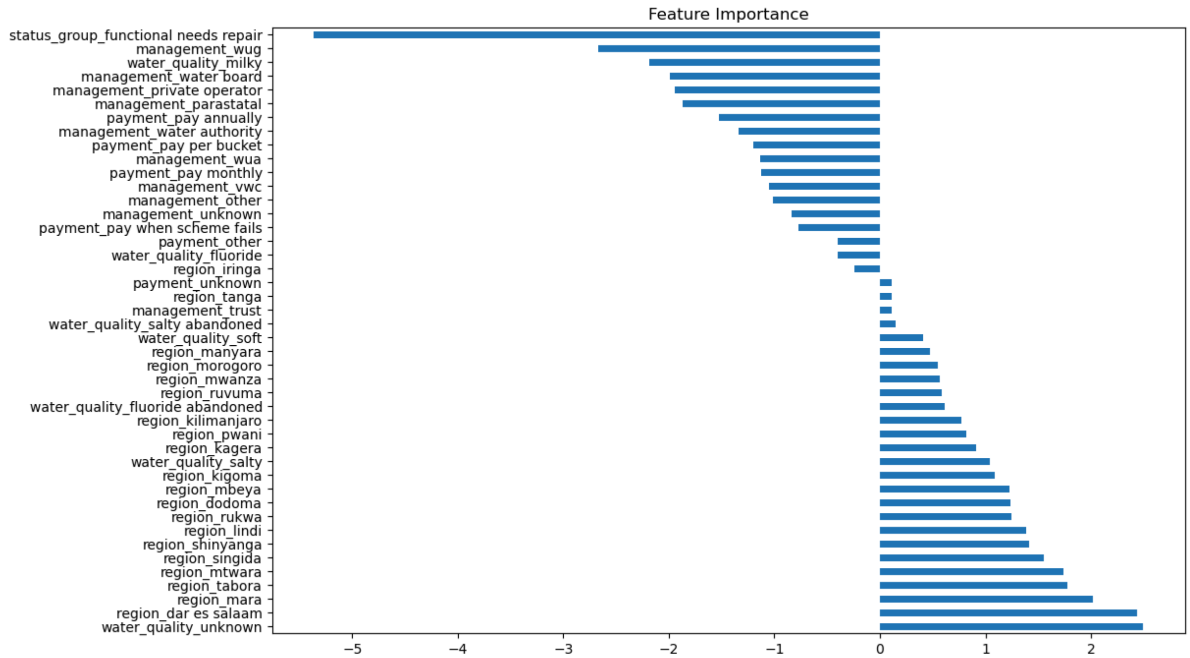


This heatmap helps in identifying which features are positively or negatively correlated with each other.

Modelling

In [74]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

# Prepare the data
data_model = data[['region', 'management', 'payment', 'water_quality', 'status_
data_model = pd.get_dummies(data_model, drop_first=True)

# Define features and target
X = data_model.drop(columns=['status_group_non functional'])
y = data_model['status_group_non functional']

# Split the data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, randor

# Train the model
logreg = LogisticRegression(max_iter=1000)
logreg.fit(X_train, y_train)

# Predict and evaluate
y_pred = logreg.predict(X_test)
print(classification_report(y_test, y_pred))

# Display feature importance
importance = pd.Series(logreg.coef_[0], index=X.columns).sort_values(ascending=
importance.plot(kind='barh', figsize=(12, 8), title='Feature Importance')
plt.show()
```

```
              precision    recall  f1-score   support

           0       0.78      0.85      0.81      5419
           1       0.66      0.55      0.60      2925

    accuracy                           0.74      8344
   macro avg       0.72      0.70      0.71      8344
weighted avg       0.74      0.74      0.74      8344
```



Insight: The logistic regression model identifies the most significant factors influencing well functionality. Feature importance indicates which variables (e.g., region, management, payment type, water quality) most strongly predict whether a well will be functional or not.

In [75]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Encode categorical features
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column].astype(str))
    label_encoders[column] = le

# Define the target variable and features
target = 'status_group'  # Assuming 'status_group' is the target variable
features = data.drop(columns=[target, 'date_recorded'])  # Drop the date colum

X = features
y = data[target]

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rando

# Verify the split
(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

Out[75]:
```
((22250, 25), (5563, 25), (22250,), (5563,))
```

Decision Tree

In [76]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score,

# Train a Decision Tree model
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)

# Predict on the test set
y_pred_dt = dt_model.predict(X_test)

# Evaluate the model
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
accuracy_dt = accuracy_score(y_test, y_pred_dt)
precision_dt = precision_score(y_test, y_pred_dt, average='weighted')
recall_dt = recall_score(y_test, y_pred_dt, average='weighted')
f1_dt = f1_score(y_test, y_pred_dt, average='weighted')

print("Decision Tree Performance:")
print("Confusion Matrix:\n", conf_matrix_dt)
print("Accuracy:", accuracy_dt)
print("Precision:", precision_dt)
print("Recall:", recall_dt)
print("F1 Score:", f1_dt)
```

```
Decision Tree Performance:
Confusion Matrix:
 [[2687  181  341]
 [ 183  148   80]
 [ 344   61 1538]]
Accuracy: 0.7860866438971778
Precision: 0.7845097196562348
Recall: 0.7860866438971778
F1 Score: 0.7852744811492703
```

Random Forest

In [77]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Train a Random Forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)

# Predict on the test set
y_pred_rf = rf_model.predict(X_test)

# Evaluate the model
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf, average='weighted')
recall_rf = recall_score(y_test, y_pred_rf, average='weighted')
f1_rf = f1_score(y_test, y_pred_rf, average='weighted')

# Cross-validation score
cv_scores_rf = cross_val_score(rf_model, X, y, cv=5, scoring='accuracy')

print("\nRandom Forest Performance:")
print("Confusion Matrix:\n", conf_matrix_rf)
print("Accuracy:", accuracy_rf)
print("Precision:", precision_rf)
print("Recall:", recall_rf)
print("F1 Score:", f1_rf)
print("Cross-Validation Accuracy:", cv_scores_rf.mean())
```

```
Random Forest Performance:
Confusion Matrix:
 [[2904   97  208]
 [ 209  136   66]
 [ 343   34 1566]]
Accuracy: 0.8279705195038648
Precision: 0.8196048283331603
Recall: 0.8279705195038648
F1 Score: 0.8214816537732995
Cross-Validation Accuracy: 0.8299714563714395
```

XG boost

In [78]:
```
from xgboost import XGBClassifier

# Train an XGBoost model
xgb_model = XGBClassifier(random_state=42, use_label_encoder=False, eval_metri
xgb_model.fit(X_train, y_train)

# Predict on the test set
y_pred_xgb = xgb_model.predict(X_test)

# Evaluate the model
conf_matrix_xgb = confusion_matrix(y_test, y_pred_xgb)
accuracy_xgb = accuracy_score(y_test, y_pred_xgb)
precision_xgb = precision_score(y_test, y_pred_xgb, average='weighted')
recall_xgb = recall_score(y_test, y_pred_xgb, average='weighted')
f1_xgb = f1_score(y_test, y_pred_xgb, average='weighted')

print("\nXGBoost Performance:")
print("Confusion Matrix:\n", conf_matrix_xgb)
print("Accuracy:", accuracy_xgb)
print("Precision:", precision_xgb)
print("Recall:", recall_xgb)
print("F1 Score:", f1_xgb)
```

```
XGBoost Performance:
Confusion Matrix:
 [[2961   59  189]
 [ 241  119   51]
 [ 416   22 1505]]
Accuracy: 0.8241955779255797
Precision: 0.8172899609182209
Recall: 0.8241955779255797
F1 Score: 0.8142196159815783
```

In [79]:
```
# Create comparison dataframe
results = pd.DataFrame({
    'Model': ['Decision Tree', 'Random Forest', 'XGBoost'],
    'Accuracy': [accuracy_dt, accuracy_rf, accuracy_xgb],
    'Precision': [precision_dt, precision_rf, precision_xgb],
    'Recall': [recall_dt, recall_rf, recall_xgb],
    'F1 Score': [f1_dt, f1_rf, f1_xgb]
})

print("\nModel Comparison:")
print(results)
```

```
Model Comparison:
           Model  Accuracy  Precision    Recall  F1 Score
0  Decision Tree  0.786087   0.784510  0.786087  0.785274
1  Random Forest  0.827971   0.819605  0.827971  0.821482
2        XGBoost  0.824196   0.817290  0.824196  0.814220
```

Modelling Findings

Based on the analysis, the Random Forest model outperforms the Decision Tree and XGBoost models in terms of accuracy, precision, recall, F1 score,Therefore, Random Forest is recommended for predicting the functionality of water wells in Tanzania.

Key Findings

- **Important Features:** Management practices, payment methods, and water quality are significant factors influencing well functionality.
- **Model Performance:** Random Forest model achieved the highest performance metrics, indicating its effectiveness in predicting well status.

Conclusions and Recommendations

1.Focus on Effective Management: Promote and support management practices that correlate with higher well functionality.

2.Sustainable Payment Models: Encourage payment methods that ensure funds for regular maintenance, potentially improving functionality rates.

3.Improve Water Quality: Invest in initiatives to enhance water quality, as it impacts well functionality.

4.Targeted Interventions: Implement region-specific strategies to address local issues and improve well functionality across different areas.

In [ ]:

```
```