WEAP and Audit Analysis for ACGIP Reporting

CY 2024

Analysis and Documentation Prepared by Kathryn Rominger

Contents

Introduction

This analysis was undertaken to understand the interactions between WEAP notifications – specifically Test Case 1: Excessive Watering and Test Case 2: Water Spike – and customer water audits. This analysis was done in support of the Annual Conservation Goal Implementation Plan (ACGIP) reporting to the Southwest Florida Water Management District and was an extension of a larger analysis of the effectiveness and efficiency of the WEAP system. The ACGIP delivers information on water conservation efforts that OUC has implemented and attempts to quantify the savings (both water and money) each initiative provides customers.

This document aims to serve as a guide for replication for future reporting needs. The analysis was done using Python on the Jupyter Notebook platform and Excel. The document follows the workflow of the analysis, switching between both tools; each section will be clearly labeled so that the reader can follow along. It is also important to note that most of the work done for the analysis can be done using only Excel, but the use of Python does significantly improve time efficiency.

Data Overview

Datasets

All datasets were pulled for calendar year 2024 (Jan 1st, 2024-Dec 31st, 2024) from Cognos.

- CCBSS1411 – WEAP TC1 Watering Too Many Days
- CCBSS1415 – WEAP TC2 Water Spike
- CCBSS1416 – WEAP TC3 Higher Than Normal Consumption
- CCBCI1417 – DataRaker Customer Contact Upload
- CCBSS1383 – FA Audit Status

Data Structure

The final datasets used after cleaning and combining (detailed later) contained the following variables:

Variable	Description
Premise ID	Unique ID for a property serviced by OUC; chosen to be the unique key for each data point.
WEAP Flagged Date	The date when a premise was flagged by internal algorithms as meeting one of the conditions needed to receive a WEAP notification.

FA Type	Indicates whether a premise is categorized as residential or commercial.
FA ID	Unique ID for field activity, specifically audits in the case of this analysis.
Premise Type	Describes the general use of a property.
Account Number	Account number associated with the premise.
Customer Name	Customer associate with the premise.
Audit Create Date	The date when an audit field activity instance was entered into the system; indicates the need for a water audit at the premise.
Difference in Days Between WEAP and Audit	A calculated field showing the time between the 'WEAP Flagged Date' and the 'Audit Create Date'; negative values indicate an audit was created first and positive values indicate a WEAP was sent first.
Validity	Indicates whether a data point is valid or an outlier.

Initial Preparation

Because this analysis came about as an extension of a larger project, initial preparation was only done on the four Cognos reports containing WEAP data (that is, all except for CBSS1383). The data used from this initial preparation and larger project was a count of how many WEAPs each premise received in 2024, along with a breakdown of each type of WEAP received. This dataset also included the WEAP Flagged Date variable.

Data Selection Rationale
Premise ID was chosen as the unique identifier for this analysis because of the stationary nature of a premise. In other words, Person ID and Account Number were not chosen because customers and their associated accounts can exist at multiple premises throughout the year, and therefore do not provide a stable reference point for tracking water usage over time. By using Premise ID, the analysis ensures that each data point is tied to a fixed location rather than a potentially changing customer or account, allowing for more accurate comparisons and trend assessments.

See Appendix A for full Python code.

Finally, working directly in the newly created Excel file, the WEAP count columns were renamed to include their proper names:

- 'Count_tc1' became 'TC1: Excess Watering'
- 'Count_tc2' became 'TC2: Water Spike'
- 'Count_t3' became 'TC3: High Water Consumption'
- 'Count_hius' became 'High Usage'

Analysis Workflow

Additional Data Preparation

Additional data preparation was needed to incorporate the Cognos report on audit data. This was also done using Python and is documented below:

*The following code and code in subsequent sections shows the process for the analysis of overlap between WEAP Test Case 1 and Proactive Audits. With a few minor changes, this is the same code used for all analyses between WEAP and audits. These changes will be outlined at the end of each section and the complete code is included in Appendices B, C, and D.

Commented [KR1]: Add appendices

```
import pandas as pd
```

```
weap = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CY 2024 WEAP Analysis.xlsx")
tc1 = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CCBSS1411 - WEAP TC1 Watering Too
                    Many Days.xlsx")
audit = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CCBSS1383 - FA Audit
                      Status.xlsx")
```

```
print(weap.columns)
print(tc1.columns)
print(audit.columns)
```

```
      Index(['Premise ID', 'TC1: Excess Watering', 'TC2: Water Spike',
             'TC3: High Water Consumption', 'High Usage', 'Total WEAPs Received',
             'Person ID', 'Account Number', 'Meter Number', 'Service Point ID',
             'Bill Route Type', 'Customer Class Description'],
            dtype='object')
      Index(['Customer Contact ID', 'Person ID', 'Account Number', 'Meter Number',
             'Premise ID', 'Service Point ID', 'Last Name', 'First Name', 'Entity Name',
             'Person or Business', 'Email', 'Bill Route Type', 'Primary Phone',
             'Home Phone', 'Cell Phone', 'Business Phone', 'Mailing Address 1',
             'Mailing Address 2', 'Mailing City', 'Mailing State', 'Mailing Postal',
             'Premise Address 1', 'Premise Address 2', 'Premise City', 'Premise State',
             'Premise Postal', 'Bill Print Intercept', 'Account Representative',
             'Customer Class Description', 'Flagged Date', 'Last 6 Months'],
            dtype='object')
      Index(['FA Type', 'FA Type Description', 'FA ID', 'Premise ID', 'Premise Type',
             'Account ID', 'Customer Name', 'Address', 'Postal', 'Phone', 'FA Status',
             'Create Date', 'Sched Date', 'Canceled Date', 'Canceled by',
             'Canceled Reason Description', 'Completed Date', 'Completed by',
             'Created by', 'CWTRAUDT', 'RWTRAUDT', 'AUDTJUST', 'ADTJSTRS', 'AUDITAST'],
            dtype='object')
```

```
# Find premises that only received 1 tc1 WEAP.
weap = weap[(weap['TC1: Excess Watering'] == 1) & (weap['Total WEAPs Received'] == 1)]
weap.head()
```

```
      Should return the top 5 rows of the resulting DataFrame.
```

```
# Specify columns to drop from tc1.
columns_to_drop = ['Customer Contact ID', 'Person ID', 'Account Number', 'Meter Number',
                   'Service Point ID', 'Last Name', 'First Name', 'Entity Name',
                   'Person or Business', 'Email', 'Bill Route Type', 'Primary Phone',
                   'Home Phone', 'Cell Phone', 'Business Phone', 'Mailing Address 1',
                   'Mailing Address 2', 'Mailing City', 'Mailing State',
                   'Mailing Postal', 'Premise Address 1', 'Premise Address 2',
                   'Premise City', 'Premise State', 'Premise Postal',
                   'Bill Print Intercept', 'Account Representative',
                   'Customer Class Description', 'Last 6 Months']

# Drop columns from tc1.
tc1.drop(columns=columns_to_drop, inplace=True)
tc1.head()
```

	Premise ID	Flagged Date
0	1282110001	2023-12-30
1	1686710001	2023-12-30
2	5825700001	2023-12-30
3	2208010001	2023-12-30
4	1424733218	2023-12-30

```
# Specify columns to drop from audit.
columns_to_drop = ['FA Type Description', 'Address', 'Postal', 'Phone', 'FA Status',
                   'Sched Date', 'Canceled Date', 'Canceled by',
                   'Canceled Reason Description', 'Completed Date', 'Completed by',
                   'Created by', 'AUDTJUST', 'ADTJSTRS', 'AUDITAST']

# Drop columns from audit.
audit.drop(columns=columns_to_drop, inplace=True)
audit.columns
```

```
Index(['FA Type', 'FA ID', 'Premise ID', 'Premise Type', 'Account ID',
       'Customer Name', 'Create Date', 'CWTRAUDT', 'RWTRAUDT'],
      dtype='object')
```

```
# Rename columns in audit to match columns in weap and facilitate merging.
audit = audit.rename(columns = {'Account ID': 'Account Number',
                                'Create Date': 'Flagged Date'})
```

```
# Keep only proactive water audit data.
audit = audit[audit['FA Type'].str.contains('WTR', na = False)]
audit = audit[audit['CWTRAUDT'].str.contains('WEAP', na = True)]
audit = audit[audit['RWTRAUDT'].str.contains('WEAP', na = True)]
```

```
audit.head()
```
```
      Should return the top 5 rows of the resulting DataFrame.
```

```
# Ensure 'Account Number' in both DataFrames are the same type of data (string) to
facilitate merging
weap['Account Number'] = weap['Account Number'].astype(str)
audit['Account Number'] = audit['Account Number'].astype(str)
```

```
# Merge weap and tc1 DataFrames on 'Premise ID'
weap = weap.merge(tc1, on = ['Premise ID'], how = 'inner')
weap.head()
```
```
      Should return the top 5 rows of the resulting DataFrame.
```

```
# Keep only the first and last rows, which should be 'Premise ID' and 'Flagged Date'
weap = weap.iloc[:, [0, -1]]
weap.head()
```

	Premise ID	Flagged Date
0	40189904	2024-10-20
1	78410001	2024-05-26
2	78900001	2024-11-04
3	80310001	2024-08-22
4	95936169	2024-10-03

```
# Merge weap and audit DataFrames on 'Premise ID'
overlap = weap.merge(audit, on = ['Premise ID'], how = 'inner')
overlap.head()
```
```
      Should return the top 5 rows of the resulting DataFrame.
```

```
# Ensure duplicate rows are dropped.
overlap = overlap.drop_duplicates()
overlap
```
```
      Should return a condensed number of rows of the specified DataFrame and a
      description of the shape (#rows x #columns)
```

```
# Export overlap DataFrame to an Excel file.
overlap.to_excel("CY 2024 WEAP TC1 vs Audit Overlap.xlsx", index = False)
```

Code Explanation

To ensure that the data is relevant and manageable, unnecessary columns from both the TC1 and audit datasets are removed before merging. In the TC1 dataset, a wide range of customer information, such as names, contact details, mailing addresses, and account representatives, is dropped. This reduces redundancy and ensures that only the necessary fields—Premise ID and Flagged Date—are retained for analysis. Similarly, in the audit dataset, columns related to field activity type descriptions, addresses, and audit status updates are removed. The remaining fields, including Premise ID, Account Number, and Flagged Date, provide essential details for tracking premises flagged for excessive watering and linking them to proactive water audits.

Additionally, column names in the audit dataset are standardized to match those in WEAP, ensuring consistency when merging. To facilitate accurate merging, the Account Number column is explicitly converted to a string in both datasets. The merging process involves first joining WEAP with TC1 on Premise ID, retaining only relevant premises and flagged dates. Then, the refined WEAP dataset is merged with the audit dataset to identify overlapping premises that received both excessive watering notices and proactive water audits. Duplicate records are removed to ensure data integrity, and the final overlap dataset is exported to an Excel file for further review analysis.

Changes for Additional Analyses

1. For analyses comparing **WEAP Test Case 2** with audit data:
    a. The file containing WEAP TC2 data should be loaded in instead of TC1 in the second block of code, and the DataFrame should be named ‘tc2’.
        i. Subsequently, all instances of ‘tc1’ throughout the code should be changed to ‘tc2’.
    b. In the fourth block of code, the first column name should be changed from ‘TC1: Excess Watering’ to ‘TC2: Water Spike’.
    c. In the fifth block of code, ensure that the correct columns are being dropped.
        i. Specifically, the ‘Last 6 Months’ column does not exist in the Test Case 2 data and therefore does not need to be dropped.
2. For analyses comparing WEAP data with **traditional audit** data:
    a. The eighth block of code should read as follows:

```
# Keep only traditional water audit data.
audit = audit[audit['FA Type'].str.contains('WTR', na = False)]
audit = audit[(audit['CWTRAUDT'] == 'W-AUDIT') | (audit['CWTRAUDT'].isna())]
audit = audit[(audit['RWTRAUDT'] == 'W-AUDIT') | (audit['RWTRAUDT'].isna())]

audit.head()
```

3. For all additional analyses, change the exported file name in the last block of code to ensure outputs from previous work are not overwritten or lost.

Manual Data Adjustments and Analysis in Excel

Once all the analyses were completed, the exported Excel files were manually combined into the same file on different sheets.

A new column was created to calculate the number of days between when a WEAP notification was sent and when an audit was created. This was done simply by subtracting the 'Audit Create Date' from the 'WEAP Flagged Date' (i.e. WEAP – audit = # of days).

The final step in the data cleaning process was to decide which premises were outliers. That is, how many days between a WEAP and an audit indicates that they were sent for the same water consumption incident, or separate incidents. In other words, determining which WEAPs and audits were successful in changing a customer's behavior because they did not receive another intervention for the same issues.

For example, if a customer received an audit 40 days after they received a WEAP notification, how can it be determined if the audit was created in response to the same water consumption increase that triggered a WEAP notification? Manually checking consumption history for each premise was outside the scope of the analysis. Several outlier identification methods were tested, but due to complexity of water consumption behavior, a simple approach was chosen: if the time between a WEAP notification and an audit was more than 30 days, the premise was considered an outlier.

Data Selection Rationale
It is understood that the algorithms used to produce WEAP notifications do not allow for the same type of WEAP to be sent to the same premise more than once within 30 days. Therefore, an assumption was made that if a customer did not receive additional intervention (i.e. an audit) within 30 days of receiving a WEAP notification, or vice versa, the customer likely responded to the intervention and changed their water consumption behavior or fixed the issue that prompted the need for either a WEAP notification or an audit.

These numbers were used to get an overall idea of the overlap between the two WEAP types analyzed and the different types of audits.

Final Breakdown by WEAP and Audit Type

Each of the four WEAP notifications and the two audit types are reported separately in the ACGIP; the audits are also broken down by commercial and residential premises. Therefore, it was necessary to break down how the WEAPs and audits in this analysis interacted. This was completed using Python, but could be done manually in Excel. The code to find this breakdown is as follows:

```
import pandas as pd
```

```
audit = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CCBSS1383 - FA Audit
                      Status.xlsx")
tc1_pro = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CY 2024 WEAP TC1 vs Audit
                        Overlap.xlsx")
tc1_trad = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CY 2024 WEAP TC1 vs Audit
                         Overlap.xlsx", sheet_name = 'Traditional Audits')
tc2_pro = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CY 2024 WEAP TC2 vs Audit
                        Overlap.xlsx")
tc2_trad = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CY 2024 WEAP TC2 vs Audit
                         Overlap.xlsx", sheet_name = 'Traditional Audits')
```

```
audit.head()
```
```
       Should return the top 5 rows of the DataFrame.
```

```
# Keep only the necessary columns.
audit = audit[['FA ID', 'CWTRAUDT', 'RWTRAUDT']]
```

```
print(tc1_pro.columns)
print(tc1_trad.columns)
print(tc2_pro.columns)
print(tc2_trad.columns)
```
```
       Index([                                        'Premise ID',
                                          'WEAP Flagged Date',
                                                    'FA Type',
                                                      'FA ID',
                                               'Premise Type',
                                             'Account Number',
                                              'Customer Name',
                                          'Audit Create Date',
                        'Difference in Days Between WEAP and Audit',
                                                     'Validity',
                                               'Unnamed: 10',
              'Premises that received 1 WEAP total that was TC1:',
                                               'Unnamed: 12',
                                               'Unnamed: 13',
                                               'Unnamed: 14',
                                                        940],
             dtype='object')
       Index([                                        'Premise ID',
                                          'WEAP Flagged Date',
                                                    'FA Type',
                                                      'FA ID',
                                               'Premise Type',
```

```
                                          'Account Number',
                                           'Customer Name',
                                        'Audit Create Date',
                'Difference in Days Between WEAP and Audit',
                                                 'Validity',
                                              'Unnamed: 10',
       'Premises that received 1 WEAP total that was TC1:',
                                              'Unnamed: 12',
                                              'Unnamed: 13',
                                              'Unnamed: 14',
                                                      940],
      dtype='object')
Index([                                       'Premise ID',
                                     'WEAP Flagged Date',
                                                 'FA Type',
                                                   'FA ID',
                                           'Premise Type',
                                         'Account Number',
                                           'Customer Name',
                                        'Audit Create Date',
                'Difference in Days Between WEAP and Audit',
                                                 'Validity',
                                              'Unnamed: 10',
       'Premises that received 1 WEAP total that was TC2:',
                                              'Unnamed: 12',
                                              'Unnamed: 13',
                                                      848,
            '(from CY 2024 WEAP Analysis; WEAP Breakdown)'],
      dtype='object')
Index([                                       'Premise ID',
                                     'WEAP Flagged Date',
                                                 'FA Type',
                                                   'FA ID',
                                           'Premise Type',
                                         'Account Number',
                                           'Customer Name',
                                  'Audit Scheduled Date',
                'Difference in Days Between WEAP and Audit',
                                                 'Validity',
                                              'Unnamed: 10',
       'Premises that received 1 WEAP total that was TC2:',
                                              'Unnamed: 12',
                                              'Unnamed: 13',
                                              'Unnamed: 14',
                                                      848,
            '(from CY 2024 WEAP Analysis; WEAP Breakdown)'],
      dtype='object')
```

```
# Drop unnecessary columns.
tc1_pro = tc1_pro.iloc[:, :-6]
tc1_trad = tc1_trad.iloc[:, :-6]
tc2_pro = tc2_pro.iloc[:, :-6]
tc2_trad = tc2_trad.iloc[:, :-7]
```

```
#Drop rows where the validity is 'outlier'.
tc1_pro = tc1_pro[tc1_pro['Validity'] != 'Outlier']
```

```
tc1_trad = tc1_trad[tc1_trad['Validity'] != 'Outlier']
tc2_pro = tc2_pro[tc2_pro['Validity'] != 'Outlier']
tc2_trad = tc2_trad[tc2_trad['Validity'] != 'Outlier']
```

Test Case 1

Proactive

```
# Merge test case 1 data with proactive water audit data.
tc1_pro = tc1_pro.merge(audit, on = ['FA ID'], how = 'inner')
tc1_pro
```

Should return the resulting DataFrame.

```
# Create a DataFrame containing commercial audit data.
com_tc1_pro = tc1_pro.dropna(subset = ['CWTRAUDT'])
com_tc1_pro
```

Should return the resulting DataFrame.

```
# Create a DataFrame containing residential audit data.
res_tc1_pro = tc1_pro.dropna(subset = ['RWTRAUDT'])
res_tc1_pro
```

Should return the resulting DataFrame.

Traditional

```
# Merge test case 1 data with traditional water audit data.
tc1_trad = tc1_trad.merge(audit, on = ['FA ID'], how = 'inner')
tc1_trad
```

Should return the resulting DataFrame.

```
# Create a DataFrame containing commercial audit data.
com_tc1_trad = tc1_trad.dropna(subset = ['CWTRAUDT'])
com_tc1_trad
```

Should return the resulting DataFrame.

```
# Create a DataFrame containing residential audit data.
res_tc1_trad = tc1_trad.dropna(subset = ['RWTRAUDT'])
res_tc1_trad
```

Should return the resulting DataFrame.

Test Case 2

Proactive

```
# Merge test case 2 data with proactive water audit data.
tc2_pro = tc2_pro.merge(audit, on = ['FA ID'], how = 'inner')
tc2_pro
```

Should return the resulting DataFrame.

```
# Create a DataFrame containing commercial audit data.
com_tc2_pro = tc2_pro.dropna(subset = ['CWTRAUDT'])
com_tc2_pro
```

Should return the resulting DataFrame.

```
# Create a DataFrame containing residential audit data.
res_tc2_pro = tc2_pro.dropna(subset = ['RWTRAUDT'])
res_tc2_pro
```

Should return the resulting DataFrame.

Traditional

```
# Merge test case 2 data with traditional water audit data.
tc2_trad = tc2_trad.merge(audit, on = ['FA ID'], how = 'inner')
tc2_trad
```

```
       Should return the resulting DataFrame.
```

```
# Create a DataFrame containing commercial audit data.
com_tc2_trad = tc2_trad.dropna(subset = ['CWTRAUDT'])
com_tc2_trad
```

```
       Should return the resulting DataFrame.
```

```
# Create a DataFrame containing residential audit data.
res_tc2_trad = tc2_trad.dropna(subset = ['RWTRAUDT'])
res_tc2_trad
```

```
       Should return the resulting DataFrame.
```

```
# Export DataFrames to individual sheets in the same Excel file.
dfs = {'TC1 vs Proactive Com Audits': com_tc1_pro,
       'TC1 vs Proactive Res Audits': res_tc1_pro,
       'TC1 vs Traditional Com Audits': com_tc1_trad,
       'TC1 vs Traditional Res Audits': res_tc1_trad,
       'TC2 vs Proactive Com Audits': com_tc2_pro,
       'TC2 vs Proactive Res Audits': res_tc2_pro,
       'TC2 vs Traditional Com Audits': com_tc2_trad,
       'TC2 vs Traditional Res Audits': res_tc2_trad}

with pd.ExcelWriter('WEAP vs Audits Complete Breakdown.xlsx', engine = 'xlsxwriter') as
writer:
    for sheet_name, df in dfs.items():
        df.to_excel(writer, sheet_name = sheet_name, index = False)

print("Excel file saved successfully.")
```

```
       Excel File saved successfully.
```

Code Explanation

This script processes and analyzes water audit data by merging test case data from WEAP with audit records, filtering based on validity, and categorizing the results into commercial and residential audits. The process begins by loading multiple Excel files containing proactive and traditional test case data, as well as audit records. The audit data is cleaned by retaining only essential columns, while unnecessary columns from the test case datasets are removed. Additionally, rows labeled as ‘Outlier’ in the “Validity” column are filtered out to ensure data accuracy.

Next, the script merges the proactive and traditional test cases with the audit dataset using “FA ID” as the key. This allows for a structured comparison between test case results and actual audit findings. The merged data is then categorized into commercial audits, which contain non-null values in the “CWTRAUDT” column, and residential audits, which have non-null values in the “RWTRAUDT” column.

Interpreting Output and Results

The final exported Excel sheets provide a structured comparison between WEAP notifications and water audits, offering insights into the effectiveness of the WEAP system in driving customer action. By merging and categorizing data, the analysis identifies the extent to which proactive and traditional audits coincide with WEAP notifications. The key metric—the difference in days between a WEAP notification and an audit—serves as an indicator of whether audits were likely triggered by the same water consumption issues flagged by WEAP.

A negative value in this metric indicates that a WEAP notification preceded an audit, potentially indicating that the notification was ineffective in prompting customer action, thus requiring further intervention. Conversely, a positive value indicates that an audit occurred first, followed by a WEAP, which could imply that the initial audit did not resolve the issue or that the problem resurfaced. The threshold of 30 days was used to define whether a WEAP and an audit were related to the same water consumption event, with outliers exceeding this threshold being excluded from the final counts.

From a reporting perspective, those were included in the final breakdown represented "failed" WEAPs and audits, and therefore should not be included in the final ACGIP report, as they were not successful in reducing customers' water use nor saving them money.

Recommendations for Future Analysis

This analysis was the first attempt at a deep-dive into understanding how WEAP notifications support OUC's water conservation goals. In addition, as mentioned before, this analysis came about as an extension of a larger project aimed at deepening the understanding of the WEAP notification system. As this understanding grows, adjustments should be made to the ACGIP analysis of WEAPs and audits to further refine metrics that are being reported. Some recommended adjustments include:

- **Refining the 30-Day Threshold:** While the 30-day window was chosen based on operational understanding of WEAP frequency, further analysis using historical consumption trends or behavior patterns could validate or adjust this threshold for greater accuracy.
- **Incorporating Consumption Data:** Linking actual water usage data would provide deeper insights into whether customers effectively reduced their consumption following a WEAP or audit intervention.

- **Automating Data Processing:** While this analysis involved manual adjustments in Excel, future efforts could streamline this step by integrating more automation into the Python workflow, reducing the potential for human error.

Appendix A

The following code includes blocks of code and accompanying comments (notated with a '#' and italicized) and any code output that appear if the code is running correctly. It is important to note that the file paths in the second block of code will need to be updated to reflect where the files are saved on the individual user's computer.

```
import pandas as pd
import numpy as np
```

```
tc1 = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CCBSS1411 - WEAP TC1 Watering Too
                    Many Days.xlsx")
tc2 = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CCBSS1415 - WEAP TC2 Water
                    Spike.xlsx")
tc3 = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CCBSS1416 - WEAP TC3 Higher Than
                    Normal Consumption.xlsx")
hius = pd.read_excel("C:/Users/ROMIK456/OneDrive/WEAP/CCBCI1417 - DataRaker Customer
                     Contact Upload (4).xlsx", skiprows = 3)
```

```
print(tc1.columns)
print(tc2.columns)
print(tc3.columns)
print(hius.columns)
```

```
      Index(['Customer Contact ID', 'Person ID', 'Account Number', 'Meter Number',
             'Premise ID', 'Service Point ID', 'Last Name', 'First Name', 'Entity Name',
             'Person or Business', 'Email', 'Bill Route Type', 'Primary Phone',
             'Home Phone', 'Cell Phone', 'Business Phone', 'Mailing Address 1',
             'Mailing Address 2', 'Mailing City', 'Mailing State', 'Mailing Postal',
             'Premise Address 1', 'Premise Address 2', 'Premise City', 'Premise State',
             'Premise Postal', 'Bill Print Intercept', 'Account Representative',
             'Customer Class Description', 'Flagged Date', 'Last 6 Months'],
            dtype='object')
      Index(['Customer Contact ID', 'Person ID', 'Account Number', 'Meter Number',
             'Premise ID', 'Service Point ID', 'Last Name', 'First Name', 'Entity Name',
             'Person or Business', 'Email', 'Bill Route Type', 'Primary Phone',
             'Home Phone', 'Cell Phone', 'Business Phone', 'Mailing Address 1',
             'Mailing Address 2', 'Mailing City', 'Mailing State', 'Mailing Postal',
             'Premise Address 1', 'Premise Address 2', 'Premise City', 'Premise State',
             'Premise Postal', 'Bill Print Intercept', 'Account Representative',
             'Customer Class Description', 'Flagged Date'],
            dtype='object')
      Index(['RowNumber', 'Contractor List', 'DataRaker Point ID',
             'Past Quarter Avg Daily Usage (Gal)', 'Past Quarter Sum (Gal)',
             'Past Quarter Read Count', 'Avg Monthly Usage (Gal)', 'LSPID', 'Premise ID',
             'Service Point ID ', 'Account Age (Days)', 'Account Number ',
             'Gross Square Feet', 'Lot Size', 'Rate Code', 'Service Point Type ',
             'Status', 'Group', 'Customer Contact ID', 'Person ID', 'Meter Number',
             'Last Name', 'First Name', 'Entity Name', 'Person or Business', 'Email',
             'Bill Route Type', 'Primary Phone ', 'Home Phone', 'Cell Phone',
             'Business Phone', 'Mailing Address 1', 'Mailing Address 2', 'Mailing City ',
             'Mailing State', 'Mailing Postal', 'Premise Address 1', 'Premise Address 2',
             'Premise City ', 'Premise State', 'Premise Postal', 'Bill Print Intercept',
             'Account Representative', 'Customer Class Description', 'Flagged Date',
             'Acre Description'],
            dtype='object')
      Index(['Customer Contact ID', 'Person ID', 'Account ID', 'Premise ID',
             'Service Point ID', 'Badge Number', 'Comment', 'Letter Print Date'],
            dtype='object')
```

```
# Specify columns to drop from tc1.
columns_to_drop = ['Customer Contact ID', 'Last Name', 'First Name', 'Entity Name',
                   'Person or Business', 'Email', 'Primary Phone', 'Home Phone',
                   'Cell Phone', 'Business Phone', 'Mailing Address 1',
                   'Mailing Address 2', 'Mailing City', 'Mailing State', 'Mailing Postal',
                   'Premise Address 1', 'Premise Address 2', 'Premise City',
                   'Premise State', 'Premise Postal', 'Bill Print Intercept',
                    'Account Representative', 'Last 6 Months']

# Drop columns from tc1.
tc1.drop(columns=columns_to_drop, inplace=True)
```

```
# Specify columns to drop from tc2.
columns_to_drop = ['Customer Contact ID', 'Last Name', 'First Name', 'Entity Name',
                   'Person or Business', 'Email', 'Primary Phone', 'Home Phone',
                   'Cell Phone', 'Business Phone', 'Mailing Address 1',
                   'Mailing Address 2', 'Mailing City', 'Mailing State', 'Mailing Postal',
                   'Premise Address 1', 'Premise Address 2', 'Premise City',
                   'Premise State', 'Premise Postal', 'Bill Print Intercept',
                   'Account Representative']

# Drop columns from tc22
tc2.drop(columns=columns_to_drop, inplace=True)
```

```
# Specify columns to drop from tc3.
columns_to_drop = ['RowNumber', 'Contractor List', 'DataRaker Point ID',
                   'Past Quarter Avg Daily Usage (Gal)', 'Past Quarter Sum (Gal)',
                   'Past Quarter Read Count', 'Avg Monthly Usage (Gal)', 'LSPID',
                   'Account Age (Days)', 'Gross Square Feet', 'Lot Size', 'Rate Code',
                   'Service Point Type ', 'Status', 'Group', 'Customer Contact ID',
                   'Last Name', 'First Name', 'Entity Name', 'Person or Business', 'Email',
                   'Primary Phone ', 'Home Phone', 'Cell Phone', 'Business Phone',
                   'Mailing Address 1', 'Mailing Address 2', 'Mailing City ',
                   'Mailing State', 'Mailing Postal', 'Premise Address 1',
                   'Premise Address 2', 'Premise City ', 'Premise State', 'Premise Postal',
                   'Bill Print Intercept', 'Account Representative', 'Acre Description']

# Drop columns from tc1, tc2, tc3
tc3.drop(columns=columns_to_drop, inplace=True)
```

```
# Remove trailing (and leading) spaces from column names in tc3
tc3.columns = tc3.columns.str.strip()
```

```
# Specify columns to drop from hius.
columns_to_drop = ['Customer Contact ID', 'Comment']

# Drop columns from hius.
hius.drop(columns=columns_to_drop, inplace=True)
```

```
# Rename hius columns to match other DataFrames and facilitate merging.
hius = hius.rename(columns = {'Account ID': 'Account Number',
                              'Badge Number': 'Meter Number',
                              'Letter Print Date': 'Flagged Date'})
```

```
# Ensure consistent columns across all DataFrames.
print(tc1.columns)
print(tc2.columns)
print(tc3.columns)
print(hius.columns)
```

```
      Index(['Person ID', 'Account Number', 'Meter Number', 'Premise ID',
             'Service Point ID', 'Bill Route Type', 'Customer Class Description',
             'Flagged Date'],
```

```
      dtype='object')
Index(['Person ID', 'Account Number', 'Meter Number', 'Premise ID',
       'Service Point ID', 'Bill Route Type', 'Customer Class Description',
       'Flagged Date'],
      dtype='object')
Index(['Premise ID', 'Service Point ID', 'Account Number', 'Person ID',
       'Meter Number', 'Bill Route Type', 'Customer Class Description',
       'Flagged Date'],
      dtype='object')
Index(['Person ID', 'Account Number', 'Premise ID', 'Service Point ID',
       'Meter Number', 'Flagged Date'],
      dtype='object')
```

```
# List DataFrames to clean
dataframes = [tc1, tc2, tc3, hius]


# Loop to clean data in all DataFrames.
for df in dataframes:
    for column in df.columns:
        if df[column].dtype == 'object':
            df[column] = (
                df[column]
                .str.replace('.0', '', regex=False)  # Remove '.0'
                .str.replace(',', '')                # Remove commas if any
                .replace('nan', pd.NA)               # Replace 'nan' strings with NaN
            )

        # Skip datetime columns before attempting integer conversion
        if not np.issubdtype(df[column].dtype, np.datetime64):
            try:
                df[column] = df[column].astype('Int64')
            except ValueError:
                pass  # Skip columns that can't be converted
```

```
# Count occurrences of 'Premise ID' in each DataFrame and create new DataFrames to store
the information
count_tc1 = tc1['Premise ID'].value_counts().reset_index()
count_tc1.columns = ['Premise ID', 'Count_tc1']

count_tc2 = tc2['Premise ID'].value_counts().reset_index()
count_tc2.columns = ['Premise ID', 'Count_tc2']

count_tc3 = tc3['Premise ID'].value_counts().reset_index()
count_tc3.columns = ['Premise ID', 'Count_tc3']

count_hius = hius['Premise ID'].value_counts().reset_index()
count_hius.columns = ['Premise ID', 'Count_hius']

# Merge all counts into a single DataFrame using 'Premise ID'
counts_merged = count_tc1.merge(count_tc2, on='Premise ID', how='outer') \
                         .merge(count_tc3, on='Premise ID', how='outer') \
                         .merge(count_hius, on='Premise ID', how='outer')

# Fill missing values with 0 (if necessary)
counts_merged = counts_merged.fillna(0)

# Convert counts to integers
counts_merged[['Count_tc1', 'Count_tc2', 'Count_tc3', 'Count_hius']] =
    counts_merged[['Count_tc1', 'Count_tc2', 'Count_tc3', 'Count_hius']].astype(int)

counts_merged
```

	Premise ID	Count_tc1	Count_tc2	Count_tc3	Count_hius
0	10001	0	0	0	1
1	320001	0	0	0	1
2	410001	0	0	0	1
3	510001	0	0	0	1
4	586463	0	0	0	1
...	...	...	...	...	...
72702	9999455136	3	0	0	0
72703	9999500001	0	0	0	1
72704	9999510001	0	0	0	4
72705	9999600001	0	0	0	1
72706	9999900001	0	0	0	1

72707 rows × 5 columns

```
# Merge counts_merged with other DataFrames on 'Premise ID' to retain original
information
merged = counts_merged
for df in [tc1, tc2, tc3, hius]:
    merged = pd.merge(merged, df, on='Premise ID', how='outer', suffixes=('', '_dup'))

# Print the merged DataFrame columns for debugging
print("Merged DataFrame columns:")
print(merged.columns)

# Define a function to identify duplicate columns and combine their values, separated
with a comma
def combine_columns(df):
    for column in df.columns:
        if '_dup' in column:
            original_column = column.replace('_dup', '')
            if original_column in df.columns:
                # Check if the suffixed column exists before accessing it
                if column in df.columns:
                    # Convert all values to strings and combine
                    df[original_column] = df[[original_column, column]].apply(
                        lambda x: ', '.join(x.dropna().astype(str).unique()), axis=1
                    )
                    df.drop(columns=[column], inplace=True)
    return df

# Run merged through combine_columns
final_aggregated = combine_columns(merged)

# Drop any remaining duplicate columns
final_aggregated = final_aggregated.loc[:, ~final_aggregated.columns.duplicated()]
```

```
Merged DataFrame columns:
Index(['Premise ID', 'Count_tc1', 'Count_tc2', 'Count_tc3', 'Count_hius',
       'Person ID', 'Account Number', 'Meter Number', 'Service Point ID',
       'Bill Route Type', 'Customer Class Description', 'Flagged Date',
       'Person ID_dup', 'Account Number_dup', 'Meter Number_dup',
```

```
            'Service Point ID_dup', 'Bill Route Type_dup',
            'Customer Class Description_dup', 'Flagged Date_dup',
            'Service Point ID_dup', 'Account Number_dup', 'Person ID_dup',
            'Meter Number_dup', 'Bill Route Type_dup',
            'Customer Class Description_dup', 'Flagged Date_dup', 'Person ID_dup',
            'Account Number_dup', 'Service Point ID_dup', 'Meter Number_dup',
            'Flagged Date_dup'],
           dtype='object')
```

```
# Pring final_aggregated to ensure proper merging.
final_aggregated
```

Should return a condensed number of rows of the specified DataFrame and a description of the shape (#rows x #columns)

```
# Drop duplicate Premise IDs, keeping the first occurrence
weap_counts = final_aggregated.drop_duplicates(subset='Premise ID')

# Reset the index if you want a clean index after dropping duplicates
weap_counts.reset_index(drop=True, inplace=True)

weap_counts
```

Should return a condensed number of rows of the specified DataFrame and a description of the shape (#rows x #columns)

```
# Calculating the total WEAPs received.
weap_counts['Total WEAPs Received'] = (
    weap_counts['Count_tc1'] +
    weap_counts['Count_tc2'] +
    weap_counts['Count_tc3'] +
    weap_counts['Count_hius']
)

weap_counts.head()
```

Should return a condensed number of rows of the specified DataFrame, with the newly created column added to the end, and a description of the shape #rows x #columns)

```
 # Export weap_counts DataFrame to an Excel file
 weap_counts.to_excel("CY 2024 WEAP Counts.xlsx", index = False)
```