DIAGNOSTIC ANALYSIS

Talia Arauzo

PURPOSE

Assess HRL's **accuracy** in determining under-registration based on **AWWA** standards

CURRENT ANALYSIS

COMPONENTS OF ANALYSIS

01 DATA IMPORTATION
Import test results and append metadata

02 DATA CLEANING
Drop meters with insufficient flows and metadata

03 VALIDATION RESULT
Calculate meter validation rate based on AWWA standards

04 VALIDATION MATCHING
Determine if utility and AWWA validation results match

05 HIT RATES
Calculate Valor and utility hit rates in determining under-registration

01

DATA IMPORTATION

Import test results and append metadata

IMPORT DATA

FLOW TEST DATA

Given .csv file,
import necessary columns
and convert to desired
data types

METADATA

Append respective meter
metadata to flow test data,
mapping on
cis_meter_num

FLOW TEST DATA

Import utility flow test data.

In [40]:
```
# Function for importing flow test data
def import_flow_data(file_name):

    # Read utility csv file, select certain columns, make type(cis_meter_num) = str
    utility_df = pd.read_csv(file_name, na_values = 'nan')[['cis_meter_num', 'cis_location_id',
                                        'hidden_revenue_id', 'validation_result', 'flow_test_gpm_1',
                                        'flow_test_accuracy_1', 'flow_test_gpm_2', 'flow_test_accuracy_2',
                                        'flow_test_gpm_3', 'flow_test_accuracy_3']].dropna(axis = 1, how = 'all')

    # Converting to correct data type
    # If Suez test results
    if 'suez' in file_name:
        utility_df['cis_meter_num'] = ['{:.0f}'.format(float(id)) for id in utility_df['cis_meter_num']]
    # If HRL ID present
    if utility_df.columns.contains('hidden_revenue_id'):
        utility_df = utility_df.astype({'cis_meter_num': 'str', 'cis_location_id': 'str'})
        utility_df['hidden_revenue_id'] = ['{:.0f}'.format(float(id)) for id in utility_df['hidden_revenue_id']]
        utility_df = utility_df.replace('nan', np.nan)
    # If no HRL ID present
    else:
        utility_df = utility_df.astype({'cis_meter_num': 'str', 'cis_location_id': 'str'}).replace('nan', np.nan)

    return utility_df
```

METADATA

Append metadata (customer type, meter size, manufacturer) to flow test data.

In [7]:

```
# Function for matching meter metadata with flow test data
def match_data(utility_df, file_name):

    # If American Water data
    if 'aw' in file_name:

        # American Water metadata
        meter_metadata = pd.read_csv('metadata/aw_metadata.csv', na_values = 'nan').astype({'cis_meter_num': 'str'})

    # If Clayton County data
    elif 'ccwa' in file_name:

        # Clayton County metadata
        meter_metadata = pd.read_csv('metadata/ccwa_metadata.csv', sep = '|', na_values = 'nan').astype({'cis_meter_num
        meter_metadata['cis_meter_num'] = meter_metadata['cis_meter_num'].str.lstrip('0').str.replace(r'[^0-9]', "")

    # If San Gabriel data
    elif 'sgvwc' in file_name:

        # San Gabriel metadata
        meter_metadata = pd.read_csv('metadata/sgvwc_metadata.csv', sep = '|',  na_values = 'nan').astype({'cis_meter_r

    # If Newport data
    elif 'newport' in file_name:

        # Newport metadata
        meter_metadata = pd.read_csv('metadata/newport_metadata.csv', sep = '|', na_values = 'nan').astype({'cis_meter_

    # If Suez data
    elif 'suez' in file_name:

        # Suez metadata
        meter_metadata = pd.read_csv('metadata/suez_metadata.csv', sep = '|', na_values = 'nan').astype({'cis_meter_num

    # Add metadata by matching cis_meter_num
    utility_df = utility_df.merge(meter_metadata, how = 'left', on = 'cis_meter_num').dropna(axis = 1, how = 'all')

    return utility_df
```

02

DATA CLEANING

Drop meters with insufficient test flows and metadata

DATA CLEANING

INSUFFICIENT METADATA

Drop meter if meter size not available

INSUFFICIENT FLOWS

Drop meter if all flow test values are missing

DATA CLEANING

Drop rows with insufficient flows and metadata.

In [8]:
```
# Function for determining which meters have insufficient test data
def determine_drop(r):

    # Flow columns of interest
    flow_col = ['flow_test_accuracy_1', 'flow_test_accuracy_2', 'flow_test_accuracy_3']

    # If meter size not available in metadata, drop meter
    if np.isnan(r['meter_size']):
        return 'drop'
    else:
        # If all flow accuracy values are NaN, drop meter
        if np.isnan(r[flow_col[0]]) & np.isnan(r[flow_col[1]]) & np.isnan(r[flow_col[2]]):
            return 'drop'
        else:
            return 'check'
```

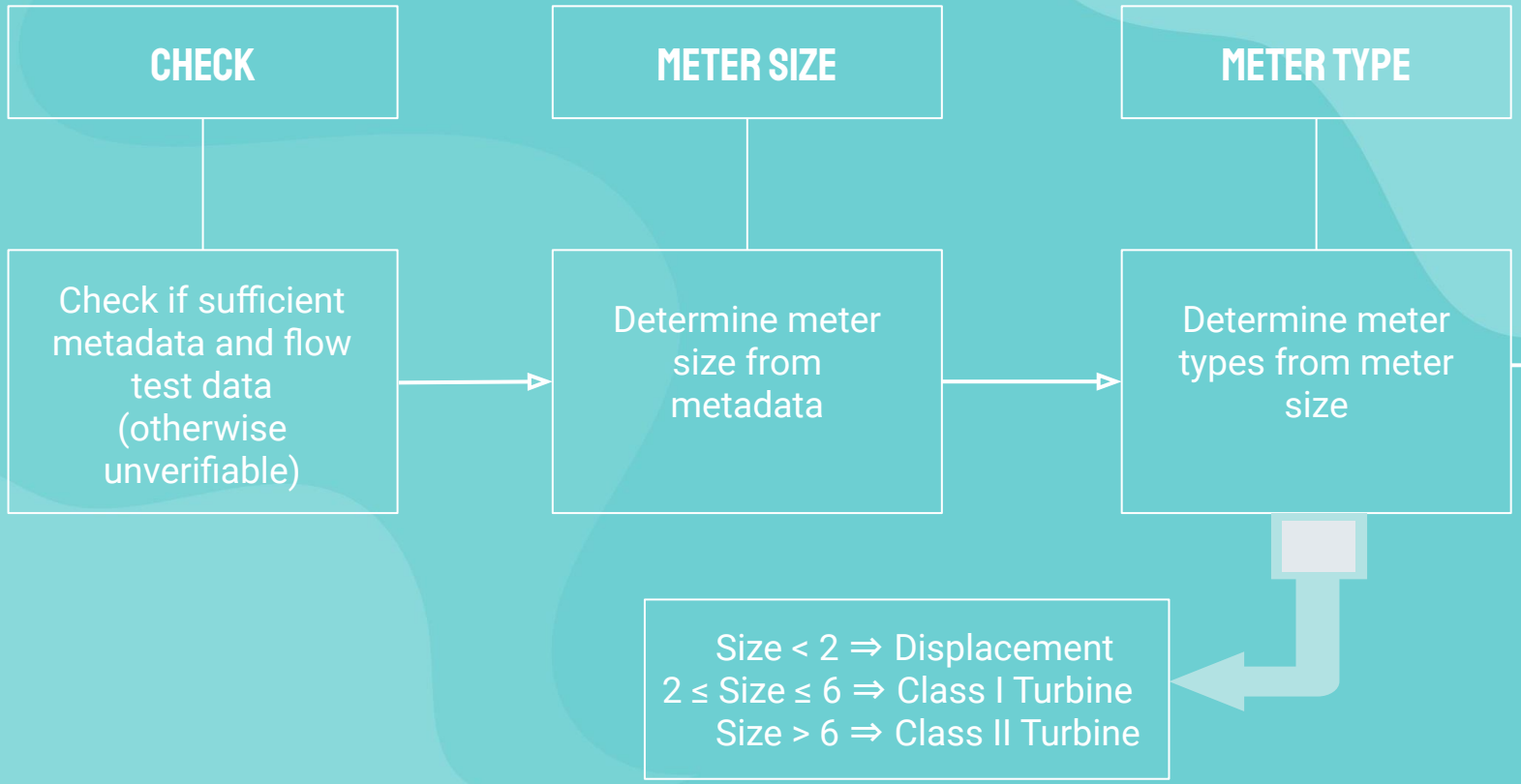
In [9]:
```
# Function for dropping meters with insufficient flow test data
def drop_flow_data(utility_df):

    # Determine which meters to drop
    utility_df['drop'] = utility_df.apply(determine_drop, axis = 1)

    return utility_df
```

03

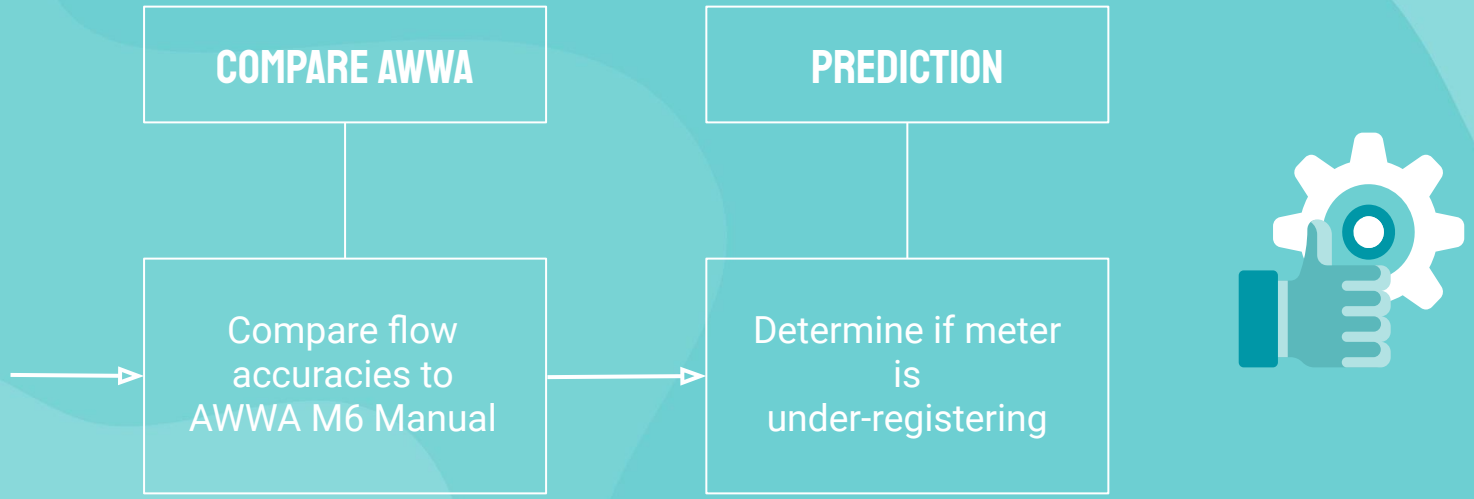
VALIDATION RESULT

Calculate meter validation rate based
on AWWA standards

VALIDATION RESULT



VALIDATION RESULT



VALIDATION RESULT

COMPARE AWWA

FLOWS LIMITS:
- ➔ Low flow: ± 5%
- ➔ Int & high flow: ± 25%

VALIDATION RESULTS:
- ➔ Under-register: 1
- ➔ Correct & over-register: 0

NO FLOWS	ALL FLOW LIMITS	SOME FLOW LIMITS
1. No flows provided • Any accuracies below limits ⇒ 1 • Otherwise ⇒ 0	1. All flows within limits • Any accuracies below limits ⇒ 1 • Otherwise ⇒ 0	1. No flow provided 2. Flow within limits • Accuracy below limits ⇒ 1

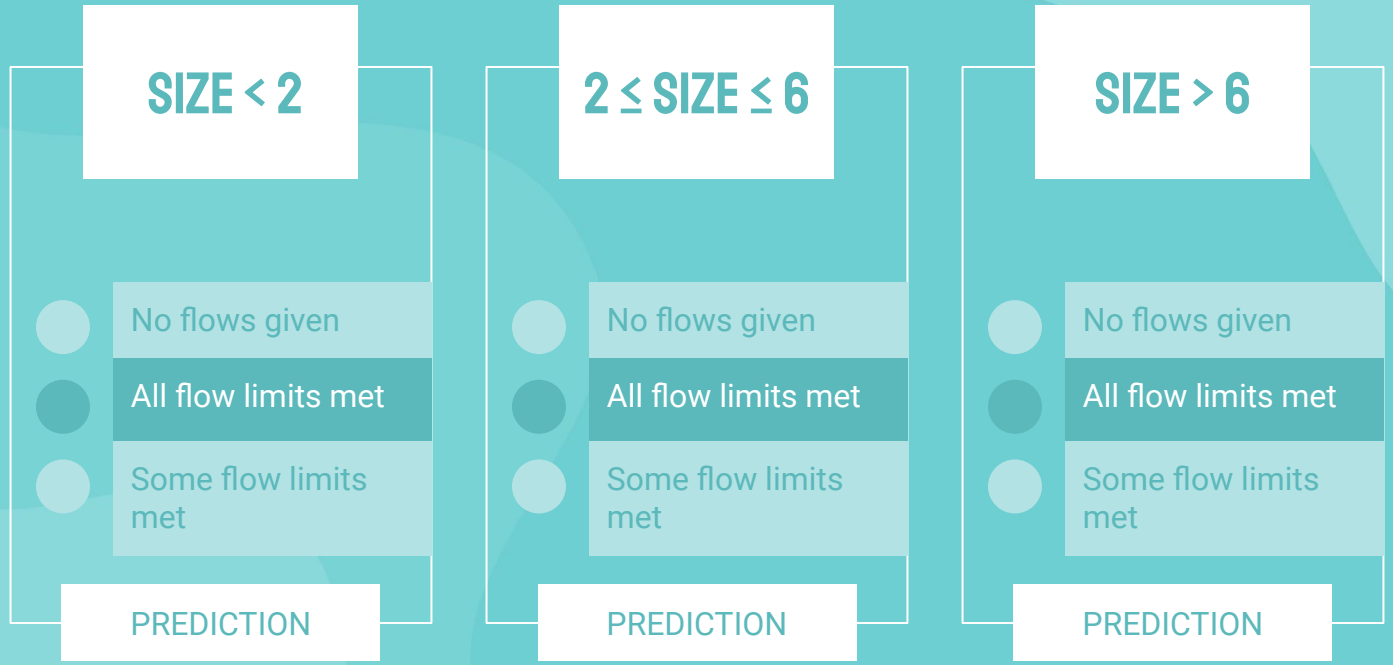
AWWA MANUAL

Table 5-3 Test requirements for new, rebuilt, and repaired cold–water meters*

Displacement Meters (AWWA C700 and C710)

	Maximum Rate (All Meters)				Intermediate Rate (All Meters)				Minimum Rate (New and Rebuilt)				Minimum (Repaired)
Size	Flow Rate†	Test Quantity††		Accuracy Limits	Flow Rate**	Test Quantity††		Accuracy Limits	Flow Rate	Test Quantity††		Accuracy Limits	Accuracy Limits
in.	*gpm*	*gal*	*ft³*	*percent*	*gpm*	*gal*	*ft³*	*percent*	*gpm*	*gal*	*ft³*	*percent*	*percent (min)*
½	8	100	10	98.5–101.5	2	10	1	98.5–101.5	¼	10	1	95–101	90
½ × ¾	8	100	10	98.5–101.5	2	10	1	98.5–101.5	¼	10	1	95–101	90
⅝	15	100	10	98.5–101.5	2	10	1	98.5–101.5	¼	10	1	95–101	90
⅝ × ¾	15	100	10	98.5–101.5	2	10	1	98.5–101.5	¼	10	1	95–101	90
¾	25	100	10	98.5–101.5	3	10	1	98.5–101.5	½	10	1	95–101	90
1	40	100	10	98.5–101.5	4	10	1	98.5–101.5	¾	10	1	95–101	90
1½	50	100	10	98.5–101.5	8	100	10	98.5–101.5	1½	100	10	95–101	90
2	100	100	10	98.5–101.5	15	100	10	98.5–101.5	2	100	10	95–101	90

VALIDATION RESULT



04

VALIDATION MATCHING

Determine if utility and AWWA validation results match

MATCHING VALIDATIONS

UTILITY PREDICTION

From flow test data

=

VALOR (AWWA) PREDICTION

From AWWA comparison
(if unverifiable, skip)

⇒ 1

MATCHING VALIDATIONS

Determine which validation results - utility & AWWA - match.

Note: If AWWA prediction is NaN, meter is unverifiable and excluded from matching.

In [12]:
```python
# Function for determining which utility & AWWA predictions match.
def determine_match(r):

    # Utility prediction
    utility_pred = r['validation_result']

    # AWWA prediction
    awwa_pred = r['awwa_validation_result']

    # If meter is unverifiable, ignore
    if np.isnan(awwa_pred):
        return 'ignore'

    # If meter is verifiable
    else:

        # If utility and AWWA predictions match
        if utility_pred == awwa_pred:
            return 1
        # Else utility and AWWA predictions do not match
        else:
            return 0
```

Add prediction match determination to utility data.

In [13]:
```python
# Function for adding match data to utility data
def match(utility_df):

    # Determine which meters to drop
    utility_df['match'] = utility_df.apply(determine_match, axis = 1)

    return utility_df
```

05

HIT RATES

Calculate Valor and utility hit rates in determining under-registration

HIT RATES

VALOR HIT RATE

Proportion of meters that were correctly flagged by Valor according to AWWA

UTILITY HIT RATE

Proportion of meters that were correctly flagged by Valor according to utility

$$hit\ rate = \frac{\#AWWA\ predicts\ under-register}{\#verifiable\ tested\ meters}$$

HIT RATES

Calculate Valor hit rate.

Note: If AWWA prediction is NaN, meter is unverifiable and excluded from hit rate.

In [29]:
```
# Function for determining valor hit rate
def valor_hit_rate(utility_df):

    # Exclude unverifiable meters
    utility = utility_df[utility_df['awwa_validation_result'].notnull()]

    # Number AWWA predicts under-registering
    no_awwa_pred = utility[utility['awwa_validation_result'] == 1].shape[0]

    # Number flagged under-registering
    no_flagged = utility.shape[0]

    # If all meters were unverifiable
    if no_flagged == 0:
        return 'Indeterminable'
    # Else if any meters were verifiable
    else:
        # Calculate hit rate
        hit_rate = no_awwa_pred / no_flagged
        return hit_rate
```

NEXT STEPS

NEXT STEPS OF ANALYSIS

01 **REVENUE INCORPORATION**
Access utility rate structure to calculate revenue associated with hit rate

02 **TIME SERIES IMPLEMENTATION**
Track change in utility hit rate over time

03 **SCRIPT CONVERSION**
Convert code from Jupyter notebooks to script

04 **SUGGESTIONS?**

THANKS!

Would love take any suggestions or questions