**#The following code is divided into two sections:**

**#1) Programs aggregating data into reportable categories after cleaning in the markdown report**

**# (Note that this is finally applied in the report() function for all variables)**

**#2) In-line programs that print key phrases depending on the result**

**####################  
# DATA AGGREGATION #  
####################**

# m.prep is used to read whether a date observation falls between two dates

# “rep\_date” is a MMDDYYYY variable representing the month of data reported by the entry technician.

# An initial IF statement determines if “rep\_date” falls after the last day before the first month, and (&&) before the first day of the following month. If the condition is met then an integer will be returned representing the number of months since the intervention began. In the first IF statement, for example, “rep\_date” will return 0 if the reported date is listed as 04/05/2019, as it falls within the first range. The mdy() lubridate function us used to read the variable as a date.

**# MONTH SINCE DAYS OF INTERVENTION**

# Month\_Int applies m.prep to read and report observations by row. Results are returned as a matrix and printed in the final readable report() function at the end of this section.

**# RESIDENT DAYS**

# This function returns a column with the total number of resident days in a month. The grep() function is used to search for all columns named “resident\_days”. The dataset “fac” is subset out by brackets. NA values will exist in the data if the Entry Technician is returning back to the survey (which will be a common occurrence), therefore, na.rm = T. The rowSums() function is applied to the results to replace NA values with 0. Note there is only one 'resident\_days' column in the dataset.

**#URINE CULTURES AVAILABLE**

# Returns a column with the number of urine culture and sensitivity tests ordered in a month. Similar to the Res\_Days function (above), all rows are summed with rowSums(), grep() subsets out the ‘orgavail’ name, and NAs are removed.

**# URINE CULTURES WITH >100K CFUS**

#Returns a column with the number of urine cultures with greater than 100,000 colony forming units (CFUs) in each row. First, the dataframe ‘urin’ is created to subset out all variable names starting with org and including cnt; [\\d](file:///\\d) is included between these two strings to include all columns with this name. A total of 108 columns are included in ‘urin’. Each four columns in ‘urin’ contains a numeric from 0-3 representing a urine lab culture result. 0 == no lab result, 1 == <50,000 CFUs, 2 == 50,000 – 100,000 CFUs, 3== >100,000 CFUs. In each row, every 4 columns are lab results for the same patient. For example, row 2 columns 5-8 represents 4 results for the same patient. The purpose of the urine\_pos function is to identify whether any one of these 4 patient cultures contained >100,000 CFUs (represented as a 3).

Next, the container ‘result’ is created, as this function will need to assess whether any of the four cultures contained a 3. A for loop then reads each row (which represents 27 patients) by chunks of 4 columns (representing the patient’s 4 lab results). The IF statement assesses whether each 4 columns in each row contains a 3, and prints 1 to the ‘result’ container. rowSums then adds each 1 into a column.

**# URINE CULTURES WITH 1--99K CFUS**

# Returns a column with the number of urine cultures between 0 to 100,000 CFUs. This function is identical to the urine\_pos function above, except the loop will print 1 to the 'result' when either a 1 or 2 exists. First 'urin' is created to include all lab results for patients. The container 'result' is created. A loop prints 1 to 'result' if any 4 columns for each patient contained a 1 (<50,000 CFUs) or 2 (50,000 - 100,000 CFUs). rowSums then adds each 1 into a column.

**#URINE CULTURES WITH ANY COLONY COUNT**

# This function returns a vector column if any colony count was reported for a patient in the dataset. This function is an extension of the two functions above by printing 1 to the ‘result’ container if a numeric 1, 2, or 3 is noted in the dataset. As before, the numeric 0 == no lab result, 1 == <50,000 CFUs, 2 == 50,000 – 100,000 CFUs, 3== >100,000 CFUs. The dataframe ‘urin’ is first created to subset out all lab results. The lab result variable names are subset out by grep and include the strings org and cnt; [\\d](file:///\\d) is included between these two strings as a numeric distinguishes different variable names. In each row, every 4 columns are lab results for the same patient.

A ‘result’ container is similarly created. The for loop reads ‘urin’ by row by chunks of 4. If statements print 1 to the ‘result’ container if 1, 2, or 3 are observed in the chunk of 4. rowSums sums all 1’s in the ‘result’ row into a single vector column.

**#ANTIBIOTICS PRESCRIBED PRIOR TO LAB RESULT AVAILABILITY**

# This function returns a vector of the total number of antibiotics prescribed for a UTI prior to availability of lab results. The purpose of this vector is to summarize whether the date reported in ‘repp’ (which represents the date when a urine culture result was reported) falls after dates in ‘antt’ (which represents dates at which 3 potential antibiotics were prescribed). First the dataframe ‘rep’ is created by subsetting out all columns including the string ‘repdate’. The dataset ‘repp’ then applies the mdy() function to convert the dates into a readable date format. The process is repeated for antibiotic prescription dates with ‘ant’ and applying mdy() for the ‘antt’ dataset. Note that ‘antt’ has three times as many columns, as three potential antibiotic prescription fields are available for each ‘repp’ field.

# A ‘result’ container is established. A for loop then determines if each ‘repp’ observation is greater than the three corresponding ‘antt’ observations. If the condition is met, 1 is printed to the ‘result’ container. rowSums sums all 1’s in the ‘result’ row into a single vector column.

# ANTIBIOTICS PRESCRIPTIONS ISSUED (For presumed UTI)

# Row sums are printed for all columns with ‘anticnt’. These columns represent the number of prescriptions given for a patient. Similar to the Res\_Days function (above), all rows are summed with rowSums(), grep() subsets out the ‘anticnt’ name, and NAs are removed.

#ANTIBIOTIC DOT (For presumed UTI)

#Row sums are printed for all columns with ‘anti#days’, representing the days of therapy (DOT) for each antibiotic prescribed. The # sign ranges from 1-3 as three DOT counts can be reported per patient. This function therefore sums the total number of days, for all patients, for all antibiotics prescribed into a single count. Similar to the Res\_Days function (above), rows are summed with rowSums(), grep() subsets out the ‘anti\\ddays’ name, and NAs are removed. The [\\d](file:///\\d) is included between org and cnt to ignore numbers 1-3, and include all columns with these names.

#POSITIVE ISOLATES FOR C. DIFF INFECTION

#A single question on the survey asks for the total number of positive clostridium difficile infections for the month. Because this is a single question, the grep() function is not necessary. Instead, rowSums() is applied to transform the tibble into a vector and make NA's 0.

**# URINE CULTURE & SENSITIVITY TESTS ORDERED MEETING CLINICAL SYMPTOMS**

# See UTI\_Criteria.pdf attachment section A Part 1 (For residents without indwelling catheter) and section B Part 1 (For residents with indwelling catheter) for criteria details.

# The purpose of this function is to count the number urine culture tests ordered that met clinical symptoms. There are two types of patients to consider, those with and without indwelling catheters. The first section of the function determines whether patients without an indwelling catheter met clinical criteria. The second section of the function determines whether patients with a catheter met clinical criteria.

# The dataset ‘nocath’ subsets all columns containing 'nocathsym\_'. There are 13 columns for each patient, each representing the following symptoms (only the first 9 are included in this assessment):

# nocathsym\_\_\_1 == Acute dysuria

# nocathsym\_\_\_2 == Acute pain, swelling, or tenderness of the testes, epididymis, or prostate

# nocathsym\_\_\_3 == Fever or leukocytosis

# nocathsym\_\_\_4 == Acute costovertebral angle pain or tenderness

# nocathsym\_\_\_5 == Suprapubic pain

# nocathsym\_\_\_6 == Gross hematuria

# nocathsym\_\_\_7 == New or marked increase in incontinence

# nocathsym\_\_\_8 == New or marked increase in urgency

# nocathsym\_\_\_9 == New or marked increase in frequency

# nocathsym\_\_\_10 == Foul urine

# nocathsym\_\_\_11 == Dark urine

# nocathsym\_\_\_12 == Other

# nocathsym\_\_\_13 == None

# The ‘result’ container is then created. A for loop adds 1 when a sufficient constellation of symptoms is met. There are three ways the symptoms can be met. 1) nocathsym\_\_\_1 OR nocathsym\_\_\_2 == 1. 2) nocathsym\_\_\_3 AND at least one of nocathsym\_\_\_4 OR nocathsym\_\_\_5 OR nocathsym\_\_\_6 OR nocathsym\_\_\_7 OR nocathsym\_\_\_8 OR nocathsym\_\_\_9. 3) At least two of nocathsym\_\_\_5 OR nocathsym\_\_\_6 OR nocathsym\_\_\_7 OR nocathsym\_\_\_8 OR nocathsym\_\_\_9. These sufficient combinations are described in detail in the UTI\_Criteria.pdf attachment.

###

#For patients with a catheter the process is repeated, with a different constellation supporting clinical criteria. First, the ‘cath’ dataset subsets out all symptom variables for cultured patients with an indwelling catheter. The ‘result2’ container is established matching the number of rows of ‘cath’, and creating a single column for every 8 possible symptoms. The for loop assess whether at least one of the reported symptoms was reported (shown as 1 in ‘cath’), and prints 1 to the ‘result2’ column. There are 8 columns for each patient, each representing the following symptoms (only the first 4 are included in the assessment):

# cathsym\_\_\_1 == Fever, rigors, or new-onset hypotension, with no alternate site of infection

# cathsym\_\_\_2 == Either acute change in mental status or acute functional decline, with no alternate diagnosis and leukocytosis

# cathsym\_\_\_3 == New-onset suprapubic pain or costovertebral angle pain or tenderness

# cathsym\_\_\_4 == Purulent discharge from around the catheter or acute pain, swelling, or tenderness of the testes, epididymis, or prostate

# cathsym\_\_\_5 == Foul urine

# cathsym\_\_\_6 == Dark urine

# cathsym\_\_\_7 == Other

# cathsym\_\_\_8 == None

# URINE CULTURE & SENSITIVITY TESTS ORDERED MEETING MICROBIOLOGIC CRITERIA

# The purpose of this function is to count the number of patients that met the McGeer criteria based on microbiologic subcriteria. Subcriteria are listed in the UTI\_Criteria.pdf attachment section A Part 2A (For residents without indwelling catheters) and section B Part 2 (For residents with an indwelling catheter). These criteria differ for patients without a catheter and patients with a catheter. As in the functions above, “org\\dcnt” variables represent various lab culture results. Each four columns in ‘mic’ (below) represents data from the same patient and contains a numeric from 0-3 representing various urine culture bacterial counts. The CFU response to the numeric values in ‘mic’ are listed below:

# 0 == no lab coulter

# 1 == <50,000 CFUs

# 2 == 50,000 – 100,000 CFUs

# 3== >100,000 CFUs.

# Because microbiologic data for both catheter and non-catheter patients, and subcriteria are different for the two patient types, a separate variable is used to determine catheter status. In the code below the new dataframe ‘catheter’ is created to assess catheter status. The dataframe ‘mic’ is then created to subset out urine culture results. The ‘result’ container is created. The for loop then assess if at least 100,000 CFUs of no more than 2 species of microorganisms in a voided urine sample. In addition, the ‘catheter’ dataframe must equal ‘0’, indicating that the patient did not have a catheter. Within the loop, only a minimum combination of the numeric ‘1’ and ‘2’ is sufficient for a patient to meet the condition. This is because three ‘1’ values may fall well below 50,000 CFUs each, thereby not reaching the 100,000 CFU minimum threshold.

##

# This process is repeated for patients with a catheter, however, the sufficient criteria to meet the microbiologic subcriteria is reduced. The urine culture must report a culture with at least 100,000 CFUs from any organism(s). As before, the ‘result2’ container is created. The for loop then assess if the sum of all four patient columns is greater than two, AND if the patient had a catheter (1).

NUMBER OF PATIENTS MEETING THE MCGEER CRITERIA

# This is function is a combination of the ‘urine\_clin’ and ‘micro’ functions above and is a programmed to apply the UTI\_Criteria.pdf to the data collected in this study. These criteria are referred to as the mcgeer criteria. As in the functions before, non-catheter and catheter patients are assessed separately in distinct for loops.

# First, clinical symptoms are subset into the ‘nocath’ dataframe. Microbiology symptoms are then subset into the ‘mic’ dataframe. The ‘catheter’ dataframe is additionally included, as assessing microbiologic results includes catheter status (see ‘micro’ function notes above). The container ‘result’ is then established to store loop data. The for loop evaluates if both clinical and microbiologic criteria are met. Although the two dataframes have a different number of columns, they are looped over the same number of times. If both conditions are met ‘1’ is printed to the ‘result’ container.

#The process is repeated for catheter patients. A new dataframe is established, ‘cath’ as these symptoms capture different variables than ‘nocath’. The ‘result2’ container is then created to store loop results. Finally, a loop including both clinical and microbiologic criteria is run on the ‘cath’ and ‘mic’ datasets. If both conditions are met, ‘1’ is printed to the ‘result2’ dataset. Both resulting dataframes are combined and summed by row.

# NUMBER OF ANTIBIOTICS PRESCRIBED WITHOUT MEETING THE MCGEER CRITERIA

# This is function counts the total number of antibiotics prescribed that did not meet the McGeer criteria. Similar to the function above, both microbiologic and clinical criteria for both catheter and non-catheter patients are assessed. An additional step is added in part 3 of the function, wherein it is determined whether an antibiotic was prescribed.

# In this final loop, if both the ‘result3’ condition is met (signifying the McGeer criteria was not met), and the ‘result\_presc’ data frame reports a value of 1, 2, 3 or 4 (representing the number of antibiotics prescribed), the same number is printed to the final ‘result4’ dataframe and summed by row.

# COST OF ANTIBIOITCS

# This is a function that calculates the wholesale cost of a antibiotics for the month based on a predefined multiplier. The first date frame ‘name\_list’ subsets out columns with antibiotic names. The regular expression (?!.\*unlist) removes columns including the string ‘unlist’, as these columns may have misspellings or different antibiotic names reported altogether.

# The second data frame ‘name\_unls’ (commented out) includes antibiotic names with the ‘unlis’ string. The difference between ‘name\_list’ and ‘name\_unls’ is that the former only accepts selected names, while ‘name\_unls’ allows any antibiotic name to be manually written to the field. This option is available in the event that an antibiotic was not originally available to be selected in ‘name\_list’. Additional options will be added ‘name\_list’ later (manually).

# The dose of each antibiotic is set into a separate data frame, and the final result container is created. A for loop applied the dosage in ‘dose’ to the predefined multiplier. The rowSums function then sums each row.

#NOTE: Not all multipliers have been created, so the function will not apply to every antibiotic listed in the dummy data set.

#RETURN A DATAFRAME APPLYING FUNCTIONS

#A final function is used to combine all the functions above into a final dataframe. The dataframe is subsequently used to support some of the in-line text and graphics in the associated markdown document.