BC X PMC Guidebook: Anomaly Detection Algorithms

May 28th, 2020

Claire Li, Akhil Rastogi, Conner Yang, Nehariika Kakkar, Yash Tandon

TABLE OF CONTENTS:

Data Modification

New_CSV.R: CSV for Algorithm Use

Modifications:

1. Rename existing columns
    a. **adunit** = column denoting ad unit names
    b. **date** = column denoting date of entry
    c. **dayimp** = column denoting daily impressions sum

```
df <- df %>%
  rename(
    adunit = pubops_revenue_and_impressions_by_ordertype.adunit,
    date = pubops_revenue_and_impressions_by_ordertype.date_date,
    dayimp = pubops_revenue_and_impressions_by_ordertype.impressions_sum
  )
```

2. Replace empty ad unit names with “Unknown”

3. Create new columns from date
    a. **date** = date of entry formatted Y/M/D
    b. **weekday** = day of week (Sun, Mon, Tue, etc.)
    c. **month** = month (Jan, Feb, etc.)

```
df$date <- as.Date(df$date)
df <- df %>% mutate(date = ymd(date),
                    weekday = wday(date, label = TRUE),
                    month = month(date, label = TRUE))
```

4. Create new columns from impressions by ad unit
    a. **impmean** = average impressions by ad unit
    b. **impmed** = median impressions by ad unit

```
findmean <- tapply(df$dayimp, df$adunit, mean, na.rm = T)
df$impmean <- as.numeric(findmean[adunit = df$adunit])

findmed <- tapply(df$dayimp, df$adunit, median, na.rm = T)
df$impmed <- as.numeric(findmed[adunit = df$adunit])
```

Drop to Zero

Approach 1: 0_Drop_1.py

Goal:
Determine ad units and corresponding dates where number of impressions was zero, accounting for ad units commonly fluctuating around zero impressions.

Input: data frame
Output:

- **zero_imp:** list of ad units and corresponding date of drop to zero

1. New column: impstd
    a. Calculates standard deviation of daily impressions sum by ad unit

```
df['impstd'] = df.groupby('adunit')['dayimp'].transform('std')
df['impstd'].fillna(0, inplace=True)
```

2. Set threshold
    a. **threshold** = cutoff for minimum number of daily impressions; used to filter out ad units commonly around zero impressions
    b. Default threshold = 100

3. Function: detect_zero
    a. For ad units with low average daily impressions:
        i. Use threshold to find uncommon drops to zero by determining if zero lies within 1.96 standard deviations of the mean
    b. For other ad units:
        i. Locates drops to zero

```
def detect_zero(df):
        zero = []
        for i in df.index:
                if (df['impmean'][i] <= threshold) and (df['impmean'][i] - 1.96*df['impstd'][i] > 0)
                or (df['impmean'][i] >= threshold):
                        if (df['dayimp'][i] == 0):
                                zero.append([df['adunit'][i], df['date'][i]])

        return zero

zero_imp = detect_zero(df)
#print(zero_imp)
```

Approach 2: 0_Drop_2.R

Goal:
Determine ad units that have dropped to zero or close to zero for a given day, adjusting for ad units whose impressions sum is commonly zero.

Input: data frame, day, month, year
Output:

- **output:** list of ad units and corresponding number of impressions

1. Set threshold
    a. **around_zero_number** = number of impressions considered to be "around zero"
    b. Default around_zero_number = 100

2. Function: around_zero
    a. Determine if zero impressions is "common" – i.e. lies within 1.96 standard deviations of the mean impressions by ad unit

```
around_zero <- function(nums) {
  if (length(nums) > 1) {
    nums_mean <- mean(nums, na.rm = T)
    nums_sd <- sd(nums, na.rm = T)
    two_sd_below <- nums_mean - (1.96*nums_sd)
    two_sd_above <- nums_mean + (1.96*nums_sd)
    two_sd_above >= 0 & two_sd_below <= 0
  } else {
    nums <= around_zero_number
  }
}
```

3. Separate ad units that qualify as "around zero"
    a. **cos_near_zero** = list of ad units that are "around zero"
    b. **cos_not_near_zero** = list of ad units that are not "around zero"

```
cos_near_zero <- names(tapply(df[["dayimp"]], df[["adunit"]], around_zero)
                       [which(tapply(df[["dayimp"]], df[["adunit"]], around_zero) == TRUE)])
cos_not_near_zero <- names(tapply(df[["dayimp"]], df[["adunit"]], around_zero)
                           [which(tapply(df[["dayimp"]], df[["adunit"]], around_zero) == FALSE)])
```

4. Function: adunits_at_zero
    a. Determines ad units at or near zero impressions on a given date, filtering out ad units commonly around zero

    Note: day, month, and year must be inputted in quotes

Ensure correct input format:

```
adunits_at_zero <- function(df, day, month, year) {
  if (mode(day) != "character" || mode(month) != "character" || mode(year) != "character") {
    stop("day, month, & year need to be character values. i.e. in quotes")
  }
  if (nchar(day) != 2 || nchar(month) != 2) {
    stop("day & month need to be 2 characters")
  }
  full_date <- paste(year, month, day, sep = "-")
```

Extract impressions and ad units for inputted date:

```
row_idx <- which(df[["date"]] == full_date)
imp_for_date <- df[["dayimp"]][row_idx]
adunit_for_date <- df[["adunit"]][row_idx]
```

Determine ad units with impressions near zero for inputted date:

```
near_zero_idx <- which(imp_for_date <= around_zero_number)
adunits_near_zero <- adunit_for_date[near_zero_idx]
```

Filter our ad units "commonly" around zero impressions:

```
filter_out_idx <- adunits_near_zero %in% not_near_zero

output <- adunit_for_date[filter_out_idx]
dayimp_col <- imp_for_date[near_zero_idx][filter_out_idx]
```

Biweek_Anomaly.R: 25% Change over Biweekly Period

Goal:
Determine ad units that experienced an unusual 25% change (spike or drop) in biweekly impressions, using quarters of the year as the time period for relevant calculations.

Input: data frame
Outputs:

1. **unusual_biweek:** data frame of all independent spikes and drops flagged
2. **spikes_and_falls:** data frame of spike and drop consecutive events

Important New Variables/Data Frames Created:

1. Variable: biweek_in_year
    a. Assigns number 1-26 to each entry in original data frame based on date to identify corresponding bi-week

```
biweek <- numeric(length(df$week_in_year))

for(i in 1:nrow(df)){
  if(df$week_in_year[i] %% 2 == 0){
    biweek[i] <- df$week_in_year[i] %/% 2
  } else{
    biweek[i] <- (df$week_in_year[i] %/% 2) + 1
  }
}
df$biweek_in_year <- as.numeric(biweek)
```

2. Data Frame: biweek_sum_df
    a. Summary data frame from original data containing adunit, biweek_in_year, year, month, and additional columns below (#3 – 6)
    b. Variable: biweek_impressions:
        i. Sum of impressions over bi-week by ad unit

    Note: month of first bi-week appearance is taken for simplicity

```
biweek_sum_df <- summarize(group_by(df, adunit, biweek_in_year, year),
                           biweek_impressions = sum(dayimp),
                           month = month[1])
```

3. Variable: biweek_diff
    a. Differences in biweekly impression sums by ad unit

For ad units with data present during that year:

```
for(web in unique(biweek_sum_df$adunit)){
  for(year in unique(biweek_sum_df$year)){
    web_year_index <- which((biweek_sum_df$adunit == web) & (biweek_sum_df$year == year))
    if (length(web_year_index) > 0) {
```

Accounting for the following special cases:

1. First year of ad unit appearance (no difference values will exist)
2. All impressions in previous year were the same

```
  if (all(biweek_sum_df$biweek_diff[which(biweek_sum_df$adunit == web)] == 0)) {
    biweek_sum_df$biweek_diff[web_year_index] <-
      c(0,diff(biweek_sum_df$biweek_impressions[web_year_index]))
```

3. First entry in the new year
    a. Use previous year's data to calculate first biweekly difference of new year

```
  } else {
    years_w_data <- sort(unique(biweek_sum_df[which(biweek_sum_df$adunit == web),][["year"]]))
    prev_yr <- years_w_data[which(years_w_data == year) - 1]
    prev_biweek_imp <- tail(biweek_sum_df$biweek_impressions[which((biweek_sum_df$adunit == web) &
                                                                   (biweek_sum_df$year == prev_yr))], 1)
    biweek_sum_df$biweek_diff[web_year_index] <-
      diff(c(prev_biweek_imp, biweek_sum_df$biweek_impressions[web_year_index]))
  }
```

4. Variable: biweek_diff_perc
    a. Percent change between biweekly impression sums of consecutive weeks by ad unit

Account for special cases of no data in previous year (see above):

```
if (length(prev_yr_indices) == 0) {

  for(i in seq(length(indices))){
    if(i != 1){
      if(biweek_sum_df$biweek_impressions[indices[i-1]] != 0){
        biweek_diff_perc[indices[i]] <-
          biweek_sum_df$biweek_diff[indices[i]]/biweek_sum_df$biweek_impressions[indices[i-1]]
      } else{
        biweek_diff_perc[indices[i]] <- biweek_sum_df$biweek_diff[indices[i]]
      }
    } else{
      biweek_diff_perc[indices[i]] <- 0
```

If this is not the first year of data, account for previous bi-week impressions:

```
if ((length(prev_yr_indices) != 0) & (length(indices) != 0)) {
  # If this is not the first year of data
  years_w_data <- sort(unique(biweek_sum_df[which(biweek_sum_df$adunit == web),][["year"]]))
  prev_yr <- years_w_data[which(years_w_data == year) - 1]
  prev_biweek_rev <- tail(biweek_sum_df$biweek_impressions[which((biweek_sum_df$adunit == web) &
                                                                 (biweek_sum_df$year == prev_yr))], 1)
  # Account for previous biweek impressions
  for(i in seq(length(indices))){
    if(i != 1){
      if(biweek_sum_df$biweek_impressions[indices[i-1]] != 0){
        biweek_diff_perc[indices[i]] <-
          biweek_sum_df$biweek_diff[indices[i]]/biweek_sum_df$biweek_impressions[indices[i-1]]
      } else{
        biweek_diff_perc[indices[i]] <- biweek_sum_df$biweek_diff[indices[i]]
      }
    } else{
      biweek_diff_perc[indices[i]] <-
        biweek_sum_df$biweek_diff[indices[i]]/prev_biweek_rev
```

5. Variable: z_diff
    a. Z-score of biweekly impression differences calculated using mean and standard deviation of biweekly impression differences grouped by ad unit and quarter of the year

```
for(web in unique(biweek_sum_df$adunit)){
  for(year in unique(biweek_sum_df$year)){
    for(quarter in 1:4){
      indices <- which((biweek_sum_df$adunit == web) & (biweek_sum_df$year == year) & (biweek_sum_df$quarter == quarter))
      if(length(indices) == 0){
        break
      }
      std <- sd(biweek_sum_df$biweek_diff[which((biweek_sum_df$adunit == web) & (biweek_sum_df$year == year)
                                                & ((biweek_sum_df$quarter == quarter)))], na.rm = T)
      avg <- mean(biweek_sum_df$biweek_diff[which((biweek_sum_df$adunit == web) & (biweek_sum_df$year == year)
                                                  & ((biweek_sum_df$quarter == quarter)))], na.rm = T)
      for(i in seq(length(indices))){
        biweek_z[indices[i]] <- (biweek_sum_df$biweek_diff[indices[i]] - avg)/std
```

6. Variable: quarter
    a. Assigns number 1-4 to each entry in biweek_sum_df based on date to identify corresponding quarter

```
q1_idx <- which((biweek_sum_df$month == 1) | (biweek_sum_df$month == 2) | (biweek_sum_df$month == 3))
q2_idx <- which((biweek_sum_df$month == 4) | (biweek_sum_df$month == 5) | (biweek_sum_df$month == 6))
q3_idx <- which((biweek_sum_df$month == 7) | (biweek_sum_df$month == 8) | (biweek_sum_df$month == 9))
q4_idx <- which((biweek_sum_df$month == 10) | (biweek_sum_df$month == 11) | (biweek_sum_df$month == 12))
quarter <- numeric(nrow(biweek_sum_df))
quarter[q1_idx] <- 1
quarter[q2_idx] <- 2
quarter[q3_idx] <- 3
quarter[q4_idx] <- 4
biweek_sum_df$quarter <- quarter
```

Output Data Frame Construction:

1. Set thresholds
    a. **set_perc** = percentage difference in biweekly impressions
        i. Default set_perc = 0.25
    b. **set_z** = z-score cutoff of biweekly impression differences
        i. Default set_z = 1.5

2. Data Frame: unusual_biweek
    a. Data frame of all independent spikes and drops flagged

```
unusual_biweek <- biweek_sum_df[which((abs(biweek_sum_df$biweek_diff_perc) >= set_perc) &
                                      (abs(biweek_sum_df$z_diff) > set_z)),]
```

3. Data Frame: spikes_and_falls
    a. Data frame of spike and drop consecutive events

    Note: Consecutive weeks are determined using difference in bi-weeks from unusual_biweek data frame (difference = 1)

```
for(web in unique(unusual_biweek$adunit)){
  for(year in unique(unusual_biweek$year)){
    sf_indices <- c(sf_indices,
                    which((unusual_biweek$adunit == web) & (unusual_biweek$year == year))
                    [which(c(diff(unusual_biweek$biweek_in_year
                                  [which((unusual_biweek$adunit == web)
                                         & (unusual_biweek$year == year))])) == 1)])
```

Single_Day_Anomaly.py: 50% Change on One Day

Goal:
Determine ad units that experienced an unusual 50% change (spike or drop) on one day compared to previous "x" number of appearances on the same weekday, using three different time periods (7 days, 14 days, 30 days) for relevant calculations.

Input: data frame
Outputs:
Five data frames / CSV files per ad unit organized by the following flags:

1. df_7day_flag / '.7day_flag.csv'
2. df_14day_flag / '.14day_flag.csv'
3. df_30day_flag / '.30day_flag.csv'
4. df_50_flag / '.50percent_flag.csv'
5. df_sig_anom / '.significant_anomalies.csv'
    a. Flagged in both 50% change and 30-day flag

Preparation:

1. Set paths
    a. DATASET_PATH = path to data file
    b. PATH_TO_ADUNIT_FOLDER = path to folder to store CSV file for each unique ad unit for easy access
    c. PATH_TO_OUTPUT_FOLDER = path to store output data frames

2. Read in data and drop unnecessary columns
    a. Data must be in order from most recent (at the top) to oldest (at the bottom) by date

3. Create separate CSV files for easier access to each individual ad unit
    a. Set SAVE_PATH to same path as PATH_TO_ADUNIT_FOLDER

```
websiteNames = df.adunit.unique()

for i in websiteNames:
  current_web_df = df[df.adunit == i]
  # Save_PATH = PATH_TO_ADUNIT_FOLDER
  SAVE_PATH = ''
  SAVE_PATH += str(i)
  SAVE_PATH += '.csv'
  current_web_df.to_csv(SAVE_PATH)
```

4. Create two constants
    a. NUM_SAME_WEEKDAY
        i. Previous number of appearances of an ad unit on the same weekday used to find the average used in the calculation of percent change in impressions (from average)
        ii. Default NUM_SAME_WEEKDAY = 10
    b. Z_THRESHOLD
        i. Z-score (of daily impressions difference) threshold
        ii. Default Z_THRESHOLD = 1.96 for 95% confidence interval

5. Variable: websiteNames
    a. List of unique ad units in data frame
    b. In order to run the algorithm for specific ad unit(s), modify websiteNames to contain ad unit(s) of interest

    Note: Do not change the previous definition of websiteNames that is also websiteNames = df.adunit.unique( ) unless the goal is to save only certain ad units in their own CSV. *Change the websiteNames that is after the definition of the constants.*

6. Dictionary: flagToCount
    a. Dictionary to keep a running count of how many total flags exist for each type of flag

```
flagToCount = {
      "50%_spike": 0,
      "50%_drop": 0,
      "7day_spike": 0,
      "7day_drop": 0,
      "14day_spike": 0,
      "14day_drop": 0,
      "30day_spike": 0,
      "30day_drop": 0,
      "sig_spike": 0,
      "sig_drop": 0
}
```

Algorithm Components:

1. Variable: numAds
    a. Number of unique ad units

2. Data Frame: dfWeb
    a. Create data frame for current ad unit in "for" loop

```
for ii in range(numAds):
  #Create dataframe for current adunit
  curWeb = websiteNames[ii]
  SAVE_PATH = PATH_TO_ADUNIT_FOLDER
  SAVE_PATH += str(curWeb)
  SAVE_PATH += '.csv'
  dfWeb = pd.read_csv(SAVE_PATH, index_col=0)
```

Flag 50% Change

Purpose: Flag for 50% change in impressions for each ad unit from the average of the last NUM_SAME_WEEKDAY (constant set by user) appearances of the ad unit on the same day of the week

Note: Value of 1 means the 50% increase threshold has been met. Value of -1 means the 50% decrease threshold has been met. Otherwise, a value of 0 is set.

1. Section I: Iterate backwards until NUM_SAME_WEEKDAY occurrences of the ad unit are found. Find the average revenue of these
    a. Variables Created
        i. Variable: numLines
            1. Number of lines in the data frame for the current ad unit
        ii. Variable: impSum
            1. Tracks the total sum of impressions of the last NUM_SAME_WEEKDAY
        iii. Variable: numDaysToCheckb
            1. Tracks how many days there are left to check
        iv. Variable: percentageChange
            1. Percentage change of impressions from the average of the last NUM_SAME_WEEKDAY appearances of the ad unit on the same of day of the week

b. Create break (stop) if there are no appearances of the ad unit on the same weekday left to check
c. Check to make sure there are no out-of-bounds errors

```
numLines = len(dfWeb)
for i in range(numLines-1):
  impSum = 0.0
  numDaysToCheck = NUM_SAME_WEEKDAY
  percentageChange = 0.0
  for a in range(i, numLines):
    if numDaysToCheck < 1:
      break
    if (i+a) > numLines-1:
      break
```

d. If appearance of ad unit on the same weekday is found, add to impSum and decrement numDaysToCheck

```
if dfWeb['weekday'].iloc[i] == dfWeb['weekday'].iloc[i+a]:
  impSum += dfWeb['dayimp'].iloc[i+a]
  numDaysToCheck = numDaysToCheck - 1
```

2. Section II: Calculate the percentage change
   a. If impressions for all NUM_SAME_WEEKDAY occurrences found are zero (impSum = 0), set percentageChange to zero
   b. If the impressions for all NUM_SAME_WEEKDAY occurrences found are zero (impSum = 0) and the current day's revenue is non-zero, then set the percentageChange to 99 (if current day's revenue is positive) or -99 (if current day's revenue is negative)
      i. (-)99 is used as an arbitrarily high/low percentage change. Technically it should be infinite, but (-)99 is sufficient to exceed the threshold

```
if numDaysToCheck != NUM_SAME_WEEKDAY:
  if impSum != 0:
    # Calculate percentageChange normally
    meanImp = impSum / (NUM_SAME_WEEKDAY - numDaysToCheck)
    percentageChange = (dfWeb['dayimp'].iloc[i] - meanImp) / meanImp
  else:
    if dfWeb['dayimp'].iloc[i] == 0:
      percentageChange = .00
    elif dfWeb['dayimp'].iloc[i] > 0:
      percentageChange = .99
    elif dfWeb['dayimp'].iloc[i] < 0:
      percentageChange = -.99
```

c. Otherwise, if no appearances on the same weekday are found, set percentageChange to default of zero

```
else:
  percentageChange = .00
```

3. Section III: Flag 1 for 50% increase, -1 for 50% decrease

```
if (percentageChange >= .50):
   dfWeb['50%_flag'].iloc[i] = 1
   flagToCount["50%_spike"] = flagToCount["50%_spike"] + 1
if (percentageChange <= -.50):
   dfWeb['50%_flag'].iloc[i] = -1
   flagToCount["50%_drop"] = flagToCount["50%_drop"] + 1
```

Flag Z-Scores

Purpose: Use z-score to flag to unusual changes in impressions for each ad unit using three time periods (7 days, 14 days, 30 days).

1. Create new columns
   a. New Column: delta_imp
      i. Iteratively calculate change in impressions for each appearance of an ad unit from the previous date (excluding the oldest date)

```
dfWeb['delta_imp'] = 0
for i in range(numLines - 1):
  impressions_difference = dfWeb['dayimp'].iloc[i] - dfWeb['dayimp'].iloc[i+1]
  dfWeb['delta_imp'].iloc[i] = impressions_difference
```

   b. New Columns: 7day_zScore, 14day_zScore, 30day_zScore
      i. Z-Score of impressions change for the last 7, 14, or 30 appearances of the ad unit
      ii. Create mini data frames for last 7, 14, 30 days

```
for i in range(numLines-31):
  df_7days = dfWeb[i:i+8]
  df_14days = dfWeb[i:i+15]
  df_30days = dfWeb[i:i+31]
```

iii. Calculate the standard deviations of change in impressions for the three specified time frames

Note: In the case of a standard deviation of 0 change in impressions, simply set it to a low value (0.01), in order to avoid dividing by 0. This will cause the z-score to be large, which is the desired effect.

```
std_dev_7days = df_7days['delta_imp'].std(ddof=0)
if std_dev_7days == 0:
  std_dev_7days = 0.01

std_dev_14days = df_14days['delta_imp'].std(ddof=0)
if std_dev_14days == 0:
  std_dev_14days = 0.01

std_dev_30days = df_30days['delta_imp'].std(ddof=0)
if std_dev_30days == 0:
  std_dev_30days = 0.01
```

iv. Calculate the z-score of change in impressions for three specific time frames

```
z_score = (dfWeb['delta_imp'].iloc[i] - df_7days['delta_imp'].mean()) / std_dev_7days
dfWeb['7day_zScore'].iloc[i] = z_score

z_score = (dfWeb['delta_imp'].iloc[i] - df_14days['delta_imp'].mean()) / std_dev_14days
dfWeb['14day_zScore'].iloc[i] = z_score

z_score = (dfWeb['delta_imp'].iloc[i] - df_30days['delta_imp'].mean()) / std_dev_30days
dfWeb['30day_zScore'].iloc[i] = z_score
```

2. If a z-score is above Z_THRESHOLD, flag the value
    a. Flag value '1' indicates that the positive Z_THRESHOLD has been met (spike) whereas flag value '-1' indicates that the negative Z_THRESHOLD has been met (drop). Otherwise a value of 0 is set.

Note: "for" loop for flagging z-scores iterates to "numLines – n" because the first "n" dates will have no valid z-score (not enough history), where "n" is the length of the longest specified time frame (because no z-scores are calculated for the previous "n" lines)

b. New Column: 7day_flag

```
for i in range(numLines - 31):
  if float(dfWeb['7day_zScore'].iloc[i]) >= Z_THRESHOLD:
    dfWeb['7day_flag'].iloc[i] = 1
    flagToCount["7day_spike"] = flagToCount["7day_spike"] + 1
  if float(dfWeb['7day_zScore'].iloc[i]) <= -1 * Z_THRESHOLD:
    dfWeb['7day_flag'].iloc[i] = -1
    flagToCount["7day_drop"] = flagToCount["7day_drop"] + 1
```

c. New Column: 14day_flag

```
for i in range(numLines - 31):
  if float(dfWeb['14day_zScore'].iloc[i]) >= Z_THRESHOLD:
    dfWeb['14day_flag'].iloc[i] = 1
    flagToCount["14day_spike"] = flagToCount["14day_spike"] + 1
  if float(dfWeb['14day_zScore'].iloc[i]) <= -1 * Z_THRESHOLD:
    dfWeb['14day_flag'].iloc[i] = -1
    flagToCount["14day_drop"] = flagToCount["14day_drop"] + 1
```

d. New Column: 30day_flag

```
for i in range(numLines - 31):
  if float(dfWeb['30day_zScore'].iloc[i]) >= Z_THRESHOLD:
    dfWeb['30day_flag'].iloc[i] = 1
    flagToCount["30day_spike"] = flagToCount["30day_spike"] + 1
  if float(dfWeb['30day_zScore'].iloc[i]) <= -1 * Z_THRESHOLD:
    dfWeb['30day_flag'].iloc[i] = -1
    flagToCount["30day_drop"] = flagToCount["30day_drop"] + 1
```

3. Flag “significant anomalies”
   a. Method: entry is flagged as “significant anomaly” if an impressions change is flagged ‘1’ or ‘-1’ in both the 50% change and 30-day z-score

      Note: 30-day z-score was chosen to reduce volatility and better capture seasonal trends instead of trends within a week, as the typical number of impressions of a week is already accounted for by the 50% change from average of the weekday flag.

      Note: Flags for the z-scores are included in the output as well so that z-scores can be observed separately from the “significant anomaly flag”. This is because significant anomalies will be flagged for obvious cases, but certain special cases will not be flagged, such as:

1. If the last 7 days had highly increasing impressions, but the impressions dropped today, the 7-day z-score would be flagged (applicable to any of the time frames). However, if the last NUM_SAME_WEEKDAY appearances of the ad unit on the same weekday had roughly the same number of impressions as today, then the "significant anomaly" would not be flagged.

b. Column: significant_anomaly
   i. Value of '1' means both z-score and 50% change flag have value of '1'. Value of '-1' means both z-score and 50% change flag have value of '-1'. Otherwise, a value of 0 is set.

```
dfWeb['significant_anomaly'] = 0

for i in range(numLines - 31):
  positive_z_flag = dfWeb['30day_flag'].iloc[i] == 1
  negative_z_flag = dfWeb['30day_flag'].iloc[i] == -1
  if positive_z_flag and dfWeb['50%_flag'].iloc[i] == 1:
    dfWeb['significant_anomaly'].iloc[i] = 1
    flagToCount["sig_spike"] = flagToCount["sig_spike"] + 1
  if negative_z_flag and dfWeb['50%_flag'].iloc[i] == -1:
    dfWeb['significant_anomaly'].iloc[i] = -1
    flagToCount["sig_drop"] = flagToCount["sig_drop"] + 1
```

Output

1. Reset the index to be the row number of the current data frame and drop the old index from the data frame imported with pd.read_csv( )

```
dfWeb.reset_index(inplace=True)
dfWeb = dfWeb.drop(['index'], axis=1).copy()
```

2. Generate five output data frames:
    a. **df_7day_flag, df_14day_flag, df_30day_flag, df_50_flag, df_sig_anom**

```
df_7day_flag = dfWeb[(dfWeb['7day_flag'] == 1) | (dfWeb['7day_flag'] == -1)]
df_14day_flag = dfWeb[(dfWeb['14day_flag'] == 1) | (dfWeb['14day_flag'] == -1)]
df_30day_flag = dfWeb[(dfWeb['30day_flag'] == 1) | (dfWeb['30day_flag'] == -1)]
df_50_flag = dfWeb[(dfWeb['50%_flag'] == 1) | (dfWeb['50%_flag'] == -1)]
df_sig_anom = dfWeb[(dfWeb['significant_anomaly'] == 1) | (dfWeb['significant_anomaly'] == -1)]
```

3. Save all output data frames as CSV files (sample for 7day_flag shown below)

```
SAVE_PATH_7_DAYS = PATH_TO_OUTPUT_FOLDER
SAVE_PATH_7_DAYS += str(curWeb)
SAVE_PATH_7_DAYS += '.7day_flag.csv'
df_7day_flag.to_csv(SAVE_PATH_7_DAYS)
```

4. Print out counts of flags

```
for key, value in flagToCount.items():
  print(key, ':', value, 'flags')
```

Additional Resources

Imputations.py: Imputations Code (Not Used)

Motivation:
Some ad units did not have entries for every date in the data, so a possible solution is to impute daily impressions sum as 0 for each missing date per ad unit starting from each ad unit's first appearance in the data (so all imputed values make sense).

General approach:
Associate (using a dictionary) each ad unit to the first date of appearance in the data (taking advantage of the ordering by date of the dataset). Then, create a temporary data frame for each unique ad unit, re-indexed (pandas library) by date ('D') to fill in impressions of 0 for all missing dates (not discriminating between dates that appear after the first date in the original dataset and those that do not). Concatenate the data frames back together. Flag and remove all dates that are before the date of first appearance (stored in the dictionary) and save the new data frame to a CSV file.

1. Set data paths and remove any unnecessary columns
    a. DATASET_PATH = original data file to be imputed
    b. SAVE_PATH = new data file with imputations

2. Create dictionary mapping each ad unit to date of first appearance
    a. Change 'date' column to dateTime object that can be used to compare dates to dates (strings cannot do this as easily)
    b. Dictionary: webToDateDict
        i. Dictionary mapping each ad unit to its date of first occurrence
    c. Reverse the order of the data frame so earliest date that an ad unit (key) appears in the data frame can be added

```
df['date'] = pd.to_datetime(df['date'])
webToDateDict = {}
df = df.iloc[::-1]
```

d. Iterate through each row, associating the ad unit (row[0]) as the key to the date (row[adunit]) as the value if it does not already exist in the dictionary
    i. Since the data frame is in reverse order (starting from earliest date, sorted by date), it should only associate each ad unit with its date of first appearance

```
for index, row in df.iterrows():
  adunit = row['adunit']
  date = row['date'];

  if webToDateDict.get(adunit) == None:
    webToDateDict[adunit] = date;
```

3. Create data frame with imputations for all missing dates (without consideration of date of first appearance)
    a. Split each ad unit into unique data frame, checking that 'Unknown' websites are not re-indexed (do not want to impute missing data for 'Unknown' websites)
    b. Data Frame: tmp_df
        i. Temporary data frame for current unique website

```
for website in df['adunit'].unique():
    if website == 'Unknown':
      continue

    tmp_df = df.loc[df['adunit'] == website].copy()
```

    c. Re-index by 'date' to impute missing values of 0 for all missing dates for this data frame
        i. Assign the correct website name for the 'adunit' column
        ii. Append the temporary data frame for each individual ad unit into a list of data frames
        iii. Concatenate all data frames in the list into one combined data frame of all ad units again

```
    idx = pd.date_range(start=df.date.min(), end=df.date.max())
    tmp_df = tmp_df.set_index('date').reindex(idx, fill_value=0).rename_axis('date').reset_index().copy()
    tmp_df['adunit'] = website

    df_list.append(tmp_df)

df = pd.concat(df_list).copy()
```

4. Flag and remove ad unit entries that occur before date of first appearance in original data frame
    a. New Column: badDate
        i. If column value is '1', this ad unit entry has a bad date and should be removed

```
df.reset_index(inplace=True)
df = df.drop(['index'], axis=1)
df['badDate'] = 0;
```

    b. Use dictionary webToDateDict that maps ad unit to first date of appearance to flag and remove dates that are before the first date

```
for index, row in df.iterrows():
  if row['date'].date() < webToDateDict[row['adunit']].date():
    df['badDate'][index] = 1
  print(index)
```

5. Create and save new data frame out of non-flagged rows

```
df = df[df.badDate == 0]
df = df.drop(columns = ['badDate'])

# Export to CSV
df.to_csv(SAVE_PATH)
```

6. Determine number of rows in original dataset, new dataset, and the difference

```
print('Length of original data is', str(len(original.index)))
print('Length of new data is', str(len(df.index)))
print(str(len(df.index) - len(original.index)), 'rows of data were imputed')
```

Note: The new dataset with imputed values is grouped by ad unit and is no longer organized by date from most recent (at the top) to oldest (at the bottom). It can be re-ordered by date if desired.

Concluding Remarks:

Due to the large number of imputed zeros in the new dataset produced by this script, much of the data analysis performed for this project was no longer useful. Therefore, the new dataset was not used during algorithm development. However, the code has been included here in case it should come of use – for example if any future dataset were to have few missing data entries.